Basic Image Processing

PPKE-ITK

Basic Image Processing Algorithms

- Course responsible and lecturer:
  - Csaba Benedek, PhD
    - Address:
      - *Primary*: MTA SZTAKI (Institute for Computer Science and Control, Hungarian Academy of Sciences), Machine Perception Laboratory, 1111 Budapest, Kende utca 13-17, room 306
      - *Secondary*: PPKE ITK room 408 (only for preliminary fixed meetings)
      - E-mail: benedek.csaba@itk.ppke.hu

- Course webpage:
  - http://kep.itk.ppke.hu/

About myself…

Personal site: http://mplab.sztaki.hu/~bcsaba

- **Research:** computer vision, pattern recognition, 3D sensors (laser scanning), biometrics (gait recognition)
- MTA SZTAKI – leader of Research Group on Geo-Information Computing (GeoComp) @ Machine Perception Laboratory
- PPKE ITK – associate professor (in part time, since 2015)

Introduction

The Research Group on Geo-information Computing (GeoComp) is a subunit of the Machine Perception Research Laboratory (MPLab) in the Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI). GeoComp has been funded in 2017 by senior researchers Csaba Benedek and Zsolt Jankó, based on experience from various successful projects on spatial data processing completed within the MPLab and its predecessors. The group is strongly cooperating with other units the laboratory under the supervision of MPLab head Prof. Tamás Szirányi.

Context. In the past decade we have witnessed an explosion of new technologies for understanding and using location information in our environment. Airborne, terrestrial and mobile mapping systems equipped with laser scanners, calibrated cameras and navigation sensors are able to provide very accurate and high resolution 3D measurement flows. However, due to qualitative and quantitative factors, conventional Geo-Information Systems (GIS) cannot efficiently utilize yet the available very dense 3D data, and the quickly evolving size of the databases means nowadays a real barrier for applications needing up-do-date semantic evaluation.

Links

Geo-Information Computing @ Machine Perception Lab.

GeoComp Research Group

Machine Perception Laboratory

SZTAKI main page

GeoComp Demos:

Demo page

Publications

Contact

GeoComp Group leader:
Dr. Csaba Benedek
benedek.csaba@sztaki.mta.hu

i4D project manager:
Dr. Zsolt Jankó
janko.zsolt@sztaki.mta.hu

SZTAKI GeoComp Research Group

http://mplab.sztaki.hu/geocomp

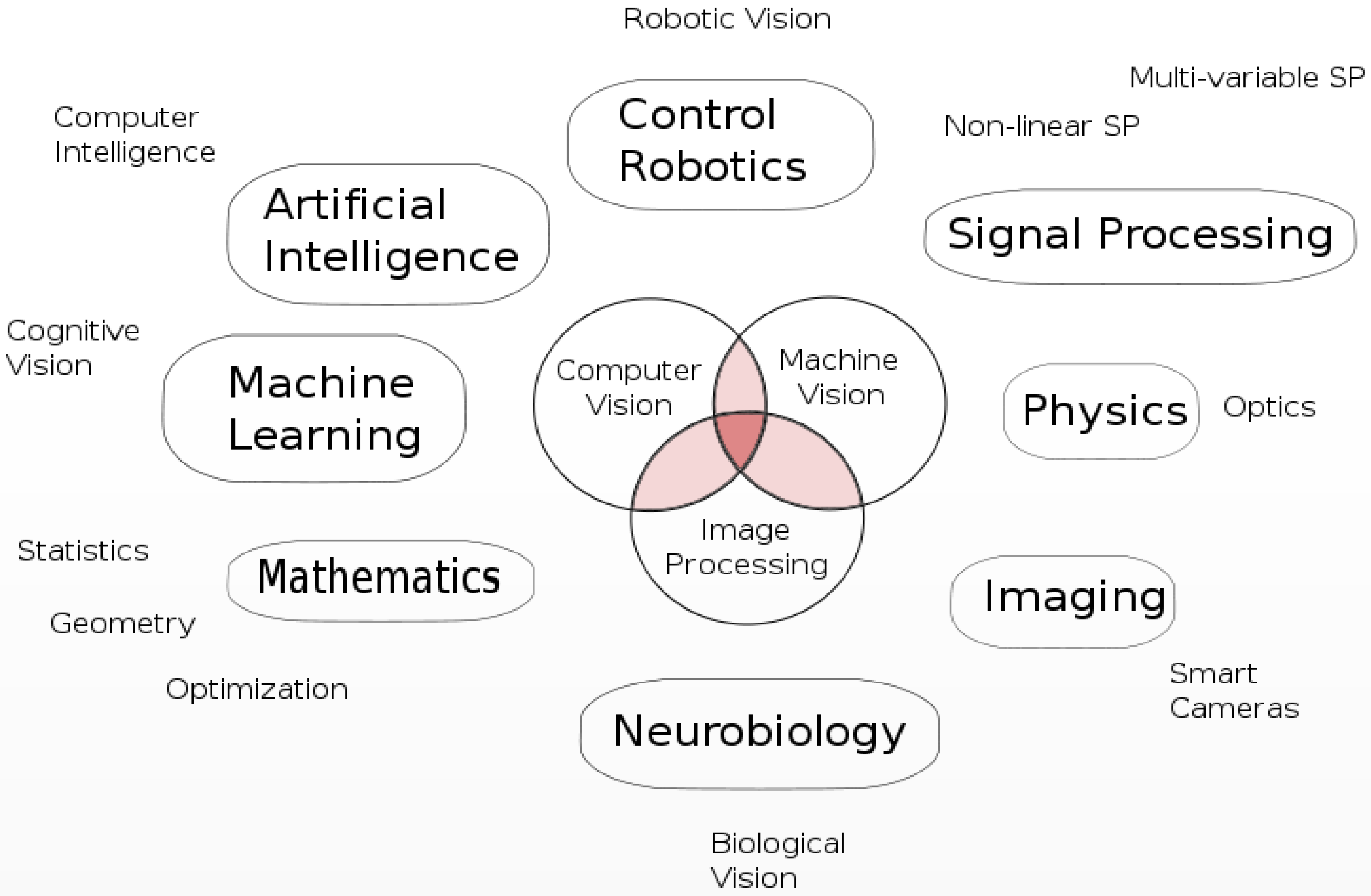
PPKE | HTK | BTK | JÁK | KJPI | ITK | Alumni | Beszerzések és közbeszerzések
Magyar | Webmail | Neptun | Phonebook | | Login
PÁZMÁNY PÉTER CATHOLIC UNIVERSITY – UNIVERSITY OF NATIONAL EXCELLENCE
FACULTY OF HUMANITIES AND SOCIAL SCIENCES
1635
About
Information on Studies
Institutes and Departments
Academic Affairs
International Office
Search
Main page
Information on Studies
Departments
Institute of Archaeology
Archaeological GIS Laboratory
News
Courses in English
Curricula
Academic Staff
Demonstrators
Research Projects
On Scholarship
Archaeological GIS Laboratory
Faculty Brochure
Guide U

Archaeological GIS Laboratory

Following several years of careful planning and active fieldwork, the Council of the Faculty of Humanities and Social Sciences established the Archaeological GIS (Geographic Information System) Laboratory in Esztergom in 2014. The members of the Laboratory are mainly lecturers and researchers of the Department of Archaeology. The purpose of AGL is to support the Department

PPCU Archaeological GIS Laboratory

What is this Course About?



Robotic Vision

Multi-variable SP

Computer Intelligence

Control Robotics

Non-linear SP

Artificial Intelligence

Signal Processing

Cognitive Vision

Machine Learning

Computer Vision

Machine Vision

Physics

Optics

Image Processing

Statistics

Mathematics

Imaging

Geometry

Optimization

Smart Cameras

Neurobiology

Biological Vision

Source: Wikipedia

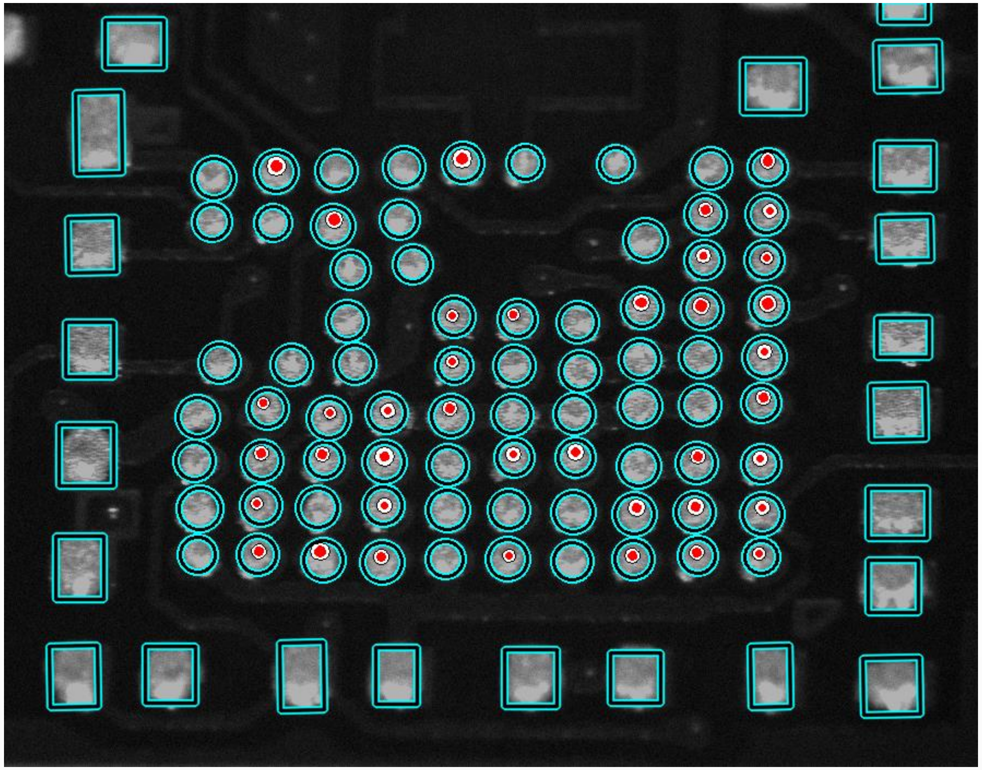
What is this Course About?

- **Machine Vision** is the technology and methods used to provide imaging-based automatic inspection and analysis for such applications as automatic inspection, process control, and robot guidance in industry.”

Sources: Wikipedia, http://automation.com, http://www.isquaredt.com/

Machine vision - example

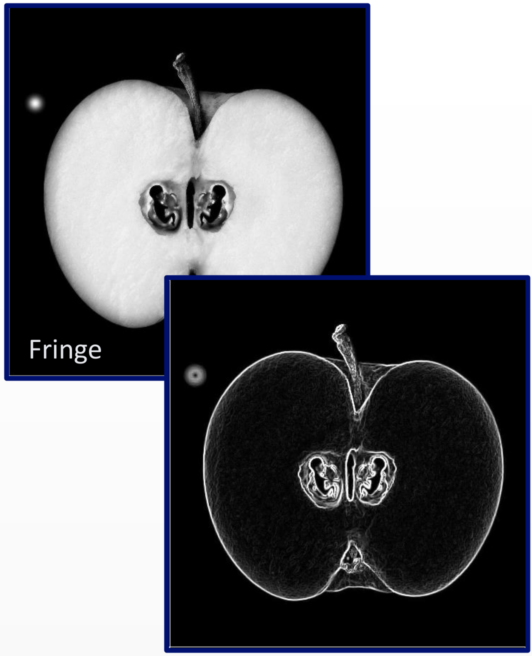
- Optical analysis of scooping artifacts in printed circuit boards



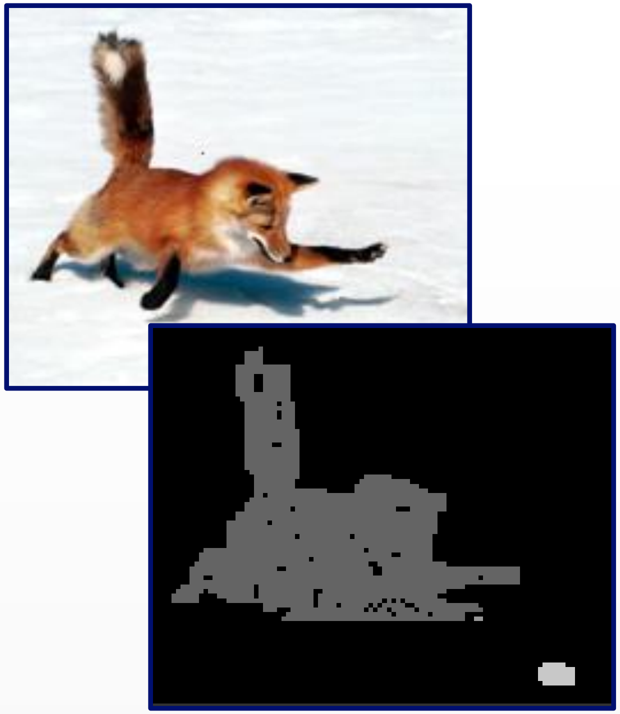
Cs. Benedek, O. Krammer, M. Janóczki and L. Jakab: ”Solder Paste Scooping Detection by Multi-Level Visual Inspection of Printed Circuit Boards”, *IEEE Trans. on Industrial Electronics*, vol. 60, no. 6, pp. 2318 - 2331, 2013

What is this Course About?

- **Image Processing** ”is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image.”



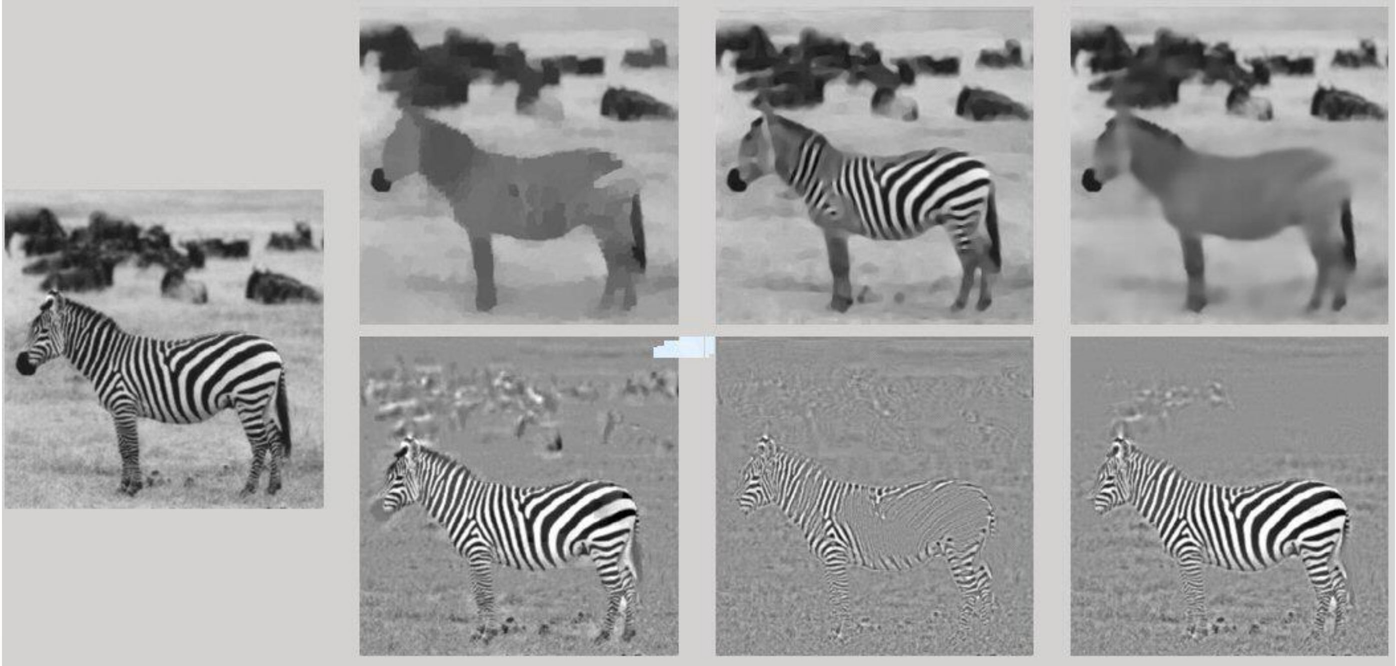
The windmill at Wijk bij Duurstede by Jacob van Ruisdael (1670)



Source: Wikipedia

Image processing example

- Image Decomposition Into Cartoon and Texture Parts



Cartoon (main object shapes)

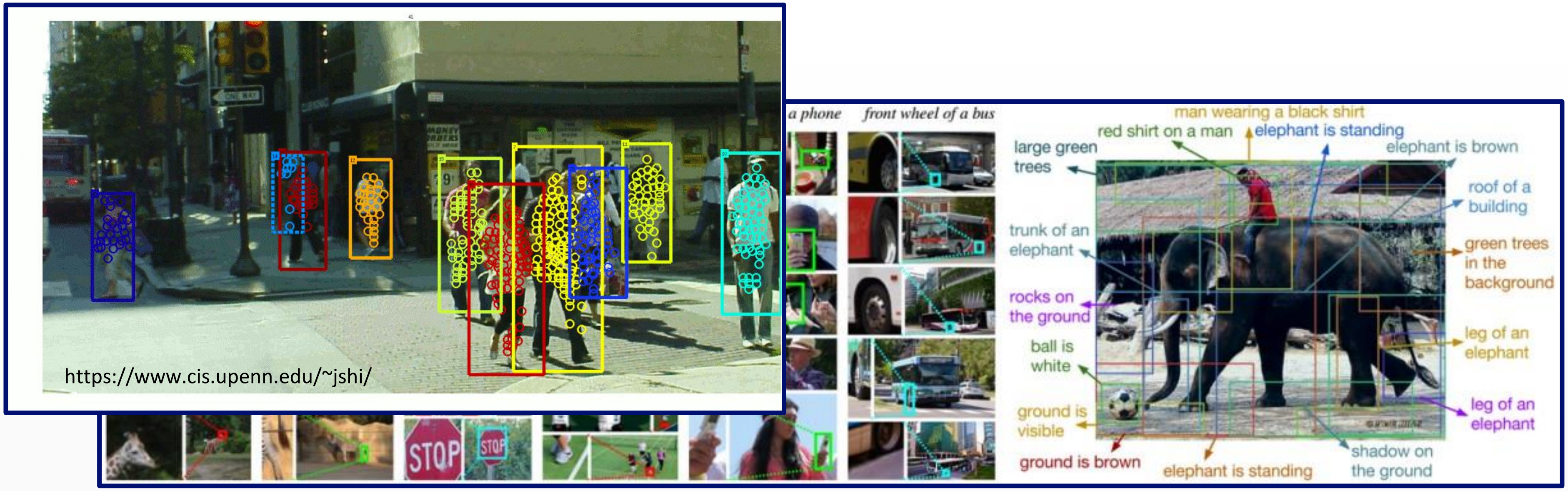
Texture (pattern on the object cover)

Input image Method 1 Method 2 Method 3

Dániel Szolgay, Tamás Szirányi, „Adaptive Image Decomposition Into Cartoon and Texture Parts Optimized by the Orthogonality Criterion” *IEEE Trans. on Image Processing*, 21 (8). pp. 3405-3415, 2012

What is this Course About?

- „**Computer Vision** is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world”
  „A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image.”



https://www.cis.upenn.edu/~jshi/

http://cs.stanford.edu/people/karpathy/

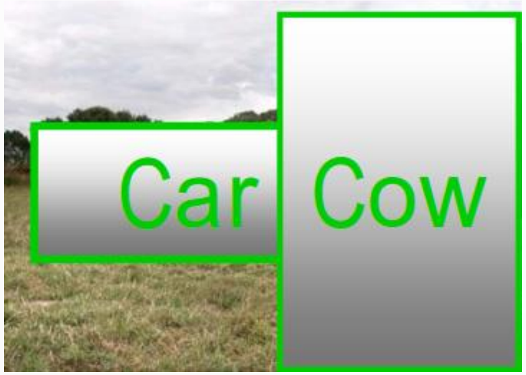
Source: Wikipedia

Various levels of computer vision tasks

- Image classification: assigning a class label to the image

Car: present
Cow: present
Bike: not present
Horse: not present
…

- Object localization: define the location and the category



Location

Category

Video Traffic on the Internet

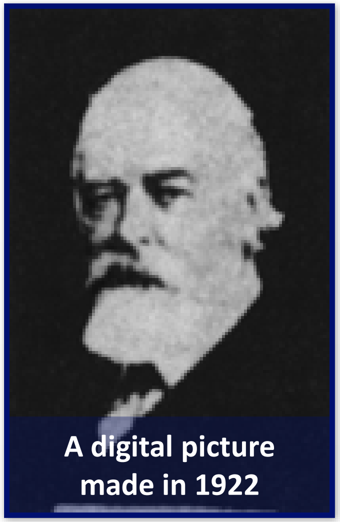
- A few crazy predictions from Cisco for 2020:
  - video traffic will be 82% of all IP traffic (both business and consumer), up from 70% in 2015,
  - it would take more than 5 million years to watch the amount of video that will cross global IP networks each month,
  - every second, a million minutes of video content will cross the network.

http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

Little bit of History

- Early 1920s - Bartlane cable picture transmission system
  - Used to transmit newspaper images via submarine cable between London and New York.
  - Took about three hours to send an image, first systems supported 5 gray levels

A digital picture produced in 1921



A digital picture made in 1922

  - But these images were not created with computer, hence not considered as a result of digital image processing.
  - The real era of digital images started only after computers got powerful enough for the task.

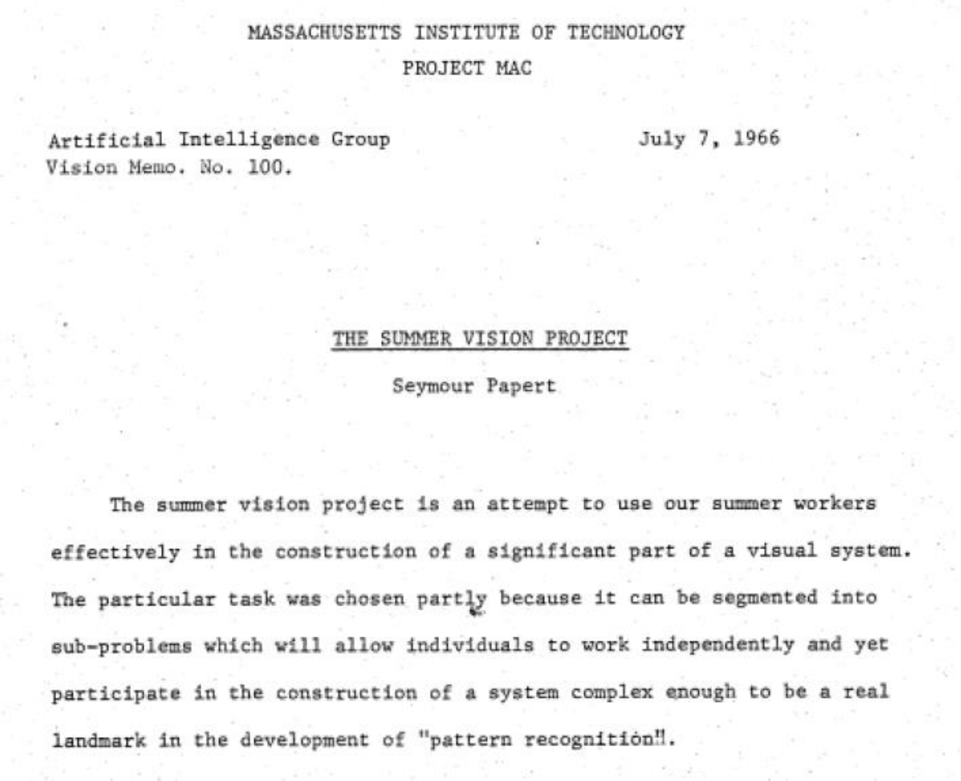
Little bit of History

- In the early 1960s:
  - 1964:NASA’s Jet Propulsion Laboratory began working on computer algorithms to improve images of the Moon.
    - images were transmitted by Ranger 7 probe.
    - corrections were desired for distortions inherent in on-board camera
- In the late 1960s and early 1970s:
  - medical imaging (CT),
  - remote Earth resources observations,
  - and astronomy
- So far this was image processing, what about computer vision?

The first picture of the moon by a U.S. spacecraft. (Courtesy of NASA.)

Little bit of History

- In 1966, Marvin Minsky (MIT) asked his student to "spend the summer linking a camera to a computer and getting the computer to describe what it saw".



MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

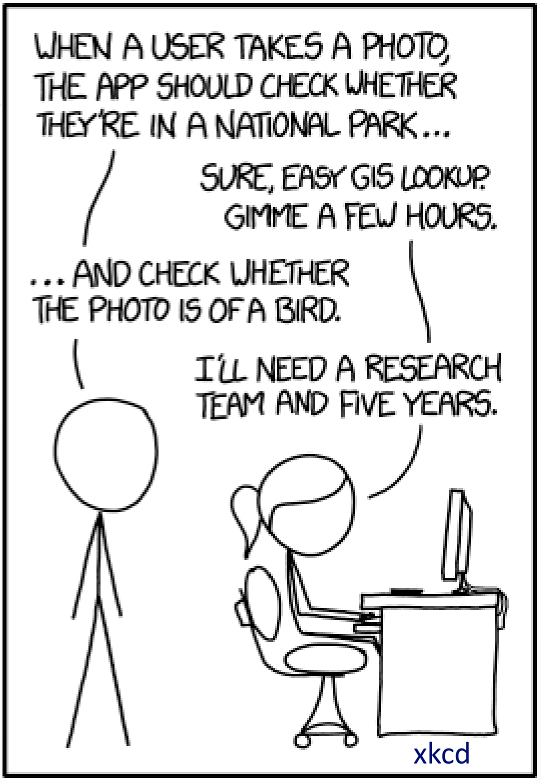
Artificial Intelligence Group
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".


WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
xkcd

IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Little bit of History

Neural Networks:

- 1958: Frank Rosenblatt introduced the Perceptron model
- 1970's: Backpropagation algorithm for larger network training
- 1980's: Appearance of Convolutional NN
- 1998: First success of CNN:

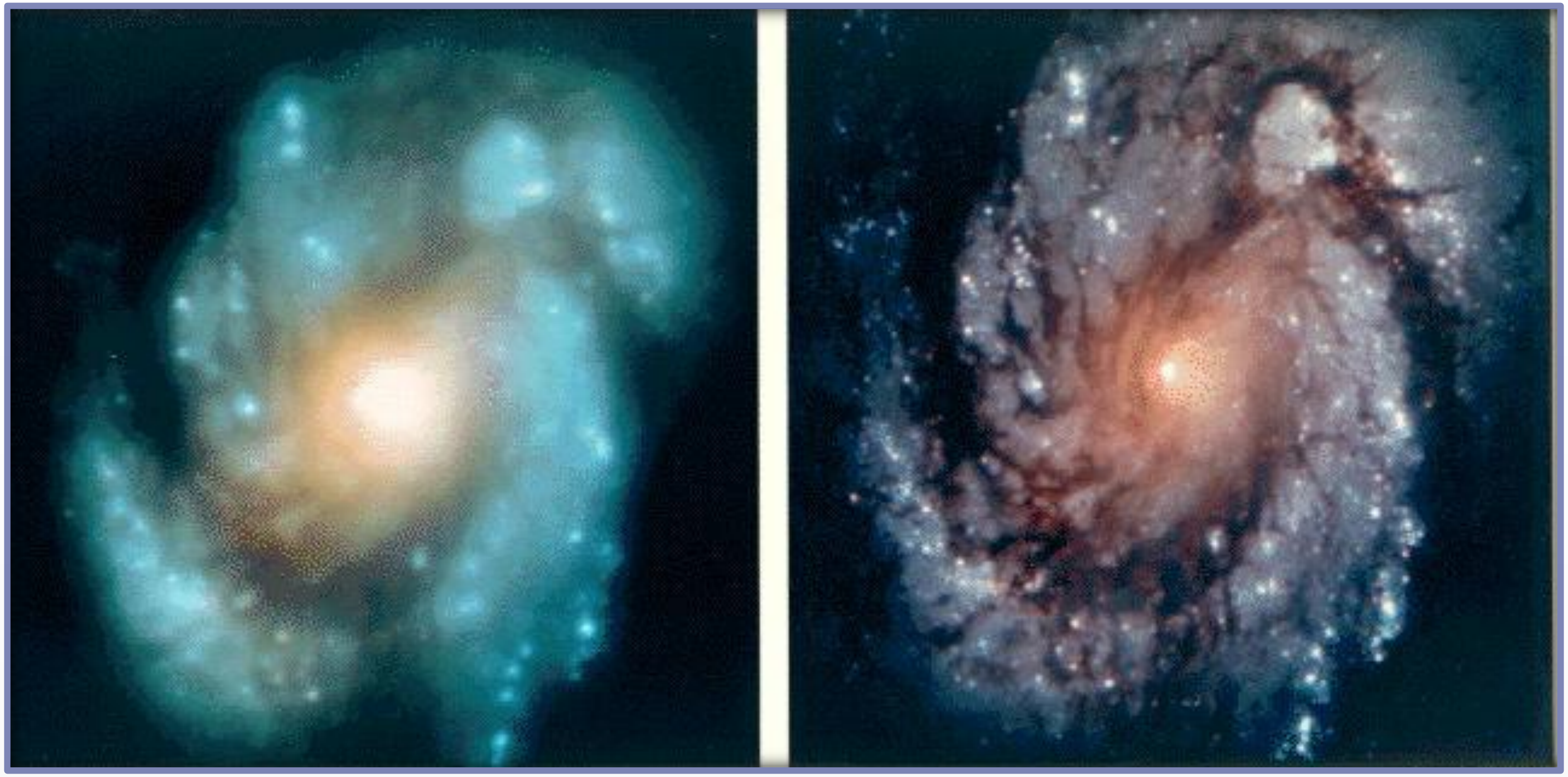
  "Gradient-based learning applied to document recognition"
- Still other methods are favored over NN
- 2012: Imagenet classification challenge won by deep CNN*

  Their error rate was 15.3%, whereas the second closest was 26.2%
- Since 2012 we are witnessing the golden age of CV

*Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105)

Applications

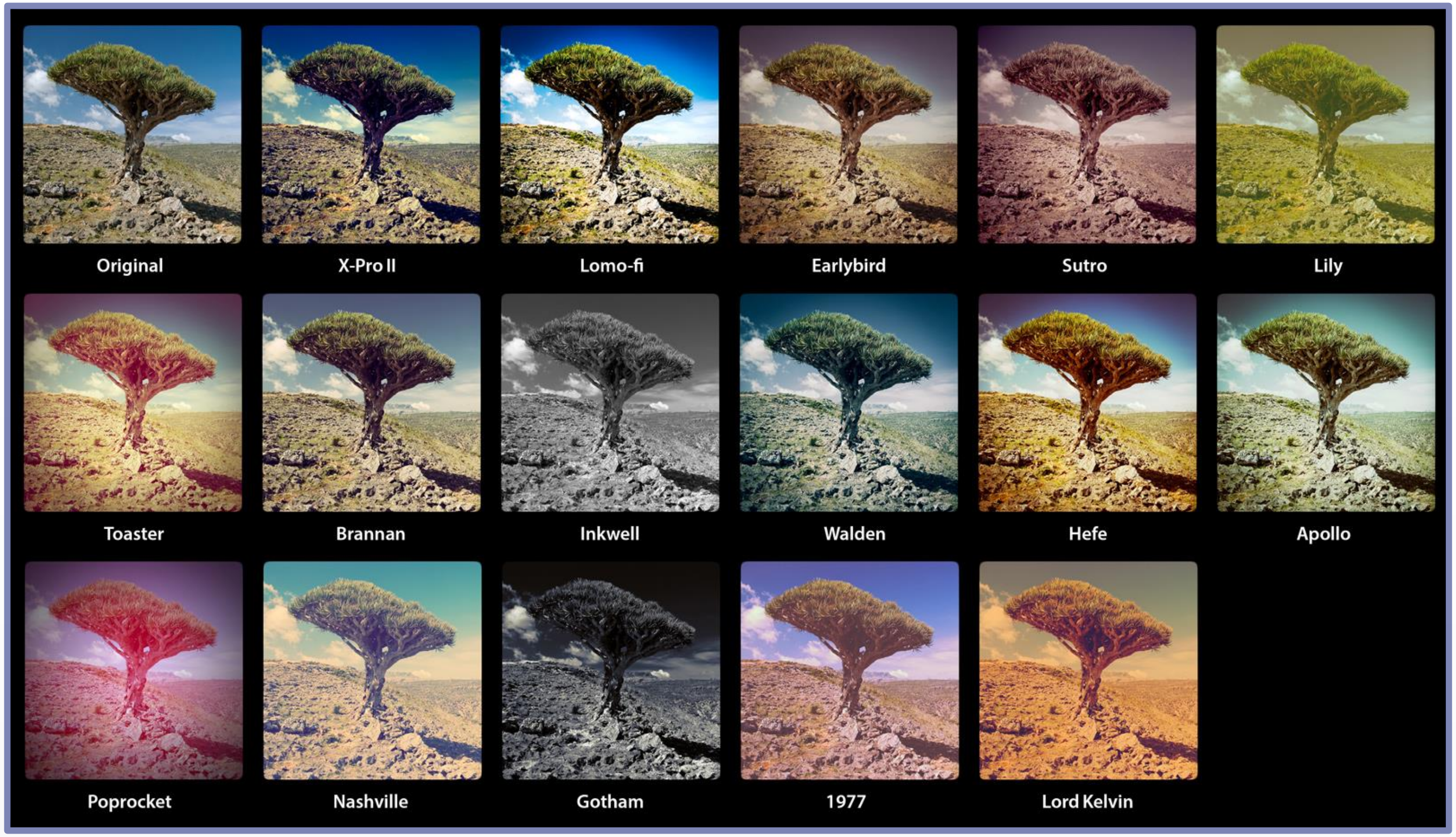
- Early Hubble Space Telescope images were distorted by a flawed mirror and could be sharpened by deconvolution.



Source: http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1077064

Applications

- Instagram filters



Source: http://www.thephoblographer.com/2013/01/17/instagrams-presets-come-to-lightroom/#.U5SDT_nV-PQ

Applications

- Panoramic images

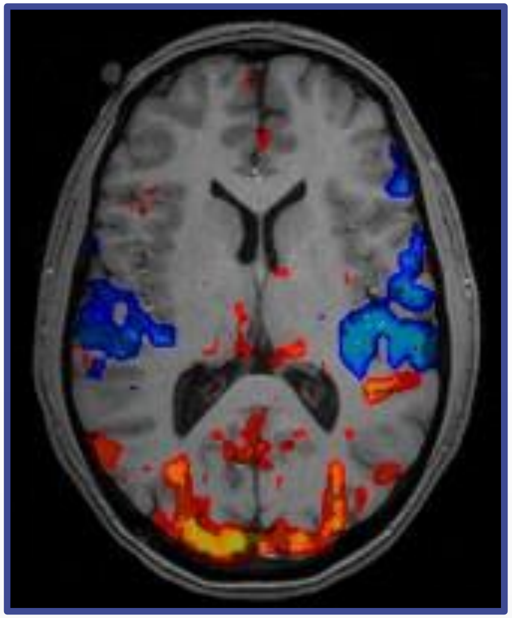
Source: http://en.wikipedia.org/wiki/User:Diliff

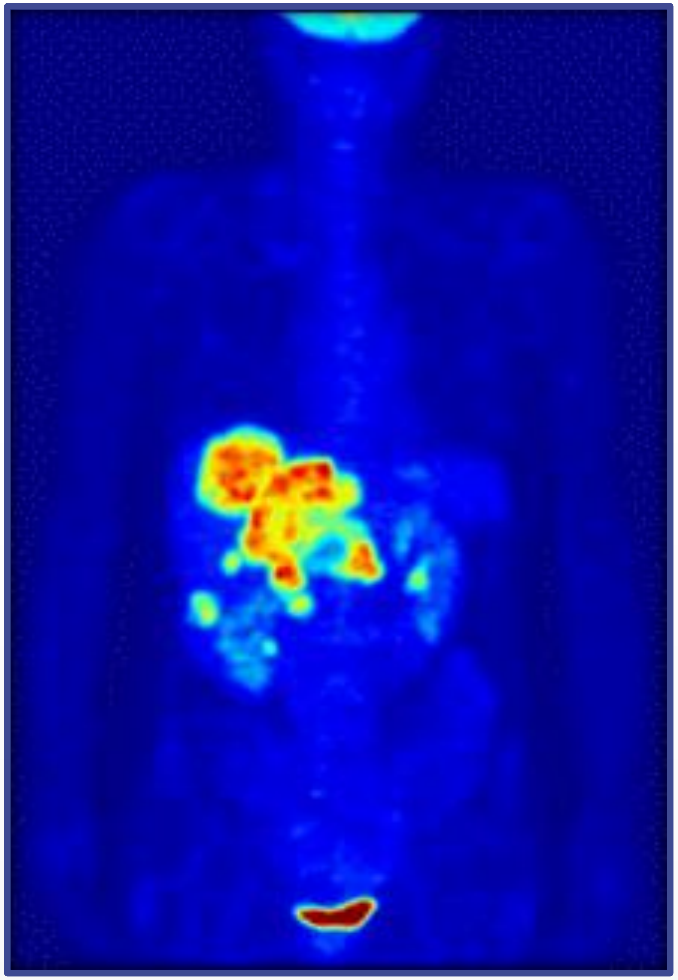
Thesis topic option @ PPKE/SZTAKI

Contact: Csaba Benedek

Applications

- Medical image processing:
  - Ultrasound
  - 3D imaging, MRI, CT, PET
  - Image guided surgery
  - …

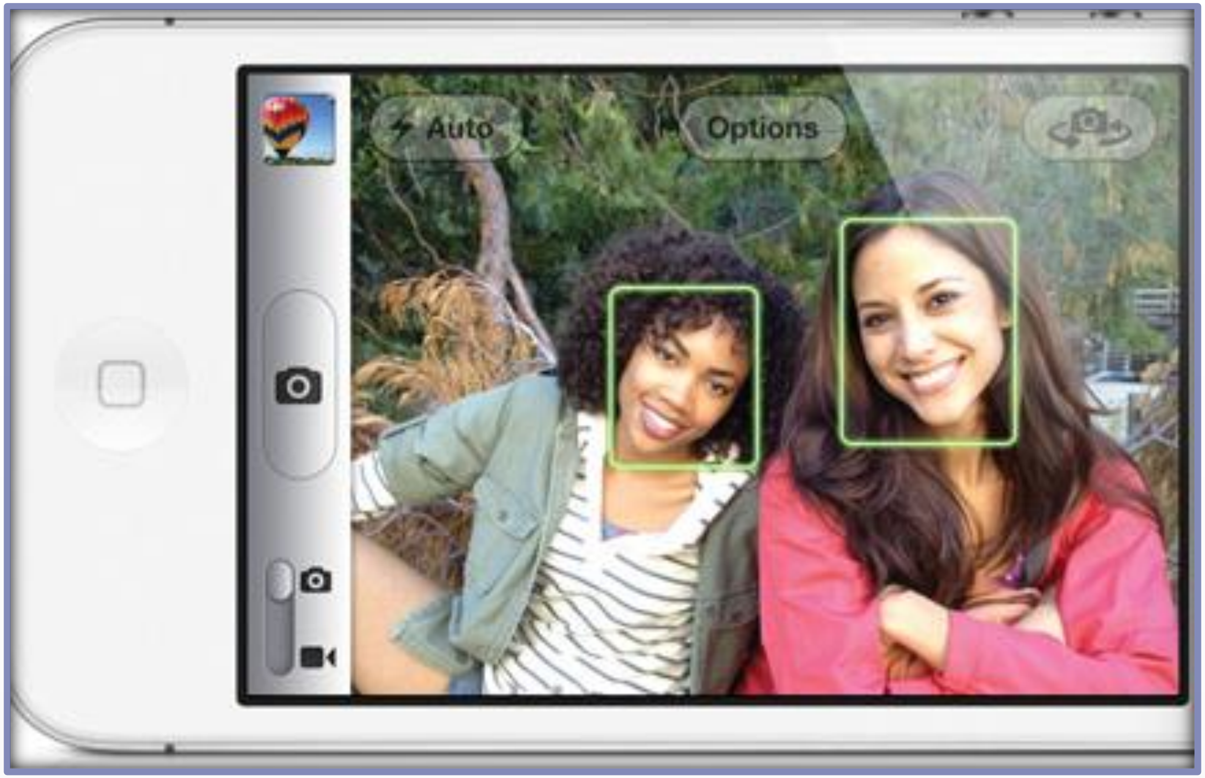




Source: http://en.wikipedia.org/wiki/Positron_emission_tomography

Applications

- Face detection is a standard feature in smart phones/cameras:

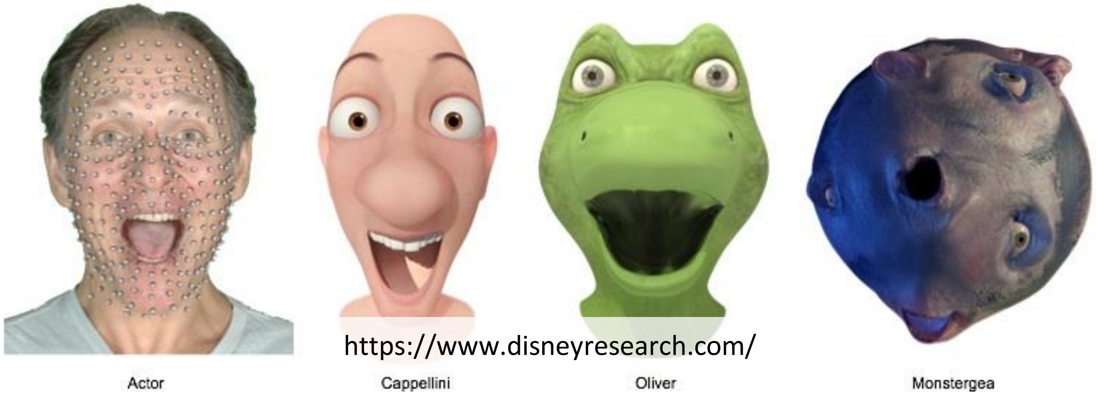


Source: http://www.imore.com/

Applications

- Facial retargeting:

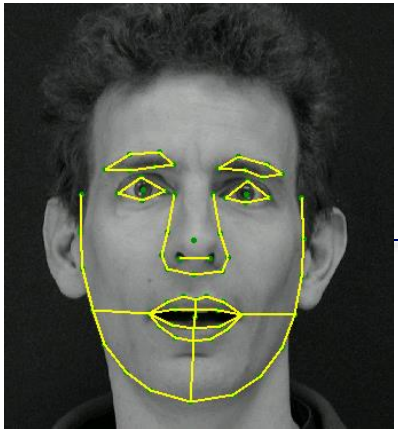
Entertainment

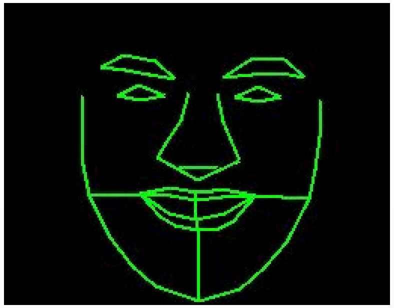


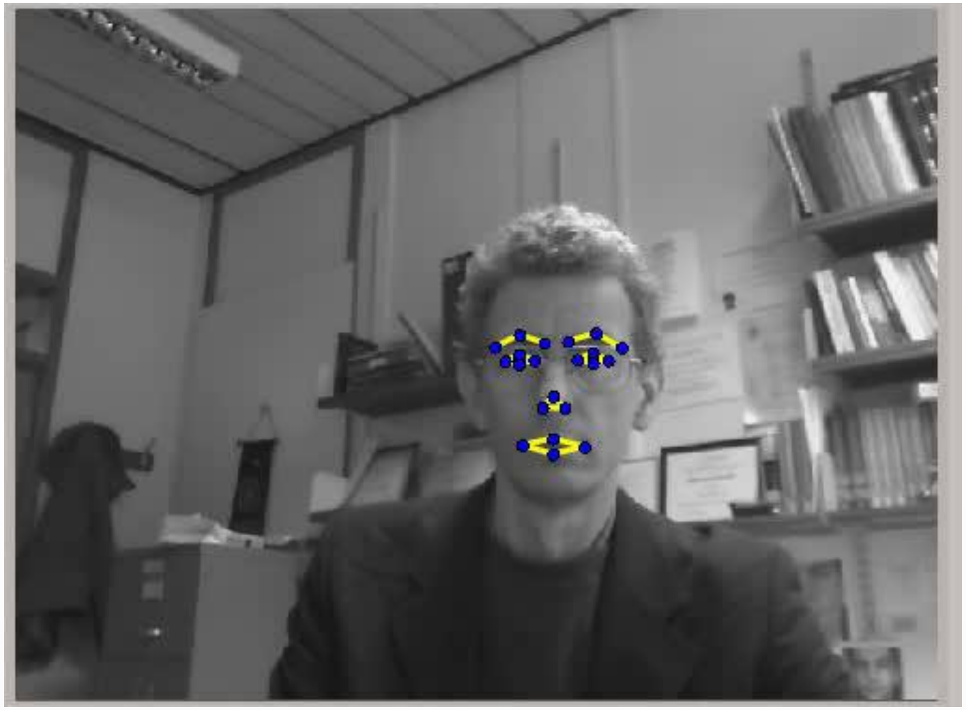
https://www.disneyresearch.com/

FMX 2017, Stuttgart

Active Shape Models (ASM)







T.F.Cootes, M.Ionita, C.Lindner and P.Sauer, "Robust and Accurate Shape Model Fitting using Random Forest Regression Voting", ECCV 2012

Applications

- Emotion recognition:

For market research

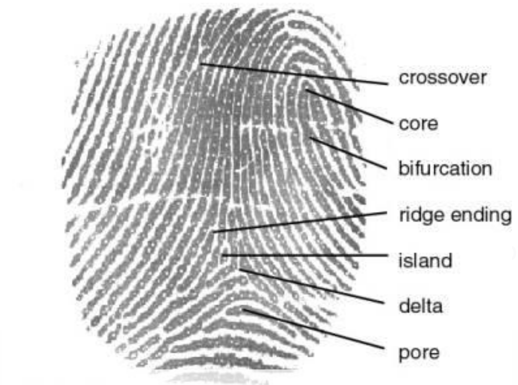
https://www.realeyesit.com/

Gaming

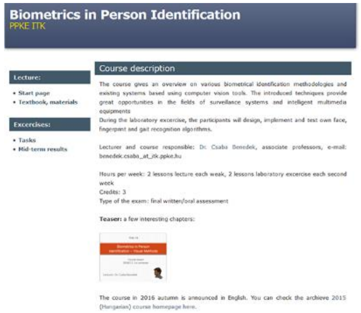
http://www.affectiva.com/

Applications

- Identity verification and recognition based on biometrics:



http://www.biometrics.gov/



Biometrics in Person Identification
PPKE ITK

Lecture:
- Start page
- Textbook, materials

Excercises:
- Tasks
- Mid-term results

Course description

The course gives an overview on various biometrical identification methodologies and existing systems based using computer vision tools. The introduced techniques provide great opportunites in the fields of surveillance systems and intelligent multimedia equipments
During the laboratory excercise, the participants will design, implement and test own face, fingerprint and gait recognition algorithms.

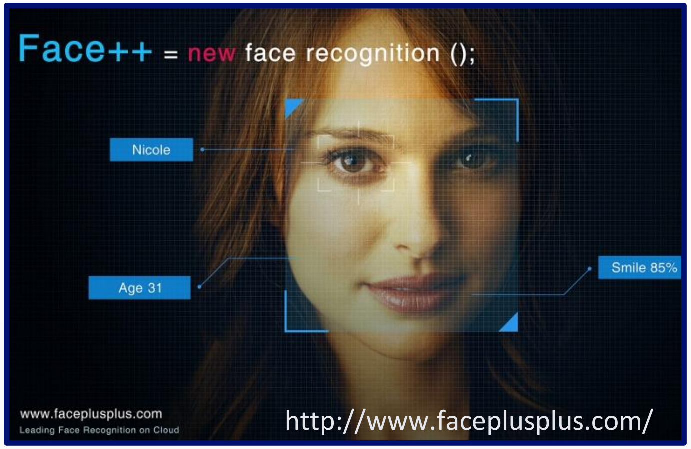
Lecturer and course responsible: Dr. Csaba Benedek, associate professors, e-mail: benedek.csaba_at_itk.ppke.hu

Hours per week: 2 lessons lecture each weak, 2 lessons laboratory excercise each second week
Credits: 3
Type of the exam: final written/oral assesament

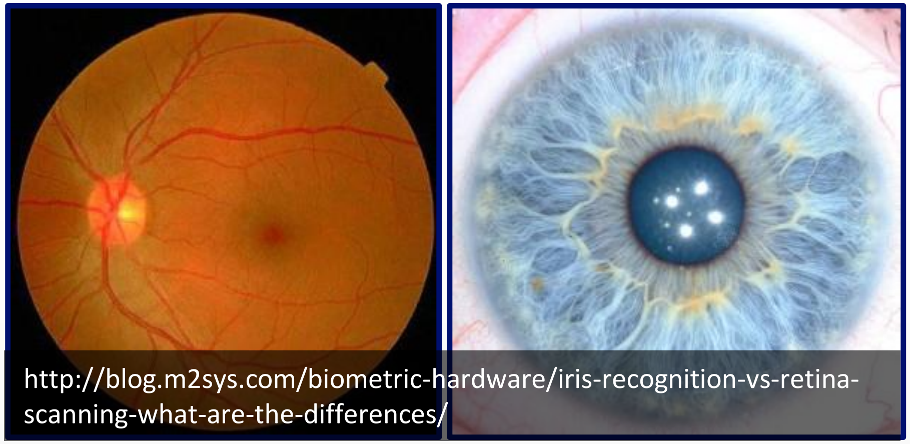
Teaser: a few interesting chapters:

The course in 2016 autumn is announced in English. You can check the archieve 2015 (Hungarian) course homepage here.

- See more: http://biometrika.itk.ppke.hu/


Face++ = new face recognition ();
Nicole
Age 31
Smile 85%
www.faceplusplus.com
Leading Face Recognition on Cloud

http://www.faceplusplus.com/



http://blog.m2sys.com/biometric-hardware/iris-recognition-vs-retina-scanning-what-are-the-differences/

Applications

- Video surveillance:

SUBITO EU project

http://vision.seecs.edu.pk/abnormal-event-detection/

http://www.trafficvision.com/

Thesis topic option @ PPKE/SZTAKI
http://mplab.sztaki.hu/geocomp
Contact: Csaba Benedek

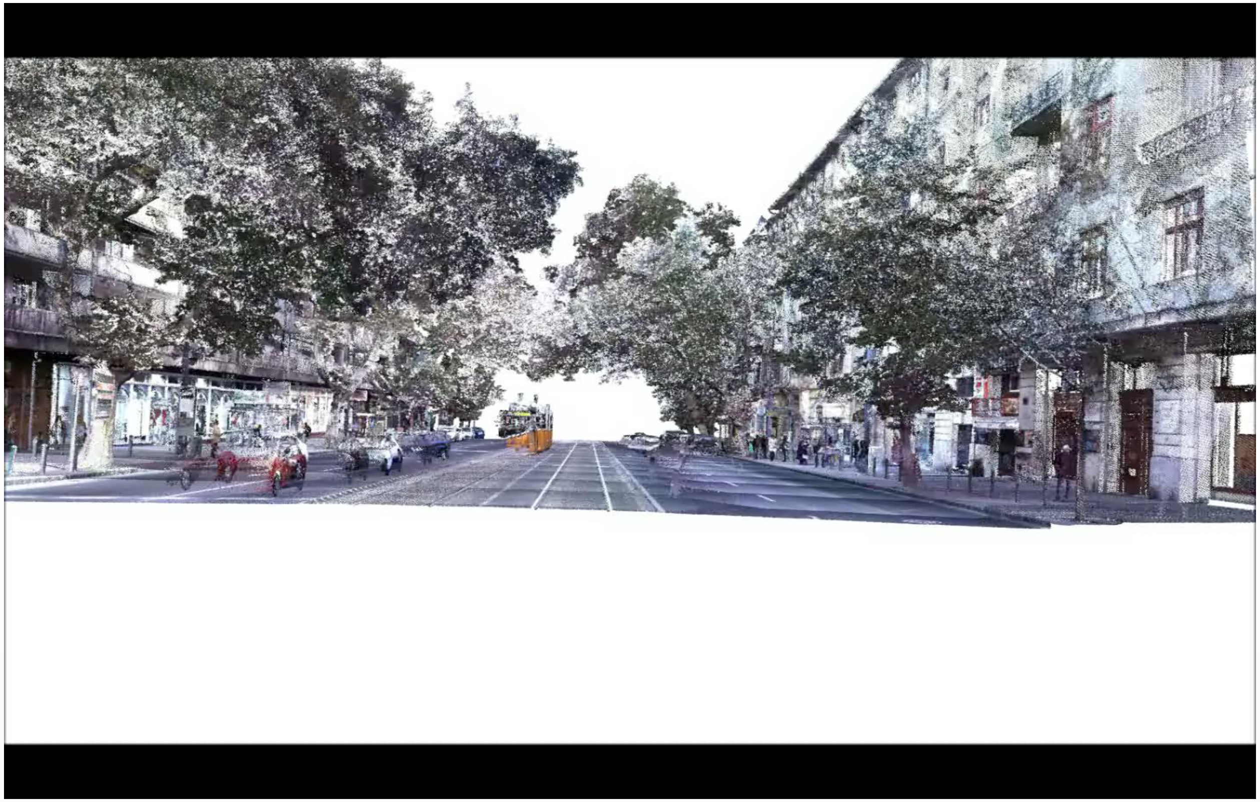
Applications
3D scene understanding via laser scanning (Lidar) data

Automated data filtering, correction, vectorization and interpretation

Thesis topic option @ PPKE/SZTAKI
http://mplab.sztaki.hu/geocomp
Contact: Csaba Benedek

Laser scanning

Point cloud classification



🟪	pedestrian
🟦	moving car
🟧	parking car
🟦	tall vehicle
⬜	column/bole
🟩	vegetation
⬛	road
⬛	building wall

B. Nagy, and Cs. Benedek: "3D CNN Based Semantic Labeling Approach for Mobile Laser Scanning Data", ***IEEE Sensors Journal***, to appear 2019

Applications

- Advanced driver assistance systems (ADAS)
  - Automatic parking
  - Collision avoidance system
  - Driver Monitoring System
  - Emergency driver assistant
  - Lane departure warning system
  - Lane change assistance
  - Pedestrian protection system
  - Traffic sign recognition
  - …

https://www.tesla.com/

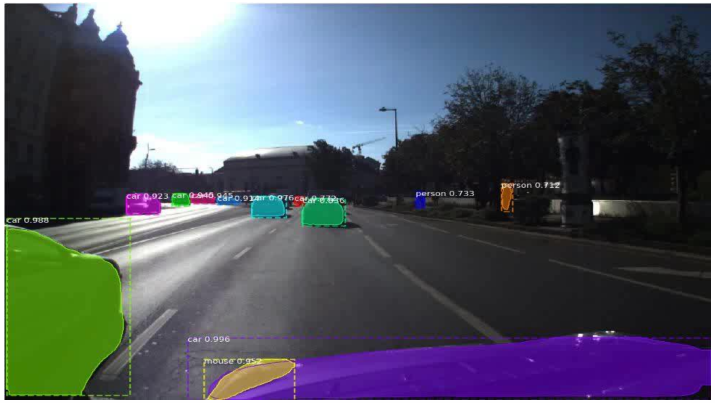
© Google

UBER

Thesis topic option @ PPKE/SZTAKI

http://mplab.sztaki.hu/geocomp

Lidar and image data fusion for self-driving cars in SZTAKI



Administration

Administration and Course Requirements

- Web:
  - http://kep.itk.ppke.hu
- Mailing list:
  - https://lists.ppke.hu/cgi-bin/mailman/listinfo/kepelemzes
  - You will be subscribed automatically
- Contacts:
  - Csaba Benedek: benedek.csaba at itk.ppke.hu
  - Miklós Koller: koller.miklos at itk.ppke.hu
  - Márton Bese Naszlady : naszlady.marton.bese at itk.ppke.hu

Administration and Course Requirements

- ***Lectures/Seminars***:
  - Start time: 8:15 AM
  - The attendance is obligatory
  - In every lecture between Week 2 and 12 there will be a short test from the previous lecture’s topic
    - Each time 2 short questions for a maximum of 2 points (in total 22 points)
    - > 60% (13.5/22 points) is a requirement to take the final exam
    - Recovery option: if (and only if) you fail to pass the threshold by the end of the semester, you will have a chance on Week 13 to correct your two worst/missing results.
    - no other occasions during the semester to replace missing tests!
  - No midterm or final test.

- ***Lab practice***:
  - The attendance is obligatory
  - There will be programming tasks (the default language is Matlab) that you have to complete and submit to the online submission system on time
  - You have to understand your code and be able to explain it.
  - Submitting all tasks in time is a requirement of passing the course.
  - You can collect maximum 11 points, threshold is 7 (details clarified by Miklós and Márton, see Lab01 „General course info”)

Administration and Course Requirements

- ***Assignments***:
  - During the semester we will hand out 4 longer programming assignments.
  - You have to complete and submit your solution via the online submission system
  - Maximum 22 points, threshold for passing: 14
- ***Oral exam*** at the end of the semester:
  - To be able to participate to the exam you have to pass **all three threshold** for the (i) short tests, (ii) programing practices and (iii) assignments.
  - ***Mid term points count in 30% into the final grade, 70% is coming from the oral exam***
- ***Offered Grade – only in expetional cases:***
  - On Week 13, you should write a 12th short test from the previous lecture, and you can recover 1 test from the previous ones
  - 90% rule: minimum 22 (out of 24) points in total from the tests, 10 points from the programing practices and 20 from the assignments

  -> You get a 5 as final mark
- ?

Human Vision

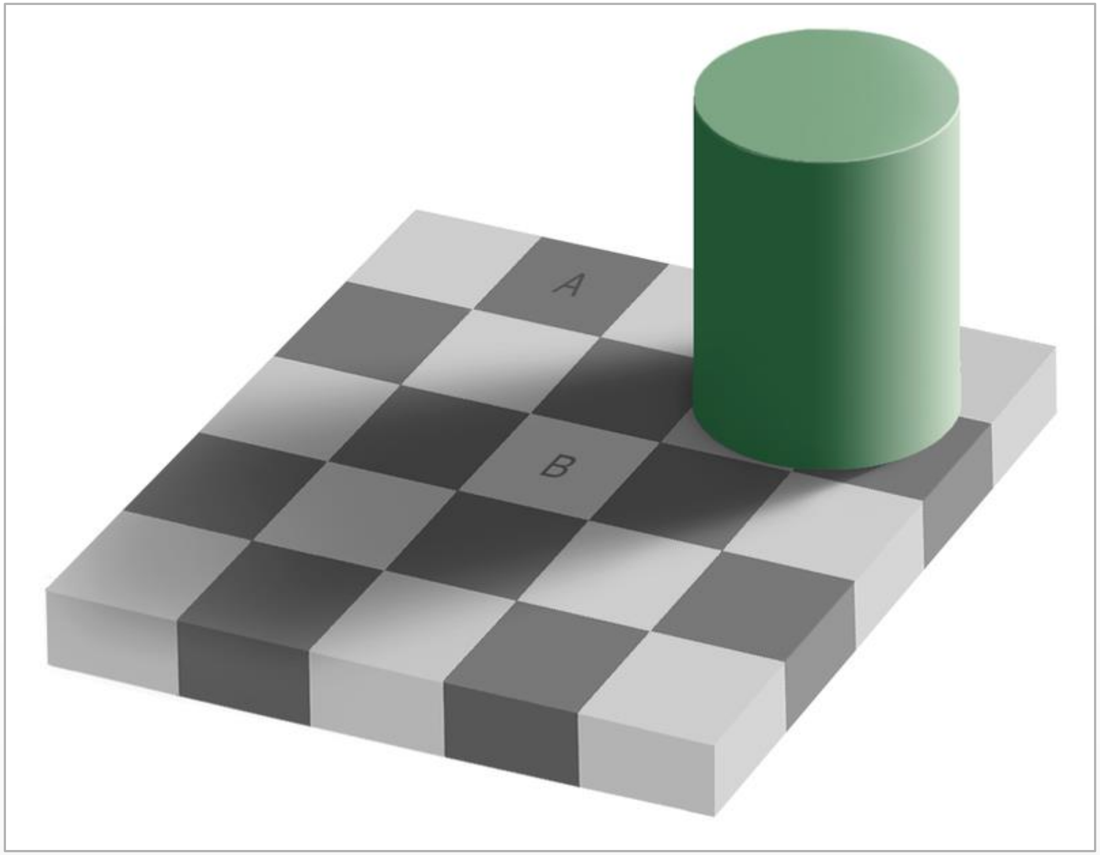
- The **human visual system:**
  - Gives us the ability to process visual stimulus, to be able to detect and interpret information from visible light (build a representation of the surrounding environment).
- The ultimate goal of computer vision is to build a system that is capable of seeing as a human can (or even better).
- It is not easy! Human vision was trained through the many years of the evolution. It can perform complex tasks (face/facial expression recognition) easily.
- For a computer it is still an unsolved problem: there is a gap between how a human and how a computer sees an image.
- Yet human vision is fallible. Illusions and ambiguities are encountered all the time.

Illusions

- The visual system is optimized to process natural images (through evolution)
- It is faced with an ill-posed problem:
  - Ambiguity due to projection from 3D to 2D image
  - Uncertainty due to incomplete knowledge of the environment
  - Uncertainty due to noise in photoreceptors and neurons
- The visual system relies on a set of assumptions to solve this ill-posed problem
  - Assumptions presumably learned via evolution
  - Assumptions tailored for the natural visual world
  - Assumptions cause illusions/failures under impoverished conditions
- Illusions can provide insights into the brain's assumptions.

Illusions

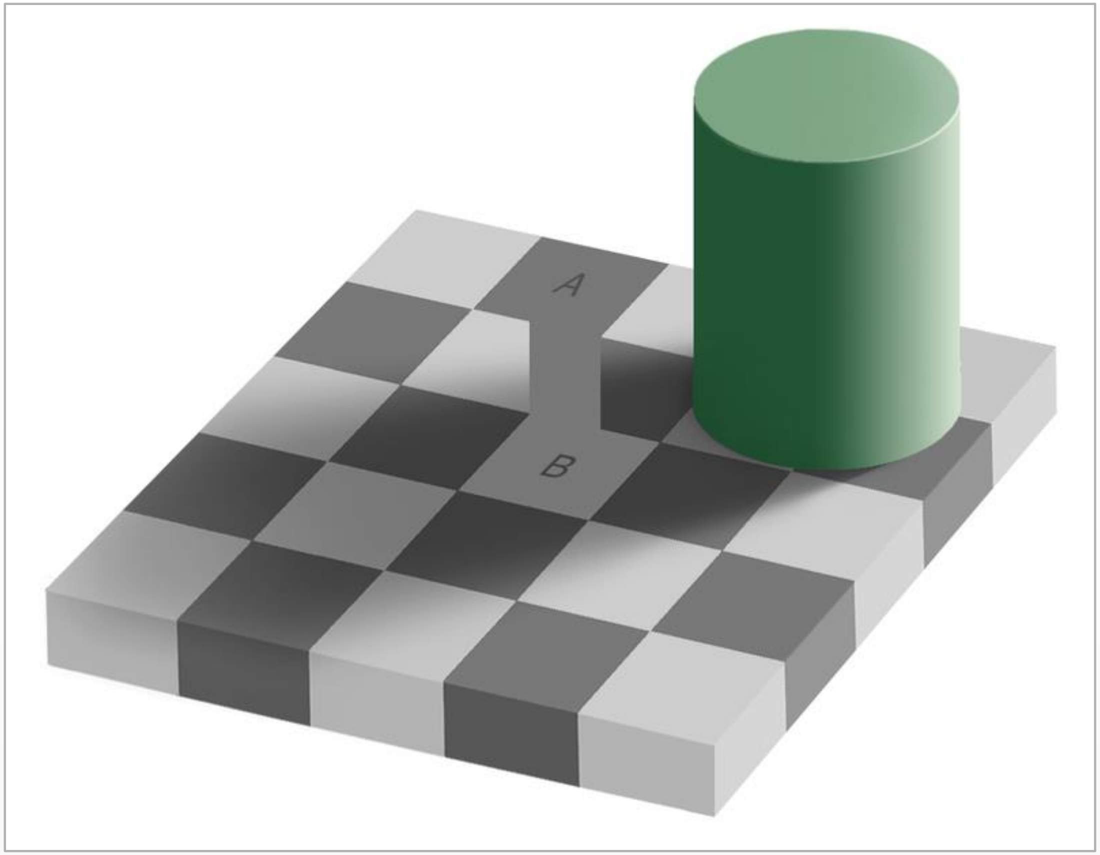
- Lateral inhibition + assumptions tailored for the natural visual world



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

Illusions

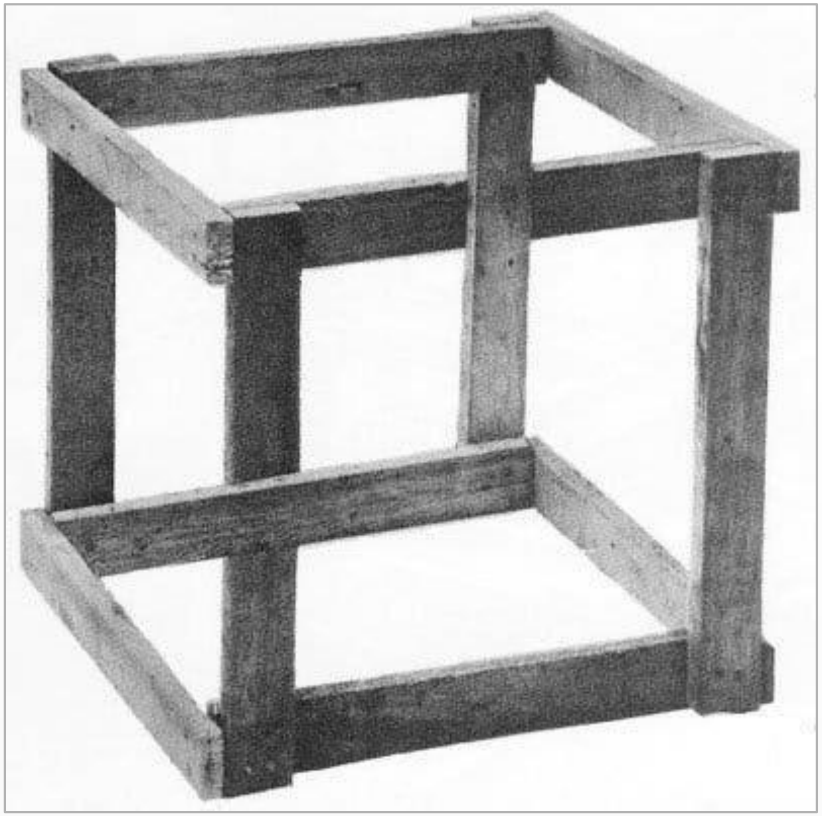
- Lateral inhibition + assumptions tailored for the natural visual world



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

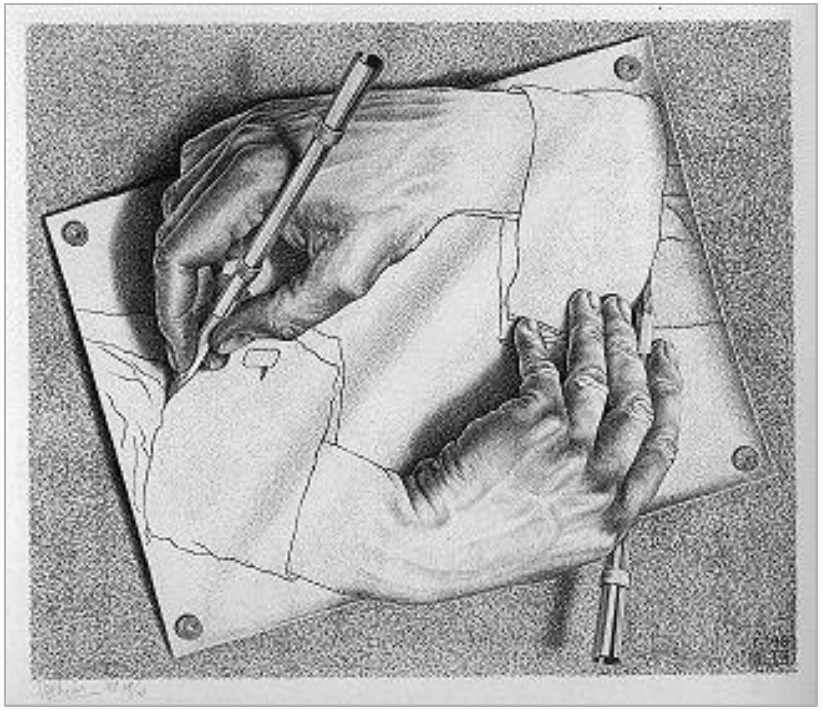
Illusions

- Ambiguity due to projection from 3D to 2D image



Illusions

- Ambiguity due to projection from 3D to 2D image

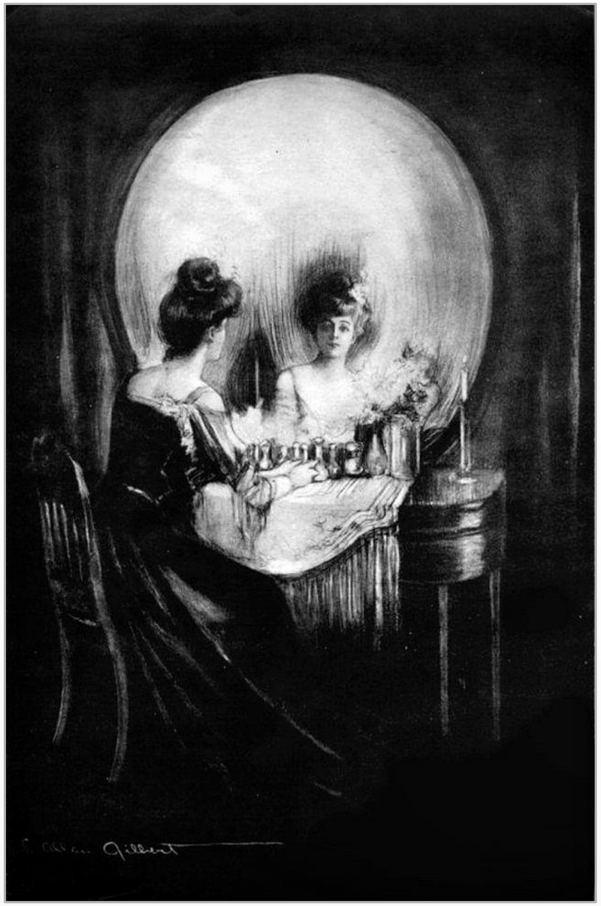


Maurits Cornelis Escher

Drawing Hands (1948)

Illusions

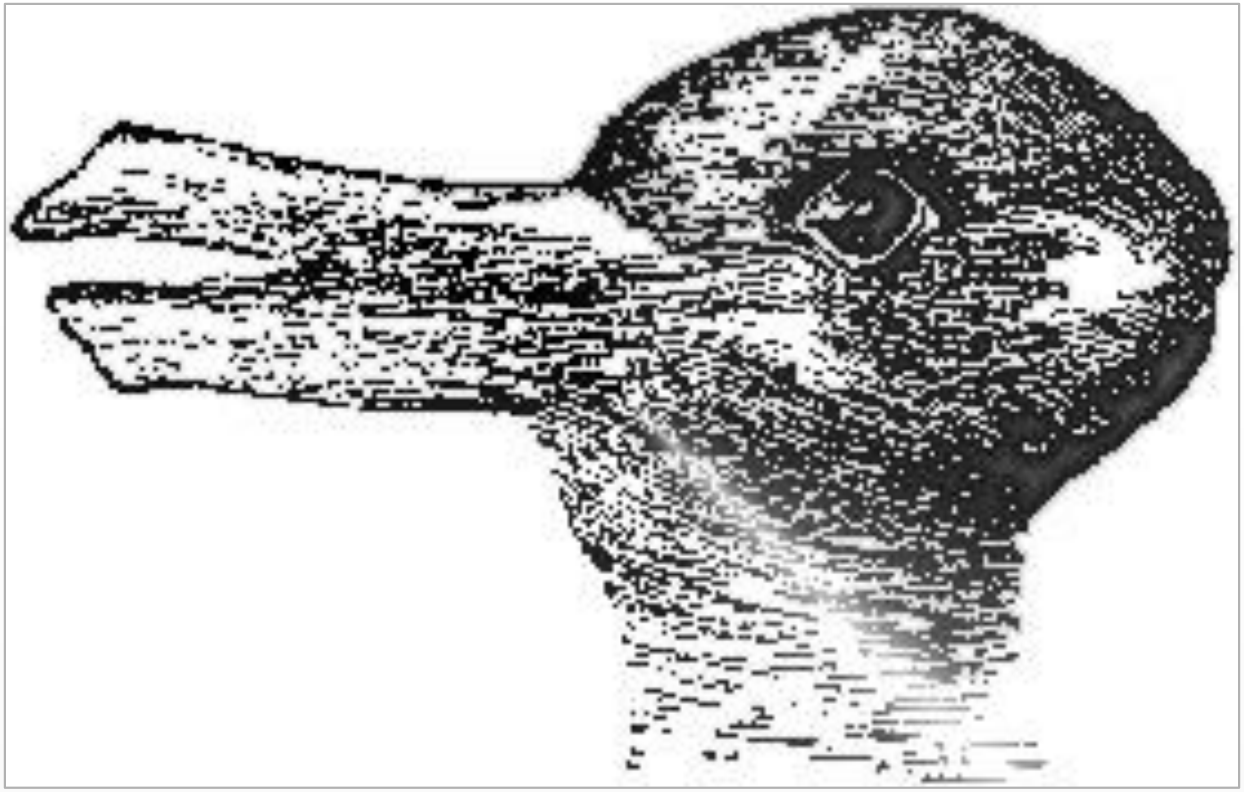
- Uncertainty due to incomplete knowledge of the environment



Charles Allan Gilbert
All Is Vanity (1892)

Illusions

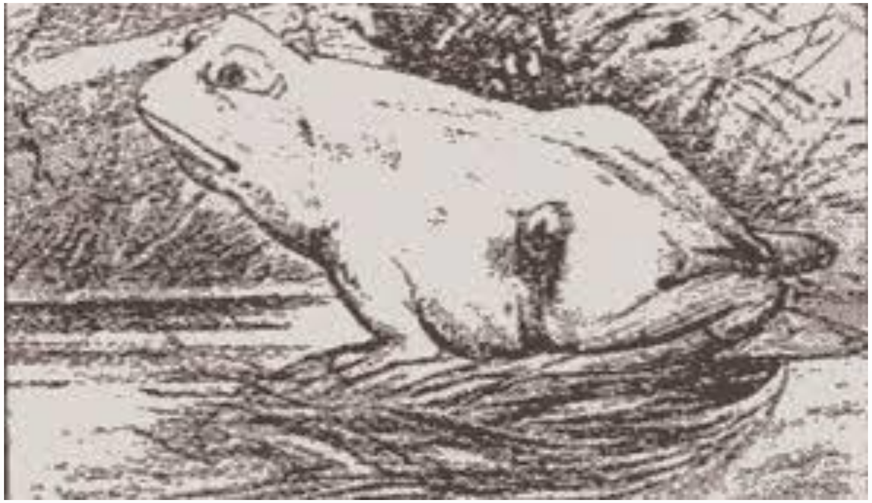
- Uncertainty due to incomplete knowledge of the environment



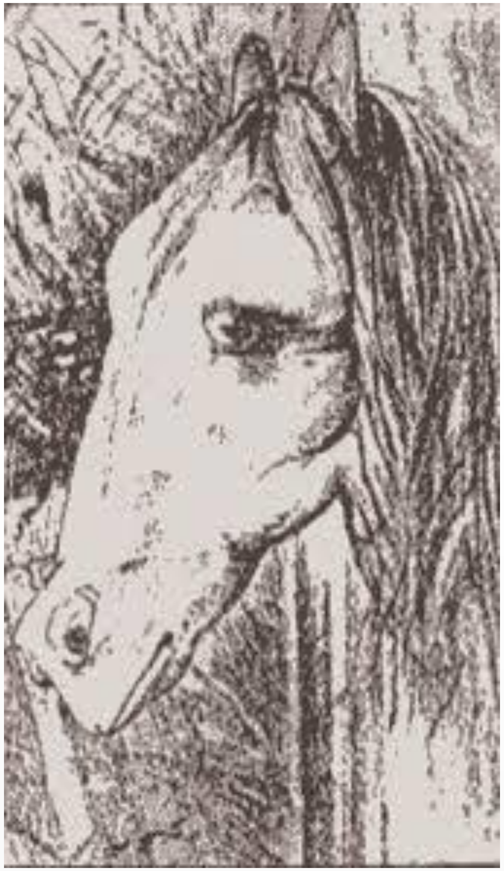
Ludwig Wittgenstein

Rabbit and Duck (1892)

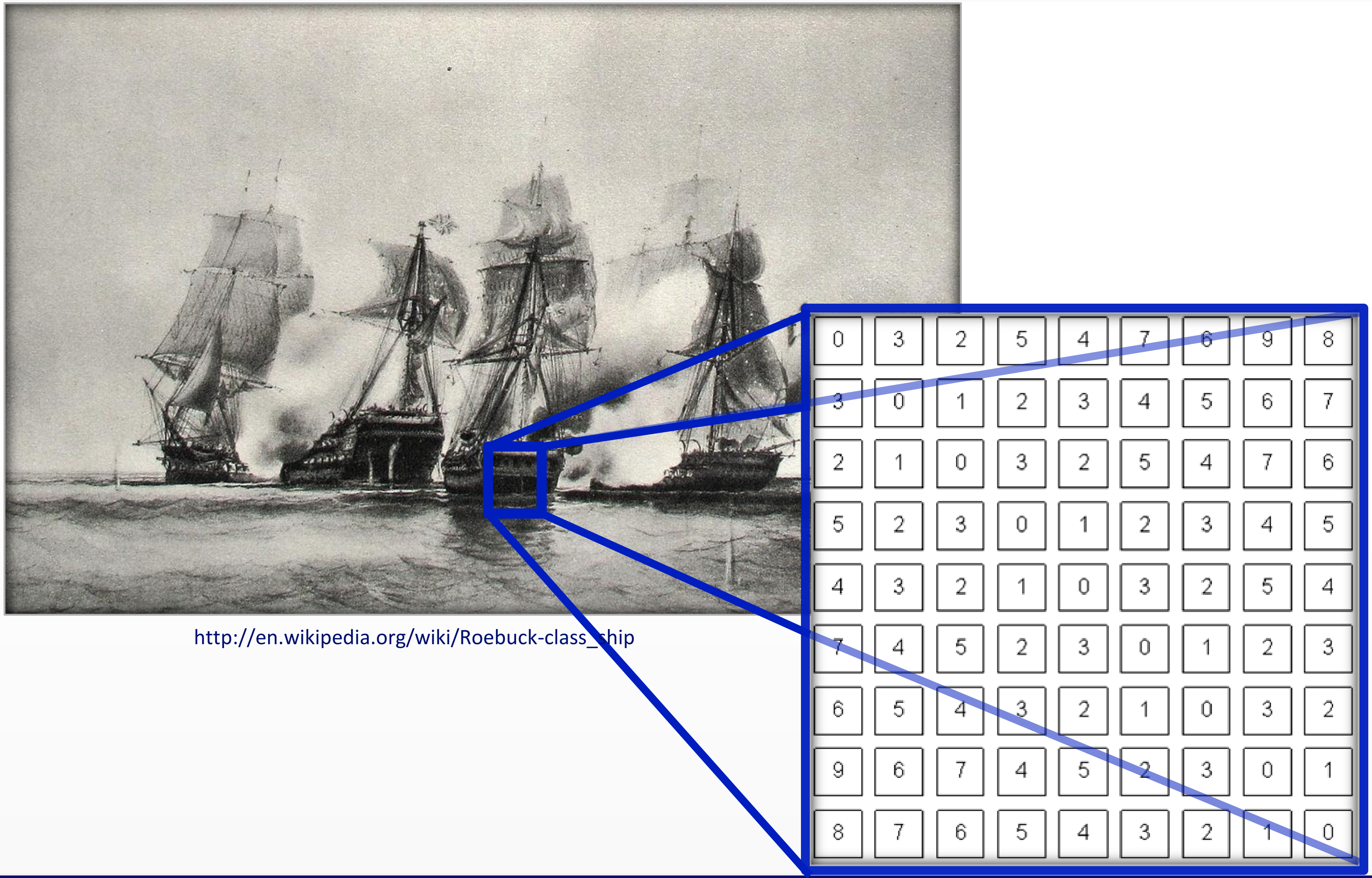
Illusions



Frog & horse



What does a computer „see”?

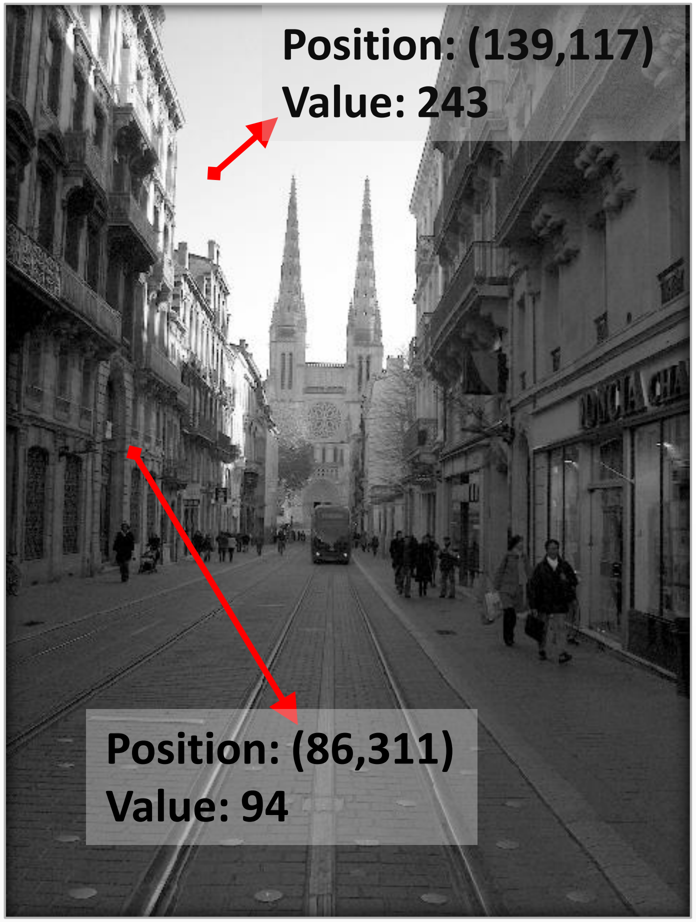


0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

http://en.wikipedia.org/wiki/Roebuck-class_ship

What does a computer „see”?

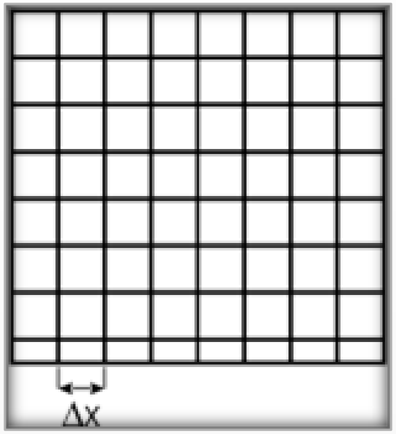
- A **digital image** is discreet representation of a continuous measurement, usually a **2 or 3 dimensional array**.
- An element of this array is a **pixel** (picture element).
- A pixel has a **position** (its coordinates on the image) and an **intensity value**.
- A digital image is discretised both in space and intensity:
  - Spatial discretisation is referred to as **sampling.**
  - Intensity discretisation is referred to as **quantization.**



Sampling

- **Sampling** is the reduction of a continuous signal to a discrete signal.
- A finite set of values (called **samples**) are selected to represent the original continuous signal.

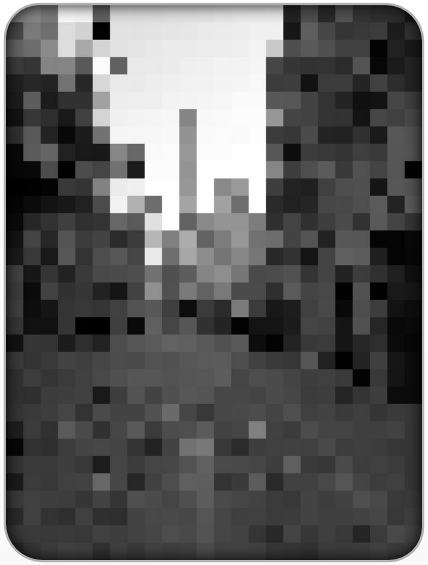
- In case of 2D signals (images) a grid is used for sampling
- The grid points will be represented as pixels.
- The frequency of the sampling defines:
  - How many grid points we have?
  - What is the resolution of the image?
  - How detailed the discretised image is?



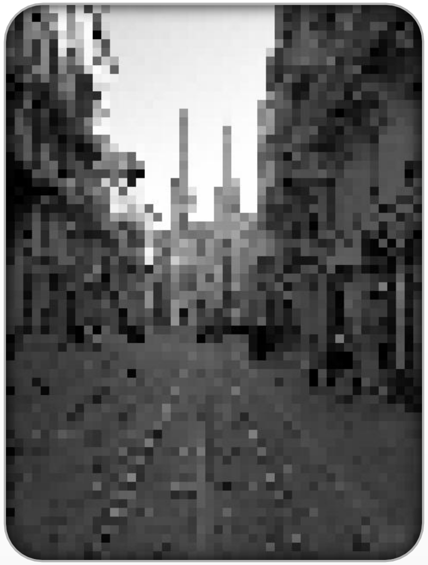
Sampling

- Sampling usually leads to information loss.
- The sampling frequency determines how much information we lose.
- We have to decide what is the smallest detail that we want to keep:

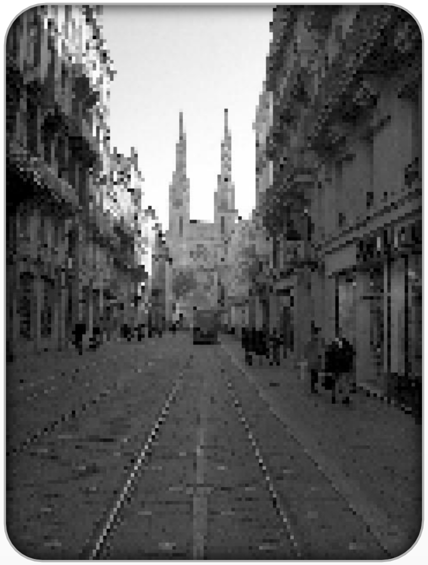
24x32



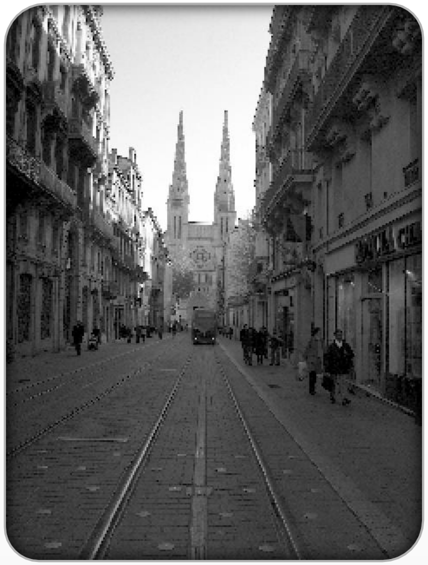
48x64



120x160

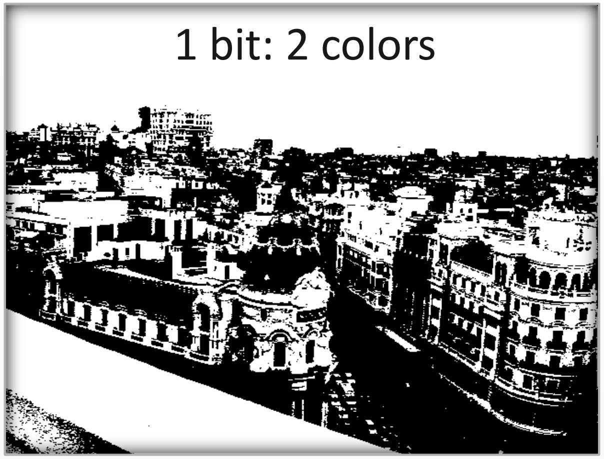


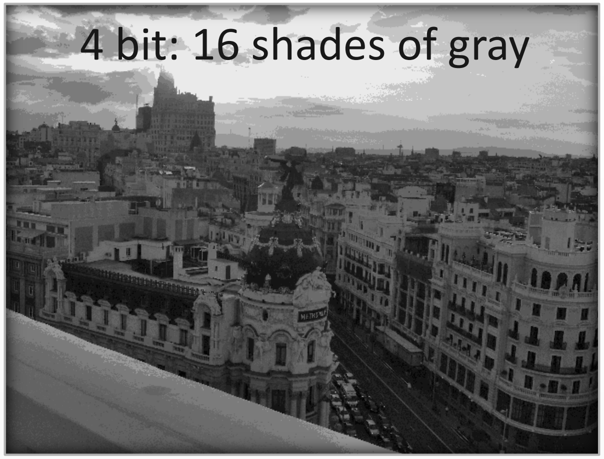
480x640

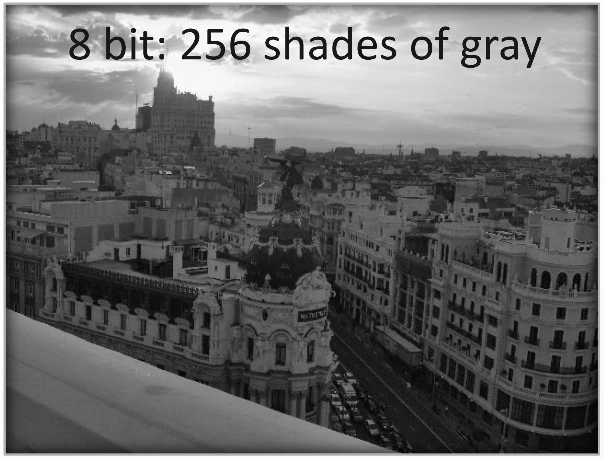


Quantisation

- Intensity discretisation is referred to as **quantization.**
- The digital image quality is highly depending on how many bits we use for coding the discreet intensity values:
  - Binary: each pixel is coded on 1 bit (zero or one, black or white)
  - Gray scale coded on 2/4/8/16/24/32 bits


1 bit: 2 colors


4 bit: 16 shades of gray


8 bit: 256 shades of gray

The Histogram of an Image

- **Histogram**:
  h(k) = the number of pixels on the image with value *k.*

Original Image*

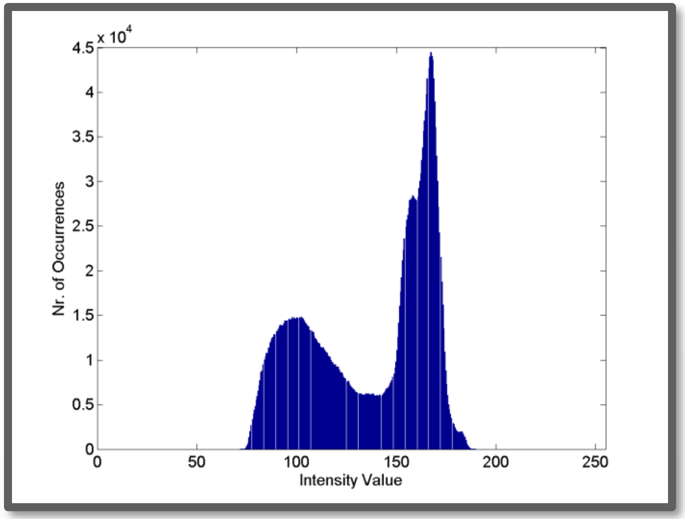
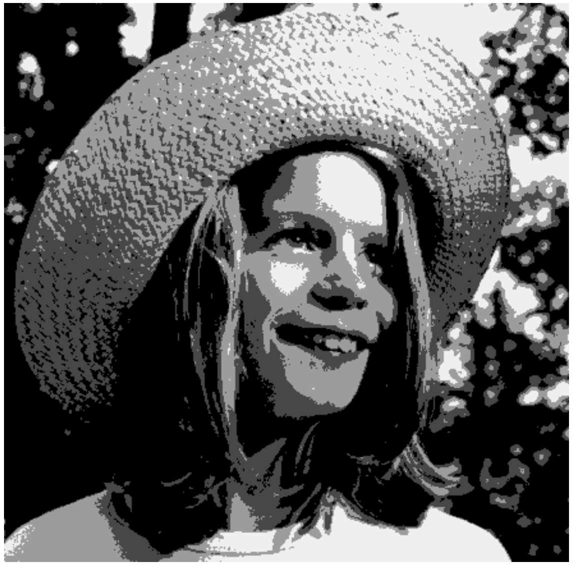


Image Histogram

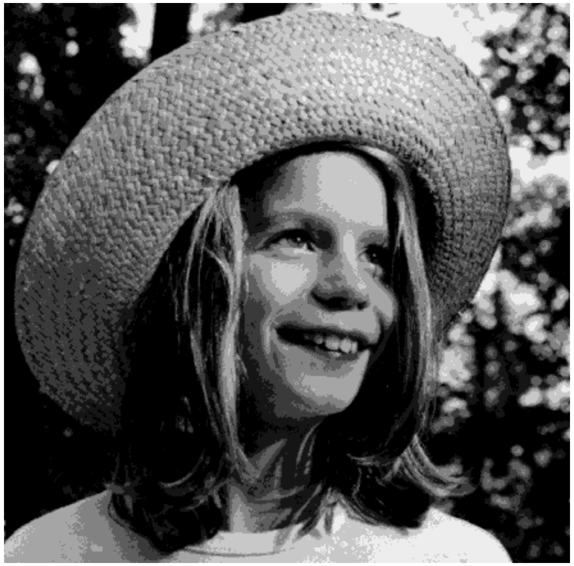
- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

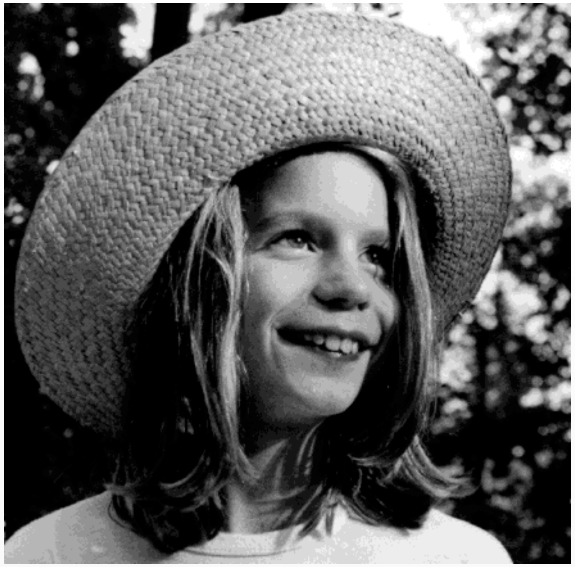
Image representation with different gray level depths



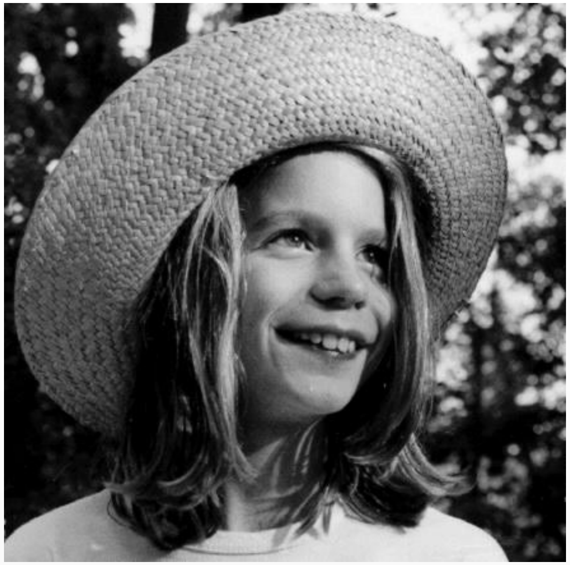
2 bit
(4 values)



3 bit
(8 values)



4 bit
(16 values)

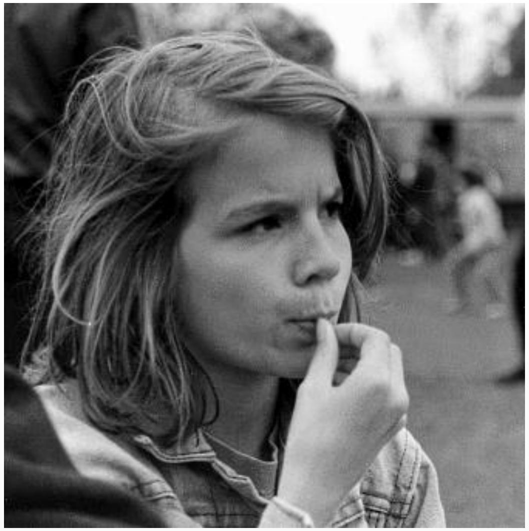


6 bit
(64 values)

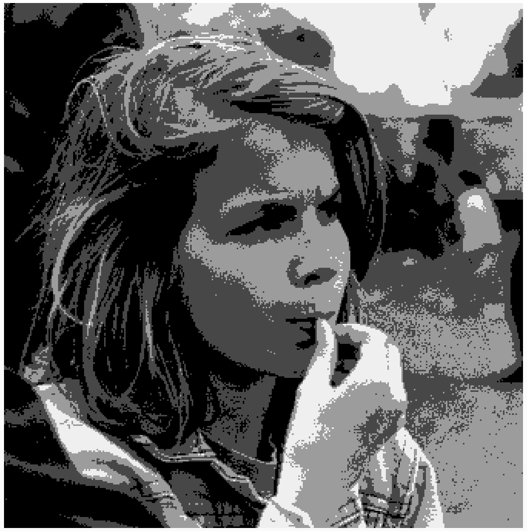
Slide credit ® Vladimir Székely, BME

Dithering

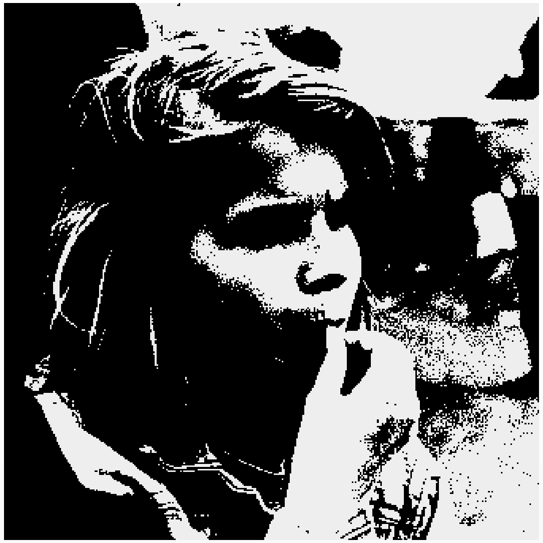
- Mapping to reduced bit number
  - Trivial solution: truncating the digital word representing the gray value of each pixel
    - pixel=pixel>>4



Original image



4 gray levels



2 gray levels

Dithering with white noise

- Quantitatization for a single bit
  - Each pixel becomes black or white
  - Probability of a given pixel being black or white depends on its original gray level
  - For a grayscale image f let the domain of gray levels of 0…F_{max}
    - A given pixel receives 1 (white) color, if the following condition holds:

$$f \geq random(\ )\cdot F_{\max}$$

- where random() is a randomly generated number from [0,1]

Slide credit © Prof. Vladimir Székely, BME

Dithering with white noise

- With some re-arrangement:

$$f \geq random(\ )\cdot F_{\max}$$

$$f + (1 - random(\ ))\cdot F_{\max} \geq F_{\max}$$

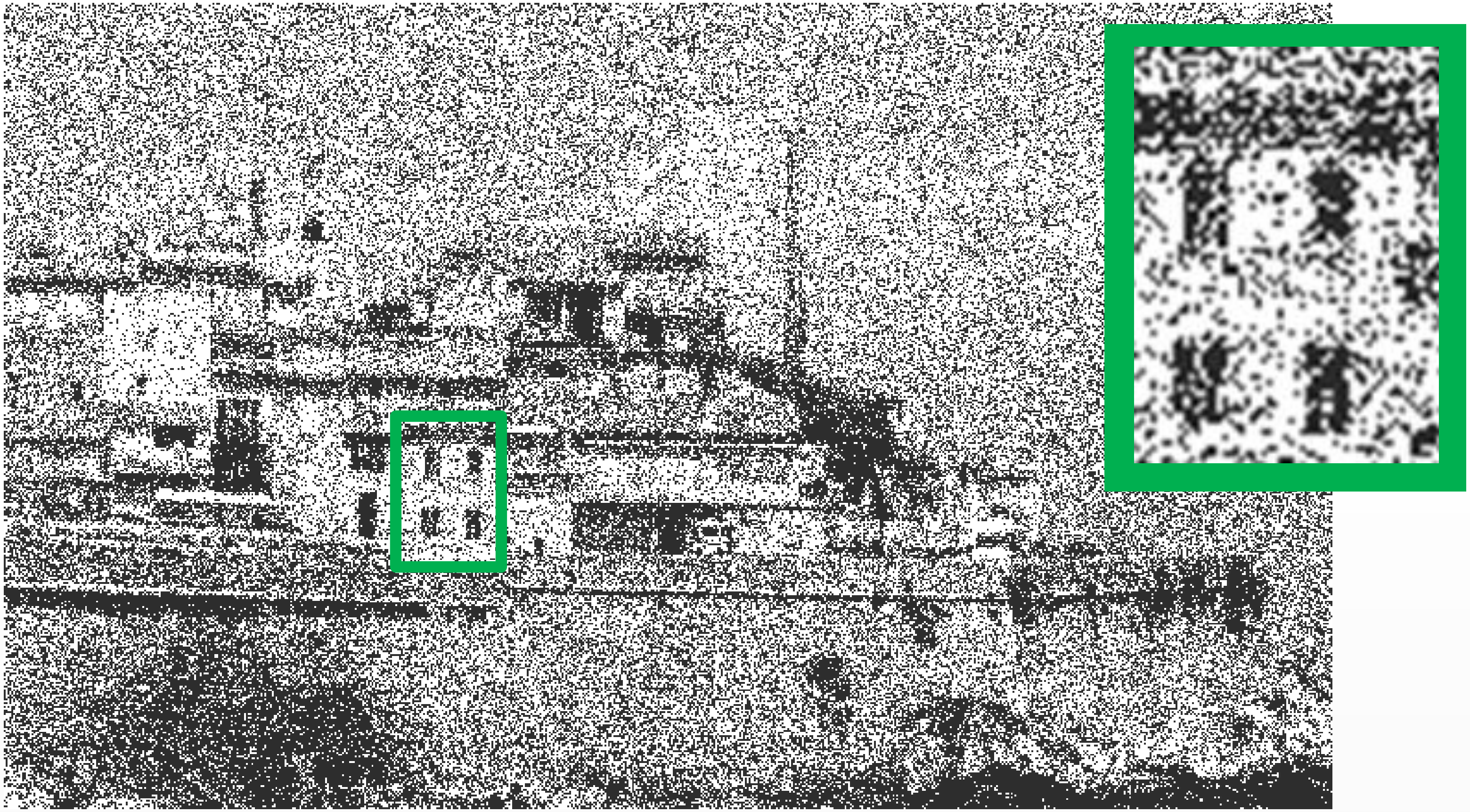
- The algorithm:

$$f + random(\ )\cdot F_{\max} \geq F_{\max}$$

  - we add random numbers („noise”) with values between 0 és F_{max} to the original grayscale image
  - Pixels with values above F_{max} receive label 1 (white), the other ones 0 (black)

Slide credit © Prof. Vladimir Székely, BME

Dithering with white noise

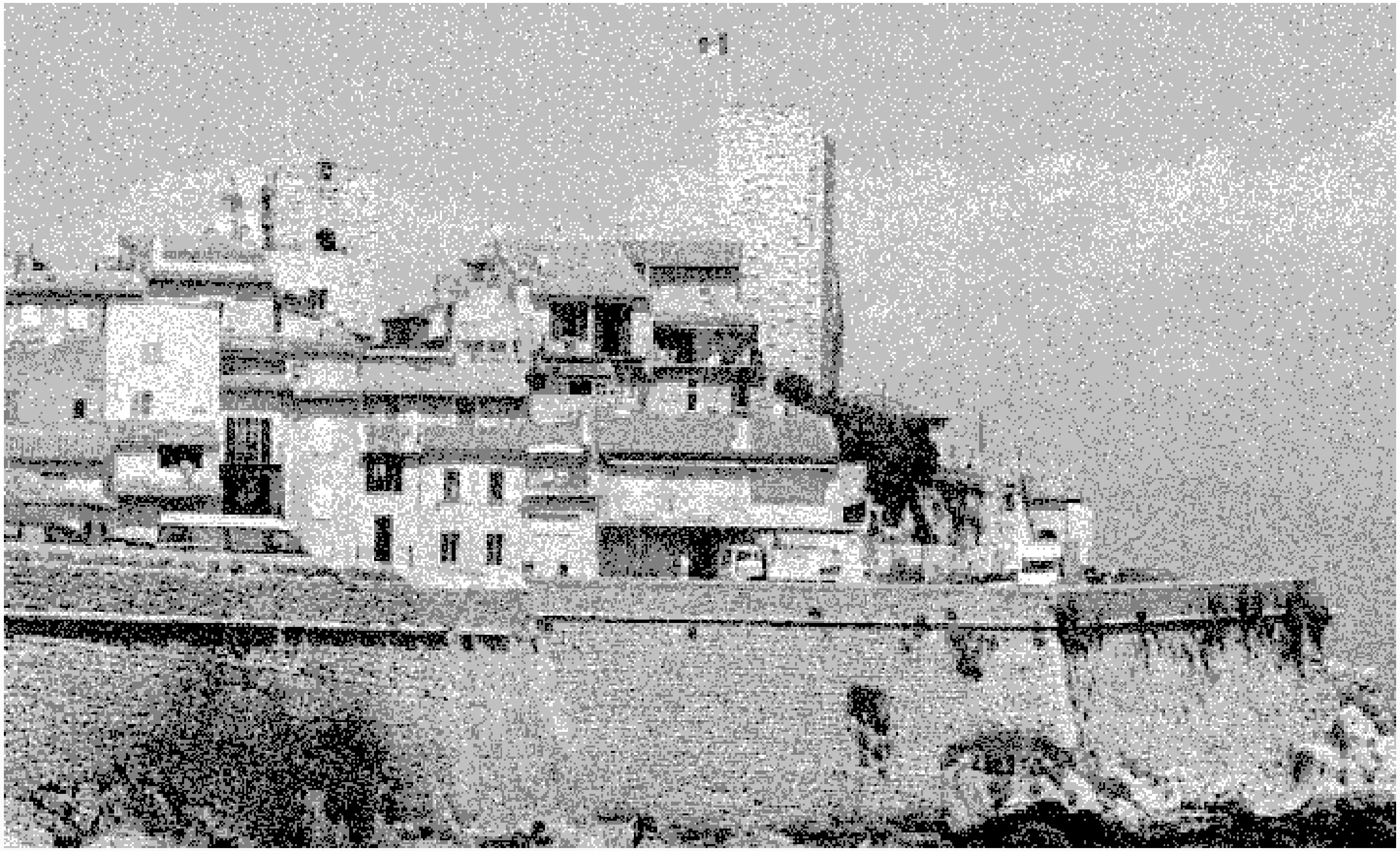


Mapping to 1 bit

Dithering with white noise – multiple bits

- If the gray value of the output image can be represented by multiple (2,3,…) bits, the result of dithering can be significantly enhanced.
- Let n be the number of different gray levels in the output image
- Algorithm
  - Add random numbers (noise) between 0 and $F_{max}/(n-1)$ to the image pixel values
  - Pixels with the modified gray levels larger than $k \cdot F_{max}/(n-1)$ but not larger than $(k+1) \cdot F_{max}/(n-1)$ receive the output value k .

Dithering with white noise



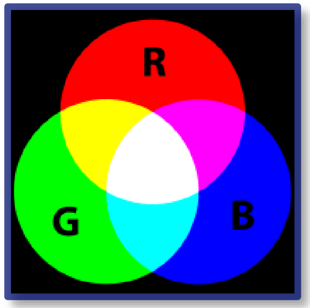
Mapping to 2 bits

Color Images

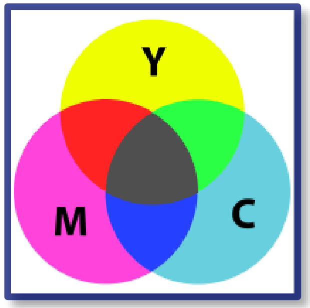
- Humans can distinguish thousands of color shades and intensities, but only a few dozens of gray.
- Color can be a useful descriptor for image segmentation, tracking, detection,…

Color models

- Two main color mixing models:
  - Additive model:



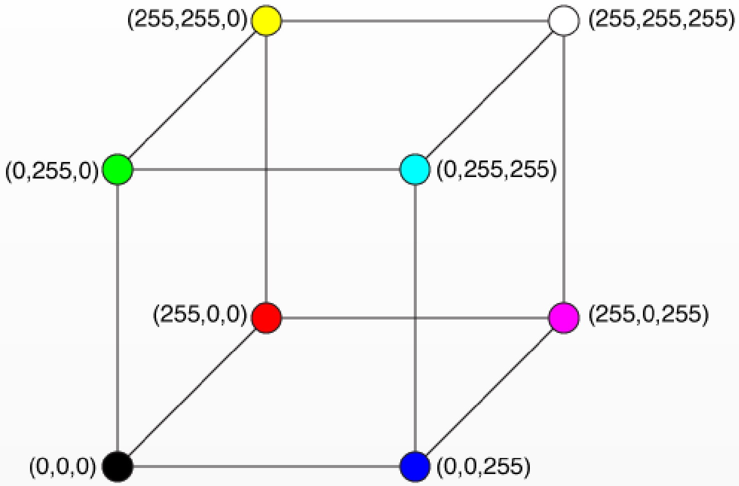
Subtractive model:



- Color spaces:
  - They specify a coordinate system and a subspace within that system, where each color is represented by a single point.
    - RGB
    - CMY, CMYK
    - HSL/HSV/HSI
    - YUV, YCbCr

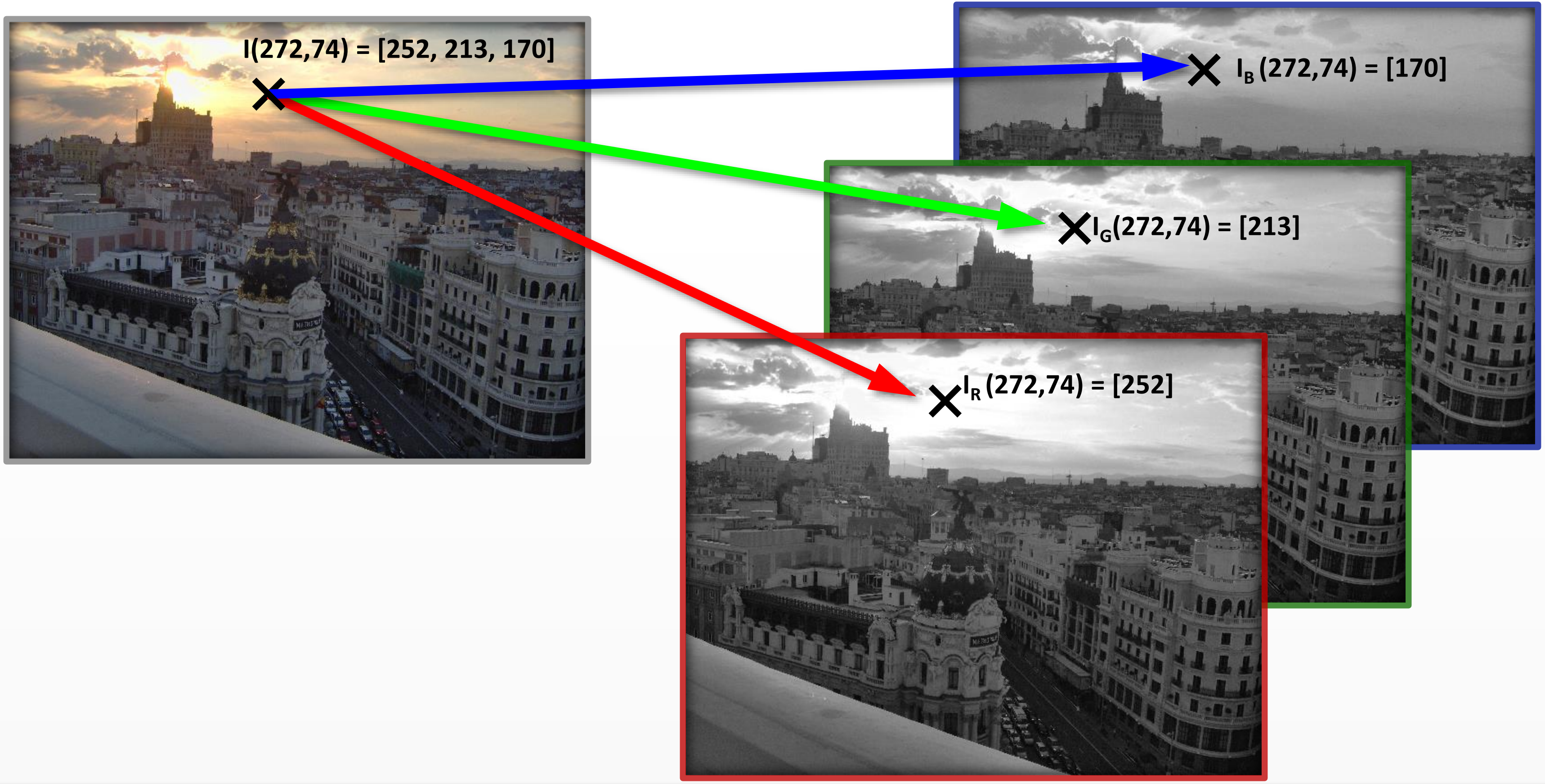
Color Spaces

- **RGB**:
  - Most common color model
  - Channels: Red, Green, Blue
  - All components are depending on luminosity
  - All channel needs to be coded with the same bandwidth
  - Changing the intensity level is not efficient, all 3 channels has to be modified

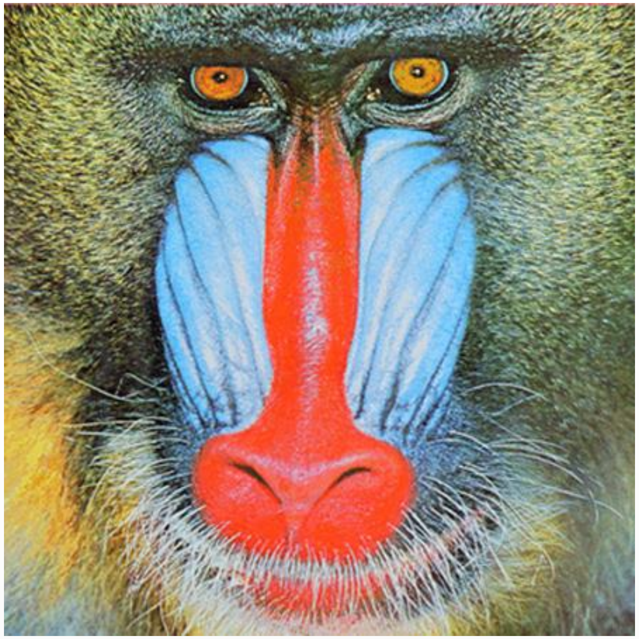


Color Images

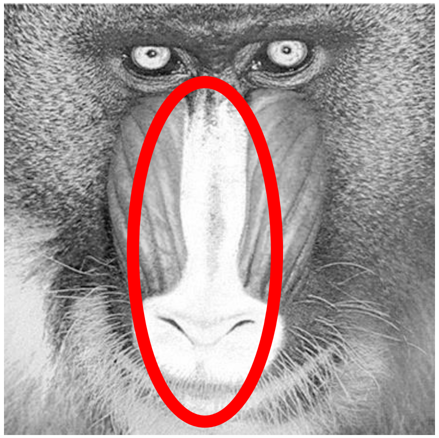
- Color images are formed by combination of different color planes



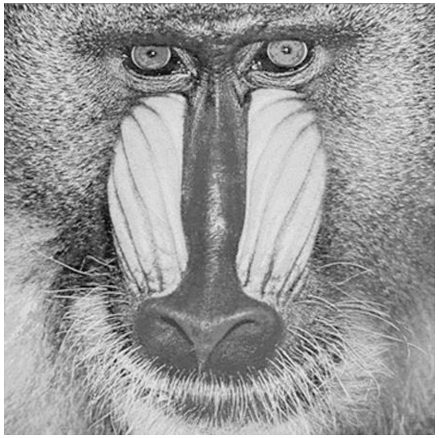
RGB channels



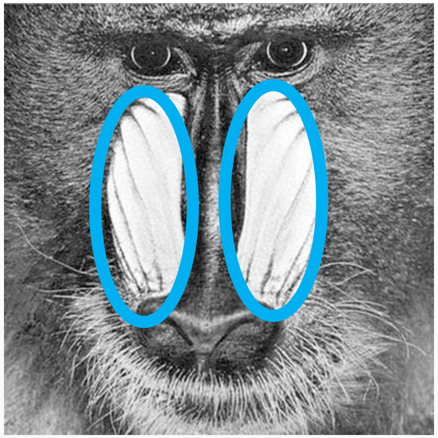
RGB color image



Red channel



Green channel



Blue channel

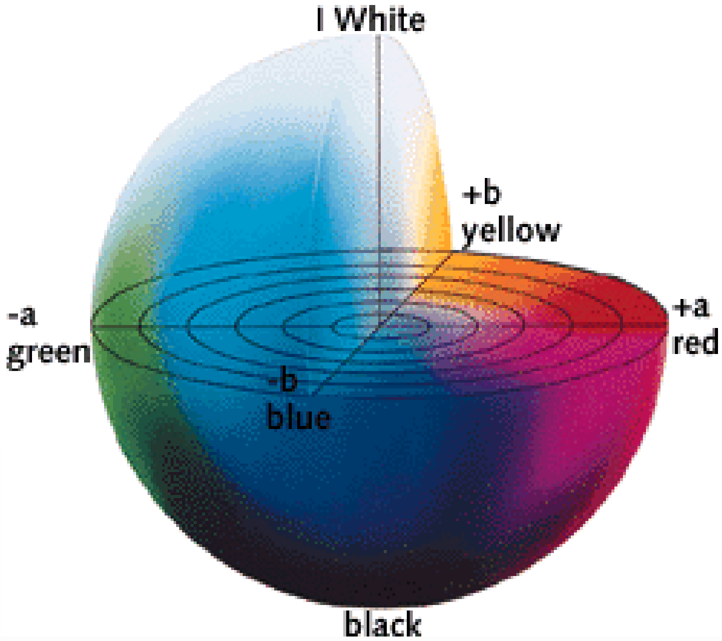
Color Spaces

- **CMY**:
  - used in printing
  - Based on the subtractive color model: describes what kind of inks need to be applied, so the reflected light produces the given color.
- **CMYK**:
  - The black produced by the mixture of CMY is not really black in practice
  - Black ink is added as 4th component.
- **CIE**:
  - the CIE color model is based on how humans perceive color
  - was developed to be completely independent of any device
  - (CIE stands for *Comission Internationale de l'Eclairage*)

Source: http://dba.med.sc.edu/price/irf/Adobe_tg/models/cie.html

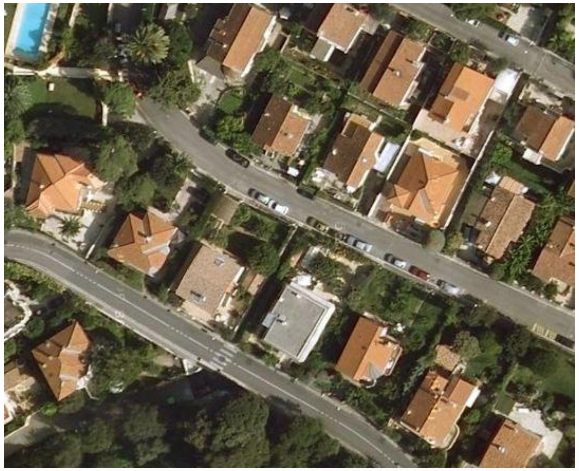
CIELAB, Lab, L*a*b

- One luminance channel (L)
  and two color channels (a and b).
- In this model, the color differences which you perceive correspond to Euclidian distances in CIELab.
- The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b). The brightness (L) increases from the bottom to the top of the three-dimensional model.

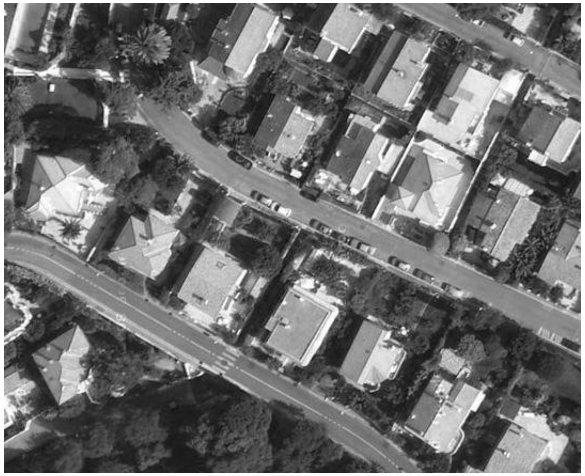


CIE L*a*b*

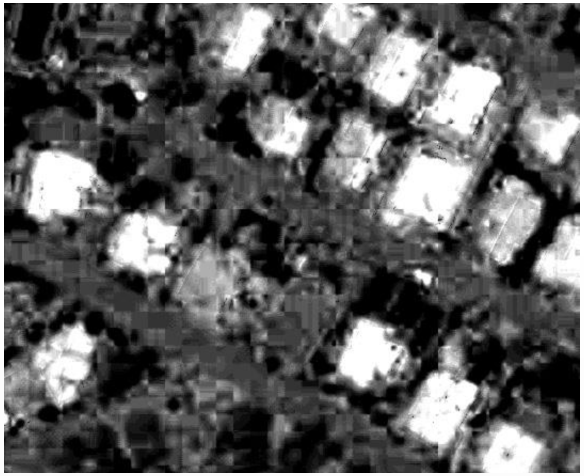
- Color filtering in CIE L*a*b*: where are the red roods?



RGB image



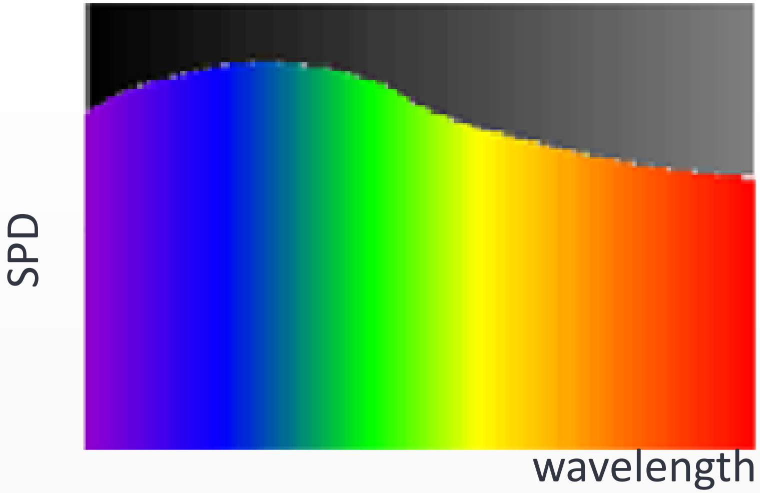
R channel of the RGB image



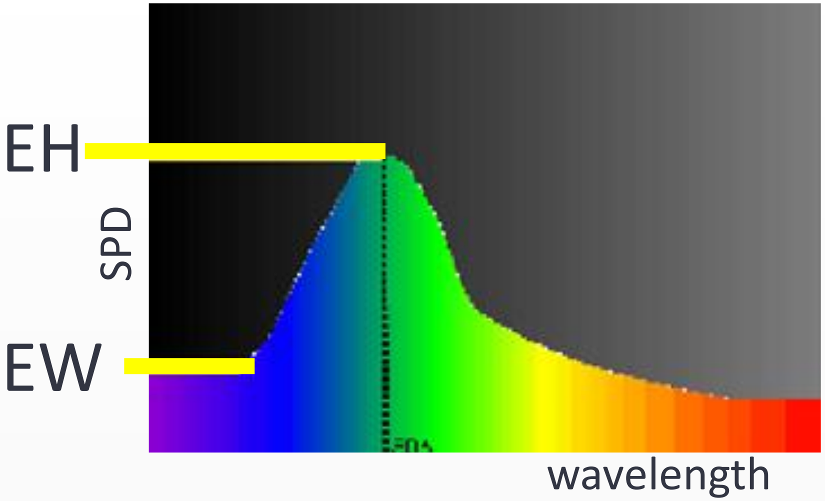
a* channel of the CIE L*a*b* image

HSI color space - fundamentals

- **Hue:** dominant wavelength of the Spectral Power Distribution: EH
- **Saturation:** relative purity or the amount of white light in the mixture EH-EW
- **Intensity:** indicates the dominant wavelength in the mixture of light waves: EW



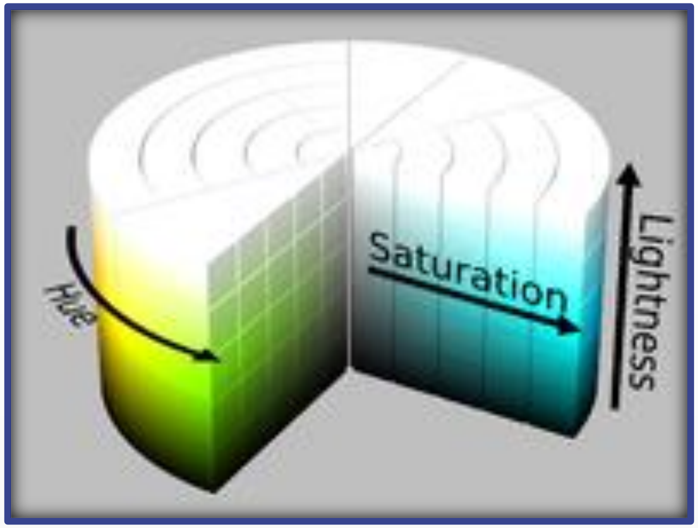
white light

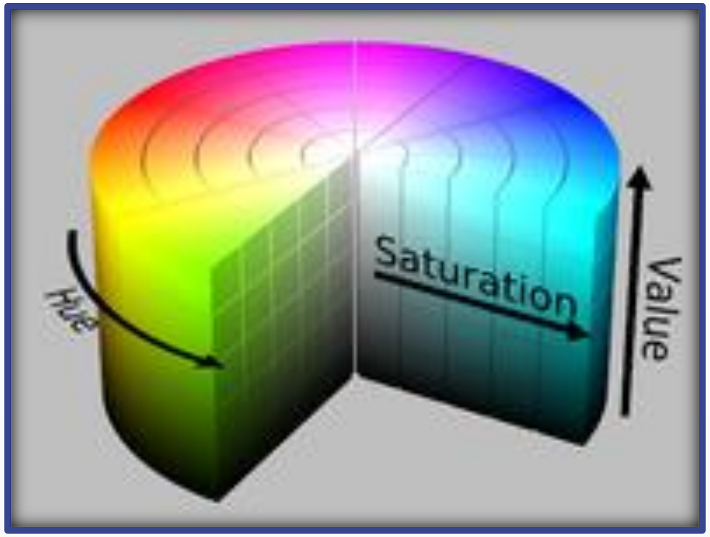


green light

Color Spaces

- **HSI/HSL, HSV**:
  - The components are more intuitive
    - **Hue**: the angle around the central vertical axis (defined in degrees)
    - **Saturation**: the distance from the central axis
    - **Intensity/Lightness** or **Value**: the height





Source: http://en.wikipedia.org/wiki/HSV_color_space

Sources

Fundamentals of Digital Image and Video Processing lectures by Aggelos K. Katsaggelos
Slides of Prof. Vladimir Székely (BME)

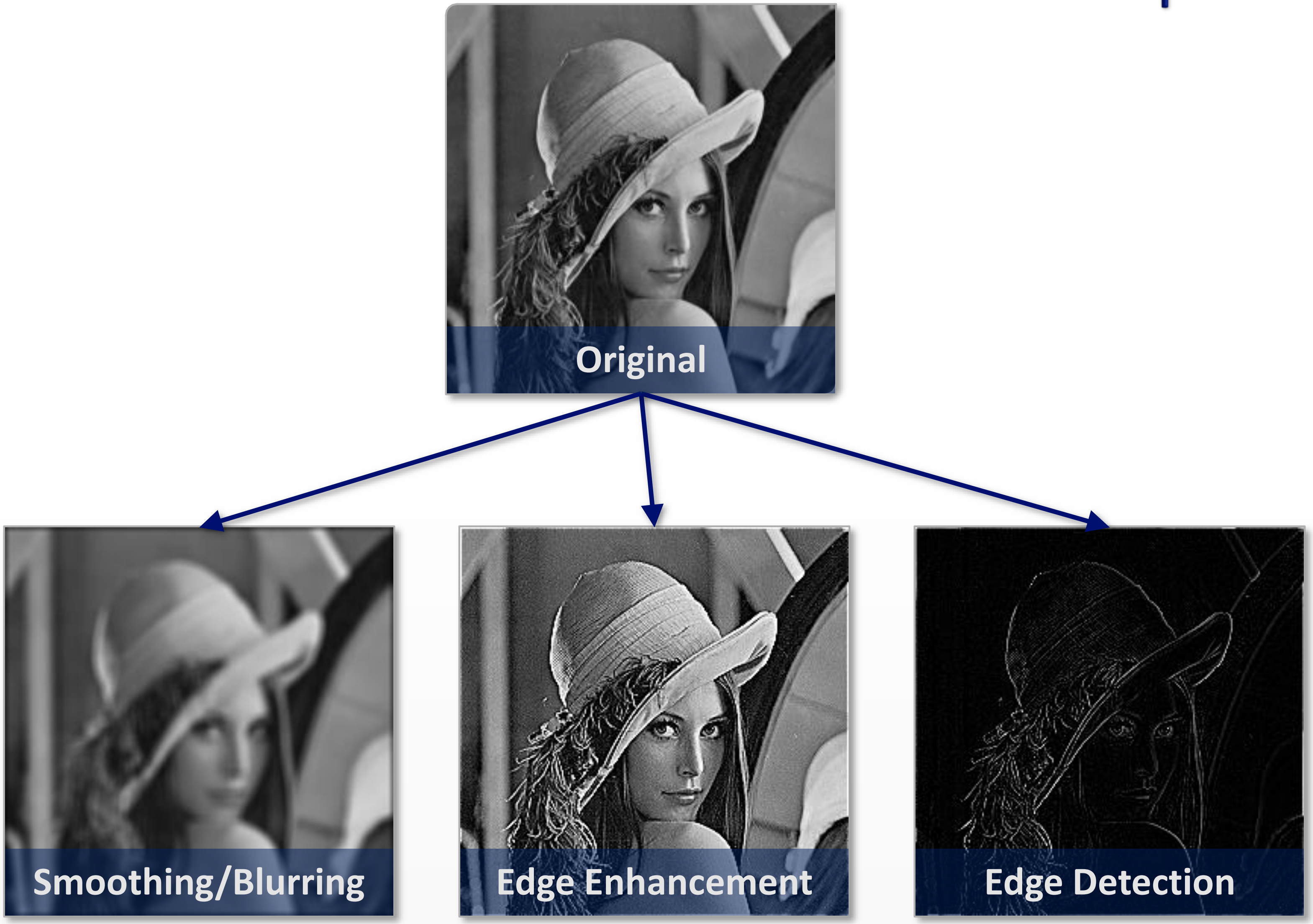
Basic Image Processing

PPKE-ITK

Lecture 2.

2D Convolution

Examples



Original

Smoothing/Blurring

Edge Enhancement

Edge Detection

Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - x and y are the pixel coordinates
  - f is a gray level from $[0,255]$

Image f

- We can define different transformations:
  - Intensity value inversion:

$$g(x, y) = 255 - f(x, y)$$

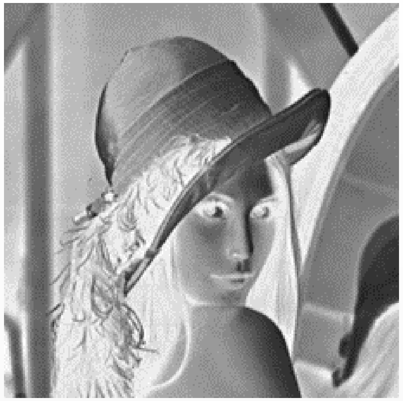


Image g

Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - x and y are the pixel coordinates
  - f is a gray level from $[0,255]$

Image f

- We can define different transformations:
  - Intensity shift with constant:
    $$g(x, y) = f(x, y) + 100$$

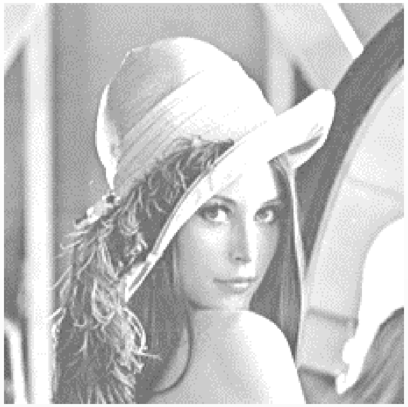


Image g

Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - x and y are the pixel coordinates
  - f is a gray level from $[0,255]$

Image f

- We can define different transforms:
  - Weighting :

$$g(x, y) = f(x, y) \cdot w(x, y)$$

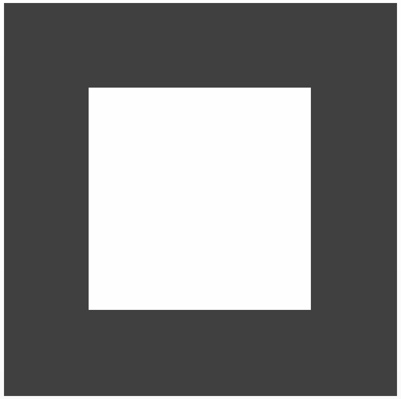


Image w

$w(x, y) \in [0.5,2]$

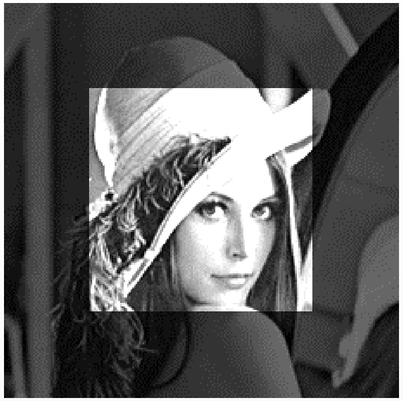


Image g

Mathematical background

- We look at the image as a 2D function: $f(x, y)$
  - x and y are the pixel coordinates
  - f is a gray level from $[0,255]$

Image f

- We can define different transformations:
  - Average on an *N* neighborhood :

$$f(x, y) = \text{average } N\big(f(x, y)\big)$$

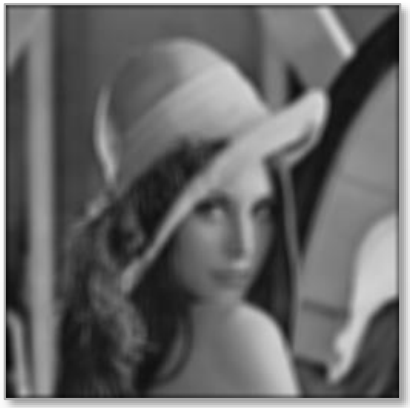


Image g

Mathematical background

- We look at the image as a 2D function: $f(x, y)$

$x(n_1, n_2)$

- We can define different transformations:
  - Intensity value inversion: $g(x, y) = 255 - f(x, y)$
  - Intensity shift with constant: $g(x, y) = f(x, y) + 100$
  - Weighting: $g(x, y) = f(x, y) \cdot w(x, y)$
  - Average on an *N* neighborhood: $g(x, y) = \text{average}\, N\big(f(x, y)\big)$

- *In this lecture*, there are two important properties of the transformations we want to use on images: ***linearity*** and ***shift invariance***

Mathematical background

- Linearity:

$$T[f_1(x,y) + f_2(x,y)] = T[f_1(x,y)] + T[f_2(x,y)]$$
$$T[\alpha \cdot f(x,y)] = \alpha \cdot T[f(x,y)]$$

  - e.g.: weighting is linear, intensity inversion is non-linear

- Spatial Invariance (SI): for any $[k,l]$ spatial shift vector,

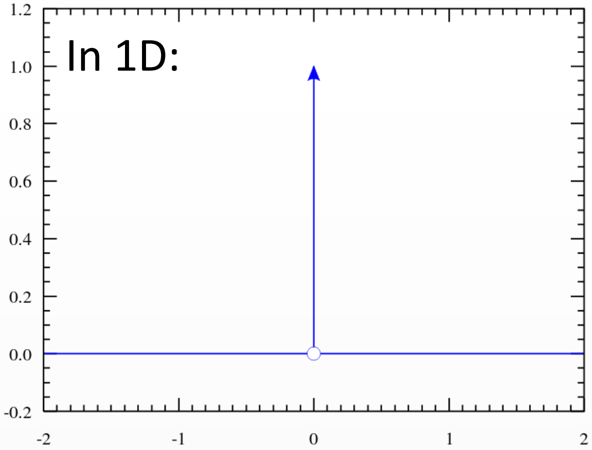
$$T[f(x,y)] = g(x,y)$$
$$T[f(x-k,y-l)] = g(x-k,y-l)$$

  - e.g.: weighting is not SI, intensity inversion is SI
  - e.g.: averaging on neighborhood is both linear and SI, we call it LSI

Unit Impulse Function

- 2D Unit Impulse function (Delta function) on $\mathbb{Z}$ as follows:

$$\delta(x,y)=\begin{cases}1 & \text{when } x=0 \text{ and } y=0\\ 0 & \text{otherwise}\end{cases}$$



- For any 2D function $f(x,y)$:

$$f(x,y)=\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}\delta(x-k,y-l)\cdot f(k,l)$$

Convolution

- ***Impulse response*** is the output of an LSI transformation if the input was the Delta function: $\delta(x,y) \rightarrow T \rightarrow h(x,y)$

  If *T* is an LSI system:

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

  Then we can define convolution as follows:

  $$g(x,y) = f(x,y) * h(x,y) =$$

  $$= h(x,y) * f(x,y) =$$

  $$= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k,l) \cdot h(x-k, y-l)$$

Derivation of Convolution

$$g(x,y)=T\left[f(x,y)\right]=$$

$$=T\left[\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}f(k,l)\delta(x-k,y-l)\right]=$$

Linearity

$$=\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}f(k,l)\cdot T\left[\delta(x-k,y-l)\right]=$$

$$=\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}f(k,l)\cdot h(x-k,y-l)$$

Spatial Invariance

The Properties of Convolution

- Commutative:

$$f * g = g * f$$

- Associative:

$$f * (g * h) = (f * g) * h$$

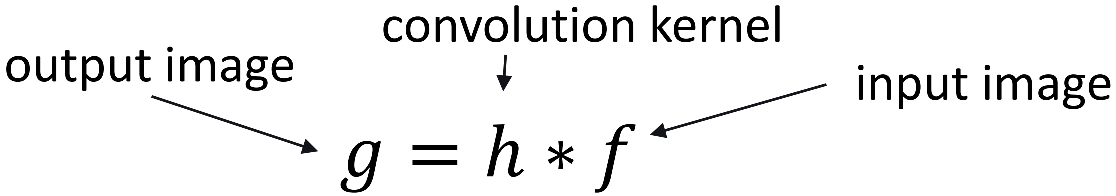
- Distributive:

$$f * (g + h) = f * g + f * h$$

- Associative with scalar multiplication:

$$\alpha(f * g) = (\alpha f) * g$$

2D convolution for image processing



convolution kernel

output image

input image

$$g = h * f$$

- In practice both the h kernel and the f image have finite size.
- Typically the size of h is much smaller than the image size (3×3, 5×5, 5×7 etc.)

2D convolution in Practice

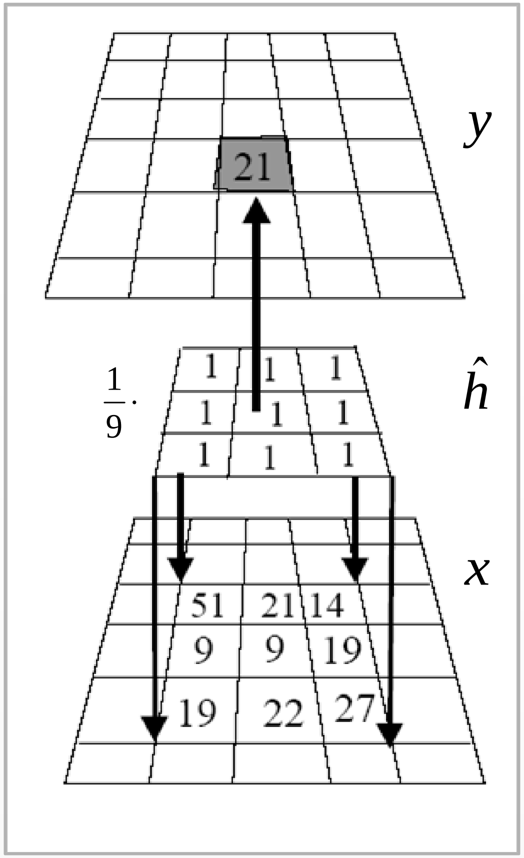
- Let h and $\hat{h}$ be $(2r_1 + 1) \times (2r_2 + 1)$ sized kernels where $\hat{h}$ is the rotated version of h with 180°

$$h = \begin{bmatrix} a_{-r_1,-r_2} & \cdots & a_{-r_1,r_2} \\ \vdots & \ddots & \vdots \\ a_{r_1,-r_2} & \cdots & a_{r_1,r_2} \end{bmatrix} \text{and } \hat{h} = \begin{bmatrix} a_{r_1,r_2} & \cdots & a_{r_1,-r_2} \\ \vdots & \ddots & \vdots \\ a_{-r_1,r_2} & \cdots & a_{-r_1,-r_2} \end{bmatrix}$$

$$g(x,y) = \sum_{l=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} f(k,l) \cdot h(x-k, y-l) =$$

$$= \sum_{k=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} h(k,l) \cdot f(x-k, y-l) =$$

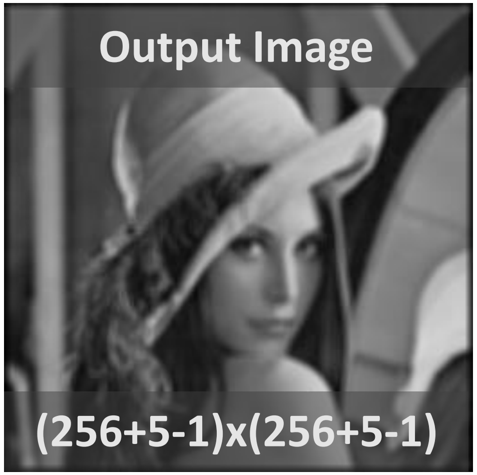
$$\boxed{= \sum_{k=-r_1}^{r_1} \sum_{l=-r_2}^{r_2} \hat{h}(k,l) \cdot f(x+k, y+l)}$$



Size of the Convolved Image

Convolutional Kernel

$$* \frac{1}{25}\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\end{bmatrix} =$$



5x5

In general:

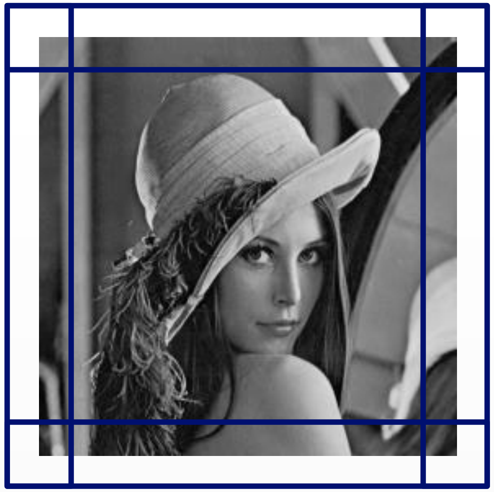
Size of the input image: $A \times B$

Size of the kernel: $C \times D$

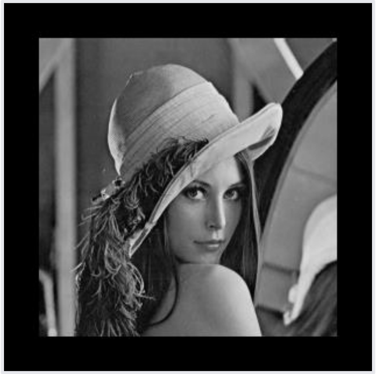
Size of the output image: $(A + C - 1) \times (B + D - 1)$

Boundary Effects

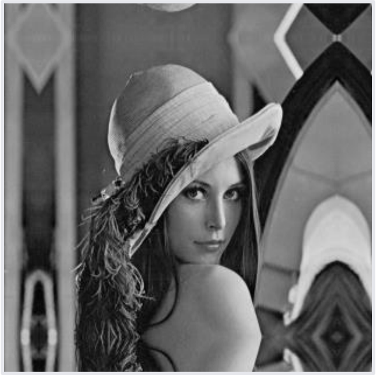
- What happens at the border of the image?



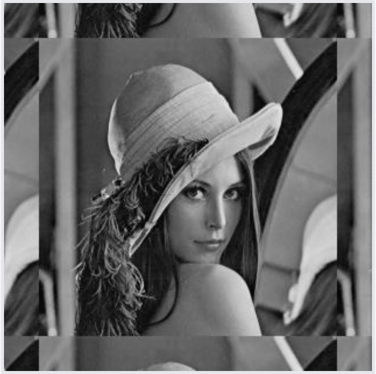
Original image with the problematic area



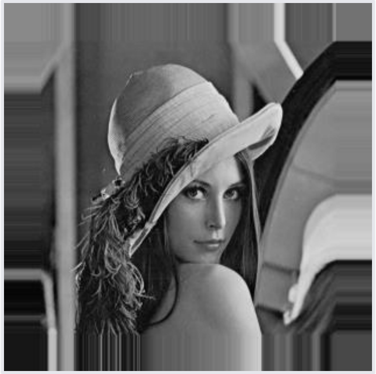
Zero padding



Mirroring



Circular padding



Repeating border

Applications

- Possible application of convolution:
  - Smoothing/Noise reduction
  - Edge detection
  - Edge enhancement
- Depending on the task the *sum of the elements of the kernel matrix* can be different:
  - 1: smoothing, edge enhancement

    E.g.: $\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$ $\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$
  - 0: edge detection

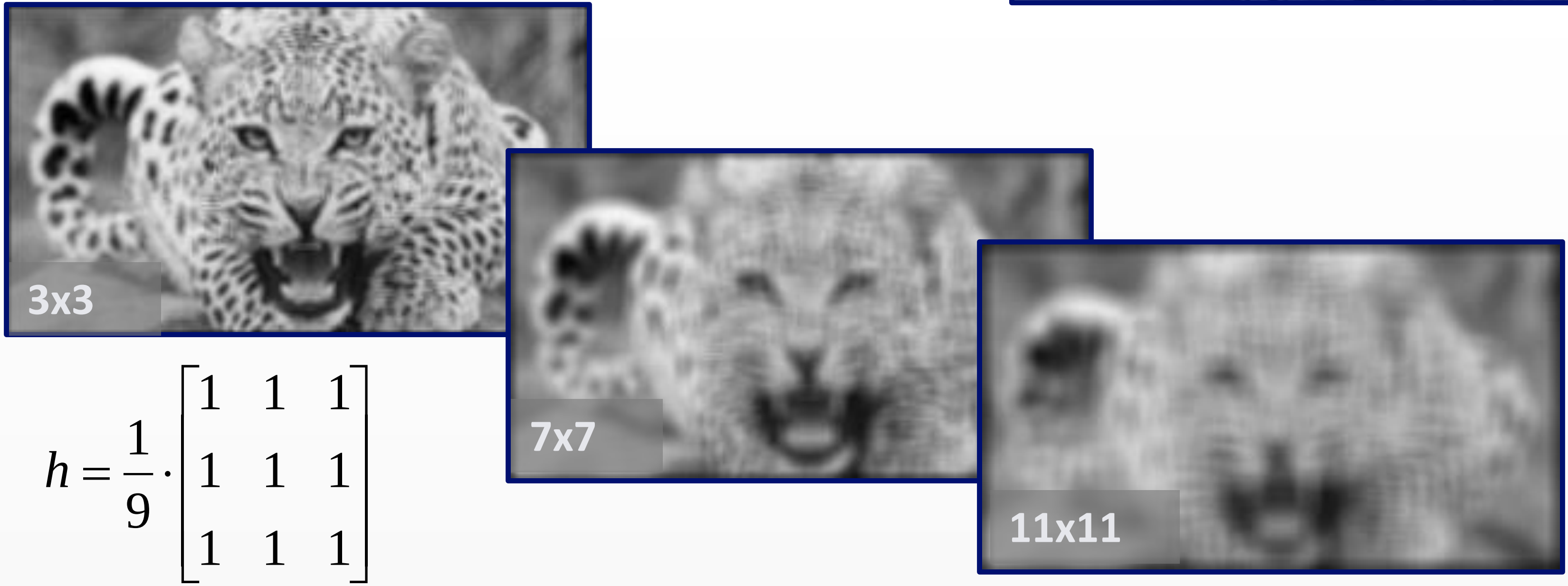
    E.g.: $\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}$ $\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix}$

Smoothing/Blurring

- Simple average:

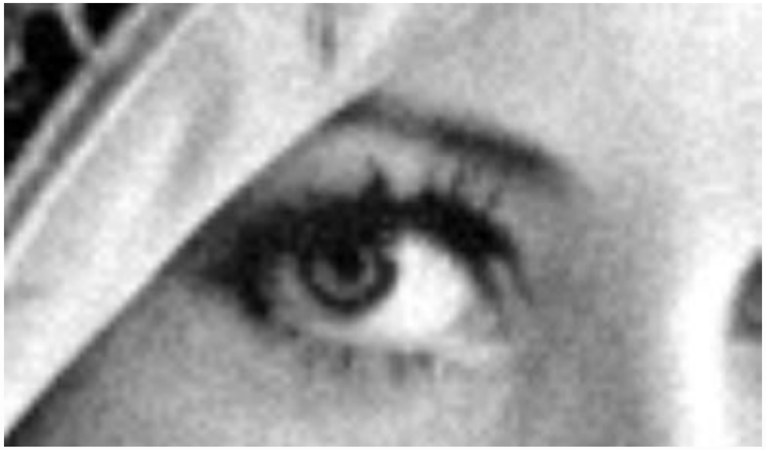
$$g(x,y)=\frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r}f(x+i,y+j)$$

Original

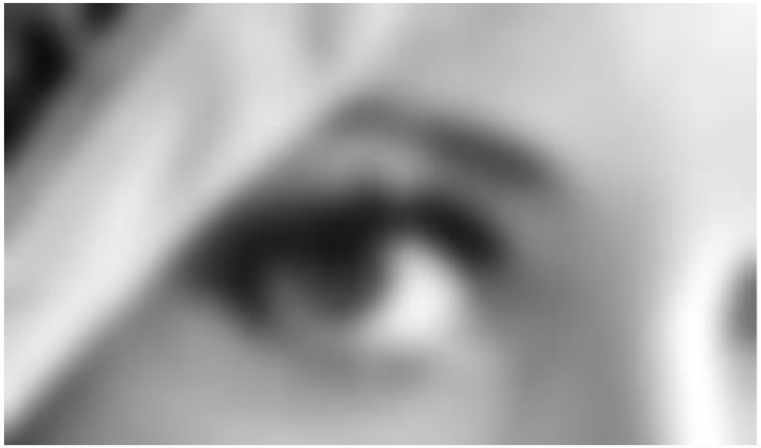


$$h=\frac{1}{9}\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Bluring for noise filtering



Noisy image



Result of bluring

Computational requirements

- For k_s kernel size and P image size (area, measured in pixels) approximately $\sim k_s P$ operations are needed.
- For large kernel size the execution may be slow

Decreasing the computational need for a simple (averaging) blur operation

- Integral image: $f \rightarrow I_f$ auxilliary representation

$$I_f(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} f(i, j)$$

- E.g. $I_f(3{,}3)$ = sum of the values of pixels 🟩 = 11

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

f

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Calculation of I_f with dynamic programing in ~P time:

- Auxiliary-auxiliary image: $t(x,y) = \sum_{j=1}^{y} f(x,j)$
  - Calculation of image t:

$$t(x,1) \coloneqq f(x,1), x = 1 \dots w; \quad t(x,y) = t(x,y-1) + f(x,y)$$

  - Calculation of I_f using image t:

$$I_f(1,y) \coloneqq t(1,y), y = 1 \dots h; \quad I_f(x,y) = I_f(x-1,y) + t(x,y)$$

4+7=11

f :

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

t :

1	0	2	1
3	0	3	1
6	1	4	1
7	1	5	5

I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Utilization of the integral image

- Sum of pixel values in an **arbitrary sized** sub-rectangle can be calculated by applying **3 additive operations** using the integral image:

$$\sum_{i=a}^{c}\sum_{j=b}^{d} f(i,j) = I_f(c,d) - I_f(a-1,d) - I_f(c,b-1) + I_f(a-1,b-1)$$

- Example (a=1, b=1, c=2, d=2): 11-6-3+1=3

f

1	0	2	1
2	0	1	0
3	1	1	0
1	0	1	4

I_f

1	1	3	4
3	3	6	7
6	7	11	12
7	8	13	18

Using integral image for quick blurring (simple averaging kernel)

$$\tilde{f}(x,y)=\frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r} f(x+i,y+j)$$

$(2r+1)^2$ addition
+ 1 division operations

Example: r=5 → For the whole image ~122P operations

$$\tilde{f}(x,y)=\frac{1}{(2r+1)^2}(I_f(x+r,y+r)-I_f(x-r-1,y+r)-$$

$$-I_f(x+r,y-r-1)+I_f(x-r-1,x-r-1))$$

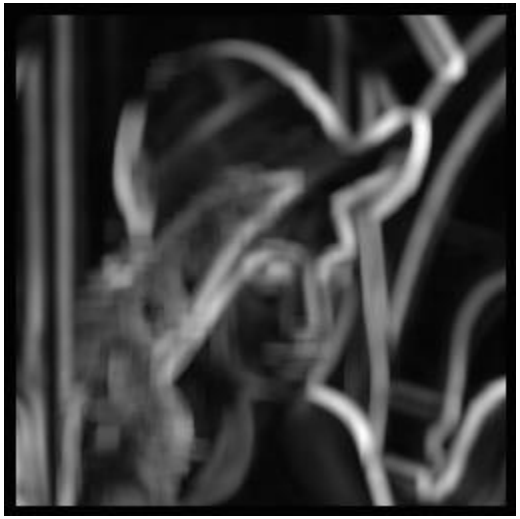
3 addition
+1 division

Example: r=5 → For the whole image ~ 2P+4P=6P operations

calc. integral image   calc. bluring

Optional homework (a bit more than a convolution)

- Construct an efficient contrast calculating algorithm using the integral image! Contrast is calculated as the standard deviation of pixel values of the (2r+1)^2 size neighborhood of each pixel.



$$\sigma^2(x,y) = \frac{1}{(2r+1)^2} \sum_{i=-r}^{r} \sum_{j=-r}^{r} \left[f(x+i,y+j) - \tilde{f}(x,y) \right]^2$$

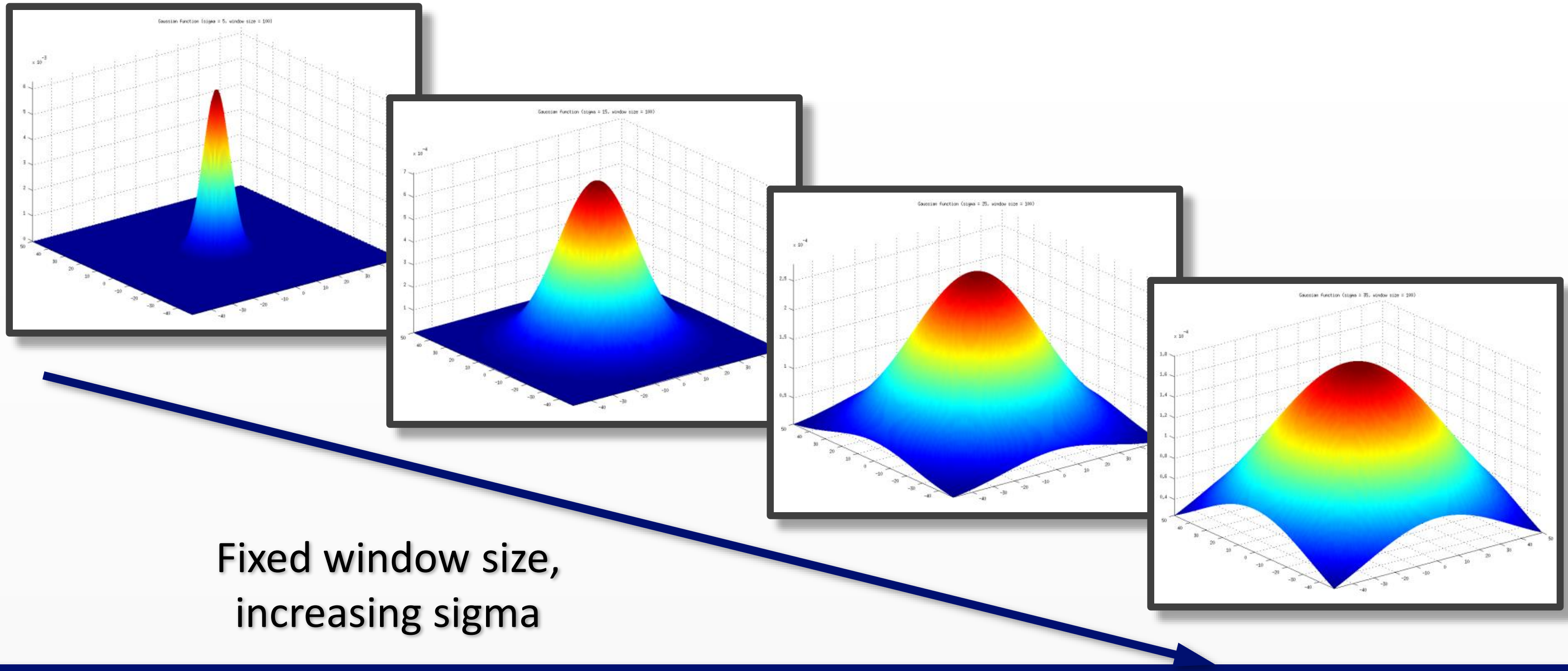
where: $$\tilde{f}(x,y) = \frac{1}{(2r+1)^2} \sum_{i=-r}^{r} \sum_{j=-r}^{r} f(x+i,y+j)$$

Hint:

$$\sigma^2(x,y) = \left\{ \frac{1}{(2r+1)^2} \sum_{i=-r}^{r} \sum_{j=-r}^{r} \left[f(x+i,y+j) \right]^2 \right\} - \left[\tilde{f}(x,y) \right]^2$$

Smoothing/Blurring

- Gaussian blur:
  - Weights are defined by a 2D Gaussian function
  - 2 parameters: **size of the window** and the **standard deviation** of the Gaussian

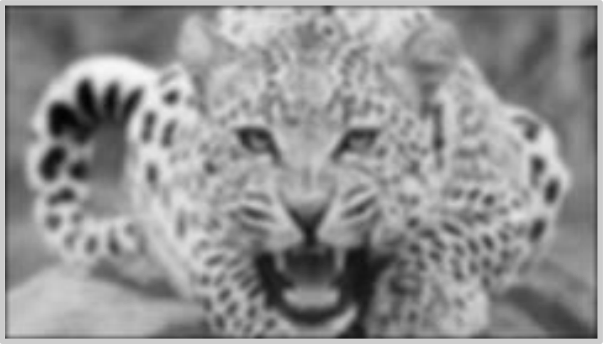


Fixed window size, increasing sigma

Smoothing/Blurring

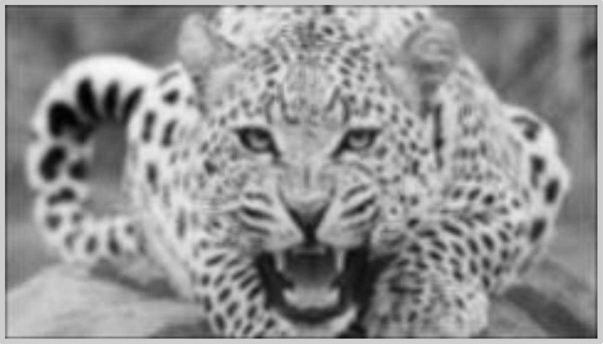
- Gaussian blur:
  - Weights are defined by a 2D Gaussian function
  - 2 parameters: window size and the width of the Gaussian
  - E.g. kernel size = 5x5; σ = 1.5;

$$\begin{bmatrix} 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0351 & 0.0683 & 0.0853 & 0.0683 & 0.0351 \\ 0.0281 & 0.0547 & 0.0683 & 0.0547 & 0.0281 \\ 0.0144 & 0.0281 & 0.0351 & 0.0281 & 0.0144 \end{bmatrix}$$



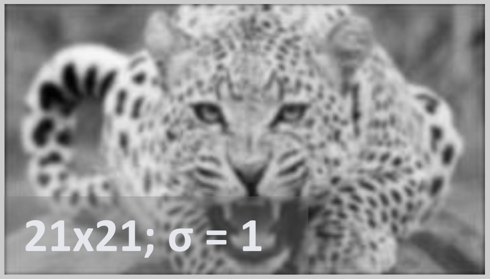
  - E.g. kernel size = 3x3; σ = 1.5;

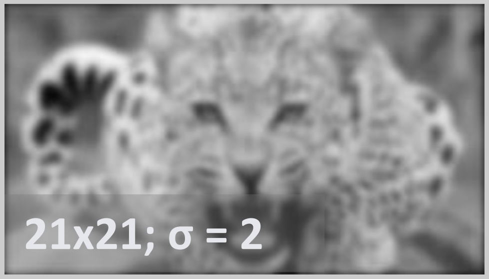
$$\begin{bmatrix} 0.0947 & 0.1183 & 0.0947 \\ 0.1183 & 0.1478 & 0.1183 \\ 0.0947 & 0.1183 & 0.0947 \end{bmatrix}$$

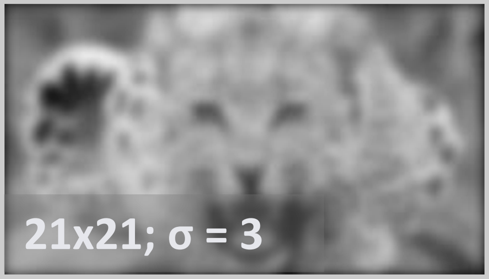


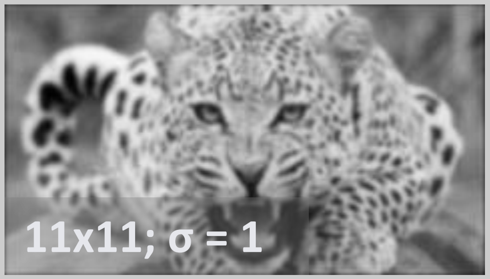
Smoothing/Blurring

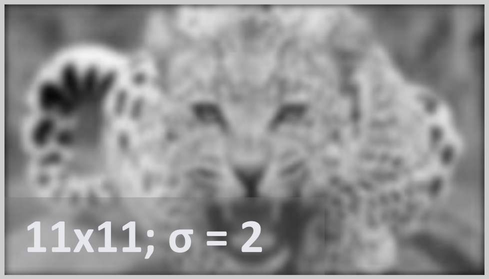
- Gaussian blur:

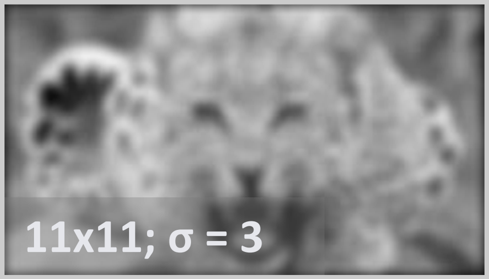

21x21; σ = 1

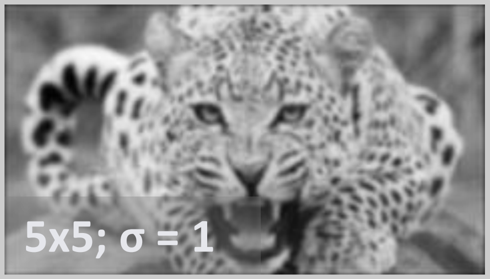

21x21; σ = 2

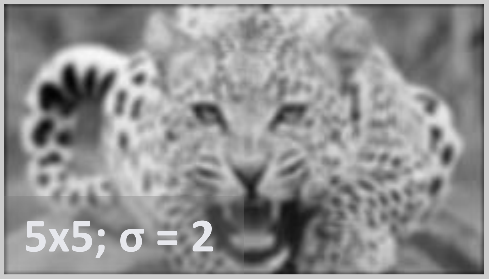

21x21; σ = 3

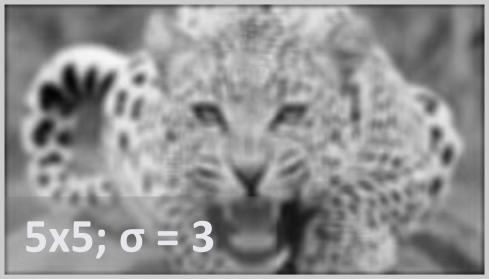

11x11; σ = 1


11x11; σ = 2

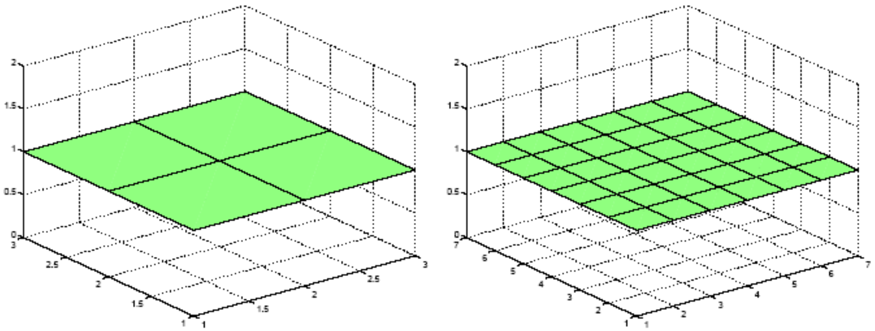

11x11; σ = 3


5x5; σ = 1

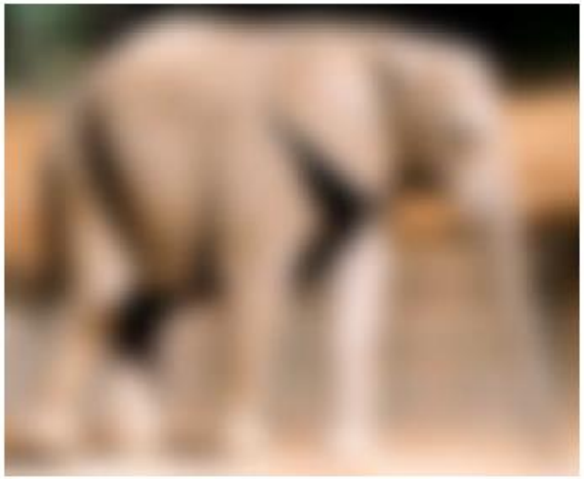

5x5; σ = 2


5x5; σ = 3

Convolution examples – averaging blur

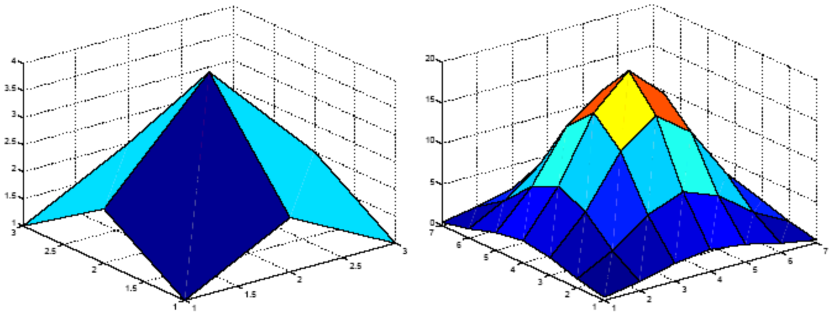


Input Image



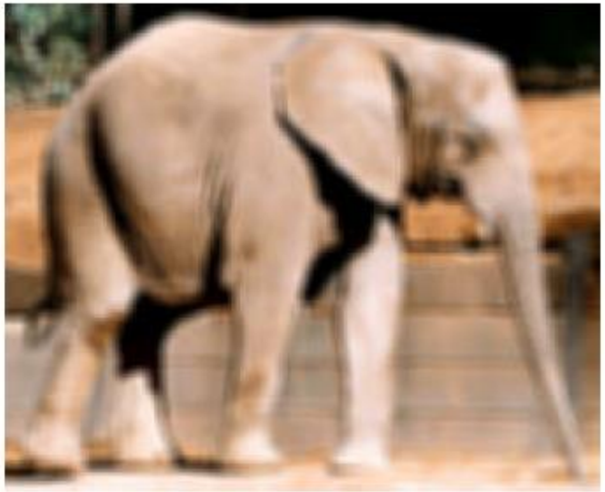
Average blur

Convolution examples– Gaussian blur



```
                [1   2   3   2   1]
                [2   7   11  7   2]
K=1/123  *      [3   11  17  11  3]
                [2   7   11  7   2]
                [1   2   3   2   1]
```

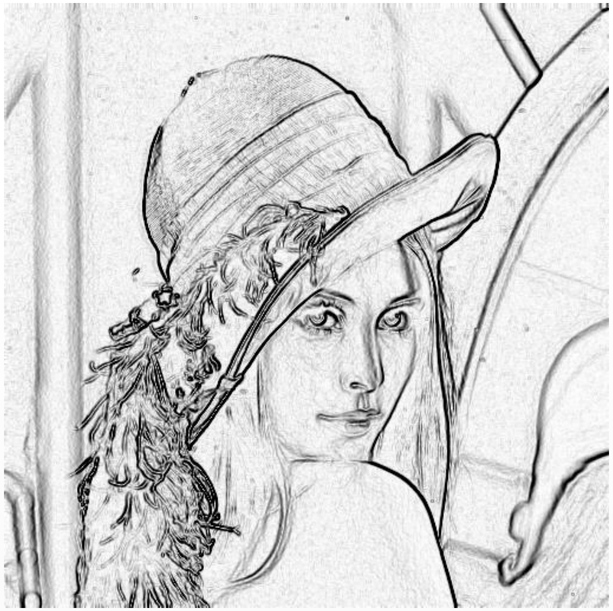
Input Image



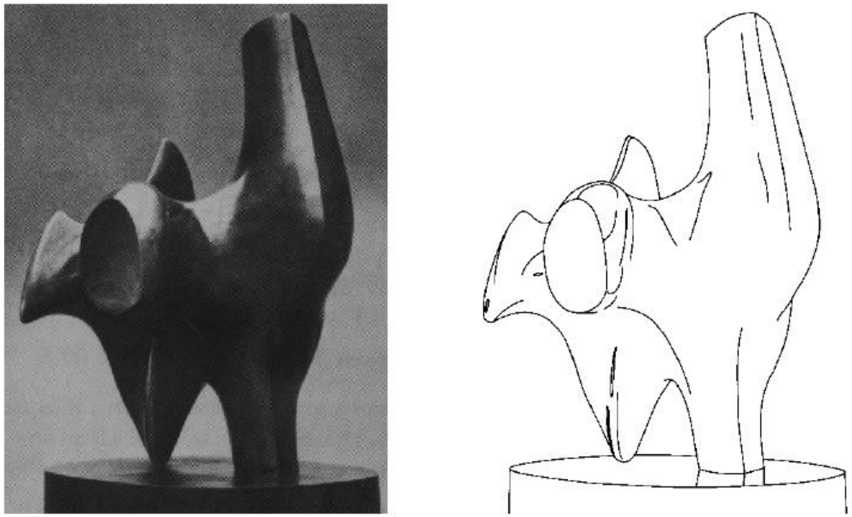
Gaussian blur

Edge detection

- Goal: extracting the object contours
- Edge points: brightness changes sharply



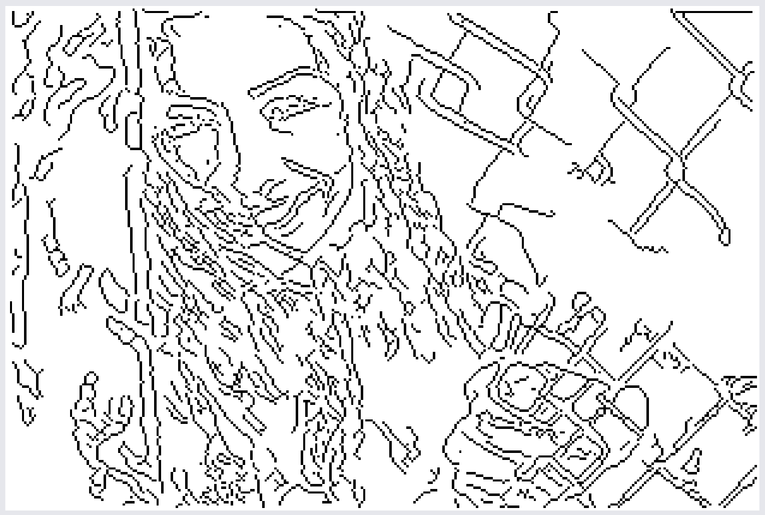
Goals of edge detection

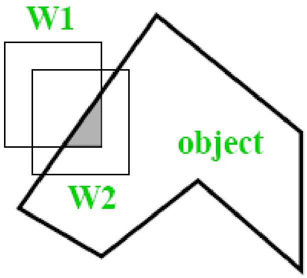


- Goal: extracting curves from 2D images
  - More compact content representation then pixel
  - Segmentation, recognition, scratch filtering

Goals of edge detection

- Extracting image information, structures
  - Corners, lines, borders
- Not always simple...



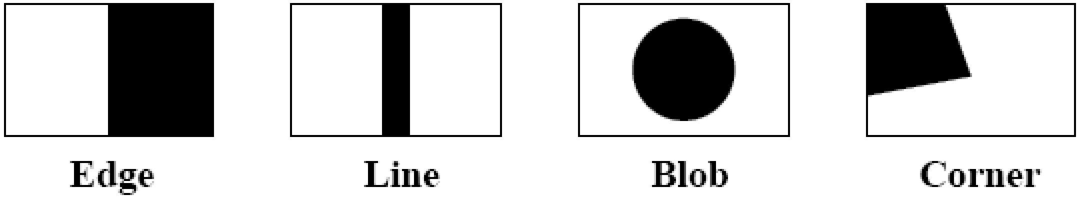


Edge detection

- Properties of a good edge filter:
  - (Near) zero output in homogeneous regions (constant intensity)
  - Good detection :
    - detects as many real edges as possible
    - does not create false edges (because of e.g. image noise)
  - Good localization: detected edges should be as close as possible to the real edges
  - Isotropic: filter response independent on edge directions
    - all edges are detected regardless of their direction

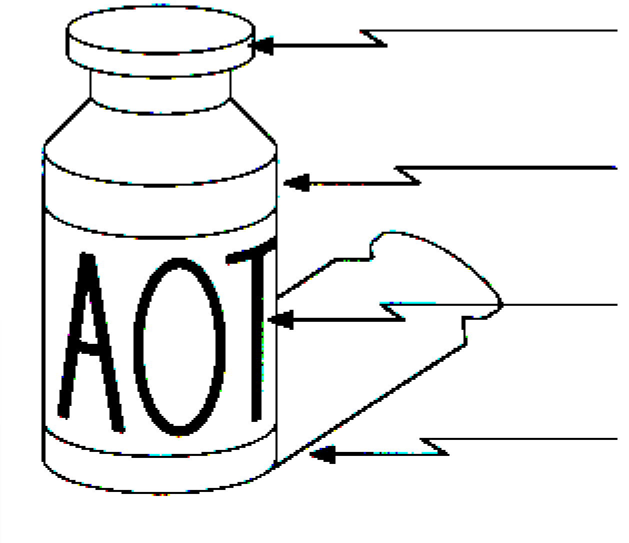
Basic structures

- **Edge**: sharp intensity change (steep or continuous)
- **Line**: thin, long region with approx. uniform width and intensity level
- **Blob**: closed region with homogeneous intensity
- **Corner**: breaking or direction change of a contour or edge



Origin and types of edges

- Various effects may cause edges



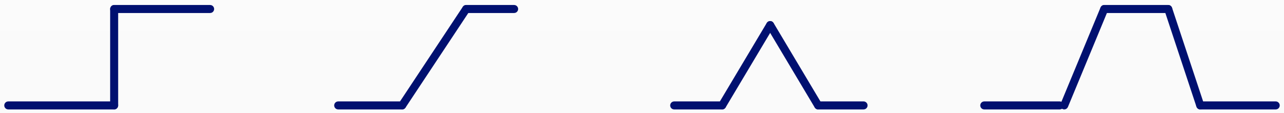
Sharp change in surface normals

Continuos change in surface depth

Change in surface color

Changes cased by illumination/shadows

- Basic edge types



step ramp roof line

Parameters of an edge

- **Edge normal**: vector, perpendicular to the edge, pointing toward the steepest intensity change
  - Alternatively: **edge direction** – a vector pointing towards the direction of the line
- **Position**: center point
- **Strength**: intensity ratio w.r.t. neighborhood

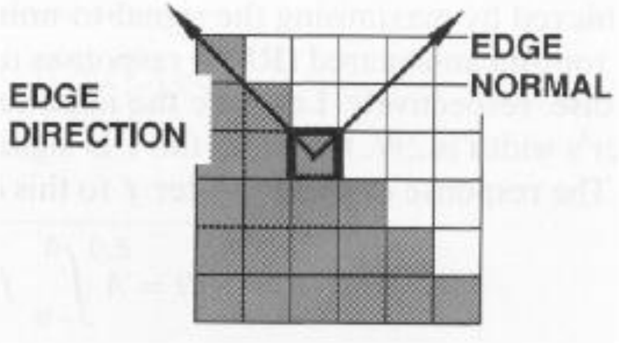
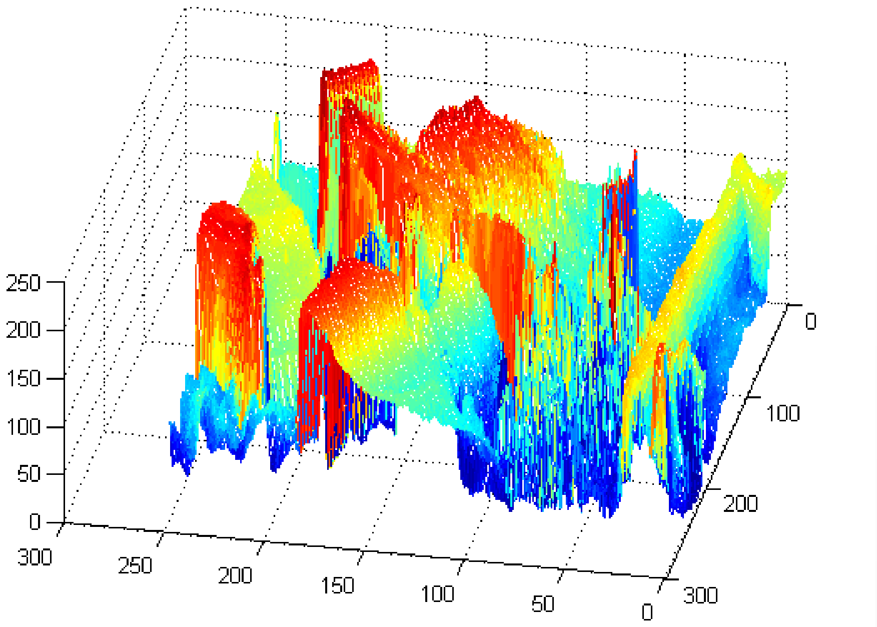


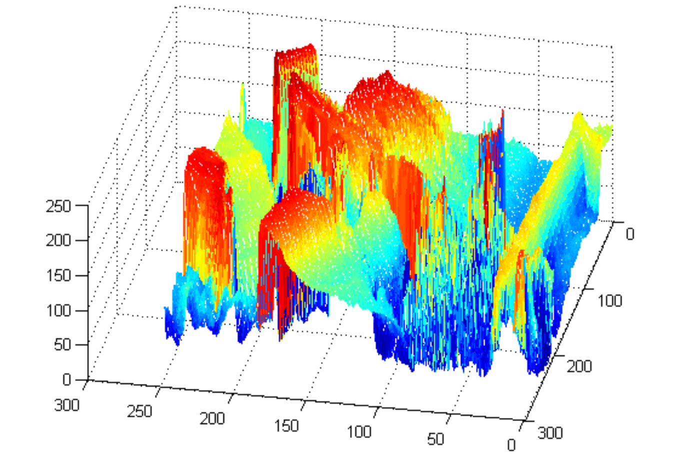
Image representation

- Image: gray value is function of the *x* and *y* coordinates (intensity function): $f(x,y)$



Edge Detection

- ***Edge:*** locations on the image where the ***intensity changes sharply*** (usually at the contour of objects)
- We are searching for places where the **gradient** of the 2D function (the image) is high.
- Main types of edge detection:
  - First order derivative
  - Second order derivative
  - Others:
    - Complex methods e.g. Canny method
    - Phase Congruancy



Edge Detection

- **Edge detection with first order derivative:**
  - Using the gradient vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^T$$

  - The approximation of the partial derivatives:

$$\frac{\partial f}{\partial x} = \lim \frac{f(x+dx, y) - f(x, y)}{dx}$$

$$\approx f(x+1, y) - f(x, y)$$

$$\frac{\partial f}{\partial y} = \lim \frac{f(x, y+dy) - f(x, y)}{dy}$$

$$\approx f(x, y+1) - f(x, y)$$

Since the smallest meaningful
discrete value is *dx*=1 and *dy* =1.

Edge Detection

- **Approximation of the x directional partial derivative:**
  - *For better localization*, use a symmetric formula around pixel (x, y)

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

    - Corresponding convolutional kernel:

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

  - *For noise reduction*, apply y directional smoothing (i.e. do not blur a sharp vertical edge)

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

x directional Prewitt operator: **vertical** edge detector

Edge Detection

- **Approximation of the y directional partial derivative:**
  - *For better localization*, use a symmetric formula around pixel (x, y)

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

  - Corresponding convolutional kernel: $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

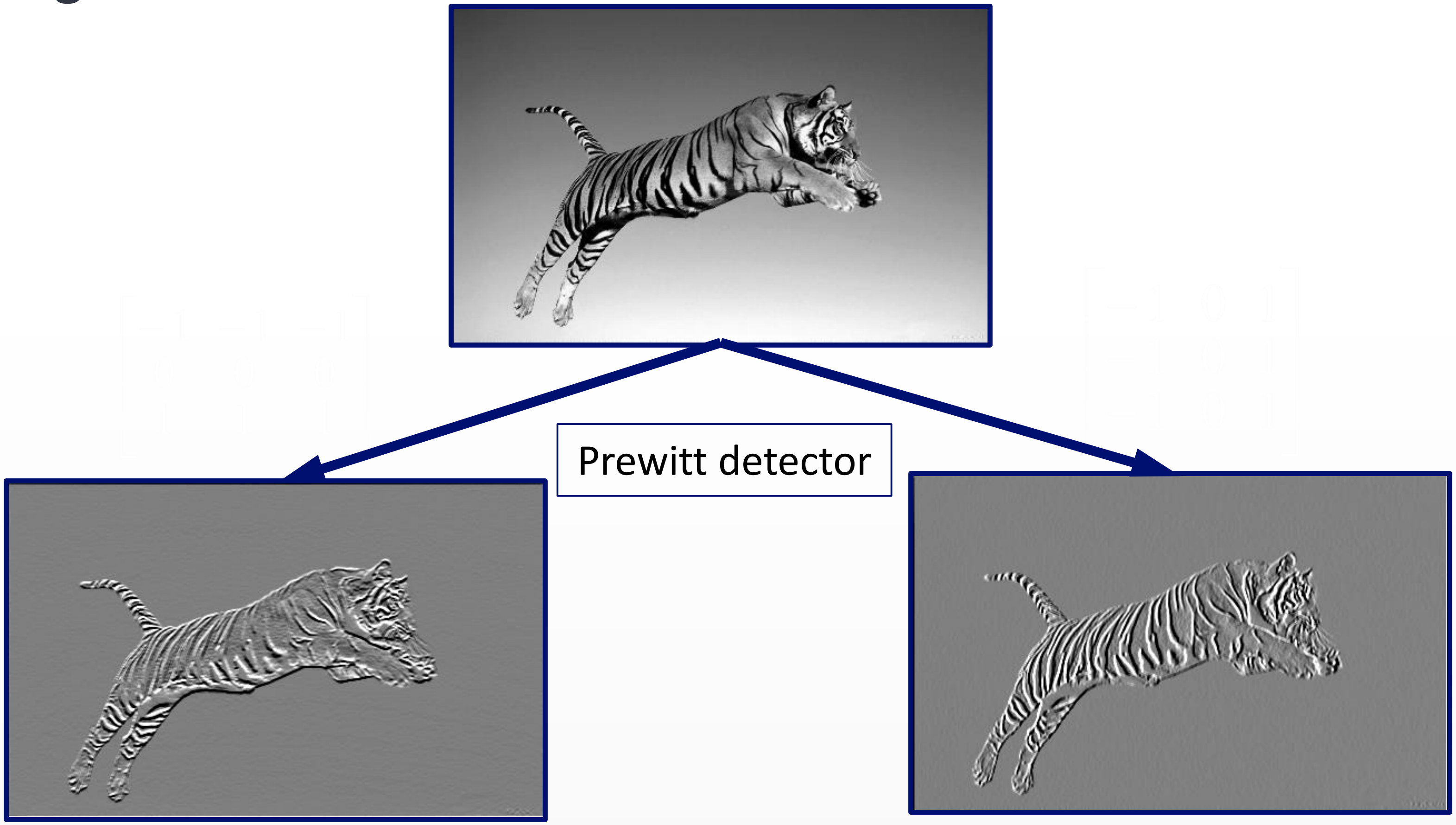
  - *For noise reduction*, apply x directional smoothing (i.e. do not blur a sharp horizontal edge)

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

y directional Prewitt operator:
horizontal edge detector

Edge Detection

- **Edge detection with first order derivative:**



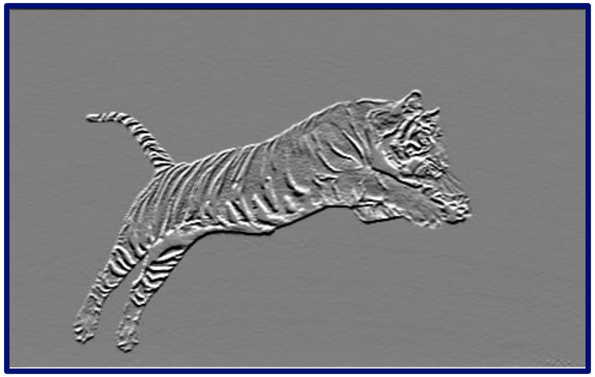
Prewitt detector

Horizontal gradient image

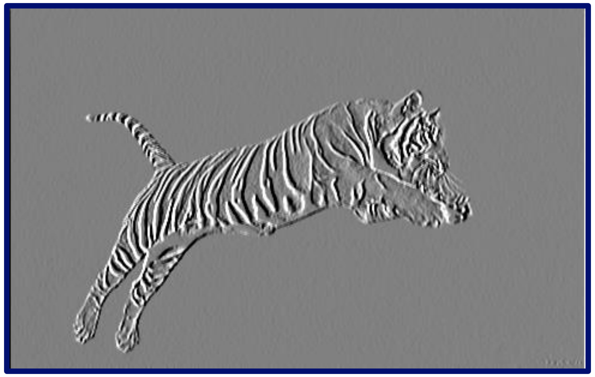
Vertical gradient image

Edge Detection

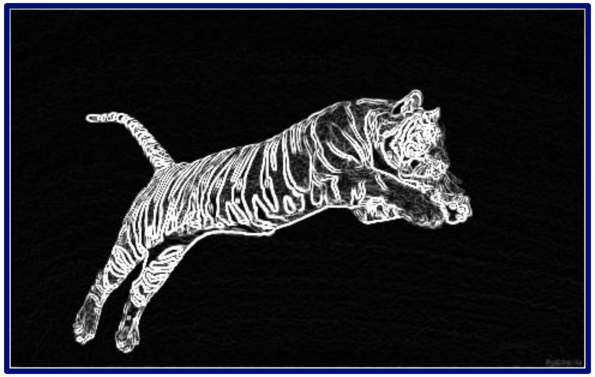
- **Edge detection with first order derivative:**



Horizontal gradient image



Vertical gradient image

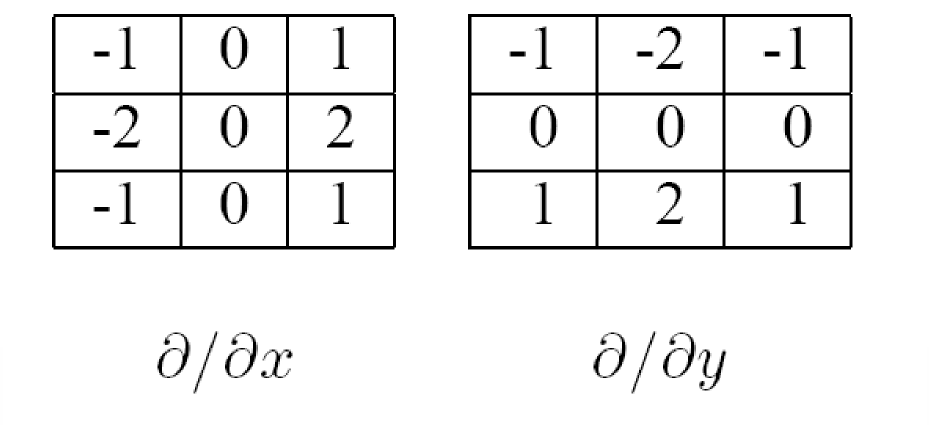


gradient image

$$\left\|\nabla f\right\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Other first-order methods

- Sobel operator



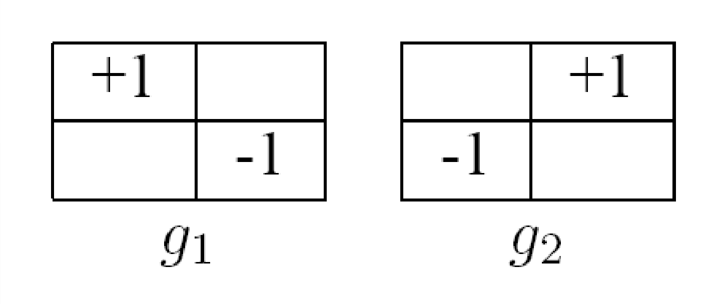
-1	0	1
-2	0	2
-1	0	1

$\partial/\partial x$

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

$\partial/\partial y$

- Roberts operator



+1	
	-1

g_1

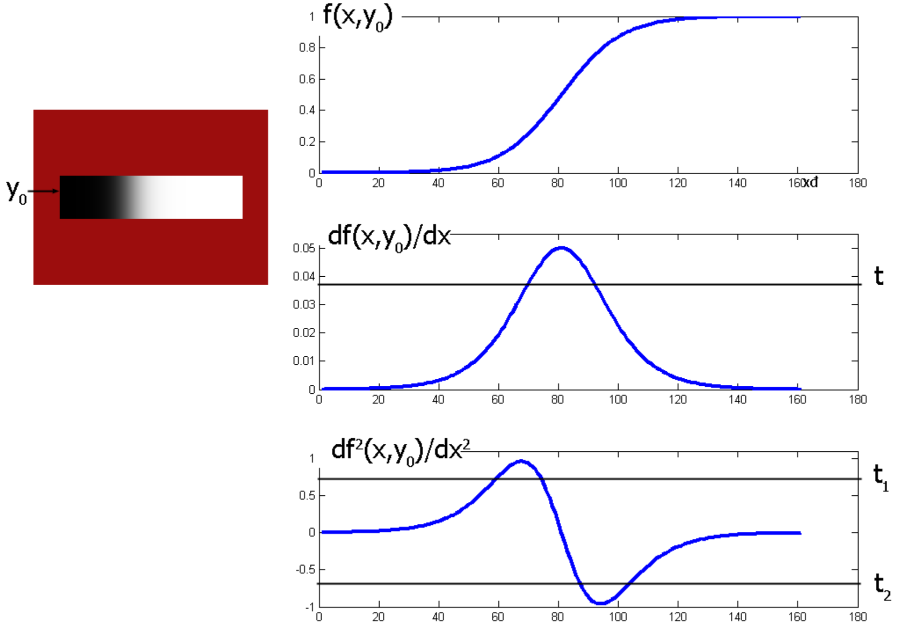
	+1
-1	

g_2

Emphasize edges with 45 degree slopes

Second order edge detection: motivation

Horizontal edge detection in the following image:

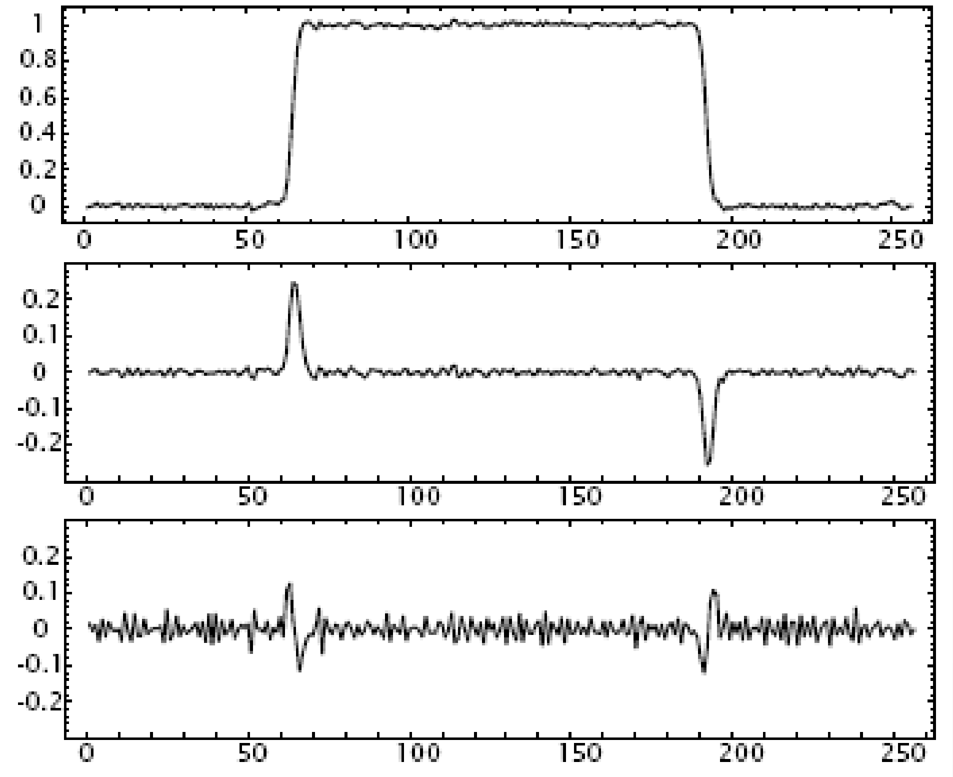


Top: intensity function along a selected horizontal line

Center: x directional first derivative

Bottom: x directional second derivative

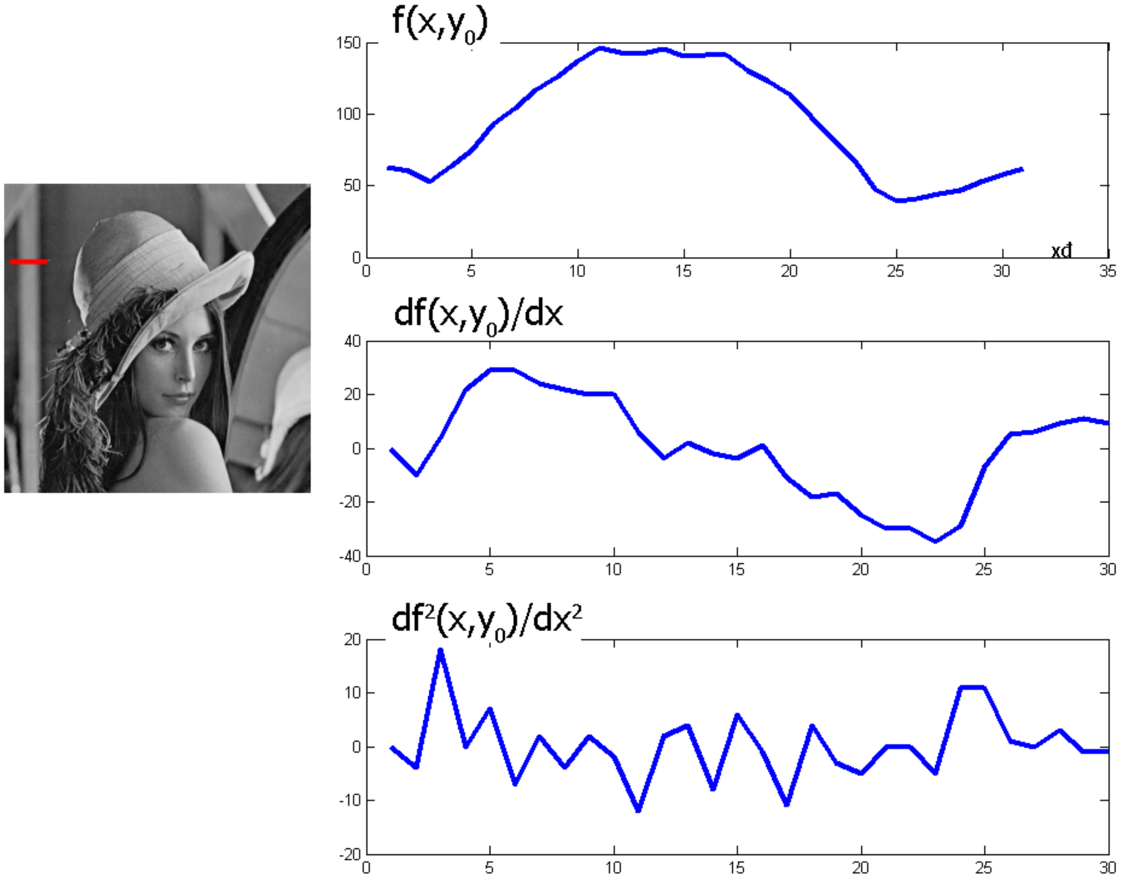
Second order case: instead of extreme values, search for zero crossing



Extreme values
(first order)

zero crossing
(second order)

Real photo: intensity profile below the red line segment



Edge detection with second order derivative

- Calculating the divergence of the gradient vector

$$\nabla^2 f = \left(\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial y}\right) \cdot \nabla f = \frac{\partial^2}{\partial x^2} f + \frac{\partial^2}{\partial y^2} f$$

- Approximation for x direction:

$$\frac{\partial^2 f(x,y)}{\partial x^2} \cong \frac{\frac{f(x+1,y)-f(x,y)}{v} - \frac{f(x,y)-f(x-1,y)}{v}}{v} =$$

$$= \frac{1}{v^2} \cdot [f(x+1,y) - 2f(x,y) + f(x-1,y)]$$

just a constant – v: distance of neighboring pixel centers

Edge detection with second order derivative

- Approximation of the second order derivatives for x and y directions

$$\frac{\partial^2 f(x,y)}{\partial x^2} \propto f(x+1,y) - 2f(x,y) + f(x-1,y)$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} \propto f(x,y+1) - 2f(x,y) + f(x,y-1)$$

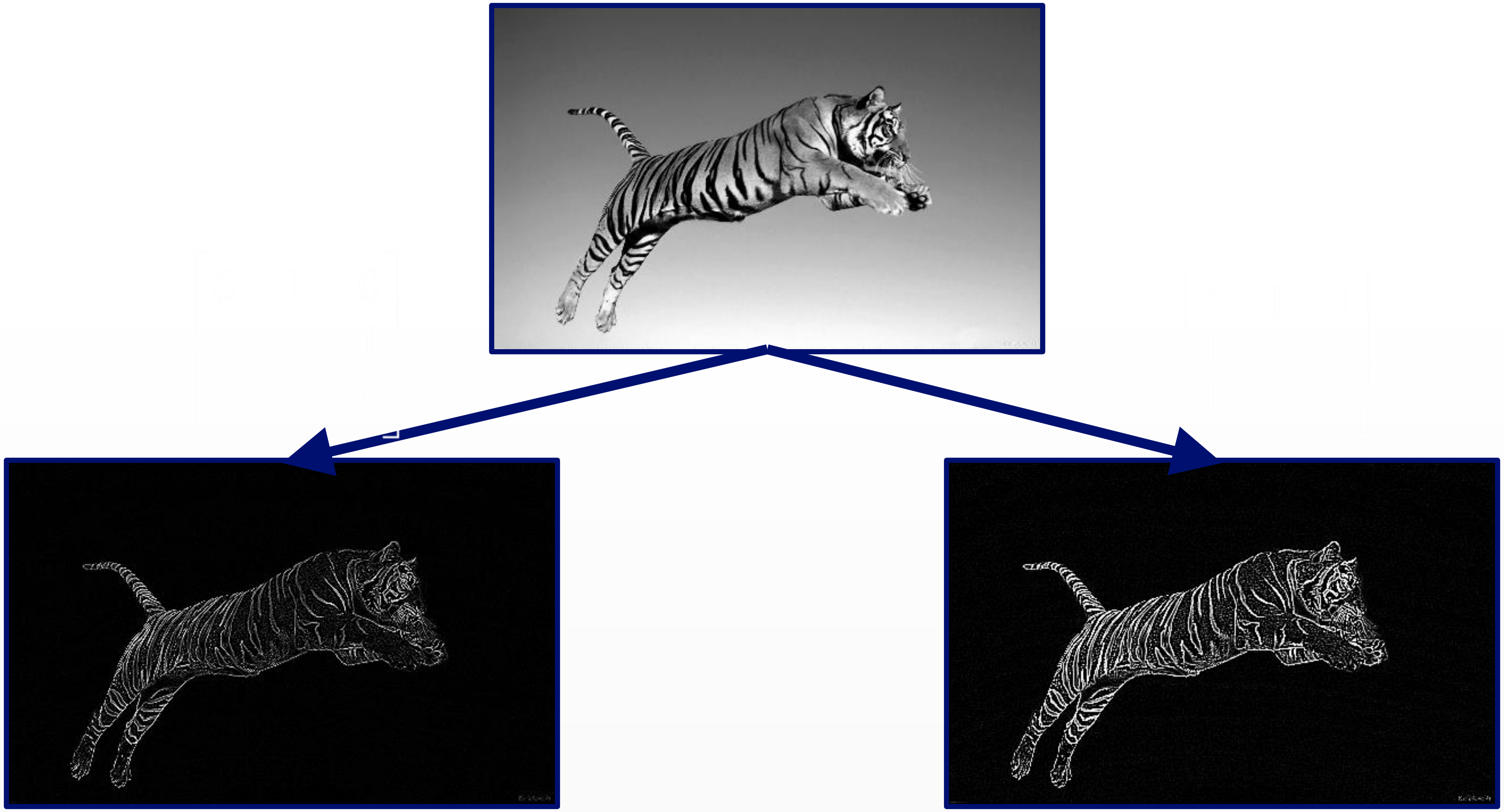
- Kernel for the second order gradient calculation with convolution:
  - Laplace operator:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \Rightarrow \qquad \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \boxed{\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}}$$

  - There are other variations. (e.g. Second order Prewitt)

Edge Detection

- **Edge detection with second order derivative:**

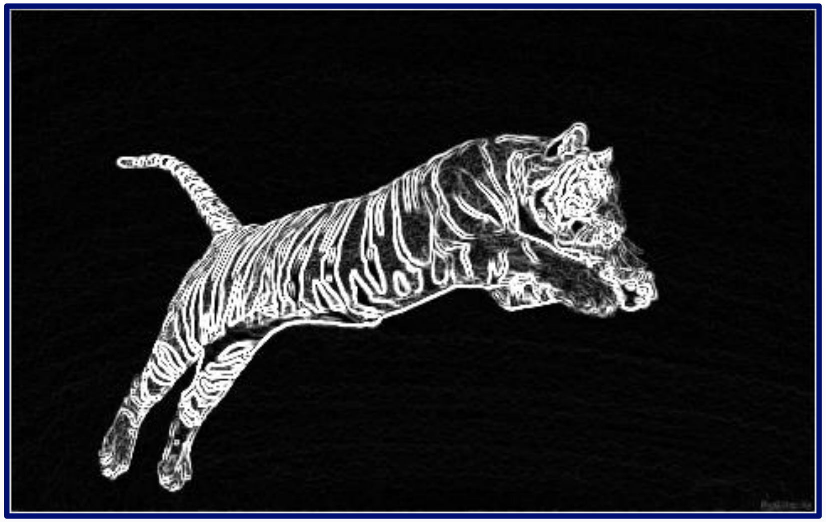


Laplace edge detector

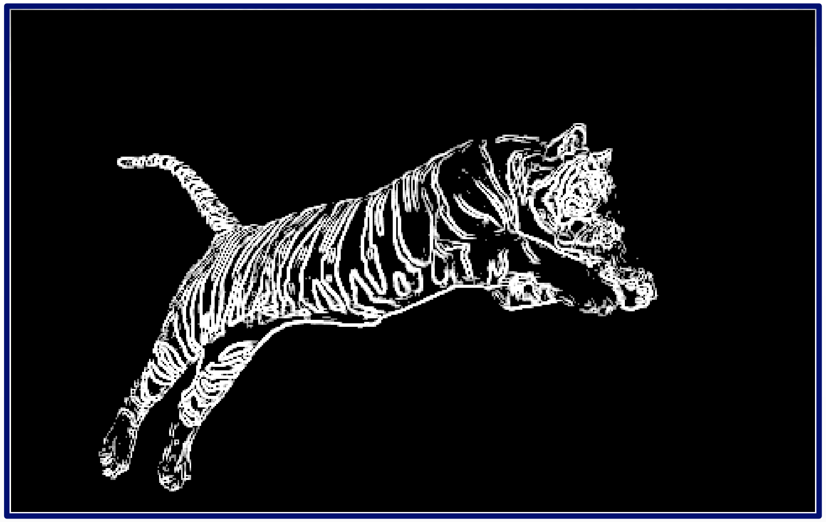
Prewitt 2nd order detector

Edge Detection

- **Thresholding**:
  - To eliminate weak edges, a threshold can be used on the gradient image:



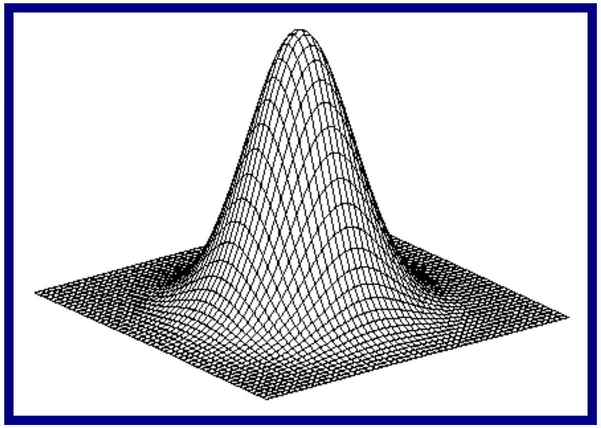
Prewitt first order gradient image



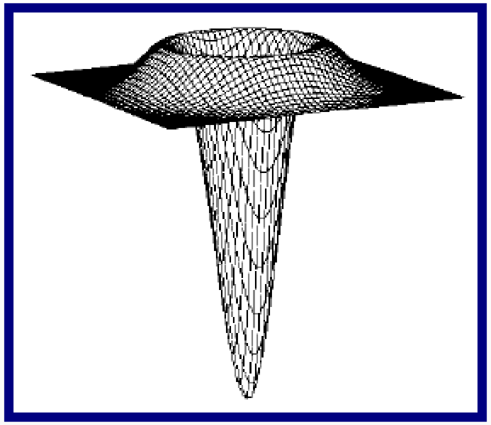
Prewitt first order gradient image with threshold = 120

Coming next week: Reducing the effect of noise on edge images

- Edge detection with noise reduction:
  - 1. step: Noise reduction by convolution with Gaussian filter
  - 2. step: Edge detection by convolution with Laplacian kernel
- Since convolution operation is associative we can convolve the Gaussian smoothing filter with the Laplacian filter first, and then convolve this hybrid filter (***Laplacian of Gaussian: LoG***) with the image.



Gaussian function



Laplacian of Gaussian

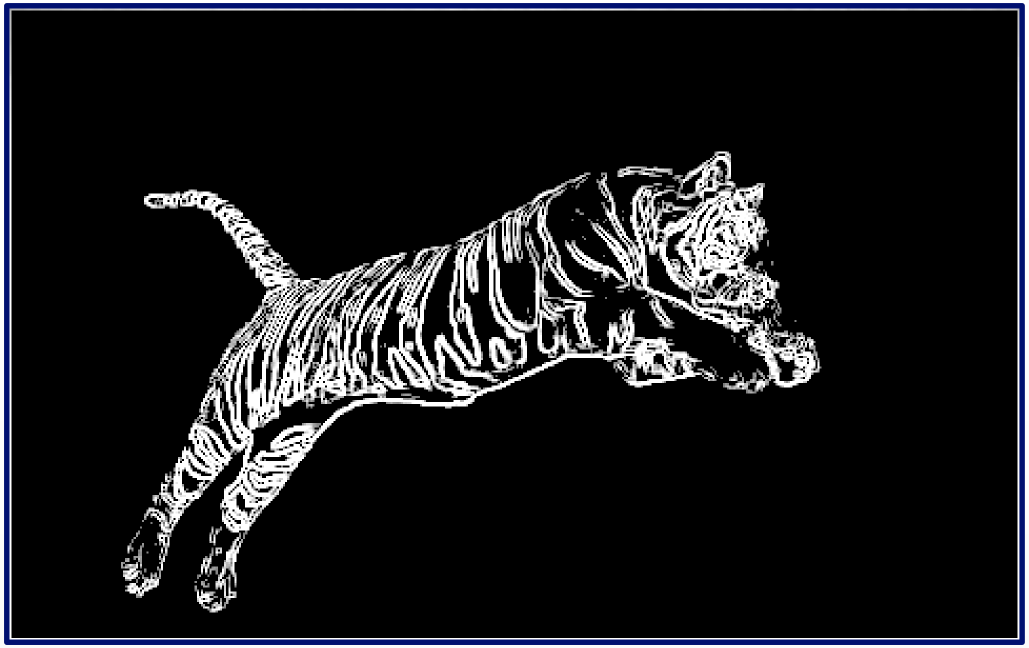
Basic Image Processing Algorithms

PPKE-ITK

Lecture 3.

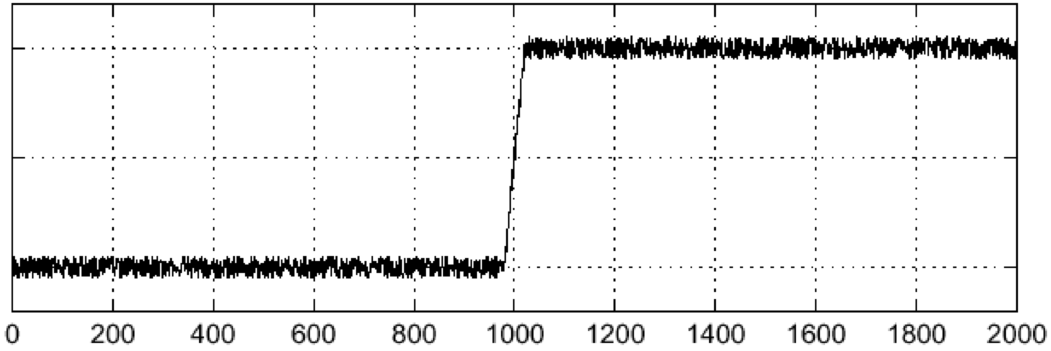
Recap: first/second order edge detection

- Noise filtering is required…

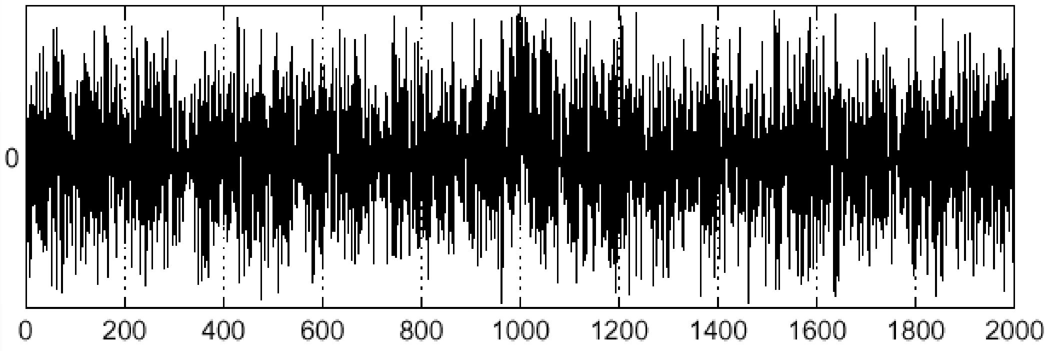


Noise filtering (1D demonstration)

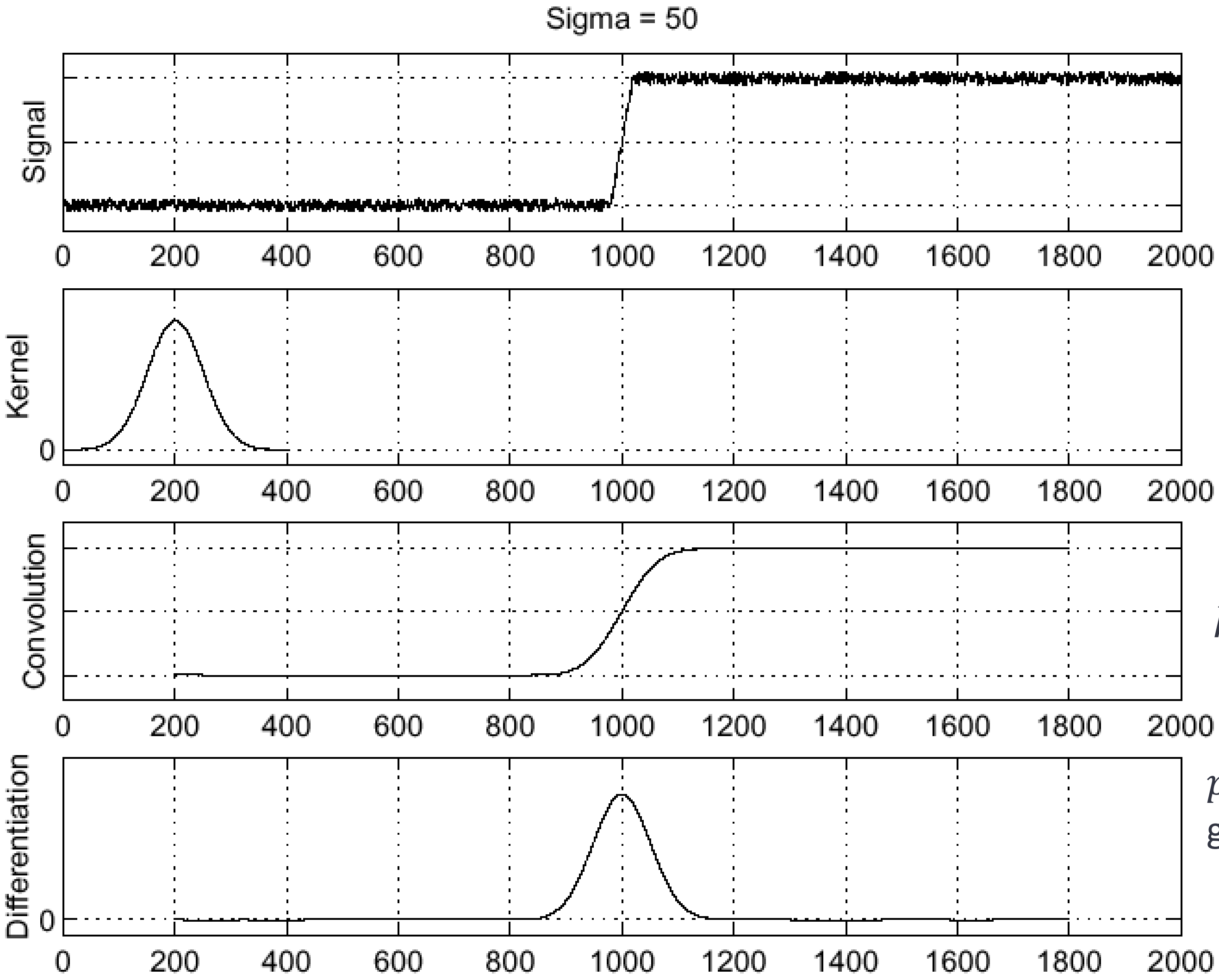
$f(x)$



$\frac{d}{dx}f(x)$



- Where is the edge?



f — f: original signal

h — h: Gaussian blur kernel

$h * f$ — $h * f$: filtered signal

$p * (h * f)$ — p: Prewitt (first order gradient kernel)

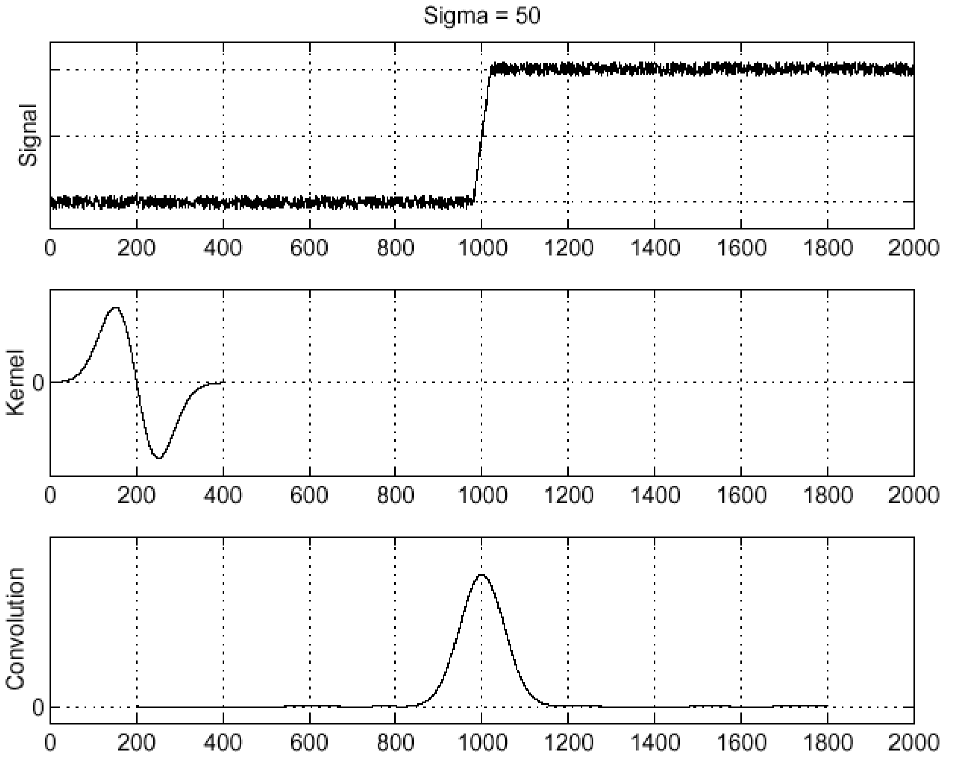
-1	0	1

Smoothing the signal with Gaussian kernel, followed by applying a first order Prewitt kernel

Associativity of convolution: $p * (h * f) = (p * h) * f$

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- No need for applying 2 convolutions, only one with the derivative of Gaussian operator (can also be approximated by a discrete kernel)



f

$\frac{\partial}{\partial x}h$ ≈ p*h

Derviative of the Gaussian kernel

$(\frac{\partial}{\partial x}h) \star f$

Second order case: Laplacian of Gaussian (LoG)

- Smoothing + Laplace = conv. with LoG operator

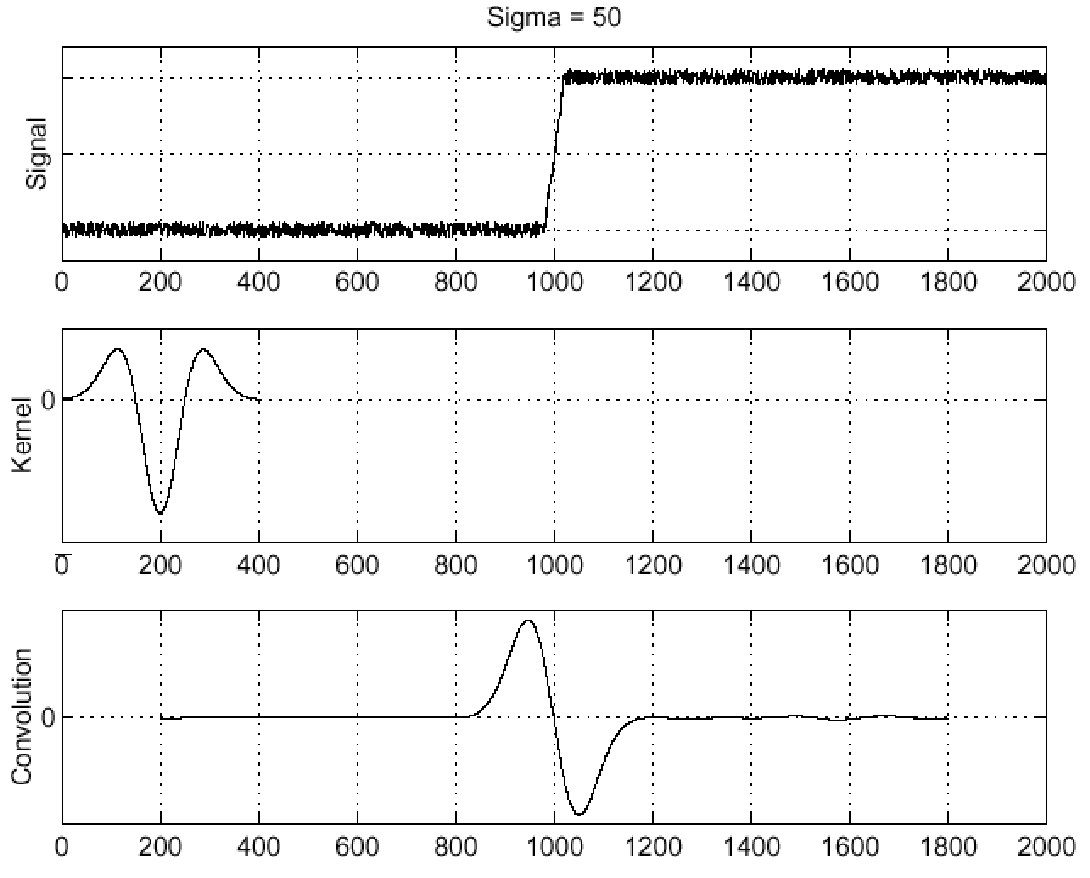
f

h: Gaussian smoothing kernel

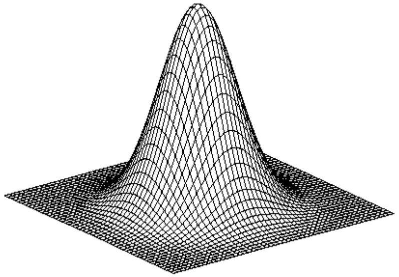
l: Laplace-kernel

$$\frac{\partial^2}{\partial x^2}h \approx l * h$$

$$(\frac{\partial^2}{\partial x^2}h) \star f$$

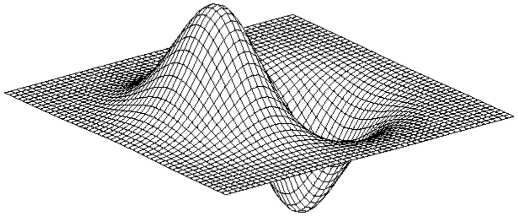


2D edge detection with filtering:



2D Gaussian smoothing kernel

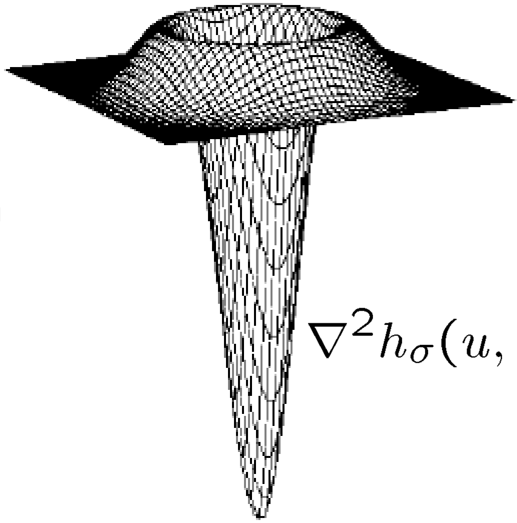
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



„Derivative" of the 2D Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 henceforward the **Laplace** operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Determining kernel coefficients with discrete approximation of the 2D function

Edge Enhancement

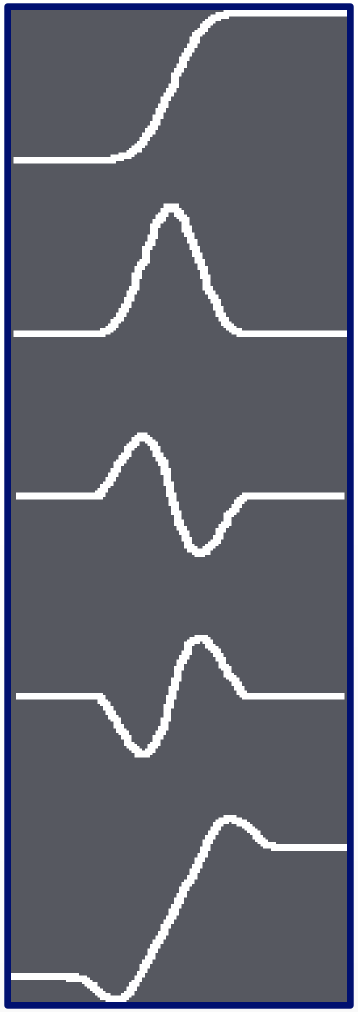
$$f(x)$$

$$\frac{\partial f}{\partial x}$$

$$\frac{\partial^2 f}{\partial x^2}$$

$$-\frac{\partial^2 f}{\partial x^2}$$

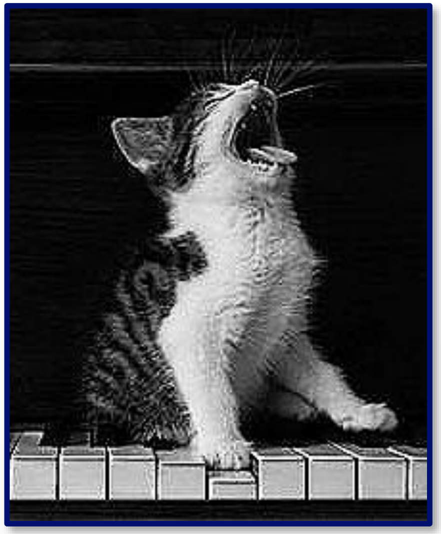
$$f(x)-\frac{\partial^2 f}{\partial x^2}$$



Kernel for edge enhancement with Laplace operator:

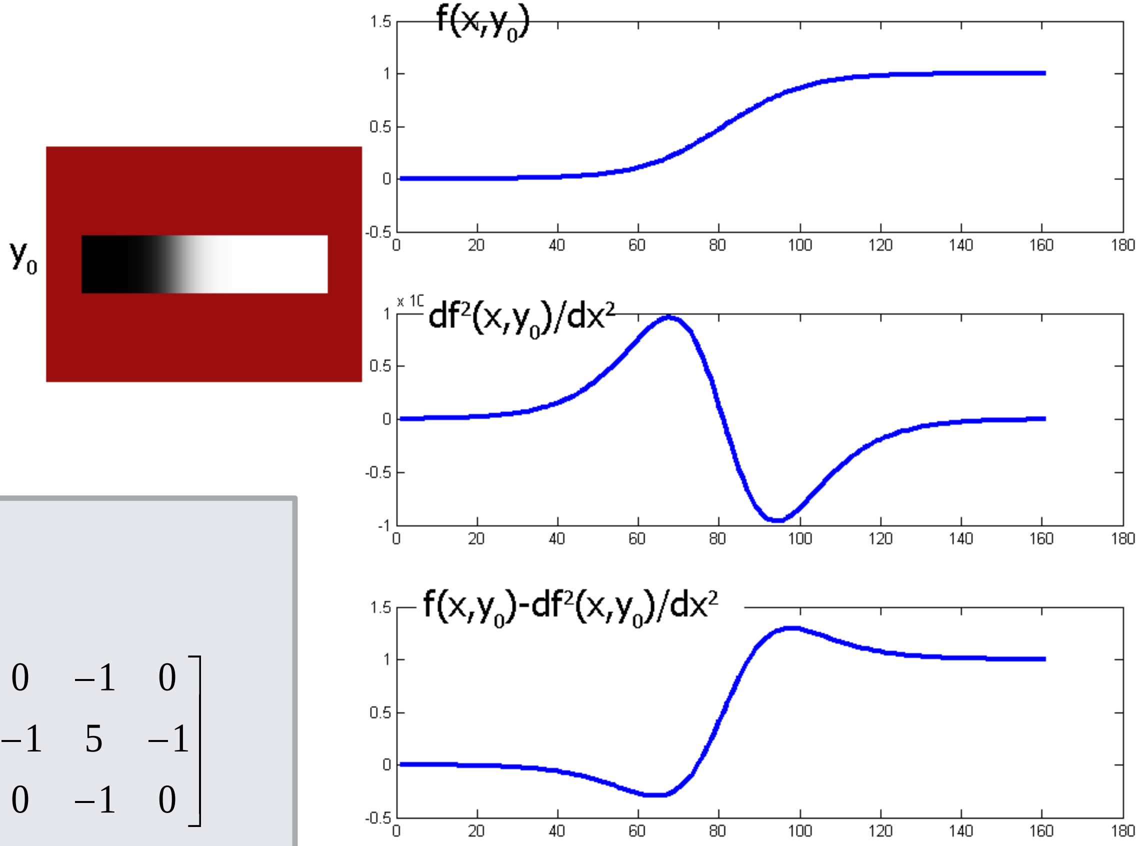
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \boxed{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}}$$

Original image



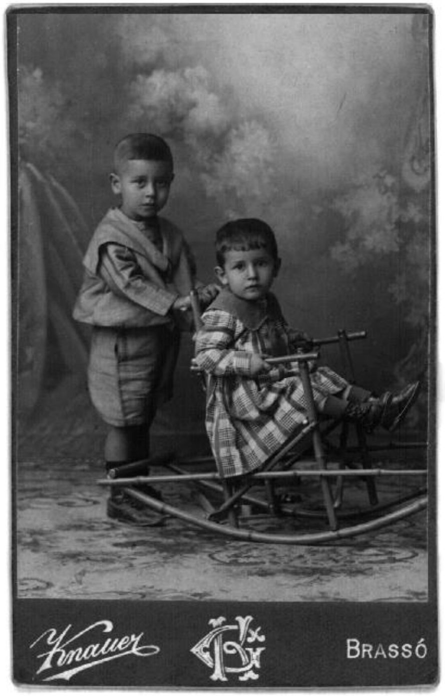
Edge enhanced image

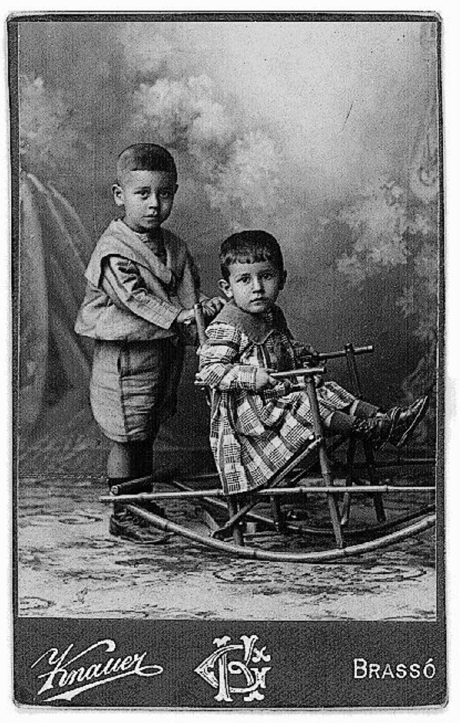
Edge crispening

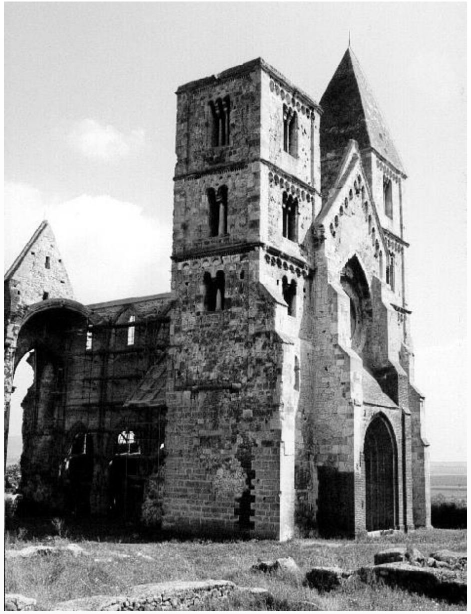


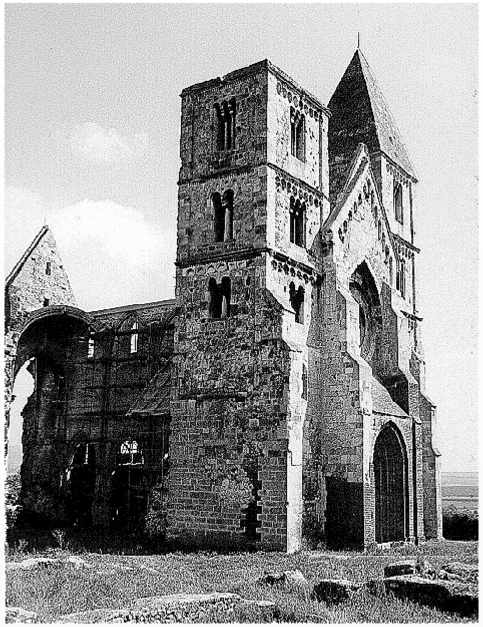
Convolution matrix:

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

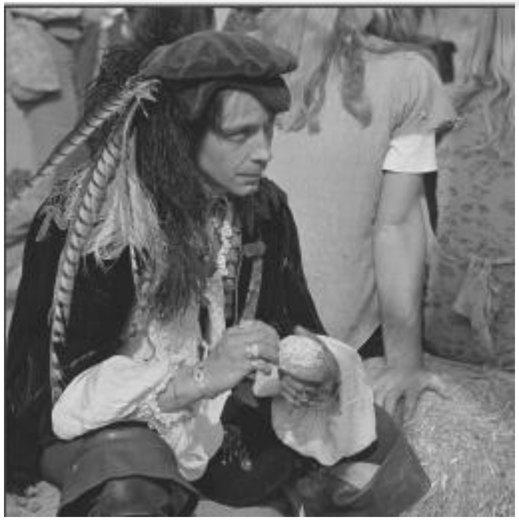

Knauer
BRASSÓ


Knauer
BRASSÓ



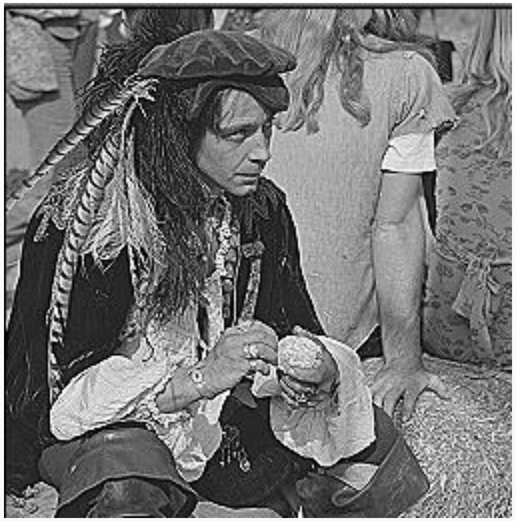


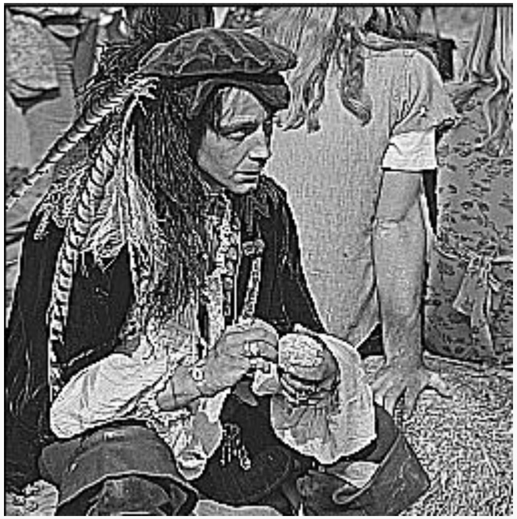
Edge crispening variants

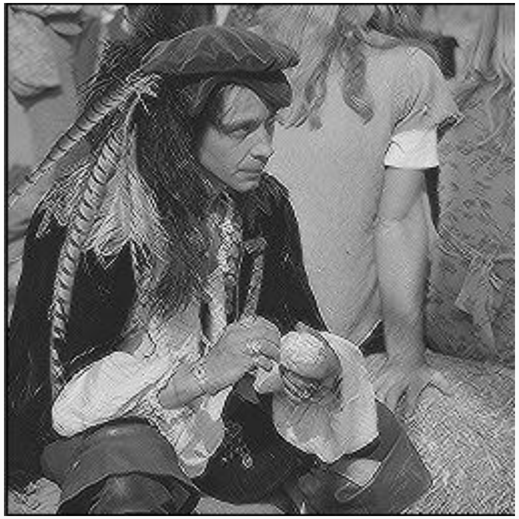


- Often enhances the image qualiy

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$



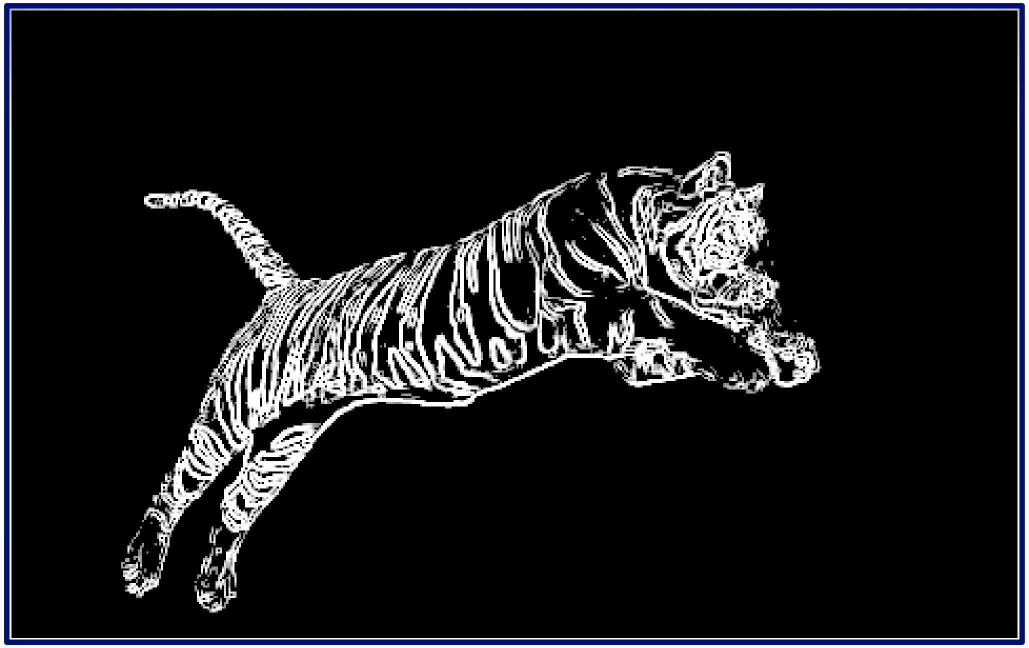


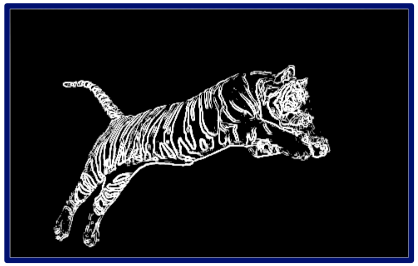


Canny edge detector

Evaluation of first/second order edge detection

- Prewitt kernel + threshold
  - is it a good edge detector?





Prewitt: is it a good edge detector?

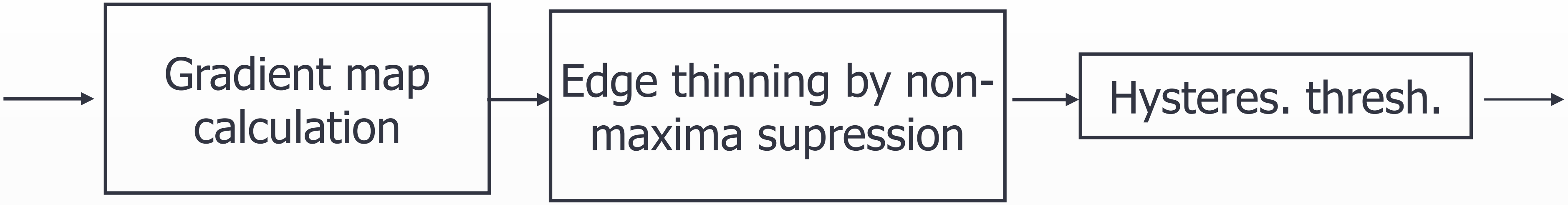
Canny edge detector

- Remember: properties of a good edge detector:
  - Good detection:
    - detects as many real edges as possible
    - does not create false edges
  - Good localization:
    - the detected edges should be as close to the real edges as possible
  - Isotropic:
    - all edges are detected regardless of their direction

- John F. ***Canny*** has developed an edge detector in 1986 to meet these requirements.

Canny edge detector

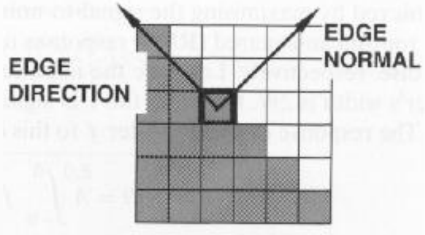
- Goal: extracting a **connected**, ***one-pixel-thick*** edge network
- Filtering Gaussian noise
- Three main steps:



Canny - 1st step: gradient map

- ***Noise reduction:***
  - The original image is convolved with a Gaussian kernel to reduce image noise.
- ***Gradient intensity and direction calculation:***
  - The horizontal and vertical derivative image is calculated (e.g. with Prewitt kernel)
  - At each pixel (i, j) calculate:
    - $d(i, j)$ gradient magnitude (how sharp is the edge – proportional to the gradient magnitude)

$$\|\nabla f\|_{ij} \propto d(i,j) = \sqrt{[d^x(i,j)]^2 + [d^y(i,j)]^2}$$

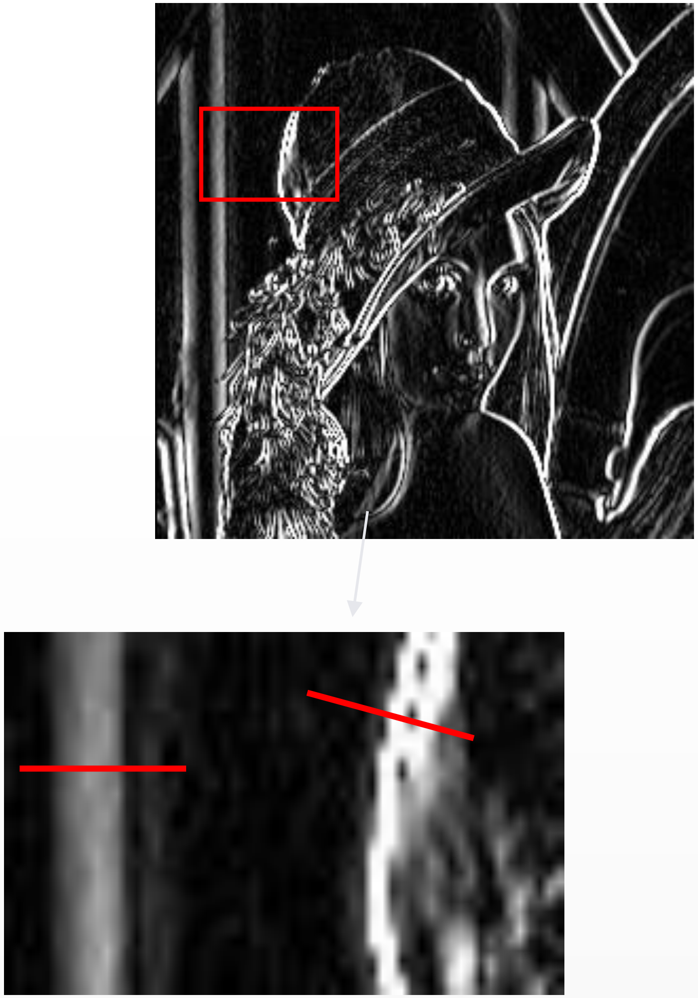


    - $n(i, j)$ edge normal (perpendicular to the direction)

$$n(i,j) \ = \arctan\left(\frac{d^x(i,j)}{d^y(i,j)}\right)$$

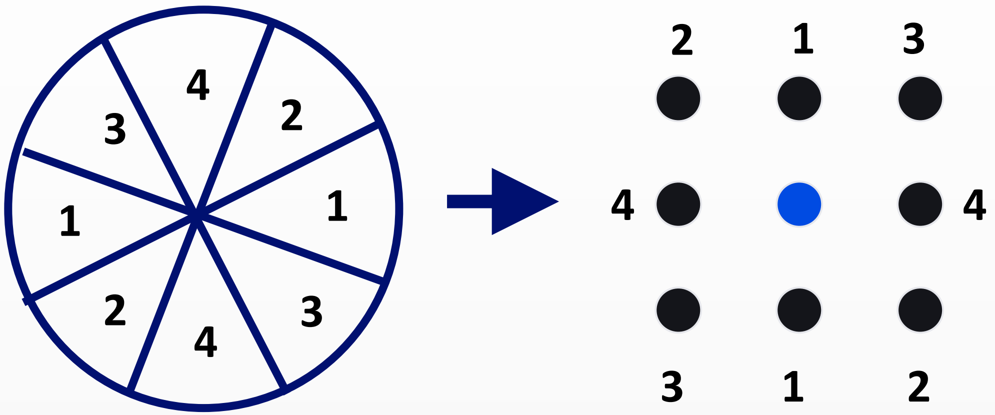
Canny – 2nd step: Non-max Suppression

- Goal: thinning the edges
- The gradient map may contain „thick" regions with large gradient values. Earlier methods may classify all of these points as edges.
- Along the highlighted line segments perpendicular to the edges (marked with red) we should only mark a single point as edge point, the one which is **locally the brightest**



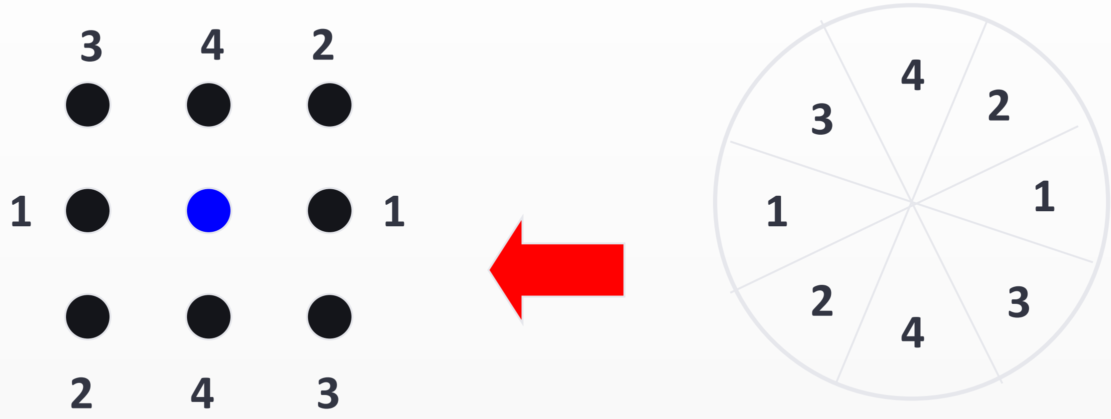
Canny Edge detector

3. ***Non-Maximum Suppression step for edge thinning:***
   - Each edge is categorized into one of *4 main edge directions* (0°, 45°, 90°, 135°), based on the gradient direction image (θ).
   - At every pixel, it suppresses the edge, by setting its value to 0, if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction:



Canny – 2nd step: Non-max Suppression

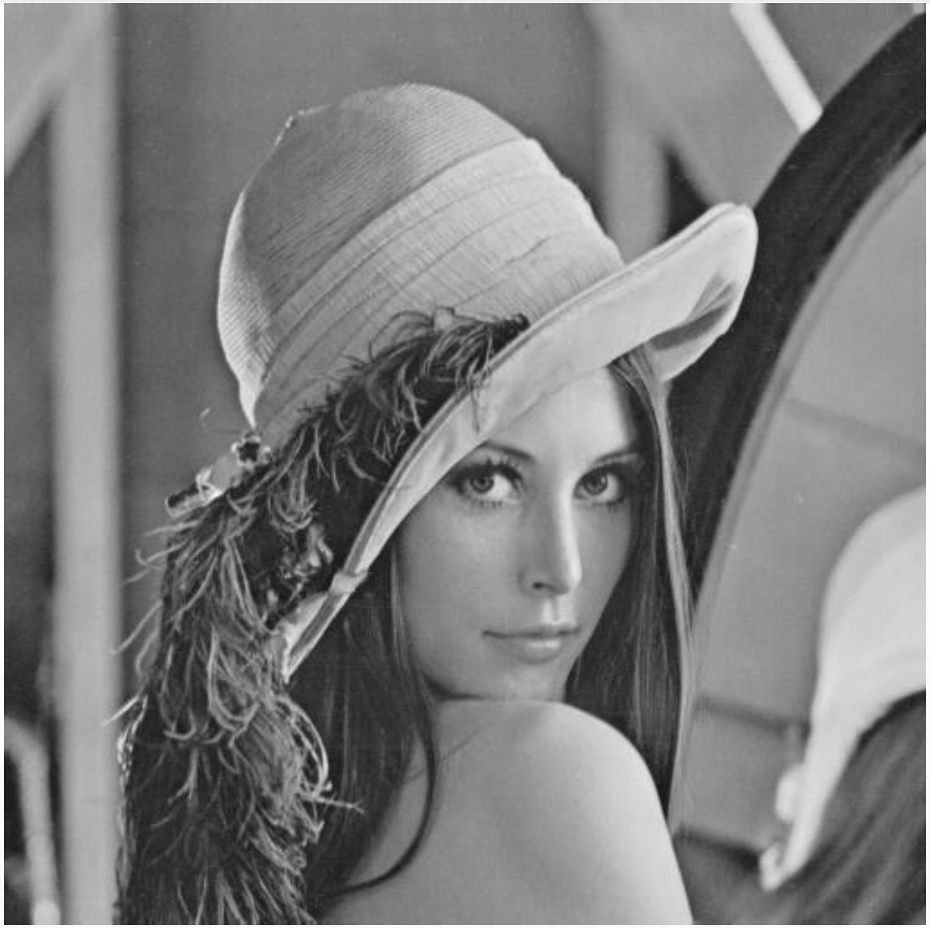
1. Each edge is categorized into one of 4 main edge directions (0°, 45°, 90°, 135°), based on the gradient direction image
   - to each pixel (i,j) we assign the principal direction $a(i,j)$, which one is the closest to local edge normal $n(i,j)$
2. At every pixel, it suppresses the edge, if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction:
   - If local edge magnitude $d(i,j)$ is smaller than in any neighboring pixel in the $a(i,j)$ direction set $G(i,j) := 0$. Otherwise (local max) : $G(i,j) := d(i,j)$
3. Result: G image obtained from the *d* gradient-magnitude map, where the edge-candidate regions become thin



Canny – 3rd step: Thresholding

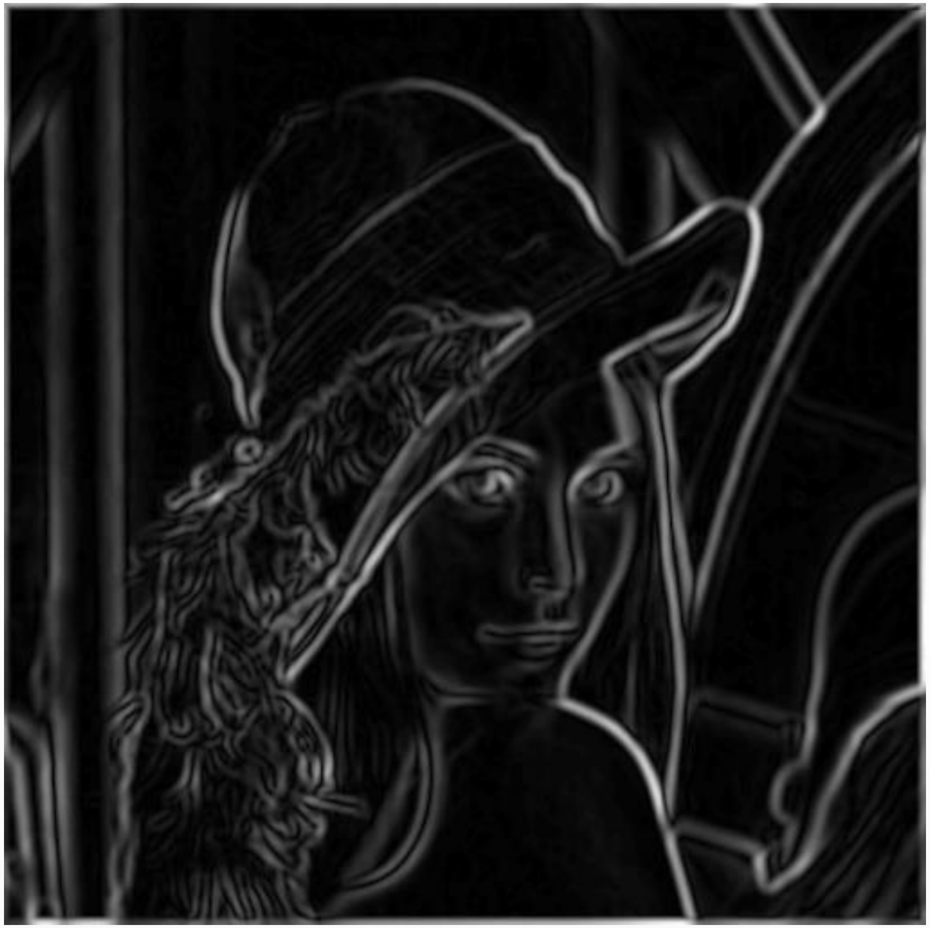
- Naive solution: thresholding the G map with a threshold t
  - If t is too small, we obtain many false edge points. If t is too large: valid edges disappear.
  - If the gradient magnitude of the edges fluctuates around the threshold, many disruptions (broken edge segments) may appear
- Improved solution: hysteresis thresholding
  - Using 2 thresholds t_1 and t_2 $(t_1 < t_2)$:
    - If the value of $G(x, y)$ is larger than t_2, (x, y) is certainly edge point
    - If the value of $G(x, y)$ is smaller than t_1, (x, y) is certainly *not an* edge point
    - If the value of $G(x, y)$ is between the threshold, we mark it as edge point if and only if it has a neighboring pixel already classified as edge in the direction perpendicular to the **edge normal**

Canny edge detector - result



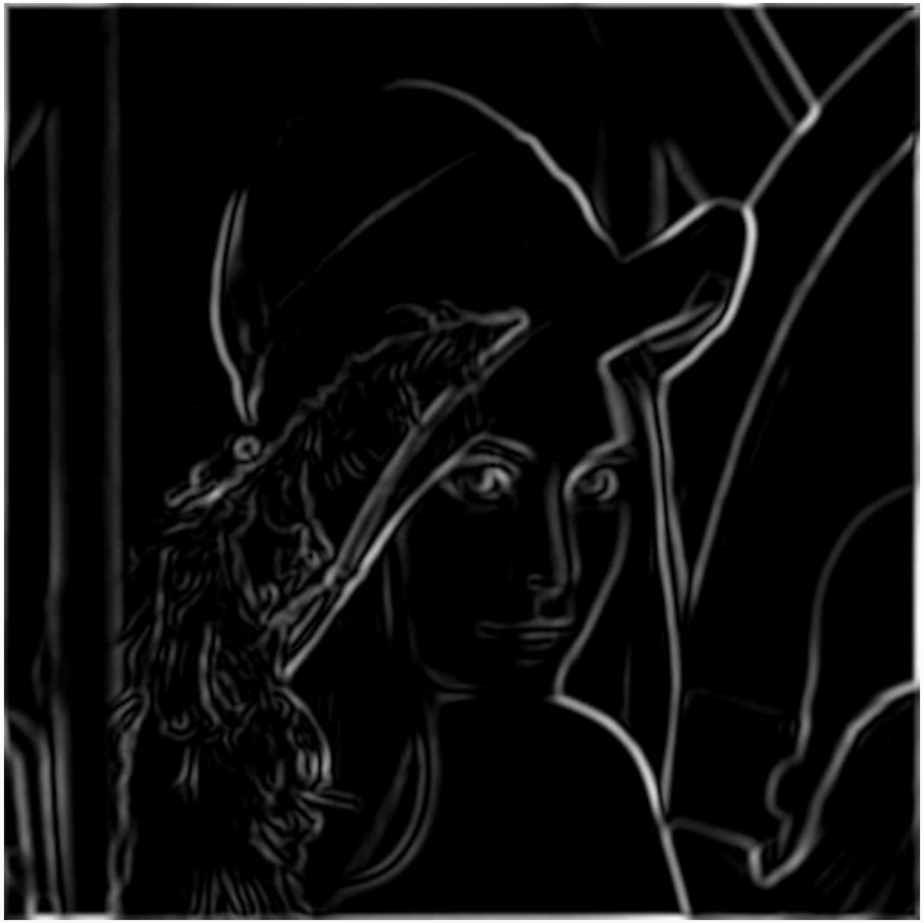
Input image

Canny edge detector - result



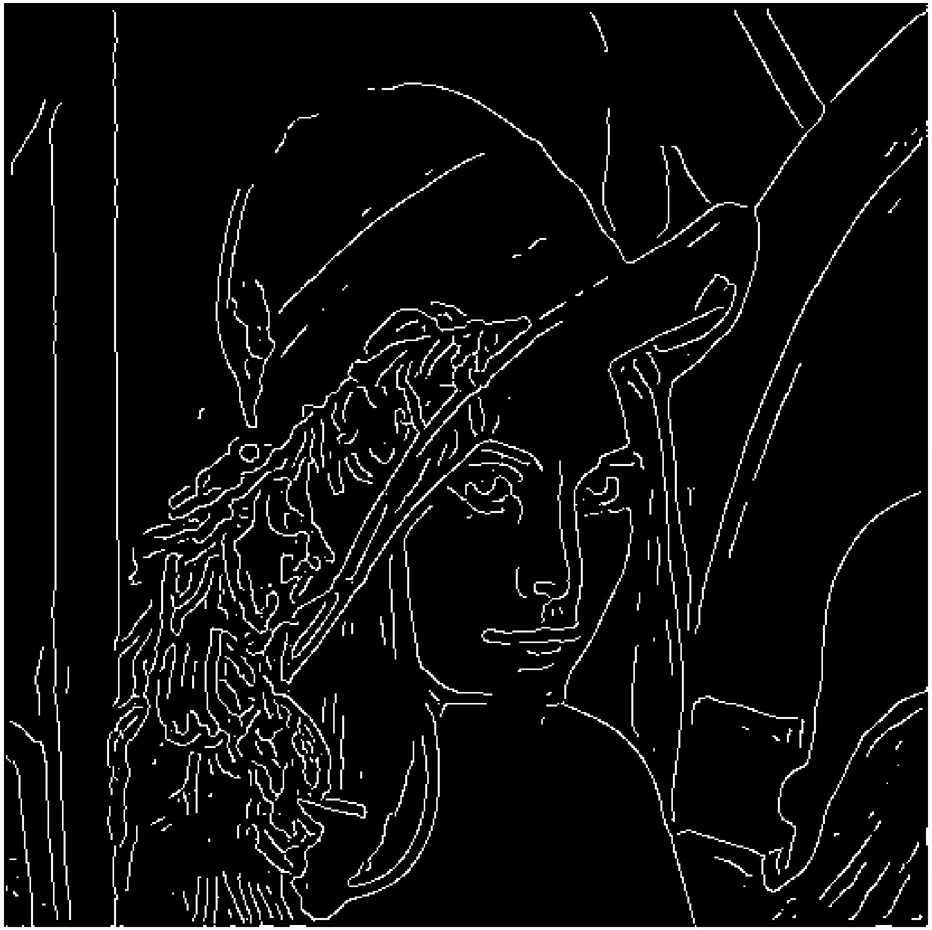
Norm of gradient: „d”

Canny edge detector - result

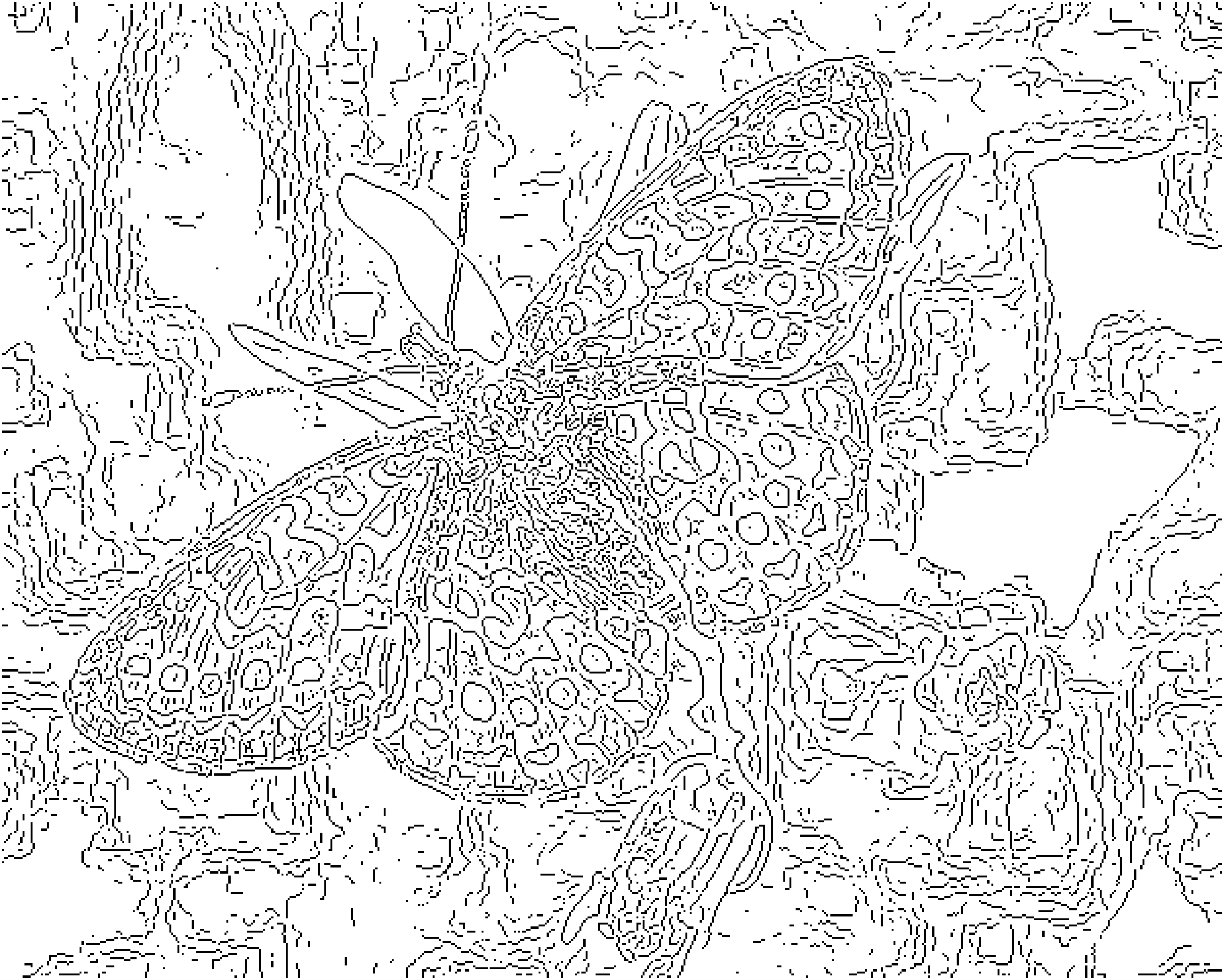


After thinning (E) (non-maximum suppression)

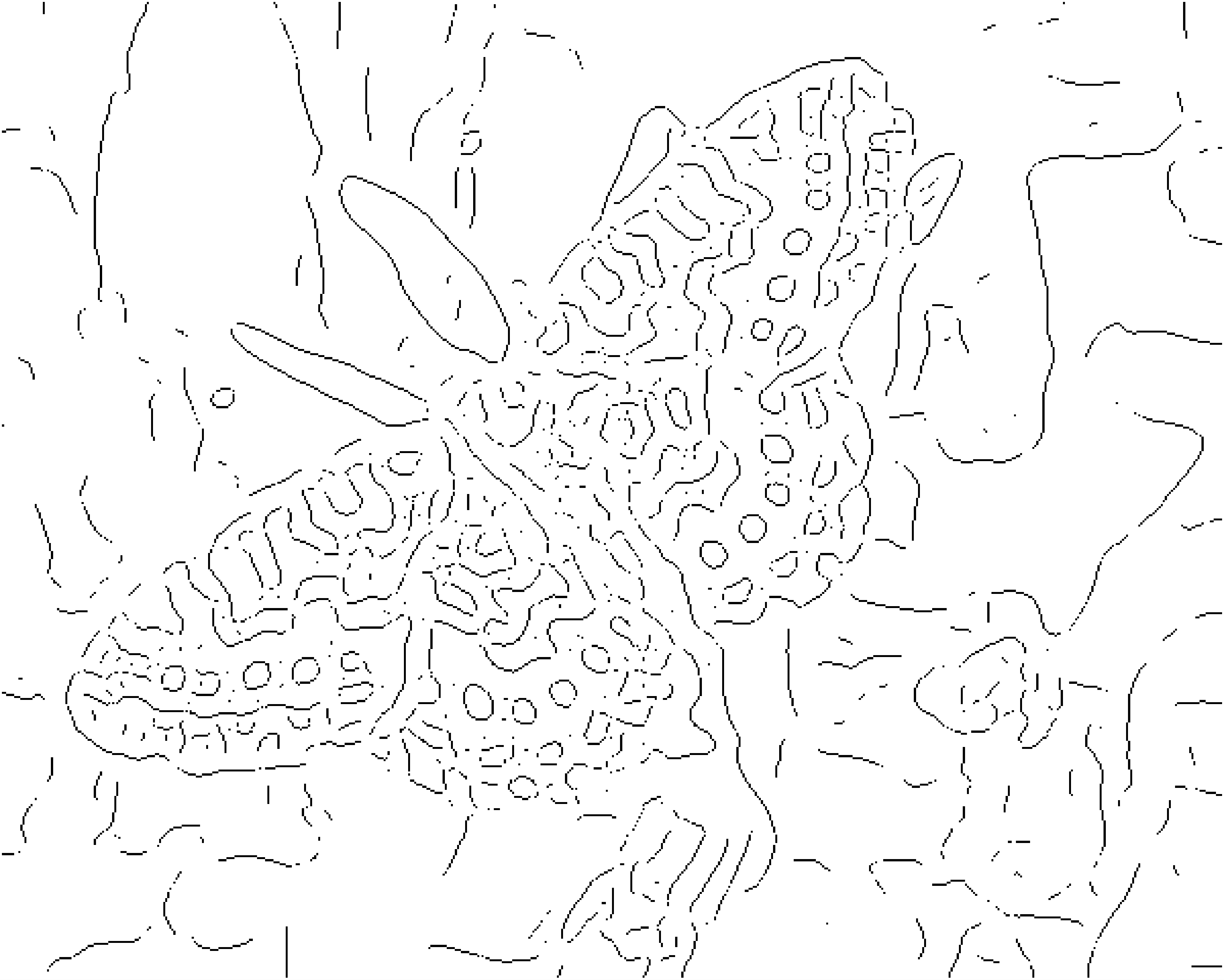
Canny edge detector - result



hysteresis thresholding

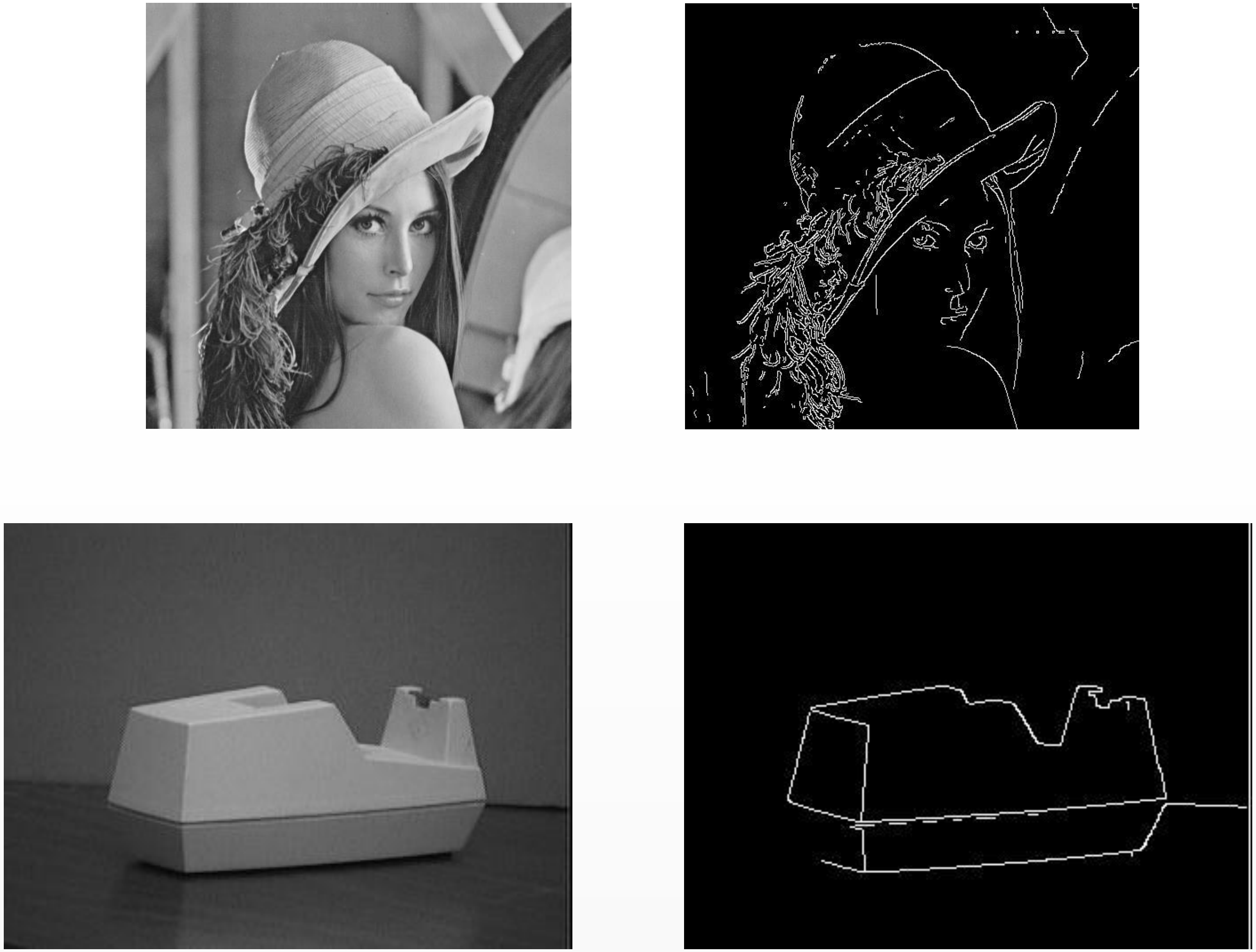


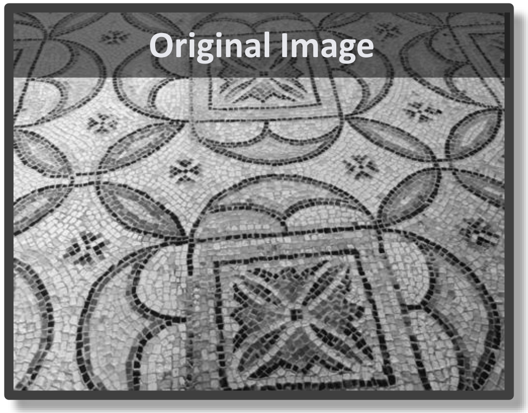
fine scale
high
threshold



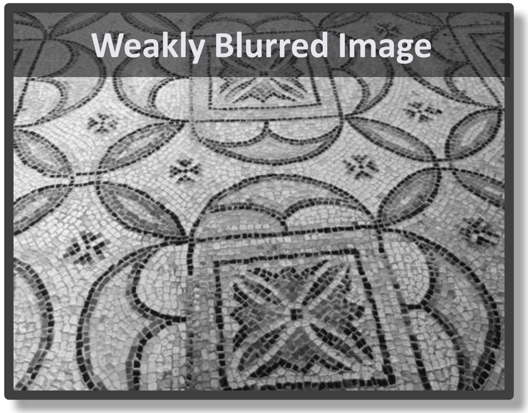
coarse scale, high threshold

Canny edge detector - results

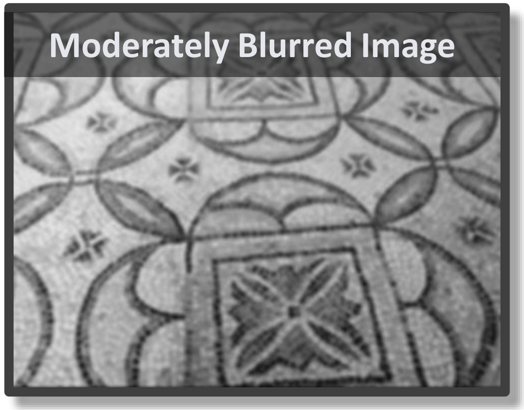




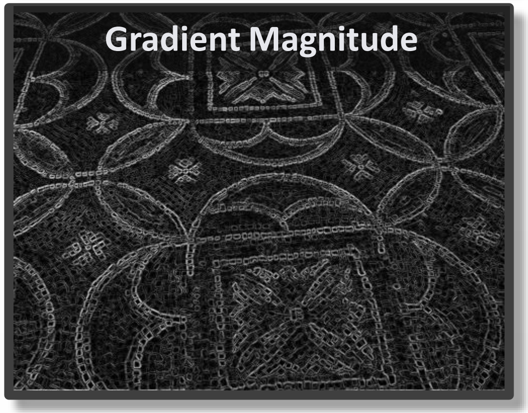
Original Image



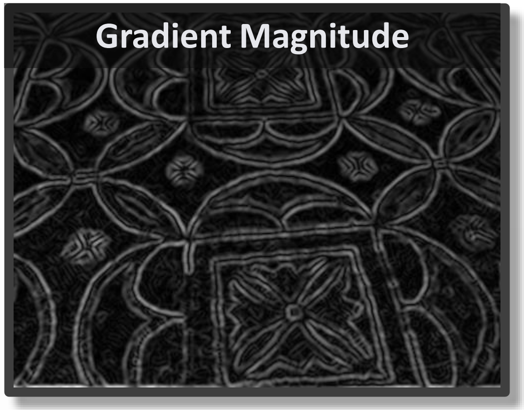
Weakly Blurred Image



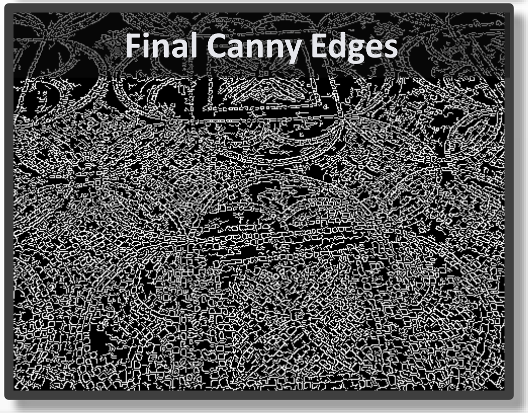
Moderately Blurred Image



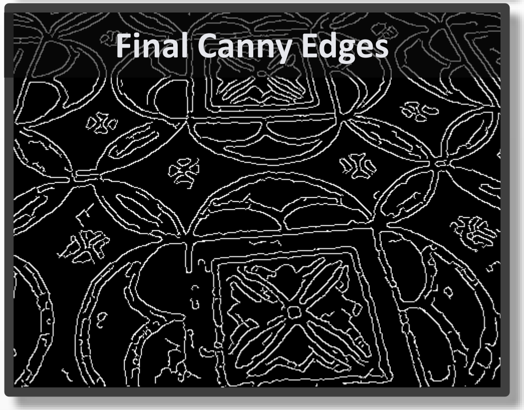
Gradient Magnitude



Gradient Magnitude



Final Canny Edges

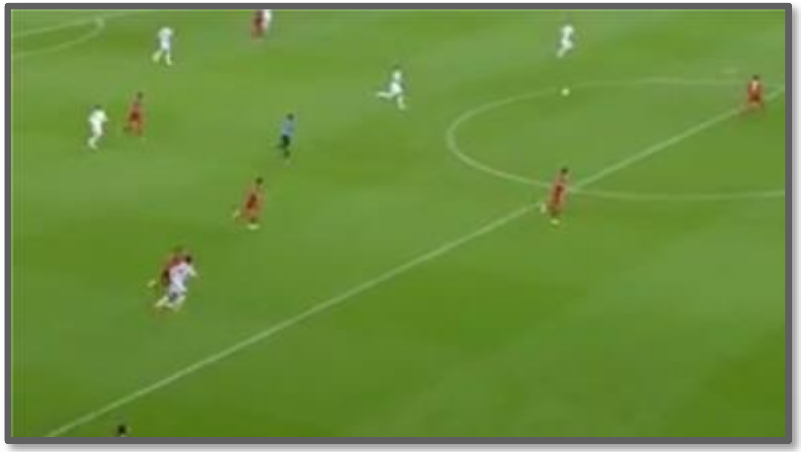


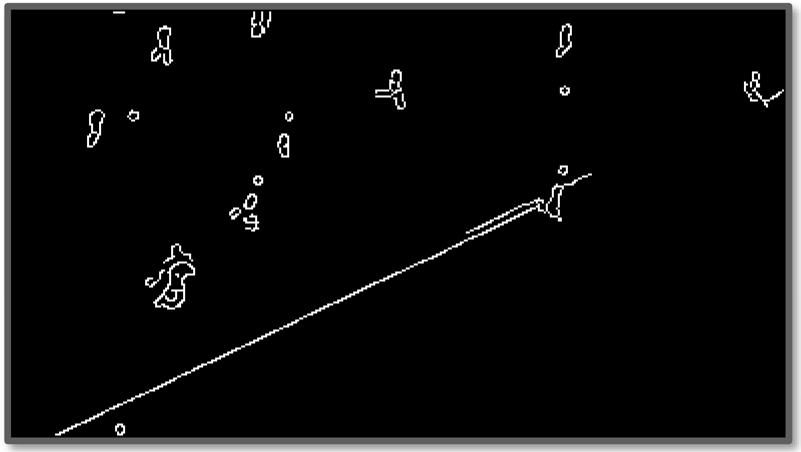
Final Canny Edges

Line detection with Hough Transfrom

Hough Transformation

- An example of Canny edge detector…





- …where straight lines are not detected perfectly.
- The objective of the Hough transformation is to find the lines on a binary image, from fragments/points of the line.

Finding lines in an image

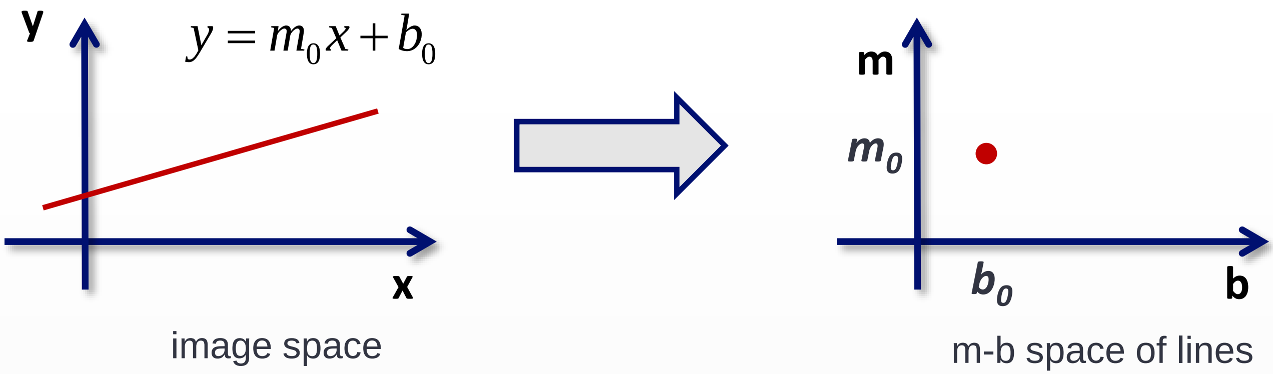
- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- Option 2:
  - Use a voting scheme: Hough transform

Finding lines in an image

- The basic idea:
  - A line can be written in the following form:

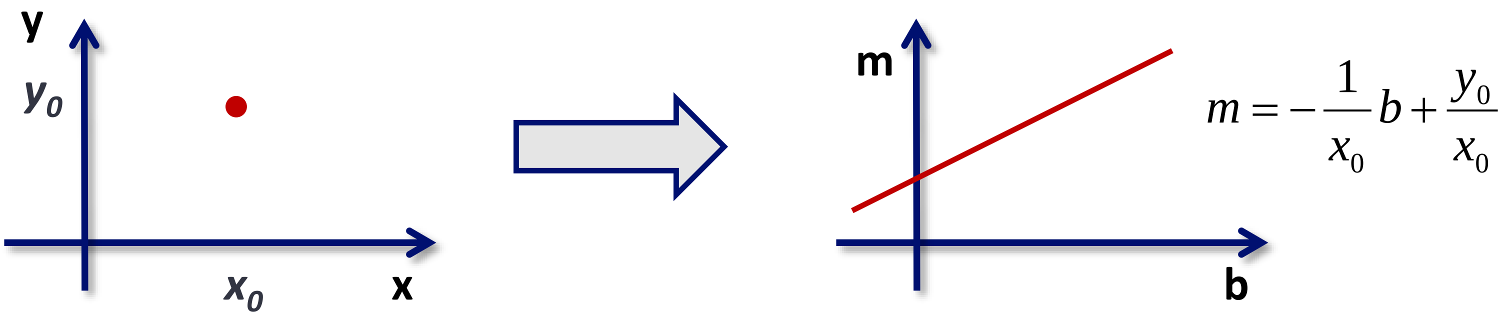
$$y = mx + b$$

where ***m*** is the slope of the line and ***b*** is the y-intercept.



- Connection between image ***(x,y)*** and the ***(m,b)*** spaces
  - A line in the image corresponds to a point in ”m-b” space
  - To go from image space to (m-b) space:
    - given a set of points ***(x,y)***, find all ***(m,b)*** such that ***y = mx + b***

Finding lines in an image

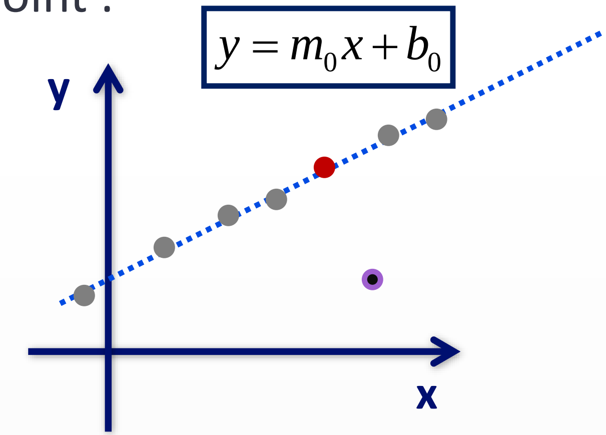


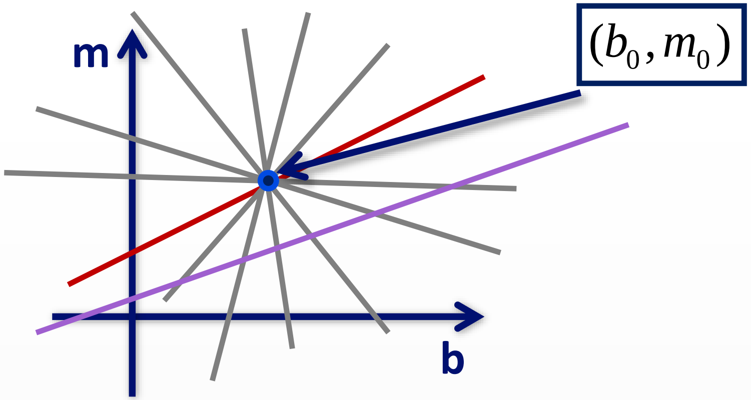
- Connection between image ***(x,y)*** and ***(m,b)*** spaces
  - What does a point ***(x₀, y₀)*** in the image space map to?
    - For a fixed ***y = y₀, x = x₀*** point in the image space, we get a line in the (m, b) space with a slope ***-1/ x₀*** and an m-intercept: ***y₀/ x₀*** :

$$m = -\frac{1}{x_0} b + \frac{y_0}{x_0}$$

Hough Transformation

- The basic idea:
  - For the points that lie on the same line in the Euclidian space, their corresponding line in the parameter space will cross each other in one point :

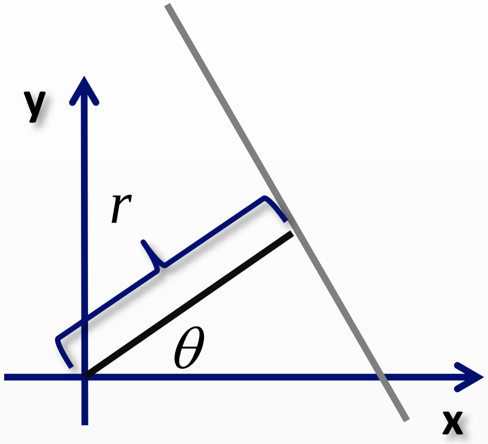




  - This point will be ***m=m₀*** and ***b=b₀***, the slope and intercept of the line in the image space. ⇨ We have the equation of the line!
- But, there is a problem with this equation of the line: vertical lines cannot be described (their slope would be infinite).

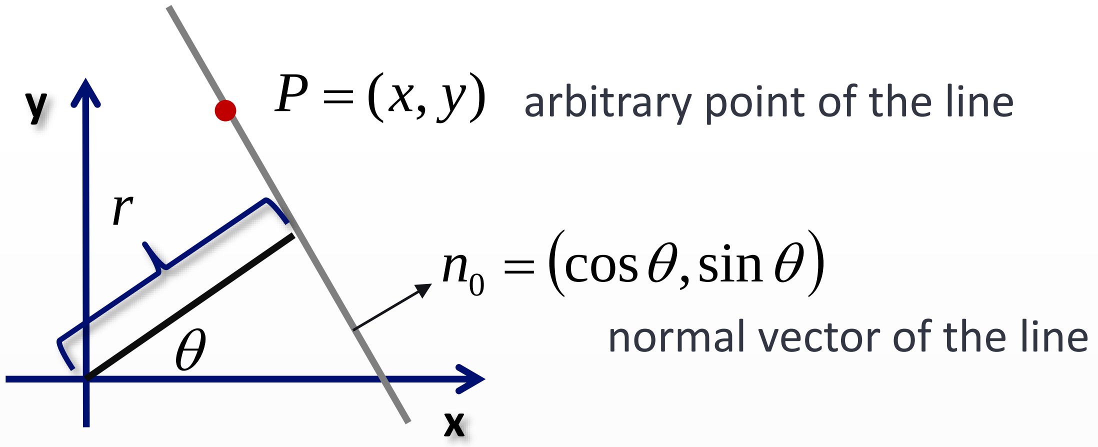
Hough Transformation

- To be able to describe all possible lines with two scalar parameters, we will use a **polar representation** of the line
- Each line is described by (r, θ) instead of (m, b), where
  - r is the perpendicular distance from the line to the origin
  - θ is the angle this perpendicular makes with the x axis



Hough Transformation

- Mathematical basis for using the **polar equation** of the line is the Hesse normal form*: $0 = P \cdot n_0 - r$



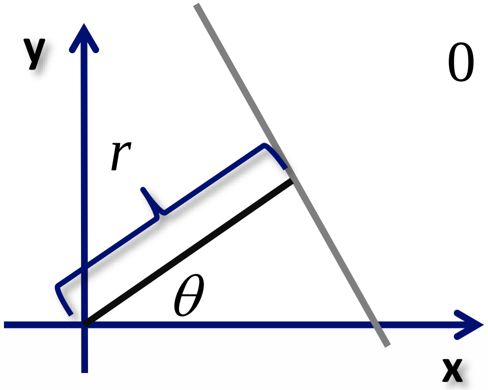
y
x
r
θ
$P = (x, y)$ arbitrary point of the line
$n_0 = (\cos\theta, \sin\theta)$
normal vector of the line

- *r:* perpendicular distance from the line to the origin
- θ: the angle this perpendicular makes with the x axis

* https://en.wikipedia.org/wiki/Hesse_normal_form

Hough Transformation

- Hesse normal form based polar equation of the line:



$$0 = P \cdot n_0 - r = (x, y) \cdot (\cos\theta, \sin\theta) - r$$

$$\Downarrow$$

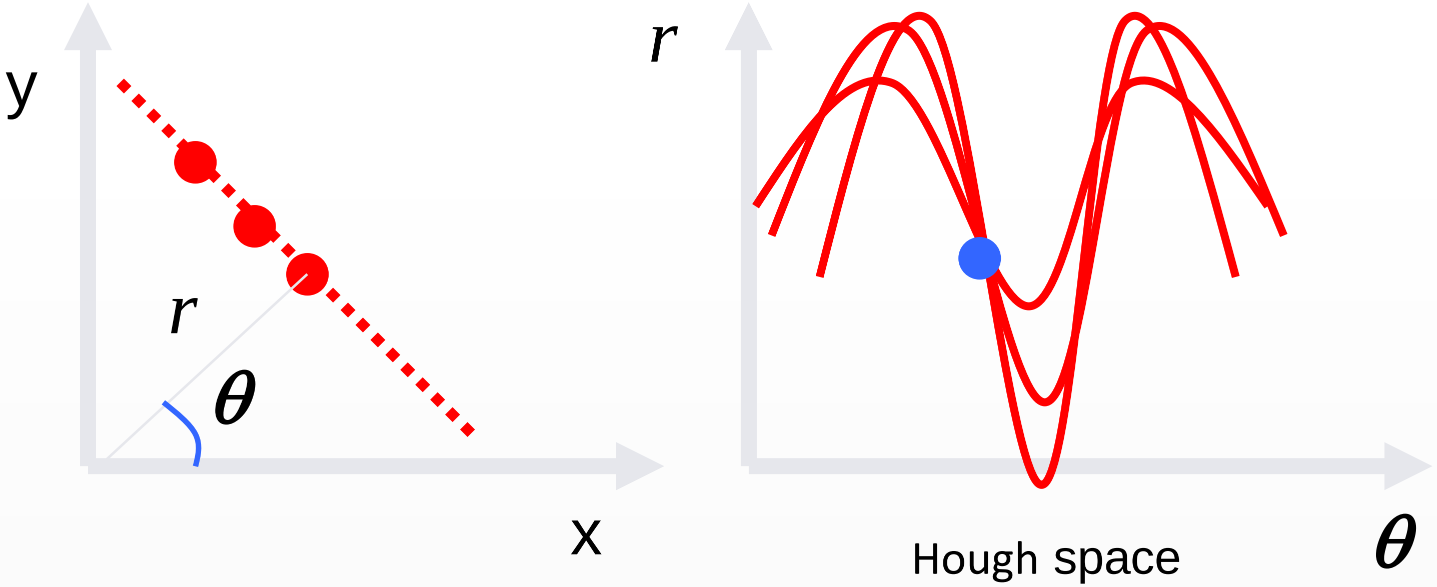
$$r = x \cdot \cos\theta + y \cdot \sin\theta$$

- The (r, θ) parameter space is called Hough space.
- A point in the Euclidian space is a sinusoid in the Hough space, described by the following equation:

$$r(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$$

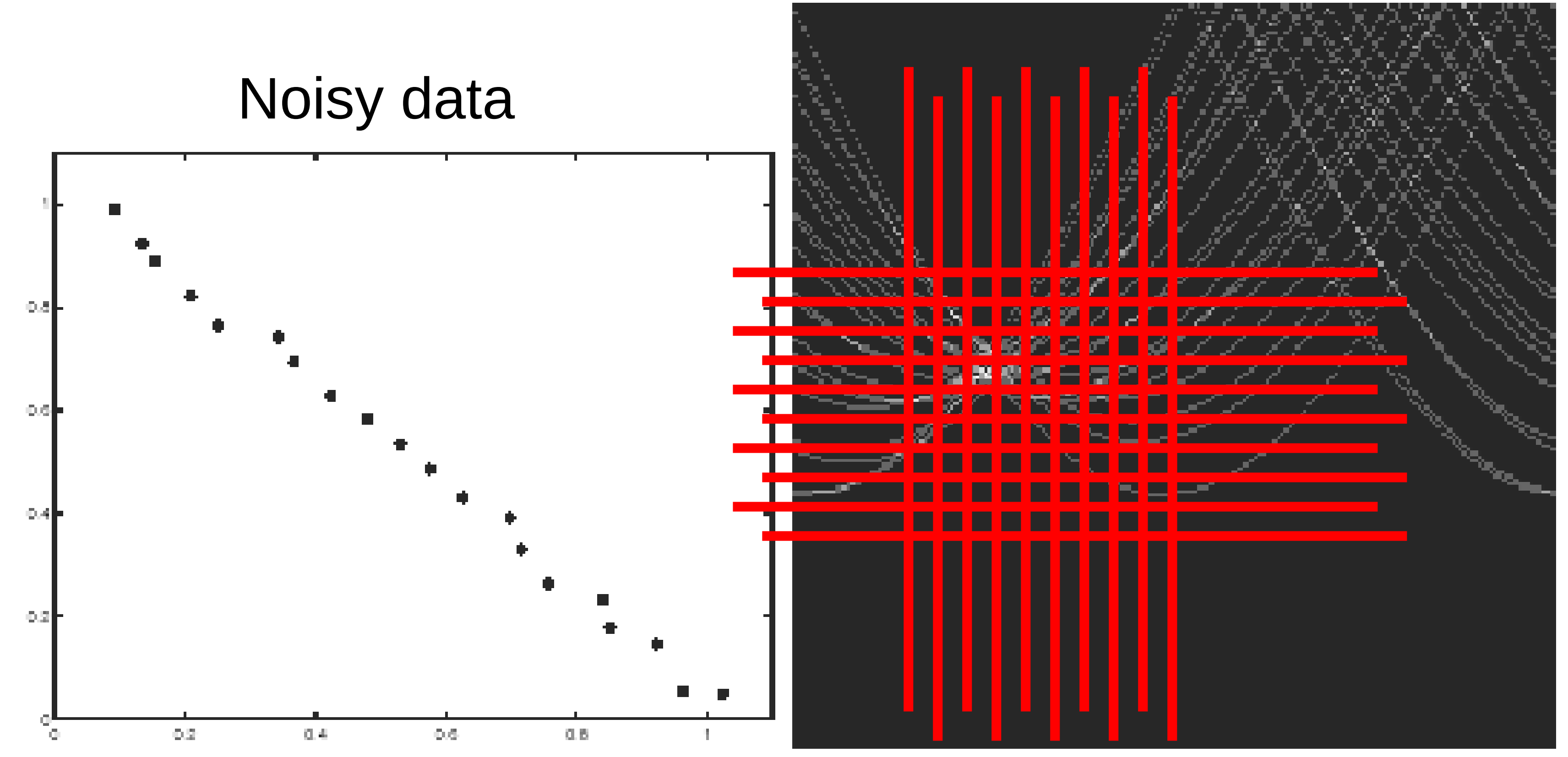
Hough Transformation

- All the sinusoid curves of the points in one line in the Euclidian space, cross each other in one point in the Hough space.



$$r(\theta) = x \cdot \cos\theta + y \cdot \sin\theta$$

Hough Transformation



features

votes

Issue: Grid size needs to be adjusted…

Hough transform algorithm

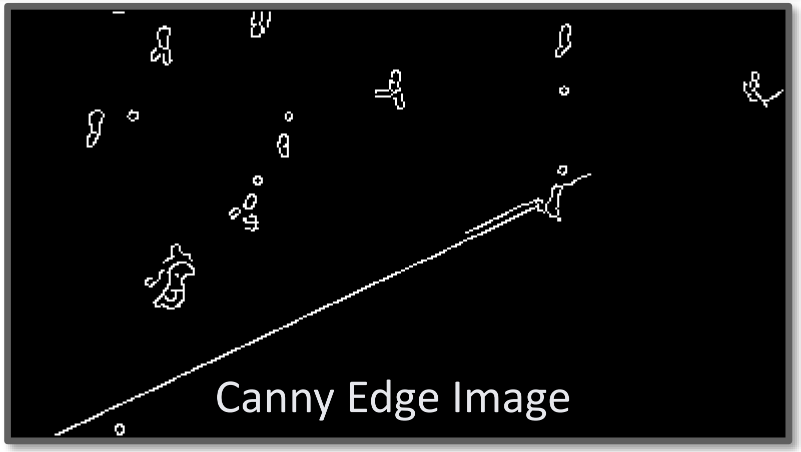
- Basic Hough transform algorithm

```
1. for all r, θ: initialize H[r, θ]=0
2. for each edge point I[x,y] in the image
       for θ = 0 to 180
           $r = x \cdot \cos\theta + y \cdot \sin\theta$
           H[r, θ] += 1
3. Find the value(s) of (r, θ) where H[r, θ] is maximum
4. The detected line in the image is given by
           $r = x \cdot \cos\theta + y \cdot \sin\theta$
```

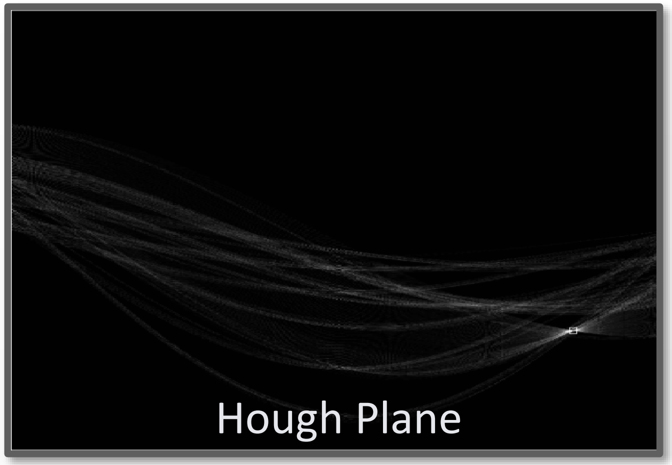
- What’s the running time (measured in # votes)?

Hough Transformation

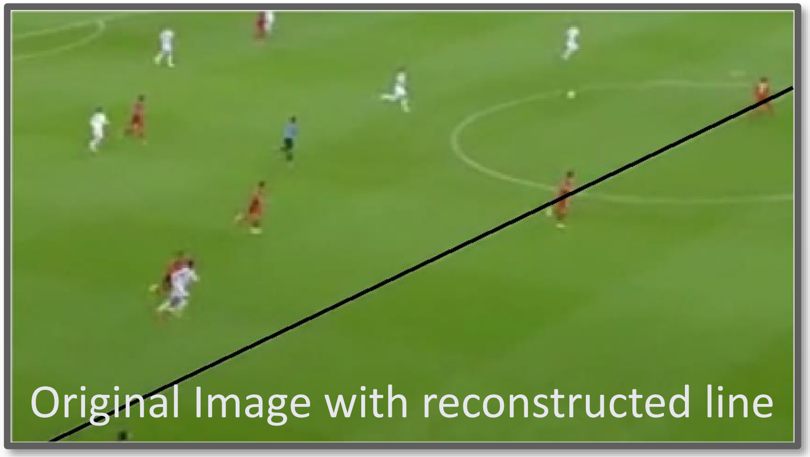
Original Image



Canny Edge Image



Hough Plane



Original Image with reconstructed line

Extensions

- Extension 1: Use the image gradient

```
1. same
2. for each edge point I[x,y] in the image
       compute unique (r, θ) based on local image gradient at (x,y)
           H[r, θ] += 1
3. same
4. same
```

- Extension 2
  - give more votes for stronger edges

- Extension 3
  - change the sampling of (r, θ) to give more/less resolution

- Extension 4
  - The same procedure can be used with circles, squares, or any other shape

Hough demos

- Lines, circles and ellipses: http://dersmon.github.io/HoughTransformationDemo/
- Circle : http://www.markschulze.net/java/hough/

Image Enhancement

What is Image Enhancement?

- **Image enhancement** is the manipulation or transformation of the image to improve the visual appearance or to help further automatic processing steps.
- There is no general theory behind it, the result is highly application dependent and subjective.
  - e.g. in many cases the goal is to improve the quality for human viewing (Medical Imaging, Satellite Images)
- Enhancement is closely related to image recovery.
- Examples:
  - Contrast enhancement
  - Edge enhancement
  - Noise removal/smoothing

Types of Image Enhancement

- There are two main categories:
  - Spatial Domain Methods
  - Frequency Domain Methods
- In the Spatial Domain we are directly manipulating pixel values, through..
  - Point-wise Intensity Transformation
  - Histogram Transformations
  - Spatial Filtering
    - LSI (*Linear Shift-Invariant*)
    - Non-Linear
  - etc.

The Histogram of an Image

- **Histogram**:
  $h(k)$ = the number of pixels on the image with value k.

Original Image*

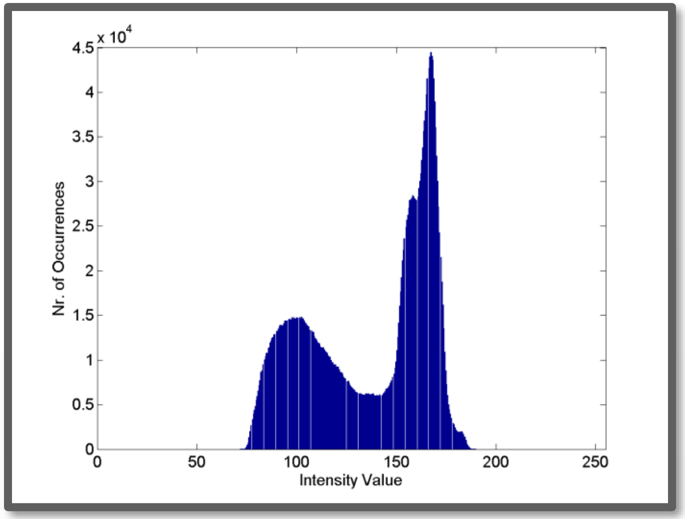


Image Histogram

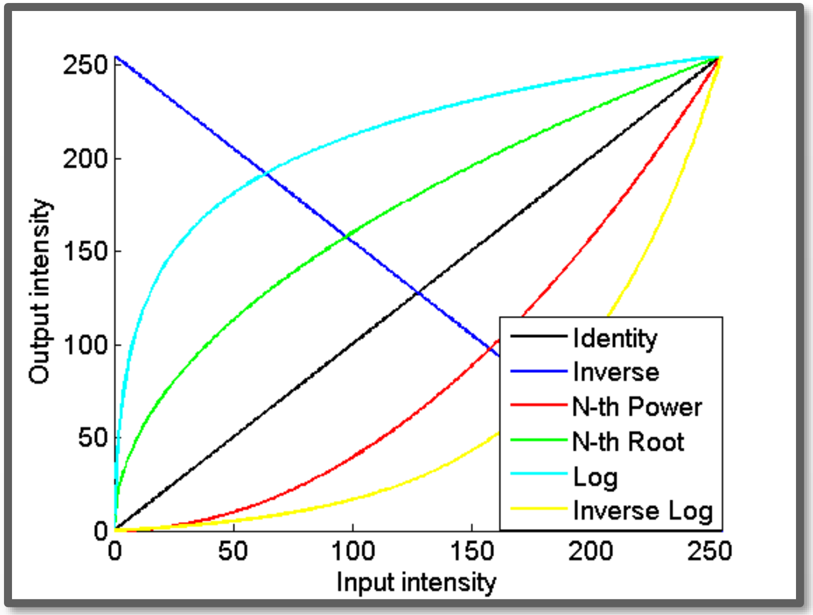
- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

Point-wise Intensity Transformation

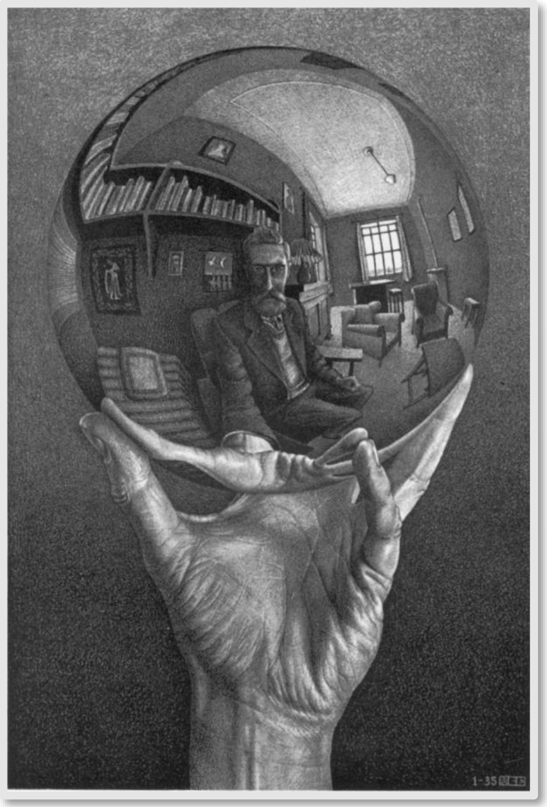
- Point wise transformations are operating directly on pixel values, independently of the values of its neighboring pixels.
- We can describe the transformation as follows:
  - Let *x* and *y* be two grayscale images, and let *T* be a point-wise image enhancement transformation that transforms *x* to *y*:

$$y(n_1, n_2) = T[x(n_1, n_2)]$$

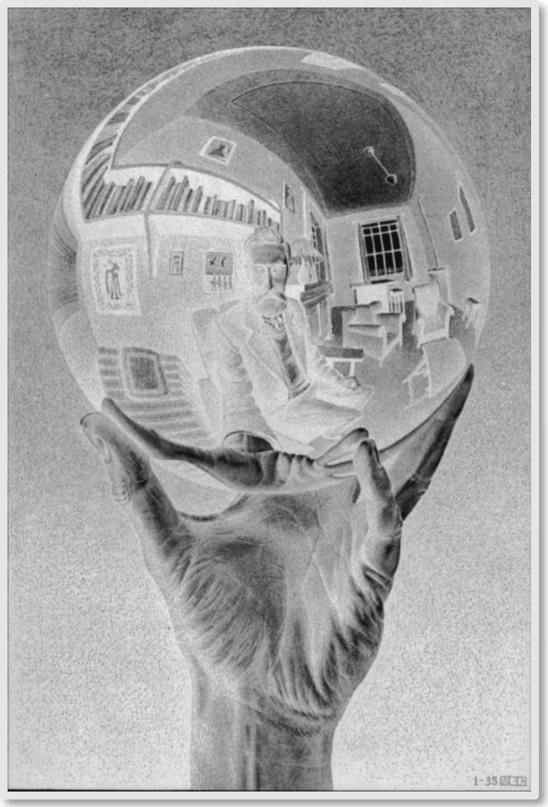


Point-wise Intensity Transformation

- **Inverse transformation:** $y(n_1, n_2) = 255 - x(n_1, n_2)$



Original Image*

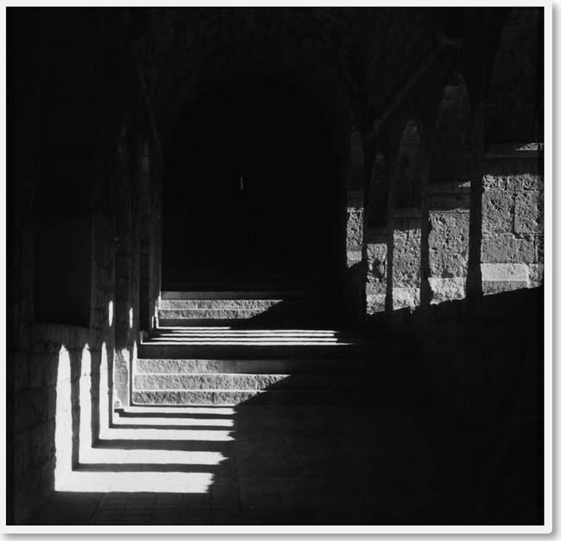


Inverse Image

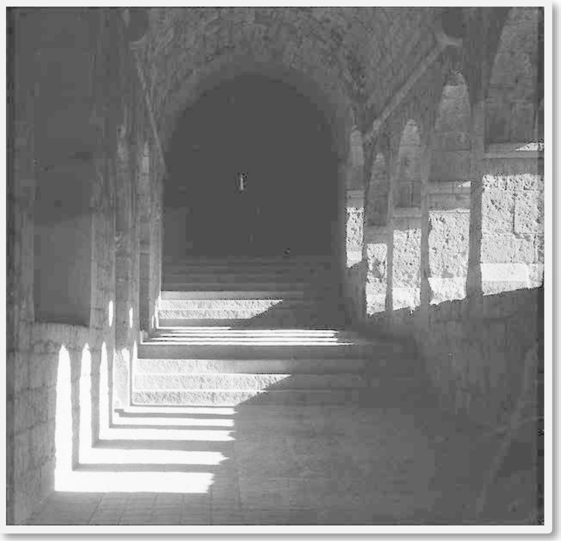
*Hand with Reflecting Sphere by M. S. Escher (1935)

Point-wise Intensity Transformation

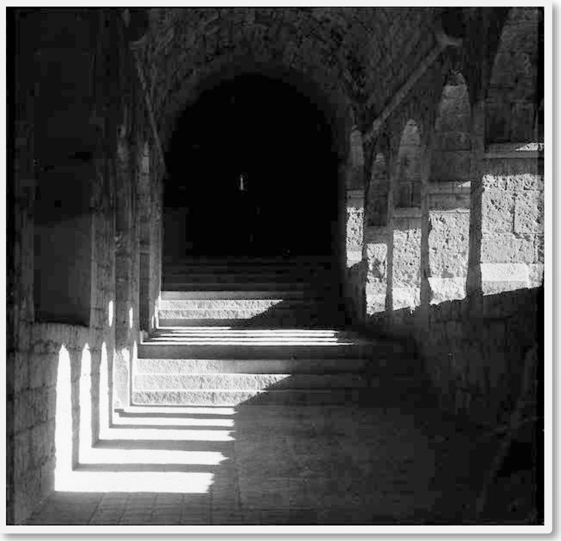
- **Log transformation:** $y(n_1, n_2) = c \cdot \log\left(x(n_1, n_2) + 1\right)$
  - Expands low and compresses high pixel value range



Original Image*



Log Image

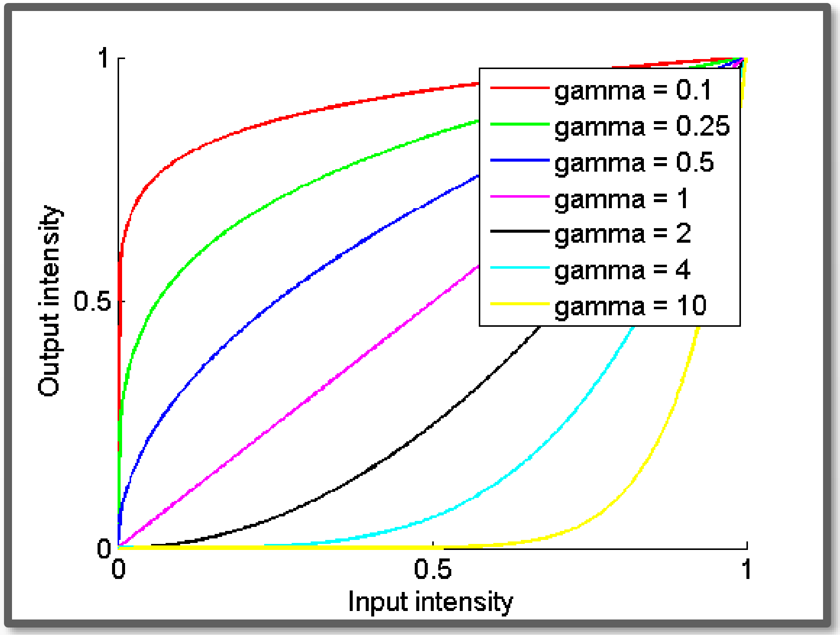


Log Image
after histogram stretching

* Abbaye du Thoronet by Lucien Hervé (1951)

Point-wise Intensity Transformation

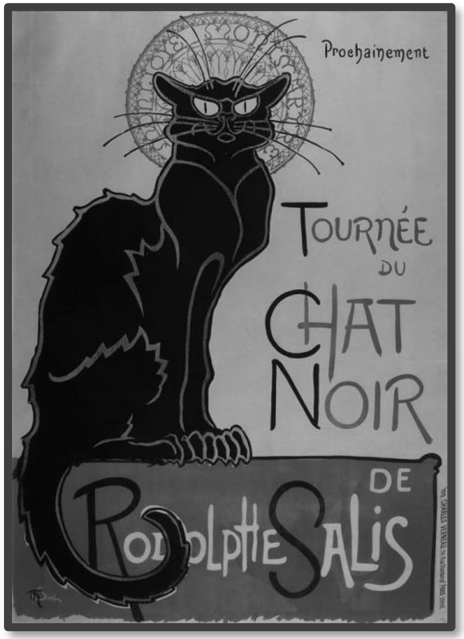
- **Power-law transformation**: $y(n_1, n_2) = c \cdot x(n_1, n_2)^{\gamma}$
  - Commonly referred to as ***gamma transformation***
  - Originally it was developed to compensate the input-output characteristics of CRT displays.
  - The expended/compressed region depends on γ:



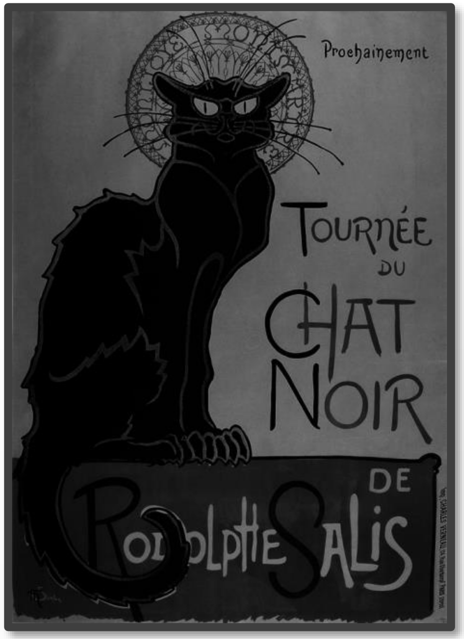
Here *c = 1*

Point-wise Intensity Transformation

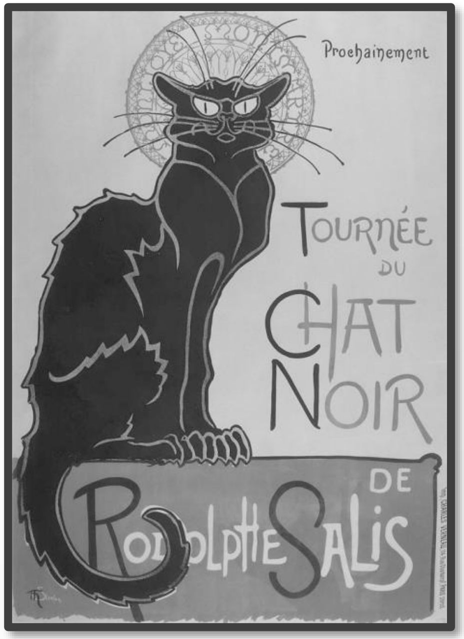
- **Power-law transformation:** $y(n_1, n_2) = c \cdot x(n_1, n_2)^{\gamma}$



Original Image*



γ = 2

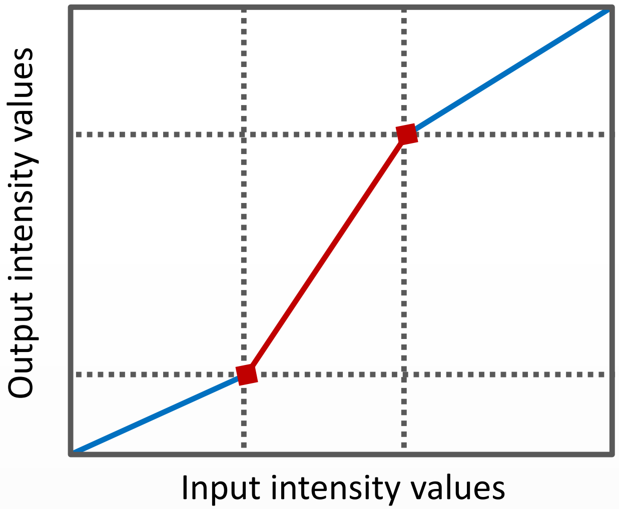


γ = 0.5

* Le chat Noir, Poster of Théophile Steinlen (1896)

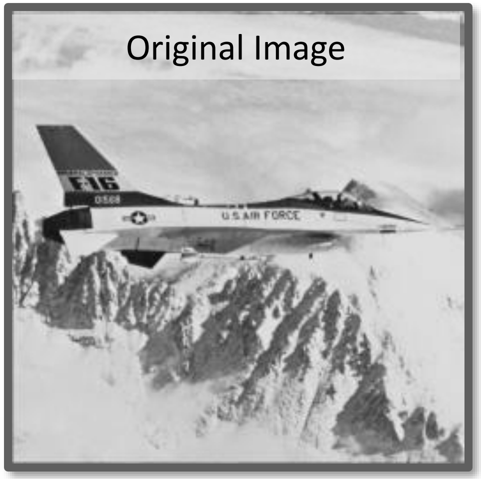
Dynamic Range Expansion

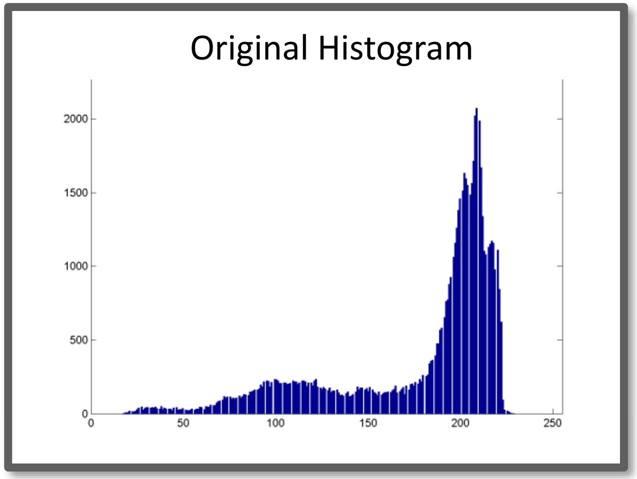
- Piecewise linear expansion/compression of predefined intensity ranges:

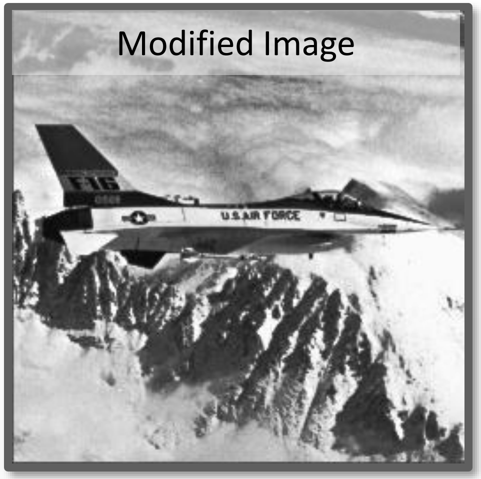


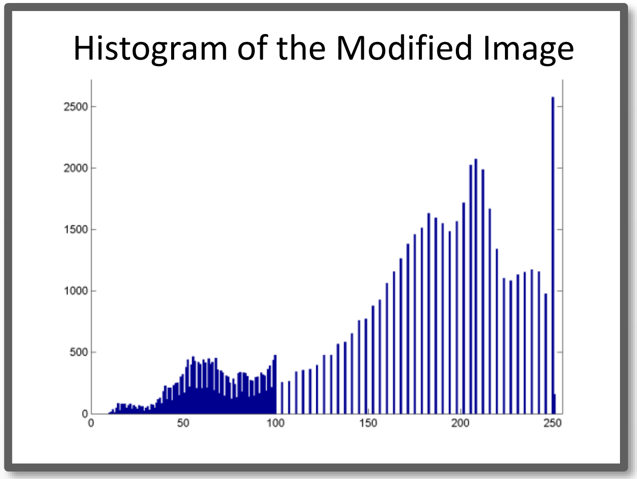
  - The red intensity range was expanded, while the blue ranges were compressed.

Dynamic Range Expansion


Original Image

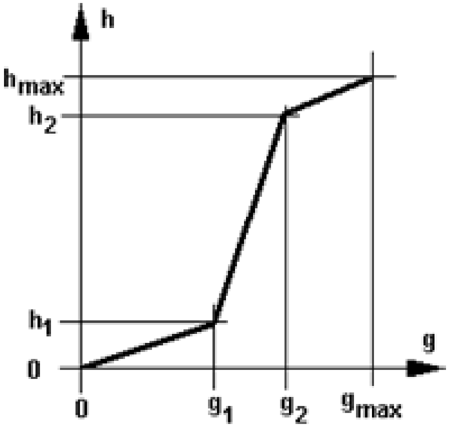

Original Histogram


Modified Image


Histogram of the Modified Image

Dynamic Range Expansion

- Example: extracting the intensity values from the $[g_1, g_2]$ interval to a wider $[h_1, h_2]$ domain
  - Enhanced contrast in the selected region, details are better observable and distinguishable.
  - In the remaining image regions the contrast decreases



Histogram Transformations

- **Histogram Stretching**:
  - Based on the histogram we can see that the image does not use the whole range of possible intensities:
    - Minimum intensity level: 72
    - Maximum intensity level: 190
  - With the following transformation we can stretch the intensity values so they use the whole available range:

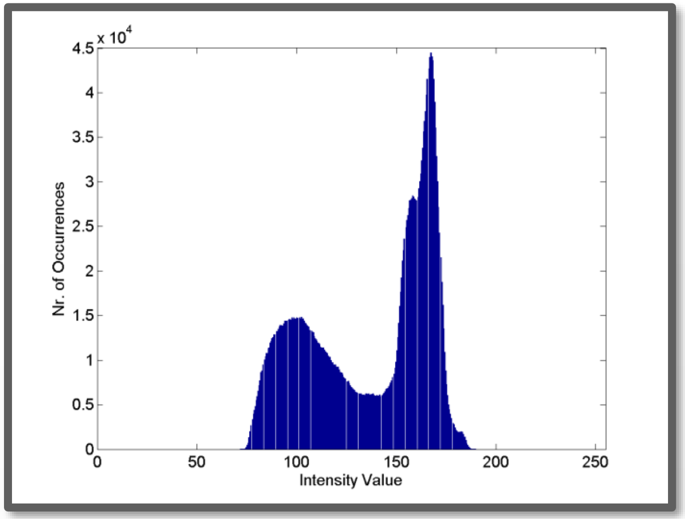


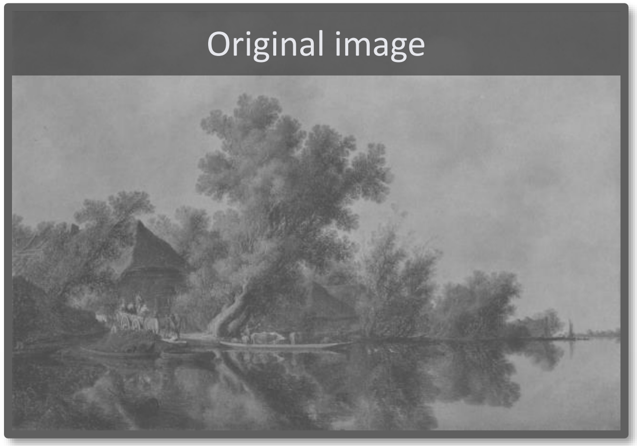
Image Histogram

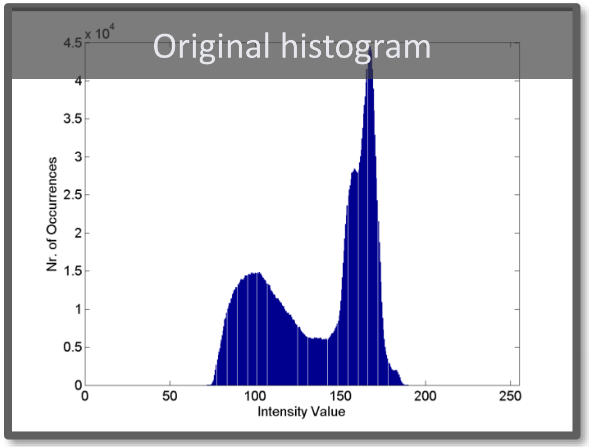
$$y(n_1, n_2) = \frac{255}{x_{\max} - x_{\min}} \cdot (x(n_1, n_2) - x_{\min})$$

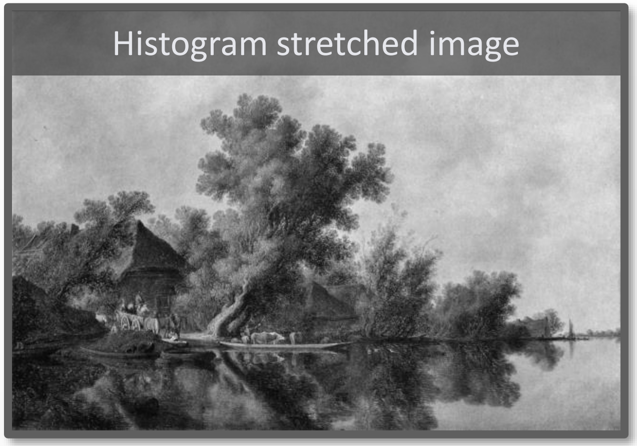
$$x_{\max} = \max_{n_1, n_2}(x(n_1, n_2)) \qquad x_{\min} = \min_{n_1, n_2}(x(n_1, n_2))$$

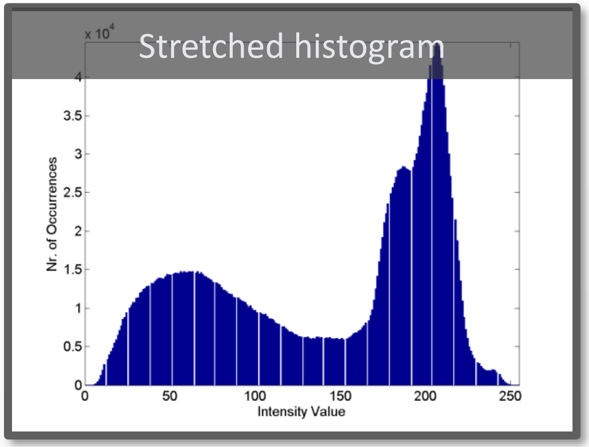
Histogram Transformations

- **Histogram Stretching:**


Original image


Original histogram


Histogram stretched image


Stretched histogram

Histogram Transformations

- Histogram stretching with various transfer functions:
  - Linear:

$$y(n_1,n_2)=\frac{255}{x_{\max}-x_{\min}}\cdot(x(n_1,n_2)-x_{\min})=255\cdot\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}$$

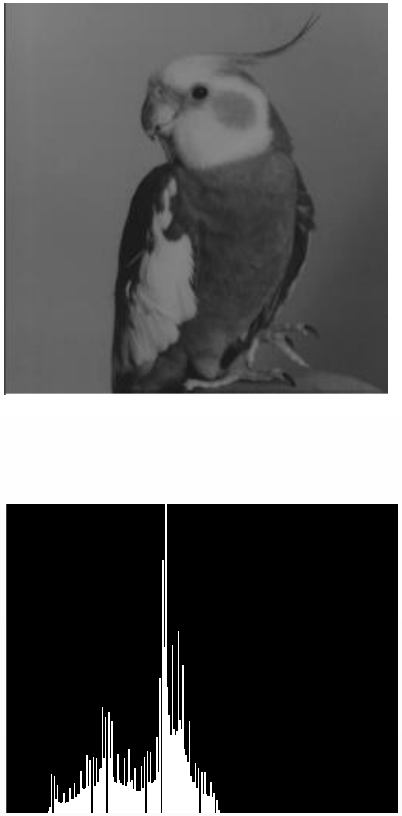
  - Quadratic:

$$y(n_1,n_2)=255\cdot\left(\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}\right)^2$$

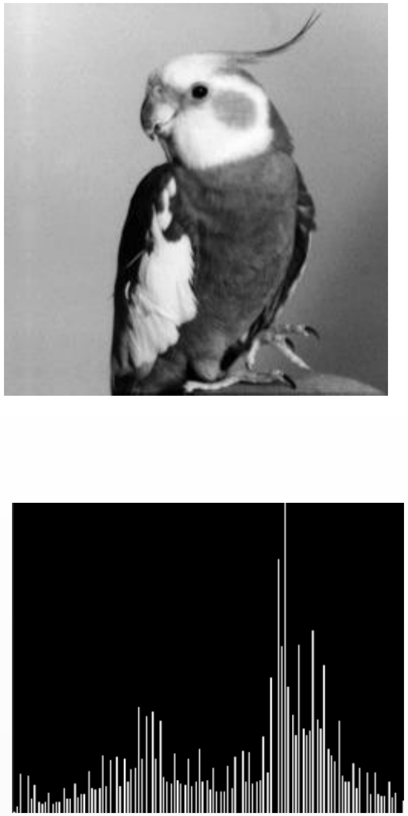
  - Square root

$$y(n_1,n_2)=255\cdot\sqrt{\frac{x(n_1,n_2)-x_{\min}}{x_{\max}-x_{\min}}}$$

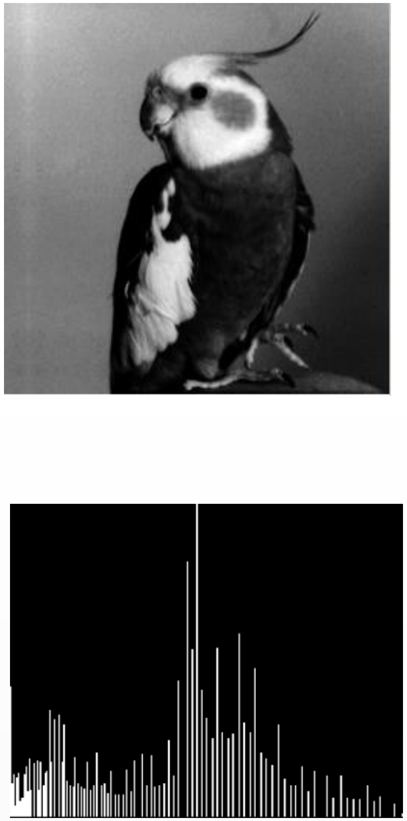
Histogram stretching - results



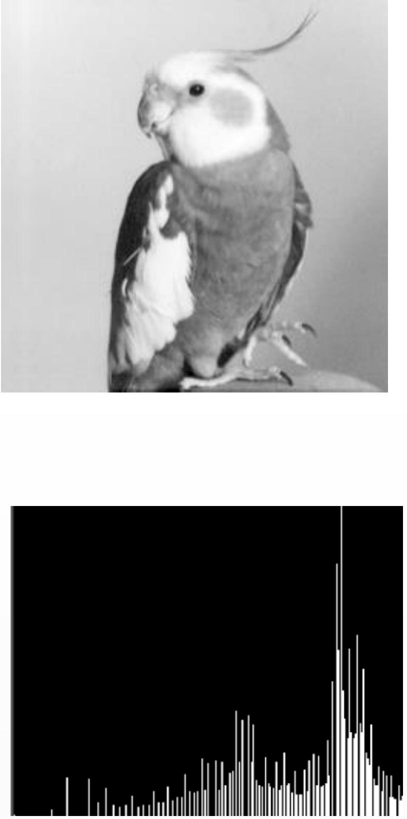
original



linear f()



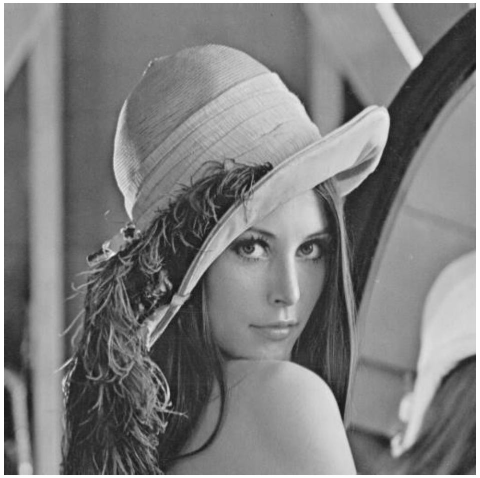
quadratic

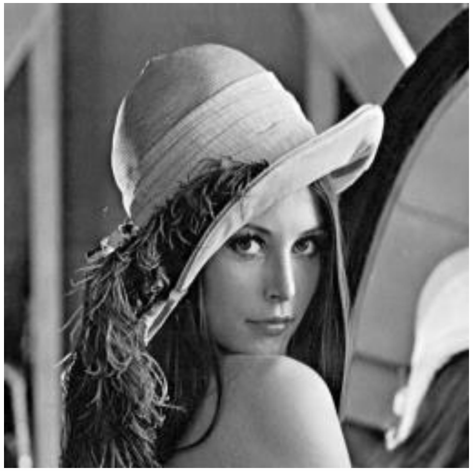


square root

Histogram stretching - results

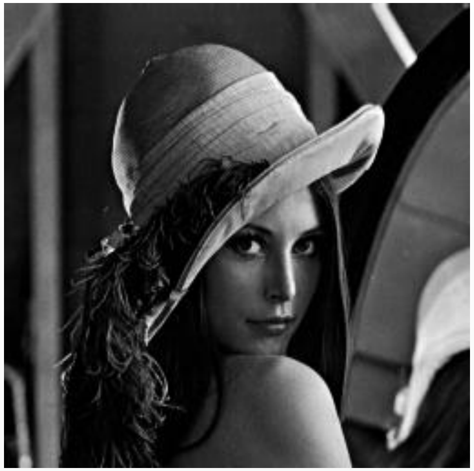
original

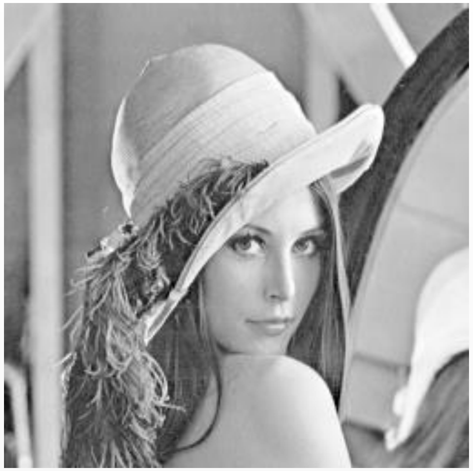




linear f()

quadratic

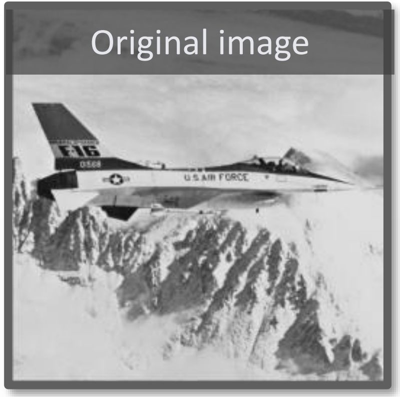




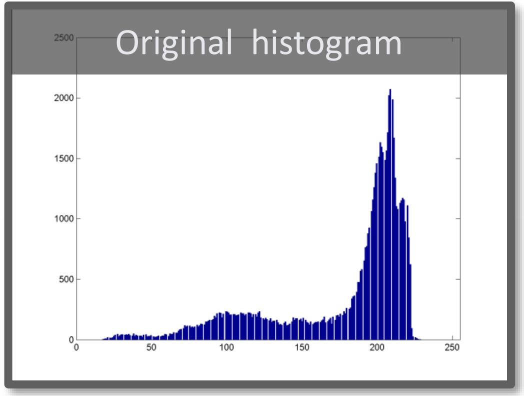
square root

Histogram Transformations

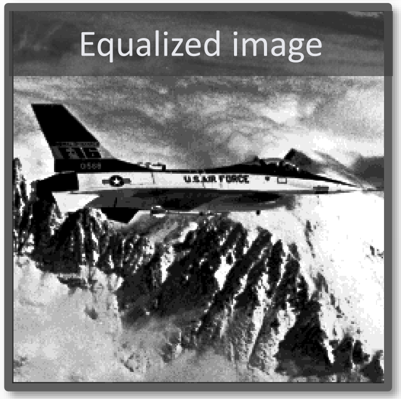
- **Histogram Equalization**:
  - The goal is to increase the contrast, by distributing the occurrences of the intensity values evenly through the entire dynamic range.



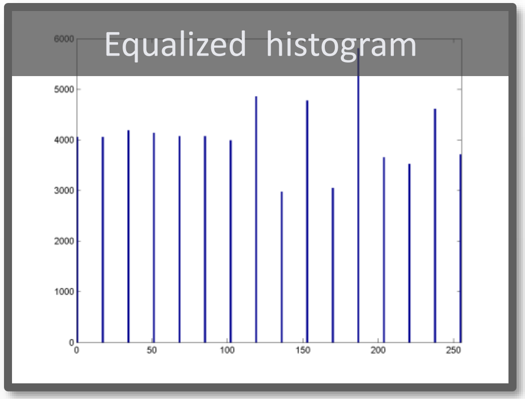
Original image



Original histogram



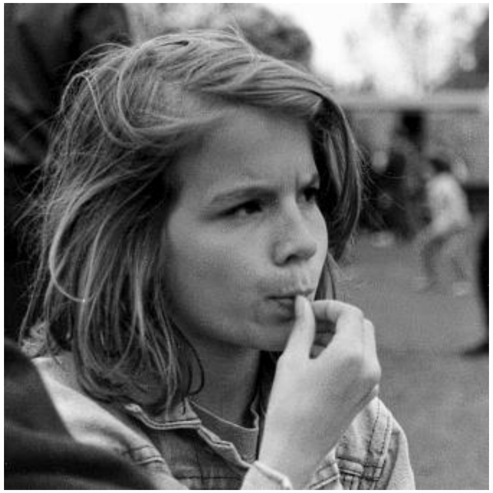
Equalized image

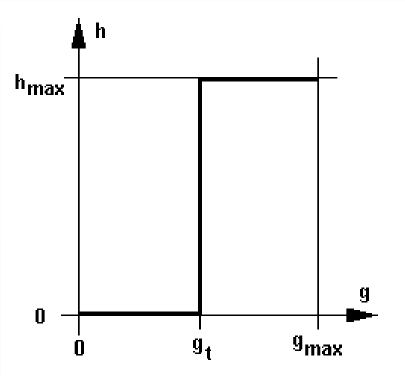


Equalized histogram

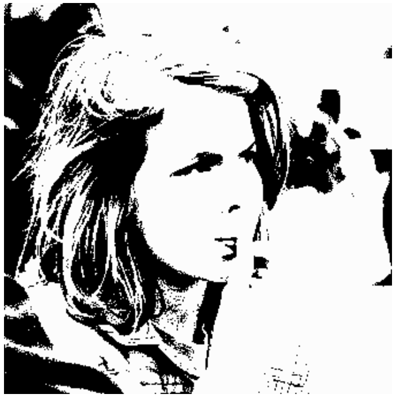
Histogram equalization background

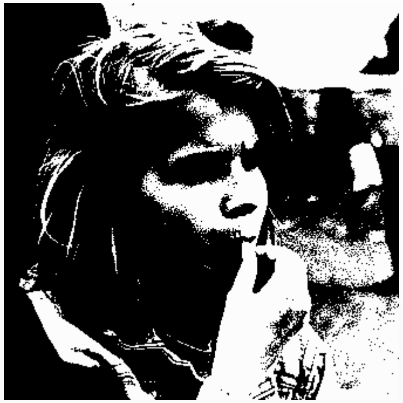
- Simple thresholding

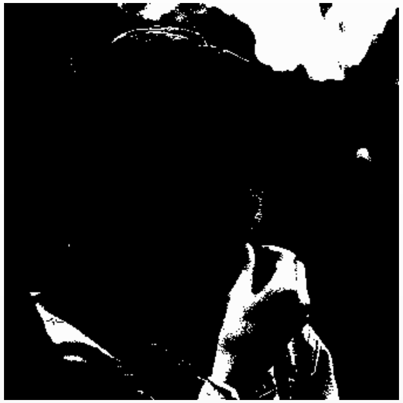




- For different g_t values







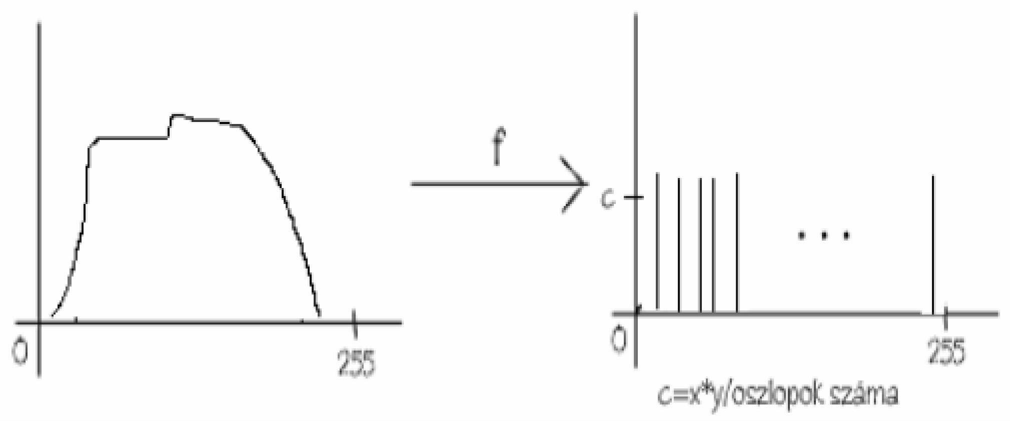
Optimal threshold value

- Task: converting a grayscale image to binary (black&white). What is the optimal threshold value?
  - A possible good solution is to prescribe that the number of black and white pixels should be approximately the same in the output image.
    - The g_t threshold value can be calculated from the histogram, (P is the total number of pixels):

$$\sum_{i=0}^{g_t} h[i] \approx \sum_{i=g_t+1}^{255} h[i] \approx \frac{P}{2}$$

Generalization: histogram equalization

- **Goal**: contrast enhancement
- Transform: step (staircase) function. The number of columns determines number of color (intensity) values appearing in the output, (e.g. number of columns=16, 32, 64, etc.).



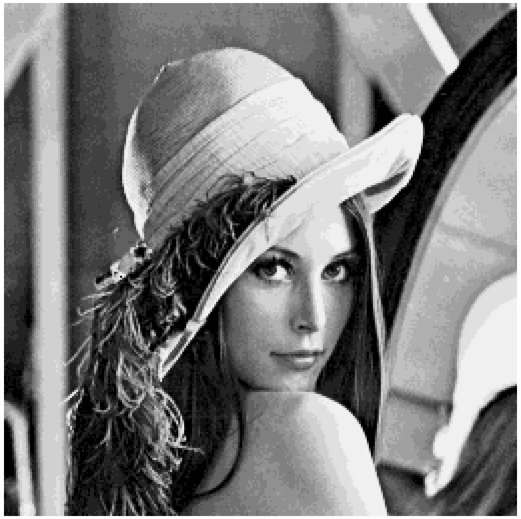
Histogram equalization – c output values

- Goal: determining the t_0=0, t_1, … t_{c-1},t_c=255 dividing points, where:

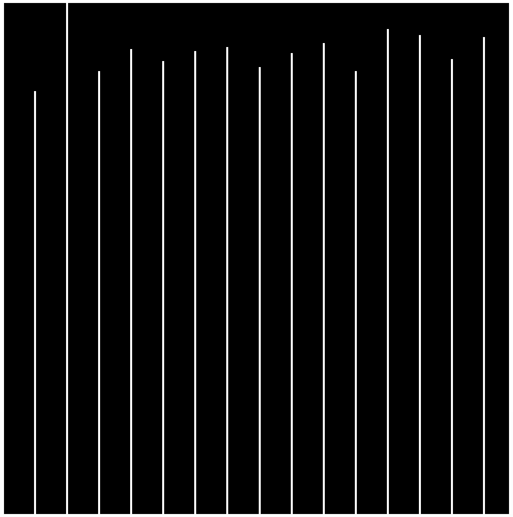
$$\sum_{i=0}^{t_j} h[i] \approx P \cdot \frac{j}{c} \quad j \in \{1 \ldots c\}$$

  - c is the number of different gray levels in the output image (c=2 for thresholding, but it can also be 16, 32, … 256 as well)
  - P is the total number of pixels again.

Histogram equalization - result



16 level ouptut



Histogram of the output image

Histogram Transformations

- Adaptive Histogram Equalization:
  - applies histogram equalization on parts of the image (called tiles) independently
  - Use post processing to reduce artifacts at the borders of the tiles.

Original image

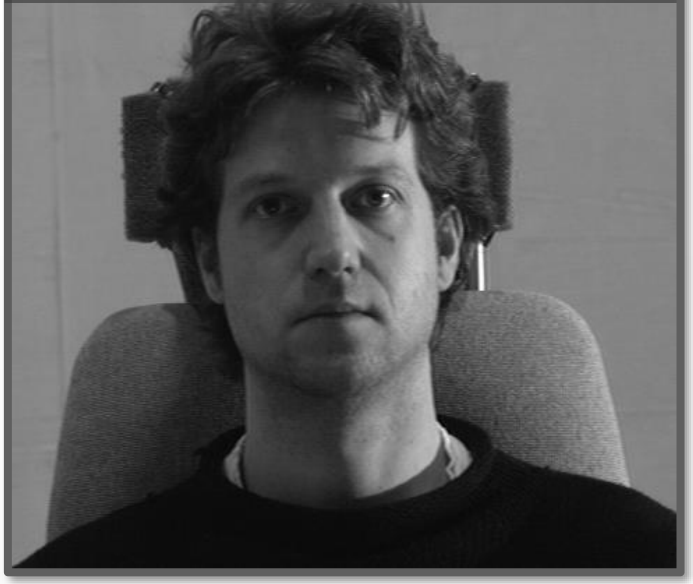
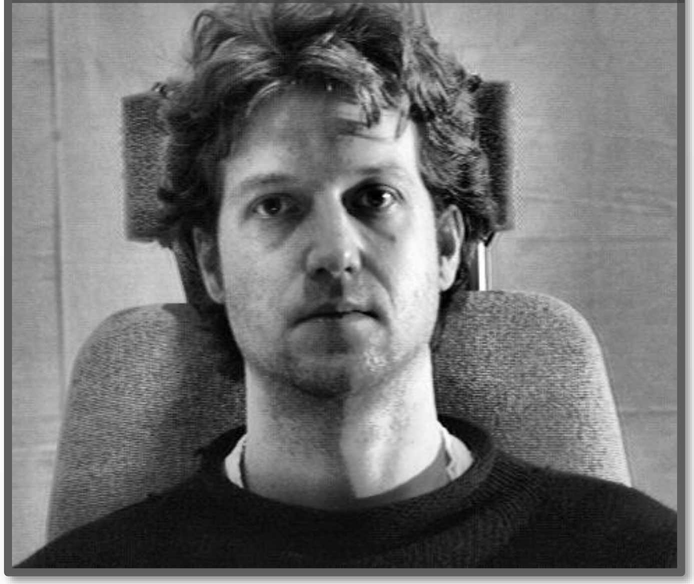


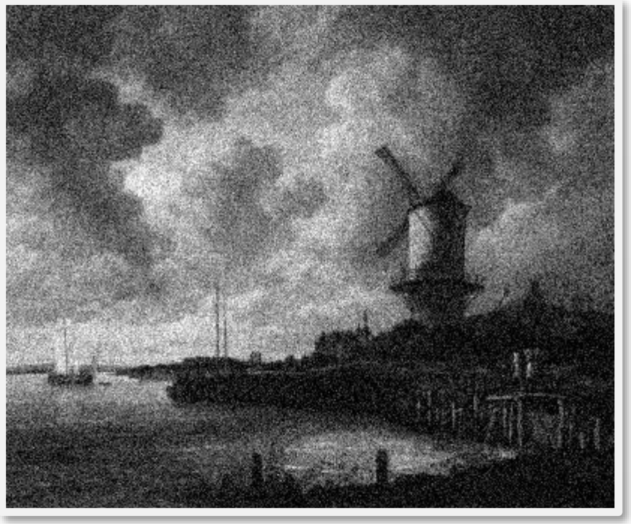
Image after CLAHE



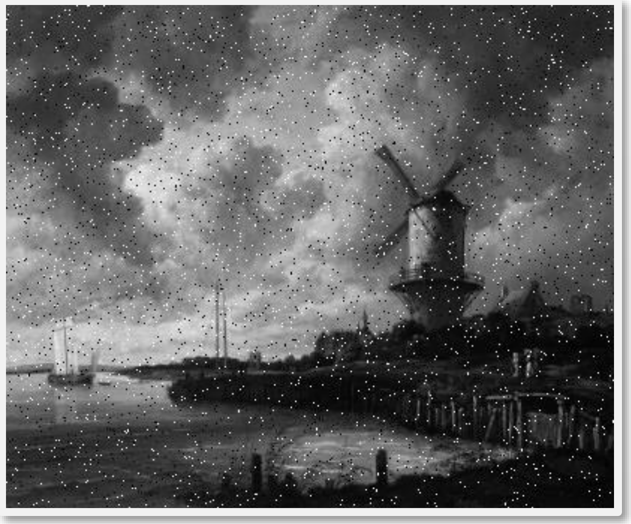
[1] Zuiderveld, Karel. "Contrast Limited Adaptive Histograph Equalization." *Graphic Gems IV*. San Diego: Academic Press Professional, 1994. 474–485.

Spatial Filtering

- **Smoothing:**
  - Reduce the noise that may corrupt the image.
- A few noise types we will work with:
  - Impulse noise, (aka salt and pepper noise)
  - Additive Gaussian Noise



Additive Gaussian Noise



Impulse Noise

The windmill at Wijk bij Duurstede by Jacob van Ruisdael (1670)

Spatial Filtering

- **Gaussian Smoothing:**
  - With *σ=0.75*

Original image

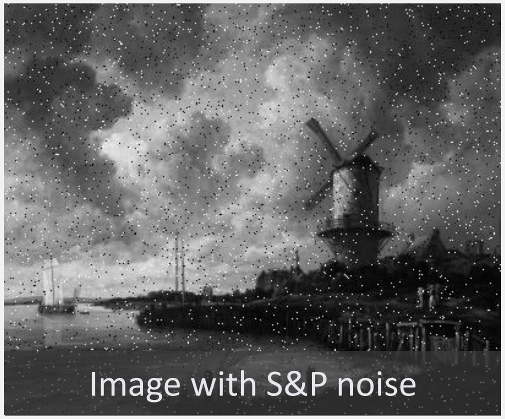
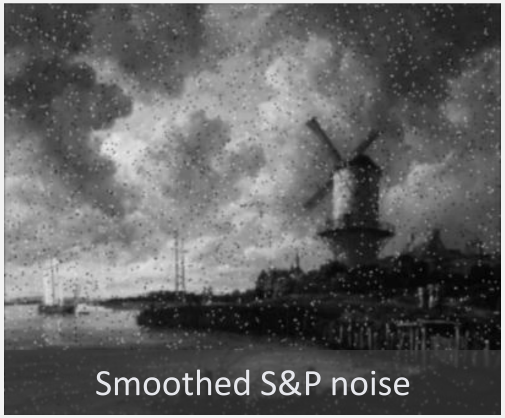


Image with S&P noise



Smoothed S&P noise

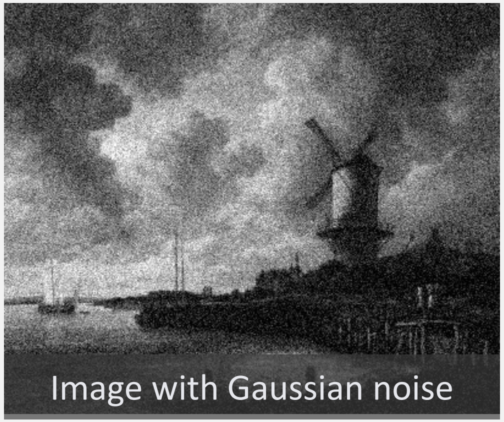
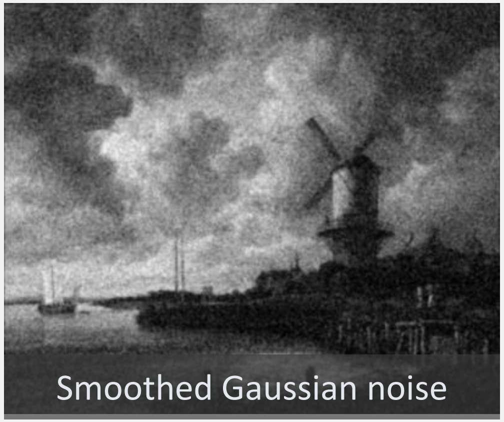


Image with Gaussian noise



Smoothed Gaussian noise

Spatial Filtering

Gaussian Smoothing:

- With *σ=1.5*

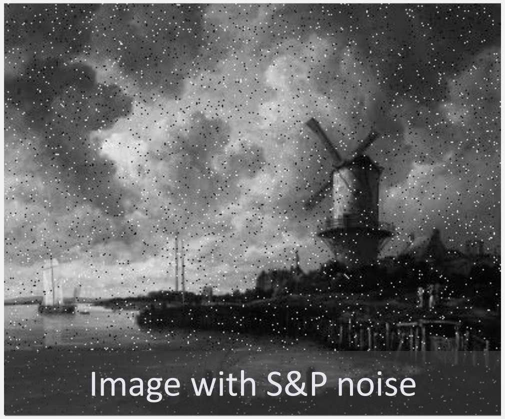
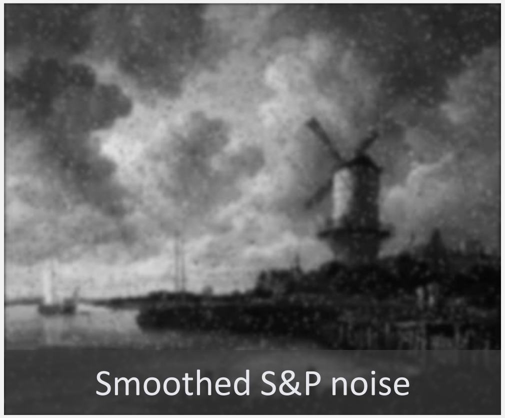


Image with S&P noise



Smoothed S&P noise

Original image

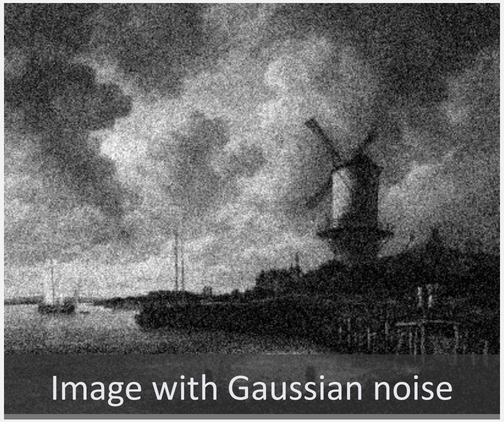
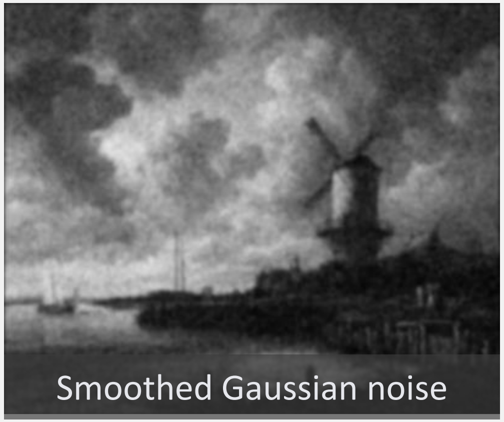


Image with Gaussian noise



Smoothed Gaussian noise

Spatial Filtering

Spatially Adaptive Noise Smoothing:

- The smoothing takes into account the local characteristics of the image:

$$y(n_1, n_2) = \left(1 - \frac{\boxed{\sigma_n^2}}{\boxed{\sigma_l^2}}\right) \cdot x(n_1, n_2) + \frac{\boxed{\sigma_n^2}}{\boxed{\sigma_l^2}} \cdot \boxed{\bar{x}(n_1, n_2)}$$

$$\boxed{\sigma_l^2(n_1,n_2) = \sum \sum_{(n_1,n_2)\in N} \left(x(n_1,n_2) - \bar{x}(n_1,n_2)\right)^2}$$

Local variance of the image

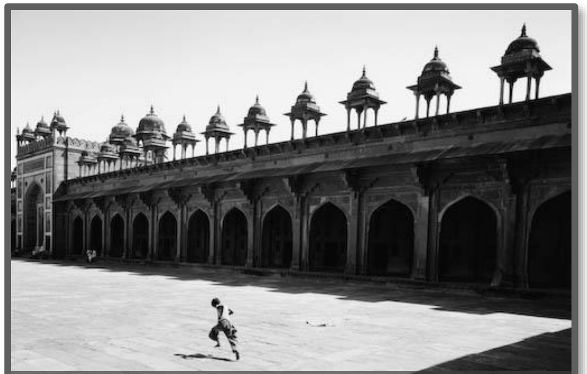
$$\boxed{\bar{x}(n_1, n_2) = \frac{1}{|N|} \sum \sum_{(n_1,n_2)\in N} x(n_1, n_2)}$$

Local average of the image

Variance of the noise: either known a priori, or has to be measured

Spatial Filtering

Spatially Adaptive Noise Smoothing:



Original image*

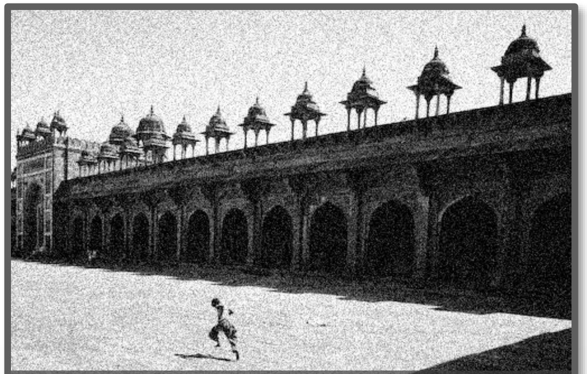
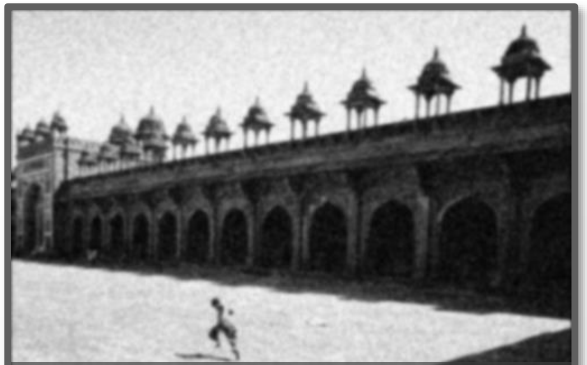
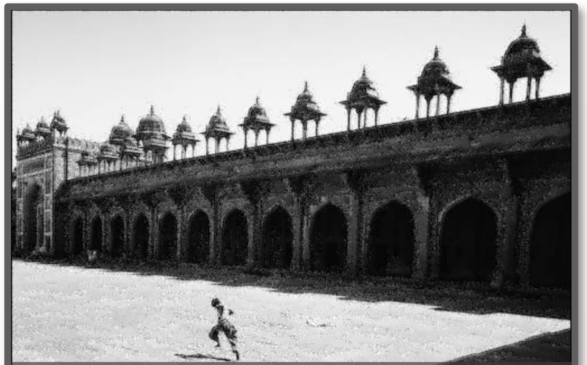


Image with Gaussian Noise



Gaussian Smoothing



Spatially Adaptive Smoothing

* Fatepuhr Sikri, Inde by Lucien Hervé (1955)

Basic Image Processing

PPKE-ITK

Lecture 4.

Spatial Filtering

- **Recap: Gaussian Smoothing – efficient for Gaussian noise, but…**

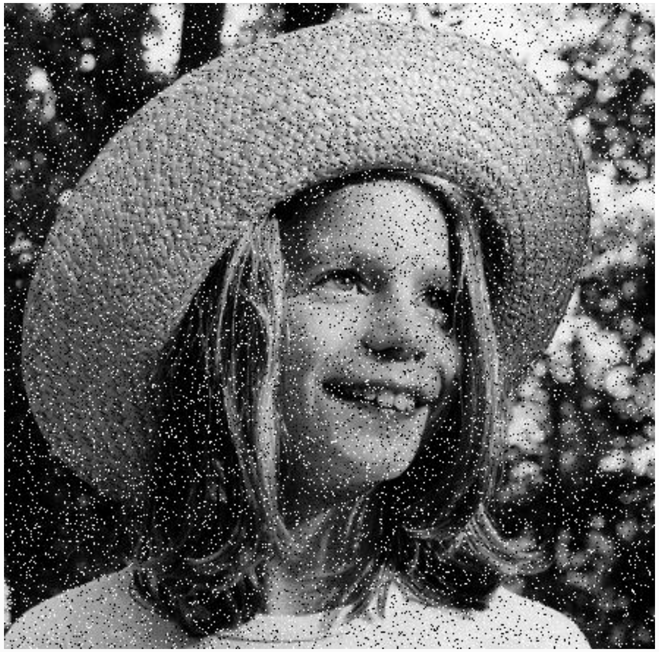
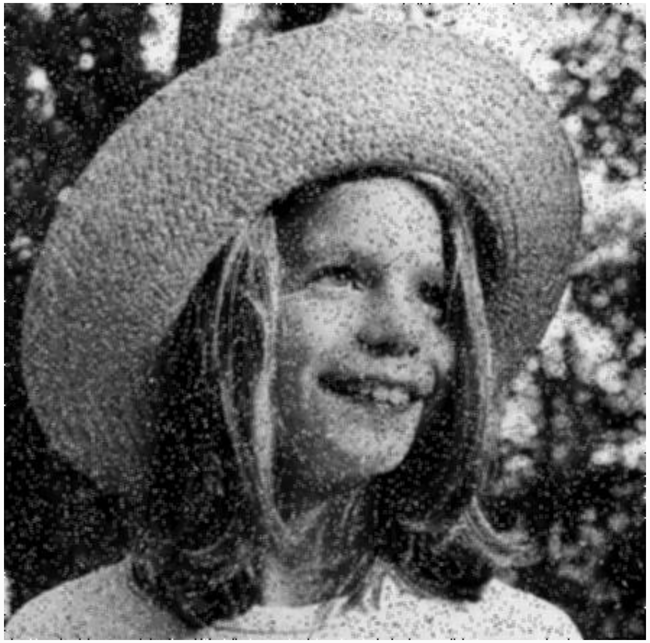


Image with salt and pepper noise



Gaussian blur with convolution

Image credit ® Prof. Vladimir Székely, BME

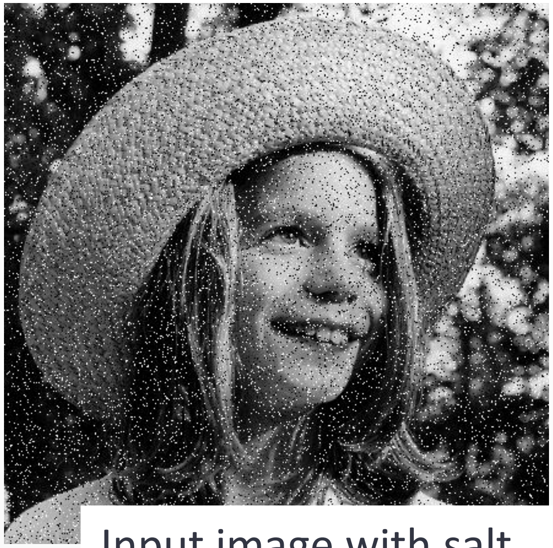
Spatial Filtering

- **Rank filter:**
  1. Consider the actual pixel and its neighborhood (e.g. 3×3=9 pixel sized window),
  2. Sort the observed pixel values according to gray level,
  3. Take the *k-th* value from this row as the new pixel value
- **Median filter**: k is the middle pixel value in the row:

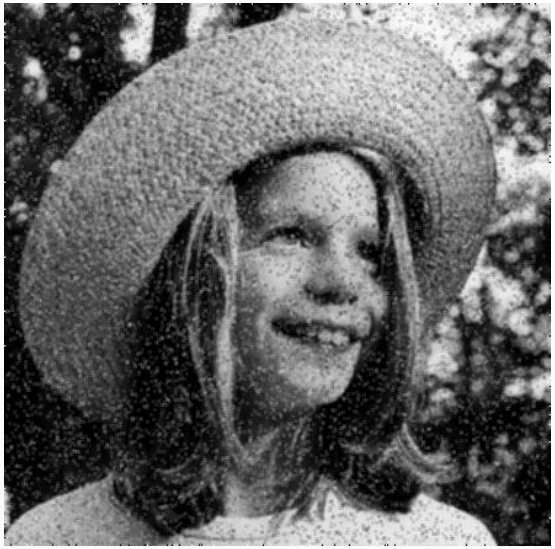
  $k=[(2W+1)^2-1]/2$, if W is the half side size of the neighborhood
- Non-Linear filter

Spatial Filtering

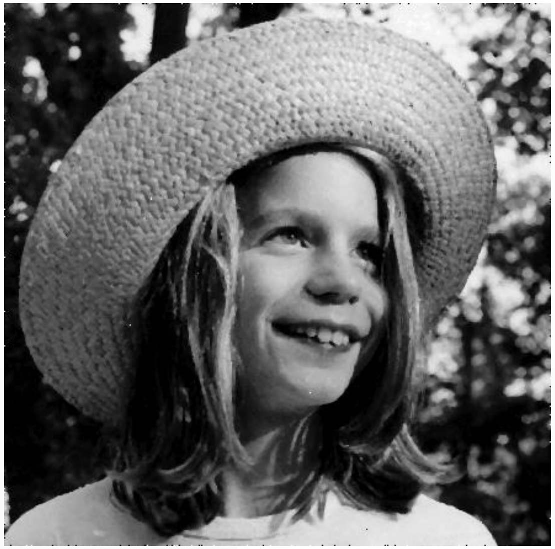
- **Median filter:** replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
  - Very effective against impulse („salt and pepper”) noise:



Input image with salt and pepper noise



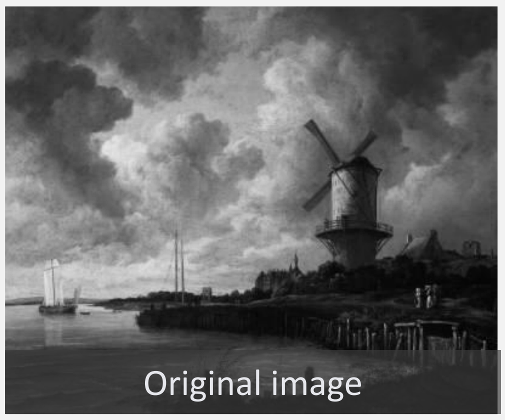
Blur with convolution



Median filter

Spatial Filtering

- **Median filter:** replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
  - Very effective against impulse („salt and pepper”) noise:



Original image

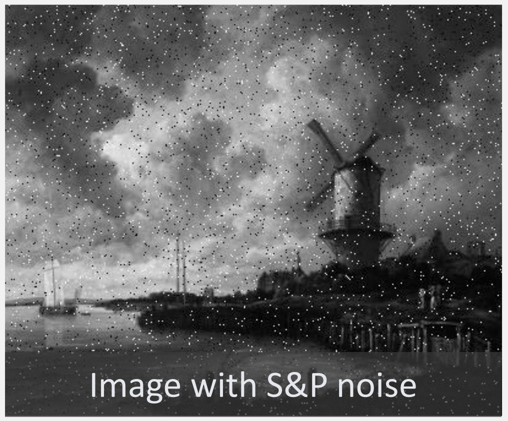
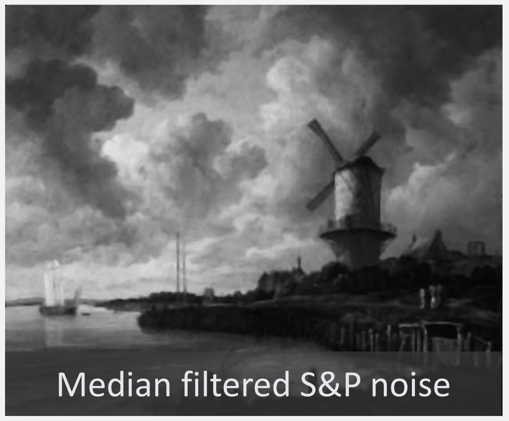


Image with S&P noise



Median filtered S&P noise

  - Not so effective against Gaussian noise.

Spatial Filtering

⦿ Order statistic filtering:

- Based on the sorted pixel intensity levels in the analyzed neighborhood.
- If after sorting…
  - we take the middle element, we get back the median filter.
  - We take the maximum element to filter „pepper” and min to filter „salt” noise.

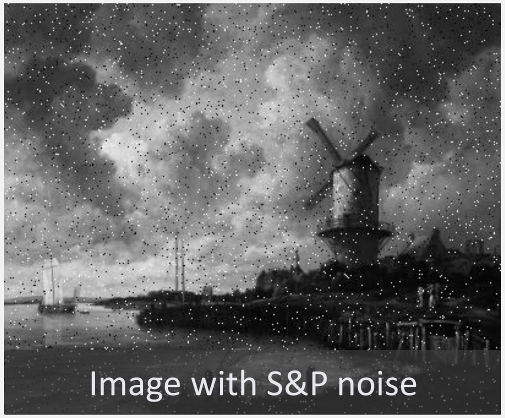
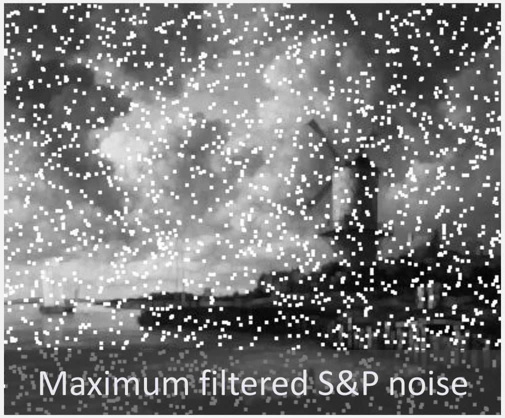
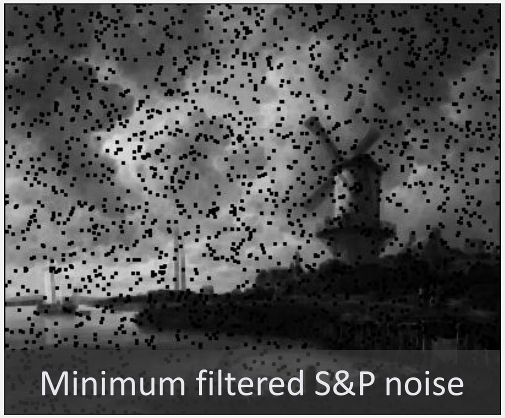


Image with S&P noise


Maximum filtered S&P noise


Minimum filtered S&P noise

- But max filter will highlight „salt”, while min filter will highlight „pepper”.

Spatial Filtering

⊙ Order statistic filtering:

- Mid-point filtering:
  - works well on Gaussian or uniform noise

$$\mathrm{y}(\mathrm{n}_1,\mathrm{n}_2) = \frac{1}{2}\left(\max_{(m_1,m_2)\in N}\{x(m_1,m_2)\} + \min_{(m_1,m_2)\in N}\{x(m_1,m_2)\}\right)$$

- Alpha-trimmed mean filter:

$$\mathrm{y}(\mathrm{n}_1,\mathrm{n}_2) = \frac{1}{|\mathrm{N}| - \alpha}\sum_{(m_1,m_2)\in N_r} x(m_1,m_2)$$

  - Where N_r is a reduced neighborhood, not containing the lowest and highest α element of N.
  - If α = 0, we get back the arithmetic mean.
  - If α = |N|-1, we get back the median filter.

Wallis Operator

- The Wallis operator can help to adjust local contrast:

$$y(n_1,n_2)=[x(n_1,n_2)-\bar{x}(n_1,n_2)]\frac{A_{\max}\sigma_d}{A_{\max}\sigma_l(n_1,n_2)+\sigma_d}$$
$$+[p\bar{x}_d+(1-p)\bar{x}(n_1,n_2)]$$

- where σ_l the local contrast: $\sigma_l(n_1,n_2)=\frac{1}{|N|}\sqrt{\sum\sum_{(n_1,n_2)\in N}(x(n_1,n_2)-\bar{x}(n_1,n_2))^2}$
- $\bar{x}$ is the local average: $\bar{x}(n_1,n_2)=\frac{1}{|N|}\sum\sum_{(n_1,n_2)\in N}x(n_1,n_2)$
- σ_d is the desired local contrast, $\bar{x}_d$ is the desired mean value of all pixels, p is a weighting factor of the mean compensation, while A_{max} is maximizing the local contrast modification.

Wallis Operator

- We can describe the image the following way:

$$x(n_1,n_2)=\underbrace{\left[x(n_1,n_2)-\bar{x}(n_1,n_2)\right]}_{(1)}+\underbrace{\bar{x}(n_1,n_2)}_{(2)}$$

where (2) is the local mean and (1) is the deviation from the local mean.

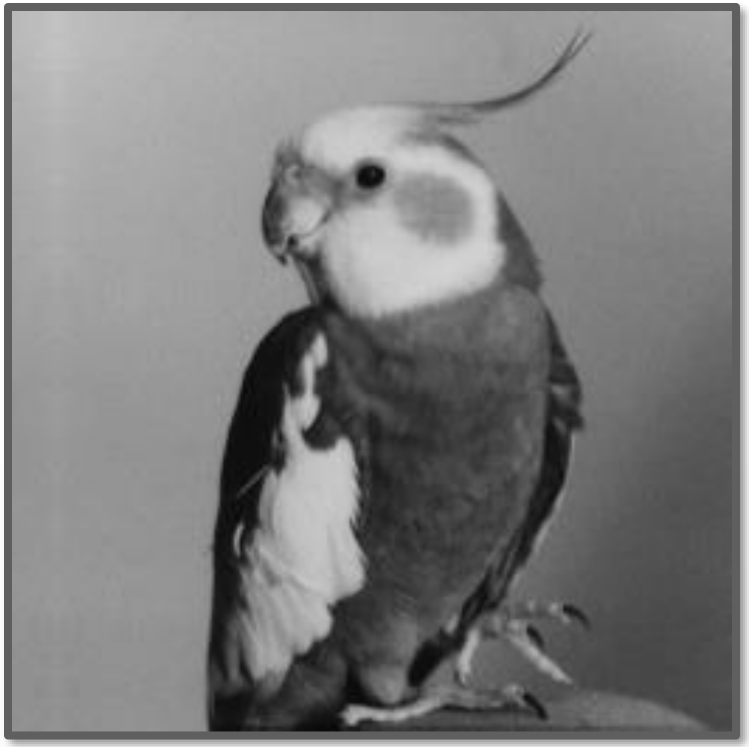
- With the transformation we want to „push” the local mean and standard deviation to a predefined desired value:

$$y(n_1,n_2)=\underbrace{\left[x(n_1,n_2)-\bar{x}(n_1,n_2)\right]\frac{\sigma_d}{\sigma_l(n_1,n_2)}}_{(1)}+\underbrace{\left[p\bar{x}_d+(1-p)\bar{x}(n_1,n_2)\right]}_{(2)}$$

- We are almost there, but if the local contrast is too low, the weighting in (1) may get too high, this is why we maximize it with A_{max}:

$$\frac{\sigma_d}{\sigma_l(n_1,n_2)}\Rightarrow\frac{A_{\max}\sigma_d}{A_{\max}\sigma_l(n_1,n_2)+\sigma_d}$$

Wallis Operator



Original Image

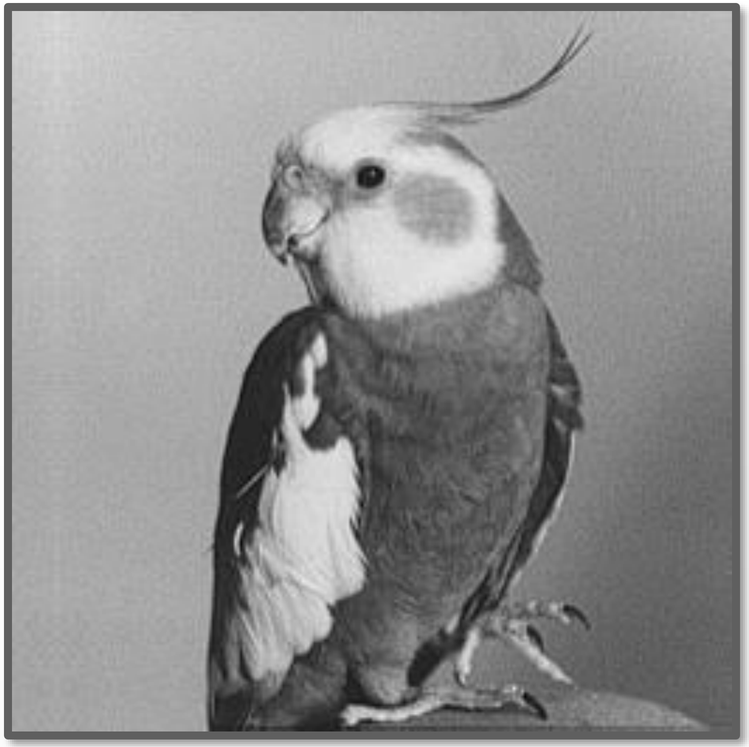


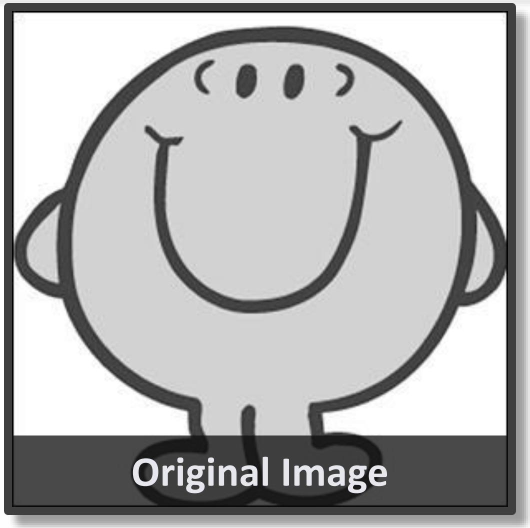
Image after applying Wallis operator

Anisotropic Diffusion

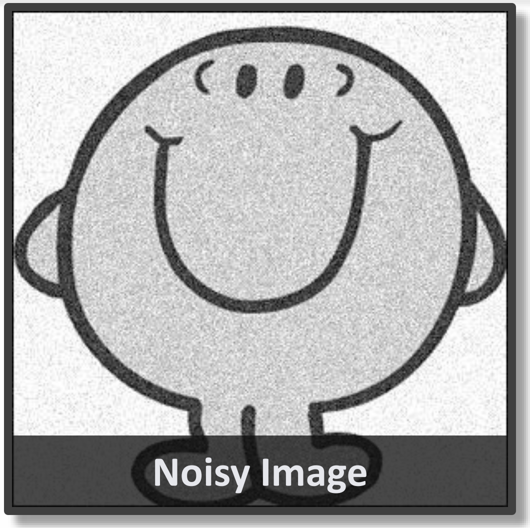
- The anisotropic diffusion is a technique aiming at reducing image noise without blurring significant parts of the image content.
- It was first proposed by Dénes Gábor in 1965 and later by Perona and Malik around 1990.
- ***Non-linear*** and ***space-variant*** transformation.
- The main idea is that the effect of blurring in each direction is inversely proportional to the gradient value in that direction:
  - allows diffusion along the edges or in edge-free territories, but penalizes diffusion orthogonal to the edge direction.
- AD is an iterative process

P. Perona, J Malik (July 1990). "Scale-space and edge detection using anisotropic diffusion". IEEE Tr. PAMI, 12 (7): 629–639.
D. Gabor, “Information theory in electron microscopy,” Laboratory Investigation, vol. 14/6, pp. 801–807, 1965.

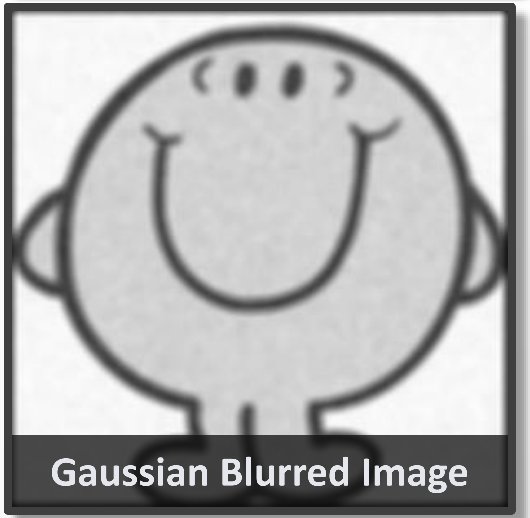
Anisotropic Diffusion



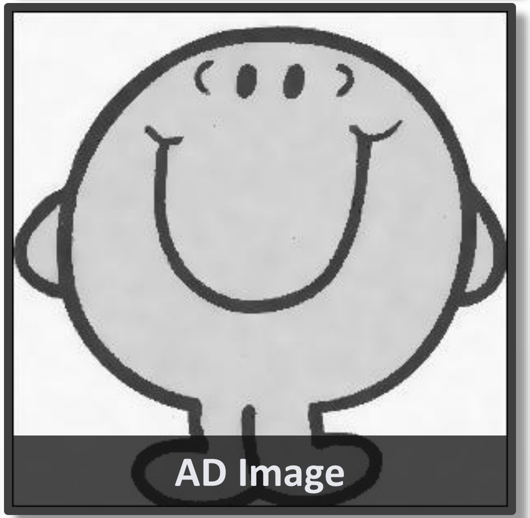
Original Image



Noisy Image

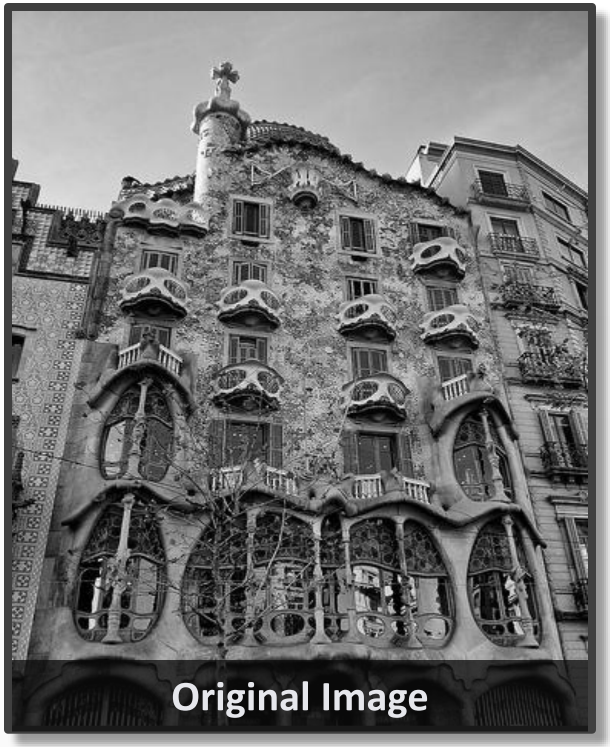


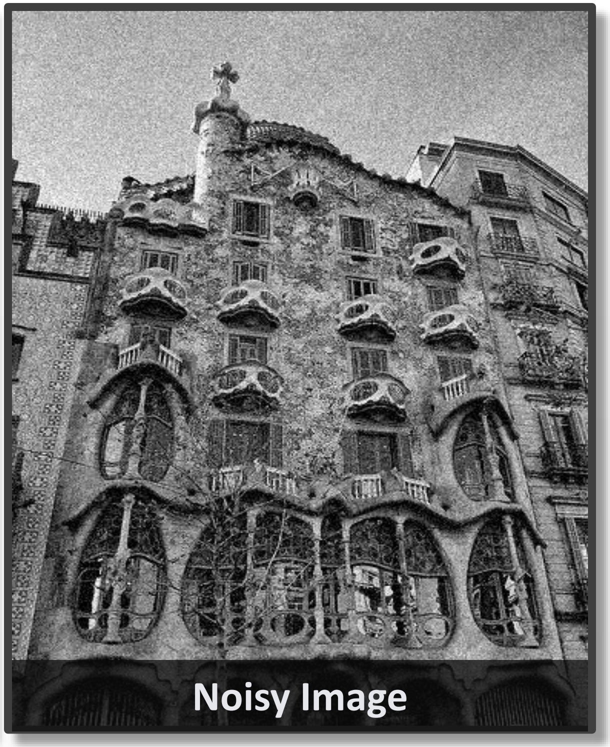
Gaussian Blurred Image

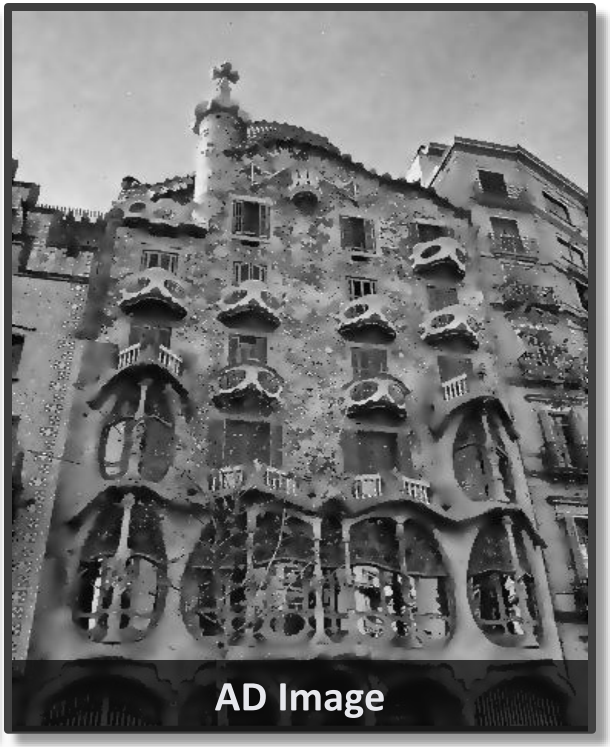


AD Image

Anisotropic Diffusion


Original Image


Noisy Image


AD Image

Total Variation Regularization

- Assumption:
  - The image is smooth inside the objects, with jumps across the boundaries.
  - The noise component has high variation.
- The goal of ***Total Variation based noise removal*** is to minimize the total variation of the image while keep the result as close to the original input image as possible.
- It was introduced by Rudin, Osher and Fatemi in 1992.

Rudin, L. I.; Osher, S.; Fatemi, E. (1992). "Nonlinear total variation based noise removal algorithms". *Physica D* **60**: 259–268

Total Variation Regularization

- TV of the output image y is defined as the integral of the absolute gradient of the signal:

$$\mathrm{V}(y) = \sum_{n_1}\sum_{n_2}\sqrt{\left|y(n_1+1, n_2) - y(n_1, n_2)\right|^2 + \left|y(n_1, n_2+1) - y(n_1, n_2)\right|^2}$$

- On the other hand, we also measure the difference between the original image x and the output image y by L_2 norm E:

$$\mathrm{E}(\mathrm{x}, \mathrm{y}) = \sum_{n_1, n_2}\left(x(n_1, n_2) - y(n_1, n_2)\right)^2$$

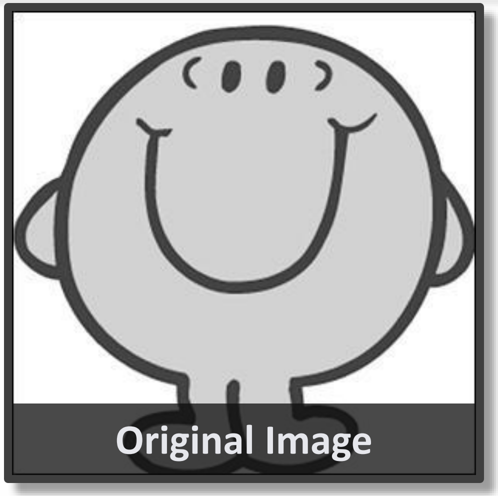
- The goal function for Total Variation based regularization:

$$\hat{\mathrm{y}} = \arg\min_{y}\left[\mathrm{E}(\mathrm{x}, \mathrm{y}) + \lambda \mathrm{V}(y)\right]$$

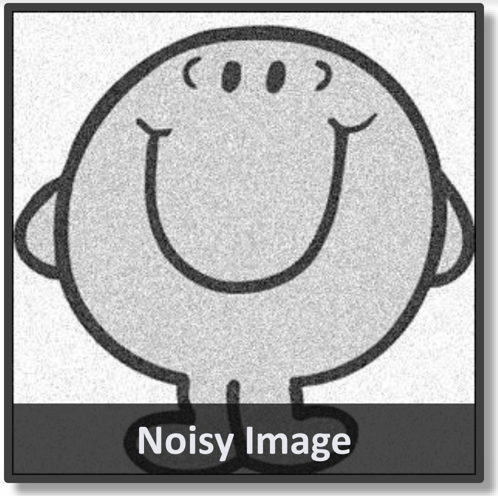
where λ is the regularization parameter.

Rudin, L. I.; Osher, S.; Fatemi, E. (1992). "Nonlinear total variation based noise removal algorithms". *Physica D* **60**: 259–268

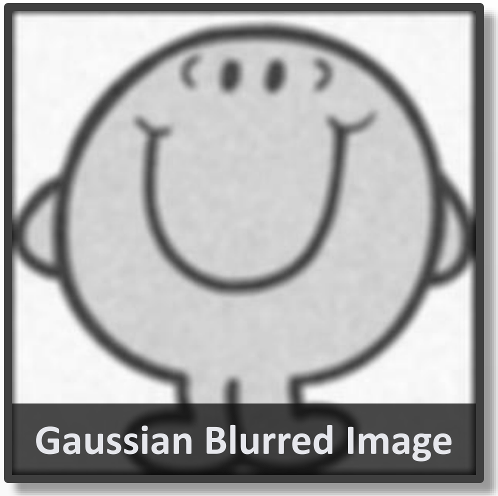
Total Variation Regularization



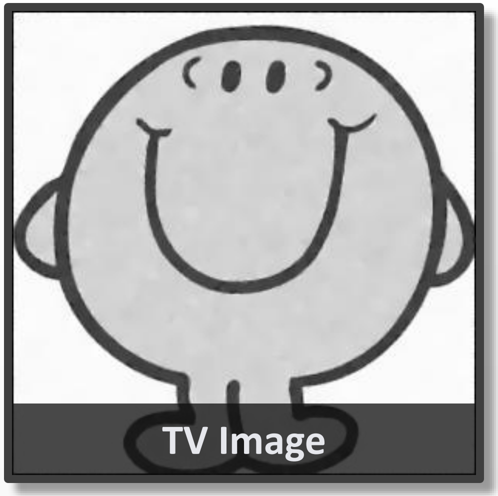
Original Image



Noisy Image



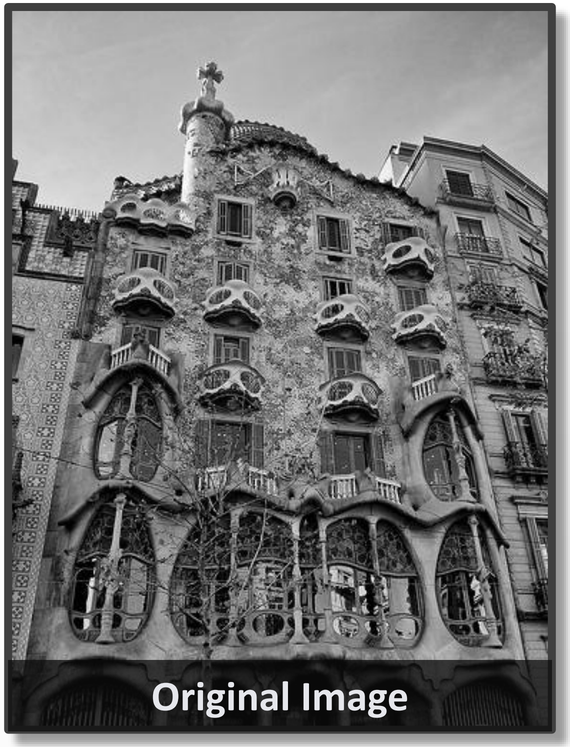
Gaussian Blurred Image



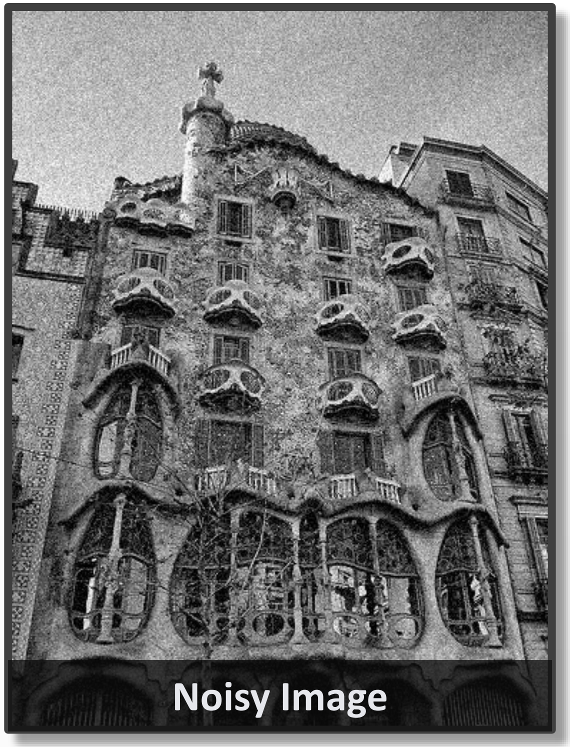
TV Image

Matlab Code: http://visl.technion.ac.il/~gilboa/PDE-filt/tv_denoising.html

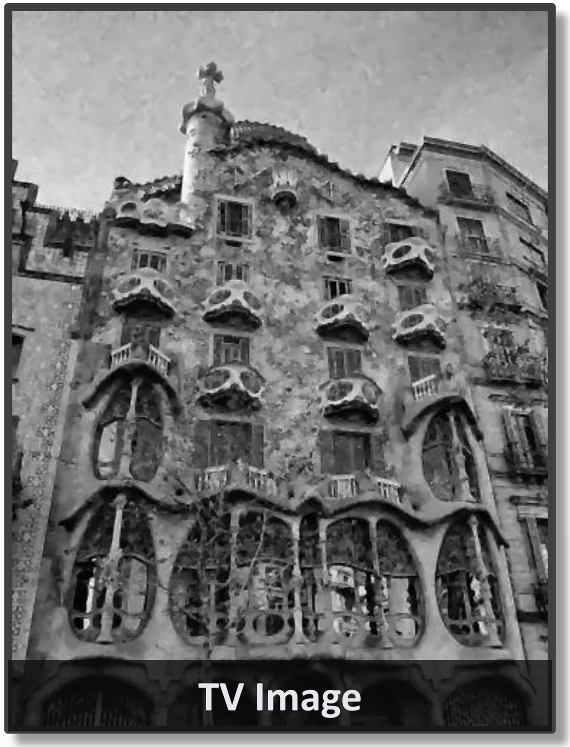
Total Variation Regularization



Original Image



Noisy Image



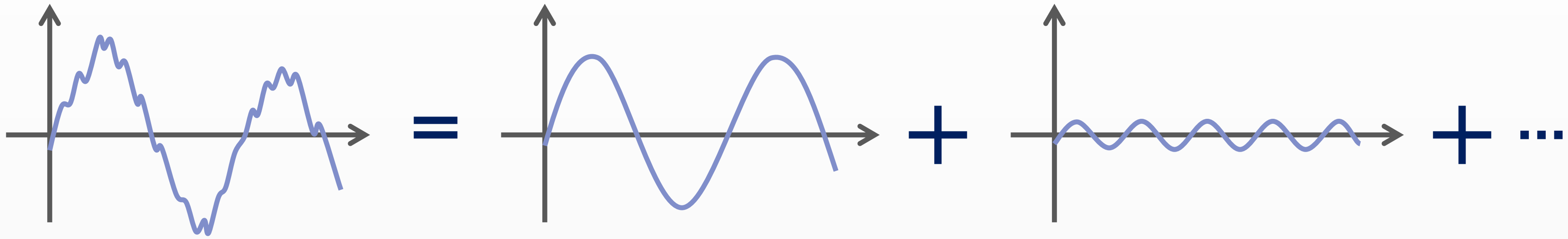
TV Image

Matlab Code: http://visl.technion.ac.il/~gilboa/PDE-filt/tv_denoising.html

Fourier Transformation
Image filtering in the frequency domain

What is Fourier Transform?

- A transformation maps data between (different) domains.
- The Fourier Transform changes between the representation in the **time domain** and in the **frequency domain**.
- The information is the same in both domains, only the representation is different.
- It is a reversible transform.
- It builds on the fact that any function can be represented as a weighted sum of sinusoid functions:



- If we can describe sinusoids we can describe every function.

2D Fourier transform

- Forward transform: map an image $x(n_1, n_2)$ of size $N_1 \times N_2$ from the **spatial domain** into the $X(\omega_1, \omega_2)$ **frequency domain**

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

  - even if the image is real the spectrum is complex due to the complex exponential factors
  - ω_1, ω_2 frequencies: continuous variables
  - $X(\omega_1, \omega_2)$ continuous Fourier transform or spectrum of the discrete image
  - Drawback: no computable representation of $X(\omega_1, \omega_2)$
  - Solution: **Discrete Fourier Transform (DFT)**: sample the continuous spectrum with equally spaced frequencies

Discrete Fourier Transform

- We sample one period of the Fourier transform in evenly spaced frequencies:

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

The size of the image in the spatial domain is N$_1$xN$_2$

The size of the image in the frequency domain will be the same: N$_1$xN$_2$

$$X(k_1, k_2) = X(\omega_1, \omega_2)\Big|_{\omega_1 = \frac{2\pi}{N_1}k_1, \omega_2 = \frac{2\pi}{N_2}k_2}$$

$$k_1 = 0,1..., N_1 - 1$$
$$k_2 = 0,1..., N_2 - 1$$

Only one period is kept

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N_1}k_1 n_1} e^{-j\frac{2\pi}{N_2}k_2 n_2}$$

Discrete Fourier Transform

- Forward formula: gives the description of the image in the discrete frequency domain

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N_1}k_1 n_1} e^{-j\frac{2\pi}{N_2}k_2 n_2}$$

- Inverse Fourier transform: maps from the discrete frequency domain back to the discrete spatial domain

$$x(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) e^{j\frac{2\pi}{N_1}k_1 n_1} e^{j\frac{2\pi}{N_2}k_2 n_2}$$

  - algorithmically it has the same structure as the forward transform,

Discrete Fourier Transform

- DFT is an exact transform, there is no transformation error.
  - Not surprising, since we use the same image size for the representation of both $x(n_1, n_2)$ and $X(\omega_1, \omega_2)$.
- Most of the properties of continuous FT hold for DFT
  - Except linear shift of FT becomes circular shift for DFT.
- DFT and inverse DFT are computable transformations
- There are fast ways to compute the DFT: ***Fast Fourier Transform***
  - If the size of the image is *NxN*, then the **naive implementation** requires N^4 multiplications: N^2 for each (k_1,k_2) point.
  - The **FFT** with row/column decomposition requires only $N^2 \log_2 N$ multiplications.
- **FFT makes the Fourier transformation applicable in many practical cases.**

Interpretation of Fourier coefficients

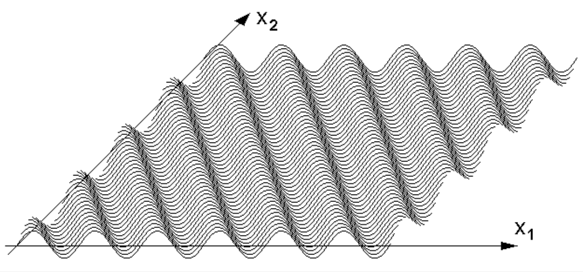
- Analogy of Fourier coefficient based representation:
  - Consider the image as a superposition of sinusoid/cosine waves with different amplitudes, frequencies and directions
  - 1D case in formulas: (x_n: signal, X_k Fourier coefficient)

$$x_n = \sum_{k=0}^{N-1} X_k \exp(j \frac{2\pi}{N} k\, n)$$

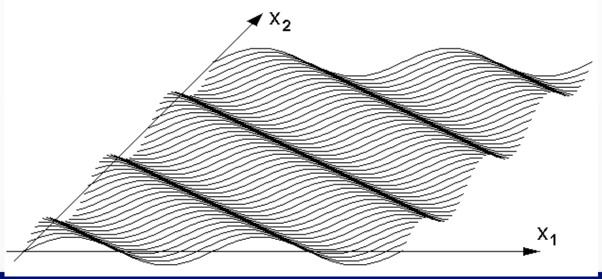
$$x_n = X_0 + X_{N/2}\cos(\pi n) + \sum_{k=1}^{N/2-1} 2\left(\mathrm{Re}(X_k)\cos\left(\frac{2\pi}{N}kn\right) - \mathrm{Im}(X_k)\sin\left(\frac{2\pi}{N}kn\right)\right)$$

X_0: real number, average of the function

  - 2D case (e.g. image) visualization:



+

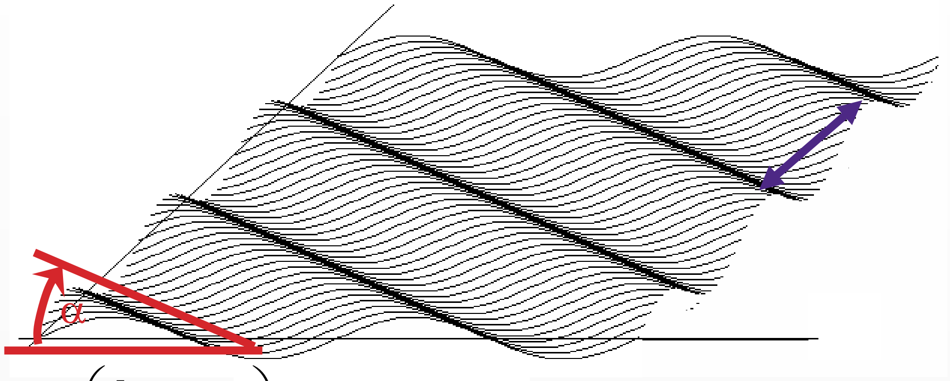


Interpretation of 2D Fourier coefficients

$$X(k_1,k_2)\cdot\exp\left(j2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)+X(-k_1,-k_2)\cdot\exp\left(-j2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)=$$

$$\mathrm{Re}\{X(k_1,k_2)\}\cdot 2\cdot\cos\left(2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)-\mathrm{Im}\{X(k_1,k_2)\}\cdot 2\cdot\sin\left(2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)$$

- real part of an $X(k_1,k_2)$ Fourier coefficient is the **amplitude** of a cos-wave, while the imaginary part is the amplitude of a sinusoid wave
- **wavelength** and **orientation** of the waves are encoded in the (k_1,k_2) *position* coordinates of the coefficients in the 2D Fourier map



orientation:

$$\alpha=\arctan\left(\frac{k_2/N_2}{k_1/N_1}\right)$$

k_1=3 k_2=2

wavelength:

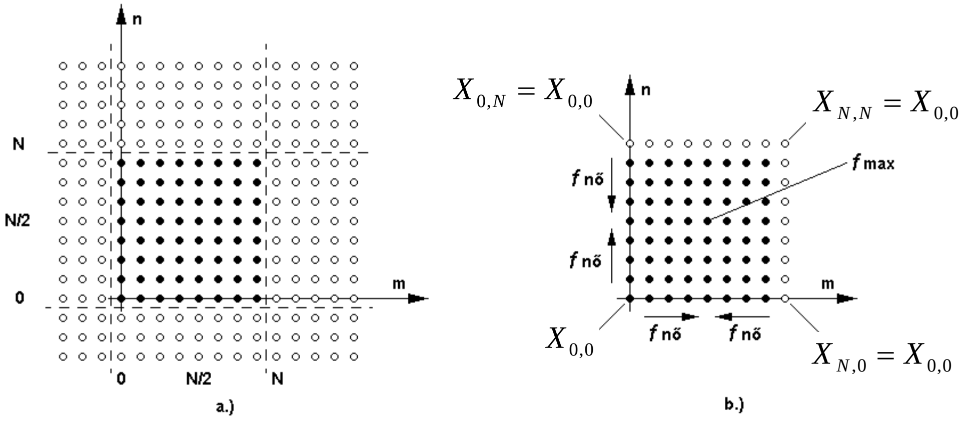
$$\frac{1}{\lambda}=\sqrt{\left(\frac{k_1}{N_1}\right)^2+\left(\frac{k_2}{N_2}\right)^2}$$

Spatial frequency:

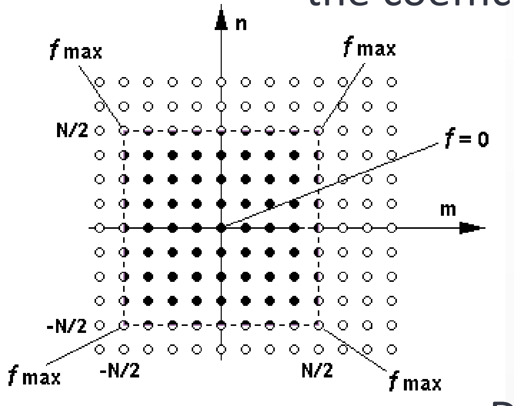
$$\boxed{f\ =1/\lambda}$$

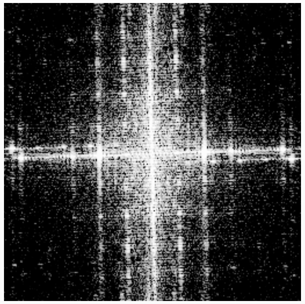
Slide credit ® Prof. Vladimir Székely, BME

Illustration of the periodicity of coefficients
Centered DFT: better visualization



Re-arrangement of the coefficient matrix





DFT abs. value image

- In the center of the DFT array X_{00} is the *zero-frequency coefficient (DC component)*
- Distance from the center
  - Frequency of the corresponding sin/cos wave

$$f = \sqrt{\left(k_1 / N_1\right)^2 + \left(k_2 / N_2\right)^2}$$

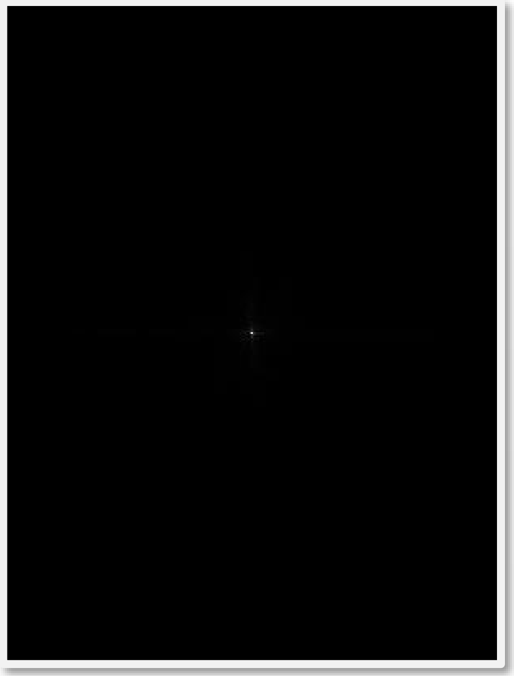
- Orientation
  - Direction perpendicular to the wavefront

$$\alpha = \arctan\left(\left(k_2 / N_2\right) / \left(k_1 / N_1\right)\right)$$

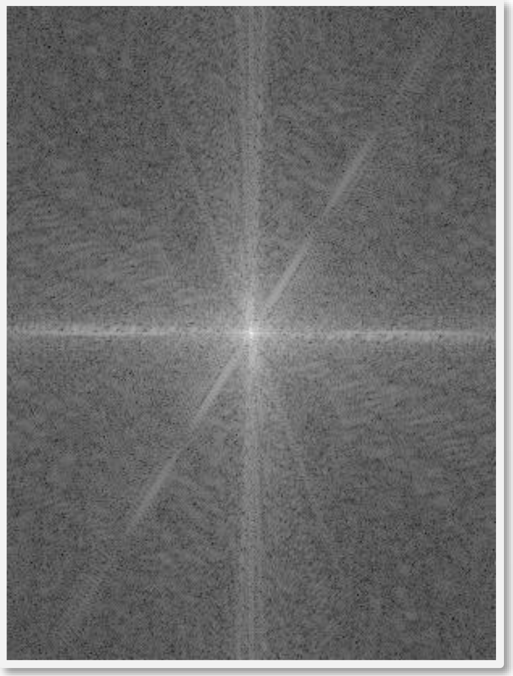
Point-wise Intensity Transformation

- **Log transformation**: $y(n_1, n_2) = c \cdot \log\left(x(n_1, n_2) + 1\right)$
  - Commonly used to visualize the Fourier transform of an image

Original Image*



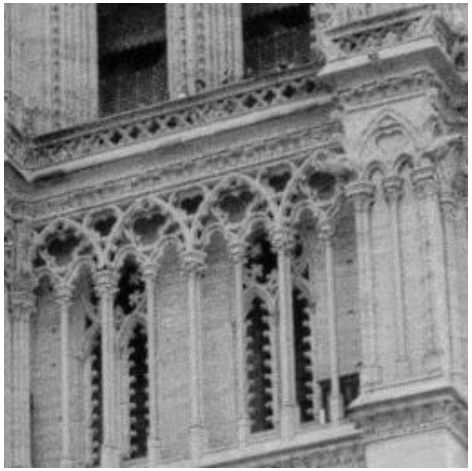
The magnitude of the DFT

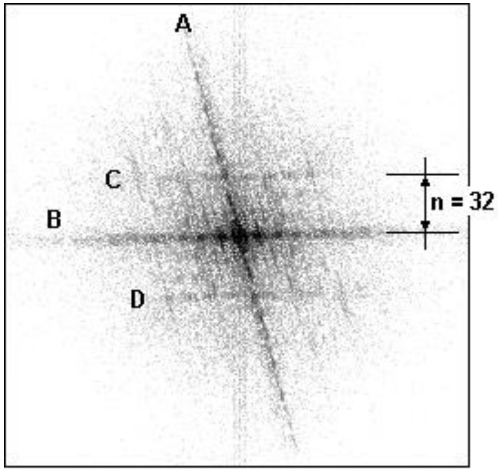


Log of the magnitude of the DFT

*Chez Mondrian by András Kertész (1926)

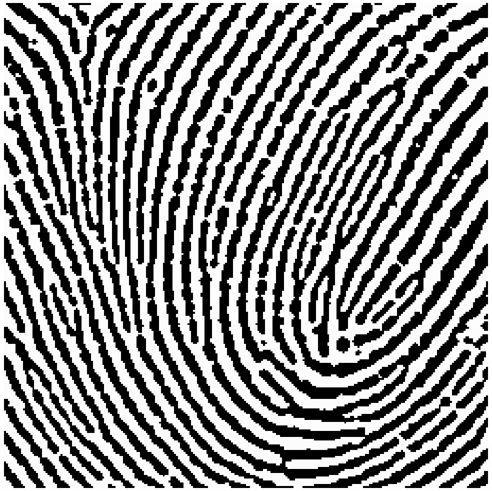
Imaged absolute values of DFT coefficients – facade of the Notre Dame Paris

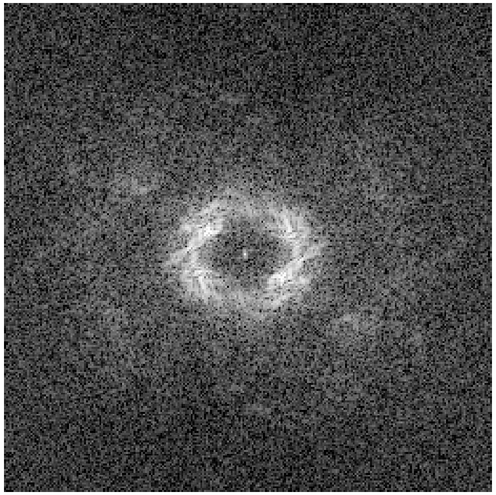




- In the transformed map, directions of strong lines are perpendicular to the major contours in the image:
  - **A** line- horizontal ledges (párkányok)
  - **B** line- slim vertical columns.
  - **C** and **D** lines – periodic vertical patterns with the frequency „*n* = ±32” : decoration of the windows behind the columns

Imaged absolute values of DFT coefficients– analysing fingerprint images





- No characteristic lines in the transform

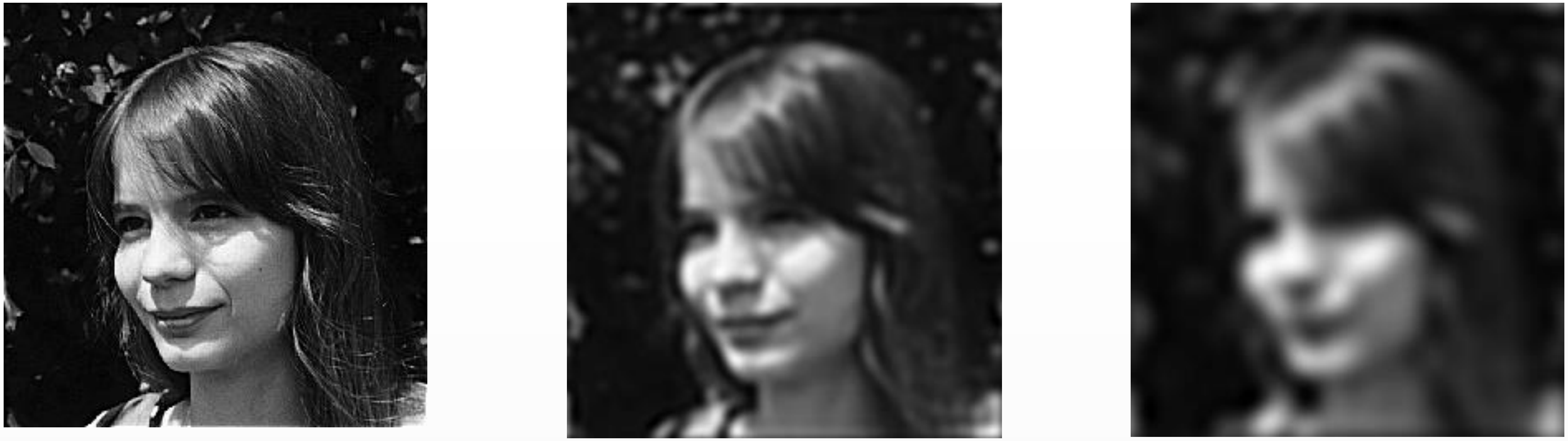
  ← ridges of fingerprints run in any directions

- At *d* distance from the center a significant ring shaped maximum

  ← the average spatial frequency of the fingerprint ridges is *d*-times the basic frequency f_b, and there exist no nominant directions

Filtering in the DFT space

- Low-Pass Filter (LPF):
  - Filtering out the large spatial frequencies



a)

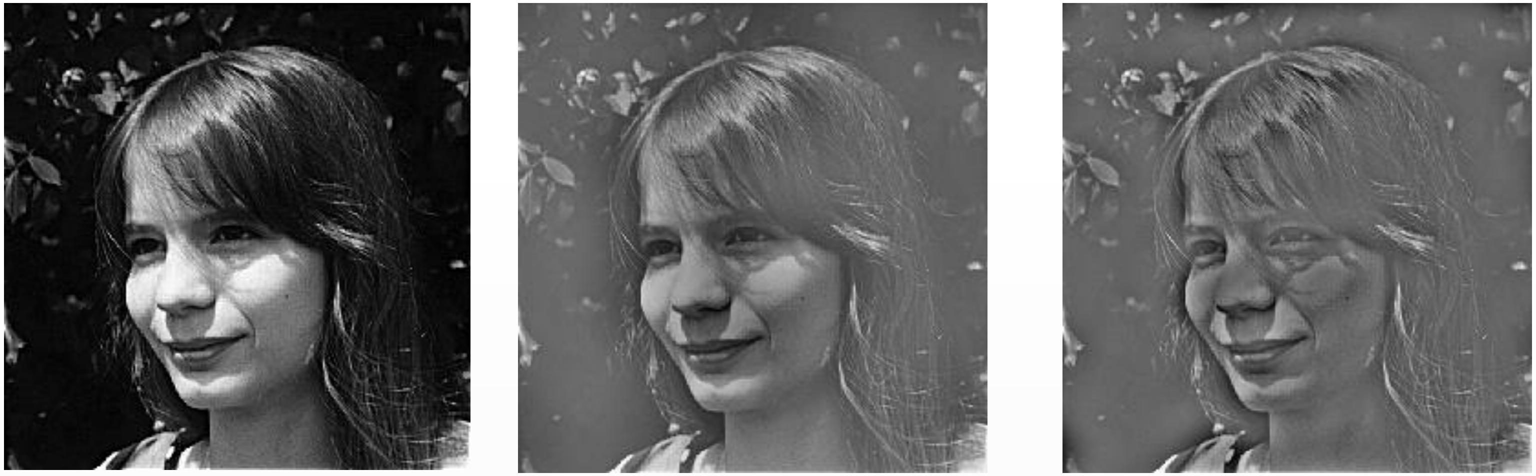
b)

Result of filtering out large frequencies. Erased all coeff. a.) above 16 f_b, b.) above 8 f_b

Slide credit ® Prof. Vladimir Székely, BME

Filtering in the DFT space

- High-Pass Filter (HPF)
  - Filtering out the low spatial frequencies

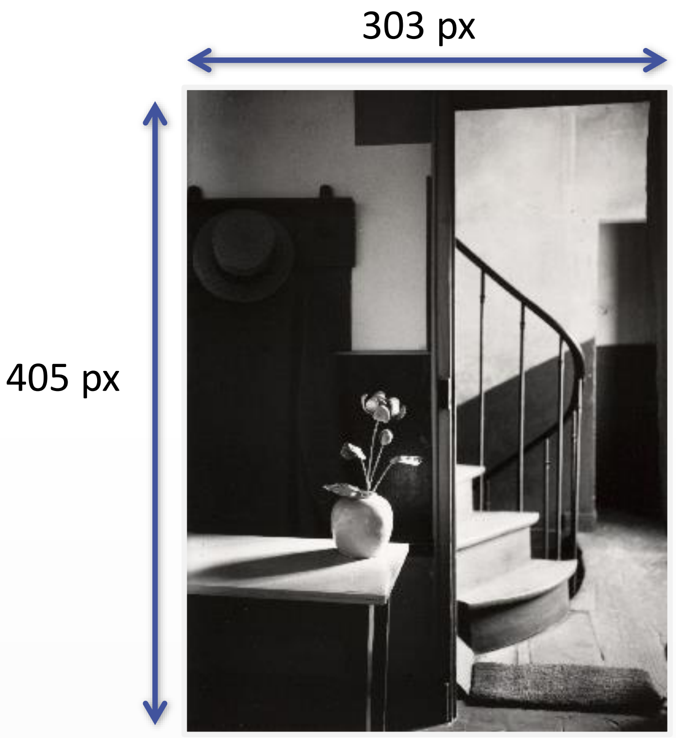


a) b)

Result of filtering out large frequencies. Erased all coeff. a.) below 4 f_b, b.) below 10 f_b

Slide credit ® Prof. Vladimir Székely, BME

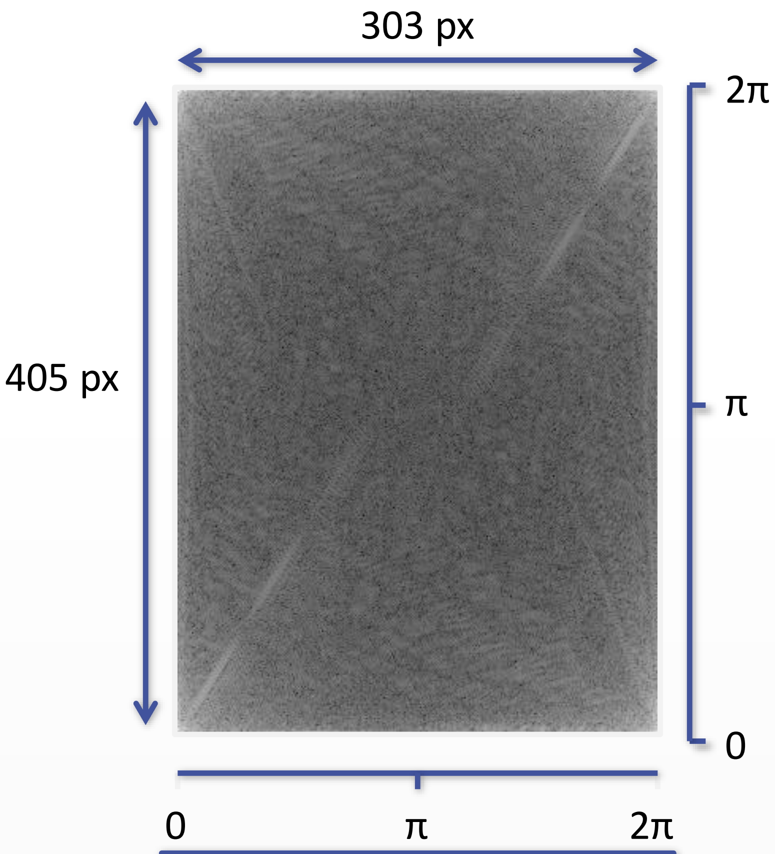
Discrete Fourier Transform



303 px

405 px

Original Image*



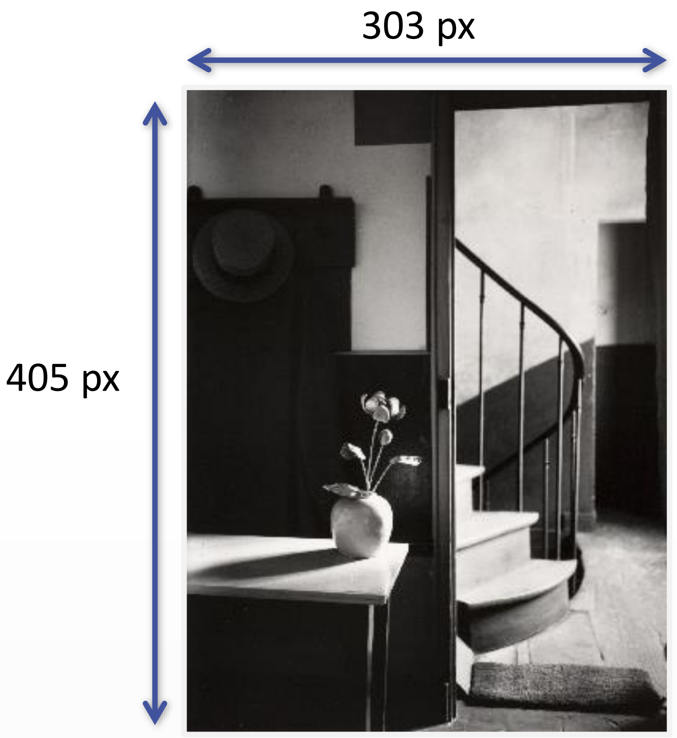
303 px

405 px

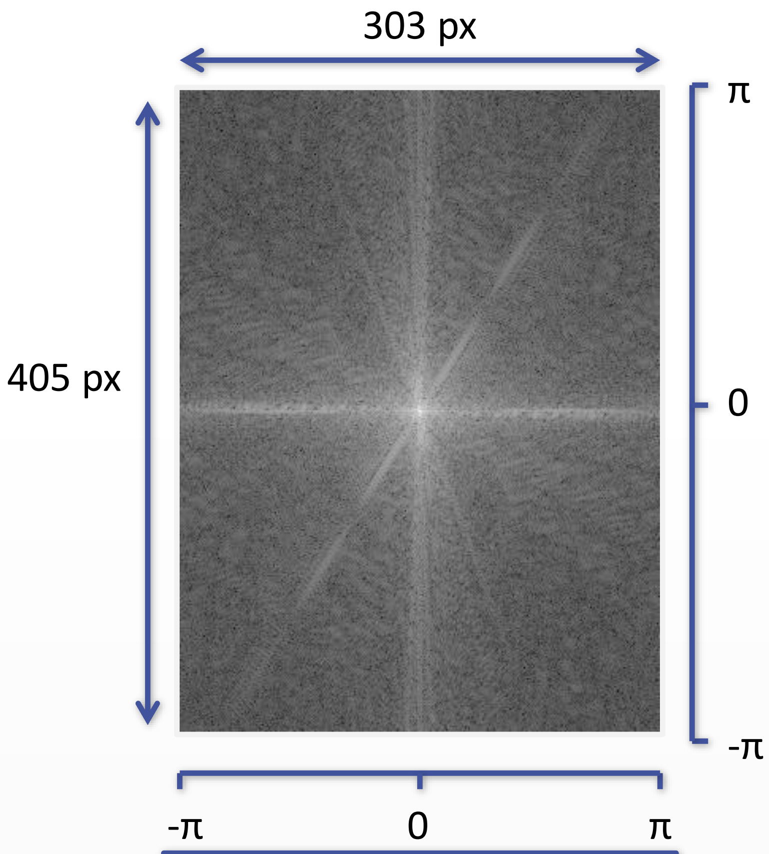
Magnitude of the Discrete Fourier Transform

*Chez Mondrian by András Kertész (1926)

Discrete Fourier Transform



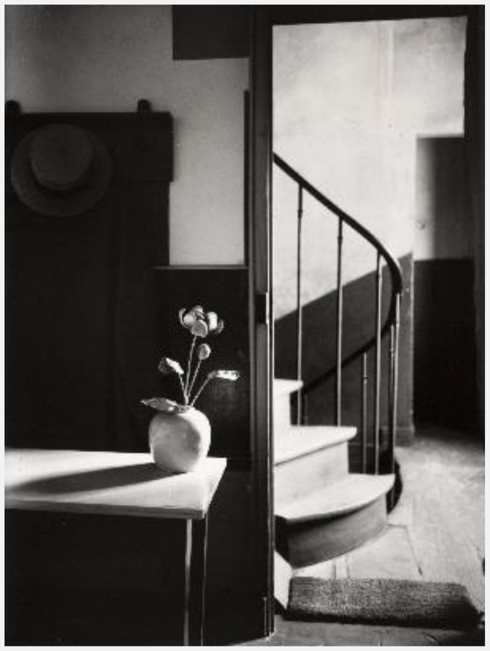
Original Image*



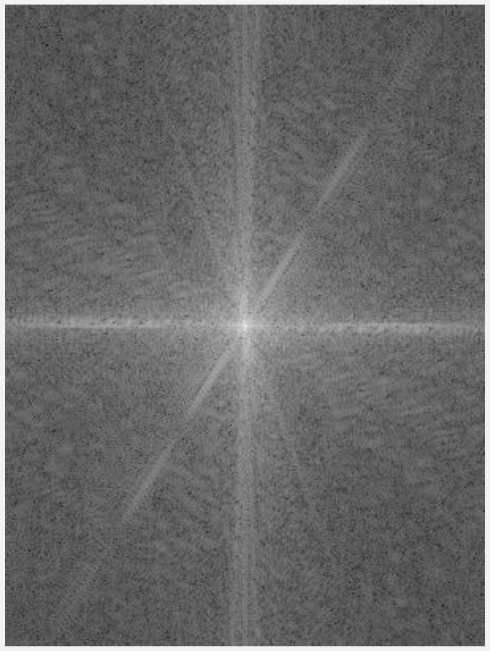
Magnitude of the **centered** Discrete Fourier Transform

*Chez Mondrian by András Kertész (1926)

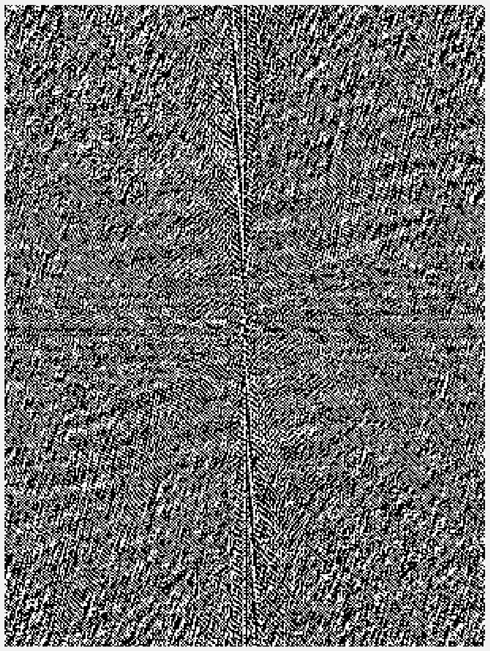
Discrete Fourier Transform



Original Image*



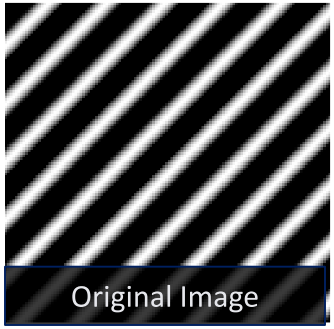
Magnitude of the **DFT**

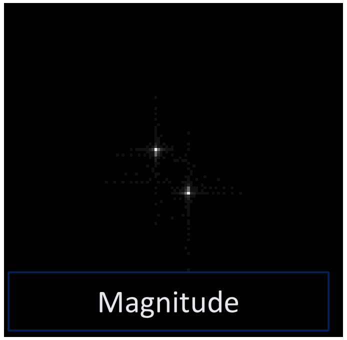


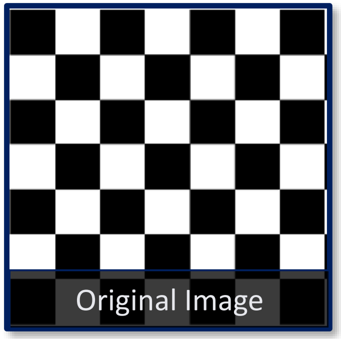
Phase of the **DFT**

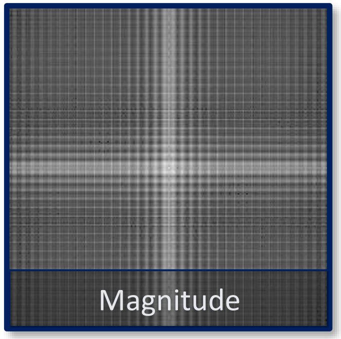
*Chez Mondrian by András Kertész (1926)

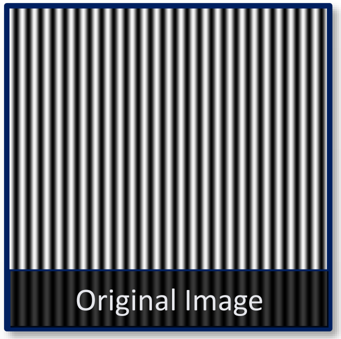
Discrete Fourier Transform - Examples

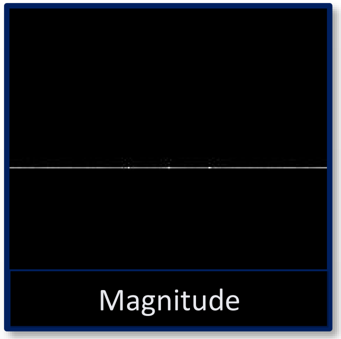


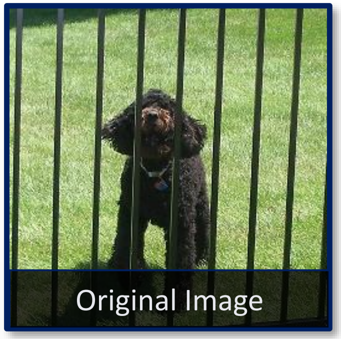


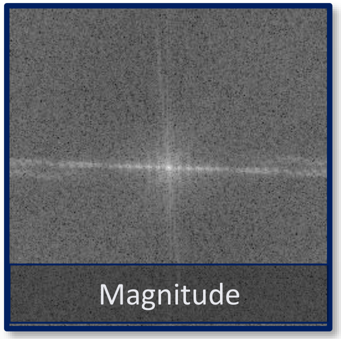




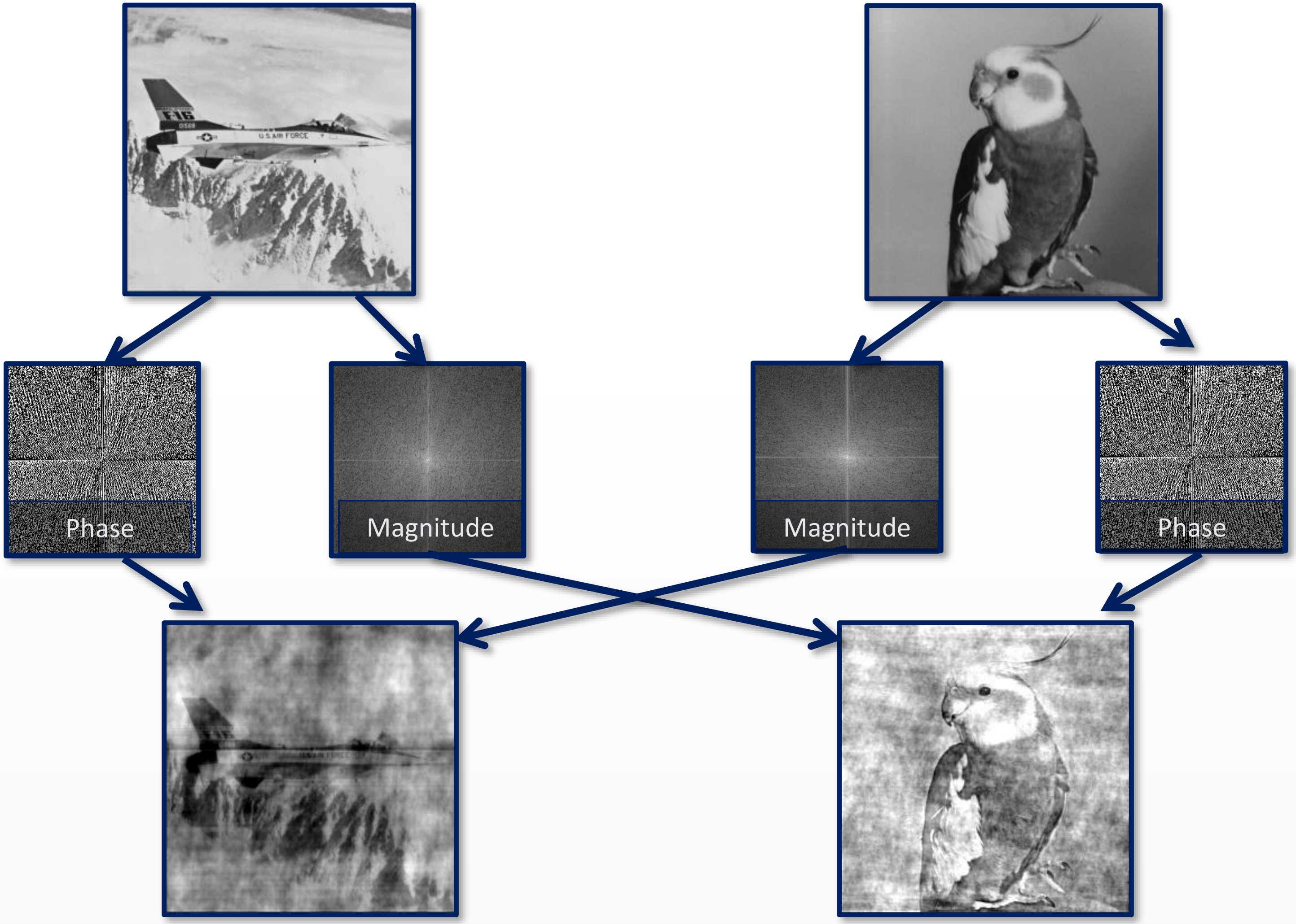








Discrete Fourier Transform

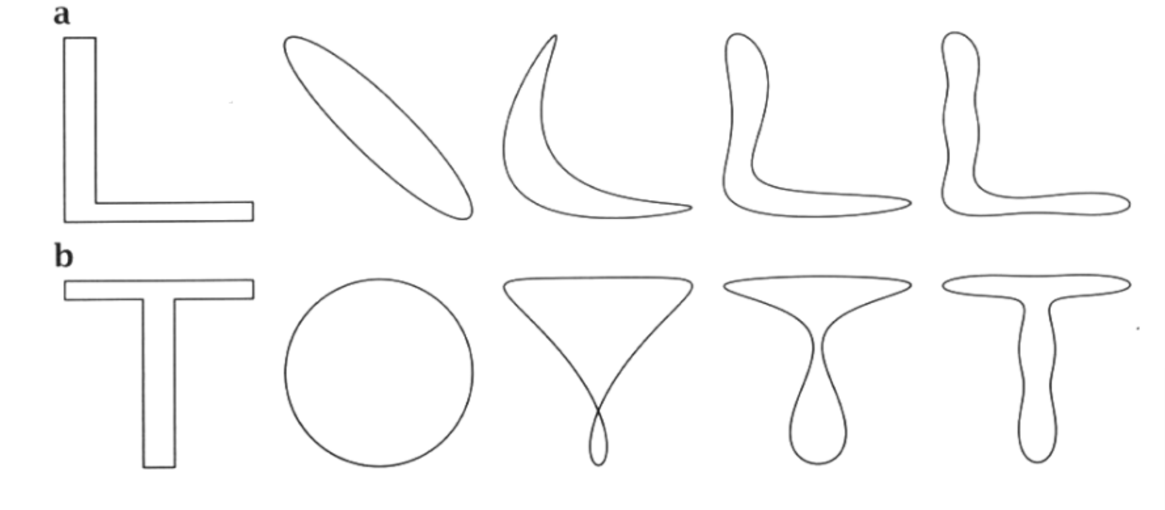


Representing curves with Fourier-descriptors

1. Complex numbers from the 2D curve points: $z_k = x_k + j \cdot y_k$
2. 1D DFT transform calculated for the complete closed curve

$$C_n = \sum_{k=0}^{K-1} z_k \exp(j \frac{2\pi}{K} n \cdot k)$$

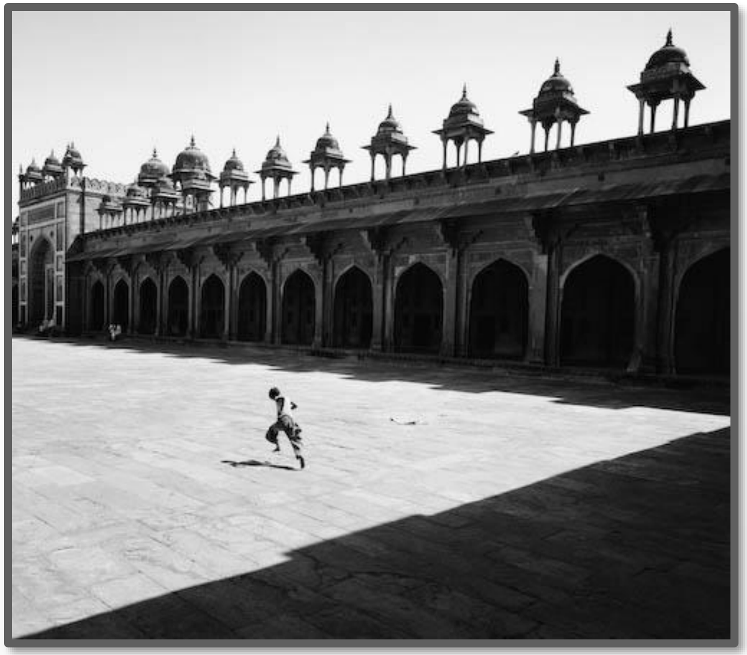
3. Setting high frequency C_n coefficient to zero, then recovering of the approximate contour points by inverse transform



Reconstruction of letters „L” and „T” with 2, 3, 4 and 8 Fourier coefficients

Homomorphic filtering

- ***Motivation*: image** with **large dynamic range**, e.g. natural scene on brightly sunny day, recorded on a **medium** with **small dynamic range** results in image contrast significantly reduced especially in dark and bright regions



- ***Goal***: reduce the dynamic range, increase contrast

- Example for a spatial filter also using Fourier-based steps

Homomorphic filtering

- It simultaneously ***normalizes the brightness*** across an image and ***increases contrast***.
- Assumes the following image model: the image is formed by recording the light reflected from the objects illuminated by a light source.

$$x(n_1, n_2) = \boxed{i(n_1, n_2)} \cdot \boxed{r(n_1, n_2)}$$

Illumination: slowly varying, main contributor to dynamic range

Reflectance: rapidly varying, main contributor to local contrast

- We want to reduce the illumination component, and increase the reflectance component.

Homomorphic filtering

- The main steps of homomorphic filtering:

1. To separate the two components we first use log transformation:

$$\log(x(n_1,n_2)) = \log(i(n_1,n_2)) + \log(r(n_1,n_2))$$

2. Since we assume that the illumination component varies slowly and the reflectance varies rapidly, we can get the two component by using (Fourier-based) low and high pass filters:

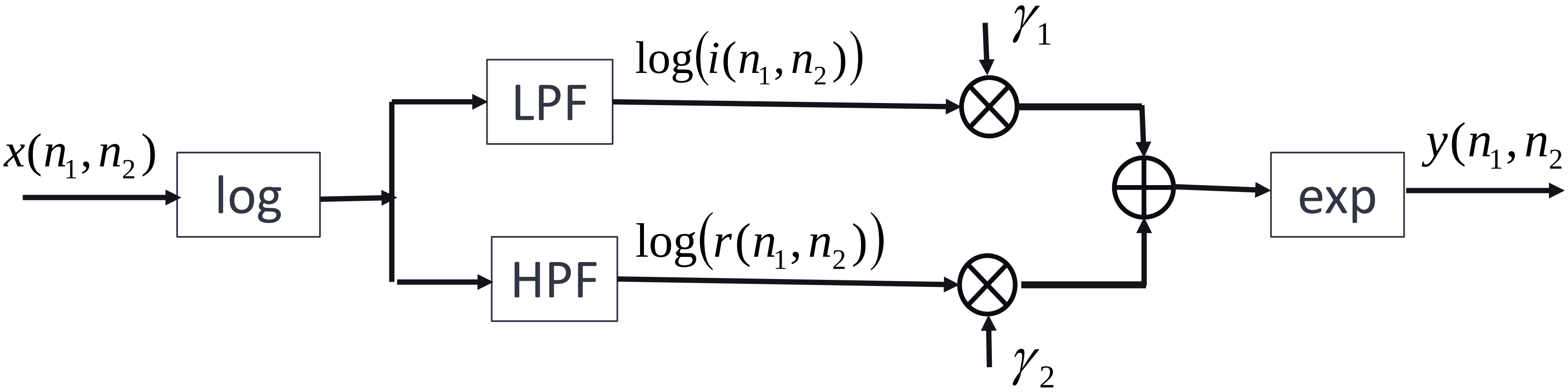
$$\log(i(n_1,n_2)) = LPF[\log(x(n_1,n_2))]$$
$$\log(r(n_1,n_2)) = HPF[\log(x(n_1,n_2))]$$

3. Weight the two component:

$$\log(y(n_1,n_2)) = \gamma_1 \log(i(n_1,n_2)) + \gamma_2 \log(r(n_1,n_2)), \text{ where } \gamma_1 < 1, \gamma_2 > 1$$

4. Transform back to the original range, using the exponential transform.

Homomorphic filtering

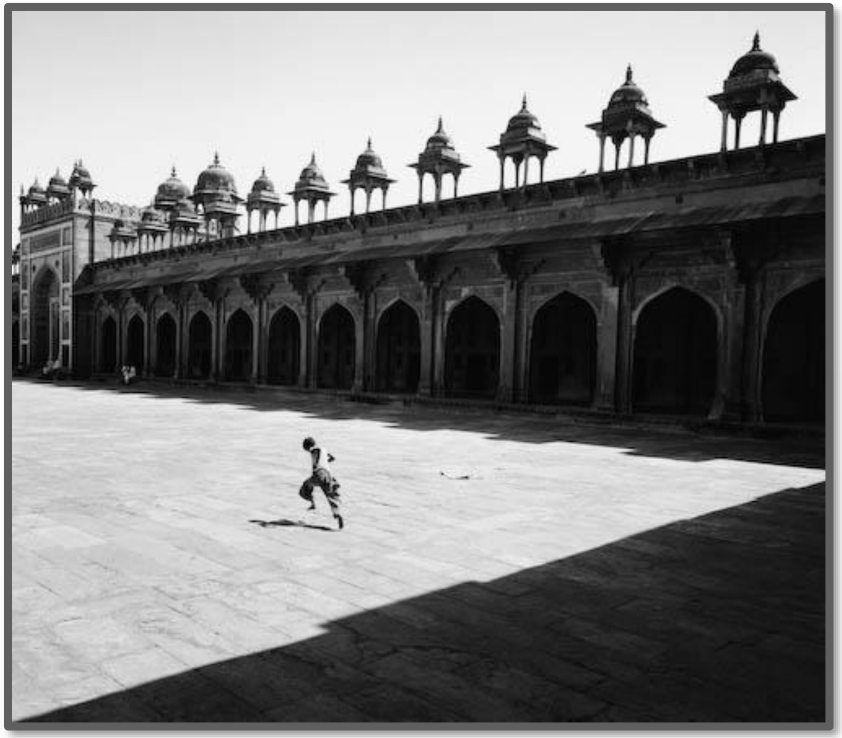


$$x(n_1,n_2) = i(n_1,n_2)r(n_1,n_2)$$

$$\log(y(n_1,n_2)) = \gamma_1 \log(i(n_1,n_2)) + \gamma_2 \log(r(n_1,n_2))$$

$$y(n_1,n_2) = [i(n_1,n_2)]^{\gamma_1}[r(n_1,n_2)]^{\gamma_2}$$

Homomorphic filtering



Original Image

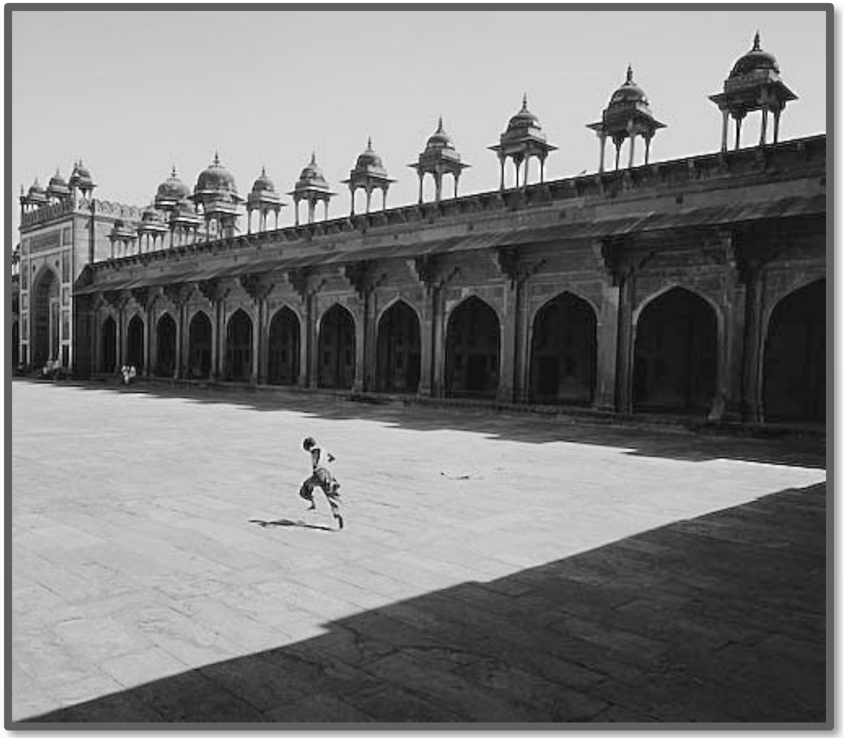


Image after homomorphic filtering

Main Sources

Fundamentals of Digital Image and Video Processing lectures by Aggelos K. Katsaggelos

Introduction to Fourier Transform (https://www.youtube.com/watch?v=1JnayXHhjlg)

Introduction to Compex Exponential Function (https://www.youtube.com/watch?v=qjT3XvS7Qno)

Basic Image Processing

PPKE-ITK

Lecture 5.

Texture analysis

Motivation: find the object!

Solution: color filtering

KIFESTŐ LUSTÁKNAK

Solution: texture segmentation

Textures - definition

- Textures demonstrate the difference between an artificial world of objects whose surfaces are only characterized by their color and reflectivity properties to that of real world imagery
- How we can define texture: microstructure
  - certainly not: an arbitrary pattern that extends over a large image
- Basic properties
  - Small elementary pattern which is repeated periodically or quasi-periodically in space (like pattern on a wall paper)
- It is sufficient to describe:
  - Small elementary pattern
  - Repetition rules (characteristic scales)
- Types
  - Artificial (Julesz, Pratt, Gagalowic)
  - Natural (Brodatz)

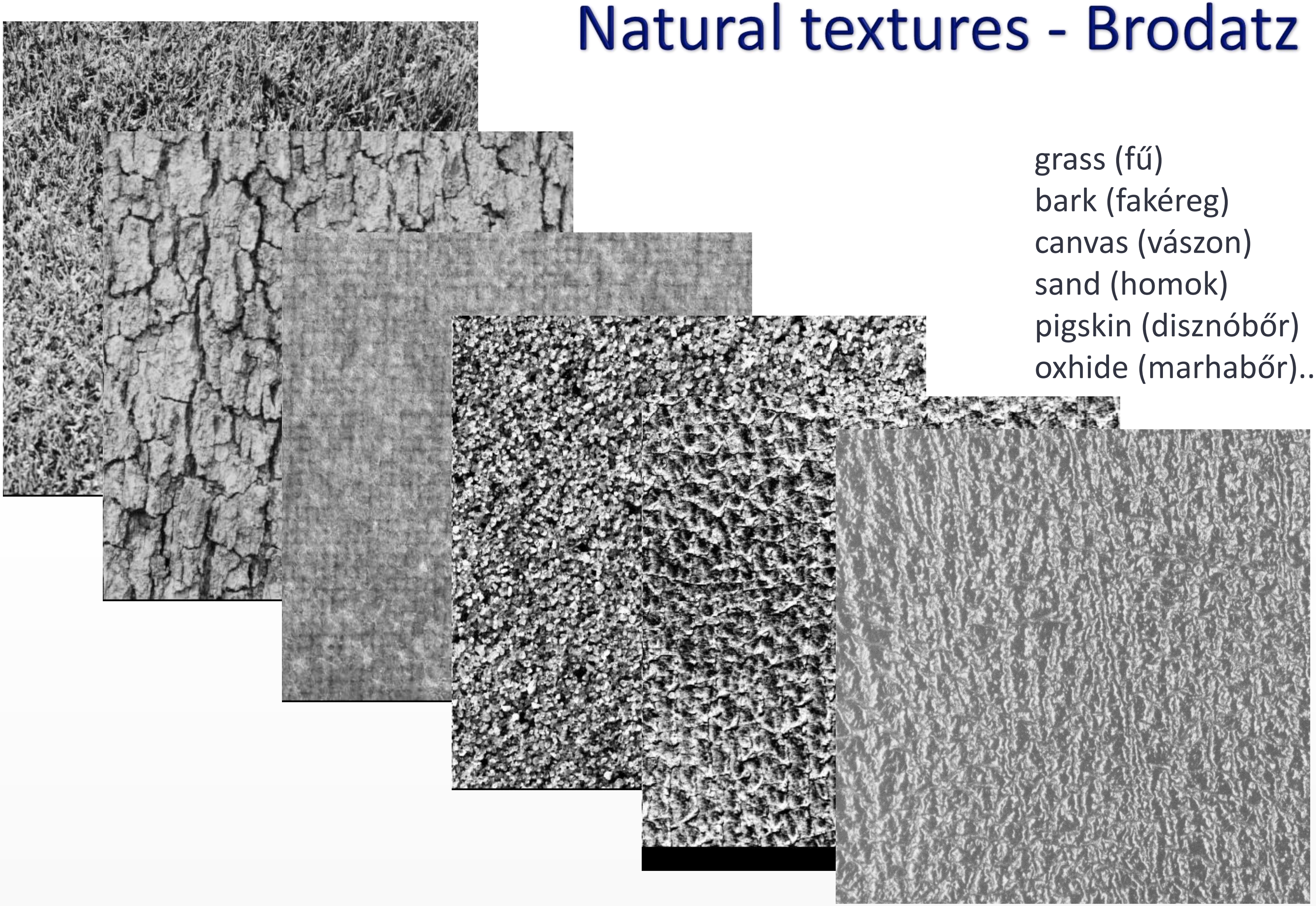
Textures - definition

- Hawkins:
  - Some local ‚order’ is repeated over a region which is large in comparison to the order’s size,
  - The order consists in the nonrandom arrangement of elementary parts
  - The parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region
- Description:
  - *coarseness* ~ period of repetition
    - e.g. wool is "coarser" than silk, under the same conditions
  - fineness / rudeness, contrast, orientation, arrangement ...

Texture

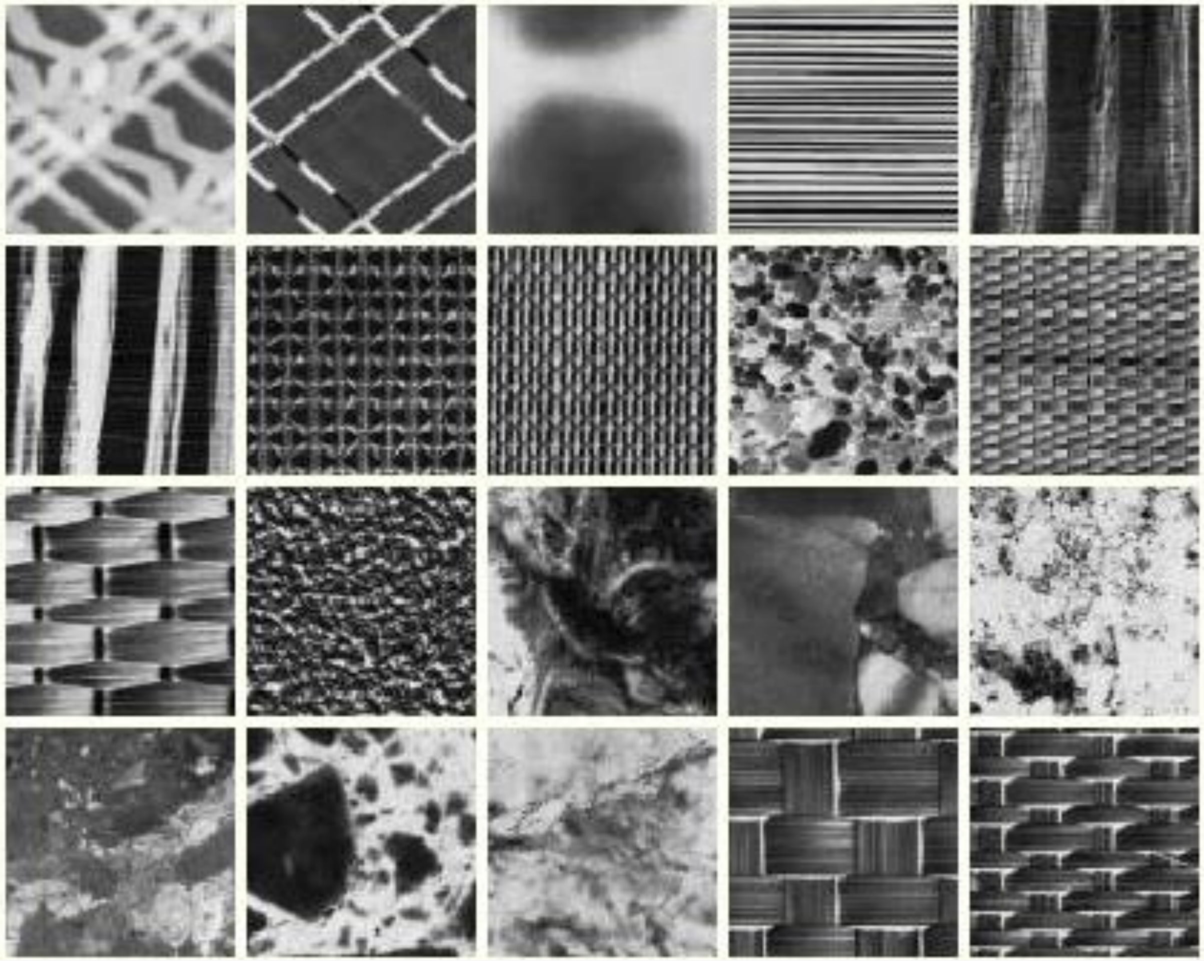
- A texture is an image that follows some statistical properties
- It has similar structures repeated over and over again

Natural textures - Brodatz

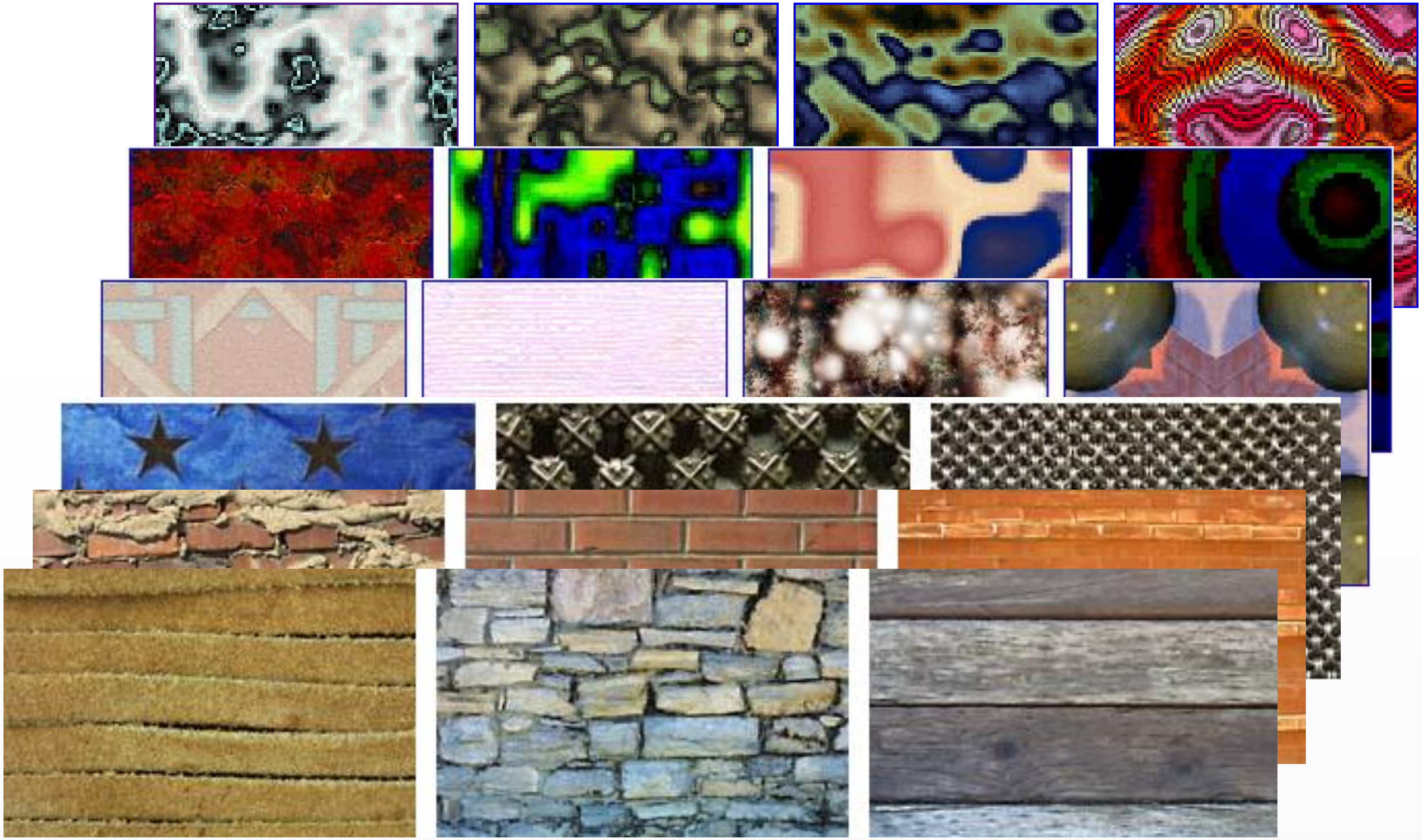


grass (fű)
bark (fakéreg)
canvas (vászon)
sand (homok)
pigskin (disznóbőr)
oxhide (marhabőr)...

Further natural textures from Brodatz



Artificial textures

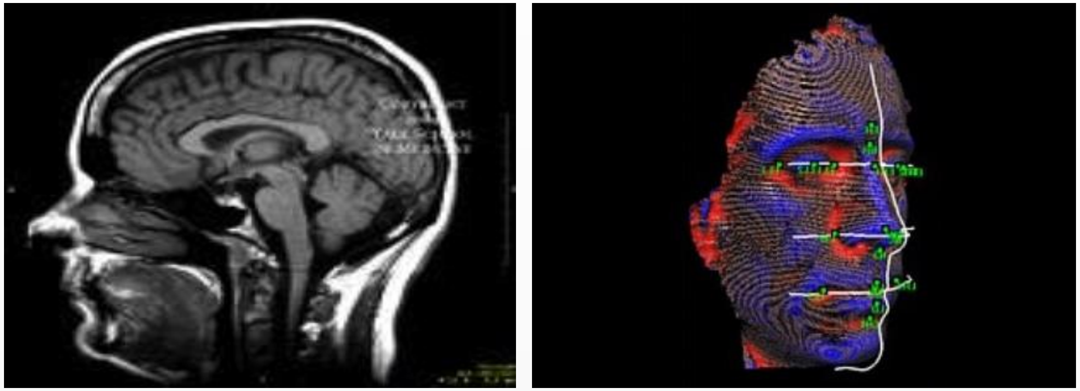


Application Areas of Texture Analysis

Food processing industry

Biometrics analysis (fingerprint, iris or retina, etc.)

Medical image analysis



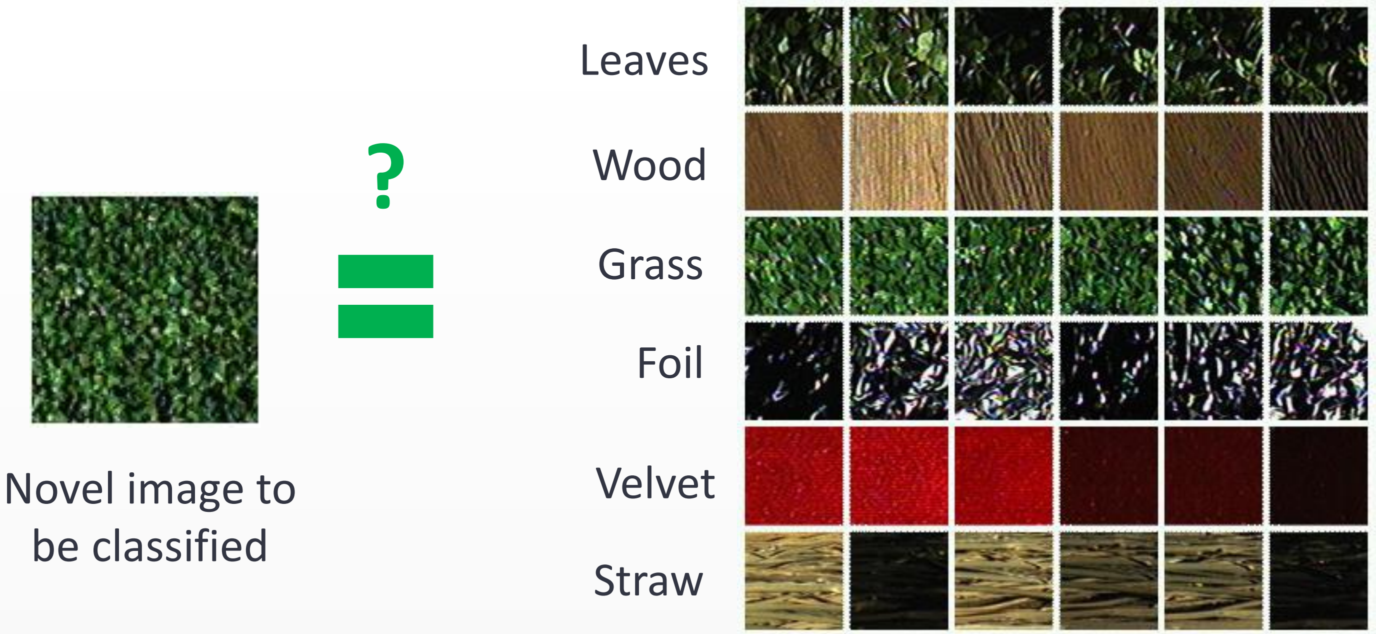
Global information system (GIS) (for land, etc. analysis)

Texture analysis:

- There are various primary issues in texture analysis:
  - *TEXTURE CLASSIFICATION*
  - *TEXTURE SEGMENTATION*
  - *SHAPE RECOVERY FROM TEXTURE, and*
  - *MODELING*.

Texture classification

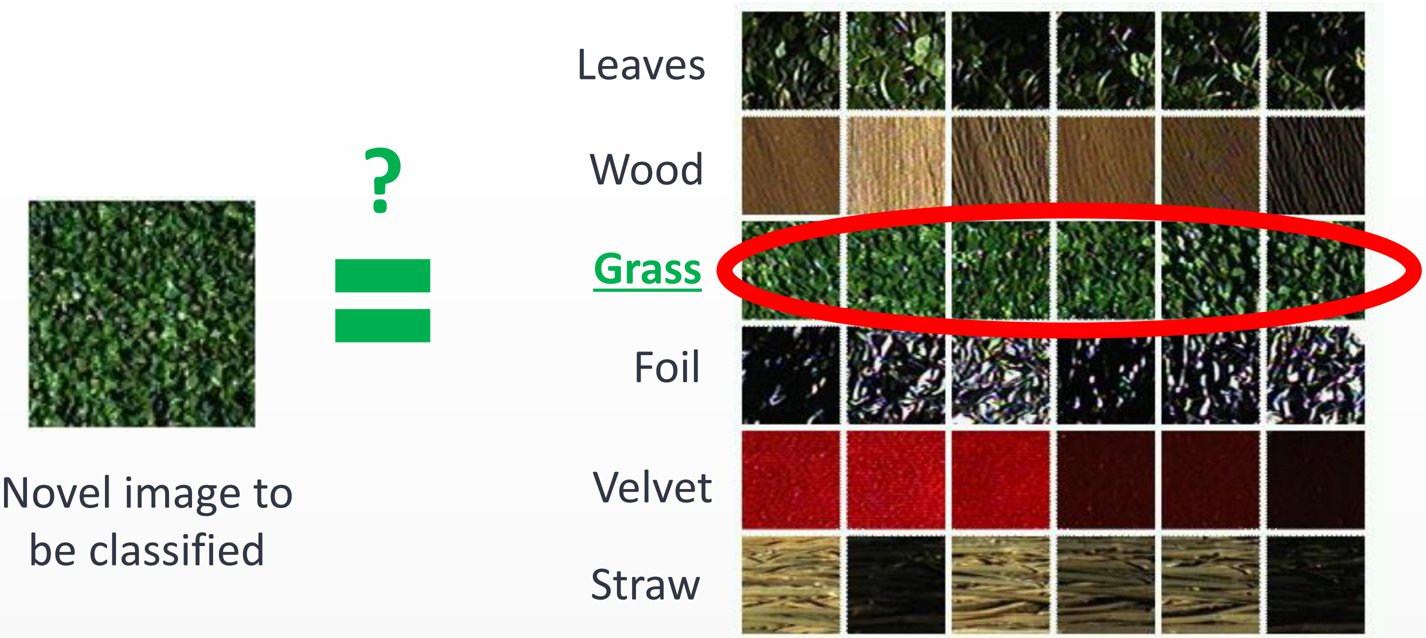
- In texture **classification**, the problem is **identifying** the given textured region from a given set of texture classes.
  - The texture analysis algorithms **extract distinguishing feature** from each region to facilitate classification of such patterns.



Novel image to be classified

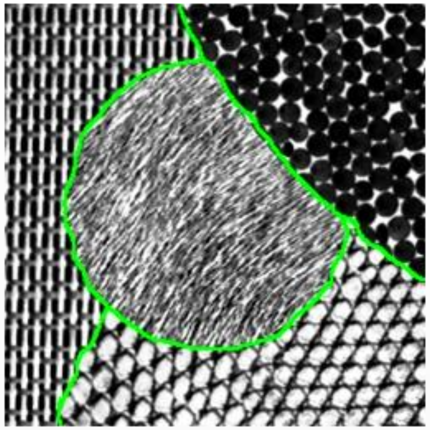
Texture classification

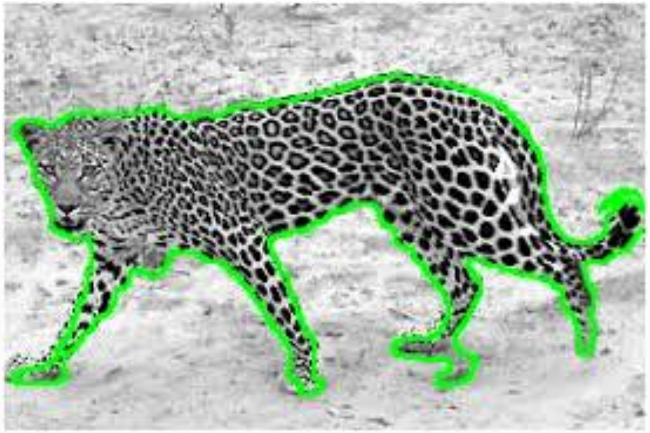
- In texture **classification**, the problem is **identifying** the given textured region from a given set of texture classes.
  - The texture analysis algorithms **extract distinguishing feature** from each region to facilitate classification of such patterns.

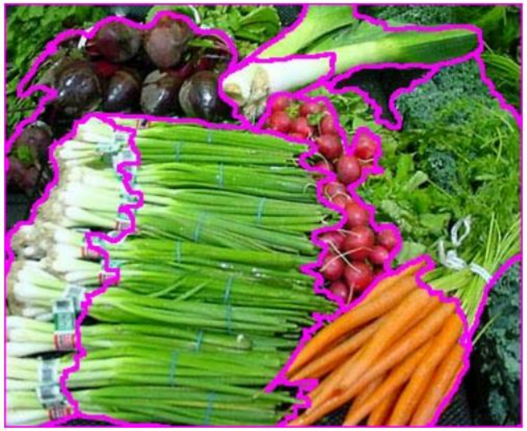


Texture segmentation

- Unlike texture classification, texture **segmentation** is concerned with automatically determining the **boundaries** between various textured regions in an image.
- Both reign-based methods and boundary-based methods have been attempted to segments texture images.





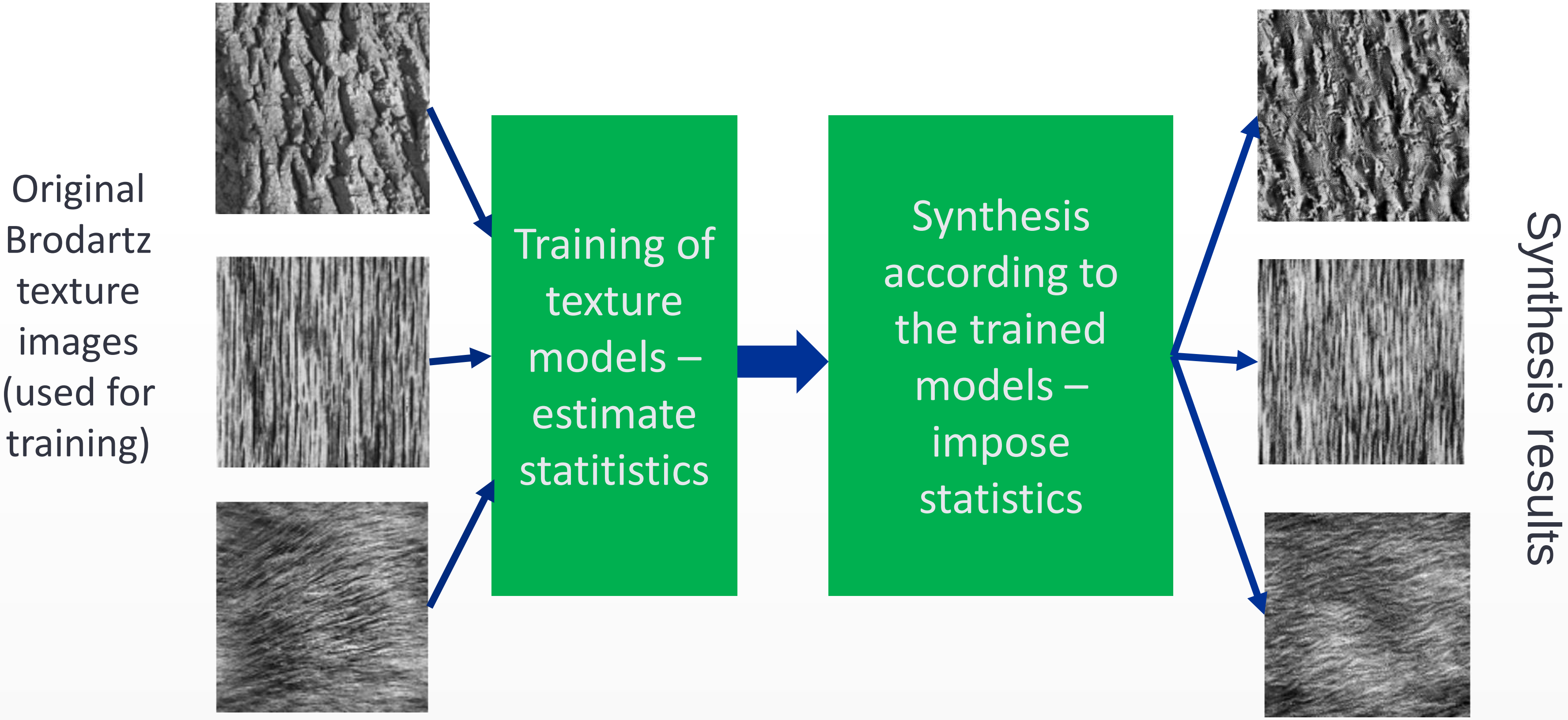


Shape recovery from texture

- Image plane variation in the texture properties, such as density, size and orientation of texture primitives, are the cues exploited by shape –from-texture algorithms.
- Quantifying the changes in the shape of texture elements is also useful to determine surface orientation.

Texture modeling

- Specify a model that clearly identifies the given pattern sample



Original Brodartz texture images (used for training)

Training of texture models – estimate statititistics

Synthesis according to the trained models – impose statistics

Synthesis results

Techniques for Texture Extraction

- There are various techniques for texture extraction. Texture feature extraction algorithms can be grouped as follows:
  - **Statistical**
  - **Geometrical**
  - **Model based**
  - **Signal Processing**

Statistical methods

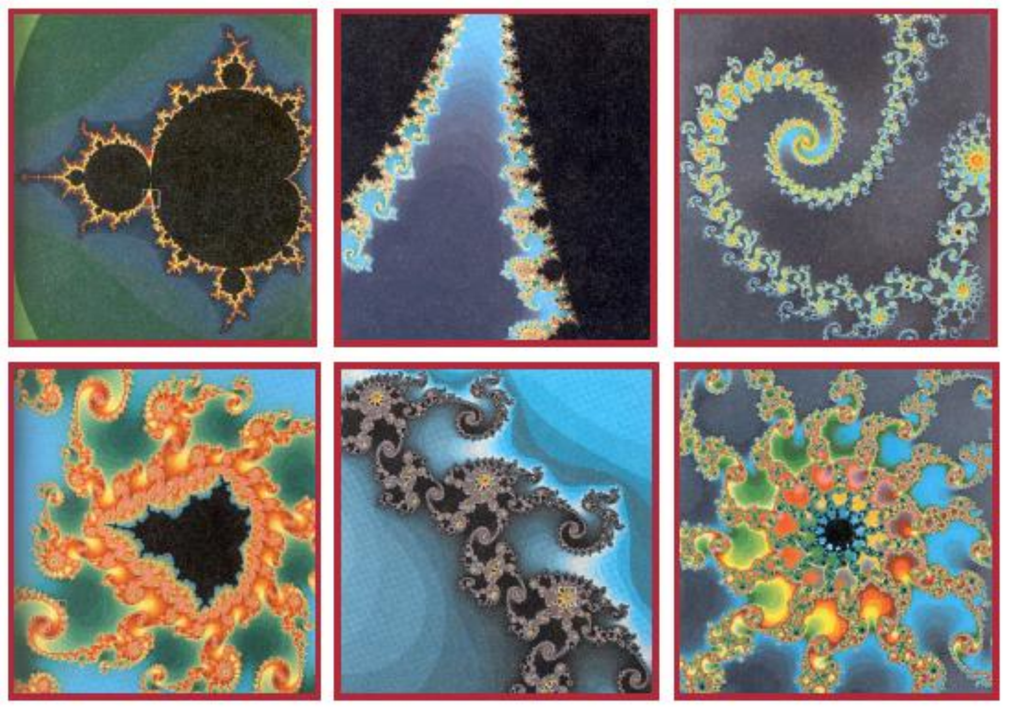
1. Local features
   - Grey level of central pixels,
   - Average of grey levels in window,
   - Median,
   - Standard deviation of grey levels,
   - Difference of maximum and minimum grey levels,
   - Difference between average grey level in small and large windows,
   - Kirsch feature,
   - Combine features
2. Galloway
   - run length matrix
3. Haralick
   - co-occurrence matrix

Geometrical methods

- Steps
  1. **Threshold** images into binary images of *n* grey levels.
  2. Calculate **statistical** features of **connected areas**.

Model based methods

- These involve building mathematical models to describe textures:
  - **Markov random fields** (see: later – image segmentation lecture)
  - Fractals:



Signal processing includes

- These methods involve transforming original images using filters and calculating the energy of the transformed images.

- **Law’s masks (see: today – later)**
- Laines – Daubechies wavelets
- **Fourier transform (see: last lecture)**
- **Gabor filters (see: today – later)**

Flowchart for Texture Analysis

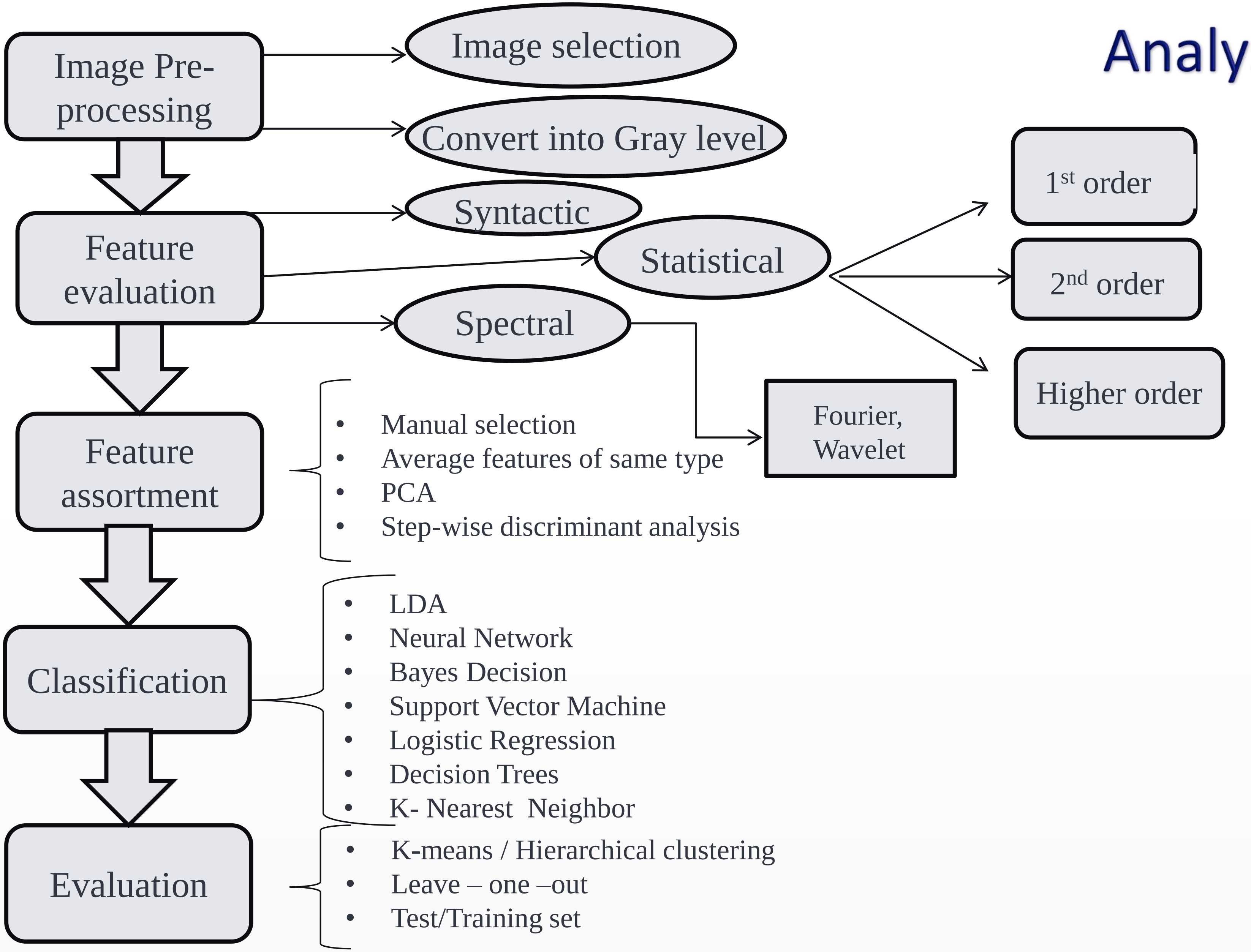


Image Pre-processing

- Image selection
- Convert into Gray level

Feature evaluation

- Syntactic
- Statistical
  - 1st order
  - 2nd order
  - Higher order
- Spectral
  - Fourier, Wavelet

Feature assortment

- Manual selection
- Average features of same type
- PCA
- Step-wise discriminant analysis

Classification

- LDA
- Neural Network
- Bayes Decision
- Support Vector Machine
- Logistic Regression
- Decision Trees
- K- Nearest Neighbor

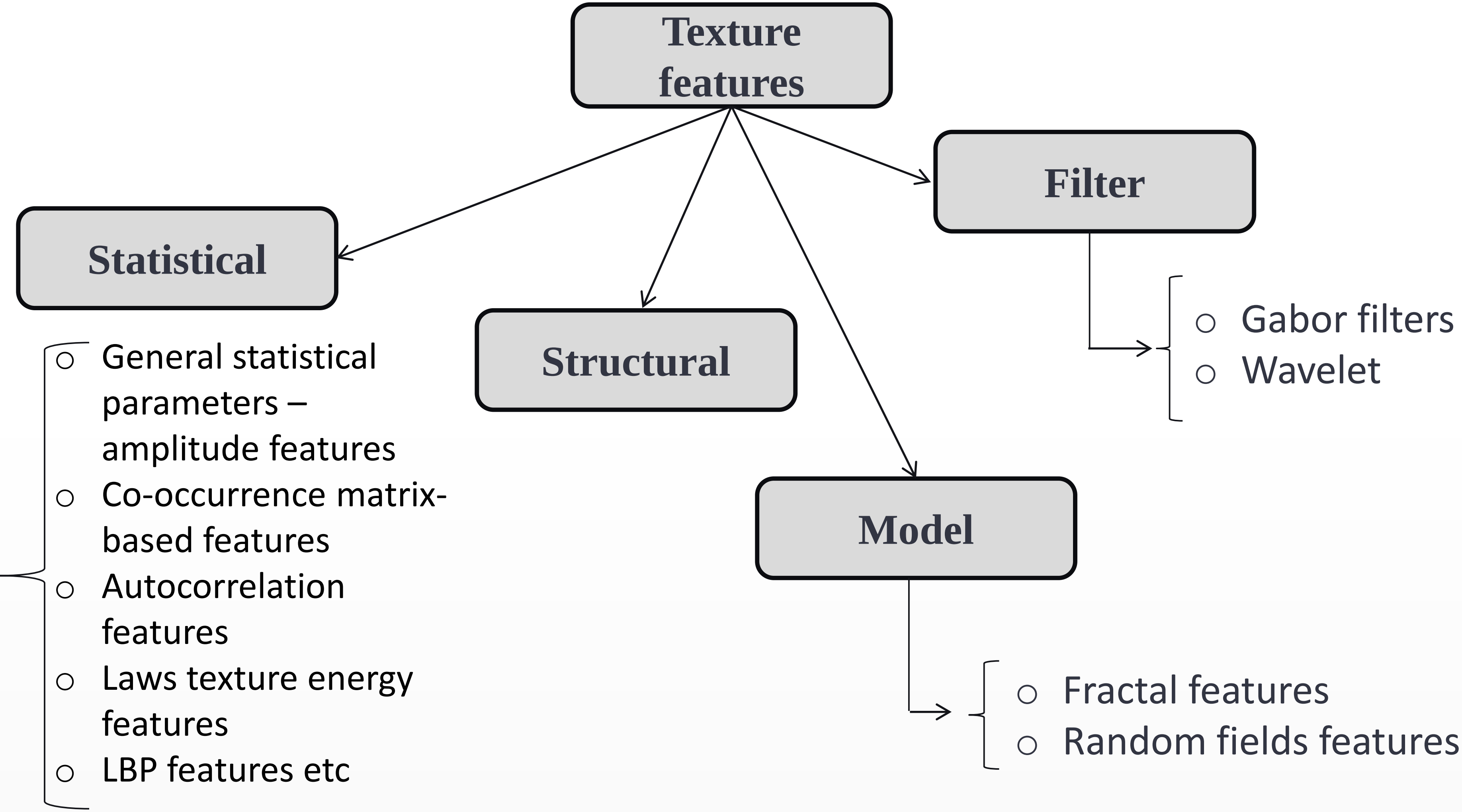
Evaluation

- K-means / Hierarchical clustering
- Leave – one –out
- Test/Training set

Discrimination vs. classification of textures

- Discrimination
  - Classification - grouping of blobs or points, and classifying them into various classes (local attributes)
  - Segmentation - separation of spots / areas (local properties + neighborhoods)
- Classification
  - Supervised approach
    - take samples of textures (statistics, metrics) then
    - examine the similarity of new textures
  - Unsupervised
    - evaluate statistics
    - sample categorization into classes

Methods for Texture Features



Texture features

Statistical

- ○ General statistical parameters – amplitude features
- ○ Co-occurrence matrix-based features
- ○ Autocorrelation features
- ○ Laws texture energy features
- ○ LBP features etc

Structural

Filter

- ○ Gabor filters
- ○ Wavelet

Model

- ○ Fractal features
- ○ Random fields features

Statistical features
Amplitude features

- Amplitude-features
  - Mean

$$M(j,k)=\frac{1}{(2w+1)^2}\sum_{m=-w}^{w}\sum_{n=-w}^{w}F(j+m,k+n)$$

  - Deviation

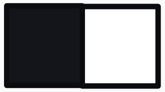
$$S(j,k)=\frac{1}{(2w+1)^2}\left[\sum_{m=-w}^{w}\sum_{n=-w}^{w}\left[F(j+m,k+n)-M(j+m,k+n)\right]^2\right]^{1/2}$$

Statistical features
Gray Level Co-occurrence Matrix (GLCM)

- Also referred as **co-occurrence distribution**.
- It is the most classical second-order statistical method for texture analysis.
- An image is composed of **pixels** each with an intensity (a specific gray level), the GLCM is a tabulation of how often different combinations of gray levels co-occur in an image or image section.
- **Gray Level Co-occurrence Matrix** (GLCM) filters operate by computing, for each filter window position, how often specific pairs of image cell values occur in neighboring cell positions (such as one cell to the right).
- The results are tabulated in a co-occurrence matrix, and specific statistical measures are computed from this matrix to produce the filtered value for the target cell

Statistical features
First vs. second order statistics examples

- First order statistics example: simple histogram
  - Measures the number of different gray value occurrences of independent pixels
  - h_i: number of pixels in the image with gray value i
- Second order statistics example: GLCM
  - Measures the frequencies of joint gray value occurrences of different pixel pairs with a pre-defined spatial offset
  - $p_{ij}(\Delta x, \Delta y)$: frequency of pixel pairs with an offset $(\Delta x, \Delta y)$, where the gray value of the first pixel is i and the gray value of the second pixel is j
  - For example: Δx=1, $\Delta y = 0$, i=0, j=255: frequency of one-pixel-wide vertical black-white transitions in an image



Statistical: Co-occurrence Matrix-based Features

- It is a matrix of *frequencies* at which → two pixels, separated by a certain vector, occur in the image.
- Co-occurrence matrix is defined as,

$$p_{ij}(\Delta x, \Delta y) = W \cdot Q(i, j | \Delta x, \Delta y)$$

where,

$$W = \frac{1}{(M - \Delta x)(N - \Delta y)} \qquad Q(i, j | \Delta x, \Delta y) = \sum_{n=1}^{N-\Delta y} \sum_{m=1}^{M-\Delta x} A$$

where,

$$A = \begin{cases} 1 & \text{if } f(m,n) = i \text{ and } f(m+\Delta x, n+\Delta y) = j \\ 0 & \text{otherwise} \end{cases}$$

Computation of Co-occurrence Matrix

- It has size $N \times N$ (N = Number of gray-values) i.e., the rows & columns represent the set of possible pixel values.
- Polar representation of the offset:
  - two parameters d, θ (instead of $\Delta x, \Delta y$):

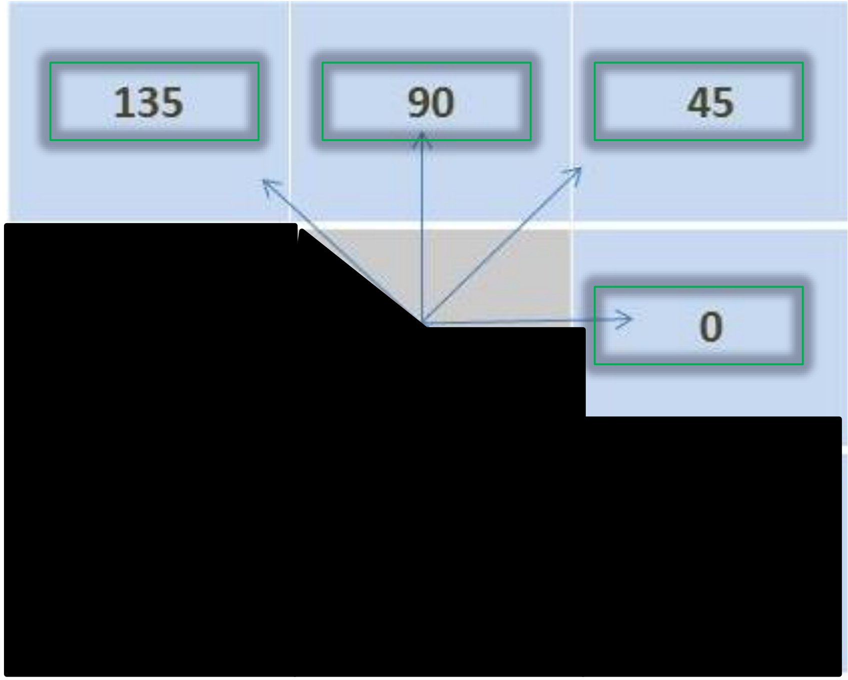
d → Relative **distance** between the pixel pair

(measured in pixel number. e.g., 1, 2, …)

θ → Relative **orientation** / rotational angle.

(e.g., 0º, 45º, 90º, 135º, …)

8 Directions/orientations (θ) of Adjacency



[redacted]

we consider θ as horizontal (0°), front diagonal (45°), vertical (90°) and back diagonal (135°)

Computation of Co-occurrence Matrix

Image matrix

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Find the **number of co-occurrences** of pixel *i* to the neighboring pixel value *j*

i/j	0	1	2	3
0	#(0,0)	#(0,1)	#(0,2)	#(0,3)
1	#(1,0)	#(1,1)	#(1,2)	#(1,3)
2	#(2,0)	#(2,1)	#(2,2)	#(2,3)
3	#(3,0)	#(3,1)	#(3,2)	#(3,3)

Pixel values: 0,1,2,3. **Thus,** $N = 4$

Thus, *size* of CM = 4x4

$d = 1$

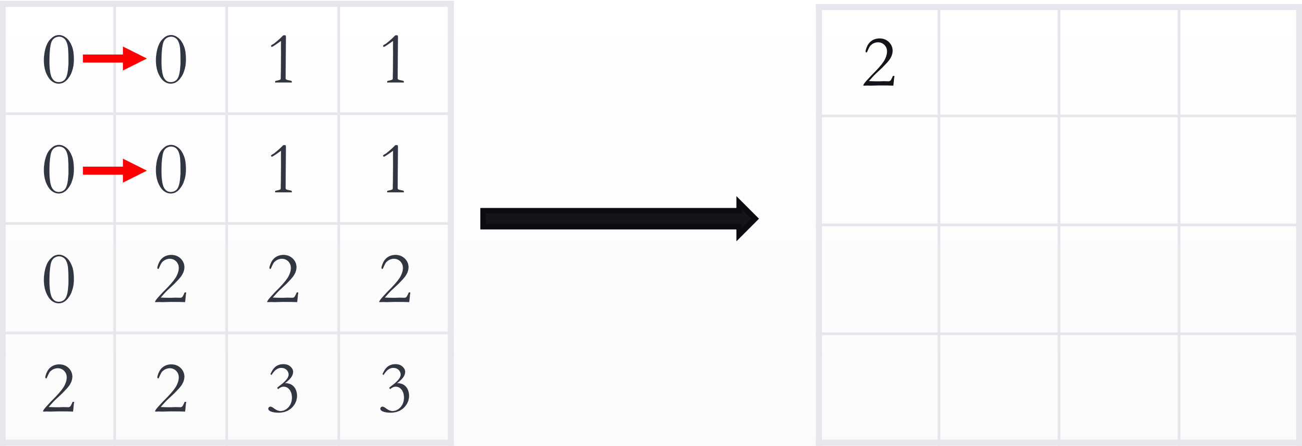
θ = horizontal (0°)

Example: Computation (contd.)

i/j	**0**
0	**#(0,0)** →

$d = 1$

θ = horizontal (0°) →

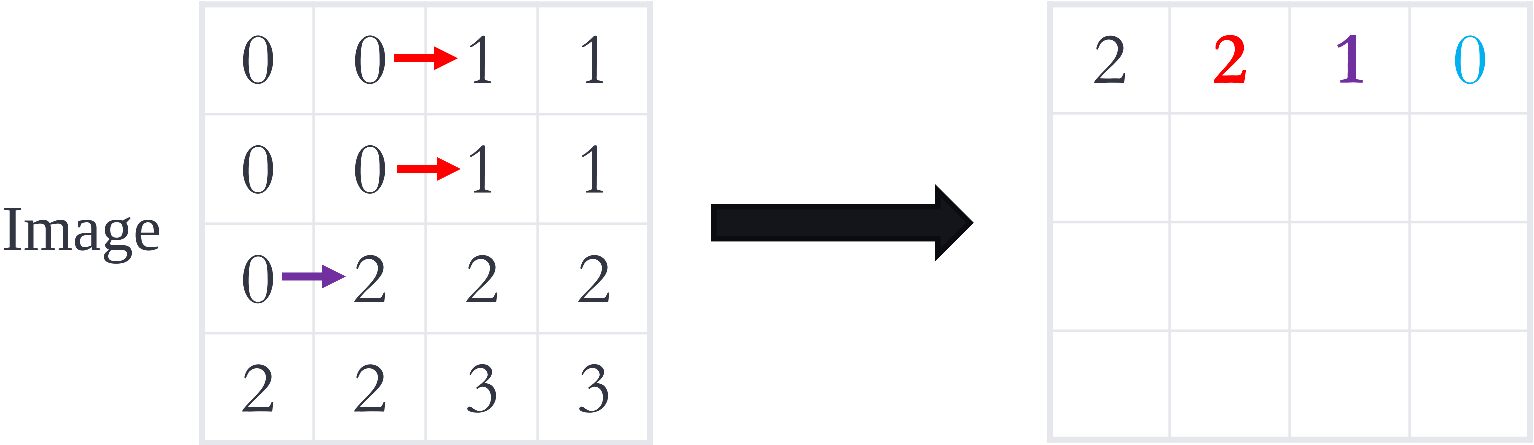


0 →	0	1	1
0 →	0	1	1
0	2	2	2
2	2	3	3

2			

Example: Computation (contd.)

$d = 1$ θ = horizontal (0°)



Image

0	0 →	1	1
0	0 →	1	1
0 →	2	2	2
2	2	3	3

2	2	1	0

CM for the Image

i/j	0	1	2	3
0		#(0,1)	#(0,2)	#(0,3)
1				
2				
3				

Example: Computation (contd.)

$d = 1$ $\theta =$ **horizontal**$(0°)$

Image

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

→

2	2	1	0
0	2	0	0
0	0	3	1
0	0	0	1

CM for the Image

i/j	**0**	**1**	**2**	**3**
0	**#(0,0)**	**#(0,1)**	**#(0,2)**	**#(0,3)**
1	**#(1,0)**	**#(1,1)**	**#(1,2)**	**#(1,3)**
2	**#(2,0)**	**#(2,1)**	**#(2,2)**	**#(2,3)**
3	**#(3,0)**	**#(3,1)**	**#(3,2)**	**#(3,3)**

Example: Computation (contd.)

$d = 1$ $\theta = \textbf{vertical}(90°)$

Image

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

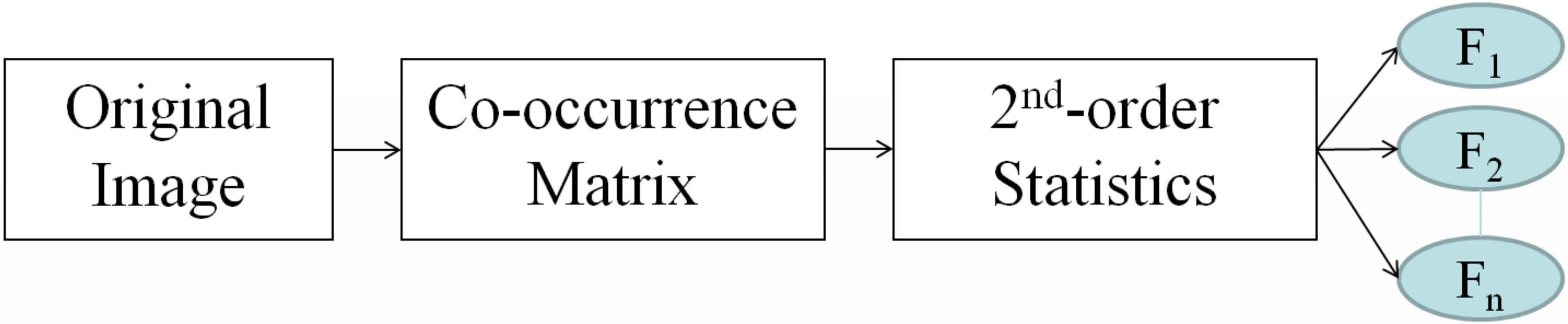
3	0	2	0
0	2	2	0
0	0	1	2
0	0	0	0

CM for the Image

i/j	**0**	**1**	**2**	**3**
0	**#(0,0)**	**#(0,1)**	**#(0,2)**	**#(0,3)**
1	**#(1,0)**	**#(1,1)**	**#(1,2)**	**#(1,3)**
2	**#(2,0)**	**#(2,1)**	**#(2,2)**	**#(2,3)**
3	**#(3,0)**	**#(3,1)**	**#(3,2)**	**#(3,3)**

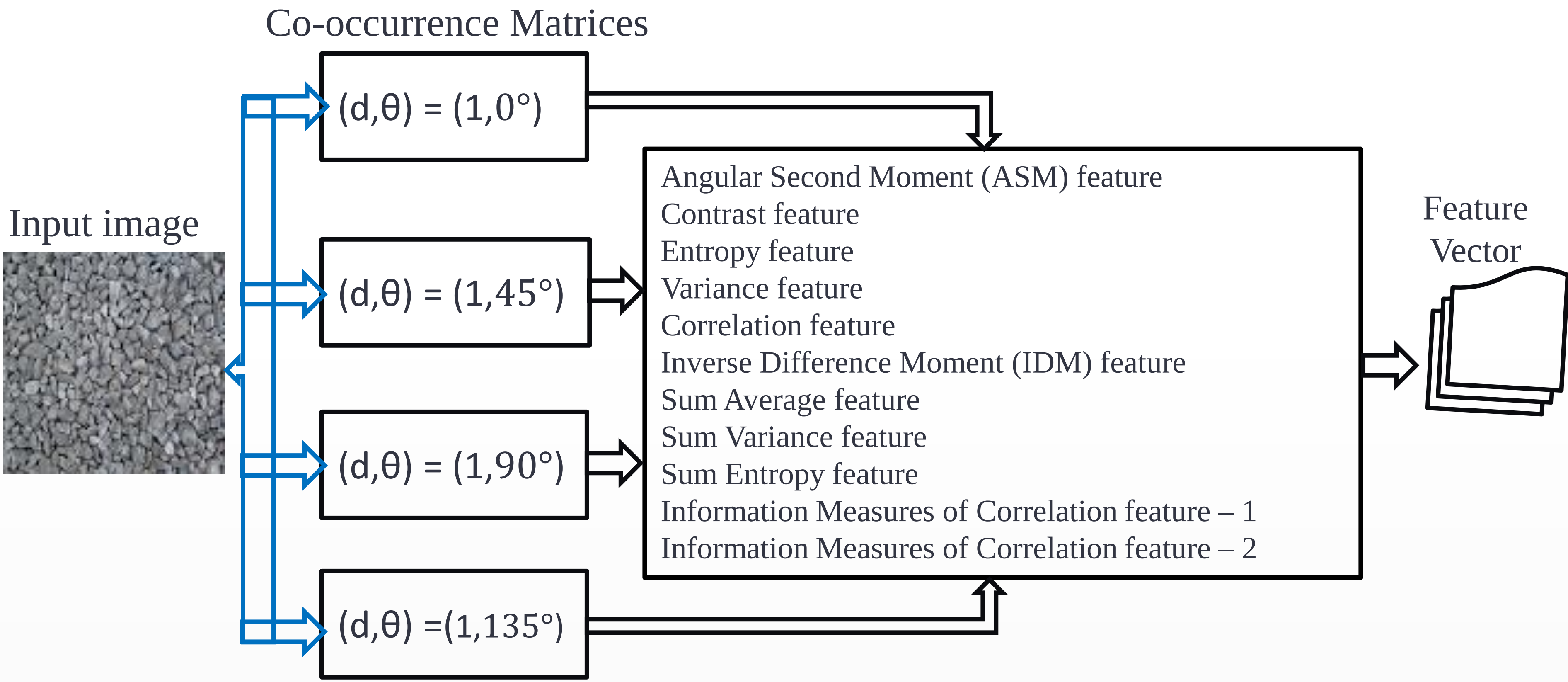
Features on co-occurrence matrix

- Co-occurrence matrices capture properties of a texture
- But they are *not directly useful* for further analysis (e.g., comparison of two textures)



11 Numeric features are computed from a matrix

Features on co-occurrence matrix



Co-occurrence Matrices

Input image

(d,θ) = (1,0°)

(d,θ) = (1,45°)

(d,θ) = (1,90°)

(d,θ) =(1,135°)

Angular Second Moment (ASM) feature
Contrast feature
Entropy feature
Variance feature
Correlation feature
Inverse Difference Moment (IDM) feature
Sum Average feature
Sum Variance feature
Sum Entropy feature
Information Measures of Correlation feature – 1
Information Measures of Correlation feature – 2

Feature Vector

Energy

- Also called **Uniformity or Angular second moment**.
- **Measures the textural uniformity that is pixel pair repetitions.**
- Detects disorders in textures.
- Energy reaches a maximum value equal to one

$$Energy=\sum_i \sum_j p_{ij}^2$$

Entropy

- **Measures the disorder or complexity of an image.**
- The entropy is large when the image is not texturally uniform.
- Complex textures tend to have high entropy.
- Entropy is strongly, but inversely correlated to energy.

$$Entropy = -\sum_i \sum_j p_{ij} \log_2 p_{ij}$$

Contrast

- **Measures the spatial frequency of an image and is difference moment of GLCM**.
- It is the difference between the highest and the lowest values of a contiguous set of pixels.
- It measures the amount of local variations present in the image.

$$Contrast=\sum_i \sum_j (i-j)^2 p_{ij}$$

Homogeneity

- Also called as **Inverse Difference Moment**.
- **Measures image homogeneity as it assumes larger values for smaller gray tone differences in pair elements**.
- It is more sensitive to the presence of near diagonal elements in the GLCM.
- It has maximum value when all elements in the image are same.
- Homogeneity decreases **if** contrast increases while energy is kept constant.
- Homogeneity(hom) = $\sum_i \sum_j \frac{1}{1+(i-j)^2} p_{ij}$

Variance

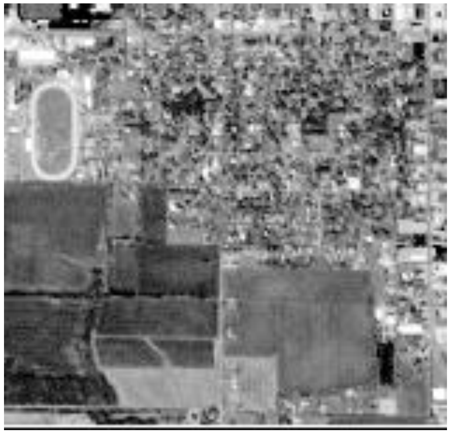
- **This statistic is a measure of heterogeneity and is strongly correlated to first order statistical variable such as standard deviation.**
- Variance increases when the gray level values differ from their mean

Variance(var)= $\sum_i \sum_j (i-\mu)^2 \, p_{ij}$

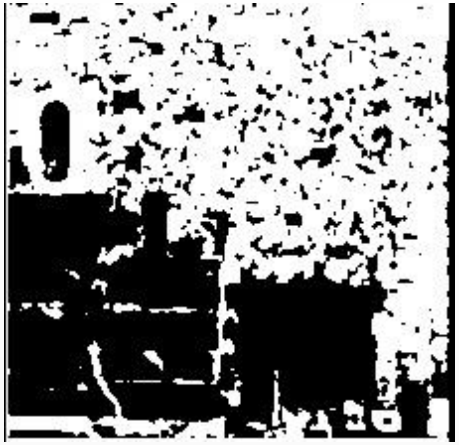
where μ is the mean of p_{ij}

GLCM Filters

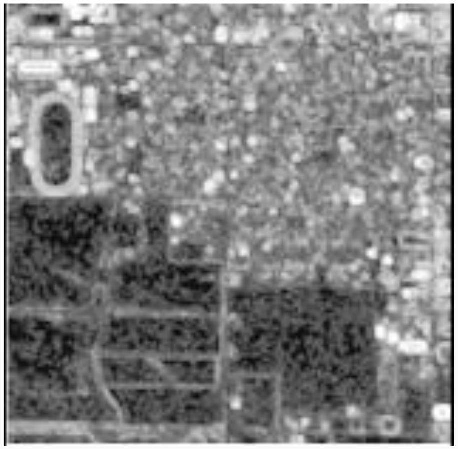
- **Contrast** (Sum of Squares Variance)
  - measures gray-level contrast by using GLCM weighting factors equal to the square of the gray level difference. Thus the averaging weights are 0 for matrix position on the main diagonal and increase exponentially away from the diagonal. The filter result is 0 for areas with identical image values and is high where there are large differences in tone.



Input image



Thresholded Contrast



Contrast filter output

Features on co-occurrence matrix Examples

Feature	Comment
F2: Contrast	- Have discriminating ability. - Rotationally-variant.
F3: Entropy	- Have strong discriminating ability. - Almost rotational-invariant.
F4: Variance	- Have discriminating ability. - Rotational-invariant.
F5: Correlation	- Have strong discriminating ability. - Rotational-dependent feature.

Features on co-occurrence matrix

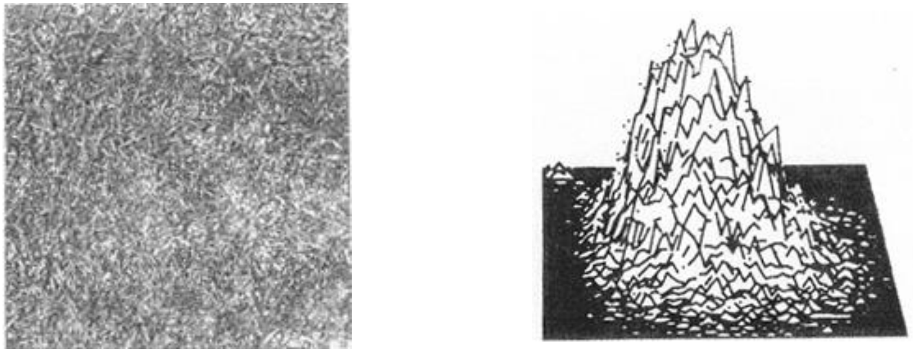
Feature	Comment
F7: Sum average	- Characteristics are similar to ‘variance’/F4 - Rotational-invariant.
F10: Information Measure of Correlation–1	- It has almost similar pattern of ‘sum average’/F7 but vary for various classes - Varies significantly with rotation
F11: Information Measure of Correlation–2	- It is computationally expensive compare to others. - Rotation-variant

Features on co-occurrence matrix

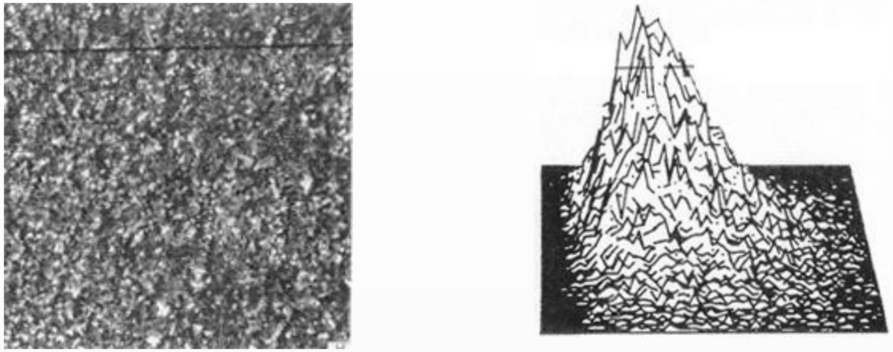
Feature	Comment
F1: Angular Second Moment / Energy	- No distinguishing ability
F6: Inverse Different Moment	- Similar to ‘angular second moment’/F1
F8: Sum Variance	- Similar to ‘variance’/F4
F9: Sum Entropy	- Similar to ‘entropy’/F3

Visualization of co-occurence histograms

- Dependency matrices
  - co-occurrence histograms
  - calculated in a given direction, and distance
  - smoother texture implies more steady response (less dependencies)
  - Coarse texture: dominant response along the main diagonal



Grass



Ivy (borostyán)

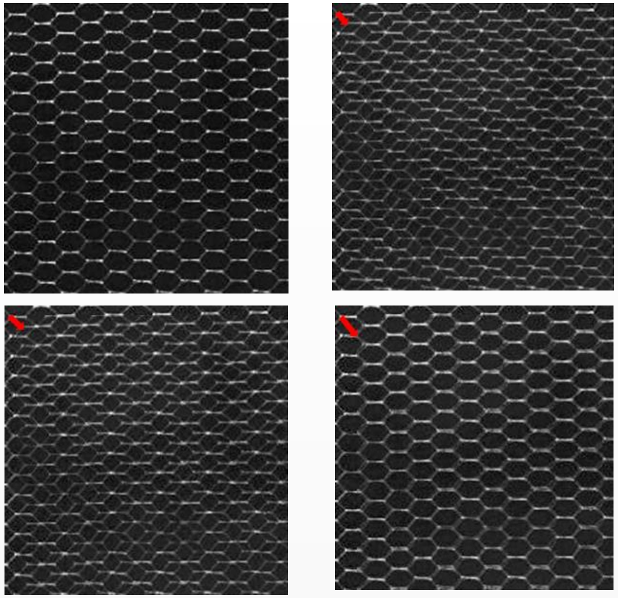
Grayscale dependency matrices for $r = 4, \theta = 0^{\circ}$

Statistical features
Autocorrelation function

- **Definition of autocorrelation:** compare the dot product (energy) of non shifted image with a shifted image

$$R_{II}(\Delta x, \Delta y) = \frac{\sum_{x=0}^{N} \sum_{y=0}^{N} I(x, y) I(x + \Delta x, y + \Delta y)}{\sum_{x=0}^{N} \sum_{y=0}^{N} I^2(x, y)}$$

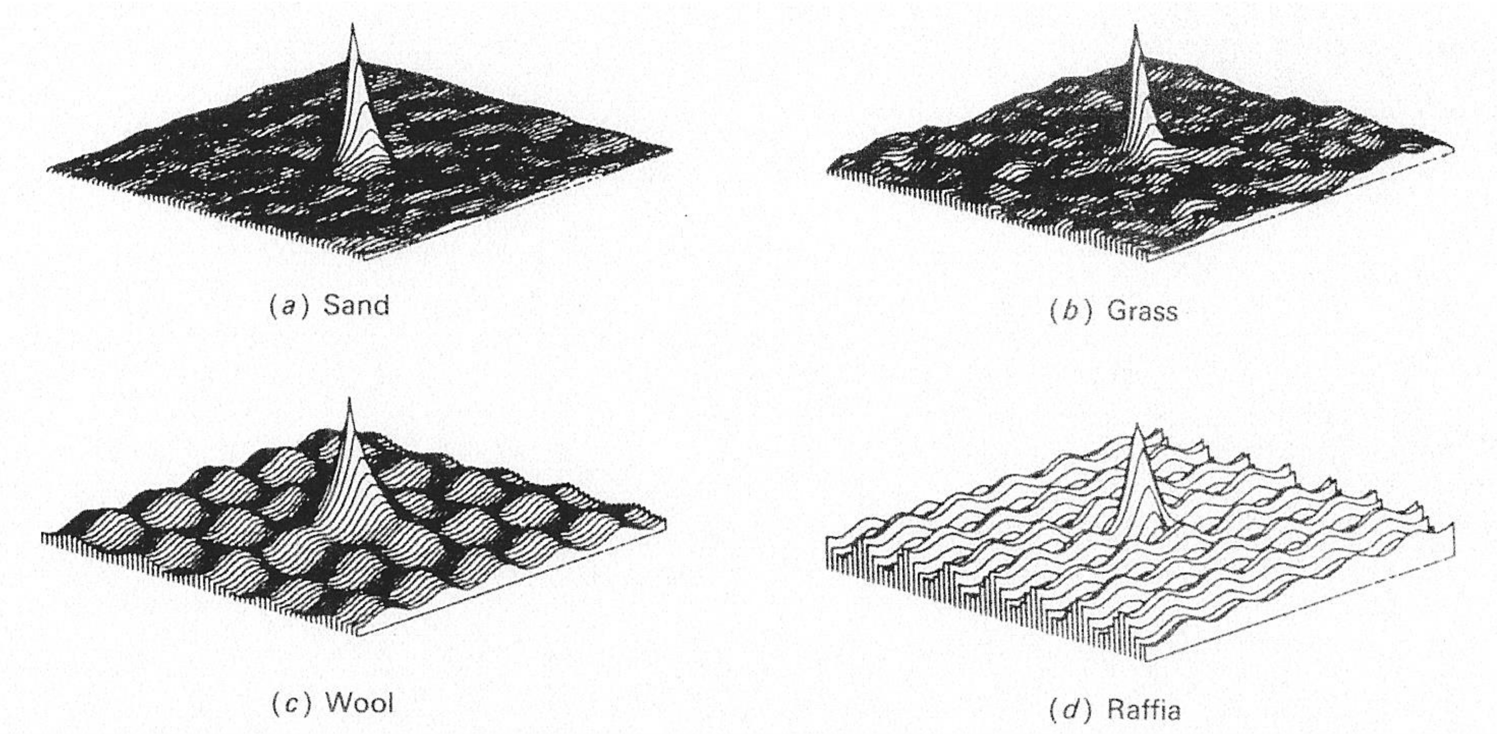
- Features:
  - Autocorrelation function can detect repetitive patterns of texels
  - Also defines fineness/coarseness of the texture



Different shift values

$\Delta x, \Delta y$

Textures – image features



(*a*) Sand

(*b*) Grass

(*c*) Wool

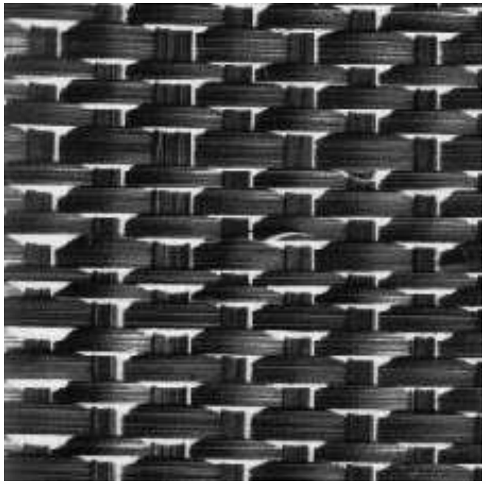
(*d*) Raffia

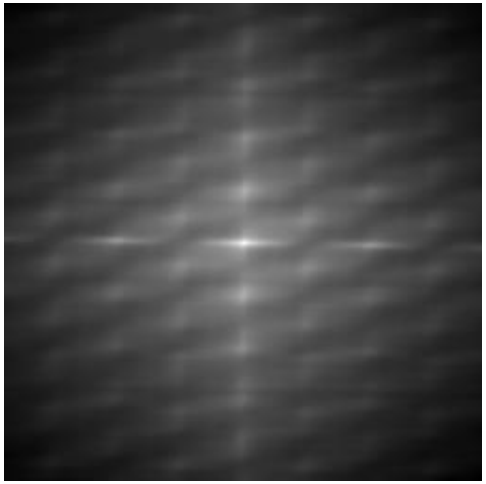
Principle: a coarse pattern texture for the same shift value shows greater autocorrelation than a finer pattern one

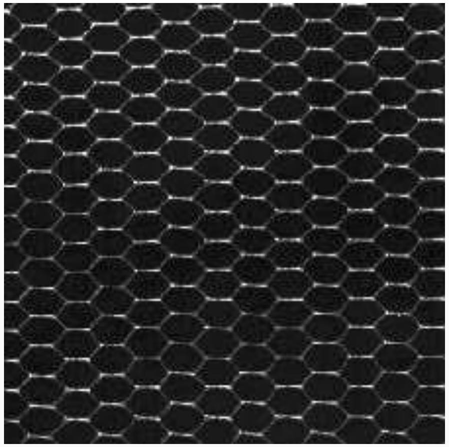
Interpreting autocorrelation

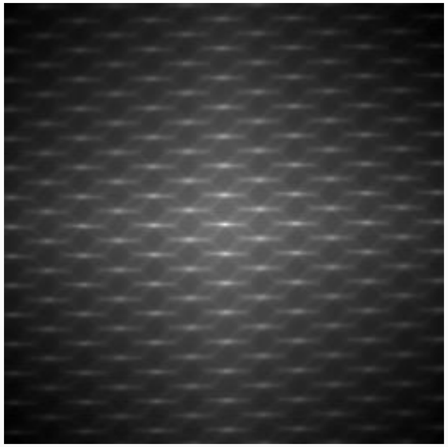
- Coarse texture → function drops off slowly
- Fine texture → function drops off rapidly
- Regular textures → function will have peaks and valleys; peaks can repeat far away from [0, 0]
- Random textures → only peak at [0, 0]; breadth of peak gives the size of the texture

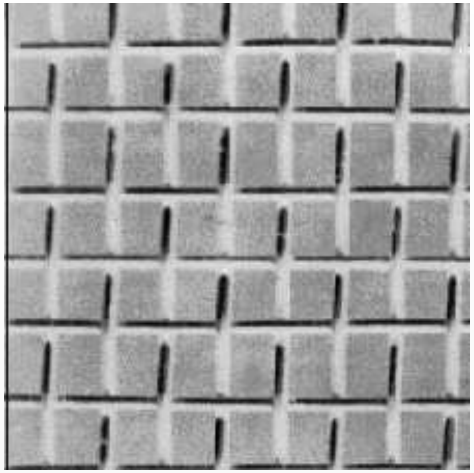
Autocorrelation

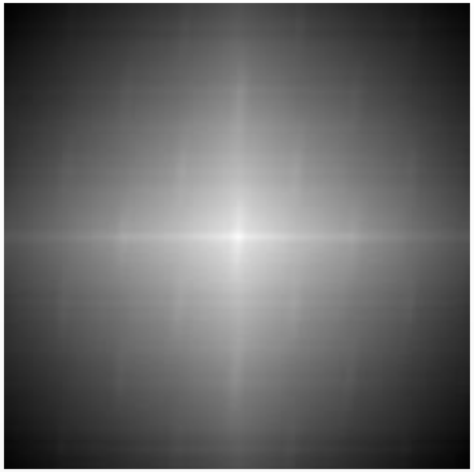




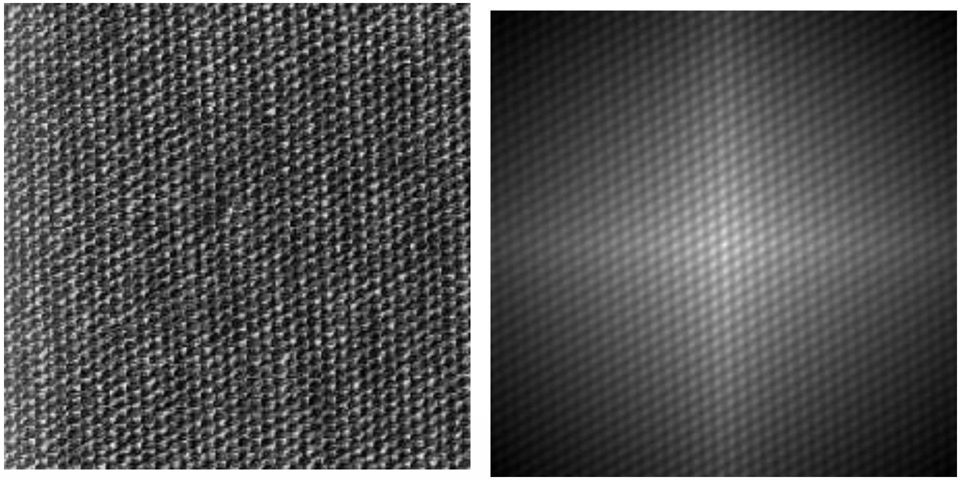


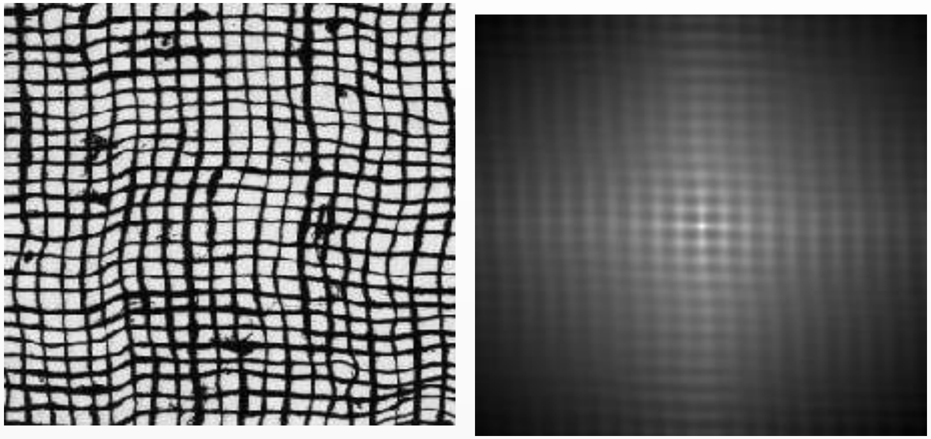


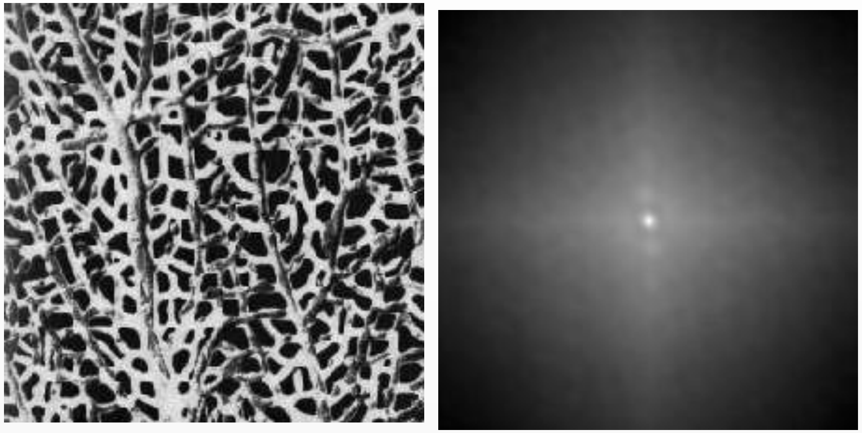




Autocorrelation





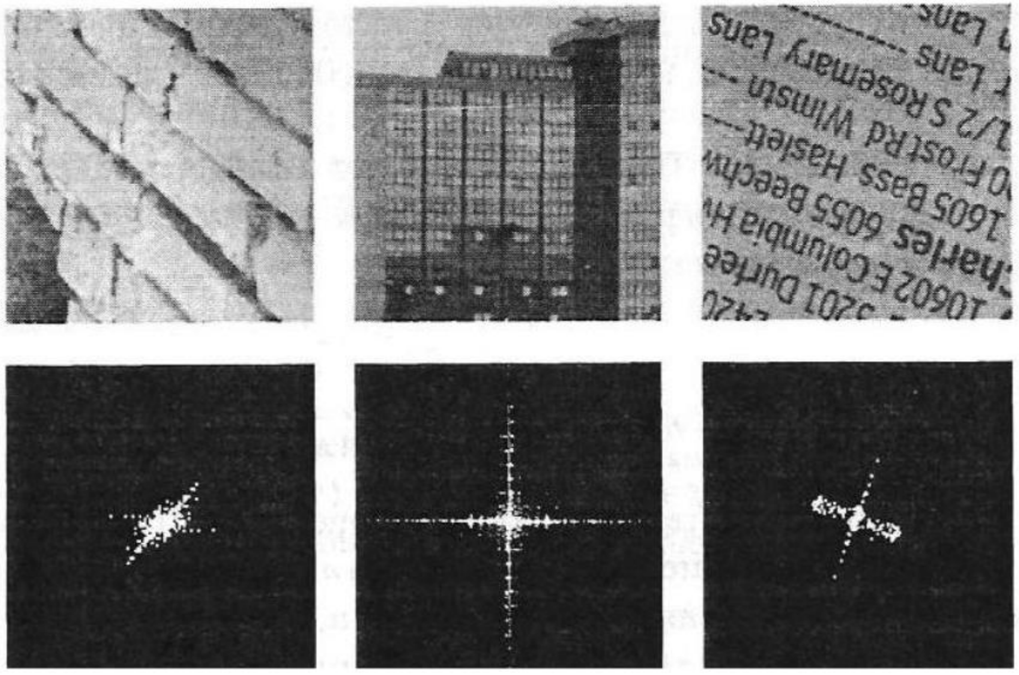


Autocorrelation calculation the Fourier domain

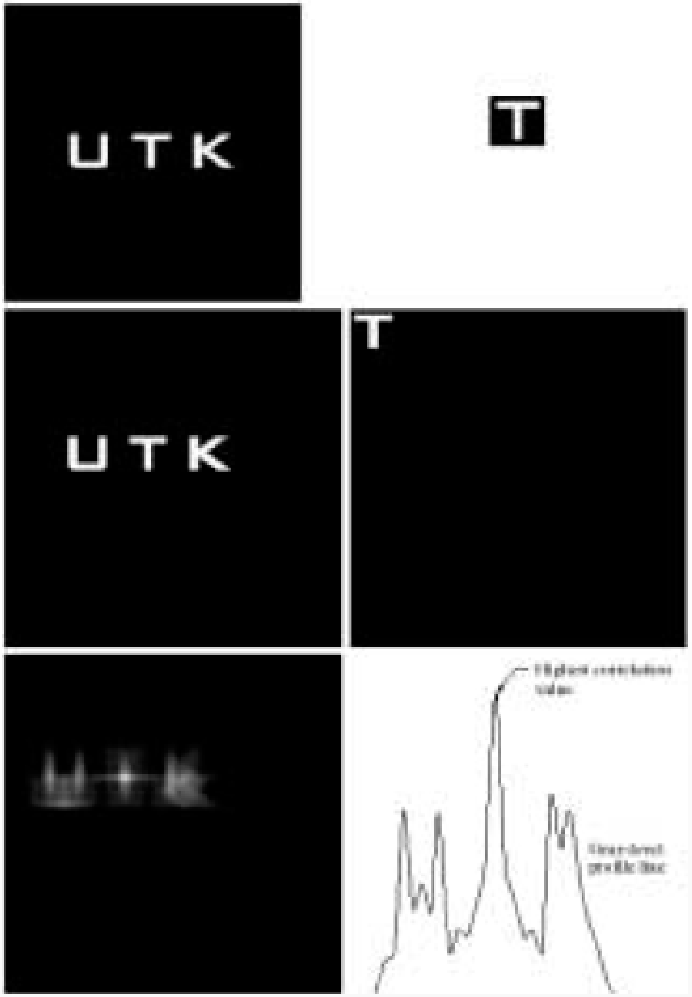
- Wiener-Khinchin Theorem
  - Input image: $I(x, y)$
  - Fourier transform: $\{X(\omega_1, \omega_2)\} = \mathrm{DFT}\{I(x, y)\}$
  - Power spectrum: $P_{XX}(\omega_1, \omega_2) = X(\omega_1, \omega_2) \cdot X^*(\omega_1, \omega_2)$=$|X(\omega_1, \omega_2)|^2$
  - Autocorrelation: inverse Fourier transform of the power spectrum: $\{R_{II}(\Delta x, \Delta y)\} = \mathrm{IDFT}\{P_{XX}(\omega_1, \omega_2)\}$

Fourier domain analysis

- Power spectrum: $X\{\omega_1, \omega_2\} \cdot X^*\{\omega_1, \omega_2\}$= =$|X\{\omega_1, \omega_2\}|^2$
- Concentrated power → regularity
- High frequency power → fine texture
- Directionality → directional texture



Correlation (pattern recognition)



Textures – image features

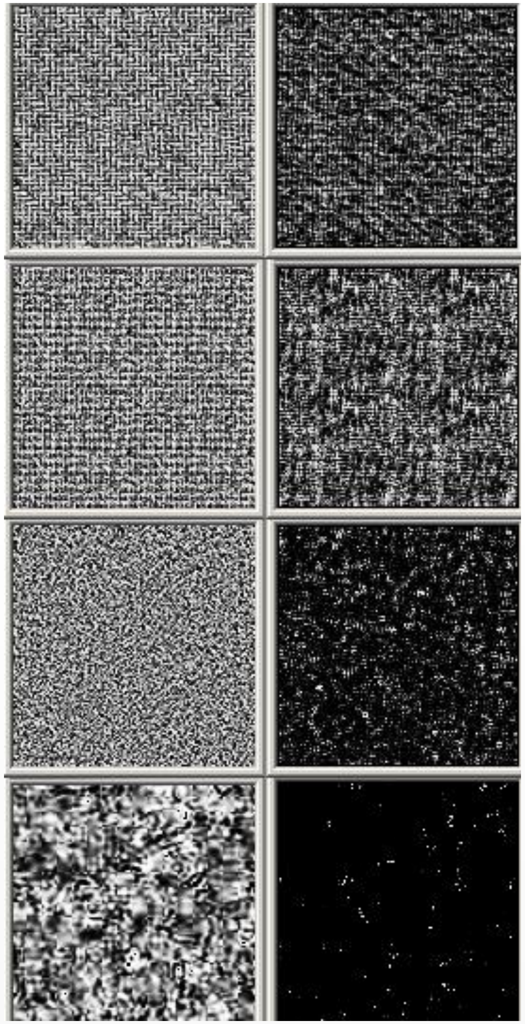
- Edge detection based procedures

E(j,k) = binary edge image obtained by some edge detector (eg. Sobel)

Use low threshold for binarization

Measure of local edge content

$$T(j,k) = \frac{1}{(2w+1)^2} \sum_{m=-w}^{w} \sum_{n=-w}^{w} E(j+m,k+n)$$



Laws filters

- Enhancing micro-structure of the texture
- Main elements:
  - Averaging
  - Edges
  - Points

$$L_3 = \frac{1}{6}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad E_3 = \frac{1}{2}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \qquad S_3 = \frac{1}{2}\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

level detection filter | edge detection filter | spot detection filter

Laws filters

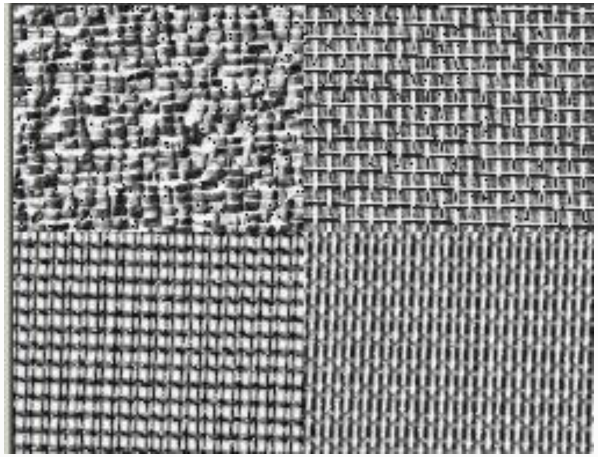
$$H_1 = L_3 L_3^T = \frac{1}{36}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad H_2 = L_3 E_3^T = \frac{1}{12}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad H_3 = L_3 S_3^T = \frac{1}{12}\begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$H_4 = E_3 L_3^T = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad H_5 = E_3 E_3^T = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad H_6 = E_3 S_3^T = \frac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix}$$

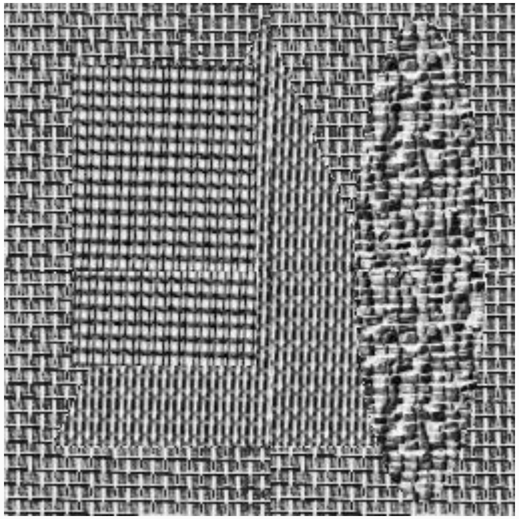
$$H_7 = S_3 L_3^T = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix} \quad H_8 = S_3 E_3^T = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ -2 & 0 & 2 \\ 1 & 0 & -1 \end{bmatrix} \quad H_9 = S_3 S_3^T = \frac{1}{4}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Texture segmentation - Laws

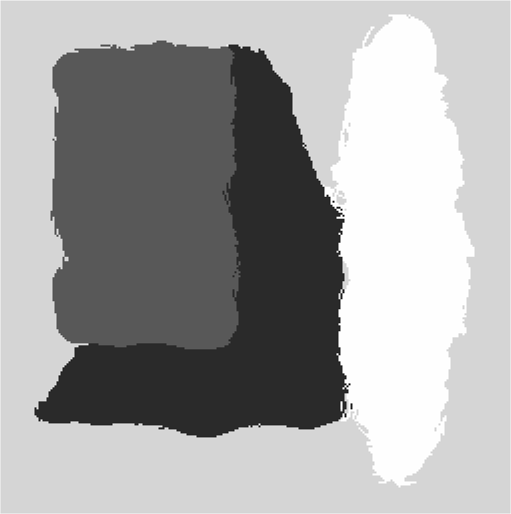
- Training step, thereafter recognizing the trained textures
- Utilizing Law matrices



Training image



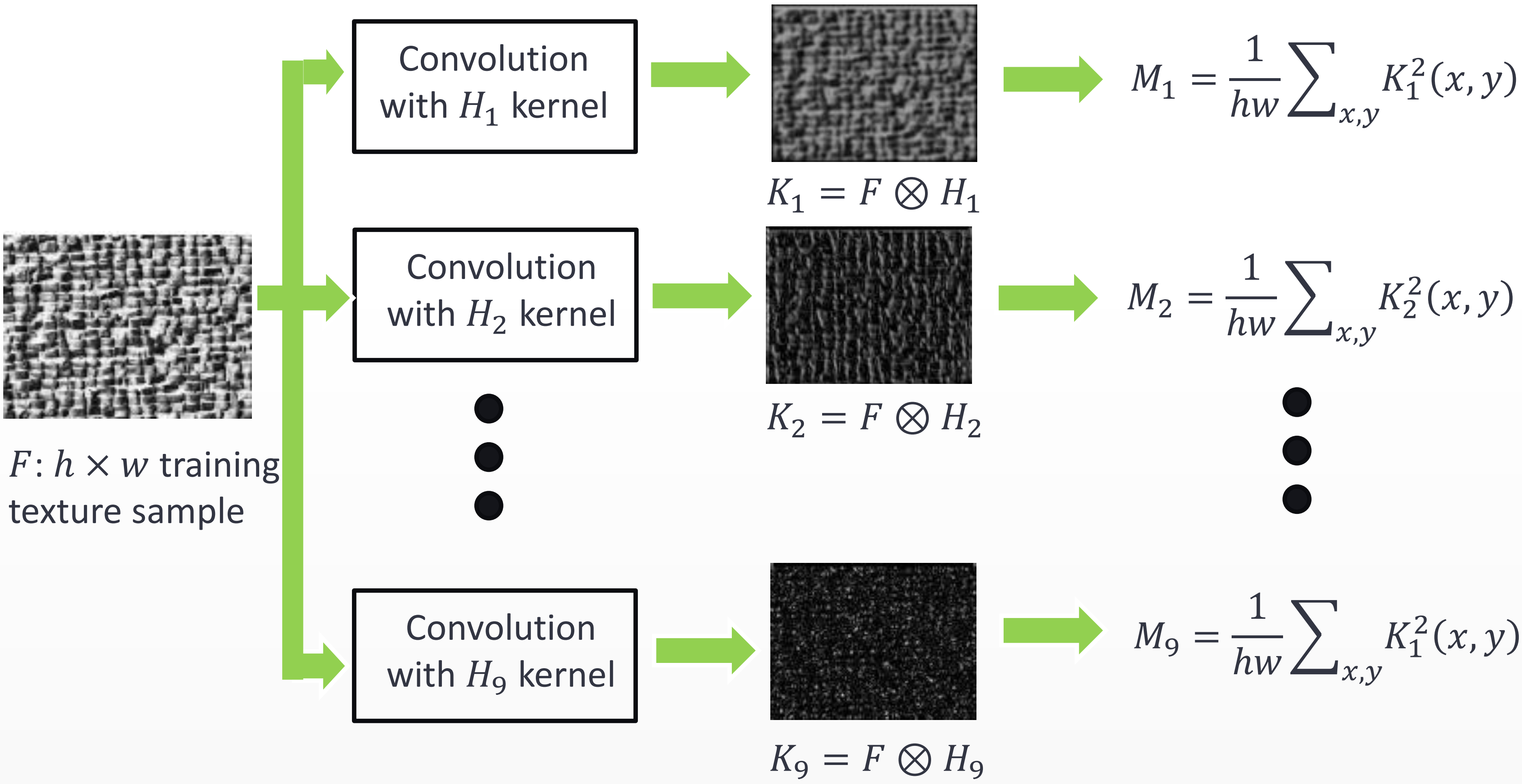
Input test image



Ground truth for the test image

Learning a training texture model

Squared averaging



$F: h \times w$ training texture sample

Convolution with H_1 kernel

Convolution with H_2 kernel

Convolution with H_9 kernel

$K_1 = F \otimes H_1$

$K_2 = F \otimes H_2$

$K_9 = F \otimes H_9$

$$M_1 = \frac{1}{hw} \sum\nolimits_{x,y} K_1^2(x,y)$$

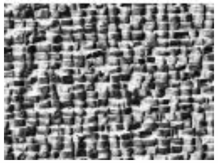
$$M_2 = \frac{1}{hw} \sum\nolimits_{x,y} K_2^2(x,y)$$

$$M_9 = \frac{1}{hw} \sum\nolimits_{x,y} K_1^2(x,y)$$

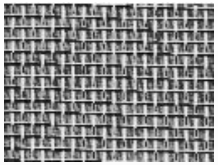
Laws filter– training phase

- Each training texture j is represented by 9 scalars: $M_1^j \ldots M_9^j$

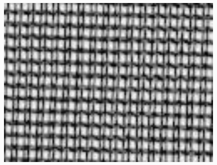
$$M_i^j = \frac{1}{hw}\sum_{x,y}\left(K_i^j(\mathrm{x},\mathrm{y})\right)^2, \text{ where } i = 1, \ldots, 9, j = 1, \ldots 4, \quad \text{and} \quad K_i^j = F_j \otimes K_i,$$



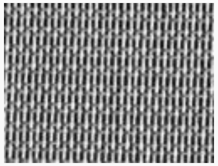
$M_1^1, M_2^1, M_3^1, M_4^1, M_5^1, M_6^1, M_7^1, M_8^1, M_9^1$



$M_1^2, M_2^2, M_3^2, M_4^2, M_5^2, M_6^2, M_7^2, M_8^2, M_9^2$



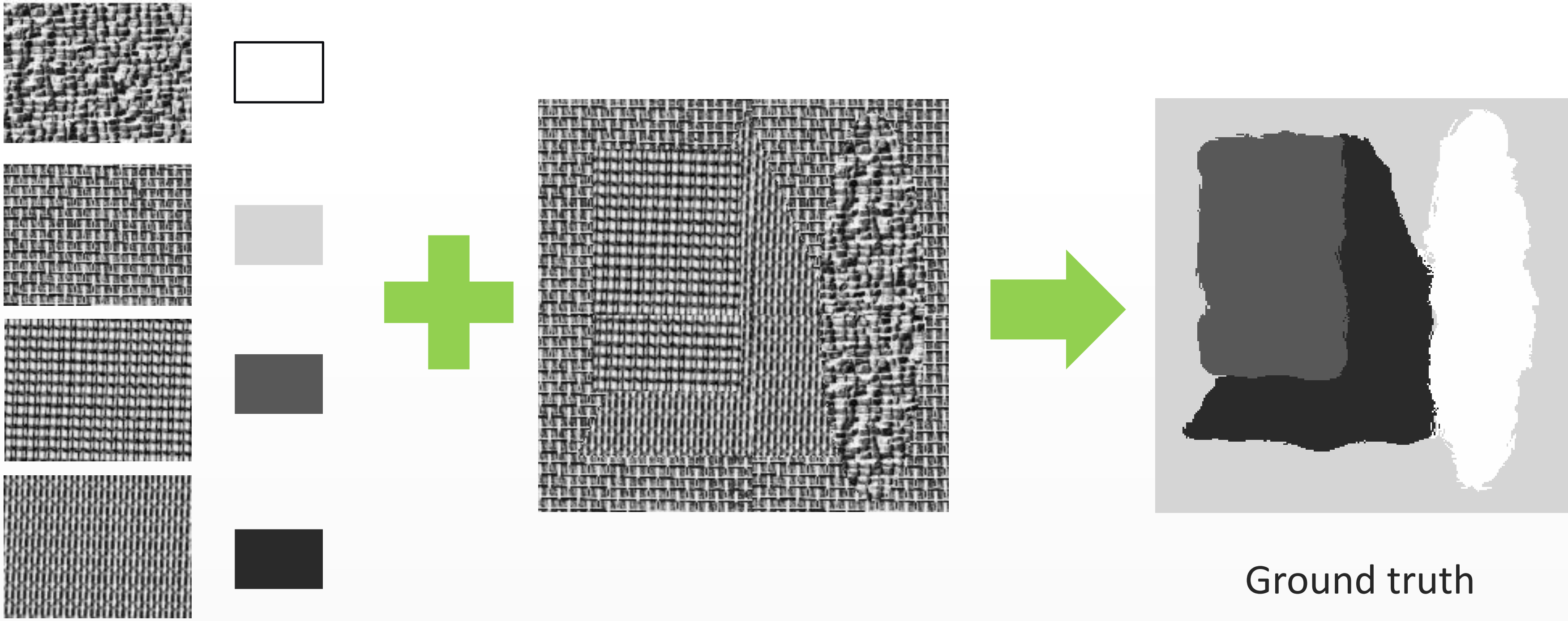
$M_1^3, M_2^3, M_3^3, M_4^3, M_5^3, M_6^3, M_7^3, M_8^3, M_9^3$



$M_1^4, M_2^4, M_3^4, M_4^4, M_5^4, M_6^4, M_7^4, M_8^4, M_9^4$

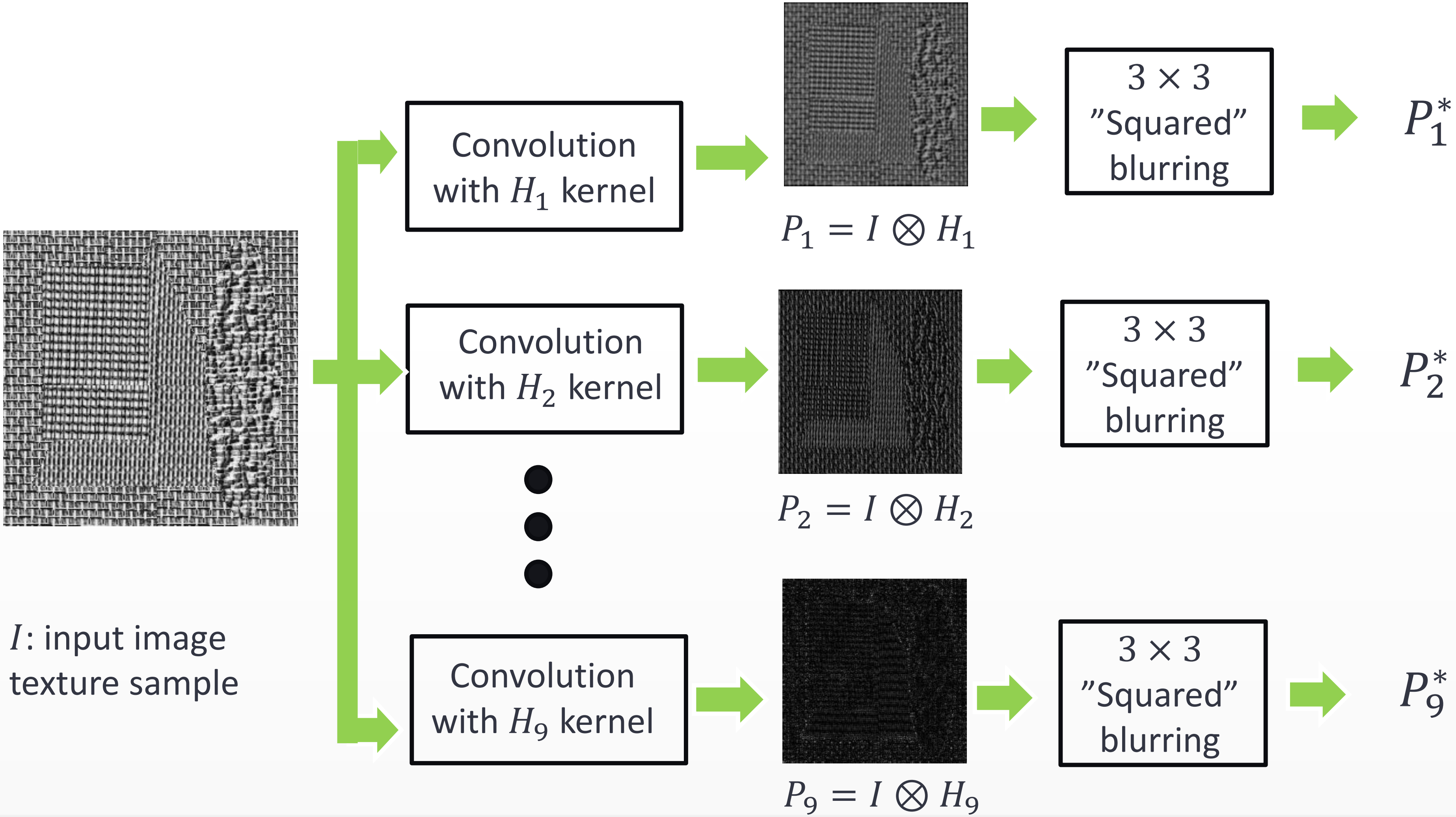
Laws filter– recognition phase

- Input image: consists of arbitrary regions of pre-trained textures



Ground truth

Laws filter– recognition phase



Convolution with H_1 kernel

$P_1 = I \otimes H_1$

3×3 "Squared" blurring

P_1^*

Convolution with H_2 kernel

$P_2 = I \otimes H_2$

3×3 "Squared" blurring

P_2^*

Convolution with H_9 kernel

$P_9 = I \otimes H_9$

3×3 "Squared" blurring

P_9^*

I: input image texture sample

Laws filter– recognition phase

- Pixel level decision of the input map

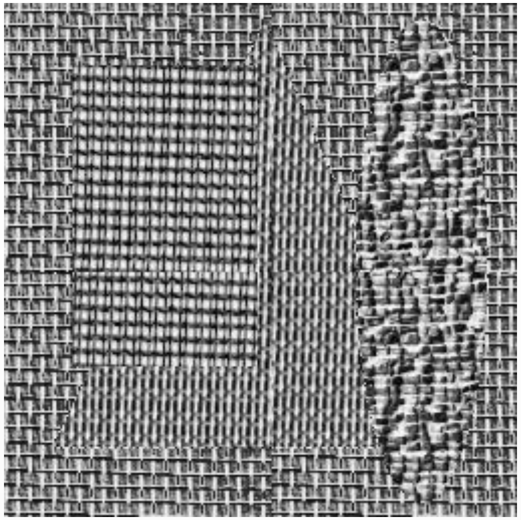
$$\mathrm{ClassMap}(x,y) = \underset{j=1\ldots4}{\operatorname{argmin}} \sum_{i=1\ldots9} \left| P_i^*(x,y) - M_i^j \right|$$

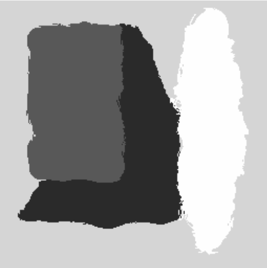
where

$$P_i^*(x,y) = \frac{1}{9} \sum_{r=x-1}^{x+1} \sum_{s=y-1}^{y+1} P_i^2(x+r, y+s)$$

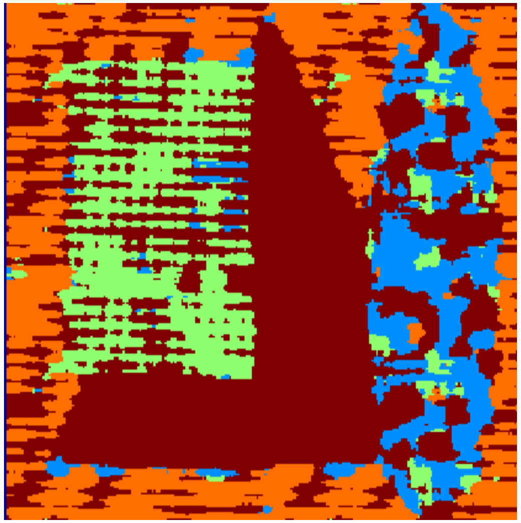
Texture segmentation result

- Output „winner class” maps for the four textures – enhanced with some morphology





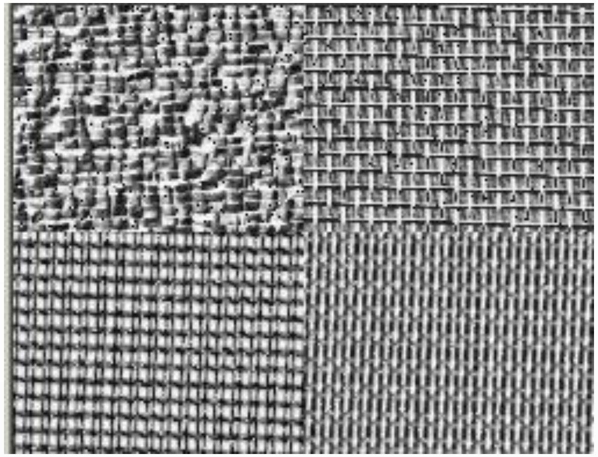
Ground truth

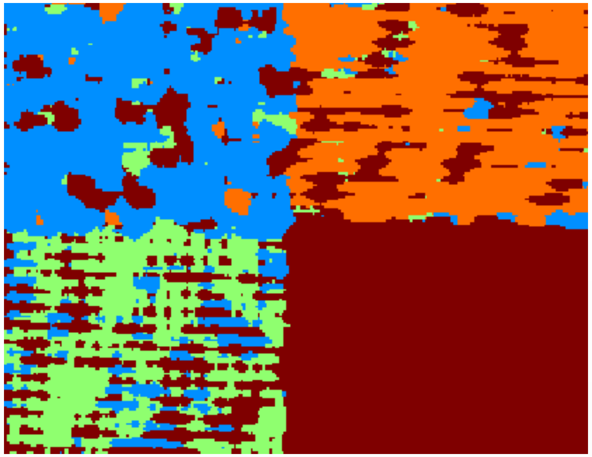


Laws segmentation results

Texture segmentation result

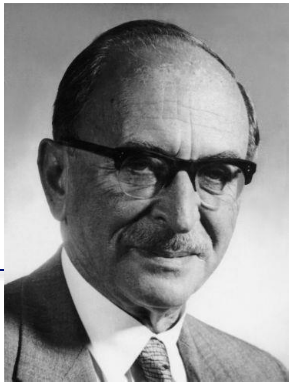
- Output „winner class” maps for the four textures – enhanced with some morphology

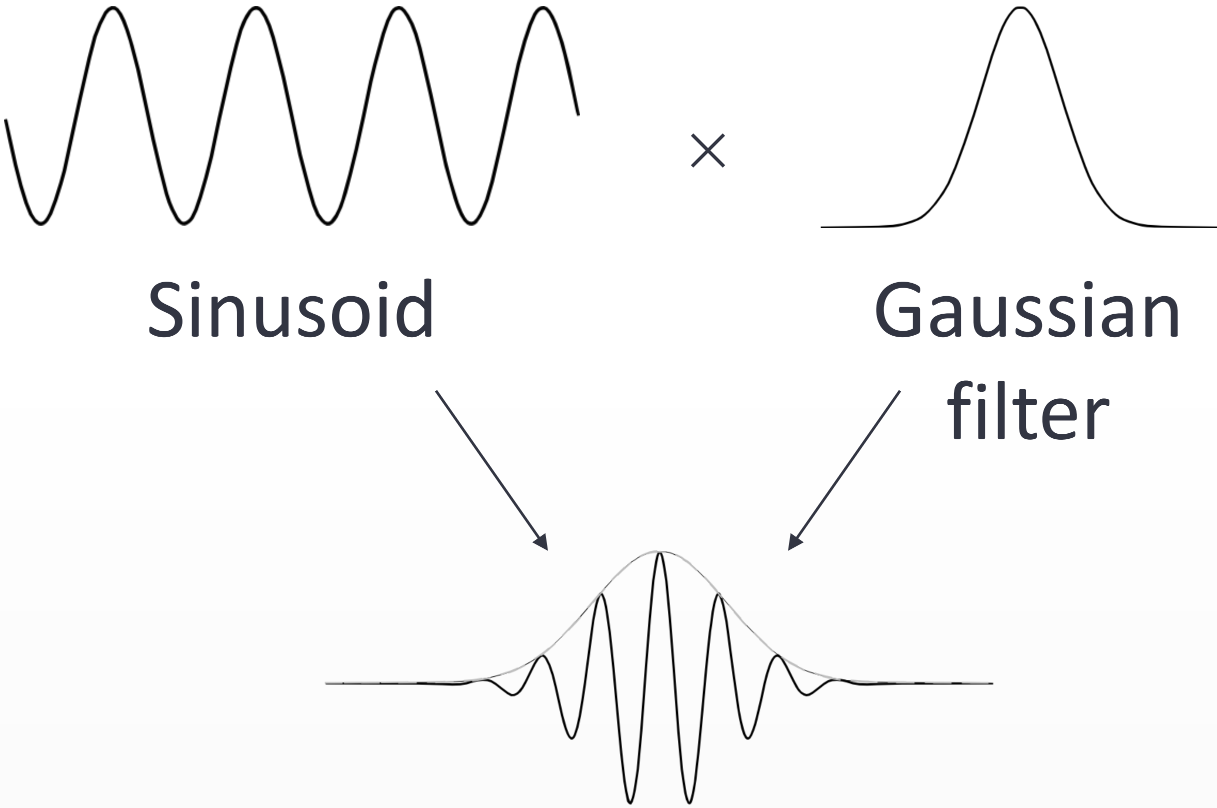




Laws segmentation results

Gabor filters – 1D illustration





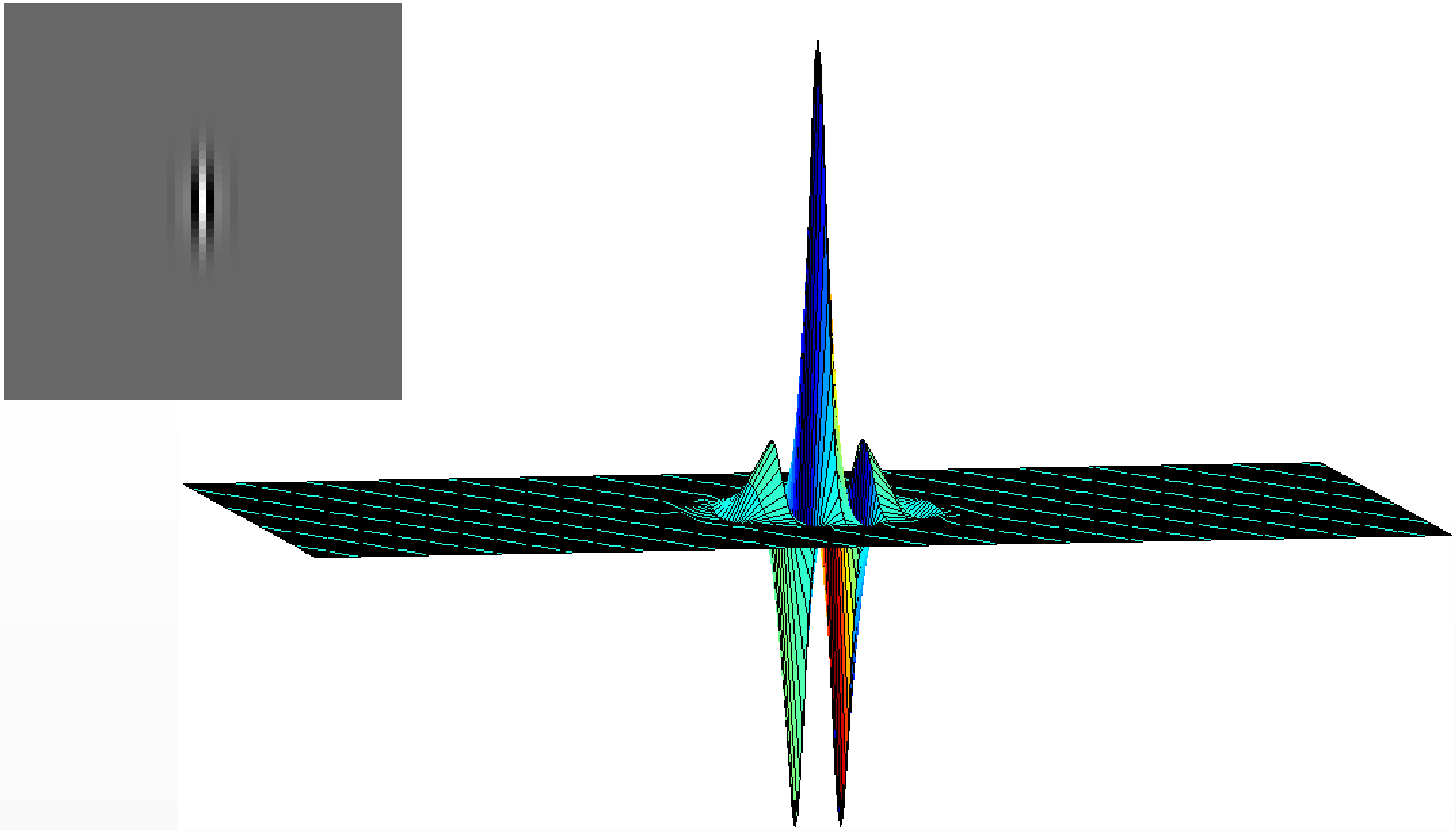
×

Sinusoid

Gaussian filter

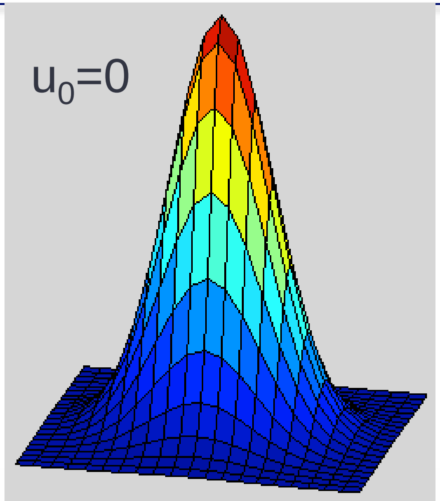
Gabor filter

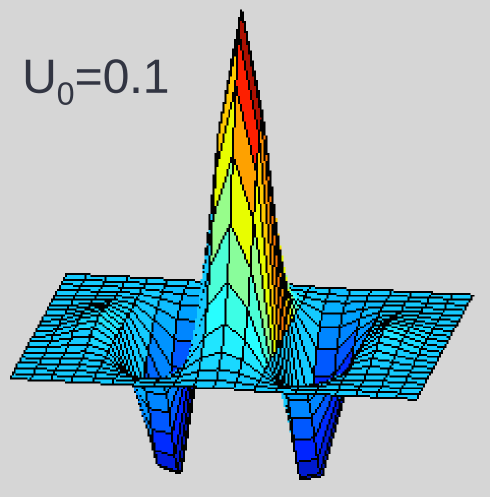
Gabor filters- 2D kernel example

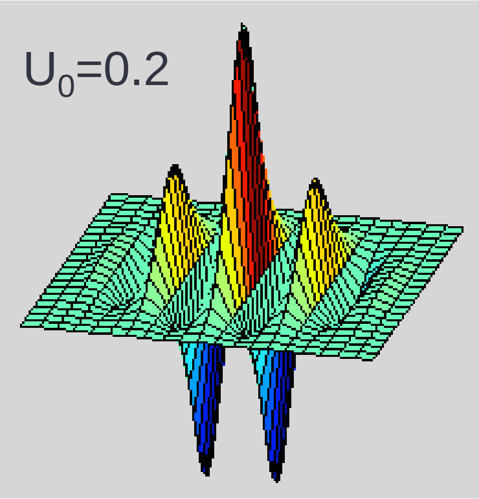


2D - Gabor wavelets

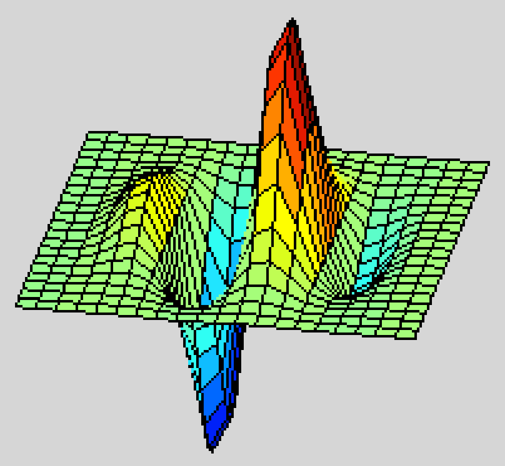
$$\psi_c(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi u_0 x)$$

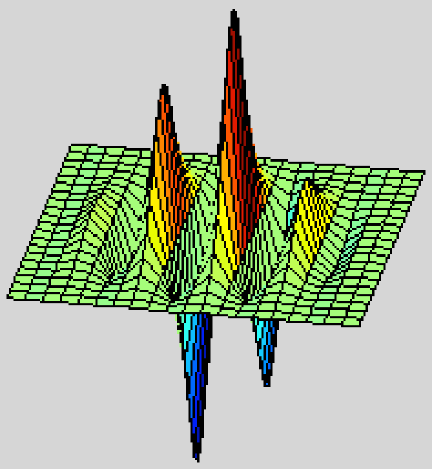


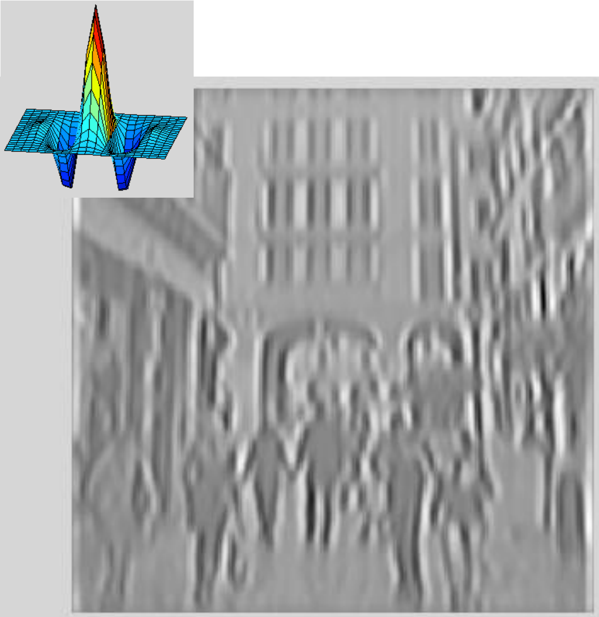




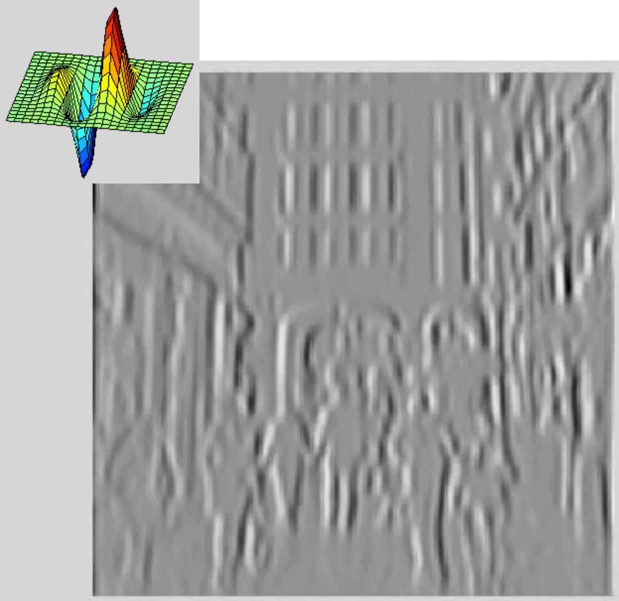
$$\psi_s(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \sin(2\pi u_0 x)$$



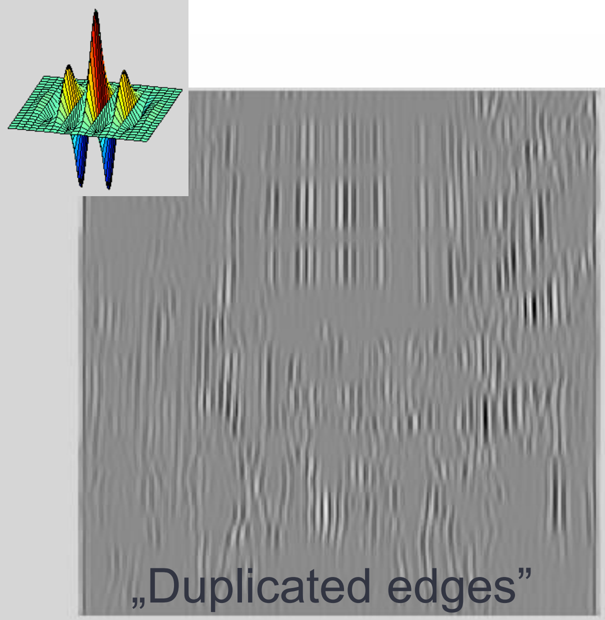




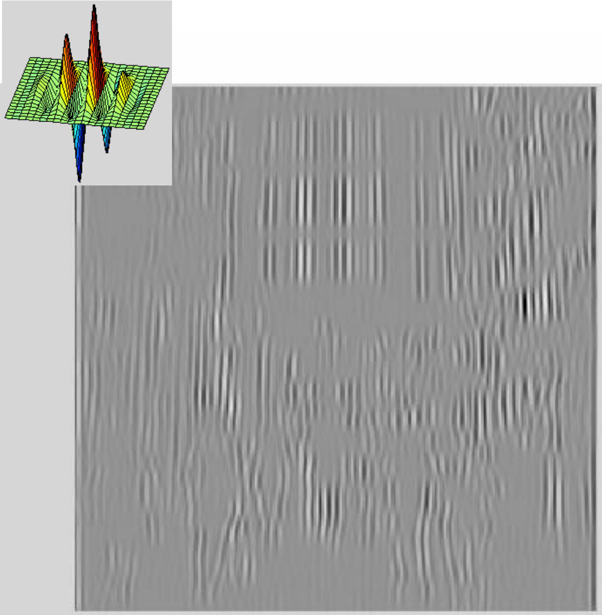
Salient bright thin vertical lines



Dark-bright transitions in horizontal direction



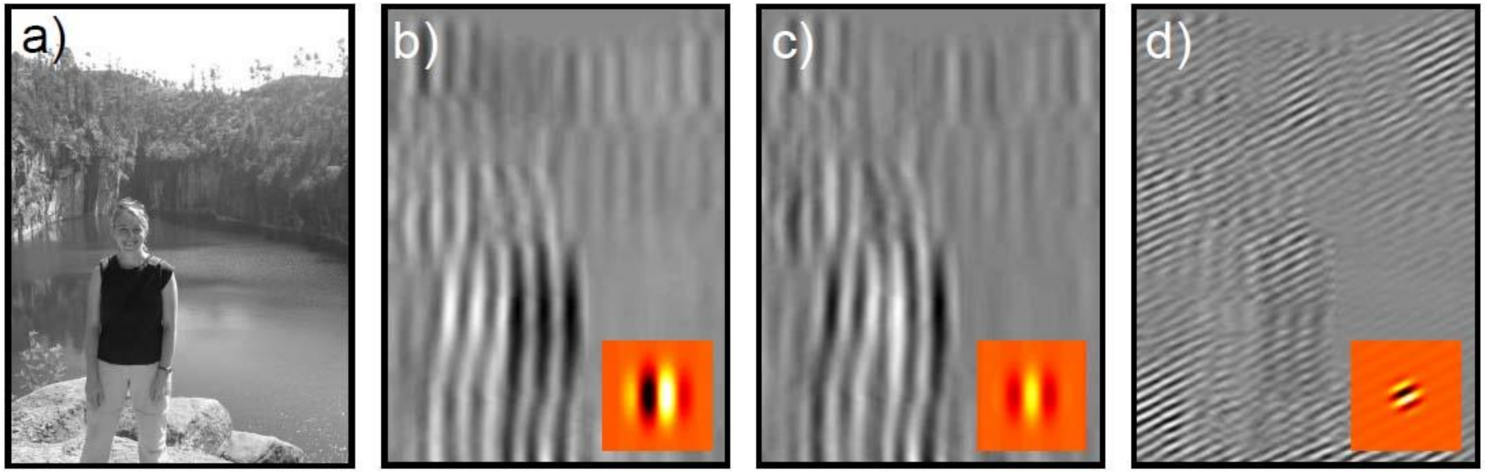
„Duplicated edges”



Application of Gabor filters in image processing

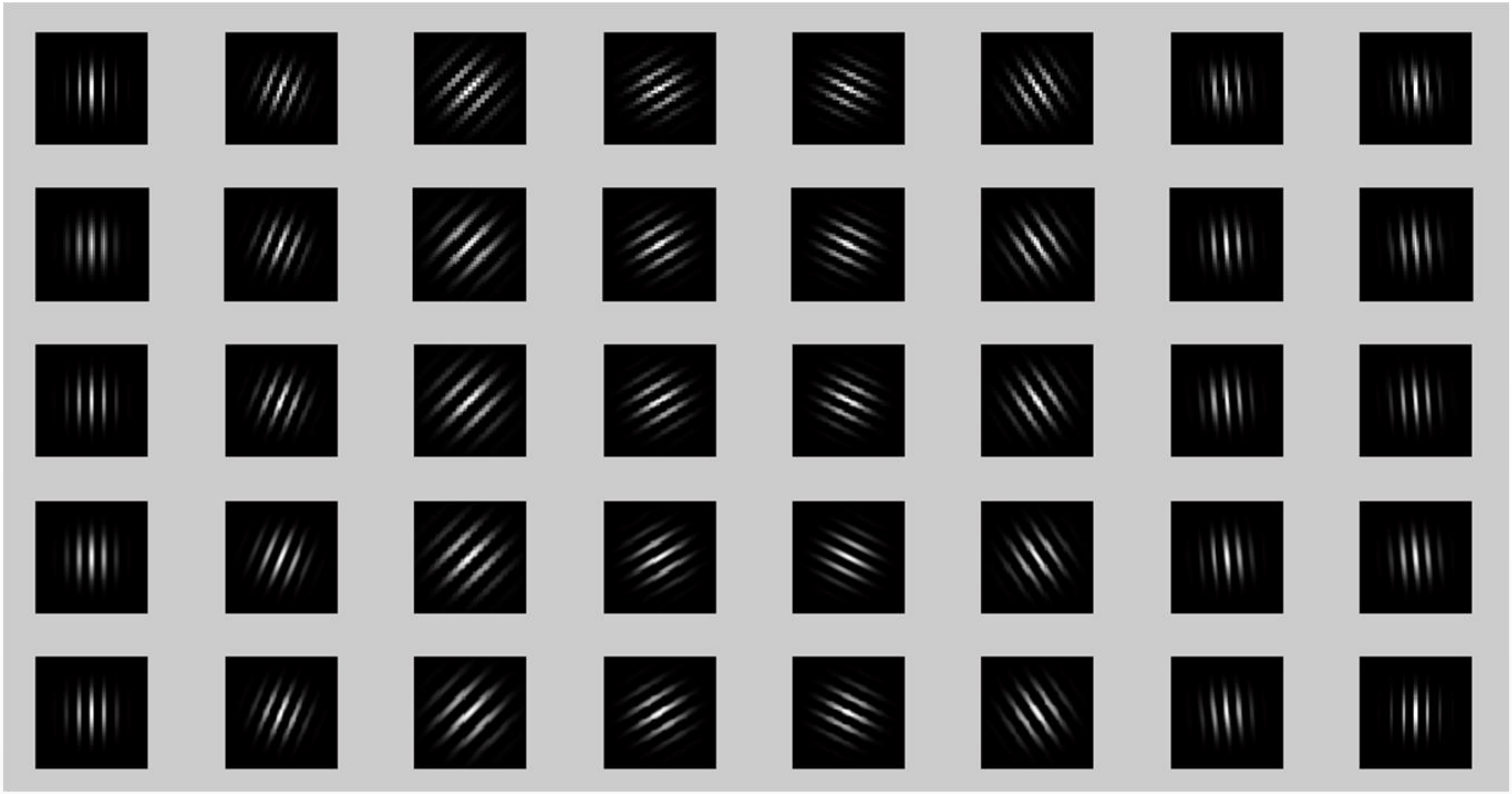
- Gabor filters can selectively highlight specific image elements according to their appropriately set frequency, orientation and phase parameters
- Invariant for additive changes of illumination (in case of asymmetric sinusoid functions)
- Motivation: vision mechanism of mammals – one of the first processing operations of visual stimuli in the brain

Application of Gabor filters in image processing



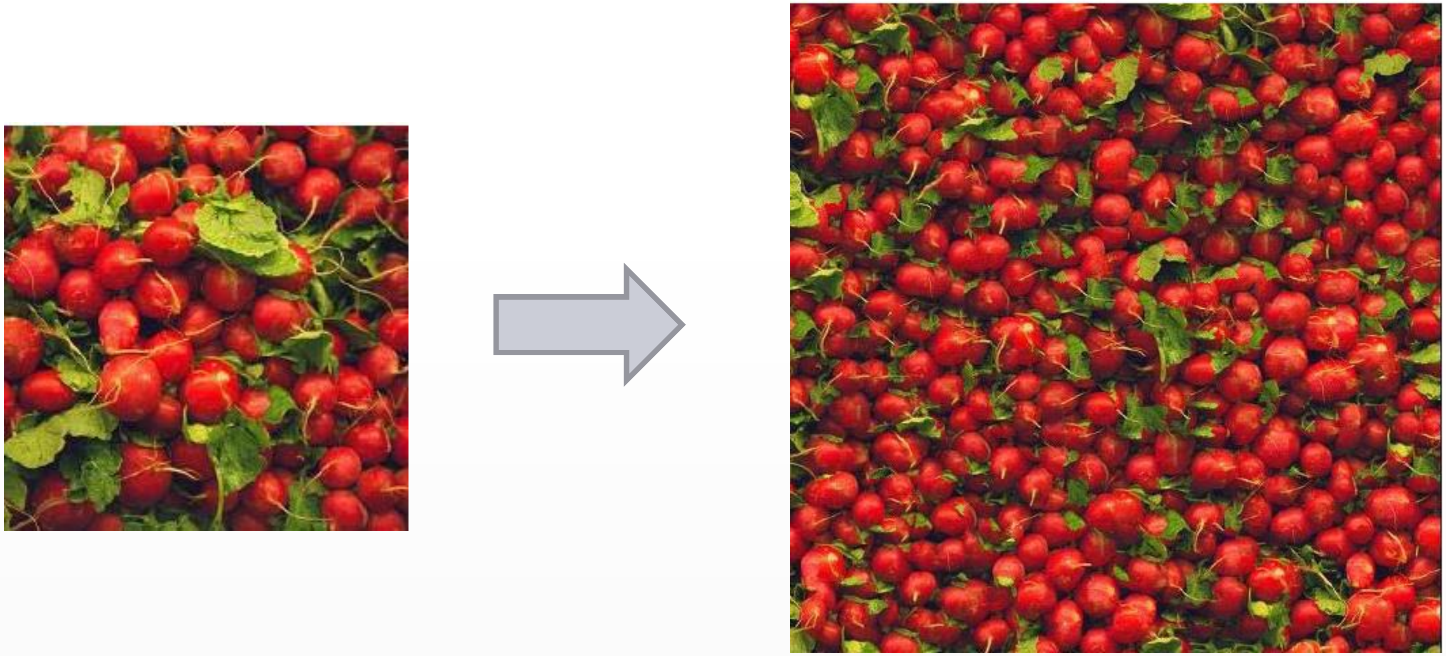
a) Input image. b) Output of a low frequency, horizontal, asymetric Gabor filter. c) Output of a low frequency, horizontal, symetric Gabor filter d) Output of a diagonal Gabor filter

Direction selective Gabor filter bank



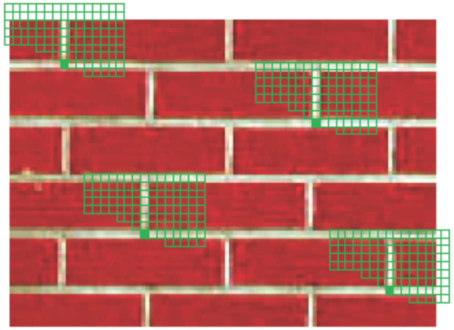
Texture Synthesis

- Given a small sample, generate larger realistic versions of the texture

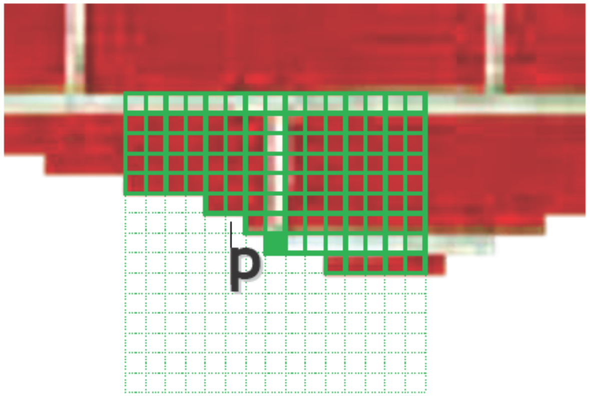


Alexei A. Efrosand Thomas K. Leung, “Texture Synthesis by Non-parametric Sampling,”Proc. International Conference on Computer Vision (ICCV), 1999.

Synthesizing One Pixel



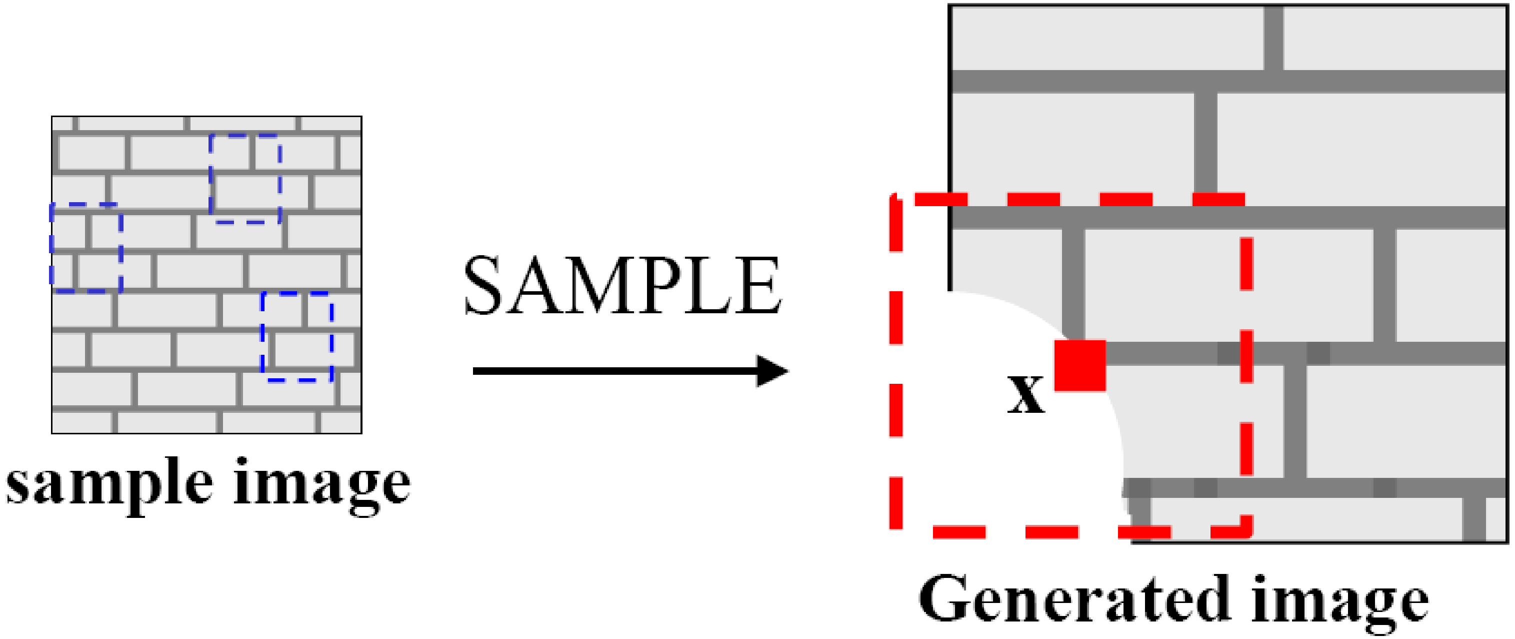
input image



synthesized image

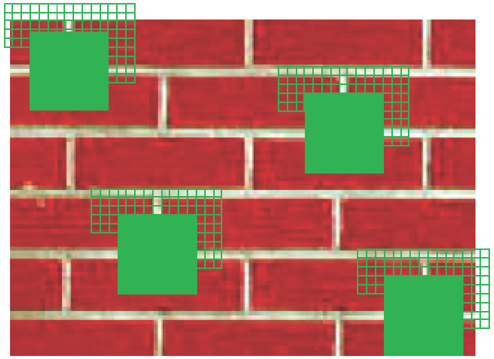
- What is $P(x|\text{neighborhood of pixels around } x)$?
    - Find all the windows in the image that match the neighborhood– consider only pixels in the neighbourhood that are already filled in
    - To synthesize **x**
        - pick one matching window at random
        - assign **x** to be the centerpixel of that window

Really Synthesizing One Pixel

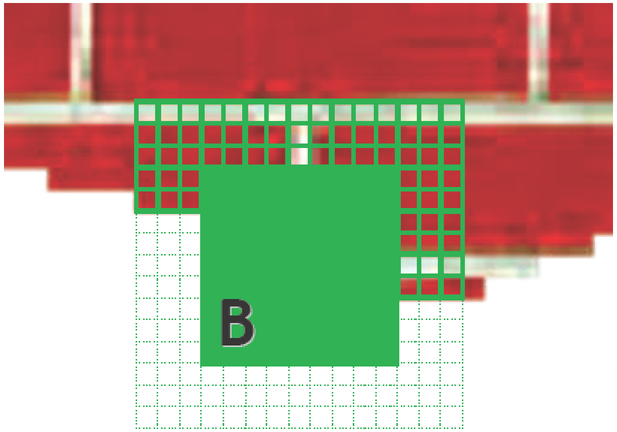


- An exact neighborhood match might not be present
- So we find the **best** matches using SSD error and randomly choose between them, preferring better matches with higher probability

Block-based texture synthesis



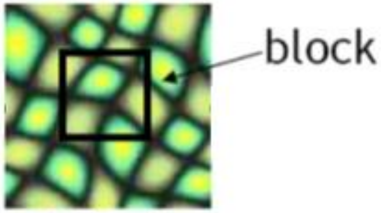
Input image



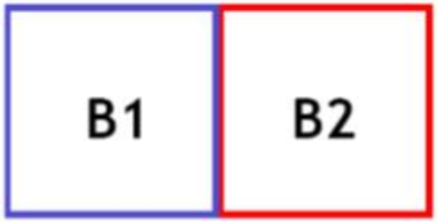
Synthesizing a block

- **Observation**: neighbor pixels are highly correlated
- **Idea: unit of synthesis = block**
  - Exactly the same but now we want P(**B**|N(**B**))
  - Much faster: synthesize all pixels in a block at once

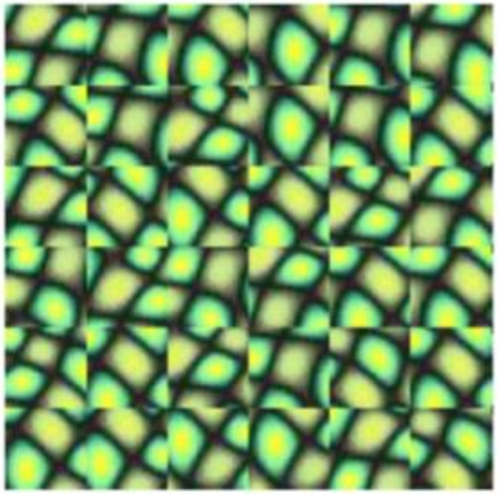
Image Quilting for Texture Synthesis and Transfer', Efros& Freeman, SIGGRAPH, 2001.

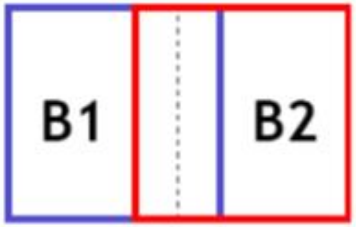


Input texture

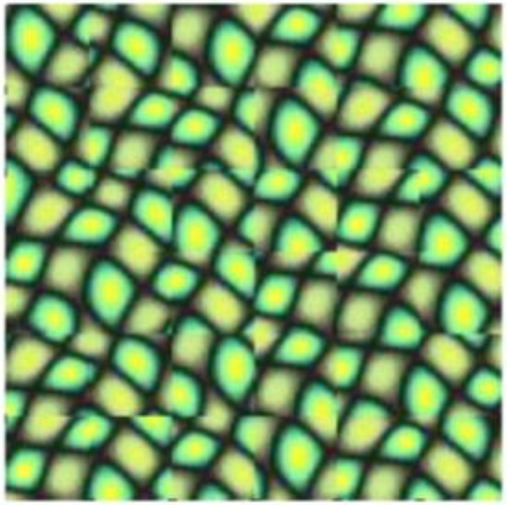


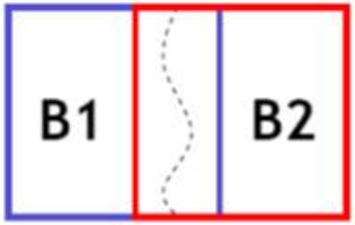
Random placement of blocks



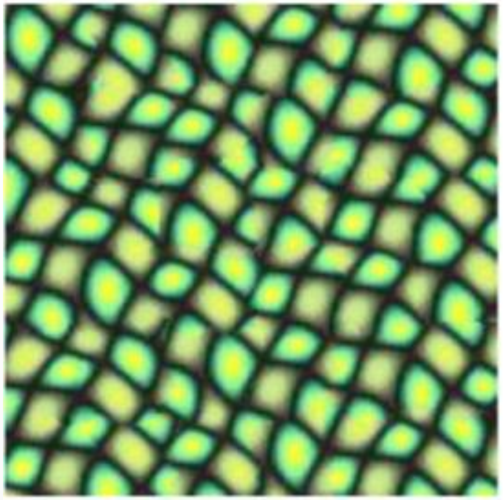


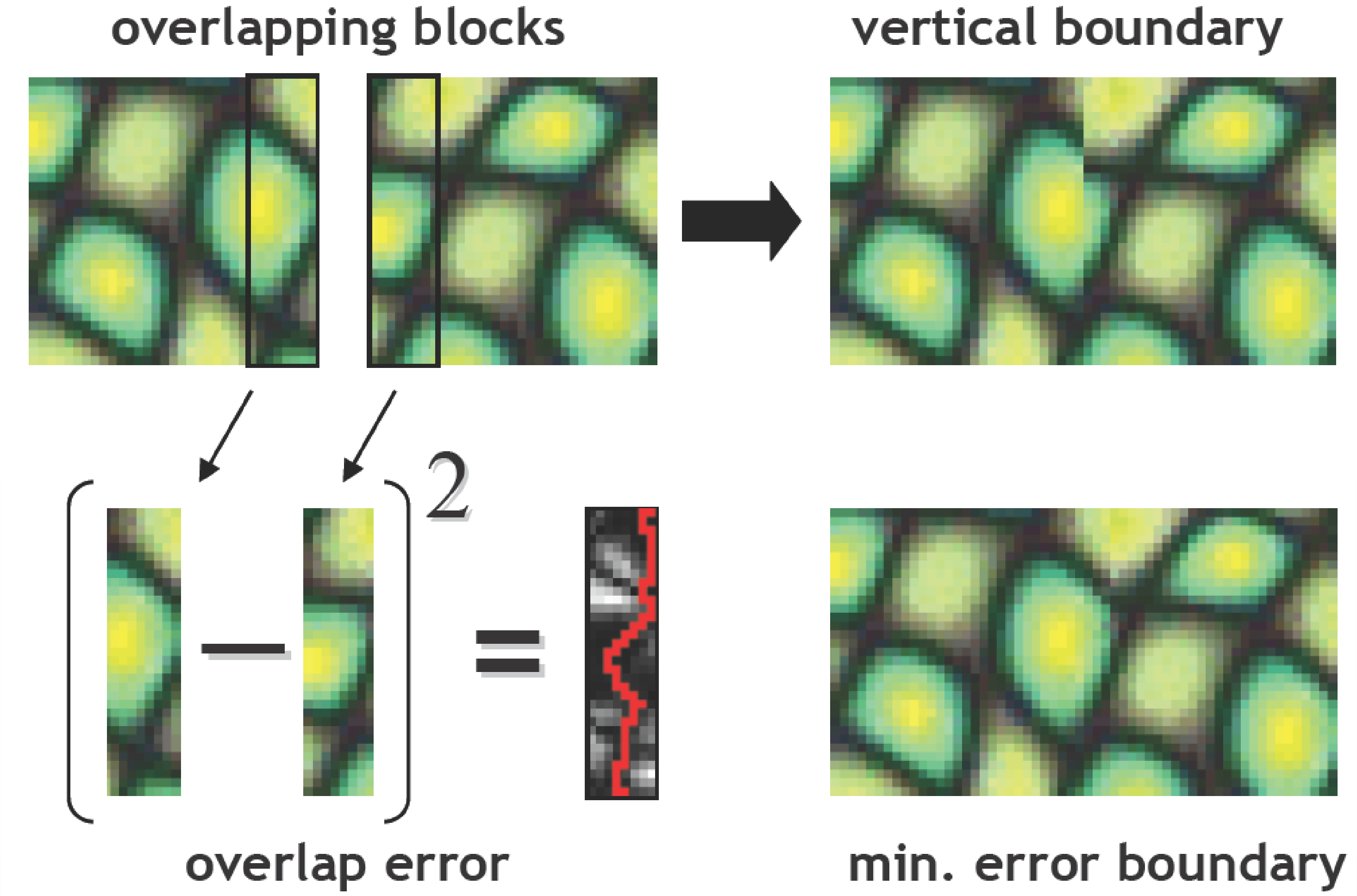
Neighboring blocks constrained by overlap



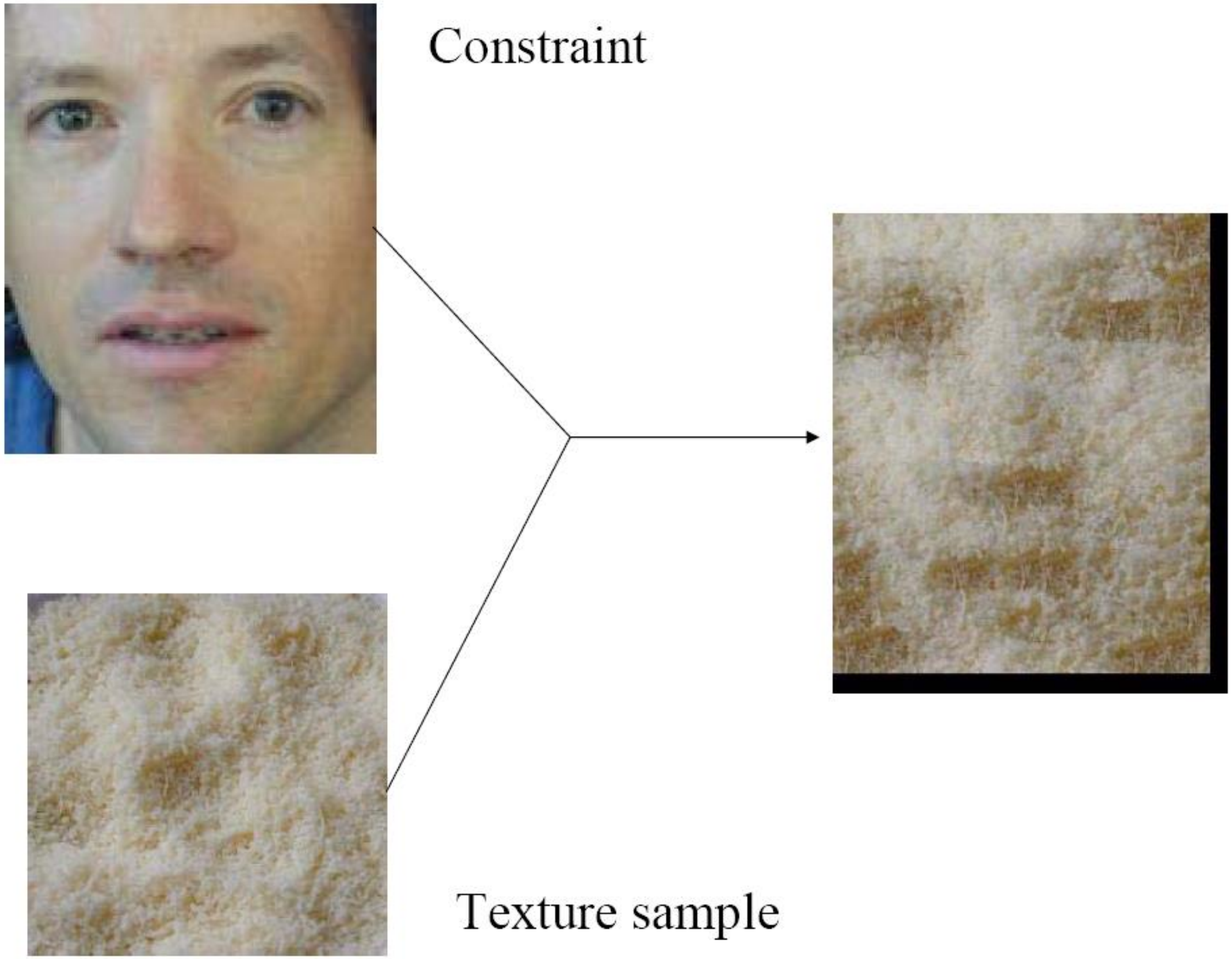


Minimal error boundary cut




overlapping blocks
vertical boundary
2
overlap error
min. error boundary

Texture Transfer



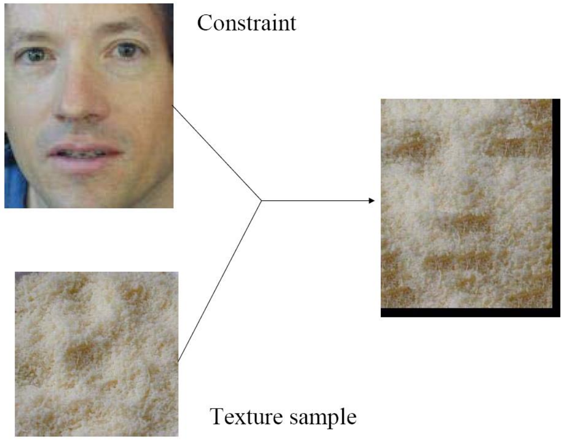
Constraint

Texture sample

Texture Transfer

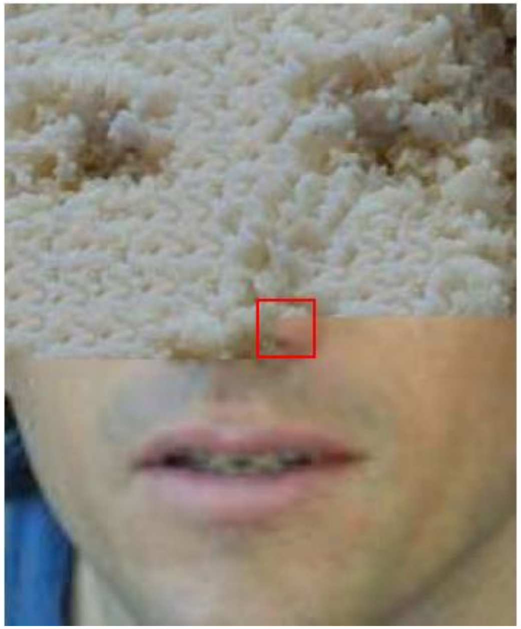
- Each patch satisfy a desired correspondence map C as well as satisfy the texture synthesis requirements.
- C: a spatial map of some corresponding quantity over both the texture source image and a controlling target image.
  - E.g. image intensity, blurred image intensity, local image orientation angles, or other derived quantities.

Here: bright patches of face and bright patches of rice are defined to have a low correspondence error.

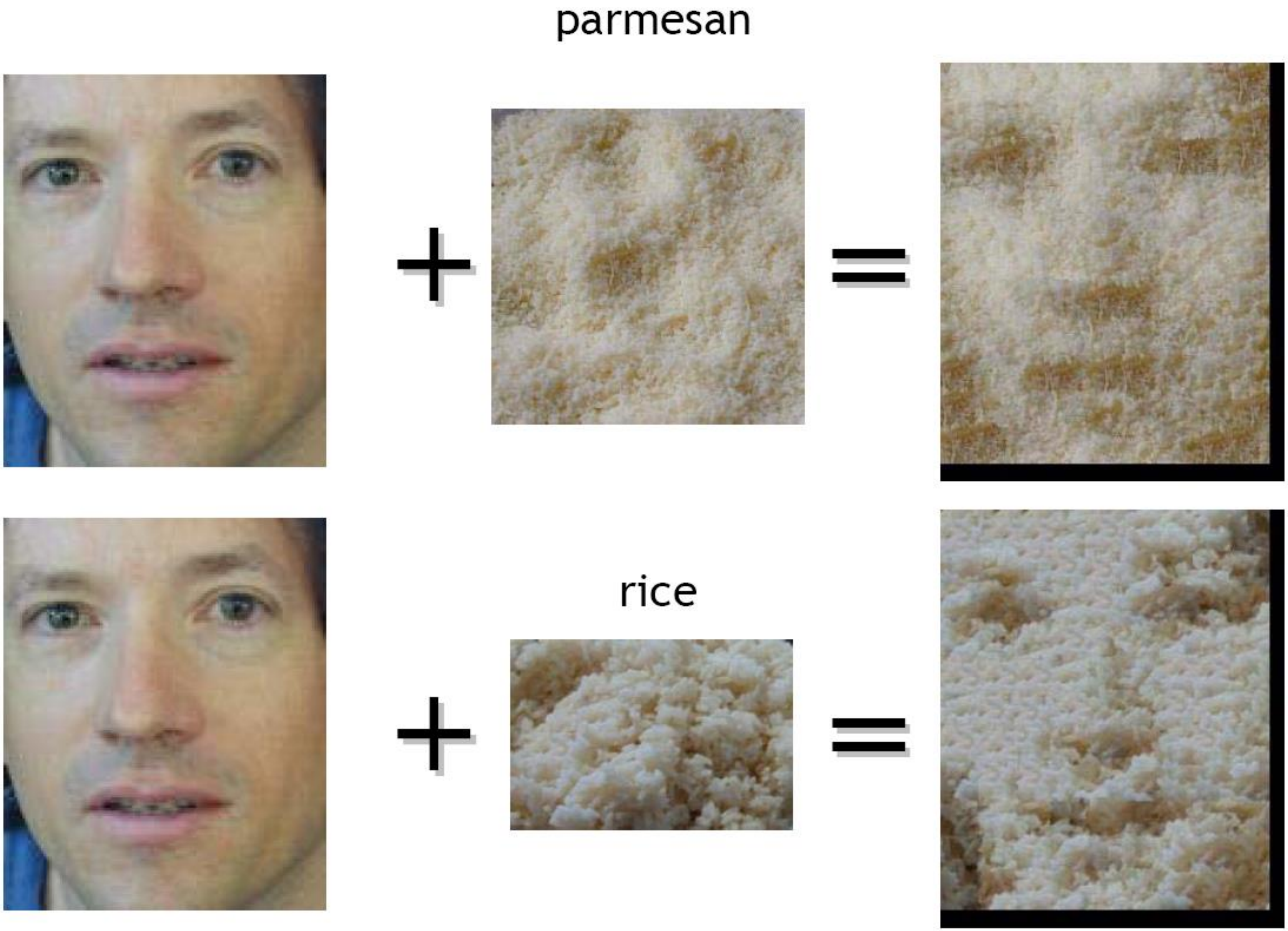


Texture Transfer

- Take the texture from one image and "paint" it onto another object

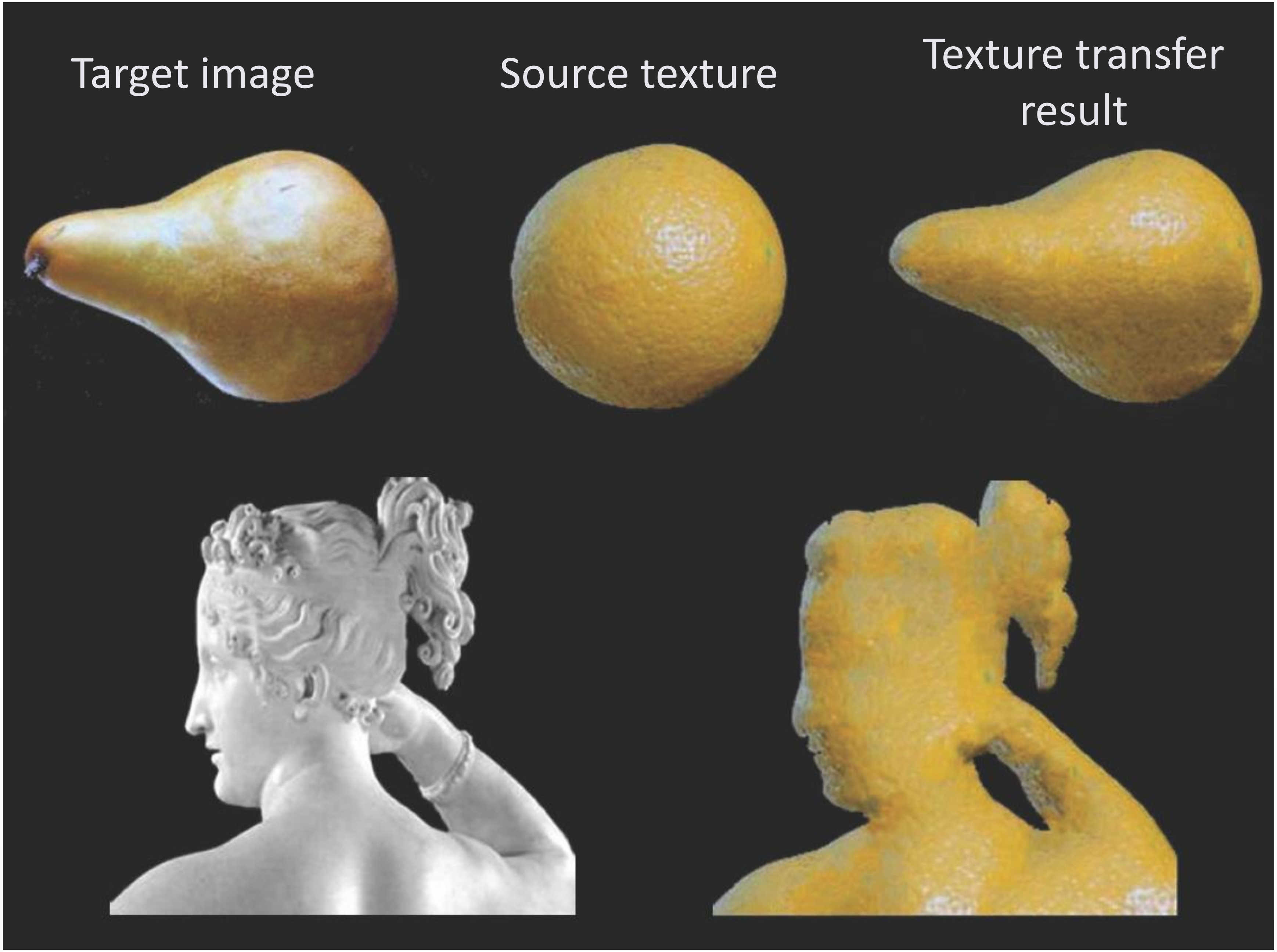


- Same algorithm as before with additional term
  - do texture synthesis on image1, create new image (size of image2)
  - add term to match intensity of image2



parmesan

rice

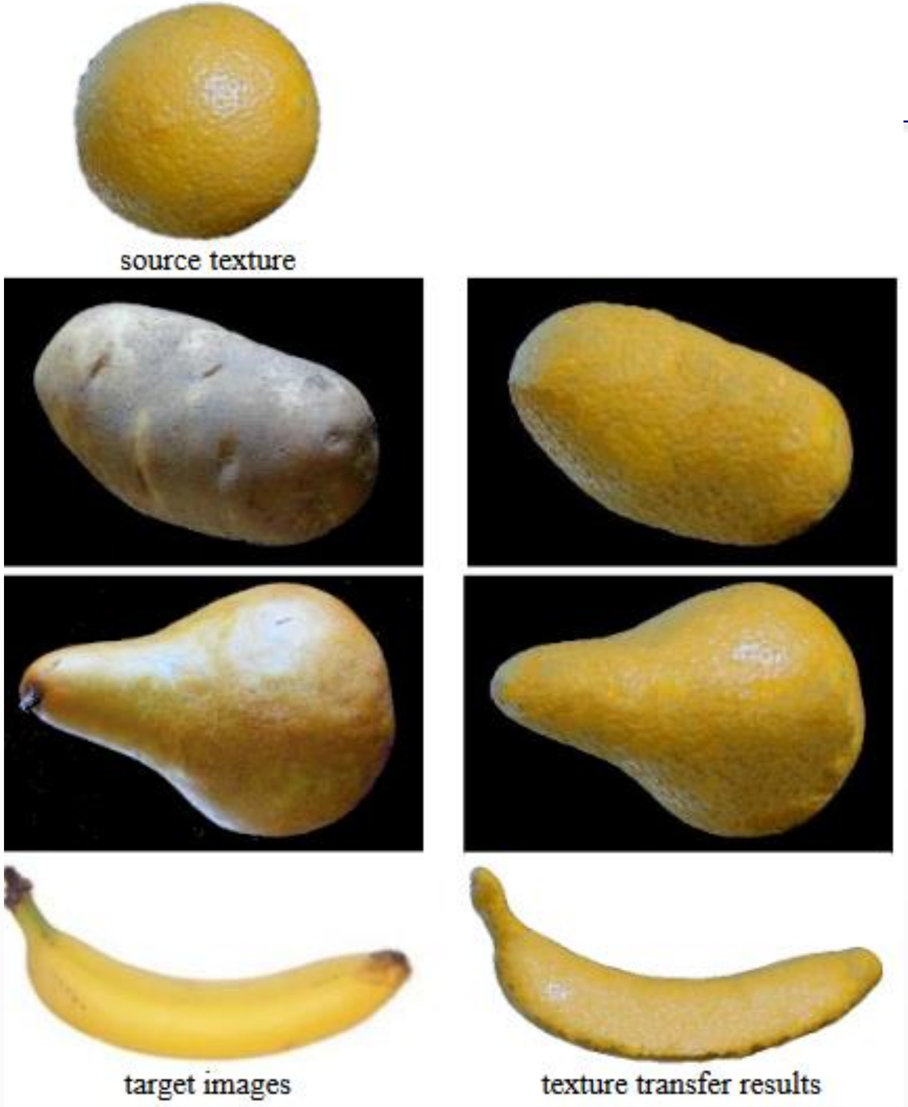


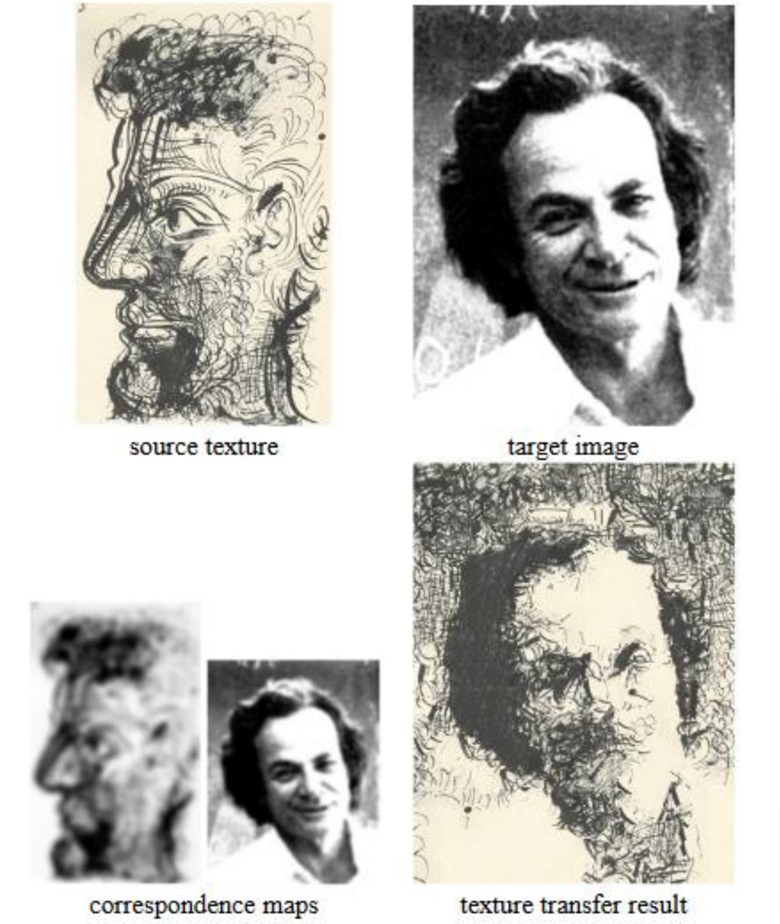
Target image

Source texture

Texture transfer result

Texture transfer





Basic Image Processing

PPKE-ITK

Lecture 6.

Image Recovery

Image Recovery

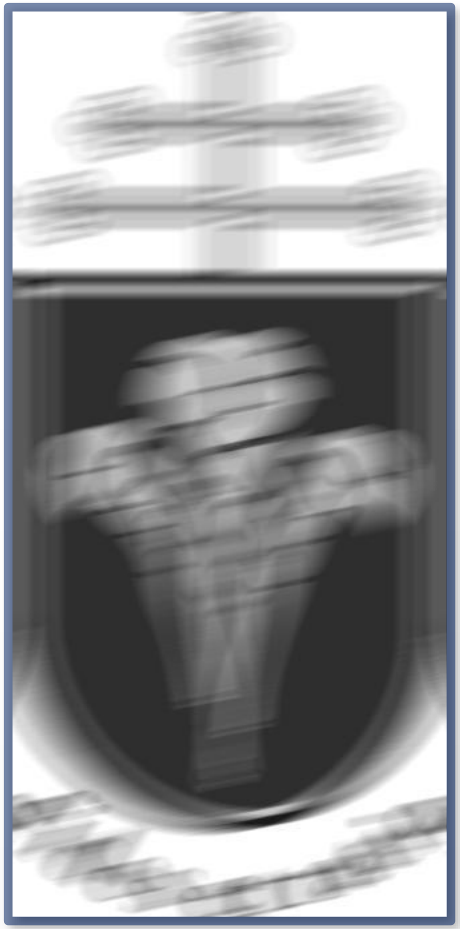
What is Image Recovery?

Recovery vs Enhancement

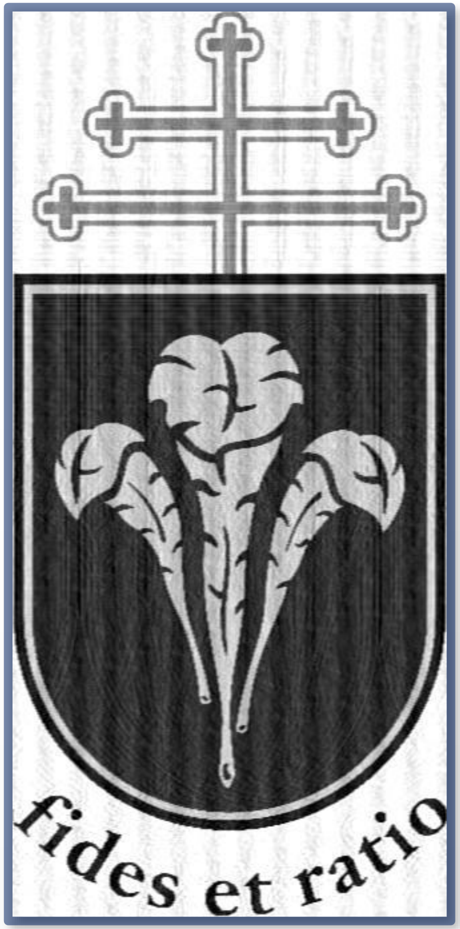
- (Recap) **Image enhancement** is the manipulation or transformation of the image to **improve the visual appearance** or to **help further** automatic **processing steps**.
  - We don't add new information to the image, just make it more visible (e.g. increasing contrast) or highlight a part of it (e.g. dynamic range slicing).

- **Recovery:** the **modeling** and **removal** of the degradation the image is subjected to, based on some **optimality criteria**.
  - In case of recovery there is a ***degradation we want to remove***, lost ***information we want to recover*** (e.g. make blurred text readable again). It is done by modeling the degradation and making assumptions about the degradation and the original image.

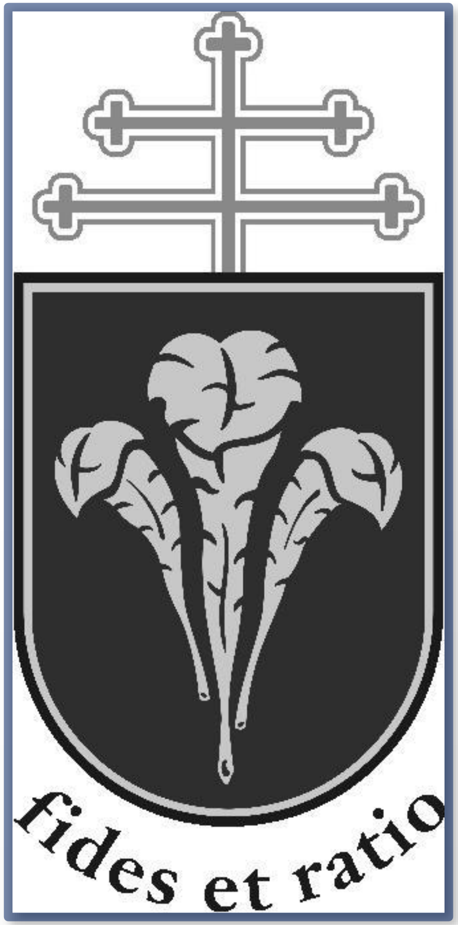
Image Recovery Examples



Blurred Image



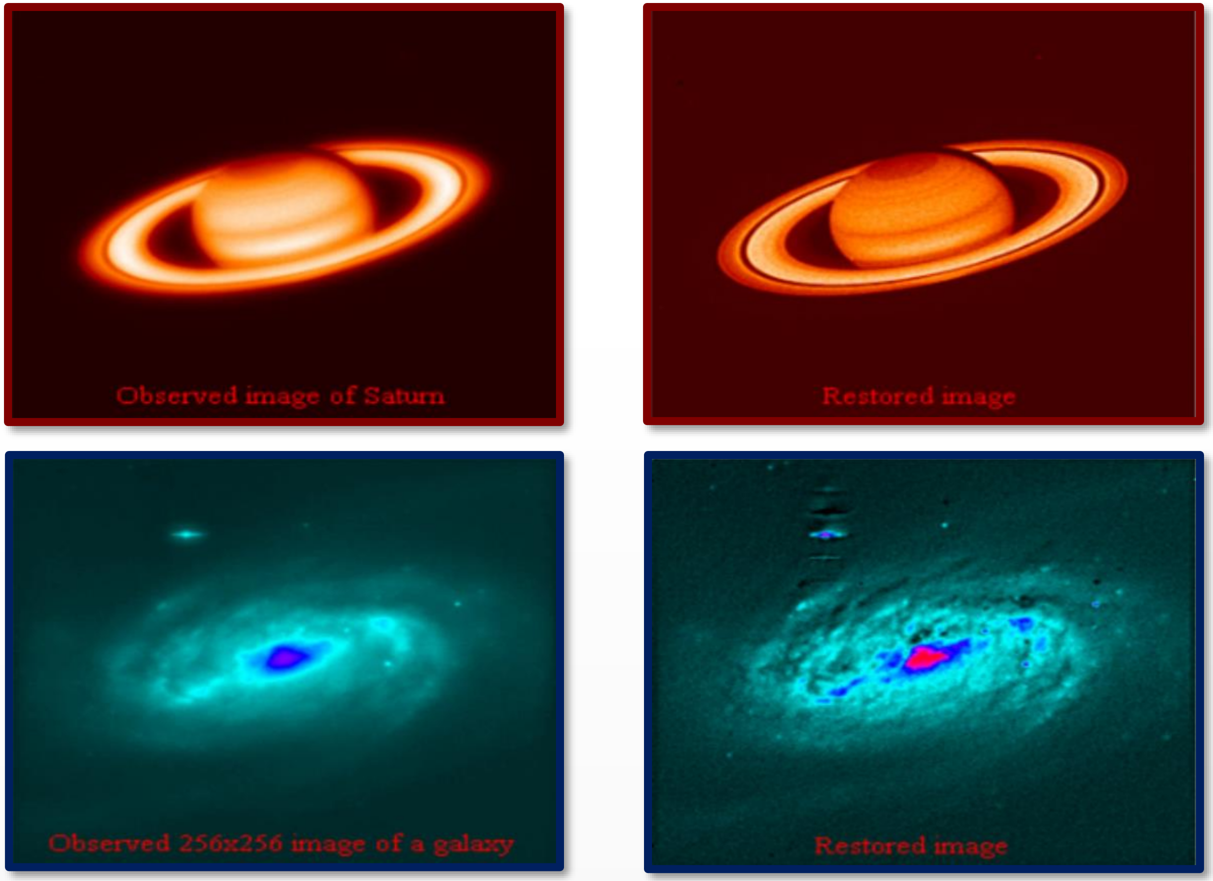
Restored Image



Original Image

Image Recovery Examples

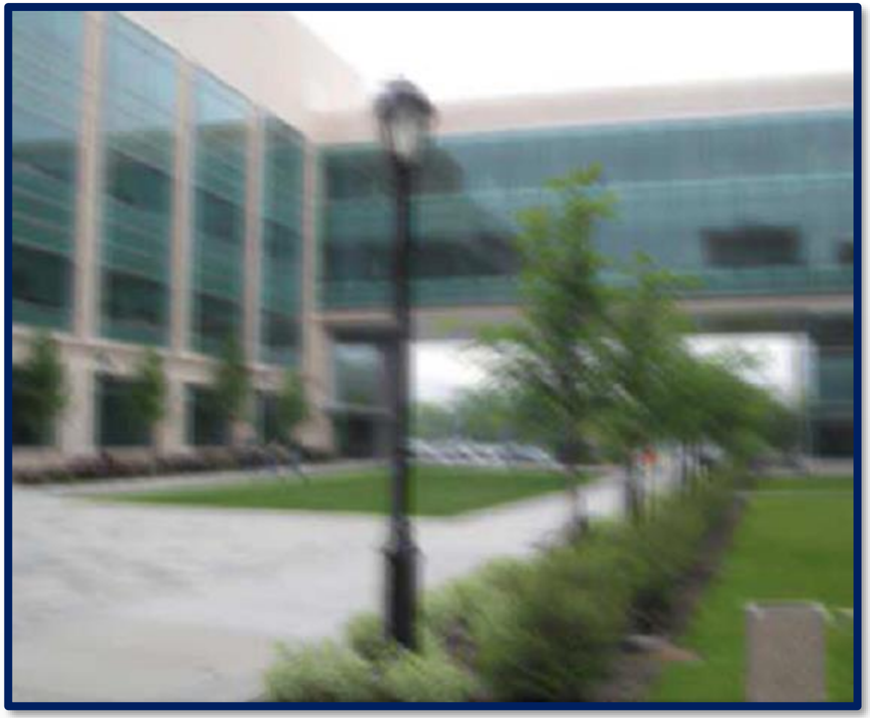
- Images from the Hubble Space Telescope, taken with a defective mirror.



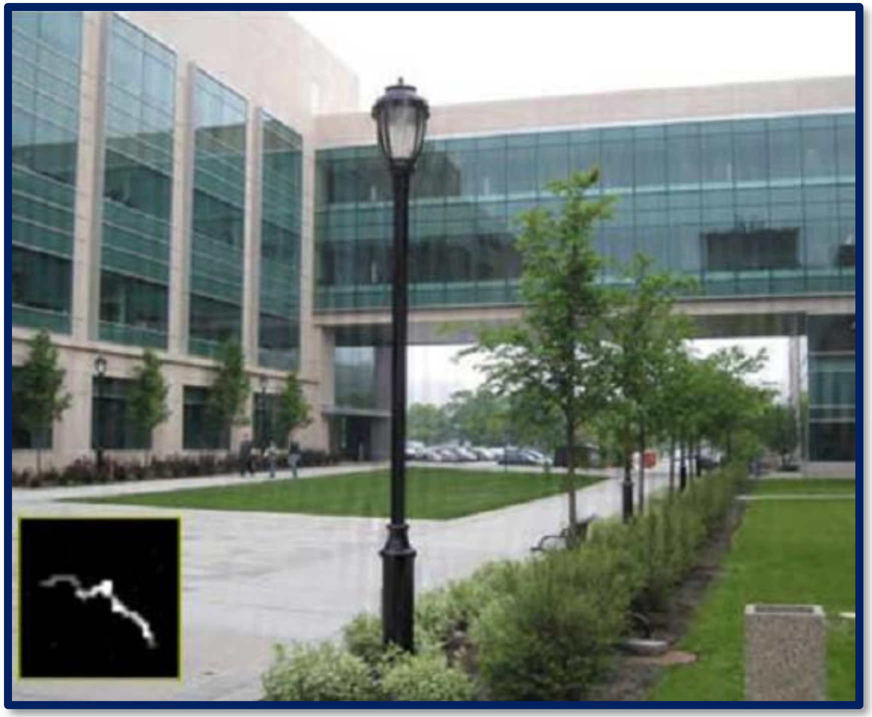
Source of the images: Fundamentals of Digital Image and Video Processing lectures by Aggelos K. Katsaggelos

Image Recovery Examples

- Blind restoration of image corrupted by motion blur:



Original Image with Motion Blur

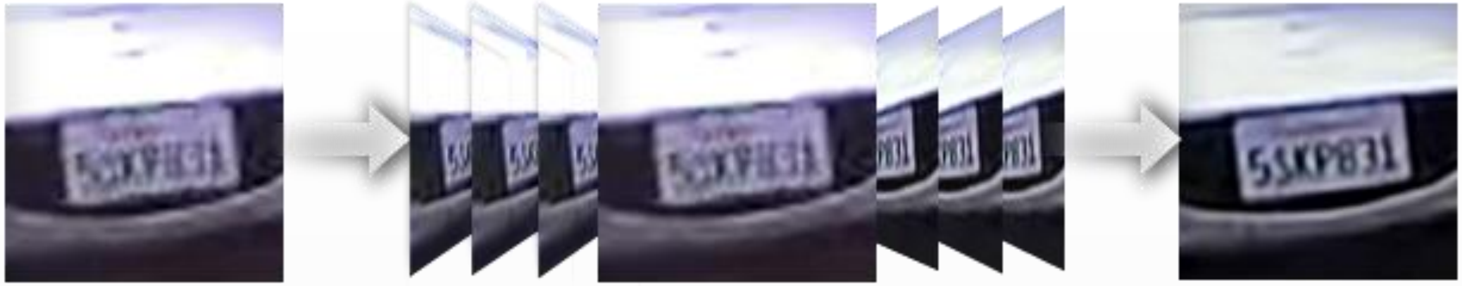


Restored Image

Zhaofu Chen; Derin Babacan, S.; Molina, R.; **Katsaggelos**, AK., "Variational Bayesian Methods For Multimedia Problems," *Multimedia, IEEE Transactions on* , vol.16, no.4, pp.1000,1017, June 2014.

Image Recovery Examples

- Super-resolution:
  - The process of combining multiple low resolution images to form a high resolution image.



Single, non-enhanced frame

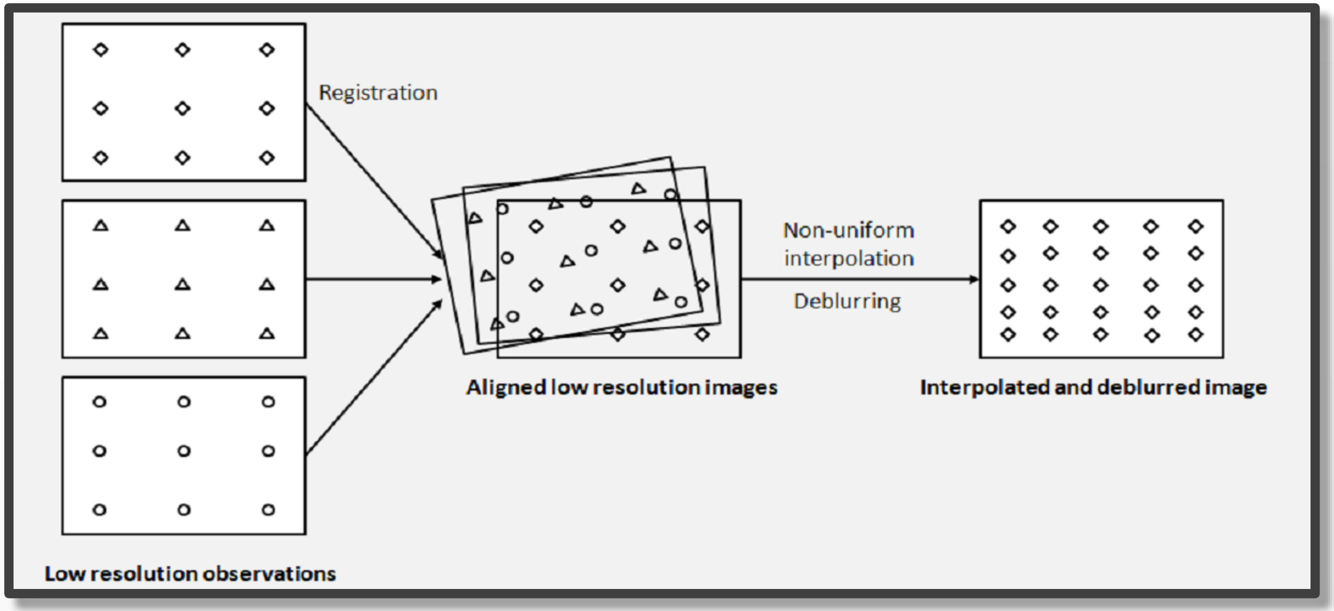
Multiple Frames of the same scene

Reconstructed Frame

Source of the Image: http://www.motiondsp.com/products/ikena/super-resolution

Super-Resolution

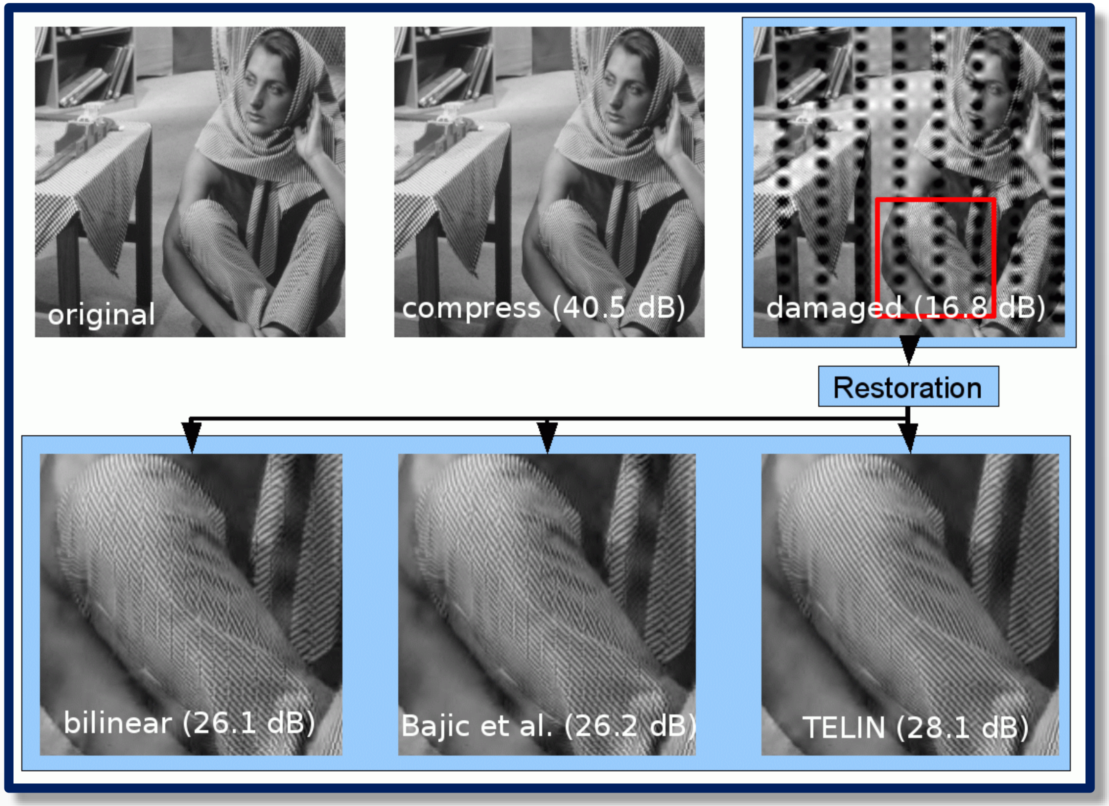
- Concept:
  - We have a series of snapshots of the same scene (e.g. video).
  - Due to camera or subject motion, each image provides a slightly different view.
  - Together, they provide a much more of information about the scene.


Registration
Non-uniform interpolation
Deblurring
Aligned low resolution images
Interpolated and deblurred image
Low resolution observations

http://www.ifp.illinois.edu/~jyang29/papers/chap1.pdf

Image Recovery Examples

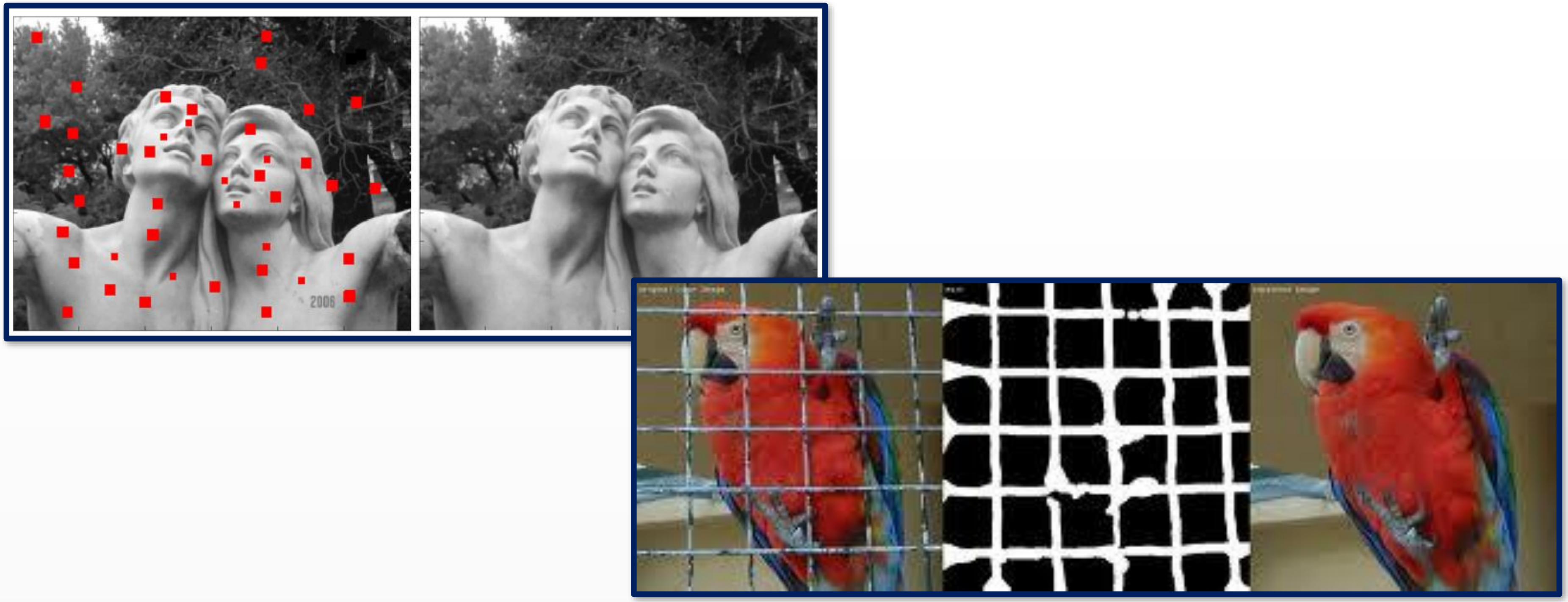
- Error Concealment: reconstruction of data that was lost during transmission of images e.g. over a network where data packets are lost



J. Rombaut, A. Pizurica, and W. Philips, "Locally adaptive passive error concealment for wavelet coded images," IEEE Signal Processing Letters, 2008.

Image Recovery Examples

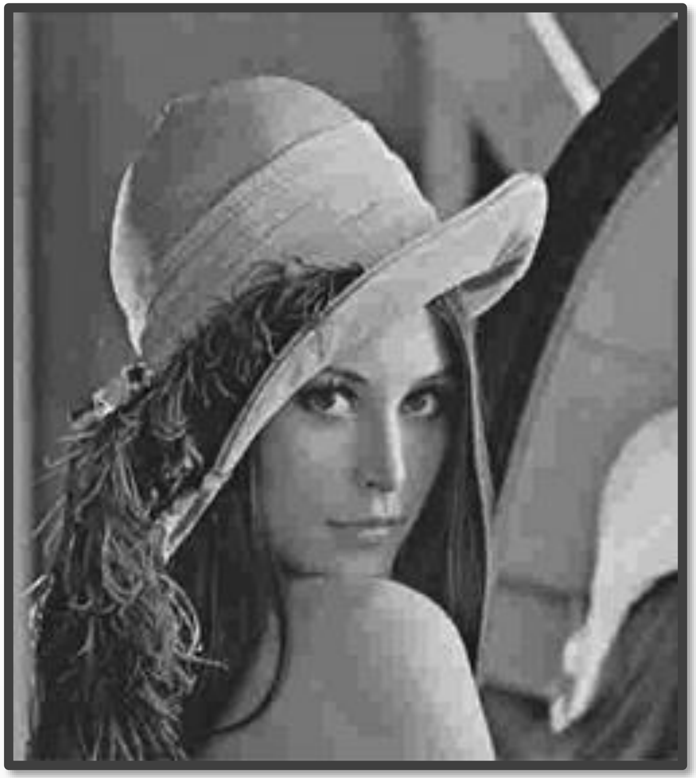
- Inpainting:
  - Similar to error concealment but the location of the missing information is not so well structured and not known apriori, so we have to find it first.

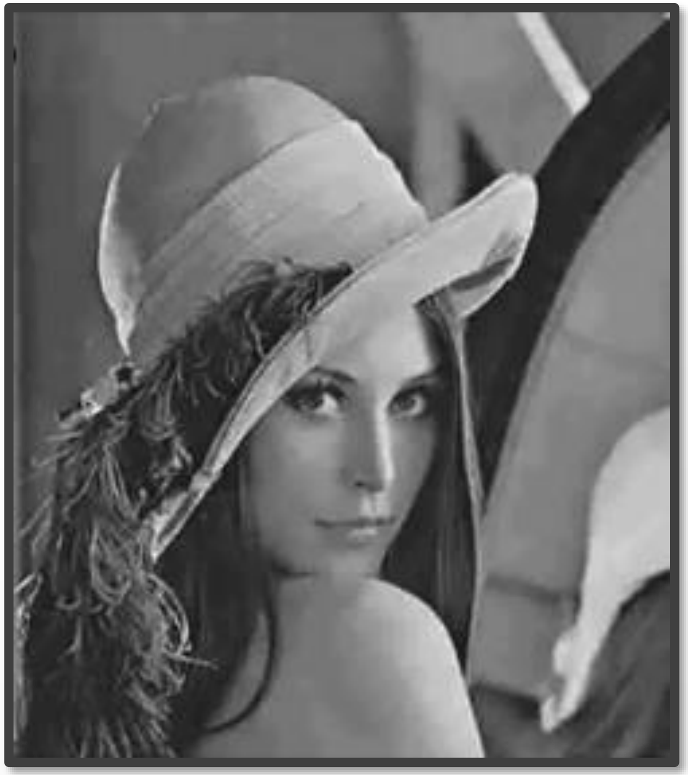


http://www.mathworks.com/company/newsletters/articles/applying-modern-pde-techniques-to-digital-image-restoration.html
http://nbviewer.ipython.org/github/chintak/inpainting-demo/blob/master/Hello_ShopSense.ipynb

Image Recovery Examples

- Deblocking:
  - removal of blocking artifacts introduced by compression





S. Alireza Golestaneh, D. M. Chandler, "An Algorithm for JPEG Artifact Reduction via Local Edge Regeneration" *Journal of Electronic Imaging (JEI),* Jan 2014

Sources of Degradation and Forms of Recovery

Sources of Degradation

1. Motion
2. Atmospheric turbulence
3. Out-of-focus lens
4. Finite resolution of the sensors
5. Limitations of the acquisition system
6. Transmission error
7. Quantization error
8. Noise

Forms of Restoration

1. Restoration/Deconvolution
2. Removal of Compression Artifacts
3. Super-Resolution
4. Inpainting/Concealment
5. Noise smoothing

Inverse problem formulation of Recovery

- The original image *x* goes through a system (*H*), that introduces some type of degradation resulting the observed image *y*:

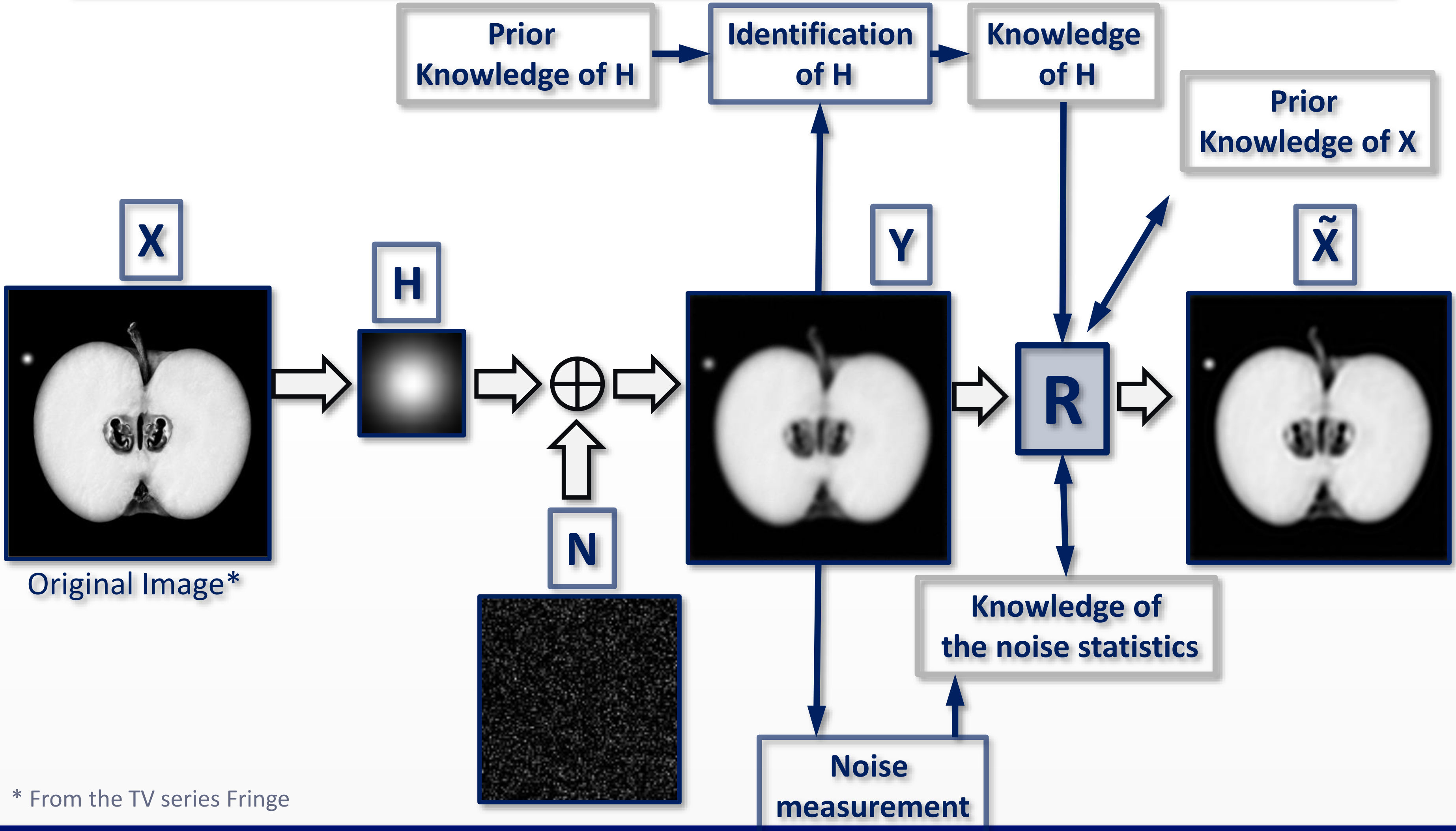
$$x(n_1, n_2) \rightarrow H \rightarrow y(n_1, n_2)$$

- The objective is to reconstruct *x* based on…
  - *y* and *H* → recovery
  - *y* → blind recovery
  - *y* and partially *H* → semi-blind recovery

Inverse problems

- If we know *x* and
  - *y* → system identification
  - *H* → system implementation

Degradation and Restoration



* From the TV series Fringe

Degradation Model

- The model of degradation for **restoration** problems:

$$y(n_1, n_2) = H\left[x(n_1, n_2)\right] + n(n_1, n_2)$$

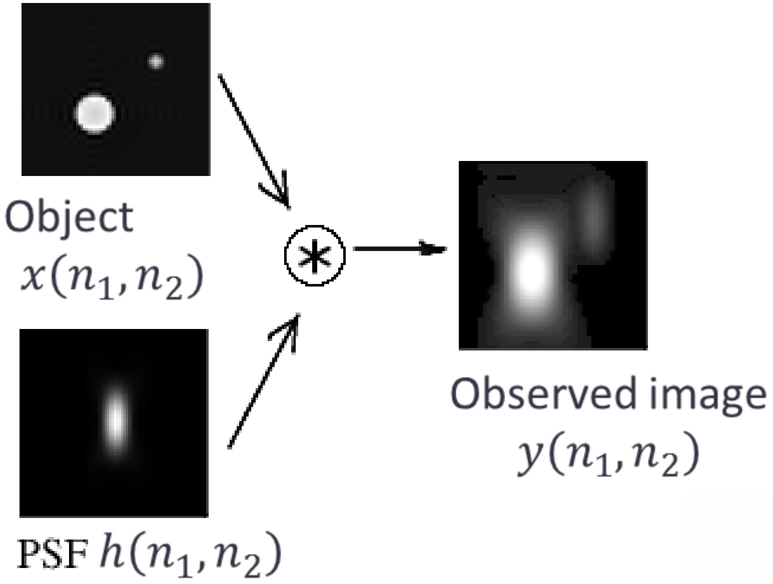
- If an LSI degradation system is assumed, with signal independent additive noise:

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) + n(n_1, n_2)$$

- The restoration problem in this case is called **deconvolution**.

Point spread function (PSF) of an imaging system

- PSF: system's impulse response
  - An image $h(n_1,n_2)$ which describes the response of an imaging system to a point source or a point object
  - The degree of spreading (blurring) of the point object is a measure for the quality of the imaging system
  - The observed image $y(n_1,n_2)$ can be taken as the convolution of the object and the PSF



$$y(n_1,n_2) = x(n_1,n_2) * h(n_1,n_2)$$

- 2D convolution calculation in a naive form: quite slow, $O(N^4)$

Faster convolution calculation – 1st approach: Convolution in matrix-vector form (MVF)

- 1D convolution can be represented in a matrix-vector form:

$$y(n) = x(n) * h(n) = \sum_k x(k)h(n-k)\,,\ \text{where}\quad \begin{aligned} x &: 1\times N \\ h &: 1\times L \\ y &: 1\times\left(N+L-1\right) \end{aligned}$$

$$\begin{bmatrix} y(0) \\ y(1) \\ \\ \vdots \\ \\ y(N+L-2) \end{bmatrix} = \begin{bmatrix} h(0) & 0 & \cdots & \cdots & 0 \\ h(1) & h(0) & 0 & \cdots & 0 \\ h(2) & h(1) & h(0) & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ h(L-1) & \vdots & & & h(0) \\ 0 & h(L-1) & & & \vdots \\ \vdots & \vdots & & & h(L-1) \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ \\ \vdots \\ \\ x(N-1) \end{bmatrix}$$

$$y \qquad = \qquad H \qquad \cdot \qquad x$$

H: ***block circulant*** **matrix.**

LSI degradation model in MVF

- The $N_1 \times N_2$ images involved must be lexicographically ordered. That means that an image is converted to a column vector by pasting the rows one by one after converting them to columns.
  - An image of size 256×256 is converted to a column vector of size 65536×1.
- An LSI degradation model can be written in a matrix form, where the images are vectors and the degradation process is a **huge but sparse block circulant** matrix H, and n is a noise component

$$y = Hx + n$$

  - x, n and y are column vectors of size $N_1 N_2 \times 1$

Faster convolution calculation – 2nd approach: Operation in the Fourier domain

- ***Convolution theorem:*** convolution in the spatial (PSF) domain becomes simple element wise multiplication in the Fourier domain

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) \rightarrow$$

$$\rightarrow Y(\omega_1, \omega_2) = X(\omega_1, \omega_2) \cdot H(\omega_1, \omega_2)$$

- LSI degradation model representation in the frequency domain:

$$\mathrm{Y}(\omega_1, \omega_2) = \mathrm{H}(\omega_1, \omega_2) \cdot \mathrm{X}(\omega_1, \omega_2) + \mathrm{N}(\omega_1, \omega_2), \text{ where } \begin{cases} \omega_1 = 0, \ldots, \mathrm{N}_1 - 1 \\ \omega_2 = 0, \ldots, \mathrm{N}_2 - 1 \end{cases}$$

Here all matrices have a size of $N_1 \times N_2$:

- $X(\omega_1, \omega_2)$: DFT of the $x(n_1, n_2)$ 2D input image (matrix!)
- H(ω_1, ω_2): DFT of the point spread function (***optical transfer function***),
- N(ω_1, ω_2): DFT of the noise
- Y(ω_1, ω_2): DFT of the output

Image Recovery

Deconvolution Algorithms

Inverse Filter

- Simplest deconvolution filter, developed for LSI systems.
- Can be easily implemented in the frequency domain as the inverse of the degradation filter.
- Main limitations and drawbacks:
  - Strong noise amplification
  - The degradation system has to be known a priori.
- The degradation equation: $\boxed{\boldsymbol{y} = \boldsymbol{Hx} + \boldsymbol{n}}$
  - In this problem we know ***H*** and ***y*** and we are looking for a descent **x**
  - The objective is to find ***x*** that minimizes the Euclidian norm of the error:

$$\underset{x}{\operatorname{argmin}}\left(J(x)\right) = \underset{x}{\operatorname{argmin}}\left(\|y - Hx\|^2\right)$$

Inverse Filter

- The problem is formulated as follows: we are looking to minimize the Euclidian norm of the error:

$$\underset{x}{\operatorname{argmin}}\left(J(x)\right) \quad = \underset{x}{\operatorname{argmin}}\left(\|y - Hx\|^2\right)$$

- The first derivative of the minimization function must be set to zero.

$$\frac{\partial J(x)}{\partial x} = 0 \Rightarrow \frac{\partial}{\partial x}\left(y^T y - 2x^T H^T y + x^T H^T Hx\right)$$

$$= -2H^T y + 2H^T Hx = 0$$

$$H^T Hx = H^T y$$

Generalized Inverse

$$x = \left(H^T H\right)^+ H^T y$$

Inverse Filter

- We have that in Matrix-vector form (Mvf):

$$HH^T x = H^T y$$

- Frequency domain representation: if we take the DFT of the above relationship in both sides we have:

$$|H(\omega_1, \omega_2)|^2 \cdot X(\omega_1, \omega_2) = H^*(\omega_1, \omega_2) \cdot Y(\omega_1, \omega_2)$$

  - *Recap:* connection between the Mvf and the DFT representation of LSI systems
  - *We do not prove here*: if the DFT of an LSI transform H is $H(\omega_1, \omega_2)$, then the DFT of H^T is $H^*(\omega_1, \omega_2)$ (complex conjugate)
  - Easy to prove: for any complex number $c \cdot c^* = |c|^2$:
    - $c \cdot c^* = (x + jy) \cdot (x - jy) = x^2 - jxy + jxy - jjy^2 = x^2 + y^2 = |c|^2$

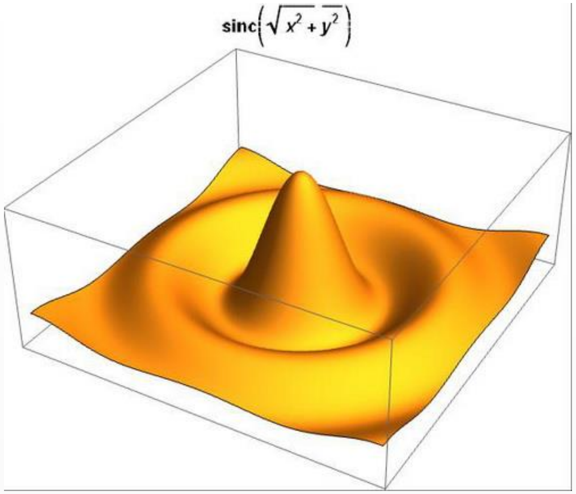
Inverse Filter

- We have that:

$$X(\omega_1,\omega_2) = \frac{H^*(\omega_1,\omega_2)Y(\omega_1,\omega_2)}{\left|H(\omega_1,\omega_2)\right|^2}$$

- **Problem:** It is very likely that $H(\omega_1,\omega_2)$ is 0 or very small at certain frequency pairs.

- For example, $H(\omega_1,\omega_2)$ could be a $sinc$ function.
- In general, since $H(\omega_1,\omega_2)$ is a low pass filter, it is very likely that its values drop off rapidly as the distance of (ω_1,ω_2) from the origin $(0,0)$ increases.



Slide credit: Tania Stathaki Imperial College London

Inverse filtering for noisy scenarios

- Simplification: consider a system where $H(\omega_1, \omega_2)$ is real*. In this case, the inverse filter output is calculated as:

$$\hat{X}_{inv}(\omega_1,\omega_2) = \frac{Y(\omega_1,\omega_2)}{H(\omega_1,\omega_2)}$$

- Assume, that in fact the degradation system output is affected by noise $N(\omega_1, \omega_2)$:

$$Y(\omega_1,\omega_2) = H(\omega_1,\omega_2) \cdot X(\omega_1,\omega_2) + N(\omega_1,\omega_2)$$

- In this case:

Real signal

$\hat{X}_{inv}(\omega_1,\omega_2)$

Reconstruction error

$$\mathrm{X}(\omega_1,\omega_2) = \frac{\mathrm{Y}(\omega_1,\omega_2) - \mathrm{N}(\omega_1,\omega_2)}{\mathrm{H}(\omega_1,\omega_2)} = \frac{\mathrm{Y}(\omega_1,\omega_2)}{\mathrm{H}(\omega_1,\omega_2)} - \frac{\mathrm{N}(\omega_1,\omega_2)}{\mathrm{H}(\omega_1,\omega_2)}$$

*holds for central symmetric PSF

Inverse filtering for noisy scenarios

- Filter output is affected by noise $N(\omega_1, \omega_2)$:

$$X(\omega_1, \omega_2) = \frac{Y(\omega_1, \omega_2) - N(\omega_1, \omega_2)}{H(\omega_1, \omega_2)} = \frac{Y(\omega_1, \omega_2)}{H(\omega_1, \omega_2)} - \boxed{\frac{N(\omega_1, \omega_2)}{H(\omega_1, \omega_2)}}$$

- **Problem:** It is definite that while $H(\omega_1, \omega_2)$ is 0 or very small at certain frequency pairs, $N(\omega_1, \omega_2)$ is large.
- Note that $H(\omega_1, \omega_2)$ is a low pass filter, whereas $N(\omega_1, \omega_2)$ is an all pass function. Therefore, the term $\frac{N(\omega_1,\omega_2)}{H(\omega_1,\omega_2)}$ (error of estimation for $X(\omega_1, \omega_2)$) can be huge!
- The drawback of this method is the **strong amplification of noise** - Inverse filtering fails in that case ☹

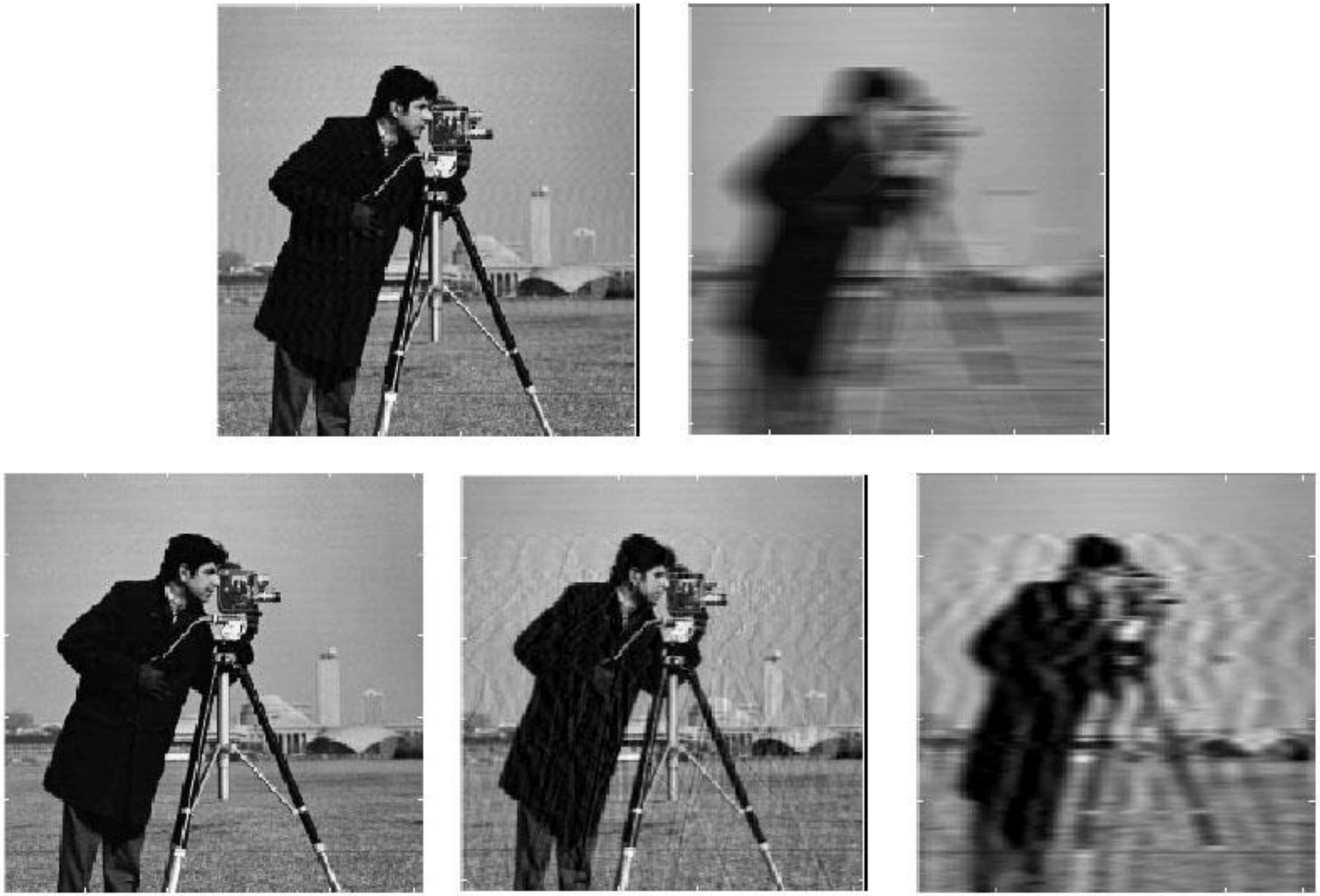
Pseudo-inverse filtering

- Instead of the conventional inverse filter, we implement the following:

$$X(\omega_1,\omega_2)=\begin{cases} \dfrac{H^*(\omega_1,\omega_2)Y(\omega_1,\omega_2)}{\left|H(\omega_1,\omega_2)\right|^2} & \text{if } \left|H(\omega_1,\omega_2)\right| \geq T \\ T & \text{if } \left|H(\omega_1,\omega_2)\right| < T \end{cases}$$

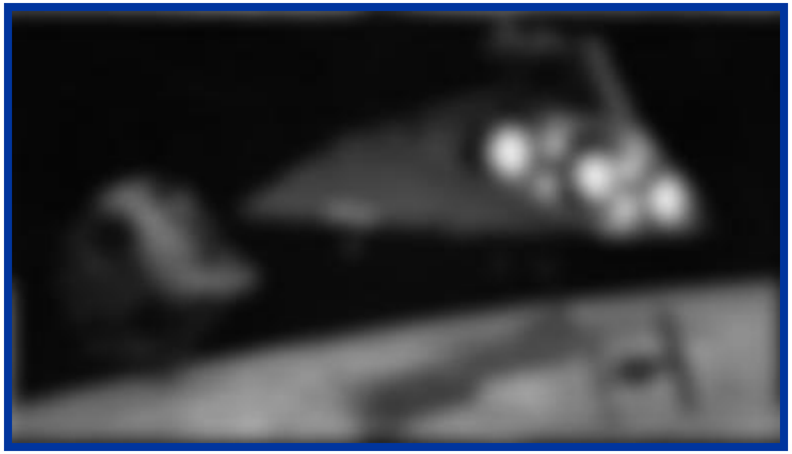
- The parameter T (called **threshold** in the figures in the next slides) is a small number chosen by the user.
- This filter is called **pseudo-inverse** or **generalized** inverse filter.

Pseudo-inverse filtering with different thresholds

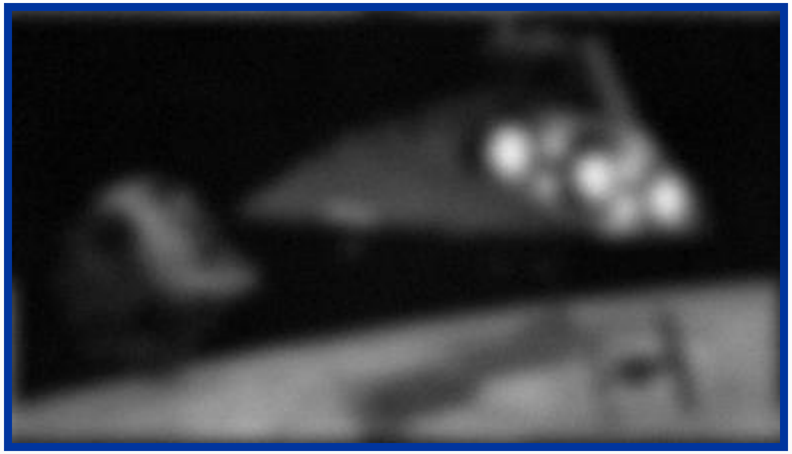


Inverse Filter

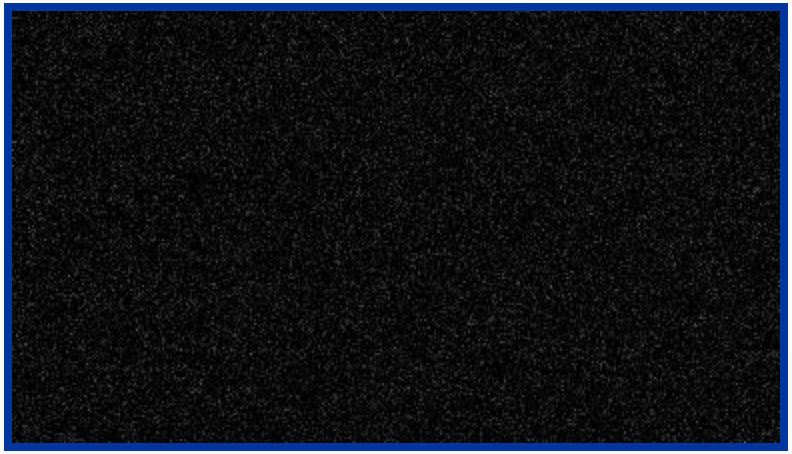
Original Image



Blurred Image

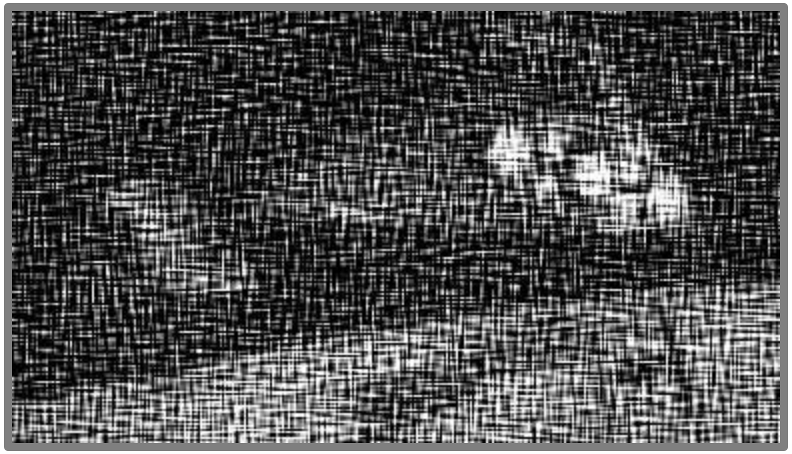


Blurred Image with Additional Noise

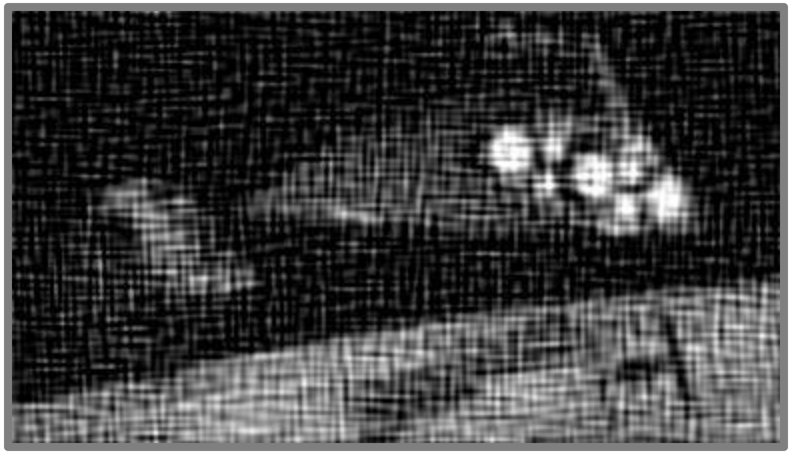


The Noise Component (amplified 10 times)

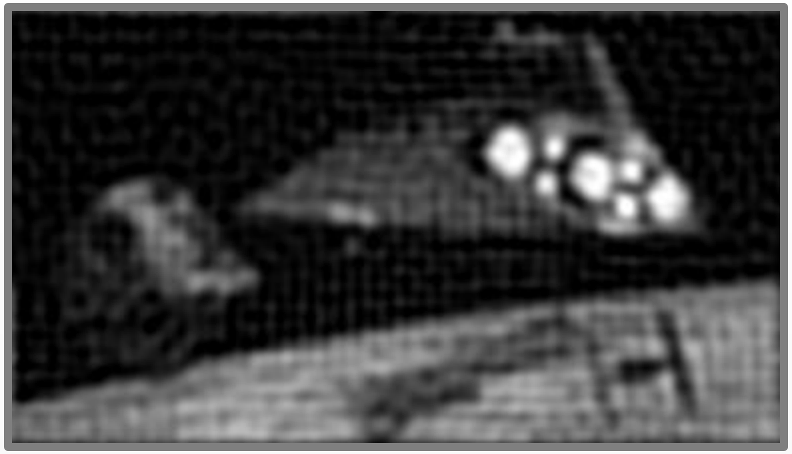
Inverse Filter



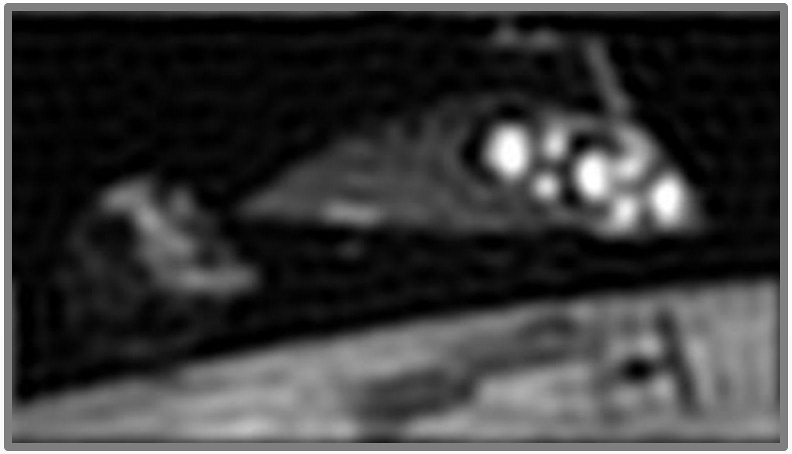
Reconstructed Image (with *T=0.005*)



Reconstructed Image (with *T=0.01*)

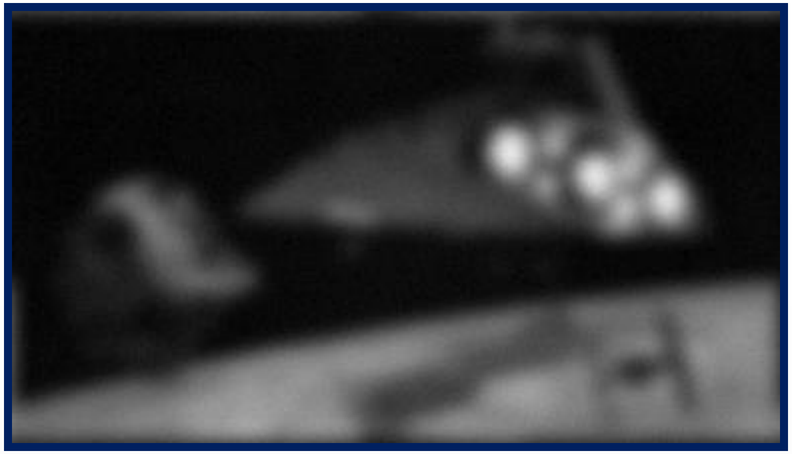


Reconstructed Image (with *T=0.02*)

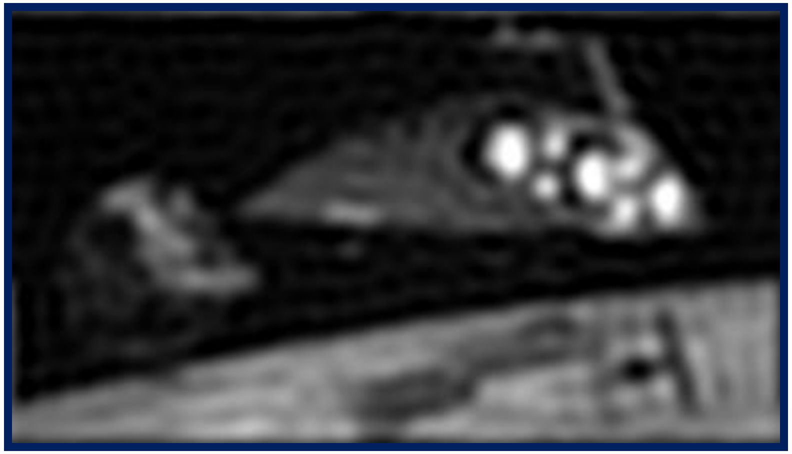


Reconstructed Image (with *T=0.1*)

Inverse Filter



Blurred Image with Additional Noise



Reconstructed Image (with *T=0.1*)

Matlab code of an inverse filter

```
T=.2;
x=double(imread('lena.bmp')); %read original image
N1=size(x,1); N2=size(x,2);
figure(1); imagesc(x); colormap(gray); %display original image
w=5; h=ones(w,w)/w^2; % PSF of bluring
X=fft2(x); % DFT of original image
H=fft2(h,N1,N2); % DFT of PSF
Y=X.*H; % DFT of blurred image
y=ifft2(Y)+10*randn(N1,N2); %observed image: blurred + additive
noise
Y=fft2(y); % DFT of the observed image
figure(2); imagesc(abs(ifft2(Y))); colormap(gray); %display
observed image
BF=find(abs(H)<T);
H(BF)=T;
invH=ones(N1,N2)./H;
X1=Y.*invH;
im=abs(ifft2(X1)); % reconstructed image
figure(3); imagesc(im); colormap(gray) %display result
```

Constrained Least Square Methods

- The objective is to reduce the noise amplification effect of the inverse filter by adding extra constraints about the restored image:
  - On one hand we still have the term describing the solution's fidelity to the data:

$$\underset{x}{\operatorname{argmin}}\left(J(x)\right) = \underset{x}{\operatorname{argmin}}\left(\left\|y - Hx\right\|^2\right)$$

  - But we also have a second term, incorporating some ***prior knowledge*** about the smoothness of the original image:

$$\left\|Cx\right\|_2^2 < \varepsilon$$

  - Putting the two together with the introduction of α:

$$\underset{x}{\operatorname{argmin}}\left(\left\|y - Hx\right\|_2^2 + \alpha\left\|Cx\right\|_2^2\right)$$

Constrained Least Square Methods

$$\underset{x}{\operatorname{argmin}} \left(\|y - Hx\|_2^2 + \alpha \|Cx\|_2^2 \right) \Rightarrow \quad x = \left(H^T H + \alpha C^T C\right)^+ H^T y$$

- *C* is a high pass filter
  - Intuitively this means that we want to keep under control the amount of energy contained in the high frequencies on the restored image.
- α is the regularization parameter

- In the frequency domain (for H and C block circulant) we have the following formula:

$$X(\omega_1, \omega_2) = \frac{H^*(\omega_1, \omega_2)}{|H(\omega_1, \omega_2)|^2 + \alpha |C(\omega_1, \omega_2)|^2} Y(\omega_1, \omega_2)$$

  - if α is 0, we get back the simple Least Square method (Inverse Filter)

Constrained Least Square Methods

- (Optional) Different types of regularization:
  - CLS:

$$\tilde{x}(\alpha)_{CLS} = \underset{x}{\operatorname{argmin}}\left(\|y - Hx\|_2^2 + \alpha\|Cx\|_2^2\right)$$

  - Maximum Entropy Regularization:

$$\tilde{x}(\alpha)_{ME} = \underset{x}{\operatorname{argmin}}\left(\|y - Hx\|_2^2 + \alpha\sum_{i=1}^{N} x_i \log(x_i)\right)$$

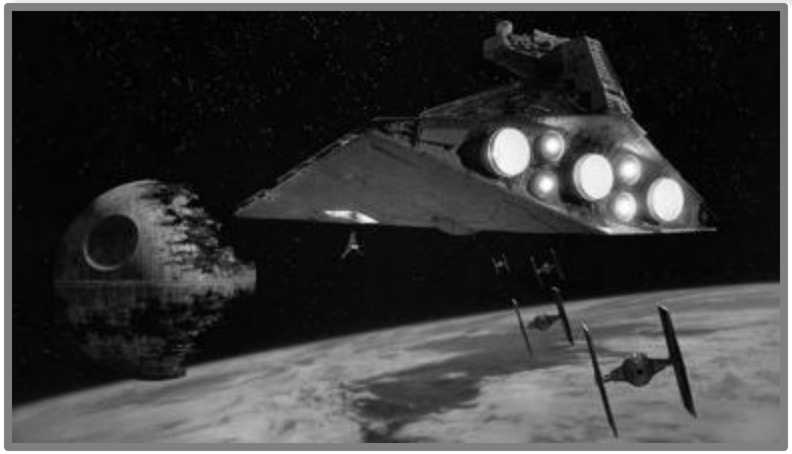
  - Total Variation Regularization:

$$\tilde{x}(\alpha)_{TV} = \underset{x}{\operatorname{argmin}}\left(\|y - Hx\|_2^2 + \alpha\sum_{i=1}^{N} \left|[\Delta x]_i\right|\right)$$

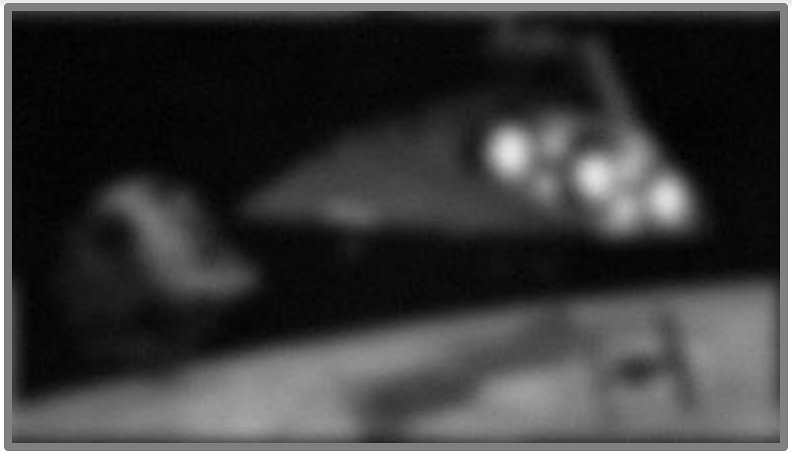
  - l_p – norms:

$$J(z) = \|z\|_p^p = \sum_{i=1}^{N} |z_i|^p, \qquad 1 \le p \le 2$$

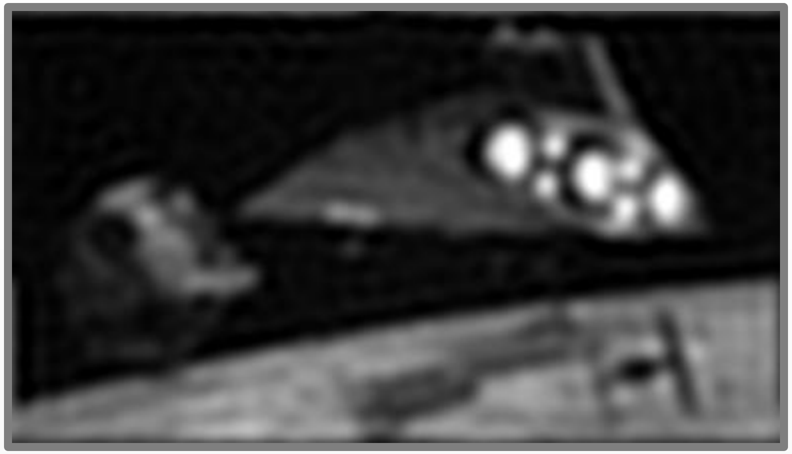
Constrained Least Square Methods



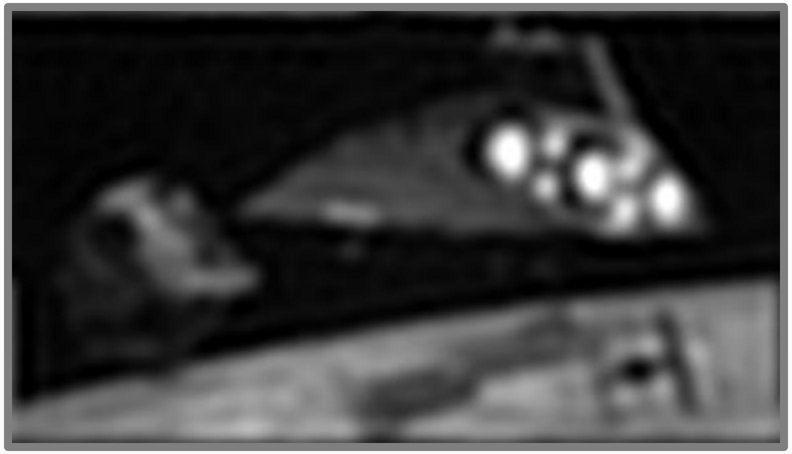
Original Image



Blurred Image with Additional Noise

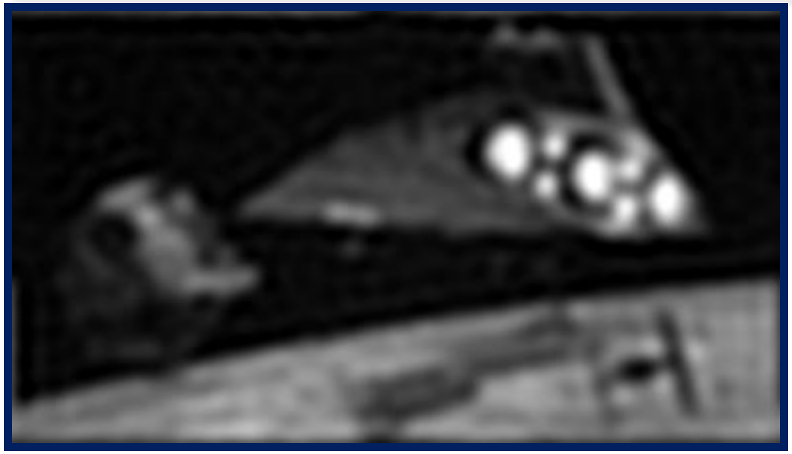


Reconstructed Image (CLS with α=0.1)

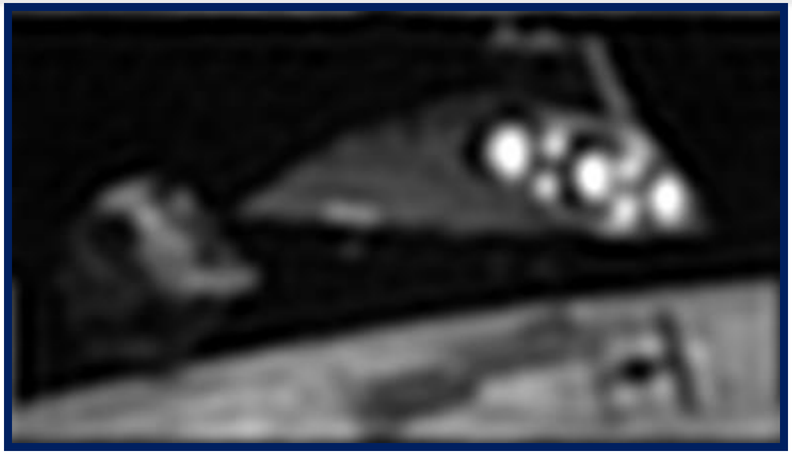


Reconstructed Image (CLS with α=0.5)

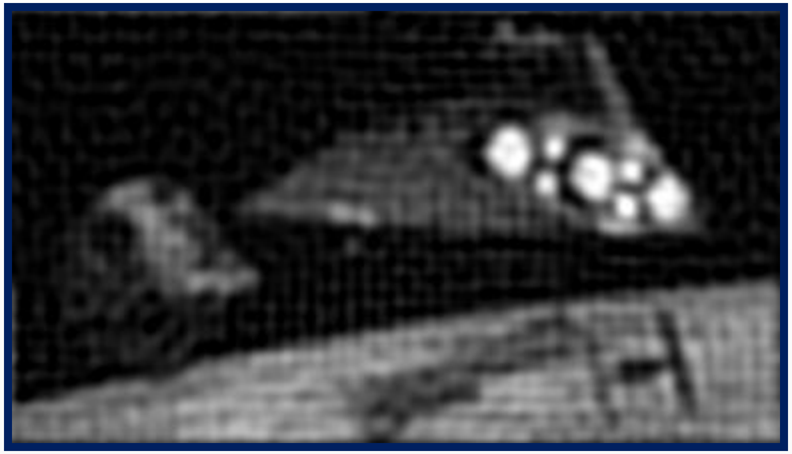
CLS vs. LS



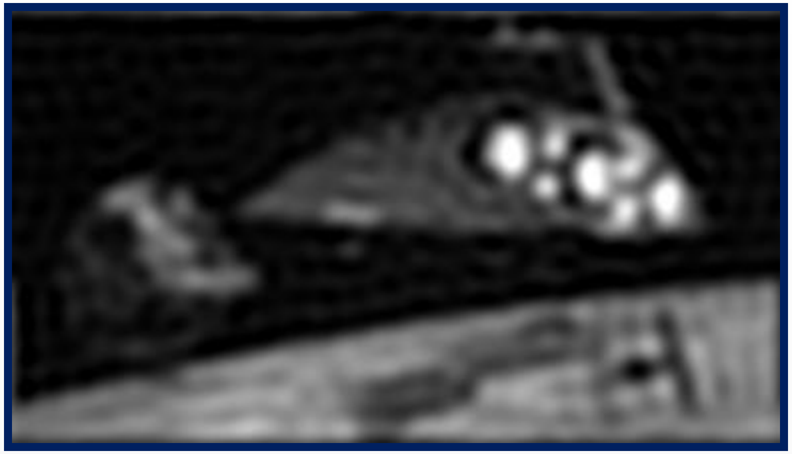
Reconstructed Image (CLS with α=0.1)



Reconstructed Image (CLS with α=0.5)



Reconstructed Image (LS with *T=0.02*)



Reconstructed Image (LS with *T=0.1*)

Wiener Filter

- Stochastic restoration approach:
  - Treat the image as a sample from a 2D random field.
  - The image is part of a class of samples (an ensemble), realizations of the same random field.

$$\boxed{R(\omega_1,\omega_2)=\frac{H^*(\omega_1,\omega_2)\cdot P_{xx}(\omega_1,\omega_2)}{\left|H(\omega_1,\omega_2)\right|^2\cdot P_{xx}(\omega_1,\omega_2)+P_{NN}(\omega_1,\omega_2)}}$$

Expected value over many realizations of the 2D random field

  - Autocorrelation:

$$R_{ff}(n_1,n_2,n_3,n_4)=E\left[f(n_1,n_2)f^*(n_3,n_4)\right]$$

  - Power-Spectrum (**Wide Sense Stationarity (WSS)** input)

$$P_{ff}(\omega_1,\omega_2)=\mathrm{DFT}\left\{R_{ff}(d_1,d_2)\right\}\Big|_{d_1=n_1-n_3,d_2=n_2-n_4}$$

Fourier transformation

Autocorrelation calculation - some details

- Definition of autocorrelation:

Expected value over many realizations of the 2D random field

$$R_{ff}(n_1, n_2, n_3, n_4) = E\left[f(n_1, n_2) f^*(n_3, n_4)\right]$$

- Wide Sense Stationarity property (WSS):

$$R_{ff}(n_1, n_2, n_3, n_4) = R_{ff}(n_1 - n_3, n_2 - n_4) = R_{ff}(d_1, d_2)$$

- Ergodicity:
  - ensemble average is equal to spatial average

$$R_{ff}(d_1, d_2) = \lim_{N \to \infty} \frac{1}{(2N+1)^2} \sum_{k_1=-N}^{N} \sum_{k_2=-N}^{N} f(k_1, k_2) f^*(k_1 - d_1, k_2 - d_2)$$

Autocorrelation calculation the Fourier domain

- Recap (from textures lecture) Wiener-Khinchin Theorem
  - Input image: $x(n_1, n_2)$
  - Fourier transform: $\{X(\omega_1, \omega_2)\} = \mathrm{DFT}\{x(n_1, n_2)\}$
  - Power spectrum: $P_{xx}(\omega_1, \omega_2) = X(\omega_1, \omega_2) \cdot X^*(\omega_1, \omega_2)$
  - Autocorrelation: inverse Fourier transform of the power spectrum: $\{R_{xx}(d_1, d_2)\} = \mathrm{IDFT}\{P_{xx}(\omega_1, \omega_2)\}$

Wiener Filter

- The degradation model: $y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) + n(n_1, n_2)$
- The objective:

$$\tilde{x}(n_1, n_2) = \underset{\tilde{x}(n_1,n_2)}{\operatorname{argmin}}\, E\left[\left|x(n_1, n_2) - \tilde{x}(n_1, n_2)\right|^2\right]$$

- We look for the solution in the following format, assuming an LSI restoration model:

$$\tilde{x}(n_1, n_2) = r(n_1, n_2) * y(n_1, n_2)$$

  - The input image is assumed to be WSS with autocorrelation $R_{xx}(n_1,n_2)$.
- The recovered image is obtained in the Fourier domain:

$$\boxed{X(\omega_1, \omega_2) = R(\omega_1, \omega_2) \cdot Y(\omega_1, \omega_2)}$$

Recovered image

Wiener filter

Observed degraded image

Wiener Filter

- Assumptions:
  - that both the input image and the noise are WSS.
  - The restoration error and the signal (the observed image) is orthogonal:

$$E\left[e(n_1,n_2)y^*(n_3,n_4)\right] = E\left[(x(n_1,n_2)-\tilde{x}(n_1,n_2))y^*(n_3,n_4)\right] = 0, \quad \forall(n_1,n_2),(n_3,n_4)$$

  - It can be shown* that the following transfer function implements the above constraint:

$$R(\omega_1,\omega_2) = \frac{H^*(\omega_1,\omega_2)\cdot P_{xx}(\omega_1,\omega_2)}{\left|H(\omega_1,\omega_2)\right|^2 \cdot P_{xx}(\omega_1,\omega_2) + P_{NN}(\omega_1,\omega_2)}$$

*proof available as supplementary material
optional to read, not part of exam

Wiener Filter and CLS Filter

- Wiener filter:

$$R(\omega_1,\omega_2)=\frac{H^*(\omega_1,\omega_2)\cdot P_{xx}(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2\cdot P_{xx}(\omega_1,\omega_2)+P_{NN}(\omega_1,\omega_2)}=$$

$$=\frac{H^*(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2+\dfrac{P_{NN}(\omega_1,\omega_2)}{P_{xx}(\omega_1,\omega_2)}}=\frac{H^*(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2+\dfrac{\sigma_N^2}{P_{xx}(\omega_1,\omega_2)}}$$

Assuming white noise

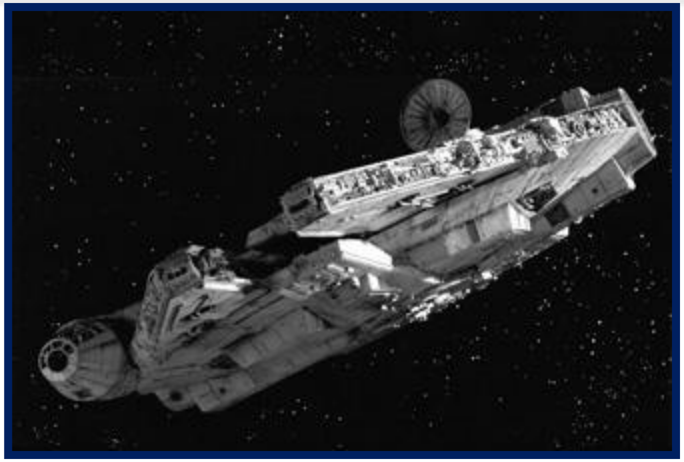
Noise to signal ratio

- CLS filter:

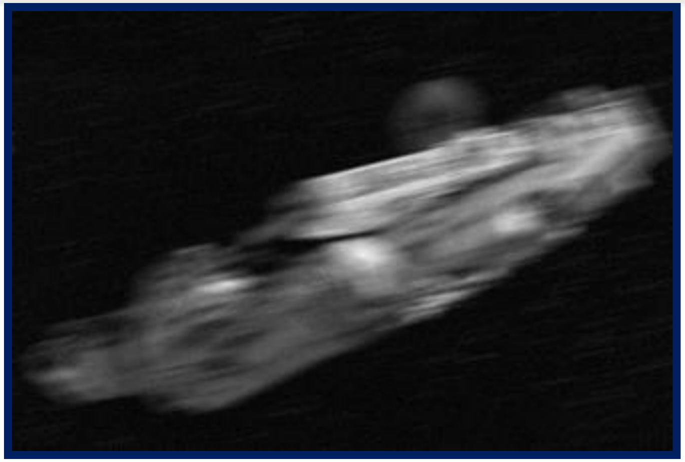
$$R(\omega_1,\omega_2)=\frac{H^*(\omega_1,\omega_2)}{|H(\omega_1,\omega_2)|^2+\alpha|C(\omega_1,\omega_2)|^2}$$

- With the right choice of *C* and *α*, CLS filter is the same as the Wiener filter.

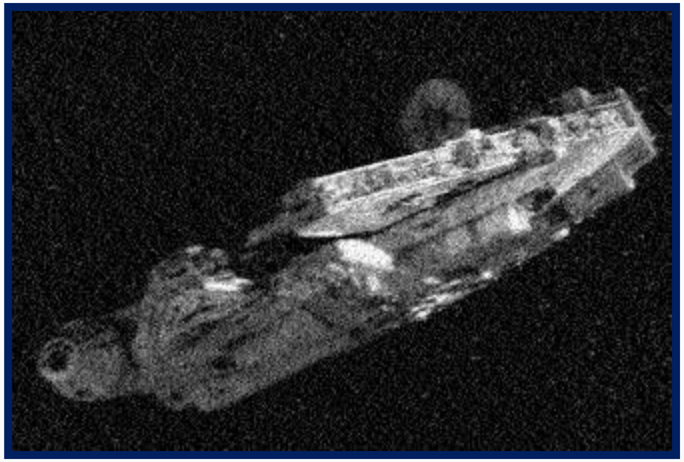
Wiener Filter



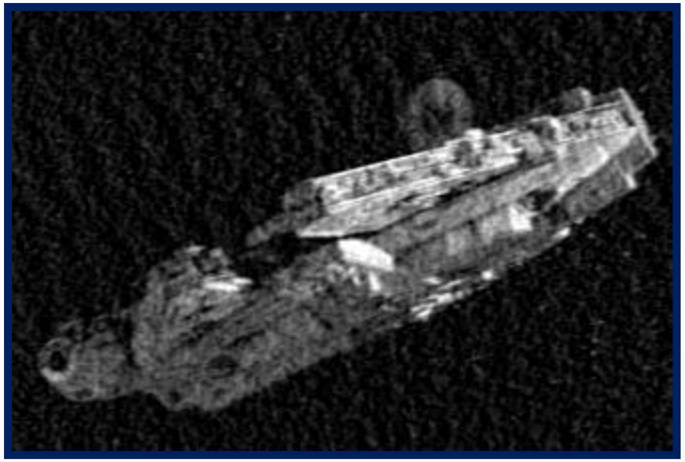
Original Image



Motion Blurred Noisy Image



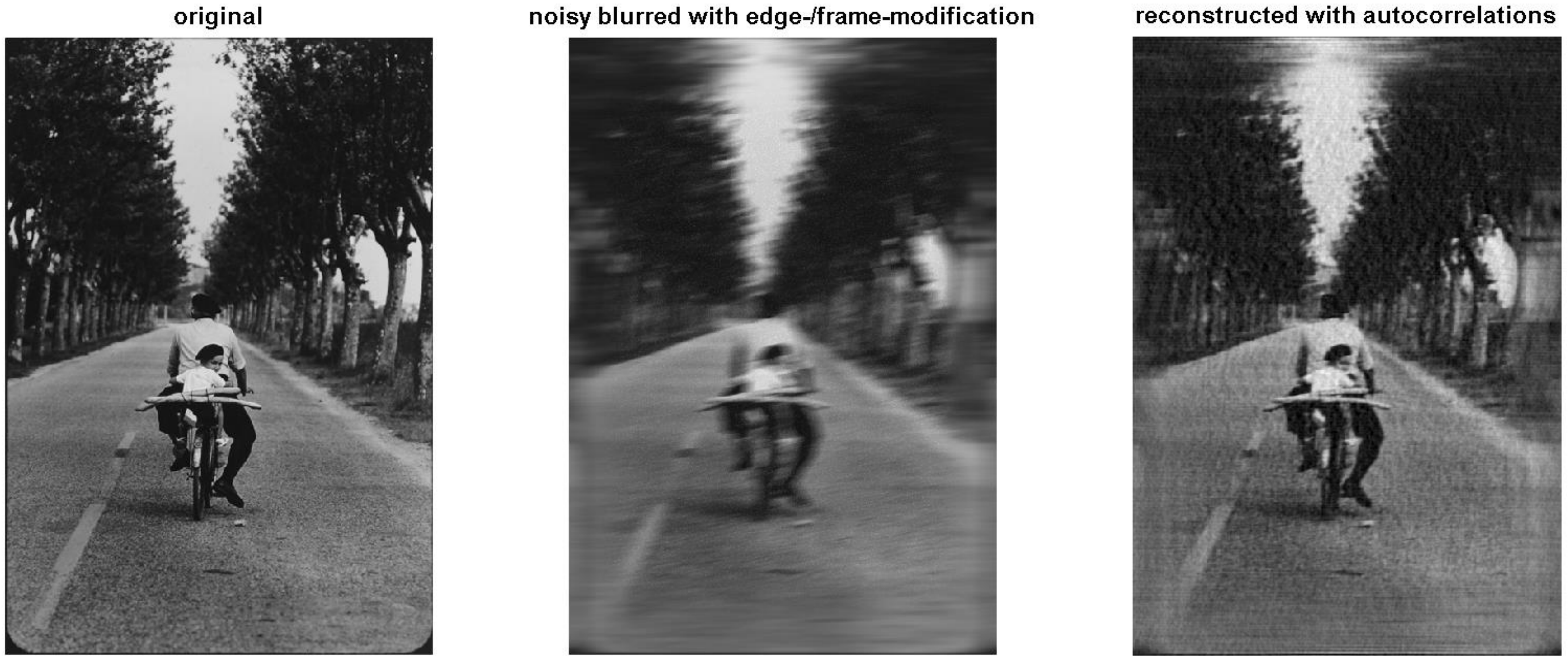
Reconstructed with Wiener filter



Reconstructed with CLS filter

Wiener Lab task on week 7

- See more practical details on the corresponding laboratory excercise!



original

noisy blurred with edge-/frame-modification

reconstructed with autocorrelations

Main Sources and Further Readings

- Fundamentals of Digital Image and Video Processing lectures by Aggelos K. Katsaggelos
- Babacan, D. S., R. Molina, and A. K. Katsaggelos, "Total Variation Super Resolution Using A Variational Approach", IEEE International Conf. on Image Processing 2008, San Diego, USA, 10/10/2008.
- J. Rombaut, A. Pizurica, and W. Philips, "Locally adaptive passive error concealment for wavelet coded images," IEEE Signal Processing Letters, vol. 15, pp. 178-181, 2008.
- Zhaofu Chen; Derin Babacan, S.; Molina, R.; Katsaggelos, AK., "Variational Bayesian Methods For Multimedia Problems," *Multimedia, IEEE Transactions on* , vol.16, no.4, pp.1000,1017, June 2014
- http://www.mathworks.com/company/newsletters/articles/applying-modern-pde-techniques-to-digital-image-restoration.html
- S. Alireza Golestaneh, D. M. Chandler, "An Algorithm for JPEG Artifact Reduction via Local Edge Regeneration" *Journal of Electronic Imaging (JEI),* Jan 2014

Basic Image Processing

PPKE-ITK

Lecture 7.

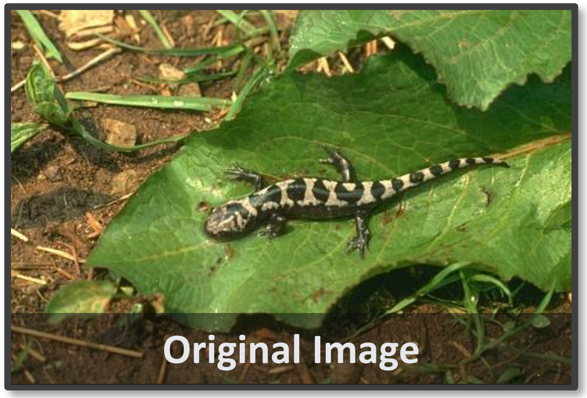
Image Segmentation

Image Segmentation

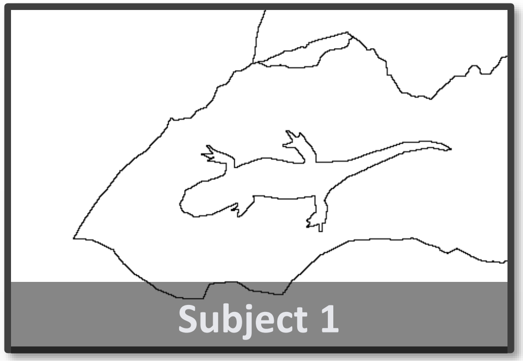
- What is on the image? – This is maybe the most important question we want to answer about an image.
- For a human observer it is a trivial task, for a machine it is still an unsolved problem.
- An important step toward our goal is to segment the image into meaningful parts.
- The objective is to group pixels together based on some common characteristics:
  - they belong to the same physical object
  - they have the same intensity level/color/texture
  - they belong to the background/foreground
  - …

Image Segmentation

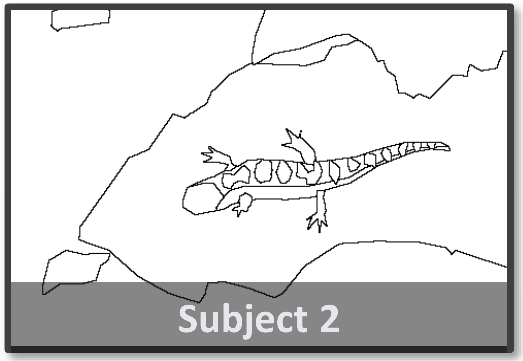
- Sometimes even humans cannot agree on a unique solution!



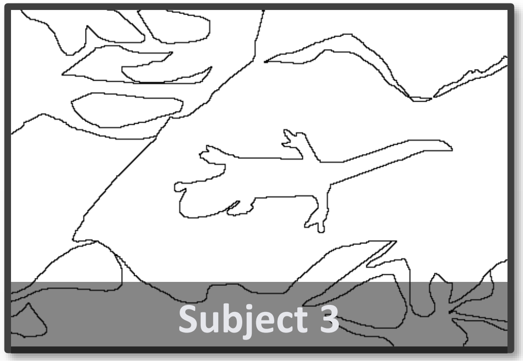
Original Image



Subject 1



Subject 2

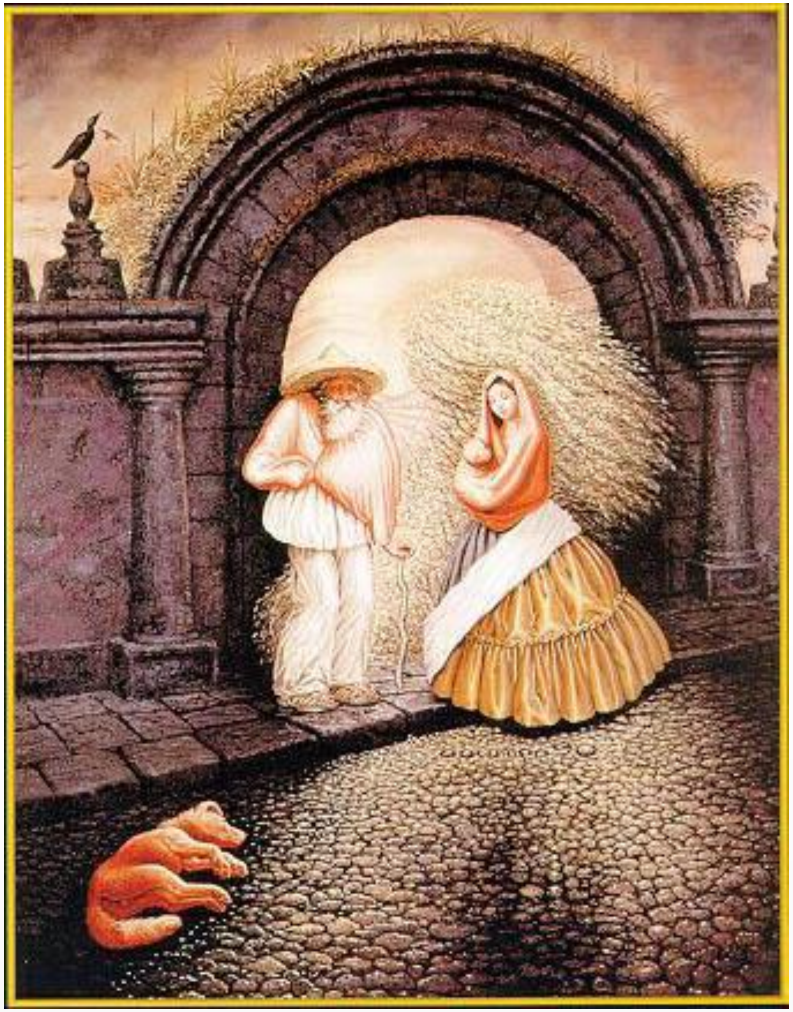


Subject 3

Sample from BSDS500 (Berkeley Segmentation Data Set and Benchmarks 500):
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

Gestalt grouping

- Gestalt definition: a configuration or pattern of elements so unified as a whole that it cannot be described merely as a sum of its parts

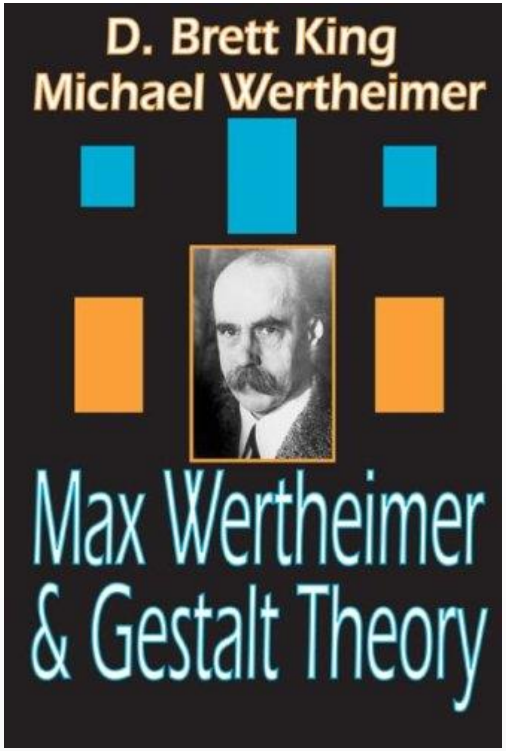


Gestalt psychology or gestaltism

- German: *Gestalt* - "form" or "whole”
  - Berlin School, early 20th century
  - Kurt Koffka, Max Wertheimer, and Wolfgang Köhler

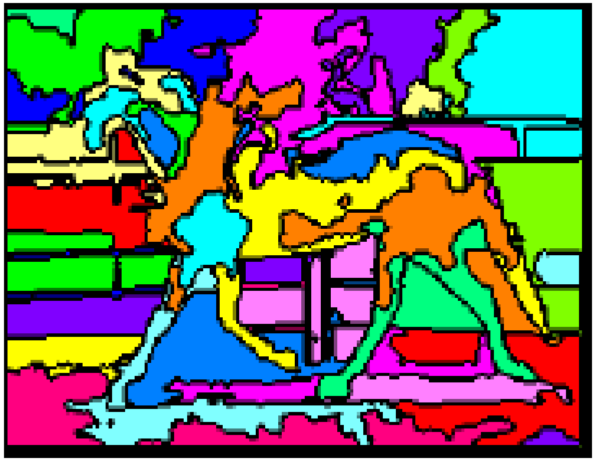
- View of brain:

- whole is more than the sum of its parts
- holistic
- parallel
- analog
- self-organizing tendencies

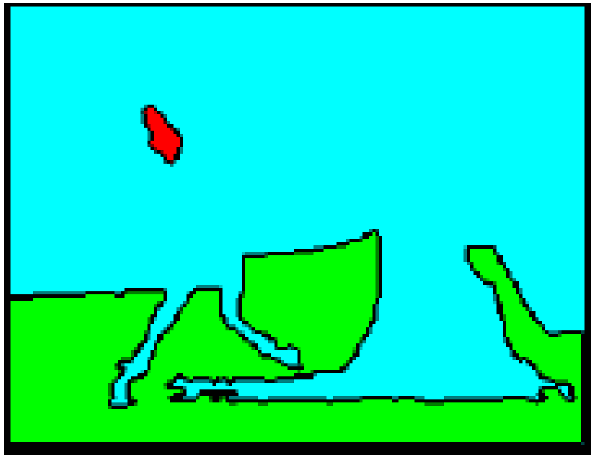


Slide from S. Saverese

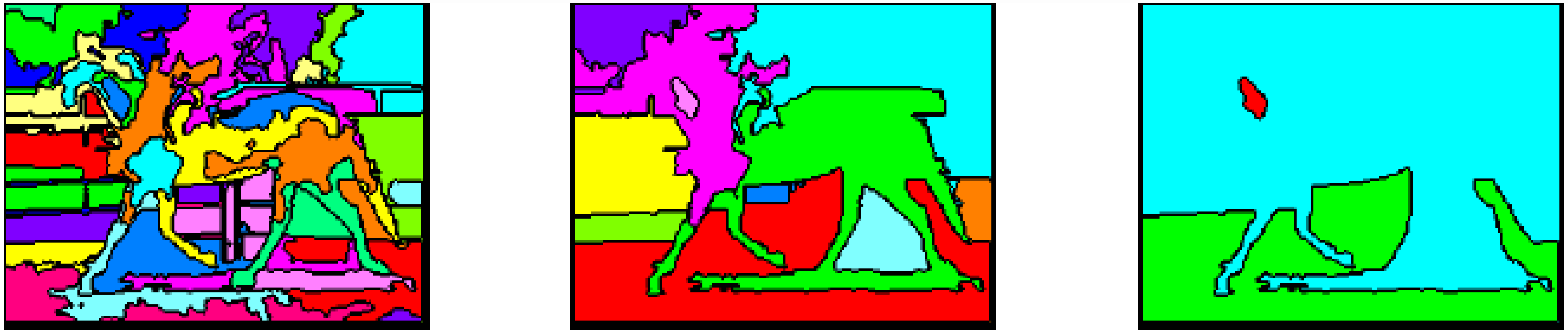
Types of segmentations



Oversegmentation



Undersegmentation



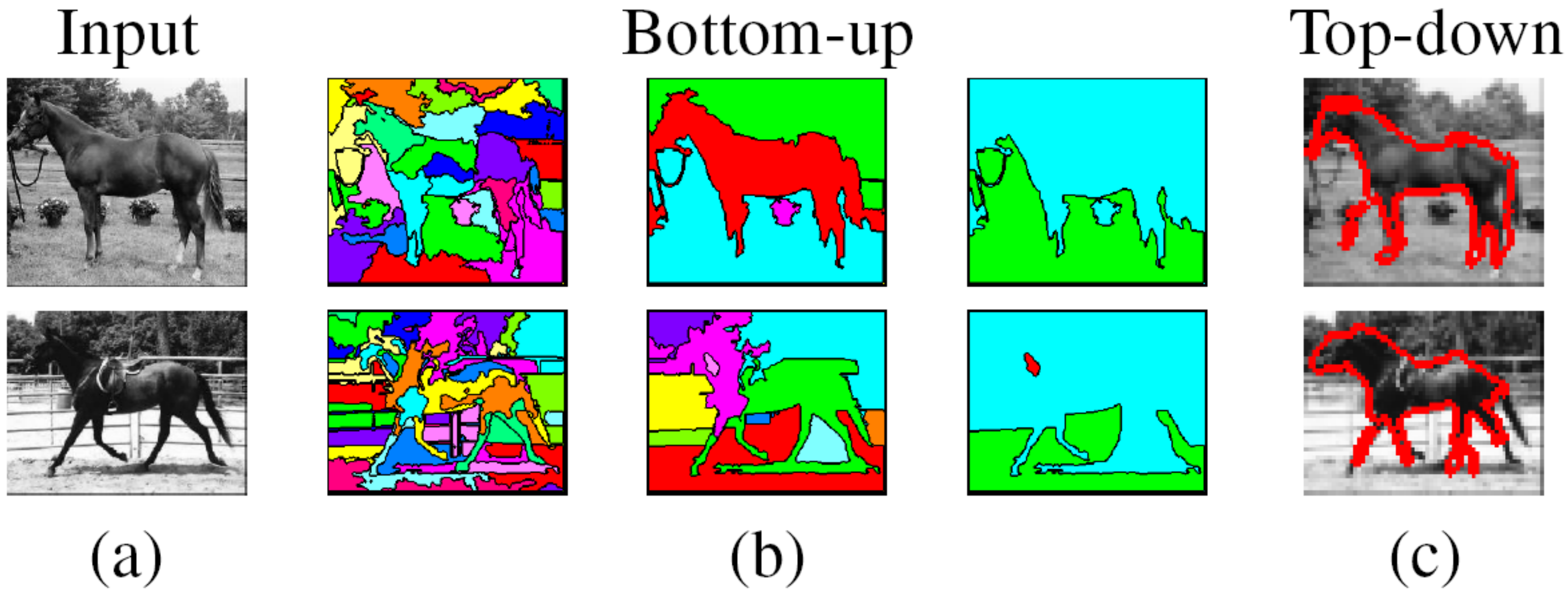
Multiple Segmentations

Image Segmentation

- The segmentation can be **knowledge-driven** (top-down) or **data-driven** (bottom-up).
- Knowledge driven segmentation methods builds prior knowledge into the segmentation algorithm:
  - Hard to implement
  - Cannot stand alone: need cues from bottom-up segmentation
- Data-driven methods builds on the raw pixel data:
  - they are easier to implement
  - they often fail on real life images
- There is the so-called **semantic gap** between the two approach.
- The complex, high level definitions of top-down methods are hard to embed efficiently into low level algorithms.

Major processes for segmentation

- Bottom-up: group tokens with similar features
- Top-down: group tokens that likely belong to the same object



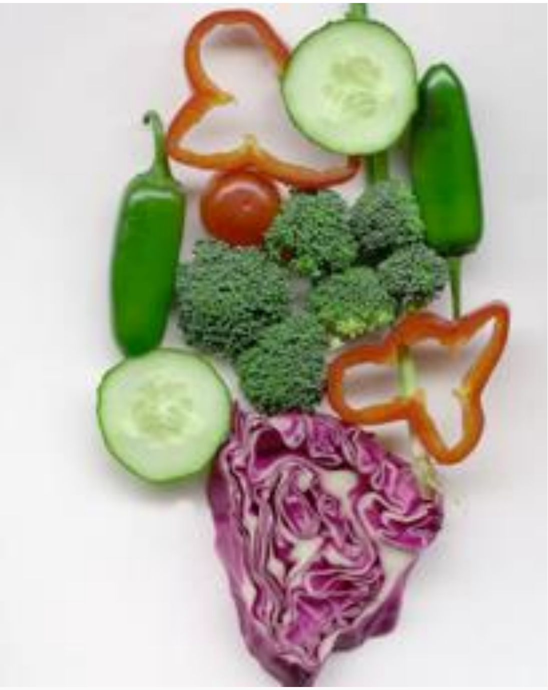
[Levin and Weiss 2006]

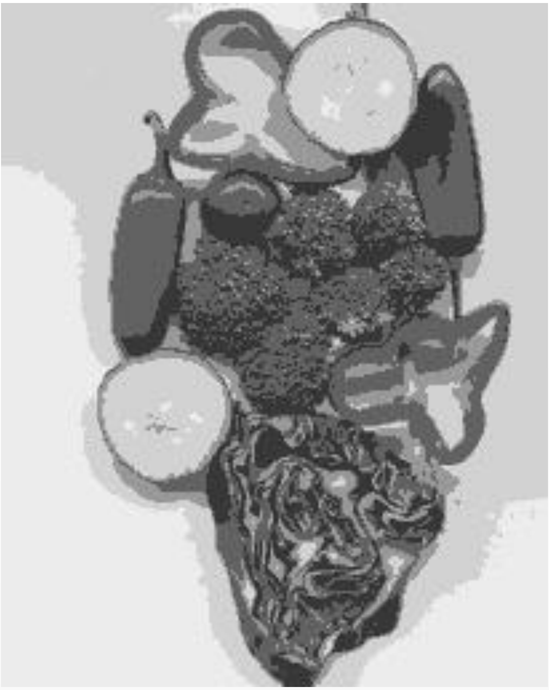
K-means clustering using intensity alone and color alone

Image

Clusters on intensity

Clusters on color





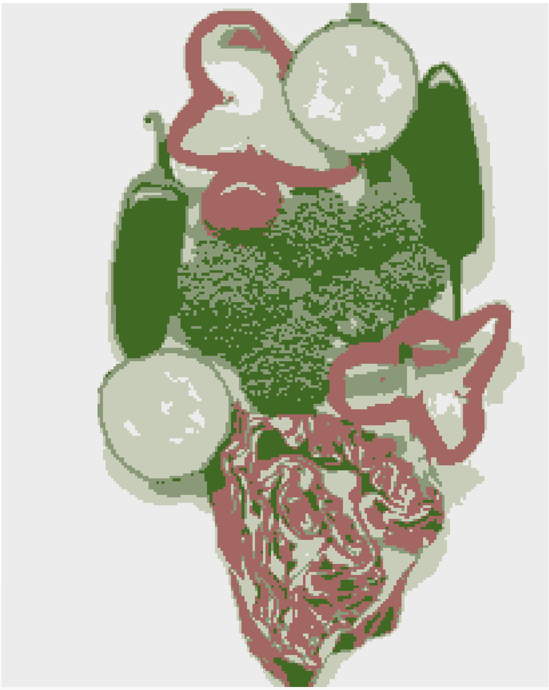


Image Segmentation

- Intensity Level Based Segmentation
  - Otsu’s Method
- Region-based Segmentation
  - Region growing
  - Region Splitting and Merging
- Clustering in the Feature Space

Intensity Level Based Segmentation

- Thresholding
  - **Assumption**: the image parts (e.g. object and background) can be separated based on their intensity level.

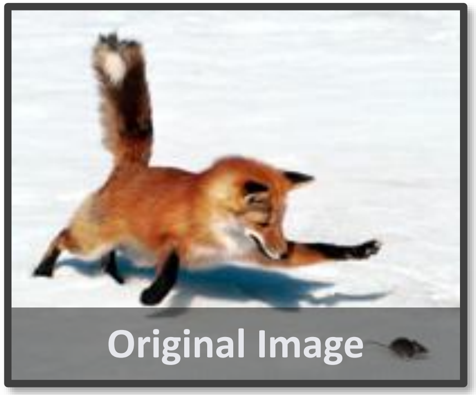
$$s(n_1, n_2) = \begin{cases} \text{object} & x(n_1, n_2) < T \\ \text{background} & x(n_1, n_2) \geq T \end{cases}$$

…where *$s(n_1,n_2)$* is the cluster of the *(n_1,n_2)* pixel of the *x* image and *T* is a threshold.

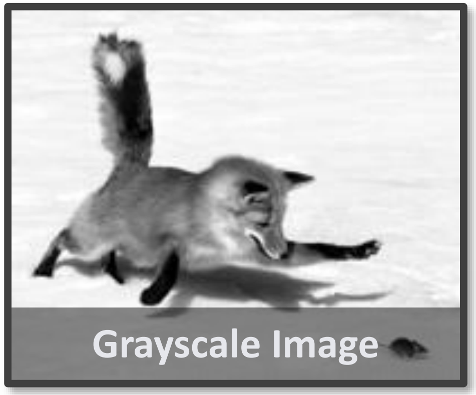
  - The main question is how to determine the threshold?

Intensity Level Based Segmentation

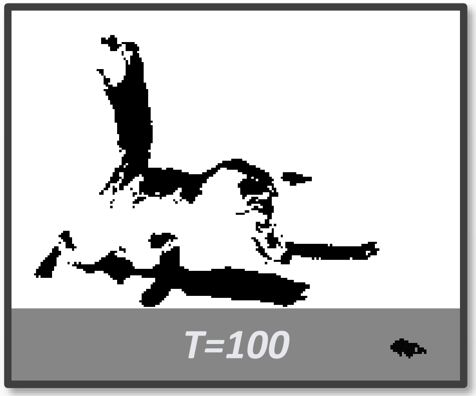
- Thresholding:
  - The main question is how to find the optimal threshold?



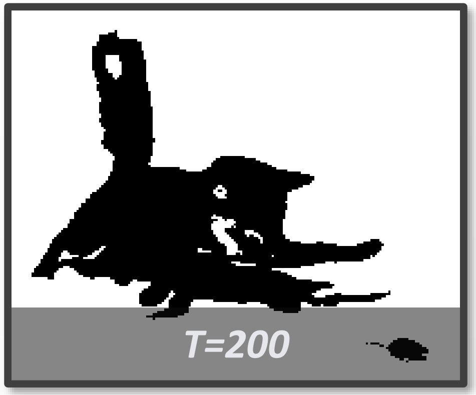
Original Image



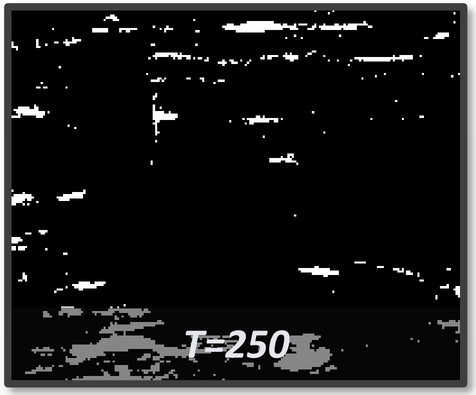
Grayscale Image



T=100



T=200



T=250

Intensity Level Based Segmentation

- Otsu's method:
  - Automatically determines the optimal global threshold by minimizing the intra-class variance.
  - The intra-class variance is defined as follows:

$$\sigma_w^2(k) = \omega_1(k)\sigma_1^2(k) + \omega_2(k)\sigma_2^2(k)$$

where ω_i and σ_i are the probability and the variance of the two classes separated by the threshold k.

  - Otsu showed that ***minimizing the intra-class variance is the same as maximizing inter-class variance***:

$$\sigma_b^2(k) = \sigma^2 - \sigma_w^2(k) = \omega_1(k)\omega_2(k)(\mu_1(k) - \mu_2(k))^2$$

where μ_i are the means of the two classes separated by threshold k.

Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". *IEEE Trans. Sys., Man., Cyber.* **9** (1): 62–66.

Intensity Level Based Segmentation

- Otsu’s method:

$$\sigma_b^2(k) = \sigma^2 - \sigma_w^2(k) = \omega_1(k)\omega_2(k)\left(\mu_1(k) - \mu_2(k)\right)^2$$

  - To calculate ω_i and μ_i the normalized histogram of the image is used:

$$\omega_1(k) = \sum_{i=0}^{k} p_i \qquad \omega_2(k) = \sum_{i=k+1}^{L-1} p_i$$

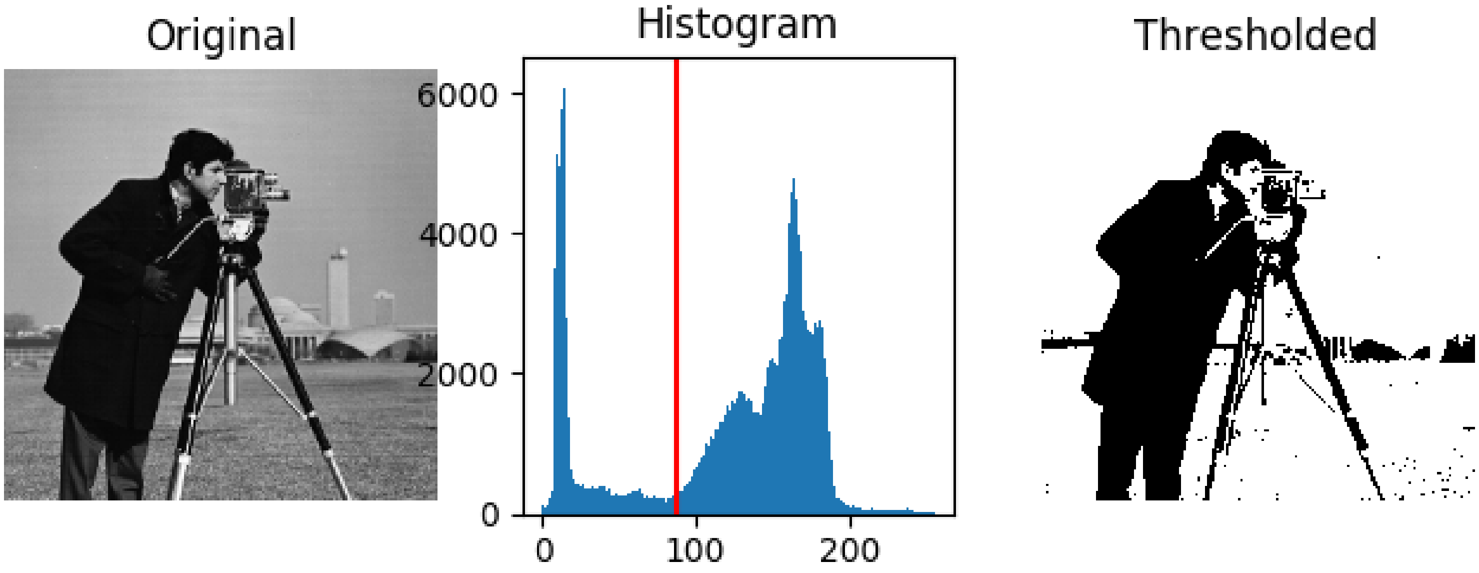
$$\mu_1(k) = \left(\sum_{i=0}^{k} i p_i\right) \Big/ \omega_1 \qquad \mu_2(k) = \left(\sum_{i=k+1}^{L-1} i p_i\right) \Big/ \omega_2$$

where p_i is the *i*-th entry in the normalized histogram of the image (probability of the *i*-th intensity level).

The Otsu threshold is the value that maximizes the inter-class variance.

* Nobuyuki Otsu (1979). "A threshold selection method from gray-level histograms". *IEEE Trans. Sys., Man., Cyber.* **9** (1): 62–66.

Result of Otsu’s method

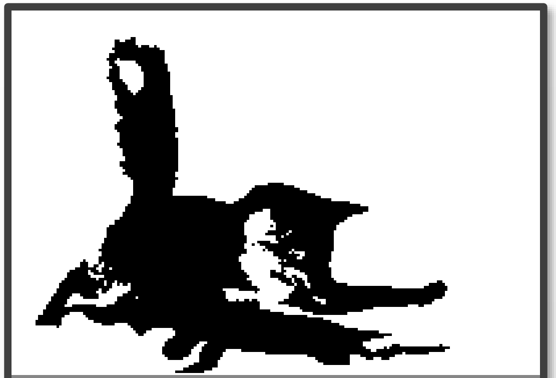


Result of Otsu’s method

- Otsu’s method:

Original Image

Grayscale Image



Otsu threshold: T=168

Region-Based Segmentation Methods

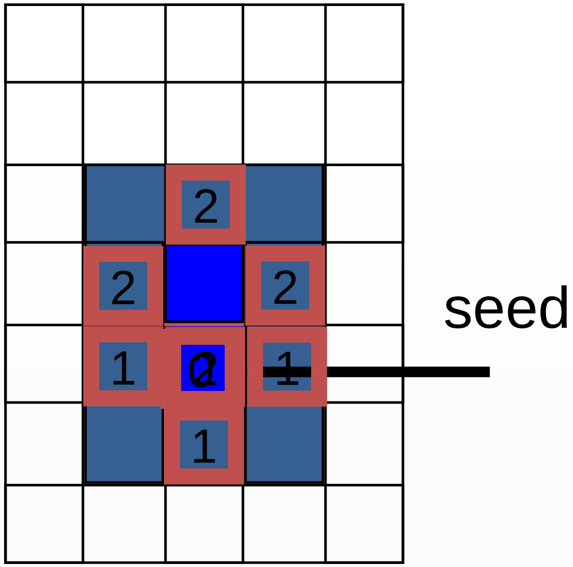
- Let R be the entire image region, and $R_1, \dots, R_n$ are subregions.
- We want to find a segmentation that is..
  - Complete: $\bigcup_{i=1}^{n} R_i = R$
  - Points in the region R_i $(i = 1, \dots, n)$ are connected
  - The regions are disjoint: $R_i \ \cap R_j = \emptyset$ for $\forall i \neq j$
  - All the pixels in a region has common properties…
  - …that they don't share with pixels from other regions.

Region-Based Segmentation Methods

- Region growing:
  - The method is initialized with a set of **seed points** as regions
  - We start growing the regions by adding neighboring pixels to the region if they has **similar predefined properties** as the seed points.
  - The seeds can be selected based on prior information, or evenly, or random...
  - The similarity criteria is usually depending on the segmentation result we want. (Commonly used properties are the intensity level, color, texture, motion,…)
  - **Pros:** simple, works well on images with clear edges, prior knowledge can be easily utilized, robust to noise…
  - **Cons:** time consuming

Region growing

- We start growing the regions by adding neighboring pixels to the region if they has **similar predefined properties** as the seed points

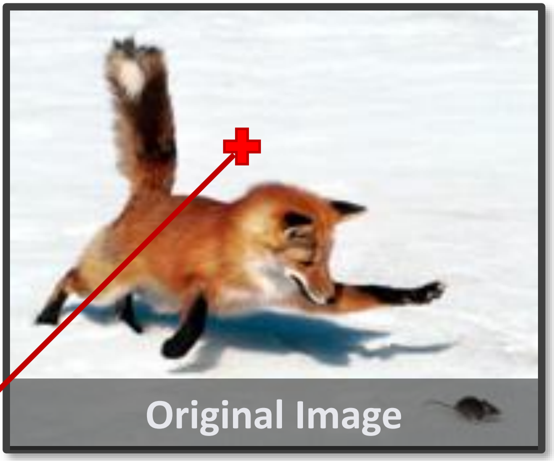


Internal point= dark blue

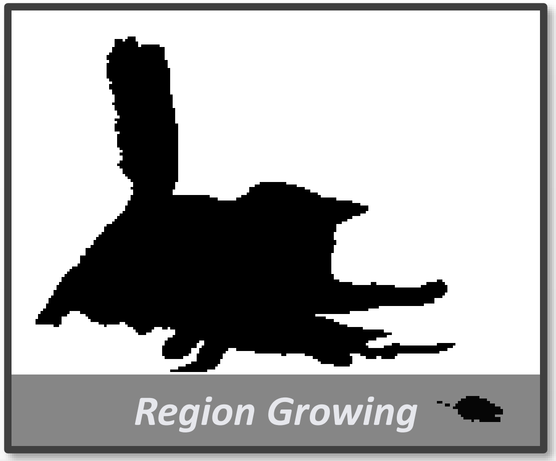
```
Flood(x, y) {
        if (pixel[x][y] is internal point ) {
                addToRegion(x, y);
                Flood(x, y-1);
                Flood(x, y+1);
                Flood(x-1, y);
                Flood(x+1, y);
        }
}
```

Region-Based Segmentation Methods

- Region growing:

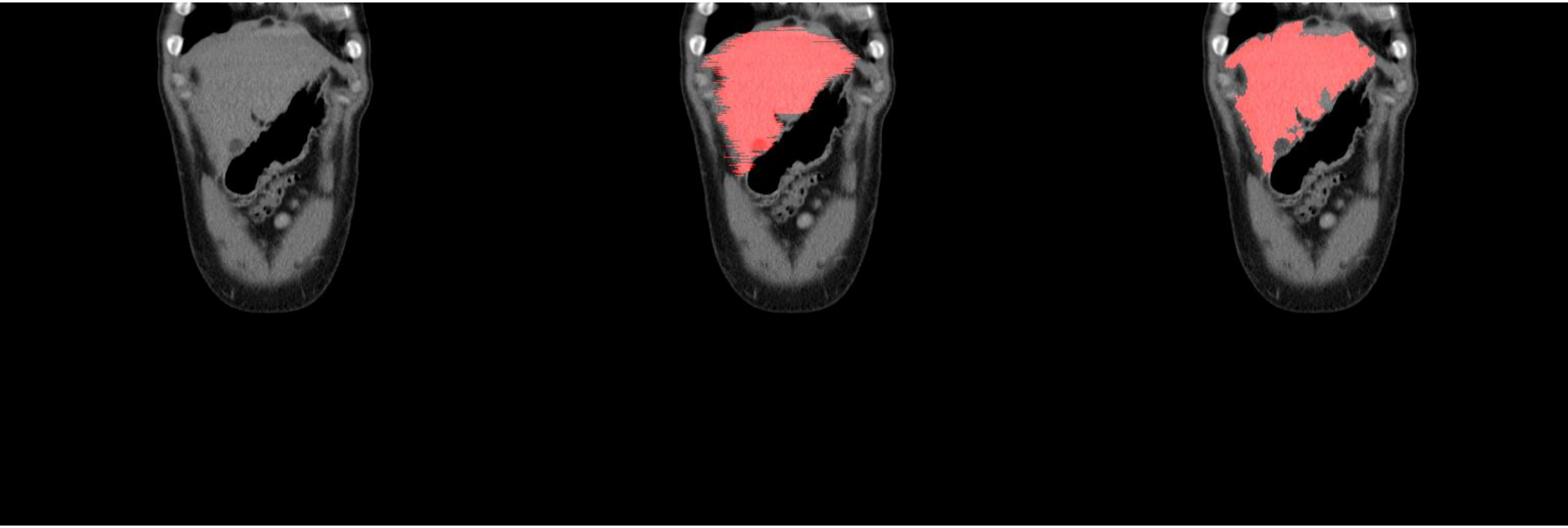


seed point



Region growing results on medical data

- Liver segmentation from CT

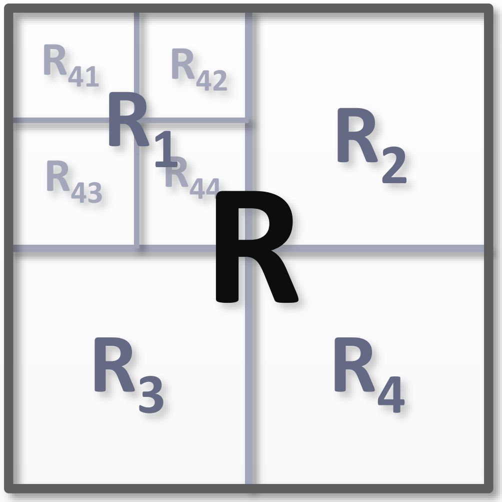


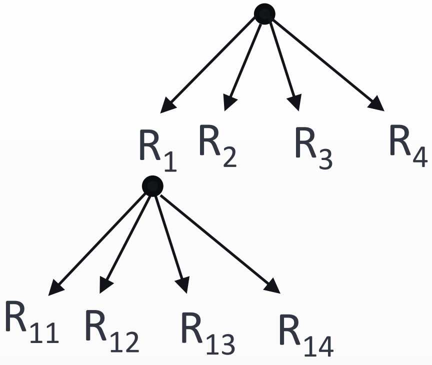
Manual Ground Truth

Region growing result

Region-Based Segmentation Methods

- Region splitting and merging:
  - Let ***R*** represent the entire image region and ***P*** be a predicate.
  - The splitting and merging steps are alternating:
    - We split the region $\boldsymbol{R_i}$ into 4 sub regions if $\boldsymbol{P(R_i)}$ = false
    - We merge 2 neighboring regions $\boldsymbol{R_i}$ and $\boldsymbol{R_j}$ if $\boldsymbol{P(R_i \cup R_j)}$ = true
  - The minimum region size has to be selected.



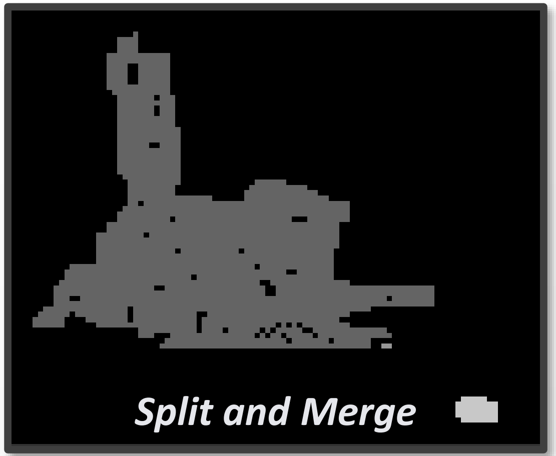


Region-Based Segmentation Methods

- Region splitting and merging:

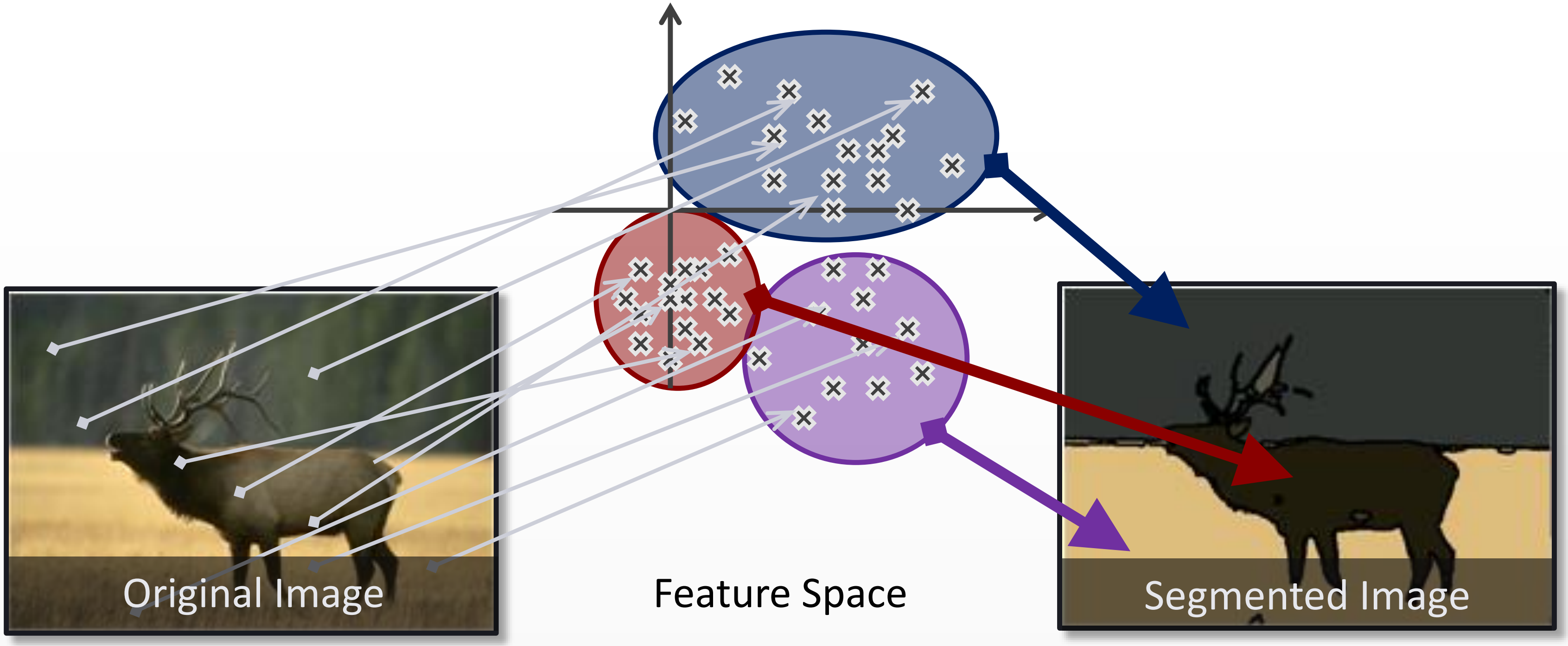
Original Image

Grayscale Image


Split and Merge

Clustering in the Feature Space

- A clustering algorithm is used to find structure in the data.
- The pixels are represented in the feature space.
- Usual features: colors, pixel coordinates, texture descriptors,..



Source of the Images: http://ivrgwww.epfl.ch/supplementary_material/RK_CVPR09/

Clustering in the Feature Space

- Partitioning-Clustering Approach
  - Learning a partition on a data set to produce several non-empty clusters
  - Assume that the number of clusters, K, is given in advance
  - Given a *K*, find a partition of *K clusters* to optimize the chosen partitioning criterion (cost function)
  - In principle, optimal partition $S = \{S_1, \dots S_K\}$ achieved *via minimizing the sum of squared distance to its "representative object" in each cluster*

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} d^2(x, \mu_i)$$

  - global optimum: exhaustively search all partitions: **too expensive!**
  - a typical clustering analysis approach via **iteratively** partitioning training data set to learn a partition of the given data space

K-means Algorithm

- *K-means* algorithm (MacQueen'67): a heuristic method
  - Each cluster is represented by the centre of the cluster and the algorithm converges to stable centroids of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.
  - Each sample will belong to the cluster with the nearest mean.
- The objective is to minimize the within-cluster sum of squares:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in S_i} d^2(x, \mu_i), \qquad d^2(x, \mu_i) = \|x - \mu_i\|^2 = \sum_{n=1}^{N} (x_n - \mu_{in})^2$$

  - where $x \in \mathbb{R}^N$ are the data samples, μ_i is the mean (prototype) of the points in the cluster S_i $(i = 1 \ldots K)$.

K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialization:

1) **Initialisation**: set the K cluster seed points (randomly)

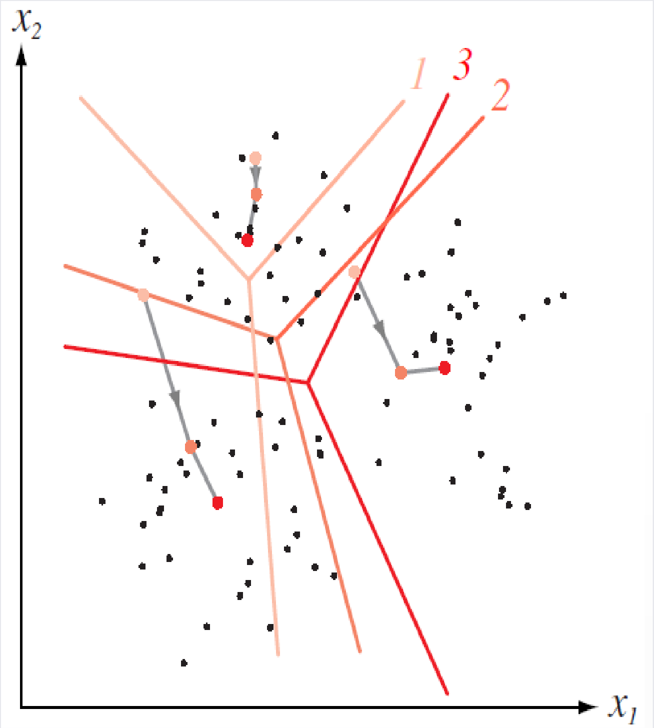
2) **Assignment step**: Assign each object to the cluster of the nearest seed point measured with a specific distance metric

3) **Update step**: Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., ***mean point***, of the cluster)

$$\mu_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

4) **Go back** to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

Understanding K-means

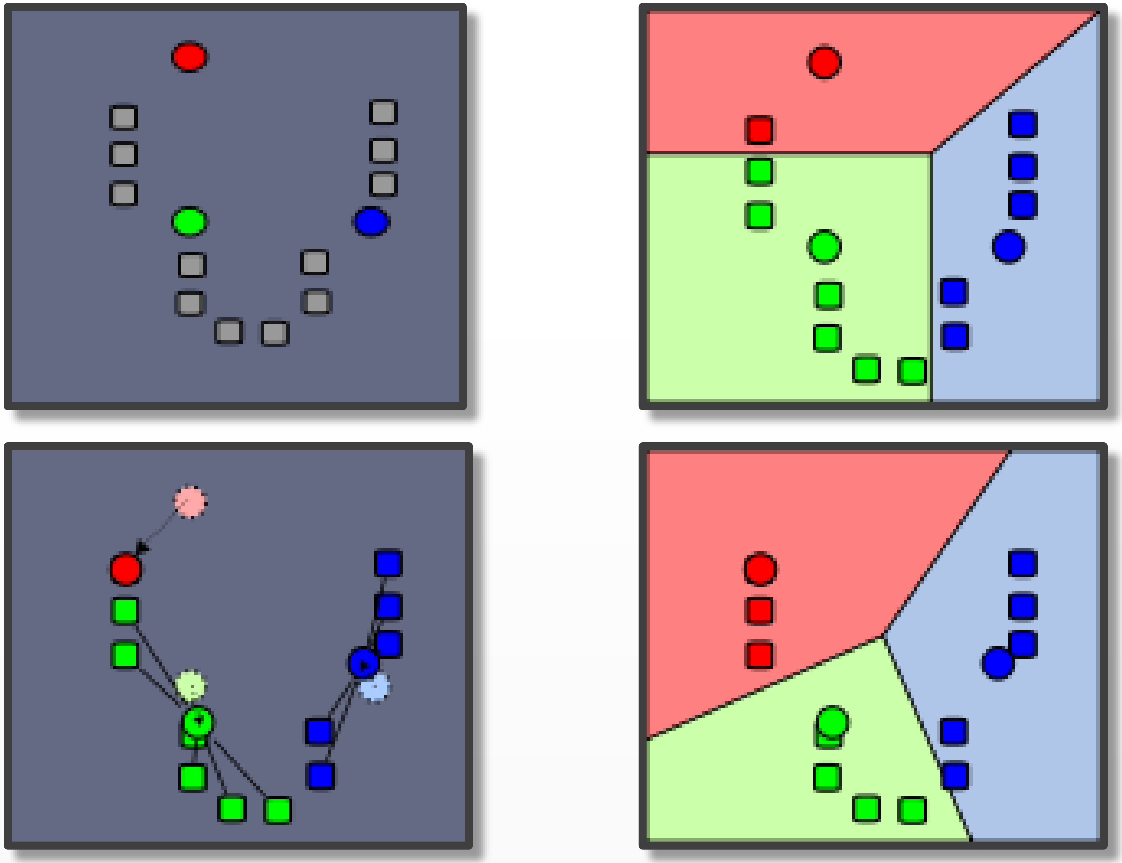


- How K-means partitions?
  - When *K* centroids are set/fixed, they partition the whole data space into *K* mutually exclusive subspaces to form a partition.
  - A partition amounts to a *Voronoi* diagram
  - Changing positions of centroids leads to a new partitioning.

- Efficient in computation ☺
  - $O(tKn)$, where n is number of objects (eg. pixels), K is number of clusters, and t is number of iterations. Normally, $K, t \ll n$.

K-Means Clustering

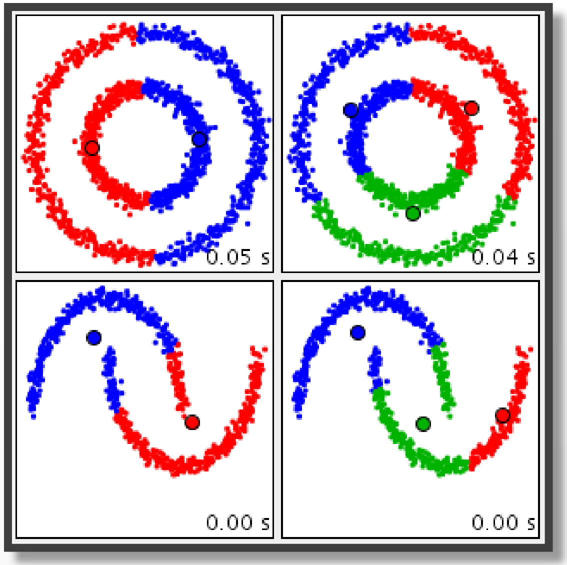
- Illustration of K-means iteration:

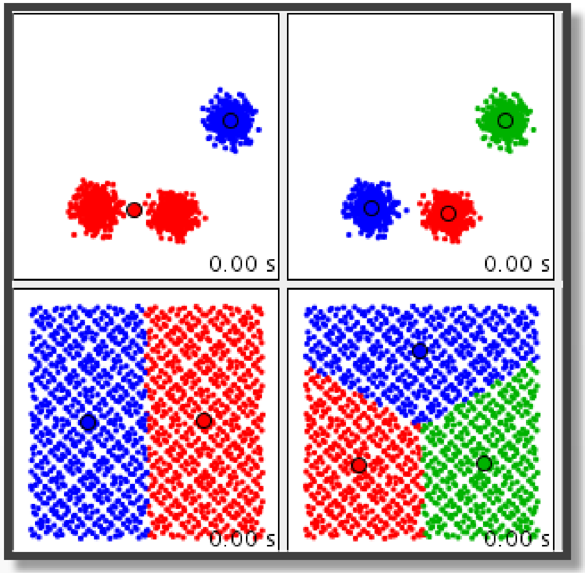


Source of the images: http://en.wikipedia.org/wiki/K-means_clustering

K-Means Clustering - Relevant Issues

- Limitation of K-means:
  - Number of clusters has to be known a priori priori (specify K in advance)
  - Sensitive to initial seed points, could stuck in a local minimum
  - Spherical clusters: not suitable for discovering clusters with non-convex shapes





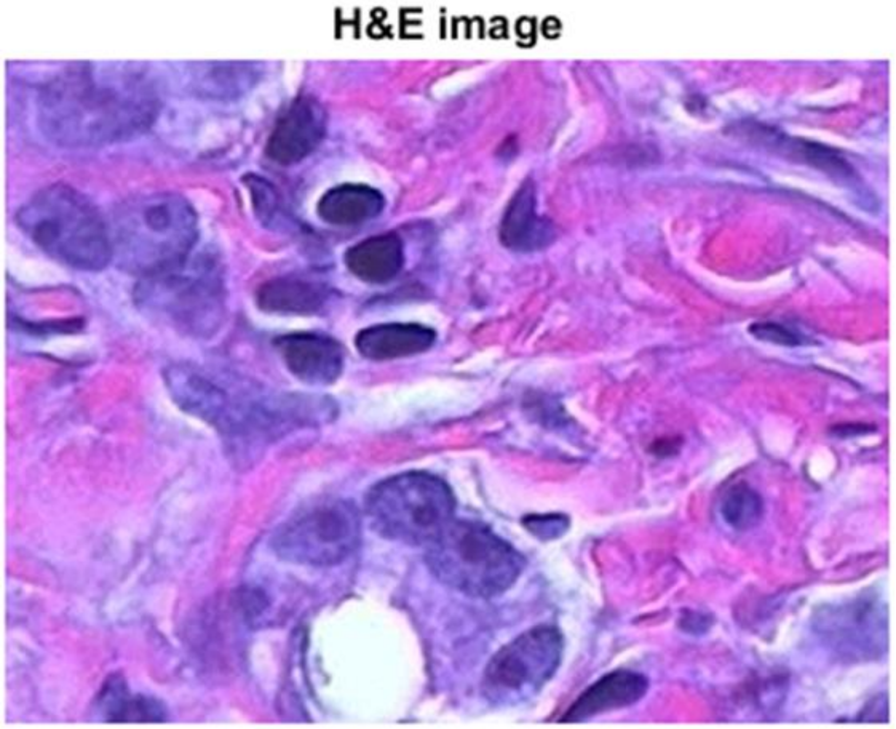
Source of the images: http://commons.apache.org/proper/commons-math/userguide/ml.html

Relevant Issues

- Other issues
  - Unable to handle noisy data and outliers (K-Medoids algorithm)
  - Applicable only when mean is defined, then what about categorical data? (K-mode algorithm)
  - How to evaluate the K-mean performance?

Color-Based Image Segmentation Using K-means

- **Step 1**: Loading a color image of tissue stained with hemotoxylin and eosin (H&E)

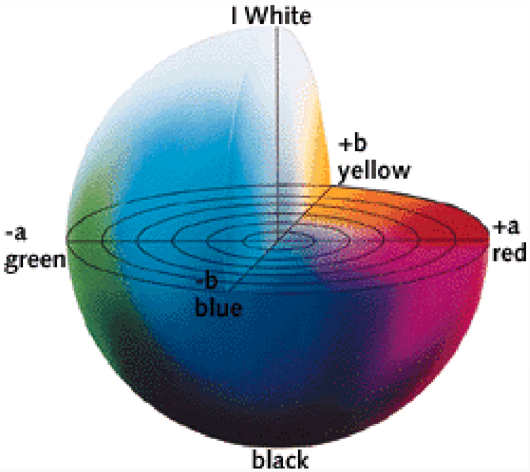


H&E image

Image courtesy of Alan Partin, Johns Hopkins University

Color-Based Image Segmentation Using K-means

- **Step 2:** Convert the image from *RGB* color space to $CIE\ L^{*}a^{*}b^{*}$color space (ReCap from *Lecture 1*)
  - Unlike the RGB color model, $CIE\ L^{*}a^{*}b^{*}$ color is designed to approximate human vision: *brightness* and *color shade* components of the pixel values are encoded in different channels
  - There is a complicated transformation between *RGB* and $CIE\ L^{*}a^{*}b^{*}$
    - $(L^{*}a^{*}b^{*})$= T*(R, G, B).*
    - *(R, G, B)* = T'$(L^{*}a^{*}b^{*})$
  - The **brightness (L)** increases from the bottom to the top of the three-dimensional model.
  - **Color shades:** The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b).

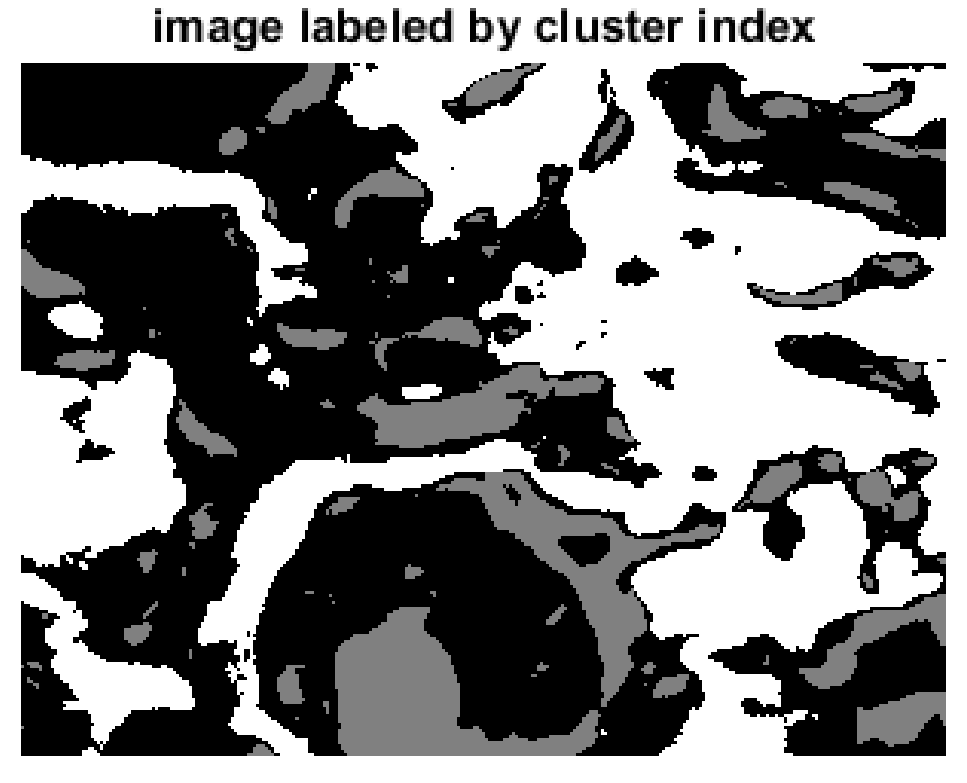


Color-Based Image Segmentation Using K-means

- **Step 3:** Undertake clustering analysis in the (a*, b*) color space with the *K-means* algorithm
  - During feature selection, L* feature is discarded. As a result, each pixel has a 2D feature vector $x = [\mathrm{a}*, \mathrm{b}*] \in \mathbb{R}^2$.
  - Applying the *K-means* algorithm to the image in the a*b* feature space where K = 3 (by applying the domain knowledge).

Color-Based Image Segmentation Using K-means

- **Step 4:** Label every pixel in the image using the results from
  - K-means clustering (indicated by three different grey levels



Color-Based Image Segmentation Using K-means

- **Step 5:** Create Images that Segment the H&E Image by Color
  - Apply the label and the color information of each pixel to achieve separate color images corresponding to three clusters.

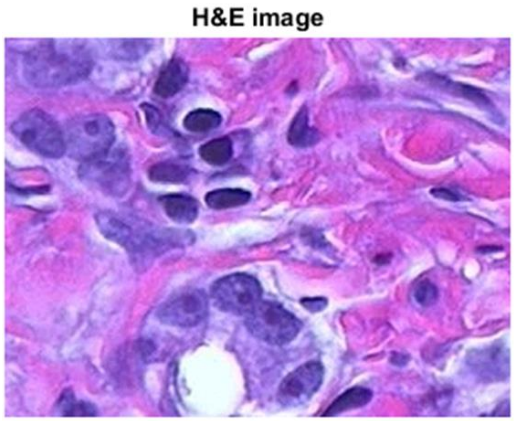
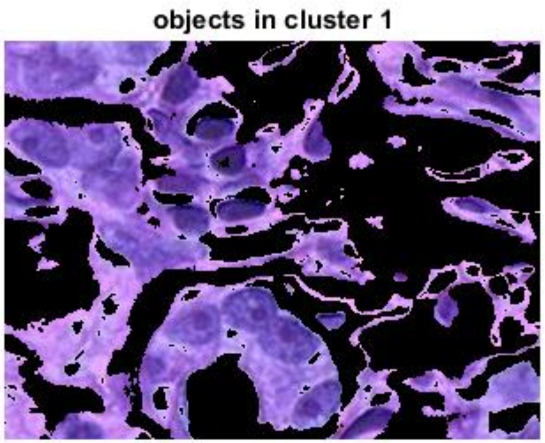
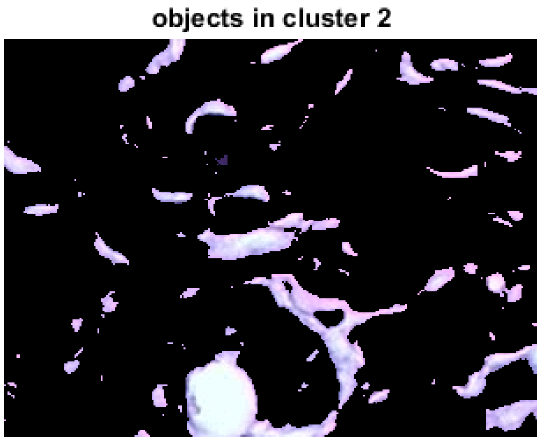


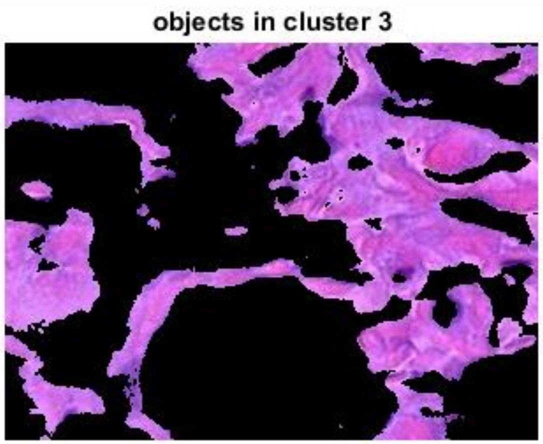
Image courtesy of Alan Partin, Johns Hopkins University



“blue” pixels



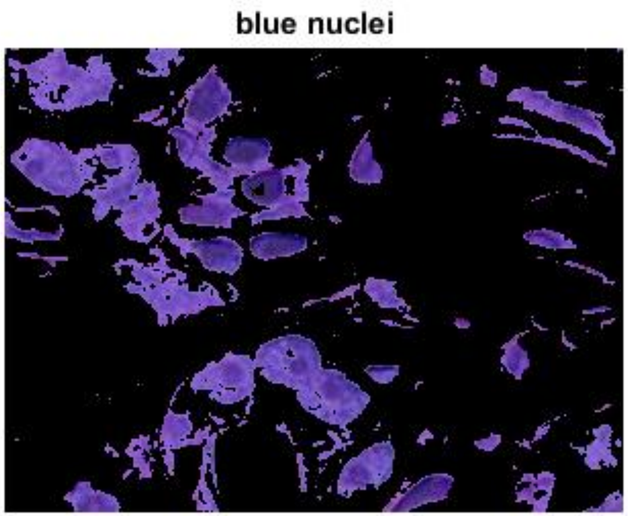
“white” pixels



“pink” pixels

Color-Based Image Segmentation Using K-means

- **Step 6:** Segment the nuclei into a separate image with the L* feature
  - In cluster 1, there are dark and light blue objects (pixels). The dark blue objects (pixels) correspond to nuclei (with the domain knowledge).
  - L* feature specifies the brightness values of each colour.
  - With a threshold for L*, we achieve an image containing the nuclei only.



Summary: K-means

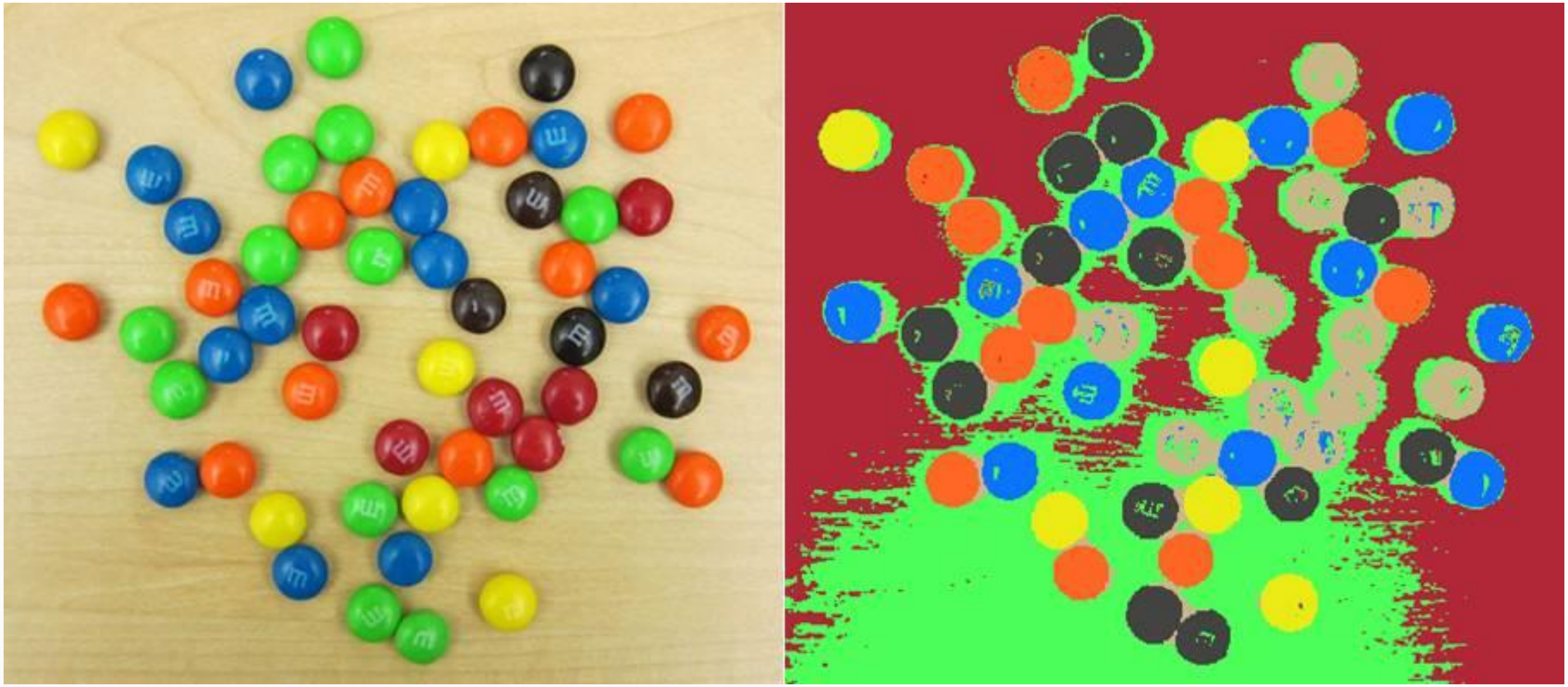
- ***K*-means** algorithm is a simple yet popular method for clustering analysis
- Its performance is determined by initialisation and appropriate distance measure
- There are several **variants** of *K*-means to overcome its weaknesses
  - *K*-Medoids: resistance to **noise and/or outliers**
  - *K*-Modes: extension to **categorical data** clustering analysis
  - CLARA: extension to deal with **large data** sets
  - Mixture models (EM algorithm): handling **uncertainty** of clusters

Online tutorial: how to use **the *K*-means function in Matlab**
https://www.youtube.com/watch?v=aYzjenNNOcc

Discussed image segmentation example:
https://www.mathworks.com/help/images/examples/color-based-segmentation-using-k-means-clustering.html

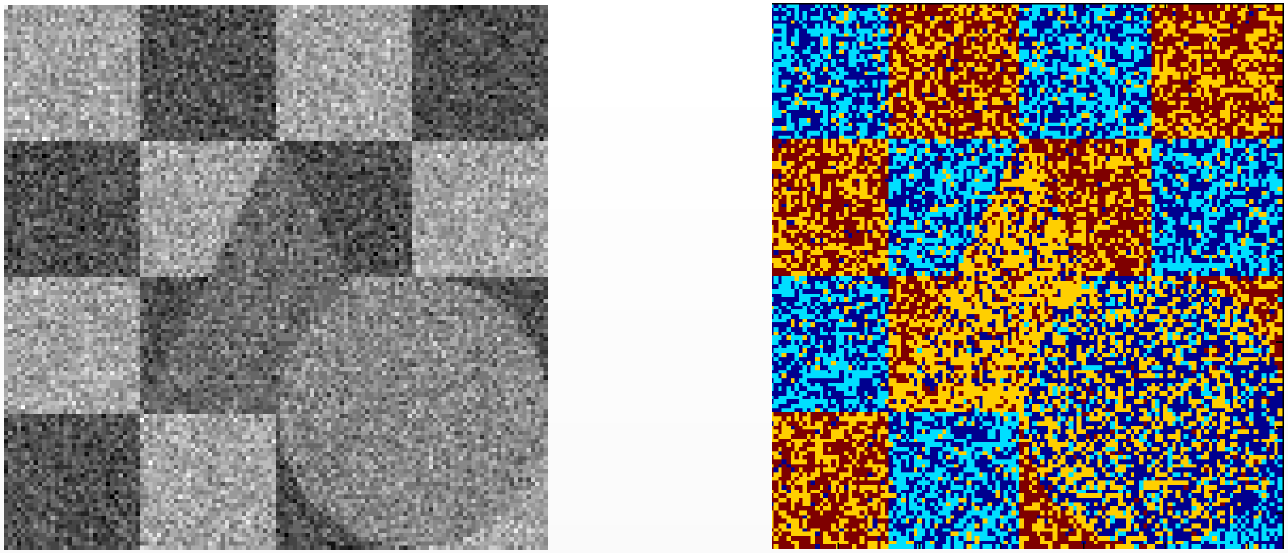
K-Means : some further results

- Segmentation in RGB color space can also work…



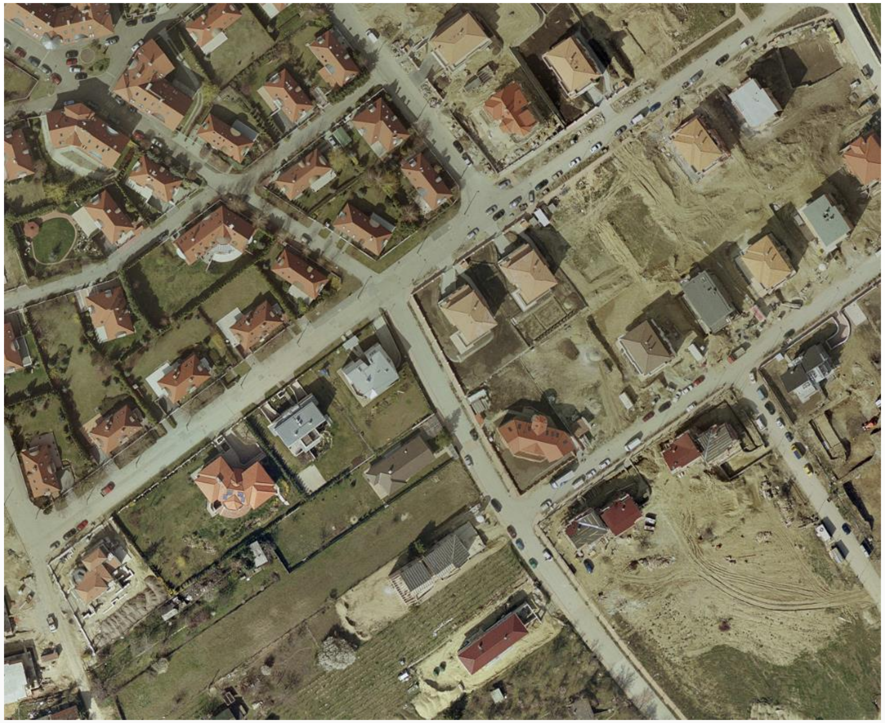
K-Means: some further results

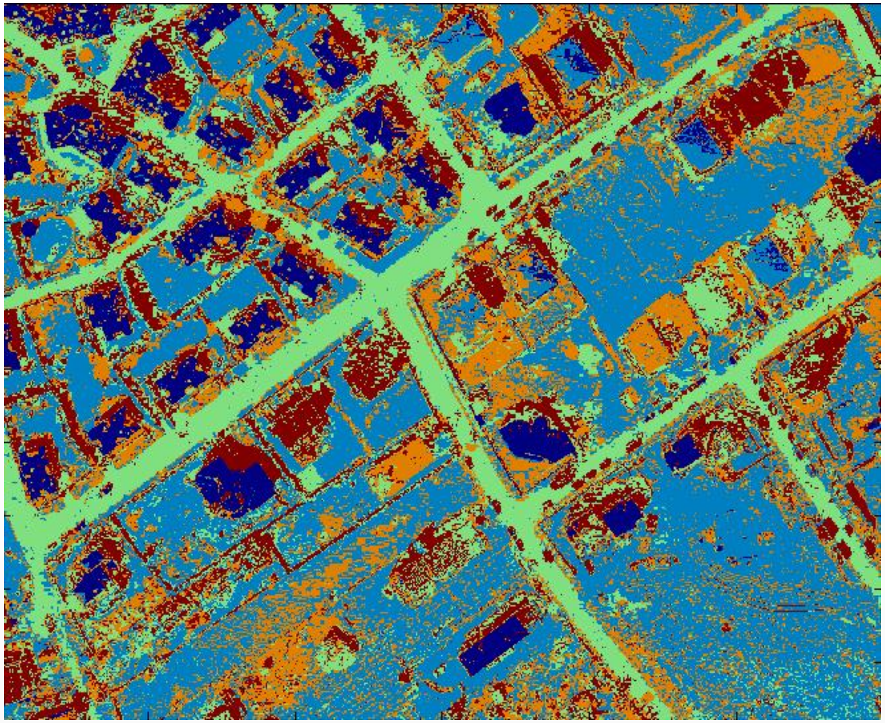
- Segmentation of a noisy grayscale image
  - Gaussian white noise, K=4



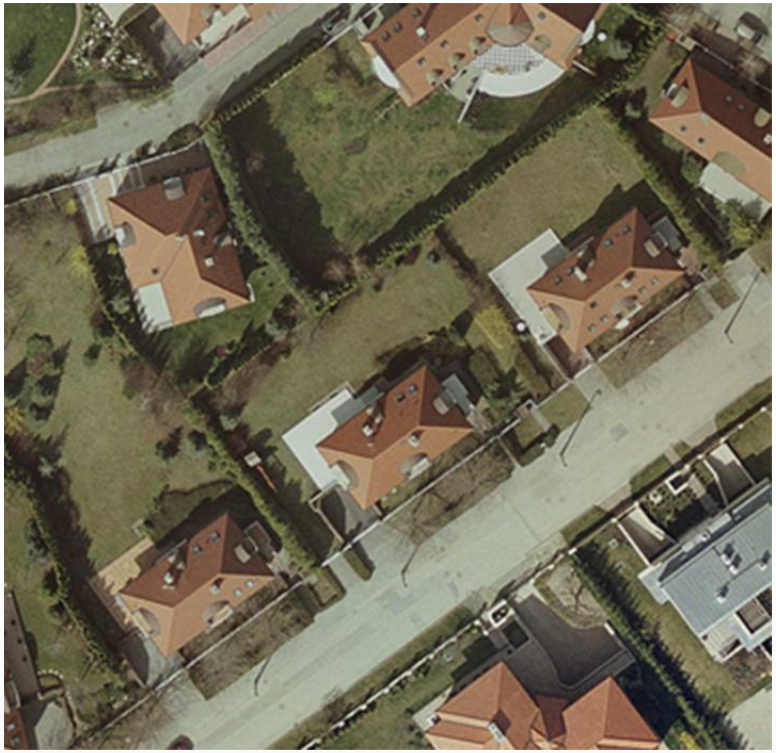
K-Means result

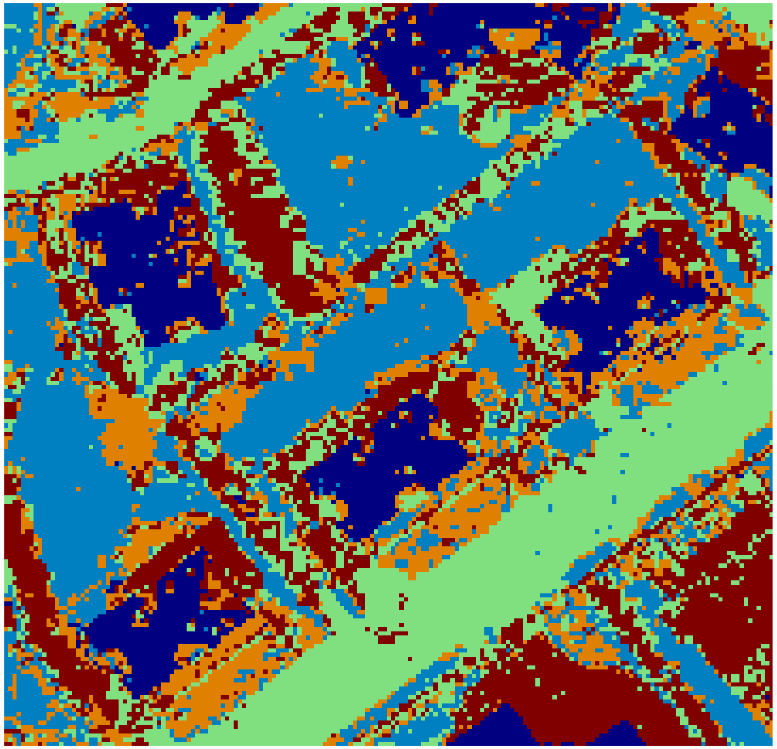
- Segmentation of an aerial image in $CIE\ L^*a^*b^*$, feature channels: (a^*b^*), $K = 5$



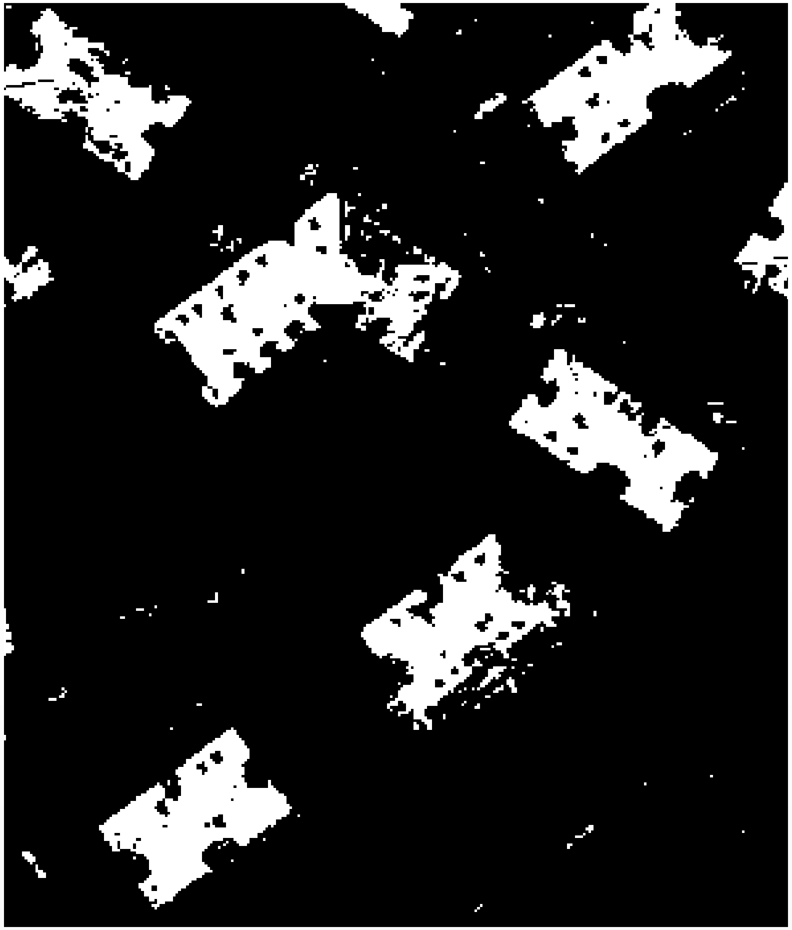


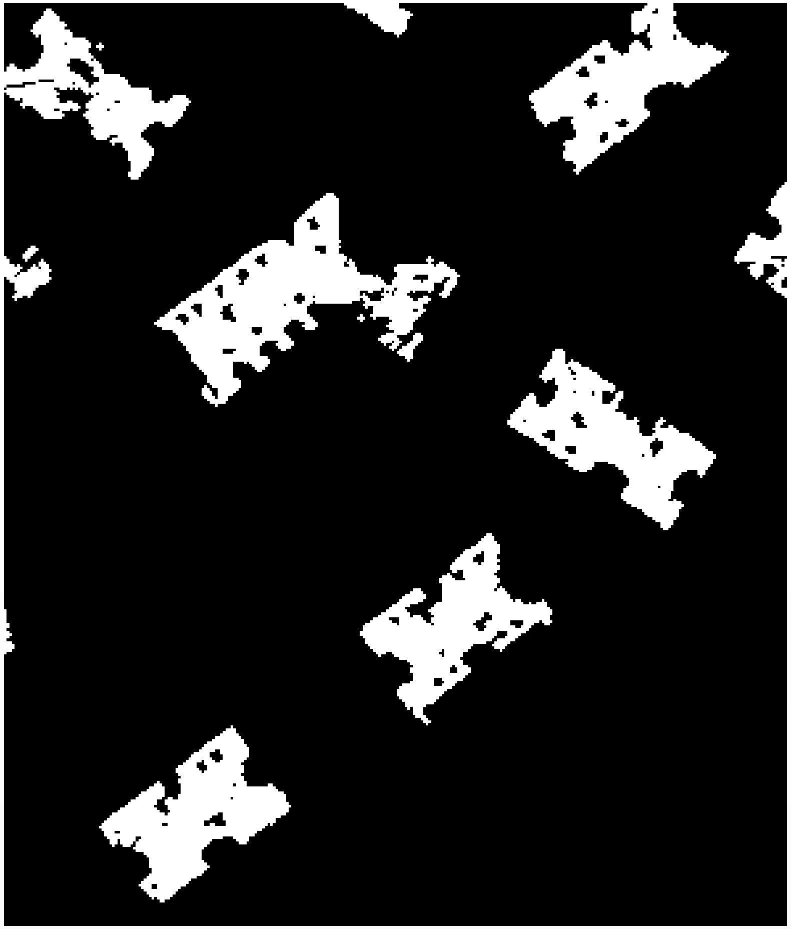
Per-pixel segmentation: noisy ouptut





Post processing: enhancing the regions by compactness and shape analysis





Morphology - overview

- Once segmentation is complete, morphological operations can be used to remove imperfections in the segmented image and provide information on the form and structure of the image
- In this section we will consider
  - What is morphology?
  - Simple morphological operations
  - Compound operations
  - Morphological algorithms

Slides for dilation/erosion credits: Dublin Institute of Technology

What Is Morphology?

- Morphological image processing (or **morphology**) describes a range of image processing techniques that deal with the shape (or morphology) of features in an image
- Morphological operations are typically applied to **remove imperfections** introduced during segmentation, and so typically operate on **bi-level images**

Morphological Operations: details 1, 0, Black, White?

- Throughout all of the following slides whether 0 and 1 refer to white or black is a little interchangeable
- All of the discussion that follows assumes segmentation has already taken place and that images are made up of 0s for background pixels and 1s for object pixels
- After this it doesn’t matter if 0 is black, white, yellow, green…….

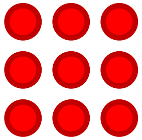
Morphological Operations

- Morphological operations are affecting the form, structure or shape of an object.
- They are used in pre- or postprocessing (filtering, thinning, and pruning) or for getting a representation or description of the shape of objects/regions (boundaries, skeletons convex hulls).
- Two basic operations:
  - **Dilation**: expands the object, fills in small holes and connects disjoint objects.
  - **Erosion**: shrinks objects by removing (eroding) their boundaries.
- The basic idea in binary morphology is to probe an image with a ***structuring element*** (a simple, pre-defined shape), drawing conclusions on how this shape fits or misses the shapes in the image.

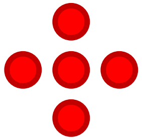
Morphological Operations

- Structuring element:

  e.g.:



8 neighbors



4 neighbors

- Dilation:
  - A shift-invariant operator, that expands the object, fills in small holes and connects disjoint objects.
  - Steps:
    - The structuring element is placed on each pixel on the image
    - If the pixel belongs to the foreground pixel, we do nothing
    - If the pixel belongs to the background, we change it to a foreground pixel if any pixel covered by the structuring element is a foreground pixel.

Morphological Operations

- Erosion:
  - A shift-invariant operator, that erodes away the boundaries of regions of foreground pixels. Thus areas of foreground pixels shrink in size, and holes within those areas become larger.
  - Steps:
    - The structuring element is placed on each pixel on the image
    - If the pixel is a background pixel, we do nothing
    - If the pixel is a foreground pixel, we change this pixel to a background if any pixel covered by the structuring element is a background pixel.
- Erosion on the image has the same effect as dilatation on the inverse image.
- **Opening**: Erosion + Dilation
- **Closing**: Dilation + Erosion

Quick Example

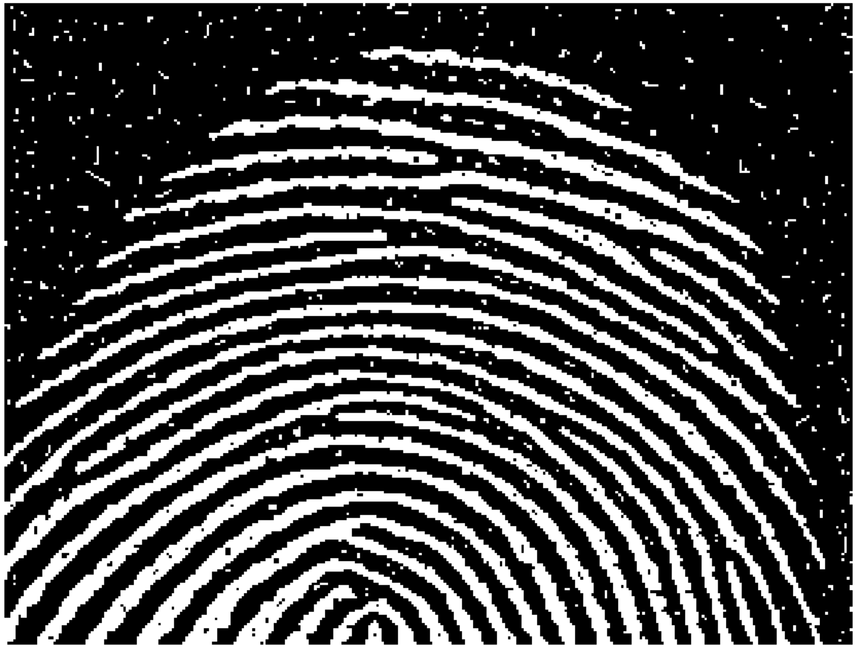


Image after segmentation

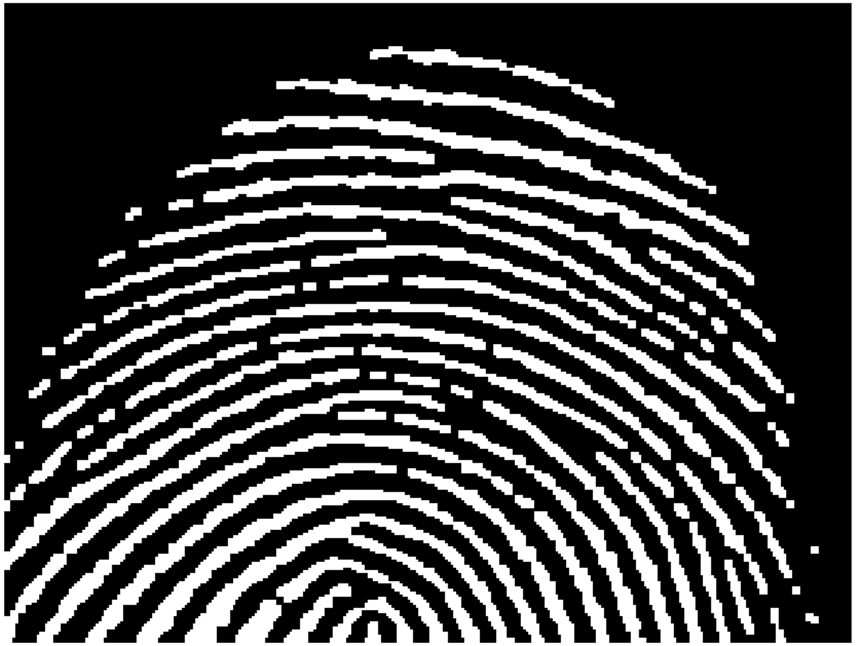
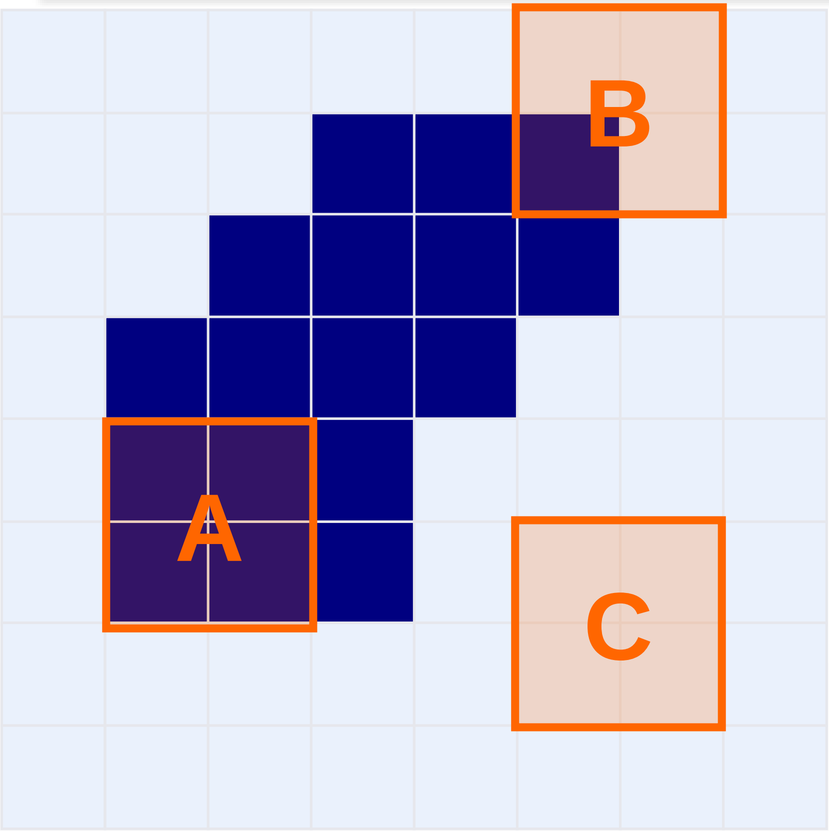
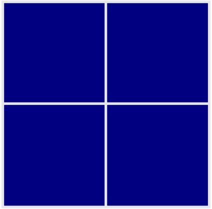


Image after segmentation and morphological processing

Structuring Elements, Hits & Fits





Fit: All *on pixels* in the structuring element cover *on pixels* in the image

Hit: Any *on pixel* in the structuring element covers an *on pixel* in the image

All morphological processing operations are based on these simple ideas

Structuring Elements

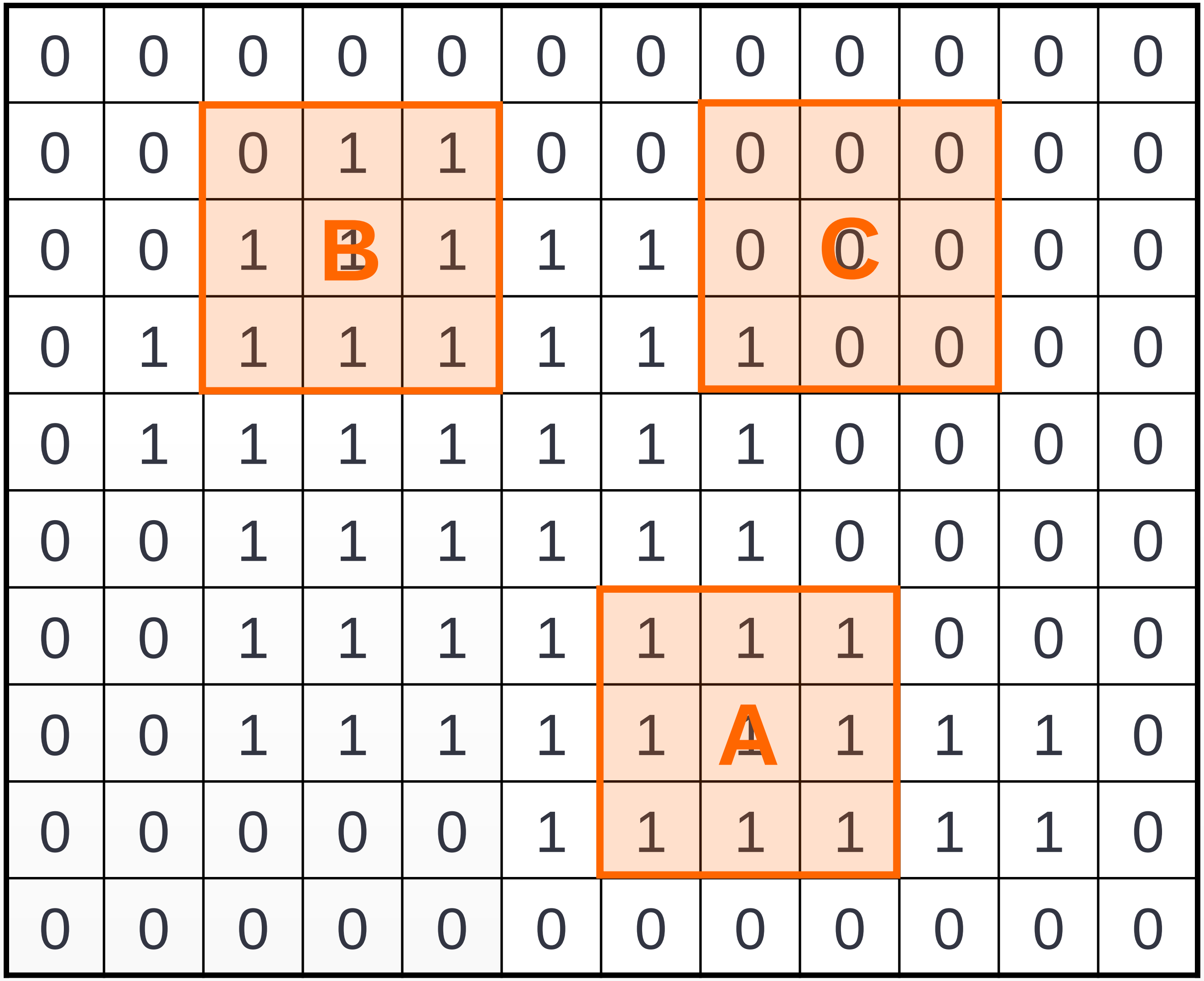
- Structuring elements can be any size and make any shape
- However, for simplicity we will use rectangular structuring elements with their origin at the middle pixel.
    - 1s represent the *on pixels* of the structuring element

1	1	1
1	**1**	1
1	1	1

0	1	0
1	**1**	1
0	1	0

0	0	1	0	0
0	1	1	1	0
1	1	**1**	1	1
0	1	1	1	0
0	0	1	0	0

Fitting & Hitting



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

1	1	1
1	1	1
1	1	1

Structuring Element 1

0	1	0
1	1	1
0	1	0

Structuring Element 2

Fundamental Operations

- Fundamentally morphological image processing is very like spatial filtering
- The structuring element is moved across every pixel in the original image to give a pixel in a new processed image
- The value of this new pixel depends on the operation performed
- There are two basic morphological operations: **erosion** and **dilation**

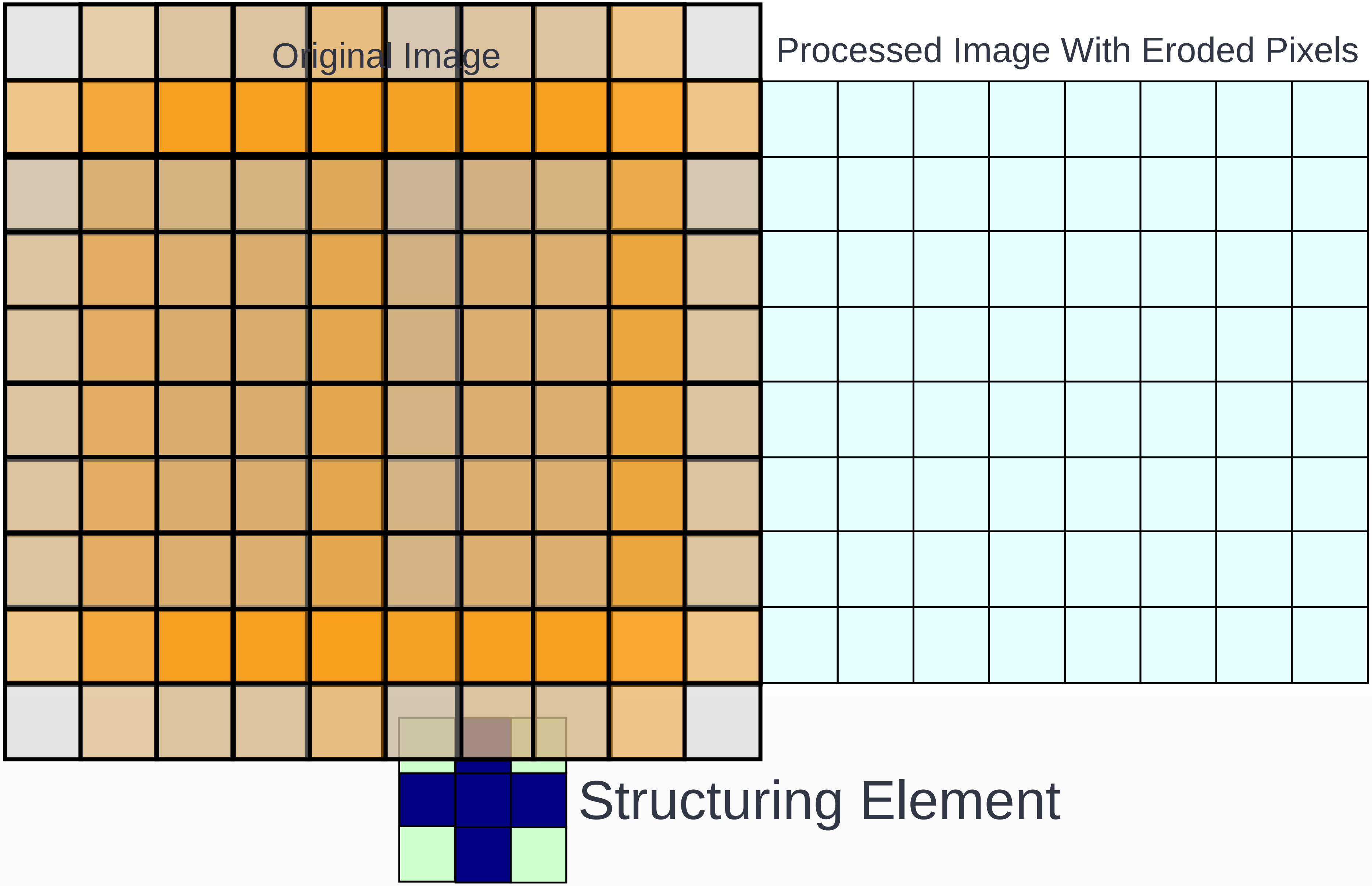
Erosion

- **Erosion** of image f by structuring element s is given by $f \ominus s$

The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1 & \text{if } s \text{ fits } f \\ 0 & \text{otherwise} \end{cases}$$

Erosion Example

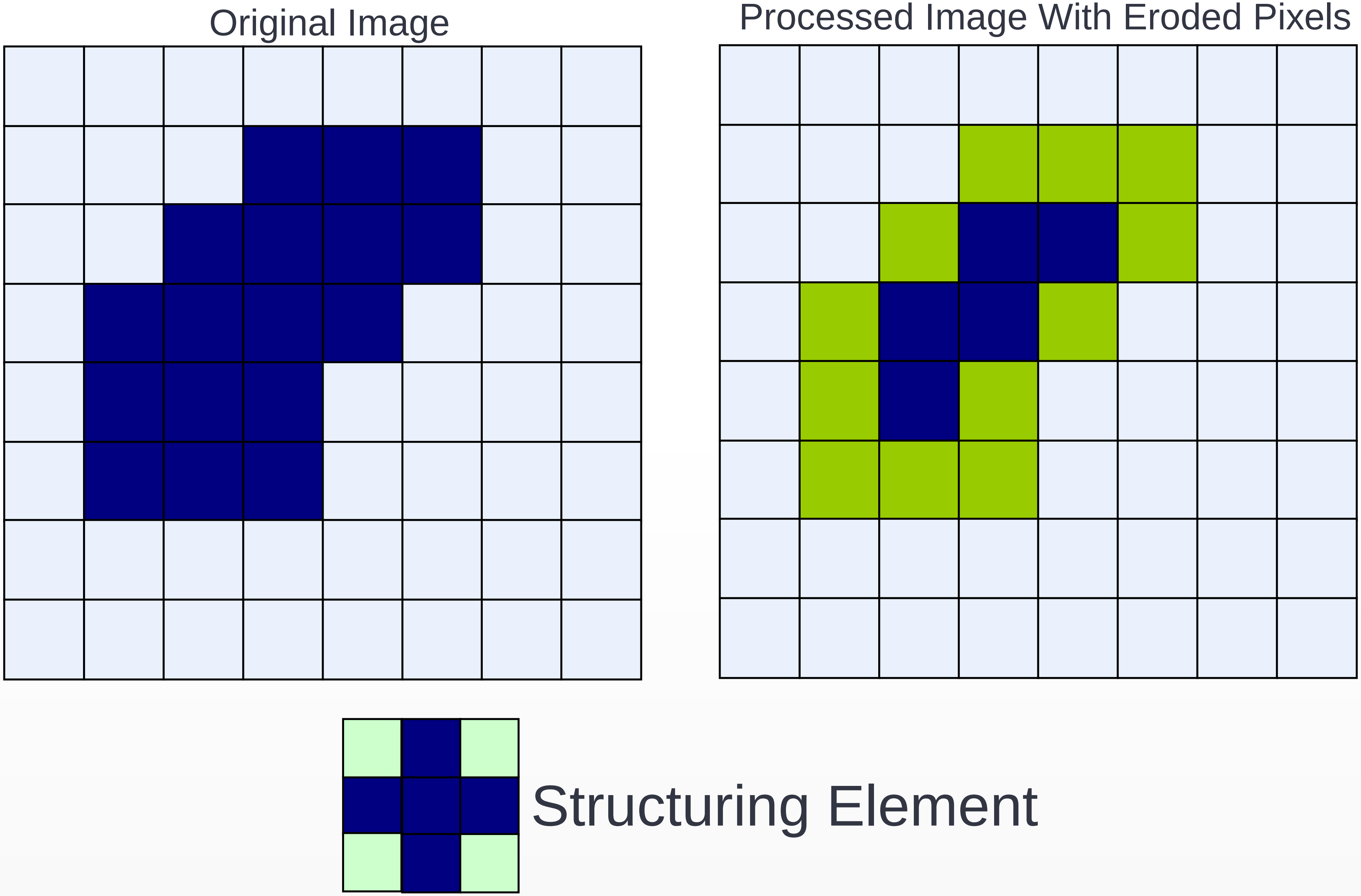


Original Image

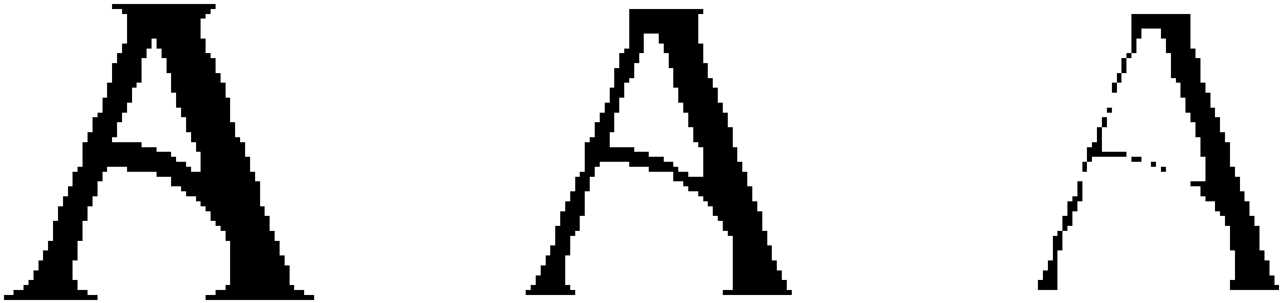
Processed Image With Eroded Pixels

Structuring Element

Erosion Example

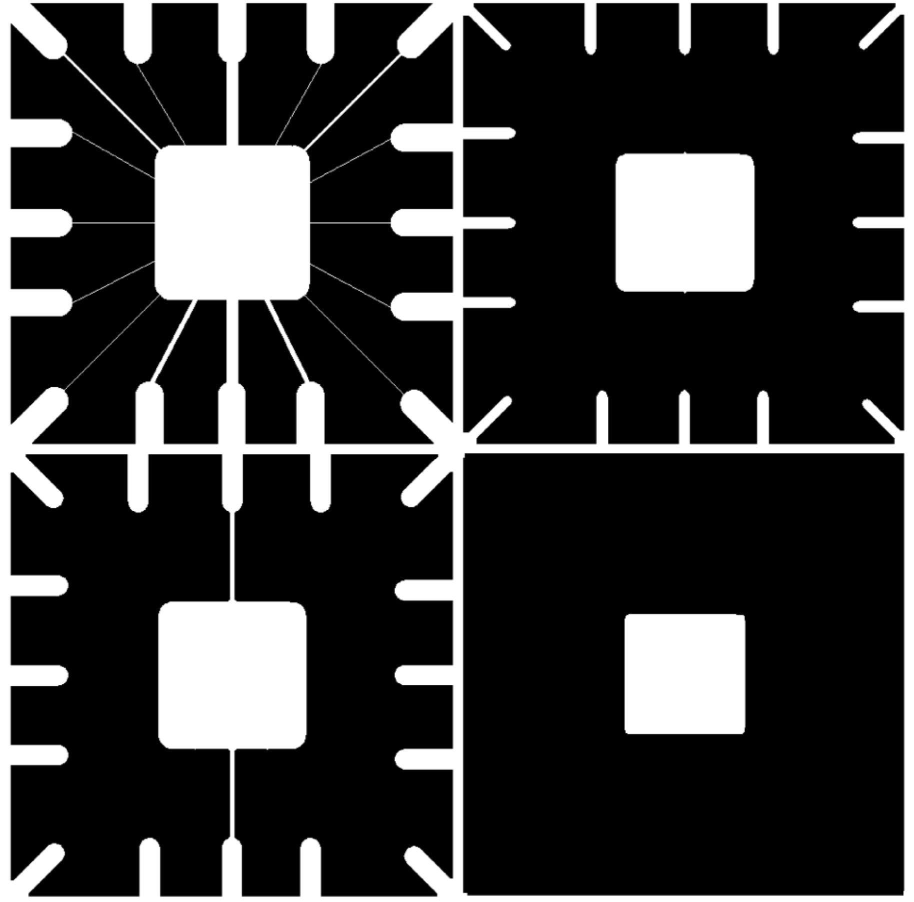


Erosion Example 1



Watch out: In these examples a 1 refers to a black pixel!

Erosion Example 2



Original image

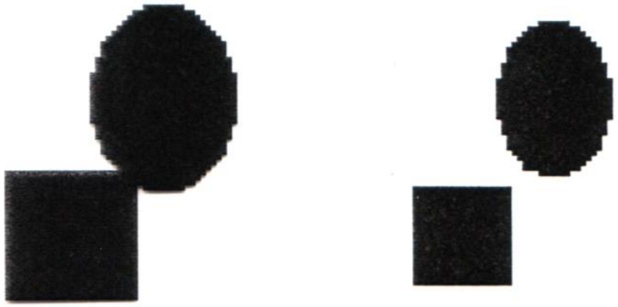
After erosion with a disc of radius 10

After erosion with a disc of radius 5

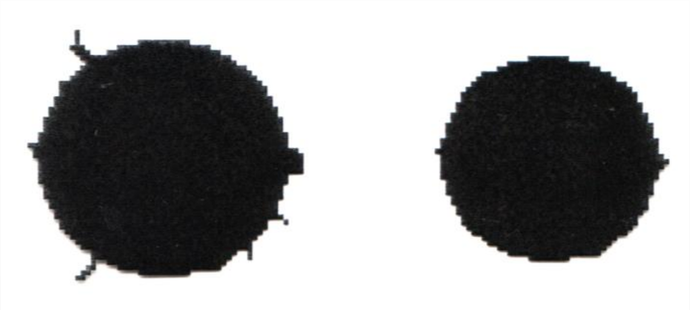
After erosion with a disc of radius 20

What Is Erosion For?

Erosion can split apart joined objects



Erosion can strip away extrusions



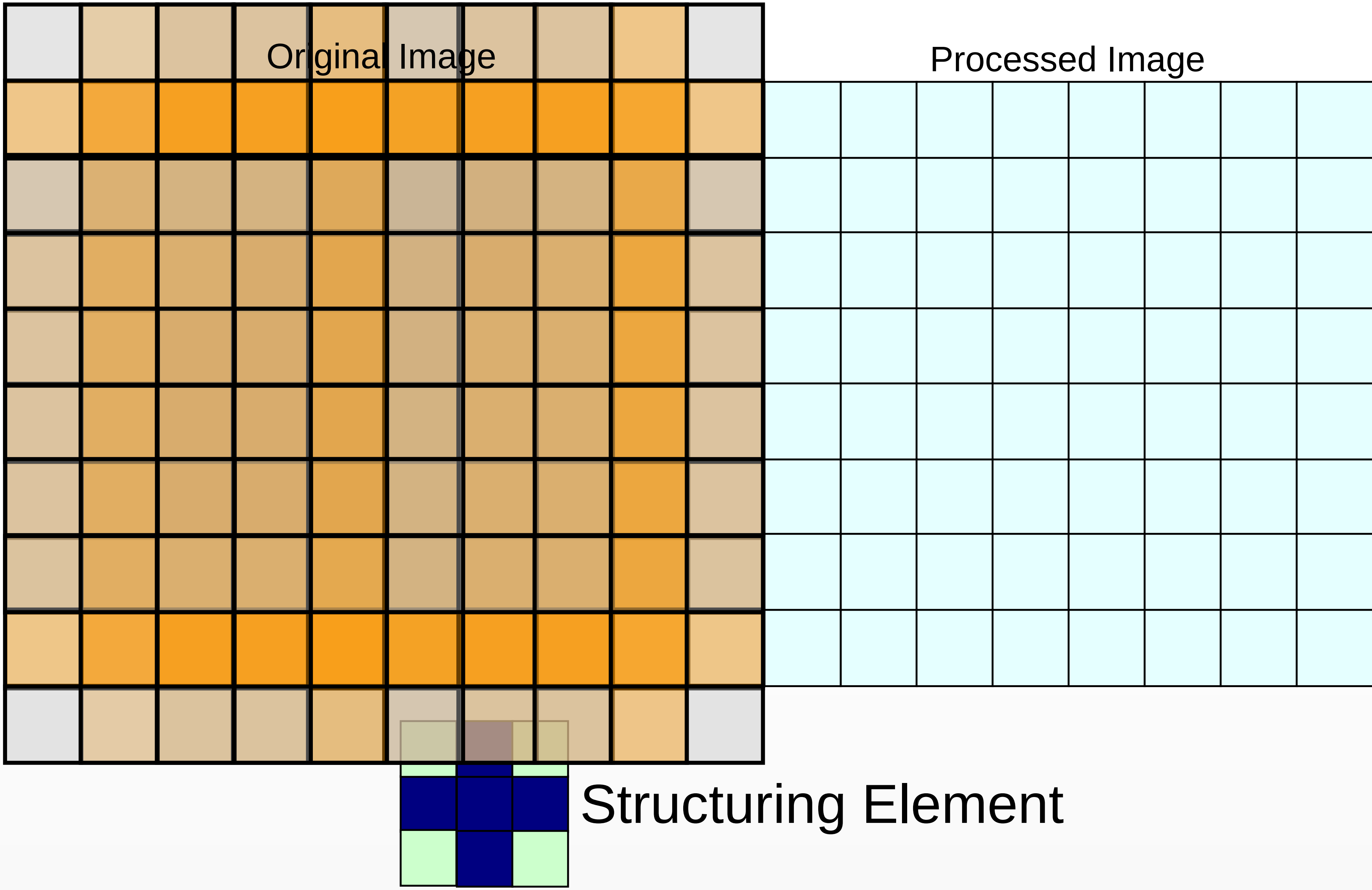
Watch out: Erosion shrinks objects

Dilation

- **Dilation** of image f by structuring element s is given by $f \oplus s$
- The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

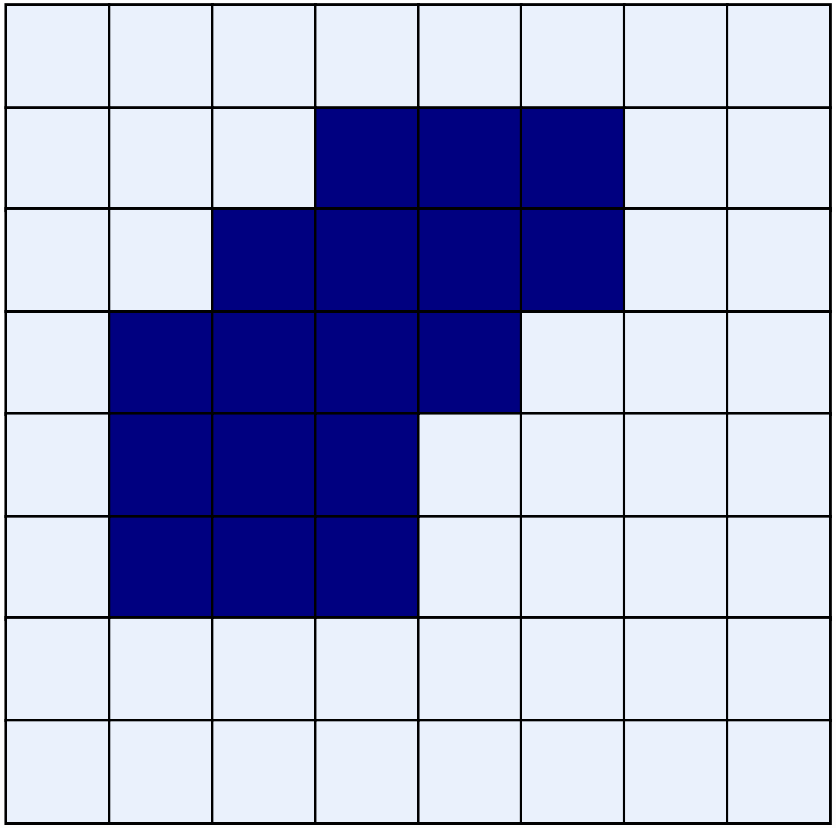
$$g(x,y) = \begin{cases} 1 \text{ if } s \text{ hits } f \\ 0 \text{ otherwise} \end{cases}$$

Dilation Example

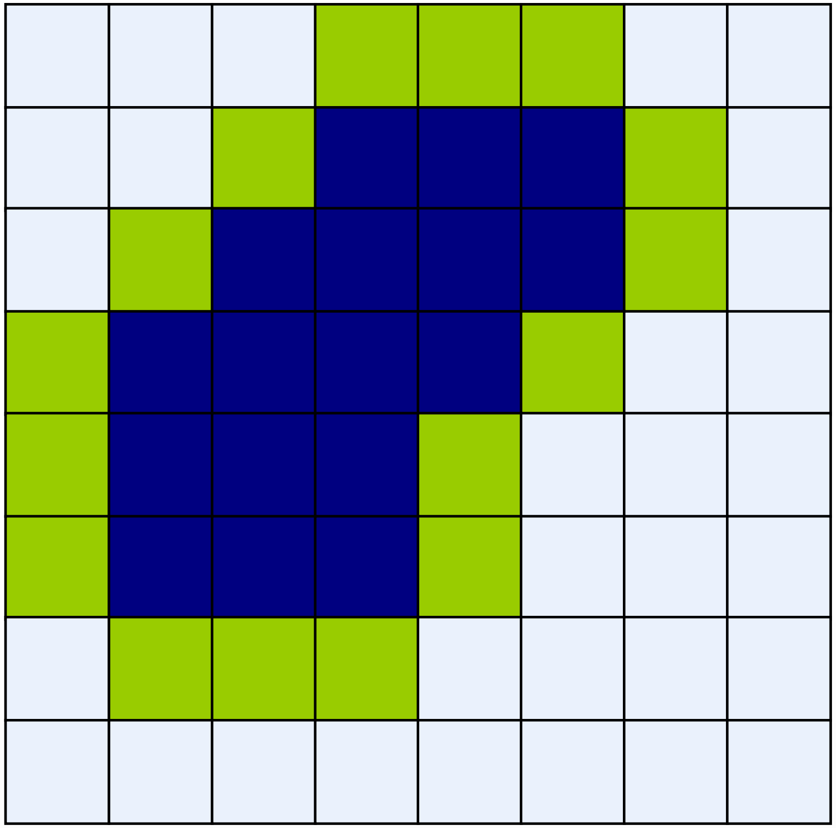


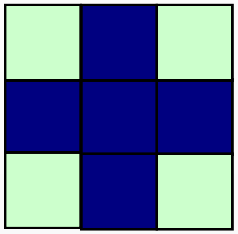
Dilation Example

Original Image



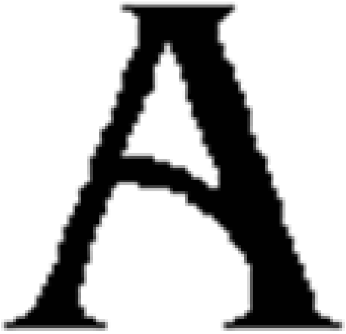
Processed Image With Dilated Pixels



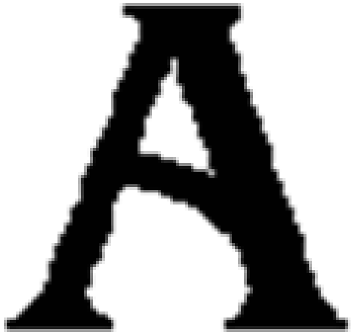


Structuring Element

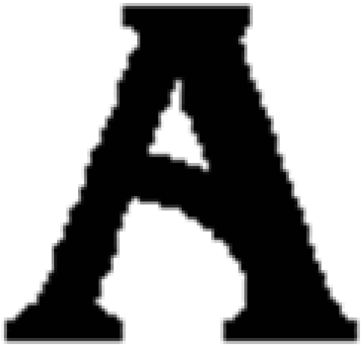
Dilation Example 1



Original image



Dilation by 3*3 square structuring element

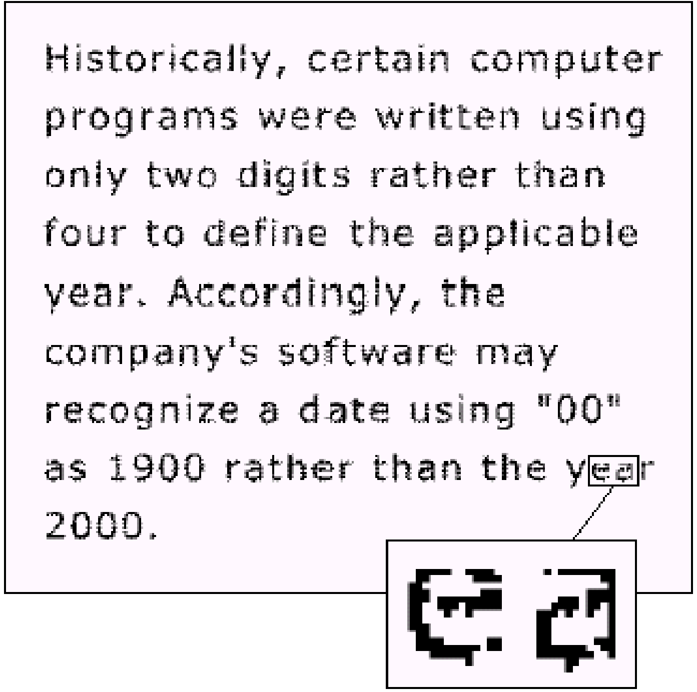


Dilation by 5*5 square structuring element

Watch out: In these examples a 1 refers to a black pixel!

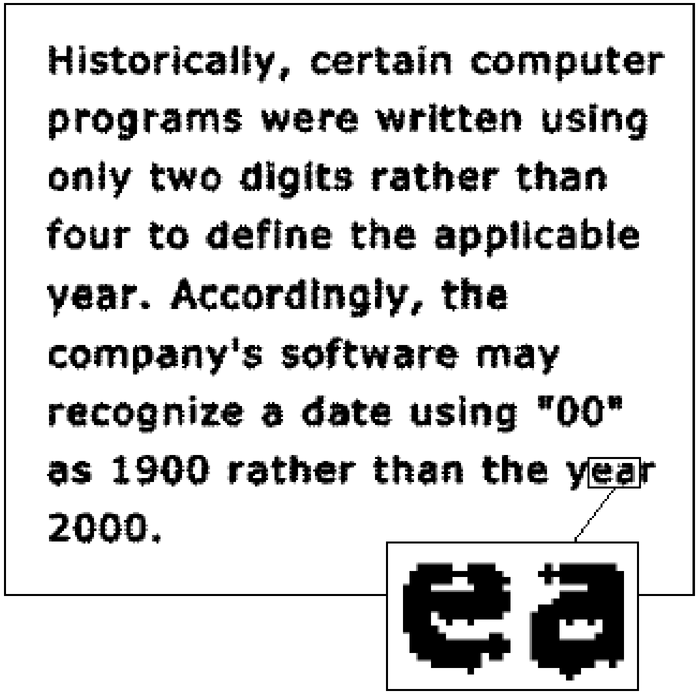
Dilation Example 2

Original image



Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

After dilation



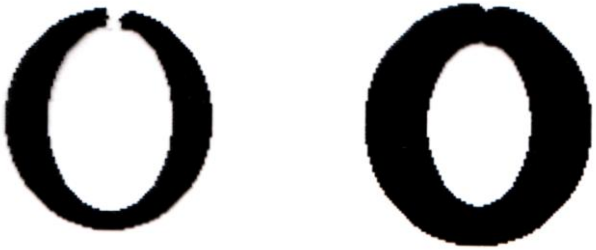
Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.

Structuring element

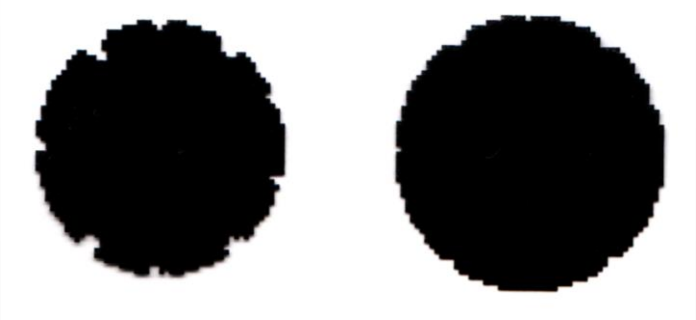
0	1	0
1	1	1
0	1	0

What Is Dilation For?

Dilation can repair breaks



Dilation can repair intrusions



Watch out: Dilation enlarges objects

Compound Operations

More interesting morphological operations can be performed by performing combinations of erosions and dilations

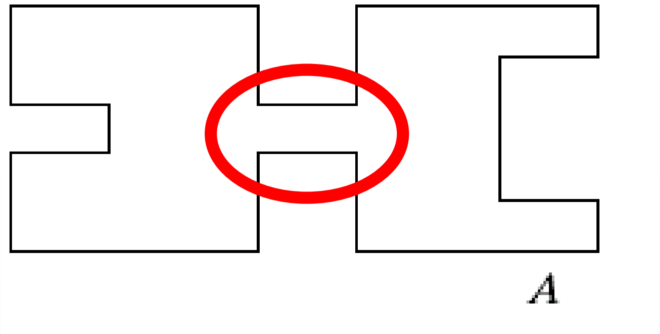
The most widely used of these *compound operations* are:

- Opening
- Closing

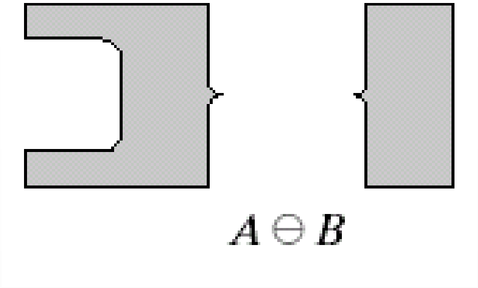
Opening

- The opening of image f by structuring element s, denoted $f \circ s$ is simply an erosion followed by a dilation

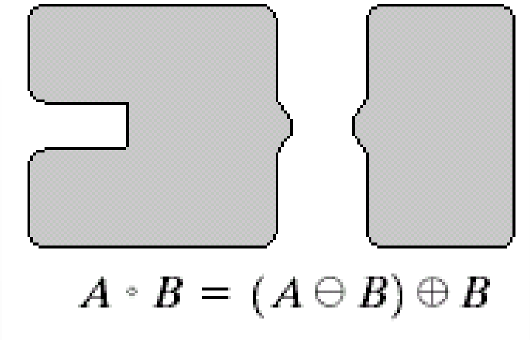
$$f \circ s = (f \ominus s) \oplus s$$



Original shape



After erosion



After dilation (opening)

Note: a disc shaped structuring element is used

Opening Example

Original Image

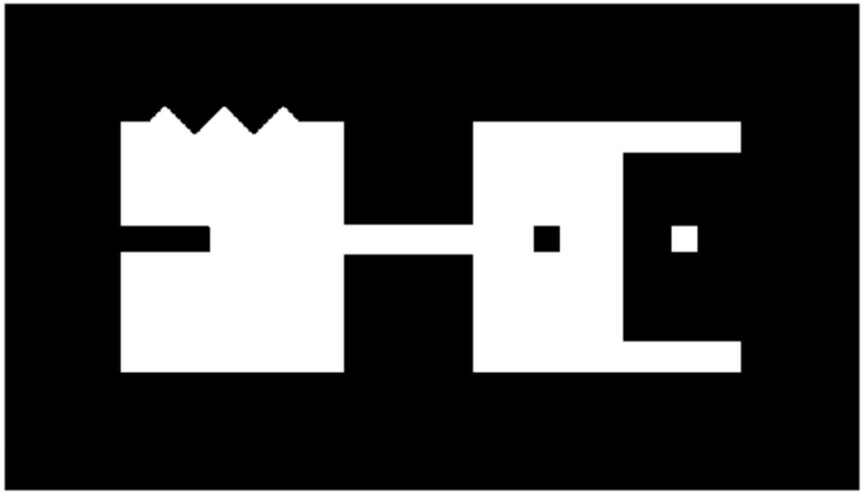
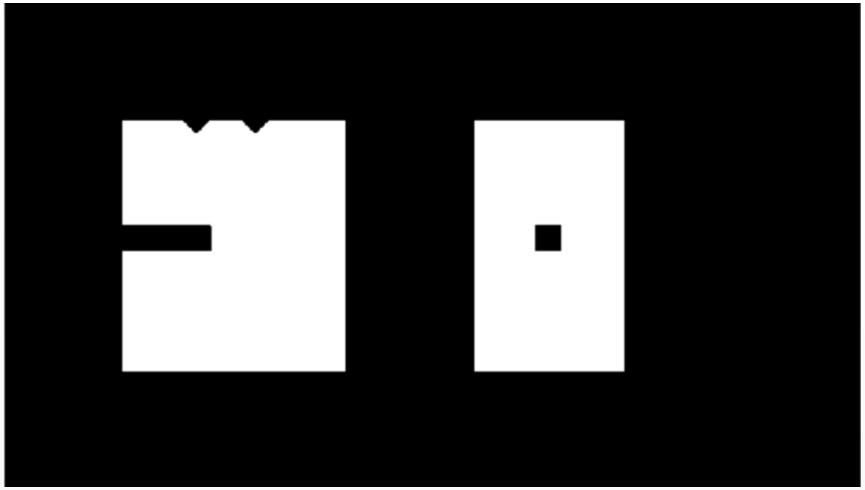


Image After Opening



Opening Example

Original Image

□	□	□	□	■	■	■	□
□	□	□	□	■	■	■	□
□	□	□	□	■	■	■	□
□	□	□	■	■	□	□	□
□	■	■	■	□	□	□	□
□	■	■	■	□	□	□	□
□	■	■	■	□	□	□	□
□	□	□	□	□	□	□	□

Processed Image

□	□	□	□	□	■	□	□
□	□	□	□	■	■	■	□
□	□	□	□	□	■	□	□
□	□	□	□	□	□	□	□
□	□	■	□	□	□	□	□
□	■	■	■	□	□	□	□
□	□	■	□	□	□	□	□
□	□	□	□	□	□	□	□

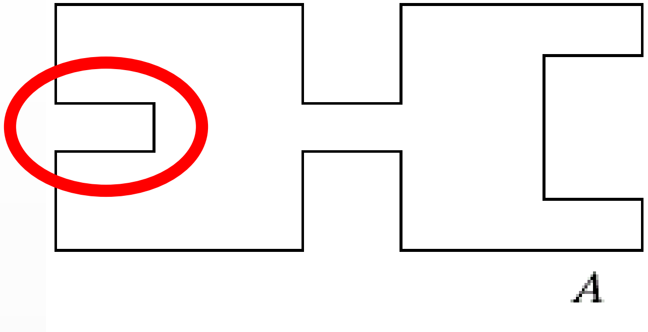
□	■	□
■	■	■
□	■	□

Structuring Element

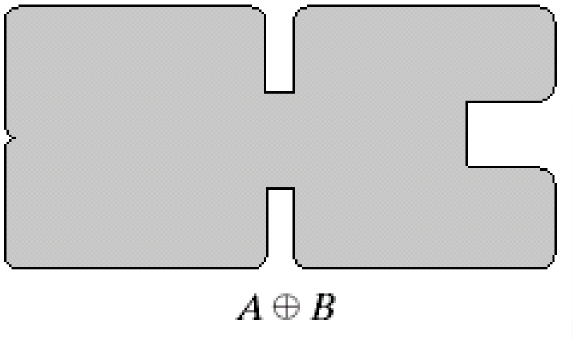
Closing

- The closing of image f by structuring element s, denoted $f \bullet s$ is simply a dilation followed by an erosion

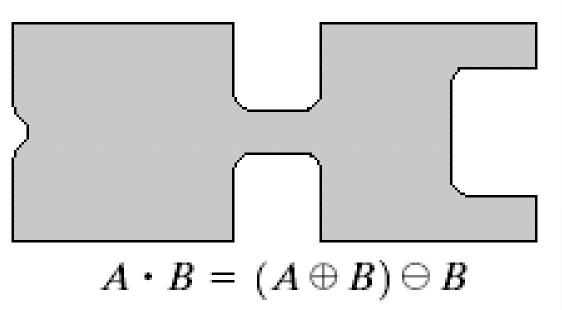
$$f \bullet s = (f \oplus s) \ominus s$$



Original shape



After dilation



After erosion (closing)

Note: a disc shaped structuring element is used

Closing Example

Original Image

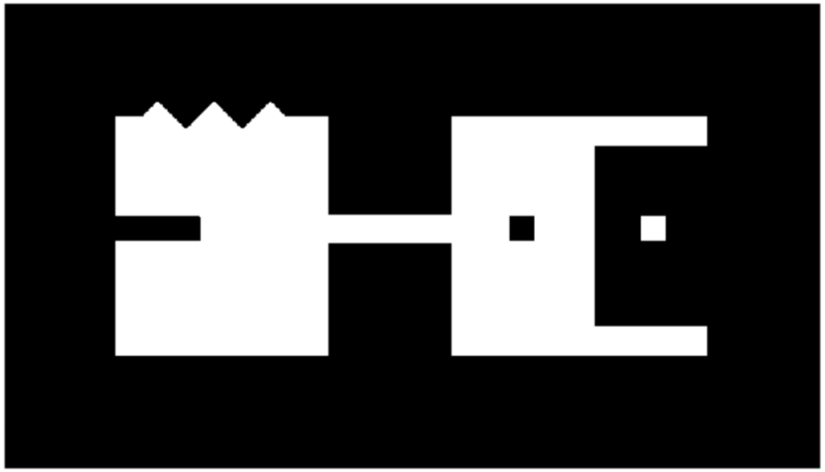
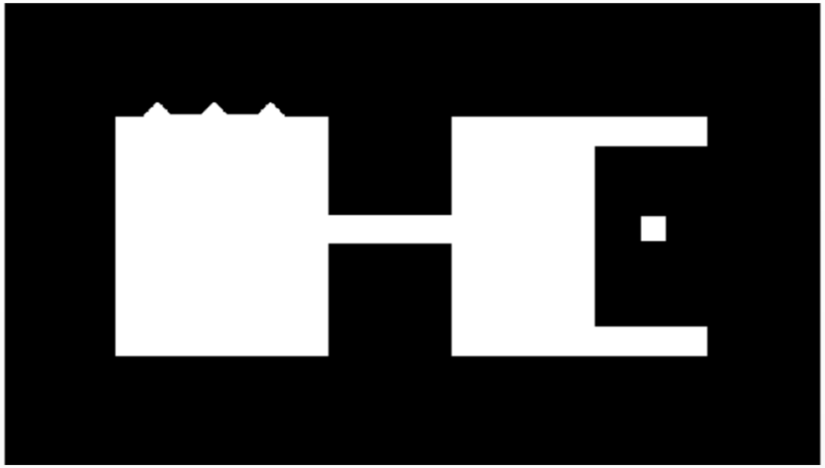
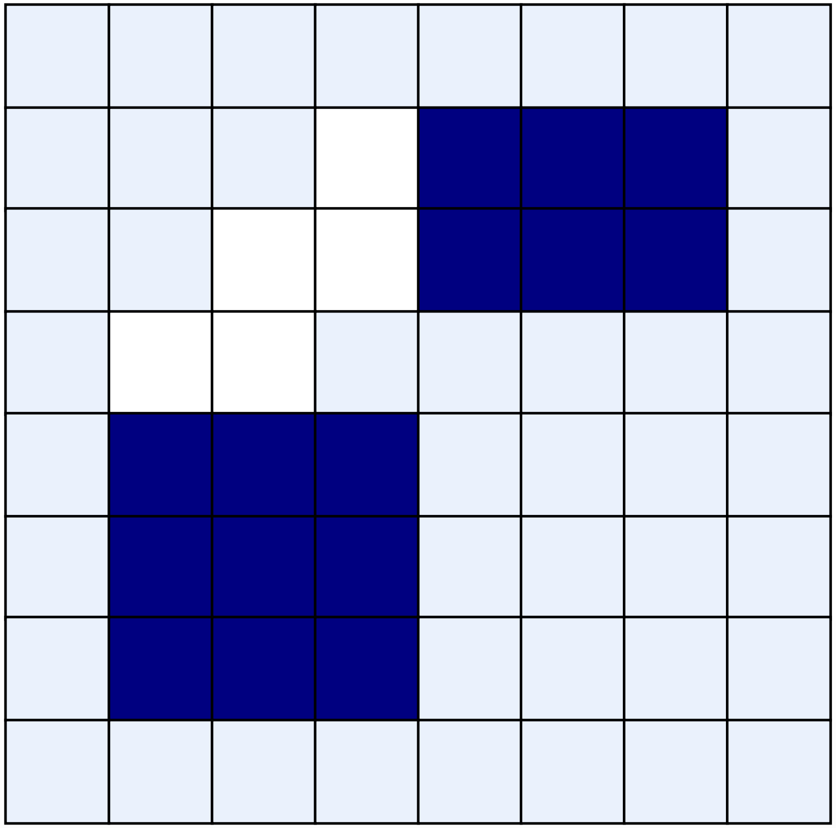


Image After Closing

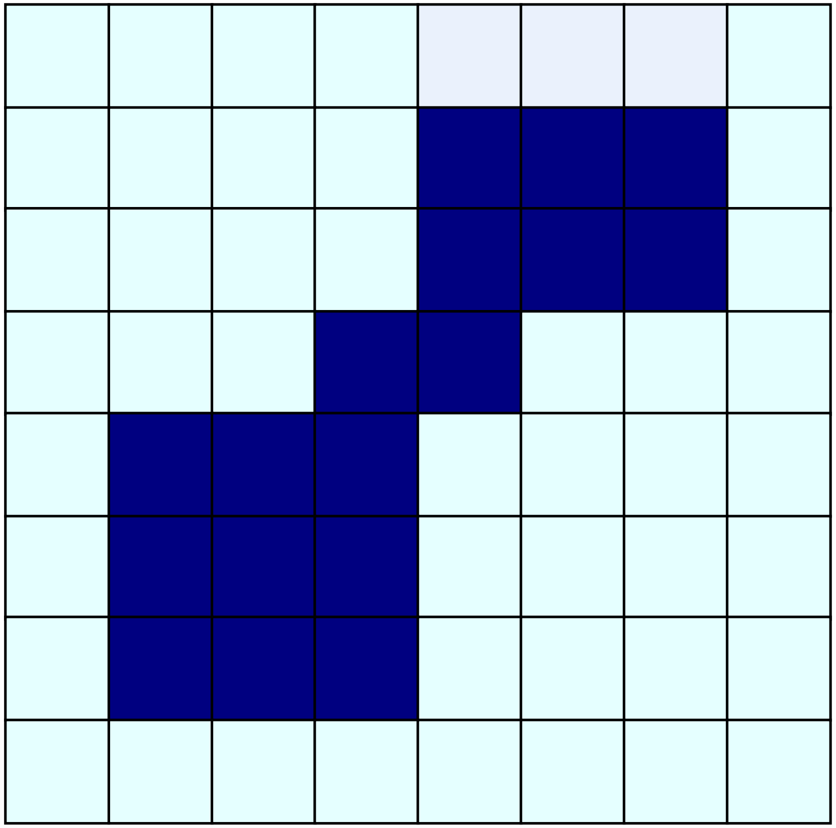


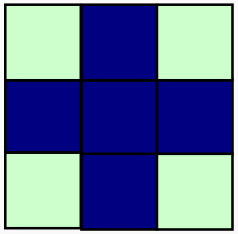
Closing Example

Original Image



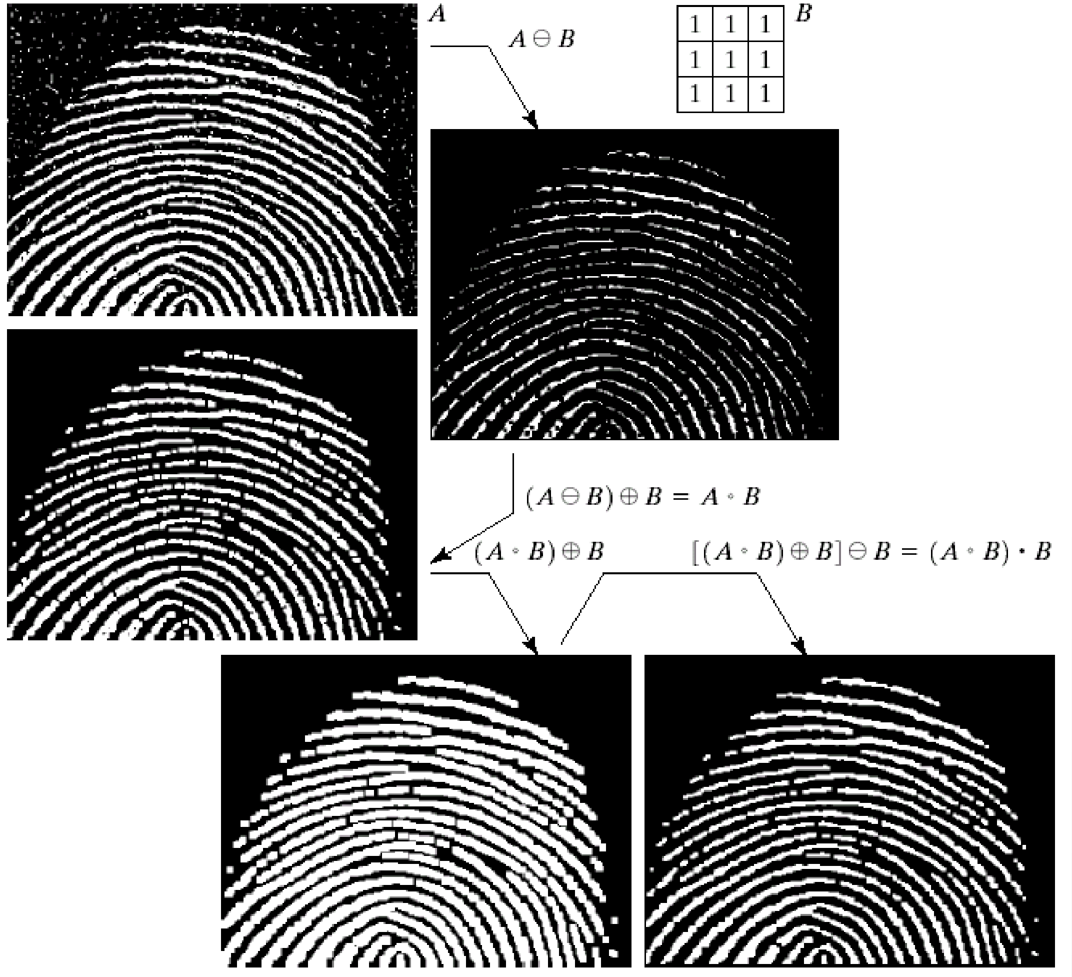
Processed Image



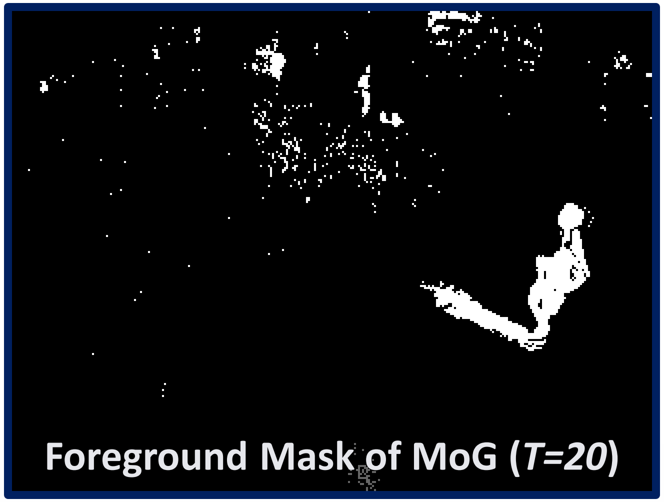


Structuring Element

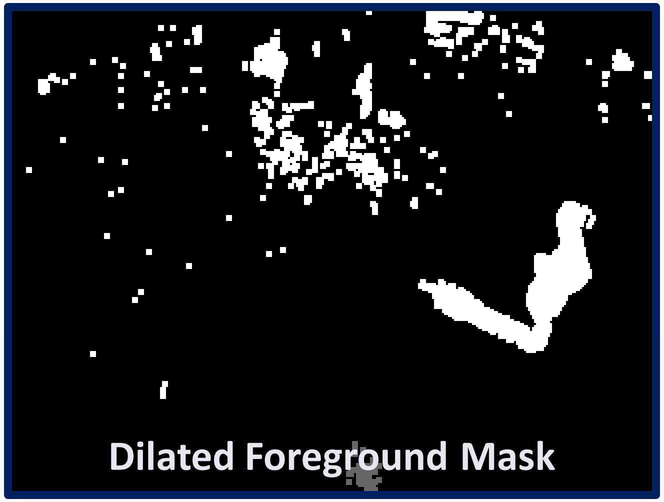
Morphological Processing Example


A
$A \ominus B$
1 1 1
1 1 1
1 1 1
B
$(A \ominus B) \oplus B = A \circ B$
$(A \circ B) \oplus B$
$[(A \circ B) \oplus B] \ominus B = (A \circ B) \bullet B$

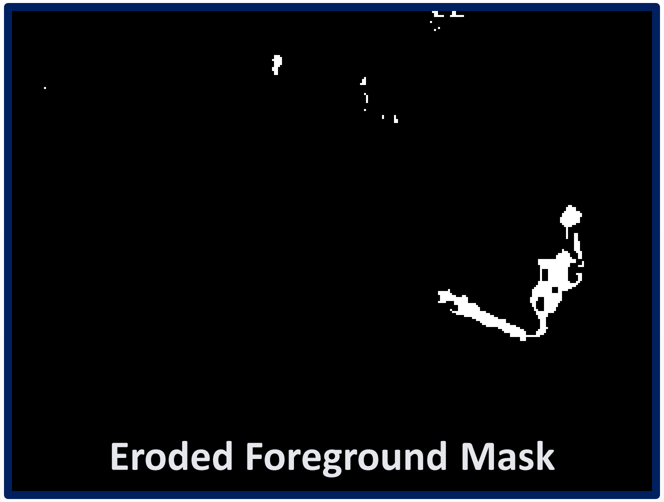
Morphological Operations



Foreground Mask of MoG (*T=20*)

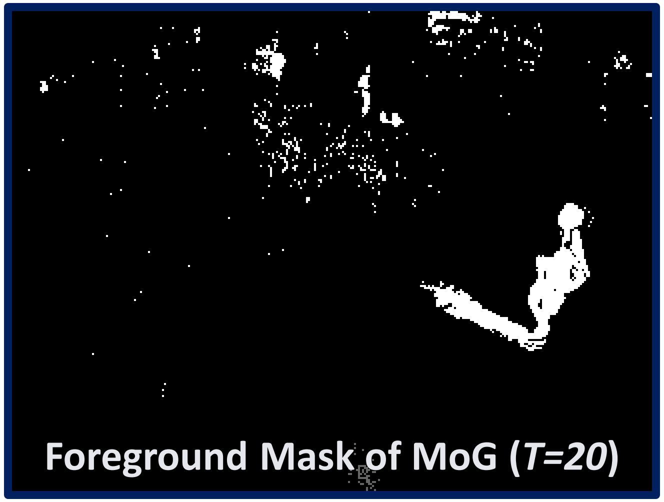


Dilated Foreground Mask

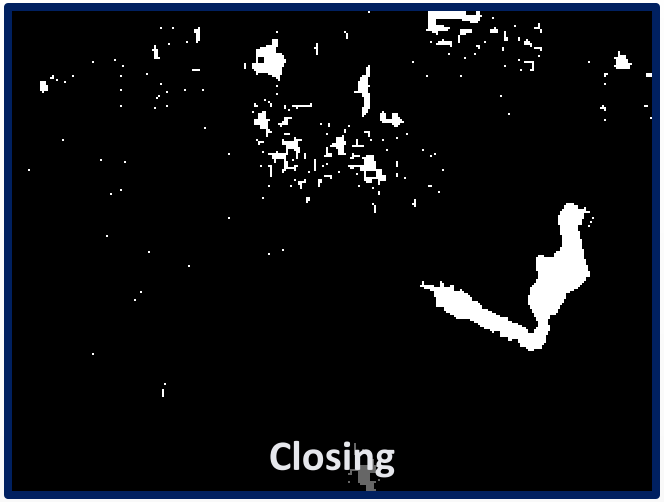


Eroded Foreground Mask

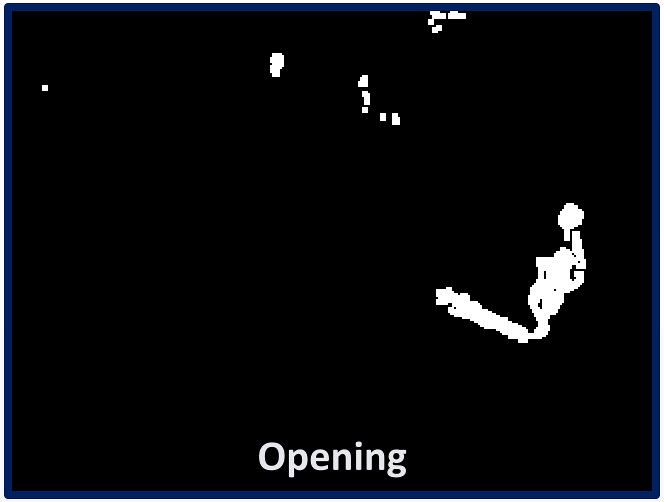
Morphological Operations



Foreground Mask of MoG (*T=20*)

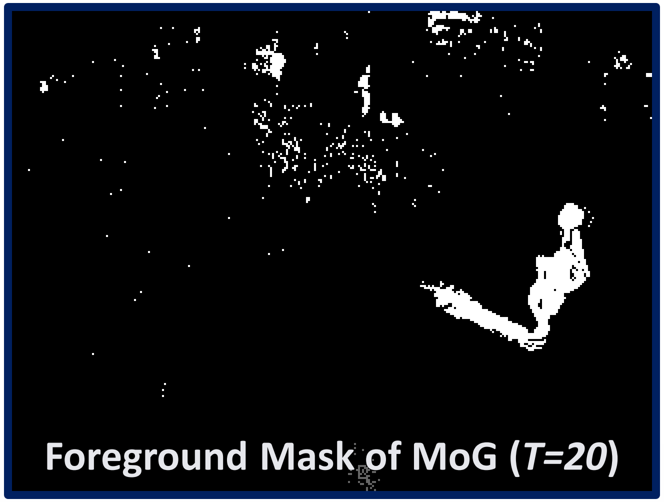


Closing

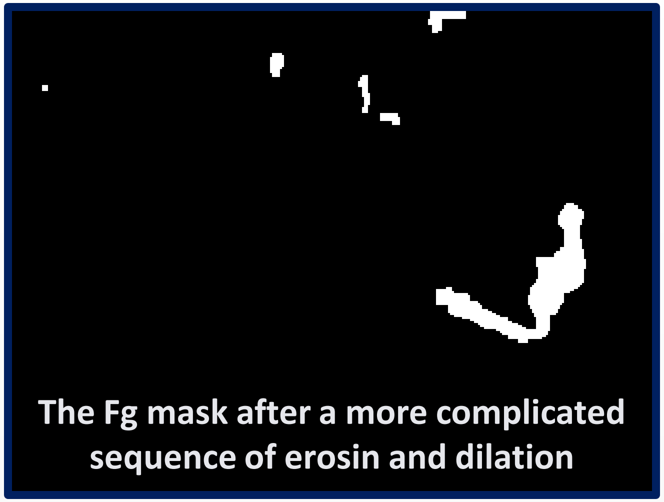


Opening

Morphological Operations



Foreground Mask of MoG (*T=20*)



The Fg mask after a more complicated sequence of erosin and dilation

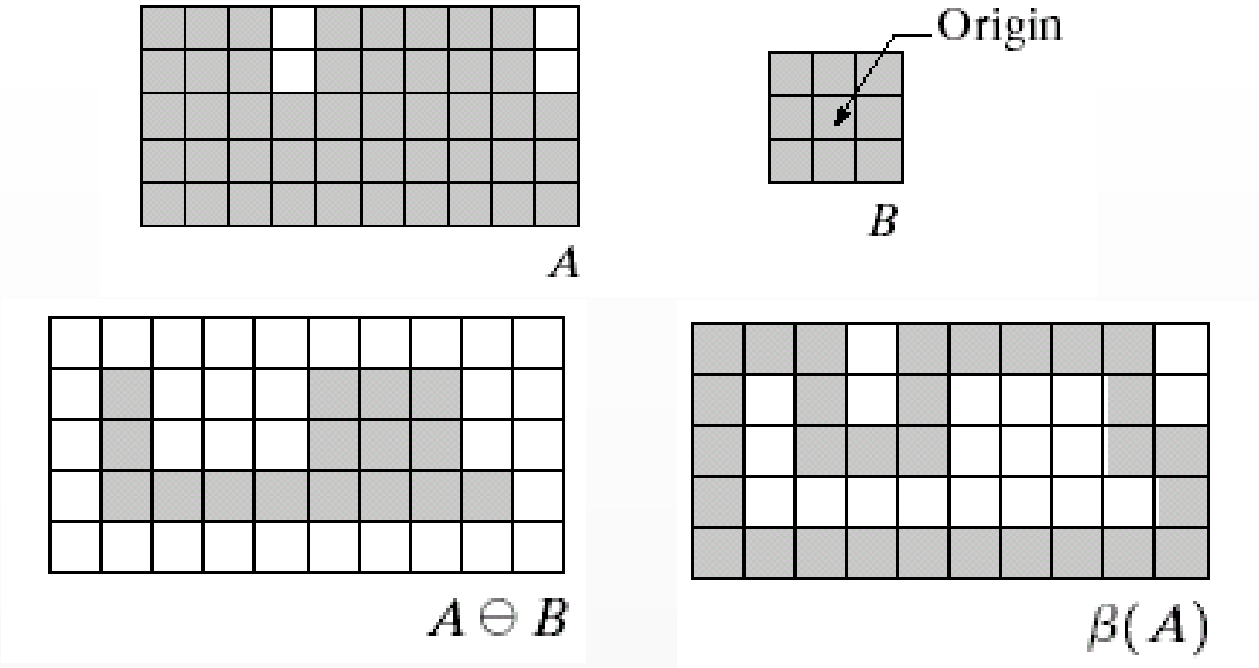
Morphological Algorithms

- Using the simple technique we have looked at so far we can begin to consider some more interesting morphological algorithms
- We will look at:
  - Boundary extraction
  - Region filling
- There are lots of others as well though:
  - Extraction of connected components
  - Thinning/thickening
  - Skeletonisation

Boundary Extraction

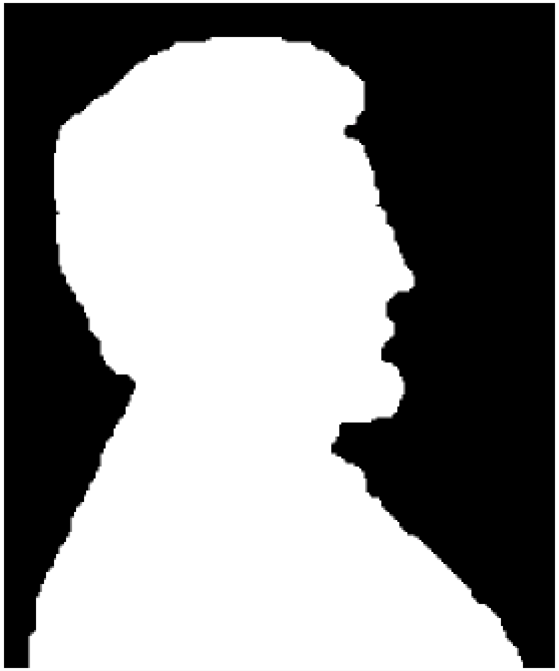
- Extracting the boundary (or outline) of an object is often extremely useful
- The boundary can be given simply as

$$\beta(A) = A - (A \ominus B)$$

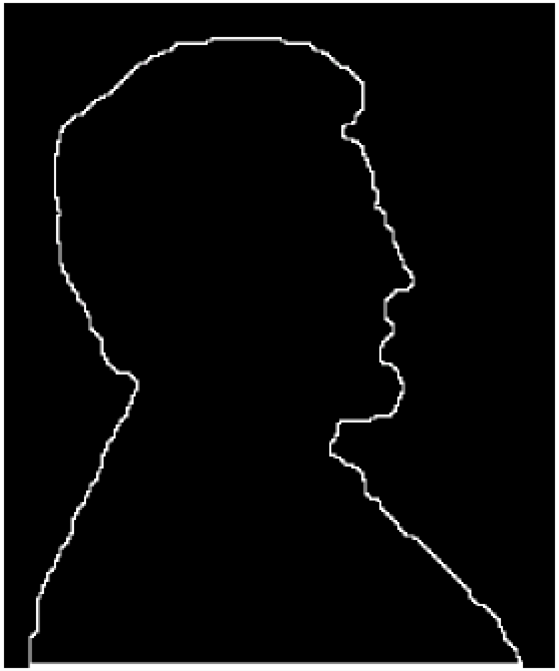


Boundary Extraction Example

- A simple image and the result of performing boundary extraction using a square 3×3 structuring element



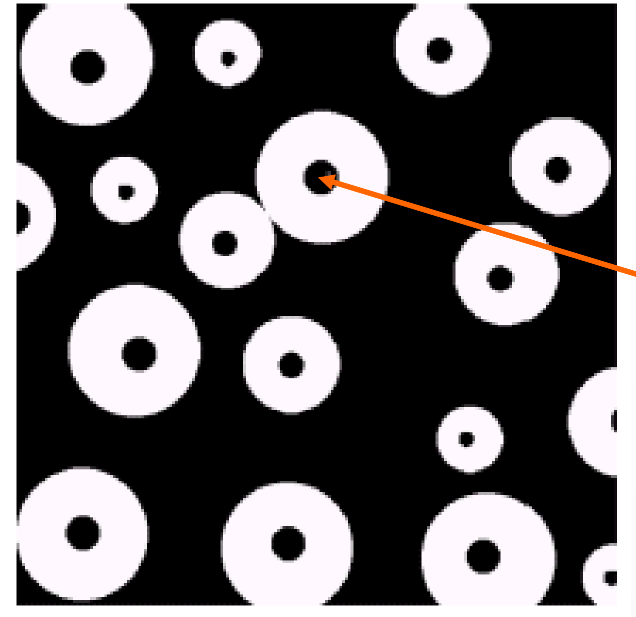
Original Image



Extracted Boundary

Region Filling

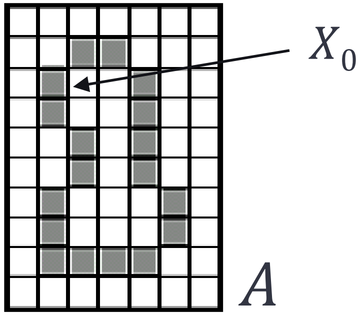
- Given a pixel inside a boundary, *region filling* attempts to fill that boundary with object pixels (1s)

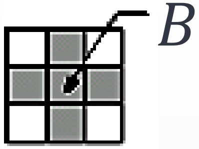


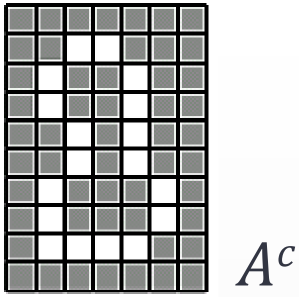
Given a point inside here, can we fill the whole circle?

Region Filling

- The key equation for region filling is:
  $X_k = (X_{k-1} \oplus B) \cap A^c$, where
  - A is the original (boundary) image,
  - X_0 is simply the starting point (single pixel) inside the boundary,
  - B is a simple structuring element and
  - A^c is the complement of A

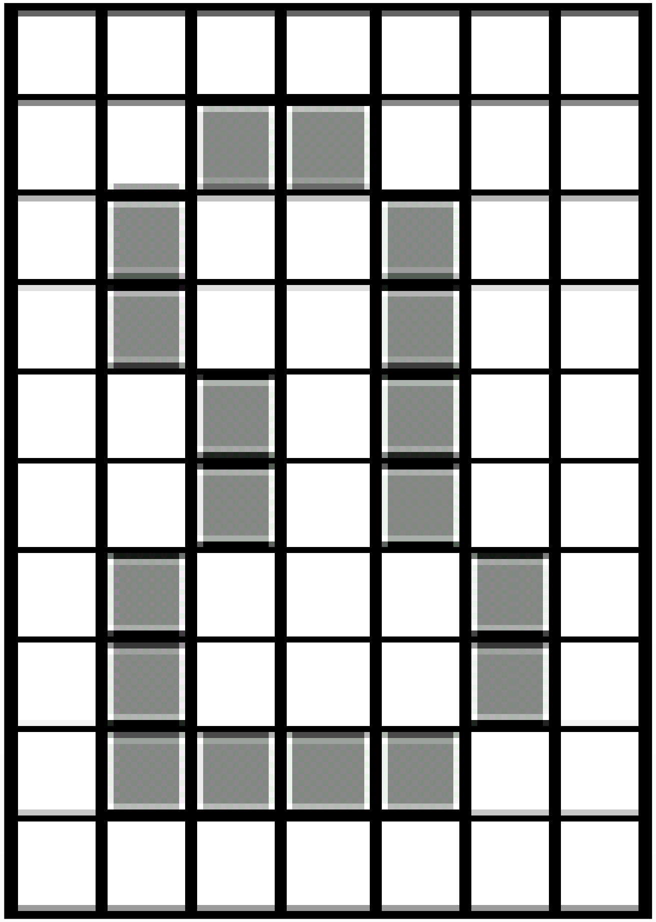


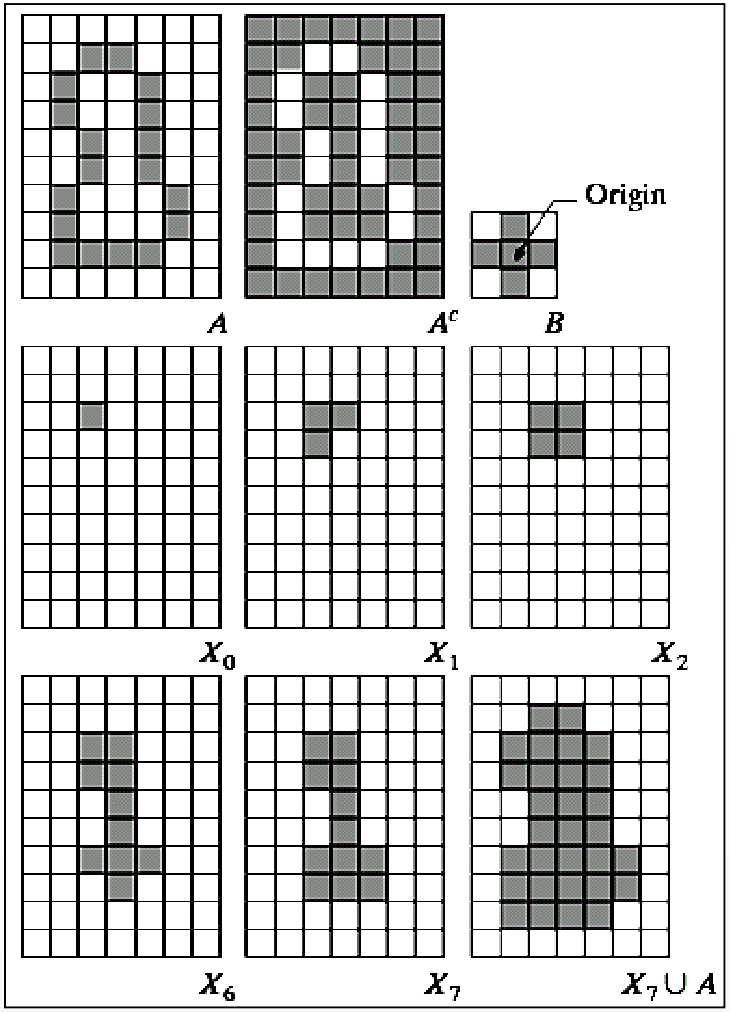




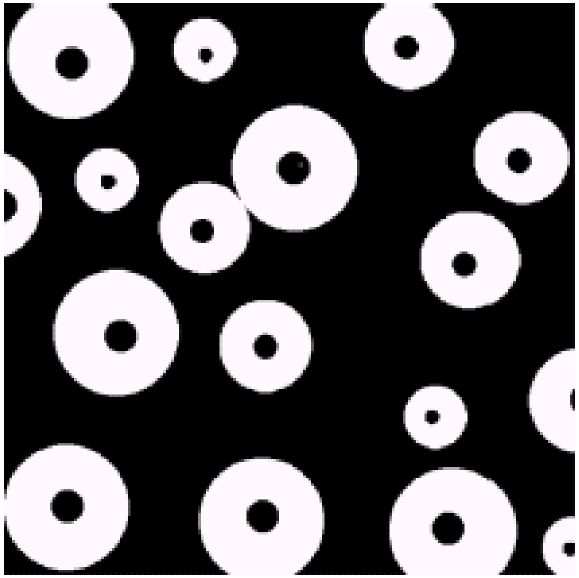
- This equation is applied repeatedly until X_k is equal to X_{k-1}
- Finally the result is unioned with the original boundary

Region Filling Step By Step

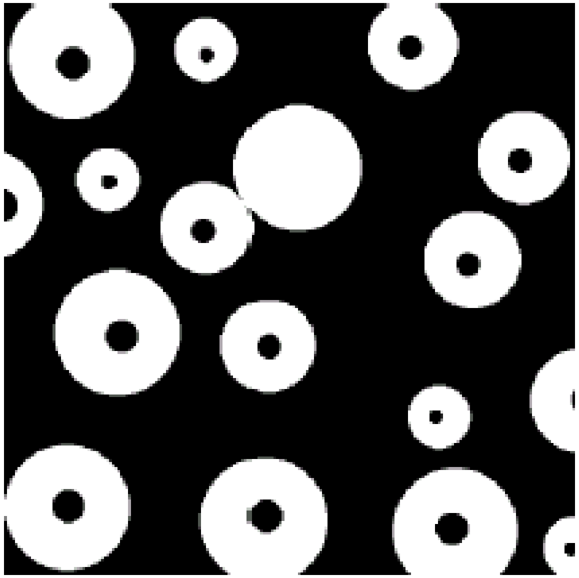




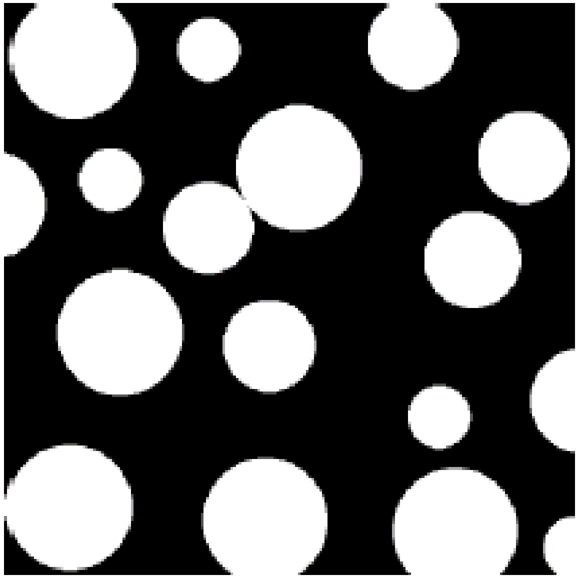
Region Filling Example



Original Image



One Region Filled



All Regions Filled

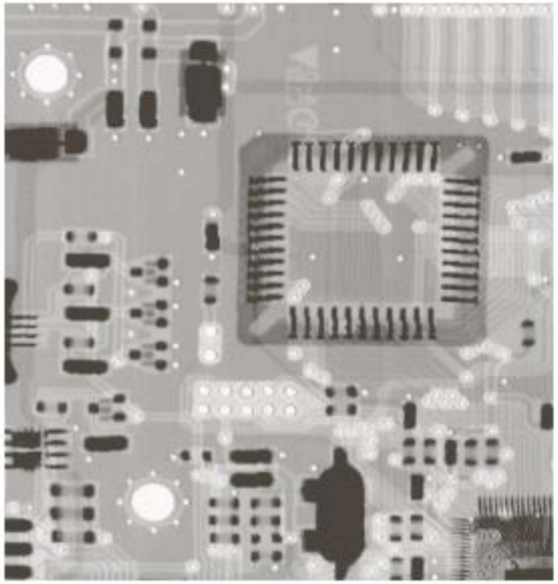
Grayscale morphology

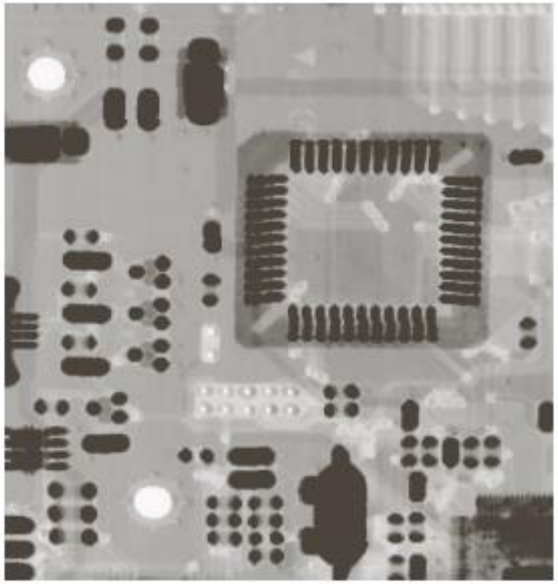
- Gray-Scale Morphology: Erosion and Dilation by Flat Structuring

$$\left[f - b \right](x, y) = \min_{(s,t)\in b} \left\{ f(x+s, y+t) \right\}$$

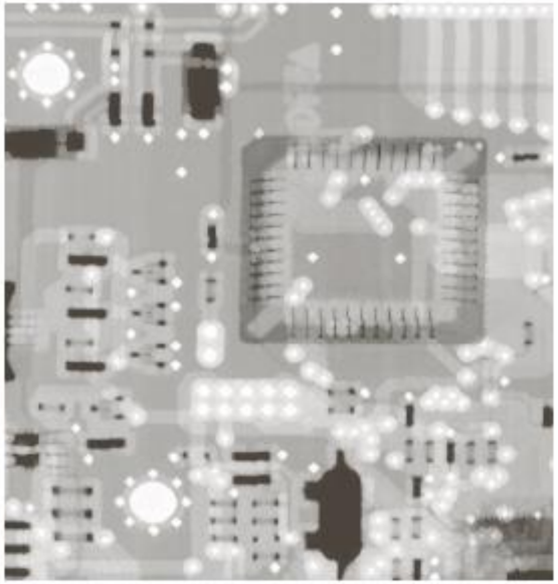
$$\left[f \oplus b \right](x, y) = \max_{(s,t)\in b} \left\{ f(x-s, y-t) \right\}$$

Grayscale morphology





Erosion



Dilation

Summary-morphology

- The purpose of morphological processing is primarily to remove imperfections added during segmentation
- The basic operations are *erosion* and *dilation*
- Using the basic operations we can perform *opening* and *closing*
- More advanced morphological operation can then be implemented using combinations of all of these

Basic Image Processing Algorithms

PPKE-ITK

Lecture 8.

Image and Video Segmentation

Previously on... Basic Image Processing

- Previous topics:
  - Color Spaces, dithering
  - 2D convolution, Canny edge detector
  - Hough transformation & Image Enhancement
  - Fourier analysis
  - Texture analysis
  - Image recovery
  - Segmentation: Otsu, K-means and Morphology

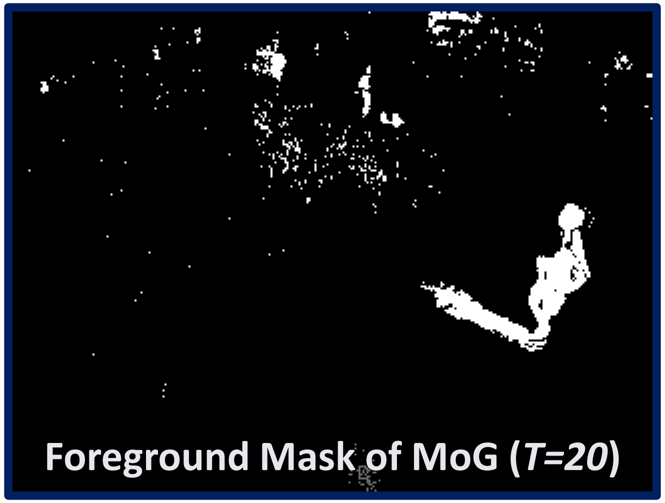
Classical era mainly from ’60s-’80s (with exceptions)

- Remaining topics:
  - Markov Random Fields, Marked Point Processes
  - Mean shift
  - Descriptors: SIFT, HOG, Local Binary Patterns
  - Video processing
  - Machine Learning
  - Deep Learning

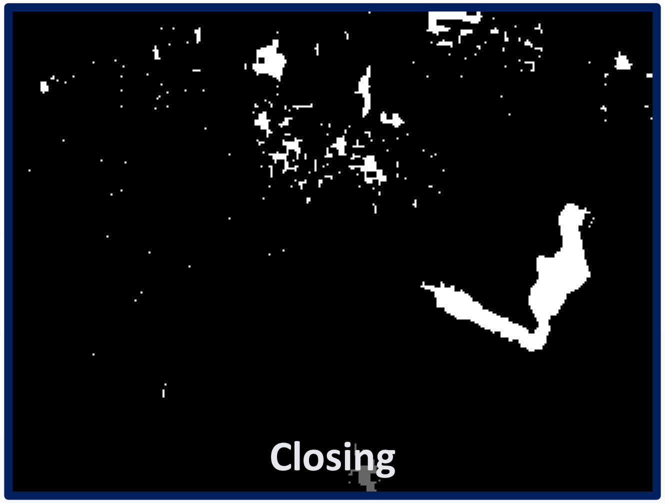
Modern era mainy from ’00s-’10s (with exceptions)

Recap: Morphological Operations

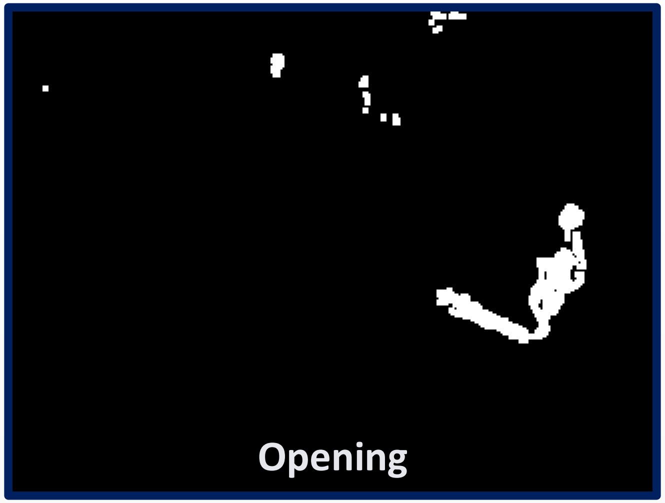
Limitations: distortion of object shapes



Foreground Mask of MoG (*T=20*)



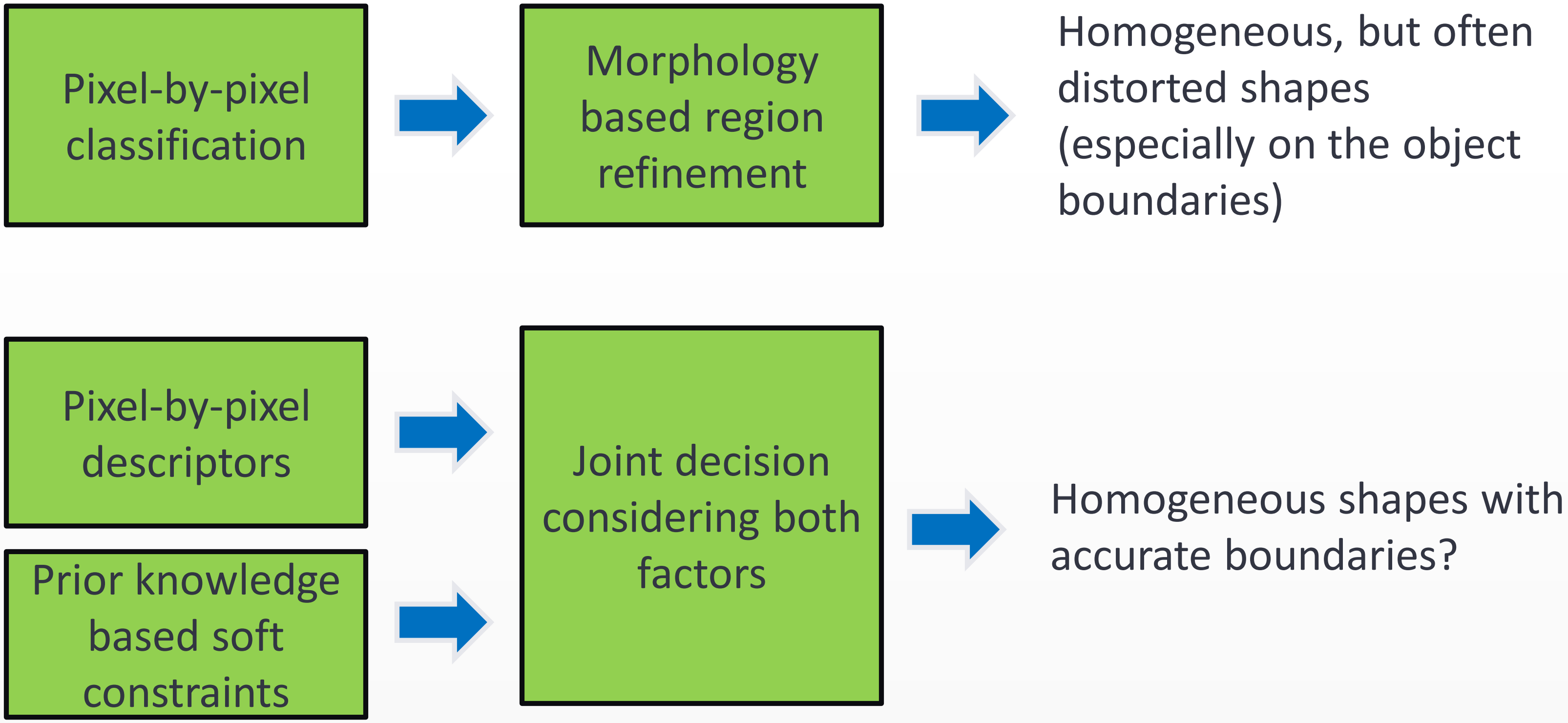
Closing



Opening

Beyond morphology based approaches?

- Pixel-by-pixel classification: observation (image) based knowledge, e.g. pixel color values, local texture features etc.
- Morphology to obtain homogeneous regions: prior knowledge



Pixel-by-pixel classification

Morphology based region refinement

Homogeneous, but often distorted shapes (especially on the object boundaries)

Pixel-by-pixel descriptors

Prior knowledge based soft constraints

Joint decision considering both factors

Homogeneous shapes with accurate boundaries?

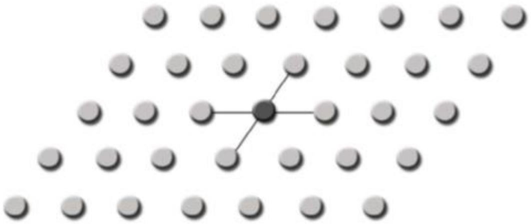
Markov Random Fields in Image Segmentation

- Segmentation as pixel labeling
- Probabilistic approach
  - Segmentation as MAP estimation
  - Markov Random Field (MRF)
  - Gibbs distribution & Energy function
- Classical energy minimization
  - Simulated Annealing
  - Markov Chain Monte Carlo (MCMC) sampling
- Example MRF model & Demo
- Parameter estimation (EM)

MRF slides adopted © Zoltan Kato, University of Szeged, http://www.inf.u-szeged.hu/~kato/

Markov Random Fields in Image Segmentation
main principle

- Mapping the image to a graph
  - nodes are assigned to the different pixels, and the edges connect pixels which are in interaction
- Segmentation as pixel labeling:
  - each pixel gets a class-label from a task-dependent label set Λ



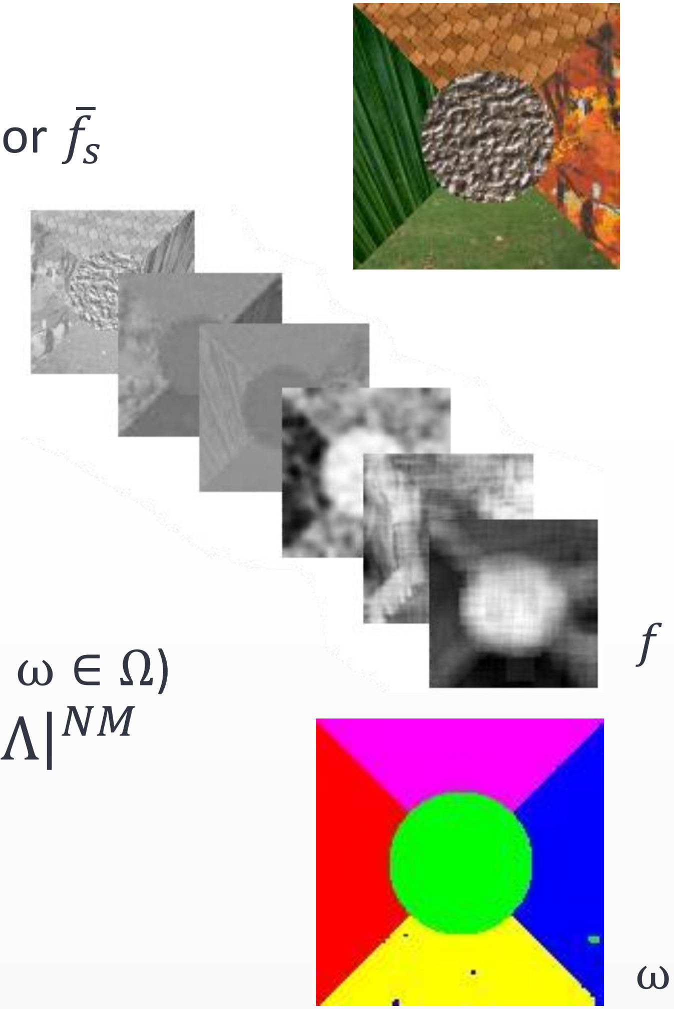
- Inverse problem formulation:
  - Instead of finding a direct algorithm to find the optimal labeling, we construct a (pseudo-) probability function which assigns a likelihood value to each possible global segmentation, then an optimization process attempts to find the labeling with the highest confidence
- What does the probability function depend on?
  - local feature vectors at each pixel (color, texture etc)
    - classes in Λ are as stochastic processes, described by different feature distributions
  - label consistency (soft) constraints between neighboring pixels
    - e.g. for preferring smooth segmentation map we penalize if two neighboring nodes have different labels

Segmentation as a Pixel Labelling Task

- Extract features from the input image
  - Each pixel s in the image has a feature vector $\bar{f}_s$
  - For the whole image, we have:

    $f = \{\bar{f}_s : s \in S\}$: global observation
- Define the set of labels Λ
  - Each pixel s is assigned a label $\omega_s \in \Lambda$
  - For the whole image, we have:

    $\omega = \{\omega_s : s \in S\}$: global labeling
  - Ω: set of all possible ω global labelings (i.e. $\omega \in \Omega$)
- For an $N \times M$ image, there are $|\Omega| = |\Lambda|^{NM}$ possible global labelings.
  - **Which one is the right segmentation?**



Source: Zoltan Kato, http://www.inf.u-szeged.hu/~kato/

Probabilistic Approach, MAP

- Define a ***probability measure*** on the set of all possible global labeling and select the most likely one.
- $P(\omega|f)$ measures the probability of a global labeling ω, given the observed features f
- Our goal is to find an optimal labeling $\widehat{\omega}$ which ***maximizes*** $P(\omega|f)$
- This is called the ***Maximum a Posteriori*** **(MAP)** estimate:

$$\widehat{\omega} = \underset{\omega \in \Omega}{\operatorname{argmax}} P(\omega|f)$$

Bayesian Framework

- By Bayes Theorem, we have

$$P(\omega|f) = \frac{P(f|\omega)P(\omega)}{P(f)} \propto \overbrace{P(f|\omega)}^{\text{likelihood}}\overbrace{P(\omega)}^{\text{prior}}$$

- $P(f)$ is constant
  - it does not depend on the actual labeling!
- We need to define $P(f|\omega)$ and $P(\omega)$ in our model

We will use **Markov Random Fields**

Why MRF Modelization?

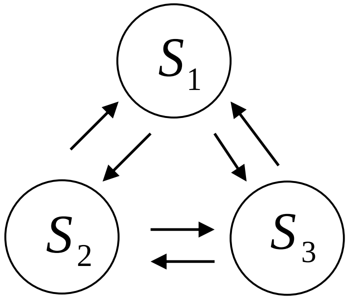
- In real images, regions are often **homogenous**; neighboring pixels usually have similar properties (intensity, color, texture, …) → prior neighborhood constraints vs. noisy pixel level descriptors
- **Markov Random Field (MRF)** is a probabilistic model which captures such contextual constraints
  - Well studied, strong theoretical background
  - Allows Monte-Carlo Markov Chain (MCMC) sampling of the (hidden) underlying structure → **Simulated Annealing**
  - Fast and exact solution for certain type of models → **Graph cut** [Kolmogorov]

What is MRF?

- To give a formal definition for Markov Random Fields, we need some basic building blocks
  - Observation Field and (hidden) Labeling Field
  - Pixels and their Neighbors
  - Cliques and Clique Potentials
  - Energy function
  - Gibbs Distribution

Markov Chains vs Markov Random Fields

- Recap: Discrete Markov Chains: discrete time, discrete state stochastic processes
  - Given: set of possible states $S_1, S_2, \ldots S_N$
  - q_t: state at time t, $(t = 1, \ldots T)$
  - Observed state sequence: $q_1, q_2, \ldots q_T$
  - Markov property:

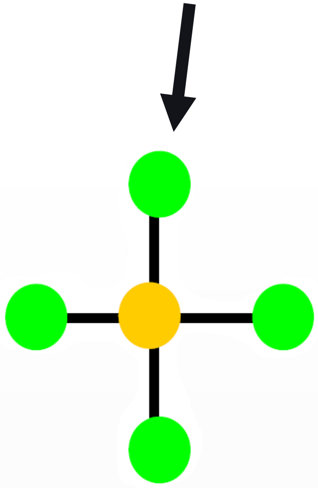


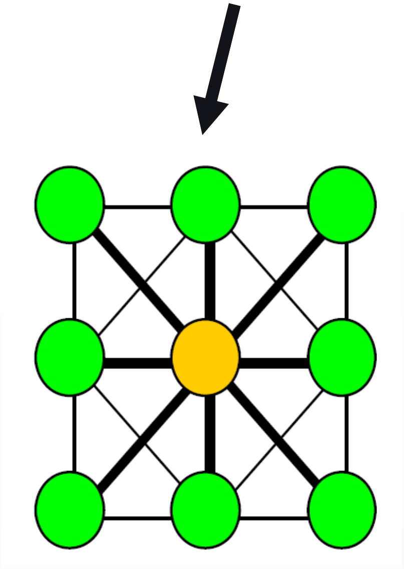
$$P(q_t = S_j | q_{t-1} = S_i) = P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots q_1 = S_l)$$

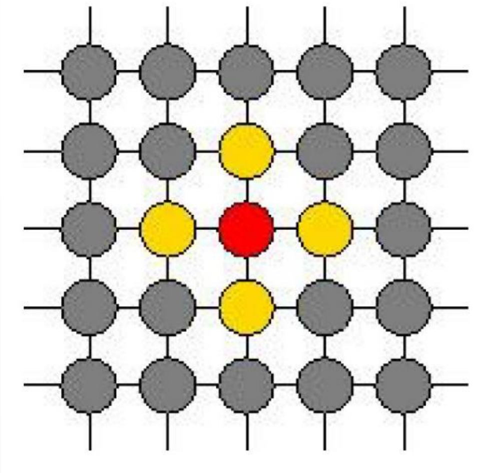
  - Conditional probability of the current state only depends on the previous state (i.e. only neighboring states interact – in time)
- Markov Random Fields: instead of temporal neighboring states, we consider the spatially neighboring pixels
  - Pixel labels are not independent, however, direct dependence is only considered between the spatial neighbors

Definition – Neighbors

- For each pixel, we can define some surrounding pixels as its neighbors.
- Example: 1st order neighbors and 2nd order neighbors





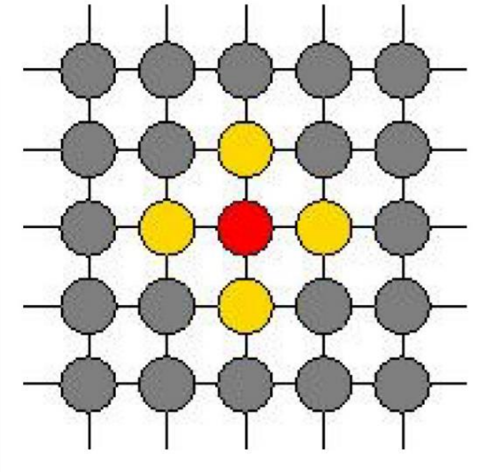


Definition – MRF

- The labeling field X can be modeled as a Markov Random Field (MRF) if
  1. For all $\omega \in \Omega$: $\mathrm{P}(X = \omega) > 0$
  2. For every $s \in S$ and $\omega \in \Omega$:

$$\mathrm{P}(\omega_s | \omega_r, r \neq s) = \mathrm{P}(\omega_s | \omega_r, r \in N_s)$$

  - N_s denotes the neighbors of pixel s



Hammersley-Clifford Theorem

- The **Hammersley-Clifford Theorem** states that a random field is a MRF if and only if $P(\omega)$ follows a **Gibbs distribution**.

$$P(\omega) = \frac{1}{Z}\exp\big(-U(\omega)\big) = \frac{1}{Z}\exp\left(-\sum_{c\in C} V_c\,(\omega)\right)$$

  - where $\mathrm{Z} = \sum_{\omega\in\Omega}\exp\big(-U(\omega)\big)$ is a normalization constant

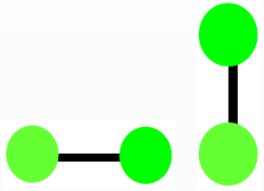
- *Practical consequence:*
  - probability functions of MRFs have a special form: they can be factorized into small terms $V_c(\omega)$ called ***clique potentials***, which can be locally calculated on the graph
  - this property makes possible to design the $P(\omega)$ ***probability function*** in a modular way, and enables using efficient iterative optimization techniques
  - Technical note: instead of maximizing this probability function we usually minimize the minus logarithm of it, $U(\omega)$, which is called the ***energy function***

Definition – Clique

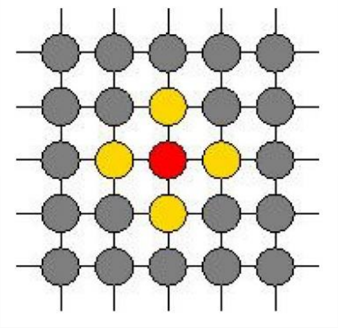
- The H-C theorem provides us an easy way of defining MRF models via ***clique potentials***.
- A subset $\mathrm{C} \subseteq S$is called a ***clique*** if every pair of pixels in this subset are neighbors.
- A clique containing *n* pixels is called ***nth order clique***, denoted by C_n
- The set of cliques in an image is denoted by

$$\mathrm{C} = C_1 \cup C_2 \cup \ldots \cup C_K$$

singleton



doubleton

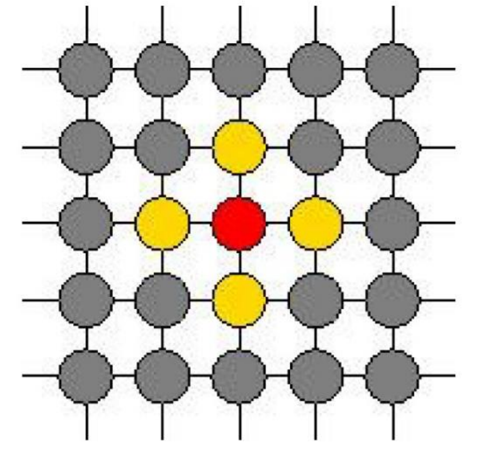


Definition – Clique Potential

- For each clique *c* in the image, we can assign a value $V_c(\omega)$ which is called ***clique potential*** of *c*, where ω is the configuration of the labeling field
- The sum of potentials of all cliques gives us the energy $U(\omega)$ of the configuration ω.

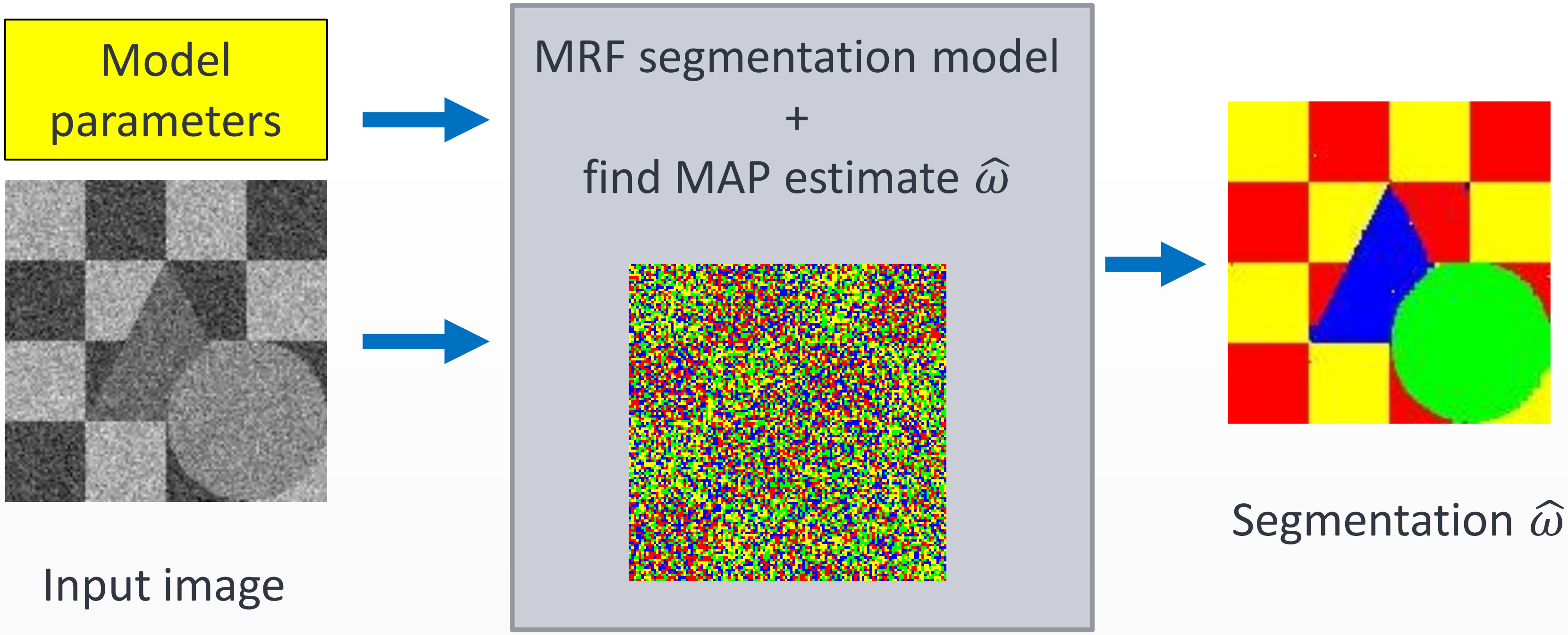
$$U(\omega) = \sum_{c \in C} V_c(\omega) =$$

$$= \sum_{i \in C_1} V_{C_1}(\omega_i) + \sum_{(i,j) \in C_2} V_{C_2}(\omega_i, \omega_j) + \cdots$$



Segmentation of grayscale images: A simple MRF model

- Construct a segmentation model where regions are formed by spatial clusters of pixels with similar intensity:



Model parameters

MRF segmentation model + find MAP estimate $\widehat{\omega}$

Input image

Segmentation $\widehat{\omega}$

MRF segmentation model

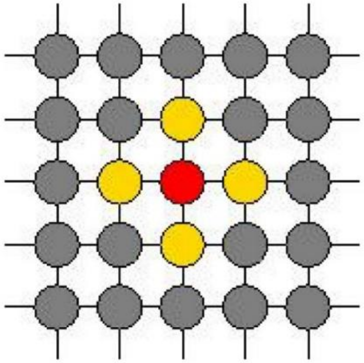
- Pixel labels (or classes) are represented by (for example) Gaussian distributions:

$$P(f_s|\omega_s) = \frac{1}{\sqrt{2\pi}\sigma_{\omega_s}} \exp\left(-\frac{\left(f_s - \mu_{\omega_s}\right)^2}{2\sigma_{\omega_s}^2}\right)$$

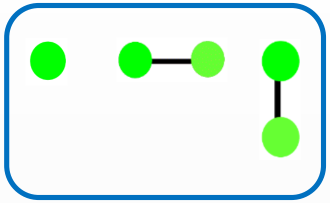
- Clique potentials
  - ***Singleton***: proportional to the likelihood of features given ω : $\log P(f|\omega)$
  - ***Doubleton***: favors similar labels at neighboring pixels – ***smoothness prior***

$$V_{C_2}(i,j) = \beta\delta\left(\omega_i, \omega_j\right) = \begin{cases} -\beta & \text{if } \omega_i = \omega_j \\ +\beta & \text{if } \omega_i \neq \omega_j \end{cases}$$

    - as β increases, regions become more homogenous

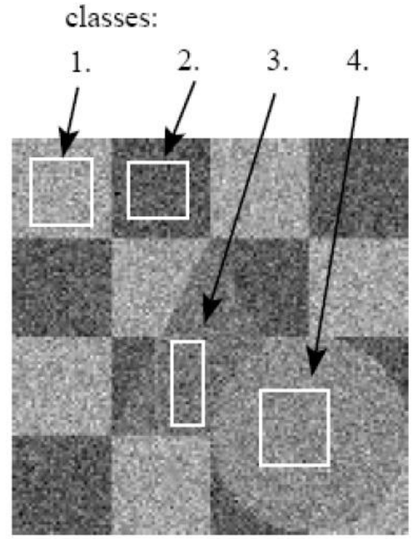


Cliques



Model parameters

- Doubleton potential β
  - less dependent on the input →
    - can be fixed a priori
- Number of labels $|\Lambda|$
  - Problem dependent →
    - usually given by the user or
    - inferred from some higher level knowledge
- Each label $\lambda \in \Lambda$ is represented by a Gaussian distribution $\mathrm{N}(\mu_\lambda, \sigma_\lambda)$:
  - estimated from the input image



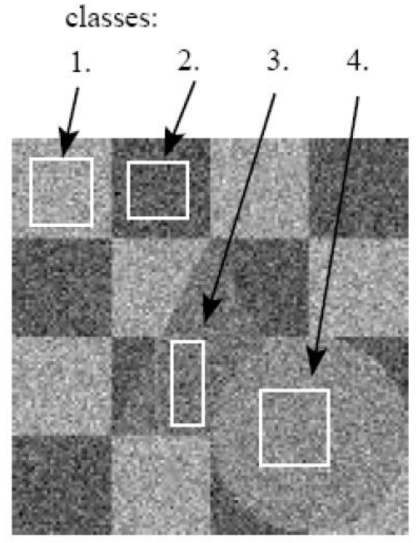
Model parameters

- The class statistics (mean and variance) can be estimated via the ***empirical mean and variance***:

$$\forall \lambda \in \Lambda: \qquad \mu_\lambda = \frac{1}{|S_\lambda|} \sum_{s \in S_\lambda} f_s$$

$$\sigma_\lambda^2 = \frac{1}{|S_\lambda|} \sum_{s \in S_\lambda} (f_s - \mu_\lambda)^2$$

  - where $S_\lambda$denotes the set of pixels in the training set of class λ
  - a training set consists in a representative region selected by the user



Energy function

- Now we can define the energy function of our MRF model:

$$U(\omega)=\sum_{s}\left(\log\left(\sqrt{2\pi}\sigma_{\omega_s}\right)+\frac{\left(f_s-\mu_{\omega_s}\right)^2}{2\sigma_{\omega_s}^2}\right)+\sum_{s,r}\beta\delta(\omega_s,\omega_r)$$

- Recap: the $P(\omega)$ probability can be directly derived from the energy

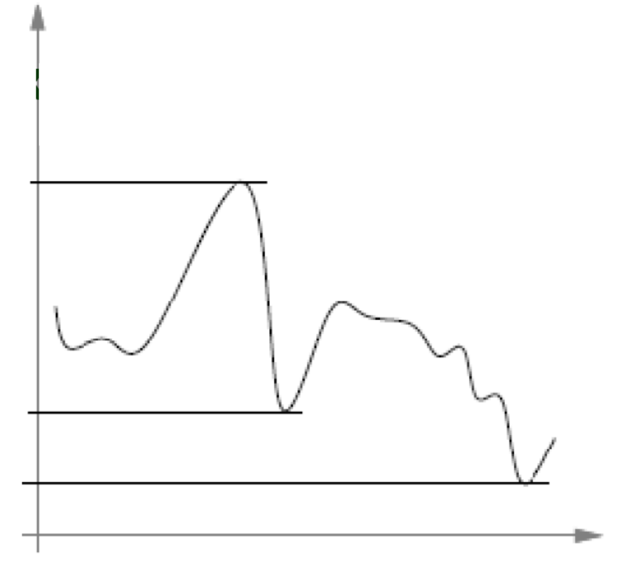
$$P(\omega)=\frac{1}{Z}\exp\left(-U(\omega)\right)=\frac{1}{Z}\exp\left(-\sum_{c\in C}V_c\left(\omega\right)\right)$$

- Hence:

$$\widehat{\omega}^{MAP}=\underset{\omega\in\Omega}{\operatorname{argmax}}\,P(\omega|f)=\underset{\omega\in\Omega}{\operatorname{argmin}}\,U(\omega)$$

Optimization

- Problem reduced to the minimization of a **non-convex** energy function
  - Many local minima
- Gradient descent?
  - Works only if we have a *good* initial segmentation
- Simulated Annealing
  - Always works (at least in theory)

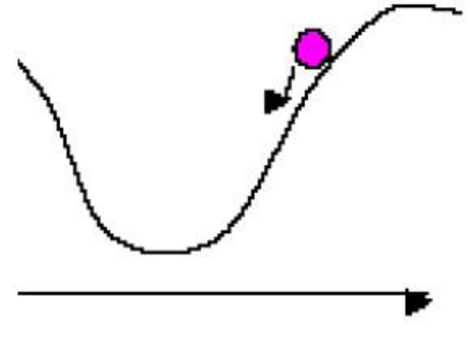


ICM (Iterated Conditional Mode)

~Gradient descent approach [Besag86]

1. Start at a „good” initial configuration ω^0 and set $k = 0$.
2. For each configuration which differs *at most* in one element from the current configuration ω^k (they are denoted by $\mathcal{N}_{\omega^k}$), compute the energy $U(\eta)$ ($\eta \in \mathcal{N}_{\omega^k}$).
3. From the configurations $\mathcal{N}_{\omega^k}$, select the one which has the minimal energy:

$$\omega^{k+1} = \underset{\eta \in \mathcal{N}_{\omega^k}}{\operatorname{argmin}} U(\eta)$$



4. Goto Step 2, with $\mathrm{k} = \mathrm{k} + 1$until convergence obtained (for example the energy change is less than a certain threshold).

ICM (Iterated Conditional Mode)

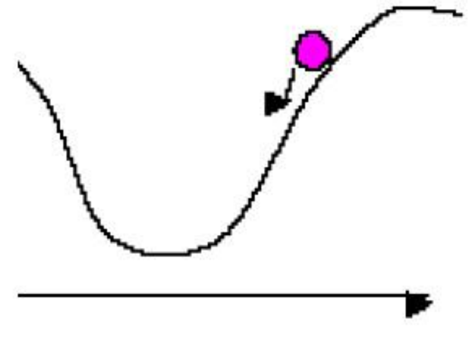
ICM for mage segmentation models

1. Start at a „good” initial segmentation ω^0 and set $k = 0.$
2. For each segmentation which differs *at most* in one pixel’s label (pixel *s*) from the current segmentation ω^k (they are denoted by $\mathcal{N}_{\omega^k}$), compute the energy $\Delta U(\eta) = U(\eta) - U\left(\omega^k\right)$ $(\eta \in \mathcal{N}_{\omega^k})$.
3. From the configurations $\mathcal{N}_{\omega^k}$, select the one which has the minimal energy:

$$\omega^{k+1} = \underset{\eta \in \mathcal{N}_{\omega^k}}{\operatorname{argmin}} \Delta U(\eta)$$

4. Goto Step 2, with $\mathrm{k} = \mathrm{k} + 1$until convergence obtained (for example the energy change is less than a certain threshold).

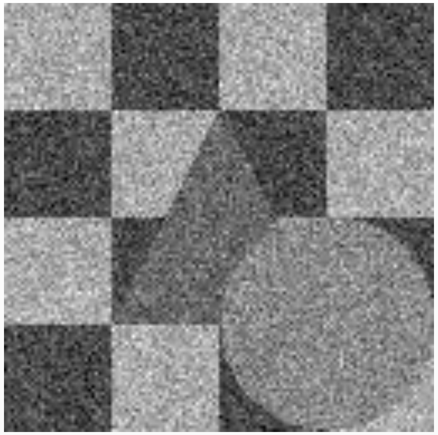
Only depens on pixel s and its four neighbors



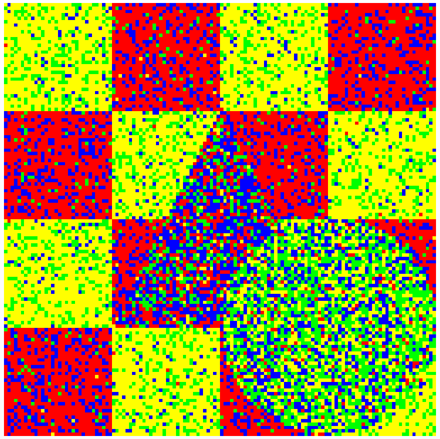
ICM initialization

- Per-pixel Maximum a Posteriori (MAP) estimate:

$$\omega_s^0 = \underset{\lambda\in\Lambda}{\operatorname{argmin}}\left(\log\left(\sqrt{2\pi}\sigma_\lambda\right) + \frac{(f_s-\mu_\lambda)^2}{2\sigma_\lambda^2}\right)$$

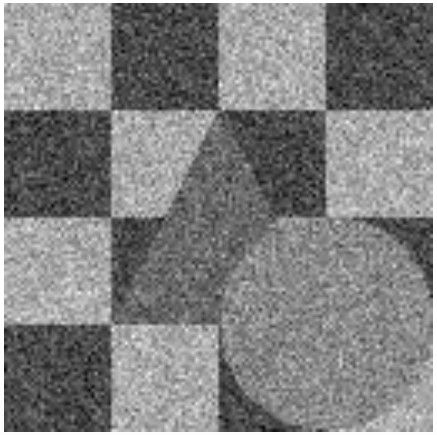


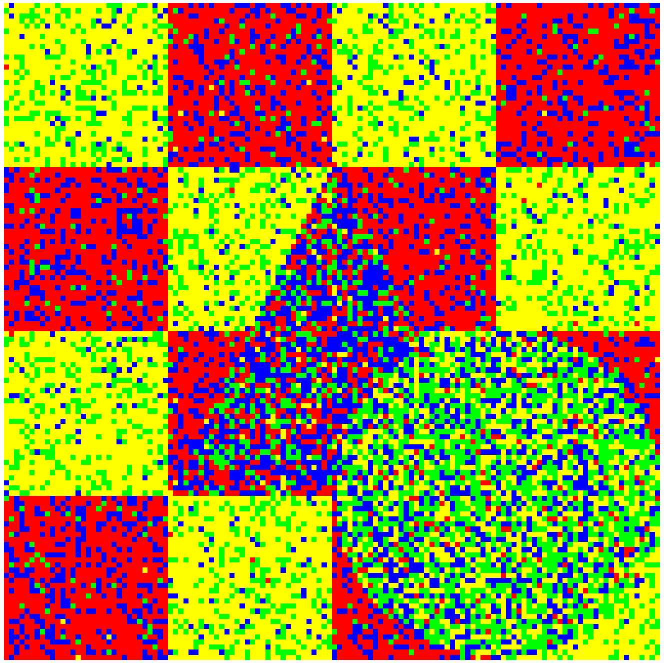
Input image



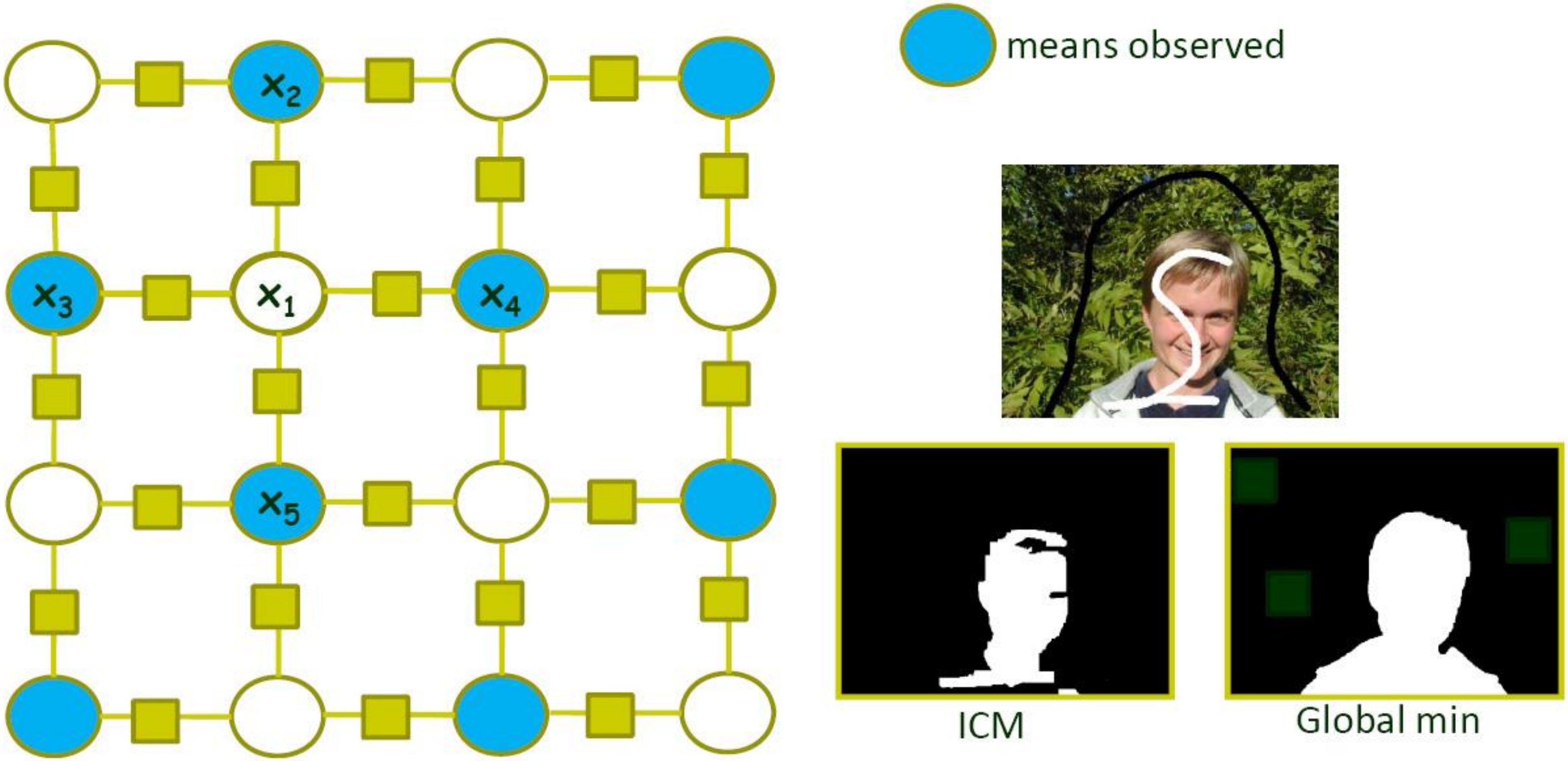
Initial label map

ICM optimization steps





ICM vs. Simulated Annealing



Can get stuck in local minima!

Simulated Annealing: accept a move even if energy increases (with certain probability)

Slide adopted from C. Rother ICCV'09 tutorial: `http://research.microsoft.com/`

Simulated Annealing
Modified Metropolis Dynamics (MMD)

1. Set $k = 0$ and initialize ω randomly. Choose a sufficiently high initial temperature $T = T_{0.}$
2. Construct a trial perturbation η from the current configuration ω such that η differs only in one element from ω.
3. **(Metropolis criteria)** Compute $\Delta U = U(\eta) - U(\omega)$ and accept η if $\Delta U < 0$ else accept with probability $\exp(-\Delta U/T)$ (analogy with thermodynamics):

$$\omega = \begin{cases} \eta & \text{if } \Delta U \leq 0 \\ \eta & \text{if } \Delta U > 0 \text{ and } \xi < \exp(-\Delta U/T) \\ \omega & \text{otherwise} \end{cases}$$

where ξ is a uniform random number in $[0,1[$.

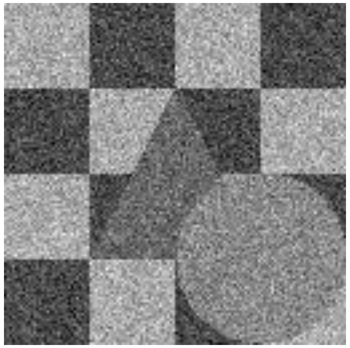
4. Decrease the temperature $T = T_{k+1}$ and goto step 2 with $\mathrm{k} = \mathrm{k} + 1$ until the system is frozen.

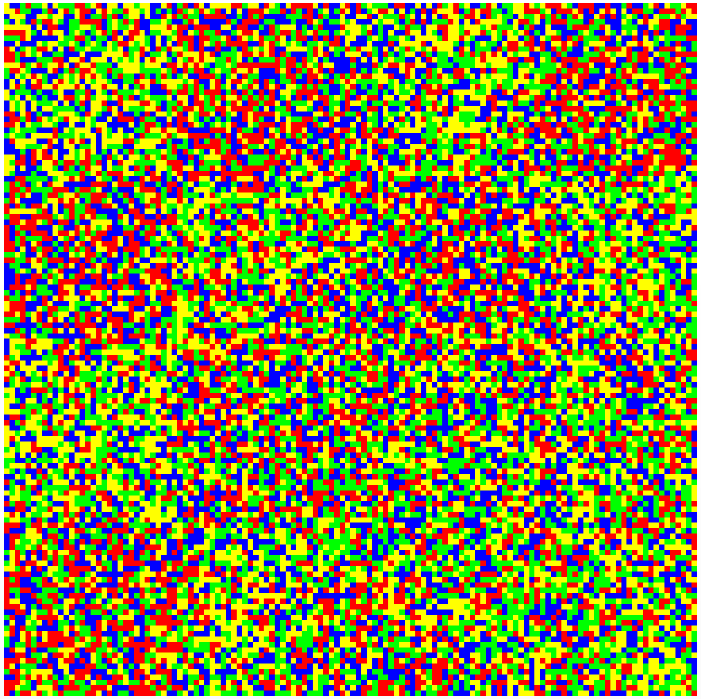
Temperature Schedule

- **In theory:** should be logarithmic – **in practice:** exponential schedule is reasonable
- **Initial temperature:** set it to a relatively low value (~4) → faster execution
  - must be high enough to allow random jumps at the beginning!
- **Schedule:** $T_{k+1} = c \cdot T_k, \quad k = 0,1,2, \dots$ (e.g. $c = 0.95$).
- **Stopping criteria:**
  - Fixed number of iterations
  - Energy change is less than a thresholds

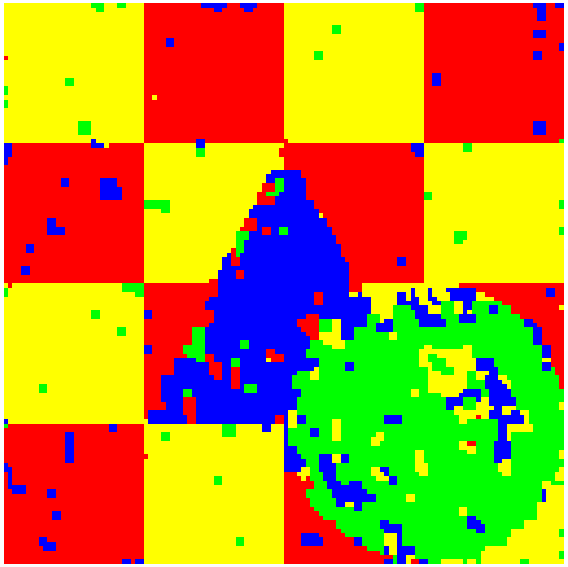
MMD segmentation

- Starting MMD: random label map!

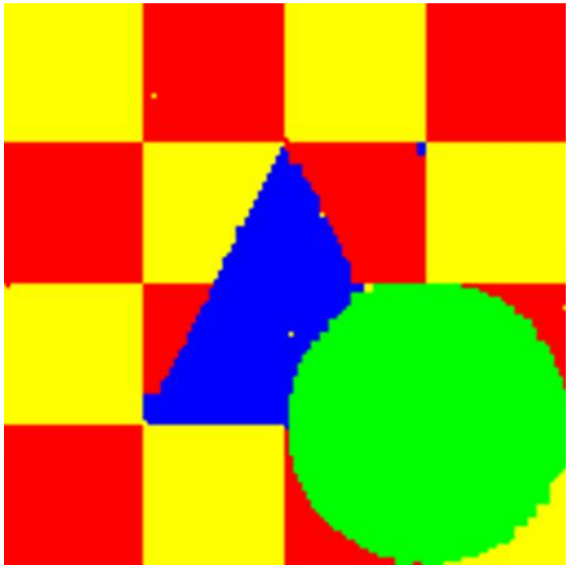




ICM vs MMD



ICM result

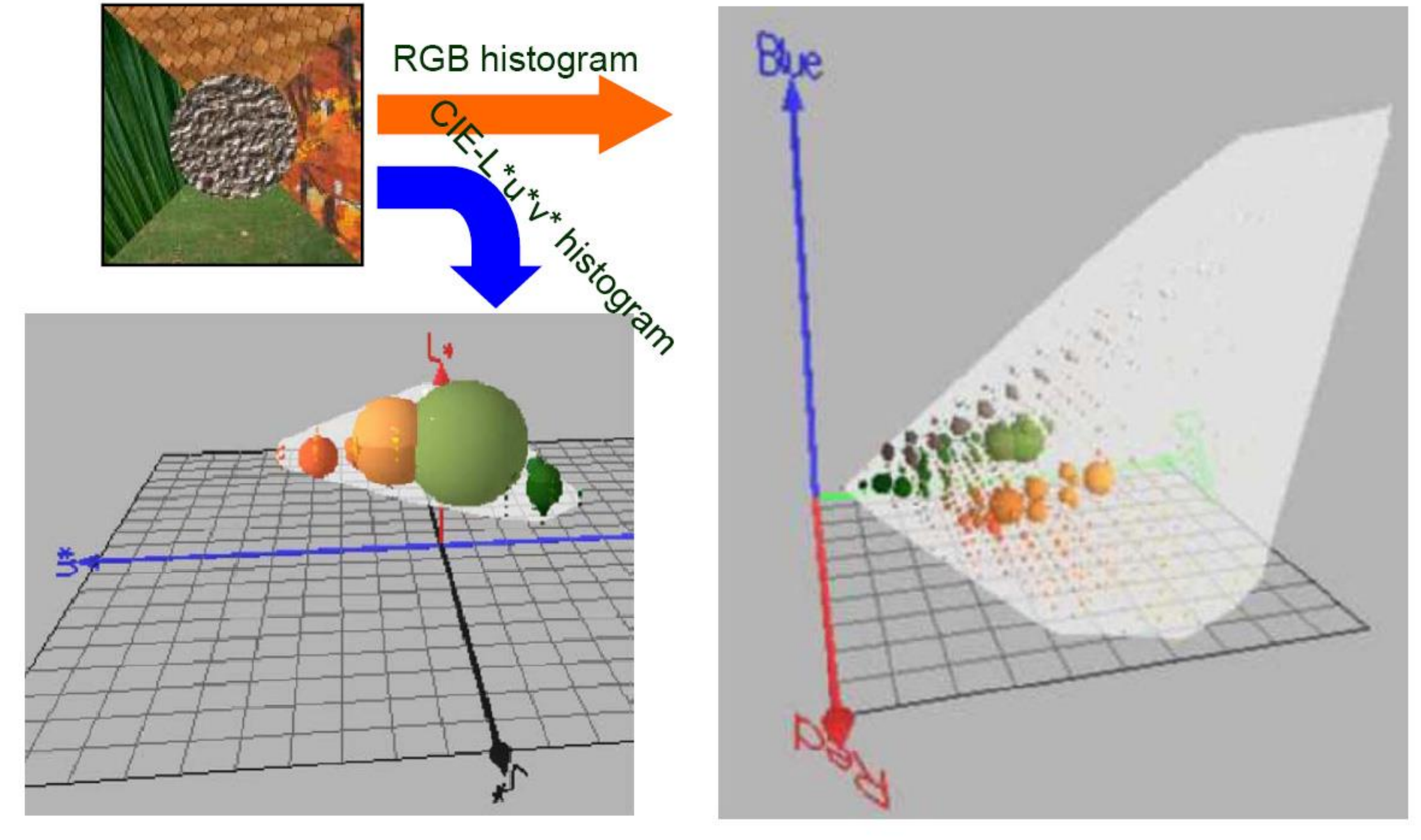


MMD result

MRF Summary

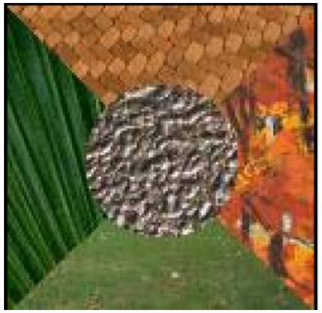
- Design your model carefully
  - Optimization is just a tool, do not expect a good segmentation from a wrong model
- What about other than graylevel features?
  - Extension to color is relatively straightforward

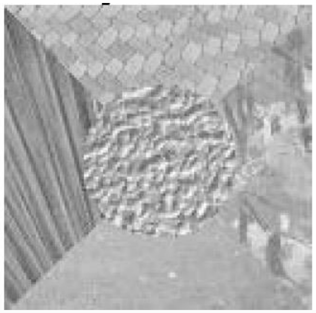
What color features?



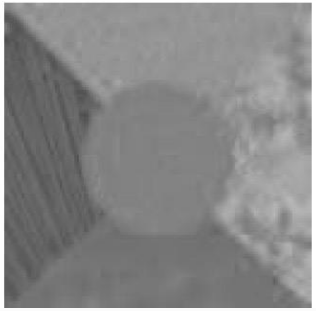
Extract Color Feature

- We adopt the CIE-L*u*v* color space because it is ***perceptually uniform***.
  - Recap from earlier slides: similarly to CIE-L*a*b*, color difference can be measured here by Euclidean distance of two color vectors.
- We convert each pixel from RGB space to CIEL*u*v* space
  - We have 3 color feature images

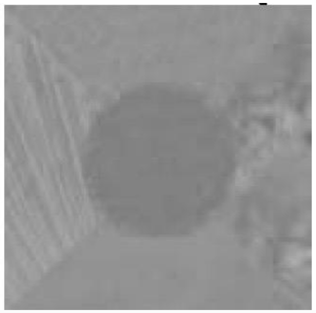




L*

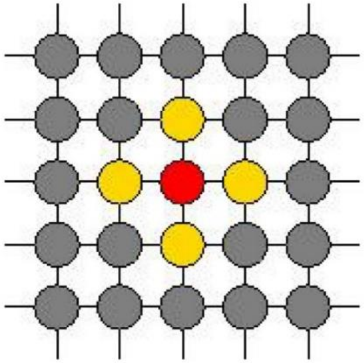


u*



v*

Color MRF segmentation model



- Pixel labels (or classes) are represented by three-variate Gaussian distributions

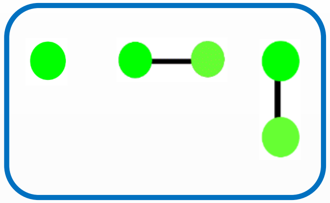
$$P(f_s|\omega_s) = \frac{1}{\sqrt{2\pi}|\Sigma_{\omega_s}|}\exp\left(-\frac{1}{2}(\bar{f_s} - \bar{\mu}_{\omega_s})\Sigma_{\omega_s}^{-1}(\bar{f_s} - \bar{\mu}_{\omega_s})^T\right)$$

- Clique potentials
  - ***Singleton***: proportional to the likelihood of features given ω : $\log P(f|\omega)$
  - ***Doubleton***: favors similar labels at neighboring pixels – ***smoothness prior***

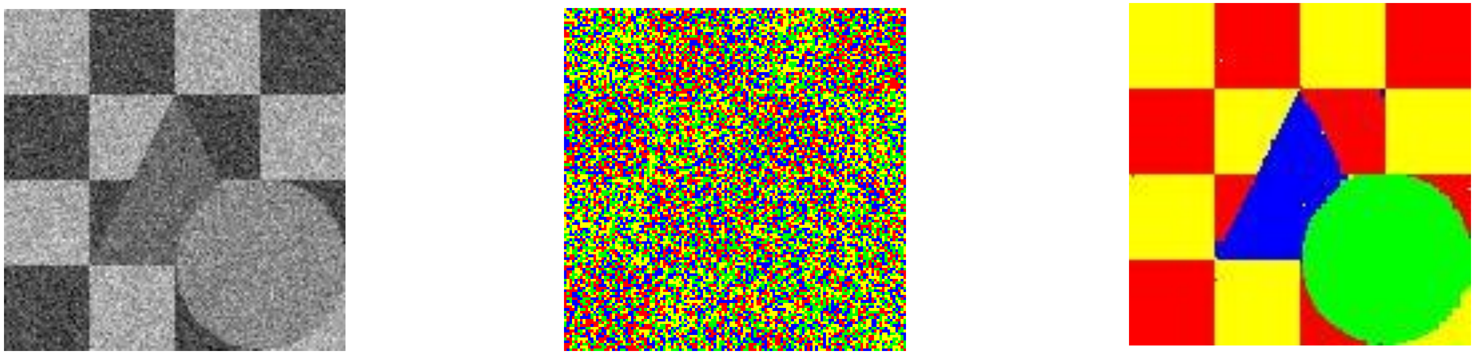
$$V_{C_2}(i,j) = \beta\delta(\omega_i, \omega_j) = \begin{cases} -\beta & \text{if } \omega_i = \omega_j \\ +\beta & \text{if } \omega_i \neq \omega_j \end{cases}$$

  - as β increases, regions become more homogenous

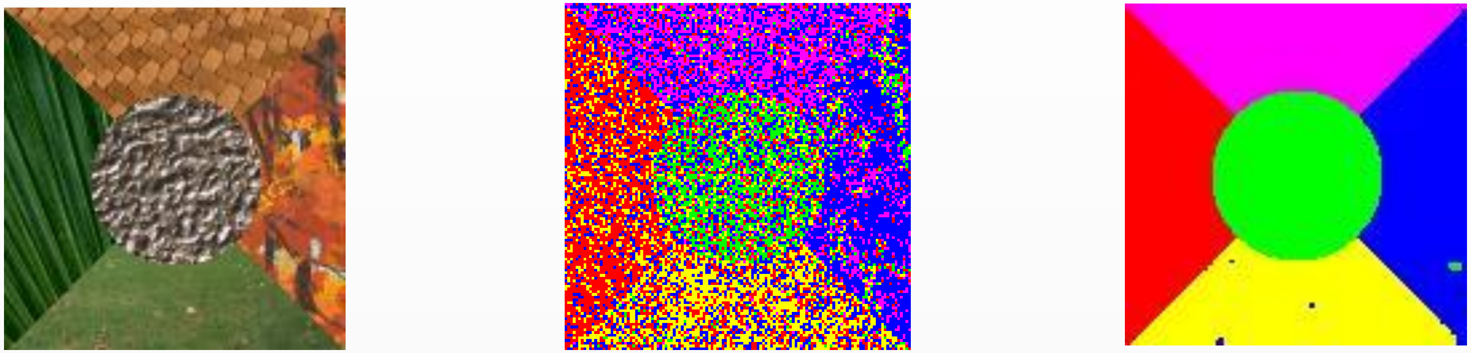
Cliques



Segmentation examples



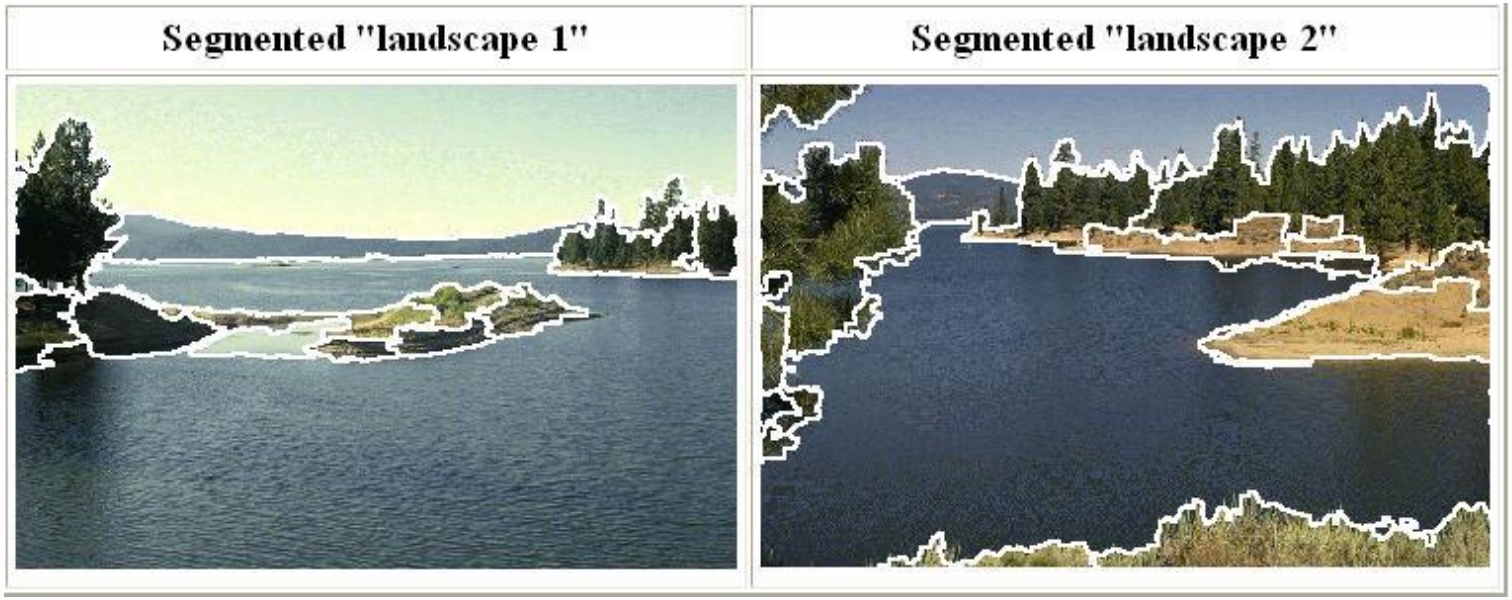
gray level based segmentation



color image segmentation

Mean shift segmentation

- Versatile technique for clustering-based segmentation



D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. 603-619, May 2002

Mean shift clustering

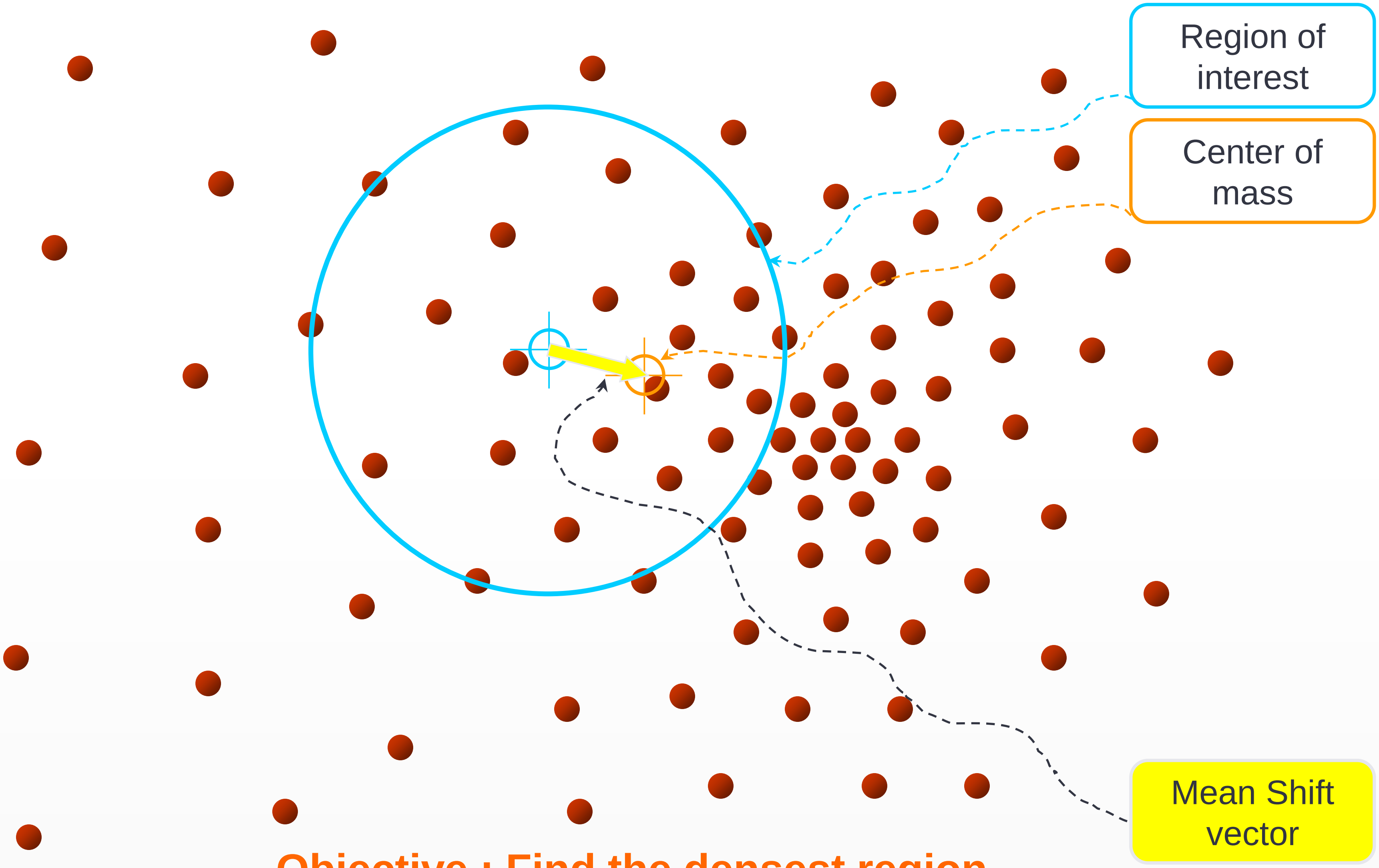
- Sources: Mean Shift Theory and Applications, presentation of Yaron Ukrainitz & Bernard Sarel
- Further credits:
  - Alper Yilmaz, Afshin Dehghan
  - Lecture of Mubarak Shah, UCF FL, USA
    - https://www.youtube.com/watch?v=M8B3RZVqgOo

Origin: Mean-Shift Clustering

- Non-parametric iterative clustering technique introduced in 1975 by Fukunaga and Hostetler.
- Do not need to know the number of clusters a priori.
- Does not constrain the shape of the cluster.
- Mean shift considers the points in the feature space as samples from an underlying probability density function.
- The objective of the algorithm is to find the modes of this PDF, and associate each point with the node it is „attracted to”.

Fukunaga and Hostetler, "The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition", IEEE Transactions on Information Theory vol 21 , pp 32-40 ,1975

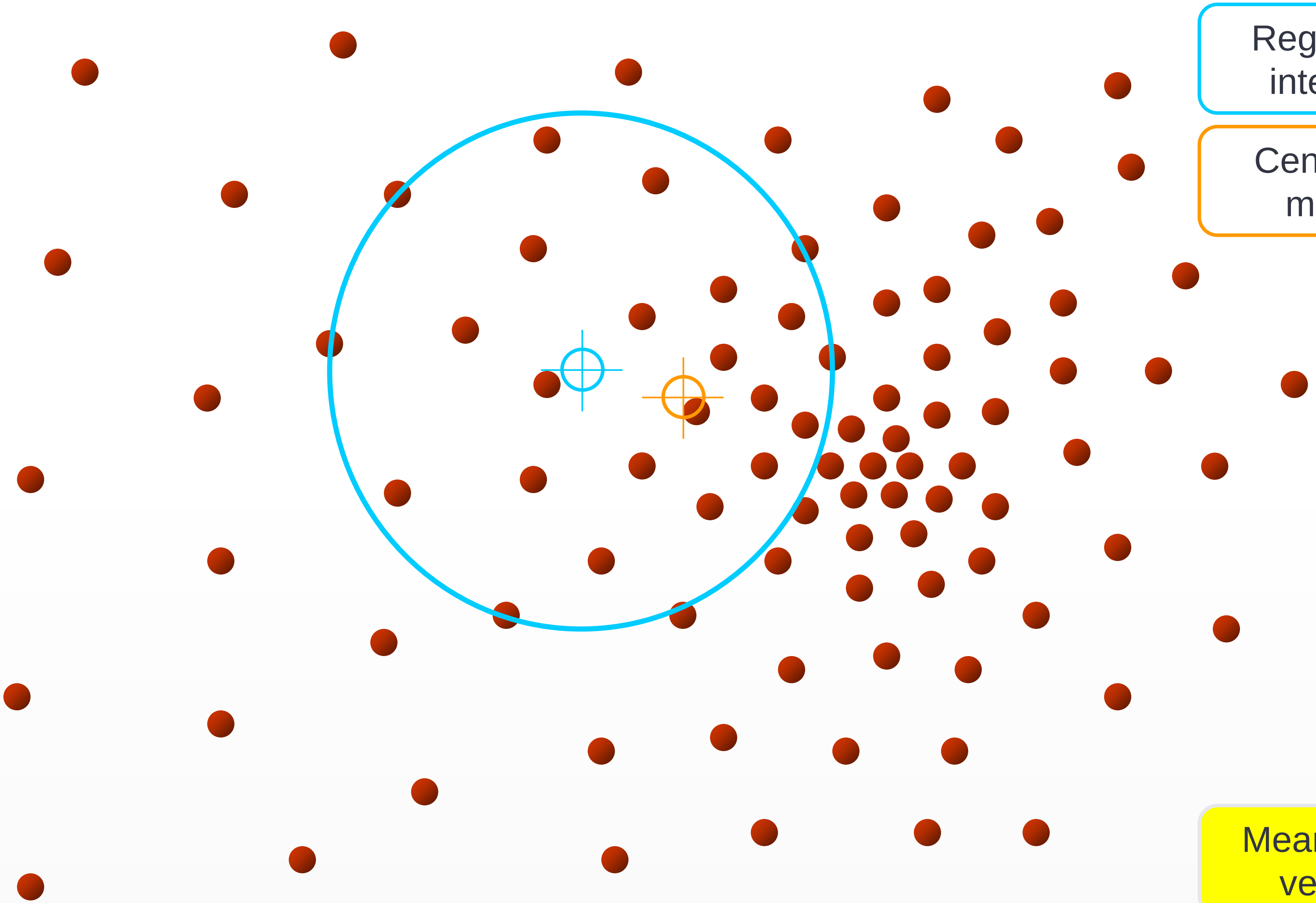
Mean shift – intuitive description



Objective : Find the densest region

Distribution of identical billiard balls

Mean shift – intuitive description



Region of interest

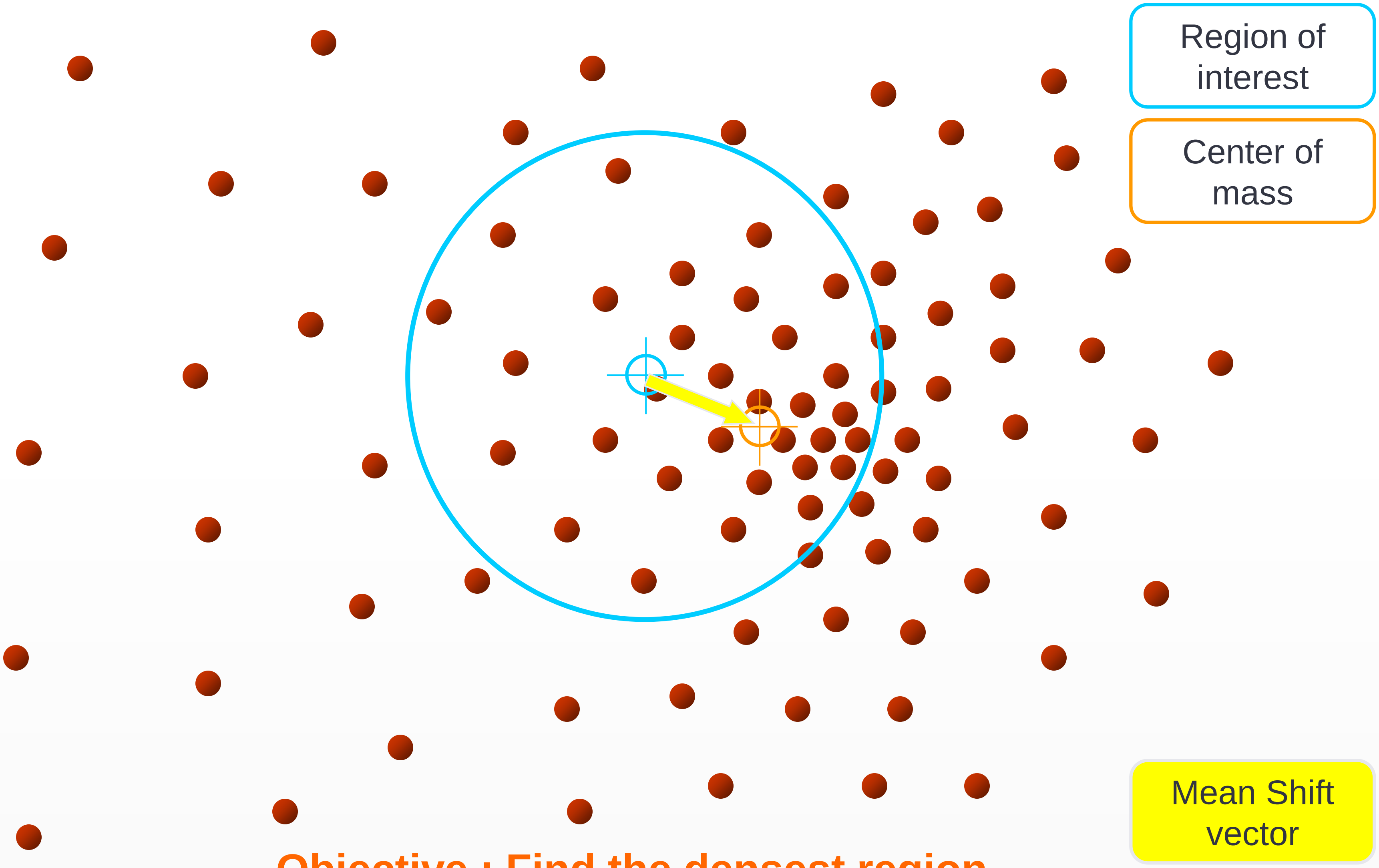
Center of mass

Mean Shift vector

Objective : Find the densest region

Distribution of identical billiard balls

Mean shift – intuitive description



Region of interest

Center of mass

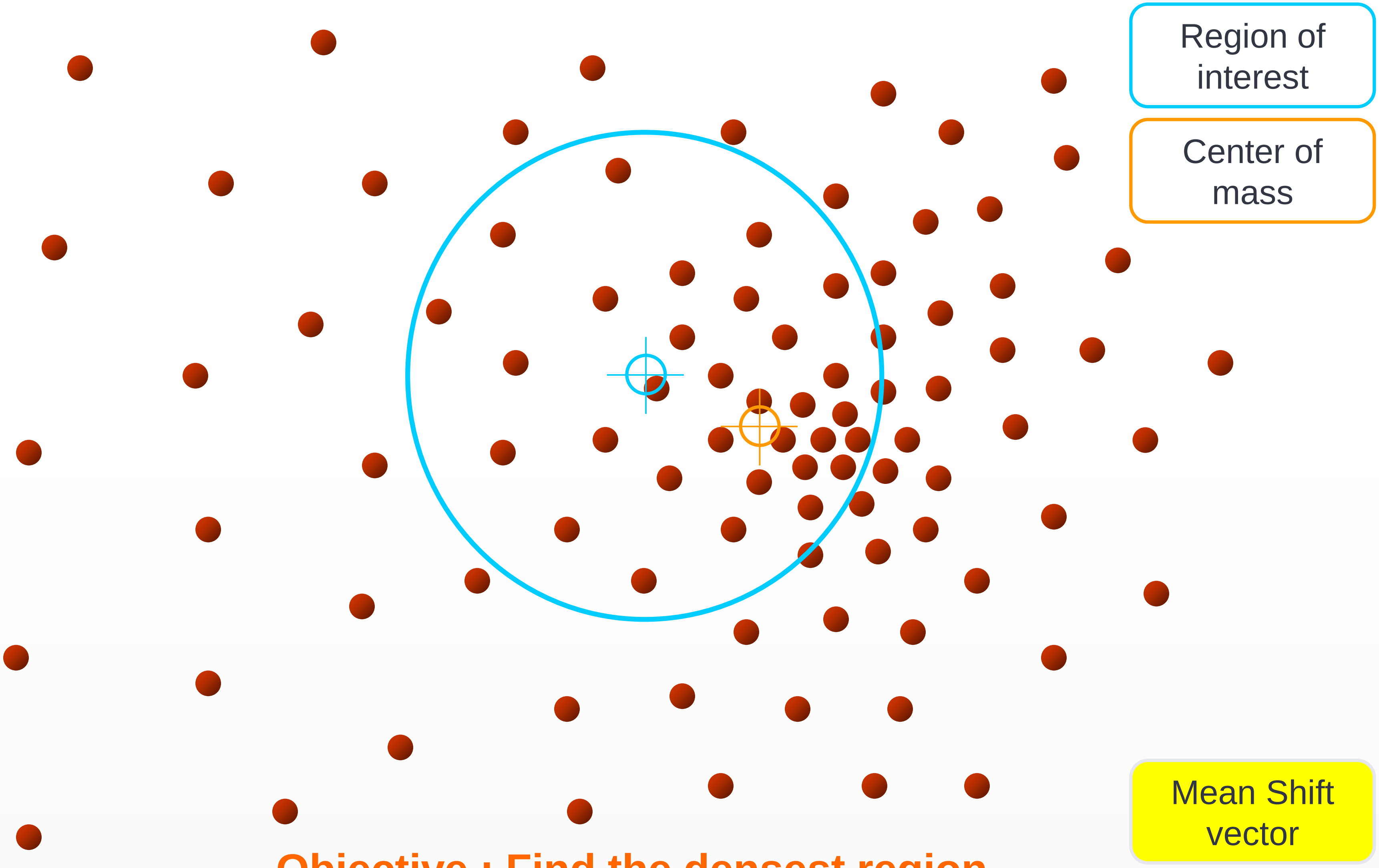
Mean Shift vector

Objective : Find the densest region

Distribution of identical billiard balls

Slide by Y. Ukrainitz & B. Sarel

Mean shift – intuitive description

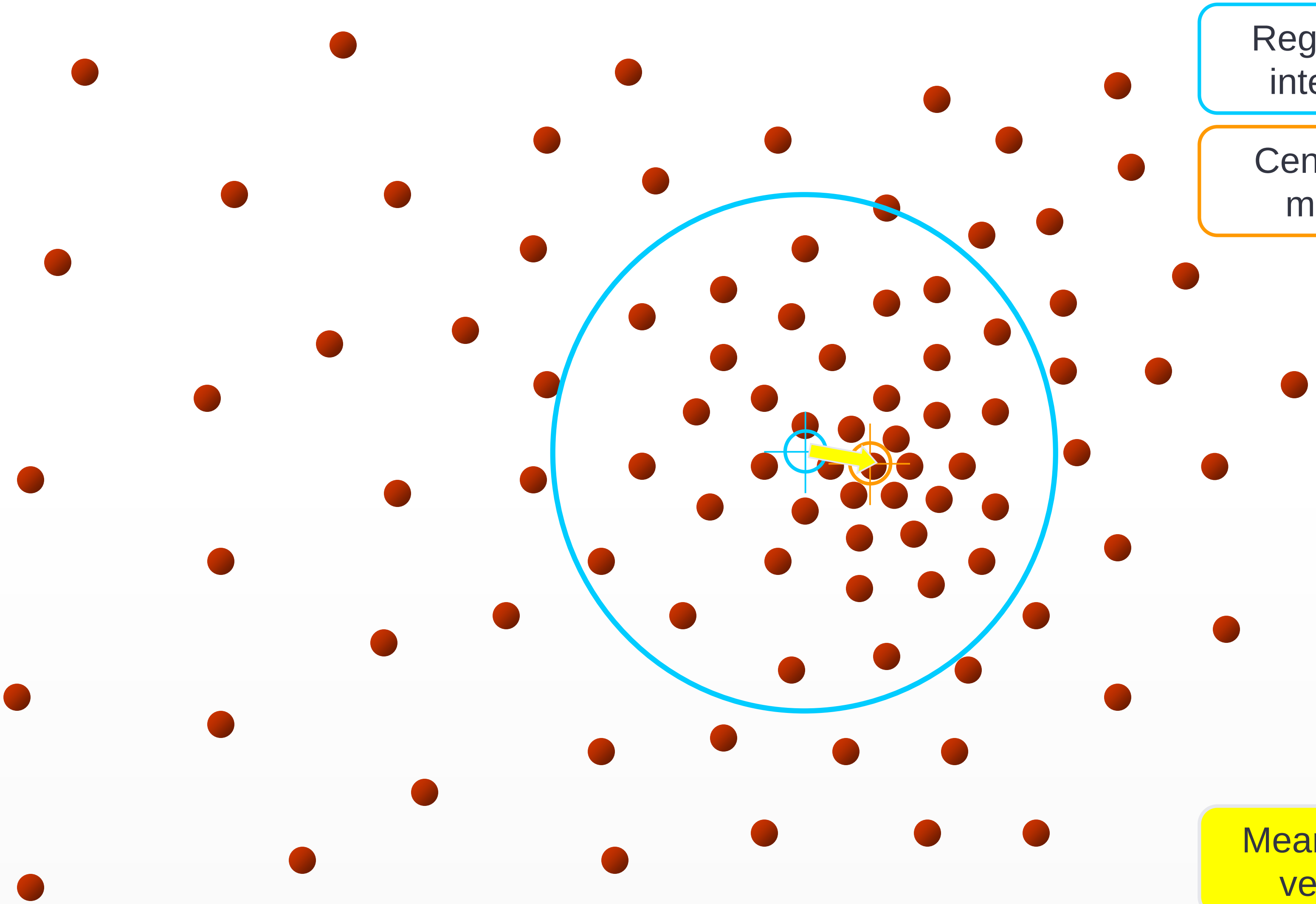


Objective : Find the densest region

Distribution of identical billiard balls

Slide by Y. Ukrainitz & B. Sarel

Mean shift – intuitive description



Region of interest

Center of mass

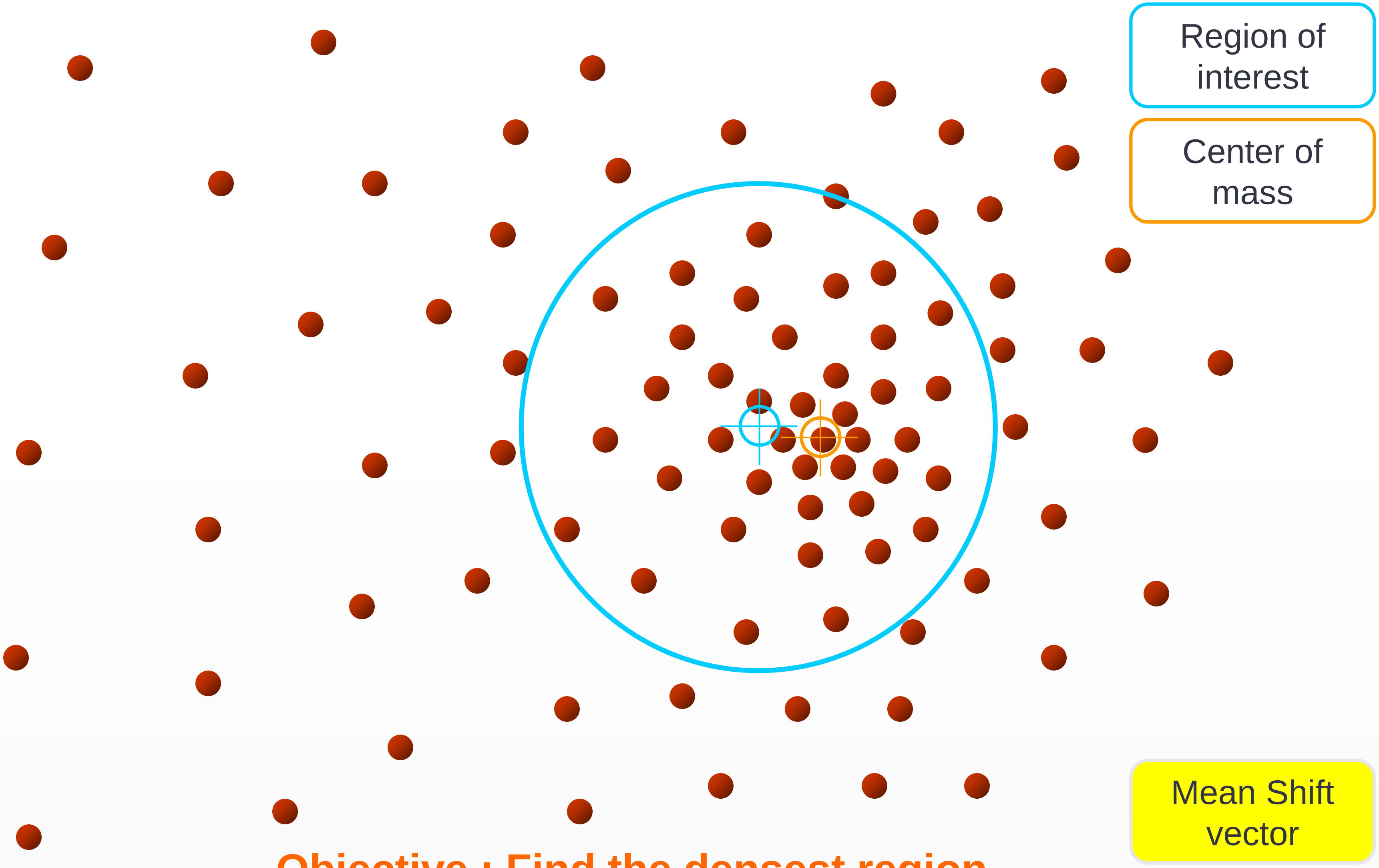
Mean Shift vector

Objective : Find the densest region

Distribution of identical billiard balls

Slide by Y. Ukrainitz & B. Sarel

Mean shift – intuitive description

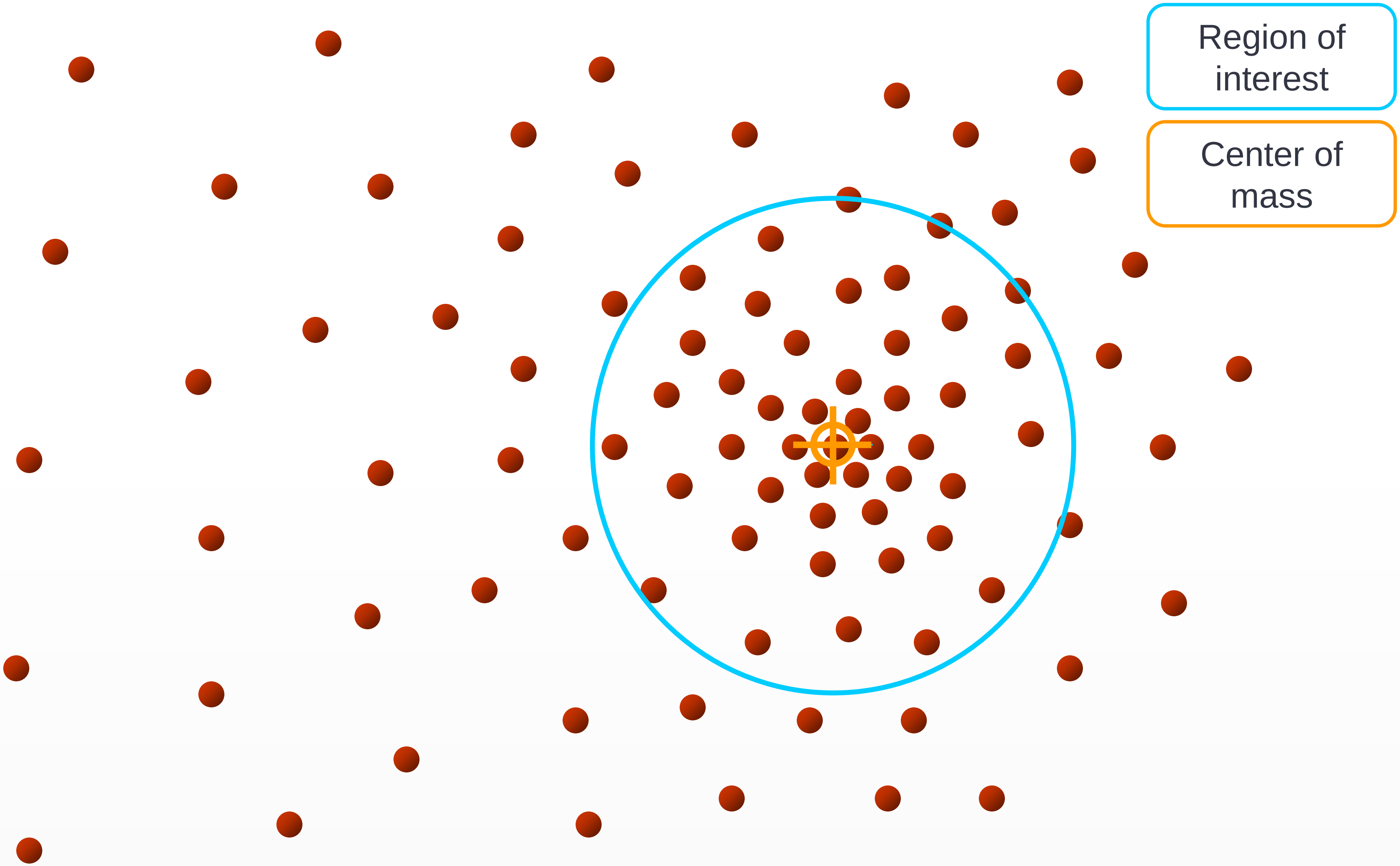


Objective : Find the densest region

Distribution of identical billiard balls

Slide by Y. Ukrainitz & B. Sarel

Mean shift – intuitive description



Objective : Find the densest region

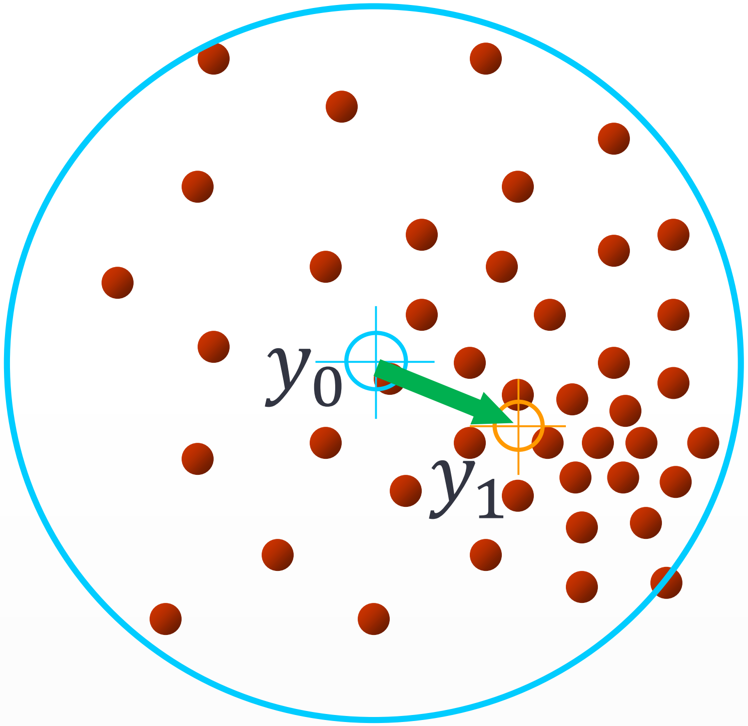
Distribution of identical billiard balls

Slide by Y. Ukrainitz & B. Sarel

Mean shift vector

- Given:
  - Data points and approximate location of the mean of this data
- Task:
  - Estimate the exact location of the mean of the data by determining the shift vector from the initial mean
  - We do this iteratively, until we do not have to move (mean shift vector equals to zero)

Mean shift vector example



$$m_h(y) = \left[\frac{1}{n_x}\sum_{i=1}^{n_x} x_i\right] - y_0$$

- Mean shift vector always points towards the direction of the maximum increase in the density

Mean shift with point weights

$$m_h(y_0) = \left[\frac{\sum_{i=1}^{n_x} w_i(y_0) \cdot x_i}{\sum_{i=1}^{n_x} w_i(y_0)}\right] - y_0$$

- n_x: number of points in the kernel
- y_0: initial mean location
- x_i: data points
- h: kernel radius

- Weights are determined by different kernels:
  - Uniform, Gaussian, Epanechnikov

What is Mean Shift ?

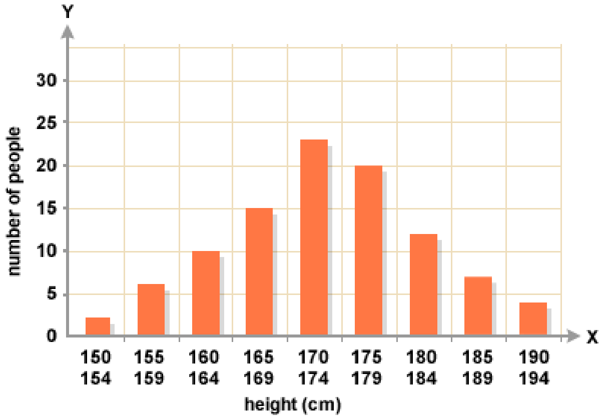
- A tool for:
  - Finding modes in a set of data samples, manifesting an underlying probability density function (PDF) in R^{N}
- PDF in feature space
  - Color space
  - Scale space
  - Actually any feature space you can conceive
  - …

Parametric vs. nonparametric distributions

- Problem: model the height distribution of people in the class
  - Approximate the histogram with a Gaussian density:

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

    - μ and σ are empirical mean and stdev values calculated from the samples (i.e. people in the class)



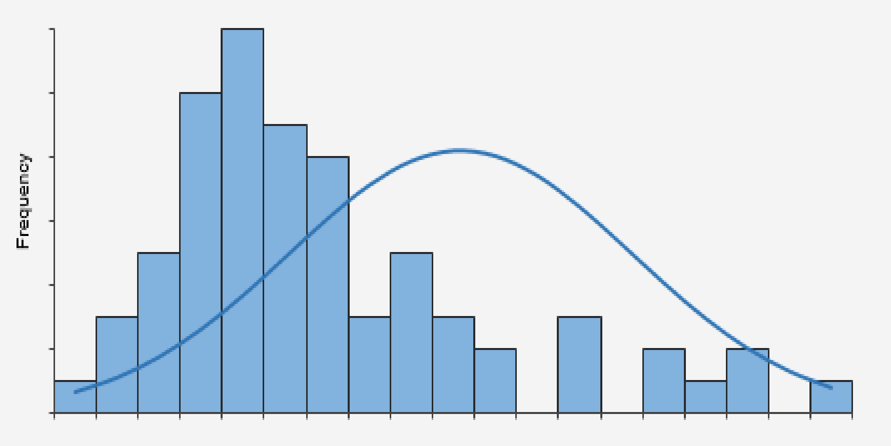
$$P(x_1 < x \le x_2) = \int_{x_1}^{x_2} f(x)dx$$

- Parametric distributions:
  - We have a closed formula for the probability density function (PDF) with a few parameters
  - Estimate the PDF parameters from the samples, then forget the samples and use the pdf directly for probability calculation
  - Various distributions exist: Gaussian, Poisson, Gamma, Beta, etc…

Parametric vs. **nonparametric** distributions

- What happens if the distribution of samples…
  - … does not fits any well known parametric pdf formula, or…
  - … we cannot decide what sort of formula we need the use (too few samples)

$$f(x) = ???$$

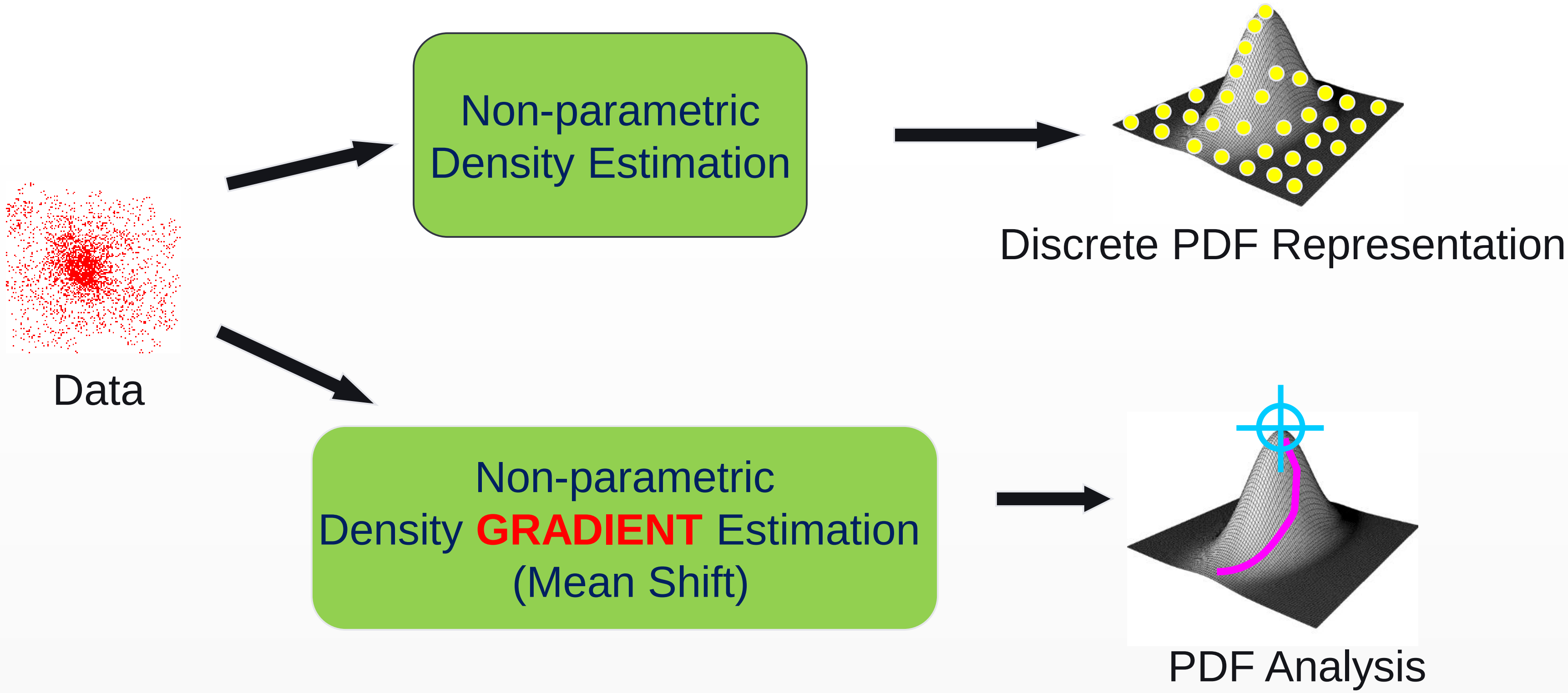


Non-Gaussian distribution

- Non-Parametric distributions:
  - We do not have a closed formula for the probability density function (PDF)
  - Instead, we need to store the samples, and use the samples directly to model the PDF
  - *Our desire:* the value of $f(x)$ should be „high”, if we find „a lot of samples” around x

What is Mean Shift ?

- A tool for:
  - Finding modes in a set of data samples, manifesting an underlying probability density function (PDF) in R^N



Data

Non-parametric Density Estimation

Discrete PDF Representation

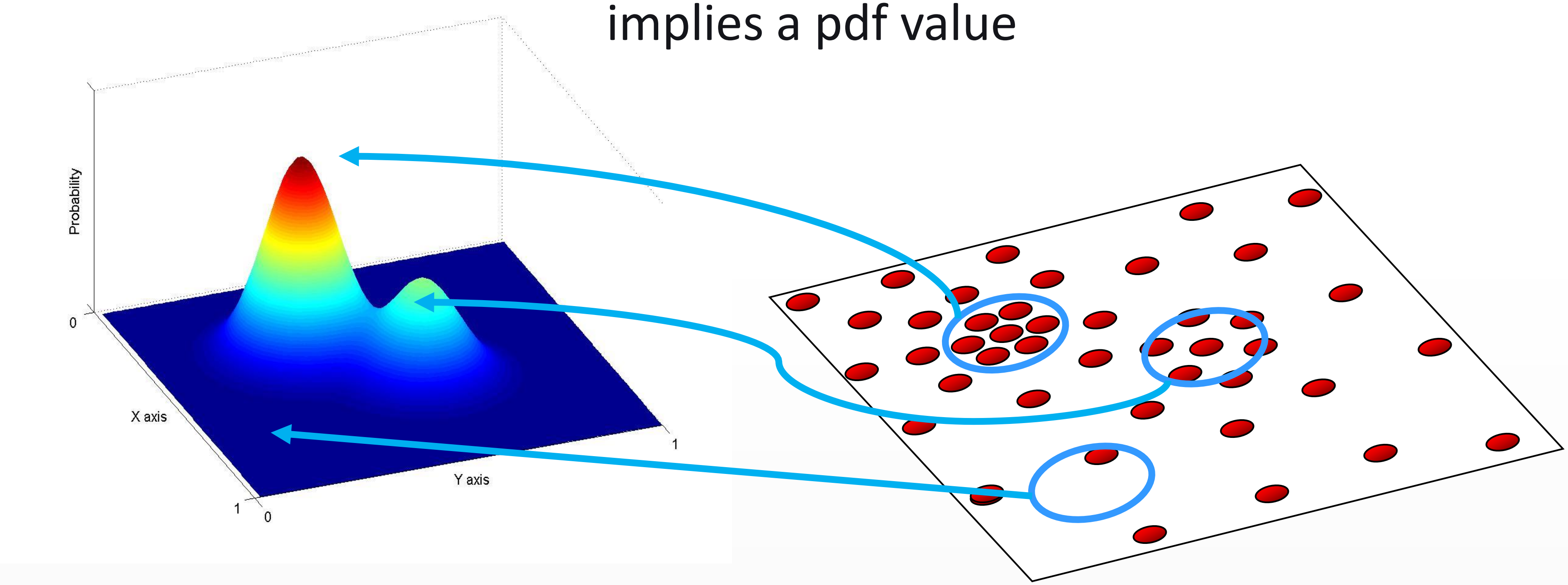
Non-parametric Density **GRADIENT** Estimation (Mean Shift)

PDF Analysis

Non-Parametric Density Estimation

2D data points

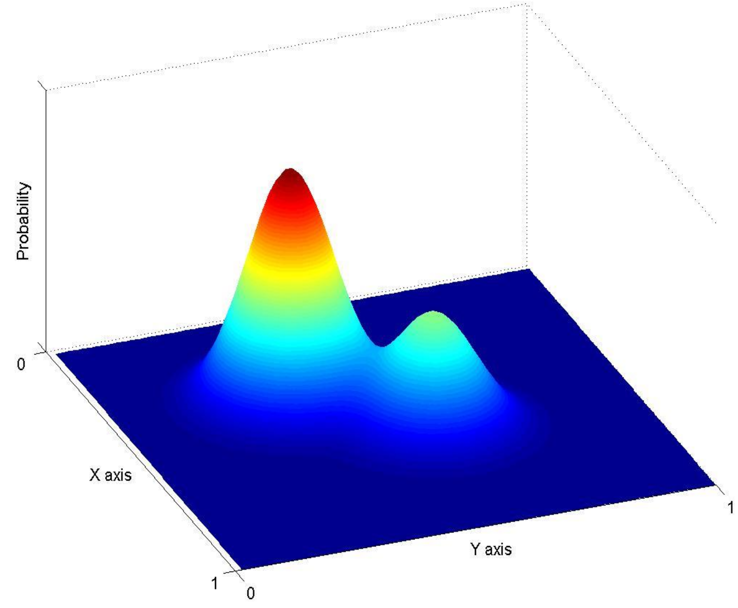
The data point density implies a pdf value

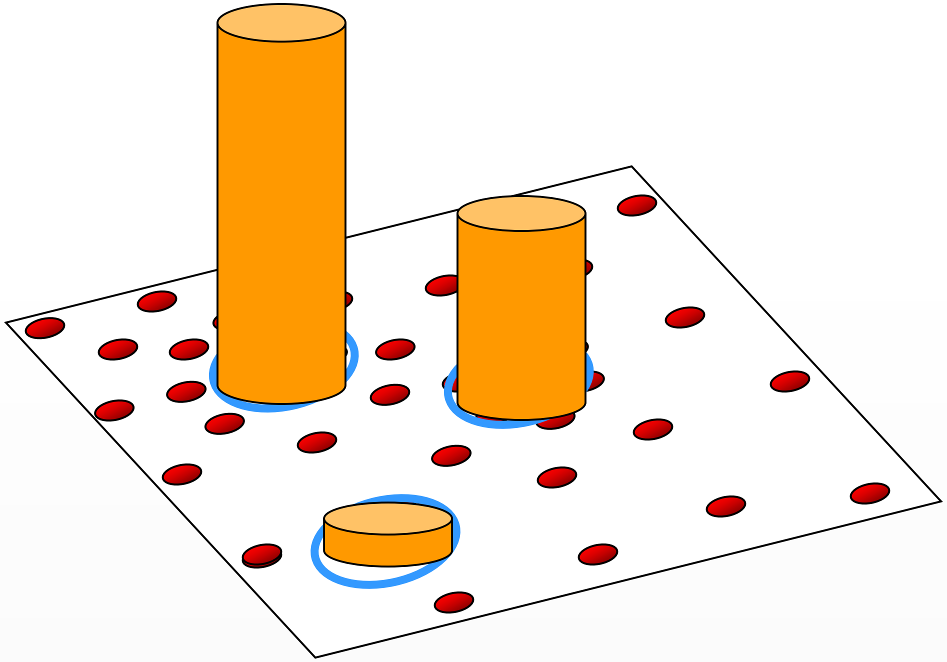


Assumed Underlying PDF

Real Data Samples

Non-Parametric Density Estimation

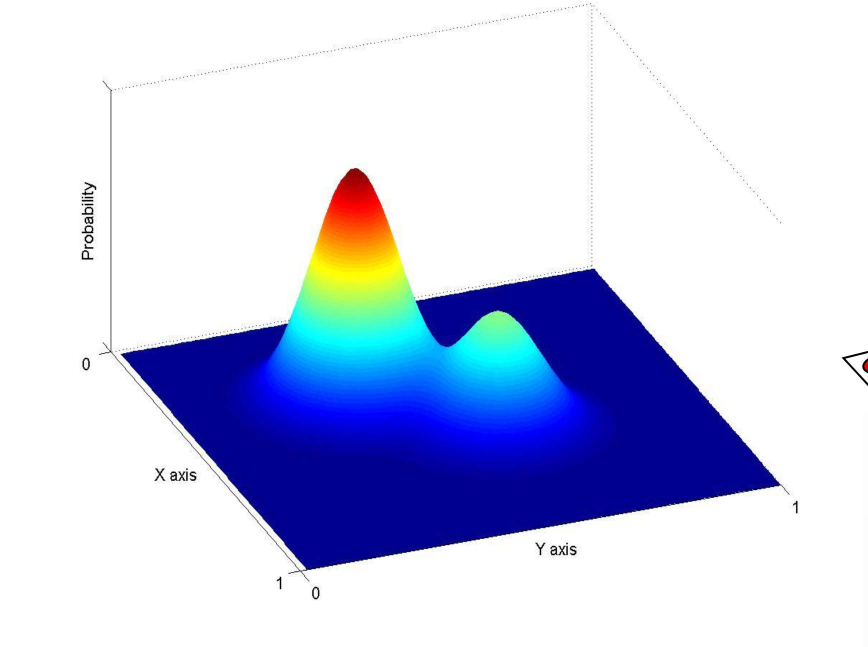


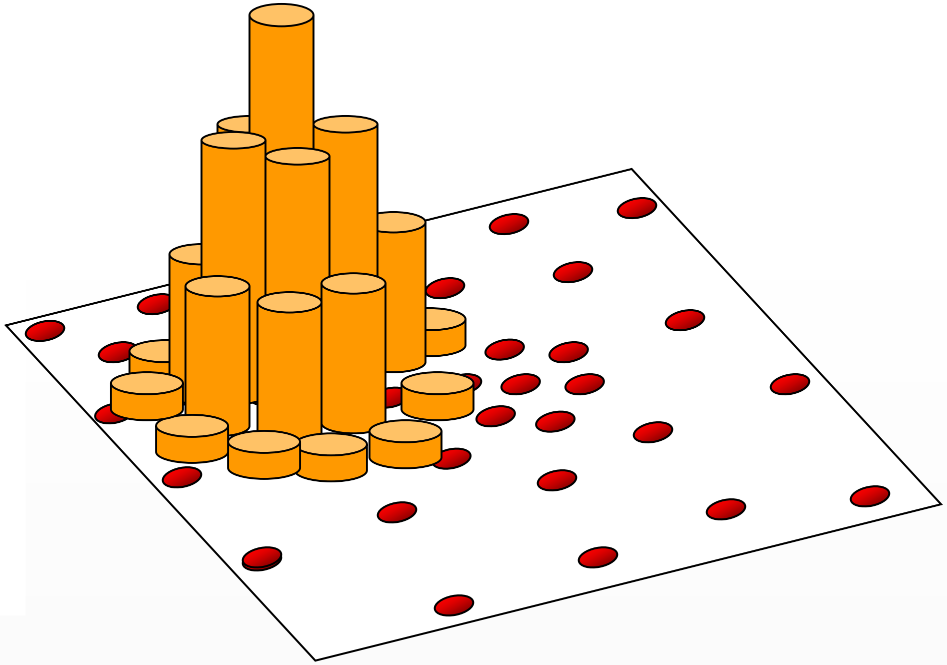


Assumed Underlying PDF

Real Data Samples

Non-Pa?rametric Density Estimation



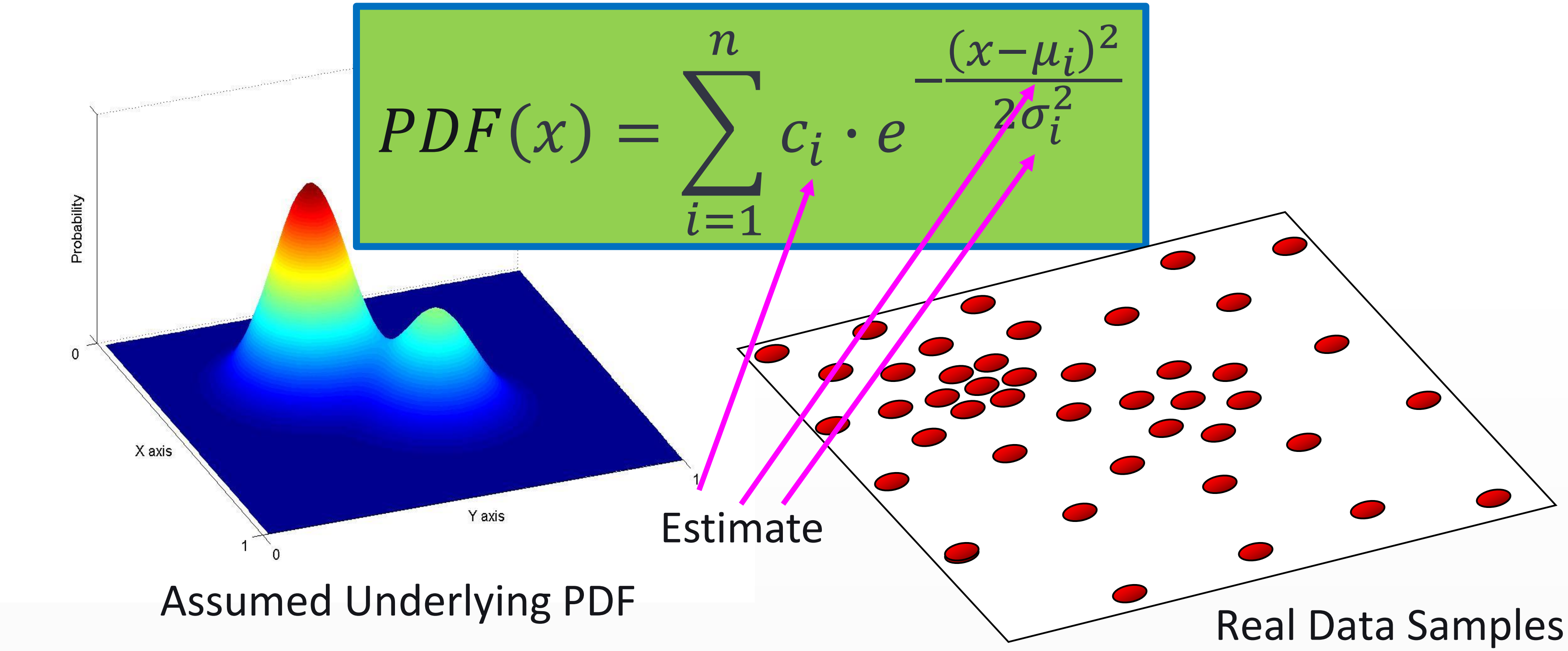


Assumed Underlying PDF

Real Data Samples

Kernel Density Estimation

Assumption : The data points are sampled from an underlying PDF



$$PDF(x) = \sum_{i=1}^{n} c_i \cdot e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

Estimate

Assumed Underlying PDF

Real Data Samples

- Each sample point contributes to the *PDF* with an additive term (here: Gaussian) - μ_i : equal to the ith sample

Kernel Density Estimation

$$PDF(x) = \sum_{i=1}^{n} c_i \cdot e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

- Non-parametric *PDF* with Gaussian kernel:
  - Seems like a mixture of Gaussians, where the number of components is equal to the number of samples, and the mean values of the components are at the sample points $\mu_1, \mu_2, \dots, \mu_n$
- Probability calculation for particular x value:
  - We calculate it as a weighted sum from the surrounding sample points - all the points contribute!
  - We look at the distance of x from each sample point
  - The *PDF* value is high for x which has a lot of samples around it

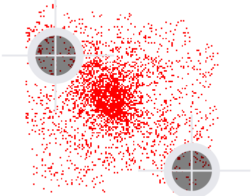
Kernel Density Estimation

Various Kernels

- ***Roles of kernels:*** they determine the weights of nearby points in the density calculation.

$$P(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x}_i)$$

A function of some finite number of data points $x_1 \ldots x_n$

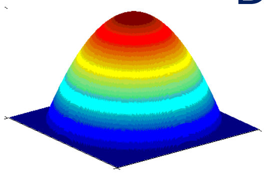


Data

Examples:

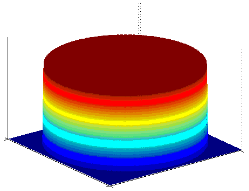
- Epanechnikov Kernel

$$K_E(\mathbf{x}) = \begin{cases} c\left(1-\|\mathbf{x}\|^2\right) & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



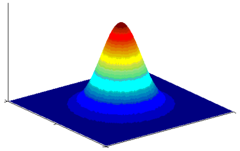
- Uniform Kernel

$$K_U(\mathbf{x}) = \begin{cases} c & \|\mathbf{x}\| \le 1 \\ 0 & \text{otherwise} \end{cases}$$



- Normal Kernel

$$K_N(\mathbf{x}) = c \cdot \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right)$$



Profile and kernel

- Radially symmetric kernel

$$K(x) = ck(\|x\|^2)$$

Profile

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} K(x - x_i) = \frac{1}{n}c\sum_{i=1}^{n} k(\|x - x_i\|^2)$$

Kernel Density Estimation

- Non parametric probability function (pdf)
  - We do not have any assumptions about the closed form of the distribution (such as Gaussian or mixture of Gaussians)
  - We estimate the pdf directly from the sample points $x_1 \dots x_n$

$$P(x) = \frac{1}{n} c \sum_{i=1}^{n} k(\|x - x_i\|^2)$$

Non parametric probability density calculation

$$P(x) = \frac{1}{n} c \sum_{i=1}^{n} k(\|x - x_i\|^2)$$

- Given feature vector x
  - e.g. 1D gray value, 3D color vector, 6D vector of color + texture components etc.
- Task: calculate the probability (density) value of x directly from the sample points $x_1 \dots x_n$
  - Calculate the Euclidean distance d_i of x from each x_i.
  - Use a kernel profile $k(.)$ which assigns a weight to x_i as a function of the calculated d_i distance (for lower distance higher weight, see different kernels)
  - Take the *pdf* value as a the normalized sum of the weights
  - High *pdf* values corresponds to x features which have several x_i-s „nearby”

Kernel Density Estimation

- Relations of nonparametric pdfs and means shift

$$P(x) = \frac{1}{n} c \sum_{i=1}^{n} k(\|x - x_i\|^2)$$

- Derivative of the pdf (gradient of the density):

$$\nabla P(x) = \frac{1}{n} c \sum_{i=1}^{n} \nabla k(\|x - x_i\|^2)$$

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} (x - x_i) k'(\|x - x_i\|^2)$$

Kernel Density Estimation

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} (x - x_i) k'(\|x - x_i\|^2)$$

$$g(x) \coloneqq -k'(x)$$

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} (x_i - x) g(\|x - x_i\|^2)$$

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} x_i g(\|x - x_i\|^2) \; - \frac{1}{n} 2c \sum_{i=1}^{n} x g(\|x - x_i\|^2)$$

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} g(\|x - x_i\|^2) \left[\frac{\sum_{i=1}^{n} x_i g(\|x - x_i\|^2)}{\sum_{i=1}^{n} g(\|x - x_i\|^2)} - x \right]$$

Kernel Density Estimation

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} g(\|x - x_i\|^2) \left[\frac{\sum_{i=1}^{n} x_i g(\|x - x_i\|^2)}{\sum_{i=1}^{n} g(\|x - x_i\|^2)} - x \right]$$

$$\nabla P(x) = \frac{1}{n} 2c \sum_{i=1}^{n} g_i \left[\frac{\sum_{i=1}^{n} x_i g_i}{\sum_{i=1}^{n} g_i} - x \right]$$

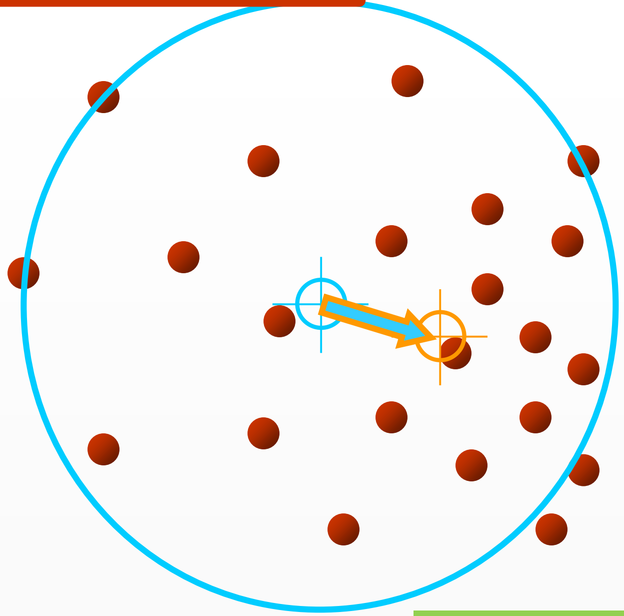
Mean shift & nonparametric density analysis

$$\nabla P(\mathbf{x}) = \frac{c}{n}\sum_{i=1}^{n}\nabla k_i = \frac{c}{n}\left[\sum_{i=1}^{n} g_i\right] \square \boxed{\left[\frac{\sum_{i=1}^{n}\mathbf{x}_i g_i}{\sum_{i=1}^{n} g_i} - \mathbf{x}\right]} \rightarrow$$

$m(x)$ mean shift vector

$$\nabla P(x) = \frac{c}{n}\sum_{i=1}^{n} g_i \times m(x)$$

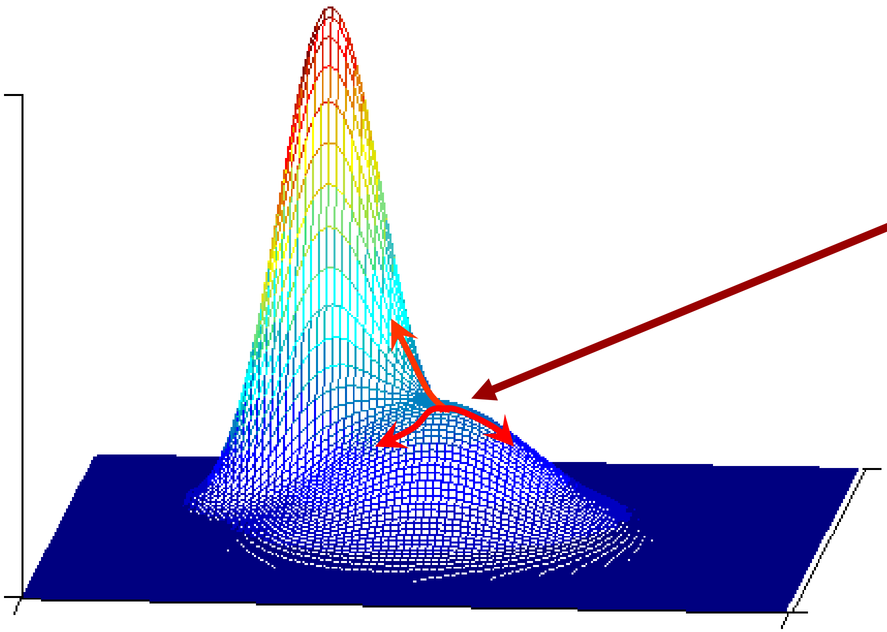
$$m(x) = \frac{\nabla P(x)}{\frac{c}{n}\sum_{i=1}^{n} g_i}$$



$$g(\mathbf{x}) = -k'(\mathbf{x})$$

Main theoretic result: Mean shift vector is proportional to the gradient of the *nonparametric pdf*, therefore it is appropriate for mode seeking

Mean shift mode detection



What happens if we reach a saddle point **?**

Perturb the mode position and check if we return back

Updated Mean Shift Procedure:

- Find all modes using the Simple Mean Shift Procedure
- Prune modes by perturbing them (find saddle points and plateaus)
- Prune nearby – take highest mode in the window

Mean-Shift Clustering

- Main steps:
  1. A density estimation window (e.g. a Gaussian window) is placed on each sample point.
  2. Within each window the mean shift vector is calculated, which points toward the maximum density:

$$m_h(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)} - x$$

where …

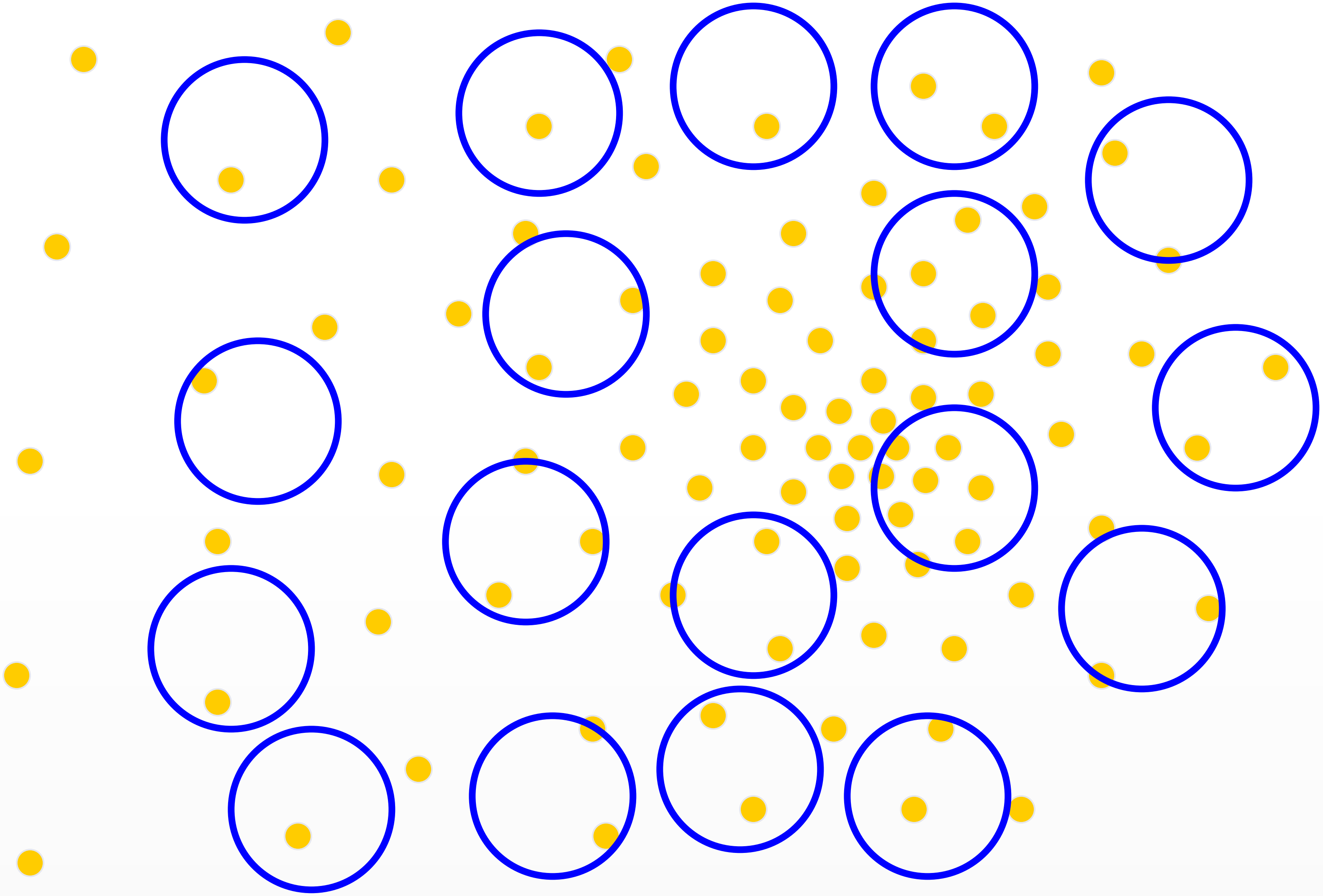
x is a *d* dimensional feature point,

g(x)=-K'(x), where ***K*** is a kernel function (e.g. Gaussian kernel)

h is the bandwidth parameter of the kernel

  3. The window is shifted with the mean shift vector.
  4. Step 2 and 3 are repeated until convergence to a local density maximum.
  5. The sample points that converged to the same local maximum will belong to the same cluster.

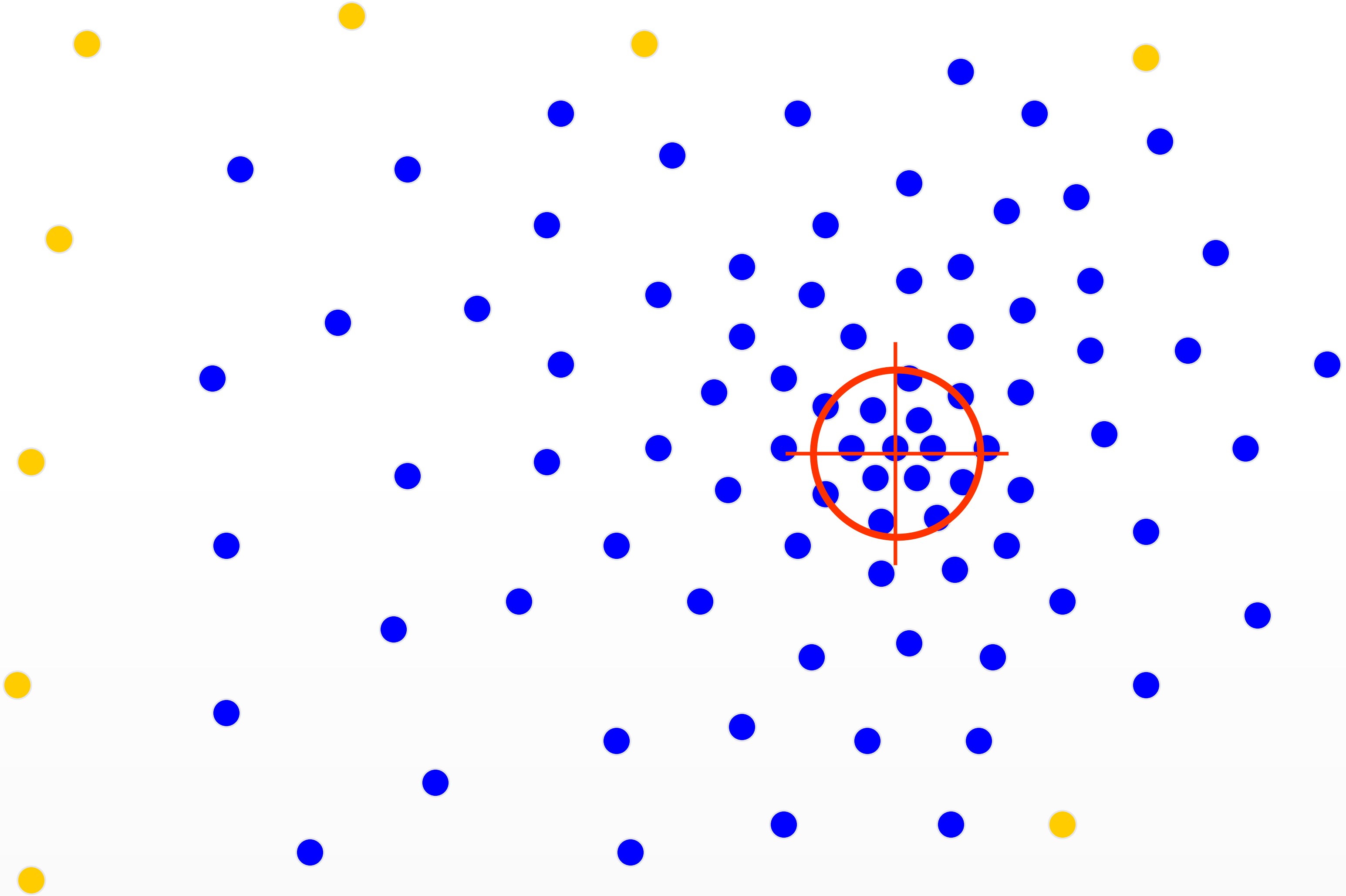
Real Modality Analysis



Tessellate the space with windows

Run the procedure in parallel

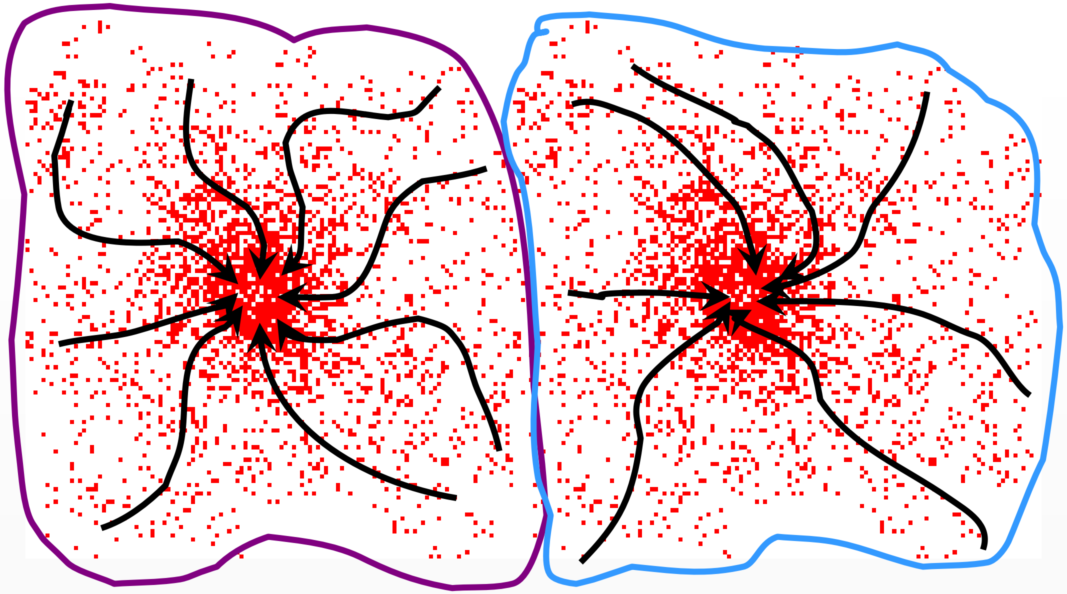
Real Modality Analysis



The blue data points were traversed by the windows towards the mode

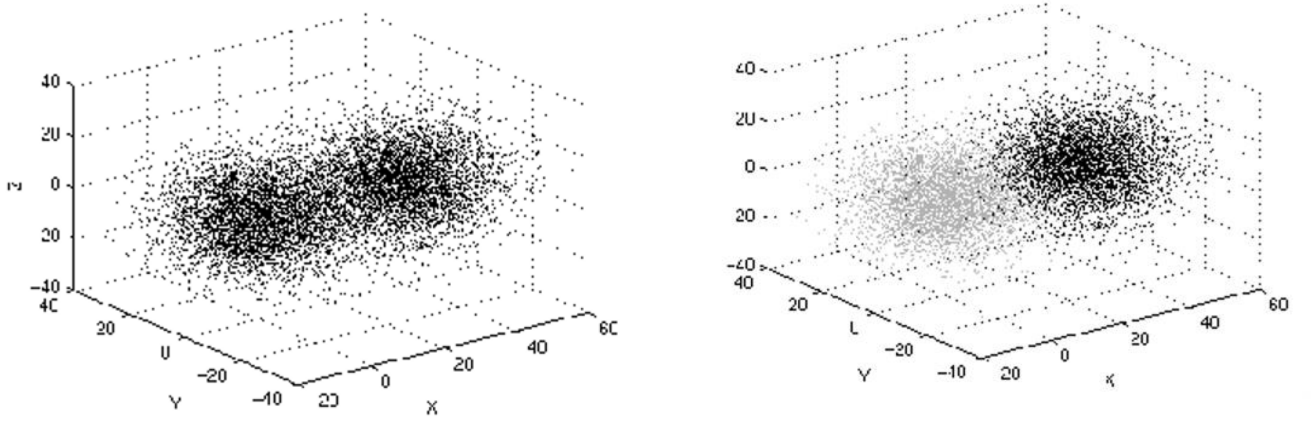
Attraction basin

- Attraction basin: the region for which all trajectories lead to the same mode
- Cluster: all data points in the attraction basin of a mode

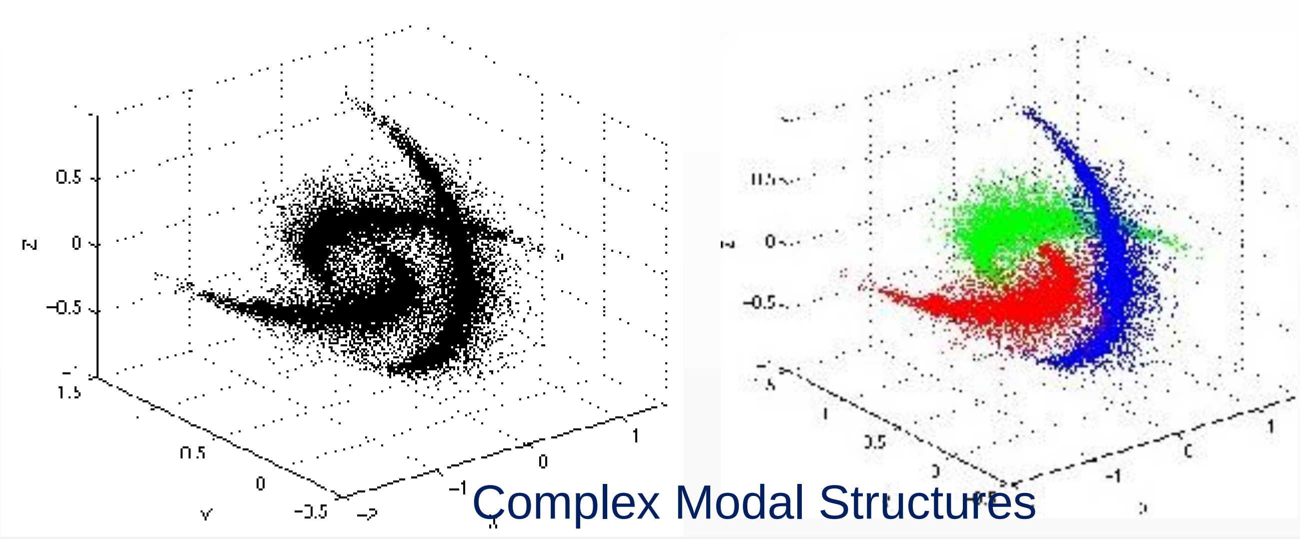


Slide by Y. Ukrainitz & B. Sarel

Clustering
Synthetic Examples



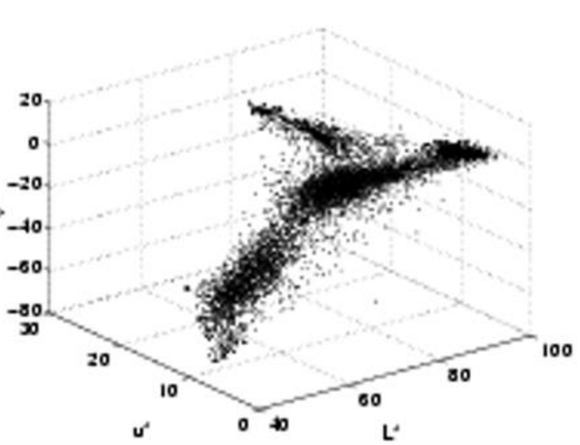
Simple Modal Structures

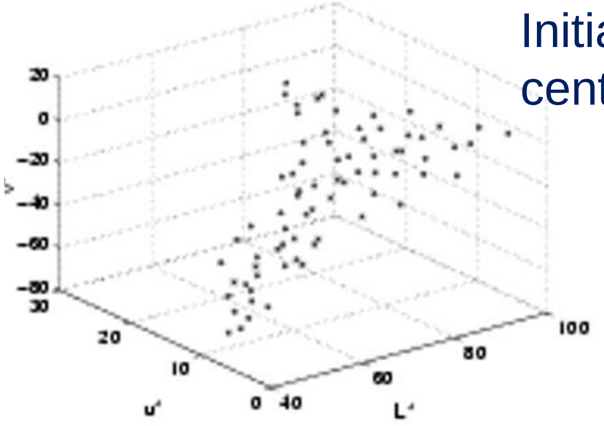


Complex Modal Structures

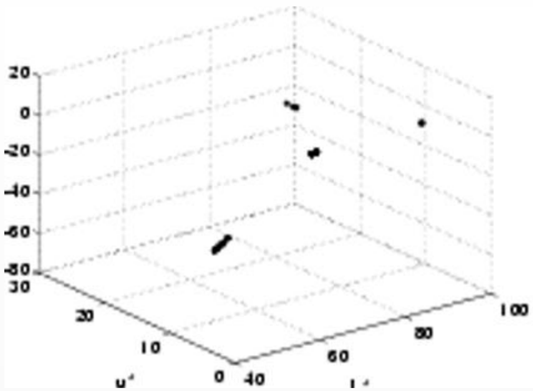
Clustering
Real example

Feature space:
L*u*v representation

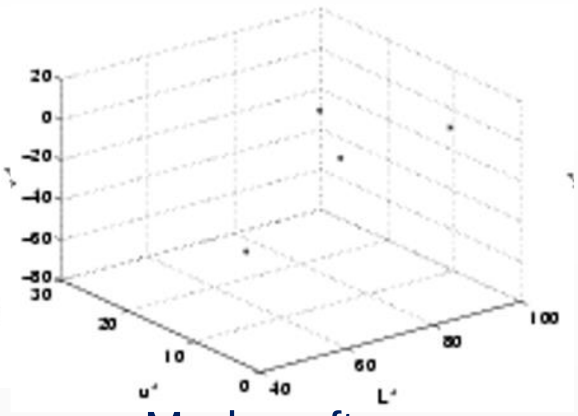




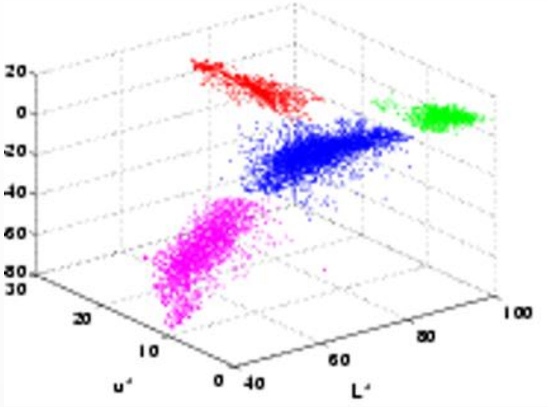
Initial window centers



Modes found

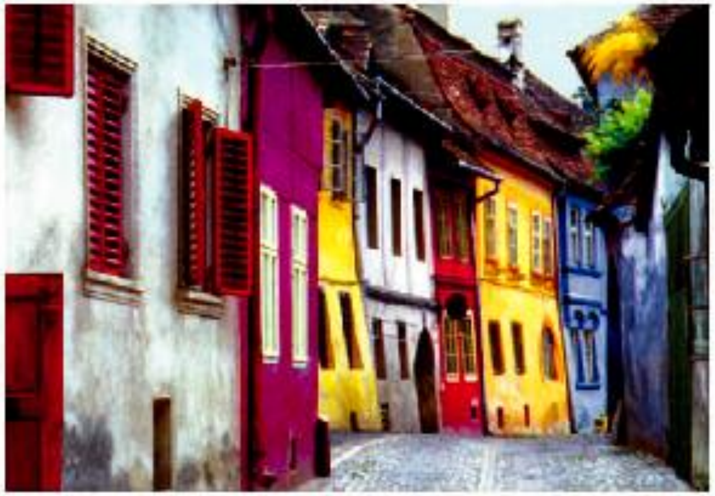


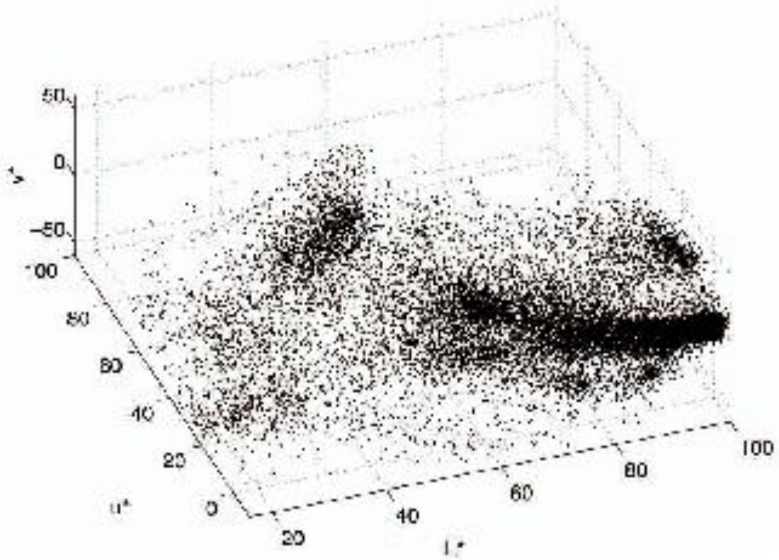
Modes after pruning



Final clusters

Clustering
Real example



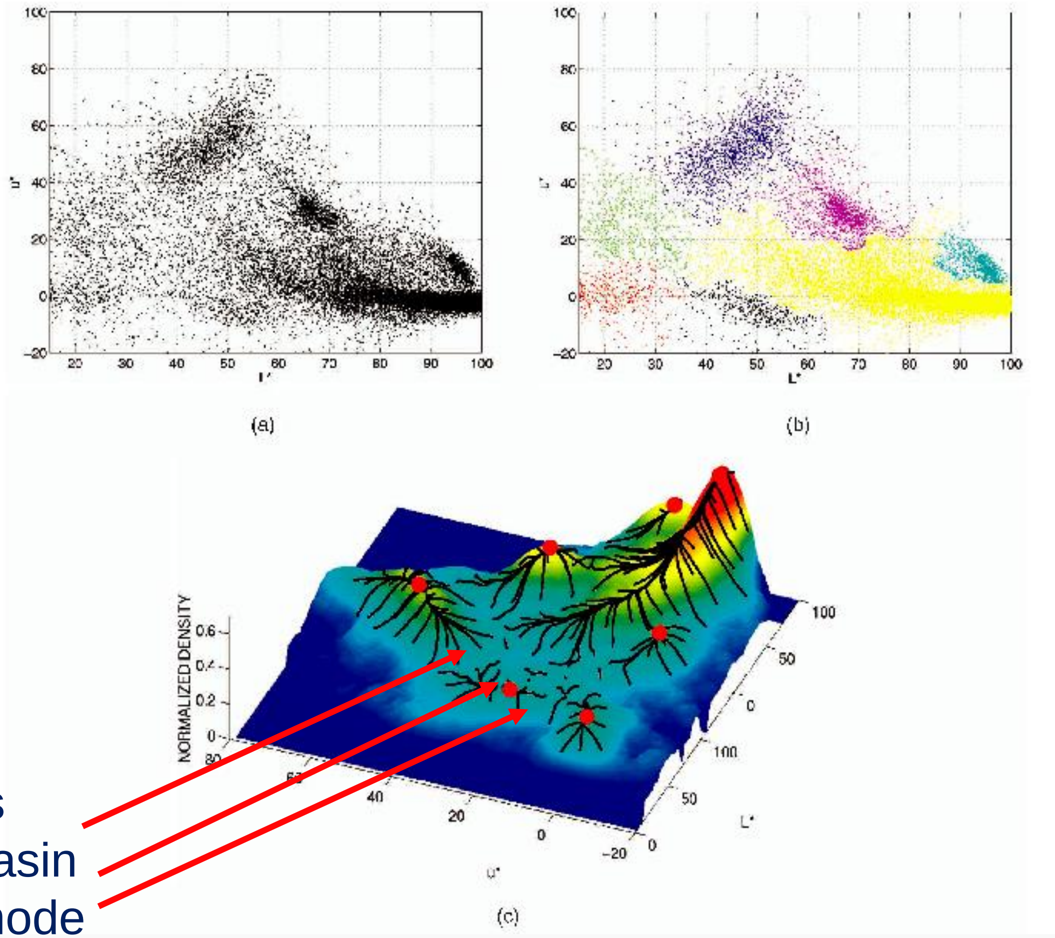


L*u*v space representation

Clustering
Real example

2D (L*u)
space
representation

Final clusters



(a)

(b)

(c)

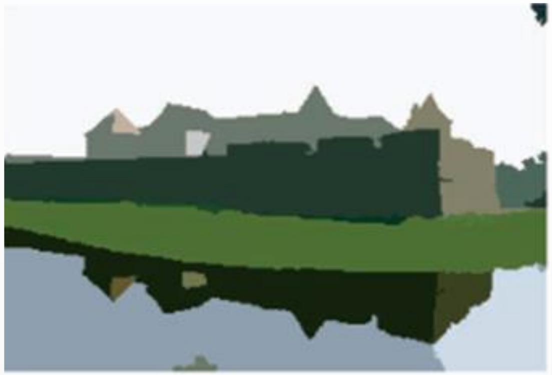
Not all trajectories
in the attraction basin
reach the same mode

Mean shift for image segmentation

- Segmented regions
  - Similar color/texture values
  - Spatially connected pixels
- Grayscale image segmentation model
  - Each pixel = a „billiard ball” $\boldsymbol{x}$ in the 3D joint spatial-intensity space:

    $$\boldsymbol{x} = [x, y, z(x, y)\ ] \in \mathbb{R}^3$$

    where $z(x,y)$ is the gray level of pixel (x,y)
  - *Segmentation:* find the modes of this 3D distribution – i.e. dense regions with their attraction basins



Discontinuity Preserving Smoothing

Feature space : Joint domain = spatial coordinates + color space

$$K(\mathbf{x}) = C \cdot k_s\left(\left\|\frac{\mathbf{x}^s}{h_s}\right\|\right) \cdot k_r\left(\left\|\frac{\mathbf{x}^r}{h_r}\right\|\right)$$

Meaning : treat the image as data points in the spatial and gray level domain:

$\boldsymbol{x} = [x^s, x^r] = [x, y, z(x,y)\,] \in \mathbb{R}^3$ where $z(x,y)$ is the gray level of pixel (x,y)

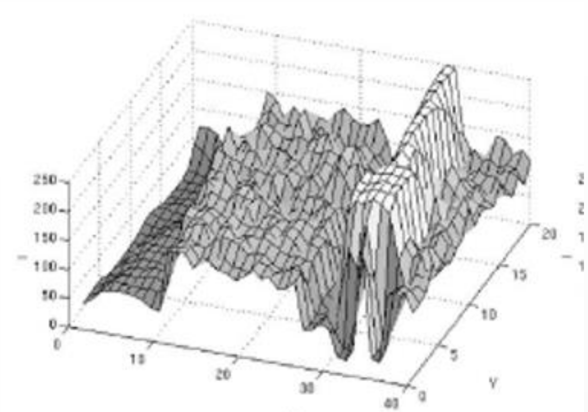
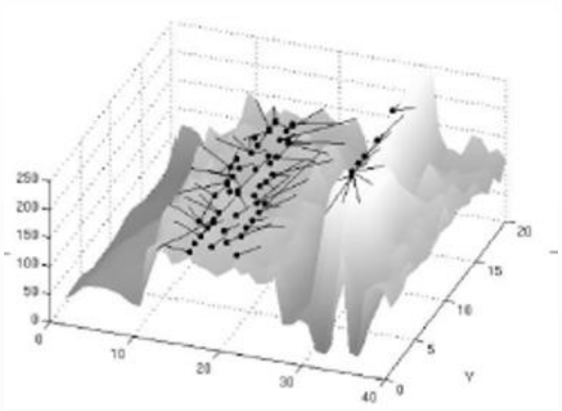
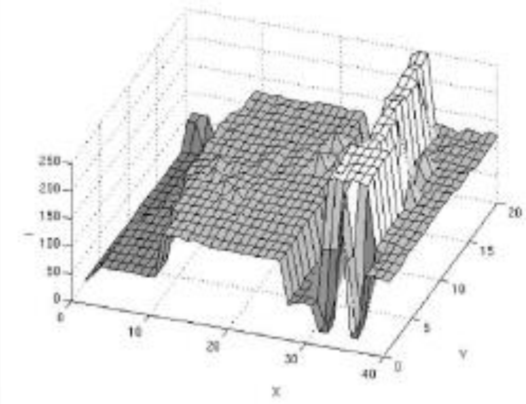


Image Data (slice)

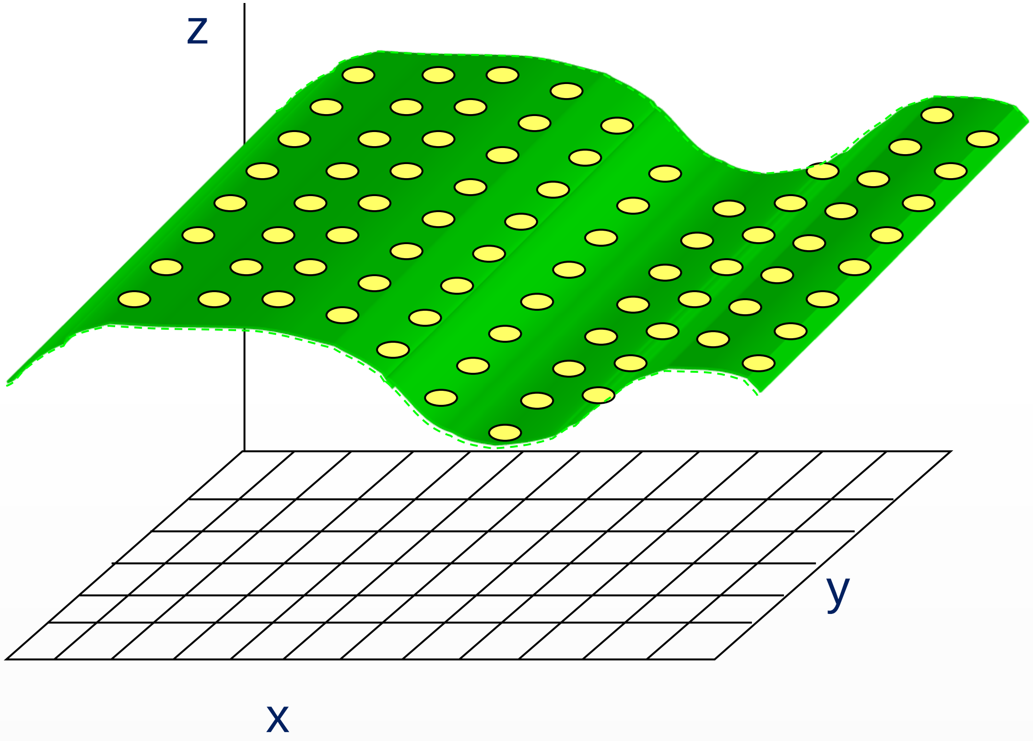


Mean Shift vectors



Smoothing result

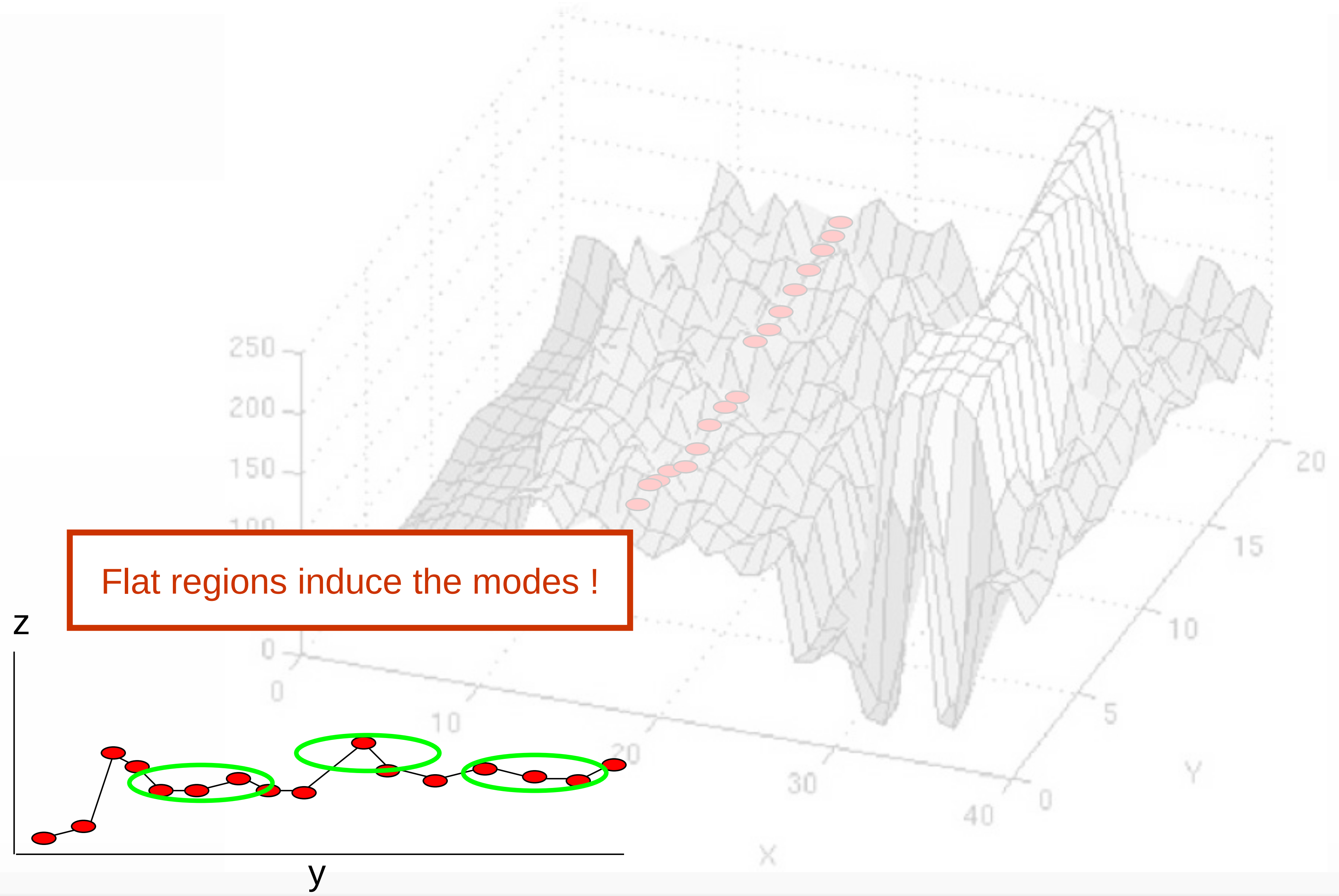
Discontinuity Preserving Smoothing



The image gray levels…

… can be viewed as data points in the *x*, *y*, *z* space (joined spatial and color space)

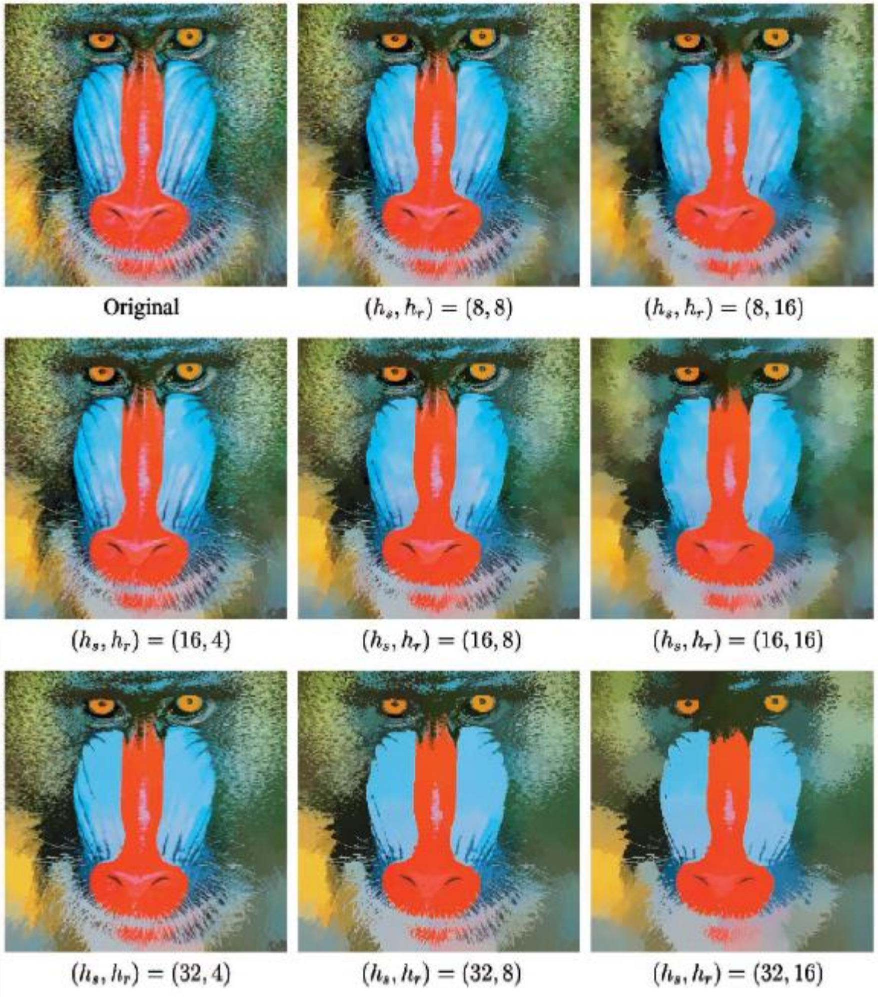
Discontinuity Preserving Smoothing



Flat regions induce the modes !

Discontinuity Preserving Smoothing

- The effect of window size in spatial and range spaces



Original

$(h_s, h_r) = (8, 8)$

$(h_s, h_r) = (8, 16)$

$(h_s, h_r) = (16, 4)$

$(h_s, h_r) = (16, 8)$

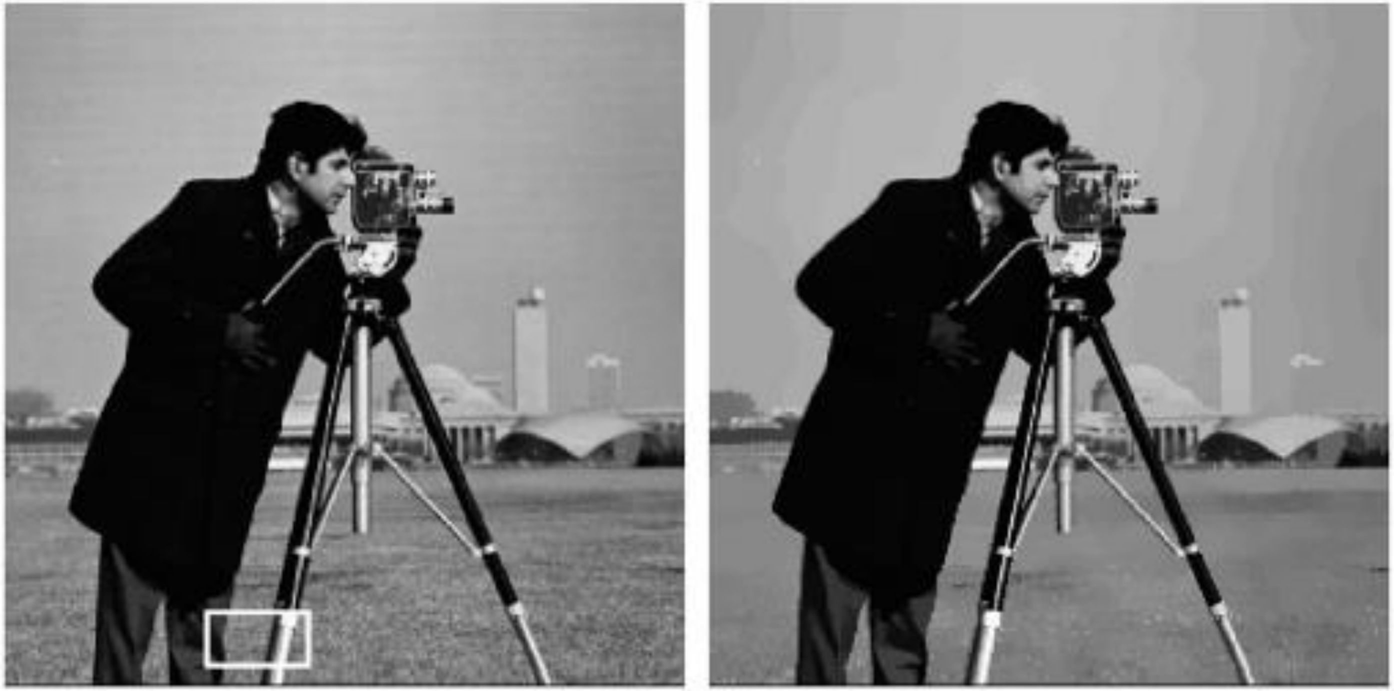
$(h_s, h_r) = (16, 16)$

$(h_s, h_r) = (32, 4)$

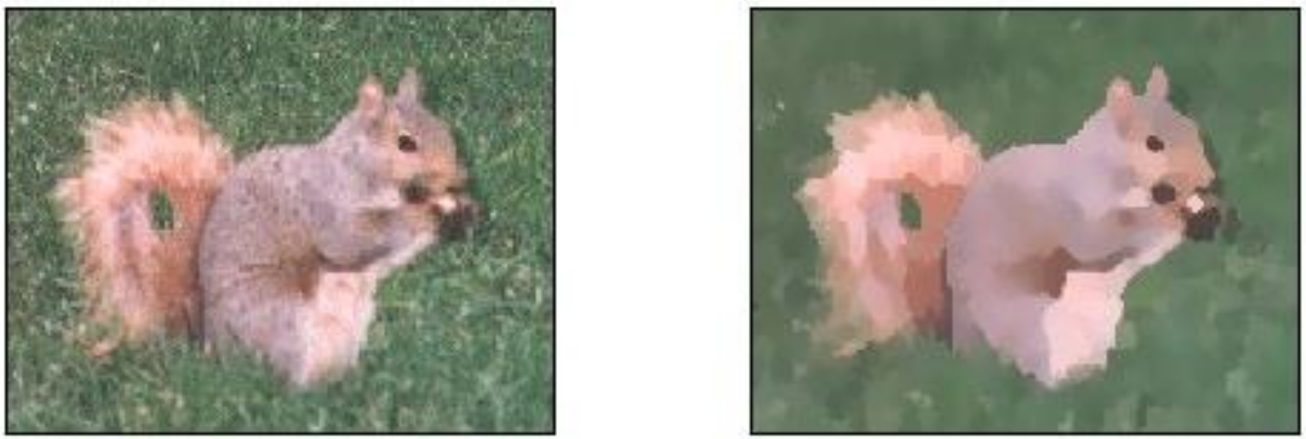
$(h_s, h_r) = (32, 8)$

$(h_s, h_r) = (32, 16)$

Discontinuity Preserving Smoothing



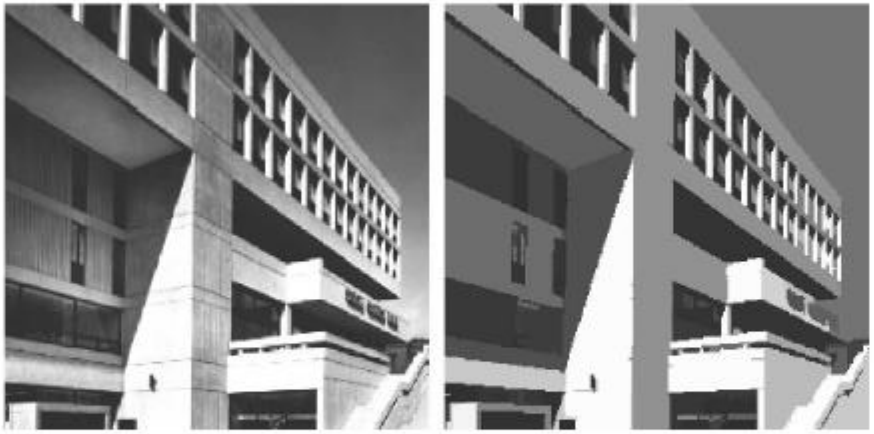
Discontinuity Preserving Smoothing

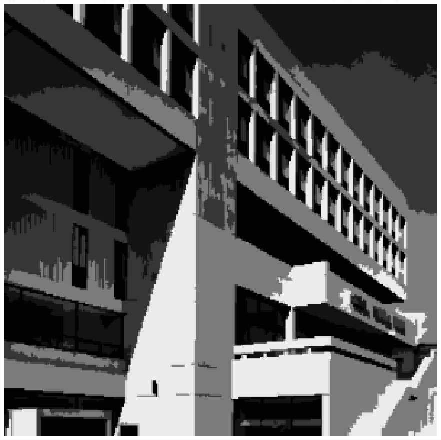


Segmentation

- Segment = Cluster, or Cluster of Clusters
- Algorithm:
  - Run Filtering (discontinuity preserving smoothing)
  - Cluster the clusters which are closer than window size

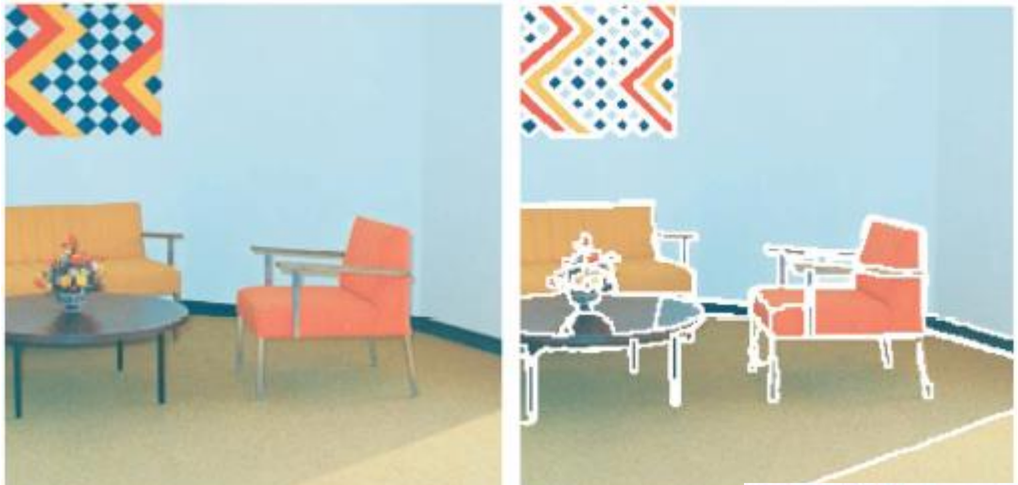
Segmentation

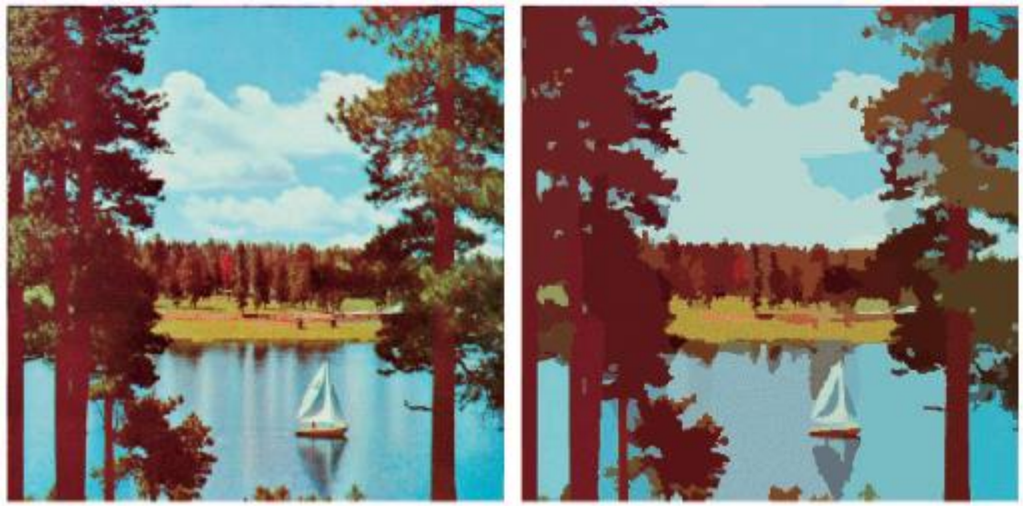




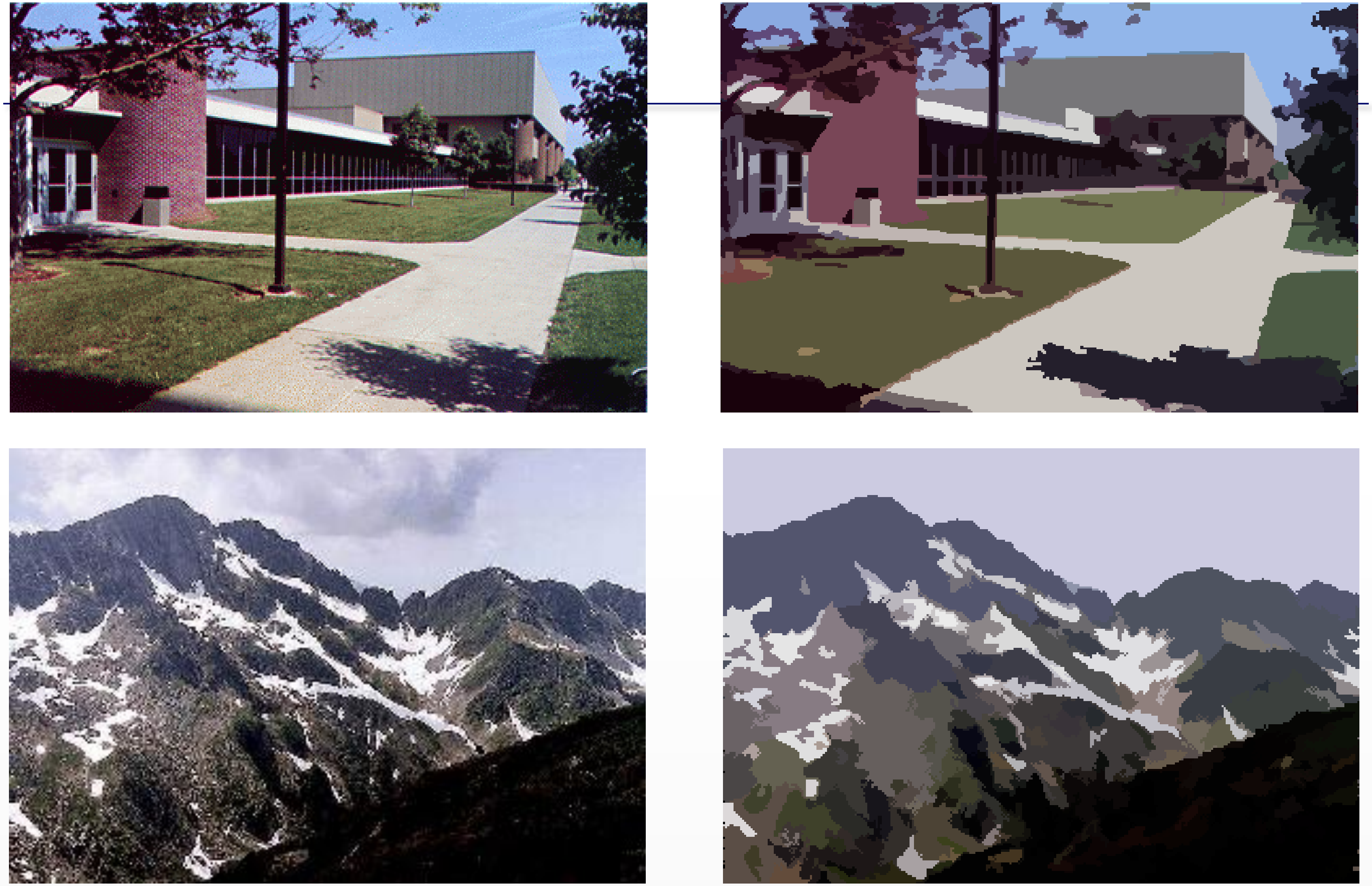
…when feature space is only gray levels…

Segmentation

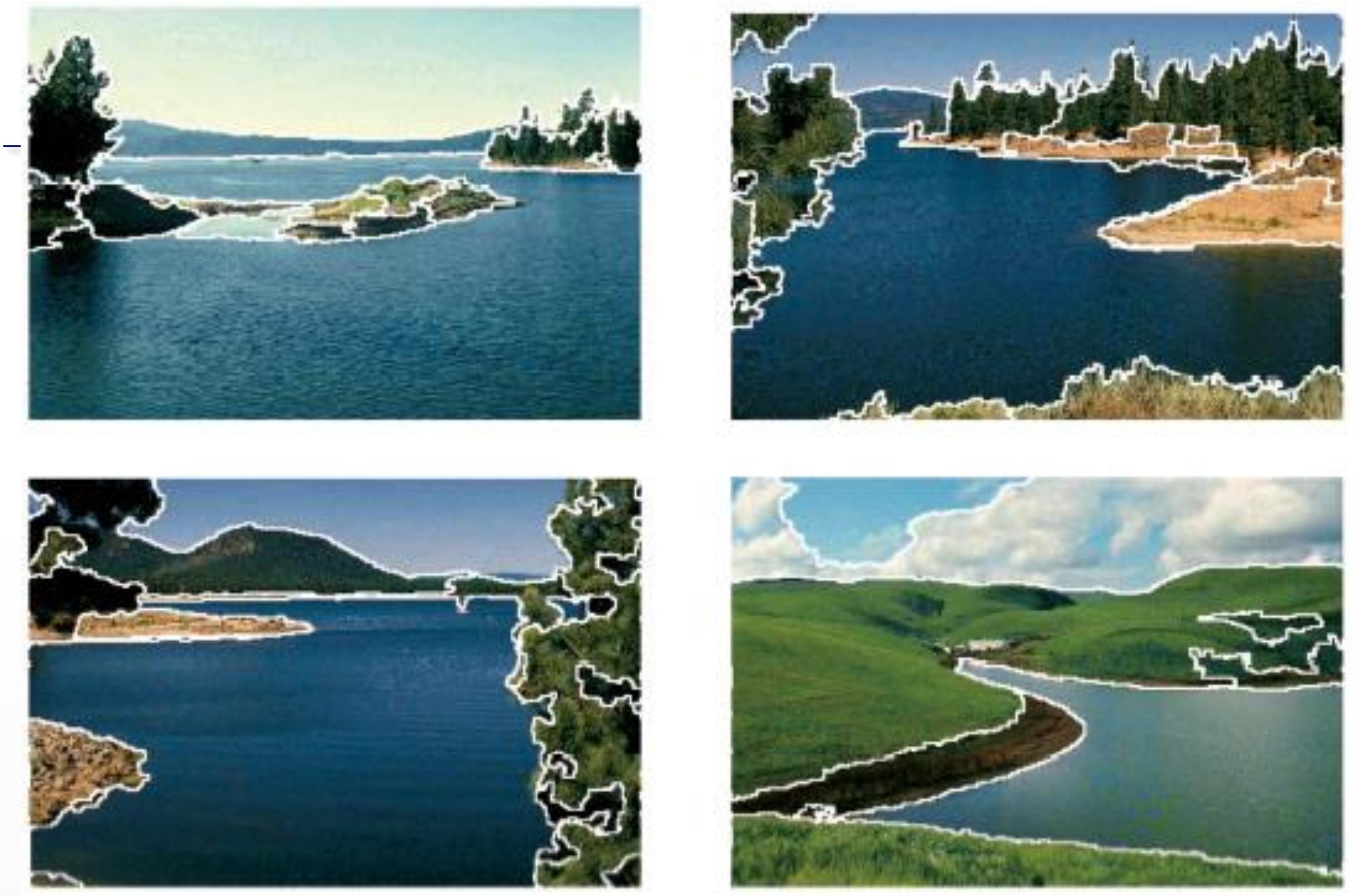




Mean shift segmentation results



http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html



http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html

Mean-shift: other issues

- Speedups
  - Uniform kernel (much faster but not as good)
  - Binning or hierarchical methods
  - Approximate nearest neighbor search
- Methods to adapt kernel size depending on data density
- Lots of theoretical support

  D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

Mean shift pros and cons

- Pros
  - Good general-practice segmentation
  - Finds variable number of regions
  - Robust to outliers
- Cons
  - Have to choose kernel size in advance
  - Original algorithm doesn’t deal well with high dimensions
- When to use it
  - Oversegmentatoin
  - Multiple segmentations
  - Other tracking and clustering applications

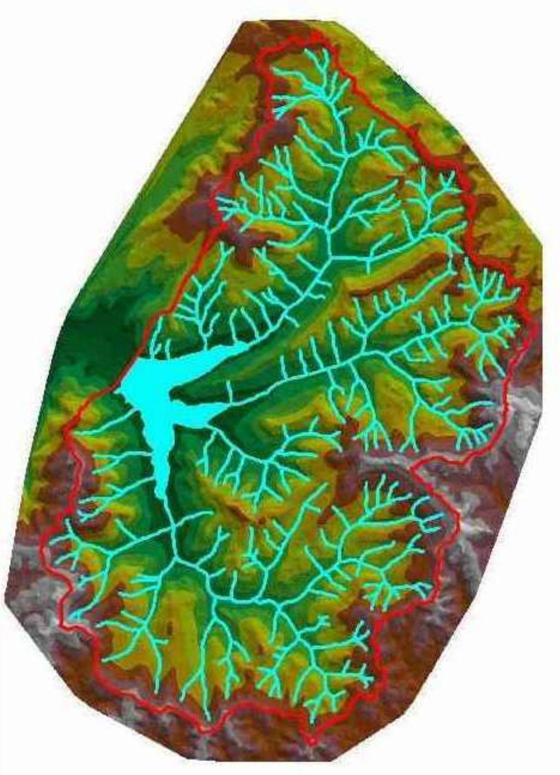
Basic Image Processing Algorithms

PPKE-ITK

Lecture 9.

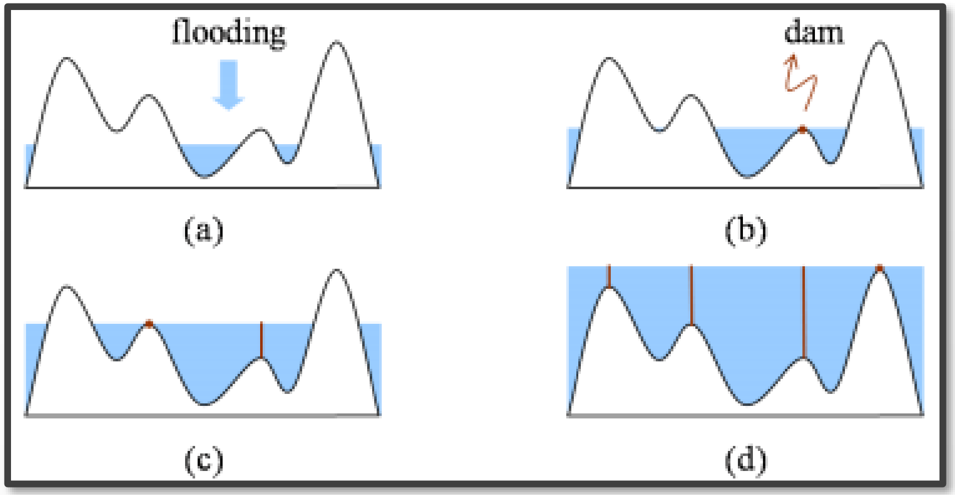
Watershed algorithm

- A mathematical morphology based approach on image segmentation



Watershed Segmentation

- A grey-level image may be seen as a topographic surface, where the grey level of a pixel is interpreted as its altitude in the surface.
- The goal of the algorithm is to find the „watersheds” that are separating the „catchment basins” from each other.



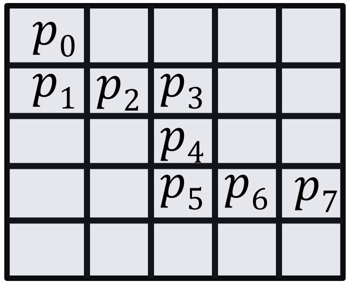
Concept of the watershed algorithm*

*Yu-Hsiang Wang: „Tutorial: Image Segmentation” (http://disp.ee.ntu.edu.tw/meeting/%E6%98%B1%E7%BF%94/Segmentation%20tutorial.pdf)

Watershed-Basic Definitions

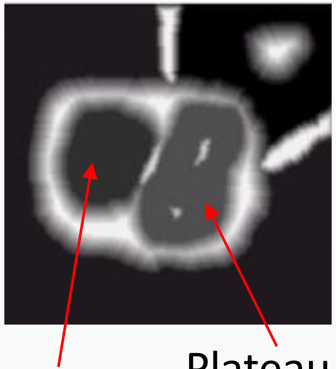
- I: 2D gray level image

- Path P of length ℓ between p and q in I
  - A $(\ell + 1)$-tuple of pixels $(p_0 = p, p_1, \dots, p_\ell = q)$ such that p_i, p_{i+1} are adjacent (4 adjacent, 8 adjacent, or m adjacent)
  - $\ell(P)$: the length of a given path P



4-adjacent path P with $\ell(P)$=6

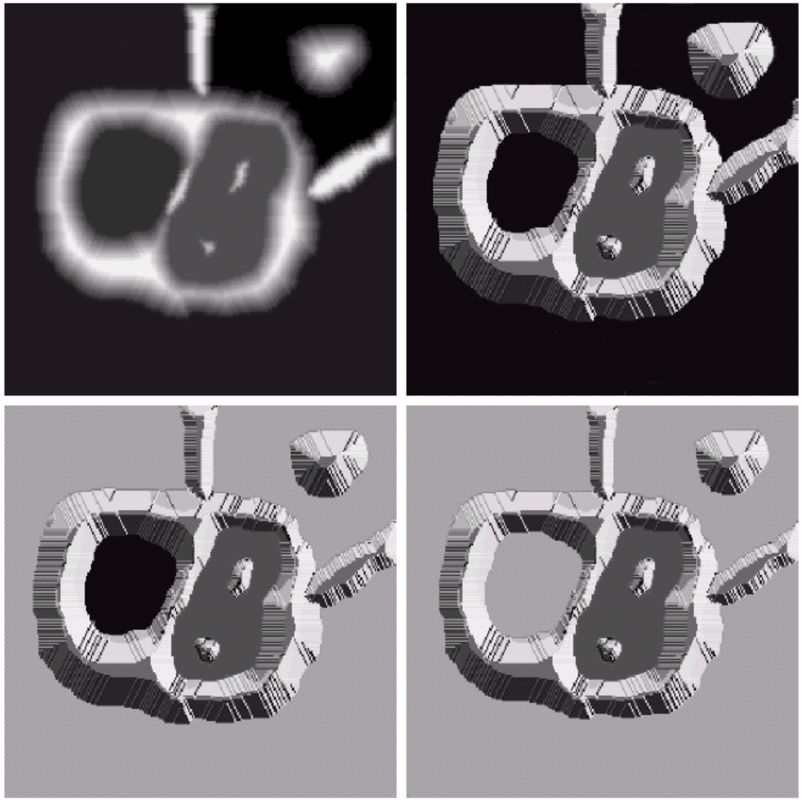
- Minimum
  - A minimum M of I is a connected plateau of pixels from which it is impossible to reach a point of lower altitude without having to climb



Plateau M_1 Plateau M_2

Basic Steps

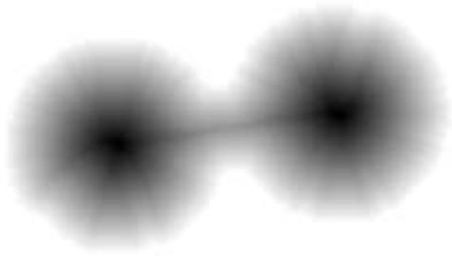
- Piercing holes in each regional minimum of I
- The 3D topography is flooded from below gradually
- When the rising water in distinct catchment basins is about to merge, a **dam** is built to prevent the merging

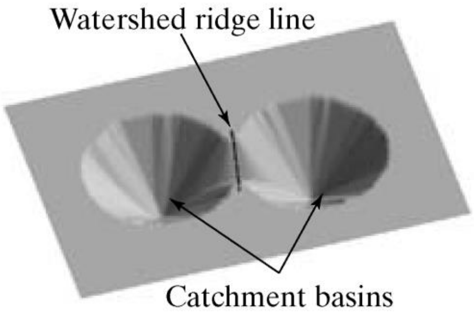


- Instead of working on an image itself, this technique is often applied on its gradient image.

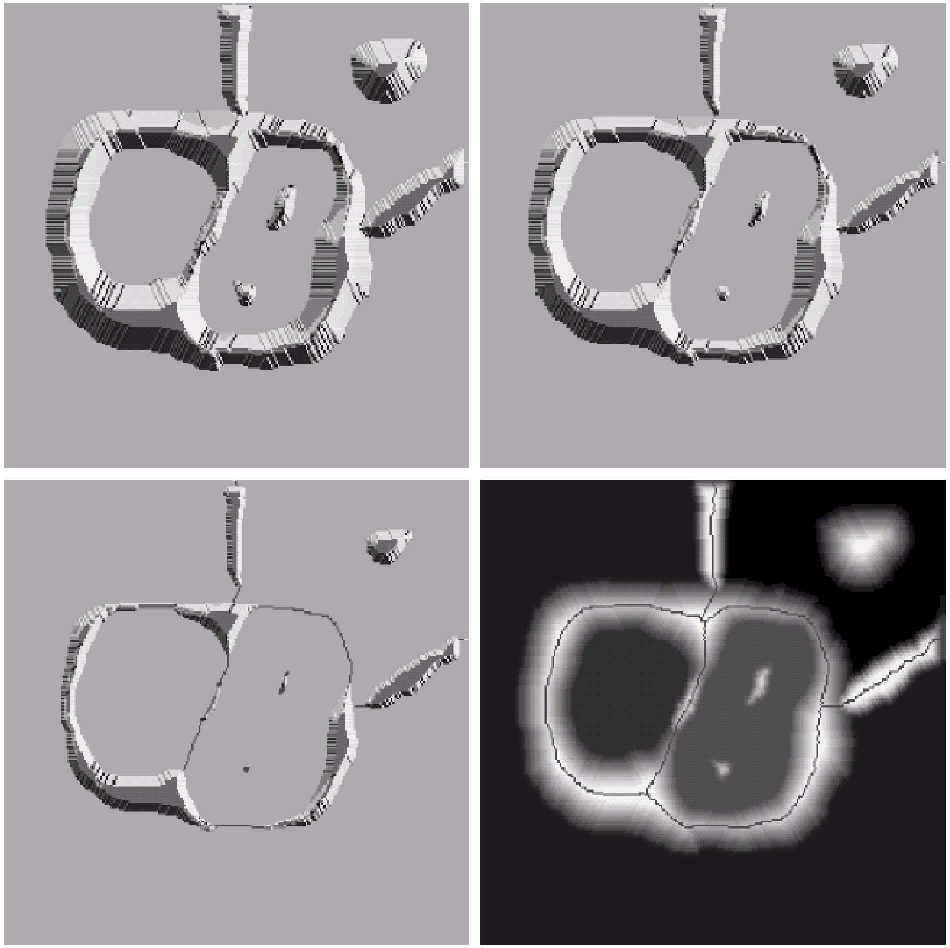
Watershed-Basic Definitions

- Three types of points
  - Points belonging to a regional minimum
  - Catchment basin / watershed of a regional minimum
    - Points at which a drop of water will certainly fall to a single minimum
  - Divide lines / Watershed lines
    - Points at which a drop of water will be equally likely to fall to more than one minimum
    - Crest lines on the topographic surface
- This technique is to identify all the third type of points for segmentation





Basic Steps



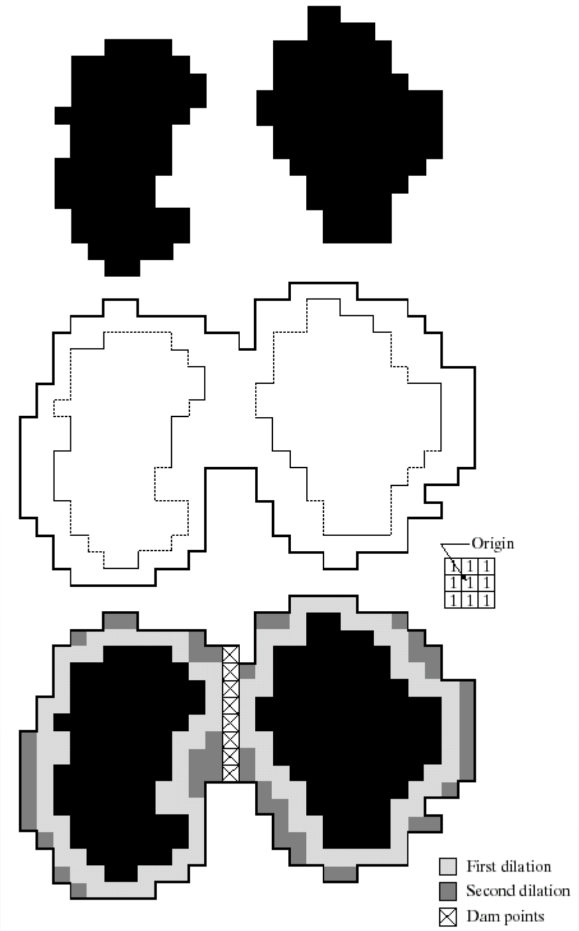
- The dam boundaries correspond to the watershed lines to be extracted by a water-shed segmentation al-gorithm
  - Eventually only const-ructed dams can be seen from above

Dam Construction

- Based on binary morphological dilation
- At each step of the algorithm, the binary image in obtained in the following manner:
    1. Initially, the set of pixels with minimum gray level are 1, others 0.
    2. In each subsequent step, we flood the 3D topography from below and the pixels covered by the rising water are 1s and others 0s. (See previous slides)

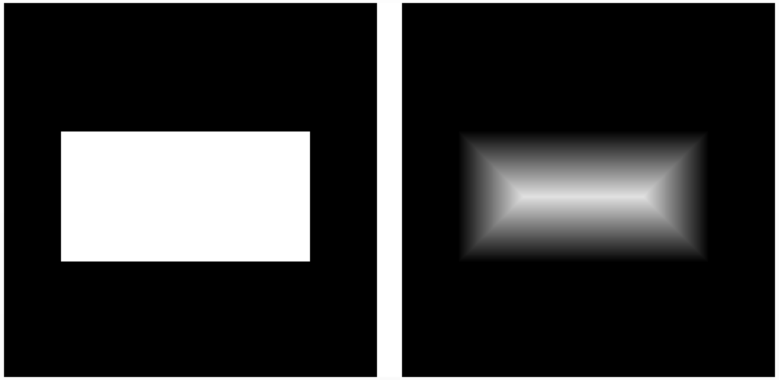
Dam Construction

- The dam is constructed by the points on which the dilation would cause the sets being dilated to merge.
  - Result: one-pixel thick connected path
  - Setting the gray level at each point in the resultant path to a value greater than the maximum gray value of the image. Usually max+1



Distance transform

- Distance transform operator:
  - Input: binary image (showing foreground/background regions)
  - Result: a graylevel image, where the graylevel intensities of points inside foreground regions are show the distance to the closest boundary from each point
  - Implementation: through morphological operations
  - Often used as input of the Watershead transform (instead of the gradient image)



https://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm

Example 1 - Watershed Transform of Binary Image Using the Distance transform

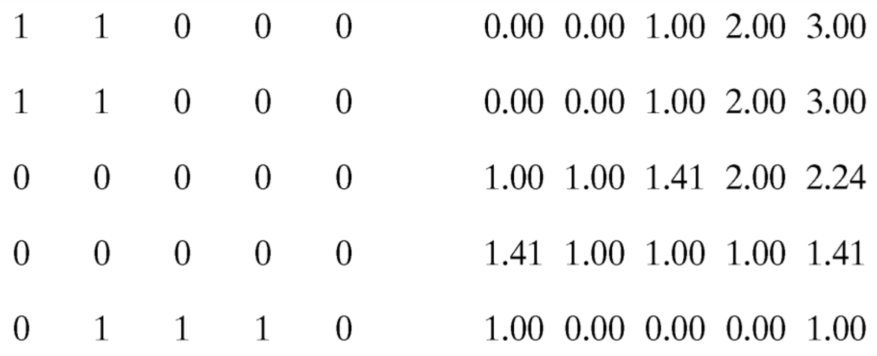
A: Original image B: Negative of image A

C: Distance transform of B D: Watershed transform of C

A | B

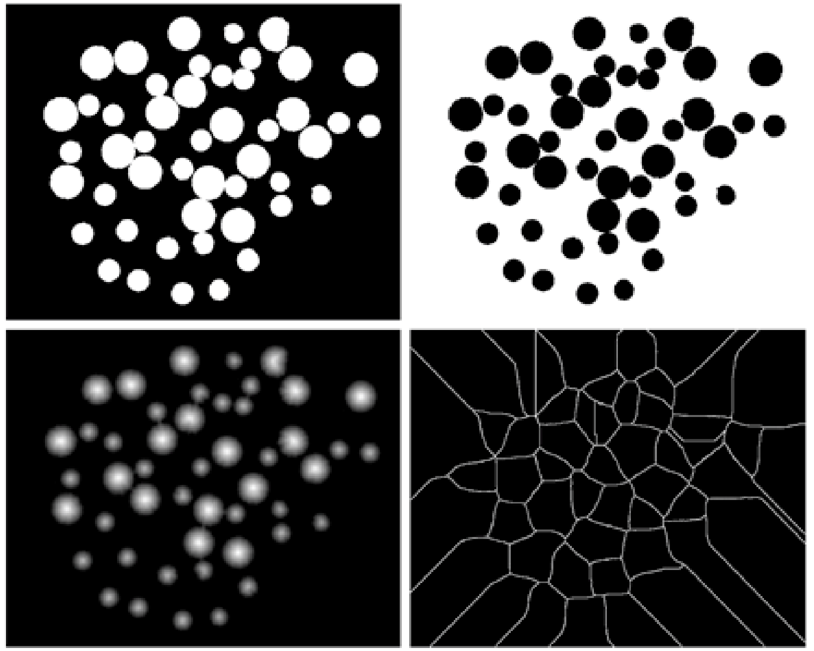
C | D

Distance transform of a binary image is defined by the distance from every pixel to the nearest non-zero valued pixel



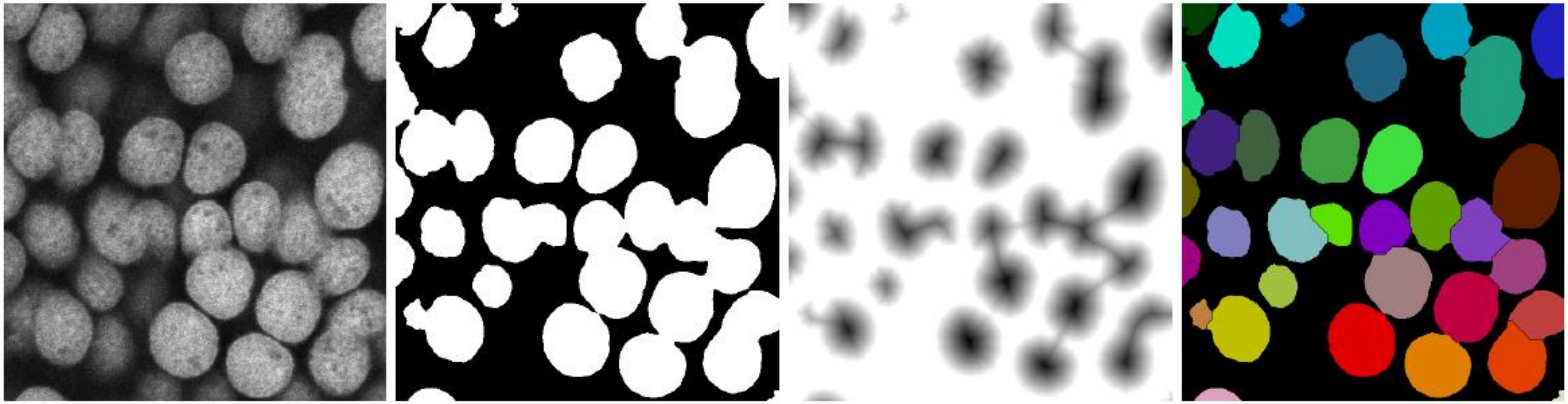
1	1	0	0	0
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	1	1	0

0.00	0.00	1.00	2.00	3.00
0.00	0.00	1.00	2.00	3.00
1.00	1.00	1.41	2.00	2.24
1.41	1.00	1.00	1.00	1.41
1.00	0.00	0.00	0.00	1.00



Example 1 - Watershed Transform of Binary Image Using the Distance transform

- Segmentation example applying watershed to the inverse distance image using the binary mask



Examples 2 - oversegmentation

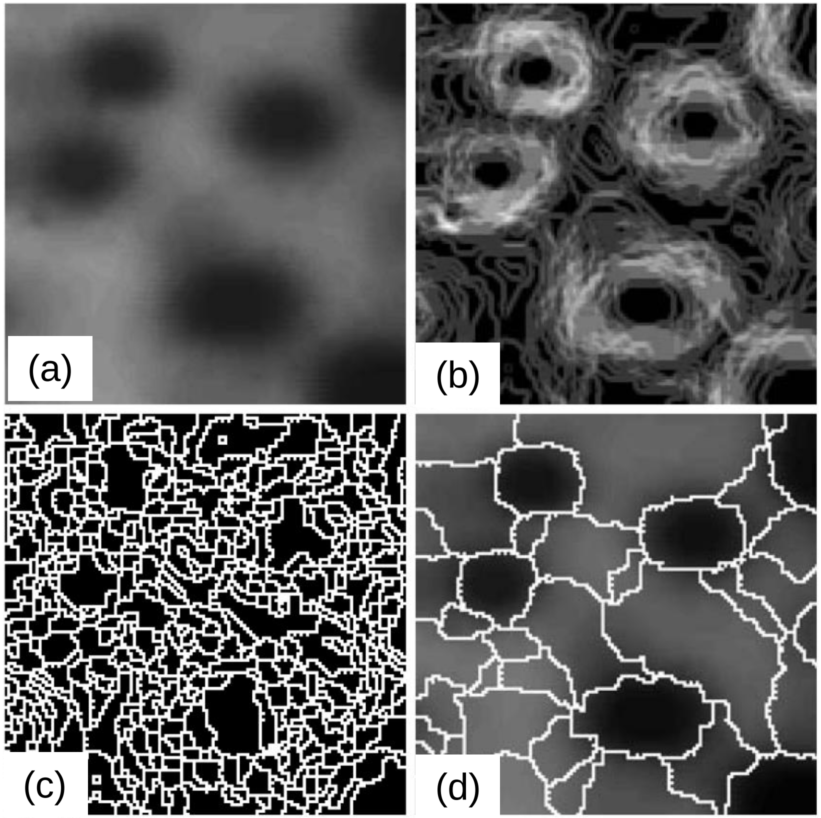
(a) Original image

(b) Gradient image of image (a)

(c) Watershed lines obtained from image b (oversegmentation)

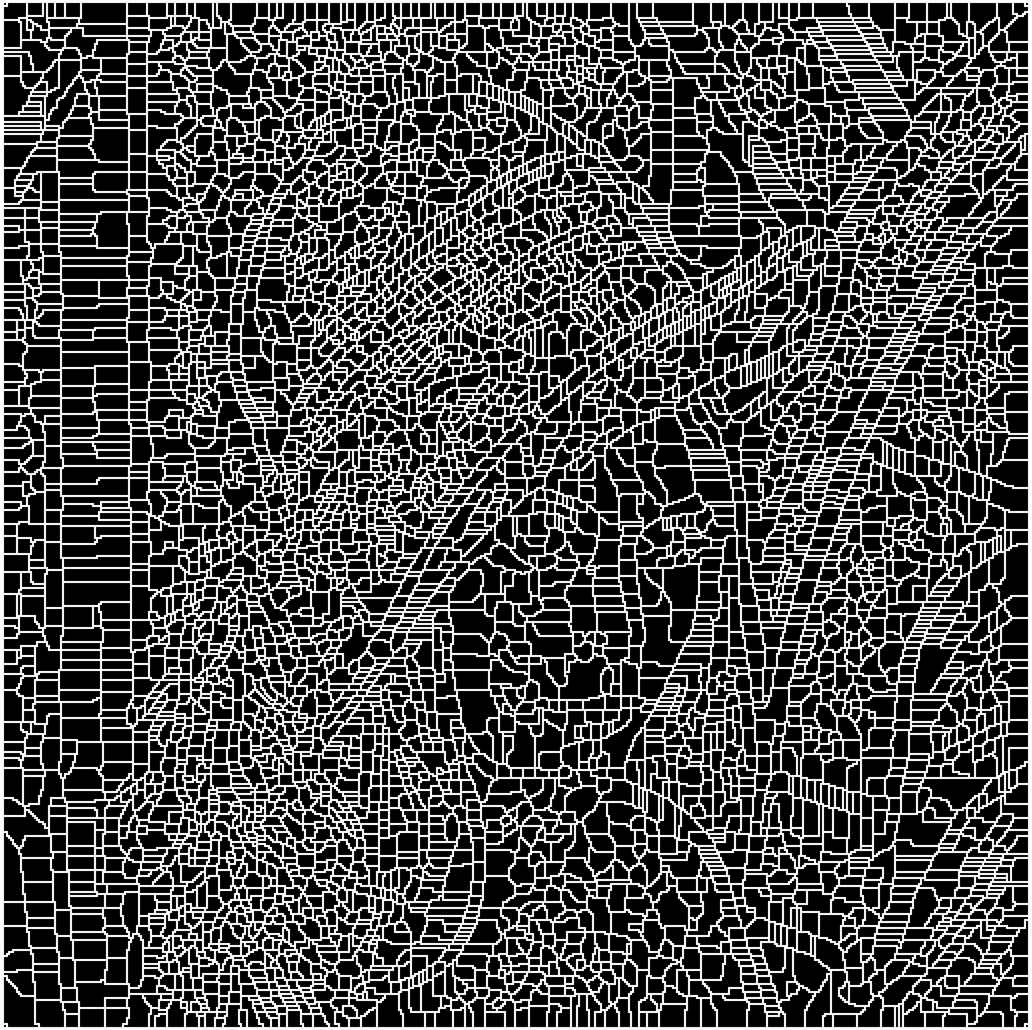
- ➔ Each connected region contains one local minimum in the corresponding gradient image

(d) Watershed lines obtained from smoothed image (b)



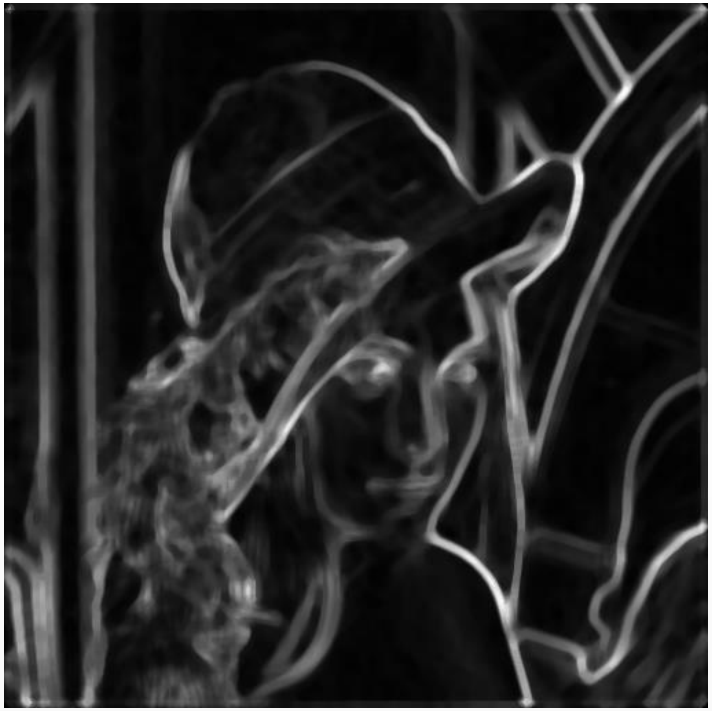
Simple trick

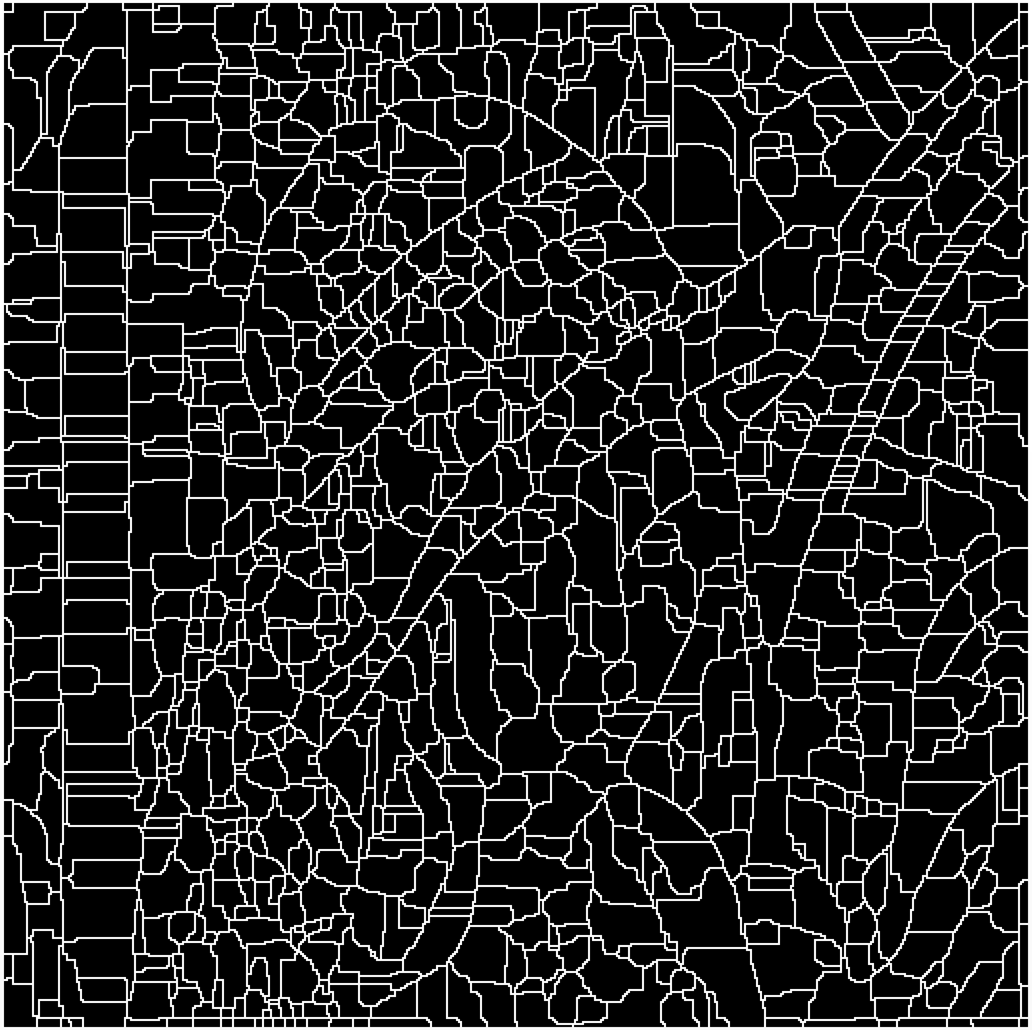
- Use median filter to reduce number of regions



Simple trick

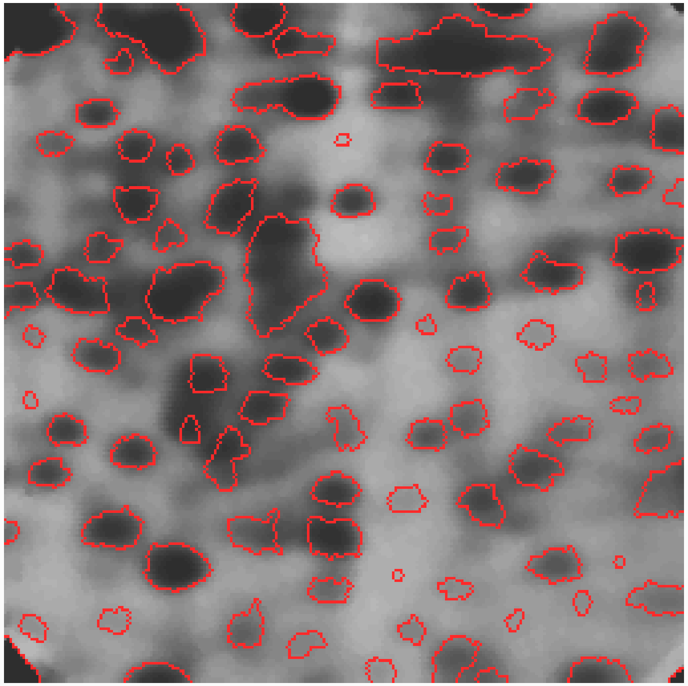
- Use median filter to reduce number of regions





Object segmentation by watershed algorithm

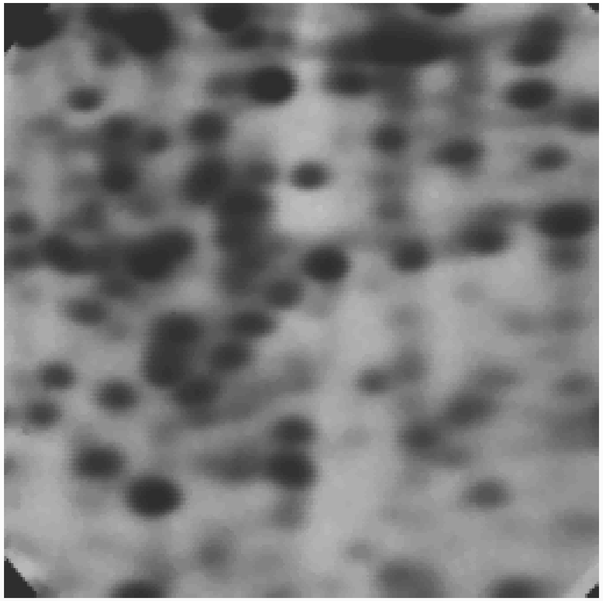
- Task: segmentation of (possibly touching) objects in front of a background



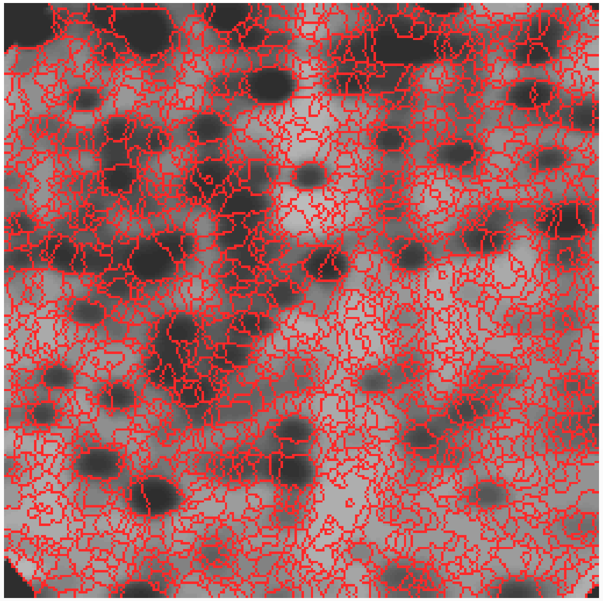
Electrophoresis image

Watershed Segmentation

- Over-segmentation problem:
  - most times the real watershed transform of the gradient present many catchment basins, each one corresponds to a minimum of the gradient that is produced by small variations, mainly due to noise.



Original Image*



Segmentation Result*

*Yu-Hsiang Wang: „Tutorial: Image Segmentation” (http://disp.ee.ntu.edu.tw/meeting/%E6%98%B1%E7%BF%94/Segmentation%20tutorial.pdf)

The Use of Markers

- Over-segmentation problem
  - Usually, we cannot overcome it with simple filtering (like median)
  - Use of markers can be a solution
- Internal markers are used to limit the number of regions by specifying the objects of interest
  - Like seeds in region growing method
  - Can be assigned manually or automatically
  - Regions without markers are allowed to be merged (no dam is to be built)
- External markers: pixels where we are confident to belong to the background
  - Watershed lines are typical external markers and they belong the same (background) region

Watershed Based Image Segmentation

- Use internal markers to obtain watershed lines of the gradient of the image to be segmented.
- Use the obtained watershed lines as external markers
- Each region defined by the external markers contains a single internal marker and part of the background
- The problem is reduced to partitioning each region into two parts: object (containing internal markers) and a single background (containing external markers)

Over-segmentation: solution

- FIRST STEP: we mark each blob of protein of the original image

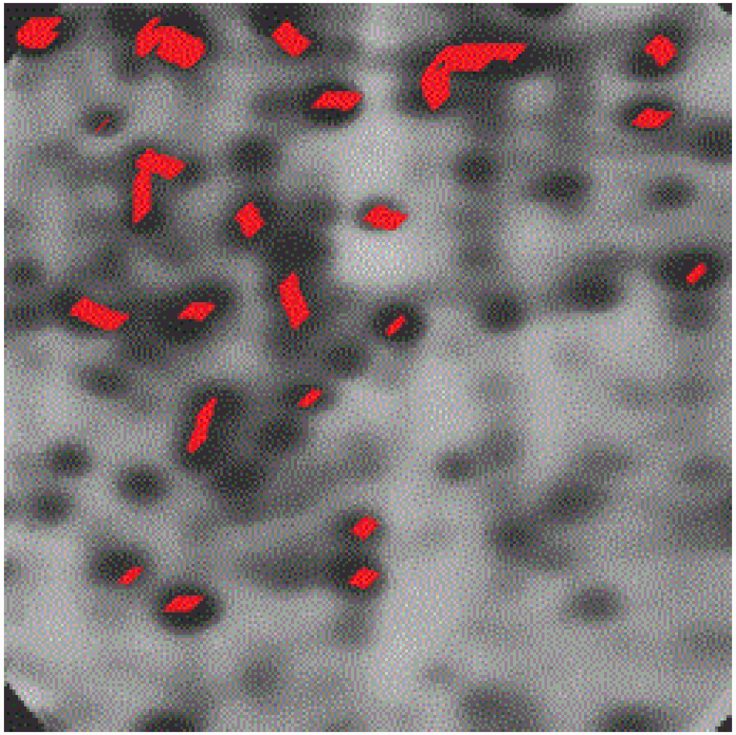
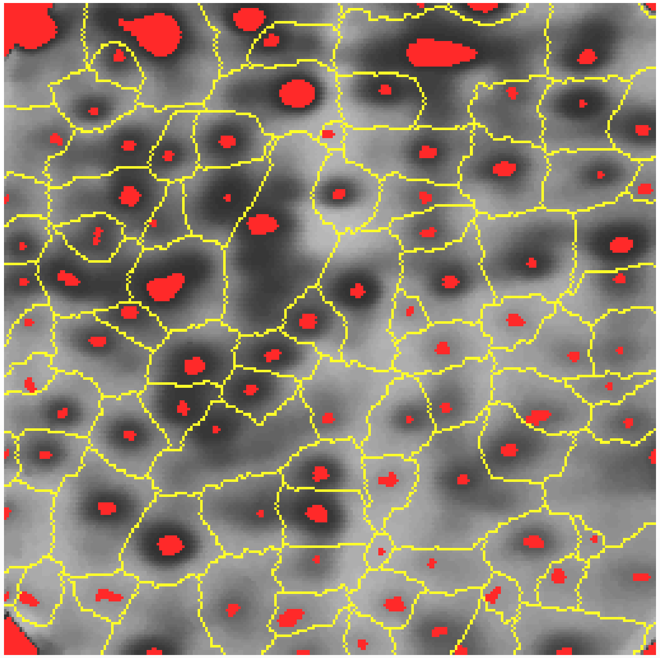


Image with a few markers (not all blobs are marked here)

Usage of internal markers

- Now we look at the final result of the marking as a topographic surface, but in the flooding process instead of piercing the minima, we only make holes through the components of the marker set that we produced

- This way the flooding will produce as many catchment basins as there are markers in M, and the watershed lines of the contours of the objects will be on the crest lines of this topographic surface



Initial image marked with the set M and the resulting watershed lines

Watershed Segmentation

- Partitioning each region into two parts: object (containing internal markers) and a single background (containing external markers)
  - Global thresholding, region growing, region splitting and merging…

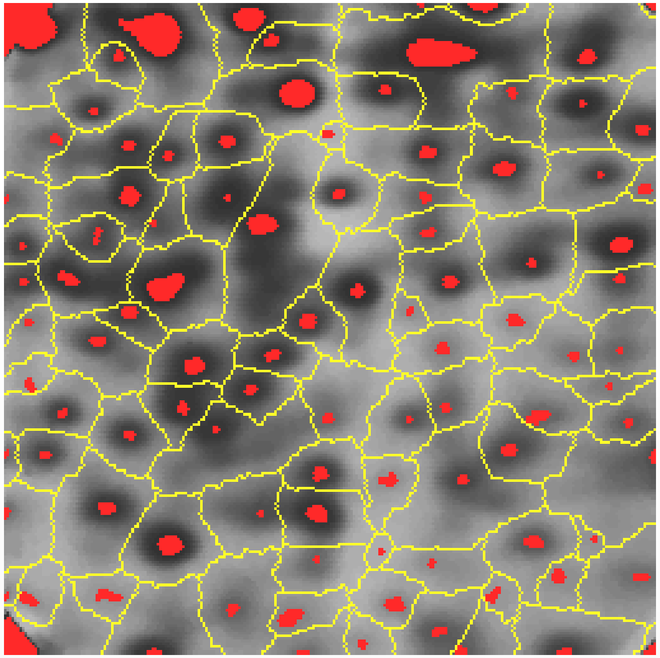
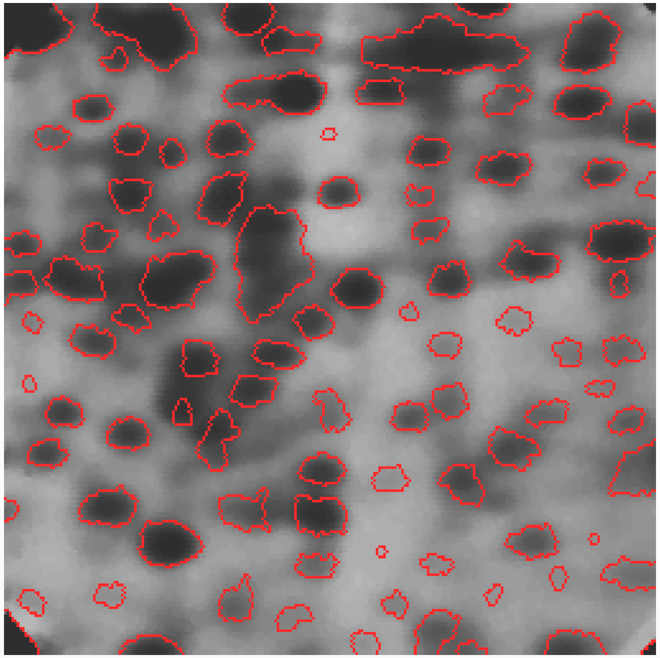


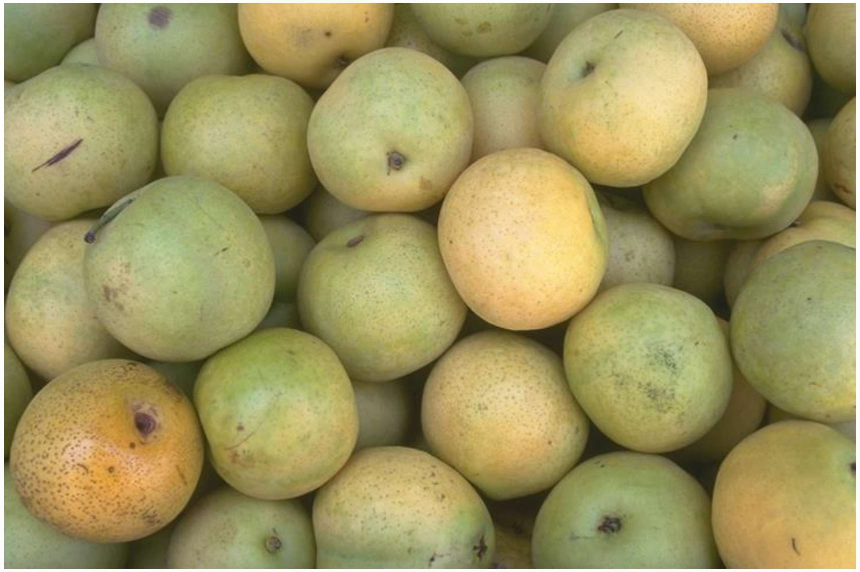
Image with internal and external markers



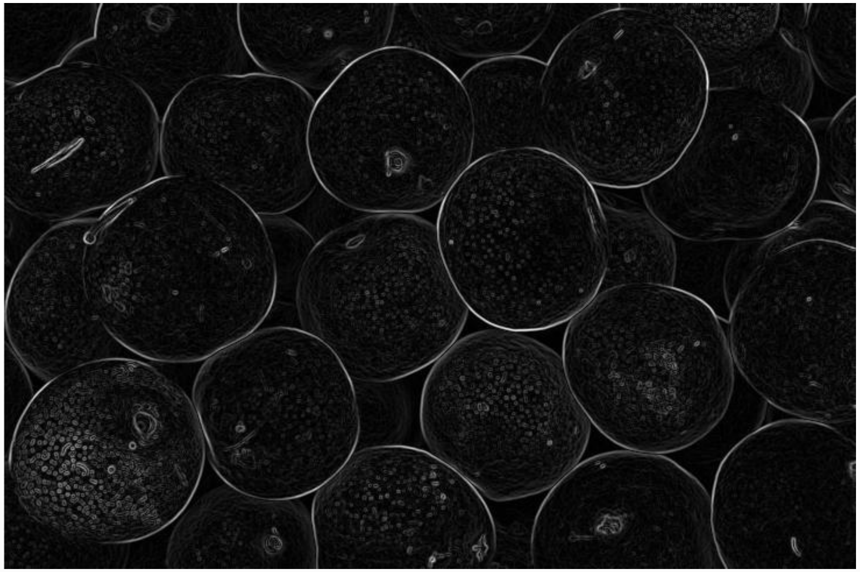
Final segmentation result

Watershed segmentation example

- Use the Gradient Magnitude as the Segmentation Function
  - The gradient is high at the borders of the objects and low (mostly) inside the objects.



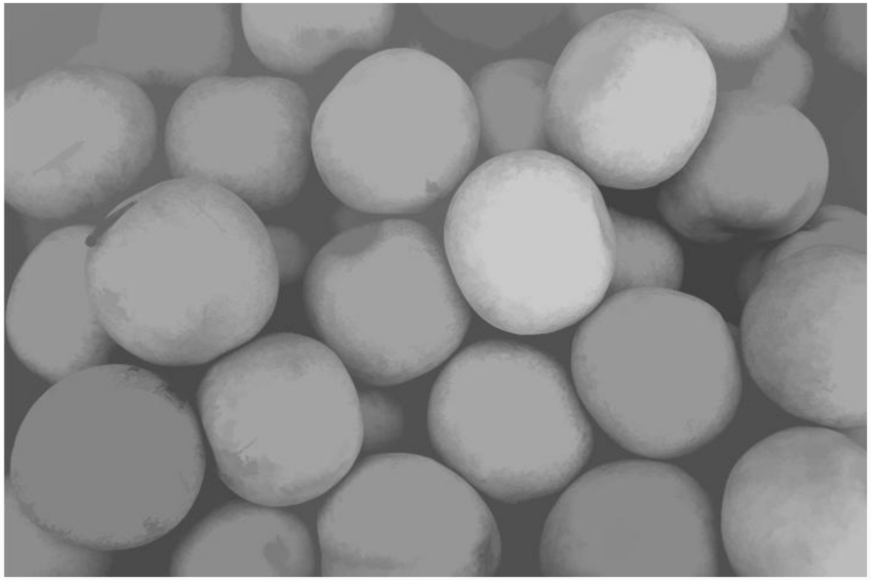
Original image (I)



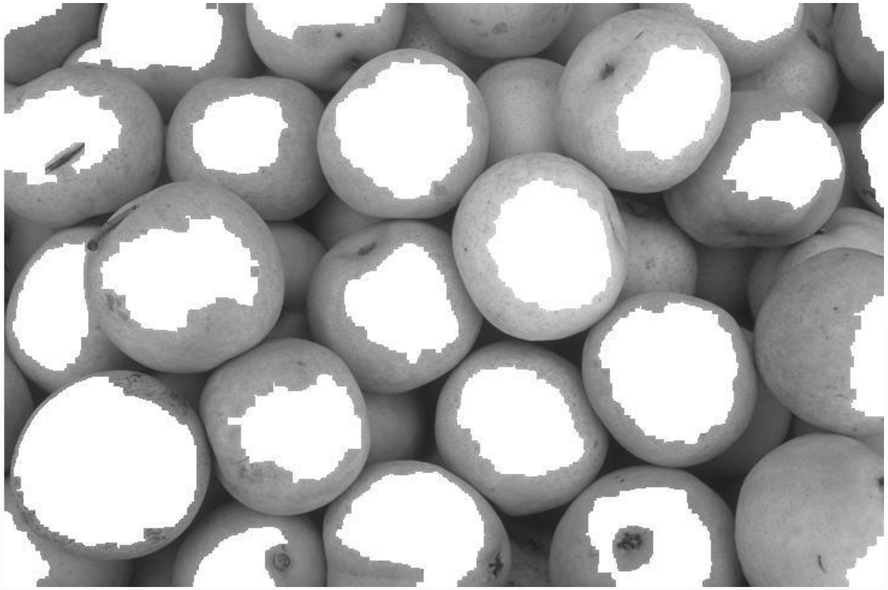
Gradient magnitude image

Watershed segmentation example

- Obtaining good foreground markers: regional maxima of the morphology enhanced input image



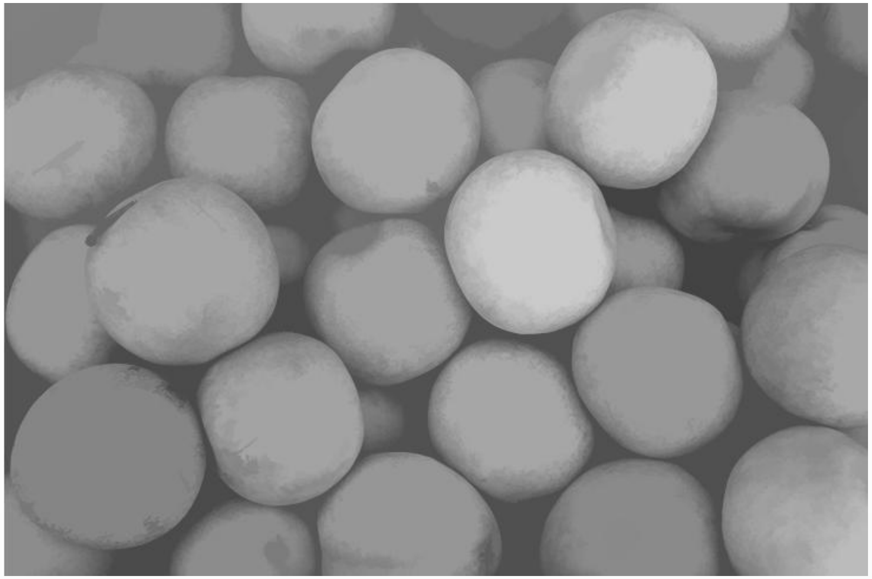
Result of grayscale morhpology (M)



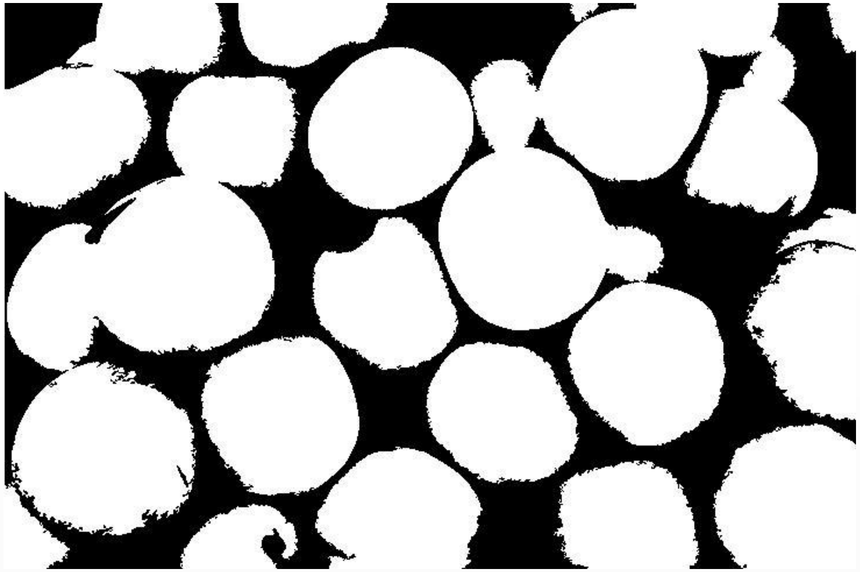
Regional maxima of (M) superimposed on original image (I)

Watershed segmentation example

- Obtaining good background markers
  - Step 1: threshold the morphology enhance image



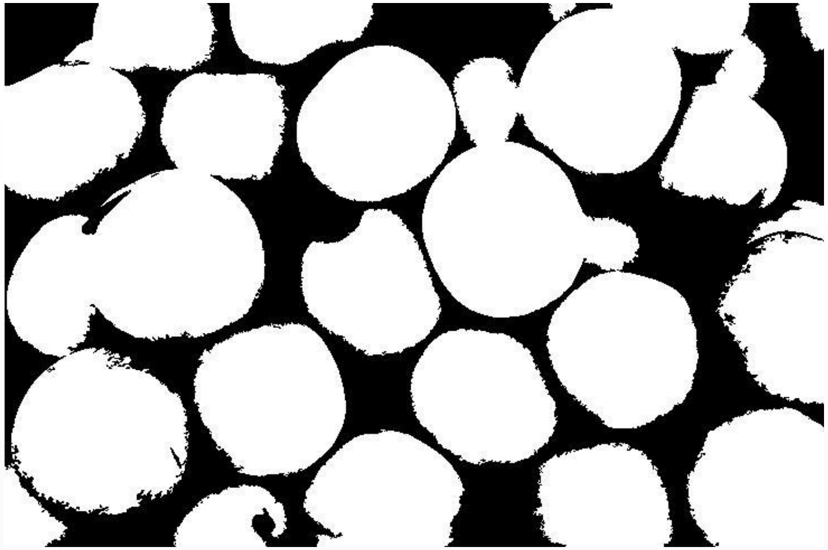
- Result of grayscale morhpology (M)



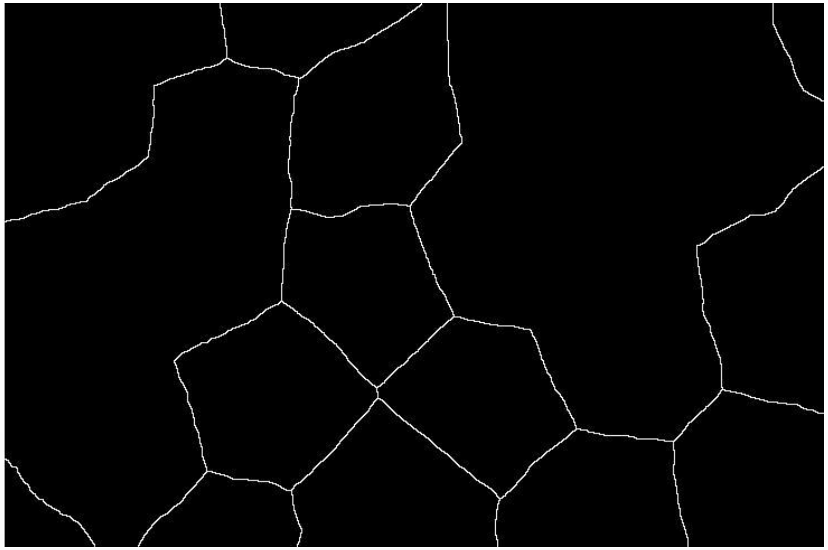
- T: result of Otsu threshold on M

Watershed segmentation example

- Obtaining good background markers
  - Step 1: threshold the morphology enhance image
  - Step 2: using the watershed transform of the distance transform of T, and then looking for the watershed ridge lines of the result

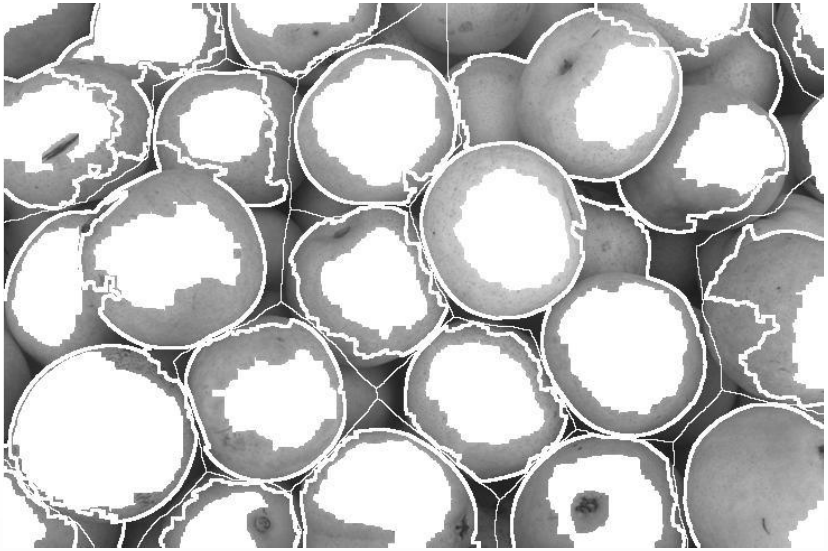


- T: result of Otsu threshold on M

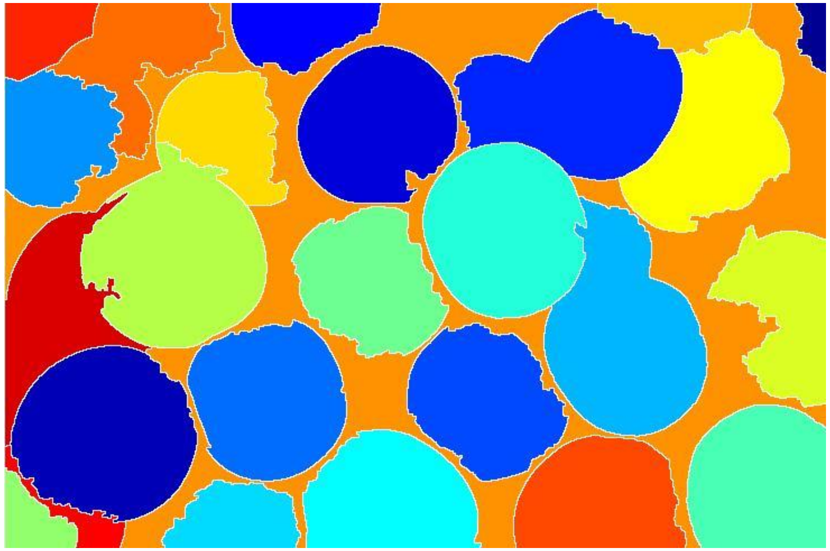


- Watershed lines (background markers)

Watershed segmentation:
Visualization of the results



Superimpose the foreground markers, background markers, and segmented object boundaries.



Segmentation results: display the label matrix as a color image

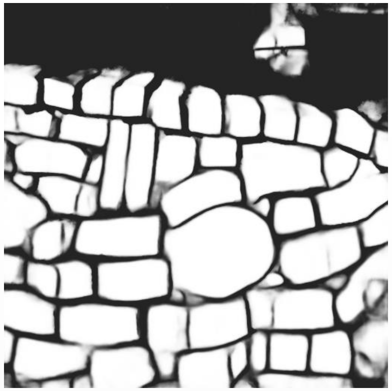
Matlab tutorial example, with source code: https://www.mathworks.com/help/images/examples/marker-controlled-watershed-segmentation.html?prodcode=IP&language=en

Watershed application example

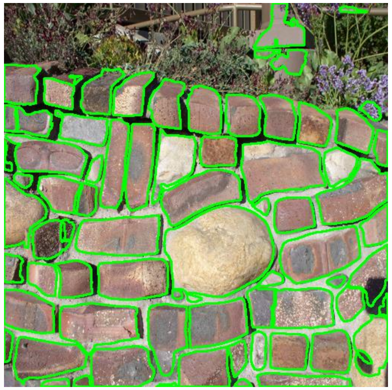
- Segmentation of masonry wall images

Input

Marker image by Deep Learning



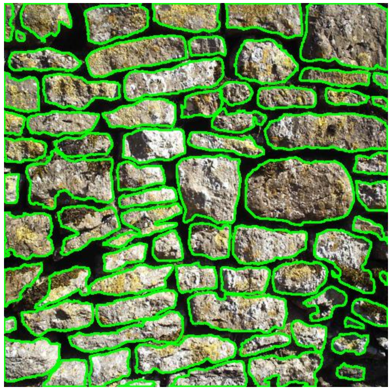
Segmentation result

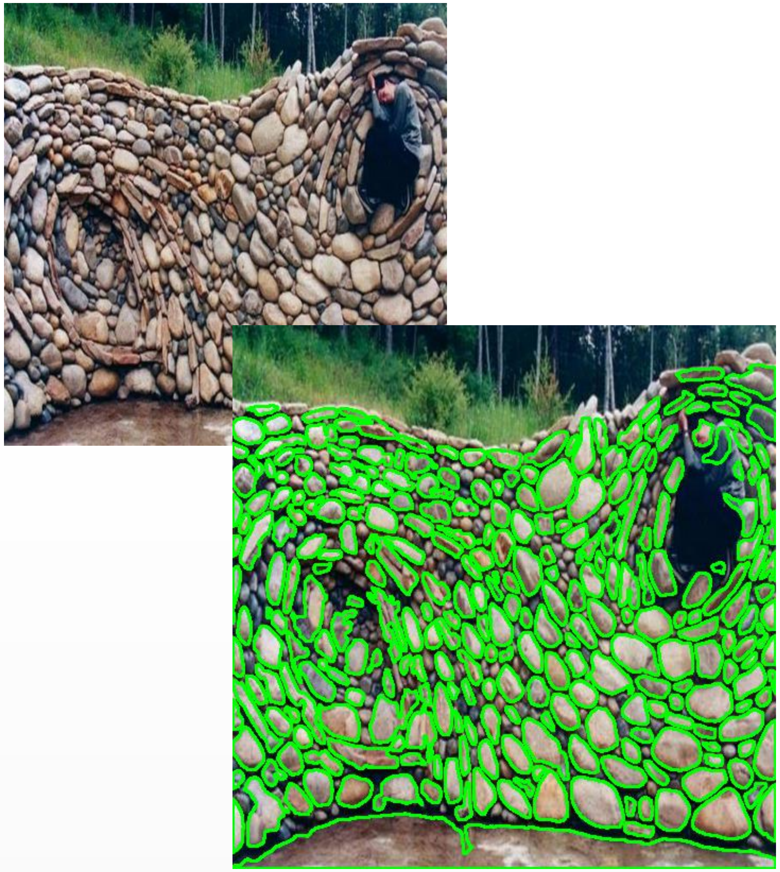


Ground Truth

Reference: Y. Ibrahim, B. Nagy and *Cs. Benedek*: "CNN-based Watershed Marker Extraction for Brick Segmentation in Masonry Walls", *International Conference on Image Analysis and Recognition (ICIAR)*, Waterloo, Canada, August 27-29, 2019

Further results…





Y. Ibrahim, B. Nagy and *Cs. Benedek*: "CNN-based Watershed Marker Extraction for Brick Segmentation in Masonry Walls", *International Conference on Image Analysis and Recognition (ICIAR)*, Waterloo, Canada, August 27-29, 2019

Summary: Watershed Segmentation

- There are 3 types of pixels:
  - Points belonging to a regional minimum
  - Point belonging to the catchment basin of a regional minimum
  - Points belonging to a watershed line
- The resulted boundaries of the regions are continuous.
- But it is time consuming and has over-segmentation problems.
- The solution to the over-segmentation is to use markers:
  - Internal markers:
    - Each one correspond to one object
    - Surrounded by points with higher altitude
    - Points in a region form a connected component
    - The points of the connected component has the same intensity
  - External markers:
    - Segment the image into regions with one internal marker object and background points.

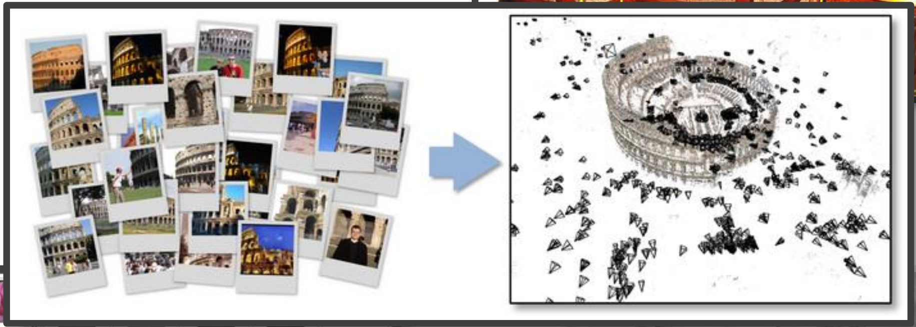
Basic Image Processing Algorithms

PPKE-ITK

Lecture 11.

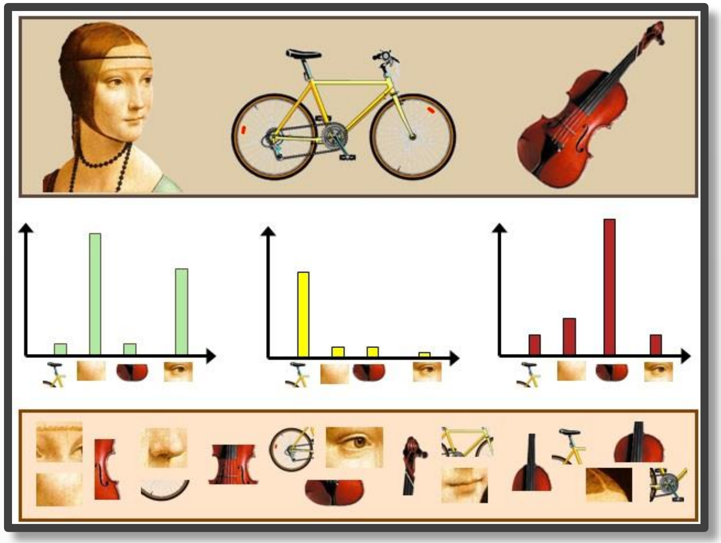
Local Feature Descriptors

- The detection and description of local features has an important role in many applications:
  - Object recognition/detection/tracking
  - Image and video retrieval
  - Image registration, motion estimation
  - Wide baseline matching
  - Texture classification
  - Structure from Motion
  - etc.



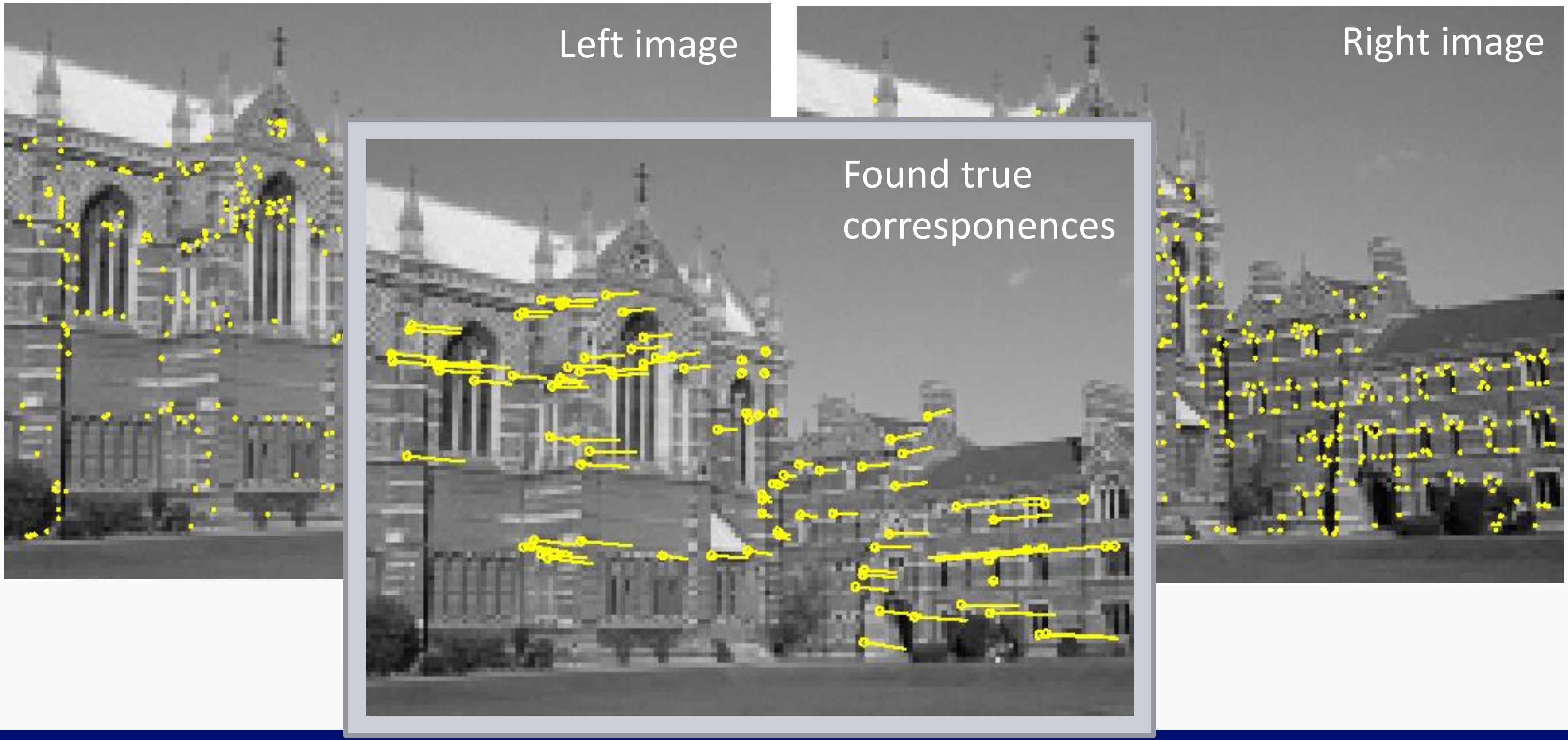
Local Feature Descriptors

- There are different types of use of the descriptors:
  - Description and matching of key points:
    1. Feature/Keypoint detection
    2. Local feature description around the key points
    3. Keypoint matching
  - Bag-of-Features (or bag-of-words)
    1. Feature detection
    2. Feature description
    3. Feature clustering
    4. Frequency histogram construction for image or image part description
  - Description of a specific area:
    1. Find the region of interest (ROI) (e.g. scanning through the image)
    2. Description of the ROI
    3. Classification/Clustering of the ROI descriptor

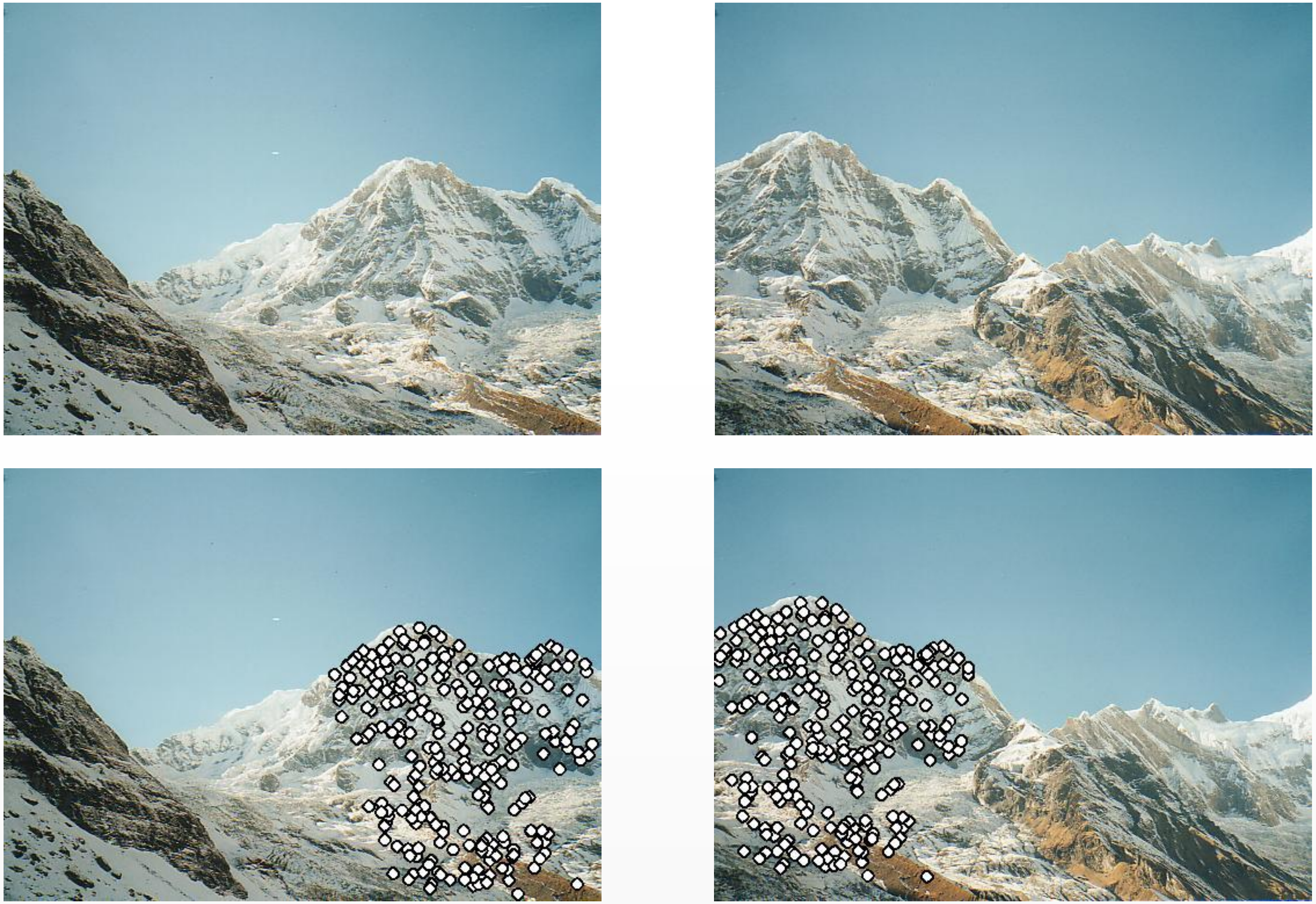


Keypoint matching example

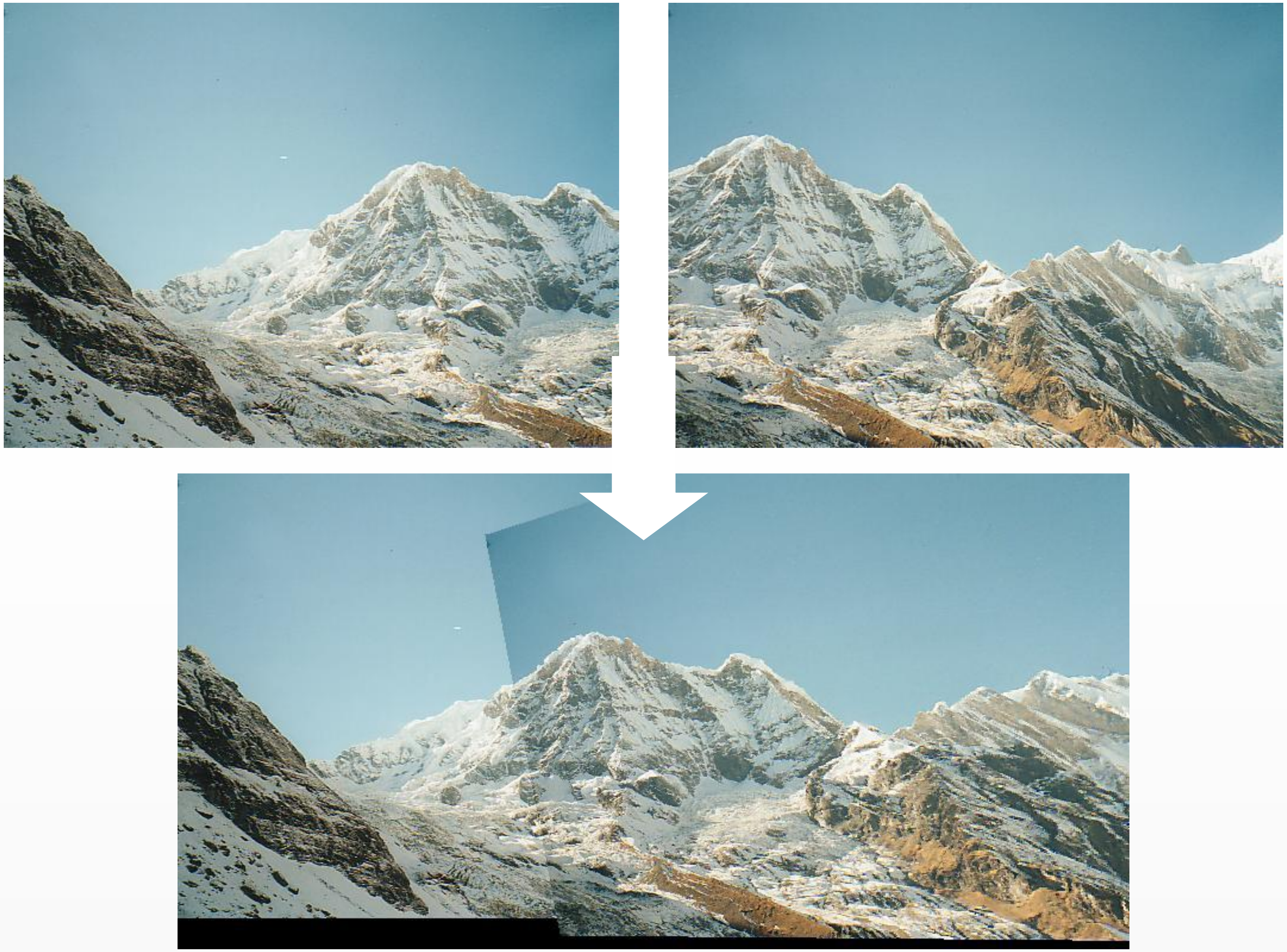
- Application: pixel level image matching from stereo images for depth map calculation



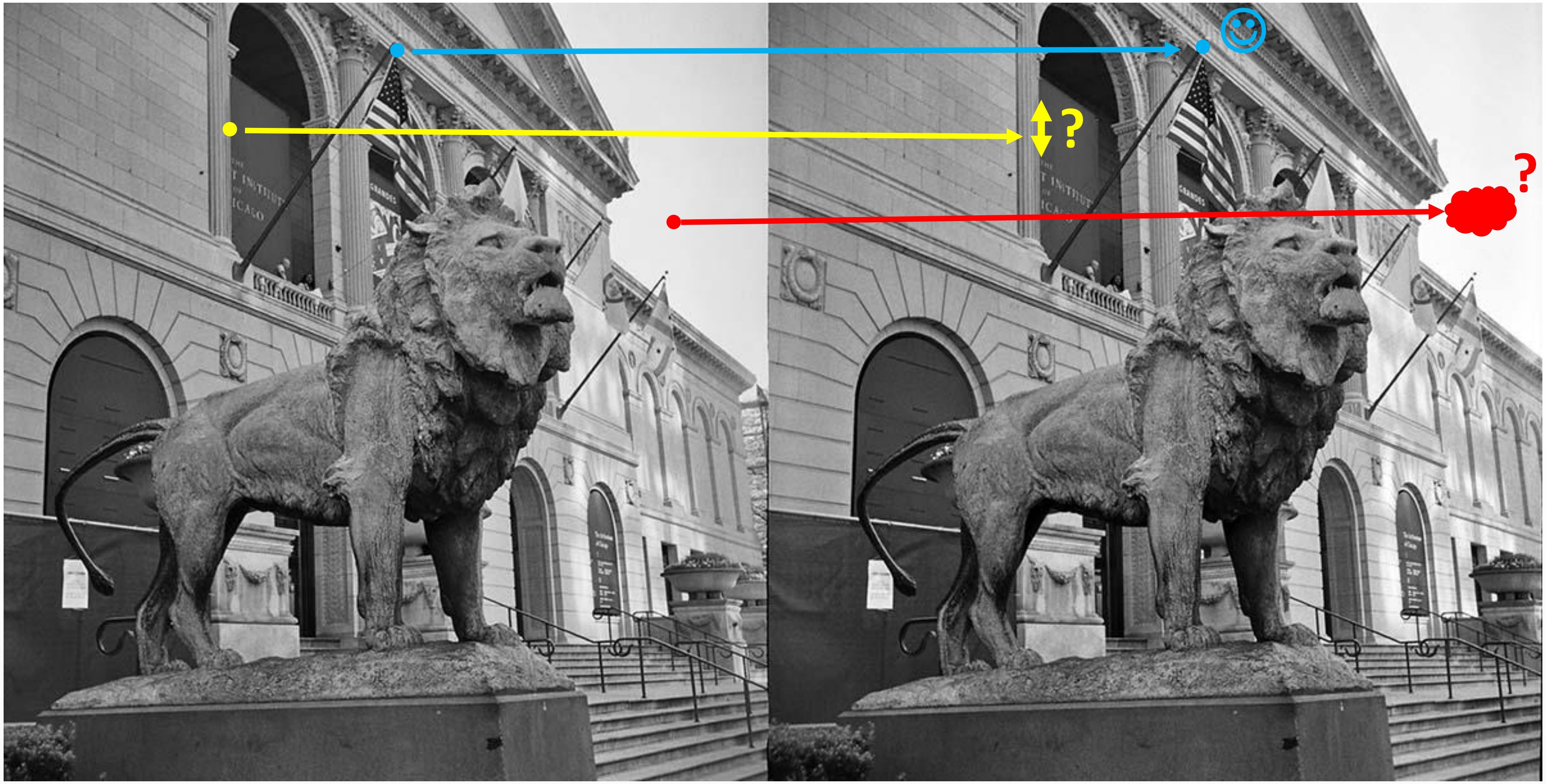
Feature matching for panoramic image creation (results from Matt Brown)



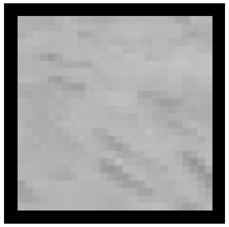
Some Matching Results



Which pixels are easy to match?

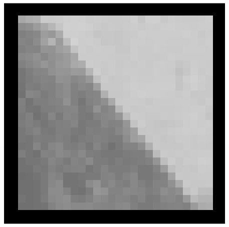


Interest points



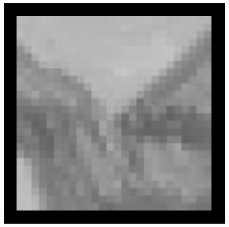
0D structure: **single points**

not useful for matching



1D structure: **lines**

edge, can be localised in 1D,
subject to the aperture problem



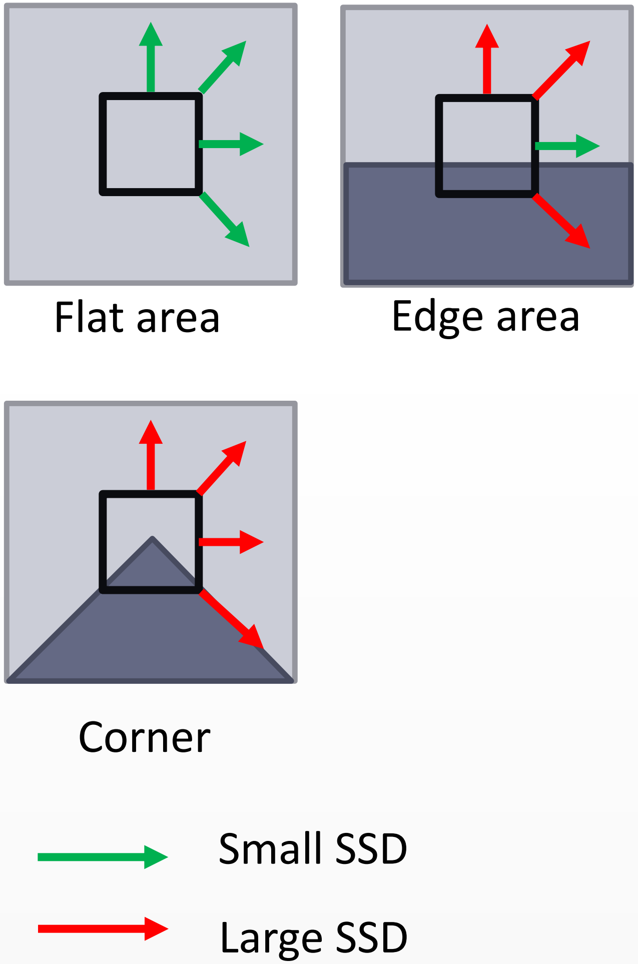
2D structure: corners

corner, or **interest point**, can be localized
in 2D, good for matching

Interest Points have **2D** structure.

How to find good feature points?

- Based on the idea of auto-correlation
  - Sum of squared differences (SSD) matching



- Important difference in all directions
  => interest point

Background: Moravec Corner Detector (1980)

- take a window W in the image
- shift it in four directions

  $[\Delta x, \Delta y] \in \{[1,0],[0,1],[1,1],[-1,1]\}$
- compute a SSD difference for each direction
- compute the *min* difference at each pixel
- *local maxima* in the *min* image are corners

$$\mathrm{SSD}(\Delta x, \Delta y) = \sum_{(x_k,y_k)\in W} \left(I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y)\right)^2$$

- Limitation: not isotropic
  - if an edge is present that is not in the direction of the neighbors (horizontal, vertical, or diagonal), then the smallest SSD will be large and the edge will be incorrectly chosen as an interest point.

Harris detector

- Auto-correlation function (SSD) for a point (x, y) and an **arbitrary** shift $(\Delta x, \Delta y)$ - *not only 4 directions*!

$$f(x,y) = \sum_{(x_k,y_k)\in W} \left(I(x_k,y_k) - I(x_k + \Delta x, y_k + \Delta y)\right)^2$$

- Discrete shifts can be avoided with the auto-correlation matrix (Taylor approximation):

$$I(x_k + \Delta x, y_k + \Delta y) = I(x_k,y_k) + [I_x(x_k,y_k) \quad I_y(x_k,y_k)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

  - where I_x and I_y are the derivative images (e.g Prewitt). Then:

$$f(x,y) = \sum_{(x_k,y_k)\in W} \left([I_x(x_k,y_k) \quad I_y(x_k,y_k)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}\right)^2$$

Harris detector

- Rewrite as inner (dot) product

$$f(x,y) = \sum_{(x_k,y_k)\in W} \left(\begin{bmatrix} I_x(x_k,y_k) & I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 =$$

$$= \sum_{(x_k,y_k)\in W} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x(x_k,y_k) \\ I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} I_x(x_k,y_k) & I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} =$$

- The center portion is a 2x2 matrix:

$$= \sum_W \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \sum_W \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} =$$

Harris detector

$$= [\Delta x \quad \Delta y] \underbrace{\begin{bmatrix} \sum_{(x_k,y_k)\in W} \left(I_x(x_k,y_k)\right)^2 & \sum_{(x_k,y_k)\in W} I_x(x_k,y_k) I_y(x_k,y_k) \\ \sum_{(x_k,y_k)\in W} I_x(x_k,y_k) I_y(x_k,y_k) & \sum_{(x_k,y_k)\in W} \left(I_y(x_k,y_k)\right)^2 \end{bmatrix}}_{M} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

- M: auto-correlation matrix of the local gradient map
  - captures the structure of the local neighborhood (recap: PCA)
  - measure based on **eigenvalues** λ_1, λ_2 of M
    - 0 strong eigenvalue ($\lambda_1 \approx 0, \lambda_2 \approx 0$) => uniform region
    - 1 strong eigenvalue ($\lambda_1 \gg 0, \lambda_2 \approx 0$) => contour
    - **2 strong eigenvalues ($\lambda_1 \gg 0, \lambda_2 \gg 0$) => interest point (corner)**
- Interest point detection:
  - threshold on the eigenvalues, then find maximum for localization

Some Details from the Harris Paper

- Alternative measure for corner strength to avoid eigenvalue computation:

$$R = \mathrm{Det}(M) - \kappa\,\mathrm{Tr}^2(M)$$

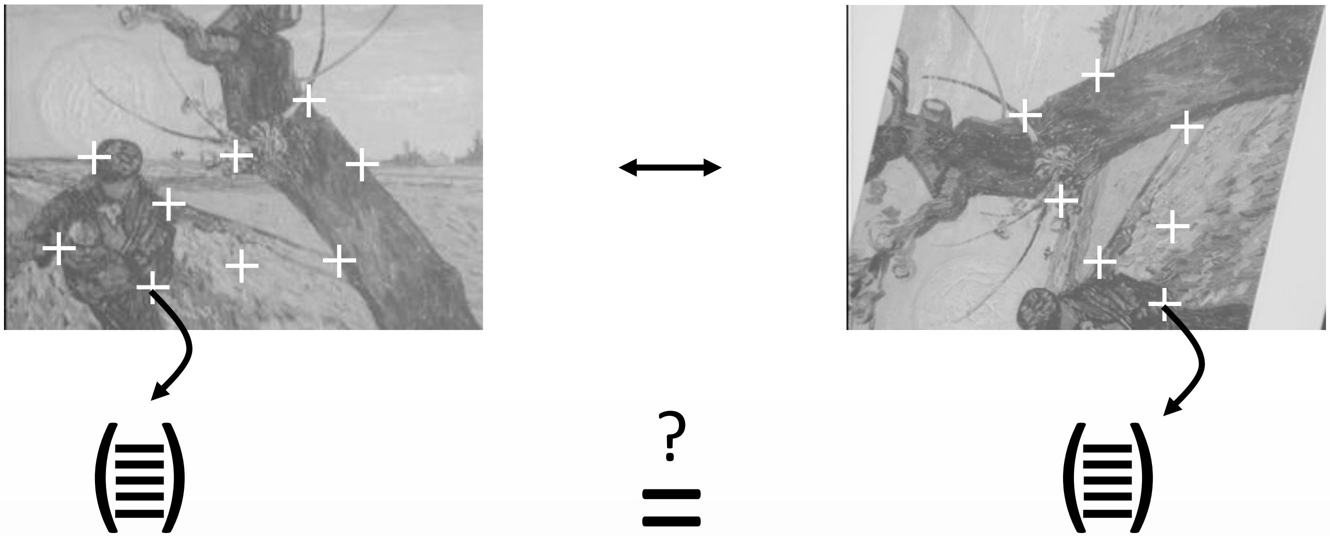
$$M = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathrm{Tr}(M) = a_{11} + a_{22} \quad \mathrm{Det}(M) = a_{11}a_{22} - a_{12}a_{21}$$

- It can be shown:

$$\mathrm{Tr}^2(M) = \lambda_1 + \lambda_2 \qquad \mathrm{Det}(M) = \lambda_1\lambda_2$$

  - *instead of calculating the* λ_1 and λ_2 eigenvalues, we use trace and determinant (κ parameter is usually between $0.04 - 0.15$)
- Classification based on R:
  - *positive* for corners ($R \gg 0$),
  - *negative* for edges ($R \ll 0$) ,
  - small for flat regions ($R \approx 0$)
- *Select corner pixels that are 8-way local maxima*

Determining correspondences



Vector comparison using a distance measure

What are some suitable distance measures?

Distance Measures

- Let W_1 and W_2 be two rectangular windows in image I_1 and I_2 respectively
  - The two windows have same size, but not necessarily the same center positions
- To compare W_1 and W_2 we can use the sum-square difference of the values of the pixels in a square neighborhood about the points being compared. *This is the simplest measure.*

$$\mathrm{SSD}(W_1, W_2) = \sum_{(x_k, y_k)} \left(W_1(x_k, y_k) - W_2(x_k, y_k)\right)^2$$

Harris based feature matching

- Basic feature matching = **Harris Corners & Correlation**
- Very good results in the presence of occlusion and clutter
  - local information
  - discriminant grayvalue information
  - invariance to illumination
- No invariance to scale and affine changes, limited invariance to image rotation
- Solution for more general view point changes
  - local invariant descriptors to scale and rotation
  - extraction of invariant points and regions

Rotation/Scale Invariance

original

translated

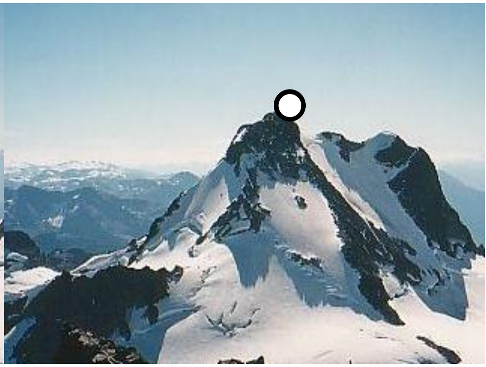
rotated

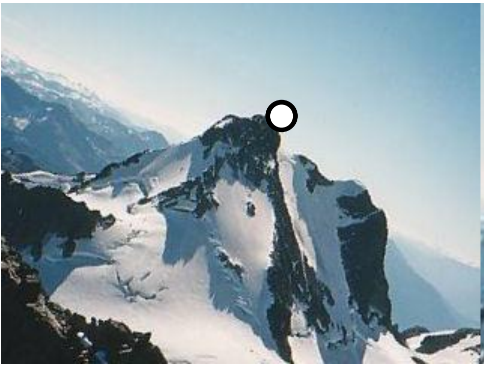
scaled

	Translation	Rotation	Scale
Is Harris invariant?	?	?	?
Is correlation invariant?	?	?	?

Rotation/Scale Invariance

original

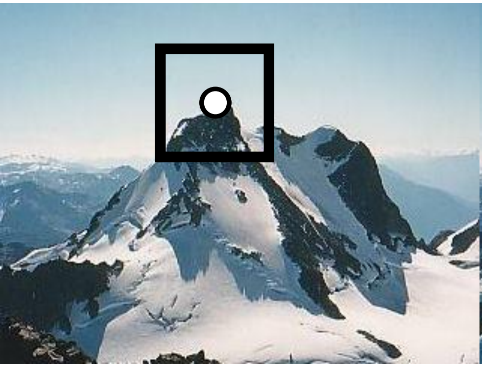

translated


rotated

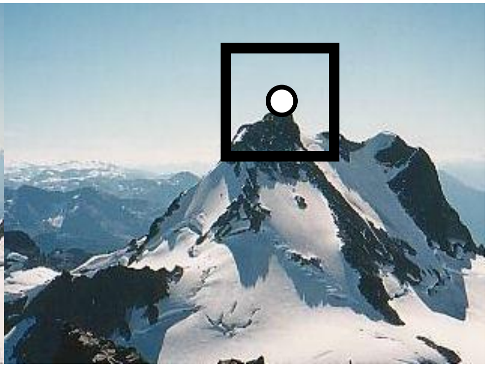
scaled

	Translation	Rotation	Scale
Is Harris invariant?	YES	YES	NO
Is correlation invariant?	?	?	?

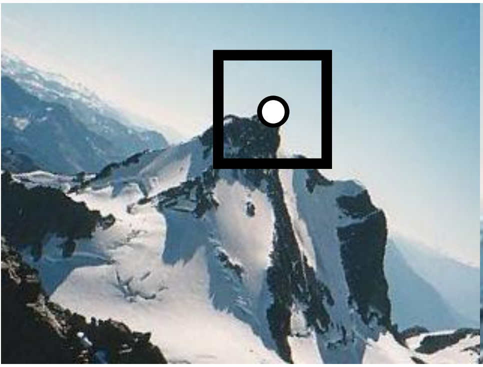
Rotation/Scale Invariance



original



translated

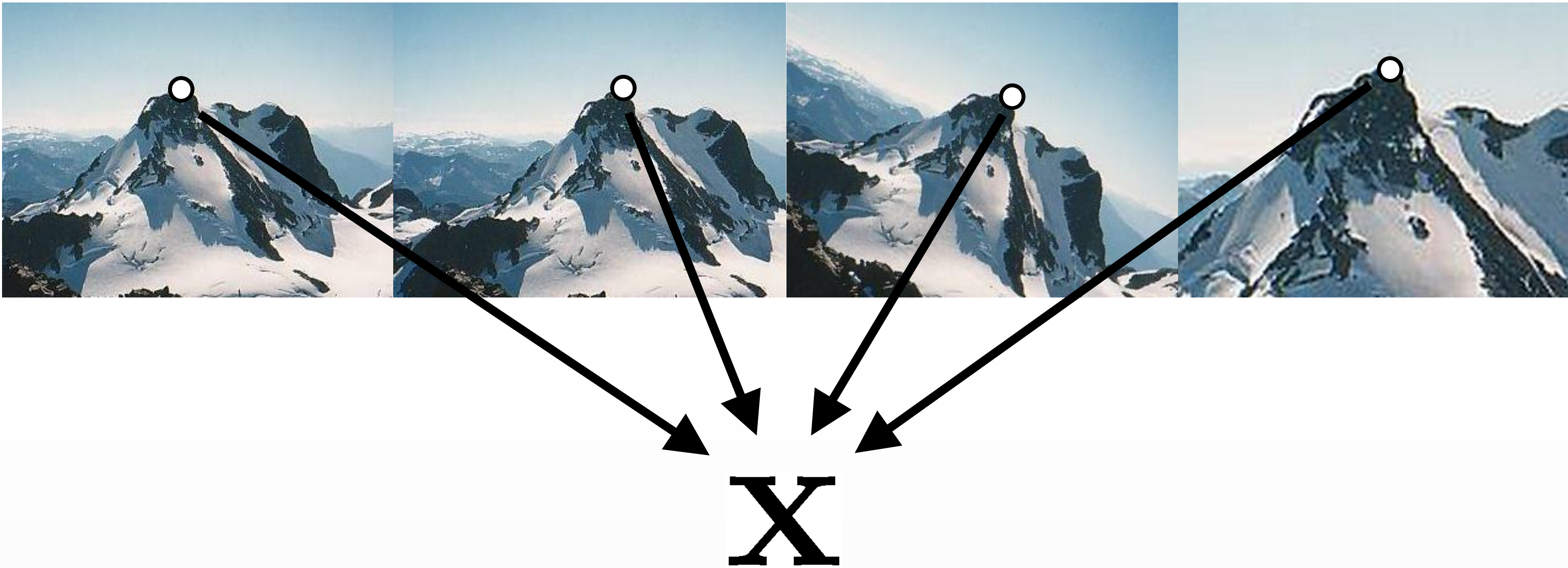


rotated

scaled

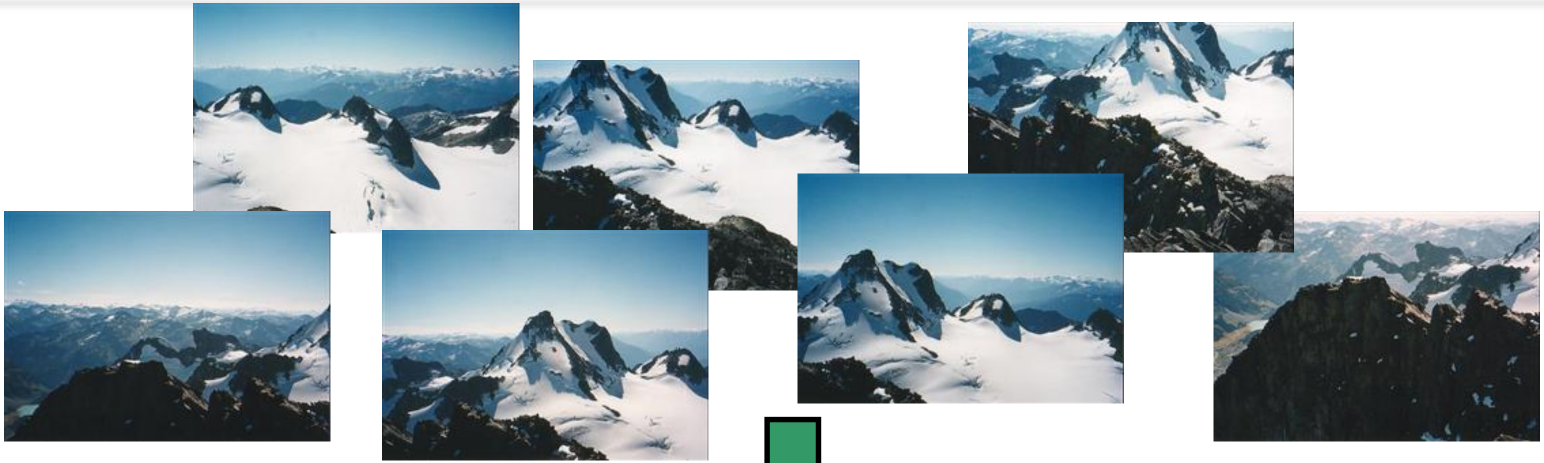
	Translation	Rotation	Scale
Is Harris invariant?	YES	YES	NO
Is correlation invariant?	YES	NO	NO

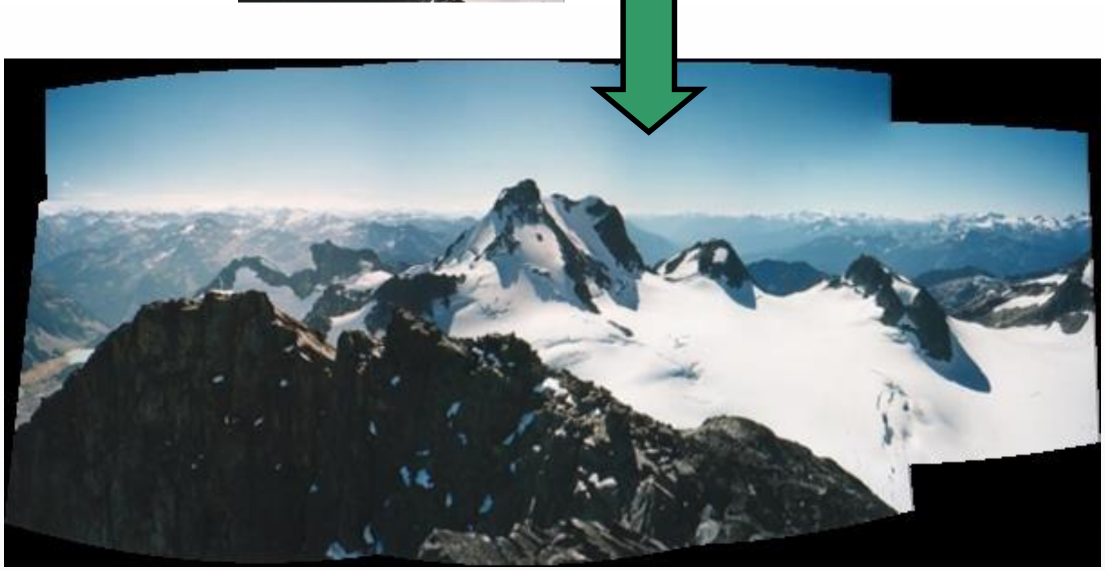
Matt Brown's Invariant Features



- Local image descriptors that are *invariant* (unchanged) under image transformations

Application: Image Stitching

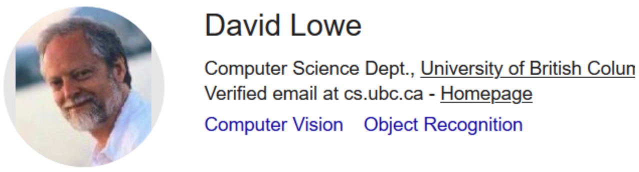




[Microsoft Digital Image Pro version 10]

SIFT: Scale-Invariant Image Transform

- Developed by David Lowe, University of British Columbia

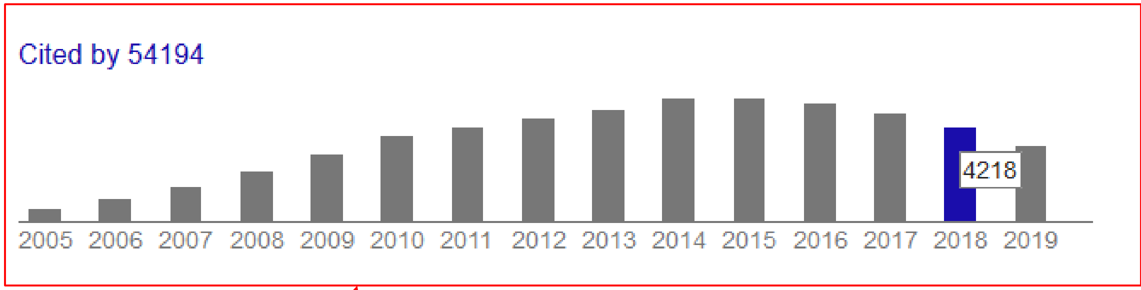


David Lowe

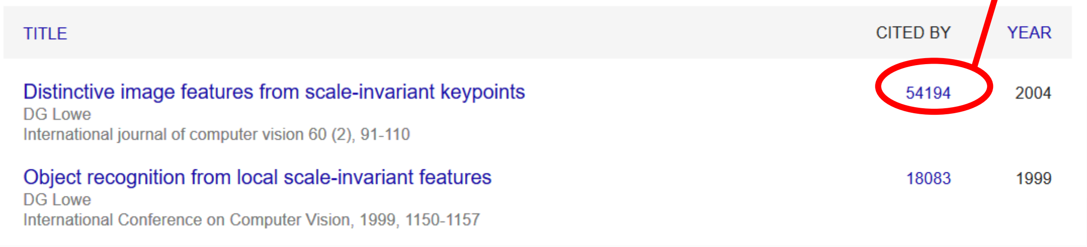
Computer Science Dept., University of British Colun

Verified email at cs.ubc.ca - Homepage

Computer Vision Object Recognition



Cited by 54194



TITLE	CITED BY	YEAR
Distinctive image features from scale-invariant keypoints DG Lowe International journal of computer vision 60 (2), 91-110	54194	2004
Object recognition from local scale-invariant features DG Lowe International Conference on Computer Vision, 1999, 1150-1157	18083	1999

Citation data from 25.11.2019

- Lowe, „*Object recognition from local scale-invariant features*”, *In: IEEE International Conference on Computer Vision*, Vol. 2 (1999), pp. 1150-1157 vol.2.
- Lowe, „ Distinctive image features from scale-invariant keypoints” International journal of computer vision 60 (2), 91-110, 2004

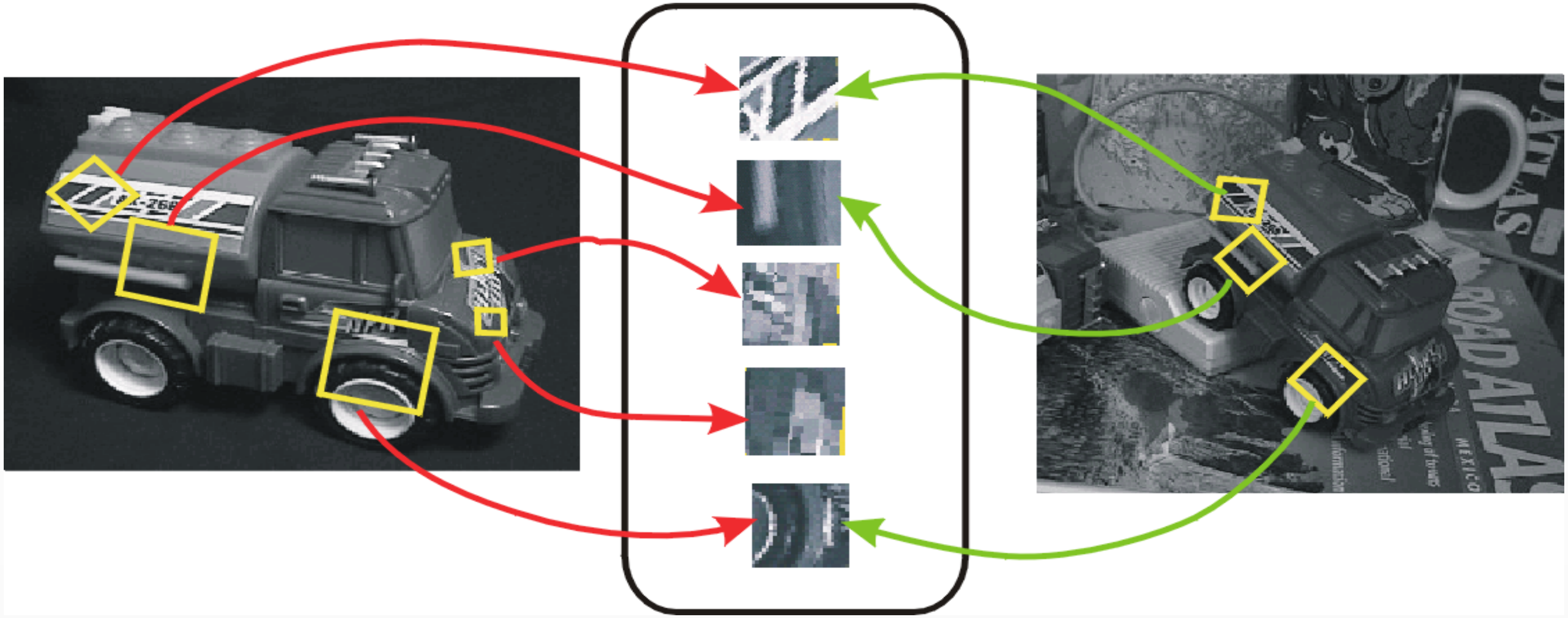
SIFT: Motivation

- The Harris operator is not invariant to scale and correlation is not invariant to rotation[1].
- For better image matching, Lowe's goal was to develop an interest operator that is invariant to scale and rotation.
- Also, Lowe aimed to create a *descriptor* that was robust to the variations corresponding to typical viewing conditions. *The descriptor is the most-used part of SIFT.*

[1] But Schmid and Mohr developed a rotation invariant descriptor for it in 1997.

Idea of SIFT

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

SIFT: Scale-Invariant Image Transform

- Advantages:
  - Invariant to translation, scaling, and rotation
  - Robust to illumination changes, noise, minor changes in viewpoint
  - Robust to local geometric distortion
  - Highly distinctive
  - SIFT based object detectors are robust to partial occlusion
- Steps of the Algorithm:
  1. Scale-space extrema detection
  2. Keypoint localization
  3. Orientation assignment
  4. Keypoint description

I. Scale-space extrema detection

Scale change via Gaussian blur

- Blurring image with a Gaussian kernel:
  - Loosing details i.e. transforming the image into a different scale

$$L(x, y, k \cdot \sigma) = G(x, y, k \cdot \sigma) * I(x, y)$$

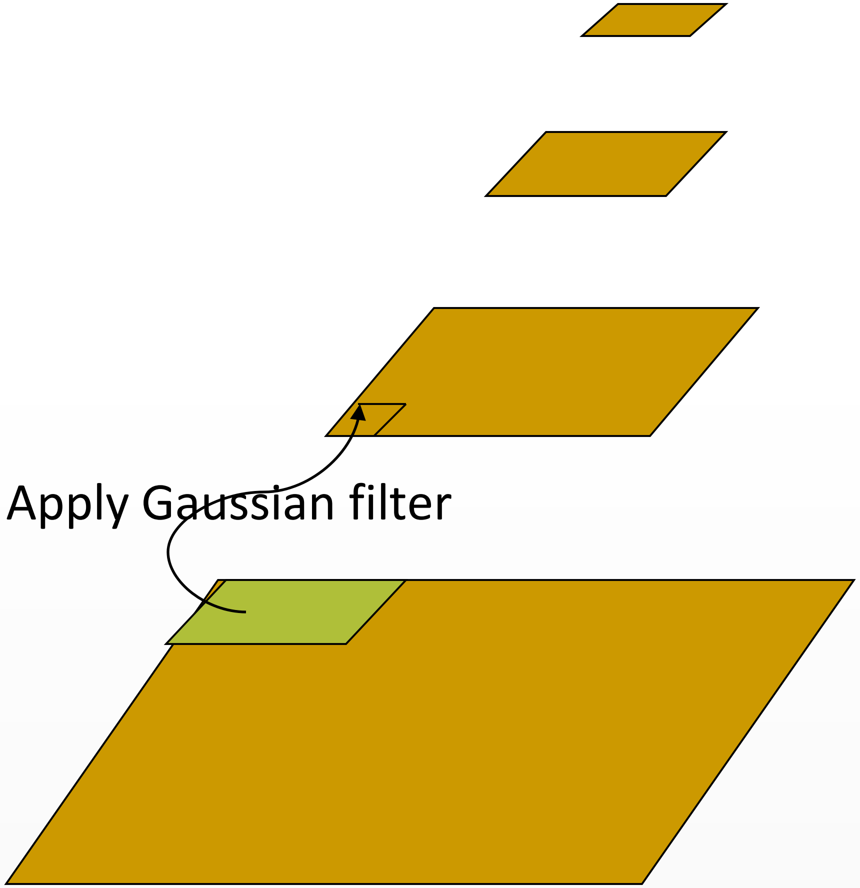
  - Where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  - Increasing k will increase the amount of blur, i.e. yields a lower scale representation of the image content
  - At a certain level of blur, the image can be spatially downscaled

Gaussian Pyramid

At each level, image is smoothed and reduced in size.

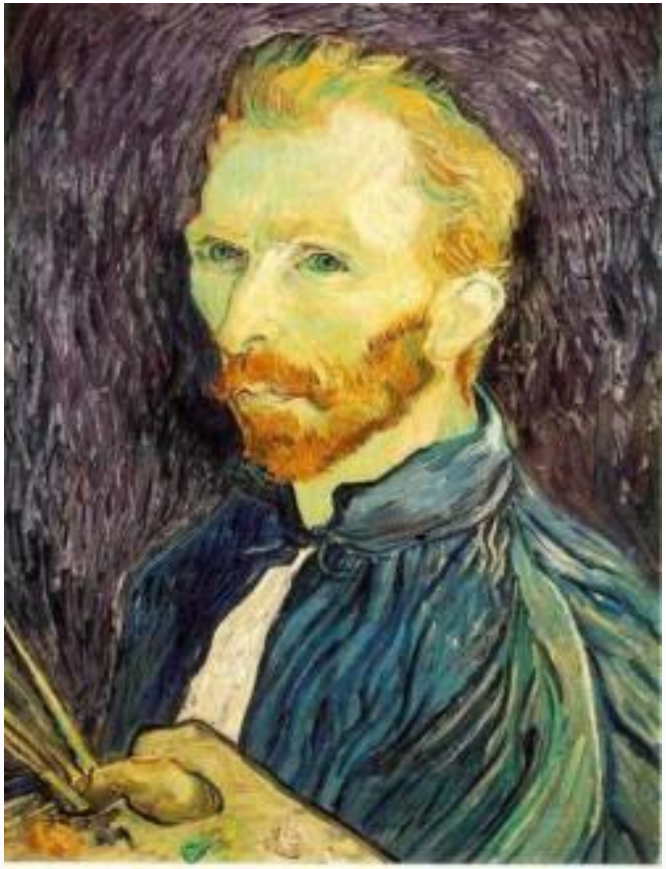


And so on.

At 2nd level, each pixel is the result of applying a Gaussian mask to the first level and then subsampling to reduce the size.

Bottom level is the original image.

Example:
Subsampling with Gaussian pre-filtering



Gaussian 1/2

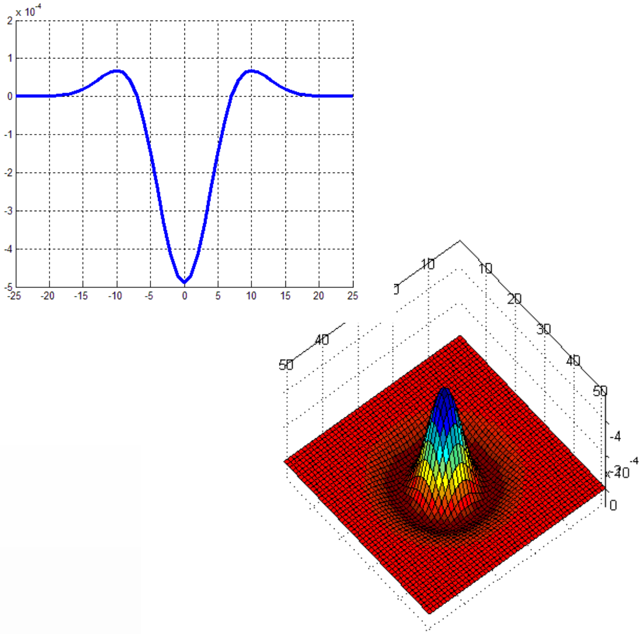
G 1/4

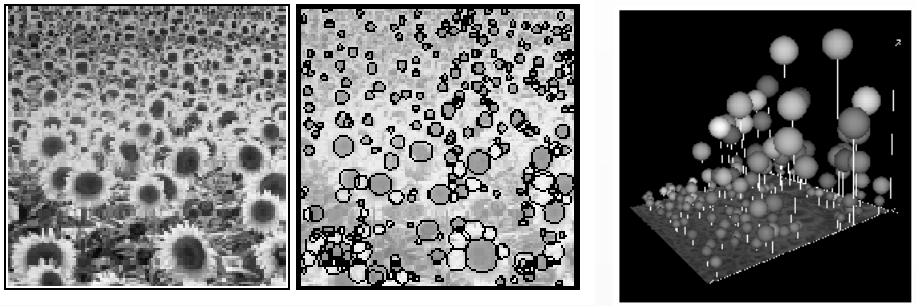
G 1/8

I. Scale-space extrema detection

Lowe's Scale-space Interest Points

- **Laplacian of Gaussian (LoG)** kernel
    - Scale normalised (x by scale^2)
    - Proposed by Lindeberg
- Scale-space detection
    - Find local maxima across scale/space
    - A good "blob" detector



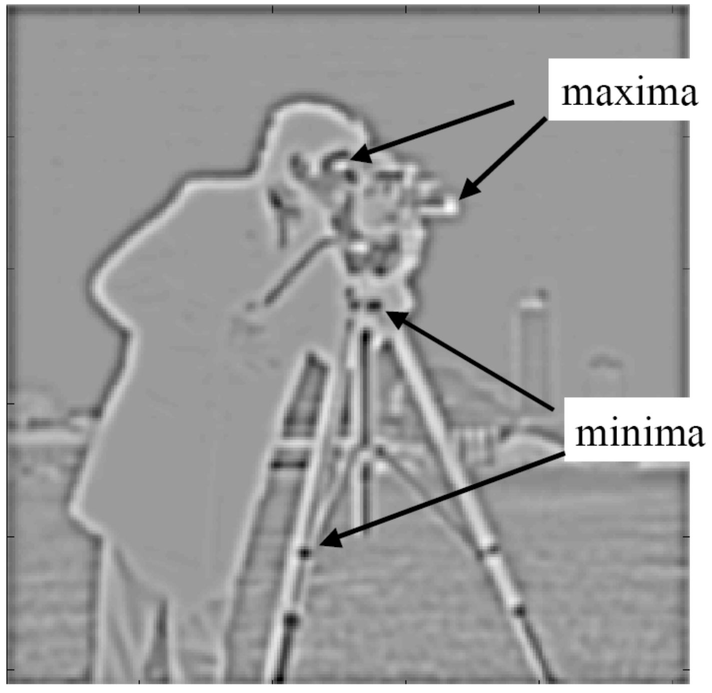


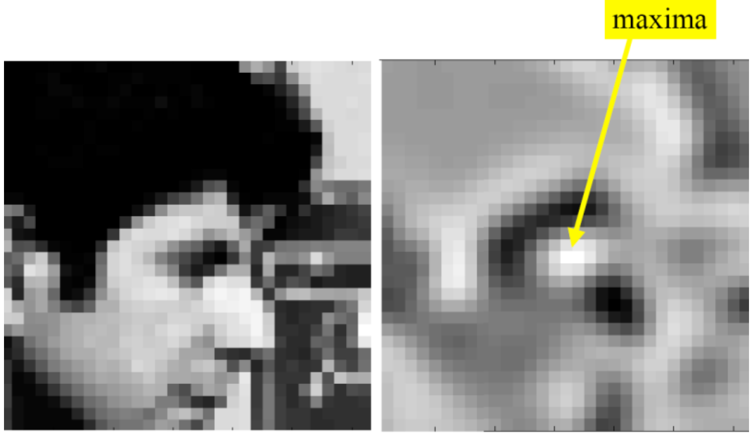
[T. Lindeberg IJCV 1998]

$$G(x,y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}}$$

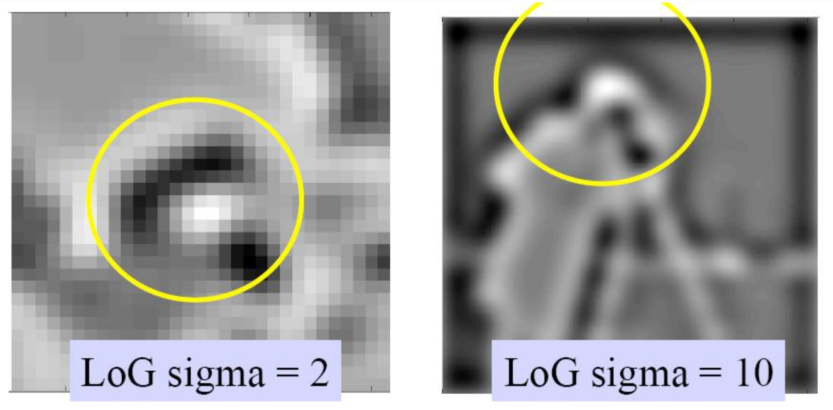
$$\nabla^2 G(x,y,\sigma) = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

LoG extrema examples

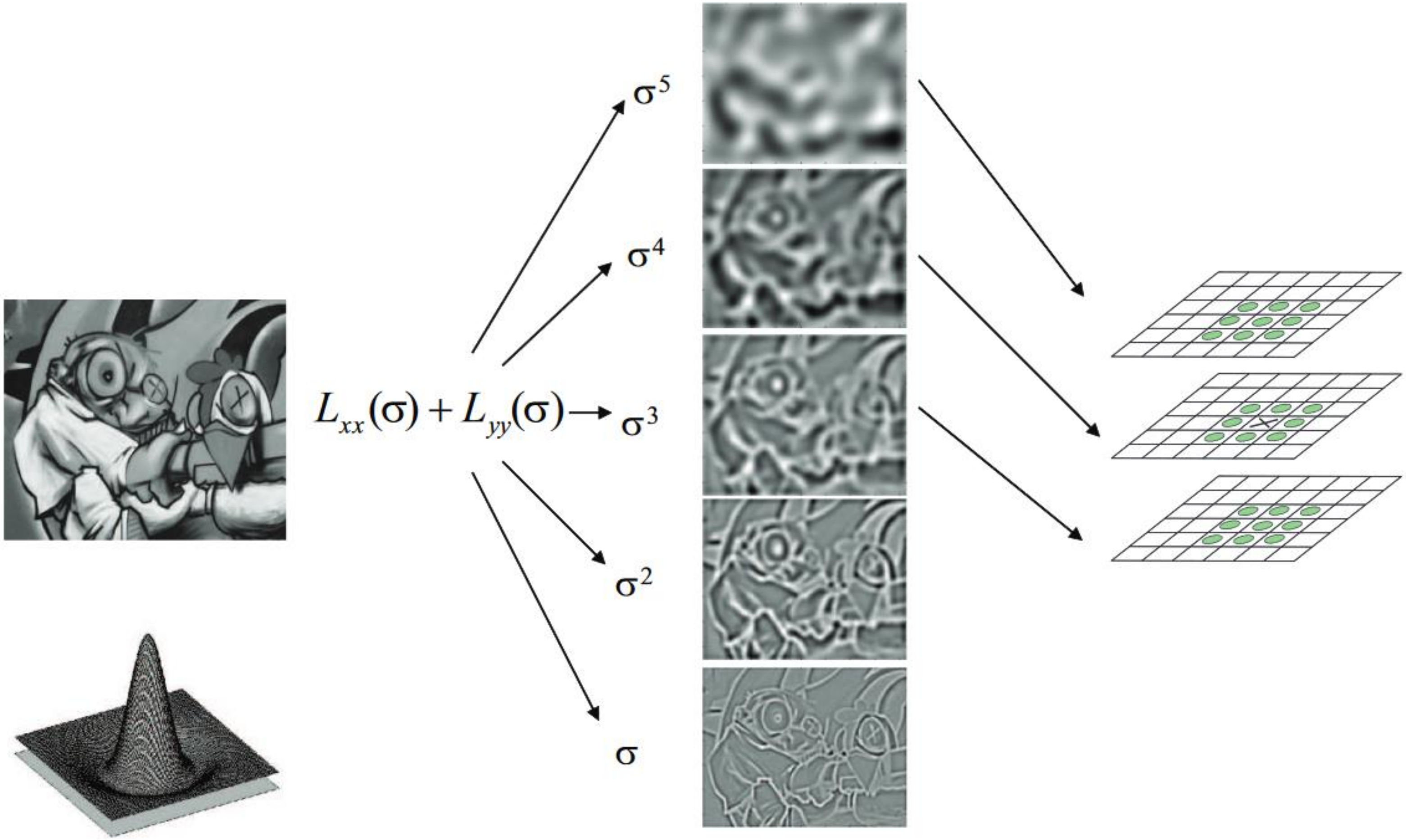




- Dependence on sigma of blurring (i.e. scale):



LoG as blob detector



I. Scale-space extrema detection

Lowe’s Scale-space Interest Points

- Using Laplacian of Gaussian (LoG) directly - $\sigma^2 \nabla^2 G$
  - Extrema useful: found stable feature and gives excellent notion of scale
  - Calculation costly instead…
- Approximation of LoG:

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2 \nabla^2 G$$

  - Difference of Gaussians (DoG) approximates LoG:

$$D(x,y,\sigma) = \big(G(x,y,k\sigma) - G(x,y,\sigma)\big) * I(x,y) =$$

$$= L(x,y,k\sigma) - L(x,y,\sigma)$$

I. Scale-space extrema detection

Workflow

- I. Scale-space extrema detection:
  - Key point detection with ***Difference of Gaussians*** (DoG):
    - $I(x, y)$ is the original image
    - $G(x, y, k\sigma)$ is the Gaussian blur at scale $k\sigma$
    - The original image convolved with Gaussian kernel at different scales:

$$L(x, y, k \cdot \sigma) = G(x, y, k \cdot \sigma) * I(x, y)$$

    - The convolved images are grouped by octave (in an octave σ is doubled). The difference of consecutive convolved images is taken in an octave:

$$D(x, y, \sigma) = L(x, y, k_i \cdot \sigma) - L(x, y, k_j \cdot \sigma)$$

I. Scale-space extrema detection

Lowe's Pyramid Scheme

s+2 filters

$\sigma_{s+1}=2^{(s+1)/s}\sigma_0$

.

.

$\sigma_i=2^{i/s}\sigma_0$

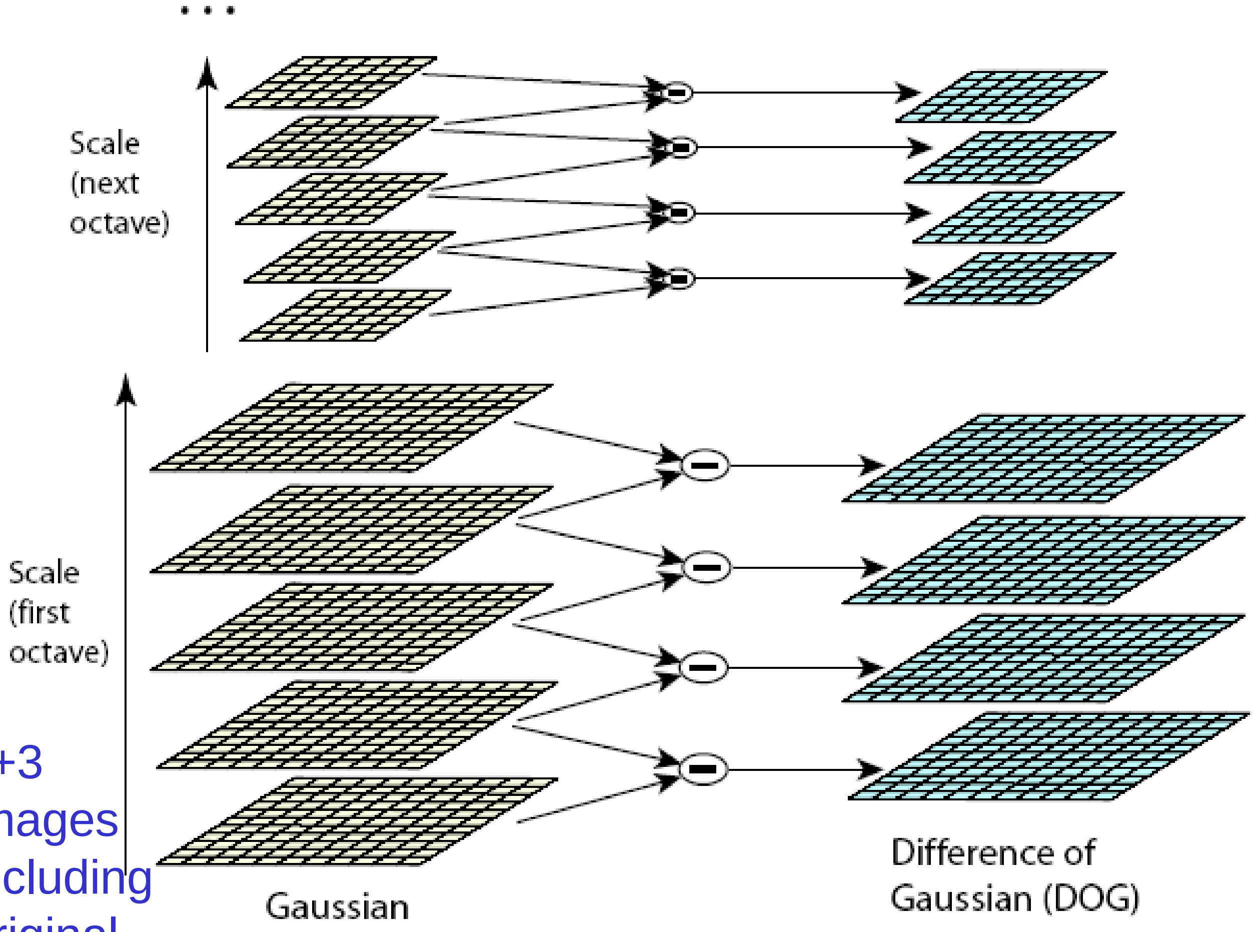
.

.

$\sigma_2=2^{2/s}\sigma_0$

$\sigma_1=2^{1/s}\sigma_0$

σ_0



s+3 images including original

s+2 differ-ence images

The parameter **s** determines the number of images per octave.

I. Scale-space extrema detection
Lowe's Pyramid Scheme

- Scale space is separated into **octaves**:
  - Octave 1 uses scale σ
  - Octave 2 uses scale 2σ
  - etc.

- In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images.

- Adjacent Gaussians are subtracted to produce the DOG

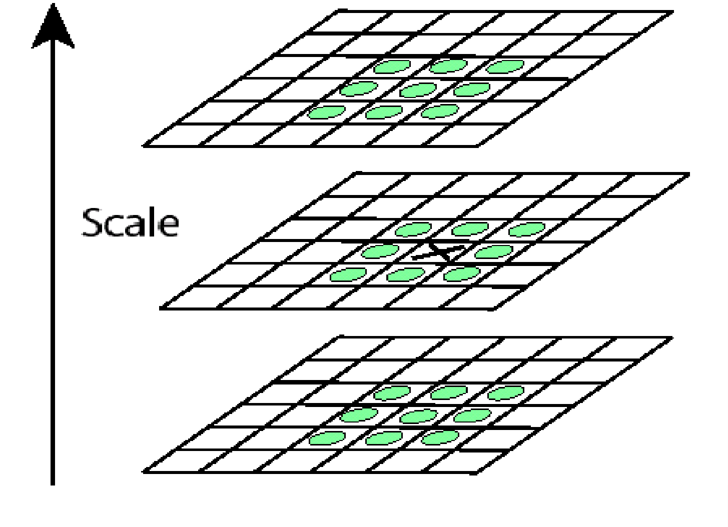
- After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image ¼ the size to start the next level.

II. Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space

- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below

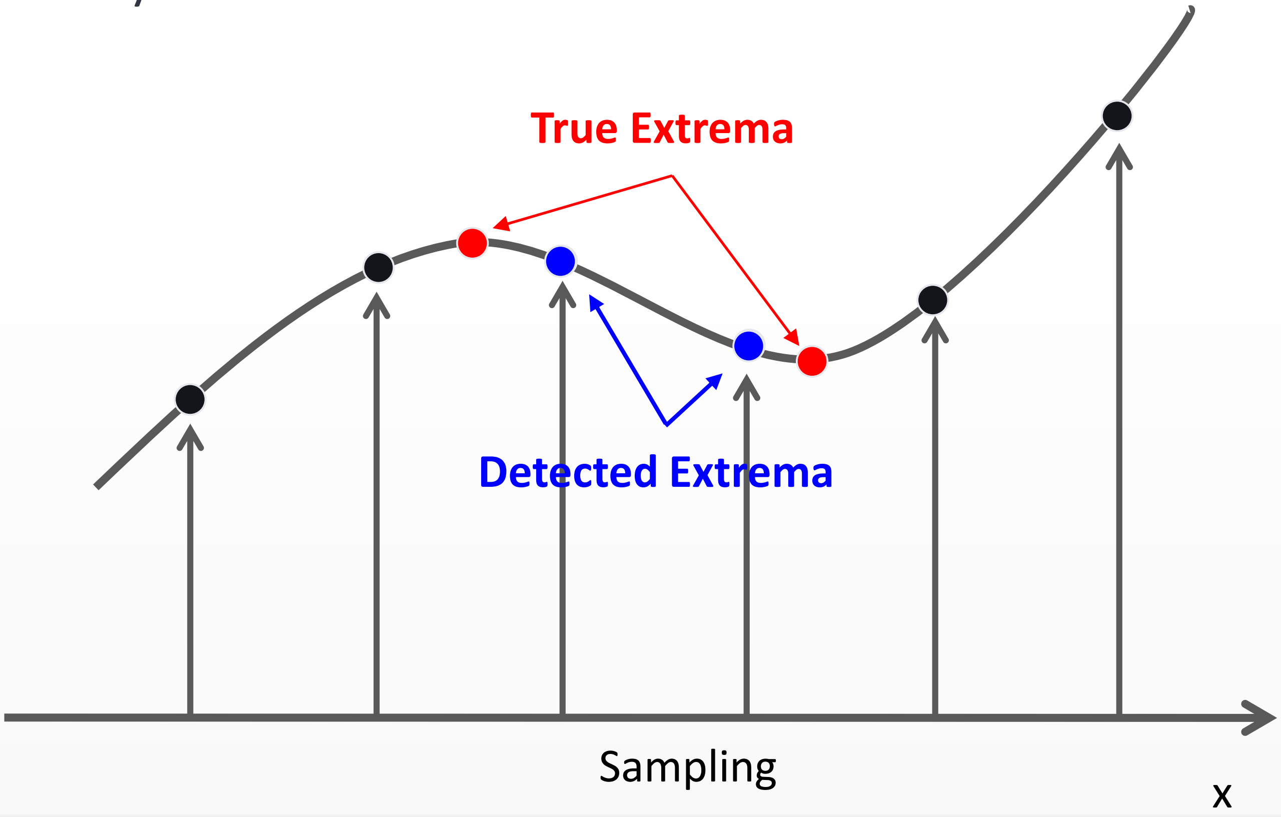
s+2 difference images.
top and bottom ignored.
s planes searched.



For each max or min found, output is the **location** and the **scale**.

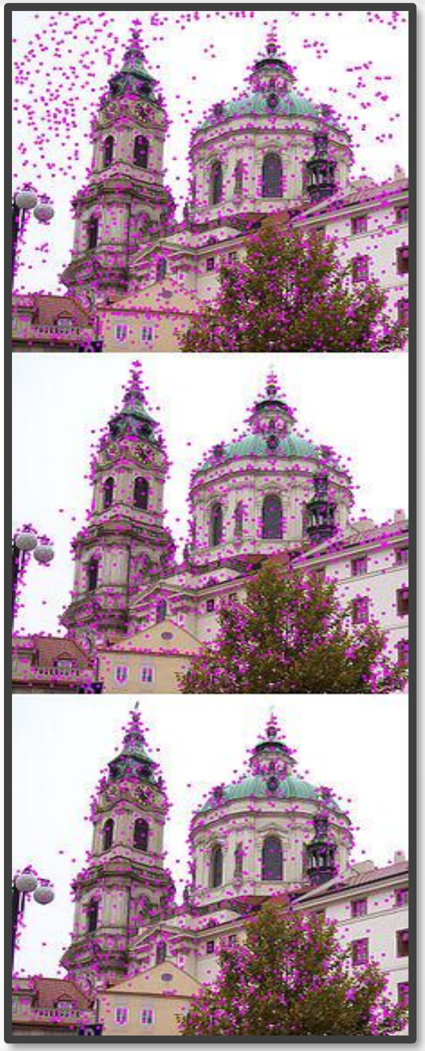
II. Key point localization

- II. Keypoint localization:
  - Localization is done with sub pixel accuracy, based on the interpolation of nearby data:



II. Key point localization

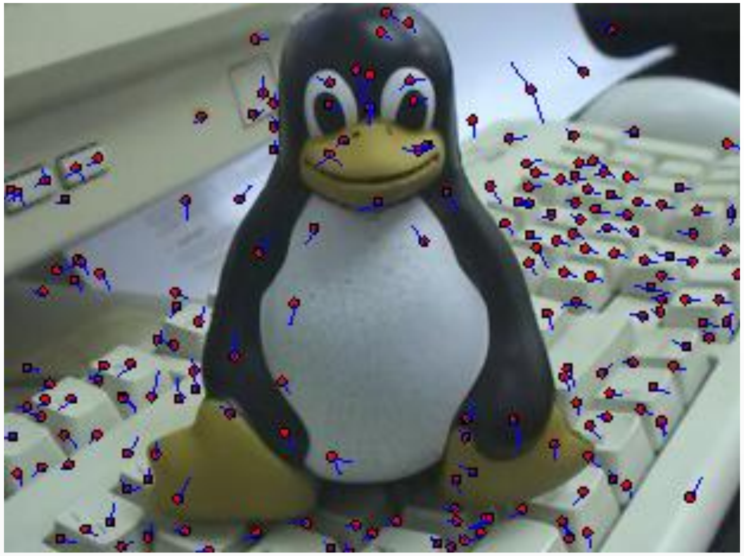
- II. Keypoint localization:
  - Rejection of weak candidates:
    - Low contrasted points
    - Poorly localized points along edges:
      - The DoG function will have strong responses along edges, but these points are not stable, since their location is poorly defined.
      - These points will be removed based on the principal curvature across and along the edge.
  - Similar approach to Harris, but here the ratio of the trace^2 and determinant of the Hessian detector matrix (Beaudet, 1978) is calculated



http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

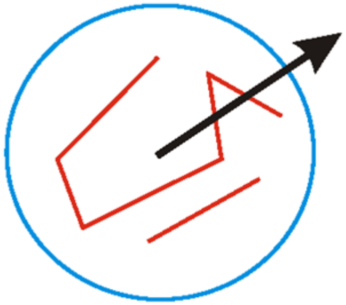
III. Orientation Assignment

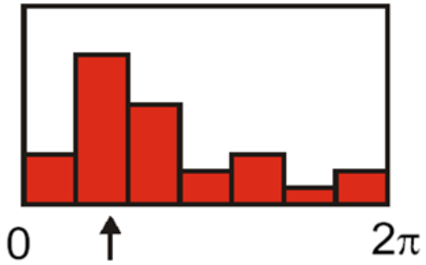
- Orientation Assignment: goal is to ensure rotation invariance
  - Find the main orientation(s)
  - Assign it to the key point and
  - Give the description of the keypoint relative to this orientation.



III. Orientation Assignment

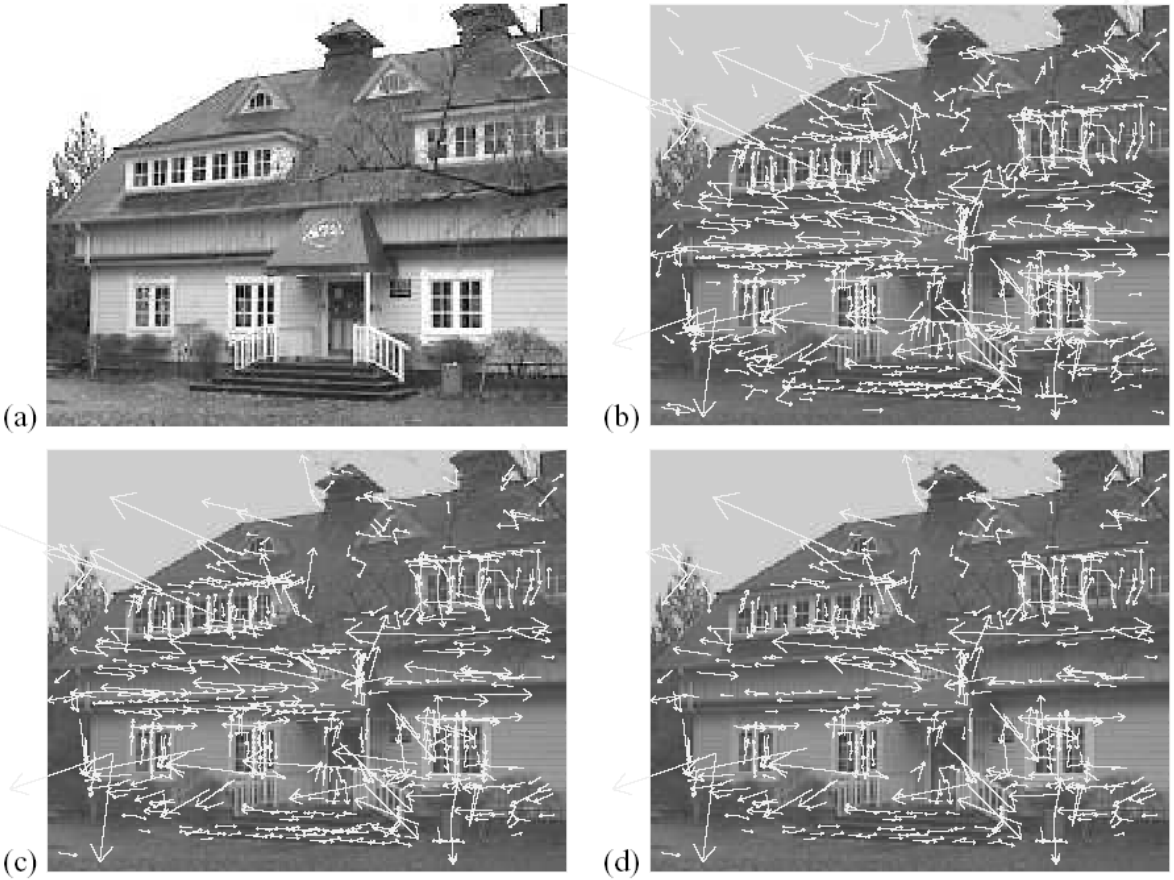
- Steps of orientation assignment
  - ***Gaussian smoothed*** image is taken at the scale of the keypoint.
  - The ***edge magnitude and orientation*** is calculated ***for each point*** in the neighborhood.
  - A ***36 bin orientation histogram*** is composed, where each bin represents a 10 degree interval, and each neighboring point's bin is determined based on its edge orientation and its weight based on the edge magnitude.





- Also the points are ***weighted with a Gaussian*** window, so the points farther away has less effect than the points closer to the keypoint.
- The canonical orientation of the keypoint will correspond to the peak of the histogram.

Result of Keypoint localization with orientation assignment



233x189

832

initial keypoints

729

keypoints after low contrast based rejection

536

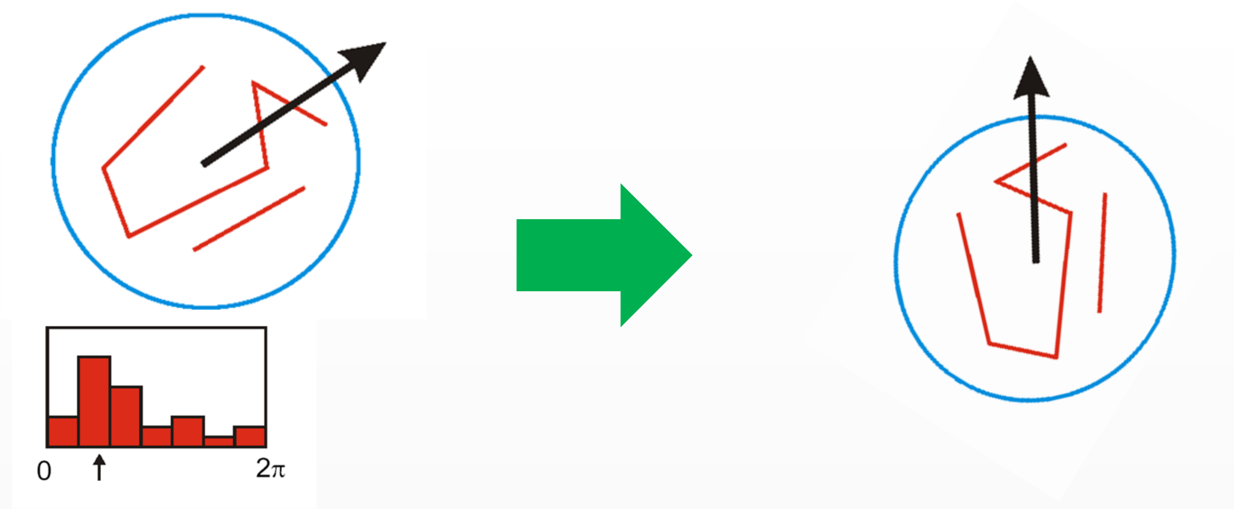
keypoints after ratio threshold

IV. Keypoint Descriptors

- At this point, each keypoint has
  - location
  - scale
  - orientation
- Next is to compute a descriptor for the local image region about each keypoint that is
  - highly distinctive
  - invariant as possible to variations such as changes in viewpoint and illumination

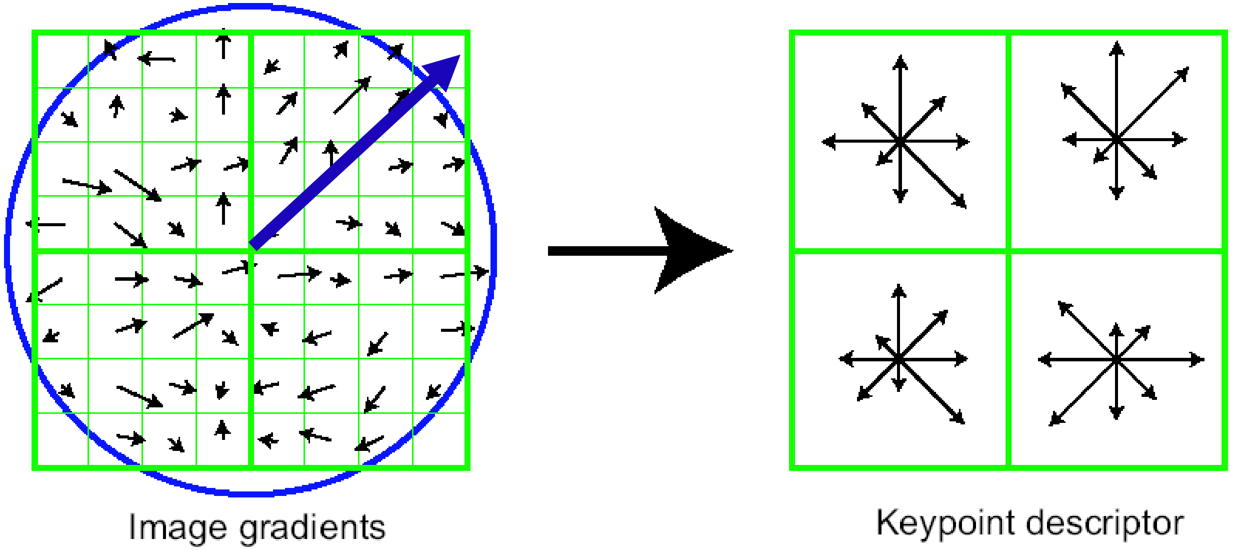
Normalization

- Rotate the window to standard orientation (e.g. the calculated canonical orientation vector should point upwards)
- Scale the window size based on the scale at which the point was found.



Lowe's Keypoint Descriptor
(shown with 2 X 2 descriptors over 8 X 8)

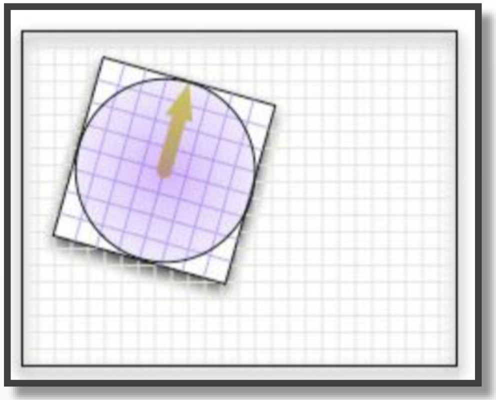
- *Demonstration example*: take here a 8x8 point neighborhood around the keypoint and divide it into 2x2 gradient window.
- Build the orientation histogram of the 2x2 samples in each window with 8 direction bins – concatenate the 8bin histograms to obtain feature vector.



- In practice: 4x4 arrays (form 16x16 point neigborhoods), with 8 bin histogram is used, a total of 4x4x8=128 features for one keypoint

IV. Keypoint Descriptor: Overview

- Use the normalized region about the keypoint
- Take a 16x16 point neighborhood around the keypoint and divide it into 4x4 gradient window.
- Compute gradient magnitude and orientation at each point in the region (weight them by a Gaussian window overlaid on the circle)
- Build the orientation histogram of the 4x4 samples in each window with 8 direction bins.
- 4 X 4 times 8 directions gives a vector of 128 values.



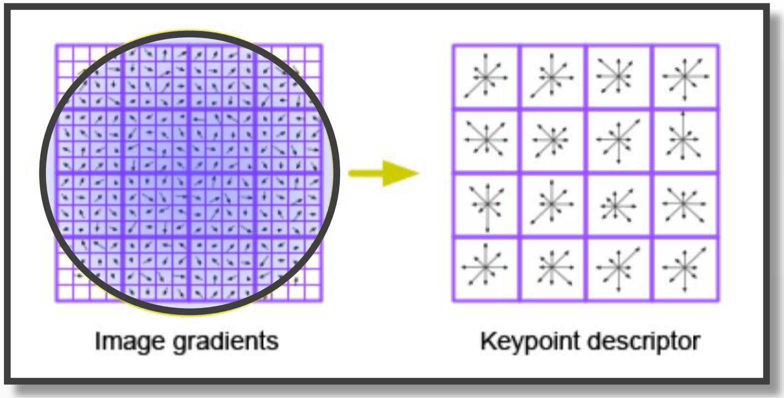
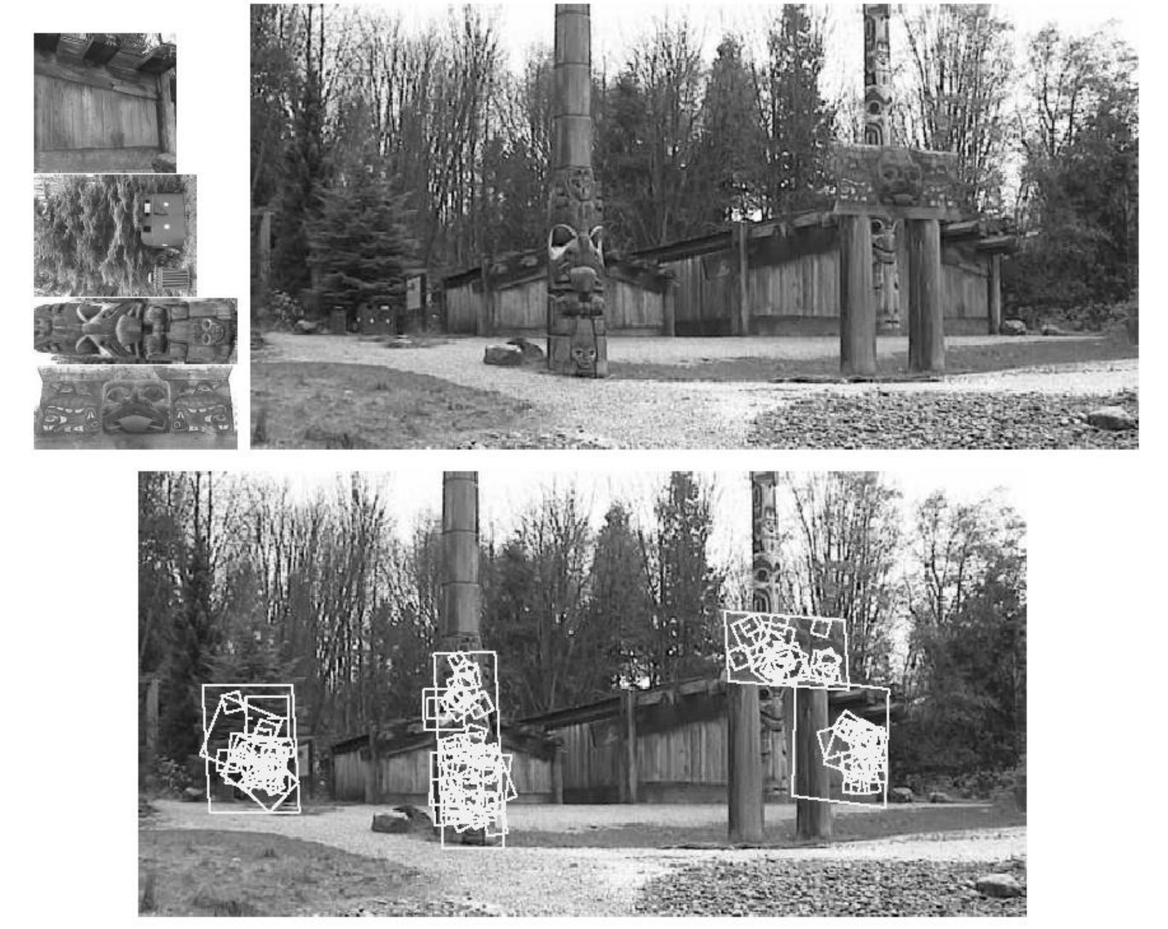
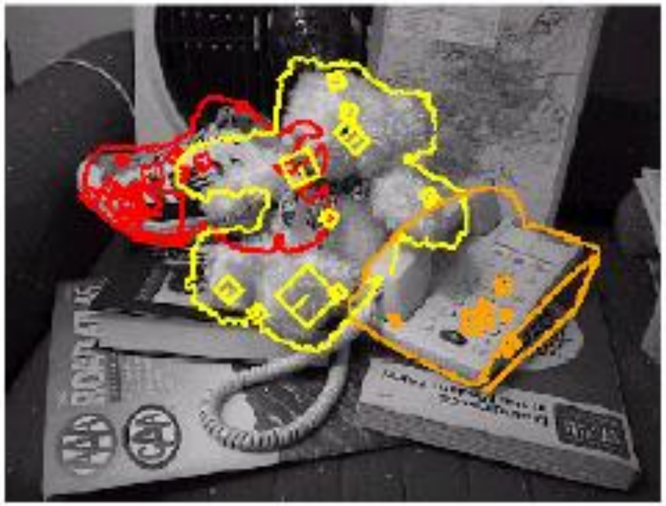


Image from: Ofir Pele

Using SIFT for Matching “Objects”





SIFT: Scale-Invariant Image Transform

- SIFT inspired methods:
  - PCA-SIFT:
    - Reduce dimensionality, only keeps 20 dimension out of 128.
  - SURF:
    - Inspired by SIFT, but uses box filters (Haar like filters) with Integral Image implementation for fast calculation.
    - Has similar results as SIFT, but more sensitive to viewpoint and illumination changes.
  - …

Y. Ke and R. Sukthankar, *“PCA-SIFT: A More Distinctive Representation for Local Image Descriptors,”* Proc. Conf. Computer Vision and Pattern Recognition, pp. 511-517, 2004.
H. Bay, T. Tuytelaars, L. Van Gool "*SURF: Speeded Up Robust Features*", Proceedings of the 9th European Conference on Computer Vision, Springer LNCS volume 3951, part 1, pp 404--417, 2006.

Basic Image Processing Algorithms

PPKE-ITK

Lecture 12.

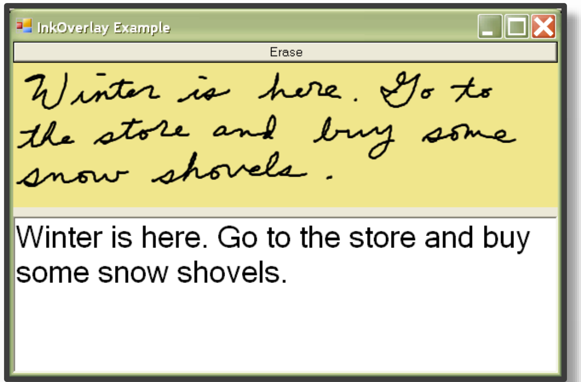
Introduction to Machine Learning

What is Machine Learning?

- Face detection

- E-mail spam filter
- Page ranking in Google search
- Road sign recognition in cars

- Advertisements on web pages and other recommendation systems
- Handwriting recognition



Winter is here. Go to
the store and buy some
snow shovels.

Winter is here. Go to the store and buy
some snow shovels.

- Credit card fraud detection

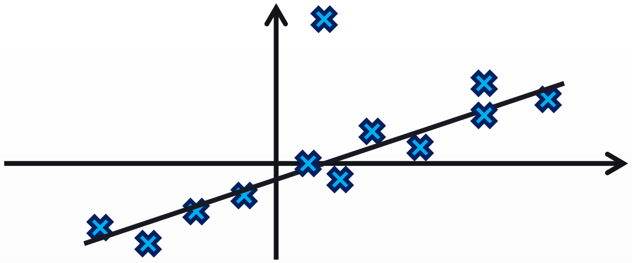
What is Machine Learning?

- Arthur Samuel (1959): "Field of study that gives computers the ability to learn without being explicitly programmed".

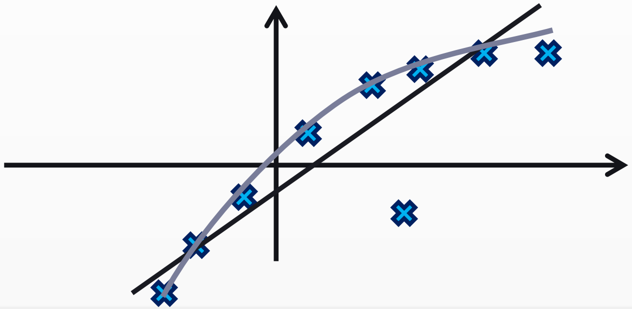
- Tom M. Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

Machine Learning Algorithms

- Supervised Learning:
  - The supervised algorithms are trained on labeled data, where the desired output is known. The goal is to train a classifier that can work on previously unknown data.
    - **Regression**: prediction of continuous valued output



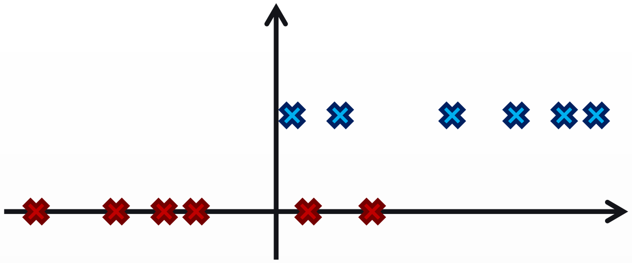
Y: Stock prices
X: Company sells data



What is the right model?

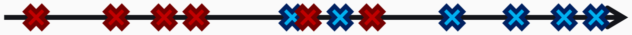
Machine Learning Algorithms

- Supervised Learning:
  - The supervised algorithms are trained on labeled data, where the desired output is known. The goal is to train a classifier that can work on previously unknown data.
    - **Classification**: prediction of discrete valued output



Y: Spam/Not Spam
X: frequency of a certain word
or
Y: Face/Not Face
X: Haar-feature

We can represent the problem in a different way

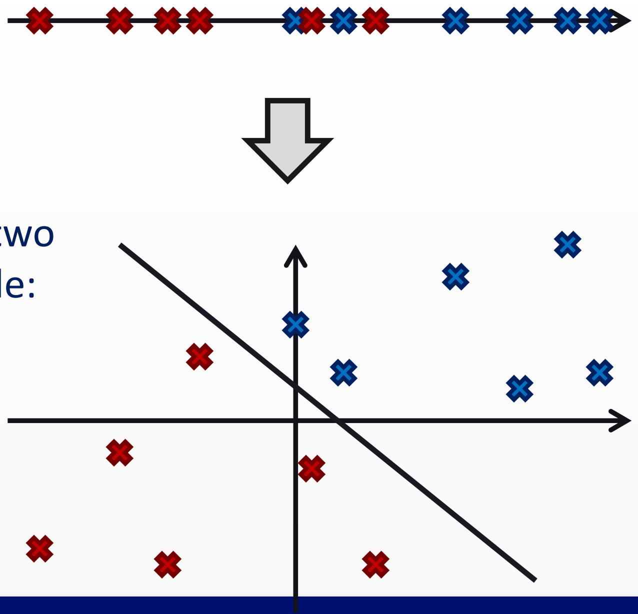


Based on only one feature we cannot make a good decision

Machine Learning Algorithms

- Supervised Learning:
  - The supervised algorithms are trained on labeled data, where the desired output is known. The goal is to train a classifier that can work on previously unknown data.
    - **Classification**: prediction of discrete valued output

In 1D the problem is not separable:



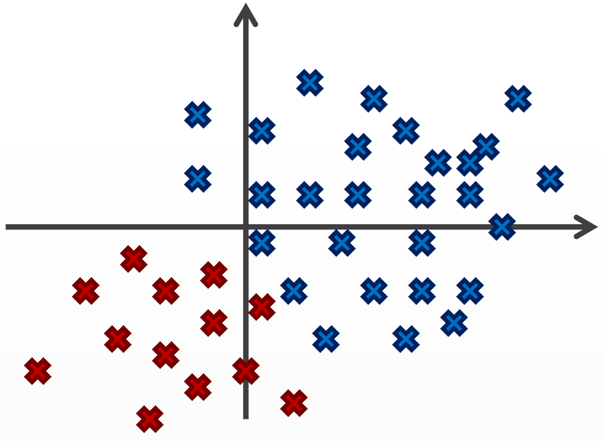
By adding a new feature, the two classes are becoming separable:

Spam/Not Spam
X: frequency of a certain word
Y: frequency of another word
or
Face/Not Face
X: Haar-feature
Y: Haar-feature II.

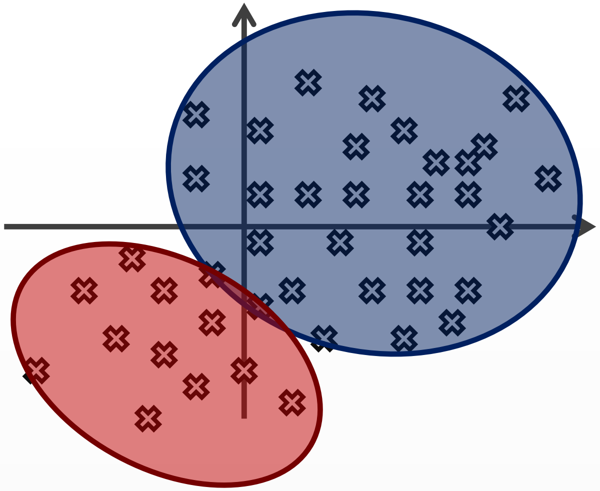
Machine Learning Algorithms

Unsupervised Learning

- In case of unsupervised learning the training data is not labeled.



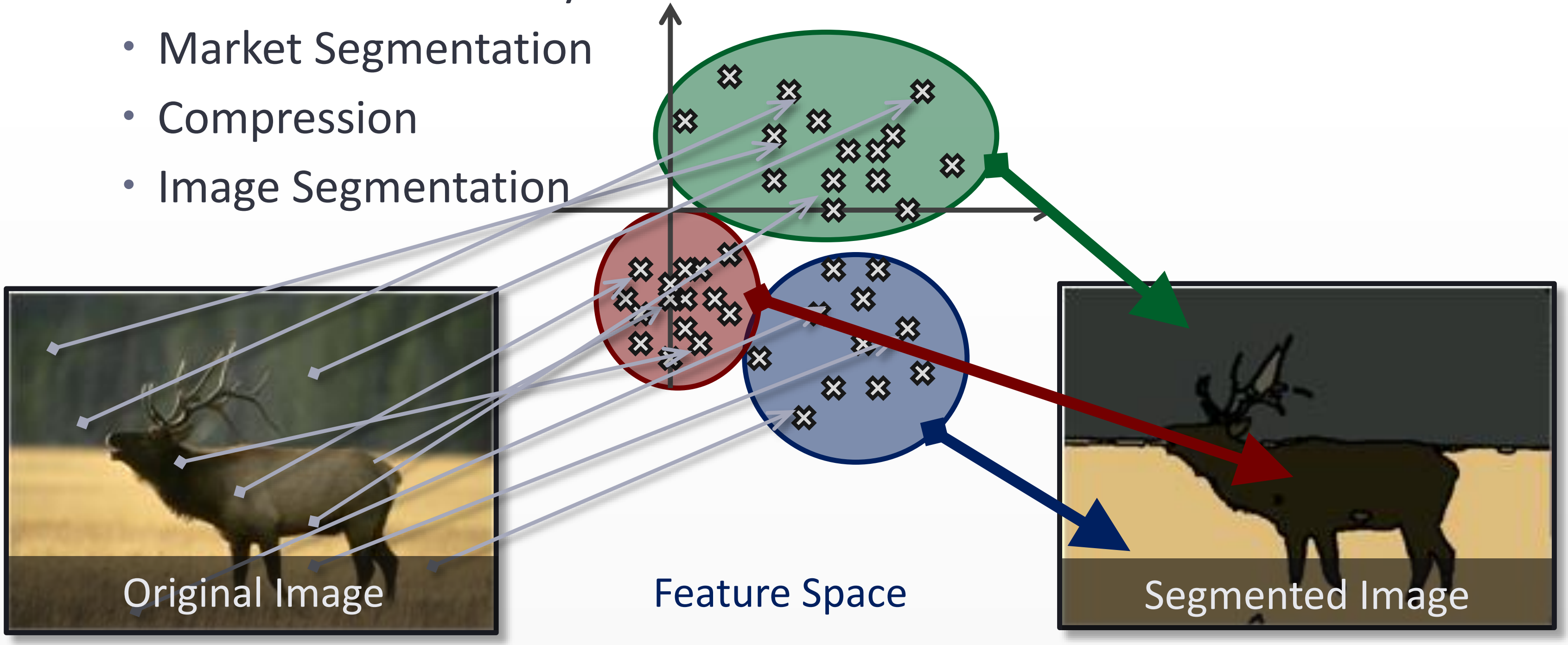
Supervised learning



Unsupervised learning

Machine Learning Algorithms

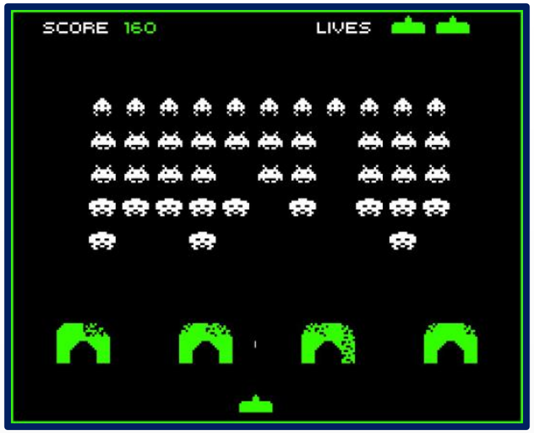
- Unsupervised Learning
  - The goal is to find meaningful structure in the data.
  - Applications:
    - Social Network Analysis
    - Market Segmentation
    - Compression
    - Image Segmentation

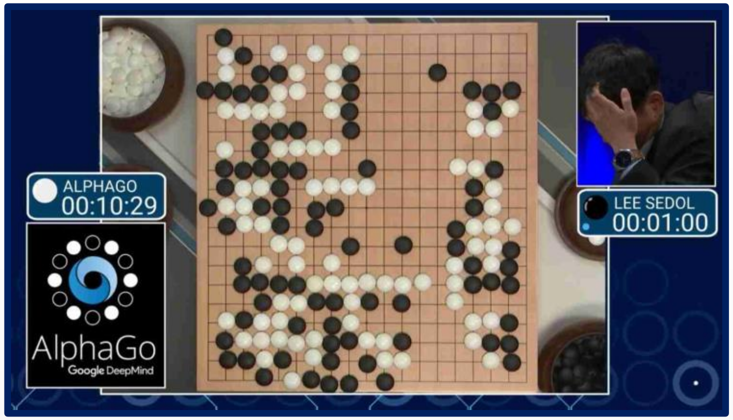


Source of the Images: http://ivrgwww.epfl.ch/supplementary_material/RK_CVPR09/

Machine Learning Algorithms

- Reinforcement Learning
  - The goal is to get an agent to act in the world so as to maximize its rewards.
- Recently very hot topic:
  - Computers can automatically learn to play ATARI games...
  - ...can beat humans in go (AlphaGo)
  - ...can learn to walk





Supervised Learning

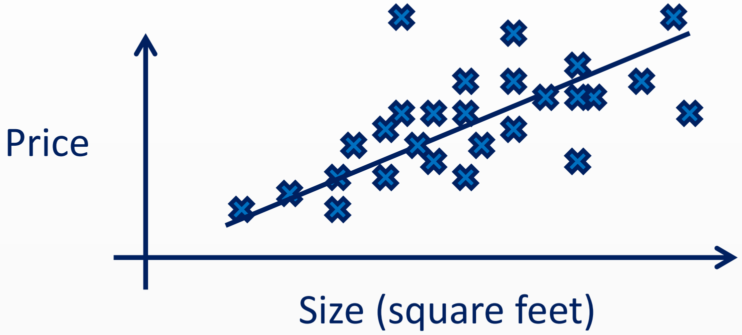
Linear Regression

- Example: Housing Prices
  - It is a supervised learning problem: we have data with ground truth.
  - We know the size of the houses and the price they were sold for:

Training data:

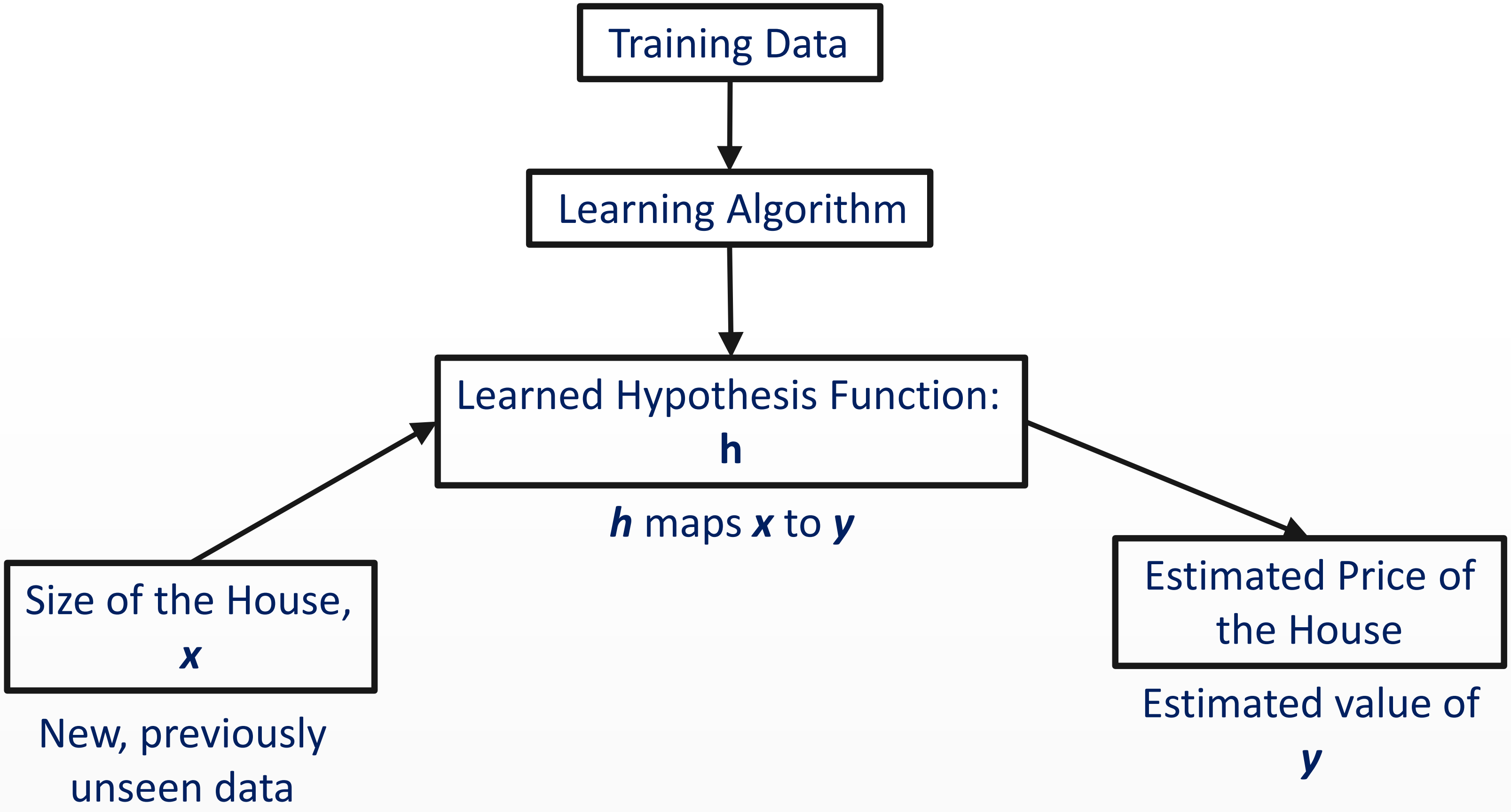
Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

First training example:
$(x^{(1)}, y^{(1)})$



Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Summarization of a Learning Algorithm



Training Data

Learning Algorithm

Learned Hypothesis Function: **h**

h maps ***x*** to ***y***

Size of the House, ***x***

New, previously unseen data

Estimated Price of the House

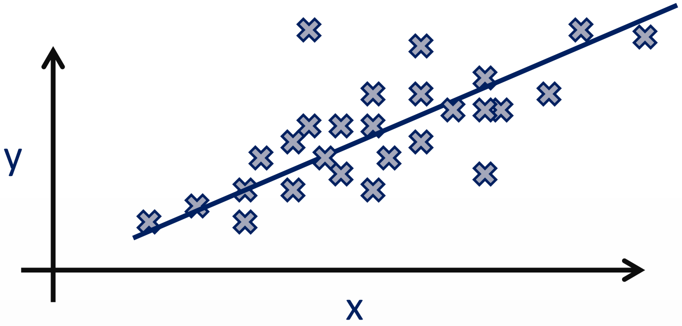
Estimated value of ***y***

Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

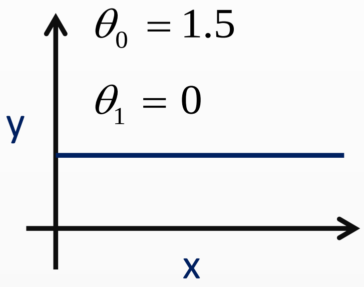
Supervised Learning
Linear Regression

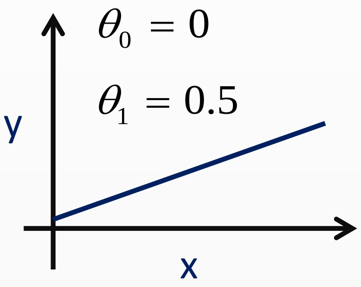
- Representation of ***h*** for linear regression with one variable:

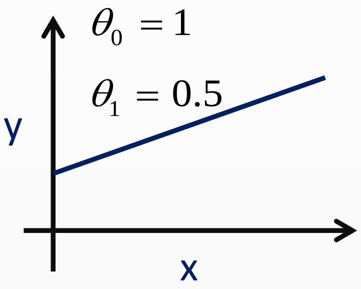
$h_\theta = \theta_0 + \theta_1 x$ → θ_i's are the parameters



- How to find the best values for the parameter θ?







Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Linear Regression

- How to find the best values for the parameter θ?
- Idea: Find parameters (θ_0, θ_1), so that $h_\theta(x)$ is close to y for the training examples.

$$\min_{\theta_0\theta_1} \frac{1}{2m}\sum_{i=1}^{m}\left(\underbrace{h_\theta\left(x^{(i)}\right)}_{h_\theta\left(x^{(i)}\right)=\theta_0+\theta_1 x^{(i)}} - y^{(i)}\right)^2$$

where ***m*** is the number of training examples

- The conventional notation of the above expression:

$$\min_{\theta_0\theta_1} J(\theta_0, \theta_1)$$

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$J(\theta_0, \theta_1)$ is called the **cost function**

Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Linear Regression

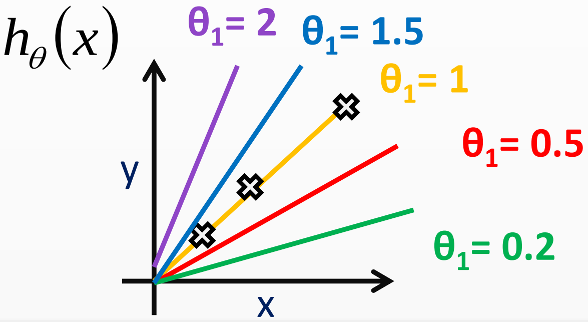
- We have a hypothesis function to map the features to the labels: x to y, house size to price, etc.

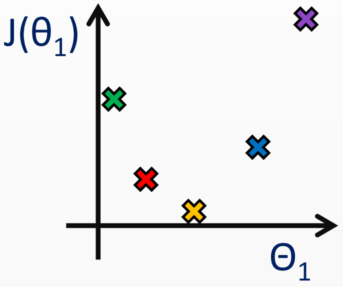
$$h_\theta = \theta_0 + \theta_1 x$$

- The hypothesis function has parameters (θ_0, θ_1), which are optimized during the training by the minimization of the cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

- Simplified example (with $\theta_0 = 0$):



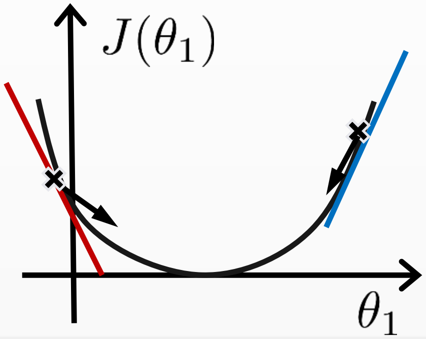


Gradient Descent

- Gradient Descent method will be used to find the minimum of $J(\theta_0, \theta_1)$:
  - Start with ***arbitrary initial values*** (e.g. $\theta_0 = 0,\ \theta_1 = 0$)
  - In each iteration change θ_0 and θ_1 so that J is reduced, until it reaches its minimum value. To achieve this the following ***update rule*** is used:

$$\theta_j = \theta_j - \underset{\text{learning rate}}{\alpha} \underbrace{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}_{\text{derivative of J}} \qquad \text{for } j\text{=0,1}$$

  - The update is done simultaneously for all the θ_j.
- Intuition in 1D:

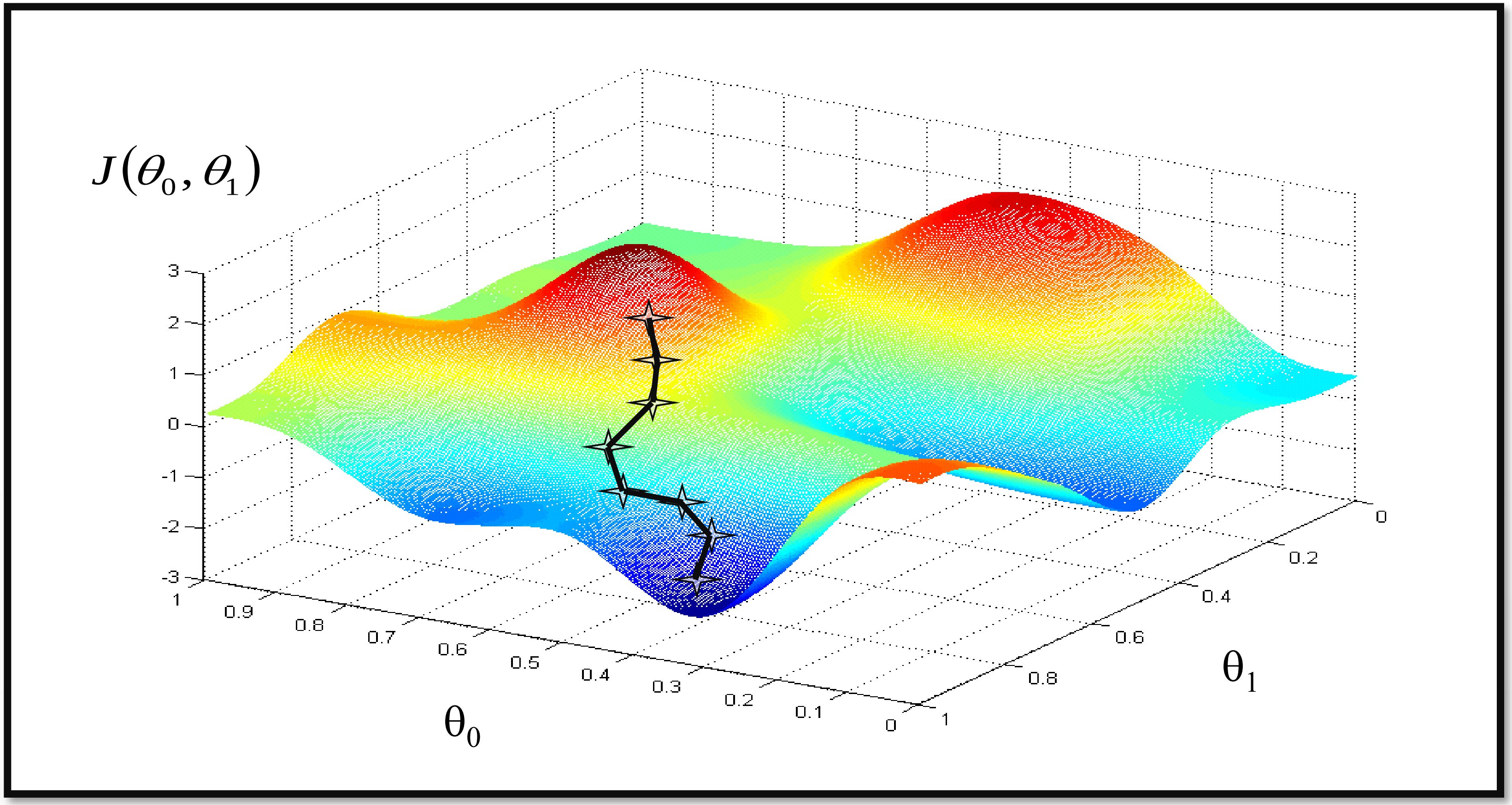


The tangential has a positive slope, the derivative is positive, θ will be decreased.

The tangential has a negative slope, the derivative is negative, θ will be increased.

Gradient Descent

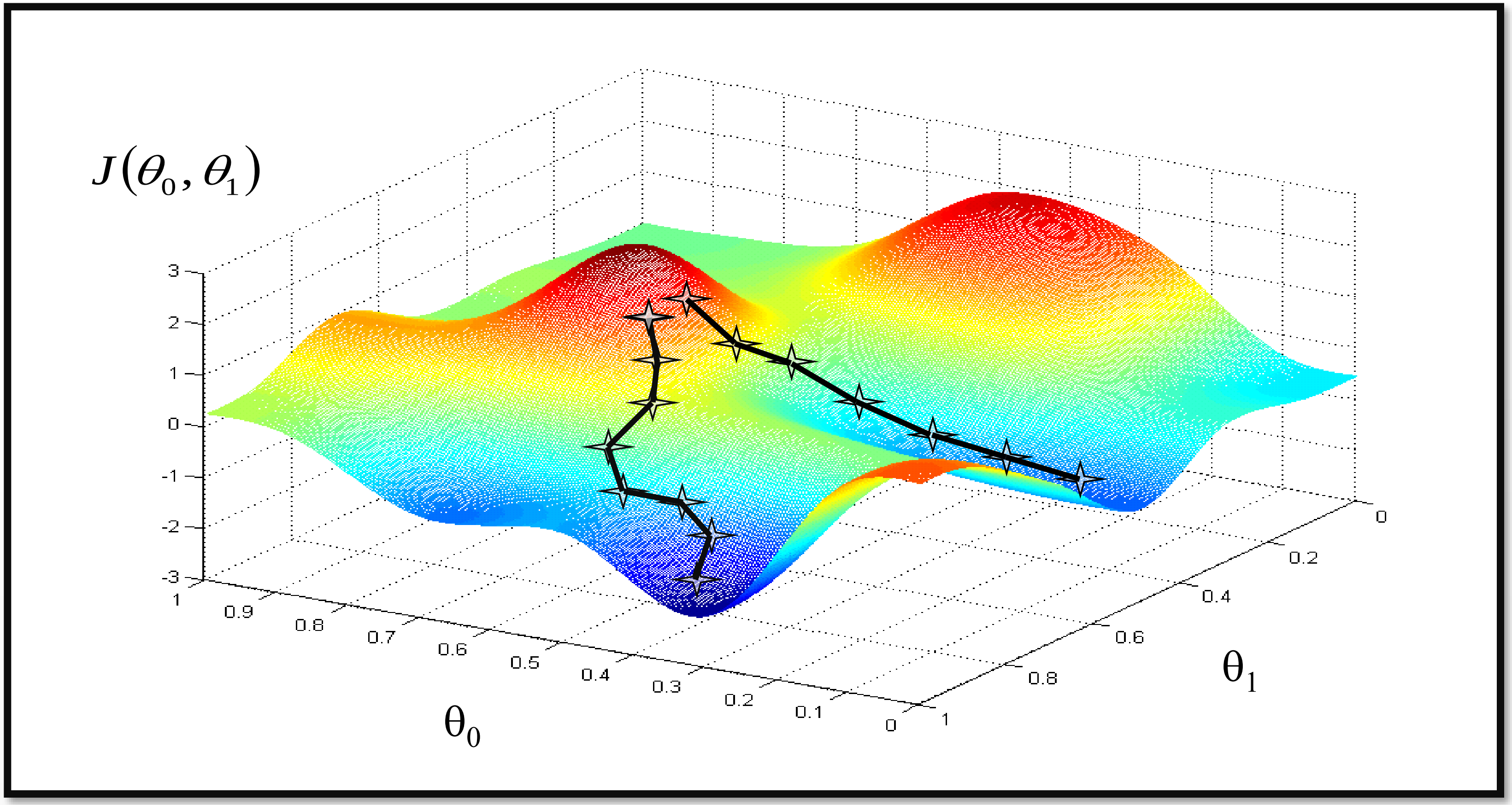
- In each iteration we move toward the minimum:



Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Gradient Descent

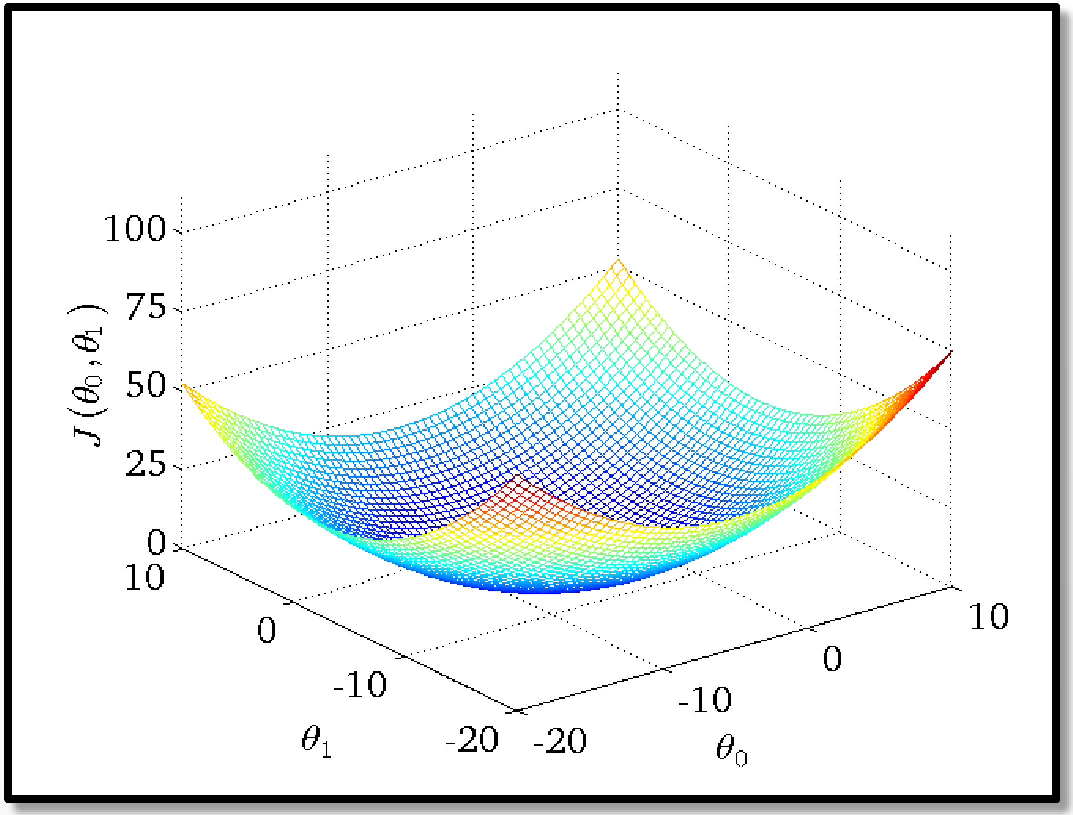
- In each iteration we move toward the (local) minimum:



Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Linear Regression

- With a convex *J* function, there is only one minimum, the global minimum:



Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Linear Regression

- Linear regression can be more powerful with **multiple variables**:
  - Size of the house, # bedrooms, age, # floors, …
  - The new hypothesis function

$$h_\theta(x_1, x_2, x_3, \ldots, x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

  - More convenient to write it in a matrix-vector form:

$$h_\theta(x) = \theta^T \cdot x = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_n \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

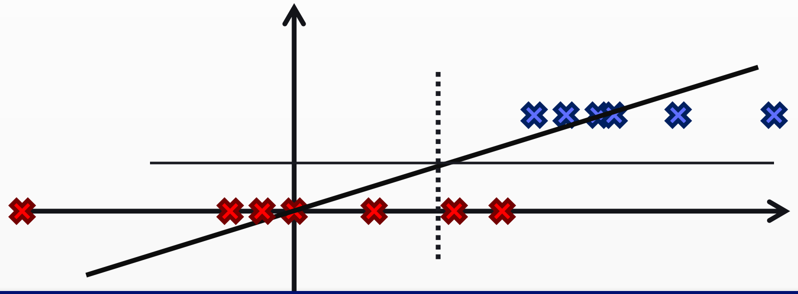
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

  - Features may have different scale (#bedrooms: 1-5, size of the house: 500-2000). Scaling the features to the same range + normalizing the mean often helps the learning algorithm to perform better.

Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Logistic Regression

- Supervised classification algorithm:
  - From the input features (x) it predicts a discrete output (y):
    - Face/Not Face, Spam/Not Spam, …
  - In the training data y is a vector of 0's and 1's:
    - 0 denotes that the samples belongs to the negative class: not face, not spam, …
    - 1 denotes that the samples belongs to the positive class: face, spam, …
  - There are multiclass classification problems with N different classes, where y = 1,2,3,..N. (e.g. car recognition: Opel, Honda, Peugeot,…)
  - Can we use linear regression for this problem?



$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$

$$h_\theta(x) < 0.5 \rightarrow y = 0$$

Logistic Regression

- Logistic regression produces answers between [0,1]:

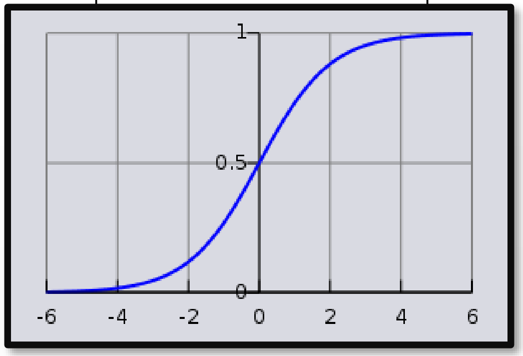
$$0 \leq h_\theta(x) \leq 1$$

- To achieve this we take the logistic function of $\theta^T x$:

$$h_\theta(x) = g(\theta^T \cdot x) \quad \text{where} \quad g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T \cdot x}}$$

Logistic or sigmoid function



- Interpretation of the hypothesis:
  - If for some x, $h_\theta(x) = 0.8$, it means that x, has 80% probability to belong to the positive class:

$$h_\theta(x) = P(y=1 \mid x;\theta) = 0.8$$

$$\rightarrow P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta) = 0.2$$

Decision Boundary

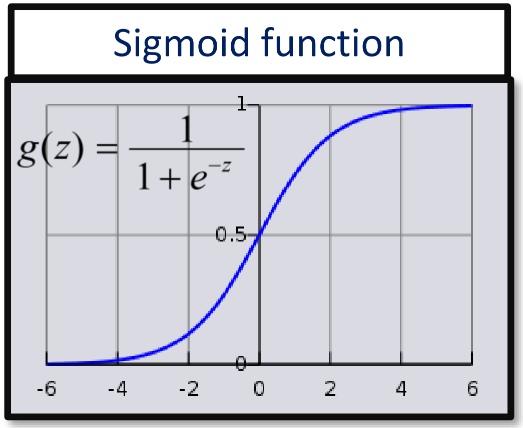
- Interpretation of the hypothesis:
  - To predict binary class labels we use a threshold 0.5:

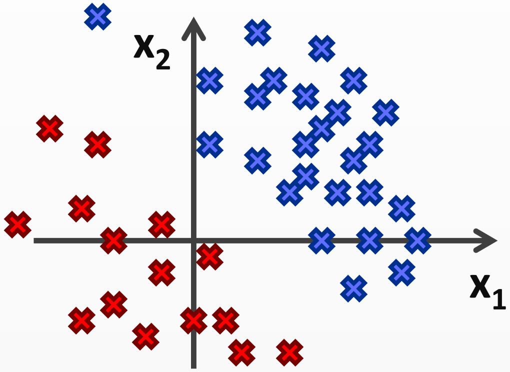
$$P(y=1\mid x;\theta)\geq 0.5 \rightarrow y=1$$
$$P(y=1\mid x;\theta)< 0.5 \rightarrow y=0$$

  - Using a sigmoid function this mean:

$$g(z)\geq 0.5 \quad \text{when} \quad z\geq 0 \quad \Rightarrow \quad \theta^T\cdot x\geq 0$$
$$g(z)< 0.5 \quad \text{when} \quad z< 0 \quad \Rightarrow \quad \theta^T\cdot x< 0$$

  - How can we classify our dataset, assuming we have the trained parameters θ?

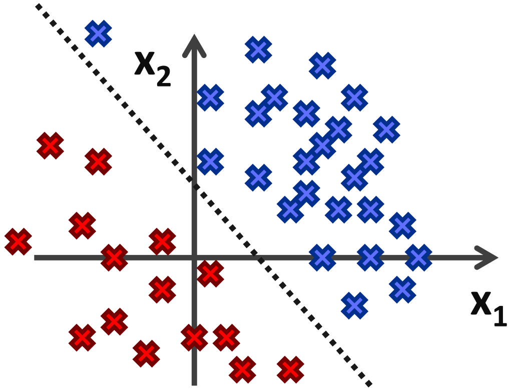

Sigmoid function



Decision Boundary

- Interpretation of the hypothesis:
  - Example I:
    - We have the following hypothesis function:

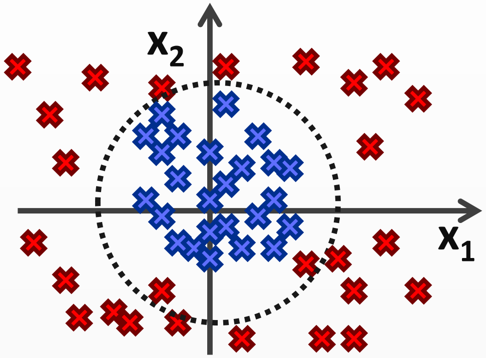
$$h_\theta(x) = g\left(\theta^T x\right) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



Decision boundary:
$x_1+x_2=3$

    - With parameters: $\theta_0 = -3$, $\theta_1 = 1$, $\theta_2 = 1$
    - We predict „1" if $-3 + x_1 + x_2 \geq 0 \;\Rightarrow\; x_1 + x_2 \geq 3$
    - We predict „0" if $-3 + x_1 + x_2 < 0 \;\Rightarrow\; x_1 + x_2 < 3$
  - Example II:
    - The decision boundary can be nonlinear

$$h_\theta(x) = g\left(\theta^T x\right) = g\left(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2\right)$$



    - With parameters: $\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

Decision boundary:
$x_1^2+x_2^2=1$

Logistic Regression

- How to chose parameter θ?
  - We have *m* training examples: $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, …, $(x^{(m)}, y^{(m)})$
  - Each training example has an *n* dimensional feature vector *x* and a label *y*.
  - The hypothesis function is $h_\theta(x) = \frac{1}{1 + e^{-\theta^T \cdot x}}$
  - What cost function should we use?
    - In linear regression the cost function was the following:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

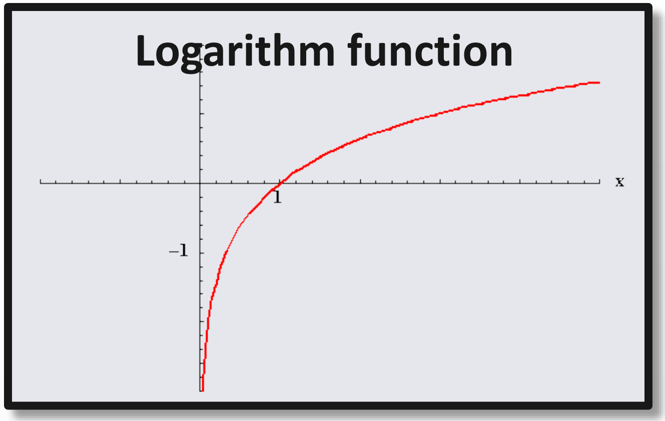
    - The problem is now we have a nonlinear hypothesis function and if we plug it into J(θ) the result will be a non-convex cost function.
    - We need to replace the $\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$

Logistic Regression

- How to chose parameter θ?
  - We will use the following cost term:

$$\mathrm{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if} \quad y = 1 \\ -\log(1 - h_\theta(x)) & \text{if} \quad y = 0 \end{cases}$$

  - If **y=1**:
    - The cost is equal to zero if $h_\theta(x) = 1$, and as $h_\theta(x)$ goes to 0, the cost goes to infinity.
  - If **y=0**:
    - The cost is equal to zero if $h_\theta(x) = 0$, and as $h_\theta(x)$ goes to 1, the cost goes to infinity.


Logarithm function

- The unified cost function of logistic regression is as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)$$

Softmax Regression

- If the classification problem is not binary, e.g.:
  - Cat, Dog, Car, Airplain, Boat, etc.
  - Handwritten digits/characters
  - Facial expressions

Mutually exclusive categories

- The training set is {($x^{(1)}$,$y^{(1)}$), ..., ($x^{(n)}$,$y^{(n)}$)}, where $y^{(i)}$ is in {1,...,K}.
- For multiclass classification there are different strategies:
  - Transformation to Binary:
    - 1 vs. All (need K classifiers)
    - 1 vs. 1 (need K*(K-1)/2 classifiers)
  - Extension from Binary:
    - Softmax
    - ...
  - Hierarchical

Onehot encoding:

$$\begin{array}{ccc} y=1 & \Rightarrow & \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \\ y=2 & \Rightarrow & \begin{bmatrix} 0 & 1 & \cdots & 0 \end{bmatrix} \\ & \vdots & \vdots \\ y=K & \Rightarrow & \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \end{array}$$

http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Softmax Regression

- For K exclusive categories we can use Softmax classifier, where the hypothesis function maps the input x to the following a K-dimensional hypothesis vector:

$$h_\theta(x) = \frac{1}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}^{(j)T} \cdot x}} \begin{bmatrix} e^{\boldsymbol{\theta}^{(1)T} x} \\ e^{\boldsymbol{\theta}^{(2)T} x} \\ \vdots \\ e^{\boldsymbol{\theta}^{(K)T} x} \end{bmatrix}$$

Now θ is a matrix and each of its columns is the parameterization of the x feature vector for one class.

$$\boldsymbol{\theta} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\theta}^{(1)} & \boldsymbol{\theta}^{(2)} & \cdots & \boldsymbol{\theta}^{(K)} \\ | & | & & | \end{bmatrix}$$

- The kth element the hypothesis vector can be interpreted as probability of membership of the kth category:

$$\mathrm{P}(\mathrm{y}_\mathrm{i} = \mathrm{k} \mid \mathrm{x}_\mathrm{i}, \theta) = \frac{\mathrm{e}^{\theta^{(k)T} \mathrm{x}_\mathrm{i}}}{\sum_{\mathrm{j}=1}^{\mathrm{K}} \mathrm{e}^{\theta^{(j)T} \cdot \mathrm{x}_\mathrm{i}}}$$

- Logistic regression can be regarded as a special case (when $K = 2$) of the Softmax classifier.

Softmax Regression

- The cost function (cross-entropy loss) is the following:

$$J(\theta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} 1\{y_i = k\}\log\frac{e^{\boldsymbol{\theta}^{(k)T}x}}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}^{(j)T}\cdot x}} = -\sum_{i=1}^{n}\sum_{k=1}^{K} 1\{y_i = k\}\log P(y_i = k \mid x_i, \boldsymbol{\theta})$$

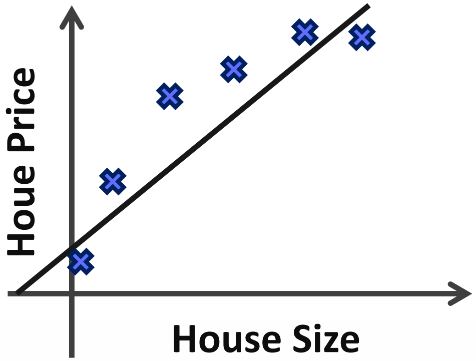
- We cannot solve for the minimum of $J(\theta)$ analytically, and thus as usual we'll resort to an iterative optimization algorithm. Taking the derivatives, one can show that the gradient is:

$$\nabla_{\theta^{(k)}} \mathrm{J}(\theta) = -\sum_{i=1}^{m}\left[\mathrm{x}_i\left(1\{\mathrm{y}_i = \mathrm{k}\} - \mathrm{P}(\mathrm{y}_i = \mathrm{k} \mid \mathrm{x}_i, \theta)\right)\right]$$

  - where $\nabla_{\theta^{(k)}} J(\theta)$ is itself a vector, so that its j-th element is $\frac{\partial J(\theta)}{\partial \theta_{lk}}$ the partial derivative of $J(\theta)$ with respect to the j-th element of $\theta^{(k)}$.
  - Armed with this formula for the derivative, one can then plug it into a standard optimization package and have it minimize $J(\theta)$.

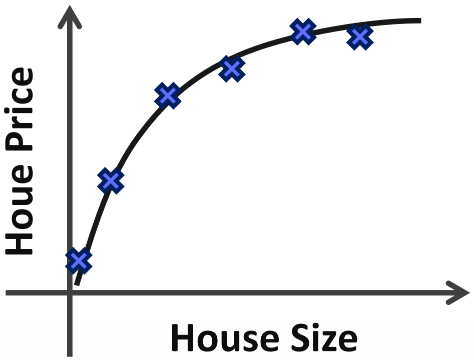
Regularization

- Example: Linear regression



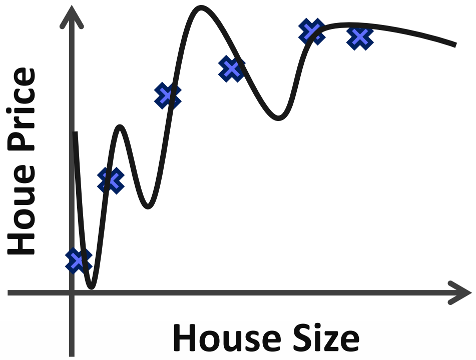
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Underfit or High bias



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Right



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \\ + \theta_3 x^3 + \ldots + \theta_n x^n$$

Overfit or High variance

- If we have too many features we can learn a hypothesis that fits the training data very well, but fails on new samples (= does not generalize well)

Source: Andrew Ng Machine Learning Course on Coursera https://www.coursera.org/course/ml

Regularization

- To handle underfitting we can introduce new features.
- To handle overfitting:
  - We can **reduce the number of features** (but this might mean we lose useful information):
    - We can select manually which features to keep.
    - Use a model selection algorithm.
  - We can apply **regularization**:
    - We can keep all the features but we reduce their magnitude (the value of the θ parameters).
    - Works well if we have a lot of features and each contributes a little bit to predict *y*.
    - The idea is to keep the parameters low, to get a simpler hypothesis function, which is less prone to overfitting.

Regularization

- How can we keep the parameters low?
  - The cost function for linear/logistic regression with regularization:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) + \boxed{\lambda\sum_{j=1}^{n}\theta_j^2}$$

Note: θ_0 is not regularized

L2 regularization term

  - The regularization parameter λ controls the trade-off between two goals:
    - Fitting the data well
    - Keeping the parameters low, to avoid overfitting
  - If λ is too large all the parameters (except θ_0) will be close to 0, the model won’t fit the data, we will see underfitting.

Main Sources

- Andrew Ng Machine Learning Course
  - On Coursera: https://www.coursera.org/course/ml
  - At Stanford: http://cs229.stanford.edu/

- Further reading:
  - Lectures by Nando de Freitas:
    - Undergraduate Machine Learning at UBC 2012:
      https://www.youtube.com/playlist?list=PLE6Wd9FR--Ecf_5nCbnSQMHqORpiChfJf&feature=view_all
    - Machine Learning at UBC 2013
      http://www.cs.ubc.ca/~nando/540-2013/lectures.html
  - A Few Useful Things to Know about Machine Learning:
    http://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf
  - Linear classification Loss Visualization:
    http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

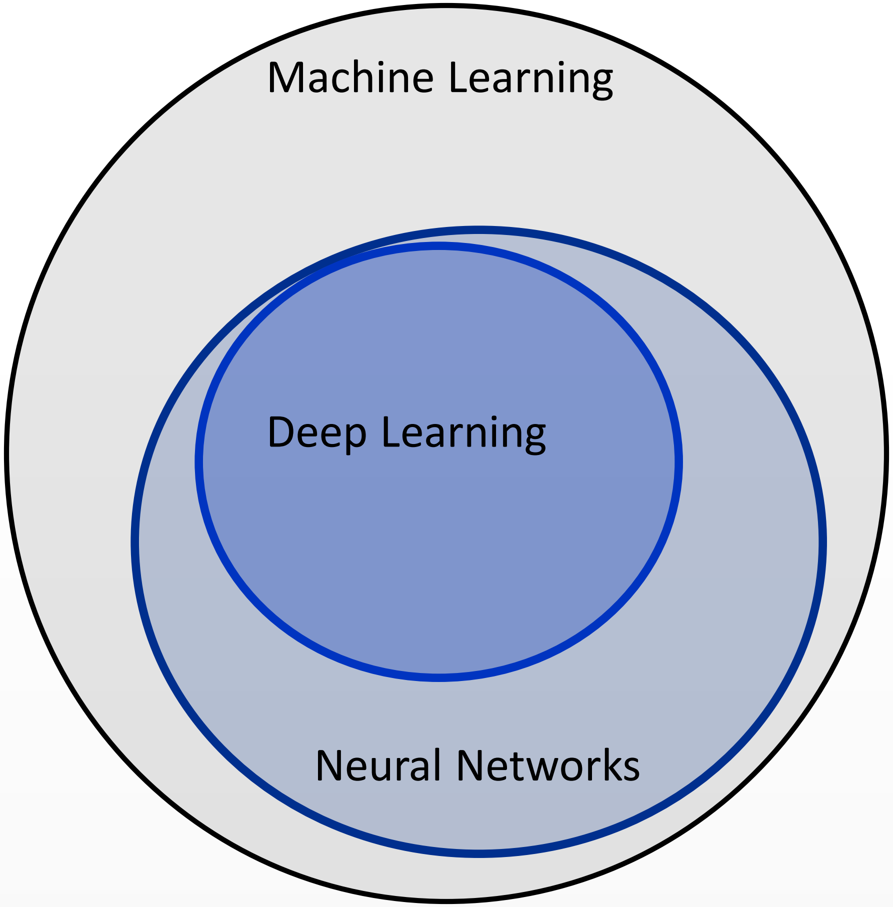
Basic Image Processing Algorithms

PPKE-ITK

Lecture 13.

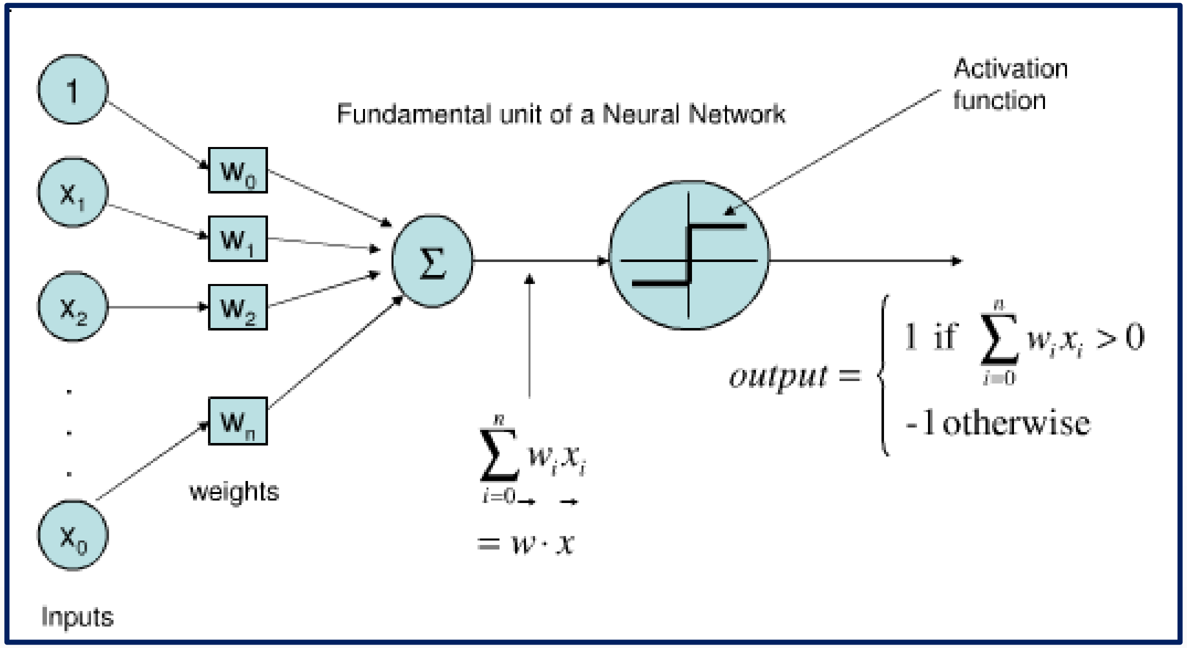
Introduction to Deep Learning

What is Deep Learning?



History of Neural Networks

- 1943: McCulloch and Pitts proposed a model to mimic how the brain operates.
- 1958: Rosenblatt introduced the Perceptron


1
x_1
x_2
x_0
Inputs
w_0
w_1
w_2
w_n
weights
Fundamental unit of a Neural Network
Σ
Activation function
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Perceptron Model

History of Neural Networks

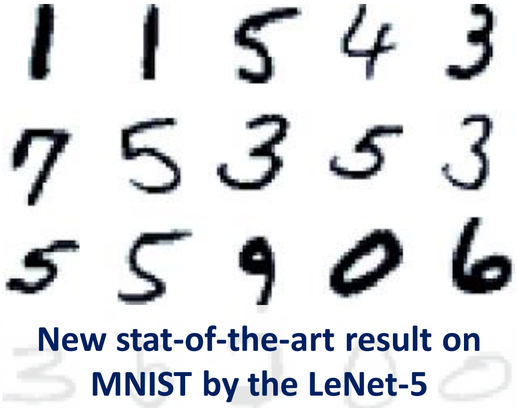
- Thanks to the success of the Perceptron model, there was a big hype around AI (not unlike now!!)
- 1969: It was shown that a perceptron may fail to separate seemingly simple patterns (e.g. cannot learn the XOR function).
- Research in the area nearly stopped completely (AI winter)
- 1974: Paul J. Werbos introduced backpropagation algorithm: efficient training of multi layer networks.

History of Neural Networks

- 1989-98: Convolutional Neural Networks in action:

Lecun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). *Backpropagation applied to handwritten zip code recognition. Neural computation*, *1*(4), 541-551.

LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition„ Proceedings of the IEEE.86 (11): 2278–2324.



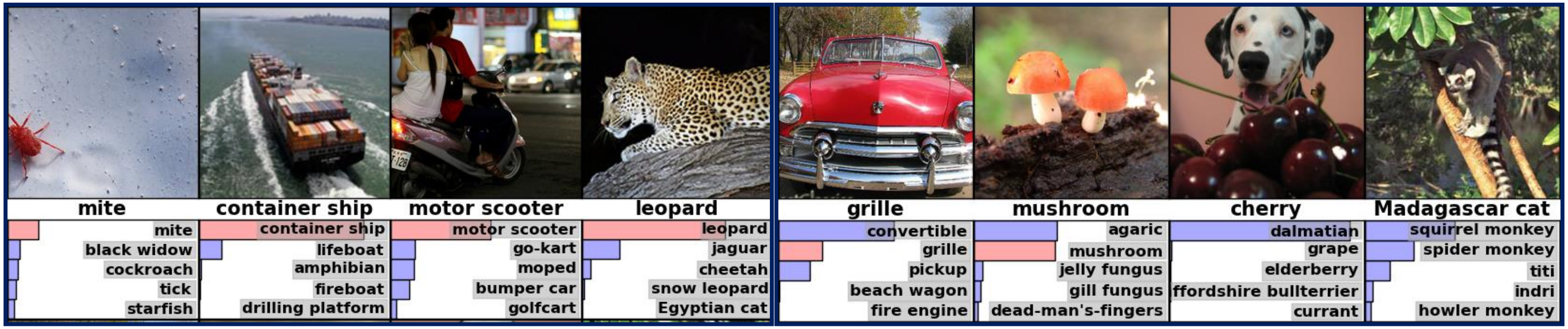
New stat-of-the-art result on MNIST by the LeNet-5

- Since 2012 there is a huge buzz around neural networks again.
- Recap: What happened in 2012?
  - The ImageNet challenge was won by a deep neural network architecture, the AlexNet:

Krizhevsky, A., Sutskever, I. and Hinton, G. E. „*ImageNet Classification with Deep Convolutional Neural Networks*” NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada

ImageNet Challenge 2012

- Task: For each image produce a list of at most 5 object categories in the descending order of confidence.



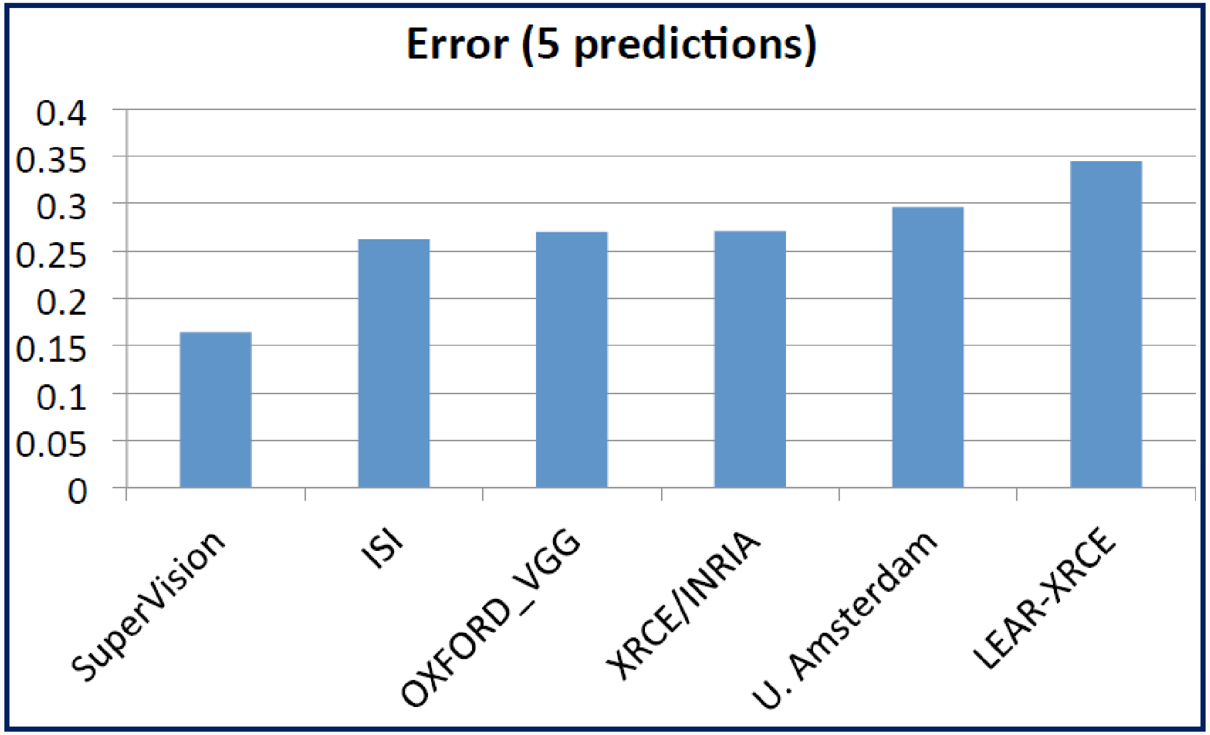
- The training data: 1.2 million images 1000 categories
- Summary of the Challenges:
  - A large number of images in training
  - A large number of classes
  - Diversity of classes
  - Diversity of images within classes

ImageNet Challenge 2012

Test images of „Hammer”

ImageNet Challenge 2012

- Results:

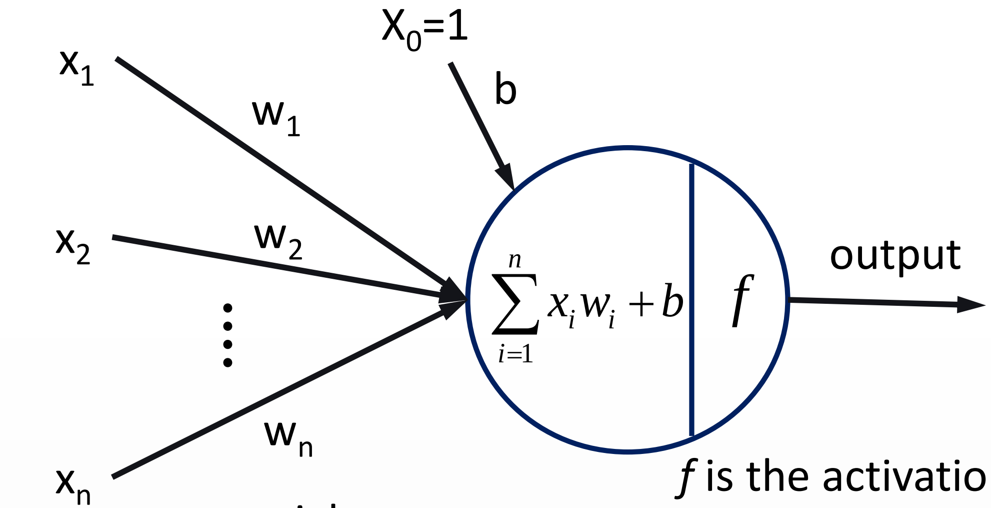


- Since then it is widely used in computer vision (and also in many other fields)

Artificial Neural Networks

Artificial Neural Networks

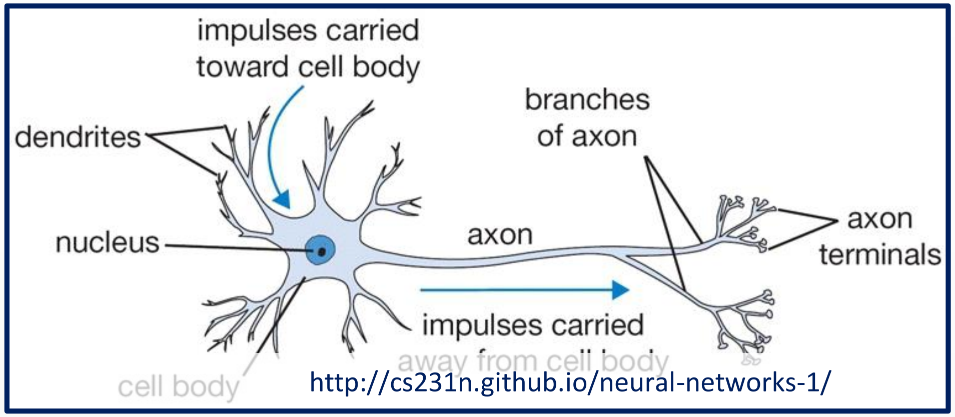
- Simplest ANN: One Neuron



$$y = f(z) = f\left(\sum_{i=1}^{n} x_i w_i + b\right) = f(xw+b)$$

f is the activation function

Note: If *f* is a sigmoid function then this is exactly the logistic regression.

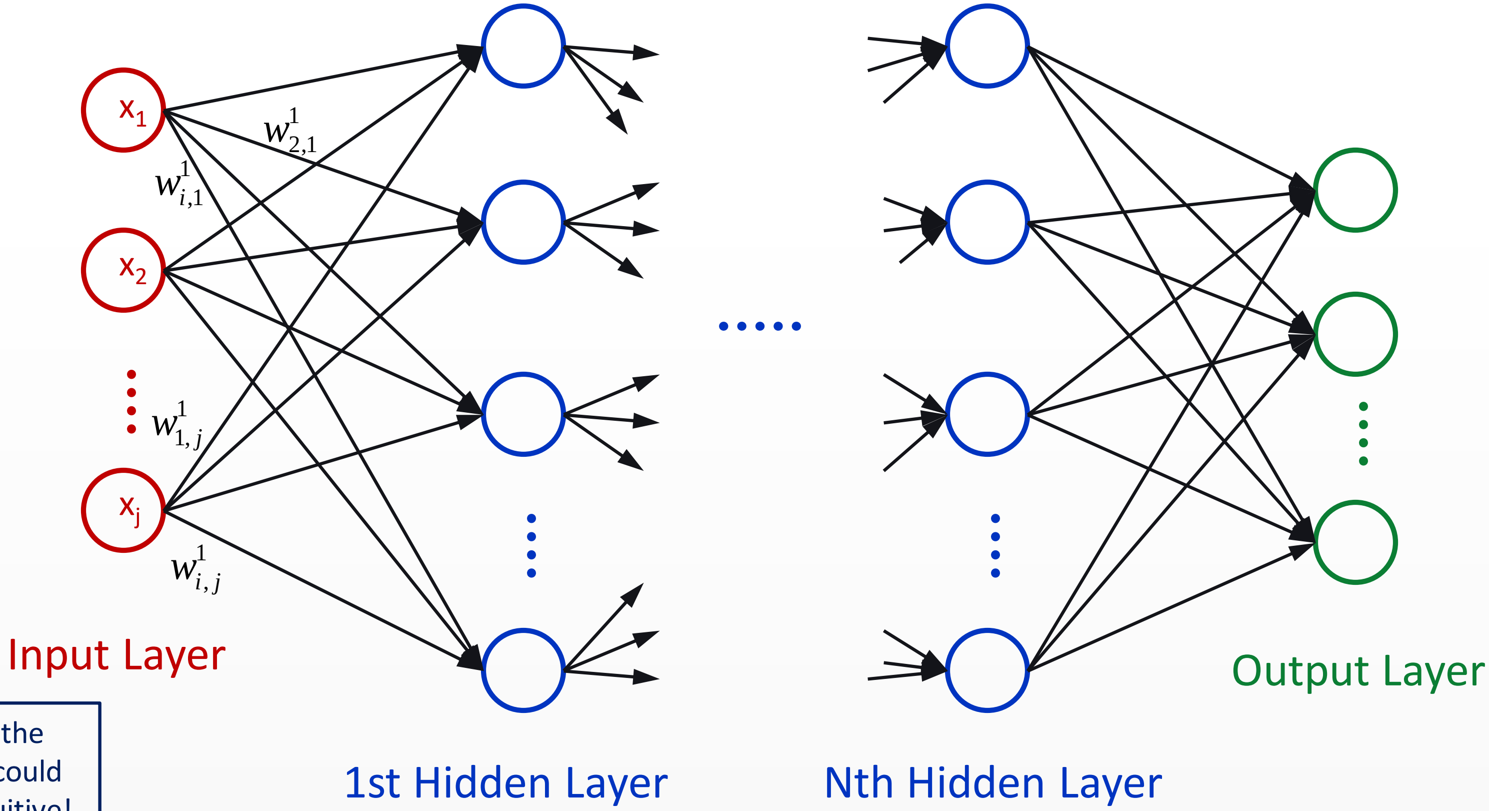


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

http://cs231n.github.io/neural-networks-1/

Artificial Neural Networks

- These simple computational units (the artificial neurons) can be organized into networks. (Like the „real” neurons in the human nervous system)



Input Layer

1st Hidden Layer

Nth Hidden Layer

Output Layer

Watch out for the indexation, it could be counterintuitive!

Artificial Neural Networks

- Feed-Forward network
  - Contains only forward connections: the neurons in layer *l* are only connected to the neurons in layer *l+1*.
  - No backward or within-layer connections.
- Fully Connected layers:
  - ***All*** units in layer *l* are connected to ***all*** units in layer *l+1*.
- Each neuron has a bias connection:
  - Acts as a connection where the input is always 1.
- ANNs are universal function approximators [Cybenko 1989]

  „The universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of R^n." (Wikipedia)

Artificial Neural Networks - Forward Pass

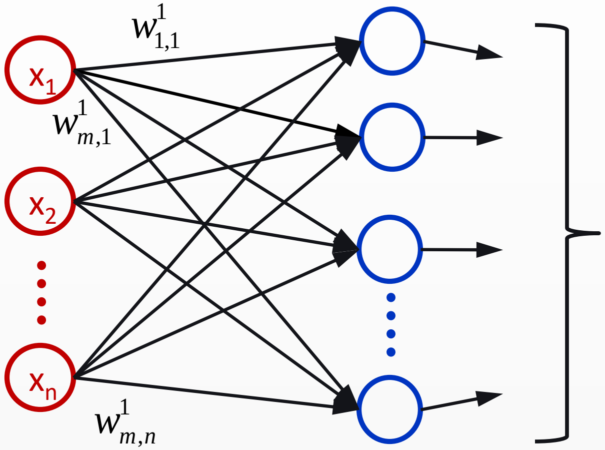
- The *Forward Pass* is the calculation of the response of the network to an input.
- Assuming you have a trained network, all trainable parameters (θ) are tuned for the task:

$F\Big($ $,\theta\Big) = \begin{bmatrix} 0.91 \\ 0.03 \\ 0.06 \end{bmatrix}$

$F\Big($ $,\theta\Big) = \begin{bmatrix} 0.04 \\ 0.95 \\ 0.01 \end{bmatrix}$

$F\Big($ $,\theta\Big) = \begin{bmatrix} 0.15 \\ 0.08 \\ 0.77 \end{bmatrix}$

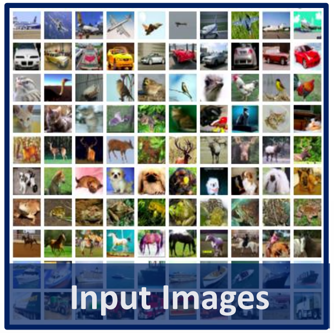
- It can be calculated a series of matrix-vector operation:



$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ \vdots \\ a_m^1 \end{bmatrix} = f\left(\begin{bmatrix} z_1^1 \\ z_2^1 \\ \vdots \\ z_m^1 \end{bmatrix}\right) = f\left(\begin{bmatrix} - & w_1^1 & - \\ - & w_2^1 & - \\ & & \\ - & w_m^1 & - \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_m^1 \end{bmatrix}\right)$$

$$a^{(1)} = f\left(z^{(1)}\right) = f\left(W^{(1)}x + b^{(1)}\right)$$

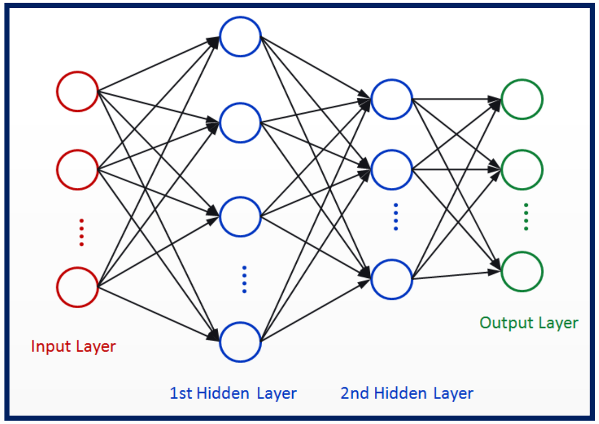
Artificial Neural Networks - Forward Pass



$I \in$ set of images

Extracted Features: HOG/SIFT/Pixels

$$x = \xi(I)$$



$$a^{(1)} = f\left(z^{(1)}\right) = f\left(W^{(1)}x + b^{(1)}\right)$$
$$a^{(2)} = f\left(W^{(2)}a^{(1)} + b^{(2)}\right)$$
$$z^{(3)} = W^{(3)}a^{(2)} + b^{(3)}$$

Predicted Labels: Cat/Car/Dog/Truck

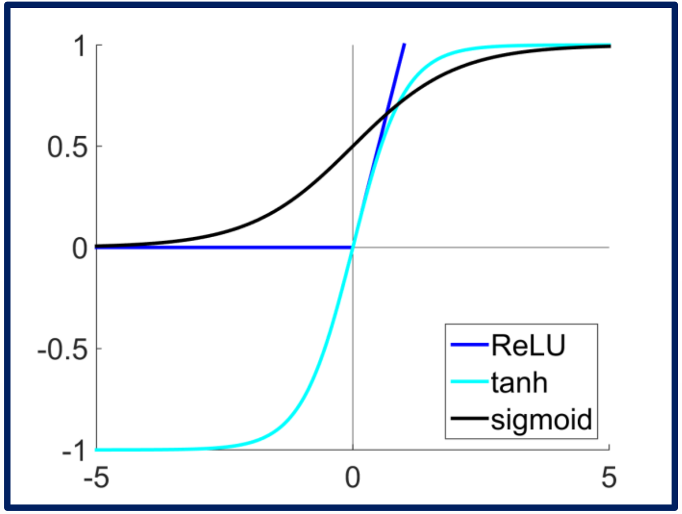
$$\tilde{y} = a^{(3)} = g\left(z^{(3)}\right)$$

g():
can be a softmax function for classification,
or it can be omitted in case of regression.

Artificial Neural Networks - Activation Functions

- The activation function is a non-linear function applied in each neuron on ***z***, the weighted sum of the neuron's input.
- Commonly used activation functions:
  - Sigmoid
  - Hyperbolic tangent (tanh)
  - **Rectified Linear Unit (ReLU, $y = \max(0, x)$)**
  - ...

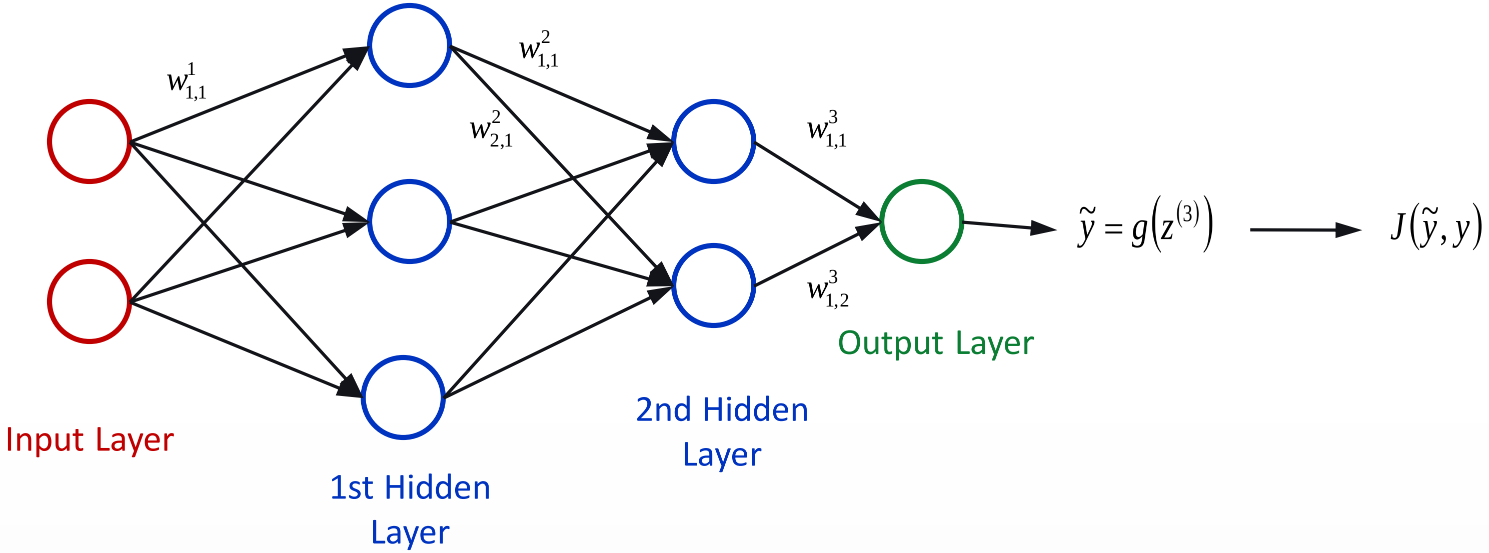
Lately ReLU is the default choice for deep nets.



- What happens if we don't use activation function? (== If the activation function is linear?)
  - The composition of linear functions is a linear function. It would be equivalent to a 1-layer logistic/linear regression.

Network Training

Training of the Network



$$W^{(1)} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & w^1_{1,3} \\ w^1_{2,1} & w^1_{2,2} & w^1_{2,3} \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} b^1_1 & b^1_2 & b^1_3 \end{bmatrix}$$

- Forward propagation: $\tilde{y} = g\left(W^{(3)} f\left(W^{(2)} f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)}\right)$
- Loss function:
  - For classification, we can use cross-entropy loss (see softmax)
  - For regression, we can use L2 loss (see linear regression)
- How can we minimize the loss?

Training of the Network

- To optimize the weights and biases so that the *loss is minimal* we use ***Gradient Descent*** algorithm:
  - The weights and biases are initialized as small random numbers.
  - Each parameter (weights and biases) are modified simultaneously in each iteration:

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \alpha \frac{\partial}{\partial w_{i,j}^{(l)}} J \qquad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J$$

- How can we compute the *gradient of the loss wrt each parameter*?
- With ***Backpropagation algorithm***:
  - It is based on the application of the *chain rule:*

$$F(x) = f(g(x)) \;\Rightarrow\; F'(x) = f'(g(x))g'(x)$$

  - This is the most efficient way to compute the exact gradients.

Training of the Network

- Practical considerations:
  - Initialization with small random numbers (instead of all zeros) to *break the symmetry*, otherwise all the hidden units would learn the same function of the input.
  - Training Strategies:
    - Adjust the weights based on the ...
      - ...error of one training sample (***Stochastic Gradient Descent***)
      - ...average error of all the training samples (***Batch Gradient Descent***)
      - ...average error of a few dozens/hundreds of training samples (***Mini-Batch Gradient Descent***)
        - The average error on the mini-batch usually approximates well the average error on all the training samples, but it is much faster.
        - Still we will use all the training samples many times: after we go through all mini-batches (== we complete one epoch), we reshuffle the samples, divide them into mini-batches again and start the next epoch.

Training of the Network

- Regularization:
  - The goal is to prevent the network from overfitting.
  - Commonly used types of regularization:
    - L2 regularization:
      - An extra term is added to the cost to penalize peaky weights:

$$\frac{\lambda}{2}\|w\|_{L2} = \frac{\lambda}{2}\sum_{j=1}^{n} w_j^2 \longleftarrow \text{L2-norm of the weights}$$

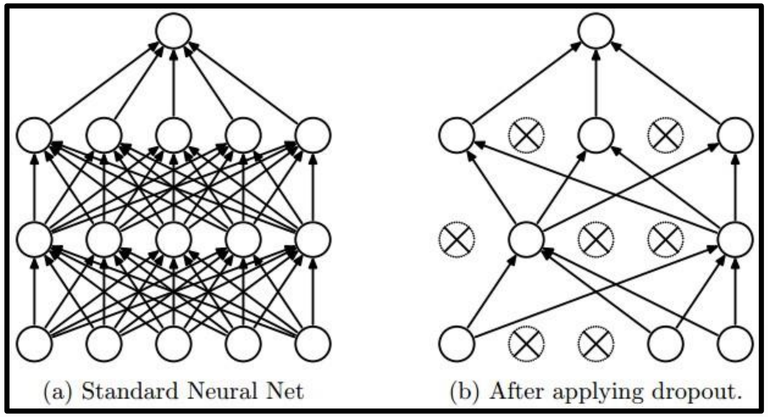
      - This regularization prefers diffuse weights.
    - L1 regularization:
      - Regularization term:

$$\frac{\lambda}{2}\|w\|_{L1} = \frac{\lambda}{2}\sum_{j=1}^{n} |w_j| \longleftarrow \text{L1-norm of the weights}$$

      - Favours sparse weight vectors. (Sparse means that only a few elements in the vector are non-zero)

Training of the Network

- Regularization:
  - Dropout:
    - During training each neuron can be deactivated with a certain probability.
    - In each iteration of the training, a different sub-network is optimized.
    - In test time there is no dropout!



http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

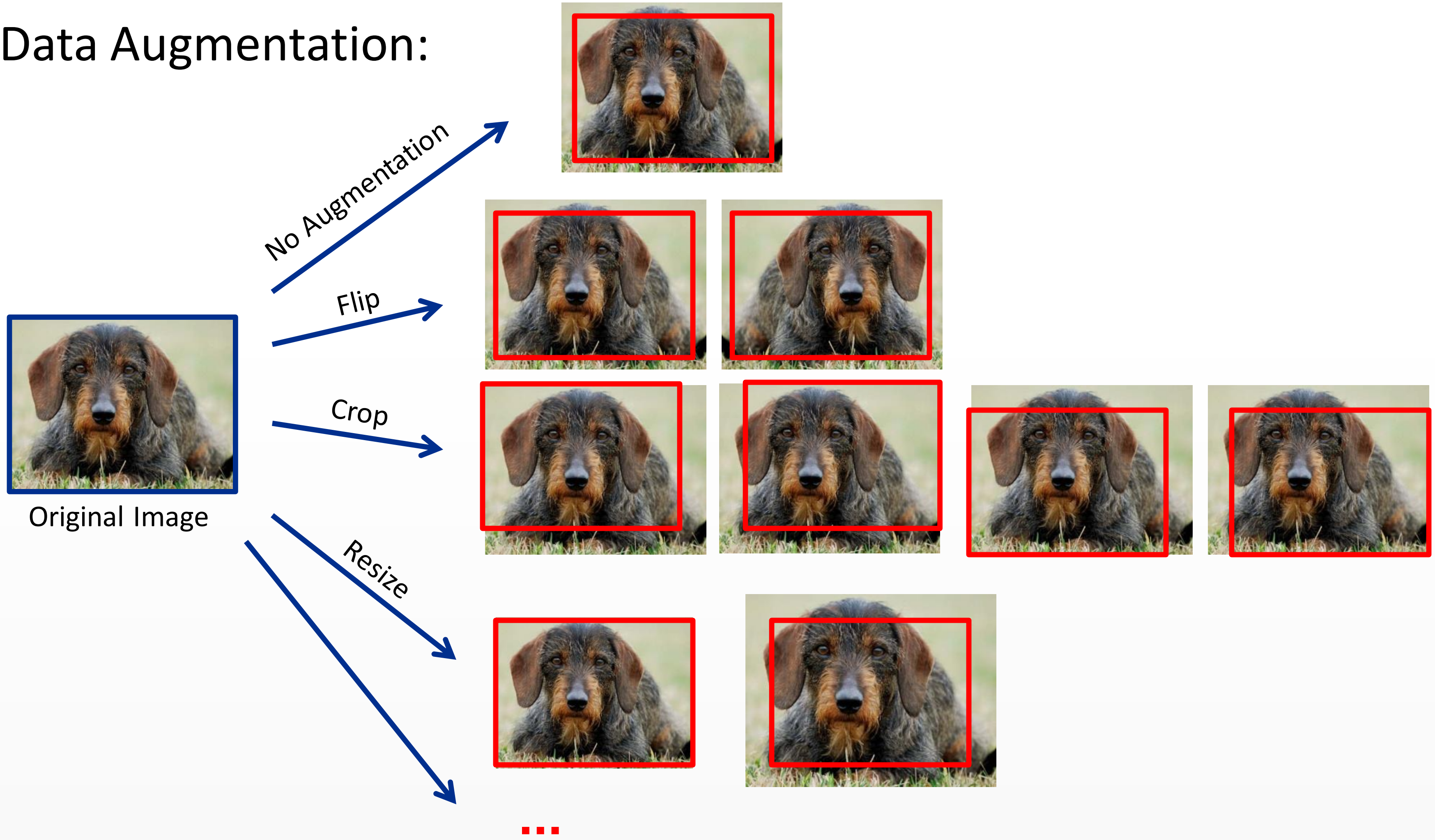
  - Early stopping:
    - Monitor the training:
      calculate performance metrics
      (accuracy, f1-score, ROC-AUC, etc.)
      on a separate validation set after each round.
    - Stop the training if the performance drops on the validation set!

Training of the Network

- In general we can say that the more parameters we use the larger the training dataset needs to be to be able to train without overfitting.
- Creating a large dataset takes time and expensive.
- Have to get the most out of the available data!
- Data Augmentation:
  - Increasing the size of the available training dataset by adding modified versions of the original samples.
- The augmentation has to be task specific (e.g. for handwritten digit recognition, flip is not a good idea..)

Training of the Network

Data Augmentation:



Original Image

No Augmentation

Flip

Crop

Resize

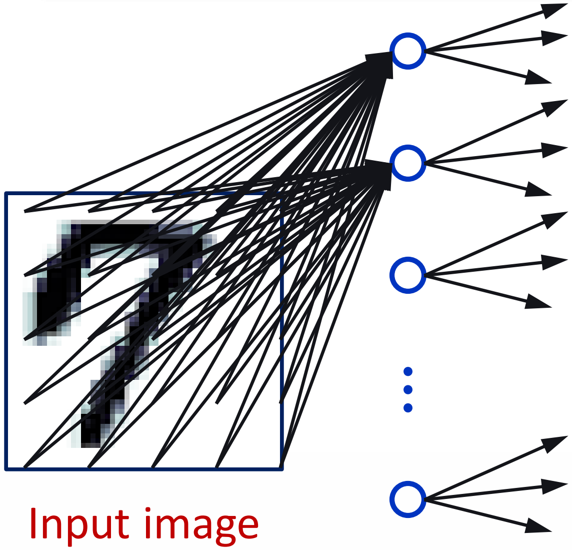
...

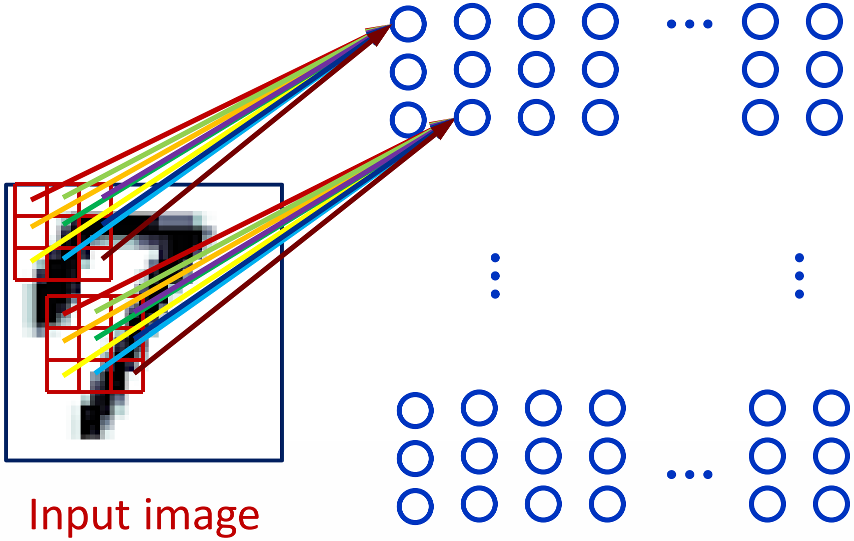
Convolutional Neural Networks

Convolutional Neural Network

- Fully connected layers on the raw pixels are not efficient. Why?
  - Regular neural nets don’t scale well to full images:
    - For a small image of 128x128 we have 16384 pixels.
    - If in the first hidden layer we have 100 neurons that is already 1.6 million parameters!!
  - To tune this many parameters we would need a lot of training samples to avoid overfitting, and it would be slow.
- Natural images...
  - contain strong local correlation, we should take it into account
  - have similar local statistics over the image, we could share the used parameters.
- Convolutional Neural Networks look at small parts of the image, one at a time using the same set of weights for each part.
- For CNN the weights are organized in 3D (width, height, depth).

Convolutional Neural Network





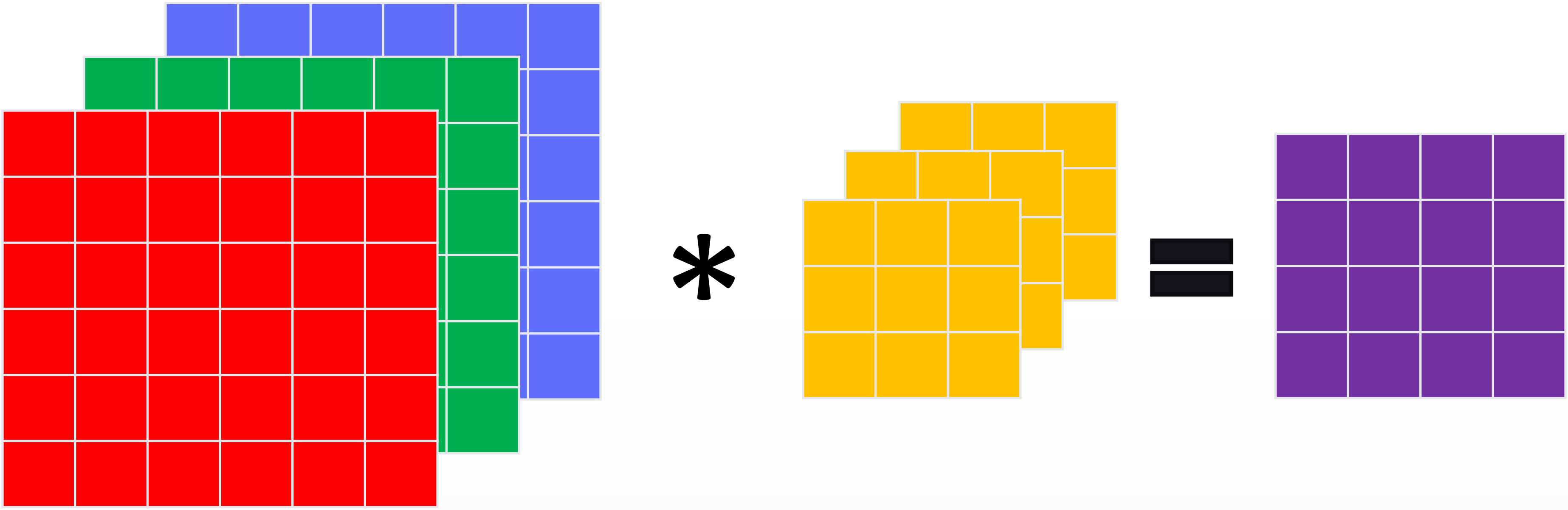
Fully Connected Layer

Convolutional Layer

Each pixel is one input for the network -> the number of parameters can be very high even for a medium sized image.

- The weights are reused in different parts of the image -> much less parameters.
- Practically convolutional kernels will be learned. Each kernel has 3 dimensions (only 2 are visualized above)

Convolution on volumes



Input: n-channel array:

$h \times w \times n$

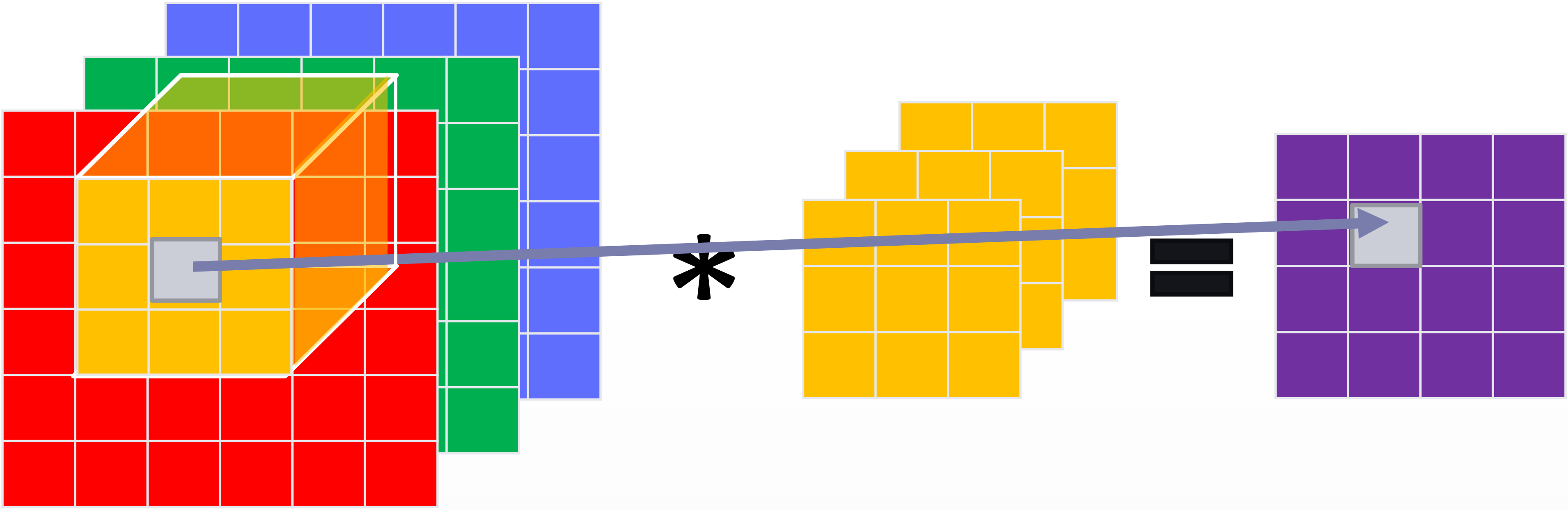
Filter: n-channel kernel of size

$k_h \times k_w \times n$

Output: 1-channel

$(h - k_h + 1) \times$
$(w - k_w + 1) \times 1$

Convolution on volumes



Input: n-channel array:

$h \times w \times n$

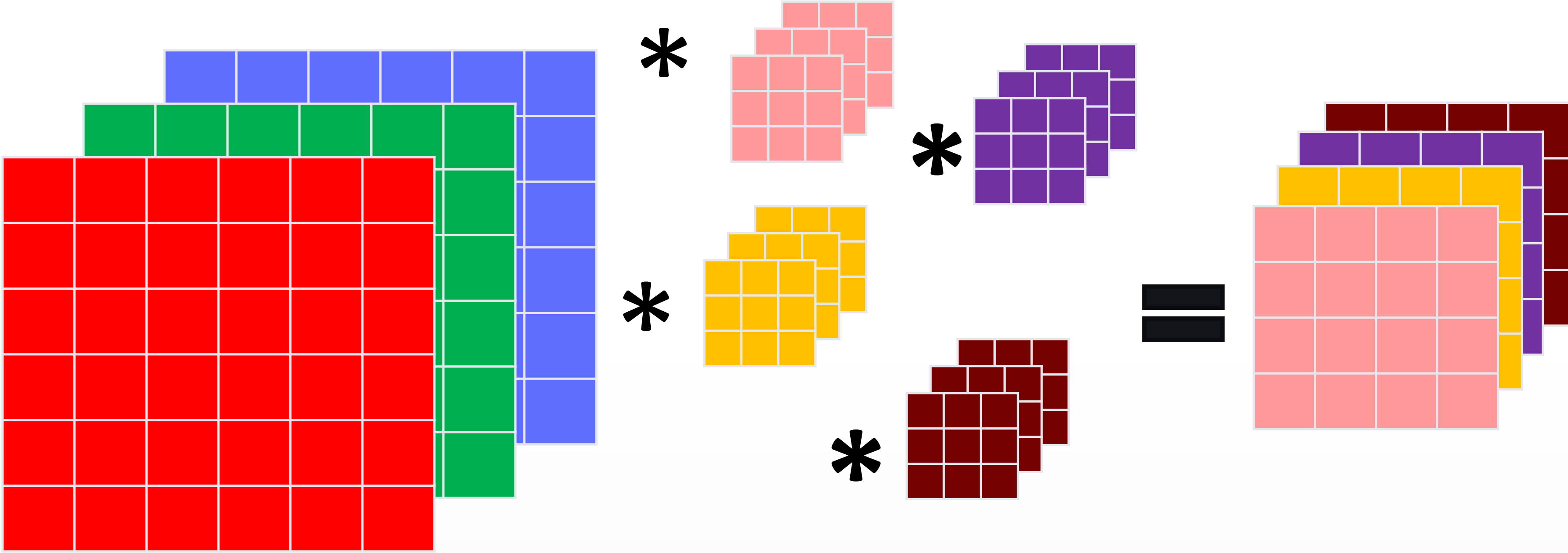
Filter: n-channel kernel of size

$k_h \times k_w \times n$

Output: 1-channel

$(h \ - k_h + 1) \times$
$(w \ - k_w + 1) \times 1$

A convolution layer with $f = 4$ filters



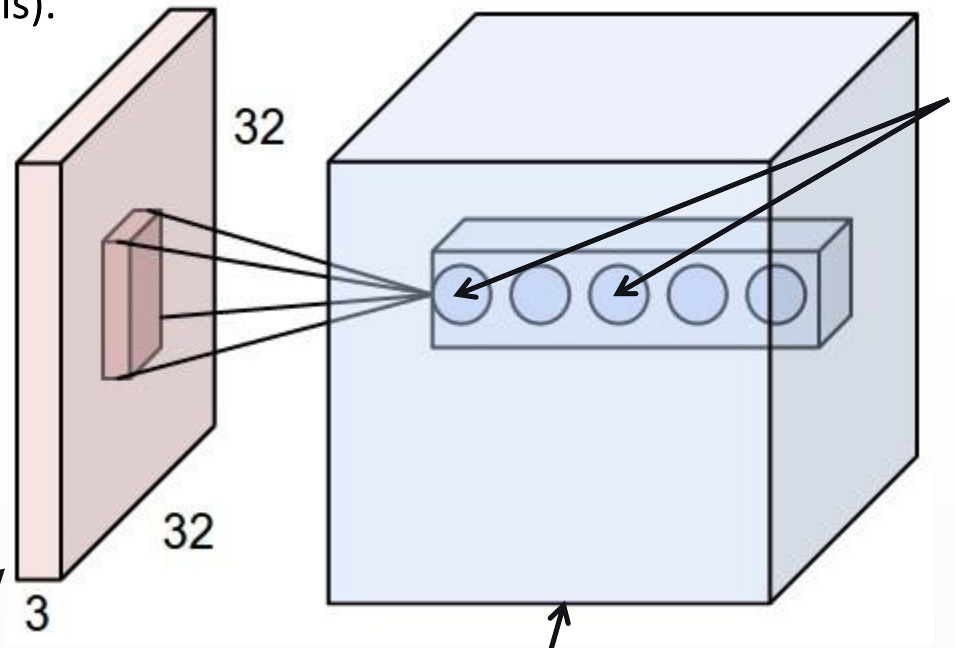
Input: n-channel array: $h \times w \times n$

f filter: each one is an n-channel kernel of size $k_h \times k_w \times n$

Output: f-channel array of size $(h - k_h + 1) \times (w - k_w + 1) \times f$

Convolutional Neural Network

Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels).



The neurons along the depth are all looking at the same region of the image. But they learn a different set of weights.

3-channel (R,G,B) input image with 32x32 pixels

Volume of neurons calculating 5 different convolutions

http://cs231n.github.io/convolutional-networks/

Convolutional Neural Network

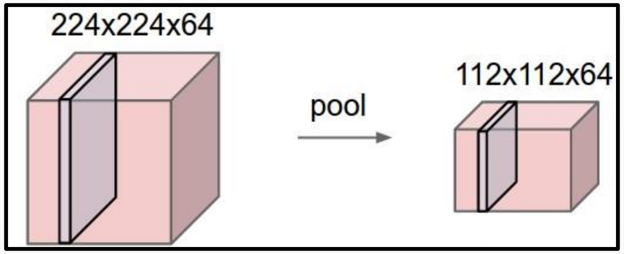
- Convolutional layers:
  - Has learnable weights and biases.
  - Has an activation function in the neurons.
  - The performed computation is a differentiable function.
  - It assumes that the input is an *image*:
    - Using this assumption it can be more efficient with less parameters

Pooling Layer

- It is common practice to insert a Pooling layer in-between successive convolutional layers.
- The goal is
  - to gain robustness against small changes in the location of a feature

The algorithm should recognize the wolf on both images, regardless of its location on the image.

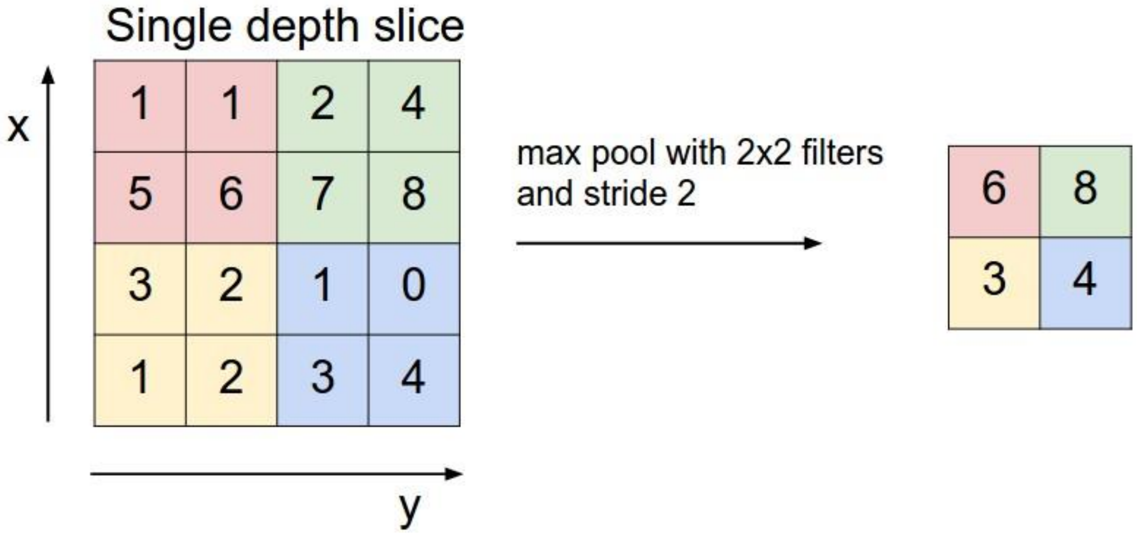
  - to reduce dimensionality (downsample along the spatial dimensions)



http://cs231n.github.io/convolutional-networks/

How mutch the feature maps are downsampled is defined by the parameters of the pooling.

Pooling Layer



Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

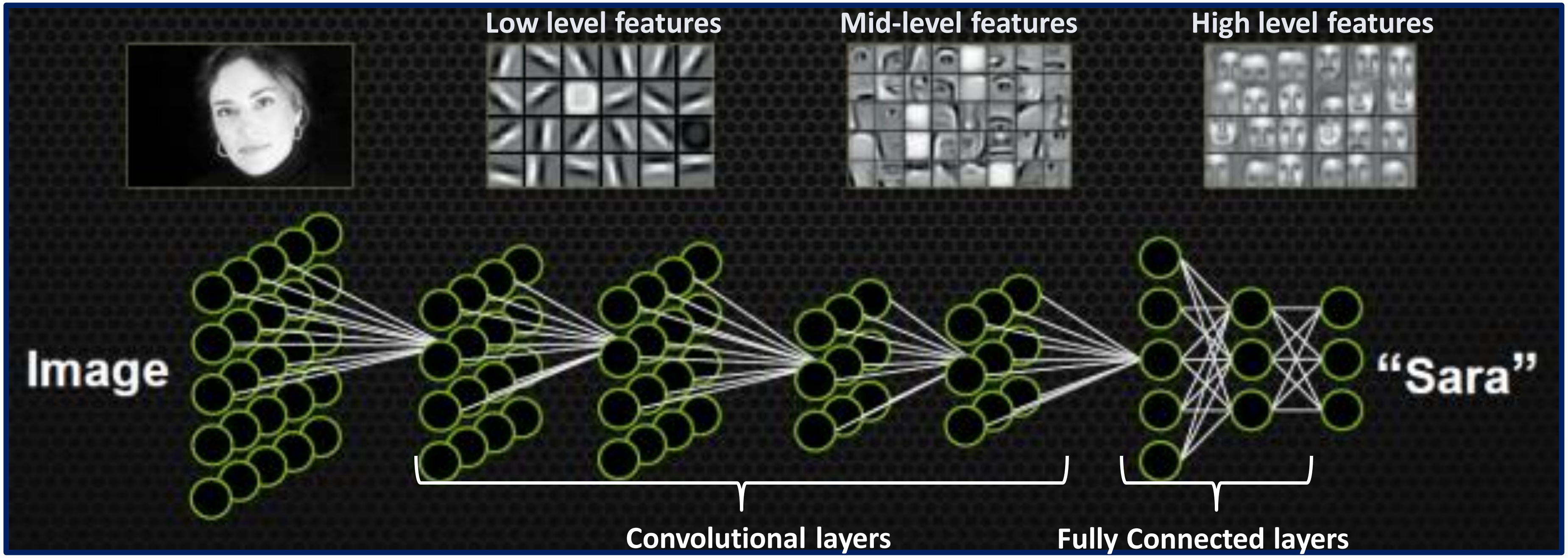
max pool with 2x2 filters and stride 2

6	8
3	4

- Parameters of the pooling:
  - Filter size: usual sizes are 2x2 or 3x3
  - Stride: defines the spatial shift of the filter
- Introduces zero parameters since it computes a fixed function of the input.
- Types of pooling:
  - Max pooling
  - Average pooling

http://cs231n.github.io/convolutional-networks/

What does the CNN Learn?



Source of the image:
https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/
H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." In ICML 2009.

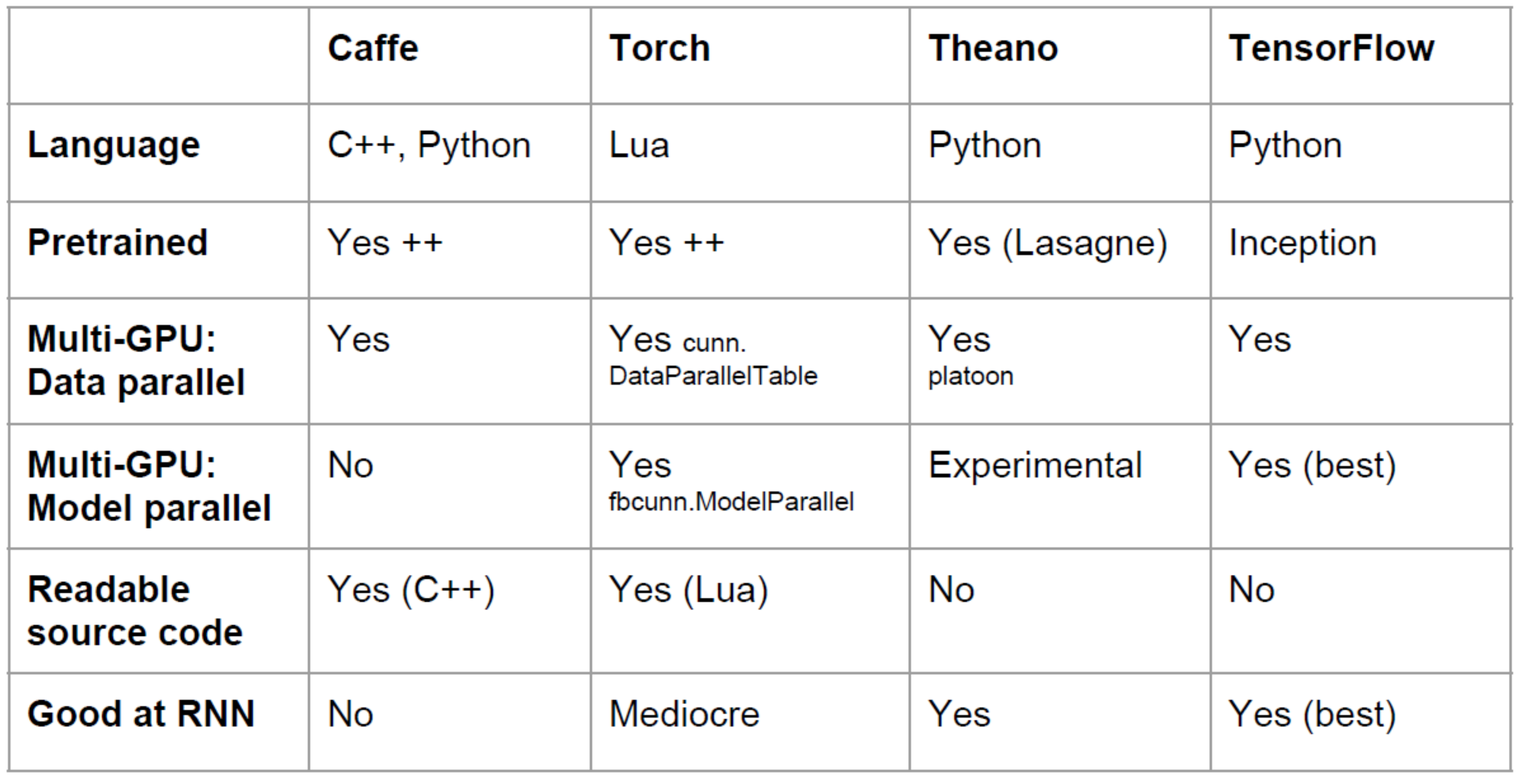
Transfer Learning

- For natural images the low level features are similar, hence the learned features will be very similar for different tasks.
- Big databases are not so easy to come by.
- The models that were trained on a big dataset could be partly reused on other tasks:
  - Reuse a convnet as *fixed feature extractor*: Take a trained model, remove its final fully connected layer and use the rest as a fixed feature extractor for a classifier (like a linear SVM or a softmax).
  - *Fine tuning*: use the network pre-trained on an other data and use it as initialization for the training on the data of interest. Usually this fine tuning is done with low learning rate, or even with the first few layers kept fixed.
- Using pre-trained models also helps against over fitting!

More info: http://cs231n.github.io/transfer-learning/

Deep Learning Frameworks

Frameworks



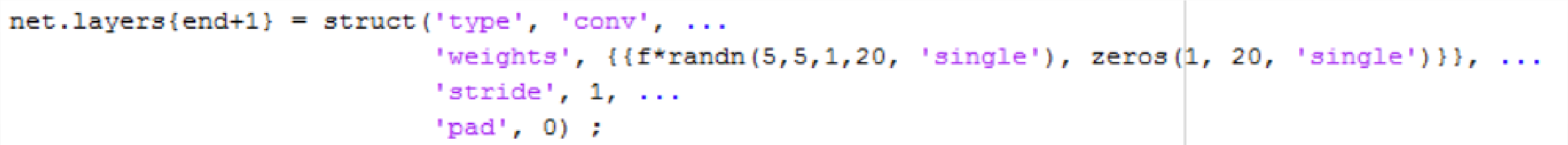
	Caffe	Torch	Theano	TensorFlow
Language	C++, Python	Lua	Python	Python
Pretrained	Yes ++	Yes ++	Yes (Lasagne)	Inception
Multi-GPU: Data parallel	Yes	Yes cunn. DataParallelTable	Yes platoon	Yes
Multi-GPU: Model parallel	No	Yes fbcunn.ModelParallel	Experimental	Yes (best)
Readable source code	Yes (C++)	Yes (Lua)	No	No
Good at RNN	No	Mediocre	Yes	Yes (best)

Source: http://cs231n.stanford.edu/slides/winter1516_lecture12.pdf

For more info: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

Frameworks

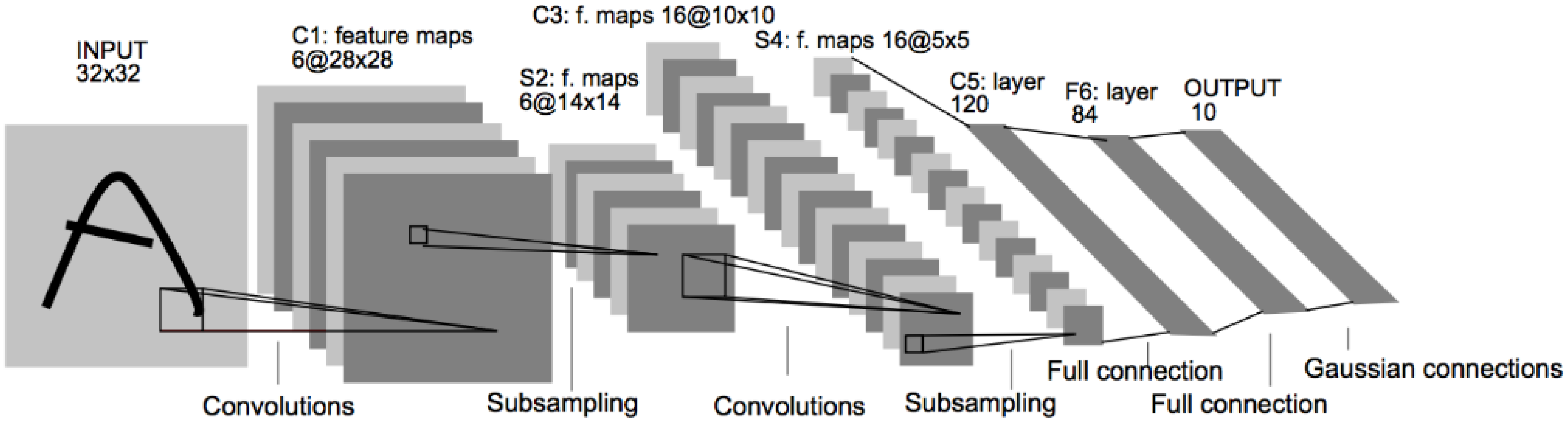
- MatConvNet:
  - Developed in Oxford for CNN training.
  - The network architecture is defined in a cell array of structs:
    - Each element in the array is one layer of the network:
      - Convolutional layer
      - Pooling layer
      - Activation layer
      - Dropout layer
      - Normalization layer
      - …
    - Each layer is defined in one struct:



```
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,1,20, 'single'), zeros(1, 20, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
```

CNN Architectures

LeNet-5 Architecture



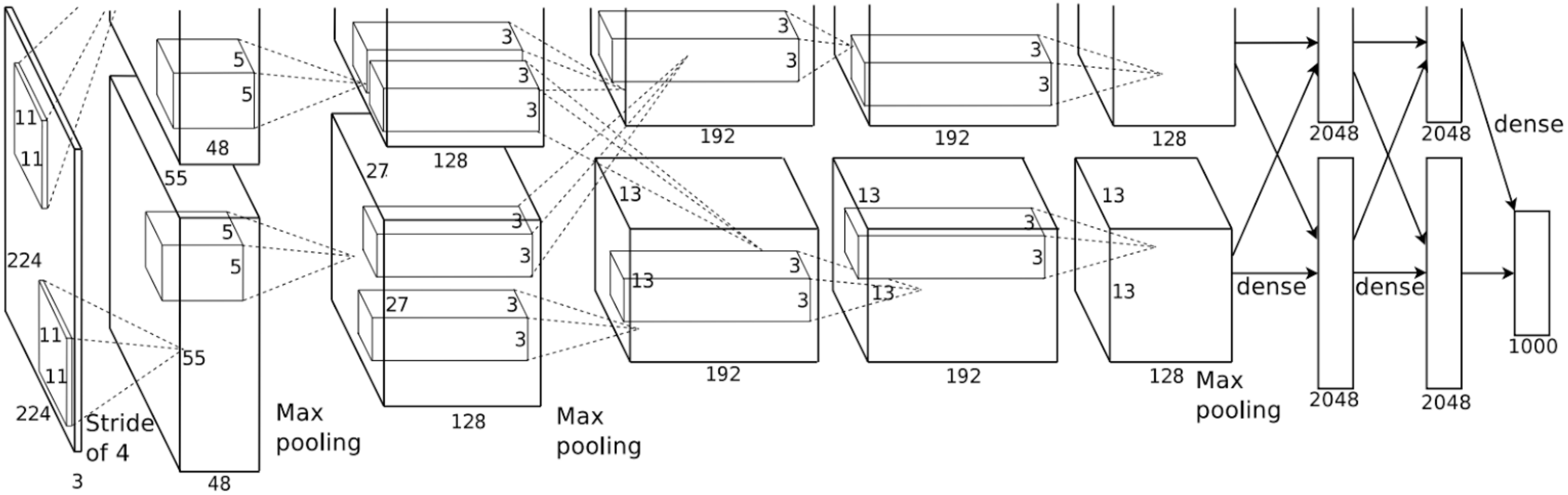
CNN called LeNet by Yann LeCun (1998)

- The first successfull application of convolutional networks.
- It was used for handwritten digit/zip code recognition.

http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Alexnet

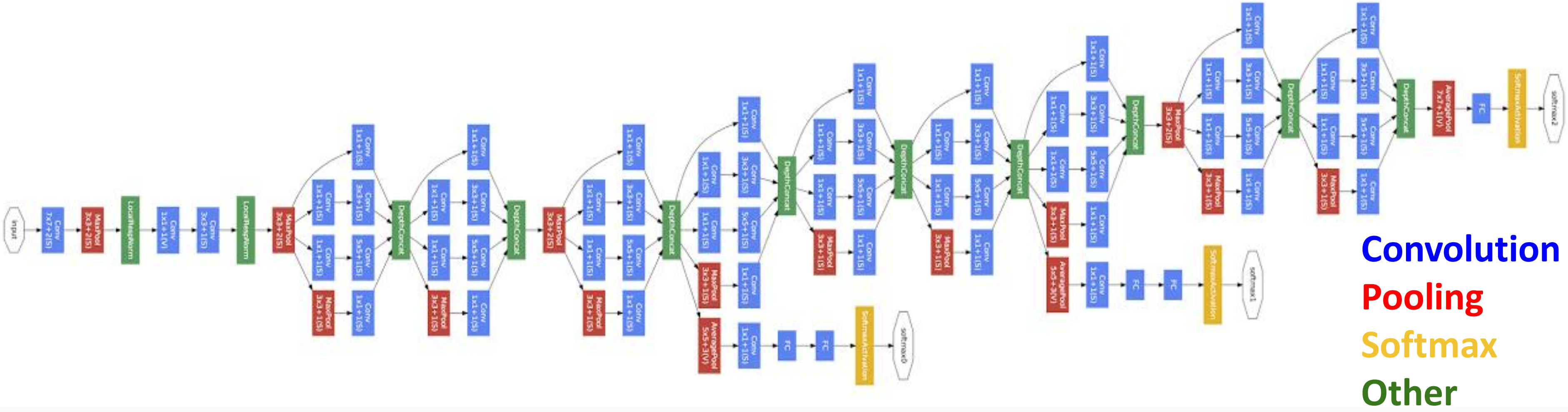
- Alex Krizhevsky's architecture that won the Imagenet in 2012.
- Similar to the LeNet architecture, but deeper.
- #parameters: 60 million



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012

GoogLeNet

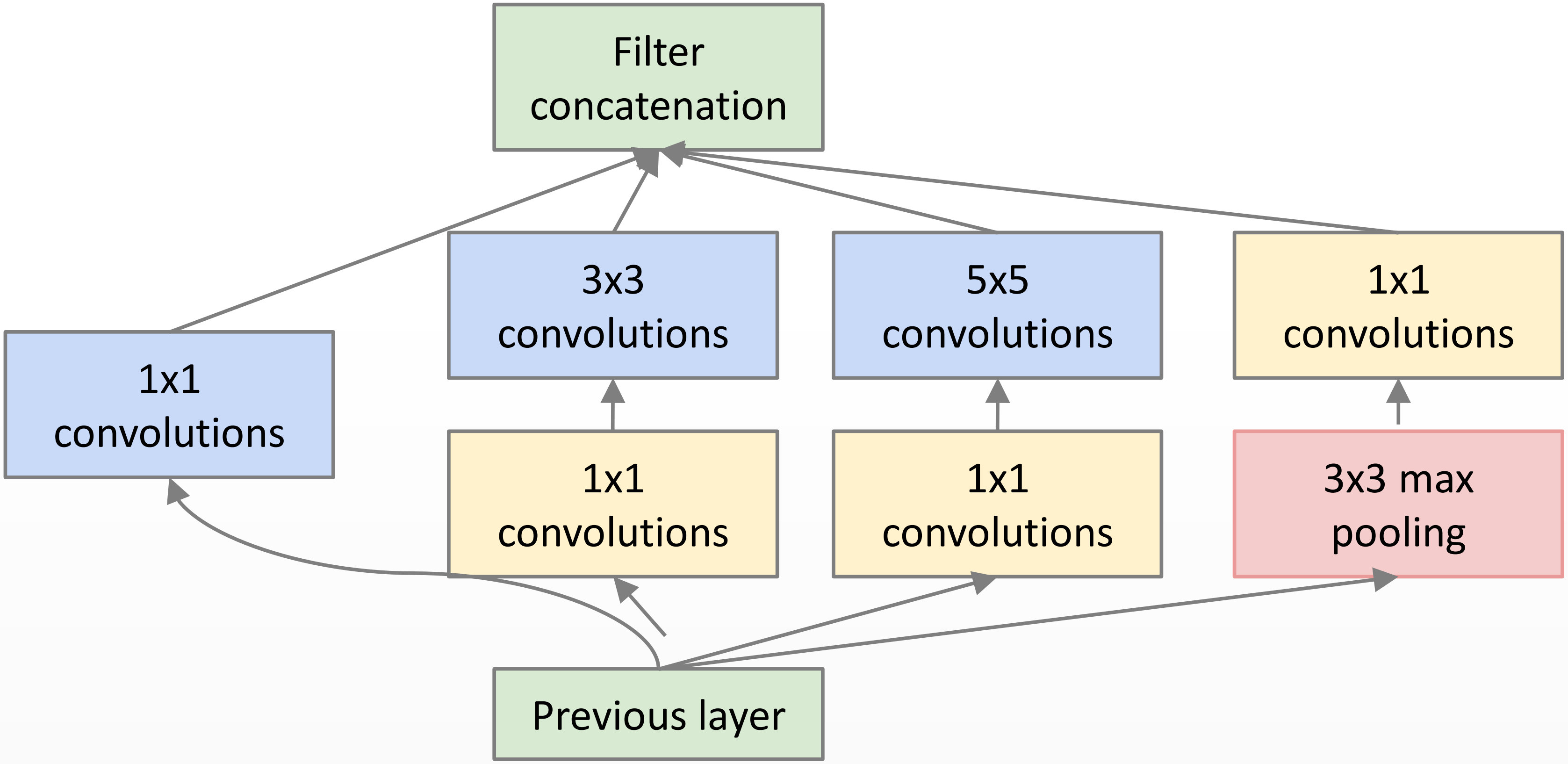
- Winner of the 2014 Imagenet challenge.
- Introduced the Inception module
- #parameters: 5 million



Convolution
Pooling
Softmax
Other

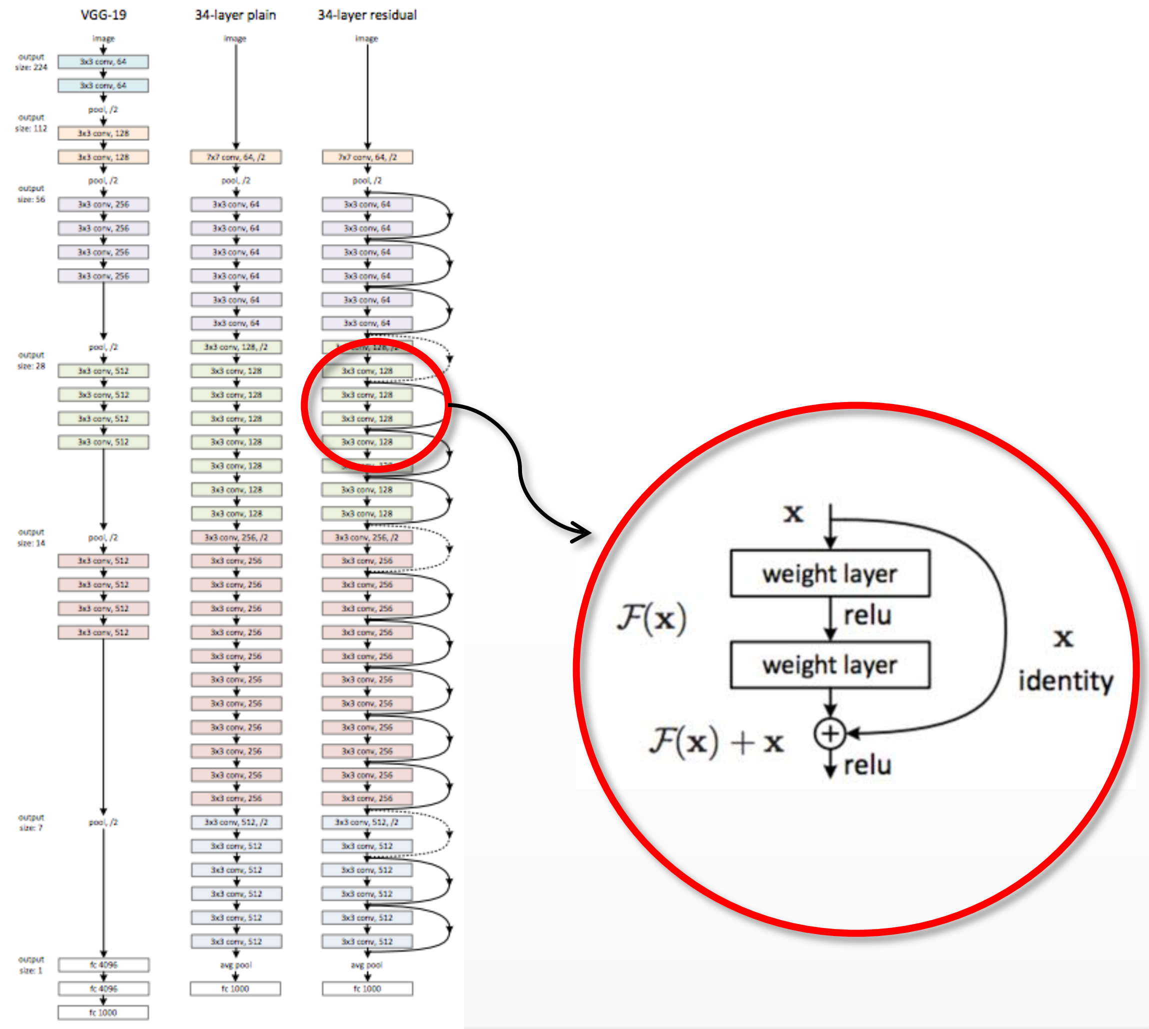
Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, „Going Deeper withConvolution," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

GoogLeNet – Inception Module



Deep Residual Networks

- Winner in 2015.
- 152 layers
- Introduction of the Residual blocks
- It can be regarded as a special case of Highway networks.



Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015.

Further Reading

- http://cs231n.stanford.edu/
- http://deeplearning.stanford.edu/tutorial/
- http://neuralnetworksanddeeplearning.com/
- http://deeplearning.net/
- http://www.deeplearningbook.org/
- https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf
- http://www.computervisionblog.com/2015/01/from-feature-descriptors-to-deep.html
- http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html