

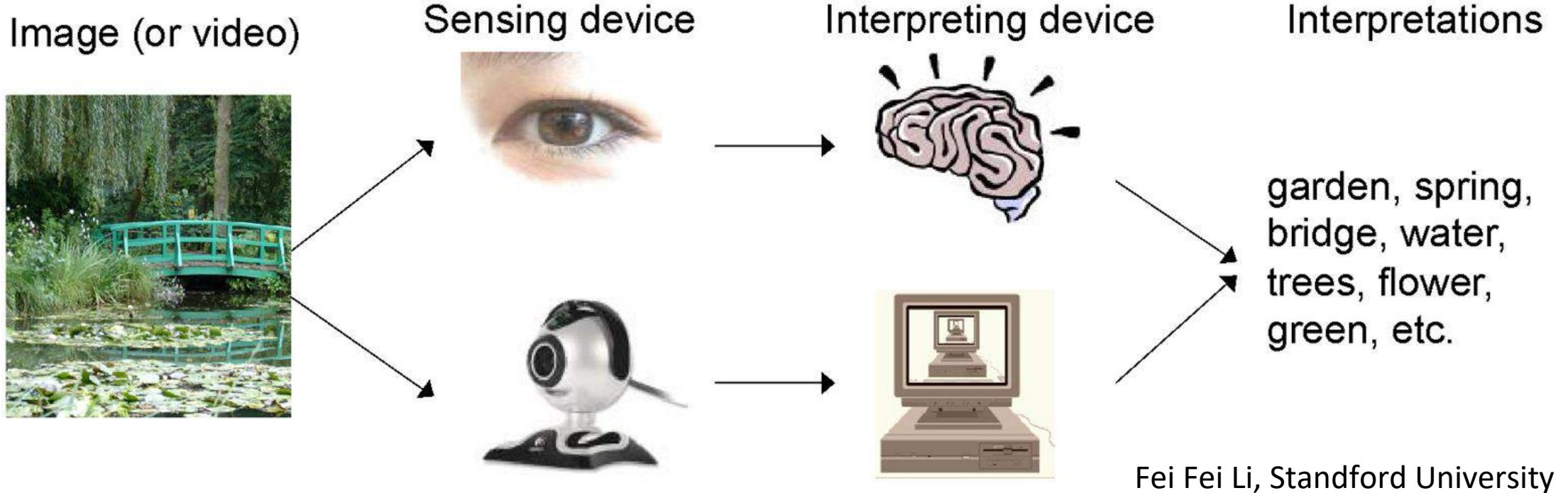
lecture1	2
lecture2	42
lecture3	77
lecture4	103
lecture5	131
lecture6	149
lecture7	170
lecture8	206
Lecture9	212
Lecture10	245
Lecture12	260
Lecture13	272
Lecture14	295
lecture16	310
Lecture20	335
Lecture21	338
Lecture23	349
Lecture24	359
Lecture25	372
Lecture26	390
Lecture27	403
Lecture28	426
Lecture29	441
Lecture30	464
Lecture31	493
Lecture32	512
Lecture33	532
Lecture34	552

Introduction to Computer Vision

DSE312-LECTURE1

BHAVNA R

What is (computer) vision?



Computer vision: understanding object recognition, motion interpretation (tracking), autonomous navigation, robotic manipulation of objects in a scene.
Original AI methods developed to tackle problems in computer vision

Vision is multidisciplinary

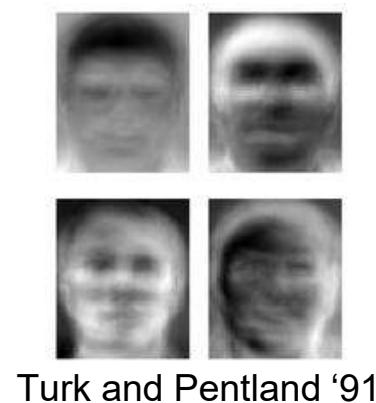
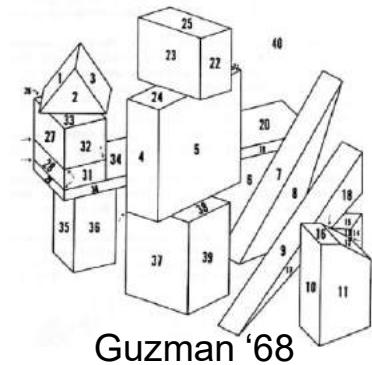
The idea of processing images by computer was conceived in the late 1950's, and over the decades to follow was further developed. The ideas have been applied to diverse fields :

- astronomy and space exploration
- remote sensing for earth resources research,
- diagnostic radiology,
- Biological image processing
- Biomedical fields
- surveillance,
- forensics,
- military defense,
- vehicle guidance,
- document processing,
- weather prediction,
- quality inspection in automated manufacturing processes.

Our knowledge of the human visual system and how to excel it is still very fragmentary and mostly confined to the early stages of computer vision.

A very very brief history of computer vision

- 1960's: interpretation of synthetic worlds
- 1966: Minsky assigns computer vision as an undergrad summer project
- 1970's: some progress on interpreting selected images
- 1980's: ANNs come and go; shift toward geometry and increased mathematical rigor
- 1990's: face recognition; statistical analysis in vogue
- 2000's: broader recognition; large annotated datasets available; video processing starts



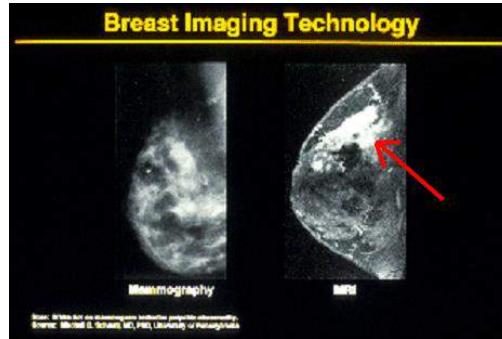
Vision is multidisciplinary



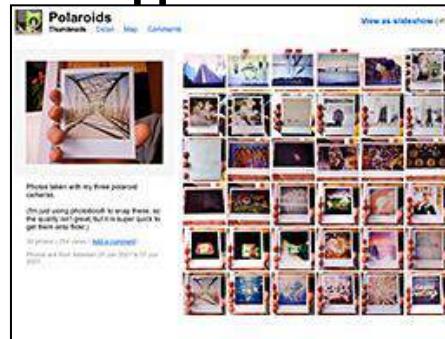
Safety



Security

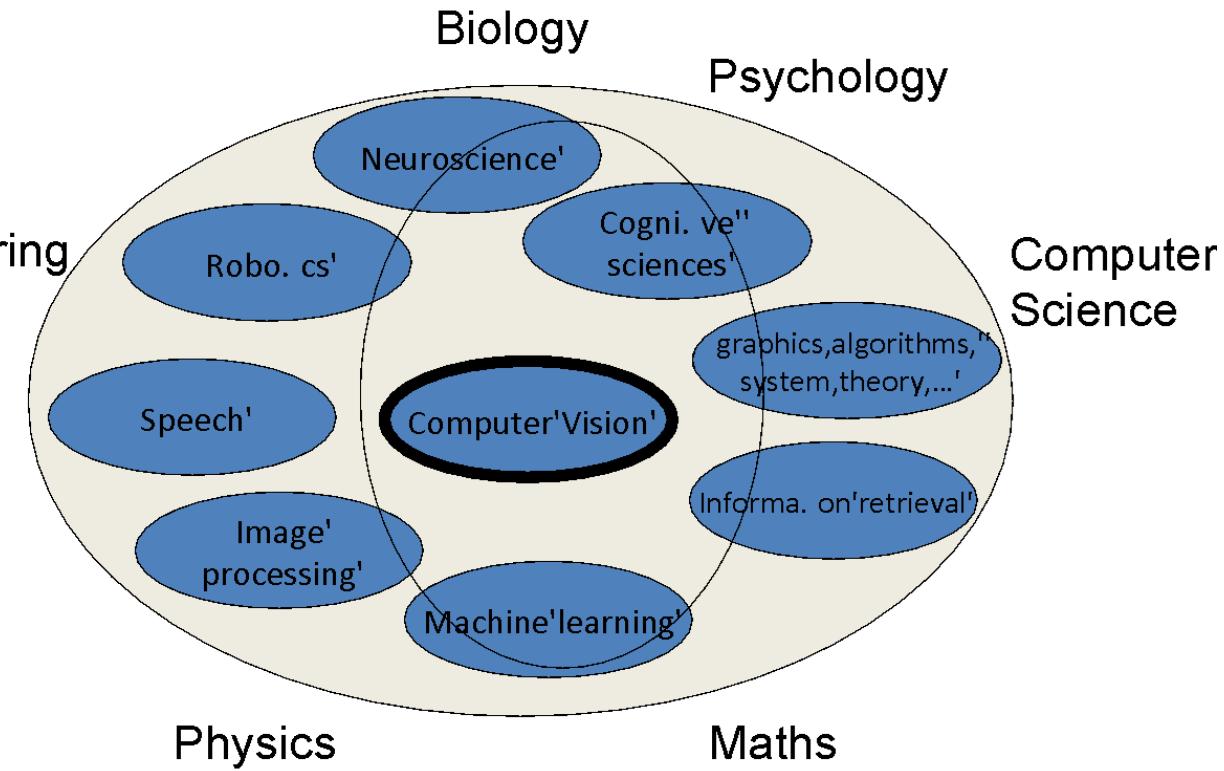


Health



Access

Engineering



Slides from James Hays, Brown University,
Fei Fei Li, Stanford University

The 'goal' of computer vision

- To bridge the gap between pixels and “meaning”



What we 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

What a computer sees

What does it mean, to see? “to know what is where by looking”.

How to discover from images what is present in the world, where things are, what actions are taking place

What do we want to achieve in this course?

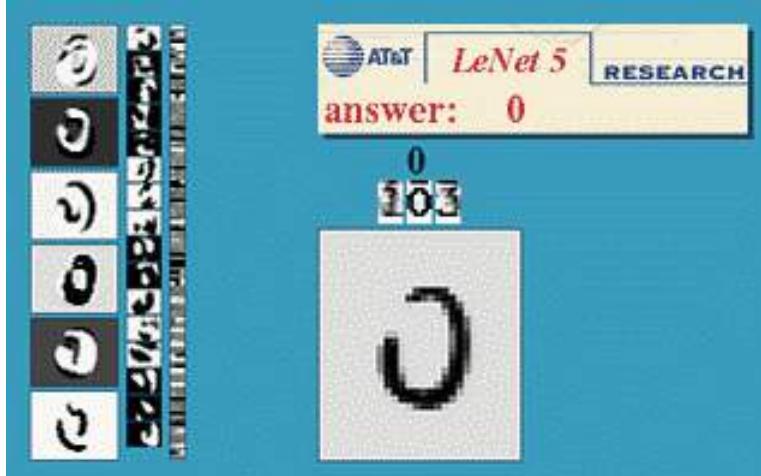
- The computer vision field is ripe with plenty of problems that are intellectually challenging, requiring new innovative methods across several disciplines.
- Here our objective is to understand the basic techniques to have prerequisite knowledge to tackle unsolved problems to self-develop solutions.

CV Examples from several disciplines

Optical character recognition (OCR)

Technology to convert scanned docs to text

- If you have a scanner, it probably came with OCR software



Digit recognition, AT&T labs
<http://www.research.att.com/~yann/>



License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

Face detection



- Many new digital cameras now detect faces
 - Canon, Sony, Fuji, ...

Face Detection vs. Face Recognition

Face Detection exploits the **similarities** between human faces.

- Using Probabilistic/Statistical Matching

Face Recognition exploits the **differences** between human faces.

- Using Principle Component Analysis

Smile detection

The Smile Shutter flow

Imagine a camera smart enough to catch every smile! In Smile Shutter Mode, your Cyber-shot® camera can automatically trip the shutter at just the right instant to catch the perfect expression.



[Sony Cyber-shot® T70 Digital Still Camera](#)

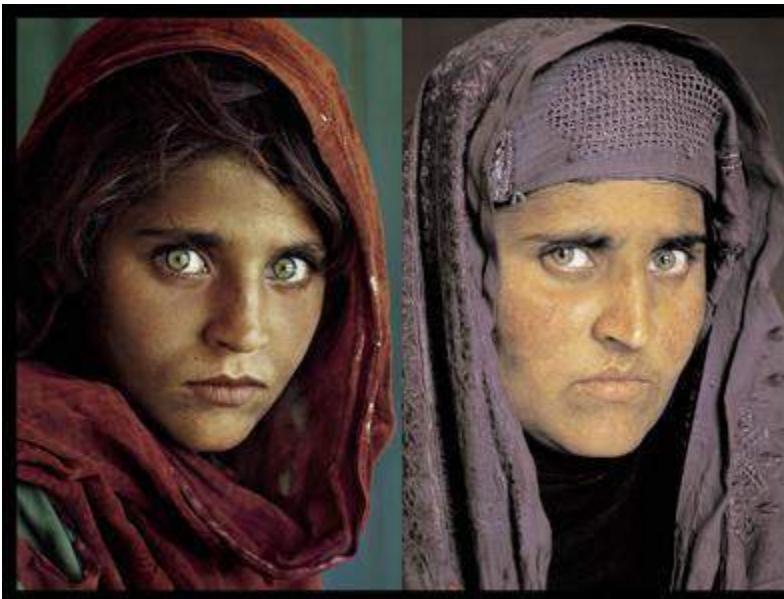
Object recognition (in supermarkets)



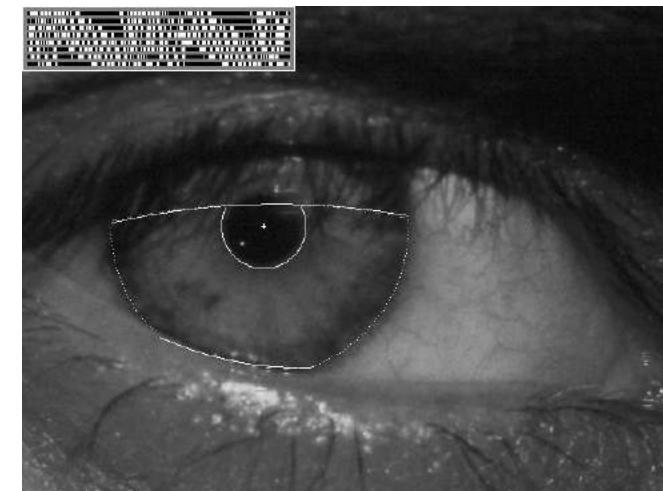
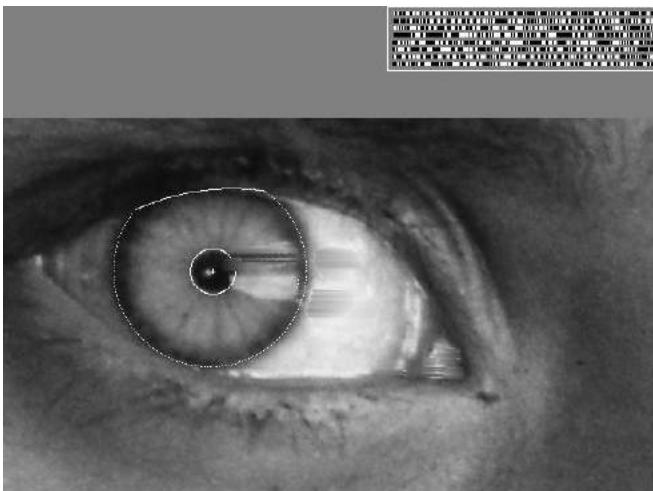
[LaneHawk by EvolutionRobotics](#)

“A smart camera is flush-mounted in the checkout lane, continuously watching for items. When an item is detected and recognized, the cashier verifies the quantity of items that were found under the basket, and continues to close the transaction. The item can remain under the basket, and with LaneHawk, you are assured to get paid for it... “

Vision-based biometrics



“How the Afghan Girl was Identified by Her Iris Patterns” Read the [story](#)
[wikipedia](#)



Login without a password...



Fingerprint scanners on
many new laptops,
other devices



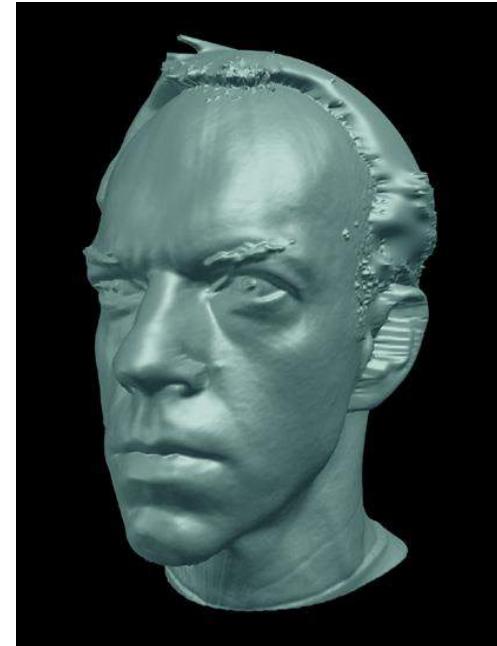
Face recognition systems now
beginning to appear more widely
<http://www.Sensiblevision.com/>

Object recognition (in mobile phones)



Point & Find, Nokia
Google Goggles

Special effects: shape capture



The Matrix movies, ESC Entertainment, XYZRGB, NRC

Special effects: motion capture



Pirates of the Caribbean, Industrial Light and Magic

Sports



Sportvision first down line

<http://www.sportvision.com>

Smart cars

Slide content courtesy of Amnon Shashua

The screenshot shows the Mobileye website homepage. At the top, there are two tabs: "manufacturer products" on the left and "consumer products" on the right. Below the tabs, the slogan "Our Vision. Your Safety." is displayed. A central image shows a car from above with three cameras highlighted: "rear looking camera" on the left, "forward looking camera" on the right, and "side looking camera" at the bottom. Below this, there are three main sections: "EyeQ Vision on a Chip" featuring an image of a chip, "Vision Applications" featuring an image of a person walking across a crosswalk, and "AWS Advance Warning System" featuring an image of a dashboard display. To the right, there are two columns: "News" with links to "Mobileye Advanced Technologies Power Volvo Cars World First Collision Warning With Auto Brake System" and "Volvo: New Collision Warning with Auto Brake Helps Prevent Rear-end" (with a "read more" link), and "Events" with links to "Mobileye at Equip Auto, Paris, France" and "Mobileye at SEMA, Las Vegas, NV" (also with a "read more" link).

- Mobileye
 - Vision systems currently in high-end BMW, GM, Volvo models
 - By 2010: 70% of car manufacturers.

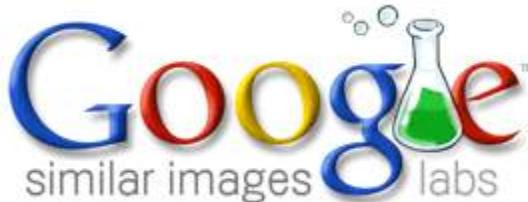
Google cars



<http://www.nytimes.com/2010/10/10/science/10google.html?ref=artificialintelligence>

Google Similar Images

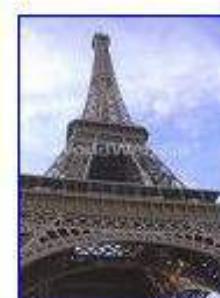
<http://www.youtube.com/watch?v=6fD2t4d2Ln4>



Showing only similar images - [Back to results for paris](#)



360 x 480 - 43k - jpg
www.johnnyjet.com
[Similar images](#)



270 x 360 - 40k - jpg
graphics.worldweb.com
[Similar images](#)



337 x 450 - 36k - jpg
tripadvisor.com
[Similar images](#)

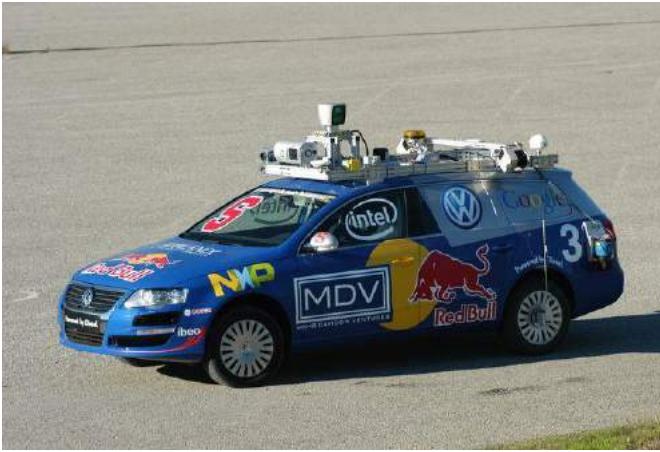
Systems that learn about the world.

<http://similar-images.googlelabs.com/>

Cell Tracking / Analysis



Vision Guided Robots



Autonomous
Vehicles



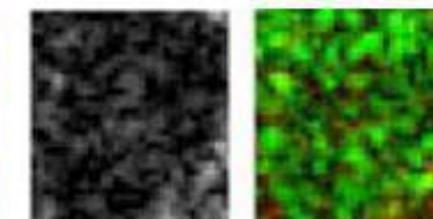
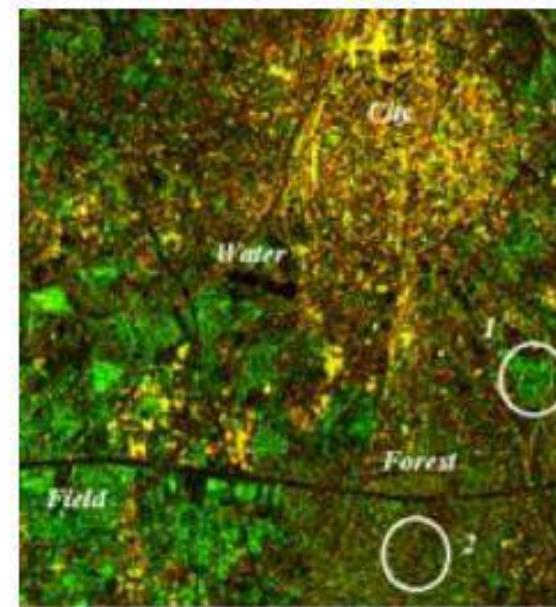
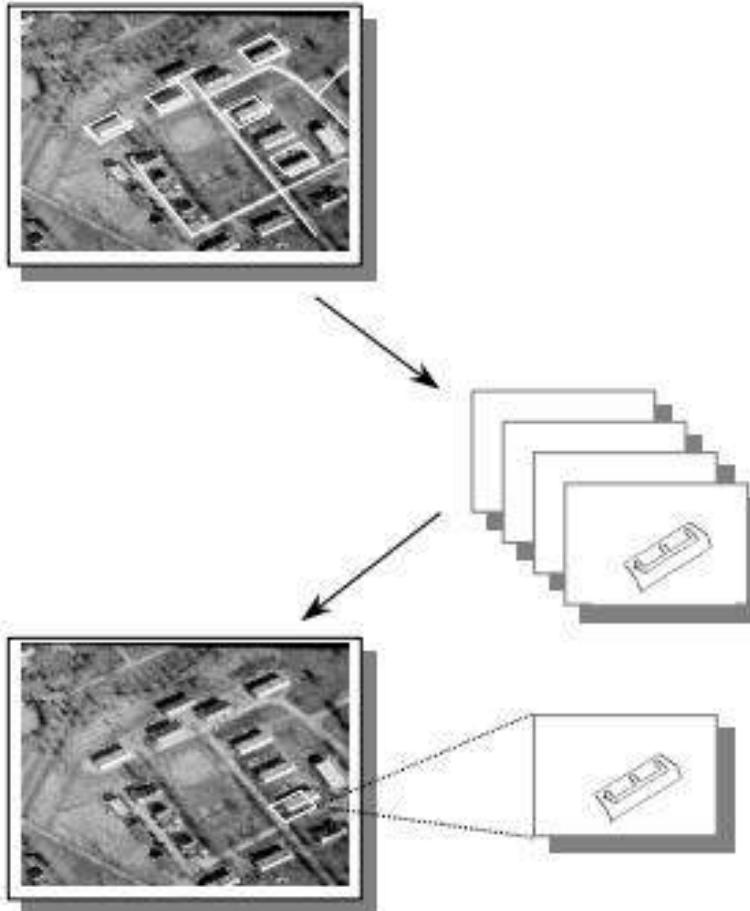
Assistive Robots



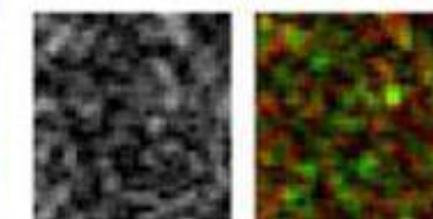
Tele-presence
Robots

Manufacturing

Remote Sensing (Geography)



"Field" texture
(extract 1)



"Forest" texture
(extract 2)

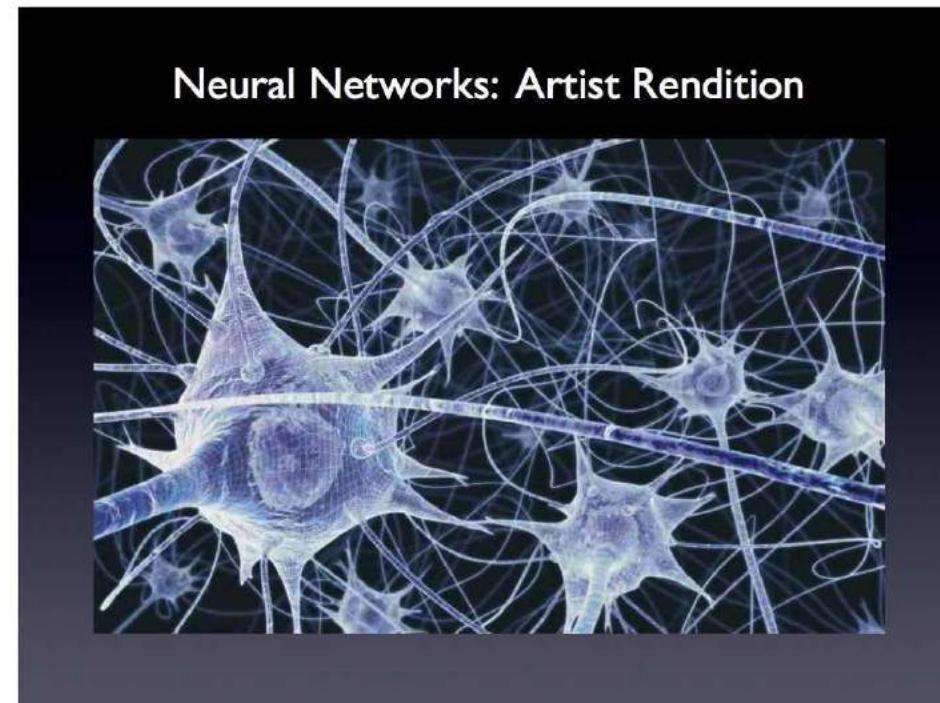
Computational Neuroscience

Biologically Inspired Vision:

Machine Learning, Artificial Neural Networks

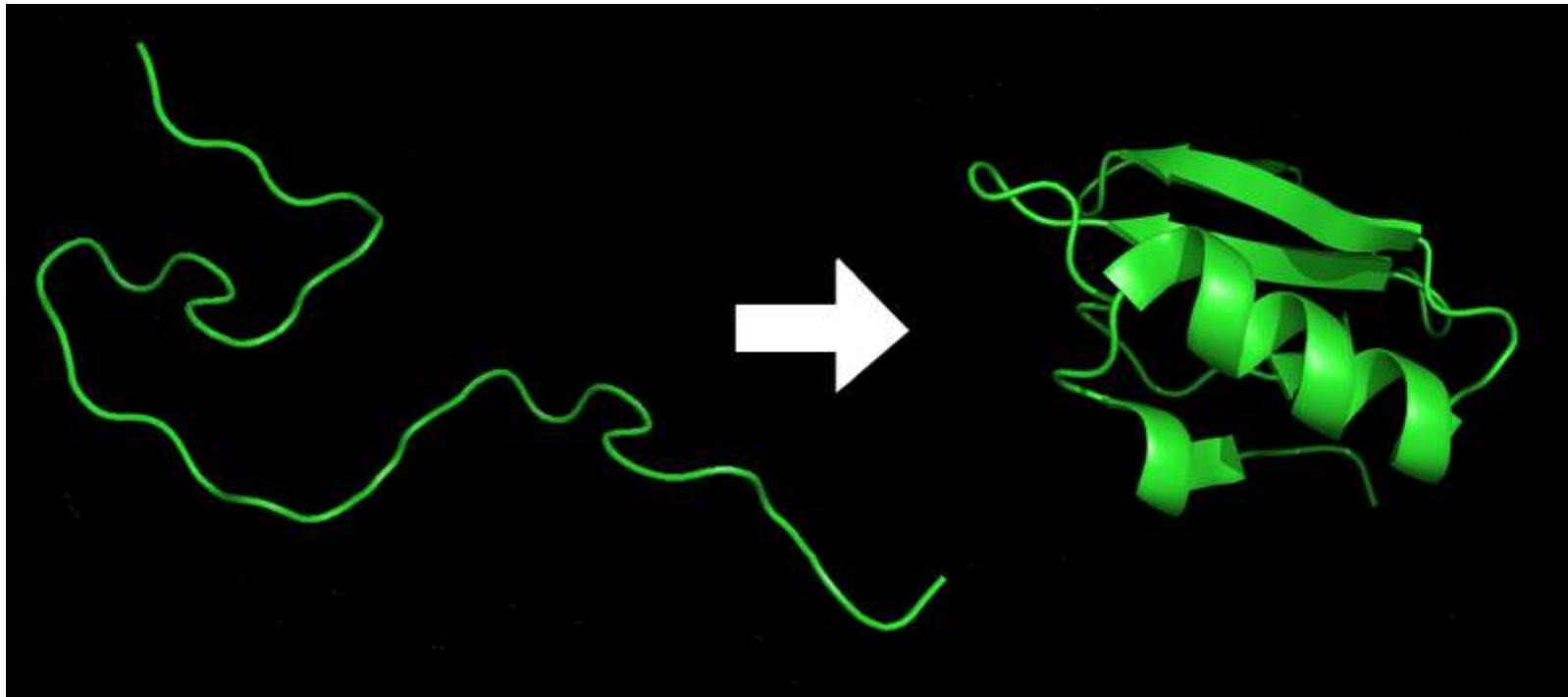
Brain Modelling

Brain-Computer Interfaces



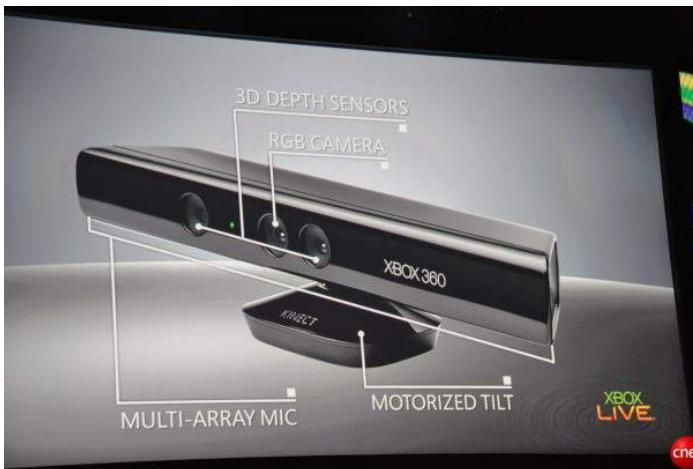
Protein Folding (Biochemistry)

Many Computer Vision techniques are used in computer simulations.



Interactive Games: Kinect

- Object Recognition: <http://www.youtube.com/watch?feature=iv&v=fQ59dXOo63o>
- Mario: <http://www.youtube.com/watch?v=8CTJL5IUjHg>
- 3D: <http://www.youtube.com/watch?v=7QrnwoO1-8A>
- Robot: <http://www.youtube.com/watch?v=w8BmgtMKFbY>
- 3D tracking, reconstruction, and interaction: <http://research.microsoft.com/en-us/projects/surfacerecon/default.aspx>



Vision in space



NASA'S Mars Exploration Rover Spirit captured this westward view from atop a low plateau where Spirit spent the closing months of 2007.

Vision systems (JPL) used for several tasks

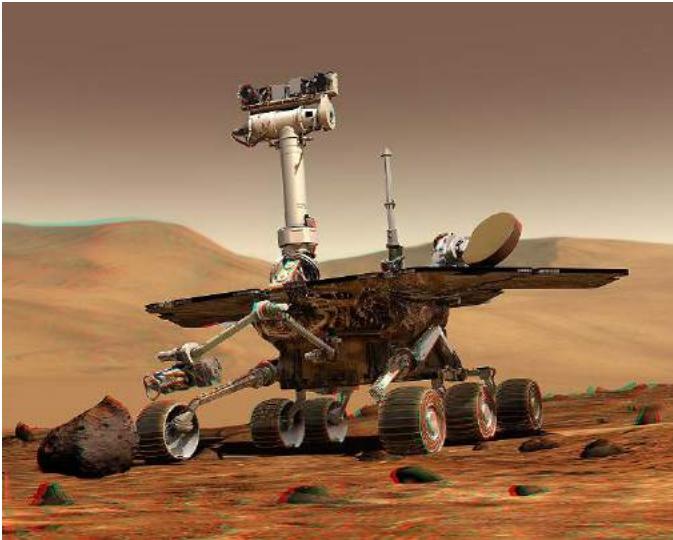
- Panorama stitching
- 3D terrain modeling
- Obstacle detection, position tracking
- For more, read “Computer Vision on Mars” by Matthies et al.

Industrial robots



Vision-guided robots position nut runners on wheels

Mobile robots



NASA's Mars Spirit Rover
http://en.wikipedia.org/wiki/Spirit_rover

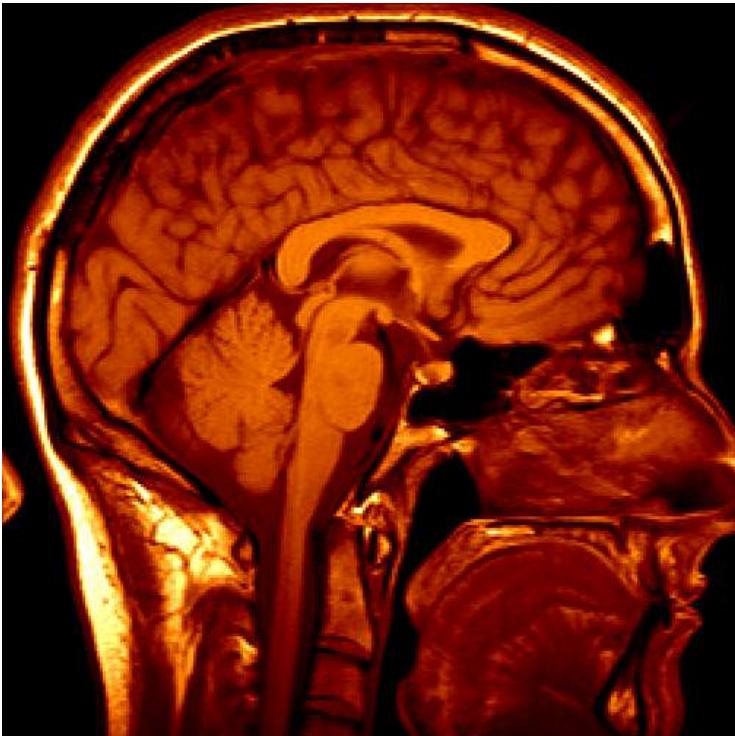


<http://www.robocup.org/>



Saxena et al. 2008
[STAIR at Stanford](#)

Medical imaging



3D imaging
MRI, CT



Image guided surgery
Grimson et al., MIT

Vision as a source of semantic information



slide credit: Fei-Fei, Fergus

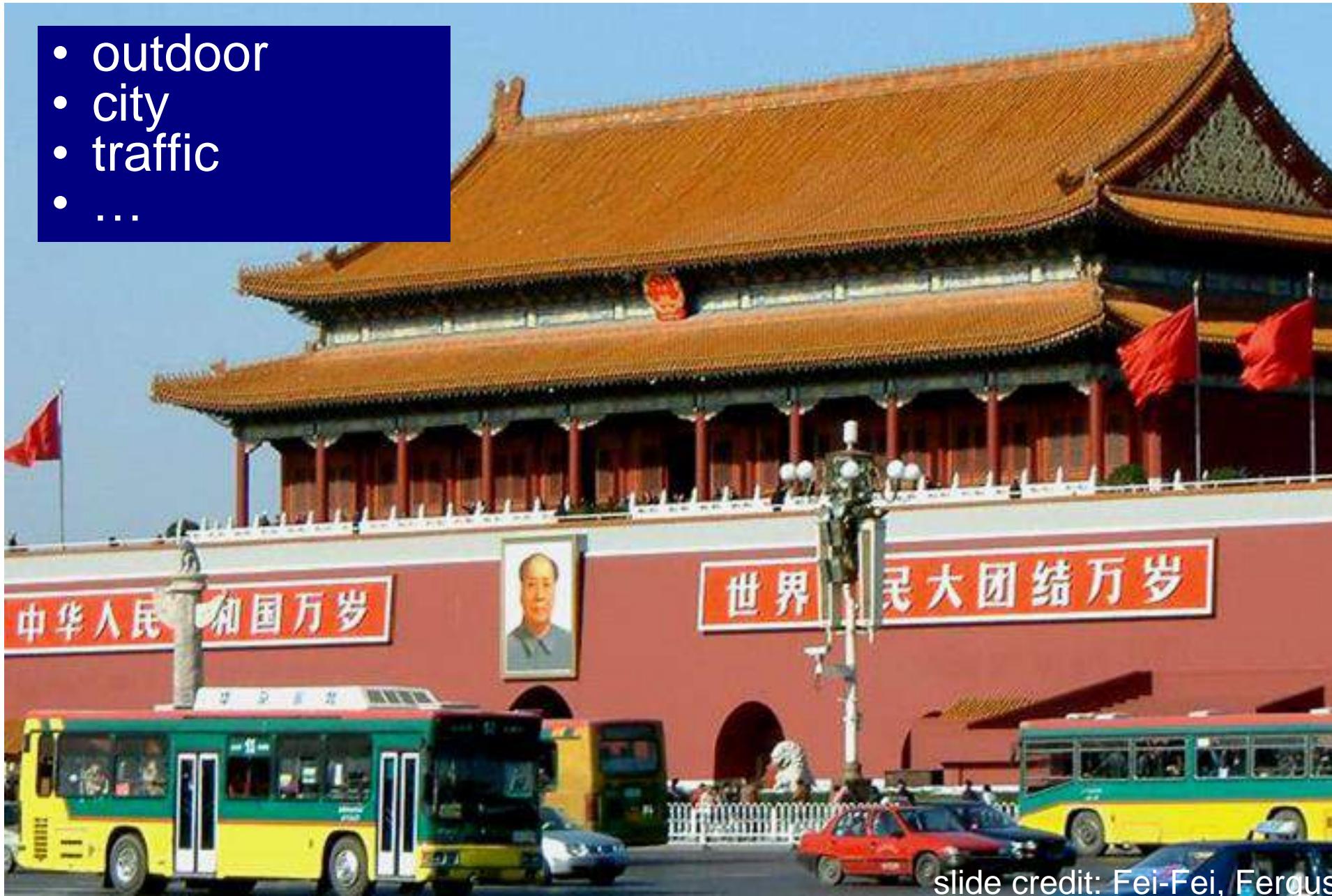
Object categorization



slide credit: Fei-Fei, Fergus

Scene and context categorization

- outdoor
- city
- traffic
- ...



slide credit: Fei-Fei, Fergus

Qualitative spatial information



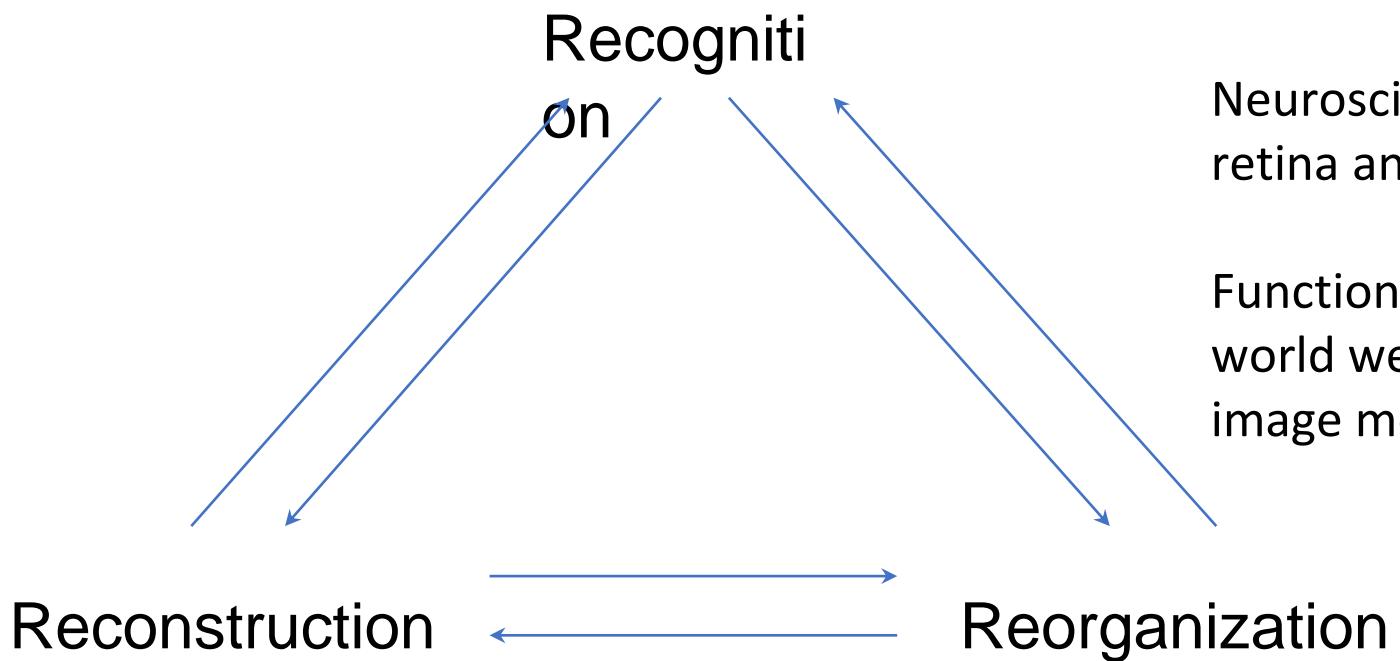
slide credit: Fei-Fei, Fergus

Challenges or opportunities?

Multidisciplinary efforts involving vision research, mathematics, physics, computer science, and artificial intelligence will be required before we can begin to build highly sophisticated computer vision systems that outperform human observers in all respects.

The Three R's of Vision

Different aspects of vision



Perception: study the “laws of seeing” -predict what a human would perceive in an image.

Neuroscience: understand the mechanisms in the retina and the brain

Function: how laws of optics, and the statistics of the world we live in, make certain interpretations of an image more likely to be valid

Each of the 6 directed arcs in this diagram is a useful direction of information flow

Jitendra Malik
UC Berkeley

Introduction to Computer Vision

Image formation model

EARLY VISION

Linear filters, Shift invariant linear systems

Convolution, correlations

Edge detection

Local image features

Image smoothing/sharpening

Frequency domain filtering

MID-LEVEL VISION

Segmentation, tracking

processes that extract attributes from images.

HIGH-LEVEL VISION

Registration, Reconstruction

Classification

Using elements of differential geometry for shapes

References (books)

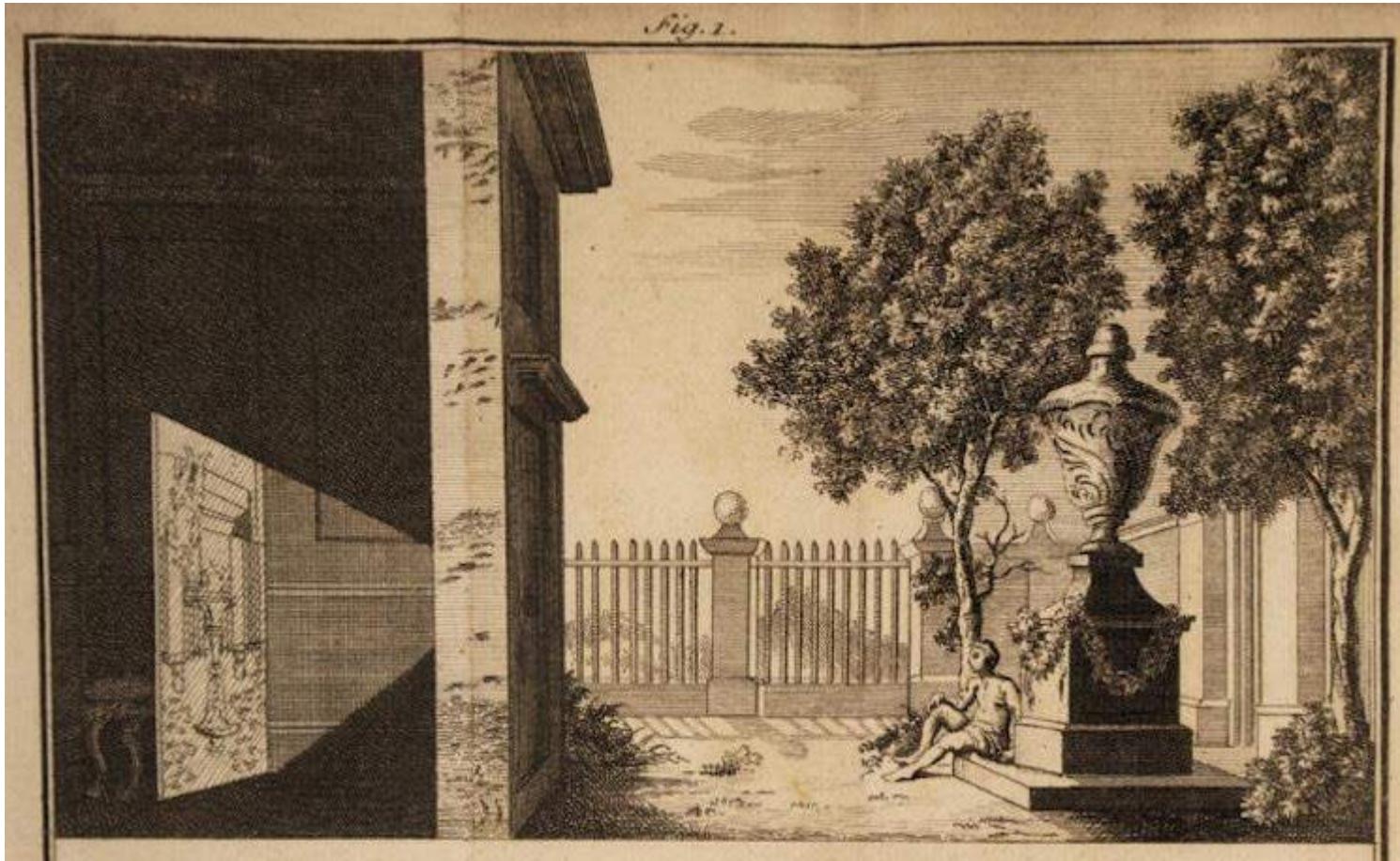
- Computer Vision: Algorithms and applications by Richard Szelski
- Computer Vision: A Modern Approach by David Forsyth and Jean Ponce
- Digital Image Processing Rafael C. Gonzalez and Richard E.Woods
- Multiple View Geometry in Computer Vision by Richard Hartley and Andrew Zisserman

Image Formation

DSE312 - LECTURE 2

BHAVNA R

The camera obscura: “A short account of the eye and nature of vision”

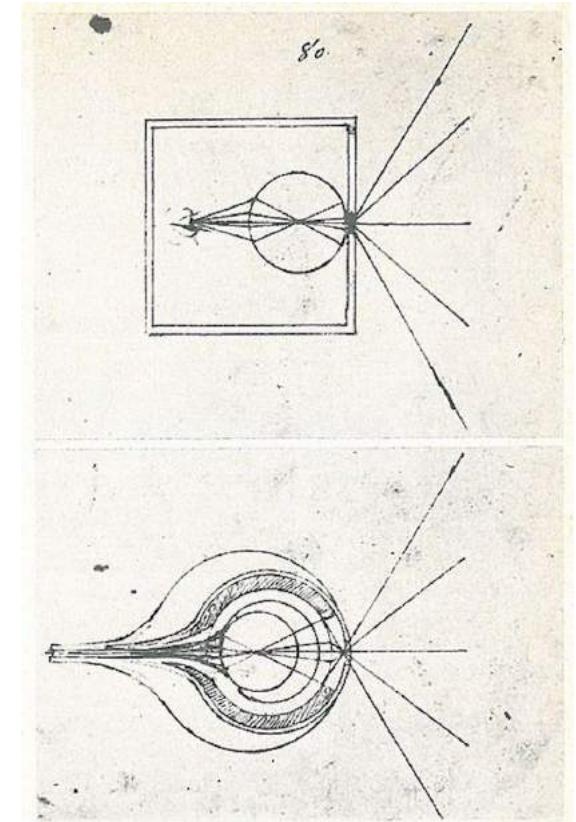


Camera obscura (meaning “dark room” in Latin) is a box-shaped device used as an aid for drawing or entertainment.

Also referred to as a **pinhole image**, it lets light in through a small opening on one side and projects a reversed and inverted image on the other.

The History of Camera Obscura

- Earliest written record of the camera obscura theory and the idea that the image is flipped **upside down** because light travels in straight lines from its source in the studies of Chinese philosopher and the founder of Mohism, Mozi (470 to 390 BCE).
- 4th century, Greek philosopher Aristotle noticed that sunlight passing through gaps between leaves projects an image of an eclipsed sun on the ground.
- 9th century, Arab philosopher, mathematician, physician, and musician Al-Kindi also experimented with light and a pinhole.
- Leonardo da Vinci published the first clear description of the camera obscura in *Codex Atlanticus* (1502)



A drawing comparing the human eye to a camera obscura from Leonardo da Vinci's "Codex Atlanticus" (1490-1495).

Camera obscura: dark room

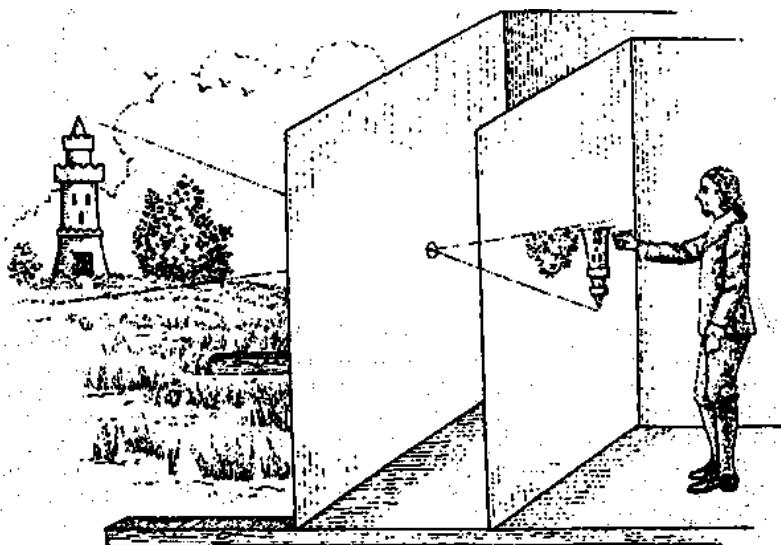
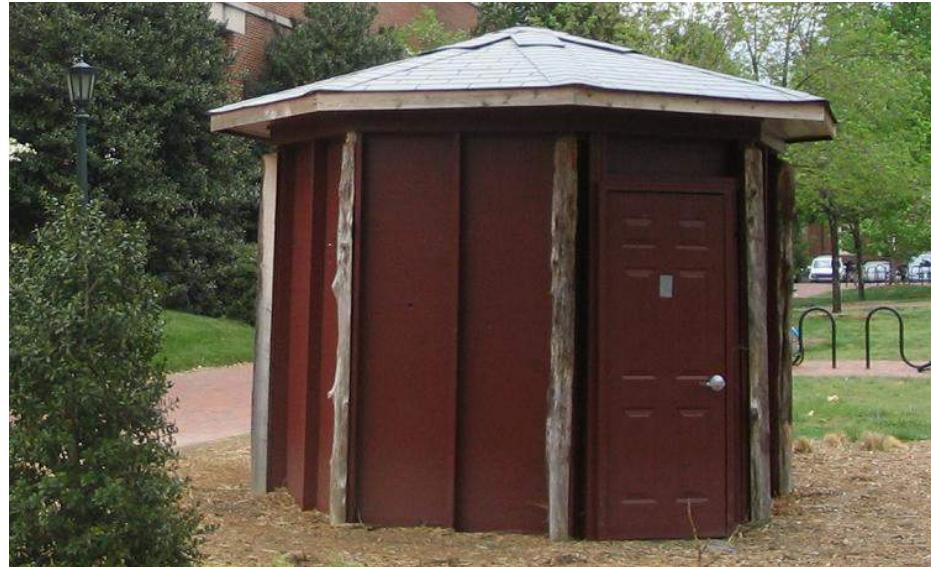


Illustration of Camera Obscura

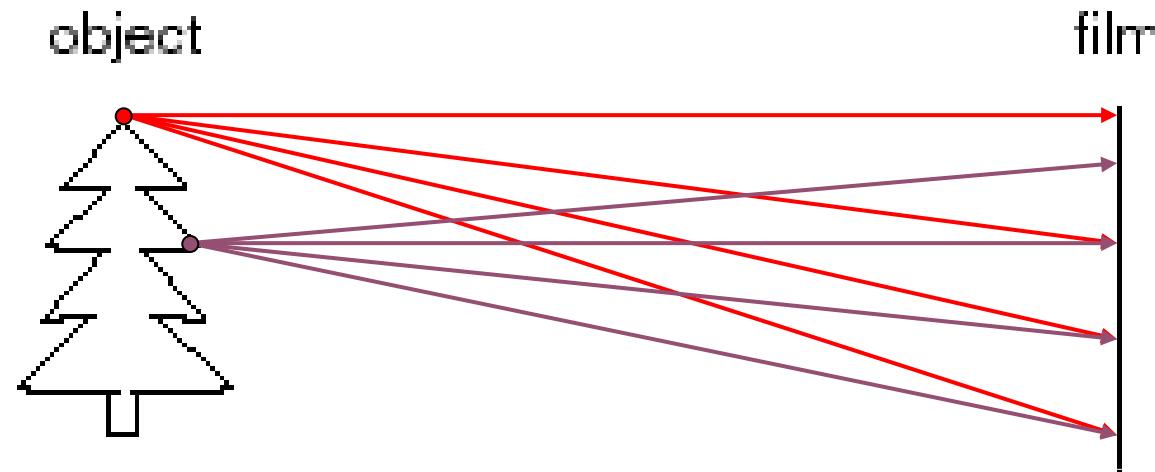


Freestanding camera obscura at UNC Chapel Hill

Photo by Seth Ilys

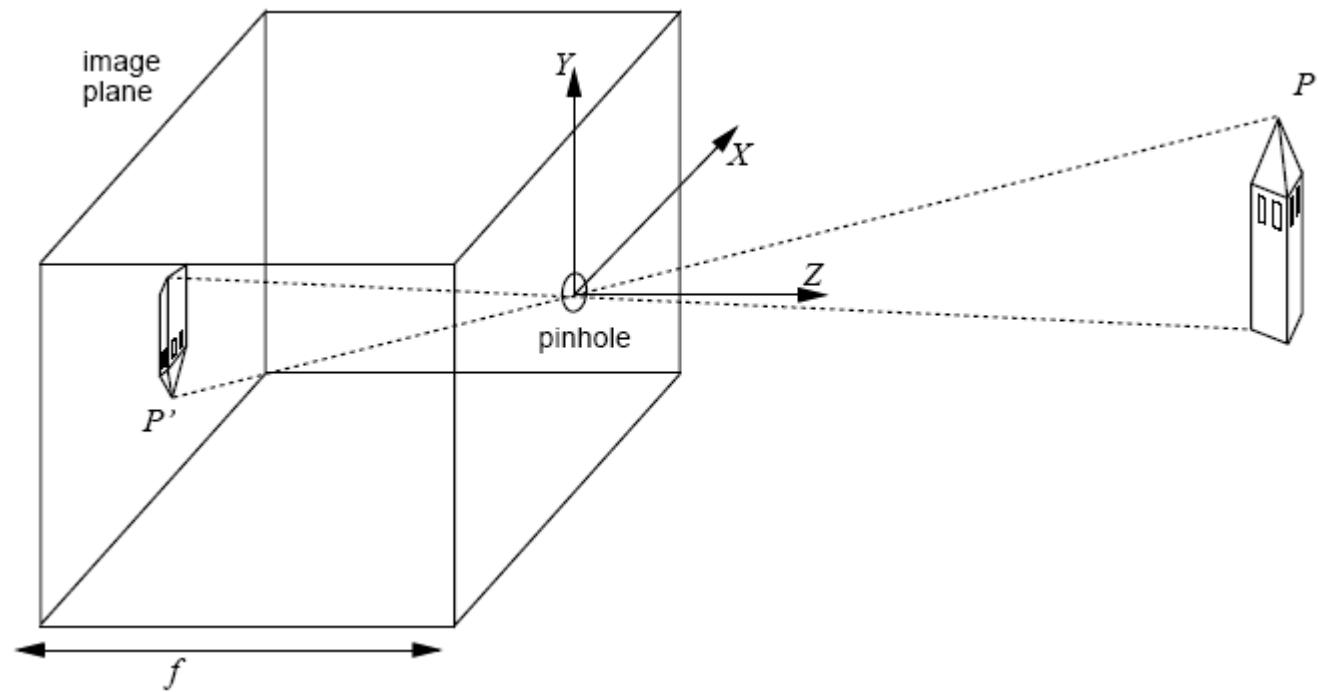
James Hays

Let's design a simple pinhole camera

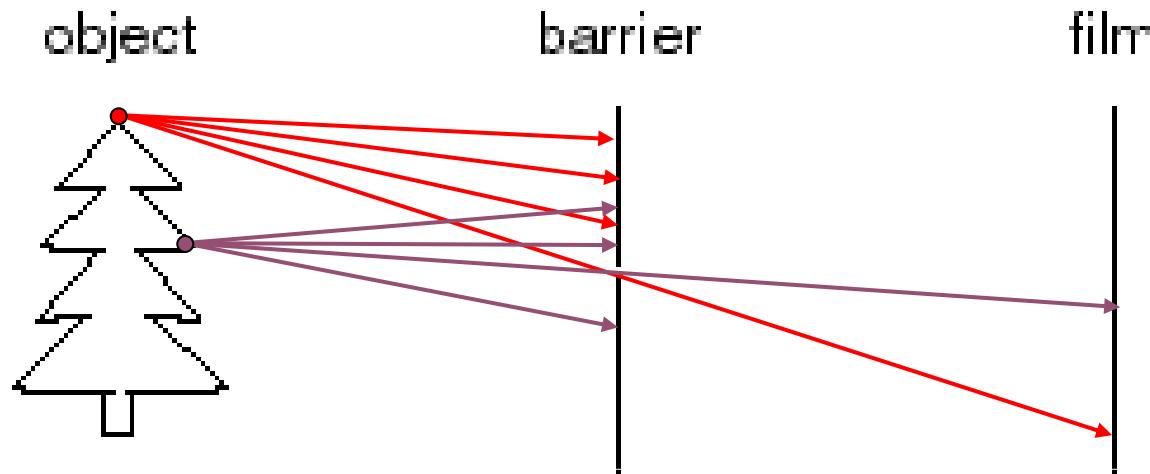


- Put a piece of film in front of an object - do we get a reasonable image?
 - Blurring - need to be more selective!

The pinhole camera



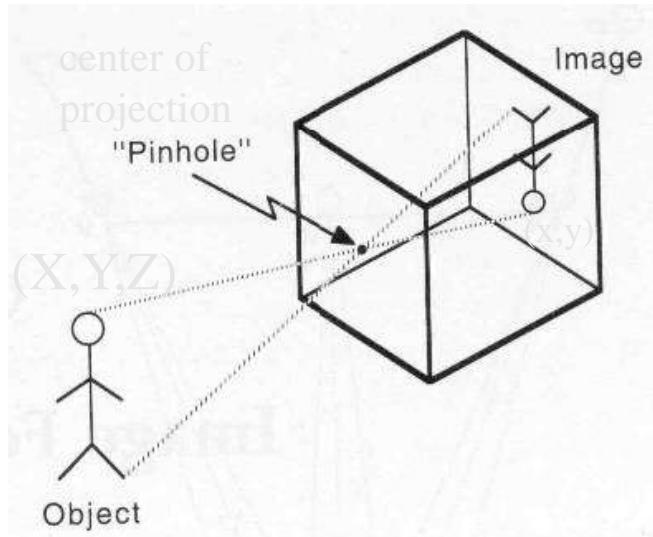
a simple pinhole camera(cont'd)



- Add a barrier with a small opening (i.e. **aperture**) to block off most of the rays
 - Reduces blurring

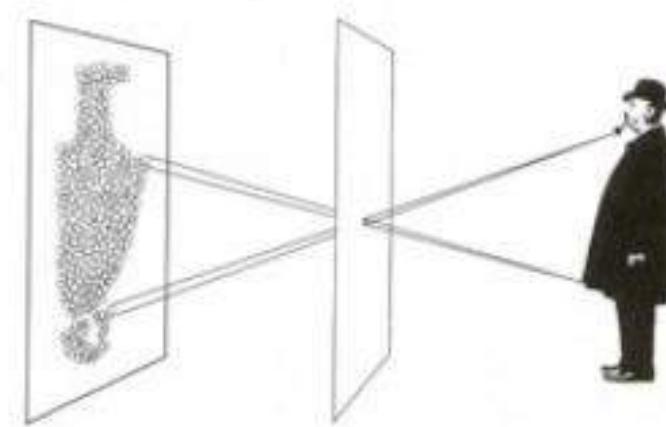
“Pinhole” camera model

- The simplest device to form an image of a 3D scene on a 2D surface.
- Rays of light pass through a "pinhole" and form an inverted image of the object on the image plane.

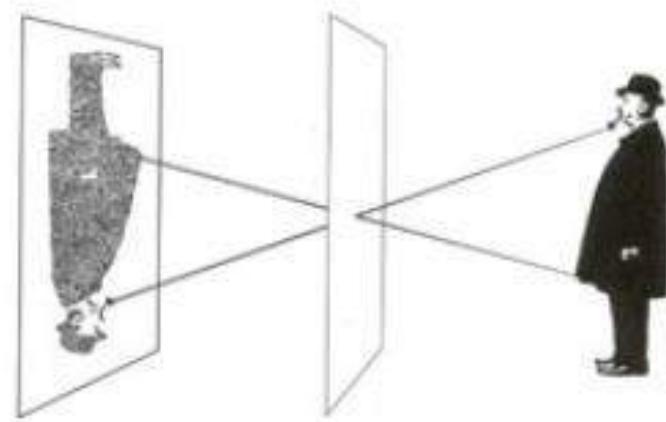


What is the effect of aperture size?

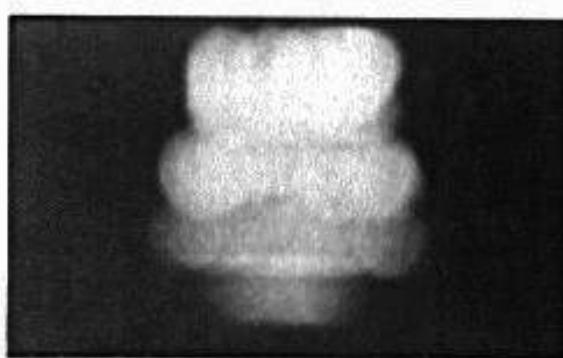
Large aperture: light from the source spreads across the image (i.e., not properly focused), making it **blurry**!



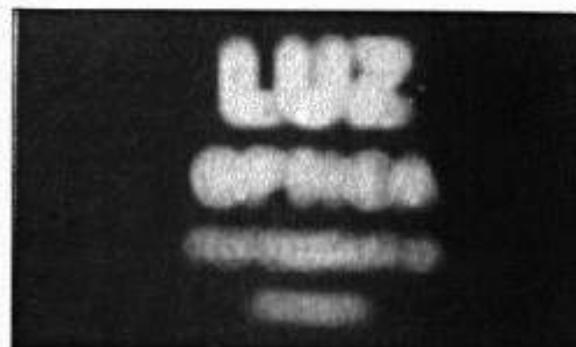
Small aperture: reduces **blurring** but (i) it limits the amount of light entering the camera and (ii) causes light **diffraction**.



Example: varying aperture size



2 mm



1 mm



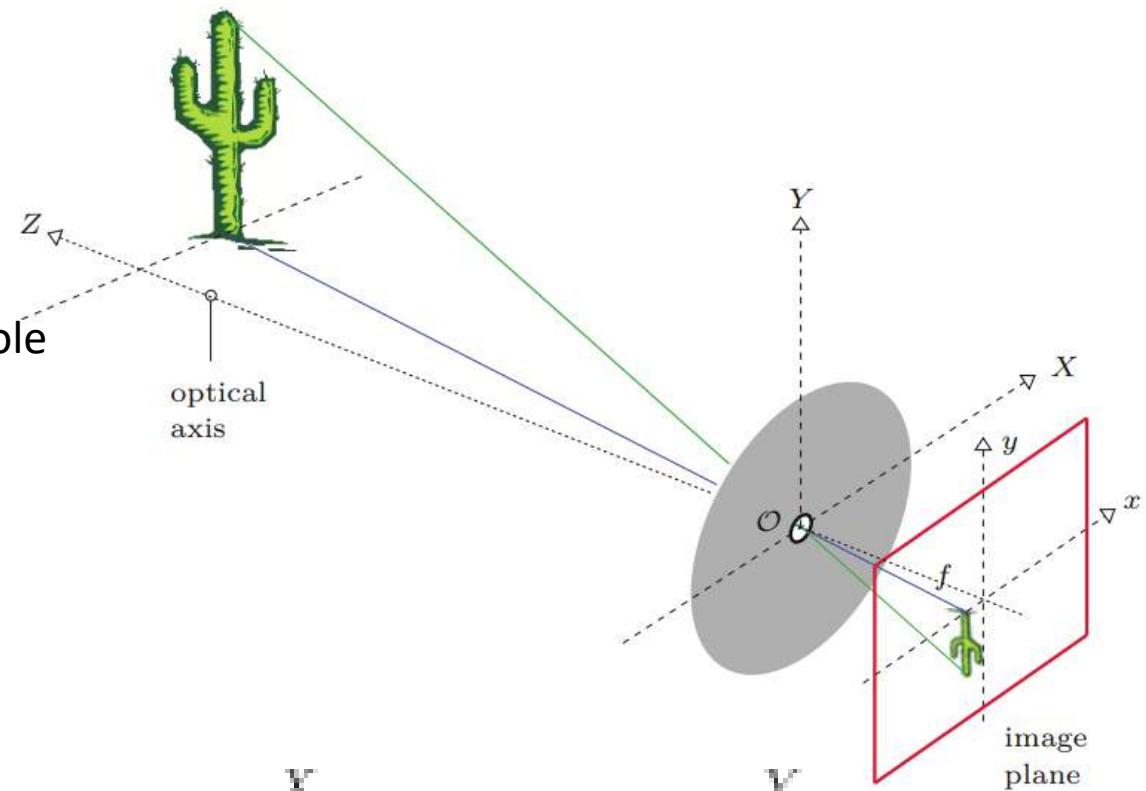
0.6mm



0.35 mm

“Pinhole” camera model

- optical axis runs through the pinhole perpendicular to the image plane between the object and pinhole
- object (cactus) is located at a horizontal distance Z from the pinhole and vertical distance Y from the optical axis.
- the pinhole has to be a small opening to produce a sharp image.
- height of the projection y is determined by two parameters: the fixed depth of the camera box f and the distance Z to the object from the origin of the coordinate system
- pinhole opening serves as the origin (O) of the 3D coordinate system (X, Y, Z) for the objects in the scene
- 2D coordinate system (x, y) describes the projection points on the image plane at a single Z .
- distance f (“focal length”) between the opening and the image plane determines the scale of the projection.



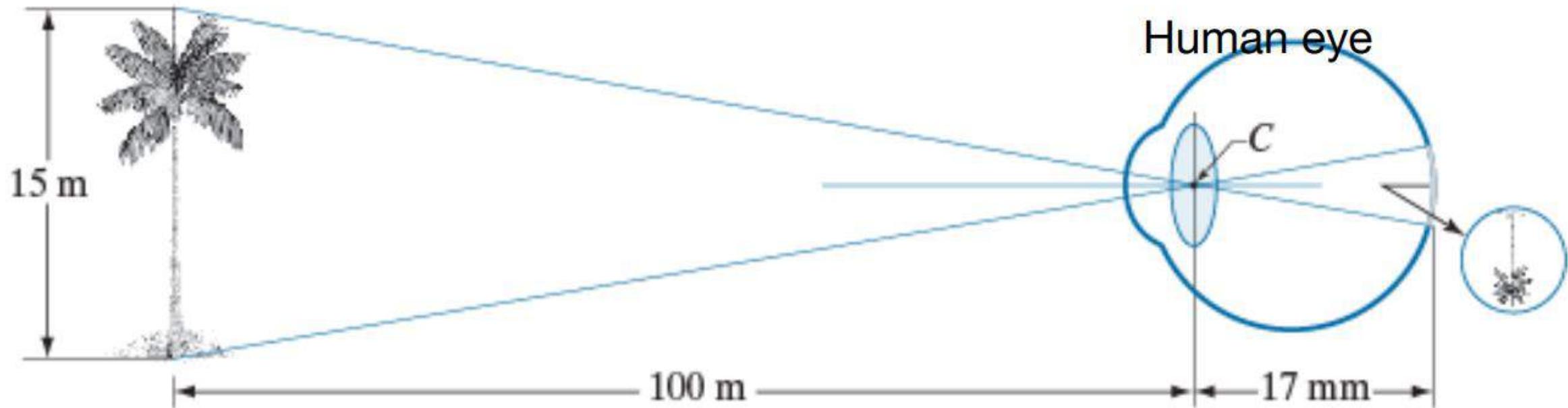
$$x = -f \cdot \frac{X}{Z} \quad \text{and} \quad y = -f \cdot \frac{Y}{Z}$$

negative sign means that the projected image is flipped in the horizontal and vertical directions and rotated by 180°.

For a fixed image, a small f (i.e., short focal length) results in a small image and a large viewing angle, just as occurs when a wide-angle lens is used, while increasing the “focal length” f results in a larger image and a smaller viewing angle

The optics of the human eye: How do we see?

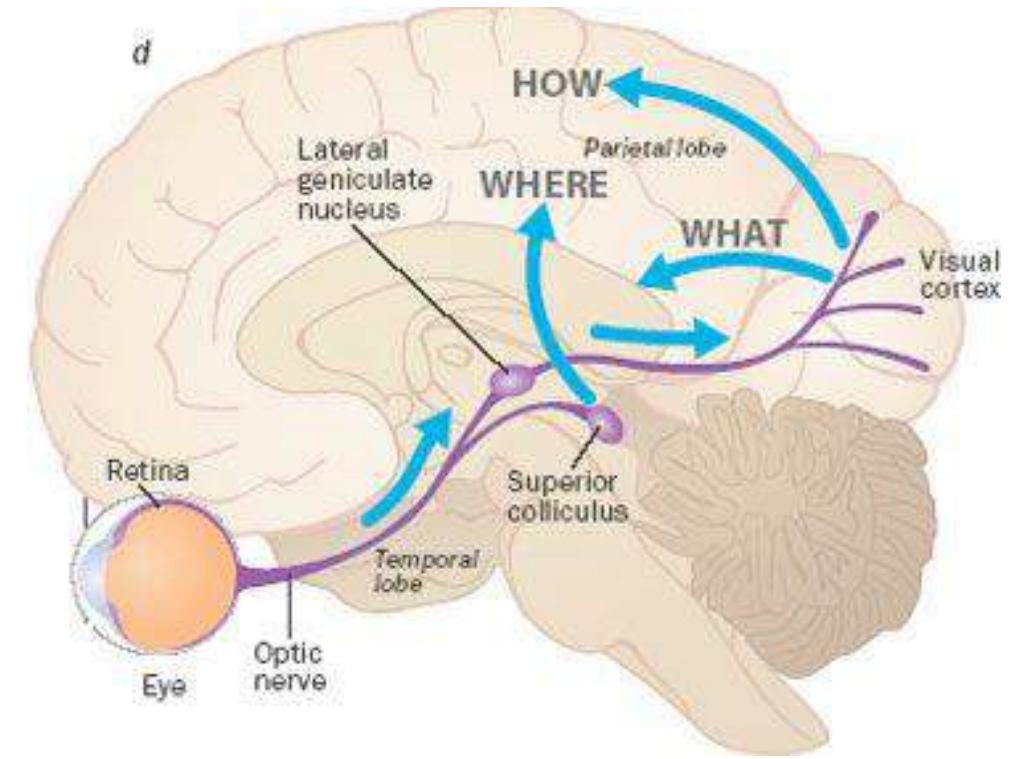
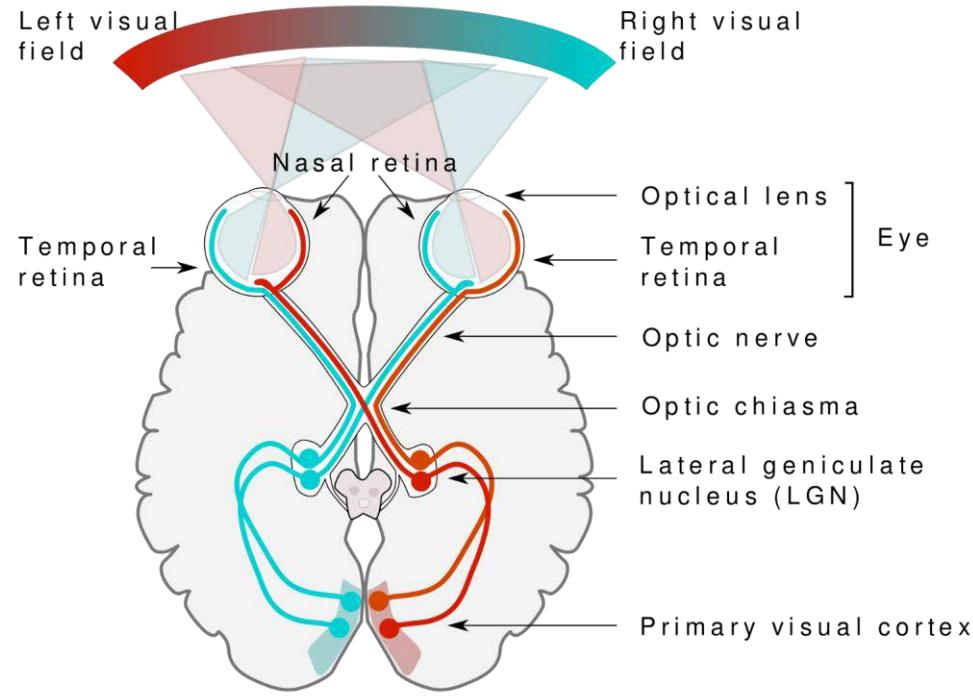
-eye lens and image formation



In the human eyes, the distance between the center of the lens and the retina is fixed (approximately 17 mm), and the focal length is changed by varying the shape of the lens (14-17 mm)!

The evolution of an eye

'Where- spatial vision' & 'What- object recognition' pathways



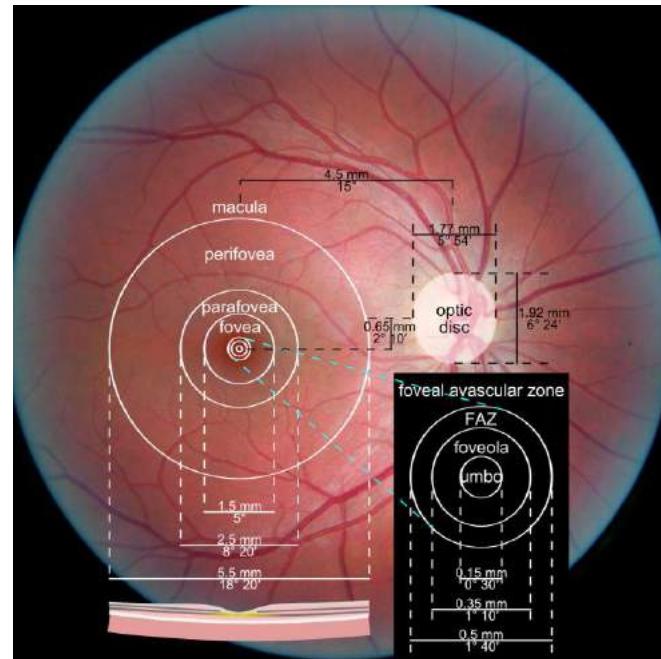
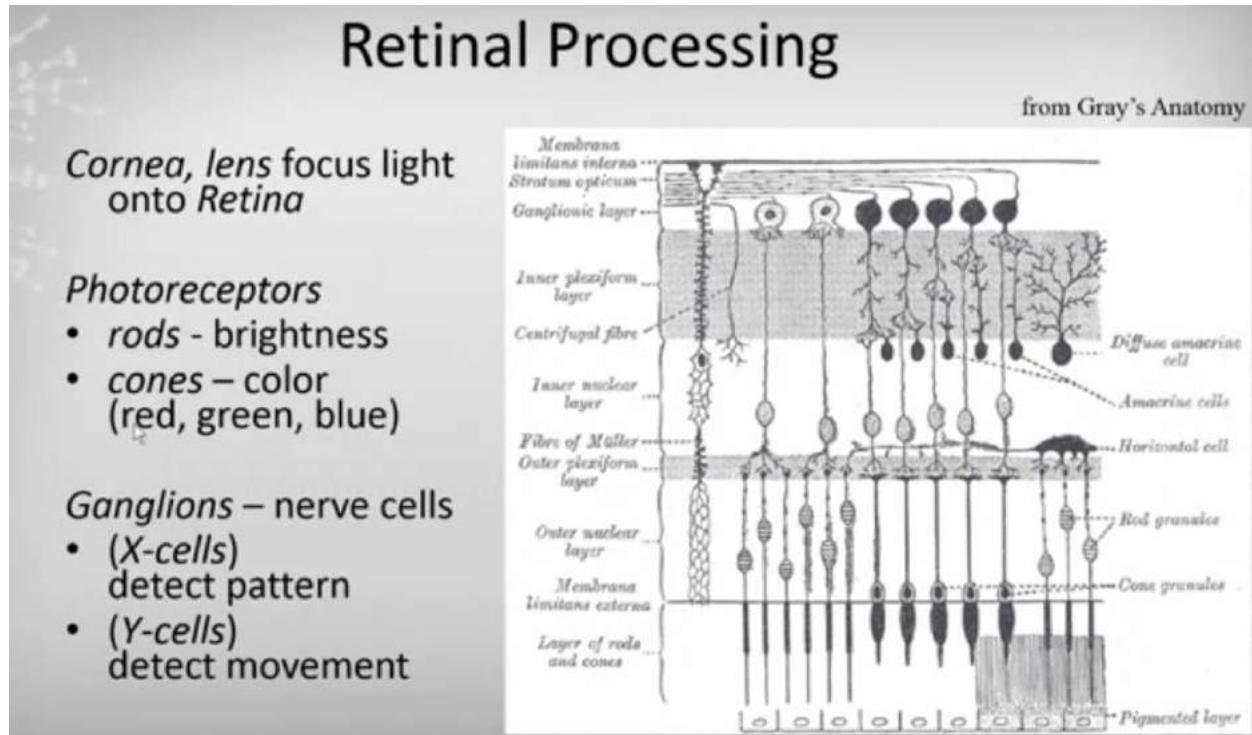
Visual cortex is hierarchically organised and specific cells respond selectively to different images

When we see an object, the light receptors in the eyes send signals via the optic nerve to the **primary visual cortex**, where the input is being processed.

<https://visionhelp.wordpress.com/2018/07/03/who-what-when-where-and-how-in-the-visual-system/>

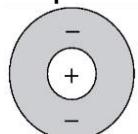
The Visual Pathway. — Source: https://commons.wikimedia.org/wiki/File:Human_visual_pathway.svg

How does the retina process an image?



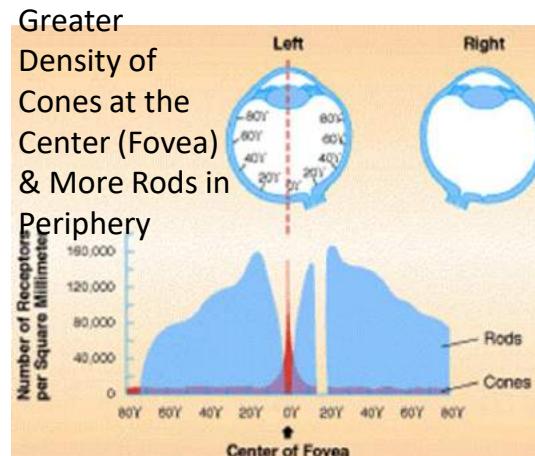
Photograph: Danny Hope from Brighton & Hove,
Photograph:Righ_eye_retina.jpg

- eye movements are used to bring interesting image regions into the fovea, usually determined by task goals
 - high resolution captured in the fovea whereas a low acuity image is formed in the perifoveal
- Image filtering in space and time in the retina



On center, Off surround cell

- light influences the response of a ganglion cell called “receptive field”
- receptive field acts as space-time filter



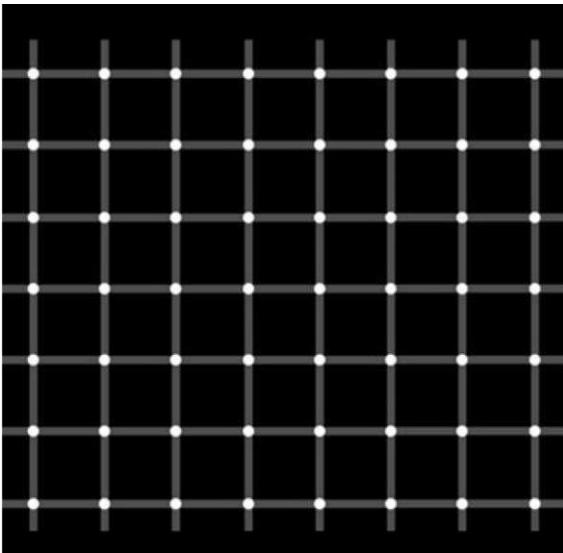
Source: Adapted from Lidsay & Norman, 1977.

Copyright © 2001 by Allyn & Bacon

Computer vision inspired by human vision: mimic the same strategy to design image descriptors

Analogy: from retinal photoreceptors to pixels

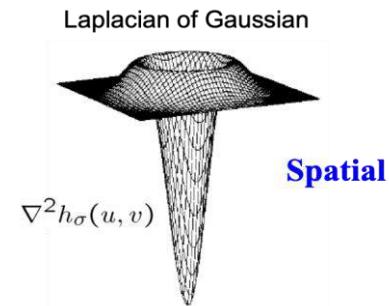
Our retinal filters at work: black dots or white dots?



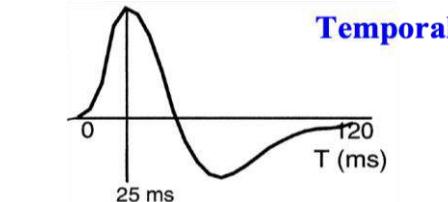
Eg of optical illusion

Retina takes spatial and temporal derivatives : DoG filter
(identifies edges, used as feature enhancement algorithm)

LoG



Spatial

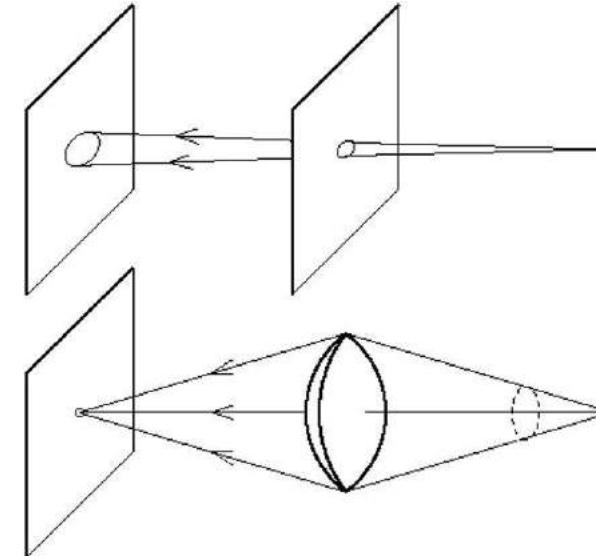
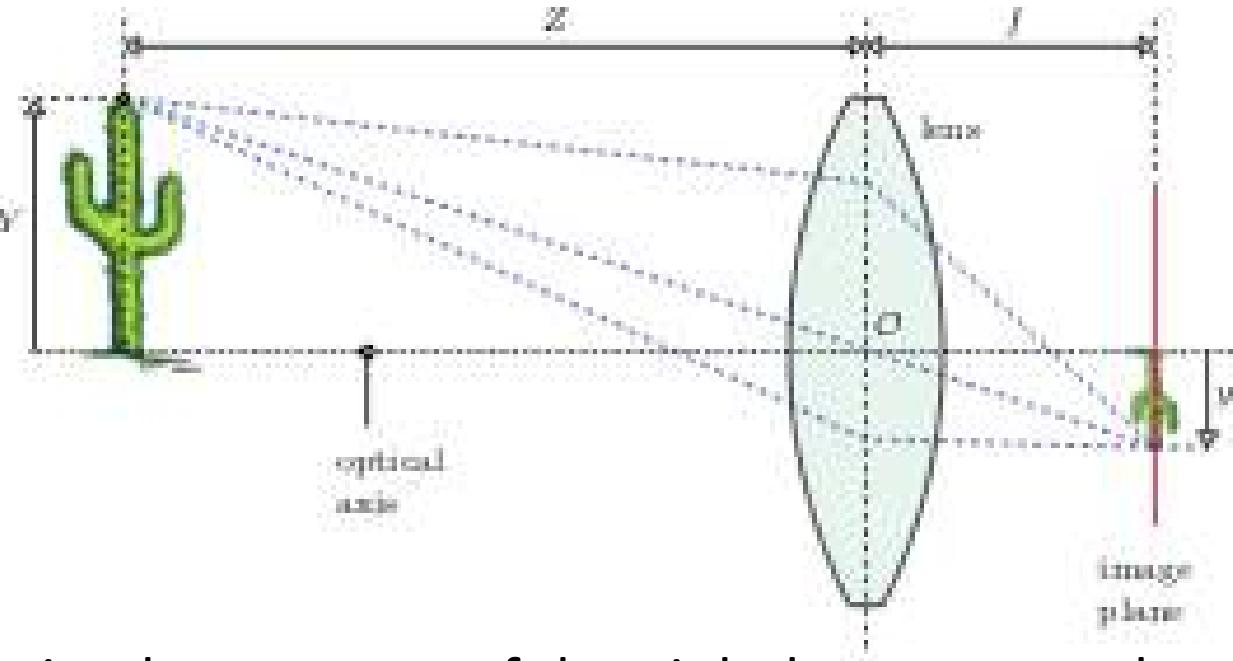


(Rao & Ballard, 1999a; 1999b)

Computer vision tools

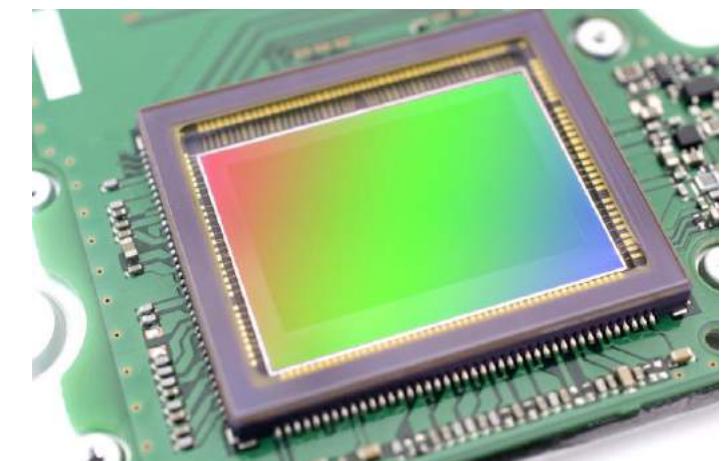
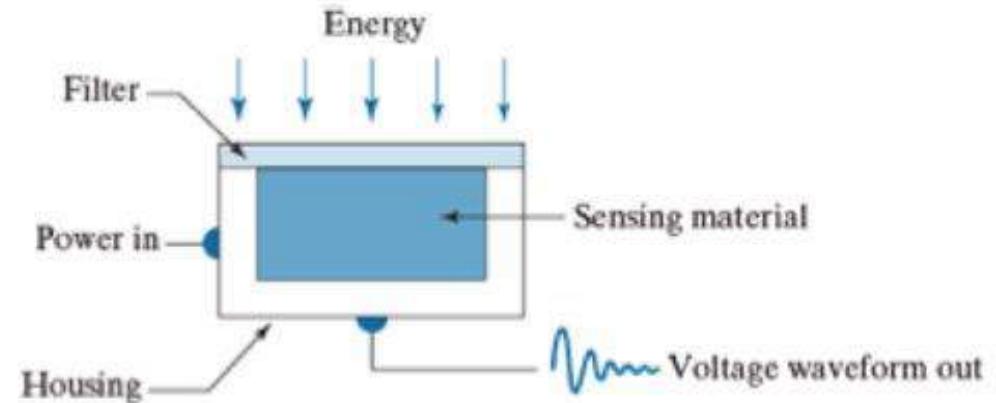
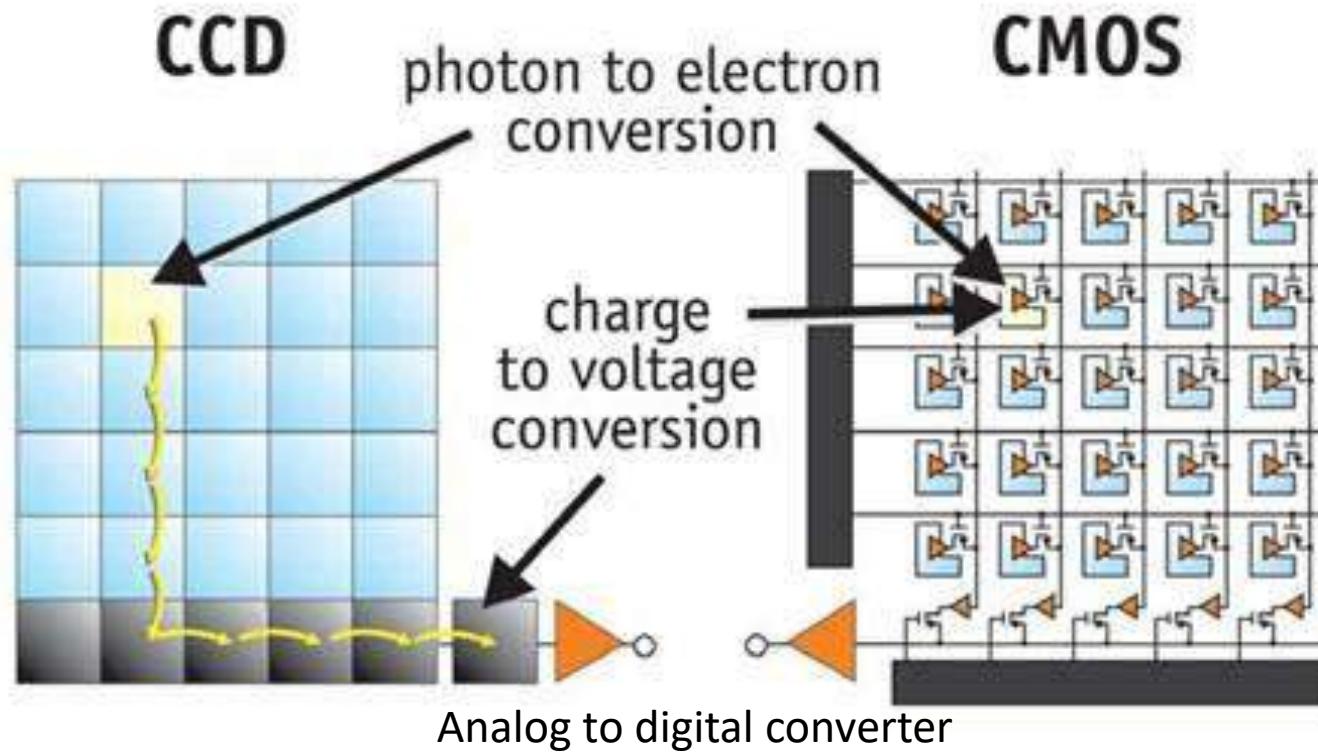
- macro-level analysis
- problem specific features
- local features to better handle scale changes, rotation

Thin lens (/pinhole) model



- simple geometry of the pinhole camera makes it useful for understanding its basic principles
- in practice a very small opening (aperture) is required to produce a sharp image (much long exposure time).
- In reality, a system of systems of optical lenses are used with superior optical properties.
- For our purposes, we replace the "pinhole" with a thin lens such that the resulting image geometry is the same as that of the pinhole camera.
- lens is assumed to be symmetric and infinitely thin, such that all light rays passing through it cross through a virtual plane in the middle of the lens.

Image sensors & digitizers



Picture: https://meroli.web.cern.ch/lecture_cmos_vs_ccd_pixel_sensor.html

A physical device such as charged coupled device (CCD) or (complementary metal-oxide semiconductor (CMOS) produce electrical outputs proportional to the light intensity

A Simple model of image formation

- The scene is illuminated by a single source.
- The scene reflects radiation towards the camera.
- The camera senses it via solid state cells (CCD cameras)

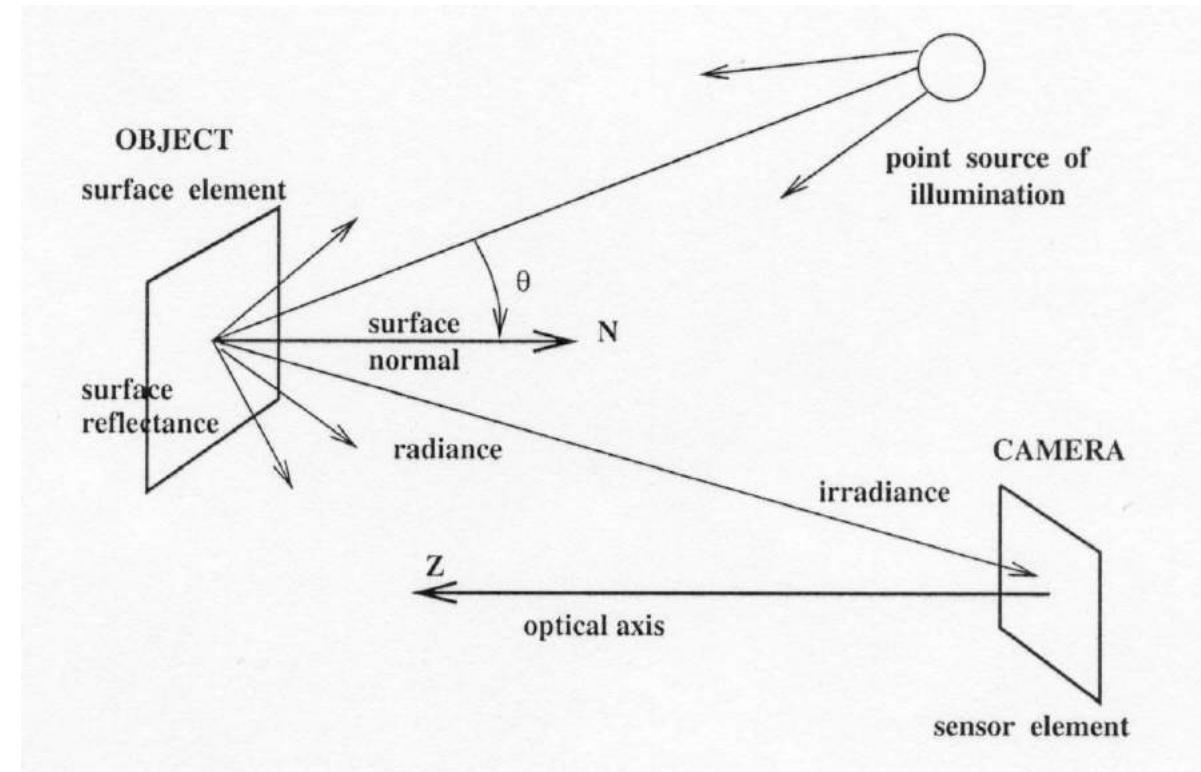


Image Formation

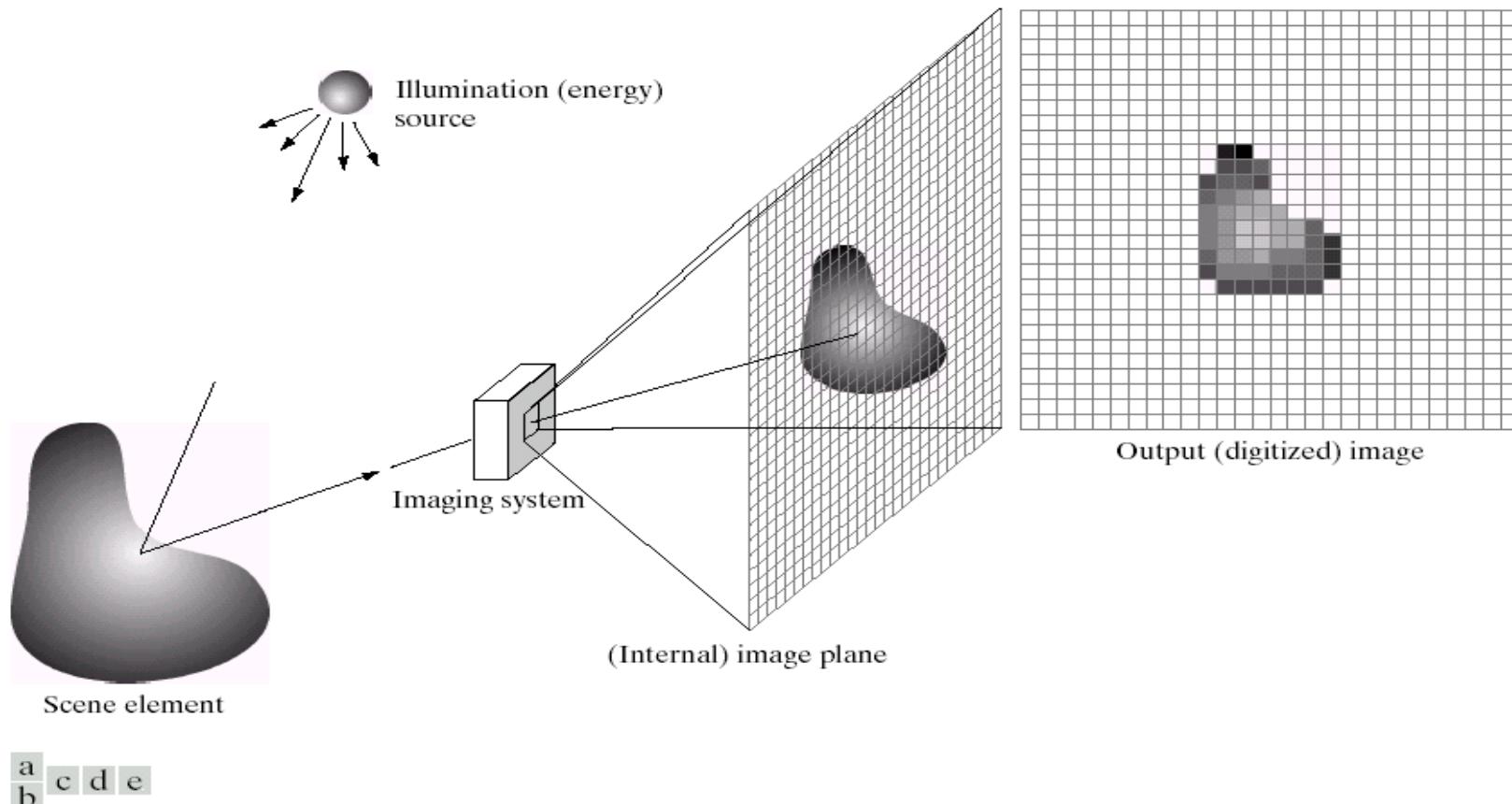


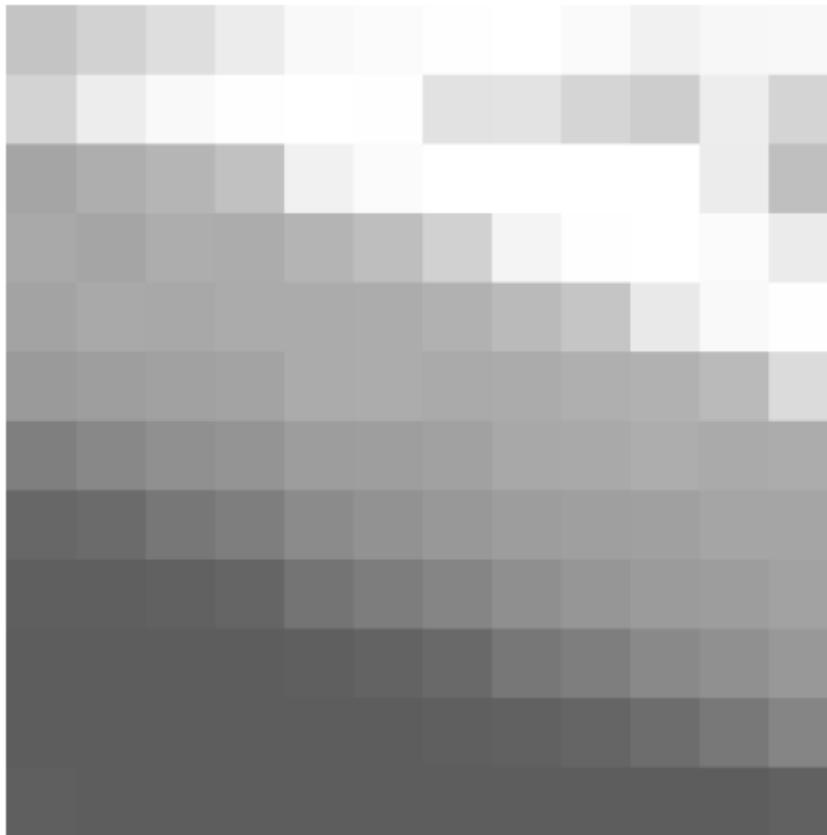
FIGURE 2.15 An example of the digital image acquisition process. (a) Energy (“illumination”) source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

Image formation (cont'd)

- There are two parts to the image formation process:
 - (1) The **geometry**, which determines where in the image plane the projection of a point in the scene will be located.
 - (2) The **physics of light**, which determines the brightness of a point in the image plane.

Simple model: $f(x,y) = i(x,y) r(x,y)$
i: illumination, r: reflectance

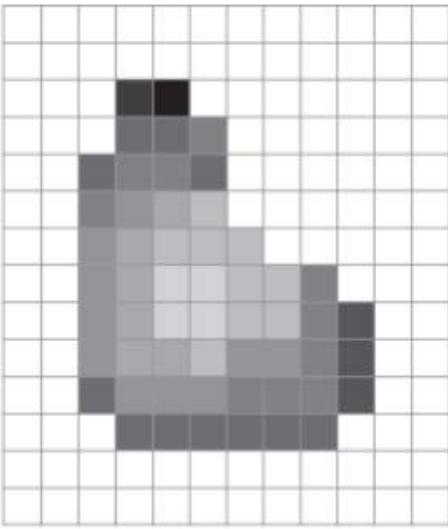
Image Formation



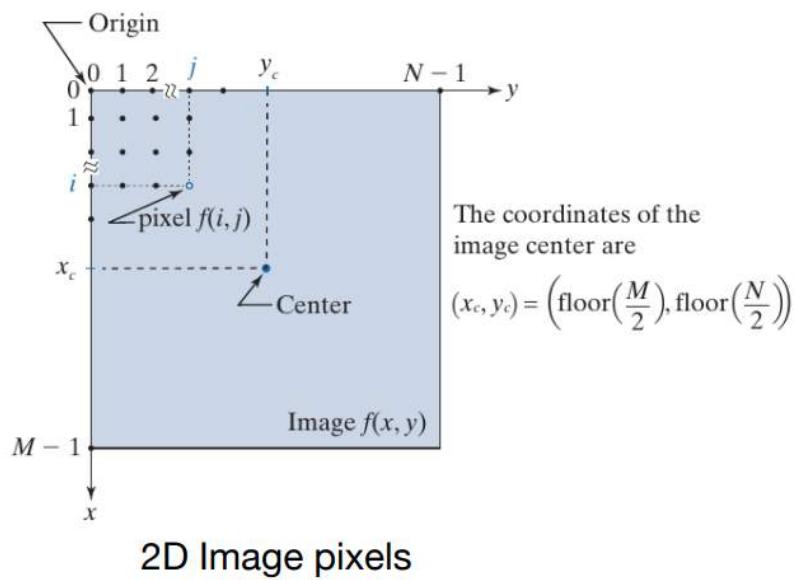
195	209	221	235	249	251	254	255	250	241	247	248
210	236	249	254	255	254	225	226	212	204	236	211
164	172	180	192	241	251	255	255	255	255	235	190
167	164	171	170	179	189	208	244	254	255	251	234
162	167	166	169	169	170	176	185	196	232	249	254
153	157	160	162	169	170	168	169	171	176	185	218
126	135	143	147	156	157	160	166	167	171	168	170
103	107	118	125	133	145	151	156	158	159	163	164
095	095	097	101	115	124	132	142	117	122	124	161
093	093	093	093	095	099	105	118	125	135	143	119
093	093	093	093	093	093	095	097	101	109	119	132
095	093	093	093	093	093	093	093	093	093	093	119

$f(x,y,t)$ is the intensity at (x,y) at time t

The digitized image



Digital image



- A two-dimensional function, $f(x, y)$, with x and y as spatial coordinates (2D), or $f(x,y,z)$ in 3D.
- Each digital image is composed of a finite number of elements called pixels or in 3D, they are called voxels.
- In case of high spatial and temporal imaging, you can process 4-dimensional images i.e. in $f(x,y,z)$ 3D space + T(time).
- A digital image I is a 2D function that maps from the domain of integer coordinates $N \times M$ to a range of possible pixel values P such that $I(x,y) \in P$ and $x,y \in N,M$.
- A two-dimensional function, $f(x, y)$, with x and y as spatial coordinates containing M rows and N columns, and the amplitude of f at any pair of coordinates (x, y) as the intensity or gray level of the image at that point.
- The spatial coordinate values are shown by integers as: $x=0, 1, 2, \dots, M-1$ and $y=0, 1, 2, \dots, N-1$.
- For image $f(x, y)$, we have L number of intensity levels, represented as a power of 2 such that $L=2^k$. For example, in an 8-bit image, we have 256 intensity levels ranging from 0 to $255(2^k - 1)$, with $k=8$.

2D Image matrix

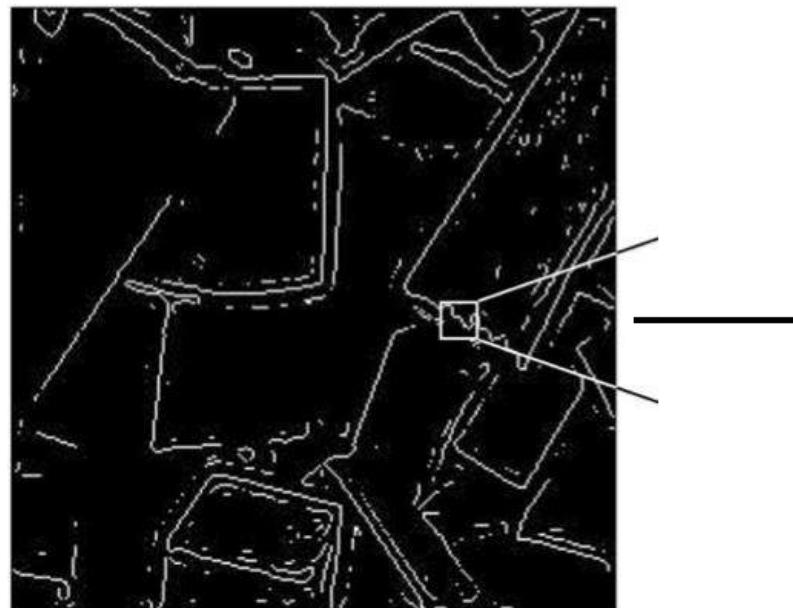
$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix}$$

Images are matrices, or discrete functions, with the number of dimensions typically ranging from one to five and where each dimension corresponds to a parameter, a degree of freedom, or a coordinate needed to uniquely locate a sample value.

Types of digital images

Binary images

Binary: Each pixel is just black or white. Since there are only two possible values for each pixel (0,1), we only need one bit per pixel.



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

Grayscale images

Grayscale: Each pixel is a shade of gray, normally from 0 (black) to 255 (white), represented by 8-bits, or exactly one byte. The images are generally powers of 2.



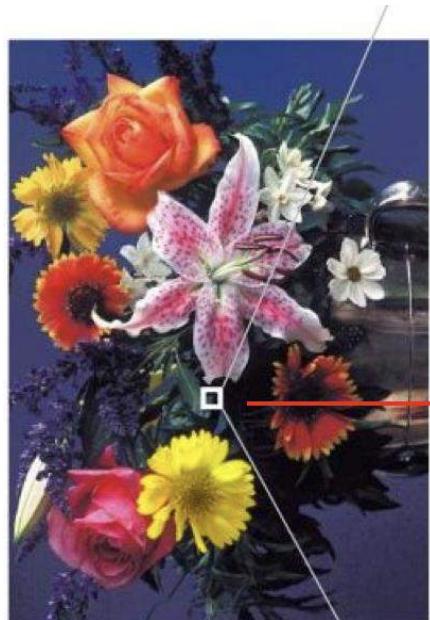
An arrow points from the highlighted pixel in the image to the first cell of the following 8x8 grid, representing the 8-bit grayscale values for that image area.

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

An 8-bit grayscale image

True Color, or RGB images

- Each pixel has a particular color; that color is described by the amount of red, green and blue in it.
- If each of these components has a range 0–255, this gives a total of 256^3 different possible colors.
- Such an image is a “stack” of k three matrices; representing the red, green and blue values for each pixel. This means that for every pixel there are corresponding 3 values.



Red	Green	Blue
49	55	56
58	60	60
58	58	54
83	78	72
88	91	91
69	76	83
61	69	73
64	76	82
93	93	91
88	82	88
125	119	113
137	136	132
105	108	114
96	103	112
81	93	96
83	83	91
135	128	126
141	129	129
95	99	109
84	93	107
80	80	80
86	86	86
88	88	88
111	111	110
126	126	126
117	117	115
112	112	109
105	105	105
107	107	102

Image size, resolution, bit depth

Image size: The size of an image is determined from the width M (number of columns) and the height N (number of rows) of the image matrix I .

Image resolution: The spatial dimensions of the image in the real world and is given as the number of image elements per measurement; for example, dots per inch (dpi); pixels per kilo-meter for satellite images ; pixels per micrometer for microscopy images.

Pixel values and image depth: Pixel values are binary words of length k so that a pixel represents any of 2^k different values. The value k is called the bit depth (or just “depth”) of the image. The exact bit-level layout of an individual pixel depends on the kind of binary, grayscale, or RGB color.

Spatial Resolution

- the size of the smallest perceptible details in an image measured by the number of pixels per unit distance.
- Spatial resolution is dependent on the sampling rate or the number of samples.
- The spatial resolution of an image is important during image acquisition step and affects what information we may want to extract of the image. It is the measure of the smallest discernible detail in an image.



Effects of reducing spatial resolution. The images shown are at:
(a) 930 dpi,
(b) 300 dpi,
(c) 150 dpi, and
(d) 72 dpi.

(basic image processing steps such as "point operations" or "filters" operate the same manner for all spatial resolutions. spatial resolution is important when distances within an image need to be measured.

Becomes extremely relevant to resolve two closely spaced objects (need more sampling to resolve them). Typically, the image operations are performed on pixel-images and the image is scaled to real-world dimensions to extract quantitative information.)

Intensity Resolution

- Intensity resolution: the smallest discernible change in the intensity level.
- Measured in the number of bits (k) used for quantisation. (higher the k, more information).

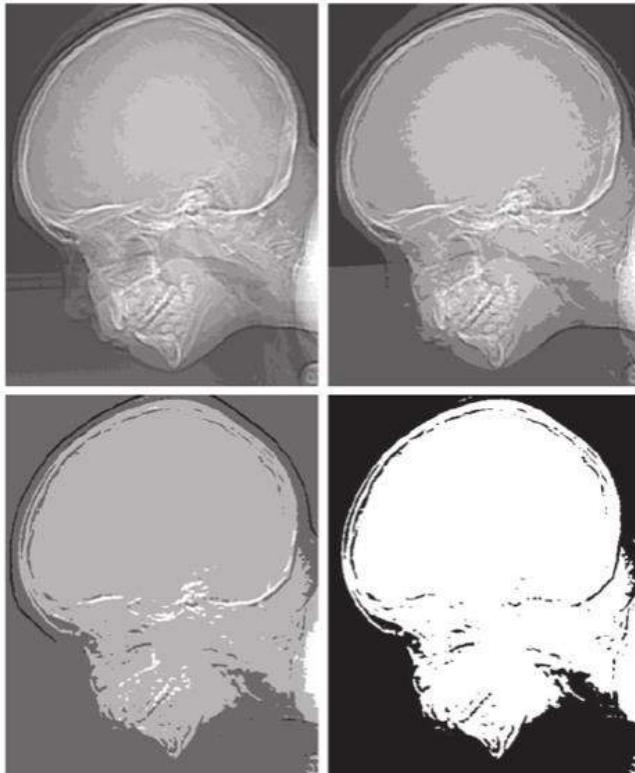


Image displayed in 16, 8, 4, and 2 intensity levels.

Eg: 8 bit image pixels (0-(2^8)-1)/ 0-255

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

Spatial Resolution & intensity resolution

Spatial resolution and intensity resolution depend upon digitization of the image,

- Spatial resolution depends on the number of samples (N)
- Intensity resolution depends on the number of bits (k)

The artefacts in images due to low resolution affects the image differently, for example:

- Too low spatial resolution results in jagged lines
- Too low intensity resolution results in false contouring

Sensitivity:

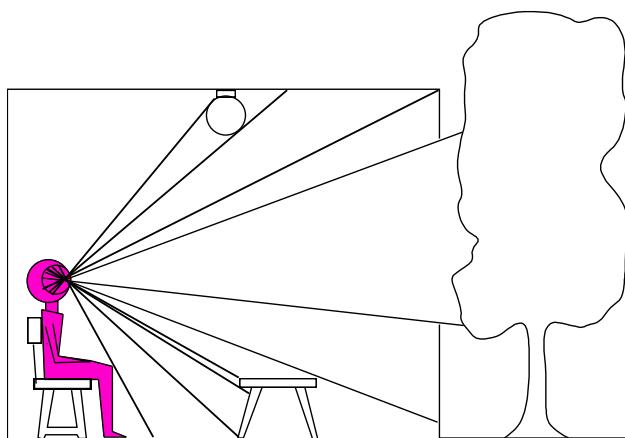
- Spatial resolution is more sensitive to the shape variations
- Intensity resolution is more sensitive to the lighting variations

Mathematical tools required for CV

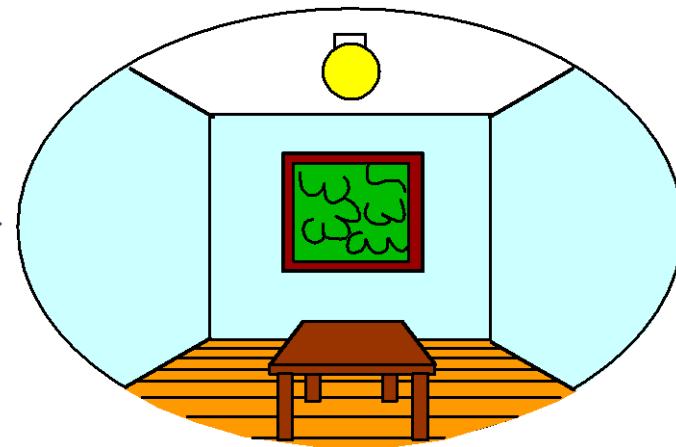
Camera projections

the formation of a two-dimensional representation of a three-dimensional world, and what we may deduce about the 3D structure of what appears in the images.

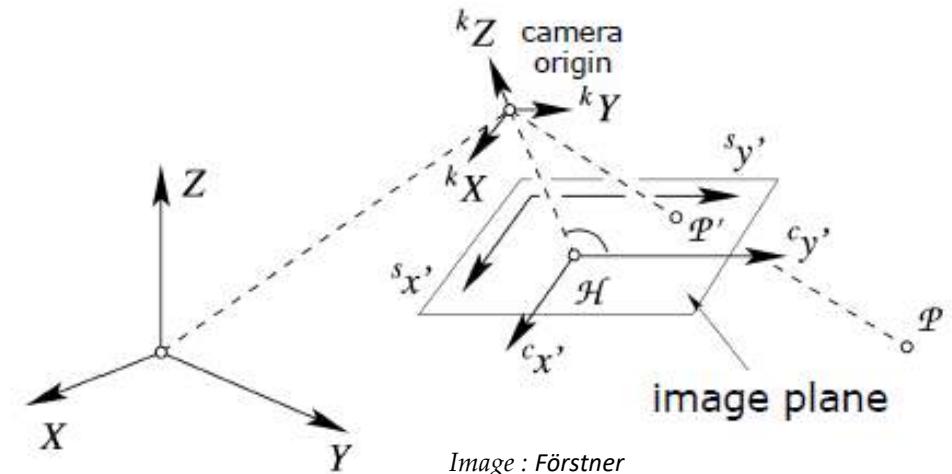
3D world



2D image



Point of observation



Topics for revision required for the course

For processing digitized images (within the plane)

- Linear systems
- Matrices, vectors
- Probability & statistics
- Set theory
- Differentiation & integration

For camera calibration & understanding image geometry

- Linear systems
- Matrices, vectors
- least square problems, SVD, QR factorization
- image transformations (next lecture, only within image coordinate frame)



Course Notes

BHAVNA R, IISER-BHOPAL 2021/22
DSE312-CVLecture3
DSE-409/609-DIPlecture2

Mathematical tools in image processing

WEEK-2

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Mathematical tools in image processing

In a digital image the values are all discrete, usually only integer values.

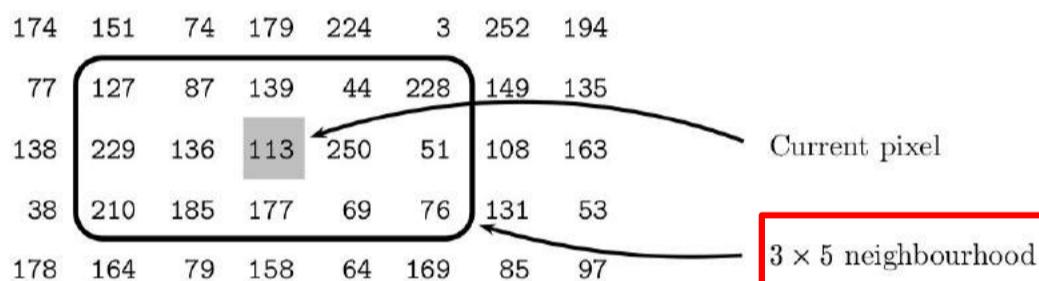
A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it. These dots are simply pixels.

Relationships between pixels in a digital image

Neighborhood pixels: The pixels surrounding a given pixel constitute its neighborhood.

A neighborhood can be characterized by its shape in the same way as a matrix: we can speak of a 3x3 neighborhood, or of a 3x5 neighborhood.

Eg:



Typically, we consider 4-neighbourhood or 8-neighbourhood while operating 2D images.

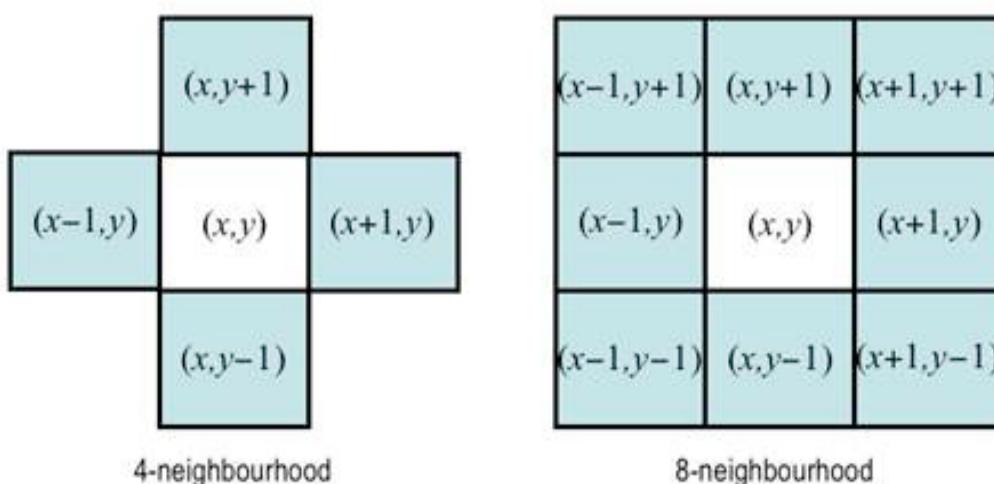
The 4-neighborhood of pixel $p(x,y)$ is the set ($N_4(p)$):

$$\{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\} \text{ or } \{(i, j): |x-i| + |y-j| = 1\}$$

The 8-neighborhood of pixel $p(x,y)$ is the set ($N_4(p) + N_D(p)$):

$$\{(x-1, y), (x+1, y), (x, y-1), (x, y+1), (x-1, y-1), (x+1, y-1), (x-1, y+1), (x+1, y+1)\}$$

or $\{(i, j): \max(|x-i|, |y-j|) = 1\}$



Adjacency

4-adjacency: Two pixels p and q with values from V are 4-adjacent if q is in the set $N_4(p)$.

8-adjacency: Two pixels p and q with values from V are 8-adjacent if q is in the set $N_8(p) (=N_4(p)+ N_D(p))$.

Mixed adjacency is a modification of 8-adjacency, and is introduced to eliminate the ambiguities that may result from using 8-adjacency.

Let V be the set of intensity values used to define adjacency. In a binary image, $V = \{1\}$ if we are referring to adjacency of pixels with value 1.

m-adjacency (also called mixed adjacency). Two pixels p and q with values from V are m-adjacent if

- q is in $N_4(p)$, or
- q is in $N_D(p)$ and the set $N_4(p) \cap N_4(q)$ has no pixels whose values are from V. ('D' stands for diagonal)

For example, consider the pixel arrangement below and let $V = \{1\}$.

The three pixels at the top show multiple (ambiguous) in 8-adjacency, as indicated by the dashed lines. This ambiguity is removed by using m-adjacency. In other words, the center and upper-right diagonal pixels are not m-adjacent because they do not satisfy condition in 8-adjacency.

3×3 pixels 8-adjacency m-adjacency

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

Pixel connectivity

A 4-connected path from a pixel p_1 to another pixel p_n is defined as the sequence of pixels $\{p_1, p_2, \dots, p_n\}$ such that p_{i+1} is a 4-neighbour of p_i for all $i = 1, \dots, n-1$.

In other words, two pixels p and q are 4-adjacent (or connected) if q is in $N_4(p)$ or p is in $N_4(q)$. N stands for neighbourhood here.

The path is 8-connected if p_{i+1} is an 8-neighbour of p_i .

- A set of pixels is a 4-connected region if there exists at least one 4-connected path between any pair of pixels from that set.
- The 8-connected region has at least one 8-connected path between any pair of pixels from that set.

A digital path (or curve) from pixel p with coordinates (x,y) to pixel q with coordinates (s,t) is a sequence of distinct pixels with coordinates

Path between $p(x,y)$ and $q(s,t)$:

$$(x_0, y_0), (x_1, y_1) \dots (x_n, y_n)$$

where

$$(x, y) = (x_0, y_0), (s, t) = (x_n, y_n)$$

and pixels

(x_i, y_i) and (x_{i-1}, y_{i-1}) are adjacent for $1 \leq i \leq n$

where, n is the length of the path. closed path: If $(x_0, y_0) = (x_n, y_n)$.

Connected components and connected set of pixels

Let S represent a subset of pixels in an image. Two pixels p and q are said to be connected in S if there exists a path between them consisting entirely of pixels in S . For any pixel p in S , the set of pixels that are connected to it in S is called a connected component of S . If it only has one component, and that component is connected, then S is called a connected set.

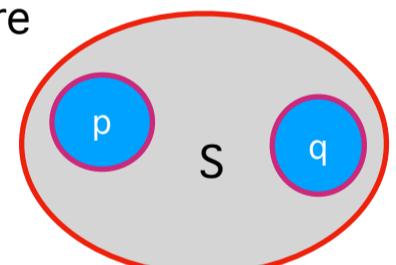


Image region

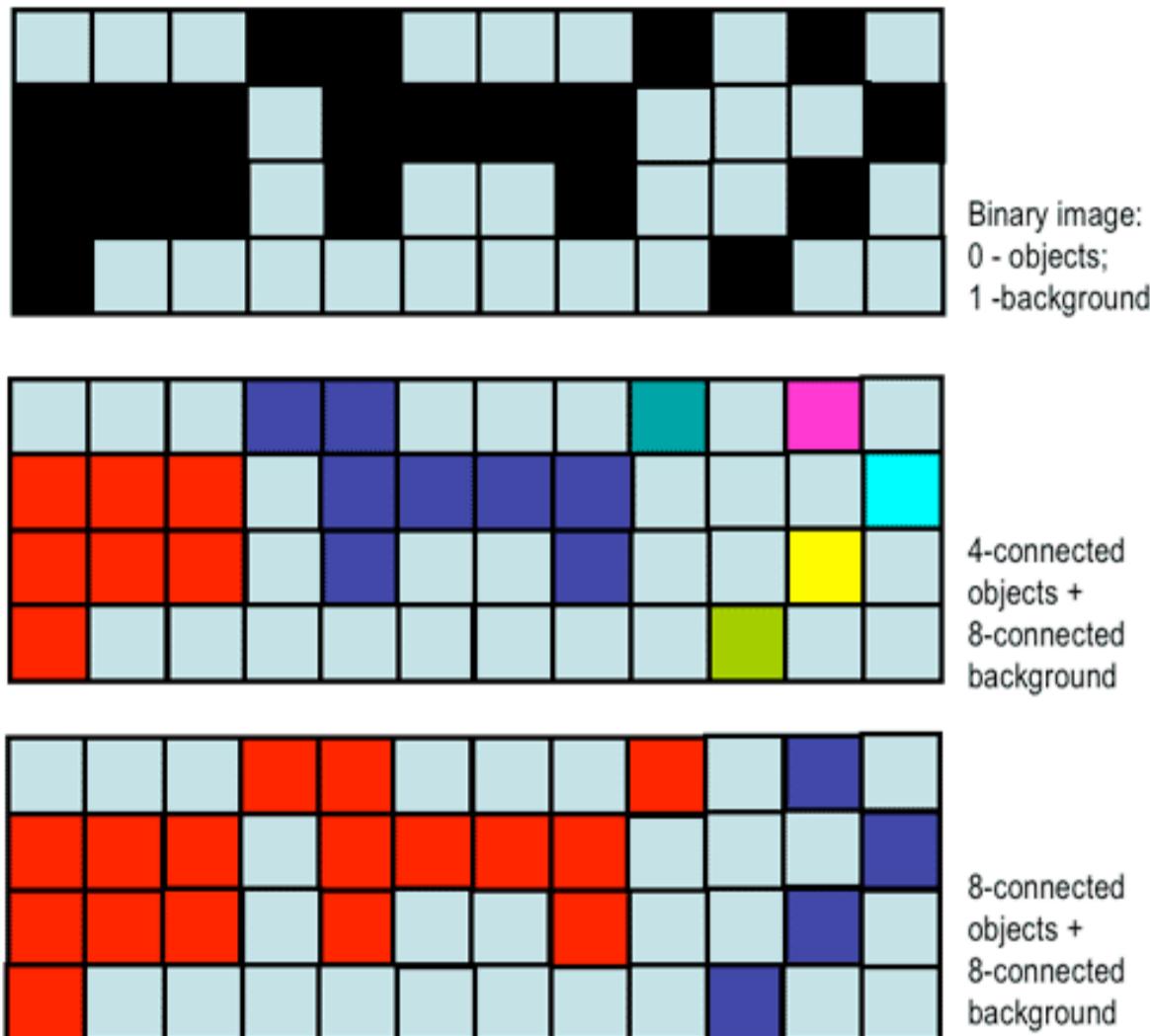
Let R represent a subset of pixels in an image. We call R a region of the image if R is a connected set. Two regions, R_i and R_j are said to be adjacent if their union forms a connected set. Regions that are not adjacent are said to be disjoint. We consider 4- and 8-adjacency when referring to regions.

1	1	1	R_i
1	0	1	
0	1	0	
0	0	1	R_j
1	1	1	
1	1	1	

How does connectivity affect objects identified within an image?

Pixel connectivity is useful for defining object boundary and image regions.

The 4- and 8-connectivity produce different segmentation results, here all pixels of the same color are connected to each other and hence defined as ‘single objects’.



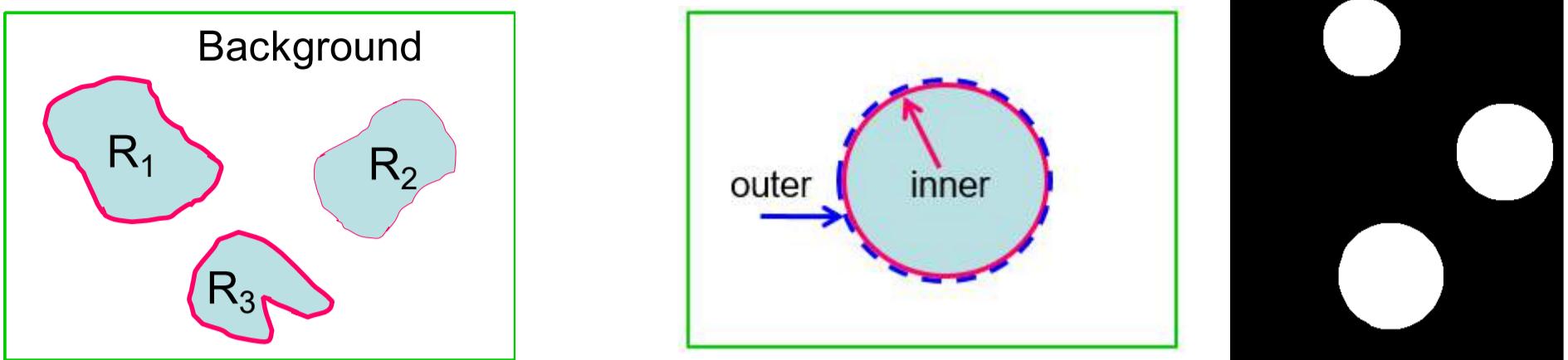
This also shows us how we label the connected pixels based on the connectivity.

Pixel Labeling

To assign different unique labels to all the disjoint connected components of an image.

How is this achieved automatically?: Scan through pixel coordinates, check the value (0 or 1) and establish connected component or not.

Object boundary



Suppose an image contains K disjoint regions, R_k , $k=1,2,3,\dots,K$. Let R_u denote the union of all of the K regions, and let $(R_u)^C$ denote its complement.

All the points in R_u are called the foreground and all points in the $(R_u)^C$ are called the background.

The inner boundary (border, or contour) of a region R is the set of pixels in R that are adjacent to pixels in the complement of R .

The outer boundary of a region is the set of pixels in the background that have at least one neighbor in R . The outer border of the region forms a closed path around the region.

Boundary (or border) is defined as the set of pixels in the first and last rows and columns of the image. Normally, when we refer to a region, we are referring to a subset of an image, and any pixels in the boundary of the region that happen to coincide with the border of the image are included implicitly as part of the region boundary.

Image Arithmetic Operations

These operations act by applying a simple function $y=f(x)$ to each gray value in the image.

- Simple functions include adding or subtracting a constant value (C) to each pixel:
- $y = f(x) \pm C$
- Multiplying each pixel by a constant: $y = C \cdot f(x)$

- Dividing each pixel by a constant: $y = C \cdot f(x)$

Example of an element-wise multiplication and addition operation with a constant ‘C’.

$$f(x) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}; \quad c \cdot f(x) = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} \\ c \cdot a_{21} & c \cdot a_{22} \end{bmatrix} \quad c + f(x) = \begin{bmatrix} c + a_{11} & c + a_{12} \\ c + a_{21} & c + a_{22} \end{bmatrix}$$

Element-wise vs. Matrix Operations

The element-wise product of two 2-by-2 images is:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \xrightarrow{\hspace{1cm}} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

The matrix product of the same images is:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \xrightarrow{\hspace{1cm}} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Mostly, we use element-wise operations in image processing.

Arithmetic operations between two images are defined as:

$$s(x, y) = f(x, y) + g(x, y)$$

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

$$p(x, y) = f(x, y) \times g(x, y)$$

$$v(x, y) = f(x, y) \div g(x, y)$$

All are element-wise operations, performed between corresponding pixel pairs in f and g for $x=0,1,2,\dots,M-1$ and $y=0,1,2,\dots, N-1$, resulting in images of the same size as inputs. These are defined for input images of the same size.

Distance Measures between pixels

For pixels $p(x,y)$, $q(u,v)$ and $s(w,z)$, D is a distance function or metric if:

- (a) $D(p,q) \geq 0$ ($D(p,q) = 0$ iff $p = q$),
- (b) $D(p,q) = D(q,p)$, and
- (c) $D(p,s) \leq D(p,q) + D(q,s)$.

Euclidean distance between p and q is defined as: $D_e(p,q) = \sqrt{(x-u)^2 + (y-v)^2}$

Manhattan distance is calculated as the sum of the absolute differences between the two vectors, which is also the D_4 (4-adjacency) distance (city-block distance) between p and q and is defined as:

$$D_4(p,q) = |x - u| + |y - v|$$

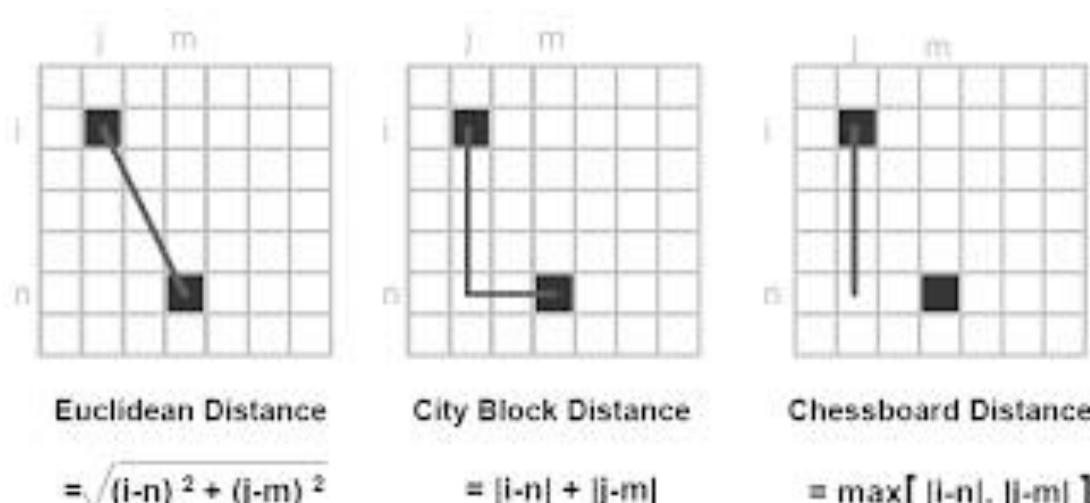
Eg; measures the distance between two points in a city if you can only travel along orthogonal city blocks, hence it is also called city-block distance.

Euclidean distance gives the minimum distance between 2 points, whereas Manhattan distance is measured along the axes at right angles.

Computational cost: For large dimensional data (say, ten thousand distance pairs), computing city-block distance is lesser expensive than Euclidean distance.

Chess-board distance (D_8 distance or 8-adjacency) between p and q is defined as:

$$D_8(p,q) = \max(|x - u|, |y - v|)$$



Linear vs. Non-Linear Operations

Assume a general operator that produces an output image, $g(x,y)$, from a given input image $f(x,y)$:

$$\mathcal{H}[f(x,y)] = g(x,y)$$

The operator is said to be linear, if it satisfies two properties: i) Homogeneity, ii) Additivity

- output of a linear operation applied to the sum of two inputs is the same as performing the operation individually on the inputs and then summing the results (Additivity).
- output of a linear operation on a constant multiplied by an input is the same as the output of the operation due to the original input multiplied by that constant (Homogeneity).

$$\begin{aligned}\mathcal{H}[af_1(x,y) + bf_2(x,y)] &= a\mathcal{H}[f_1(x,y)] + b\mathcal{H}[f_2(x,y)] \quad \text{summation is distributive} \\ &= ag_1(x,y) + bg_2(x,y)\end{aligned}$$

For example sum and mean operators are linear operator, while max operator is not.

More on Arithmetic Operations: Examples

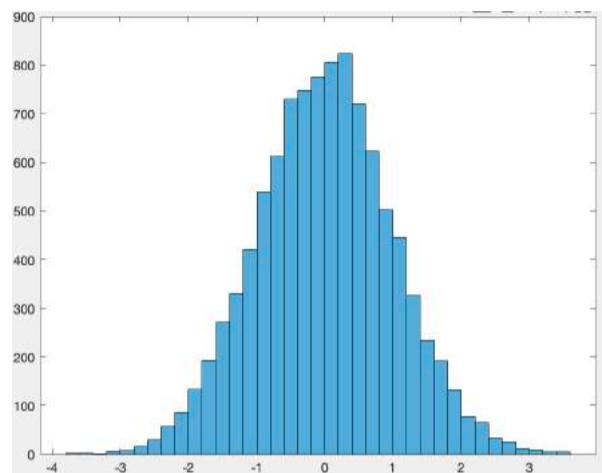
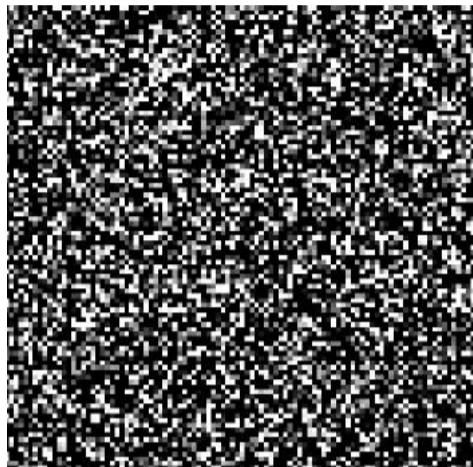
Assume that in image acquisition, zero-mean uncorrelated noise or (white noise) is added to the noiseless image:

$$g(x,y) = f(x,y) + \eta(x,y)$$

White noise has zero mean, constant variance, and is uncorrelated in time. Let us say you go to your imaging instrument for image acquisition, and there is no sample. You just record the plain background, you will may typically find such a noise. For example, here I have generated such an image (using `randn(100)` in Matlab), a normal distribution of `mean= 0` and `std=1` and is uncorrelated in time.

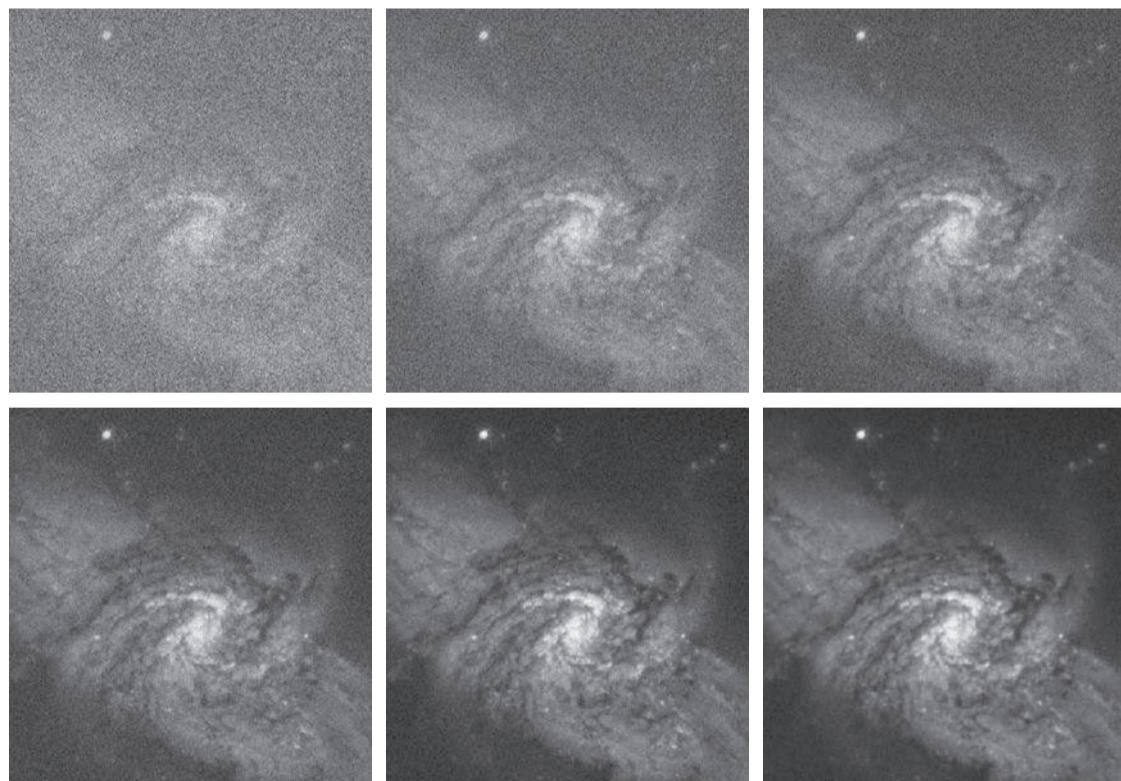
On the left is the 2D image and the histogram of all pixels is shown on the right

$$\eta(x, y)$$



If noise is zero-mean and uncorrelated with the noise-less image, averaging K different noisy images can help in reducing the noise.

For example:



(a) Image of Galaxy Pair NGC 3314 corrupted by additive Gaussian noise. (b)-(f) Result of averaging 5, 10, 20, 50, and 100 noisy images, respectively. All images are of size 566×598 pixels, and all were scaled so that their intensities would span the full $[0, 255]$ intensity scale.

Let us take an example where image difference is a useful operator to reduce noise

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

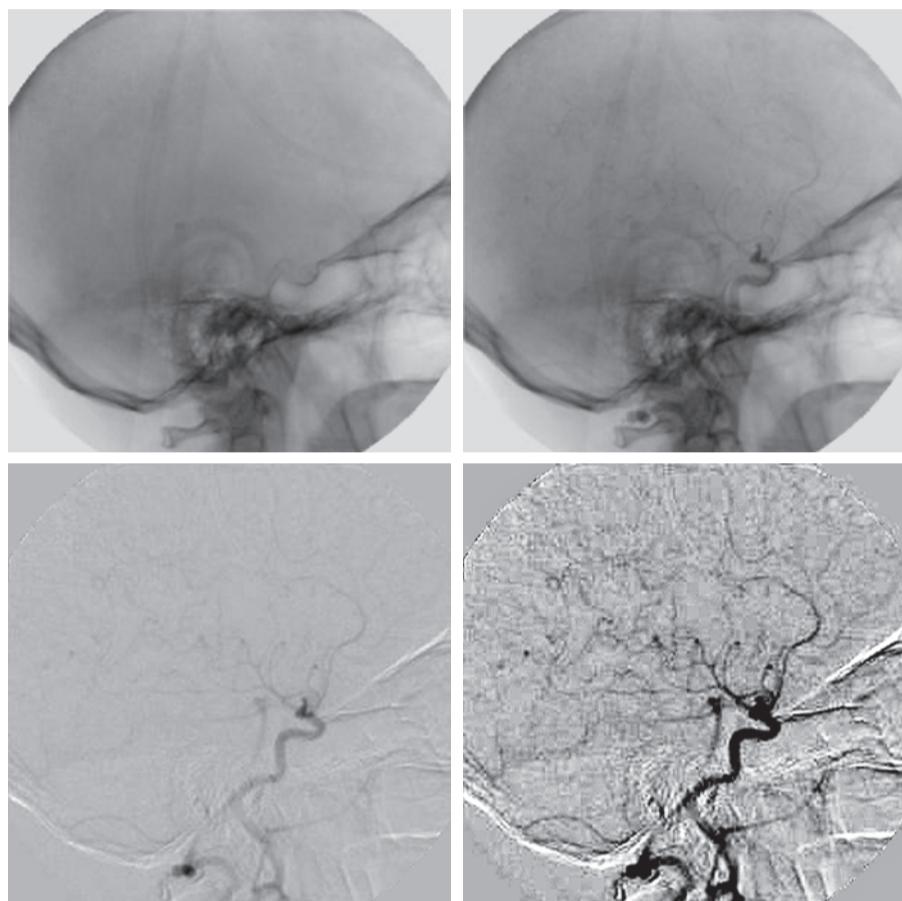
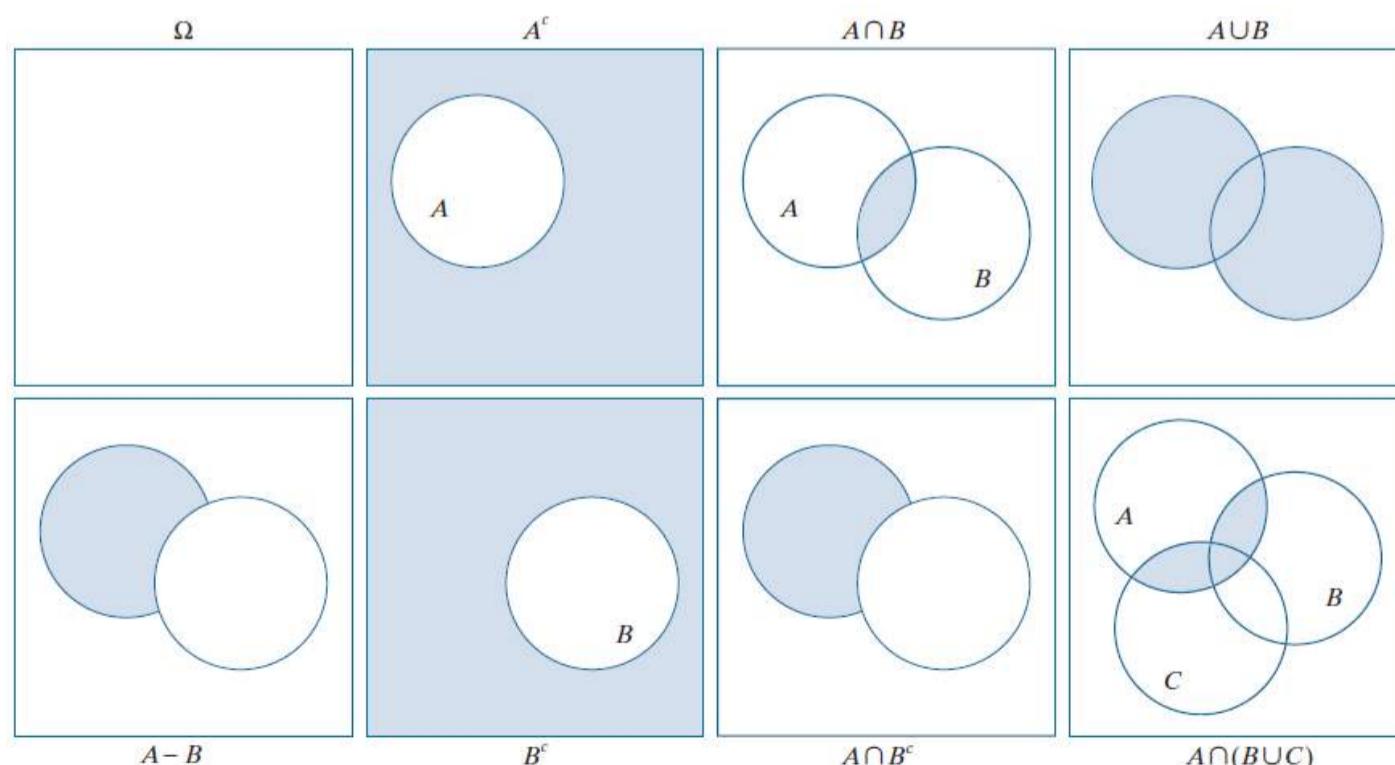


Image (a) is before iodine injection, (b) is taken after.

Image(a) -Image (b)=Image (c)

Sharpen(Image (c))=Image(d), shows fine blood vessel structures are visible in this image.

Set and logical operations on images



Logical operators are useful for finding the complement of an image.

Let the elements of a grayscale image be represented by a set A whose elements are triplets of the form (x, y, z), where x and y are spatial coordinates, and z denotes intensity values. We define the complement of A as the set

$$A^c = \{(x, y, K - z) | (x, y, z) \in A\}$$

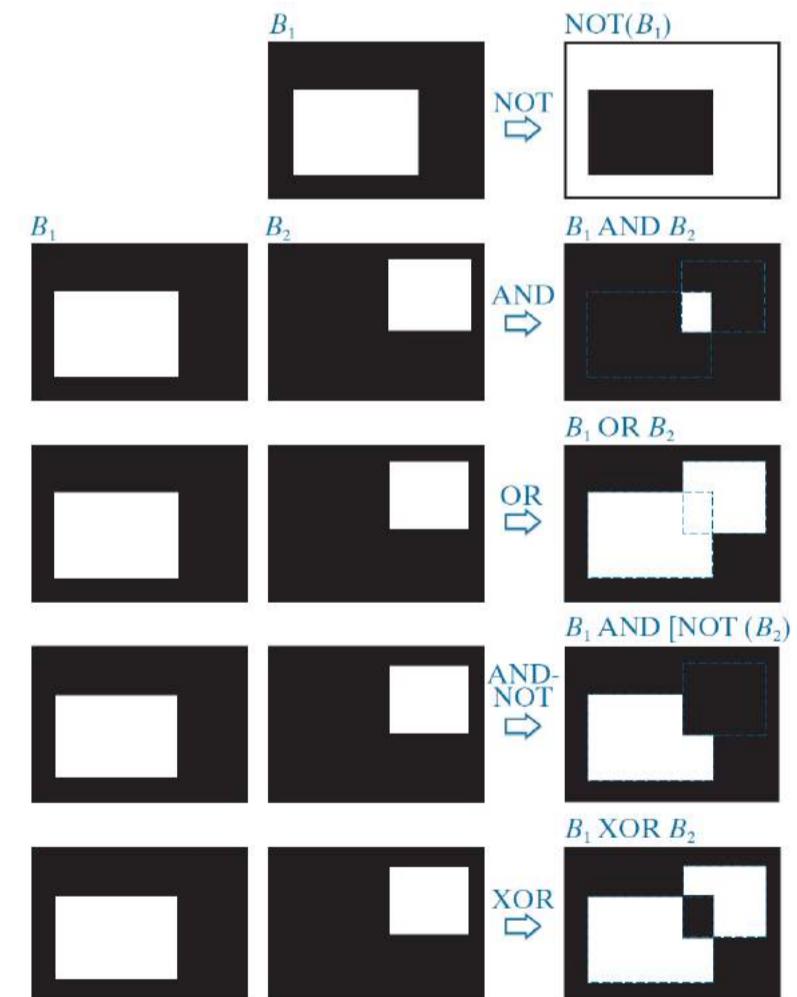
which is the set of pixels of A whose intensities have been subtracted from a constant K. This constant is equal to the maximum intensity value in the image, $2^k - 1$, where k is the number of bits used to represent z. So, for a 8-bit image K=255.

Image complement can also be applied to binary images.

Logical operators

Logical operators are only applied to binary images. A binary image can be viewed as a Venn diagram in which the coordinates of individual regions of 1-valued pixels are treated as sets.

a	b	a AND b	a OR b	NOT(a)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0



Logical operations deal with TRUE (1) and FALSE (0) variables and expressions, which means that we can apply them to binary images with foreground (1-valued) pixels and background (0-valued) pixels.

Histograms and Image Statistics

Histograms are frequency distributions, and histograms of images describe the frequency of the intensity values that occur in an image.

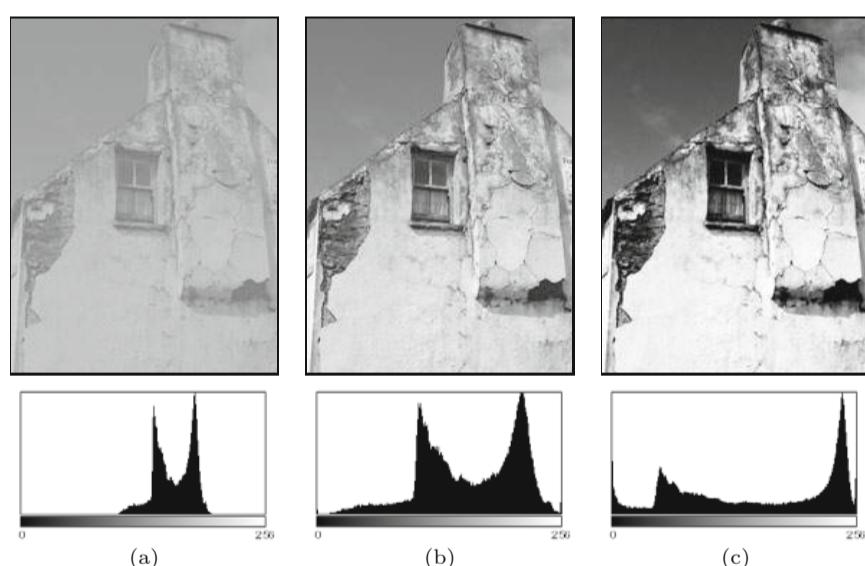
Histograms are useful in understanding image contrast, image defects, dynamic range and saturation.

Given a grayscale image, its histogram consists of the histogram of its gray levels; that is, a graph indicating the number of times each gray level occurs in the image.

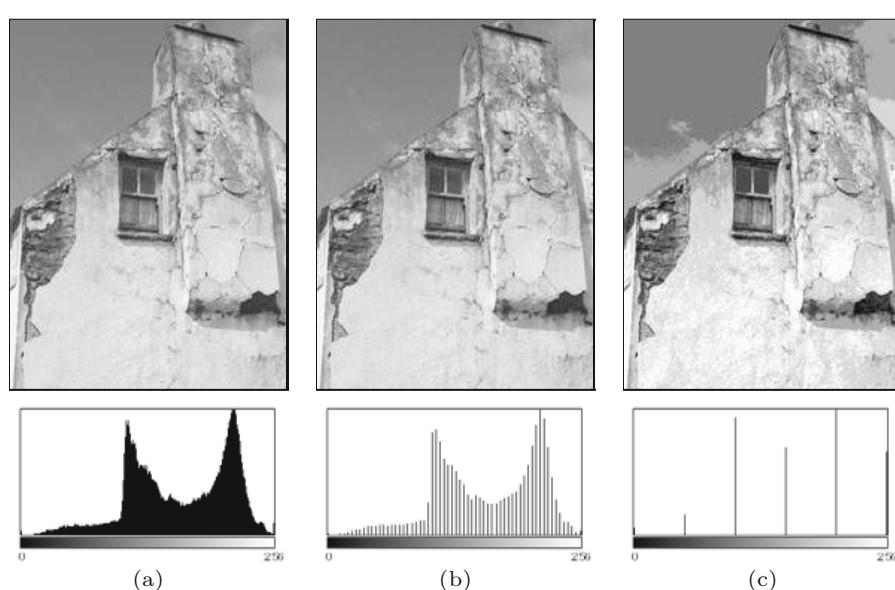
We can infer a great deal about the appearance of an image from its histogram.

- In a dark image, the gray levels would be clustered at the lower end
- In a uniformly bright image, the gray levels would be clustered at the upper end.
- In a well contrasted image, the gray levels would be well spread out over much of the range.

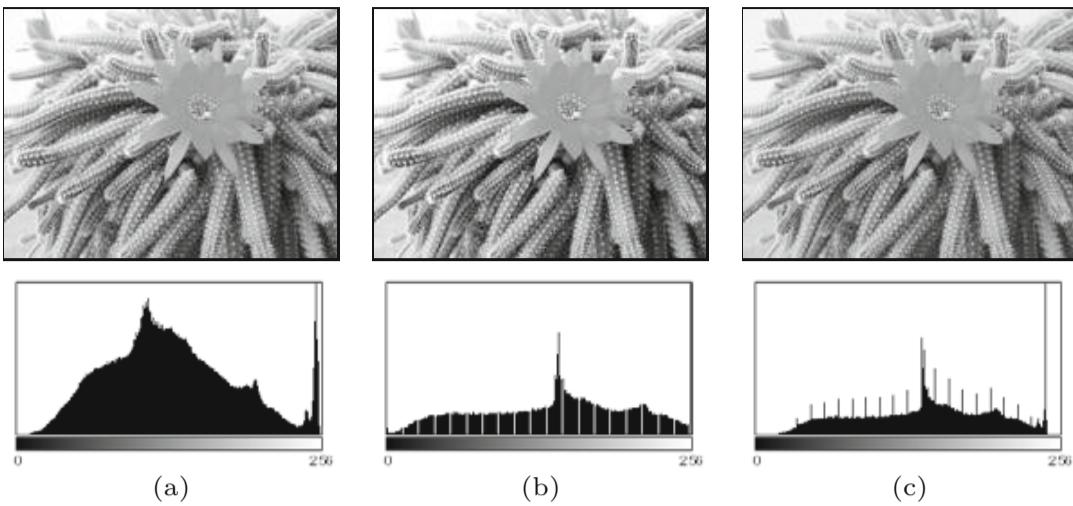
What is the dynamic range of an image? (from previous lecture)



How changes in contrast affect the histogram: low contrast (a), normal contrast (b), high contrast (c).



How changes in dynamic range affect the histogram: high dynamic range (a), low dynamic range with 64 intensity values (b), extremely low dynamic range with only 6 intensity values (c).



Effect of image capture errors on histograms: saturation of high intensities (a), histogram gaps caused by a slight increase in contrast (b), and histogram spikes resulting from a reduction in contrast (c).

Normally histograms are computed in order to visualize the image's distribution.

images with $2^8 = 256$ entries can visualised with values from 1 to 255.

but when an image uses a larger range of values, for instance 16- and 32-bit then the growing number of necessary histogram entries makes this no longer practical. This can be circumvented by 'binning'.

let a given entry in the histogram represent a range of intensity values. In a binned histogram of size B, each bin $h(j)$ contains the number of image elements having values within the interval $[a_j, a_{j+1}]$.

Sum, Mean, Variance, SNR of an image

The mean value μ of an image I (of size $M \times N$) can be calculated as

$$\mu = \frac{1}{MN} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} I(u, v)$$

the variance of the pixel values (measure of image contrast):

$$\sigma^2 = \frac{1}{MN} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [I(u, v) - \mu]^2$$

These are the mean intensity and variance of all the pixels within an image. You can also compute mean within a part of the image by selecting a specific region within the image.

The signal-to-noise ratio (SNR) is a common measure for quantifying the loss of image quality defined as the ratio between the average signal energy I_{signal} and the average noise energy I_{noise} .

Histogram Stretching (Contrast Stretching) or Contrast Adjustment

Lets, say you have a poorly contrasted image of range $[a,b]$, where a is minimum and b is the maximum value found in the current image, whose full intensity range is $[c,d]$. We can stretch the gray levels in the center of the range out by applying a piecewise linear function and stretch the image to the full intensity range. We first map the smallest pixel value a to zero, subsequently increase the contrast by the factor $(c-d)/(b-a)$, and finally shift to the target range by adding c .

This function has the effect of stretching the gray levels $[a,b]$ to gray levels $[c,d]$, where $a < c$ and $d > b$ according to the equation:

$$j = \frac{(c-d)}{(b-a)} \cdot (i-a) + c$$

Pixel values less than c are all converted to c , and pixel values greater than d are all converted to d .

Spatial operations

Spatial operations are performed directly on the pixels of an image. We classify spatial operations into three broad categories: (1) single-pixel operations, (2) neighborhood operations, and (3) geometric spatial transformations.

Single pixel or point operations

Point operations perform a modification of the pixel values without changing the size, geometry, or local structure of the image. Each new pixel value $b = I'(u, v)$

depends exclusively on the previous value $a = I(u,v)$ at the same position and is thus independent from any other pixel value, in particular from any of its neighbouring pixels.

$$b = f(I(u,v)) \quad \text{or} \quad b = f(a).$$

The arithmetic operators that uniformly add, subtract, multiply or divide an image with a constant value are also such operators.

Homogeneous point operations

If the function $f()$ is independent of the image coordinates (i.e., the same throughout the image), the operation is called “global” or “homogeneous”.

- modifying image brightness or contrast,
 - applying arbitrary intensity transformations,
 - quantizing (or “posterizing”) images,
 - global thresholding,

For instance, increasing the image contrast by 50% (i.e., by the factor 1.5) or raising the brightness by 10 units can be expressed by the mapping functions;

$$f_{\text{contr}}(a) = a \cdot 1.5 \quad \text{or} \quad f_{\text{bright}}(a) = a + 10 ,$$

When implementing arithmetic operations on pixel values, we must keep in mind that the calculated results must not exceed the admissible range of pixel values for the given image type (e.g., [0, 255] in the case of 8-bit grayscale images).

Threshold Operation

Thresholding an image is a special type of quantization that separates the pixel values in two classes, depending upon a given threshold value ‘q’ that is usually constant. The threshold operation maps all pixels to one of two fixed intensity values a_0 or a_1 ,

$$f_{\text{threshold}}(a) = \begin{cases} a_0 & \text{for } a < q, \\ a_1 & \text{for } a \geq q, \end{cases}$$

with $0 < q \leq a_{\max}$. A common application is binarizing an intensity image with the values $a_0 = 0$ and $a_1 = 1$.

Non-homogeneous point operation

the mapping function $g()$ for a non-homogeneous point operation would also take into account the current image coordinate (x,y) , that is,

$$b = g(I(x,y), x, y)$$

$I(x,y)$ are the pixel values at coordinates (x,y) before the operation.

For instance, Neighborhood Operations :

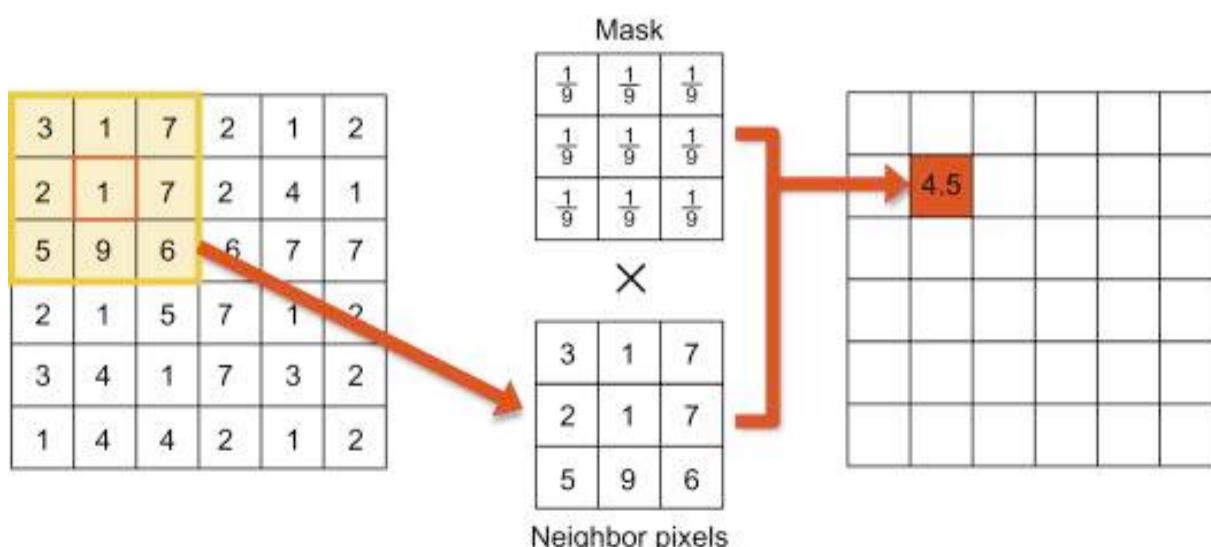
Assume S_{xy} as a set of neighborhood pixels around a center (x,y) in image f .

Neighborhood operations generate an intensity value for the center pixel (x,y) such that its value is determined by a specified operation on the neighborhood of the pixels in the input image.

$$g(x, y) = \frac{1}{mn} \sum_{(r, c) \in S_{xy}} f(r, c)$$

For example, if we want to perform the neighborhood average:

Neighborhood processing generates a corresponding pixel at the same coordinates in an output (processed) image, g , such that the value of that pixel is determined by a specified operation on the neighborhood of pixels in the input image with coordinates in the set S_{xy} . For example, suppose that the specified operation is to compute the average value of the pixels in a rectangular neighborhood of size $m \times n$ centered on (x, y) . Window size in this example is 3×3 .



Eg:

Here is an example showing local averaging using neighbourhood operation, the output image has a blur effect.

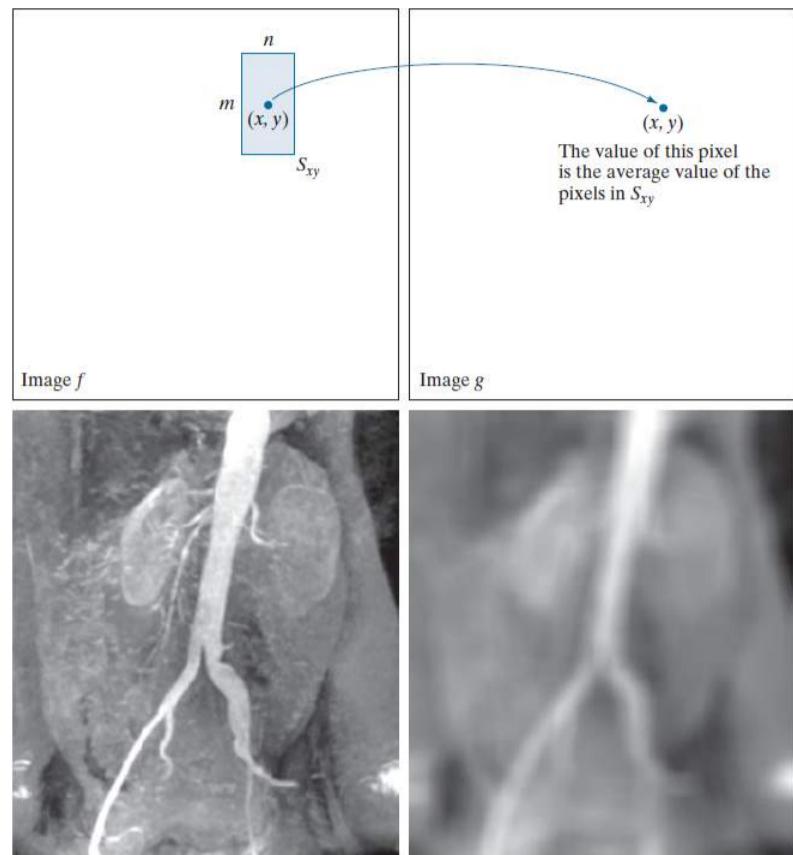


Image Interpolation

Interpolation is used in tasks such as zooming, shrinking, rotating, and geometrically correcting digital images.

Here we will see interpolation for image resizing (shrinking and zooming), which are basically image resampling methods. Uses of interpolation in applications such as rotation and geometric corrections will be done using geometric transformations.

Interpolation is the process of using known data to estimate values at unknown locations.

Image resizing uses nearest neighbour to interpolate pixel values. For example,

Resize to a larger image (double) than the original:

We look for a closest pixel in the underlying original image and assign the intensity of that pixel to the new pixel (also called nearest neighbor interpolation because it assigns to each new location the intensity of its nearest neighbor in the original image)

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	...
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	...
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	...
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	...
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	...
x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	...
:	:	:	:	:	:	...

x_{22}	x_{22}	x_{24}	x_{24}	x_{26}	x_{26}	...
x_{22}	x_{22}	x_{24}	x_{24}	x_{26}	x_{26}	...
x_{42}	x_{42}	x_{44}	x_{44}	x_{46}	x_{46}	...
x_{42}	x_{42}	x_{44}	x_{44}	x_{46}	x_{46}	...
x_{62}	x_{62}	x_{64}	x_{64}	x_{66}	x_{66}	...
x_{62}	x_{62}	x_{64}	x_{64}	x_{66}	x_{66}	...
:	:	:	:	:	:	...

This approach can produce undesirable artefacts, such as severe distortion of straight edges.

Bilinear interpolation

A more suitable approach is bilinear interpolation, in which we use the four nearest neighbors to estimate the intensity at a given location. Let (x, y) denote the coordinates of the location to which we want to assign an intensity value, and let $v(x, y)$ denote that intensity value. For bilinear interpolation, the assigned value is obtained using the equation

$$v(x_n, y_n) = ax_n + by_n + cx_ny_n + d$$

$n=[i,j]$ are the image coordinates. The four coefficients $[a,b,c,d]$ are determined from the four equations in four unknowns that can be written using the four nearest neighbors of point (x,y) .

Bicubic interpolation

The next level of complexity is bicubic interpolation, which involves the sixteen nearest neighbors of a point. The intensity value assigned to point (x, y) is obtained using the equation

$$v(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

The sixteen coefficients are determined from the sixteen equations with sixteen unknowns that can be written using the sixteen nearest neighbors of point (x, y) .

Geometric Spatial Transformations

Geometric transformations modify the spatial arrangement of pixels in an image. Geometric operations of digital images consists of two basic operations:

- Spatial transformation of coordinates;

- Intensity interpolation for assigning intensity values to spatially transformed pixels.

The transformation of coordinates may be expressed as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

after transformation : $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_{11}x + t_{12}y \\ t_{21}x + t_{22}y \end{bmatrix}$

where (x,y) are pixel coordinates in the original image and (x',y') are the corresponding pixel coordinates of the transformed image. For example, the transformation $(x',y') = (x/2, y/2)$ shrinks the original image to half its size in both spatial directions.

Affine Transformation provides scaling, shearing (non-rigid) and translation, rotation (rigid).

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine transformations preserve points, straight lines, and planes.

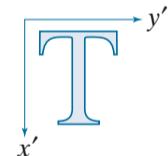
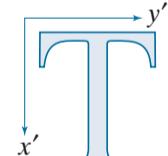
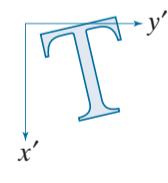
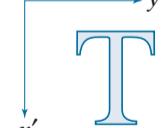
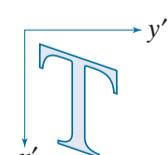
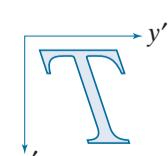
Transformation Name	Affine Matrix, \mathbf{A}	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $y' = y$	
Scaling/Reflection (For reflection, set one scaling factor to -1 and the other to 0)	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = c_x x$ $y' = c_y y$	
Rotation (about the origin)	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x \cos \theta - y \sin \theta$ $y' = x \sin \theta + y \cos \theta$	
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + t_x$ $y' = y + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & s_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + s_v y$ $y' = y$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ s_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $y' = s_h x + y$	

Image Registration

In all the above cases, the transformation is known. These can find direct applications when you want to introduce a rotation/translation of images.

However, there are applications where ‘transform’ has to geometrically produce an output image that is somehow aligned with the input image and the exact transformation is not known.

Image registration is an important application of digital image processing used to align two or more images of the same scene.

Typically, you are given a reference image and an input image. The objective is to transform the input image geometrically to produce an output image that is aligned (registered) with the reference image.

Typically, the exact transformation functions are not known in these cases. However, having rough estimate as a starting point helps to realise which transformation will be most helpful. We need to estimate the geometric transformation between the reference image and the output image.

There are several applications of image registration:

- aligning two or more images taken at approximately the same time, but using different imaging systems such as MRI (magnetic resonance imaging) scanner and a PET (positron emission tomography) scanner.
- images were taken at different times using the same instruments, such as satellite images of a given location taken several days, months, or even years apart.
- Images of growing live organism taken on a selective plane illumination microscopy (SPIM) using different view that needs to be re-constructed.
- In live cell experiments, if certain intracellular components appear in different spatial location and over time, and in addition there is cell motion, in such cases image alignments are helpful

Combining images or performing quantitative analysis and comparisons between them requires compensating for geometric distortions caused by differences in viewing angle, distance, orientation, sensor resolution, shifts in object location, and other factors.

How do you bring two images in to spatial correspondence? i.e. to map pixel-by-pixel to the same geometrical position.

1. Similarity measures: using either extrinsic information or intrinsic image based information such as correlation, mutual information, sum of differences
2. Types of transformation: rigid, affine, non-rigid, rotations
3. Interpolation techniques: nearest neighbour, linear, cubic?

1. Mapping the spatial coordinates of the pixels in the input image with reference image

i. Extrinsic information

- A principal approach for solving the problem is to use tie points (also called control points). These are corresponding points whose locations are known precisely in the input and reference images (i.e. their spatial coordinates in both the images).

- Points selection can be done interactively or using automatic methods.

ii. Intrinsic information

- These are image based method such as image cross-correlation, mutual information, sum of differences, joint histograms

▪ **Normalized cross-correlation**

$$NCC = \frac{\sum [I_1(x, y) - \bar{I}_1] [I_2(T(x, y)) - \bar{I}_2]}{\sqrt{\sum [I_1(x, y) - \bar{I}_1]^2 \cdot \sum [I_2(T(x, y)) - \bar{I}_2]^2}} \quad (\text{maximize})$$

▪ **Sum of squared differences**

$$SSD = \sum [I_1(x, y) - I_2(T(x, y))]^2 \quad (\text{minimize})$$

▪ **Mutual image information**

$$MI = E(I_1) + E(T(I_2)) - E(I_1, T(I_2)) \quad (\text{maximize})$$

$$\text{where } E(I) = \sum_{\substack{\uparrow \\ \text{Entropy}}} P(i) \log \left(\frac{1}{P(i)} \right) \quad \text{and} \quad E(I_1, I_2) = \sum_{\substack{\uparrow \\ \text{Probability}}} P(i, j) \log \left(\frac{1}{P(i, j)} \right)$$

2. Challenge of finding the right transformation

- i. If you use fiducial markers, then estimating the transformation function is one of modeling. For example, suppose that we have a set of four control points each in an input image (c_1, c_2, c_3, c_4) and a reference image (c_5, c_6, c_7, c_8). A simple model based on a bilinear approximation is given by

$$x = c_1v + c_2w + c_3vw + c_4$$

$$y = c_5v + c_6w + c_7vw + c_8$$

During the estimation phase, (v, w) and (x, y) are the coordinates of tie points in the input and reference images, respectively. If we have four pairs of corresponding tie points in both images, we can write eight equations using the above two equations and use them to solve for the eight unknown coefficients, c_1 through c_8 . With the coefficients, we let (v, w) denote the coordinates of each pixel in the input image, and (x, y) become the corresponding coordinates of the output image. We compute all coordinates (x, y) ; we just step through all (v, w) in the input image to generate the corresponding (x, y) in the output, registered image. If the tie points were selected correctly, this new image should be registered with the reference image, within the accuracy of the bilinear approximation model. You can make the algorithm more complex and hence more accurate by selecting larger number of tie points and then we need to employ more complex models, such as polynomials fitted by least squares algorithms.

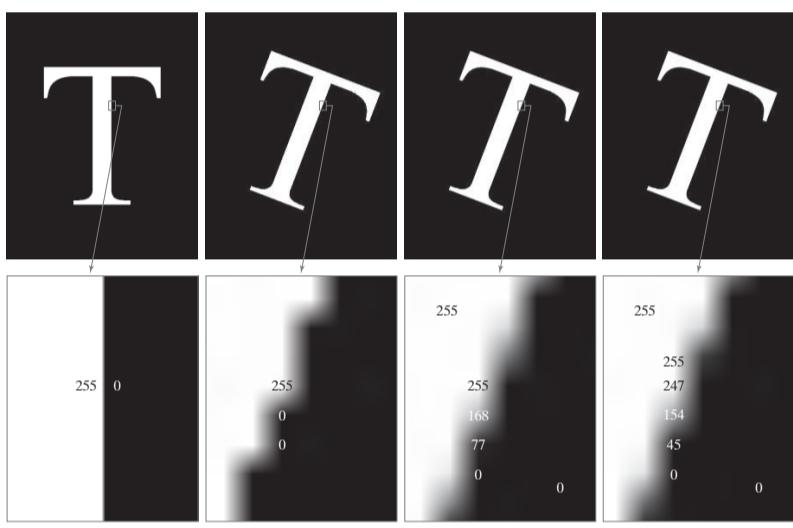
- ii. In the second case (using intrinsic information from the images itself), i.e. when the points are measured automatically based on similarity, you may need to solve it as optimisation problem between input image and reference image. In this case, having an idea of the type of transformation is helpful. For example, rigid transformations (like rotations, translations) or affine transformations that allow flexibility (like scaling, shearing) may be used.

Or you may combine the two approaches, first selecting control points and then computing spatial correlations between input and target image. Finally estimating the type of transformations (that uses both information; control points and spatial correlations based on inverse mapping).

Reading material: What is inverse mapping?

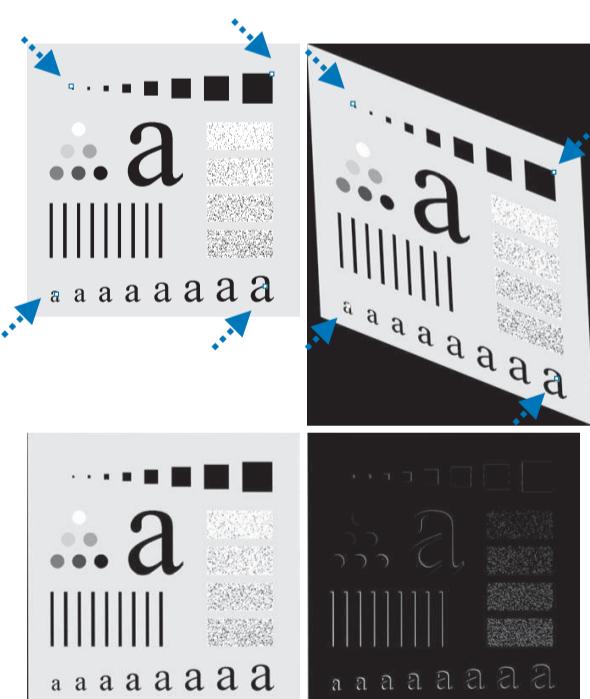
3. Finally, we still need to perform intensity interpolation using any of the methods discussed previously to assign intensity values to the transformed pixels.

Below is an example showing image rotation by -21° in all the cases, however the interpolation methods (for intensity interpolation) are different in each case. Notice how the pixel values are changed depending upon the interpolation technique. (We already discussed intensity interpolation previously in this class, please see above.)



(a) A 541×421 image of the letter T. (b) Image rotated -21° using nearest-neighbor interpolation for intensity assignments. (c) Image rotated -21° using bilinear interpolation. (d) Image rotated -21° using bicubic interpolation. (e)-(h) Zoomed sections (each square is one pixel, and the numbers shown are intensity values).

Example of image registration using control points.



(a) Reference image. (b) Input (geometrically distorted image). Corresponding four tie/control points are shown as small squares near the corners (see next to blue arrows). Distortion is linear shear in both directions
 (c) Registered (output) image (note the errors in the border).
 (d) Difference between (a) and (c), showing more registration errors.

The reason for the discrepancies is error in the manual selection of the tie points. It is difficult to achieve perfect matches for tie points when distortion is so severe.

Vectors and Matrix operations for multispectral image processing

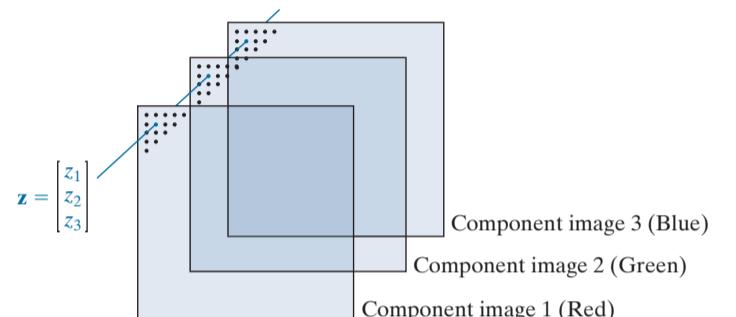
Multispectral image processing is a typical area in which vector and matrix operations are used routinely. For processing RGB color space by using red, green, and blue component images. We saw in Lecture 1 that the 8-bit RGB image has 3 channels (one each for red, blue and green channel). Hence, each pixel in an RGB image has three components, which can be organised in the form of a column vector where z_1 , z_2 and z_3 are the corresponding pixel intensities in the red, green and blue images, respectively. The same concept can be extended to multispectral images involving n component images resulting in n -dimensional vectors.

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

3-channel

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

multi-spectral



The inner product (also called the dot product) of two n -dimensional column vectors \mathbf{a} and \mathbf{b} is defined as

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &\triangleq \mathbf{a}^T \mathbf{b} \\ &= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \\ &= \sum_{i=1}^n a_i b_i \end{aligned}$$

where T indicates the transpose.

The Euclidean vector norm, denoted by z , is defined as the square root of the inner product:

$$\|\mathbf{z}\| = (\mathbf{z}^T \mathbf{z})^{\frac{1}{2}}$$

Then the generalisation of the 2-D Euclidean distance $D(z, a)$ between points (vectors) z and a in n -dimensional space is defined as the Euclidean vector norm:

$$D(\mathbf{z}, \mathbf{a}) = \|\mathbf{z} - \mathbf{a}\| = \left[(\mathbf{z} - \mathbf{a})^T (\mathbf{z} - \mathbf{a}) \right]^{\frac{1}{2}} = \left[(z_1 - a_1)^2 + (z_2 - a_2)^2 + \cdots + (z_n - a_n)^2 \right]^{\frac{1}{2}}$$

Linear transformations are applicable on pixel vectors:

$$\mathbf{w} = \mathbf{A}(\mathbf{z} - \mathbf{a})$$

$$\mathbf{g} = \mathbf{Hf} + \mathbf{n}$$

Here A is a matrix of size $m \times n$, and z and a are column vectors of size $n \times 1$ and the output w is of size $m \times n$.

In the second example, f is input image of $MN \times 1$ vector, n is the noise pattern of $MN \times 1$ vector, g is processed image of $MN \times 1$ vector and H is an $MN \times MN$ matrix representing a linear process applied to the input image.

Photoshop/ paint
brush are not image
processing tools!!

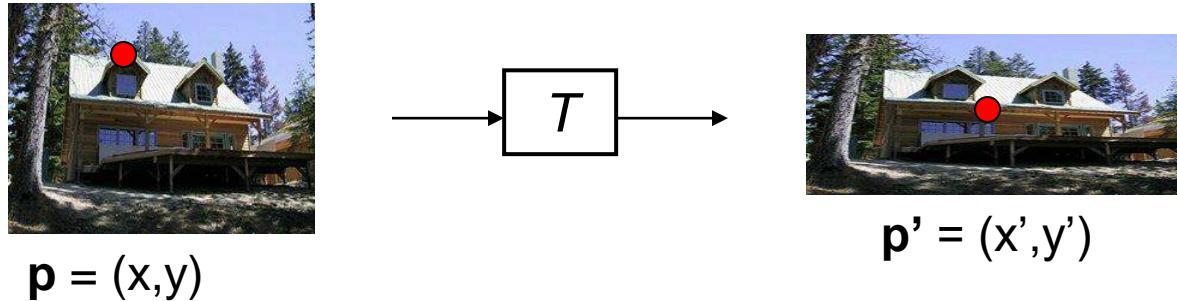


Geometry of image formation

DSE312 - LECTURE 4

BHAVNA R

Parametric (global) transformations



Transformation T is a coordinate-changing machine:

$$p' = T(p)$$

What does it mean that T is global?

- T is the same for any point p
- T can be described by just a few numbers (parameters)

For linear transformations, we can represent T as a matrix

$$p' = \mathbf{T}p$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common transformations



Original

Transformed



Translation



Rotation



Scaling

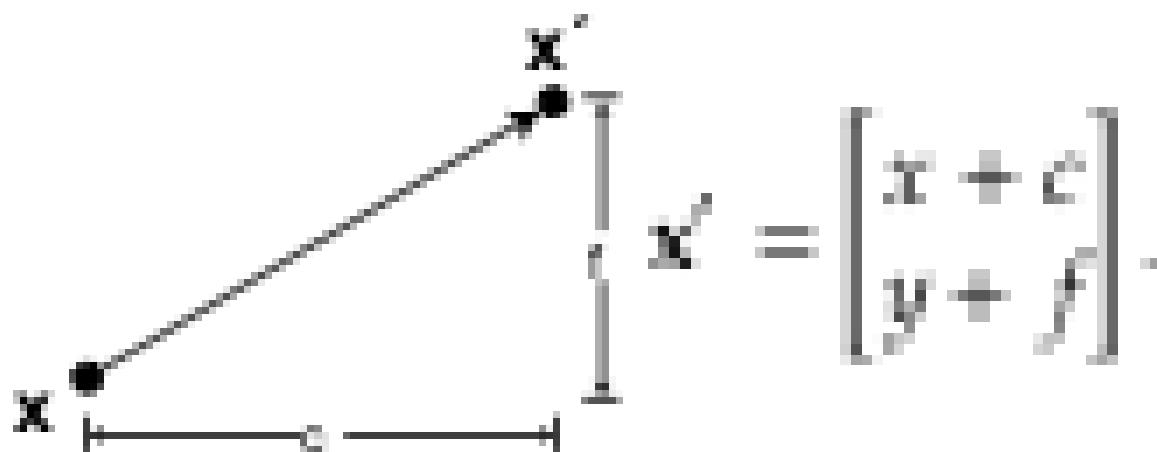


Affine

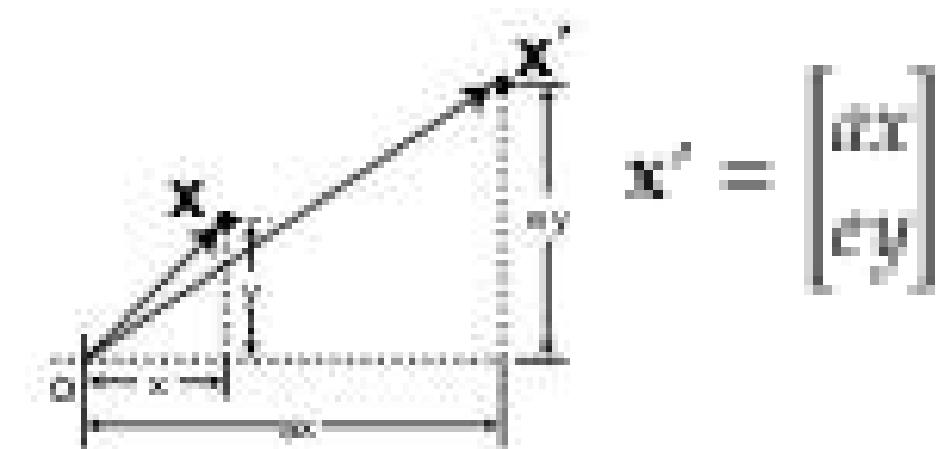


Perspective

Slide credit (next few slides):
A. Efros and/or S. Seitz



A pure translation



A pure scale

Uniform scaling means this scalar
is the same for all components

OR

Non-uniform scaling i.e. different
scalars per component

A diagram showing a point x at position (x, y) and a point x' at position $(x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$. An angle θ is shown between the x-axis and the vector from the origin to point x . The transformation is represented by the matrix equation:

$$x' = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$$

A pure rotation

Basic 2D transformations in matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \alpha_x \\ \alpha_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine

Affine is any combination of translation, scale, rotation, and shear

Matrix Representation of Linear Transformations

Affine transforms scale, rotate and shear are linear transforms and can be represented by a matrix multiplication of a point represented as a vector

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

or $\mathbf{x}' = M\mathbf{x}$, where M is the matrix

Matrix representations allow complex transform into a set of simpler transforms.
Let S be the scale matrix, H be the shear matrix and R be the rotation matrix.

$\mathbf{x}' = (RHS)\mathbf{x}$ or $M = RHS$

Scale: $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$, Rotate: $\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$, Shear: $\begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix}$.

However, the problem is that translation is not a linear transform.

Let us take an example:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_x \\ b_y \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$$

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{matrix} \text{scaling} & \text{original coordinates} & \text{translation} \end{matrix} \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \end{pmatrix} + \begin{pmatrix} 20 \\ 20 \end{pmatrix} = \begin{pmatrix} 22 \\ 22.5 \end{pmatrix} \text{ coordinates after transformation}$$

Lets say, we have pixel value in image at coordinate (4,5):

Scales the size of the image by 1/2

Shifts the scaled image by 20 pixels in x and y direction

A pixel located at coordinate (4,5), will have a new value $(a_x, a_y) = (22, 22.5)$

Oh, these are not integer values, however these are coordinates , and coordinates are always integer values!

So, what do we do?

We would perform interpolation to obtain integer coordinate values (please see lecture-2). Remember, we still need to perform intensity interpolation.

Where do we need geometric transformations?

- geometric transformations between images (lecture-2)

-We want to create a mapping between 2 images (within the image coordinate frame).

-For solving image registration problems

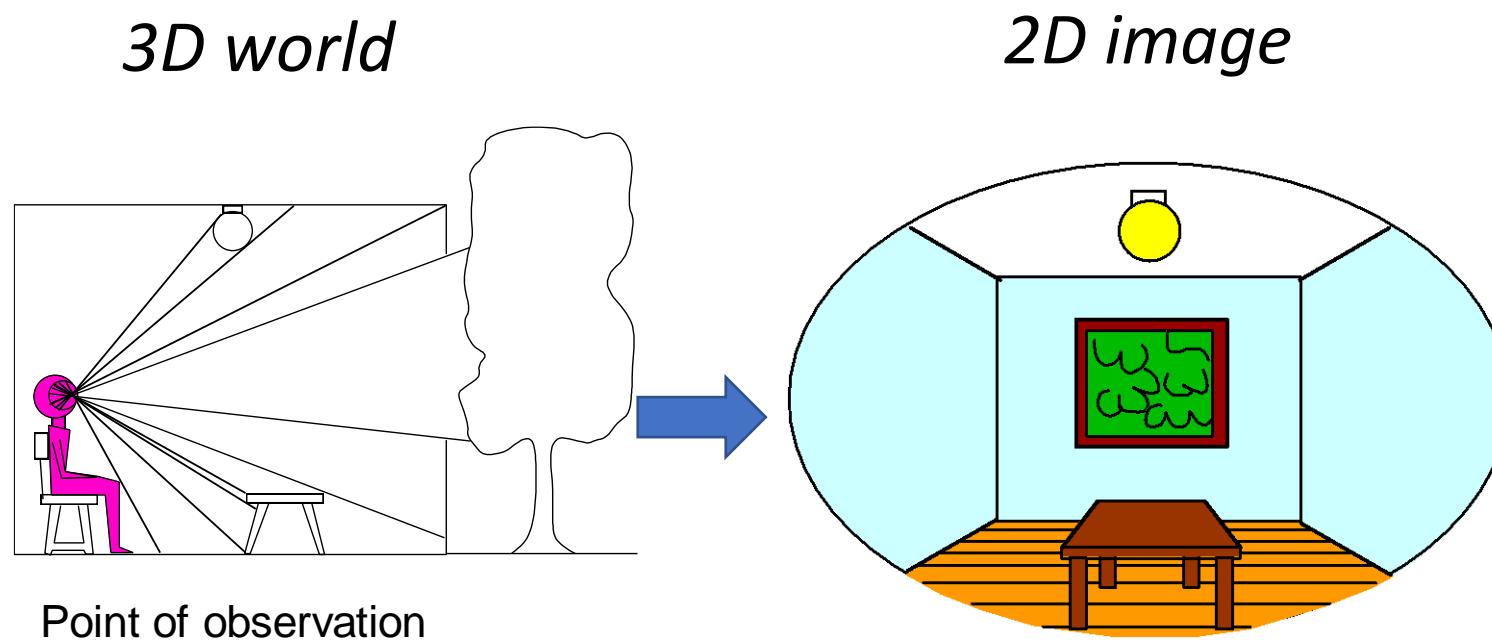
We do these through transformations in the Euclidean space.

- transformations between coordinate systems

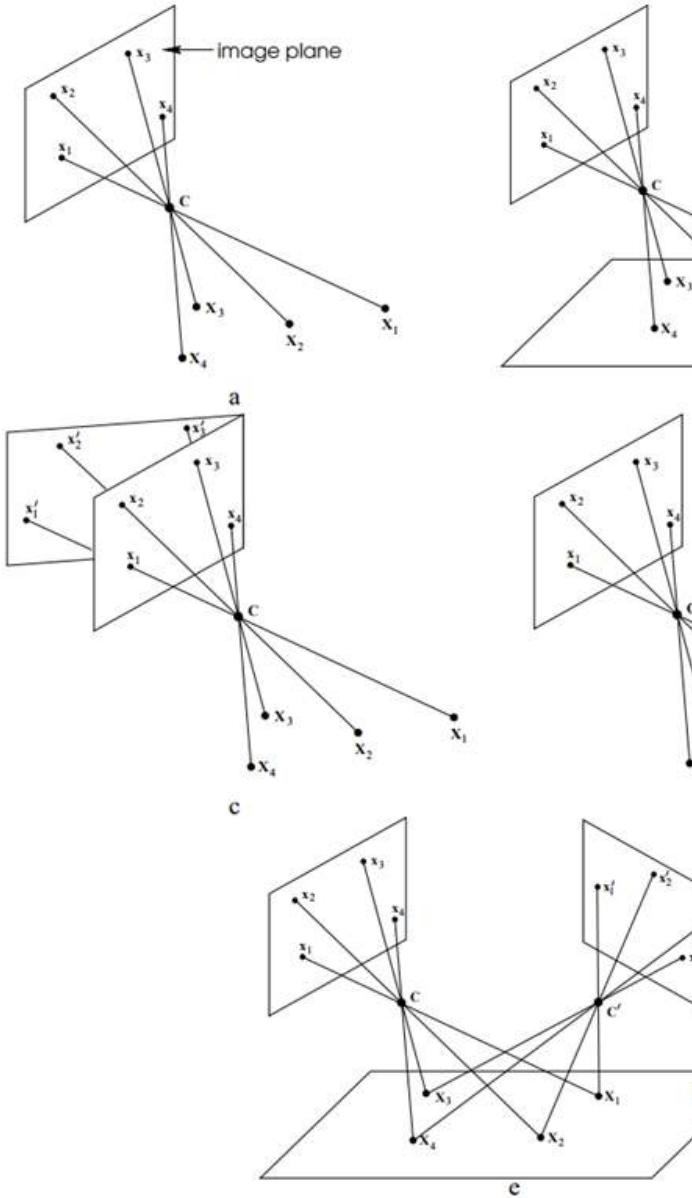
We need a mapping between image and world coordinates (we need another set of coordinate system which enable us to carry out the geometric transformations).

Camera projections

- the formation of a two-dimensional representation of a three-dimensional world, and what we may deduce about the 3D structure of what appears in the images.



The camera centre is the essence

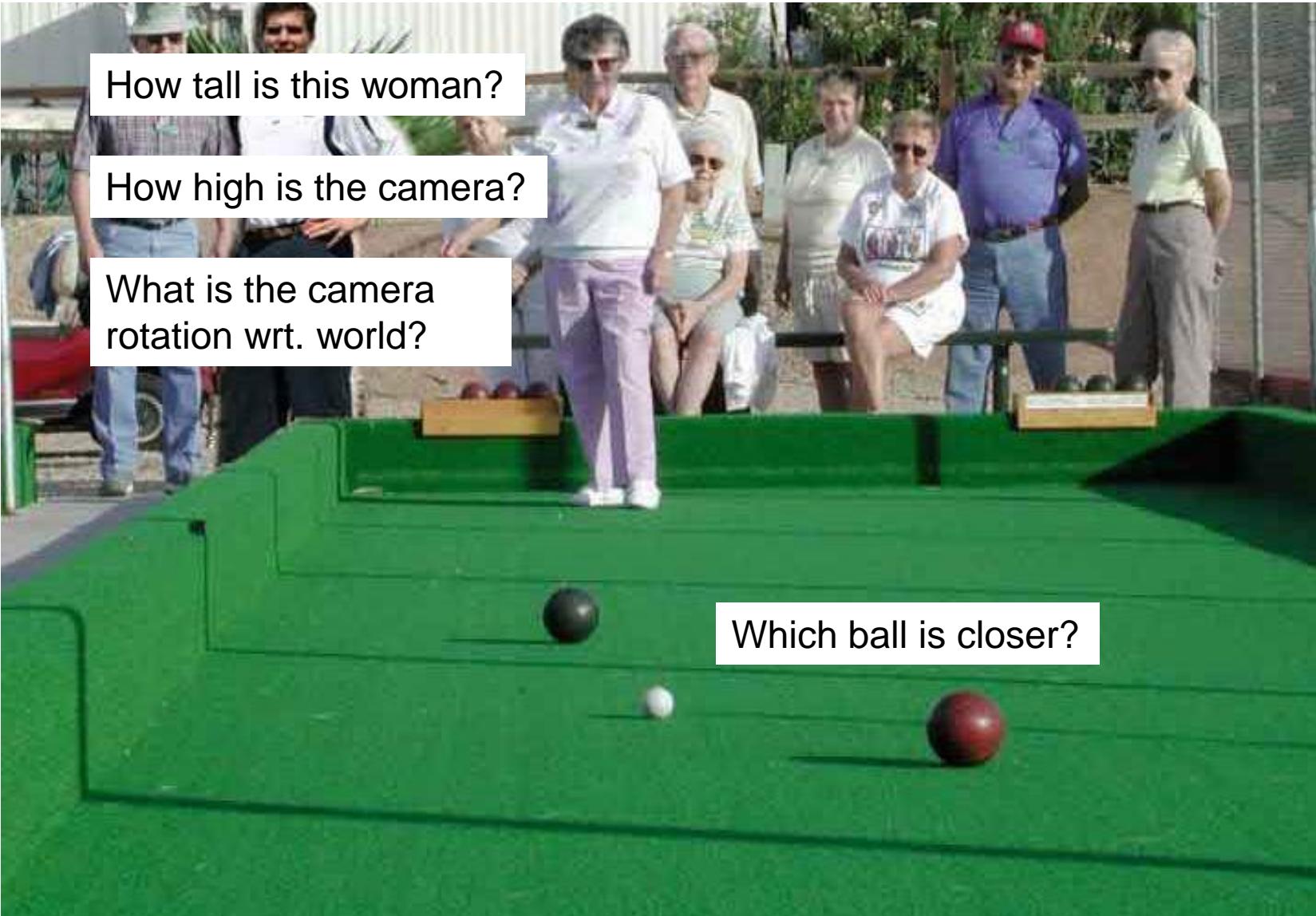


(a) Image formation: the image points x_i are the intersection of a plane with rays from the space points X_i through the camera centre C . (b) If the space points are coplanar then there is a projective transformation between the world and image planes, $x_i = H_{3 \times 3} X_i$. (c) All images with the same camera centre are related by a projective transformation. Compare (b) and (c) – in both cases planes are mapped to one another by rays through a centre. In (b) the mapping is between a scene and image plane, in (c) between two image planes. (d) If the camera centre moves, then the images are in general not related by a projective transformation, unless (e) all the space points are coplanar.

Camera projections

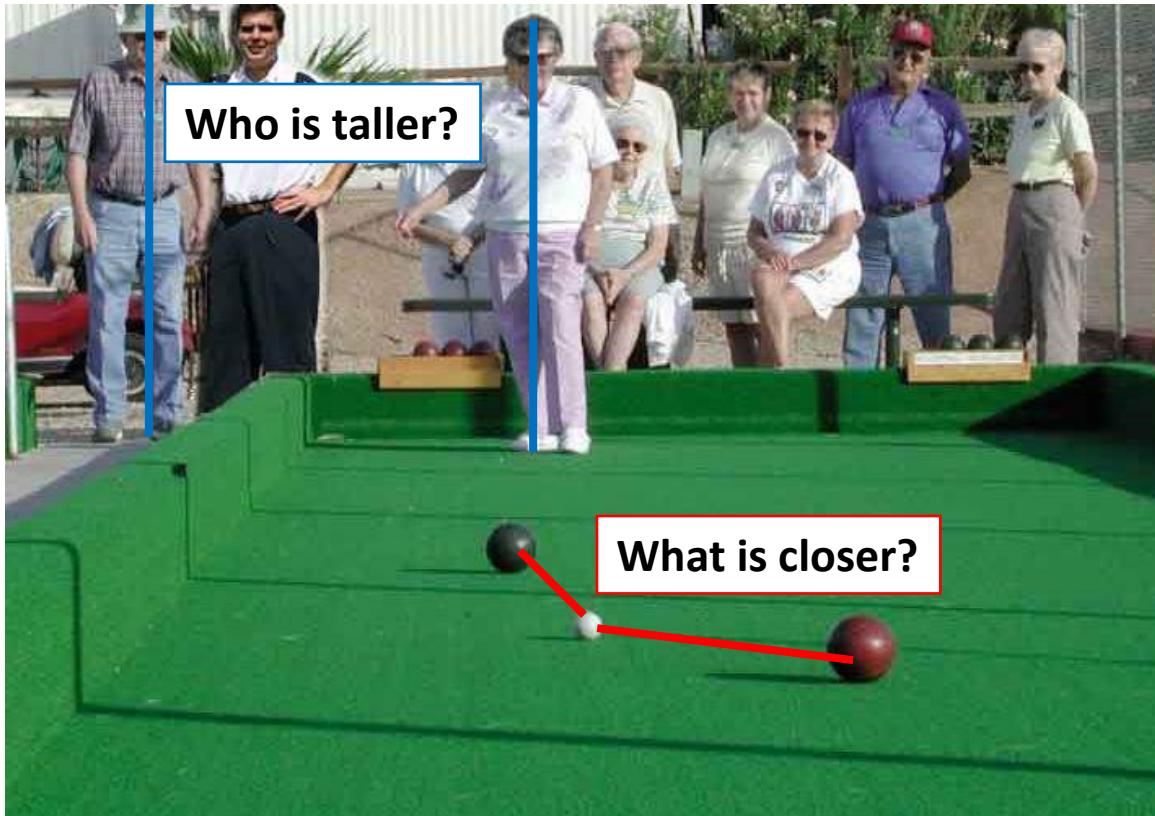
- The drop from three-dimensional world to a two-dimensional image is a projection process in which we lose one dimension.
- We use central projection in which a ray from a point in space is drawn from a 3D world point through a fixed point in space, the centre of projection.
- This ray will intersect a specific plane in space chosen as the image plane. The intersection of the ray with the image plane represents the image of the point.
- The method is in accord with a simple model of a camera, in which a ray of light from a point in the world passes through the lens of a camera and impinges on a film or digital device, producing an image of the point.

Cameras and World Geometry



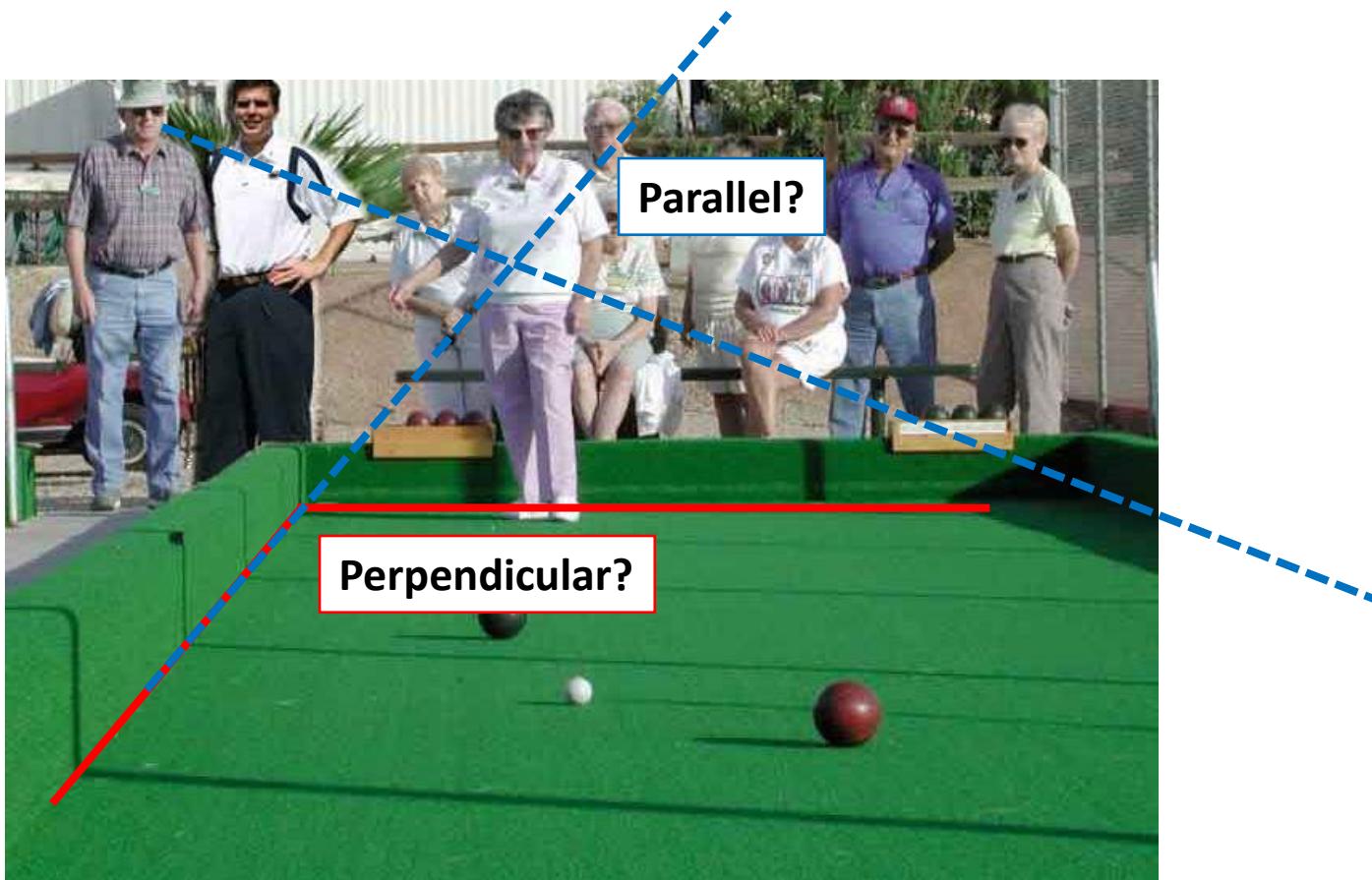
Projective Geometry

Length (and so area) is lost.



Projective Geometry

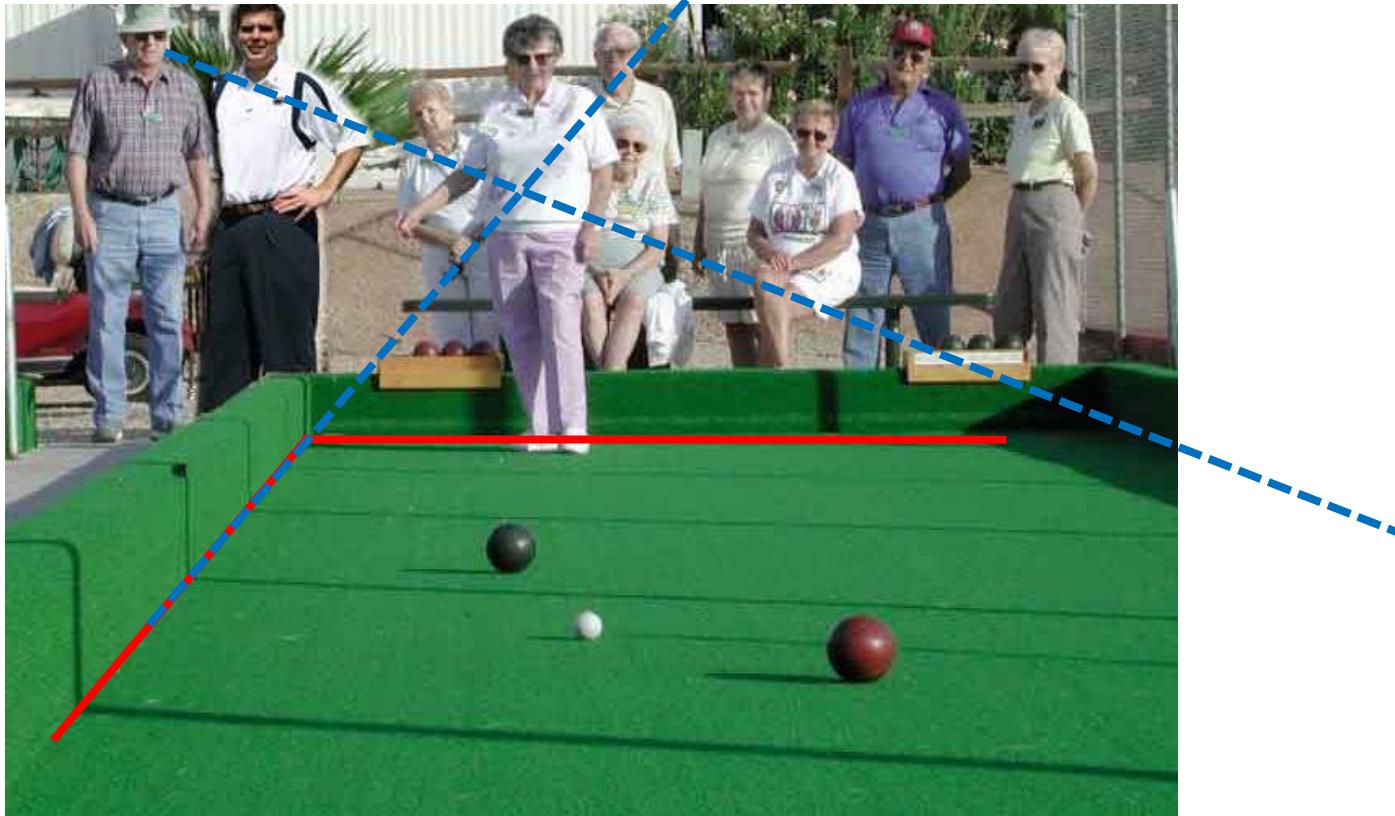
Angles are lost.



Projective Geometry

What is preserved?

- Straight lines are still straight.

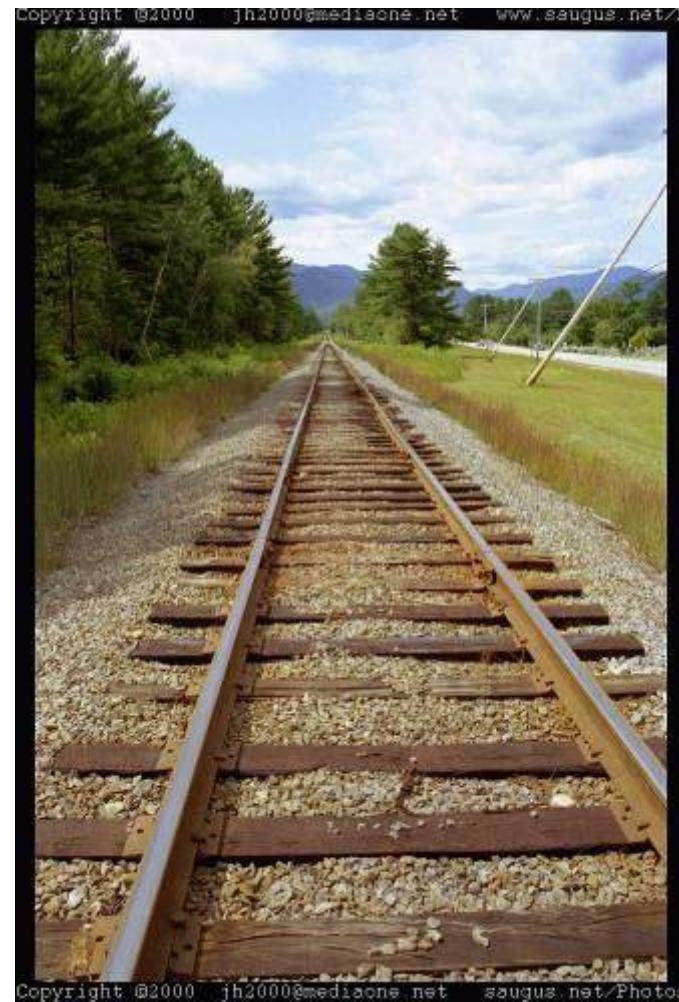
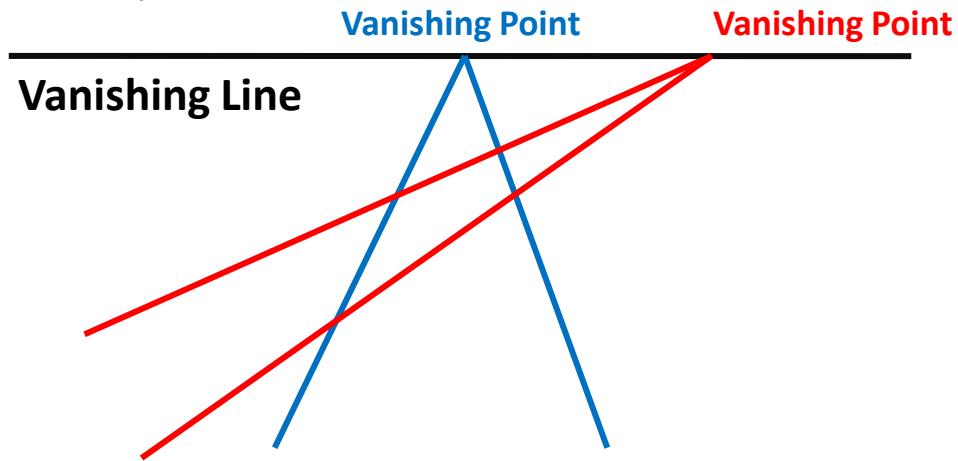


Vanishing points and lines

Parallel lines in the world
intersect in the projected
image at a “vanishing point”.

Parallel lines on the same
plane in the world converge
to vanishing points on a
“vanishing line”.

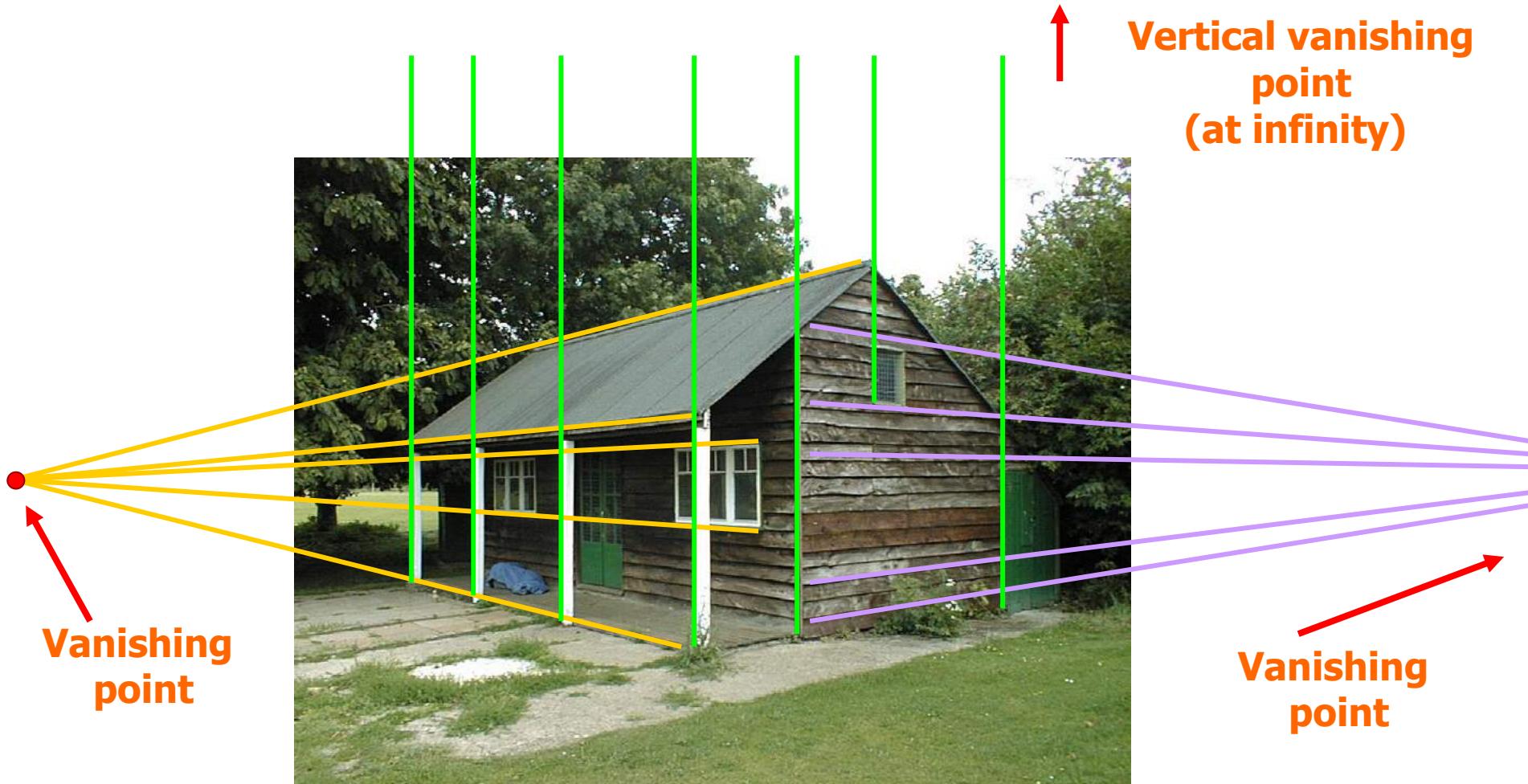
E.G., the horizon.



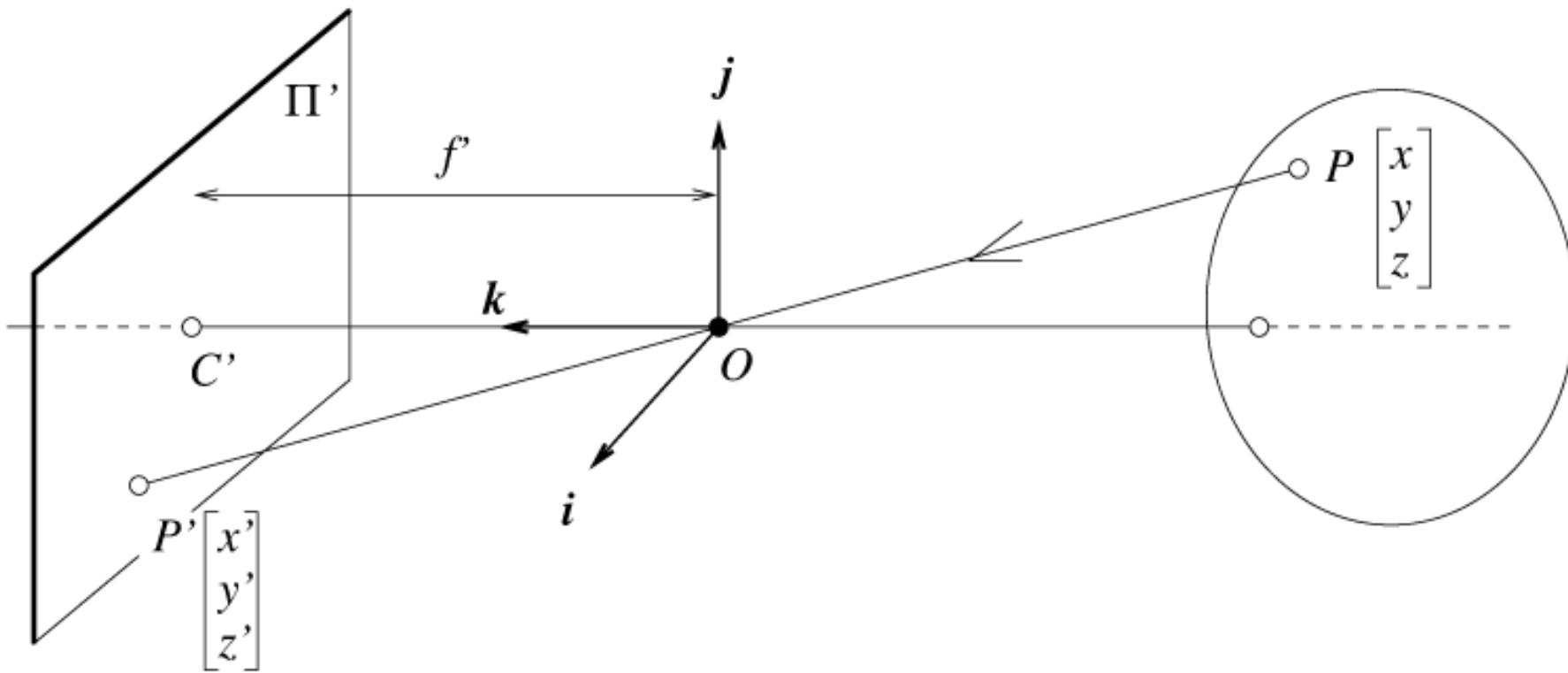
Vanishing points

- each set of parallel lines (=direction) meets at a different point
 - The *vanishing point* for this direction
- Sets of parallel lines on the same plane lead to *collinear* vanishing points.
 - The line is called the *horizon* for that plane
- Good ways to spot faked images
 - scale and perspective don't work
 - vanishing points behave badly
 - supermarket tabloids are a great source.

Vanishing points and lines

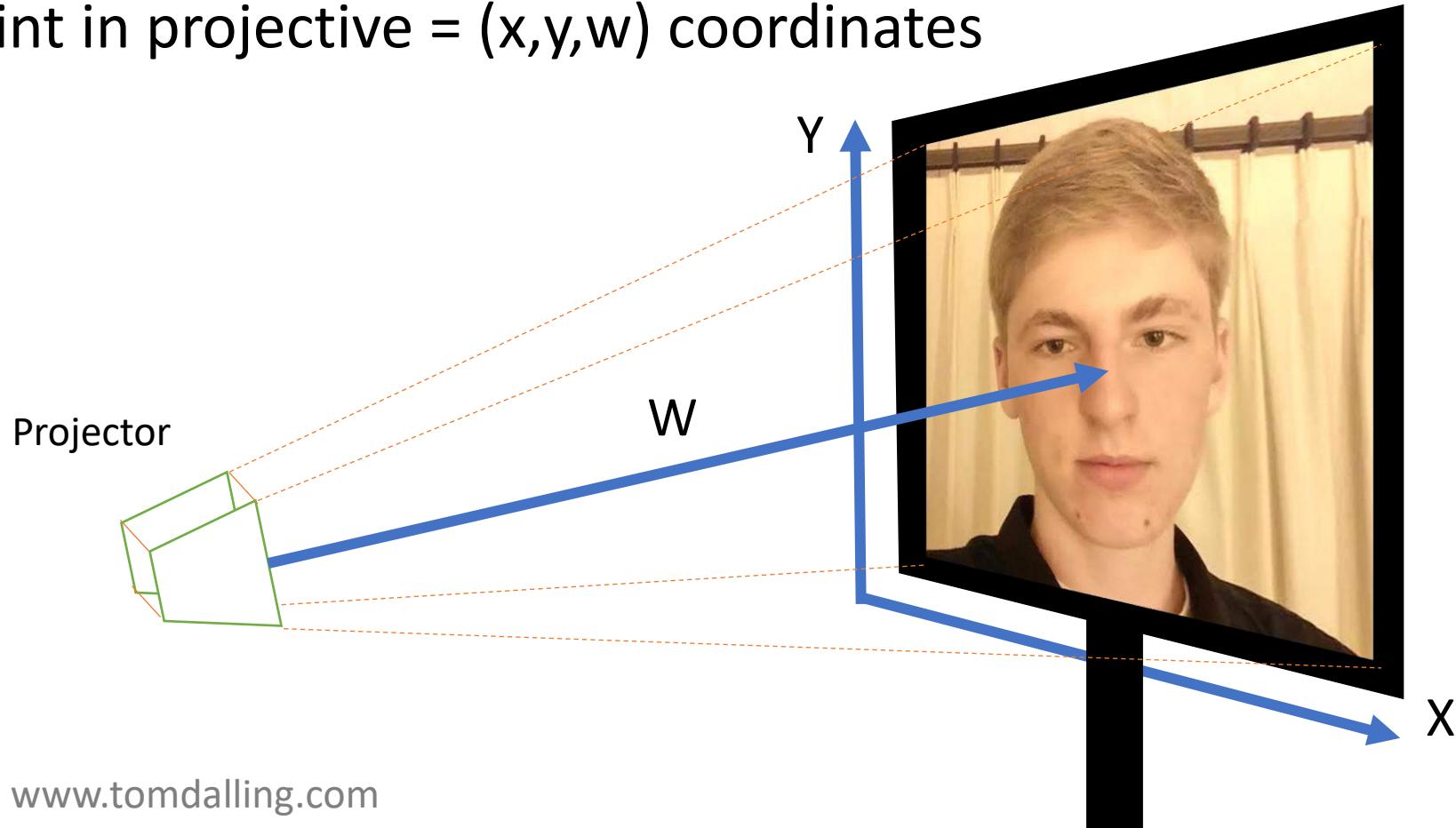


We want to derive the equation of projection



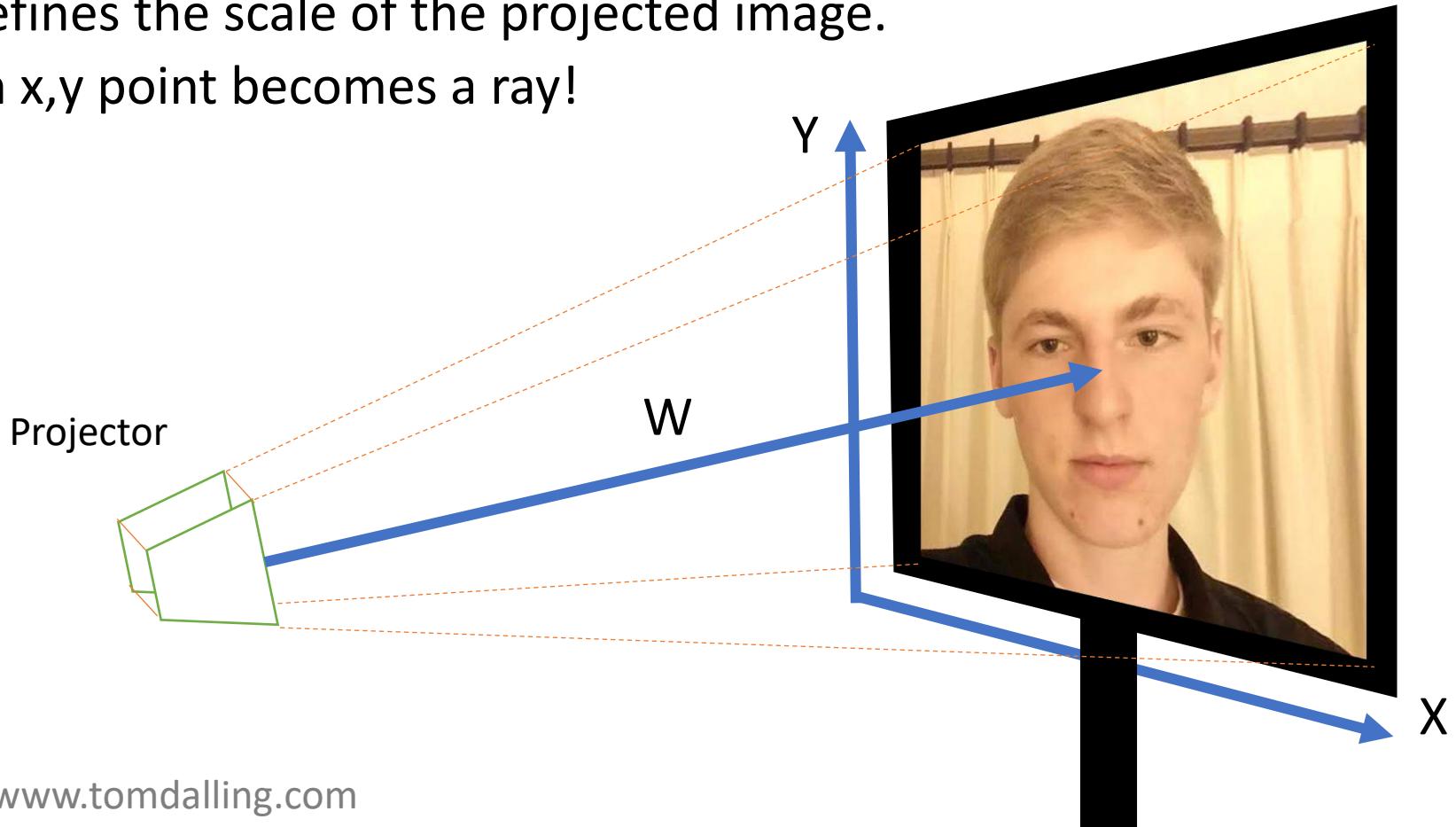
Projective geometry

- 2D point in cartesian = (x,y) coordinates
- 2D point in projective = (x,y,w) coordinates

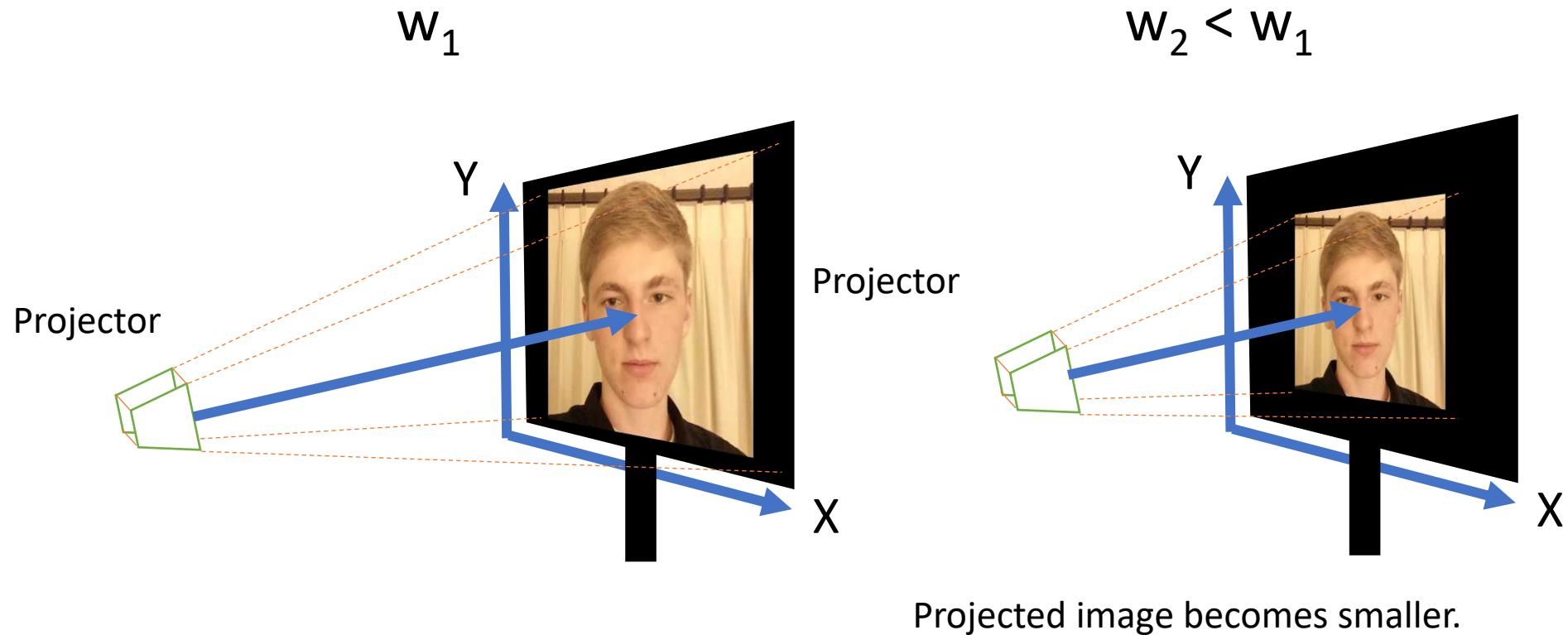


Projective geometry

- 2D point in projective = (x,y,w) coordinates
 - w defines the scale of the projected image.
 - Each x,y point becomes a ray!

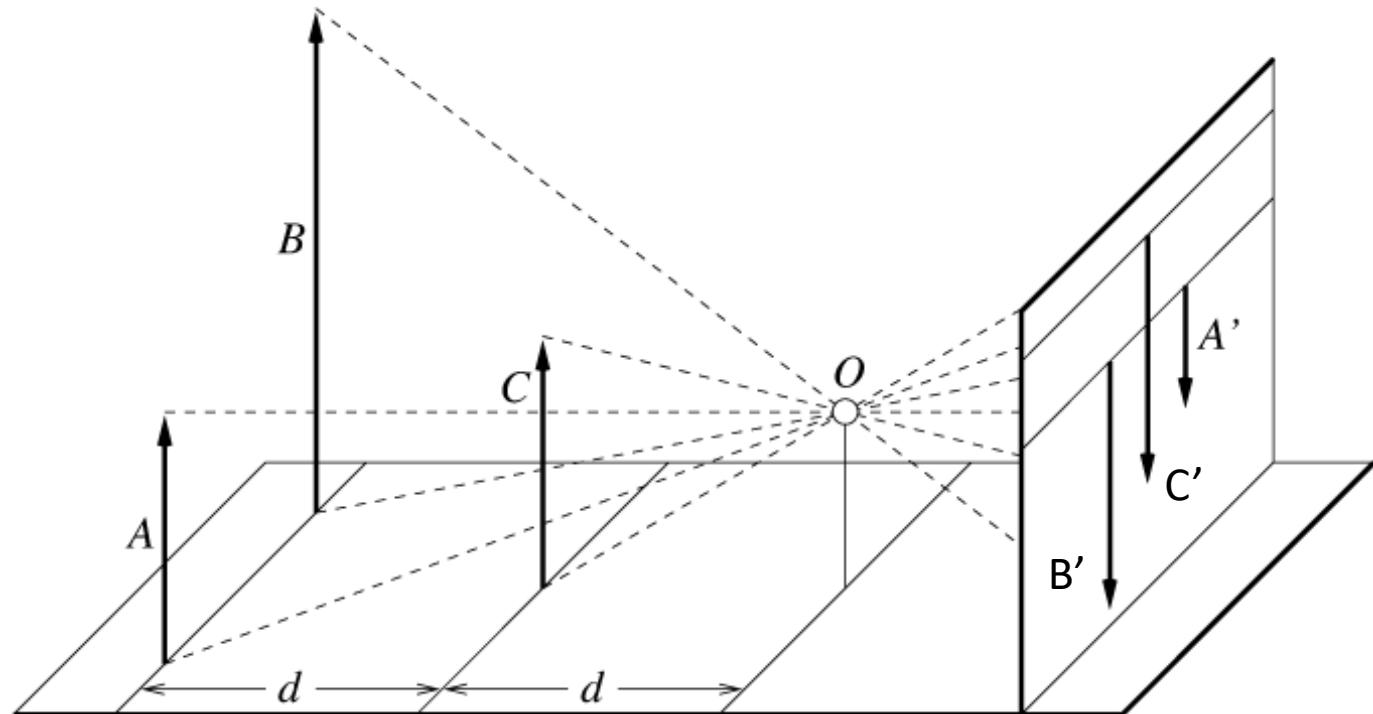


Varying w

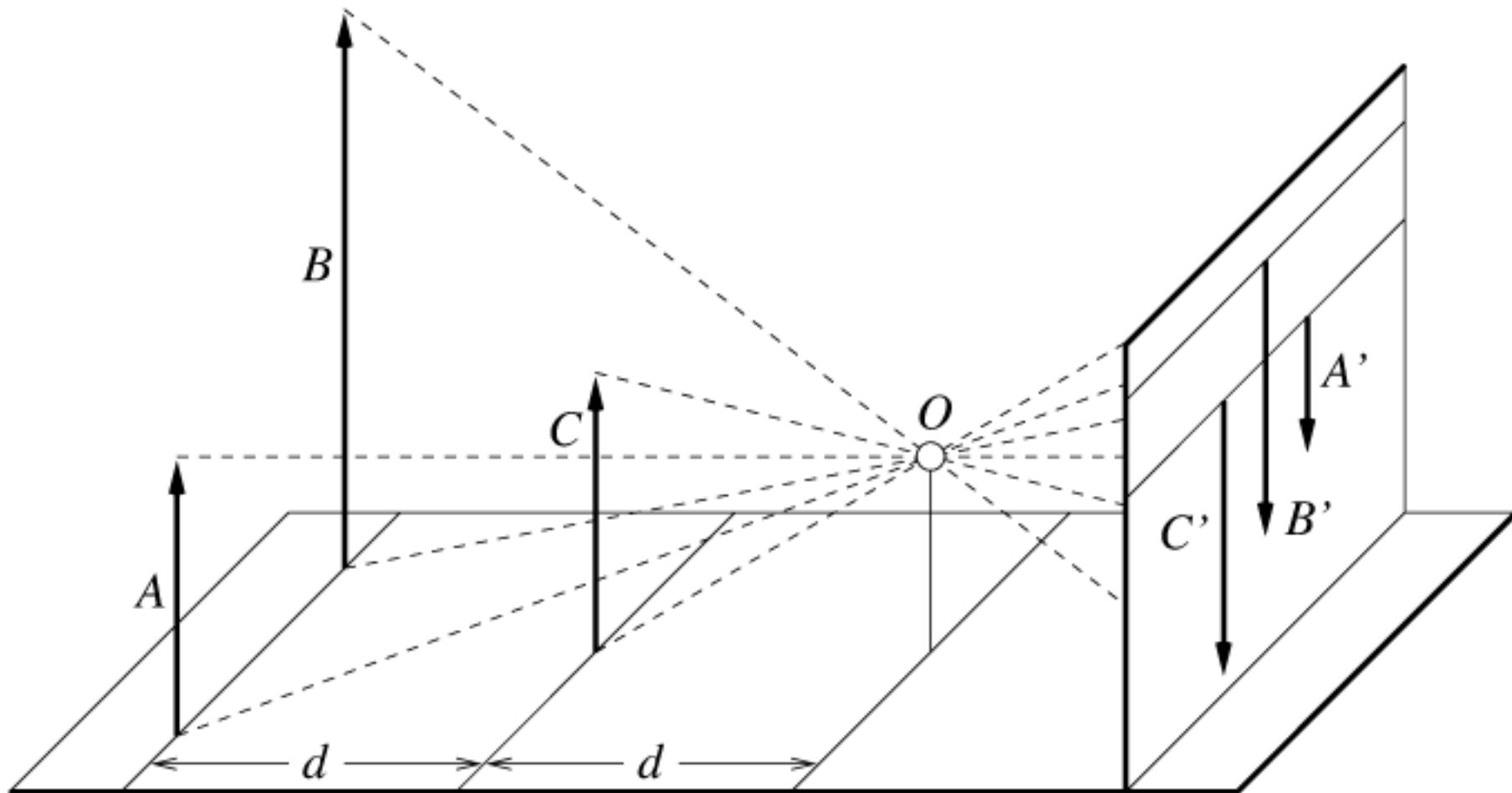


Projective geometry

- Length and area are not preserved
- Perspective projection of a line in 3D is a line in 2D
- In 3D, point (x,y,z) becomes (x,y,z,w)
- Perspective is w varying with z :
 - Objects far away appear smaller



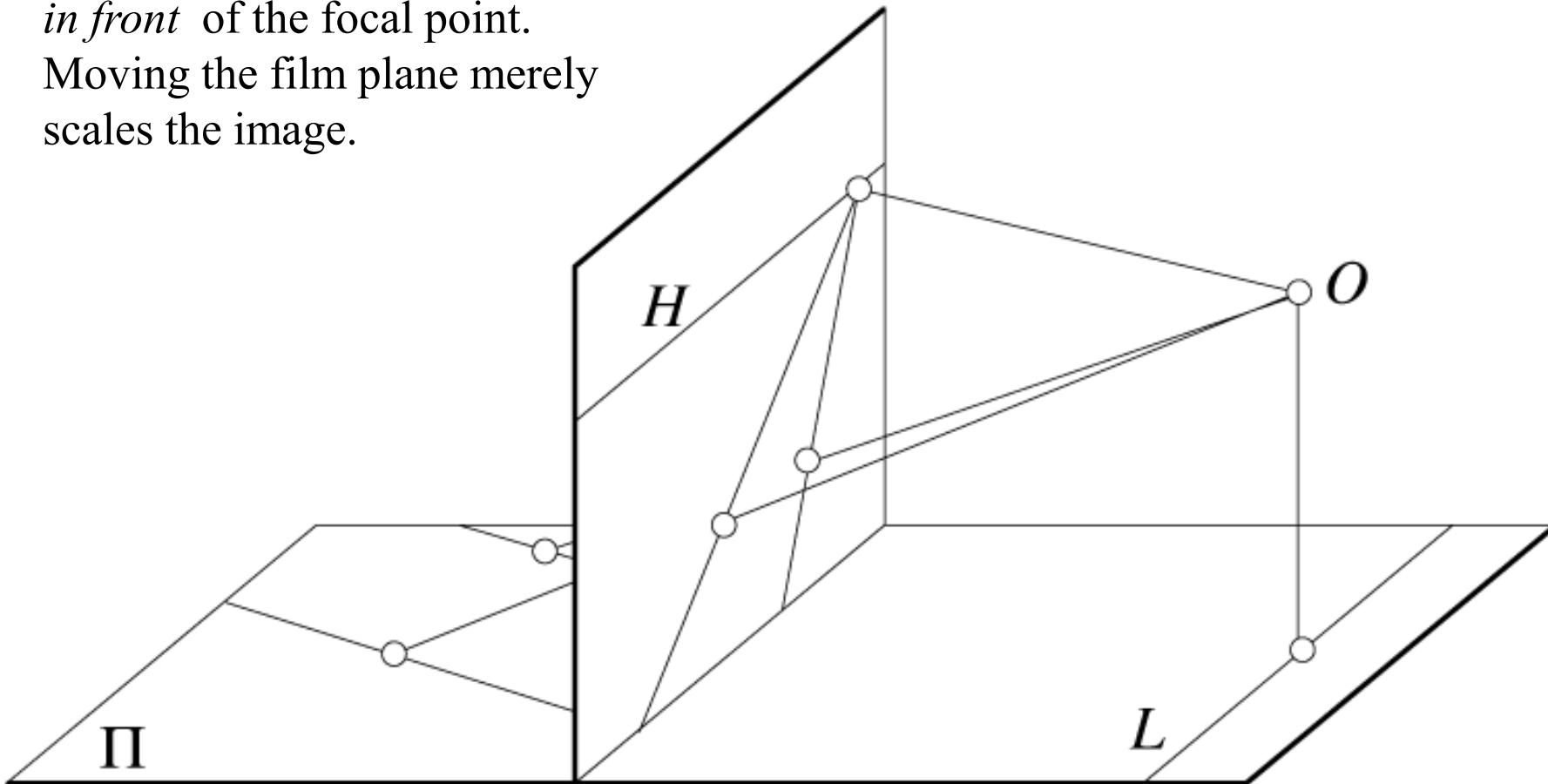
Distant objects are smaller



Parallel lines meet

Common to draw film plane
in front of the focal point.

Moving the film plane merely
scales the image.



Projective Transformations

Projective transformations are combos of

- Affine transformations, and
- Projective warps

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

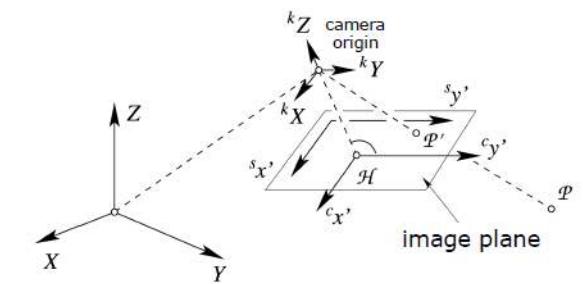
Properties of projective transformations:

- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved

Projective matrix is defined up to a scale (8 DOF)

reasons to use homogeneous coordinate system

- We need calculations of graphics and geometry possible in projective space
- homogeneous coordinates or projective coordinates is a system of coordinates used in projective geometry
- algebraically treats all points in the projective plane (both Euclidean and ideal) equally.
- In Euclidean space (geometry), two parallel lines on the same plane cannot intersect ever.
- In projective space,, the train railroad becomes narrower while it moves far away from eyes. The two parallel rails meet at the horizon, a point at infinity.
- Points at infinity can be represented with finite coordinates.
- they allow for an easy combination of multiple transformations by concatenating several matrix-vector multiplication, including translations
- Affine transformation can be represented with a single matrix



Next lecture: Homogeneous coordinates

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-314

Homogeneous coordinates

WEEK-2: Lecture 5

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Lecture5: Homogeneous coordinates

The Euclidean space that we typically use is a non-curved space where in all the Euclidean geometry is applicable. For example: the distances to a point w.r.t. the origin are measured along orthogonal axes (in Cartesian coordinates).

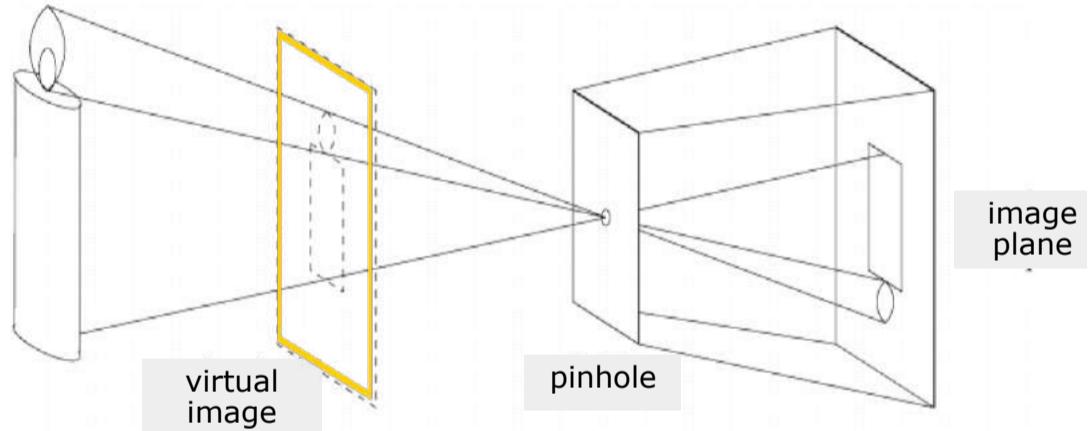
The homogeneous coordinates are commonly used tools in computer vision, robotics.

It allows us to perform mathematical tasks in an elegant & compact fashion compared to Euclidean space. Eg: perform transformation operations in the projective space.

It allows us use finite numbers for objects that are far in infinity for example in the Euclidean space.

Pinhole Camera is used to model to approximate the imaging process of a perspective camera.

Image Courtesy: Forsyth and Ponce



The distance between the camera center and image plane is the camera constant.

Camera center is the point where the rays of light intersect.

An inverted image (2D projection) of the object (3D) is formed at the back whose scale depends on the focal length & the camera constant.

For practical purposes, we rotate the image in front of the object s.t. it lies between the camera & the object.

The homogeneous coordinates allow us to perform transformations in the projective space.

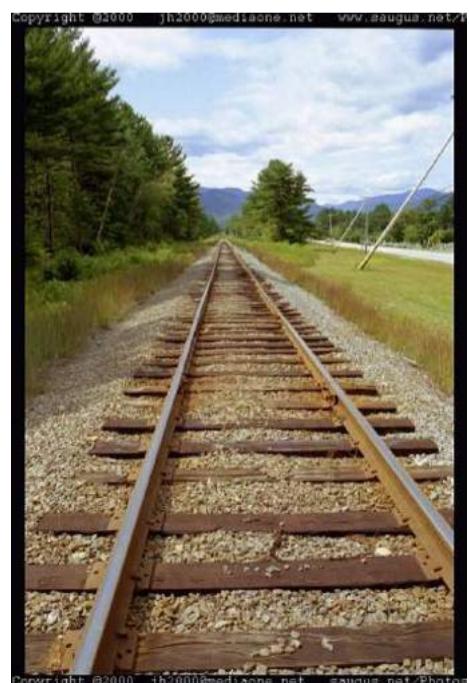
From the 3D world to image->

What happens to the geometrical properties of the image (perspective projection)?

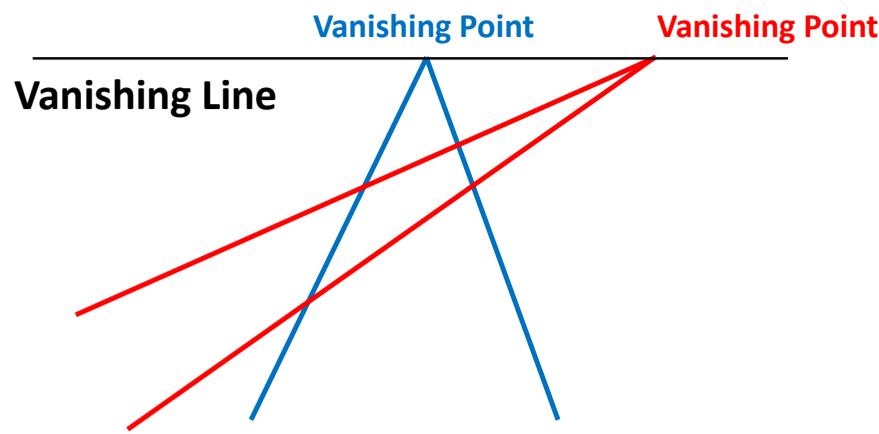


- Straight lines stay straight
- Parallel lines may not remain parallel
- Line-preserving: straight lines are mapped to straight lines
- Not length-preserving: size of objects is inverse proportional to the distance
- Not angle-preserving mapping: Angles between lines change

Vanishing points



- Parallel lines in the world intersect in the projected image at a “vanishing point”.
- Parallel lines on the same plane in the world converge to vanishing points on a “vanishing line”.
- The vanishing point is the “point at infinity” for the parallel lines Every direction has exactly one vanishing point
- E.G., the horizon.



How do we describe points at infinity? So, the parallel lines actually intersect at infinity. We want to encode such features in our projective geometry.

This, as we know is not possible in the Euclidean space (where parallel lines remain parallel). Hence, the Euclidian geometry is suboptimal to describe the central projection.

Projective geometry is an alternative algebraic representation of geometric objects and transformations.

Homogeneous coordinates (H.C.) are a system of coordinates used in projective geometry where the maths gets simpler to describe projective geometry than the Cartesian world (where we typically use Euclidean coordinates).

Two important properties of H.C. :

- Points at infinity can be represented using finite coordinates
- A single matrix can represent affine and projective transformations

Using matrix multiplications, we can describe the transformations including translations. This is not possible in the Euclidian space.

Representation \mathbf{X} of a geometrical object is called homogeneous, if \mathbf{X} and $\lambda\mathbf{X}$ represent the same object for $\lambda \neq 0$.

$$\mathbf{x} = \lambda \mathbf{x}$$

homogeneous

$$\mathbf{x} \neq \lambda \mathbf{x}$$

Euclidian

In Euclidean space, $\chi = \lambda\chi$ only when $\lambda=1$.

In the homogeneous coordinates λ represents the scale factor, meaning upto certain scale, it is possible to define this. For any value of λ (scaler), we are referring to the same object.

How is a point in Euclidean space defined in homogeneous coordinates (2D projective space)?

$$x = \begin{bmatrix} x \\ y \end{bmatrix} \quad \xrightarrow{\text{red arrow}} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Euclidian

homogeneous

In H.C., we use a $n+1$ dimensional vector to represent the n -dimensional Euclidian point and set dimension $n+1$ to the value 1 (or the last dimension).

Example:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 2 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2 \end{bmatrix} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

Euclidian homogeneous

A point X in 2D plane is represented by a 3D vector in H.C. s.t. its vector norm $\neq 0$ (meaning that coordinates $(0,0,0)$ is not allowed here):

$$\chi : \quad \mathbf{x} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad \text{with} \quad |\mathbf{x}|^2 = u^2 + v^2 + w^2 \neq 0$$

In order to convert from H.C. to Euclidian coordinates, we simply normalise by the last coordinate (or the scale factor):

$$\chi : \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \end{bmatrix} \text{ with } w \neq 0$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

When the last coordinate is 1, we can drop the last dimension in H.C. to go to Euclidian coordinates

homogeneous

Euclidian

So, what happens if the scale factor ($w=0$)?

It means that in Euclidian coordinates, the points are infinitely far away (since we divide by 'w'). So that point is infinity (∞).

We can represent points that are infinitely far away in H.C. as $\mathbf{x} = [x, y, 0]^\top$

So, H.C.'s are a good representation for vanishing points that we see in projective transformations, e.g., when representing the mapping from 3D object space to 2D image space when using a pinhole camera.

Remember that in the projective plane, the coordinates $[0, 0, 0]^\top$ do not exist.

How do we represent 3D points?

$$\mathbf{X} = \begin{bmatrix} U \\ V \\ W \\ T \end{bmatrix} = \begin{bmatrix} U/T \\ V/T \\ W/T \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} U/T \\ V/T \\ W/T \end{bmatrix}$$

homogeneous

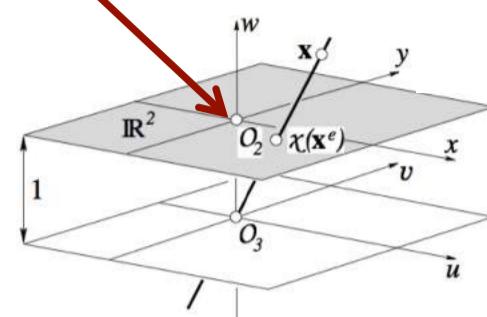
Euclidian

How do we represent origin of the Euclidian coordinates (E.C.) in H.C.?

The top grey-plane could represent the 2d-Euclidian space with origin at \mathbf{O}_2 represented in H.C. as $\mathbf{O}_2 = [0, 0, 1]^T$.

The plane below represents an infinitely far away plane that cannot be represented in the 2d-Euclidian space.

$$\mathbf{O}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{O}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



Transformations

Important property of H.C.'s is that a chain of transformations can be applied, this property is extensively used when we transform world coordinates objects to projective plane (we will take this up in the coming lectures).

The transformations are expressed as matrix vector multiplication.

The projective transformation is given as

The projective transformation is an invertible linear mapping.

Very quickly, what is an invertible linear transformation?

Let V and W be vector spaces over the field \mathbb{F} having the same finite dimension. Let $T:V \rightarrow W$ be a linear transformation. T is said to be invertible if there is a linear transformation $S:W \rightarrow V$

such that $S(T(x))=x$ for all $x \in V$.

S is called the inverse of T . In casual terms, S undoes whatever T does to an input x .

How can the homogenous matrix be used for translation? Suppose there is a translation in x, y & z. The homogeneous matrix can be expressed as:

$$H = \lambda \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix}$$

homogeneous property

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The matrix H here is 4X4 matrix for the 3d Euclidean space.

't': represents translation in x,y &z (3-parameters).

When we write this together, we have:

$$H = \lambda \begin{pmatrix} 1 & 0 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 & t_y \\ 0 & 0 & 1 & 0 & t_z \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Now when we multiply a 3D point 'X' with this 'H', it is equivalent to translating that point in the 3D-world.

Through matrix multiplication, we have translated a point here. This is not possible to achieve in this manner (mathematically) in the Euclidean space.

It turns out that it is possible to express all the transformations using matrix multiplications using H.C.s!

Now what if we want to do a rigid body transformation (rotation in 3D has 3 parameters)? Here, R^{3D} is the 3D rotation matrix.

$$H = \lambda \begin{pmatrix} R^{3D} & 0 \\ 0^T & 1 \end{pmatrix}$$

$$R_x^{3D}(\omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{bmatrix} \quad R_y^{3D}(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

$$R_z^{3D}(\kappa) = \begin{bmatrix} \cos(\kappa) & -\sin(\kappa) & 0 \\ \sin(\kappa) & \cos(\kappa) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R^{3D}(\omega, \phi, \kappa) = R_z^{3D}(\kappa) R_y^{3D}(\phi) R_x^{3D}(\omega)$$

The rotation specified in this manner determine an amount of rotation about each of the individual axes of the coordinate system. The angles in x,y,z are also called the Euler angles. They can be used to describe an off-axis rotation, by combining Euler angle rotations via matrix multiplication. The order of rotation affects the end result, so besides specifying Euler angles, an order of rotation must be specified.

These are associative but are not commutative, meaning that the order in which operations are done is highly important. Hence, in the above example, rotation in 'z' then 'y' and then 'x' (as seen above) is not same as in 'x', then 'y' and then 'z'.

For the 2D case, we will use the rotation matrix

$$R^{2D}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

And so, the matrix H will be a 3X3 matrix for the 2D case.

When we want to perform rotation & translation together (motion) at the same time in 3D. We have,

$$H = \lambda \begin{pmatrix} R^{3D} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

Where, we have 6 parameters (3 for translation & 3 for rotation).

Similarity transformation (angle preserving):

This takes into account the scale factor.

We have 7 parameters in this case (3 for translation & 3 for rotation as seen in the previous case & 1 parameter for the scale):

$$H = \lambda \begin{pmatrix} mR^{3D} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

This transformation will make the objects larger or smaller. Here 'm' is the scaling factor.

Common scaling transformation

$$H = \lambda \begin{pmatrix} sI & 0 \\ \mathbf{0}^T & 1 \end{pmatrix}$$

If $s > 1$, it will lead to magnification, if $0 < s < 1$, then the size will be reduced than the original size. If $s < 1$, this will cause in addition to the scaling, a mirroring at the origin, thus a rotation by 180° around the origin.

Shear of axes symmetrically with a parameter s:

$$H = \lambda \begin{pmatrix} 1 & s/2 & 0 \\ s/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Affine transformations

Not angle preserving but parallel lines remain parallel

$$H = \lambda \begin{pmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

(12 parameters : 3 translation+ 3 rotation+ 3 scale+ 3 shear)

Here 'A' is an arbitrary matrix with 9 parameters (3 rotation+ 3 scale+ 3 shear) in 3D.

In 2d: six parameters: 2 translations, 1 rotation, 2 individual scalings, and 1 shear of the axes.

Projective transformation

$$H = \lambda \begin{pmatrix} A & \mathbf{t} \\ \mathbf{a}^T & 1 \end{pmatrix}$$

They have the same number parameters for affine('A') and additional 3 parameters representing projection (total 15 parameters). i.e. mapping parallel lines to converging lines. This projective transformation will be extensively applied in the analysis of one and two views.

It can easily be verified that the more general transformation matrices can be composed of some of the simpler transformation matrices by multiplication.

How are transformations using H.C.s helpful?

- The transformation matrix can be used for converting the location of vertices between different coordinate systems.
- These can be applied sequentially since we only know the sequential transformations between different coordinate systems.
- The homogeneous transform provides a convenient means of constructing compound transformations.

For instance, we have a series of coordinate systems, a, b, c, d.

We ask, "What is the position of a point v_d defined in coordinate d, when viewed from coordinate a?"

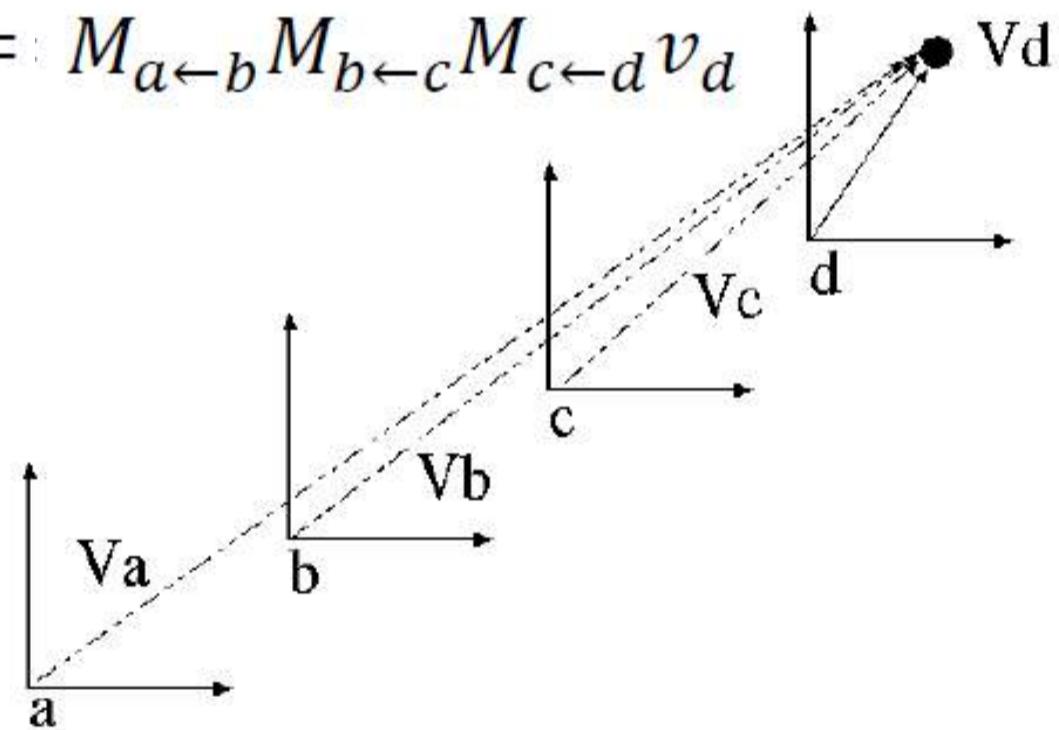
We know the relative transformations between the adjacent coordinate systems:

M_{cf} is the transformation matrix, where 'cf' is the coordinate frame.

$$v_c = M_{c \leftarrow d} v_d \quad (\text{expresses } 2 \text{ transformations})$$

$$v_b = M_{b \leftarrow c} v_c = M_{b \leftarrow c} M_{c \leftarrow d} v_d$$

$$v_a = M_{a \leftarrow b} v_b = M_{a \leftarrow b} M_{b \leftarrow c} M_{c \leftarrow d} v_d$$



However, the matrix product are not commutative, meaning $\mathbf{X}' = \mathbf{H}_1 \mathbf{H}_2 \mathbf{X}$ that
 $\neq \mathbf{H}_2 \mathbf{H}_1 \mathbf{X}$

The transformations are invertible:

$$\mathbf{X}' = \mathbf{H}\mathbf{X}, \& \mathbf{X} = \mathbf{H}^{-1}\mathbf{X}'$$

Points and lines

Lines, distances, angles

- Lines in 3D project to lines in 2D.
- Distances and angles are not preserved.
- Parallel lines do not in general project to parallel lines (unless they are parallel to the image plane).

Vanishing points: The parallel lines in space project perspectively onto lines that on extension intersect at a single point in the image plane called vanishing point or point at infinity.

Vanishing line:

- The vanishing points of all the lines that lie on the same plane form the vanishing line.
- It is the intersection of a parallel plane through the center of projection with the image plane.

in Cartesian: $P = (x, y) \text{ s.t } P \in \mathbb{R}^2$

In H.C. : $\tilde{P} = (\tilde{x}, \tilde{y}, 1) \text{ s.t } \tilde{P} \in \mathbb{P}^2$

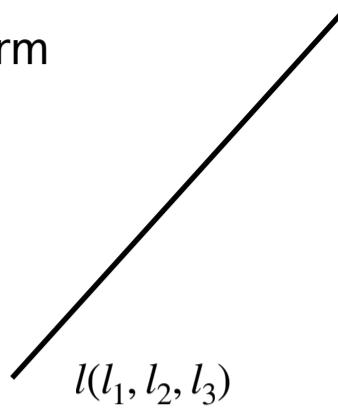
Hmogeneous \rightarrow Cartesian

$$\tilde{P} = (\tilde{x}, \tilde{y}, \tilde{z}) \rightarrow P = (\tilde{x}/\tilde{z}, \tilde{y}/\tilde{z}, 1) \text{ or } P = (x, y); x = \frac{\tilde{x}}{\tilde{z}}, y = \frac{\tilde{y}}{\tilde{z}}$$

Representation of a line

$$(\cos \phi)x + (\sin \phi)y - d = 0 \quad \text{Hesse normal form}$$

$$\left(\frac{1}{x_0} \right) x + \left(\frac{1}{y_0} \right) y - 1 = 0 \quad \text{Intercept form}$$



$$ax + by + c = 0 \quad \text{Standard form}$$

$$\begin{array}{lll} \text{Line forms:} & \mathbf{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} & \mathbf{l} = \begin{bmatrix} \cos \phi \\ \sin \phi \\ -d \end{bmatrix} \\ \text{standard} & & \text{Hesse} \\ & & \text{intercept} \end{array} \quad \mathbf{l} = \begin{bmatrix} \frac{1}{x_0} \\ \frac{1}{y_0} \\ -1 \end{bmatrix}$$

$$\begin{array}{ll} \mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} & \text{The condition that a point lies on a line is:} \\ \text{point} & \end{array}$$

$$\boxed{\mathbf{x} \cdot \mathbf{l} = 0}$$

The vectors (p_1, p_2) and (q_1, q_2) determines the same line through the origin in projective space iff $(p_1, p_2) = \lambda(q_1, q_2), \lambda \neq 0$

In H.C.s, of a line in the plane is a 3-dim. vector:

$$\ell : \quad \mathbf{l} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} \quad \text{with } |\mathbf{l}|^2 = l_1^2 + l_2^2 + l_3^2 \neq 0$$

Which in Euclidean coordinates corresponds to : $l_1x + l_2y + l_3 = 0$

A point $x = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ lies on a line $l = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}$ if $\mathbf{x} \cdot \mathbf{l} = 0$

Intersection of two lines

Given two lines l, m expressed in H.C., we look for the intersection

$$\kappa = l \cap m$$

Find the point $\mathbf{x} = [x, y]^T$ through the following system linear equations

$$\begin{bmatrix} \mathbf{l} \cdot \mathbf{x} \\ \mathbf{m} \cdot \mathbf{x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Which mean that we find the point such that

$$l \cdot x = 0 \quad \& \quad m \cdot x = 0$$

Let us see how we solves an equation of the form:

$$\begin{bmatrix} l_1 & l_2 \\ m_1 & m_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -l_3 \\ -m_3 \end{bmatrix}$$

A system of linear equation of the $A\mathbf{x} = \mathbf{b}$ form can be solved via Cramer's rule

$$\begin{bmatrix} l_1 & l_2 \\ m_1 & m_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -l_3 \\ -m_3 \end{bmatrix}$$

$$x = \frac{D_1}{D_3} \quad y = \frac{D_2}{D_3} \quad \begin{aligned} D_1 &= \det(A_1) = l_2m_3 - l_3m_2 \\ D_2 &= \det(A_2) = l_3m_1 - l_1m_3 \\ D_3 &= \det(A) = l_1m_2 - l_2m_1 \end{aligned}$$

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} D_1/D_3 \\ D_2/D_3 \\ 1 \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix}$$

Homogeneously expressed as :

In vector form:

$$\mathbf{x} = \frac{1}{D_3} \mathbf{D} = \mathbf{l} \times \mathbf{m}$$

Which is the cross product of the two lines.

So, the intersection of two lines in H.C. is

$$\mathcal{C} = l \cap m : \mathbf{x} = \mathbf{l} \times \mathbf{m}$$

A line (l) through two given points x and y :

$$l = x \wedge y : \mathbf{l} = \mathbf{x} \times \mathbf{y}$$

Points and lines at infinity:

It is possible to explicitly model infinitively distant points with finite coordinates

A point at infinity is expressed as

$$\mathbf{x}_\infty = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix}$$

All lines l with $(\mathbf{l} \cdot \mathbf{x}_\infty = \mathbf{0})$ pass through x_∞ . This hold true for all lines parallel to l

All parallel lines meet at one point at infinity!



For 2 parallel lines l, m , their intersection in H.C. is

$$\mathbf{l} \times \mathbf{m} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \times \begin{bmatrix} a \\ b \\ d \end{bmatrix} = \begin{bmatrix} bd - bc \\ ac - ad \\ ab - ab \end{bmatrix} = \begin{bmatrix} bd - bc \\ ac - ad \\ 0 \end{bmatrix} = \begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}$$

An infinitely distant point

$$x_\infty : \quad \mathbf{x}_\infty = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix}$$

An infinitely distant line

$$l_\infty : \quad \mathbf{l}_\infty = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The line l_∞ can be interpreted as the horizon.

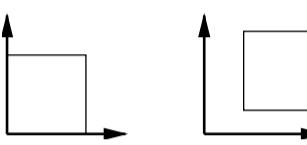
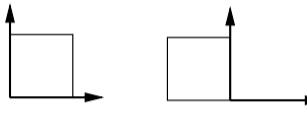
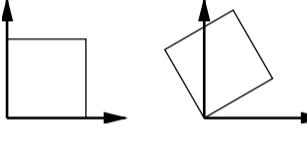
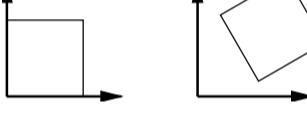
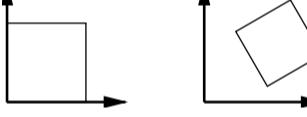
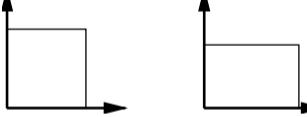
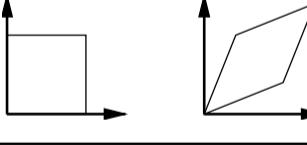
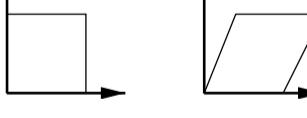
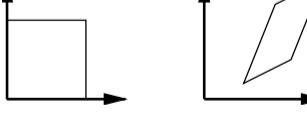
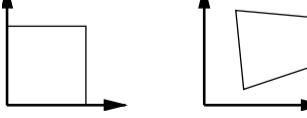
The ideal line (or horizon line): All points at infinity lie on the line at infinity called the ideal line given by :

$$\mathbf{x}_\infty \cdot \mathbf{l}_\infty = \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

Which represent infinitely far away objects. Here, (u,v) indicate the direction or orientation of the line so that we can maintain direction to the infinitely far away point.

So, in the projective plane \mathbb{P}^2 ,

- all point of the Euclidean plane \mathbb{R}^2 with $x = [x,y]^T$ are taken as $x=(x,y,1)^T$.
- All the points at infinity are included represented as $x=(x,y,0)^T$.
- The point $(0,0,0)$ is not part of projective plane \mathbb{P}^2 .

2D Transformation	Figure	d.o.f.	H	H
Translation		2	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} I & t \\ \mathbf{0}^T & 1 \end{bmatrix}$
Mirroring at y -axis		0	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} Z & 0 \\ \mathbf{0}^T & 1 \end{bmatrix}$
Rotation		1	$\begin{bmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} R & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Motion		3	$\begin{bmatrix} \cos \varphi & -\sin \varphi & t_x \\ \sin \varphi & \cos \varphi & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} R & t \\ \mathbf{0}^T & 1 \end{bmatrix}$
Similarity		4	$\begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \lambda R & t \\ \mathbf{0}^T & 1 \end{bmatrix}$
Scale difference		1	$\begin{bmatrix} 1 + m/2 & 0 & 0 \\ 0 & 1 - m/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Shear		1	$\begin{bmatrix} 1 & s/2 & 0 \\ s/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} S & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Asym. shear		1	$\begin{bmatrix} 1 & s' & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} S' & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Affinity		6	$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} A & t \\ \mathbf{0}^T & 1 \end{bmatrix}$
Projectivity		8	$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$	$\begin{bmatrix} A & t \\ \mathbf{p}^T & 1/\lambda \end{bmatrix}$

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-314

Camera models and parameters

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Camera models and parameters

Basic definitions:

Frame of reference: measurements are made with respect to a particular coordinate system called the frame of reference.

World frame: a fixed coordinate system for representing objects (points, lines, surfaces, etc.) in the world.

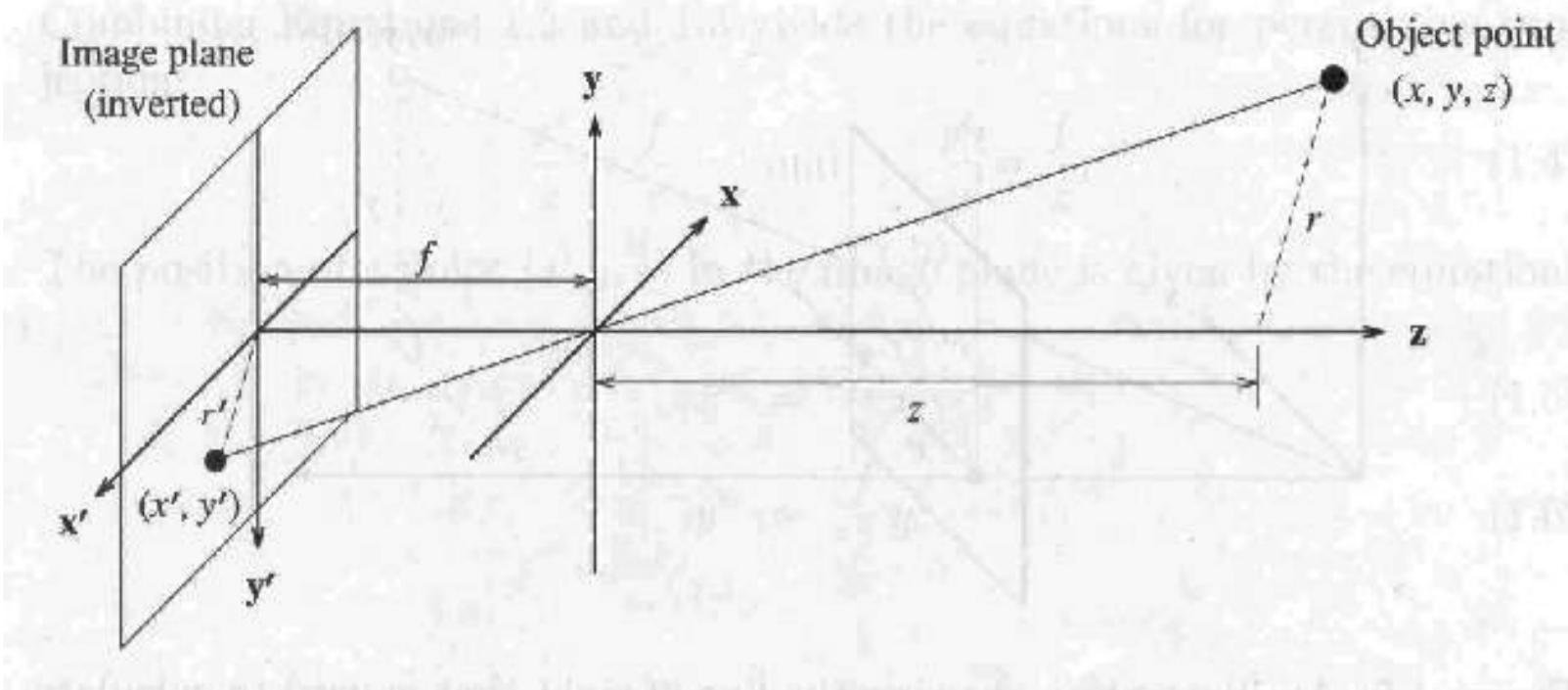
Camera frame: coordinate system that uses the camera center as its origin (and the optic axis as the Z-axis).

Image or retinal plane: plane on which the image is formed, note that the image plane is measured in camera frame coordinates (mm).

Image frame: coordinate system that measures pixel locations in the image plane.

Intrinsic parameters allow a mapping between camera coordinates and pixel coordinates in the image frame. These are camera parameters that are internal and fixed to a particular camera/digitization setup.

Extrinsic parameters define the location and orientation of the camera with respect to the world frame. These are camera parameters that are external to the camera and may change with respect to the world frame.

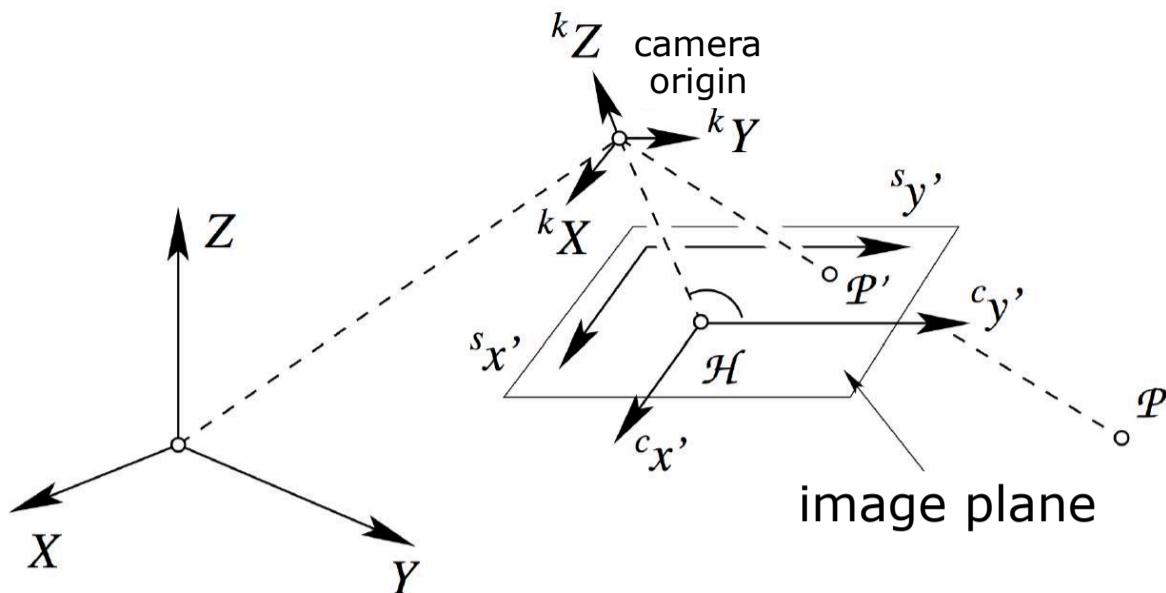


- We are concerned with the pinhole or perspective camera.
- So we need the mapping from points in the 3-D world (object points) to 2-D image points.
- Mapping is expressed (equivalently) as a linear mapping of homogeneous coordinates.
- A point in the 3D world is expressed as a 4-vector $x = (x, y, z, t)^\top$ representing the point $(x/t, y/t, z/t)$ and an image point is expressed as $u = (u, v, w)^\top$ representing the point $(u/w, v/w)$ in Euclidean coordinates.

The 2-types of parameters need to be recovered in order for us to reconstruct the 3D structure of a scene from the pixel coordinates of its image points:

- Extrinsic camera parameters: the parameters that define the location and orientation of the camera reference frame with respect to a known world reference frame.
- Intrinsic camera parameters: the parameters necessary to link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

Camera orientation in the world



Pinhole camera revisited: We want to describe “How a 3D Point is Mapped to a 2D Pixel Coordinate?”

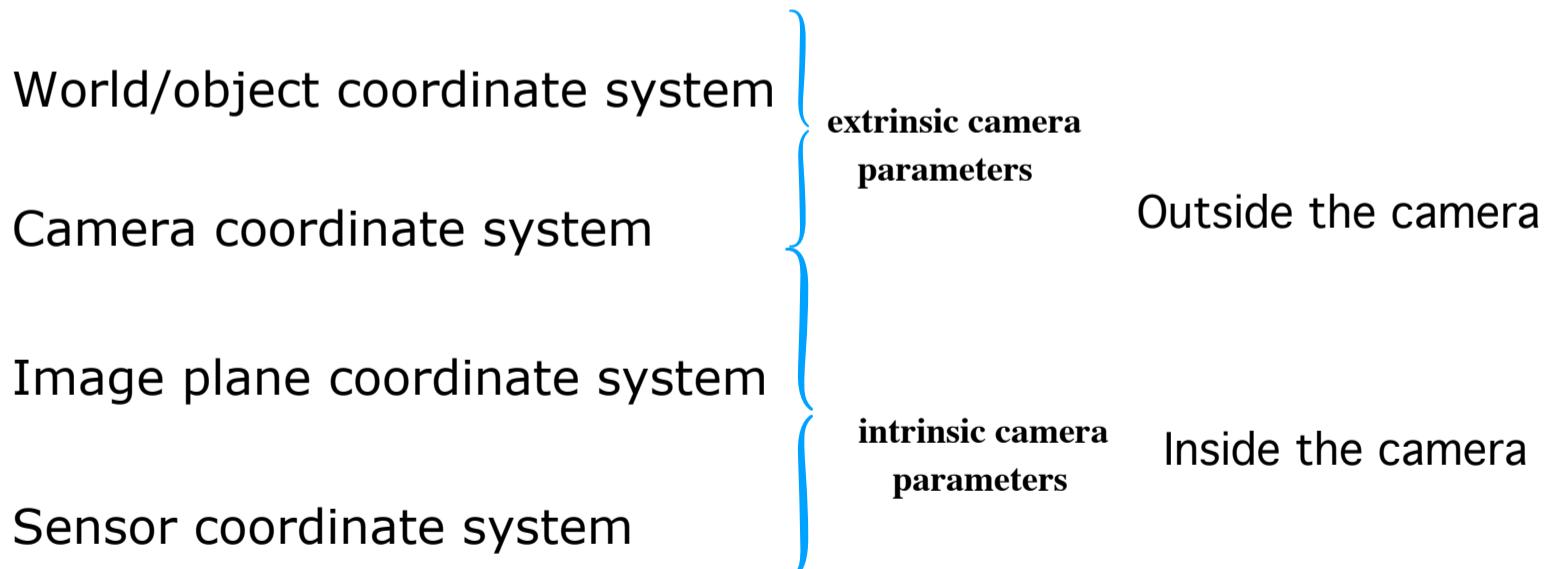
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D pixel coordinate transformation 3D world coordinate

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

pixel coordinate transformation world coordinate

Goal: The position of the camera in the world must be first recovered



World/object coordinate system (S_0): $[X_{\mathcal{P}}, Y_{\mathcal{P}}, Z_{\mathcal{P}}]^T$

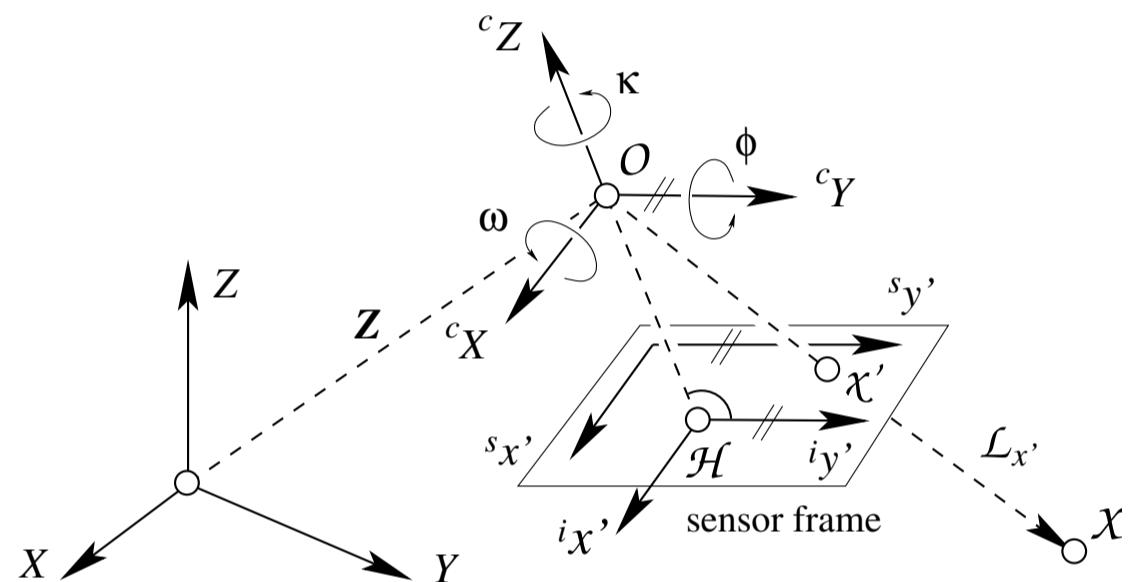
Camera coordinate system (S_K): $[kX, kY, kZ]^T$

Image plane coordinate system (S_c): $[^c x, ^c y]^T$

Sensor coordinate system (S_s) (2D pixel coordinates): $[s_x, s_y]^T$

From the world to sensor:

How are 3D points mapped to 2D image?

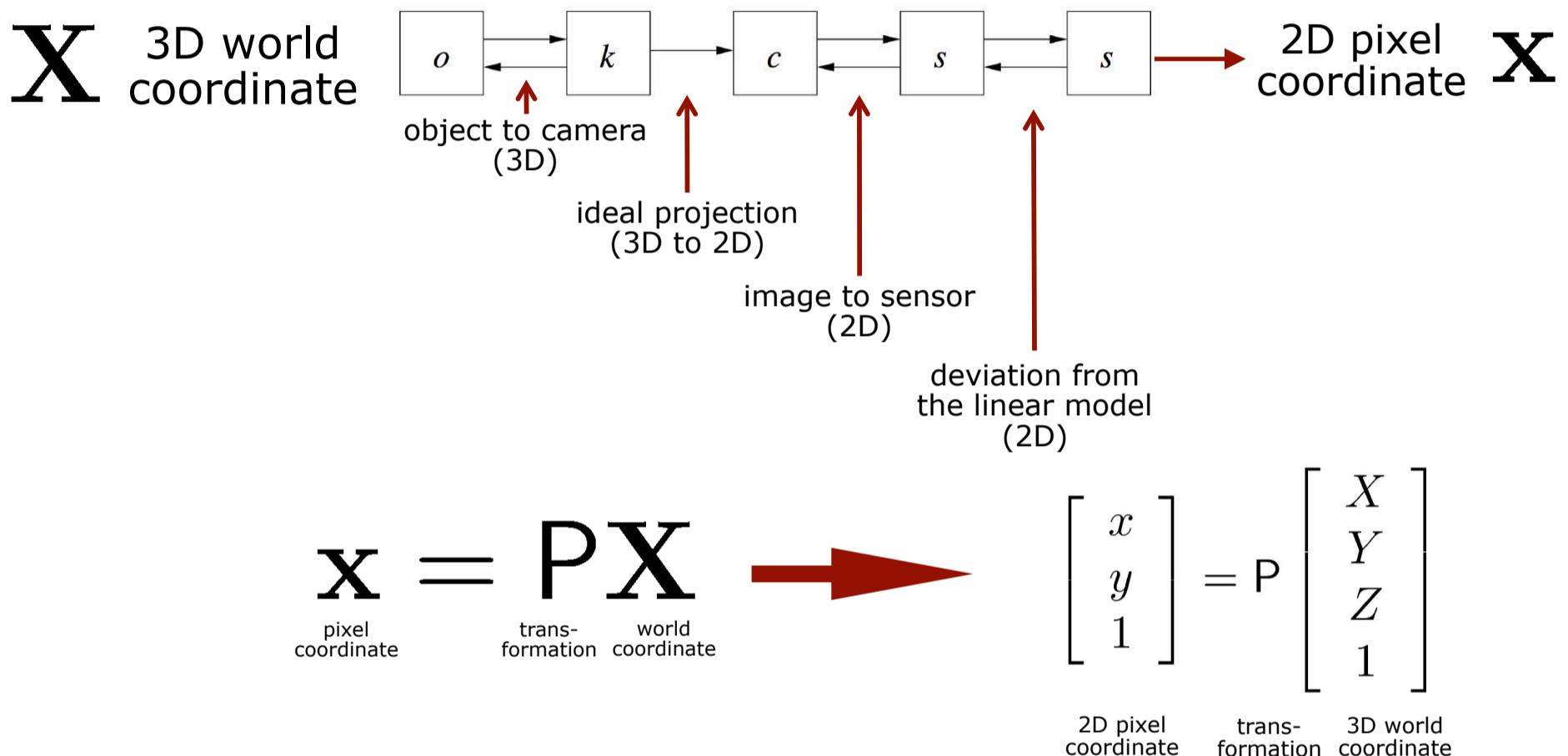


- Perspective projection of an object point X with a camera into the image point x' . Coordinate systems: object coordinate system $[X, Y, Z]$,

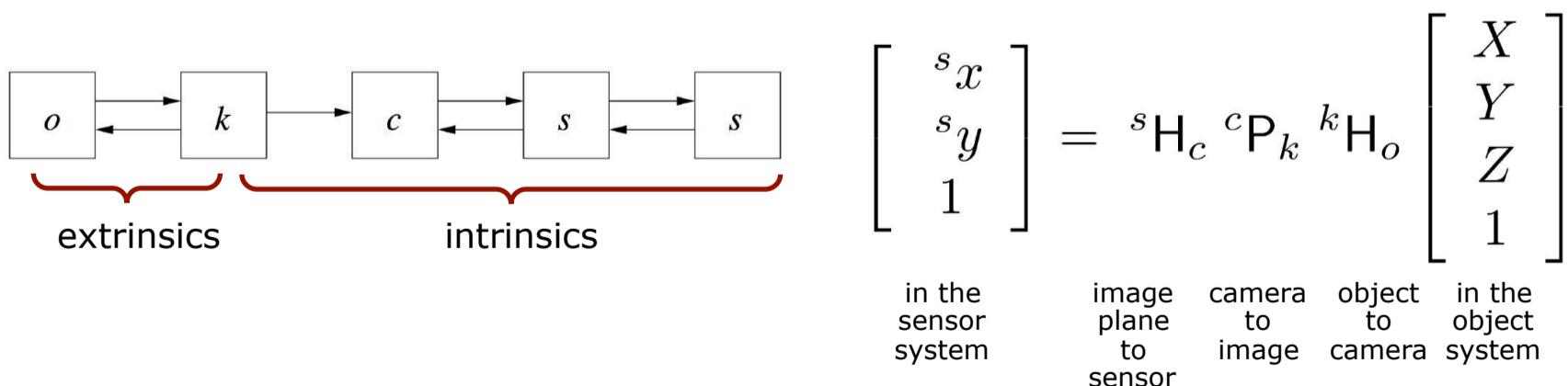
- projection centre O ,
- camera coordinate system $[cX, cY, cZ]$,
- image coordinate system $S_i, [i_x', i_y']$, with origin in the principal point H ,
- sensor coordinate system $[s_x', s_y']$.
- We assume the y -axis of the sensor and the y -axes of the camera to be parallel to the rows of the sensor. When taking the normalized directions Ox' as observable entities in the camera system we arrive at the spherical projection. The right-hand

rotations around the three coordinate axes of the camera system are denoted by ω , ϕ , and κ .

- The unit camera and the ideal spherical camera without distortions are both characterised by the six parameters of the exterior orientation only.
- The non-linear errors are taken care in the end through a transformation in the sensor frame. Until then, we are in linear frame.



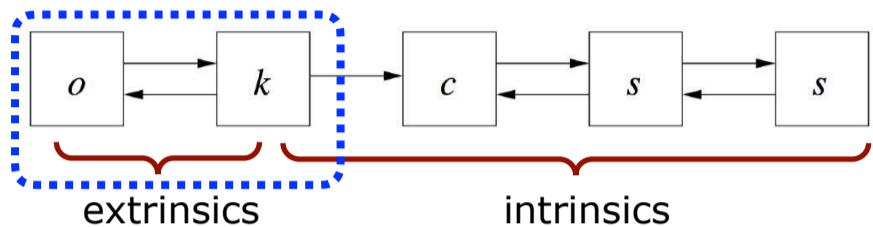
We need to recover the transformation matrix ' \mathbf{P} '. What parameters are needed to do mapping & why?



How are extrinsics defined?

The extrinsic are affected with the position of the camera in this world, however the camera intrinsics remain the same for ‘a’ given camera.

This mapping is required to make geometric estimates: such as, where is the camera? Or where where is 3D word object?s



Extrinsic parameters (from 3D object world to camera)

What is the pose of the camera in the 3d world?

- We have world coordinates for an object.
- All models for single cameras represent a mapping from object space to image space with a unique projection centre, as the projection rays all pass through a single point. This is valid for both perspective as well as spherical cameras.
- Camera is in world, centred at $(0,0,c)$ (projection centre of the camera/camera origin).
$$[{}^kX, {}^kY, {}^kZ]^T$$
- The image plane is shifted in the world by a constant ‘c’ or camera constant. ‘c’ defines where the image plane is located from the projection centre of the camera. The $(0,0)$ of image plane lies on the optical axis.
- Finally, we have the sensor coordinate system such that pixel coordinates start from $(0,0)$.
- The image plane is at an offset ‘c’ or ‘f’ along the optical axis from the camera, and so origin of image plane in the camera coordinate system is

$${}^kO_c = {}^k[0, 0, c]^T \text{ (with } c < 0\text{)}$$

What parameters required?

The camera cannot change the world or scale-up/down, so the only possible movements are translations and rotations.

There are 6 parameters: (invertible transformations)

We have a point \mathcal{P} with coordinates in world coordinates

$$\mathbf{X}_{\mathcal{P}} = [X_{\mathcal{P}}, Y_{\mathcal{P}}, Z_{\mathcal{P}}]^T$$

Center O of the projection (origin of the camera coordinate system)

$$\mathbf{X}_O = [X_O, Y_O, Z_O]^T$$

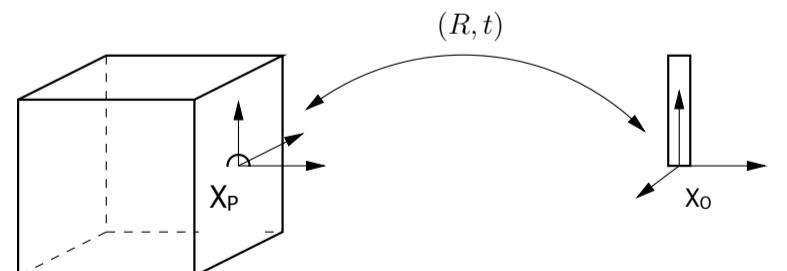
The possible transformations are translation and rotations from the origin of the world coordinates to the camera coordinate system.

In Euclidean system:

$${}^k \mathbf{X}_{\mathcal{P}} = R(\mathbf{X}_{\mathcal{P}} - \mathbf{X}_O)$$

It is often the case that the relation between object coordinates system and camera coordinate system are not known. For this purpose it is necessary to model the relation between two different coordinate systems in 3D. The natural way to do this is to model the relation as a Euclidean transformation.

Using different coordinate systems for the camera and the object.



In E.C.

$$\begin{aligned} \begin{bmatrix} {}^k \mathbf{X}_{\mathcal{P}} \\ 1 \end{bmatrix} &= \begin{bmatrix} R & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} I_3 & -\mathbf{X}_O \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathcal{P}} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} R & -R\mathbf{X}_O \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathcal{P}} \\ 1 \end{bmatrix} \end{aligned}$$

The Euclidean transformation can be written in homogeneous coordinates as:

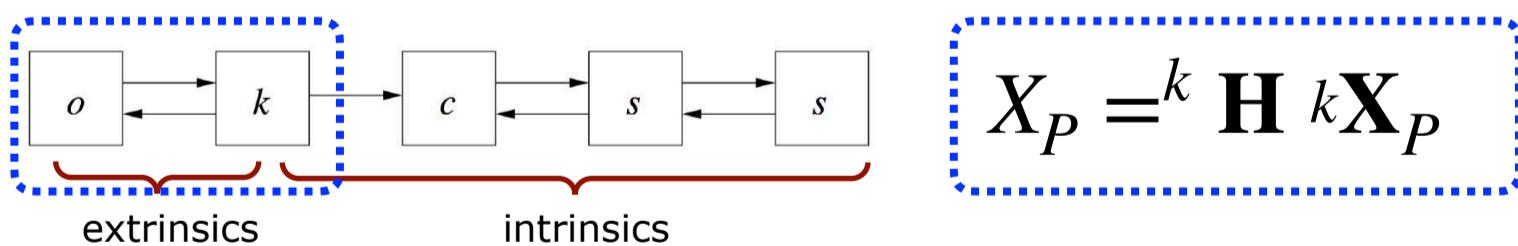
$${}^k \mathbf{X}_P = {}^k \mathbf{H} \mathbf{X}_P \quad {}^k \mathbf{H} = \begin{bmatrix} R & -RX_O \\ \mathbf{0}^T & 1 \end{bmatrix}$$

These operations here are carried out in sequence where 'R' denotes an orthogonal matrix and 't' a vector, encoding the rotation and translation in the rigid transformation.

It is also a practice (not so uncommon, that you will see at several places), to express the world coordinates in camera coordinates. Then, it is possible to express the same in H.C.s using matrix multiplication such that both rotation and translation are expressed as a single, composed transform. Here, this means that it is possible to perform rotation and translation simultaneously. Instead, off writing a sequence of matrix operations, both transformations are included within a single matrix. This is expressed in H.C.

We represent a 4×4 matrix ${}^k \mathbf{H}$ which performs the motion of the object system into the camera system (having rotation & translation in 3D: 6 parameters):

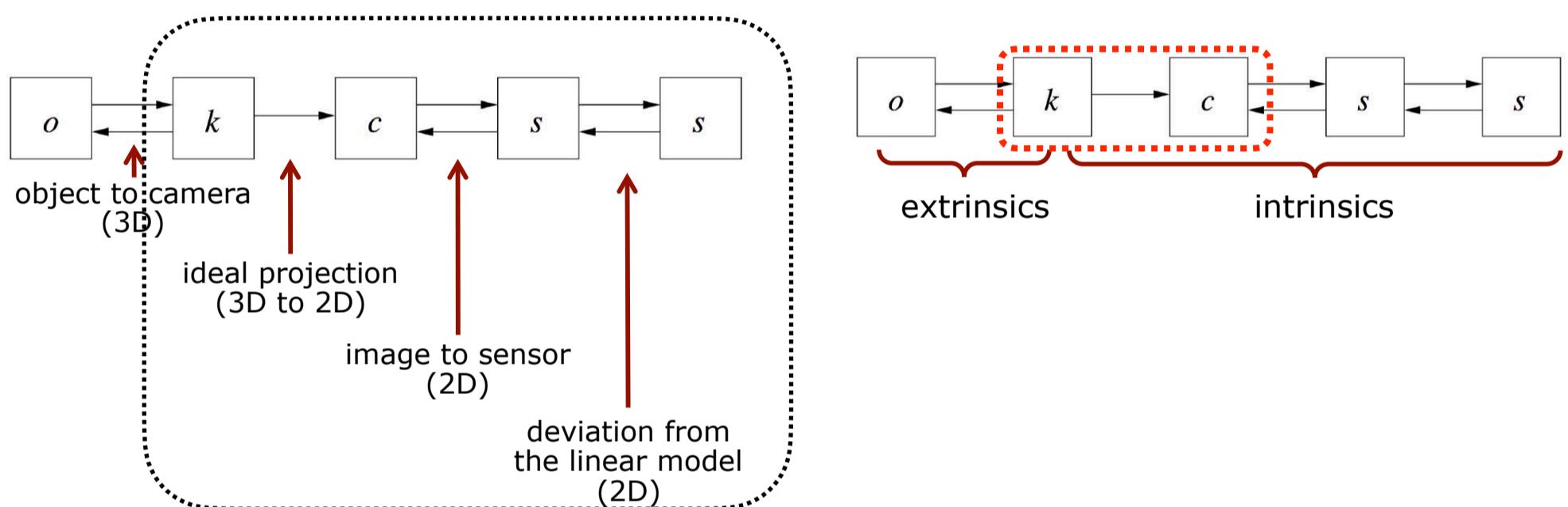
$${}^k \mathbf{H}(R, X_O) = \begin{pmatrix} R & \mathbf{X}_O \\ \mathbf{0}^T & 1 \end{pmatrix}$$



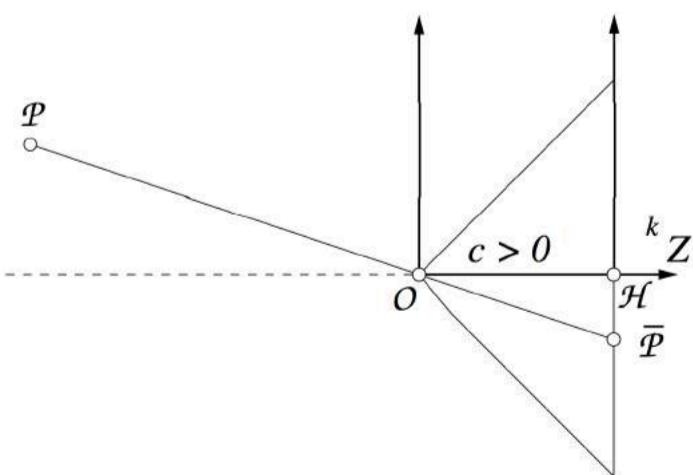
Intrinsic parameters

We split up the (intrinsic) mapping into 3 steps:

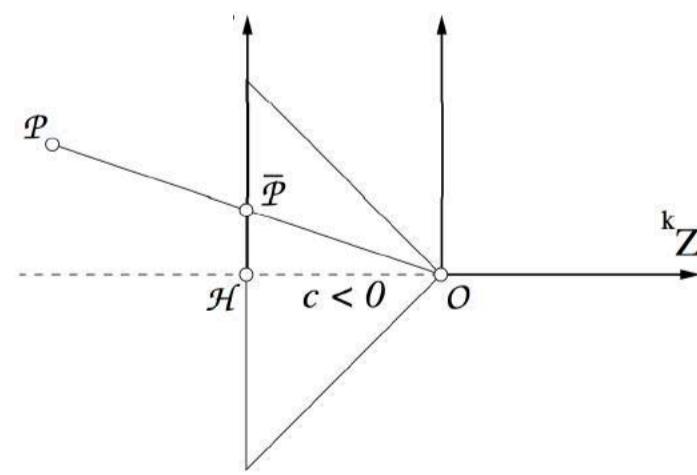
1. Ideal perspective projection to the image plane
2. Mapping from image plane to the sensor coordinate system (“where the pixels are”)
3. Compensation for non-linear errors/distortions since our mapping so far are idealized



1. Ideal perspective projection to the image plane

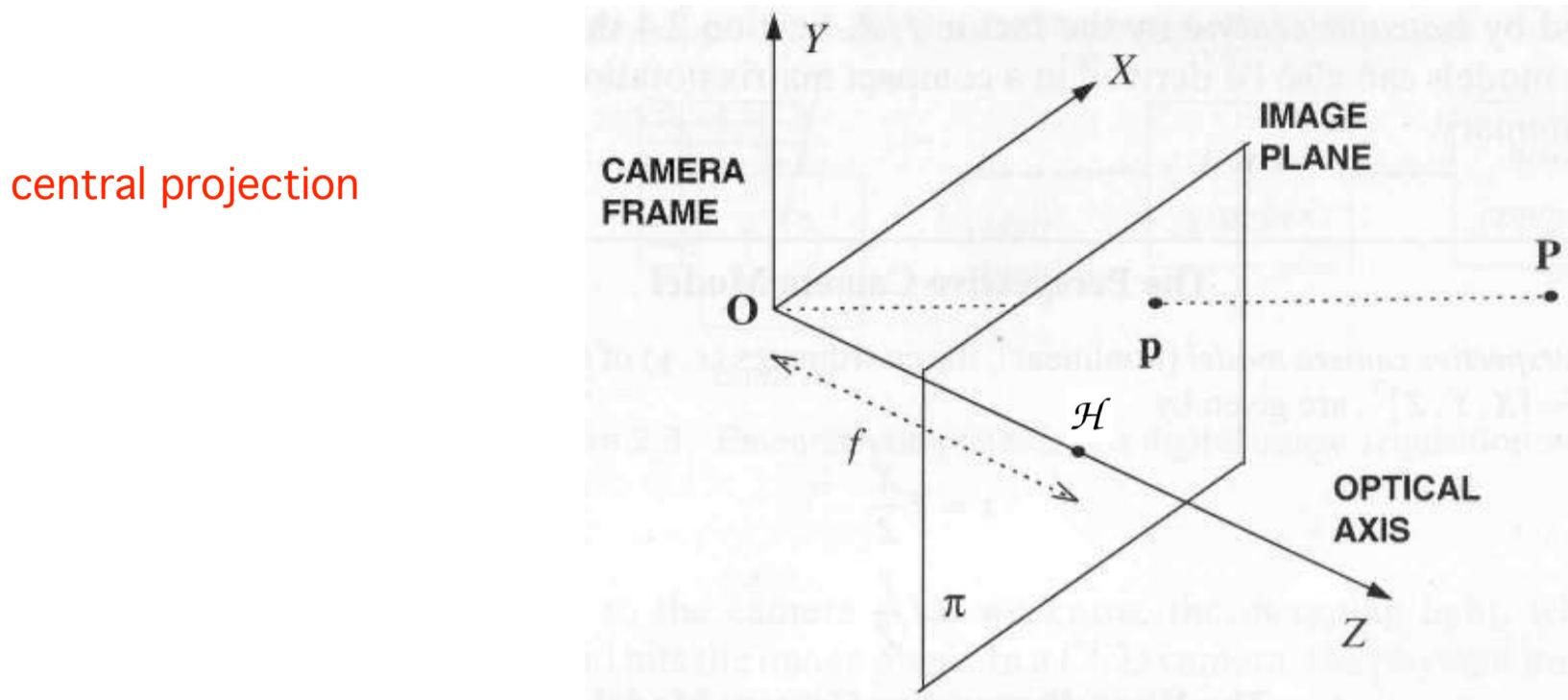


Physically motivated: $c > 0$



Popularly used: $c < 0$

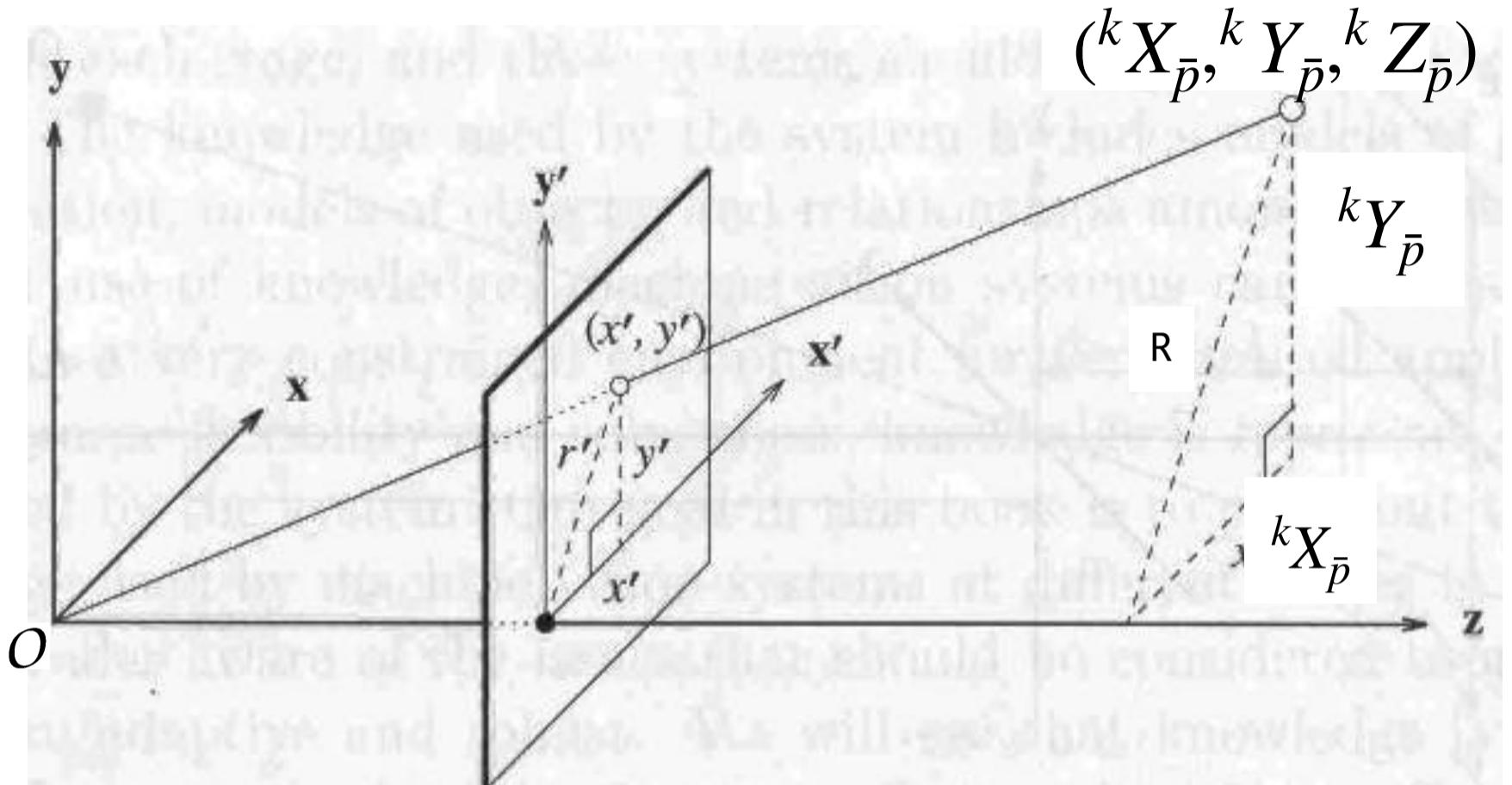
We avoid image inversion by assuming that the image plane is in front of the center of projection.



Considering ideal perspective projection

- Distortion-free lens
- From 3D to 2D (inside camera)
- Centre of projection coincides with the origin of the world
- Camera axis (optical axis) is aligned with the world coordinate z -axis.
- Non-invertible transform as we loose 1 d.o.f.
- All rays are straight lines and pass through the projection center. This point is the origin of the camera coordinate system O
 - The line through O and perpendicular to the image plane is the optical axis.
 - The model consists of a plane (image plane) and a 3D point O (center of projection).
 - The intersection of the optical axis with the image plane is called principal point or image center (\mathcal{H}).
 - Focal point and principal point lie on the optical axis.

- The distance from the camera origin (O) to the image plane is the constant 'f' or 'c' (here we flip the image in front of the camera s.t. it lies between the 3D world and the camera coordinates i.e. rotation of 180° , so $f < 0$).
 (note: the principal point is not always the "actual" center of the image)



Using triangle similarity, for $((^kX_{\bar{p}}, ^kY_{\bar{p}}, ^kZ_{\bar{p}}), R) \& ((x', y', z'), r')$, we write:

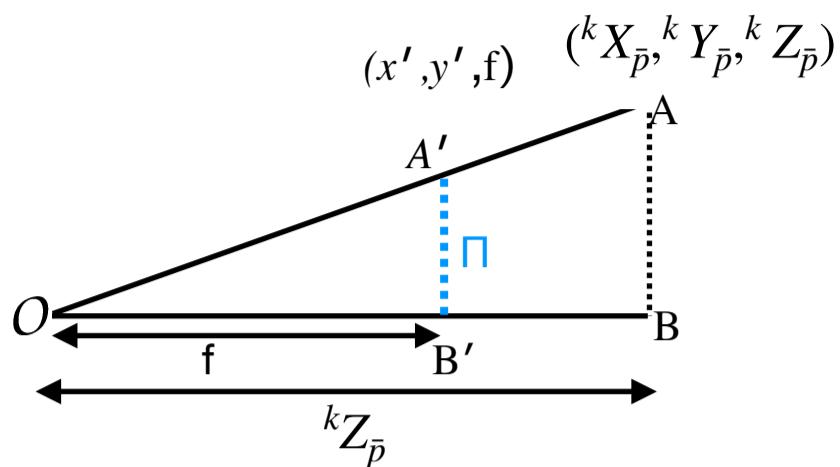
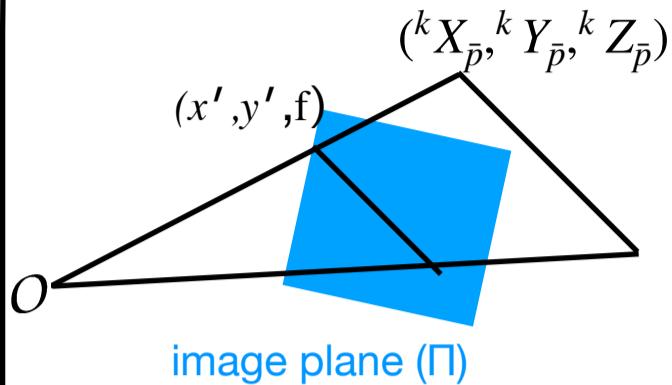
$$\frac{f}{kZ_{\bar{p}}} = \frac{r'}{R} \quad \frac{x'}{kX_{\bar{p}}} = \frac{y'}{kY_{\bar{p}}} = \frac{r'}{R}$$

$$x' = f \frac{kX_{\bar{p}}}{kZ_{\bar{p}}} \quad y' = f \frac{kY_{\bar{p}}}{kZ_{\bar{p}}} \quad z' = f$$

$$\text{Let } (x', y') = (^c x_{\bar{p}}, ^c y_{\bar{p}})$$

$$^c x_{\bar{p}} = f \frac{kX_{\bar{p}}}{kZ_{\bar{p}}} \quad ^c y_{\bar{p}} = f \frac{kY_{\bar{p}}}{kZ_{\bar{p}}}$$

2D representation showing the same derivation:



Here, OA'B' and OAB are similar triangles, using this , we have:

$\frac{kZ_p}{kX_p} = \frac{f}{x'} \text{, so } \frac{f}{x'} = \frac{kZ_p}{kX_p} \text{ so } x' = \frac{f kX_p}{kZ_p}$ similarly $y' = \frac{f kY_p}{kZ_p}$. This shows that the object gets smaller, as we move further away from the screen (as seen that kZ_p is in the denominator). The H.C.s provide a good way to represent this fact.

The above is the ideal perspective projection equation.

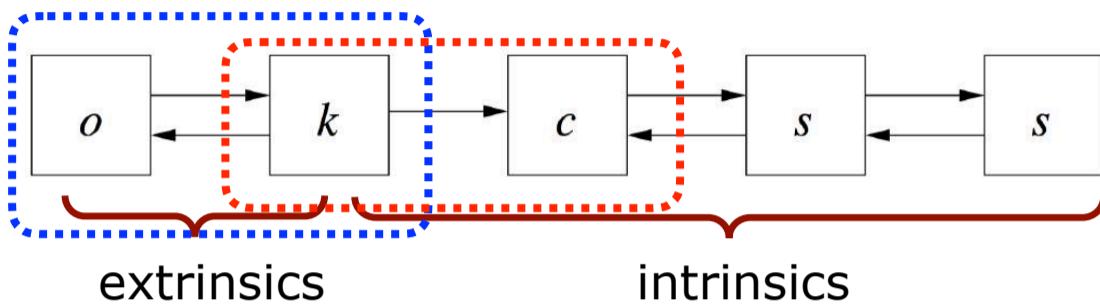
Using matrix notation & H.C :

$$\begin{pmatrix} {}^c x_{\bar{p}} \\ {}^c y_{\bar{p}} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} kX_{\bar{p}} \\ kY_{\bar{p}} \\ kZ_{\bar{p}} \end{pmatrix}$$

$$\begin{pmatrix} {}^c x_{\bar{p}} \\ {}^c y_{\bar{p}} \\ {}^c w_{\bar{p}} \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} kX_{\bar{p}} \\ kY_{\bar{p}} \\ kZ_{\bar{p}} \\ 1 \end{pmatrix}$$

Note that the mapping is from 3D world onto the 2D plane through this matrix. We have dropped a dimension through this projection and the Z-coordinate of this point on the image plane will be the focal distance. Here, when ${}^c w_{\bar{p}} = 1$, we can verify that the matrix expression in H.C. gives the ideal perspective projection equation.

Our mapping so far:



So, we have

$${}^c \mathbf{x} = {}^c \mathbf{P} \mathbf{X}$$

From world to camera coordinates

$${}^c \mathbf{P} = {}^c \mathbf{P}_k {}^k \mathbf{H} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R & -R \mathbf{X}_0 \\ 0^T & 1 \end{pmatrix}$$

From camera to image plane

$${}^c \mathbf{P}_k = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad {}^k \mathbf{H} = \begin{pmatrix} R & -R \mathbf{X}_0 \\ 0^T & 1 \end{pmatrix}$$

We have done a mapping using the intrinsic and extrinsic parameters.

We can define the calibration matrix for the ideal camera , where ${}^c K$ is the calibration matrix for the ideal camera.

$${}^cK = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad {}^cP = {}^cK[R] - RX_O = {}^cK R [I_3] - X_O$$

3x4 matrices

$$[I_3] - X_O = \begin{bmatrix} 1 & 0 & 0 & -X_O \\ 0 & 1 & 0 & -Y_O \\ 0 & 0 & 1 & -Z_O \end{bmatrix}$$

With this calibration matrix, we can map a point ${}^c\mathbf{x}$ to the image plane:

$${}^c\mathbf{x} = {}^cKR[I_3] - X_O \mathbf{X}$$

Expanding above (from world to image plane):

$$\begin{pmatrix} {}^c\mathbf{x}' \\ {}^c\mathbf{y}' \\ {}^c\mathbf{z}' \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \end{pmatrix} \begin{pmatrix} X - \mathbf{X}_O \\ Y - \mathbf{Y}_O \\ Z - \mathbf{Z}_O \end{pmatrix}$$

In the E.C., the same equation (called as collinearity equation) is as the following :

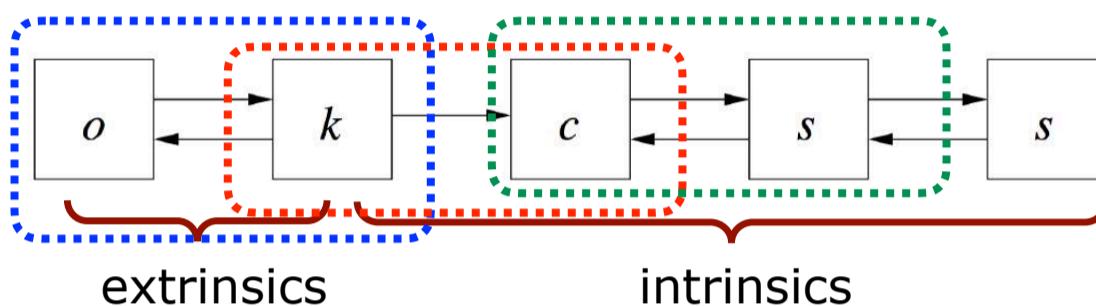
$${}^c\mathbf{x} = c \frac{r_{11}(X - X_O) + r_{12}(Y - Y_O) + r_{13}(Z - Z_O)}{r_{31}(X - X_O) + r_{32}(Y - Y_O) + r_{33}(Z - Z_O)}$$

$${}^c\mathbf{y} = c \frac{r_{21}(X - X_O) + r_{22}(Y - Y_O) + r_{23}(Z - Z_O)}{r_{31}(X - X_O) + r_{32}(Y - Y_O) + r_{33}(Z - Z_O)}$$

This looks more cumbersome in E.C. than the manner in which we derived using H.C.

2. Mapping from the image plane to the sensor coordinate system

The location of the principal point in the image (where the optical axis passes through the image plane), is the point where (0,0) is on the image plane. Then, what we need is which coordinate on the sensor does this point correspond to?



Here, we typically use an affine transformation. We need to compensate for :

- 1) Scale difference in x and y based on the chip design and image plane size
- 2) any shear compensation (largely is 0 for digital cameras)
- 3) and of course the coordinate shift since the origin of the sensor system is not at the principal point

$${}^s\mathbf{H}_c = \begin{bmatrix} 1 & 0 & x_H \\ 0 & 1 & y_H \\ 0 & 0 & 1 \end{bmatrix}$$

Shifts the coordinates

Therefore, we need to consider any scale differences in x & y dimensions between the chip size and image plane size and shear compensation (if any):

$${}^s\mathbf{H}_c = \begin{bmatrix} 1 & s & x_H \\ 0 & 1 + m & y_H \\ 0 & 0 & 1 \end{bmatrix}$$

Here 'm' is the scale difference. For analog cameras, 's' has a value for shear compensation, whereas for digital camera s~0.

So, now we have:

$${}^s \mathbf{x} = {}^s \mathbf{H}_c {}^c \mathbf{K} \mathbf{R} [\mathbf{I}_3] - \mathbf{X}_O \mathbf{X}$$

Now the new calibration matrix K is an affine transformation:

$$\mathbf{K} = {}^s \mathbf{H}_c {}^c \mathbf{K}$$

$$\mathbf{K} = \begin{pmatrix} 1 & s & x_H \\ 0 & 1+m & y_H \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} f & fs & x_H \\ 0 & f(1+m) & y_H \\ 0 & 0 & 1 \end{pmatrix}$$

This contains standard parameters (5):

- constant (f, which is the distance between the camera and the image plane determines the scale factor)
- Principal point (x_H, y_H)
- scale differences (m)
- shear (s)

$${}^s \mathbf{x} = \boxed{{}^s \mathbf{H}_c {}^c \mathbf{K}} \mathbf{R} [\mathbf{I}_3] - \mathbf{X}_O \mathbf{X}$$

$$\mathbf{K} \mathbf{R} [\mathbf{I}_3] - \mathbf{X}_O$$

We have 11 5 parameters 3 parameters 3 parameters parameters:

With 3 from rotation

matrices, 3 from translation (that together form extrinsic parameters) & 5 intrinsic parameters (of the affine transformation).

$$\begin{pmatrix} {}^c x \\ {}^c y \\ {}^c z \end{pmatrix} = \begin{pmatrix} f & fs & x_H \\ 0 & f(1+m) & y_H \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \end{pmatrix} \begin{pmatrix} X - \mathbf{X}_0 \\ Y - \mathbf{Y}_0 \\ Z - \mathbf{Z}_0 \end{pmatrix}$$

\uparrow
 $R^{3D}(\omega, \phi, \kappa) = R_z^{3D}(\kappa)R_y^{3D}(\phi)R_x^{3D}(\omega)$

$$R_x^{3D}(\omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{bmatrix} \quad R_y^{3D}(\phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

$$R_z^{3D}(\kappa) = \begin{bmatrix} \cos(\kappa) & -\sin(\kappa) & 0 \\ \sin(\kappa) & \cos(\kappa) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This type of transformation is called “Direct linear transformation” (DLT). This is used for mapping the coordinates from world to sensor using an affine transformation with ideal projection mapping. The DLT is used for mapping an affine camera. This also means that the entire mapping is done considering only linear errors.

So, the 3×4 matrix is given by:

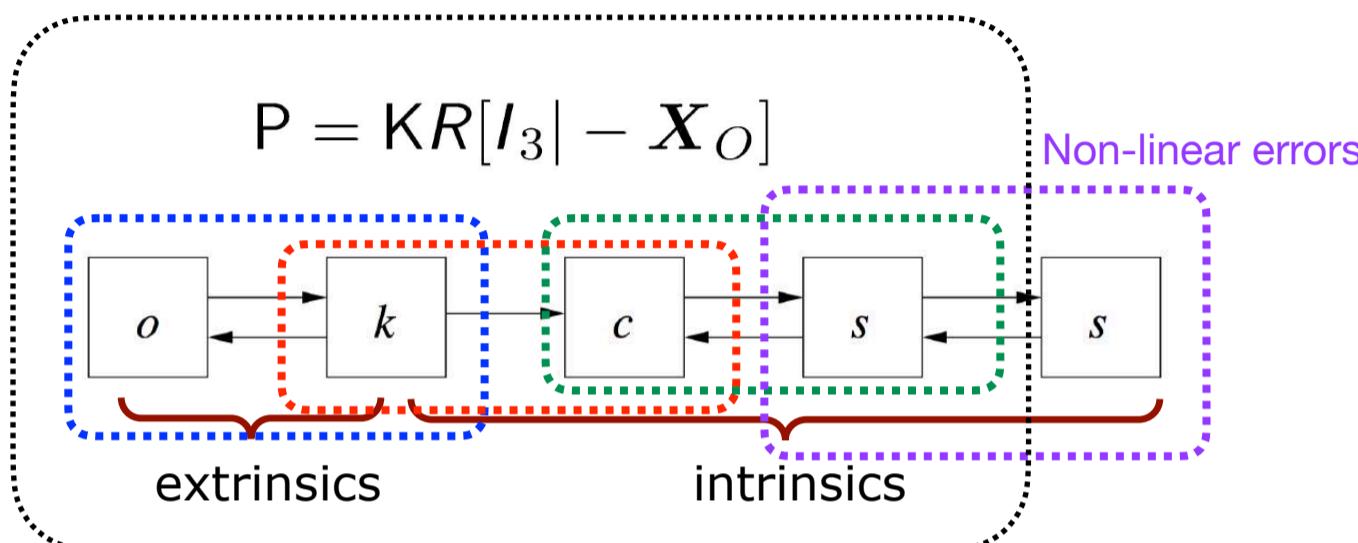
$$\mathsf{P} = \mathsf{K}R[\mathbf{I}_3] - \mathbf{X}_O \quad P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix}$$

In the homogeneous matrix P , the last row is taken by the scale factor within the homogeneous coordinate system. So, the scale factor can take any value and it would still define the same object upto a certain scale factor.

In E.C. expressed as:

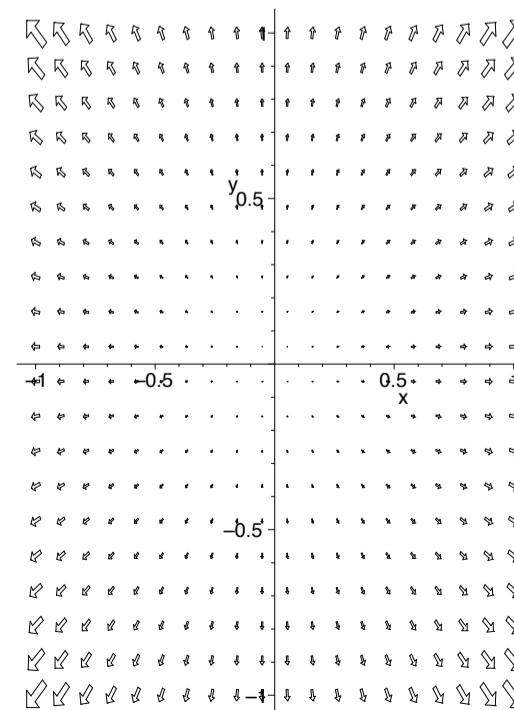
$$\begin{aligned} {}^s x &= \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \\ {}^s y &= \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \end{aligned}$$

3. Compensation for the non-linear errors/distortions



There are non-linear errors throughout in this process. However, we assumed so far that all the errors were linear and it is only in the end that we take care of the non-linear errors. This compensation is done within the sensor system itself. These can arise already in the beginning when the image was captured by the camera, eg: lens distortions. Here we have modelled an affine camera, that are known to preserve straight lines (i.e. straight lines remain straight lines). For straight line distortions within the image, the affine camera model compensates for such errors at this stage of mapping.

There could be radial distortions



In all, this requires a non-linear mapping, i.e. each coordinate within the sensor system is required to be treated un-equally. This also can be understood as “location-dependent shift in the sensor coordinate system”. For example, for the radial distortions, each pixel will have different sensitivity towards the distortion depending upon its coordinate location within the sensor coordinate system.

If we are using an affine camera, and therefore we know that straight lines are preserved, then if for example, straight lines appear curved, these can be compensated.

Let $(^s x, ^s y)$ be the coordinate from the sensor system that need to be compensated for non-linear error. So, the mapping is as follows:

$$^a x = ^s x + \Delta x(q, r)$$

$$^a y = ^s y + \Delta y(q, r)$$

$$^a \mathbf{x} = {}^a \mathbf{H}_s(\mathbf{x}) {}^s \mathbf{x} \quad \text{where, } {}^a \mathbf{x} = (^a x, ^a y), {}^s \mathbf{x} = (^s x, ^s y)$$

$${}^a H_s(x) = {}^a H_s(q, r) = \begin{pmatrix} 1 & 0 & \Delta x(q, r) \\ 0 & 1 & \Delta y(q, r) \\ 0 & 0 & 1 \end{pmatrix}$$

Our new calibration matrix: ${}^a \mathbf{x} = {}^a \mathbf{H}_s(\mathbf{x}) K R [I_3] - \mathbf{X}_O \mathbf{X}$

$${}^a H_s(q, r) \quad K \quad = \quad {}^a K(q, r)$$

$$\begin{pmatrix} 1 & 0 & \Delta x(q, r) \\ 0 & 1 & \Delta y(q, r) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & fs & x_H \\ 0 & f(1+m) & y_H \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} f & fs & x_H + \Delta x(q, r) \\ 0 & f(1+m) & y_H + \Delta y(q, r) \\ 0 & 0 & 1 \end{pmatrix}$$

The calibration matrix is the model for an affine camera that also compensates for non-linear errors.

camera calibration matrix #parameters

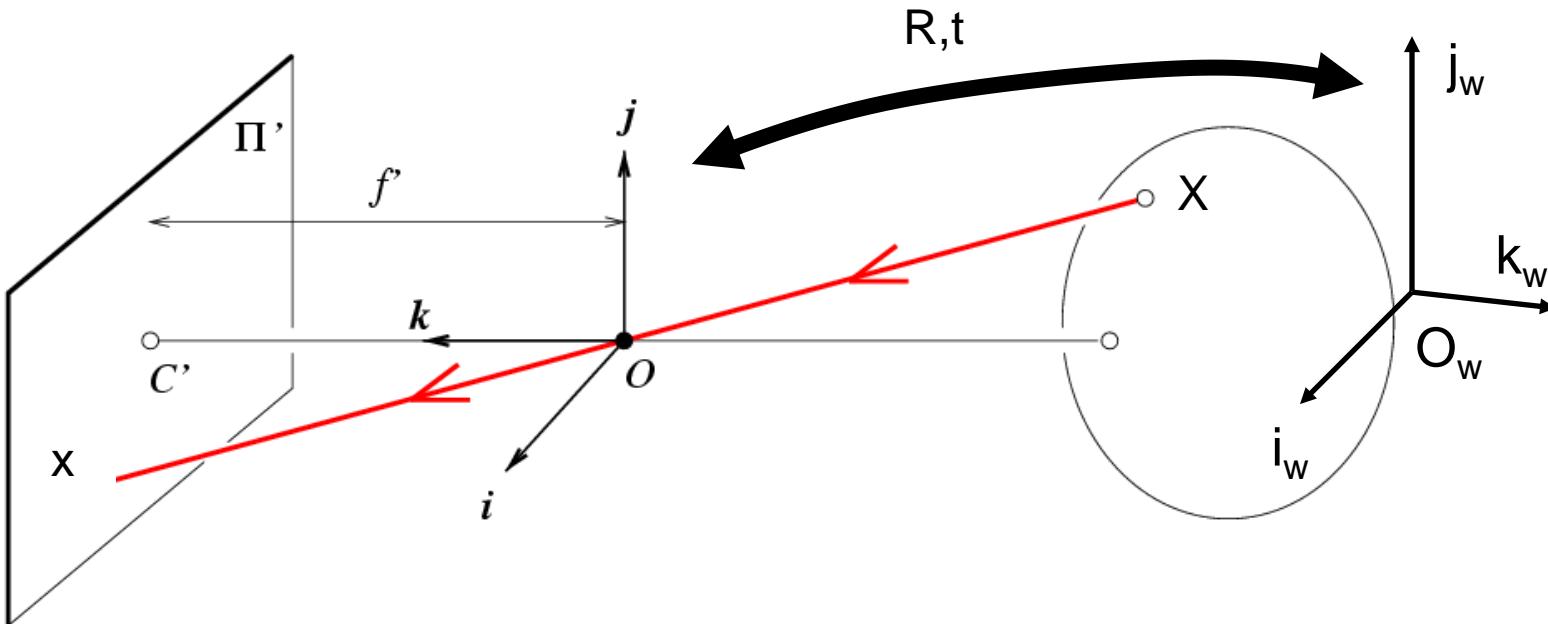
unit	${}^0\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	6 (6+0)	<ul style="list-style-type: none"> • image plane origin is principal point and distance between camera origin and the image plane is 1.
ideal	${}^k\mathbf{K} = \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$	7 (6+1)	<ul style="list-style-type: none"> • image plane origin is principal point
Euclidian	${}^p\mathbf{K} = \begin{bmatrix} c & 0 & x_H \\ 0 & c & y_H \\ 0 & 0 & 1 \end{bmatrix}$	9 (6+3)	<ul style="list-style-type: none"> • Euclidean sensor in the image plane
affine	$\mathbf{K} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}$	11 (6+5)	<ul style="list-style-type: none"> • preserves straight lines
general	${}^a\mathbf{K} = \begin{bmatrix} c & cs & x_H + \Delta x \\ 0 & c(1+m) & y_H + \Delta y \\ 0 & 0 & 1 \end{bmatrix}$	11+N	<ul style="list-style-type: none"> • camera with non-linear distortions

Pinhole camera model and camera projection matrix

DSE312-LECTURE7

BHAVNA R

Camera (projection) matrix



$$\mathbf{x} = \mathbf{K} [\mathbf{R} \quad \mathbf{t}] \mathbf{X}$$

Extrinsic Matrix

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

"Forward imaging model"

1. A single 3D point 'X' in the world coordinate frame is mapped into the image plane.
2. Coordinate transformation from 3D world to 3D camera coordinates.
3. Transform 3D camera coordinates using "Perspective projection" to obtain 2D image coordinates ('x').

x: Image Coordinates: (u,v,1)

K: Intrinsic Matrix (3x3)

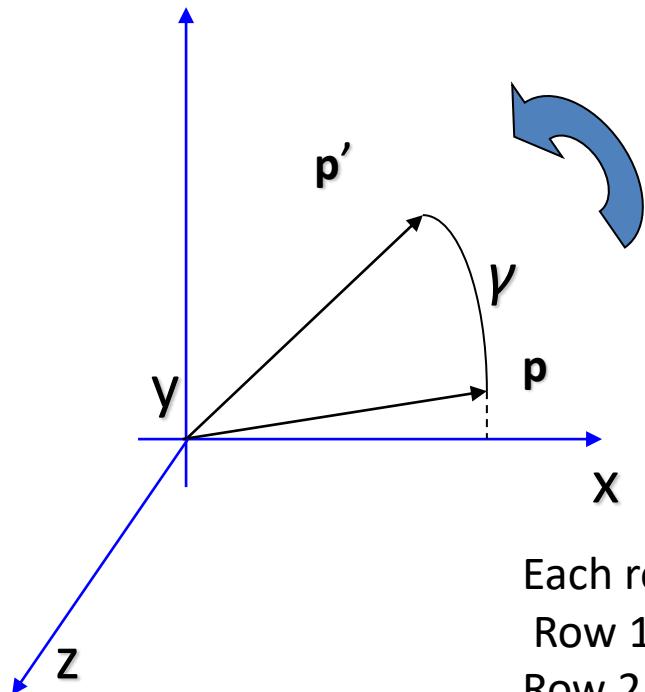
R: Rotation (3x3)

t: Translation (3x1)

X: World Coordinates: (X,Y,Z,1)

3D Rotation of Points

Rotation around the coordinate axes, **counter-clockwise**:



$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$
$$R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$
$$R_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Each row in \mathbf{R} corresponds to the direction in the camera coordinate frame C (x_c, y_c, z_c).

Row 1 of \mathbf{R} [$r_{11} \ r_{12} \ r_{13}$] is the direction of x_c in the world coordinate frame..

Row 2 of \mathbf{R} [$r_{21} \ r_{22} \ r_{23}$] corresponds to direction of y_c in the world coordinate frame

Row 3 of \mathbf{R} [$r_{31} \ r_{32} \ r_{33}$] direction of z_c in the world coordinate frame

Here \mathbf{R} is an orthonormal matrix i.e. a square matrix whose row (or column) vectors are orthonormal $\mathbf{R}^{-1} = \mathbf{R}^T$, $\mathbf{R}^T \mathbf{R} = \mathbf{R} \mathbf{R}^T = \mathbf{I}$

vectors (\mathbf{p}, \mathbf{q}) are orthonormal if, $\text{dot}(\mathbf{p}, \mathbf{q}) = \mathbf{p}^T \mathbf{q} = 0$; $\mathbf{p}^T \mathbf{p} = \mathbf{q}^T \mathbf{q} = 1$

Perspective Projection: properties

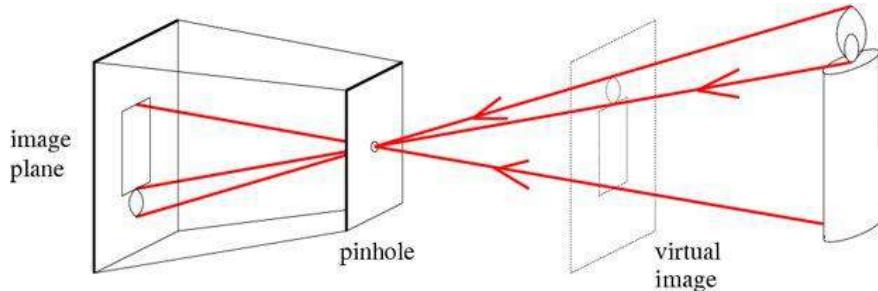
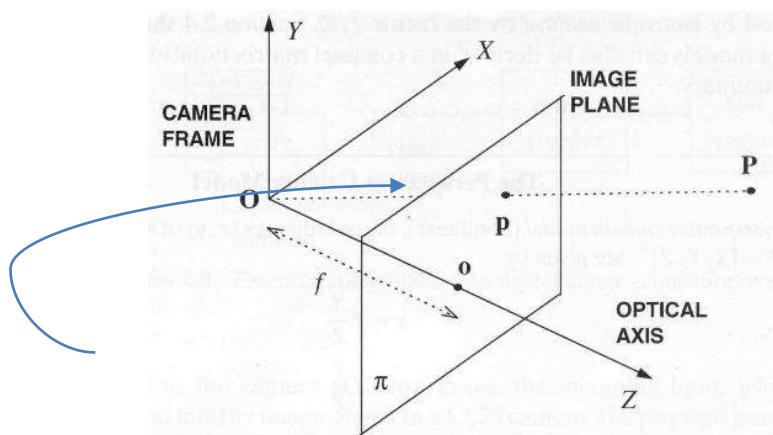
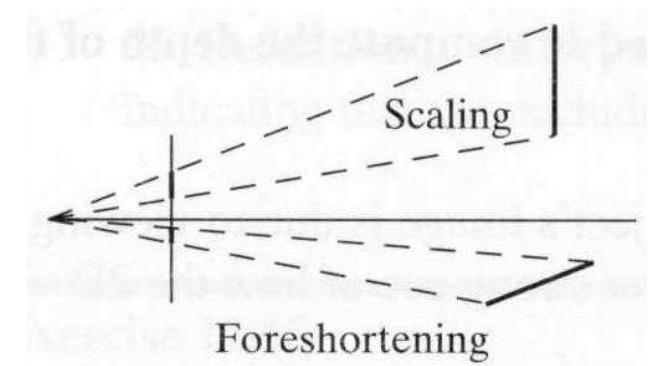


Image plane lines in front of the camera frame

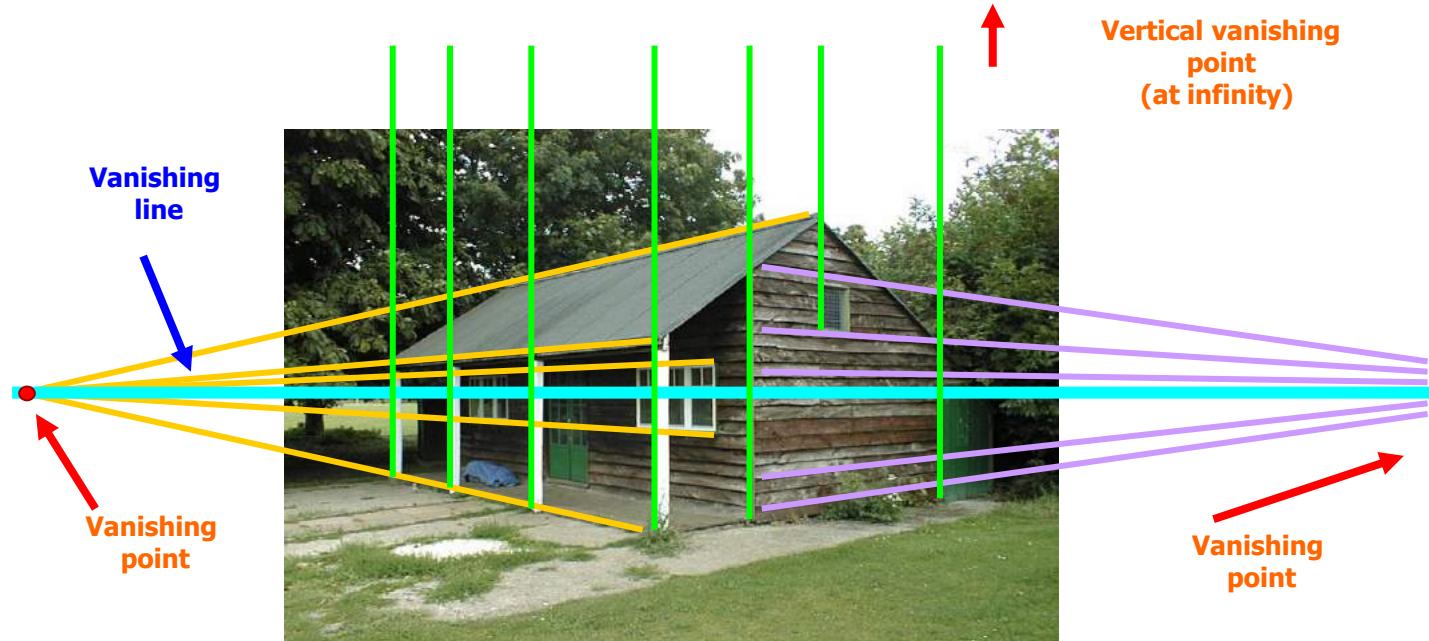


- When a line (or surface) is parallel to the image plane, the effect of perspective projection is scaling.
- When a line (or surface) is not parallel to the image plane: foreshortening to describe the projective distortion (i.e., the dimension parallel to the optical axis is compressed relative to the frontal dimension).

- The projection of a point is not unique (any point on the line OP has the same projection), it's a many to one mapping.
- The distance to an object is inversely proportional to its image size.
- Focal length f (larger/smaller) is inverse of field of view (smaller/larger)
- Lines in 3D project to lines in 2D.
- Distances and angles are not preserved.
- Parallel lines do not in general project to parallel lines (unless they are parallel to the image plane).



Vanishing point & vanishing line



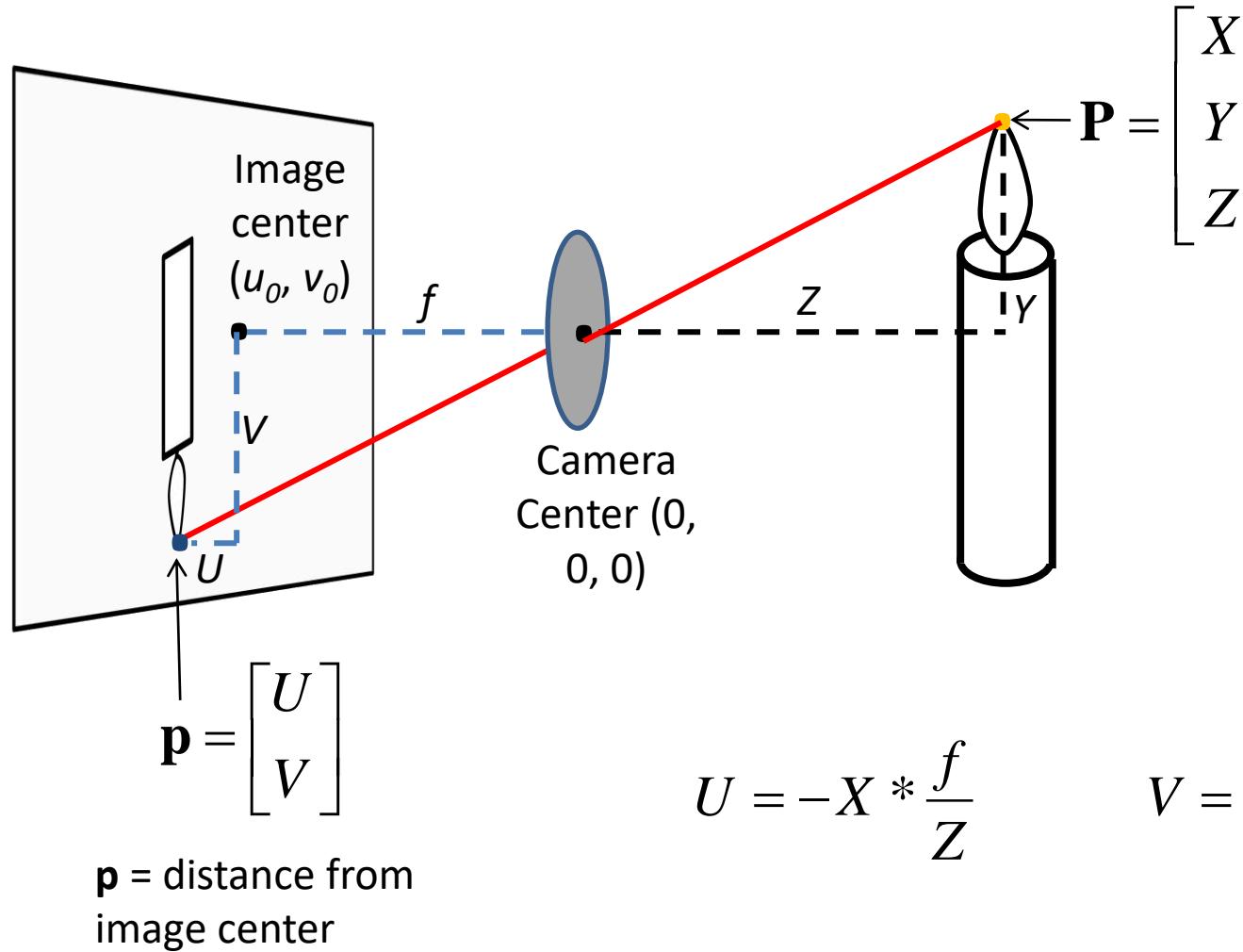
Vanishing point

- parallel lines in space project perceptively onto lines that on extension intersect at a single point in the image plane called *vanishing point* or *point at infinity*.
- the vanishing point of a line depends on the orientation of the line and not on the position of the line.
- the vanishing point of any given line in space is located at the point in the image where a parallel line through the center of projection intersects the image plane.

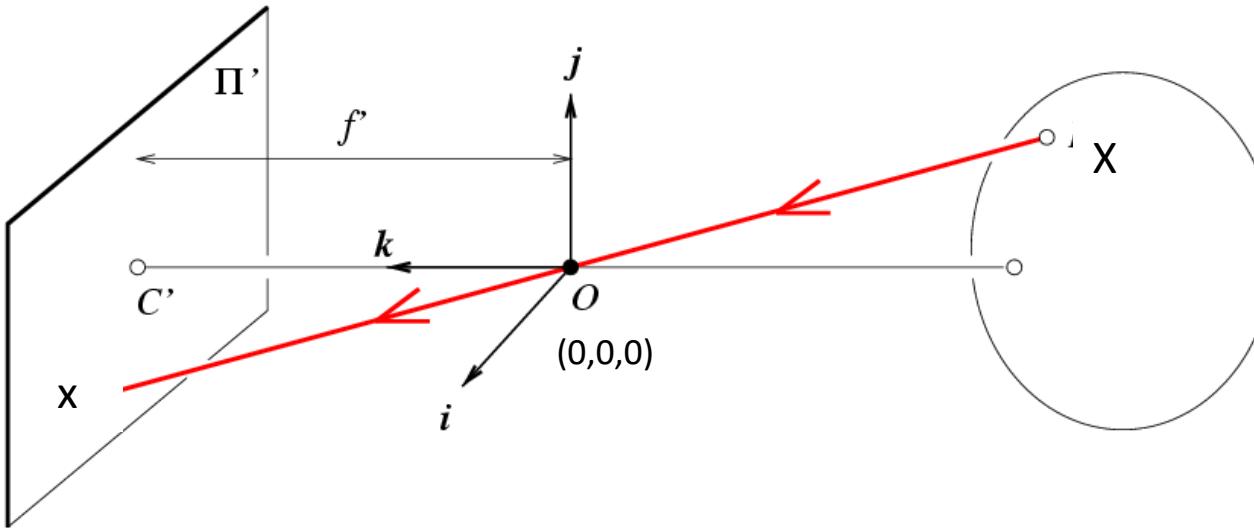
Vanishing line

- the vanishing points of all the lines that lie on the same plane form the *vanishing line*.
- the intersection of a parallel plane through the center of projection with the image plane.

Projection: world coordinates \rightarrow image coordinates



Projection matrix and its parameters



Intrinsic Assumptions

- Unit aspect ratio
- Optical center at (0,0)
- No skew

Extrinsic Assumptions

- No rotation, no translation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K} [\mathbf{I} \quad \mathbf{0}] \mathbf{X} \rightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Remove assumption: known optical center

Intrinsic Assumptions

- Unit aspect ratio
- No skew

Extrinsic Assumptions

- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K} [\mathbf{I} \quad \mathbf{0}] \mathbf{X} \quad \xrightarrow{\text{w}} \quad w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Remove assumption: equal aspect ratio

Intrinsic Assumptions

- No skew

Extrinsic Assumptions

- No rotation
- Camera at (0,0,0)

$$\mathbf{x} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \mathbf{X} \rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x x + z u_0 \\ f_y y + z v_0 \\ z \\ 1 \end{bmatrix}$$

Perspective projection equation

$$u = f_x \frac{x}{z} + u_0, v = f_y \frac{y}{z} + v_0$$

We typically use f , for focal length.

f_x & f_y are the effective focal lengths in x & y

u_0 & v_0 are the principal points on the image plane

Remove assumption: non-skewed pixels

Intrinsic Assumptions

$$\mathbf{x} = \mathbf{K}[\mathbf{I} \quad \mathbf{0}] \mathbf{X} \rightarrow$$

Extrinsic Assumptions

- No rotation
- Camera at (0,0,0)

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Camera rotation, translation (extrinsic) & perspective projection (intrinsic)

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}] \mathbf{X}$$



$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

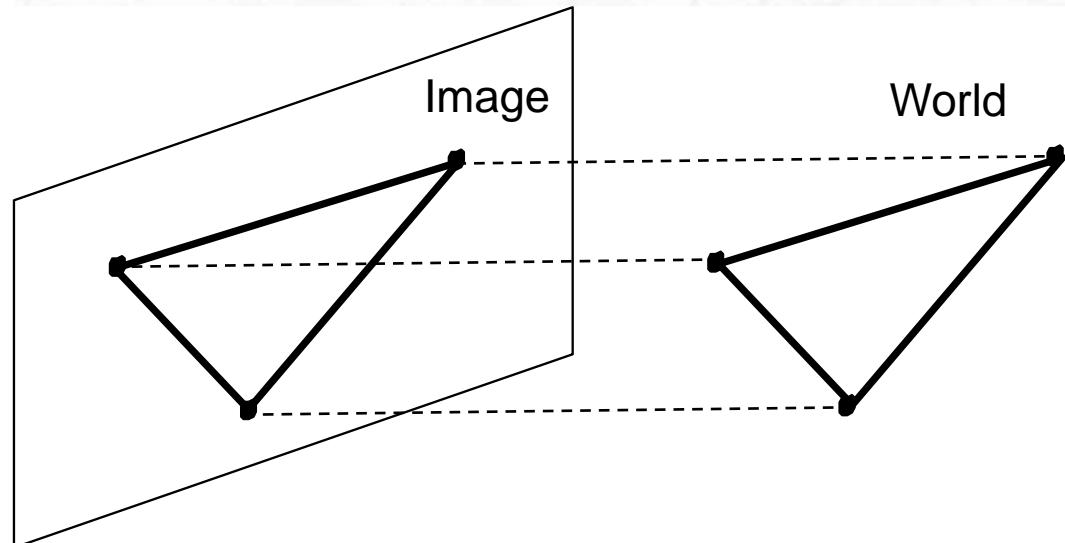
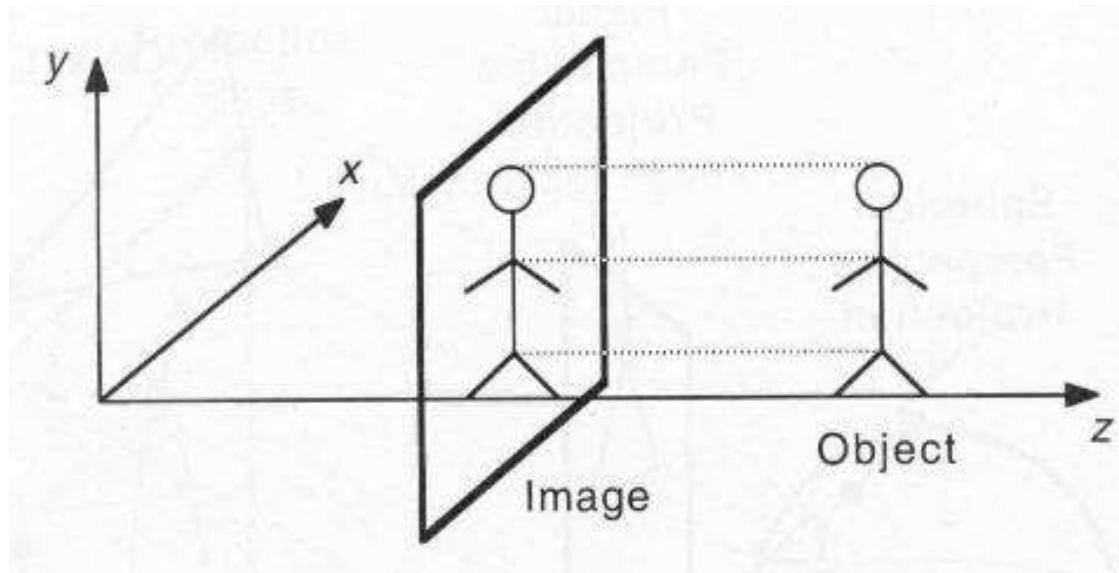
$$\mathbf{x} = \mathbf{K}R[I_3] - \mathbf{X}_O \mathbf{X}$$



$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix}$$

Variations (presumptions) in projection matrix parameters

Orthographic Projection



- Special case of perspective projection
- (Also called “parallel projection”)
- projection of a 3D object onto a plane by a set of parallel rays orthogonal to the image plane
- Distance from the COP to the image plane is infinite
- the limit of perspective projection as $f \rightarrow \infty$
(i.e., $f/Z \approx 1$)
- Parallel lines project to parallel lines.
- Size does not change with distance from the camera.

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

DLT Mapping using projection matrix \mathbf{M}

$$\mathbf{x} = KR[I_3] - \mathbf{X}_O \mathbf{X}$$

$$\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}] \mathbf{X}$$

$$\mathbf{x} = \mathbf{MX}$$

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$3 \times 1 \qquad \qquad \qquad 3 \times 4 \qquad \qquad \qquad 4 \times 1$

Given the points \mathbf{x} and \mathbf{X} , determine the Projection matrix

Linear least-squares regression!

Least squares line fitting

- Data: $(x_1, y_1), \dots, (x_n, y_n)$

- Line equation: $y_i = mx_i + b$

- Find (m, b) to minimize

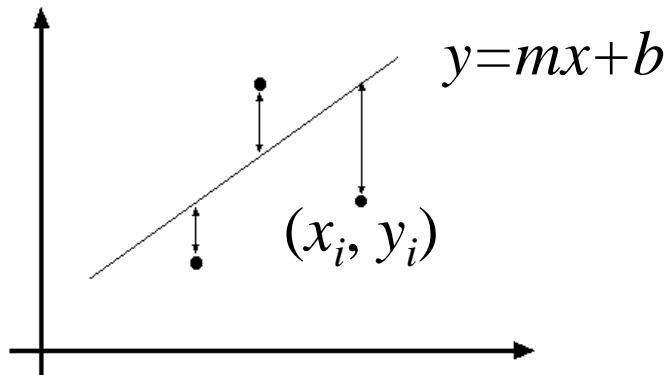
- Loss function E ,
$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$

- To minimize E , we compute

$$\frac{dE}{dp} = 0$$

$$E = \sum_{i=1}^n \left(\begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$
$$= \mathbf{y}^T \mathbf{y} - 2(\mathbf{Ap})^T \mathbf{y} + (\mathbf{Ap})^T (\mathbf{Ap})$$

$$\frac{dE}{dp} = 2\mathbf{A}^T \mathbf{Ap} - 2\mathbf{A}^T \mathbf{y} = 0$$



Matlab: $\mathbf{p} = \mathbf{A} \setminus \mathbf{y};$

Python:

`p = np.linalg.lstsq(A, y)[0]`

$$\mathbf{A}^T \mathbf{Ap} = \mathbf{A}^T \mathbf{y} \Rightarrow \mathbf{p} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \quad (\text{Closed form solution})$$

Solving for a System of Linear Equations

$$Ax = b$$

- ❖ A is a square matrix with full rank: Best-case scenario, several methods & has a unique solution
 - Gauss elimination
 - Inversion of : A with $x = A^{-1}b$
 - Cholesky decomposition $\text{chol}(A) = LL^T$ with lower triangular matrix L and solving and for $Ly = b$
 - and $L^T x = y$
 - QR decomposition: A is a square matrix and has unique solution
 - Conjugate gradients: iterative approach, start with a guess value (for x_0) & minimize $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$.

- ❖ A is **over**determined: Common real-world situation (minimize the error): $x^* = \arg \min_x \|Ax - b\|$ using ordinary least squares approach, solve

$$x = (A^T A)^{-1} A^T b$$

- ❖ A is **under**determined: Infinitively many solutions exist: find x that minimizes $\|x\|$

$$x = A^T (AA^T)^{-1} b$$

Homogeneous system $Ax = 0$

Find a solution $x \neq 0$ fulfilling $Ax = 0$

Means system is underdetermined

There exists a null space of A called $\text{null}(A)$ and all x fulfilling $Ax = 0$ are elements of it

A 's rank deficiency defines the dimensionality of the null space

For each Eigenvector ν holds $A\nu = \lambda\nu$

Thus, for those with Eigenvalue 0 we have $A\nu = 0\nu = 0$

all Eigenvectors corresponding to an Eigenvalue of 0 solve $Ax = 0$

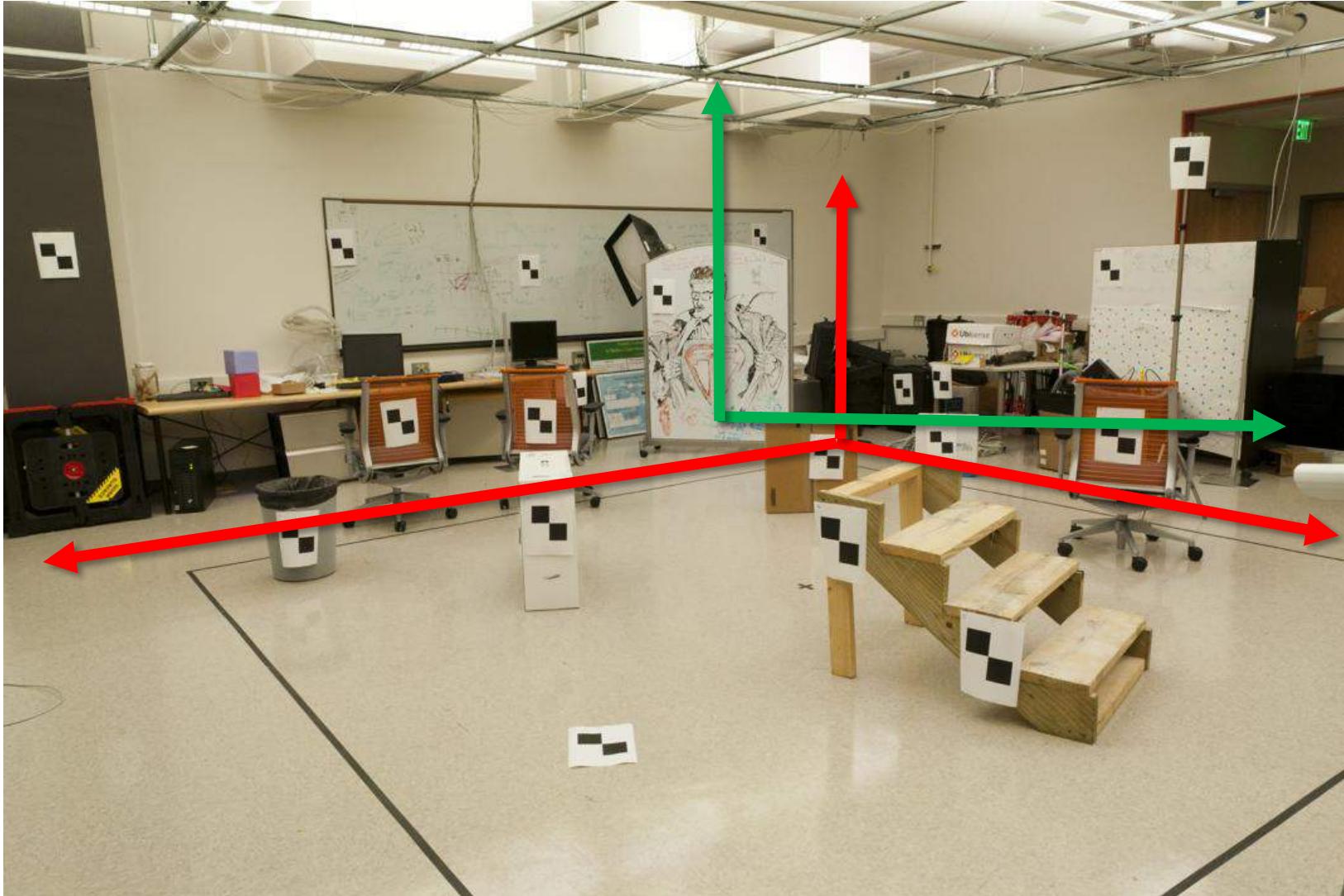
SVD decomposes a matrix A into

$$A = UDV^T$$

$$\begin{matrix} A \\ M \times N \end{matrix} = \begin{matrix} U \\ M \times M \end{matrix} \begin{matrix} D \\ M \times N \end{matrix} \begin{matrix} V^T \\ N \times N \end{matrix}$$

- D is a diagonal matrix of singular values sorted from large to small
- V^T stores the corresponding singular vectors to the values
- U, V are orthogonal matrices
- We check the last element of D , if $D_{MN} = 0$,
- Then last column of V is a non-trivial solution x for $Ax = 0$
- The last column of V represents the vector that minimizes $\|Ax\|$ under the constraint $\|x\|=1$

World vs Camera coordinates



James Hays

Spatial resection and camera calibration

Spatial Resection

- Given a calibrated camera, (intrinsic parameter, \mathbf{K} is known)
- 6 unknown parameters (extrinsic)
- Require at least 3 points
- Problem solved by spatial resection

Determine **Projection matrix \mathbf{M}** and then extrinsic parameters \mathbf{R}, \mathbf{t}

Camera calibration

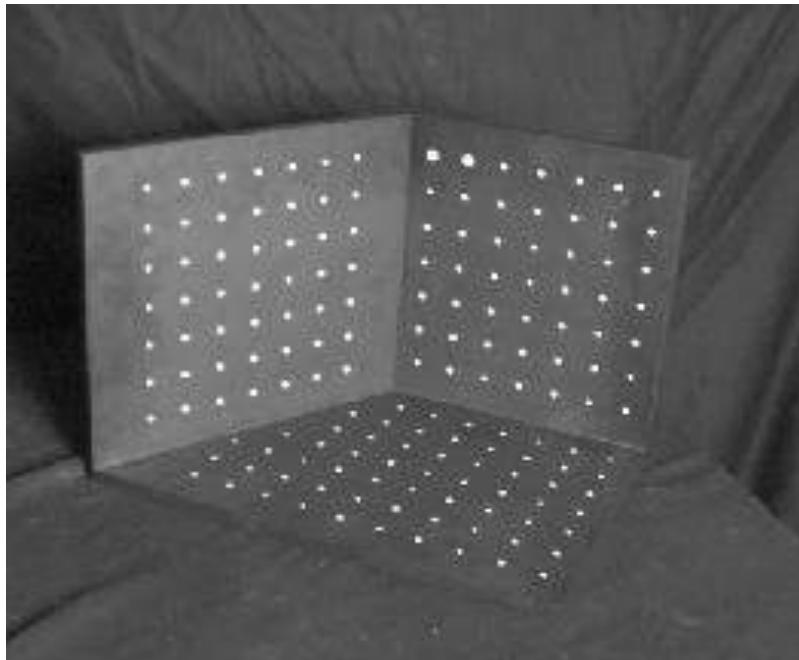
- Given an Uncalibrated camera
- 11 unknowns
- We need at least 6 points
- Assuming the model of an affine camera
- Problem solved by DLT

Determine **Projection matrix \mathbf{M}** and then the intrinsic and extrinsic parameters ($\mathbf{K}, \mathbf{R}, \mathbf{t}$)

Calibrating the Camera

Use a scene with **known** geometry

- Correspond image points to 3d points
- Get least squares solution (or non-linear solution)

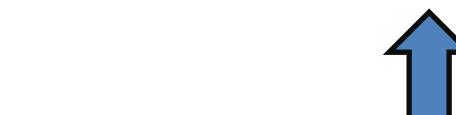


Known 2d
image coords



$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d
world locations



Unknown Camera Parameters

How many points do I need to fit the model?

$$\mathbf{x} = \mathbf{M}\mathbf{X}$$



$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

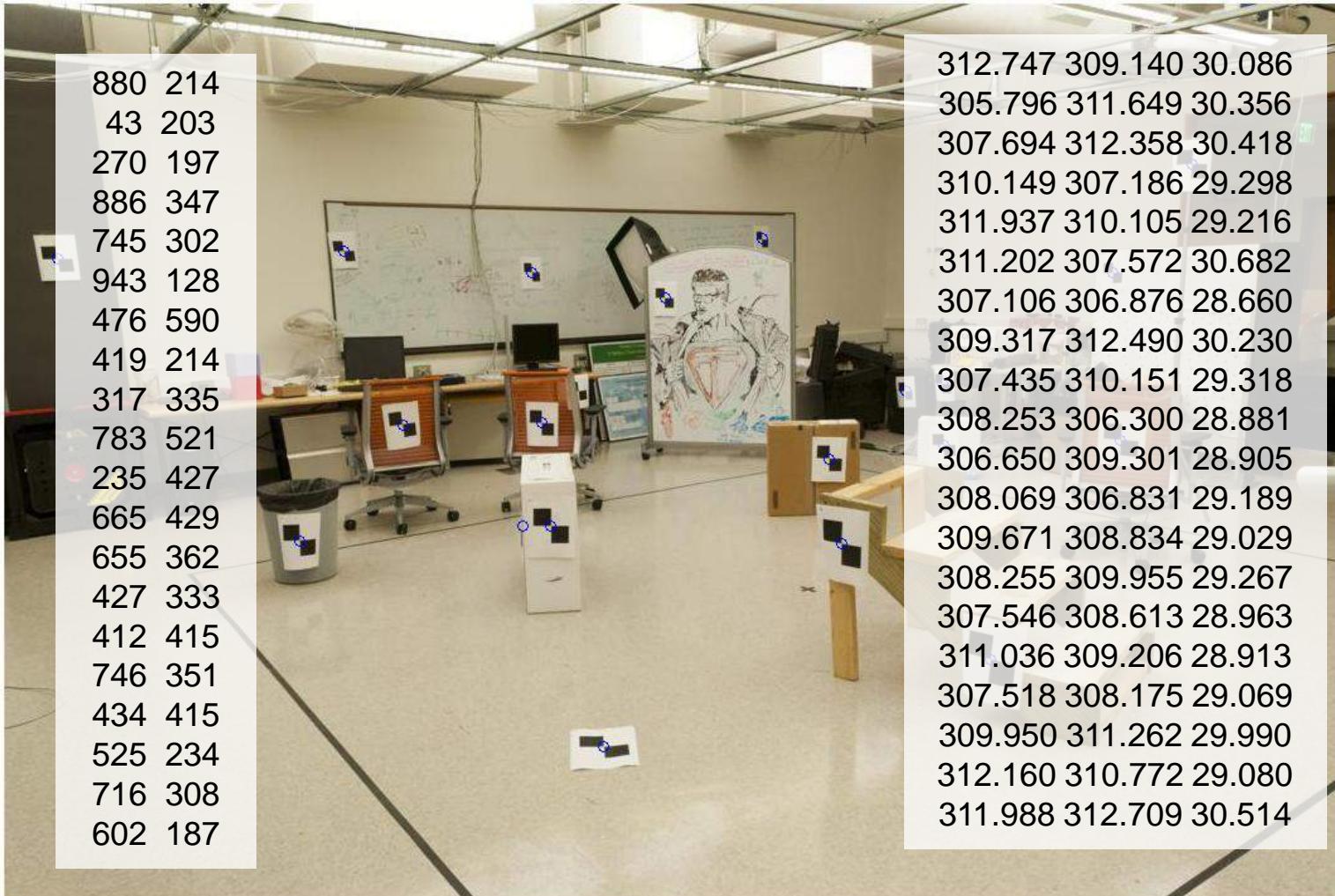
\mathbf{M} is 3x4, so 12 unknowns, but projective scale ambiguity – 11 deg. freedom.
One equation per unknown -> 5 1/2 point correspondences determines a solution
(e.g., either u or v).

More than 5 1/2 point correspondences -> overdetermined, many solutions to \mathbf{M} .
Least squares is finding the solution that best satisfies the overdetermined system.

Why use more than 6? Robustness to error in feature points.

How do we calibrate a camera?

Known 2d
image coords

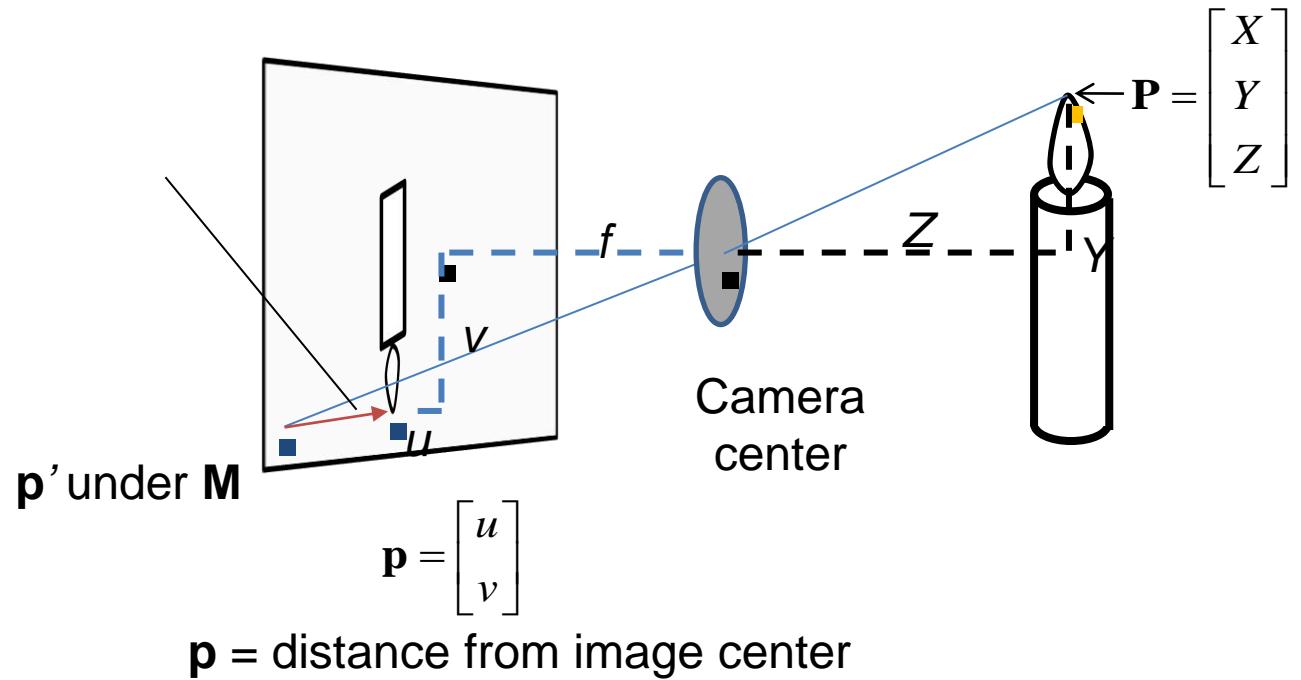


Known 3d
world locations

least squares fitting

What is least squares doing?

Minimize Error between \mathbf{M} estimate
and known projection point



- Given 3D point evidence, find best \mathbf{M} which minimizes error between estimate (p') and known corresponding 2D points (\mathbf{p}).
- Best \mathbf{M} occurs when $p' = \mathbf{p}$, or when $p' - \mathbf{p} = 0$
- Form these equations from all point evidence
- Solve for model via closed-form regression

Unknown Camera Parameters

Known 2d
image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$


Known 3d
locations

First, work out
where X,Y,Z
projects to under
candidate **M**.

$$su = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$sv = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$s = m_{31}X + m_{32}Y + m_{33}Z + m_{34}$$

Two equations
per 3D point
correspondence

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

- Each point gives two observation equations: one for each image coordinate
- Data association between points is known

Unknown Camera Parameters

Known 2d
image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d
locations

Next, rearrange into form
where all **M** coefficients are
individually stated in terms
of X,Y,Z,u,v.

-> Allows us to form lsq
matrix.

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34})v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

Unknown Camera Parameters

↓

$$\begin{matrix} \text{Known 2d} \\ \text{image coords} \end{matrix} \left[\begin{array}{c} su \\ sv \\ s \end{array} \right] = \left[\begin{array}{cccc} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{array} \right] \left[\begin{array}{c} X \\ Y \\ Z \\ 1 \end{array} \right] \begin{matrix} \text{Known 3d} \\ \text{locations} \end{matrix}$$

Next, rearrange into form where all **M** coefficients are individually stated in terms of X,Y,Z,u,v.

Allows us to form lsq matrix.

$$m_{31}uX + m_{32}uY + m_{33}uZ + m_{34}u = m_{11}X + m_{12}Y + m_{13}Z + m_{14}$$

$$m_{31}vX + m_{32}vY + m_{33}vZ + m_{34}v = m_{21}X + m_{22}Y + m_{23}Z + m_{24}$$

$$0 = m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}uX - m_{32}uY - m_{33}uZ - m_{34}u$$

$$0 = m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}vX - m_{32}vY - m_{33}vZ - m_{34}v$$

Estimate the 11/12 elements of the Projection Matrix M

Known 2d image coords

2nd point

880 214
43 203
270 197
886 347
745 302
943 128
476 590
419 214
317 335

(u_2, v_2)



Known 3d world locations

312.747 309.140 30.086 1
0 0 0 0 312.747 309.140 30.086 1
305.796 311.649 30.356 1 0 0 0 -43×305.796
0 0 0 0 305.796 311.649 30.356 1 -203×305.796
⋮
$X_n \quad Y_n \quad Z_n \quad 1_n \quad 0 \quad 0 \quad 0 \quad -u_n X_n \quad -u_n Y_n \quad -u_n Z_n \quad -u_n$
0 0 0 0 $X_n \quad Y_n \quad Z_n \quad 1 \quad -v_n X_n \quad -v_n Y_n \quad -v_n Z_n \quad -v_n$

(X_2, Y_2, Z_2)

....

....

Projection error defined by two equations – one for u and one for v

$$\begin{bmatrix}
 312.747 & 309.140 & 30.086 & 1 & 0 & 0 & 0 & -880 \times 312.747 & -880 \times 309.140 & -880 \times 30.086 & -880 \\
 0 & 0 & 0 & 0 & 312.747 & 309.140 & 30.086 & 1 & -214 \times 312.747 & -214 \times 309.140 & -214 \times 30.086 & -214 \\
 305.796 & 311.649 & 30.356 & 1 & 0 & 0 & 0 & -43 \times 305.796 & -43 \times 311.649 & -43 \times 30.356 & -43 \\
 0 & 0 & 0 & 0 & 305.796 & 311.649 & 30.356 & 1 & -203 \times 305.796 & -203 \times 311.649 & -43 \times 30.356 & -203 \\
 & & & & & & & \vdots & & & \\
 X_n & Y_n & Z_n & 1_n & 0 & 0 & 0 & -u_n X_n & -u_n Y_n & -u_n Z_n & -u_n \\
 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_n X_n & -v_n Y_n & -v_n Z_n & -v_n
 \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix}$$

Solution1 : Unknown Camera Parameters

Known 2d
image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d locations



The projection matrix is in homogeneous coordinates

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, s \neq 0$$

Therefore $M \equiv sM$.

Implies: Simultaneously scaling the world and the camera does not change the image.

set $m_{34}=1$

- Finally, solve for m's entries using linear least squares
- Method 1 – $\mathbf{Ax}=\mathbf{b}$ form
- To set scale, we artificially set m_{34} to 1

A

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 \\ & & & & \vdots & & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n \end{bmatrix}$$

$$\begin{bmatrix} x \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_n \\ v_n \\ \vdots \\ u_n \\ v_n \\ \vdots \\ u_n \\ v_n \\ \vdots \end{bmatrix}$$

MATLAB:

```
M = A\b;
M = [M;1];
M = reshape(M,[],3)';
```

Python Numpy:

```
M = np.linalg.lstsq(A,b)[0];
M = np.append(M,1)
M = np.reshape(M, (3,4))
```

Note – you will see this form called ‘inhomogeneous’ linear system -> nothing to do with homogeneous coordinates

Note: Must reshape M afterwards!

Solution 2 : Unknown Camera Parameters

Known 2d
image coords

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Known 3d
locations

Note – you will see this form called ‘homogeneous’ linear system -> nothing to do with homogeneous coordinates

- Or, solve for m’s entries using total linear least-squares, set the scale s.t. $\|m\|^2 = 1$
- Method 2 – **Ax=0** form
 - Find non-trivial solution (not A=0) $\mathbf{p}' - \mathbf{p} = 0$
- SVD – singular value decomposition
- Computes pseudo-inverse

A

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & \vdots & & & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix}$$

$$= \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

MATLAB:

```
[U, S, V] = svd(A);
M = V(:, end);
M = reshape(M, [], 3)';
```

Python Numpy:

```
U, S, Vh = np.linalg.svd(a)
# V = Vh.T
M = Vh[-1, :]
M = np.reshape(M, (3, 4))
```

- Solving a system of linear equations of the form is equivalent to finding the null space of A using SVD
- Choose as the singular vector belonging to the singular value of 0
- In reality, the singular vector belonging to the smallest singular value minimizes equation

Under what conditions do we not recover the Projection matrix?

- M is of rank 11, if
- Number of points ≥ 6
- Assumption: no gross errors
- **No solution, if all points X_i are located on a plane**
- Solution is unstable the control points lie approximately on a plane

$$\begin{aligned} M &= \begin{bmatrix} \dots \\ a_{x_i}^T \\ a_{y_i}^T \\ \dots \end{bmatrix} \\ &= \begin{bmatrix} -X_i & -Y_i & -Z_i & -1 & 0 & 0 & 0 & \dots & 0 & 0 & x_i X_i & x_i Y_i & x_i Z_i & x_i \\ 0 & 0 & 0 & 0 & -X_i & -Y_i & -Z_i & -1 & y_i X_i & y_i Y_i & y_i Z_i & y_i & y_i \end{bmatrix} \\ &= \begin{bmatrix} -X_i & -Y_i & 0 & -1 & 0 & 0 & \dots & 0 & 0 & x_i X_i & x_i Y_i & 0 & x_i \\ 0 & 0 & 0 & 0 & -X_i & -Y_i & -Z_i & -1 & y_i X_i & y_i Y_i & y_i Z_i & y_i \end{bmatrix} \end{aligned}$$

e.g., assume all $Z_i = 0$

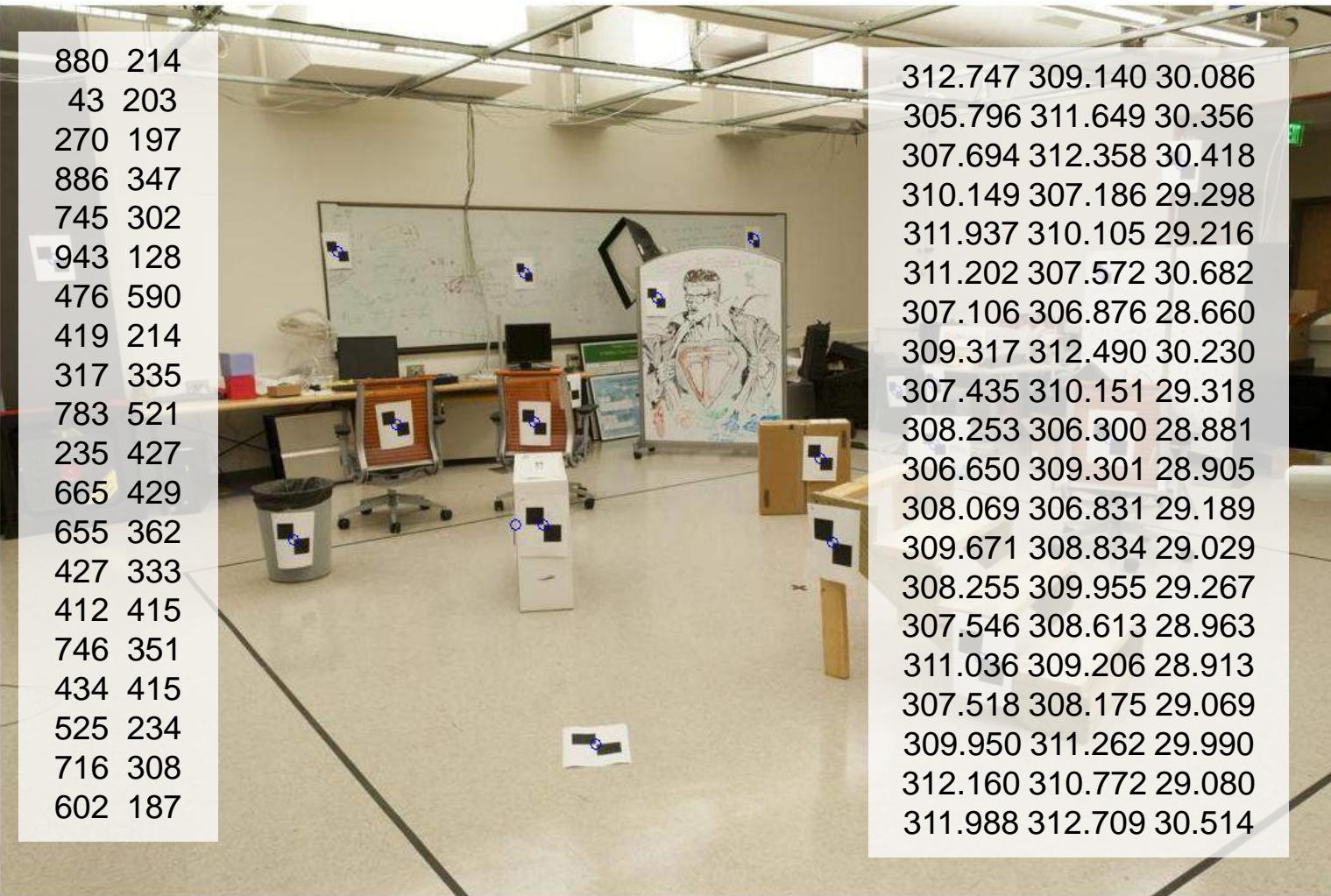
3D reference object based camera calibration procedure

- We capture image of an object of known geometry in 3D space
- calibration object consists of two or three planes orthogonal to each other
- Take several images to generate the corresponding points in the image plane
- identify correspondences between 3D scene points and image plane



Homework 1: Determine the projection matrix

Known 2d image coords



Homework1:

- i) Given the number 20 points of correspondences, between world and image, determine the projection matrix using the 2 methods.
- ii) By taking only 10 pairs of those data associations, solve the same problem using the two methods.

Homework 2:

Given the projection matrix M, compute the image plane coordinate of a point at the world coordinate (4,0,0).

$$M = \begin{bmatrix} 512 & -800 & 0 & 800 \\ 512 & 0 & -800 & 1600 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Demo – Kyle Simek

“Dissecting the Camera Matrix”

Three-part blog series

- <http://ksimek.github.io/2012/08/14/decompose/>
- <http://ksimek.github.io/2012/08/22/extrinsic/>
- <http://ksimek.github.io/2013/08/13/intrinsic/>

“Perspective toy”

- http://ksimek.github.io/perspective_camera_toy.html

Next Lecture:
Decomposition of the projection matrix to recover
intrinsic & extrinsic parameters

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-314

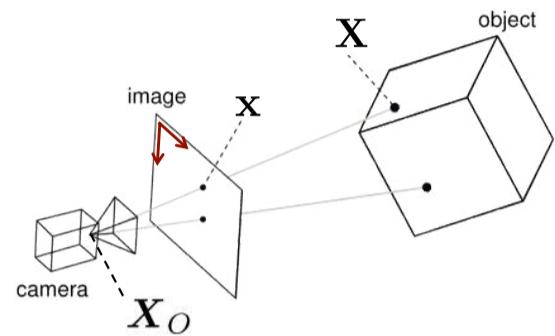
Decomposition of the transformation matrix

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Decomposition of the transformation matrix

3D Point to 2D Pixels: Estimating the Parameters of the transformation matrix

The last lecture, we learnt to deduce the transformation matrix by using the Direct linear transform (DLT) mapping for any object point from world coordinate to the image point coordinates. We required a set of points (minimum set of 6 points) to solve the system of linear equations.



Our camera parameters are :

Intrinsic: Camera-internal parameters

Extrinsic: Pose parameters of the camera

The matrix consists both Intrinsic and extrinsic represented as a single matrix with 11 parameters.

$$\underset{3 \times 1}{\mathbf{x}} = \underset{3 \times 3}{K} \underset{3 \times 3}{R} \underbrace{\left[\begin{matrix} I_3 \\ \vdots \\ 3 \times 3 \end{matrix} \right] - \underset{3 \times 1}{X_O}}_{3 \times 4} \underset{4 \times 1}{\mathbf{X}}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

A

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & -u_1 X_1 & -u_1 Y_1 & -u_1 Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1 X_1 & -v_1 Y_1 & -v_1 Z_1 & -v_1 \\ & & & & & & & \vdots & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & -u_n X_n & -u_n Y_n & -u_n Z_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_n X_n & -v_n Y_n & -v_n Z_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

We solved for the matrix M and recovered the projection matrix using the system of linear equations of the form ($Ax=0$) using SVD in the last lecture.

$$\underset{2I \times 12}{M} = \underset{2I \times 12}{U} \underset{12 \times 12}{S} \underset{12 \times 12}{V^T} = \sum_{i=1}^{12} s_i \underset{12 \times 1}{u_i} \underset{1 \times 12}{v_i^T}$$

By choosing the singular vector belonging to the smallest singular value minimizes the error and thus we obtain the transformation matrix M.

However, we still need to know the intrinsic and extrinsic camera parameters and the camera centre and well. Now, we would like to perform the decomposition of the transformation matrix M.

So, far the manner in which we defined M, was forward mapping, where we went from world coordinates to sensor coordinates.

Problem statement: Given M, how do we find K,R, t (or X_0)?

Let us see the structure of the transformation matrix:

$$\mathbf{x} = \mathbf{K} [\mathbf{R} \quad \mathbf{t}] \mathbf{X} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{x} = \mathbf{M} \mathbf{X}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

$$\mathbf{M} = \underbrace{\quad}_{\text{Intrinsic Matrix}} \mathbf{K} \underbrace{\quad}_{\text{Extrinsic Matrix}} [\mathbf{R} \quad \mathbf{t}]$$

$$\begin{matrix} \mathbf{x}_{3 \times 1} & = & \mathbf{K}_{3 \times 3} \mathbf{R}_{3 \times 3} \underbrace{\left[\mathbf{I}_3 | -\mathbf{X}_O \right]}_{3 \times 4} \mathbf{X}_{4 \times 1} \end{matrix}$$

$$= \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{23} & r_{33} \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix}$$

The matrix M the transformation matrix is: $\mathbf{x} = \mathbf{M} \mathbf{X}$

We split the matrix M s.t.:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \quad \begin{bmatrix} m_{14} \\ m_{24} \\ m_{34} \end{bmatrix}$$

$H = M_{3 \times 3}$

$h = M_{3 \times 1}$

M

$$[KR] - KR\mathbf{X}_O = [H|h]$$

Where, $H = KR$ $h = -KR\mathbf{X}_O$

We need to recover the matrix H , here ($H=KR$).

In the last lecture, we studied the properties of the rotational matrix, which is that it is an orthonormal matrix and the matrix K is an upper triangular matrix.

The QR decomposition technique decomposes a matrix into Q and R , where Q is an orthonormal matrix and R is an upper-triangular calibration matrix (note the order of output matrices).

But what we actually need is an upper-triangular calibration matrix (K) times an orthonormal matrix (R).

$$H = KR$$

$$H = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{23} & m_{33} \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{23} & r_{33} \end{bmatrix}$$

Therefore, compute the QR decomposition on H^{-1} , which gives rotation and calibration matrix:

$$H^{-1} = (K R)^{-1} = R^{-1} K^{-1} = R^T K^{-1}$$

↑ ↑

$$QR(H^{-1}) = R^T K^{-1}$$

The matrix H is a homogeneous matrix and therefore we perform normalisation.

We normalise K with $\frac{1}{K_{33}} K$

Finally, the decomposition of H^{-1} yields K with positive diagonal elements. To have our image plane between the camera coordinates and the world system (as taken conventionally), the diagonal elements are typically negative using g this convention.

We choose

$$R(z, \pi) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K \leftarrow KR(z, \pi) \quad R \leftarrow R(z, \pi)R$$

$$H = KR(z, \pi) R(z, \pi)R = KR$$

We basically multiply the output matrices with a rotation matrix $R(z,\pi)$ which does a rotation of 180° around the z-axis and brings the image plane between the camera coordinates and the world system (does flipping).

Once we have H , we can recover the projection centre as:

$$X_O = -H^{-1} h$$

For the transformation matrix of the form: $\mathbf{x} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}] \mathbf{X}$

We recover H using the same procedure as seen above and then to recover the translation vector here, we compute $t=K^{-1} h$ or,

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = K^{-1} \begin{bmatrix} m_{14} \\ m_{24} \\ m_{34} \end{bmatrix}$$

In summary,

- We computed the direct linear transforms estimates of the intrinsic and extrinsic parameters of a camera.
- Our approach computes the parameters for the mapping of the uncalibrated camera (an affine camera with 11 parameters)
- This requires us to know at least 6 control points in 3D
- If the control points lie approximately on a plane, we do not have a proper solution for the transformation matrix itself.

COURSE
DIGITAL IMAGE PROCESSING AND APPLICATIONS IN BIOIMAGE ANALYSIS

BHAVNA R, IISER-BHOPAL 2021/22

DSE312-CVLecture9

DSE-409/609- DIPlecture5

Chapter3: Image filtering in the spatial domain
WEEK-3

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Chapter 3: Image filtering in the spatial domain

Two components in any spatial filter:

- A neighborhood
- An operation defined in the neighborhood

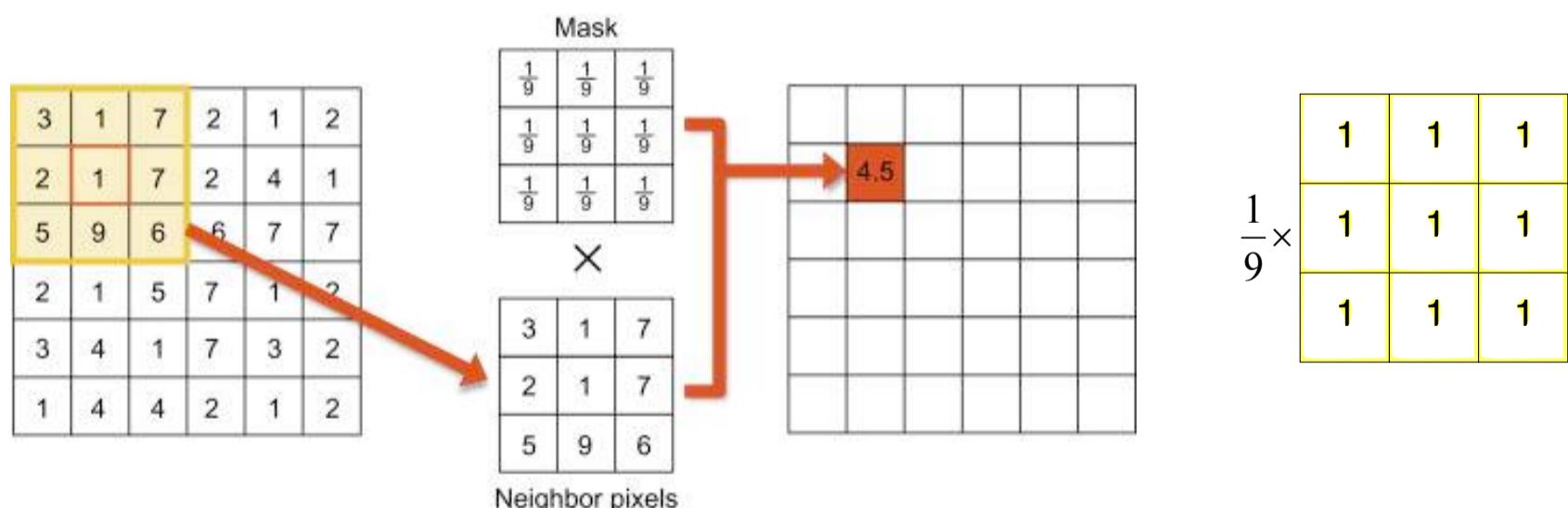
Spatial filtering modifies an image by replacing the value of each pixel by a function of the values of the pixel and its neighbors. If the operation performed on the image pixels is linear, then the filter is called a linear spatial filter. Otherwise, the filter is a nonlinear spatial filter.

We already had a glimpse on spatial filtering in the previous lecture where we looked at neighborhood operations (neighborhood averaging, which is a linear filter).

The generated intensity value for the center pixel (x,y) such that its value is determined by a specified operation on the neighborhood of the pixels in the input image (Depending upon the size of the neighborhood and the type of operation)

Let us start with the same example that we saw in the previous lecture, the standard neighbourhood averaging operation. Averaging reduces any extreme values of neighbourhood in the spatial domain.

The mask/kernel/template/window of the filter sub-image here is 3×3 . The sum of pixels within the mask= 9 pixels , hence, we divide this by 9.



$$\text{Sum of products: } ((1*3)+(1*1)+(1*7)+(1*2)+(1*1)+(1*7)+(1*5)+(1*9)+(1*6))/9 = 4.5$$

Here, I have written $1*$ from the mask image.

However, the mask can have other values. Instead of simple neighborhood averaging, you can also use a weighted averaging filter:

$$\frac{1}{16} \times \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

Here the sum of pixels within the sub-image is 16, hence, we use 1/16.

What will be the centre pixel value in the above image if you applied this weighted mask?

The weighting allows you to weight the pixels (high/low) and allows to choose weighting location within the neighbourhood.

In the above example, we have used 3×3 sub-image filter. You can choose the sub-image size use $k \times l$ where $k=3, 5, 7 \dots$ so on (k, l are odd numbers).

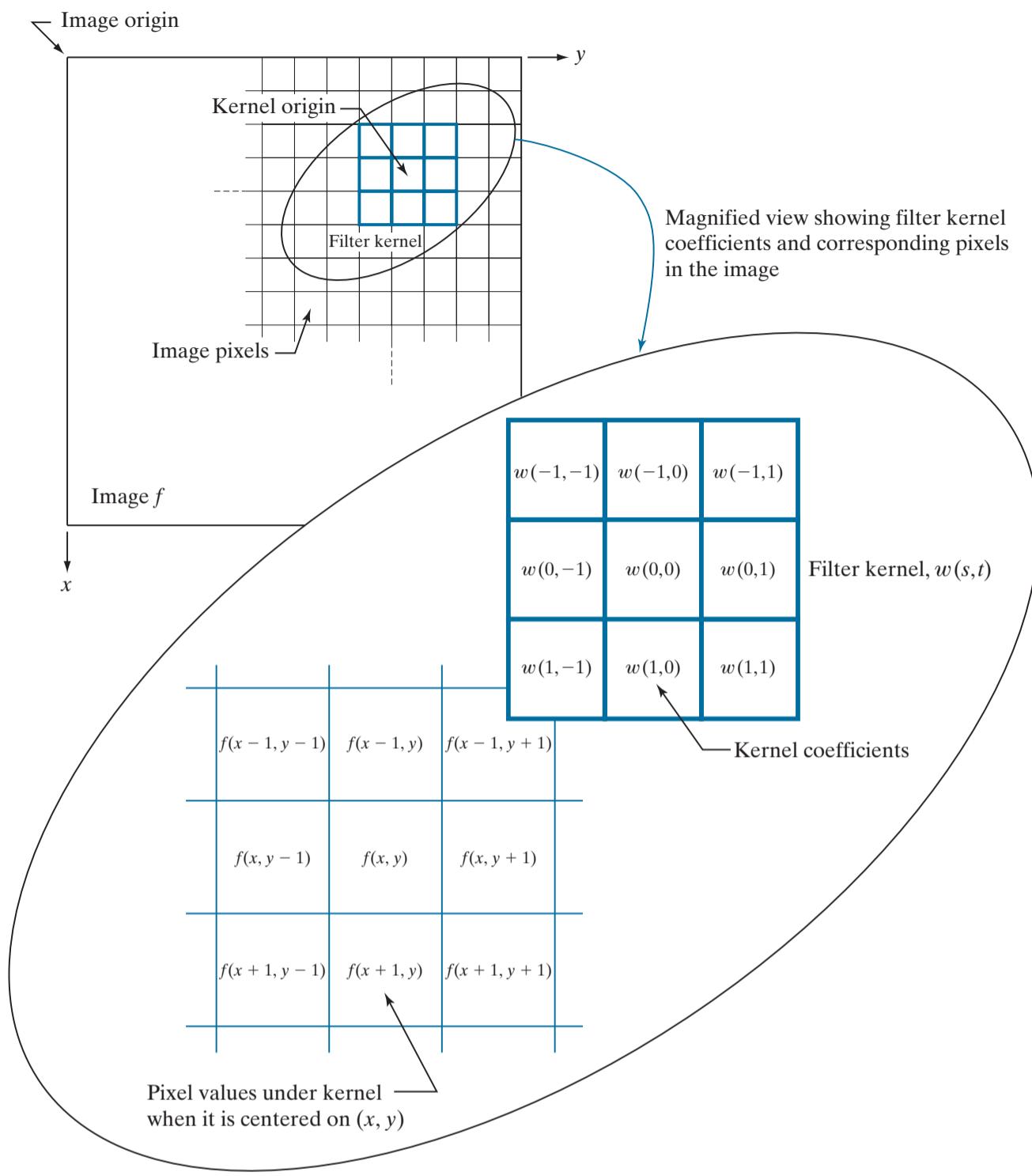
However, the filter size choice should be chosen relative to the size of the original image.

We will see the effect of various filter size on the image in the practical session.

The idea of such a filter is to produce a blur effect and remove noise from the image. Such operations are carried out to enhance features of interest within the image.

Let us generalise this concept and understand the overall picture here. Understanding of spatial filtering is very important in the DIP field.

Here is a pictorial representation of filter kernel applied on an image.



The linear spatial filtering using a 3×3 kernel at any point (x, y) in the image, produces a response, $g(x, y)$, of the filter such that the sum of products of the kernel coefficients and the image pixels encompassed by the kernel, given by;

$$g(x, y) = w(-1, -1)f(x - 1, y - 1) + w(-1, 0)f(x - 1, y) + \dots \\ + w(0, 0)f(x, y) + \dots + w(1, 1)f(x + 1, y + 1)$$

For a fixed value of (x, y) , we take the sum of products.

We traverse the image pixel-to-pixel, placing the kernel on a coordinate centre and generating a filtered image ($g(x, y)$).

The center coefficient of the kernel, $w(0, 0)$, aligns with the pixel at location (x, y) .

For a kernel of size $m \times n$, we assume that $m = 2a + 1$ and $n = 2b + 1$, where a and b are nonnegative integers, in other words both m and n are odd numbers.

This means that our focus is on kernels of odd size in both coordinate directions. In general, linear spatial filtering of an image of size $M \times N$ with a kernel of size $m \times n$ is given by the expression

$$g(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t)f(x + s, y + t)$$

S.t. x and y are varied so that the center (origin) of the kernel visits every pixel in f once.

The above equation is also defined as spatial correlation.

What happens when the kernel operates at the edge pixels?

When the kernel centre is placed at an edge pixel, a part of w lies outside f . To solve this problem, we used zero padding i.e.

For a 2D image, if the kernel is of size $m \times n$, we need $(m - 1)/2$ rows of zeros on top and bottom of f and $(n - 1)/2$ columns of zeros on left and right side to handle the beginning and ending configurations of w with respect to f . Then the centre of the kernel will coincide with $f(0,0)$, which is the left-top most corner position.

Spatial correlation and convolution

Spatial correlation is moving the center of a kernel over an image, and computing the sum of products at each location (as we saw above).

Spatial convolution requires the correlation kernel to be rotated by 180° . Then the same mathematical operation is applied as above.

Hence, when the values of a kernel are symmetric about its centre, correlation and convolution yield the same result.

What is an impulse response?

Impulse response

Let us consider the concept of ‘impulse’. A discrete impulse of strength (amplitude) A located at coordinates (x_0, y_0) is defined as

$$\delta(x - x_0, y - y_0) = \begin{cases} A & \text{if } x = x_0 \text{ and } y = y_0 \\ 0 & \text{otherwise} \end{cases}$$

As seen in the 2D example,

Origin	f
0	0 0 0 0 0
0	0 0 0 0 0
0	0 0 1 0 0
0	0 0 0 0 0
0	0 0 0 0 0

If you plot this as a 2D function ‘ f ’, you will get a single peak at the centre and the rest as 0!

As an image, there will be just 1-pixel in the centre as 1 and remaining as 0. When an impulse function is used as a filter kernel, in a linear convolution, the result is identical to the original image (I).

$$I \otimes \delta = I$$

However, when the image itself is an impulse function (δ) and a filter w is applied to the image, the result of this filter operation is identical to the filter w itself (in case of convolution) or a flipped image (in case of correlation).

How is that?

In case of convolution: $I \otimes \delta = \delta \otimes I = I$;

Here \otimes denotes convolution operator.

We will take a 2D example for the purpose of illustration and understand the properties of convolution and correlation operators in the following.

Correlation vs. Convolution (and zero padding)

		Padded f																																																																																																																																																																								
Origin	f	<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>4</td><td>5</td><td>6</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>7</td><td>8</td><td>9</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	2	3	0	0	0	0	0	4	5	6	0	0	0	0	0	7	8	9	0																																																																																																																
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	1	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	1	2	3	0																																																																																																																																																																			
0	0	0	0	4	5	6	0																																																																																																																																																																			
0	0	0	0	7	8	9	0																																																																																																																																																																			
(a)		(b)																																																																																																																																																																								
Initial position for w	Correlation result	Full correlation result																																																																																																																																																																								
<table border="1"> <tr><td>1</td><td>2</td><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	1	2	3	0	0	0	0	0	4	5	6	0	0	0	0	0	7	8	9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>9</td><td>8</td><td>7</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>6</td><td>5</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	8	7	0	0	0	0	0	6	5	4	0	0	0	0	0	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>9</td><td>8</td><td>7</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>6</td><td>5</td><td>4</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>3</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	8	7	0	0	0	0	0	6	5	4	0	0	0	0	0	3	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	3	0	0	0	0	0																																																																																																																																																																			
4	5	6	0	0	0	0	0																																																																																																																																																																			
7	8	9	0	0	0	0	0																																																																																																																																																																			
0	0	0	1	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	9	8	7	0	0	0	0																																																																																																																																																																			
0	6	5	4	0	0	0	0																																																																																																																																																																			
0	3	2	1	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	9	8	7	0	0	0																																																																																																																																																																			
0	0	6	5	4	0	0	0																																																																																																																																																																			
0	0	3	2	1	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
(c)	(d)	(e)																																																																																																																																																																								
Rotated w	Convolution result	Full convolution result																																																																																																																																																																								
<table border="1"> <tr><td>9</td><td>8</td><td>7</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>6</td><td>5</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>3</td><td>2</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	9	8	7	0	0	0	0	0	6	5	4	0	0	0	0	0	3	2	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>4</td><td>5</td><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>7</td><td>8</td><td>9</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	0	0	0	0	0	4	5	6	0	0	0	0	0	7	8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>2</td><td>3</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>4</td><td>5</td><td>6</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>7</td><td>8</td><td>9</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	0	0	0	0	0	4	5	6	0	0	0	0	0	7	8	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	8	7	0	0	0	0	0																																																																																																																																																																			
6	5	4	0	0	0	0	0																																																																																																																																																																			
3	2	1	0	0	0	0	0																																																																																																																																																																			
0	0	0	1	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	1	2	3	0	0	0	0																																																																																																																																																																			
0	4	5	6	0	0	0	0																																																																																																																																																																			
0	7	8	9	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	1	2	3	0	0	0																																																																																																																																																																			
0	0	4	5	6	0	0	0																																																																																																																																																																			
0	0	7	8	9	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
0	0	0	0	0	0	0	0																																																																																																																																																																			
(f)	(g)	(h)																																																																																																																																																																								

Correlation (middle row) and convolution (last row) of a 2-D kernel with an image consisting of a discrete unit impulse. Note that correlation and convolution are functions of x and y . As these variable change, they displace one function with respect to the other. Prior to convolution, we rotate the kernel (w) by 180° .

- As you can see from the above example, pre-rotating the kernel (in case of convolution) results in exact copy of the kernel.
- Linear spatial filtering and spatial convolution are synonymous.

The convolution or correlation of a kernel w of size $m \times n$ with an image $f(x, y)$, is given by the following:

Correlation:

$$(w \oplus f)(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x+s, y+t)$$

Convolution:

$$(w \otimes f)(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x-s, y-t)$$

In the convolution equation, the minus signs align the coordinates of f and w when one of the functions is rotated by 180° .

Properties	Convolution	Correlation
Commutative	$f \otimes g = g \otimes f$	-
Associative	$f \otimes (g \otimes h) = (f \otimes g) \otimes h$	-
Distributive	$f \otimes (g+h) = (f \otimes g) + (f \otimes h)$	$f \oplus (g+h) = (f \oplus g) + (f \oplus h)$

- Because of the commutative property of convolution, it is not important whether the kernel or the image is pre-rotated. However, it is conventional to rotate the kernel and convolve with the image.
- Also, performing multistage filtering is possible, if the kernels are convolved first, and the resulting kernel is applied to the image (when multiple kernels are required to be applied to the image for processing).

$$w = w_1 \otimes w_2 \otimes \dots \otimes w_N \quad \text{← Applies only to convolution}$$

Since correlation is not commutative, the ordering of kernels does affect the final value (we cannot write an equation like above).

The values of x and y needed to obtain a full convolution are $x = 0, 1, 2, \dots, M - 1$ and $y = 0, 1, 2, \dots, N - 1$. The size of the result is $M \times N$.

If the kernel and image are of sizes $m \times n$ and $M \times N$, respectively, the padding would have to increase to $(m - 1)$ padding elements above and below the image, and $(n - 1)$ elements to the left and right. The size of the resulting full correlation or convolution array will be of size $S_v \times S_h$, where,

$$S_v = m + M - 1$$

$$S_h = n + N - 1$$

- The order of the functions input into a correlation algorithm makes a difference, because correlation is neither commutative nor associative.

- To filter an image using one of these kernels, we perform a convolution of the kernel with the image.
- Convolution filter, convolution mask, or convolution kernel denote filter kernels for spatial filtering
- Correlation measures the similarity between two signals (or 2 images here, order of operation is important here), whereas convolution gives the effective measure between the 2 signals.

Separable Filter Kernels

A 2D function $G(x,y)$ is said to be separable if it can be written as the outer product of two 1D functions, $G_1(x)$ and $G_2(y)$: $G(x,y) = G_1(x) \otimes G_2(y)$.

For instance, the 2×3 kernel ‘w’ is separable as it can be expressed as an outer product of 2 vectors (c & r)

$$w = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } r = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad w = c r^T$$

- A separable kernel of size $m \times n$ can be expressed as the outer product of two vectors v and w :

$$w = v w^T$$

where v and w are vectors of size $m \times 1$ and $n \times 1$ respectively.

- For a square kernel of size $m \times m$: $w = v v^T$
- Product of a column vector and a row vector is the same as the 2-D convolution of the vectors

How is this property useful?

Recall that for convolution, both commutative and associative properties are applicable

Properties	Convolution
Commutative	$f \otimes g = g \otimes f$
Associative	$f \otimes (g \otimes h) = (f \otimes g) \otimes h$

If we have separable kernel, we can take computational advantage while performing convolution operations, since

$$w \star f = (w_1 \star w_2) \star f = (w_2 \star w_1) \star f = w_2 \star (w_1 \star f) = (w_1 \star f) \star w_2$$

For an image of size $M \times N$ and a kernel of size $m \times n$, implementation of convolution requires on the order of $MNmn$ multiplications and additions.

Convolution:

$$(w \otimes f)(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x-s, y-t)$$

This is because it follows directly from that equation that each pixel in the output (filtered) image depends on all the coefficients in the filter kernel.

But since the kernel is separable then the first convolution, $w_1 \star f$,

requires on the order of MNm since w_1 is of size $m \times 1$. Similarly, w_2 is of size $n \times 1$.

The two 1-D kernels, w_1 and w_2 , are implemented in order for 1-D convolution.

The assumption here also is that the values of M and N include any padding of f prior to performing convolution.

Computational cost

Computing the results of filters is computationally expensive in most cases, especially with large images, large filter kernels, or both. If both the image and the filter are simply assumed to be of size $N \times N$, the time complexity of direct filtering is $O(N^4)$. Substantial savings are possible when large, 2D filters can be decomposed (separated) into smaller, possibly 1D filters (as just seen above).

This results in total of $MN(m + n)$ multiplication and addition operations for $M \times N$ image and $m \times n$ kernel size.

This directly affects the execution time for the operation on a computer. Hence, there is a certain computational cost (in this case computational advantage) in doing such a operation since the kernels are separable).

Computational advantage given by

$$C = \frac{MNmn}{MN(m+n)} = \frac{mn}{m+n}$$

For a kernel of 11x11, this is 5.5. What is it for a kernel of 51x51?

The convolution filter operation is used a lot while building deep learning algorithms on images, and such computational cost can be taken care of within the deep models.

Other properties of separable kernels

What is rank of a matrix?

the rows and columns of a matrix whose rank is 1 are linearly dependent.

When a matrix has a rank of 1, you can form separable kernels

How to Build Spatial Filter Kernels?

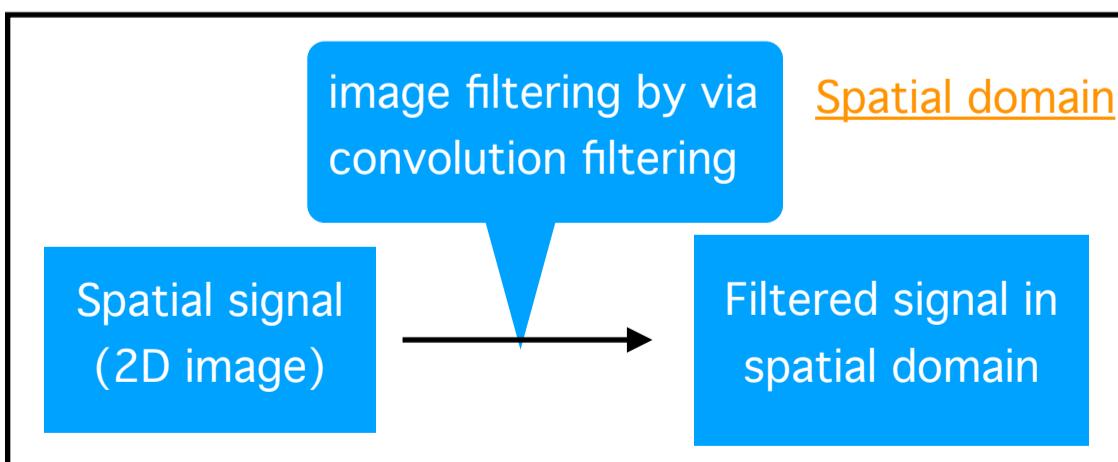
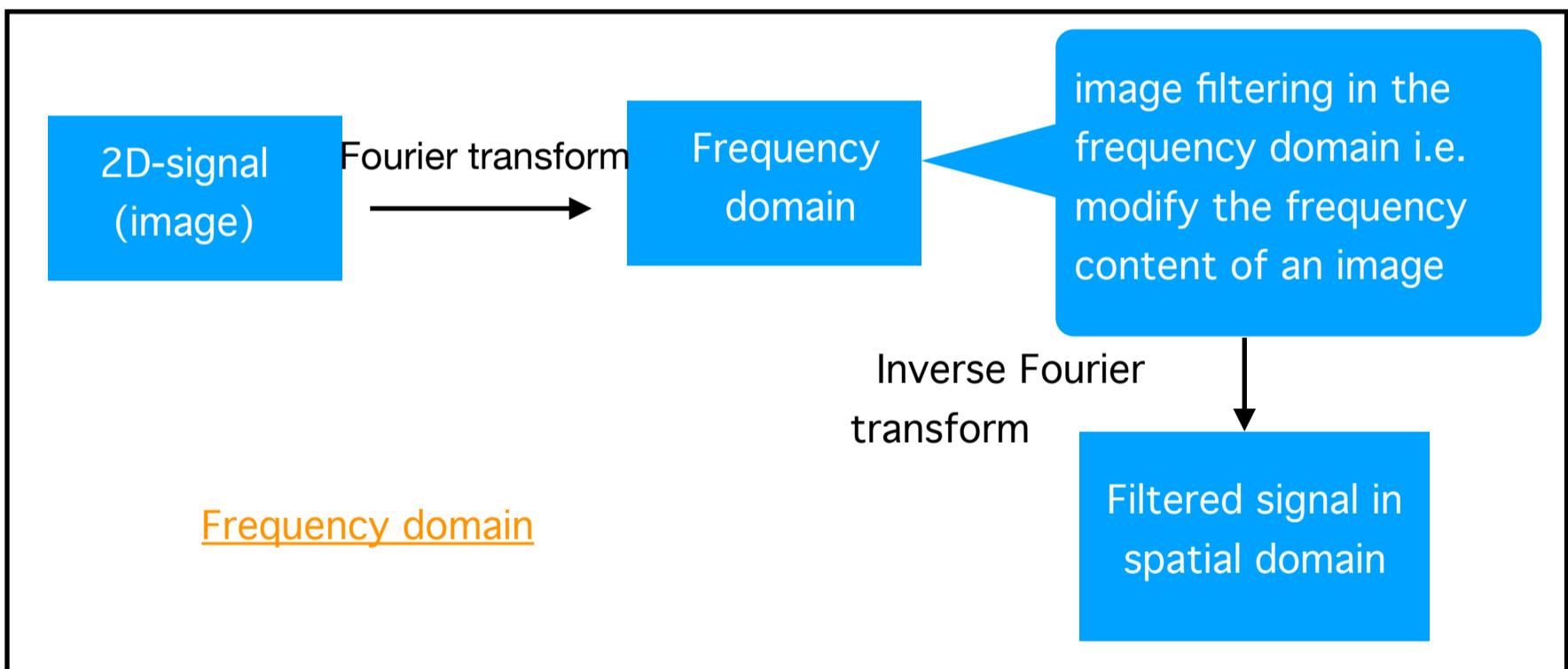
There are three main approaches for building spatial kernel filters:

- Based on mathematical properties; for example using integration (averaging) for blurring an image, or computing local derivatives allow sharpening an image.
- Sampling a 2D spatial function whose shape has a desired property; for example creating a weighted-average filter by sampling a 2D Gaussian function (also known as lowpass filter), or sampling the inverse FT of a 2D filter that is specified in the frequency domain (next topic).
- Designing a spatial filter with a specified frequency response; for example designing a 1D spatial filter and then forming a separable or circularly symmetric 2D version (more details in the next topic).

Here, we would like to discuss briefly the fundamental differences between filtering operation performed in spatial domain versus frequency domain.

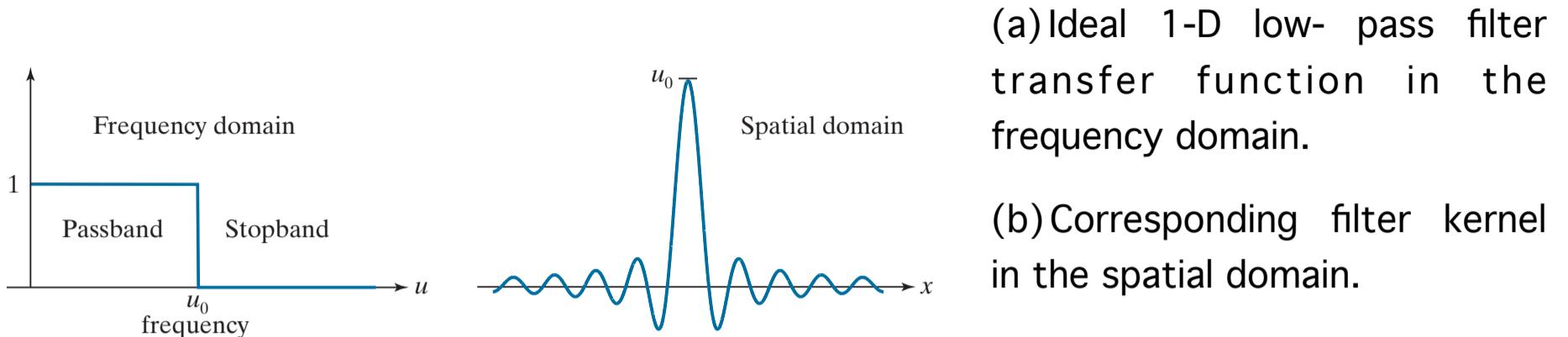
- We consider the image (a 2D function) depends on the frequencies of its sinusoidal components, which means that an image can be expressed as the sum of sinusoids of different frequencies and amplitudes.
- So modifying the frequency components of an image, modifies its appearance.
- Image intensities that vary slowly (sinusoids of low frequencies) or sharp intensity transitions (characterised by high frequencies). Then reducing that specific frequency component within the image in the frequency domain can blur the image of specific frequencies accordingly. One can design sharp cut-off of frequencies for filtering in the frequency domain.

Let us look at the basic filtering steps involved in spatial domain versus frequency domain.



1-D signal example of a low pass filter in both frequency and spatial domain

Here is an example showing response of 1-d low pass filter in both the domains.



(c) u_0 is the cut-off frequency, all frequencies above u_0 are eliminated, so you can get a sharp transition of the filter in the frequency domain. The same operation in the spatial domain will be achieved via convolution operation, however there will be some small frequencies present after applying the spatial filter.

- This is because FT decomposes the signal (image) into its consisting frequency components. So it is possible to sharply cut the signal in the frequency domain.
- An impulse function of amplitude 'A' in the spatial domain, is a constant function of value A in the frequency domain, and vice versa.
- Mathematically, convolution in the spatial domain, is equivalent to multiplication in the frequency domain, and vice versa.

From here on, we will first consider the spatial domain filtering operations and then detail on the frequency domain filtering operations in the next lecture.

Types of spatial filters

Linear filters

Box Filter Kernels (average filters)

Lowpass Gaussian Filter Kernels

Nonlinear filters

Order-Statistic Filters
(median, max filters)

Linear or non-linear?

Linear filters When the output is expressed as linear combination of the inputs (sum of the products).

Non-linear filters There is no linear relationship between the input and output. The order-statistic filters arrange the pixels under mask as per the operation (eg, median or max values).

Low pass filters

- Smoothing (averaging) spatial filters are used for reducing sharp transitions in intensity.
- Random noise consists sharp transitions; therefore they reduce noise too.
- Smoothing can reduce false contouring.
- Convolving a smoothing kernel with an image blurs the image, with the degree of blurring being determined by the size of the kernel and the values of its coefficients.

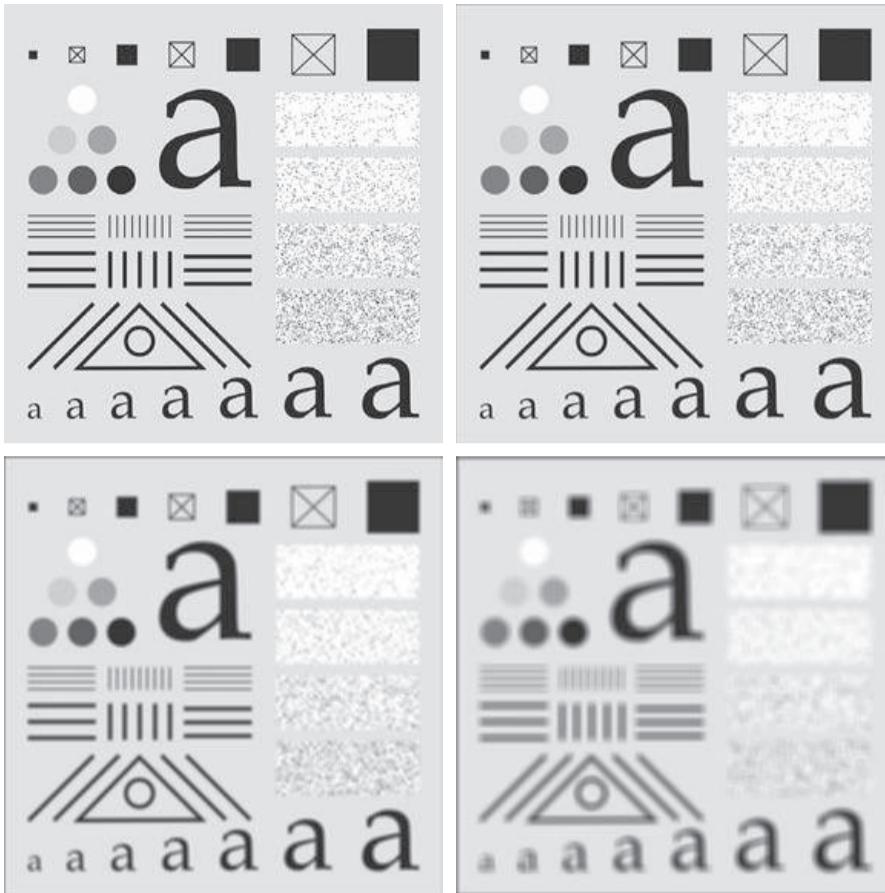
Box Filter Kernels

We don't go into the details, as we have already seen the working of this filter. So , I hope by now we can understand how this filter operation works on an image (convolution). You may have a simple averaging filter or weighted averaging filter:

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$
$$\frac{1}{16} \times \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

A note on the significance of having normalisation constant (1/9 or 1/16):

- An area of constant intensity will remain the same after filtering;
- Prevents bias during filtering, sum of the pixels in the original and filtered images are the same.



(a) Test pattern of size 1024×1024 pixels. (b)-(d) Results of simple lowpass filtering with box kernels of sizes 3×3, 11×11, and 21×21, respectively.

Larger kernel size produces more blurring effect.

What should be our choice?: filter, then which size/or no filter? Or

Are there better options?

Lowpass Gaussian Filter Kernels

The box filters are a good choice when smoothing edges. However, they have limitations on images with a high level of detail, or with strong geometrical components, the directionality of box filters often produces undesirable results.

A Gaussian kernel is a smoothing kernel that gives less weight to pixels further from the center. It is circularly symmetric (also called isotropic, meaning their response is independent of orientation).

A 2D- Gaussian function is defined as:

$$w(s, t) = \frac{1}{2\pi\sigma^2} e^{-\frac{s^2 + t^2}{\sigma^2}}$$

Where (s,t) are defined in the 2D spatial domain, σ controls the spread of the Gaussian about its mean.

For simplicity, we replace the constant $1/(2\pi\sigma^2)= K$, since we are interested in the bell shape of the Gaussian function.

So we have a 2D Gaussian kernel;

$$w(s, t) = G(s, t) = Ke^{-\frac{s^2 + t^2}{2\sigma^2}}$$

Gaussian kernels are separable, they can be written as:

$$G_\sigma = e^{-\frac{s^2+t^2}{2\sigma^2}} = \exp\left(-\frac{s^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{t^2}{2\sigma^2}\right) = g_\sigma x \cdot g_\sigma y$$

So we can separate the x and y filters as follows

$$I * G_\sigma = I * g_\sigma x * g_\sigma y$$

Hence the computational advantages with Gaussian filters are same as box filters.

- The ordering of the two 1D filters is not relevant in this case.
- The product and convolution of two Gaussians are also Gaussian functions.
- With different σ -values along the x and y axes, elliptical 2D Gaussians can be realised as separable filters in the same fashion.
- We consider a circularly symmetric 2-D function for simplicity here (examples shown below)

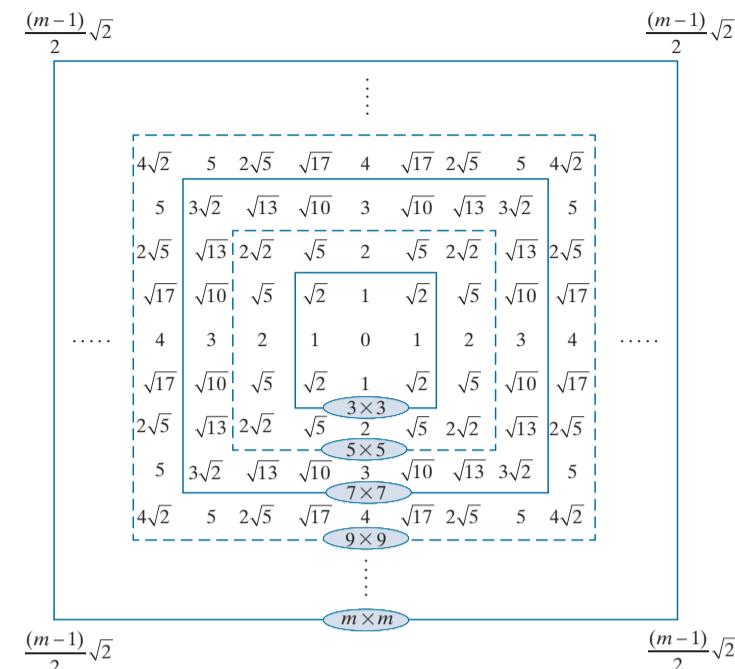
Let $r = \sqrt{s^2 + t^2}$ we can write

$$G(r) = K e^{-\frac{r^2}{2\sigma^2}}$$

such that r is the distance from the center to any point on function G.

Earlier the example that we saw is in fact a discrete approximation of Gaussian filter. Observe the centre value of the kernel versus the others.

However the size of this kernel is small in the above example. In reality, to avoid visible truncation errors, discrete approximations of the Gaussian should have a sufficiently large extent of about $\pm 2.5 \sigma$ to $\pm 3.5 \sigma$ samples, so that the kernel is really smoothed.

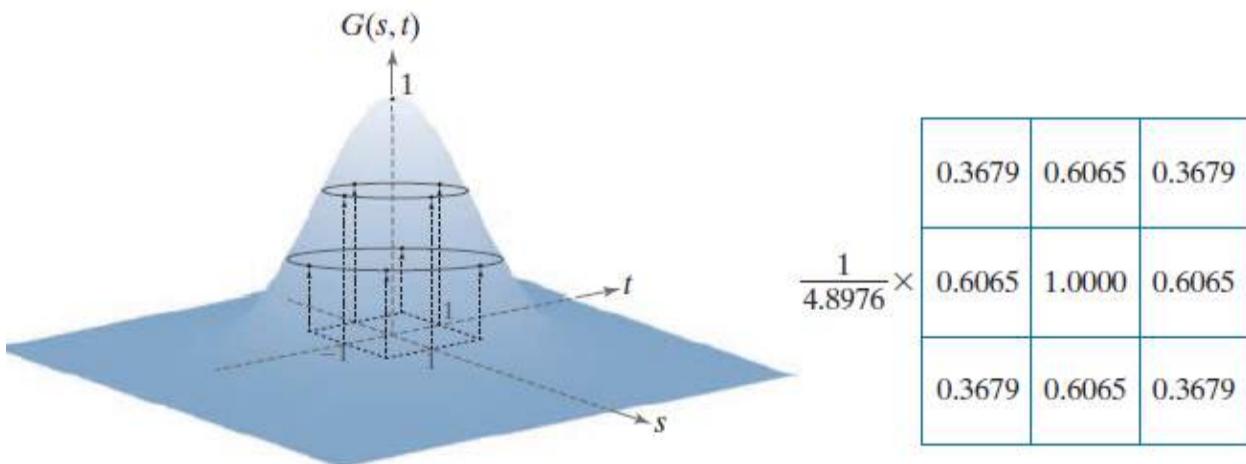


$$\frac{1}{16} \times \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

We assume a symmetric Gaussian kernel in the example showing distances from the center for various sizes of square kernels.

Because we work generally with odd kernel sizes, the centers of such kernels fall on integer values, and it follows that all values of r^2 are integers also.

- The Gaussian function decays relatively slowly with increasing distance from the center.
- The Gaussian function is a continuous function, and cannot be applied directly to the images.
- We need to sample it at integer spatial locations, and make a discrete kernel.
- Normalizing the kernel by dividing its coefficients by the sum of the coefficients completes the process of specifying the kernel.
- Small Gaussian kernels cannot capture the characteristic Gaussian bell shape, and thus behave more like box kernels. Approximately, the size for Gaussian kernels is kept in the order of $6\sigma \times 6\sigma$ (31×31 kernel if $\sigma=5$;) at the maximum since there is no gain (in smoothing) beyond $6\sigma \times 6\sigma$ kernel size. Within this size, the smoothing function can vary, too small a kernel size will behave like box filter kernel.



(a) Sampling a Gaussian function to obtain a discrete Gaussian kernel. The values shown are for $K = 1$, $\sigma = 1$. (b) Resulting 3×3 kernel .

We can write the kernel of two 1-D Gaussian functions, f and g (since we already saw that the kernel is separable). Below is the table showing mean and standard deviation of the product and convolution of two 1-D Gaussian functions, f and g :

	f	g	$f \times g$	$f \star g$
Mean	m_f	m_g	$m_{f \times g} = \frac{m_f \sigma_g^2 + m_g \sigma_f^2}{\sigma_f^2 + \sigma_g^2}$	$m_{f \star g} = m_f + m_g$
Standard deviation	σ_f	σ_g	$\sigma_{f \times g} = \sqrt{\frac{\sigma_f^2 \sigma_g^2}{\sigma_f^2 + \sigma_g^2}}$	$\sigma_{f \star g} = \sqrt{\sigma_f^2 + \sigma_g^2}$

We are only interested in the standard deviation of the kernel (that controls the width of the kernel, mean is zero), as seen in the kernel equation shown above.



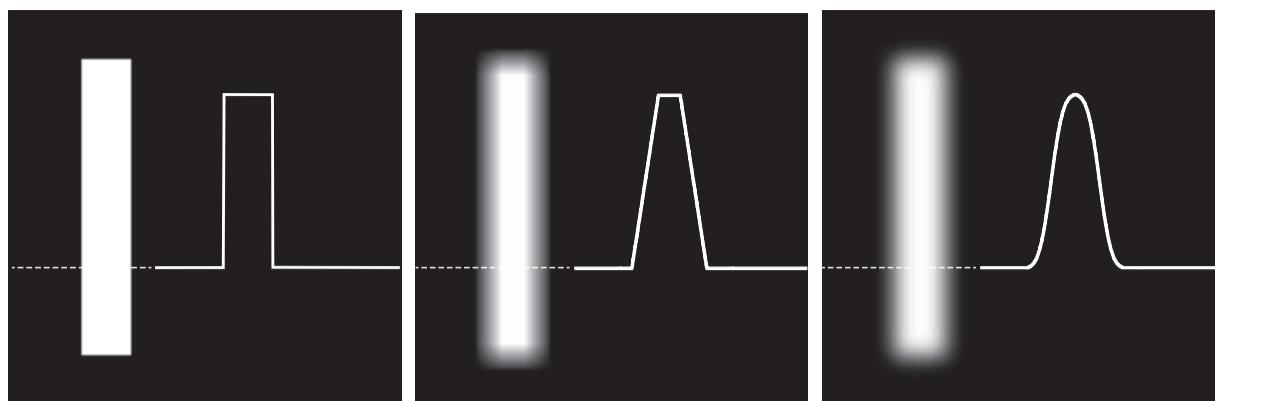
(a)image of size 1024×1024 . (b)-(c) lowpass filtering with a Gaussian kernel of sizes 21×21 , $\sigma = 3.5$ and size 43×43 , with $\sigma = 7$. K = 1 in all cases.

Let us roughly compare the results of box kernel filtering versus Gaussian filtering:

Gaussian kernels have to be larger than box filters to achieve the same degree of blurring/smoothing.

Why so ?

Because box kernel assigns the same weight to all pixels, whereas Gaussian kernel coefficients decreases as a function of distance from the kernel center.



Original image
1024 × 1024

box kernel of
size 71×71

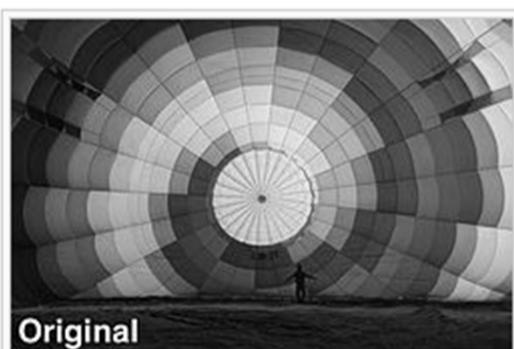
Gaussian kernel, size
151×151,K = 1, $\sigma= 25$

There are a few important points for comparison in this operation:

You are given a black background with white pixels. How is the transition from black to white (i.e., at an edge) different in the two cases? observe carefully..

- The box filter produces a hard transition , hence preferable when less smoothing of edges is desired.
- Gaussian filter yielded smoother results around the edge transitions, preferable for uniform smoothing.
- Also the Gaussian kernel gives 2 control parameters : window size (radius from the centre) and standard deviations, whereas the box kernel uses only window size
- Notice how the rectangle has a smoother edge effect with the Gaussian kernel case.
- Notice how a smoothed peak is obtained using the Gaussian Kernel.
- Another important point is that the relative blurring produced by a smoothing kernel of a given size depends directly on image size, whatever kernel type is used.

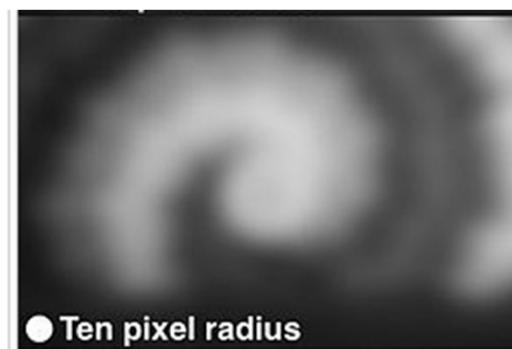
Effect of smoothing radius



Original



• Three pixel radius



● Ten pixel radius

Handling image borders using padding

We already saw previously zero padding, i.e. adding zeros at the edges prior to performing convolution/filtering operations.

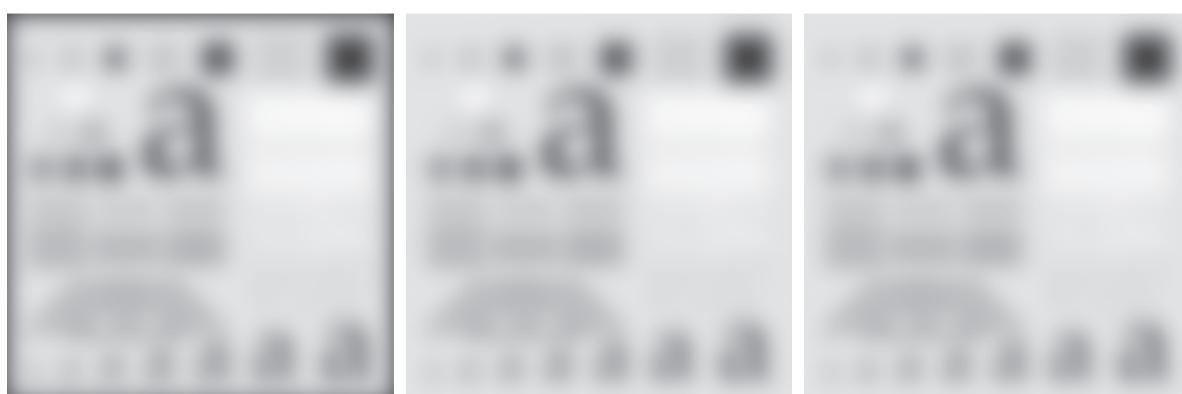
However, if noticed carefully in those images (with white background), in both the cases, zero padding an image introduces dark borders in the filtered result, with the thickness of the borders depending on the size and type of the filter kernel used.

There are two other methods of image padding that attempt to “extend” the characteristics of an image past its borders:

Mirror (also called symmetric) padding: values outside the boundary of the image are obtained by mirror-reflecting the image across its border; applicable when the areas near the border contain image details.

Replicate padding: values outside the boundary are set equal to the nearest image border value, useful when the areas near the border of the image are constant.

Notice the image edges in the following and the objects close to the border:



Gaussian kernel of size 187×187 , $K = 1$, $\sigma = 31$ in all the 3 cases using (a) zero padding, (b) mirror padding, and (c) replicate padding.

Mirror padding example

3	5	1
3	6	1
4	7	9

1	6	3	6	1	6	3
1	5	3	5	1	5	3
1	6	3	6	1	6	3
9	7	4	7	9	7	4
1	6	3	6	1	6	3

Mirror/Reflection padding “reflects” the row into the padding. This is useful because it ensures that the outputs will transition “smoothly” into the padding.

Replicated padding

These pixel values are replicated from boundary pixels.

17	24	1	8	15
23	5	7	8	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

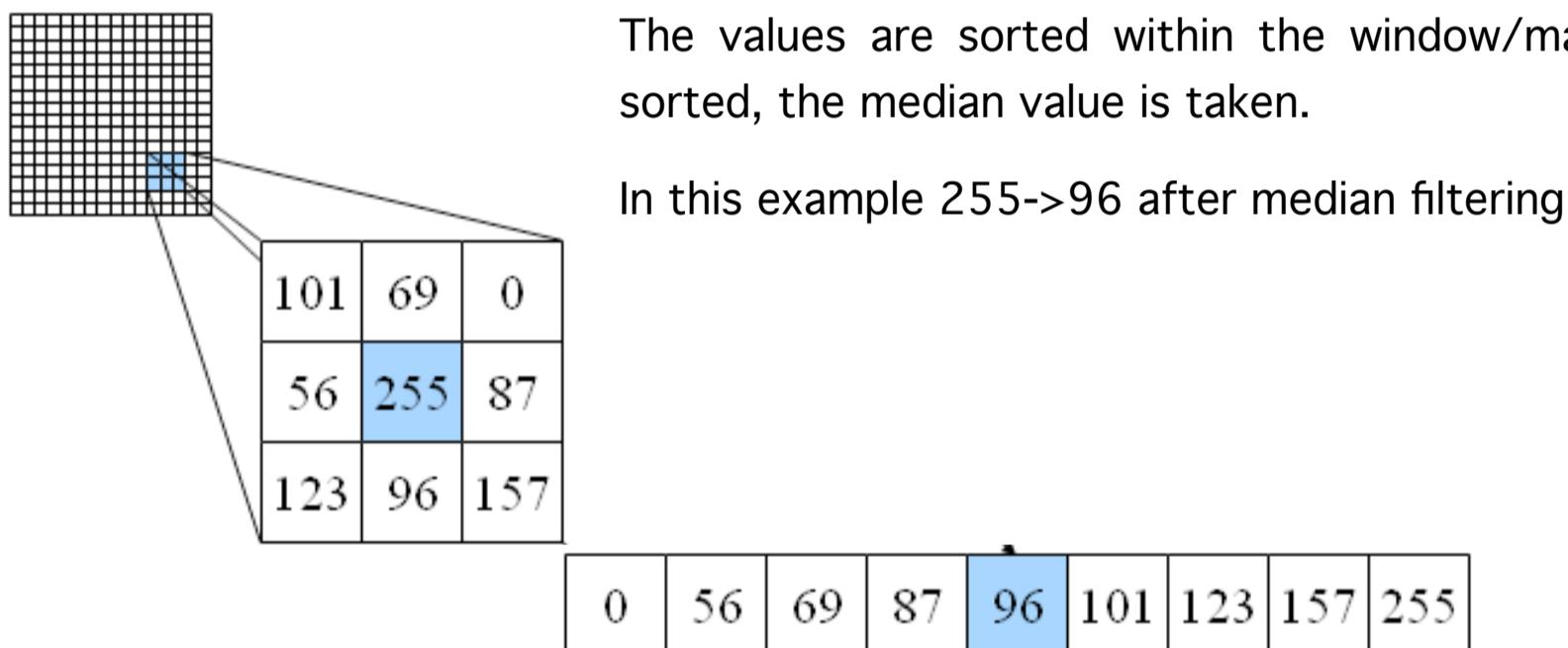
The distribution of the data is disturbed as little as possible here. The value of any pixel outside the image is determined by replicating the value from the nearest border pixel.

Order statistics or non-linear filters

- Order-statistic filters are nonlinear filters that work based on ordering (ranking) of the pixels in the region encompassed by the neighbourhood.
- Median Filter replaces the value of the centre pixel by the median of the intensity values defined in the neighbourhood of that pixel. Very useful for reducing salt-and-pepper noise.
- Median of a set of values is such that half of the values in the set are less than or equal to the median. When several values are the same, they will be grouped.

Median filter is not similar to convolution:

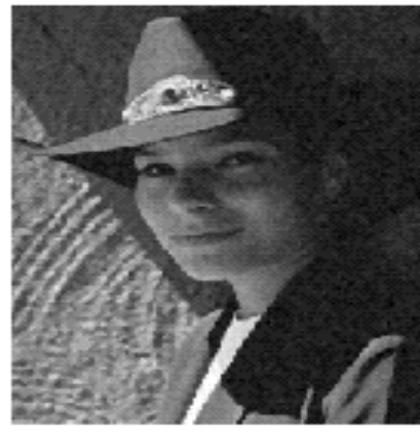
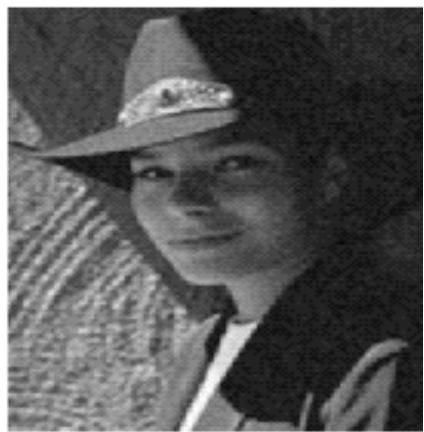
$$\text{Median}(f_1(x)+f_2(x)) \neq \text{Median}(f_1(x))+\text{Median}(f_2(x))$$



Similarly, you can also have a minimum or a maximum order statistics filters.

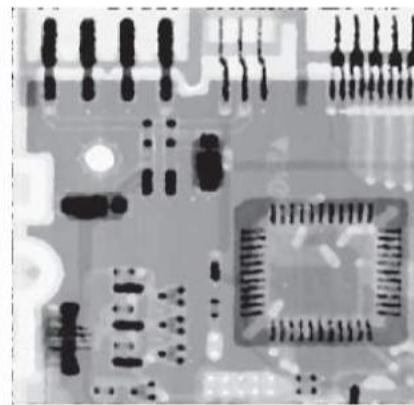
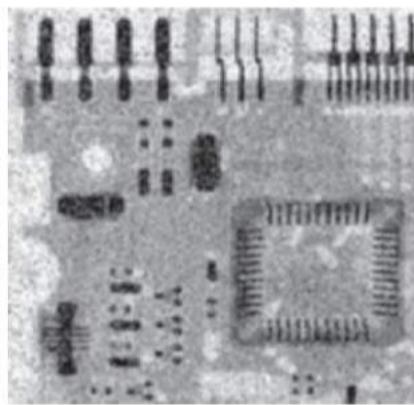
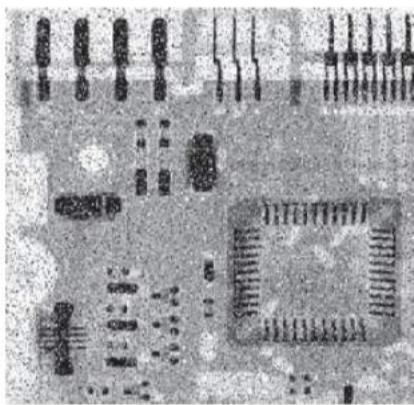
Salt-pepper noise is filtered out using median filter.





Whereas the Gaussian noise is filtered out better using the Gaussian filter

Here is another example



(a) X-ray image of a circuit board, corrupted by salt-and-pepper noise. (b) Noise reduction using a 19×19 Gaussian lowpass filter kernel with $\sigma = 3$. (c) Noise reduction using a 7×7 median filter.

Major differences between smoothing & sharpening operations

<u>Smoothing filters</u>	<u>Sharpening filters</u>
• Averaging (smoothing) is analogous to integration	• <u>sharpening</u> is analogous to <u>spatial differentiation</u>
• reduces noise and eliminates fine details	• highlights/enhances fine details
• mask weights is <u>1 (after normalization)</u> for smoothing filters	• <u>Sum of the mask weights is 0 (after normalization)</u>
• constant intensity in input and output be kept same; prevents a bias during filtering	• uniform/non-changing intensity values are kept zero (based on derivatives)
• mask kernel values are positive	• mask kernel has both positive and negative elements

High pass filters (sharpening filters)

- Sharpening highlights transitions in intensity.
- Image differentiation enhances edges and other discontinuities (like noise) and de-emphasizes areas of slowly varying intensities.

Sharpening filters are based on image derivatives.

Derivatives of a digital function are defined in terms of differences between adjacent pixels.

First order image derivatives and second order image derivatives

First derivative

- nonzero at the onset of an intensity step or ramp (eg. edge).
- zero in areas of constant intensity.
- nonzero along intensity ramps.

Second derivative

- zero in areas of constant intensity.
- nonzero at the onset and end of an intensity step or ramp.
- zero along intensity ramps.

1st derivative of $f(x)$ is the difference

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

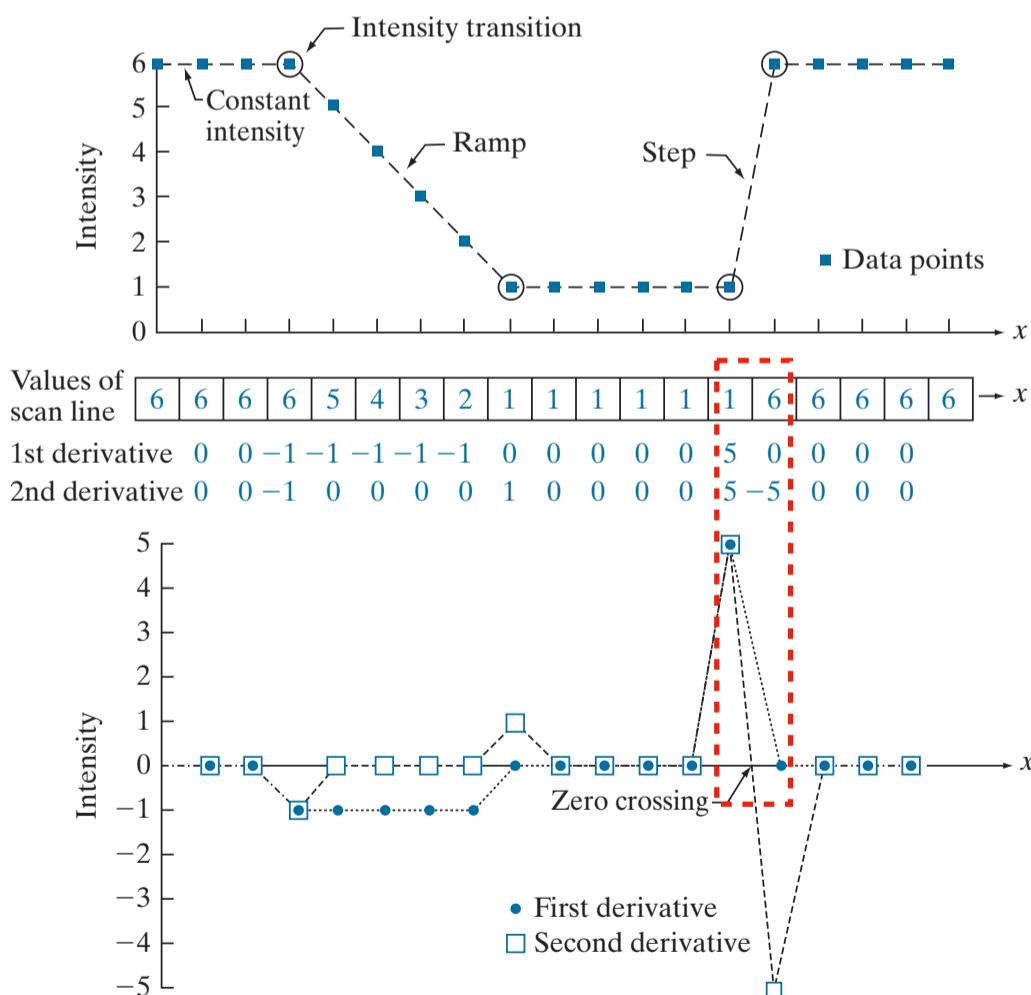
2nd derivative of $f(x)$ is:

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

We write them in the form of partial derivatives since an image is a function of 2 variable $f(x,y)$.

For the 1-d case, $\partial f / \partial x = df / dx$ or $\partial^2 f / \partial x^2 = d^2 f / dx^2$.

Let us take an example to see how derivatives would operate on images?



(a) A section of a horizontal scan line from an image, showing ramp and step edges, as well as constant segments. (b) Values of the scan line and its derivatives.

(c) Plot of the derivatives, showing a zero crossing. In (a) and (c) points were joined by dashed lines as a visual aid.

Red Dotted region shows the Zero-crossing

Edges in digital images

- Edges are ramp-like transitions in intensity. First derivative of the image would result in thick edges because the derivative is nonzero along a ramp.
- second derivative would produce a double edge one pixel thick, separated by zeros.
- second derivative enhances fine detail much better than the first derivative

We can use the property of the second derivative for sharpening images or for finding edges within the images.

Gradients (1st derivatives) and Laplacian (2nd derivatives) are used in various context in image processing applications, however in the following sections we will exclusively focus on their image sharpening qualities.

Second Order Derivatives for Image Sharpening – The Laplacian

We write the Laplacian form of a function with 2 variables (for a 2D image) as:

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

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

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

Summing up;

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

The equation for Laplacian can be used to generate a discrete form of the filter/mask and then be implemented on the image using convolution with the kernel (same procedure as seen for image smoothing).

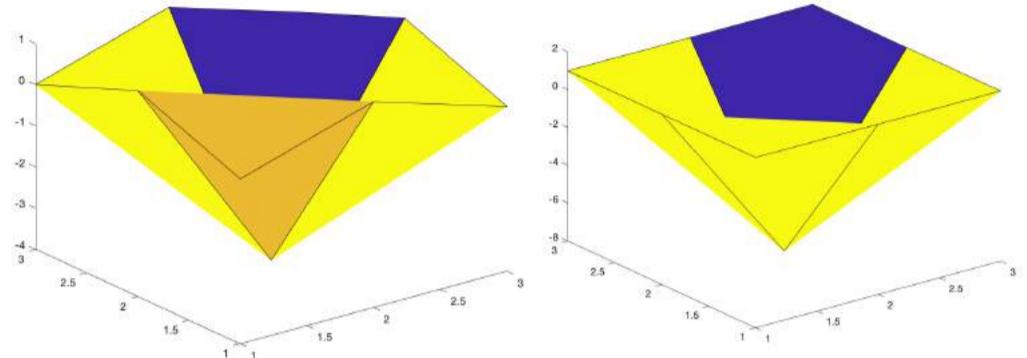
So $\nabla^2 I = L \otimes I$, where the 2D image $I(x,y)$ is convolved with the Laplacian kernel (L)

As derivatives of any order are linear operations, the Laplacian is also a linear operator.

Essentially, we need to design a Laplacian filter in the discrete form to have an $m \times n$ kernel, here are some examples.

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

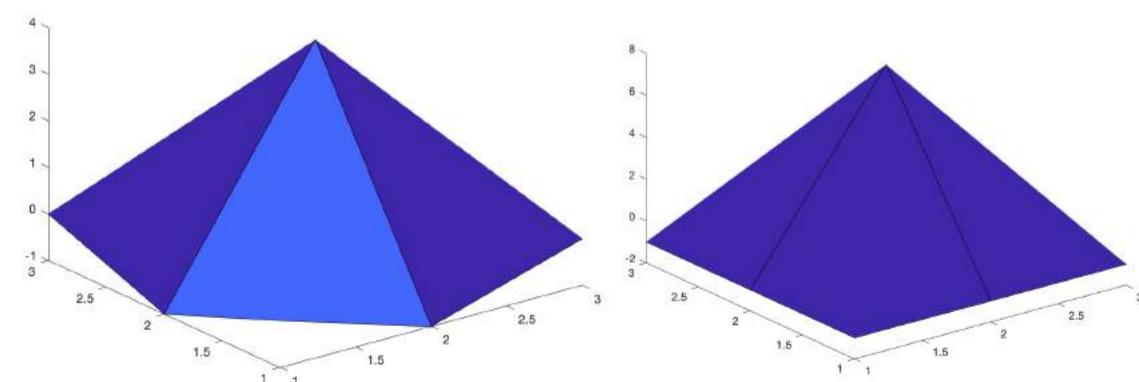


- a) Laplacian kernel (given by the standard equation above, note that kernel is isotropic for rotations in increments of 90° with respect to the x- and y-axes. Only 4-neighbours are considered (top, bottom, left right)).
- b) Laplacian kernel including diagonal elements. For the case to include diagonal elements, we need to include more terms in the equation for $\nabla^2 f(x,y)$. Because each diagonal term would contain a $-2 f(x, y)$ term, the total subtracted from the difference terms is $-8f(x,y)$.
- c) and d) are the 2D plots of the Laplacian kernel shown in a) and b)

Now we take the same 2 kernels and multiply by (-1). We have inverted the kernel2 in this case. We can see the 2D plots showing peaks at the centre instead.

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1



The 2D plots are for demo purpose to show the shape of the filter. You can also design filters of larger kernel size, of similar characteristics.

Laplacian is a derivative operator, it highlights sharp intensity transition. C Important to note the definition of Laplacian for image sharpening, given by,

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$$

where $f(x,y)$ and $g(x,y)$ are input and sharpened images, respectively. $c = -1$ for negative centre and $c = 1$ when positive centre values are used. So, you can either subtract or add the kernel to obtain a sharpened image.

Coefficients of each kernel sum to zero

Convolution-based filtering implements a sum of products, so when a derivative kernel encompasses a constant region in a image, the result of convolution in that location is zero. So the coefficients sums can achieve this.



0	1	0
1	-4	1
0	1	0

- (a) Blurred image of the North Pole of the moon.
- (b) Laplacian image obtained using the kernel shown
- (c) Image sharpened gives $g(x,y)$ (adding images a) and b) with $c=-1$.
- (d) Image sharpened with kernel (more information due to diagonal elements)

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$$

$c=-1$

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

First Order Derivatives for Image Sharpening – The Gradient

First derivatives in image processing are implemented using the magnitude of the gradient. The gradient of an image f at coordinates (x, y) is defined as the 2D column vector

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y) .

The magnitude (length) of vector ∇f , denoted as $M(x, y)$, which is norm of vector ∇f is given by,

$$\begin{aligned} M(x, y) &= \|\nabla f\| = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \\ &\approx |g_x| + |g_y| \quad \text{approximation} \end{aligned}$$

You can also compute the gradient direction, given by;

- $M(x, y)$ is the value at (x, y) of the rate of change in the direction of the gradient vector.
- Note that g_x alone will give the component along the x-direction, and g_y gives the y-direction component, these are rotation variant.
- Components of the gradient vector are derivatives, hence they are linear operators.
- $M(x, y)$ is not a linear operator because of the squaring and square root operations.
- $M(x, y)$ is rotation invariant.

For mathematical simplicity, let's consider the 3×3 region of an image, where the z_s are intensity values

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Recall a first-order derivative is given by $\frac{\partial f}{\partial x} = f(x+1) - f(x)$

Then a simplest operation is: $g_x = (z_8 - z_5)$ and $g_y = (z_6 - z_5)$.

One of the simplest gradient operators, known as Roberts cross-gradient operator (proposed way back in 1965), a 2×2 kernel is given by

-1	0	0	-1
0	1	1	0

$$g_x = (z_9 - z_5) \quad \text{and} \quad g_y = (z_8 - z_6) \quad M(x, y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2 \right]^{1/2}$$

However, we prefer kernels of odd sizes because they have a unique, (integer) center of spatial symmetry. Approximations to g_x and g_y using a 3×3 neighborhood centered on z_5 are as follows (the matrix is also called Sobel operators):

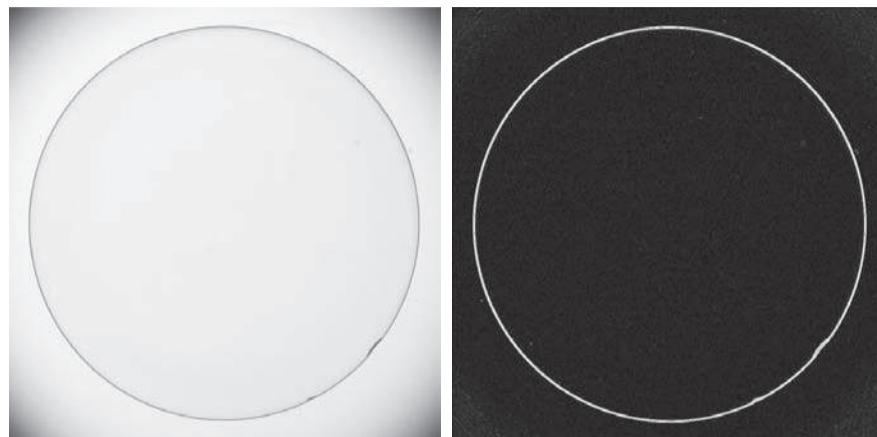
$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

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

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

$$M(x, y) = \left[g_x^2 + g_y^2 \right]^{\frac{1}{2}} = \left[[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2 \right]^{\frac{1}{2}}$$

There are more such pre-defined operators that can do edge enhancement (we will discuss these later).



(a) Image of a contact lens
 (b) Sobel gradient.

Unsharp masking and high boost filtering

We can subtract an unsharp (smoothed) version of an image from the original image to create a mask for sharpening the original image.

Steps:

Blur the original image $f(x, y) \Rightarrow \bar{f}(x, y)$

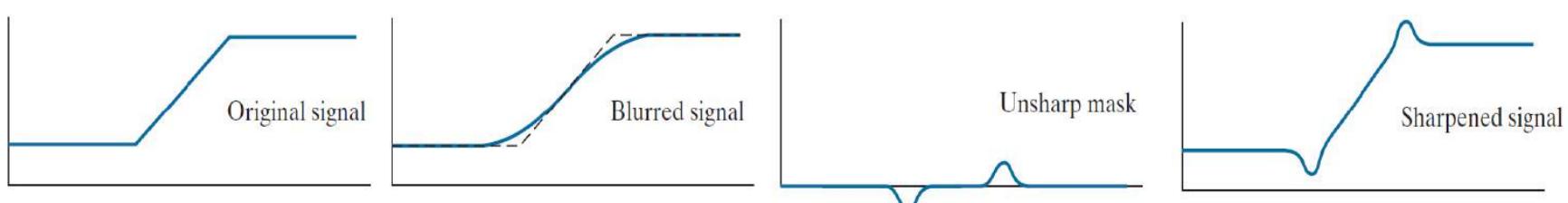
Subtract the blurred image from the original to create the unsharp mask:

$$g_{mask}(x, y) = f(x, y) - \bar{f}(x, y)$$

Add the mask to the original image ('k' is the control parameter)

$$g(x, y) = f(x, y) + k g_{mask}(x, y)$$

When $k = 1$ we have unsharp masking (defined above). $k > 1$, produces high-boost filtering. $k < 1$ reduces the contribution of the unsharp mask.





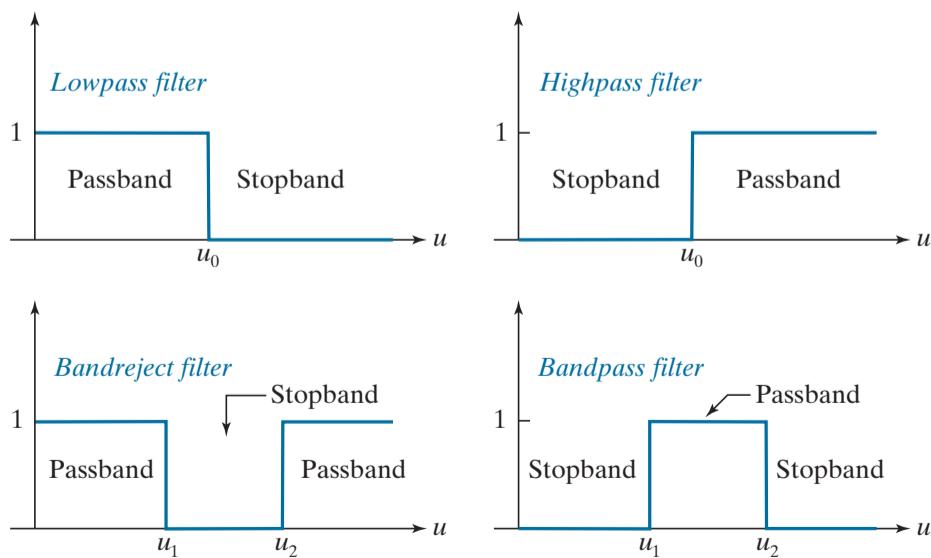
- a) image size 600×259 pixels. (b) Image blurred using a 31×31 Gaussian lowpass filter with $\sigma = 5$. (c) Mask. (d) $k = 1$ (sharpen). (e) high-boost filtering with $k = 4.5$.

High-pass, Band-reject and Bandpass Filters from Lowpass Filters

$$\xrightarrow{f(t)} g(t) = \mathcal{H}\{f(t)\} \xrightarrow{g(t)}$$

In ‘signals and system’, an abstract operator $H\{\cdot\}$ can be described completely using the impulse response $h(t)$ or transfer function $H(\xi)$. The impulse response is combined with the signal by the operation of convolution. This is sufficient to describe all linear shift-invariant systems.

We use the transfer function of a 1-D ideal lowpass filter in the frequency domain. A constant in the frequency domain is an impulse in the spatial domain. It is easier to understand the filtering in the frequency domain.



Transfer functions of ideal 1-D filters in the frequency domain (u denotes frequency). Shown only positive frequencies for simplicity.

A highpass filter transfer function \Rightarrow 1- lowpass function(frequency domain).

In the spatial domain,

highpass filter kernel= unit impulse kernel -lowpass filter kernel

Or

high boost filter=original image -lowpass filter kernel. \Rightarrow unsharp mask filter

Filter type	Spatial kernel in terms of lowpass kernel, lp
Lowpass	$lp(x, y)$
Highpass	$hp(x, y) = \delta(x, y) - lp(x, y)$
Bandreject	$br(x, y) = lp_1(x, y) + hp_2(x, y)$ $= lp_1(x, y) + [\delta(x, y) - lp_2(x, y)]$
Bandpass	$bp(x, y) = \delta(x, y) - br(x, y)$ $= \delta(x, y) - [lp_1(x, y) + [\delta(x, y) - lp_2(x, y)]]$

COURSE
DIGITAL IMAGE PROCESSING AND APPLICATIONS IN BIOIMAGE ANALYSIS

BHAVNA R, IISER-BHOPAL 2021/22
DSE312-CVLecture10
DSE-409/609- DIPlecture6

Chapter4: Image filtering in the spatial domain-part2
WEEK-3

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Chapter 4: Padding, non-linear filters, sharpening filters

So far we saw basic smoothing filters (box filters and Gaussian filters), convolution and correlation of two images and their fundamental properties. We also learnt zero-padding while doing spatial filtering/ convolution at image borders.

Handling image borders using padding

We already saw previously zero padding, i.e. adding zeros at the edges prior to performing convolution/filtering operations.

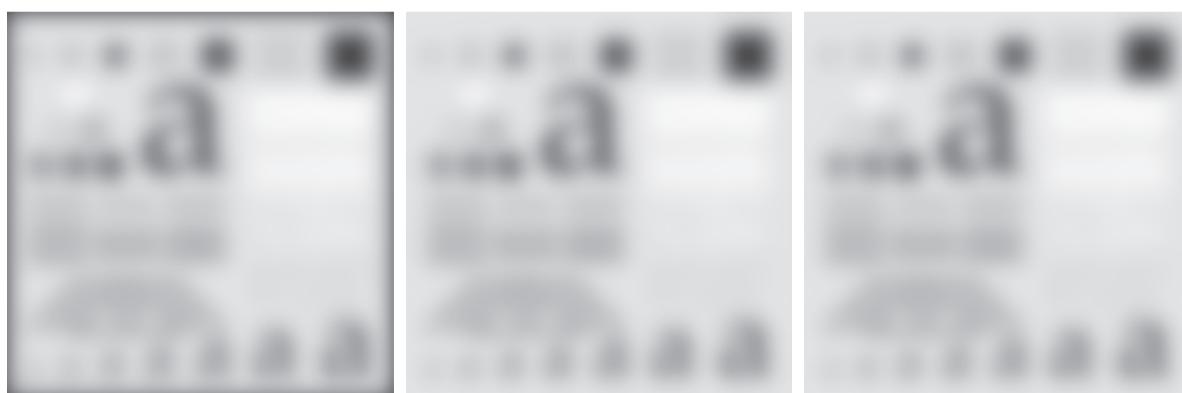
However, if noticed carefully in those images (with white background), in both the cases, zero padding an image introduces dark borders in the filtered result, with the thickness of the borders depending on the size and type of the filter kernel used.

There are two other methods of image padding that attempt to “extend” the characteristics of an image past its borders:

Mirror (also called symmetric) padding: values outside the boundary of the image are obtained by mirror-reflecting the image across its border; applicable when the areas near the border contain image details.

Replicate padding: values outside the boundary are set equal to the nearest image border value, useful when the areas near the border of the image are constant.

Notice the image edges in the following and the objects close to the border:



Gaussian kernel of size 187×187 , $K = 1$, $\sigma = 31$ in all the 3 cases using (a) zero padding, (b) mirror padding, and (c) replicate padding.

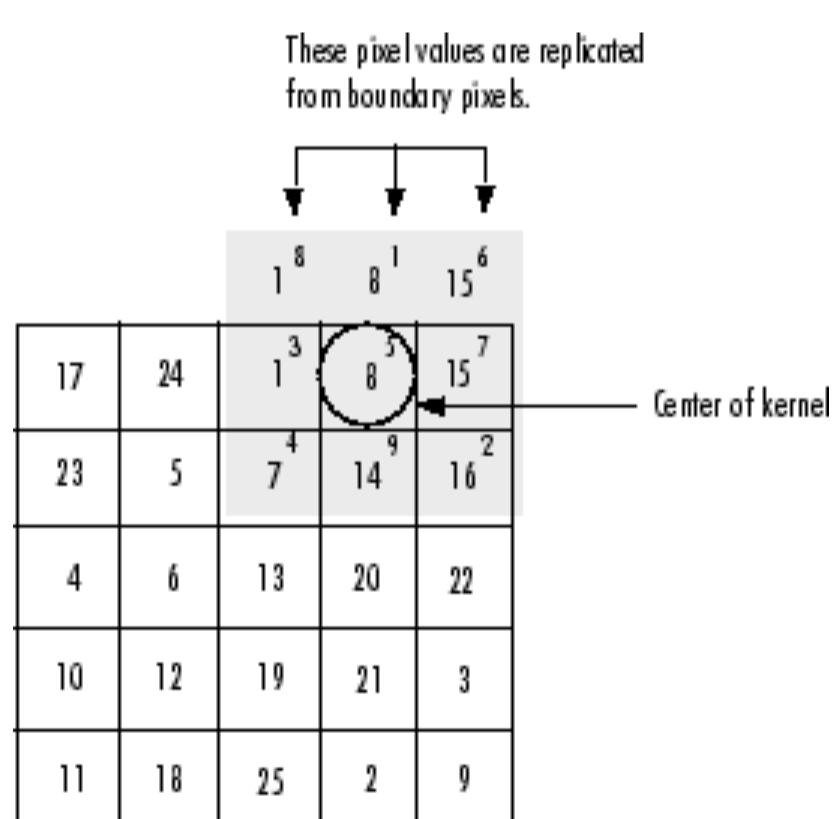
Mirror padding example

3	5	1
3	6	1
4	7	9

1	6	3	6	1	6	3
1	5	3	5	1	5	3
1	6	3	6	1	6	3
9	7	4	7	9	7	4
1	6	3	6	1	6	3

Mirror/Reflection padding “reflects” the row into the padding. This is useful because it ensures that the outputs will transition “smoothly” into the padding.

Replicated padding



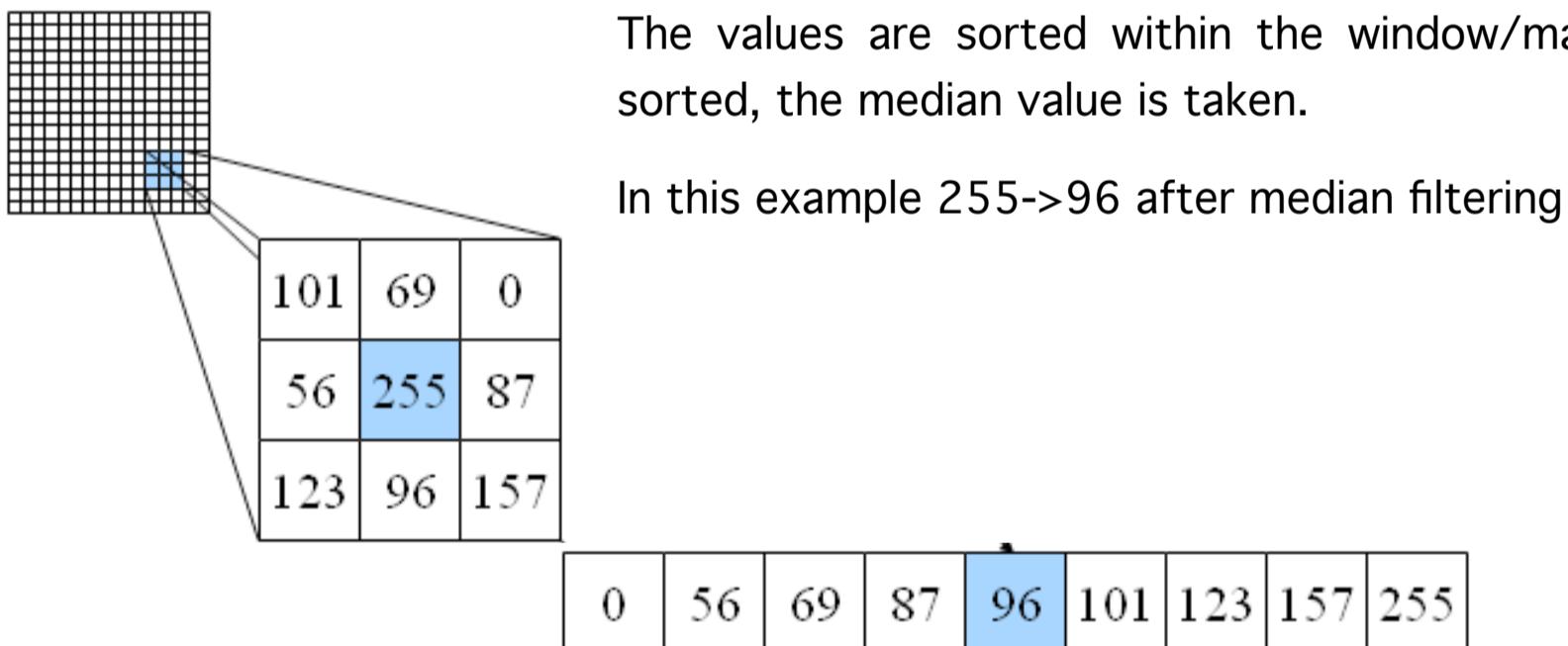
The distribution of the data is disturbed as little as possible here. The value of any pixel outside the image is determined by replicating the value from the nearest border pixel.

Order statistics or non-linear filters

- Order-statistic filters are nonlinear filters that work based on ordering (ranking) of the pixels in the region encompassed by the neighbourhood.
- Median Filter replaces the value of the centre pixel by the median of the intensity values defined in the neighbourhood of that pixel. Very useful for reducing salt-and-pepper noise.
- Median of a set of values is such that half of the values in the set are less than or equal to the median. When several values are the same, they will be grouped.

Median filter is not similar to convolution:

$$\text{Median}(f_1(x)+f_2(x)) \neq \text{Median}(f_1(x))+\text{Median}(f_2(x))$$



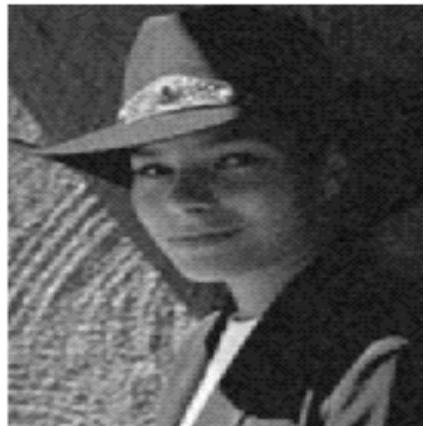
Similarly, you can also have a minimum or a maximum order statistics filters.

Salt-pepper noise is filtered out using median filter.



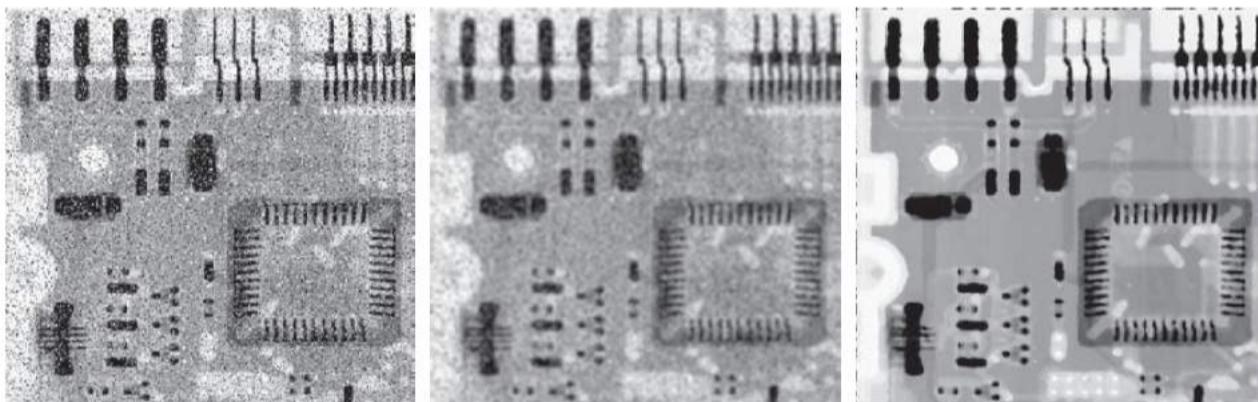
Salt-pepper
noise/speckles
are better

filtered with median filter



Whereas the Gaussian noise is filtered out better using the Gaussian filter

Here is another example



(a) X-ray image of a circuit board, corrupted by salt-and-pepper noise. (b) Noise reduction using a 19×19 Gaussian lowpass filter kernel with $\sigma = 3$. (c) Noise reduction using a 7×7 median filter.

Major differences between smoothing & sharpening operations

<u>Smoothing filters</u>	<u>Sharpening filters</u>
• Averaging (smoothing) is analogous to integration	• <u>sharpening</u> is analogous to <u>spatial differentiation</u>
• reduces noise and eliminates fine details	• highlights/enhances fine details
• mask weights is <u>1 (after normalization)</u> <u>for smoothing filters</u>	• <u>Sum of the mask weights is 0 (after normalization)</u>
• constant intensity in input and output be kept same; prevents a bias during filtering	• uniform/non-changing intensity values are kept zero (based on derivatives)
• mask kernel values are positive	• mask kernel has both positive and negative elements

High pass filters (sharpening filters)

- Sharpening highlights transitions in intensity.
- Image differentiation enhances edges and other discontinuities (like noise) and de-emphasizes areas of slowly varying intensities.

Sharpening filters are based on image derivatives.

Derivatives of a digital function are defined in terms of differences between adjacent pixels.

First order image derivatives and second order image derivatives

First derivative

- nonzero at the onset of an intensity step or ramp (eg. edge).
- zero in areas of constant intensity.

- nonzero along intensity ramps.

Second derivative

- zero in areas of constant intensity.
- nonzero at the onset and end of an intensity step or ramp.
- zero along intensity ramps.

1st derivative of $f(x)$ is the difference

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

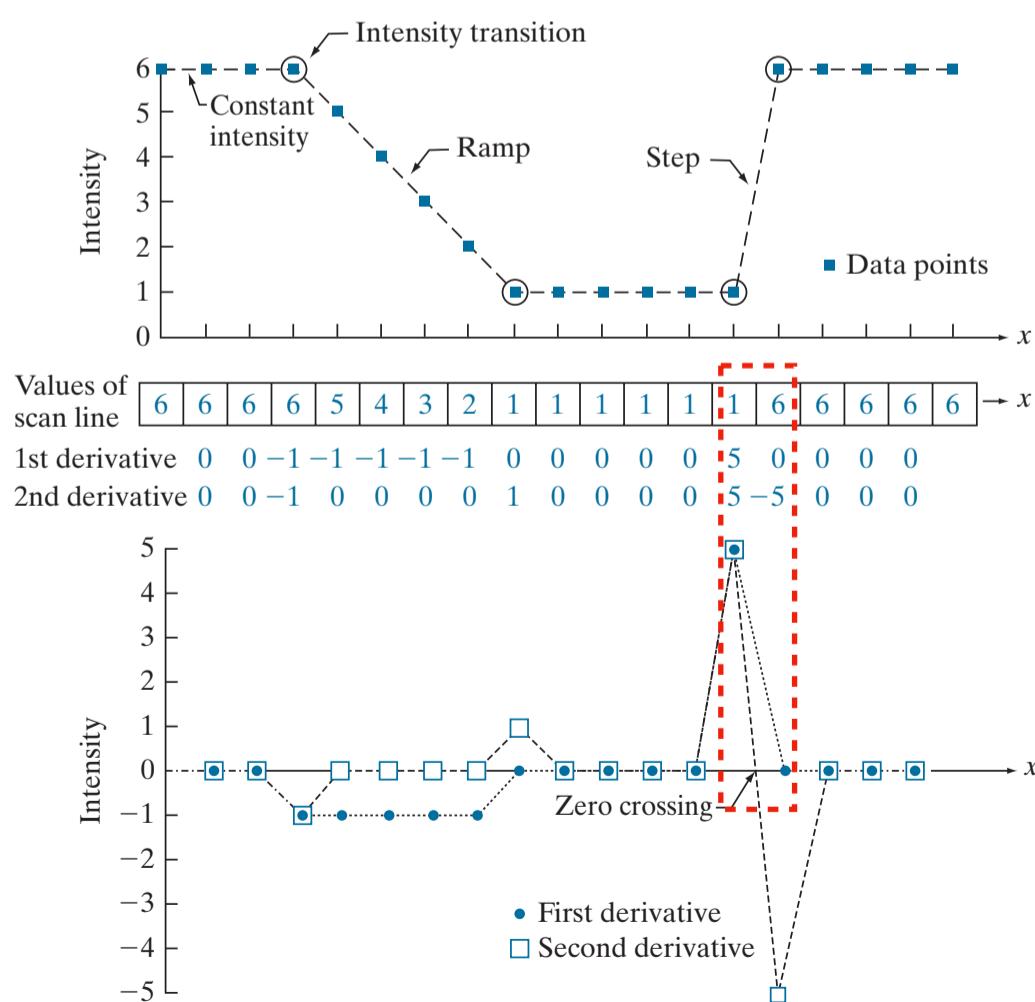
2nd derivative of $f(x)$ is:

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

We write them in the form of partial derivatives since an image is a function of 2 variable $f(x,y)$.

For the 1-d case, $\partial f / \partial x = df/dx$ or $\partial^2 f / \partial x^2 = d^2f/dx^2$.

Let us take an example to see how derivatives would operate on images?



(a) A section of a horizontal scan line from an image, showing ramp and step edges, as well as constant segments. (b) Values of the scan line and its derivatives.

(c) Plot of the derivatives, showing a zero crossing. In (a) and (c) points were joined by dashed lines as a visual aid.

Red Dotted region shows the Zero-crossing

Edges in digital images

- Edges are ramp-like transitions in intensity. First derivative of the image would result in thick edges because the derivative is nonzero along a ramp.
- second derivative would produce a double edge one pixel thick, separated by zeros.
- second derivative enhances fine detail much better than the first derivative

We can use the property of the second derivative for sharpening images or for finding edges within the images.

Gradients (1st derivatives) and Laplacian (2nd derivatives) are used in various context in image processing applications, however in the following sections we will exclusively focus on their image sharpening qualities.

Second Order Derivatives for Image Sharpening – The Laplacian

We write the Laplacian form of a function with 2 variables (for a 2D image) as:

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

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

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

Summing up;

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

The equation for Laplacian can be used to generate a discrete form of the filter/mask and then be implemented on the image using convolution with the kernel (same procedure as seen for image smoothing).

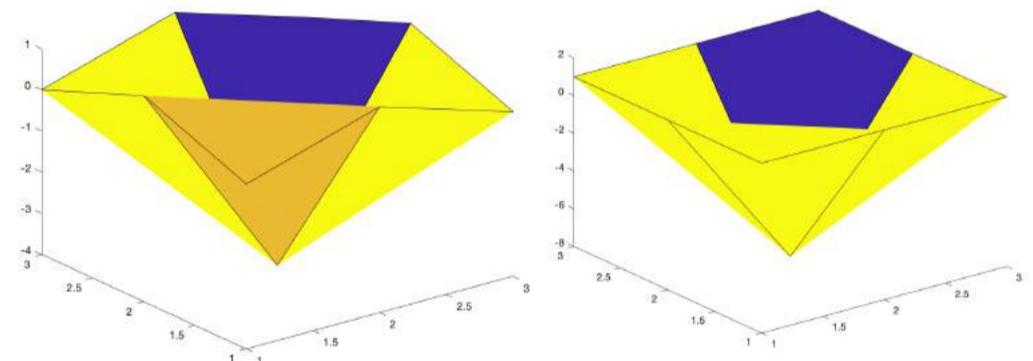
So $\nabla^2 I = L \otimes I$, where the 2D image $I(x,y)$ is convolved with the Laplacian kernel (L)

As derivatives of any order are linear operations, the Laplacian is also a linear operator.

Essentially, we need to design a Laplacian filter in the discrete form to have an $m \times n$ kernel, here are some examples.

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

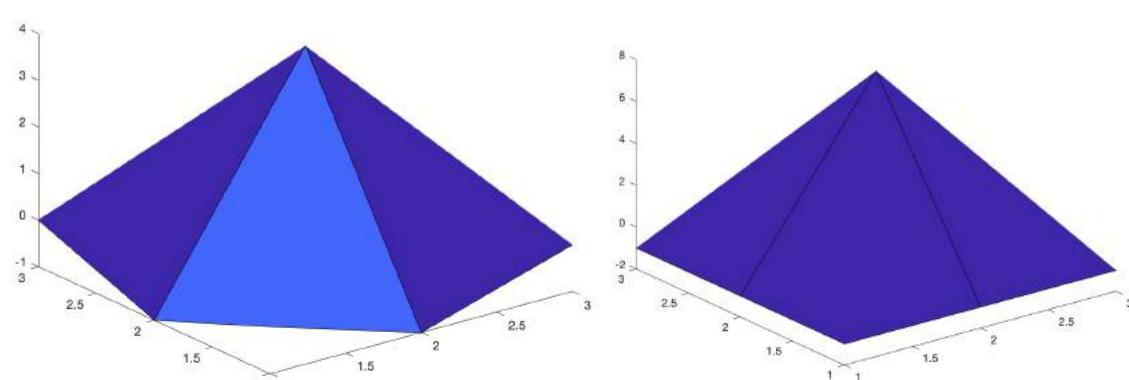


- a) Laplacian kernel (given by the standard equation above, note that kernel is isotropic for rotations in increments of 90° with respect to the x- and y-axes. Only 4-neighbours are considered (top, bottom, left right)).
- b) Laplacian kernel including diagonal elements. For the case to include diagonal elements, we need to include more terms in the equation for $\nabla^2 f(x,y)$. Because each diagonal term would contain a $-2 f(x, y)$ term, the total subtracted from the difference terms is $-8f(x,y)$.
- c) and d) are the 2D plots of the Laplacian kernel shown in a) and b)

Now we take the same 2 kernels and multiply by (-1). We have inverted the kernel in this case. We can see the 2D plots showing peaks at the centre instead.

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1



The 2D plots are for demo purpose to show the shape of the filter. You can also design filters of larger kernel size, of similar characteristics.

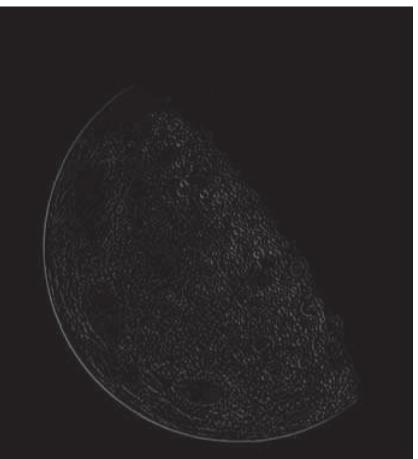
Laplacian is a derivative operator, it highlights sharp intensity transition. C Important to note the definition of Laplacian for image sharpening, given by,

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$$

where $f(x,y)$ and $g(x,y)$ are input and sharpened images, respectively. $c = -1$ for negative centre and $c = 1$ when positive centre values are used. So, you can either subtract or add the kernel to obtain a sharpened image.

Coefficients of each kernel sum to zero

Convolution-based filtering implements a sum of products, so when a derivative kernel encompasses a constant region in a image, the result of convolution in that location is zero. So the coefficients sums can achieve this.



0	1	0
1	-4	1
0	1	0

(a) Blurred image of the North Pole of the moon.



1	1	1
1	-8	1
1	1	1

(b) Laplacian image obtained using the kernel shown (c) Image sharpened gives $g(x,y)$ (adding images a) and b) with $c=-1$. (d) Image sharpened with kernel (more information due to diagonal elements)

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$$

$c=-1$

0	1	0
1	-4	1
0	1	0

First Order Derivatives for Image Sharpening – The Gradient

First derivatives in image processing are implemented using the magnitude of the gradient. The gradient of an image f at coordinates (x, y) is defined as the 2D column vector

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y) .

The magnitude (length) of vector ∇f , denoted as $M(x, y)$, which is norm of vector ∇f is given by,

$$M(x, y) = \|\nabla f\| = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

$$\approx |g_x| + |g_y|$$

approximation

You can also compute the gradient direction, given by;

- $M(x, y)$ is the value at (x, y) of the rate of change in the direction of the gradient vector.
- Note that g_x alone will give the component along the x-direction, and g_y gives the y-direction component, these are rotation variant.
- Components of the gradient vector are derivatives, hence they are linear operators.
- $M(x, y)$ is not a linear operator because of the squaring and square root operations.
- $M(x, y)$ is rotation invariant.

For mathematical simplicity, lets consider the 3×3 region of an image, where the z_s are intensity values

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Recall a first-order derivative is

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \quad \text{given by}$$

Then a simplest operation is: $g_x = (z_8 - z_5)$ and $g_y = (z_6 - z_5)$.

One of the simplest gradient operators, known as Roberts cross-gradient operator (proposed way back in 1965), a 2×2 kernel is given by

-1	0	0	-1
0	1	1	0

$$g_x = (z_9 - z_5) \quad \text{and} \quad g_y = (z_8 - z_6) \quad M(x, y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2 \right]^{1/2}$$

However, we prefer kernels of odd sizes because they have a unique, (integer) center of spatial symmetry. Approximations to g_x and g_y using a 3×3 neighborhood centered on z_5 are as follows (the matrix is also called Sobel operators):

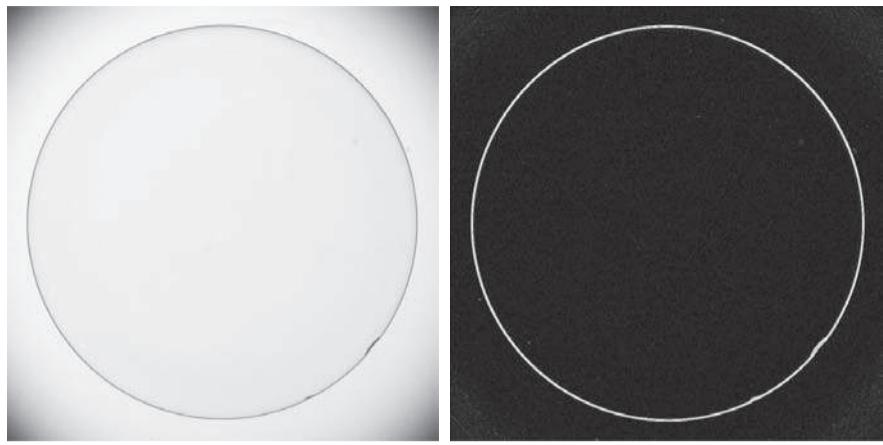
$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

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

$$M(x, y) = \left[g_x^2 + g_y^2 \right]^{\frac{1}{2}} = \left[[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2 \right]^{\frac{1}{2}}$$

There are more such pre-defined operators that can do edge enhancement (we will discuss these later).



(a) Image of a contact lens
 (b) Sobel gradient.

Unsharp masking and high boost filtering

We can subtract an unsharp (smoothed) version of an image from the original image to create a mask for sharpening the original image.

Steps:

Blur the original image $f(x, y) \Rightarrow \bar{f}(x, y)$

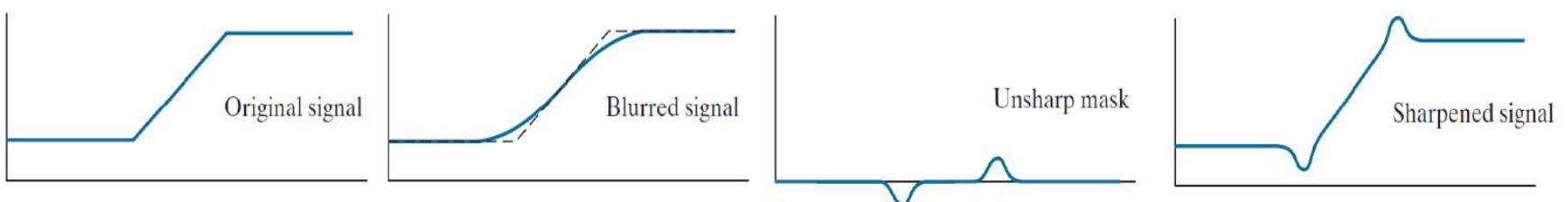
Subtract the blurred image from the original to create the unsharp mask:

$$g_{mask}(x, y) = f(x, y) - \bar{f}(x, y)$$

Add the mask to the original image ('k' is the control parameter)

$$g(x, y) = f(x, y) + k g_{mask}(x, y)$$

When $k = 1$ we have unsharp masking (defined above). $k > 1$, produces high-boost filtering. $k < 1$ reduces the contribution of the unsharp mask.





a) image

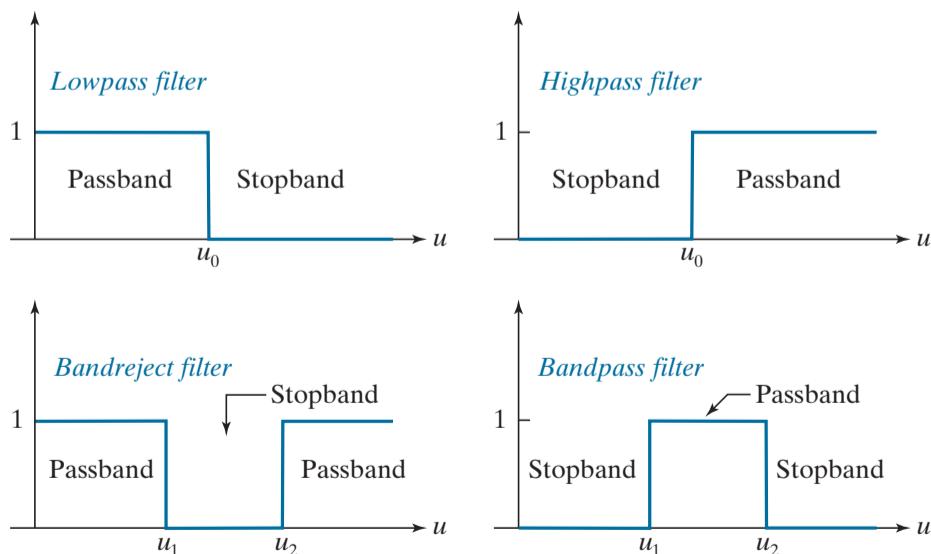
size 600×259 pixels. (b) Image blurred using a 31×31 Gaussian lowpass filter with $\sigma = 5$. (c) Mask. (d) $k = 1$ (sharpen). (e) high-boost filtering with $k = 4.5$.

High-pass, Band-reject and Bandpass Filters from Lowpass Filters

$$\xrightarrow{f(t)} g(t) = \mathcal{H}\{f(t)\} \xrightarrow{g(t)}$$

In ‘signals and system’, an abstract operator $H\{\cdot\}$ can be described completely using the impulse response $h(t)$ or transfer function $H(\xi)$. The impulse response is combined with the signal by the operation of convolution. This is sufficient to describe all linear shift-invariant systems.

We use the transfer function of a 1-D ideal lowpass filter in the frequency domain. A constant in the frequency domain is an impulse in the spatial domain. It is easier to understand the filtering in the frequency domain.



Transfer functions of ideal 1-D filters in the frequency domain (u denotes frequency). Shown only positive frequencies for simplicity.

A highpass filter transfer function $\Rightarrow 1 - \text{lowpass function(frequency domain)}$.

In the spatial domain,

highpass filter kernel= unit impulse kernel -lowpass filter kernel

Or

high boost filter=original image -lowpass filter kernel. \Rightarrow unsharp mask filter

Filter type	Spatial kernel in terms of lowpass kernel, lp
Lowpass	$lp(x, y)$
Highpass	$hp(x, y) = \delta(x, y) - lp(x, y)$
Bandreject	$br(x, y) = lp_1(x, y) + hp_2(x, y)$ $= lp_1(x, y) + [\delta(x, y) - lp_2(x, y)]$
Bandpass	$bp(x, y) = \delta(x, y) - br(x, y)$ $= \delta(x, y) - [lp_1(x, y) + [\delta(x, y) - lp_2(x, y)]]$

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-314

Advanced techniques for edge detection

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Laplacian of a Gaussian (LoG) detector

The LoG detector was proposed since the intensity changes are not independent of image scale.

Two salient features of an operator were proposed:

- i) it should be a differential operator capable of computing a digital approximation of the first or second derivative at every point in the image.
- ii) it should be capable of being “tuned” to act at any desired scale, so that large operators can be used to detect blurry edges and small operators to detect sharply focused fine detail.

The filter of the form $\nabla^2 G$ where, G is the 2-D Gaussian function:

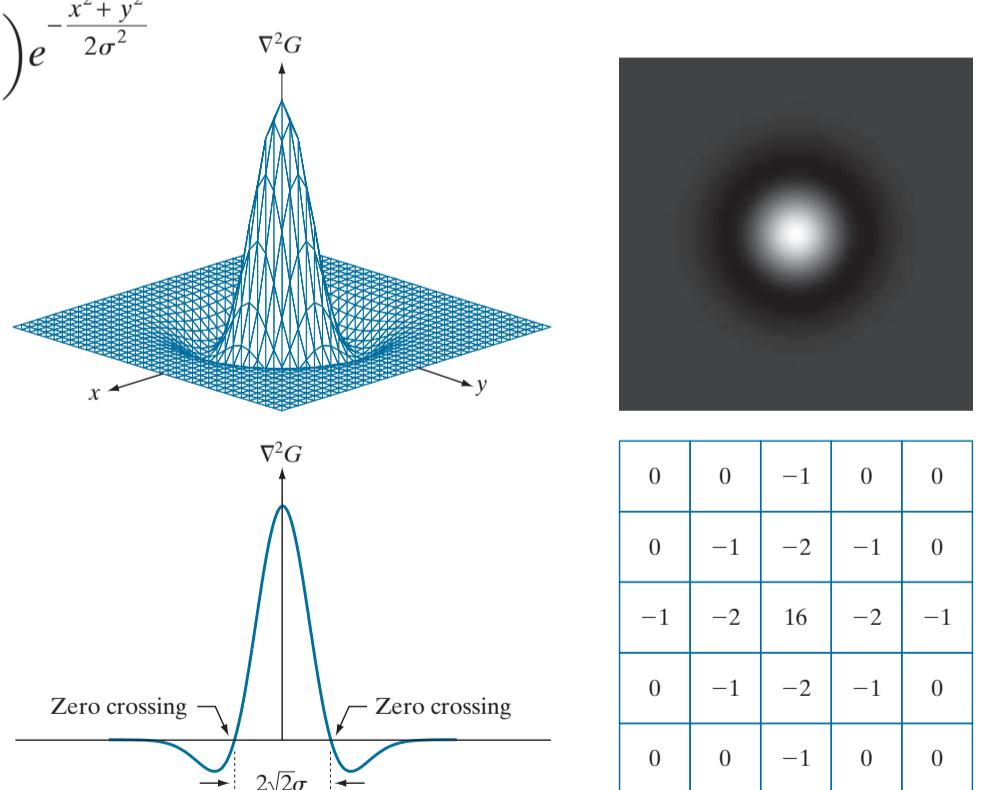
with standard deviation σ . Applying ∇^2 (the Laplacian on G):

$$\begin{aligned}\nabla^2 G(x, y) &= \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} \\ &= \frac{\partial}{\partial x} \left(\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) + \frac{\partial}{\partial y} \left(\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) \\ &= \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} + \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}\end{aligned}$$

$$\nabla^2 G(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

(a) 3-D plot of the negative of the LoG. (b) Negative of the LoG displayed as an image.

(c) Cross section of (a) showing zero crossings. Note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centred on the peak of the Gaussian function). (d) 5X5 kernel approximation to the shape in (a). negative of this kernel would be used in practice.



Other names of LoG filter: Marr-Hildreth edge-detection (who found the method) & Mexican-hat filter (due to the shape of the kernel).

Implementation is convolving the LoG kernel with an input image, and then finding the zero crossings of $g(x, y)$ to determine the locations of edges in $f(x, y)$.

$$g(x, y) = [\nabla^2 G(x, y)] \star f(x, y)$$

Because the Laplacian and convolution are linear processes, we can first smoothen & then compute the Laplacian:

$$\nabla^2(f(x, y) \otimes G(x, y)) = \nabla^2 G(x, y) \otimes f(x, y)$$

```

graph LR
    subgraph Left [Image smoothing/filtering]
        1_1[1. Image smoothing/filtering]
        1_2[2. Laplacian of Gaussian]
        1_1 --- 1_2
    end
    subgraph Right [Laplacian of Gaussian]
        2_1[1. Laplacian of Gaussian]
        2_2[2. Image smoothing/filtering]
        2_1 --- 2_2
    end
    Left <-->|Linear operation| Right
  
```

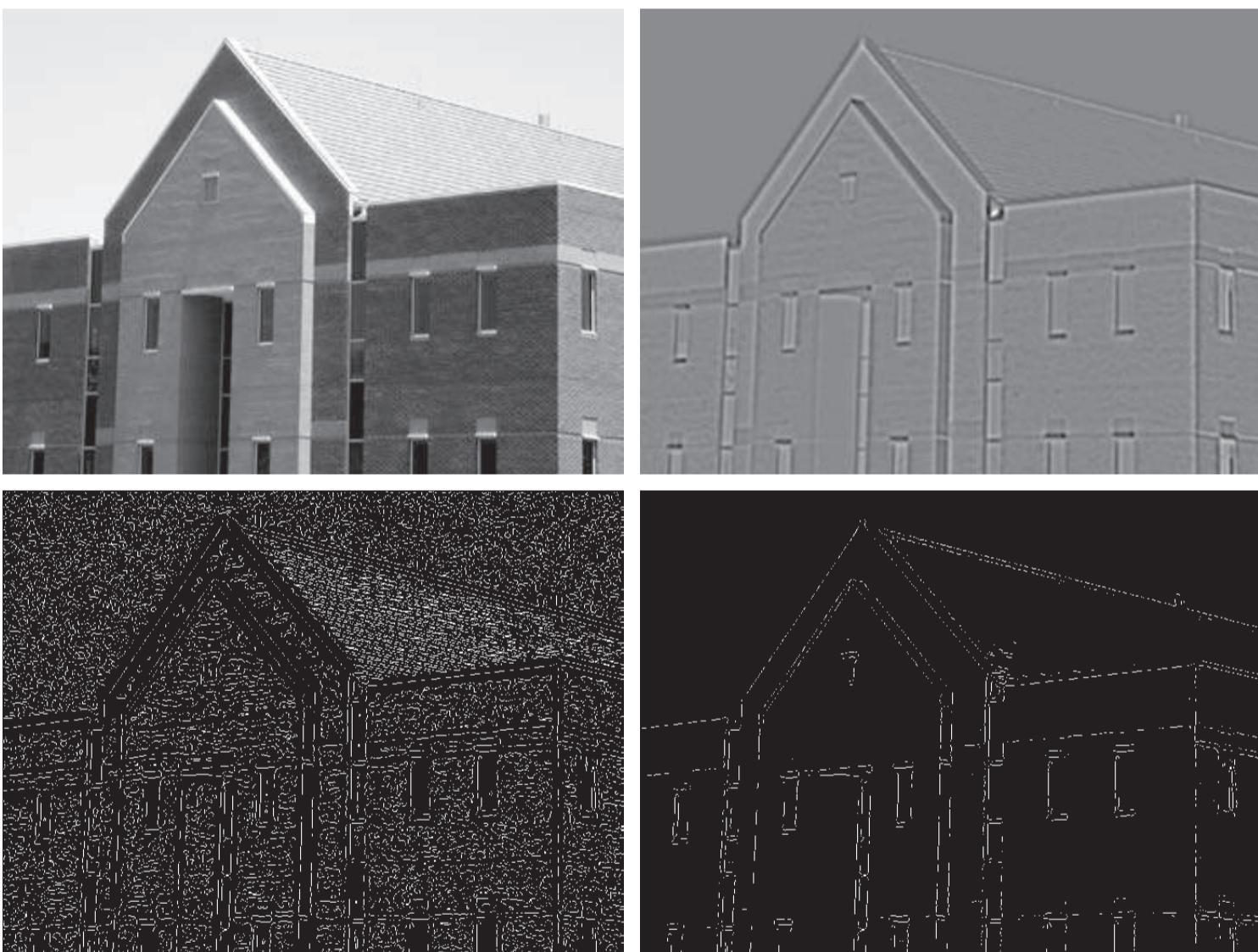
Here again the size of the Gaussian kernel matters, just like it was implemented for smoothing:

- Gaussian part of the operator blurs the image, thus reducing the intensity of structures (including noise) at scales much smaller than σ
- Kernels are of odd dimensions and therefore the size of the Gaussian kernel chosen is $6\sigma \times 6\sigma$, where the result is rounded to the nearest odd integer. Eg, if $\sigma=7$, then the kernel size is 43X43

Finding the zero crossings at any pixel

- Let us say, we have already filtered the image ($g(x, y)$), and now we need to find the zero crossings at any pixel, p .
- We use a 3X3 neighbourhood centred at p

- a zero crossing at p implies that the signs of at least two of its opposing neighbouring pixels must differ leading to four cases : left/right, up/down, and the two diagonals.
- By using a threshold, we compare the value at p in $g(x,y)$ to the possible neighbours.
- So, we check two properties- i) signs of opposing neighbours be different, ii) the absolute value of their numerical difference must also exceed the threshold before we can call p a zero-crossing pixel.



- (a) Image of size 834X1114 pixels, with intensity values scaled to the range [0, 1].
(b) Result of Steps 1 and 2 of the LoG filter using $\sigma = 4$ and $n = 25$ for Gaussian filter kernel size ($n \times n$). (c) Zero crossings of (b) using a threshold of 0 or rather say no thresholding (note the closed-loop edges). (d) Zero crossings using a threshold equal to 4% of the maximum value of the image in (b). Note the thin edges.

We have refined the edge detection by including a thresholding step while detecting the zero-crossing.

Let us see some examples where LoG is used as an edge detector (without any thresholding):

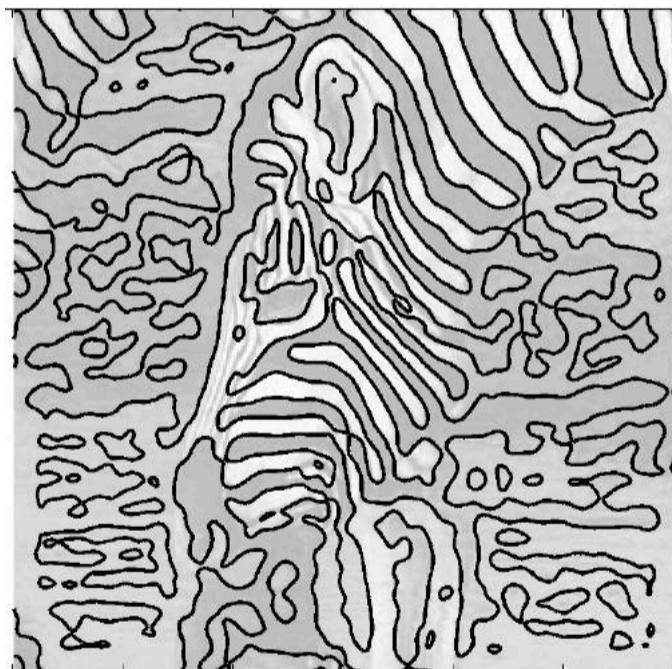
grayscale image



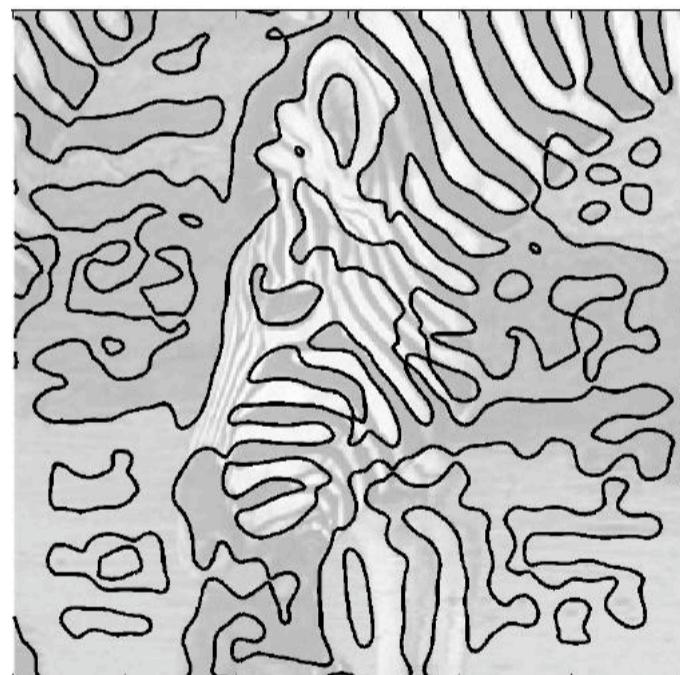
LoG $\sigma=2$



LoG $\sigma=4$

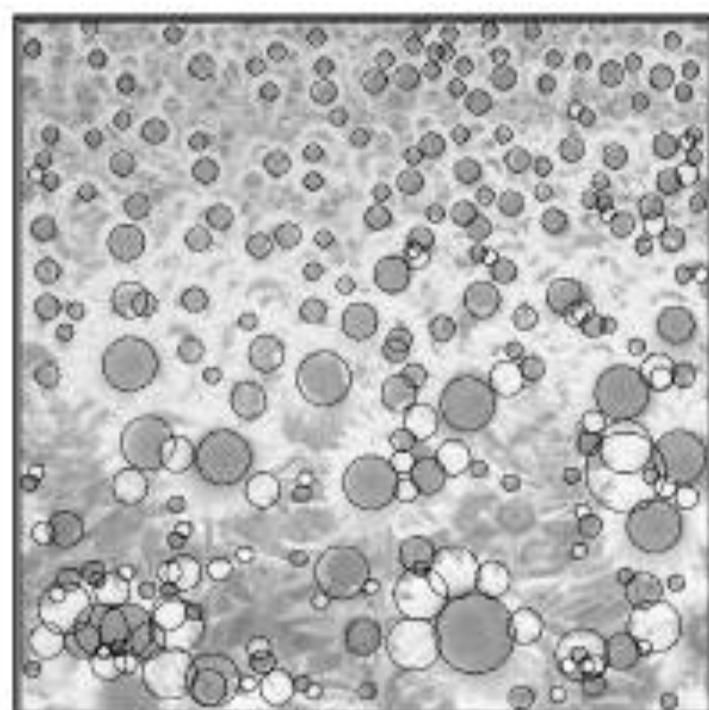


LoG $\sigma=8$



LoG detector for blob detection (scale-space detector)

The scale of blob size (radius in pixels) is determined by σ parameter



LoG filter extrema locates “blobs”

maxima = dark blobs on light background

minima = light blobs on dark background

How do we detect maxima or minima?

a point is regarded as a bright (dark) blob if the value at this point is greater (smaller) than the value in all its (26) neighbours.

To check if the same pixel belongs to the same blob or not:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} > \text{max } r$$

Set $r=2\sigma$, then if the distance between two blobs centres are smaller than ‘ r ’, they belong to the same blob, else not.

Using LoG operator for scale-space representation:

The size of the Gaussian kernel used for pre-smoothing in the LoG operator also determines the size to structures within an image. An image consisting of different sizes, therefore needs to be handled by setting the size of the Gaussian kernel itself. This is done by choosing several values of the σ parameter and identifying the pixels that are blobs for all the σ . With this approach, we set-up a multi-scale blob detector with an automatic scale selection that considers the scale-normalised Laplacian operator:

$$\nabla_{\text{norm}}^2 L = t(L_{xx} + L_{yy})$$

Where t is the scale. The operator computes the LoG at different scales ' t ', where the blob radius size r is s.t. $r = \sqrt{dt}$

Here 'd' is image dimension. Simultaneous selection of interest points (\hat{x}, \hat{y}) and scales \hat{t}

is performed according to $(\hat{x}, \hat{y}; \hat{t}) = \text{argmaxminlocal}_{(x,y;t)}((\nabla_{\text{norm}}^2 L)(x, y; t))$

If a scale-space maximum is assumed at a point , $(x_0, y_0; t_0)$

Then, under a rescaling of the image by a scale factor s , there will be a scale-space maximum at $(sx_0, sy_0; s^2t_0)$ in the rescaled image.

Scale-space as image descriptor

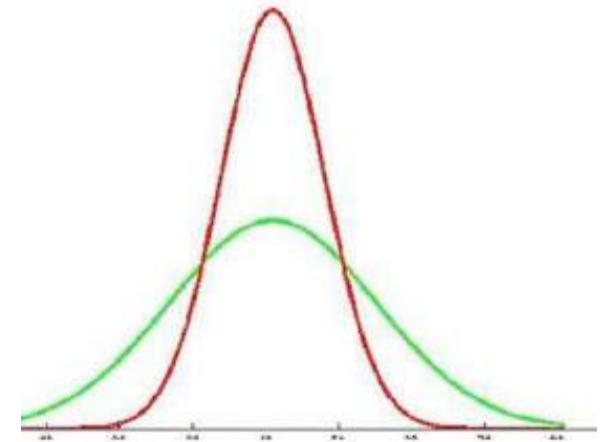
Local maxima/minima of the scale-normalised Laplacian used for scale selection in other contexts, such as in corner detection, scale-invariant feature transform as well as other image descriptors for image matching and object recognition.

Difference of Gaussians (DoG)

A related technique is the DoG detector which is an approximate go the LoG function:

$$D_G(x, y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}}$$

$$\nabla^2 G_\sigma \approx G_{\sigma_1} - G_{\sigma_2} \quad \text{with } \sigma_1 > \sigma_2.$$

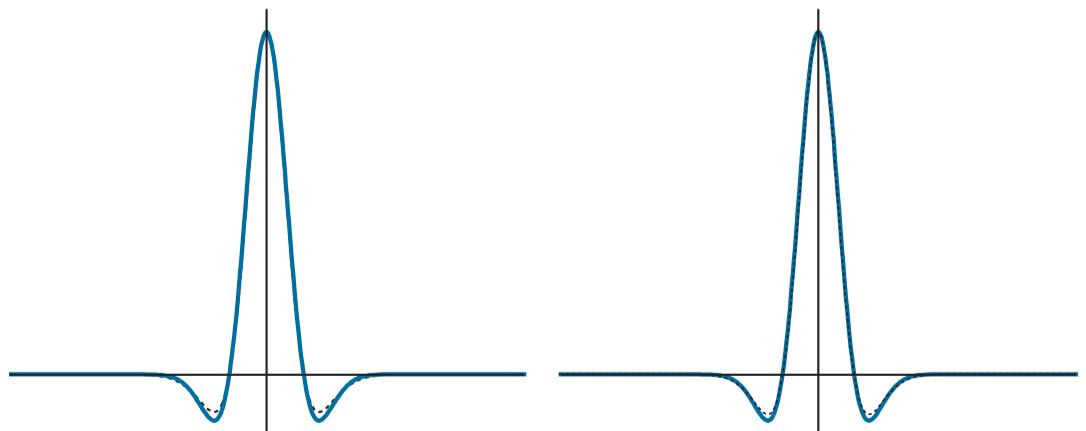


- Keeping a ratio of standard deviations of 1.75:1 for σ_1 & σ_2 , approximately is similar to the selective approach of the human vision system with respect to orientation and frequency.
- Using the ratio 1.6:1 preserves the basic characteristics of the LoG function.
- In order for the LoG and DoG to have the same zero crossings, the value of σ for the LoG must be selected based on the following equation:

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 - \sigma_2^2} \ln \left[\frac{\sigma_1^2}{\sigma_2^2} \right]$$

Although the zero crossings of the LoG and DoG will be the same when this value of σ is used, their amplitude scales will be different.

The profiles generated with standard deviation ratios of 1:1.75 and 1:1.6, respectively (by convention, the curves shown are inverted). The LoG profiles are the solid lines, and the DoG profiles are dotted. The 1:1.6 ratio is a closer approximation of the LoG and DoG functions.

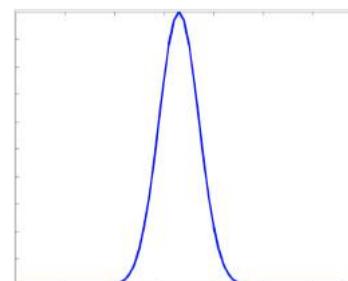


Computational cost:

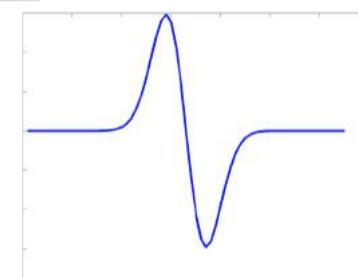
Gaussian kernels are separable, therefore, both the LoG and the DoG filtering operations can be implemented with 1-D convolutions instead of using 2-D convolutions directly. For an image of size $M \times N$ and a kernel of size $n \times n$, doing so reduces the number of multiplications and additions for each convolution from being proportional to $n^2 M N$ for 2-D convolutions to being proportional to $n M N$ for 1-D convolutions.

1D Gaussian and its Derivatives:

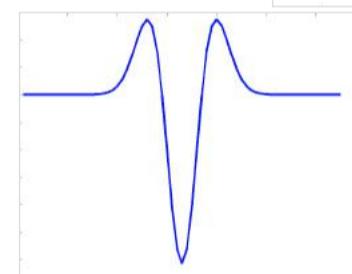
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



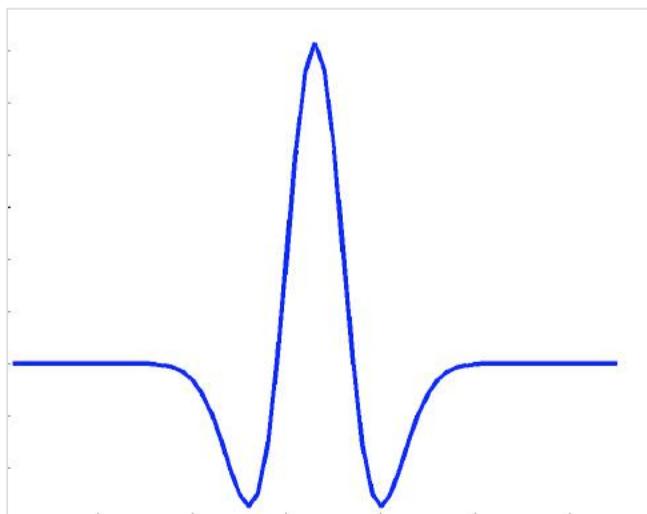
$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$



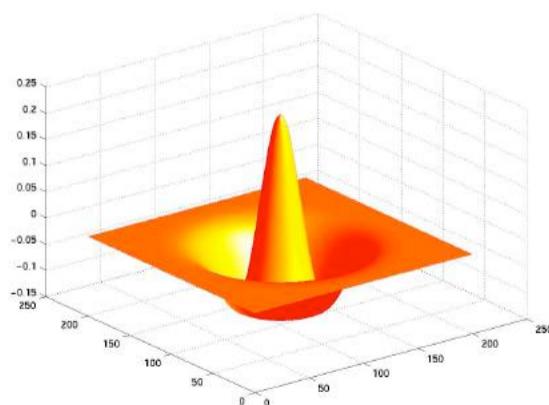
$$g''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{-\frac{x^2}{2\sigma^2}}$$



The second derivative of Gaussian ($g''(x)$) is analogous to the LoG filter



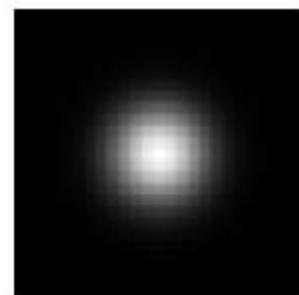
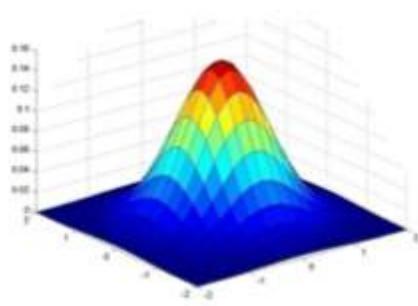
2D
analog
→



LoG "Mexican Hat"

Canny edge detector

1. Gaussian smoothing of the image (using convolution)



$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

2. Compute the gradient magnitude and direction (angle) of the image

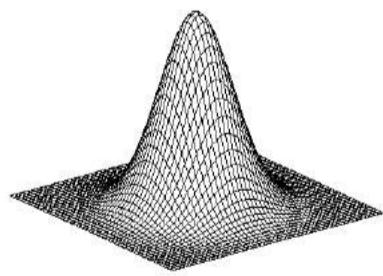
$$M_s(x, y) = \|\nabla f_s(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right]$$

S.t. $g_x(x, y) = \partial f_s(x, y)/\partial x$ $g_y(x, y) = \partial f_s(x, y)/\partial y$

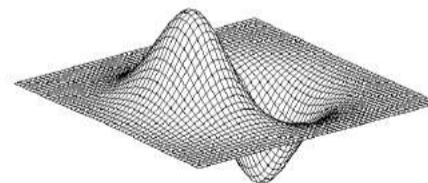
Here, Roberts, Sobel or Prewitt kernels can be used on the image to compute the gradient.

1-2. Alternatively directly compute the Gauss gradient of the image



Gaussian

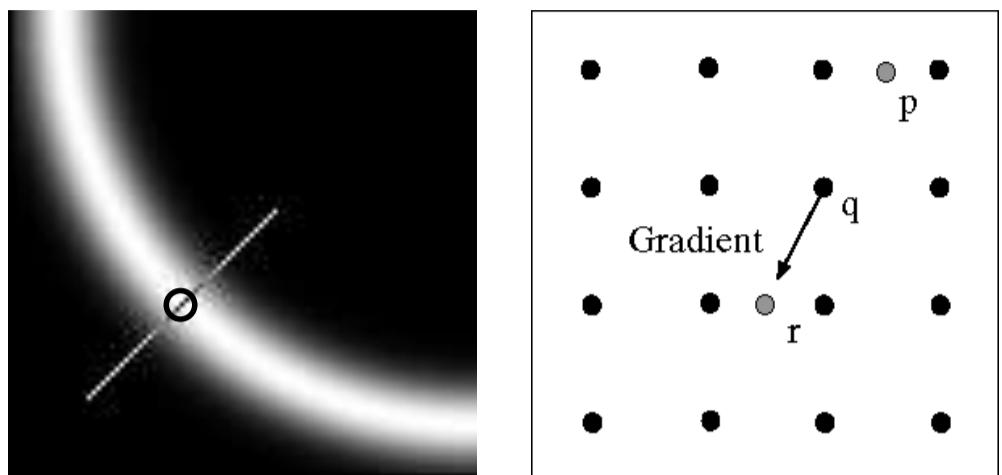
$$h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



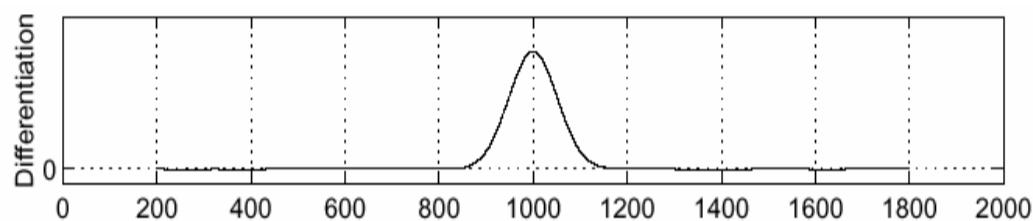
derivative of Gaussian (x)

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

3. Find magnitude and orientation of gradient non-maximum suppression i.e. check if the pixel is local maximum along gradient direction.



For this we use both magnitude and the direction within a local neighbourhood within the image. Along the orientation, we look for the maximal value within the gradient image, choose that as a value and a 0 elsewhere.



4. Thresholding appropriately to reduce false edge points, called as hysteresis thresholding.

Linking and thresholding (hysteresis): We set a low threshold, T_L and a high threshold, T_H . such that the ratio of the high to low threshold should be in the range of 2:1 to 3:1.

Let $g_N(x,y)$ be the output of previous step i.e. non-maximum suppression

$$g_{NH}(x,y) = g_N(x,y) \geq T_H$$

$$g_{NL}(x,y) = g_N(x,y) \geq T_L$$

After thresholding, $g_{NH}(x, y)$ will usually have fewer nonzero pixels than $g_{NL}(x, y)$, but all the nonzero pixels in $g_{NH}(x, y)$ will be contained in $g_{NL}(x, y)$ because the latter image is formed with a lower threshold. We eliminate from $g_{NL}(x,y)$ all the nonzero pixels from $g_{NH}(x,y)$ by letting

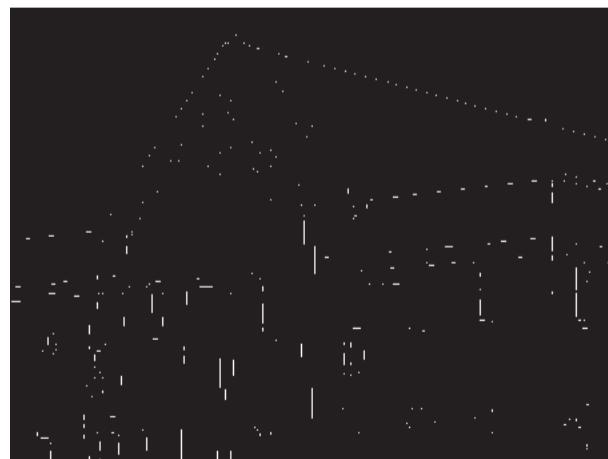
$$g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$$

For connectivity analysis to link edges: Use the high threshold to start edge curves and the low threshold to continue them.

The nonzero pixels of $g_{NH}(x,y)$ and $g_{NL}(x,y)$ are taken to be “strong” and “weak” edge pixels, respectively.

After the thresholding operations, all strong pixels in $g_{NH}(x,y)$ are assumed to be valid edge pixels, and are so marked immediately. However the edges in $g_{NH}(x,y)$ typically have gaps. To connect the edges, longer edges are formed using the following procedure:

- a. Locate the next unvisited edge pixel, p , in $g_{NH}(x,y)$.
- b. Mark as valid edge pixels all the weak pixels in $g_{NL}(x,y)$ that are connected to p using, 8-connectivity.
- c. If all nonzero pixels in $g_{NH}(x,y)$ have been visited go to Step (d). Else, return to Step (a).
- d. Set to zero all pixels in $g_{NL}(x,y)$ that were not marked as valid edge pixels.



(a) Original image of size 834X1114 pixels, with intensity values scaled to the range [0, 1].

(b) Thresholded gradient of the smoothed image.

(c) the LoG detector .

(d) Canny algorithm. Note the significant improvement of the Canny image compared to the other two.

COURSE
DIGITAL IMAGE PROCESSING AND APPLICATIONS IN BIOIMAGE ANALYSIS

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2021
DSE-409/609

Chapter: Segmentation

WEEK-13

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Image Segmentation

Image segmentation is to partition an image into regions.

The fundamental definition of segmentation should meet the following conditions:

Every pixel must be in a region, meaning the segmentation must be complete for the entire image.

$$\bigcup_{i=1}^n R_i = R.$$

The points in a region be connected in some predefined sense (e.g., the points must be 8-connected).

R_i is a connected set, for $i = 0, 1, 2, \dots, n$.

The segmented regions must be disjoint.

$R_i \cap R_j = \emptyset$ for all i and j , $i \neq j$.

Within a segmented region $Q(R_i) = \text{TRUE}$, if all pixels in R_i have the same intensity.

$Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \dots, n$.

where $Q(R_k)$ is a logical predicate defined over the points in set R_k .

The two adjacent regions R_i and R_j must be different in the sense of predicate

$Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j .

Segmentation algorithms for grayscale images are based on one of two basic categories dealing with properties of intensity values:

i) discontinuity

eg: boundaries of regions are sufficiently different from each other, and from the background, to allow boundary detection based on local discontinuities in intensity (edge-based segmentation).

ii) similarity

Region-based segmentation approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

Point, Line and Edge detection

- These methods are based on detecting sharp, local changes in intensity.
- The three types of image characteristics are isolated points, lines, and edges.
- As we have already seen, local averaging smoothes an image. The abrupt, local changes in intensity can be detected using derivatives. Therefore, first- and second-order derivatives are particularly well suited for this purpose.
- Edge pixels are pixels at which the intensity of an image changes abruptly, and edges (or edge segments) are sets of connected edge pixels.
- Edge detectors are local image processing tools designed to detect edge pixels.
- A line may be viewed as a (typically) thin edge segment in which the intensity of the background on either side of the line is either much higher or much lower than the intensity of the line pixels.
- An isolated point may be viewed as a foreground (background) pixel surrounded by background (foreground) pixels.

Please revise the image derivatives again. We studied them already during spatial filtering. I have put few points very briefly here.

Derivatives of a digital function are defined in terms of finite differences.

Approximation used for first derivatives:

- (1) must be zero in areas of constant intensity;
- (2) must be nonzero at the onset of an intensity step or ramp; and
- (3) must be nonzero at points along an intensity ramp.

Approximation used for second derivatives:

- (1) must be zero in areas of constant intensity;
- (2) must be nonzero at the onset and end of an intensity step or ramp; and
- (3) must be zero along intensity ramps.

Because we are dealing with digital quantities whose values are finite, the maximum possible intensity change is also finite, and the shortest distance over which a change can occur is between adjacent pixels.

We obtain an approximation to the first-order derivative at an arbitrary point x of a one-dimensional function $f(x)$ by expanding the function $f(x + \Delta x)$ into a Taylor series about x

$$\begin{aligned}f(x + \Delta x) &= f(x) + \Delta x \frac{\partial f(x)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\&= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} \frac{\partial^n f(x)}{\partial x^n}\end{aligned}$$

where Δx is the separation between samples of f . For our purposes, this separation is measured in pixel units. Thus, following the convention in the book, $\Delta x = 1$ for the sample preceding x and $\Delta x = -1$ for the sample following x .

When $\Delta x = 1$,

$$\begin{aligned}f(x+1) &= f(x) + \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\&= \sum_{n=0}^{\infty} \frac{1}{n!} \frac{\partial^n f(x)}{\partial x^n}\end{aligned}$$

When $\Delta x = -1$,

$$\begin{aligned} f(x-1) &= f(x) - \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} - \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned}$$

In what follows, we compute intensity differences using just a few terms of the Taylor series. For first-order derivatives we use only the linear terms, and we can form differences in one of three ways.

The forward difference is (using only linear terms): $\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x)$

The backward difference is $\frac{\partial f(x)}{\partial x} = f'(x) = f(x) - f(x-1)$

The central difference is $\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x-1)}{2}$

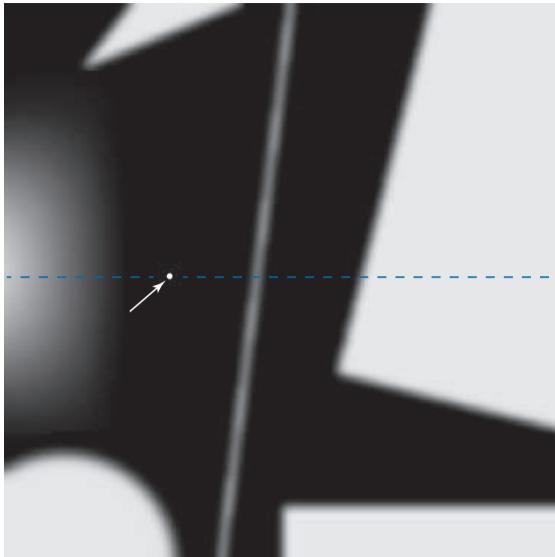
The higher terms of the Taylor series represent the error between an exact and an approximate derivative expansion. In general, the more terms we use from the Taylor series to represent a derivative, the more accurate the approximation will be.

However, it is a general practice to use central differences that have a lower error

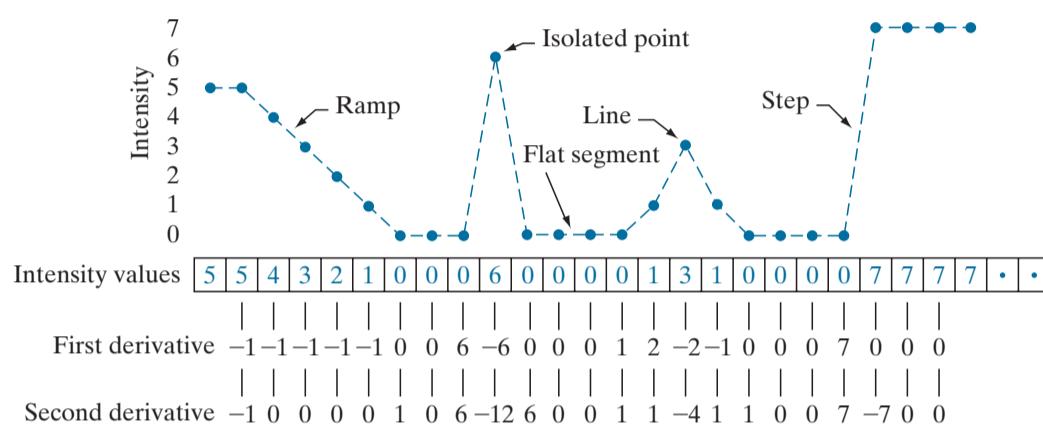
The second order derivative based on a central difference:

$$\frac{\partial^2 f(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1)$$

	$f(x+2)$	$f(x+1)$	$f(x)$	$f(x-1)$	$f(x-2)$
$2f'(x)$		1	0	-1	
$f''(x)$		1	-2	1	
$2f'''(x)$	1	-2	0	2	-1
$f''''(x)$	1	-4	6	-4	1



First four central digital derivatives (finite differences) for samples taken uniformly, $\Delta x = 1$ units apart.



(a) Image. (b) Horizontal intensity profile that includes the isolated point indicated by the arrow. (c) Subsample profile; the dashes were added for clarity. The numbers in the boxes are the intensity value of the dots shown

in the

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x)$$

$$\frac{\partial^2 f(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1)$$

(1) First-order derivatives generally produce thicker edges.

(2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise.

(3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity.

(4) The sign of the second derivative can be used to determine whether a transition into an edge is from light to dark or dark to light.

For the 3X3 filter kernel, we compute the sum of products of the kernel coefficients with the intensity values in the region encompassed by the kernel, (using spatial convolution). The response of the filter at the center point of the kernel is:

where z_k is the intensity of the pixel whose spatial location corresponds to the location of the k th kernel coefficient.

$$\begin{aligned} Z &= w_1 z_1 + w_2 z_2 + \dots + w_9 z_9 \\ &= \sum_{k=1}^9 w_k z_k \end{aligned}$$

Detection of isolated points

The second derivative using the Laplacian can be used for point detection:

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

where the partial derivatives are computed using the second-order finite differences. The Laplacian is then

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

The above expression can be implemented using the Laplacian kernel that measures the weighted differences between a pixel and its 8-neighbours.

A ‘point’ is considered detected at a location (x, y) on which the kernel is centred if the absolute value of the response of the filter at that point exceeds a specified threshold (T). Where,

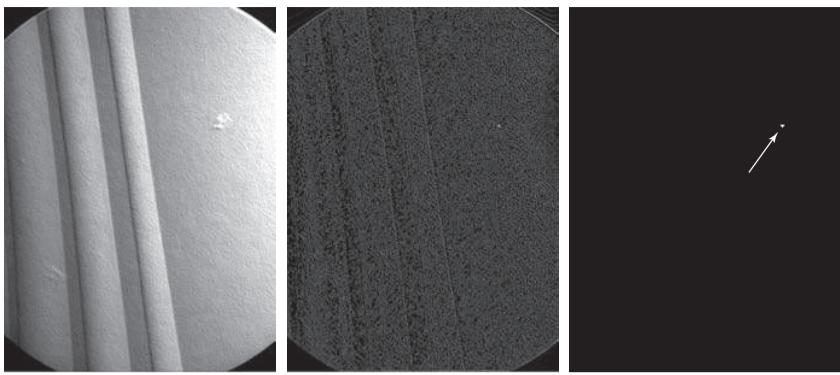
$$g(x, y) = \begin{cases} 1 & \text{if } |Z(x, y)| > T \\ 0 & \text{otherwise} \end{cases} \quad \begin{aligned} Z &= w_1 z_1 + w_2 z_2 + \dots + w_9 z_9 \\ &= \sum_{k=1}^9 w_k z_k \end{aligned}$$

where $g(x, y)$ is the output image (binary), T is a nonnegative threshold.

Intuitively, the idea is that the intensity of an isolated point will be quite different from its surroundings, and thus will be easily detectable by this type of kernel.

1	1	1
1	-8	1
1	1	1

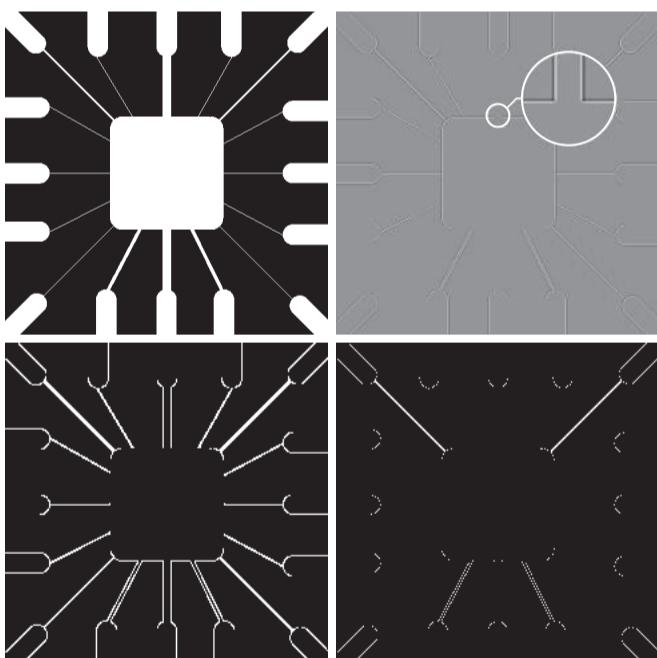
kernel with point.



- (a) Laplacian kernel for point detection (coefficients sum is zero).
- (b) X-ray image of a turbine blade with a porosity(single black pixel).
- (c) convolving the image detects the point.

Line detection

The second derivatives such as Laplacian can be used for line detection. The Laplacian detector kernel shown above is isotropic, so its response is independent of direction (with respect to the four directions of the 3X3 kernel: vertical, horizontal, and two diagonals). The only thing to keep in mind, is the double-line effect of the second derivative. be handled properly. The following example illustrates the procedure.



- (a) Original image.
- (b) Laplacian image; the magnified section shows the positive/negative double-line effect characteristic of the Laplacian.
- (c) Absolute value of the Laplacian.
- (d) Positive values of the Laplacian.

Another method uses a series of kernels oriented in different directions to detect line within an image. For an image with a constant background and containing various lines oriented at different directions, a series of kernels oriented in different directions maybe used to detect line within the image. The coefficients in each kernel sum to zero, indicating a zero response in areas of constant intensity.

Let Z_1, Z_2, Z_3, Z_4 denote the responses of the kernels from left to right, where

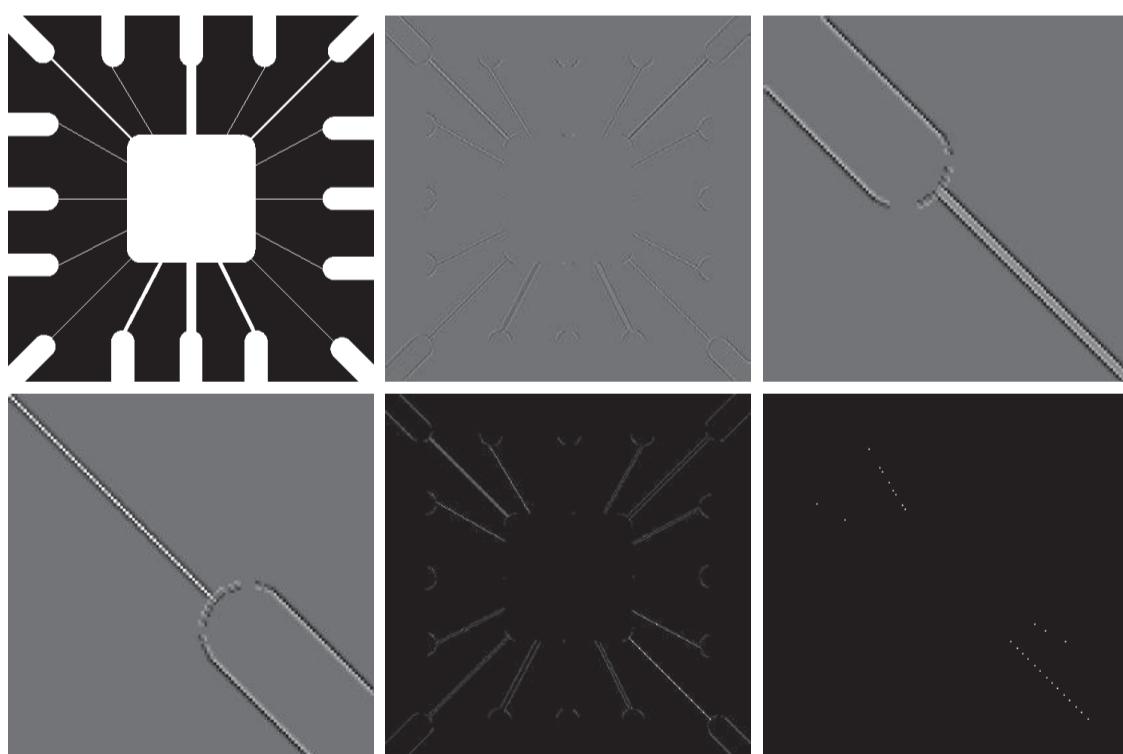
$$Z = \sum_{k=1}^9 w_k z_k$$

When an image is filtered with these four kernels, one at a time, if, at a given point in the image, $|Z_k| > |Z_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of kernel k . For example, if at a point in the image, $Z_1 > Z_j$ for $j = 2, 3, 4$, that point is said to be more likely associated with a horizontal line.

$\begin{matrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{matrix}$	$\begin{matrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{matrix}$	$\begin{matrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{matrix}$	$\begin{matrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{matrix}$
Horizontal	$+45^\circ$	Vertical	-45°

(vertical) x-axis.

Line detection kernels.
Detection angles are with respect to the axis system, with positive angles measured counterclockwise with respect to the



(a) Image of a wire-bond template. (b) Result of processing with the $+45^\circ$ line detector kernel. (c) Zoomed view of the top left region of (b). (d) Zoomed view of the bottom right region of (b). (e) The image in (b) with all negative values set to zero. (f) All points (in white) whose values satisfied the condition $g > T$, where g is the image in (e) and $T = 254$ (the maximum

pixel value in the image minus 1). (The points in (f) were enlarged to make them easier to see.)

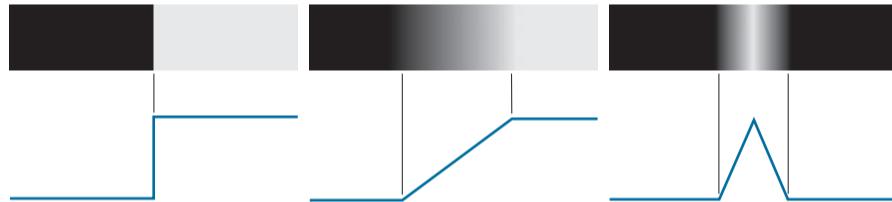
Edge detection

Please refer your notes on derivatives within the spatial filtering chapters as well when you study this section.

Edge detection is an approach used frequently for segmenting images based on abrupt (local) changes in intensity.

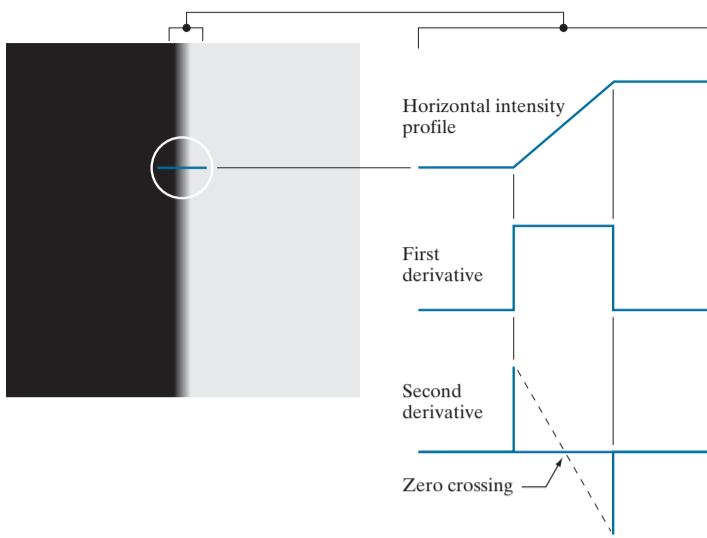
Edge models are classified according to their intensity profiles.

- A step edge is characterised by a transition between two intensity levels occurring ideally over the distance of one pixel.
- Ramp edges are blurred and noisy closely modeled as having an intensity ramp profile.
- A third type of edge is the roof edge modeled as lines through a region, with the base (width) of the edge being determined by the thickness and sharpness of the line.



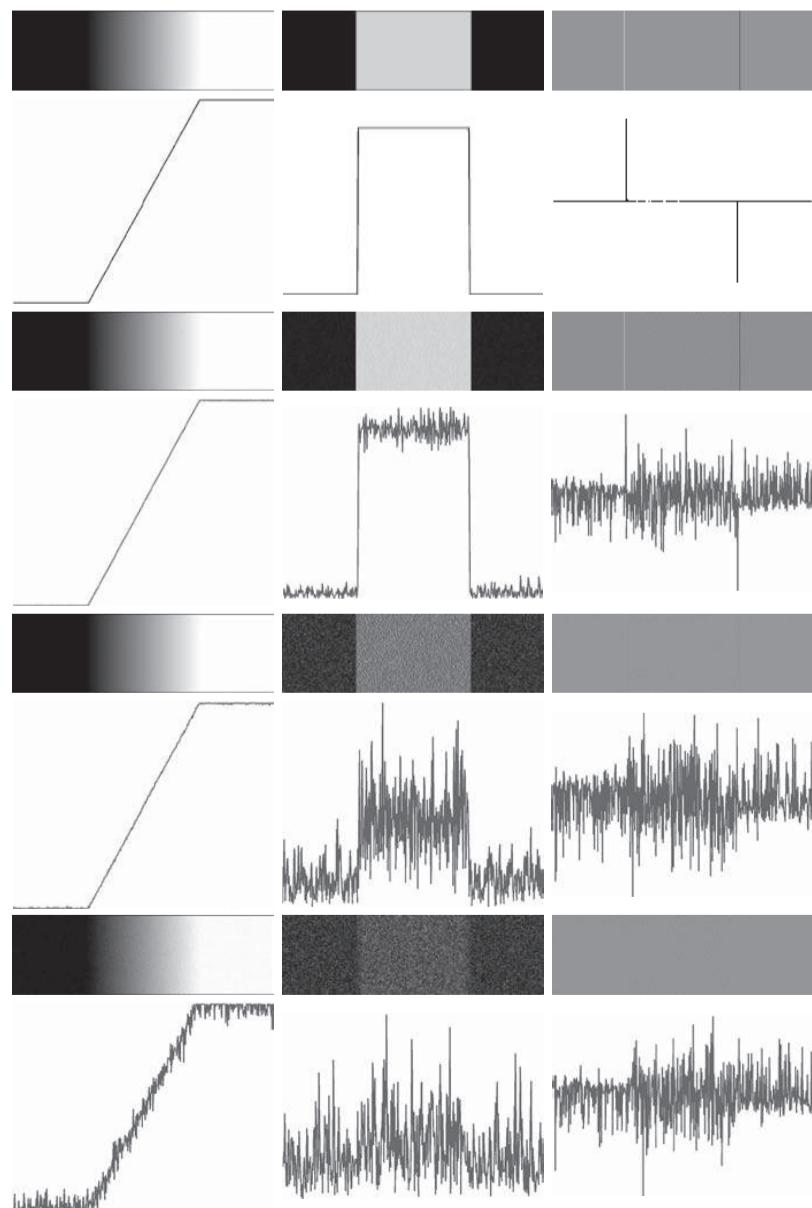
From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

- We write mathematical expressions for different types of ‘edges’ in the development of image processing algorithms.
- The magnitude of the first derivative can be used to detect the presence of an edge at a point in an image.
- The sign of the second derivative can be used to determine whether an edge pixel lies on the dark or light side of an edge.
- Two additional properties of the second derivative around an edge are: (1) it produces two values for every edge in an image; and (2) its zero crossings can be used for locating the centers of thick edges.



In summary, the three steps are performed typically for edge detection are:

1. Image smoothing for noise reduction.
2. Detection of edge points. A local operation that extracts from an image all points that are potential edge-point candidates.
3. Edge localisation. To select from the candidate points only the points that are members of the set of points comprising an edge.



First column: 8-bit images with values in the range [0, 255], and intensity profiles of a ramp edge corrupted by Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

Edge detection using image gradients:

Refer to previous notes and your assignments:

Expression for 1st derivative, the kernels (Prewitt, Sobel and Roberts).

Kirsch compass kernels

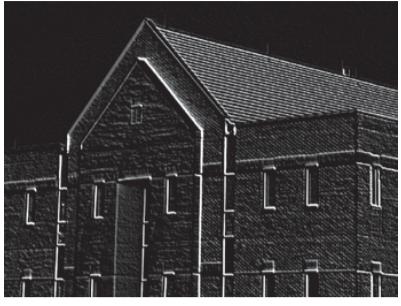
The 3X3 kernels for Prewitt, Sobel and Roberts exhibit their strongest response predominantly for vertical and horizontal edges.

- The Kirsch compass kernels are designed to detect edge magnitude and direction (angle) in all eight compass directions.
- Instead of using the standard image gradient magnitude and angle Kirsch's approach determined the edge magnitude by convolving an image with all eight kernels and assign the edge magnitude at a point as the response of the kernel that gave strongest convolution value at that point.
- The edge angle at that point is then the direction associated with that kernel.
- For example, if the strongest value at a point in the image resulted from using the north (N) kernel, the edge magnitude at that point would be assigned the response of that kernel, and the direction would be 0° .

Although with Sobel kernels, a north or south edge are treated as vertical, the N and S compass kernels here differentiate by looking at the direction of the intensity transitions defining the edge.

-3	-3	5	-3	5	5	5	5	-3
-3	0	5	-3	0	5	-3	0	-3
-3	-3	5	-3	-3	-3	-3	-3	-3
N			NW			W		
5	-3	-3	-3	-3	-3	-3	-3	-3
5	0	-3	5	0	-3	-3	0	5
5	-3	-3	5	5	-3	5	5	5
S			SE			E		
-3	-3	-3	-3	-3	-3	-3	-3	-3
-3	0	-3	5	0	-3	-3	0	5
-3	-3	-3	5	5	-3	5	5	5
NE			N			E		

Kirsch compass kernels. The edge direction of strongest response of each kernel is labeled below it.



(a) Image of size 834X1114 pixels, with intensity values scaled to the range [0,1]. Diagonal edge detection. (a) Result of using the Kirsch kernel (W).

(b) Result of using the kernel SW.

Please revise the following topics again (you already studied these during the spatial filtering lectures). Refer to previous notes and your assignments for LoG, DoG and canny edge detectors.

Marr and Hildreth edge detector/Laplacian of Gaussian detector (LoG)

The idea behind developing the LoG detector:

(1) that intensity changes are not independent of image scale, implying that their detection requires using operators of different sizes; and (2) that a sudden intensity change will give rise to a peak or trough in the first derivative or, equivalently, to a zero crossing in the second derivative.

Difference of Gaussian detector (DoG)

Canny edge detector

Linking edge points

Ideally, edge detection should yield sets of pixels lying only on edges. In practice, these pixels seldom characterise edges completely because of noise, breaks in the edges caused by nonuniform illumination, and other effects that introduce discontinuities in intensity values. Therefore, edge detection could be followed by linking algorithms designed to assemble edge pixels into meaningful edges and/or region boundaries.

Local Processing

A simple approach for linking edge points is to analyse the characteristics of pixels in a small neighbourhood about every point (x,y) that has been detected an edge point.

All points that are similar based on local analysis on the edges (1) the strength (magnitude) and (2) the direction of the gradient vector, are linked to form an edge.

Let S_{xy} denote the set of coordinates of a neighbourhood centred at point (x, y) in an image. An edge pixel with coordinates (s,t) in S_{xy} is similar in magnitude to the pixel at (x, y) if

where E is a positive threshold. $|M(s,t) - M(x,y)| \leq E$

The direction angle of the gradient vector. An edge pixel with coordinates (s,t) in S_{xy} has an angle similar to the pixel at (x, y) if

$$|\alpha(s,t) - \alpha(x,y)| \leq A$$

where A is a positive angle threshold. As noted earlier, the direction of the edge at (x, y) is perpendicular to the direction of the gradient vector at that point.

A pixel with coordinates (s,t) in S_{xy} is considered to be linked to the pixel at (x, y) if both magnitude and direction criteria are satisfied.

However this procedure is computationally expensive and may not be suitable for all. A simplification particularly well suited for real time applications consists of the following steps:

1. Compute $M(x, y)$ and $\alpha(x, y)$, of the input image, $f(x, y)$.
2. Form a binary image, $g(x, y)$, s.t. :

$$g(x,y) = \begin{cases} 1 & \text{if } M(x,y) > T_M \text{ AND } \alpha(x,y) = A \pm T_A \\ 0 & \text{otherwise} \end{cases}$$

where T_M is a threshold, A is a specified angle direction, and $\pm T_A$ defines a “band” of acceptable directions about A .

3. Scan the rows of g and fill (set to 1) all gaps (sets of 0's) in each row that do not exceed a specified length, L .
4. To detect gaps in any other direction, u , rotate g by this angle and apply the horizontal scanning procedure in Step 3. Rotate the result back by $-u$.

However, this procedure becomes computationally complex since image rotations are involved for find links in all orientations (radial scanning procedure).

Image of the rear of a vehicle.



(b) Gradient magnitude image. (c) Horizontally connected edge pixels. (d) Vertically connected edge pixels. The connected edge pixels obtained by the algorithm, with $T_M = 30\%$ of the maximum gradient value, $A = 90^\circ$, $T_A = 45^\circ$, and filling all gaps of 25 or fewer pixels (approximately 5% of the image width). (e) The logical OR of (c) and (d). (f) Final

result, using morphological thinning.

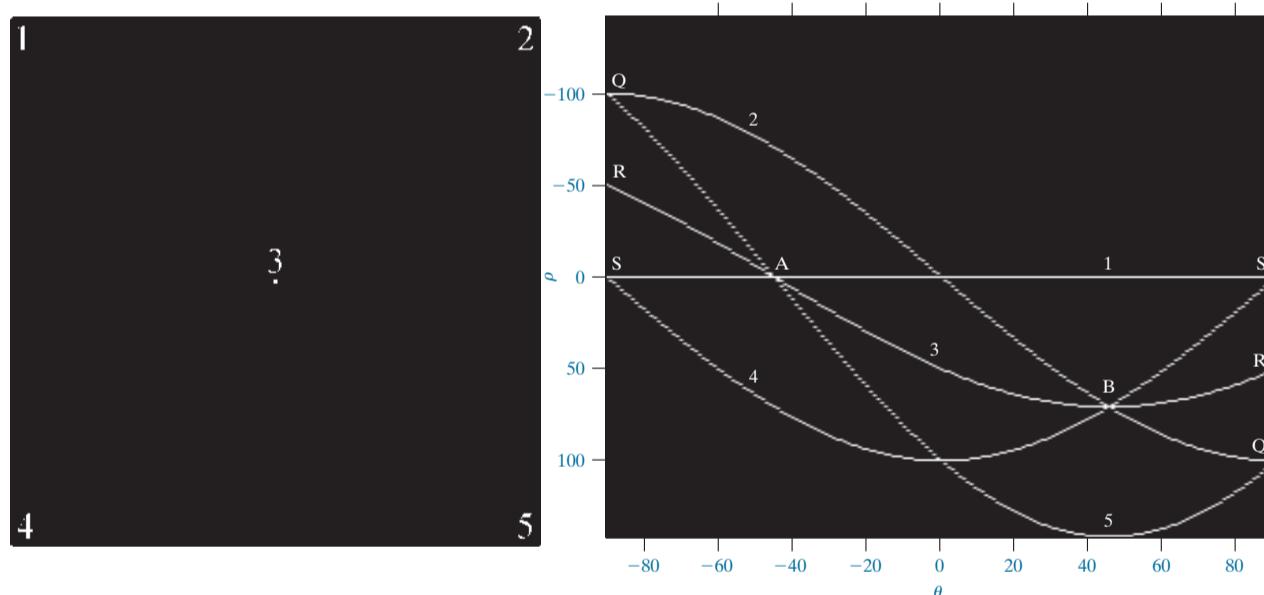
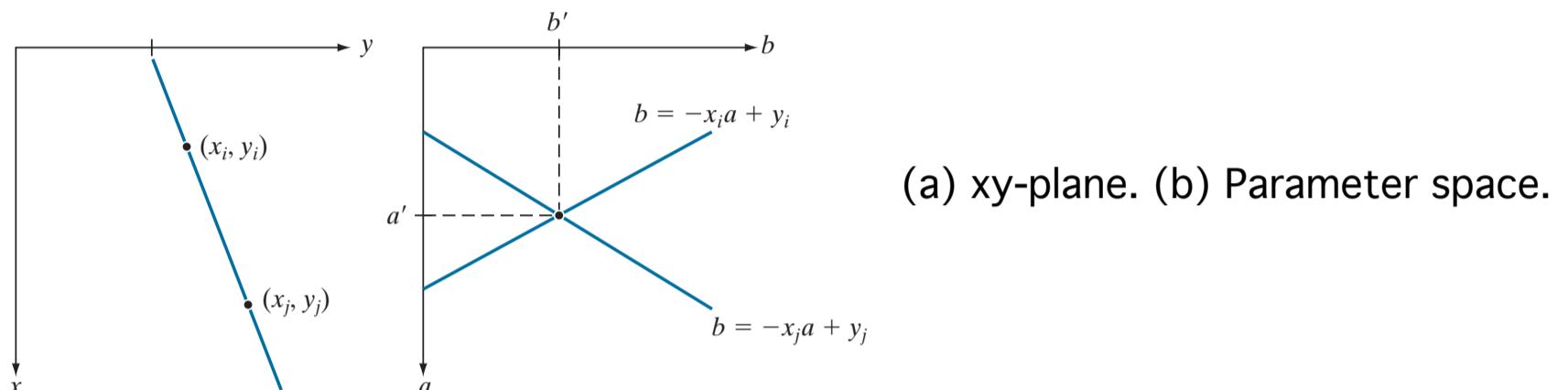
Hough transform

- The Hough transform can be used to isolate features of a particular shape within an image.
- It is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc.
- The technique is relatively unaffected by image noise.

- The Hough transform maps a straight line $y=mx+c$ in a Cartesian coordinate system into a single point in the (ρ, θ) plane such that

$$\rho = x\cos\theta + y\sin\theta$$
- For a point (x, y) in the Cartesian coordinate plane, there will be an infinite number of curves in the (ρ, θ) plane.

- When two points (x_i, y_i) and (x_j, y_j) lie on the same straight line, the curves in the (ρ, θ) plane which correspond, respectively, to the two points (x_i, y_i) and (x_j, y_j) in the Cartesian plane will intersect at a point. This intersection point determines the parameter of the line that joins these two points.
- Similarly, for the three collinear points this property between Cartesian plane and the parametric plane is useful in finding the line that fits points in the xy plane
- The Hough transform approach is to find the points of intersection in the curves, each of which corresponds to a line in the Cartesian xy plane



(a) Image of size 101×101 pixels, containing five white points (four in the corners and one in the center). (b) Corresponding parameter space for The range of u values is $\pm 90^\circ$, and the range of r values is $\pm \sqrt{(2M)}$.

Thresholding

Global thresholding on Intensity images

When the intensity histogram corresponds to an image, $f(x,y)$, composed of light objects on a dark background, in such a way that object and background pixels have intensity values grouped into two dominant modes, then we can extract the objects from the background and hence segment the image , denoted by $g(x, y)$, is given by

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

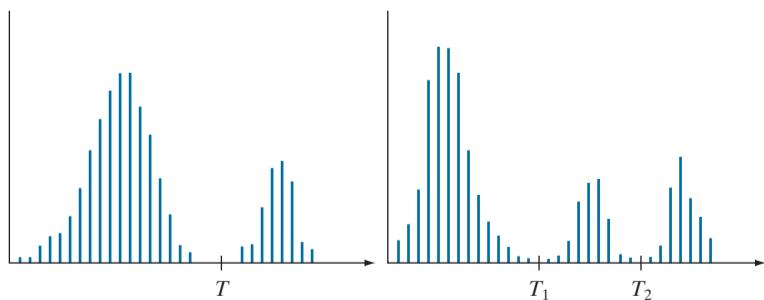
When T is a constant applicable over an entire image, the process is referred to as global thresholding.

Multiple thresholding

When the thresholding problem involving a histogram with three dominant modes corresponding, for example, to two types of light objects on a dark background, then thresholding classifies a point (x, y) as belonging to the background if:

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases}$$

where a , b , and c are any three distinct intensity values.



Intensity histograms that can be partitioned
(a) by a single threshold, and
(b) by dual thresholds.

For segmentation of intensity images correctly, appropriate thresholding is required.

This can depend upon a number of factors related directly to the width and depth of the valley(s) separating the histogram modes:

(1) the separation between peaks (the further apart the peaks are, the better the chances of separating the modes); (2) the noise content in the image (the modes broaden as noise increases); (3) the relative sizes of objects and background; (4) the uniformity of the illumination source; and (5) the uniformity of the reflectance properties of the image.

'Noise' in image thresholding

Noise within images would affect the intensity and hence the histogram. Hence, the thresholding will not work well on noisy images. It is recommended to filter the images using appropriate spatial filters and then apply thresholding. For complex images, we need to apply advanced algorithms for enabling the segmentation process and then apply thresholding.

Illumination and reflectance in image thresholding

Illumination and reflectance play a central role in the success of image segmentation using thresholding or other segmentation techniques. Therefore, controlling these factors when possible should be the first step considered in the solution of a segmentation problem.

Firstly, re-obtain the images itself with proper settings (depends on the imaging system).

If, cannot be done, using top-hat transformation can be helpful.

The third approach is to “work around” non-uniformities using variable thresholding.

The important point is to not loose information from the original images during any processing steps.

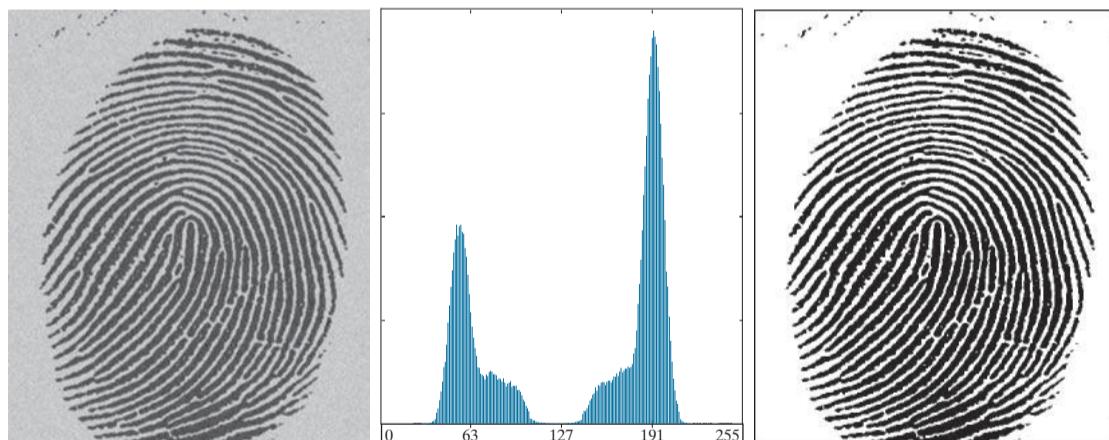
Optimising basic global thresholding

This methods implements global thresholding equation as we saw above. However, the method includes iterative steps to reach the optimum threshold value and can be implemented automatically.

When the intensity distributions of objects and background pixels are sufficiently distinct, it is possible to use a single (global) threshold applicable over the entire image. In most applications, there is usually enough variability between images that, even if global thresholding is a suitable approach, an algorithm capable of estimating the correct threshold value for each image is required. The following iterative algorithm can be used for this purpose:

1. Select an initial estimate for the global threshold, T .
2. Segment the image using T to produce two groups of pixels: G_1 , consisting of pixels with intensity values $> T$; and G_2 , consisting of pixels with values $\leq T$.
3. Compute the average (mean) intensity values m_1 and m_2 for the pixels in G_1 and G_2 , respectively.
4. Compute a new threshold value midway between m_1 and m_2 : $T=0.5(m_1 + m_2)$
5. Repeat Steps 2 through 4 until the difference between values of T in successive iterations is smaller than a predefined value, ΔT .

The algorithm works well where there is a reasonably clear valley between the modes of the histogram related to objects and background.



(a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (thin image border added for clarity).

Global thresholding using Otsu's method

Thresholding may be viewed as a statistical-decision theory problem whose objective is to minimize the average error incurred in assigning pixels to two or more groups (also called classes). This problem is known to have an elegant closed-form solution known as the Bayes decision function (i.e. probability that a pixel belongs to foreground or background). Hence the probability density function (PDF) of the

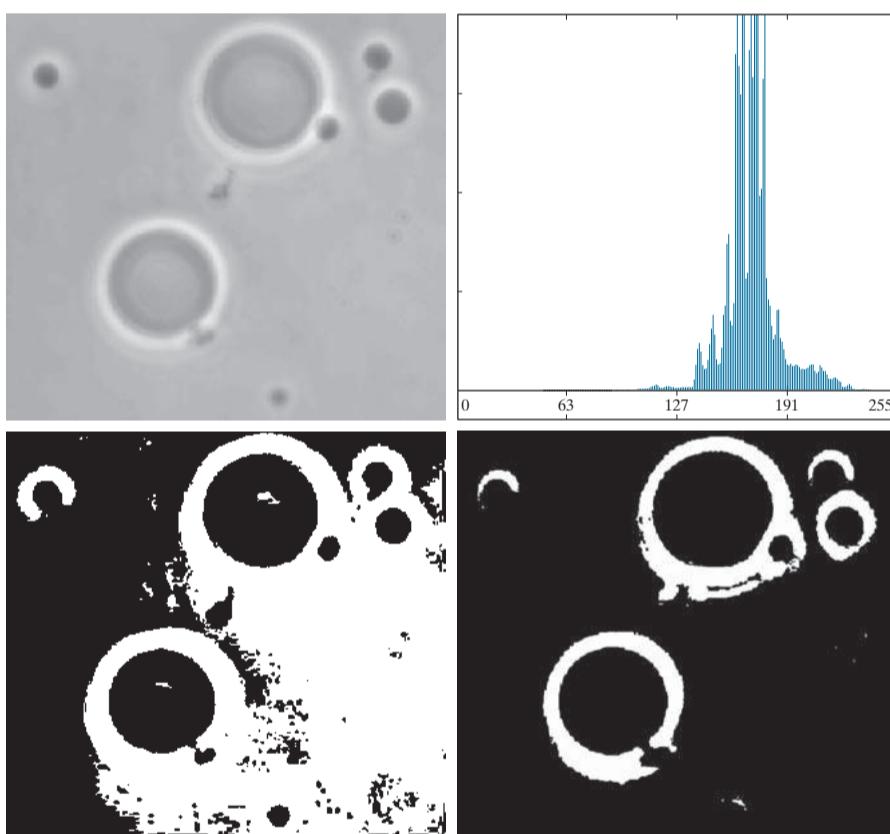
intensity levels of each class, and the probability that each class occurs are the parameters to solve such a problem.

However, estimation of PDFs could be tricky and not always well-suited for real-time applications.

Otsu devised an approach that maximizes the between-class variance, a well-known measure used in statistical discriminant analysis.

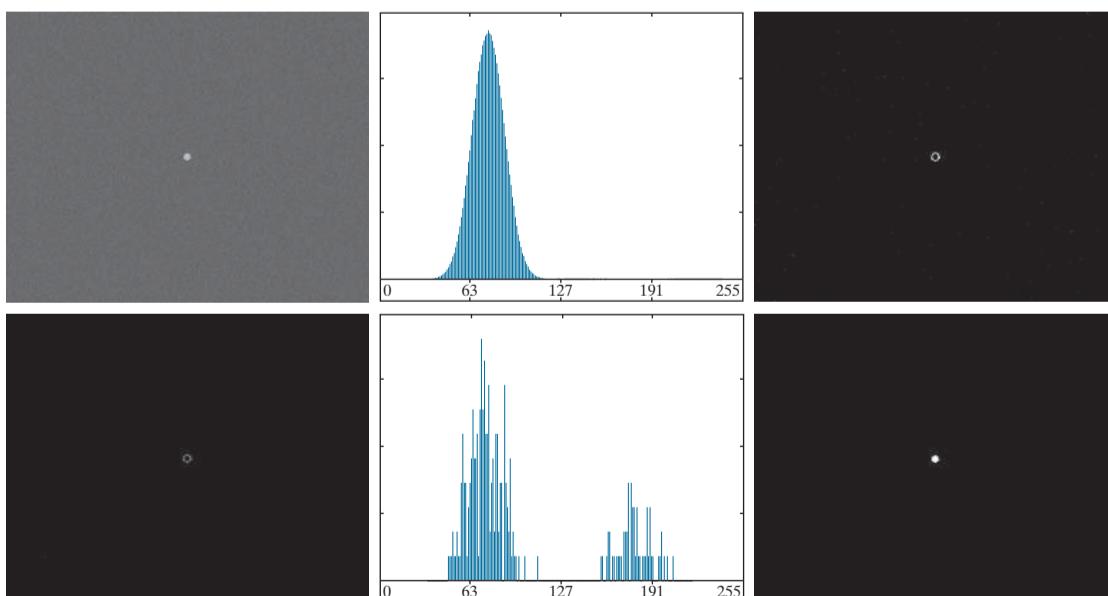
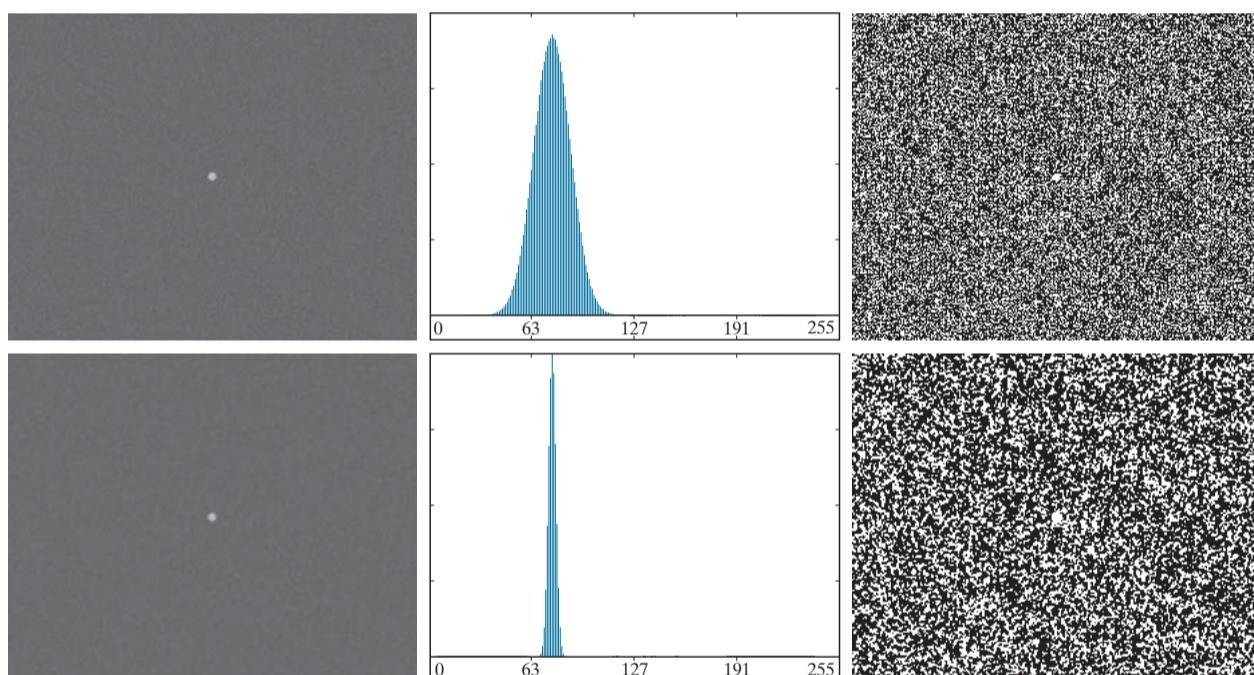
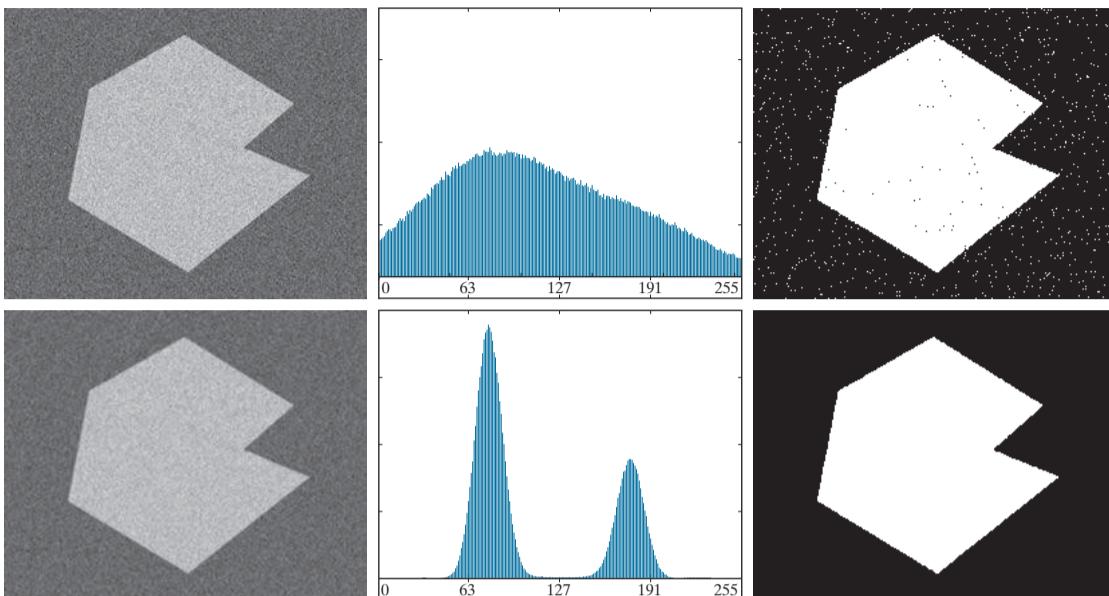
The analysis uses the histogram of an image for performing computations such as mean intensity, variance of each class.

For details on the algorithm, please read Bayes formula and revise histogram normalisation (from the chapters on mathematical tools) and refer to the research articles sent to you, the original Otsu's paper is the list.



(a) Original image. (b) Histogram (high peaks were clipped to highlight details in the lower values). (c) Segmentation result using the basic global algorithm. (d) Result using Otsu's method.

Image smoothing and edge detection help to improve global thresholding

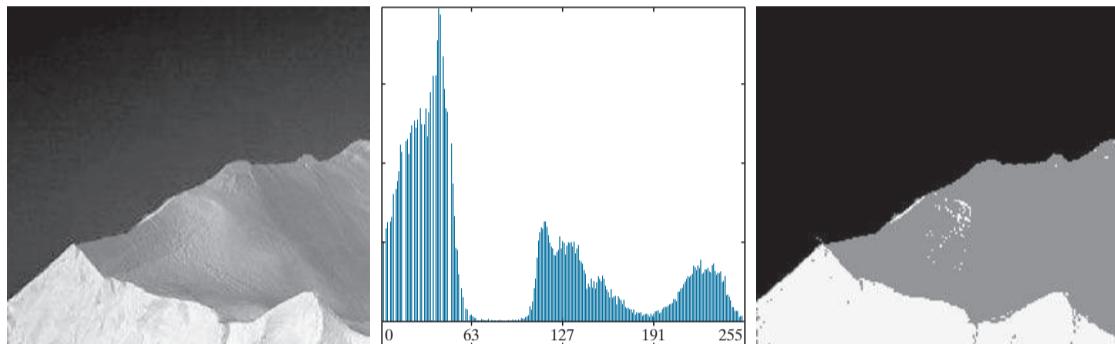


segmenting image (a) with the Otsu threshold based on the histogram.

(f) Result of

Multiple thresholds

Otsu's method can be extended to an arbitrary number of thresholds because the separability measure on which it is based also extends to an arbitrary number of classes. Hence the between-class variance can be generalised to any number of classes (pixels grouping). The requirement of multiple thresholding are solved using more than just intensity values. Instead, the approach is to use additional descriptors (e.g., color) and the application is cast as a pattern recognition problem.



(a) Image of an iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds.

Variable thresholding

The factors such as noise and nonuniform illumination play a major role in the performance of a thresholding algorithm. However, there are problems that cannot be solved by any of the thresholding methods discussed thus far.

When the value of T changes over an image, we use the term variable thresholding. The terms local or regional thresholding are used sometimes to denote variable thresholding in which the value of T at any point (x,y) in an image depends on properties of a neighborhood of (x,y) (for example, the average intensity of the pixels in the neighborhood). If T depends on the spatial coordinates (x, y) themselves, then variable thresholding is often referred to as dynamic or adaptive thresholding.

Variable thresholding based on local image properties (local thresholding)

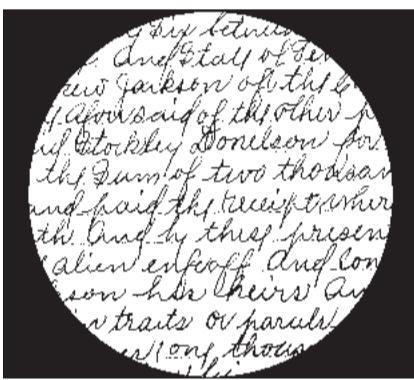
A basic approach to variable thresholding is to compute a threshold at every point, (x,y) , in the image based on one or more specified properties in a neighborhood of (x, y) . By using the local mean and local standard deviation of the pixel values in a neighbourhood of every point in an image, the local thresholds can be determined.

There are also other approaches for local thresholding (for extended reading see Bernsen local thresholding that uses the local contrast within a sub-image to determine the thresholds).

Variable thresholding based on moving averages

A special case of the variable thresholding method discussed in the previous section is based on computing a moving average along scan lines of an image. This implementation is useful in applications such as document processing, where speed is a fundamental requirement. The scanning typically is carried out line by line in a zigzag pattern to reduce illumination bias.

Ind ninety six between Stockley
of Kno. And State of Tennessee
Andrew Jackson off the County
day April said of the other part
and Stockley Donelson for A
of the sum of two thousand
and paid the receipt wherse
hath and by these presents
by alien enfeoff and confer
Jackson his heirs and a
certain traits or parcell of La
and a certain thousand p[ounds] and heis



Ind ninety six between Stockley
of Kno. And State of Tennessee
Andrew Jackson off the County
day April said of the other part
and Stockley Donelson for A
of the sum of two thousand
and paid the receipt wherse
hath and by these presents
by alien enfeoff and confer
Jackson his heirs and a
certain traits or parcell of La
and a certain thousand p[ounds] and heis

- (a) Text image corrupted by spot shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages.

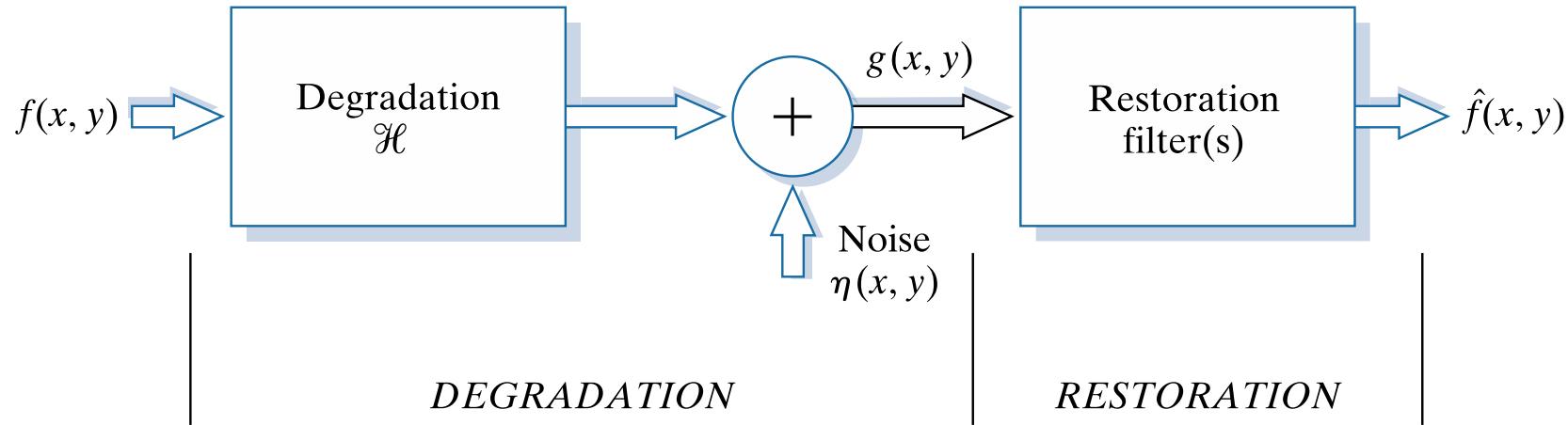
Noise models and image de-noising methods

DSE312-Lecture14

BHAVNA R

Different Types of Noise

consider linear, space invariant restoration models



$$g(x, y) = (h \star f)(x, y) + \eta(x, y)$$

degraded image

additive noise

Objective: estimate of $\bar{f}(x, y)$

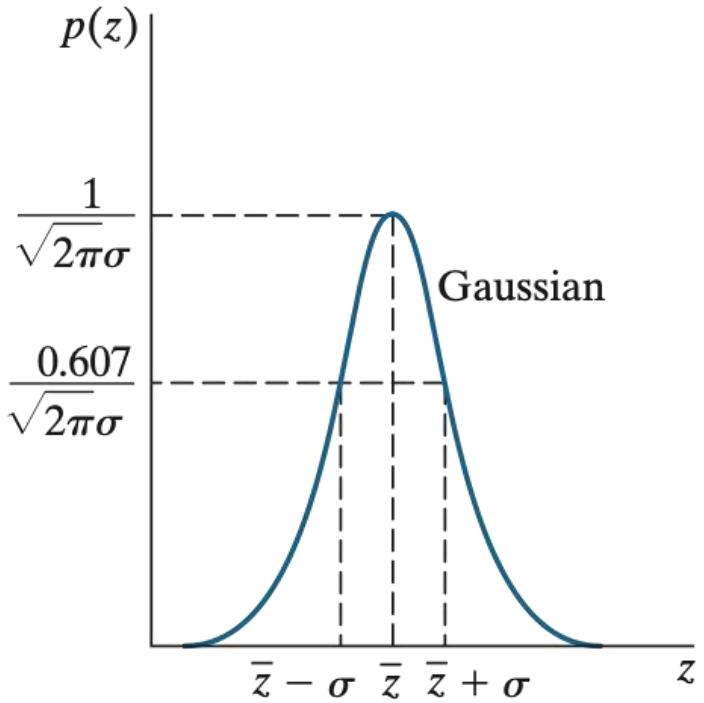
Spatial and frequency properties of noise

- Noise is always present in digital images during image acquisition, coding, transmission, and processing steps.
- Assumption: noise is independent of spatial coordinate
- Noise is uncorrelated with respect to the image itself (that is, there is no correlation between pixel values and the values of noise components). Eg: white noise.

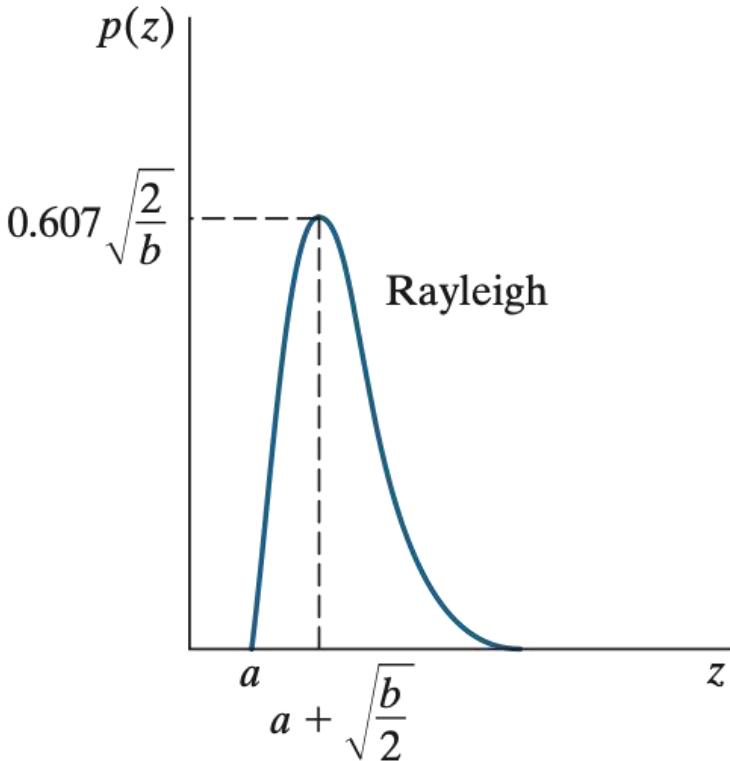
How do we model noise?

- We have an estimate of the ‘type’ of noise
- It’s a quantitative approach where noise is estimated and image is restored based on the noise model.
- model the degradation and apply the inverse process in order to recover the original image.
- characterise the statistical behaviour of the intensity values in the noise component ($\eta(x,y)$) of the model
- This component is considered as random variables, characterised by a probability density function (PDF)

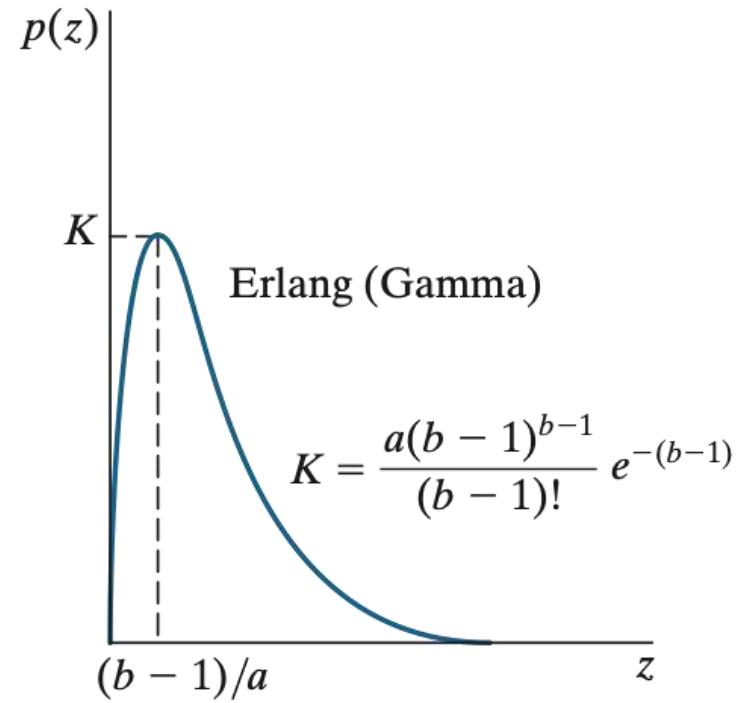
Types of PDFs



$$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\bar{z})^2}{2\sigma^2}} \quad -\infty < z < \infty$$



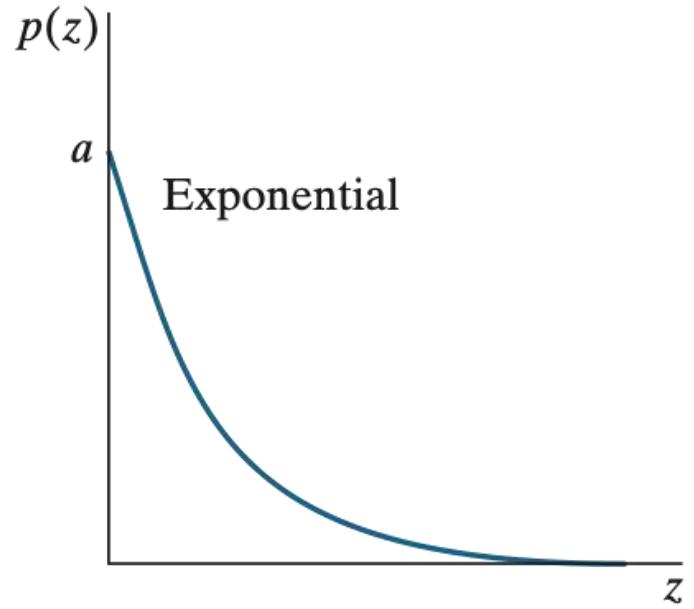
$$\bar{z} = a + \sqrt{\pi b/4}$$



$$K = \frac{a(b-1)^{b-1}}{(b-1)!} e^{-(b-1)}$$

$$\bar{z} = \frac{b}{a} \quad \sigma^2 = \frac{b}{a^2}$$

$$\sigma^2 = \frac{b(4 - \pi)}{4}$$

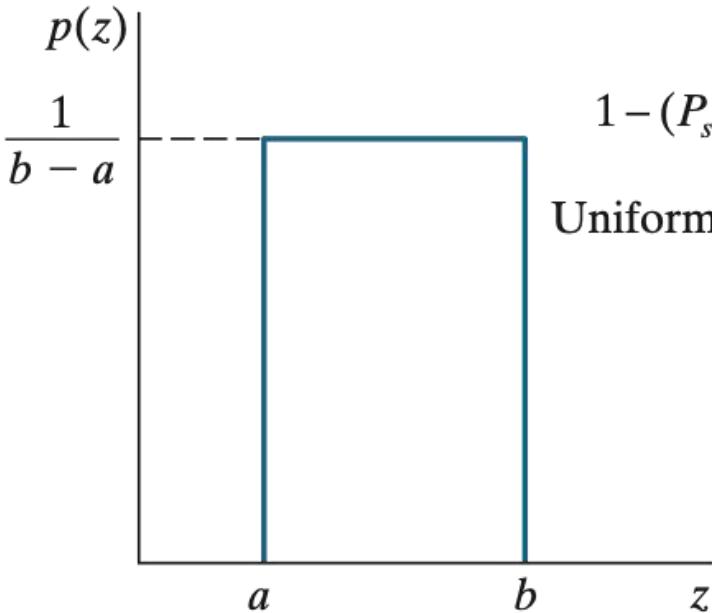


$$\begin{cases} ae^{-az} & z \geq 0 \\ 0 & z < 0 \end{cases}$$

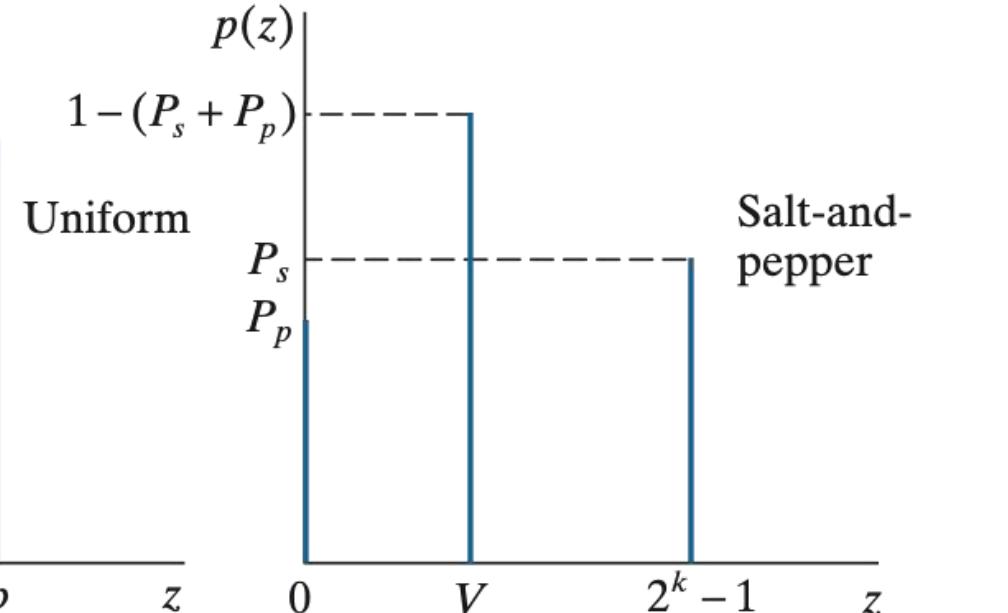
$$\bar{z} = \frac{1}{a} \quad \sigma^2 = \frac{1}{a^2}$$

$$\bar{z} = \frac{a + b}{2}$$

$$\sigma^2 = \frac{(b - a)^2}{12}$$

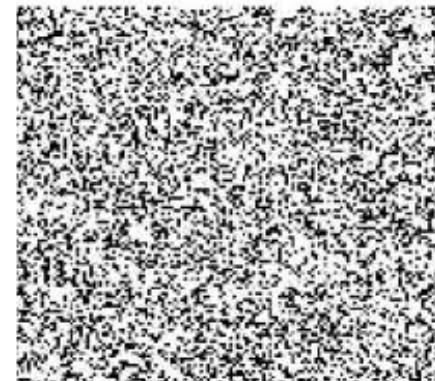
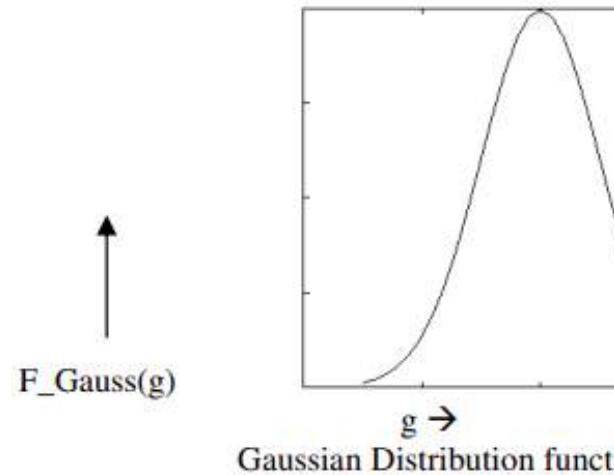


$$\begin{cases} \frac{1}{b - a} & a \leq z \leq b \\ 0 & \text{otherwise} \end{cases}$$

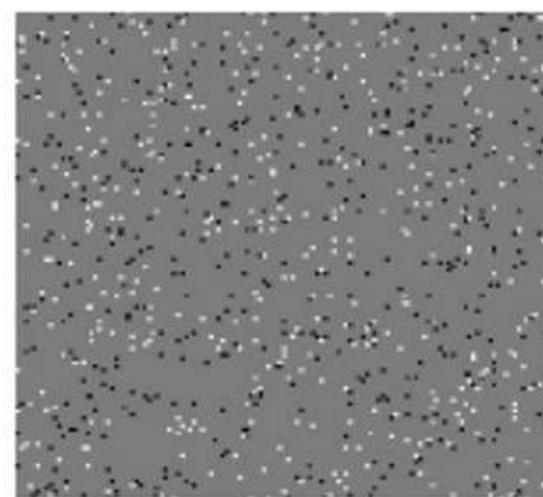
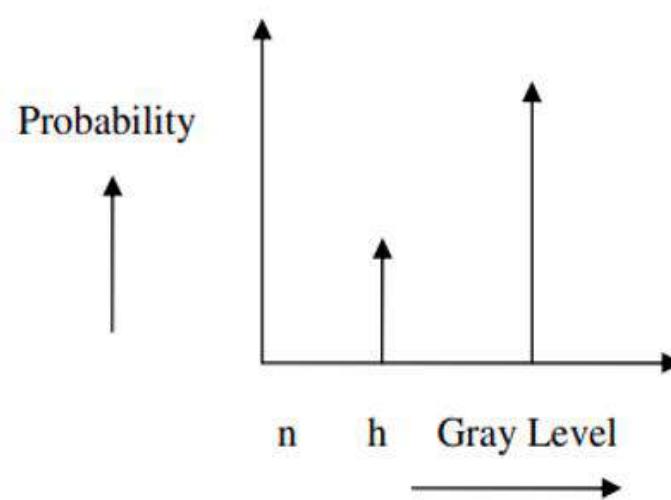


$$\begin{cases} P_s & \text{for } z = 0 \\ P_p & \text{for } z = V \\ 1 - (P_s + P_p) & \text{for other } z \end{cases}$$

Salt-and-pepper



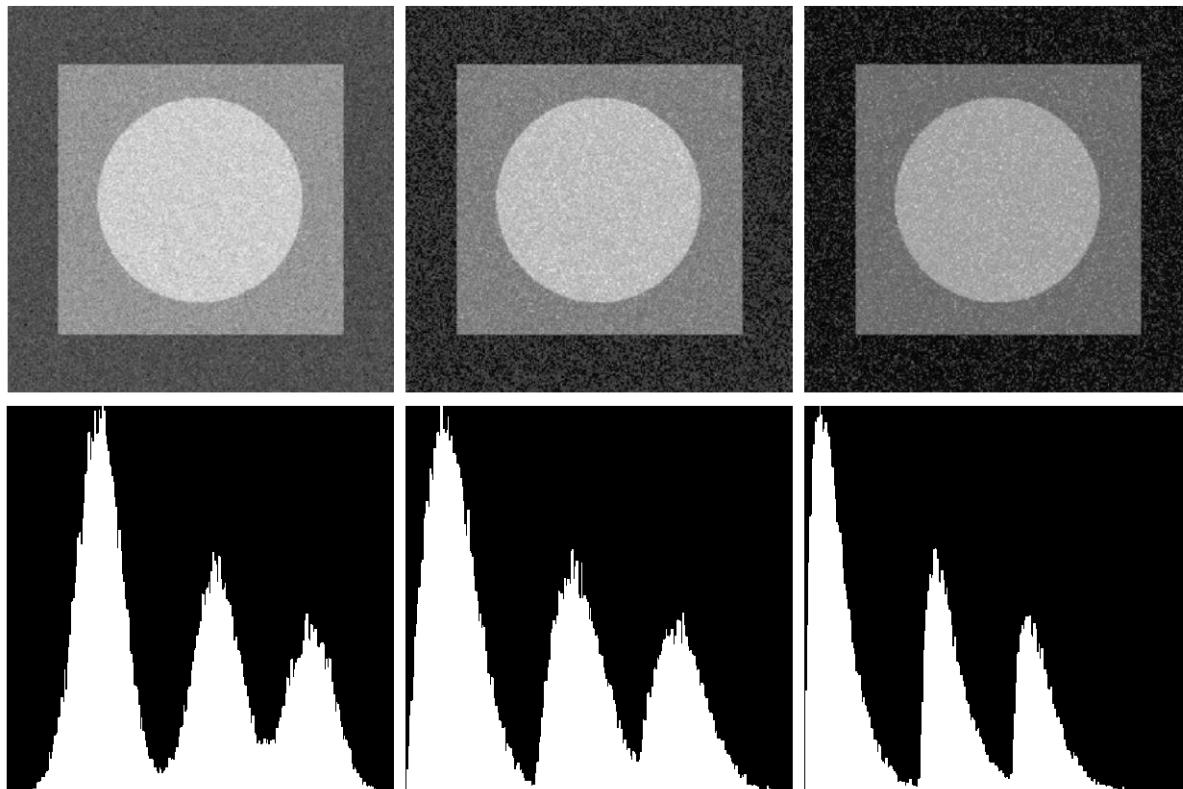
Gaussian noise



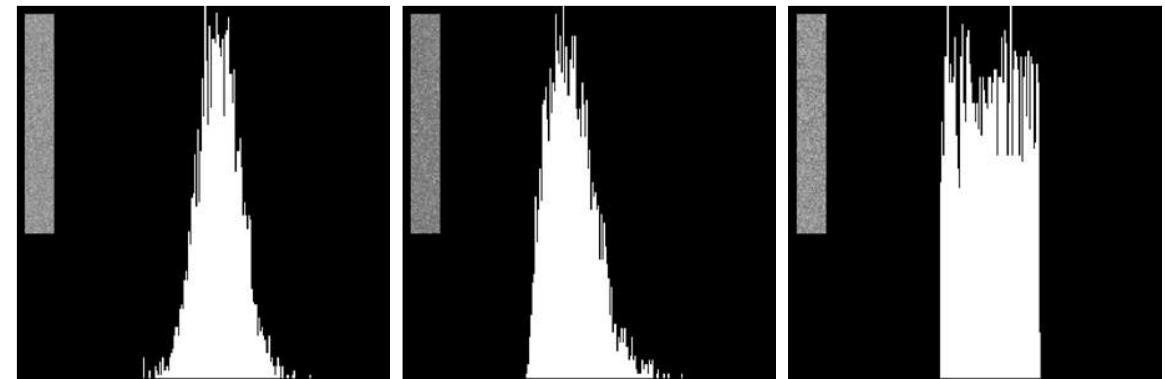
Salt & pepper noise

Estimating the noise parameters from images

For Gaussian, Rayleigh, Gamma, exponential noise



images and histograms resulting from adding Gaussian, Rayleigh, and Erlangen noise to a test pattern



shape of the histograms computed using small strips (inserts) from the Gaussian, Rayleigh, uniform noisy images.

- We need the shape of the distribution to estimate mean and variance parameters.
- Select a region within the image, (large enough to produce the characteristic histogram).
- Compute mean & variance of the pixels
- Derive any secondary parameters (depending upon the noise)

denoising image: additive random noise

Mean filters

- ✓ Box kernel filter
- ✓ Gaussian filter

Geometric Mean filters

$$\hat{f}(x,y) = \left[\prod_{(r,c) \in S_{xy}} g(r,c) \right]^{\frac{1}{mn}}$$

where Π indicates multiplication

- each restored pixel is given by the product of all the pixels in the subimage area, raised to the power $1/mn$.
- filter achieves smoothing while tends to lose less image detail in the process, so better than box filter.

Harmonic Mean filters

$$\hat{f}(x,y) = \frac{mn}{\sum_{(r,c) \in S_{xy}} \frac{1}{g(r,c)}}$$

works well for Gaussian noise & salt noise, but fails for pepper noise.

Harmonic Mean filters

$$\hat{f}(x,y) = \frac{\sum_{(r,c) \in S_{xy}} g(r,c)^{Q+1}}{\sum_{(r,c) \in S_{xy}} g(r,c)^Q}$$

- where Q is the order of the filter
- suited for eliminating the effects of salt-and-pepper noise, but not simultaneously.
- $+Q$ values=> eliminates pepper noise
- $-Q$ values=> eliminates salt noise
- reduces to arithmetic mean filter if $Q = 0$, and to the harmonic mean filter if $Q = -1$.

Order-statistics filters

- ✓ Median filter: provides less blurring than linear smoothing filters, useful for filtering both bipolar (salt & pepper) and unipolar impulse noise. (50th percentile of a ranked set of numbers.)

Max and min filters

$$\hat{f}(x,y) = \max_{(r,c) \in S_{xy}} \{g(r,c)\} \text{ reduces pepper noise (100th percentile)}$$

$$\hat{f}(x,y) = \min_{(r,c) \in S_{xy}} \{g(r,c)\} \text{ reduces salt noise (0th percentile)}$$

Wiener noise

- considers both image & noise as random variables.
- noise and image are assumed uncorrelated, with either one having a zero mean, and that the intensity levels in the estimate are a linear function of the levels in the degraded image.
- find a restored image with minimum mean square error with the uncorrupted image:

$$e^2 = E\{(f - \hat{f})^2\} \quad \text{with } E\{\cdot\} \text{ as the expected value of the argument.}$$

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} \frac{|H(u, v)|^2}{|H(u, v)|^2 + S_n(u, v)/S_f(u, v)} \right] G(u, v)$$

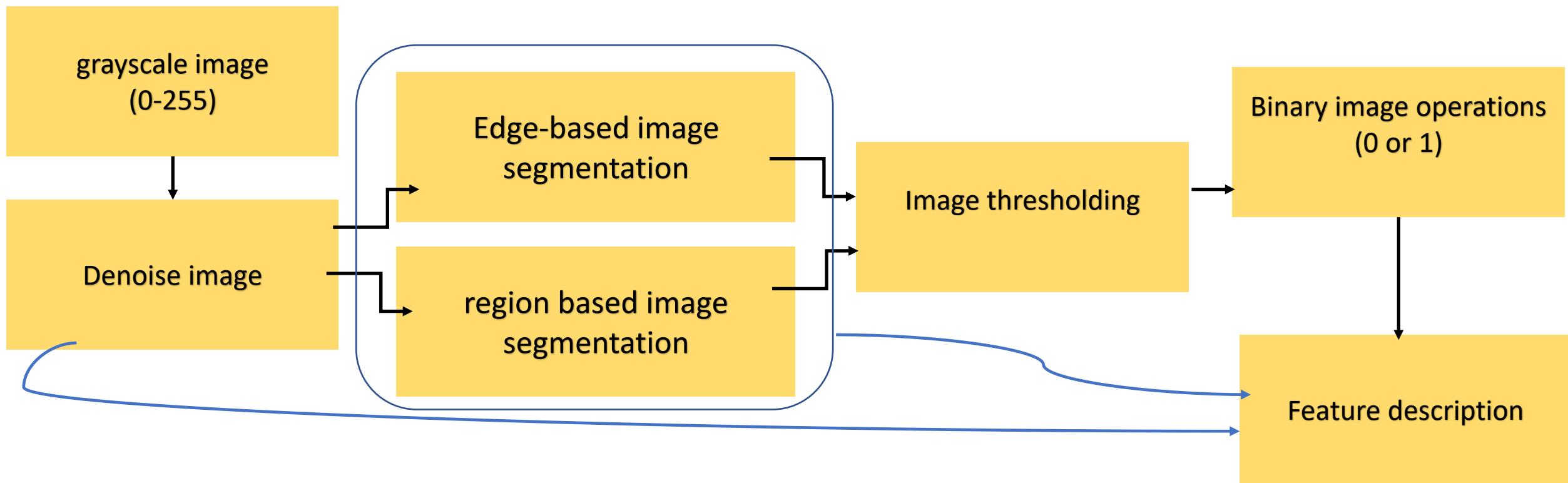
$H^*(u, v)$:	complex conjugate of $H(u, v)$
$ H(u, v) ^2$	=	$H^*(u, v)H(u, v)$
$S_n(u, v)$	=	$ N(u, v) ^2$ power spectrum of the noise
$S_f(u, v)$	=	$ F(u, v) ^2$ power spectrum of the undegraded image

We don't need to derive the expression

$\cdot S_n(u, v)/S_f(u, v)$. Assume the ratio as constant
i.e. white noise with constant spectrum

So far,

- We have addressed the noise issue within the images
- We learnt edge based techniques in the last lectures
- we have also learnt thresholding techniques



A short note on binary images

- Values are 0 or 1
- Images easy to store, process & analyze
- Apart from that, many other analysis can be performed efficiently on binary images since faster to compute
- Thresholding a grayscale image gives a binary image
- Reasonable pre-processing steps will lead to obtain the right image for thresholding to get a binary image
- Binary images used in multiple disciplines
 - Morphological operations to improve specific recognition tasks
 - geometric properties of binary images
 - Labeling objects within binary images (to do segmentation)
 - Extract binary features of images

Homework: Please try the following operators by yourself

1. Take a grayscale image
 - Perform denoising; use different methods, see the resulting images and self-judge the method and determine correct parameters (typically there is no right answer to parameters, within a close range of parameters, the do-noising would work!)
 - try the various edge detection techniques on the denoised image, use parameters, if any
 - Use Otsu's thresholding on the edge detected images
2. Perform Hough transform on a denoised image
3. Estimate the noise model in an image

Eg: use a grayscale noisy image, consider a region (without any edges or feature information) and draw the pixel histogram, then determine the the noise model from the distribution, make sure to have sufficiently large region to obtain mean and standard deviation of the pixels or any secondary parameters but the region shouldn't have any features

Next Lectures:

- Region-based image segmentation (clustering techniques)
- Processing Binary images
- Morphological operations on binary images

COURSE
DIGITAL IMAGE PROCESSING AND APPLICATIONS IN BIOIMAGE ANALYSIS

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2021
DSE-409/609

Chapter: Morphological Image Processing

WEEK-10

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Chapter : Morphological operations-1

Basic background

mathematical morphology as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull.

morphological techniques for pre- or post-processing images, such as morphological filtering, thinning, and pruning.

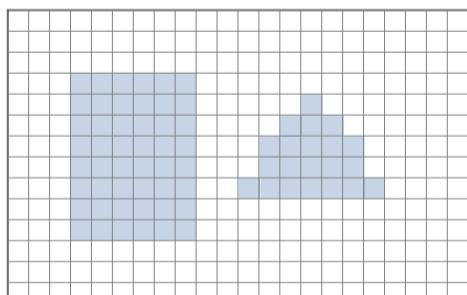
Morphological operations are defined in terms of sets. So the image objects are represented as sets and the morphological operations carried out can be expressed using concepts from set theory.

Let us understand how sets are defined in terms of mathematical morphology.

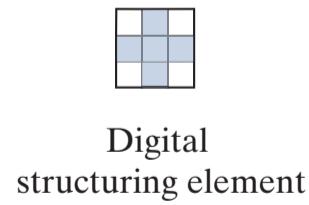
- In binary images, the sets are members of the 2-D integer space Z^2 (bi-dimensional integers that represent foreground pixels), and each element coordinates are the coordinates belong to an object pixel in the image.
- Grayscale digital images can be represented as sets whose components are in Z^3 . The two components of each element refer to the coordinates of a pixel, and the third corresponds to its discrete intensity value.
- Sets in higher dimensional spaces can contain other image attributes, such as color and time-varying components.
- Given 2 sets $A, B \in Z^2$, we can define union, intersection and complement and differences between the 2 sets.
- Complement: $A^c = \{x | x \notin A\}$
- Difference $A-B = \{x | x \in A \vee x \notin B\} = A \cap B^c$
- Translation $A_z = \{x | x=a+z, a \in A\}$
- Reflection $\hat{B} = \{x | x=-b, b \in B\}$

There are two types of sets of pixels: objects and structuring elements (SE's).

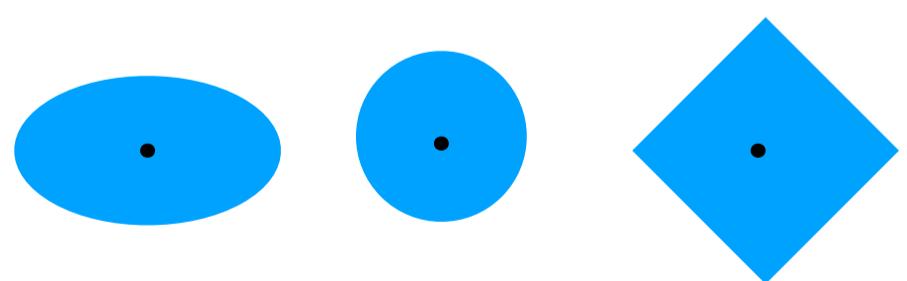
- These operations are carried out on binary images typically (represented as 0's or 1's) or grayscale images
- They can also be operated on objects defined as sets of foreground pixels within the background pixels of the images.
- foreground elements can be a single object, or disjoint subsets of foreground elements
- SE's have both foreground and background pixels
- SE's may contain "don't care" elements, denoted by x, signifying that the value of that particular element in the SE does not matter.
- SE's are used in a form similar to spatial convolution kernels and the image border just described is similar to the padding.
- SE's can have any shape and affect the operations performed on the images
- The operations are different in morphology, but the padding and sliding operations are the same as in convolution.
- SE's can have specific shapes and they affect the operations performed on the images.
- When specific operations are performed on an image using SE's, the resulting image pixels are altered



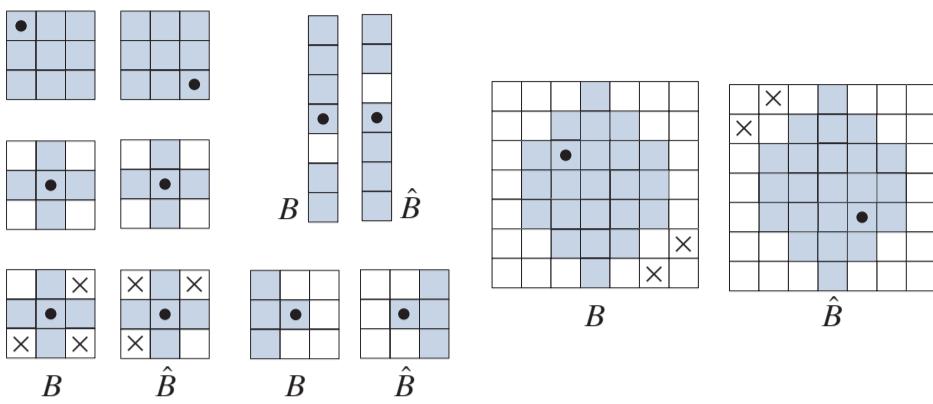
Digital image



Some shapes of SE's



Reflection and translations are used process images using SE's



SE's and their reflections about the origin (the X are don't care elements, dots denote the origin). Reflection is rotation by 180° of an SE about its origin.

Translation $A_z = \{x | x = a + z, a \in A\}$

Reflection $\hat{B} = \{x | x = -b, b \in B\}$

Here, all the morphological expressions are written in terms of SEs and a set, A of foreground pixels.

Erosion and dilation

Erosion:

With A and B as sets in Z^2 , the erosion of A by B, denoted $A \ominus B$, is defined as,

$$A \ominus B = \left\{ z \mid (B)_z \subseteq A \right\}$$

where A is a set of foreground pixels, B is a SE, and the z's are foreground values (1's).

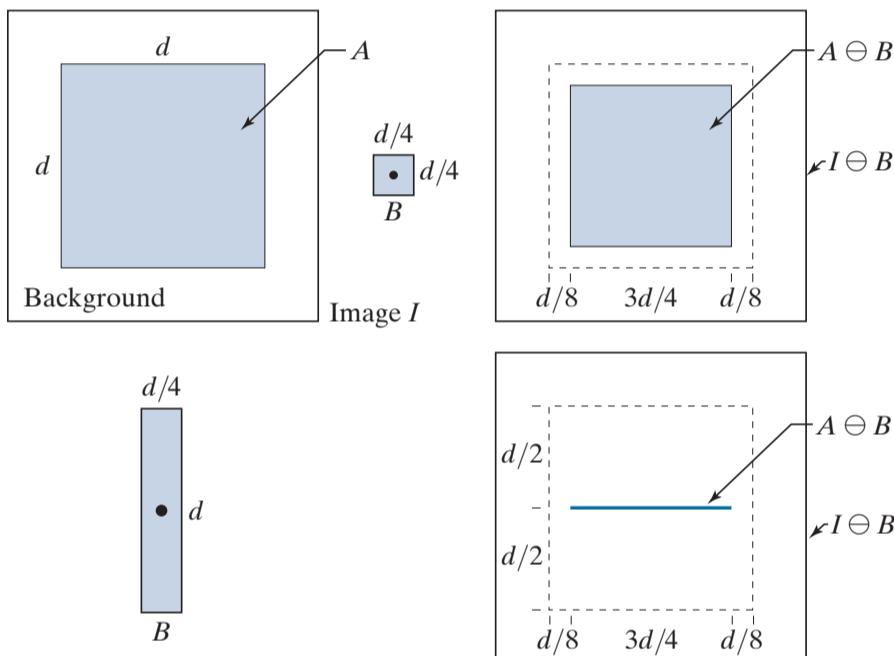
However, we are not dealing with sets here, instead images. So the sets of foreground pixels are embedded in a set of background pixels to form a complete image, I. Thus, inputs and outputs of our morphological procedures are images, not individual sets. Hence, we can rewrite the above equations as follows,

$$I \ominus B = \left\{ z \mid (B)_z \subseteq A \text{ and } A \subseteq I \right\} \cup \left\{ A^c \mid A^c \subseteq I \right\}$$

where I is a rectangular array of foreground and background pixels and A is a subset (i.e., is contained in) I . The erosion of I by B is the set of all points, z , such that B , translated by z , is contained in A . A^c denotes the set of background pixels in the image that are also considered during the operation. The result after erosion is an image of same size as I .

Because B has to be contained in A , is equivalent to B not sharing any common elements with the background (i.e., A^c), we can express erosion equivalently as

$$A \ominus B = \{z | (B)_z \cap A^c = \emptyset\}$$



(a) Image I , consisting of a set (object) A , and background. (b) Square SE, B (the dot is the origin). (c) Erosion of A by B (shown shaded in the resulting image).

(d) Elongated SE. (e) Erosion of A by B . (The erosion is a line.) The dotted border in (c) and (e) is the boundary of A (the foreground set of pixels in the image I), shown for reference.

Dilation:

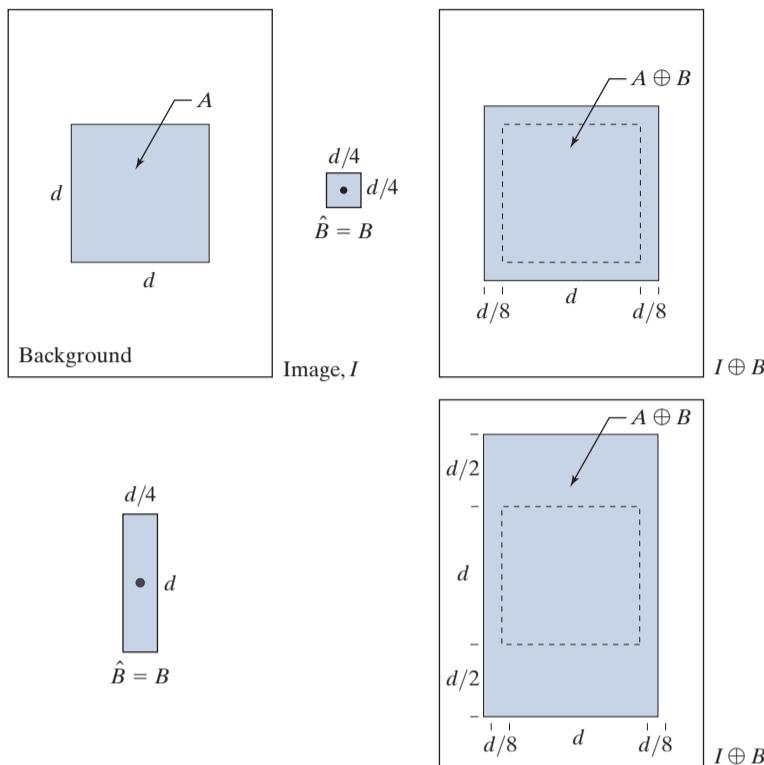
With A and B as sets in \mathbb{Z}^2 , the dilation of A by B , denoted as $A \oplus B$, is defined as

$$A \oplus B = \{z | (\hat{B})_z \cap A \neq \emptyset\}$$

This equation is based on reflecting B about its origin and translating the reflection by z , as in erosion. The dilation of A by B then is the set of all displacements, z , such that the foreground elements of \hat{B} overlap at least one element of A . (z is the displacement of the origin of \hat{B} (like growing foreground pixels)).

$$A \oplus B = \{z | [(\hat{B})_z \cap A] \subseteq A\}$$

Dilation is based on set operations and therefore is a nonlinear operation



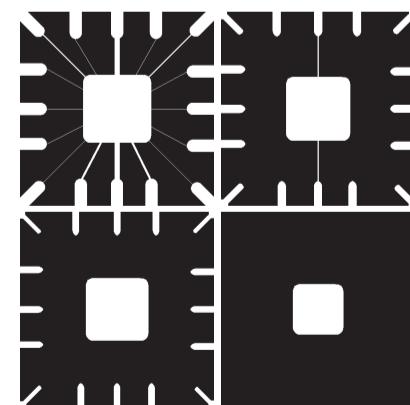
(a) Image I , composed of set (object) A and background. (b) Square SE (the dot is the origin). (c) Dilation of A by B (shown shaded). (d) Elongated SE. (e) Dilation of A by this element. The dotted line in (c) and (e) is the boundary of A , shown for reference.

Dilation	Erosion
grows/expands the foreground pixels within an image	shrinks/reduces the foreground pixels within an image
Growing features, filling holes within objects, filling gaps	Useful for removing bridges, branches, protrusions

The manner and extent of shrinking /thinning (erosion) or growing/ thickening (dilation) is controlled by the shape and size of the SE used.

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.

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.



Duality: Erosion and dilation are duals of each other with respect to set complementation and reflection.

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

$$(A \oplus B)^c = A^c \ominus \hat{B}$$

The erosion of A by B is the complement of the dilation of A^c by \hat{B} . The duality property is useful when the SEs are symmetric with respect to its origin (as often is the case), so that $B = \hat{B}$. Then, we obtain the erosion of A by dilating its background (i.e., dilating A^c) with the same SE and complementing the result.

Opening and closing

There are two other important morphological operations based on the order of erosion/dilation operations:

Opening Erosion of image I by SE, followed by a dilation

Closing Dilation of image I by SE, followed by a erosion

The basic idea of combining the erosion/dilation operations is to preserve the morphological shape of objects.

The opening of set A by SE B is given by,

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

Similarly, the closing of set A by SE B is,

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

Geometrical interpretation:

The opening of A by B is the union of all the translations of B so that B fits entirely in A, given as,

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\}$$

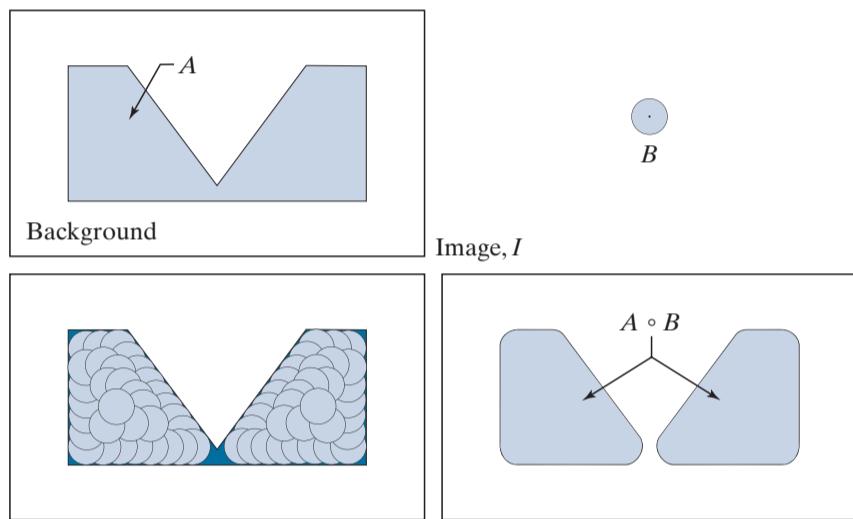
Similarly, closing is the complement of the union of all translations of B that do not overlap A.

$$A \bullet B = \left[\bigcup \{(B)_z \mid (B)_z \cap A = \emptyset\} \right]^c$$

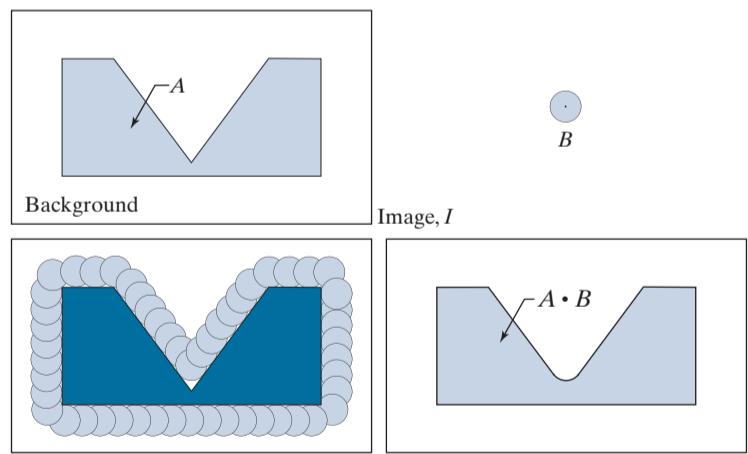
The ‘morphological opening’ operation tends to perform ‘erosion’, but as it is followed by ‘dilation’, the operation is less destructive than simple erosion.

Opening tends to remove some of the foreground (bright) pixels from the edges of regions of foreground pixels in the original image. But less number of pixels are removed than pure erosion operation.

Similarly, the ‘morphological closing’ tends to perform ‘dilation’ by increasing the number of foreground pixels within the objects, but again since it is followed by ‘erosion’, it better preserves the shape of the original objects.



(a) Image I, composed of set (object) A and background. (b) SE, B. (c) Translations of B while being contained in A. (A is shown dark for clarity.) (d) Opening of A by B.



(a) Image I, composed of set (object) A, and background. (b) SE B. (c) Translations of B such that B does not overlap any part of A. (A is shown dark for clarity.) (d) Closing of A by B.

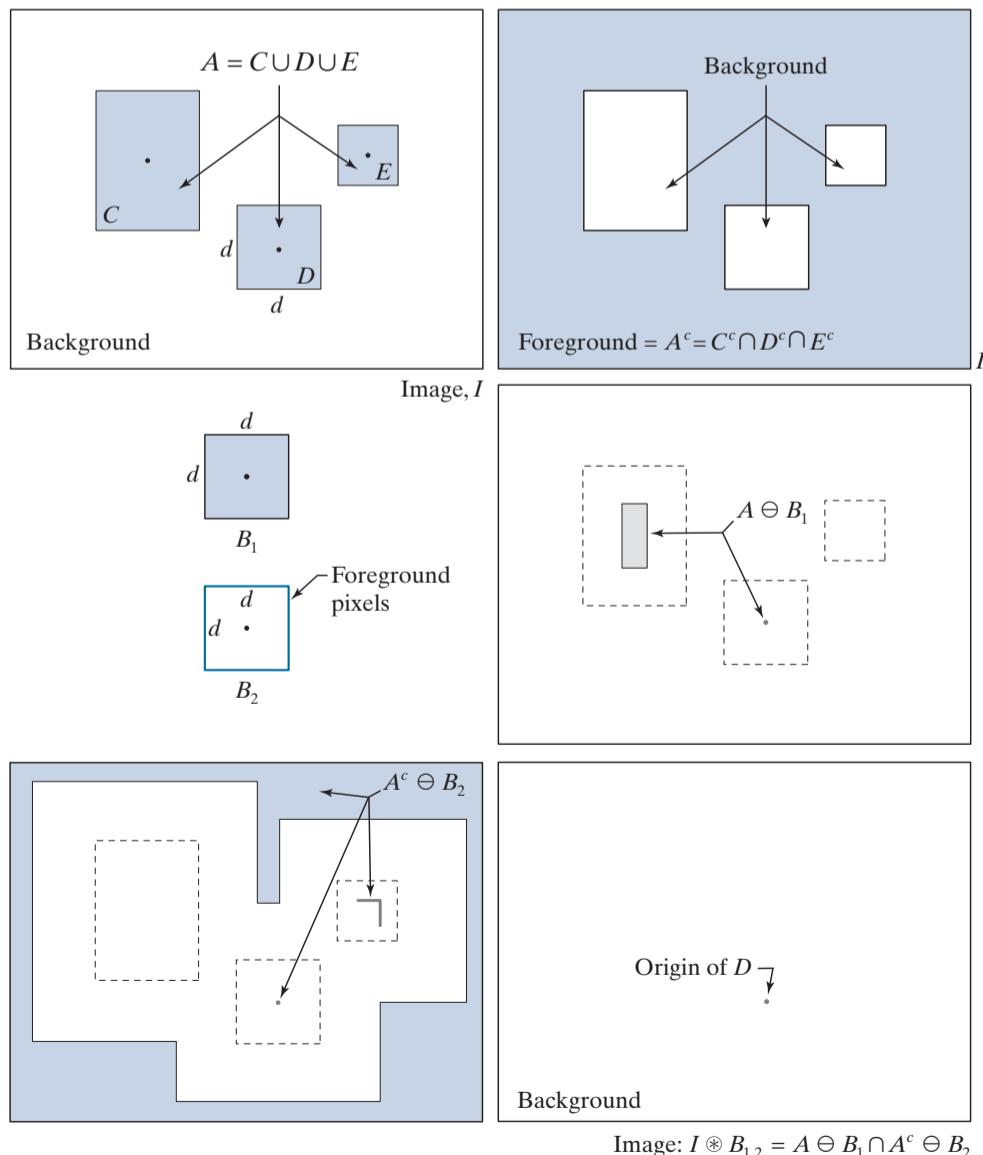
Hit-or-miss transform

The morphological hit-or-miss transform (HMT) is a basic tool for shape detection. Let I be a binary image composed of foreground (A) and background pixels, respectively. HMT utilises two SEs: B_1 , for detecting shapes in the foreground, and B_2 , for detecting shapes in the background. The HMT of image I is defined as

$$\begin{aligned} I \circledast B_{1,2} &= \left\{ z \mid (B_1)_z \subseteq A \text{ and } (B_2)_z \subseteq A^c \right\} \\ &= (A \ominus B_1) \cap (A^c \ominus B_2) \end{aligned}$$

where the second line follows from the definition of erosion.

Morphological HMT is the set of translations, z , of SEs B_1 and B_2 such that, simultaneously, B_1 found a match in the foreground (i.e., B_1 is contained in A) and B_2 found a match in the background (i.e., B_2 is contained in A^c). ‘ z ’ is the same translation of both SEs. The word “miss” arises from the fact that B_2 finding a match in A^c is the same as B_2 not finding (missing) a match in A .



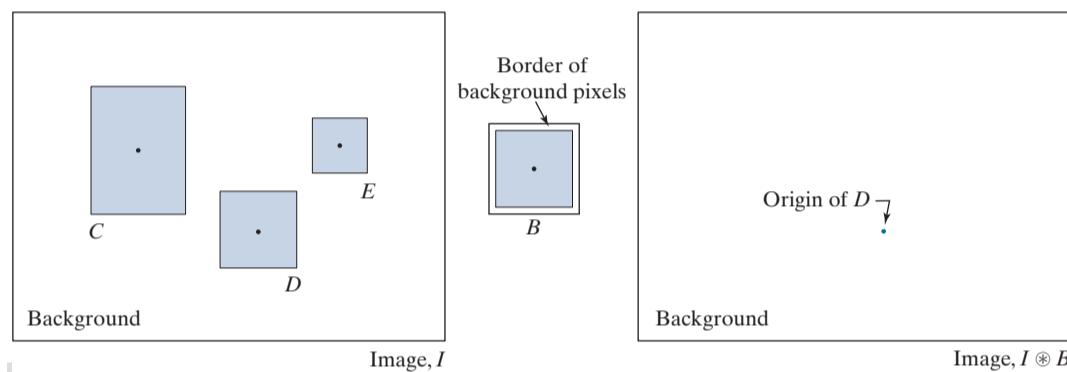
- (a) Image consisting of a foreground (1's) equal to the union, A , of set of objects, and a background of 0's.
- (b) Image with its foreground defined as A^c .
- (c) SEs designed to detect object D .
- (d) Erosion of A by B_1 .
- (e) Erosion of A^c by B_2 .
- (f) Intersection of (d) and (e), showing the location of the origin of D , as desired. The dots indicate the origin of their respective components. Each dot is a single pixel.

$$\text{Image: } I \circledast B_{1,2} = A \ominus B_1 \cap A^c \ominus B_2$$

It is also possible to perform HMT operation using a single SE. However, the erosion operation is implemented in a non-traditional manner. In the above example, we can detect D directly in image I, by processing foreground and background pixels simultaneously, rather than processing just foreground pixels, as required by the definition of erosion.

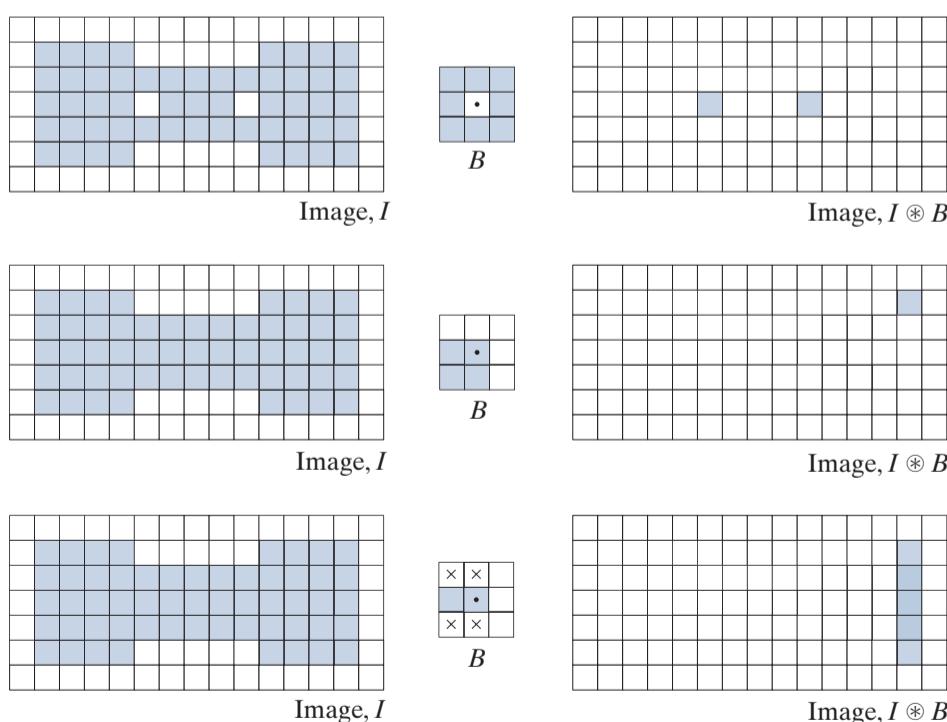
We define SE such that

$$I \circledast B = \{z \mid (B)_z \subseteq I\}$$



This SE consists of both foreground and background elements. We define a SE, B, identical to D, but having in addition a border of background elements with a width of one pixel ;

Here is another example using a single SE



Three examples of using a single SE to detect specific features. First row: detection of single-pixel holes. Second row: detection of an upper-right corner. Third row: detection of multiple features.

Using don't care elements increases the flexibility of SEs to perform multiple roles.

Basic morphological algorithms

For binary images (shown image examples here with foreground (1's) shaded and background (0's) in white), we now consider applications of morphology in extracting image components that are useful in the representation and description of shape such as morphological algorithms for extracting

- boundaries,
- connected components,
- convex hull,
- skeleton of a region.

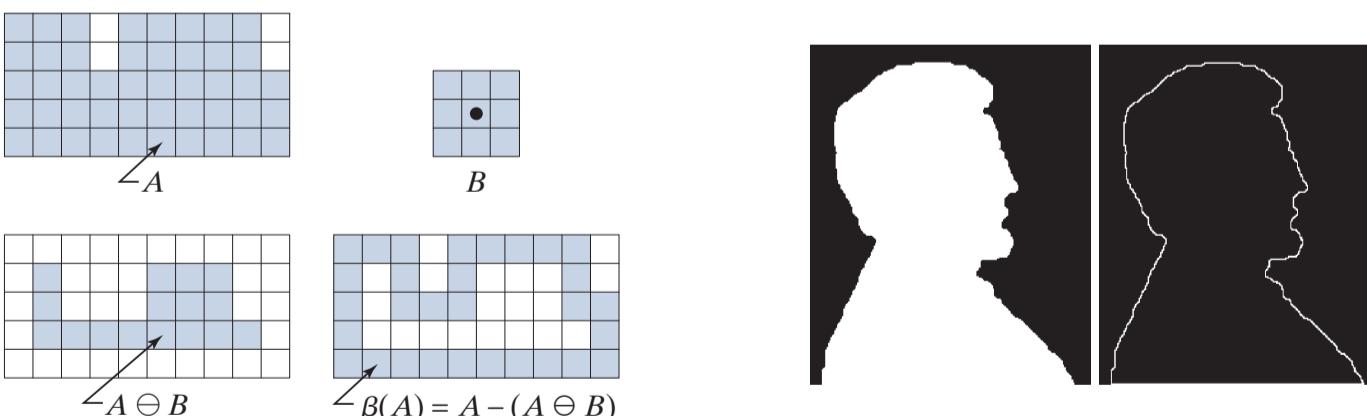
We also develop methods for:

- region filling,
- thinning,
- thickening,
- pruning

Boundary extraction

The boundary of a set A of foreground pixels, denoted by $\beta(A)$, can be obtained by first eroding A by a suitable SE, B, and then performing the set difference between A and its erosion. That is,

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



(a) Set, A, of foreground pixels.(b) 5X5 SE of 1's.

(c) A eroded by B. (d) Boundary of A.

Hole filling

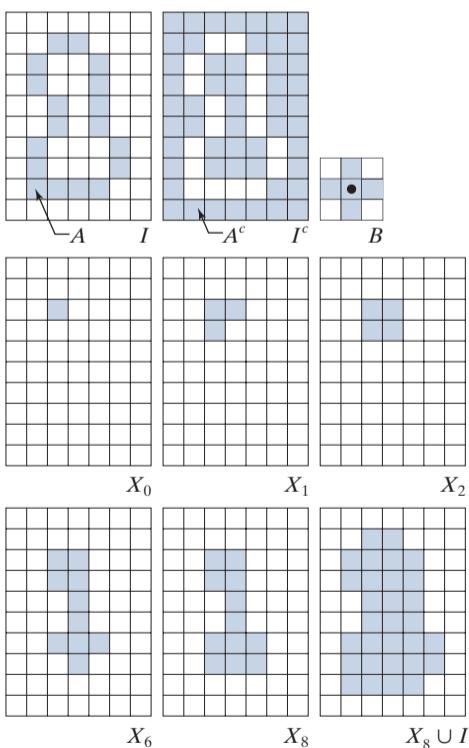
A hole is a background region surrounded by a connected border of foreground pixels. Algorithms based on set dilation, complementation, and intersection can be implemented for filling holes in an image.

Let A denote a set whose elements are 8-connected boundaries, with each boundary enclosing a background region (i.e., a hole). Given a point in each hole, the objective is to fill all the holes with foreground elements (1's). We begin by forming an array, X_0 of 0's (the same size as I , the image containing A), except at locations in X_0 that correspond to pixels that are known to be holes, which we set to 1. Then, the following procedure fills all the holes with 1's:

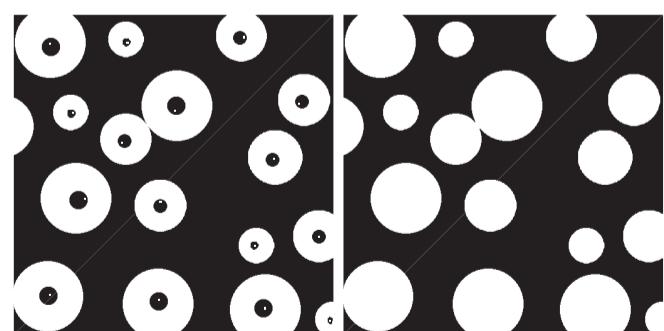
$$X_k = (X_{k-1} \oplus B) \cap I^c \quad k = 1, 2, 3, \dots$$

where B is the symmetric SE.

- The algorithm terminates at iteration step k if $X_k = X_{k-1}$. Then, X_k contains all the filled holes. The set union of X_k and I contains all the filled holes and their boundaries.
- The dilation fills the entire area if left unchecked, but the intersection at each step with I^c limits the result to inside the region of interest, called as conditional dilation.



Hole filling.(a) Set A (shown shaded) contained in image I .
(b) I^c (c) SE: B . Only the foreground elements are used in computations (d) Initial point inside hole, set to 1. Final result [union of (a) and (h)].



- (a) Binary image. White dots inside the regions (shown enlarged for clarity) are the starting points for the hole-filling algorithm.
(b) Result of filling all holes.

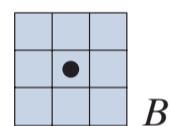
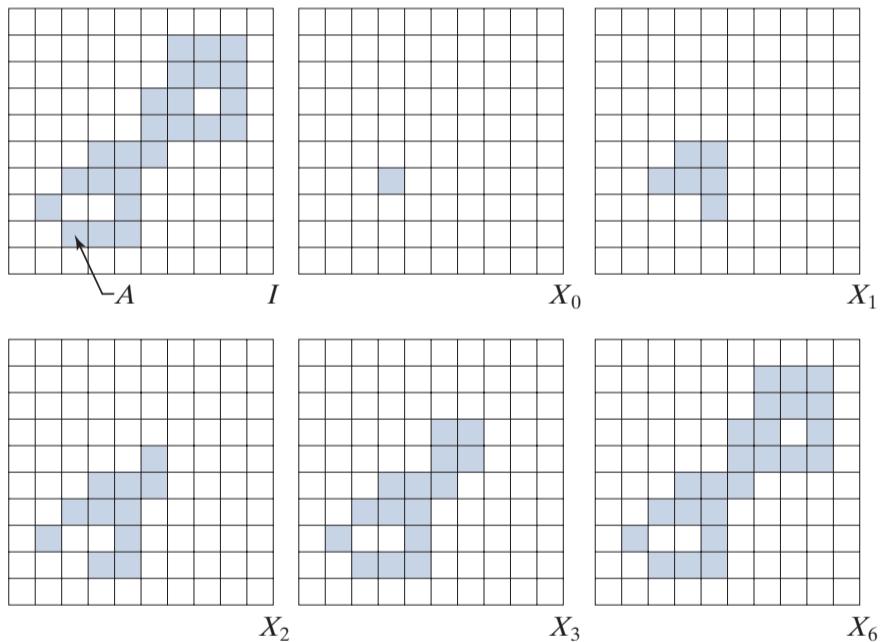
Extraction of connected components

Being able to extract connected components from a binary image is central to many automated image analysis applications. Let A be a set of foreground pixels consisting of one or more connected components, and form an image X_0 (of the same size as I , the image containing A) whose elements are 0's (background values), except at each location known to correspond to a point in each connected component in A , which we set to 1 (foreground value). The objective is to start with X_0 and find all the connected components in I . The following iterative procedure accomplishes this:

$$X_k = (X_{k-1} \oplus B) \cap I \quad k = 1, 2, 3, \dots$$

where B is the SE.

The procedure terminates when $X_k = X_{k-1}$, with X_k containing all the connected components of foreground pixels in the image (again using conditional dilation to limit the growth of set dilation on image I).



(a) SE (B) used based on 8-connectivity between pixels. (b) Image containing a set with one connected component. (c) Initial array containing a 1 in the region of the connected component. (d)-(g) Various steps in the iterations ($k=6$)

Convex hull

A set, S , of points in the Euclidean plane is said to be convex if and only if a straight line segment joining any two points in S lies entirely within S .

The convex hull, H , of S is the smallest convex set containing S .

The convex deficiency of S is defined as the set difference $H - S$.

All the points are defined at discrete coordinates since these are applied on digital images.

A digital set, A, is said to be convex if and only if its Euclidean convex hull only contains digital points belonging to A.

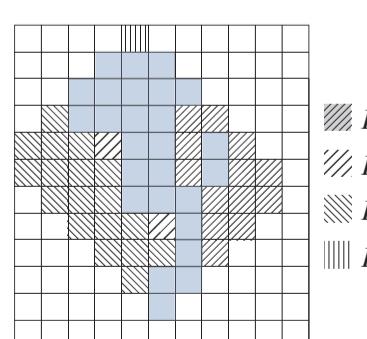
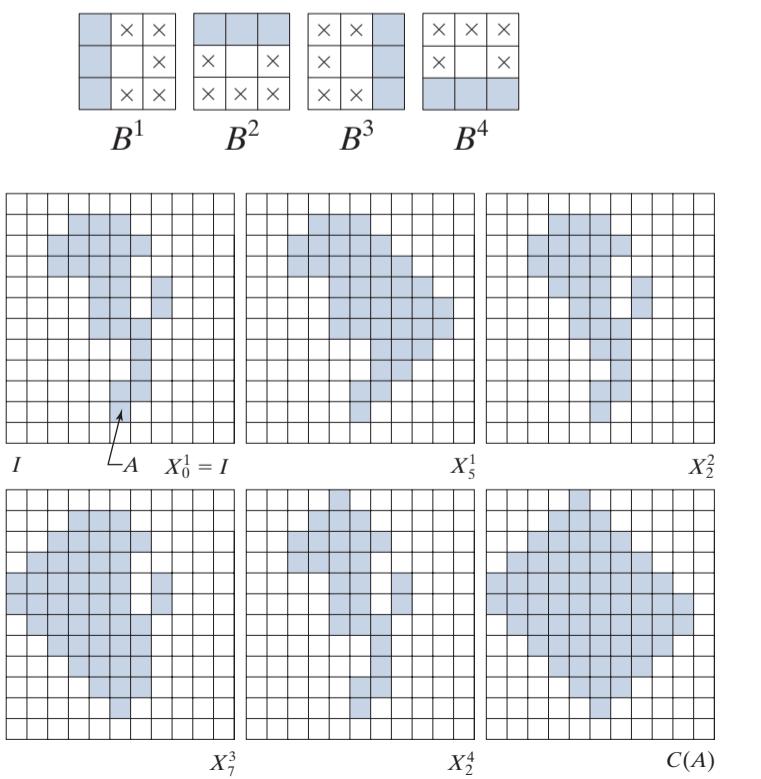
Let B^i , $i = 1, 2, 3, 4$, denote the four SEs. The procedure consists of implementing the morphological equation

$$X_k^i = (X_{k-1}^i \circledast B^i) \cup X_{k-1}^i \quad i = 1, 2, 3, 4 \quad \text{and} \quad k = 1, 2, 3, \dots$$

with $X_0^i = I$. When the procedure converges using the i th SE (i.e., when $X_k^i = X_{k-1}^i$), we let $D^i = X_k^i$. Then, the convex hull of A is the union of the four results:

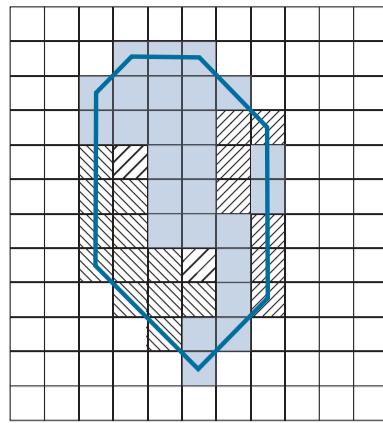
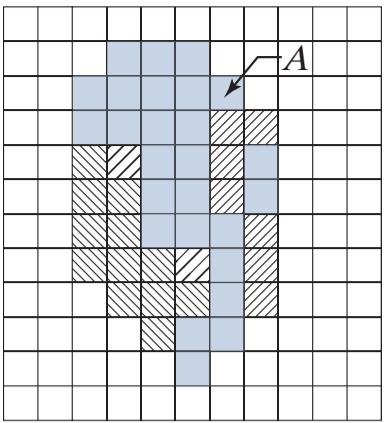
$$C(A) = \bigcup_{i=1}^4 D^i$$

Thus, the method consists of iteratively applying the hit-or-miss transform to I with B^1 until convergence, then letting $D^1 = X^1$, where k is the step at which convergence occurred. The procedure is repeated with B^2 (applied to I) until no further changes occur, and so on. The union of the four resulting D^i constitutes the convex hull of A. The algorithm is initialised with $k = 0$ and $X_0^i = I$ every time that i (i.e., the SE) changes.



(a) SEs (b) Set A.

(c)–(f) Results of convergence with the structuring elements shown in (a). (g) Convex hull. (h) Convex hull showing the contribution of each structuring element.



- (a) Result of limiting growth of the convex hull algorithm.
- (b) Straight lines connecting the boundary points show that the new set is convex also.

Thinning

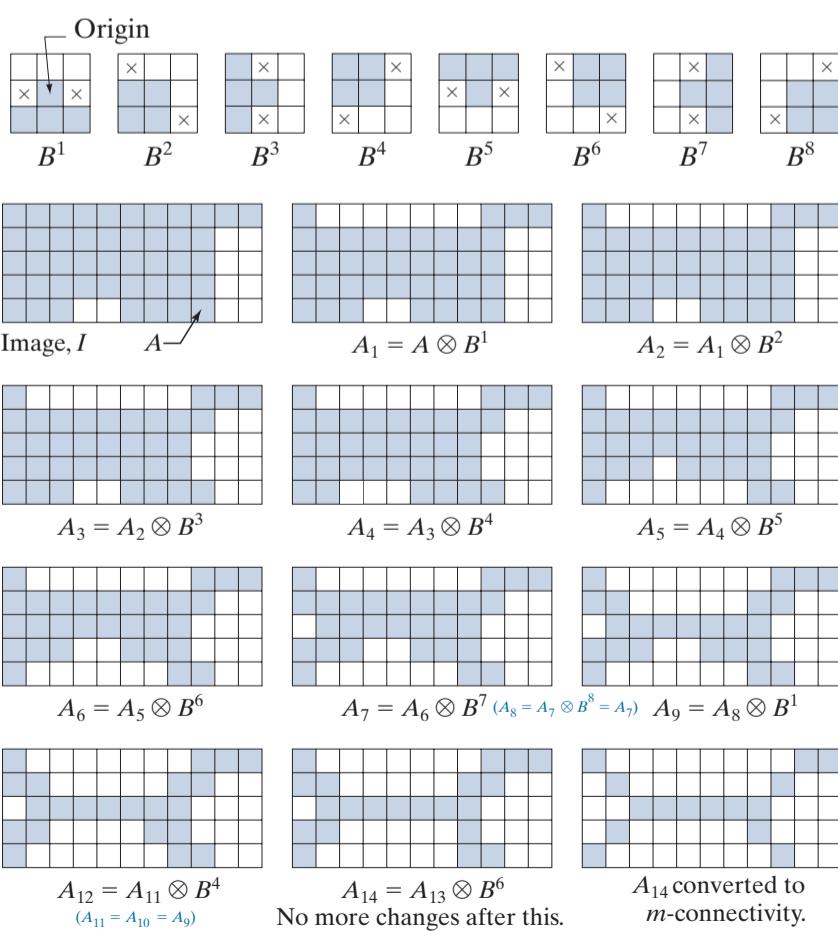
Thinning of a set A of foreground pixels by a SE B, denoted $A \otimes B$, can be defined in terms of the hit-or-miss transform:

$$\begin{aligned} A \otimes B &= A - (A \otimes B) \\ &= A \cap (A \otimes B)^c \end{aligned}$$

A more useful expression for thinning A symmetrically is based on a sequence of SEs:

$$A \otimes \{B\} = \left(\left(\dots \left((A \otimes B^1) \otimes B^2 \right) \dots \right) \otimes B^n \right) \quad \text{Where} \quad \{B\} = \{B^1, B^2, B^3, \dots, B^n\}$$

The process is to thin A by one pass with B^1 , then thin the result with one pass of B^2 , and so on, until A is thinned with one pass of B^n . The entire process is repeated until no further changes occur after one complete pass through all SEs.



- a) SEs.
- b) Set A.
- c) Result of thinning A with B^1 (shaded).
- d) Result of thinning A_1 with B^2 .
- e)-(i) Results of thinning with the next six SEs. (There was no change between A_7 and A_8 .)
- j)-(k) Result of using the first four elements again.
- l) Result after convergence.
- m) Result converted to m-connectivity.

Thickening

Thickening is the morphological dual of thinning and is defined by the expression

$$A \odot B = A \cup (A * B)$$

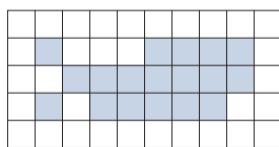
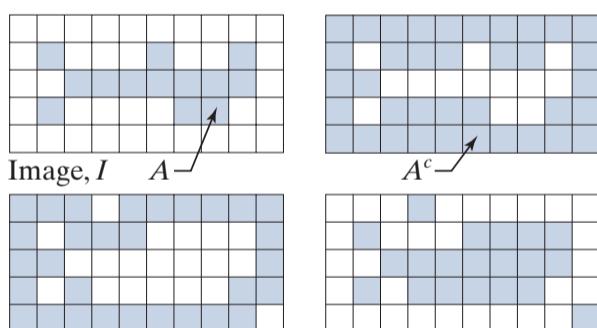
where B is a SE suitable for thickening. As in thinning, thickening can be defined as a sequential operation:

$$A \odot \{B\} = \left(\left(\dots \left((A \odot B^1) \odot B^2 \right) \dots \right) \odot B^n \right)$$

The SEs used for thickening have the same form as those used for thinning, but with all 1's and 0's interchanged.

However, instead of designing a separate algorithm for thickening, the same procedure as thinning can be applied to thin the background of the set and complement of the result will give the thickening operation.

The procedure may result in disconnected points, hence thickening by this method usually is followed by post-processing to remove disconnected points.



Skeletons

(a) If z is a point of $S(A)$, and $(D)_z$ is the largest disk centred at z and contained in A , one cannot find a larger disk (not necessarily centred at z) containing $(D)_z$ and simultaneously included in A . A disk $(D)_z$ satisfying these conditions is called a

maximum disk. (b) If $(D)_z$ is a maximum disk, it touches the boundary of A at two or more different places.

The skeleton of A can be expressed in terms of erosions and openings,

$$S(A) = \bigcup_{k=0}^K S_k(A) \quad \text{with} \quad S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$$

where B is a SE, and $(A \ominus kB)$ indicates k successive erosions starting with A ; that is, A is first eroded by B , the result is eroded by B , and so on until k times, given by,

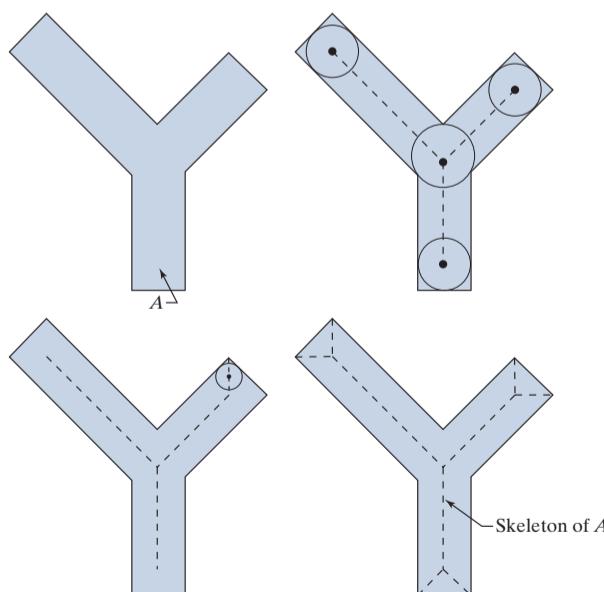
$$(A \ominus kB) = ((\dots((A \ominus B) \ominus B) \ominus \dots) \ominus B)$$

Here K is the last iterative step before A erodes to an empty set.

$$K = \max \{ k \mid (A \ominus kB) \neq \emptyset \}$$

$S(A)$ can be obtained as the union of the skeleton subsets $S_k(A)$, $k = 0, 1, 2, \dots, K$.

$$A = \bigcup_{k=0}^K (S_k(A) \oplus kB) \quad (S_k(A) \oplus kB)$$



(a) Set A . (b) Various positions of maximum disks whose centres partially define the skeleton of A . (c) Another maximum disk, whose center defines a different segment of the skeleton of A . (d) Complete skeleton (dashed).

With successive k successive dilations, starting with $S_k(A)$; that is,

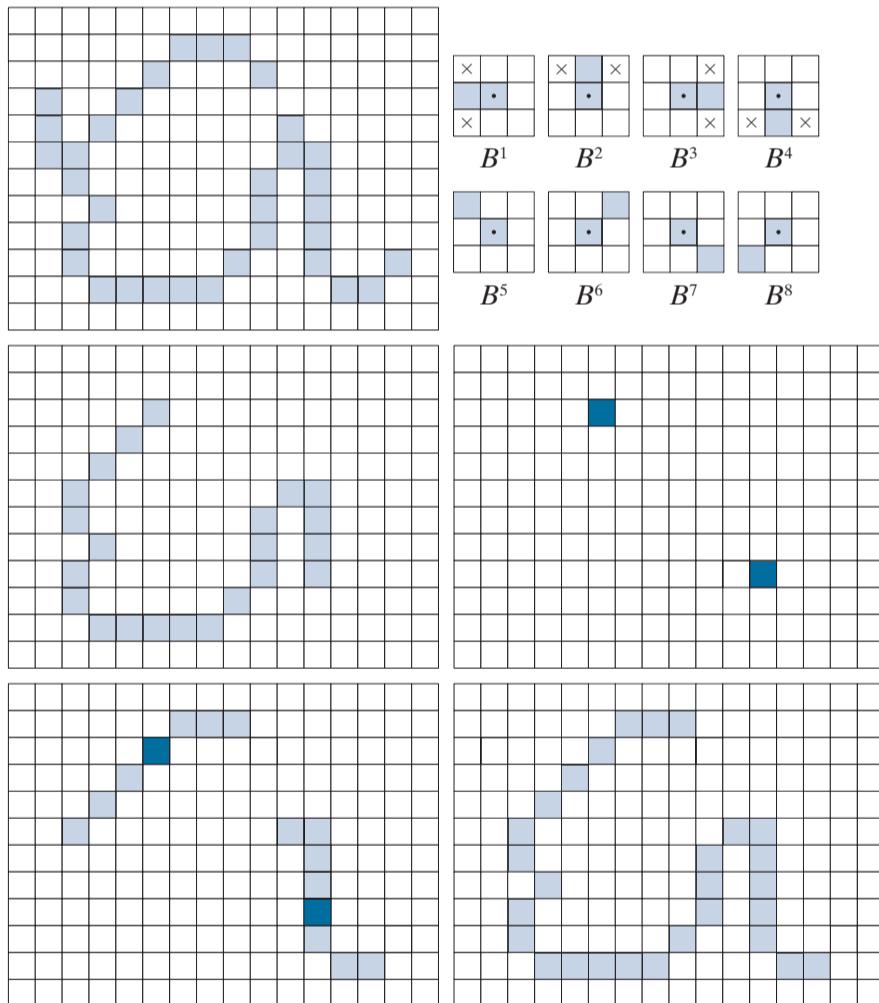
$$(S_k(A) \oplus kB) = ((\dots((S_k(A) \oplus B) \oplus B) \oplus \dots) \oplus B)$$

Pruning

Pruning methods are an essential complement to thinning and skeletonizing algorithms, because these procedures tend to leave spurs (“parasitic” components) that need to be “cleaned up” by post-processing using several of the morphological techniques.

A common approach in the automated recognition of hand-printed characters is to analyze the shape of the skeleton of a character. These skeletons often contain spurs, caused during erosion by noise and non-uniformities in the character strokes.

In this section we develop a morphological technique for handling this problem, starting with the assumption that the length of a parasitic component does not exceed a specified number of pixels.



- (a) Set A of foreground pixels (shaded).
- (b) SEs used for deleting end points.
- (c) Result of three cycles of thinning.
- (d) End points of (c).
- (e) Dilation of end points conditioned on (a).
- (f) Pruned image.

Morphological reconstruction

These operations involve two images and a SE.

The first image, the marker denoted by F, contains the starting points for reconstruction, while the other image, mask denoted by G, constrains (conditions) the reconstruction. SE defines the connectivity (for 2D, 8-connectivity is used).

Geodesic dilation and geodesic erosion

Let F denote the marker image and G the mask image. We assume in this discussion that both are binary images and that $F \subseteq G$.

The geodesic dilation of size 1 of the marker image with respect to the mask, denoted by $D_G^{(1)}(F)$, is defined as

$$D_G^{(1)}(F) = (F \oplus B) \cap G$$

The geodesic dilation of size n of F with respect to G is defined as

$$D_G^{(n)}(F) = D_G^{(1)}\left(D_G^{(n-1)}(F)\right)$$

where $n \geq 1$ is an integer, and $D_G^{(0)}(F) = F$. In this recursive expression, the set intersection as indicated above is performed at each step that ensures that the mask G will limit the growth (dilation) of marker F. By the recursive approach, F grows (dilates) this point successively, with masking of the result at each step by G, thereby yield a result whose shape is influenced by the structure of G. For simple cases, the reconstruction would result in an image identical to G.

The geodesic erosion of size 1 of marker F with respect to mask G is defined as

$$E_G^{(1)}(F) = (F \ominus B) \cup G \quad \text{such that that geodesic erosion of an image remains greater than or equal to its mask image.}$$

The geodesic erosion of size n of F with respect to G is defined as

$$E_G^{(n)}(F) = E_G^{(1)}\left(E_G^{(n-1)}(F)\right)$$

where $n \geq 1$ is an integer and $E_G^{(0)}(F) = F$.

Geodesic dilation and erosion converge after a finite number of iterative steps, because propagation or shrinking of the marker image is constrained by the mask.

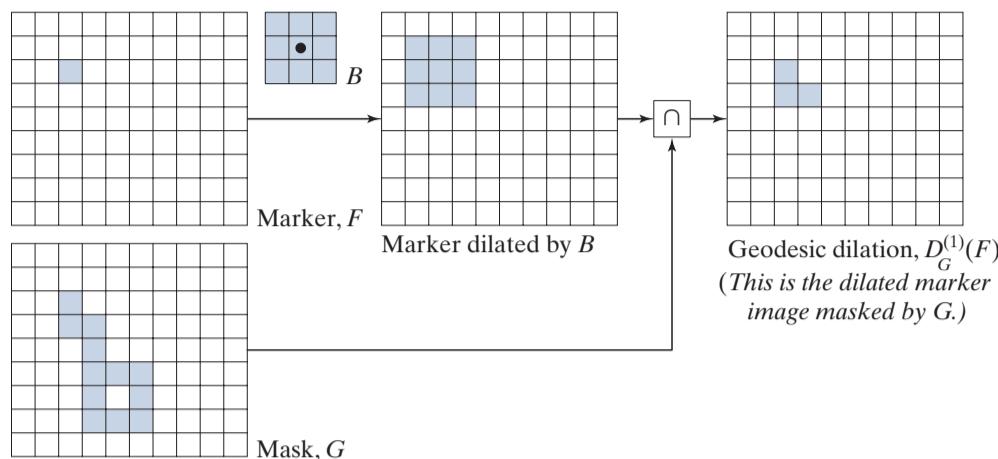


Illustration of a geodesic dilation of size 1. Note that the marker image contains a point from the object in G . If continued, subsequent dilations and maskings would eventually result in the object contained in G .

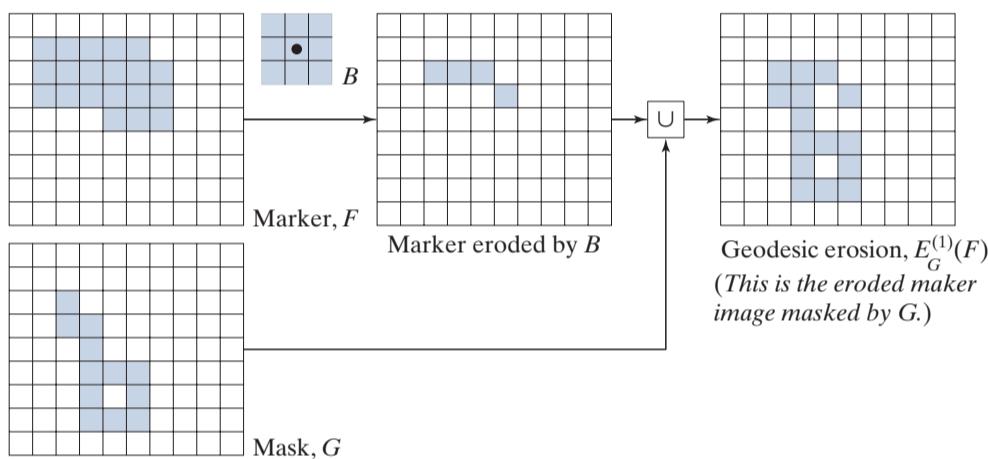


Illustration of a geodesic erosion of size 1.

Morphological reconstruction by dilation & erosion

Morphological reconstruction by dilation of a marker image F with respect to a mask image G , denoted $R_G^D(F)$, is defined as the geodesic dilation of F with respect to G , iterated until stability is achieved; that is,

$$R_G^D(F) = D_G^{(k)}(F)$$

with k such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

In a similar manner, the morphological reconstruction by erosion of a marker image F with respect to a mask image G , denoted $R_G^E(F)$, is defined as the geodesic erosion of F with respect to G , iterated until stability; that is,

$$R_G^E(F) = E_G^{(k)}(F)$$

with k such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$.

Reconstruction by dilation and erosion are duals with respect to set complementation.

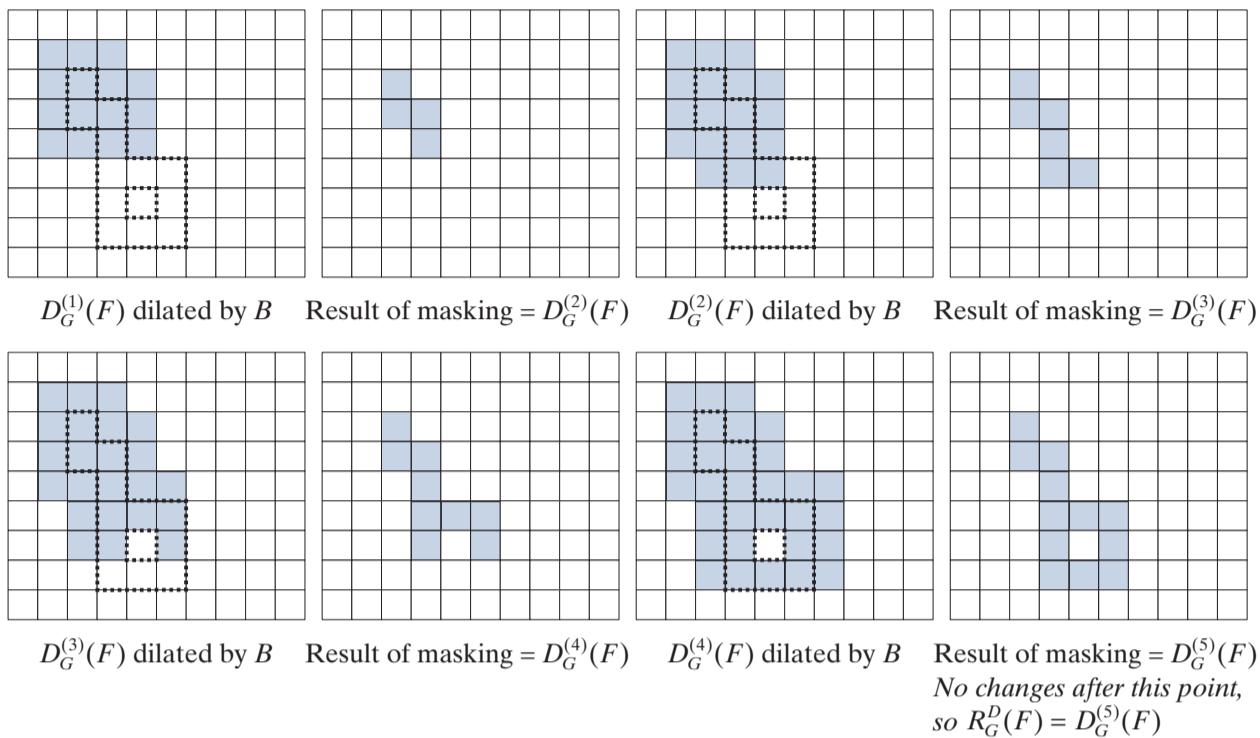


Illustration of morphological reconstruction by dilation. The mask (G) is shown dotted for reference.

Opening by reconstruction

In morphological opening, erosion removes small objects and then dilation attempts to restore the shape of the objects that remain. The accuracy of this restoration depends on the similarity of the shapes and the SEs used.

Opening by reconstruction restores exactly the shapes of the objects that remain after erosion.

The opening by reconstruction of size n of an image F is defined as the reconstruction by dilation of the erosion of size n of F with respect to F; that is,

$$O_R^{(n)}(F) = R_F^D(F \ominus nB)$$

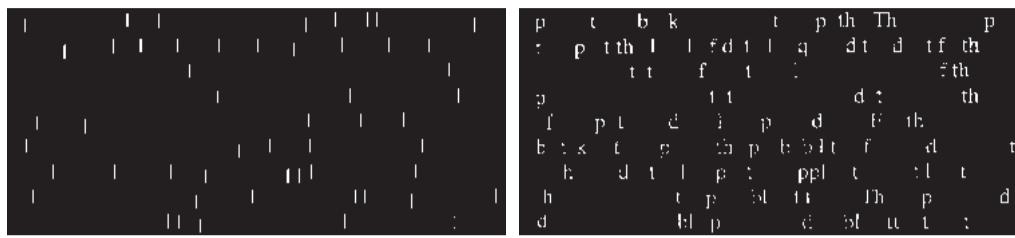
where $F \ominus nB$ indicates n erosions by B starting with F. Note that F itself is used as the mask. Typically, $n=1$.

The SE, B is designed to extract some feature of interest, based on erosion.

ponents or broken connection paths. There is no point in going past the level of detail required to identify those components.

Segmentation of nontrivial images is one of the most difficult tasks in computer vision and image processing. Segmentation accuracy determines the effectiveness of computerized analysis procedures. For this reason, considerable attention must be given to segmentation to be taken to improve the probability of rugged segmentation. In some applications, such as industrial inspection applications, at least some degree of segmentation is possible at times. The experienced image processing designer invariably pays considerable attention to such

problems. The following figure shows a text image and its various processed versions. The original image is shown in (a). (b) shows the result of erosion of (a) with a structuring element of size 51X1 elements (all 1's). (c) shows the result of opening of (a) with the same structuring element, shown for comparison. (d) shows the result of opening by reconstruction.



- (a) Text image of size 918X2018 pixels. The approximate average height of the tall characters is 51 pixels. (b) Erosion of (a) with a structuring element of size 51X1 elements (all 1's). (c) Opening of (a) with the same structuring element, shown for comparison. (d) Result of opening by reconstruction.

Closing by reconstruction

A expression similar to above can be written for closing by reconstruction (we will use erosion instead of dilation). We use R_F^E such that reconstruction works with images in which the background is black (0) and the foreground is white (1).

Operation	Equation	Comments
Translation	$(B)_z = \{c \mid c = b + z, \text{ for } b \in B\}$	Translates the origin of B to point z .
Reflection	$\hat{B} = \{w \mid w = -b, \text{ for } b \in B\}$	Reflects B about its origin.
Complement	$A^c = \{w \mid w \notin A\}$	Set of points not in A .
Difference	$A - B = \{w \mid w \in A, w \notin B\} \\ = A \cap B^c$	Set of points in A , but not in B .
Erosion	$A \ominus B = \{z \mid (B)_z \subseteq A\}$	Erodes the boundary of A . (I)
Dilation	$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\}$	Dilates the boundary of A . (I)
Opening	$A \circ B = (A \ominus B) \oplus B$	Smoothes contours, breaks narrow isthmuses, and eliminates small islands and sharp peaks. (I)
Closing	$A \bullet B = (A \oplus B) \ominus B$	Smoothes contours, fuses narrow breaks and long thin gulfs, and eliminates small holes. (I)
Hit-or-miss transform	$I \circledast B = \{z \mid (B)_z \subseteq I\}$	Finds instances of B in image I . B contains <i>both</i> foreground and background elements.
Boundary extraction	$\beta(A) = A - (A \ominus B)$	Set of points on the boundary of set A . (I)
Hole filling	$X_k = (X_{k-1} \oplus B) \cap I^c \\ k = 1, 2, 3, \dots$	Fills holes in A . X_0 is of same size as I , with a 1 in each hole and 0's elsewhere. (II)
Connected components	$X_k = (X_{k-1} \oplus B) \cap I \\ k = 1, 2, 3, \dots$	Finds connected components in I . X_0 is a set, the same size as I , with a 1 in each connected component and 0's elsewhere. (I)
Convex hull	$X_k^i = (X_{k-1}^i \circledast B^i) \cup X_{k-1}^i; \\ i = 1, 2, 3, 4 \quad k = 1, 2, 3, \dots$ $X_0^i = I; D^i = X_{conv}^i; C(A) = \bigcup_{i=1}^4 D^i$	Finds the convex hull, $C(A)$, of a set, A , of foreground pixels contained in image I . X_{conv}^i means that $X_k^i = X_{k-1}^i$. (III)

Operation	Equation	Comments
Thinning	$A \otimes B = A - (A * B)$ $= A \cap (A * B)^c$ $A \otimes \{B\} =$ $\left(\dots \left((A \otimes B^1) \otimes B^2 \right) \dots \right) \otimes B^n$ $\{B\} = \{B^1, B^2, B^3, \dots, B^n\}$	Thins set A . The first two equations give the basic definition of thinning. The last two equations denote thinning by a sequence of structuring elements. This method is normally used in practice. (IV)
Thickening	$A \odot B = A \cup (A * B)$ $A \odot \{B\} =$ $\left(\dots \left((A \odot B^1) \odot B^2 \right) \dots \right) \odot B^n$	Thickens set A using a sequence of structuring elements, as above. Uses (IV) with 0's and 1's reversed.
Skeletons	$S(A) = \bigcup_{k=0}^K S_k(A)$ $S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$ <p>Reconstruction of A:</p> $A = \bigcup_{k=0}^K (S_k(A) \oplus kB)$	Finds the skeleton $S(A)$ of set A . The last equation indicates that A can be reconstructed from its skeleton subsets $S_k(A)$. K is the value of the iterative step after which the set A erodes to the empty set. The notation $(A \ominus kB)$ denotes the k th iteration of successive erosions of A by B . (I)
Pruning	$X_1 = A \otimes \{B\}$ $X_2 = \bigcup_{k=1}^8 (X_1 * B^k)$ $X_3 = (X_2 \oplus H) \cap A$ $X_4 = X_1 \cup X_3$	X_4 is the result of pruning set A . The number of times that the first equation is applied to obtain X_1 must be specified. Structuring elements (V) are used for the first two equations. In the third equation H denotes structuring element. (I)
Geodesic dilation-size 1	$D_G^{(1)}(F) = (F \oplus B) \cap G$	F and G are called the <i>marker</i> and the <i>mask</i> images, respectively. (I)
Geodesic dilation-size n	$D_G^{(n)}(F) = D_G^{(1)}\left(D_G^{(n-1)}(F)\right)$	Same comment as above.
Geodesic erosion-size 1	$E_G^{(1)}(F) = (F \ominus B) \cup G$	Same comment as above.
Geodesic erosion-size n	$E_G^{(n)}(F) = E_G^{(1)}\left(E_G^{(n-1)}(F)\right)$	Same comment as above.
Morphological reconstruction by dilation	$R_G^D(F) = D_G^{(k)}(F)$	With k is such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

Operation	Equation	Comments
Morphological reconstruction by erosion	$R_G^E(F) = E_G^{(k)}(F)$	With k such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$.
Opening by reconstruction	$O_R^{(n)}(F) = R_F^D(F \ominus nB)$	$F \ominus nB$ indicates n successive erosions by B , starting with F . The form of B is application-dependent.
Closing by reconstruction	$C_R^{(n)}(F) = R_F^E(F \oplus nB)$	$F \oplus nB$ indicates n successive dilations by B , starting with F . The form of B is application-dependent.
Hole filling	$H = [R_{I^c}^D(F)]^c$	H is equal to the input image I , but with all holes filled. See Eq. (9-45) for the definition of marker image F .
Border clearing	$X = I - R_I^D(F)$	X is equal to the input image I , but with all objects that touch (are connected to) the boundary removed. See Eq. (9-47) for the definition of marker image F .

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-314

Implementation of operators

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Implementation of operators



Dilation - grow image regions



Erosion - shrink image regions



Opening - structured removal of image region boundary pixels



Closing - structured filling in of image region boundary pixels



Hit and Miss Transform - image pattern matching and marking



Thinning - structured erosion using image pattern matching



Thickening - structured dilation using image pattern matching



Skeletonization/Medial Axis Transform-finding skeletons of binary regions

Please check the following in the programming language/library of your choice:

1. Image thresholding
2. Binary image Hole fillings
3. Connected components labelling (very important)
4. Distance transform
5. Check ‘regionprops’ command in OpenCV/Matlab
6. Finding object boundary

- All these mentioned operators are available and well-documented with examples in both Python and Matlab.
- Kindly go through them carefully.
- Make sure that you start with the correct image type (these require binary image type as input) before implementing the operator.
- Please get yourself familiarised with the operators (i.e. work on a few examples by yourself).
- You may take any example image and follow the documentation to see the working.

ALL THE BEST FOR THE MID-SEMESTER EXAM

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE409/609/DSE-312

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

The objective of segmentation is to partition an image into regions. This can be achieved by finding boundaries between regions based on discontinuities in intensity levels (edge based methods) and then threshold operations, as we saw in the previous lecture. Other set of techniques include finding the regions directly.

Region growing

Region growing groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach is to start with a set of “seed” points, and from these grow regions by appending to each seed those neighbouring pixels that have predefined properties similar to the seed (such as ranges of intensity or color).

- In region growing algorithm the pixels are grouped based on some criteria to get sub regions in the image.
- Seed point or number of seed points in original image should be selected.
- Select similar criteria for segmentation then expand the region by adding to each seed the close pixel with properties similar to seed pixel and finally if there is no pixel has the required criterion, region growing should stop.

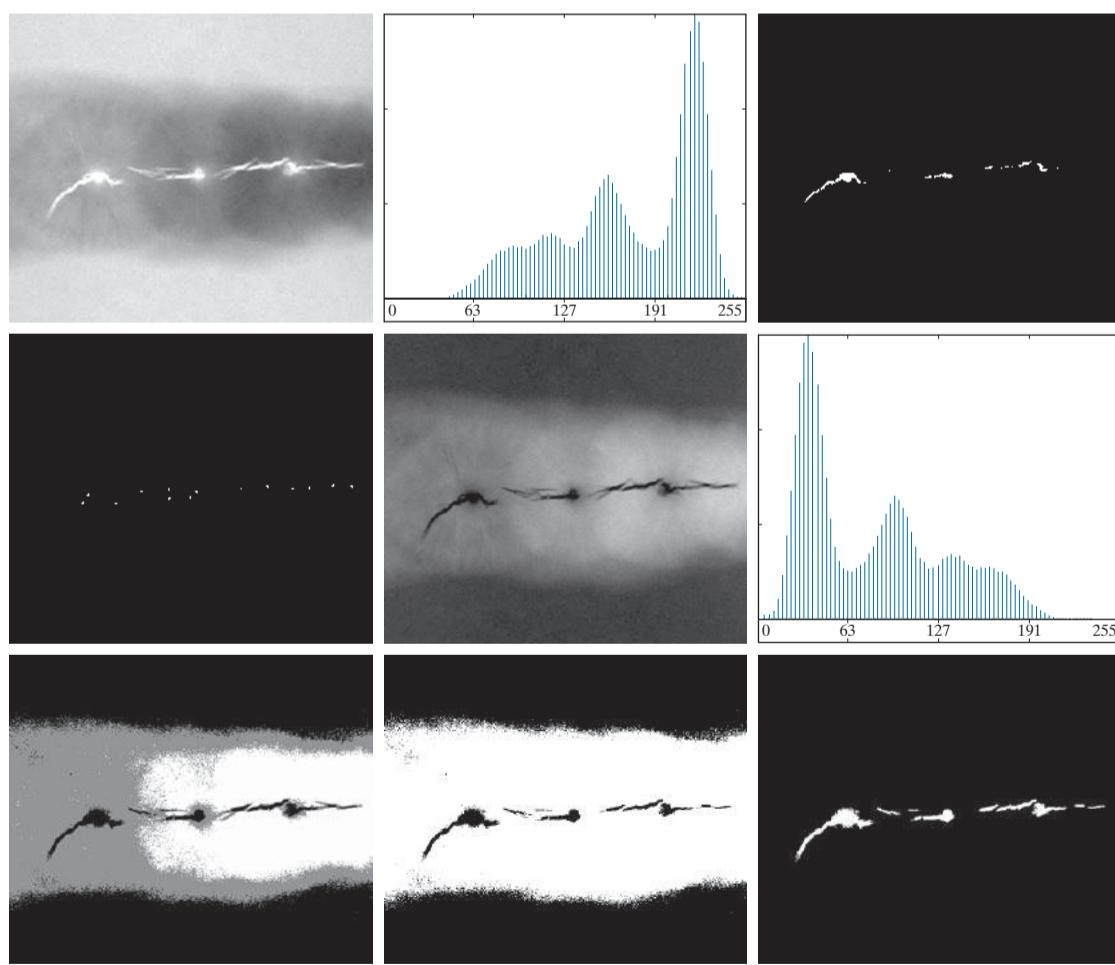
The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. The following information should be considered:

- When a priori information is not available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process. If the result of these computations shows clusters of values, the pixels whose properties place them near the centroid of these clusters can be used as seeds.
- Connectivity rule: Connectivity properties are to be used in the region-growing process. Hence, grouping pixels should go hand-in hand with connectivity information.
- Stopping rule: Region growth should stop when no more pixels satisfy the criteria for inclusion in that region. The rules can be built on descriptors such as size, likeness between a candidate pixel and the pixels grown so far (such as a

comparison of the intensity of a candidate and the averages intensity of the grown region), and the shape of the region being grown.

Let: $f(x,y)$ denote an input image; $S(x,y)$ denote a seed array containing 1's at the locations of seed points and 0's elsewhere; and Q denote a predicate to be applied at each location (x, y) . Arrays f and S are assumed to be of the same size. A basic region-growing algorithm based on 8-connectivity may be stated as follows.

1. Find all connected components in $S(x,y)$ and reduce each connected component to one pixel; label all such pixels found as 1. All other pixels in S are labeled 0.
2. Form an image f_Q such that, at each point (x, y) , $f_Q(x, y) = 1$ if the input image satisfies a given predicate, Q , at those coordinates, and $f_Q(x, y) = 0$ otherwise.
3. Let g be an image formed by appending to each seed point in S all the 1-valued points in f_Q that are 8-connected to that seed point.
4. Label each connected component in g with a different region label (e.g., integers or letters). This is the segmented image obtained by region growing.



(a) X-ray image of a defective weld. (b) Histogram. (c) Initial seed image. (d) Final seed image (points were enlarged for clarity). (e) Absolute value of the difference between the seed value (255) and (a). (f) Histogram of (e). (g) Difference image thresholded using dual thresholds. (h) Difference image thresholded with the smallest of the dual thresholds. (i) Segmentation result obtained by region growing.

Edge based methods look for boundaries, however region growing methods look for spatial information within a neighbourhood (therefore works well for good connectivity). However, the region growing method can be computationally expensive, it depends on the initial seed point and works well for good connectivity. For complex problems (more noise, low contrast), the final segmentation result could be sensitive to initial seed locations.

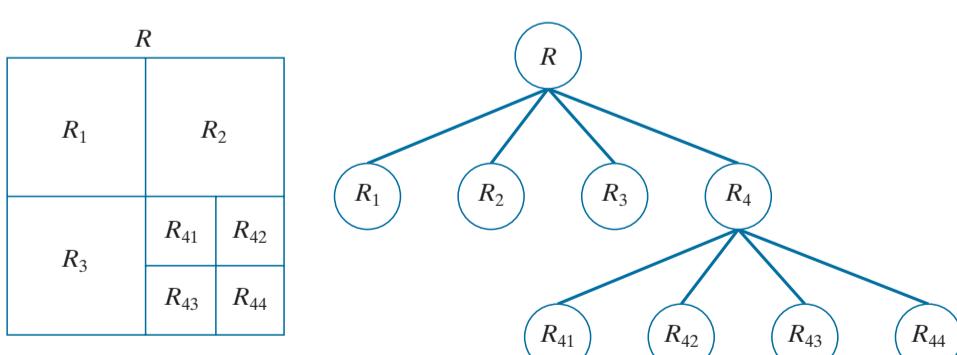
Region merging and splitting

This procedure sub-divides an image initially into a set of disjoint regions and then merge and/or split the regions to achieve segmentation.

- Let R represent the entire image region and select a predicate Q (Q is a condition).
- We subdivide the image (R) successively into smaller and smaller quadrant regions so that, for any region R_i , $Q(R_i) = \text{TRUE}$.
- We start with the entire region, R. If $Q(R) = \text{FALSE}$, we divide the image into quadrants.
- If Q is FALSE for any quadrant, we subdivide that quadrant into sub-quadrants, and so on.

The splitting technique is called quadtrees; trees in which each node has exactly four descendants (the images corresponding to the nodes of a quadtree sometimes are called quadregions or quadimages).

The root of the tree corresponds to the entire image (R), and that each node corresponds to the subdivision of a node into four descendant nodes. In this case, only R_4 was subdivided further as shown below.



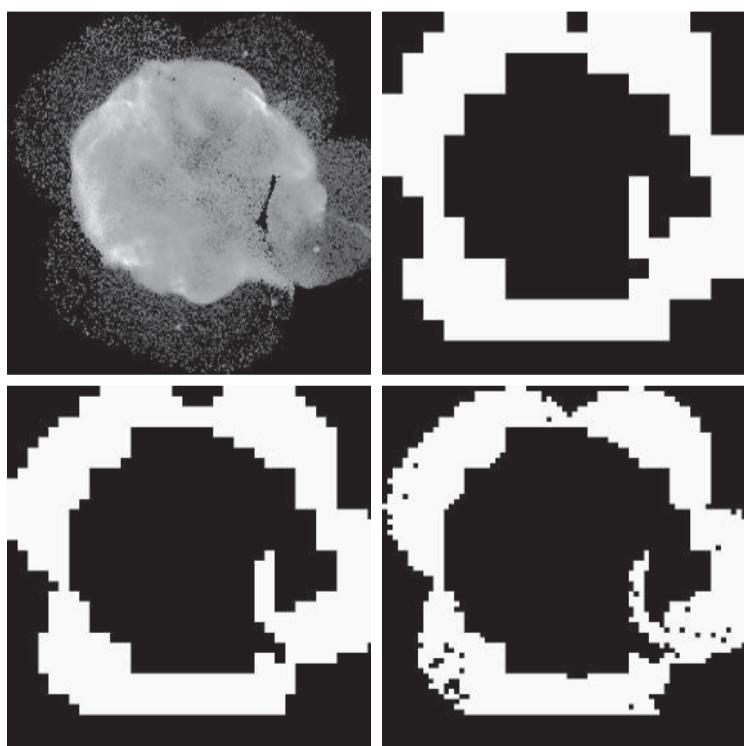
- (a) Partitioned image.
 (b) Corresponding quadtree. R represents the entire image region.

- If only splitting is used, the final partition contains adjacent regions with identical properties.
- Hence allowing merging as well as splitting, based on a predicate Q can be implemented.
- Two adjacent regions R_j and R_k are merged only if $Q(R_j \cup R_k) = \text{TRUE}$.

In summary;

1. Split into four disjoint quadrants any region R_i for which $Q(R_i) = \text{FALSE}$.
2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $Q(R_j \cup R_k) = \text{TRUE}$.
3. Stop when no further merging is possible.

Properties based on the mean and standard deviation of pixel intensities in a region attempt to quantify the texture of the region. The concept of texture segmentation is based on using measures of texture in the predicates.



(a) X-ray image of the Cygnus Loop supernova
 (b) through (d) Results of limiting the smallest allowed quad-region to be of sizes of 32X32, 16X16, and 8X8 pixels, respectively.

Using predicate:

$$Q(R) = \begin{cases} \text{TRUE} & \text{if } \sigma_R > a \text{ AND } 0 < m_R < b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where σ_R and m_R are the standard deviation and mean of the region being processed, and 'a' and 'b' are nonnegative constants.

Region segmentation using K-means clustering

In k-means clustering, each observation is assigned to the cluster with the nearest mean and each mean is called the prototype of its cluster.

- By using an iterative procedure, k-means algorithm successively refines the mean until convergence is achieved.
- Let $\{z_1, z_2, \dots, z_Q\}$ be set of vector observations (samples) of the form:

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}$$

- Each component of a vector \mathbf{z} represents a numerical pixel attribute (for grayscale images, scalar representing the intensity of a pixel).
- The objective of k-means clustering is to partition the set Q of observations into k ($k \leq Q$) disjoint cluster sets $C = \{C_1, C_2, \dots, C_k\}$, so that the following criterion of optimality is satisfied:

$$\arg \min_C \left(\sum_{i=1}^k \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2 \right)$$

- where \mathbf{m}_i is the mean vector (or centroid) of the samples in set C_i and $\|\cdot\|$ is the vector norm of the argument.
- Typically, the Euclidean norm is used, so the term $\|\mathbf{z} - \mathbf{m}_i\|$ is the familiar Euclidean distance from a sample in C_i to mean \mathbf{m}_i .
- The basic idea here is to find the sets $C = \{C_1, C_2, \dots, C_k\}$ such that the sum of the distances from each point in a set to the mean of that set is minimum.

The “standard” k-means algorithm, based on the Euclidean distance for a given set $\{z_1, z_2, \dots, z_Q\}$ of vector observation and a specified value of k , is as follows:

- 1. Initialize the algorithm:** Specify an initial set of means, $\mathbf{m}_i(1)$, $i = 1, 2, \dots, k$.
- 2. Assign samples to clusters:** Assign each sample to the cluster set whose mean is the closest (ties are resolved arbitrarily, but samples are assigned to only *one* cluster):

$$\mathbf{z}_q \rightarrow C_i \text{ if } \|\mathbf{z}_q - \mathbf{m}_i\|^2 < \|\mathbf{z}_q - \mathbf{m}_j\|^2 \quad j = 1, 2, \dots, k \quad (j \neq i); \quad q = 1, 2, \dots, Q$$

- 3. Update the cluster centers (means):**

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \quad i = 1, 2, \dots, k$$

where $|C_i|$ is the number of samples in cluster set C_i .

- 4. Test for completion:** Compute the Euclidean norms of the differences between the mean vectors in the current and previous steps. Compute the residual error, E , as the sum of the k norms. Stop if $E \leq T$, where T a specified, nonnegative threshold. Else, go back to Step 2.



- (a) Image of size 688X688 pixels.
- (b) Image segmented using the k-means algorithm with $k = 3$.

When $T=0$, this algorithm is known to converge in a finite number of iterations to a local minimum. It is not guaranteed to yield the global minimum required to minimize. The result at convergence does depend on the initial values chosen for m_i . An approach used frequently in data analysis is to specify the initial means as k randomly chosen samples from the given sample set, and to run the algorithm several times, with a new random set of initial samples each time. This is to test the “stability” of the solution.

Image segmentation using Gaussian mixture models (clustering technique)

The Gaussian mixture model is a statistical model that can well describe spatial distribution and characteristics of the data in the parameter space. Gaussian mixture model is defined as the m-linear combination of Gaussian density function, i.e.,

$$p(x) = \sum_{i=1}^M \pi_i N_i(x | \mu_i, C_i)$$

Where $N_i(x | \mu_i, C_i)$

denotes a bivariate or trivariate normal distribution with mean vector μ_i and a covariance matrix C_i . π_i is a mixture proportion for each group, and it is regarded as the ith prior probability of Gaussian distribution that data sample produces. These prior probabilities satisfy

$$\sum_{i=1}^M \pi_i = 1 \text{ and } 0 \leq \pi_i \leq 1$$

The likelihood is generally modelled as a normal/gaussian distribution, where the density function $p(x | \theta_i)$ of region segmentation using Gaussian function is modelled as

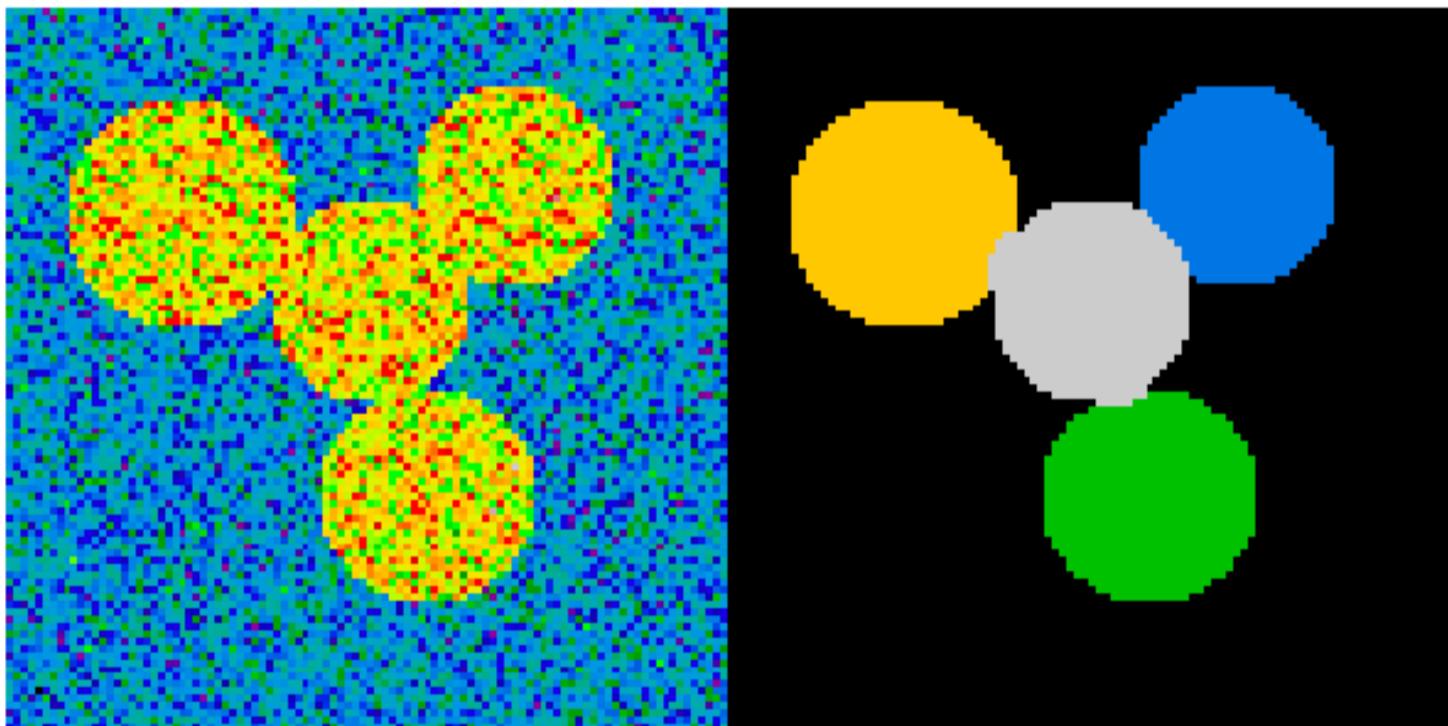
$$p(x | \theta_i) = N(x | \mu, C) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|C|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu)^T C^{-1} (x - \mu) \right\}$$

For a given mixture model, using the model for data classification or clustering must also determine unknown parameters that the various Gaussian components of the model contain, and these unknown parameters are π_i , μ_i and C_i . There are many ways to estimate these parameters, and the most commonly way is EM algorithm based on the maximum likelihood estimation.

Maximum likelihood estimation enables the likelihood function maximization to obtain the parameter estimated value.

Both, the k-means and mixtures of Gaussians techniques use a parametric model of the density function to cluster pixels. In GMM, the assumption is that the density is the superposition of a small number of simpler distributions (e.g., Gaussians) whose locations (centers) and shape (covariance) can be estimated.

Let us take this example, we have background pixels and 4 circular objects. With k-means, we can give a cluster size of 5 and the algorithm will cluster the pixels. However, note that the algorithm is sensitive to the input parameter. Suppose, you give, cluster size =2, then it will separate them into foreground and background objects. All the more, since all of these are touching each other. The GMM approach uses a probability approach to say where a pixel belongs to a particular segment or not.



Here is another example showing image segmentation with 4-clusters.



This means that instead of saying a pixel is flower/leaves or background we want to say that the pixel is flower with 50% probability and leaves with 20% probability, while background with 30% probability. Let us denote $p(C_{\text{flower}}|x)$, $p(C_{\text{leaves}}|x)$ for flower and leaves , i.e., probability that the pixel is flower or leaves respectively.

If we want to classify a pixel as flower, leaves or background we have a total of three classifiers one for each type. We formulate the problem mathematically. In each classifier, we want to find $p(C_l|x)$, where C_l denotes the ‘pixel label’ which could be flower, leaves or background label.

So, $(1 - p(C_{\text{flower}}|x))$ gives the probability that the pixel is not a flower which includes both background and leaves pixels. We use the Bayes rule,

$$p(C_l|x) = \frac{p(x|C_l) p(C_l)}{\sum_{i=1}^l p(x|C_i)}$$

$p(C_l|x)$ is the conditional probability of a pixel label given the pixel type observation and is called the Posterior.

$p(x|C_l)$ is the conditional probability of a pixel label observation, given the label and is called the Likelihood.

The prior is used to increase/decrease the probability of certain pixel label. If nothing about the prior is known, a common choice is to use a uniform distribution, i.e., all the pixel labels are equally likely.

So, why use a Gaussian distribution to model the likelihood?

This comes from the Central Limit theorem (i.e. if we take the mean of the samples (n) and plot the frequencies of their mean, we get a normal distribution! And as the sample size (n) increases --> approaches infinity, we obtain a normal distribution).

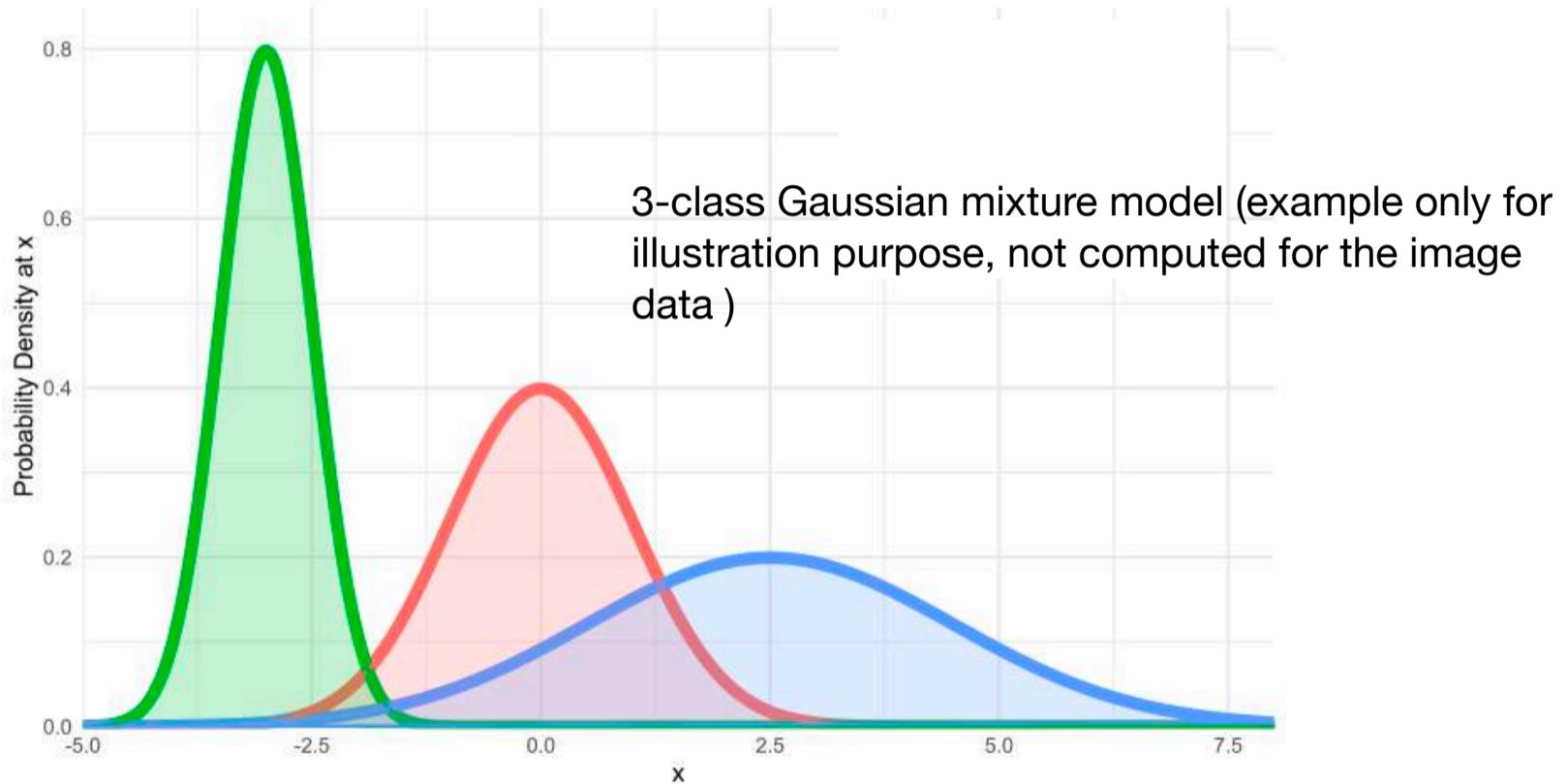
So, if we average a lot of (theoretically infinite) independently identically distributed (i.i.d) random samples, their distribution tends to become a Gaussian.

For the problem of pixel label classification, we need to simply find the mean (μ) and covariance (Σ) of the likelihood gaussian distributions.

The mean is computed from the sample mean. The covariance matrix is a square and a symmetric matrix consisting of variance in each of the individual pixel labels (in our example we have 3 pixel label classes). The off-diagonal correlation terms show the co-occurrence of one label over other. Mathematically, it signifies the

vector projection of one label over the other. In general , several gaussians can be fitted such that the covariance matrix is of the form:

$$= \begin{bmatrix} \text{Var}(R_1) & \text{Cov}(R_1, R_2) & \dots & \text{Cov}(R_1, R_d) \\ \text{Cov}(R_2, R_1) & \text{Var}(R_2) & & \text{Cov}(R_2, R_d) \\ \vdots & & \ddots & \vdots \\ \text{Cov}(R_d, R_1) & \text{Cov}(R_d, R_2) & \dots & \text{Var}(R_d) \end{bmatrix},$$



In practise, you need to Fit Gaussian mixture model to data (for segmentation, this is the pixel data) and also give the number of clusters as input parameter.

The histogram of pixel intensity is plotted as a probability distribution, for 'n' number of clusters , we will have 'n' Gaussian models.

GMM distribution: mixture of weighted sum of 'n' Gaussians

Each Gaussian has its own mean, width and sum of mixtures is equal to 1.

Instead of having the standard sigma parameters (σ), we use a covariance matrix.

This is because GMM can take high dimensional space.

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-312

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Local Feature Detectors: Harris, LBP

Keypoint is a (locally) distinct location in an image, for example, corners.

The feature descriptor summarises the local structure around the keypoint.

The feature vector can be used for tasks such as classification, detection, and recognition.

Detecting Corners (keypoint) within an image

- A corner is a rapid change of direction in a curve.
- Corners are distinctive and invariant to viewpoint.
- Corners are invariant to translation, rotation, and illumination
- Because of these characteristics, corners are used routinely for matching image features in applications such as tracking for autonomous navigation, stereo machine vision algorithms, and image database queries.
- Corner => two edges in roughly orthogonal directions
- An Edge => a sudden brightness change

The Harris Corner detector

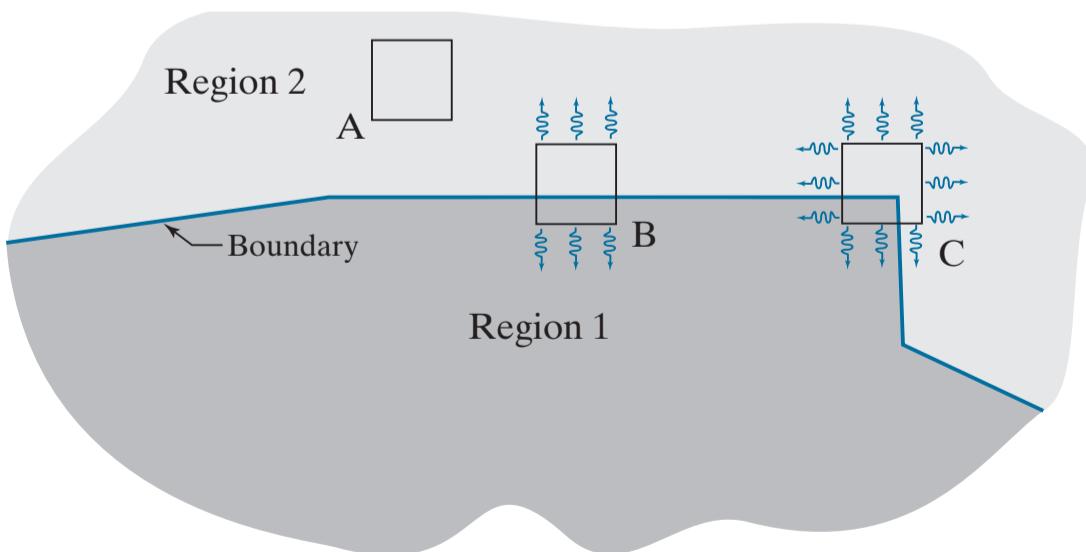


Illustration of how the Harris-corner detector operates in the three types of sub-regions indicated by A (flat), B (edge), and C (corner). The wiggly arrows indicate graphically a directional response in the detector as it moves in the three areas

In the Harris corner approach, the corners are detected by running a small window over an image,(similar to spatial filtering). The detector window is designed to compute intensity changes.

We are interested in three scenarios:

- (1) areas of zero (or small) intensity changes in all directions, which happens when the window is located in a constant (or nearly constant) region, as in location A;
- (2) areas of changes in one direction but no (or small) changes in the orthogonal direction, which this happens when the window spans a boundary between two regions, as in location B;
- (3) areas of significant changes in all directions or at least 2-directions, a condition that happens when the window contains a corner (or isolated points), as in location C.

The HS corner detector is a mathematical formulation that attempts to differentiate between these three conditions.

Let f denote an image, and let $f(s,t)$ denote a patch of the image defined by the values of (s,t) . A patch of the same size, but shifted by (x,y) , is given by $f(s+x, t+y)$. Then, the weighted sum of squared differences between the two patches is given by

$$C(x,y) = \sum_s \sum_t w(s,t) [f(s+x, t+y) - f(s,t)]^2$$

where $w(s,t)$ is a weighting function. The shifted patch can be approximated by the linear terms of a Taylor expansion

$$f(s+x, t+y) \approx f(s,t) + xf_x(s,t) + yf_y(s,t)$$

where $f_x(s,t) = \partial f / \partial x$ and $f_y(s,t) = \partial f / \partial y$, both evaluated at (s,t) . Now, we have,

$$C(x,y) = \sum_s \sum_t w(s,t) [xf_x(s,t) + yf_y(s,t)]^2$$

In the matrix form,

$$C(x,y) = [x \ y] \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{M} = \sum_s \sum_t w(s,t) \mathbf{A}$$

$$\mathbf{A} = \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

Here A is called as the 'structure matrix'.

If $w(s,t)$ is isotropic, then M is symmetric because A is.

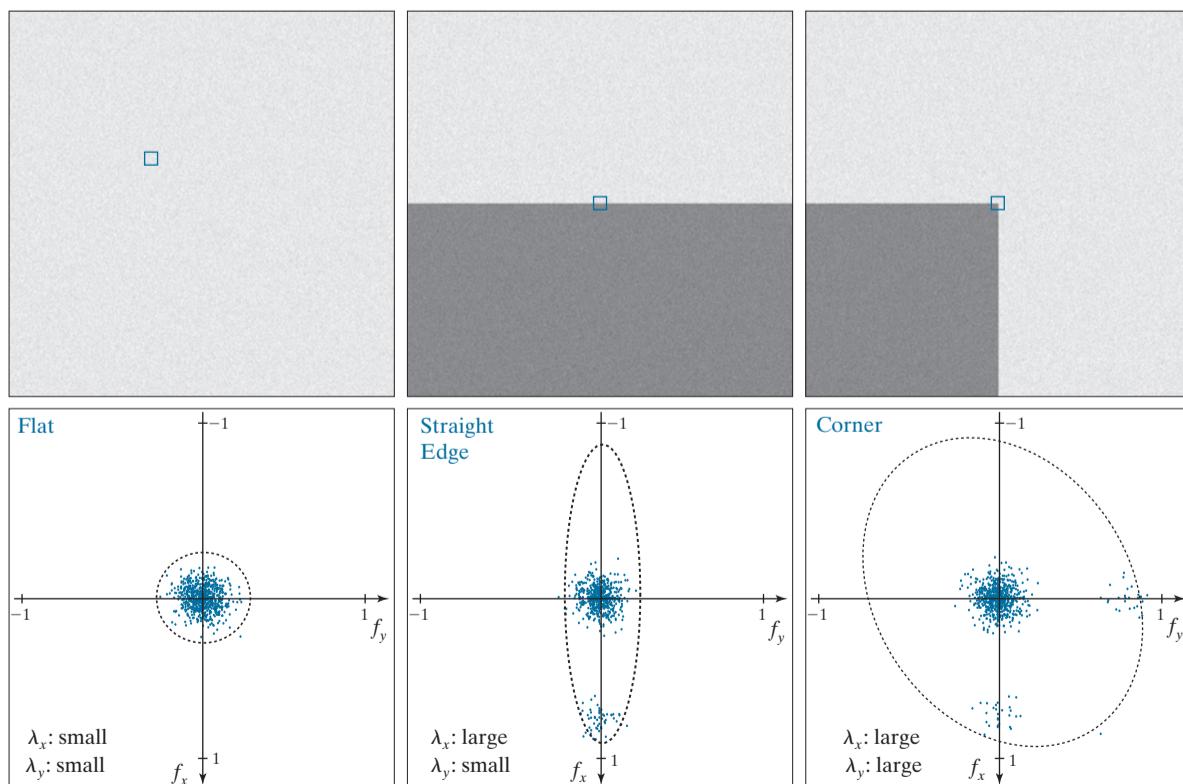
The weighting function $w(s,t)$ used in the HS detector generally has one of two forms:

- (1) it is 1 inside the patch and 0 elsewhere i.e., it has the shape of a box lowpass filter kernel (used when computational speed is paramount and the noise level is low), OR
- (2) it is an exponential function of the form (used when data smoothing is important).

$$w(s,t) = e^{-(s^2 + t^2)/2\sigma^2}$$

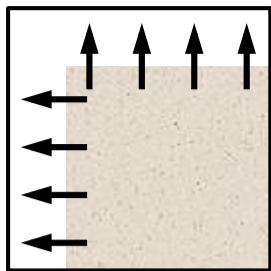
How do we actually detect corners using the structure matrix?

The eigenvectors of a real, symmetric matrix (such as M above) point in the direction of maximum data spread, and the corresponding eigenvalues are proportional to the amount of data spread in the direction of the eigenvectors. In fact, the eigenvectors are the major axes of an ellipse fitting the data, and the magnitude of the eigenvalues are the distances from the center of the ellipse to the points where it intersects the major axes.



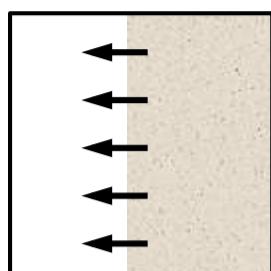
Plots of value pairs (f_x, f_y) showing the characteristics of the eigenvalues of M in image regions (noisy image with no features, edge, corner).
Large eigenvalues detect the presence of a corner in an image patch.

Eigenvalue pairs for the three types of features

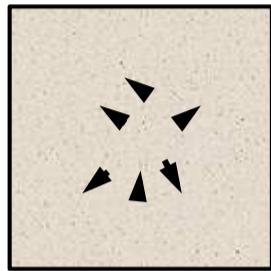


$$\rightarrow M = \begin{bmatrix} \gg 1 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$

Considers points as corners if their structure matrix has two large Eigenvalues



$$\rightarrow M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \gg 1 \end{bmatrix}$$



$$\rightarrow M = \begin{bmatrix} \sim 0 & \sim 0 \\ \sim 0 & \sim 0 \end{bmatrix}$$

The diagonal elements of the structure matrix already can distinguish between corners and edges. We consider the eigenvalues of the matrix for comparison.

Computing the Structure Matrix

Matrix is built from the image gradients

$$D_x^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad D_y^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- The eigenvalues of the matrix formed from derivatives in the image patch can be used to differentiate between the three scenarios of interest.
- Instead of using the eigenvalues (which are expensive to compute), the HS detector utilizes a measure of corner response based on:
 - the trace of a square matrix which is equal to the sum of its eigenvalues;
 - its determinant is equal to the product of its eigenvalues.

$$\begin{aligned}
R &= \lambda_x \lambda_y - k(\lambda_x + \lambda_y)^2 \\
&= \det(\mathbf{M}) - k \text{trace}^2(\mathbf{M})
\end{aligned}$$

k as a “sensitivity factor;” determined empirically. The smaller it is, the more likely the detector is to find corners.

Typically, R is used with a threshold, T . We say that a corner at an image location has been detected only if $R > T$ for a patch at that location

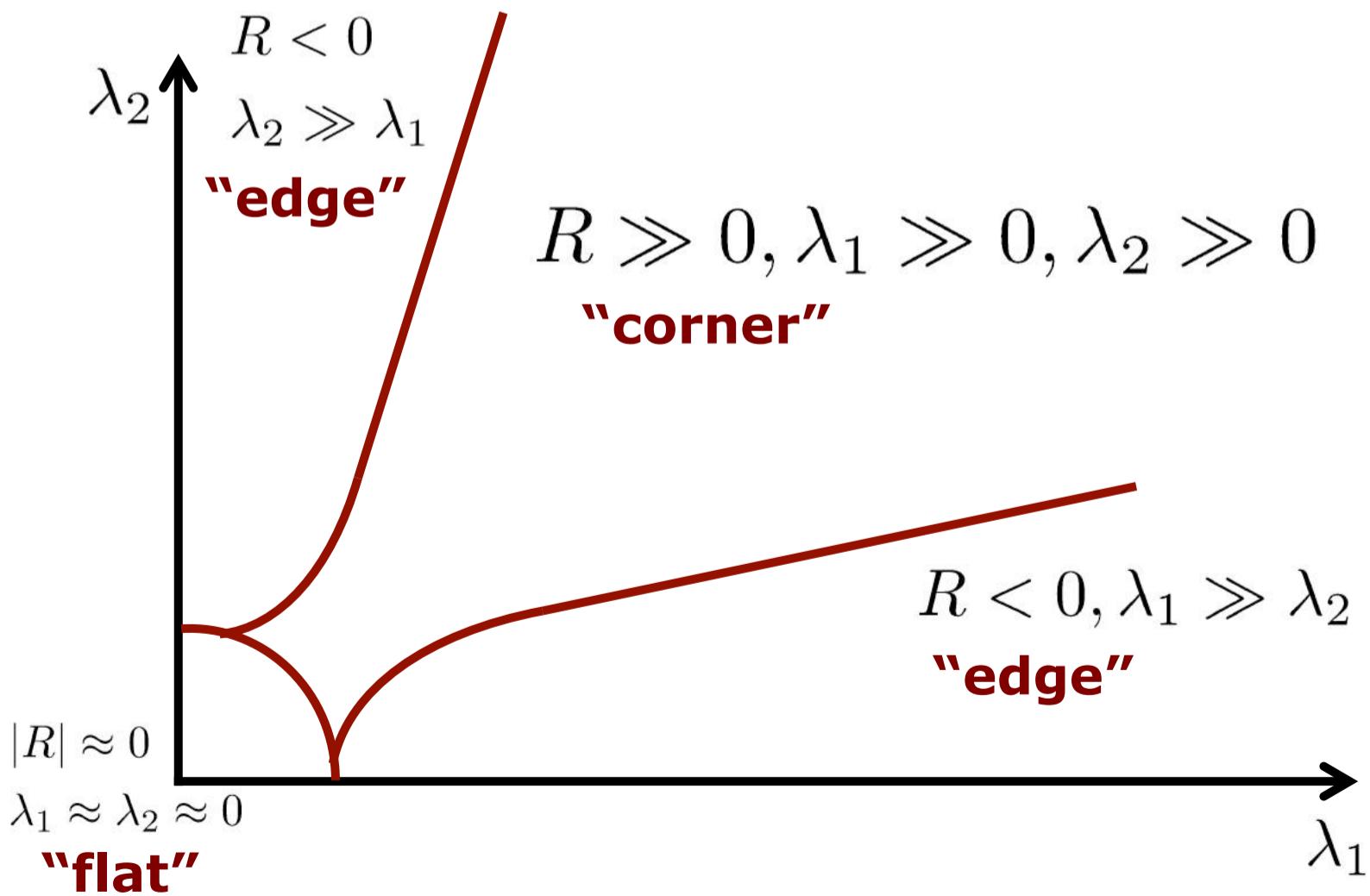
$$0 < k < 0.25.$$

Harris corner criterion

$$|R| \approx 0 \Rightarrow \lambda_1 \approx \lambda_2 \approx 0 : \mathbf{flat \ region}$$

$$R < 0 \Rightarrow \lambda_1 \gg \lambda_2 \text{ or } \lambda_2 \gg \lambda_1 : \mathbf{edge}$$

$$R \gg 0 \Rightarrow \lambda_1 \approx \lambda_2 \gg 0 : \mathbf{corner}$$





600X600 image. (b) Result of applying the HS corner detector with $k = 0.04$ and $T = 0.01$ (default) shows numerous irrelevant corners were detected. (c) Result using $k = 0.249$ and the default value for T . (d) Result using $k = 0.17$ and $T = 0.05$. (e) Result using the default value for k and $T = 0.05$. (f) Result using the default value of k and $T = 0.07$.

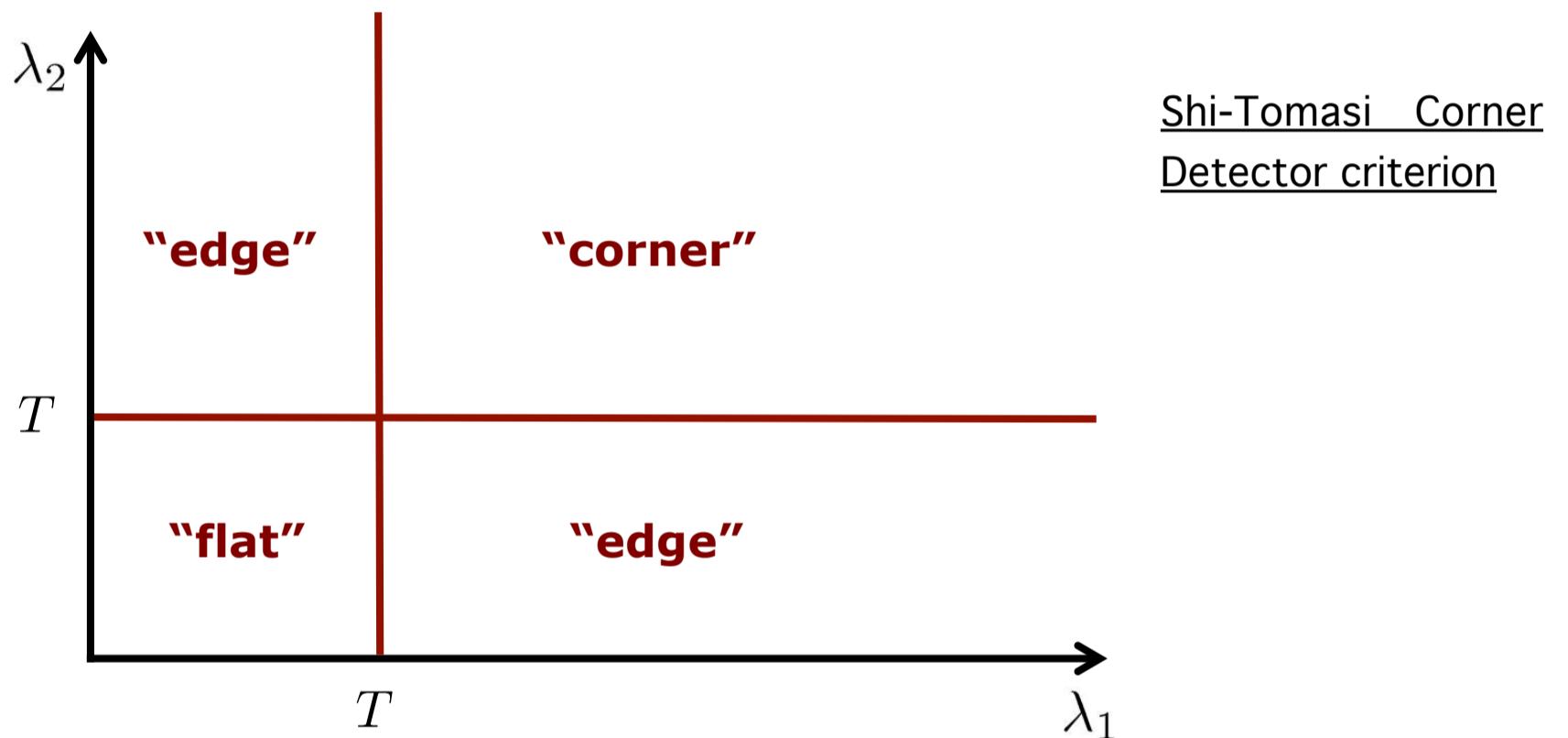


$k=0.04$ and $T=0.07$ (same parameters as last panel in the previous image) on an Image rotated 5° .

Very similar to Harris corner detector is the another popular method called the Shi-Tomasi Corner Detector that uses the same structure matrix, however,

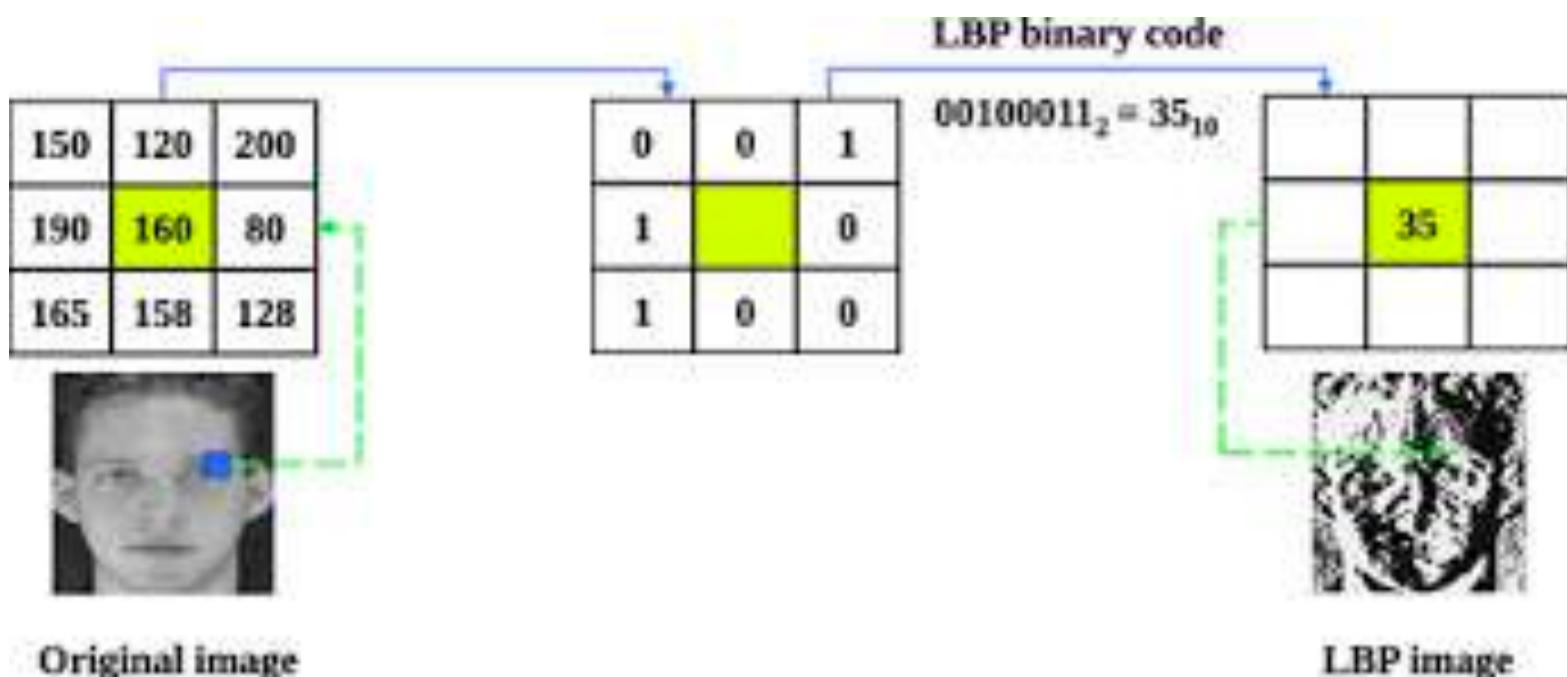
$$\lambda_{\min}(M) = \frac{\text{trace}(M)}{2} - \frac{1}{2}\sqrt{(\text{trace}(M))^2 - 4\det(M)}$$

$$\lambda_{\min}(M) \geq T \quad : \text{corner}$$



- The Shi-Tomasi corner detector is a popular method based on the same idea that was proposed by Harris-Stephan.
- Need to set only one free parameter i.e. T.

Local Binary pattern (LBP)



- Complex features such as SIFT work well, however expensive to compute
- Binary descriptors aim at generating small binary strings that are easy and fast to compute and compare and memory efficient
- Different binary descriptors differ by the strategy of selecting the order of pairs
- Define the order of pairs and fix it and use the same throughout the algorithm
- The method is sensitive to noise, hence, pass a de-noised image
- Application: facial expression detection, texture recognition

Use a smoothing filter on the input image first and for the de-noised image:

Compare values within the 3X3: neighbourhood pixel with the centre pixel.

Let P_c be centre pixel ,

P_n :be neighbourhood pixels (here 8 neighbourhood pixels)

$b=1$, if $P_n > P_c$, else $b=0$.

LBP is one of the simplest binary feature code.

- A more popular strategy is BRIEF or (Binary Robust Independent Elementary Features).
- It utilises keypoint and computes the descriptor around each keypoint.
- The neighbourhood around pixel (keypoint) is the patch, which is a square region of certain pixel width & height.
- We create a binary feature vector by using the binary test responses. We choose a set of (x,y)-location pairs ordered uniquely to define a set of binary tests within the patch.
- Once the order is set, the same order has to be used throughout the image as well as on other images (when comparing different images).
- Length of the binary feature vector could be 128, 256, and 512.

COURSE
COMPUTER VISION

TAKEN BY:

BHAVNA R, IISER-BHOPAL 2022
DSE-312

Disclaimer: Images shown in this coursework are taken from Digital image processing books or from research publications or published softwares who have the copyright. I have simply used them for the purpose of the course.

Local Feature Detectors: SIFT and HOG

Keypoint is a (locally) distinct location in an image, for example, corners.

The feature descriptor summarises the local structure around the keypoint.

The feature vector can be used for tasks such as classification, detection, and recognition.

So far, we learnt what is a keypoint (a locally distinct point within an image, eg corner) and how we can automatically spot/detect keypoint within an image.

Next question we ask is, how can we describe the local surrounding of a keypoint (or local structure around the keypoint) that will enable us to define features in a distinct manner.

The feature descriptors essentially reduce the image to selective distinct points to describe the scene. (and so not use all of the pixels within an image).

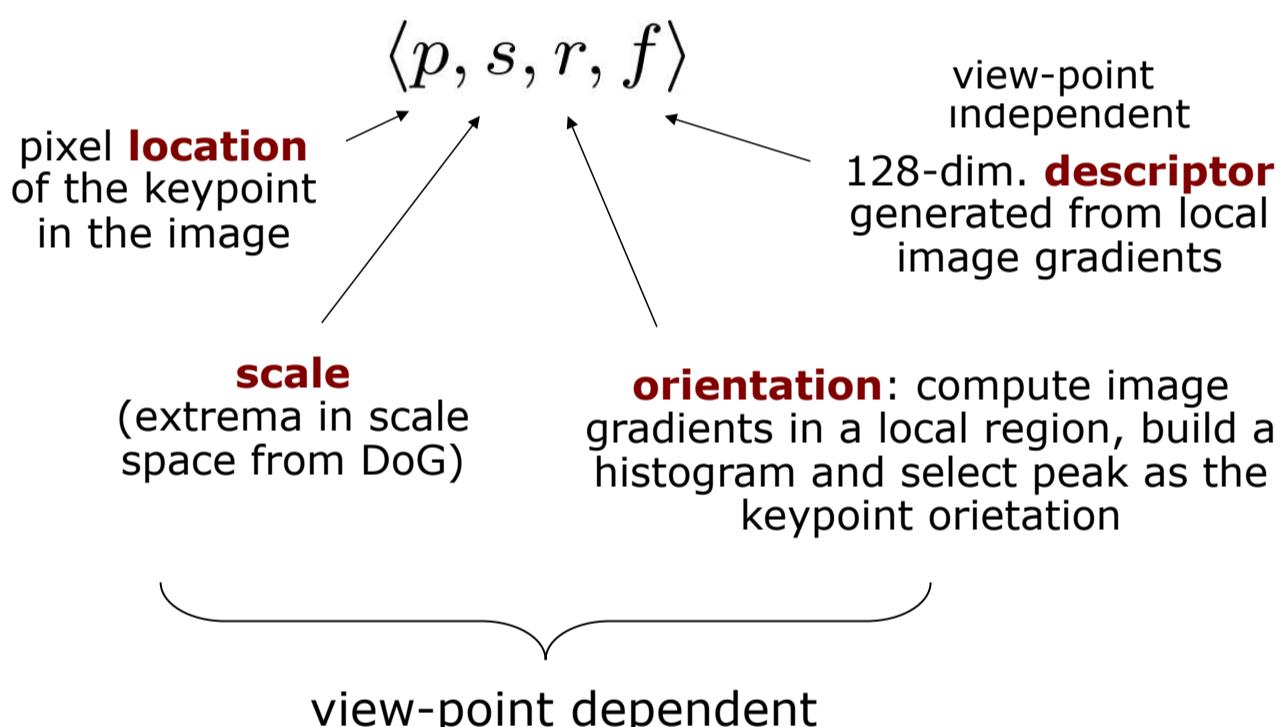
The feature vector can be used for tasks such as classification, detection, and recognition, matching/correspondences between the images taken at different viewpoints or at different time.

Scale Invariant Feature Transform (SIFT)

- SIFT is an algorithm developed by Lowe [2004] for extracting invariant features from an image.
- It is called a transform because it transforms image data into scale-invariant coordinates relative to local image features.
- SIFT features (called keypoints) are invariant to image scale and rotation, and are robust across a range of affine distortions, changes in 3-D viewpoint, noise, and changes of illumination.

- The input to SIFT is an image. Its output is an n-dimensional feature vector whose elements are the invariant feature descriptors.
- The SIFT uses a difference of Gaussian (DoG) approach (please see previous lecture for the DoG approach).
- The idea here is to use the difference of Gaussian on the same image that are differently smoothed.
- We use this DoG approach over scale space pyramid.

A SIFT feature is given by a vector computed at a local extreme point in the scale space



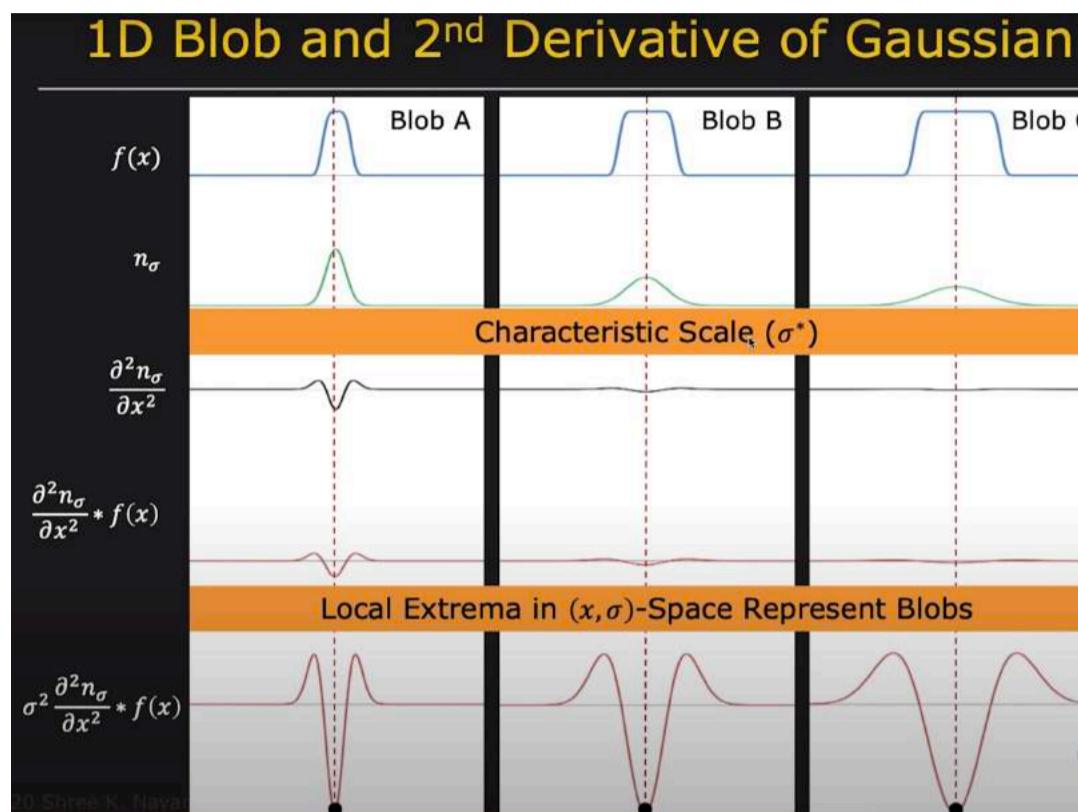
$$g_{(1)} - g_{(2)} = \text{Laplacian of Gaussian (LOG)}$$

Diagram illustrating the computation of the Laplacian of Gaussian (LOG) filter. It shows two Gaussian filters, $g_{(1)}$ and $g_{(2)}$, being subtracted to produce a result that approximates the LOG filter.

$$f * g_{(1)} - f * g_{(2)} = f * (g_{(1)} - g_{(2)})$$

Diagram illustrating the computation of the Laplacian of Gaussian (LOG) filter applied to an image. It shows the original image, the result of applying two different Gaussian filters, and the result of applying the LOG filter, which highlights edges and corners.

Taking difference of Gaussian approximates the LoG detector. While approximating the LoG, there is no actual derivative computation needed. 1D plot to understand the relation between scale and local extrema.



A Gaussian pyramid of images running from 512x512 to 8x8.

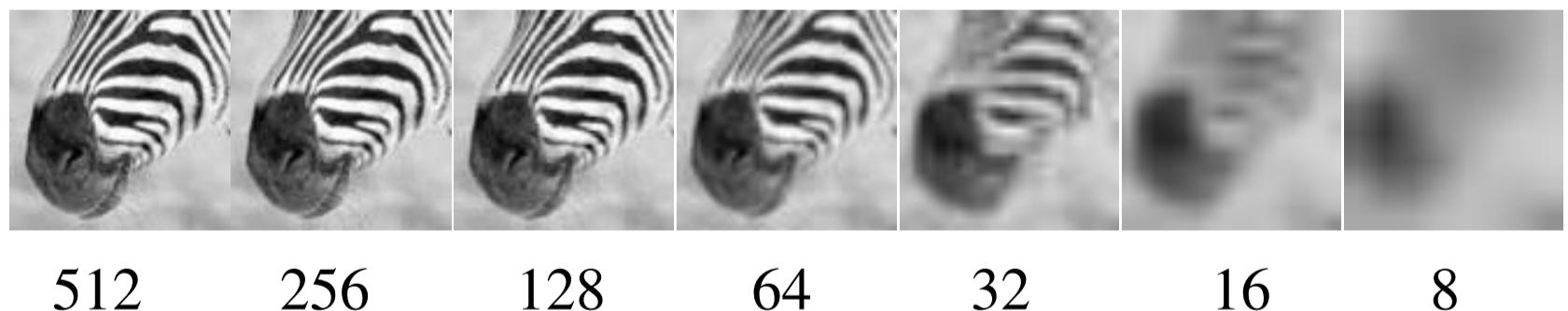
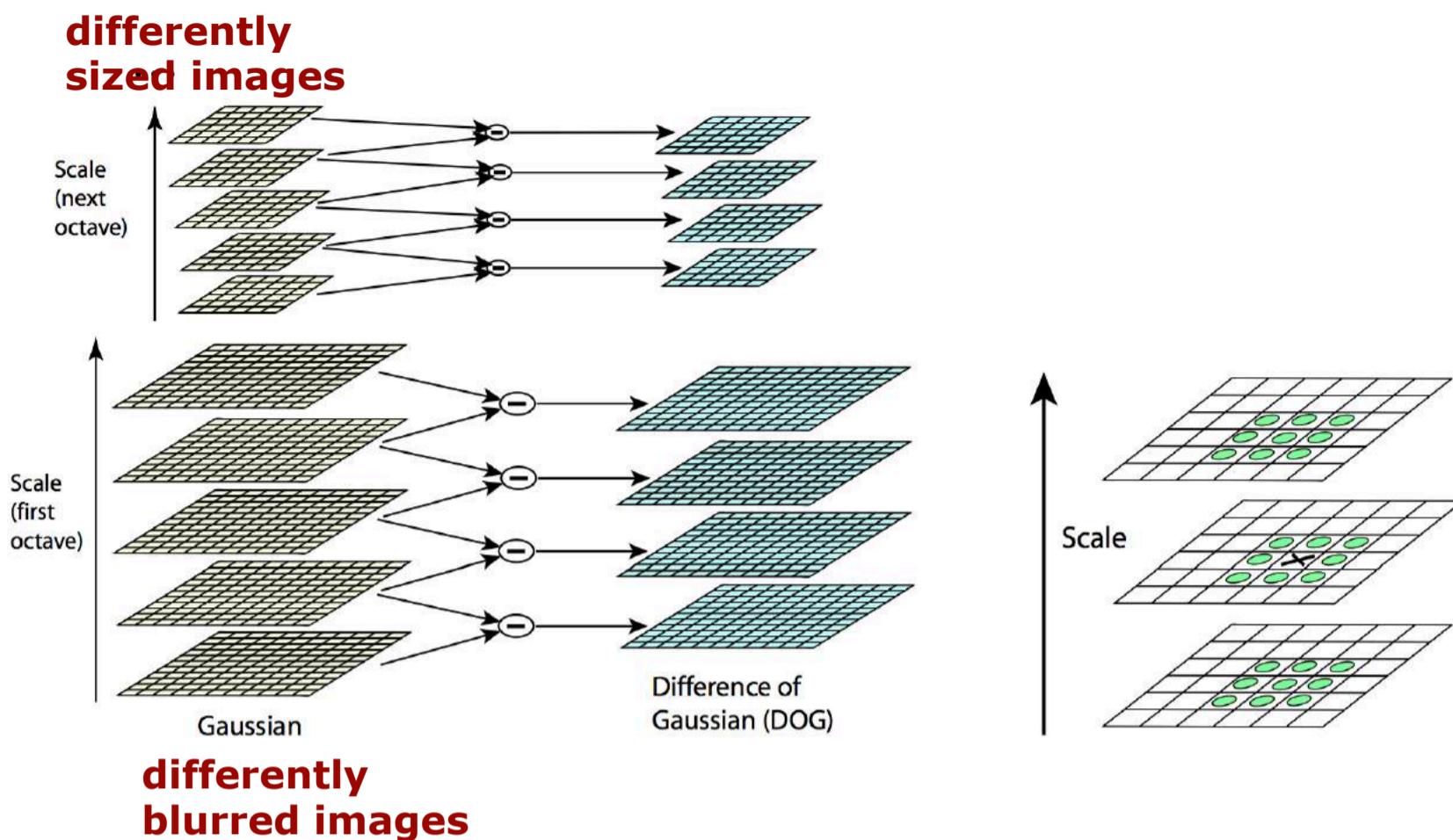


Illustration of the Difference-of-Gaussians over different image pyramid levels



Finding extrema in scale space from DoG results:

One pixel in an image is compared with its 8 neighbours as well as 9 pixels in the next scale and 9 pixels in previous scales. This way, a total of 26 checks are made. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale.

Keypoint localisation

The output of DoG filter produce a lot of keypoints including edges and low contrast features, these are not unique features.

For low contrast features, we filter out using their intensities. In the original paper, a Taylor series expansion of scale space is generated to get a more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected.

We use a similar approach to that of Harris Corner Detector for removing edge features. Firstly, compute a 2×2 Hessian matrix (H).

- Reject flats:

- $|D(\hat{x})| < 0.03$

- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let $r = \alpha/\beta$.
So $\alpha = r\beta$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

- $r < 10$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

Keypoint orientation

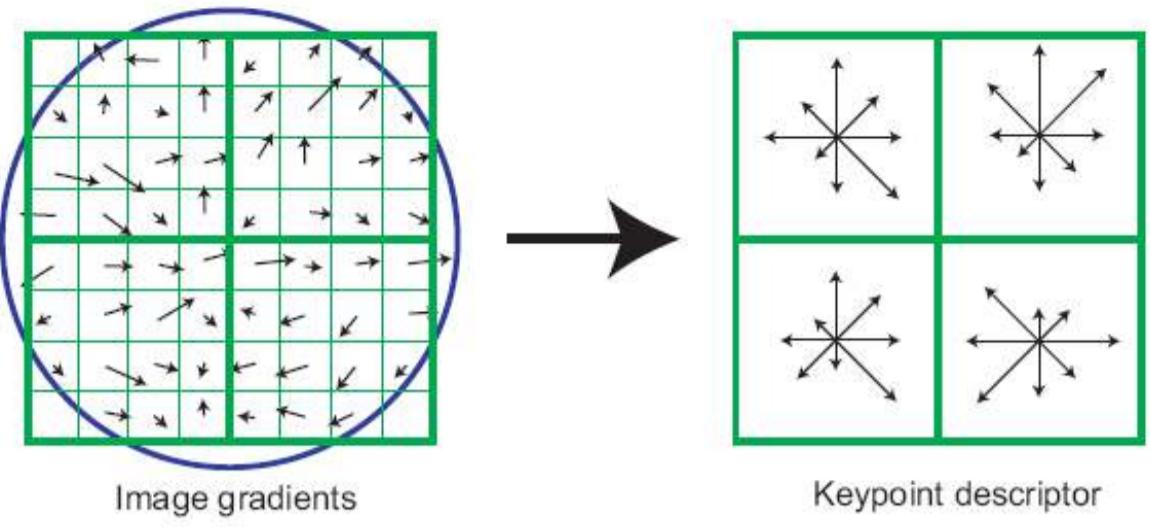
The detected keypoints are scale invariant, next we want to identify the orientation to each keypoint to make it rotation invariance.

- A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region.
- The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation.

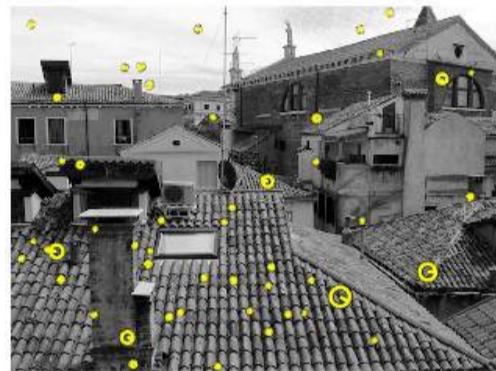
How is the f (128-D descriptor) computed?

- Compute image gradients in local 16x16 area at the selected scale
- Create an array of orientation histograms
- 8 orientations x 4x4 histogram array = 128 dimensions (yields best results)

Example using 8x8 area:



input

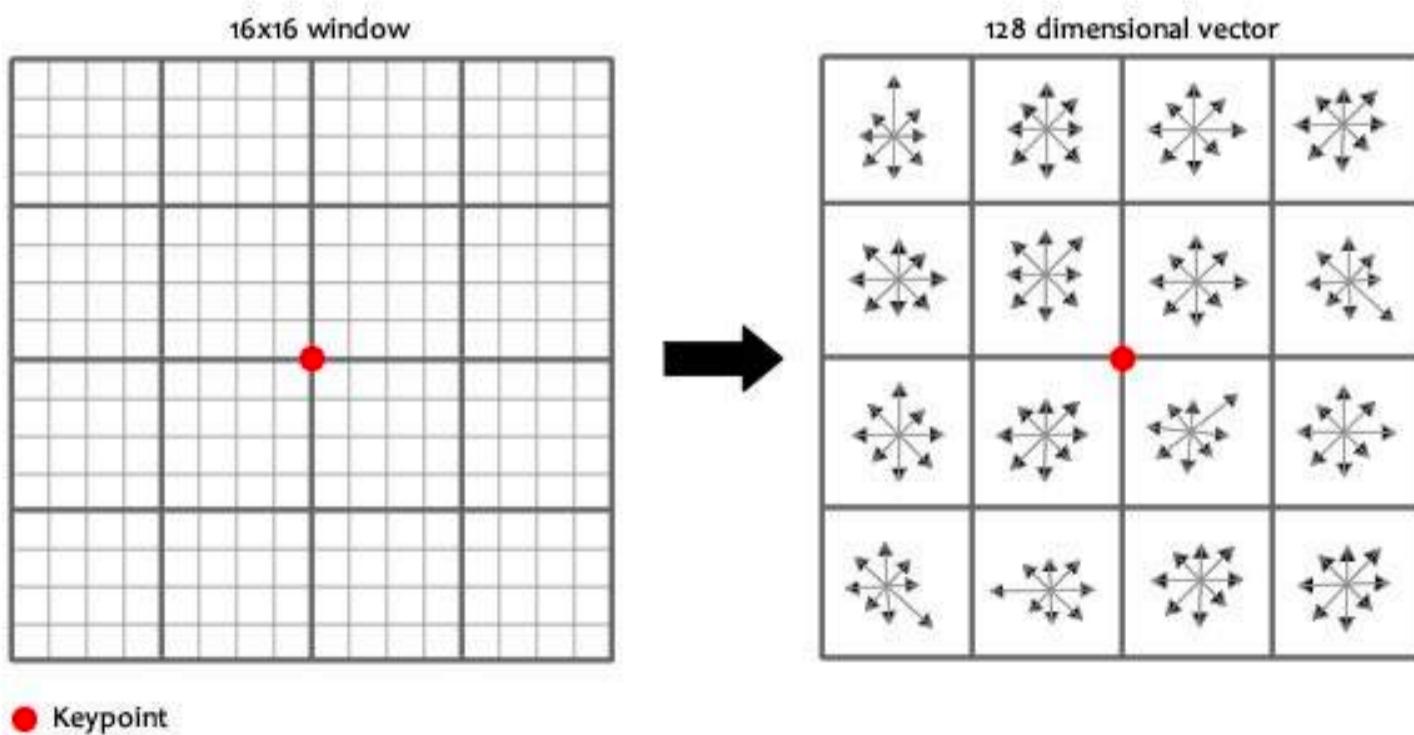


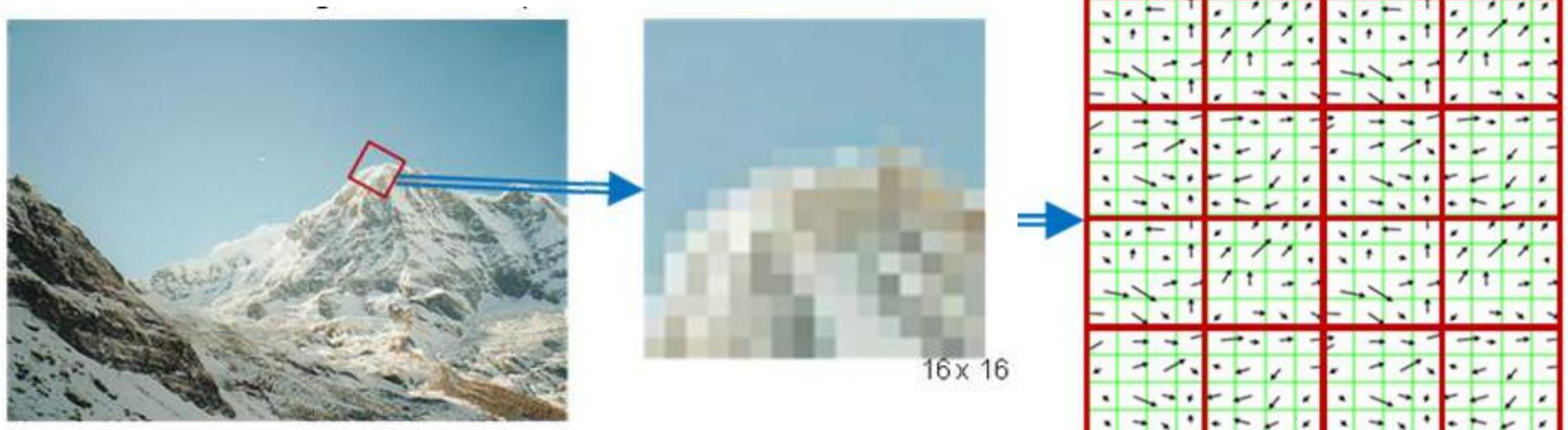
key points



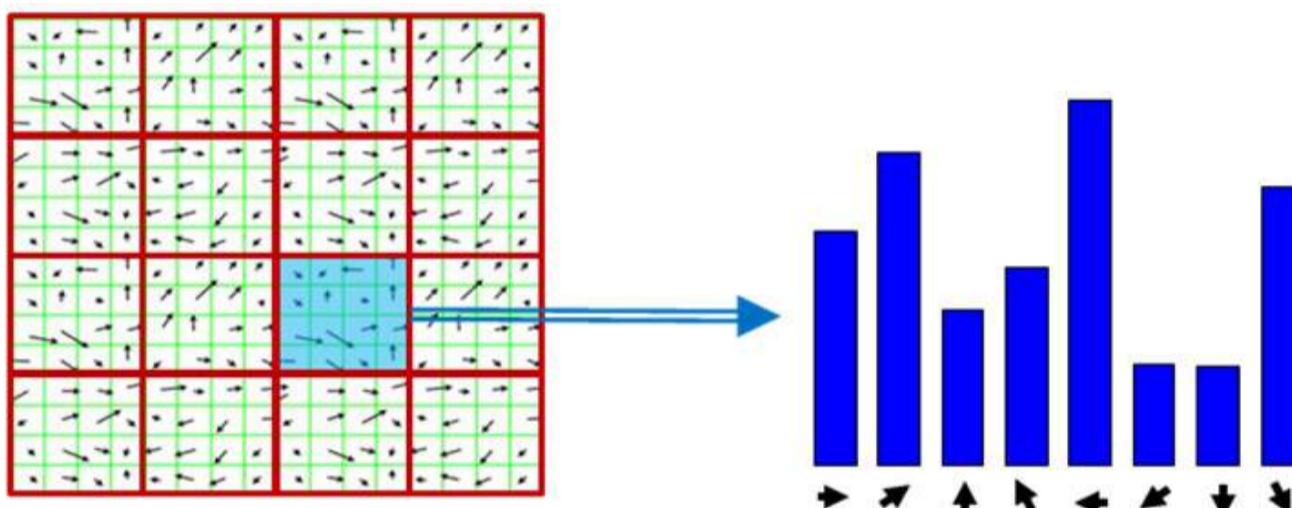
descriptor (regions)

For a given keypoint, warp the region around it to orientation and scale and resize the region to 16X16 pixels. Compute the gradients for each pixels (orientation and magnitude) and divide the pixels into 16, 4X4 pixels squares

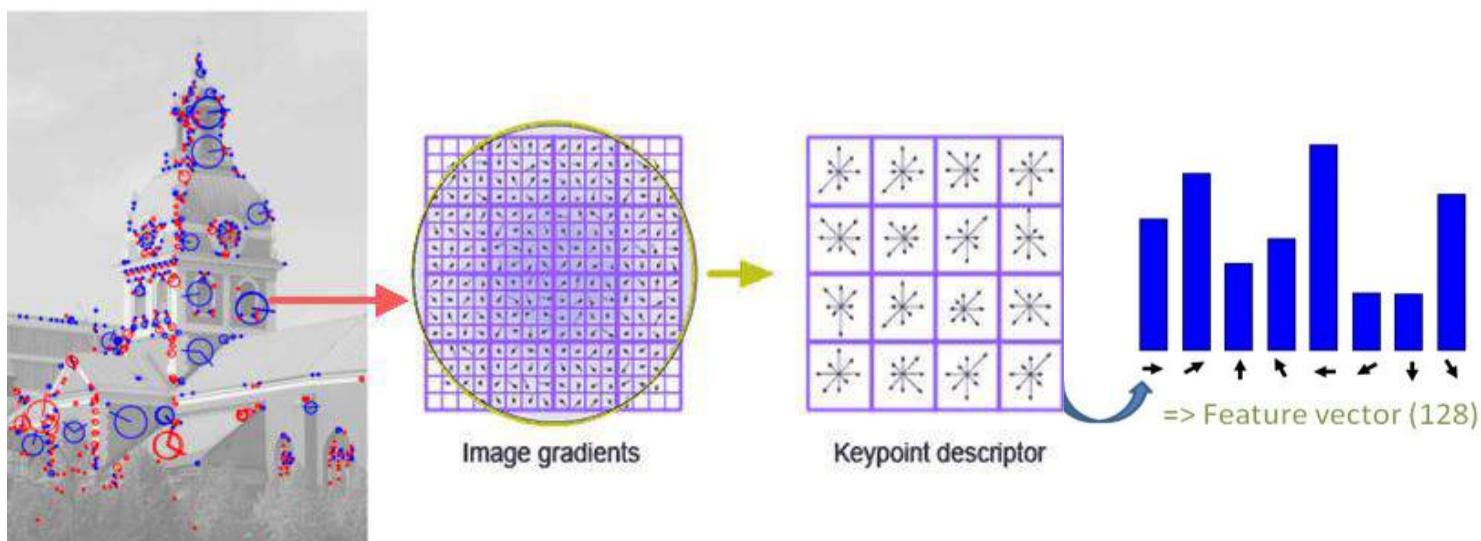




For each square, compute gradient direction histogram over 8 directions



Concatenate the histograms to obtain a 128 (16×8) dimensional feature vector:



The feature vector uses gradient orientations and so when the image is rotated, the gradient orientations also change.

To achieve rotation independence, the keypoint orientation is subtracted from each localised orientation. Thus each gradient orientation is relative to the keypoint's orientation. So, we have a rotation independent feature vector.

Histogram of Oriented Gradient (HOG) features:

The HoG is an edge based descriptor that focuses on the structure or the shape of an object by generating histograms using both the magnitude and orientations of the gradient.

Steps to calculate HOG Features:

1. Resize the input images 128x64 pixels (128 pixels height and 64 width) (originally suggested by authors in the paper, to achieve the task of pedestrian detection).
2. Compute image gradients using both magnitude and orientation from the image. Take a 3x3 sub-image and compute G_x and G_y for each pixel and then the orientation at each pixel.

$$G_x(r, c) = I(r, c + 1) - I(r, c - 1) \quad G_y(r, c) = I(r - 1, c) - I(r + 1, c)$$

(r, c refer to rows and columns)

$$\text{Magnitude}(\mu) = \sqrt{G_x^2 + G_y^2} \quad \text{Angle}(\theta) = |\tan^{-1}(G_y/G_x)|$$

The gradient is then transformed to polar coordinates, with the angle constrained to be between 0 and 180°, so that gradients that point in opposite directions are identified.

3. Cell Orientation Histograms:

- Form a block 8x8 cells. These are non-overlapping cells of size 8x8 pixels.

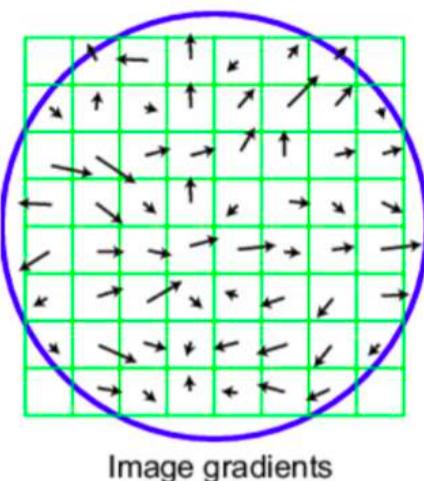
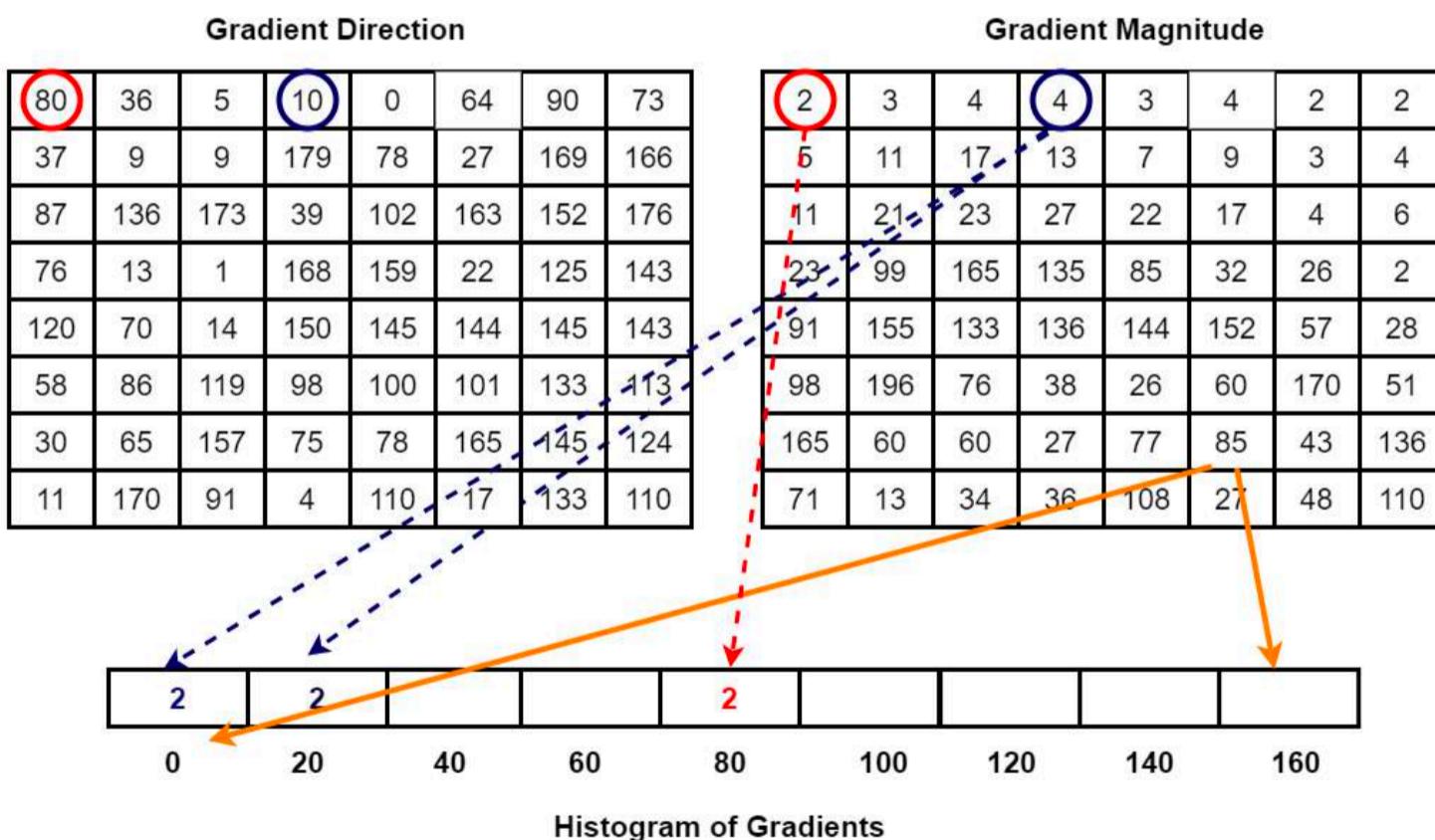


Image gradients

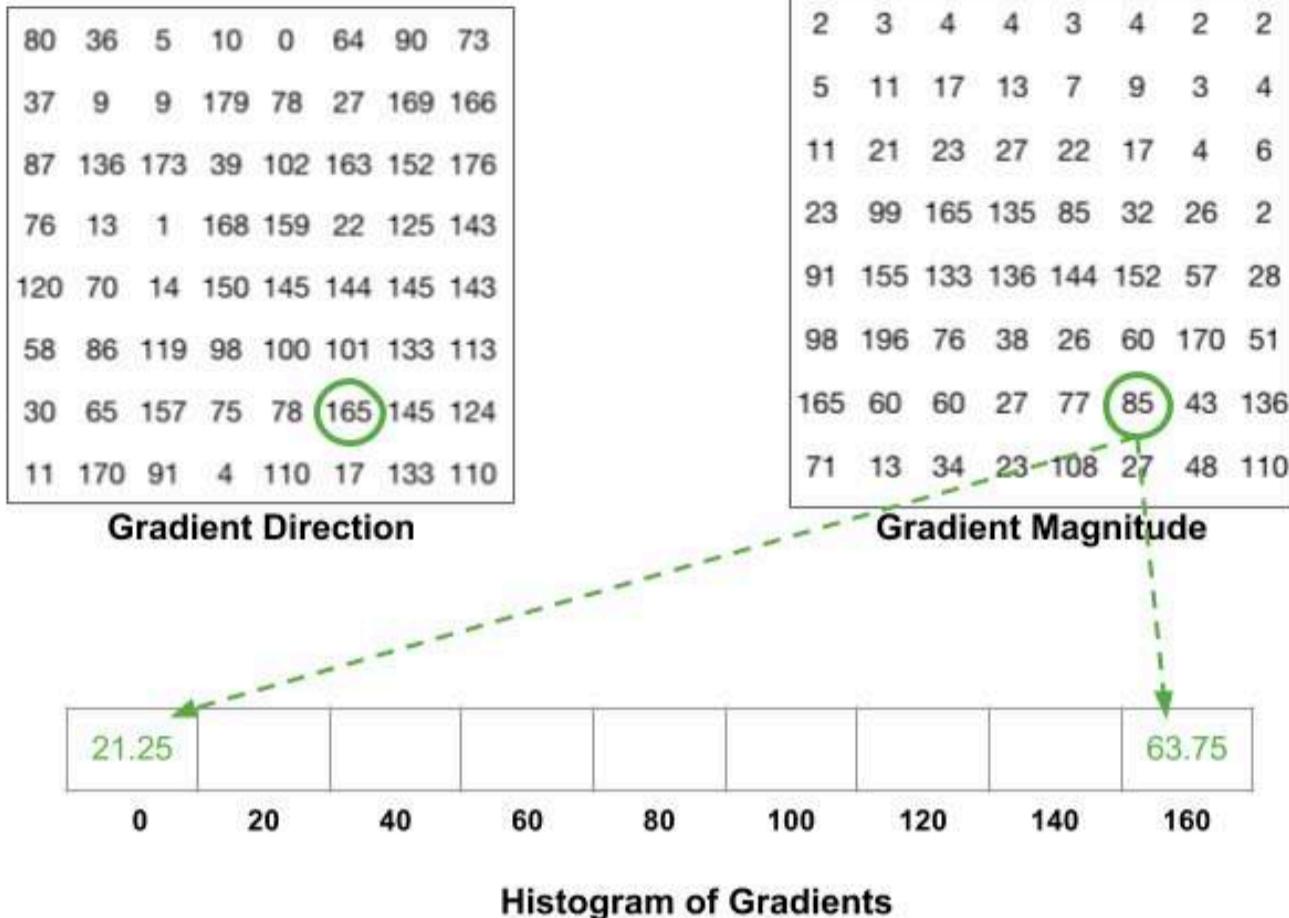
- In each cell, compute a histogram of the gradient orientations binned into 9 bins such that each bin has an angle range of 20° . Each of these 9-point histograms can be plotted as histograms with bins outputting the intensity of the gradient in that bin. As a block contains 64 different values, for all 64 values of magnitude and gradient, the binned values are computed.

$$\text{Number of bins} = 9 (\text{ranging from } 0^\circ \text{ to } 180^\circ)$$

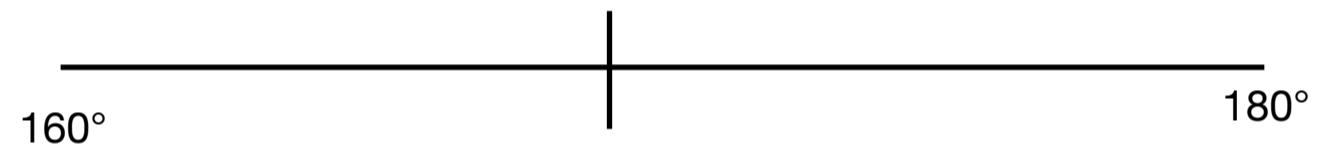
$$\text{Step size}(\Delta\theta) = 180^\circ / \text{Number of bins} = 20^\circ$$



If the angle between 160° and 180° , the angle wraps around making 0° and 180° equivalent. So 165° contributes proportionally to the 0° bin and the 160° degree bin.



The orientation of each pair corresponds to the center of a histogram bin. An orientation of zero degrees (horizontal gradient) corresponds to a vertical edge, so the slice pairs for 10 and 170 degrees are close to vertical.

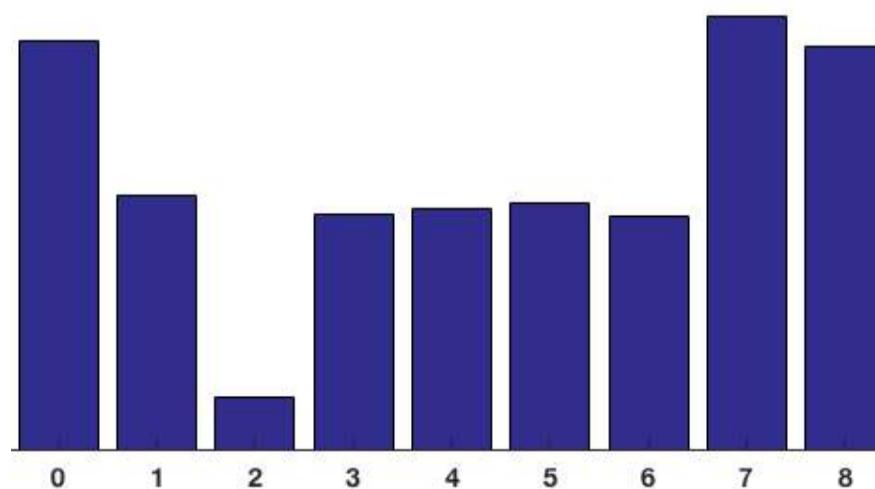


$$(\text{abs}(180-165)/20) * 85 = 63.75$$

$$(\text{abs}(160-165)/20) * 85 = 21.25$$

$$HoG \text{ feature vector} = \text{Abs}\left(\frac{(B \text{ in vector value} - \text{gradient direction})}{20(\text{bin size})}\right) * \text{gradient magnitude}$$

The contributions of all the pixels in the 8x8 cells are added up to create the 9-bin histogram. For a single patch, histogram is obtained:



The assignment of gradients into respective bins can be automatically computed :

Each Jth bin, bin will have boundaries from : $[\Delta\theta \cdot j, \Delta\theta \cdot (j + 1)]$

Value of the centre of each bin will be : $C_j = \Delta\theta(j + 0.5)$

For each cell in a block, we will first calculate the jth bin and then the value that will be provided to the jth and (j+1)th bin respectively.

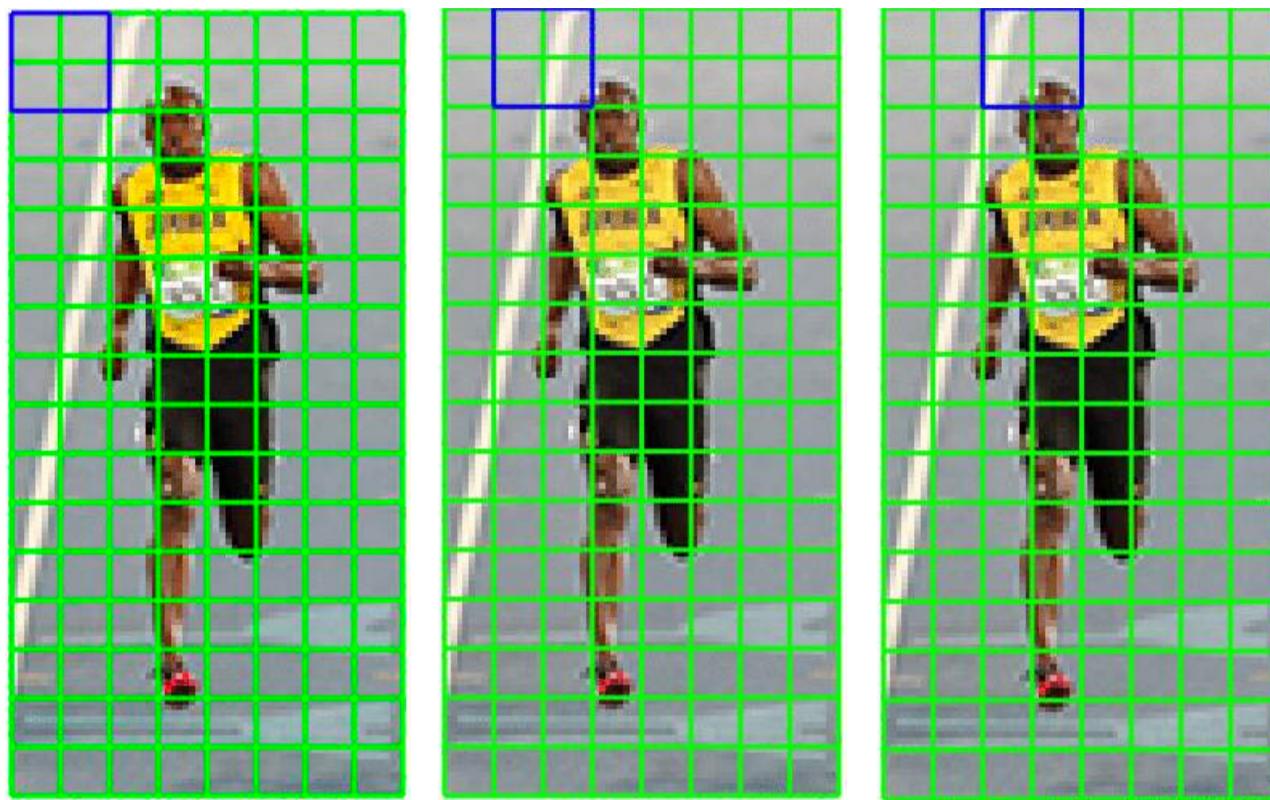
$$\begin{aligned} j &= \lfloor \left(\frac{\theta}{\Delta\theta} - \frac{1}{2} \right) \rfloor \\ V_j &= \mu \cdot \left[\frac{\theta}{\Delta\theta} - \frac{1}{2} \right] \\ V_{j+1} &= \mu \cdot \left[\frac{\theta - C_j}{\Delta\theta} \right] \end{aligned}$$

An array is taken as a bin for a block and values of V_j and V_{j+1} is appended in the array at the index of jth and (j+1)th bin calculated for each pixel. For all 4 cells in a block, we concatenate all the 9 point histograms for each constituent cell to form a 36 feature vector.

16x16 Block Normalisation:

There are 7 horizontal and 15 vertical positions making a total of $7 \times 15 = 105$ positions. Each 16x16 block is represented by a 36x1 vector. So when we concatenate them all into one vector we obtain a $36 \times 105 = 3780$ dimensional vector.

Blue box indicates 3 consecutive horizontal positions



The 1-D 36 features vector:

Values of f_b for each block is $f_{bi} = [b_1, b_2, b_3, \dots, b_{36}]$ normalized by the L2 norm :

$$f_{bi} \leftarrow \frac{f_{bi}}{\sqrt{\|f_{bi}\|^2 + \varepsilon}}$$

To normalize, the value of k is first calculated using :

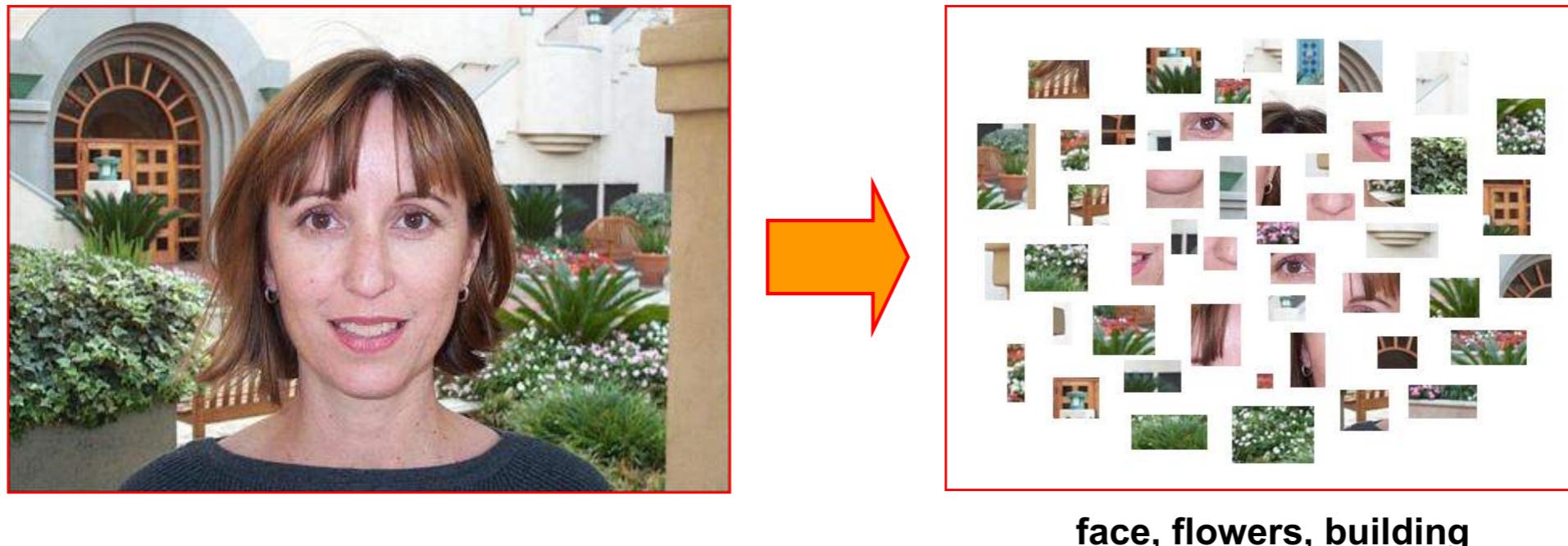
$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + \dots + b_{36}^2}$$
$$f_{bi} = \left[\left(\frac{b_1}{k} \right), \left(\frac{b_2}{k} \right), \left(\frac{b_3}{k} \right), \dots, \left(\frac{b_{36}}{k} \right) \right]$$

Bag of Visual words
Lecture 25-CV- DSE312

Bhavna R

Bag of visual words tasks

Bags of features for object recognition

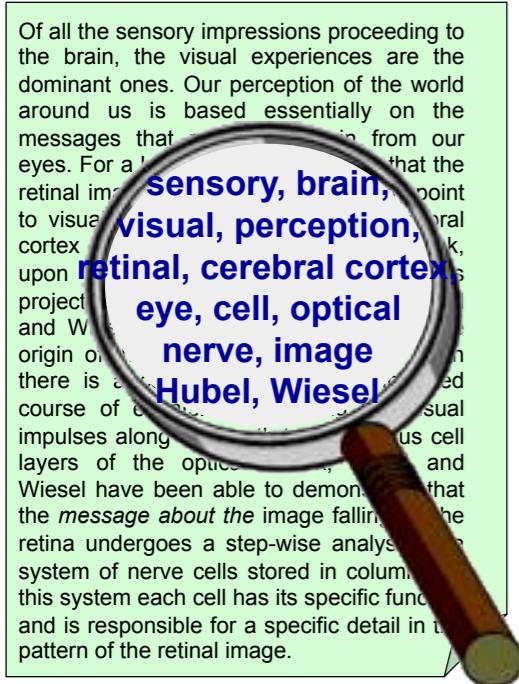


Finding images in a database, which are similar to a given query image



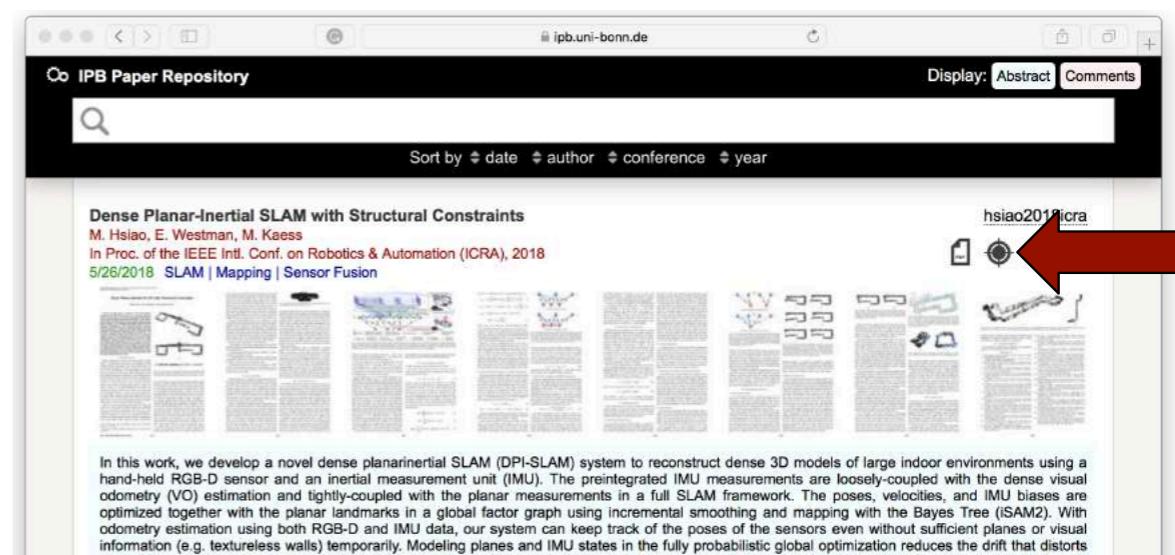
Scene or context recognition

An extension to the NLP algorithm, Bag of Words used for image classification



[Image courtesy: Fei-Fei Li]

Analogy to text mining: reduces the documents to word occurrences and identifies a distribution (returns frequently occurring words)



[Image Courtesy: Cyrill Stachniss]

Finding similar papers by counting the occurrences of certain words and returns papers based on similarity

image of an object(s)

bag of 'words'

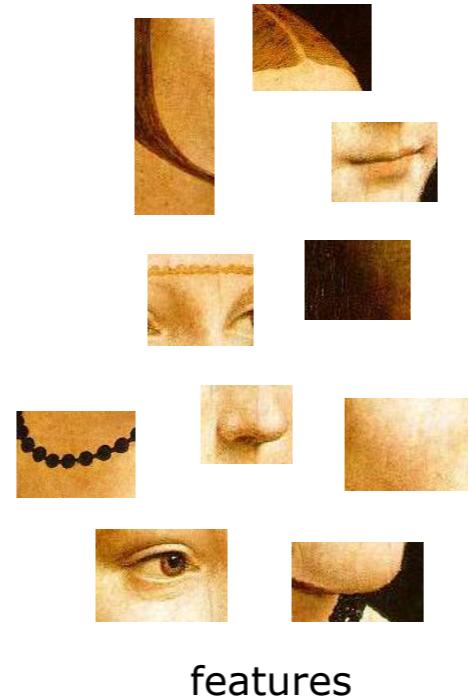


The contents are inferred from the frequency of relevant words that occur

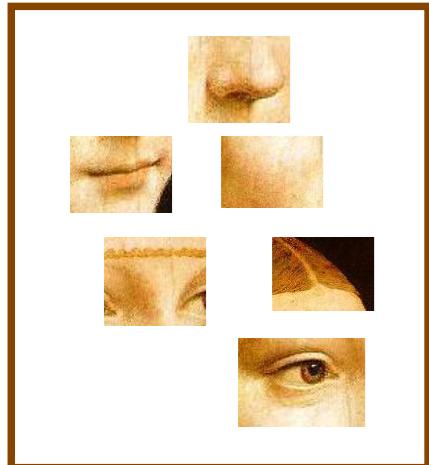
Image to localized image patches (or sub-images) to infer 'bag of visual words' (using features, eg: SIFT)

Overview: BoVW

- First, take a bunch of images, extract features, and build up a “dictionary” or “visual vocabulary” – a list of common features
- Given a new image, extract features and build a histogram – for each feature, find the closest visual word in the dictionary
- Use only words from the dictionary
- Represent the images based on a histogram of word occurrences
- Compact representation of images using histogram of words
- Image comparisons are performed based on such word histograms



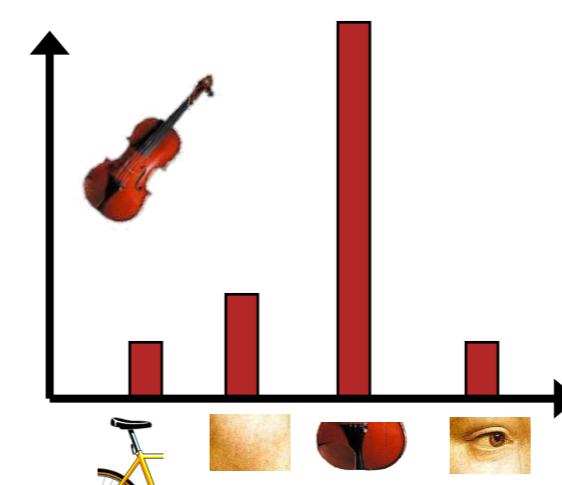
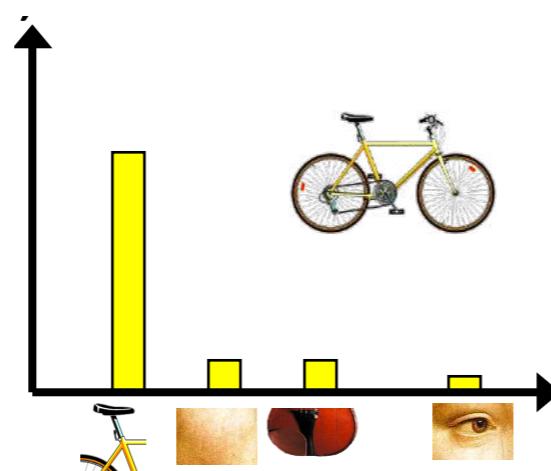
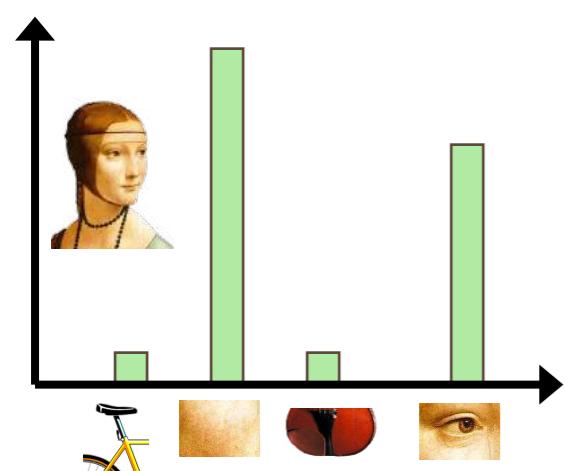
Step1: Extract features



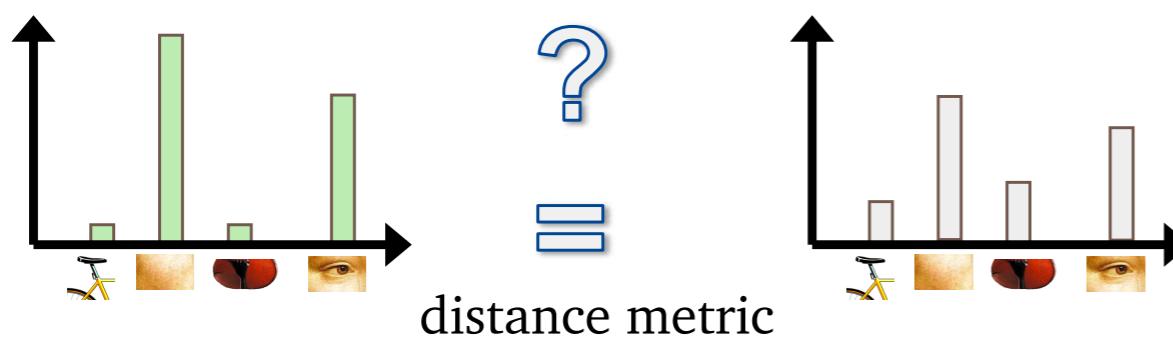
Selection of features depends on the application:
Interest point detectors, corner points, edges, blobs, DOG, Scale Invariant Feature Transform (SIFT)

Step2: Learn “visual vocabulary” by constructing a dictionary of representative words that forms x- axis of the histogram (Use only words from the dictionary)

Step3: Represent images by frequencies of “visual words” (quantise features) y-axis of the histogram

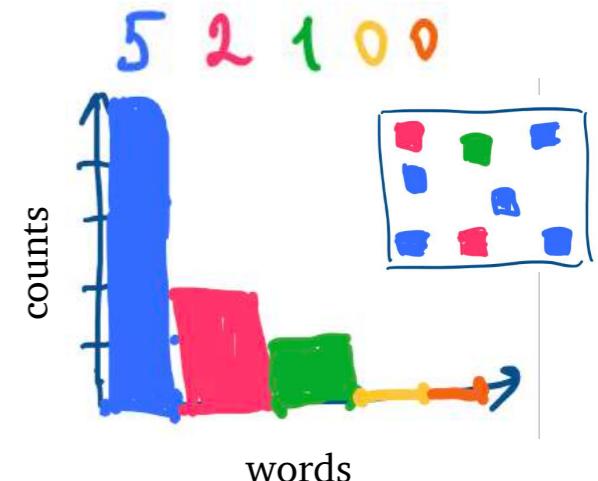
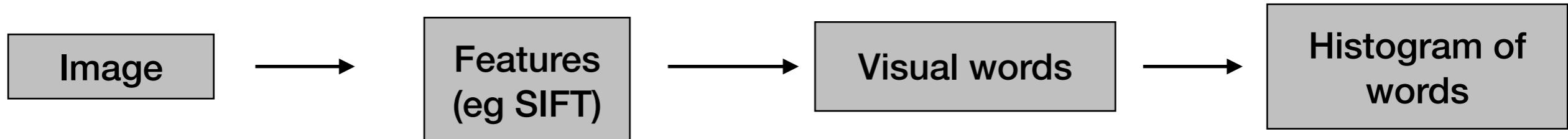


Step4: Image comparisons: performed based on such word histograms by comparing the distribution of words



We dont need the images anymore at this step!

From images to histogram of words



Not all features will have visual words!

[Slide images: Cyril Stachniss]

What about the ‘Dictionary’?

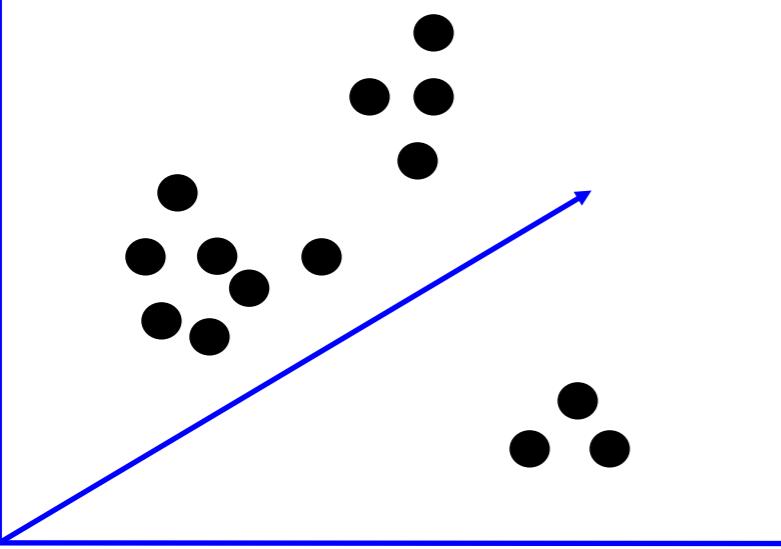
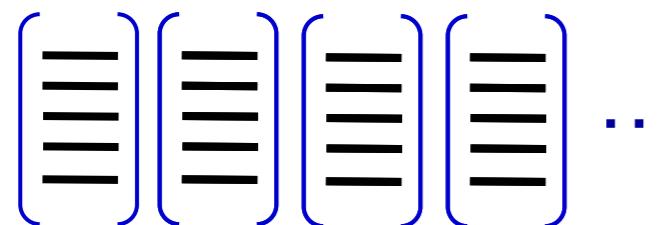
- A dictionary defines the list of words that are considered (this is built already with a separate set)
- The dictionary defines the x-axis of all the word occurrence histograms that are allowed
- (eg; words describing scene)
- The dictionary must remain fixed
- Learned from data

Learning the dictionary (visual vocabulary) from the data

1. Training dataset: a large diverse collection of images, different views

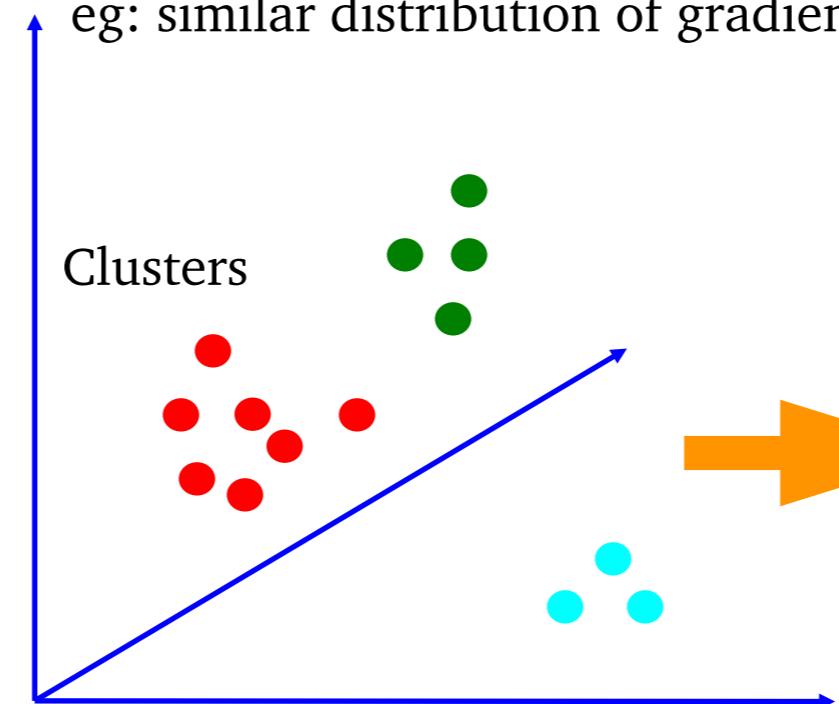


2. Visual feature descriptor vectors (e.g., SIFT)



3. Each set of feature descriptors are points in a high-dimensional space

4. Group similar descriptors
eg: similar distribution of gradients



5. **Visual vocabulary**

Turn cluster centroids to words

Clustering

Unsupervised approach: K-means

Training set can also be designed for specific scene recognition tasks

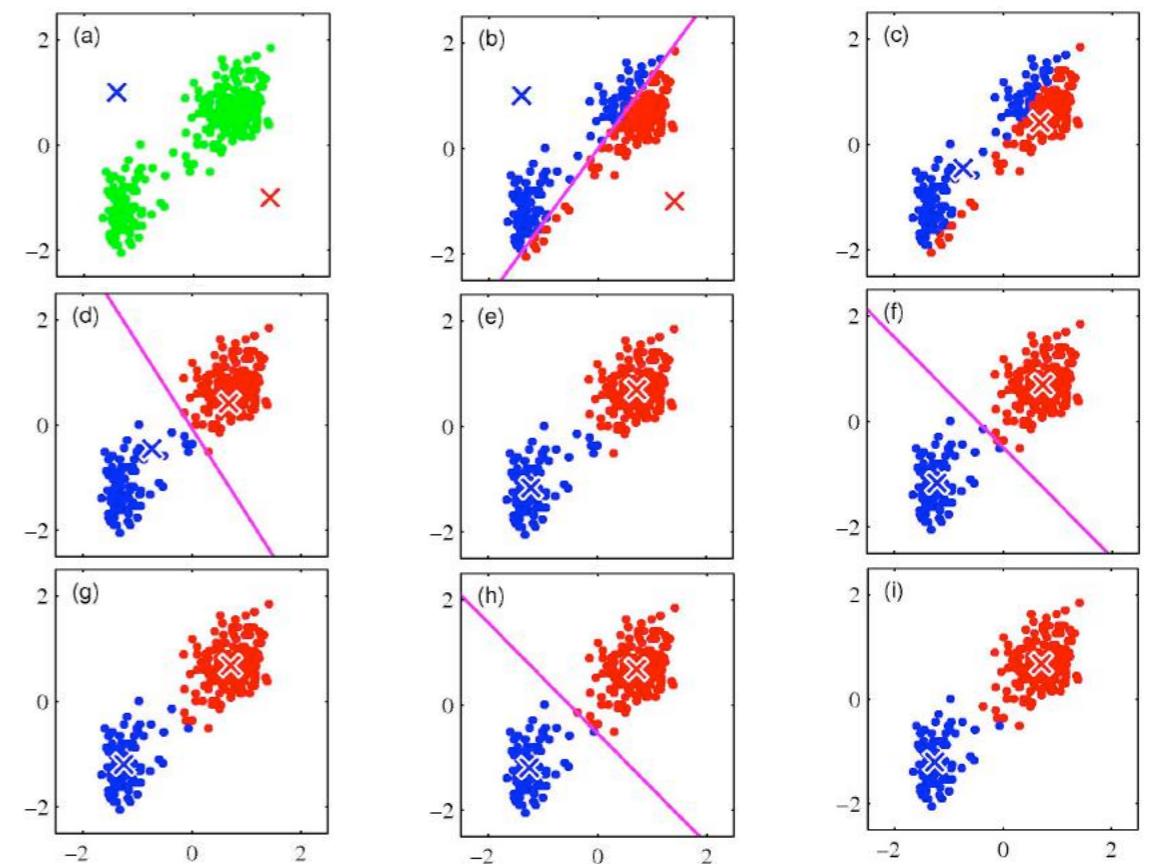
Recap: K-means clustering

Objective: Find the k cluster centres and assign the data points to the nearest one, such that sum of squared Euclidean distances between points x_i and their nearest cluster centroids m_k are minimised:

$$D(X, M) = \sum_{\text{cluster } k} \sum_{\text{point } i \text{ in cluster } k} (x_i - m_k)^2$$

K-means example: 2-d feature space
the centroids positions are adjusted for all datapoint until convergence (eg: until distances remain same)

- Randomly initialise K cluster centres
Partitions the data into k clusters
Clusters are represented by centroids
- A centroid is the mean of data points
- Iterate until convergence:
- Assign each data point to the nearest centroid
- Recompute each cluster center as the mean of all points assigned to it



[Image courtesy: Bishop]

Building dictionary: From k-means clusters to visual words

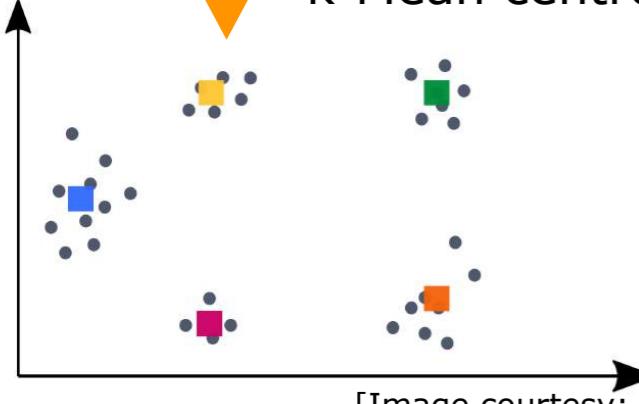
Training Data



Building dictionary from a database of images:

- Extract features from the database images for quantizing features
- Learn a vocabulary using k- means (typical k: 100,000)
- Compute *weights* for each word
- Create an inverted file mapping words to images
- Codebook = visual vocabulary (dictionary)
- training set needs to be sufficiently representative, for a “universal” codebook

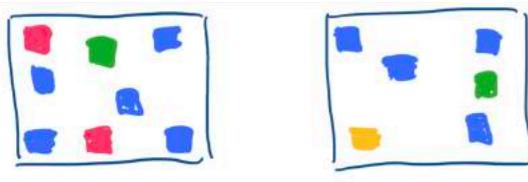
k-Mean centroids



Each cluster centroid produced by k-means becomes a code vector = visual words

[Image courtesy: Olga Vysotska]

Images represented by visual word of occurrences: a compact representation of image content



Dictionary size

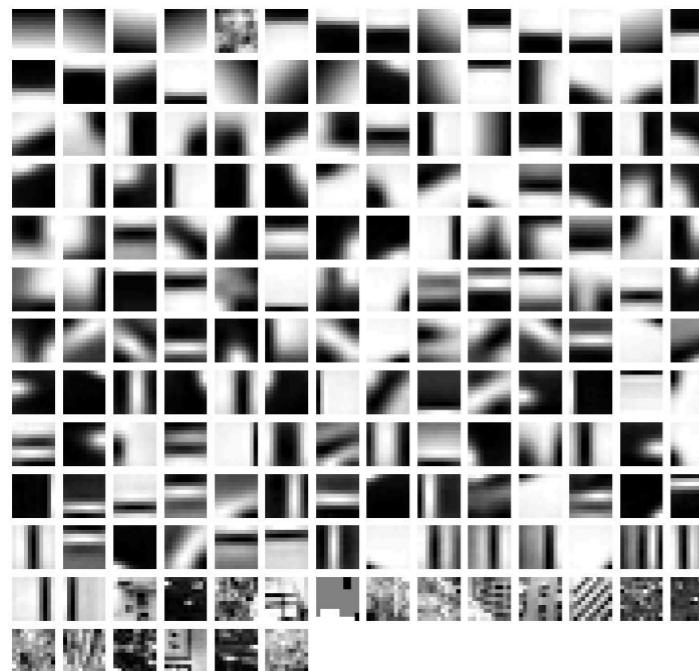
Too small: visual words not representative of all patches

Too large: quantisation artefacts, overfitting

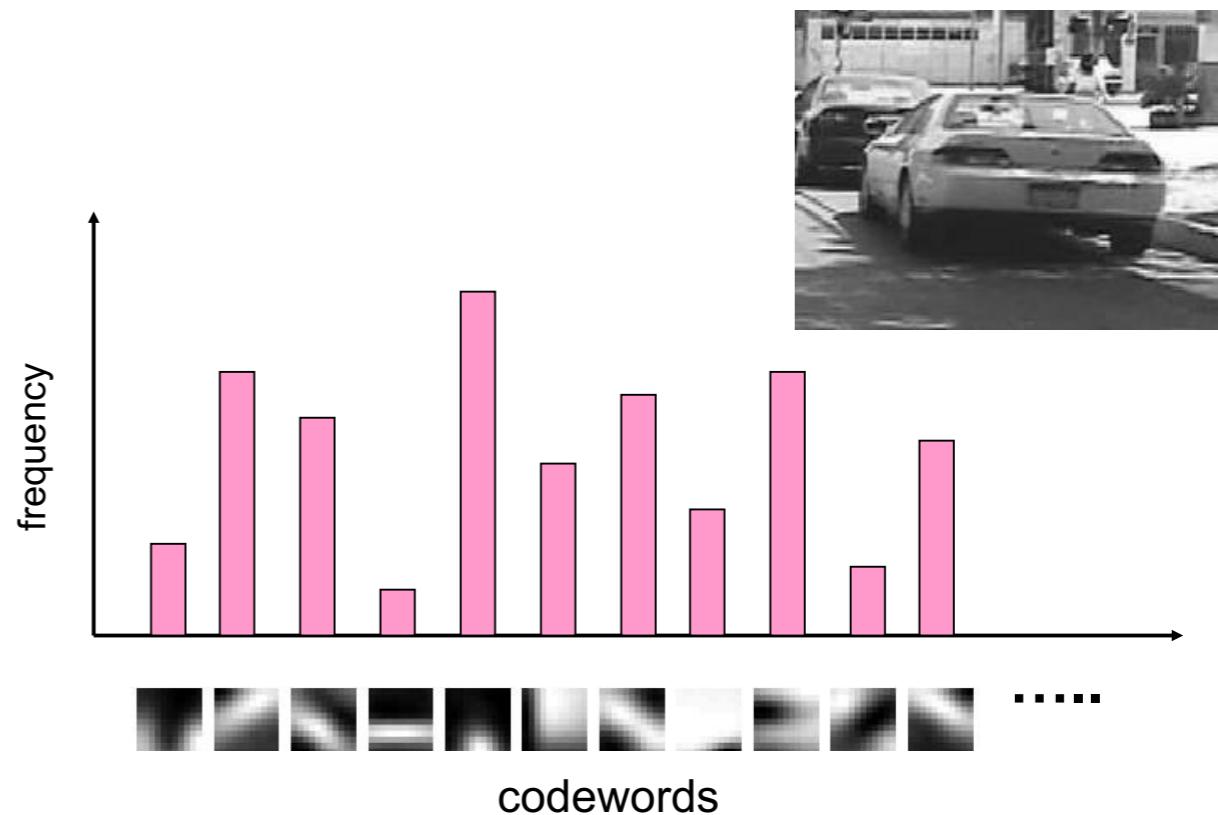
BowW model: Spatial arrangement of visual words not taken, invariant to changes in viewpoints, deformations

How do we find similar image within a database using the dictionary?

- For a new image, map a feature vector to the index of the nearest codevector (or visual word) in a codebook (dictionary)
- The dictionary defines the x-axes of all the word occurrence histograms



Example visual vocabulary



Weighting the words: TF-IDF

Words that occur in every image do not help a lot for comparisons

- some visual words are more discriminative than others
- the bigger fraction of the documents a word appears in, the less useful it is for matching
 - a word that appears in *all* documents is not helping us (*the, and, or*)
 - less frequently occurring words might be more useful : eg: *cow, AT&T,*

TF-IDF:term frequency – inverse document frequency

- Weight words considering the probability that they appear
- Instead of computing a regular histogram distance, we'll weight each word by its *inverse document frequency*

$$t_{id} = \frac{n_{id}}{n_d} \log \frac{N}{n_i}$$

term frequency **inverse document frequency**

bin of word i in image d **n_{id}** **n_d** **N** **n_i**

First term: histogram normalised to 1
Second term:Inverse probability that a word occurs in a database

t_{id} : histogram bin of word i for image d

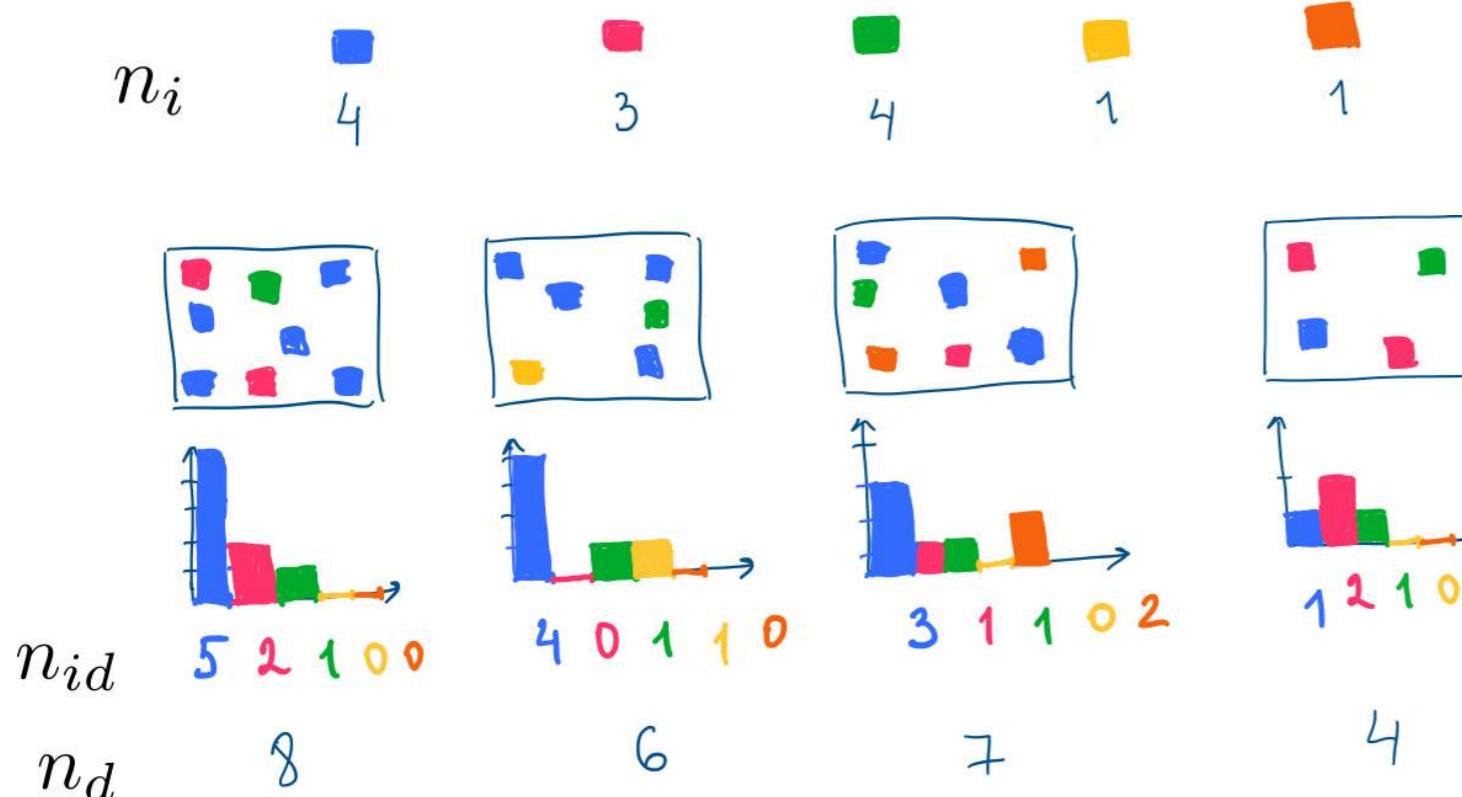
n_{id} : occurrences of word i in image d

n_d : number of word occurrences in image d

n_i : number of images that contain word i

N : number of images

Computing TF-IDF



histogram of occurrences

	n_i	4	3	4	1	1		
	n_{id}	8	6	7	4			
	n_d	8	6	7	4			
■ t_1	$\frac{5}{8} \log \frac{4}{4}$	0	$\frac{4}{6} \log \frac{4}{4}$	0	$\frac{3}{7} \log \frac{4}{4}$	0	$\frac{1}{4} \log \frac{4}{4}$	0
■ t_2	$\frac{2}{8} \log \frac{4}{3}$	0.07	$\frac{0}{6} \log \frac{4}{3}$	0	$\frac{1}{7} \log \frac{4}{3}$	0.04	$\frac{2}{4} \log \frac{4}{3}$	0.14
■ t_3	$\frac{1}{8} \log \frac{4}{4}$	0	$\frac{1}{6} \log \frac{4}{4}$	0	$\frac{1}{7} \log \frac{4}{4}$	0	$\frac{1}{4} \log \frac{4}{4}$	0
■ t_4	$\frac{0}{8} \log \frac{4}{1}$	0	$\frac{1}{6} \log \frac{4}{1}$	0.23	$\frac{0}{7} \log \frac{4}{1}$	0	$\frac{0}{4} \log \frac{4}{1}$	0
■ t_5	$\frac{0}{8} \log \frac{4}{1}$	0	$\frac{0}{6} \log \frac{4}{1}$	0	$\frac{2}{7} \log \frac{4}{1}$	0.4	$\frac{0}{4} \log \frac{4}{1}$	0

$$t_{id} = \frac{n_{id}}{n_d} \log \frac{N}{n_i}$$

t_{id} : histogram bin of word i for image d

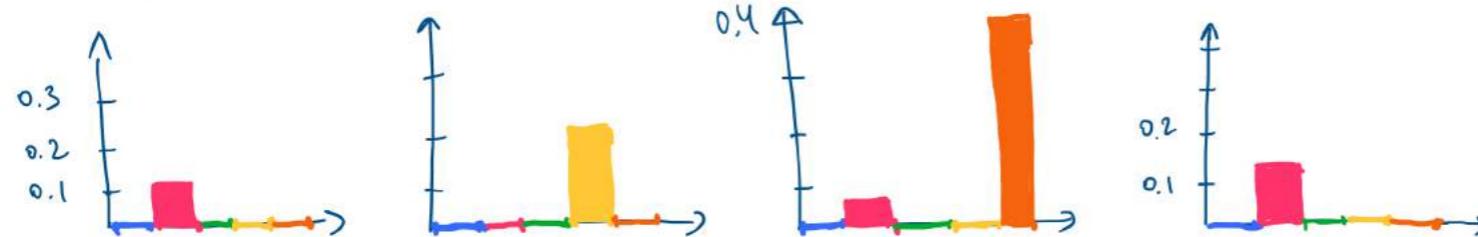
n_{id} : occurrences of word i in image d

n_d : number of word occurrences in image d

n_i : number of images that contain word i

N : number of images

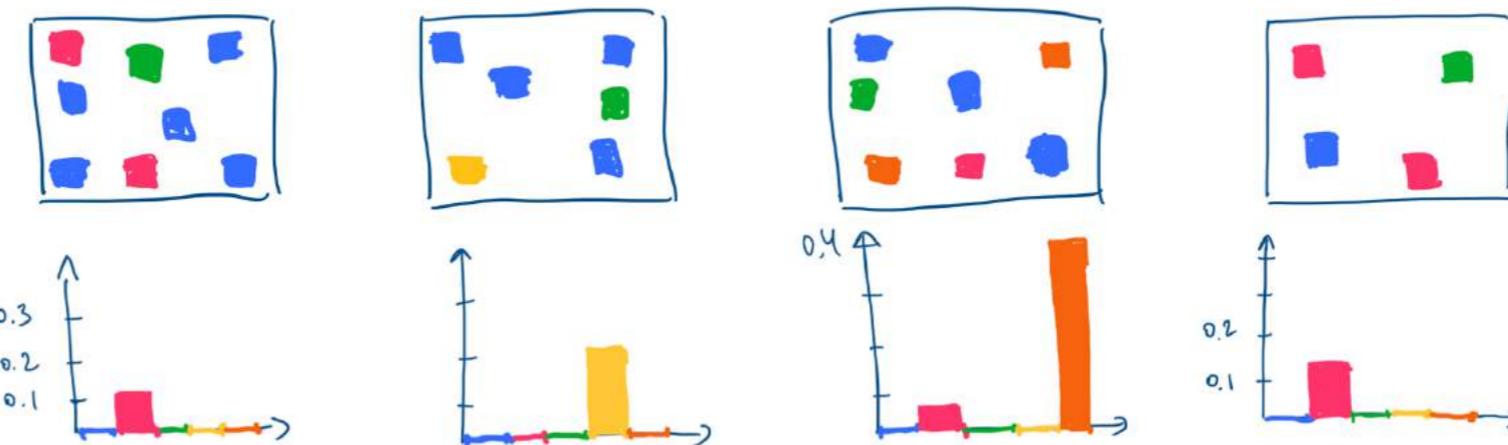
Re-weighted histograms



How do we compare the histograms (quantitatively)?

To compute similarity between images:

We first compute TF-IDF



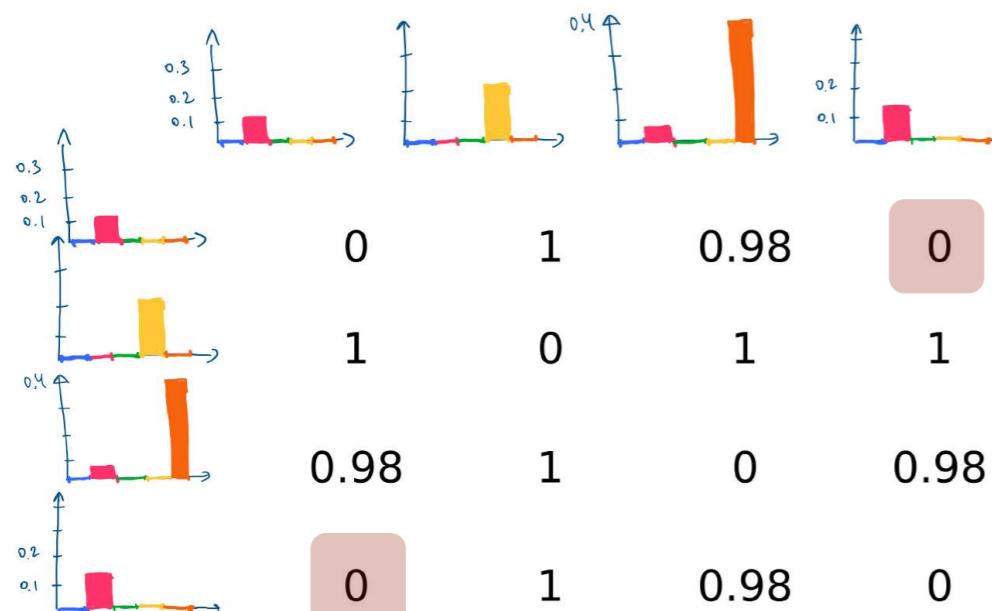
Cosine similarity considers the cosine of the angle between vectors

We compute the cosine distance

Takes values between 0 & 1

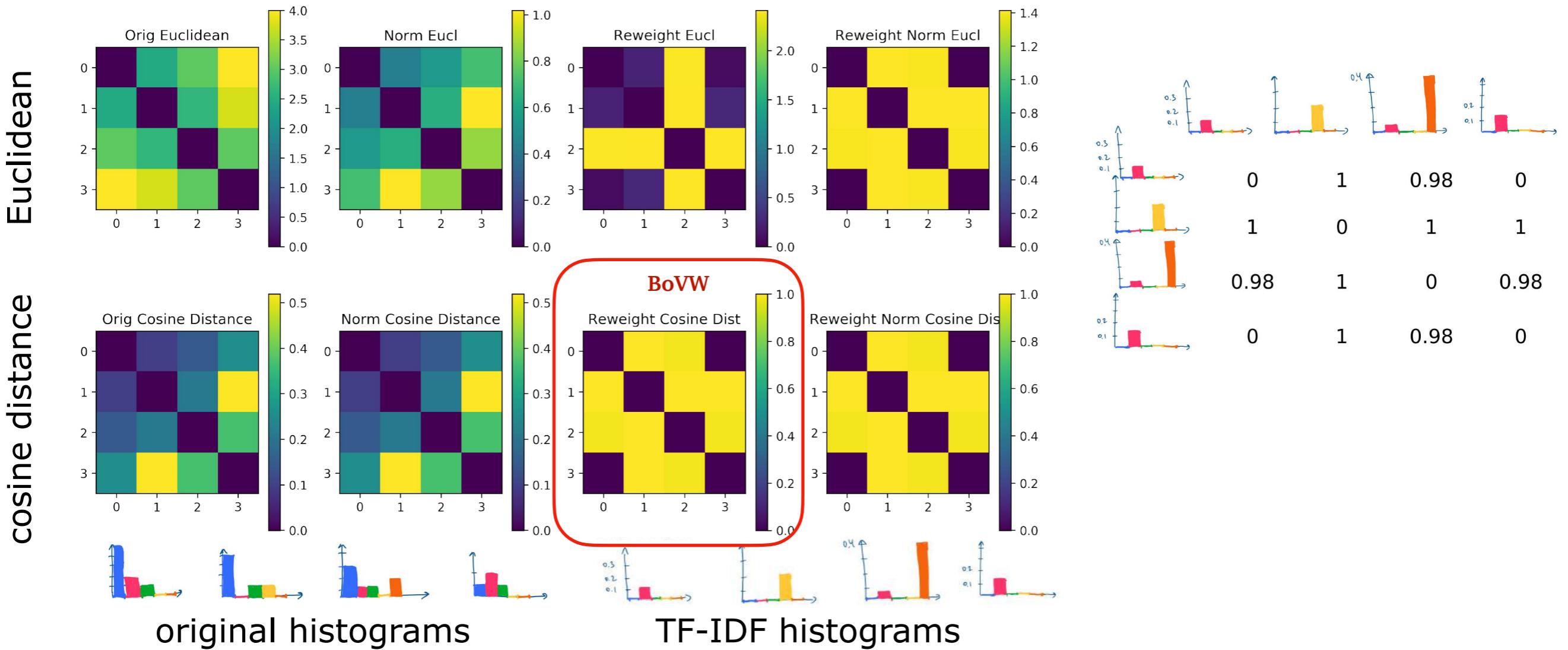
$$\text{cossim}(\mathbf{x}, \mathbf{y}) = \cos(\theta) = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

$$d_{\text{cos}}(\mathbf{x}, \mathbf{y}) = 1 - \text{cossim}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$



All the 4 images are self-similar to themselves, hence $d_{\text{cos}} = 0$ (diagonal)
images 1 and image 4 are similar

Comparison of distance metrics and with/out TF-IDF Reweighting



The normalised Euclidean distance is the distance between the two normalised vectors, which is $2(1 - d_{\cos})$
 (please see the Euclidean versus cosine slide in the end)

[Slide images: Cyrill Stachniss]

[Image courtesy: Olga Vysotska]

Large-scale image search within a database

Task:

Given a query image, find similar looking images

OR

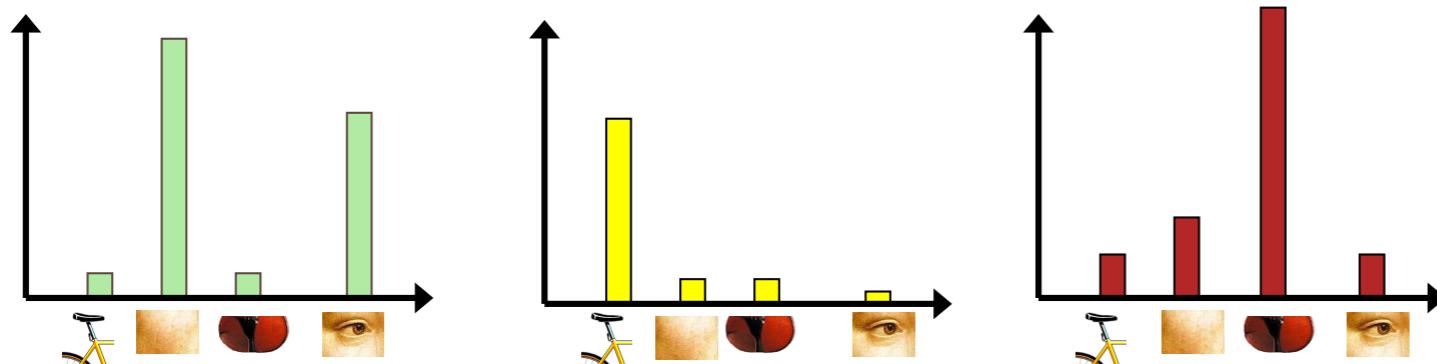
Given images from different classes, how do we learn a model for distinguishing them: classification problem

Inputs:

- i) Database of images
- ii) Dictionary
- iii) Query image(s)

Output:

The N most similar database images to the query image



Compute the bag-of-features representations of images from different classes

Treat the bag-of-features as vector for the classifier (k-means)

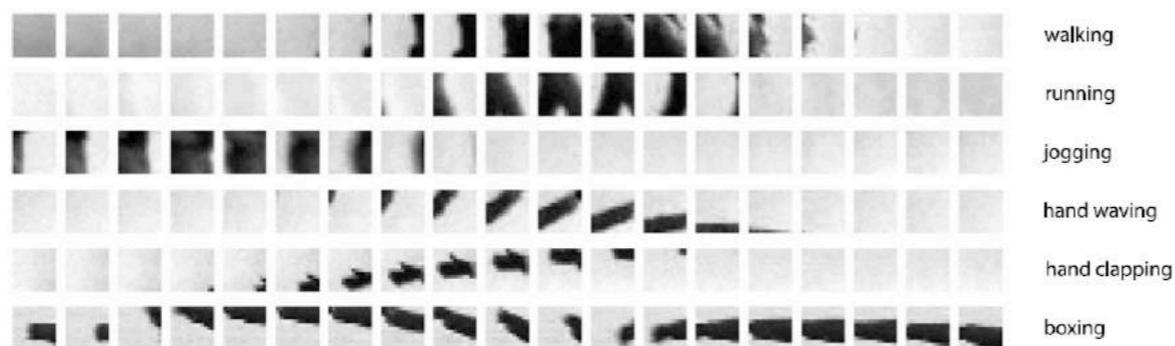
Cluster BoW vectors over image collection – Discover visual themes

Only consider database images whose bins overlap the query image

compute similarity between the query image and all the images in the database

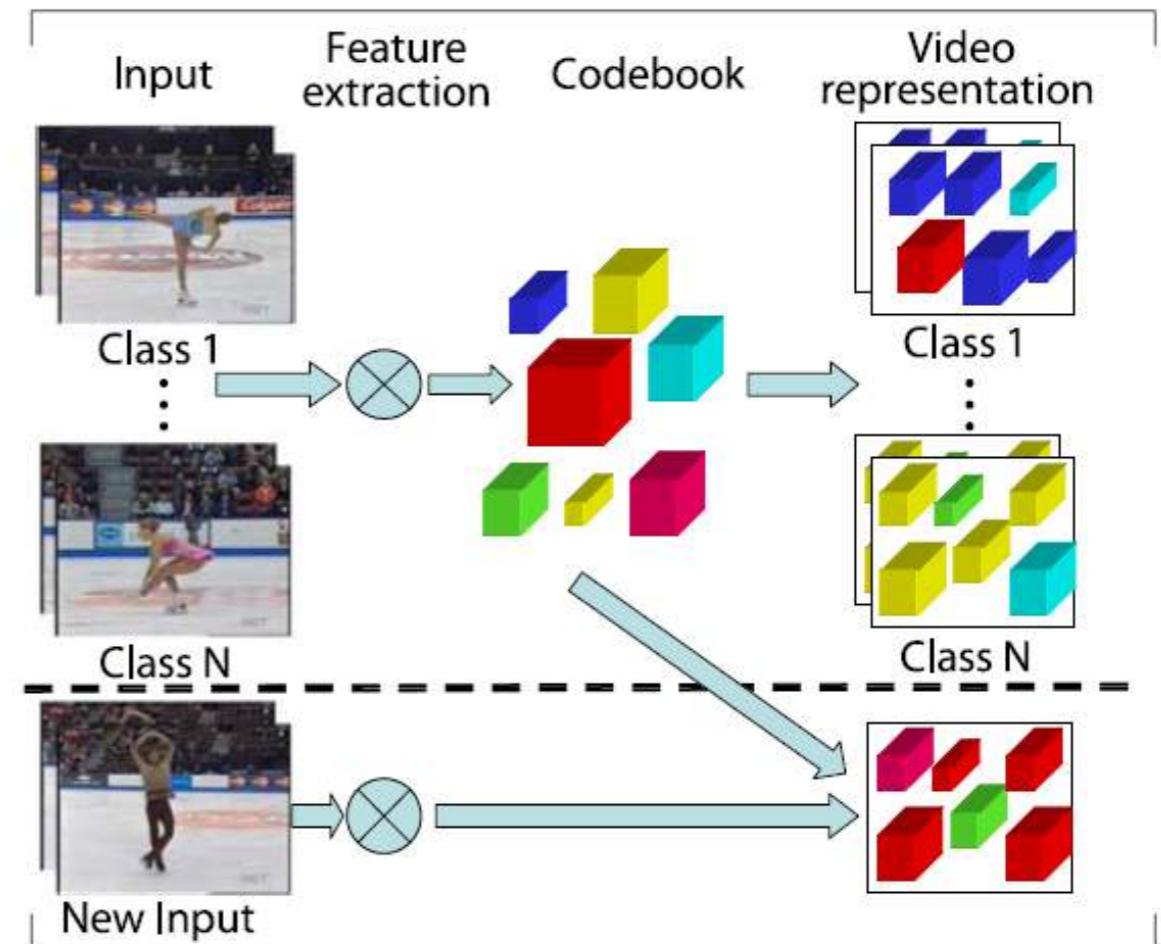
Bags of features for action recognition

Space-time interest points



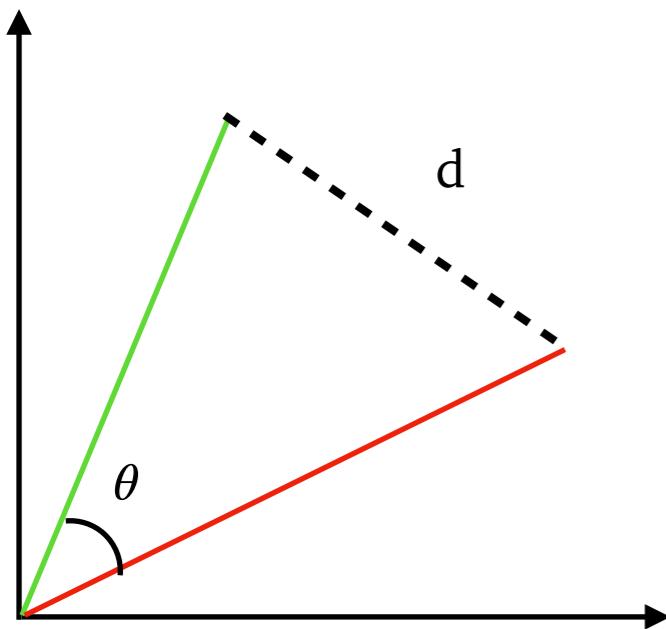
walking
running
jogging
hand waving
hand clapping
boxing

Feature extraction and description



Why cosine and not Euclidean?

- Essentially comparing frequency of ‘visual words’.
- Euclidean looks at distance (d), however often, some words might appear less frequently, but are yet significant.
- Hence, these are classified as farther apart using Euclidean measure,
- (based on 2 documents with different lengths but of similar context or here image context)
- Whereas cosine is used when magnitude does not matter (typically used for comparing text data).



- $\text{Cos}0^\circ=1, \text{Cos}90^\circ=0,$
- So $1 - \text{cos}0^\circ$ will give 0, (cos distance), angle between the vectors is 0° , then they are similar.
- For 90° , the vectors have a cos distance of 1 (apart).
- We are comparing the orientation only.
- For diametrically opposite vectors ($\text{Cos}180^\circ=-1$), which gives cos distance of 2 (they are farther apart in their orientation independently of their magnitude).

Euclidean distance between 2 vectors is given by L2-norm,

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|^2 &= (\mathbf{x} - \mathbf{y})^\top (\mathbf{x} - \mathbf{y}) \\ &= \mathbf{x}^\top \mathbf{x} - 2\mathbf{x}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} \leftarrow \text{Squared L2 norm}\end{aligned}$$

as $\|\mathbf{x}\| = \|\mathbf{y}\| = 1$

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|^2 &= 2 - 2\mathbf{x}^\top \mathbf{y} = 2 - 2\cos\theta \\ &= 2 d_{\cos}(\mathbf{x}, \mathbf{y})\end{aligned}$$

For normalised vectors: $\|\mathbf{x}\|=1, \|\mathbf{y}\|=1, \mathbf{x}^\top \mathbf{x}=1, \mathbf{y}^\top \mathbf{y}=1$,
Euclidean distance(\mathbf{x}, \mathbf{y}) = $2 * \text{cosine distance}(\mathbf{x}, \mathbf{y})$

Implementation session of BoVW:
we will use
http://www.vision.caltech.edu/Image_Datasets/Caltech101/

Viola Jones algorithm for face detection

Lecture 26-CV- DSE312

Bhavna R

Face detection:

Where are the faces?



Not who they are, that's recognition or identification.

Object Detection vs Object Recognition

- recognition involves stating whether an image contains a specific object or not.
- detection requires the position of the object inside the image.

Face recognition: human face or not face



Face detection challenge: Face views at different angles & a monkey



Classical Viola-Jones face detection method

- Identify Haar-like feature matrix to calibrate the face feature
- Rectangular features used
- Form integral images, a new image representation that allows fast calculation of image features
- Classifier learning: Formulate AdaBoost algorithm to construct strong and weak classifiers
- form a screening cascade classifier

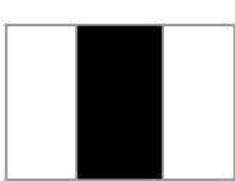
The Haar features

“Rectangular” filters



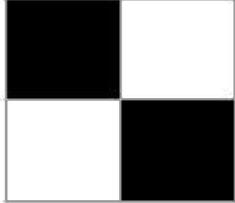
1. Edge Features

2 Rectangular Haar features



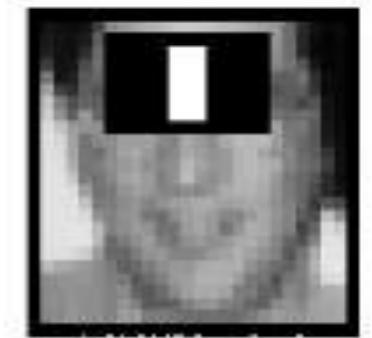
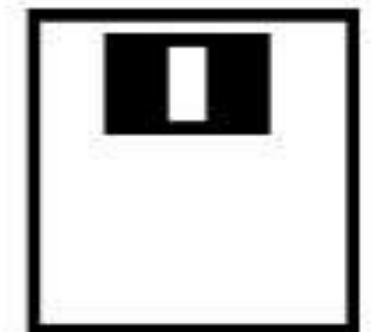
2. Line Features

3 Rectangular Haar features



3. Four rectangle Features

4 Rectangular Haar features



Haar features relevant for face detection;
implemented as convolutional kernels.

- 2 rectangular feature: difference between the sum of the pixels within 2 rectangular regions (of same shape and size and are horizontally and vertically adjacent).
- 3 rectangular feature computes the sum in a centre rectangle.
- 4 rectangular feature computes the difference between diagonal pairs of rectangles.
- Various variations of these regions of different sizes are convolved through the image in order to get multiple filters that will be inputs to the AdaBoost training algorithm.
- Calculation of these features using the standard technique would require a high computation time.
- In order to reduce this time, a new approach called the integral image was suggested by the authors.

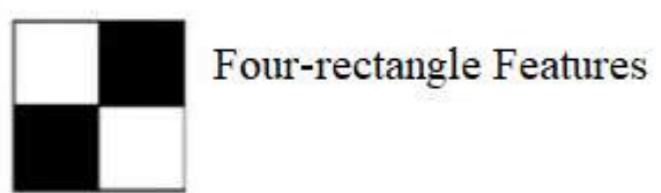
Why these features for face detection?



Edge Features



Line Features

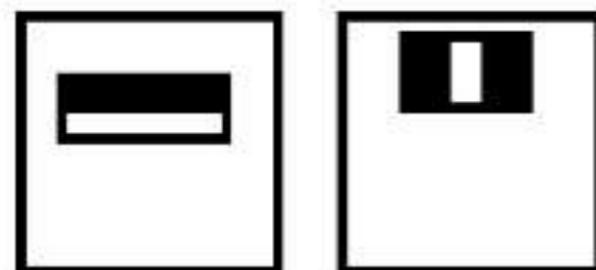


Four-rectangle Features

Important Features for Face Detection



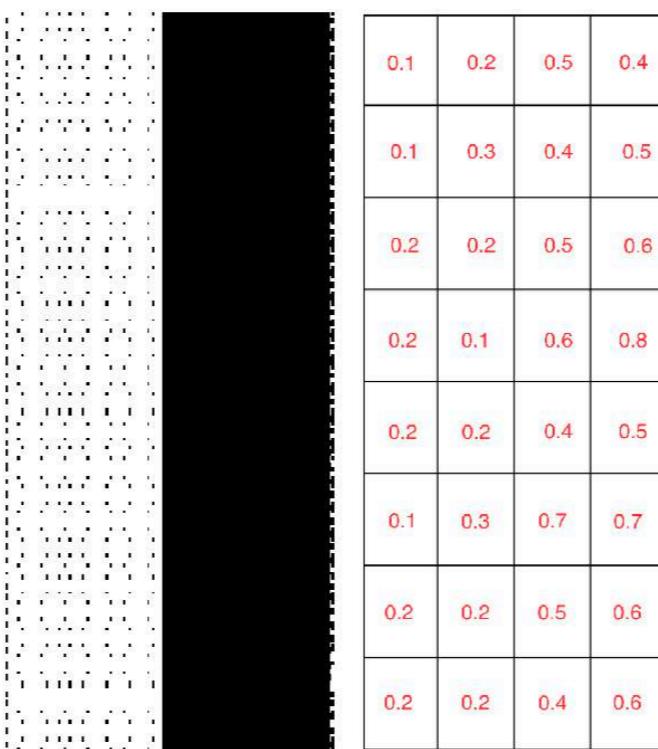
Haar features are sensitive to directionality of pattern in the image



horizontal and vertical features describe eyebrows and the nose, respectively, look like to the machine.

calculation:

Subtract White area from the Black area.



$$(0.5 + 0.4 + 0.5 + 0.6 + 0.4 + 0.7 + 0.5 + 0.4 + 0.4 + 0.5 + 0.6 + 0.8 + 0.5 + 0.7 + 0.6 + 0.6)$$

-

$$(0.1 + 0.1 + 0.2 + 0.2 + 0.2 + 0.1 + 0.2 + 0.2 + 0.2 + 0.3 + 0.2 + 0.1 + 0.2 + 0.3 + 0.2 + 0.2)$$

$$B - W = 8.7 - 3 = 5.7$$

Additions & subtractions faster than multiplications
For Haar filter of size $N \times M$, the computational cost:
 $N \times M - 1$ additions per pixel per filter per scale

Features seem good, but computing features in this manner can become computationally expensive

What is an integral Image?

1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

Input Image

0	0	0	0	0	0
0	1	3	5	9	10
0	4	10	13	22	25
0	6	15	21	32	39
0	10	20	31	46	59
0	16	29	42	58	74

Integral Image

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	43	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

Original Image

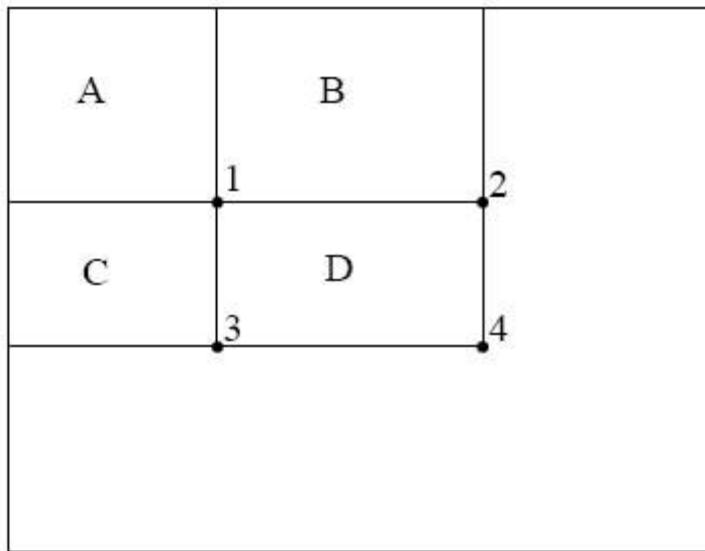
64	66	69	130	190	196	203	260
73	130	187	260	333	390	446	520
90	194	297	390	484	584	683	780
130	260	390	520	650	780	910	1040
162	326	491	650	808	976	1145	1300
203	390	577	780	983	1170	1357	1560
252	454	655	910	1166	1364	1561	1820
260	520	780	1040	1300	1560	1820	2080

Integral Image

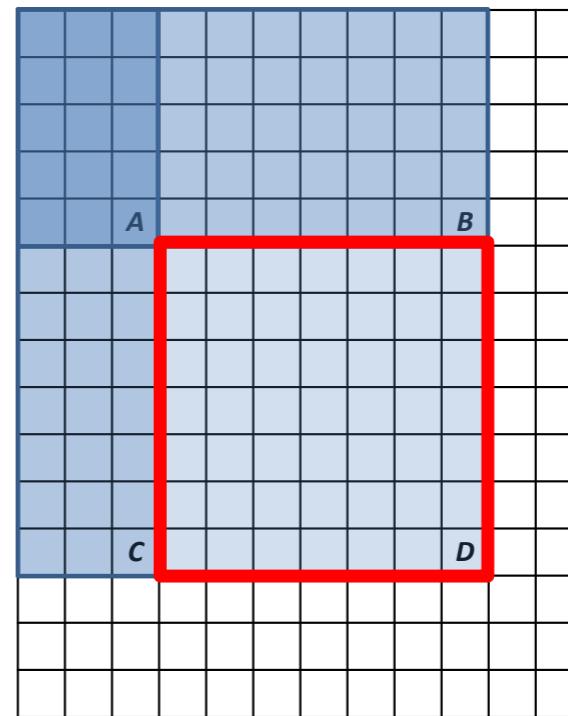
A given pixel in the integral image is the sum of all the pixels on the left and all the pixels above it.

$$3+2+5+4=14$$

$$46+10-22-20 = 14$$

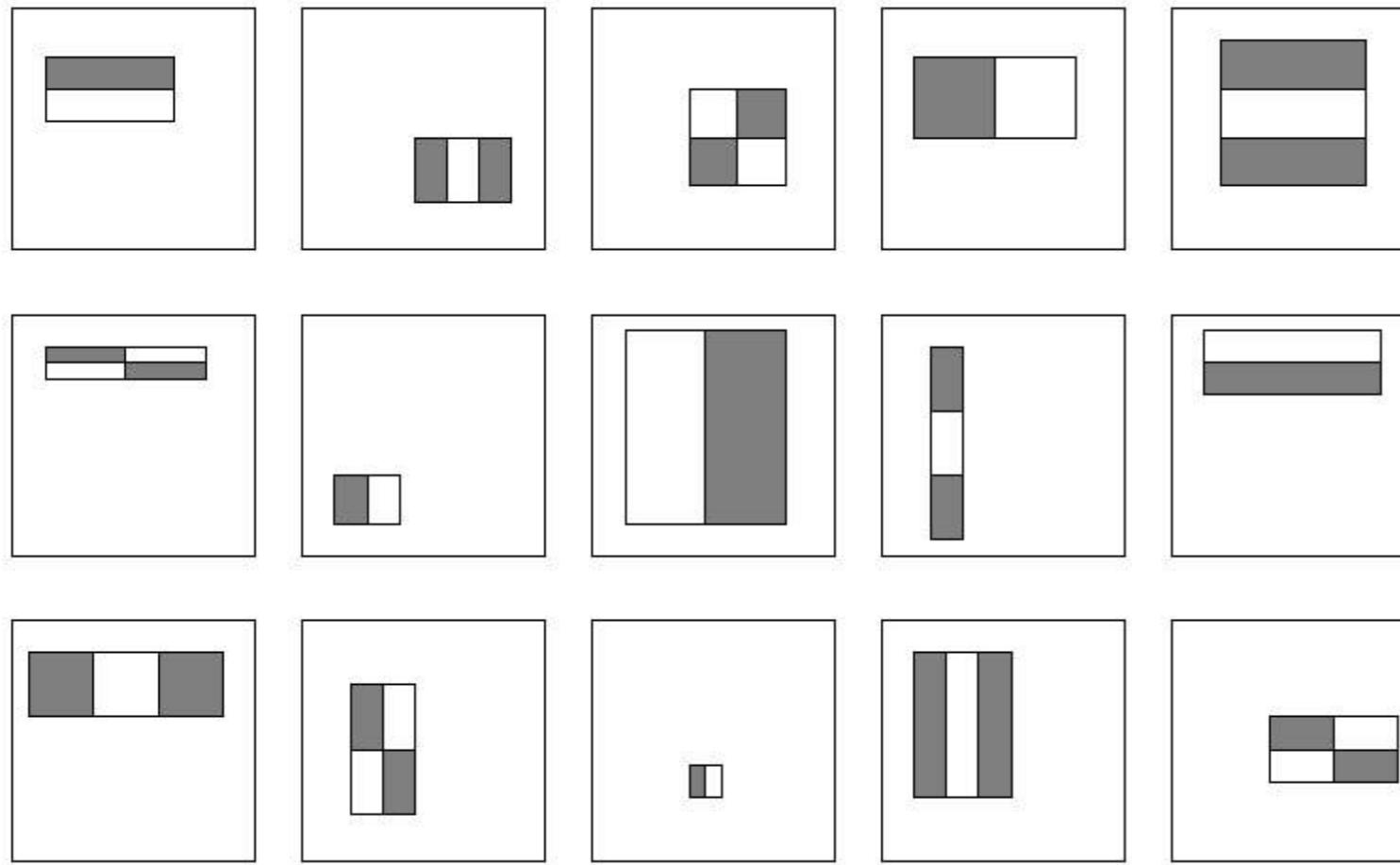


integral image values:
 location 1: Sum of pixels in A
 location 2: A+B
 location 3: A+C
 location 4: A+B+C+D
 So sum within D is $4+1-(2+3)$



sum within red box:
 $D+A-B-C$

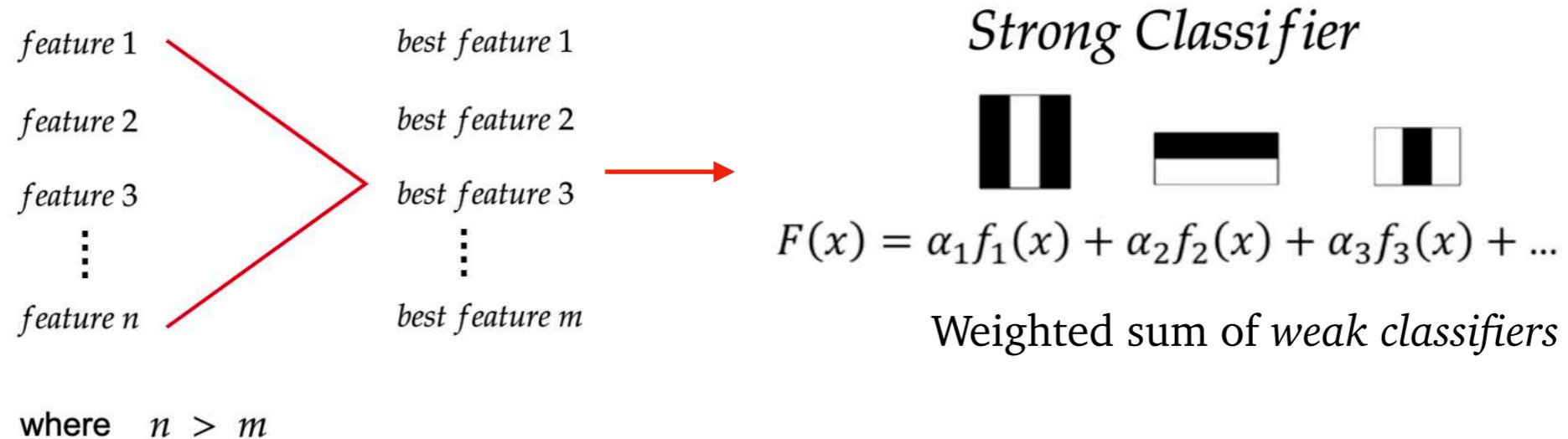
Large library of filters



- Haar-like features are rectangular and integral image process allows us to find features easily, makes the calculations much less intensive for any facial detection model.
- Features are extracted from sub-windows of a sample image.
- The base size for a sub window is 24 by 24 pixels.
- Each of the feature types are scaled and shifted across all possible combinations
- In a 24 pixel by 24 pixel sub window there are $\sim 160,000$ possible features to be calculated.
- For different sized faces, compute features at different scales

A quick way to compute features, now we have lots of features!

AdaBoost (Adaptive Boosting) Algorithm



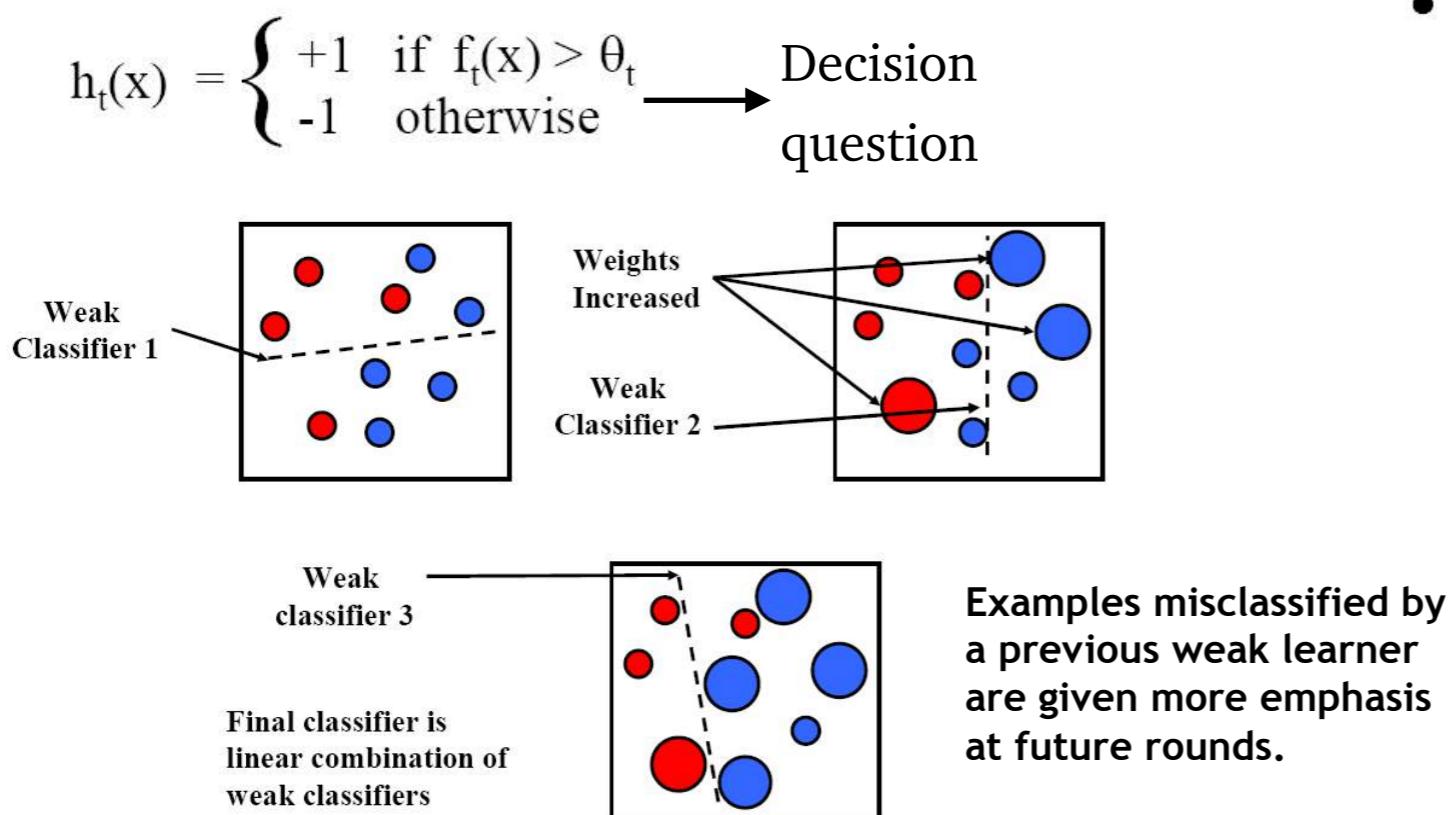
Can we make a strong classifier by combining several weak classifiers that are allowed to vote?

- We have a vast feature set with possibly many irrelevant features
- AdaBoost: a machine learning algorithm for selecting the best subset of features
- We build binary classifiers using the features
- algorithm outputs a classifier (Hypothesis Function) called a “*Strong Classifier*”.
- Strong Classifier is made up of a linear combinations of “*Weak Classifiers*” (best features).

AdaBoost (Adaptive Boosting) Algorithm: training with many features

How can we learn a classifier with only a few hundred training examples without overfitting?

- Consider 2D feature space with positive and negative examples.
- Classify the data and look for the errors (such as Decision Tree using “stumps”) for training
- Look for a “weak classifier” based on the error rate.
- Reweight the data so that the inputs where we made errors get higher weight in the learning process
- Now learn a 2nd simple classifier on the weighted data
- Combine the 1st and 2nd classifier and weight the data according to where they make errors
- Learn a 3rd classifier on the weighted data and so on
- Run for T iterations where T is the number of weak classifiers to be found by the user.
- In each iteration, the algorithm finds the error rate for all features and then choose the feature with the lowest error rate for that iteration.
- This procedure is called “Boosting”



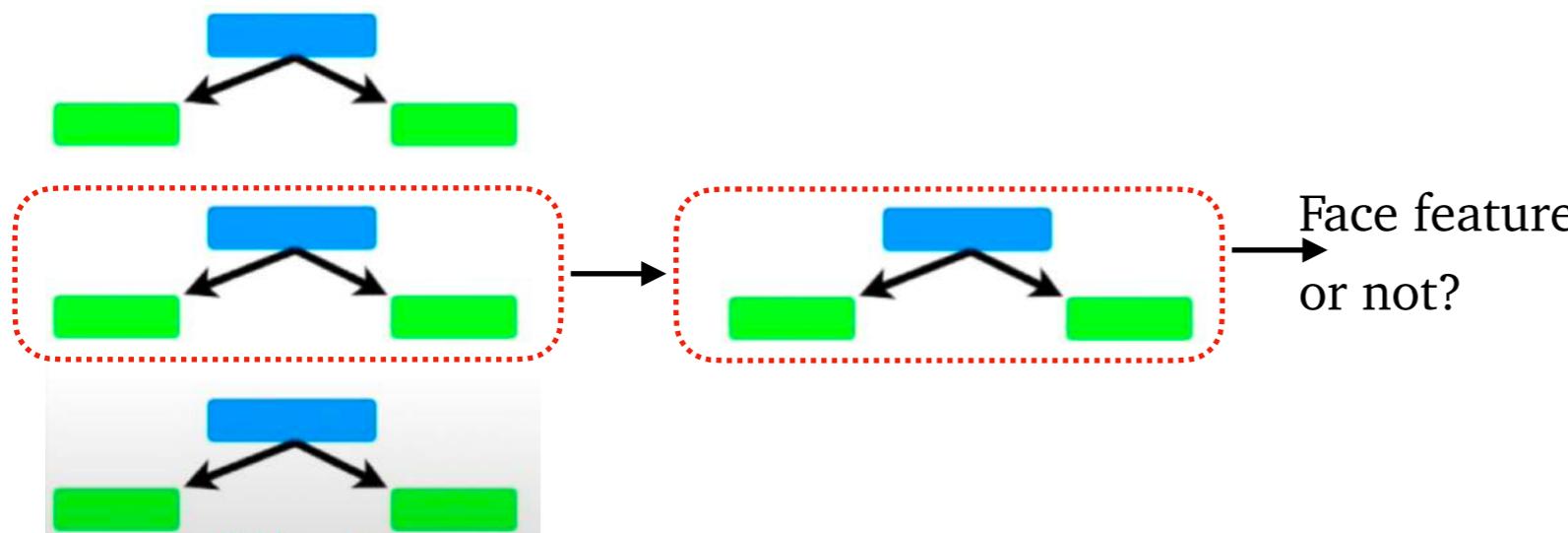
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

Iteratively build on the relative weights of the misclassified predictions are altered and increased in order to lay more emphasis on these predictions while making the next predictor.

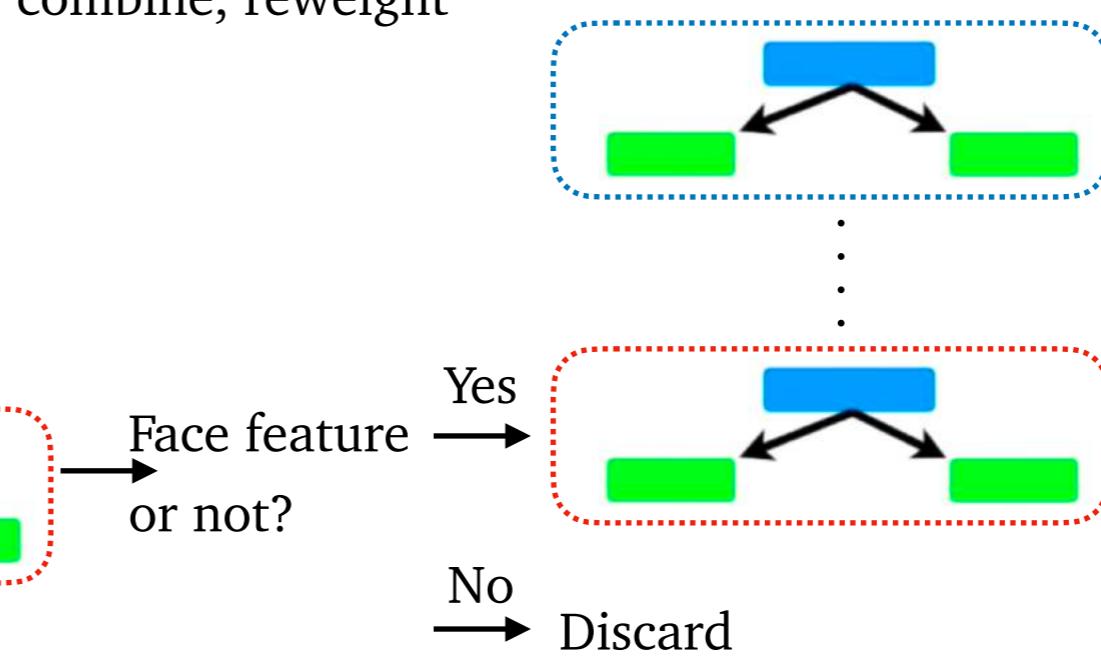
Boosting with ‘decision stumps’ and building Cascade Filter

- decision tree with only a single root node, uses only a single attribute for splitting
- looks at all possible thresholds for each attribute
- Selects the one with the max information gain
- Resulting classifier is a simple threshold (t) on a single feature (f) ($f > t$, yes, else no)
- Combine it with the other previously selected classifiers
- Re-weight the data
- Learn all K classifiers again, select the best, combine, reweight
- Repeat until we have T classifiers selected



Decision stump: one node & two leaves

Ada-boost builds several decision stumps



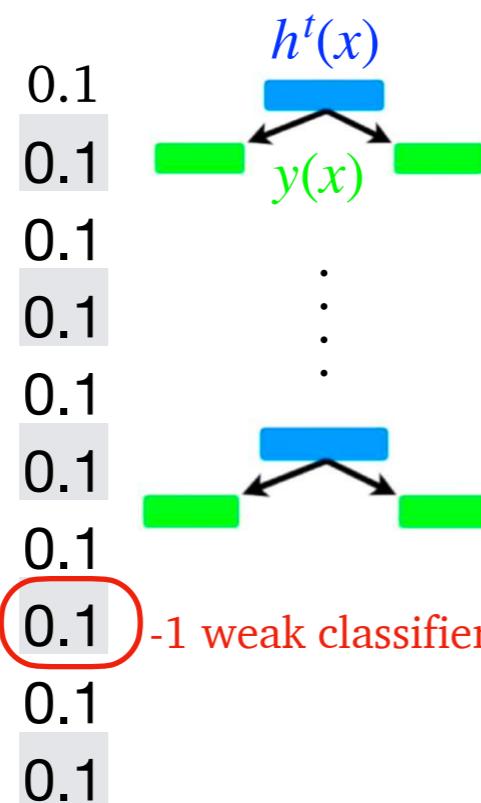
- Strong features are formed into a binary classifier. : Positive matches are sent along to the next feature. Negative matches are rejected and exit computation.

Lets say, we have 10 samples:

N=10

Initial weight

$$w_i^t = \Sigma \frac{1}{N}$$



Update weights:

$$w_i^{t+1} = \frac{w_i^t}{Z} e^{-\alpha_t} (h^t(x) y(x))$$

$\varepsilon^t = 0.1$, so $\alpha_t = 1.0986$ & Z=0.6

Let us say the decision stumps got
1 wrong, 9 correct.

Then, compute w_i^{t+1} for condition

$$(h^t(x)y(x)) = 1 \text{ or } -1$$

$$w_i^{t+1} = 0.05 \text{ for } (+1),$$

$$w_i^{t+1} = 0.5 \text{ for } (-1),$$

0.05

0.05

0.05

0.05

0.05

0.05

0.05

05

0.05

0.05

$$\Sigma w_i^{t+1} \circ$$

Update weights

Condition: $\sum w_i = 1$,
 t is the time step

$$\sum w_i^{t+1} = (0.05)^*9 + 0.5 = 0.95$$

Readjust sum of weights s.t. $\sum w_i^{t+1} = 1$,

Updated weights:

$$(0.05/0.95)*9 + (0.5/0.95) \approx 1$$

Update weights:

$$w_i^{t+1} = \frac{w_i^t}{Z} e^{-\alpha_t} (h^t(x)y(x))$$

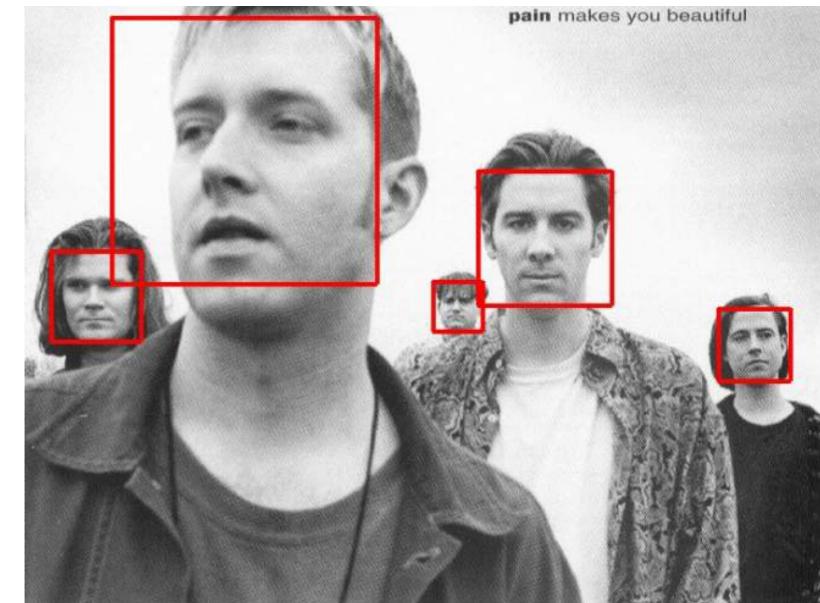
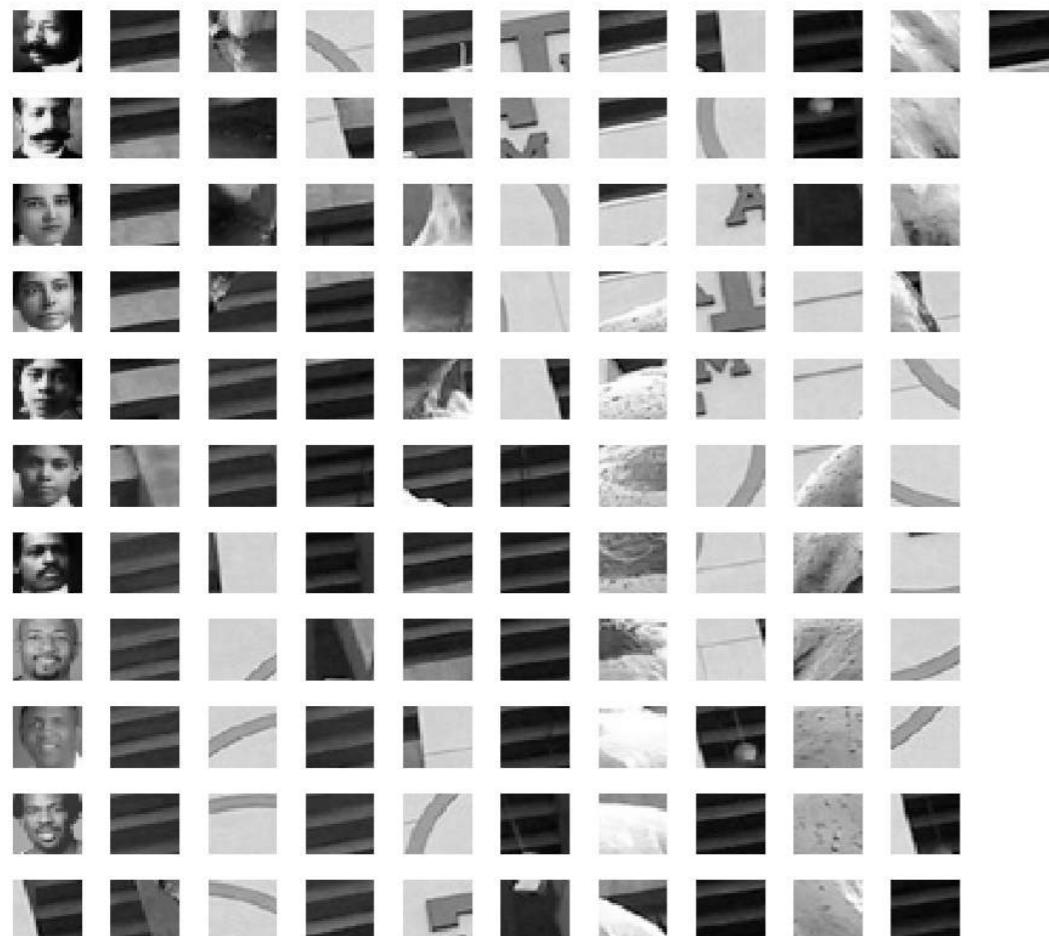
Z (normalizer): $2\sqrt{\varepsilon^t(1 - \varepsilon^t)}$,

$$\text{error bound: } \alpha_t = \frac{1}{2} \log \frac{1 - \varepsilon^t}{\varepsilon^t}$$

Examples: Viola-Jones face detection results

detection at multiple scales

Small set of 111 Training Images



Results from a viola jones face detector



Assignment: Implementing Viola Jones algorithm for face detection

Practical considerations for implementation

- Basic classifier operates on 24 x 24 subwindows

- Scaling:

- Scale the detector (rather than the images)

- Features can easily be evaluated at any scale

- Scale by factors of 1.25

- Location:

- Move detector around the image (e.g., 1 pixel increments)

- Final Detections

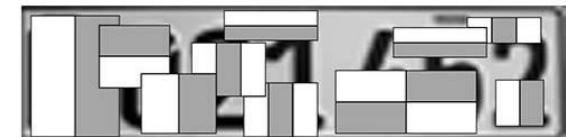
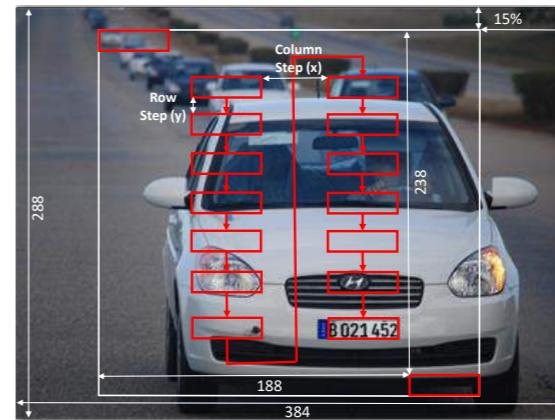
- A real face may result in multiple nearby detections

- Post-process detected sub-windows to combine overlapping detections into a single detection

OpenCV: `cv::CascadeClassifier`

Matlab: `vision.CascadeObjectDetector` (computer vision toolbox)

Other applications: eg. license plate detection



Requires training images with Haar features

Elias A. Perdomo Hourné

<https://www.kaggle.com/ciplab/real-and-fake-face-detections>

Object detection and recognition as a classification problem

DSE-312 CVLecture 27

Image Classification problem: Scene understanding

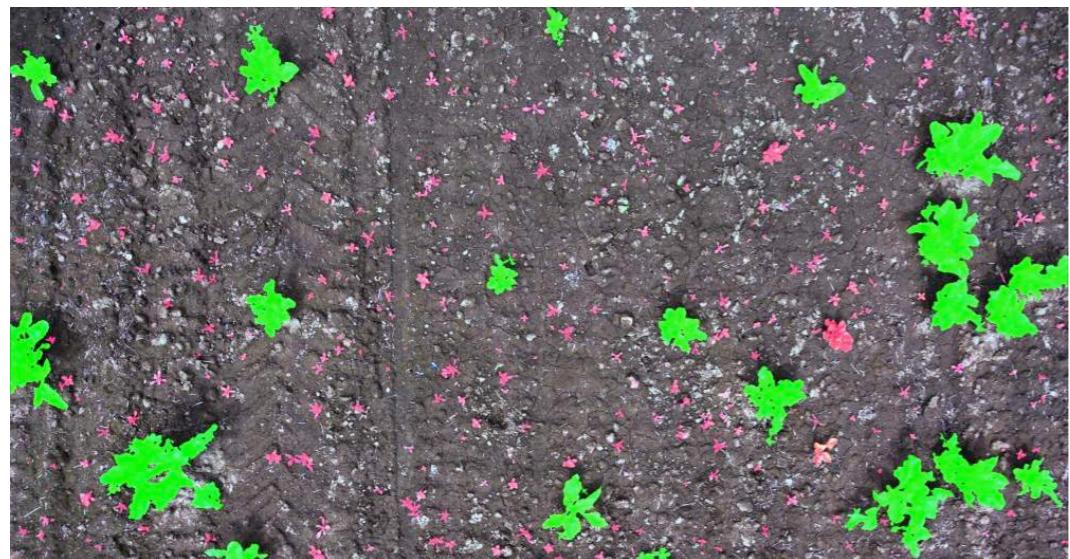
Is this an agriculture land Or street view?



OR



Classification of objects within the image



weed vs crop

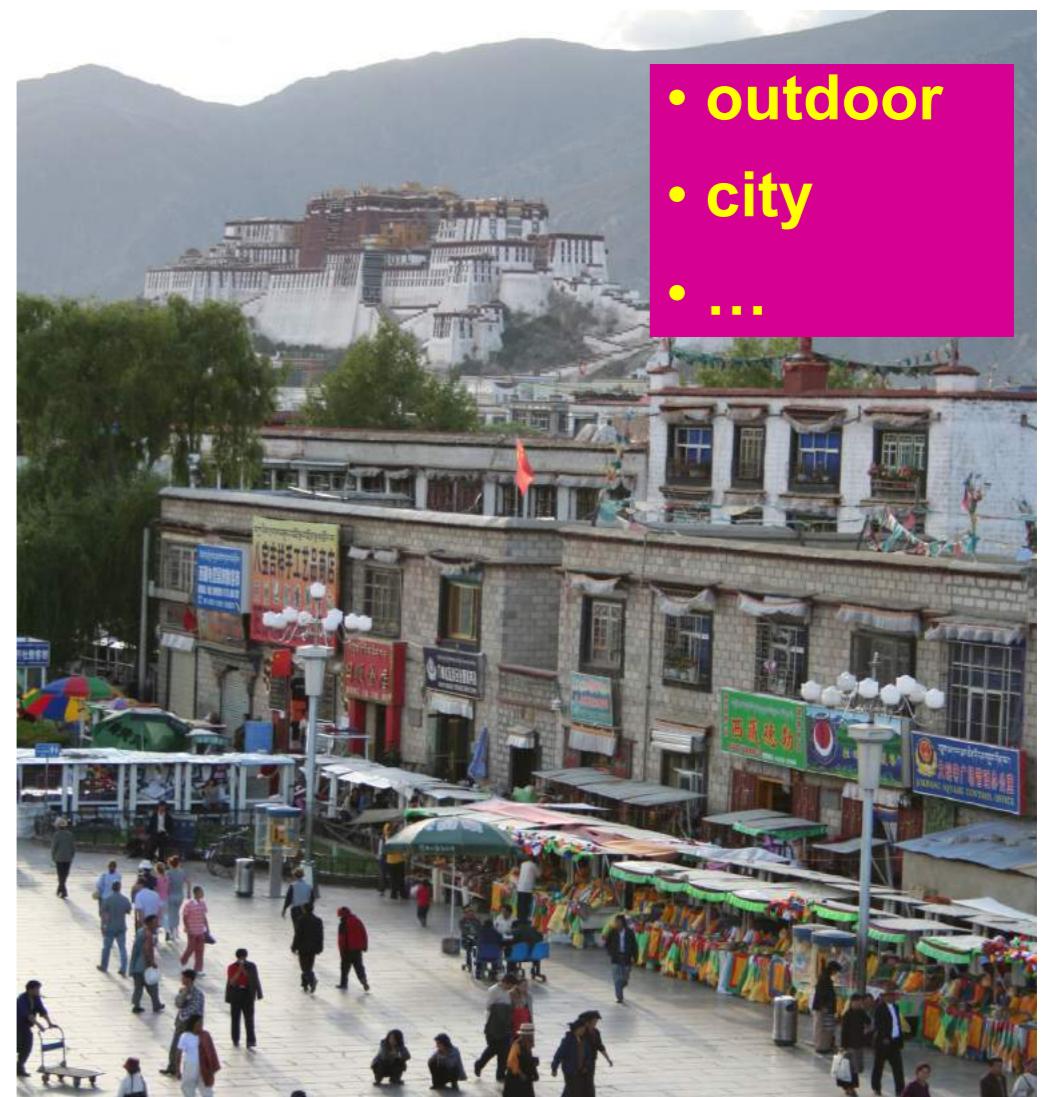


Cars, streets, pedestrian crossing

Object categorization

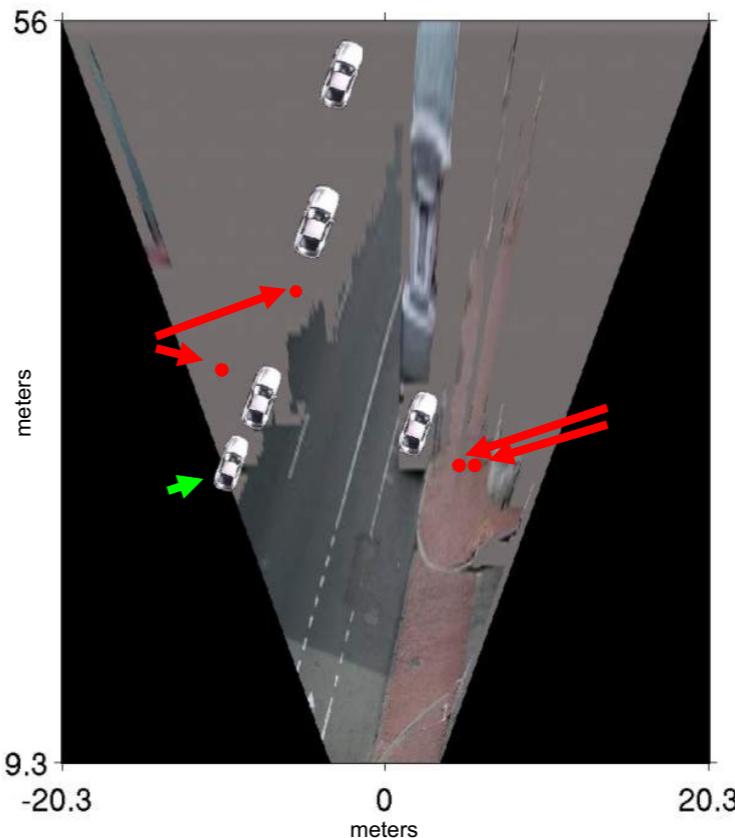


Scene and context categorization

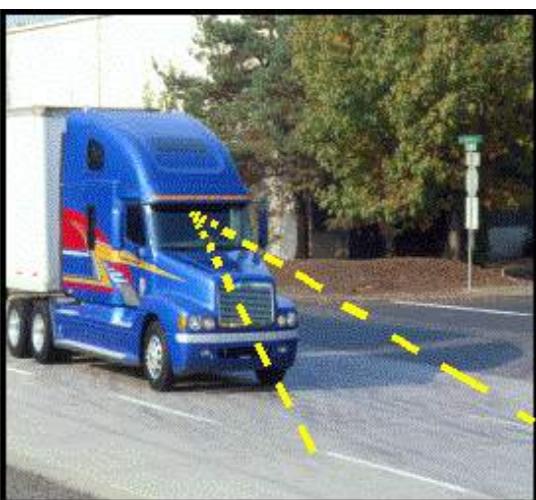


Applications: Assisted driving

Pedestrian and car detection



Lane detection



- Collision warning systems with adaptive cruise control,
- Lane departure warning systems,
- Rear object detection systems,

Applications: image search



Places

[London](#)
[New York](#)
[Egypt](#)
[Forbidden City](#)

Celebrities

[Michael Jordan](#)
[Angelina Jolie](#)
[Halle Berry](#)
[Seth Rogen](#)
[Rihanna](#)

Art

[impressionism](#)
[Keith Haring](#)
[cubism](#)
[Salvador Dalí](#)
[pointillism](#)

Shopping

[evening gown](#)
[necklace](#)
[shoes](#)

Refine your image search with visual similarity

Similar Images allows you to search for images using pictures rather than words. Click the "[Similar images](#)" link under an image to find other images that look like it. Try a search of your own or click on an example below.

[paris](#)



[Similar images](#)



[Similar images](#)



[Similar images](#)



[Similar images](#)

[temple](#)



[Similar images](#)



[Similar images](#)

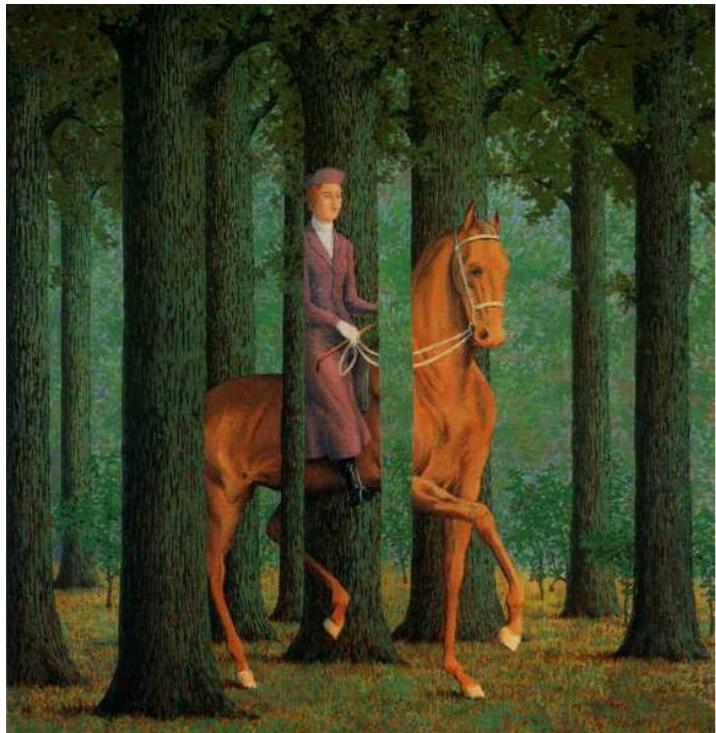


[Similar images](#)



[Similar images](#)

Challenges in image detection & classification



viewpoint variations



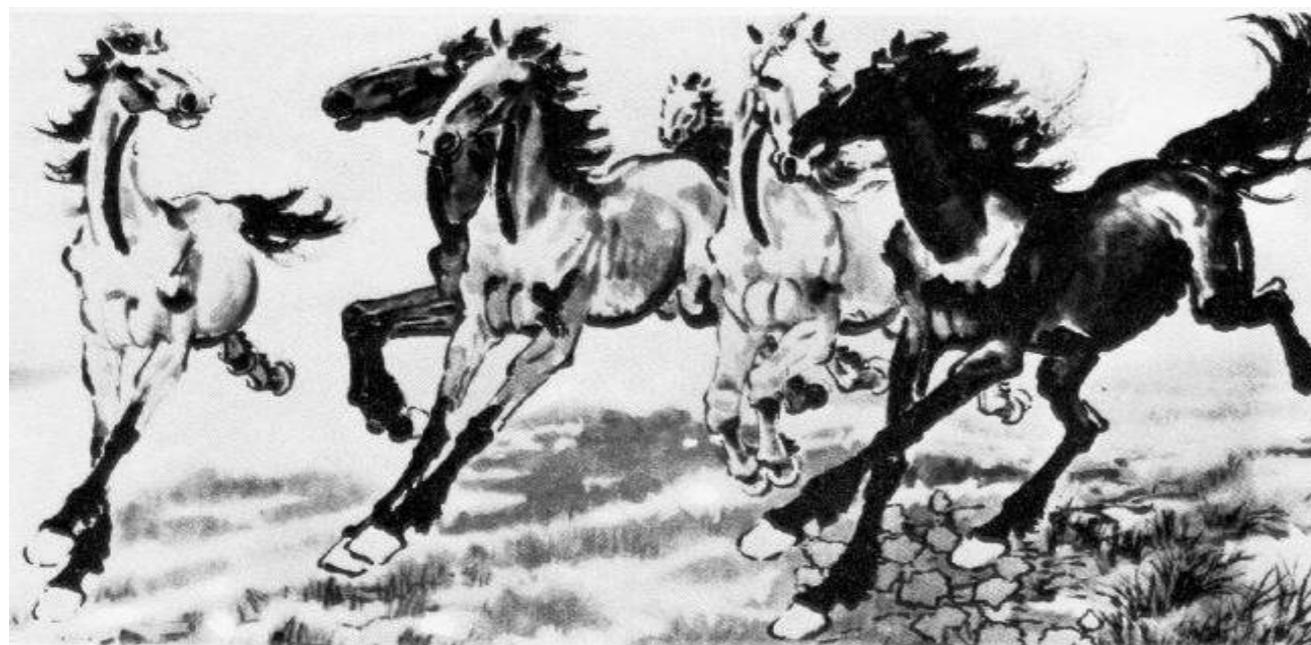
illumination variation



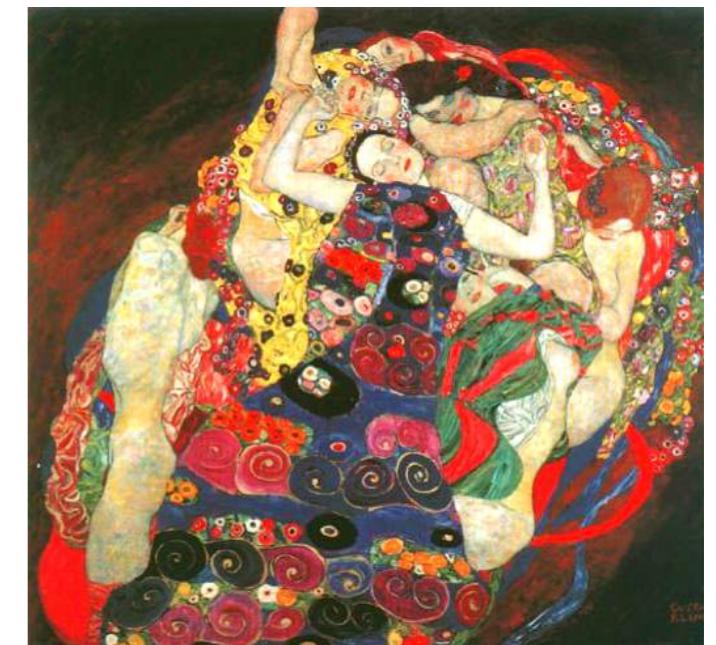
and small things
from Apple.
(Actual size)



image scales



object contour variations



background clutter

Challenges: Intra-class variations



Two aspects of classification

features/ feature sets

learning technique

- attributes of the data elements based on which the elements are assigned to various classes
- features may be qualitative (high, moderate, low) or quantitative
- feature set depends upon the domain
- prototype pixels serve as data inputs for unsupervised algorithms
- In case of images, features can be high-level attributes drawn out using sophisticated methods (for machine learning)

features: Harris, HoG, SIFT, Haar-like

- classifier are guided to learn the relationship between the data and the classes using feature sets
- when access to domain knowledge is missing or inaccessible, the data are grouped into subsets or clusters based on statistical similarity
- when good quality training data is available, machine learning is the choice
- At times, unsupervised classifiers are used to carry out preliminary analysis of data prior to supervised classification

Methods for Category Recognition

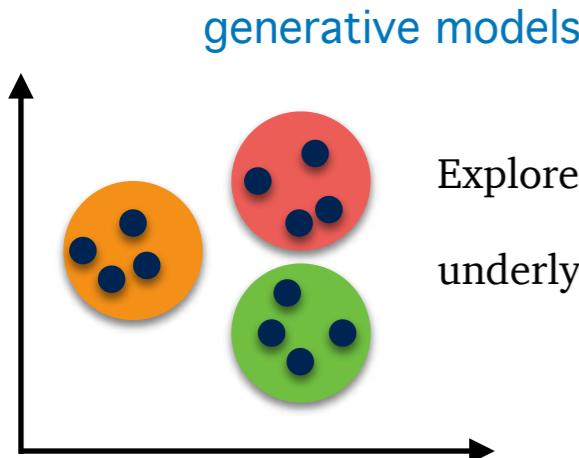
Machine learning: unsupervised

find natural groupings and patterns in data using clustering techniques / measures of similarity

eg:

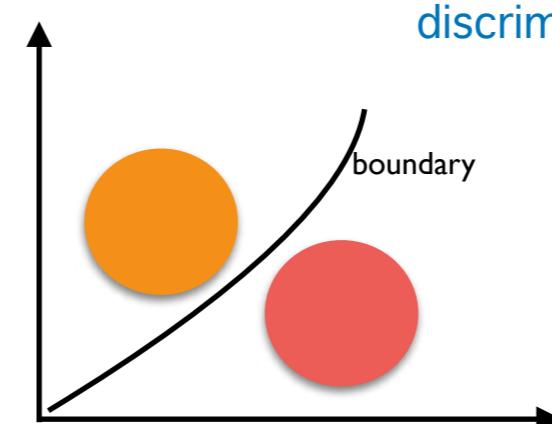
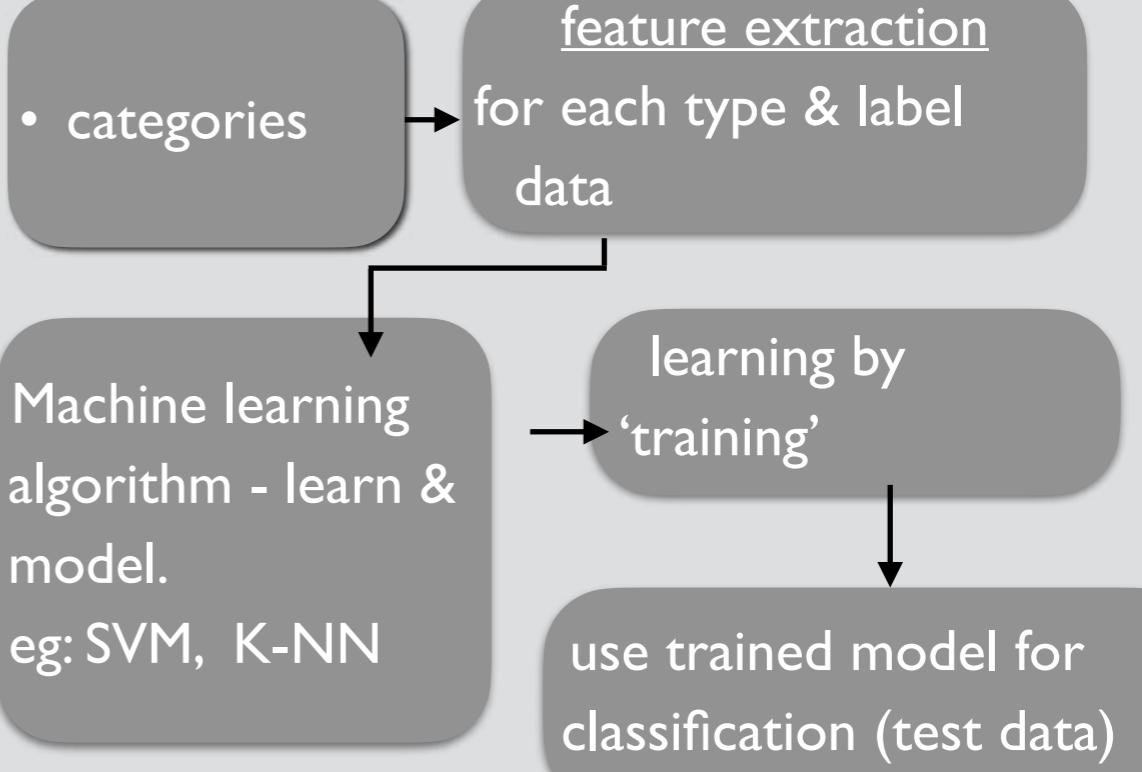
grouping based on specific feature similarity

algorithms: Kmeans clustering, principal component analysis, Gaussian mixture models



- Discovers underlying pattern in the dataset based on similarity of data points (clustering algorithms) or uncover hidden relationships of variables

Machine learning: supervised

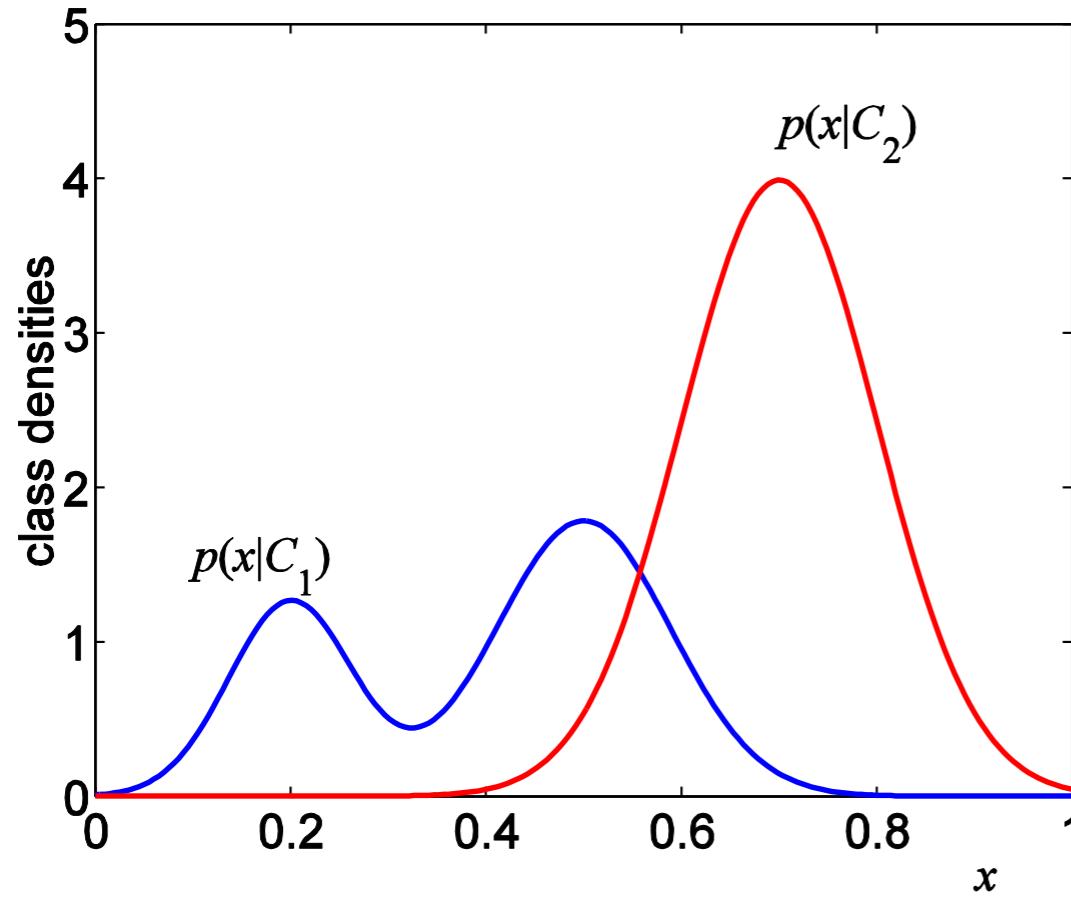


mapping from one set of variables (data) to another set of variables (information classes)

Finding decision boundary

- learns a function to make prediction of a defined label based on the input data (requires training). It can be either classifying data into a category (classification problem) or forecasting an outcome (regression algorithms).
- binary or multi-class approach

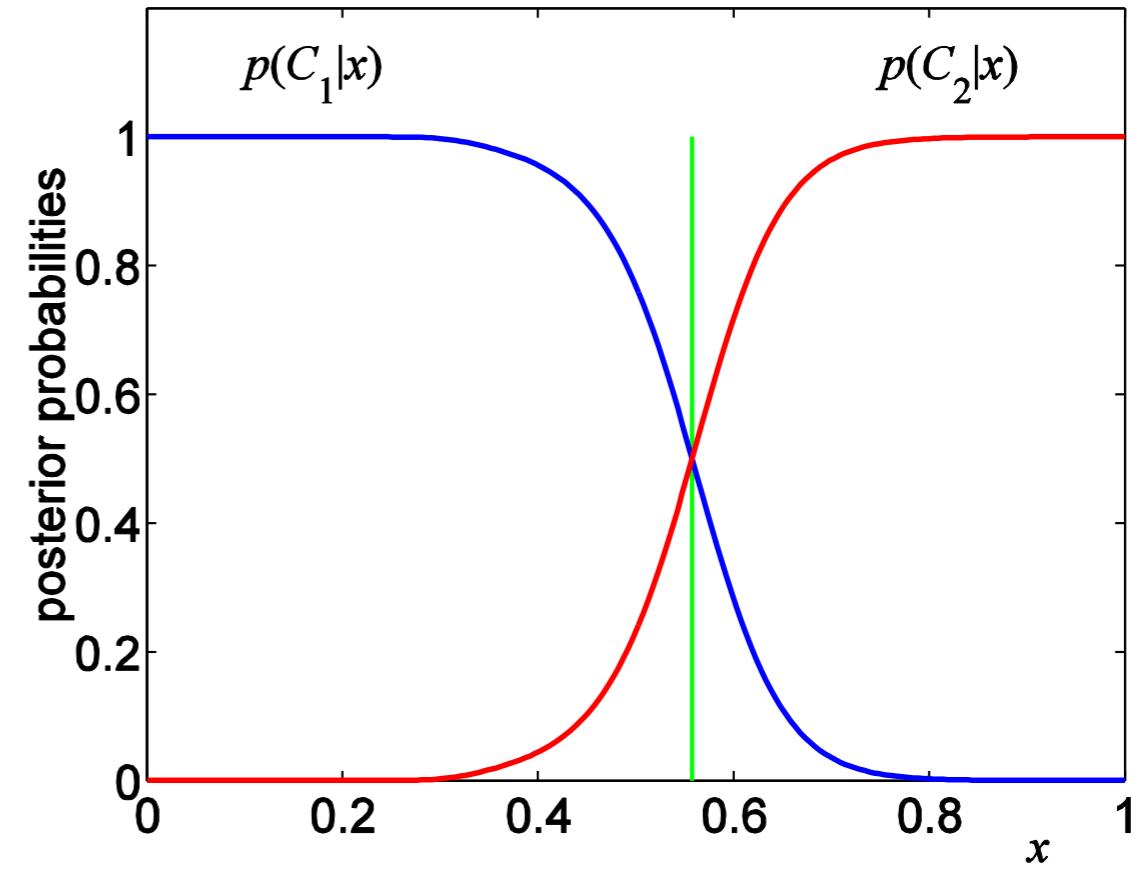
Generative and discriminative machine learning models



Generative Approach
model individual classes, priors

Eg:
Gaussian mixture models
K-means clustering

Models the “shape” of each class



Discriminative Approach
model posterior directly

Eg:
K-NN
SVM

Models boundaries between classes

Machine learning workflow for image classification

Training: find the decision boundary

Data

- Class A
- Class B

Image Features
extract discriminative
information from the images

Supervised learning
Data- Features- Labels

Data Model



Obtain
Classifier
for training

Prediction: use trained model into applications

Test Data
(not used in **training**)

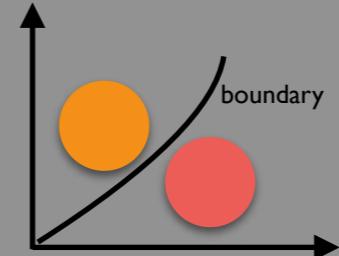
Feature Extraction
same set of features as used
for **training**

What features?

Computer vision tools

- macro-level analysis
- local image features to better handle scale changes, rotation

Use Classifier



Classify test

Prediction
on New Data:

- Class A?
- Class B?

Under-fitting & Overfitting problems

Under-fitting: model is too “simple” to represent the relevant characteristics

- High bias and low variance
- High training error and high test error

Over-fitting: model is too “complex” and fits irrelevant characteristics (noise) in the data

- Low bias and high variance
- Low training error and high test error

Ways to design classifiers

- Try simple classifiers first
- Use increasingly more powerful classifiers with more training data
- Finding good features: Better to have smart features and simple classifiers than simple features and smart classifiers

Three types of errors

- Inherent noise in data: unavoidable
- Bias: due to over-simplifications
- Variance: due to inability to perfectly estimate parameters from limited data

No classifier is inherently better than any other: we need to make assumptions to generalize

Nx2 Cross Validation

- Randomly split up labeled dataset into 2 parts of equal size
- Use one for training, the second one for testing (validation)
- Swap both sets
- Repeat N times (called folds)
- Analyze the classification errors

Results in different Nx2 classifiers

leave-one-out test

For a small dataset (N), $(N-1)$ data is used for training, 1 for testing

Repeat with randomly chosen datasets ($N-1$) and average the results

Assigning pixels to informational classes : data-driven approach

1. Collect a dataset of images and labels
 2. Use Machine Learning algorithms to train a classifier
 - i. (K)-nearest neighbour classification
 - ii. support vector machines (SVM)
 3. Evaluate the classifier on new images
- Example training set



Learn a function f that generalizes the decision to new (unseen) data

Example dataset: **CIFAR-10**

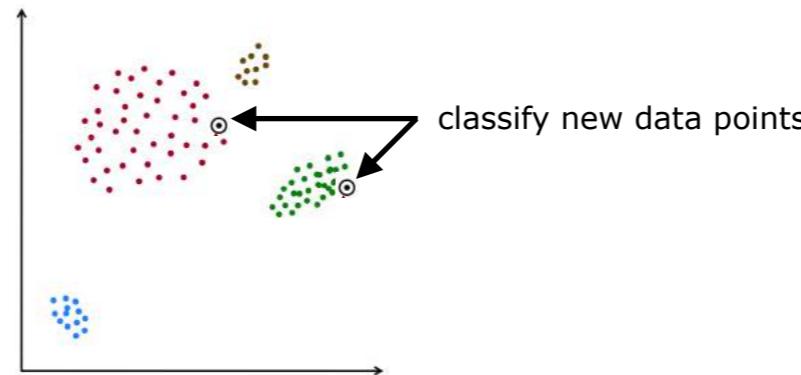
(K)-Nearest neighbour classification

Nearest neighbour classification

- memorise all data and labels
- predict the label of the most similar training image
- feature distribution is modeled by the training data (or a subset)
- class is assigned based on the closest feature in the training data



|  ,  | $\rightarrow \mathbb{R}$



L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

result class of the nearest neighbour training data point in feature space

What is the best distance metric to use?

- problem/dataset-dependent
- try metrics and see what works best

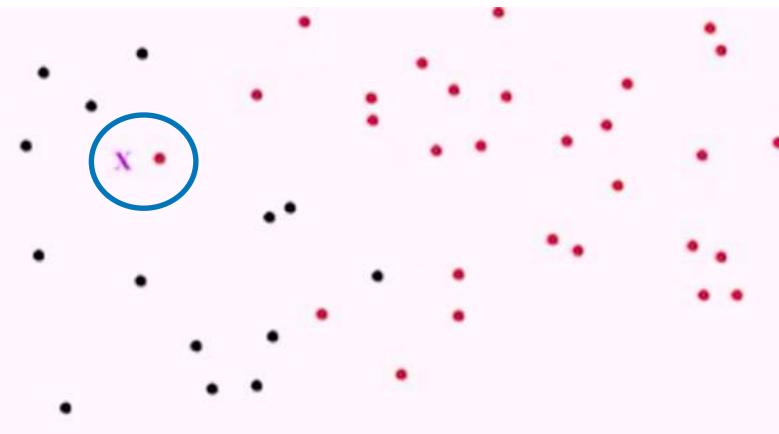
Form a distance matrix

- training data represents the distribution of features directly
- noise in data can cause problems for correct classification
- problematic for classes/features with different variances
- with N examples, how fast are training and prediction? : Train O(1), predict O(N)

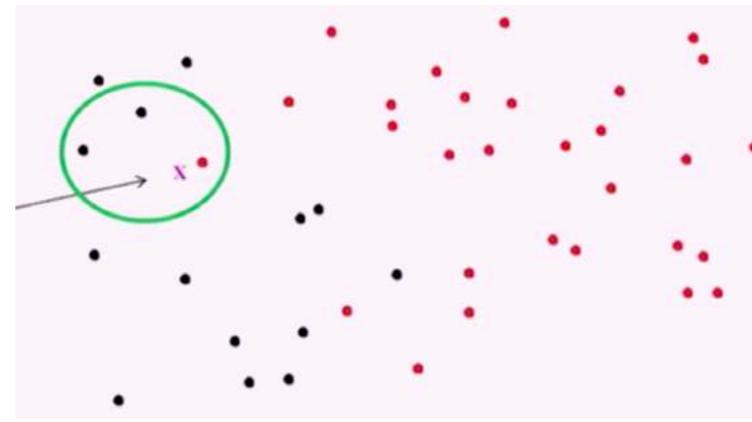
Such a classifier is bad: we want classifiers that are fast at prediction; slow for training is ok

K-Nearest neighbour (KNN) classification

- The k nearest neighbors are considered for the classification decision (majority vote or weighted)
- Robust variant of NN
- K-Nearest Neighbors classifier predicts labels based on nearest training examples
- Choice of k is often done heuristically
 - small k: less robust to noise
 - large k: may consider far away neighbors
- For a pixel to be classified, find the K closest training samples (in terms of feature vector similarity or smallest feature vector distance)
- Among the K samples, find the most frequently occurring class C_m
- Assign the pixel to class C_m



NN classifier: label assigned using one-one distance metric



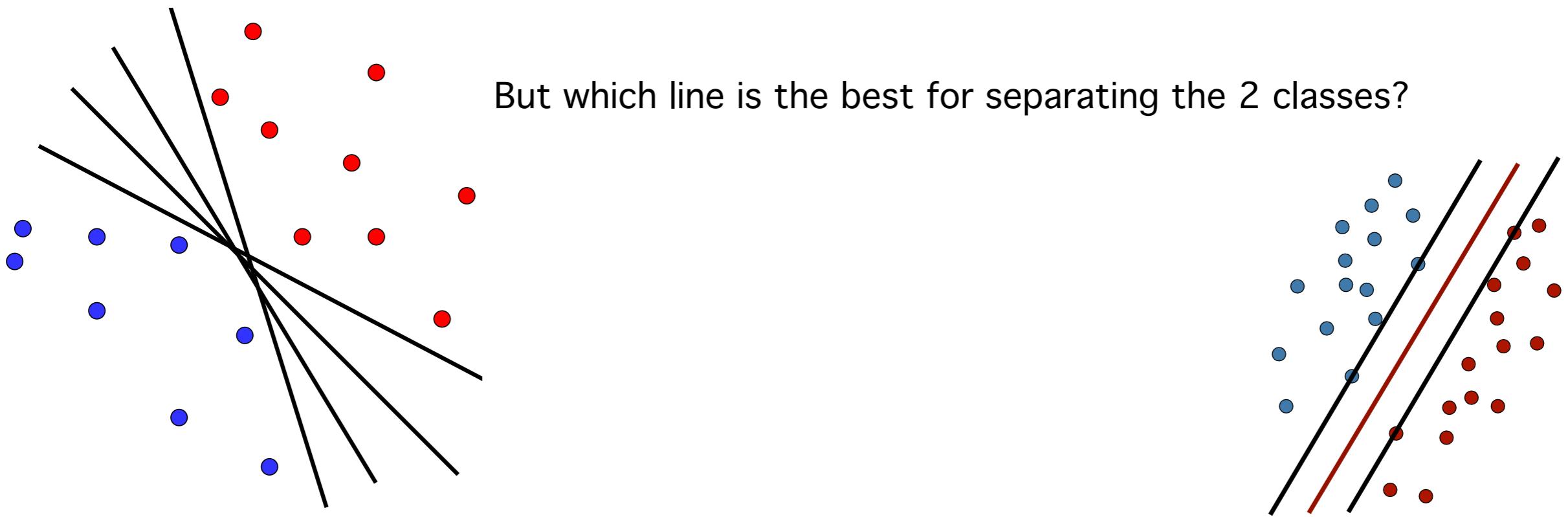
KNN classifier: label assigned by voting

- KNN sensitive to data imbalance.
eg: more red dots than black dots. If the entire set for voting is chosen, then 'x' is mis-classified to 'red' data label
- Circumvented using cross validation and choosing different k-values for each test runs.
- No MODEL for the DATA!
- kNN is a nonlinear classifier
- Memory intensive
- Predictions still take long!

Support vector machines

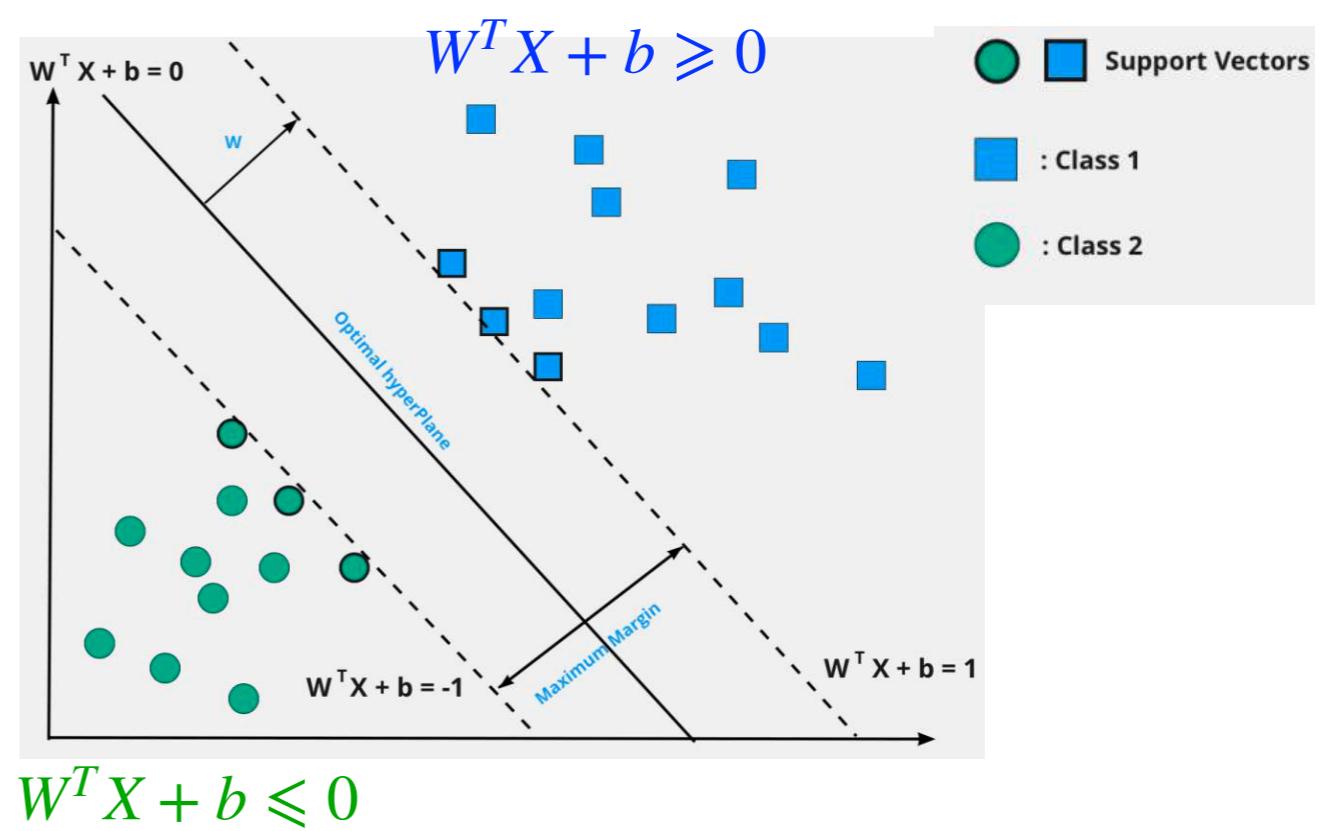
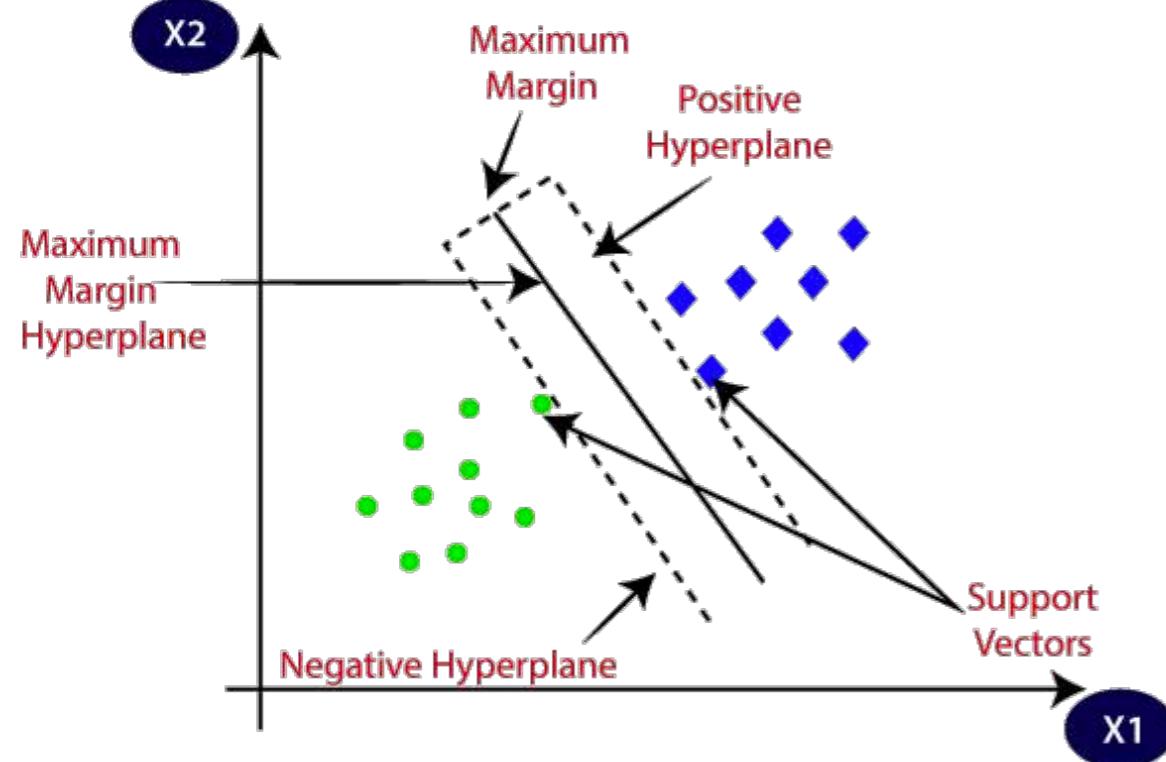
How to select the classifier with the best generalisation performance?

Main idea: select the hyperplane that maximizes the margin between two classes



- SVMs seek to maximize the margin between both classes
- Search for the separating hyperplane is formulated as a convex optimization problem
- Optimal solution for computing the hyperplane

How do we find the best hyperplane?



There is a separation of distances from -1 to $+1$ for all of the samples.

W is a unit vector perpendicular to the hyperplane

We introduce a variable y_i such that

$y_i = +1$ (+ve samples),

$y_i = -1$ (-ve samples)

we multiply:

$y_i (+)$ with $y_i^+ (W^T X + b) \geq 0$

$y_i (-)$ with $y_i^- (W^T X + b) \leq 0$

So, we have a single equation: $y_i (W^T X + b) = 0$

How do we make constraint on the equation $\mathbf{y}_i(\mathbf{W}^T \mathbf{X} + \mathbf{b}) = 0$?

Finding the projection of one vector over the other using dot product and the unit normal vector:

Here one vector is a positive sample (x_+) and the other vector, a negative sample say is (x_-) and

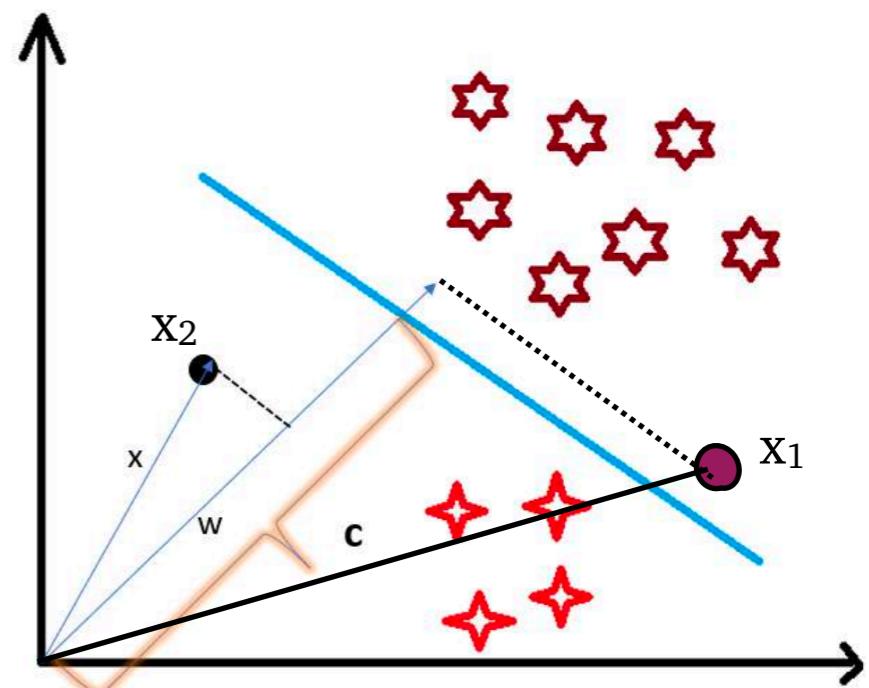
w : a unit vector perpendicular to the hyperplane

we take the dot product of x and w vectors, if

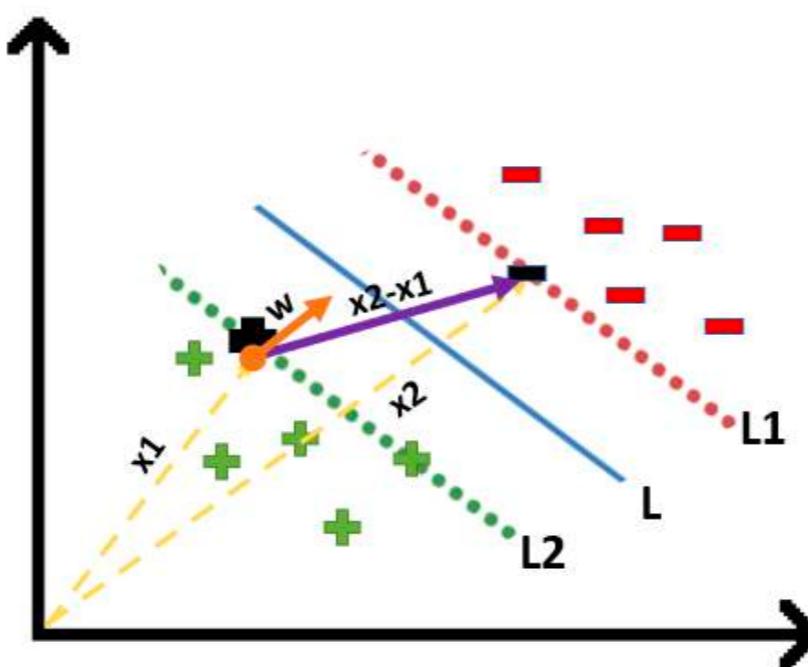
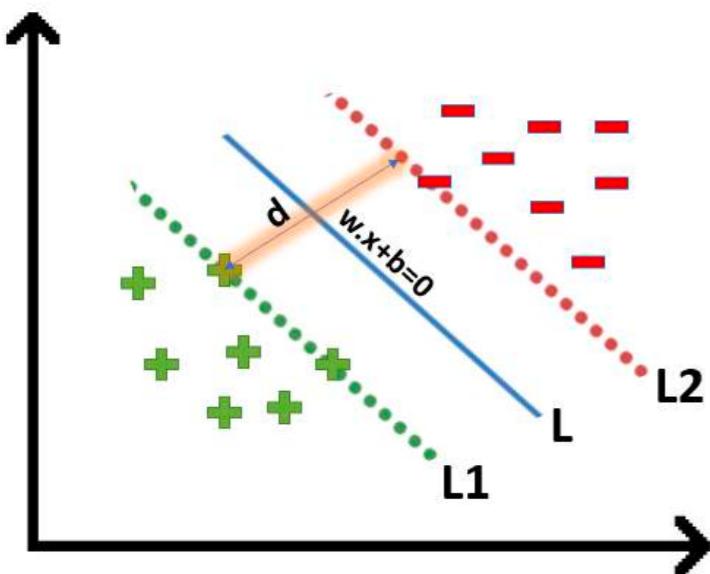
$$\vec{x} \cdot \vec{w} = c \text{ (the point lies on the decision boundary)}$$

$$\vec{x} \cdot \vec{w} > c \text{ (positive samples)}$$

$$\vec{x} \cdot \vec{w} < c \text{ (negative samples)}$$



How do we maximise the distance?



How do we maximise the distance?

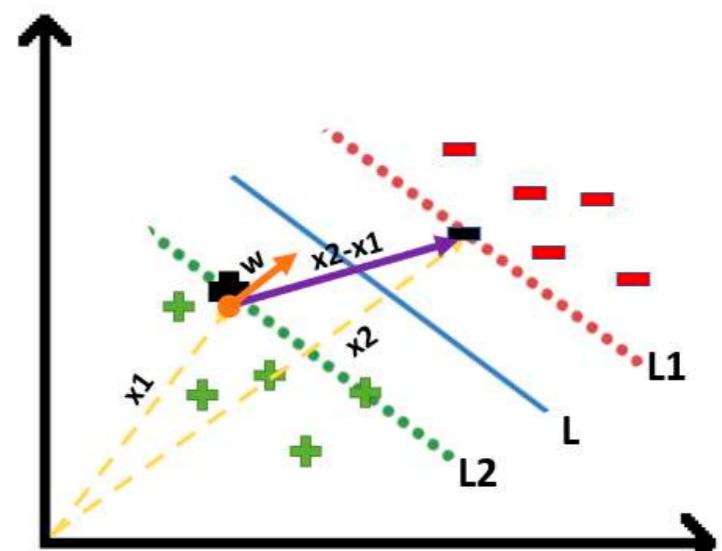
$$\mathbf{y}_i(\mathbf{W}^T \mathbf{X} + \mathbf{b}) = 0$$

When $\mathbf{y}_i = 1$, $\mathbf{y}_i(\vec{\mathbf{w}} \cdot \mathbf{x}_1 + \mathbf{b}) = 1$ $\vec{\mathbf{w}} \cdot \mathbf{x}_1 = 1 - \mathbf{b}$

Similarly $\mathbf{y}_i = -1$, $\mathbf{y}_i(\vec{\mathbf{w}} \cdot \mathbf{x}_2 + \mathbf{b}) = 1$ $\vec{\mathbf{w}} \cdot \mathbf{x}_2 = -1 - \mathbf{b}$

$$(\mathbf{x}_2 - \mathbf{x}_1) \cdot \frac{\vec{\mathbf{w}}}{\|\mathbf{w}\|} = \frac{(1 - \mathbf{b}) - (-1 - \mathbf{b})}{\|\mathbf{w}\|}$$

$$\frac{\mathbf{x}_2 \cdot \vec{\mathbf{w}} - \mathbf{x}_1 \cdot \vec{\mathbf{w}}}{\|\mathbf{w}\|} \longrightarrow \frac{1 - \mathbf{b} + 1 + \mathbf{b}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} = d$$



We want to maximize our width:

$$\frac{2}{\|\mathbf{w}\|}$$

So minimising $\|\mathbf{w}\|$ will maximise the width. Or $\min \frac{1}{2} \|\mathbf{w}\|^2$

This is a non-linear optimisation problem that can be solved using Langrange multiplier λ_i and Karush-Kuhn-Tucker (KKT) conditions:

λ_i is 0 at the point which is a support vector and the corresponding constraint is active.

$\mathbf{w} = \sum_{i=1}^l \lambda_i y_i \mathbf{x}_i$. It is zero at the point which is not a support vector and the corresponding constraint is inactive.

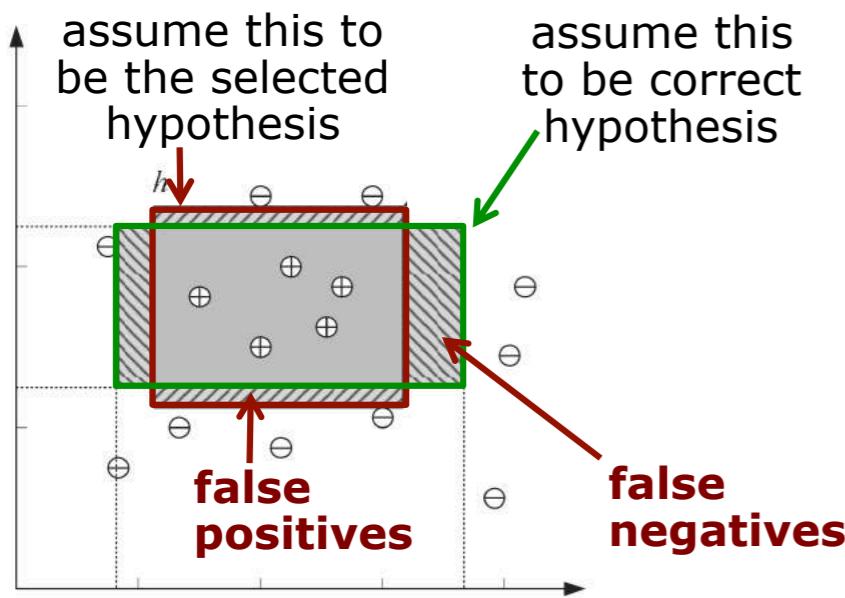
$$\sum_{i=1}^l \lambda_i y_i = 0$$

$$\lambda_i \geq 0 \quad i = 1, \dots, l$$

Solving for b: $\lambda_i(y_i(\mathbf{w} \cdot \mathbf{x}_i - b) - 1) = 0 \quad i = 1, \dots, l$

If we had just two data points we can use substitution methods, but many points and constraints, Langrange multiplier makes it lot more convenient to solve!

Classification Errors



- correctly classified class A as class A: true positive (TP)
i.e. all positive examples classified as positives
- wrongly classified class A as class B: false negative (FN)
i.e. all positive examples classified as negatives
- class B correctly classified as class B: true negative (TN)
i.e. all negative examples classified as negatives
- class B is wrongly classified as class A: false positive(FP)
i.e. all negative examples classified as positives

in reality

	Condition positive	Condition negative	
Test outcome positive	True positive	False positive (Type I error)	Precision = $\frac{\sum \text{True positive}}{\sum \text{Test outcome positive}}$
Test outcome negative	False negative (Type II error)	True negative	Negative predictive value = $\frac{\sum \text{True negative}}{\sum \text{Test outcome negative}}$
Sensitivity = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	Specificity = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Accuracy = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	
(sensitivity is also called recall or true positive rate)	(specificity is also called true negative rate)		

$$\text{false positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Recall: probability that a randomly selected positive case is classified as positive

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

False & True Positive Rate, ROC, F-measure

- False positive rate is the probability that a randomly selected and in reality negative example is classified positive
- True positive rate (=sensitivity, recall) is the probability that a randomly selected, in reality positive example is classified as positive
- Precision is the probability that a randomly selected, positively classified example is positive in reality

$$\text{false positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

true positive
rate / recall /
sensitivity

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

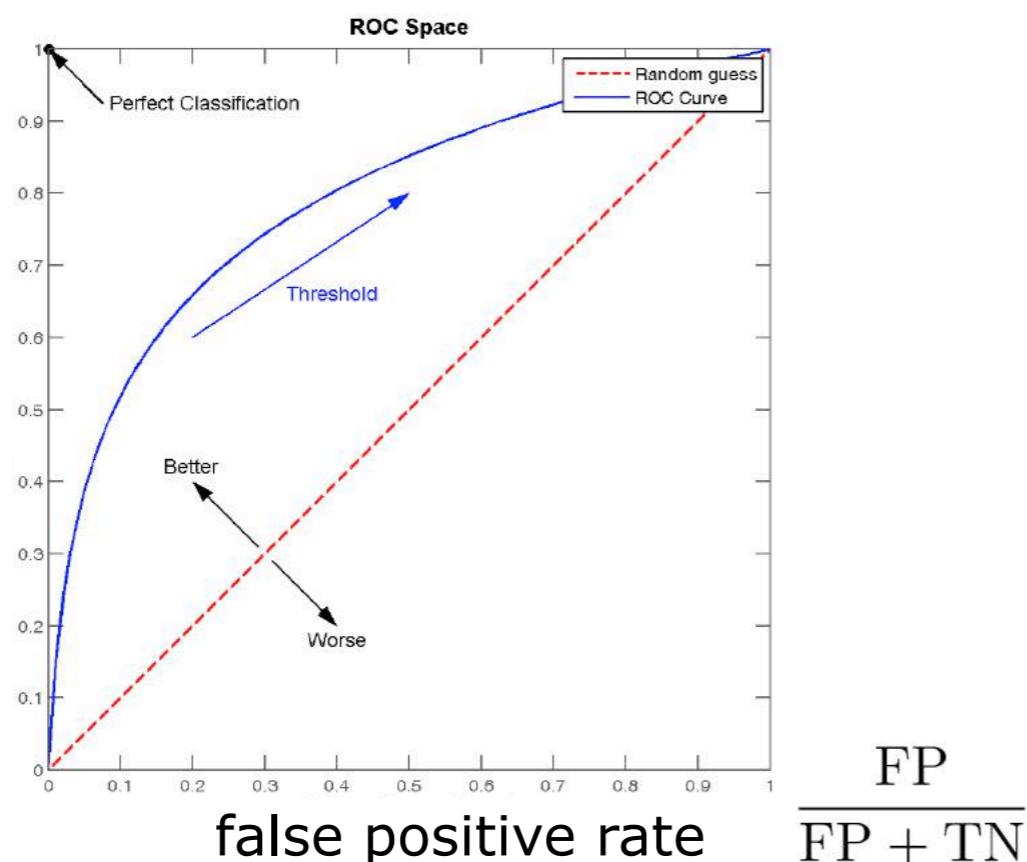


Image courtesy: Wikipedia

F-score / F_1 score / F-measure

- Combines precision and recall into one value (harmonic mean)
- F-score reaches its best value at 1 and its worst score at 0

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Introduction to neural networks

DSE312-CV
Lecture28
Bhavna R

Neural networks in the context of images

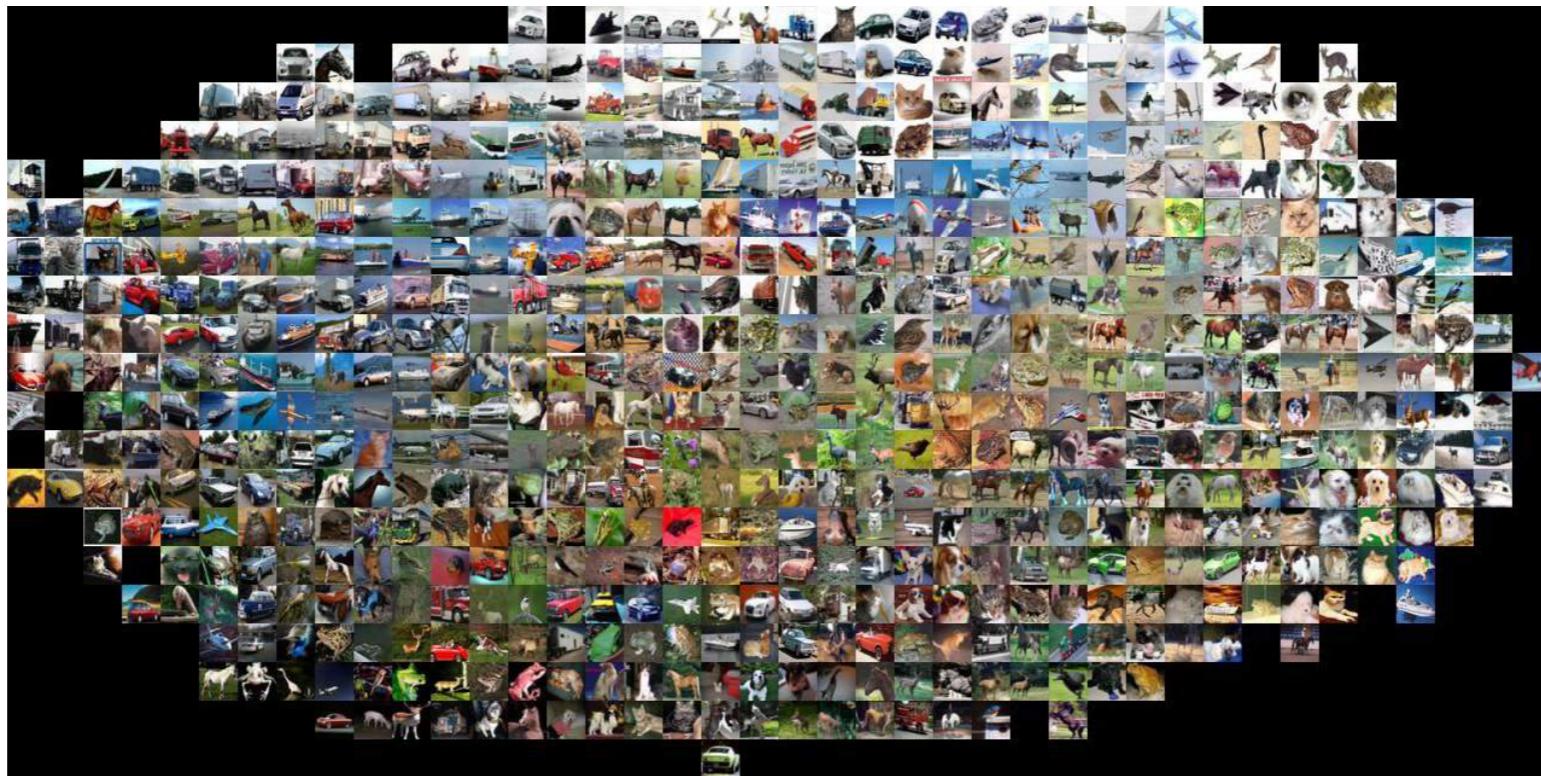


Image classification



(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat



Semantic segmentation



Label each pixel to
form a category or
a set of pixels



What is the idea behind the neural networks?

History

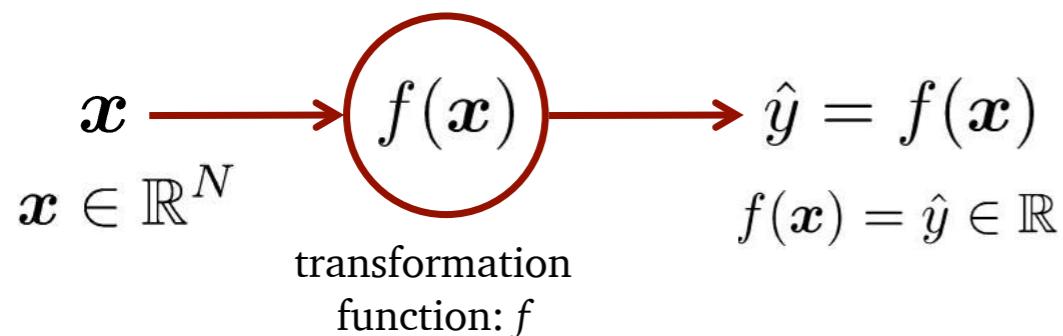
- First ideas discussed in the 1950/60ies
- Theory work on NNs in the 1990ies
- Increase in attention from 2000 on
- Deep learning 2010
- CNNs for image tasks from 2012 onwards

Inspiration

- Biological neurons: fundamental units of the brain
- Receive sensory input from the external world or from other neurons
- Transform and relay signals
- Send signals to other neurons and also motor commands to the muscles

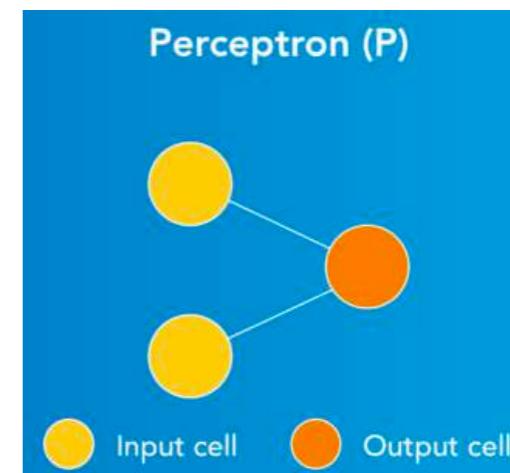
Artificial neurons

- Receive inputs / activations from sensors or other neurons
- Combine / transform information
- Create an output / activation



- nodes are neurons
- edges represent input-output connections of the data flow

I: Perceptron or TLU (threshold logic unit)

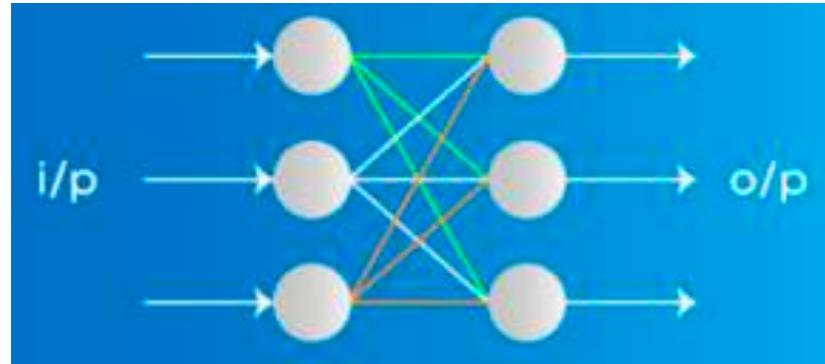


Logic Gates: AND, OR, or NAND

Using weighted inputs and threshold apply activation function to obtain output

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

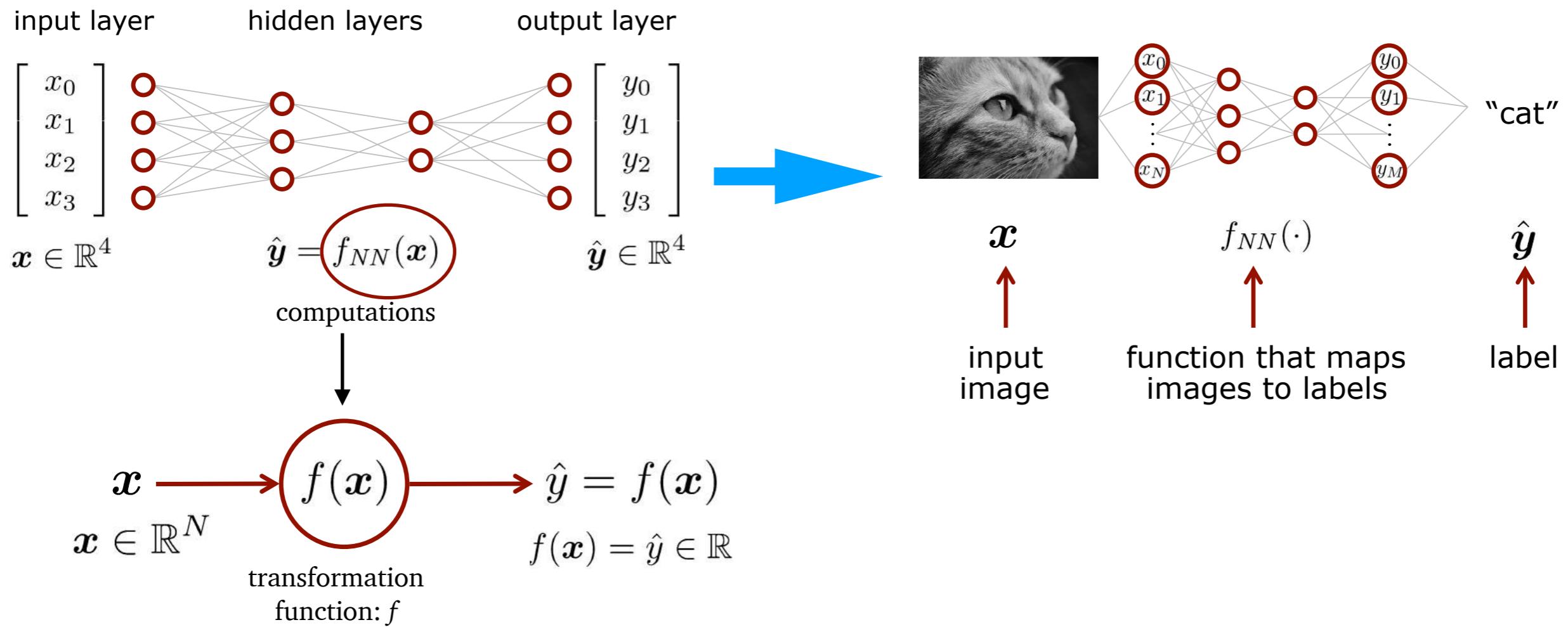
II: Feed Forward Neural Networks



- input data travels in one direction only
- hidden layers may or may not be present, can be as a single-layered or multi-layered feed-forward neural network
- weights are static here
- activation function (above or below threshold) is fed by inputs

III: multi-layered feedforward network

image classification problem



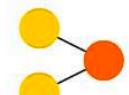
A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

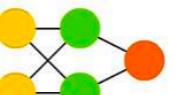
Perceptron (P)



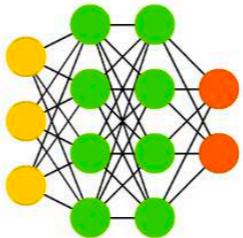
Feed Forward (FF)



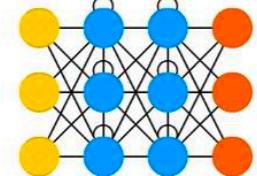
Radial Basis Network (RBF)



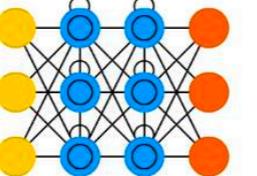
Deep Feed Forward (DFF)



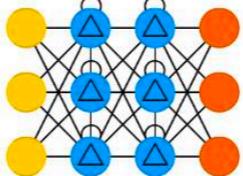
Recurrent Neural Network (RNN)



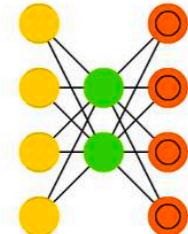
Long / Short Term Memory (LSTM)



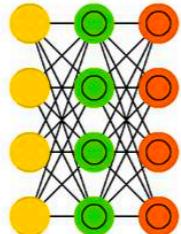
Gated Recurrent Unit (GRU)



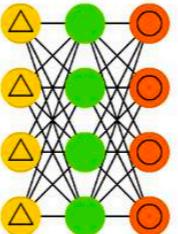
Auto Encoder (AE)



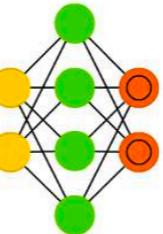
Variational AE (VAE)



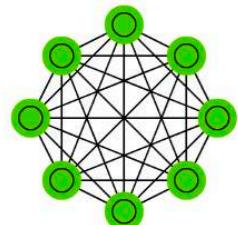
Denoising AE (DAE)



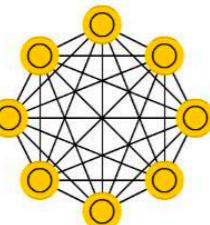
Sparse AE (SAE)



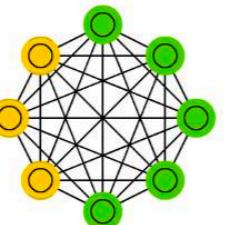
Markov Chain (MC)



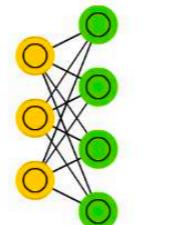
Hopfield Network (HN)



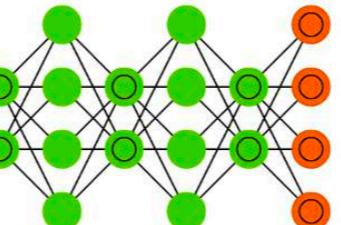
Boltzmann Machine (BM)



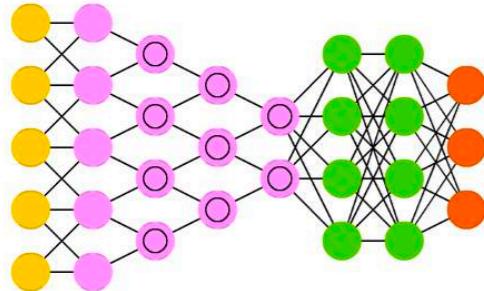
Restricted BM (RBM)



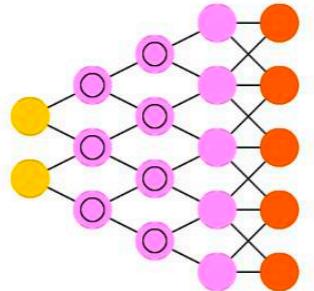
Deep Belief Network (DBN)



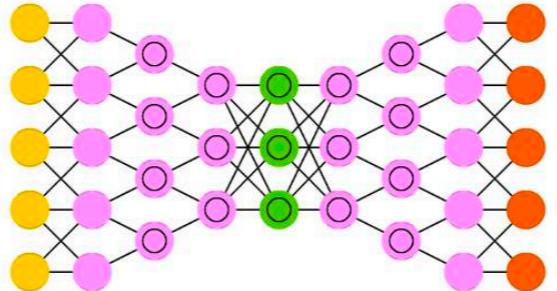
Deep Convolutional Network (DCN)



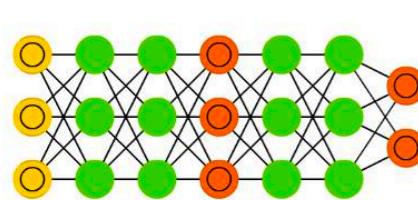
Deconvolutional Network (DN)



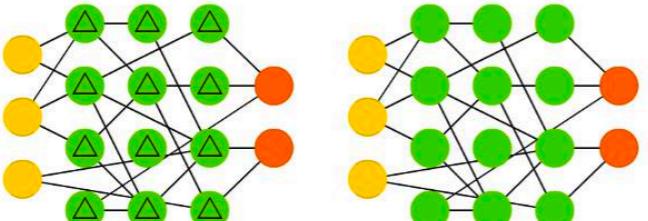
Deep Convolutional Inverse Graphics Network (DCIGN)



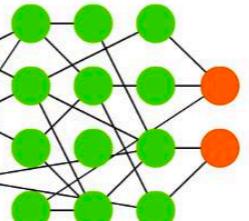
Generative Adversarial Network (GAN)



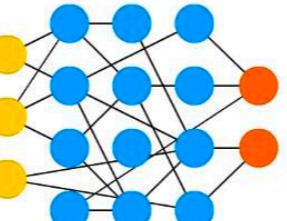
Liquid State Machine (LSM)



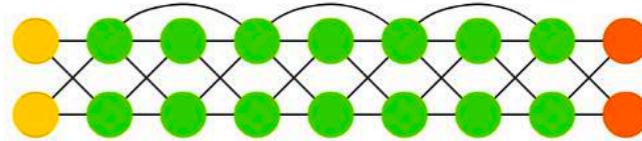
Extreme Learning Machine (ELM)



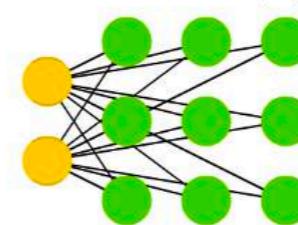
Echo State Network (ESN)



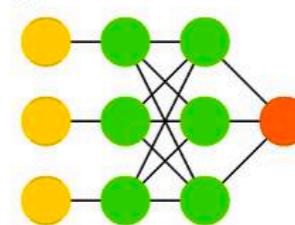
Deep Residual Network (DRN)



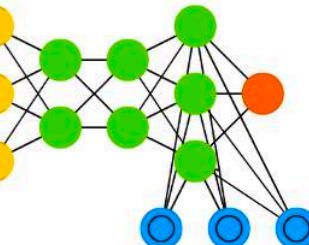
Kohonen Network (KN)



Support Vector Machine (SVM)



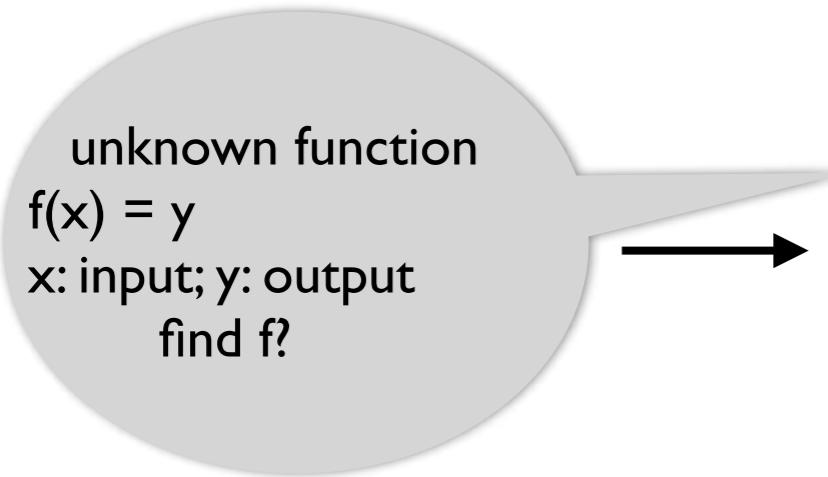
Neural Turing Machine (NTM)



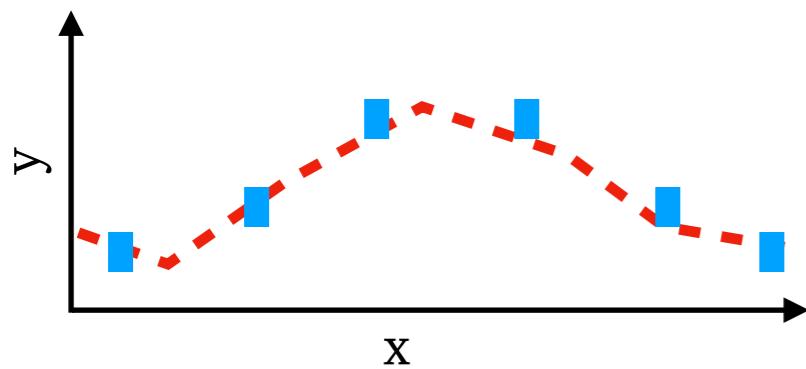
[Image courtesy: van Veen]

Neural networks: composition of functions

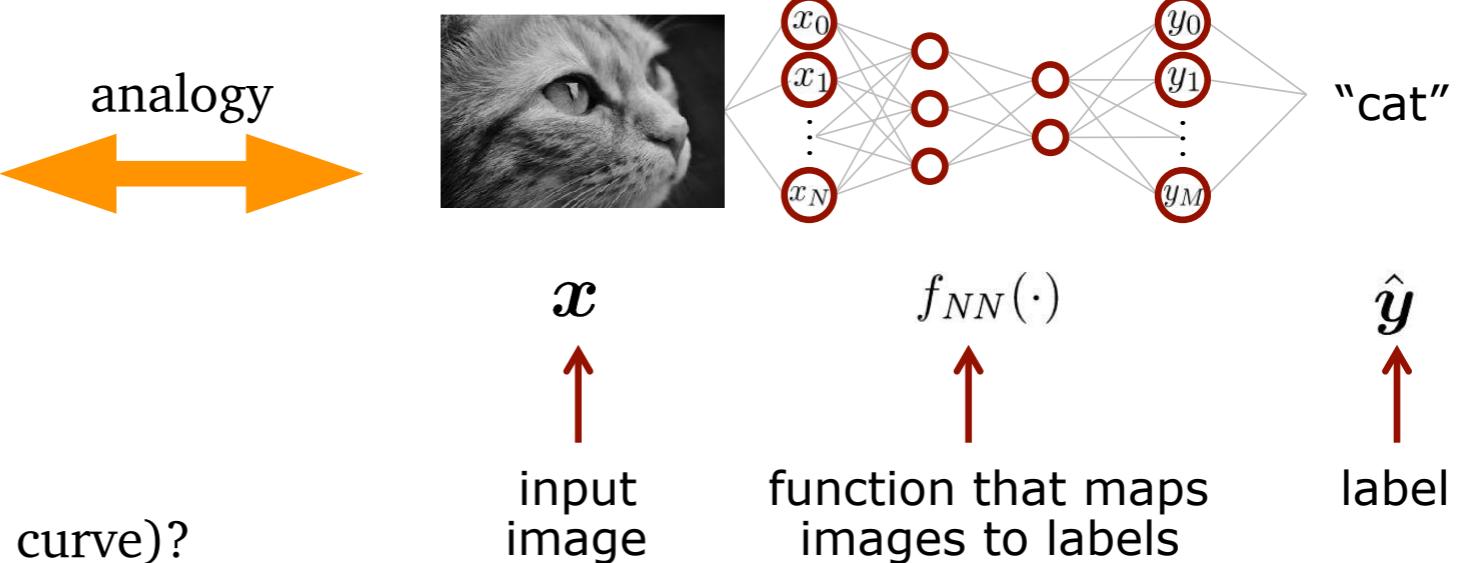
learn useful **representations** or **features** directly from **images**, text, sound



- a “universal approximator”
- Maps input to output assuming they are related at all (by correlation or causation)
 - in the process of learning, a neural network finds the right f , or the correct manner of transforming x into y



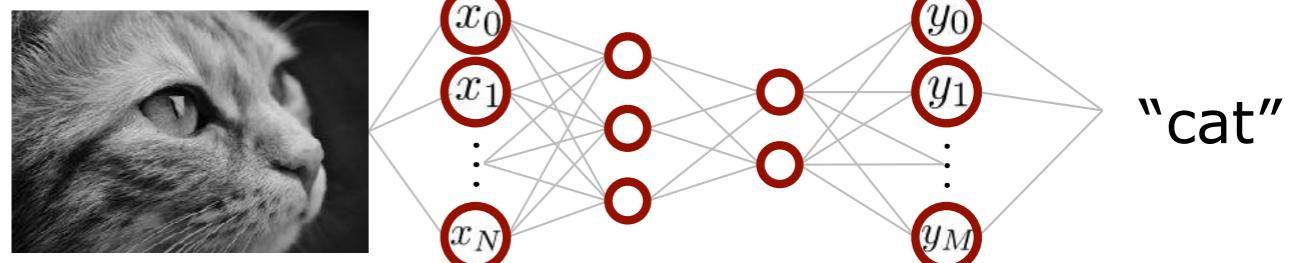
analogy



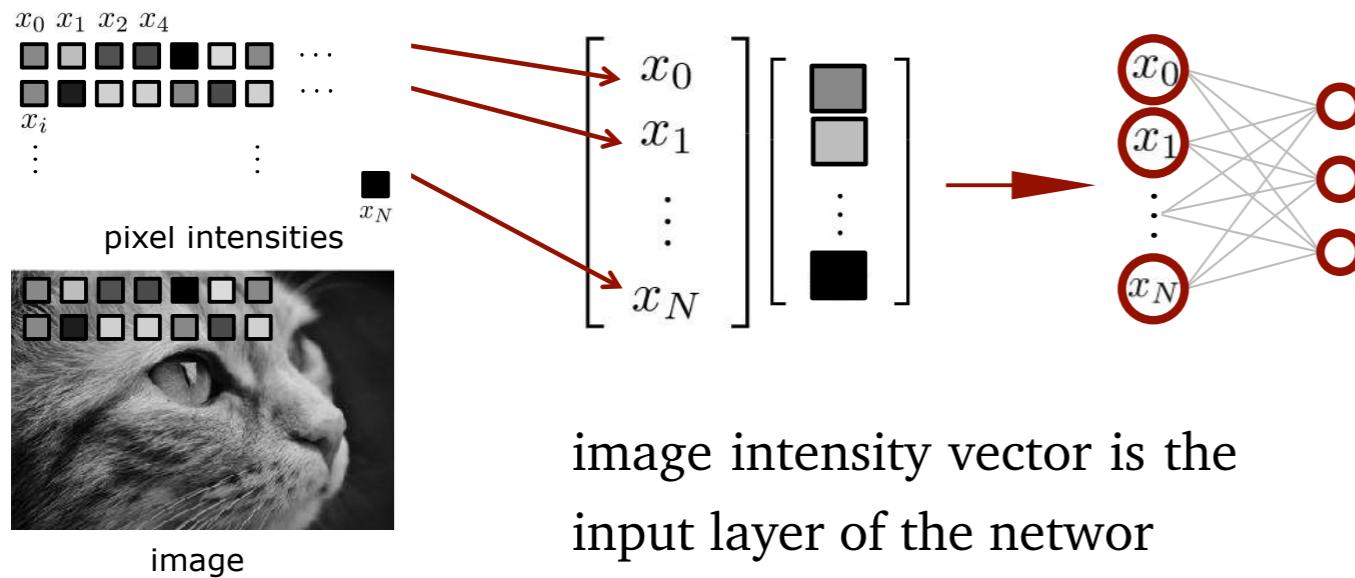
In simple words,
given x & y (blue data points), what is f (red curve)?

Too few points would not get the right f => deep learning requires large training dataset

How is the image classification task solved?



Input layer of the network



An image consists of individual pixels that stores an intensity value. We have $N+1$ such intensity values

Output layer of the network

\hat{y}

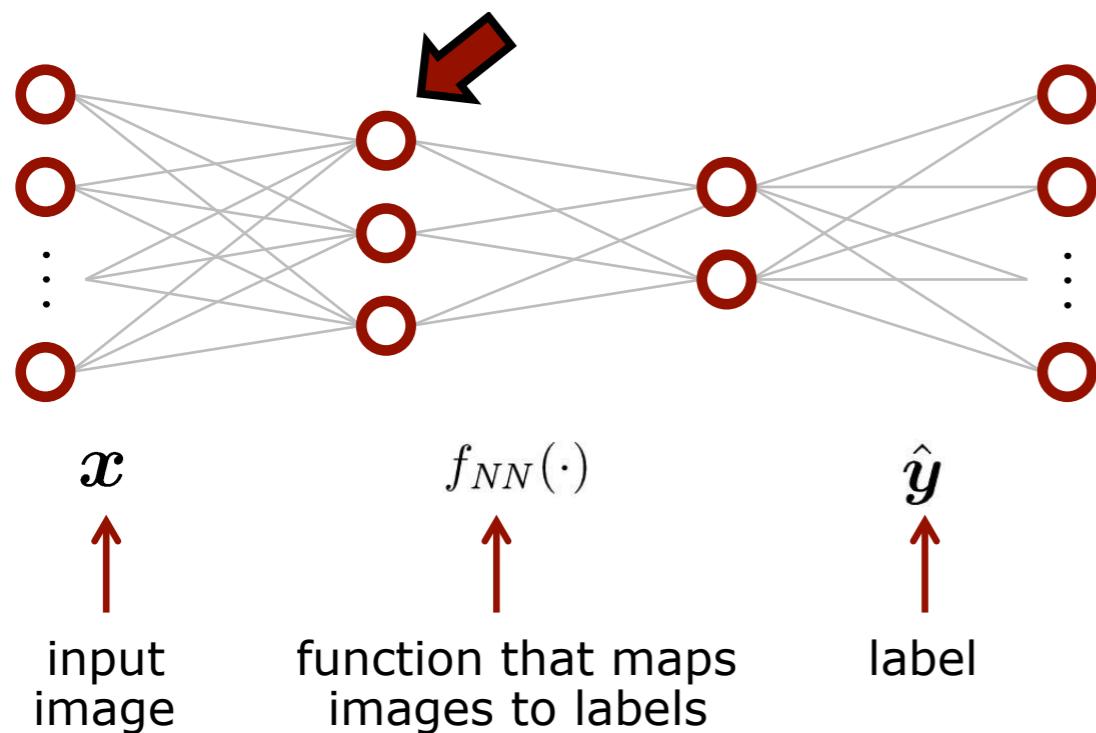
$$\begin{matrix} & y_0 & y_1 & \dots & y_M \\ \hat{y} = & \left[\begin{array}{c} y_0 \\ y_1 \\ \vdots \\ y_M \end{array} \right] & \left(\begin{array}{c} 0.98 \\ 0.01 \\ 0.001 \\ \vdots \end{array} \right) & \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ \vdots \end{array} \right] \end{matrix}$$

vector with an activation/
likelihood for the possible labels

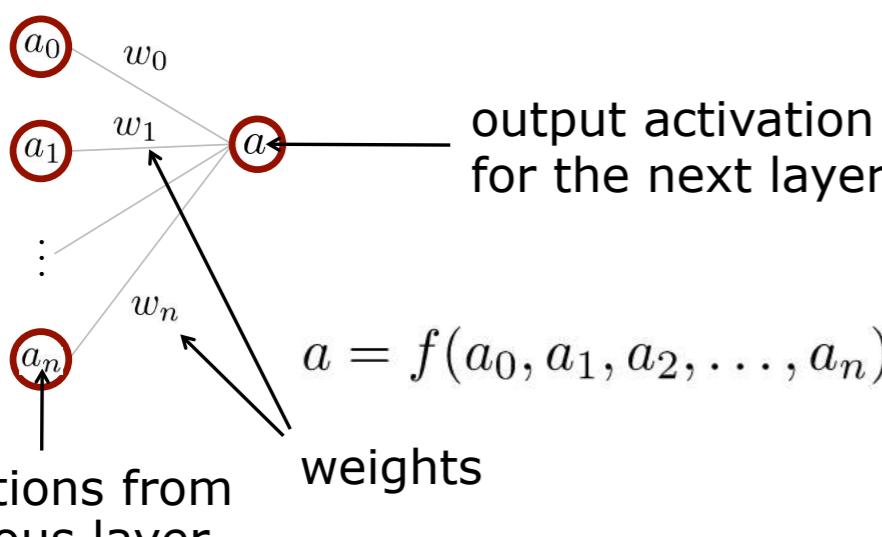
Label vector

Labels are predicted with an uncertainty

How about the functions within the intermediate layer?



If we pick a single neuron,
what does that function look like?



(input) activations a_i
weights w_i
bias b
activation function $\sigma(\cdot)$
output activation a

A neuron (a) gets activated through

- weighted sum of input activations: w_i, a_i
- bias activation, b
- An activation function $\sigma(\cdot)$

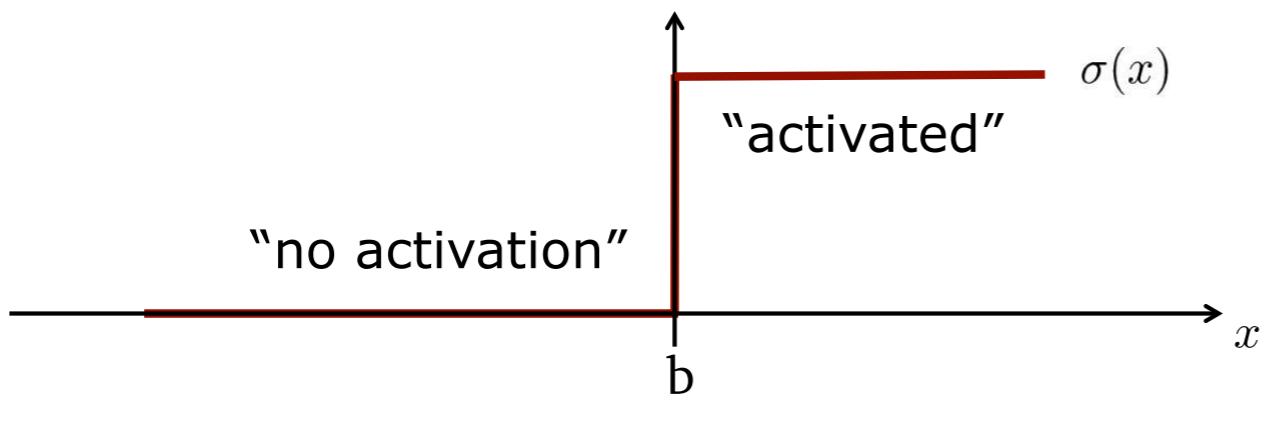
$$a = \sigma(w_0a_0 + w_1a_1 + \dots + w_na_n + b)$$

In addition, we have
activation ($\sigma(\cdot)$) and bias (b)

Sum of products: similar to convolution

Activation $\sigma(x)$ and bias (b)

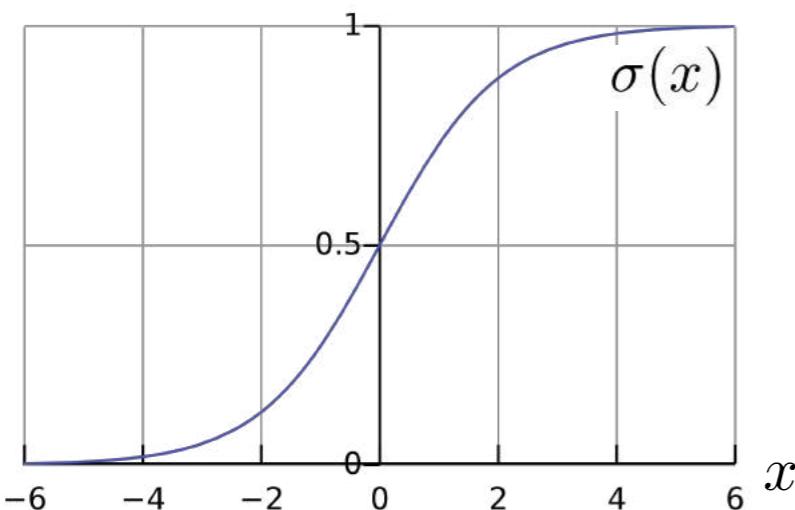
Step function: as an activation function



- neurons are either active or not active
- bias tells us where the activation happens
- we can visualise this as a step function:

$$a = \begin{cases} 0 & \sum_i w_i a_i \leq -b \\ 1 & \text{otherwise} \end{cases}$$

Sigmoid activation function



Sigmoid smoothing function (logistic)
squeezes values between [0,1]

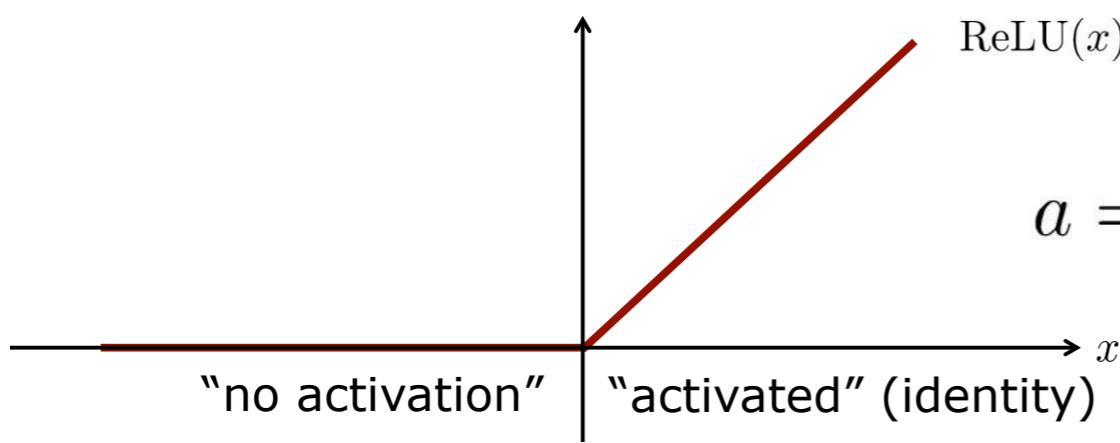
$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad \text{where, } x = -\sum_j w_j a_j - b$$

For very large positive $x \Rightarrow e^{-x} \approx 0, \sigma(x) \approx 1$.

For very large negative $x \Rightarrow e^{-x} \rightarrow \infty, \sigma(x) \approx 0$

For modest values, sigmoid regime

Rectified linear unit (ReLU) activation function



A neuron is only activated if $x > 0$

$$a = \text{ReLU}(w_0 a_0 + w_1 a_1 + \dots + w_n a_n + b) > 0$$

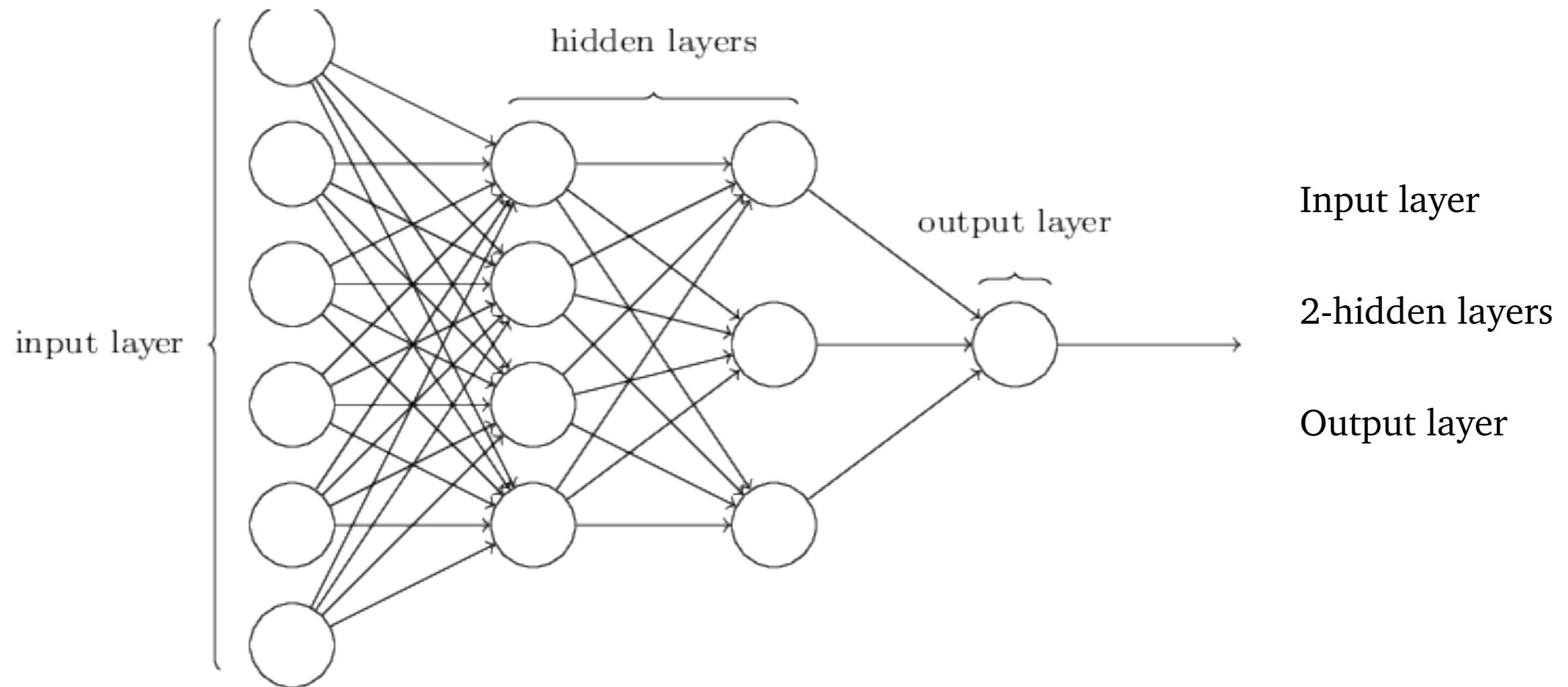
Or $a > -b$

Common activation functions within the neuron

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a. k. a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU) ^[2]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

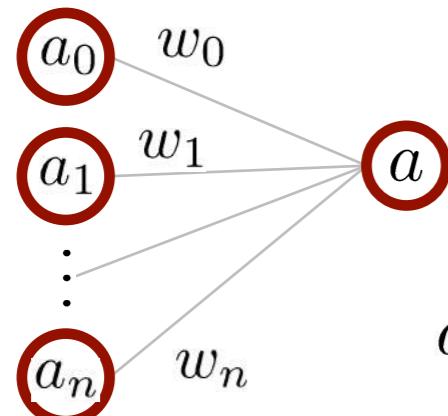
[Courtesy of S. Sharma]

Architecture of neural networks



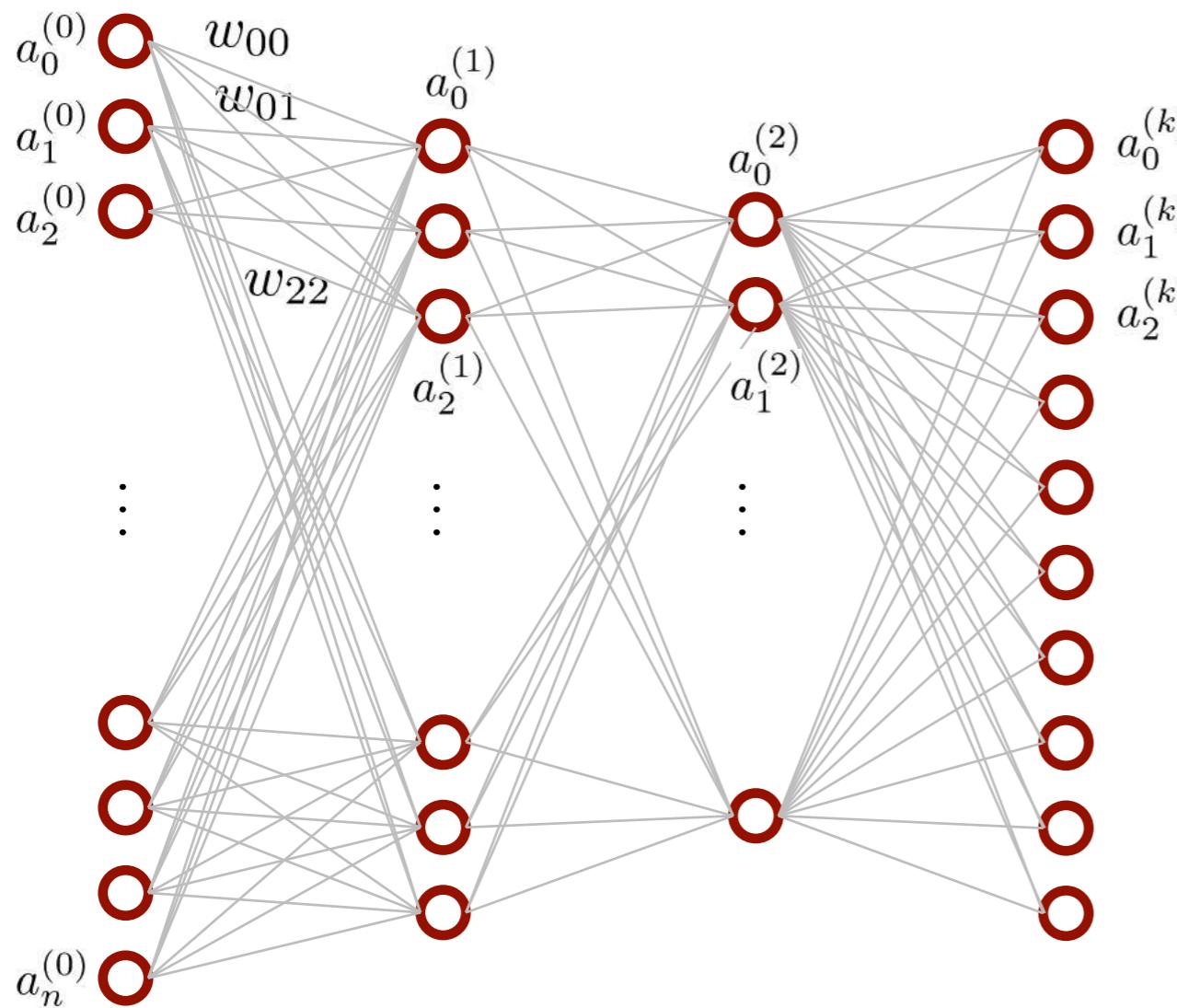
- Such multiple layer networks are sometimes called *multilayer perceptrons* or *MLPs*, despite being made up of sigmoid neurons, not perceptrons.
- We will use the term “feedforward” networks
- neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behaviour they want out of their nets.
- experiments with different number of neurons within the hidden layers (here shown 4,3).

Neuron get activated if the weighted sum of input activations is large enough ($>-b$)



For 1 neuron

$$a = \sigma(w_0a_0 + w_1a_1 + \dots + w_na_n + b)$$

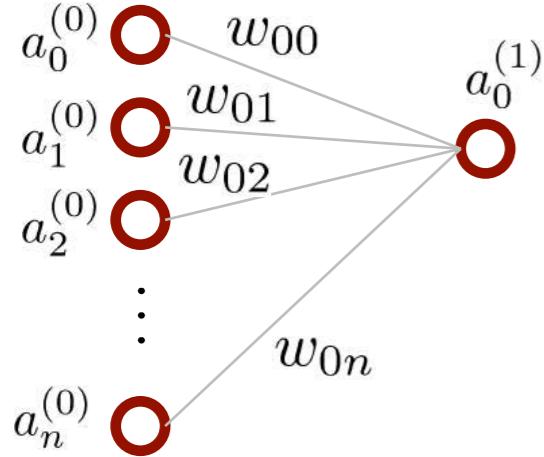


Same way computations are performed for each neuron within the network
There are lots of neurons!

How do we visualise this network?



Matrix notations



$$a_0^{(1)} = \sigma \left(w_{00}a_0^{(0)} + w_{01}a_1^{(0)} + \dots + w_{0n}a_n^{(0)} + b_0^{(1)} \right)$$

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ w_{10} & w_{11} & \dots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k0} & w_{k1} & \dots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \right)$$

layer 1

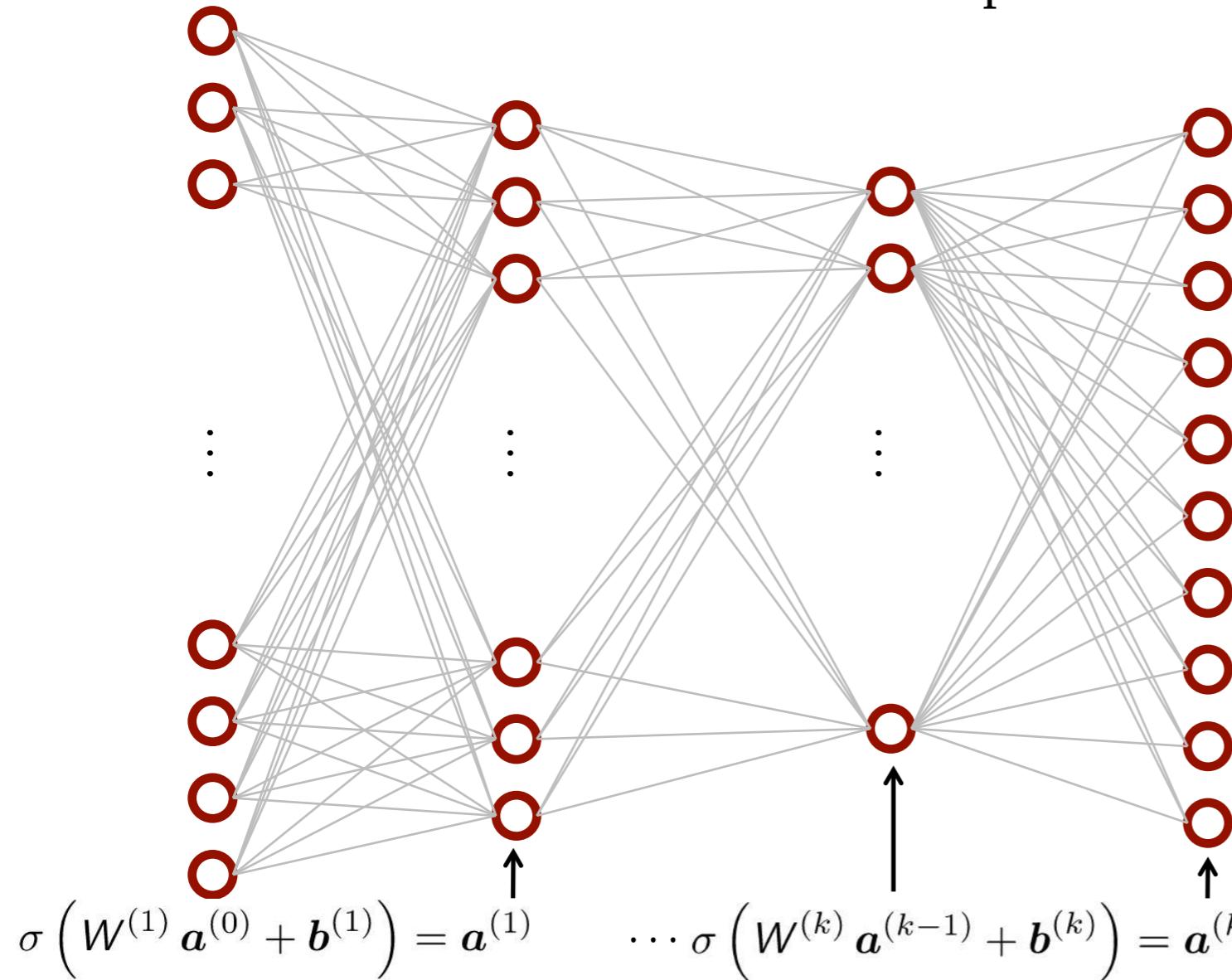
$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ w_{10} & w_{11} & \dots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k0} & w_{k1} & \dots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \right)$$

layer 0

$$a^{(1)} = \sigma \left(W a^{(0)} + b^{(1)} \right)$$

Similarly, layer-2 can be expressed in the same way, so we can build this layer-by-layer

Perform the computations layer by layer



input = layer 0 $x = \mathbf{a}^{(0)}$

layer 1

$$\sigma \left(W^{(1)} \mathbf{a}^{(0)} + \mathbf{b}^{(1)} \right) = \mathbf{a}^{(1)}$$

layer 2

$$\sigma \left(W^{(2)} \mathbf{a}^{(1)} + \mathbf{b}^{(2)} \right) = \mathbf{a}^{(2)}$$

\vdots

layer k = **output**

$$\sigma \left(W^{(k)} \mathbf{a}^{(k-1)} + \mathbf{b}^{(k)} \right) = \mathbf{a}^{(k)} = \hat{\mathbf{y}}$$

Summary

- Here we took an example of a feed forward network
- Information flow is from left to right in such a network
- There are no feedback loops (here)
- Such networks can be used for processing sequence data,
eg: recurrent neural networks (RNN)

Neural networks-part2

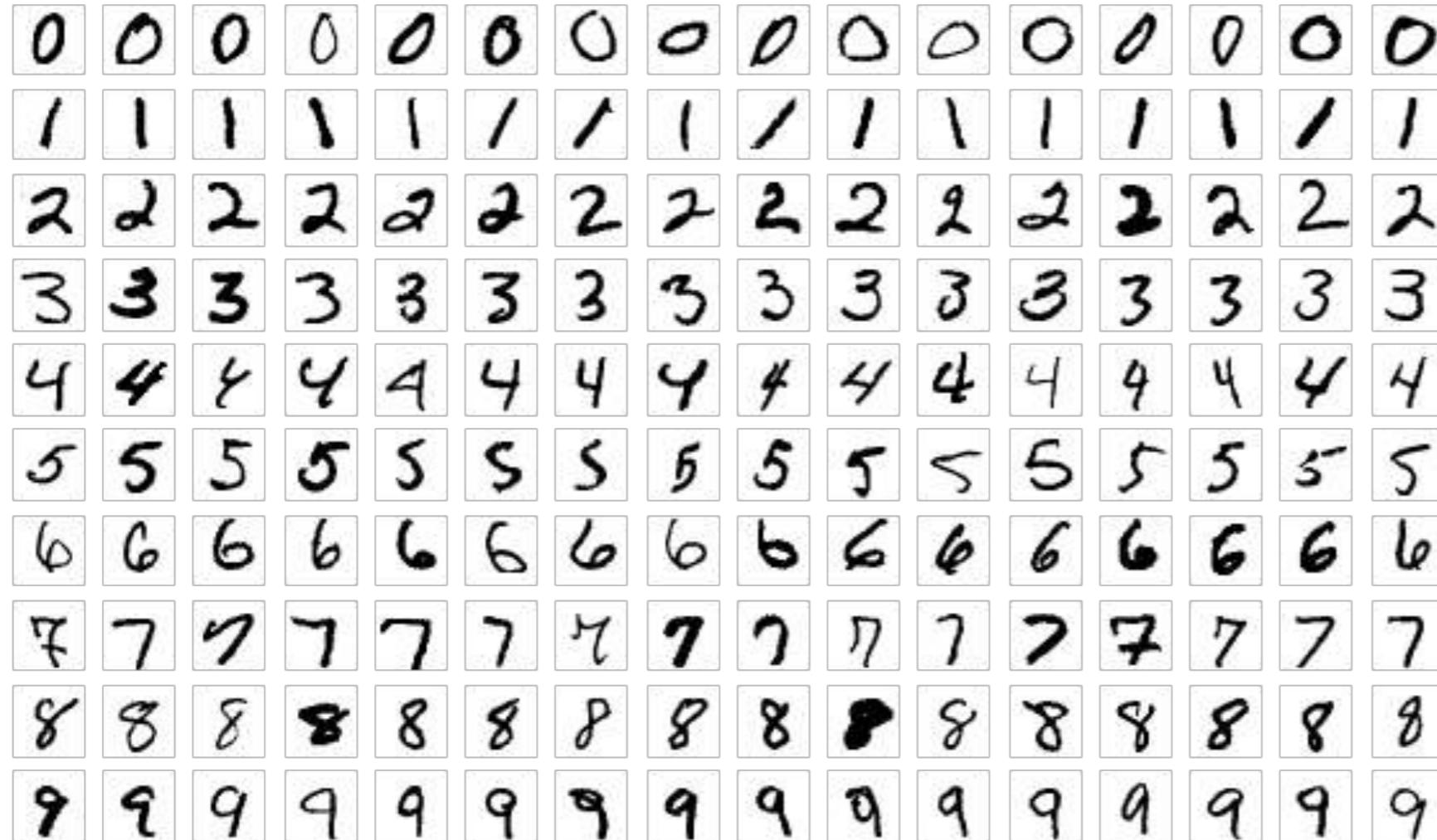
DSE312-CV

Lecture29

Bhavna R

How can a basic *feedforward* neural network
be used for digit recognition?

Handwritten Digit Recognition



A challenging task for a computer program to correctly break up the image

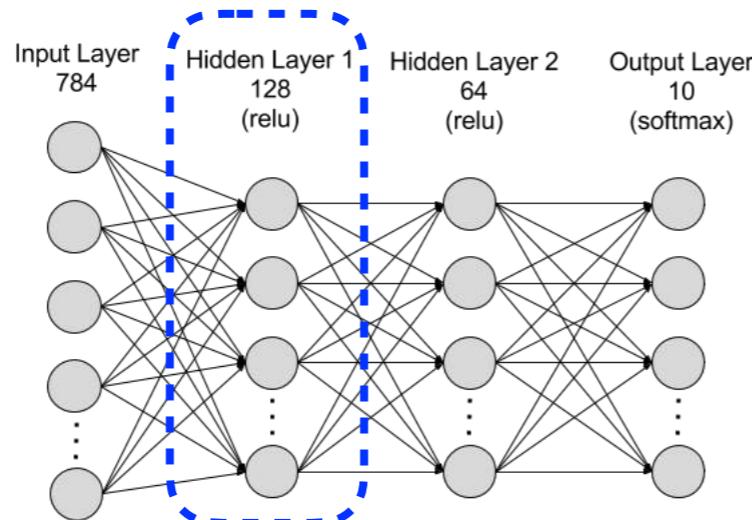
[Image courtesy: Nielsen/Lecun]

MNIST dataset contains tens of thousands of scanned images of handwritten digits, together with their correct classifications.

Computations within the hidden layers of neural networks



**28x28 pixel input images
(784 dim)**
(grayscale images)

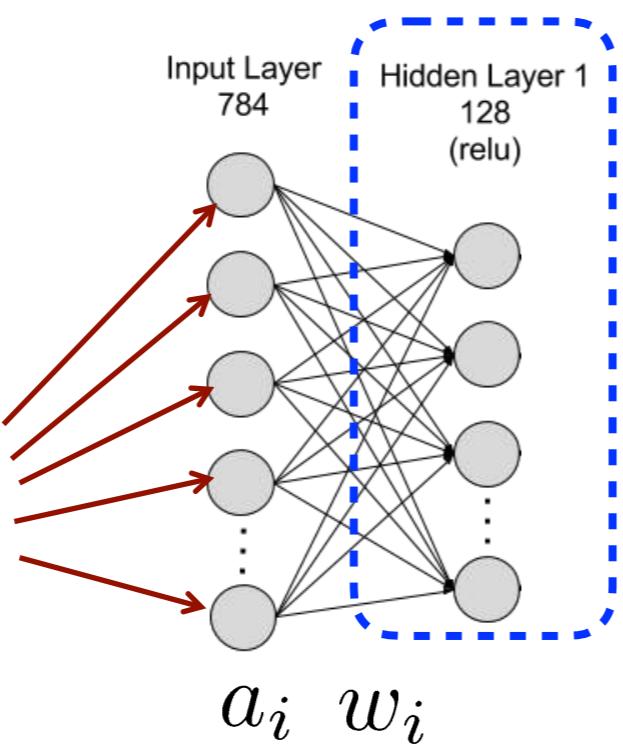


$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

**output vector
(10 dim)**



pixel values



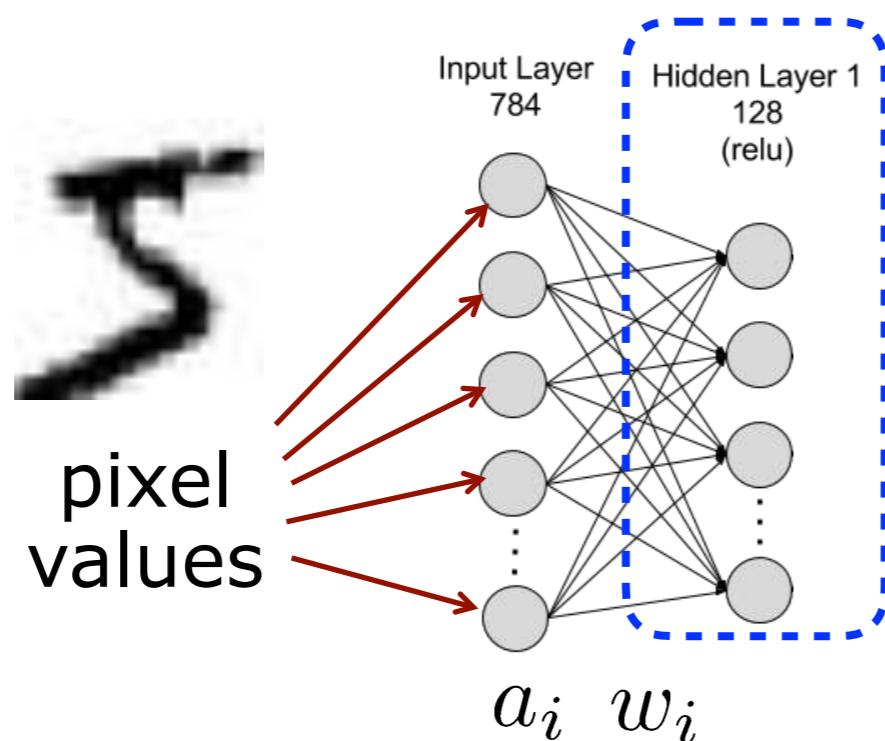
784 input activations (pixel intensities)

Hidden layer 1: 784 weights (i.e. weights
for pixel intensities)

784 input activations = pixel intensities

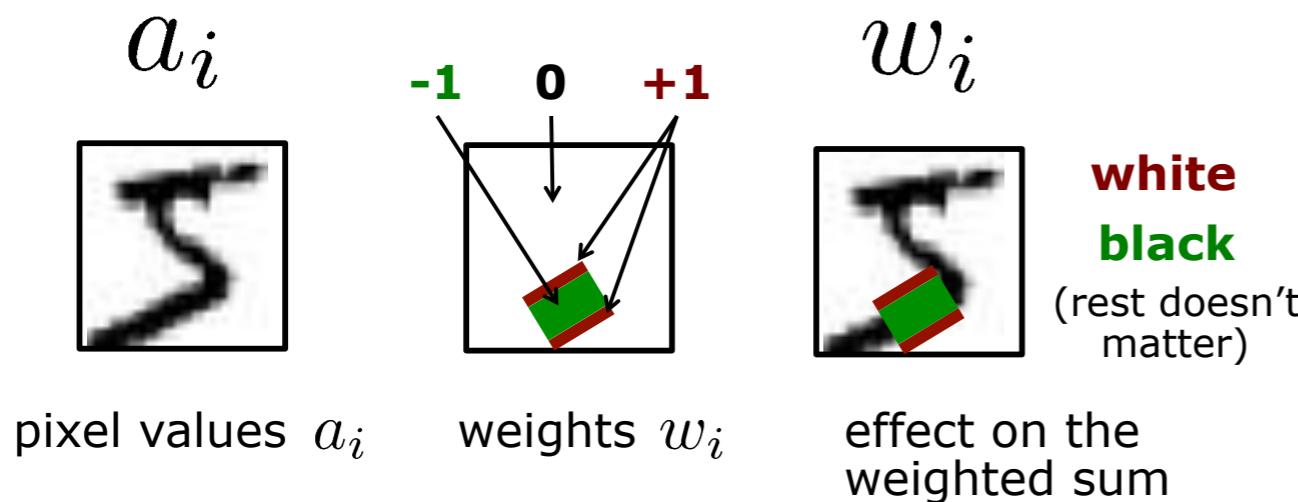
784 weights = weights for pixel intensities

What happens in the 1st hidden layer?



784 input activations = pixel intensities

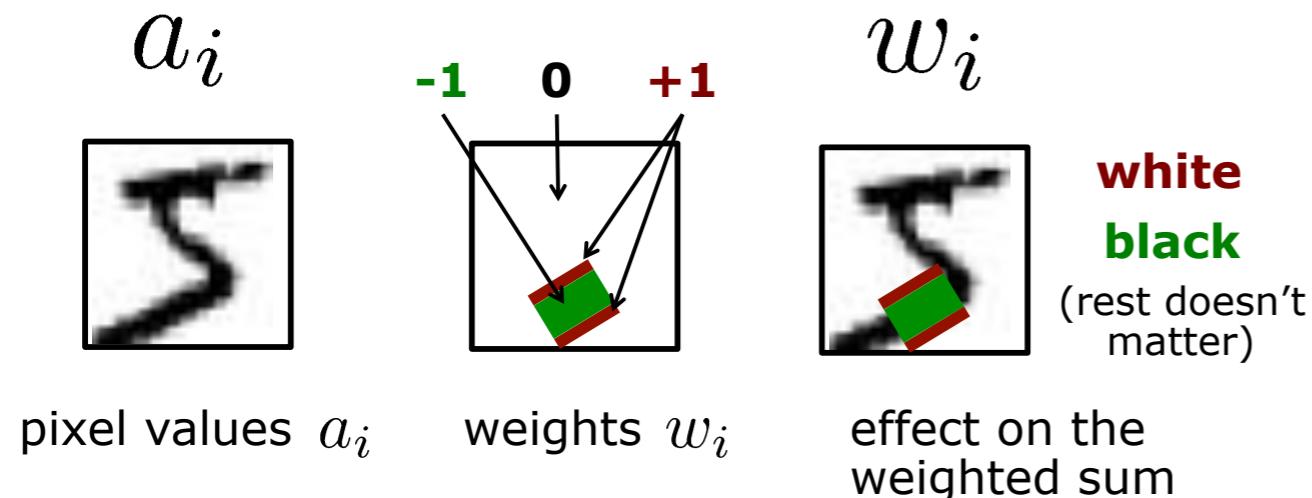
784 weights = weights for pixel intensities



We are heavily weighting input pixels which overlap with the input image, and only lightly weighting the other inputs

weights tell us what matters for activating the neuron!
each weight, 'weights' (used as a verb here) a pattern
within the image

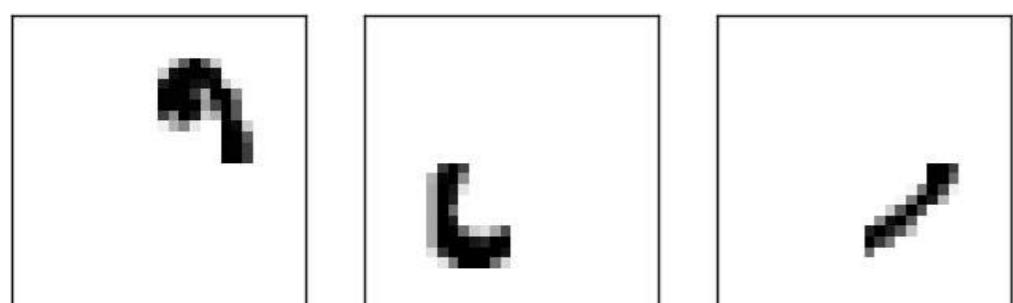
Weight and bias used for finding the pattern within the image



- Weights define the patterns to look for in the image
- Bias tells us how well the image must match the pattern (how strongly should the weight matter?)

weights tell us what matters for activating the neuron!
each weight, ‘weights’ (used as a verb here) a pattern
within the image

Activation functions “switches the neuron on” if it matches the pattern



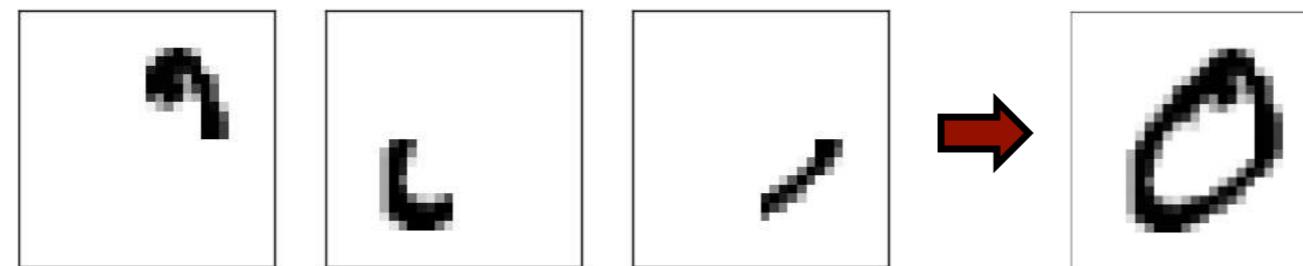
individual “weight images” for a neuron
support individual patterns in the image

The outputs of the 1st layer



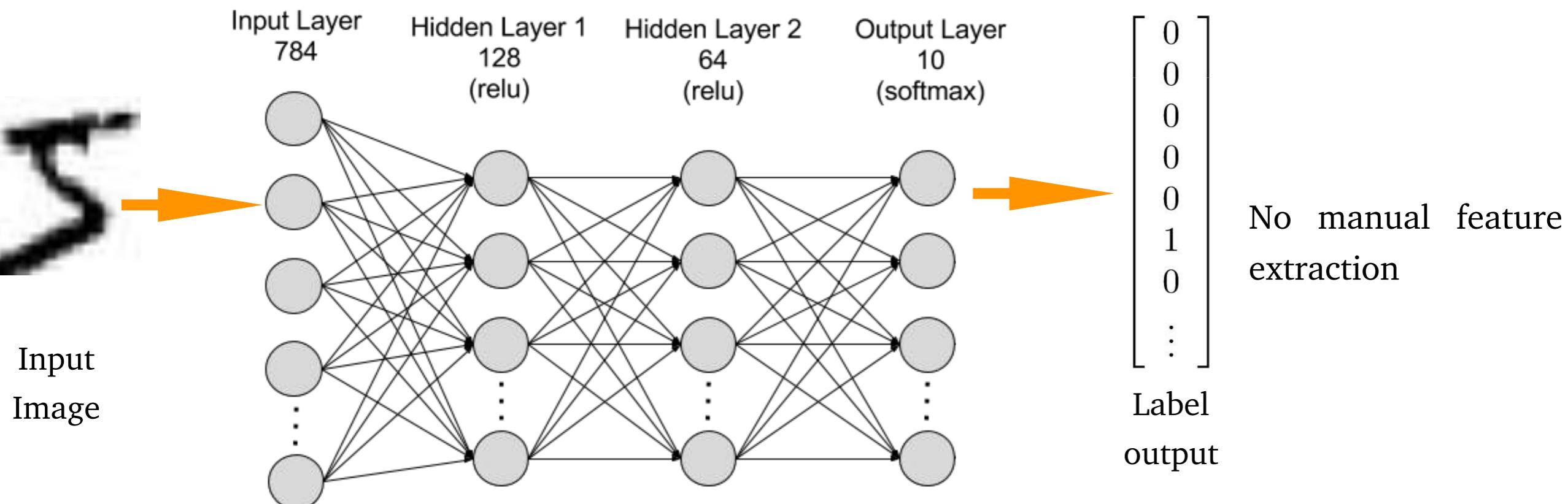
Fed into the 2nd layer

What happens in
the 2nd layer?



- The weights in layer 2 tell us which 1st layer patterns should be combined
- With more layers (depth), more patterns get arranged and combined

The last layer decides, which final patterns make up a digit



A simple feedforward achieves a classification accuracy of ~96% on the digits dataset

We understood what the weights and biases are doing within the network
& how they recognise patterns within the image

But how do we make the network compute specific things?

OR

Design to perform specific tasks

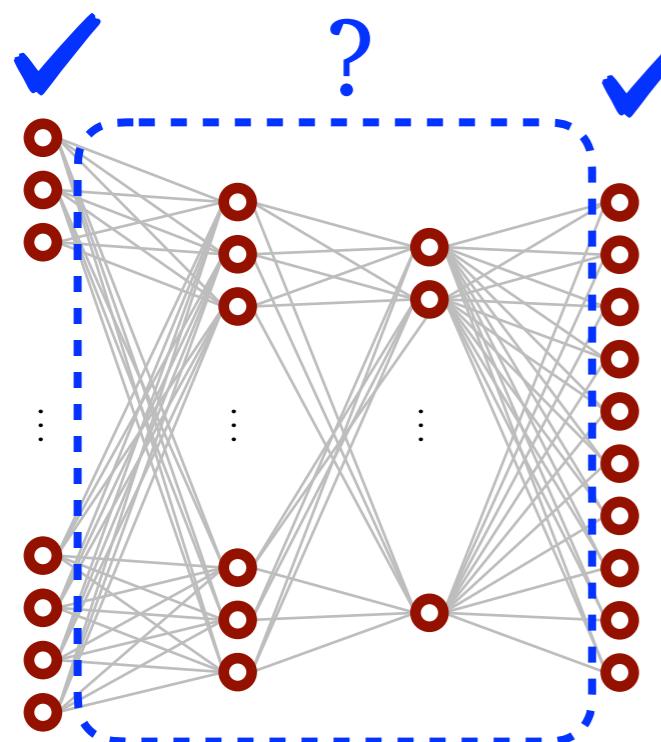
what are the parameters and how do we set them?

“Learning”

How does the neural network achieve learning?

Learning the “right parameters” or correct weights and bias

- Given a network structure, weights and biases tell the network what to do
- Each node within the network is a functions
- The weights and biases determine what this function computes
- Learning = determining parameters so that the network does what we want
- Parameters are estimated by providing labeled examples (training data)



$$\begin{aligned} \text{input} & \downarrow \\ \sigma \left(W^{(1)} a^{(0)} + b^{(1)} \right) &= a^{(1)} \\ \sigma \left(W^{(2)} a^{(1)} + b^{(2)} \right) &= a^{(2)} \\ \vdots & \\ \sigma \left(W^{(k)} a^{(k-1)} + b^{(k)} \right) &= a^{(k)} \end{aligned}$$

weights biases output

Network learns weights and bias

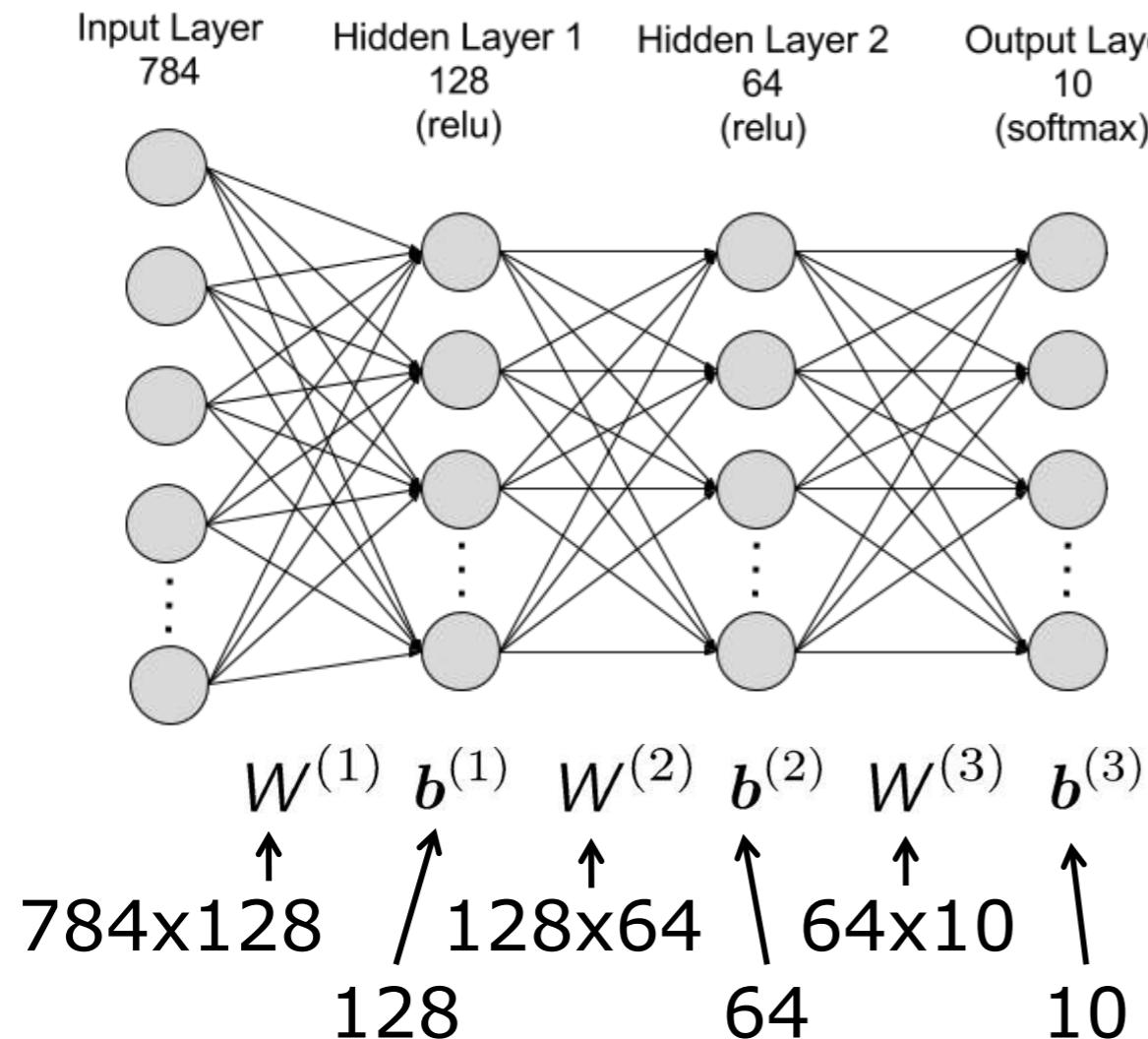
$$W^{(1)} \ b^{(1)} \ \dots \ W^{(k)} \ b^{(k)}$$

What we'd like is an algorithm which lets us find weights and biases so that the output from the network approximates $y(x)$ for all training inputs x .

What are the parameters for?

parameters encode the features

How many parameters does this network have?



$$100352 + 128 + 8192 + 64 + 640 + 10$$

1,09,368 parameters!

Given a network structure, weights and biases tell the network what to do

parameters: weights, bias

Activations are also parameters, but today we fix our activation and understand how the network learns/tunes the weights & bias

- We don't explicitly input the features.
- Features are learnt by the network.
- So we have lots of parameters!

Training is achieved through labeled data

Labels

$$\{(x_i, y_i)\}_{i=1}^I$$

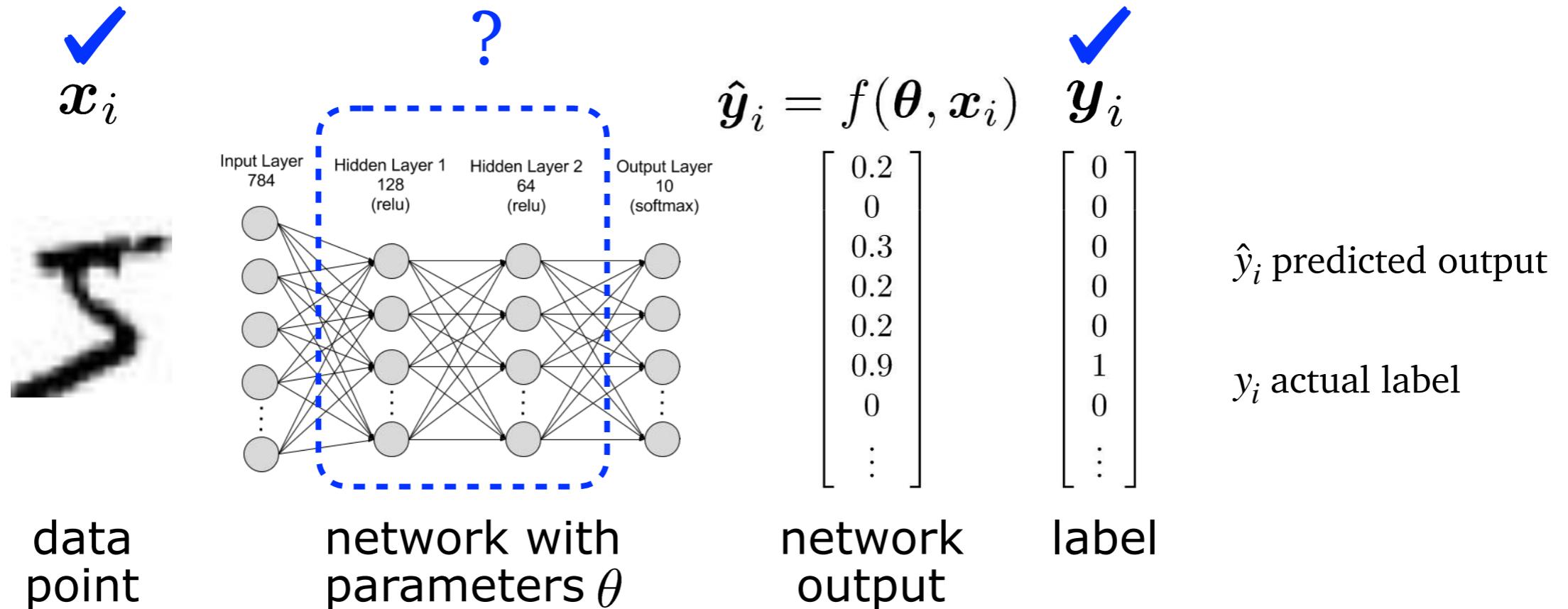
x_i : hand written training set

yi: correct labels

Estimating parameters:

The parameters are estimated through the learning process

We input the data and the labels and let the network determine the parameters



layer 1

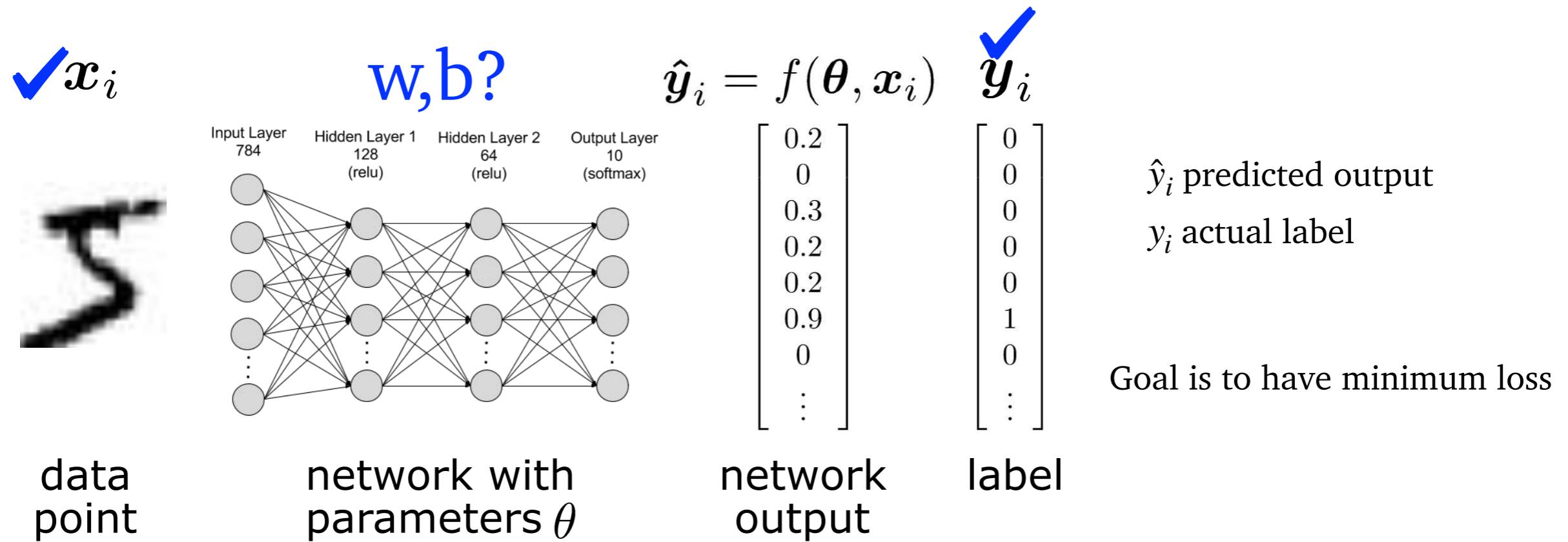
$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \dots & w_{0n} \\ w_{10} & w_{11} & \dots & w_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k0} & w_{k1} & \dots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \right)$$

Cost Function (loss)

We define a loss (= cost) function over all weights and biases of the network

Computed using training data Input to are the network parameters

Output is the error of the network with these parameters on the training data



$$L_i(\theta) = \|\text{output}_i - \text{label}_i\|^2$$

This output is small for fairly good performance and large, when network is not performing well!

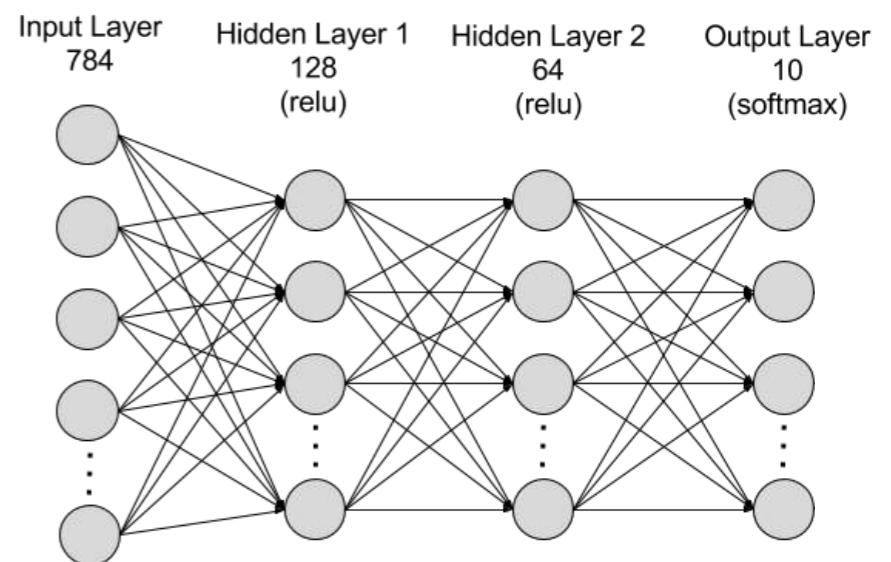
$$L(\theta) = C(w, b)$$

the output depends on x (input), w and b

A quadratic cost function; also known as the mean squared error or just MSE

Loss function is computed over all the training examples

- We initialise the network with random weights and biases
- We evaluate the performance of the network over all examples and see how well (badly) network performed



loss = all examples, avg. squared(NN(input) – label)

$$L(\theta) = \frac{1}{I} \sum_i L_i(\theta) = \frac{1}{I} \sum_i \|f(\theta, x_i) - y_i\|^2$$

Check: (**8**, 8)

Our goal is minimise the loss :

$$\arg \min_{\theta} L(\theta) \quad \theta = (w, b)$$

We want the parameter θ minimize the sum of average squared losses over all examples

$$L(\theta) \mapsto \mathbb{R}$$

high dimensional 1 dim

1,09,368 dimensions!

Loss minimisation using gradient descent

Goal:

- training a neural network to find weights and biases which minimize the quadratic cost function $C(w, b)$
- This function is fairly complex with several weights and biases
- For simplicity, we first take a 1D function

1D gradient descent algorithm

For simplicity, we first take a 1D function:

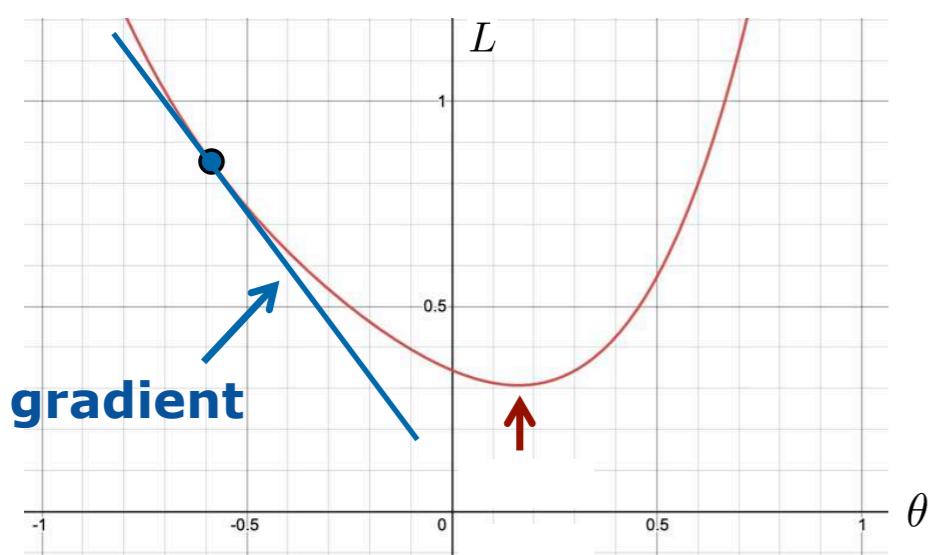
Two requirements for computing the gradient descent:

- Differentiable (i.e. has a derivative for each point in its domain)
- Convex i.e. $\frac{d^2 f(x)}{dx^2} > 0$
- Quasi-convex function (having saddle points)

$$f(x) = x^2 - x + 3$$

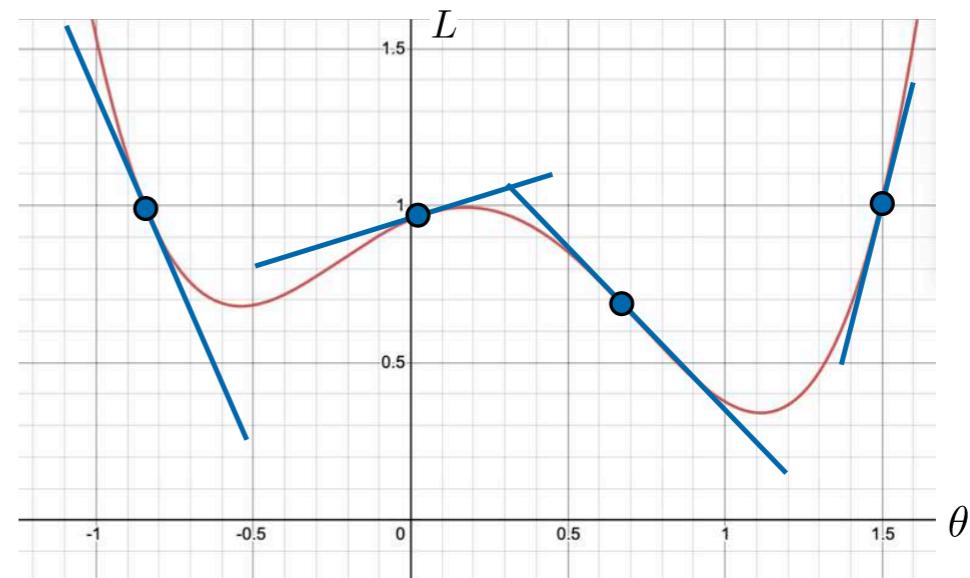
$$\frac{df(x)}{dx} = 2x - 1, \quad \frac{d^2 f(x)}{dx^2} = 2$$

Which direction should I step so that I decrease the output of the function most quickly?



We compute the gradient $\nabla L = \frac{\partial L}{\partial \theta}$

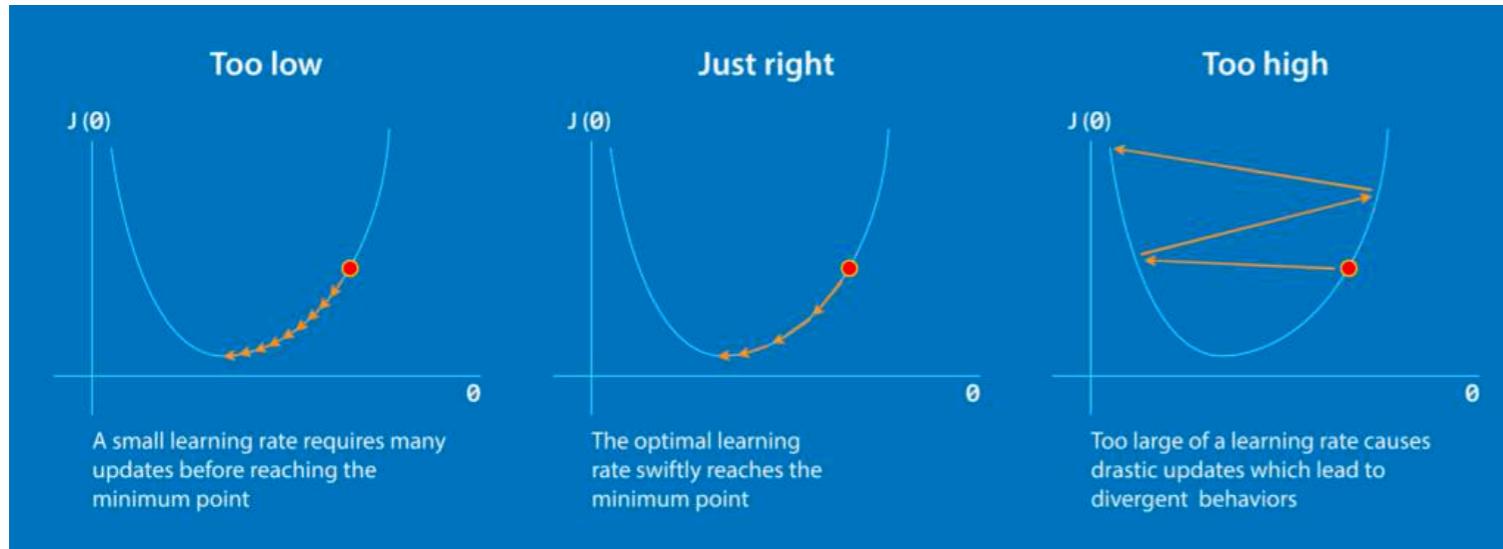
gradient tells us the direction of steepest ascent



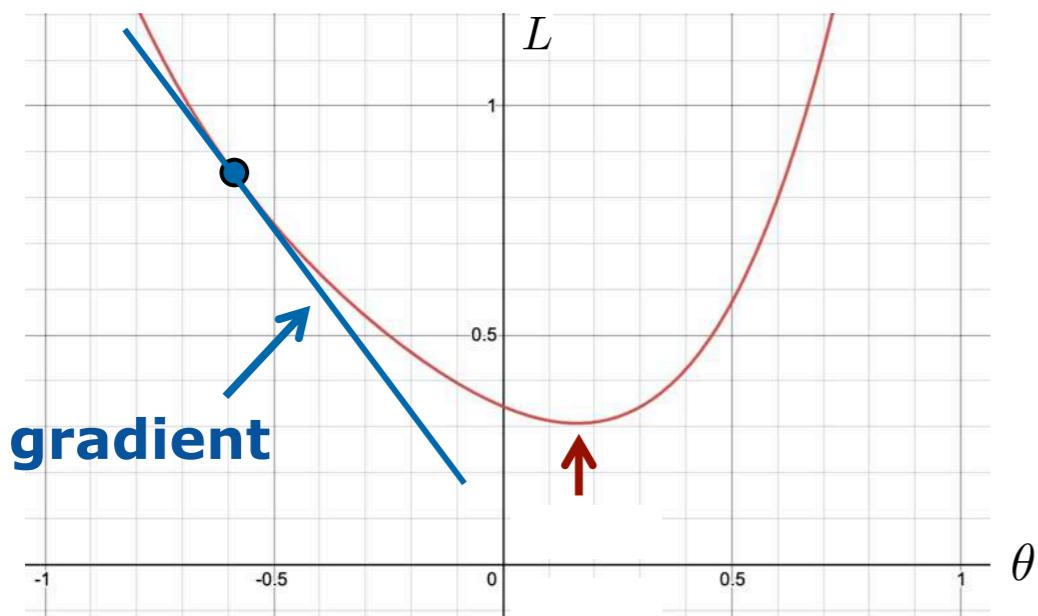
The gradient direction determines in which direction should we go next, λ is the learning rate (we go a small step downhill):

$$\text{STEP: } \theta^{(j+1)} = \theta^{(j)} - \lambda \nabla L|_{\theta^{(j)}}$$

1D gradient descent



Learning rate λ (scaler)



The gradient tells us in which direction
the next small step should be taken

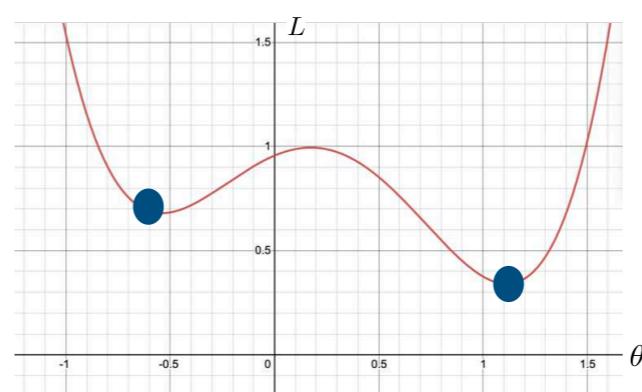
First derivative of loss: $\nabla L = \frac{\partial L}{\partial \theta}$

Learning rate (small value): $\lambda = 0.01$

Initialize θ^0 with a random value

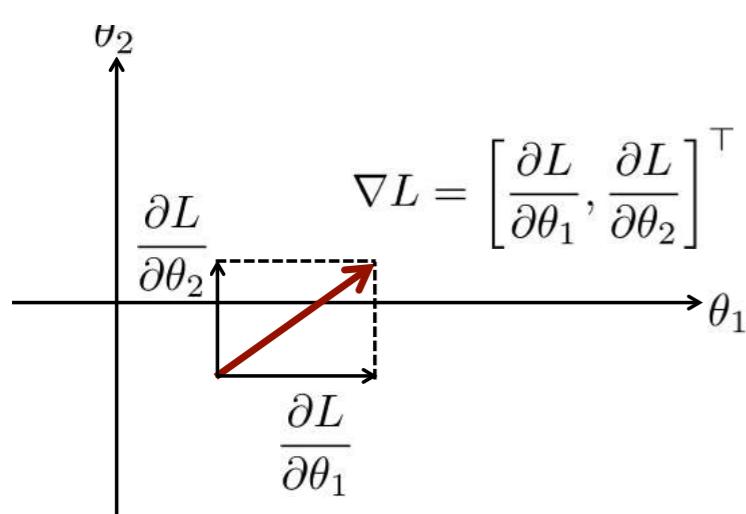
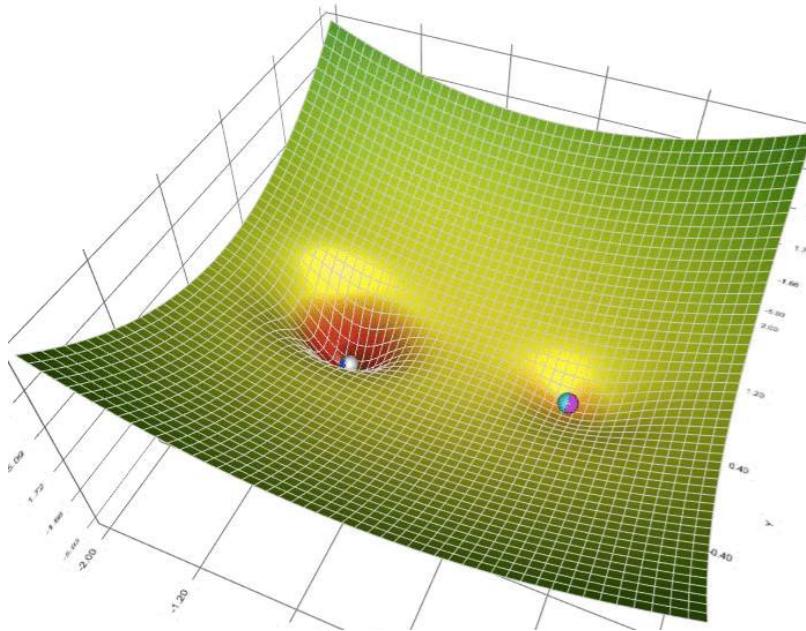
while (!converged)

$$\theta^{(j+1)} = \theta^{(j)} - \lambda \nabla L|_{\theta^{(j)}}$$

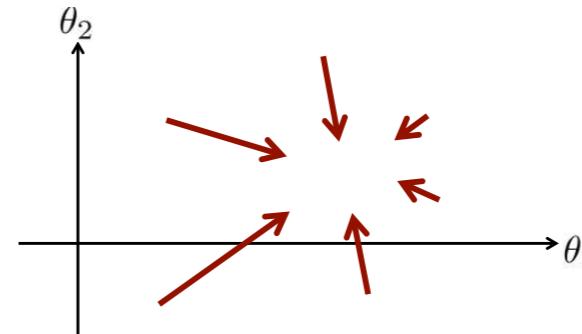


gradient descent gives the
local minima

2D gradient descent



- We can use the same approach for 2D
- We compute the direction vectors



Which direction should I step so that I decrease the output of the function most quickly?

Gradient indicates which change leads to the fastest reduction of the loss

Which dimension of the parameter vector will have a greater impact on the minimisation?

Should I move in the θ_1 or θ_2 direction? Which is a larger values?

We use both magnitude and direction of the gradient

$$\theta^{(j+1)} = \theta^{(j)} - \lambda \nabla L|_{\theta^{(j)}}$$

- choose a starting point (initialisation)
- calculate gradient at this point
- make a scaled step in the opposite direction to the gradient (objective: minimise)
- repeat points 2 and 3 until one of the criteria is met:
- maximum number of iterations reached
- step size is smaller than the tolerance (happens when a minima is reached)

For higher dimension,
gradient vector has more dimensions (1,09,368 dimensions in the digit classification example)

For a large dimension function:

$$\nabla L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_n} \end{bmatrix} \rightarrow \theta^{(j+1)} = \theta^{(j)} - \lambda \nabla L|_{\theta^{(j)}} \quad \theta = (w, b)$$

We will use the same approach
 ↓
 Negative gradient of the cost function (vector)

Loss L_i is for one training example $L_i(\theta) = ||f(\theta, \mathbf{x}_i) - \mathbf{y}_i||^2$

Loss function is averaged over all examples $L(\theta) = \frac{1}{I} \sum_i L_i(\theta)$

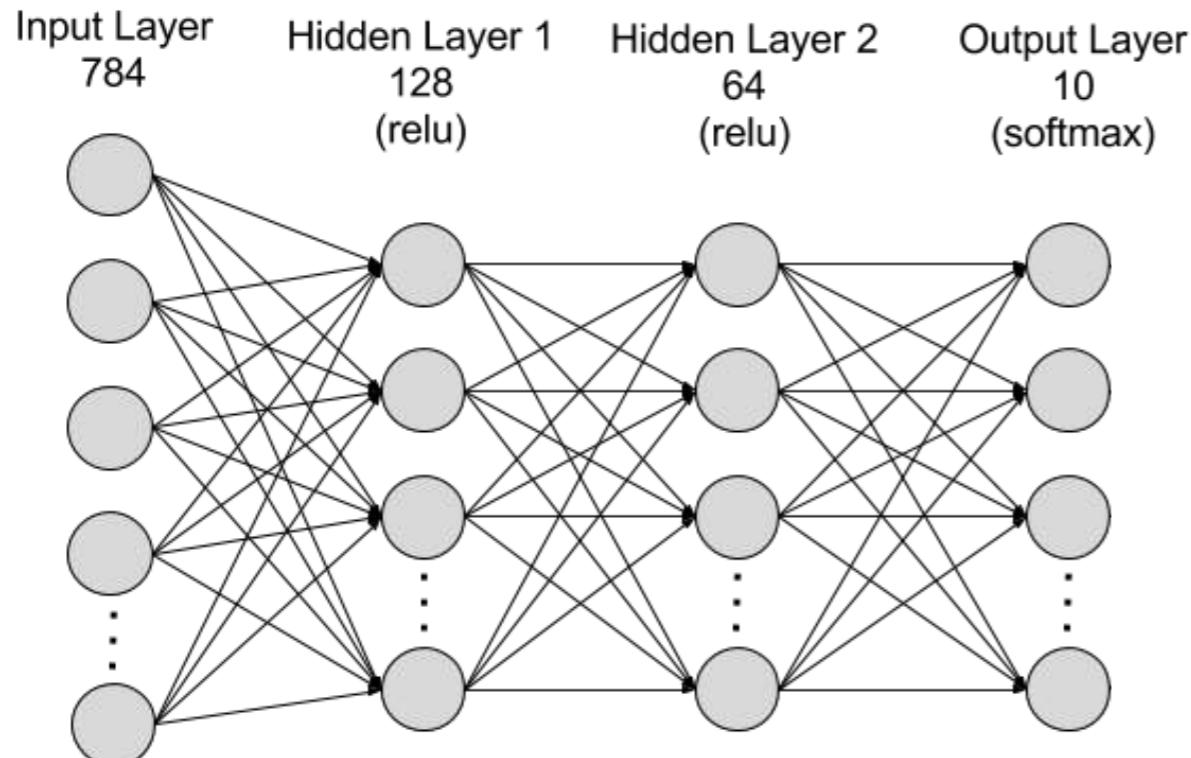
then, we need to minimise this $L(\theta)$

We need to average over all training examples and compute all gradients for each training example whenever we perform a single GD step

$$\nabla L = \frac{1}{I} \sum_i \nabla L_i \quad \text{Minimise cost function}$$

What we mean by ‘network learning’?
is
Minimising the cost function

- Input are weights and bias (high dimension)
- Parameters are like the training examples
- And we have a single number representing the cost function over all the set



The cost function encodes the relative importance of weight and bias

And at every iteration we adjust the weights and bias just a little bit!

The cost function – $\nabla L(\theta)$ carries both magnitude and sign which tells us which weight should we care about the most in the next iteration

$$\begin{bmatrix} 0.31 \\ 0.01 \\ -0.73 \\ 0.81 \end{bmatrix}$$

We need to average over all training examples and compute all gradients for each training example whenever we perform a single GD step

$$\nabla L = \frac{1}{I} \sum_i \nabla L_i \quad \text{Minimise cost function}$$

- Take the gradient of the loss function i.e. derivative of the loss function for each parameter in it
- Randomly select the initialisation values
- Substitute these parameter values in the gradient
- Calculate step size by using appropriate learning rate
- Calculate new parameters
- Repeat from step 3 until an optimal solution is obtained

How do we optimise the process for lots of training examples?

How do we compute the gradients for complex and nested functions?
(we need to compute gradients for the network parameter space)

How do we optimise the process for lots of training examples?

Stochastic gradient descent algorithm

Compute a gradient only on a small, sampled subset of examples

Mini-batch



These are only ones we will feed to
the gradient descent

- Compute a gradient only on a small, sampled subset of examples
 - We sample a mini-batch in each step of gradient descent
 - Use only mini-batch (B) to compute

$$\nabla L = \frac{1}{|B|} \sum_{i \in B} \nabla L_i$$

On average we will still find our minima using this approach!

This gives the approximation of the gradient

How do we compute the gradients for complex and nested functions?
(we need to compute gradients for the network parameter space)

Neural networks-part3 and CNNs

DSE312-CV

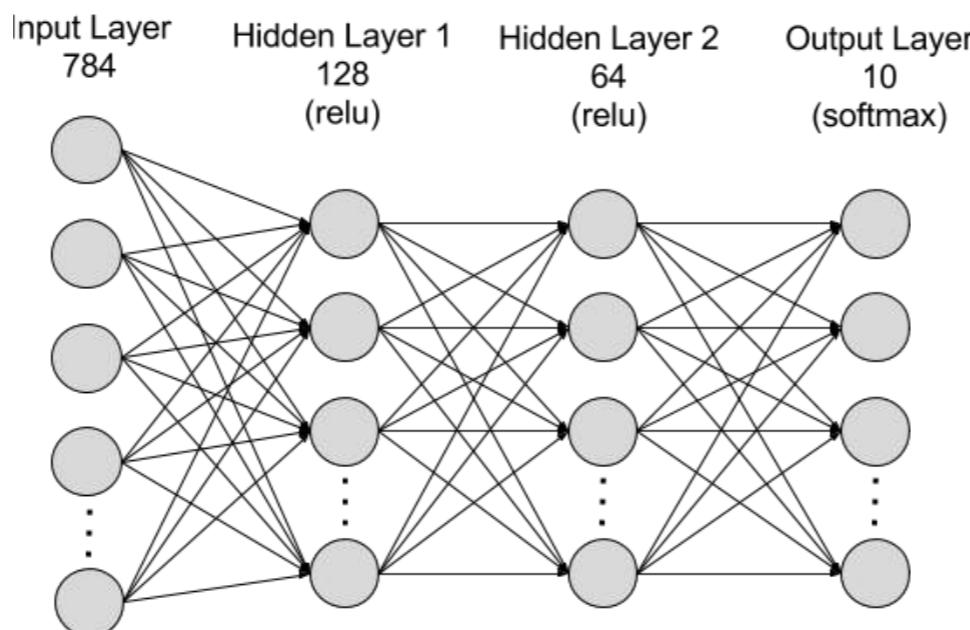
Lecture30

Bhavna R

How do we compute the gradients for complex and nested functions?
(we need to compute gradients for the network parameter space)

How do we compute the gradients for complex and nested functions?

- Compute ∇L_i step by step for each mini-batch
- Neuron activations are chains of activation functions and matrix-vector multiplications
- Many connections between the neurons
- We need to estimate the parameters running through all these layers



How do we compute this 109386 dimensional gradient ?



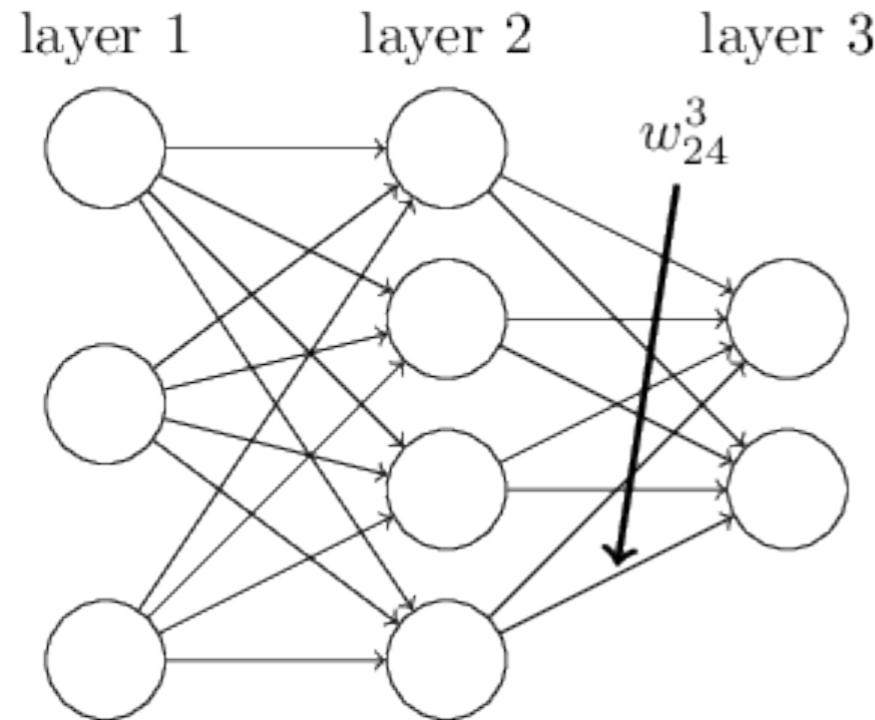
Backpropogation algorithm



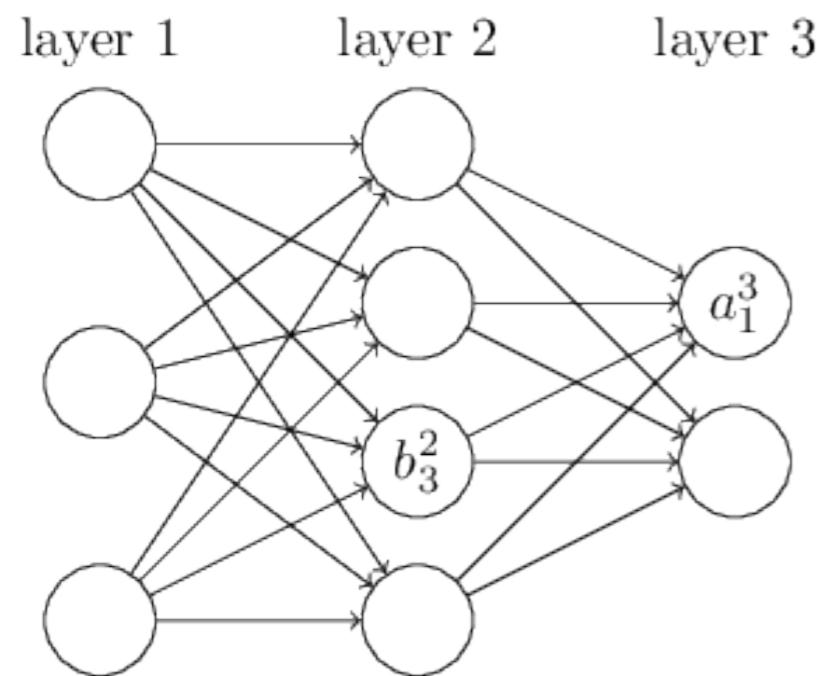
performs automatic differentiation of complex nested functions

In some sense, it allows us to compute the errors and update the weights and bias for the next iteration

Backpropagation algorithm: Notations



w_{jk}^l is the weight from the k^{th} neuron in the $(l - 1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



b_j^l bias of the j -th neuron in the l -th layer.

a_j^l j -th neuron in the l -th layer is related to the activations in the $(l - 1)$ -th layer

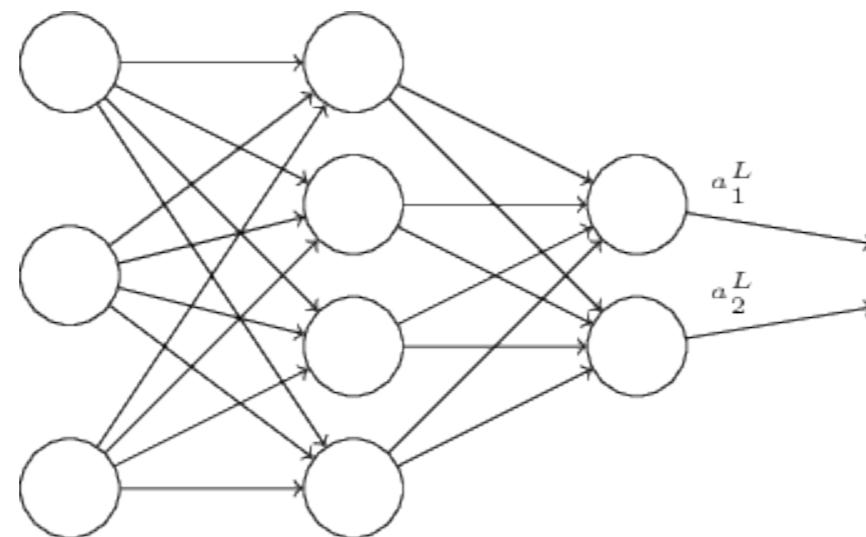
$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

We already learnt that we can use matrix notions to express the connections between the neurons layer by layer

Backpropagation: how changing the weights and biases in a network changes the cost function

The goal of back propagation algorithm is to optimize the weights and bias so that the network can learn how to correctly map arbitrary inputs to outputs.

to compute the partial derivatives $\partial L/\partial w$ and $\partial L/\partial b$ of the cost function L with respect to any weight w or bias b in the network



Cost L is a function of $L(a^L) = L(a_1^L, a_2^L)$

$$L_i(\theta) = \|\text{output}_i - \text{label}_i\|^2$$

$$L_i(\theta) = \|f(\theta, \check{x}_i) - \check{y}_i\|^2$$

θ(w,b)?

$$\nabla L = \frac{1}{I} \sum_i \nabla L_i$$

- input training example x is fixed and the outputs y are also fixed parameters,
- cost function L is the output of activations a^L which composes of weights and bias
- We also have initialised the network with random values (initial w, b).
- We have a cost function computed already to start with (not the correct one though!)

break down the gradient computation into smaller steps using chain rule

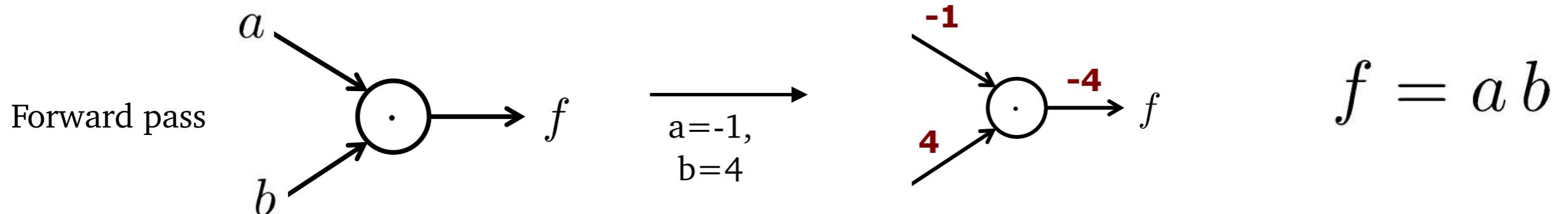
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

To understand this, let us take a directed graph that has edges and nodes

- Directed computational graph
- Nodes contain mathematical operations
- Edges encode input/output values

Looks very similar to the neurons!

eg: We represent the function $f(a,b)$, $f=ab$



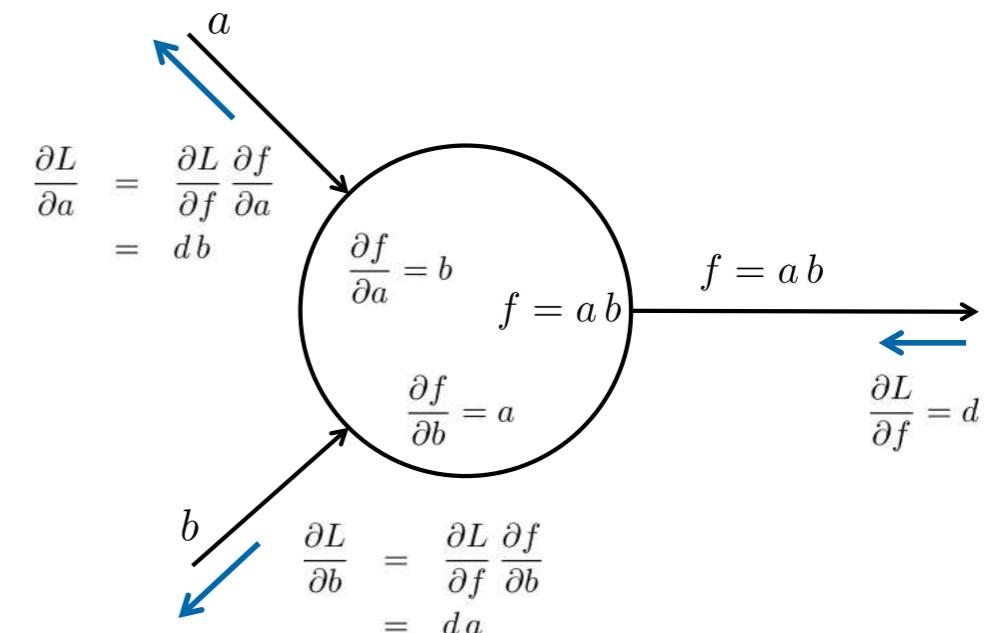
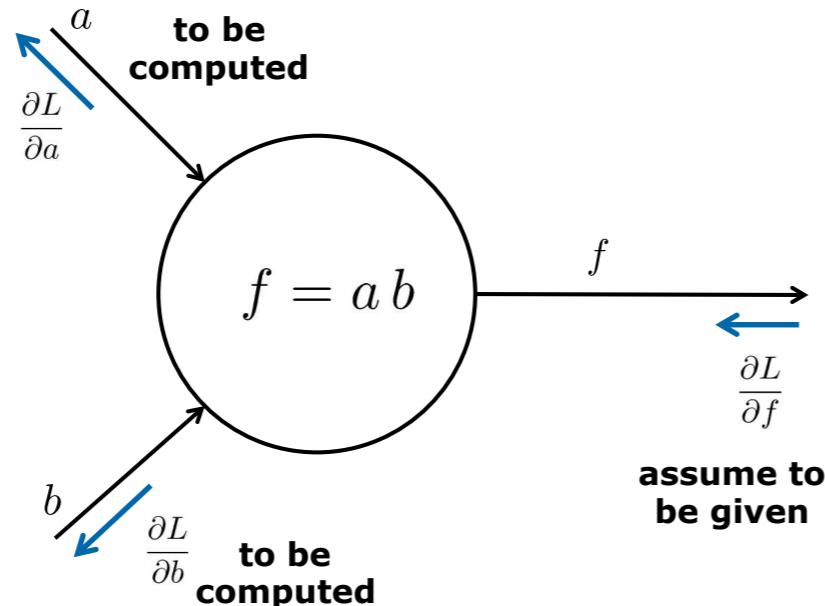
We evaluated the function at a point from inputs to outputs

- Now, we want compute the gradient of the function at the given point
- We can do this by traversing the graph backwards

- At each node, we compute the local derivative of the function w.r.t. the local inputs (backward pass)
- This f goes into computing the cost function in the neural network

This f is like the cost function in the neural network

compute the gradient
at a specific point in
the backward pass

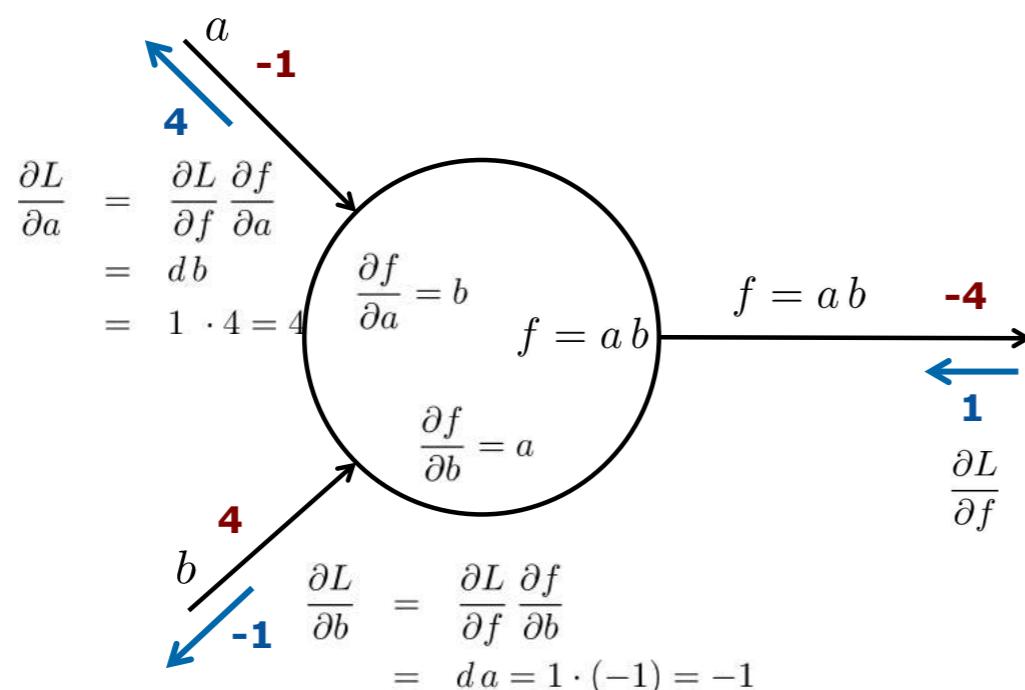


Given:

$$a = -1,$$

$$b = 4$$

$$\frac{\partial L}{\partial f} = 1$$



break down the gradient computation into smaller steps using chain rule

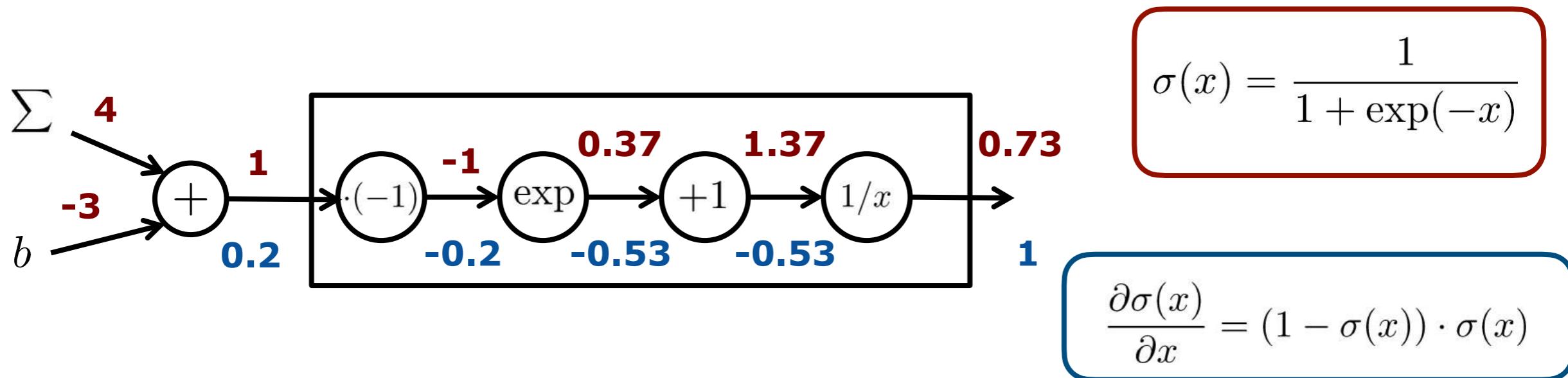
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

The derivatives of the activation functions are used in the backpropogation algorithm

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU) ^[2]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Forward pass and backpropagation: Sigmoid activation function example

$$a = \sigma \left(\sum w_n a_n + b \right)$$



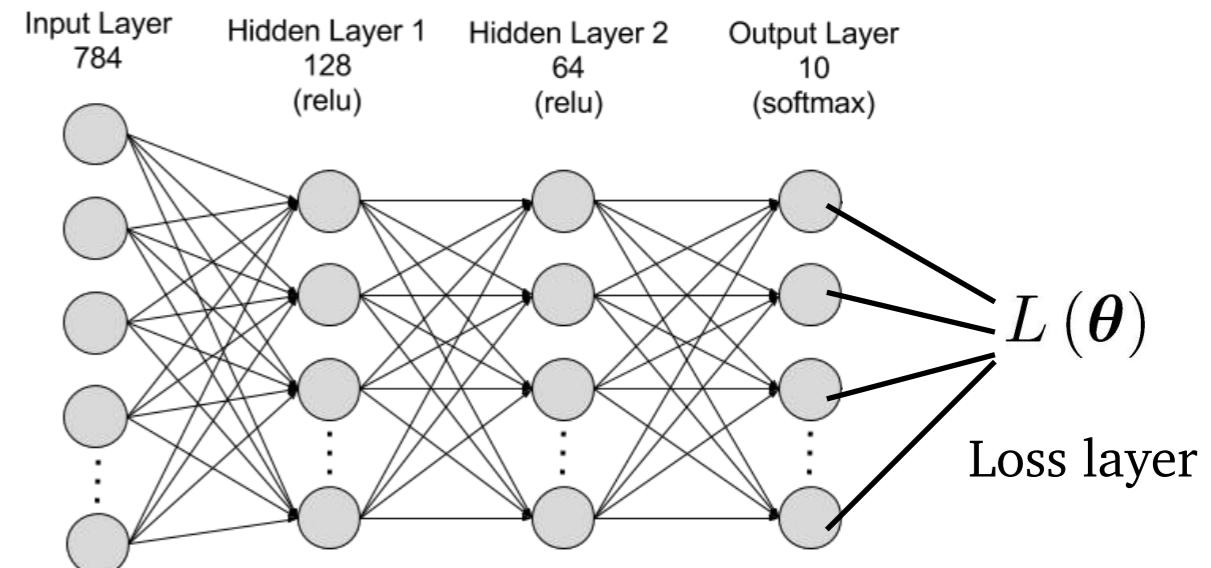
- The initial weights and bias are set randomly that lets us compute all values during the forward pass
- In the backward automatic differentiation, we just traverse in reverse and compute all the gradients

To decrease the error:

- For negative gradients, increase in weight
- For positive gradients, decrease in weight

Remember, we know our input vector and the output labels

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



$$L(\theta) = \frac{1}{I} \sum_i L_i(\theta) = \frac{1}{I} \sum_i \|f(\theta, x_i) - y_i\|^2 \quad \text{Minimise cost function}$$

$$\nabla L = \frac{1}{I} \sum_i \nabla L_i$$

We can use the backpropagation for computing the gradient of the loss function for the mini-batch size

- We initialise the network with random values (initialise w, b).
- We have fixed the inputs and also have our correct labels, which allows us to determine the loss function for the training examples in the batch
- A loss is computed to start with (not the correct one though!)
- We compute the gradients using automatic differentiation

Updating weights to decreases the error:

- When the gradient is negative, increase in weight.
- When the gradient is positive, decrease in weight

Freely available book and reads well..

Neural Networks and Deep Learning

Michael Nielsen

The original online book can be found at
<http://neuralnetworksanddeeplearning.com>

Convolutional neural networks

Neural networks in the context of images

Image classification using feed forward networks

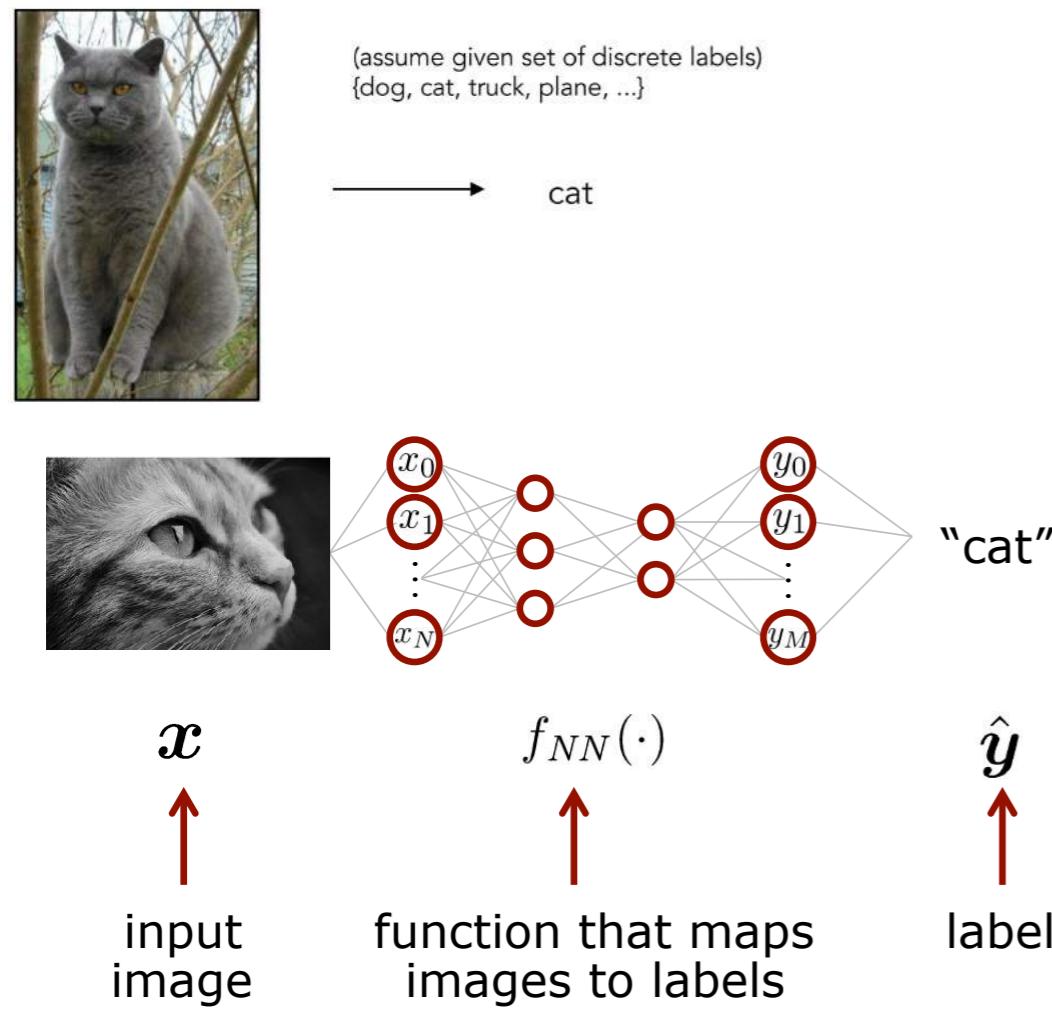
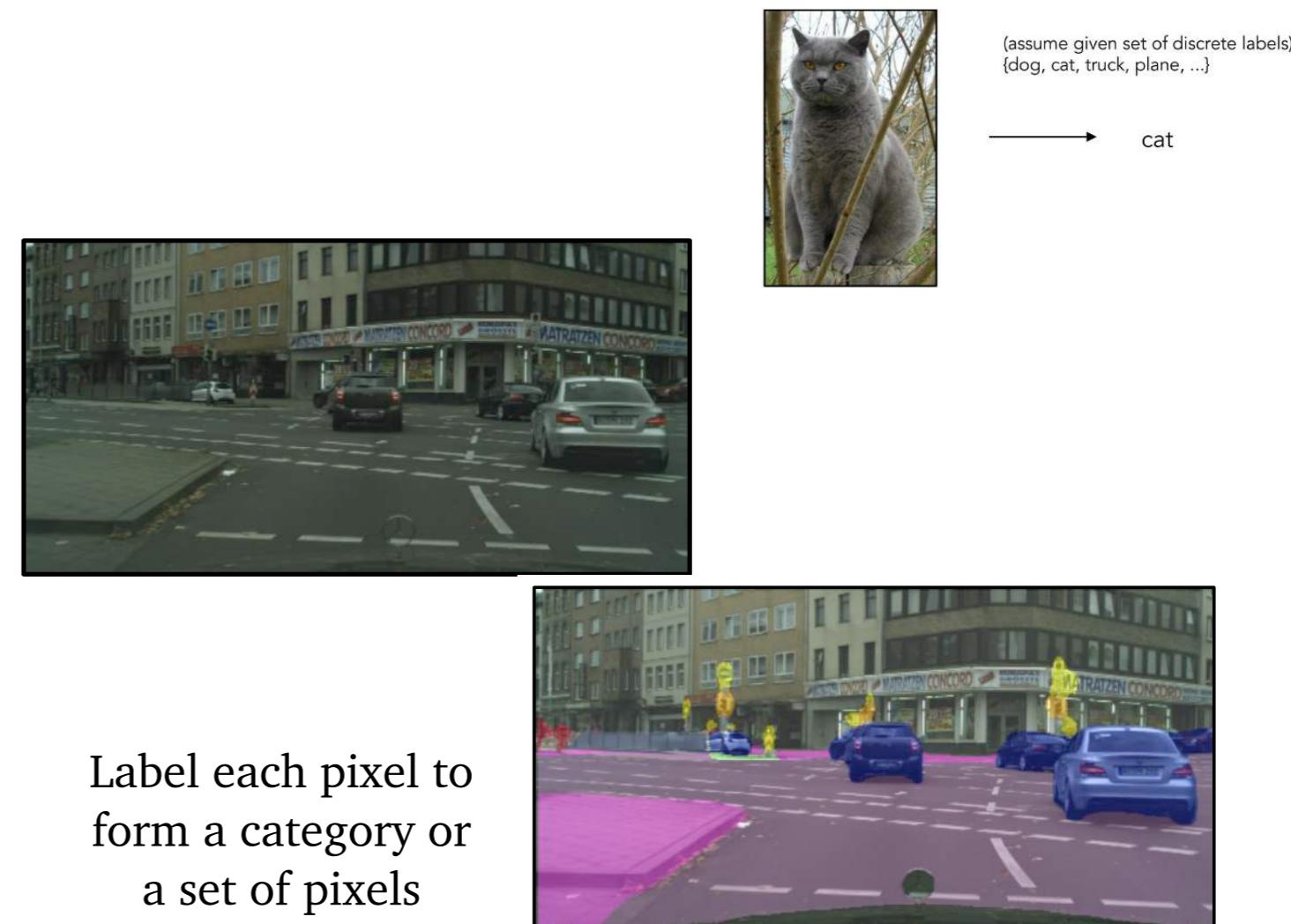


image intensity as 1D vector is the input layer

such a network does not store the spatial information to actually interpret what we see on the image

Spatial information ignored

Image Classification and Semantic segmentation using CNNs

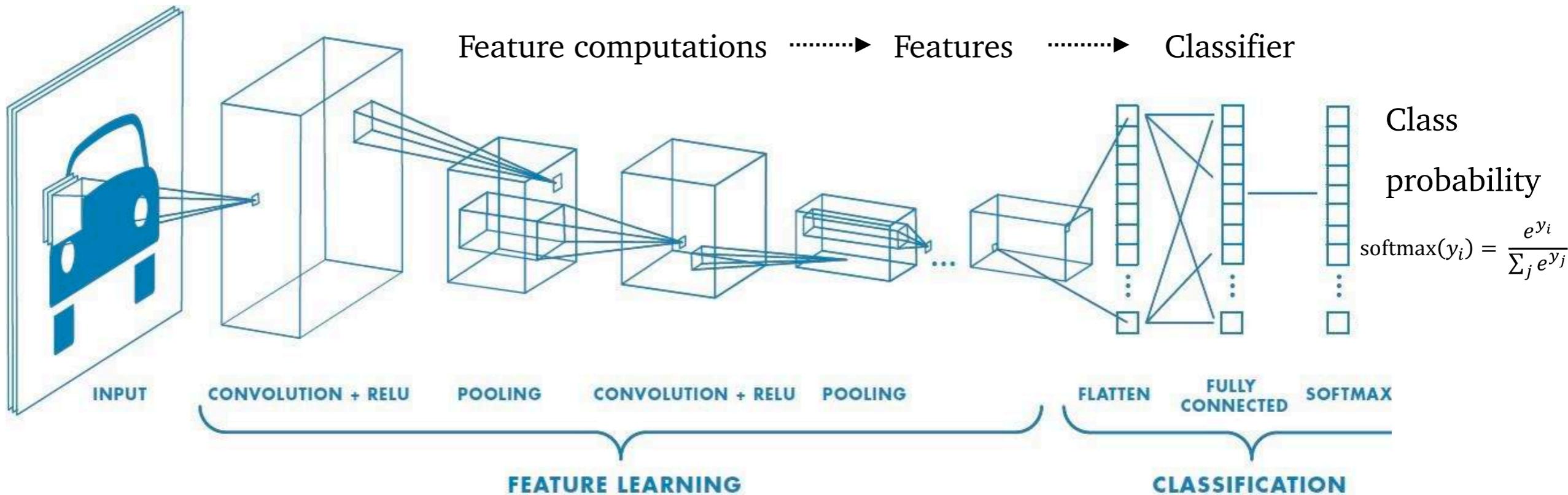


Convolutional neural networks overcome this problem

- preserve the 2D spatial order to maintain neighbourhoods
- Network layers can learn features that also encode spatial information
- CNNs use convolutions & subsampling (called pooling)

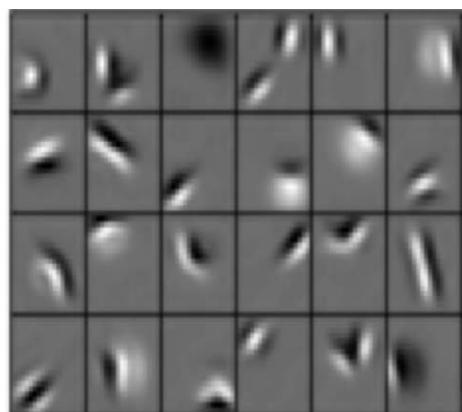
Whereas

Convolutional neural networks for end-to-end learning



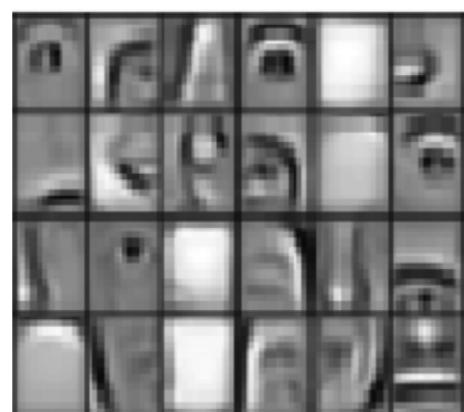
- Network layers can learn features that encode spatial information
- Convolutions are local operators
- CNNs use convolutions & subsampling (called pooling)

Low level features



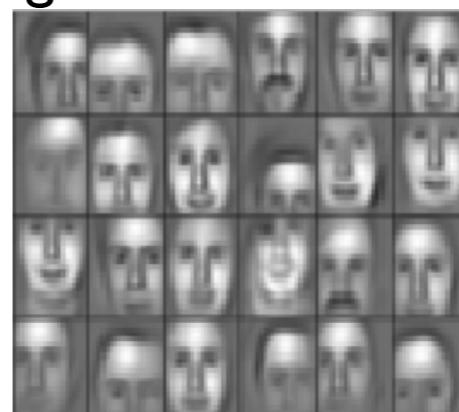
Edges, dark spots

Mid level features



Eyes, ears, nose

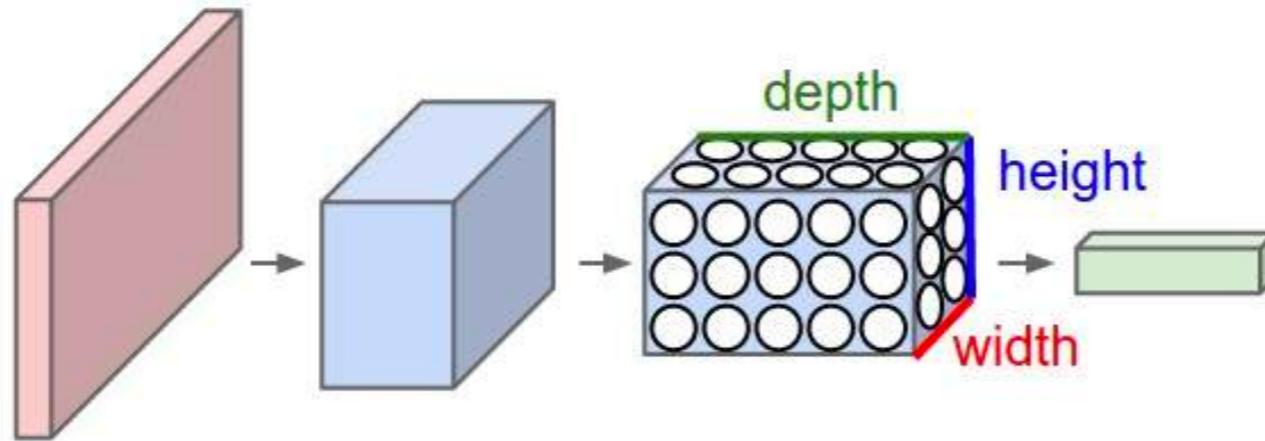
High level features



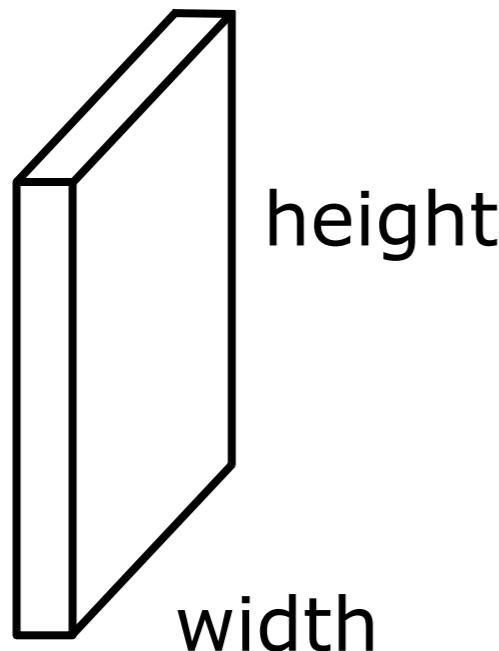
Facial structure

CNNs learn hierarchy of features directly from data

CNNs constrain the architecture of images in a sensible way by arranging neurons in 3-dimensions: width, height, depth



channels/depth



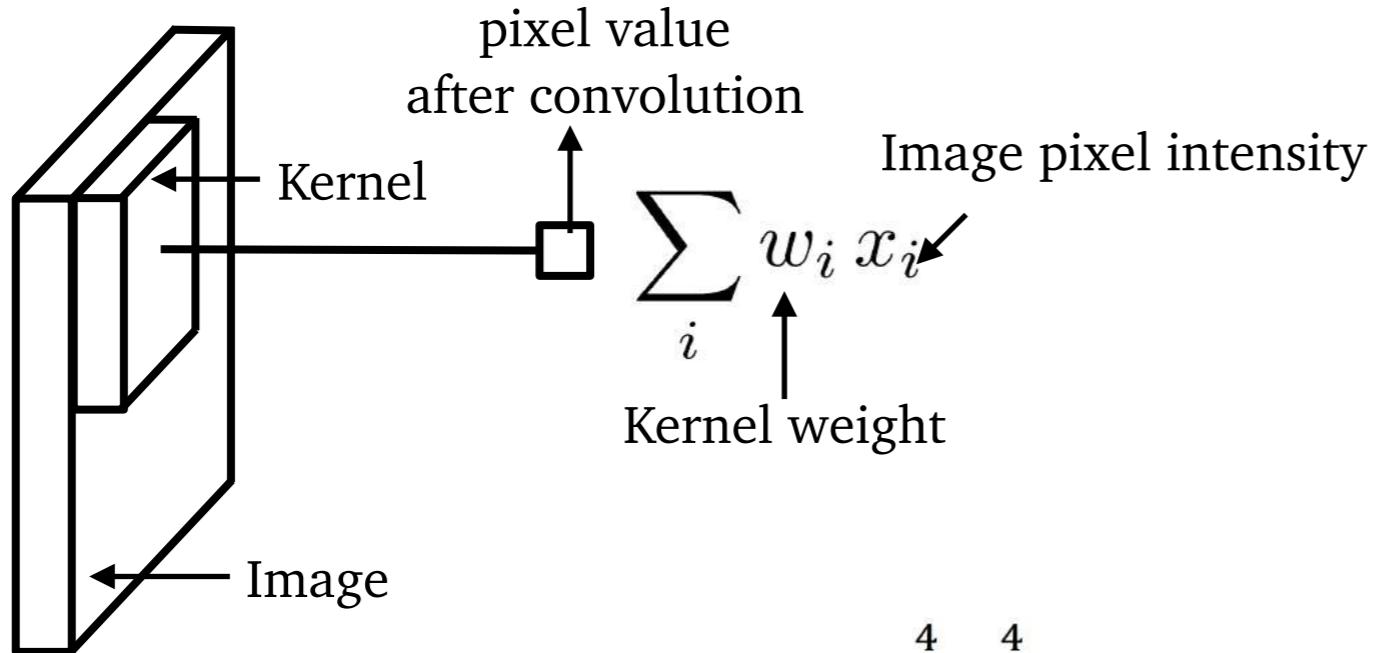
depth=1 Grayscale image



depth=3 Color image

Can also store multi-spectral images

What is “Convolution” in Convolutional neural networks?



- This operation leads to an output image as a result of convolution
- We retain spatial information inherit in the input image

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b \quad \text{4x4 filter: matrix of weights } w_{ij}$$

for neuron (p,q) in hidden layer

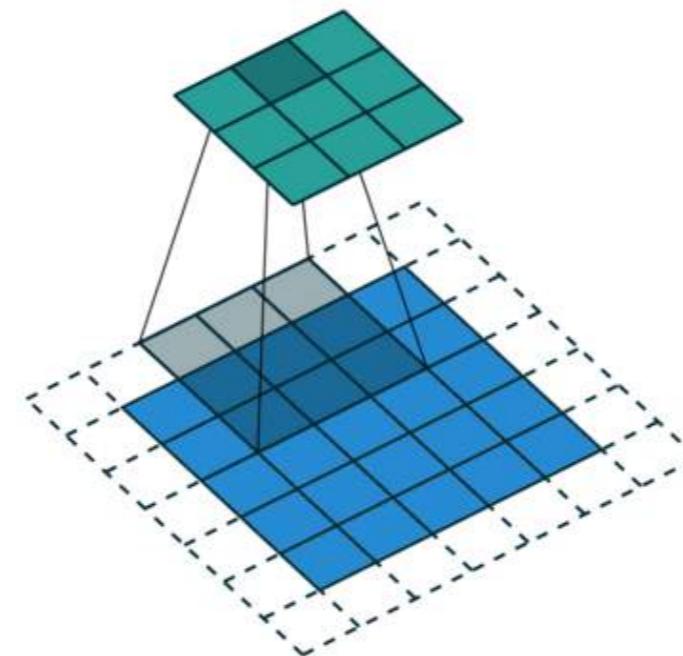


example of blurring operation using a convolution kernel with Gaussian kernel in CNNs

Neighborhood and spatial information preserved

- With zero padding, you can get the same size as input
- Else, output image size = $\text{input_size} - (\text{filter_size} - 1)$
- Eg: for 256×256 image size, 5×5 filter size will result in 252×252 image (called as ‘valid padding’)

Using stride during Convolution in CNNs



- The kernel traverses the entire image and perform convolution
- Padding is denoted by ‘dotted lines’
- Instead of moving the kernel by 1 pixel, you can also move by 2-pixels
- Stride defines by how many pixels we shift the filter forward each step
- Larger stride reduces overlaps and makes the resulting image smaller
- eg: for image size 64×64 and stride length =2 with padding results in 32×32 image

$+1$ $+1$ $+1$
 \rightarrow \rightarrow \rightarrow

10	23	31	2	3
8	35	44	33	1
2	13	0	2	1

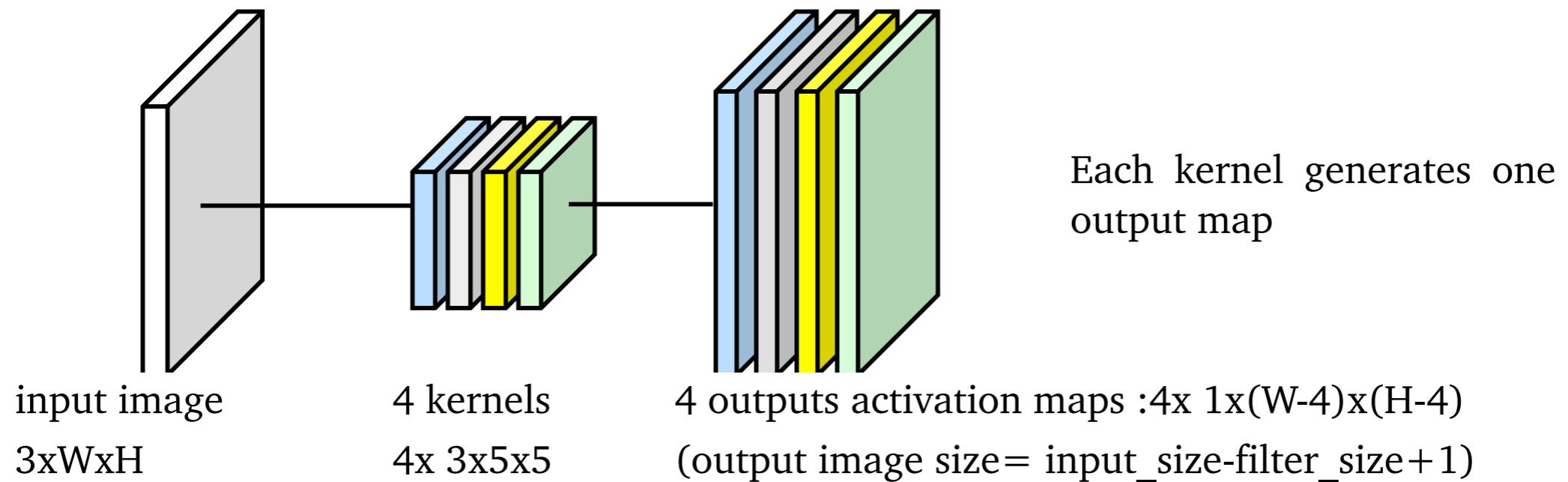
stride = 1

$+2$ $+2$ $+2$
 \longrightarrow \longrightarrow \longrightarrow

10	23	31	2	3
8	35	44	33	1
2	13	0	2	1

stride = 2

We can use multiple kernels stacked together, where each kernel is designed to detect different types of features



3 indicates 3 channels here

$$\text{output size } W' \times H' = (W - K + 1) \times (H - K + 1)$$

Feature map size after convolution:

$$C = ((S-k+2p)/t)+1$$

where,

C is the size of the convoluted matrix.

S is the size of the input matrix (use size in x and y separately).

k is the size of the filter matrix.

p is the Padding amount.

t is the Stride.

Eg: For square image size n = 12x12, kernel size = 3x3, 'no padding' and stride length = 1

the size of the convoluted matrix is 10x10.

When stride is not used: $C=S-K+1+2p$

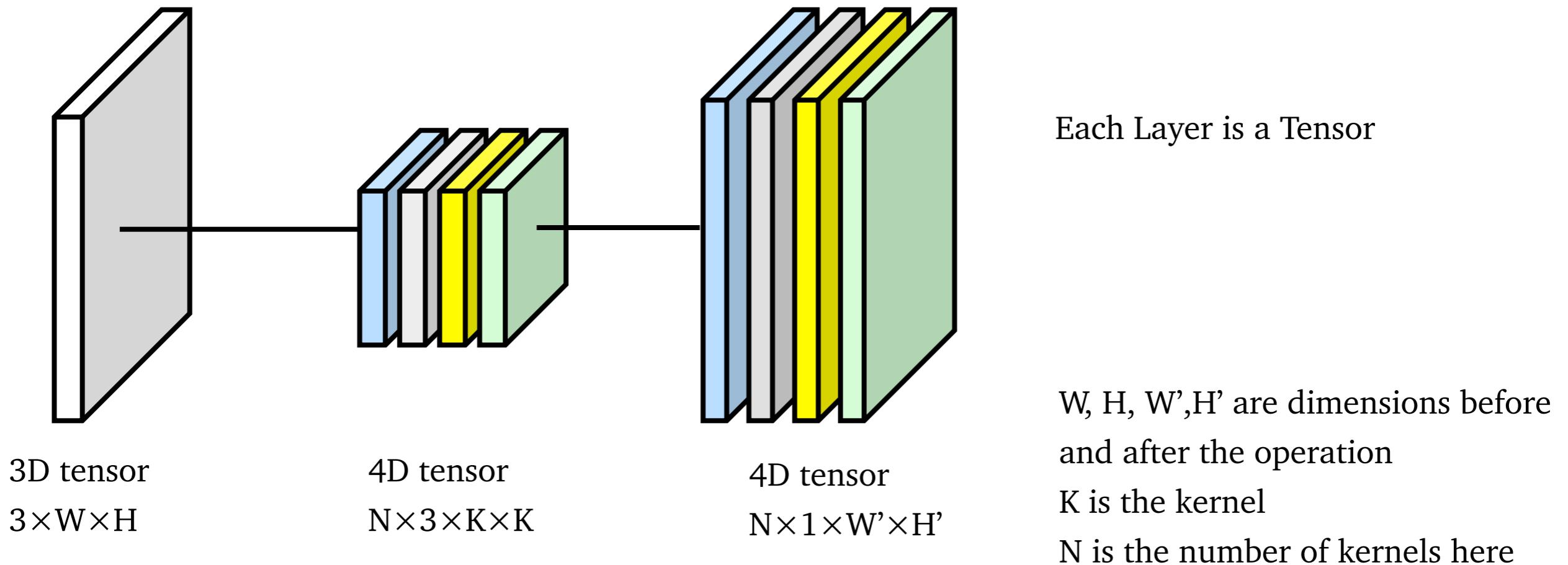
Padding is used in CNN, same as we did already in our previous lecture (please look up zero padding)

Concept of tensors

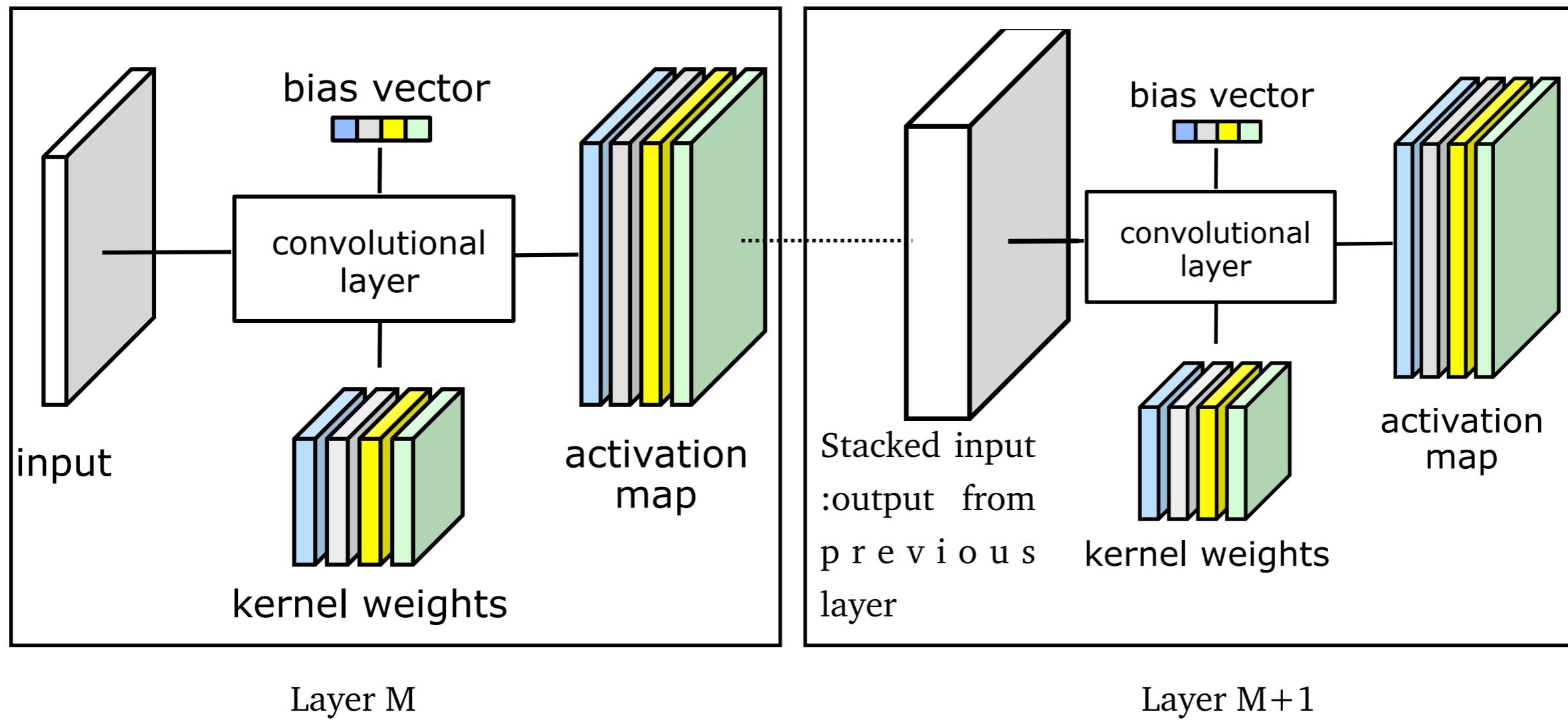
Array dimensions:

- Vector is a 1 dimensional array
- Matrix is a 2-dimensional array
- Voxelgrid is a 3-dimensional array

Tensor = generalization of the “array” (matrix) concept with a flexible number of dimensions



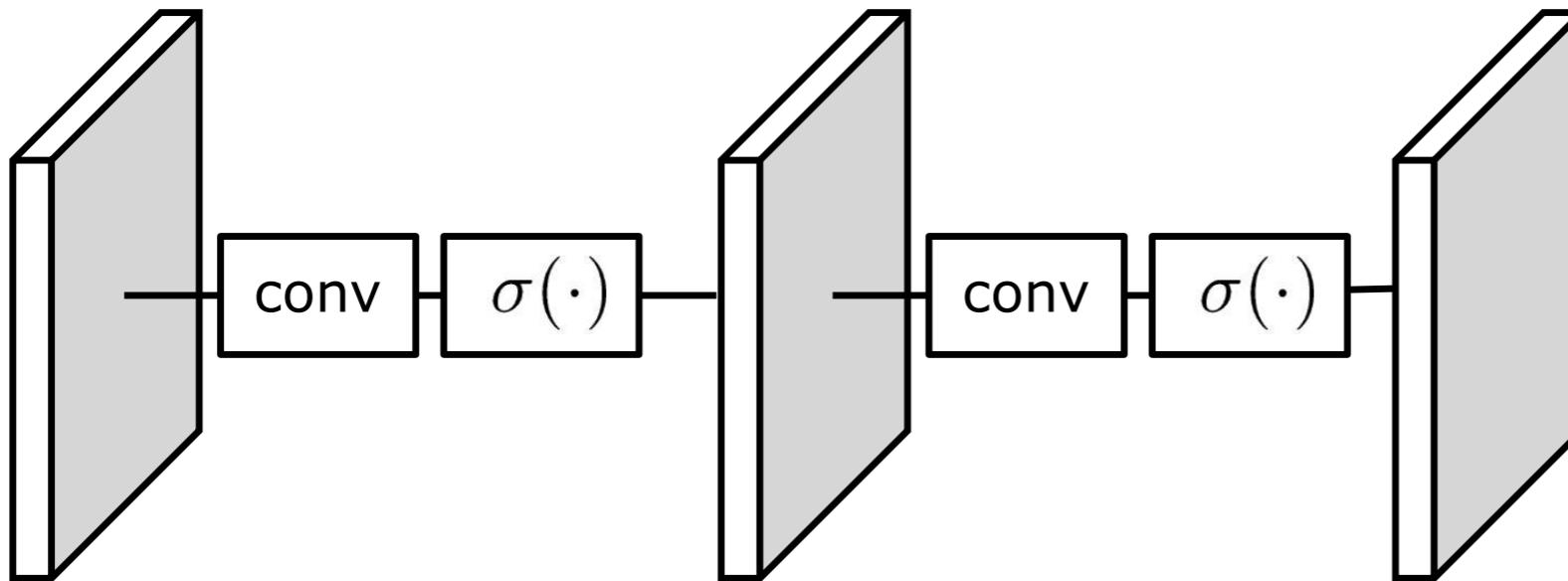
Parameters and stacking layers



Bias vector: 1 per kernel

Added to the result of kernel weight operation, (can be set to 0, for no bias)

Stacking Convolutions Layers With Activation Functions

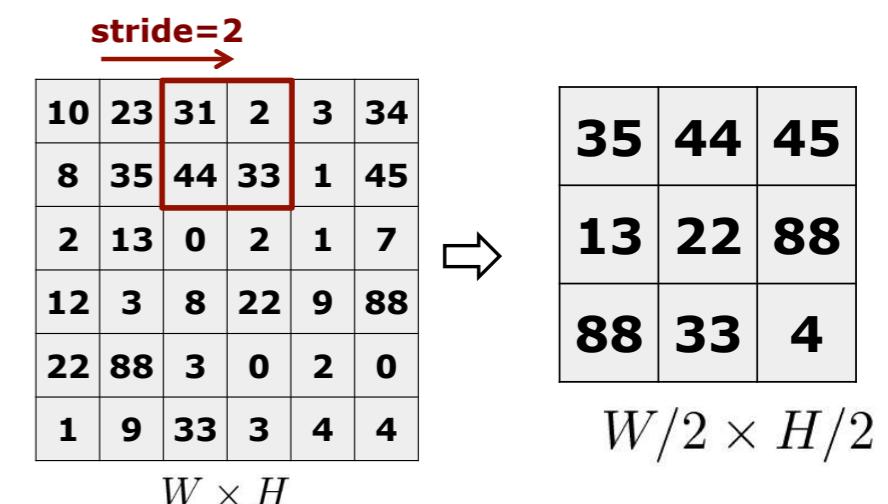
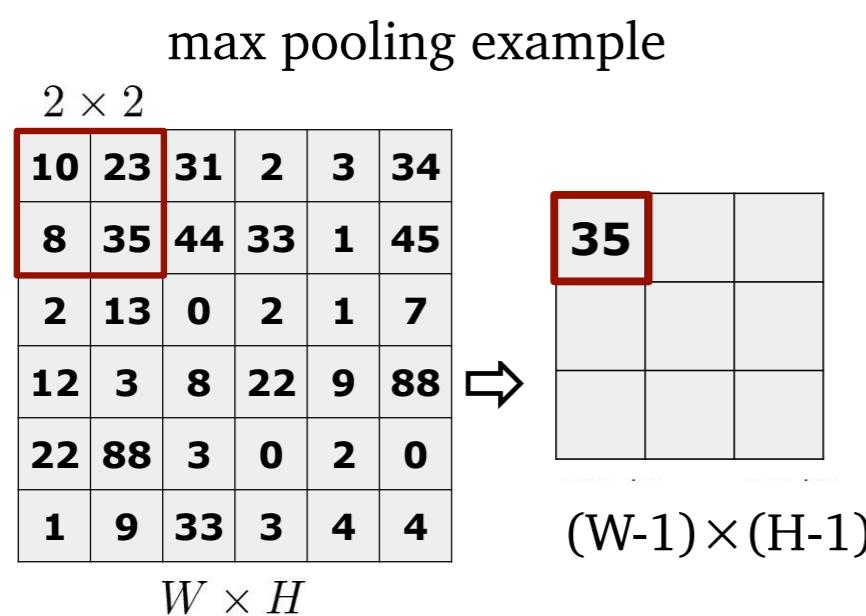
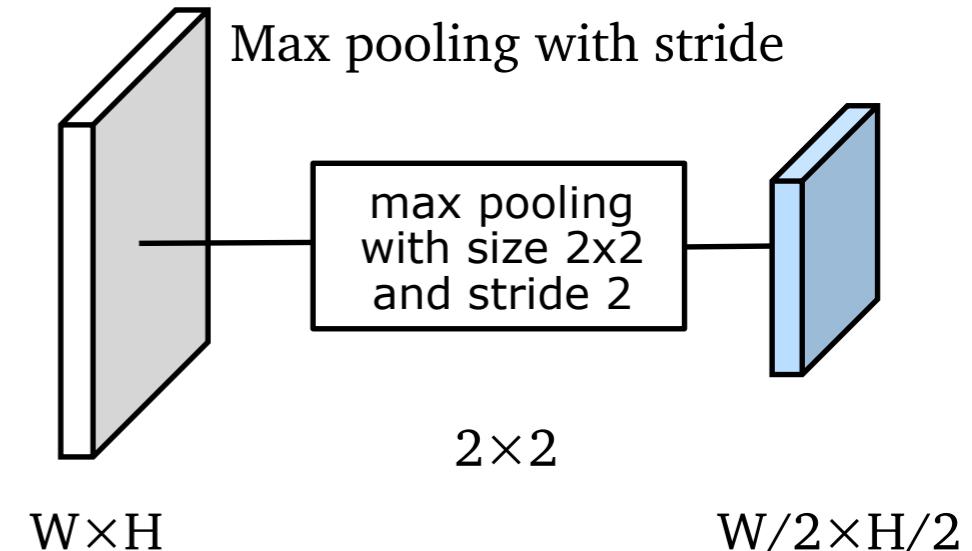
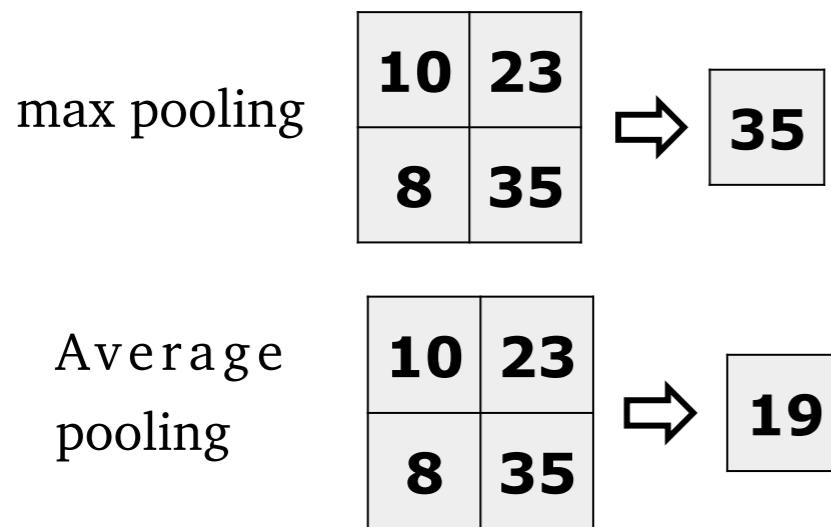


- For such linear shift-invariant kernels, we know that concatenations of convolutions are again a convolution
- Multiple layers can be combined into a single convolution
- Property breaks when introducing non-linear activation function

$$f * (g_1 * \dots * g_n) = f * g$$

Pooling

- Besides convolutions, CNNs also use pooling layers
 - Pooling combines multiple values into a single value to reduce the tensor sizes and combine information



Size tells us how many values to combine and stride define by how much to shift the mask, both determine output size

Activation outputs that store patterns (signals) can be sent into pooling layers that only pick prominent values (picks strongest response within a local neighbourhood)

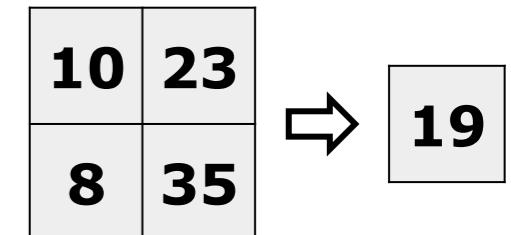
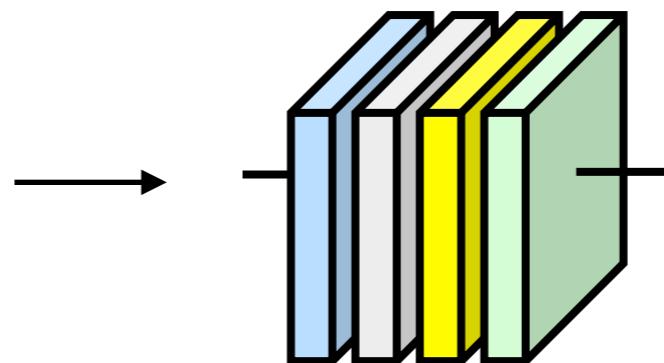
How do CNNs recognise features

-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	-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
-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	-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



-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	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
-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	-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

We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed



Each of these kernels is like a mini-image that can detect patterns within a small neighbourhood

By pooling, these patches indicate features

We can compare features from such image patch for classification tasks

Generating different feature maps with kernel types



Original



Sharpen



Edge Detect



“Strong” Edge Detect

0	-1	0
-1	5	-1
0	-1	0

0	1	0
1	-4	1
0	1	0

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

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

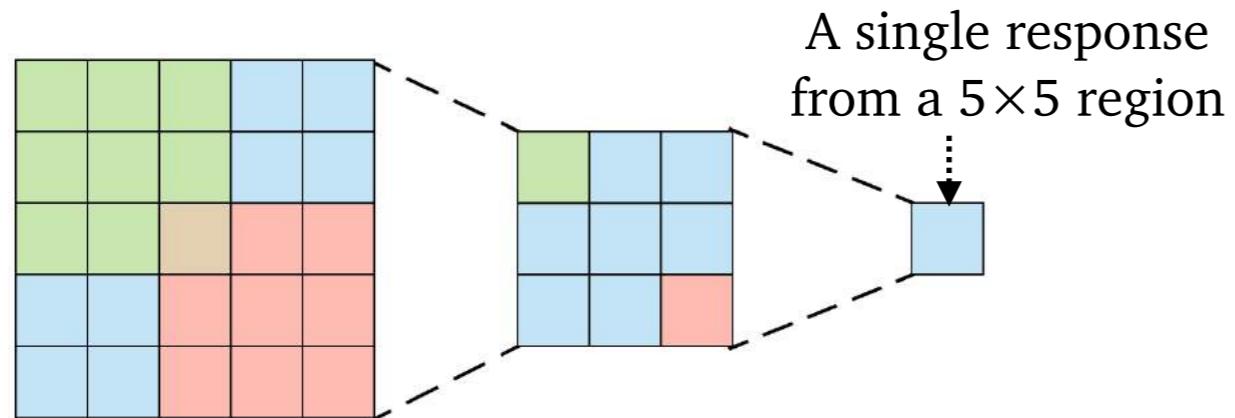
Some Simple filters (kernels)

Generating feature responses by stacking such layers

Activation outputs that store patterns (signals) can be sent into pooling layers that only pick prominent values (picks strongest response within a local neighbourhood)

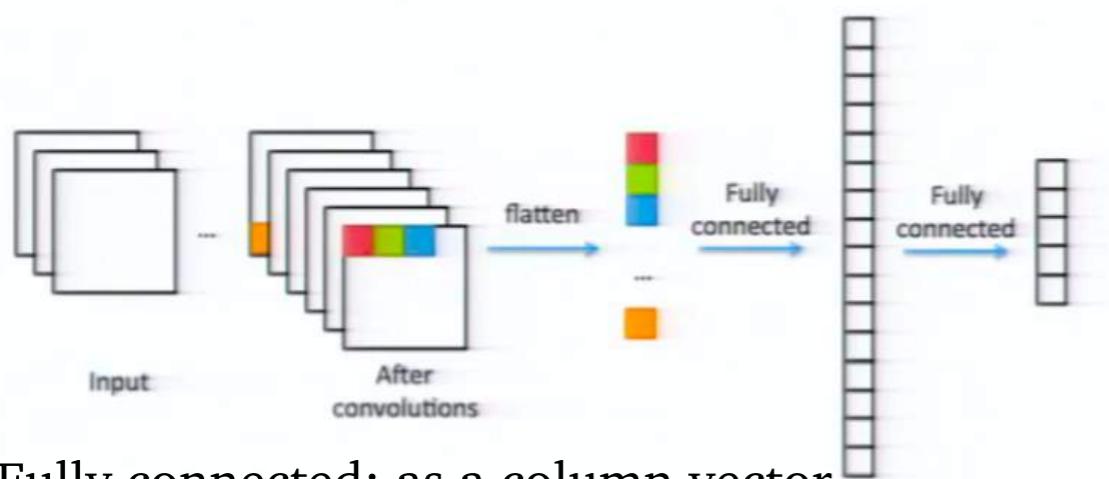
We can send the output of pooling layers further into convolutional layers

Eg: this can tell us, a certain feature is found say in the upper-left region of the image, a rough location estimate which gave a certain activation



- Stacking multiple pooling operations reduces W , H of the activation maps
- Elements in deeper layer are impacted by larger areas of the inputs

Eg: picking prominent features/patterns within a local neighbourhood using convolution & pooling



Fully connected: as a column vector

- CNNs are trained using SDG and backpropagation
- Large number of parameters need to be determined
- Normalizing the layers makes SGD converge faster
- Normalization of means and variance
- Different normalization approaches (batch, layer, instance, ...)

CONV and POOL layers output high-level features of input

Fully connected layer uses these features for classifying input image

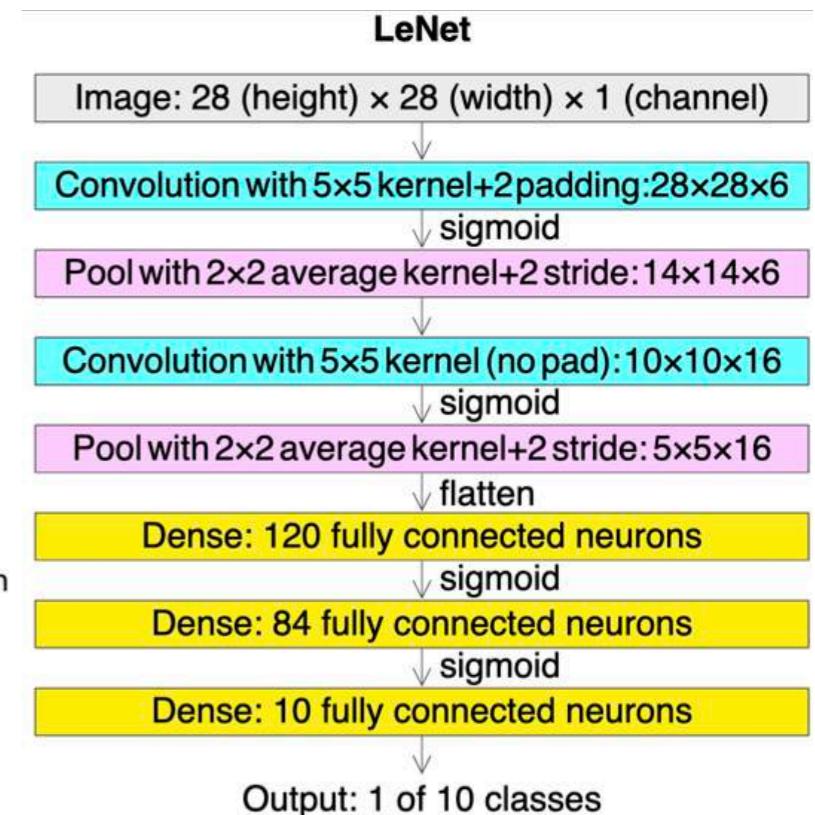
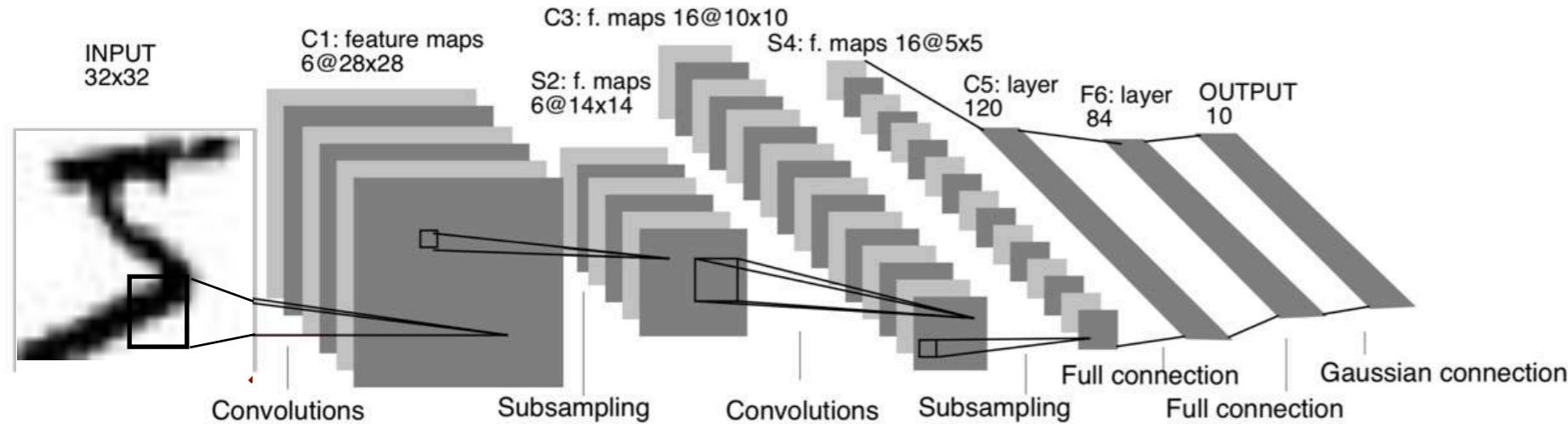
The first CNN: LeNet-5 by LeCun et al., 1989

- First convolutional network was proposed by Yann LeCun et al.
- Recognition of handwritten digits
- 5 layers: 2 convolutional and 3 fully connected ones
- Outperformed all other networks (at that time)



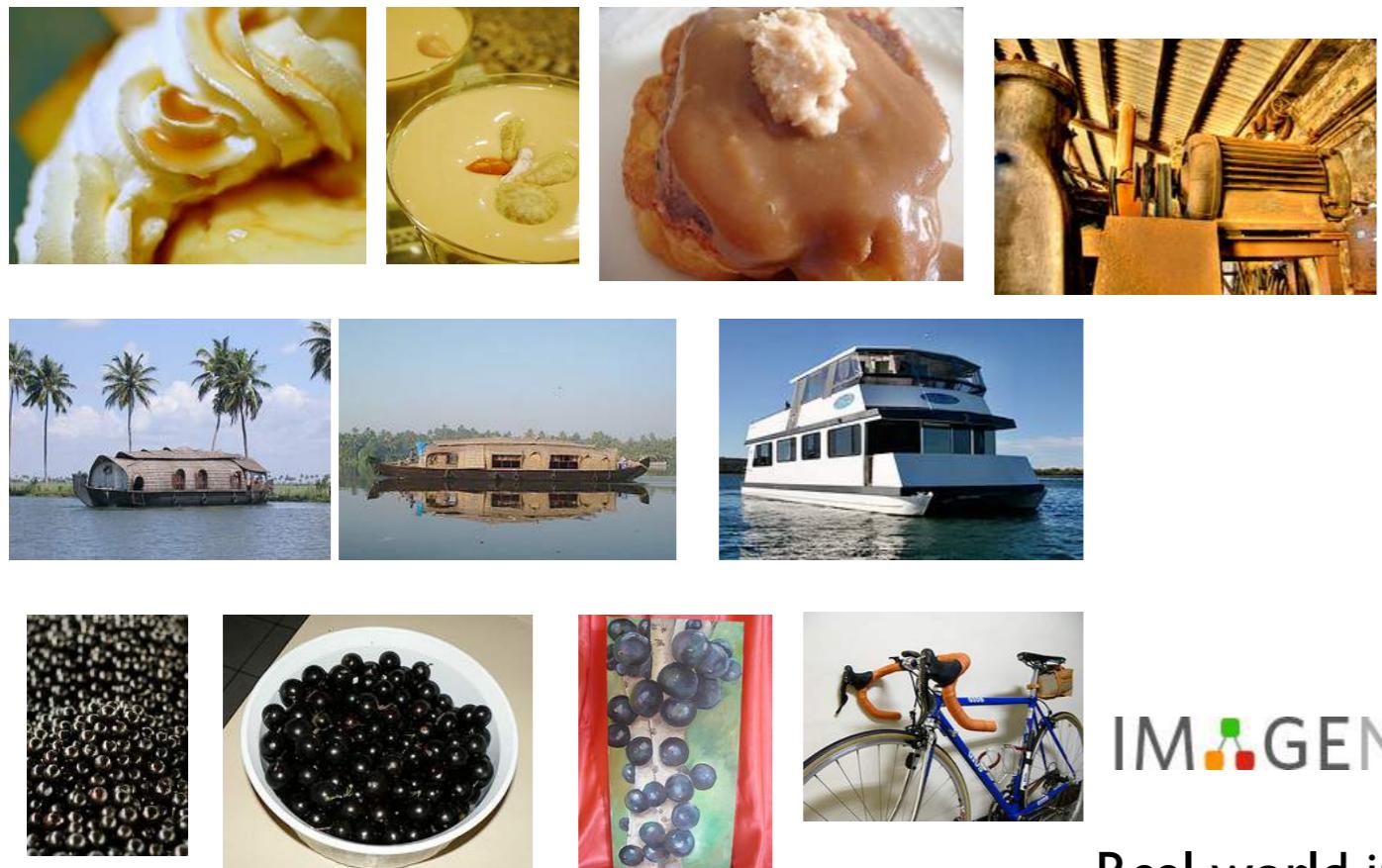
Yann LeCun

504 / 92



There are other examples: AlexNet, VGG-16/19

Deep Learning on real world images



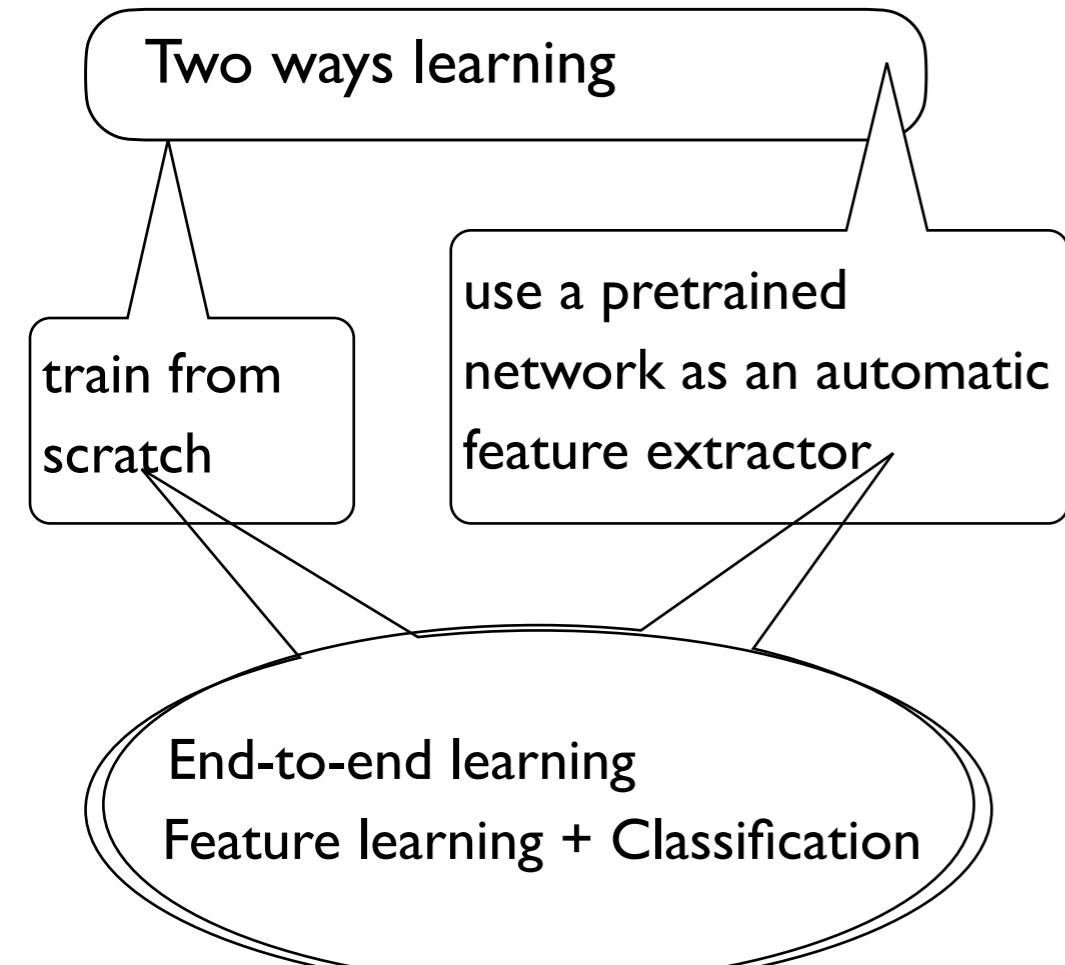
IMAGENET
Real world images

- ImageNet is an ongoing research effort to provide researchers around the world an easily accessible image database.
- ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories.
- allows to test neural network architecture on real world images and classify them
- Trained networks: AlexNet, GoogleNet, VGG-16 and VGG-19

The pretrained networks such as AlexNet, GoogleNet have learned rich feature representations for a wide range of natural images.

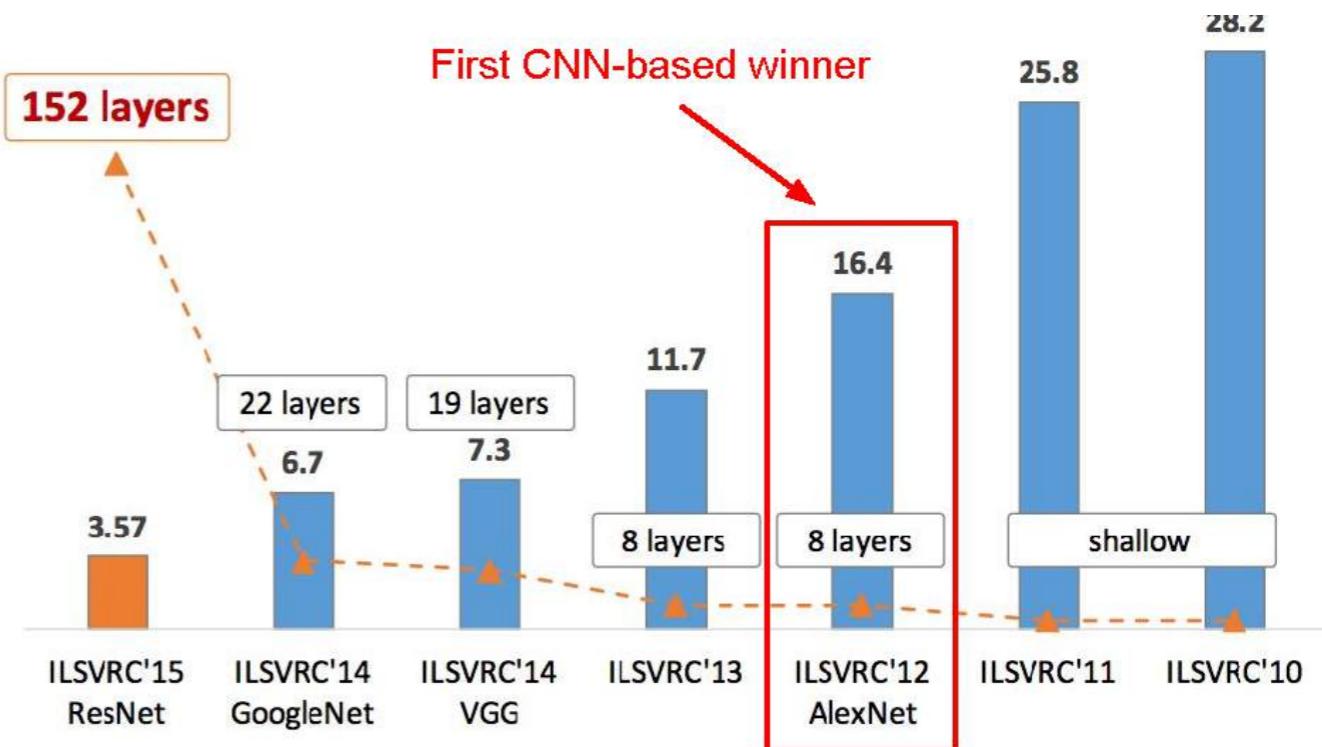
The learned features can be applied to a wide range of image classification problems.

Convolutional neural network (CNN) approach



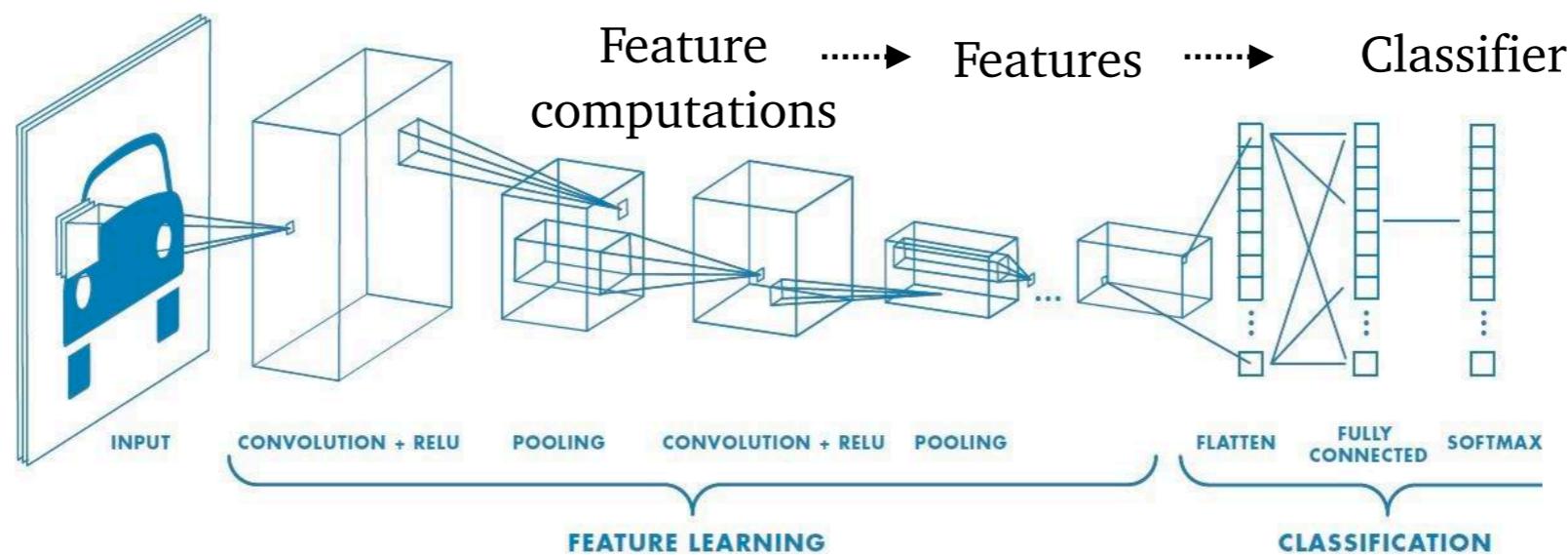
Recommendations	train from scratch	pretrained network
Training data	1000s to millions of labeled images	100s to 1000s of labeled images
Computation	Intensive	Moderate
Training time	Days to weeks	minutes to hours
Model accuracy	High (can overfit for small datasets)	Good, depends on pretrained model

ImageNet performance



Summary

- CNNs are standard tool today for vision tasks
- CNNs learn visual features
- Layers combining convolutional blocks, pooling, and normalization
- Classification layers at the end
- End-to-end trainable networks (input image to output) , so no features input
- Computes low-level - mid-level -high-level features directly from images (builds feature hierarchy)
- Can be used for classification and segmentation by tweaking the architecture



Optic flows

Lucas and Kanada method

DSE312-CV

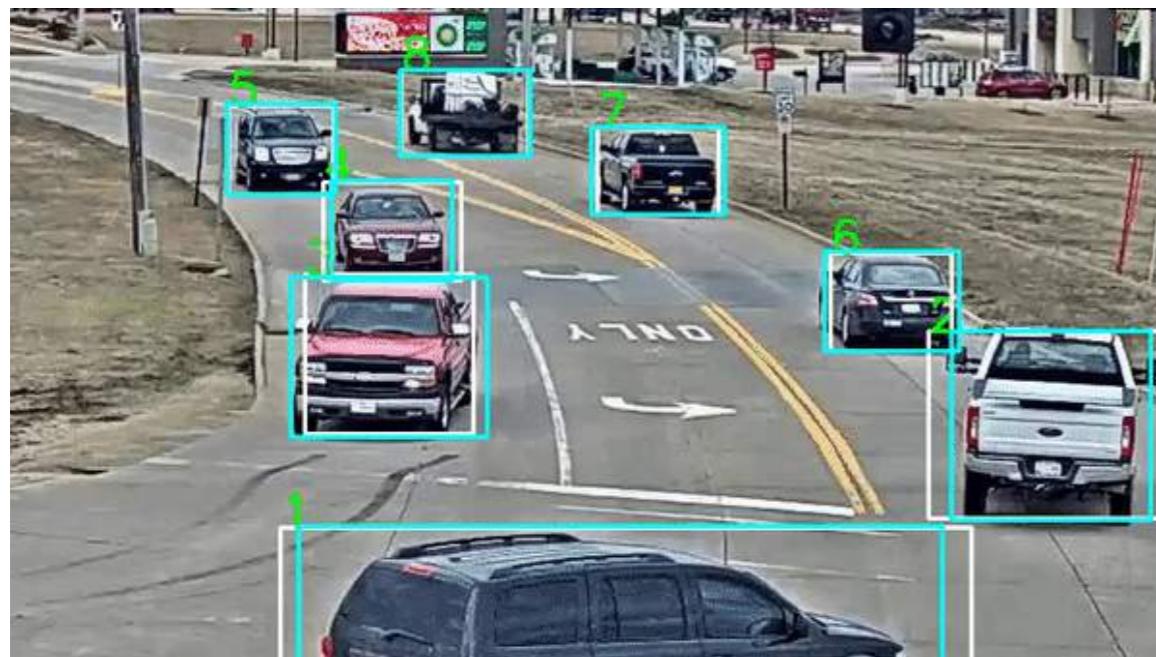
Lecture31

Bhavna R

Recovering motion

- Feature-tracking
 - Extract visual features (corners, textured areas) entire objects and “track” them over multiple frames
- Optical flow
 - Recover image motion at each pixel from spatio-temporal image brightness variations
 - Or within a patch (Patch-based motion estimation)
 - (optical flow)

Tracking using bounding box (object tracking)



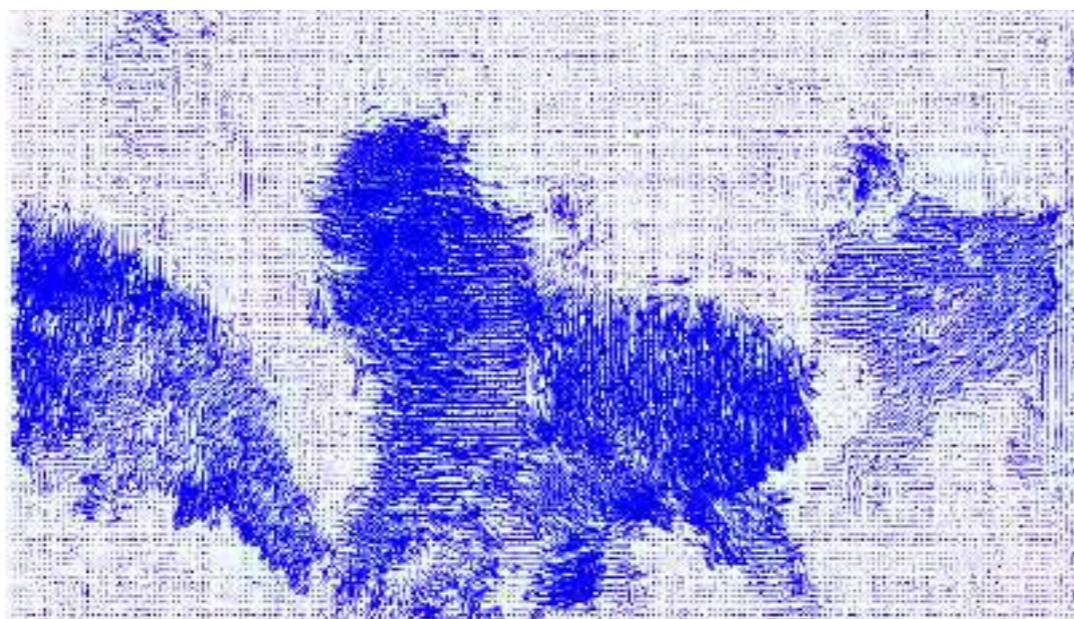
Motion Estimation with optical flow



Optical flow: Pixel motion field as observed in image

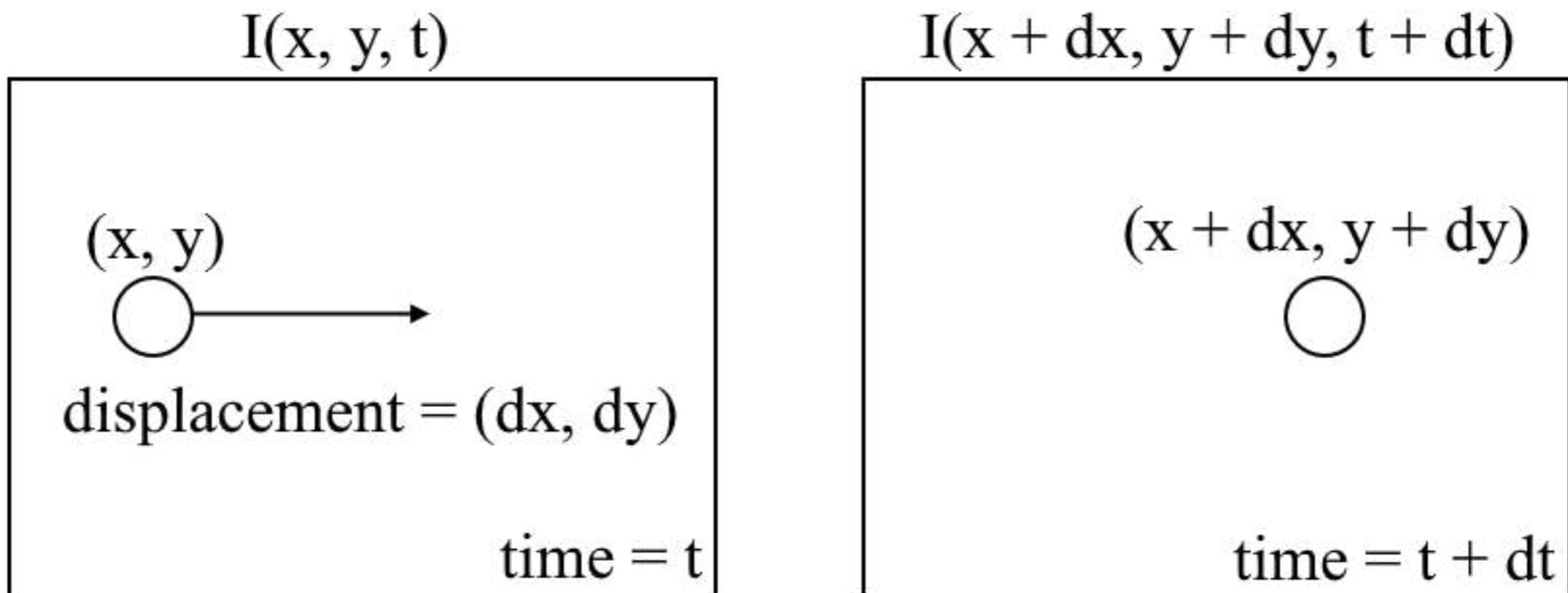
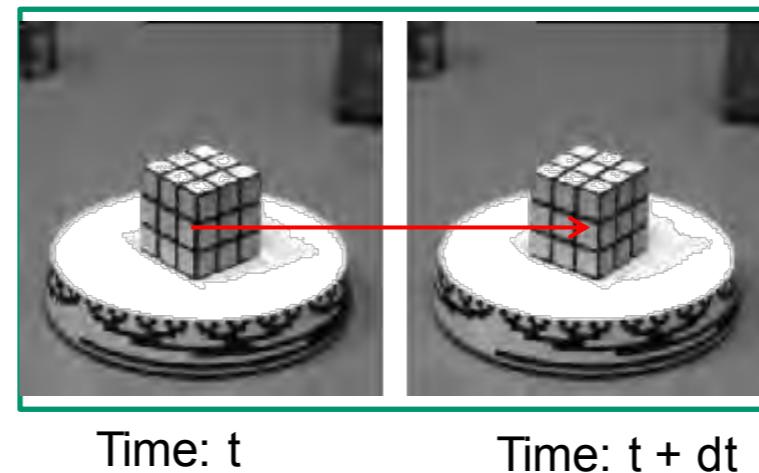


Particularly relevant for
Tracking non-rigid objects



Alper Yilmaz, Fall 2005 UCF

Optic flow setup



Take first image as $I(x,y,t)$ and move its pixels by (dx,dy) over time dt , we obtain the new image: $I(x+dx,y+dy,t+dt)$.

Optic flow constraint equations in 3 steps

We first assume that the point move by very small but the pixel brightness remain constant

1

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

$$I = \text{constant} \Rightarrow \frac{dI}{dt} = 0$$

Second, we take the Taylor Series Approximation of the RHS and remove common terms.

2

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t + \dots$$

$$\Rightarrow \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0$$

Thirdly, we take the derivative w.r.t. time (dI/dt) and apply chain rule

3

$$\frac{\partial I}{\partial x} \frac{\partial x}{\partial t} + \frac{\partial I}{\partial y} \frac{\partial y}{\partial t} + \frac{\partial I}{\partial t} \frac{\partial t}{\partial t} = 0$$

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0$$

$u = \frac{dx}{dt}$	$v = \frac{dy}{dt}$	Velocity in x& y
---------------------	---------------------	---------------------

$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$	Intensity gradient along horizontal & vertical
--	--

We have an equation with image gradients and velocity in x & y respectively

Optic flow equations

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0$$



$$I_x = \frac{\partial I}{\partial x}, \quad I_y = \frac{\partial I}{\partial y}, \quad I_t = \frac{\partial I}{\partial t}$$



$I_x u + I_y v + I_t = 0$

Single equation with 2 unknowns

$$\nabla I \cdot \vec{V} = - \nabla t$$

Optical flow is the relation of the motion field:

the 2D projection of the physical movement of points relative to the observer
to 2D displacement of pixel/ pixel patches on the image plane.

$u = \frac{dx}{dt}, v = \frac{dy}{dt}$ velocity in x& y

$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$ Intensity gradient along horizontal & vertical

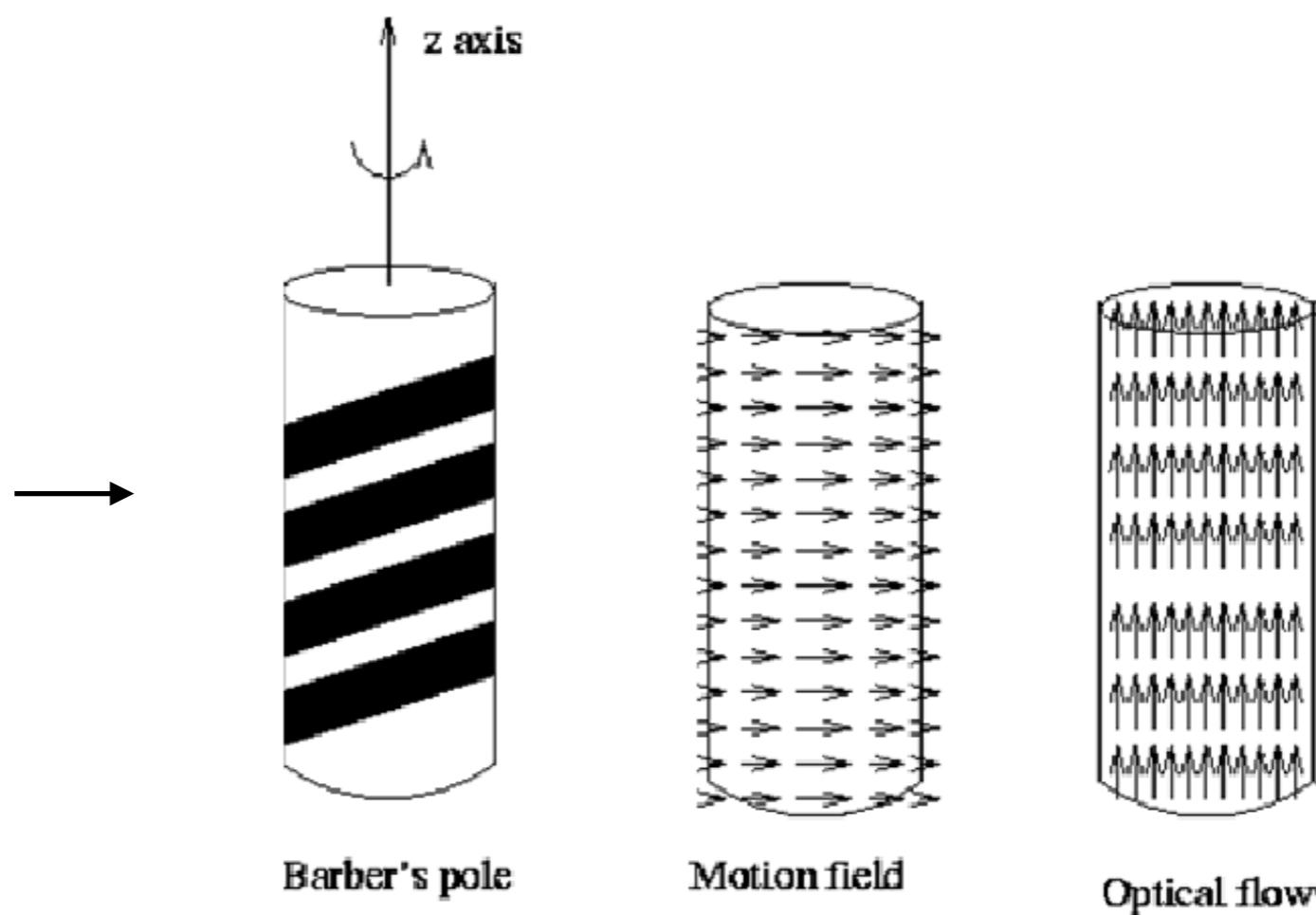
$\frac{\partial I}{\partial t}$ Intensity change over time frame

Optic flow: apparent motion of brightness patterns

Optical Flow vs. Motion: Aperture Problem



barber pole illusion



impossibility to calculate the optic flow vector in a point is called the *aperture problem*

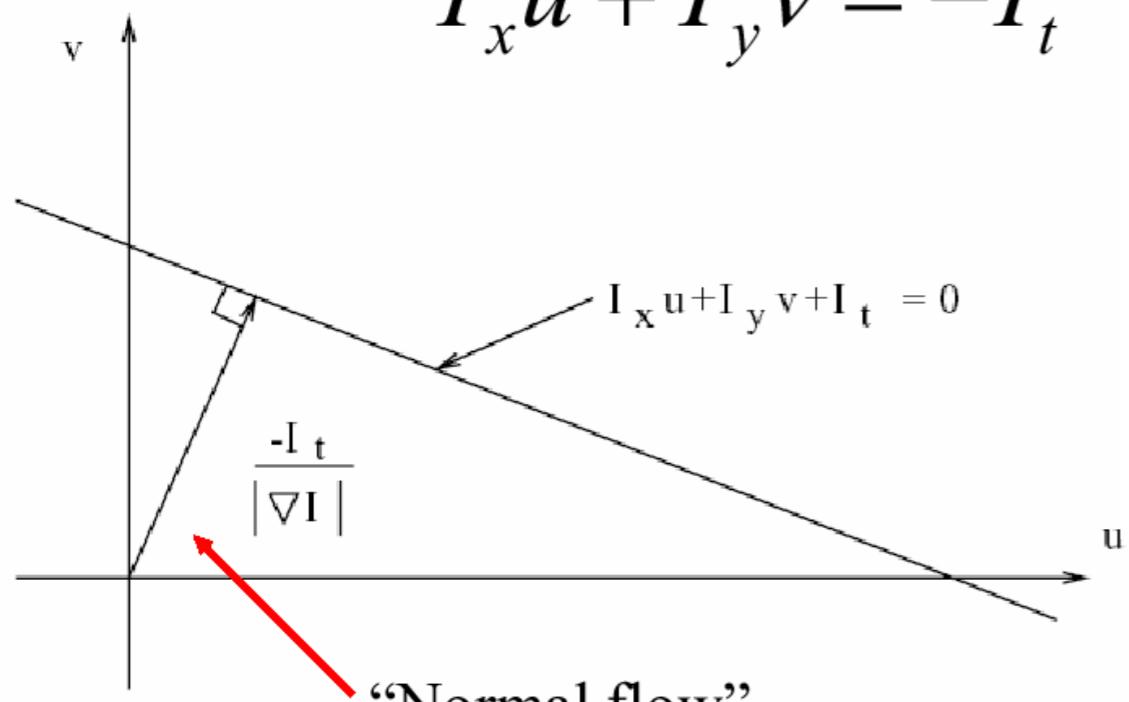


The grating appears to be moving down and to the right, perpendicular to the orientation of the bars. But it could be moving in many other directions, such as only down, or only to the right. It is impossible to determine unless the ends of the bars become visible in the aperture.

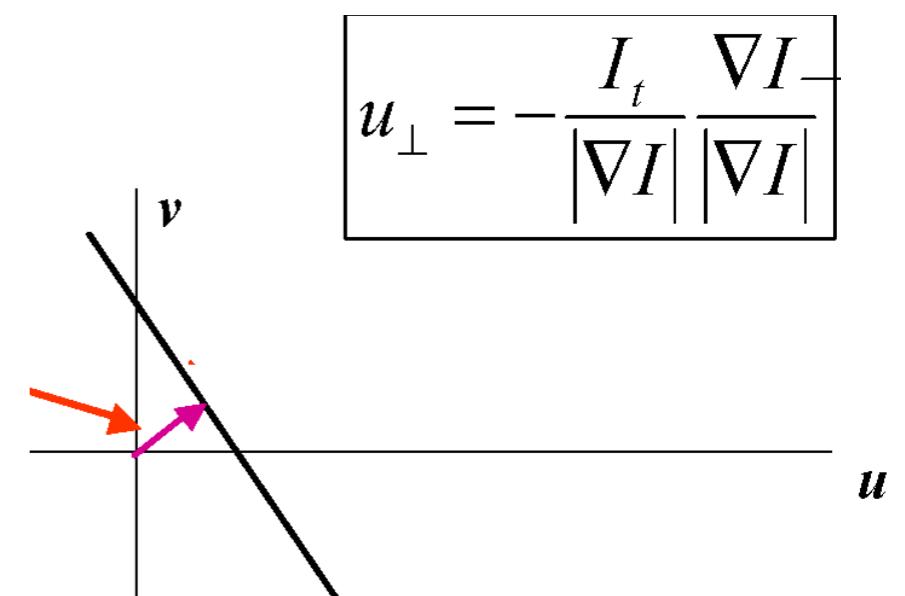
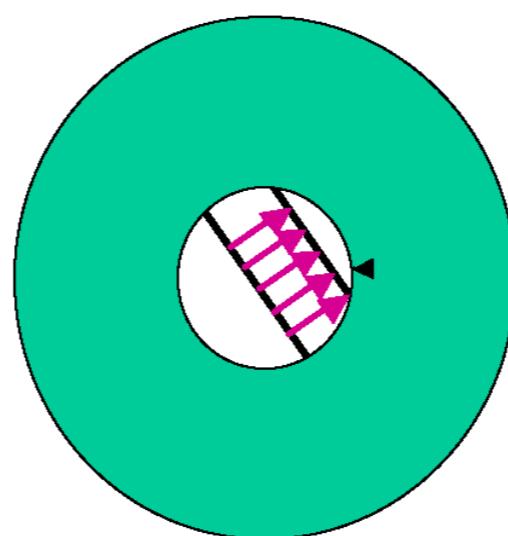
We get at most “Normal Flow” – with one point we can only detect movement perpendicular to the brightness gradient

At a single image pixel, we get a line:

$$I_x u + I_y v = -I_t$$

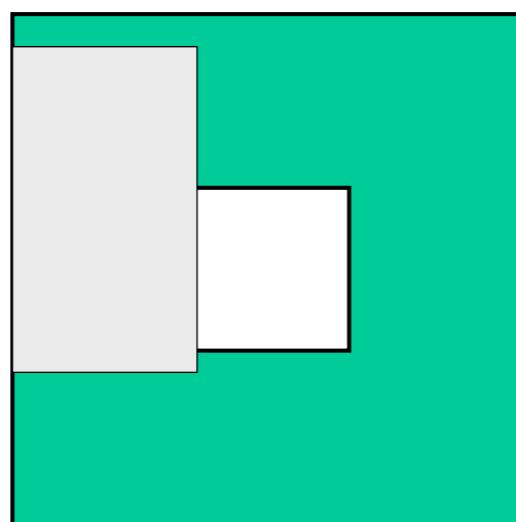


Aperture problem

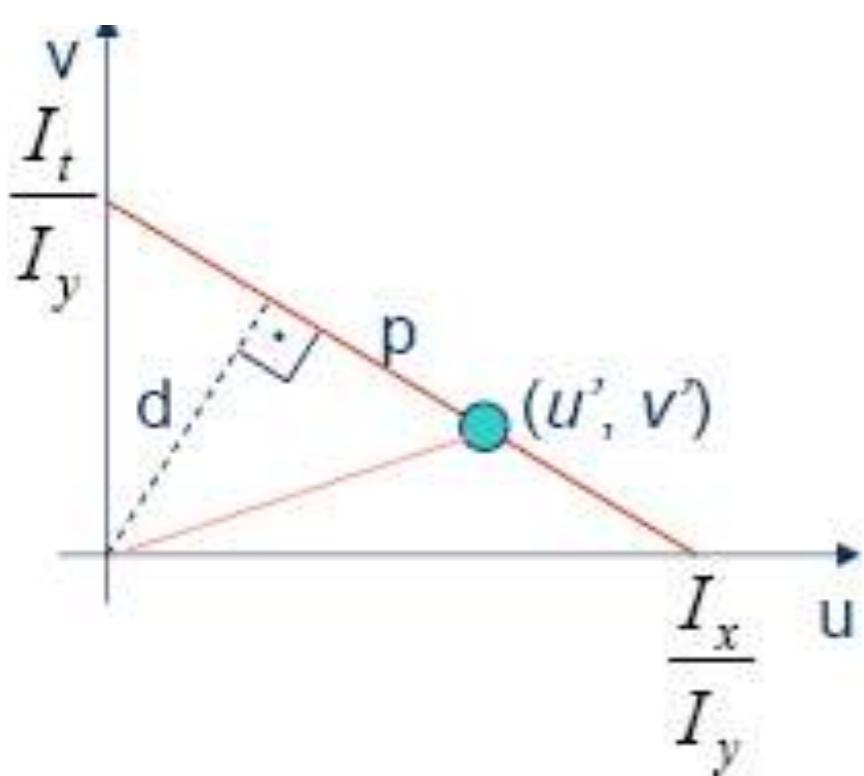
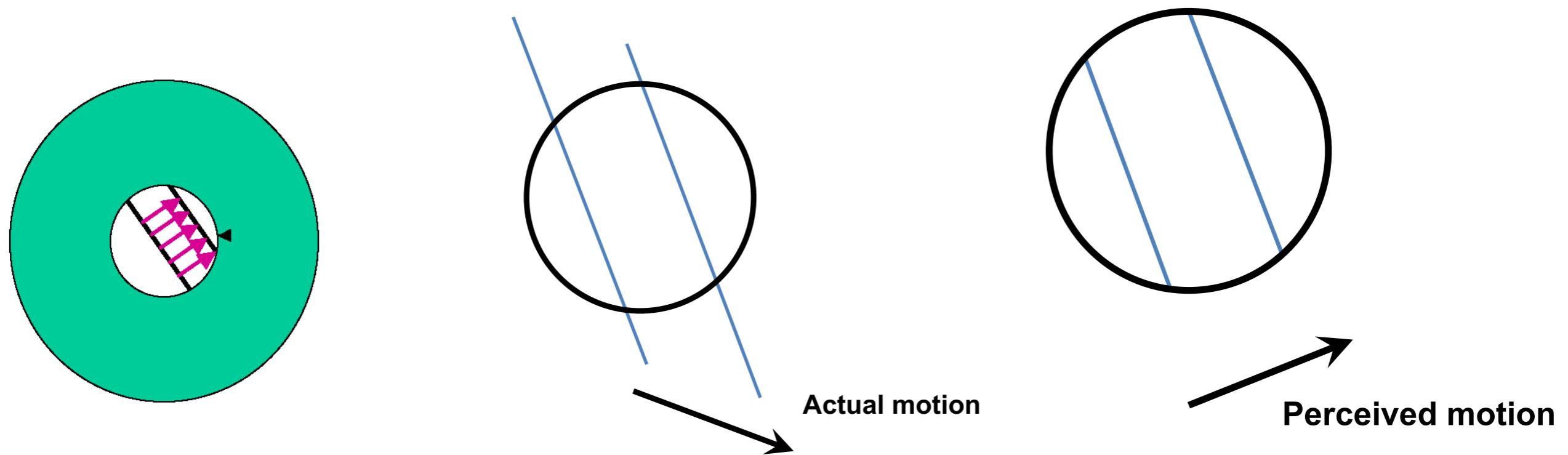


For a line in the (u,v) space within the aperture, we can detect only normal flow

Consequently, motion along
an edge is ambiguous



Aperture problem and normal versus parallel flow



$$v = u \frac{I_x}{I_y} + \frac{I_t}{I_y}$$

- Let (u', v') be true flow
- True flow has two components
 - Normal flow: d
 - Parallel flow: p
- Normal flow **can be** computed
- Parallel flow **cannot**

Methods for computing true optic flow

✓ Lukas and Kanade

- Schunck
- Horn & Schunck

Single equation with 2 unknowns: so we need additional constraints

$$I_x u + I_y v + I_t = 0$$

$$\nabla I \cdot \vec{V} = - \nabla t \quad \text{Single equation with 2 unknowns}$$

How do we get more equations for a pixel?

Lucas-Kannade method:

Combine local constraints on u and v for several pixels within a small window (image patch)

Assume the pixel's neighbors have the same (u, v)

For example: a 5x5 window, gives us 25 equations per pixel

$$\nabla I(p_i) \cdot [u \ v] = - I_t(p_i)$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix} \rightarrow$$

$$\begin{array}{ccc} A & d = b \\ 25 \times 2 & 2 \times 1 & 25 \times 1 \end{array}$$

Integrate over such a patch

$$I_x u + I_y v + I_t = 0$$



$$E(u, v) = \sum_{x, y \in \Omega} (I_x(x, y)u + I_y(x, y)v + I_t)^2$$

Assume a single velocity for all pixels within an image patch

$$\frac{\partial E}{\partial u} = \sum 2I_x(uI_x + vI_y + I_t) = 0$$

We define an energy functional and take derivatives w.r.t. u & v
and equate to 0

$$\frac{\partial E}{\partial v} = \sum 2I_y(uI_x + vI_y + I_t) = 0$$

$$\frac{\partial E}{\partial u} = \sum 2I_x(uI_x + vI_y + I_t) = 0$$

$$\sum uI_x^2 + \sum vI_xI_y + \sum I_xI_t = 0$$

$$u\sum I_x^2 + v\sum I_xI_y = -\sum I_xI_t$$

$$\begin{bmatrix} \sum I_x^2 & \sum I_xI_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\sum I_xI_t$$

$$\frac{\partial E}{\partial v} = \sum 2I_y(uI_x + vI_y + I_t) = 0$$

$$\sum uI_xI_y + \sum vI_y^2 + \sum I_yI_t = 0$$

$$u\sum I_xI_y + v\sum I_y^2 = -\sum I_yI_t$$

$$\begin{bmatrix} \sum I_xI_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\sum I_yI_t$$

$$\begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_xI_t \\ \sum I_yI_t \end{pmatrix}$$

We construct an observation matrix

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

We construct an observation matrix

$$A\mathbf{v} = -\mathbf{b}$$

For more than two points ($n > 2$), this is an overdetermined system and we be solved using a least squares procedure

$$A^T A \mathbf{v} = -A^T \mathbf{b}$$

Multiplying by A^T (A is invertible)

$$\mathbf{v} = - (A^T A)^{-1} A^T \mathbf{b}$$

$$A^T A = \begin{pmatrix} \sum_{i=1}^n f_x^2(\mathbf{x}_i, t) & \sum_{i=1}^n f_x(\mathbf{x}_i, t)f_y(\mathbf{x}_i, t) \\ \sum_{i=1}^n f_x(\mathbf{x}_i, t)f_y(\mathbf{x}_i, t) & \sum_{i=1}^n f_y^2(\mathbf{x}_i, t) \end{pmatrix} \text{Structure tensor}$$

With points 1..n in the neighbourhood

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(p_n)^2 & \sum_i I_x(p_n)I_y(p_n) \\ \sum_i I_x(p_n)I_y(p_n) & \sum_i I_y(p_n)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(p_n)I_t(p_n) \\ -\sum_i I_y(p_n)I_t(p_n) \end{bmatrix}$$

Implementation of LK method

The plain least squares solution gives the same importance to all N pixels in the window. In practice it is usually better to give more weight to the pixels that are closer to the central pixel p.

$$A^T W A V = A^T W b \rightarrow V = (A^T W A)^{-1} A^T W b$$

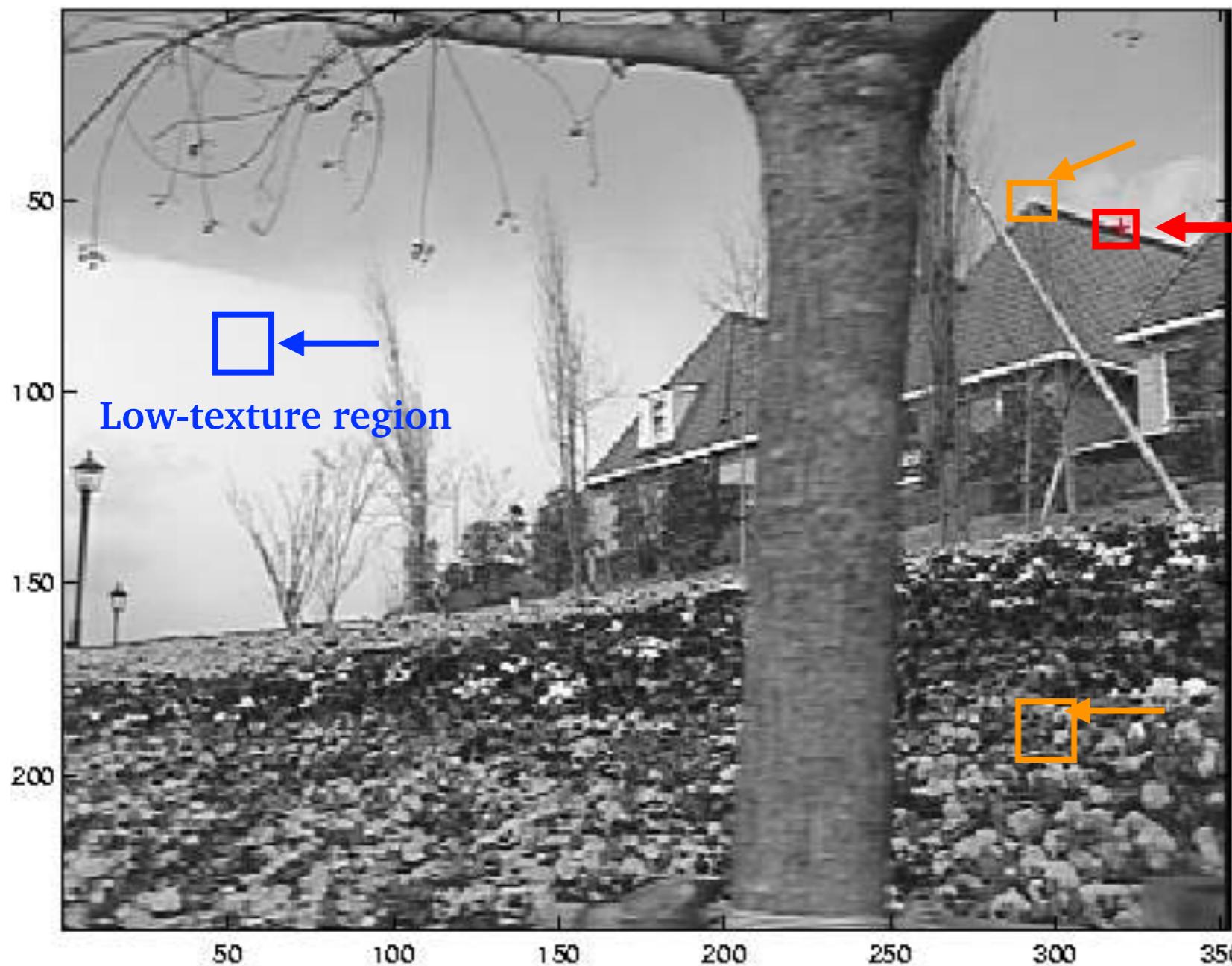
$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i w_i I_x(p_n)^2 & \sum_i w_i I_x(p_n) I_y(p_n) \\ \sum_i w_i I_x(p_n) I_y(p_n) & \sum_i w_i I_y(p_n)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i I_x(p_n) I_t(p_n) \\ -\sum_i w_i I_y(p_n) I_t(p_n) \end{bmatrix}$$

For a weight W that is an $n \times n$ diagonal matrix such that weights $W_{ii} = w_i$ are assigned to the equation of pixel p_n . The weight w_i is typically a Gaussian function of the distance between p_n and p.

$$\sum w^2(x, y) [\nabla I(x, y, t) v + I_t(x, y, t)]^2$$

Lucas and Kanade and others implemented a weighted least squares (LS) fit of local first-order constraints to a constant model for v in each small spatial neighbourhood by minimizing $\sum w(x, y) [v_x I_x(x, y, t) + v_y I_y(x, y, t) + I_t(x, y, t)]^2$ where $w(x, y)$ influence to constraints at the center of the neighbourhood. matrix containing

Values of $\sum w^2(x, y) [\nabla I(x, y, t) \cdot v + It(x, y, t)]^2$ in different image regions



Corner

large λ_1 , large λ_2

edge

- gradients very large or very small
- large λ_1 , small λ_2

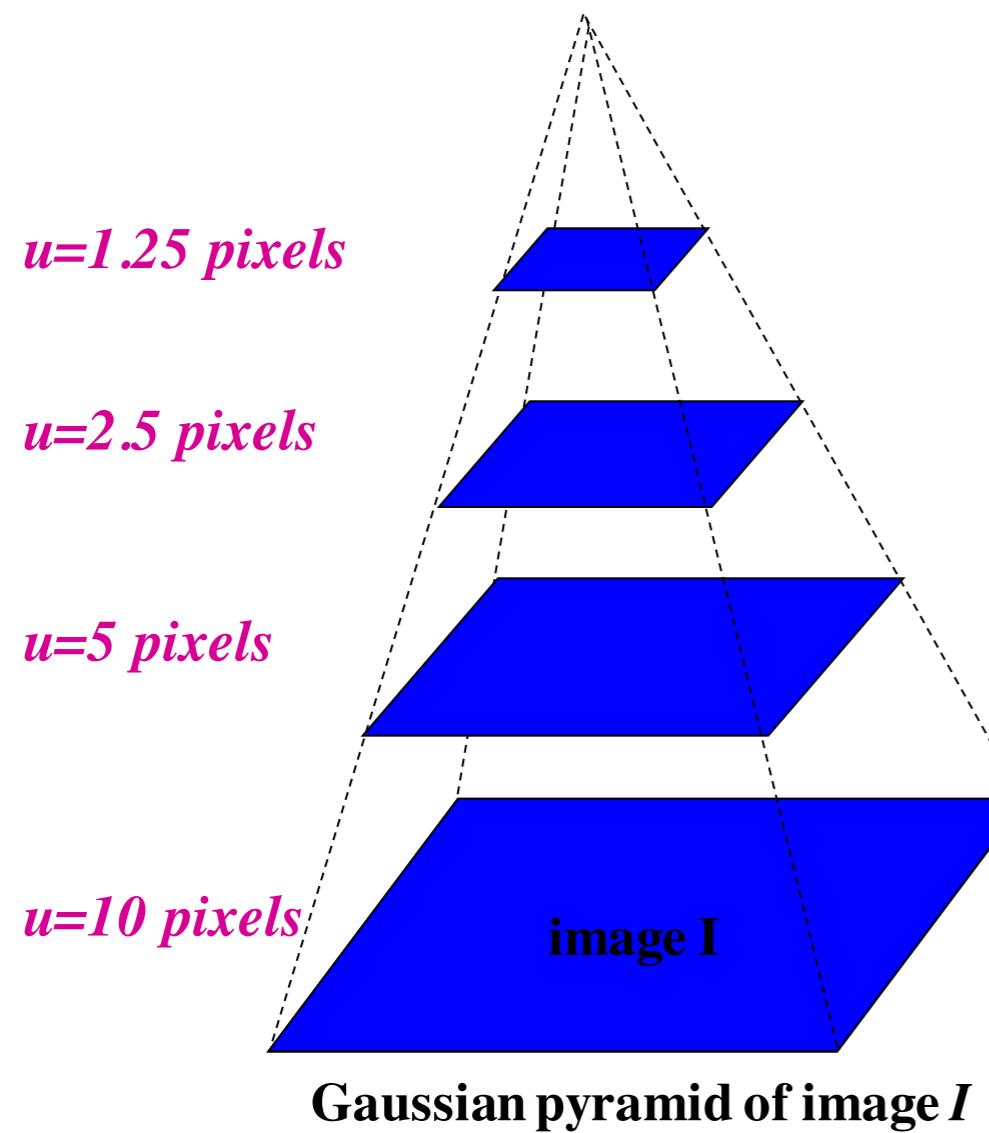
High-texture region

- gradients large and different
- large λ_1 , large λ_2

- gradients have small magnitude

- small λ_1 , small λ_2

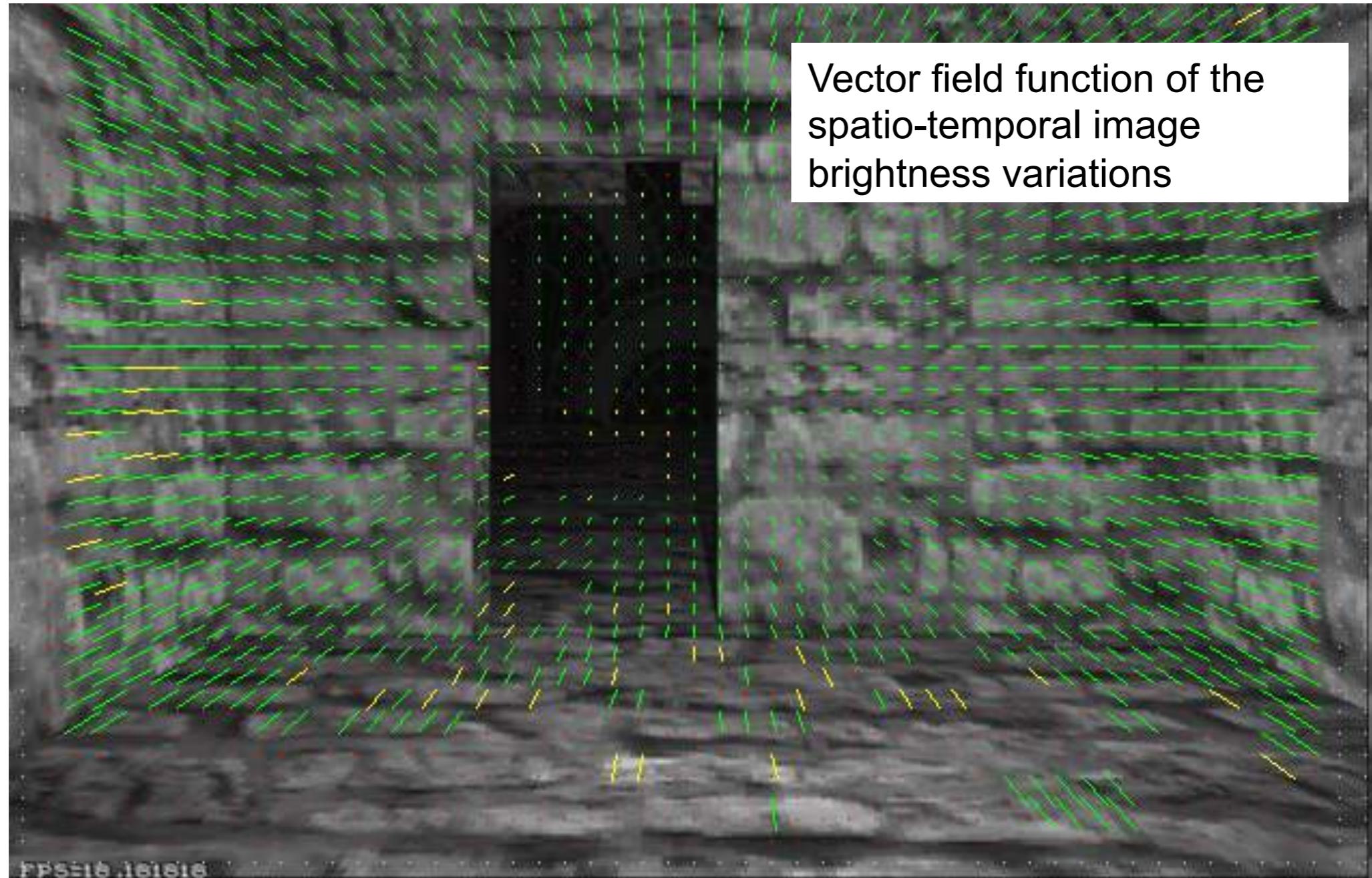
Coarse to fine optical flow estimation: tackling large displacements across time frames



For large displacements (more than say 2 pixels) the method cannot be used as such.

make an image pyramid such that large velocities are found from small displacements in subsampled images.

Motion estimation using Optic flow across time frames



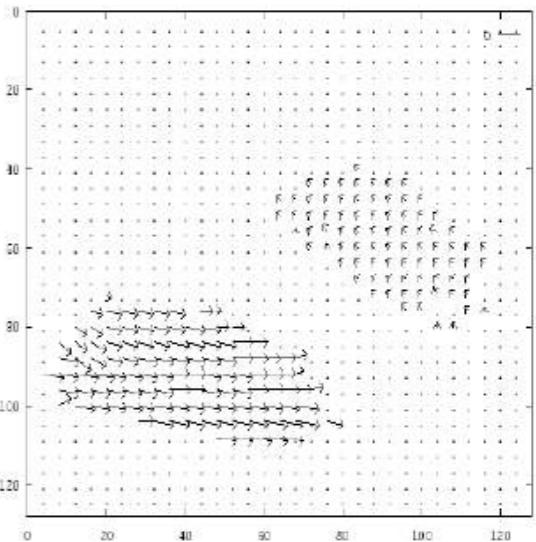
Picture courtesy of Selim Temizer - Learning and Intelligent Systems (LIS) Group, MIT

estimation of velocity vectors of image structures from one frame to another frame in time sequence of 2-D image.

LK optical Flow versus LK feature tracking

Lucas-Kanade Optical Flow:

- computed for each pixel
- works well for textured pixels
- Operations performed across time frames (on entire image)



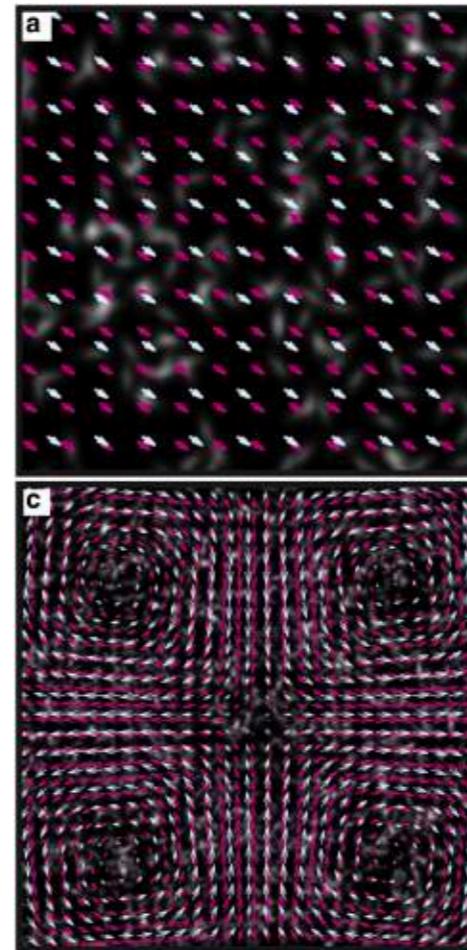
Lucas-Kanade Feature-based methods:

- Find good features using eigenvalues of second-moment matrix (e.g., Harris detector or threshold on the smallest eigenvalue, Shi-Tomasi) in all images across time frames
- This amounts to assuming a translation model for frame-to-frame feature movement
- Use affine registration with first feature patch
- Terminate tracks whose dissimilarity gets too large
- Start new tracks when needed

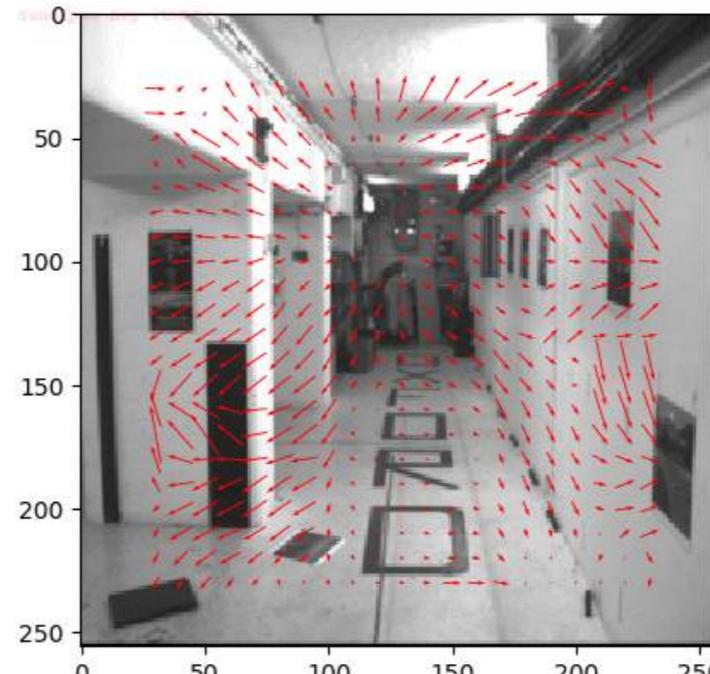
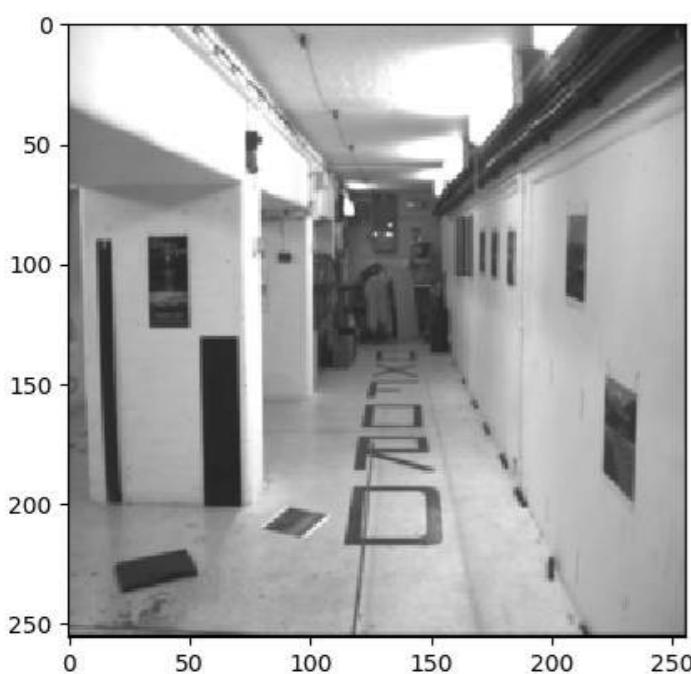
Examples: optic flow

Eg from cell biology

camera fixed, motion flow



Vig et al., Biophysical Journal 2016



Motion field + camera motion

- points closer to the camera move more quickly across the image plane
- Length of flow vectors inversely proportional to depth Z of 3d point

Planer projective transformation

Homography

DSE312-CV

Lecture32

Bhavna R

Quick recap: Camera Calibration

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

Pixel coordinates
Transformation
World coordinates

We estimated the parameters of \mathbf{P} which were
11 intrinsic and extrinsic parameters

Reminder: DLT, SVD, QR decomposition

We mapped 3D world coordinates to 2D image

camera	calibration matrix	#parameters	
unit	${}^0\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	6 (6+0)	<ul style="list-style-type: none"> image plane origin is principal point and distance between camera origin and the image plane is 1.
ideal	${}^k\mathbf{K} = \begin{bmatrix} c & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$	7 (6+1)	<ul style="list-style-type: none"> image plane origin is principal point
Euclidian	${}^p\mathbf{K} = \begin{bmatrix} c & 0 & x_H \\ 0 & c & y_H \\ 0 & 0 & 1 \end{bmatrix}$	9 (6+3)	<ul style="list-style-type: none"> Euclidean sensor in the image plane
affine	$\mathbf{K} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}$	11 (6+5)	<ul style="list-style-type: none"> preserves straight lines
general	${}^a\mathbf{K} = \begin{bmatrix} c & cs & x_H + \Delta x \\ 0 & c(1+m) & y_H + \Delta y \\ 0 & 0 & 1 \end{bmatrix}$	11+N	<ul style="list-style-type: none"> camera with non-linear distortions

We also learnt the decomposition of the transformation matrix into extrinsic and intrinsic parameters

Let us look into our 3D to 2D camera calibration matrix

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Pixel coordinates Transformation Matrix World coordinates

Pixel coordinates Intrinsic Matrix Extrinsic Matrix World coordinates

Intrinsic Matrix Extrinsic Matrix

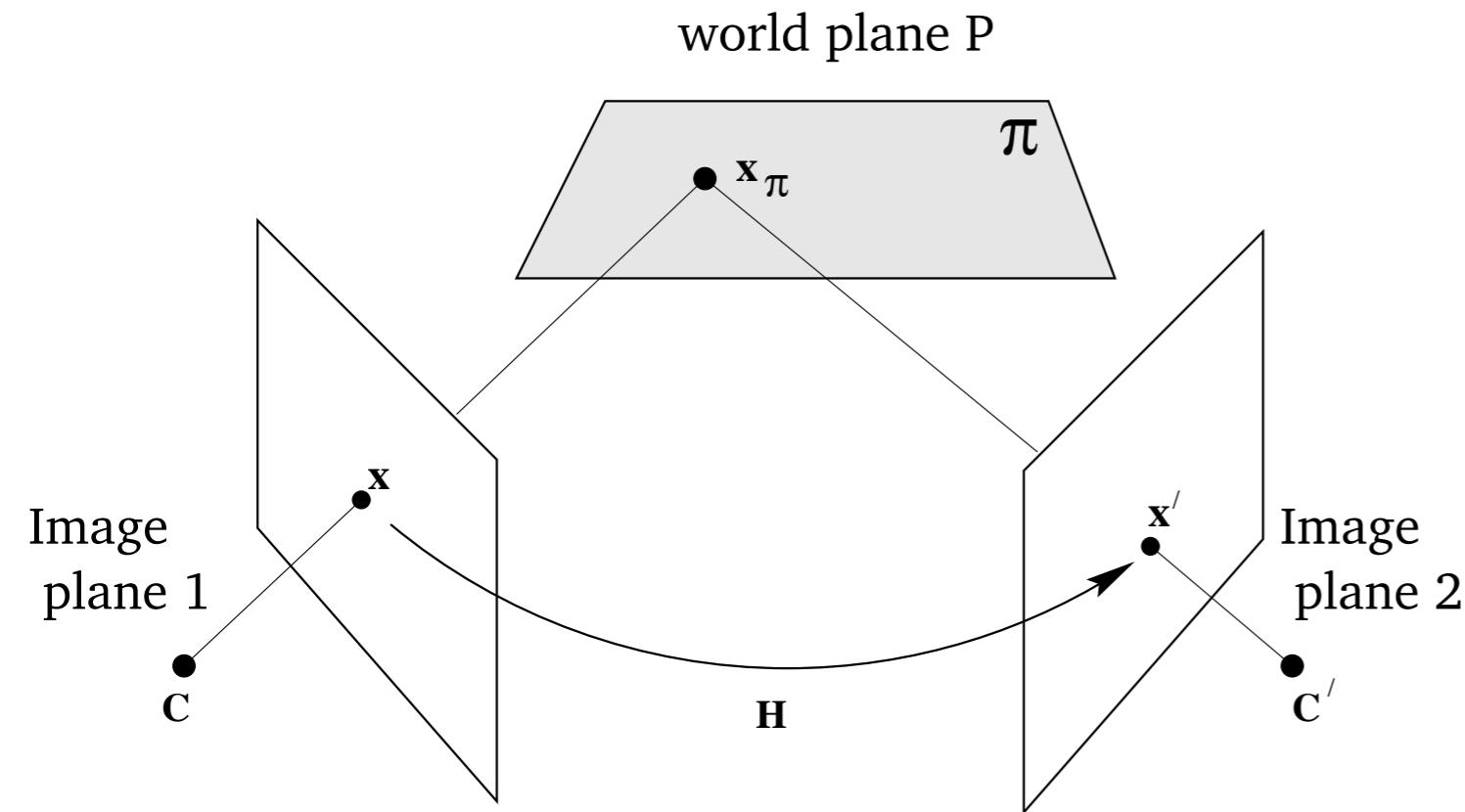
Transformation (P)

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

intrinsic extrinsic

c or f_x (in x), f_y (in y) are the focal lengths
 m is the scale difference
 s is for shear compensation
 x_H , y_H are the coordinate shifts as per the sensor system

Homography induced by a plane



map from x to x' is the homography induced by the plane π

$$x = H_{1\pi} x_\pi$$

between the world plane π and Image plane 1

$$x' = H_{2\pi} x_\pi$$

between the world plane π and Image plane 2

- Homography relates plane 1 to plane P ,
- similarly, we can map plane 2.
- So, by sharing the same centre projection, we can map multiple image planes to a single plane
- For this, we can compute homography between images

The composition of the two perspectivities is a homography, $x' = H_{2\pi} H^{-1}_{1\pi} x = Hx$, between the image planes

$$x' = Hx$$

Homography Conditions

Two images are related by a homography if and only if

- Both images are viewing the same plane from a different angle
- Scene points can be distant (scene plane at infinity)
- Both images are taken from the same camera but from a different angle
- Camera is rotated about its center of projection without any translation
- Note that the homography relationship is independent of the scene structure
- It does not depend on what the cameras are looking at
- Relationship holds regardless of what is seen in the images

In homography, the Z coordinate of each point is equal to zero

For a given point in a 3D world, the equations are:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Pixel coordinates intrinsic extrinsic World coordinates

Transformation matrix

↓

For a 2D point on a plane, the equations are:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cancel{r_{13}} \\ r_{21} & r_{22} & \cancel{r_{23}} \\ r_{31} & r_{32} & \cancel{r_{33}} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ \cancel{Z} \\ 1 \end{bmatrix}$$

so, we remove the rotation in the 3rd dimension of the extrinsic parameter and the Z-coordinate from the world coordinates vector

Simplification from one 2D plane to another

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3] = \underbrace{\begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}}_{\text{As a 3-column vector}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}]}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

One pair of points observed generates such an equation

Setting Up the equations for determining the parameters

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[r_1, r_2, t] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

H_{3×3}

One pair of points observed generates such an equation

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = H_{3\times 3} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \quad i = 1, \dots, I$$

Several pairs of points generates such an equation

H is unknown here

- We have to estimate a **3x3 homography** (H) instead of a 3x4 projection matrix
- Rest remains identical
- Solving the system of linear equations leads to an estimate of H

How many points are needed to estimate \mathbf{H} ?

- Solving the system of linear equations leads to an estimate of \mathbf{H}
- We need to identify at least 4 points as \mathbf{H} has 8 DoF (since \mathbf{H} is defined upto a scale factor) and each point consists of 2 observations (x and y)
- We will then estimate \mathbf{K} (intrinsic) from \mathbf{H}

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \underset{3 \times 3}{\mathbf{H}} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \quad i = 1, \dots, I \quad \boxed{\mathbf{x}' = \mathbf{H} \mathbf{x}} \quad \rightarrow \quad \begin{aligned} & \text{Let } (x', y') = (x'_2, y'_2) \\ & (x, y) = (x_1, y_1) \end{aligned}$$

$$\begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \quad \rightarrow$$

$$\boxed{\mathbf{x}' = \mathbf{H} \mathbf{x}}$$

$$\begin{pmatrix} x'_2 \\ y'_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} \quad \rightarrow$$

$$x'_2(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13}$$

$$y'_2(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23}$$

We re-arrange these equations just for 1 pair (just like we did for the projection matrix):

$$\begin{aligned} \mathbf{a}_x^T \mathbf{h} &= 0 \\ \mathbf{a}_y^T \mathbf{h} &= 0 \end{aligned} \quad \rightarrow$$

$$\begin{aligned} \mathbf{h} &= (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \\ \mathbf{a}_x &= (-x_1, -y_1, -1, 0, 0, 0, x'_2x_1, x'_2y_1, x'_2)^T \\ \mathbf{a}_y &= (0, 0, 0, -x_1, -y_1, -1, y'_2x_1, y'_2y_1, y'_2)^T. \end{aligned}$$

$$\rightarrow \boxed{A\mathbf{h} = \mathbf{0}}$$

$$\rightarrow A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix}$$

We write the equation for several such pairs
We need at least 4 points

H is a homogeneous matrix

H has 8 DoF (since **H** is defined unto a scale factor) and each point consists of 2 observations in x and y ($\mathbf{a}_x, \mathbf{a}_y$)

$$A\mathbf{h} = \mathbf{0}$$

$$A = \begin{pmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \vdots \\ \mathbf{a}_{xN}^T \\ \mathbf{a}_{yN}^T \end{pmatrix}$$

We write the equation for several such pairs
We need at least 4 points

We need to solve a Homogeneous Linear Least Squares (SVD)

$$A = U\Sigma V^\top = \sum_{i=1}^9 \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$$

(Please check the previous lectures for SVD)

σ_i : Singular values sorted from 1 to 9 s.t. σ_9 is the smallest value

If the homography is exactly determined, then $\sigma_9=0$, and there exists a homography that fits the points exactly.

If the homography is overdetermined, then $\sigma_9 \geq 0$. Here σ_9 represents a “residual” or goodness of fit.

Given a matrix A with SVD decomposition $A = U\Sigma V^\top$, the columns of V correspond to the eigenvectors of $A^T A$.

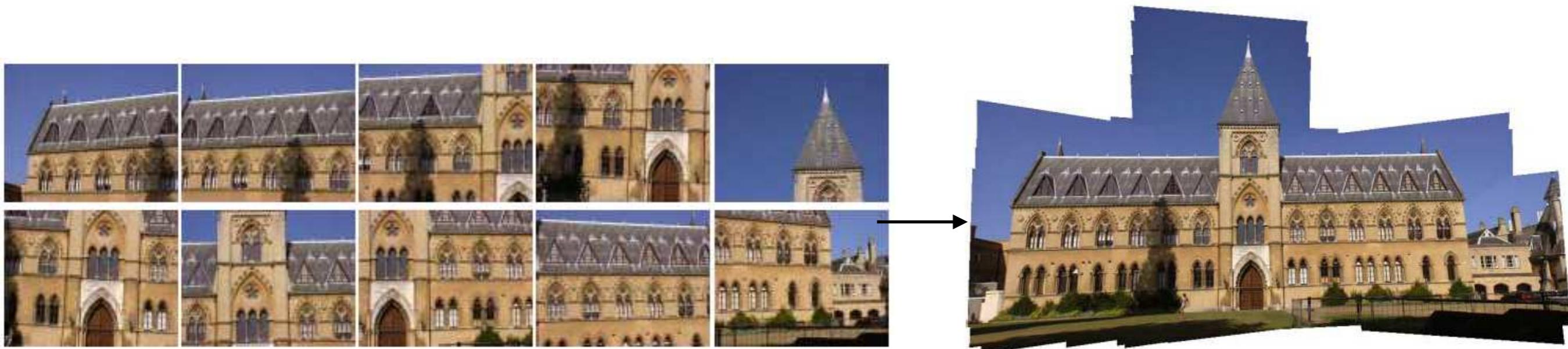
- From the SVD we take the “right singular vector” (a column from V) which corresponds to the **smallest singular value**, σ_9 .
- This is the solution, h, which contains the coefficients of the homography matrix that best fits the points.
We reshape h into the matrix H, and form the equation, s.t.

$$\mathbf{x}' \sim H \mathbf{x}$$

→ Now we have H

Homography Applications:

mosaics - building a wide angle image by stitching together several images taken under different orientations from the same position
(image stitching)



Using a few points it is possible to compute the homography matrix

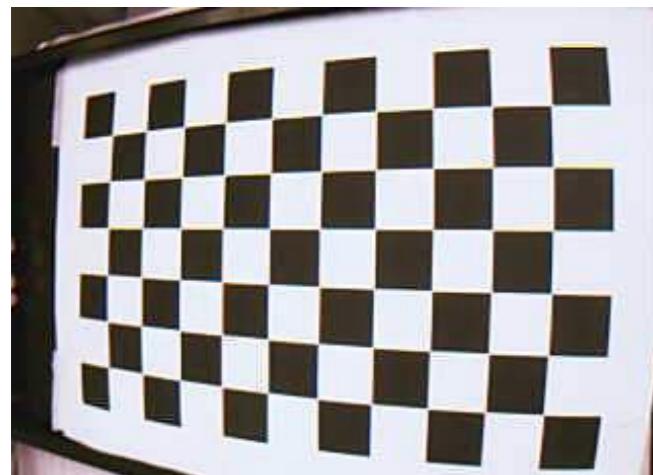
- Employs image-image mappings removing perspective distortion (computer vision) : Requires computing homographies between an image and scene surfaces
- Employs image-scene mappings : rendering textures (computer graphics) : Requires applying homographies between a planar scene surface and the image plane, having the camera as the center of projection

Camera Calibration with Homography Estimation

Camera intrinsic parameters

Zhang's method of camera calibration

Estimate the calibration matrix of the camera while not taking into account where the camera is positioned
So, we need only intrinsic parameters of the camera



- For instance, we can use a checkerboard pattern for calibration for which know the size and the structure
- We can glue pattern on a flat surface (sitting on a plane) and using a camera we take an image of such a pattern
- **Then we can use corner points and interest points from this checkerboard (you can do this now, you studied feature extraction)**

we know where points are located on the checkerboard and it is a flat surface, although we do not their location in the 3D world



- Set the world coordinate system to the corner of the checkerboard
- All points on the checkerboard lie in the X/Y plane, i.e., $Z=0$

After we have estimated \mathbf{H} , we need to compute \mathbf{K} from \mathbf{H} :

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3] = \underbrace{\begin{bmatrix} c & cs & x_H \\ 0 & c(1+m) & y_H \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Estimated using SVD}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}]}$$

Previously, we used QR decomposition that gives a rotation matrix & triangular matrix, we cannot apply QR anymore!

$$[\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3] = \mathbf{K}[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}]$$

We want \mathbf{K} (intrinsic parameter),
 \mathbf{K} is invertible

$$[\mathbf{r}_1, \mathbf{r}_2, \mathbf{t}] = \mathbf{K}^{-1}[\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$$

Since \mathbf{K} is invertible, we multiply by \mathbf{K}^{-1}

$$\boxed{\mathbf{r}_1 = \mathbf{K}^{-1}\mathbf{h}_1 \quad \mathbf{r}_2 = \mathbf{K}^{-1}\mathbf{h}_2}$$

$t = \mathbf{K}^{-1}\mathbf{h}_3$ Let us first retrieve \mathbf{K}

$\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ form an orthonormal basis as they come from a rotation matrix

Define Constraints \rightarrow

$$\boxed{\mathbf{r}_1^T \mathbf{r}_2 = 0 \quad \|\mathbf{r}_1\| = \|\mathbf{r}_2\| = 1 \longrightarrow \begin{aligned} \|\mathbf{r}_1\| &= \mathbf{r}_1^T \mathbf{r}_1 \\ \|\mathbf{r}_2\| &= \mathbf{r}_2^T \mathbf{r}_2 \end{aligned}}$$

$$so, \mathbf{r}_1^T = \mathbf{h}_1^T \mathbf{K}^{-T}$$

$$\underbrace{\mathbf{h}_1^T}_{\mathbf{r}_1^T} \underbrace{\mathbf{K}^{-T} \mathbf{K}^{-1}}_{\mathbf{r}_2} \mathbf{h}_2 = 0$$

$$\mathbf{h}_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_1 = \mathbf{h}_2^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2$$

We can relate \mathbf{H} and \mathbf{K} using the constraints by inserting \mathbf{r}_1 and \mathbf{r}_2 into the constraint equations:

$$\mathbf{h}_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_1 - \mathbf{h}_2^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2 = 0$$

We can relate \mathbf{H} and \mathbf{K} using the constraints:

$$\mathbf{h}_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2 = 0$$

$$\mathbf{h}_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_1 - \mathbf{h}_2^T \mathbf{K}^{-T} \mathbf{K}^{-1} \mathbf{h}_2 = 0$$

$$\begin{aligned} \mathbf{h}_1^T \underline{\mathbf{K}^{-T} \mathbf{K}^{-1}} \mathbf{h}_2 &= 0 \\ \mathbf{h}_1^T \underline{\mathbf{K}^{-T} \mathbf{K}^{-1}} \mathbf{h}_1 - \mathbf{h}_2^T \underline{\mathbf{K}^{-T} \mathbf{K}^{-1}} \mathbf{h}_2 &= 0 \end{aligned}$$

Define symmetric and positive definite matrix

$$\mathbf{B} := \mathbf{K}^{-T} \mathbf{K}^{-1}$$

$$\begin{aligned} \mathbf{h}_1^T \underline{\mathbf{B}} \mathbf{h}_2 &= 0 \\ \mathbf{h}_1^T \underline{\mathbf{B}} \mathbf{h}_1 - \mathbf{h}_2^T \underline{\mathbf{B}} \mathbf{h}_2 &= 0 \end{aligned}$$

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

We can use the Cholesky decomposition that decomposes such a matrix into (triangular matrices):

For a matrix \mathbf{B} cholesky gives:

$$\text{chol}(\mathbf{B}) = \mathbf{A} \mathbf{A}^T$$

So, if we know \mathbf{B} , we can compute \mathbf{K}
 \mathbf{K} is invertible

$$\mathbf{B} := \mathbf{K}^{-T} \mathbf{K}^{-1}$$

$$\mathbf{A} = \mathbf{K}^{-T}$$

Inspecting equations above:

- The symmetric matrix \mathbf{B} consists of the unknowns
- \mathbf{h} are known
- We have 2 equations that relate \mathbf{B} and \mathbf{h}

Determine B (to get K)

symmetric and positive definite matrix

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix}$$

Construct a system of linear equations:

$$\nabla b = 0$$

H is known

$$h_1^T \underline{B} h_2 = 0$$

$$h_1^T \underline{B} h_1 - h_2^T \underline{B} h_2 = 0$$

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

$$v_{ij} : \text{the coefficients of } b \quad v_{12}^T b = 0$$

(first constraint)

$$r_1^T r_2 = 0$$

$$v_{11}^T b - v_{22}^T b = 0 \quad [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$$

(second constraint)

$$\|r_1\| = \|r_2\| = 1$$

Let the i^{th} column vector of \mathbf{H} be $h_i = [h_{1i}, h_{2i}, h_{3i}]^T$. Then, we have

$$h_i^T B h_j = v_{ij}^T b$$

$$\nabla = \left(\begin{array}{c} v_{12}^T \\ v_{11}^T - v_{22}^T \end{array} \right) \quad \text{with} \quad v_{ij} = \begin{bmatrix} h_{1i}h_{1j} \\ h_{1i}h_{2j} + h_{2i}h_{1j} \\ h_{3i}h_{1j} + h_{1i}h_{3j} \\ h_{2i}h_{2j} \\ h_{3i}h_{2j} + h_{2i}h_{3j} \\ h_{3i}h_{3j} \end{bmatrix}$$

Elements of H (only using the 1st
2 columns)

v_{ij} is a 6-dimensional row vector

$$V = \begin{pmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \end{pmatrix} \quad \text{with} \quad v_{ij} = \begin{bmatrix} h_{1i}h_{1j} \\ h_{1i}h_{2j} + h_{2i}h_{1j} \\ h_{3i}h_{1j} + h_{1i}h_{3j} \\ h_{2i}h_{2j} \\ h_{3i}h_{2j} + h_{2i}h_{3j} \\ h_{3i}h_{3j} \end{bmatrix}$$

$$\begin{pmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \end{pmatrix} b = 0 \quad \text{For 1 single image}$$

For multiple images, we stack the matrices to a $2n \times 6$ matrix

image 1 → $\begin{pmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \\ \dots \\ v_{12}^T \\ v_{11}^T - v_{22}^T \end{pmatrix} b = 0$

image n → $\begin{pmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \\ \dots \\ v_{12}^T \\ v_{11}^T - v_{22}^T \end{pmatrix} b = 0$

where V is a $2n \times 6$ matrix. If $n \geq 3$, we will have in general a unique solution b defined up to a scale factor.

We need to solve the linear system $Vb = 0$ $b^* = \arg \min_b \|Vb\|$ with $\|b\| = 1$ to obtain b and then K

Summary

- We need at least 4 points per plane to compute the matrix H
- Compute the matrix H using SVD for each image
- Each plane gives us 2 equations for B

$$\begin{aligned} h_1^T \underline{B} h_2 &= 0 \\ h_1^T \underline{B} h_1 - h_2^T \underline{B} h_2 &= 0 \end{aligned}$$

- Since B has 5/6 DoF, we need at least 3 different views of a plane
- Solve:

$$\nabla b = 0$$

- Using SVD, find the eigenvector corresponding to the minimum eigenvalue
- Then use the cholesky decomposition on B to compute K

$$H_{3 \times 3} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \quad \begin{matrix} \downarrow \\ [h_1, h_2, h_3] \end{matrix}$$

$$\begin{array}{c} \text{image 1} \xrightarrow{\left(\begin{array}{c} v_{12}^T \\ v_{11}^T - v_{22}^T \\ \dots \\ v_{12}^T \\ v_{11}^T - v_{22}^T \end{array} \right)} b = 0 \\ \text{image n} \xrightarrow{\left(\begin{array}{c} v_{12}^T \\ v_{11}^T - v_{22}^T \end{array} \right)} \end{array}$$

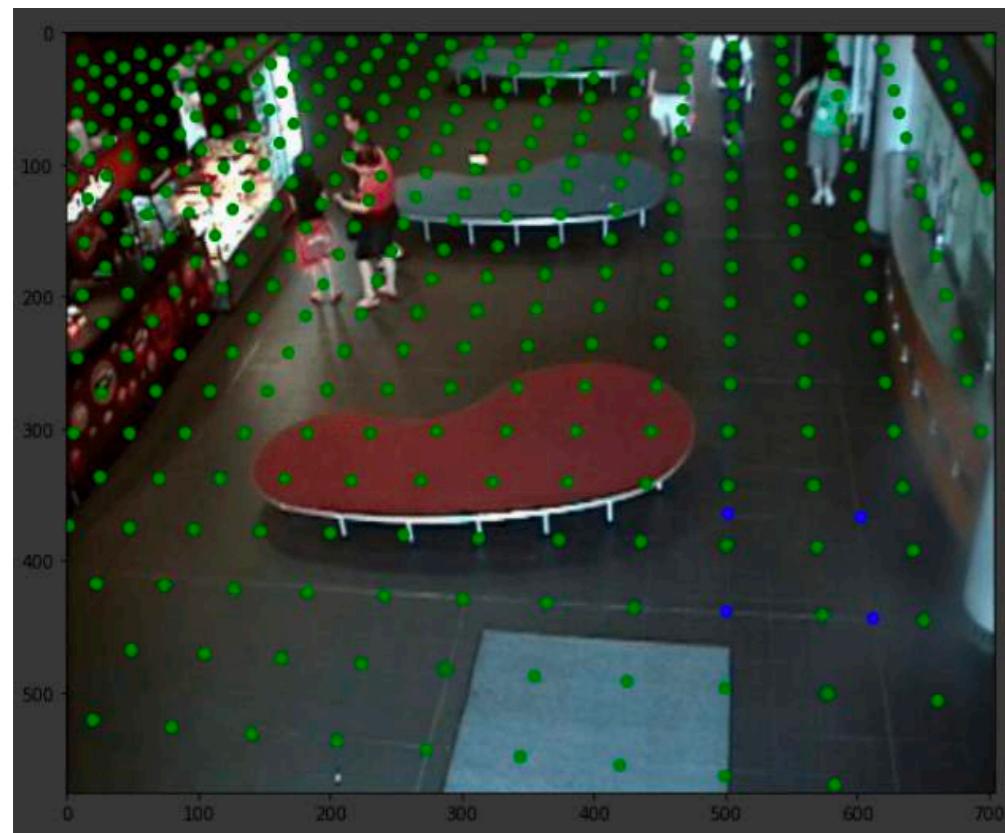


Take at least 3 images at different views

Extract points from the checkerboard

- Each plane will have its own H
- 2 equations for B for each image
- Solve for B by stacking all the points and using SVD
- cholesky decomposition on B to compute K

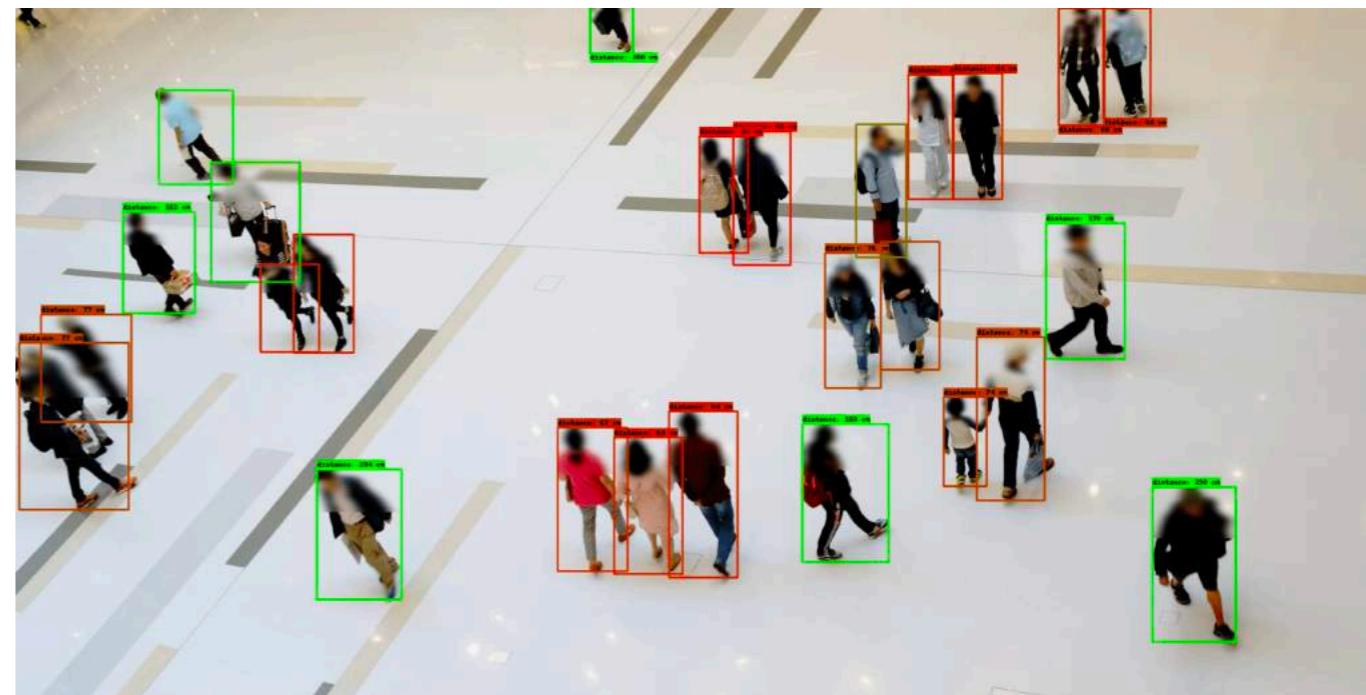
Some recent applications of camera calibration using homography



Camera calibration using homography estimation sample output. The four blue dots are the reference points to calculate matrix \mathbf{H} , and the green dots show the ground plane

Notice the distance between the green points as we move farther away from the camera

Real-world distances between the people calculated using the homography estimation camera calibration method



Part1: RANSAC

Part2: Epipolar Geometry

DSE312-CV

Lecture33

Bhavna R

RANSAC

RANSAC –Random Sample Consensus

Approach

- Find the best partition of points in inlier set and outlier and estimate the model from the inlier set
- Trial-and-error approach
- Approach to deal with high fractions of outliers in the data
- A Standard approach for fitting in the presence of outliers

RANSAC algorithm

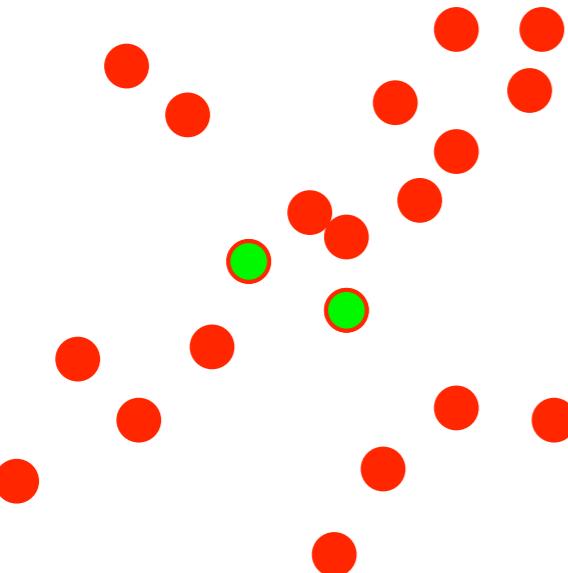
- Sample the number of data points required to fit the model
- Compute model parameters using the sampled data points
- Score by the fraction of inliers within a preset threshold of the model
- Repeat 1-3 until the best model is found with high confidence

Application

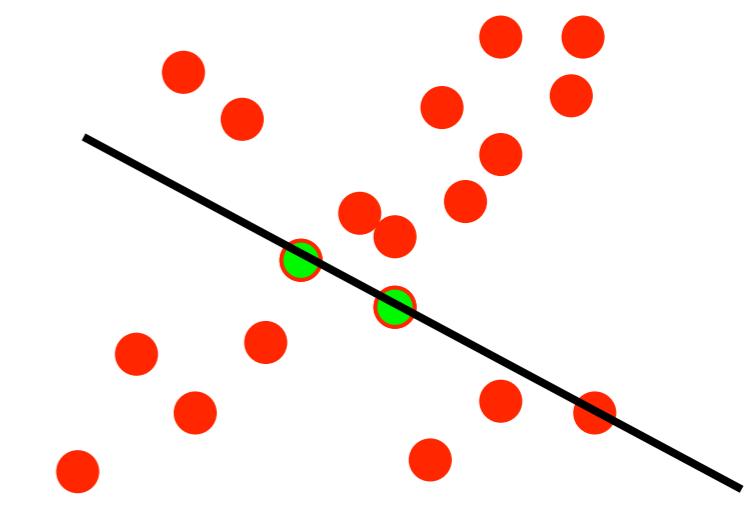
- Removing outlier features
- Useful for feature matching for finding similarities (SIFT, SURF, keypoint descriptors)
- Finding point correspondences
- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)

RANSAC Line fitting example

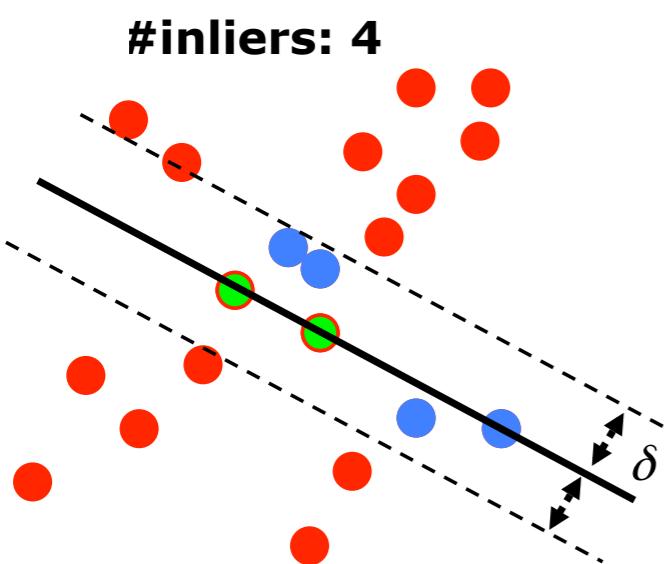
- 1 Sample the number of data points required to fit the model (here: 2 points)



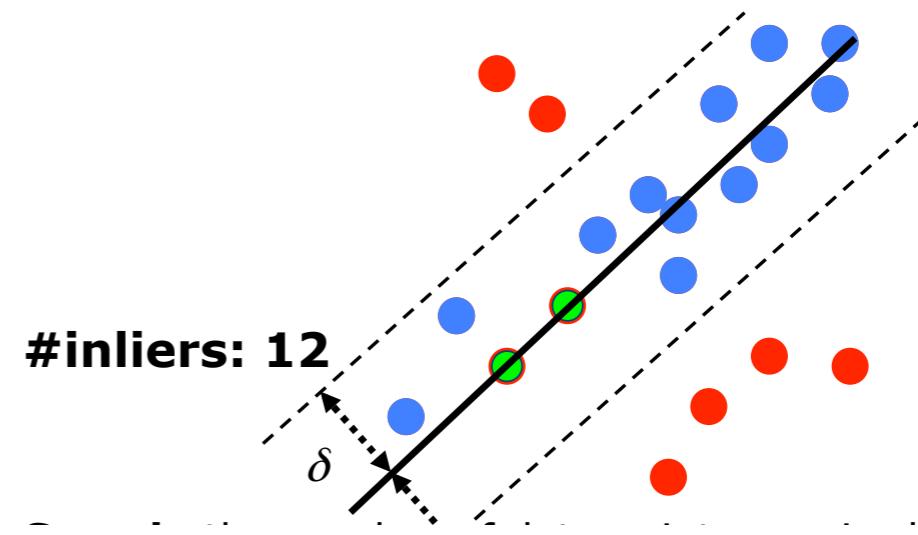
- 2 Compute model parameters using the samples



- 3 Score by the fraction of inliers within a preset threshold of the model



- 4 Repeat 1-3 until the best model is found



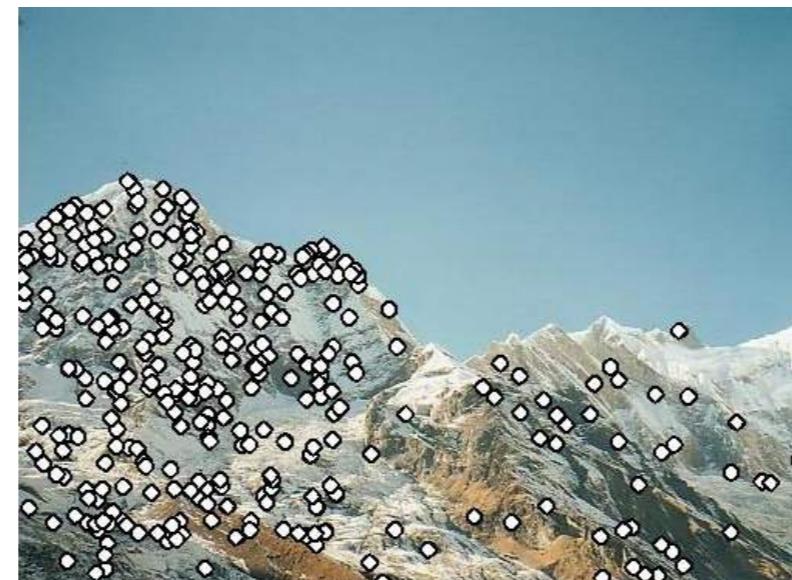
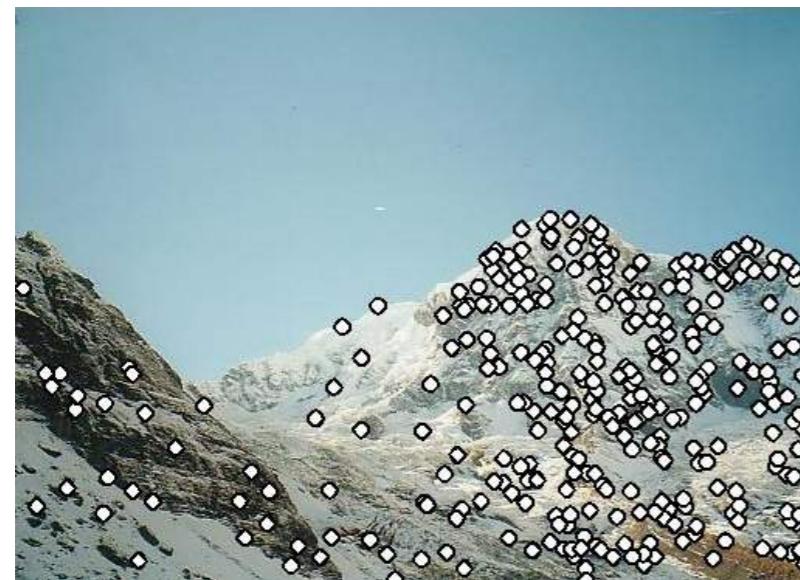
takes random samples from the data using the sample size and uses a fit function to maximize the number of inliers within maxDistance.

Image stitching problem using feature based alignment: RANSAC

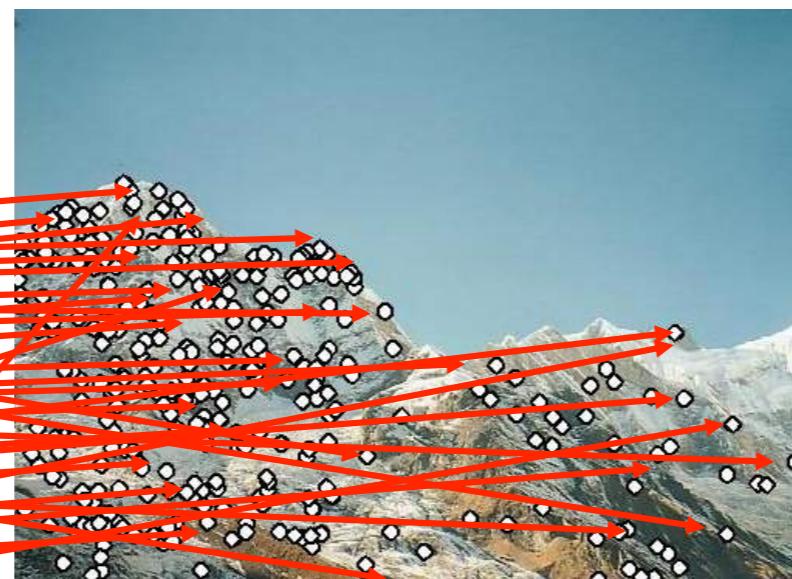
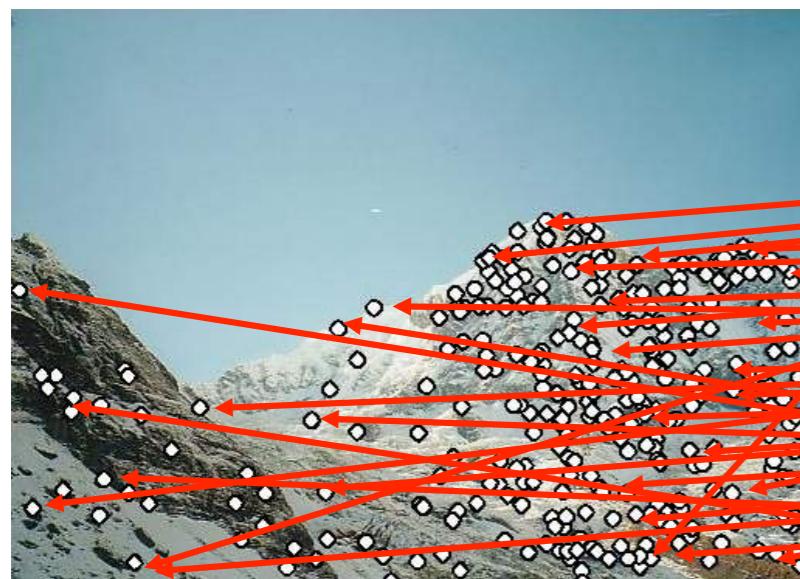


Image stitching problem

Two images that have some overlap

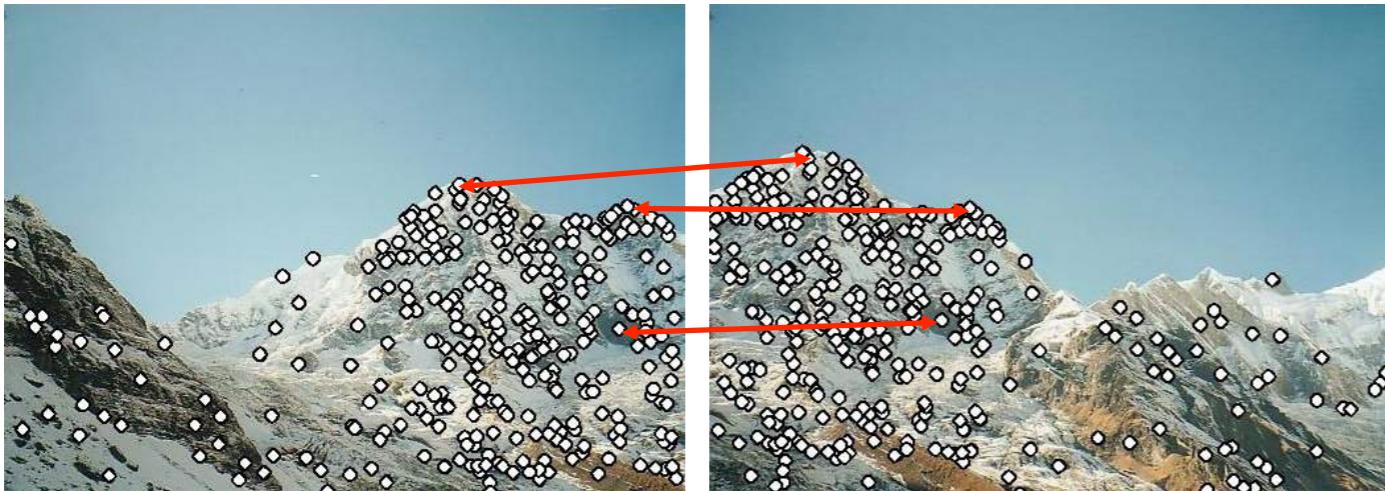


Extract features
(SIFT, SURF)

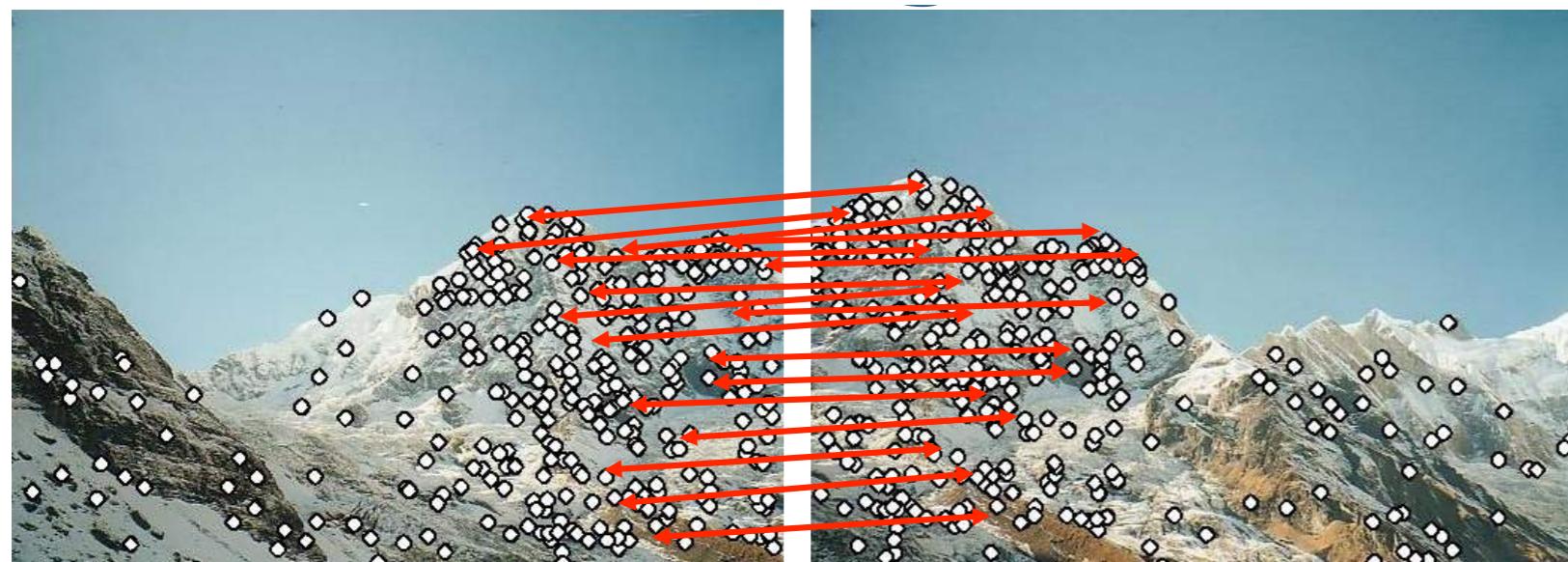


Compute putative matches

Image Stitching



Hypothesise transformation T (translation) and choose 3 point pairs



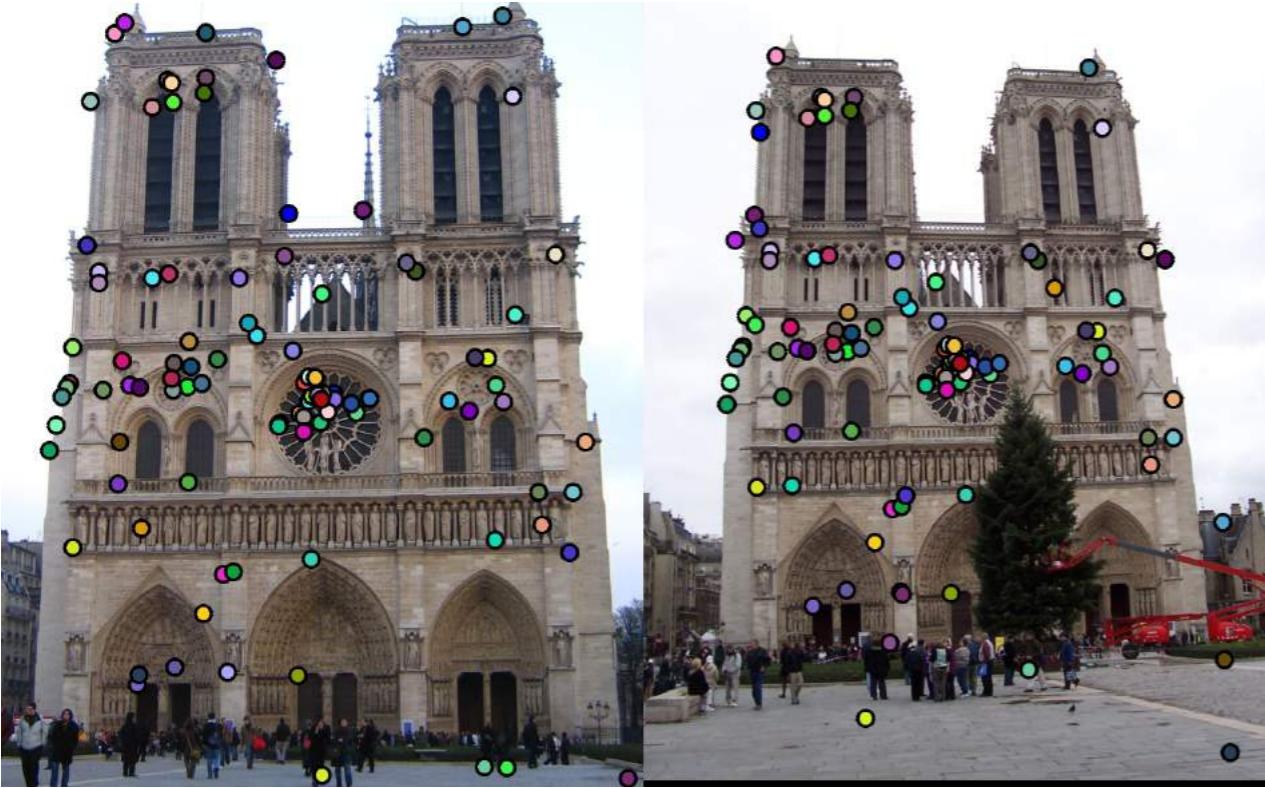
Verify transformation (search for other matches consistent with T) iteratively



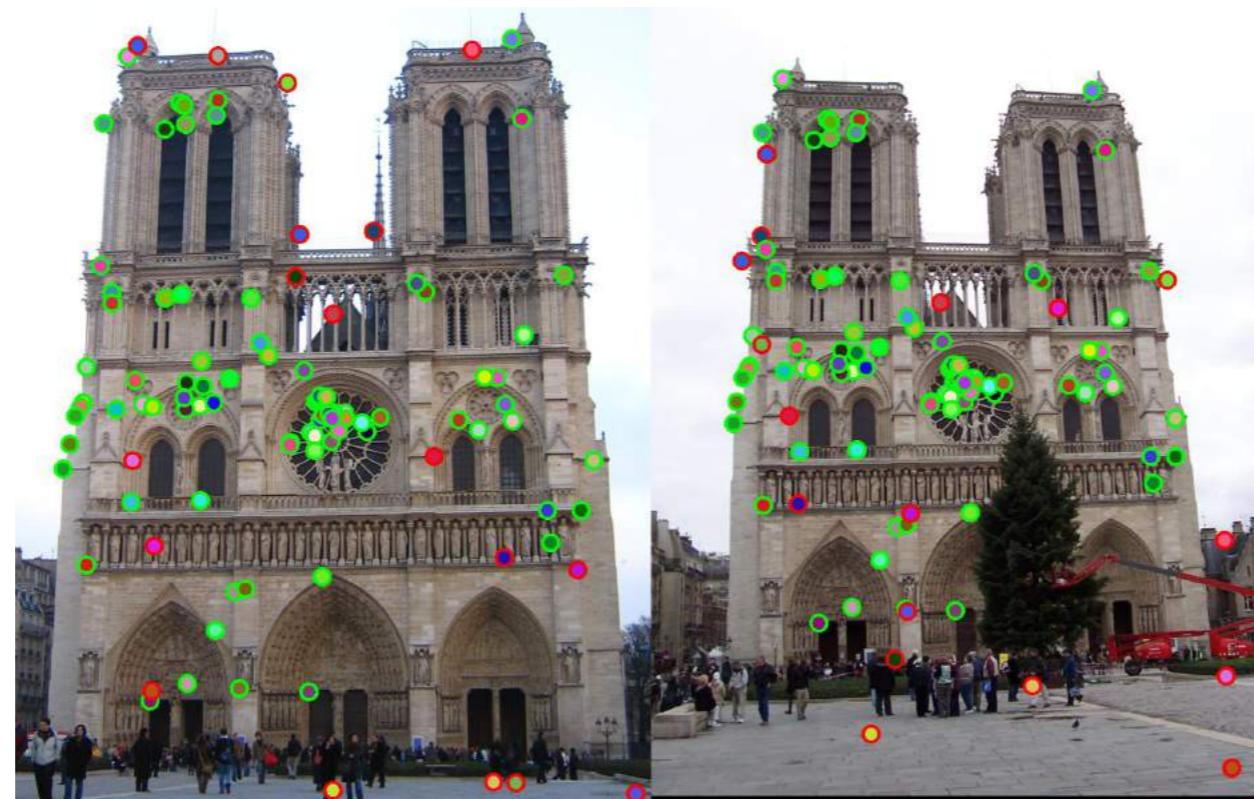
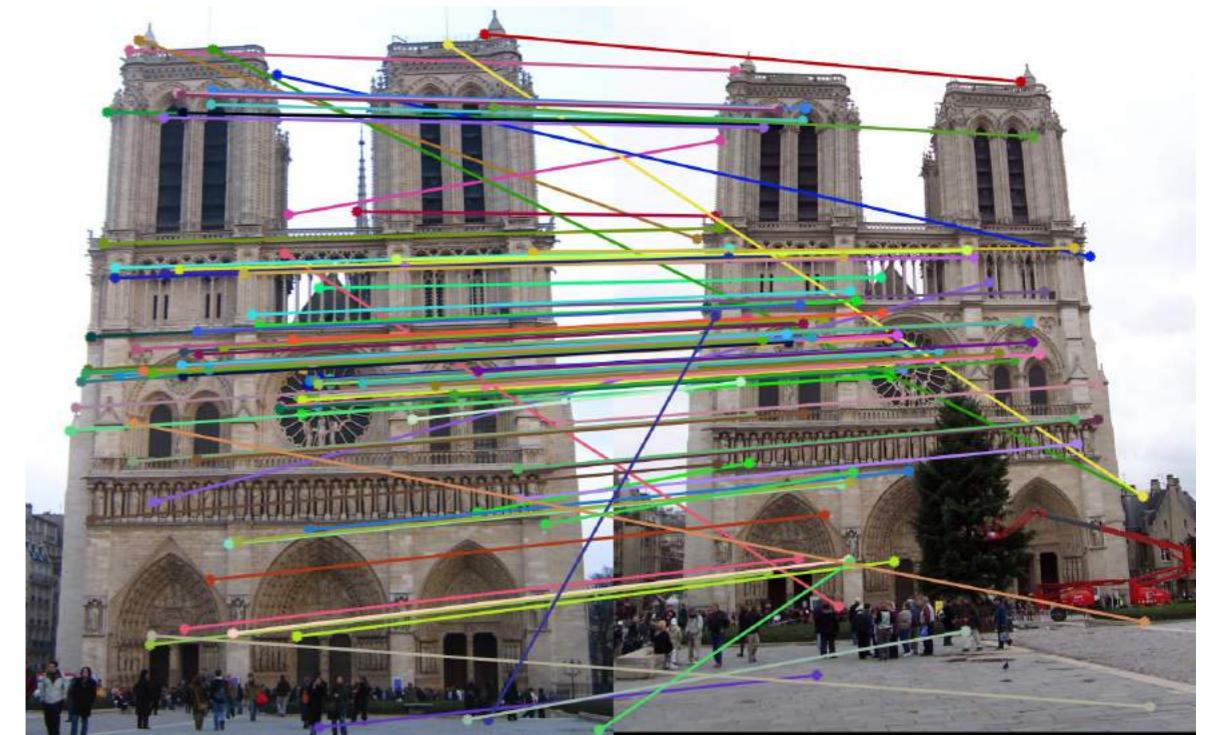
Find the correct overlap for image stitching
Correct for shading

Keypoint descriptors: identify inliers and outliers

Harris corner key points



keypoint matches



After RANSAC,
Remove outliers

RANSAC: choosing the right parameters

- Number of sampled points s (minimum number needed to fit the model)
- Estimation algorithms require different numbers of sampled points
- The smaller s , the better, especially with high outlier ratios
- Outlier ratio e ($e = \# \text{outliers} / \# \text{datapoints}$)
- Number of trials T :
 - Choose T so that, with probability p , at least one random sample set is free from outliers
 - $p(\text{fail once}) = \text{do not select only inliers}$
 - $p(\text{fail } T \text{ times}) = \text{select at least one outlier in all } T \text{ trials}$
- Distance threshold δ
- Choose δ such that a good point with noise is likely (e.g., prob=0.95) within threshold

$$1 - p = 1 - (1 - e)^s \rightarrow \log(1 - p) = T \log(1 - (1 - e)^s) \rightarrow T = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

Pros

- Robustly deal with outliers
- Works well for 1 to roughly 10 parameters (depending on the number of outliers)
- Easy to implement and understand

Cons

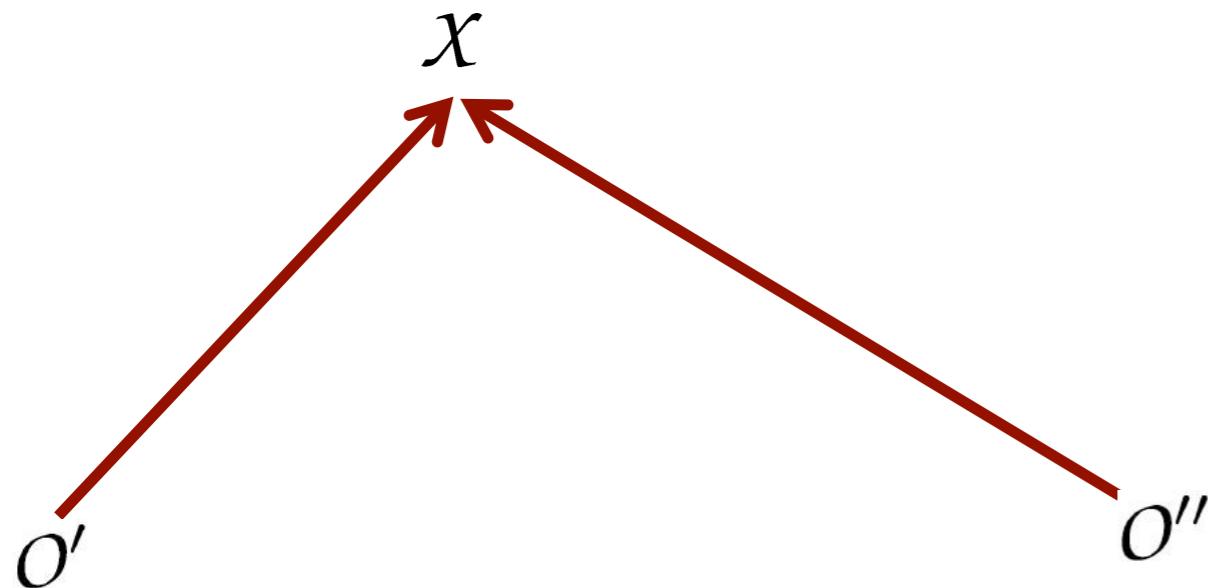
- Computational time grows quickly with fraction of outliers and number of parameters needed to fit the model
- Not good for getting multiple fits

Epipolar Geometry



Stereo cameras: a pair of cameras

Epipolar geometry : coplanarity constraint



Given a point X in the plane of the first image

Find the corresponding point x in the second image plane

Intersection of two corresponding rays

The rays lie in one plane in 3D

O' and O'' are the projection centres of the 2 cameras

They follow what we term as coplanarity constraint

- Epipolar geometry is used to describe geometric relations in image pairs
- Enables efficient search for and prediction of corresponding points
- Given a straight-line preserving mapping, the search space reduces from 2D (whole image) to 1D (line)

Homography versus epipolar geometry

If two cameras are translated then epipolar geometry holds

It maps a point in one image to a line in the other image

- Actual matching point depends on the depth of that point, that is its distance from the camera
- As the translations $\rightarrow 0$ the base line b of the two cameras also goes to zero
- Once it is zero epipolar geometry does not hold any longer

Homography

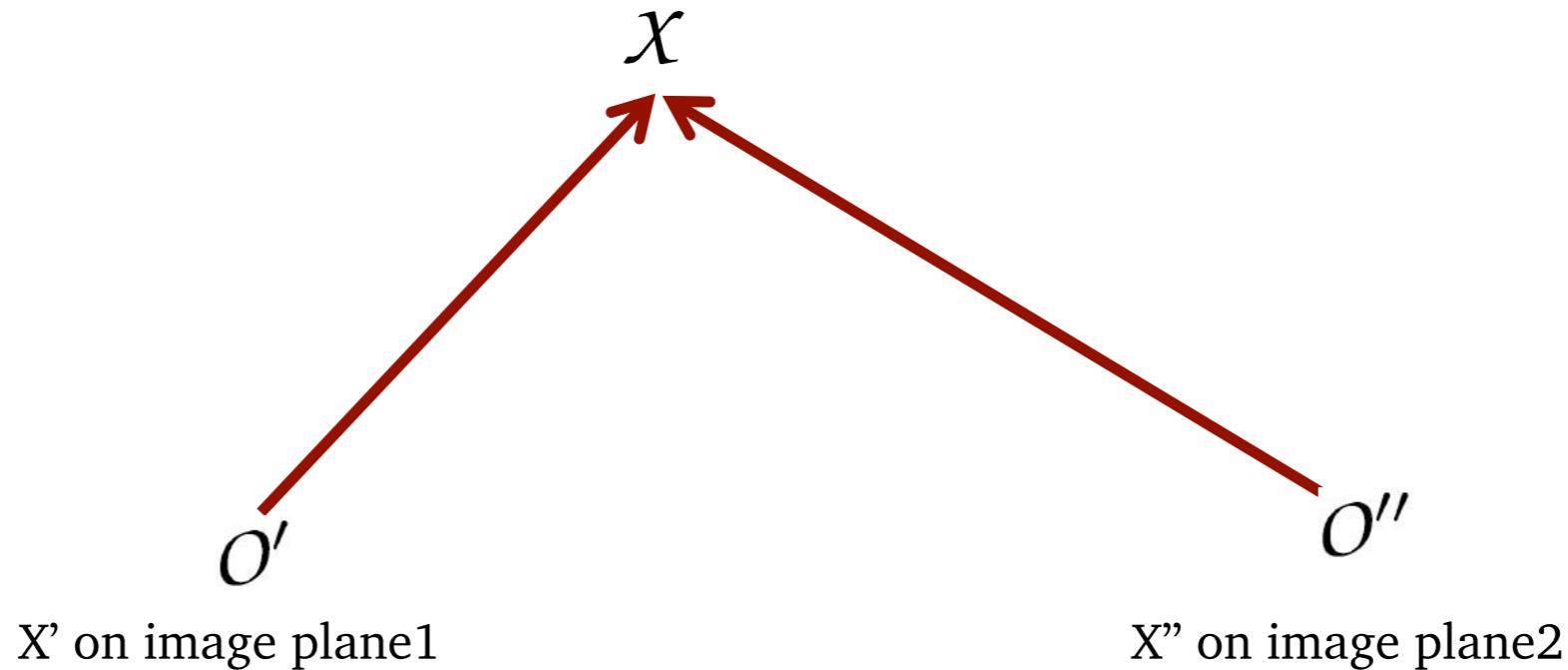
- It maps a point in one image to a point in the other image
- The mapping does not depend on the depth (distance) of point to the camera
- We do not assume any base line vector for the two images

Epipolar geometry

Given a point X' in the plane of the first image

Find the corresponding point X'' in the second image plane

We assume a straight-line preserving camera



- reduces the search space from 2d to 1D for finding correspondences
- reduced errors in finding data associations across 2 different images

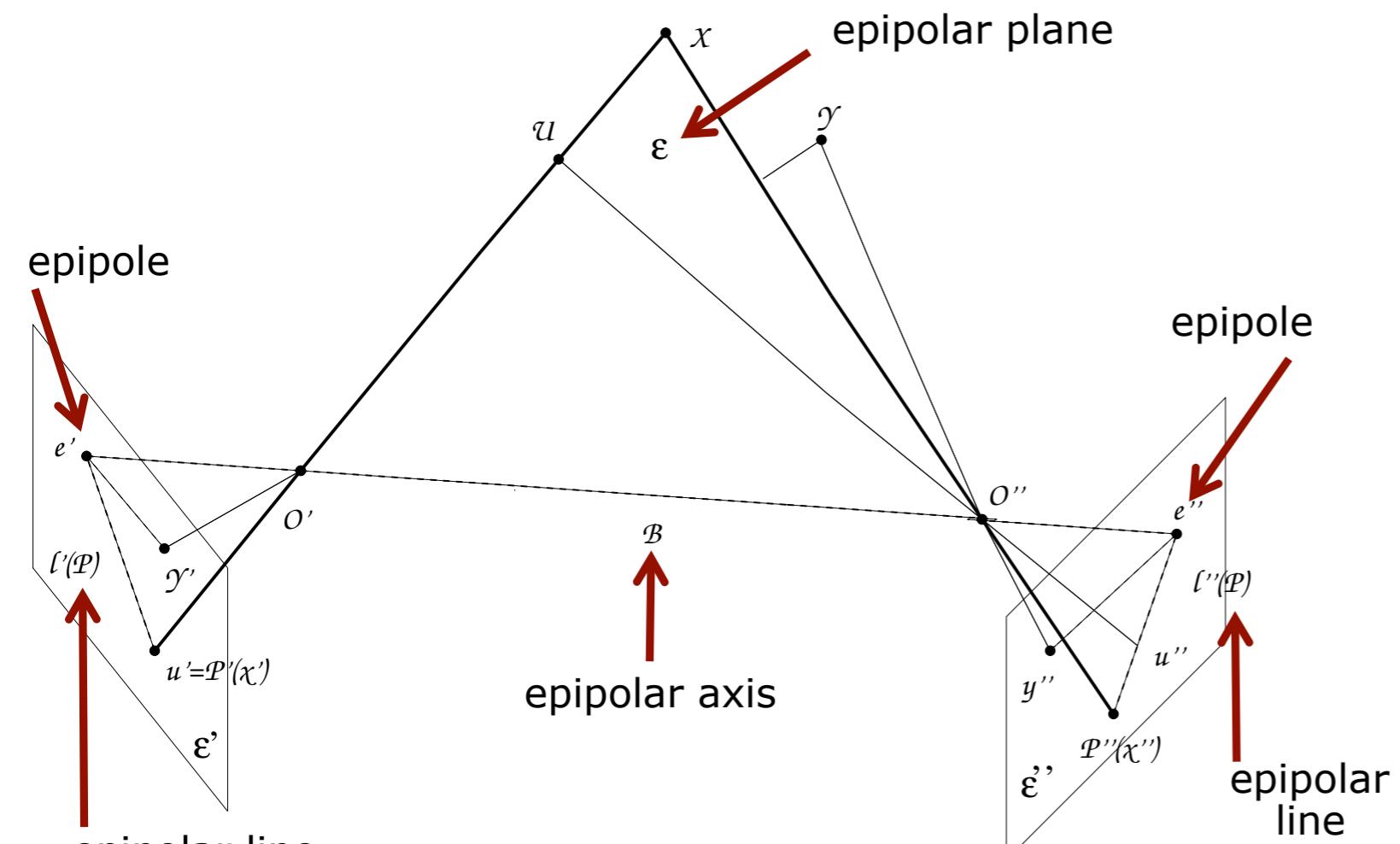


Image courtesy: Förstner

Epipolar axis (base line) ($\mathcal{B} = (O' O'')$) is the line through the two projection centers

Epipolar plane ($\varepsilon = (O' O'' X)$) depends on the projection centers and the 3D world point

Epipoles ($e' = (O'')'$, $e'' = (O')''$) are the projection points of the projection centers of the other camera onto the image plane (eg: O'' projection centre of plane 2 lies on image plane 1 at e').

Epipolar lines ($\ell'(X) = (O'' X)'$, $\ell''(X) = (O' X)''$) are the images of the rays $(O'' X)$ and $(O' X)$ in the other image respectively which are the intersection of the epipolar plane with the image planes

- X is the point in real world
- X' : point in image plane 1
- X'' : point in image plane 2
- O' , O'' are their projection centres
- Another point y is shown which is mapped at different points in the image planes

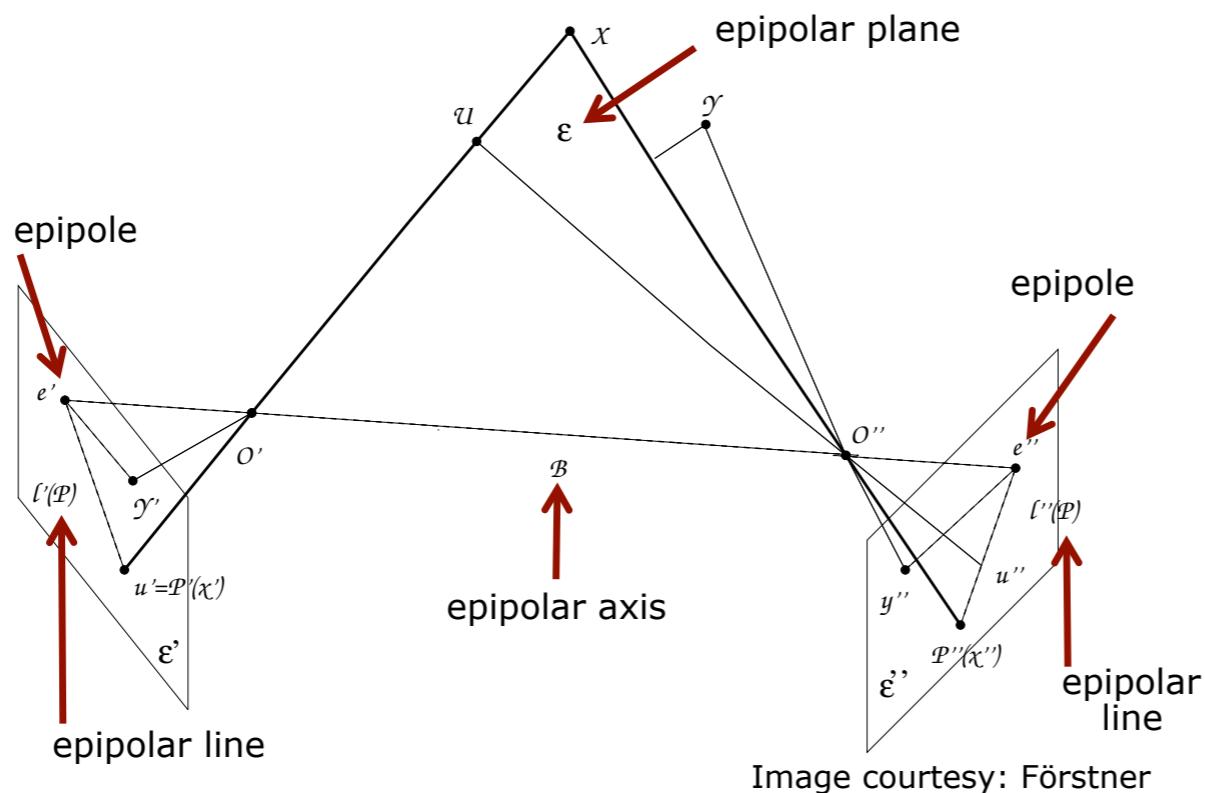


Image courtesy: Förstner

Epipoles ($e' = (O'')'$, $e'' = (O')''$) are the projection points of the projection centers of the other camera onto the image plane (eg: O'' projection centre of plane 2 lies on image plane 1 at e').

Epipoles can also be written as

$$e' = (O' O'') \cap \varepsilon' \quad e'' = (O' O'') \cap \varepsilon''$$

Epipolar lines ($\ell'(x) = (O''x)'$, $\ell''(x) = (O'x)''$) are the images of the rays $(O''x)$ and $(O'x)$ in the other image respectively which are the intersection of the epipolar plane with the image planes.

$$\ell'(x) = \varepsilon \cap \varepsilon' \quad \ell''(x) = \varepsilon \cap \varepsilon''$$

Epipolar lines are important when looking for **correspondences across images**.

Epipolar geometry simplifies the task of predicting the location of a corresponding point in the other image

Using a distortion-free lens,

the projection centers O', O''

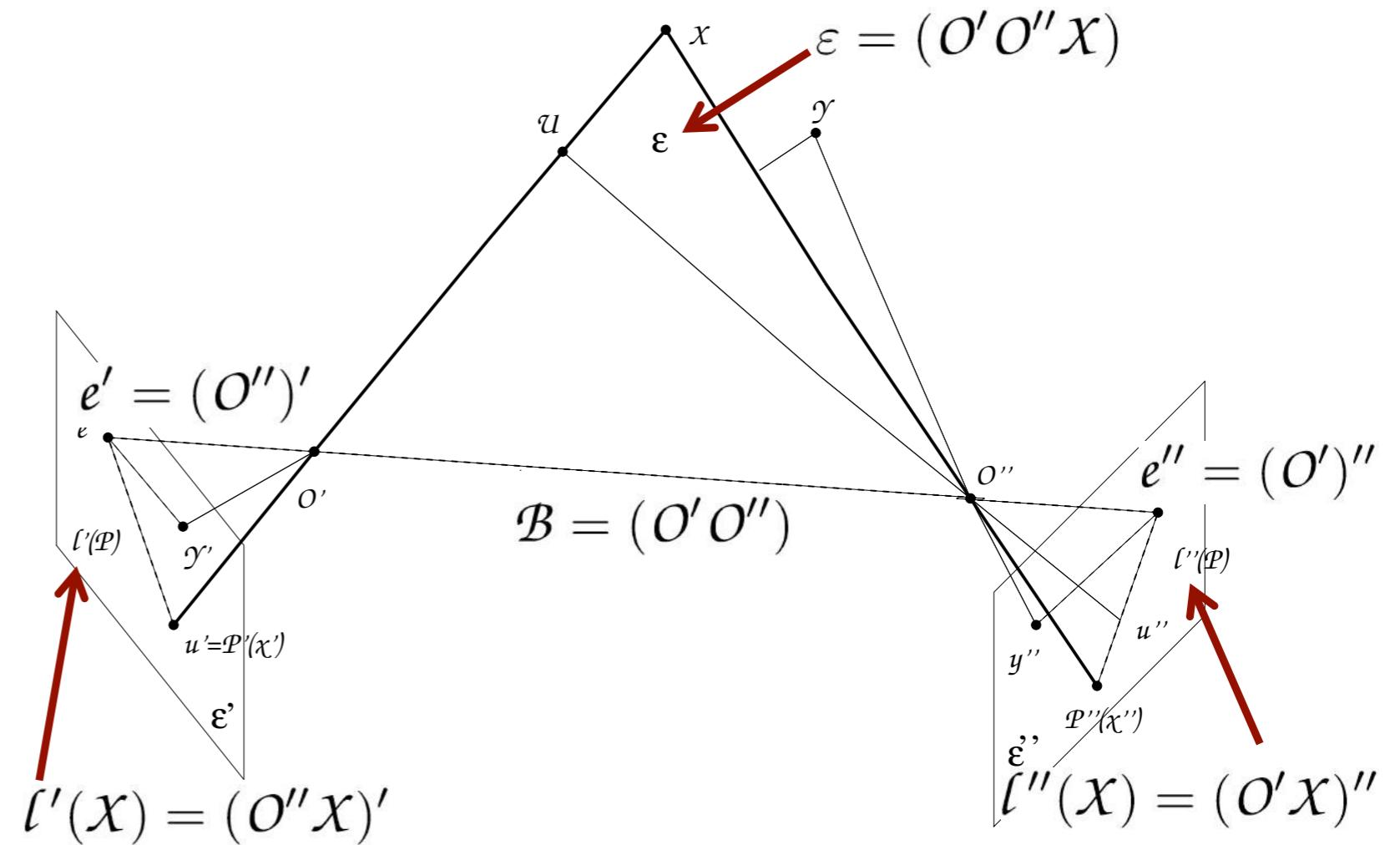
the point X

the epipolar lines $\ell'(X), \ell''(X)$

the epipoles e', e''

the image points X', X''

all lie in the epipolar plane \mathcal{E}



Predicting the location of corresponding points

- Task: Predict the location \mathcal{X}'' given \mathcal{X}'
- The epipolar plane through $\varepsilon = (O' O'' \mathcal{X}')^\top$
- The intersection of the epipolar plane and the second image plane ε'' yields the epipolar line $\ell''(\mathcal{X})$
- The corresponding point \mathcal{X}'' lies on the epipolar line $\ell''(\mathcal{X})$

Reduces the search space from 2D to 1D

Computing the elements of Epipolar geometry using projection matrices & fundamental matrix

Computing the elements of Epipolar geometry using projection matrices

Direction of the epipolar axis can directly be computed from the projection centers

We can only compute the direction,

$$\mathbf{b} = \mathbf{X}_{O''} - \mathbf{X}_{O'}$$

The epipoles are the projections of the projection centers in the other image computed using the projection matrices (they are the intersection of epipolar lines in an image)

$$\mathbf{e}' = \mathbf{P}' \mathbf{X}_{O''} \quad \mathbf{e}'' = \mathbf{P}'' \mathbf{X}_{O'}$$

Epipolar geometry and the fundamental matrix

“The epipolar geometry is the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the cameras’ internal parameters and relative pose.

Fundamental matrix F encapsulates this intrinsic geometry (it is the algebraic representation of epipolar geometry).”

Epipolar lines:

image points lie on the epipolar lines, $\mathbf{x}' \in \ell', \mathbf{x}'' \in \ell''$

For a point \mathbf{x}' (using the homogeneous coordinates property)

$$\mathbf{x}'^T \ell' = 0$$

Using the coplanarity constraint for the points \mathbf{x}' and \mathbf{x}'' , we can write the Fundamental matrix expression for a line ℓ' on image plane 1 and the point \mathbf{x}'' on image plane 2:

$$\mathbf{x}'^T \mathbf{F} \mathbf{x}'' = 0 \quad \longrightarrow$$

$$\boxed{\ell' = \mathbf{F} \mathbf{x}''}$$

Similarly, we for the point \mathbf{x}'' : $\ell''^T \mathbf{x}'' = 0 \quad \boxed{\ell'' = \mathbf{F}^T \mathbf{x}'}$

So, $\mathbf{x}'^T \mathbf{F} \mathbf{x}'' = 0$ if $\ell' = \mathbf{F} \mathbf{x}''$

$$\ell'' = \mathbf{F}^T \mathbf{x}'$$

This tells us that we can search in the 1D space for finding correspondences

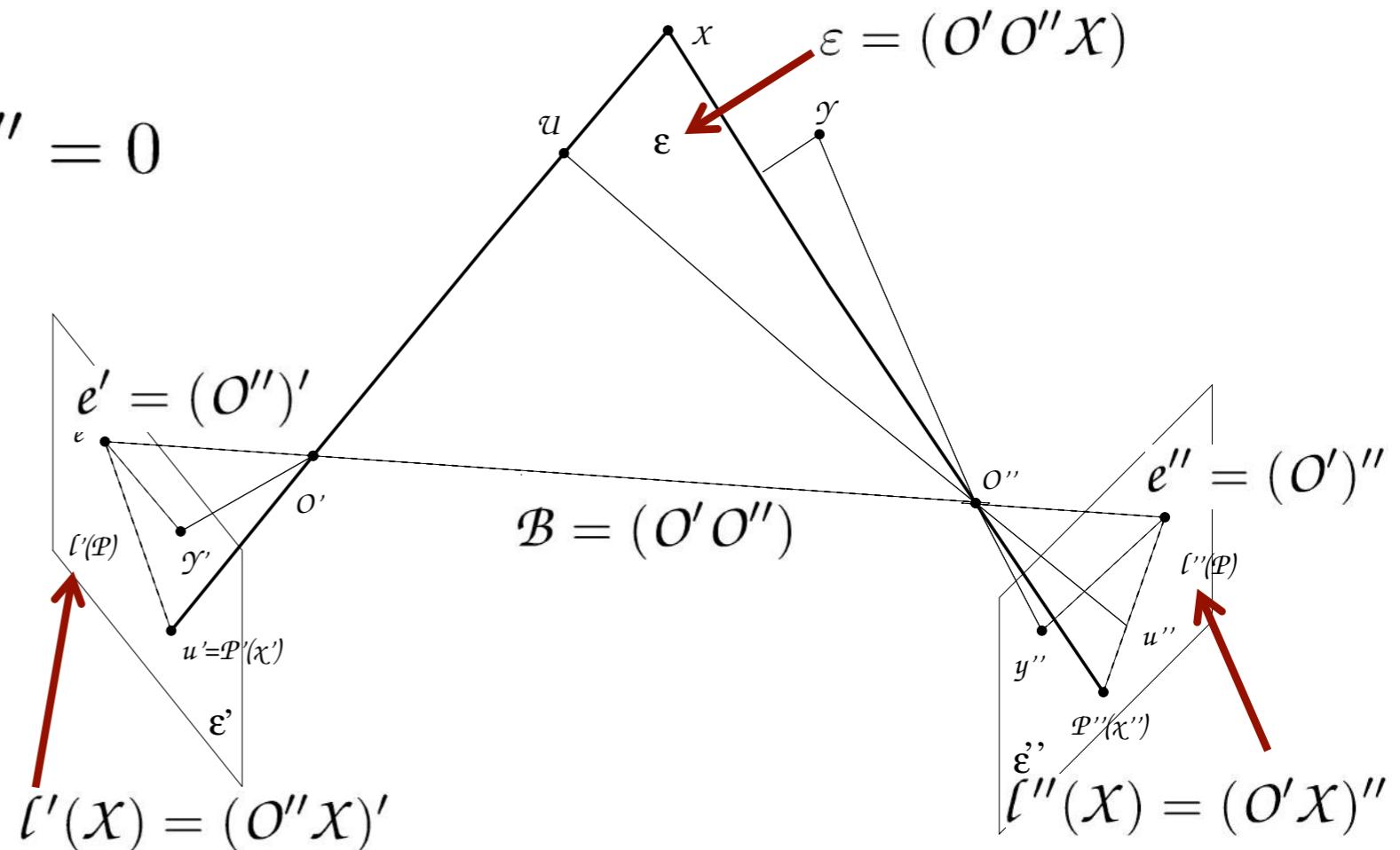
Epipoles and the fundamental matrix

All epipolar lines (l' , l'') pass through the epipoles (e' , e'')

$$\forall l' : e'^\top l' = 0 \quad \forall l'' : e''^\top l'' = 0$$

This gives:

$$l' = Fx'' \\ e'^\top Fx'' = 0$$



For the second epipole,

$$x'^\top Fe'' = 0$$

The epipoles are the left and right eigenvectors of the fundamental matrix and they correspond to the eigenvalues equal to zero

Relative Orientation, Fundamental matrix and Essential matrix

DSE312-CV

Lecture34

Bhavna R

Camera Pairs



Two types:

- A stereo camera
- One camera that can be moved

Camera pair = two configurations from which images have been taken

Relative Orientation between camera pairs or camera with 2 views

The orientation of the camera pair can be described using with independent orientations for each camera

How many parameters are needed?

- Calibrated cameras: ? parameters (angle preserving mapping)
- Uncalibrated cameras: ? parameters (straight-line preserving mapping)

Calibrated cameras: 3 rotation + 3 translation (for camera 1)

+

3 rotation + 3 translation (for camera 2)

=12 parameters

UnCalibrated cameras: 3 rotation + 3 translation + 5 intrinsic (for camera 1)

+

3 rotation + 3 translation + 5 intrinsic(for camera 2)

=22 parameters

We do not assume that the camera pairs have same intrinsics (different calibration)

Relative Orientation between camera pairs or camera with 2 views using control points

The orientation of the camera pair can be described using independent orientations for each camera

Calibrated camera pair: 12 parameters

Can be computed via two separate
spatial resection/P3P steps

Requires 3(4) known control points

Uncalibrated pair: 22 parameters

Can be computed via two separate DLT steps

Requires 6 known control points

3D world point \longrightarrow 2D point

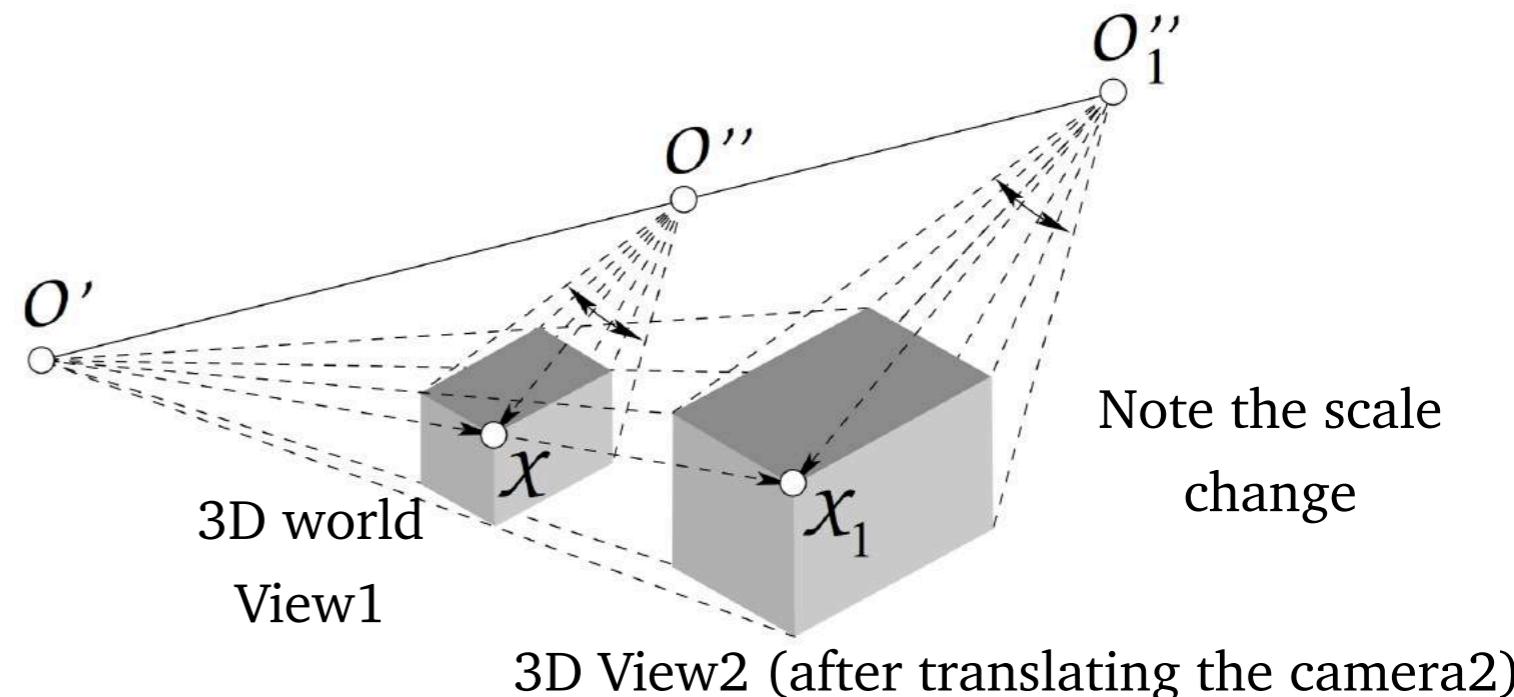
2D point image plane1 \longrightarrow 2D point image plane2

So far, we estimated the camera parameters by mapping the the scene points?
But take a step further by not knowing anything about the scene?

Can we estimate camera motion without knowing the scene?

Calibrated camera: Which parameters can we obtain and which not?

- Calibrated camera pair: 12 parameters
- Can be computed via two separate spatial resection/P3P steps
- Requires 3(4) known control points

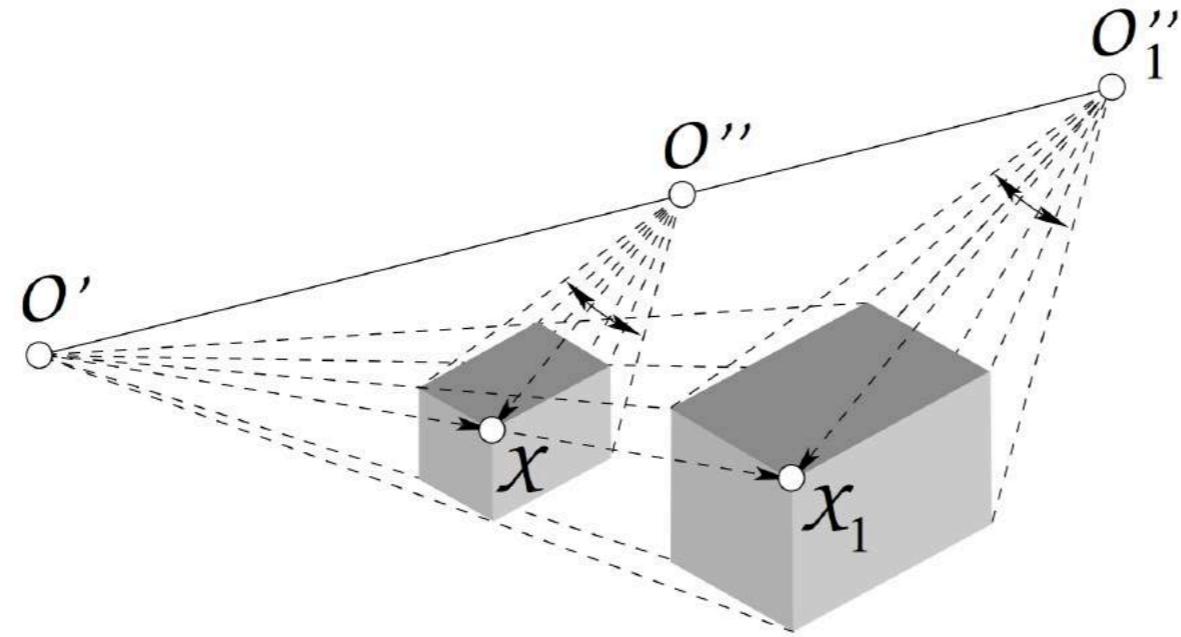


We cannot obtain the (global) translation and rotation (if the cameras maintain their relative transformation) as well as the scale (7 parameters that cannot be estimated).

We can obtain:

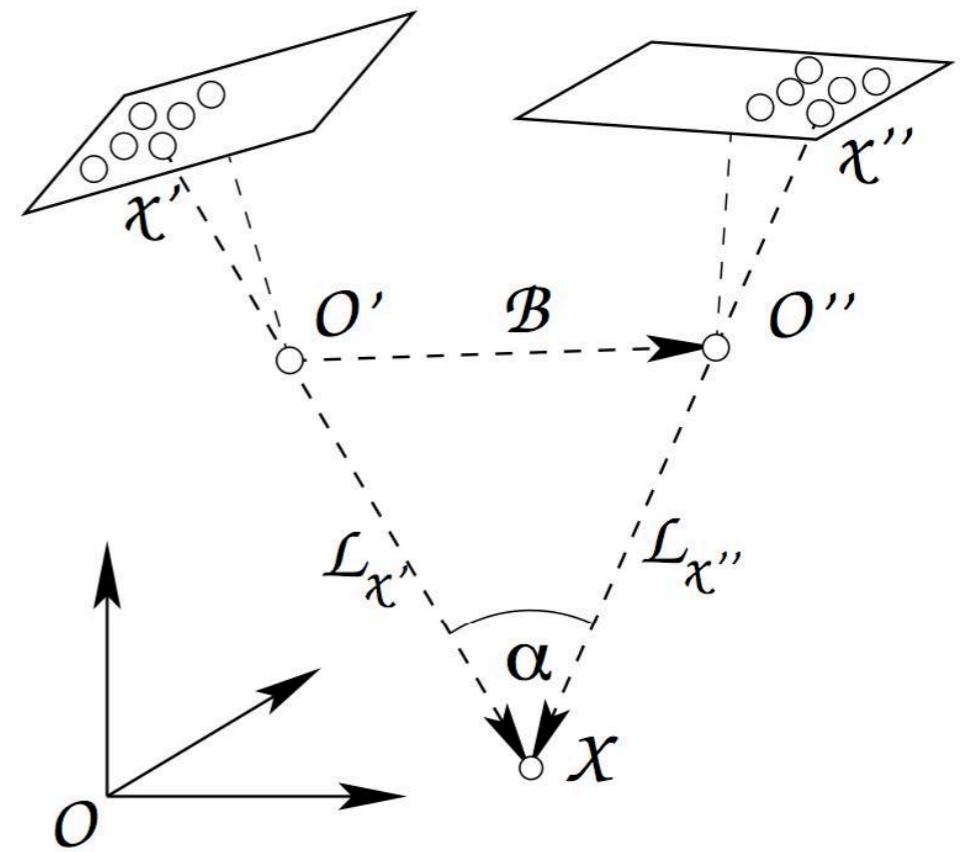
- The rotation of the second camera w.r.t. the first one (3 parameters)
- The direction of the line B (base line vector) connecting the two centers of projection (2 params)
(look up yesterday's lecture for what B is)
- We do not know the distance between the projection centres (the length of B that is related to the scale)

Calibrated camera: Which parameters can we obtain and which not?



- We need $2 \times 6 = 12$ parameters for two calibrated cameras for the orientation
- With a calibrated camera, we obtain an angle-preserving model of the object
- Without additional information, we can only obtain $12 - 7 = 5$ parameters that gives the relative orientation (not 7 = global translation and rotation, scale of the object)
- Given two cameras images, we can reconstruct the object only up to a similarity transform
- If we want to estimate the global parameters, we need at least 3 points in 3D (to estimate the 7 parameters), which will provide the absolute orientation.

How can we recover more of the orientation (absolute)?



Point pairs are marked approximately

We need at least 3 point pairs for computing the 3D Model of a Scene, which will provide the absolute orientation

OR

We need both camera positions

If we know the length of B , which is the base line, we can compute the scale and recover the 3D model of the world

When real distances in 3D world are given, then also we can use the information

UnCalibrated camera: Which parameters can we obtain and which not?

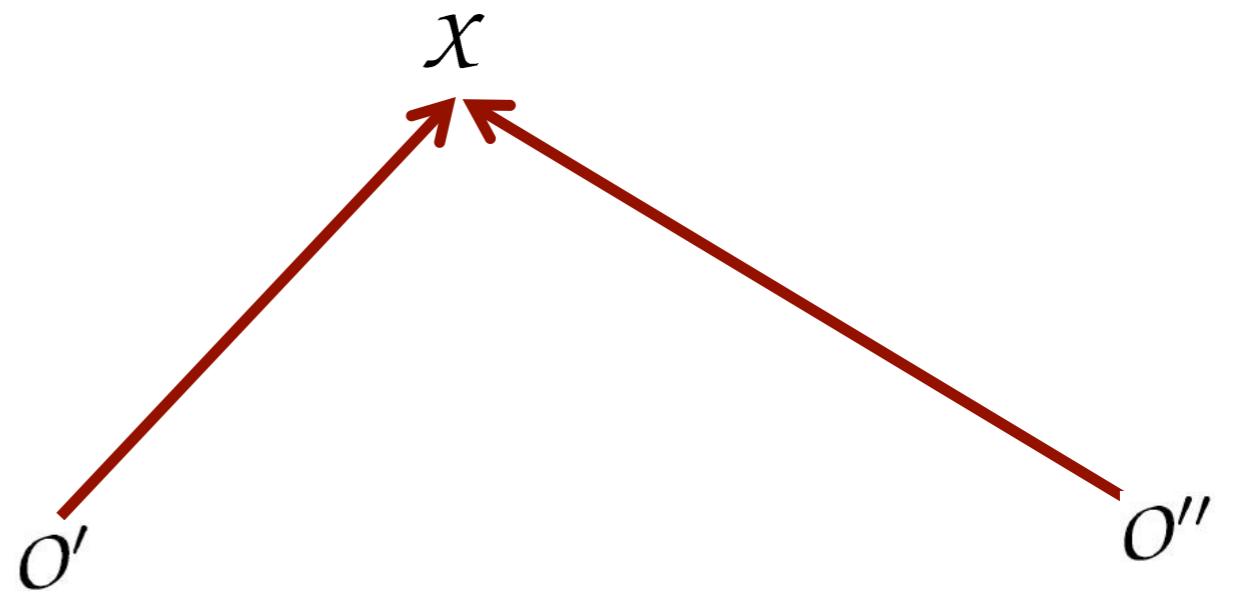
- Straight-line preserving but not angle preserving (i.e. real world straight lines are preserved but not angles)
- Object can be reconstructed up to a straight-line preserving mapping
- Projective transform cannot be resolved (15 parameters)
- Thus, for uncalibrated cameras, we can only obtain $22-15=7$ parameters given two images
- We need at least 5 points in 3D (15 coordinates) for the absolute orientation for uncalibrated cameras.

Relative orientation is an important information as it can give parameters without knowing anything about the scene

Cameras	#params /img	#params /img pair	#params for RO Relative orientation	#params for AO absolute orientation	min #P Minimum Number of
calibrated	6 (Extrinsics)	12	5	7	3
not calibrated	11 (6 Extrinsics & 5 intrinsics)	22	7	15	5

Coplanarity Constraint for Straight-Line Preserving

(Uncalibrated) Cameras to Obtain the Fundamental Matrix

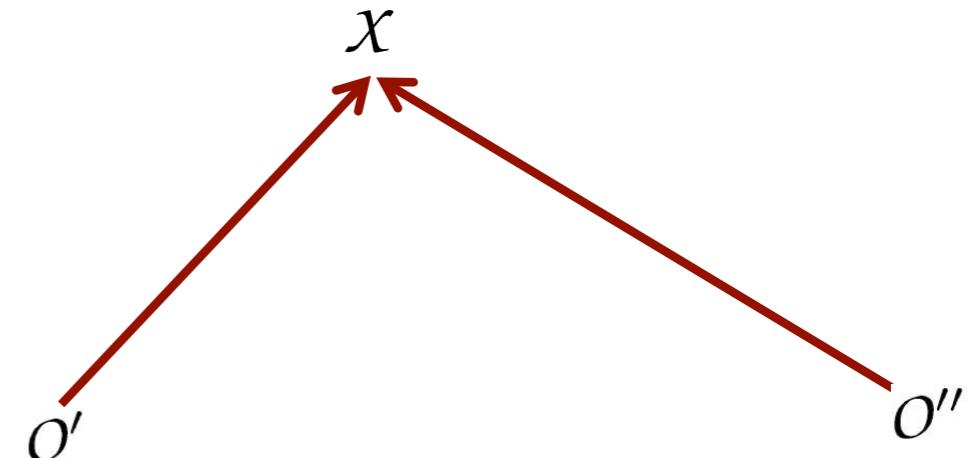


Coplanarity Constraint for Straight-Line Preserving for the (Uncalibrated) Cameras to obtain the Fundamental Matrix

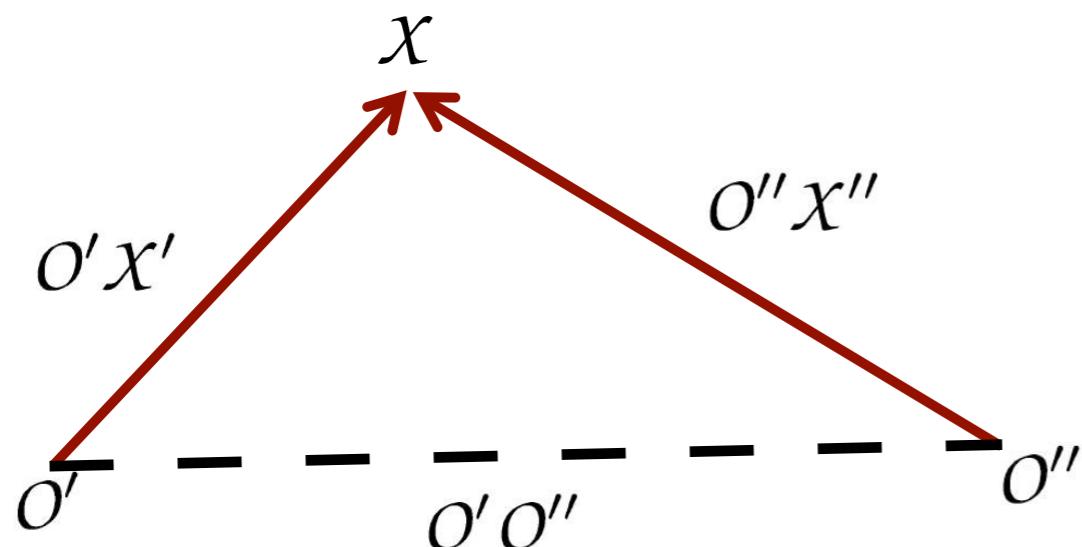
Consider perfect orientation and the intersection of two corresponding rays

The rays lie within one plane in 3D

Both cameras observe the same point (X) in the 3D world



Coplanarity can be expressed using the vectors as the scalar triple product given as



$$[O'X' \quad O'O'' \quad O''X''] = 0$$

\$X'\$ and \$X''\$ refer to the same point \$X\$ (3D world) onto the image planes (1, 2) respectively whose projection centres are \$O'\$ and \$O''\$

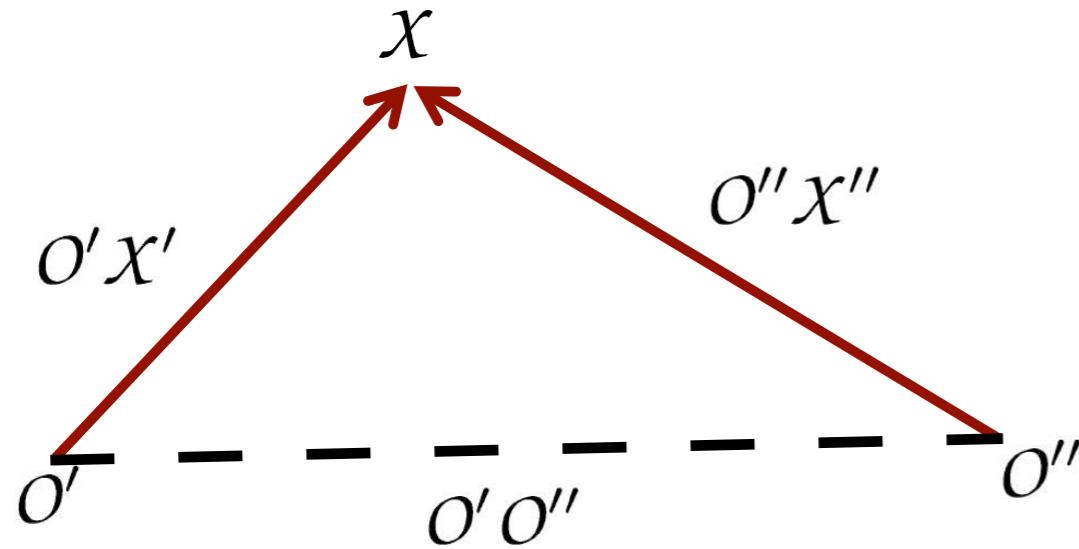
Since the three vectors lie on one plane (coplanarity constraint), their scale triple product is zero

(eg: for 3 vectors: [a,b,c], their scale triple product can be written as : \$(a \times b) \cdot c\$

If they lie on a single plane, then, \$(a \times b) \cdot c = 0\$

(cross product gives a vector orthogonal to a, b. if c lies on that orthogonal plane, then the dot product of c with resulting vector is 0.)

Coplanarity Constraint for the (Uncalibrated) Cameras to obtain the Fundamental Matrix



$$[O'X' \quad O'O'' \quad O''X''] = 0$$

- We can write the expression without any knowledge about the scene, just using the coplanarity constraint.
- We need to replace these with the parameters that know about the camera

The directions of the vectors $O'X'$ and $O''X''$ can be derived from the image coordinates (X', X'')

$$\mathbf{x}' = \mathbf{P}'\mathbf{X} \quad \text{Camera 1}$$

Both cameras observe the same point (X) in the 3D world

$$\mathbf{x}'' = \mathbf{P}''\mathbf{X} \quad \text{Camera 2}$$

The projection matrices

(in terms of projection centres of the 2 cameras): $\mathbf{P}' = \mathbf{K}'\mathbf{R}'[\mathbf{I}_3] - \mathbf{X}_{O'}$ $\mathbf{P}'' = \mathbf{K}''\mathbf{R}''[\mathbf{I}_3] - \mathbf{X}_{O''}$

The directions of the vectors $O'X'$ and $O''X''$ can be derived from the image coordinates (X', X'')

$$\mathbf{x}' = \mathbf{P}'\mathbf{X} \quad \text{Camera 1}$$

$$\mathbf{x}'' = \mathbf{P}''\mathbf{X} \quad \text{Camera 2}$$

Both cameras observe the same point (X) of the 3D world
The points x' and x'' lie on the image planes (1 & 2)

We want to now compute the direction from the center of projection to the point (X) in 3D to obtain orientations of the 2 cameras

We term this as the normalized directions of the vectors $O'X'$ and $O''X''$ which are

$${}^n\mathbf{x}' = (\mathbf{R}')^{-1}(\mathbf{K}')^{-1}\mathbf{x}' \quad \text{Camera 1}$$

$${}^n\mathbf{x}'' = (\mathbf{R}'')^{-1}(\mathbf{K}'')^{-1}\mathbf{x}'' \quad \text{Camera 2}$$

Suppose the motion of the cameras is a pure translation with no rotation and no change in the internal parameters, then we have:

$${}^n\mathbf{x}' = [\mathbf{I}_3 | -\mathbf{X}_{O'}] \mathbf{X}$$

This gives the direction from the projection center to the point in 3D

(no rotation, $X_{O'}$ is the point that lies at the projection centre and \mathbf{X} is the 3D world point)

The base vector (\mathbf{b}) results from the projection centers (O' and O'')

$$\mathbf{b} = \mathbf{X}_{O''} - \mathbf{X}_{O'}$$

Coplanarity Constraint for the (Uncalibrated) Cameras to obtain the Fundamental Matrix

Using the coplanarity constraint

$$[O'X' \quad O'O'' \quad O''X''] = 0$$

The base vector (\mathbf{b}) results from the projection centers (O' and O'')

$$\mathbf{b} = X_{O''} - X_{O'}$$

The normalized directions of the vectors $O'X'$ and $O''X''$

$${}^n\mathbf{x}' = (R')^{-1}(K')^{-1}\mathbf{x}'$$

$${}^n\mathbf{x}'' = (R'')^{-1}(K'')^{-1}\mathbf{x}''$$

$$[O'X' \quad O'O'' \quad O''X''] = 0$$

$$[{}^n\mathbf{x}' \quad \mathbf{b} \quad {}^n\mathbf{x}''] = 0$$

$$[{}^n\mathbf{x}' \quad \mathbf{b} \quad {}^n\mathbf{x}''] = 0$$

$${}^n\mathbf{x}' \cdot (\mathbf{b} \times {}^n\mathbf{x}'') = 0$$

scalar triple product

The scalar triple product can be expressed in terms of a skew-symmetric matrix

$${}^n \mathbf{x}' \cdot (\mathbf{b} \times {}^n \mathbf{x}'') = 0 \quad \text{Scalar triple vector product}$$

$$\begin{aligned} & \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array} \right] \times \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right] = \left[\begin{array}{c} -b_3 x_2 + b_2 x_3 \\ b_3 x_1 - b_1 x_3 \\ -b_2 x_1 + b_1 x_2 \end{array} \right] \\ &= \underbrace{\left[\begin{array}{ccc} 0 & -b_3 & b_2 \\ b_3 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{array} \right]}_{S_b} \underbrace{\left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right]}_{\mathbf{x}} \end{aligned}$$

Skew-symmetric matrix

$${}^n \mathbf{x}' \cdot (\mathbf{b} \times {}^n \mathbf{x}'') = 0 \longrightarrow {}^n \mathbf{x}'^\top S_b {}^n \mathbf{x}'' = 0$$

Instead of using cross product, we can write the expression as matrix multiplication (it is mathematically convenient!)

Coplanarity Constraint for the (Uncalibrated) Cameras to obtain the Fundamental Matrix

$$\begin{aligned} {}^n\mathbf{x}' &= (R')^{-1}(K')^{-1}\mathbf{x}' \\ {}^n\mathbf{x}'^\top S_b {}^n\mathbf{x}'' &= 0 \end{aligned} \quad \boxed{\text{By combining}} \quad \boxed{{}^n\mathbf{x}'^\top (K')^{-\top}(R')^{-\top}S_b(R'')^{-1}(K'')^{-1}\mathbf{x}'' = 0}$$

Fundamental Matrix, F
(for uncalibrated cameras):

$$\begin{aligned} F &= (K')^{-\top}(R')^{-\top}S_b(R'')^{-1}(K'')^{-1} \\ &= (K')^{-\top}R'S_bR''^\top(K'')^{-1} \end{aligned}$$

$$F = (K')^{-\top}R'S_bR''^\top(K'')^{-1}$$

We have an expression for F just using just the 2D image points from the image planes and 5 matrix multiplication that gives a 3×3 matrix F

$$\mathbf{x}'^\top F \mathbf{x}'' = 0$$

If we have pixel coordinates 2 points on the image planes, using the coplanarity constraint

“Transpose: If F is the fundamental matrix of the pair of cameras (P', P'') , then F^T is the fundamental matrix of the pair in the opposite order: (P'', P')

The fundamental matrix contains the all the available information about the relative orientation of two images from uncalibrated cameras

- F is a rank 2 homogeneous matrix with 7 degrees of freedom.
- **Point correspondence:** If x and x' are corresponding image points, then

$$x'^T F x = 0.$$
- **Epipolar lines:**
 - ◊ $l' = Fx$ is the epipolar line corresponding to x .
 - ◊ $l = F^T x'$ is the epipolar line corresponding to x' .
- **Epipoles:**
 - ◊ $F\mathbf{e} = \mathbf{0}$.
 - ◊ $F^T \mathbf{e}' = \mathbf{0}$.
- **Computation from camera matrices P, P' :**
 - ◊ General cameras,

$$F = [e']_\times P' P^+$$
, where P^+ is the pseudo-inverse of P , and $e' = P' C$, with $P C = \mathbf{0}$.
 - ◊ Canonical cameras, $P = [I \mid \mathbf{0}]$, $P' = [M \mid m]$,

$$F = [e']_\times M = M^{-T} [e]_\times$$
, where $e' = m$ and $e = M^{-1} m$.
 - ◊ Cameras not at infinity $P = K[I \mid \mathbf{0}]$, $P' = K'[R \mid t]$,

$$F = K'^{-T} [t]_\times R K^{-1} = [K't]_\times K' R K^{-1} = K'^{-T} R K^T [K R^T t]_\times.$$

Table 9.1. *Summary of fundamental matrix properties.*

courtesy:[Hartley & Zissermann]

Coplanarity Constraint for calibrated Cameras to obtain the Essential Matrix

Calibrated cameras simplify the orientation problem

Often, we assume that both cameras have the same calibration matrix

Assumption here: no distortions or other imaging errors

$$\text{We start with the Fundamental matrix } F, \underbrace{\mathbf{x}'^T (\mathbf{K}')^{-T} (R')^{-T} S_b (R'')^{-1} (\mathbf{K}'')^{-1} \mathbf{x}''}_{F} = 0$$

For calibrated cameras the coplanarity constraint can be simplified

Based on the calibration matrices, we obtain the directions as

$$k \mathbf{x}' = \mathbf{K}'^{-1} \mathbf{x}' \quad k \mathbf{x}'' = \mathbf{K}''^{-1} \mathbf{x}''$$

$$\underbrace{\mathbf{x}'^T (\mathbf{K}')^{-T}}_{k \mathbf{x}'^T} R' S_b R''^T \underbrace{(\mathbf{K}'')^{-1} \mathbf{x}''}_{k \mathbf{x}''} = 0$$

$$k \mathbf{x}'^T R' S_b R''^T k \mathbf{x}'' = 0$$

$$E = R' S_b R''^T$$

Coplanarity constraint

$$k \mathbf{x}'^T E k \mathbf{x}'' = 0$$

In summary, Essential Matrix

- The essential matrix has five degrees of freedom
- There are five parameters that determine the relative orientation of the image pair for calibrated cameras
- There are $4 = 9 - 5$ constraints to its 9 elements (3 by 3 matrix)
- The essential matrix is homogenous and singular

$$k_{\mathbf{x}'}^T \mathbf{E} k_{\mathbf{x}''} = 0$$

Computing F & E from image point correspondences

Computing F, given only corresponding points in images

Given n corresponding points in 2 images $(x', y')_n, (x'', y'')_n$ with $n = 1, \dots, N$

For each point, using the coplanarity constraint we have $\mathbf{x}'_n^T \mathbf{F} \mathbf{x}''_n = 0 \quad n = 1, \dots, N$

$$[x'_n, y'_n, 1] \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x''_n \\ y''_n \\ 1 \end{bmatrix} = 0$$

We simply multiply the vectors & matrix $x''_n F_{11} x'_n + x''_n F_{21} y'_n + \dots = 0$
 (linear dependency)

F appear as linear terms,
 The points are known

$$\begin{aligned} & [x''_n x'_n, x''_n y'_n, x''_n, y''_n x'_n, y''_n y'_n, y''_n, x'_n, y'_n, 1] \cdot \\ & [F_{11}, F_{21}, F_{31}, F_{12}, F_{22}, F_{32}, F_{13}, F_{23}, F_{33}]^T = 0 \\ & n = 1, \dots, N \end{aligned}$$

F appear as linear terms, $\mathbf{a}_n^T \rightarrow [x_n''x'_n, x_n''y'_n, x_n'', y_n''x'_n, y_n''y'_n, y_n'', x'_n, y'_n, 1] \cdot$

The points are known $\mathbf{f} \rightarrow [F_{11}, F_{21}, F_{31}, F_{12}, F_{22}, F_{32}, F_{13}, F_{23}, F_{33}]^T = 0$
 $n = 1, \dots, N$

We can write as coefficients times F $\mathbf{a}_n^T \mathbf{f} = 0 \quad n = 1, \dots, N$

$$(\mathbf{x}_n'' \otimes \mathbf{x}_n')^T \text{vec } F = \underbrace{\mathbf{a}_n^T}_{\mathbf{a}_n^T} \underbrace{\mathbf{f}}_{\mathbf{f}} = 0 \quad n = 1, \dots, N$$

Using Kronecker product: $(\mathbf{x}_n'' \otimes \mathbf{x}_n')^T \text{vec } F \quad \mathbf{x}^T F \mathbf{y} = (\mathbf{y} \otimes \mathbf{x})^T \text{vec } F$

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \dots \\ \mathbf{a}_n^T \\ \dots \\ \mathbf{a}_N^T \end{bmatrix} \quad Af = \mathbf{0}$$

- We solve using Singular value decomposition
- f is the right-singular vector corresponding to a singular value of A that is zero. Each corresponding point pairs give one such equation.
- We need atleast 8 corresponding points
- In real world points are noisy, so we need more than 8 points
- In reality, smallest singular value is the solution
- We stack them into a matrix

Estimating F

```
% xs, XSS: Nx3 homologous point coordinates, N > 7
% F:          3x3 fundamental matrix

% coefficient matrix
for n = 1 : size(xs, 1)
    A(n, :) = kron(XSS(n, :), xs(n, :));
end

% singular value decomposition
[U, D, V] = svd(A);

% select the singular vector with the minimal singular value
F = reshape(V(:, 9), 3, 3);
```

We estimated F, but the F is not necessarily a matrix of rank 2 (but F should have: $\text{rank}(F)=2$)

So, change the steps a bit by taking F (smallest singular value) and set to 0 to enforce $\text{rank}(F)=2$:

```
[Ua, Da, Va] = svd(F);
%
% algebraically best F, singular
F1 = Ua * diag([Da(1, 1), Da(2, 2), 0]) * Va';
```

$$F = UDV^T = U \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

$\text{rank}(F)=2$

Computing E, given only corresponding points in images

Essential Matrix for calibrated cameras

$$E = R' S_b {R''}^\top$$

Parameterisation

$$E = S_b R^\top$$

Coplanarity constraint for calibrated cameras

$${^k \mathbf{x}'}^\top E {^k \mathbf{x}''} = 0$$

For each point, using the coplanarity constraint we have:

$${^k \mathbf{x}'}_n^\top E {^k \mathbf{x}''}_n = 0 \quad n = 1, \dots, N$$

$$[{^k x'}_n, {^k y'}_n, c'] \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} {^k x''}_n \\ {^k y''}_n \\ c'' \end{bmatrix} = 0$$

Solve: **Ae=0**

Estimating E

```
% xs, XSS: Nx3 homologous point coordinates,  
% E:          3x3 essential matrix
```

```
% coefficient matrix  
for n = 1 : size(xs, 1)  
    A(n, :) = kron(XSS(n, :), xs(n, :));  
end
```

```
% singular value decomposition
```

```
[U, D, V] = svd(A);
```

```
E = reshape(V(:, 9), 3, 3)';
```



$$E = U \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

homogenous

both non-zero singular values are identical

```
% svd decomposition of E
```

```
[Ua, Da, Va] = svd(E);
```

```
E1 = Ua * diag([1, 1, 0]) * Va';
```

Estimating E

E is Singular: $|E| = 0$ (determinant is zero)

Two identical non-zero singular values

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

↑
rotation
matrices

$$E = R' S_b R''^T$$

$${}^k \mathbf{x}'^T E {}^k \mathbf{x}'' = 0$$

Because of the skew-symmetric matrix, we get:

$$2EE^T E - \text{tr}(EE^T)E = \underset{3 \times 3}{\mathbf{0}}$$