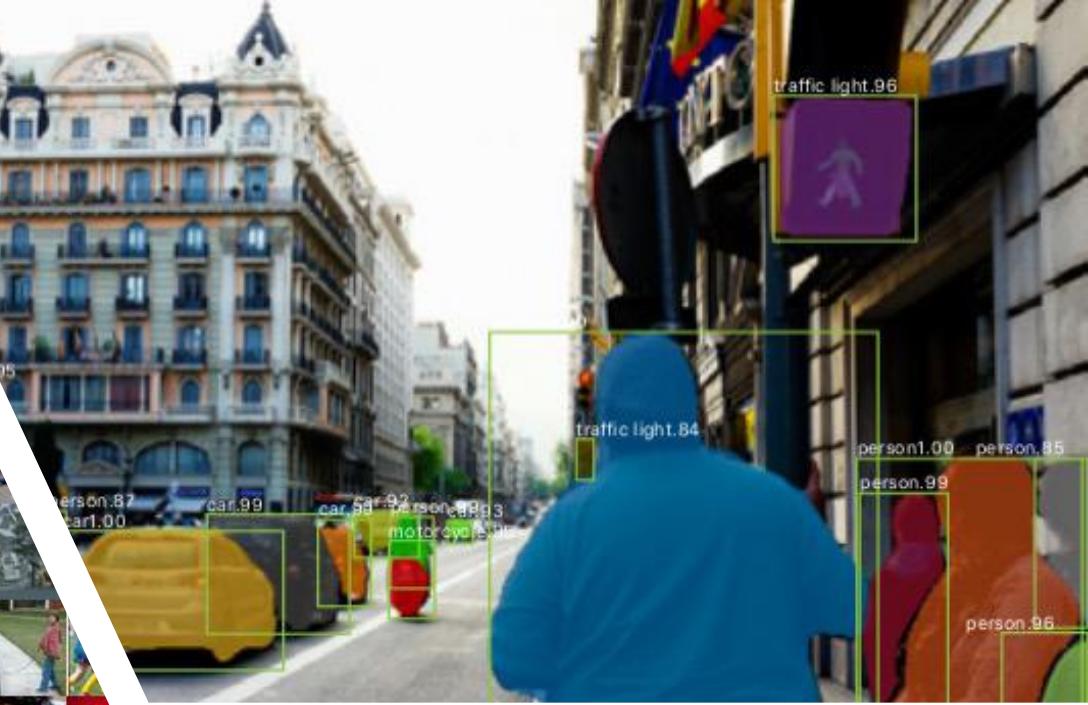
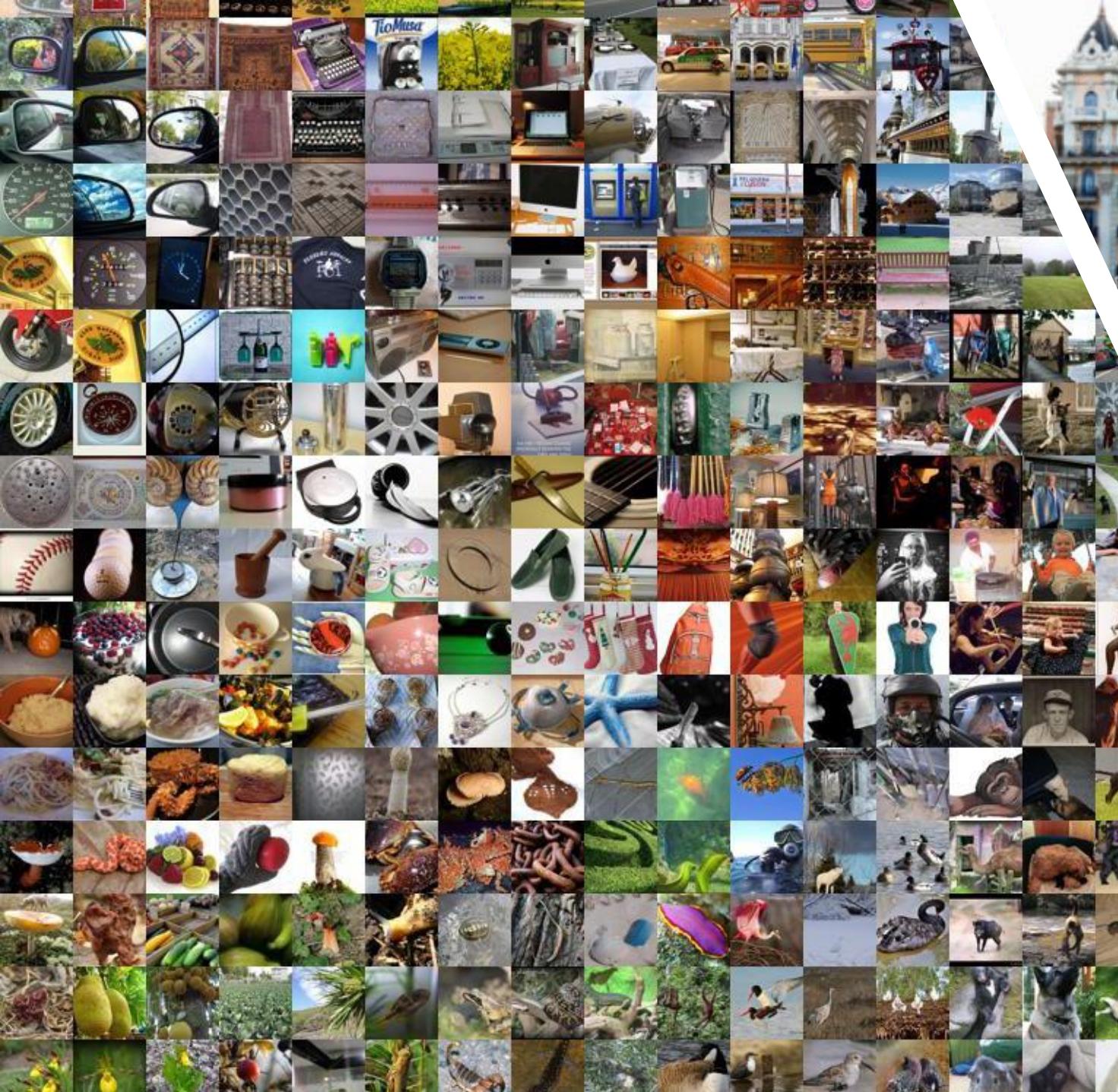


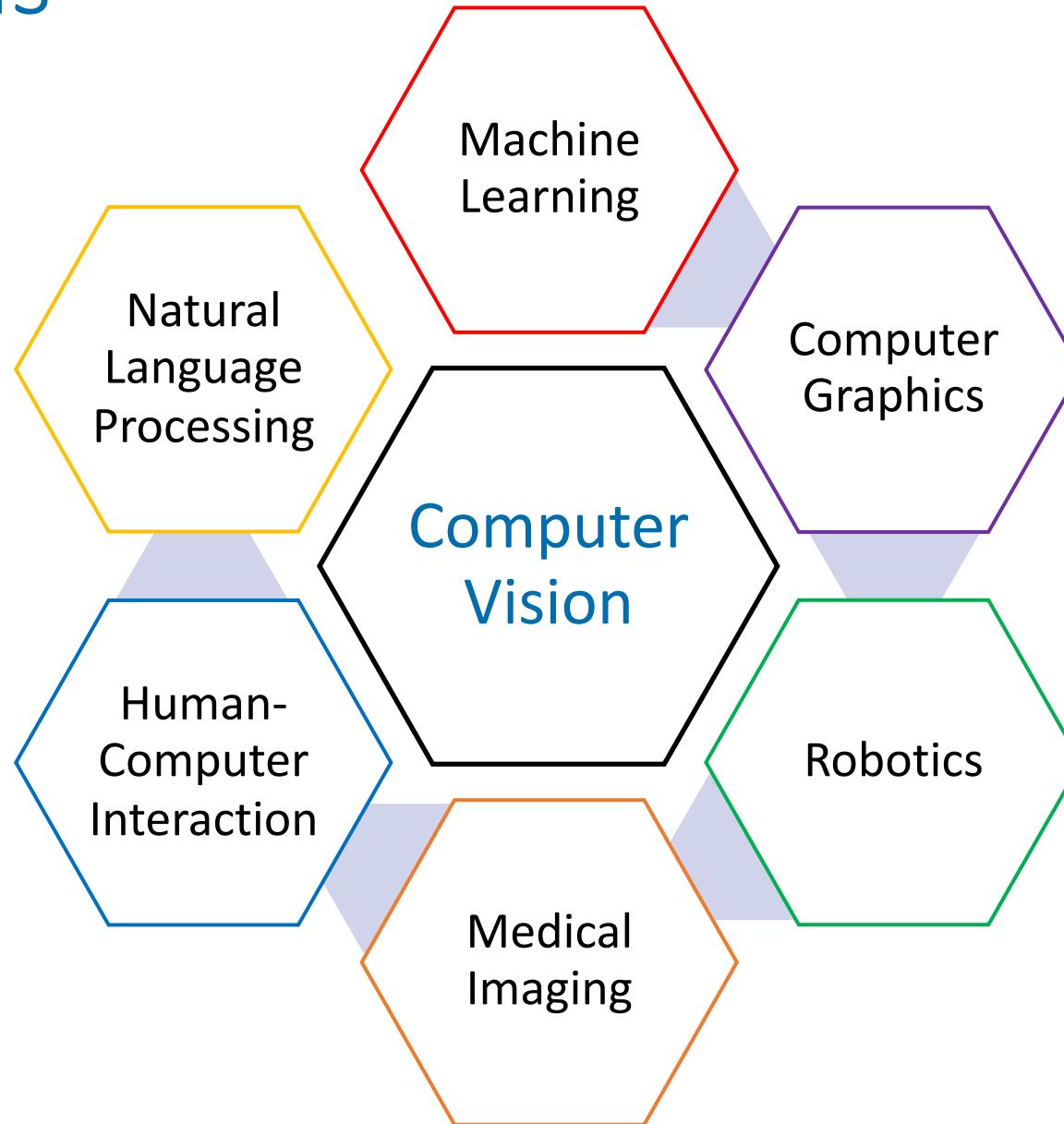
Computer Vision

Dr Wenjia Bai

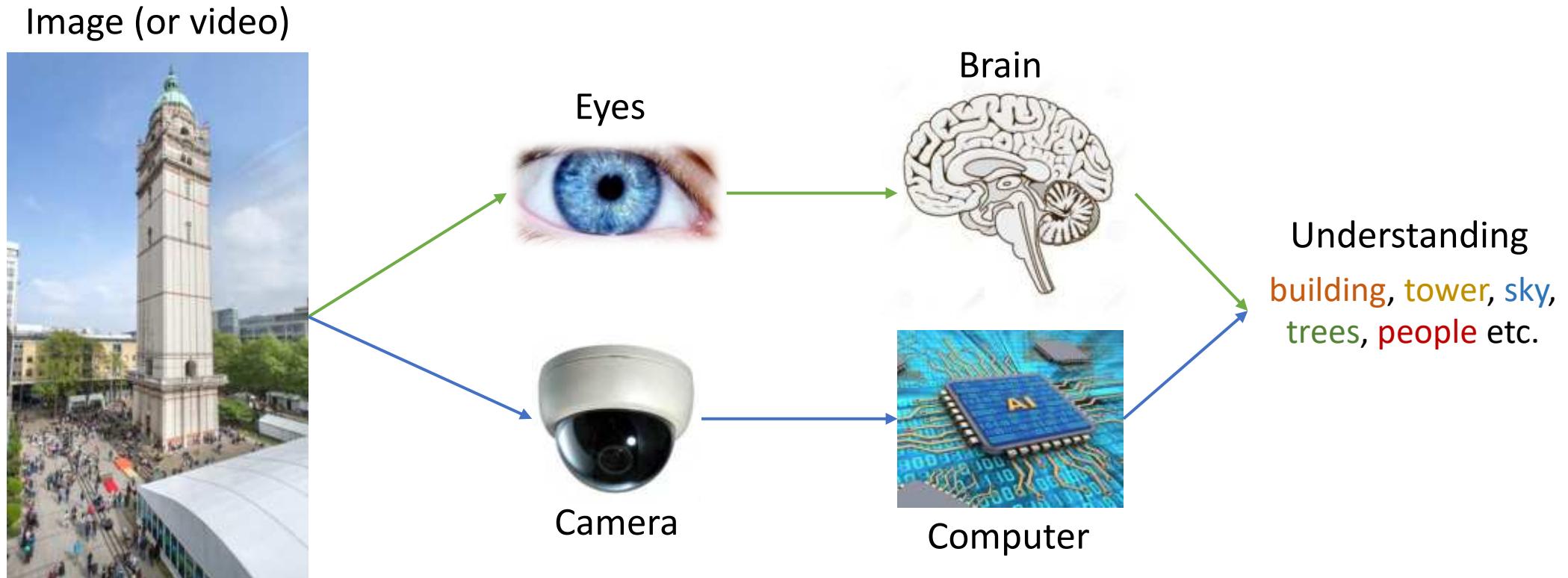
Department of Computing & Brain Sciences



Related fields

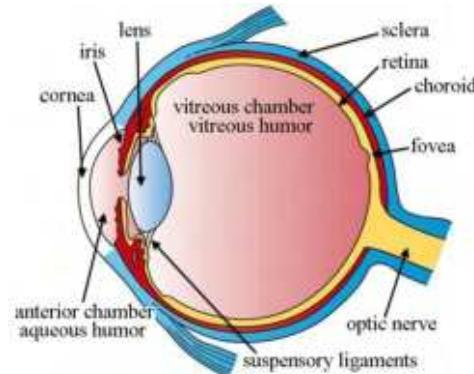


What is computer vision?

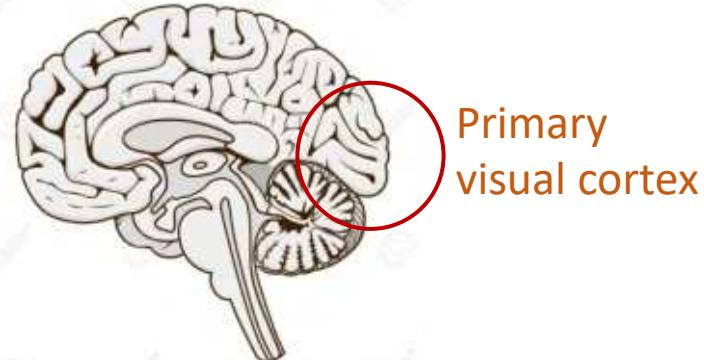


Human vision

- Sensor: eyes



- Processor: brain



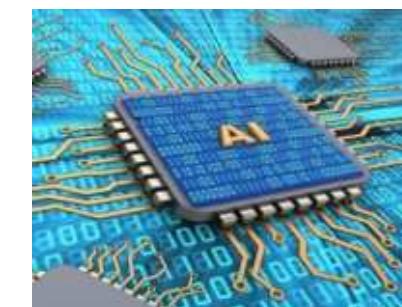
Computer vision

- Sensor

- Camera
- Microscope
- Medical imaging scanner
- ...

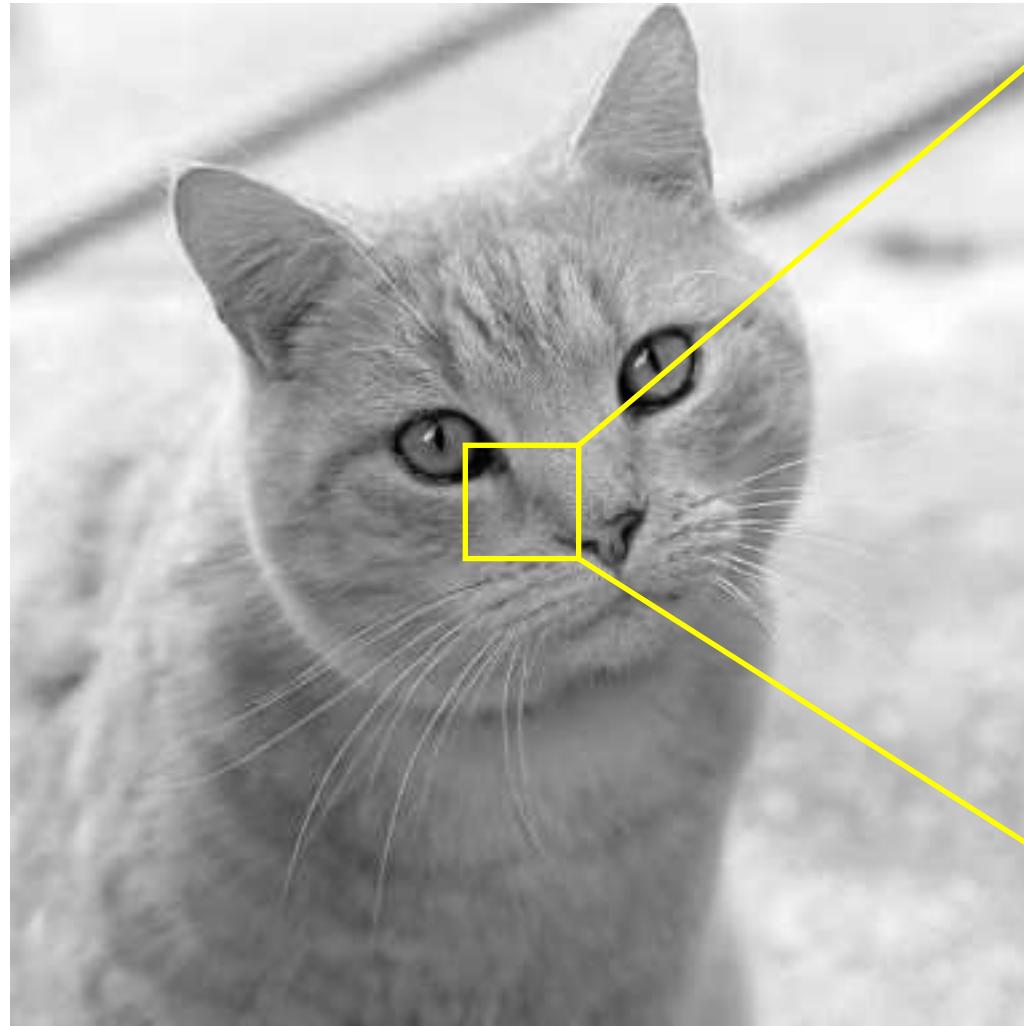


- Processor: computer



?

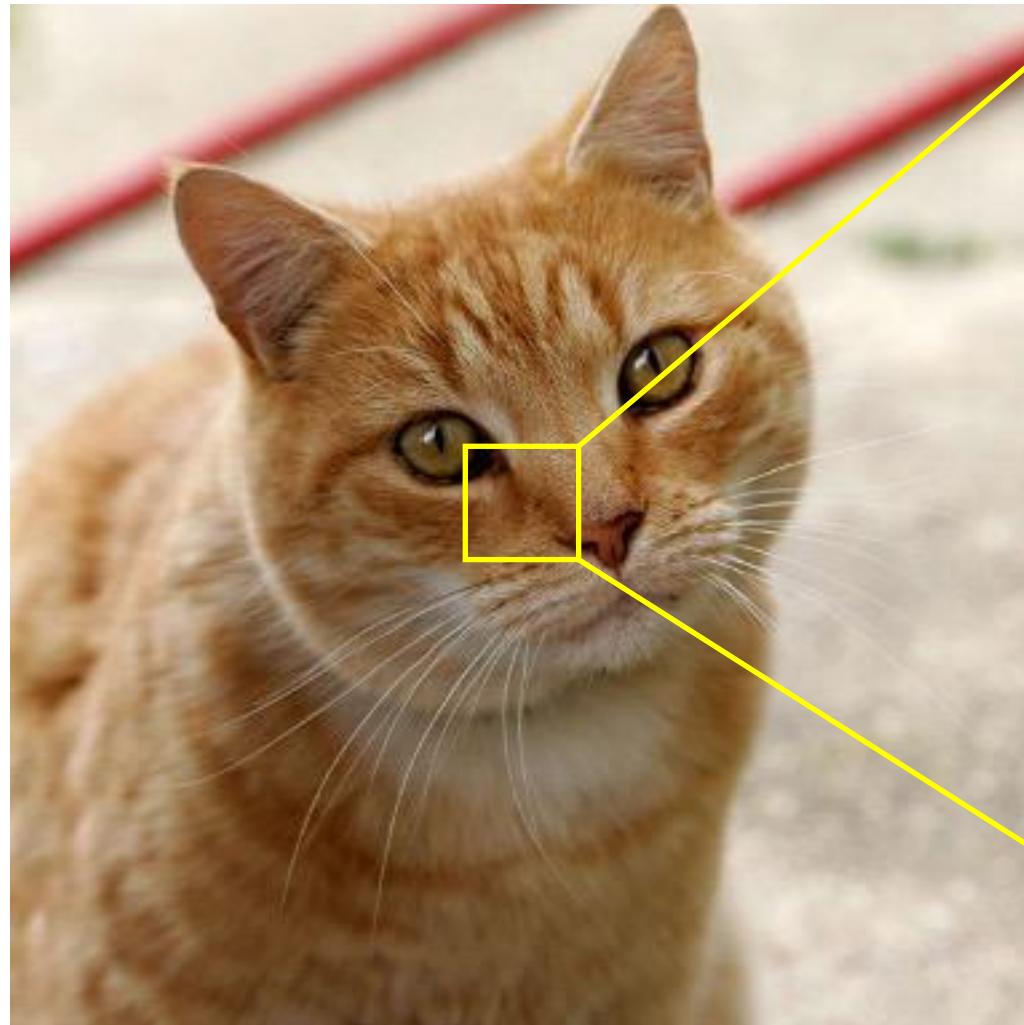
What we see



What computer sees

199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

What we see



What computer sees

222	218	210	182	190	186	198	206			
218	190	182	190	194	198	205	206	89		
202	174	108	185	192	188	180	189	151	93	
149	165	174	109	175	185	178	185	98	89	
149	174	184	175	168	172	174	176	36	93	
149	155	164	175	168	172	176	179	58	72	
145	145	164	163	189	188	181	181	42	68	
133	135	134	138	134	131	130	131	88	64	
	32	86	88	80	80	56	82	66	60	
	60	60	60	52	52	52	56	52	52	

MIT summer project

- Goal

To construct a system of programs which will divide a picture into regions such as,

- likely objects
- likely background areas
- chaos

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Group
Vision Memo. No. 100.

July 7, 1966

THE SUMMER VISION PROJECT

Seymour Papert

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

How long does it take?

- Already half a century since 1966.
- Numerous methods have been proposed.
- Still improving.
- One breakthrough happened in 2012 in the ImageNet challenge.

ImageNet challenge

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
 - Train an algorithm that can classify 1.2 million images into 1,000 classes.



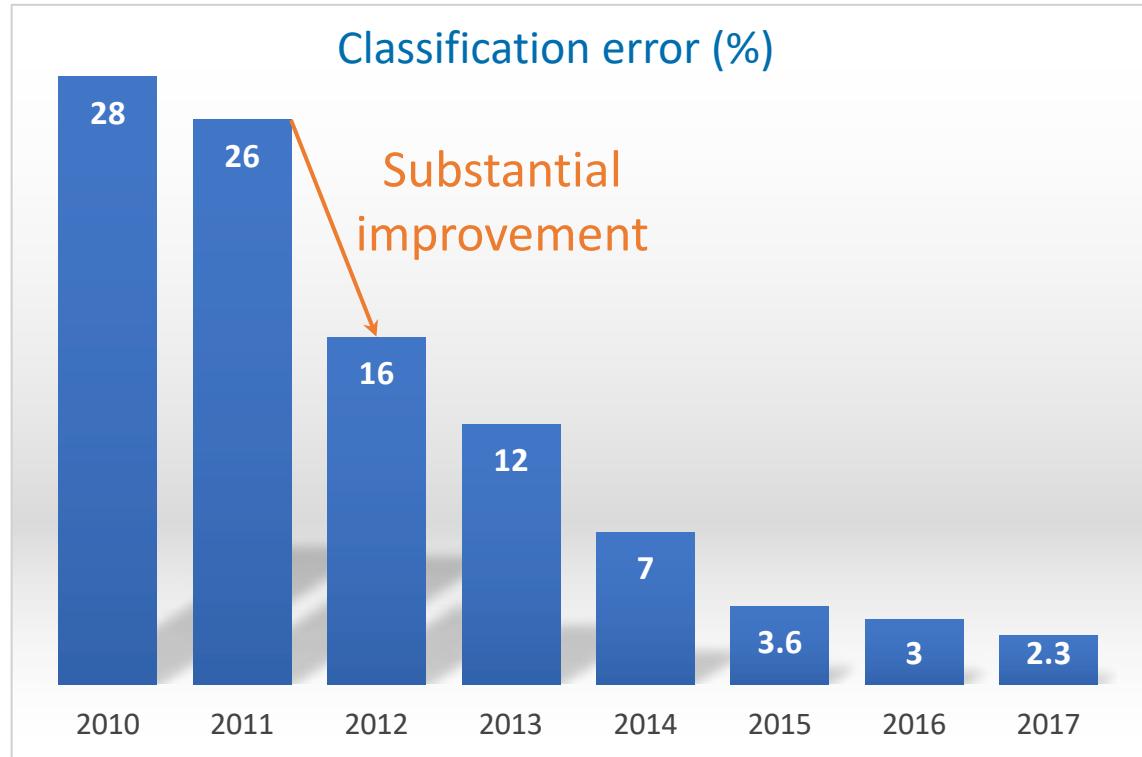
Fei-Fei Li



<http://www.image-net.org>

Classification challenge

Label: Steel drum



Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle



Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle



Classification challenge

- The improvement was made by Geoffrey Hinton's group at University of Toronto by using deep neural networks.
- A year later in 2013, a company founded by this group was acquired by Google.



Geoffrey Hinton

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.toronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.toronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.toronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

1 Introduction

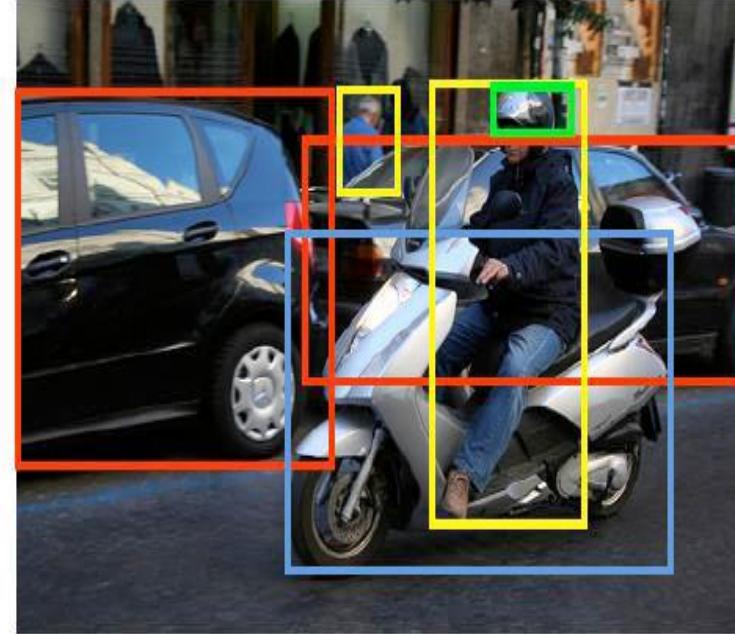
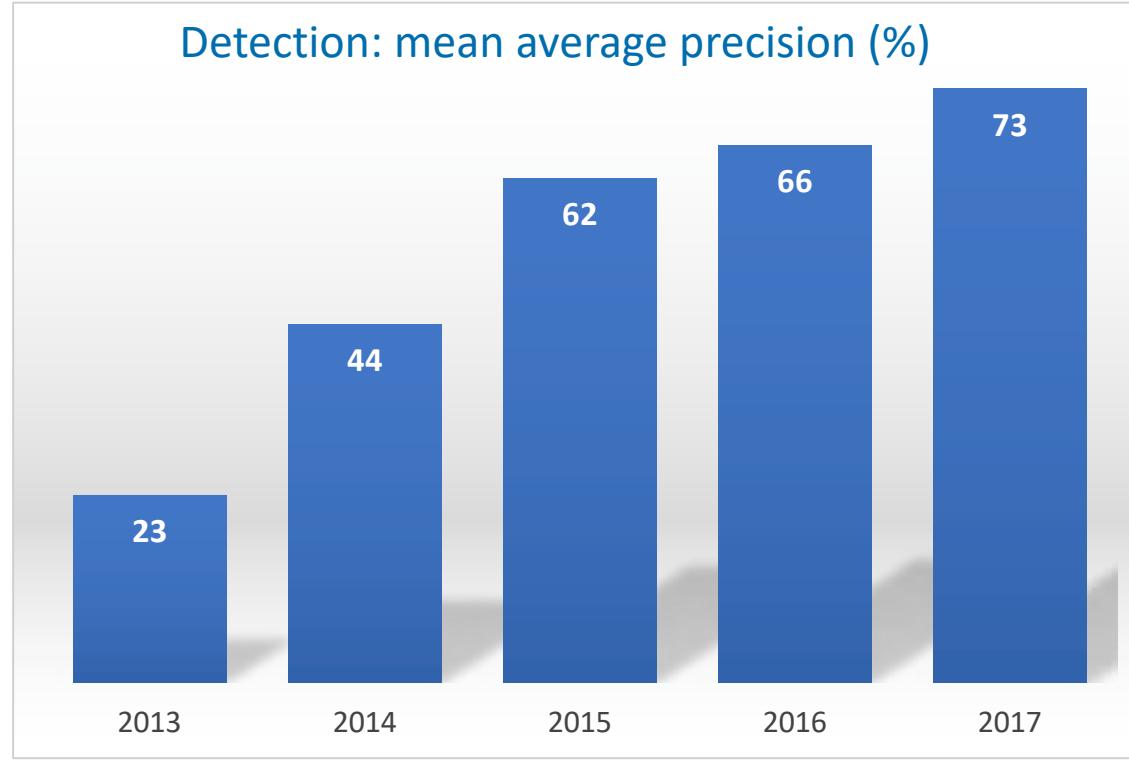
Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

1

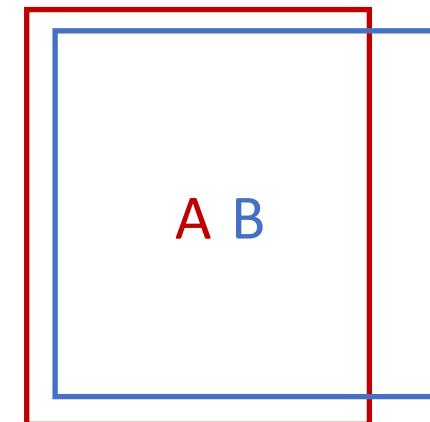
AlexNet

Detection challenge



Legend:

- Person (green)
- Car (red)
- Motorcycle (blue)
- Helmet (yellow)



$$\text{Intersection over Union} \quad IoU = \frac{A \cap B}{A \cup B}$$

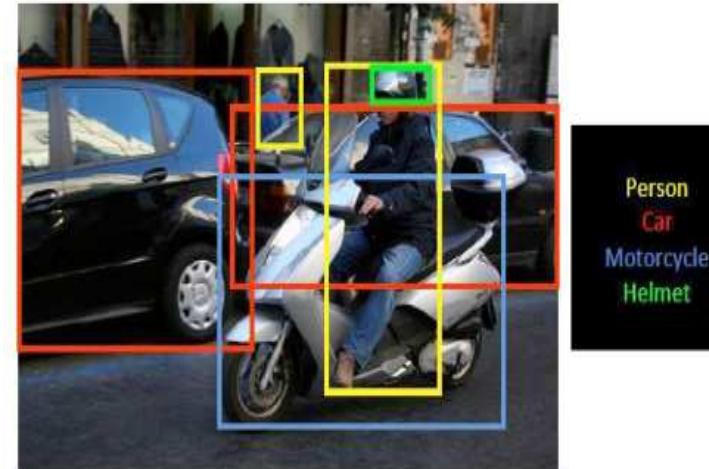
Detection is correct if $IoU > 0.5$.

What information can we extract?

- Image classification -> What is in the picture?
- Object detection -> Where are they roughly in the picture?
- Image segmentation -> Draw contours for each object accurately.



Classification: steel drum



Object detection

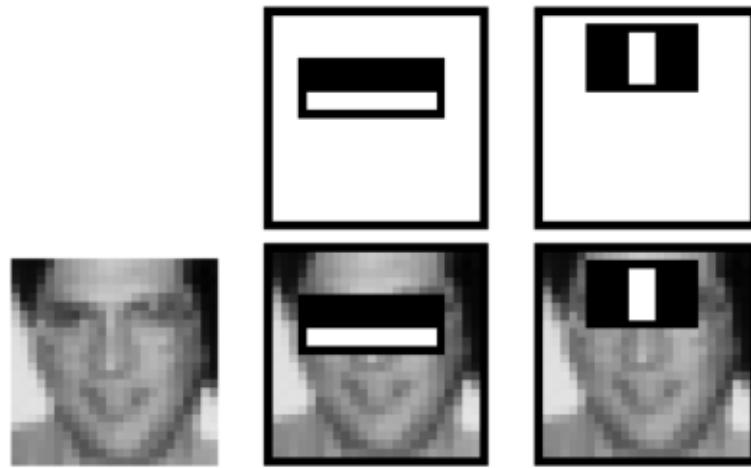


Image segmentation

Applications of computer vision

- It is already used in many applications in your daily life.

Face detection



Haar features



Face detection in camera



Face detection on Facebook

Automatic number plate recognition



<https://www.youtube.com/watch?v=VWGFr7g0qxU>

Recognition of street number



Street number of Huxley Building



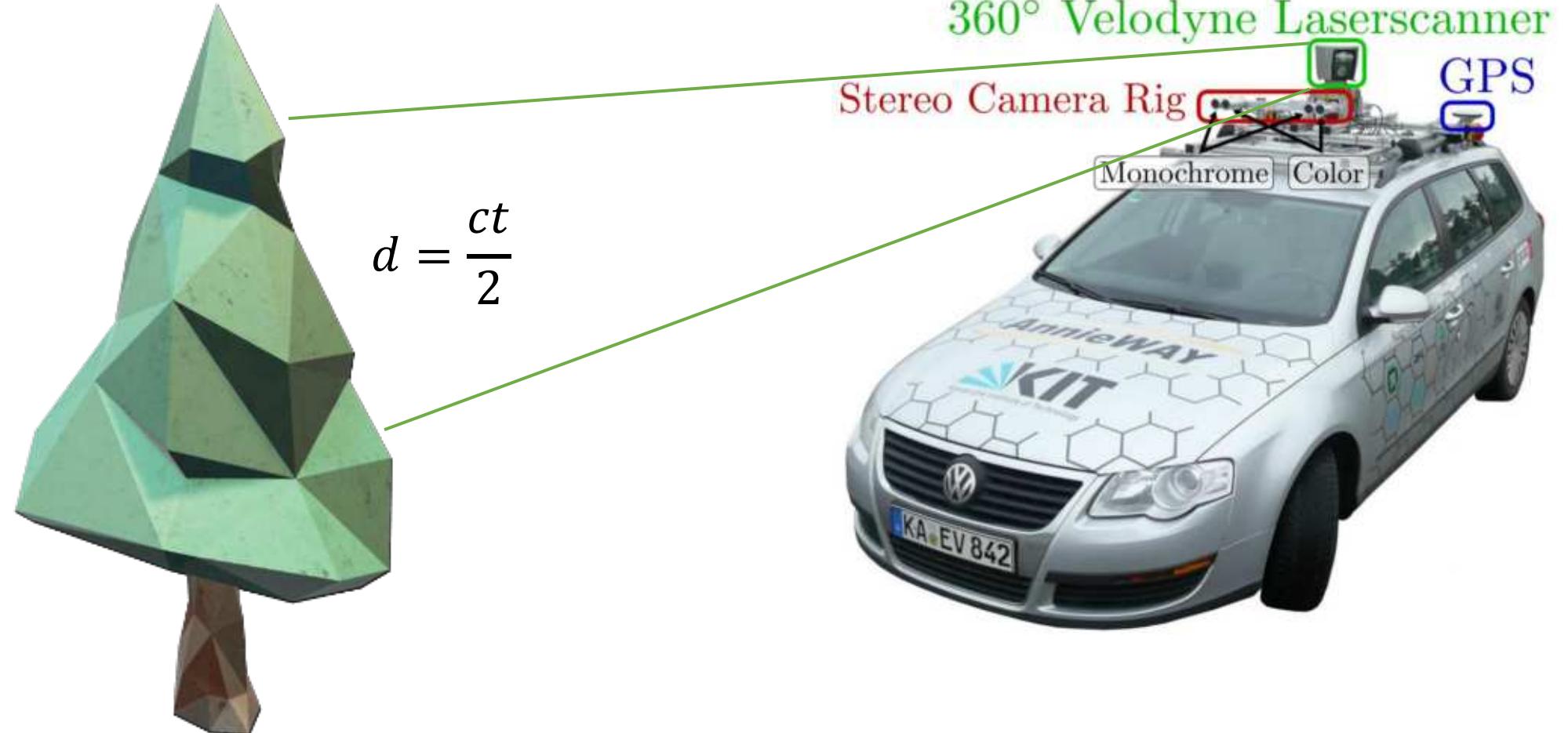
Street number overlaid on satellite image

Autonomous driving

- To get an understanding of the 3D world, autonomous driving cars use
 - Lidar (Light detection and ranging)
 - Camera



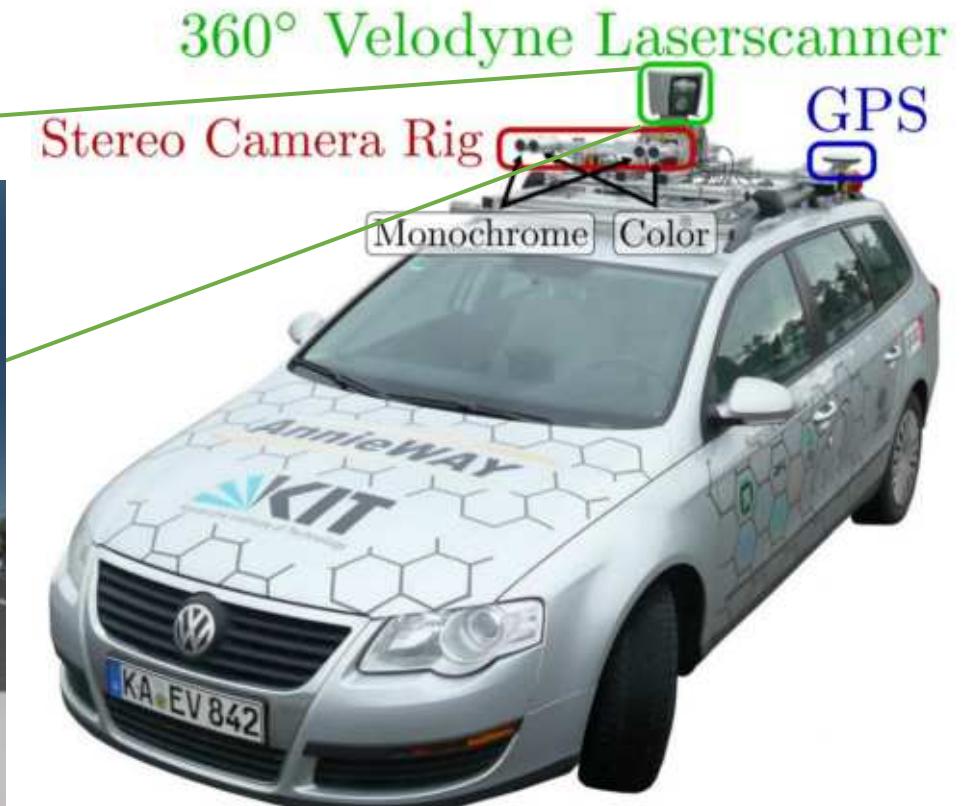
Autonomous driving



Autonomous driving



Autonomous driving



Autonomous driving

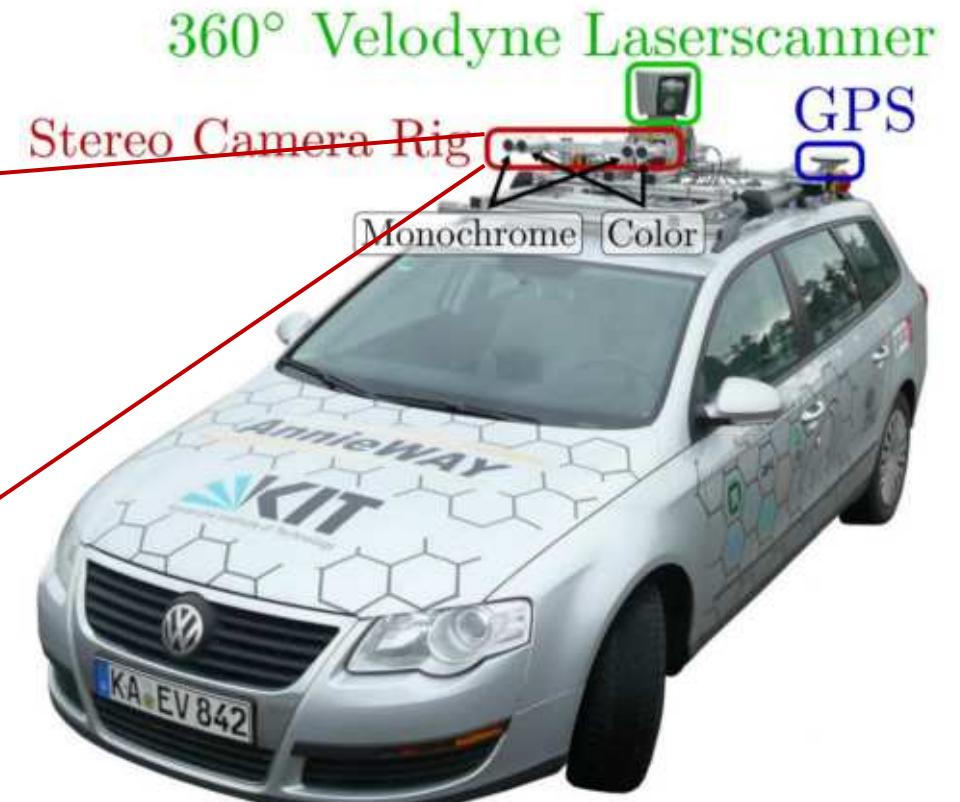
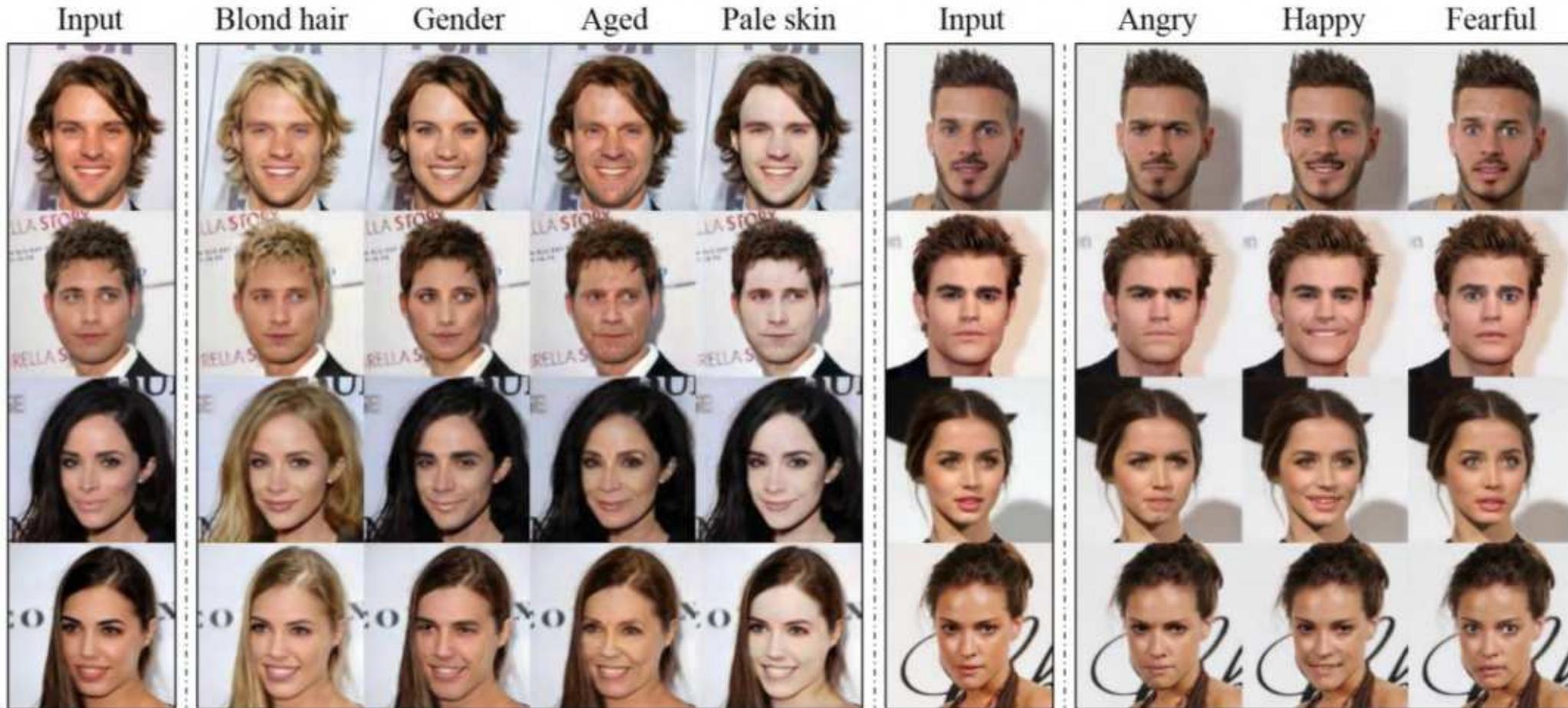


Image style transfer



Manipulating image attributes such as hair colour and mood

Face motion capture



Capture face motion by landmarks



Animate the figures in movie: Avatar, 2009

Face motion capture



Face swap

- With the availability of the technology, face swapped or fake videos can be made at home, even in real-time on Zoom meetings.
- It is sometimes called DeepFake.
- It brings other problems.
 - Fake news for politics
 - Fake videos of celebrities
 - Fake videos of normal people
 - ...



<https://www.youtube.com/watch?v=T76bK2t2r8g>

Kinect



Cameras



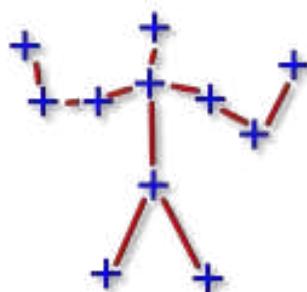
Depth image



Body parts



Key-points



Skeleton



Video assistant referee (VAR) in world cup





Camera

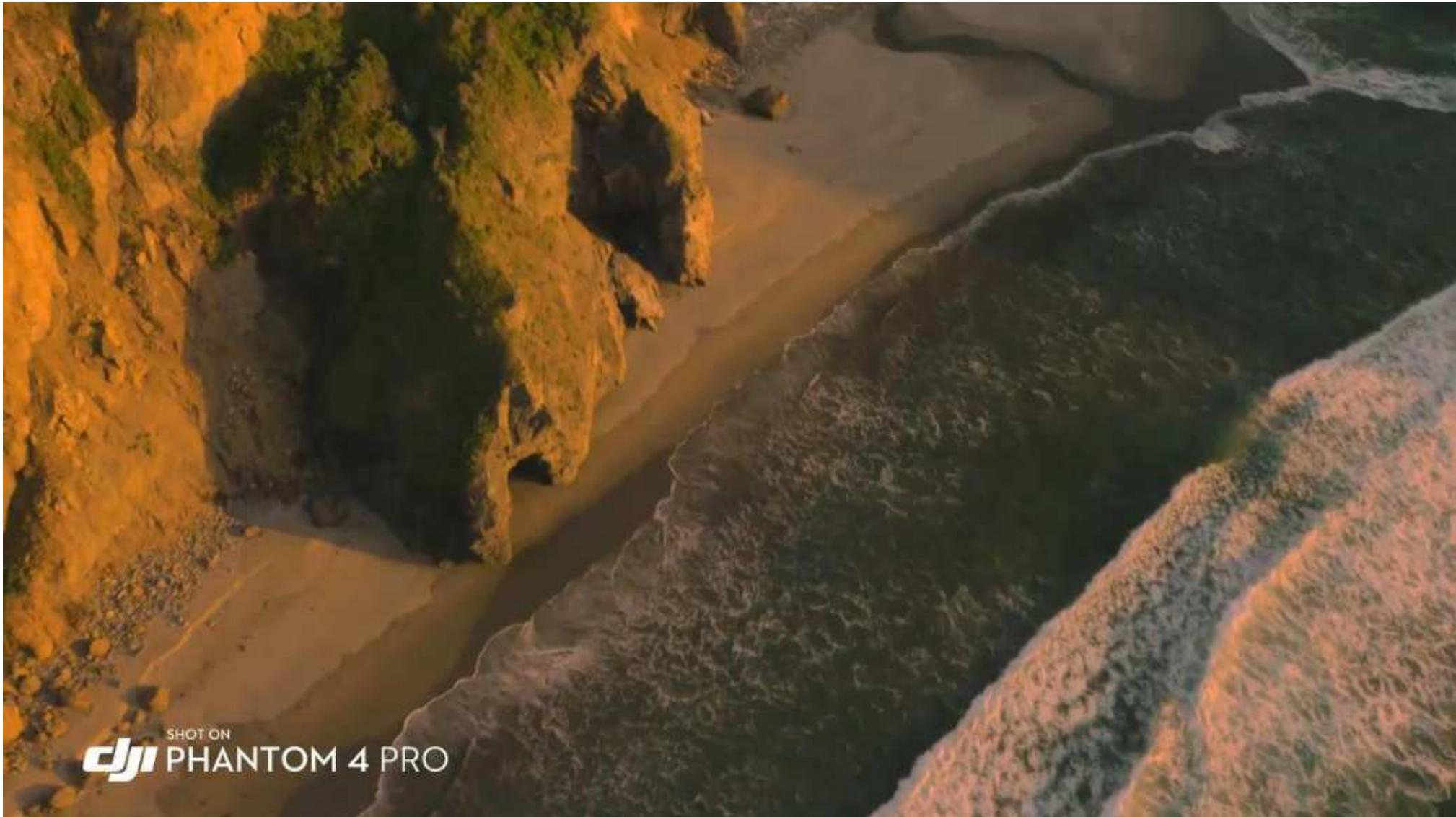
Camera

Camera

Camera



Motion tracking used by drones



Design

- OpenAI's DALL-E
 - Integrates computer vision with natural language processing.
 - Generates images by learning and combining the concepts of words in a sentence.



Healthcare

- Medical image analysis
- Machine learning for disease diagnosis



Healthcare

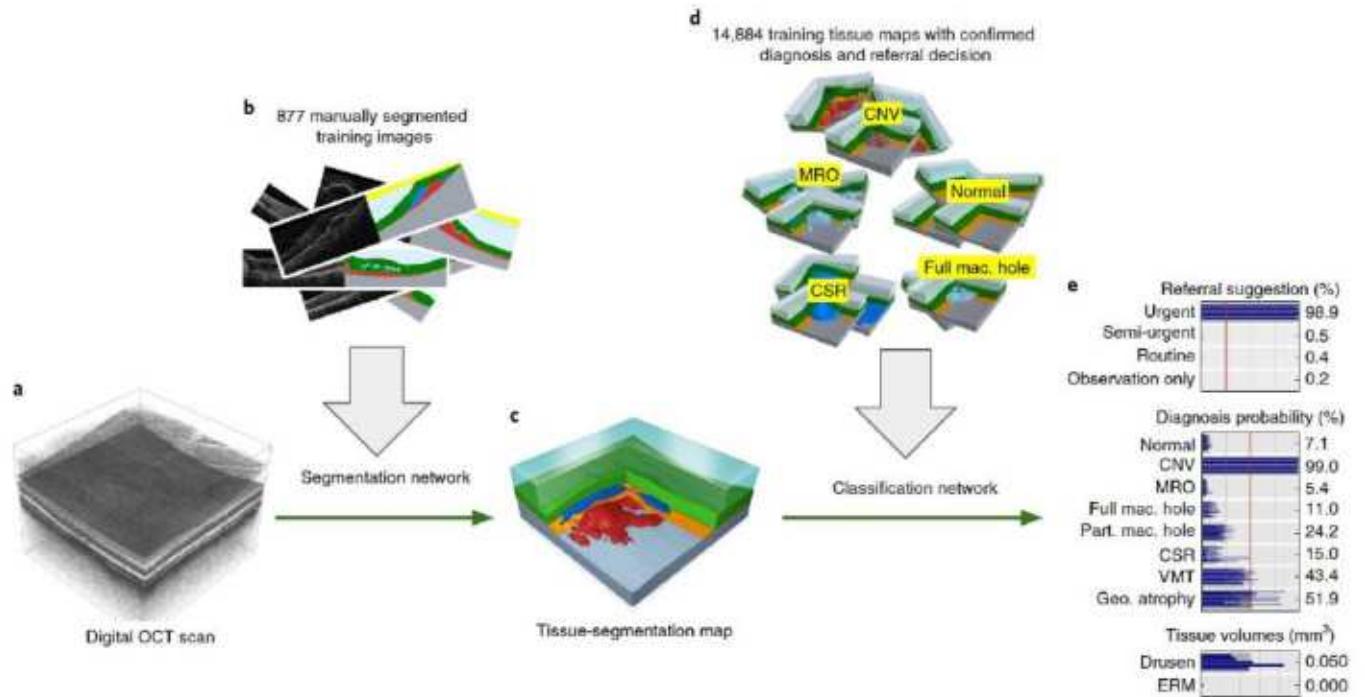


Image segmentation of retinal tissues, followed by classification.

ARTICLES

<https://doi.org/10.1038/nature2018-0107-4>

**nature
medicine**

Clinically applicable deep learning for diagnosis and referral in retinal disease

Jeffrey De Fauw¹, Joseph R. Ledsam¹, Bernardino Romera-Paredes¹, Stanislav Nikolov¹, Nenad Tomasev¹, Sam Blackwell¹, Harry Askham¹, Xavier Glorot¹, Brendan O'Donoghue¹, Daniel Visentin¹, George van den Driessche¹, Balaji Lakshminarayanan¹, Clemens Meyer¹, Faith Mackinder¹, Simon Bouton¹, Kareem Ayoub¹, Reena Chopra^{2,3}, Dominic King¹, Alan Karthikesalingam¹, Cian O. Hughes^{2,3}, Rosalind Raine³, Julian Hughes², Dawn A. Sim², Catherine Egan², Adnan Tufail², Hugh Montgomery^{2,3}, Demis Hassabis¹, Geraint Rees^{2,3}, Trevor Back¹, Peng T. Khaw², Mustafa Suleyman¹, Julien Cornebise^{1,2,4}, Pearse A. Keane^{2,4*} and Olaf Ronneberger^{1,4*}

The volume and complexity of diagnostic imaging is increasing at a pace faster than the availability of human expertise to interpret it. Artificial intelligence has shown great promise in classifying two-dimensional photographs of some common diseases and typically relies on databases of millions of annotated images. Until now, the challenge of reaching the performance of expert clinicians in a real-world clinical pathway with three-dimensional diagnostic scans has remained unsolved. Here, we apply a novel deep learning architecture to a clinically heterogeneous set of three-dimensional optical coherence tomography scans from patients referred to a major eye hospital. We demonstrate performance in making a referral recommendation that reaches or exceeds that of experts on a range of sight-threatening retinal diseases after training on only 14,884 scans. Moreover, we demonstrate that the tissue segmentations produced by our architecture act as a device-independent representation; referral accuracy is maintained when using tissue segmentations from a different type of device. Our work removes previous barriers to wider clinical use without prohibitive training data requirements across multiple pathologies in a real-world setting.

Medical imaging is expanding globally at an unprecedented rate¹, leading to an ever-expanding quantity of data that requires human expertise and judgement to interpret and triage. In many clinical specialties there is a relative shortage of this expertise to provide timely diagnosis and referral. For example, in ophthalmology, the widespread availability of optical coherence tomography (OCT) has not been matched by the availability of expert humans to interpret scans and refer patients to the appropriate clinical care². This problem is exacerbated by the marked increase in prevalence of sight-threatening disease for which OCT is the gold standard of initial assessment³.

Artificial intelligence (AI) provides a promising solution for such medical image interpretation and triage, but despite recent breakthroughs in which expert-level performance on two-dimensional photographs in preclinical settings has been demonstrated⁴, prospective clinical application of this technology remains stymied by three key challenges. First, AI (typically trained on hundreds of thousands of examples from one canonical dataset) must generalise to new populations and devices without a substantial loss of performance, and without prohibitive data requirements for retraining. Second, AI tools must be applicable in real-world scans, problems and pathways, and designed for clinical evaluation and deployment. Finally, AI tools must match or exceed the performance of human experts in such real-world situations. Recent work applying AI to

OCT has shown promise in resolving some of these criteria in isolation, but has not yet shown clinical applicability by resolving all three.

Results

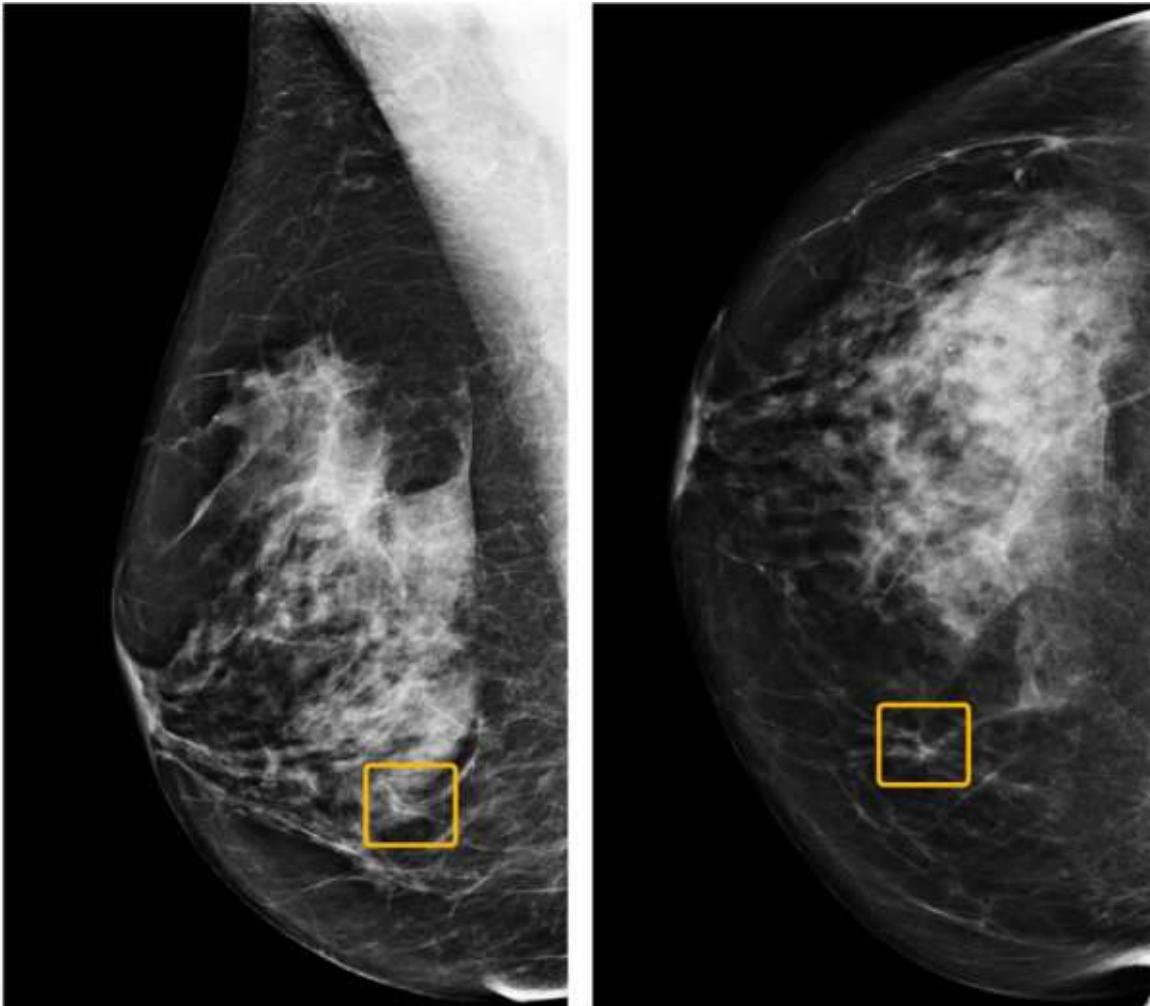
Clinical application and AI architecture. We developed our architecture in the challenging context of OCT imaging for ophthalmology. We tested this approach for patient triage in a typical ophthalmology clinical referral pathway, comprising more than 50 common diagnoses for which OCT provides the definitive imaging modality (Supplementary Table 1). OCT is a three-dimensional volumetric medical imaging technique analogous to three-dimensional ultrasound but measuring the reflection of near-infrared light rather than sound waves at a resolution for near-human tissue of ~5 µm. OCT is now one of the most common imaging procedures, with 3.35 million OCT scans performed in the US Medicare population in 2014 alone (see <https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Physician-and-Other-Supplier.html>). It has been widely adopted across the UK National Health Service (NHS) for comprehensive initial assessment and triage of patients requiring rapid non-elective assessment of acute and chronic sight loss. Rapid access 'virtual' OCT clinics have become the standard of care⁴. In such clinics, expert clinicians interpret the OCT and clinical history to diagnose and triage patients with

¹DeepMind, London, UK; ²NHRI Biomedical Research Centre at Moorfields Eye Hospital and UCL Institute of Ophthalmology, London, UK

³Present address: University College London, London, UK. ⁴These authors contributed equally: Julien Cornebise, Pearse A. Keane, Olaf Ronneberger

*e-mail: pearse.keane@medsch.ucl.ac.uk; olaf@deepradar.com

Healthcare



Detection of breast cancer lesions from mammograms

Collaboration between Google, DeepMind, Imperial College London, the NHS and Northwestern University. *Nature*, 2020.

Article

International evaluation of an AI system for breast cancer screening

<https://doi.org/10.1038/s43586-019-0793-0>

Received: 27 July 2019

Annexure 3 Annexure 3B

Published online: 8 January 2009

Scott Mayer McKinney^{13a}, Marcin Sieniek^{13b}, Varun Godbole^{13c}, Jonathan Godwin^{13d}, Natasha Antropiusova^{13e}, Hutan Ashrafian^{13f}, Trevor Back^{13g}, Mary Chesus^{13h}, Greg C. Corrado¹³ⁱ, Ara Darzi^{13j}, Muzaffer Etemadi^{13k}, Florencia Garcia-Vicente^{13l}, Rose J. Gilbert^{13m}, Mark Halling-Brown¹³ⁿ, Dennis Hassabis^{13o}, Sunny Jansen^{13p}, Alan Karthikseelamangal^{13q}, Christopher J. Kelly^{13r}, Dominic King^{13s}, Joseph R. Ladam^{13t}, David Melnick^{13u}, Hormuz Mostofi^{13v}, Lily Peng^{13w}, Joshua Jay Reicher^{13x}, Bercseniano Romeo-Paredes^{13y}, Richard Sibleton^{13z}, Mustafa Sulayman^{13aa}, Daniel Tsui^{13ab}, Kenneth C. Young^{13ac}, Jeffrey De Fauw^{13ad} & Shravya Shetty^{13ae}

Screening mammography aims to identify breast cancer at earlier stages of the disease, when treatment can be more successful¹. Despite the existence of screening programmes worldwide, the interpretation of mammograms is affected by high rates of false positives and false negatives². Here we present an artificial intelligence (AI) system that is capable of surpassing human experts in breast cancer prediction. To assess its performance in the clinical setting, we curated a large representative dataset from the UK and a large enriched dataset from the USA. We show an absolute reduction of 5.7% and 1.2% (USA and UK) in false positives and 9.4% and 2.7% in false negatives. We provide evidence of the ability of the system to generalize from the UK to the USA. In an independent study of six radiologists, the AI system outperformed all of the human readers: the area under the receiver operating characteristic curve (AUC-ROC) for the AI system was greater than the AUC-ROC for the average radiologist by an absolute margin of 11.5%. We ran a simulation in which the AI system participated in the double-reading process that is used in the UK, and found that the AI system maintained non-inferior performance and reduced the workload of the second reader by 88%. This robust assessment of the AI system paves the way for clinical trials to improve the accuracy and efficiency of breast cancer screening.

Breast cancer is the second leading cause of death from cancer in women,¹ but early detection and treatment can considerably improve outcomes.²⁻³ As a consequence, many developed nations have implemented large-scale mammography screening programmes. Major medical and governmental organizations recommend screening for all women starting between the ages of 40 and 50^{4,5}. In the USA and UK, combined, over 42 million exams are performed each year.^{6,7}

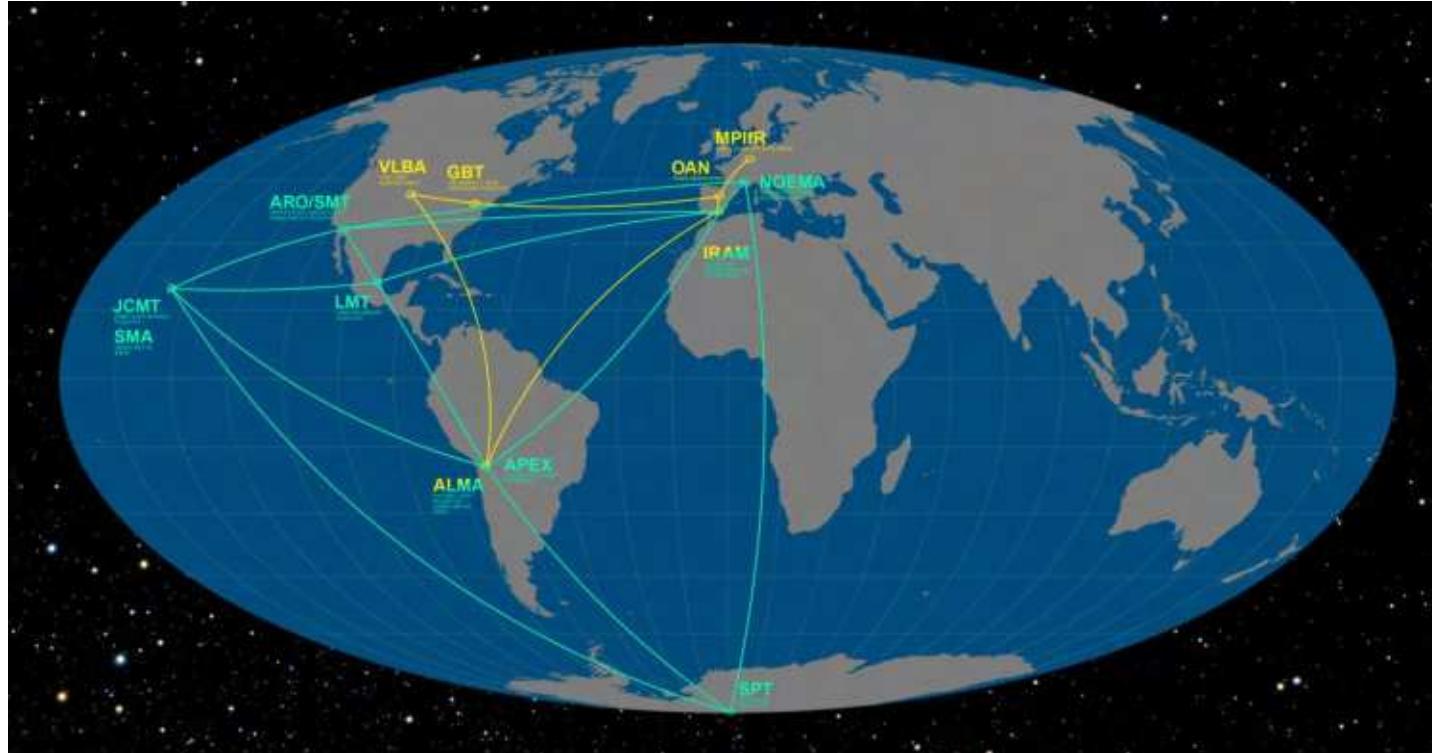
Despite the widespread adoption of mammography, interpretation of these images remains challenging. The accuracy achieved by experts in cancer detection varies widely, and the performance of even the best clinicians leaves room for improvement^{12,13}. False positives can lead to patient anxiety¹⁴, unnecessary follow-up and invasive diagnostic procedures. Cancers that are missed at screening may not be identified until they are more advanced and less amenable to treatment¹⁵.

AI may be uniquely poised to help with this challenge. Studies have demonstrated the ability of AI to meet or exceed the performance of human experts on several tasks of medical-image analysis.¹⁻⁶

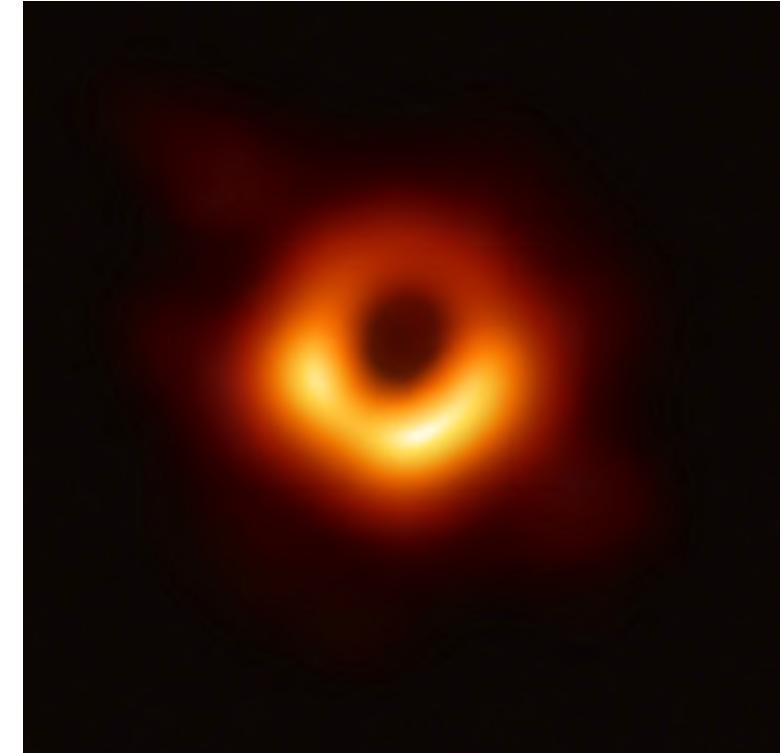
As a shortage of mammography professionals threatens the availability and adequacy of breast screening services around the world¹⁰⁻¹², the availability of AI could improve access to high-quality care for all.

Computer-aided detection (CAD) software for mammography was introduced in the 1970s, and several auxiliary tools have been approved or medical use.¹³ Despite early promise,¹⁴ this generation of software failed to improve the performance of readers in real-world settings.¹⁵ More recently, the field has seen a renaissance owing to the success of deep learning. A few studies have characterized systems for breast cancer prediction with stand-alone performance that approaches that of human experts.¹⁶⁻¹⁸ However, the existing work has several limitations. Most studies are based on small, enriched datasets with limited follow-up, and few have compared performance to readers in actual clinical practice—instead relying on laboratory-based simulations of the reading environment. So far there has been little evidence of the ability of AI systems to translate between different screening populations and settings without additional training data.¹⁹ Critically, the pervasiveness of follow-up intervals that are no longer than 12 months²⁰ makes

Astronomy



Event Horizon Telescope (EHT): a large telescope array.



First image of the shadow of a black hole (M87*) captured by the EHT.

An image reconstruction algorithm of the black hole was developed by the computer vision community.

What we will learn in this course

- Image formation
- Image filtering
- Feature description
- Image classification
- Object recognition
- Image segmentation
- Motion tracking
- Camera model

Prerequisites

- Programming in Python
- Linear algebra, calculus and probability
- Some knowledge about machine learning

	Monday 9:00 Huxley 308	Monday 10:00 Huxley 308	Thursday 9:00 Huxley 308	Thursday 10:00 Huxley 308 or Lab
Week 2 16 Jan	Introduction	Image formation	Image filtering I	Tutorial, Coursework 1 out
Week 3 23 Jan	Image filtering II	Edge detection I	Edge detection II	Lab
Week 4 30 Jan	Hough Transform	Interest point detection I	Interest point detection II	Tutorial, Coursework 1 due
Week 5 6 Feb	Feature description I	Feature description II	Image classification I	Tutorial, Coursework 2 out
Week 6 13 Feb	Image classification II	Image classification III	Image classification IV	Lab
Week 7 20 Feb	Object detection	Image segmentation	Image segmentation	Tutorial, Coursework 2 due
Week 8 27 Feb	Motion I	Motion II	Camera model	Tutorial
Week 9	Revision and self-study			
Week 10, 11	Exam week			

Logistics

- Where do we ask questions?
 - Weekly tutorial session at Huxley 308
 - EdStem discussion forum: <https://edstem.org/us/courses/29402/discussion/>
- Resources
 - Lecture slides, tutorial sheets, at Scientia:
<https://scientia.doc.ic.ac.uk/2223/modules/60006/materials>
 - Tutorial answers will be released the week after.
- Assessment
 - 2 courseworks: 20%, submission at Scientia
 - Final exam: 80%

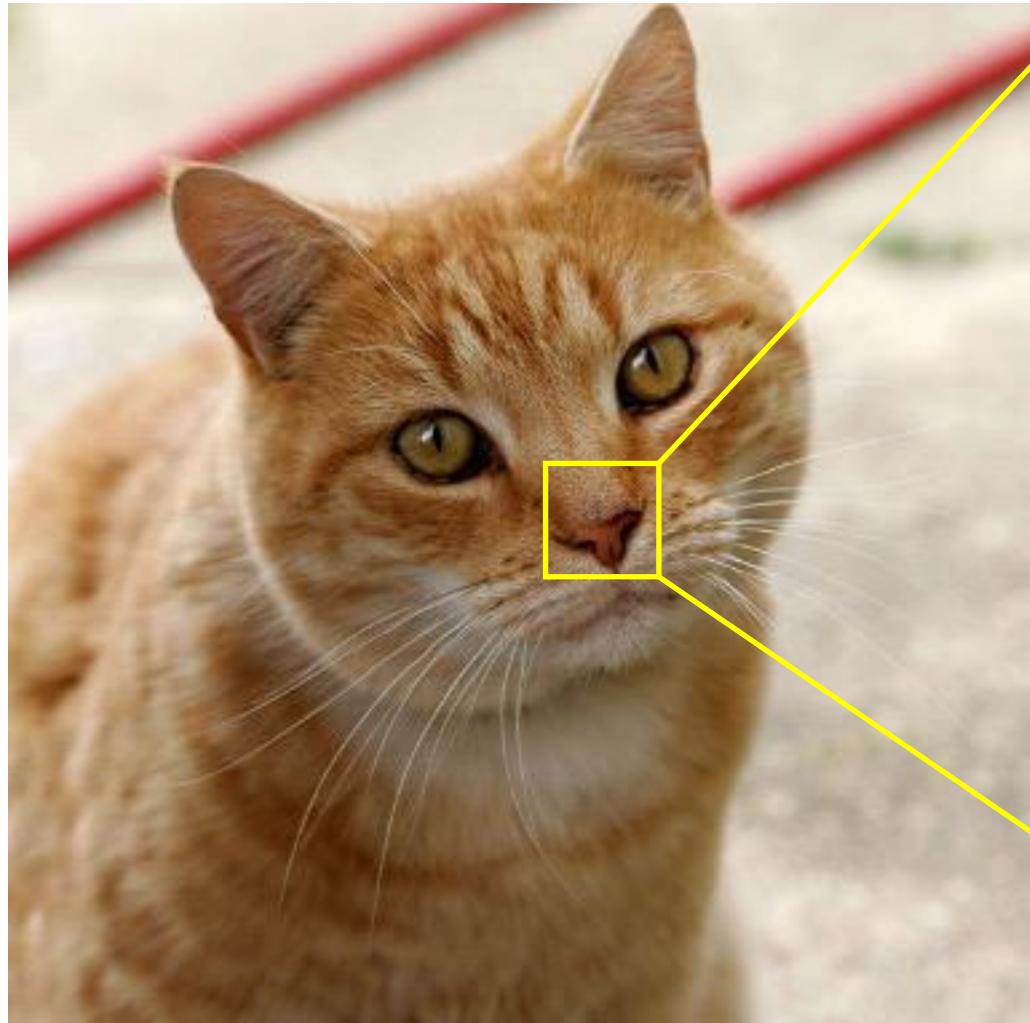
Any questions?

Image Formation

Dr Wenjia Bai

Department of Computing & Brain Sciences

How is an image formed?

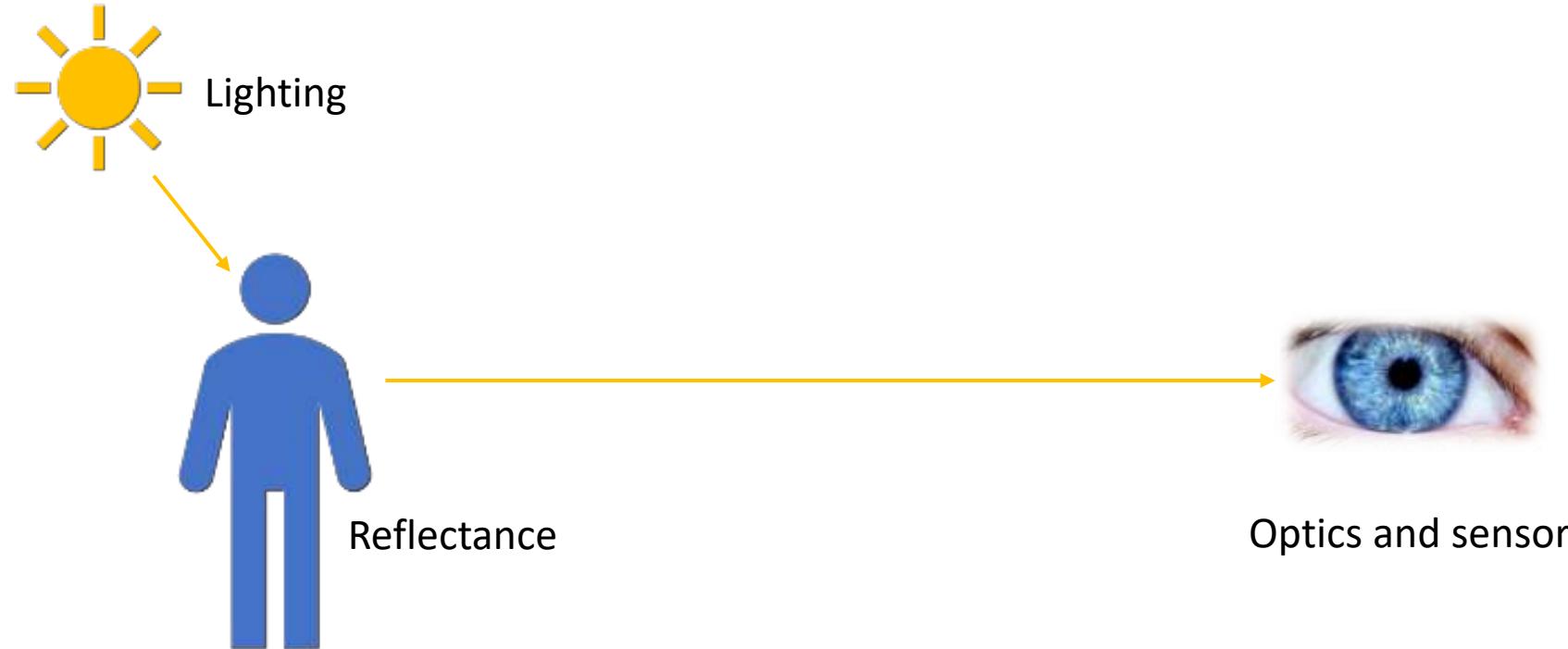


222	218	210	182	190	186	198	206		
218	190	182	198	194	198	895	206	89	
202	174	188	185	194	188	180	189	185	93
149	165	174	179	185	185	176	185	98	89
149	174	184	184	152	168	158	176	56	93
149	145	151	170	158	152	158	170	58	72
145	145	165	168	168	158	152	181	171	68
133	135	184	183	183	181	180	181	88	64
	32	86	88	80	80	56	92	66	60
	60	60	60	52	52	52	56	52	

How is an image formed?

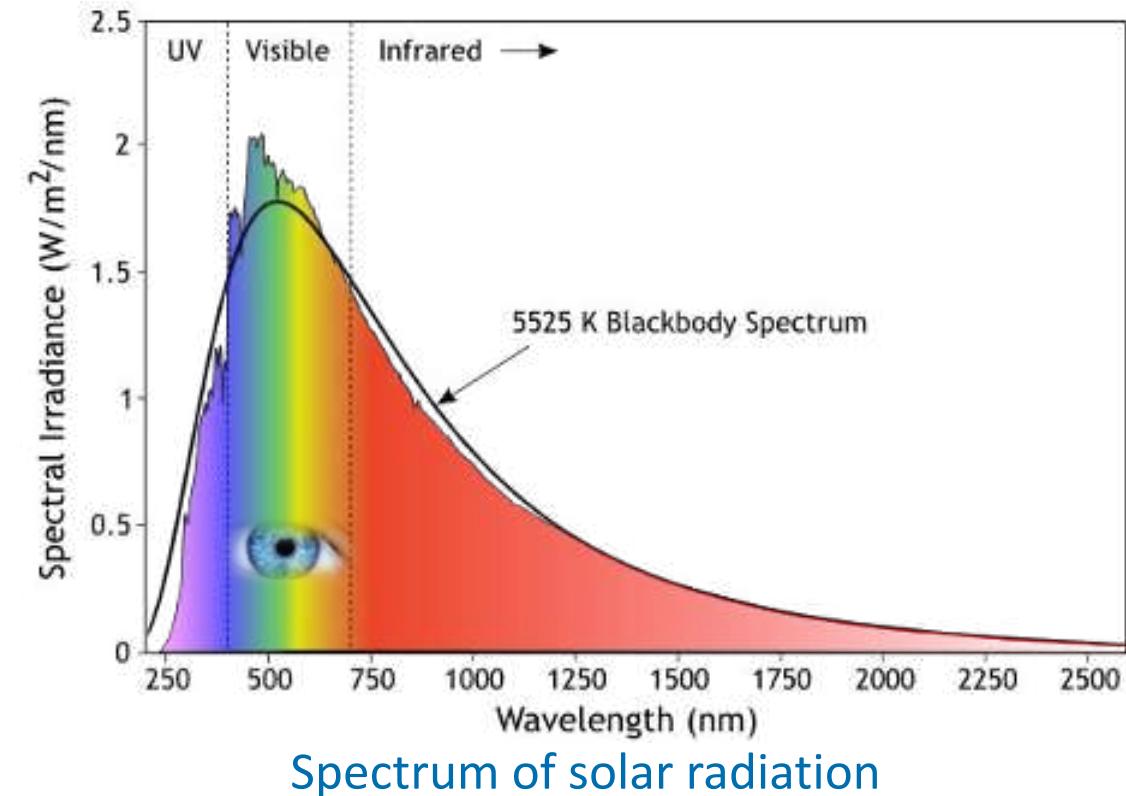
- Light
- Reflectance
- Optics and sensors
- Colours

How is a digital image formed?



Lighting

- Point light source
 - Originates from a single location in space
 - A small light bulb or the sun remotely
 - Properties
 - Location, intensity and spectrum
- Area light source
 - More complicated
 - Ceiling lights in the classroom

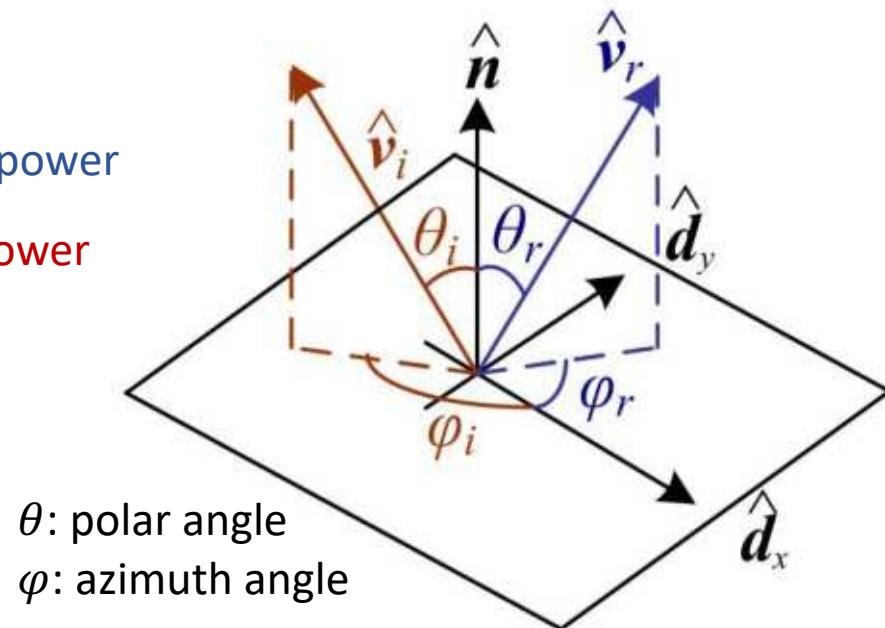


Reflectance

- Bidirectional reflectance distribution function (BRDF)
 - The most general model
 - Describes how much light arriving at an incident direction is emitted in a reflected direction.

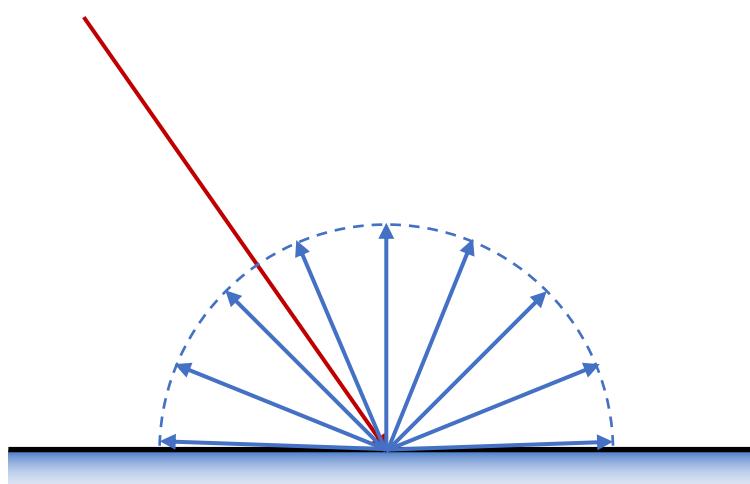
$$\text{BRDF: } f_r(\theta_i, \varphi_i, \theta_r, \varphi_r, \lambda) = \frac{dL_r}{dE_i} \frac{\text{output power}}{\text{input power}}$$

incident direction reflected direction wavelength

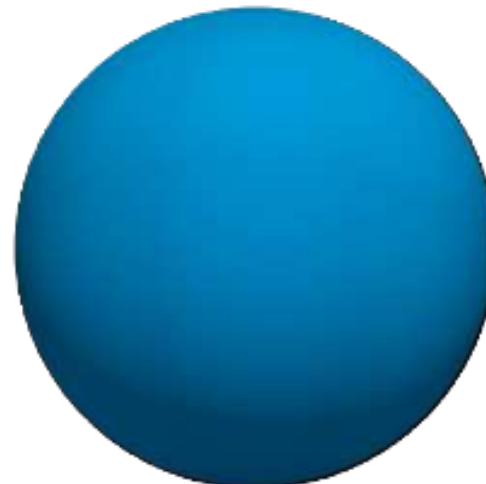


Diffuse reflection

- Light is scattered uniformly in all directions.
- The BRDF is constant: $f_r(\theta_i, \varphi_i, \theta_r, \varphi_r, \lambda) = f_r(\lambda)$

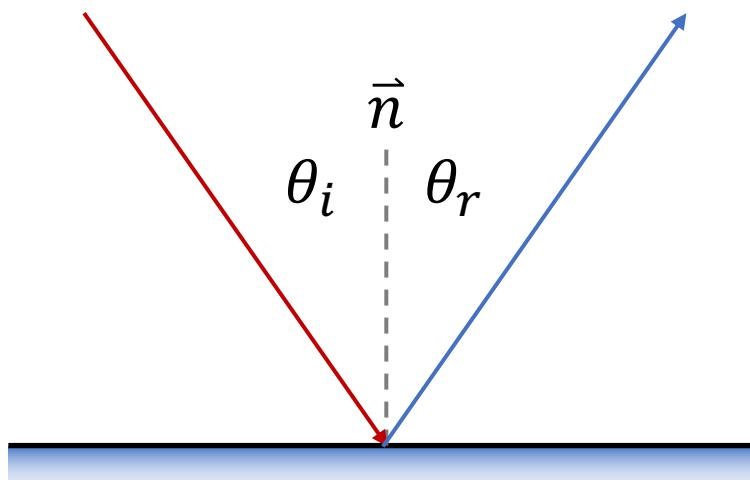


This is a rough surface.



Specular reflection

- Reflection in a mirror-like fashion.
- The reflection and incident directions are symmetric with respect to the surface normal \vec{n} : $\theta_r = \theta_i$



This is a very smooth surface.

Reflectance

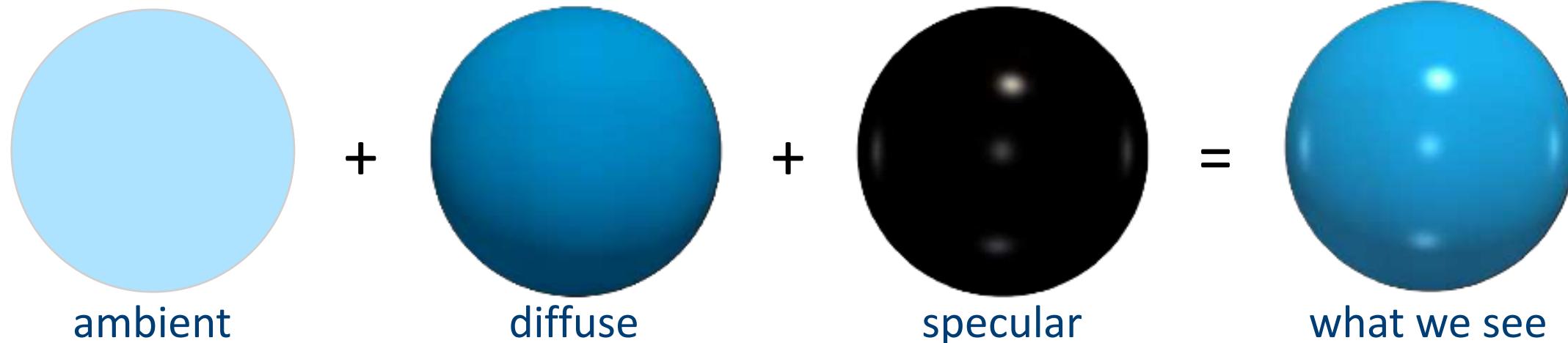
- Diffuse reflection and specular reflection are two ideal cases.
- For majority of the cases, what we see is a combination of
 - **diffuse** reflection
 - **specular** reflection
 - **ambient** illumination
- The ambient illumination accounts for general illumination which may be complicated to model, such as
 - inter-reflection between walls in a room
 - distant sources, e.g. blue sky, sunny outdoor

Phong reflection model



Bui Tuong Phong

- The Phong reflection model is an empirical model in computer graphics that describes how a surface reflects light as a combination of ambient, diffuse and specular components.

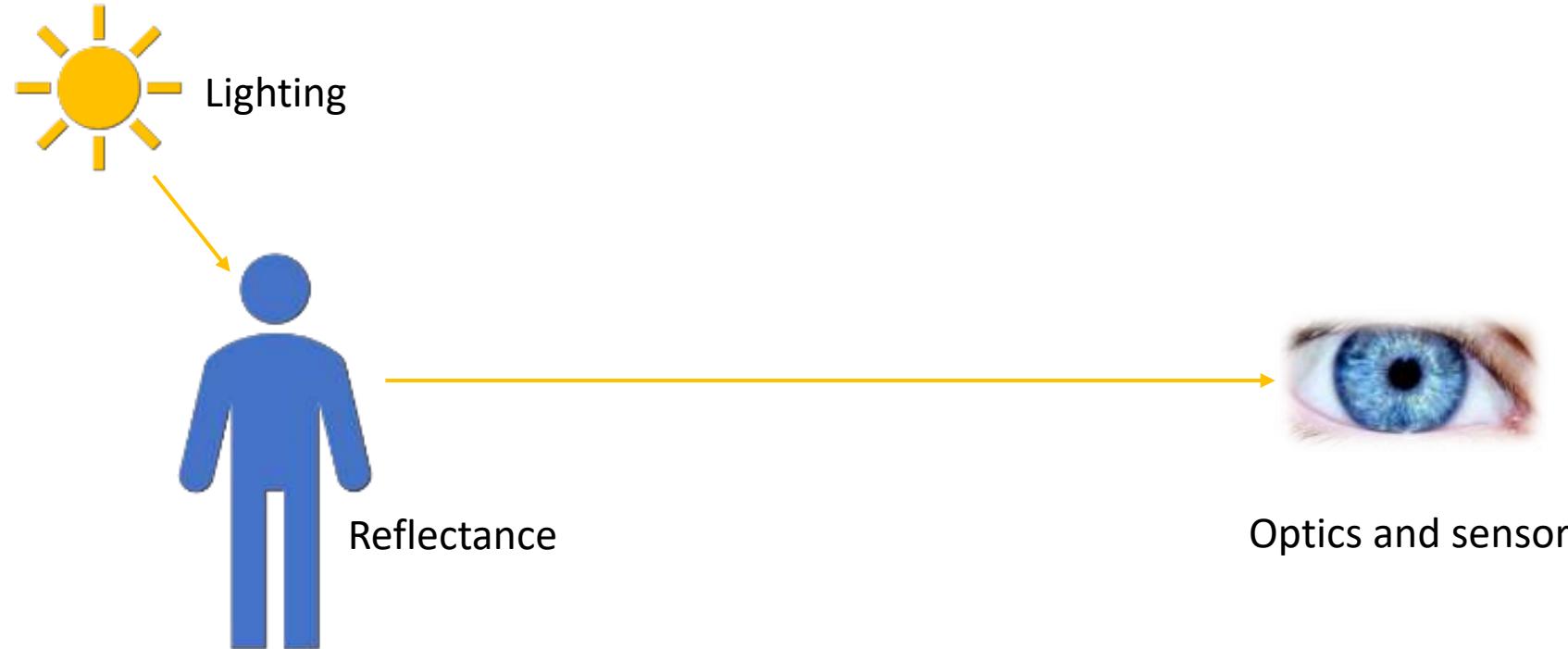


Computer graphics

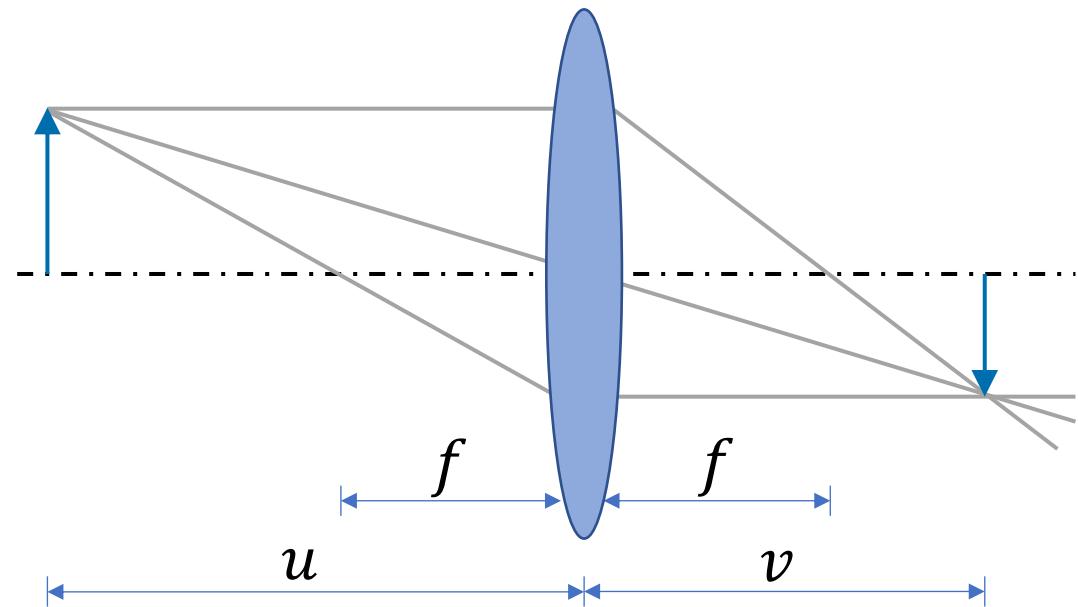


Final Fantasy: The Spirits Within, 2001 was the first photorealistic computer-animated feature film.

How is an image formed?

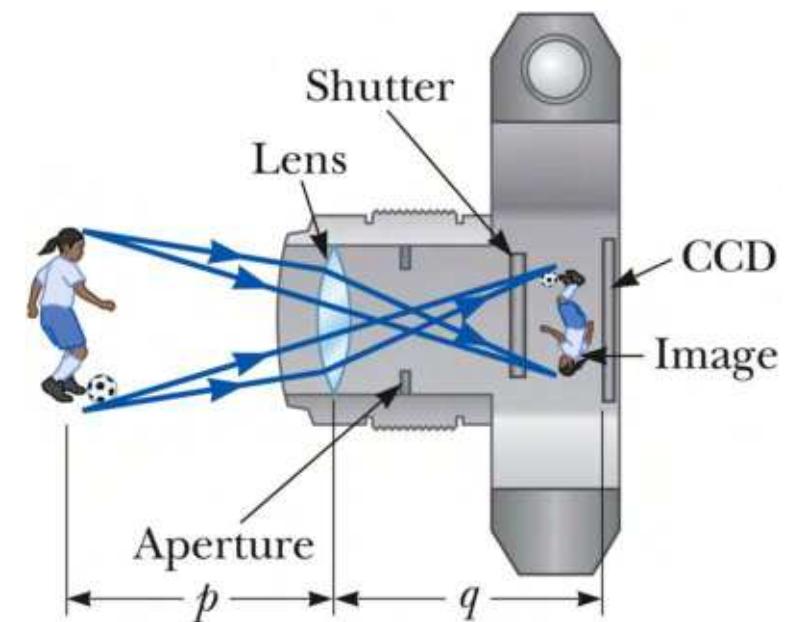
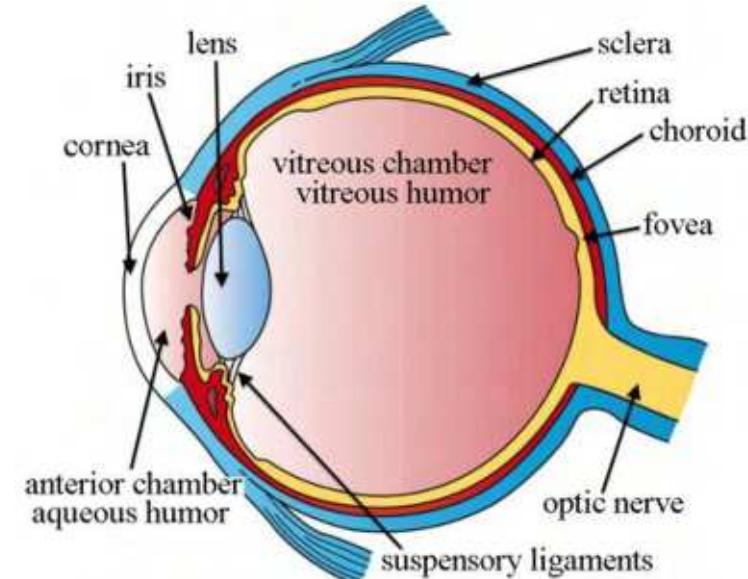


Optics



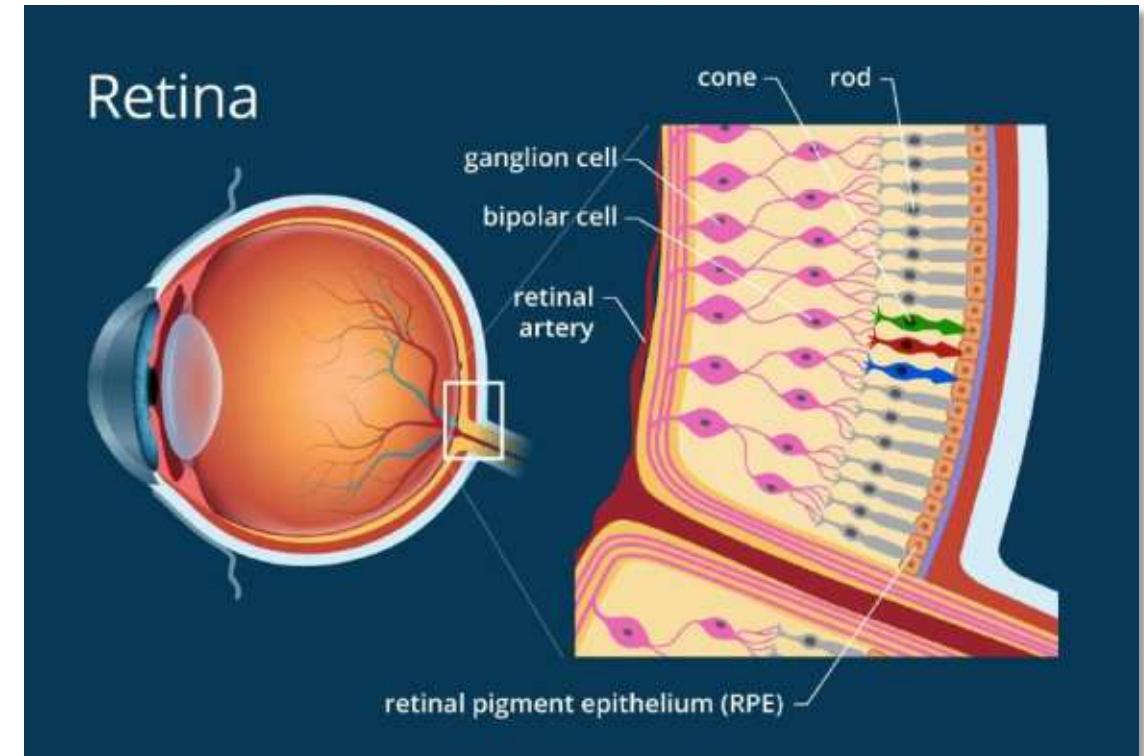
$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

Thin lens equation



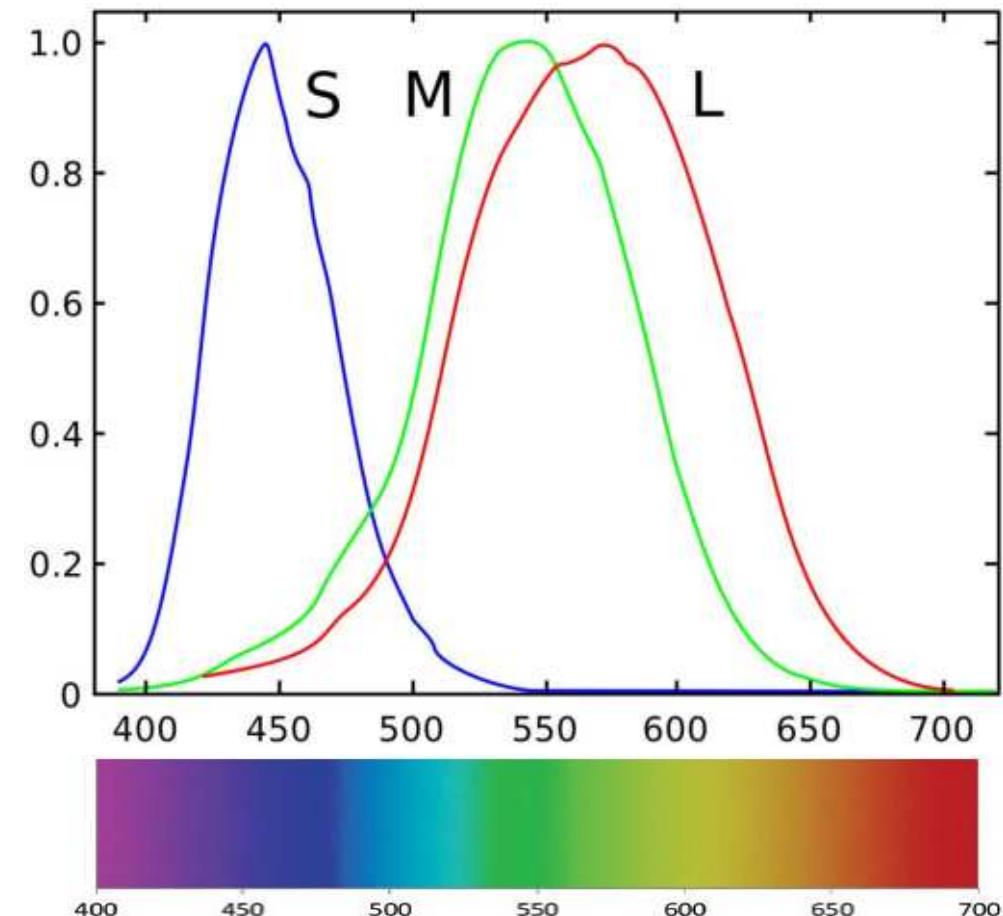
Human sensors

- Light rays are focused by the cornea and lens onto the retina, where the vision begins.
- Two types of neural cells in the retina.
- Cone cells
 - Responsible for colour vision
 - Function in bright light
- Rod cells
 - Little role in colour vision
 - Function in dim light



Cone cells

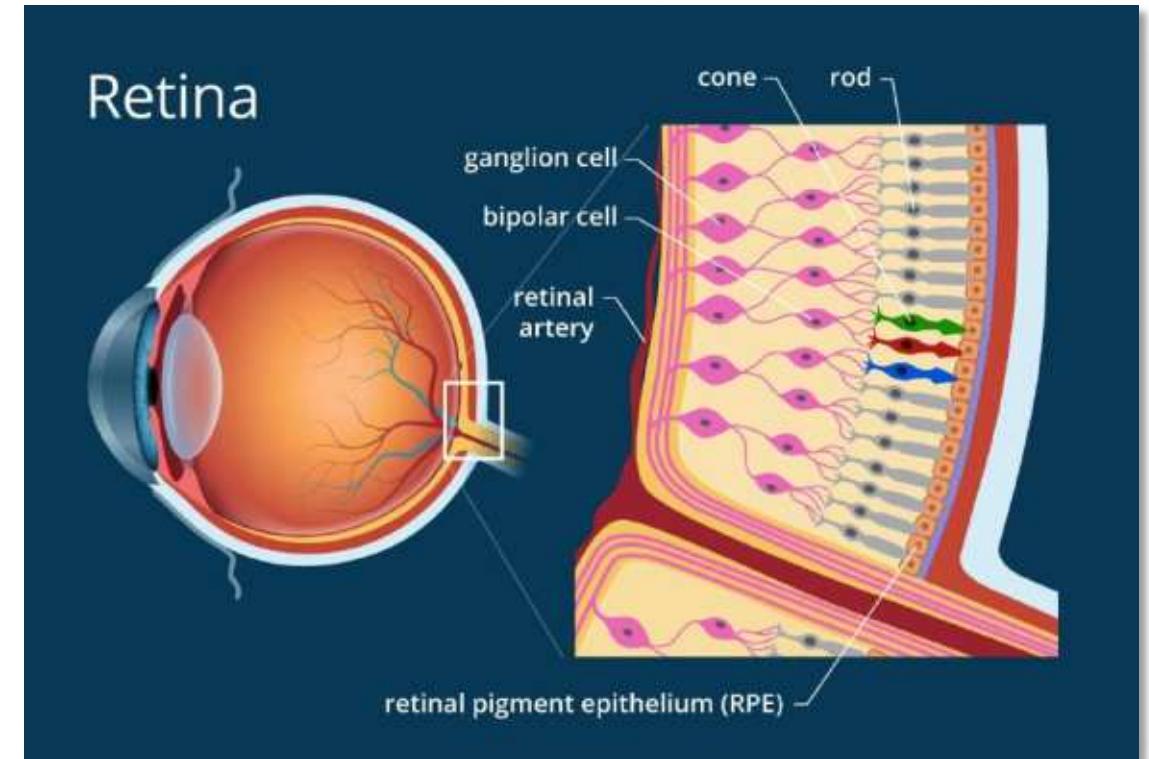
- Trichromatic vision (3)
- Humans and close related primates usually have three kinds of cones which have different response curves.
- Occasionally, you may also have
 - Dichromacy (2)
 - Tetrachromacy (4)



Responsivity spectra of three types of human cone cells, **S**, **M** and **L** types. **S** cones respond to short wavelength light; **M** to medium wavelength; **L** to long wavelength.

Rod cells

- Rod cells are more sensitive to light than cone cells.
- Since rod cells require less light to function, they are the primary source of visual information at night.



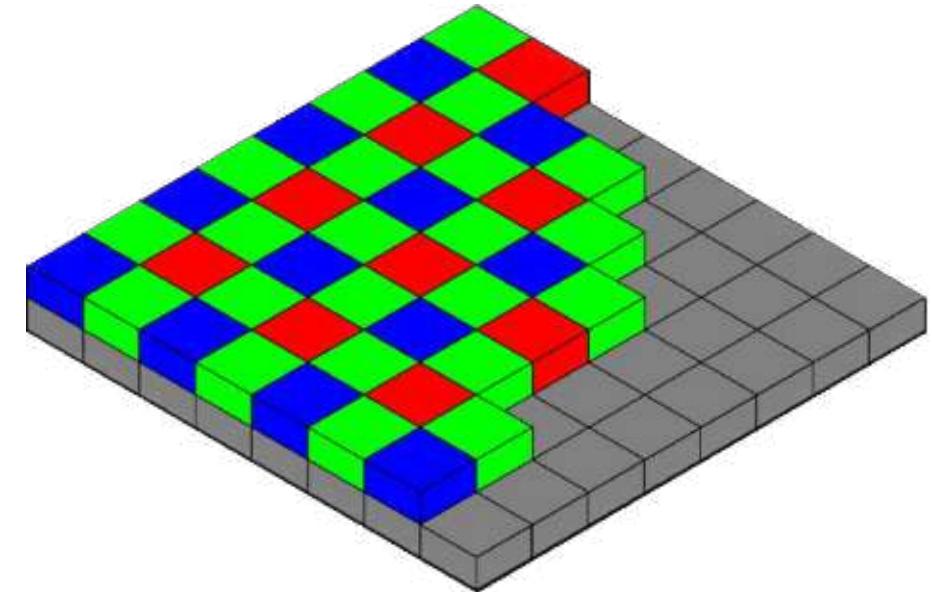
Camera sensors

- Two common types of sensors
 - CCD (charged-coupled device)
 - CMOS (complementary metal-oxide semiconductor)
 - Used by most smart phone cameras
- Sensors convert incoming photons into electron charges and read the values out.



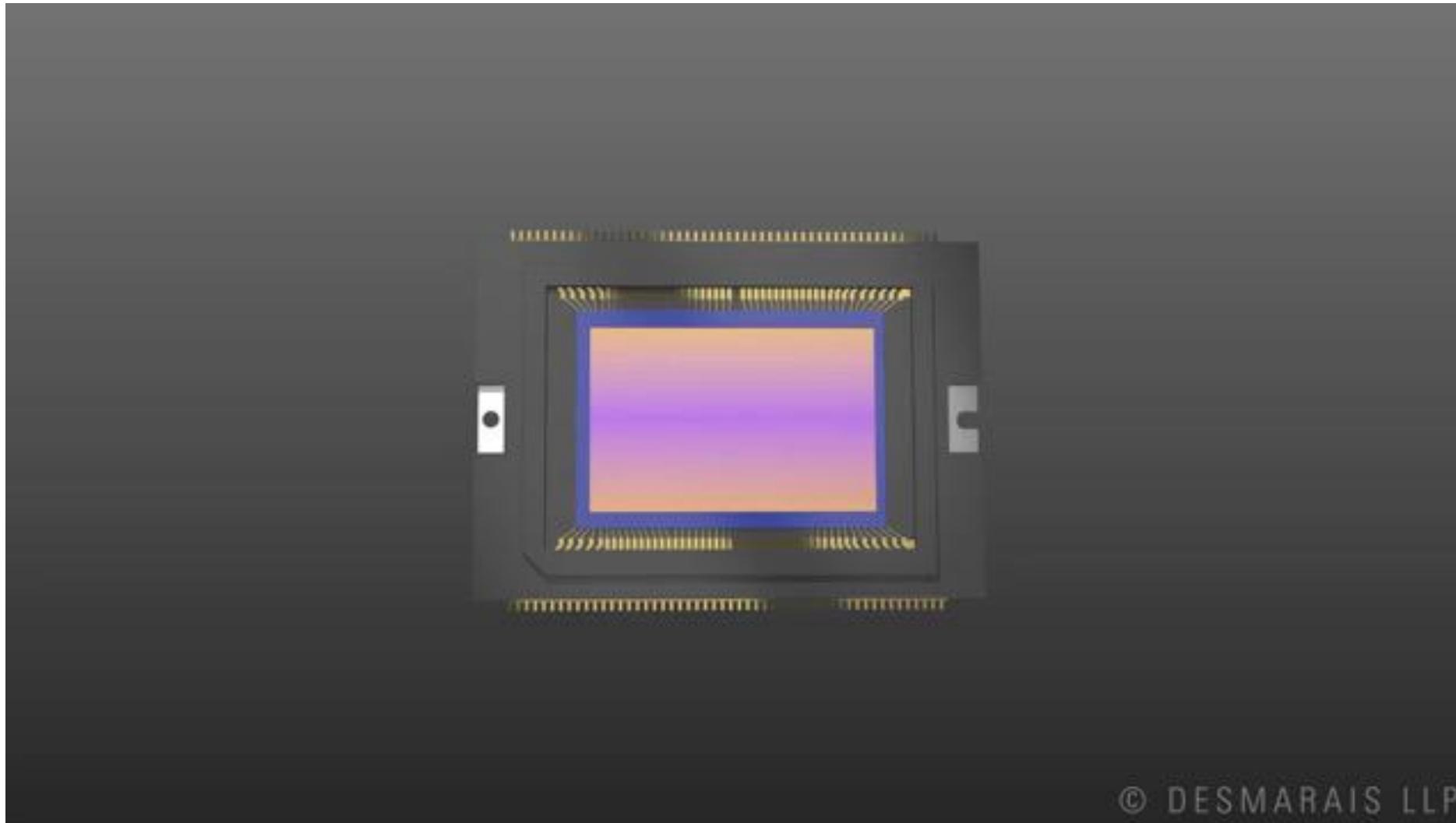
Bayer filter array on camera sensors

- A way to arrange RGB filters on sensors.
- Invented by Bryce Bayer of Kodak.
- 50% green, 25% red, 25% blue.
- Mimic the human eyes, which are most sensitive to green light.



Bayer filter

How CMOS works

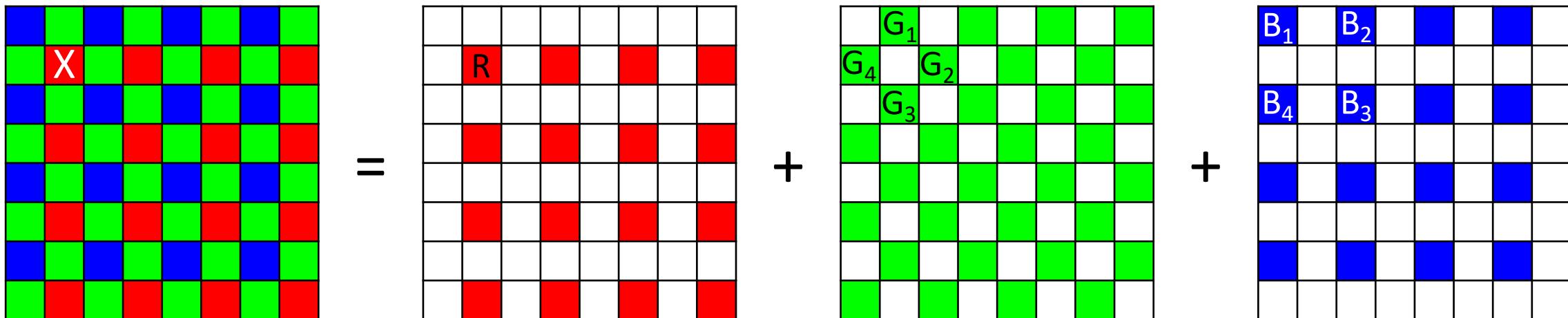


© DESMARAIS LLP

<http://digitalphotographylive.com/how-ccd-cmos-image-sensors-work>

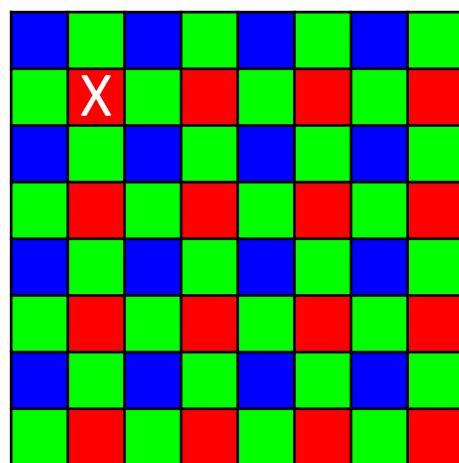
Bayer colour filter

- Only one colour is available at each pixel.
- The other two colours can be interpolated from neighbouring pixels.
- By interpolation, at each pixel, we can get (R, G, B) values.



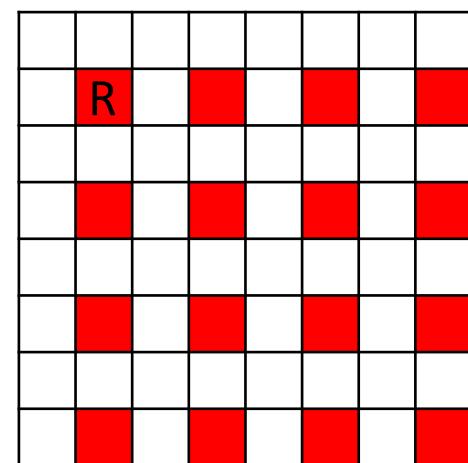
Demosaicing

- We use bilinear interpolation, simply averaging the 4 neighbours.
- When the pixel X moves, the neighbourhood will also change.



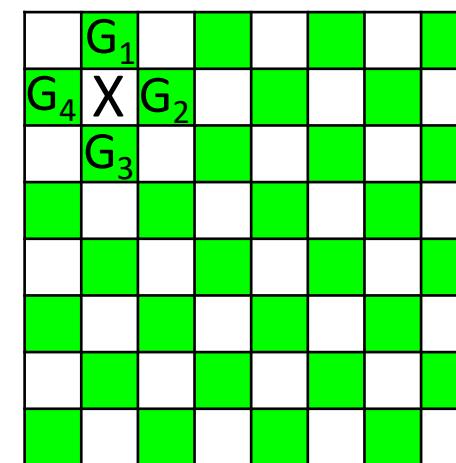
$$R_X = R$$

=



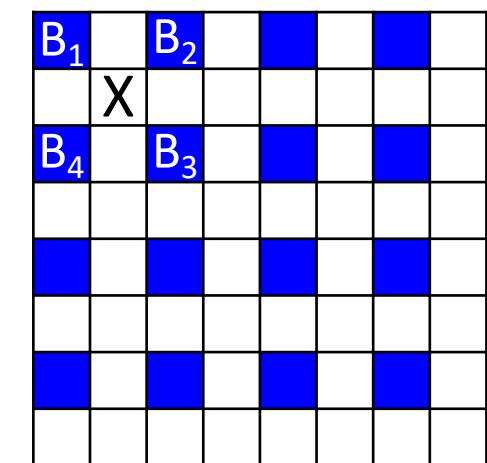
$$G_X = \frac{G_1 + G_2 + G_3 + G_4}{4}$$

+

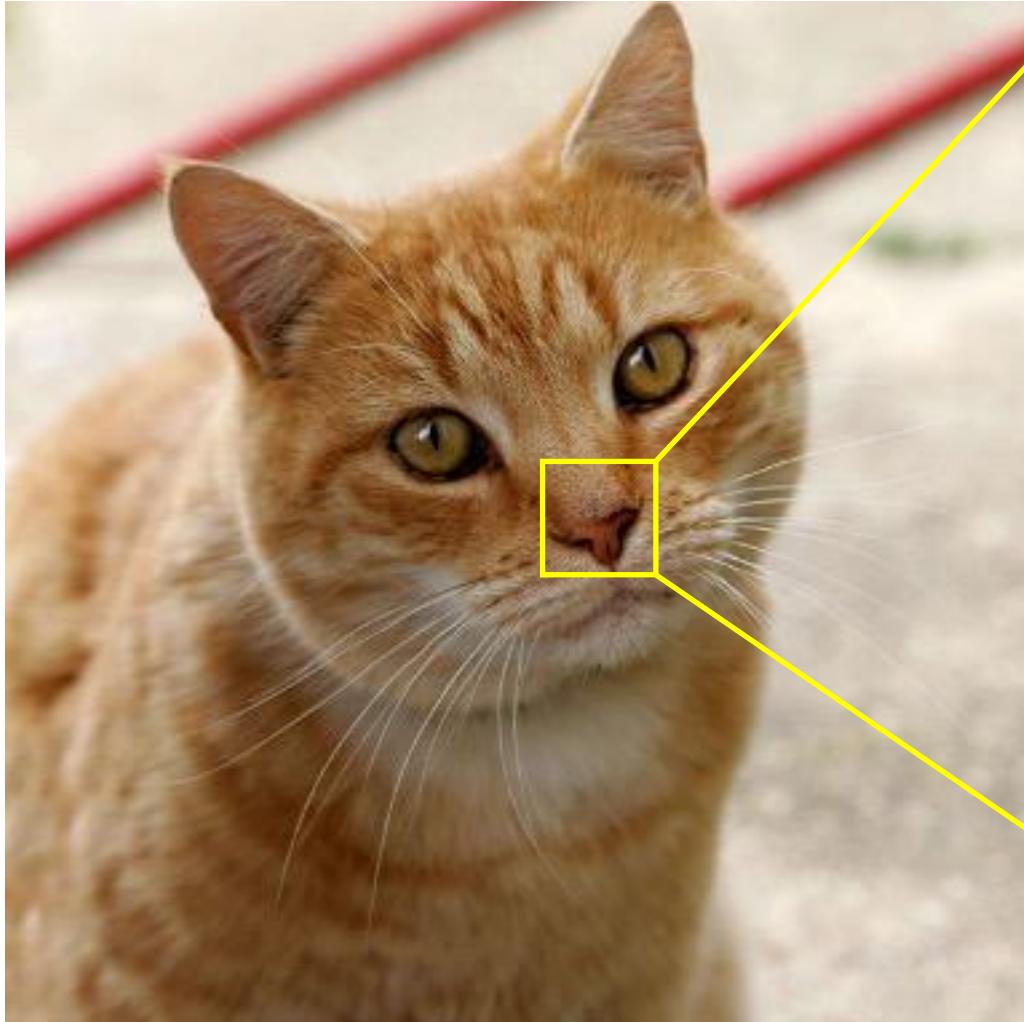


$$B_X = \frac{B_1 + B_2 + B_3 + B_4}{4}$$

+



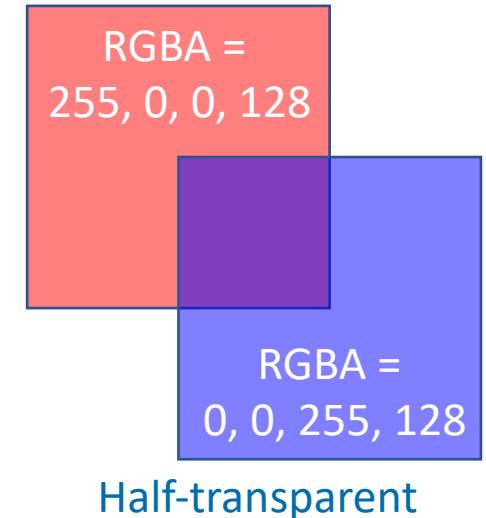
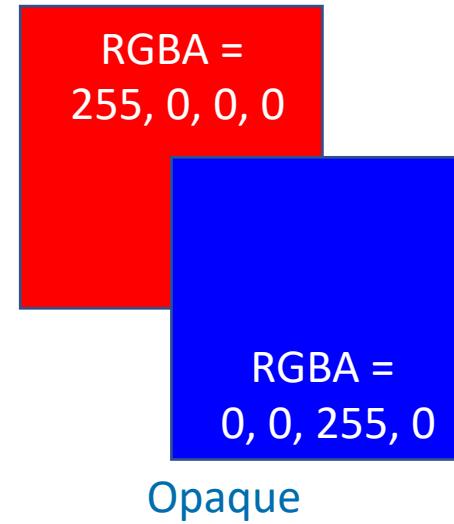
Colour representation in computer



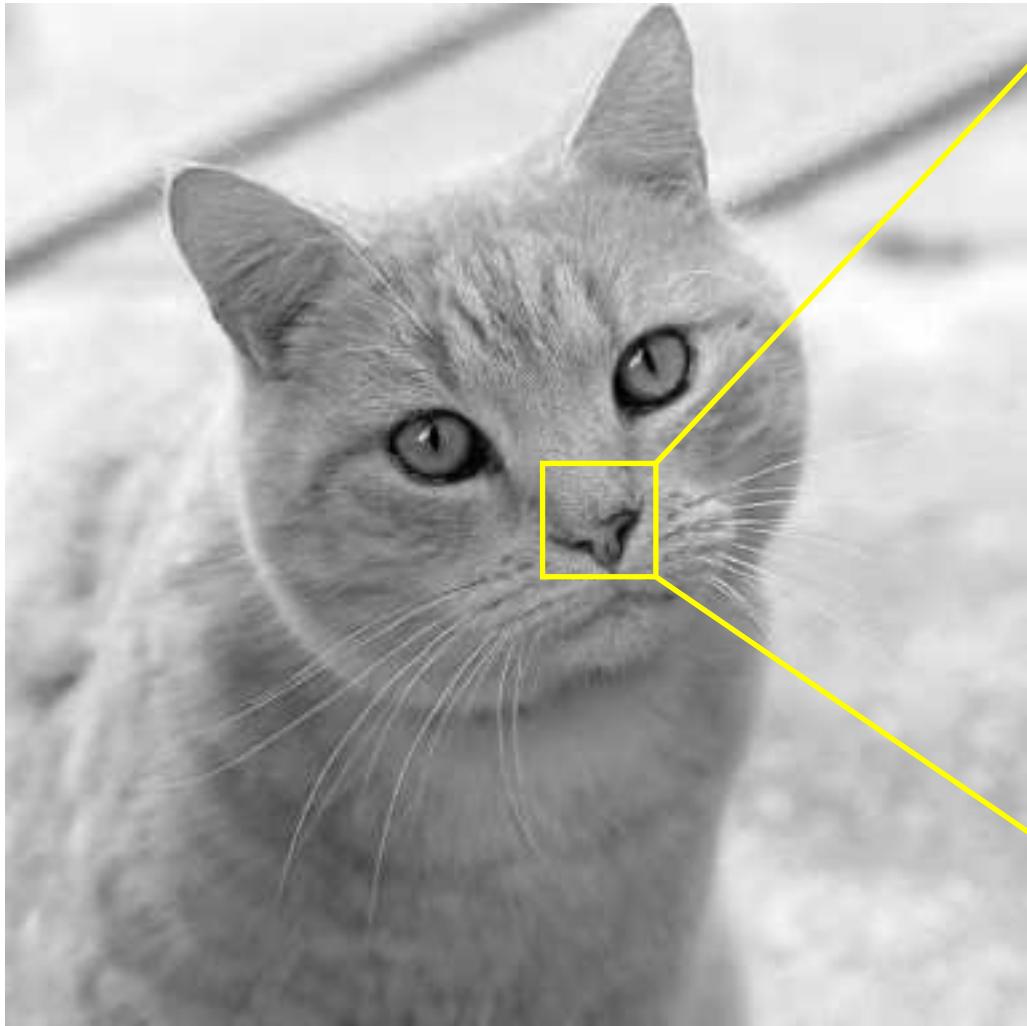
222	218	210	182	190	186	198	206		
218	190	182	198	194	198	205	206	89	
202	174	188	185	192	188	183	189	181	93
149	165	174	179	185	186	178	185	98	89
149	174	184	184	152	168	158	174	156	56
149	145	151	170	158	152	158	170	58	72
145	145	165	168	168	153	152	151	72	68
133	135	134	138	134	131	130	131	88	64
	32	66	88	80	80	56	52	66	60
	60	60	60	52	52	52	56	52	

Colour representation in computer

- RGB or RGBA
 - Red
 - Green
 - Blue
 - Alpha channel: transparency
- Grey
 - Greyscale intensity ($R = G = B$)



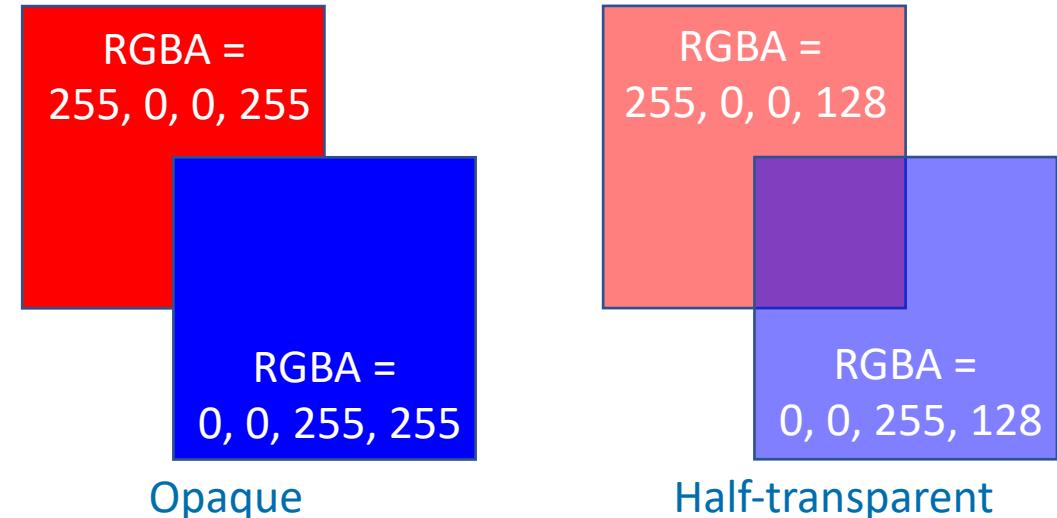
Grayscale representation



199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

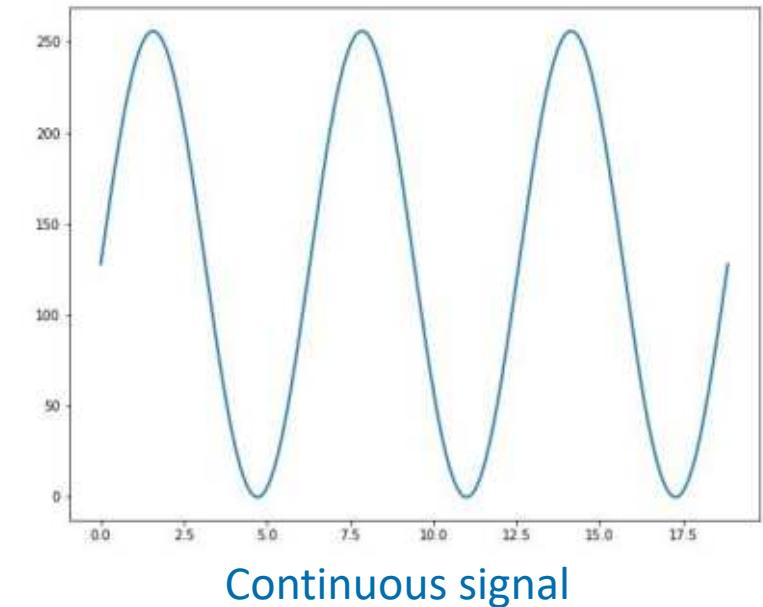
Colour representation in computer

- RGB or RGBA
 - Red
 - Green
 - Blue
 - Alpha channel: transparency
- Grey
 - Greyscale intensity ($R = G = B$)
- Representation of each channel
 - 8 bits, i.e. 0 to 255 (most common)
 - 16 bits (camera raw files)

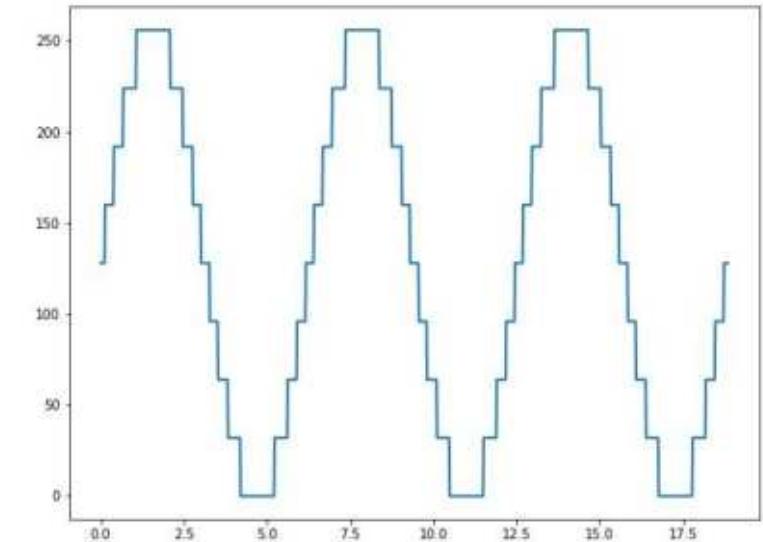


Quantisation

- Quantisation is the process that maps continuous signal to discrete signal.
- Numerical errors occur during quantisation, which depends on the number of bits we use.
- Representation of each colour channel
 - 8 bits: 0 to 255
 - 12 bits: 0 to 4096
 - 16 bits: 0 to 65535
 - The more bits, the less quantisation error.



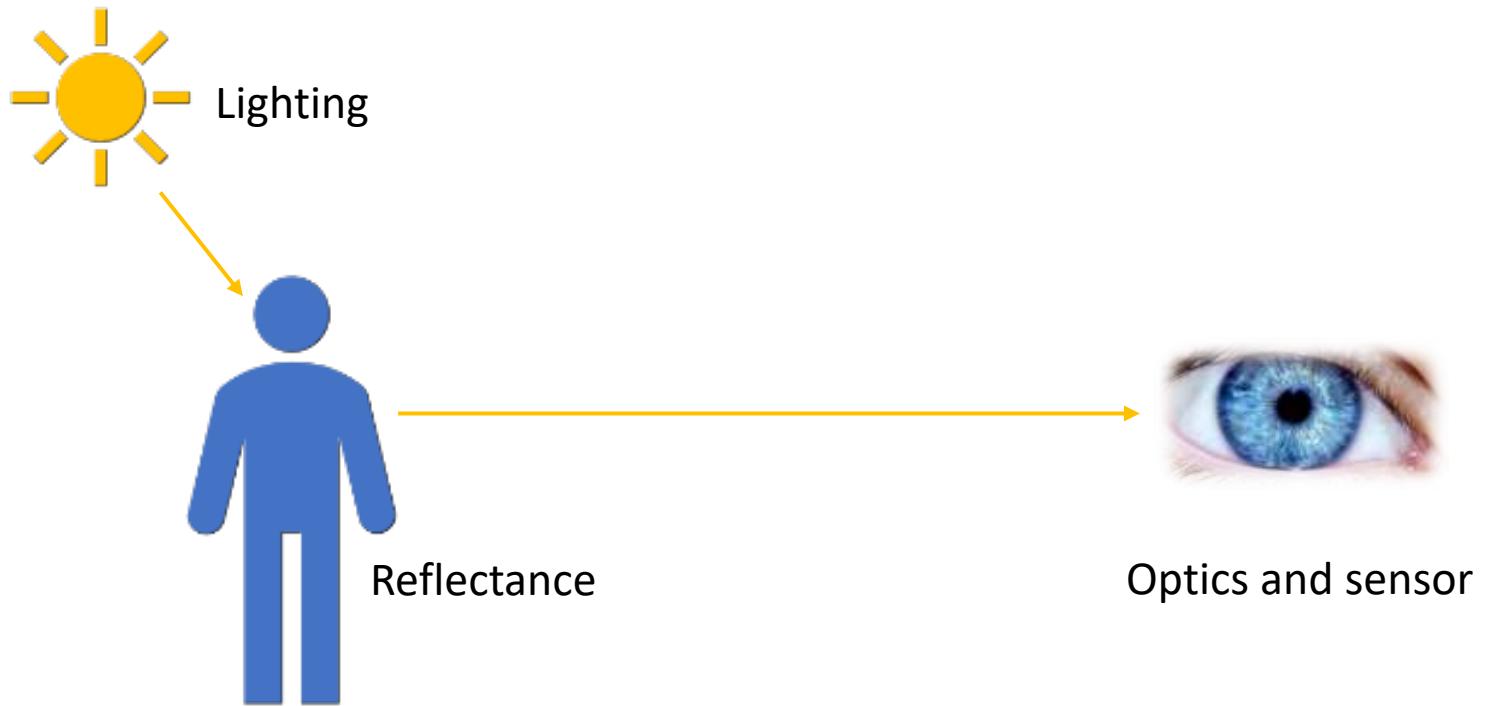
Continuous signal



Quantised to discrete signal

Image formation

- Light
- Reflectance
- Optics and sensors
- Colours



Graphics and vision

- Computer graphics
 Objects -> Image
- Computer vision
 Image -> Objects

Quotes from the early days

“We do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism.”

- Bui Tuong Phong, 1975

Vision is a process that produces from images of the external world a description that is useful to the viewer and not cluttered with irrelevant information.

- David Marr, Vision, 1979

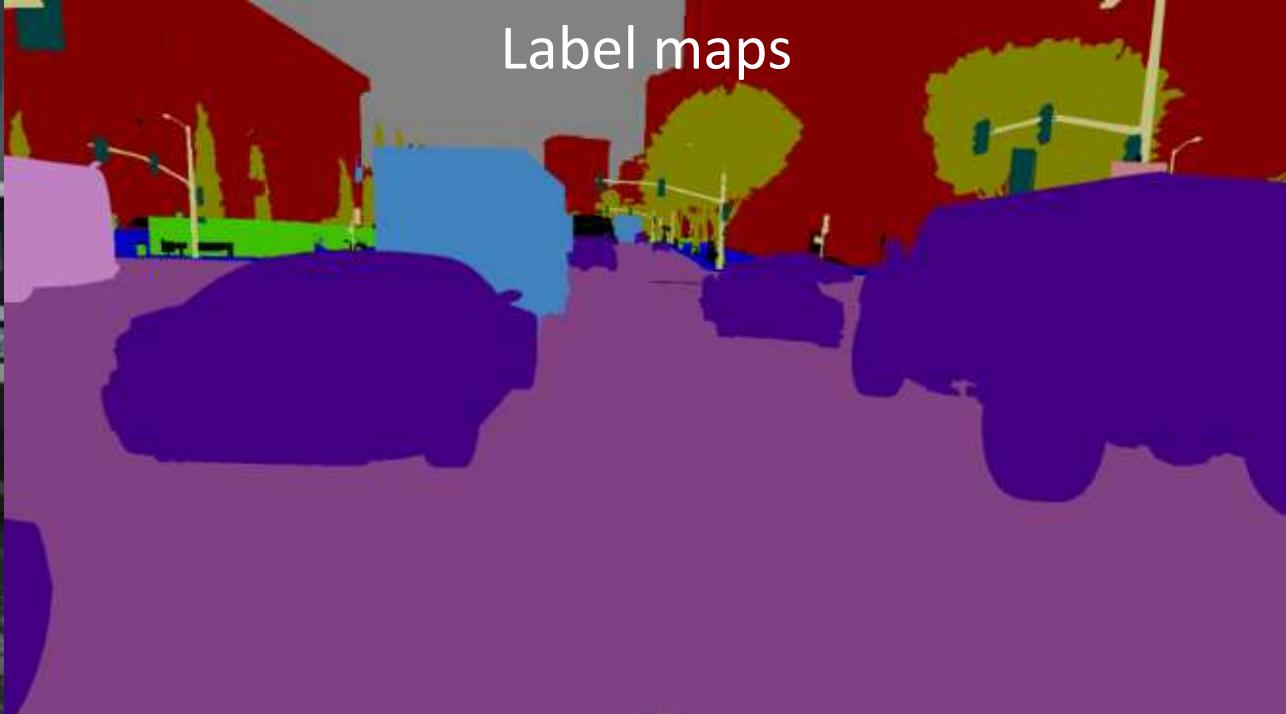
Graphics and vision

- Philosophically, computer vision is sometimes regarded as inverse computer graphics.
- Practically, computer graphics can help solving vision problems.
- For example, training a segmentation method may require both images and label maps as input.

Training images



Label maps



Computer graphics

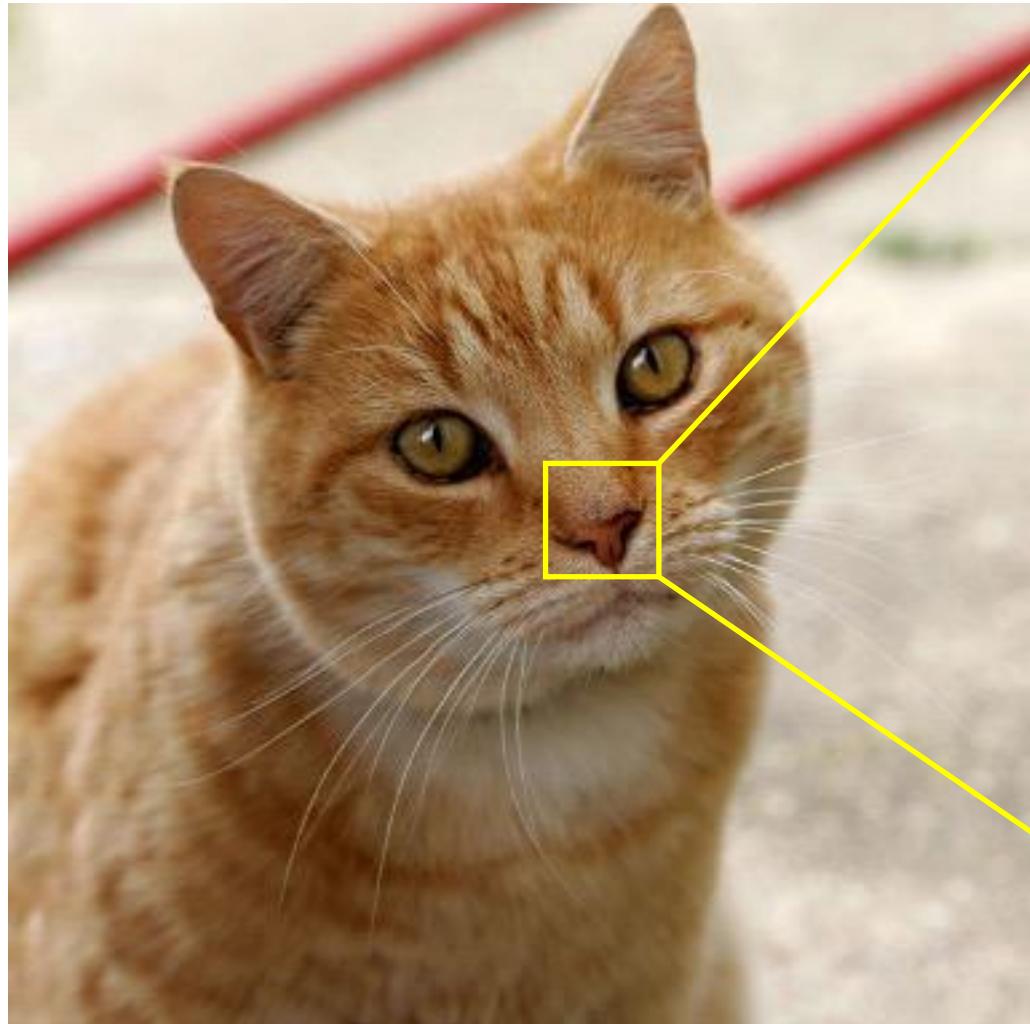


Grand Theft Auto V.

Computer graphics

- Computer graphics in photorealistic computer games can be used to synthesise data (images + label maps) for training computer vision algorithms.
- The synthetic data provides detailed semantic annotation, which is complementary to time-consuming manual annotations.

Image formation



222	218	210	182	190	186	198	206		
218	190	182	198	194	198	895	206	89	
202	174	188	185	194	188	180	189	185	93
149	165	174	179	185	185	176	185	98	89
149	174	184	184	152	168	158	174	156	93
149	145	151	170	158	152	158	170	56	72
145	145	165	168	168	153	152	151	42	68
133	135	184	183	183	161	160	161	88	64
	32	86	88	80	80	56	92	66	60
	60	60	60	52	52	52	56	52	

References

- Chapter 2: Image Formation. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Image Filtering I

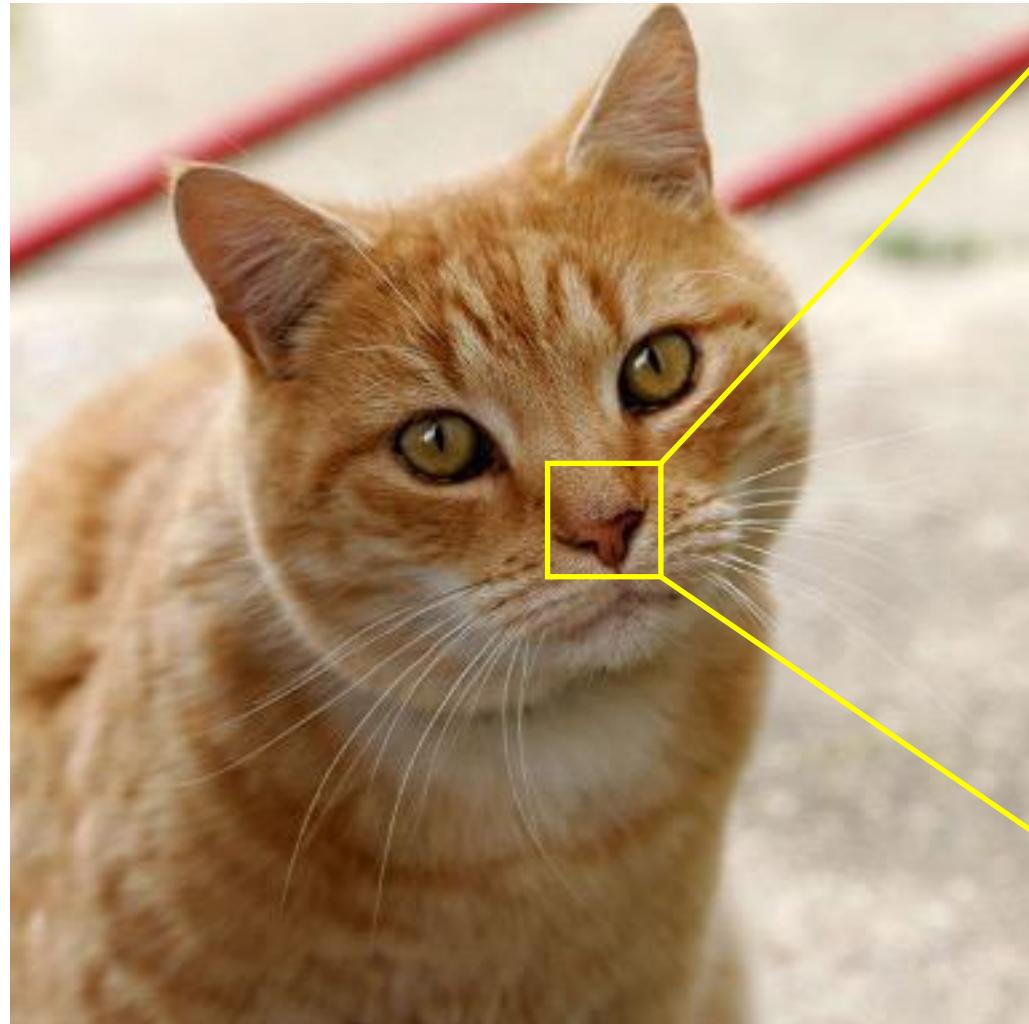
Dr Wenjia Bai

Department of Computing & Brain Sciences

What is image filtering?

- Image representation
- Image filters and applications

Image representation

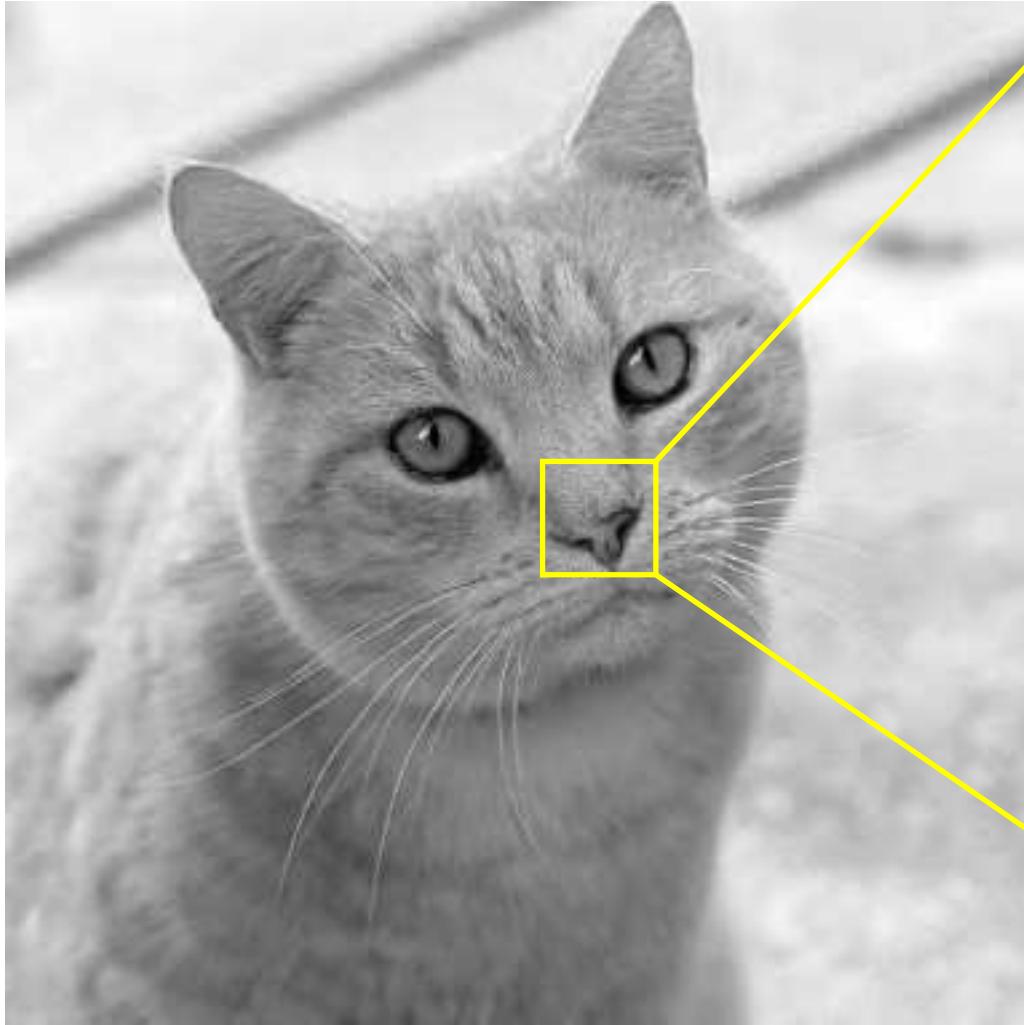


222	218	210	182	190	186	198	206		
218	190	182	198	194	198	895	206	89	
202	174	188	185	194	188	180	189	185	93
149	165	174	179	185	185	176	185	98	89
149	174	184	184	152	168	158	174	156	93
149	145	151	170	158	152	158	170	56	72
145	145	165	168	168	153	181	171	42	68
133	135	184	183	183	161	180	181	88	64
	32	86	88	80	80	56	92	66	60
	60	60	60	52	52	52	56	52	52

Colour channels



Grayscale image



199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Image filtering

- For most of the time, we will use grayscale images as examples for image filtering.
- For colour images, we can perform filtering for each of its colour channel, e.g. RGB channels.

Example of image filtering



Input



Output

Moving average filter

- We start from moving average filter, which is commonly used for both 1D signal (time series) and 2D signal (images) processing.
- It moves a window across the signal and calculates the average value within the window.



Moving average (MA) for stock market analysis.

<https://stockcharts.com>

Moving average filter

Cases in United Kingdom ▾

People tested positive

Daily	Total
45,533	3,164,051

Cases by specimen date

UK total By nation

Number of people with at least one positive COVID-19 test result (either lab-reported or lateral flow device), by specimen date. Individuals tested positive more than once are only counted once, on the date of their first positive test. Data for the period ending 5 days before the date when the website was last updated with data for the selected area, highlighted in grey, is incomplete.

[Daily](#) [Cumulative](#) [Data](#) [About](#)



<https://coronavirus.data.gov.uk/details/cases>

Moving average (3x3 window)

Let us try this
filter kernel.

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \div 9$$

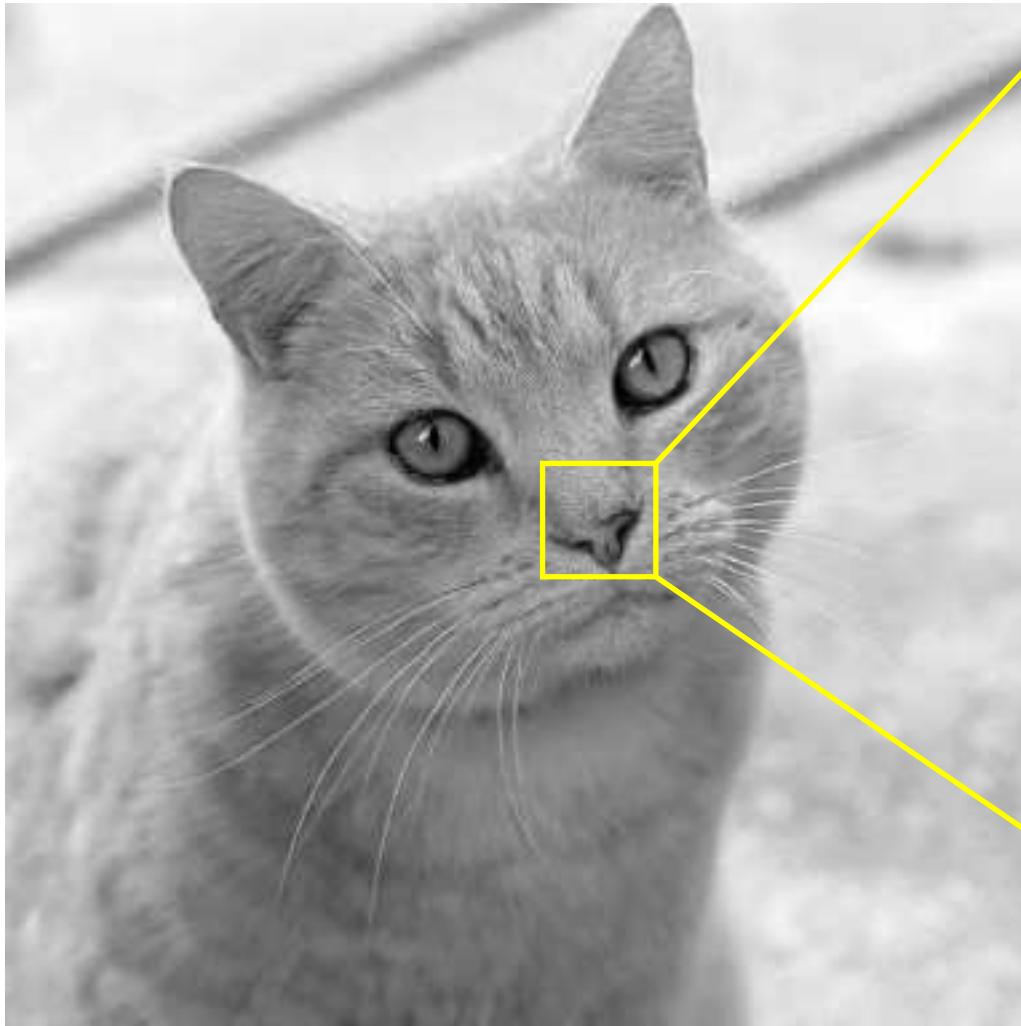


Input



Output

Grayscale image



199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

1/9	1/9	1/9	111	110	123	130	130
1/9	1/9	1/9	111	113	120	126	125
1/9	1/9	1/9	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

199	1/9	1/9	1/9	1/9	110	123	130	130
189	1/9	1/9	1/9	1/9	113	120	126	125
130	1/9	1/9	1/9	1/9	113	113	114	120
85	100	96	104	108	107	101	94	
85	95	98	96	100	103	100	96	
79	94	87	77	69	70	87	84	
77	80	72	71	60	52	59	64	
68	67	63	58	53	51	54	52	

199	192	179	179	179	123	130	130
189	149	109	179	179	120	126	125
130	100	199	109	179	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

	147	126	114				

199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	17/9	18/9	18/9
77	80	72	71	60	15/9	15/9	15/9
68	67	63	58	53	15/9	15/9	15/9

	147	126	114	114	118	122	
	117	108	107	111	113	113	
	99	99	102	106	107	105	
	91	94	93	93	94	94	
	85	86	81	78	78	79	
	76	74	68	62	62	64	

Padding

- The output image is smaller than the input image.
- How to deal with the boundary pixels?
- Padding
 - By constant value (e.g. 0 or the boundary pixel value)
 - By mirroring values
 - ...

Padding

1 ⁰ 1/9	1 ⁰ 1/9	1 ⁰ 1/9						
1 ⁰ 1/9	1 ⁰ 1/9	1 ⁰ 1/9	158	111	110	123	130	130
1 ⁰ 1/9	1 ⁰ 1/9	1 ⁰ 1/9	108	111	113	120	126	125
	130	100	98	108	113	113	114	120
	85	100	96	104	108	107	101	94
	85	95	98	96	100	103	100	96
	79	94	87	77	69	70	87	84
	77	80	72	71	60	52	59	64
	68	67	63	58	53	51	54	52

81								
	147	126	114	114	118	122		
	117	108	107	111	113	113		
	99	99	102	106	107	105		
	91	94	93	93	94	94		
	85	86	81	78	78	79		
	76	74	68	62	62	64		

199	192	158	111	110	123	130	130	
189	149	108	111	113	120	126	125	
130	100	98	108	113	113	114	120	
85	100	96	104	108	107	101	94	
85	95	98	96	100	103	100	96	
79	94	87	77	69	70	87	84	
77	80	72	71	60	52	15/9	16/9	1/9
68	67	63	58	53	51	15/9	16/9	1/9
						1/9	1/9	1/9

81	111	92	79	76	80	84	57	
107	147	126	114	114	118	122	83	
84	117	108	107	111	113	113	76	
66	99	99	102	106	107	105	69	
60	91	94	93	93	94	94	62	
57	85	86	81	78	78	79	54	
52	76	74	68	62	62	64	44	
32	47	46	42	38	37	37	25	

Moving average (3x3 window)

Let us try this
filter kernel.

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \div 9$$



Input



Output

Moving average (7x7 window)

kernel

÷ 49



Input



Output

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

Computational complexity

- Image size: $N \times N$
- Kernel size: $K \times K$
- What is the computational complexity?
 - At each pixel, K^2 multiplications, then K^2-1 summations
 - Do this for N^2 pixels
 - That is N^2K^2 multiplications and $N^2(K^2-1)$ summations
 - The complexity is $O(N^2K^2)$
- Can we accelerate this?

Separable filter

- If a big filter can be separated as the consecutive operation of two small filters, then we can first filter the input image with small filter 1, then with small filter 2.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

0	0	0
1/3	1/3	1/3
0	0	0

*

0	1/3	0
0	1/3	0
0	1/3	0

Average in a
2D window

Average convolution
across row

Average
across column

Separable filter

- If a big filter can be separated as the consecutive operation of two small filters, then we can first filter the input image with small filter 1, then with small filter 2.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

1/3	1/3	1/3
-----	-----	-----

*

1/3
1/3
1/3

Average in a
2D window

Average convolution
across row

Average
across column

1/3	1/3	1/3	158	111	110	123	130	130
189	149	108	111	113	120	126	125	
130	100	98	108	113	113	114	120	
85	100	96	104	108	107	101	94	
85	95	98	96	100	103	100	96	
79	94	87	77	69	70	87	84	
77	80	72	71	60	52	59	64	
68	67	63	58	53	51	54	52	

130								

199	192	158	111	110	123	130	130	
189	149	108	111	113	120	126	125	
130	100	98	108	113	113	114	120	
85	100	96	104	108	107	101	94	
85	95	98	96	100	103	100	96	
79	94	87	77	69	70	87	84	
77	80	72	71	60	52	59	64	
68	67	63	58	53	51	1/3	1/3	1/3

130	183	154	126	115	121	128	87	
113	149	123	111	115	120	124	84	
77	109	102	106	111	113	116	78	
62	94	100	103	106	105	101	65	
60	93	96	98	100	101	100	65	
58	87	86	78	72	75	80	57	
52	76	74	68	61	57	58	41	
45	66	63	58	54	53	52	35	

After the first filter

1/3								
1/3	183	154	126	115	121	128	87	
1/3	149	123	111	115	120	124	84	
77	109	102	106	111	113	116	78	
62	94	100	103	106	105	101	65	
60	93	96	98	100	101	100	65	
58	87	86	78	72	75	80	57	
52	76	74	68	61	57	58	41	
45	66	63	58	54	53	52	35	

After the first filter

81	111	92	79	76	80	84	57
107	147	126	114	114	118	122	83
84	117	108	107	111	113	113	76
66	99	99	102	106	107	105	69
60	91	94	93	93	94	94	62
57	85	86	81	78	78	79	54
52	76	74	68	62	62	64	44
32	47	46	42	38	37	37	25

After the second filter

Moving average (separable filter)

$$\begin{array}{|c|c|c|} \hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

$$* \begin{array}{|c|c|c|} \hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$



Input



Output

Computational complexity for separable filtering

- Image size: $N \times N$
- Two kernels: first one $1 \times K$, second one $K \times 1$
- What is the computational complexity?
 - At each pixel, K multiplications, then $K - 1$ summations
 - Do this for N^2 pixels
 - Do this twice for two kernels
 - That is $2N^2K$ multiplications and $2N^2(K - 1)$ summations
 - The complexity is $O(N^2K)$, versus the original complexity $O(N^2K^2)$
- It makes a difference if K is large.

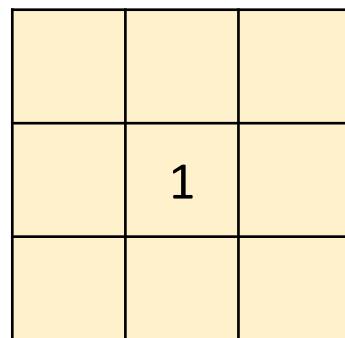
What does a moving average filter do?

- It removes high-frequency signal (noise or sharpness).
- Result in a smooth but blurry image.

Types of image filters

- Identity filter
- Low-pass or smoothing filters
 - Moving average filter
 - Gaussian filter
- High-pass or sharpening filters
- Denoising filters
 - Median filter
- ...

Identity filter



Filter kernel

Identity filter



Input



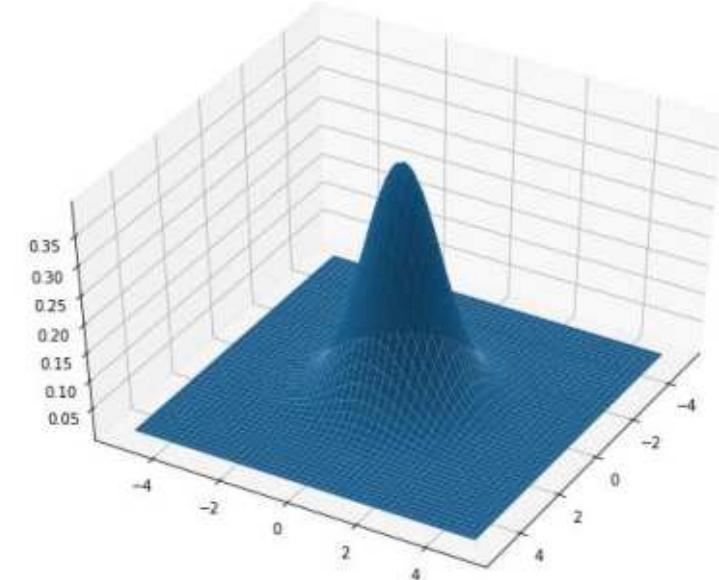
Output

Gaussian filter

- Kernel: 2D Gaussian distribution

$$h(i, j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- i, j denote the offset from the centre of the window.
- σ controls the shape of the filter.
- Its support is infinite, but we may ignore small values outside $[-k\sigma, k\sigma]$, e.g. $k = 3$ or 4 .



Gaussian distribution

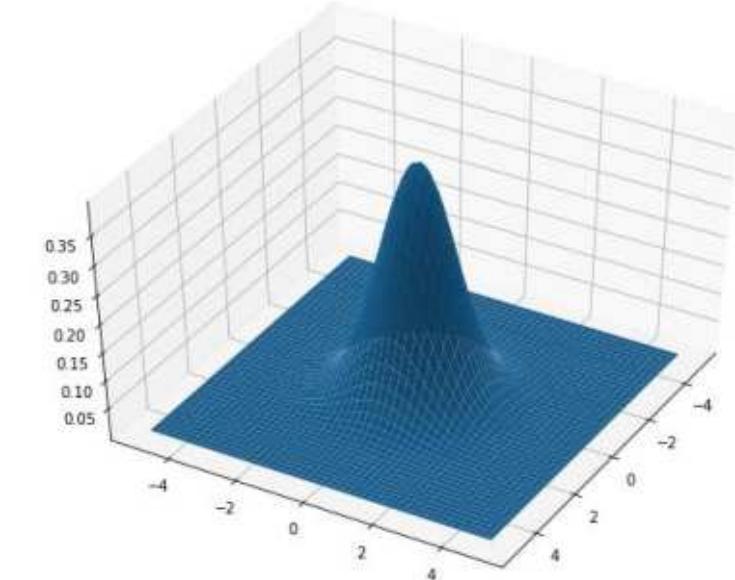
Gaussian filter

- The 2D Gaussian filter is a separable filter, equivalent to two 1D Gaussian filters with the same σ , one along x-axis and the other along y-axis.

$$h(i, j) = h_x(i) * h_y(j)$$

$$h_x(i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}}$$

$$h_y(j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{j^2}{2\sigma^2}}$$



Gaussian distribution

Gaussian filter



Input



Output

Low-pass vs high-pass filters

- The moving average filter and Gaussian filter smooth or blur the image, keeping the low-frequency signals.
- They are called low-pass filters or smoothing filters.
- There are some filters that sharpen the image and highlight the high-frequency signals.
- They are called high-pass filters or sharpening filters.

High-pass filter

Design 1

1		

Identity

	1	

High-frequency

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

-1/9	-1/9	-1/9
-1/9	1.9	-1/9
-1/9	-1/9	-1/9

Design 2

1		

Identity

-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

High-frequency

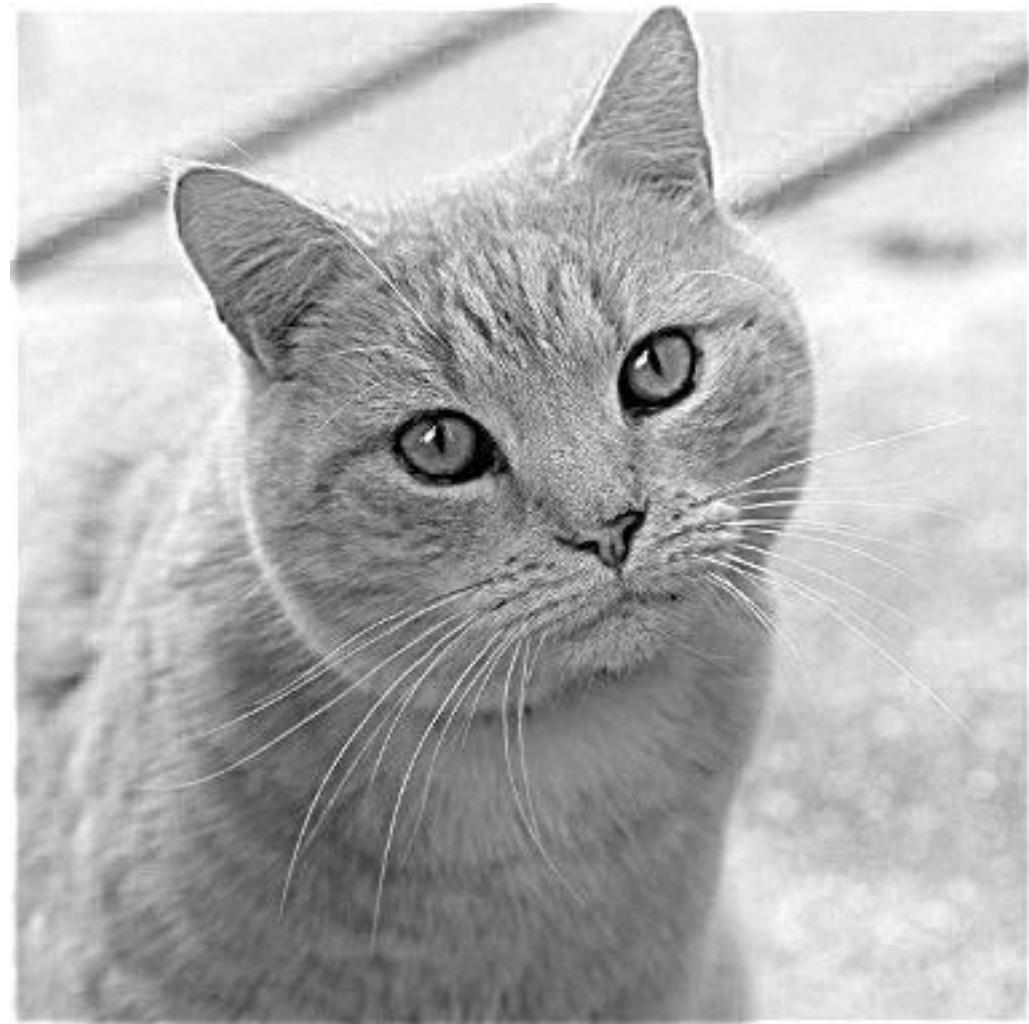
-1/8	-1/8	-1/8
-1/8	2	-1/8
-1/8	-1/8	-1/8

Design ...

High-pass filter



Input



Output

Median filter

- A non-linear filter
- Often used for denoising
- How to perform median filtering?
 - Move the sliding window
 - Replace the centre pixel using the median value in the window

199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

	149						

199	192	158	111	110	123	130	130	
189	149	108	111	113	120	126	125	
130	100	98	108	113	113	114	120	
85	100	96	104	108	107	101	94	
85	95	98	96	100	103	100	96	
79	94	87	77	69	70	87	84	
77	80	72	71	60	52	59	64	
68	67	63	58	53	51	54	52	

192	189	149	111	111	123	126	130	
189	149	111	111	113	114	123	125	
130	100	104	108	111	113	114	120	
95	98	98	100	107	107	103	100	
85	94	96	96	100	100	96	94	
80	85	87	77	71	70	84	84	
77	77	72	69	60	59	59	64	
68	68	67	60	53	53	52	54	

Median filter



Corrupted by salt and pepper noise



Denoised image

Denoising filter

- Apart from median filter, there are other more complex denoising filters.
 - Non-local means
 - Block-matching and 3D filtering (BM3D)
 - ...

Who use these filters?

- Photographers and designers
- Denoising used by cameras
- Youtubers or maybe the person you talk to on Zoom



Photoshop



Smartphone camera



Zoom touch up filter

Image filtering

- Low-pass filters
 - Moving average filter
 - Gaussian filter
 - Separable filtering
- High-pass filters
- Denoising filters
 - Median filter

References

- Section 3.2: Linear filtering; Section 3.3.1: Non-linear filtering. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Coursework 1

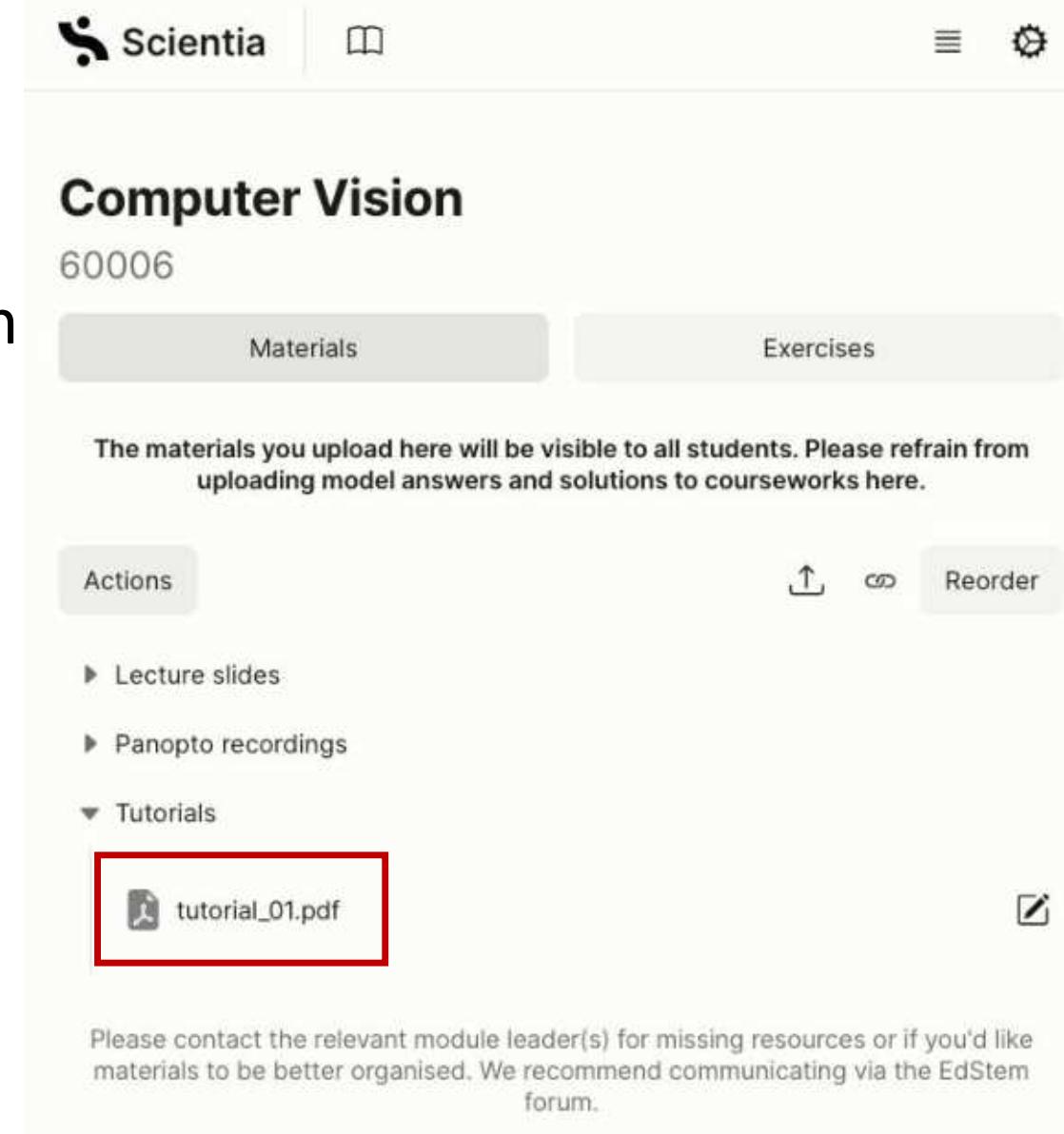
- Image filtering and edge detection
- It is on GitHub: https://github.com/ImperialCollegeLondon/Computer_Vision_2023
- Work on coursework_01.ipynb using Jupyter-Lab

Coursework 1

- What to submit?
 - Export the Jupyter notebook as a **PDF** file, which contains code, results and answers.
- How to submit?
 - Upload to Scientia
- When to submit?
 - Due in two weeks time.
- There will be a lab session next week at the Computer Lab on Level 2.
 - We will provide general technical help.
 - However, we will not provide model answers.

Tutorial 01

- On Scientia
- You can either work in the tutorial session or at home.
- Solution will be released next week.



The screenshot shows the Scientia platform interface for the "Computer Vision" course (60006). The top navigation bar includes the Scientia logo, a search icon, and a settings gear icon. The course title "Computer Vision" and code "60006" are displayed prominently. Below the title, there are two tabs: "Materials" (which is selected and highlighted in grey) and "Exercises". A note below the tabs states: "The materials you upload here will be visible to all students. Please refrain from uploading model answers and solutions to courseworks here." The "Actions" section contains buttons for "Upload" and "Reorder". The "Materials" list includes sections for "Lecture slides", "Panopto recordings", and "Tutorials". A file named "tutorial_01.pdf" is listed under the "Tutorials" section, and it is highlighted with a red box. At the bottom, a note encourages users to contact module leaders for missing resources or better organization, and it recommends using the EdStem forum for communication.

Scientia

Computer Vision

60006

Materials Exercises

The materials you upload here will be visible to all students. Please refrain from uploading model answers and solutions to courseworks here.

Actions

Upload Reorder

▶ Lecture slides

▶ Panopto recordings

▼ Tutorials

 tutorial_01.pdf



Please contact the relevant module leader(s) for missing resources or if you'd like materials to be better organised. We recommend communicating via the EdStem forum.

Tutorial 1: Image Formation, Image Filtering

1. Let (x, y) be the coordinate of a point in an image. Its homogeneous coordinate $(x, y, 1)$ is often used in computer vision or graphics, for example, transforming an image by scaling, translation, rotation etc. The transformation can be described by the following equation,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = A \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

where $(x, y, 1)$ and $(x', y', 1)$ denote the coordinate before and after transformation, A is a 3×3 transformation matrix. For the following examples of A , describe what kind of transformation it performs.

1.1 Example 1

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

1.2 Example 2

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2. For a RGB image, at each pixel, there are intensity values for three channels (R: red; G: green; B: blue). For example, $[255, 0, 0]$ represents pure red. $[0, 255, 0]$ represents pure green. $[0, 0, 255]$ represents pure blue. Each channel is represented by an integer between 0 and 255, i.e. an 8-bit unsigned char.

2.1 For a RGB image of size 1280x960 pixels, without image compression, how many bytes are needed for storing the image?

2.2 There are algorithms to convert a RGB image into a grayscale image, where each pixel only has one intensity value, which represents the brightness. For example, one algorithm is recommended by ITU-R Recommendation BT.601, which is formulated by the following equation,

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

Work out the grayscale values pure red, pure green and pure blue respectively convert to.

3. Suppose that we have an image that is corrupted by Gaussian white noise $Y = I + n$, where I denotes the clean image, $n \sim N(0, \sigma^2)$ denotes the Gaussian white noise and Y denotes the corrupted image. Perform image filtering using a low-pass filter is one approach for denoising, but it may also result in loss of fine details. Another approach is to denoise to take a lot of images of the same object and combine them. For example, you can take a lot of pictures of the moon in the sky from the same angle. Each image is described by $Y_i = I + n_i$ ($i = 1, 2, \dots, N$), where n_i is one independent realisation of the noise. Then you can take the average, $Y = \frac{1}{N} \sum_{i=1}^N Y_i$. Derive the mean and variance of the noise in the combined image Y .

4. Median filtering is often used for image denoising. Suppose a uniform region of an image is corrupted by salt and pepper noise (salt: random white pixels; pepper: random black pixels). Show that provided less than half of the pixels in a neighbourhood are corrupted, a 3×3 median filter will almost perfectly restore the uniform region.

5. Convolution is a mathematical operation to describe filtering, where a filter $h[n], n = 0, 1, 2, \dots$ is applied to an input signal $f[n], n = 0, 1, 2, \dots$ and generates an output signal $o[n], n = 0, 1, 2, \dots$. Convolution is denoted by $*$ and defined as,

$$o[n] = f * h = \sum_{m=-\infty}^{\infty} f[m]h[n-m]$$

Prove the associativity of convolution, i.e.

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

where g denotes a filter.

Image Filtering II

Dr Wenjia Bai

Department of Computing & Brain Sciences

How to describe filtering mathematically?

- Signal processing
- Filtering as convolution

Recap on image filtering

1/9	1/9	1/9	111	110	123	130	130
1/9	1/9	1/9	111	113	120	126	125
1/9	1/9	1/9	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Recap on image filtering

199	1/9	1/9	1/9	110	123	130	130
189	1/9	1/9	1/9	113	120	126	125
130	1/9	1/9	1/9	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Recap on image filtering

199	192	179	179	179	123	130	130
189	149	109	179	179	120	126	125
130	100	199	109	179	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Recap on image filtering

199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	17/9	18/9	18/9
77	80	72	71	60	15/9	15/9	15/9
68	67	63	58	53	15/9	15/9	15/9

	147	126	114	114	118	122	
	117	108	107	111	113	113	
	99	99	102	106	107	105	
	91	94	93	93	94	94	
	85	86	81	78	78	79	
	76	74	68	62	62	64	

How to describe filtering mathematically?

- For simplicity, we start from 1D case.

kernel

1/3	1/3	1/3
-----	-----	-----

189	149	108	111	113	120	126	125
-----	-----	-----	-----	-----	-----	-----	-----

input: f

	123						
--	-----	--	--	--	--	--	--

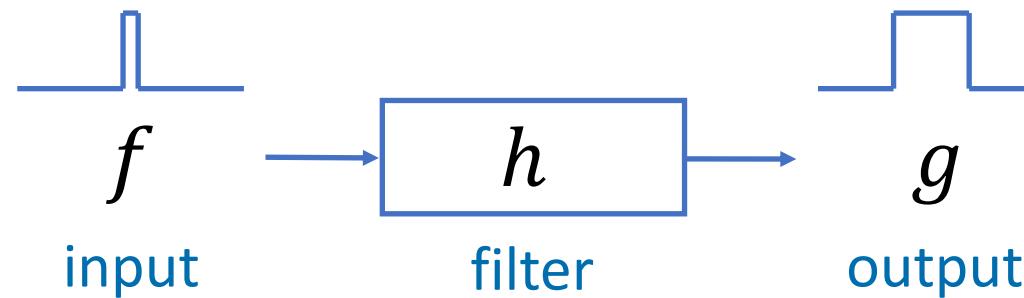
output: g

- It can be written as a weighted average in a window,

$$g[n] = \frac{1}{3} \cdot f[n-1] + \frac{1}{3} \cdot f[n] + \frac{1}{3} \cdot f[n+1]$$

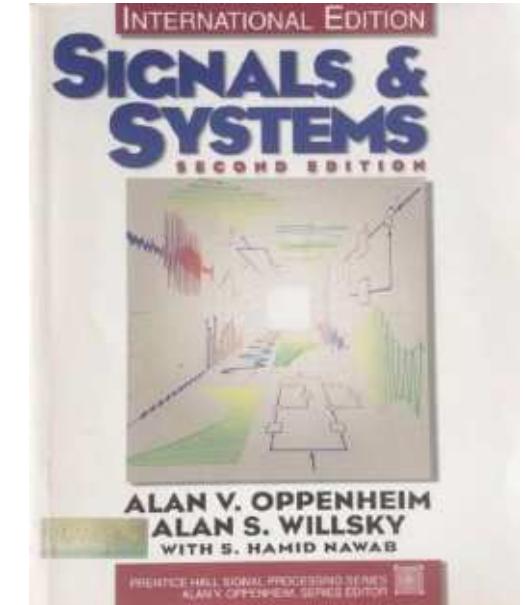
Filtering

- A filter h takes an input signal f , processes it and generates output signal g .



Filtering in signal processing

- Filtering is intensively studied in signal processing.
- A filter (or filtering) is a device (or process) that removes unwanted components or features from a signal.
- In other words, it keeps or enhances wanted features.



Signal processing

Filter

- A filter can be an electrical hardware device.



Filter

- A filter can also be a digital software.



Filtering

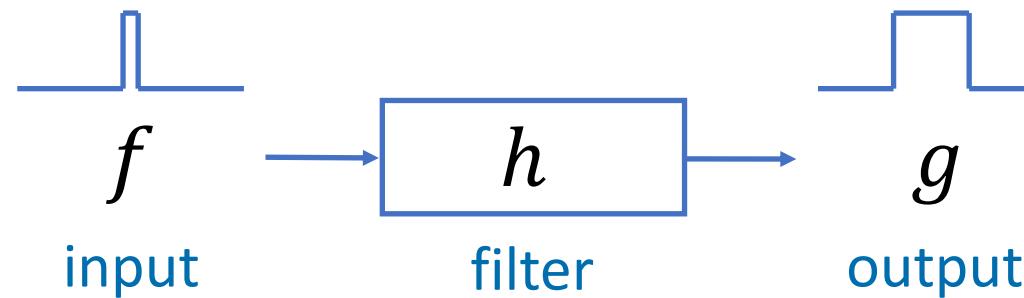
- You can apply filtering to 1D, 2D, 3D or even higher-dimensional signals.
 - Audio, speech
 - Natural images
 - Medical images
 - Radar images
 - Satellite images
 - ...
- You can smooth, sharpen, denoise images.
- You can add special effects to music.

A lot of filtering in musical performance



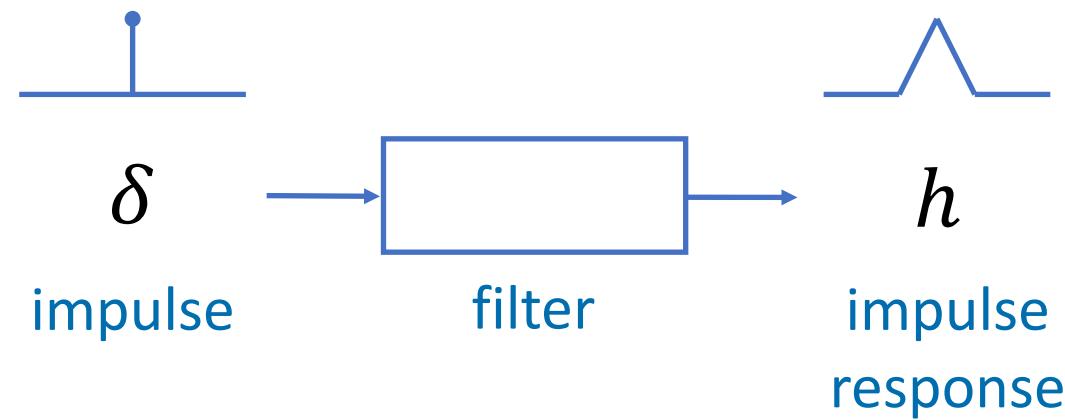
How to mathematically describe a filter?

- How do we describe h here?



Impulse response

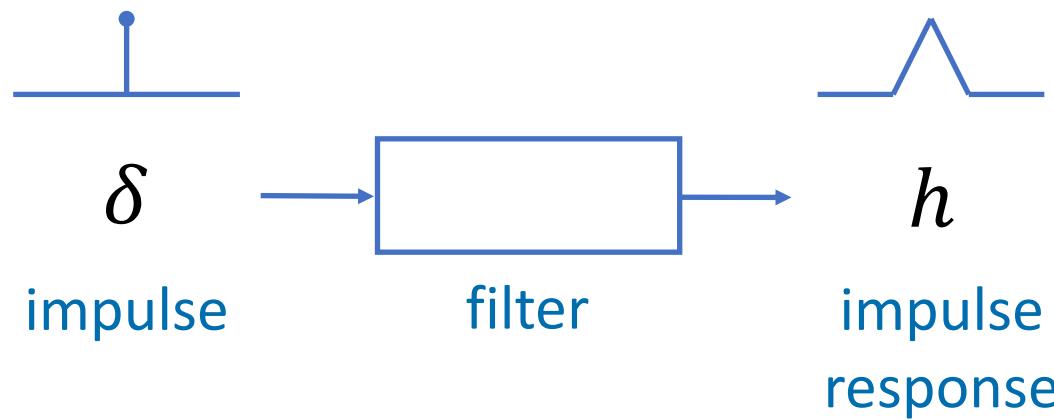
- To mathematically describe a filter, we introduce the concept of impulse response.
- The impulse response is the output of a filter when the input is an impulse signal.



Impulse response

- For continuous signal, an impulse is a Dirac delta function $\delta(x)$,

$$\delta(x) = \begin{cases} +\infty, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$$
$$\int_{-\infty}^{+\infty} \delta(x) dx = 1$$



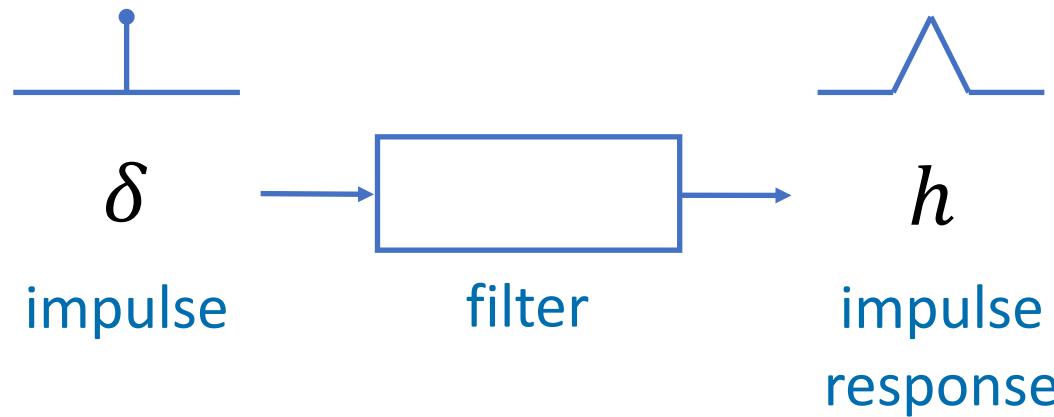
Impulse response

- For continuous signal, an impulse is a Dirac delta function $\delta(x)$,

$$\delta(x) = \begin{cases} +\infty, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$$
$$\int_{-\infty}^{+\infty} \delta(x) dx = 1$$

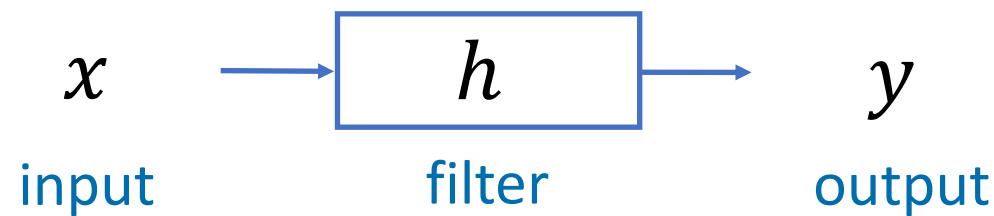
- For discrete signal, an impulse is a Kronecker delta function $\delta[i]$,

$$\delta[i] = \begin{cases} 1, & \text{if } i = 0 \\ 0, & \text{otherwise} \end{cases}$$

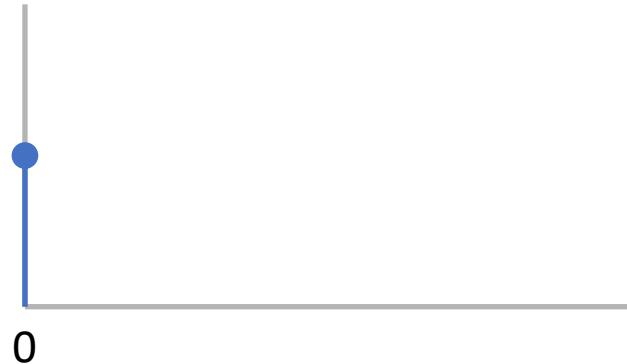


Impulse response

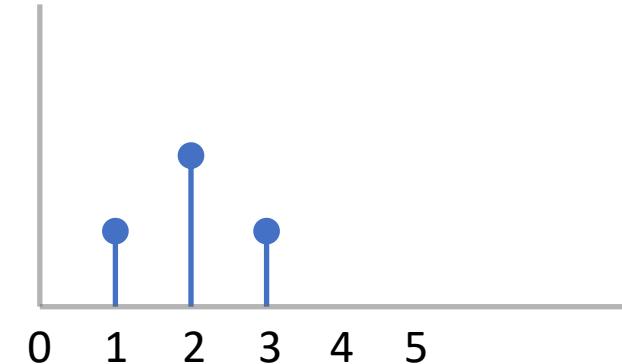
- The impulse response h completely characterises a *linear time-invariant* filter. As long as we know h , we can calculate the output signal y , given any input x .
- In signal processing, we often denote a filter by its impulse response function h .



Impulse response

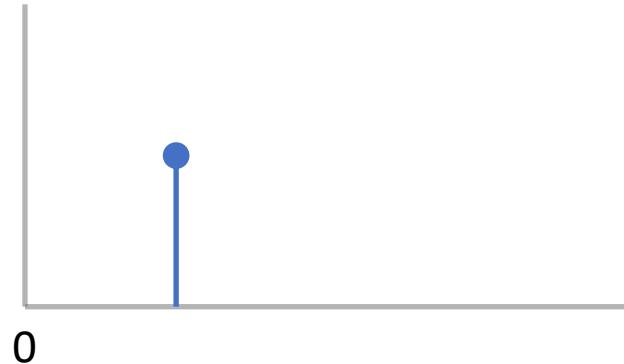


Input $f[n] = \delta[n]$

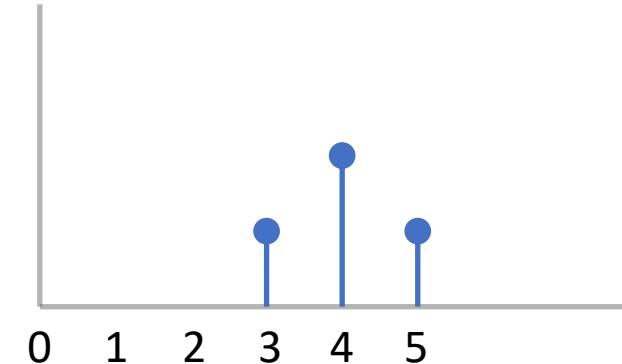


Output $g[n] = h[n]$

Time-invariant system

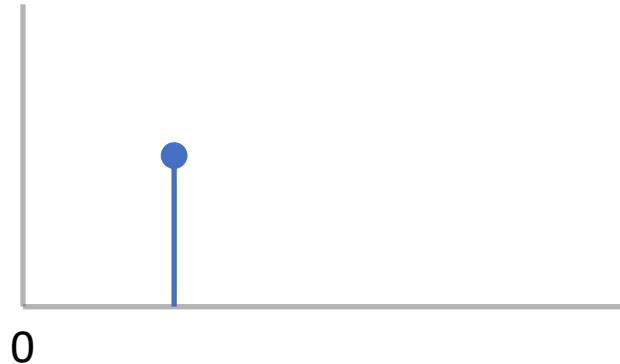


$$\text{Input } f[n] = \delta[n - 2]$$

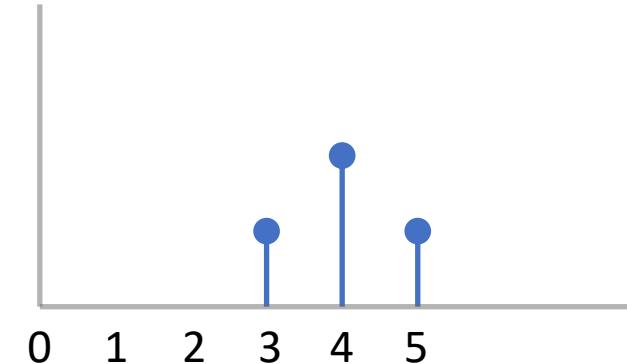


$$\text{Output } g[n] = h[n - 2]$$

Time-invariant system



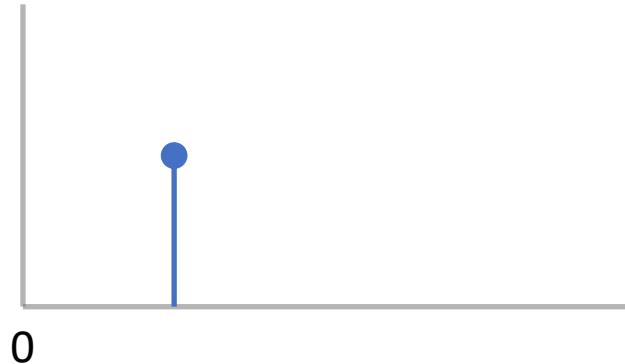
Input $f[n] = \delta[n - 2]$



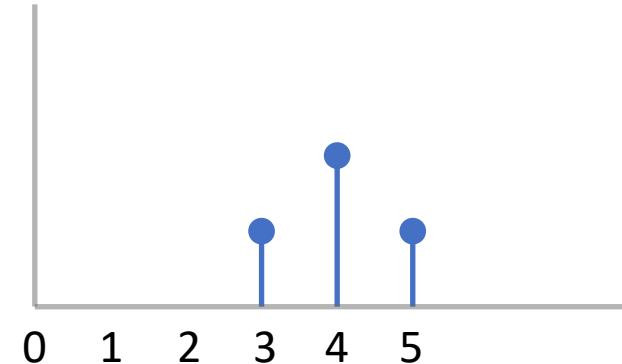
Output $g[n] = h[n - 2]$

- If a filter is a time-invariant system, when we shift the input by time step k , the output will also shift by k .
- For example,
 - $g[n] = 10 \cdot f[n]$ is time-invariant, which amplifies the input by a constant.
 - $g[n] = n \cdot f[n]$ is not time-invariant, whose output depends on time step n .

Time-invariant system



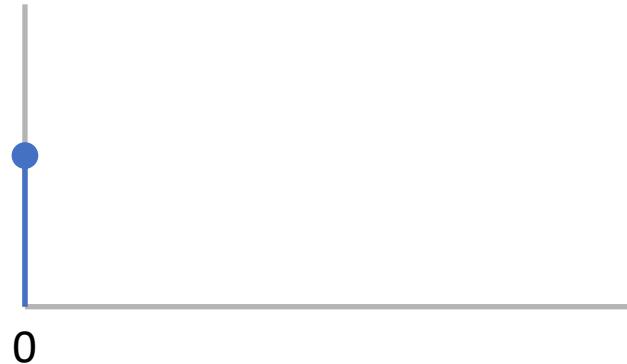
Input $f[n] = \delta[n - 2]$



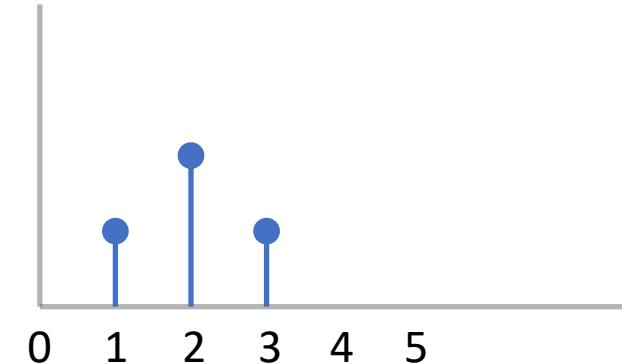
Output $g[n] = h[n - 2]$

- If a filter is a time-invariant system, when we shift the input by time step k , the output will also shift by k .
- Because when we shift the input f by k , the output
 - $10 \cdot f[n - k]$ is the same as shifting $g[n]$ by k , $g[n - k] = 10 \cdot f[n - k]$.
 - $n \cdot f[n - k]$ is not a simple shift of $g[n]$ by k , $g[n - k] = (n - k) \cdot f[n - k]$.

Impulse response



Input $f[n] = \delta[n]$

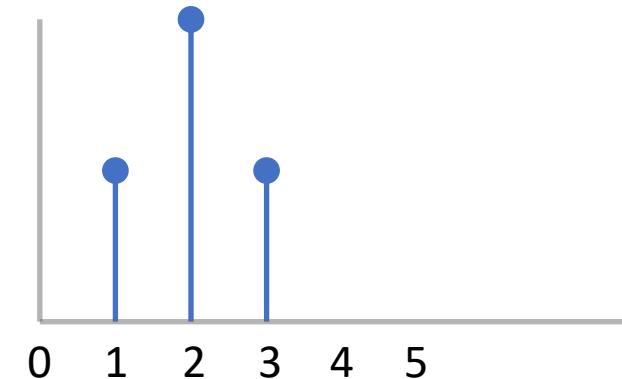


Output $g[n] = h[n]$

Linear system

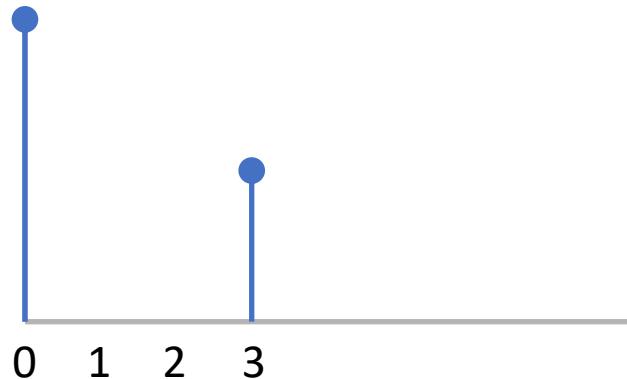


$$\text{Input } f[n] = 2 \cdot \delta[n]$$

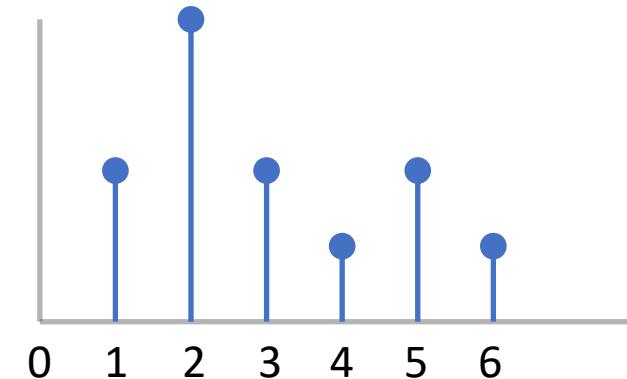


$$\text{Output } g[n] = 2 \cdot h[n]$$

Linear system

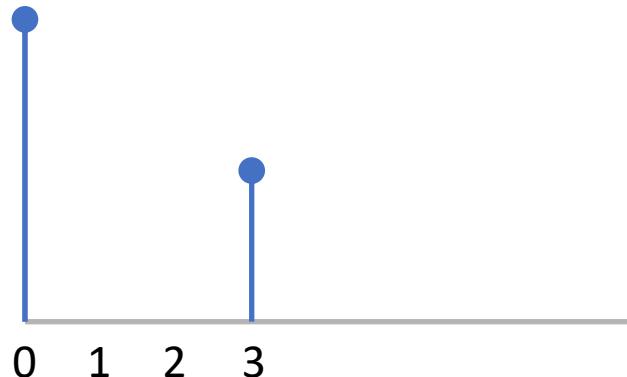


$$\text{Input } f[n] = 2 \cdot \delta[n] + \delta[n - 3]$$

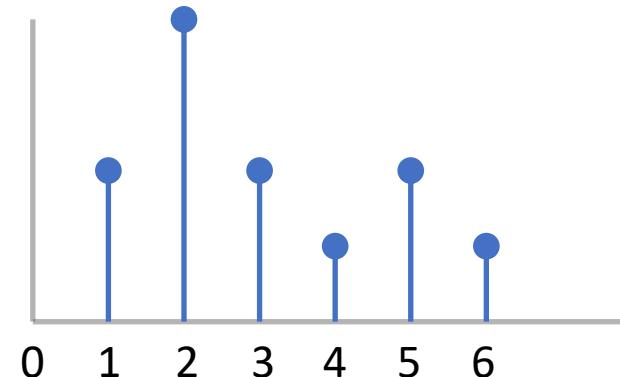


$$\text{Output } g[n] = 2 \cdot h[n] + h[n - 3]$$

Linear system



$$\text{Input } f[n] = 2 \cdot \delta[n] + \delta[n - 3]$$



$$\text{Output } g[n] = 2 \cdot h[n] + h[n - 3]$$

- If a filter is a linear system, when we combine two input signals linearly, their outputs will also be combined linearly.
- For example, if we know input $f_1[n]$ leads to output $g_1[n]$, and input $f_2[n]$ leads to output $g_2[n]$, we will have
$$\text{output}(\alpha f_1[n] + \beta f_2[n]) = \alpha g_1[n] + \beta g_2[n]$$

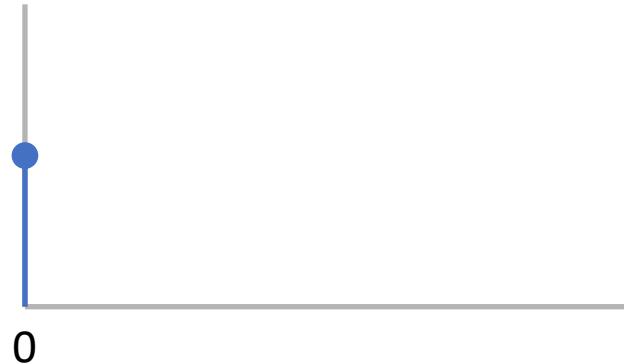
Linear time-invariant system

- Theories and methods are well developed for linear time-invariant systems.
- In our last lecture, the moving average filter, Gaussian filter and many other filters are linear time-invariant.
- For a linear time-invariant system, the impulse response h completely characterises how this system works.
- Given input f and impulse response h , we can derive the output g and define this operation as **convolution**.

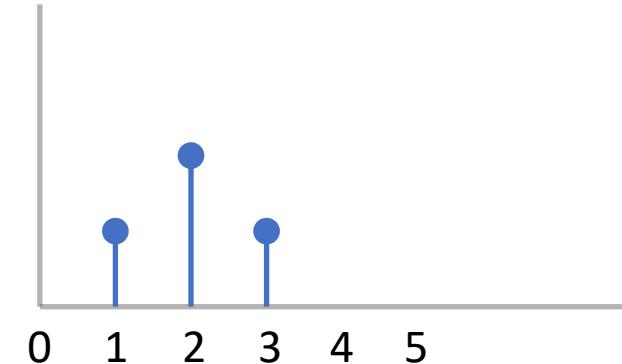
$$g[n] = f[n] * h[n]$$

↓
convolution

Impulse response

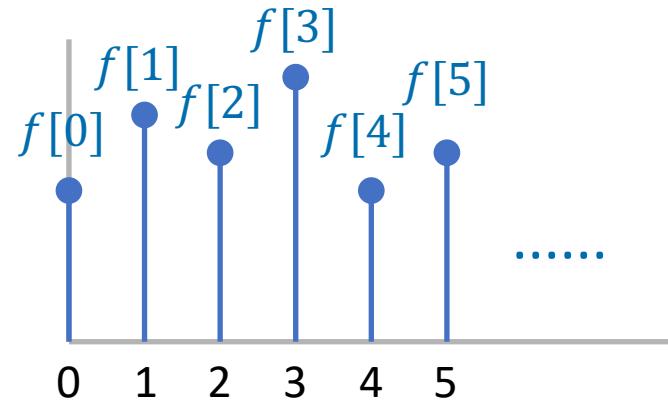


Input $f[n] = \delta[n]$

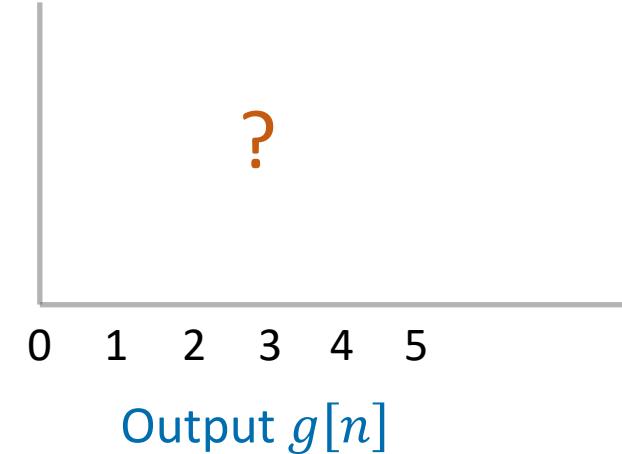


Output $g[n] = h[n]$

Linear time-invariant system

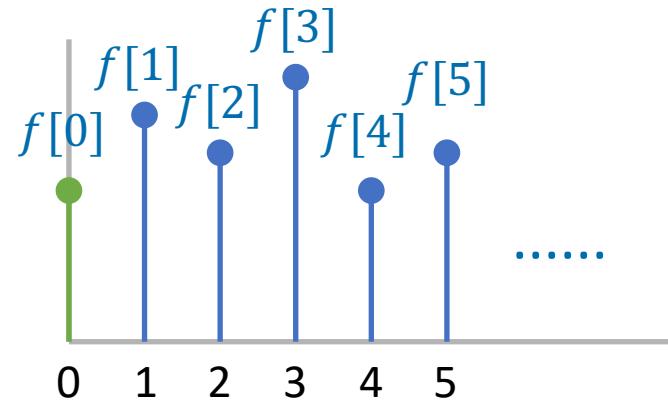


$$\text{Input } f[n] = f[0]\delta[n] + f[1]\delta[n - 1] + f[2]\delta[n - 2] + f[3]\delta[n - 3] + \dots$$

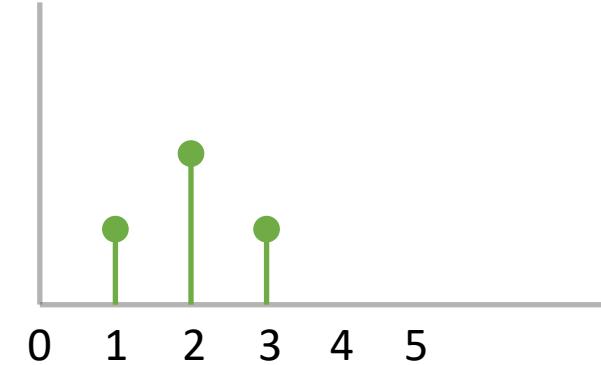


$$\text{Output } g[n]$$

Linear time-invariant system

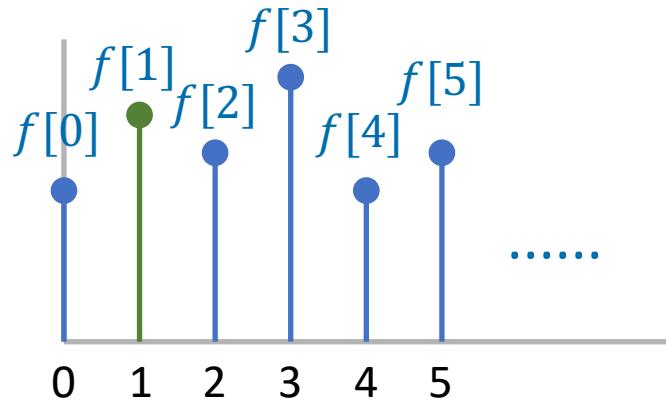


$$\begin{aligned} \text{Input } f[n] = & f[0]\delta[n] + f[1]\delta[n - 1] \\ & + f[2]\delta[n - 2] + f[3]\delta[n - 3] + \dots \end{aligned}$$

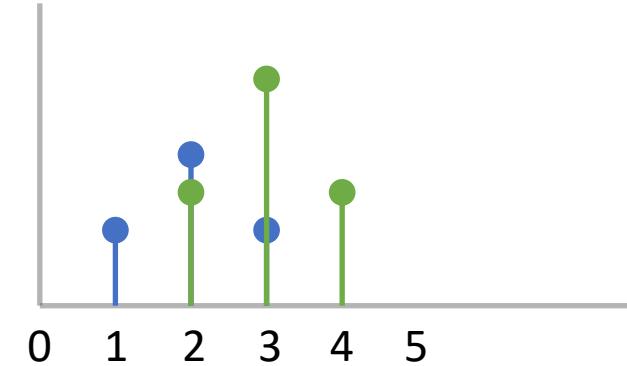


$$\text{Output } g[n] = f[0]h[n] + \dots$$

Linear time-invariant system

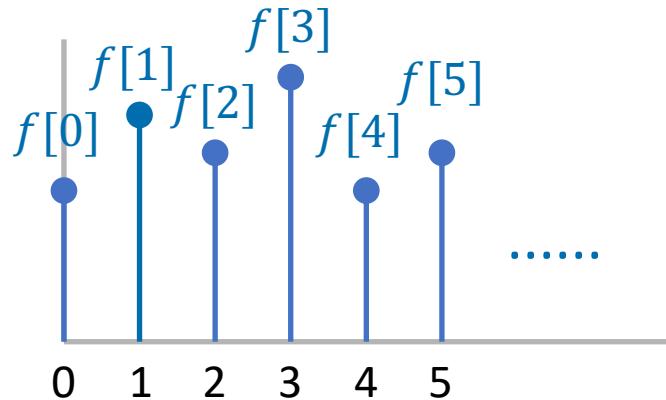


$$\text{Input } f[n] = f[0]\delta[n] + f[1]\delta[n - 1] + f[2]\delta[n - 2] + f[3]\delta[n - 3] + \dots$$

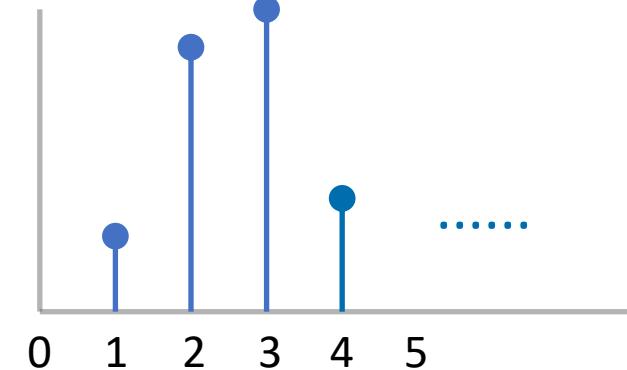


$$\text{Output } g[n] = f[0]h[n] + f[1]h[n - 1] + \dots$$

Linear time-invariant system



$$\text{Input } f[n] = f[0]\delta[n] + f[1]\delta[n - 1] + f[2]\delta[n - 2] + f[3]\delta[n - 3] + \dots$$



$$\text{Output } g[n] = f[0]h[n] + f[1]h[n - 1] + f[2]h[n - 2] + f[3]h[n - 3] + \dots$$

This mathematical operation is defined as convolution.

Convolution

- Convolution of a signal f and a filter with impulse response h is defined as,

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

↓
impulse response,
convolution kernel,
filter

Commutativity

- We notice that

$$\sum_{m=-\infty}^{\infty} f[m]h[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]h[m]$$

- This means that the convolution of f and h is equivalent to the convolution of h and f , known as commutativity,

$$f[n] * h[n] = h[n] * f[n]$$

An example of convolution

- Given a filter with impulse response h ,

$$h[-1] = 1/3$$

$$h[0] = 1/3$$

$$h[1] = 1/3$$

- The output of the filter can be calculated using convolution,

$$\begin{aligned}g[n] &= f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m] \\&= f[n-1]h[1] + f[n]h[0] + f[n+1]h[-1] \\&= f[n-1] \cdot \frac{1}{3} + f[n] \cdot \frac{1}{3} + f[n+1] \cdot \frac{1}{3}\end{aligned}$$

This shows how moving average can be formulated as a convolution operation.

Implementation

- In general, given a filter impulse response h

...	$h[-2]$	$h[-1]$	$h[0]$	$h[1]$	$h[2]$...
-----	---------	---------	--------	--------	--------	-----

and an input signal f

$f[0]$...	$f[-2]$	$f[-1]$	$f[n]$	$f[1]$	$f[2]$...	
--------	-----	---------	---------	--------	--------	--------	-----	--

their convolution can be calculated using *flip-and-shift*.

$$\begin{aligned}g[n] = f[n] * h[n] &= \sum_{m=-\infty}^{\infty} f[n-m]h[m] \\&= \dots + f[n-2]h[2] + f[n-1]h[1] + f[n]h[0] + f[n+1]h[-1] \\&\quad + f[n+2]h[-2] + \dots\end{aligned}$$

This is the definition of convolution and what we want to calculate.

Implementation

- In general, given a filter impulse response h

...	$h[-2]$	$h[-1]$	$h[0]$	$h[1]$	$h[2]$...
-----	---------	---------	--------	--------	--------	-----

and an input signal f

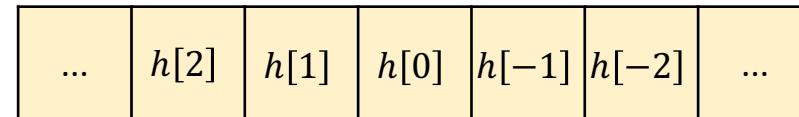
$f[0]$...	$f[-2]$	$f[-1]$	$f[n]$	$f[1]$	$f[2]$...	
--------	-----	---------	---------	--------	--------	--------	-----	--

their convolution can be calculated using *flip-and-shift*.

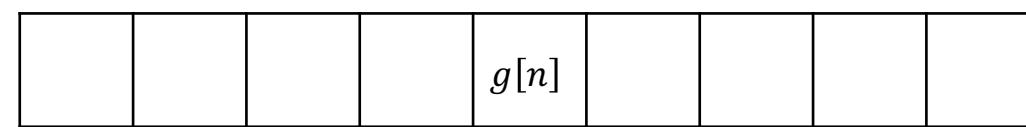
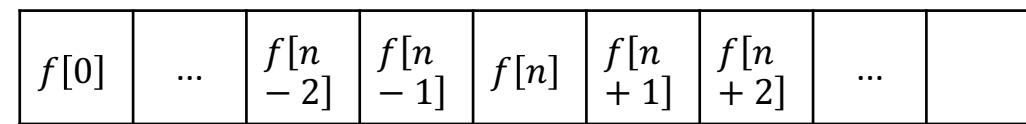
$$\begin{aligned}g[n] = f[n] * h[n] &= \sum_{m=-\infty}^{\infty} f[n-m]h[m] \\&= \dots + f[n-2]h[2] + f[n-1]h[1] + f[n]h[0] + f[n+1]h[-1] \\&\quad + f[n+2]h[-2] + \dots\end{aligned}$$

This is the definition of convolution and what we want to calculate.

Implementation



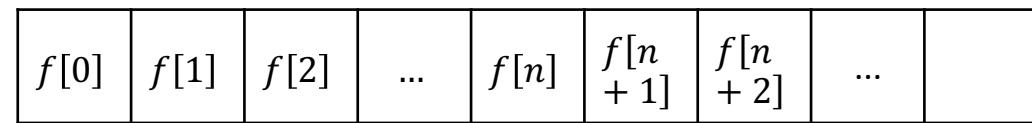
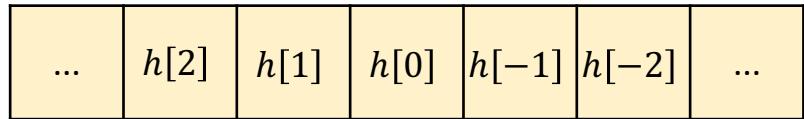
Flip the impulse response



Multiply and sum, get output $g[n]$

$$\begin{aligned} g[n] &= f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m] \\ &= \dots + f[n-2]h[2] + f[n-1]h[1] + f[n]h[0] + f[n+1]h[-1] + f[n+2]h[-2] + \dots \end{aligned}$$

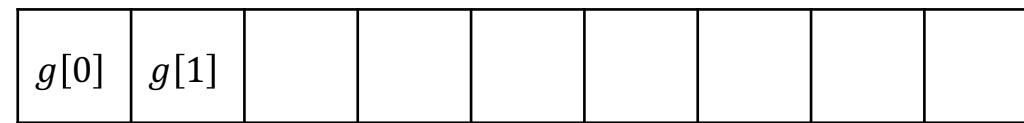
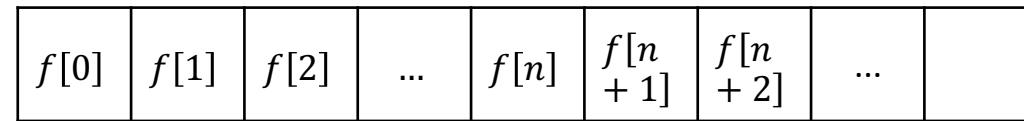
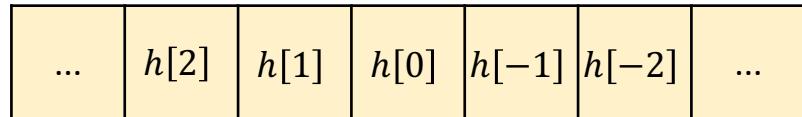
Implementation



Shift it across the signal

Multiply and sum, get output $g[0]$

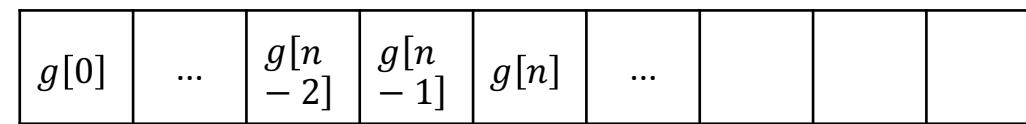
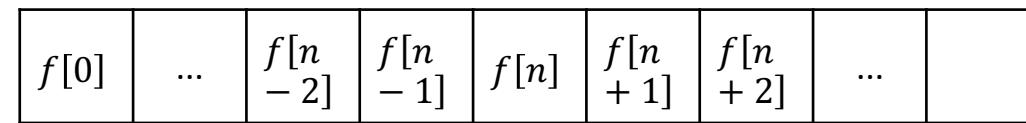
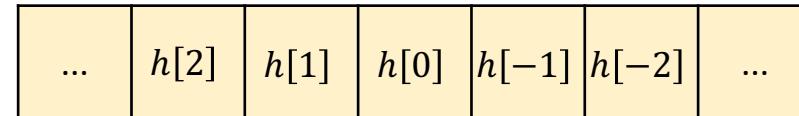
Implementation



Shift it across the signal

Multiply and sum, get output $g[1]$

Implementation



Shift it across the signal

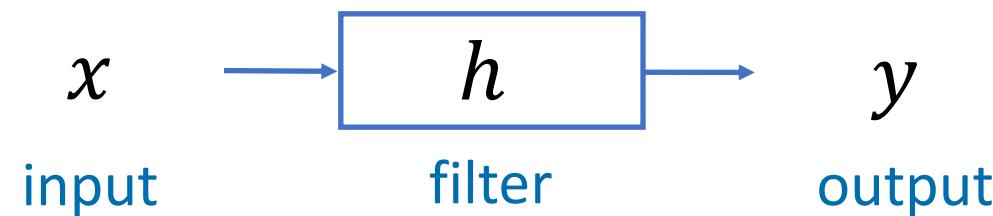
Multiply and sum, get output $g[n]$

$$\begin{aligned} g[n] &= f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m] \\ &= \dots + f[n-2]h[2] + f[n-1]h[1] + f[n]h[0] + f[n+1]h[-1] + f[n+2]h[-2] + \dots \end{aligned}$$

By using flip-and-shift, we implement the convolution operation.

2D convolution

- Previously, we define the convolution operation in 1D.
- For 2D (or high-dimensional) cases, convolution can be similarly defined, where x , h and y become 2D.



2D convolution

Kernel h

1/9	1/9	1/9	111	110	123	130	130
1/9	1/9	1/9	111	113	120	126	125
1/9	1/9	1/9	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Input $f[m, n]$

Output $g[m, n]$

2D convolution

- Convolution of 2D signal f with kernel h is defined as,

$$g[m, n] = f[m, n] * h[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j]h[m - i, n - j]$$

- It can be re-written as,

$$\begin{aligned} g[m, n] &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[m - i, n - j]h[i, j] && \text{replace } m - i, n - j \text{ by } i, j \\ &= \cdots + f[m - 1, n - 1]h[1, 1] + f[m - 1, n]h[1, 0] + f[m - 1, n + 1]h[1, -1] \\ &\quad + f[m, n - 1]h[0, 1] + f[m, n]h[0, 0] + f[m, n + 1]h[0, -1] \\ &\quad + f[m + 1, n - 1]h[-1, 1] + f[m + 1, n]h[-1, 0] + f[m + 1, n + 1]h[-1, -1] + \cdots \end{aligned}$$

2D convolution

- Convolution of 2D signal f with kernel h is defined as,

$$g[m, n] = f[m, n] * h[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j]h[m - i, n - j]$$

- It can be re-written as,

$$\begin{aligned} g[m, n] &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[m - i, n - j]h[i, j] && \text{replace } m - i, n - j \text{ by } i, j \\ &= \cdots + f[m - 1, n - 1]h[1, 1] + f[m - 1, n]h[1, 0] + f[m - 1, n + 1]h[1, -1] \\ &\quad + f[m, n - 1]h[0, 1] + f[m, n]h[0, 0] + f[m, n + 1]h[0, -1] \\ &\quad + f[m + 1, n - 1]h[-1, 1] + f[m + 1, n]h[-1, 0] + f[m + 1, n + 1]h[-1, -1] + \cdots \end{aligned}$$

2D convolution

	$f_{m-1, n-1}$	$f_{m-1, n}$	$f_{m-1, n+1}$						
	$f_{m, n-1}$	$f_{m, n}$	$f_{m, n+1}$						
	$f_{m+1, n-1}$	$f_{m+1, n}$	$f_{m+1, n+1}$						

Input $f[m, n]$

Output $g[m, n]$

2D convolution

Kernel h

$h_{-1, -1}$	$h_{-1, 0}$	$h_{-1, 1}$
$h_{0, -1}$	$h_{0, 0}$	$h_{0, 1}$
$h_{1, -1}$	$h_{1, 0}$	$h_{1, 1}$

Output $g[m, n]$

2D convolution

Flip
kernel

$h_{1,1}$	$h_{1,0}$	$h_{1,-1}$
$h_{0,1}$	$h_{0,0}$	$h_{0,-1}$
$h_{-1,1}$	$h_{-1,0}$	$h_{-1,-1}$

Output $g[m, n]$

2D convolution

$f_{m-1, n-1}$	$f_{m-1, n}$	$f_{m-1, n+1}$							
$f_{m, n-1}$	$f_{m, n}$	$f_{m, n+1}$							
$f_{m+1, n-1}$	$f_{m+1, n}$	$f_{m+1, n+1}$							

Input $f[m, n]$

Output $g[m, n]$

2D convolution

Multiply
and sum

	$f_{h_{1,-1}^{n-1}}$	$f_{h_{1,0}^{n-1}}$	$f_{h_{1,+1}^{n-1}}$						
	$f_{h_{0,-1}^m}$	$h_{0,0}^m$	$h_{0,+1}^m$						
	$h_{-1,-1}^{m+1}$	$h_{-1,0}^{m+1}$	$h_{-1,+1}^{m+1}$						

Input $f[m, n]$

Output $g[m, n]$

2D convolution

Shift

		$h_{1,1}$	$h_{1,0}$	$h_{1,-1}$					
		$h_{0,1}$	$h_{0,0}$	$h_{0,-1}$					
		$h_{-1,1}$	$h_{-1,0}$	$h_{-1,-1}$					

Input $f[m, n]$

Output $g[m, n]$

2D convolution

Shift

			$h_{1,1}$	$h_{1,0}$	$h_{1,-1}$				
			$h_{0,1}$	$h_{0,0}$	$h_{0,-1}$				
			$h_{-1,1}$	$h_{-1,0}$	$h_{-1,-1}$				

Input $f[m, n]$

Output $g[m, n]$

2D convolution

- If you define the kernel $h[m, n]$ to be

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

,

moving average can be performed by using the convolution operation

$$g[m, n] = f[m, n] * h[m, n]$$

Associativity

- Convolution satisfies the associativity,

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

- You can check the associativity by expanding the left equation.

$$\begin{aligned} f * (g * h) &= \sum_{m=-\infty}^{\infty} f[m](g * h)[n - m] \\ &= \sum_{m=-\infty}^{\infty} f[m] \sum_{k=-\infty}^{m=\infty} g[k]h[n - m - k] = \dots \end{aligned}$$

- Similarly expand the right equation $(f * g) * h$, using a scratch paper and using change of variables technique, you will find the associativity holds.

Associativity

- Convolution satisfies the associativity,

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

- What does this mean for image filtering?

- Separable filtering

- If a big filter h_{Big} is equivalent to the convolution of two small filters h_1 and h_2 , then

$$f * h_{Big} = f * (h_1 * h_2) = (f * h_1) * h_2$$

Associativity

- The moving average filter is separable.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

0	0	0
1/3	1/3	1/3
0	0	0

*

0	1/3	0
0	1/3	0
0	1/3	0

Average in a
2D window,
 h_{Big}

Average
across row,
 h_1

Average
across column,
 h_2

Image filtering as convolution

- In this lecture, we provide the mathematical foundation for image filtering.
- For a linear time-invariant system, filtering can be described by a mathematical operation, called convolution.
- Why do we introduce convolution?
 - It bridges the knowledge in image filtering to the theory of signal processing.
 - Practically, it means the convolution functions in Python (`scipy.signal.convolve`) or Matlab (`conv`) can be used for image filtering.
 - Associativity of the convolution enables us to perform separable filtering.

References

- Section 3.2: Linear filtering; Section 3.3.1: Non-linear filtering. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Edge Detection I

Dr Wenjia Bai

Department of Computing & Brain Sciences

Edge Detection

- What is edge?
- Why are we interested in edges?
- How do we detect edges?

What is edge?

- Cambridge dictionary
 - Edge: The outer or furthest point of something.
- Computer vision
 - Edge refers to lines where image brightness changes sharply and has discontinuities.
 - It happens due to various reasons, such as colour discontinuity, depth discontinuity, surface normal discontinuity etc.



colour discontinuity

depth discontinuity: building vs background

surface normal
discontinuity

What is edge?



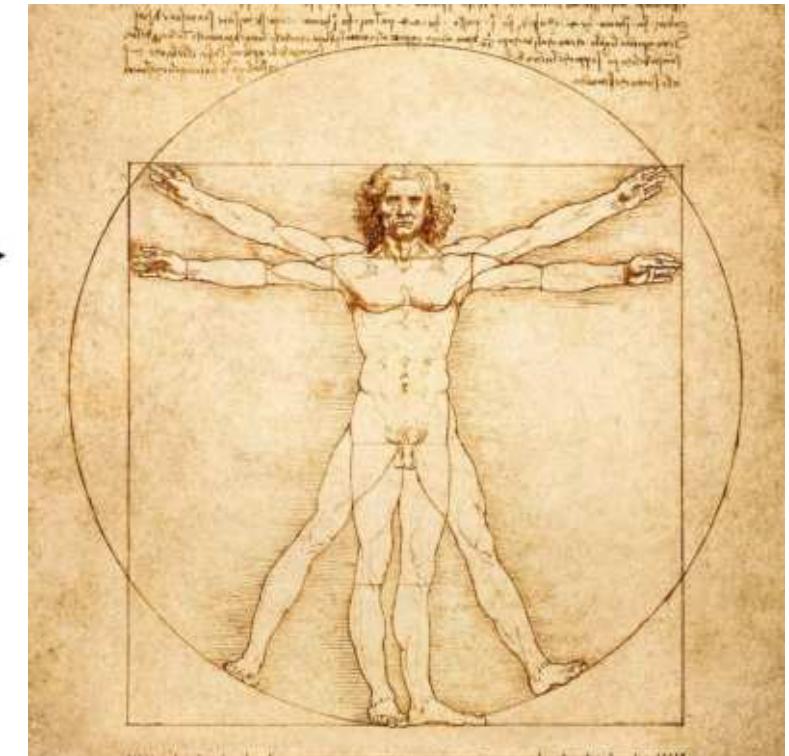
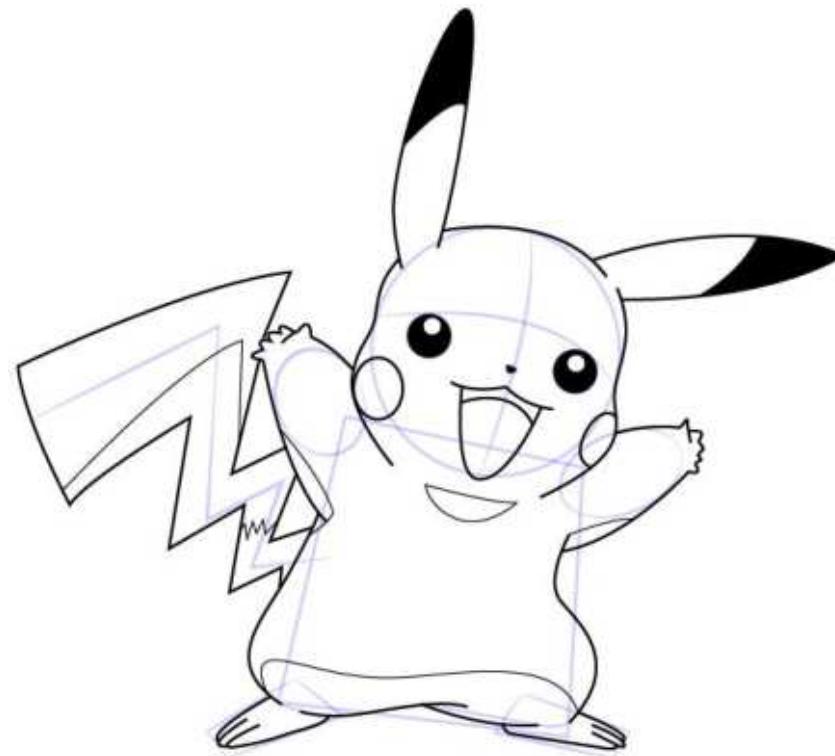
Input



Edge detection

Why are we interested in edges?

- Edges capture important properties of what we see in this world.
- Edges are important features for analysing images.



Why are we interested in edges?

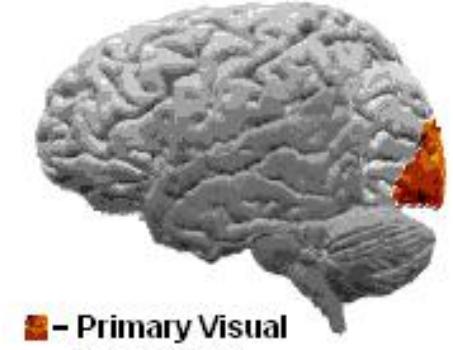
- Hubel and Wiesel are Nobel laureates in Physiology and Medicine in 1981.
- Their contributions are about understanding the underlying mechanism of the brain and information processing in the visual system.



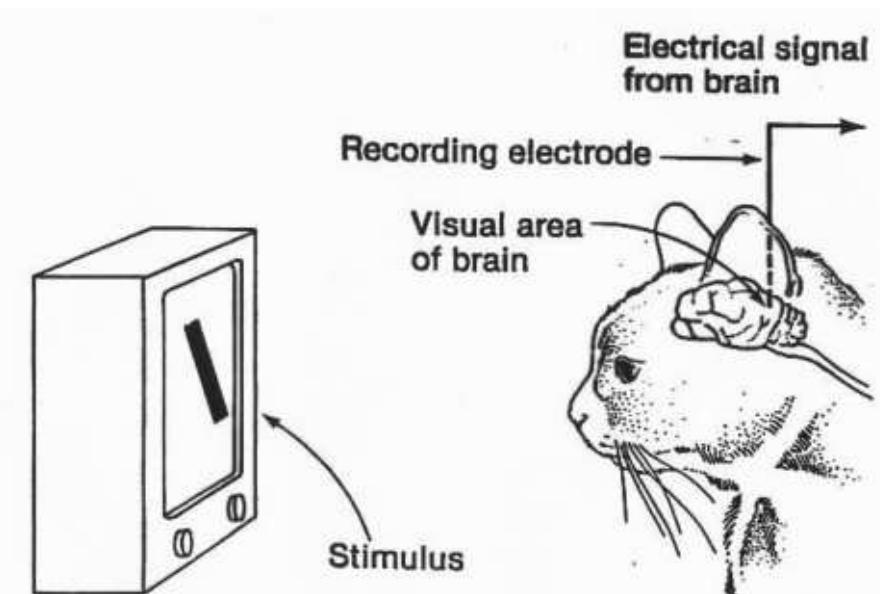
Torsten Wiesel (left) and David Hubel (right)

Hubel and Wiesel

- Vision experiments in 1959
 - At a basement at John Hopkins University
 - Recorded the brain electrical signal of a cat to certain visual stimulus.
 - Found that V1 neuron cells are orientation selective, which respond strongly to lines or **edges** of a particular orientation.



Visual area 1 (V1) of human brain



Vision experiments (cat or monkey)

RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE OF MONKEY STRIATE CORTEX

By D. H. HUBEL AND T. N. WIESEL

From the Department of Physiology, Harvard Medical School,
Boston, Mass., U.S.A.

(Received 6 October 1967)

SUMMARY

1. The striate cortex was studied in lightly anaesthetized macaque and spider monkeys by recording extracellularly from single units and stimulating the retinas with spots or patterns of light. Most cells can be categorized as simple, complex, or hypercomplex, with response properties very similar to those previously described in the cat. On the average, however, receptive fields are smaller, and there is a greater sensitivity to changes in stimulus orientation. A small proportion of the cells are colour coded.

2. Evidence is presented for at least two independent systems of columns extending vertically from surface to white matter. Columns of the first type contain cells with common receptive-field orientations. They are similar to the orientation columns described in the cat, but are probably smaller in cross-sectional area. In the second system cells are aggregated into columns according to eye preference. The ocular dominance columns are larger than the orientation columns, and the two sets of boundaries seem to be independent.

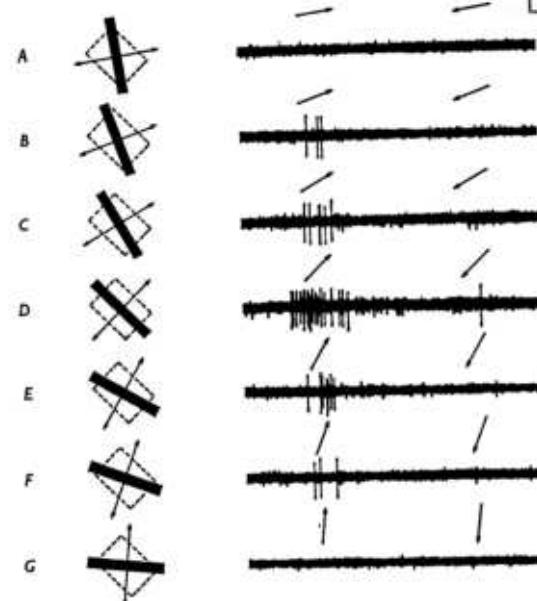
3. There is a tendency for cells to be grouped according to symmetry of responses to movement; in some regions the cells respond equally well to the two opposite directions of movement of a line, but other regions contain a mixture of cells favouring one direction and cells favouring the other.

4. A horizontal organization corresponding to the cortical layering can also be discerned. The upper layers (II and the upper two-thirds of III) contain complex and hypercomplex cells, but simple cells are virtually absent. The cells are mostly binocularly driven. Simple cells are found deep in layer III, and in IV A and IV B. In layer IV B they form a large proportion of the population, whereas complex cells are rare. In layers IV A and IV B one finds units lacking orientation specificity; it is not clear whether these are cell bodies or axons of geniculate cells. In layer IV most cells are driven by one eye only; this layer consists of a mosaic with

MONKEY STRIATE CORTEX

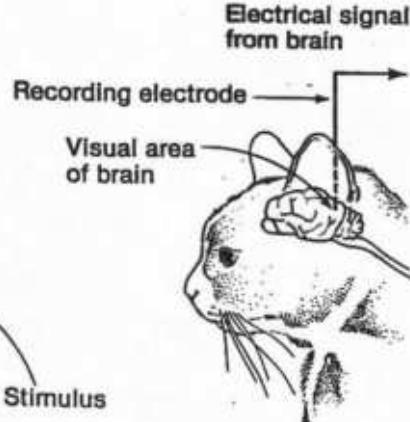
showed little or no directional preference. Even when response was asymmetrical, the less effective direction of movement usually gave some minimal response (see Text-fig. 2), but there were a few cases in which the maintained activity was actually suppressed.

Individual complex cells differed markedly in their responsiveness to slits, edges, or dark bars. The majority responded very well to one than to the other two, but some reacted briskly to two or even to all three. For a cell that was sensitive to slits, but not

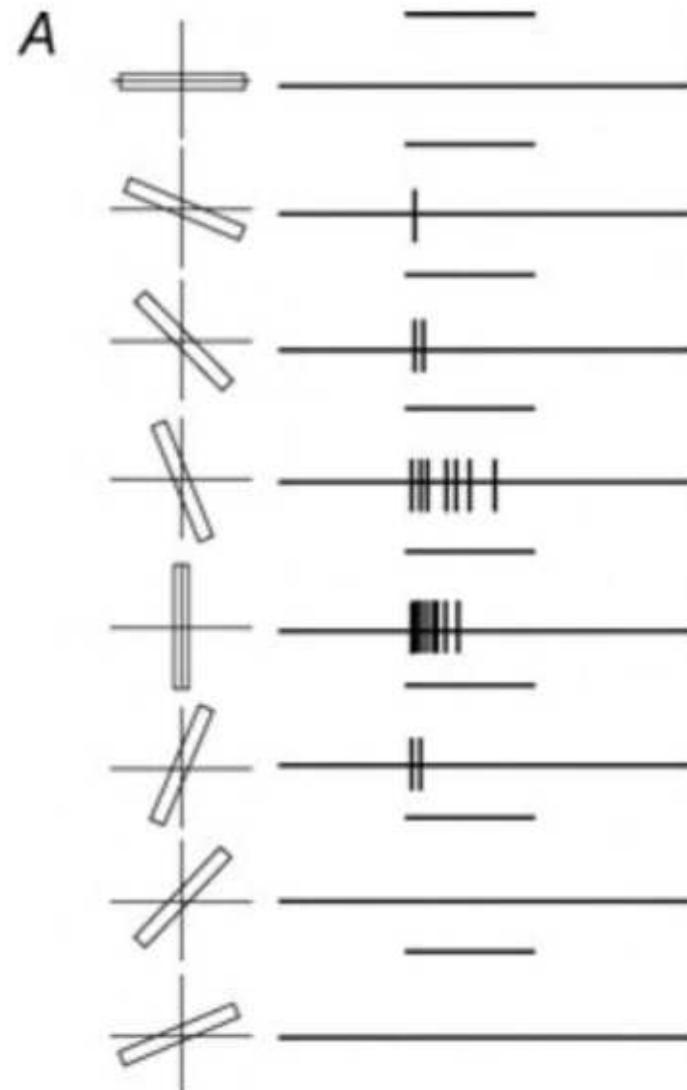


Text-fig. 2. Responses of a complex cell in right striate cortex (layer IV A) to various orientations of a moving black bar. Receptive field in the left eye indicated by the interrupted rectangles; it was approximately $\frac{1}{2} \times \frac{1}{2}$ in size, and was situated 4° below and to the left of the point of fixation. Ocular-dominance group 4. Duration of each record, 2 sec. Background intensity 1.3 log₁₀ cd/m², dark bars 0.0 log cd/m².

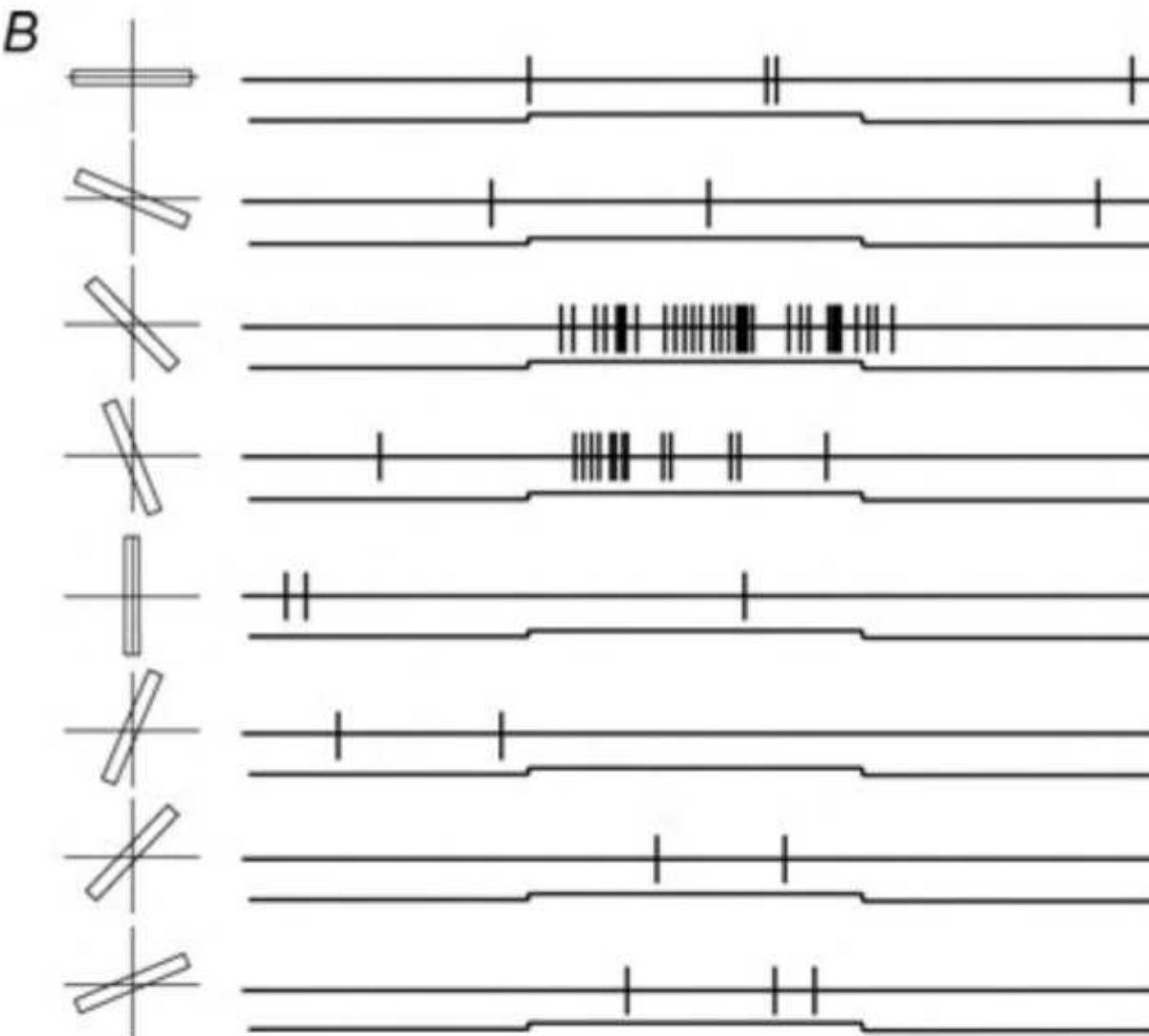
responses increased as slit width was increased up to some optimal value, and then they fell off sharply; the optimum width was always a small fraction of the width of the whole field. For complex cells that responded best to edges, some reacted to one configuration and also to its mirror



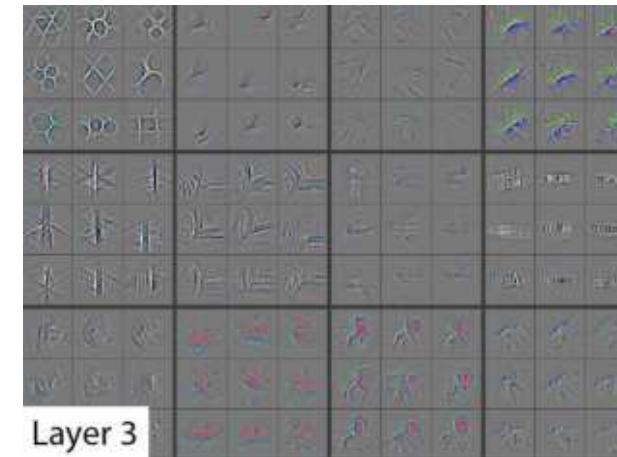
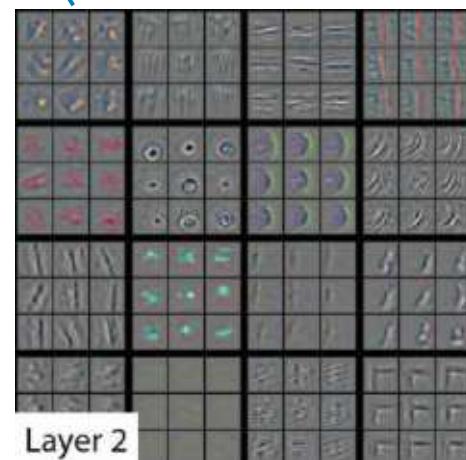
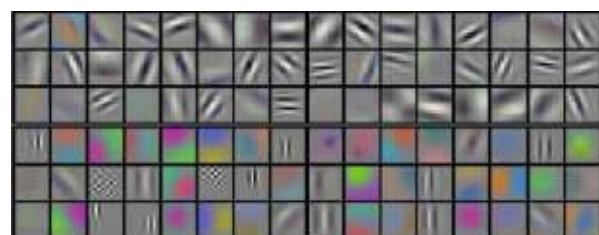
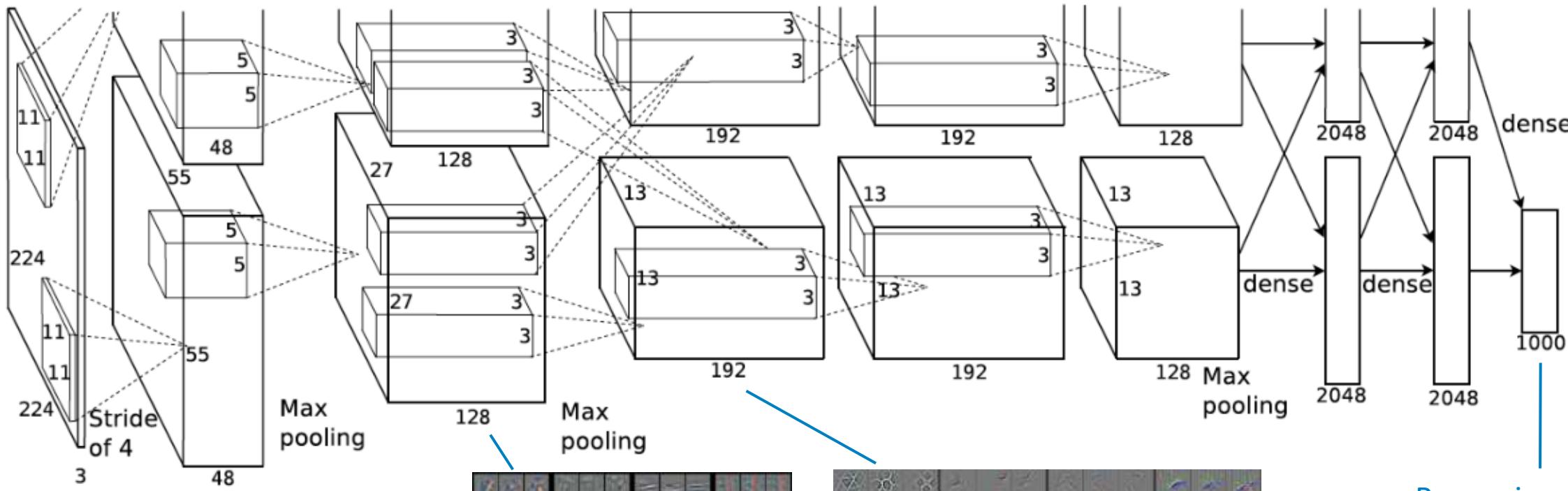
Strongest response is recorded when the bar is of a particular orientation.



(A) Anaesthetised cat.



(B) Awake monkey.



Recognise objects

Why are we interested in edges?

- Edges provide important low-level features for both animal vision system and computer vision for analysing and understanding images.

How do we detect edges?

- An image can be regarded as a function of pixel position.
- Derivatives characterise the discontinuities of a function, which can help us detect edges.



x

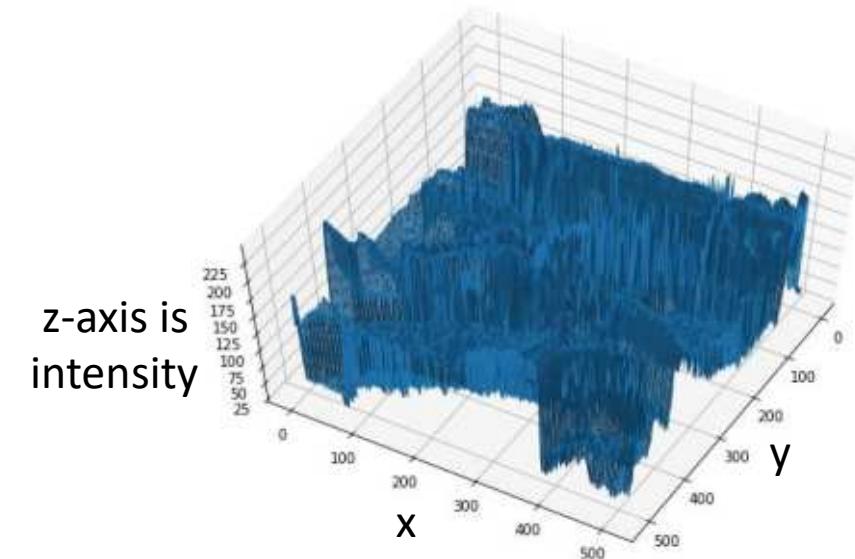
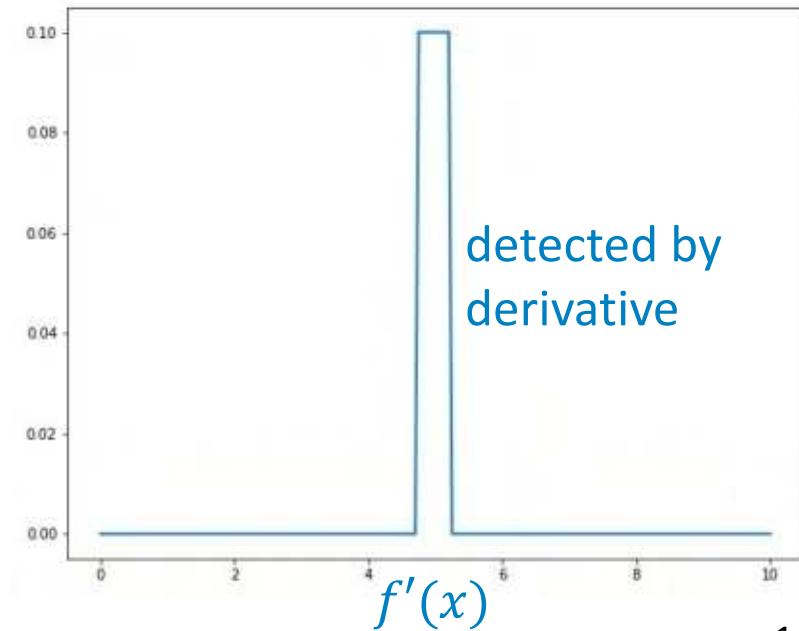
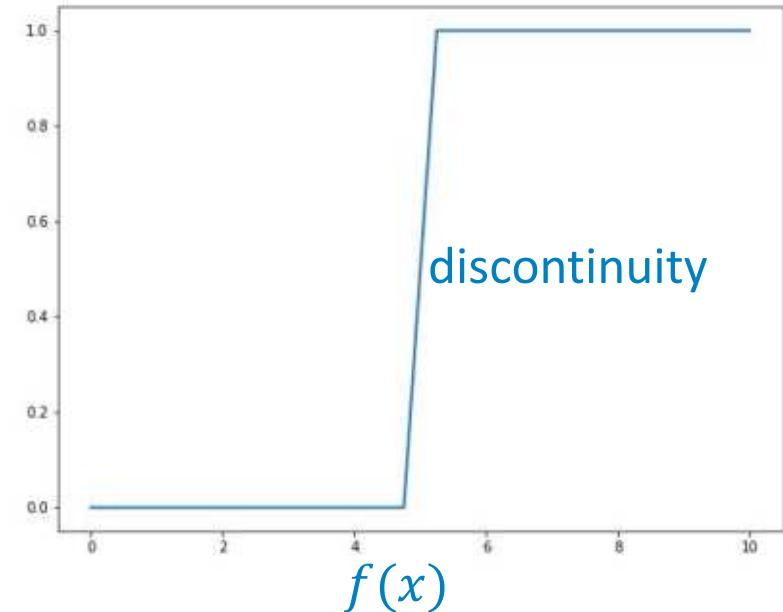


Image as a function of position

Derivatives

- Definition for a continuous function

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



Derivatives

- Definition for a continuous function

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- Definition for a discrete function, i.e. finite difference

Forward difference: $f'[x] = f[x + 1] - f[x]$

Finite difference can be performed using convolution with these kernels.

$$h = [1, -1, 0]$$

Backward difference: $f'[x] = f[x] - f[x - 1]$

$$h = [0, 1, -1]$$

Central difference: $f'[x] = \frac{f[x+1] - f[x-1]}{2}$

$$h = [1, 0, -1]$$

Derivatives implemented as convolution

- Recap of convolution:

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

- How do we implement the central difference using convolution?

$$f'[x] = \frac{f[x+1] - f[x-1]}{2}$$

Convolution

$$f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m]$$

- If we do not account the denominator of 2,

$$\begin{aligned} f'[x] &= f[x+1] - f[x-1] \\ &= f[x+1] * 1 + f[x] * 0 + f[x-1] * (-1) \\ &\quad \downarrow \quad \downarrow \quad \downarrow \\ &\quad h[-1] \quad h[0] \quad h[1] \end{aligned}$$

- Therefore, the convolution kernel is

$$h = [1, 0, -1]$$

- We can implement central difference with this convolution kernel.

$$f'[x] = f[x] * h[x]$$

Convolution

$$f[n] * h[n] = \sum_{m=-\infty}^{\infty} f[n-m]h[m]$$

$$h[n] = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$h[-n] = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

flip the kernel

* * * multiply and sum

$$f[n] = \begin{bmatrix} 10 & 10 & 10 & 50 & 20 & 0 & 0 \end{bmatrix}$$

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

Convolution

$$h[n] = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

$$h[-n] = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \text{ shift across signal}$$

$$f[n] = \begin{bmatrix} 10 & 10 & 10 & 50 & 20 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} & 0 & 40 & 10 & -50 & -20 & \end{bmatrix}$$

Large derivatives
denote discontinuities.

Derivatives

- Finite difference

$$\text{Forward difference: } f'[x] = f[x + 1] - f[x] \quad h = [1, -1, 0]$$

$$\text{Backward difference: } f'[x] = f[x] - f[x - 1] \quad h = [0, 1, -1]$$

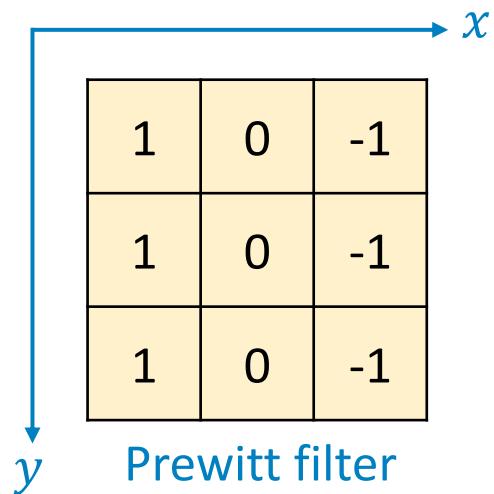
$$\text{Central difference: } f'[x] = \frac{f[x+1] - f[x-1]}{2} \quad h = [1, 0, -1]$$

- Using the finite difference, we can detect discontinuities in 1D. How do we extend to edge detection in a 2D image?
 - We can design similar filters.

Edge detection filters

- Prewitt filter
- Sobel filter
- Smoothing in edge detection

Prewitt filter



Prewitt filter
along x-axis or
horizontal
direction

A 3x3 kernel for edge detection, labeled as the Prewitt filter along the y-axis or vertical direction. The kernel is represented by a 3x3 grid of values:

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

Prewitt filter
along y-axis or
vertical
direction

Prewitt filter

- Prewitt filter is a separable filter.

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

Prewitt filter

=

0	1	0
0	1	0
0	1	0

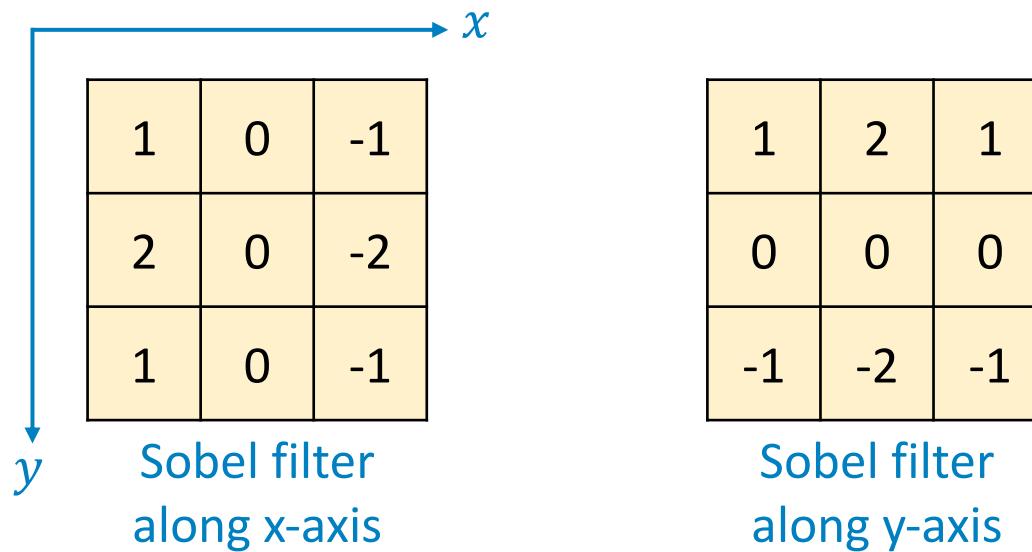
moving averaging

*

0	0	0
1	0	-1
0	0	0

finite difference

Sobel filter



Sobel filter

- Sobel filter is also a separable filter.

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

Prewitt filter

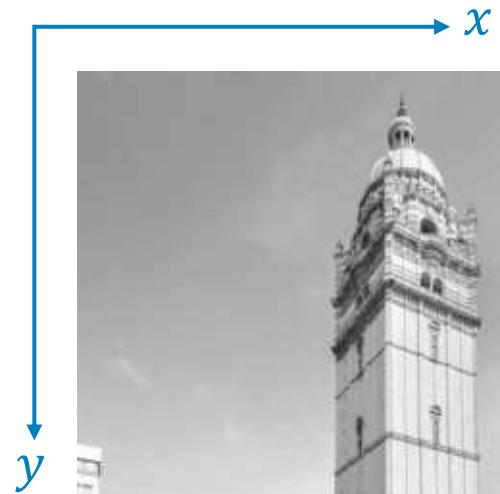
0	1	0
0	2	0
0	1	0

weighted averaging

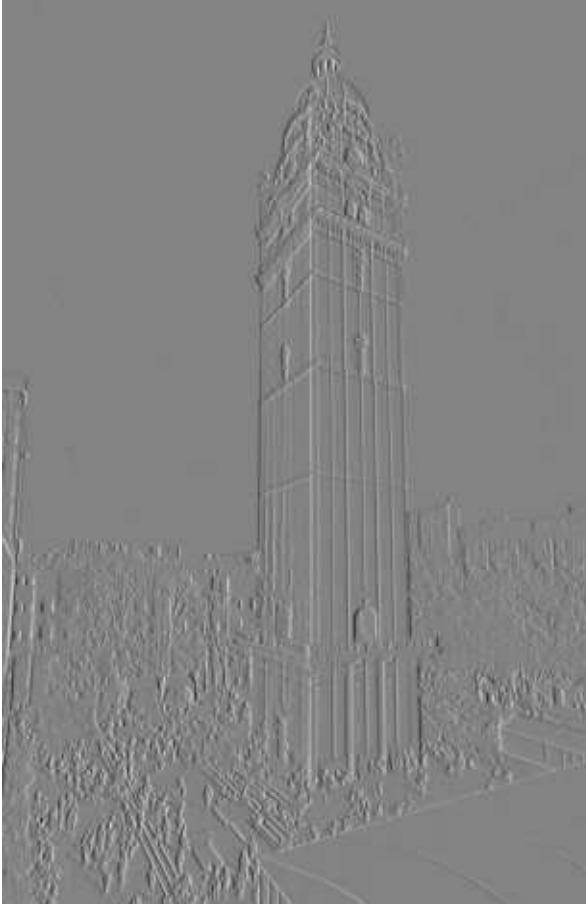
0	0	0
1	0	-1
0	0	0

finite difference

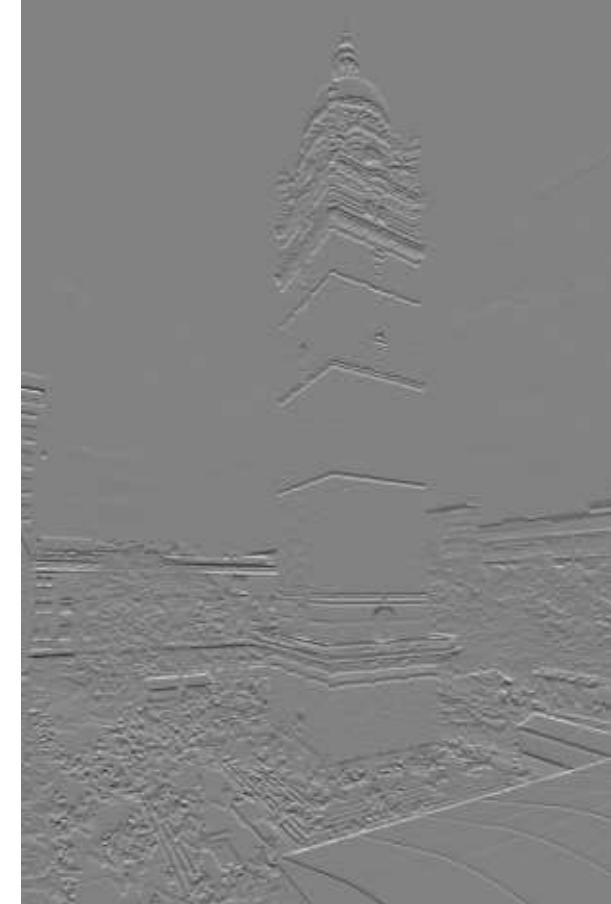
Sobel filter



Input image



Sobel x-axis



Sobel y-axis

Image gradient

- At each pixel there are two outputs from Sobel filters, one describing discontinuity along x-axis, the other describing discontinuity along y-axis.
- In mathematics, this is called gradient,

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

Let's denote the gradient as $\begin{bmatrix} g_x \\ g_y \end{bmatrix}$ to be consistent with notations in convolution.

- We can describe two properties for the gradient, namely its magnitude and its orientation.

Image gradient

- Compute the derivatives along x-axis and y-axis

$$g_x = f * h_x$$

$$g_y = f * h_y$$

- The magnitude of the gradient is

$$g = \sqrt{g_x^2 + g_y^2}$$

- The orientation or angle of the gradient

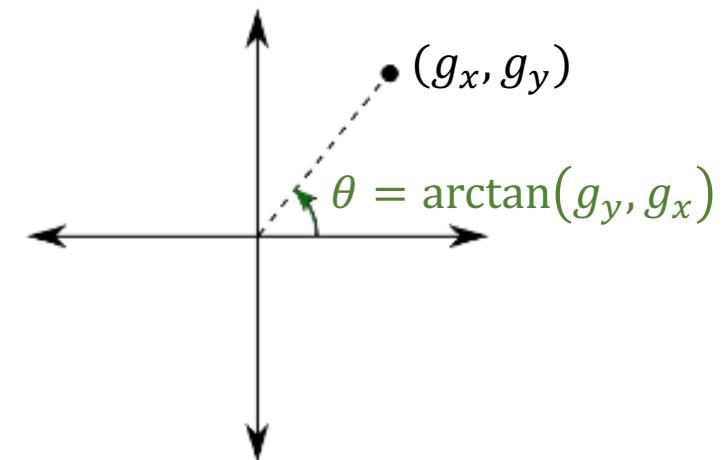
$$\theta = \arctan(g_y, g_x)$$

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

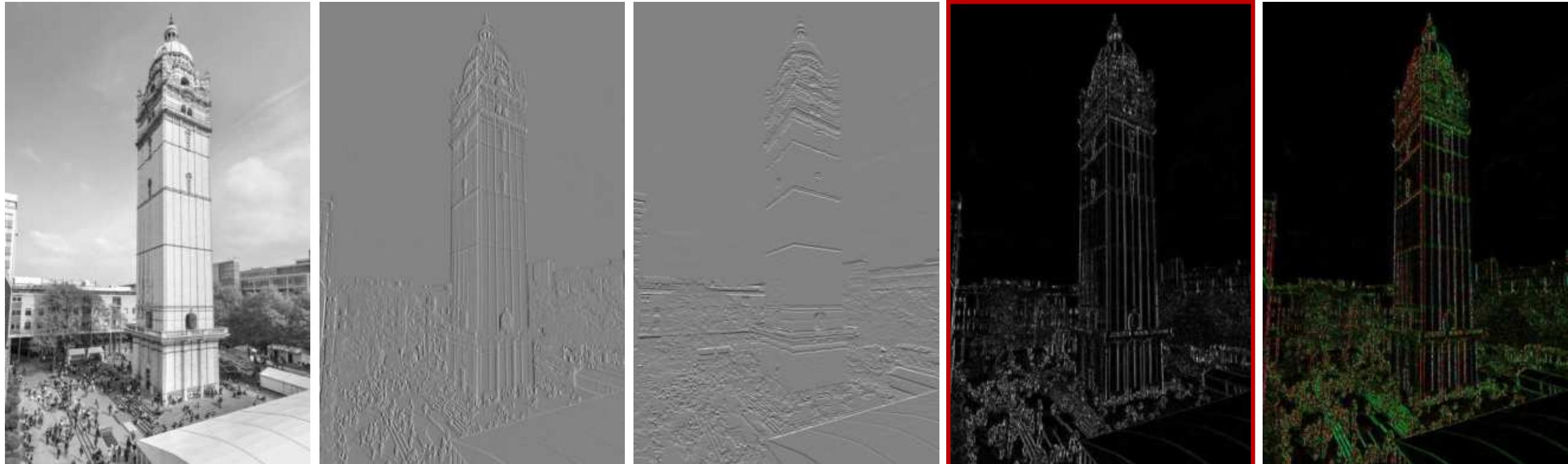
Sobel filter h_x

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

Sobel filter h_y



Sobel filter



Input image

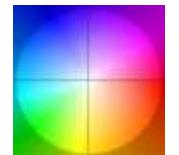
Sobel x-axis

Sobel y-axis

Magnitude

Orientation

Orientation colour wheel



Smoothing

- Why do we have a moving average filter in the Prewitt filter or similarly in the Sobel filter?
 - Derivatives are sensitive to noise.
 - The smoothing kernel can help suppress the noise.

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

Prewitt filter

0	1	0
0	1	0
0	1	0

moving averaging

0	0	0
1	0	-1
0	0	0

finite difference

Smoothing

- Why do we have a moving average filter in the Prewitt filter or similarly in the Sobel filter?
 - Derivatives are sensitive to noise.
 - The smoothing kernel can help suppress the noise.

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

Sobel filter

0	1	0
0	2	0
0	1	0

weighted averaging

0	0	0
1	0	-1
0	0	0

finite difference

Smoothing

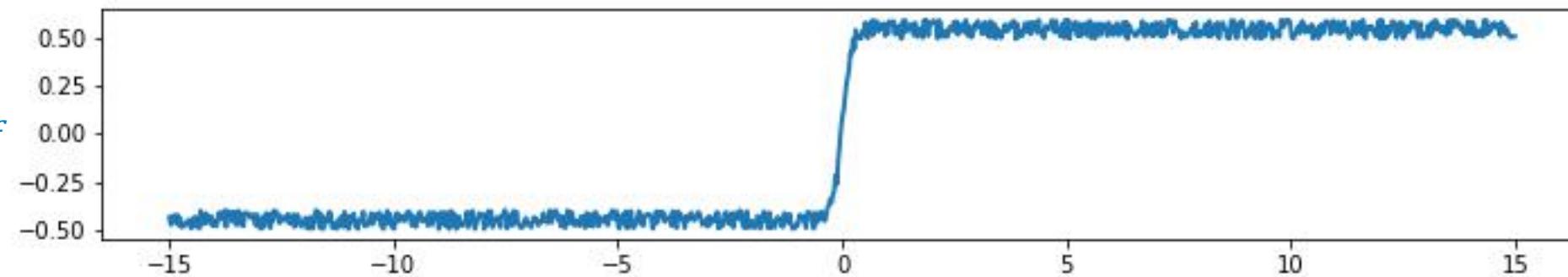
- Prewitt filters use the mean average kernel [1, 1, 1] for smoothing.
- Sobel filters use the kernel [1, 2, 1] for smoothing.
- Another option is to use the Gaussian kernel to smooth the signal before calculating the derivatives.

$$h[x] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

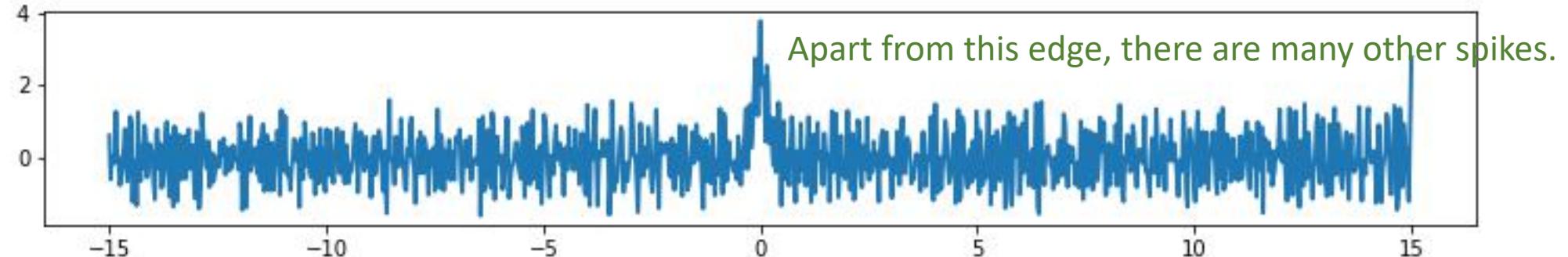
- We will show why smoothing is needed in edge detection for noisy signals.

Derivative of a noisy signal

input signal f

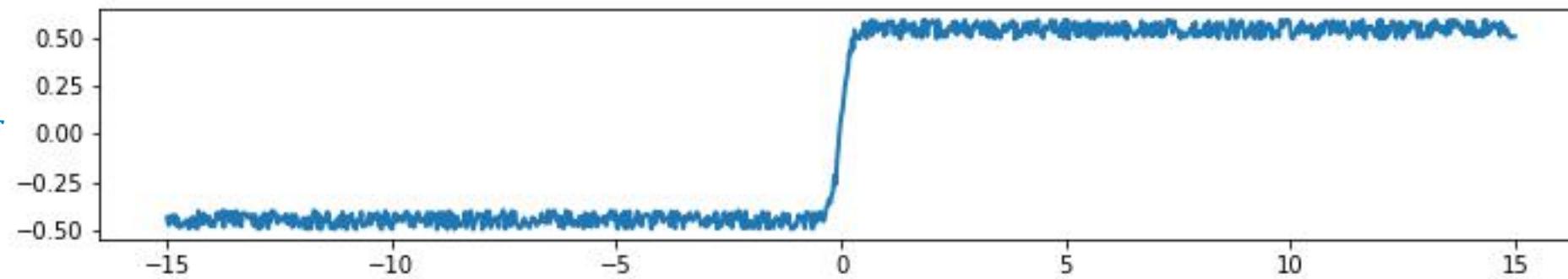


derivative $\frac{df}{dx}$

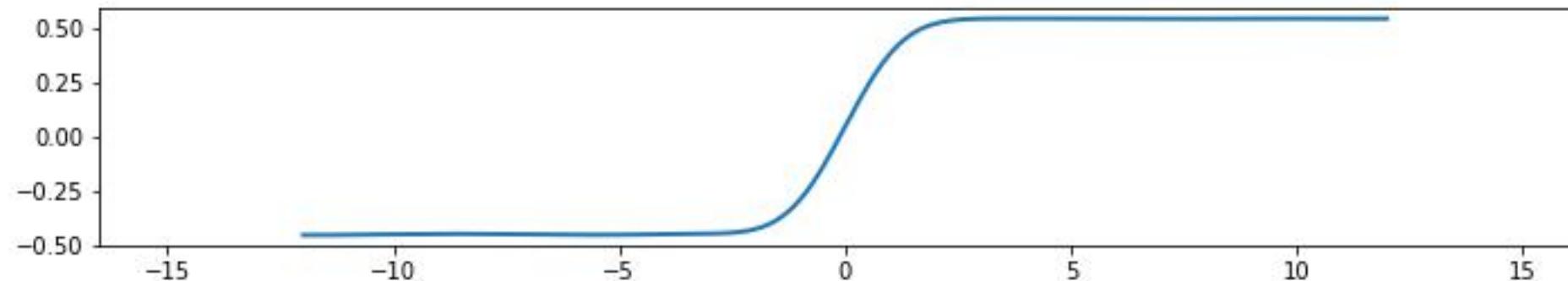


Derivative of a Gaussian smoothed signal

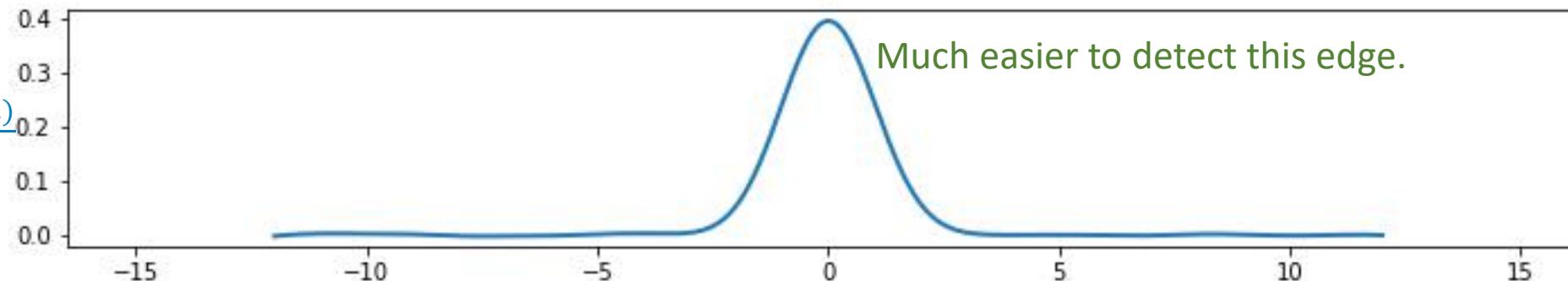
input signal f



Gaussian smoothed
 $f * h$



derivative $\frac{d(f * h)}{dx}$



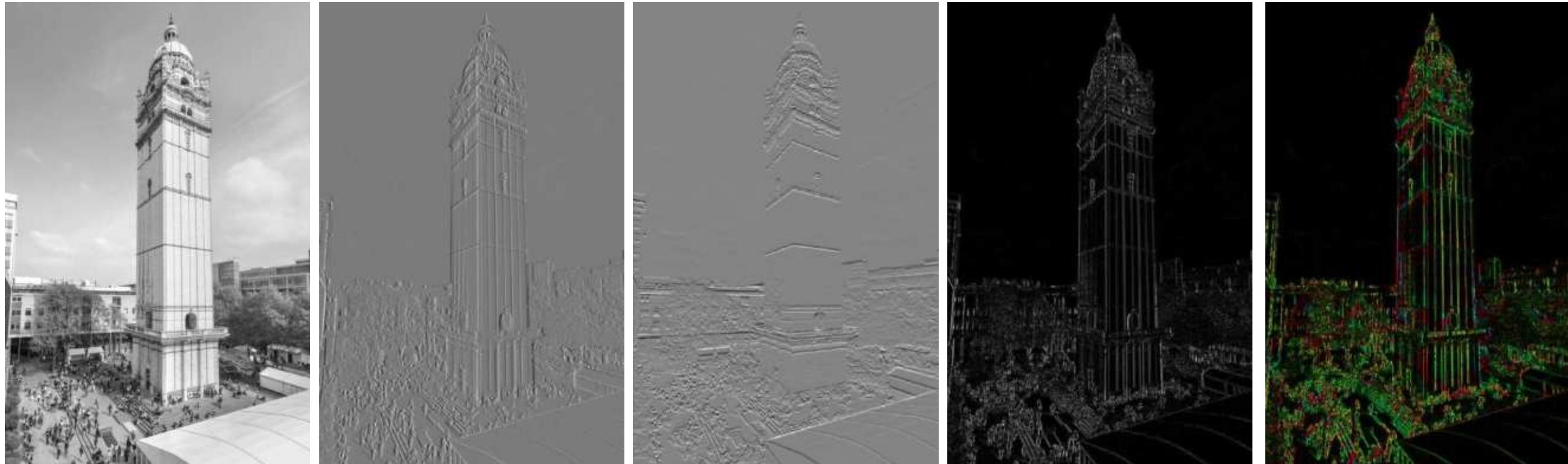
Smoothing

- For a 2D image, we can first perform smoothing with a 2D Gaussian filter.

$$h[x, y] = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Then take the derivatives along x-axis and y-axis using Prewitt or Sobel filters.
- Finally calculate the magnitude and orientation, which highlights the edges.

Image gradient after Gaussian smoothing



Input image

Derivative x-axis

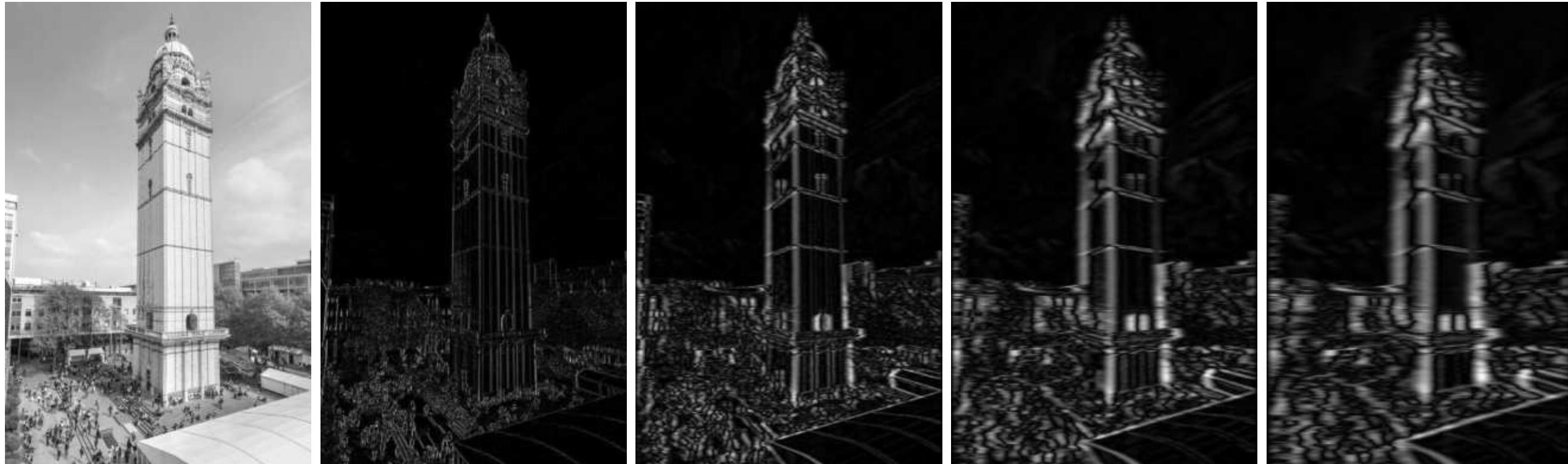
Derivative y-axis

Magnitude

Orientation

Parameter σ in the Gaussian filter

$$\text{Gaussian kernel: } h(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Input image

magnitude
 $\sigma = 1$ pixel

magnitude
 $\sigma = 5$ pixel

magnitude
 $\sigma = 9$ pixel

magnitude
 $\sigma = 13$ pixel

- Large σ value suppresses noise and results smoother derivative.
- Different σ values finds edges at different scales.

Edge detection

- Edge detection using the first derivative of an image.
 - Prewitt filter
 - Sobel filter
 - Smoothing in edge detection
- Now we can calculate both the magnitude and orientation of the edges.
- In the next lecture, we will talk about how to generate a binary edge map.

References

- Section 4.2.1: Edge detection. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Edge Detection II

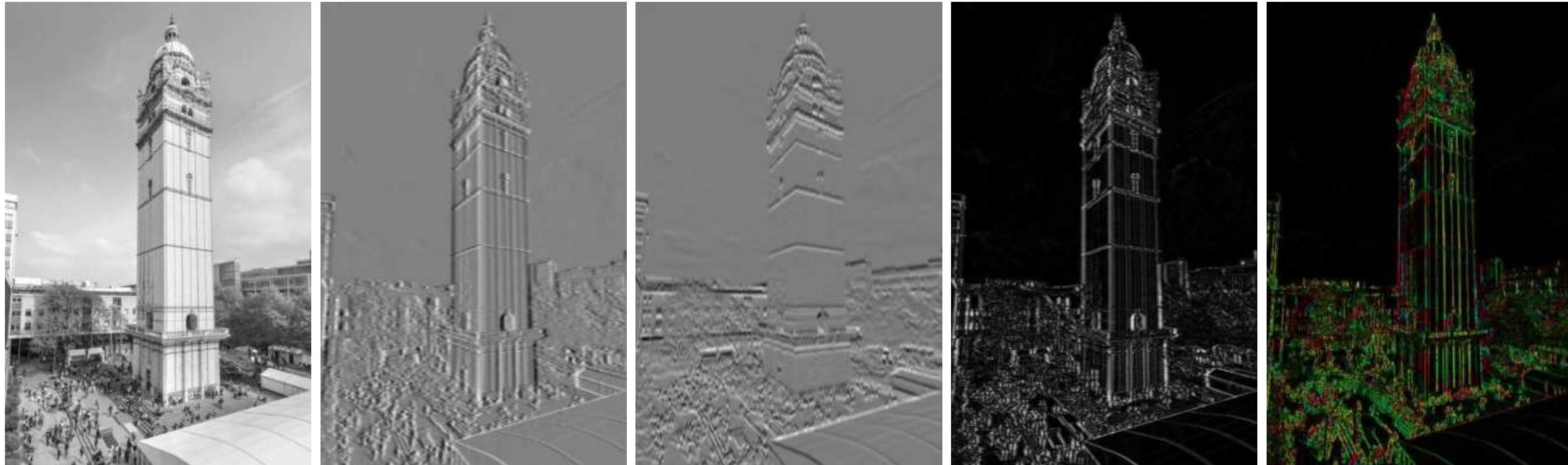
Dr Wenjia Bai

Department of Computing & Brain Sciences

How to generate a binary edge map?

- Canny edge detection
- Learning-based edge detection

Re-cap on edge detection



Input image

Derivative x-axis

Derivative y-axis

Magnitude

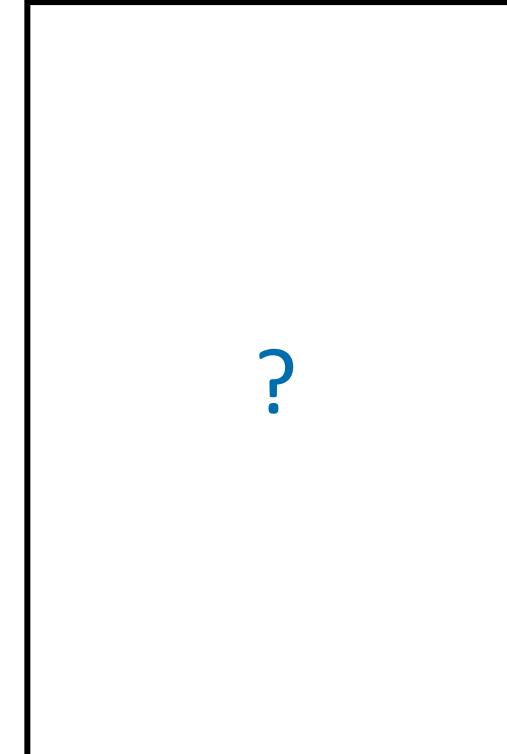
Direction

Edge detection

- Given the gradient magnitude map, where exactly are the edges?
- Can we generate a **binary** edge map?



Gradient magnitude map



Binary edge map

Canny edge detection



John Canny

- Canny proposed the following criteria for a good edge detector.
- Good detection
 - There should be a low probability of failing to mark real edge points, and low probability of falsely marking non-edge points.
- Good localisation
 - The points marked as edge points by the operator should be as close as possible to the centre of the true edge.
- Single response
 - Only one response to a single edge.

Canny edge detection

- Canny edge detection algorithm can be broken down to the following steps:
 - Perform Gaussian filtering to suppress noise.
 - Calculate the gradient magnitude and direction.
 - Apply non-maximum suppression (NMS) to get a single response for each edge.
 - Perform hysteresis thresholding to find potential edges.

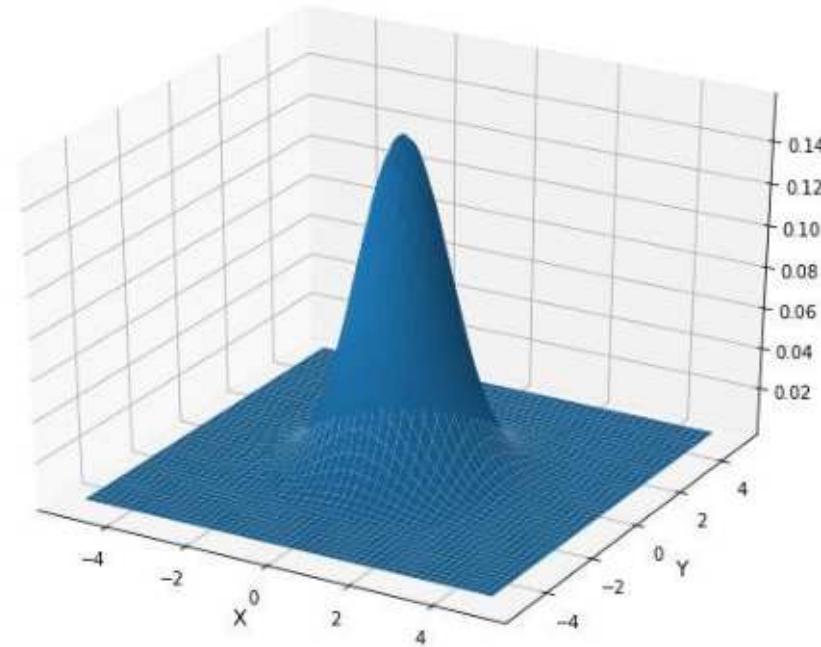


Input image

Gaussian filtering

- Gaussian filtering is performed to smooth image and suppress noise.
- The effect of smoothing can be adjusted by the parameter σ in the Gaussian kernel.

$$h(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

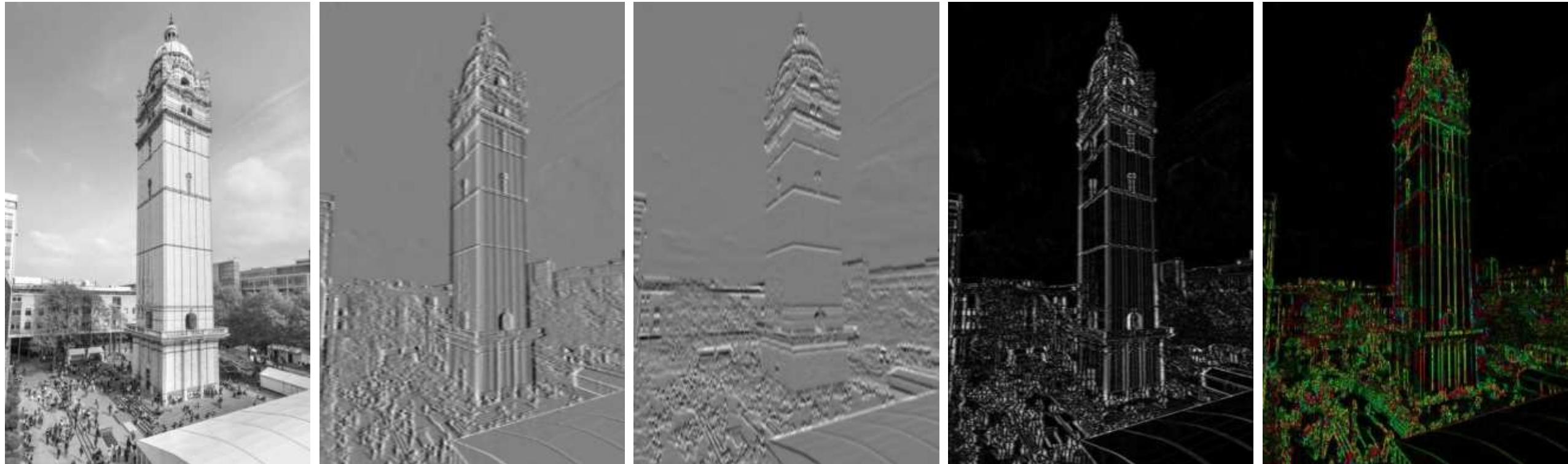


Gaussian filter

Image gradient

- The image gradient is calculated after Gaussian filtering.
- Prewitt or Sobel filtering can be applied.

Image gradient



Input image

Derivative x-axis

Derivative y-axis

Magnitude

Direction

Non-maximum suppression (NMS)

- Aim: To get a single response for an edge.
- Idea: Edge occurs where the gradient magnitude reaches the maximum.
- Let us zoom in a small region of the magnitude map in the next slide.

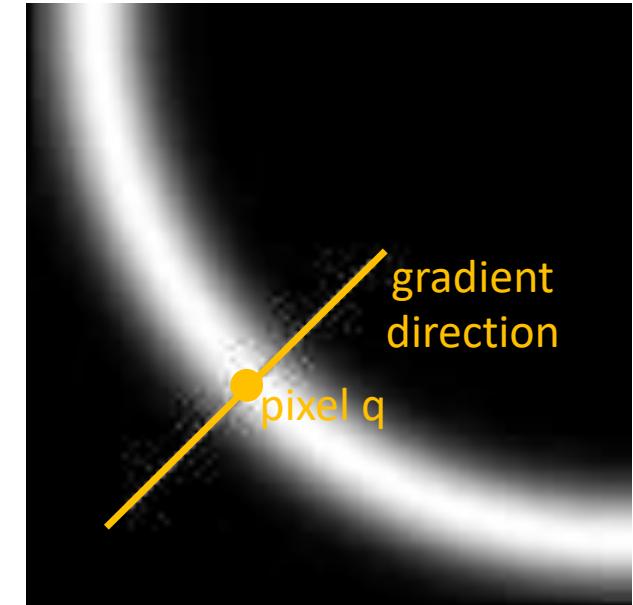


Magnitude

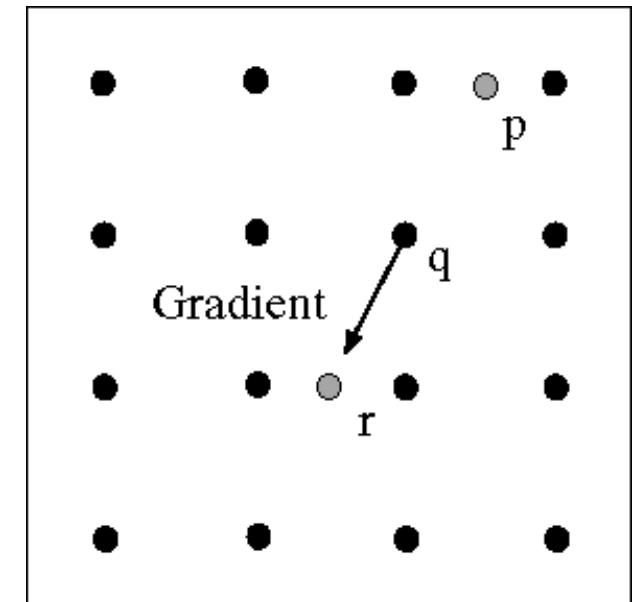
Non-maximum suppression (NMS)

- Check whether a pixel q is a local maximum along the gradient direction.
 - We move to pixel r and compare the gradient magnitudes between q and r .
 - Similarly, we move to pixel p and compare the gradient magnitudes between q and p .
 - If pixel q is the local maximum, it is an edge point and we suppress the other pixels, i.e. non-maximum pixels.
 - Suppression

$$M(x, y) = \begin{cases} M(x, y), & \text{if local maximum} \\ 0, & \text{otherwise} \end{cases}$$



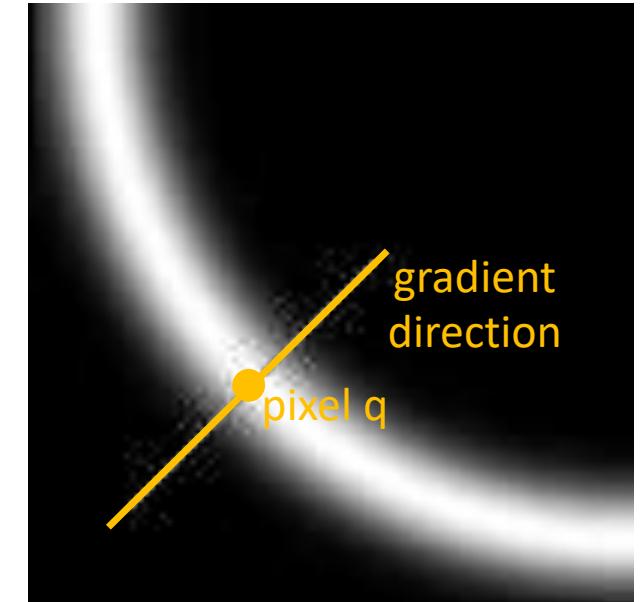
Gradient magnitude map



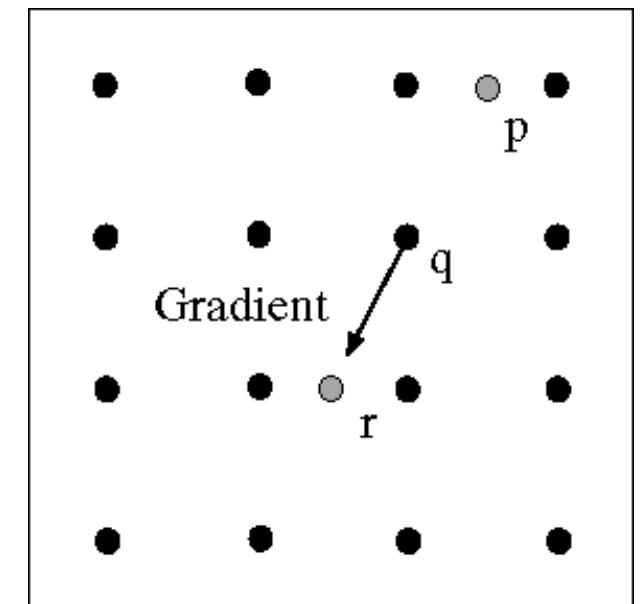
Pixels on the map

Non-maximum suppression (NMS)

- Check whether a pixel q is a local maximum along the gradient direction.
 - In implementation, we need to perform image interpolation for pixels p and r , if they are not located on the pixel lattice.
 - Using nearest neighbour or linear interpolation.
 - Alternatively, we can round the gradient direction into 8 possible angles.
 - $0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ$
 - We only check along these directions (horizontal, vertical, diagonal or anti-diagonal) so no interpolation is needed.

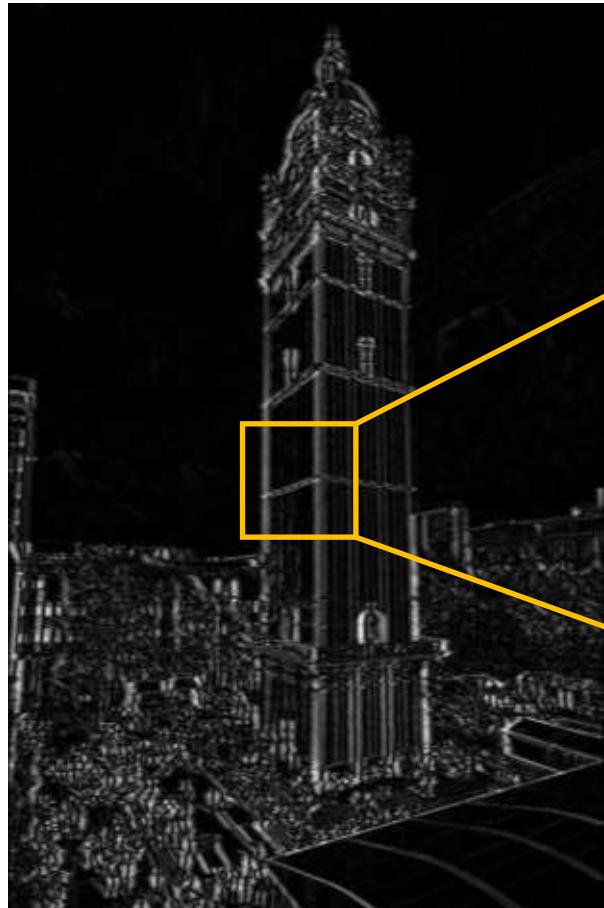


Gradient magnitude map

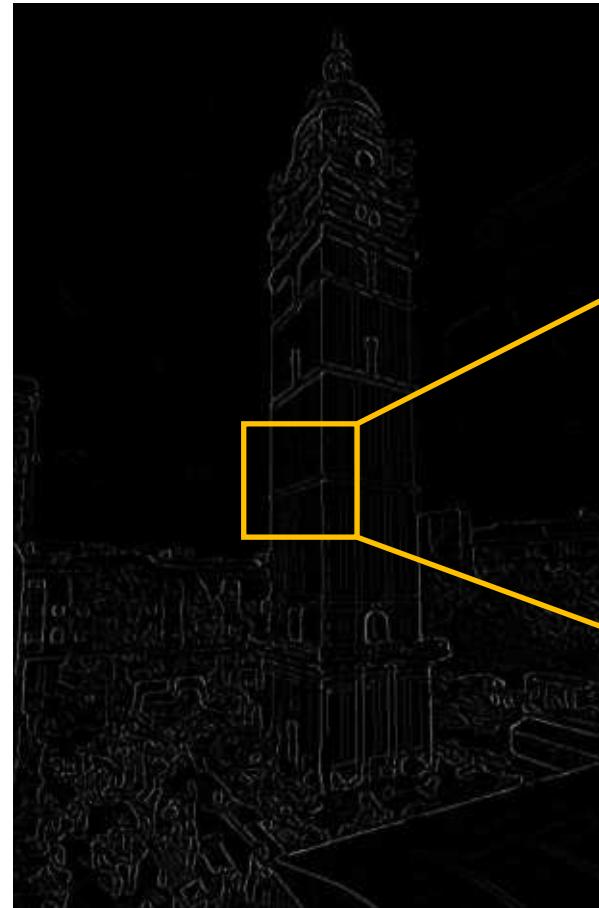


Pixels on the map

Non-maximum suppression (NMS)



Magnitude



After NMS

Thresholding

- We perform NMS for all pixels on the gradient magnitude map.
- Some pixels may be local maxima but they may have low magnitudes.
- We want edges that have high magnitudes. To this end, we perform thresholding.

Thresholding

- Purpose: to convert the magnitude map into a binary image.
- Simple thresholding
 - The most common method to convert an intensity image into a binary image with a threshold t .

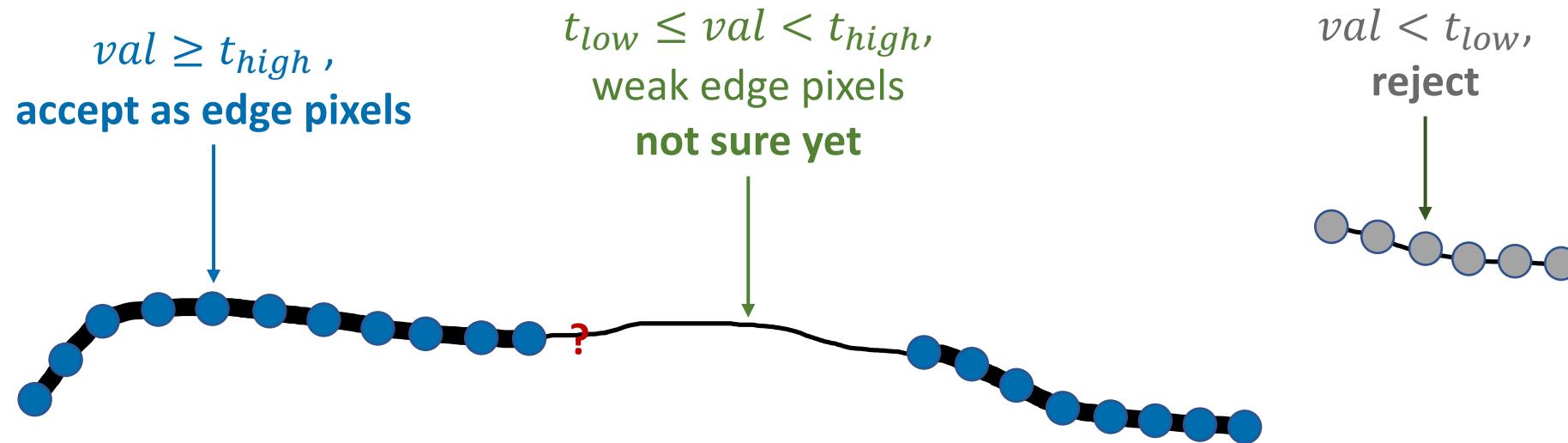
$$\text{binary}(x, y) = \begin{cases} 1, & \text{if } I(x, y) \geq t \\ 0, & \text{otherwise} \end{cases}$$

- Hysteresis thresholding
 - Canny edge detector performs hysteresis thresholding, which defines two thresholds.

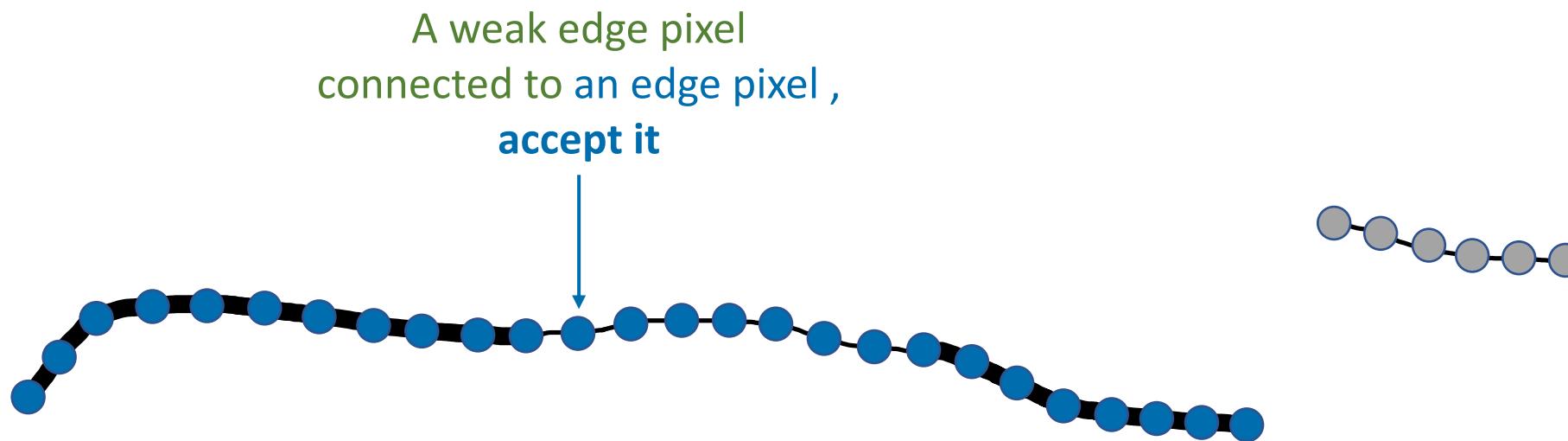
Hysteresis thresholding

- Hysteresis thresholding defines two thresholds t_{low} and t_{high} for edge detection.
 - If a pixel's gradient magnitude is $\geq t_{high}$, it is accepted as an edge pixel.
 - If a pixel's gradient magnitude is $< t_{low}$, it is rejected.
 - If it is in between, it is a weak edge pixel. It may be an edge or may be not.
 - We check its neighbouring pixels. It will be accepted if it is connected to an edge pixel.
 - This check is performed iteratively till all pixels are either accepted or rejected.

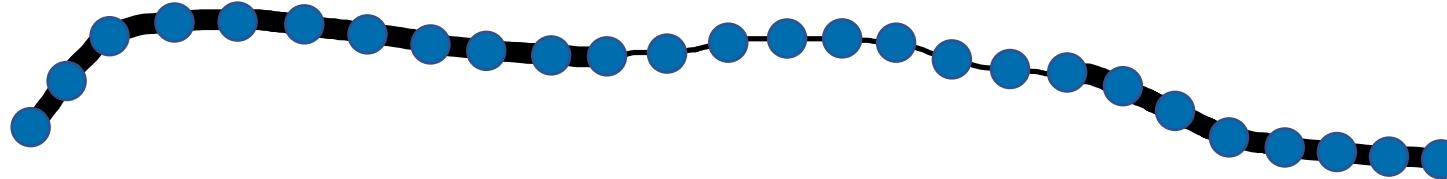
Hysteresis thresholding



Hysteresis thresholding



Thresholding result



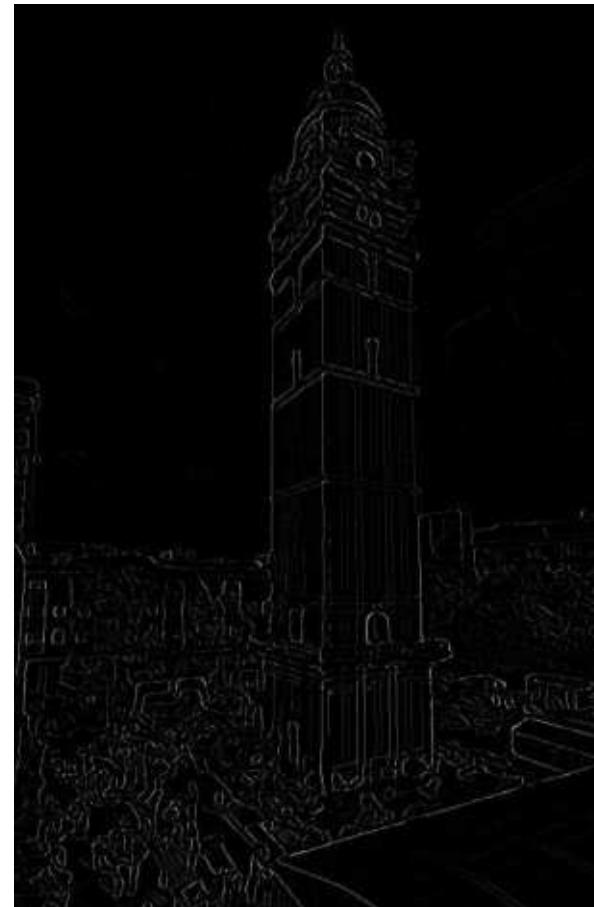
Canny edge detection



Input image



Magnitude

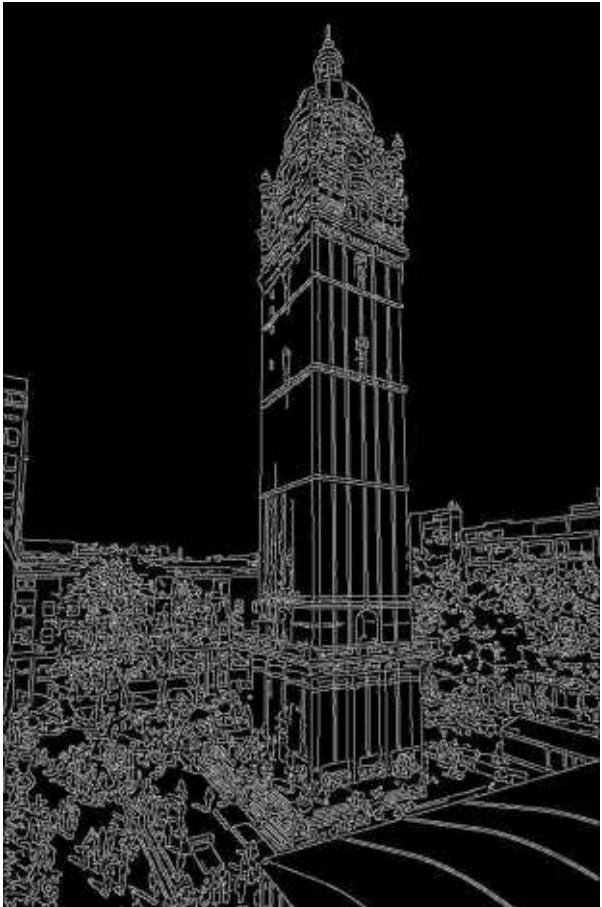


NMS



Hysteresis thresholding

Effect of Gaussian smoothing parameter σ



$\sigma = 1$



$\sigma = 3$



$\sigma = 5$



$\sigma = 7$

Effect of Gaussian smoothing parameter σ

- The choice of σ depends on desired edge detection behavior:
 - A large σ detects large-scale edges and smooths out fine details.
 - A small σ detects fine features.

Canny edge detection

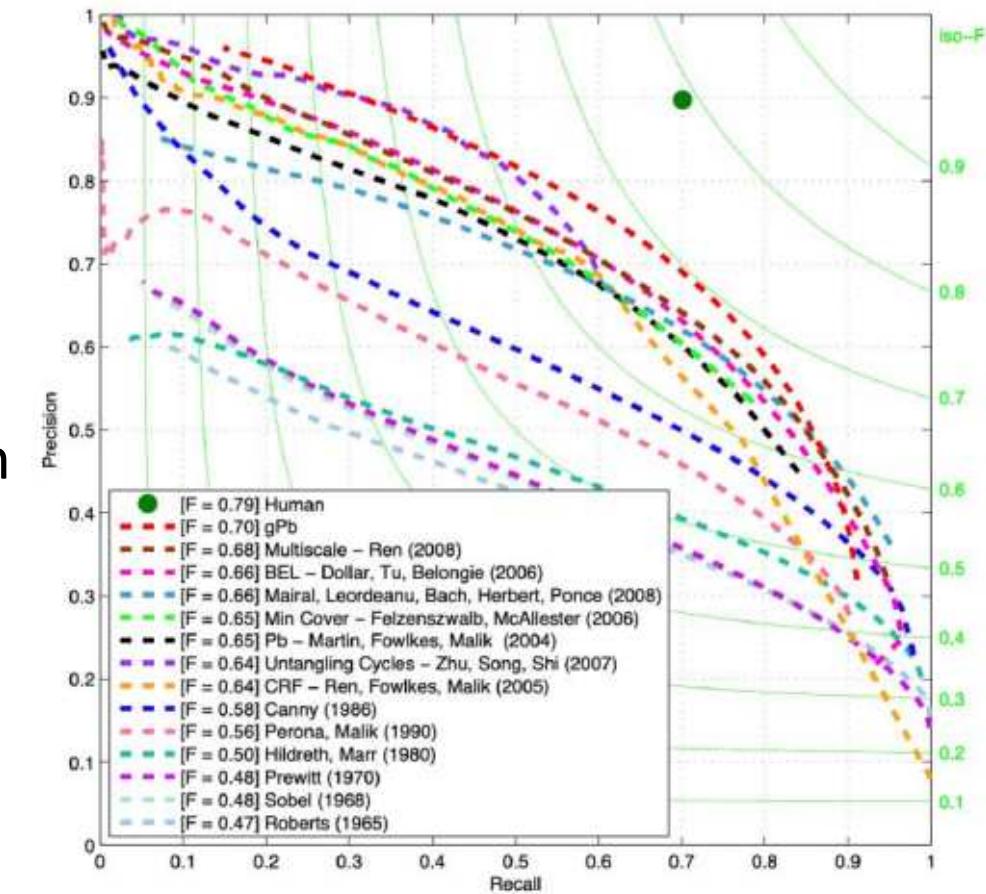
- Let us look again at Canny's initial goals and how he achieves these.
 - **Good detection**
 - Gaussian smoothing to suppress noise (reducing false positives).
 - Hysteresis thresholding to find weak edges (reducing false negatives).
 - **Good localisation**
 - Use gradient orientation and non-maximum suppression to find the location of edges.
 - **Single response**
 - Non-maximum suppression.

Canny edge detection

- One of the most popular edge detection algorithms.
 - First, think of what criteria edges need to satisfy.
 - Then, carefully design the image features (e.g. Gaussian filtering + image gradient) and procedures (e.g. NMS, hysteresis thresholding) to achieve these criteria.

Edge detection

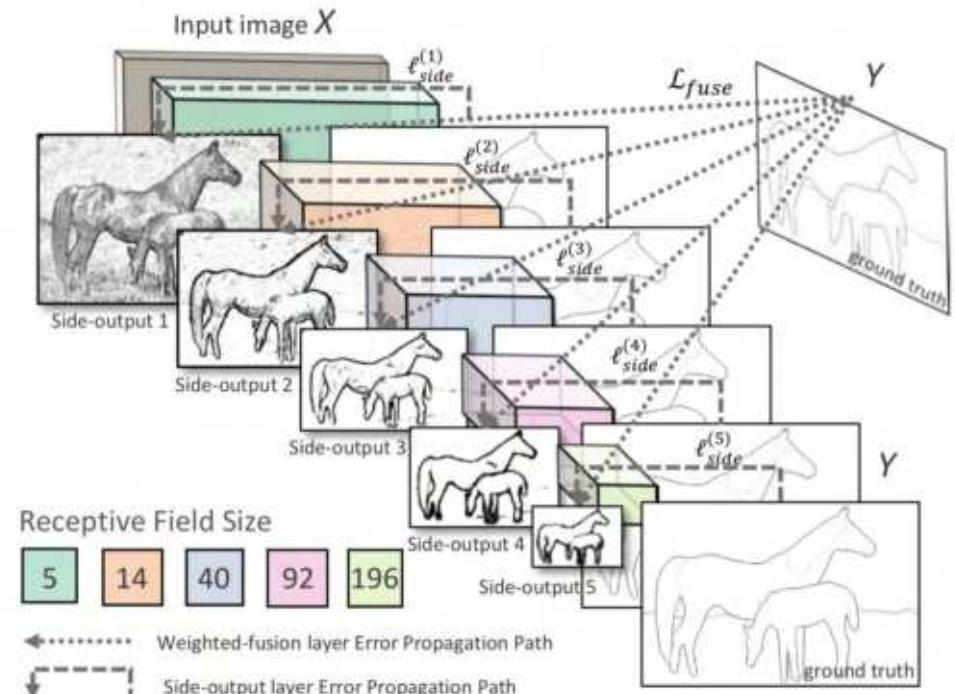
- Decades of efforts have been made for to improve edge detection accuracy.
 - Utilising richer features, e.g. colour, texture.
 - Integrating multi-scale features.
 - Enforcing smoothness.
 - Machine learning, e.g. learning the mapping from image to edge directly from paired data.



Evaluation of contour detection accuracy on BSDS300 dataset. **Green dot:** human performance.

Learning-based edge detection

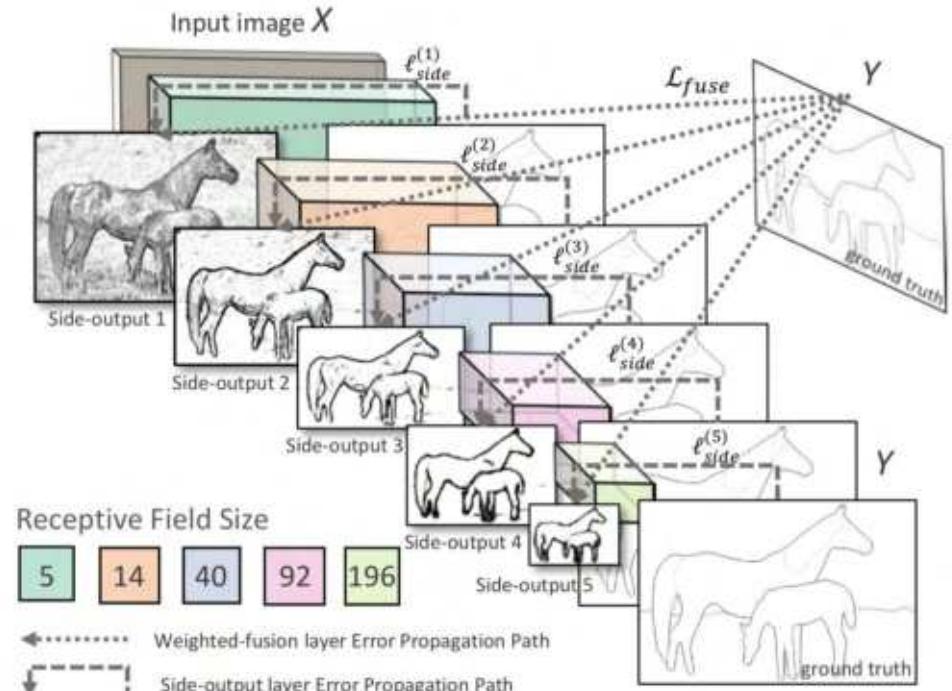
- It assumes that paired data (images x + manually defined edge maps y) are available.
- The problem becomes how to find a model f that maps x to y , i.e. $y = f(x|\theta)$, where θ denotes the model parameters.



The colourful blocks are a machine learning model (e.g. neural network) that maps an input image into an edge map.

Learning-based edge detection

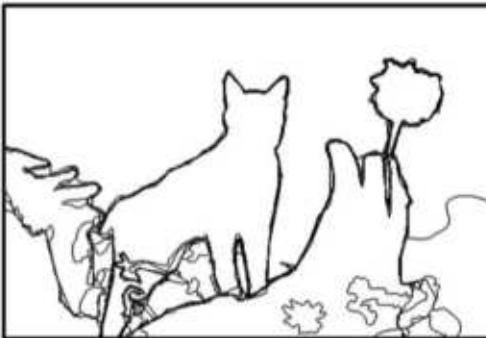
- In this example, the machine learning model integrates features from multiple scales, fusing fine-scale edges with coarse-scale edges to form the final output.
- On the contrary, Canny edge detector only uses a single scale for edge detection, controlled by the Gaussian parameter σ .



The colourful blocks are a machine learning model (e.g. neural network) that maps an input image into an edge map.



(a) original image



(b) ground truth



(c) HED: output



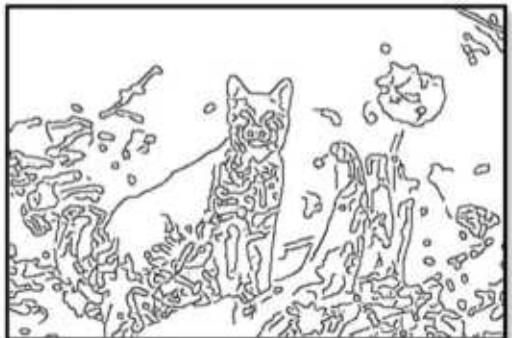
(d) HED: side output 2



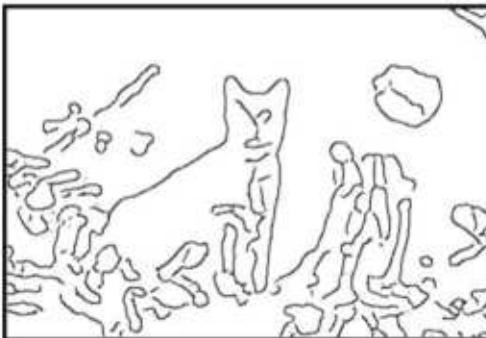
(e) HED: side output 3



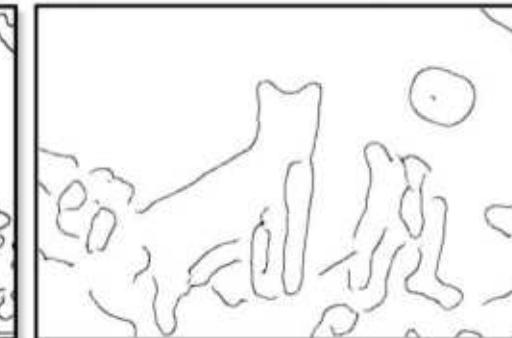
(f) HED: side output 4



(g) Canny: $\sigma = 2$



(h) Canny: $\sigma = 4$

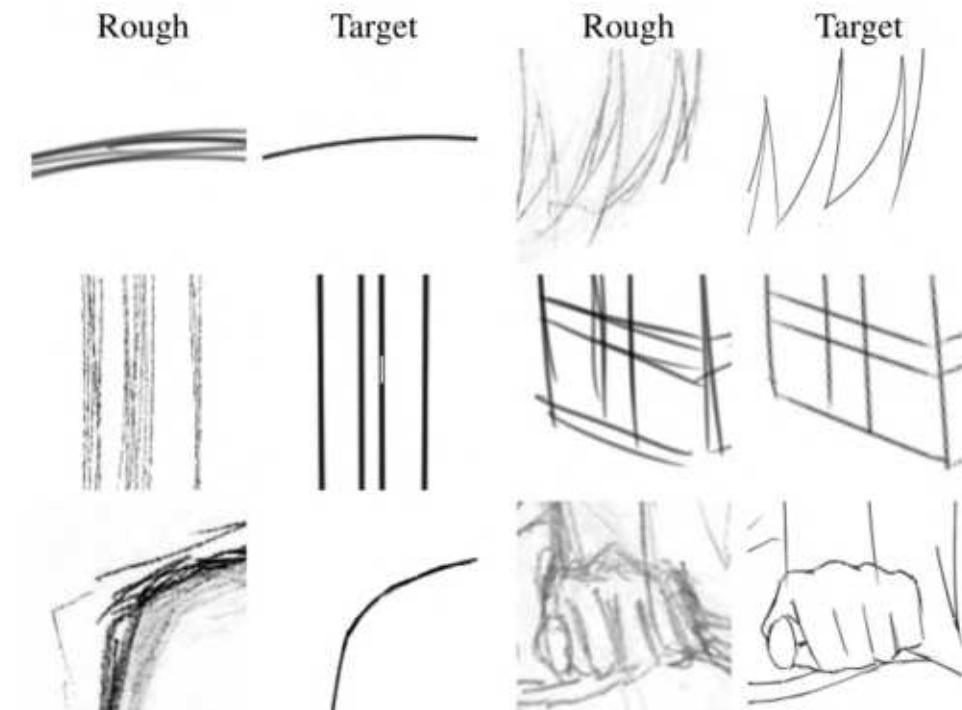


(i) Canny: $\sigma = 8$

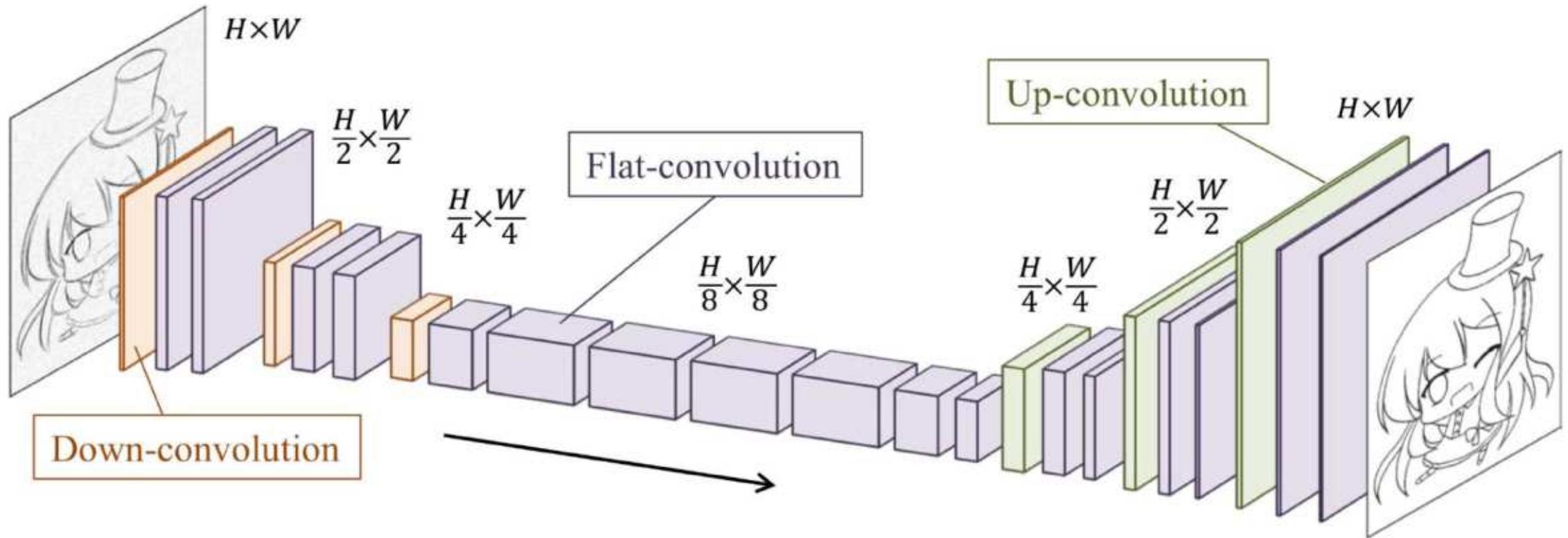
Subfigure (c) shows the edge detection results using the proposed learning-based approach.

Learning-based edge detection

- An interesting application is to learn a mapping from rough sketch to simplified sketch.
- This is not just an edge detection problem. It can not be solved using non-maximum suppression.
- When human do this job, high-level understanding about the sketch is required.
- If paired rough sketch and simplified sketch are available, maybe we can think about a machine learning solution.



Learn simplified sketch from rough sketch.



A convolution neural network is used to analyse image features and generate the simplified sketch.



Comparison of the proposed approach (d) to commercial tools (b, c) for sketch clean-up.

Edge detection

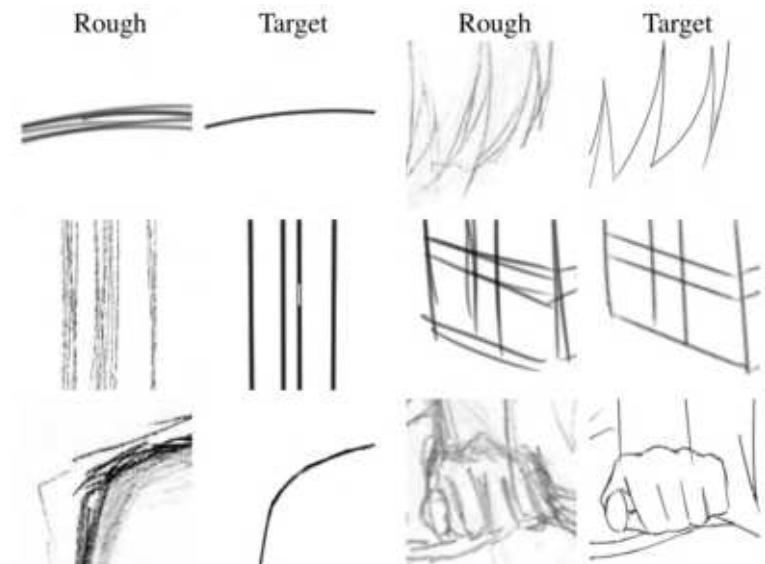
- It is a fundamental problem in image processing and computer vision.
- It aims at identifying points in an image that are **edges**, or where discontinuities occur.
- Two solutions to the problem
 - If we know explicitly the criteria of edge points, we can design computational procedures to satisfy these, like John Canny did.
 - Otherwise, we can collect data which represent what we want to achieve (e.g. manually drawn edge maps) and train a machine learning model to map images to final outputs.
- Hand engineering or model-based methods versus machine learning

Edge detection

- Edges provide important low-level features for both human vision system and computer vision for analysing and understanding images.
- The algorithms that we develop here provide ideas for other detection algorithms, e.g. region detection, centreline detection, interest point detection etc.
- It has various applications, e.g. road detection from aerial images, sketch clean-up etc.



Road detection (Laptev, MVA 2000)



Sketch clean-up (Simo-Serra, SIGGRAPH 2016)

Summary

- Canny edge detection.
 - Gaussian smoothing
 - Calculate gradient magnitude and direction
 - Apply non-maximum suppression (NMS)
 - Perform hysteresis thresholding to find the edges
- Learning-based edge detection.

References

- Sec. 4.2 Edge detection, Sec. 4.3 Lines. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Hough Transform

Dr Wenjia Bai

Department of Computing & Brain Sciences

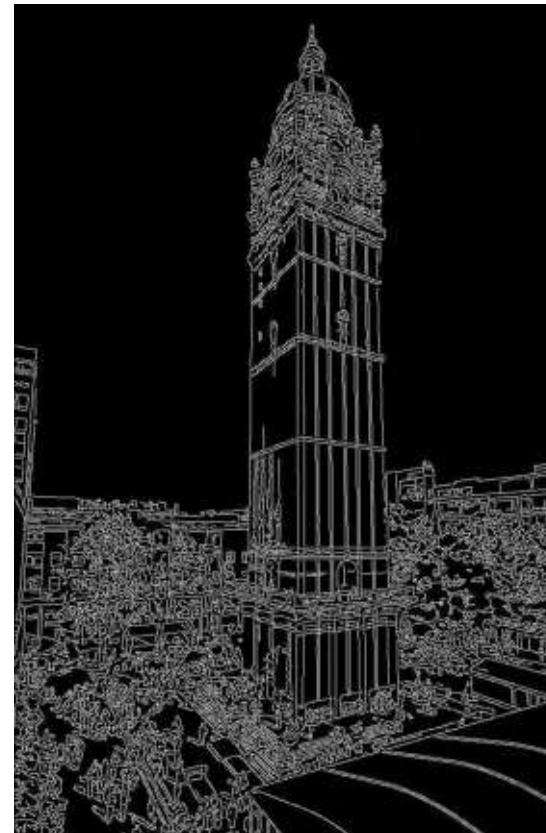
Representation of the world



Pixels



Image gradient



Edge map

$$y = mx + b$$

Math?

Hough transform

- After edge detection, we get a binary edge map, which contains a set of edge points $\{(x_i, y_i) | i = 1, 2, \dots, N\}$.
- If these edge points belong to a line, how can we get a parametric representation of the line?

Line parameterisation

- For example, a line can be represented by $y = mx + b$, i.e. just two parameters m and b .
- This is a much more efficient representation than a lot of edge points.

Line parameterisation

- Slope intercept form

$$y = mx + b$$

- Double intercept form

$$\frac{x}{a} + \frac{y}{b} = 1$$

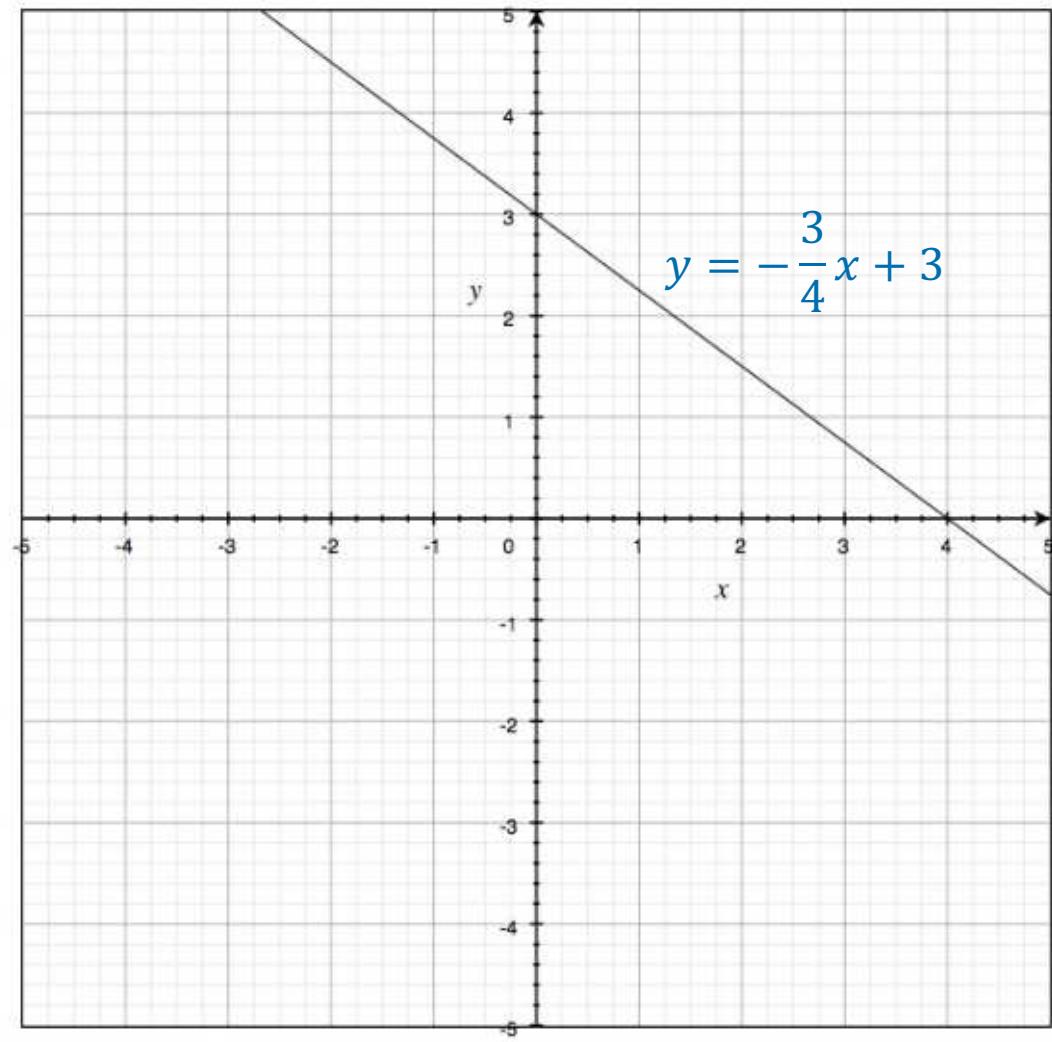
- Normal form

$$x\cos(\theta) + y\sin(\theta) = \rho$$

Slope intercept form

$$y = mx + b$$

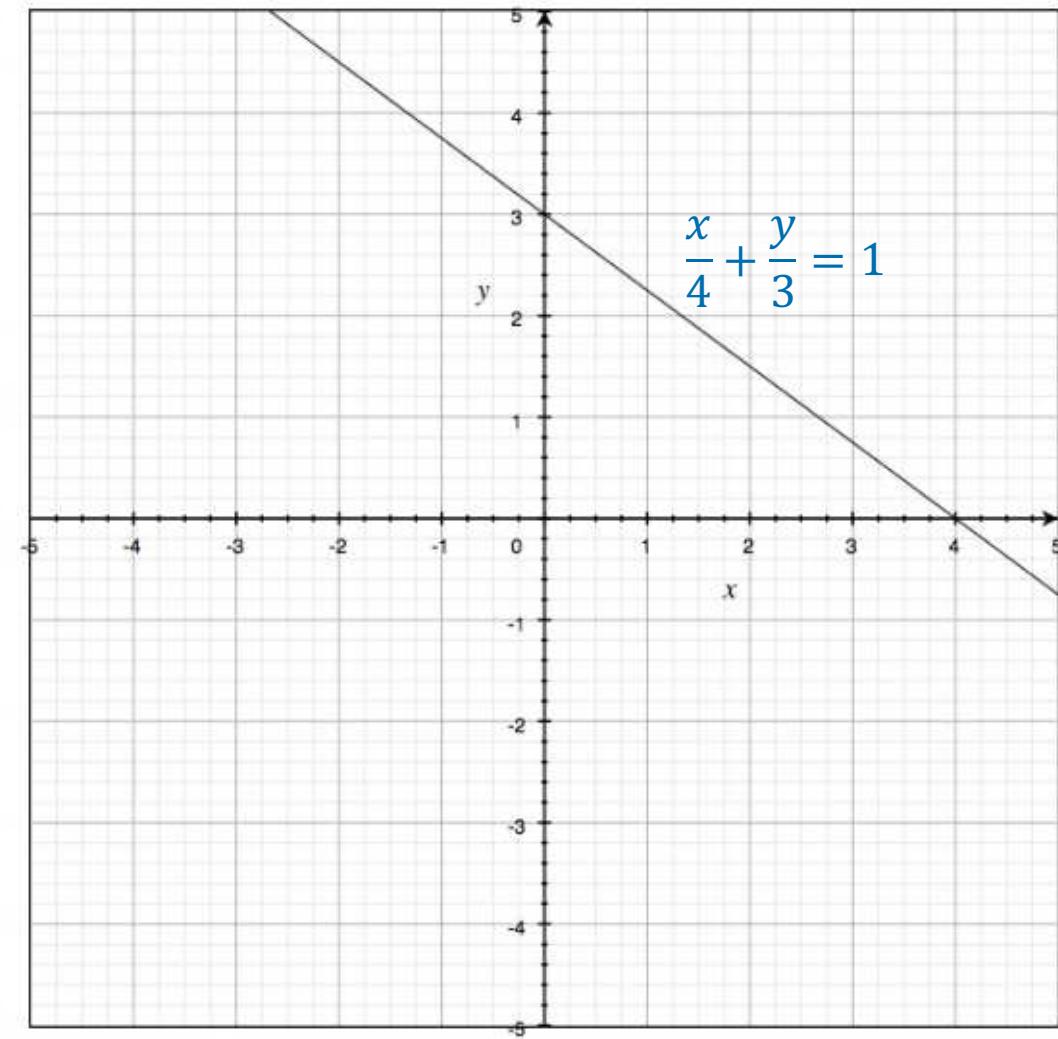
- m : slope
- b : y -intercept



Double intercept form

$$\frac{x}{a} + \frac{y}{b} = 1$$

- a : x -intercept
- b : y -intercept



Normal form

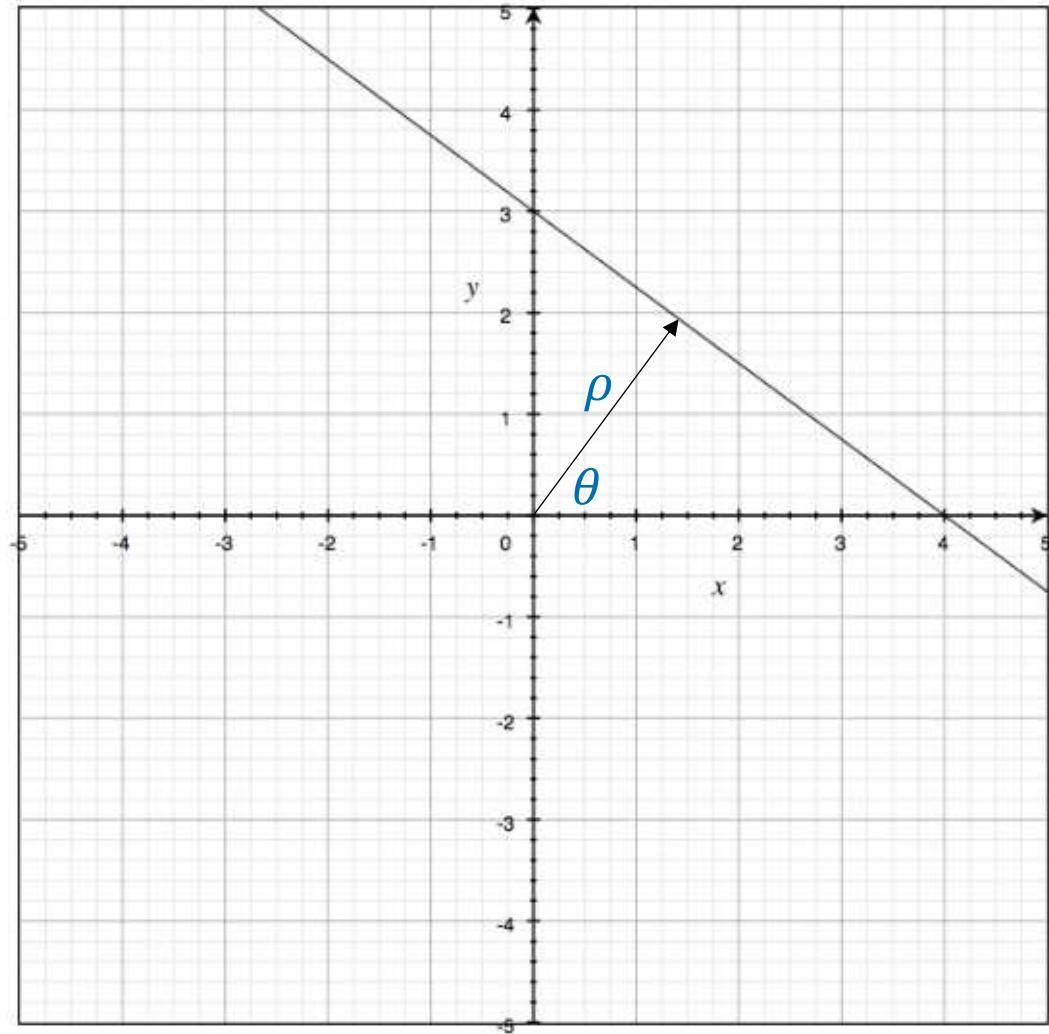
$$x\cos(\theta) + y\sin(\theta) = \rho$$

- θ : angle
- ρ : distance from origin
- Derivation

$$\text{x-intercept } a = \frac{\rho}{\cos(\theta)}, \text{ y-intercept } b = \frac{\rho}{\sin(\theta)}$$

$$\text{Plug into } \frac{x}{a} + \frac{y}{b} = 1$$

We can get the normal form.



Hough transform

- Hough transform is a transform from image space to parameter space (e.g. from an edge map to parameters of a line).
- Its output is a parametric model, given the input edge points.
- The basic idea is that each edge point votes for possible parameters in the parameter space.

Model fitting

- Some of you may have a different idea here, especially if you know optimisation and model fitting.
- One way to solve this problem is to fit a line model onto the edge points.
 - Suppose we have a set of points $(x_1, y_1), (x_2, y_2), \dots$ and we would like to fit a line model $y = mx + b$ to these points.
 - (m, b) can be estimated by minimising the fitting error

$$\min_{m,b} \sum_i [y_i - (mx_i + b)]^2$$

real y of
a point \hat{y} estimated by
our line model

- How will Hough transform solve the problem differently?

Hough transform

- Let us use the slope intercept form for a line model

$$y = mx + b$$



$$b = y - mx$$

- We have edge points in the image space $(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots$
- Each point votes for a line model in the parameter space.
- For example, the first point will vote for $b = y_1 - mx_1$.

Hough transform

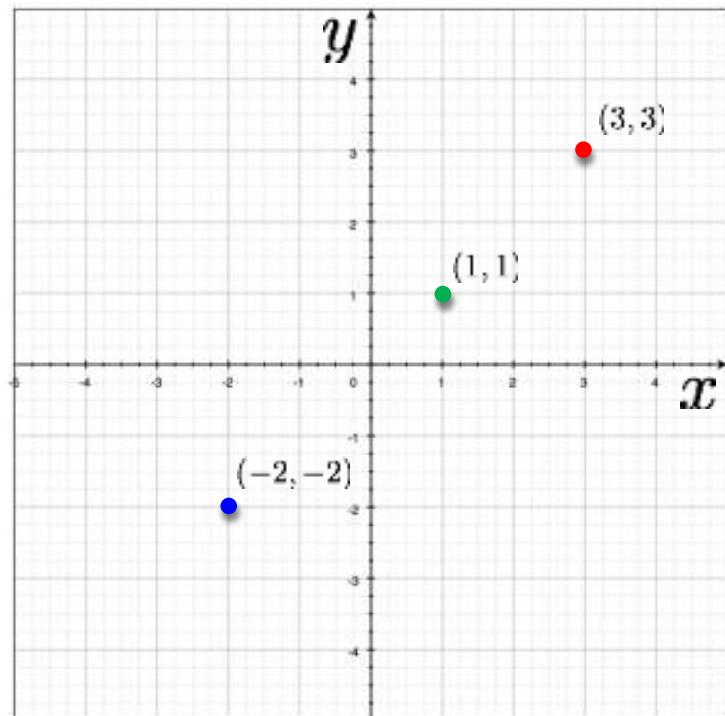
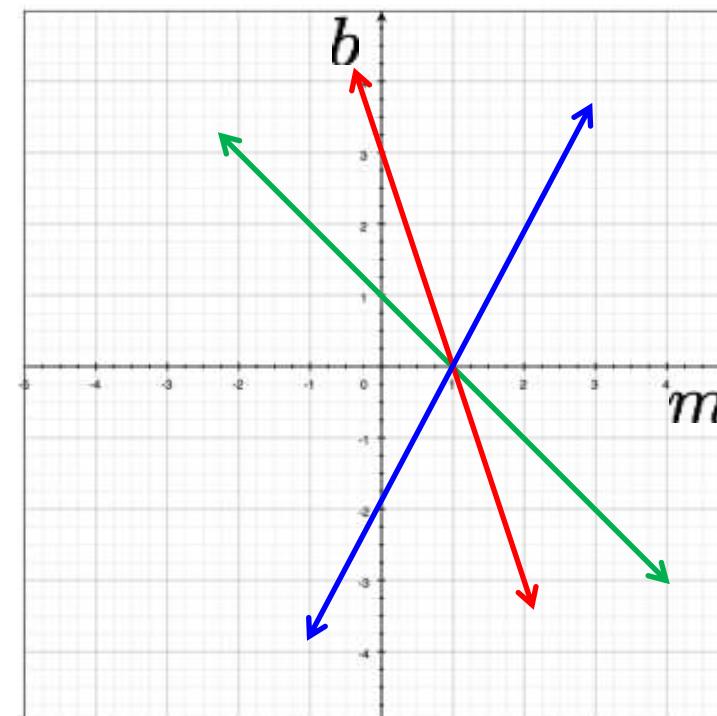


Image space



Parameter space

$$b = y - mx$$

$$b = 3 - 3m$$

$$b = 1 - m$$

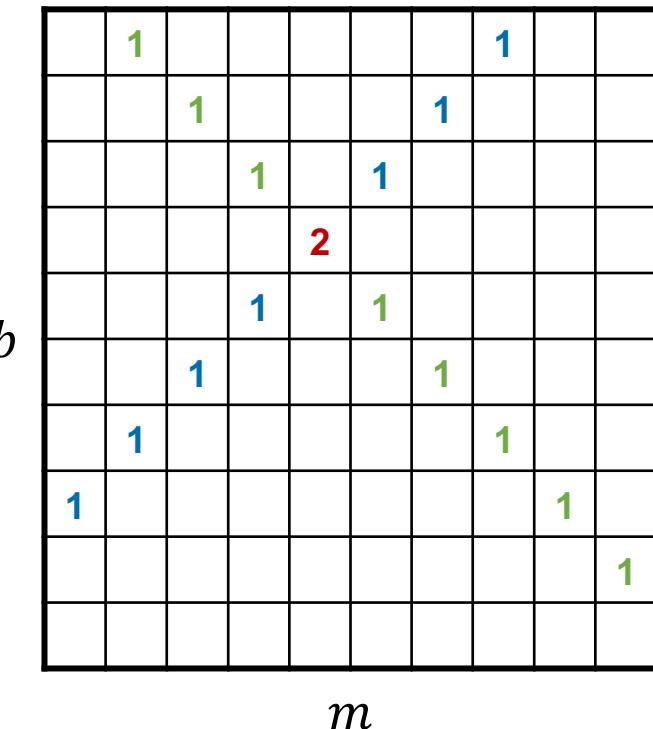
$$b = -2 + 2m$$

Vote result:
 $m = 1, b = 0$

Therefore,
 $y = x$

Hough transform

- In practice, the parameter space is divided into 2D bins.
- Each point increments the vote by 1 in one of the bins.
- One problem with the slope intercept form:
 - The parameter space is too large.
 - $m \in [-\infty, +\infty]$, $b \in [-\infty, +\infty]$
 - We need a lot of bins.



Hough transform

- Solution

- Use the normal form instead

$$x\cos(\theta) + y\sin(\theta) = \rho$$

- Although $\rho \in [-\infty, +\infty]$, at least $\theta \in [0, \pi)$.
 - We can use much fewer bins.
 - By the way, in practice ρ is not infinite either. It is limited by the image size.
- The transform from image space to parameter space will look different.

Hough transform

$$3 \cos \theta + 3 \sin \theta = \rho$$

$$\cos \theta + \sin \theta = \rho$$

$$-2 \cos \theta - 2 \sin \theta = \rho$$

$$x \cos(\theta) + y \sin(\theta) = \rho$$

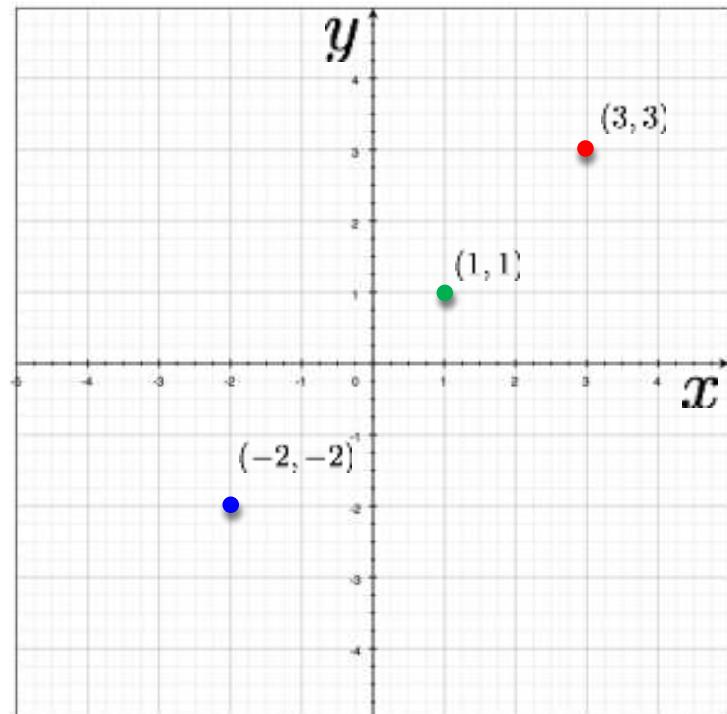
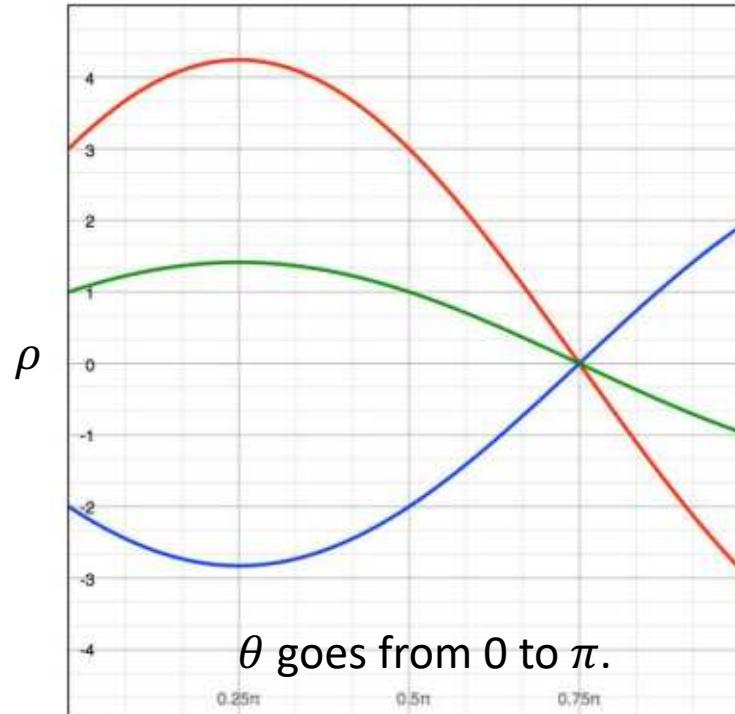


Image space



Parameter space

Vote result:
 $\theta = \frac{3}{4}\pi, \rho = 0$

Therefore,
 $y = x$

Line detection by Hough transform

Algorithm

Initialise the bins $H(\rho, \theta)$ to all zeros.

For each edge point (x, y)

 For θ from 0 to π

 Calculate $\rho = x \cos \theta + y \sin \theta$

 Accumulate $H(\rho, \theta) = H(\rho, \theta) + 1$

Find (ρ, θ) where $H(\rho, \theta)$ is a local maximum and larger than a threshold.

The detected lines are given by $\rho = x \cos \theta + y \sin \theta$.

Line detection by Hough transform

Algorithm

Initialise the bins $H(\rho, \theta)$ to all zeros.

For each edge point (x, y)

 For θ from 0 to π

 Calculate $\rho = x \cos \theta + y \sin \theta$

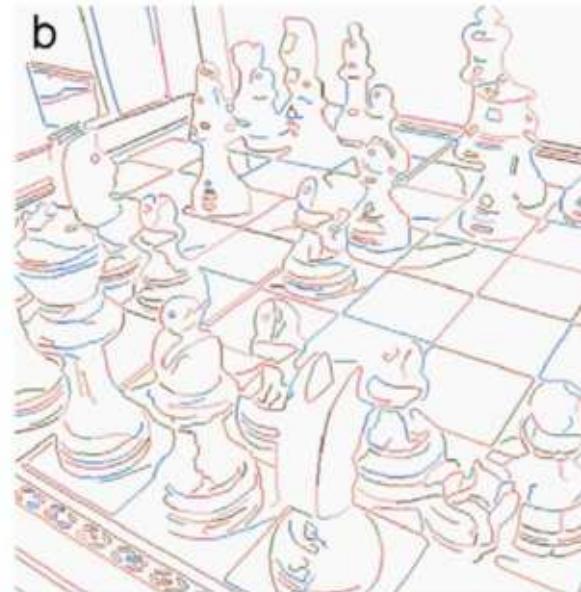
 Accumulate $H(\rho, \theta) = H(\rho, \theta) + 1$

Find (ρ, θ) where $H(\rho, \theta)$ is a **local maximum** and larger than a **threshold**.

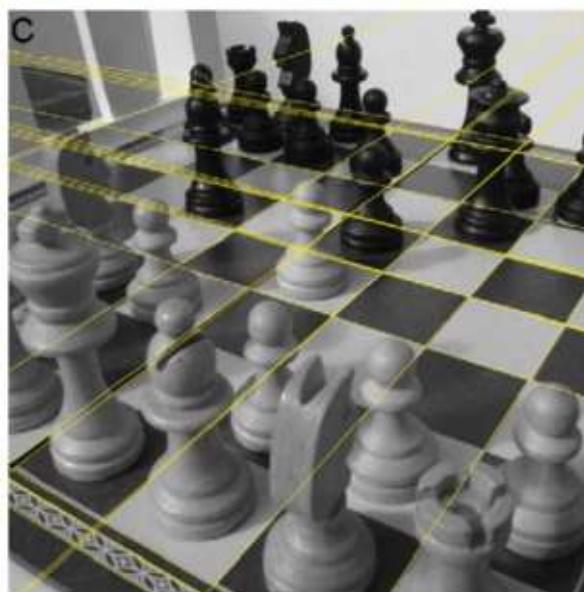
The detected lines are given by $\rho = x \cos \theta + y \sin \theta$.

- Why local maximum?
 - Similar as non-maximum suppression in edge detection.
- Why thresholding?
 - A few random points would not lead to a line being detected.

Input image



Canny edge detection



Lines detected by Hough transform

Hough transform

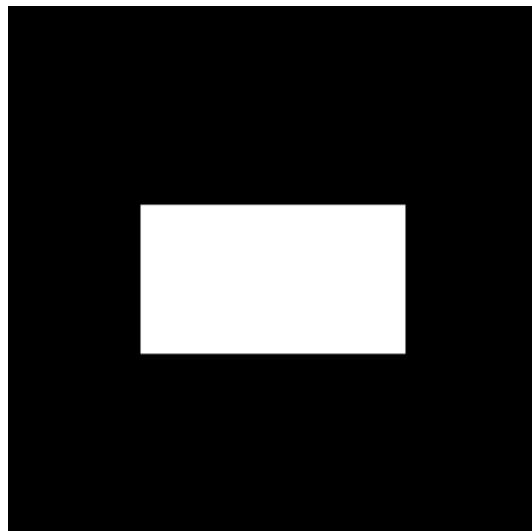
- In model fitting, (m, b) are estimated by minimising the fitting error

$$\min_{m,b} \sum_i [y_i - (mx_i + b)]^2$$

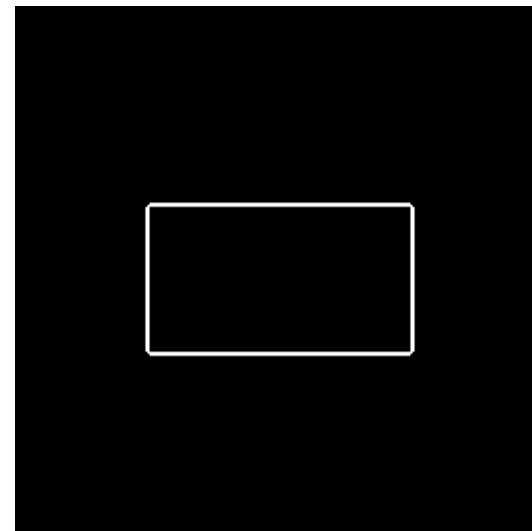
- Only one line will be detected.
- On the contrary, Hough transform can simultaneously detect multiple lines, as long as they are local maxima above a threshold.

Hough transform

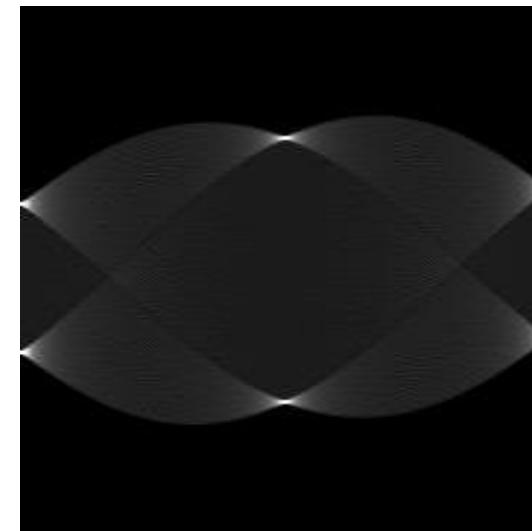
- It can detect multiple lines simultaneously.



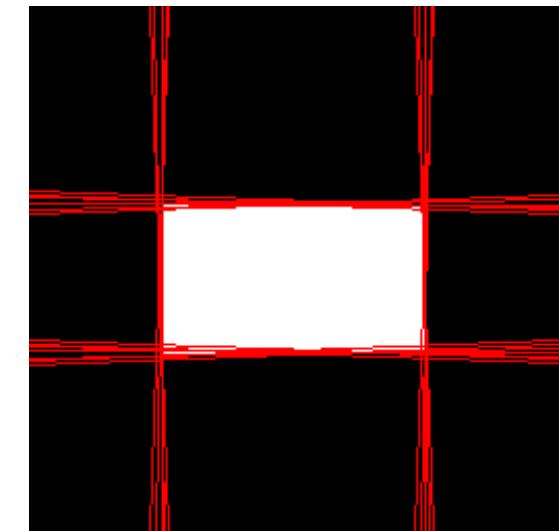
Input image



Edge map



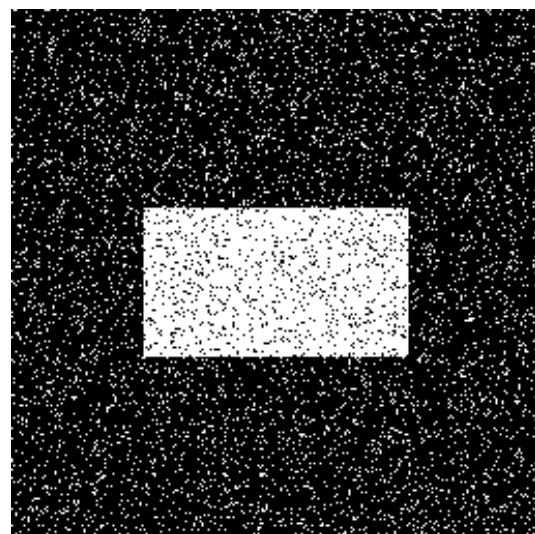
Parameter space



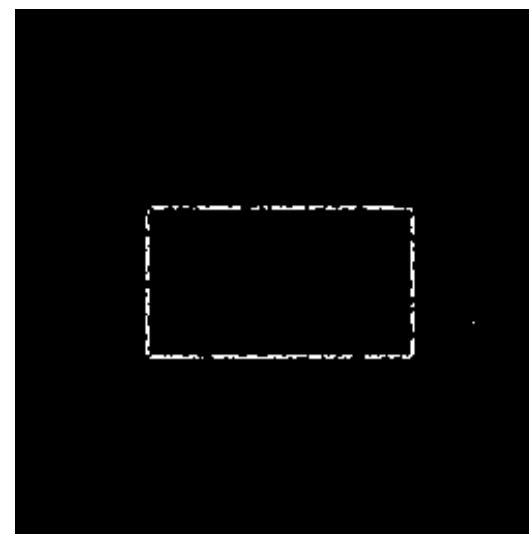
Detected lines

Hough transform

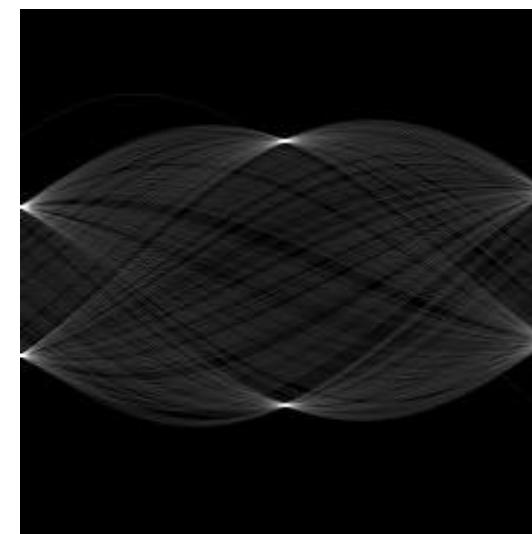
- It is robust to noise.
 - Broken edge points can still vote and contribute to line detection.



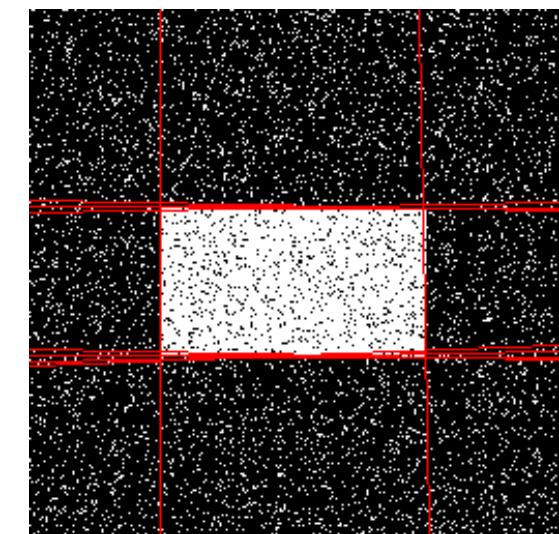
Input image



Edge map



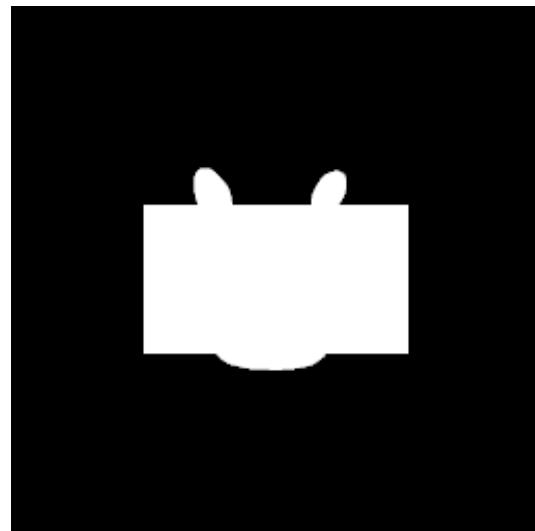
Parameter space



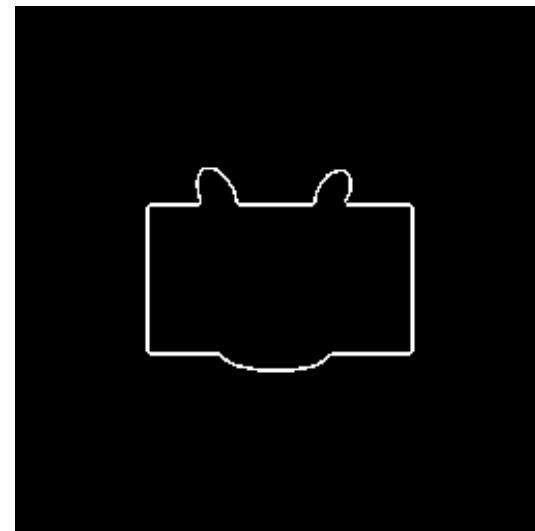
Detected lines

Hough transform

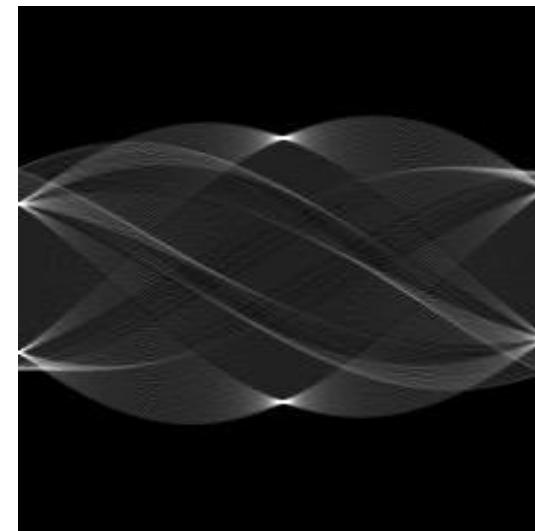
- It is robust to object occlusion.
 - The remaining edge points still contribute to line detection.



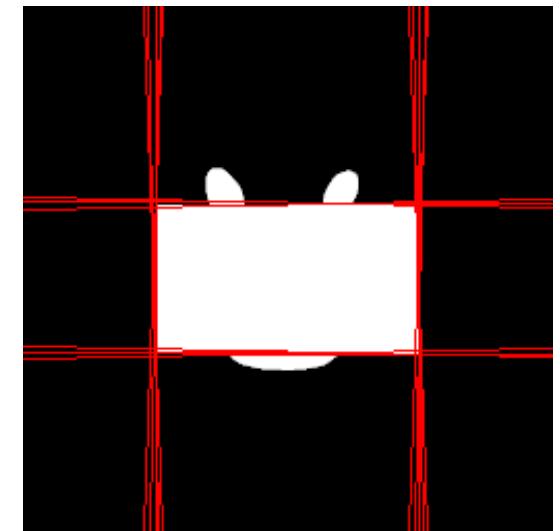
Input image



Edge map



Parameter space



Detected lines

Hough transform

- It is not just for detecting lines, but also for other shapes, such as circles.

- We can parameterise a circle as,

$$(x - a)^2 + (y - b)^2 = r^2$$

- The image space (x, y) is transformed to the parameter space (a, b, r) .
- This is a very large search space (a lot of bins).
- However, if we know the radius r , it would be easier.

Circle detection

$$(x - a)^2 + (y - b)^2 = r^2$$

- If the radius r is known, then for each edge point (x, y) , we only need to vote for possible values of (a, b) .
- It is a circle in the parameter space $H(a, b)$.

$$(a - x)^2 + (b - y)^2 = r^2$$

Circle detection

$$(a - 3)^2 + (b - 2)^2 = 1$$

$$(a - 2)^2 + (b - 1)^2 = 1$$

$$(a - 1)^2 + (b - 2)^2 = 1$$

$$(a - x)^2 + (b - y)^2 = 1, \text{ assuming } r = 1$$

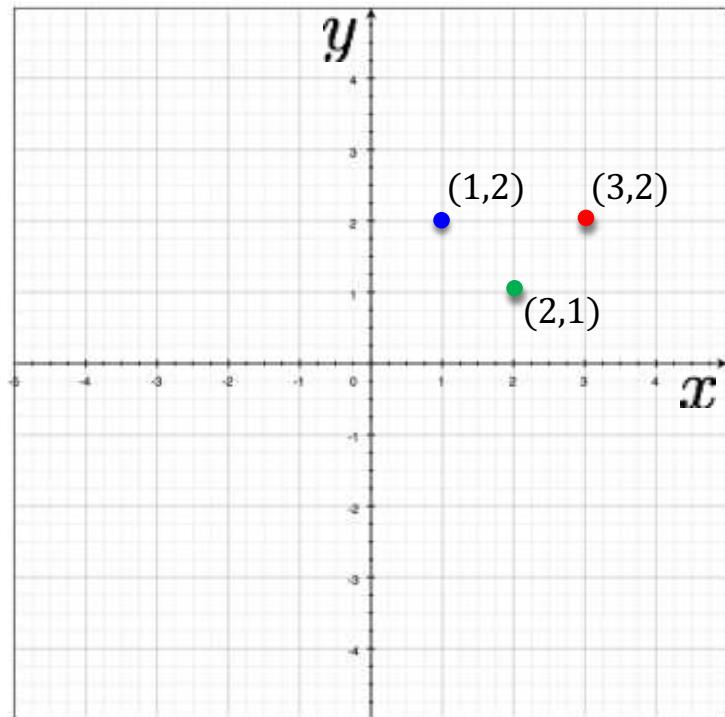
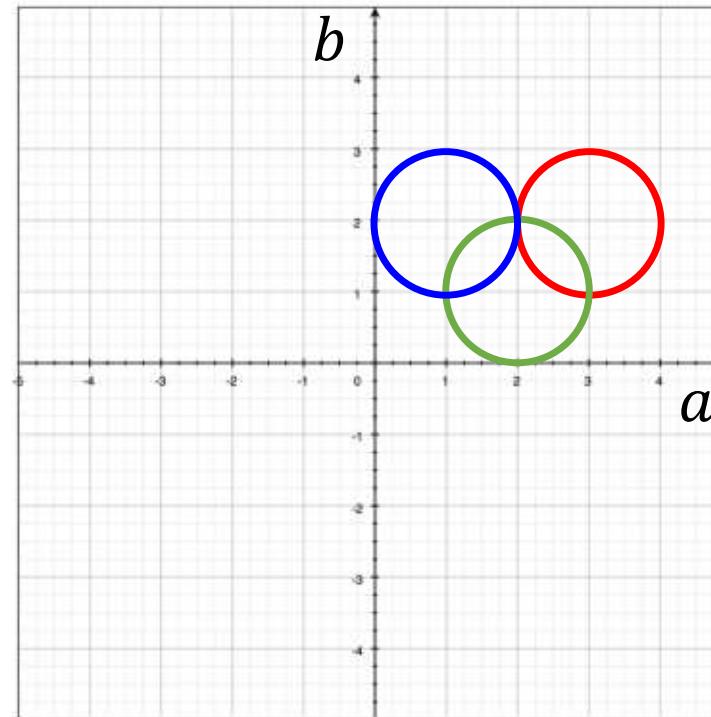


Image space



Parameter space

Vote result:
 $a = 2, b = 2$

Circle detection

$$(x - a)^2 + (y - b)^2 = r^2$$

- If the radius r is unknown, then it is a 3D parameter space $H(a, b, r)$.
- We set a range for the radius r .

For each $r \in [r_{min}, r_{max}]$

For each edge point (x, y)

We vote for possible values of (a, b) and accumulate $H(a, b, r)$.

- For example, we can start from $r = 1$ pixel to 10 pixels, each time increasing by 1 pixel.

Circle equations

- Standard form

$$(x - a)^2 + (y - b)^2 = r^2$$

- Parametric form using trigonometric functions

$$x = a + r \cdot \cos \theta$$

$$y = b + r \cdot \sin \theta$$

- This form gives us some ideas for acceleration.
- If we know the angle θ (direction) from the edge point (x, y) to the circle centre (a, b) , we can more accurately vote in the parameter space.

Circle detection

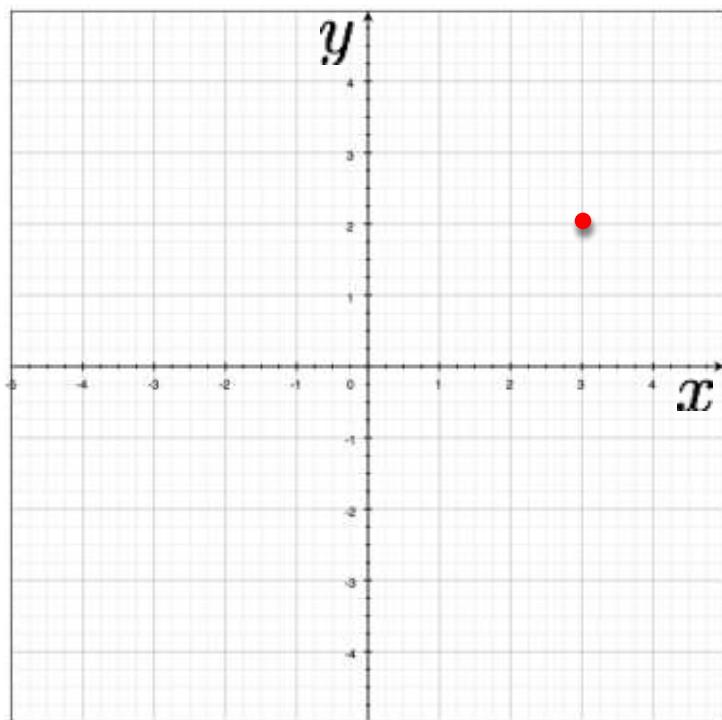
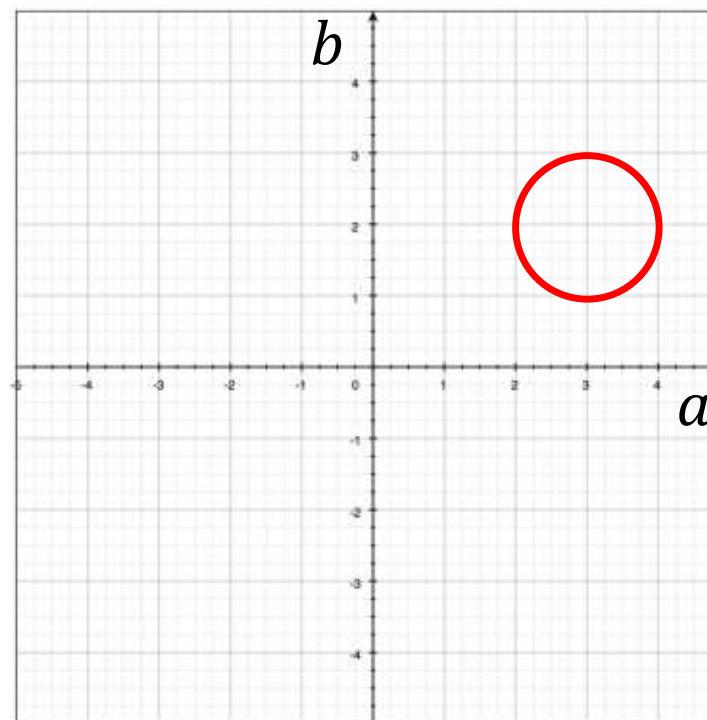


Image space



Parameter space

$$a = x - r \cdot \cos \theta$$
$$b = y - r \cdot \sin \theta$$

If we do not know θ , we vote to a whole circle.

Circle detection

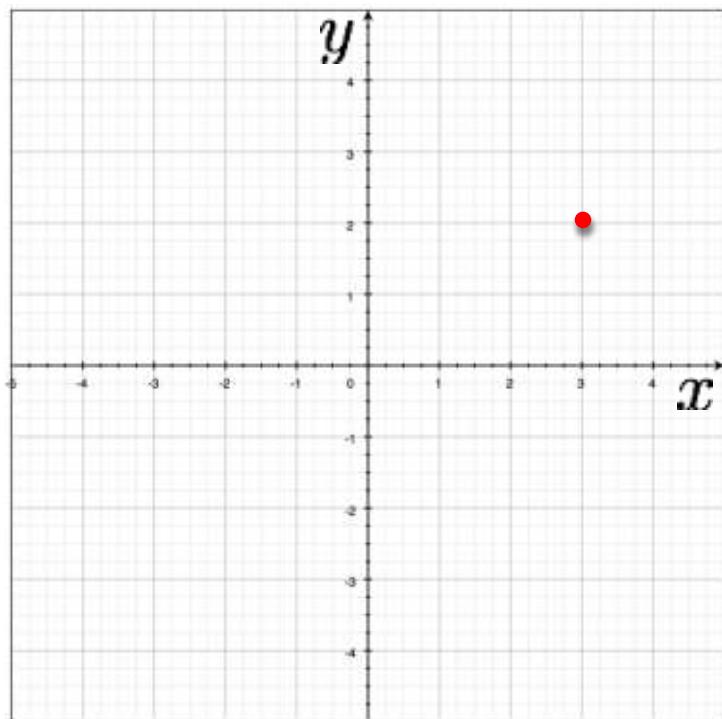
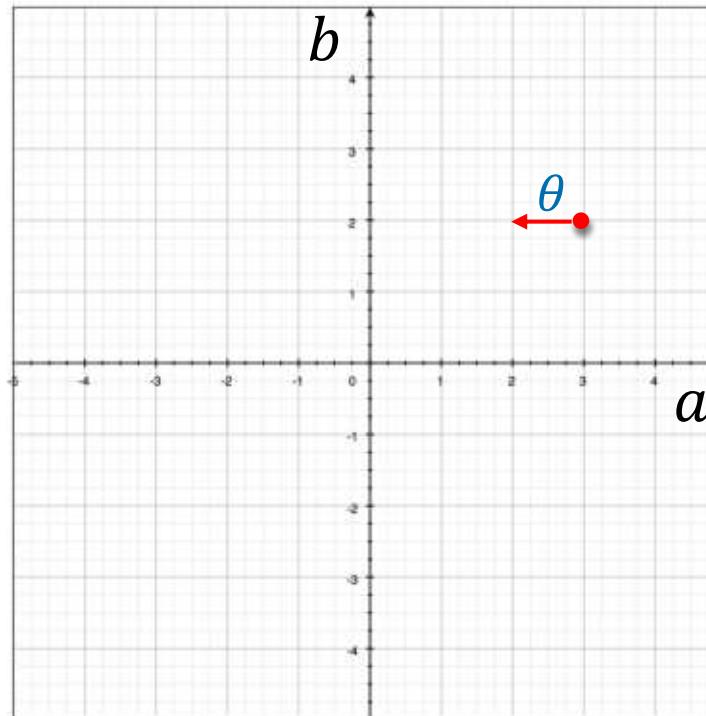


Image space



Parameter space

$$a = x - r \cdot \cos \theta$$
$$b = y - r \cdot \sin \theta$$

If we know θ , we will only vote along this direction.

Circle detection

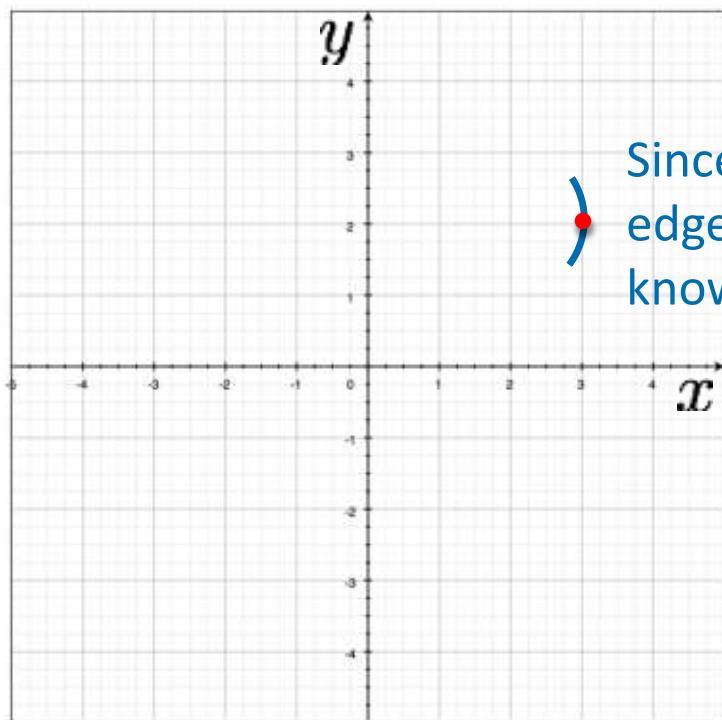
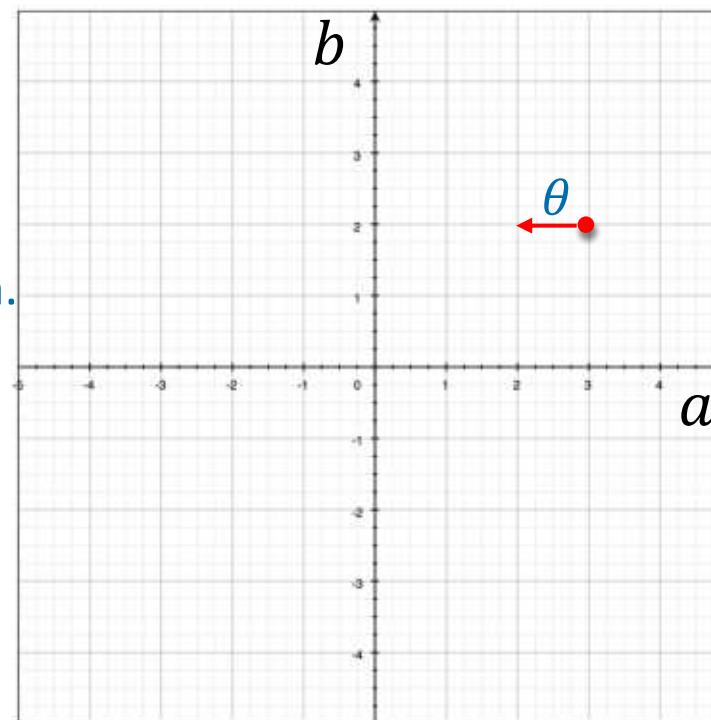


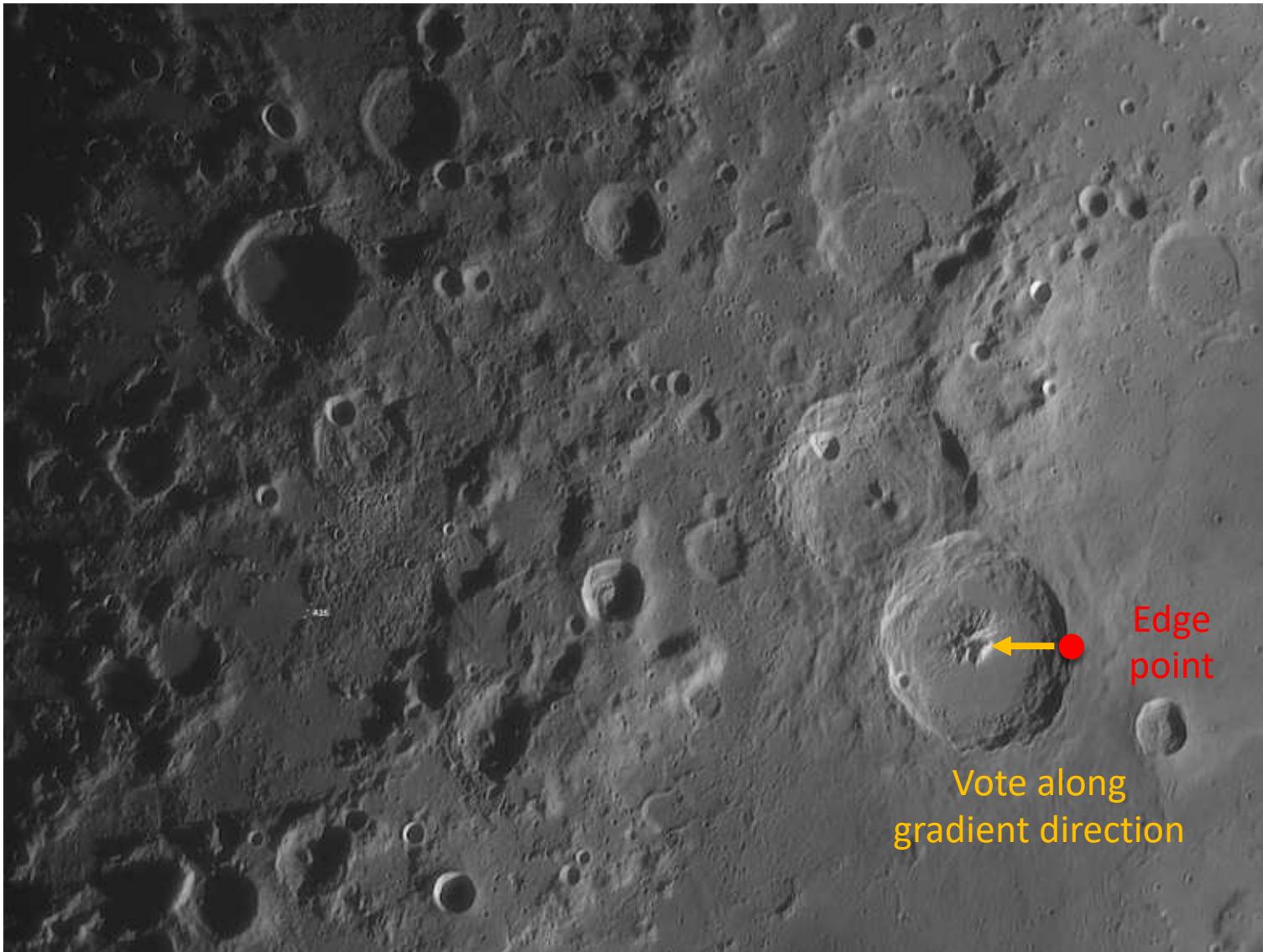
Image space

$$a = x - r \cdot \cos \theta$$
$$b = y - r \cdot \sin \theta$$



Parameter space

If we know θ , we will only vote along this direction.



Edge
point

Vote along
gradient direction

Circle detection

- Parametric form using trigonometric functions

$$a = x - r \cdot \cos \theta$$
$$b = y - r \cdot \sin \theta$$

- The edge point (x, y) comes from an edge detection algorithm, so we know its direction θ .
- This narrows down our voting area in the parameter space $H(a, b, r)$. We simply move long θ (or opposite θ) for a distance of r .

Circle detection by Hough transform

Algorithm

Initialise the bins $H(a, b, r)$ to all zeros.

For each possible radius $r \in [r_{min}, r_{max}]$

 For each edge point (x, y)

 Let θ to be gradient direction, or opposite gradient direction

 Calculate $a = x - r \cdot \cos \theta, b = y - r \cdot \sin \theta$

 Accumulate $H(a, b, r) = H(a, b, r) + 1$

Find (a, b, r) where $H(a, b, r)$ is a local maximum and larger than a threshold.

The detected circles are given by $x = a + r \cdot \cos \theta, y = b + r \cdot \sin \theta$.

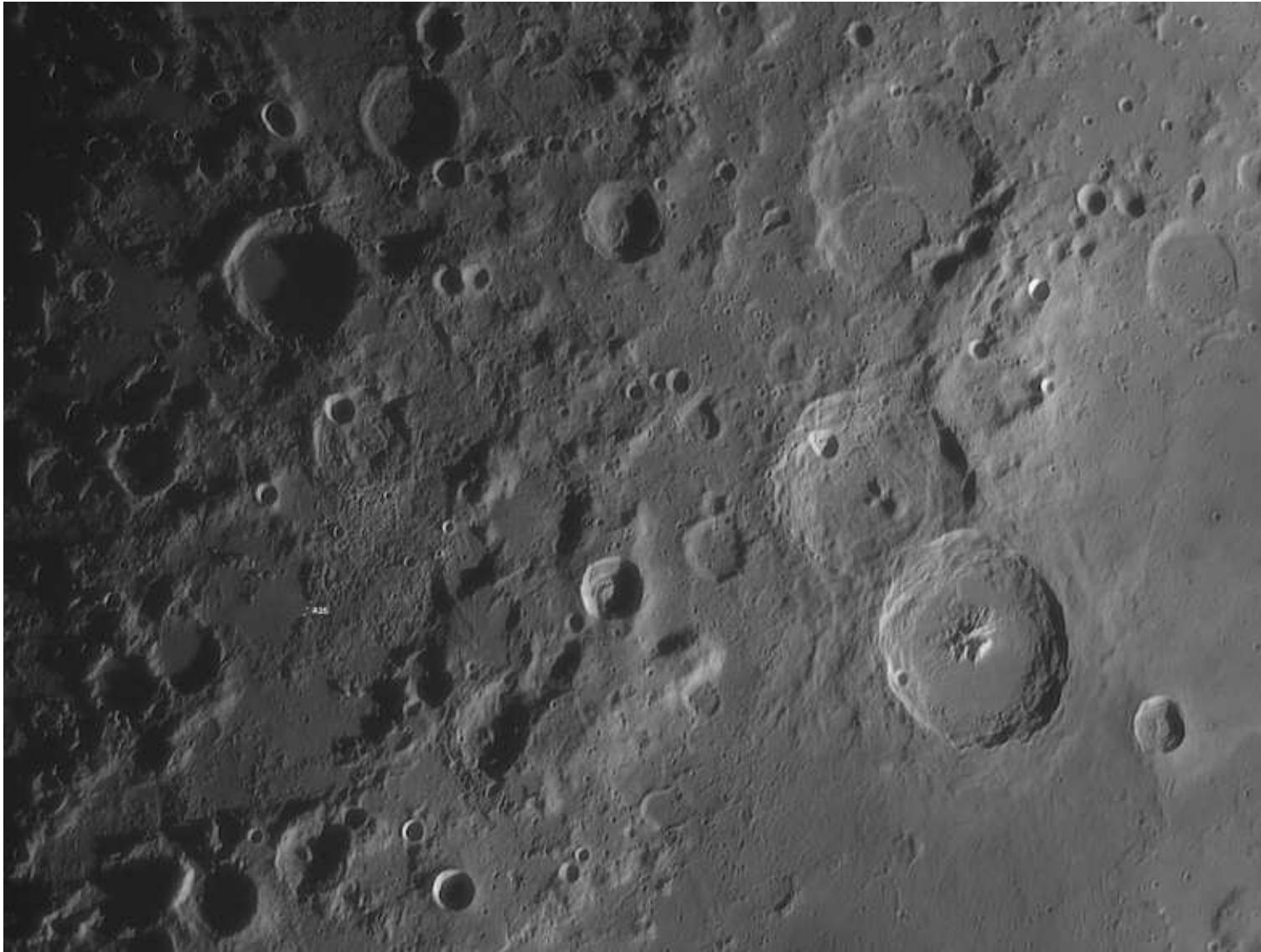
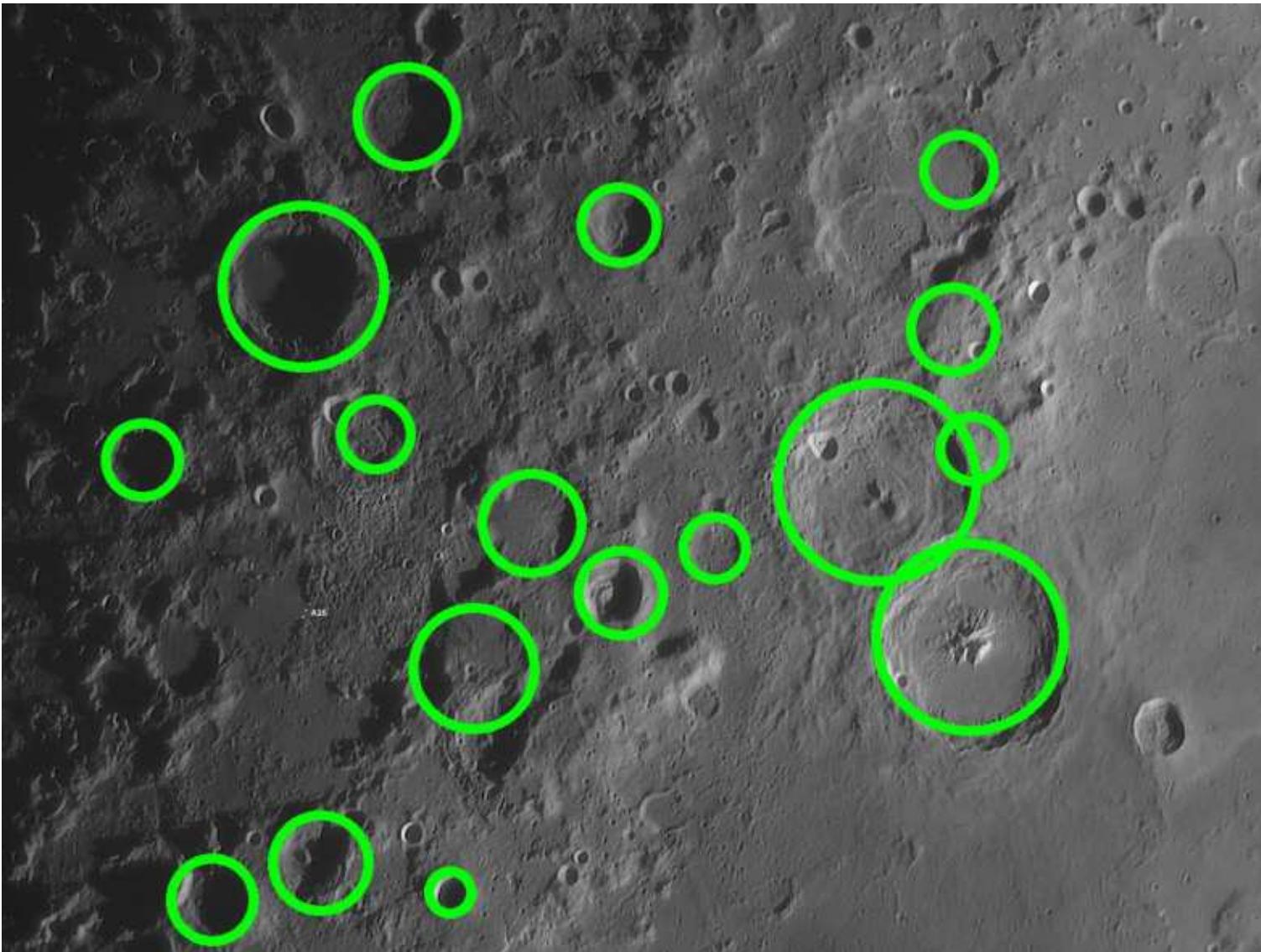


Image of the moon surface, where Apollo 16 landed.



Circles detected by Hough transform

Hough transform

- Advantages
 - It detects multiple instances.
 - Robust to image noise.
 - Robust to occlusion.
- Limitations
 - The computational complexity can be high. For each edge point, we need to vote to a 2D or even 3D parameter space.
 - We need to carefully set some parameters, such as the parameters for the edge detector, the threshold for the accumulator or the range of circle radius.

Hough transform

- Apart from lines and circles, we can also use Hough transform for detecting other shapes.

- Ellipses

$$\frac{(x - c)^2}{a^2} + \frac{(y - d)^2}{b^2} = 1$$

- Planes in a 3D space

$$\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$$

- Other shapes that can be analytically represented.

Hough transform

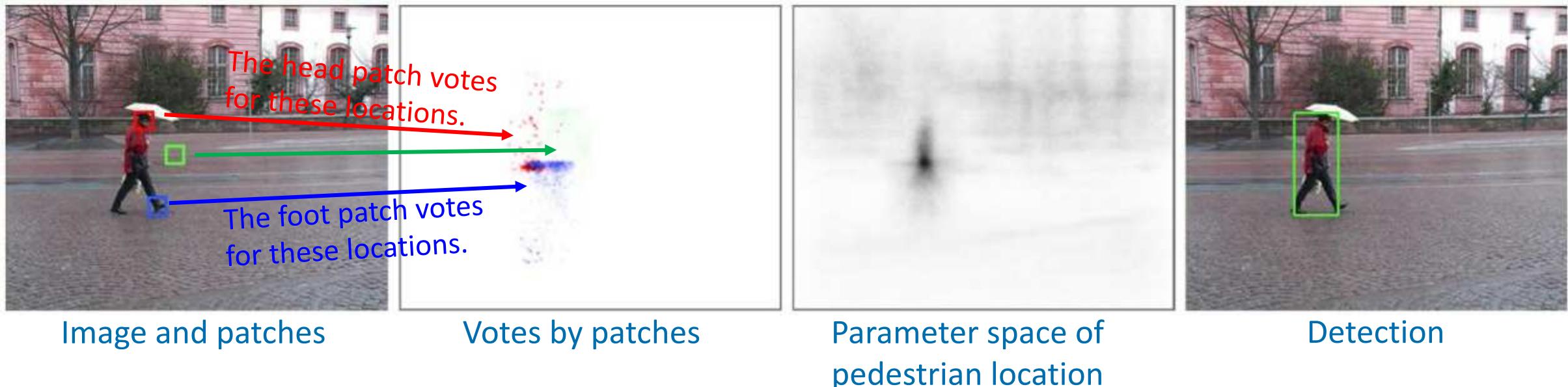
- When we vote in the parameter space, we can add some weights.
- Instead of using equal vote for each edge point, the vote can be weighted by the gradient magnitude, so stronger edge points get higher weights.

Generalise the Hough transform idea

- In Hough transform, our aim is to detect some shapes or objects.
- There are two spaces, the image space and the parameter space (Hough space).
 - If the shape can be described by some parameters using an analytical equation, we use this equation to perform voting in the parameter space.
 - If it is not a simple shape without an analytical equation, as long as we have a model to describe it, we can still vote.

Hough forests for pedestrian detection

- The vote is performed by a machine learning model (random forest).
- This model predicts a displacement vector from the patch centre, given the image feature of the patch.



Hough transform

- Hough transform is an image analysis technique, which finds shapes or patterns by a voting procedure.
 - Line detection
 - Circle detection
 - General shapes or patterns

References

- Sec. 4.3.2 Hough transforms. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Interest Point Detection I

Dr Wenjia Bai

Department of Computing & Brain Sciences

In this lecture

- What are interest points?
- Why are we interested in them?
- How do we detect them?

Interest points

- Literally, it means those points that we are interested in and that are useful for subsequent image processing and analysis.
 - Image classification
 - Image matching
 - Image retrieval
 - ...
- They may also be called keypoints, landmarks, low-level features, where local image structure is rich.

Facial image analysis

- How do we classify facial images or analyse moods?



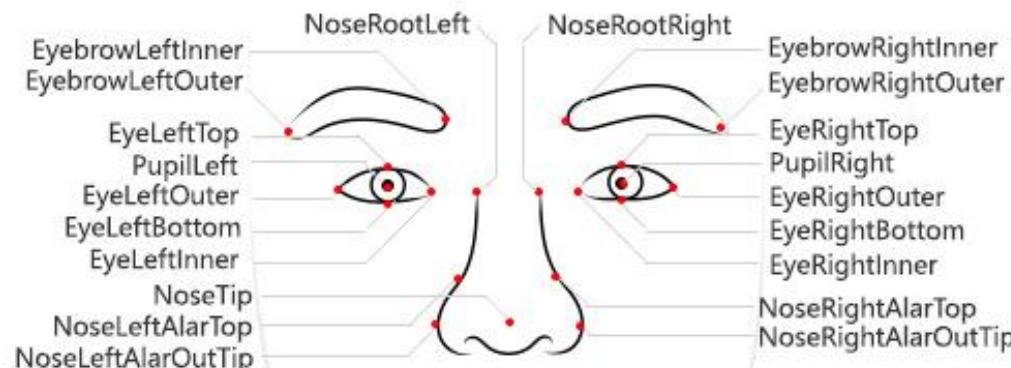
Blackadder



Mr Bean

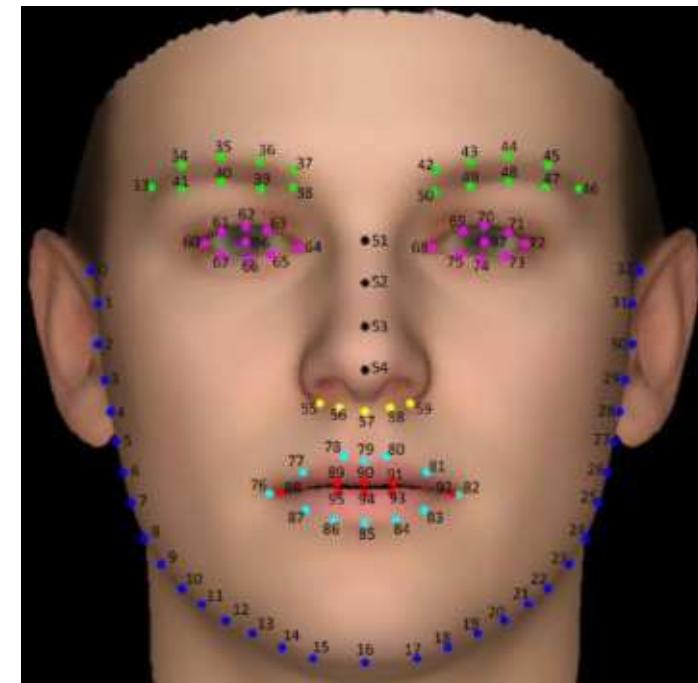
Facial image analysis

- We can define landmarks, which are most representative of a face.



27 landmarks

Copyright (c) Microsoft. All rights reserved



More landmarks

Facial image analysis

- The location of these landmarks can be detected and used for identification, mood analysis etc.

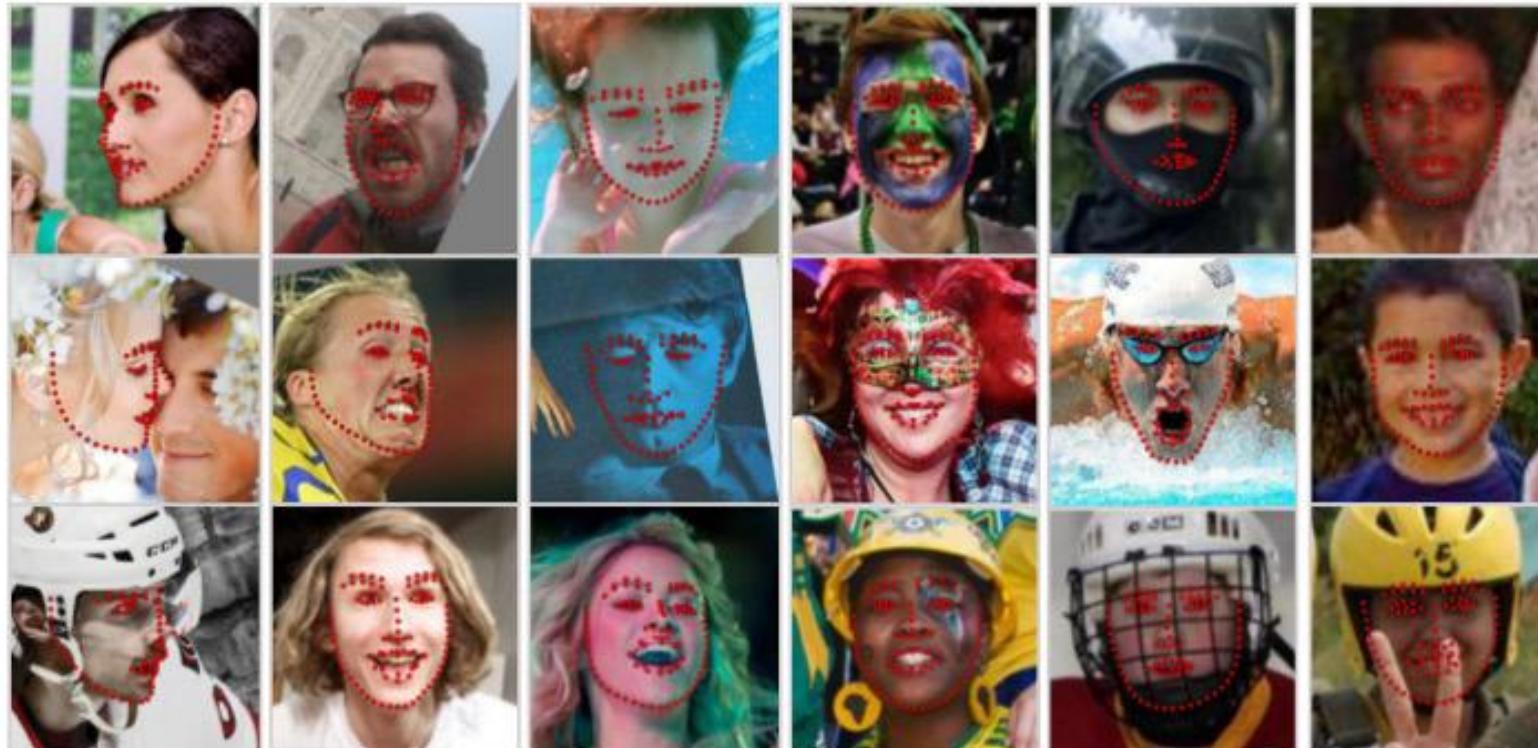


Image matching

- How to establish the correspondence between two images?



Image matching

- Sometimes also called image alignment or registration.
- How do we find spatial correspondence between two images?
 - By pixels: using all information
 - By interest points: using very little but important information

Matching using all pixels

- Image registration algorithms often work by optimising a similarity metric based on all pixels,

$$\max_T \text{Similarity}(\text{Image}_A, \text{Image}_B(T))$$

where T denotes spatial transformation.

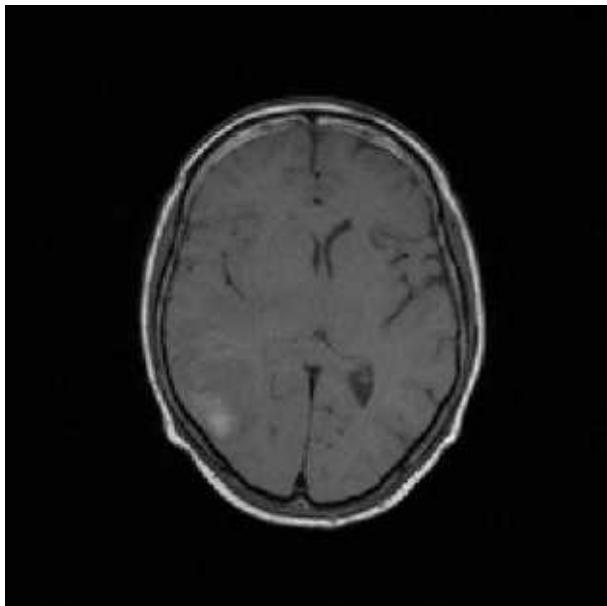


Image A (MRI)

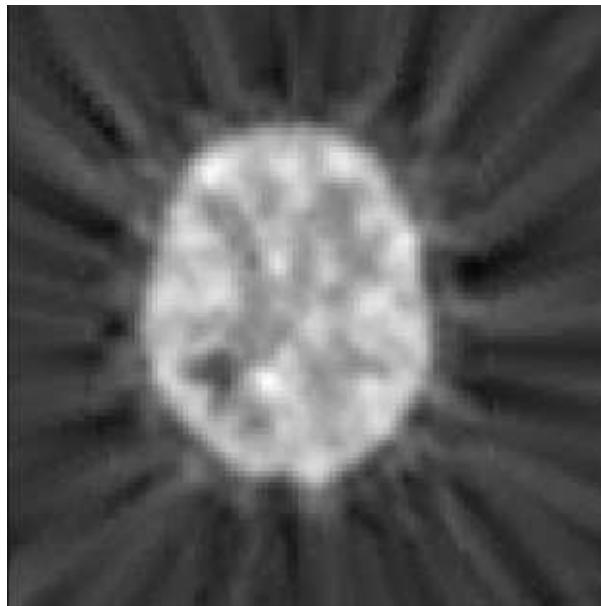
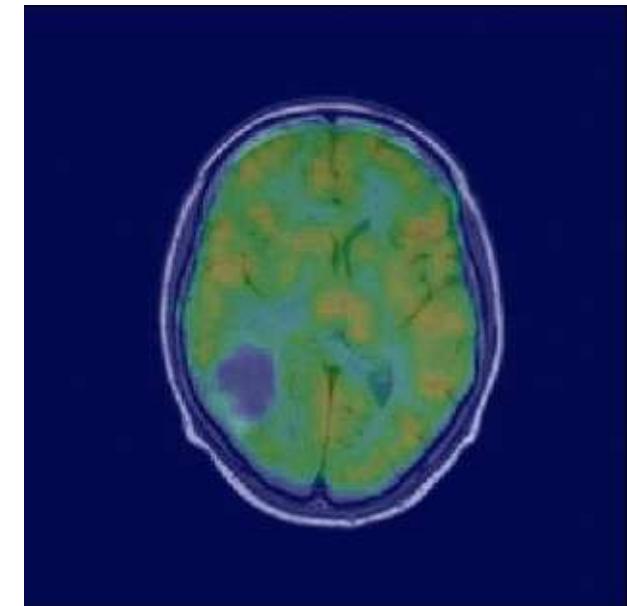


Image B (PET)

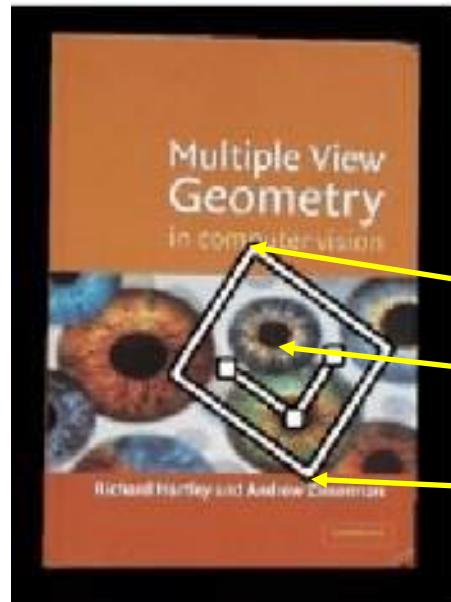


Aligned images

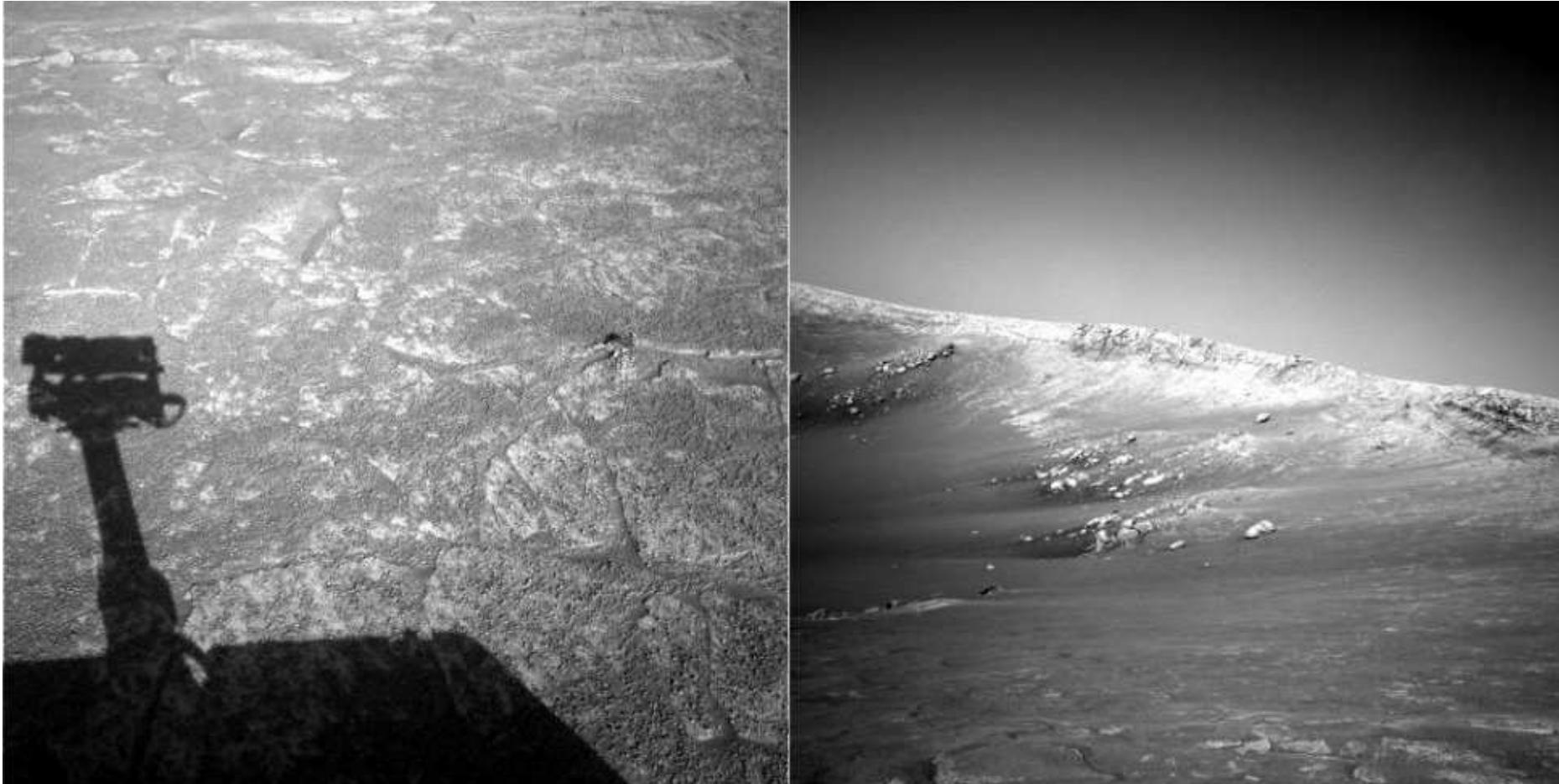
Matching by interest points



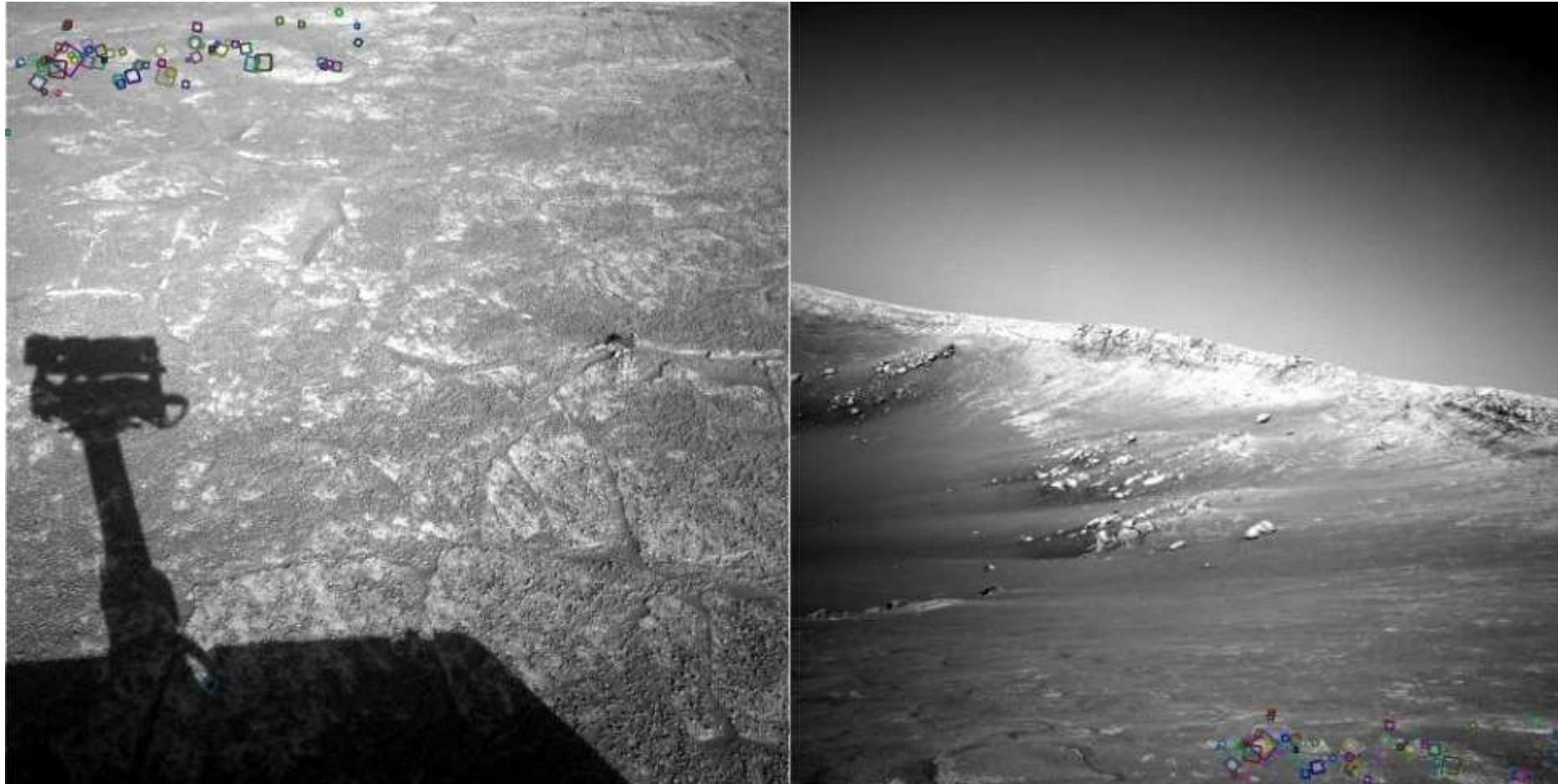
Matching and recognition by interest points



Matching images from NASA Mars Rover



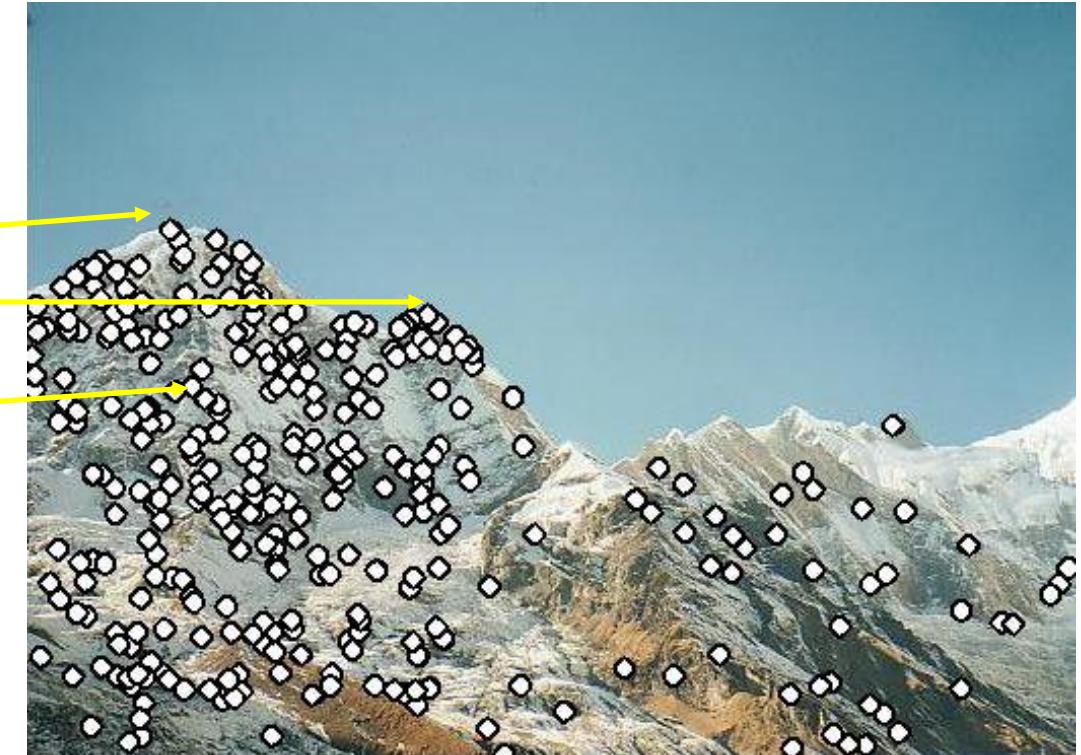
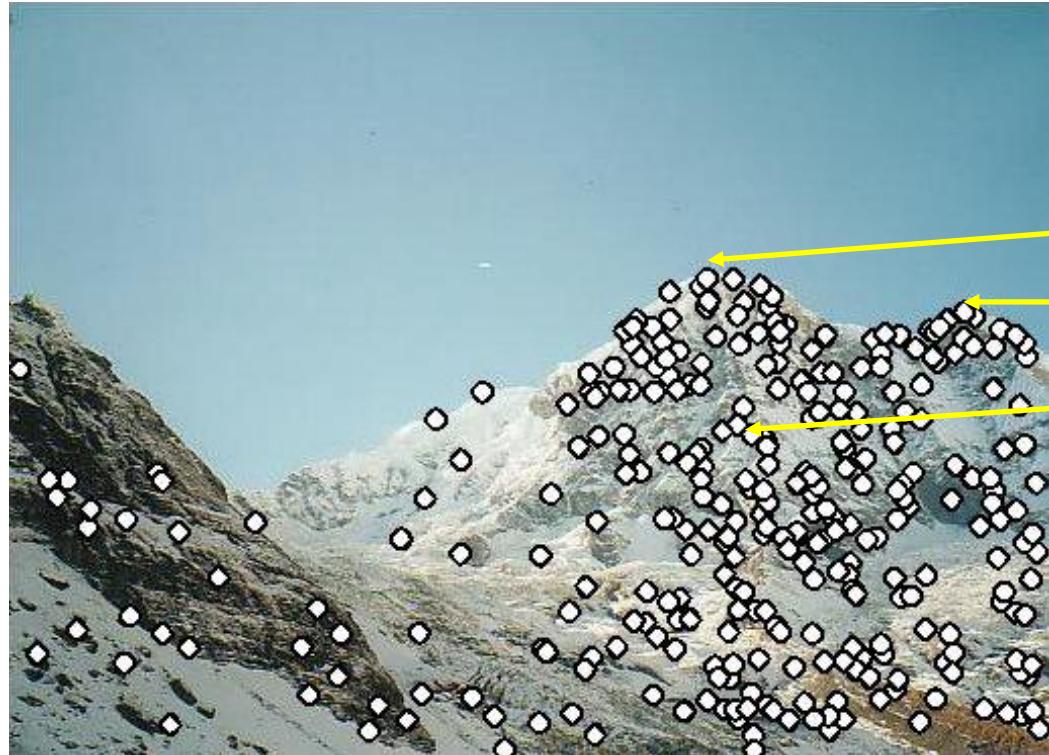
Matching images from NASA Mars Rover



Matching by interest points



Matching by interest points



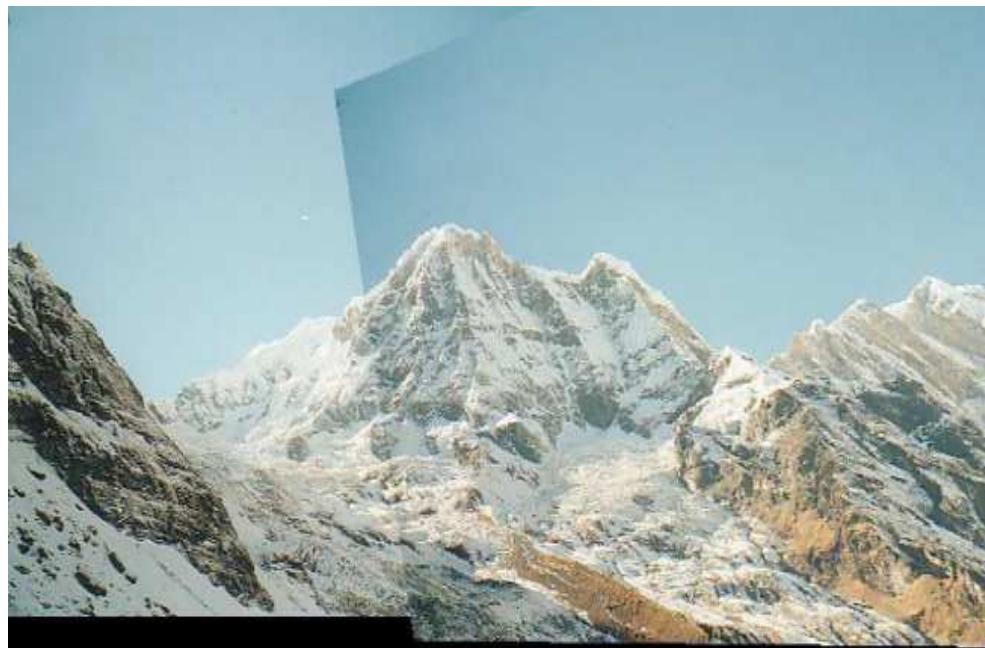


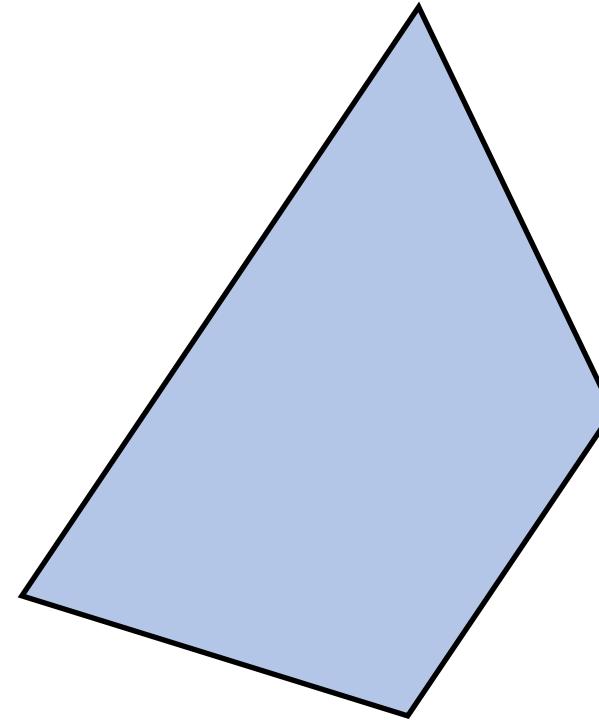
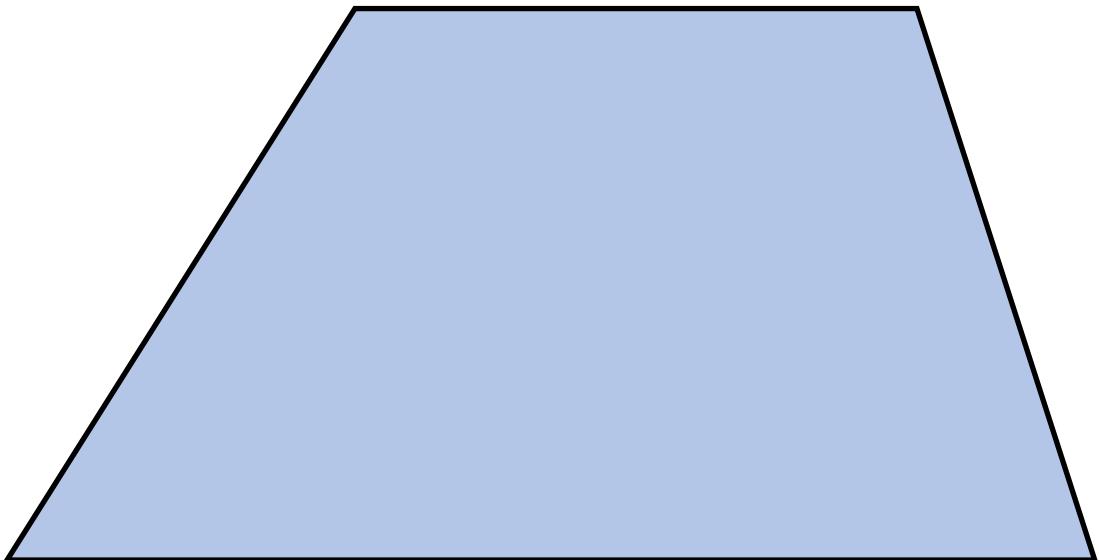
Image stitching, panorama on cameras

Interest point detection

- It is often a pre-requisite step for other tasks, such as facial image analysis, image matching, image classification etc.
- Interest points provide an efficient representation of the image content.
- We only need to deal with some points, instead of all image pixels.

Interest point detection

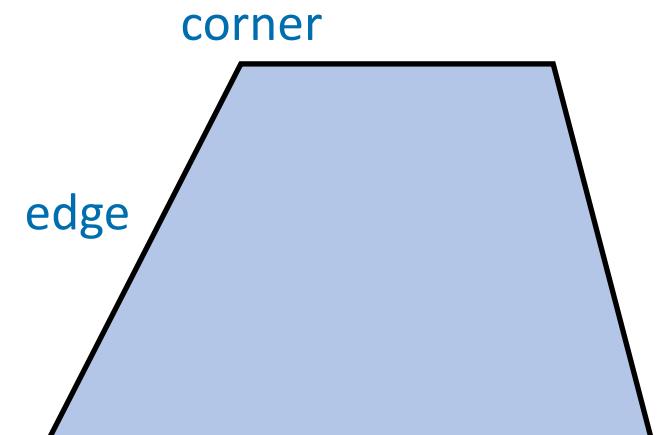
- Since these interest points are so useful, let us try to detect them using a mathematically founded algorithm.
- We will first describe the Harris detector, or Harris corner detector.



Pick a point in the left image. Find it again in the right image.
What type of point would you select?
The corners!

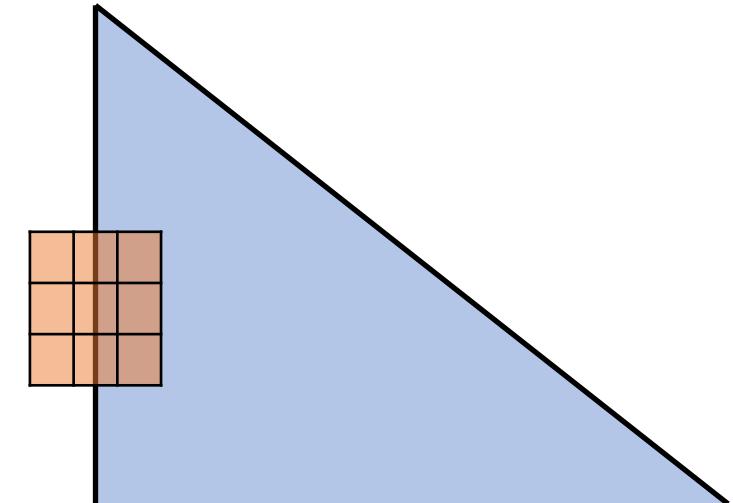
Corner detection

- We are interested in corners, which are intersections of edges.
- For edge detection, we calculate the magnitude of image gradient at each pixel.
 - Edge pixels correspond to high magnitude of gradient.
- For corner detection, if we just look at the gradient magnitude of a single pixel, we could not differentiate between an edge pixel and a corner pixel.

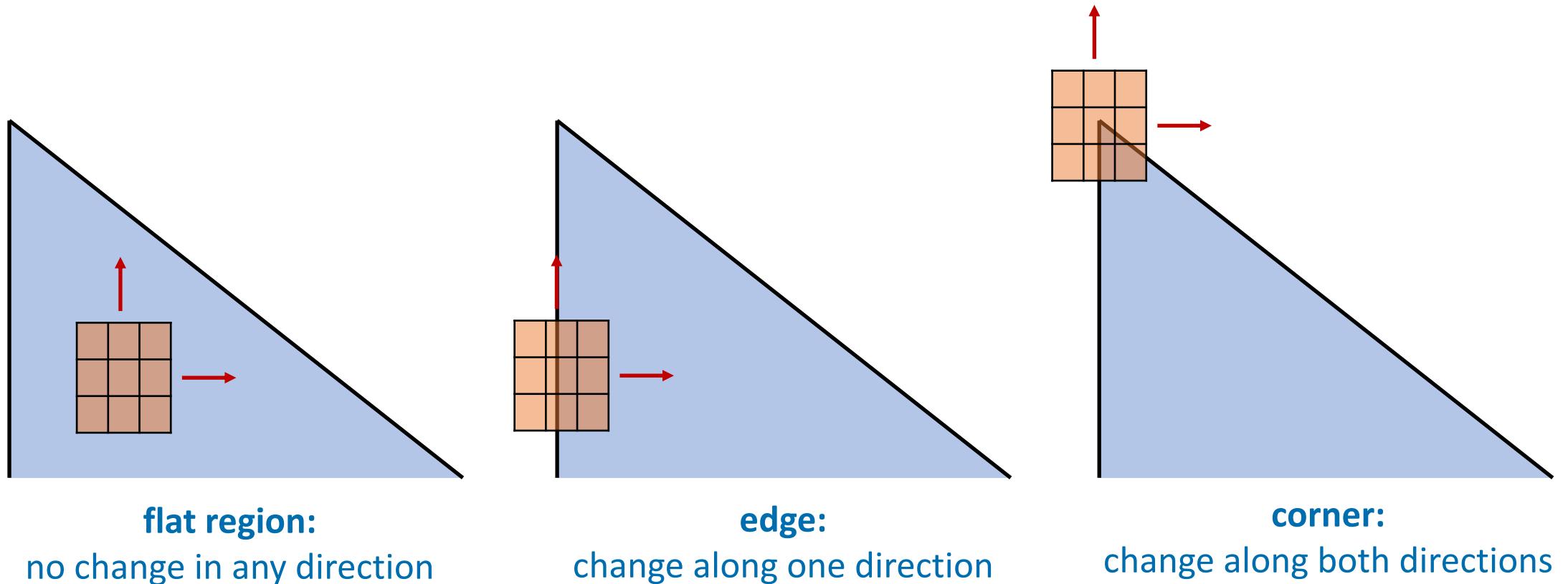


Basic idea

- However, if we look at a small window, we can tell the difference between edge pixels and corner pixels.
- We can shift the window and calculate the change of intensities for the window.
 - Edge: change of intensity along just one direction
 - Corner: change of intensity along both directions



Corner detection



flat region:
no change in any direction

edge:
change along one direction

corner:
change along both directions

Harris detector

- How do we use mathematics to describe the change of pixel intensities in this window W ?
- If the window shifts by $[u, v]$, the change of intensities is given by

$$E(u, v) = \sum_{(x,y) \in W} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

sum of squared difference (SSD)

window function

intensity in the shifted window

original intensity

or

The diagram shows two ways to define a window W over a grayscale image. On the left, a red rectangle represents a 'rectangular window', with a dashed line below it representing the 'original intensity'. The area under the rectangle is shaded blue and labeled '1 in window, 0 outside'. On the right, a red bell-shaped curve represents a 'Gaussian window', with a dashed line below it representing the 'original intensity'. The area under the curve is shaded blue and labeled 'Gaussian'.

Change of intensities

- Taylor expansion

$$I(x + u, y + v) = I(x, y) + uI_x(x, y) + vI_y(x, y) + \dots \text{ higher order terms}$$

- If we use the first order approximation,

$$\begin{aligned} E(u, v) &= \sum w(x, y)[I(x + u, y + v) - I(x, y)]^2 \\ &\approx \sum w(x, y)[uI_x(x, y) + vI_y(x, y)]^2 \\ &= \sum w(x, y)(u^2I_x^2 + 2uvI_xI_y + v^2I_y^2) \\ &= \sum w(x, y)[u, v] \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\ &= [u, v] \sum w(x, y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

first order approximation

re-write as a matrix formula

Change of intensities

- For a small shift $[u, v]$, we have the following bilinear approximation,

$$E(u, v) \approx [u, v] \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix, computed from image derivatives,

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



window function products of image derivatives

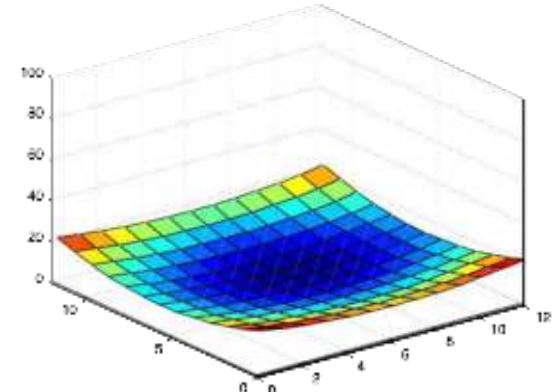
- We will get a large $E(u, v)$, where image derivatives are large.
- M is a 2×2 matrix, we can infer the directions of change from M .

Change of intensities

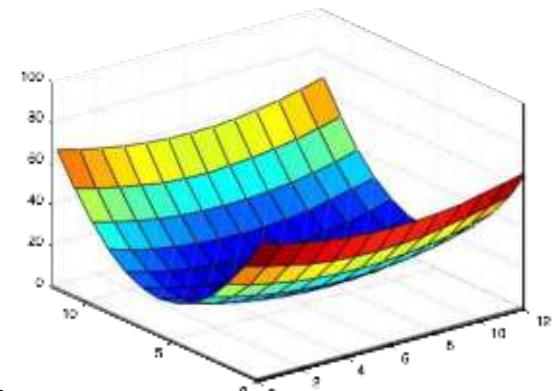
- Let us look at $E(u, v)$

$$E(u, v) \approx [u, v] \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

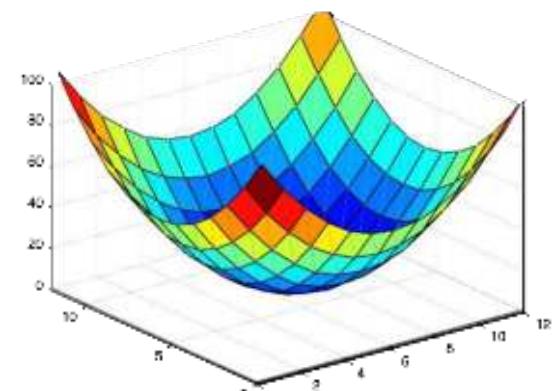
- If $M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, this is a **flat region**.
- If $M = \begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix}$, there will be a large change if we shift along u , but little change if we shift along v . This is an **edge**.
- If $M = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$, there will be a large change whichever way we shift the window. This is a **corner**.



flat region



edge



corner

Change of intensities

- What if M is more complicated than the above scenarios?

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- Make M simpler by performing eigen decomposition.
- Since M is a real symmetric matrix, we can decompose M by

$$M = P \Lambda P^T$$

each column is a eigenvector diagonal matrix with eigenvalues $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$ each row is a eigenvector

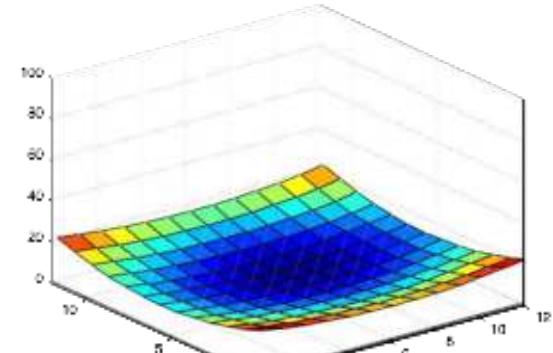
- The eigenvectors are orthogonal to each other.

Change of intensities

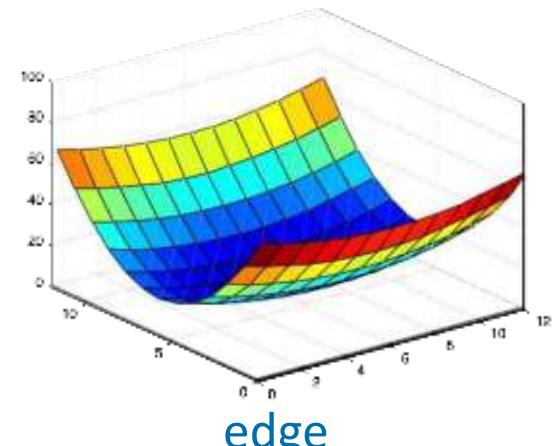
- Let us look at the $E(u, v)$

$$E(u, v) \approx [u, v] \cdot P \Lambda P^T \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

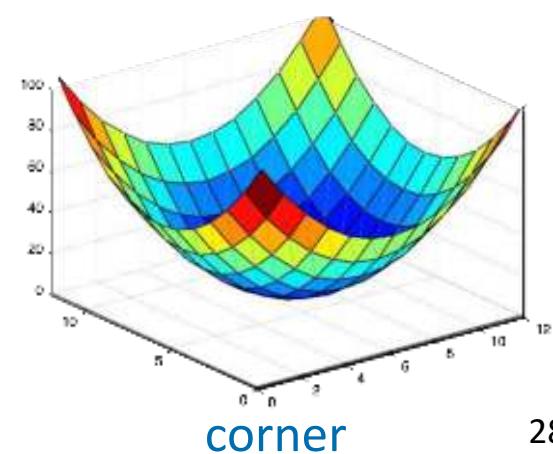
- If $\Lambda = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, this is a **flat region**.
- If $\Lambda = \begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix}$, there will be a large change if we shift along the first eigenvector, but little change if we shift along the second eigenvector. This is an **edge**.
- If $\Lambda = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$, there will be a large change whichever way we shift the window. This is a **corner**.



flat

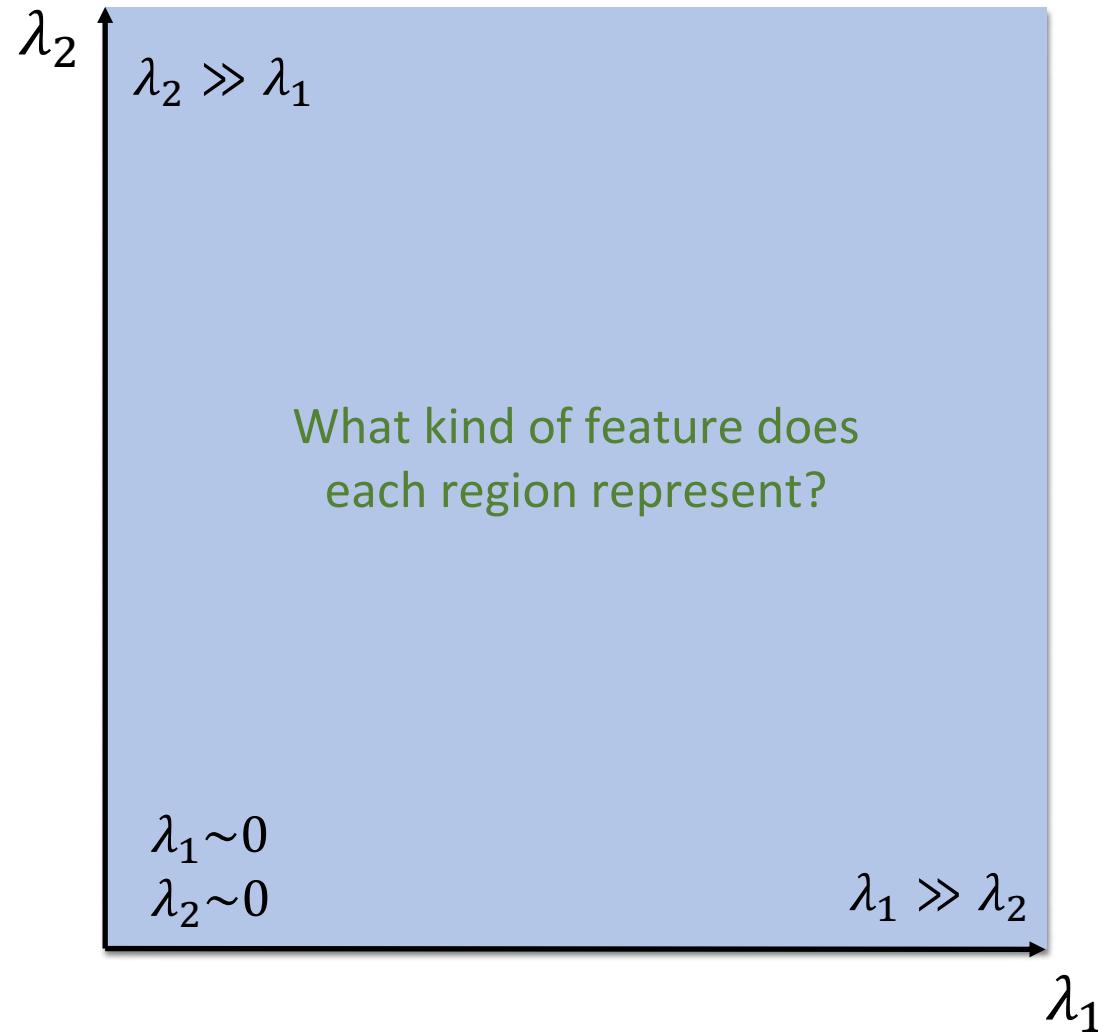


edge

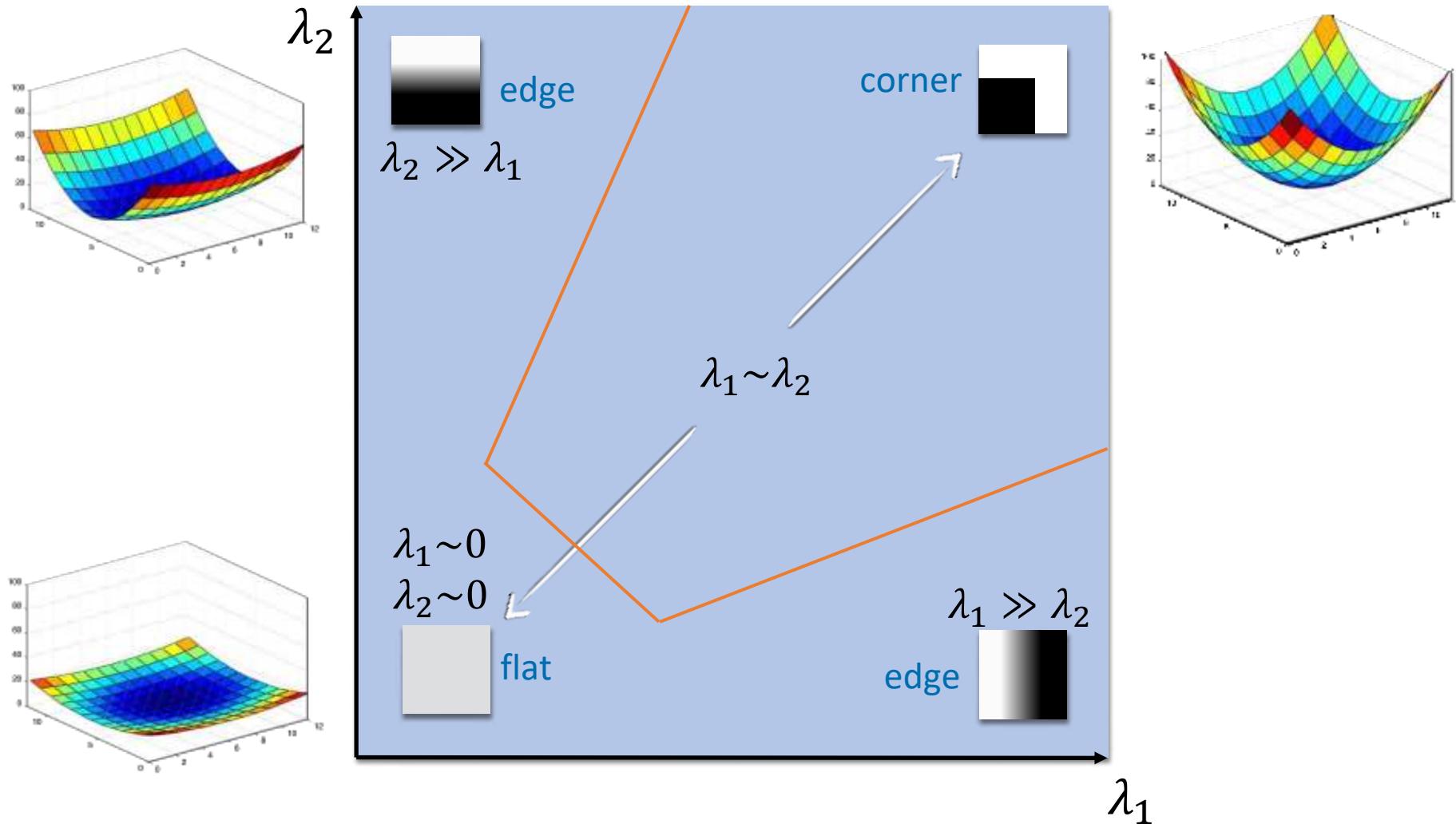


corner

Interpreting eigenvalues



Interpreting eigenvalues



Define cornerness

- Harris and Stephens (1988)

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

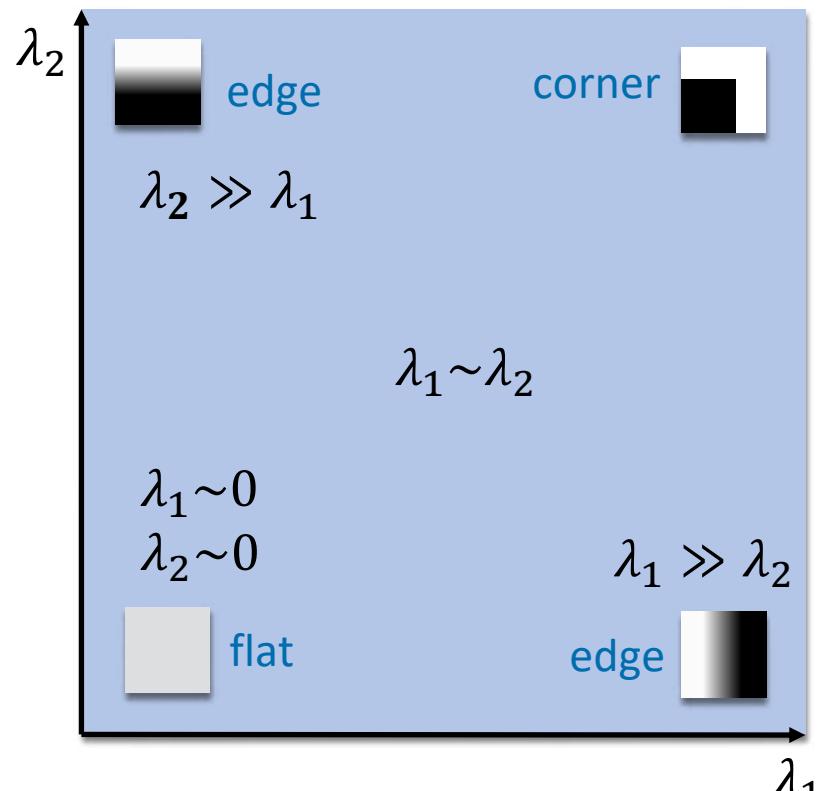
where k is a small number, e.g. $k = 0.05$.

- Kanade and Tomasi (1994)

$$R = \min(\lambda_1, \lambda_2)$$

- Noble (1998)

$$R = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2 + \epsilon}$$

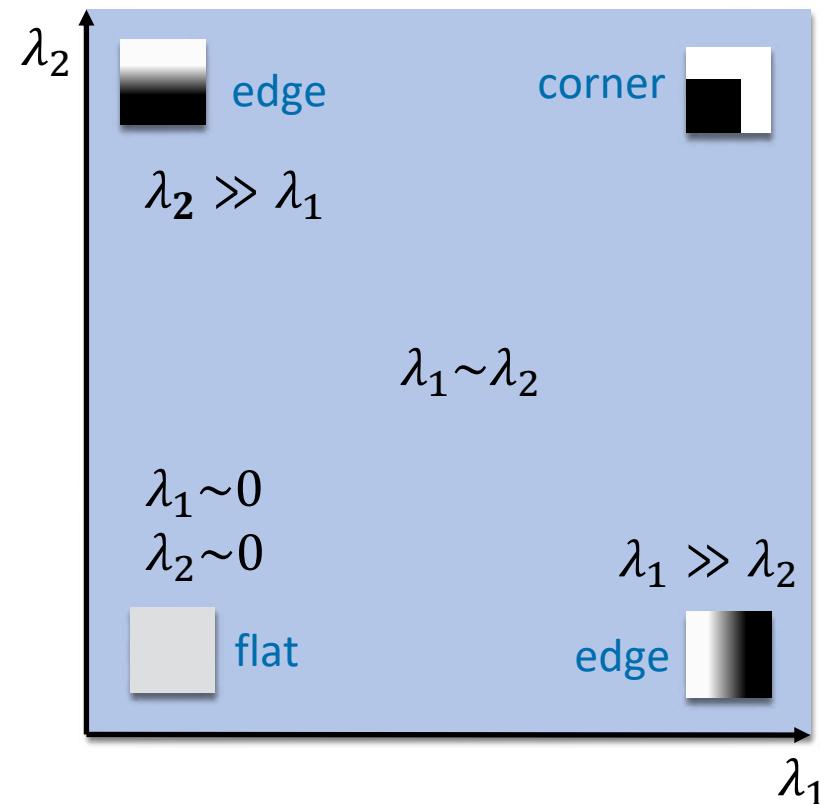


Define cornerness

- Harris and Stephens (1988)

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

where k is a small number, e.g. $k = 0.05$.



Cornerness

- Actually, if we only want the values of $\lambda_1\lambda_2$ and $\lambda_1 + \lambda_2$, we do not need to perform eigen-decomposition.

- If we use the following properties for matrix determinant and trace, for $M = P\Lambda P^T$, we have

$$\det(M) = \det(P\Lambda P^T) = \det(\Lambda) = \lambda_1\lambda_2 \quad \det(AB) = \det(A)\det(B)$$

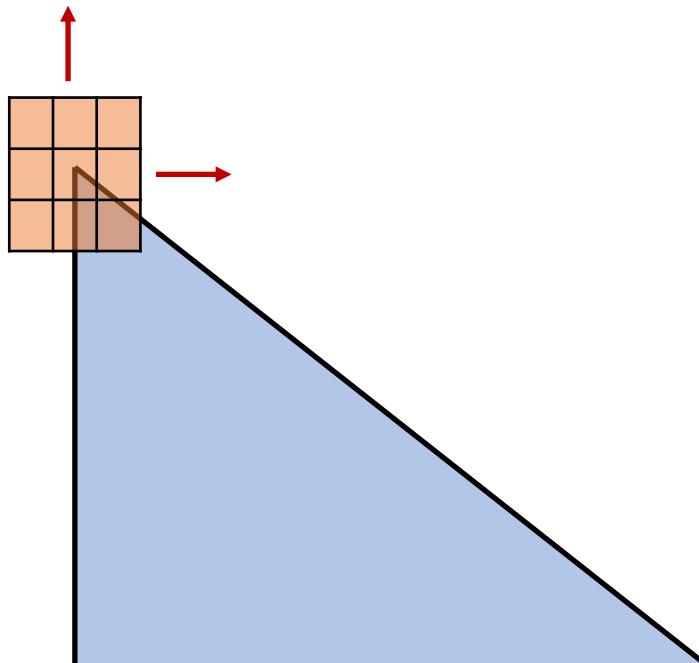
$$\text{trace}(M) = \text{trace}(P\Lambda P^T) = \text{trace}(\Lambda) = \lambda_1 + \lambda_2 \quad \text{trace}(AB) = \text{trace}(BA)$$

- We simply need to evaluate the determinant and trace for the original matrix M .

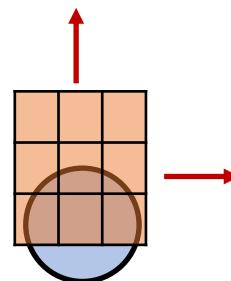
$$R = \det(M) - k(\text{trace}(M))^2$$

Harris detector

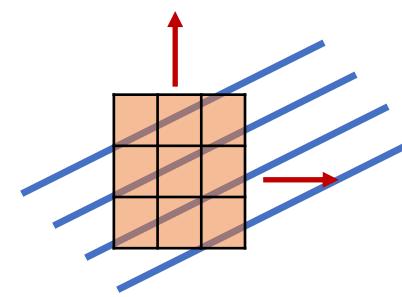
- This detector finds strong responses not only at corners, but also at blobs and textures.



corner:
change along both directions



blob:
change along both directions



texture:
change along both directions

Harris detector

Algorithm

Compute x and y derivatives of an image,

$$I_x = G_x * I, I_y = G_y * I$$

e.g. G can be the Sobel filter

At each pixel, compute the matrix M ,

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Calculate the detector response $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$

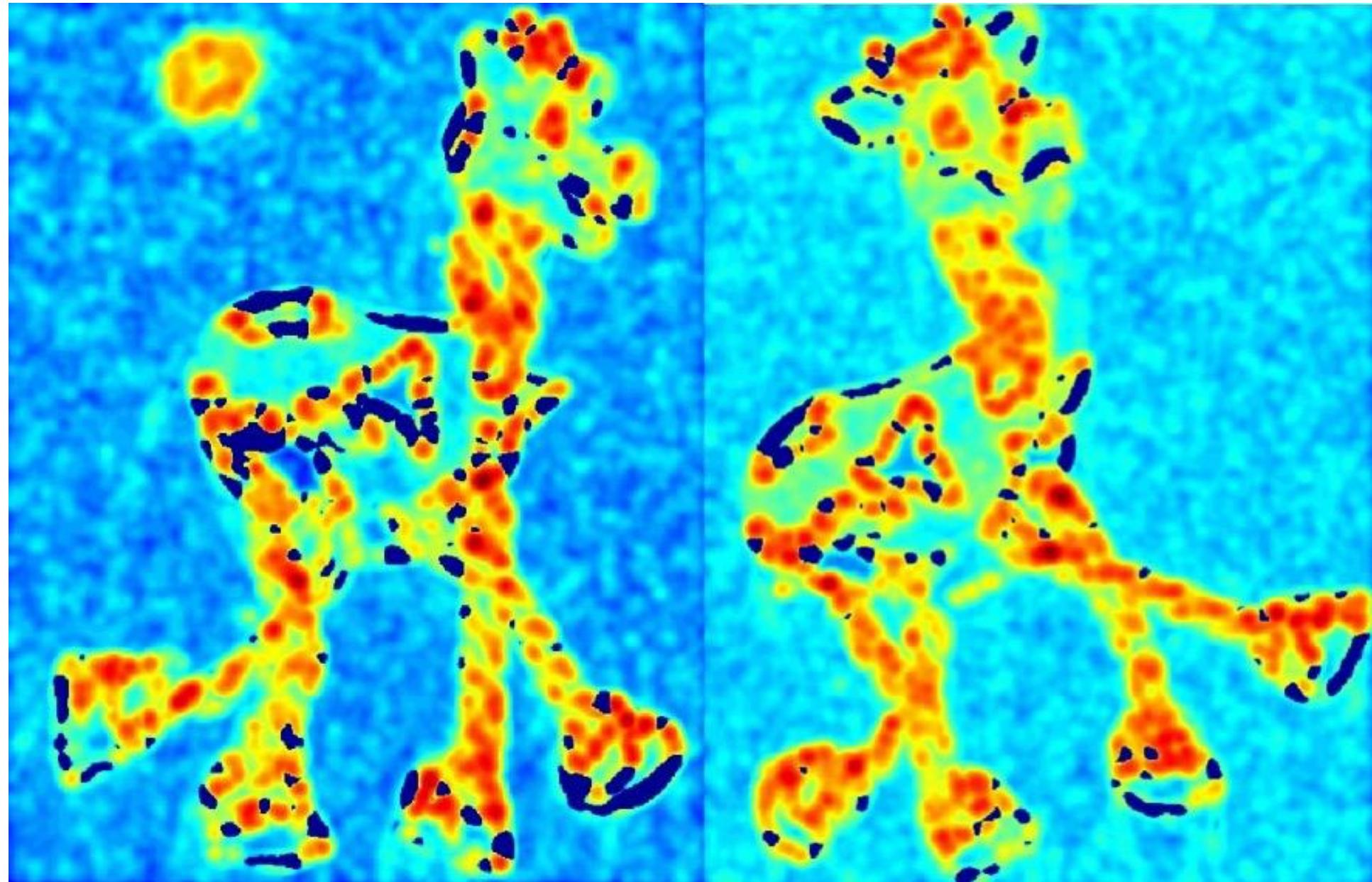
Detect interest points which are local maxima and whose response R are above a threshold.



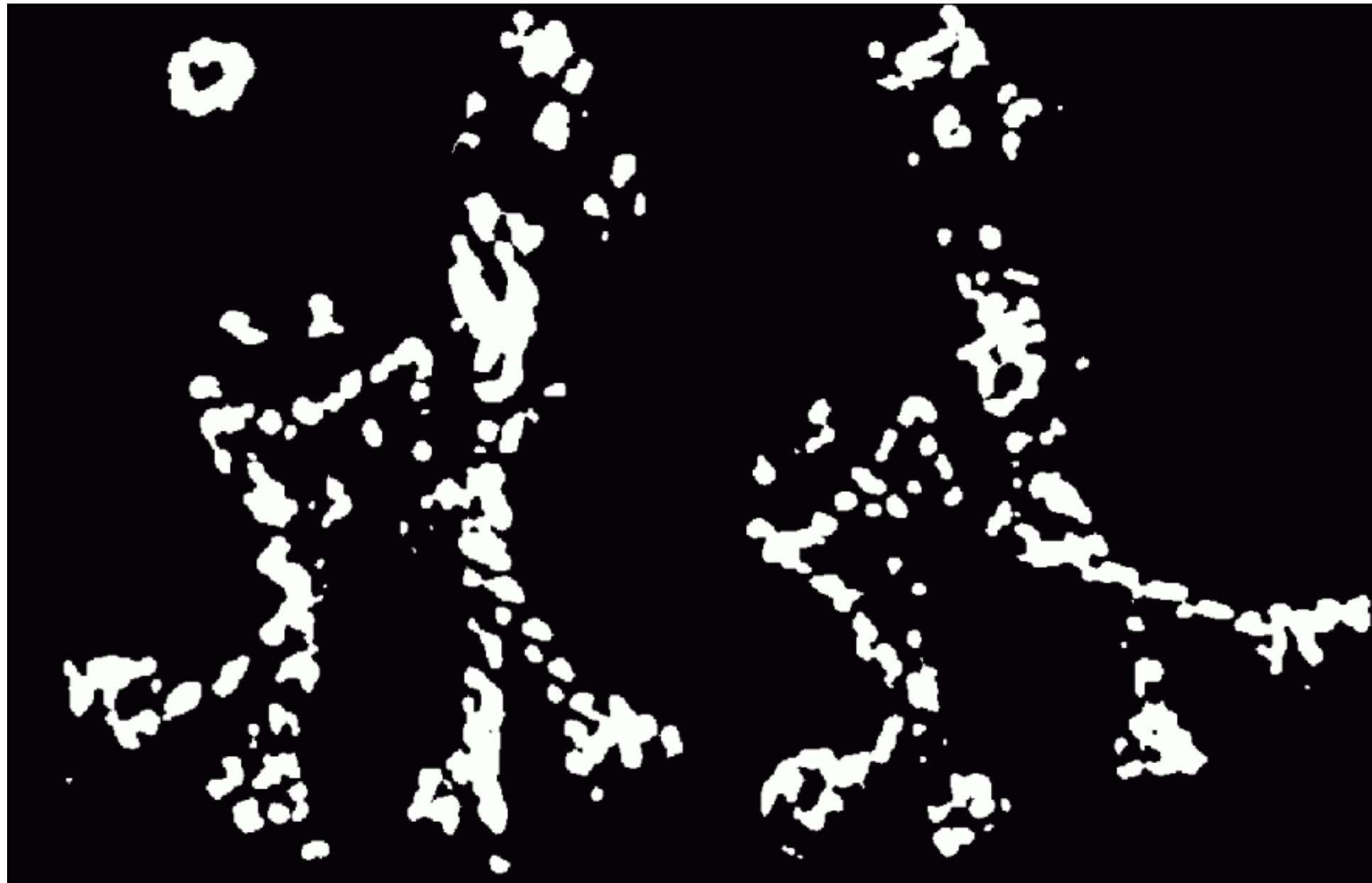
Image A



Image B



Harris detector response



Thresholding



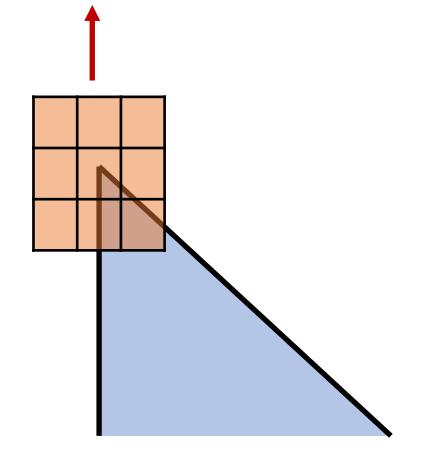
Finding local maxima (non-maximum suppression)



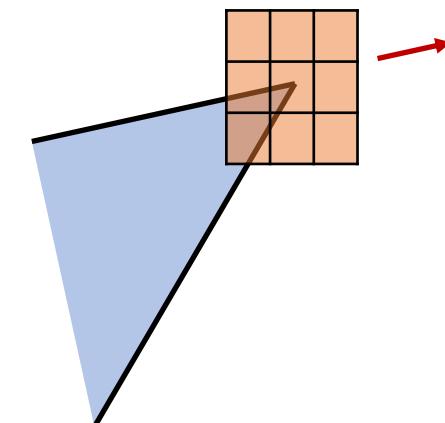
Detected interest points

Invariant to rotation

- Harris detector is rotation-invariant.
- If a corner is rotated, you can still get the same change of intensities when you shift the window along a rotated direction.



 **Rotate**



Is it invariant to scale?

- Harris corner detector is not invariant to scale.



Scale-invariant detection

- Harris detector is not scale-invariant.
- How do we detect corners at different scales?
 - We can apply Harris detector at multiple scales and find the cornerness response at the most suitable scale.
- How do we normally get multi-scale images?
 - Gaussian smoothing with different σ
 - Sampling with different spatial resolutions



$\sigma = 1$



$\sigma = 3$



$\sigma = 5$



$\sigma = 7$

Image convolved with Gaussian kernels of different σ provide information at different scale.

 $\sigma = 1$  $\sigma = 3$  $\sigma = 5$  $\sigma = 7$

Harris detector response when different Gaussian kernels are used for calculating image derivatives.

Summary

- Harris detector finds interest points with strong responses at corners and blobs.
- It calculates the change of intensities within a small window when it is shifted in the image, which can be quantified by local image derivatives.
- It is rotation-invariant, but not scale-invariant.
- We will discuss how to make the detector scale-invariant in the next lecture.

References

- Sec. 4.1.1 Feature detectors. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Interest Point Detection II

Dr Wenjia Bai

Department of Computing & Brain Sciences

Interest point detection

- How to handle scales in detection?
 - Scale adapted Harris detector
- Other interest point detectors
 - Laplacian of Gaussian
 - Difference of Gaussian

Scale

- Harris detector is not invariant to scale.





$\sigma = 1$



$\sigma = 3$

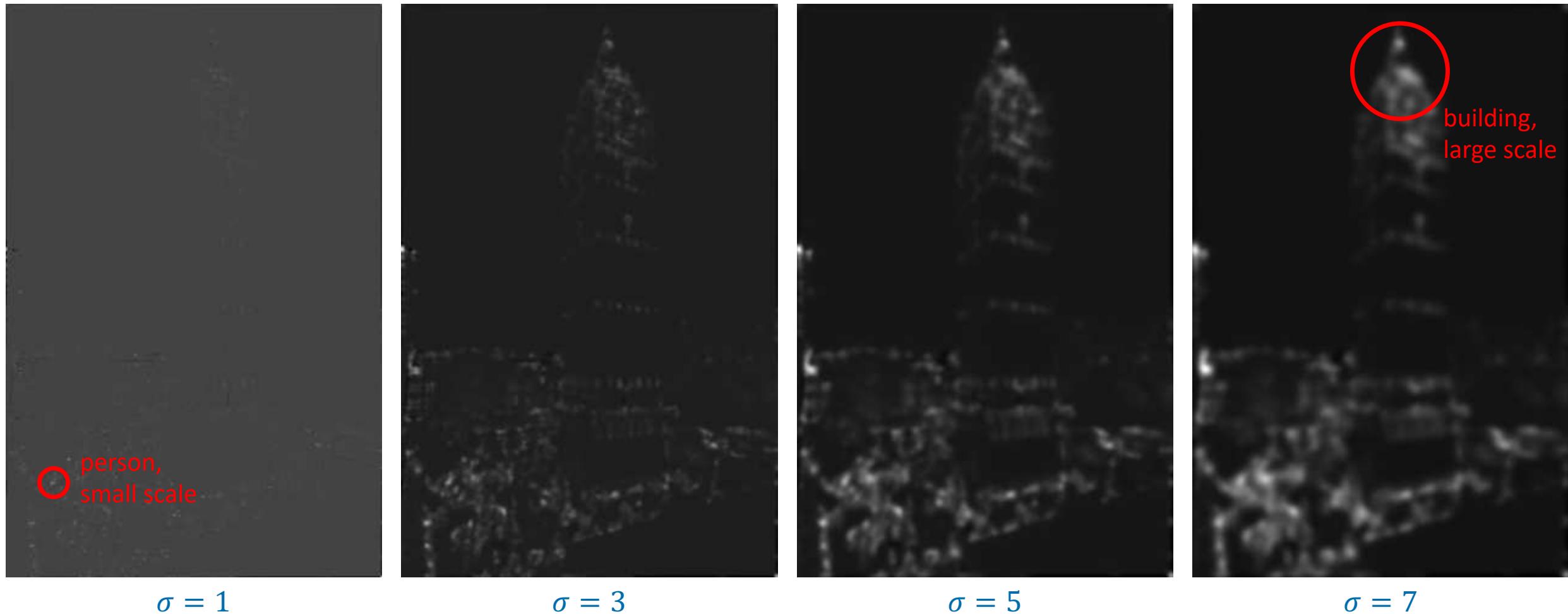


$\sigma = 5$



$\sigma = 7$

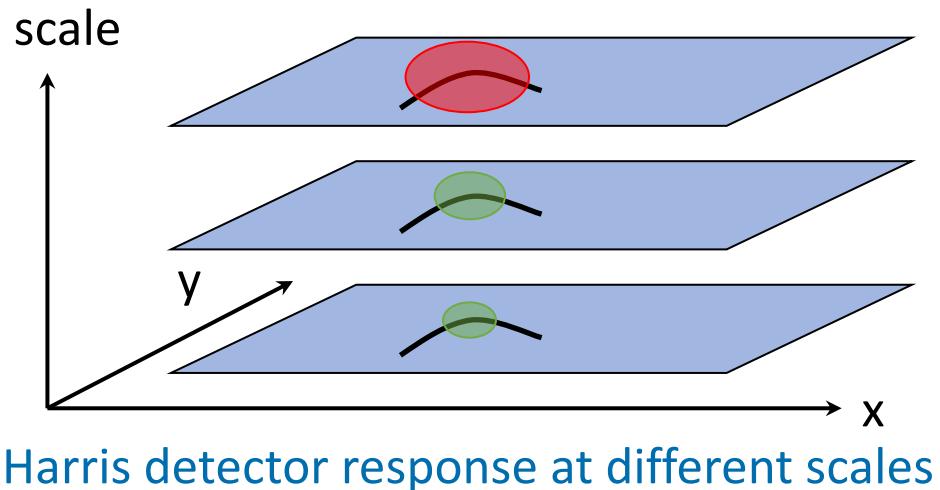
Image convolved with Gaussian kernels of different σ provide information at different scale.



Harris detector response when different Gaussian kernels are used for calculating image derivatives.

Scale

- How do we determine the scale we use at each pixel?
 - Are we looking at a big building or a small cat?
- Intuitive idea
 - We check whether the Harris detector gives the highest response at this scale.

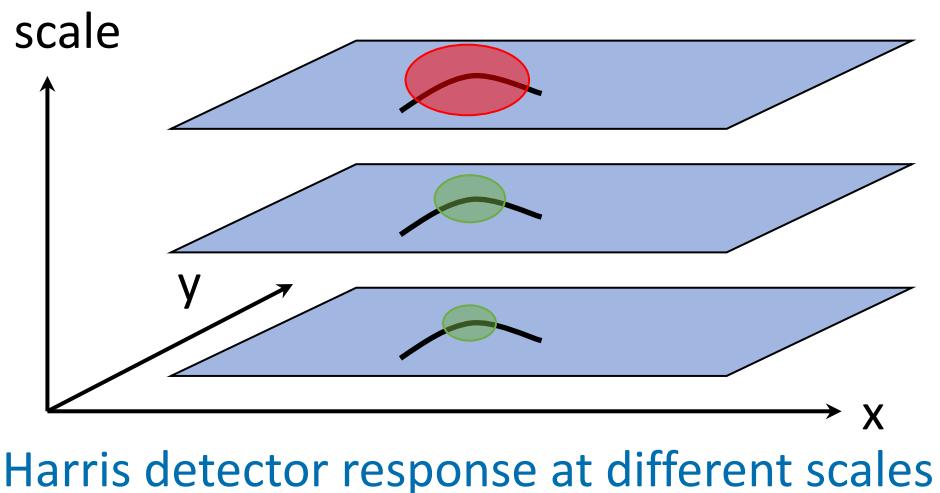


If this region looks most like a corner at this scale σ , Harris detector response should be high.

If this region does not look like a corner at this scale, Harris detector response will be low.

Scale

- There is only one problem.
- Direct comparison of Harris detector response across scales may not be fair.



If this region looks most like a corner at this scale σ , Harris detector response should be high.

If this region does not look like a corner at this scale, Harris detector response will be low.

Response at different scales

- The Harris detector response is calculated using the eigenvalues of M ,

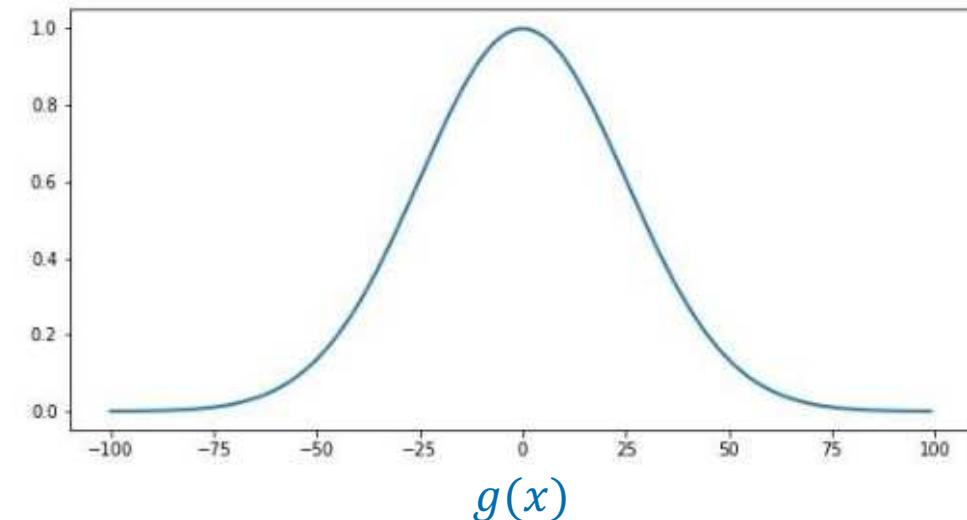
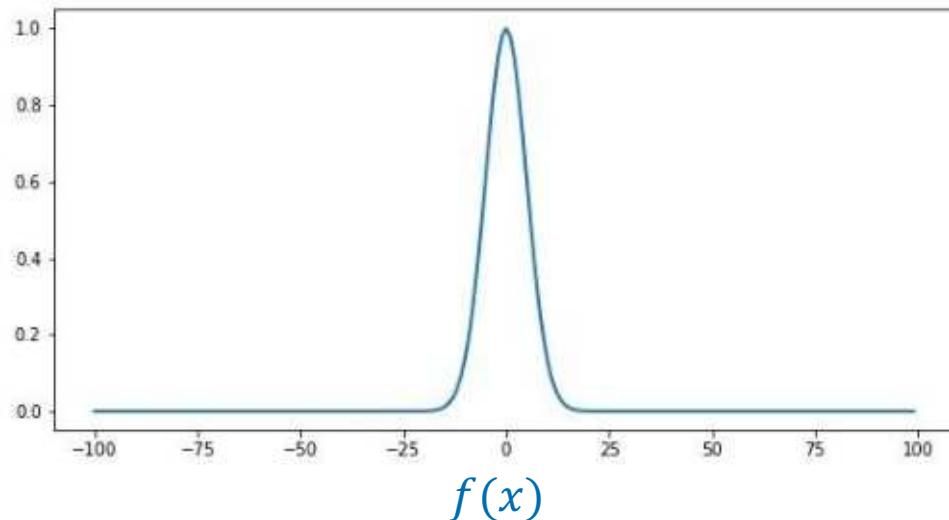
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y^2(\sigma) \end{bmatrix}$$

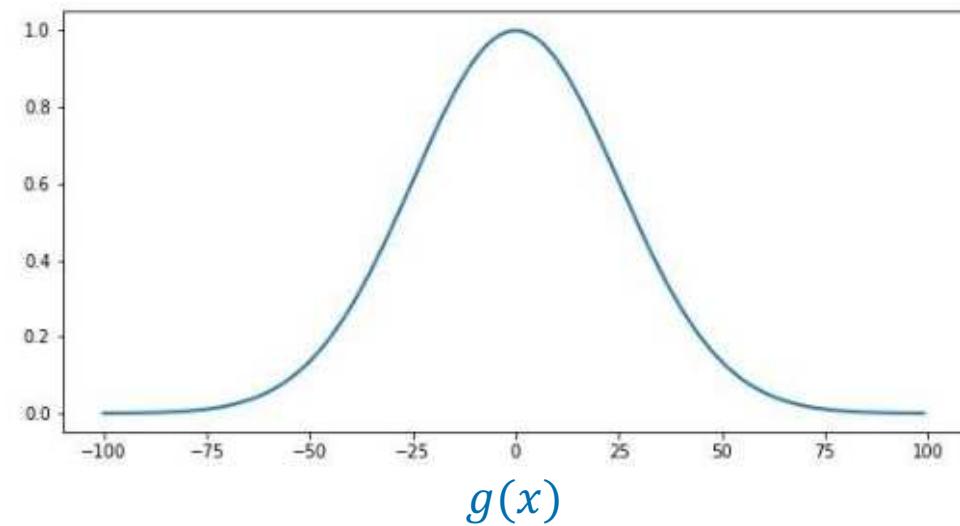
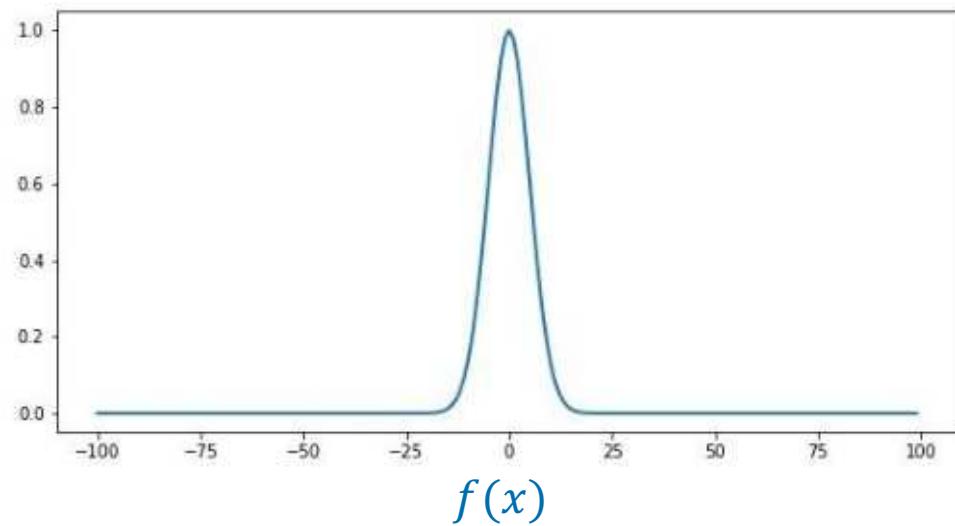
image gradient at scale σ
(Gaussian kernel σ + derivative)

- The response is determined by the eigenvalues of M , which are in turn determined by the derivatives $I_x(\sigma)$ and $I_y(\sigma)$.
- As you will see, the derivatives $I_x(\sigma)$ and $I_y(\sigma)$ are inversely proportional to scale σ .
 - The larger the scale σ , the smaller the magnitude of the derivative.

Derivatives at different scales

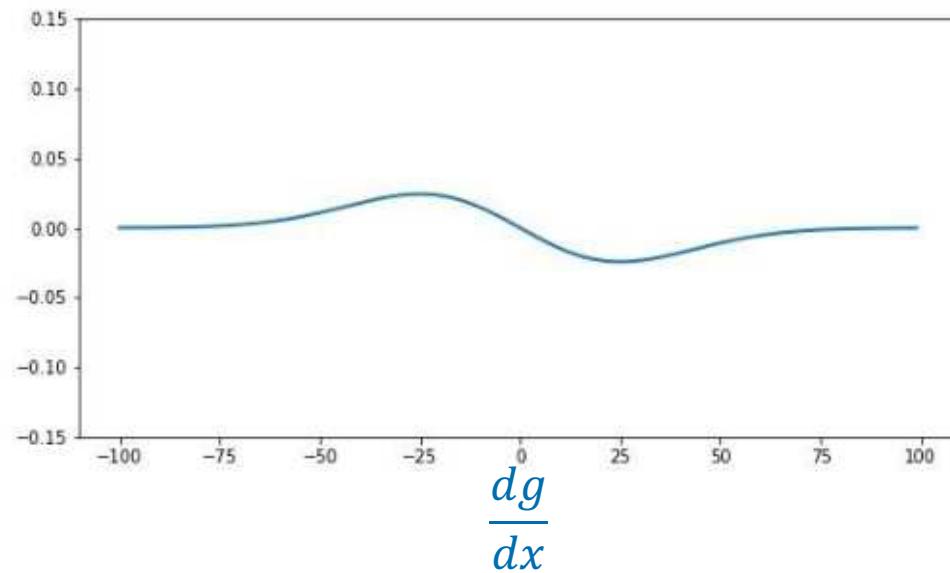
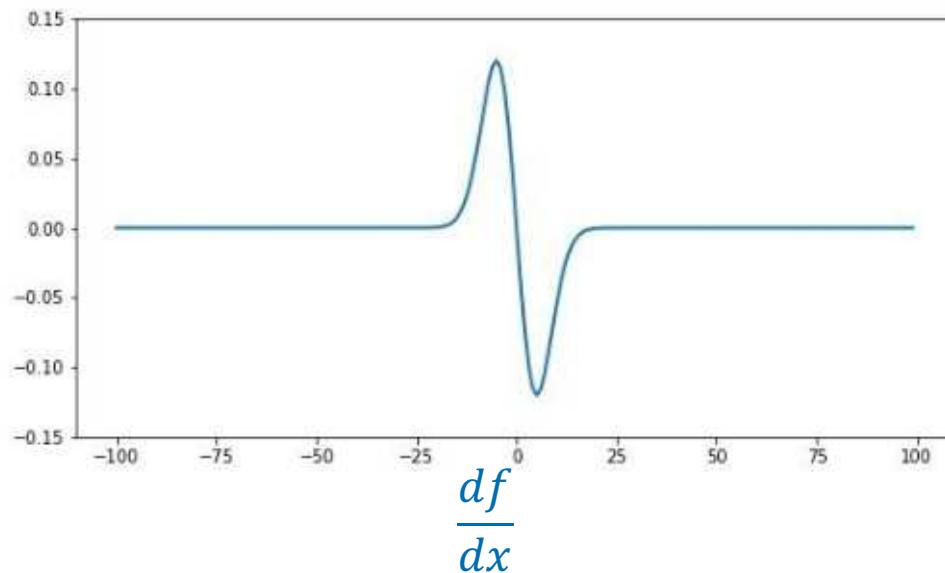
- Consider two signals $f(x)$ and $g(x)$, which are related by
$$f(x) = g(sx)$$
i.e. they only differ by scale s and they have same peak magnitude.
- This can happen, for example, when we take a picture of the same object (human, mountain etc) with different zoom factors.





Derivatives at different scales

- However, when we calculate the derivatives, their peak magnitudes differ.
- How much do they differ?



Derivatives at different scales

- Since $f(x) = g(sx)$, we have

$$f(x + \Delta x) - f(x) = g(sx + s\Delta x) - g(sx)$$

- Let us write down the Taylor expansion for $f(x + \Delta x)$ and $g(sx + s\Delta x)$,

$$f(x + \Delta x) = f(x) + \Delta x \cdot f'(x) + \dots$$

$$g(x + \Delta x) = g(x) + \Delta x \cdot g'(x) + \dots$$

$$g(sx + s\Delta x) = g(sx) + s\Delta x \cdot g'(sx) + \dots$$

- Substitute into the first equation, we have,

$$\Delta x \cdot f'(x) = s\Delta x \cdot g'(sx)$$

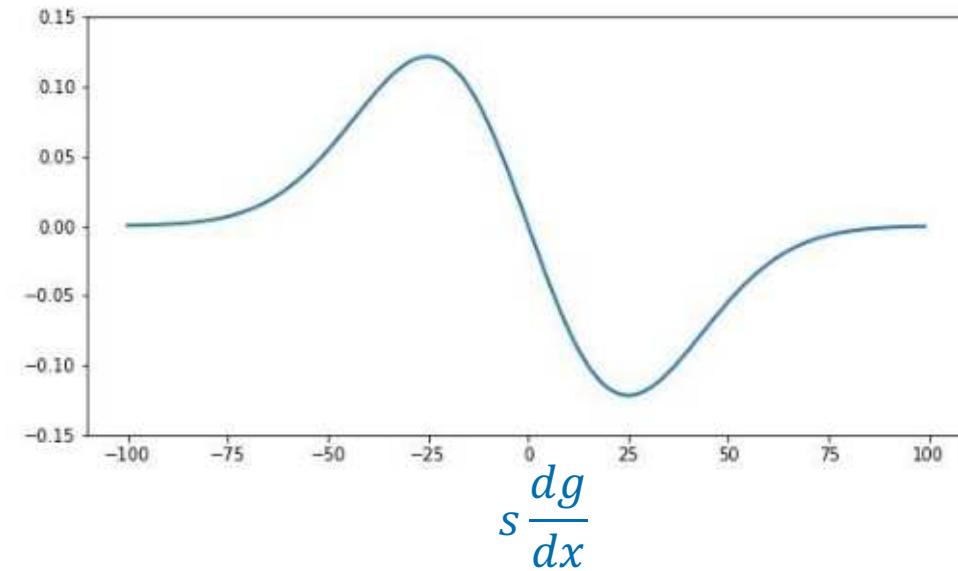
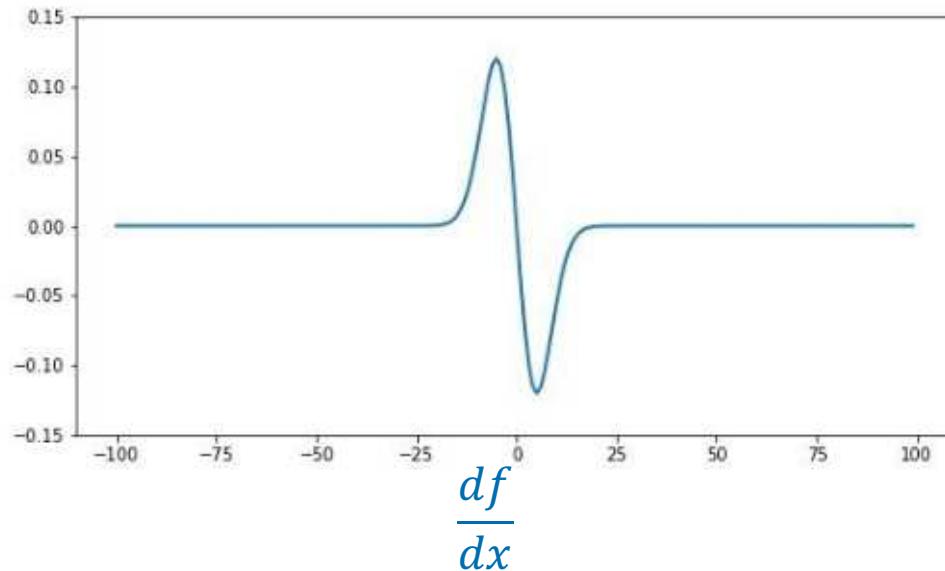
$$f'(x) = sg'(sx)$$

- In other words,

$$\frac{df}{dx} = s \cdot \frac{dg}{dx} \Big|_{sx}$$

Signal at different scales

- To make the derivative magnitude comparable across scales, we need to multiply the derivative by its scale s .
 - Then $\frac{df}{dx}$ can be compared with $s \frac{dg}{dx}$.
 - The same object will give same magnitude of response, regardless of the zoom factor.



Scale adapted Harris detector

- This is the scale adapted Harris detector,

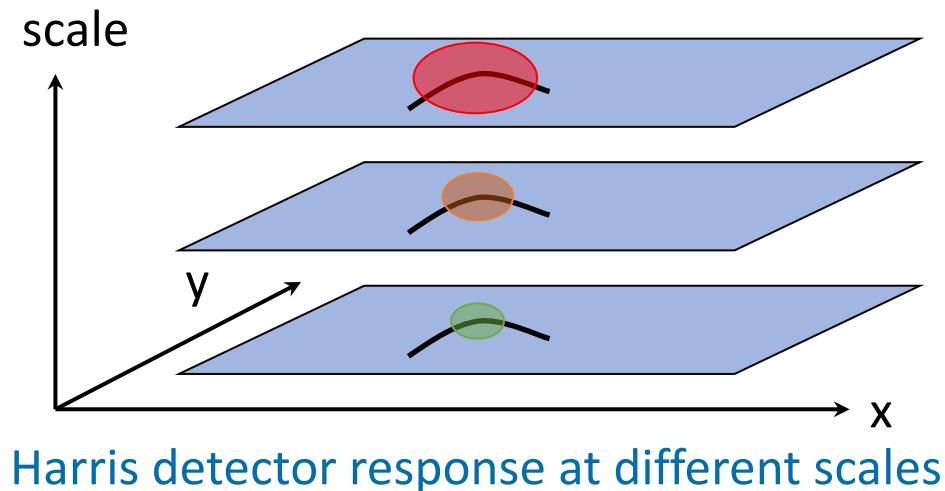
$$\begin{aligned} M &= \sum_{x,y} w(x,y) \begin{bmatrix} \sigma^2 I_x^2(\sigma) & \sigma^2 I_x(\sigma) I_y(\sigma) \\ \sigma^2 I_x(\sigma) I_y(\sigma) & \sigma^2 I_y^2(\sigma) \end{bmatrix} \\ &= \sum_{x,y} w(x,y) \sigma^2 \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma) I_y(\sigma) \\ I_x(\sigma) I_y(\sigma) & I_y^2(\sigma) \end{bmatrix} \end{aligned}$$

normaliser

- We can apply scale adapted Harris detector at multiple scales.
- At each pixel, determine the scale which gives us the largest detector response (e.g. at this scale, it looks most like a corner).

Scale adapted Harris detector

- We calculate the scale adapted detector response for a series of σ , from small-scale to large-scale.
- When we perform interest point detection, we look for local maxima both across space and across scale.



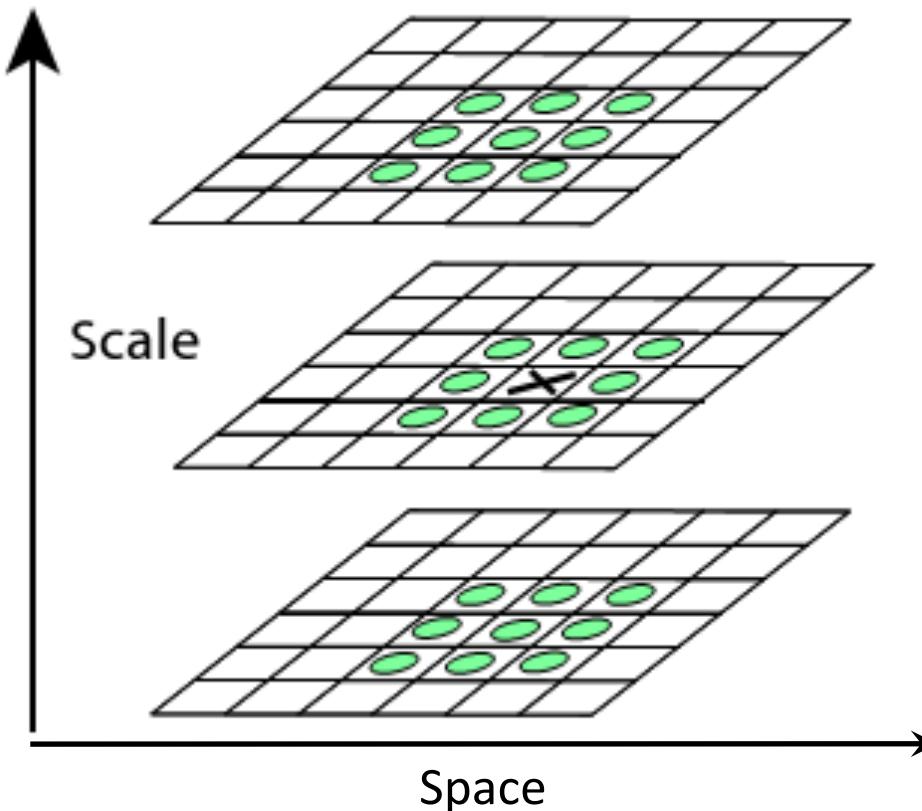
For example,

$$\sigma_3 = 4 \text{ pixel}$$

$$\sigma_2 = 2 \text{ pixel}$$

$$\sigma_1 = 1 \text{ pixel}$$

Scale-space extrema



X is detected as an interest point if it is a local maximum both along scale dimension (most appropriate scale) and across space. A threshold may also be applied.

Scale adapted Harris detector

Algorithm

For each scale σ

- Perform Gaussian smoothing with σ

- Calculate the x and y derivatives of the smoothed image $I_x(\sigma)$ and $I_y(\sigma)$

- At each pixel, compute the matrix M ,

$$M = \sum_{x,y} w(x,y) \sigma^2 \begin{bmatrix} I_x^2(\sigma) & I_x(\sigma)I_y(\sigma) \\ I_x(\sigma)I_y(\sigma) & I_y^2(\sigma) \end{bmatrix}$$

- Calculate the detector response $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$

Detect interest points which are local maxima across both scale and space and whose response R is above a threshold.

Interest point detectors

- Harris detector calculates the response

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

for detecting interest points.

- There are also other mathematical operators that can be used, such as the Laplacian of Gaussian (LoG).
 - It means performing Gaussian smoothing first, followed by Laplacian.
 - But what is Laplacian?

Laplacian filter

- The Laplacian is the sum of second derivatives, for 2D image, it is

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

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

Laplacian filter

Second derivative

- First derivative

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- Second derivative

$$\begin{aligned}f''(x) &= \lim_{\Delta x \rightarrow 0} \frac{f'(x) - f'(x - \Delta x)}{\Delta x} \\&= \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2}\end{aligned}$$

1	-2	1
---	----	---

Laplacian filter

- The Laplacian is the sum of second derivatives, for 2D image, it is

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

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

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

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

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

+

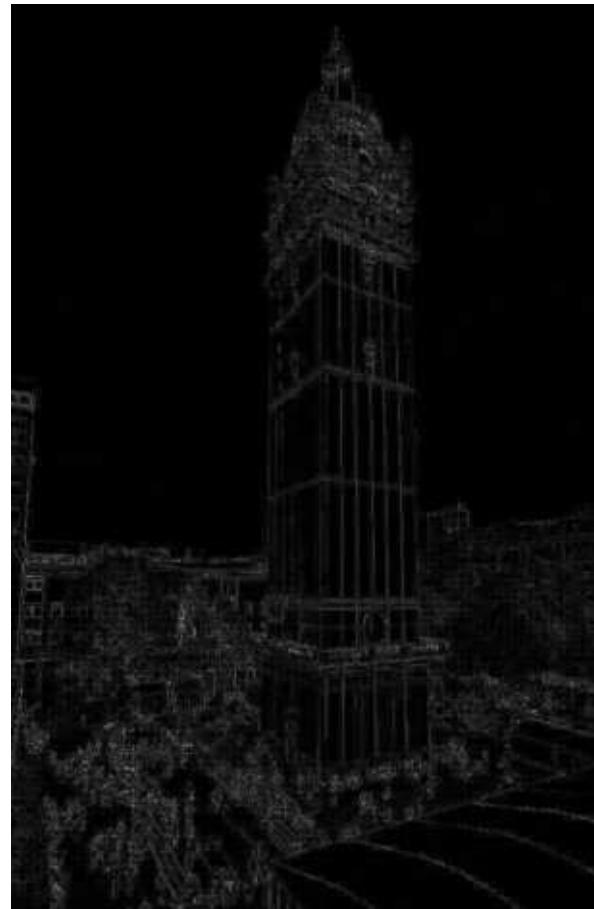
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Laplacian filter

Laplacian filter



Input image



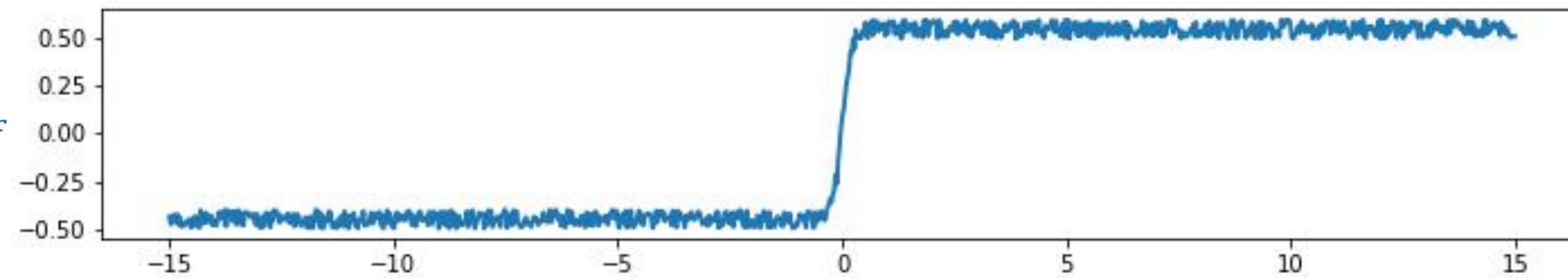
Laplacian

Second derivative

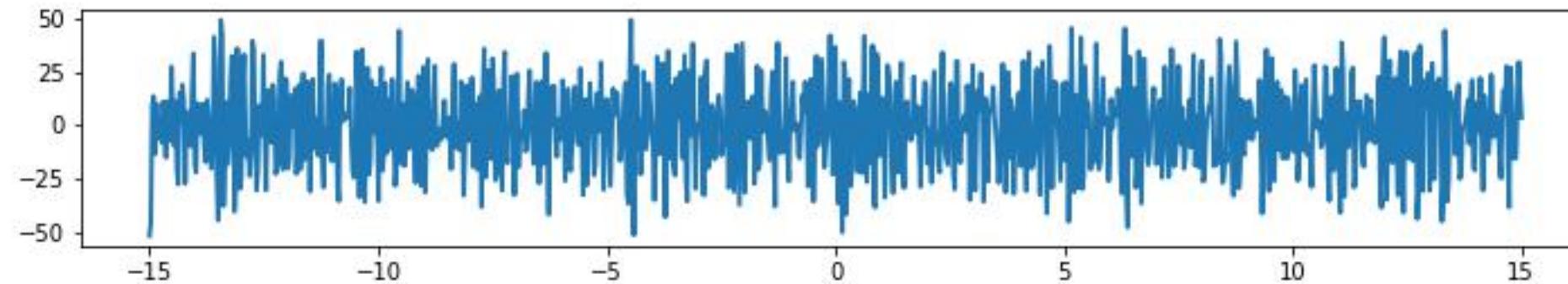
- The second derivative is even more sensitive to noise than the first derivative.

Second derivative of a noisy signal

input signal f

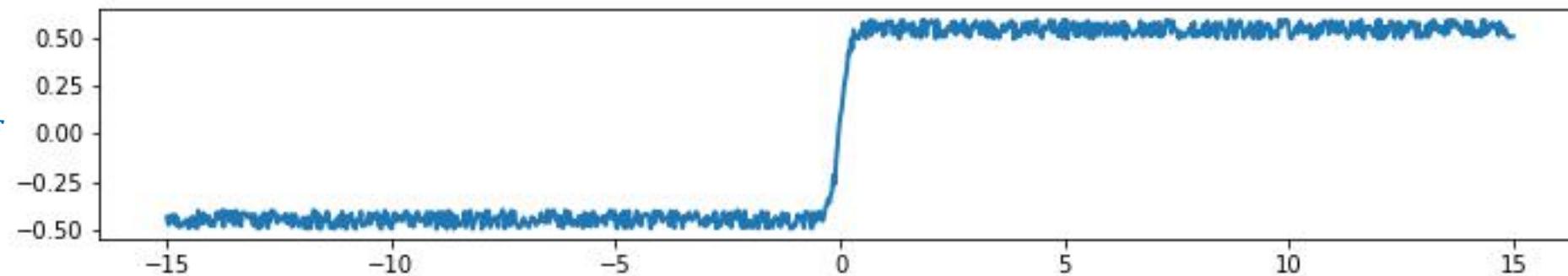


second derivative
 $\frac{d^2f}{dx^2}$

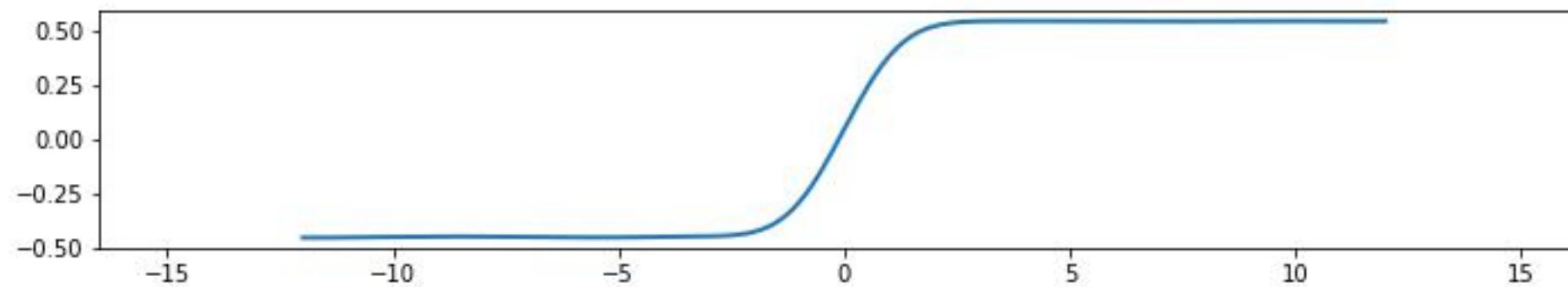


Second derivative of Gaussian smoothed signal

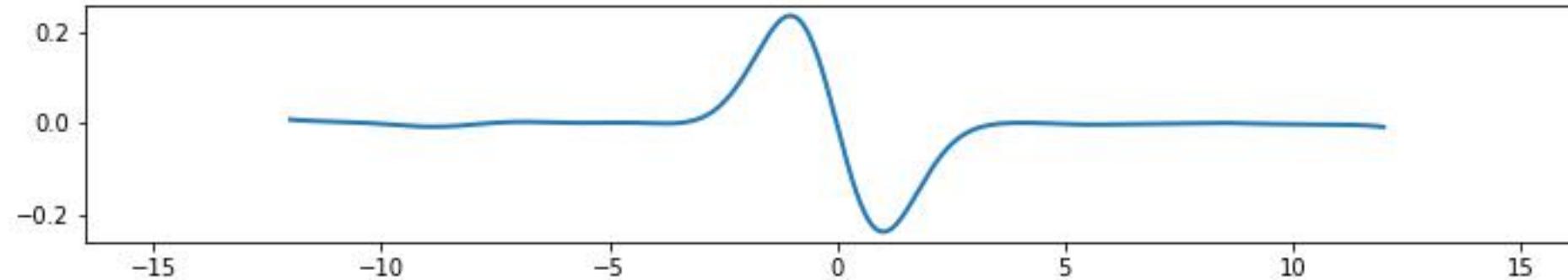
input signal f



Gaussian smoothed
 $f * h$



second derivative
 $\frac{d^2(f * h)}{dx^2}$



Laplacian of Gaussian (LoG)

- For 2D images, we can smooth the image using a Gaussian kernel before calculating the Laplacian.

$$\text{Laplacian: } \Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- This is called the Laplacian of Gaussian filter (LoG).

$$\text{LoG: } \Delta(f * h) = \frac{\partial^2(f * h)}{\partial x^2} + \frac{\partial^2(f * h)}{\partial y^2}$$

↓
Gaussian filter

Laplacian of Gaussian (LoG)

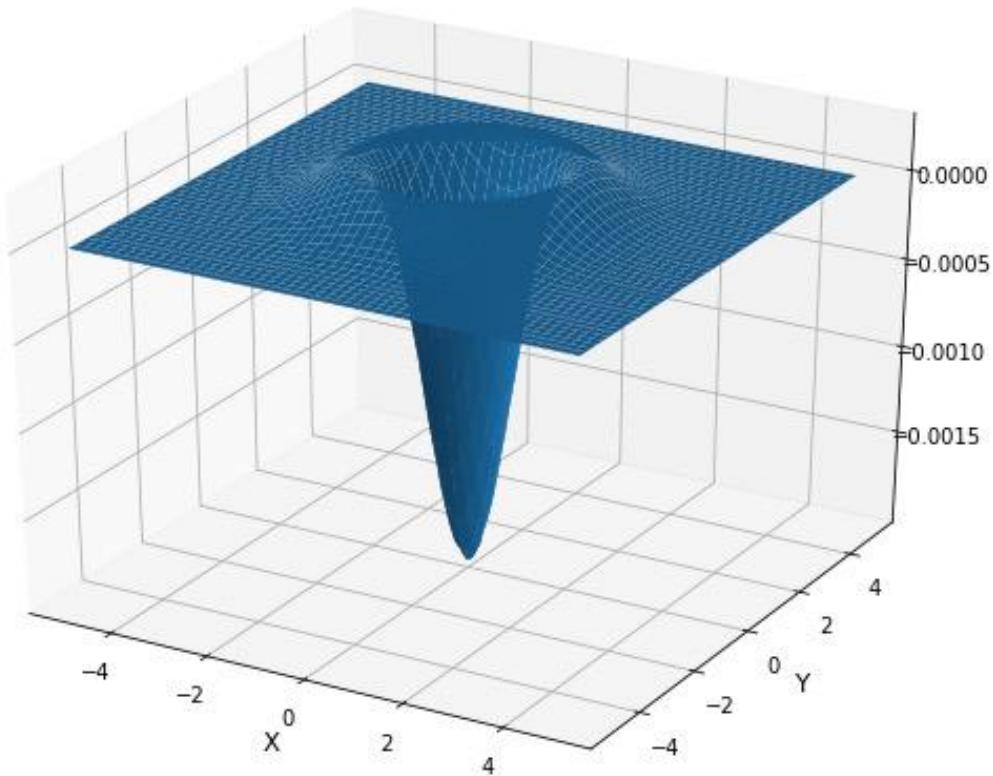
$$\frac{d}{dx}(f * h) = f * \frac{dh}{dx}$$
$$\frac{d^2}{dx^2}(f * h) = f * \frac{d^2h}{dx^2}$$

- Using the **differentiation property of convolution**, we can get the analytical form for the LoG filter,

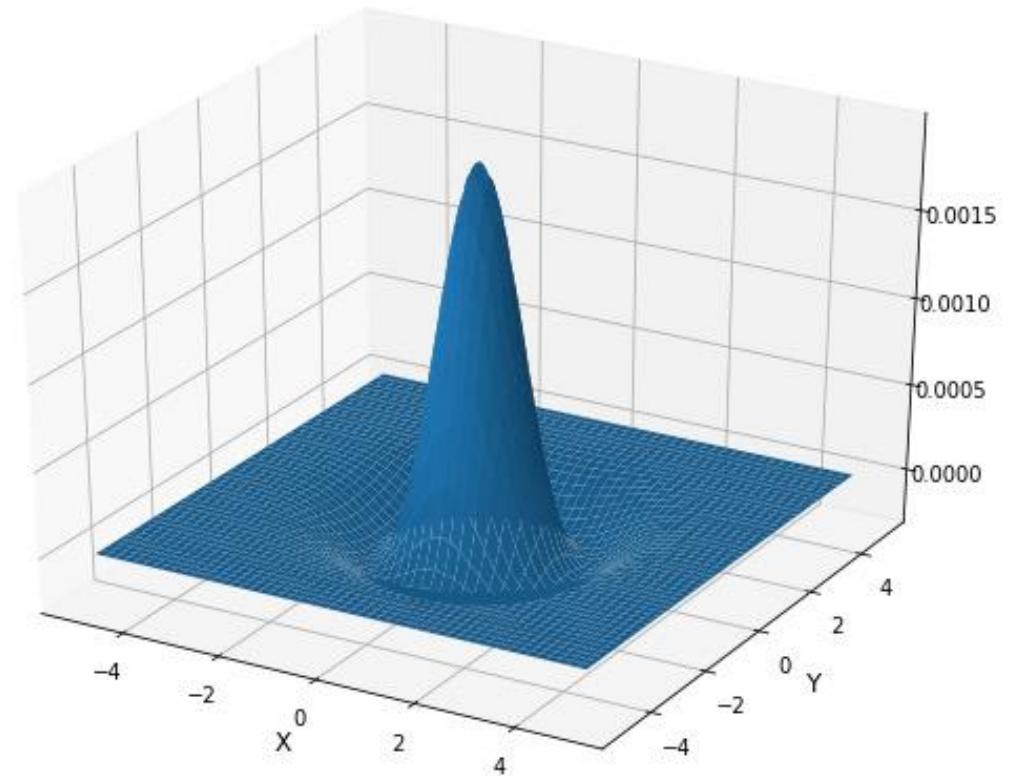
$$\text{LoG: } \Delta(f * h) = \frac{\partial^2(f * h)}{\partial x^2} + \frac{\partial^2(f * h)}{\partial y^2} = f * \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right)$$

- Since 2D Gaussian is formulated as $h(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$, we can derive
$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = -\frac{1}{\pi\sigma^4} \left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$
- The LoG filter essentially performs convolution using the above kernel.

Laplacian of Gaussian (LoG)



LoG filter



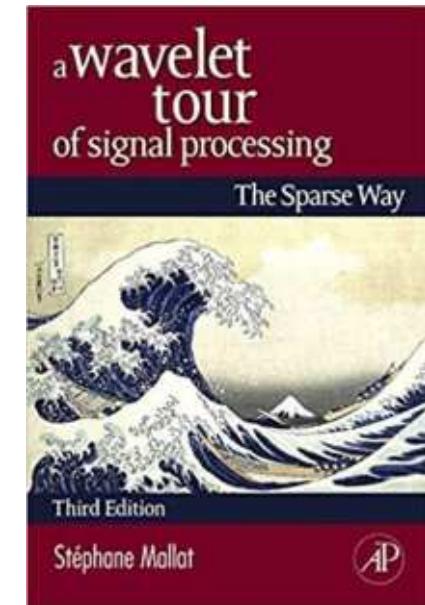
LoG filter (negative)

Another name: Mexican hat

- The negative LoG filter looks like a Mexican hat.
- Also called the Mexican hat wavelet, used in wavelet analysis to extract features from signals.
- Also called Marr's wavelet, named after David Marr.



Mexican hat

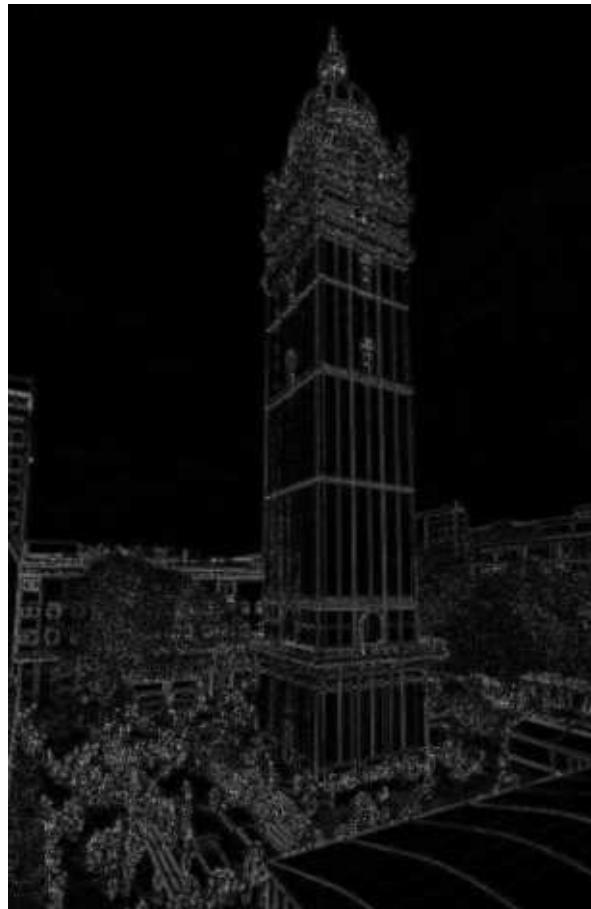


Wavelet is a way to decompose
and analyse signals.

Laplacian of Gaussian (LoG)



Input image



LoG

Laplacian of Gaussian (LoG)

- LoG is a good interest point detector.
 - Similarly to Harris detector, if we want to determine the optimal scale at each pixel, we need to make sure the LoG response is comparable between scales.

- The LoG response at scale σ is

$$LoG(x, y, \sigma) = I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma)$$

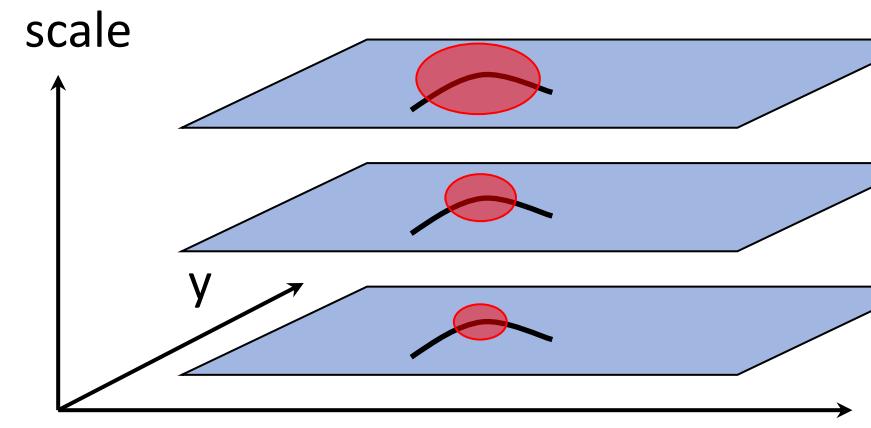
- The normalised LoG response at scale σ is

$$LoG_{norm}(x, y, \sigma) = \sigma^2 (I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$$

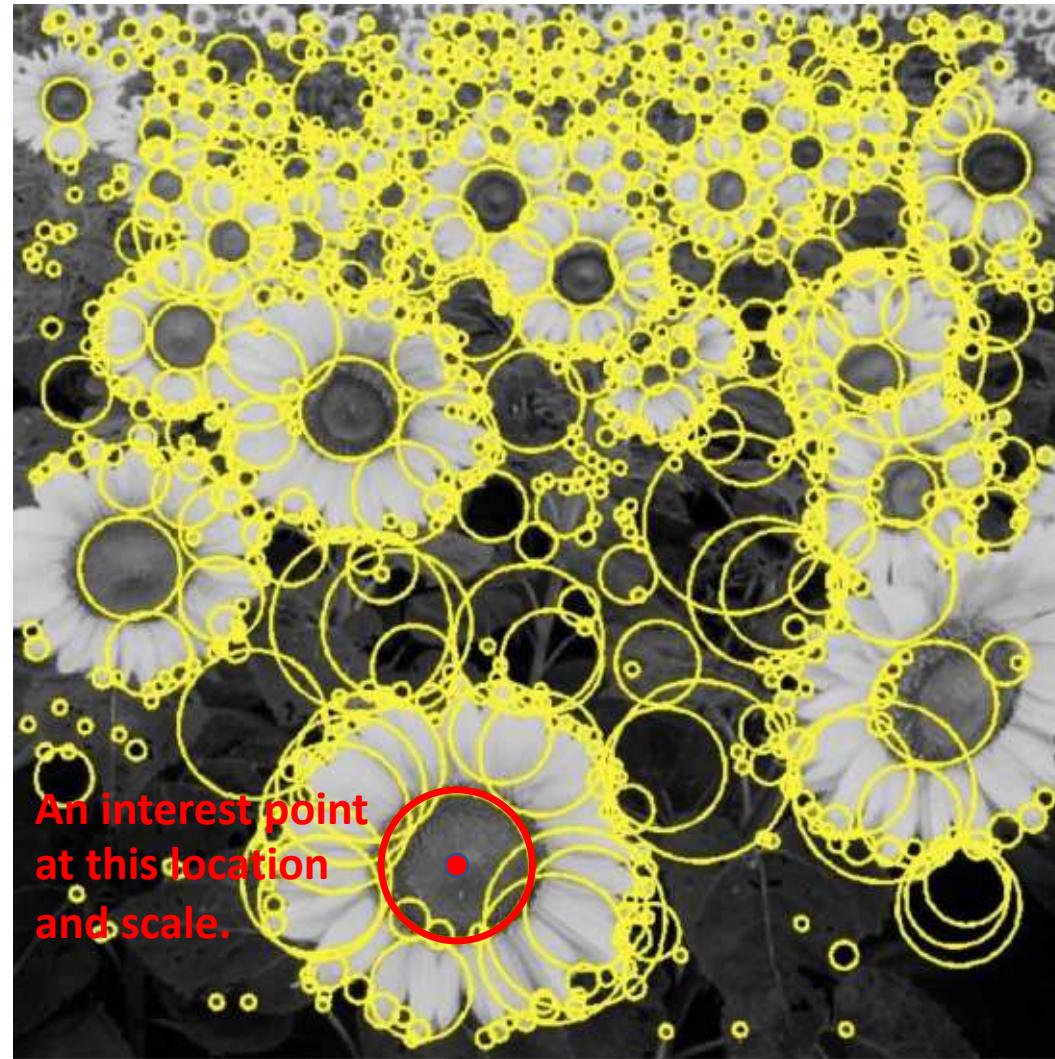
normaliser
Gaussian filter followed
by Laplacian filter

Laplacian of Gaussian (LoG)

- The interest points are detected as local extrema across both scale and space and above a threshold.



Laplacian of Gaussian response at different scales



Detected interest points using Laplacian of Gaussian. Each circle denotes the location of an interest point and its radius denotes its scale.

Difference of Gaussian (DoG)

- Difference of Gaussians (DoG) filter is defined as,

$$DoG(x, y, \sigma) = I * G(k\sigma) - I * G(\sigma)$$

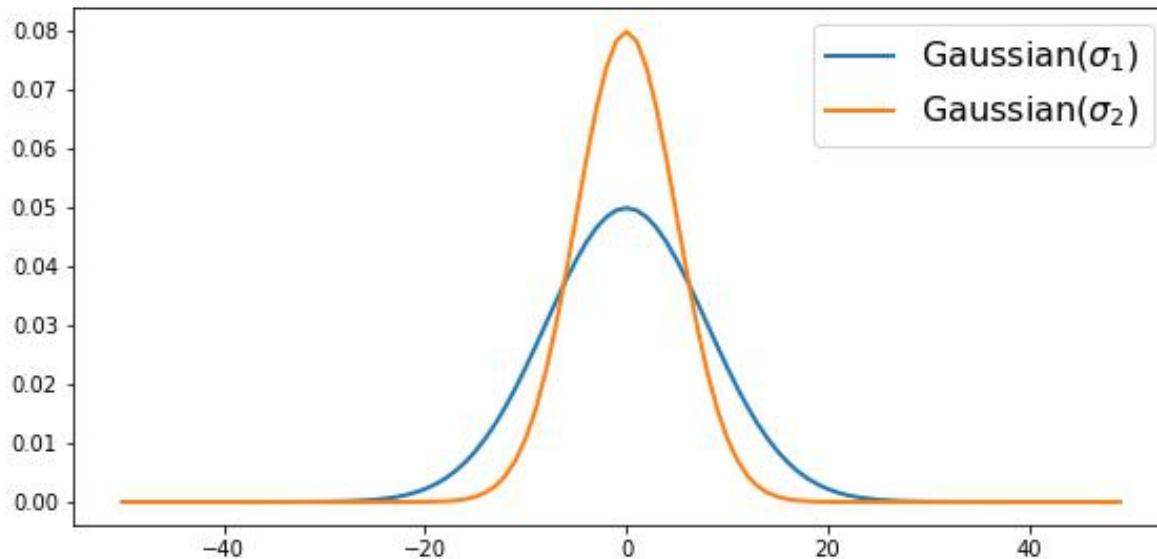
\swarrow \qquad \searrow

Gaussian filters with different scales

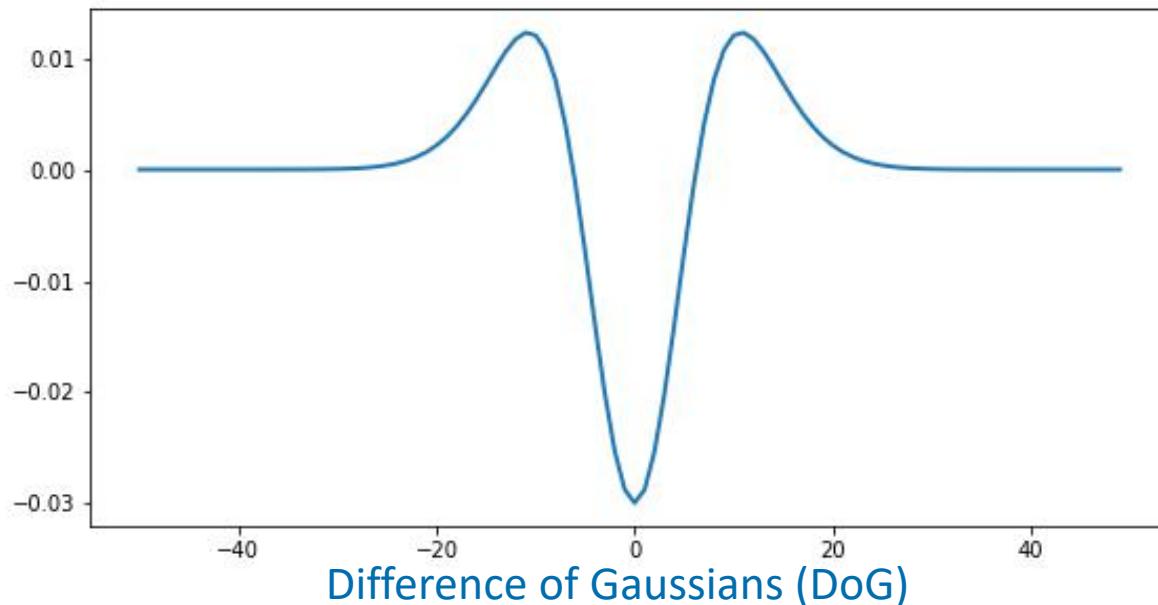
- In Lowe's paper, he suggested $k=\sqrt{2}$.

- DoG approximates the normalised Laplacian of Gaussian (LoG),

$$LoG_{norm}(x, y, \sigma) = \sigma^2(I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$$

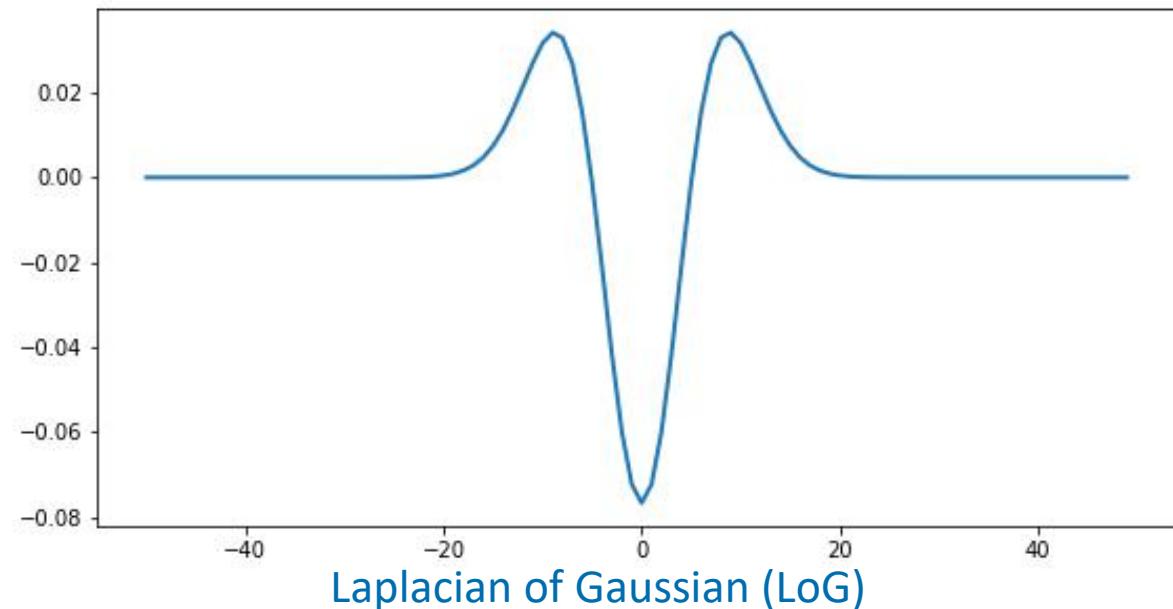


Two Gaussian filters with different scales



Difference of Gaussians (DoG)

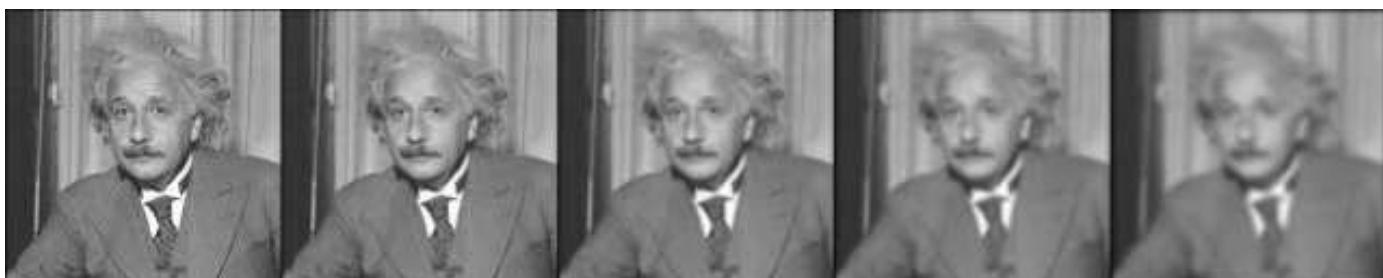
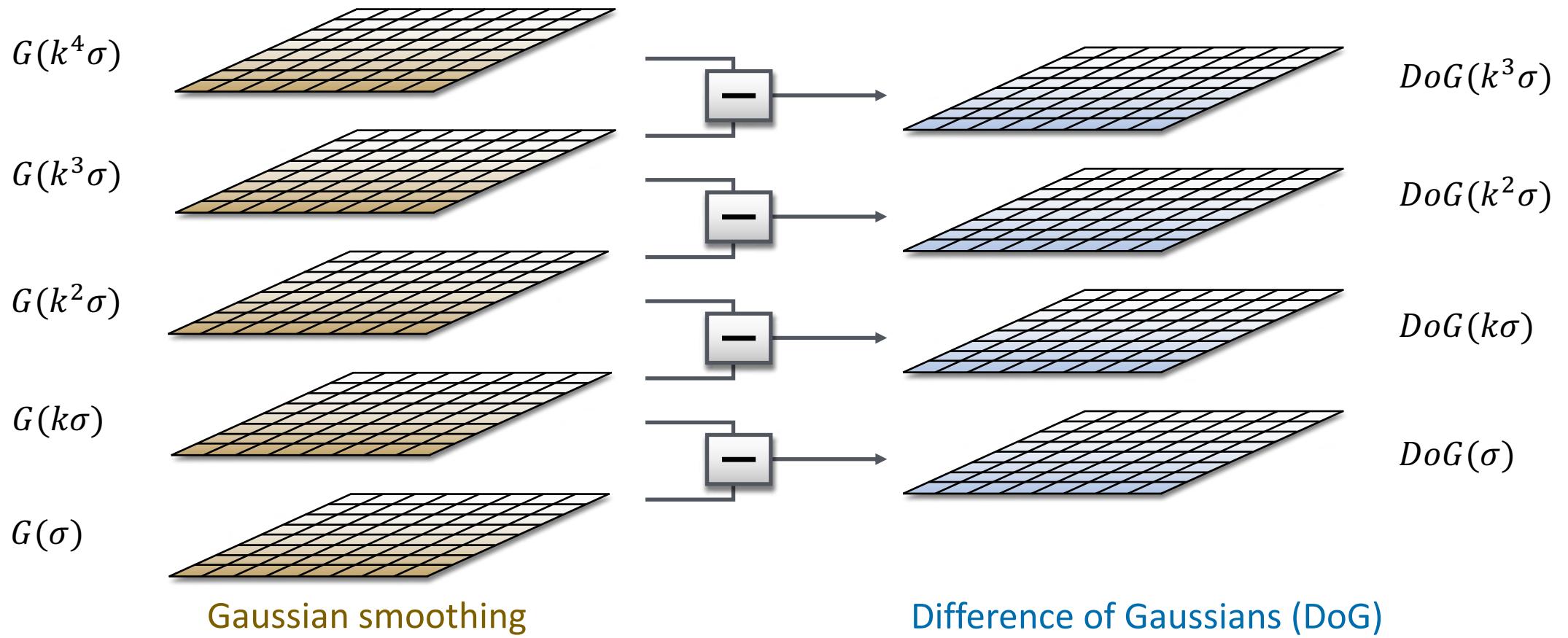
- If you take away $\text{Gaussian}(\sigma_2)$ from $\text{Gaussian}(\sigma_1)$, it looks like the Laplacian of Gaussian.
- DoG is a good approximation to LoG.
 $\text{DoG}(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G(x, y, \sigma)$
as shown in Lowe's paper.



Laplacian of Gaussian (LoG)

Difference of Gaussian (DoG)

- DoG is a good approximation to the normalised Laplacian of Gaussian (LoG).
- It provides some convenience in calculating the response across different scales.



$G(\sigma)$ $G(k\sigma)$ $G(k^2\sigma)$ $G(k^3\sigma)$ $G(k^4\sigma)$



$DoG(\sigma)$ $DoG(k\sigma)$ $DoG(k^2\sigma)$ $DoG(k^3\sigma)$

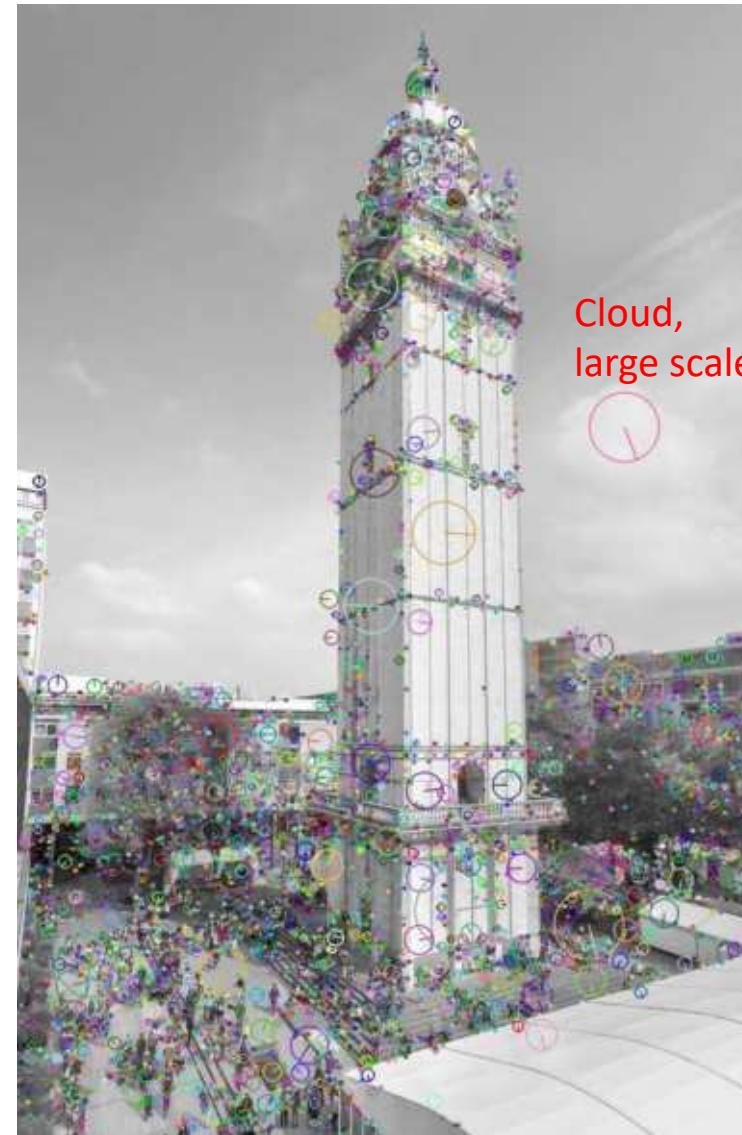
Difference of Gaussian (DoG)

- DoG filters are used in one of the most popular algorithms, called SIFT, which is a pipeline for detecting and describing interest points.



Image

Human,
small scale

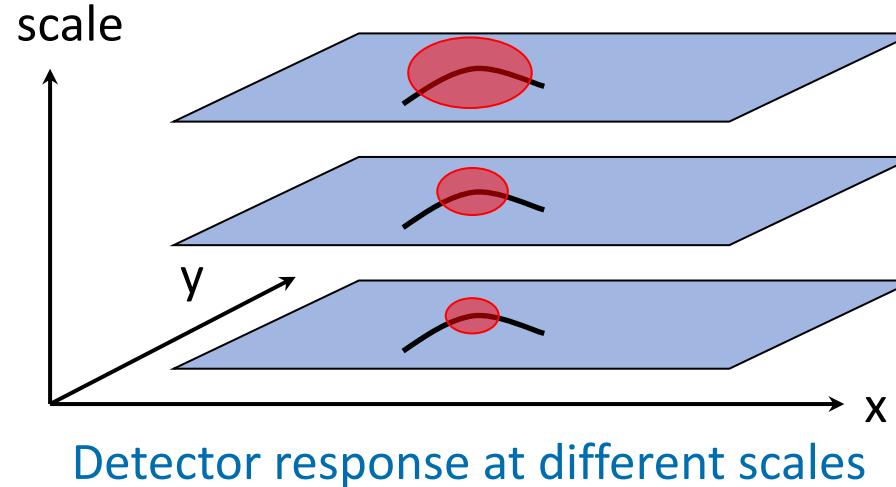


Interest points with associated scales

Interest point detectors

- Scale-variant
 - Harris detector
- Scale-invariant
 - Scale adapted Harris detector
 - Normalised Laplacian of Gaussian
 - Difference of Gaussian
- The scale-invariant detectors follow similar procedures.
 - Calculate the detector response across scales.
 - Find local extrema both across scale and across space.

Interest point detectors



Detector	Response
Scale adapted Harris detector	$\lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$
Normalised Laplacian of Gaussian	$\sigma^2(I_{xx}(x, y, \sigma) + I_{yy}(x, y, \sigma))$
Difference of Gaussian	$I * G(k\sigma) - I * G(\sigma)$

Note: calculation of λ_1, λ_2 depends on σ .

Next

- Now we can detect interest points in images, with a scale associated with each interest point.
- Suppose the next task is image matching.
- How do we know an interest point in one image correspond to another interest point in another image?
- We need to describe the features for the interest points.

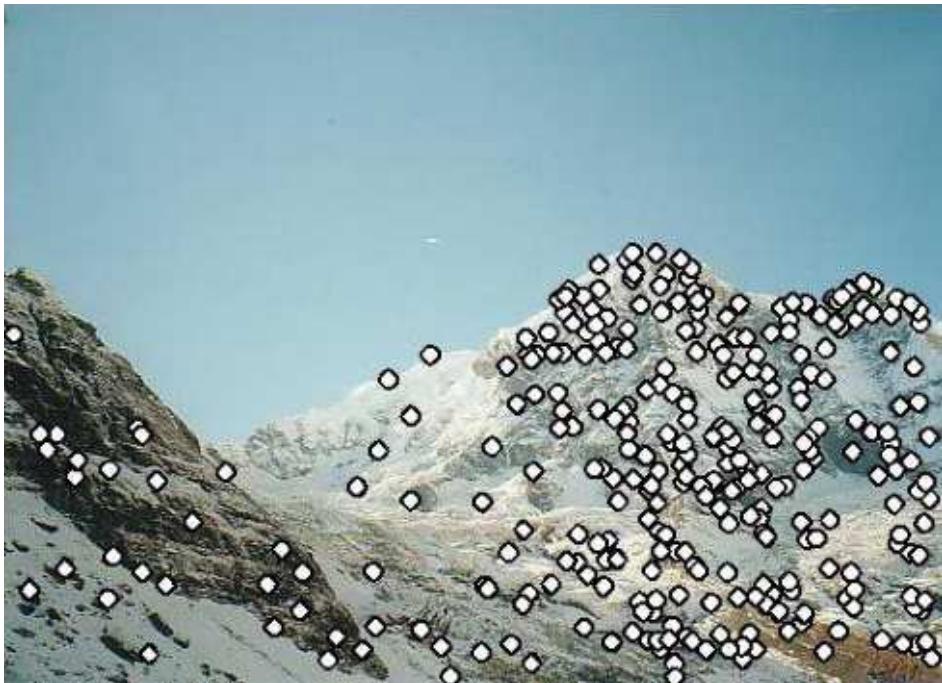
References

- Sec. 4.1.1 Feature detectors. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Feature Description I

Dr Wenjia Bai

Department of Computing & Brain Sciences



- Now we know how to detect interest points on these two images.
- How do we know which point corresponds to which?
- We need to describe the feature for each point so that they can be compared and matched.

In this lecture

- Given a point on an image, how do we describe the local feature or content around this point?
- After feature description, how do we match interest points between two images?

Feature description

- Simple descriptors
 - Pixel intensity
 - Patch intensities
 - Gradient orientation
 - Histogram
- SIFT descriptor

Pixel intensity

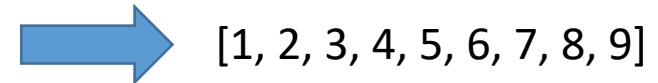
- Simply using the intensity of a single pixel.
- Problems
 - Sensitive to absolute intensity value.
 - The intensity of the same point will change under different illuminations.
 - Not very discriminative.
 - The grayscale intensity ranges from 0 to 255. There are thousands of pixels with exactly the same intensity.
 - A single pixel does not represent any local content (edge, blob, peak of the mountain etc.).

199	192	158	111	110	123	130	130
189	149	108	111	113	120	126	125
130	100	98	108	113	113	114	120
85	100	96	104	108	107	101	94
85	95	98	96	100	103	100	96
79	94	87	77	69	70	87	84
77	80	72	71	60	52	59	64
68	67	63	58	53	51	54	52

Patch intensities

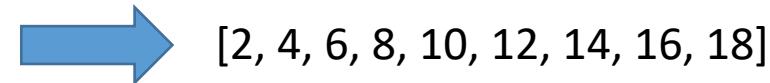
- Better than a single pixel.
 - Represent the local pattern.
 - Perform well if the images are of similar intensities and roughly aligned [1,2].
- Problems
 - Sensitive to absolute intensity value
 - Not rotation-invariant

1	2	3
4	5	6
7	8	9



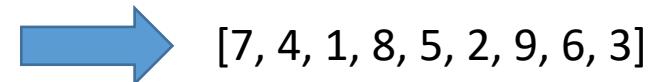
Patch representation

2	4	6
8	10	12
14	16	18



Problem: if intensity changes

7	4	1
8	5	2
9	6	3



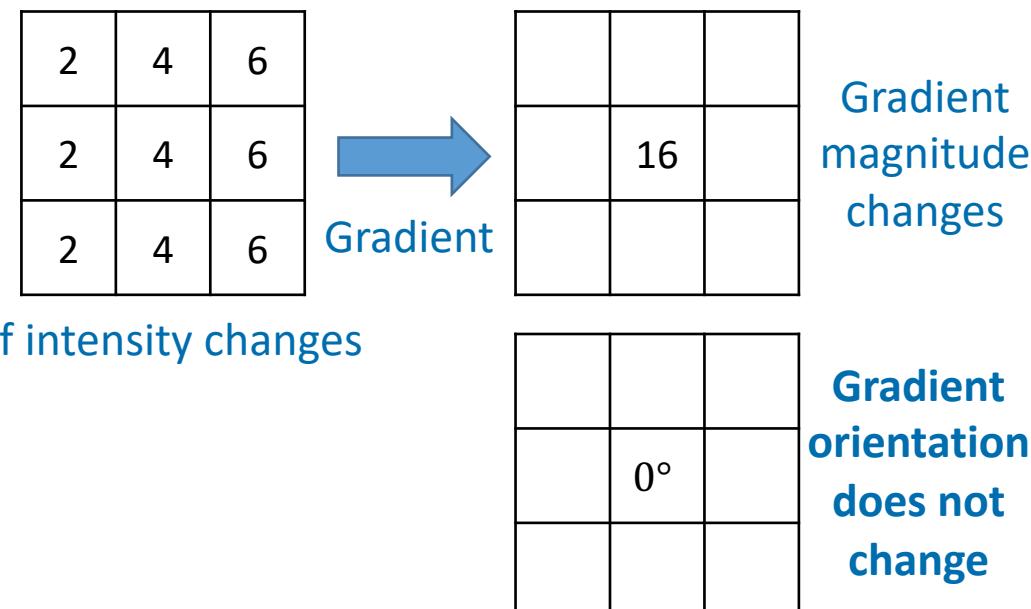
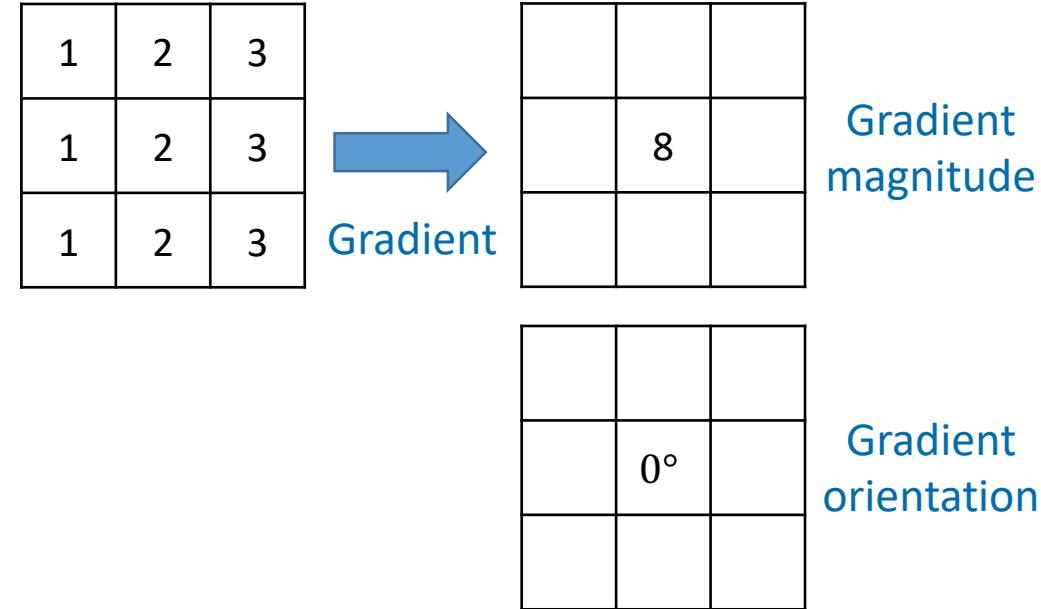
Problem: if being rotated

[1] C. Barnes et al. PatchMatch: A randomized correspondence algorithm for structural image editing, SIGGRAPH 2009.

[2] P. Coupe et al. Patch-based segmentation using expert priors. NeuroImage, 2011.

Gradient orientation

- Although gradient magnitude is sensitive to intensity changes, its orientation is robust.
- Problem
 - Not rotation-invariant.



Histogram

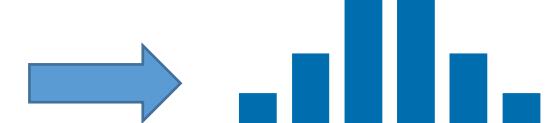
- The histogram of pixel intensities in a patch
 - Robust to rotation
 - Robust to scaling
- Problem
 - Sensitive to intensity changes

1	2	3
4	5	6
7	8	9



Histogram representation

7	4	1
8	5	2
9	6	3



If being rotated

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



If being scaled

Thoughts

- Gradient orientation
 - Robust to change of absolute intensity values
- Histogram
 - Robust to rotation and scaling
- Can we combine the advantages of the two?
 - Yes.
 - Scale-invariant feature transform (SIFT) algorithm.

Scale-invariant feature transform (SIFT)

- SIFT is an algorithm for detecting and describing local features in images.
- SIFT transforms an image into a large set of interest points, each of which is described by a feature vector that is invariant to scaling and rotation and change of illuminations.

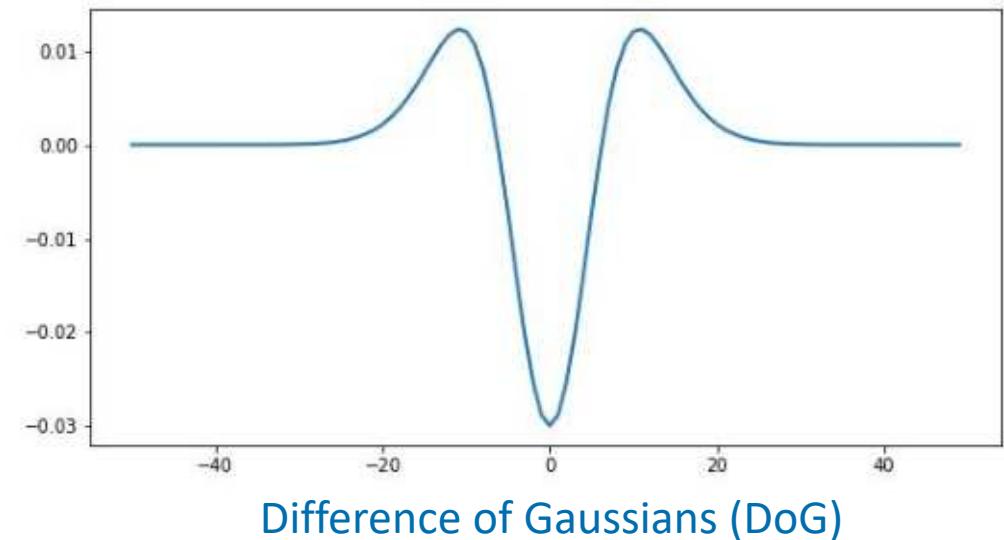
Scale-invariant feature transform (SIFT)

- SIFT consists of the following steps:

1. Detection of scale-space extrema
 2. Keypoint localization
 3. Orientation assignment
 4. Keypoint descriptor
- Detection
- Description

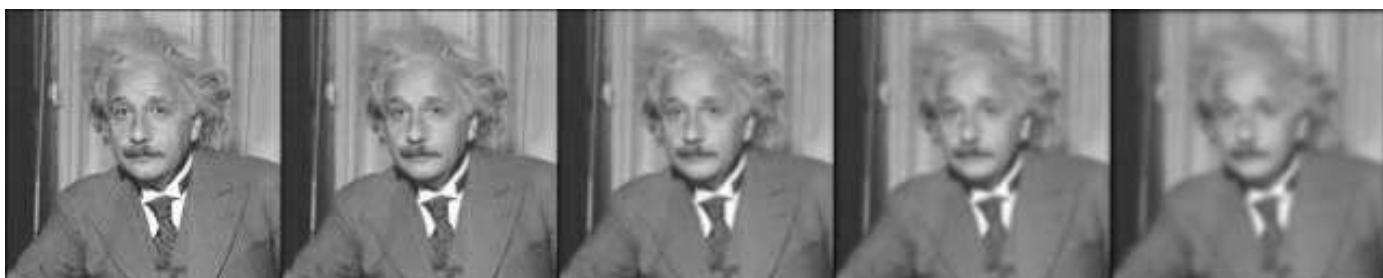
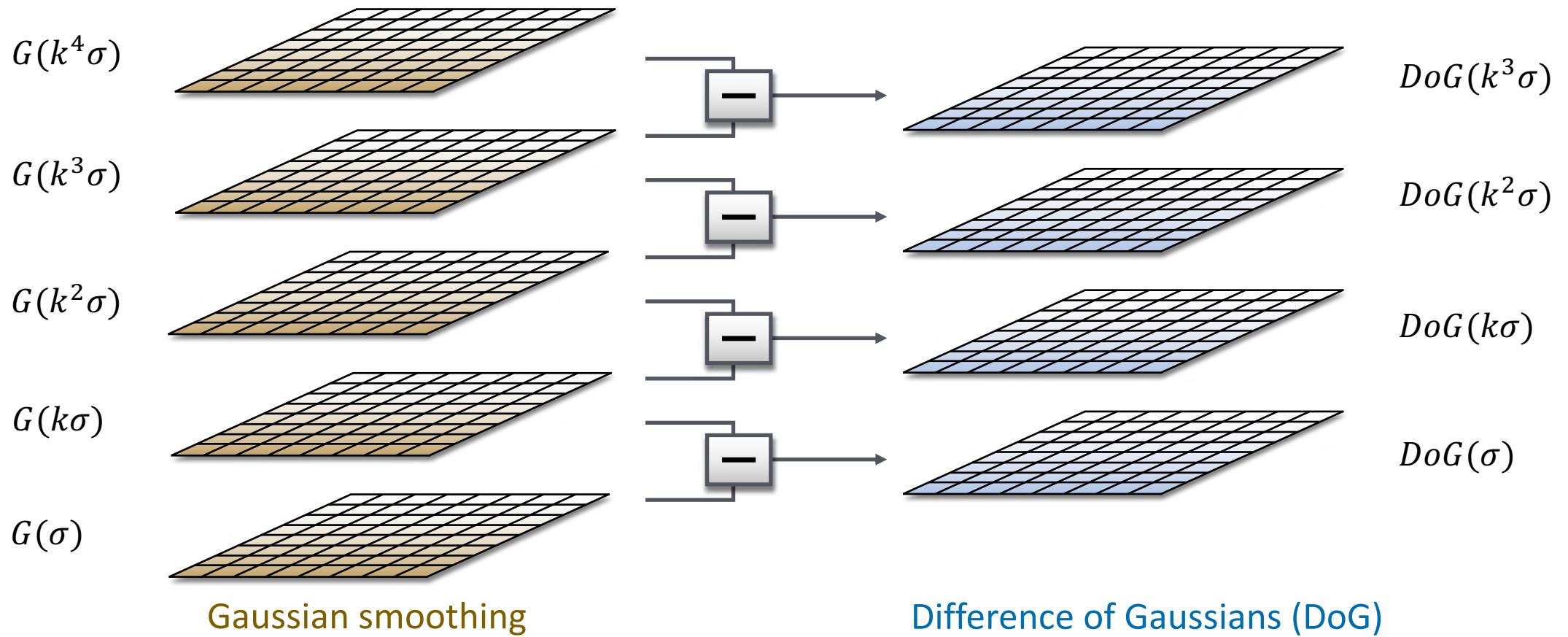
Detection of scale-space extrema

- The first step is to search across scales and pixel locations, looking for interest points.
- The difference of Gaussian filter is used to identify potential interest points.
$$DoG(x, y, \sigma) = I * G(k\sigma) - I * G(\sigma)$$
- It is similar to Laplacian of Gaussian.



Detection of scale-space extrema

- We calculate the difference of Gaussian for different scales σ , from fine to coarse.
 - For example, $\sigma, \sqrt{2}\sigma, 2\sigma, 2\sqrt{2}\sigma, 4\sigma, \dots$
 - It is implemented as first calculating a stack of Gaussian smoothed images, then calculating their differences.

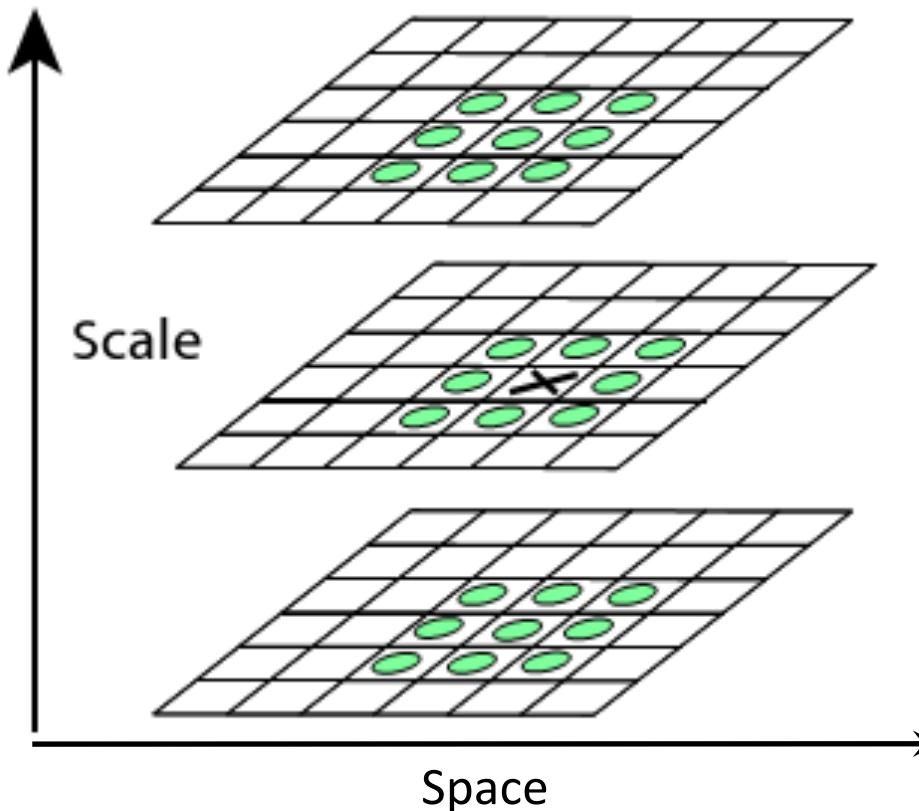


$G(\sigma)$ $G(k\sigma)$ $G(k^2\sigma)$ $G(k^3\sigma)$ $G(k^4\sigma)$



$DoG(\sigma)$ $DoG(k\sigma)$ $DoG(k^2\sigma)$ $DoG(k^3\sigma)$

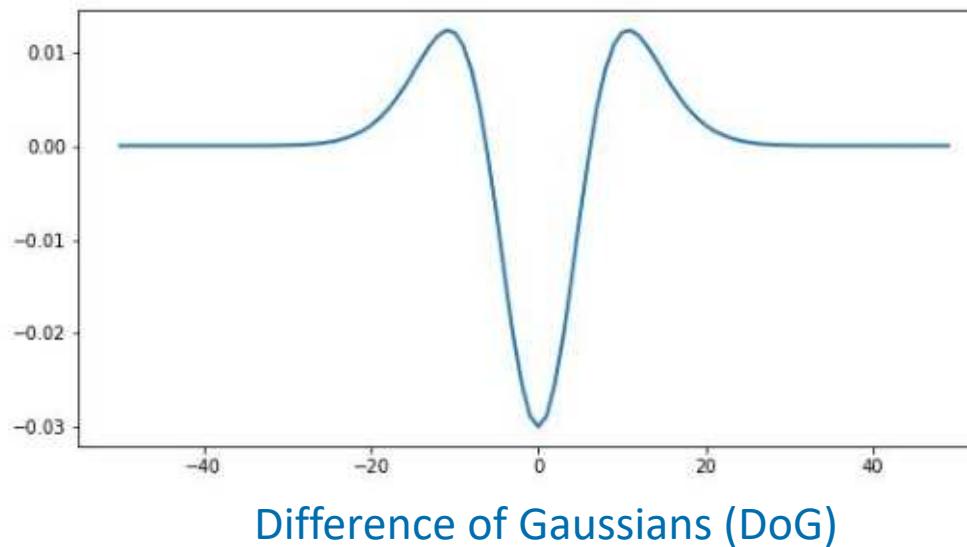
Detection of scale-space extrema



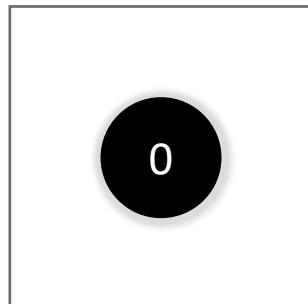
X is detected as an interest point if it is a local extremum both along scale dimension (most appropriate scale) and across space. A threshold may also be applied.

Detection of scale-space extrema

- We use the term extrema here, which means both minima and maxima.



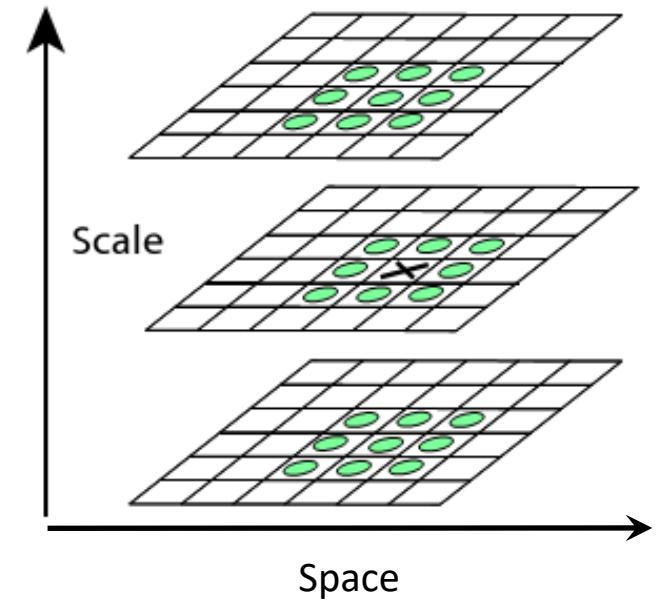
Negative response,
minimum at the
blob centre.



Positive response,
maximum at the
blob centre.

Keypoint localisation

- The initial version of SIFT [1] locates the keypoints at the scale-space extrema.
- An improved version [2] refines the localisation and scale by fitting a model onto to nearby data, which can achieve sub-pixel accuracy.

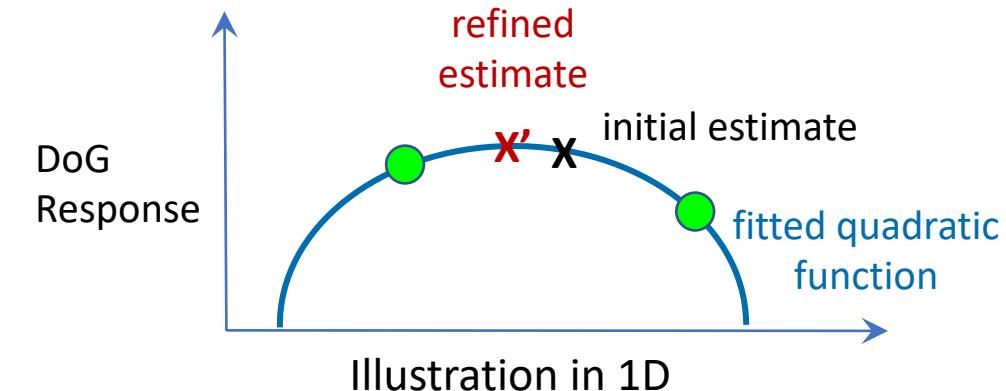
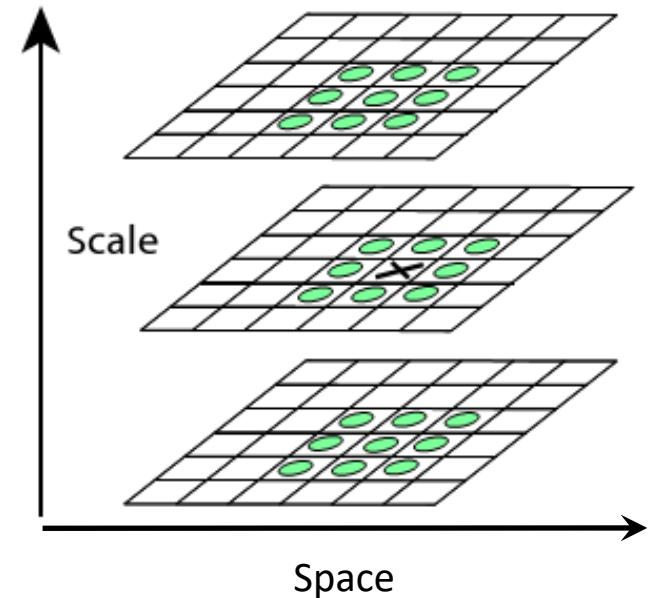


[1] D. Lowe. Object recognition from local scale-invariant features, ICCV 1999.

[2] D. Lowe. Distinctive image features from scale-invariant keypoints, IJCV 2004.

Keypoint localisation

- A quadratic function is fitted to the DoG response of neighbouring pixels.
- Then the location and scale of extremum for this quadratic function can be estimated.



Keypoint localisation

- Let us denote the DoG response by $D(x, y, \sigma)$ or $D(\mathbf{x})$, where $\mathbf{x} = (x, y, \sigma)^T$ is a 3D vector, containing both location and scale.
- By Taylor expansion, we have

$$D(\mathbf{x} + \Delta\mathbf{x}) = D(\mathbf{x}) + \frac{\partial D^T}{\partial \mathbf{x}} \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \Delta\mathbf{x}$$

We omit the higher order terms

|

The first derivatives can be estimated using finite difference, e.g.
$$\frac{\partial D}{\partial x} = \frac{D(x+1, y, \sigma) - D(x-1, y, \sigma)}{2}$$

|

The second derivatives or Hessian can be estimated using finite difference as well.

This is a quadratic function for variable $\Delta\mathbf{x}$. Here \mathbf{x} is the initial estimate, $\Delta\mathbf{x}$ is the shift to the refined estimate.

Keypoint localisation

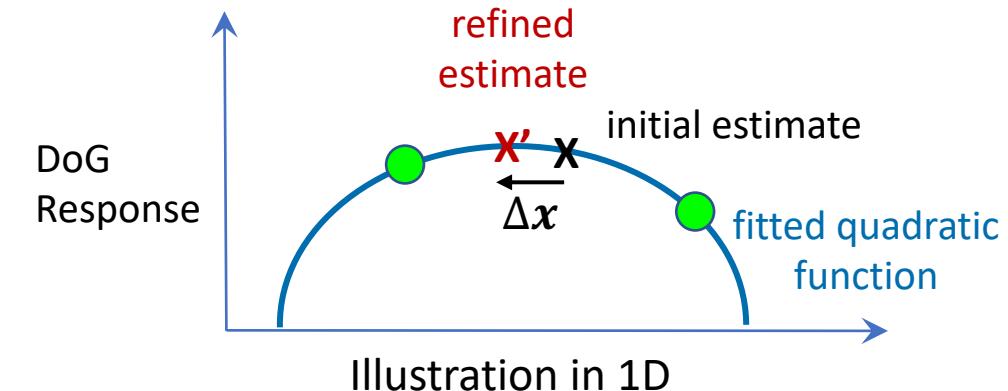
- To estimate a refined extrema for this function, we can let

$$\frac{\partial D(x + \Delta x)}{\partial \Delta x} = \frac{\partial D}{\partial x} + \frac{\partial^2 D}{\partial x^2} \Delta x = 0$$

- Therefore,

$$\Delta x = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

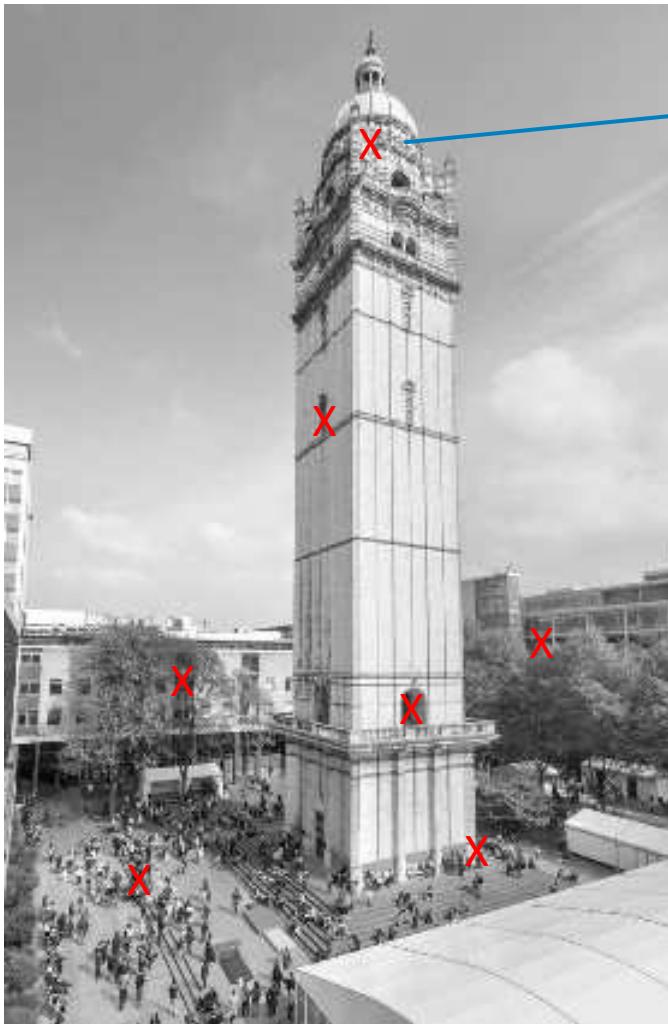
- We move from x by Δx and arrive at refined estimate.



Keypoint localisation

- Note that $\mathbf{x} = (x, y, \sigma)^T$ is a 3D vector, containing both location and scale.
- We get refined estimates for both location (x, y) and scale σ .
 - Location: sub-pixel accuracy
 - Scale: some value between these scales $\sigma, \sqrt{2}\sigma, 2\sigma, 2\sqrt{2}\sigma, 4\sigma, \dots$

Keypoint localisation



How do we describe the feature for each **keypoint**?

Remember the idea of gradient orientation and histogram?

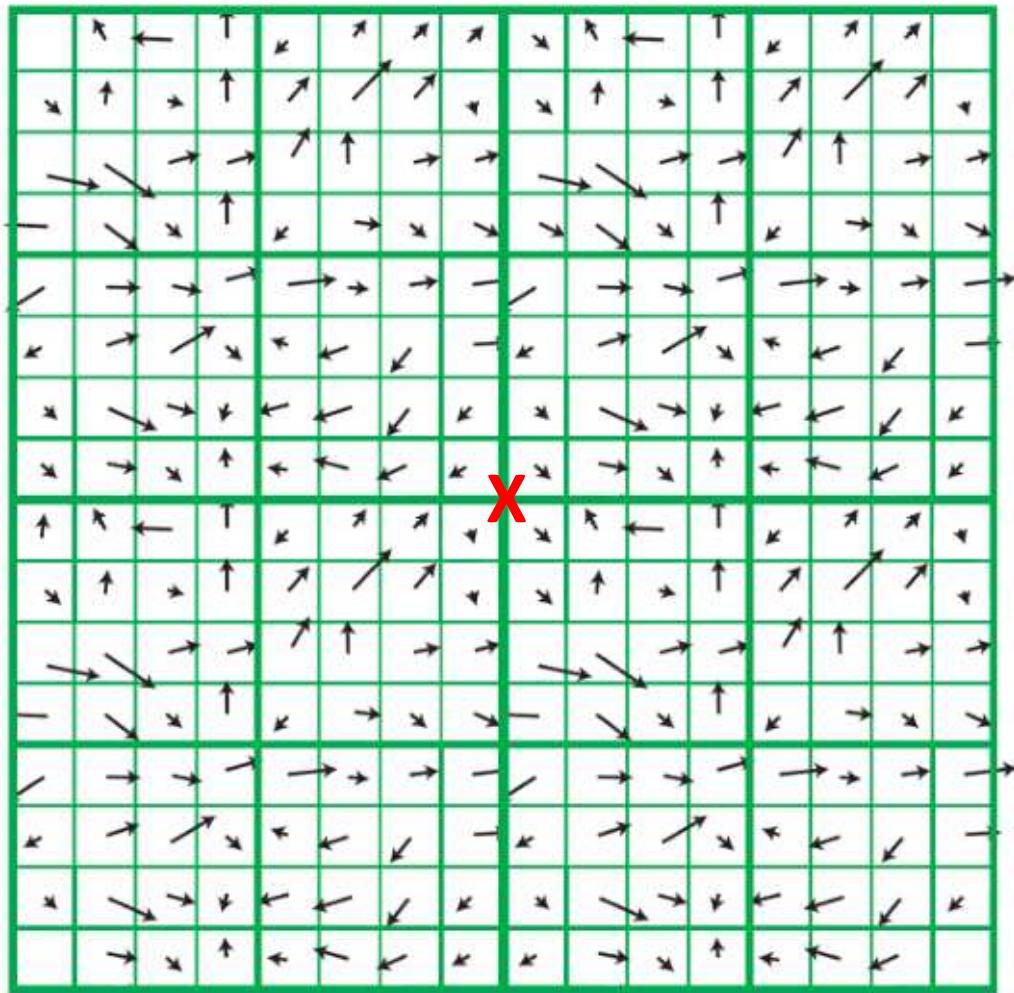
Feature description



We draw sample points in the local region, calculate their gradient orientations and form a histogram.

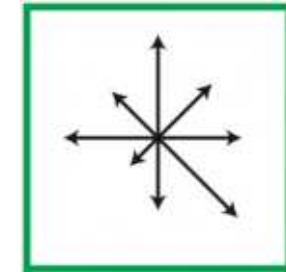
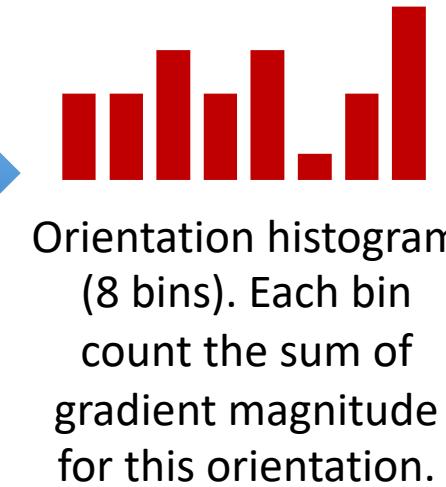
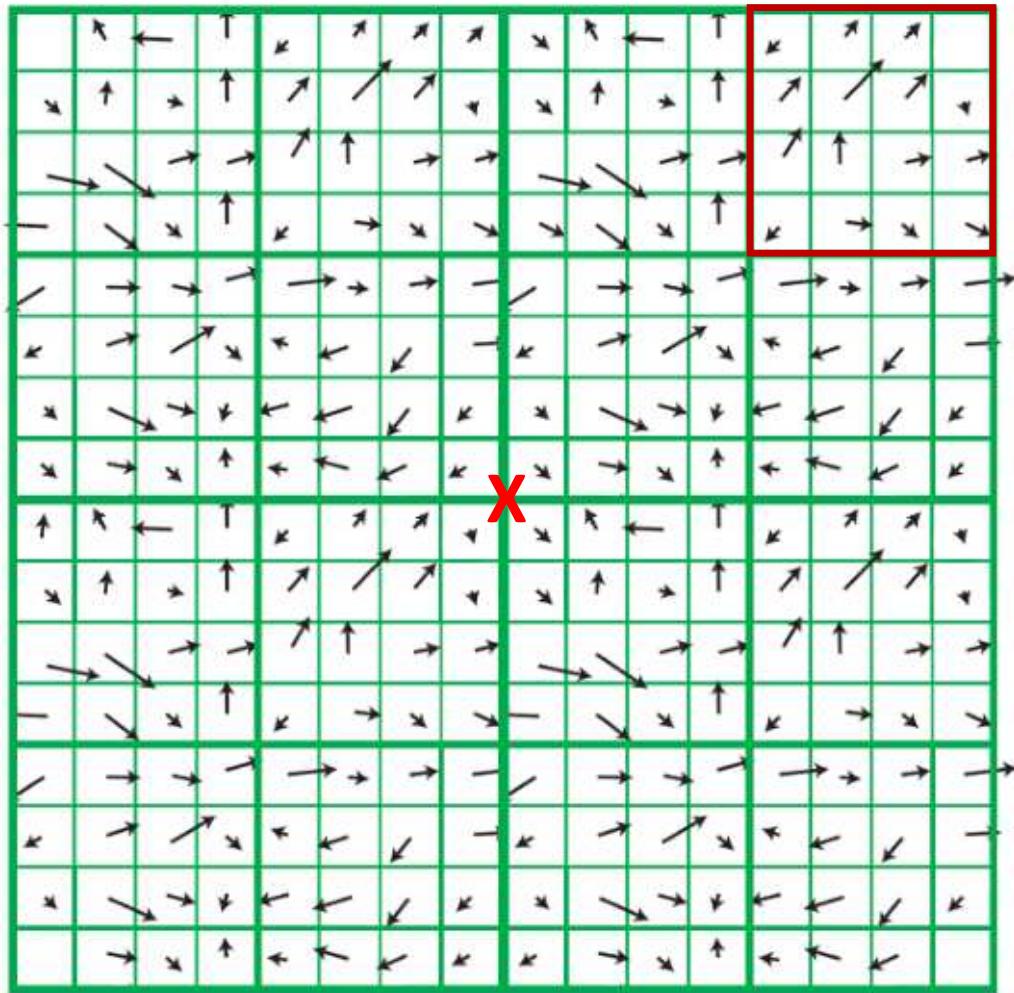
This is basically what SIFT does. The implementation is slightly more complex.

Local image descriptor

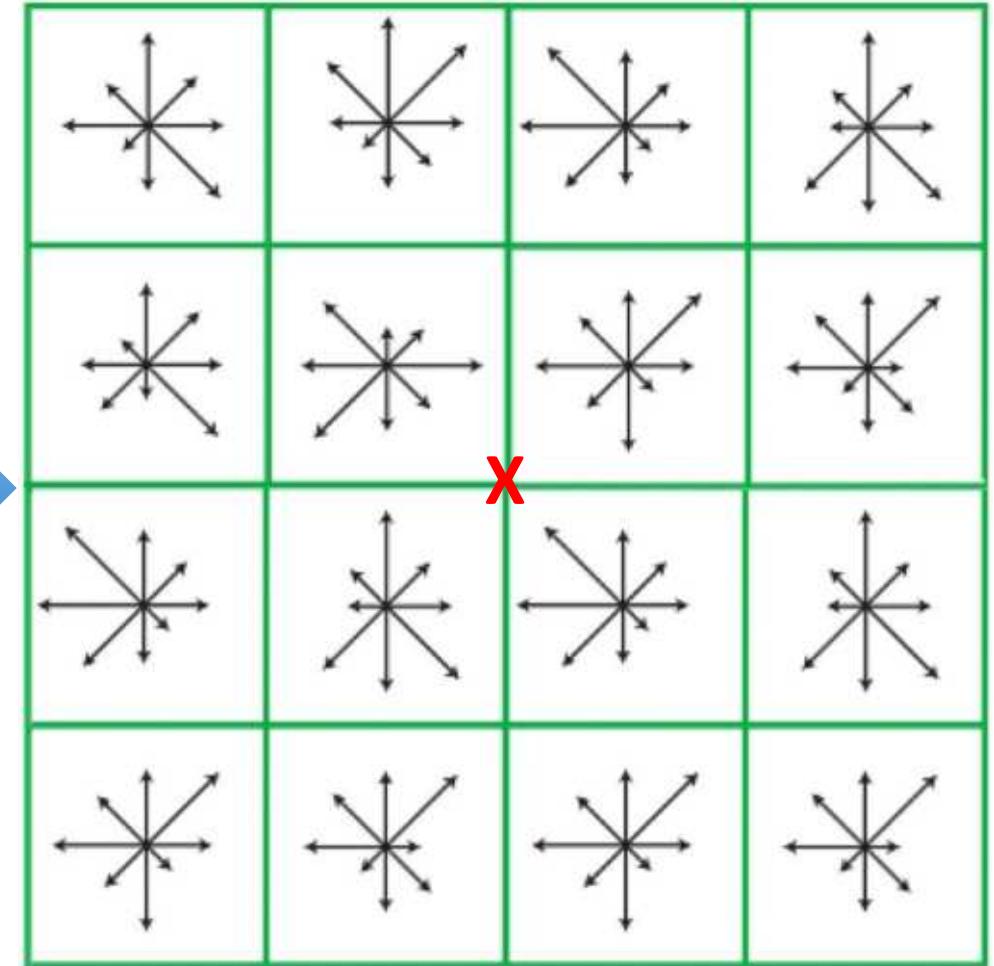
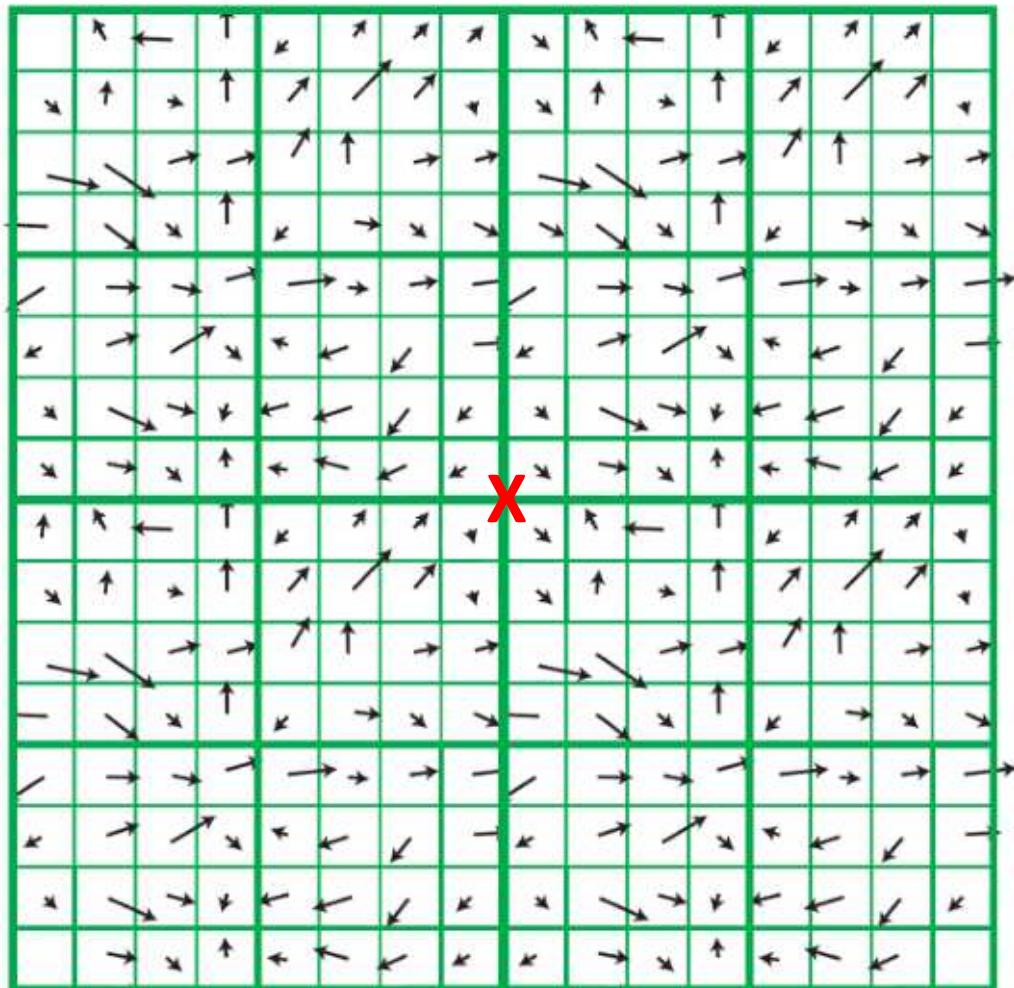


- 16 subregions describe features at different parts of the local area.
- Within each region, the gradient orientation of sample points are calculated.
- A histogram is calculated for each subregion with 8 bins (orientations of 0° , 45° , 90° , ... 315°).

Local image descriptor



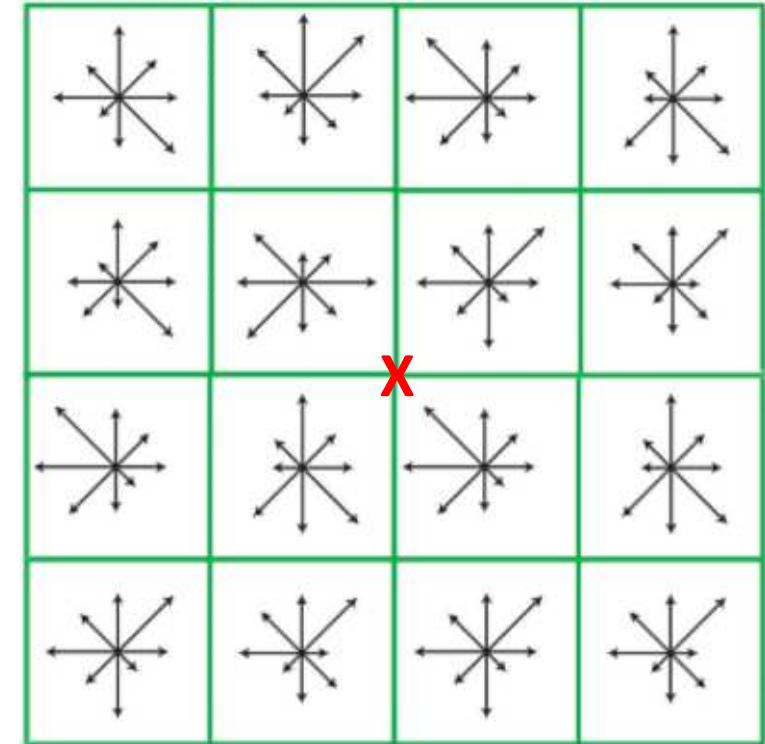
Local image descriptor



SIFT feature descriptor

Local image descriptor

- In Lowe's implementation, 16 subregions are used, so the final SIFT descriptor looks like this.
- What is the dimension of this descriptor?
 - 16 subregions x 8 bins = 128
- We use a feature vector of 128 elements to describe each keypoint.



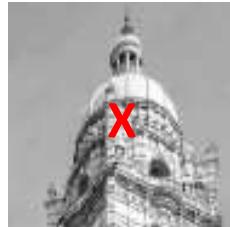
SIFT feature descriptor

[1] D. Lowe. Object recognition from local scale-invariant features, ICCV 1999.

[2] D. Lowe. Distinctive image features from scale-invariant keypoints, IJCV 2004.

Scale and rotation invariance

- One last problem is to make this descriptor invariant to scaling and rotation.



Should describe the same feature.

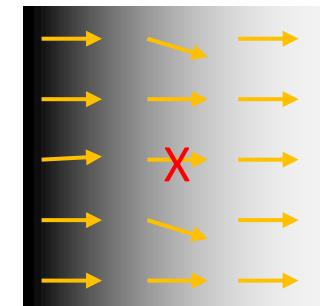


Scale and rotation invariance

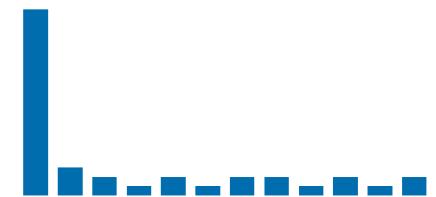
- We need to know the scale and orientation at this keypoint.
 - Scale is already known when we detect this keypoint.
 - We just need to know the **dominant orientation**.

Orientation assignment

- We can determine the **dominant orientation** for a keypoint.
- The gradient orientations of all pixels in this neighbourhood vote for the dominant orientation.
 - An orientation histogram with 36 bins covering 360 degrees is created.
 - Each pixel votes for an orientation bin, weighted by the gradient magnitude.
 - The keypoint will be assigned an orientation corresponding to the maximum of the histogram.



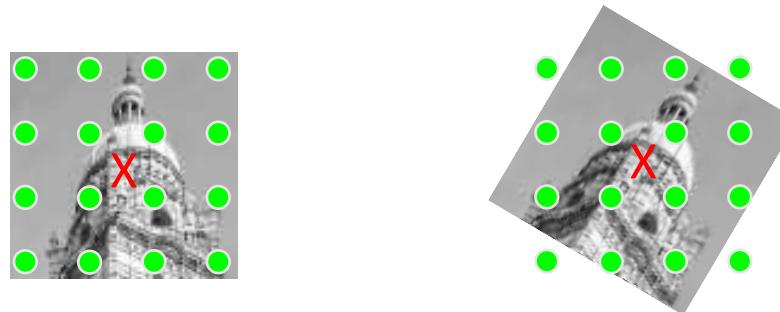
Gradient orientations in the neighbourhood of keypoint X.



Orientation histogram. For example, the pixels vote for the orientation of 0° .

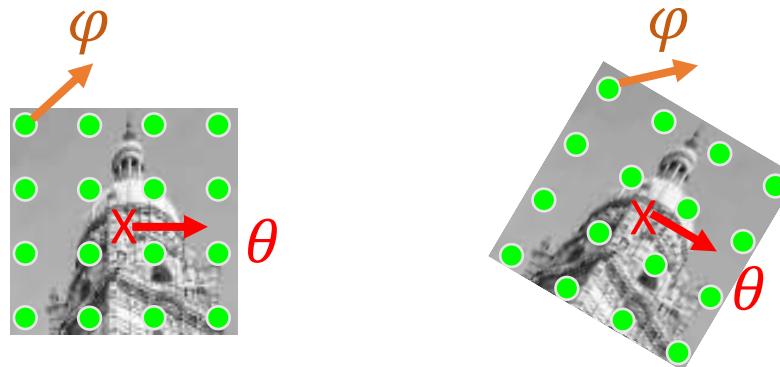
Orientation assignment

- Two uses of the dominant orientation
 - We can draw sample points relative to this orientation.



Orientation assignment

- Two uses of the dominant orientation
 - We can draw sample points relative to this orientation.
 - When calculating the SIFT features, the gradient orientation of each point is relative to the dominant orientation.

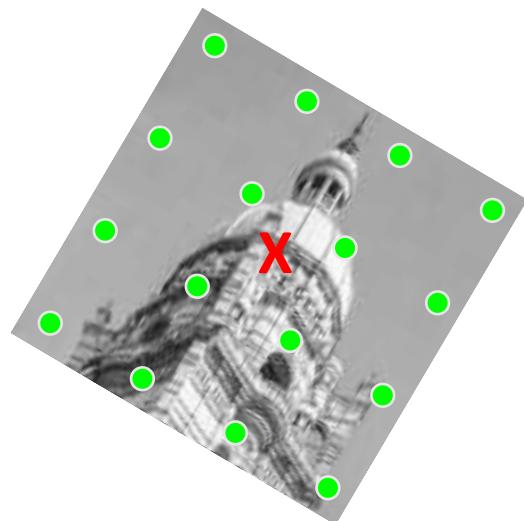
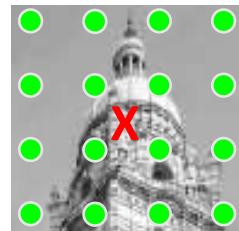


Sampling is performed in a rotated coordinate system.

The gradient orientation of each point φ needs to be adjusted by θ , i.e. $\varphi - \theta$. Here θ denotes the dominant orientation.

Scale and rotation invariance

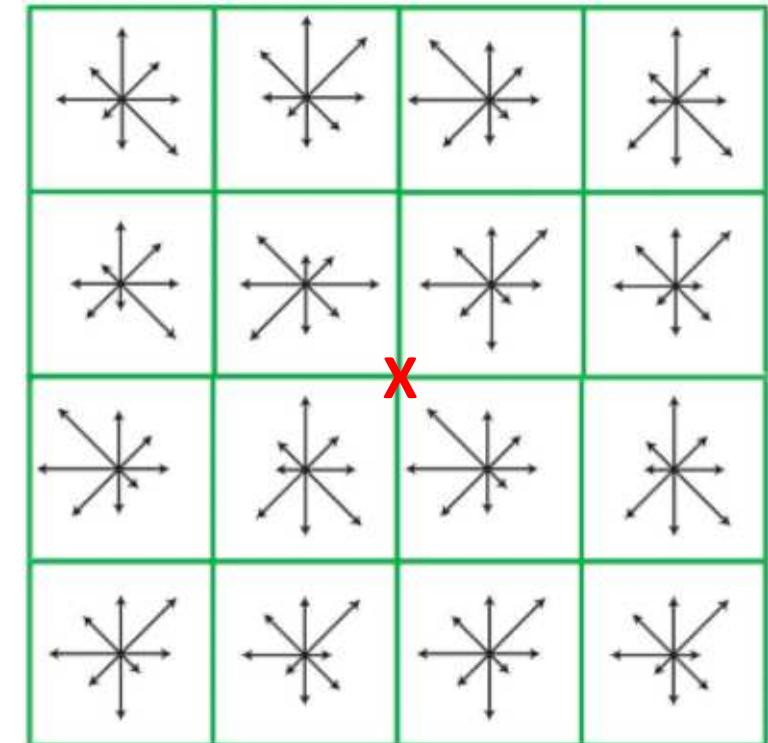
- The size of the sampled region is proportional to the scale of the keypoint.



- Now the sample points are corresponding between two images, thanks to the knowledge of scale and dominant orientation.
- Also, the gradient orientations of the sample points are consistent, due to the adjustment by dominant orientation.
- Therefore, the descriptor is invariant to scale and rotation.

SIFT image descriptor

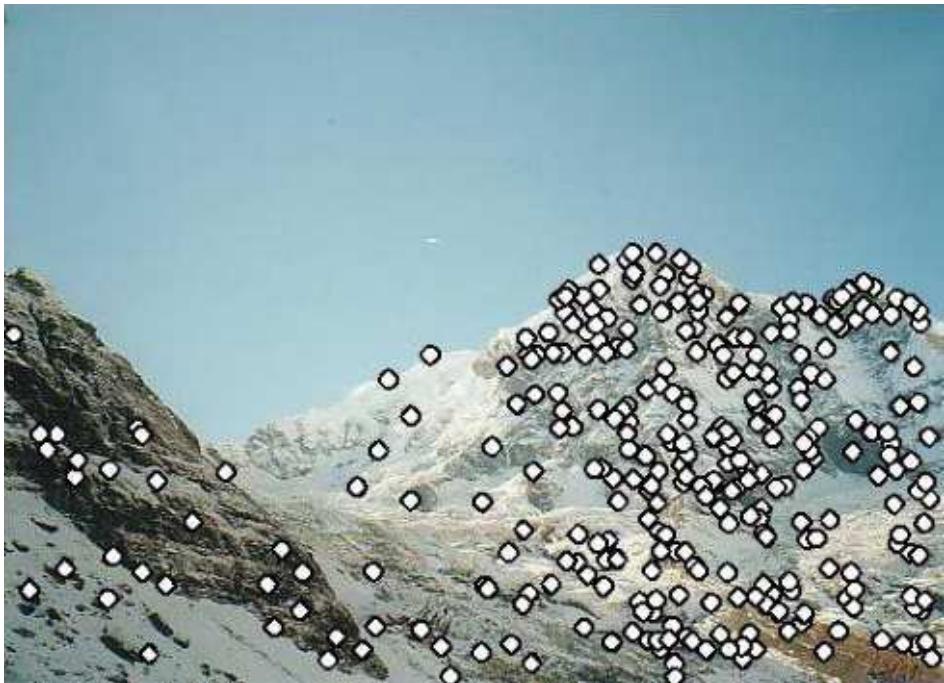
- This descriptor is robust to scaling, rotation and change of illuminations, due to
 - consideration of dominant orientation
 - a sampling window proportional to scale
 - use of gradient orientations for feature description
 - use of histograms



SIFT feature descriptor

[1] D. Lowe. Object recognition from local scale-invariant features, ICCV 1999.

[2] D. Lowe. Distinctive image features from scale-invariant keypoints, IJCV 2004.



- Now we have a good feature descriptor for each interest point in the two images.
- How do we know which point corresponds to which?

Keypoint matching

- Keypoints between two images are matched by identifying the nearest neighbours.
 - For each keypoint in image A, identify its nearest neighbours in the set of keypoints for image B.
 - The distance is defined by the Euclidean distance of SIFT descriptors.
- For efficiency, we may not need to find the exact nearest neighbour.
 - There are fast algorithms for finding approximate nearest neighbours.

Keypoint matching

- After nearest neighbour search, we find that a keypoint (x, y) in image A corresponds to another keypoint (u, v) in image B.
 - Assume that they are related by an affine transformation,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

|
|

rotation, scaling ...
translation

Keypoint matching

- For a number of pairs of corresponding keypoints, the equation can be written as,

$$\begin{array}{l} \text{keypoint 1} \\ \text{keypoint 2} \\ \text{keypoint ...} \end{array} \begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ & & & \vdots & & \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \end{bmatrix}$$

- It can be written as a linear system,

$$Am = b$$

|
unknown affine
parameters

Keypoint matching

- The least-square solution to the linear system is given by,

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

|

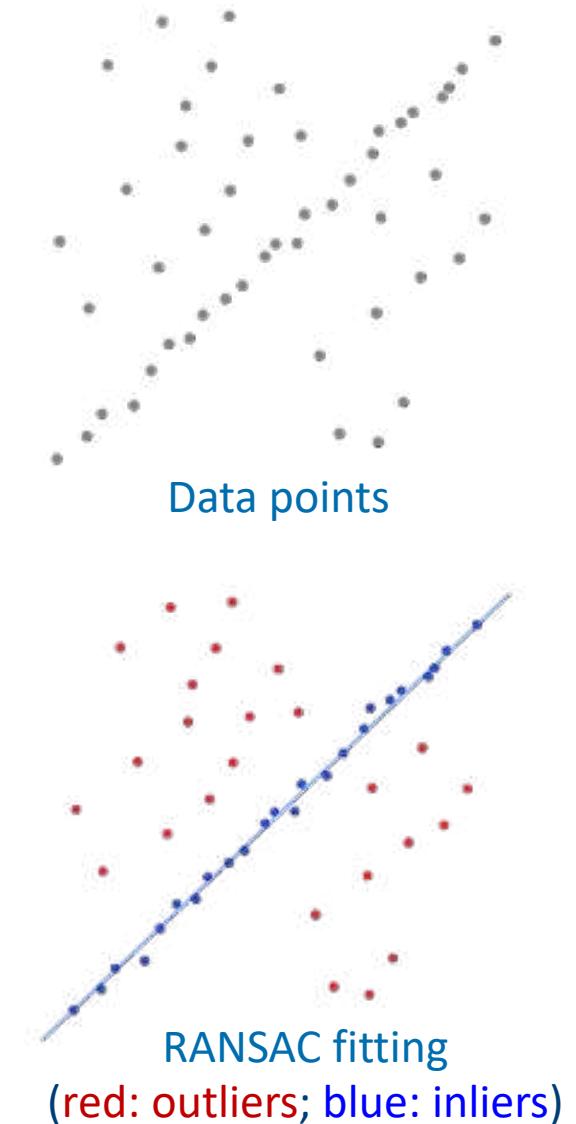
Moore-Penrose inverse

which minimises the squared difference $\|Am - b\|^2$.

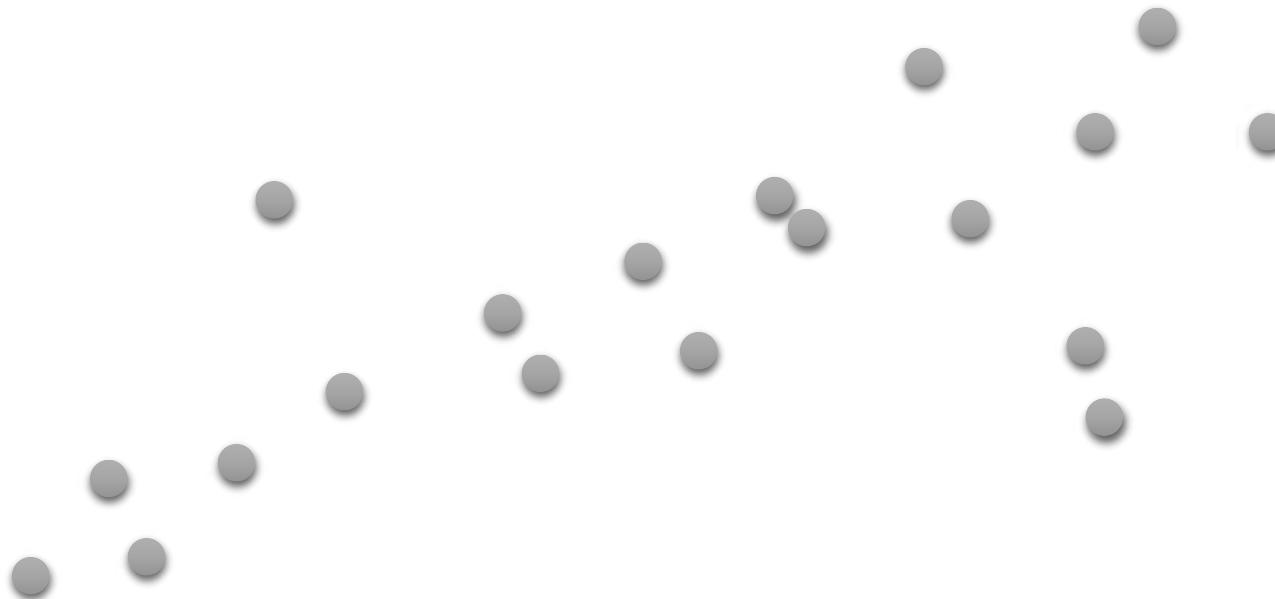
- Once the affine parameters are solved, we know the spatial transformation between the two images.
- Matching is done!

Outliers in matching

- Previously we assume that some pairs of keypoints are matched in two images. Based on this, we solve the linear system $Am = b$.
- What if some of these keypoints are mismatched, i.e. outliers?
- If we want to improve the robustness of matching, we can employ methods which consider outliers in model fitting, such as Random Sample Consensus (RANSAC).

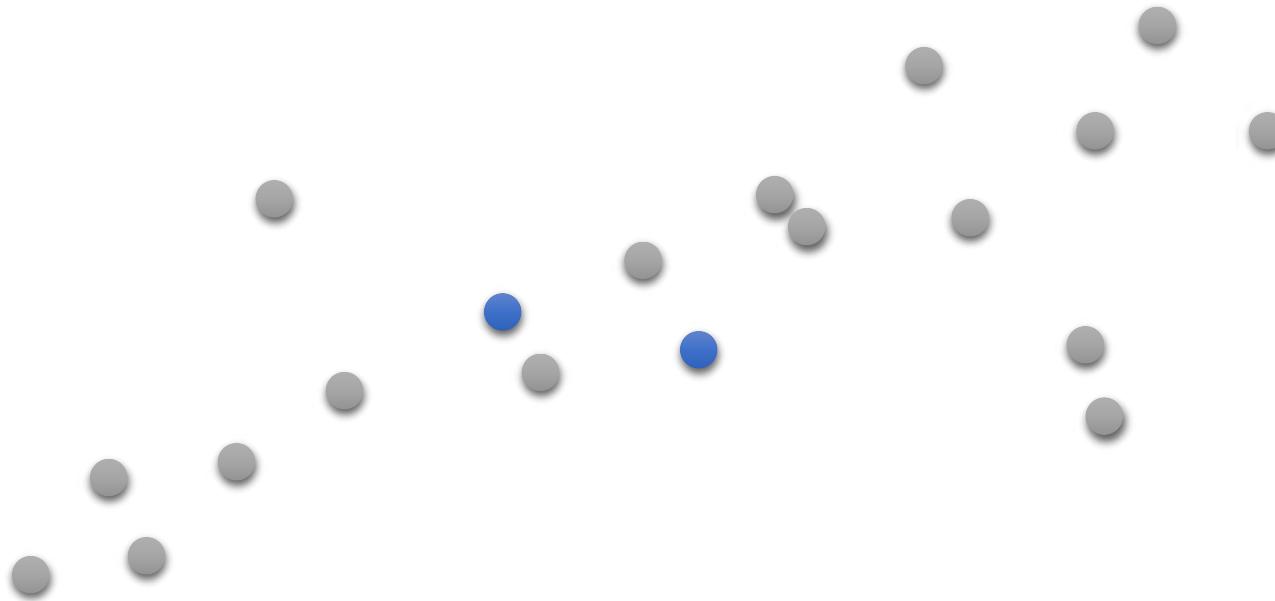


RANSAC



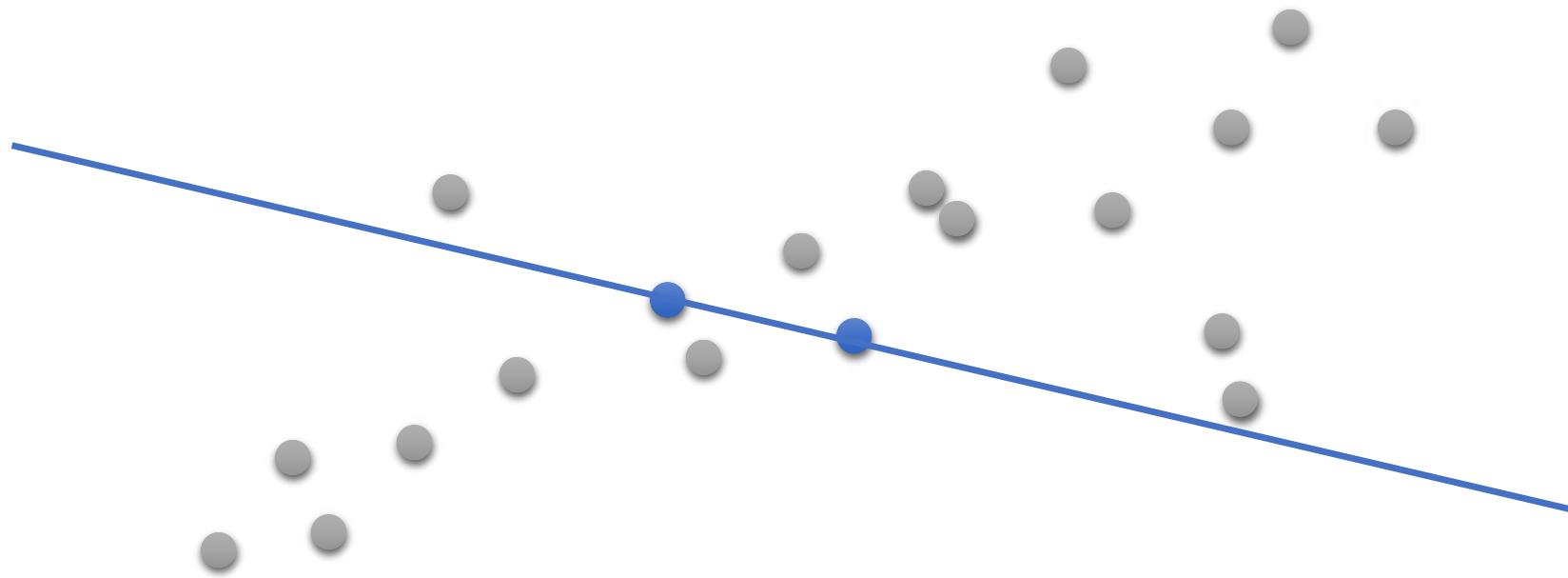
Task: find the best fitting line to the points.

RANSAC



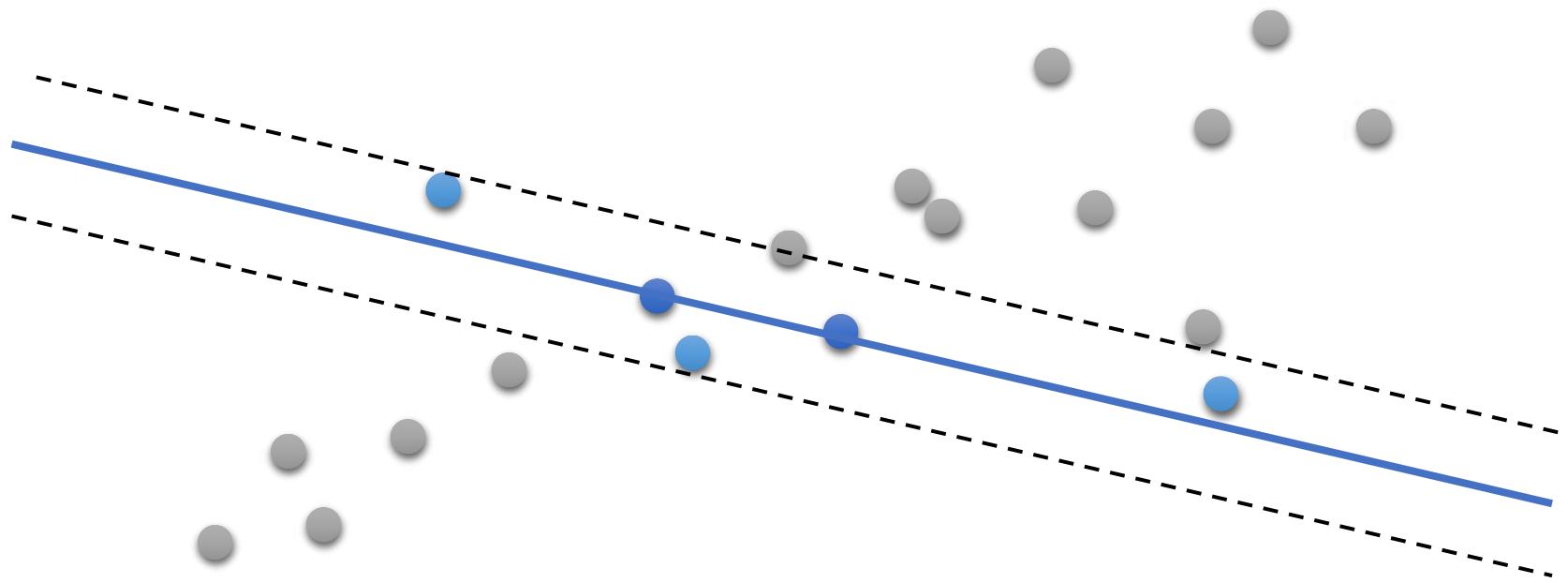
Randomly sample some points.

RANSAC



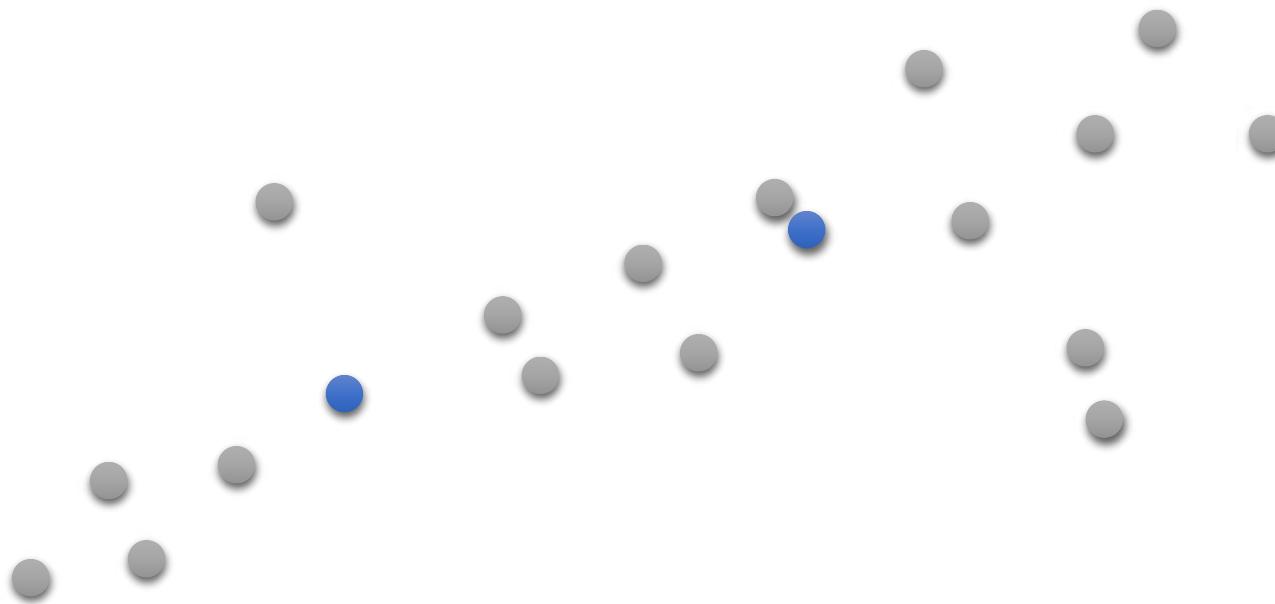
Fit a line.

RANSAC



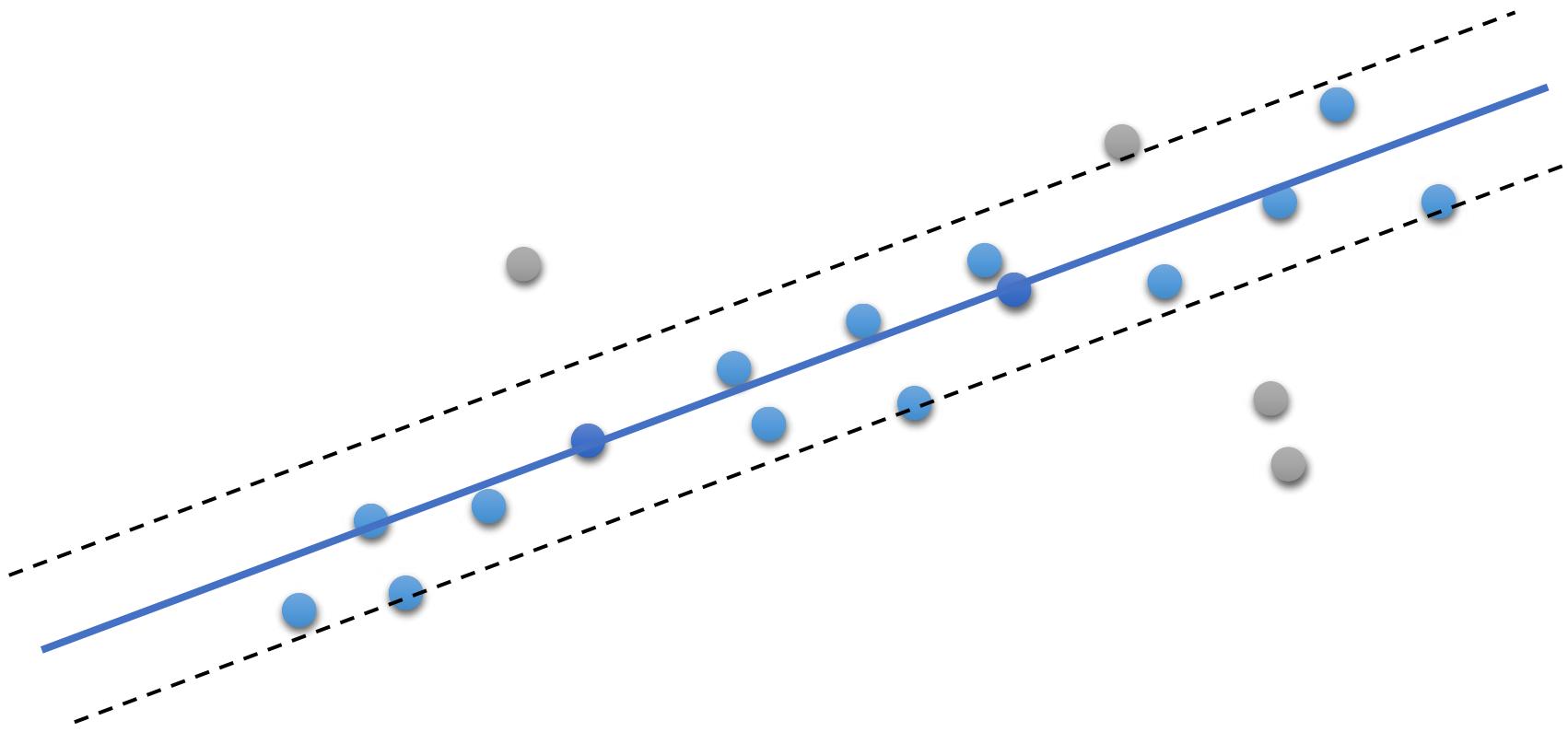
5 inliers found within a threshold to the line.

RANSAC



Repeat the random sampling.

RANSAC



15 inliers found.

Repeat until we get a good fitting line.

RANSAC

- For each iteration
 - Select random samples.
 - Fit a model.
 - Based on the model, check how many samples are inliers.
 - Terminate after a certain number of iterations or enough inliers have been found.

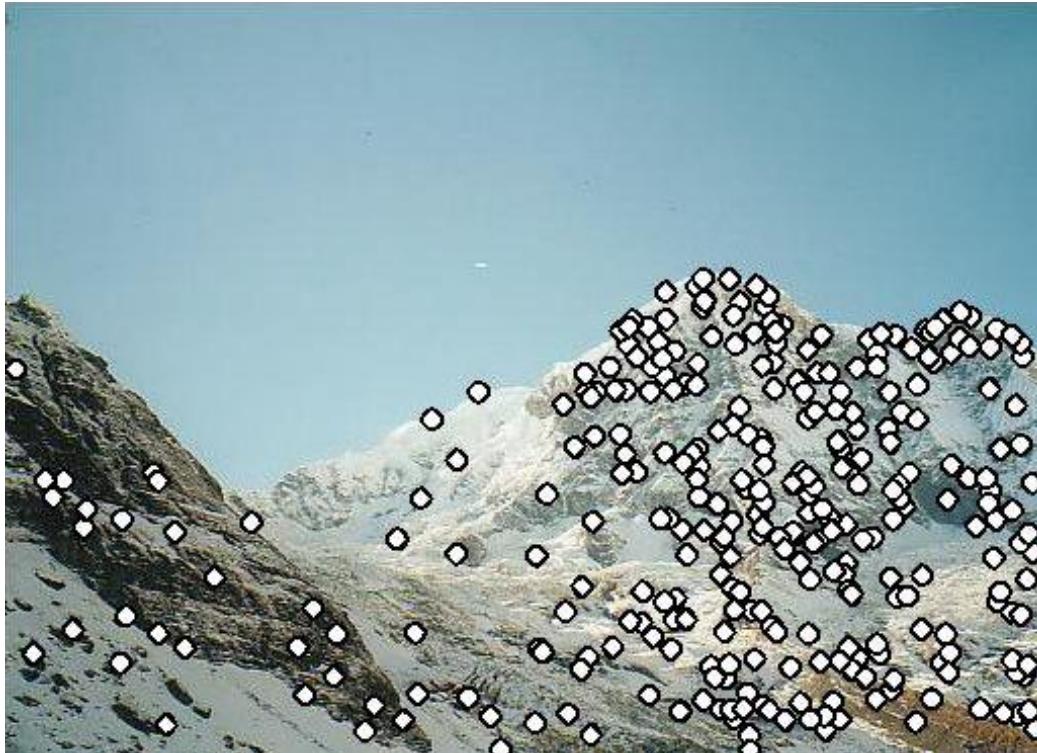
Keypoint matching

- Using RANSAC, we can find a good solution of this linear system, even when there are outliers in the input data: $\{(x_1, y_1), (x_2, y_2), \dots\}$, $\{(u_1, v_1), (u_2, v_2), \dots\}$.

$$Am = b$$

Applications of keypoint matching

- By matching keypoints and solving the linear system, we can estimate the transformation between two images and stitch them to create a panorama.



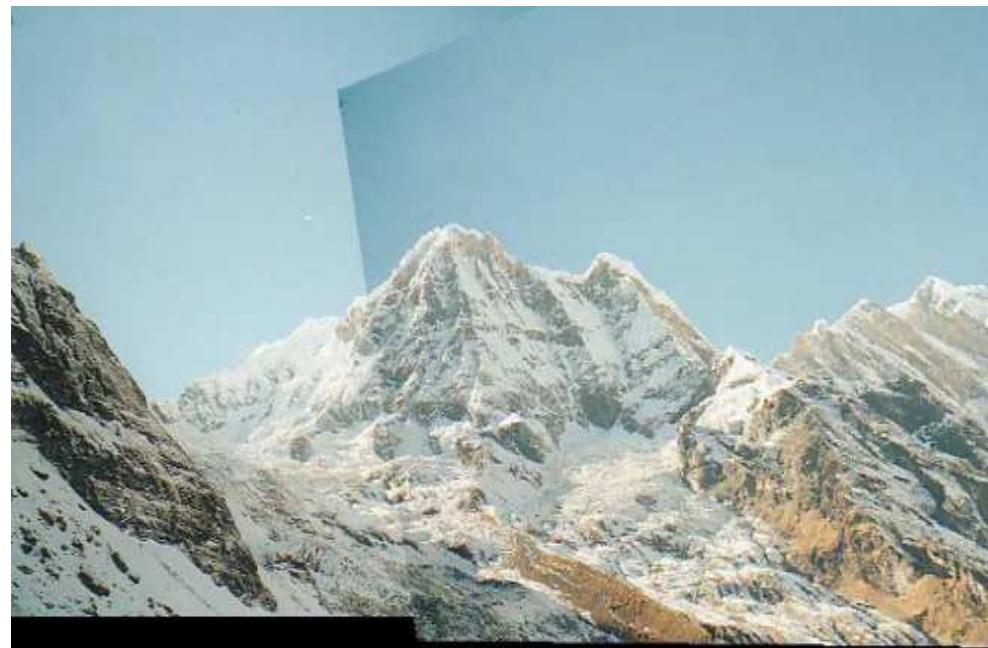


Image stitching, panorama on cameras



Multiple images



Image stitching



Image blending
(weighted average of
overlapping regions)

Applications of keypoint matching



template image



find the template
in a new image

Object recognition by image matching

Summary

- Feature descriptors extract local features near an interest point.
- Robustness to rotation, scaling and intensity changes are the three main aspects to consider in designing feature descriptors.
- We have introduced the SIFT descriptor.
- It can be used for matching keypoints between two images and estimating the spatial correspondence.

References

- Sec. 4.1.2 Feature descriptors, Sec 6.1 2D and 3D feature-based alignment. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Feature Description II

Dr Wenjia Bai

Department of Computing & Brain Sciences

Re-cap

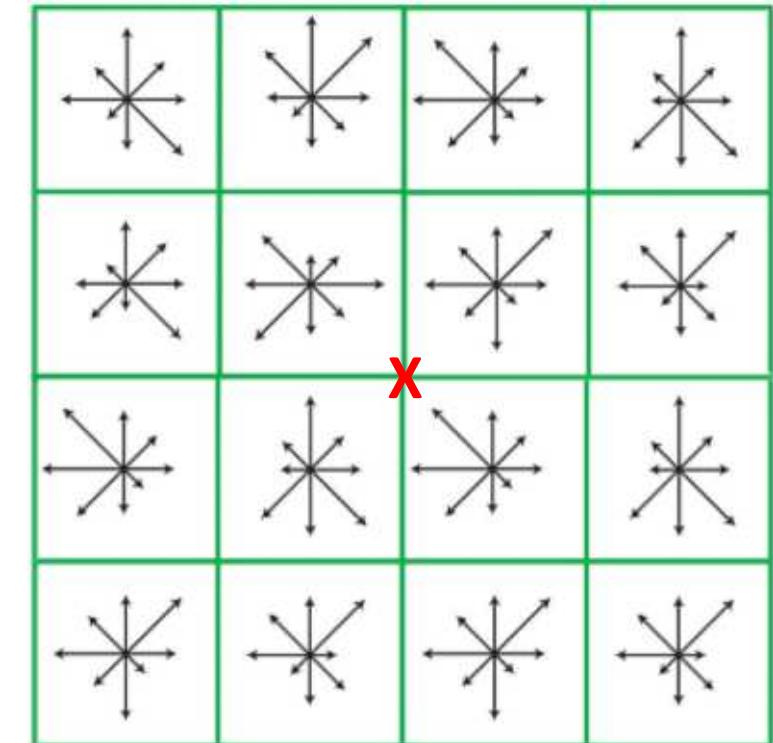
- SIFT is a classical algorithm detects and describes interest points in images.
- It is used for image matching and other applications.

In this lecture

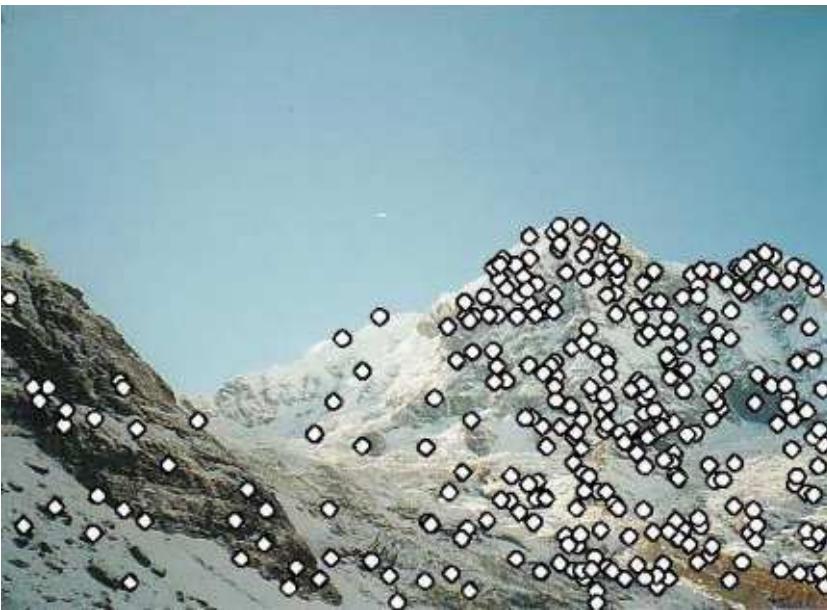
- We will describe several feature descriptors that are relevant to SIFT.
 - SURF
 - BRIEF
- We will then extend the idea of feature description for an interest point to feature description for an image.
 - HOG

SIFT

- SIFT descriptor at an interest point is formed by concatenating the histograms of gradient orientations for 16 subregions.
- However, calculating the gradient magnitudes and orientations can be slow, especially if you want to achieve real-time performance (e.g. image matching and stitching on camera) and you were still working in early 2000.



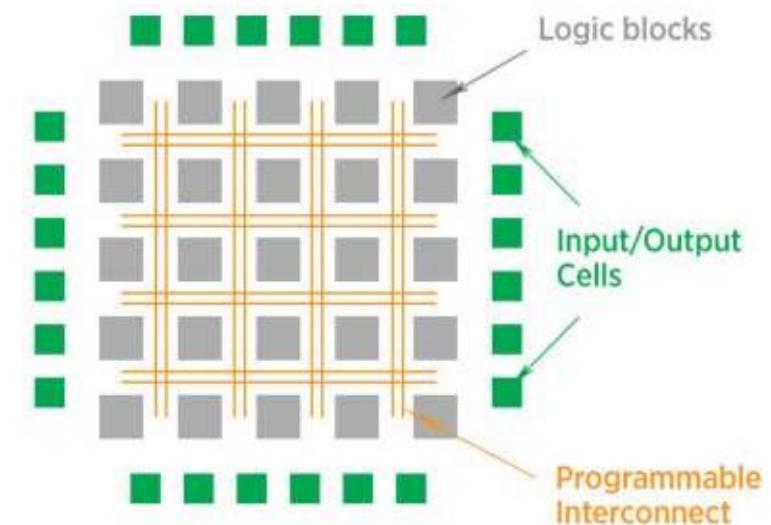
SIFT feature descriptor



How do we improve the speed for image matching?

Faster feature description

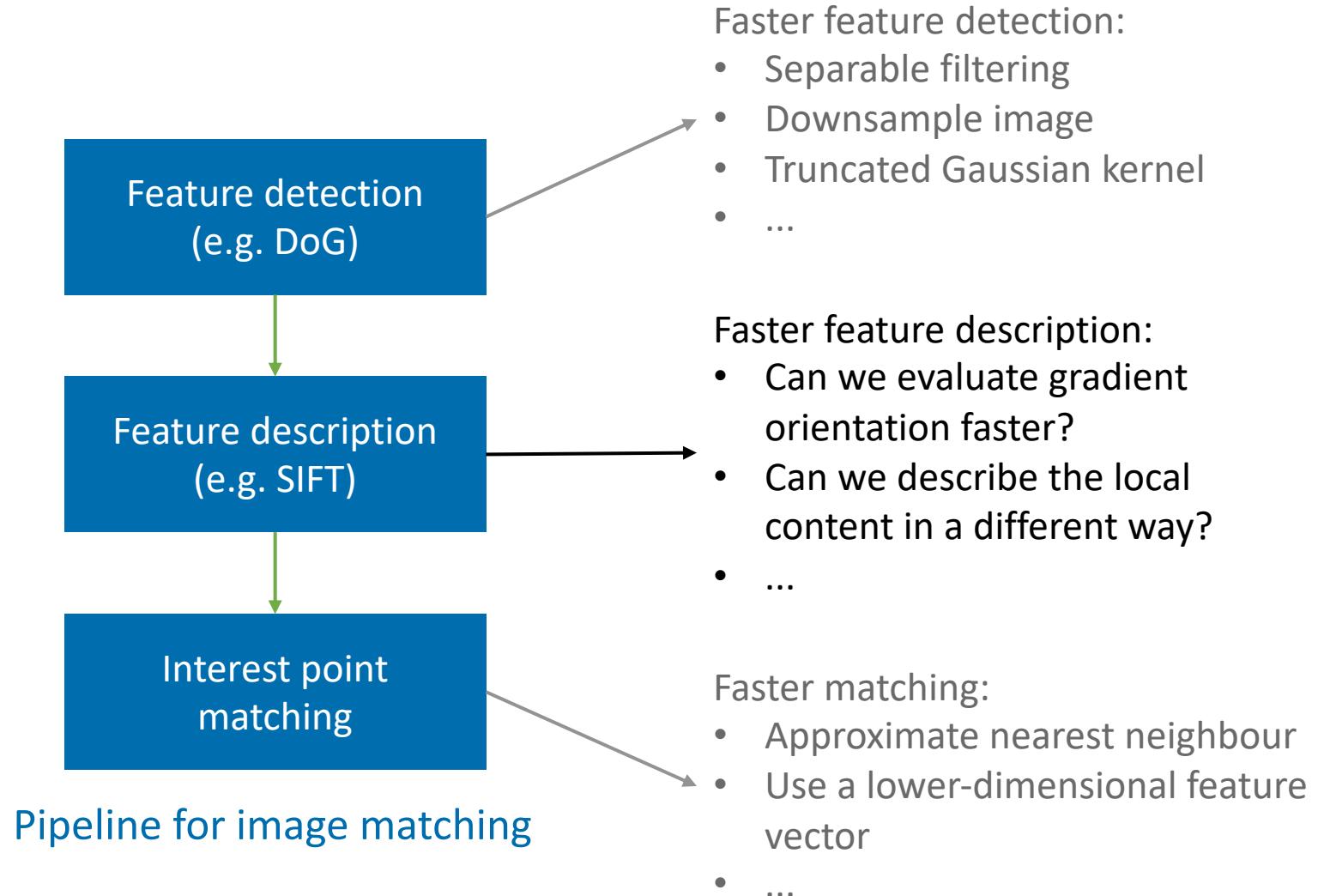
- You may have different ideas.
 - Use faster hardware
 - Optimise software implementation
 - Develop a new algorithm
- For example, one solution is to implement SIFT on field programmable gate array (FPGA), which is much faster than CPU.



FPGA is an integrated circuit that contains many logic blocks, configured to perform complex functions in a very fast way.

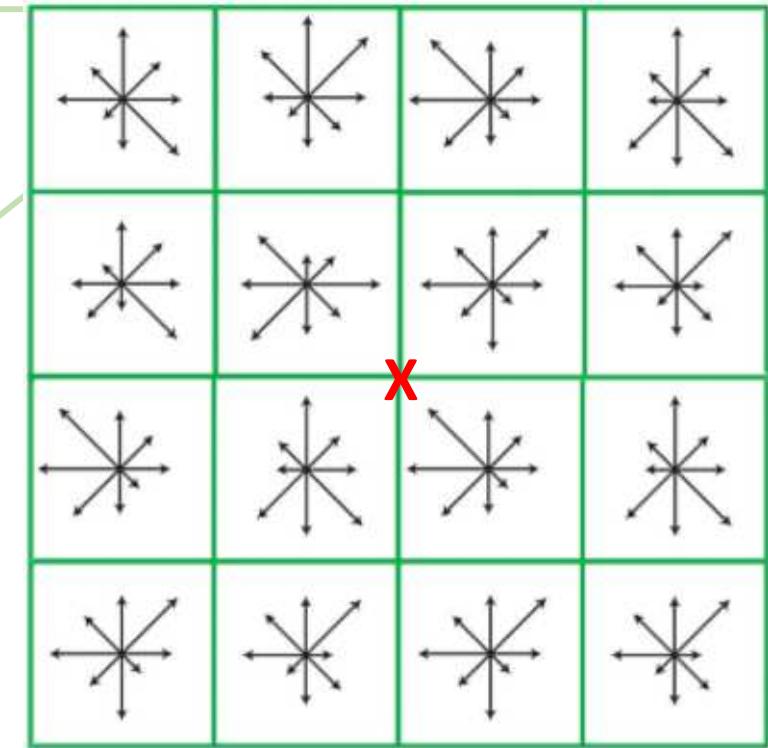
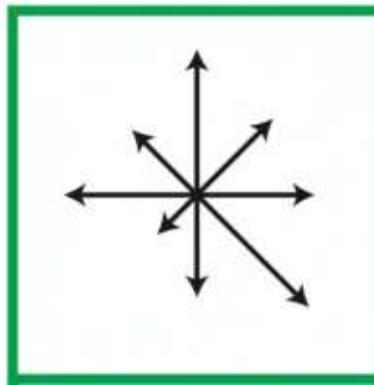
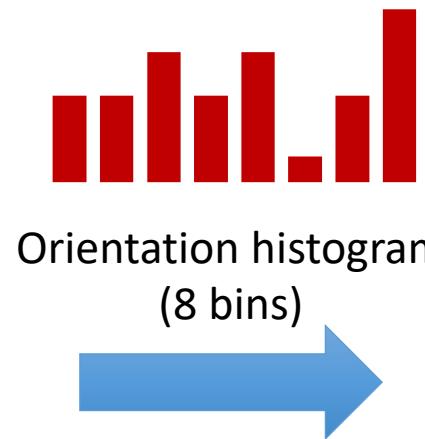
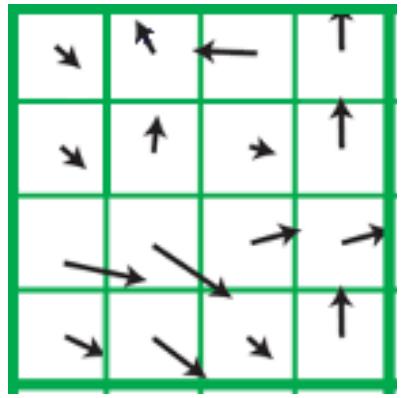
Faster feature description

- Another way is to improve the software or algorithm.
 - Decompose a pipeline into several steps
 - Reconsider and evaluate each step
 - How can we improve this step?

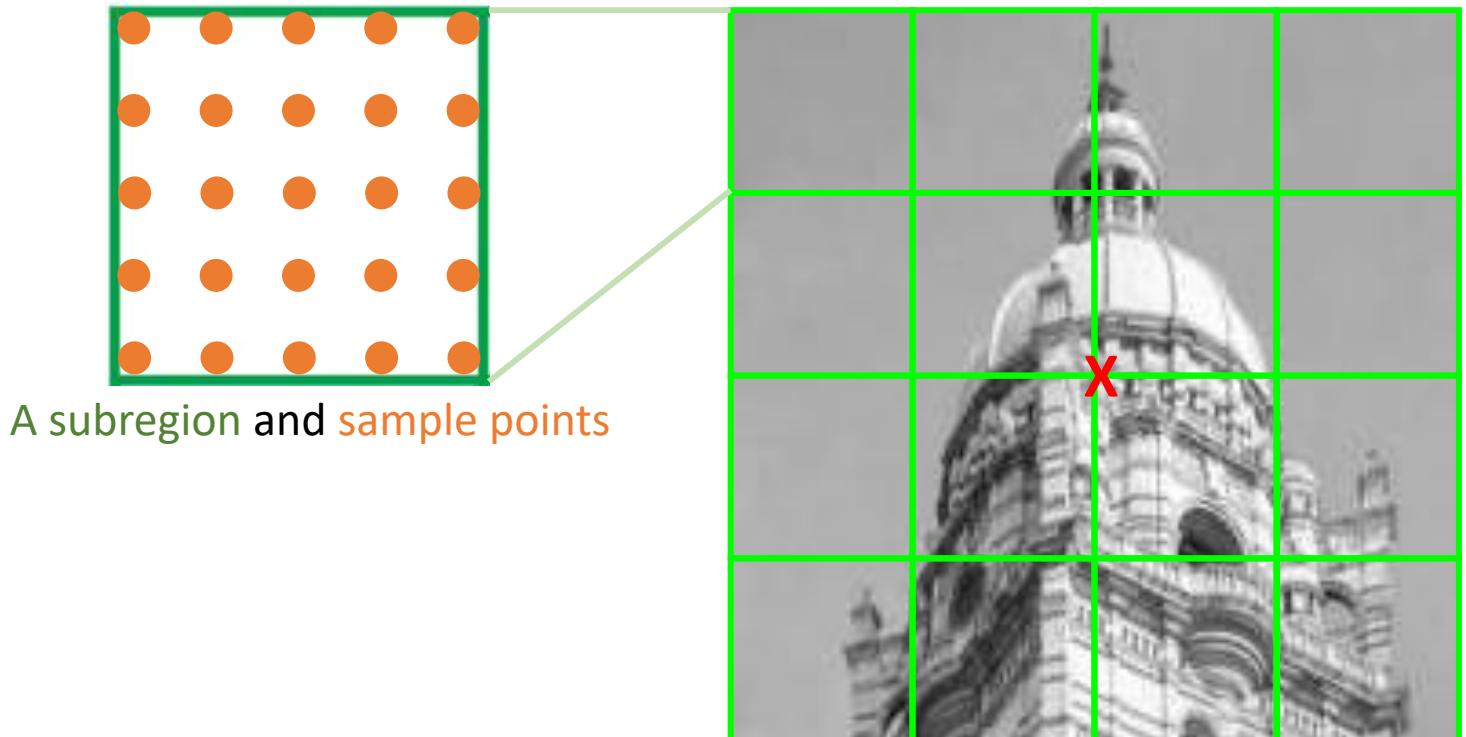


Speeded-Up Robust Features (SURF)

- SIFT uses histograms of gradient orientations for describing local features.
- To accelerate computation, SURF was proposed, which only calculates the gradients along horizontal and vertical directions by using the Haar wavelets.

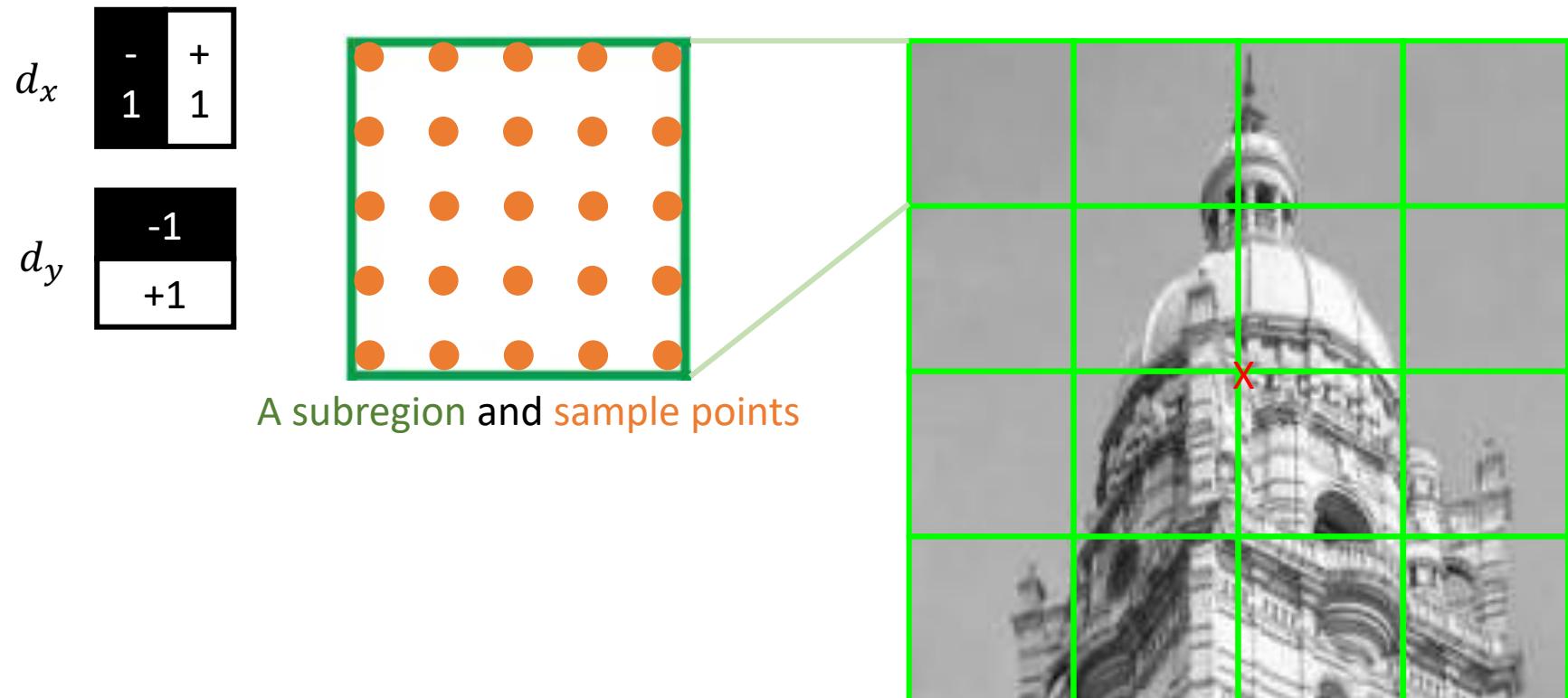


SIFT descriptor calculates a histogram of gradient orientations, for each of the 16 subregions centred at an interest point X.



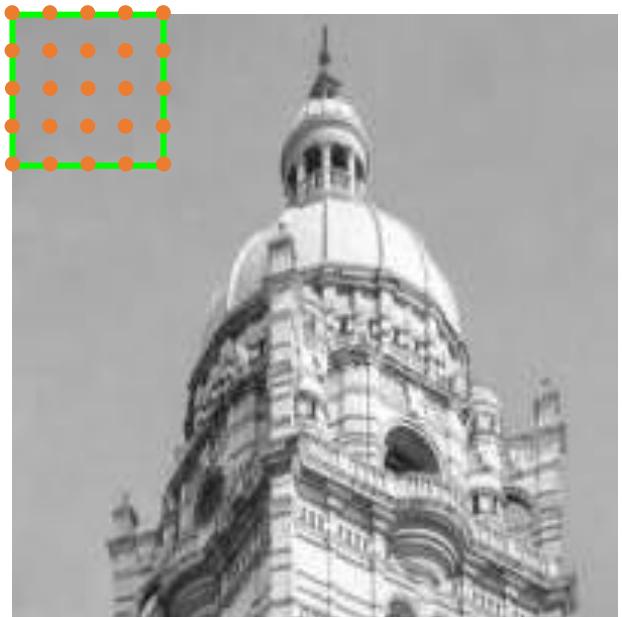
The idea of SURF is that we do not need to calculate gradients for arbitrary orientations. Let's only look at horizontal and vertical directions.

- SURF applies very simple filters d_x and d_y onto the sample points to calculate gradients along x and y.
- They are called Haar wavelets.



The idea of SURF is that we do not need to calculate gradients for arbitrary orientations. Let's only look at horizontal and vertical directions.

SURF



A subregion and sample points

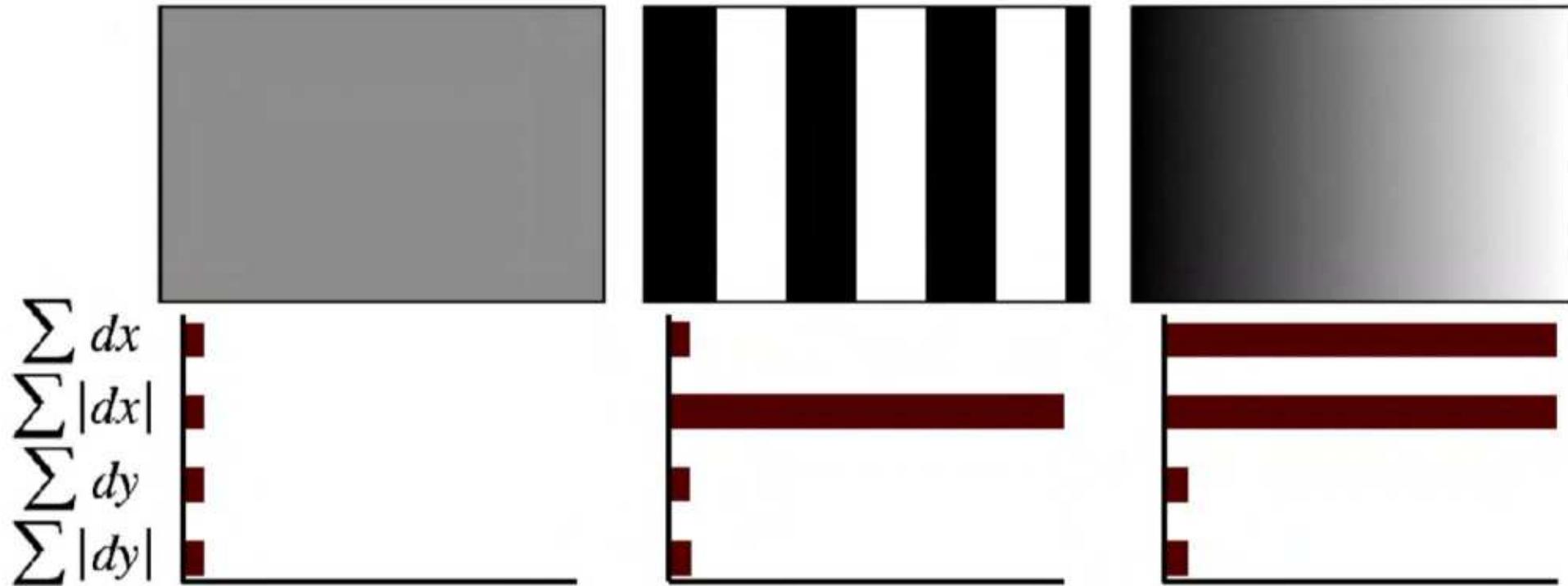
Haar wavelets,
summing up the pixel
intensities with weight +1
or -1, so very fast.

$$\begin{array}{c} -1 \\ 1 \end{array} \begin{array}{c} + \\ 1 \end{array} \begin{array}{c} d_x \\ d_y \end{array}$$

$(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$
summing up values summing up absolute values

For each subregion, sum up the Haar wavelet responses over the sample points. The descriptor for this subregion is defined by 4 elements.

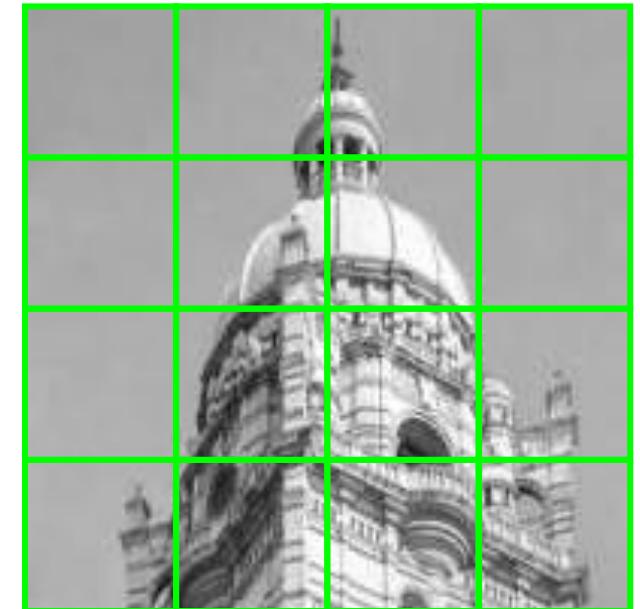
SURF



This seemingly simple descriptor ($\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|$) can represent the intensity patterns of a subregion. Left: a homogeneous region, all values are low; Middle: zebra pattern, $\sum |d_x|$ is high but other values are low; Right: gradually increasing intensities, $\sum d_x$ and $\sum |d_x|$ are high.

SURF

- Dimension of SURF descriptor
 - 4 x 4 subregions
 - 4 elements per subregion
 - 16 subregions x 4 elements = 64
- SURF is ~5 times faster than SIFT, due to the use of simple Haar wavelets for feature computation.



Even faster?

- Feature descriptors
 - SIFT: $16 \times 8 = 128$ -D vector
 - SURF: $16 \times 4 = 64$ -D vector
 - Each dimension is a floating-point number or 4 bytes.
 - To compare two feature vectors, we calculate their Euclidean distance.
- Can we make feature description and matching even faster?
- Similar motivation as we develop SURF
 - We can deploy feature description and image matching algorithms faster on devices, such as camera, mobile phones, robots etc.

Even faster?

- How to further shorten the descriptor?
 - What about quantize or even binarize the floating-point number (4 bytes)?
 - Quantization means converting a continuous number into a discrete number (e.g. 0 to 255, which means 8 bits).
 - Binarization means converting into a binary number (0 or 1, which means 2 bits).
- How to compare two feature vectors faster?
 - Distances other than Euclidean?
- ...

Binary Robust Independent Elementary Features (BRIEF)

- When we use Haar wavelets in SURF, we compare a local region to another and calculate the difference, which is a floating point number.

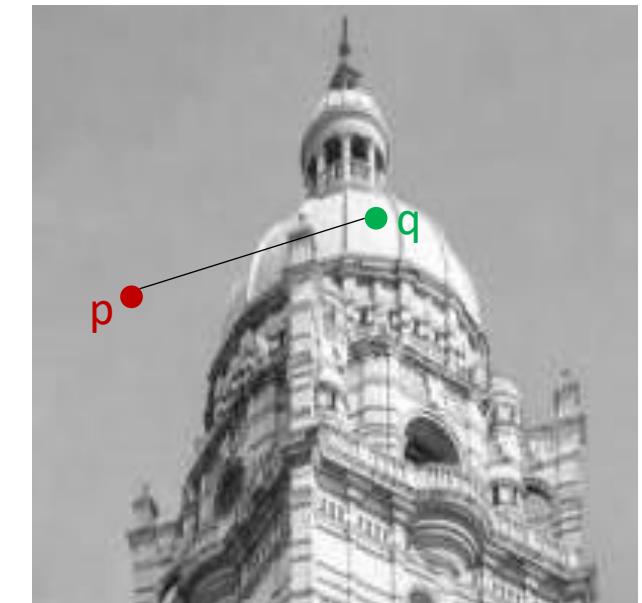
$$\begin{array}{c} -1 \quad +1 \\ d_x \end{array} \quad \begin{array}{c} -1 \\ +1 \\ d_y \end{array}$$

Haar wavelets used in SURF

- In BRIEF, we compare a point p to another point q and get a binary value as output.

$$\tau(p, q) = \begin{cases} 1, & \text{if } I(p) < I(q) \\ 0, & \text{otherwise} \end{cases}$$

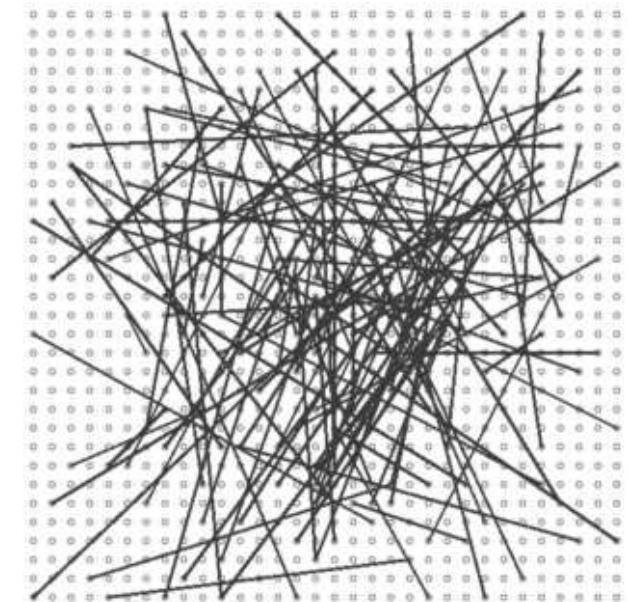
- How do we sample these point pairs (p, q) ?



Binary test in BRIEF

BRIEF

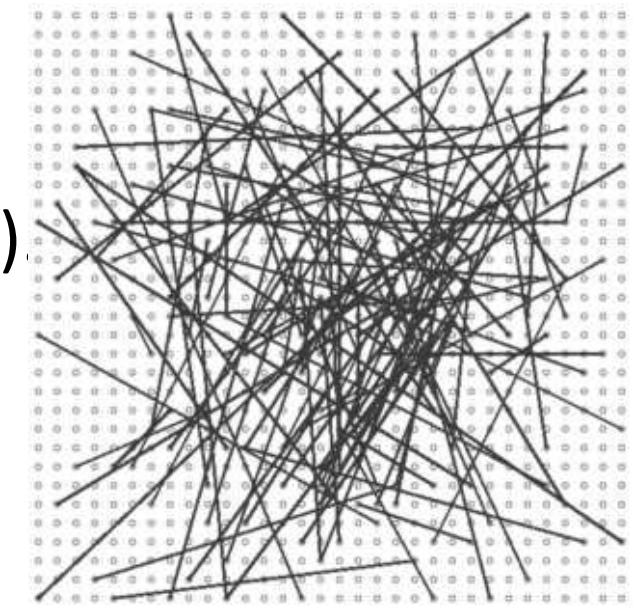
- We randomly sample n_d pairs of points for binary tests.
 - The random pattern is determined only once. Then we always apply the same pattern to all interest points.
- If $n_d = 256$, we perform 256 tests and each test gives us 1 bit (0 or 1).
- The BRIEF descriptor is described as a n_d -dimensional bitstring.
 - For example, [1, 0, 0, 0, 1, ..., 0, 0, 1].



Random sampling pattern
for binary tests

BRIEF

- If $n_d = 256$, the BRIEF descriptor is 256-bit long or 32 bytes.
- This is shorter than SIFT (512 bytes) or SURF (256 bytes).
- The computation is much faster.
 - We only compare two numbers, without calculating the gradient orientation (SIFT) or intensity difference (SURF).
 - When we compare two BRIEF descriptors, it is also faster as we do not need to calculate the Euclidean distance.



Random sampling pattern
for binary tests

Fast to compute the BRIEF descriptor

For example, to perform 8 binary tests and get one byte of descriptor:

```
descriptor = ((l(p1) < l(q1)) << 7) + ((l(p2) < l(q2)) << 6) + ((l(p3) < l(q3)) << 5)  
+ ((l(p4) < l(q4)) << 4) + ((l(p5) < l(q5)) << 3) + ((l(p6) < l(q6)) << 2)  
+ ((l(p7) < l(q7)) << 1) + ((l(p8) < l(q8)) << 0)
```

p and q are pre-defined sampling points.

“<< n” means shifting the bit by n places.

Fast to compare two BRIEF descriptors

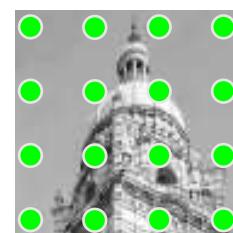
- When we perform image matching and comparing two BRIEF descriptors, we can measure their difference using the Hamming distance.
 - Hamming distance can be computed very efficiently with a bitwise XOR operation followed by a bit count.

descriptor a	1 0 0 0 1 0 0 1
XOR	
descriptor b	1 1 0 0 0 0 1 1
	0 1 0 0 1 0 1 0

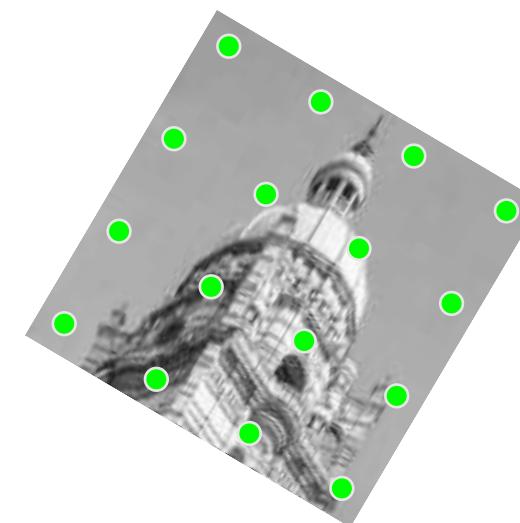
By counting the bit of **1**, we get the Hamming distance = 3.

BRIEF

- BRIEF ignores rotation- and scale-invariance.
- It assumes that the images are taken from a moving camera that only involves translation, and it does not account for rotation or scaling.



BRIEF only accounts
for translation



It does not account for
rotation or scaling.

BRIEF

- By combining all these components, BRIEF achieves ~40-fold speed-up over SURF [1].
 - Binary tests
 - Hamming distance
 - Ignore orientations and scales
- Recall that SURF is already ~5 times faster than SIFT [2].
- So BRIEF is ~200 times faster than SIFT.
- If there is no rotation or scaling involved, BRIEF achieves a performance comparable to SURF in terms of interest point matching accuracy [1].

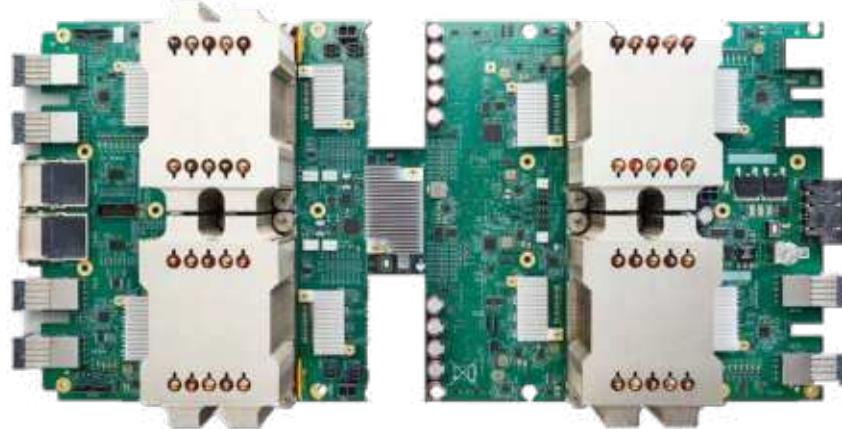
[1] M. Calonder. BRIEF: Binary Robust Independent Elementary Features. ECCV 2010.

[2] H. Bay et al. SURF: Speeded Up Robust Features. ECCV, 2006.

Acceleration

- SURF and BRIEF were proposed to accelerate feature description for interest points, built upon the SIFT algorithm.
- Similar ideas were explored for other applications.

Hardware



Google TPU for machine learning



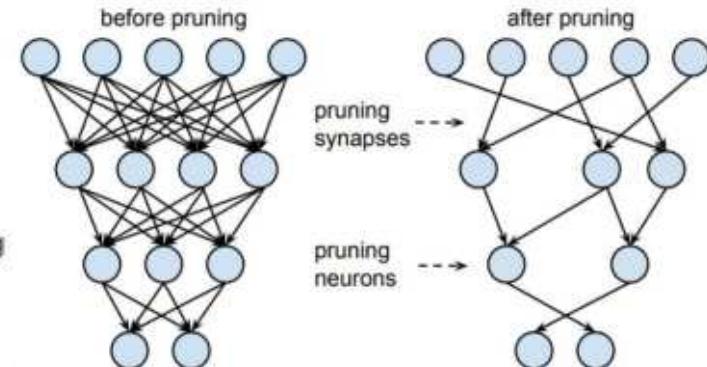
FPGA for high frequency trading

Software

CUDA Platform Mixed Precision Support

Feature	FP16x2 (HFMA2)	INT8/16 DP4A/DP2A
PTX instructions	CUDA 7.5	CUDA 8
CUDA C/C++ intrinsics	CUDA 7.5	CUDA 8
cuBLAS GEMM	CUDA 7.5	CUDA 8
cuFFT	CUDA 8	I/O via cuFFT callbacks
cuDNN	5.1	6
TensorRT	v1	v2 Tech Preview

CUDA supports 16-bit floating point format.



Neural network pruning

weights (32 bit float)	cluster index (2 bit uint)
2.09	0
-0.98	2
1.48	1
0.09	1
0.05	3
-0.14	0
-1.08	0
2.12	3
-0.91	1
1.92	0
0	3
-1.03	1
1.87	0
0	2
1.53	1
1.49	2

Weight quantisation

Feature descriptor

- These feature descriptors are defined for an interest point for describing local content, centred at this point.
- Can we extend the idea to describe the feature of a whole image or a image region?
 - Yes.





Can we describe the feature of human for an image or a region?

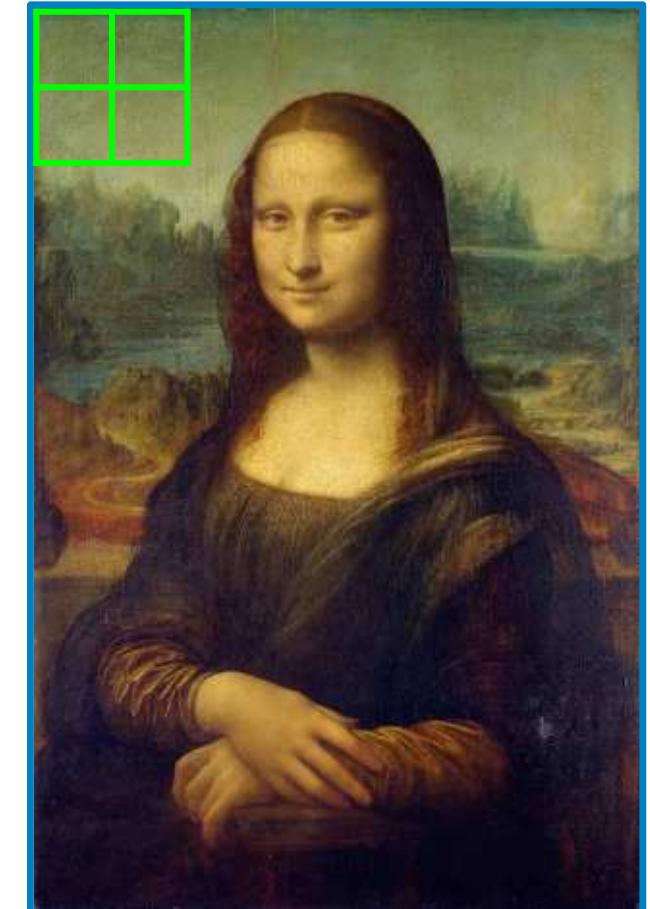
Histograms of Oriented Gradient (HOG)

- HOG is similar to SIFT. They both use gradient orientation histograms for feature description.
- The difference is that HOG describes features for a large image region, instead of just around a point.
- The idea of HOG is that it divides a large region into a dense grid of cells, describes each cell, then concatenate these local descriptions to form a global description.

HOG

- Suppose we want to describe the feature for Mona-Lisa.
- We can divide the image into equally spaced cells.
 - Each cell contains 8×8 pixels.
 - 4 cells form a block.
- The orientation histograms of this block describes the top-left corner of this image.

Calculate gradient orientation histograms for this block.



128x64 pixel image.

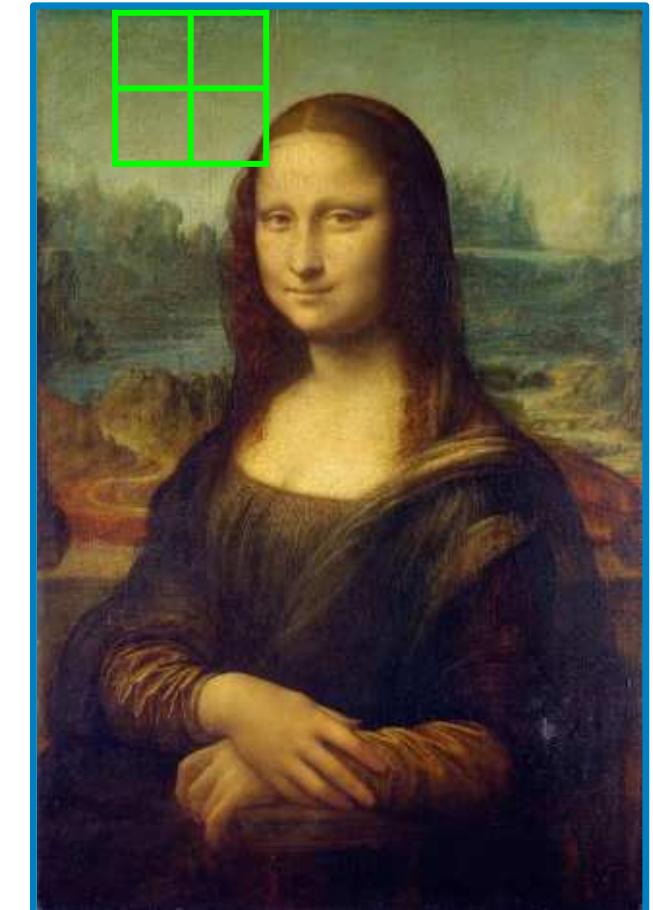
HOG

- We move to the next block and similarly describe the content there using orientation histograms.
 - There is an overlap between blocks.
 - For each block, the descriptor vector v (concatenation of 4 histograms) is normalised,
 v

$$v_{norm} = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}$$

locally normalised descriptor
a small value

Calculate the histograms at the next block.

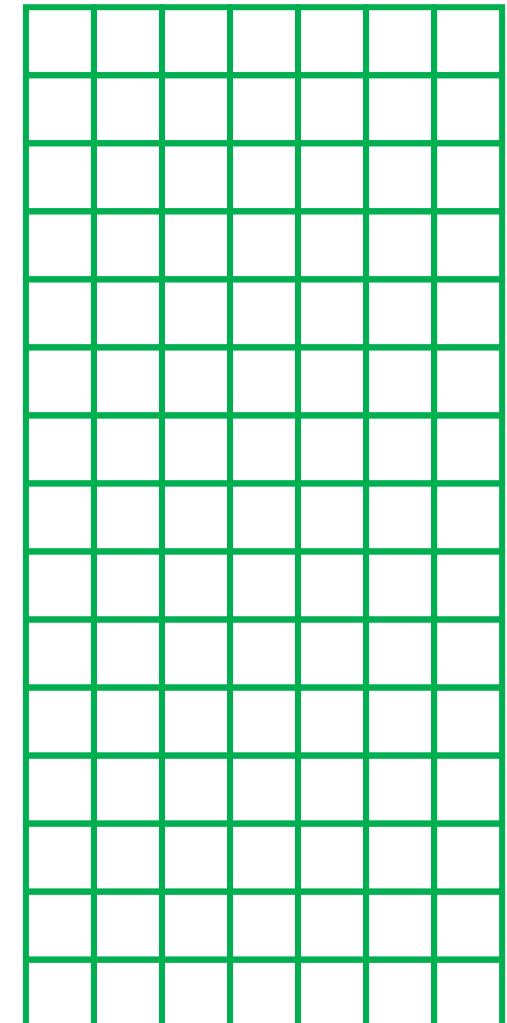


128x64 pixel image.

HOG

- The HOG descriptor is formed by concatenating the normalised local descriptors for all blocks.
- In this way, we can describe a full image or a large image region.
- We use a dense grid to cover and describe the image. In contrary, SIFT only looks at a single point.

The descriptors for all blocks in this image.



Feature descriptor

- What can we do if we can describe a full image or a region?
 - We can perform image classification based on image features.
 - We can detect whether a region contains human or not.
 - We can retrieve similar images by their features.
 - ...

Image classification

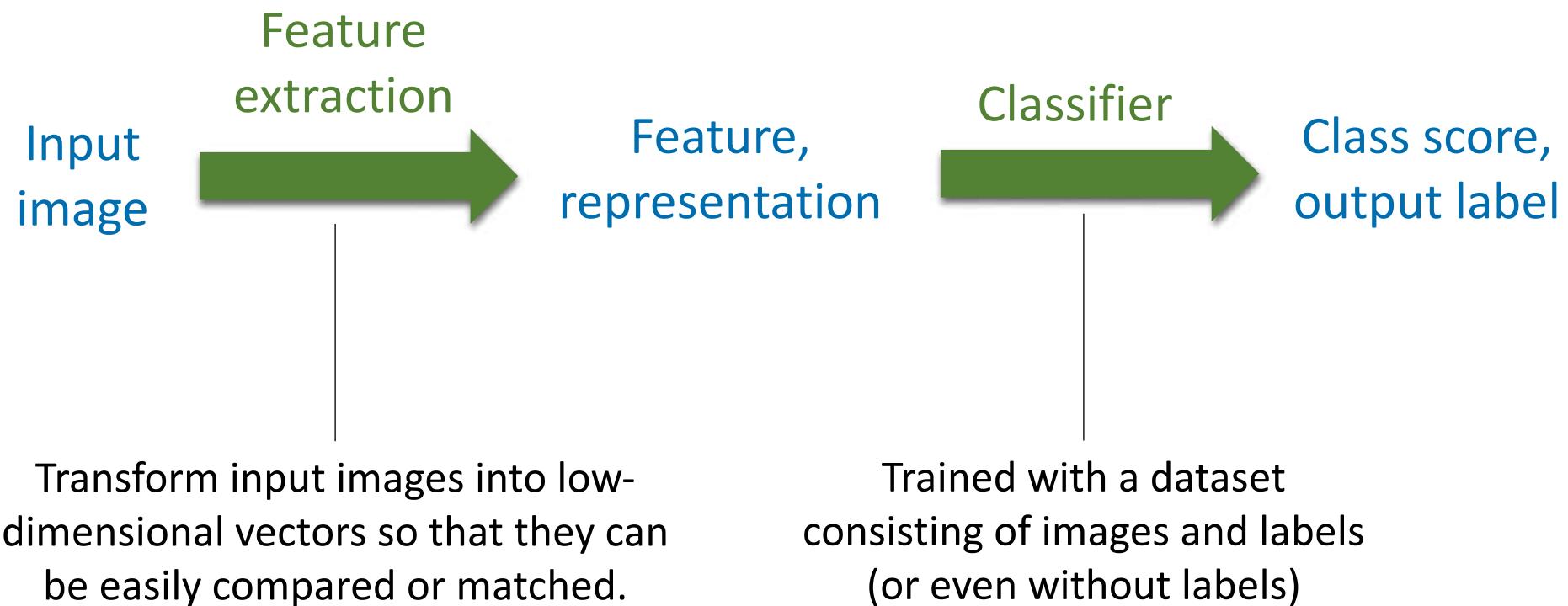


Image classification

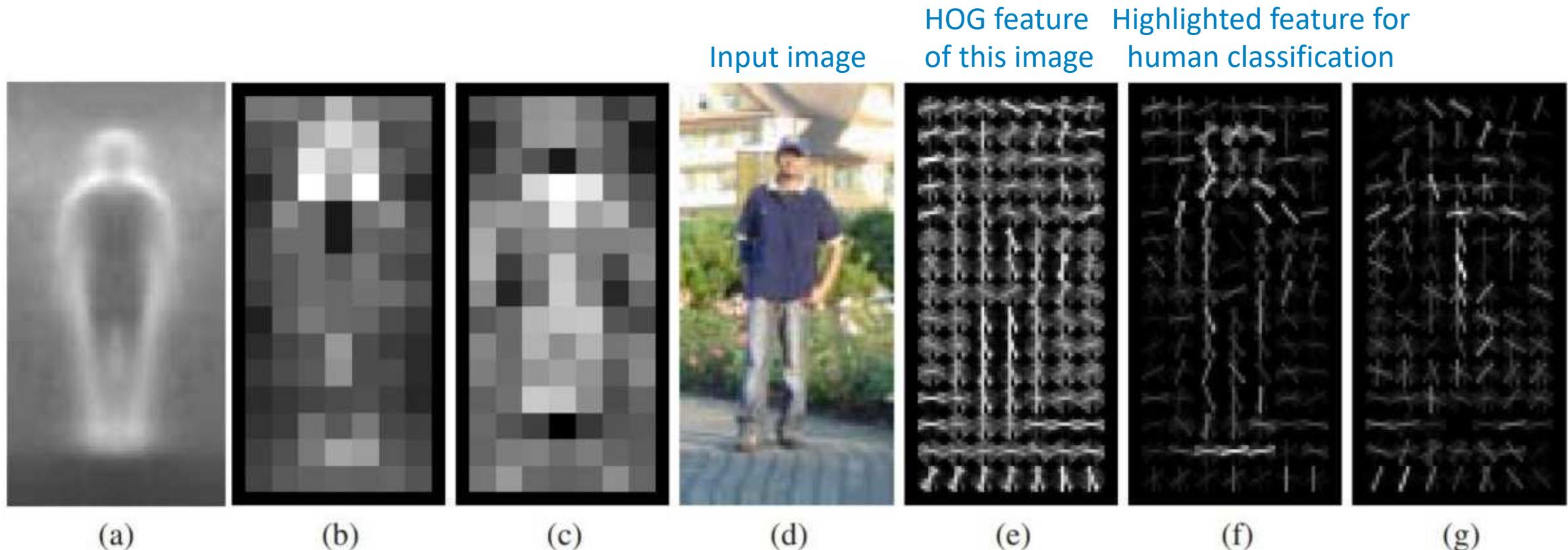


Figure 6. Our HOG detectors cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centred on the image background just *outside* the contour. (a) The average gradient image over the training examples. (b) Each “pixel” shows the maximum positive SVM weight in the block centred on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) It’s computed R-HOG descriptor. (f,g) The R-HOG descriptor weighted by respectively the positive and the negative SVM weights.



HOG feature for human detection

Credit: <https://medium.com/@madhawavidanapathirana>

Summary

- We start from feature descriptors that extract local features near an interest point,
 - SIFT
 - SURF
 - BRIEF
- Then move onto a feature descriptor that describes an image,
 - HOG
- If we can describe an image, we can perform image classification, which will start out next lecture.

References

- Sec. 4.1.2 Feature descriptors. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Image Classification I

Dr Wenjia Bai

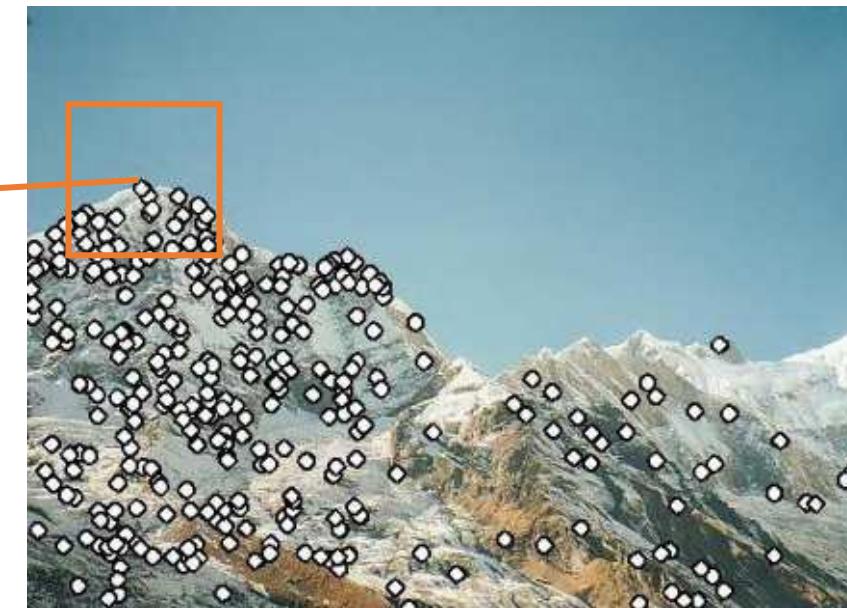
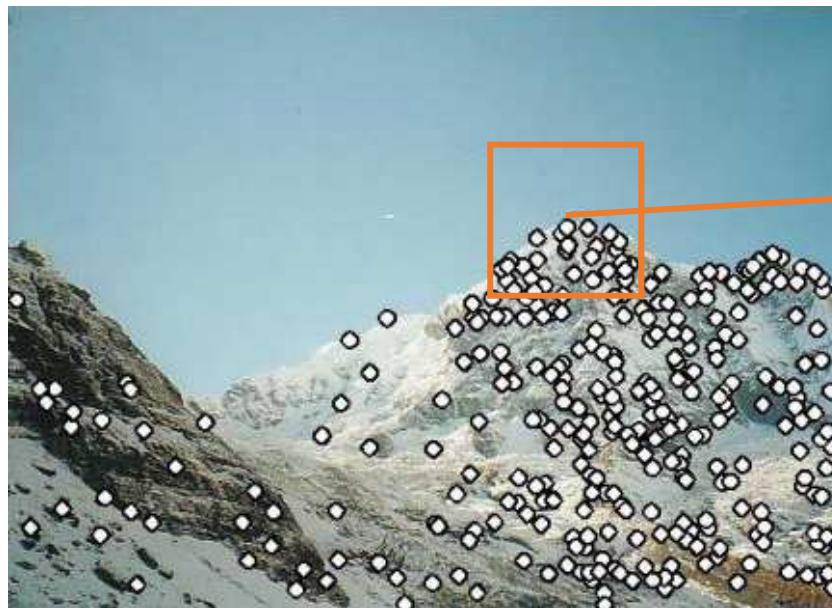
Department of Computing & Brain Sciences

Announcement

- Coursework 2 will be released next Thursday, after we have introduced image classification and convolutional neural networks.

Feature description

- We can describe the local feature for a patch centred at an interest point so that the points can be matched to each other.



Feature description

- We can also describe the feature of an image so we can learn what human or other objects looks like.



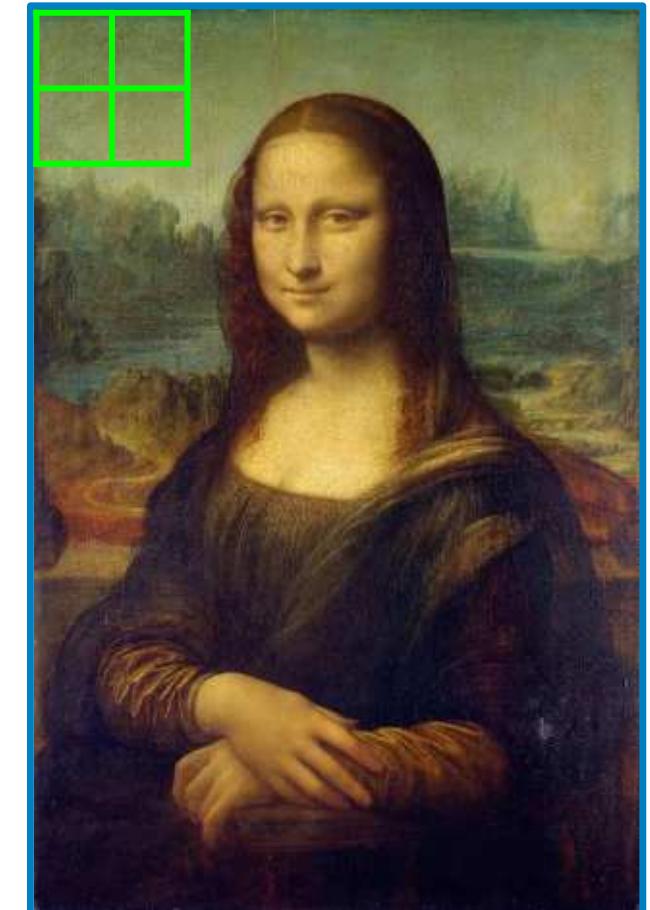
Histograms of Oriented Gradient (HOG)

- HOG is similar to SIFT. They both use gradient orientation histograms for feature description.
- The difference is that HOG describes features for a large image region, instead of just around a point.
- The idea of HOG is that it divides a large region into a dense grid of cells, describes each cell, then concatenate these local descriptions to form a global description.

HOG

- Suppose we want to describe the feature for Mona-Lisa.
- We can divide the image into equally spaced cells.
 - Each cell contains 8×8 pixels.
 - 4 cells form a block.
- The orientation histograms of this block describes the top-left corner of this image.

Calculate gradient orientation histograms for this block.



128x64 pixel image.

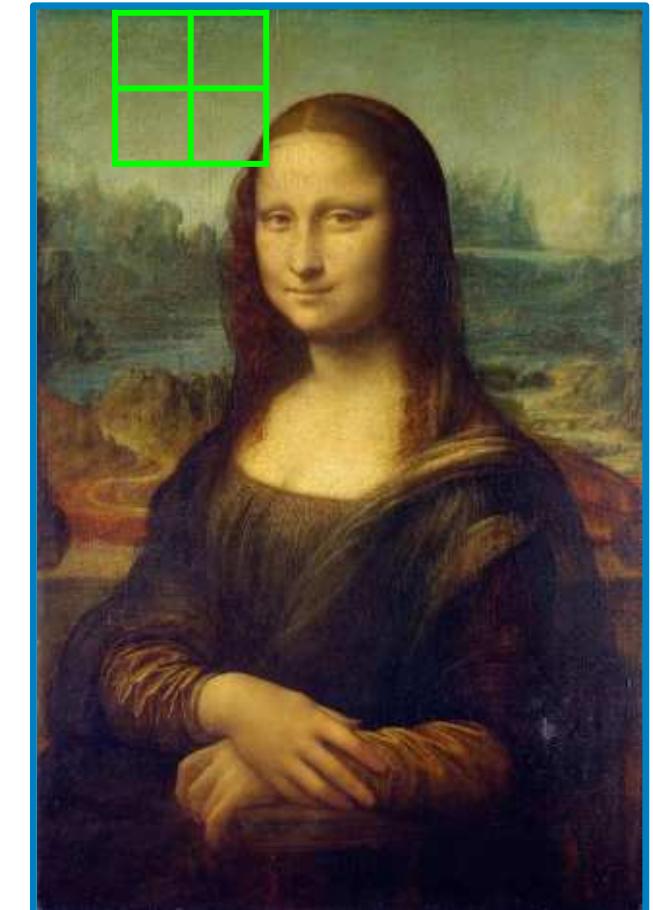
HOG

- We move to the next block and similarly describe the content there using orientation histograms.
 - There is an overlap between blocks.
 - For each block, the descriptor vector v (concatenation of 4 histograms) is normalised,
 v

$$v_{norm} = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}}$$

| locally normalised descriptor a small value

Calculate the histograms at the next block.

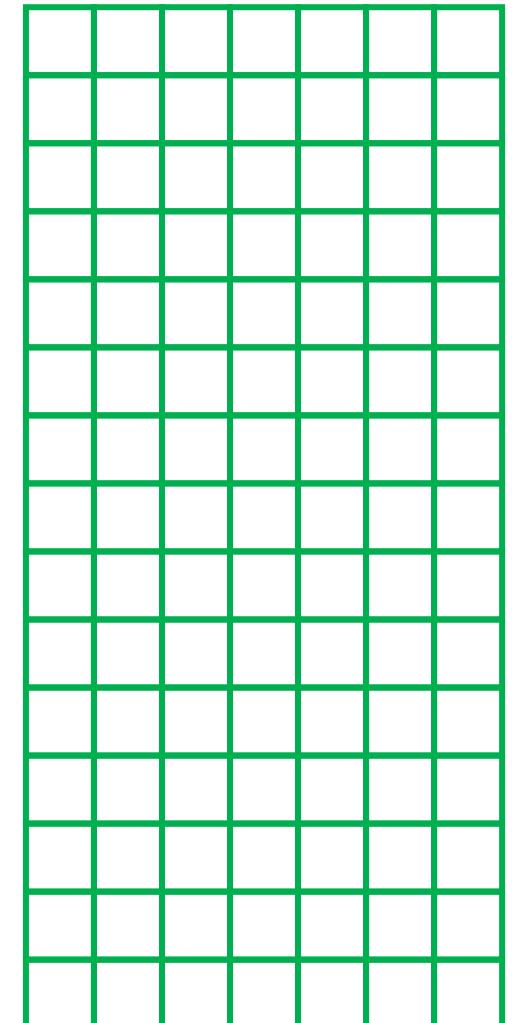


128x64 pixel image.

HOG

- The HOG descriptor is formed by concatenating the normalised local descriptors for all blocks.
- In this way, we can describe a full image.
- We use a dense grid to cover and describe the image. In contrary, SIFT only looks at a single point.

The descriptors for all blocks in this image.



Feature descriptor

- What can we do if we can describe a full image or a region?
 - We can perform image classification based on image features.
 - We can detect whether a region contains human or not.
 - We can retrieve similar images by their features.
 - ...

Image classification

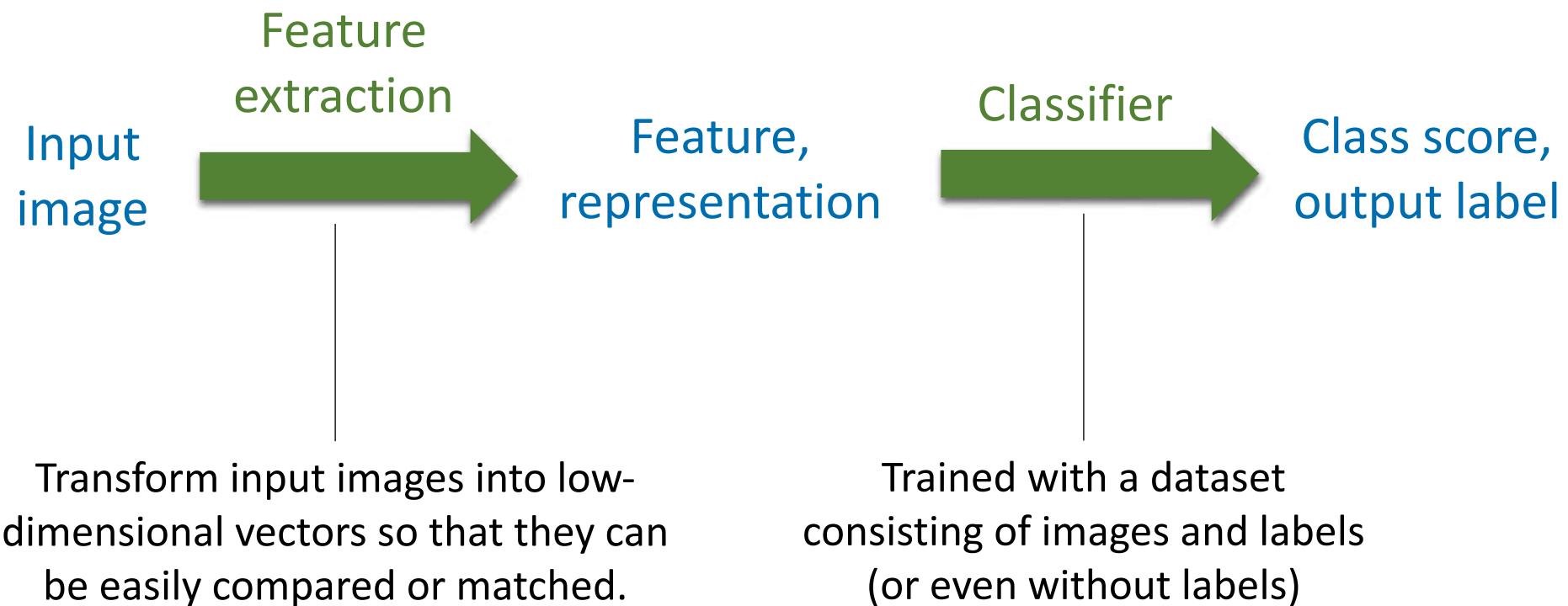


Image classification

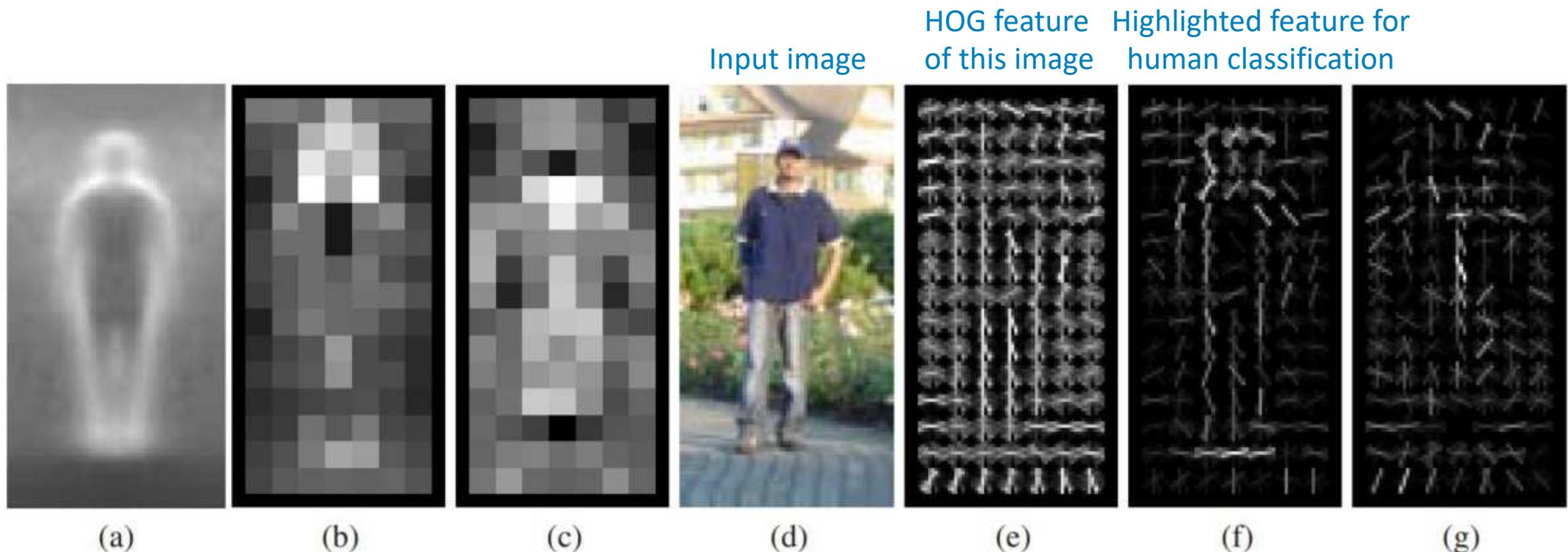


Figure 6. Our HOG detectors cue mainly on silhouette contours (especially the head, shoulders and feet). The most active blocks are centred on the image background just *outside* the contour. (a) The average gradient image over the training examples. (b) Each “pixel” shows the maximum positive SVM weight in the block centred on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) It’s computed R-HOG descriptor. (f,g) The R-HOG descriptor weighted by respectively the positive and the negative SVM weights.



HOG feature for human detection

Credit: <https://medium.com/@madhawavidanapathirana>

Image classification

- Classify an image into 10 categories of digits (0 to 9).
 - Hand written digit classification
 - Street number recognition



Street number recognition

40281508803277264755529284686500876/7112740077638420140578214711366
5071116767966414311224108763400633017113109975414895351982739901029
8468482467933943144705960444612336459685658641865284554770782237018
76953465018828357808571101378507110114527623028596972136418240510226
93477149069842728100783331376131645747595849916501320348220251514889
820499623356480928364572949128600709116759914592504108908989425198980
3551721691995516228671460403322368985385452056328399579467131366090/
94368160413174951001162198403649071654525185470670258104571851900607
8857389886823975629288168879180172075190209862393802111142972512199
14853434775074881539597690363982212868553949251514414435912233029009
93190975492010514933615252209266012030255795508950325908846884546549
6928545799218340783934056219260061287982047750564674307507420899404
1284527811303570319363177308482652973909964297211674759821445161325
90664367728608302983253980019513960141712379749939282718091017796999
21010452828351781129784050788447858498138031795516574935471208160734
28308784084458566309376893495891288681379011470817457121130621280766
41992780136134111560707232522949812161274000822922799275134941856283

Hand written digit classification

Image classification

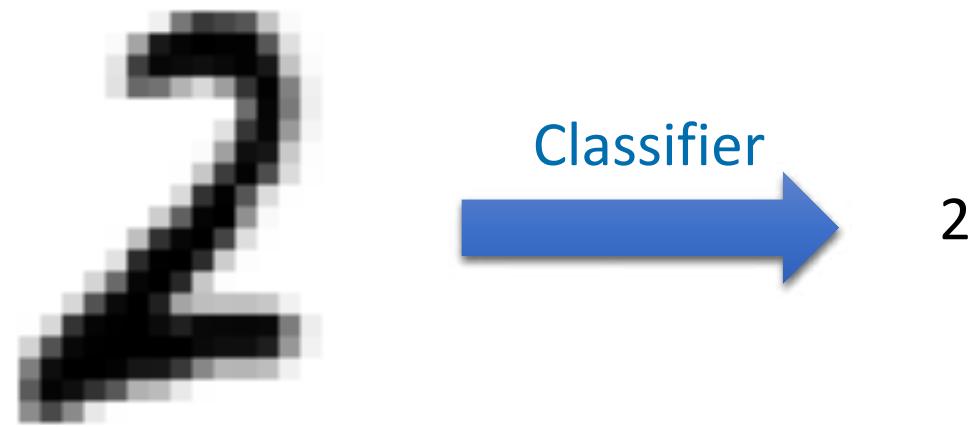


Image classification

- PASCAL Visual Object Classes (VOC) Challenge 2010
 - 21,738 images
 - 20 classes

Person	Animals	Vehicles	Indoor
person	bird cat cow dog horse Sheep	aeroplane bicycle boat bus car motorbike train	bottle chair table potted plant sofa tv/monitor

20 classes



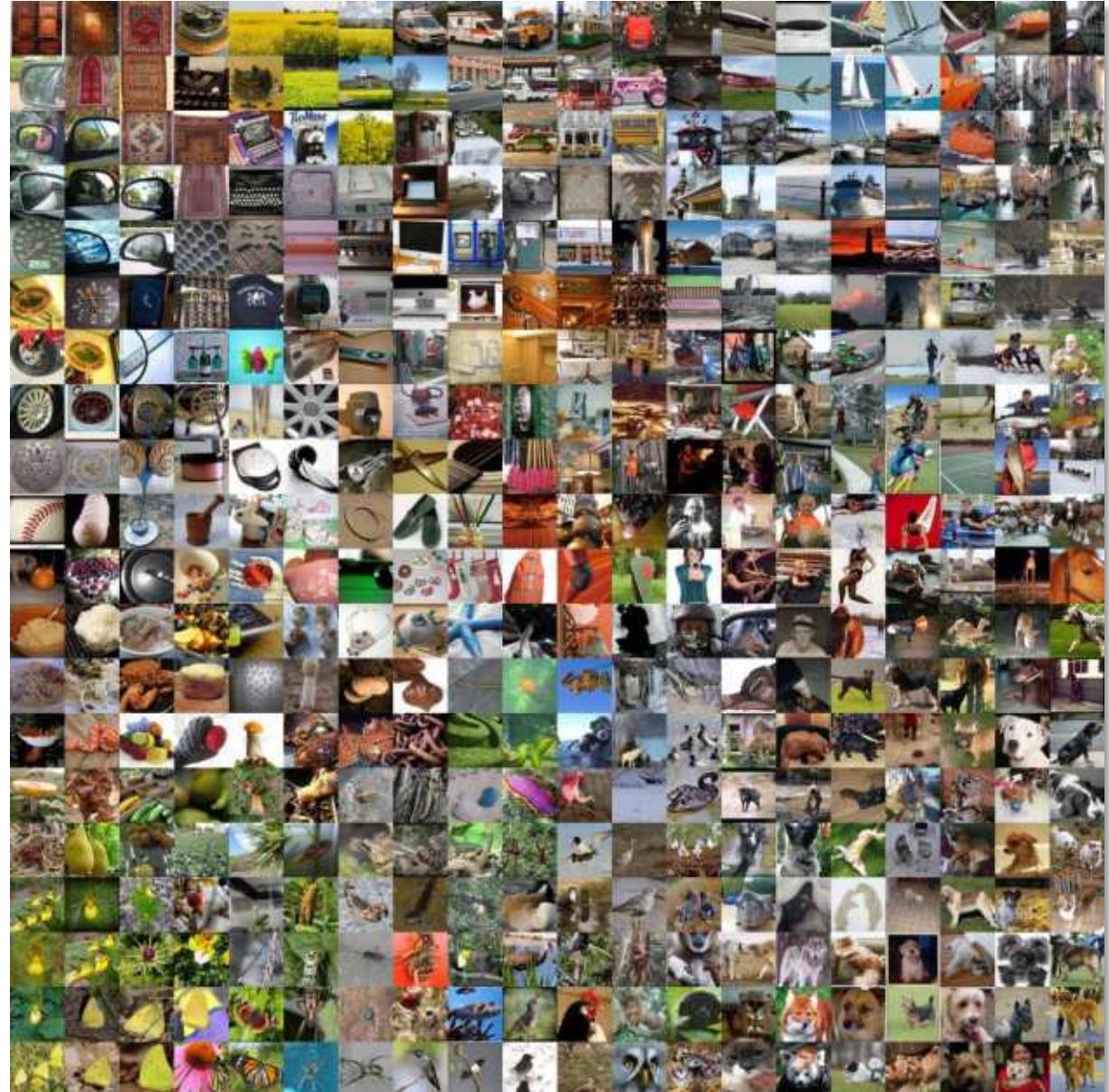
20 classes of images

ImageNet challenge

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
 - Train an algorithm that can classify 1.2 million images into 1,000 classes.



Fei-Fei Li



<http://www.image-net.org>

Class ID	Category
0	tench
1	goldfish
2	great white shark
3	tiger shark
4	hammerhead
5	electric ray
6	stingray
7	cock
8	hen
9	ostrich
...	...

The 1,000 classes in the ImageNet challenge dataset.

Image classification



Cat

Probability

Cat: 85%

Tiger: 10%

Dog: 1%

...

Image classification

- 1,000 classes seem a large number.
- But ImageNet does not have a class for handbag.
- And ImageNet does not have a class for dinosaur.

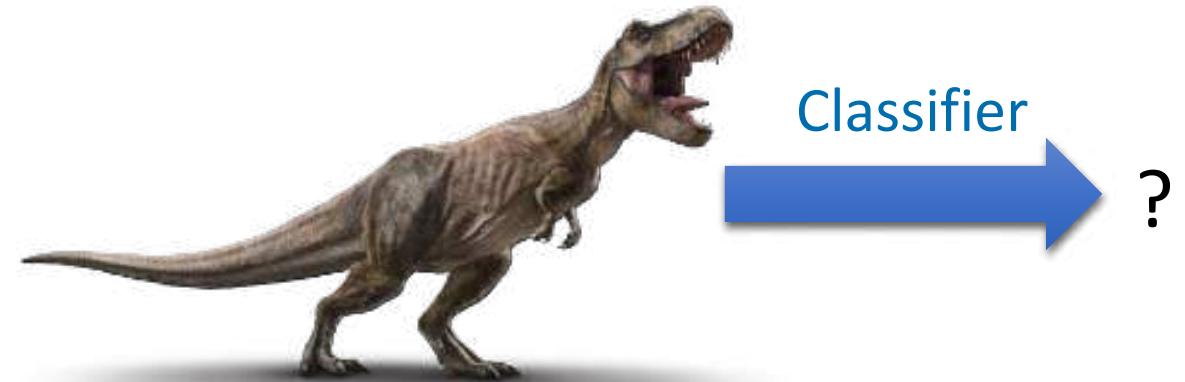


Image classification

- Facebook, Google, Microsoft, OpenAI collect their own datasets, which can have more images and more classes.
- You may have a try of their image classification algorithms.
 - <https://cloud.google.com/vision/>
 - <https://azure.microsoft.com/en-gb/services/cognitive-services/computer-vision/>

AI & Machine Learning Products

Contact sales

Try free

Labels

Web

Document

Properties

Safe Search

JSON



test.png

Handbag	99%
Bag	97%
Black	96%
Fashion Accessory	90%
Shoulder Bag	88%
Product	87%
Leather	84%
Product	80%
Tote Bag	73%
Brand	64%
Strap	59%
Luggage & Bags	55%

Image classification

- Recently there are efforts to collect even larger datasets from Internet.
- Each image is not assigned a simple label, but paired with text.
- One example is LAION-5B.
 - 5.85 billion image-text pairs.
 - Multiple languages.



french cat



french cat



How to tell if your
feline is french. He
wears a b...



イケメン猫モデル
「トキ・ナントケット」がかっこいい -
NAVERまとめ



Hilarious pics of funny
cats! funnycatsgif.com



Hipster cat



網友挑戰「加幾筆畫出最創意貓咪圖片」,
笑到岔氣之後我也手

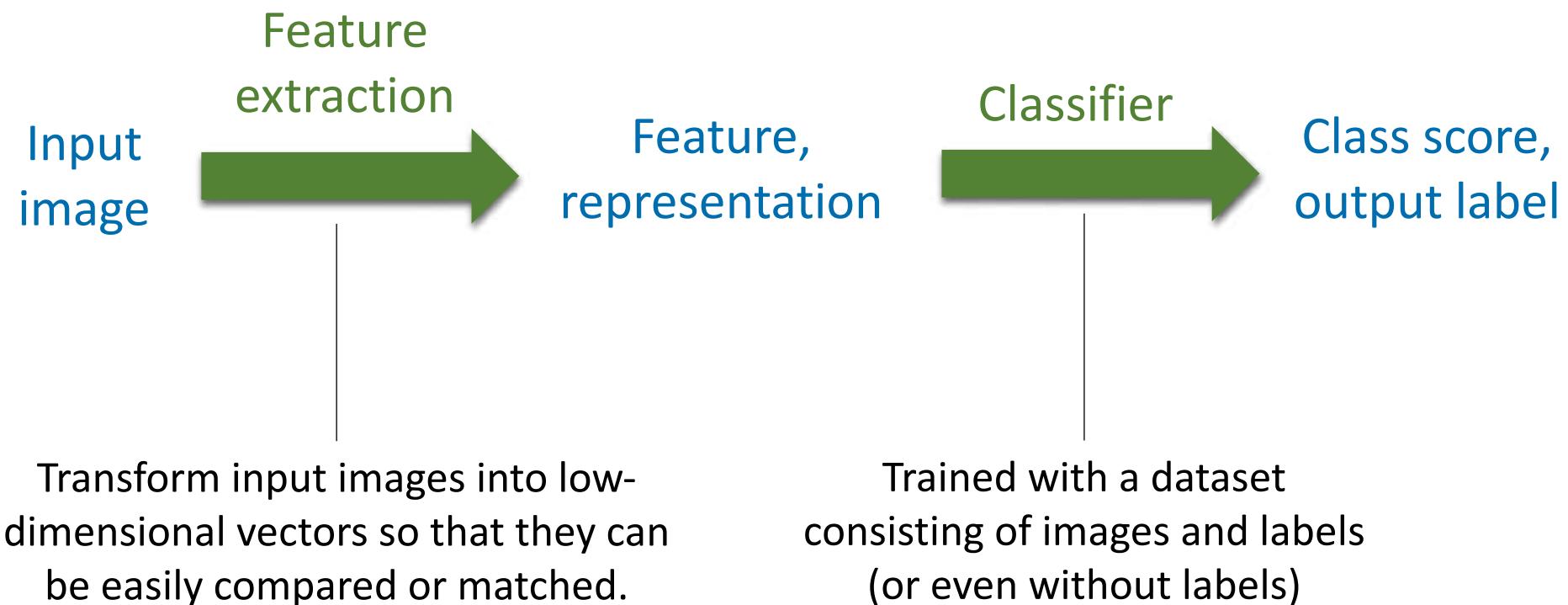


cat in a suit Georgian
sells tomatoes



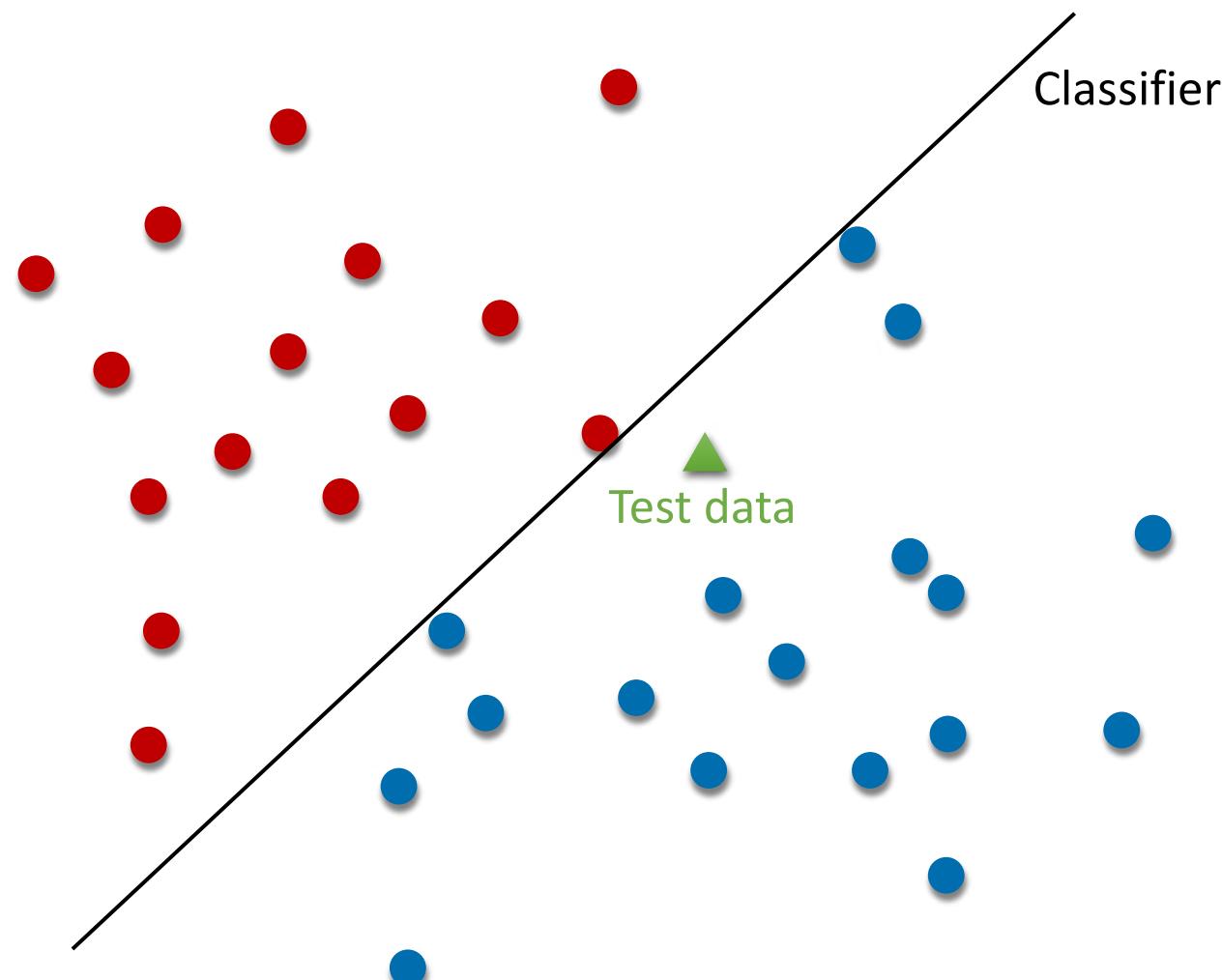
French Bread Cat Loaf
Metal Print

How does image classification work?



How does image classification work?

- Normally, we split the dataset into two parts:
 - Training set (for training the classifier)
 - Test set (for evaluating the performance of the classifier on data it has never seen)



Distribution of data from two classes in the feature space

Image classification

- We will go through a simple example and introduce basic concepts in image classification.
- Dataset
 - MNIST (Modified National Institute of Standards and Technology)
 - Handwritten digit recognition
 - 60,000 training samples, 10,000 test samples
 - Each sample is from one of the 10 classes (digit 0 to 9)

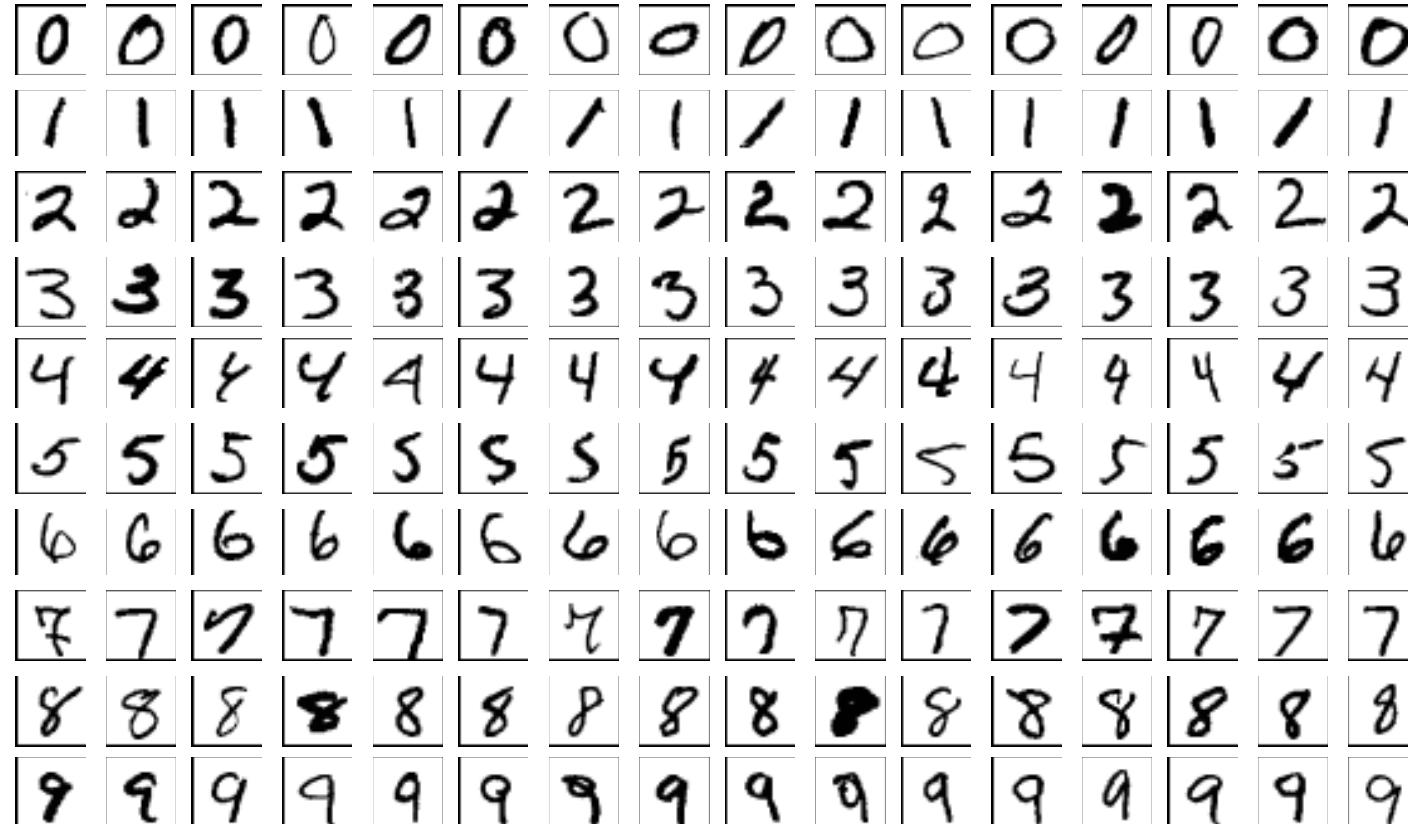
MNIST dataset

4028150880327726475572928468650087617112740077638420140578274711366
5071116767966414311224108763400633017113109975414895351982739901029
8468682467933943144705960444612336459685608641865284554770782237018
76953465018828357808571101378507110114527623028596972136418240510226
934777149069842728100783331976131605747595849918501380348220251514889
82049962335648092836957294912860709116759919592504108908989425198980
35517216919955162286714604033223689853854520563283995194671313660901
94368160413174951001162198403649071657525185470670258104571851900607
8857389886823975629288168879180172075190209862393802111142972512199
148534347750748815395976903639821286855394125151441435912233029009
93190975492010514933615252209266012030255795508950325908845884546549
6928545799218340783934056219260061287982047750564674307507420899404
12845278113035703193631773084826529739099642972116747596821445161325
90666367728608302983253980019513960141712379749939282718091017796999
21010452828351781129784030788477858498138031795516574935471208160734
28308789084458566309376893495891288681379011970817457121130621280766
41992780136134111560707232522949810161274000822922799275134941756283

Each sample is a 28x28 image. We know the class (0 - 9) for each sample.

An exemplar sample: image  , label 2.

MNIST dataset



For each of the ten classes, the samples look like this.
You will notice the variations of shape, stroke strength and writing style.

Pre-processing

- MNIST performed pre-processing so machine learning model only needs to learn the patterns of the digits, not other factors (size, orientation etc).
 - Detect where the digit is in a much bigger image;
 - Normalise the size of each digit to 28x28;
 - Normalise the location, place the mass centre of the digit in the centre of the image.
 - Perform slant correction, which shifts each row in the image so that the principal axis becomes vertical.
- MNIST provides two versions of datasets.
 - Regular dataset (centred)
 - Deslanted dataset (centred + deslanted)

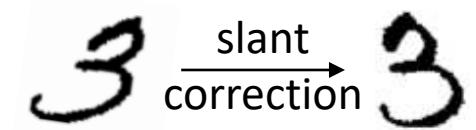
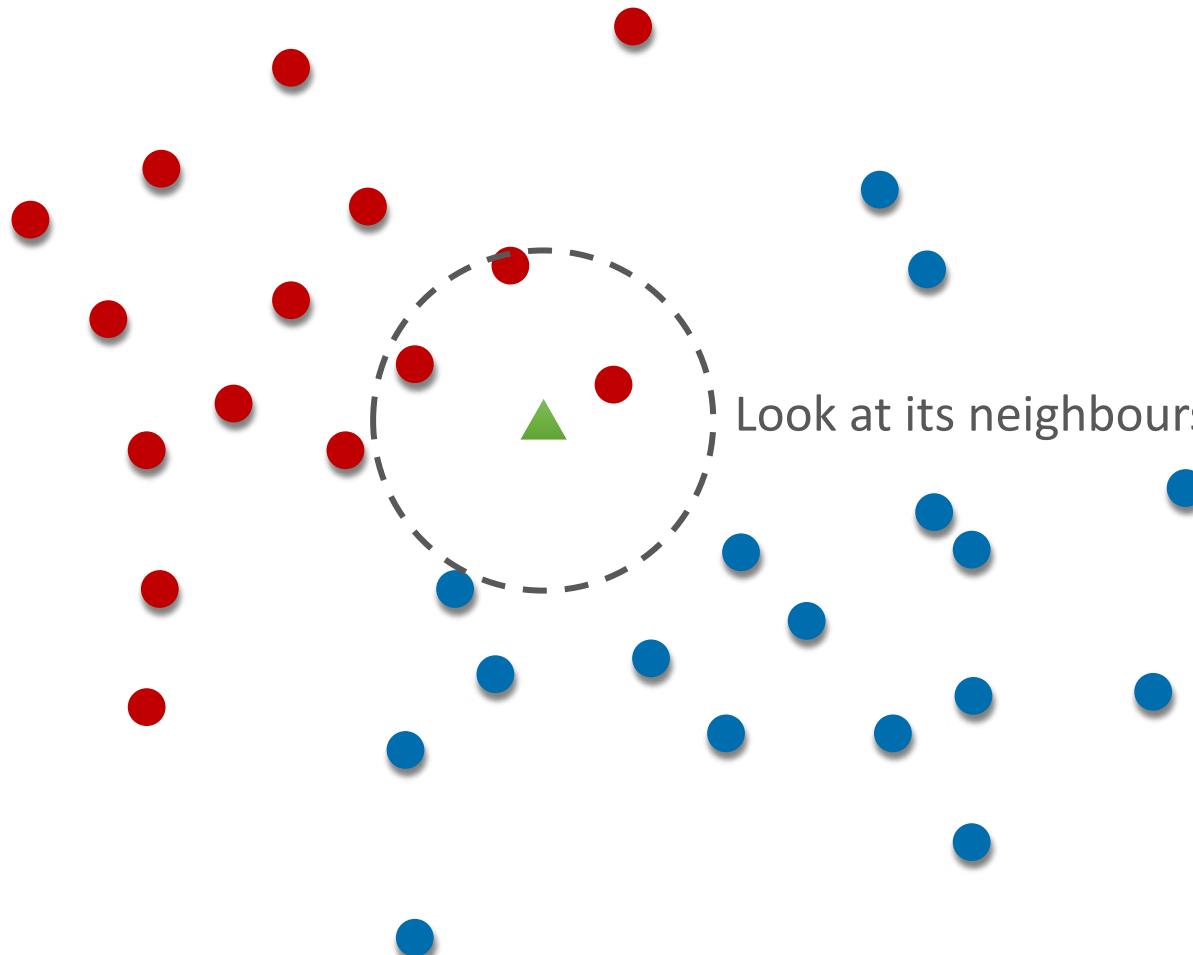


Image classification

- Feature extraction
 - Hand-crafted (e.g. pixel intensities, HOG, or other manually defined feature descriptors)
 - Learnt features (e.g. CNN, automatically learnt by the algorithm)
- Classifier
 - K nearest neighbours (KNN)
 - Support vector machine
 - Neural network
 - Vision transformer
 - ...

K nearest neighbours (KNN)



Which class does the test data (triangle) belong to?

K nearest neighbours (KNN)

- It is a non-parametric classifier.
- If $K = 1$, each test data point is assigned the class of its nearest neighbour.
- If $K > 1$, we compute K nearest neighbours and the test data point is assigned the class given by majority voting.

How do we define neighbours?

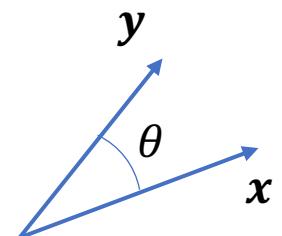
- We need to define a distance metric between data points x and y .

- Most typically, Euclidean distance

$$D(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

- In our case, x has $28 \times 28 = 784$ dimensions, which are 784 pixel intensities.
- Sometimes, if each dimension of the feature x has quite different scales, it is better to normalise the feature vector. For example, normalise each dimension to a Gaussian distribution $N(0,1)$.
- Other metric may also be used, such as the cosine distance

$$D(x, y) = \frac{x \cdot y}{\|x\| \|y\|} = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$$



Distance metrics

- The Euclidean distance is the L^2 -norm of the vector between x and y .

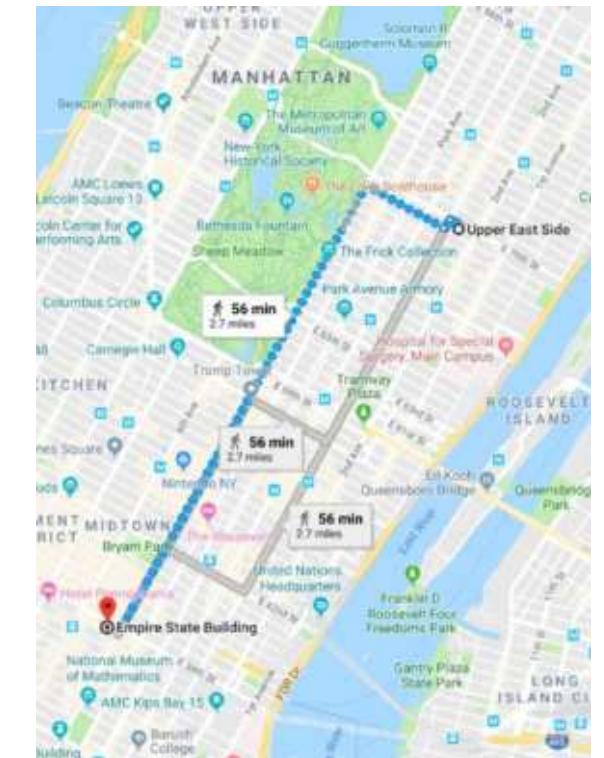
$$D_2(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^2 \right)^{1/2}$$

- The Manhattan distance uses the L^1 -norm.

$$D_1(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- The more general form is the L^p -norm.

$$D_p(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$



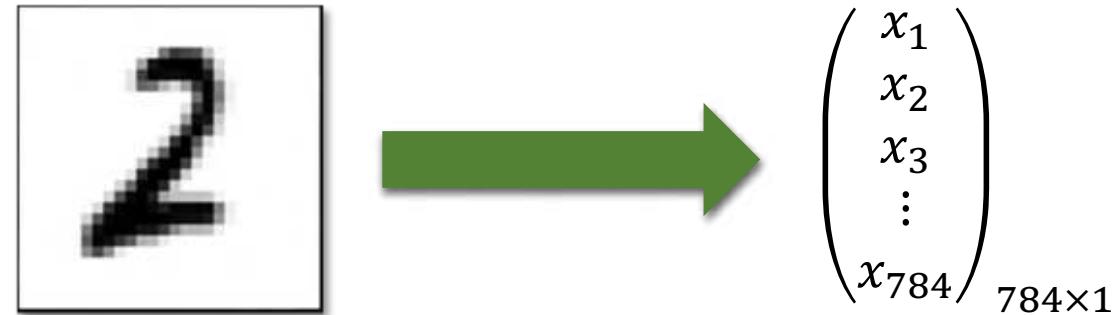
Manhattan distance

Image feature

- Simply 28×28 pixel intensity values

- 784-dimension vector
- You may remember simply using pixel intensities are sensitive to intensity change, rotation, scaling etc.
- Let us assume pre-processing has taken care of these factors.

- Let us use the Euclidean distance to compare two feature vectors.



Flatten the image into a 784-D feature vector

K nearest neighbours (KNN)

- It seems working well!
- Also, $K = 1$ already works.

2	2	2	2	2	2	2	2	2	2
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	3	7	7
8	8	8	8	5	8	8	8	5	8
6	0	6	0	6	6	6	0	6	6

input image ten nearest neighbours

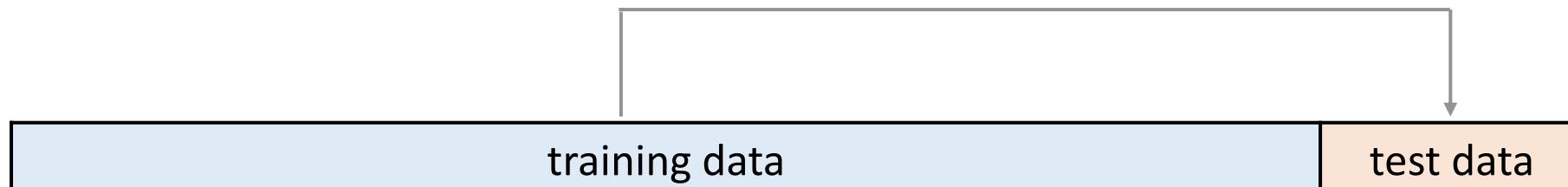
K nearest neighbours (KNN)

- There is one parameter, K, for this algorithm.
- It is called a hyper-parameter.
- What K shall we use? What distance metric shall we use?
 - It is applicant-dependent.
 - It is often just trial and error (parameter tuning). Try a lot of parameters, train your model and see which ones work the best.

Parameter tuning

- Can we tune the hyper-parameters to see which ones work best on the test set?
 - Not a very good practice. The test set is for evaluating the model performance on unseen data. We had better only use it once at the end.
 - For challenges and competitions, we often do not have access to the labels of the test set.

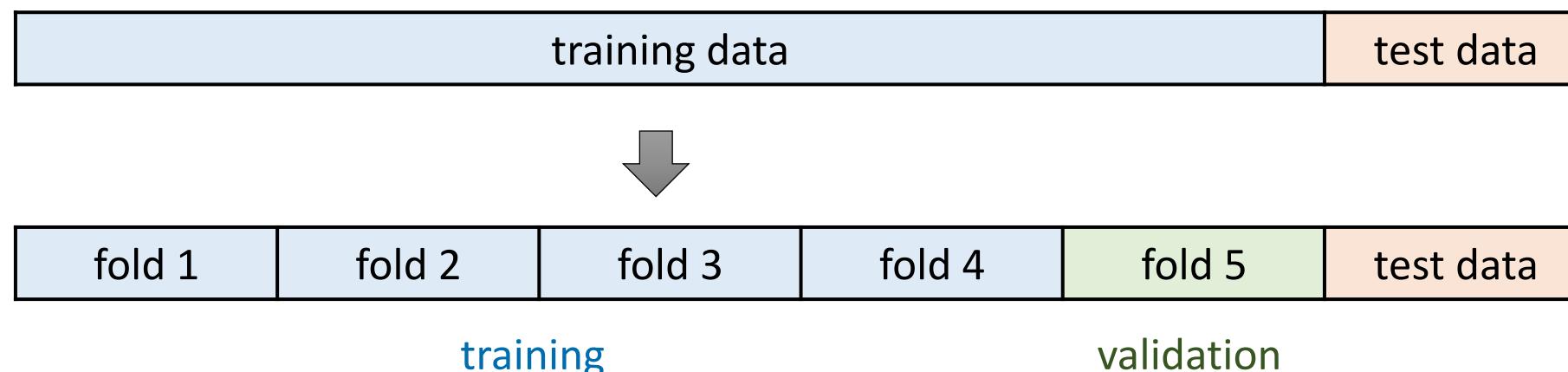
Not a very good practice for parameter tuning.



We often do not have
access to test labels.

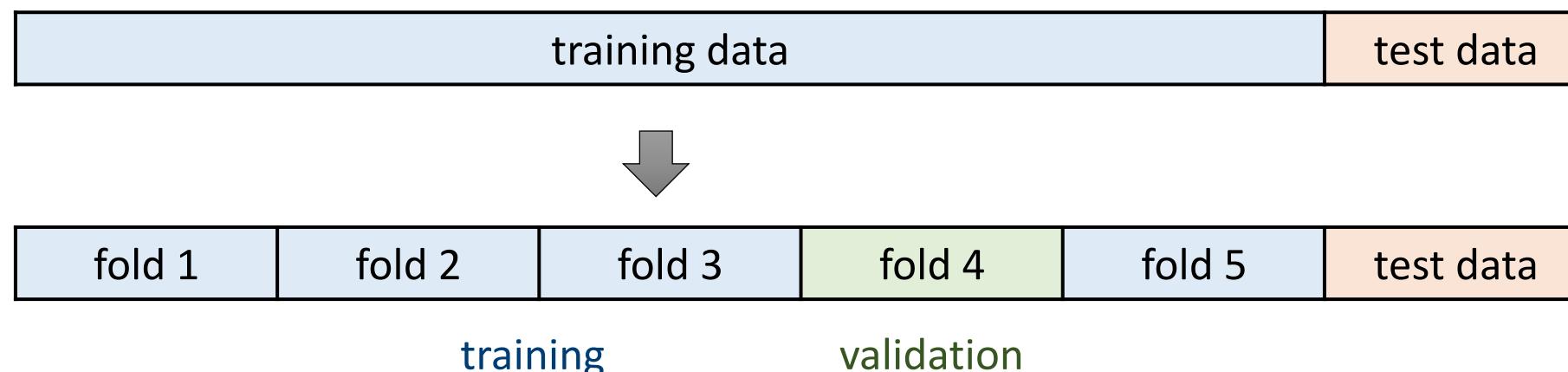
Cross-validation

- A common practice is to split the training set into 2 parts: a training set and a validation set, then tune the hyper-parameters on the validation set.



Cross-validation

- You can cycle through each fold, using it as a validation set.
- For this example, we can train the model for 5 times and calculate the average performance for some hyper-parameter values.
- Once you know the best hyper-parameters, we train model on the full training set and evaluate its performance on the test set for just one go.

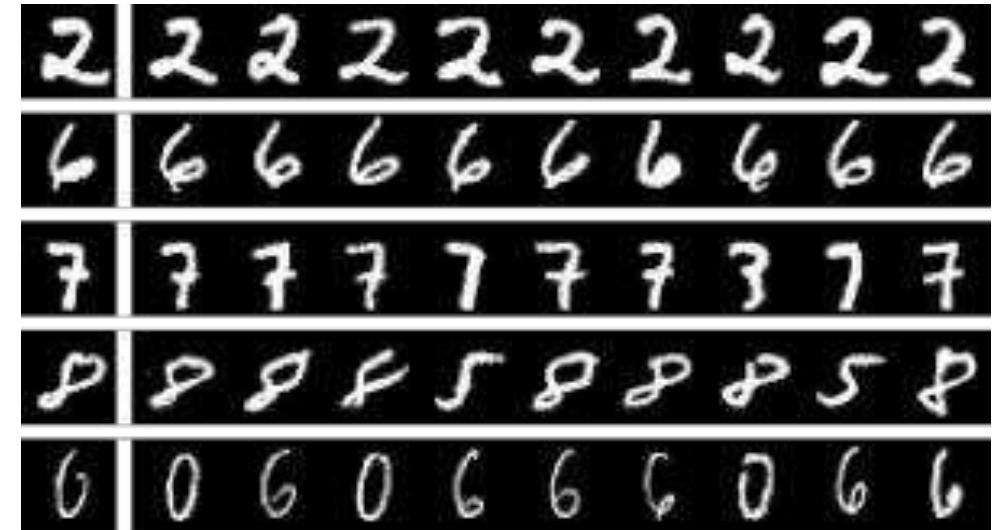


Cross-validation

- Split the dataset into
 - Training set: training the model.
 - Validation set: deciding what type of model and which hyperparameters are the best.
 - Test set: getting a final estimate of model performance.
- We expect the test performance to be possibly worse than the validation performance.
 - The reason is that we have been trying hard to fit our model onto the validation set, not the test set. They may have different distributions.
 - But this is what we expect for real applications, isn't it?

K nearest neighbours (KNN)

- We can find the best K by cross-validation.
 - We found K = 3 works best, achieving 2.4% error rate on the MNIST dataset.
- KNN seems a very good classifier for this classification task.



input
image ten nearest
 neighbours

K nearest neighbours (KNN)

- Advantages
 - No training step at all
 - Simple but effective
 - Multi-class classification
- Disadvantages
 - Storage and search are expensive.
 - All the training data need to be stored and searched.
 - For 60,000 training images in MNIST, it is affordable.
 - But for a data set of millions of training images, the computational cost becomes expensive.

Computational complexity

- N training images, M test images
- There is no training time.
- At test time, the computational cost to classify M test images is
 - $O(MN)$
- Maybe we want something opposite.
 - We are fine with slow training.
 - At test time, we want it to be fast.

Features

- For pre-processed digit images, maybe it is fine to use 28 x 28 pixel intensity values as the feature vector.
- But for general cases, simply using the Euclidean distance between the full images may not be a good choice.
 - Not invariant to scale, rotation etc.

2	2	2	2	2	2	2	2	2	2
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	3	7	7
8	8	8	8	5	8	8	8	5	8
0	0	6	0	6	6	6	0	6	6

input
image

ten nearest
neighbours



image



rotated image

Summary

- Introduce the image classification problem.
- We start from
 - a simple problem: digit recognition
 - a simple algorithm: K nearest neighbours (KNN)
 - KNN performs quite well on the MNIST dataset. But it has limitations for general image classification tasks.
 - The features we use today are also very simple.
- We will introduce more image classification algorithms.

References

- You can refer to other Machine Learning modules for more detailed description of machine learning concepts.
- In our course, we focus on the application of machine learning on image classification.

Image Classification II

Supplementary Slides

Dr Wenjia Bai

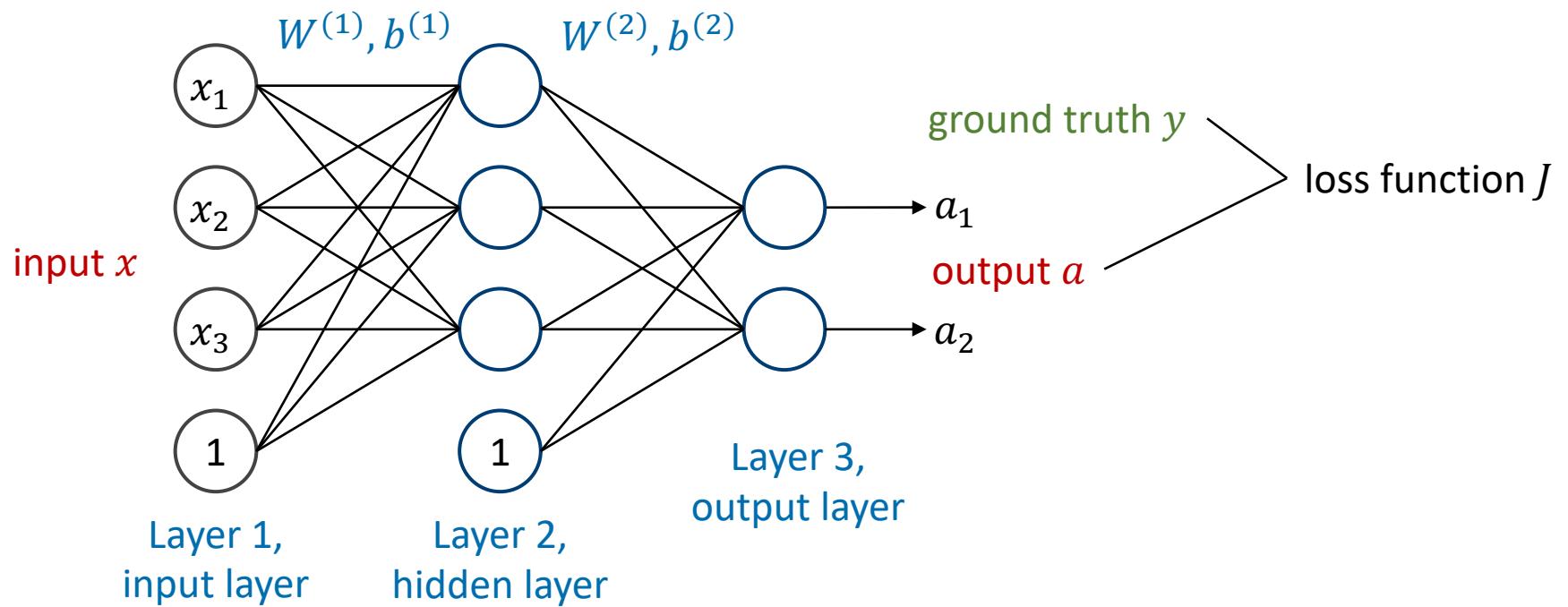
Department of Computing & Brain Sciences

Neural networks

- The mathematics of forward propagation and backpropagation is already covered in other machine learning and deep learning modules, which you can refer to.
- Therefore in the lecture, we only focus on those techniques specific to computer vision.
- Here, we provide some details of the mathematics in case you are interested in.

Neural networks

- A neural network is formed by putting many neurons into connection, where the output of a neuron can be the input to another.
- It often consists of several layers of neurons.



Multi-layer perceptron (MLP), which is a fully connected multi-layer network.

Neural networks

- To train a neural network, we need a training set $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m), \dots, (x_M, y_M)\}$, which are paired data and ground truth labels.
- We would like to find parameters W and b so that given input x , the output of the network a matches y as much as possible.
- For example, we can define a loss function like this,

$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m - y_m\|^2$$

m-th sample

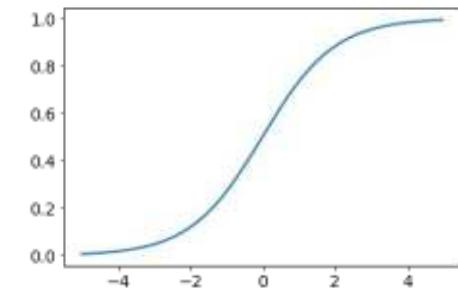
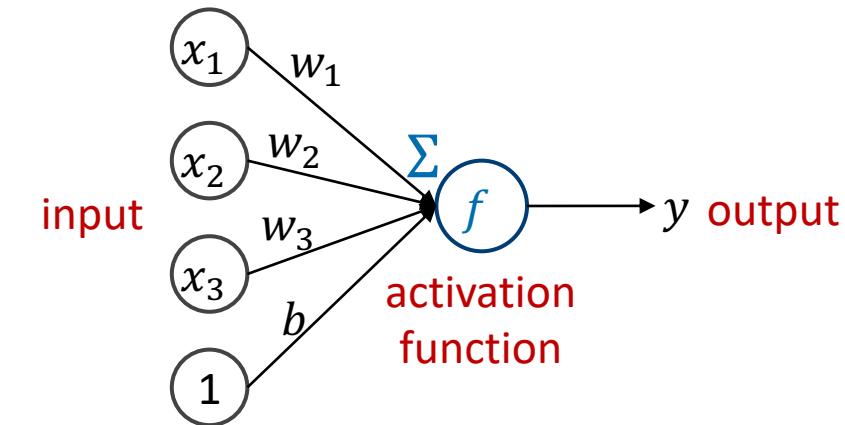
Optimising a neural network

- There are two technical questions here.
- Q1: How do we calculate the output of the network a given input x ?
 - A1: We use **forward propagation**.
- Q2: How do we find parameters W and b that minimise the loss function, so that a matching ground truth y as much as possible?
 - A2: We use gradient descent for optimisation and **backpropagation** to calculate the required gradient.

We know how a single neuron works

- The neuron is a computational unit that takes an input, applies an activation function and generates an output.

$$y = f\left(\sum_{i=1}^3 w_i x_i + b\right)$$

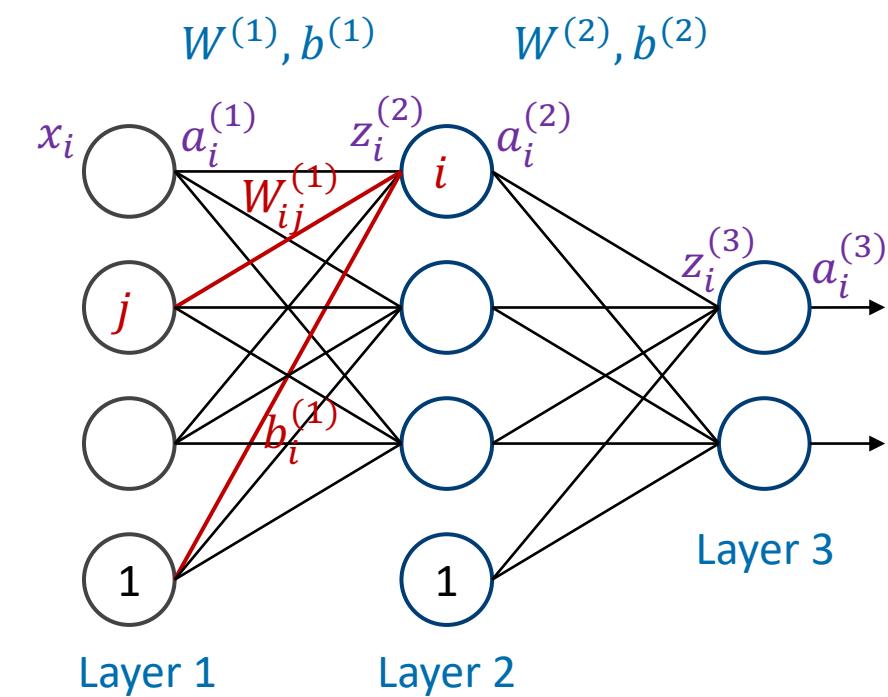


Sigmoid activation function

It works the same for multi-layer perceptron

- At Layer 1, the circles represent the input

$$a_i^{(1)} = x_i$$



It works the same for multi-layer perceptron

- At Layer 2, we calculate the input to each neuron, then apply the activation function.

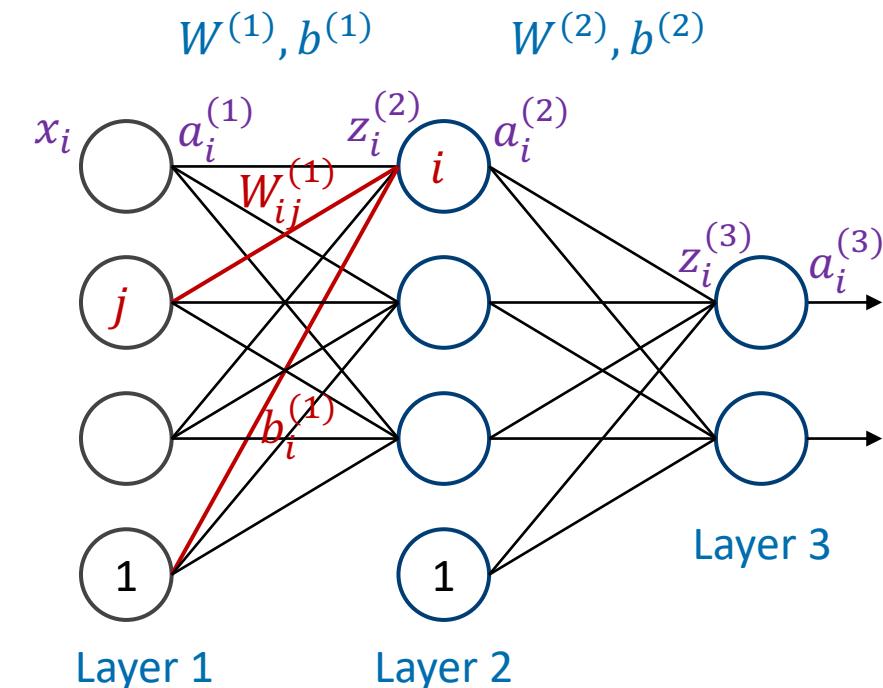
$$z_i^{(2)} = W_{i1}^{(1)} a_1^{(1)} + W_{i2}^{(1)} a_2^{(1)} + W_{i3}^{(1)} a_3^{(1)} + b_i^{(1)}$$

$$a_i^{(2)} = f(z_i^{(2)})$$

- Using matrix notation, we have

$$z^{(2)} = W^{(1)} a^{(1)} + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$
 Apply activation function element-wise.



We do the same for following layers

- At Layer 3, we calculate the input to each neuron, then apply the activation function.

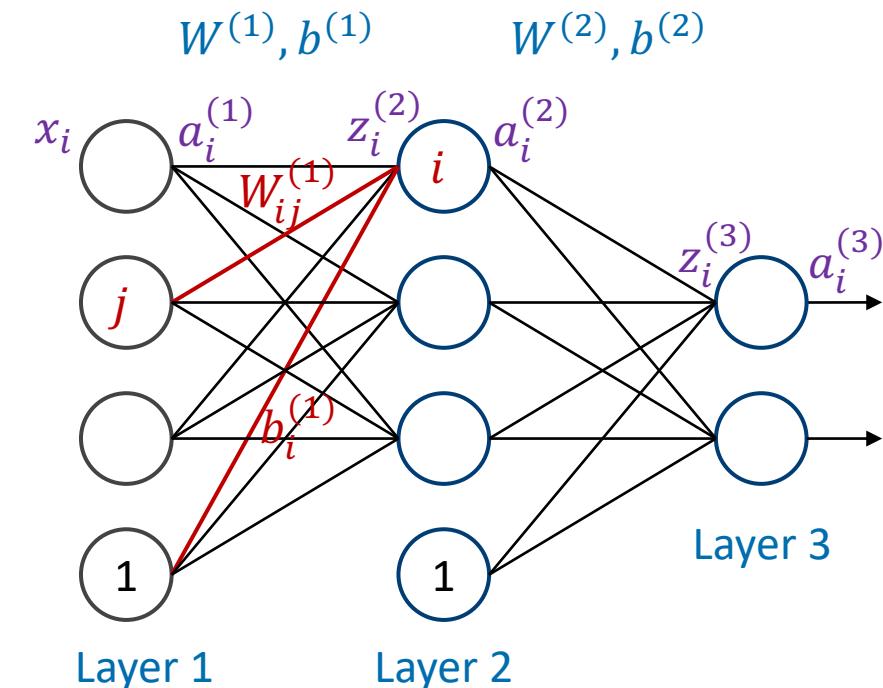
$$z_i^{(3)} = W_{i1}^{(2)} a_1^{(2)} + W_{i2}^{(2)} a_2^{(2)} + W_{i3}^{(2)} a_3^{(2)} + b_i^{(2)}$$

$$a_i^{(3)} = f(z_i^{(3)})$$

- Using matrix notation, we have

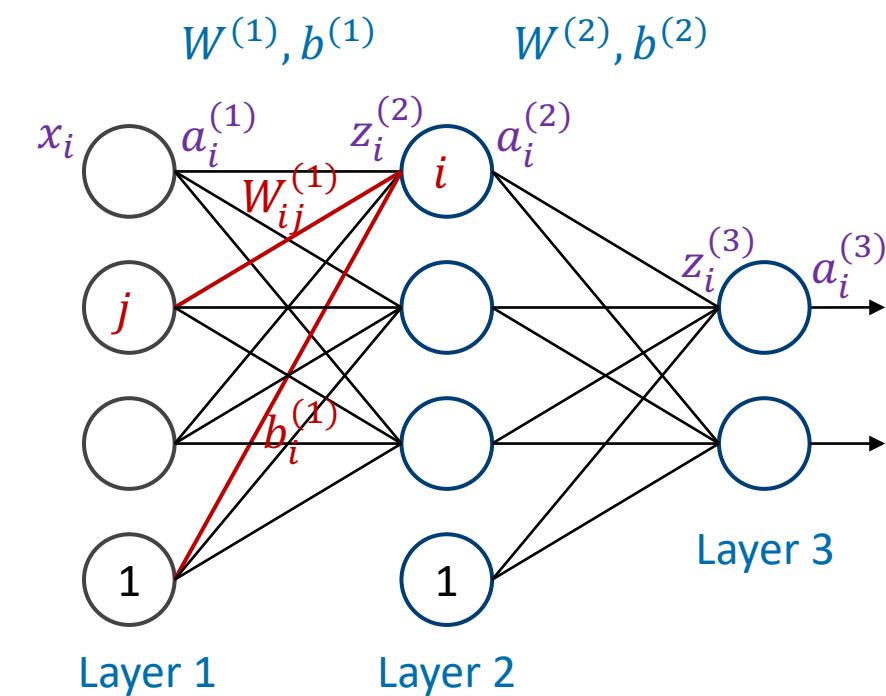
$$z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$



Notations

- $W^{(l)}$: weight matrix at layer l
- $W_{ij}^{(l)}$: weight for the connection between neuron j at layer l and neuron i at layer $l + 1$
- $b^{(l)}$: bias vector at layer l
- $b_i^{(l)}$: bias at layer l connecting to neuron i at layer $l + 1$
- $z_i^{(l)}$: total input to neuron i at layer l
- $a_i^{(l)}$: activation of neuron i at layer l
- $a_i^{(1)}$: input i at the first layer, $a_i^{(1)} = x_i$

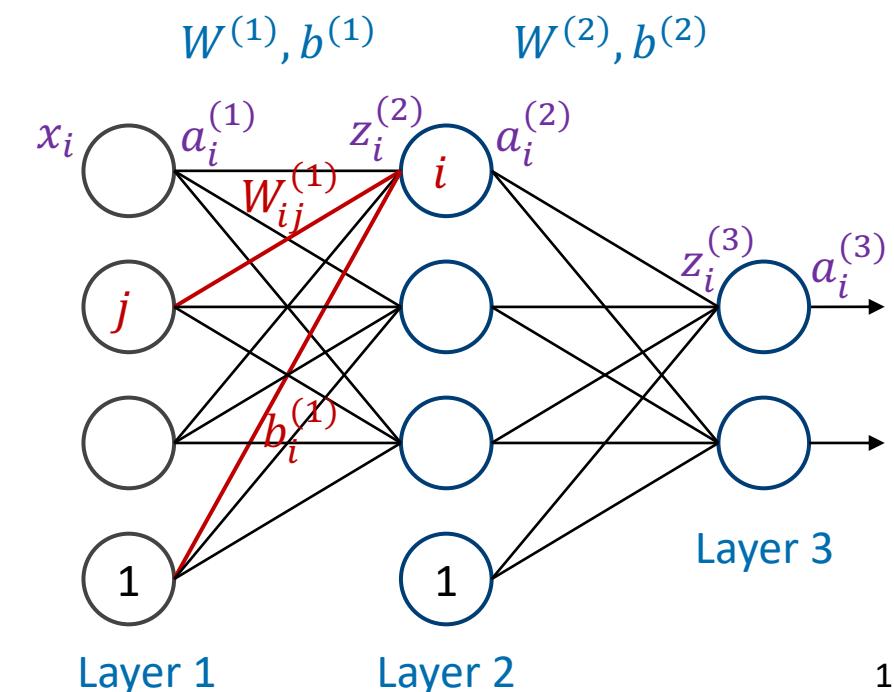


In this example,
 $W^{(1)}$ is a 3×3 matrix, $b^{(1)}$ is a 3×1 vector,
 $W^{(2)}$ is a 2×3 matrix, $b^{(2)}$ is a 2×1 vector.

Forward propagation

- Given a fixed setting of parameters W and b , the neural network computes the output given input x .
- At Layer 1, $a^{(1)} = x$
- At Layer 2, $z^{(2)} = W^{(1)}a^{(1)} + b^{(1)}$
$$a^{(2)} = f(z^{(2)})$$
- At Layer 3, $z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$
$$a^{(3)} = f(z^{(3)})$$
- In general, for Layer l , we have
$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

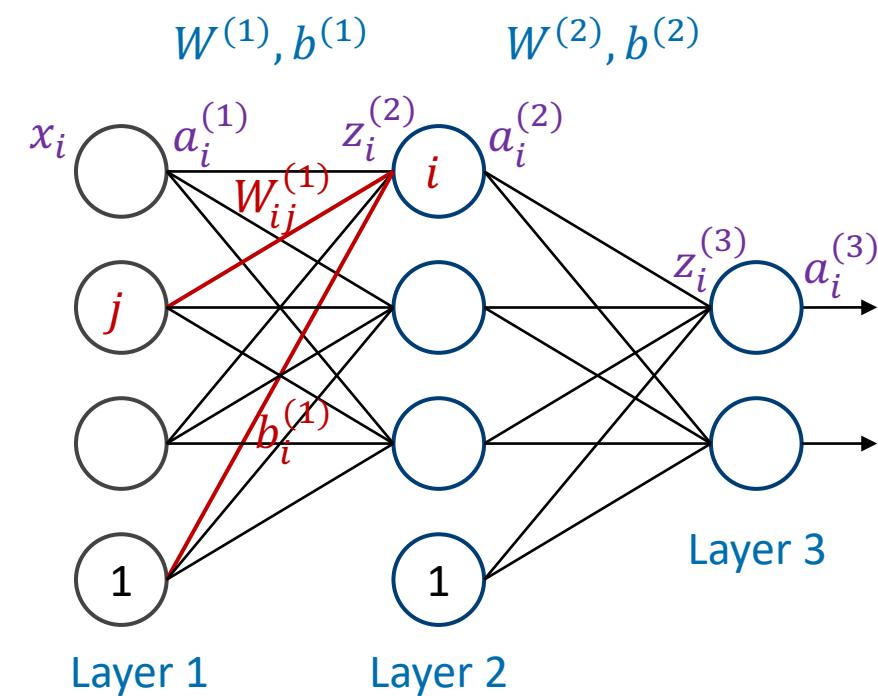


Forward propagation

- The calculation for each layer is the same.
- For Layer $l + 1$, we have

$$\begin{aligned} z^{(l+1)} &= W^{(l)} a^{(l)} + b^{(l)} \\ a^{(l+1)} &= f(z^{(l+1)}) \end{aligned}$$

- We do this layer by layer.
- This is called **forward propagation**.



Estimate parameters

- The first question is solved.
- The second question is to find parameters W and b that minimise the loss function $J(W, b)$.

$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m - y_m\|^2$$

Gradient descent

- Gradient descent is a technique for optimising a function with respect to some parameters.

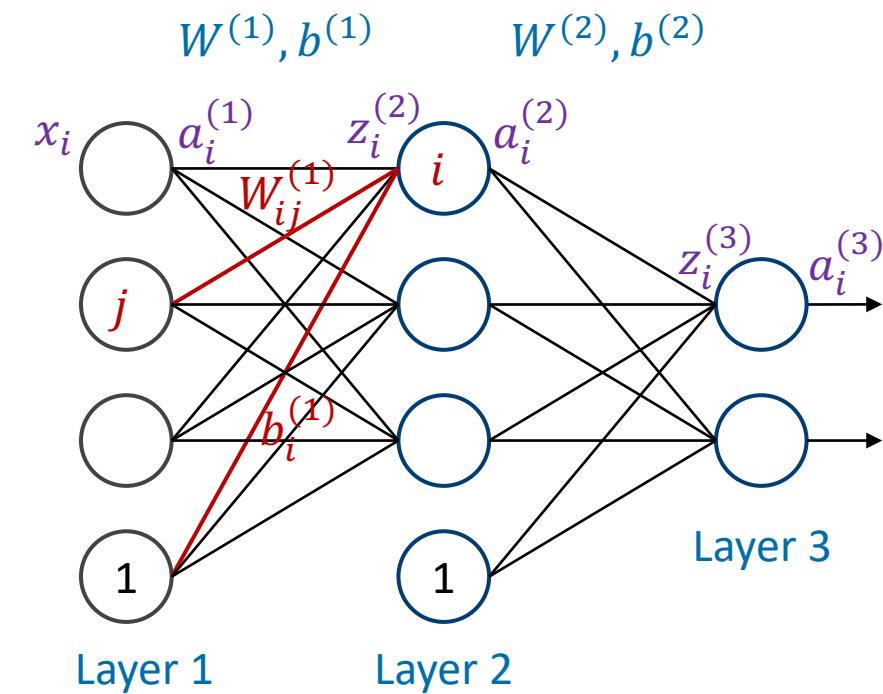
$$W = W - \alpha \frac{\partial J}{\partial W}$$
$$b = b - \alpha \frac{\partial J}{\partial b}$$

where α denotes the learning rate or step size.

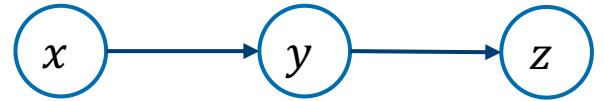
- As long as we know the gradient $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$, the second question is solvable.

Gradient descent

- The gradient $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$ can be calculated using the backpropagation algorithm.
- It is based on the chain rule in differentiation.



Chain rule



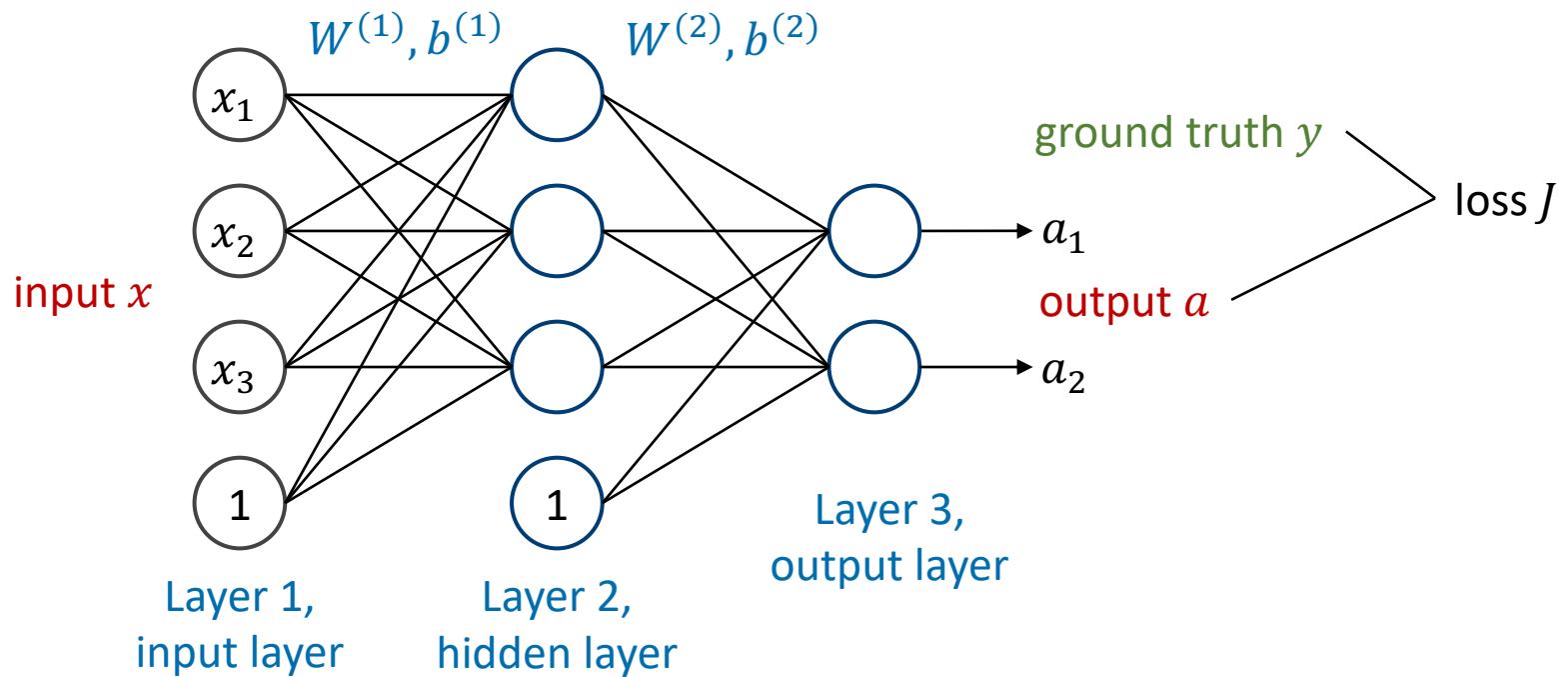
- Suppose we have a composition of two functions $z = g(f(x))$, where
$$y = f(x)$$
$$z = g(y)$$
- The chain rule expresses the derivative of the composition in terms of the derivative for each single function.

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Backpropagation

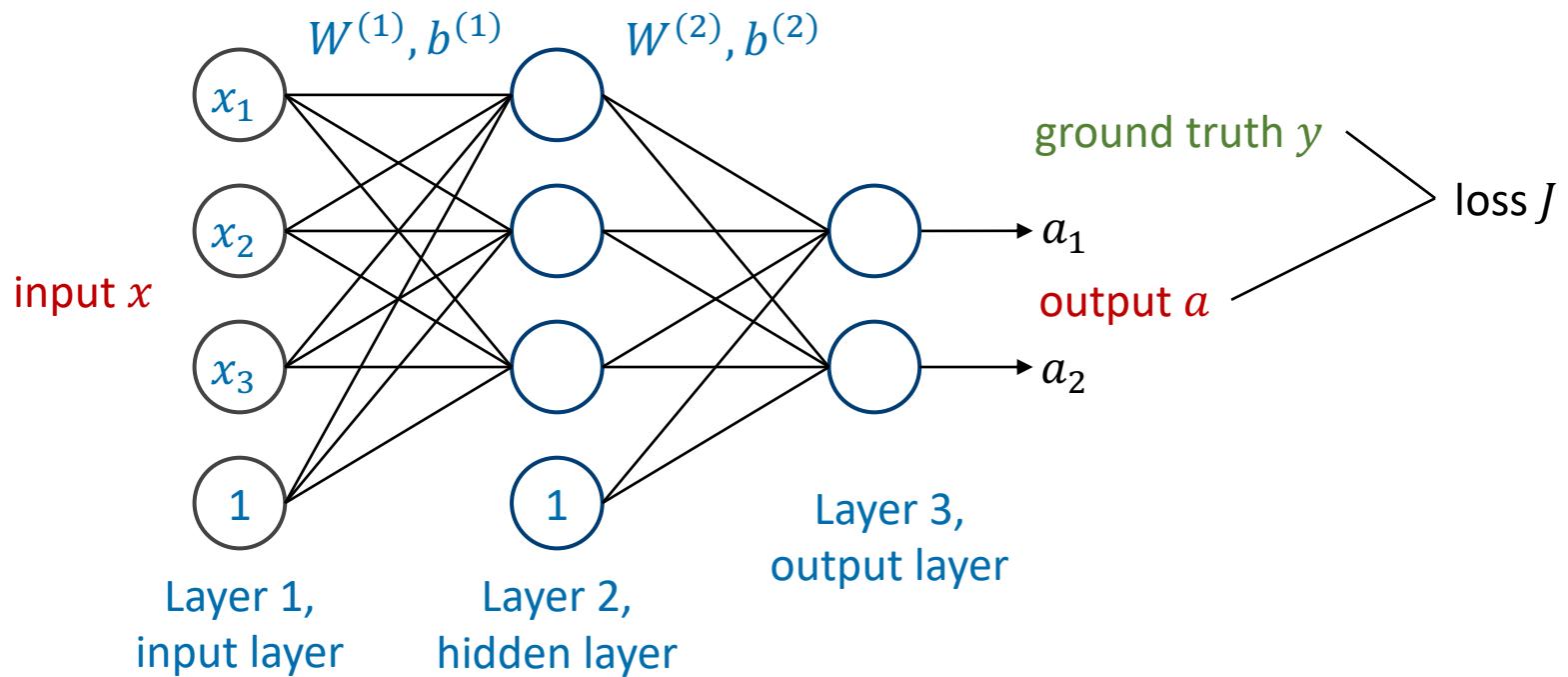
- The relation between J and a is simple,

$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m - y_m\|^2$$



Backpropagation

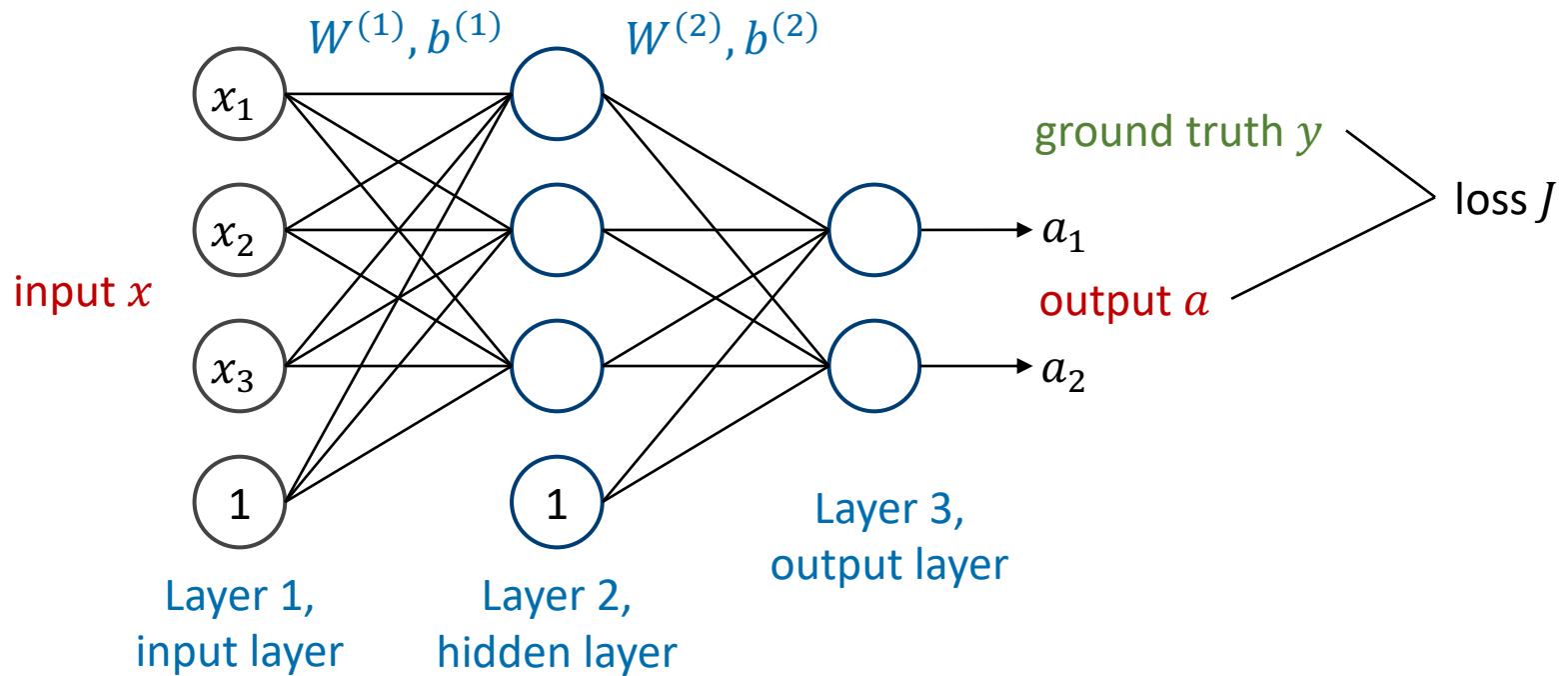
- We can easily derive $\frac{\partial J}{\partial a}$.



Backpropagation

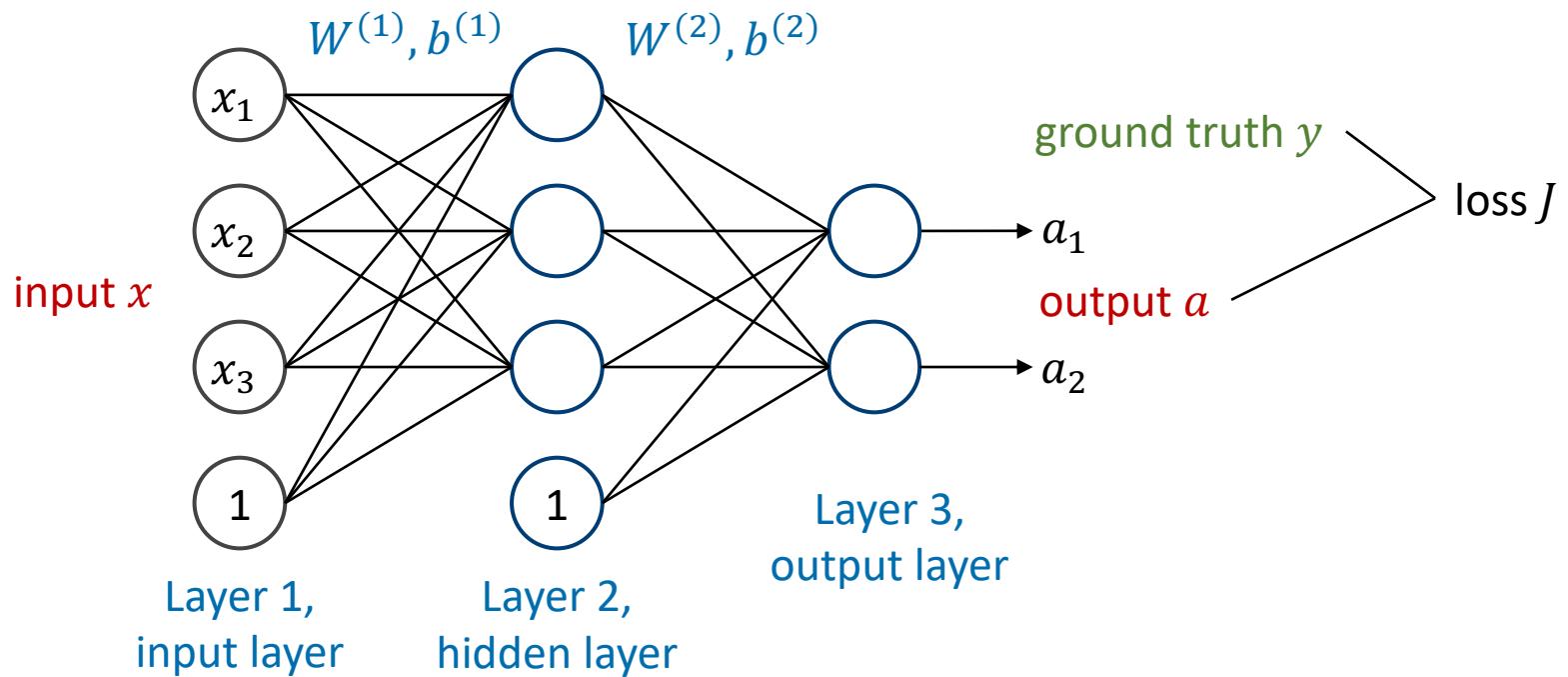
- The relation between $W^{(2)}$ and a is also simple,

$$a^{(3)} = f(W^{(2)}a^{(2)} + b^{(2)})$$



Backpropagation

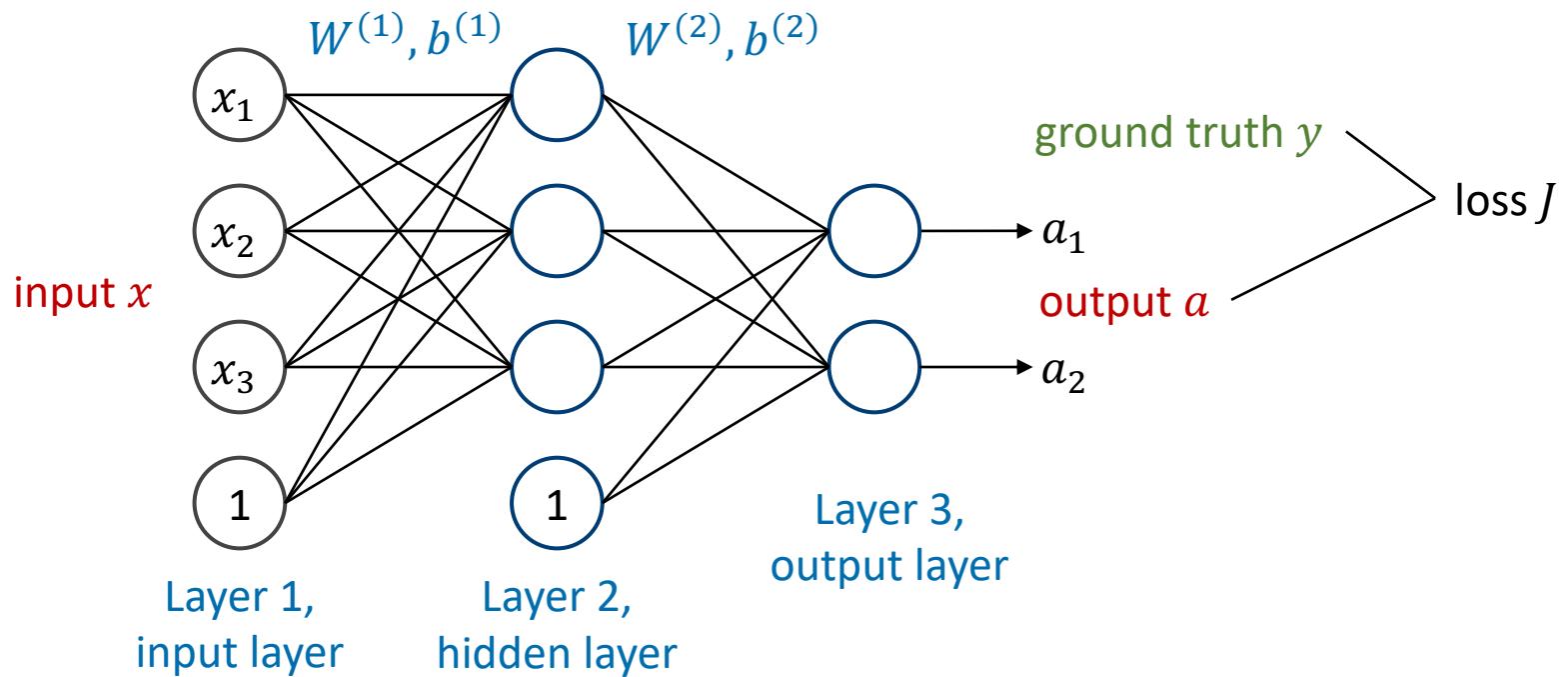
- We can easily derive $\frac{\partial a}{\partial W^{(2)}}$.



Backpropagation

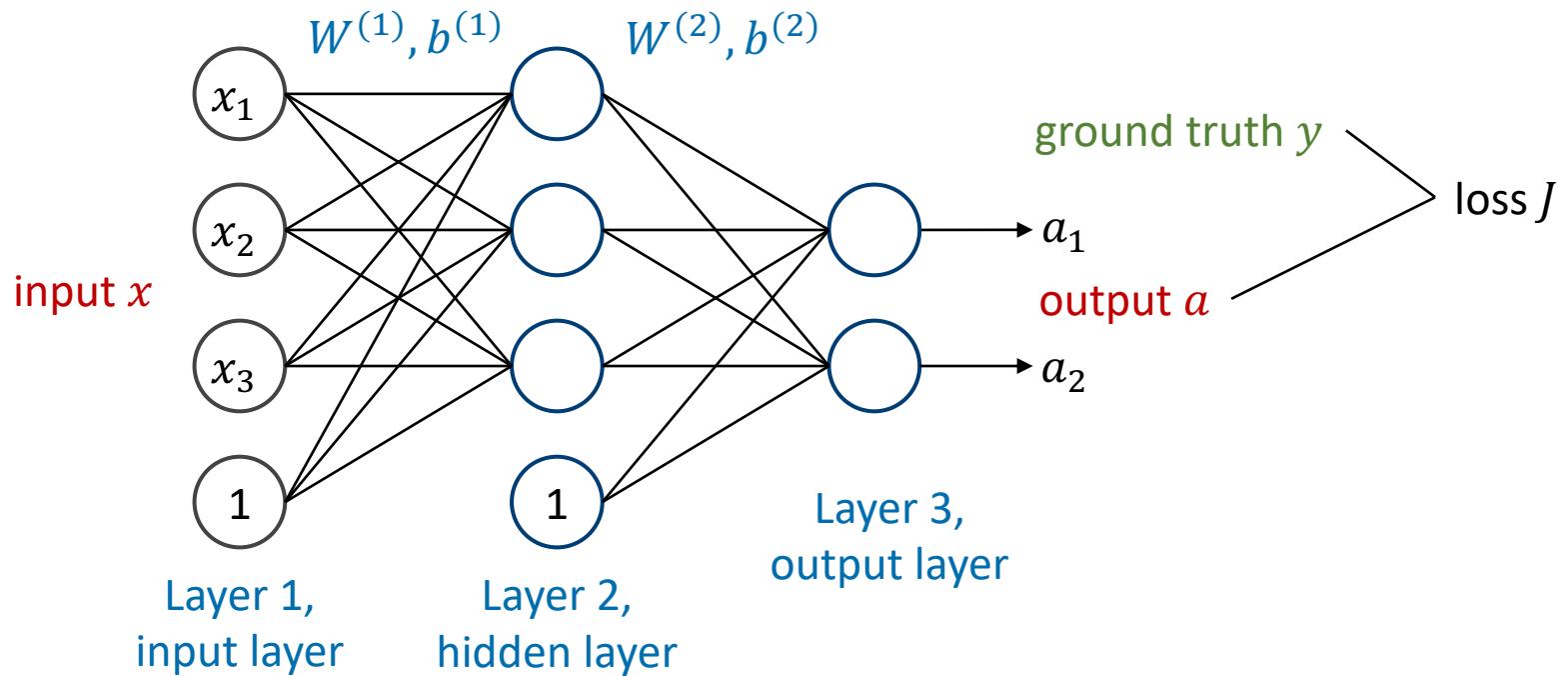
- Now using the chain rule, we can derive $\frac{\partial J}{\partial W^{(2)}}$ as

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial a} \frac{\partial a}{\partial W^{(2)}}$$



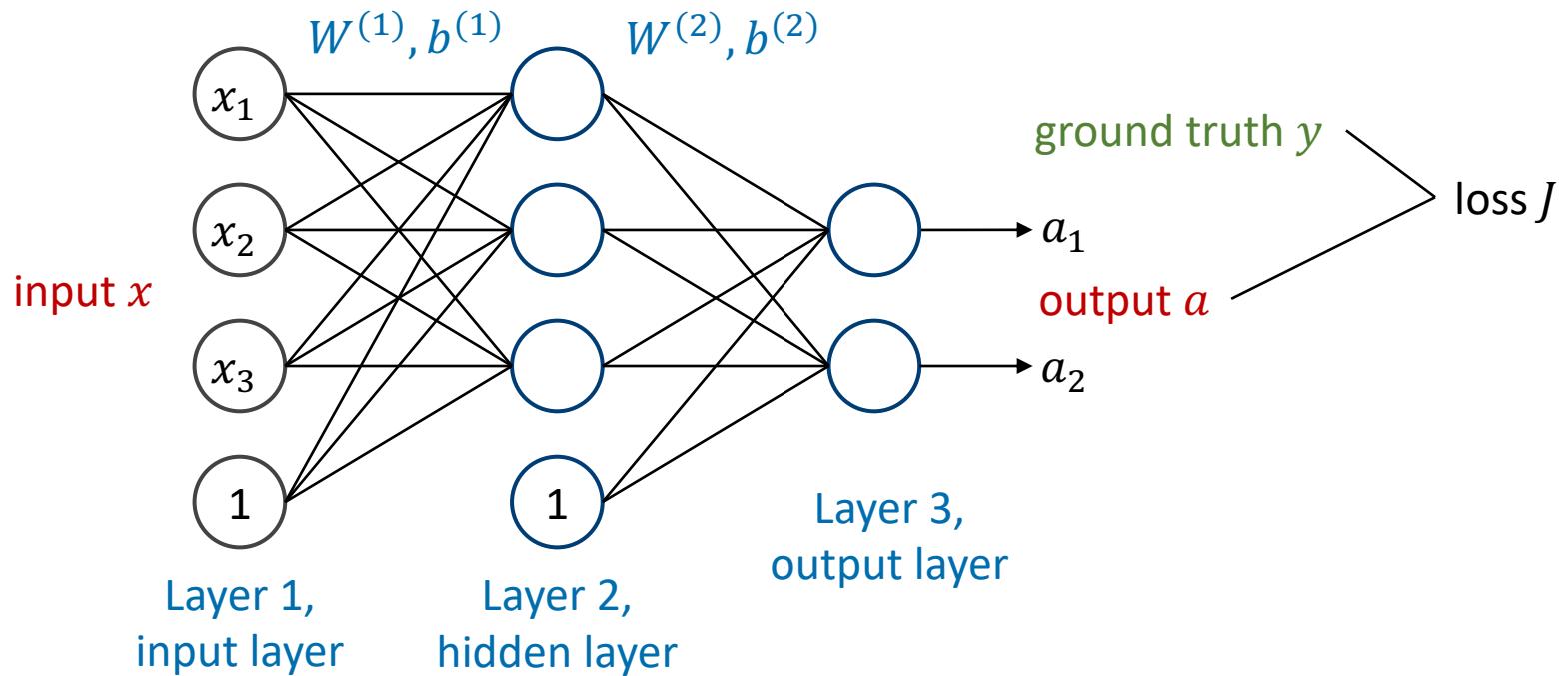
Backpropagation

- In the same way, we can propagate the gradient back layer by layer.
- As the end, we can calculate $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$ for all layers.



Backpropagation

- The detailed mathematical derivation is slightly longer.
- But the idea is simple, which is to use the chain rule and calculate an unknown derivative using known derivatives.



Backpropagation

- Let us look at the loss function

$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m^{(3)} - y_m\|^2$$

- For each sample, the loss is

$$J(W, b; x_m, y_m) = \frac{1}{2} \|a_m^{(3)} - y_m\|^2$$

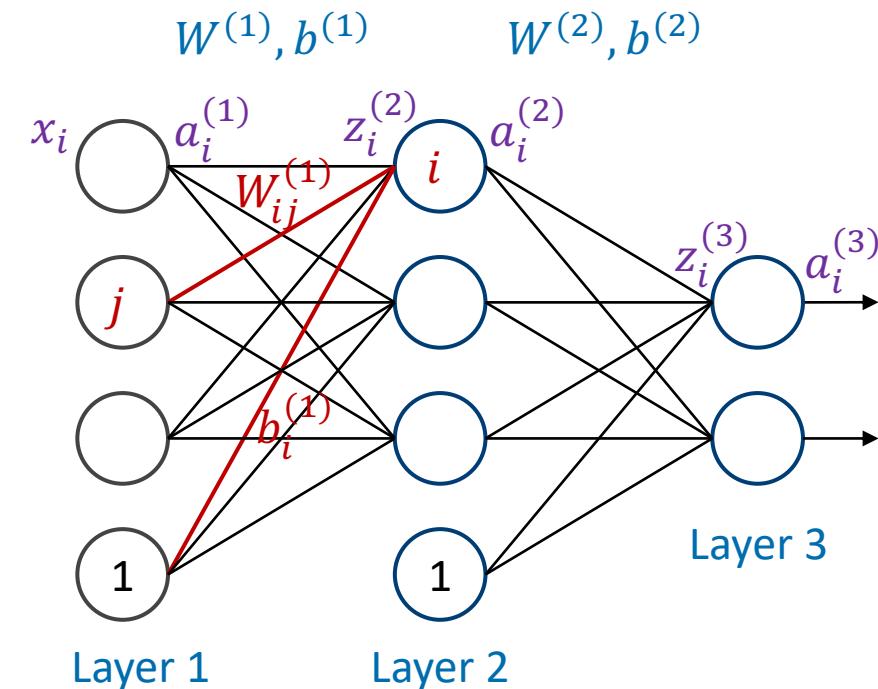
- Let us calculate the derivative for just one sample and ignore the sample index m for simplicity,

$$\frac{\partial J}{\partial a^{(3)}} = a^{(3)} - y$$
$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} = (a^{(3)} - y) \circ f'(z^{(3)})$$

Element-wise multiplication

A vector, length equal to the number of neurons on this layer

Derivative of activation function



Backpropagation

- For Layer 3,

$$\frac{\partial J}{\partial a^{(3)}} = a^{(3)} - y$$
$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} = (a^{(3)} - y) \circ f'(z^{(3)})$$

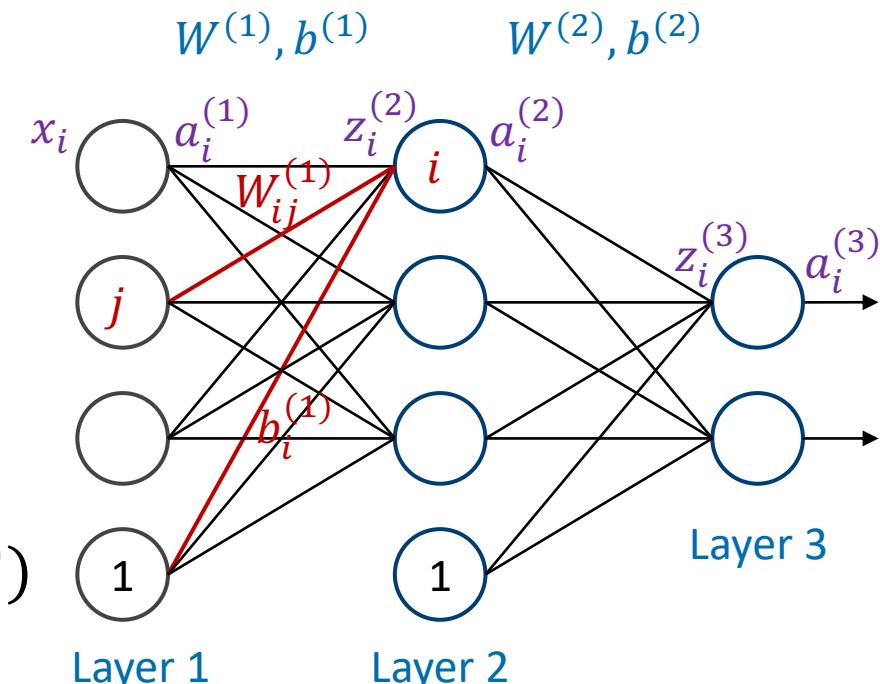
- For the previous layer, Layer 2,

$$\frac{\partial J}{\partial W_{ij}^{(2)}} = \frac{\partial J}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial W_{ij}^{(2)}} = \frac{\partial J}{\partial z_i^{(3)}} a_j^{(2)}$$

- We can use the matrix notation,

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} (a^{(2)})^T$$

| | |
I x J I x 1 1 x J
matrix vector vector



Backpropagation

- For Layer 3,

$$\frac{\partial J}{\partial a^{(3)}} = a^{(3)} - y$$

$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} = (a^{(3)} - y) \circ f'(z^{(3)})$$

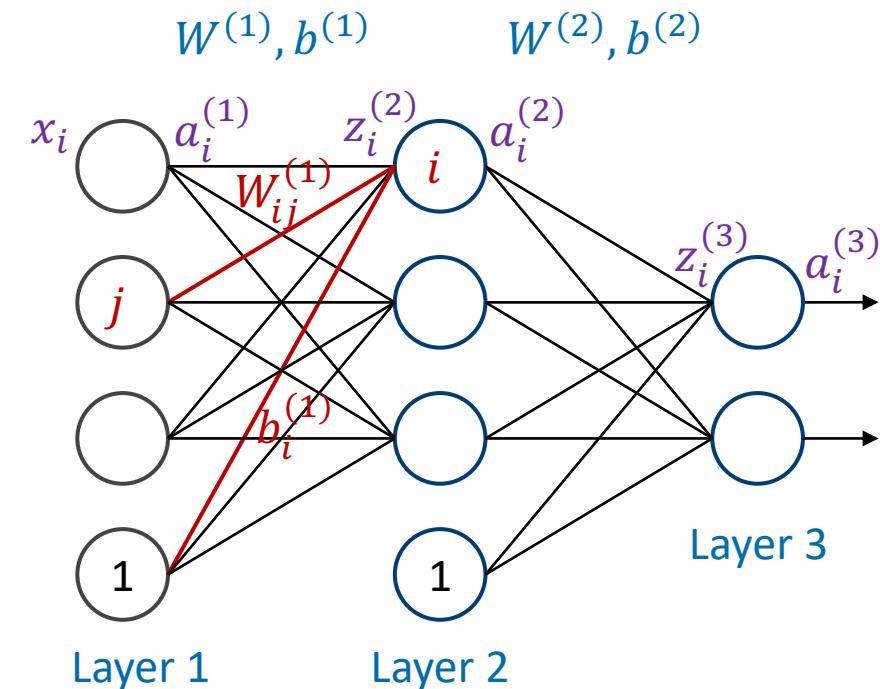
- For the previous layer, Layer 2,

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} (a^{(2)})^T$$

$$\frac{\partial b^{(2)}}{\partial J} = \frac{\partial z^{(3)}}{\partial J} \frac{\partial b^{(2)}}{\partial z^{(3)}} = \frac{\partial z^{(3)}}{\partial J}$$

$$\frac{\partial a^{(2)}}{\partial J} = \frac{\partial z^{(3)}}{\partial J} \frac{\partial a^{(2)}}{\partial z^{(3)}} = (W^{(2)})^T \frac{\partial J}{\partial z^{(3)}}$$

$$\frac{\partial z^{(2)}}{\partial J} = \frac{\partial a^{(2)}}{\partial J} \frac{\partial z^{(2)}}{\partial a^{(2)}} = ((W^{(2)})^T \frac{\partial J}{\partial z^{(3)}}) \circ f'(z^{(2)})$$



Backpropagation

- For Layer 3,

$$\frac{\partial J}{\partial a^{(3)}} = a^{(3)} - y$$

$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} = (a^{(3)} - y) \circ f'(z^{(3)})$$

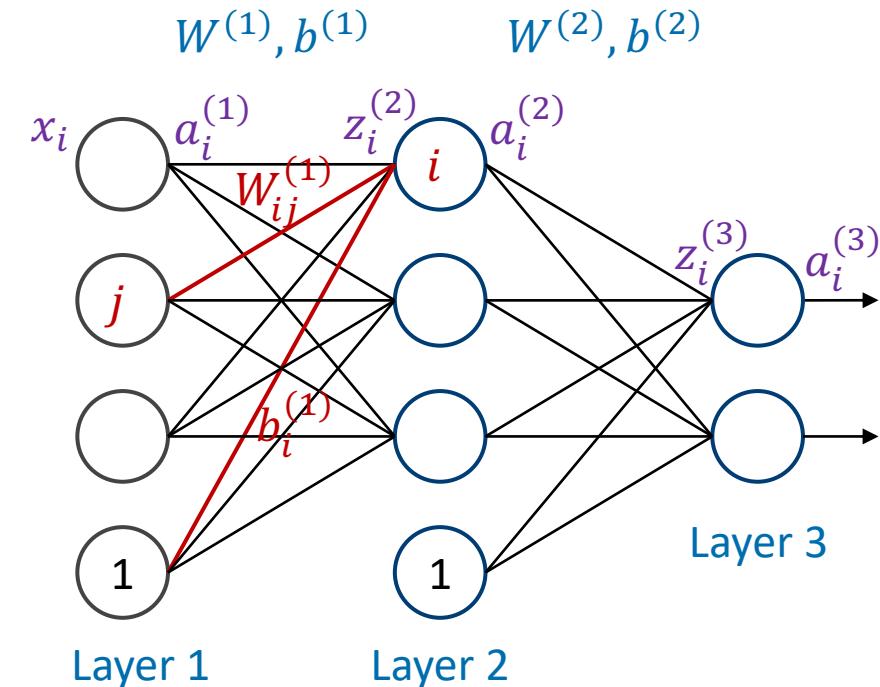
- For the previous layer, Layer 2,

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(3)}} (a^{(2)})^T$$

$$\frac{\partial b^{(2)}}{\partial J} = \frac{\partial z^{(3)}}{\partial J} \frac{\partial b^{(2)}}{\partial z^{(3)}} = \frac{\partial z^{(3)}}{\partial J}$$

$$\frac{\partial a^{(2)}}{\partial J} = \frac{\partial z^{(3)}}{\partial J} \frac{\partial a^{(2)}}{\partial z^{(3)}} = (W^{(2)})^T \frac{\partial J}{\partial z^{(3)}}$$

$$\frac{\partial z^{(2)}}{\partial J} = \frac{\partial a^{(2)}}{\partial J} \frac{\partial z^{(2)}}{\partial a^{(2)}} = ((W^{(2)})^T \frac{\partial J}{\partial z^{(3)}}) \circ f'(z^{(2)})$$



Key observation:

To calculate the derivatives at Layer 2, we only need the information of $\frac{\partial J}{\partial z^{(3)}}$.

Backpropagation

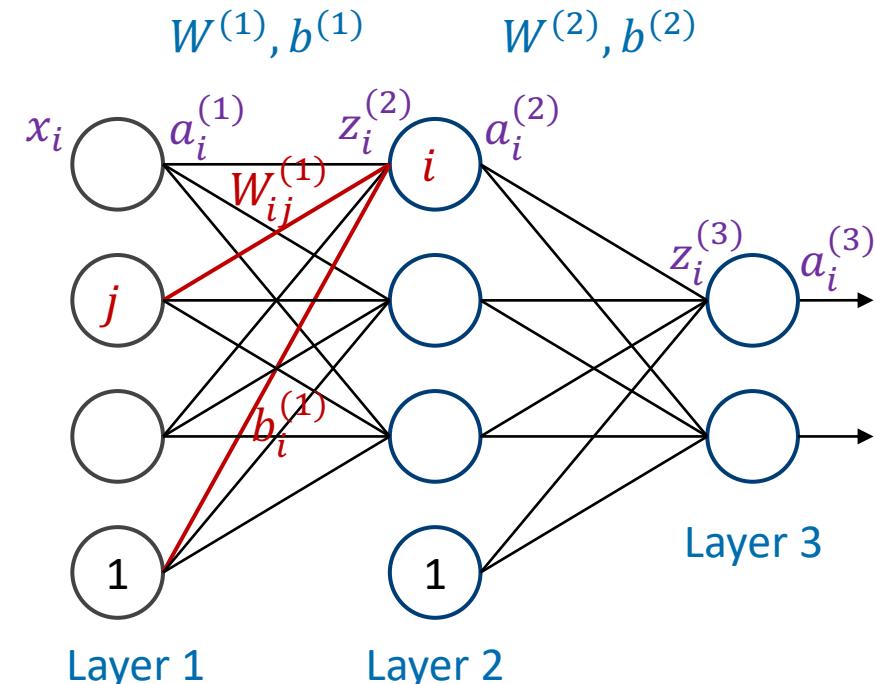
- Similarly, for Layer 1,

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial W^{(1)}} = \frac{\partial J}{\partial z^{(2)}} (a^{(1)})^T$$

$$\frac{\partial b^{(1)}}{\partial J} = \frac{\partial z^{(2)}}{\partial J} \frac{\partial b^{(1)}}{\partial z^{(2)}} = \frac{\partial z^{(2)}}{\partial J}$$

$$\frac{\partial a^{(1)}}{\partial J} = \frac{\partial z^{(2)}}{\partial J} \frac{\partial a^{(1)}}{\partial z^{(2)}} = (W^{(1)})^T \frac{\partial J}{\partial z^{(2)}}$$

$$\frac{\partial z^{(1)}}{\partial J} = \frac{\partial a^{(1)}}{\partial J} \frac{\partial z^{(1)}}{\partial a^{(1)}} = ((W^{(1)})^T \frac{\partial J}{\partial z^{(2)}}) \circ f'(z^{(1)})$$



- In general, to calculate the derivatives at Layer l , we only need the derivatives at Layer $l + 1$, in particular $\frac{\partial J}{\partial z^{(l+1)}}$.
- This enables us to calculate the gradient layer by layer in the backward direction, known as **backpropagation**.

Backpropagation algorithm

- Perform forward propagation, calculate neuron inputs $z^{(l)}$, activations $a^{(l)}$.
- At the output layer, calculate the derivative $\frac{\partial J}{\partial z^{(L)}}$,

$$\frac{\partial J}{\partial z^{(L)}} = (a^{(L)} - y) \circ f'(z^{(L)})$$

- For layer $l = L - 1, L - 2, \dots, 1$, calculate the propagated derivatives,

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}} (a^{(l)})^T$$

$$\frac{\partial J}{\partial b^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}}$$

$$\frac{\partial J}{\partial z^{(l)}} = ((W^{(l)})^T \frac{\partial J}{\partial z^{(l+1)}}) \circ f'(z^{(l)})$$

Gradient descent

- Now with the gradient $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$, we can perform gradient descent

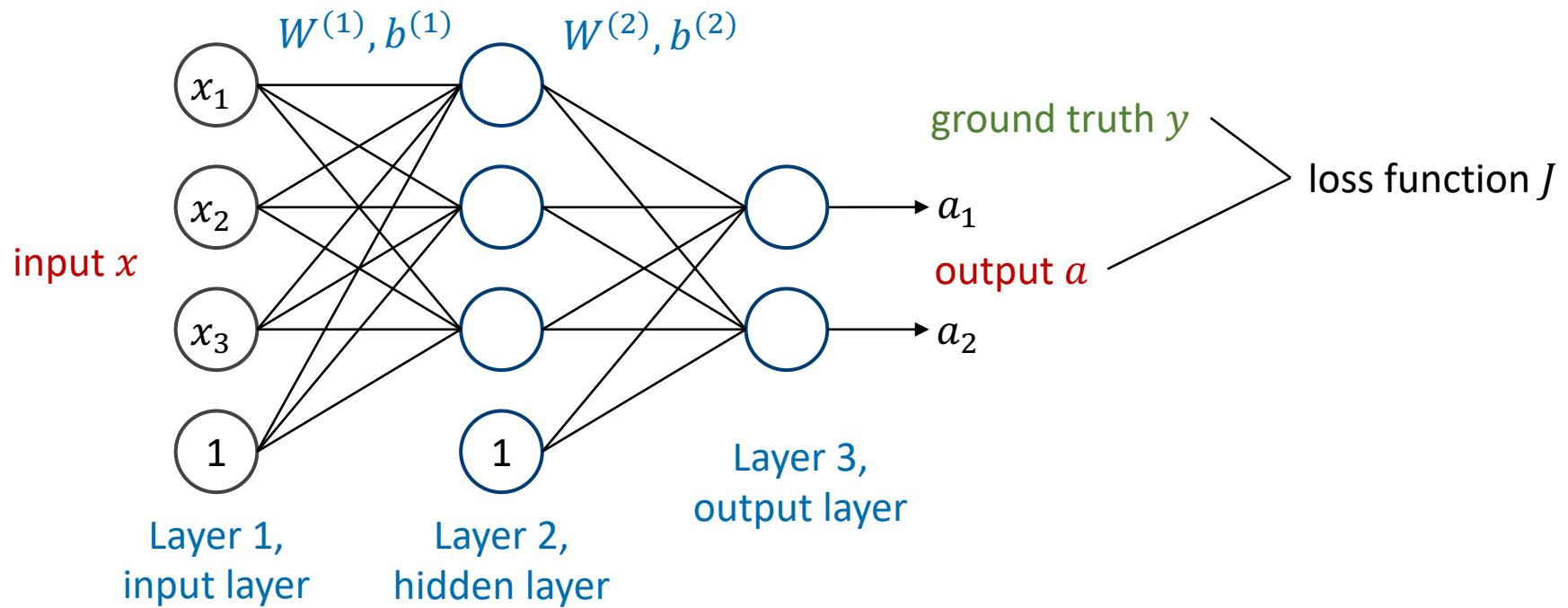
$$W = W - \alpha \frac{\partial J}{\partial W}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

- The network parameters W and b can be optimised after some iterations. The second question is also solved.

Neural networks

- Now we know how a neural network works.
 - Given input x , we know how to calculate output a .
 - We also know to how optimise parameters W and b to minimise the loss function J .



Multi-layer perceptron (MLP), which is a fully connected multi-layer network.

References

- Ch. 6, Deep Feedforward Networks. Ian Goodfellow et al. Deep Learning (<https://www.deeplearningbook.org/>).

Image Classification II

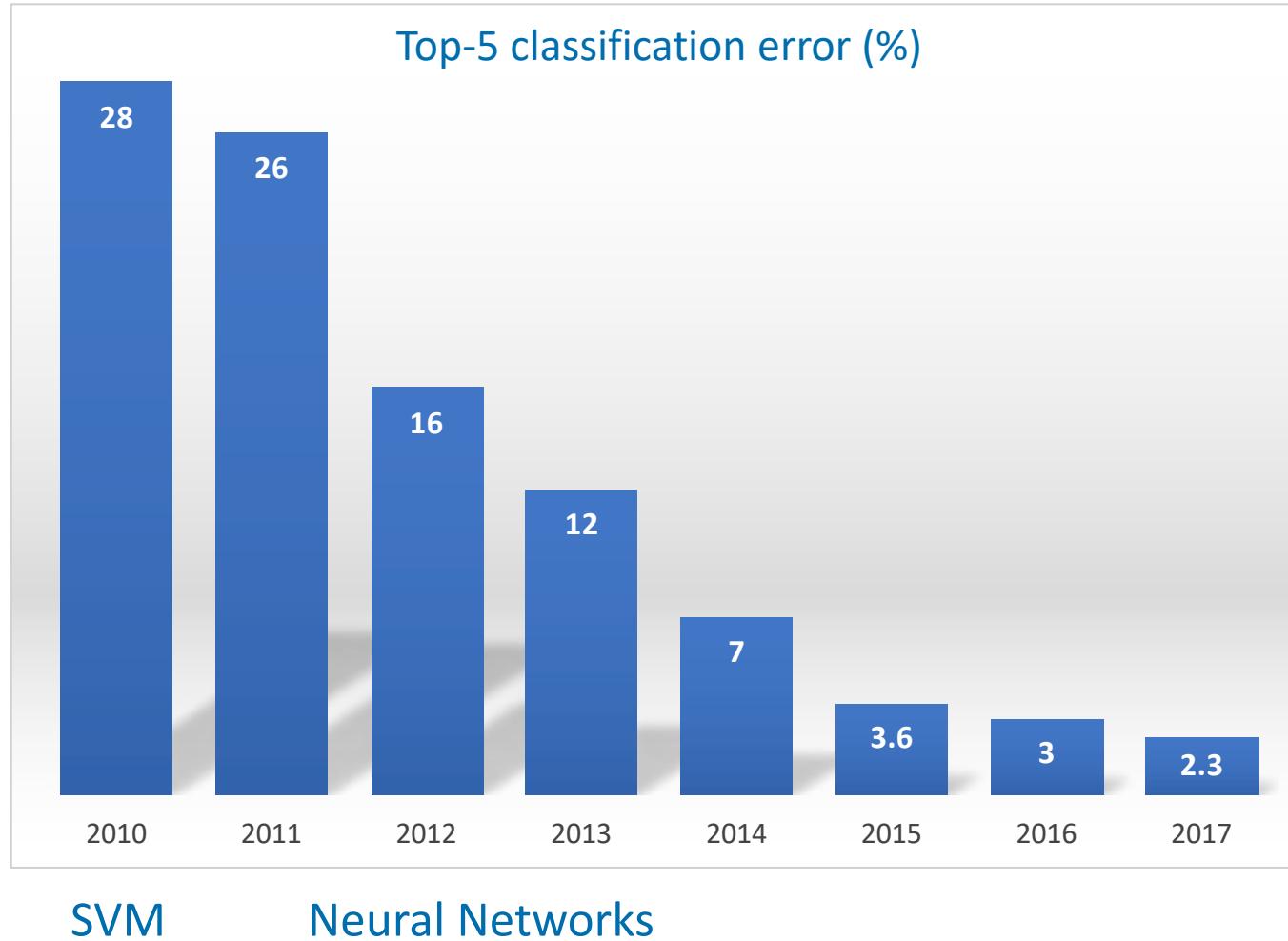
Dr Wenjia Bai

Department of Computing & Brain Sciences

Image classification

- Previously, we described K nearest neighbours (KNN) for image classification.
 - KNN is simple to implement but computationally expensive during test time.
- Today, we are going to introduce several other machine learning models for image classification.
 - Support vector machine (SVM)
 - Neural networks

ImageNet classification challenge



ILSVRC 2010

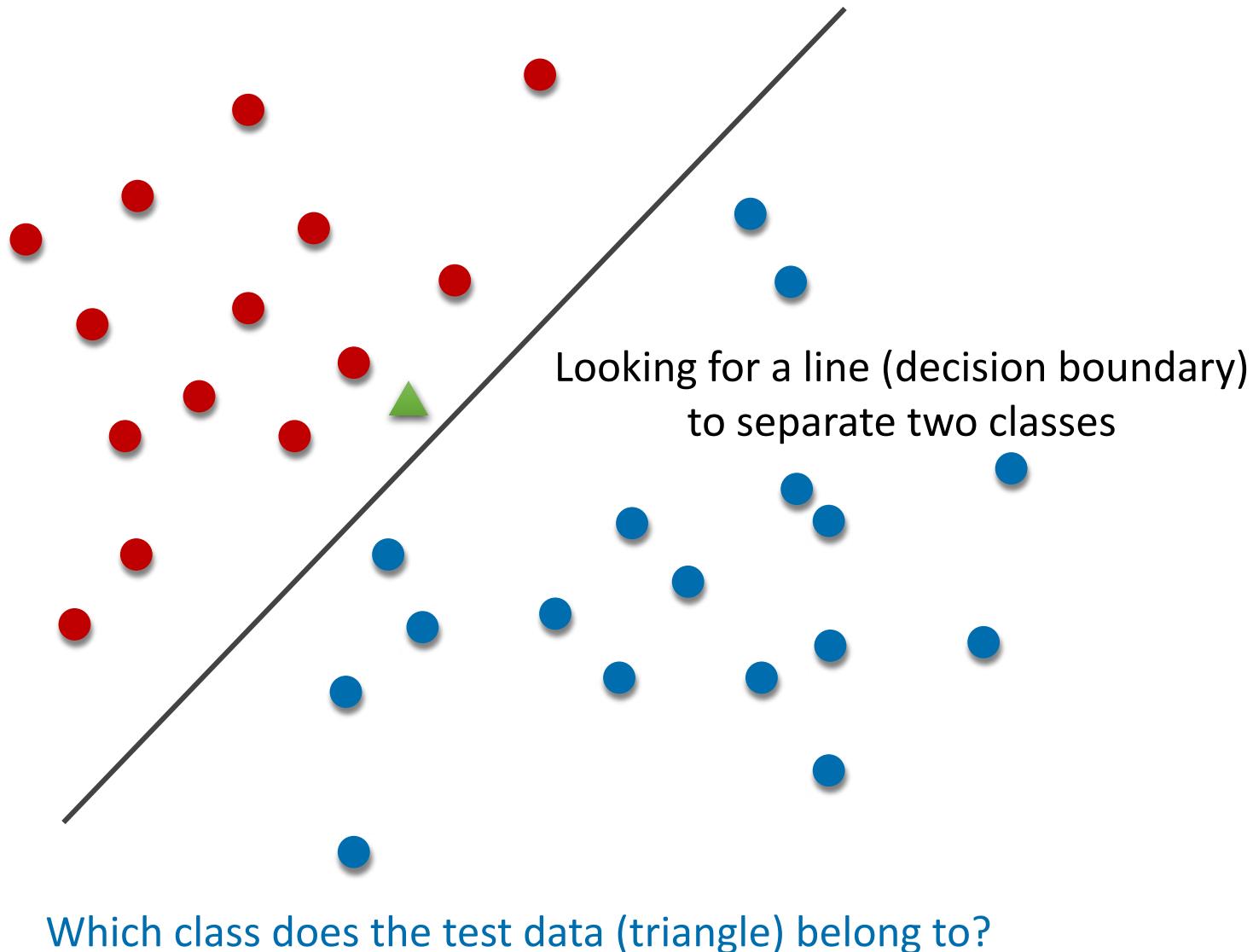
Codename	CLS	Institutions	Contributors and references
Hminmax	54.4	Massachusetts Institute of Technology	Jim Mutch, Sharat Chikkerur, Hristo Paskov, Ruslan Salakhutdinov, Stan Bileschi, Hueihan Jhuang
IBM	70.1	IBM research [†] , Georgia Tech [†]	Lexing Xie [†] , Hua Ouyang [†] , Apostol Natsev [†]
ISIL	44.6	Intelligent Systems and Informatics Lab., The University of Tokyo	Tatsuya Harada, Hideki Nakayama, Yoshitaka Ushiku, Yuya Yamashita, Jun Imura, Yasuo Kuniyoshi
ITNLP	78.7	Harbin Institute of Technology	Deyuan Zhang, Wenfeng Xuan, Xiaolong Wang, Bingquan Liu, Chengjie Sun
LIG	60.7	Laboratoire d'Informatique de Grenoble	Georges Quénot
NEC	28.2	NEC Labs America [†] , University of Illinois at Urbana-Champaign [‡] , Rutgers [‡]	Yuanqing Lin [†] , Fengjun Lv [†] , Shenghuo Zhu [†] , Ming Yang [†] , Timothee Cour [†] , Kai Yu [†] , LiangLiang Cao [†] , Zhen Li [‡] , Min-Hsuan Tsai [‡] , Xi Zhou [‡] , Thomas Huang [‡] , Tong Zhang [‡] (Lin et al., 2011)
NII	74.2	National Institute of Informatics, Tokyo, Japan [†] , Hefei Normal Univ. Hefei, China [‡]	Cai-Zhi Zhu [†] , Xiao Zhou [‡] , Shinichi Satoh [†]
NTU	58.3	CeMNet, SCE, NTU, Singapore	Zhengxiang Wang, Liang-Tien Chia
Regularities	75.1	SRI International	Omid Madani, Brian Burns
UCI	46.6	University of California Irvine	Hamed Pirsiavash, Deva Ramanan, Charless Fowlkes
XRCE	33.6	Xerox Research Centre Europe	Jorge Sanchez, Florent Perronnin, Thomas Mensink (Perronnin et al., 2010)

Year 2010: Image classification error rates in the first ImageNet challenge.
 The top team and many other teams use SVM as the classifier and HOG as the feature.

Support vector machine (SVM)

- What is SVM?
 - In its simplest form (linear SVM), it is simply a line that separates two different classes.

Linear classifier



Linear classifier

- For 2D case (input data is 2D), the line has the form

$$w_1x_1 + w_2x_2 + b = 0$$

- The rule of the linear classifier is to assign a class c to data x ,

$$c = \begin{cases} +1, & \text{if } w_1x_1 + w_2x_2 + b \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- For KNN, we need to store all the training data.
- For linear classifier, once we know w and b , we can discard training data.

Linear classifier

- For general cases, the linear model is formulated as,

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

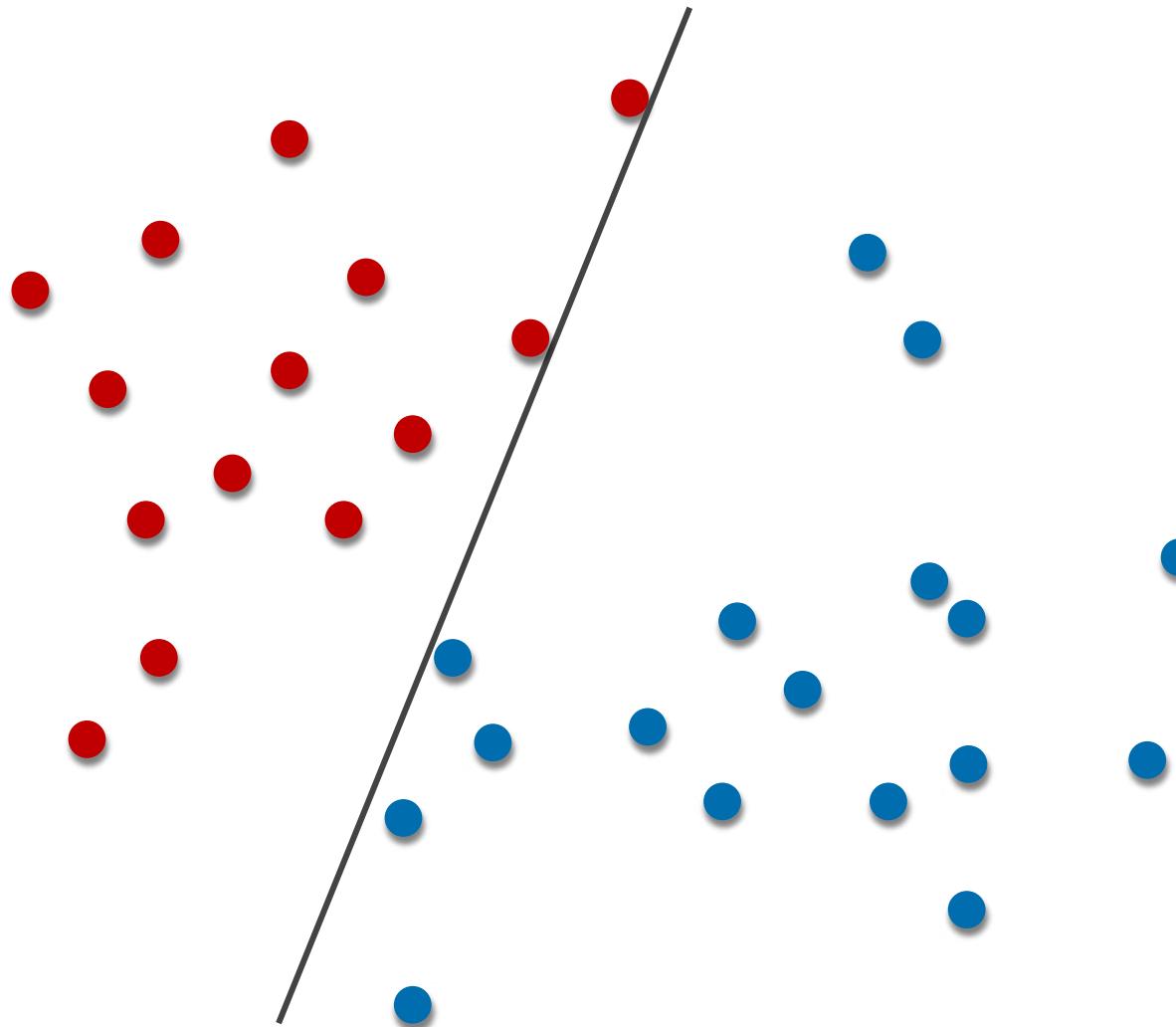
weights, | |
the normal data, bias
 feature

- The rule of the linear classifier is to assign a class c to data x ,

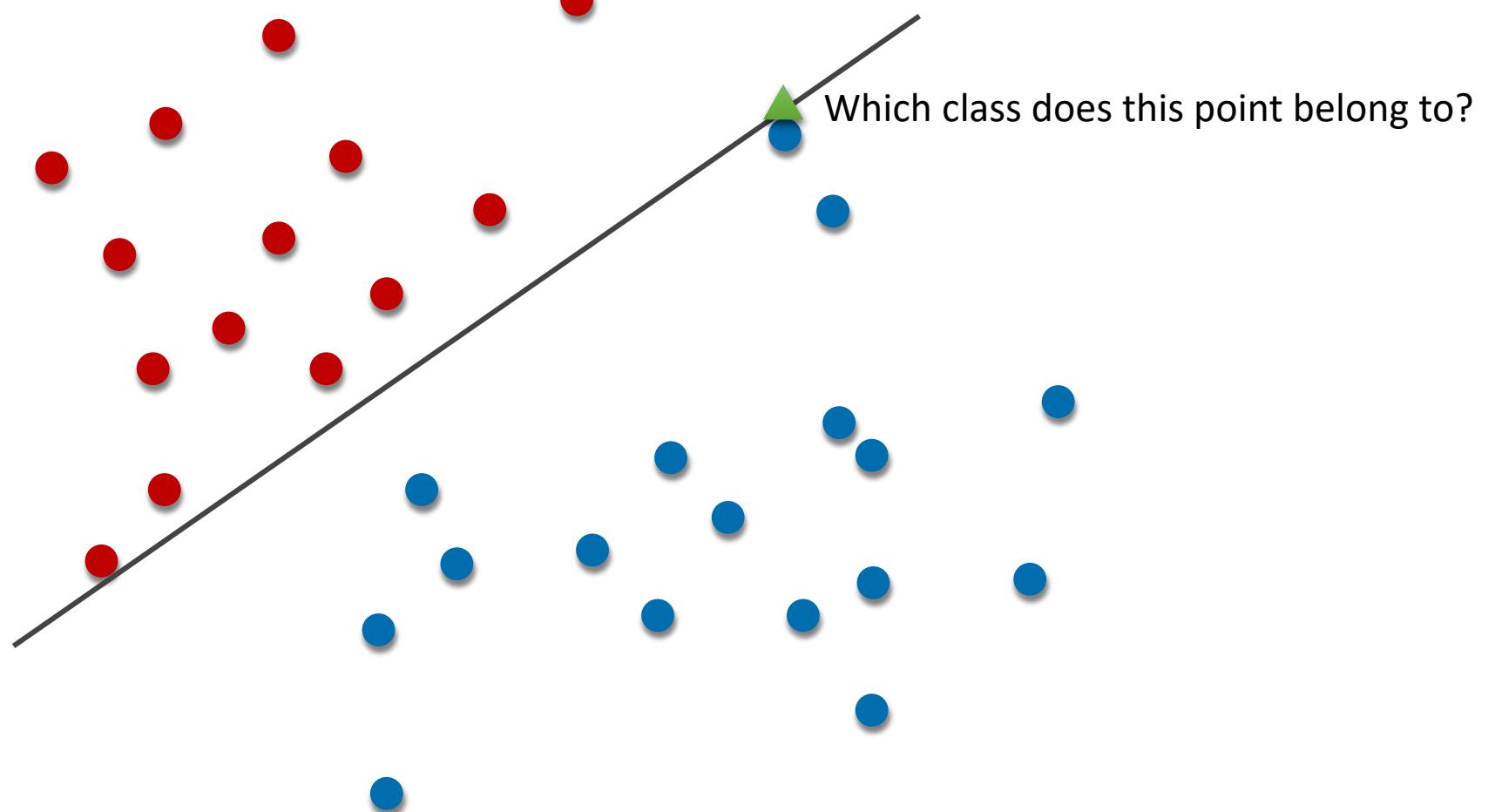
$$c = \begin{cases} +1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- To train the classifier, what we need is to estimate parameters \mathbf{w} and b , which determines the decision boundary or hyperplane.

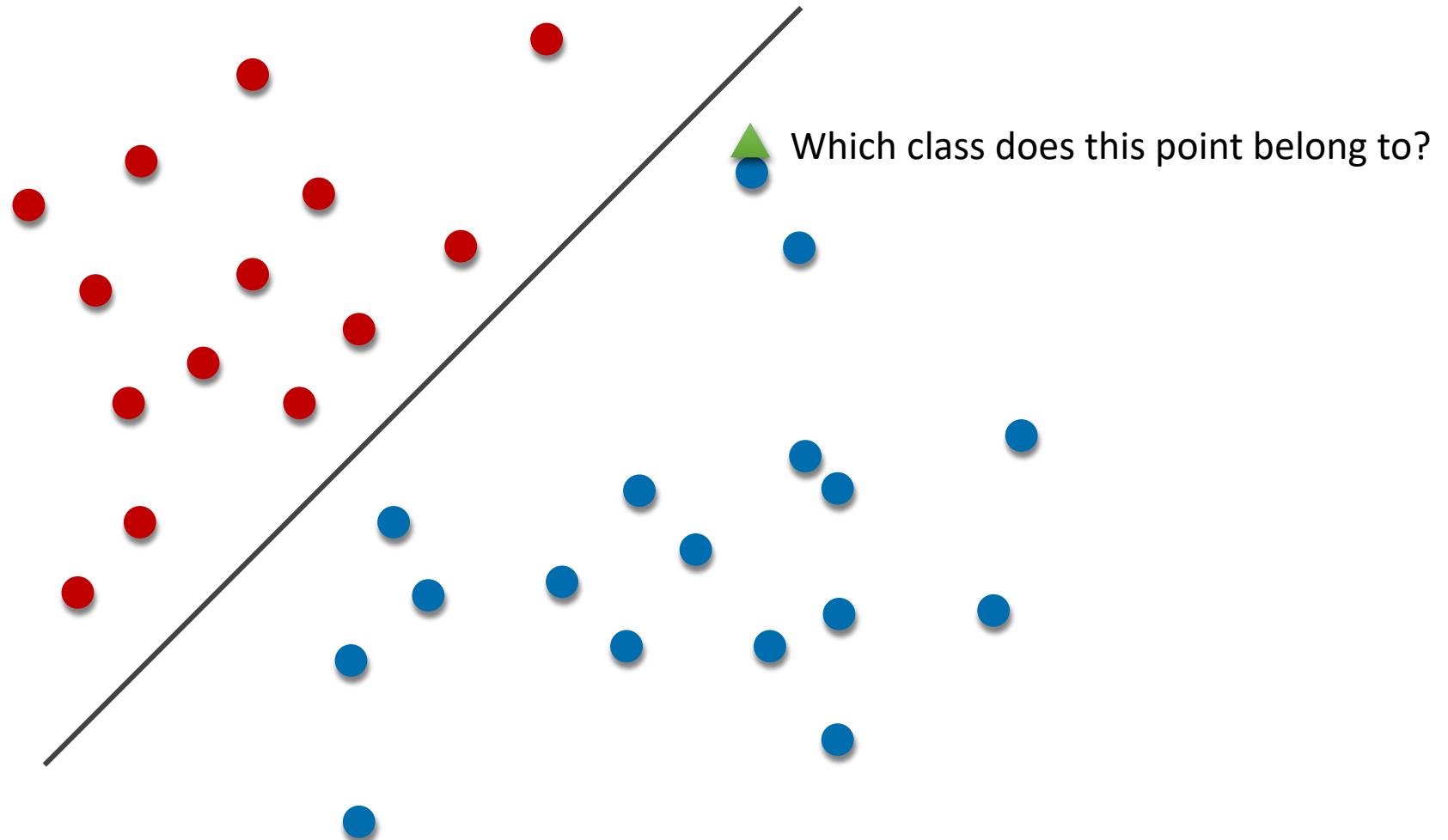
Where is best separating hyperplane?



Where is best separating hyperplane?

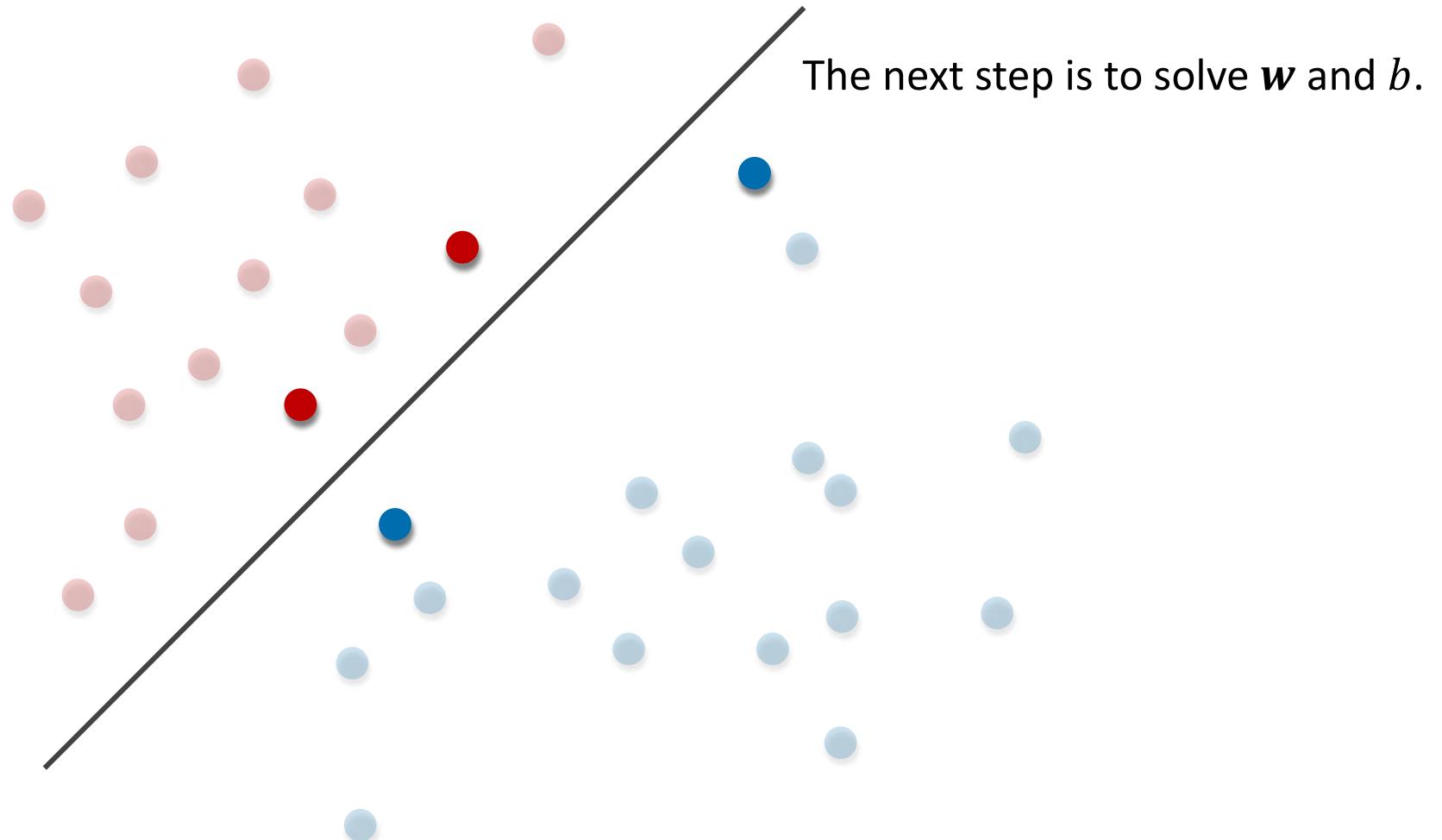


Where is best separating hyperplane?

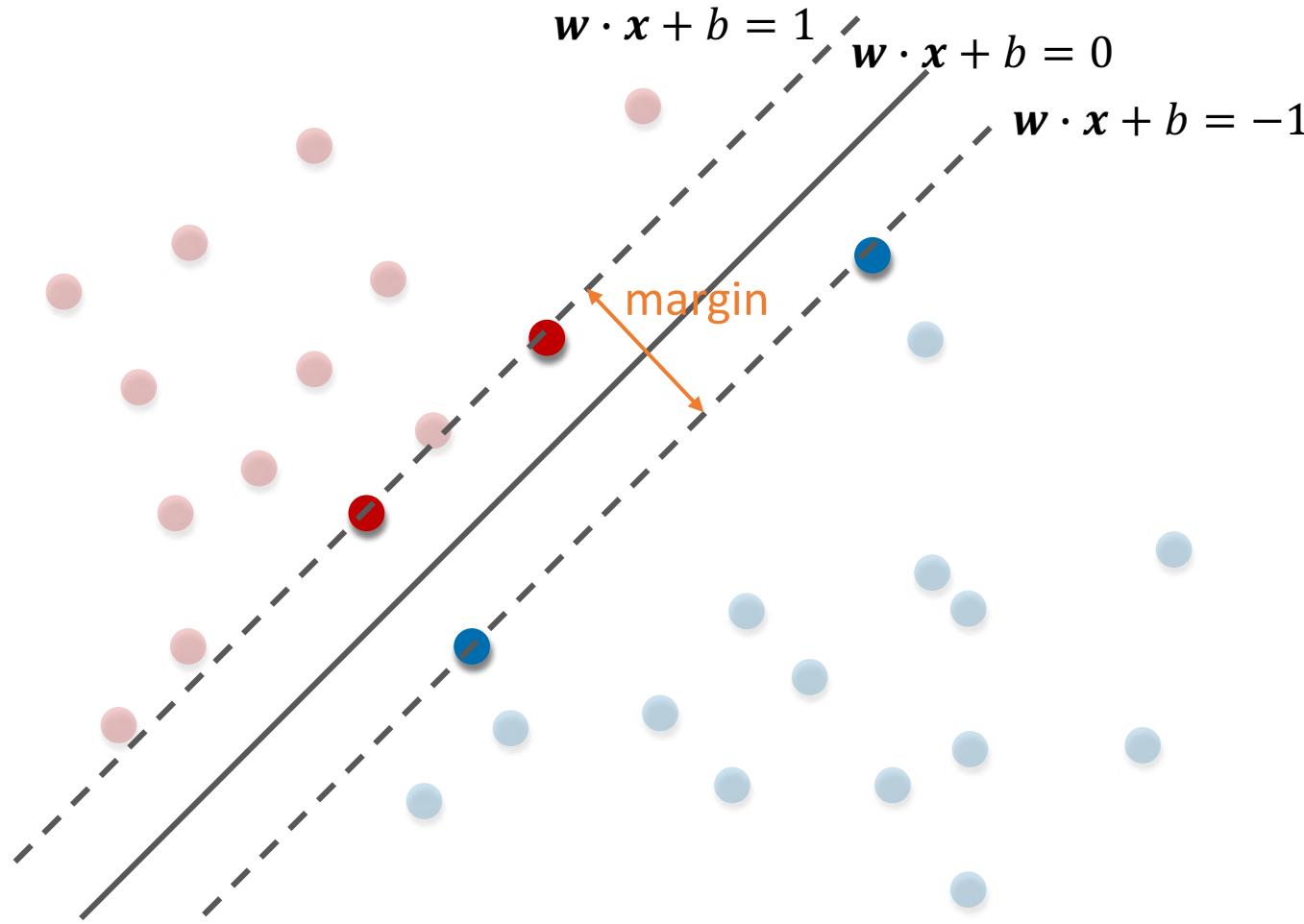


Intuitively, a line that are far from both classes of points (**maximum margin** hyperplane).

Where is best separating hyperplane?



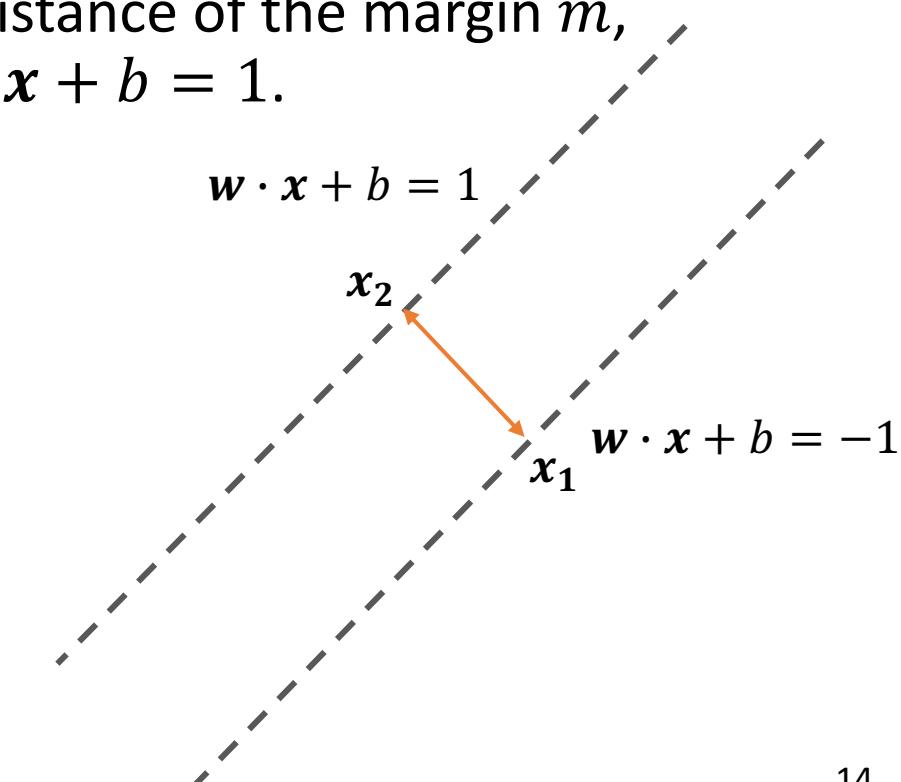
Where is best separating hyperplane?



- We would like the support vectors to fulfil the equations $w \cdot x + b = 1$ or -1 , so that all the other points are easily classified.
- We would also like to maximise the margin between the dashed lines.

Maximum margin

- What is the margin between $w \cdot x + b = 1$ and $w \cdot x + b = -1$?
- We can derive it like this.
 - Let x_1 be a point on the plane $w \cdot x + b = -1$, we have $w \cdot x_1 + b = -1$.
 - Let us move x_1 along the normal direction n for the distance of the margin m , it should arrive at the point x_2 on the other plane $w \cdot x + b = 1$.
 - We have $w \cdot (x_1 + mn) + b = 1$.
 - It follows that $mw \cdot n = 2$.
 - We know that the normal $n = \frac{w}{\|w\|}$.
 - As a result, the margin $m = \frac{2}{\|w\|}$.



Support vector machine (SVM)

- SVM aims to maximise the margin,

$$\max_{w,b} \frac{2}{\|w\|} \quad \text{maximise the margin}$$

subject to $w \cdot x_i + b \geq +1, \text{ if } y_i = +1$
 $\leq -1, \text{ if } y_i = -1$, for $i = 1, 2, \dots, N$.

separate the two classes

Optimisation

- It can be re-formulated as this optimisation problem

$$\min_{\mathbf{w}, b} \|\mathbf{w}\|^2$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$, for $i = 1, 2, \dots, N$.

- This is a quadratic optimisation problem subject to linear constraints.

Optimisation

- The following optimisation problem can be further re-formulated as,

$$\min_{\mathbf{w}, b} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$

- The second term is called the hinge loss $h = \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$.
- How to optimise this loss function?

Gradient descent

- Optimise the following loss function $L(\mathbf{w}, b)$,

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b) = \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$

- The gradient of the loss function L w.r.t. \mathbf{w} is,

$$\nabla_{\mathbf{w}} L = 2\mathbf{w} + C \sum_{i=1}^N \nabla_{\mathbf{w}} h$$

where $h = \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$. Its gradient is,

$$\nabla_{\mathbf{w}} h = \begin{cases} -y_i \mathbf{x}_i, & \text{if } y_i (\mathbf{w} \cdot \mathbf{x}_i + b) < 1 \\ 0, & \text{otherwise} \end{cases}$$

- Similarly, we can derive $\nabla_b L$.

Gradient descent

- We can optimise the loss function using gradient descent.
- For each iteration

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{(k)}, b^{(k)}) \\ &= \mathbf{w}^{(k)} - \eta (2\mathbf{w}^{(k)} + C \sum_{i=1}^N \nabla_{\mathbf{w}} h)\end{aligned}$$

↓

step length, learning rate

SVM for image classification

- Classifier

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

- What is the feature x ?
- How to perform gradient descent for millions of images?

Large-scale Image Classification: Fast Feature Extraction and SVM Training

Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour and Kai Yu
NEC Laboratories America, Cupertino, CA 95014

Liangliang Cao and Thomas Huang
Beckman Institute, University of Illinois at Urbana-Champaign, IL 61801

Abstract

Most research efforts on image classification so far have been focused on medium-scale datasets, which are often defined as datasets that can fit into the memory of a desktop (typically 4G–48G). There are two main reasons for the limited efforts on large-scale image classification. First, until the emergence of ImageNet dataset, there was almost no publicly available large-scale benchmark data for image classification. This is mostly because class labels are expensive to obtain. Second, large-scale classification is hard because it poses more challenges than its medium-scale counterparts. A key challenge is how to achieve efficiency in both feature extraction and classifier training without compromising performance. This paper is to show how we address this challenge using ImageNet dataset as an example. For feature extraction, we develop a Hadoop scheme that performs feature extraction in parallel using hundreds of mappers. This allows us to extract fairly sophisticated features (with dimensions being hundreds of thousands) on 1.2 million images within one day. For SVM training, we develop a parallel averaging stochastic gradient descent (ASGD) algorithm for training one-against-all 1000-class SVM classifiers. The ASGD algorithm is capable of dealing with terabytes of training data and converges very fast – typically 5 epochs are sufficient. As a result, we achieve state-of-the-art performance on the ImageNet 1000-class classification, i.e., 52.9% in classification accuracy and 71.8% in top 5 hit rate.

1. Introduction

It is needless to say how important of image classification/recognition is in the field of computer vision – image recognition is essential for bridging the huge semantic gap between an image, which is simply a scatter

of pixels in untrained computers, and the object it presents. Therefore, there have been extensive research efforts on developing effective visual object recognizers [10]. Along the line, there are quite a few benchmark datasets for image classification, such as MNIST [1], Caltech 101 [9], Caltech 256 [11], PASCAL VOC [7], LabelMe [19], etc. Researchers have developed a wide spectrum of different local descriptors [17, 16, 5, 22], bag-of-words models [14, 24] and classification methods [4], and they compared to the best available results on those publicly available datasets – for PASCAL VOC, many teams from all over the world participate in the PASCAL Challenge each year to compete for the best performance. Such benchmarking activities have played an important role in pushing object classification research forward in the past years.

In recent years, there is a growing consensus that it is necessary to build general purpose object recognizers that are able to recognize many different classes of objects – e.g. this can be very useful for image/video tagging and retrieval. Caltech 101/256 are the pioneer benchmark datasets on that front. Newly released ImageNet dataset [6] goes a big step further, as shown in Fig. 1 – it further increases the number of classes to 1000¹, and it has more than 1000 images for each class on average. Indeed, it is necessary to have so many images for each class to cover visual variance, such as lighting, orientation as well as fairly wild appearance difference within the same class – like different cars may look very differently although all belong to the same class.

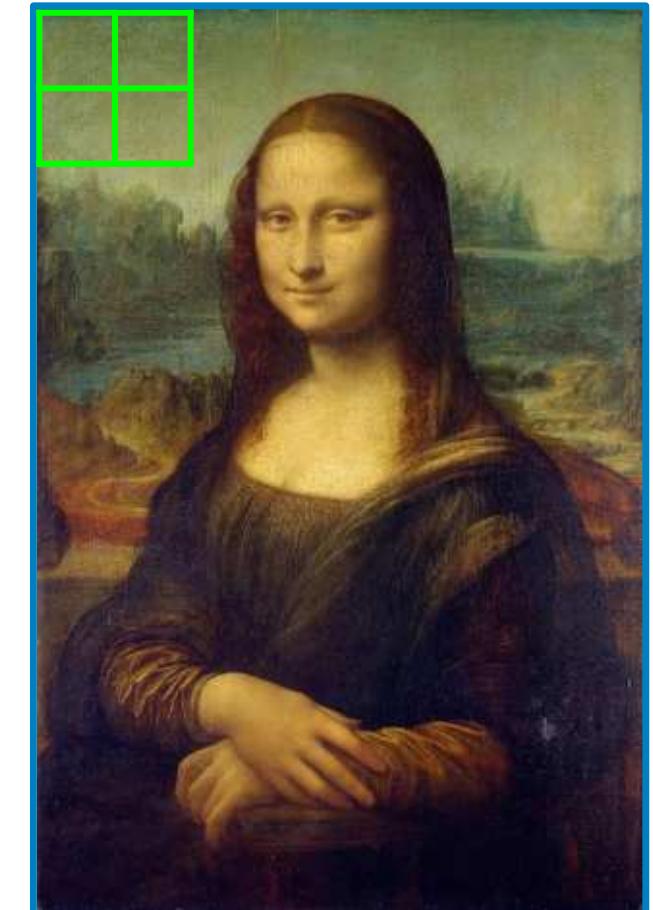
However, compared to those previous medium-scale datasets (such as PASCAL VOC datasets and Caltech 101&256, which can fit into desktop memory), large-scale ImageNet dataset poses more challenges in image classification. For example, those previous datasets

¹The overall ImageNet dataset consists of 11,231,732 labeled images of 1559 classes by October 2010. But here we only concern about the subset of ImageNet dataset (about 1.2 million images) that was used in 2010 ImageNet Large Scale Visual Recognition Challenge.

Describe image feature using HOG

- HoG divides the image into equally spaced cells.
 - 4 cells form a block.
 - Each block is described the gradient orientation histograms of the 4 cells.
- The HOG descriptor is formed by concatenating the normalised local descriptors for all blocks.
- We can use the HOG descriptor as x .

Calculate gradient orientation histograms for this block.



128x64 pixel image.

Gradient descent

N is over a million!

- Optimise the following loss function $L(\mathbf{w}, b)$,

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b) = \|\mathbf{w}\|^2 + C \sum_{i=1}^N \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$

- For each iteration

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{(k)}, b^{(k)}) \\ b^{(k+1)} &= b^{(k)} - \eta \nabla_b L(\mathbf{w}^{(k)}, b^{(k)})\end{aligned}$$

Stochastic gradient descent (SGD)

- Since N is very big (a million), it would be computationally expensive to evaluate the gradient over the whole training set.
- Instead, we can perform stochastic gradient descent (SGD).
- For each iteration in optimisation
 - Randomly select a batch of B training samples
 - Calculate the gradient $\nabla_w L_B$ and $\nabla_b L_B$ only for this batch
 - Update the parameters

$$\begin{aligned}\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \eta \nabla_w L_B(\mathbf{w}^{(k)}, b^{(k)}) \\ b^{(k+1)} &= b^{(k)} - \eta \nabla_b L_B(\mathbf{w}^{(k)}, b^{(k)})\end{aligned}$$

L_B denotes the loss
just for this batch

SVM for image classification

- Classifier

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

- So now we know what the feature \mathbf{x} is and how \mathbf{w} and b are estimated.

- At test time, we perform classification,

$$c = \begin{cases} +1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- Next question: this only separates two classes.

Large-scale Image Classification: Fast Feature Extraction and SVM Training

Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour and Kai Yu
NEC Laboratories America, Cupertino, CA 95014

Liangliang Cao and Thomas Huang
Beckman Institute, University of Illinois at Urbana-Champaign, IL 61801

Abstract

Most research efforts on image classification so far have been focused on medium-scale datasets, which are often defined as datasets that can fit into the memory of a desktop (typically 4G–48G). There are two main reasons for the limited efforts on large-scale image classification. First, until the emergence of ImageNet dataset, there was almost no publicly available large-scale benchmark data for image classification. This is mostly because class labels are expensive to obtain. Second, large-scale classification is hard because it poses more challenges than its medium-scale counterparts. A key challenge is how to achieve efficiency in both feature extraction and classifier training without compromising performance. This paper is to show how we address this challenge using ImageNet dataset as an example. For feature extraction, we develop a Hadoop scheme that performs feature extraction in parallel using hundreds of mappers. This allows us to extract fairly sophisticated features (with dimensions being hundreds of thousands) on 1.2 million images within one day. For SVM training, we develop a parallel averaging stochastic gradient descent (ASGD) algorithm for training one-against-all 1000-class SVM classifiers. The ASGD algorithm is capable of dealing with terabytes of training data and converges very fast – typically 5 epochs are sufficient. As a result, we achieve state-of-the-art performance on the ImageNet 1000-class classification, i.e., 52.9% in classification accuracy and 71.8% in top 5 hit rate.

1. Introduction

It is needless to say how important of image classification/recognition is in the field of computer vision – image recognition is essential for bridging the huge semantic gap between an image, which is simply a scatter

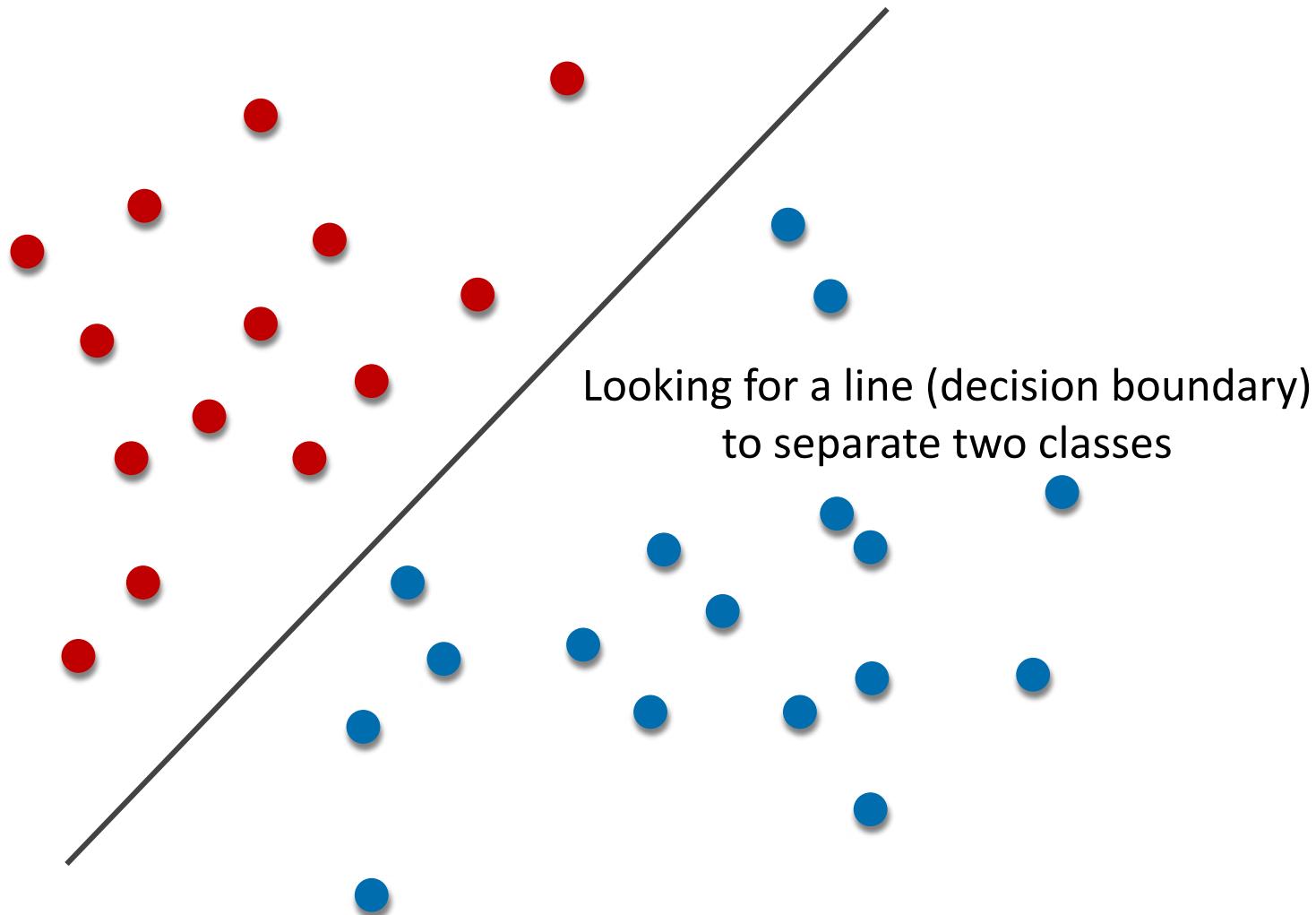
of pixels in untrained computers, and the object it presents. Therefore, there have been extensive research efforts on developing effective visual object recognizers [10]. Along the line, there are quite a few benchmark datasets for image classification, such as MNIST [1], Caltech 101 [9], Caltech 256 [11], PASCAL VOC [7], LabelMe [19], etc. Researchers have developed a wide spectrum of different local descriptors [17, 16, 5, 23], bag-of-words models [14, 24] and classification methods [4], and they compared to the best available results on those publicly available datasets – for PASCAL VOC, many teams from all over the world participate in the PASCAL Challenge each year to compete for the best performance. Such benchmarking activities have played an important role in pushing object classification research forward in the past years.

In recent years, there is a growing consensus that it is necessary to build general purpose object recognizers that are able to recognize many different classes of objects – e.g. this can be very useful for image/video tagging and retrieval. Caltech 101/256 are the pioneer benchmark datasets on that front. Newly released ImageNet dataset [6] goes a big step further, as shown in Fig. 1 – it further increases the number of classes to 1000¹, and it has more than 1000 images for each class on average. Indeed, it is necessary to have so many images for each class to cover visual variance, such as lighting, orientation as well as fairly wild appearance difference within the same class – like different cars may look very differently although all belong to the same class.

However, compared to those previous medium-scale datasets (such as PASCAL VOC datasets and Caltech 101&256, which can fit into desktop memory), large-scale ImageNet dataset poses more challenges in image classification. For example, those previous datasets

¹The overall ImageNet dataset consists of 11,231,732 labeled images of 1589 classes by October 2010. But here we only concern about the subset of ImageNet dataset (about 1.2 million images) that was used in 2010 ImageNet Large Scale Visual Recognition Challenge.

Linear classifier



ImageNet

- There are 1,000 classes in the ImageNet classification challenge.



<http://www.image-net.org>

How to perform multi-class classification?

- One vs rest strategy
 - Train a classifier for each of the 1000 classes.
 - A classifier between class 1 and others
 - A classifier between class 2 and others
 - A classifier between class 3 and others
 - ...
 - During testing, apply all the 1000 classifiers to the test data.
 - The classifier which produces the highest response will determine the result.

$$c = \operatorname{argmax}_{k=1,2,\dots,K} f_k(\mathbf{x})$$

where $f_k(\mathbf{x}) = \mathbf{w}_k \cdot \mathbf{x} + b_k$ denotes the k-th classifier.

How to perform multi-class classification?

- One vs one strategy
 - Train a classifier between each pair of classes.
 - A classifier between class 1 and class 2
 - A classifier between class 1 and class 3
 - A classifier between class 1 and class 4
 - ...
 - A classifier between class 2 and class 3
 - A classifier between class 2 and class 4
 - ...
 - For K classes, we need $\frac{K(K-1)}{2}$ classifiers.
 - At test time, each classifier will vote for a class.
 - Count the vote for each class and perform majority voting.

SVM for image classification

- In this paper, 1,000 binary classifiers were trained.
- It was implemented using parallel computing as the training of 1,000 classifiers were independent.

Large-scale Image Classification: Fast Feature Extraction and SVM Training

Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour and Kai Yu
NEC Laboratories America, Cupertino, CA 95014

Liangliang Cao and Thomas Huang
Beckman Institute, University of Illinois at Urbana-Champaign, IL 61801

Abstract

Most research efforts on image classification so have been focused on medium-scale datasets, which are often defined as datasets that can fit into the memory of a desktop (typically 4G–48G). There are two main reasons for the limited efforts on large-scale image classification. First, until the emergence of ImageNet dataset, there was almost no publicly available large-scale benchmark data for image classification. This is mostly because class labels are expensive to obtain. Second, large-scale classification is hard because it poses more challenges than its medium-scale counterparts. A key challenge is how to achieve efficiency in both feature extraction and classifier training without compromising performance. This paper is to show how we address this challenge using ImageNet dataset as an example. For feature extraction, we develop a Hadoop scheme that performs feature extraction in parallel using hundreds of mappers. This allows us to extract fairly sophisticated features (with dimensions being hundreds of thousands) on 1.2 million images within one day. For SVM training, we develop a parallel averaging stochastic gradient descent (ASGD) algorithm for training one-against-all 1000-class SVM classifiers. The ASGD algorithm is capable of dealing with terabytes of training data and converges very fast – typically 5 epochs are sufficient. As a result, we achieve state-of-the-art performance on the ImageNet 1000-class classification, i.e., 52.9% in classification accuracy and 71.8% in top 5 hit rate.

1. Introduction

It is needless to say how important of image classification/recognition is in the field of computer vision – image recognition is essential for bridging the huge semantic gap between an image, which is simply a scatter

of pixels in untrained computers, and the object it presents. Therefore, there have been extensive research efforts on developing effective visual object recognizers [10]. Along the line, there are quite a few benchmark datasets for image classification, such as MNIST [1], Caltech 101 [9], Caltech 256 [11], PASCAL VOC [7], LabelMe [19], etc. Researchers have developed a wide spectrum of different local descriptors [17, 16, 5, 22], bag-of-words models [14, 24] and classification methods [4], and they compared to the best available results on those publicly available datasets – for PASCAL VOC, many teams from all over the world participate in the PASCAL Challenge each year to compete for the best performance. Such benchmarking activities have played an important role in pushing object classification research forward in the past years.

In recent years, there is a growing consensus that it is necessary to build general purpose object recognizers that are able to recognize many different classes of objects – e.g. this can be very useful for image/video tagging and retrieval. Caltech 101/256 are the pioneer benchmark datasets on that front. Newly released ImageNet dataset [6] goes a big step further, as shown in Fig. 1 – it further increases the number of classes to 1000¹, and it has more than 1000 images for each class on average. Indeed, it is necessary to have so many images for each class to cover visual variance, such as lighting, orientation as well as fairly wild appearance difference within the same class – like different cars may look very differently although all belong to the same class.

However, compared to those previous medium-scale datasets (such as PASCAL VOC datasets and Caltech 101&256, which can fit into desktop memory), large-scale ImageNet dataset poses more challenges in image classification. For example, those previous datasets

¹The overall ImageNet dataset consists of 11,231,732 labeled images of 15589 classes by October 2010. But here we only concern about the subset of ImageNet dataset (about 1.2 million images) that was used in 2010 ImageNet Large Scale Visual Recognition Challenge.

ImageNet classification challenge

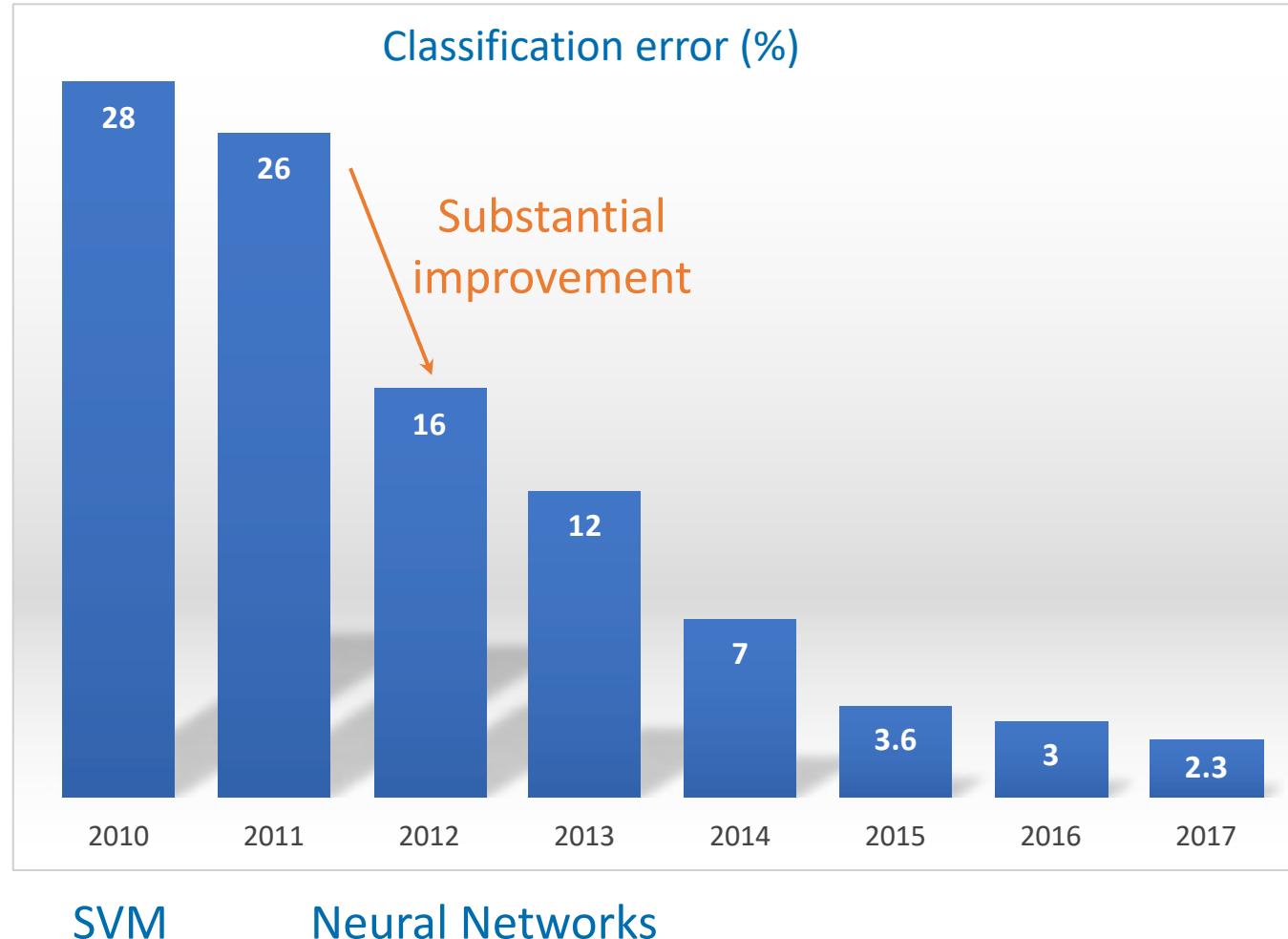


Image classification

- SVM won the ImageNet challenge in 2010.
- Something happened afterwards.
 - In 2011, a method based on deep belief networks (DBN) increased the speech recognition accuracy by a large margin, published in ICASSP [1].
 - In 2012, neural networks won the ImageNet challenge [2].

[1] G.E. Dahl et al. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. ICASSP 2011.

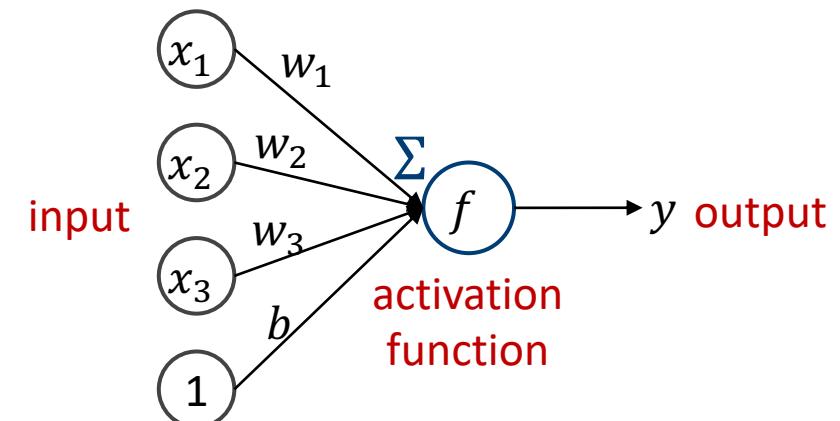
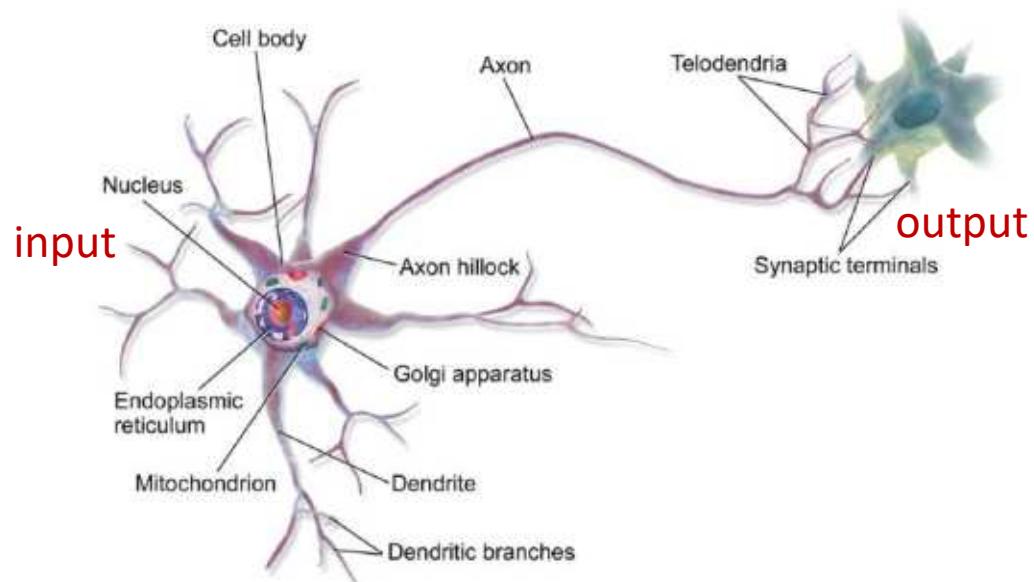
[2] A. Krizhevsky et al. ImageNet classification with deep convolutional neural networks. NIPS 2012.

Neural networks

- Perceptron
- Multi-layer perceptron (MLP)
- Convolutional neural networks (CNN)

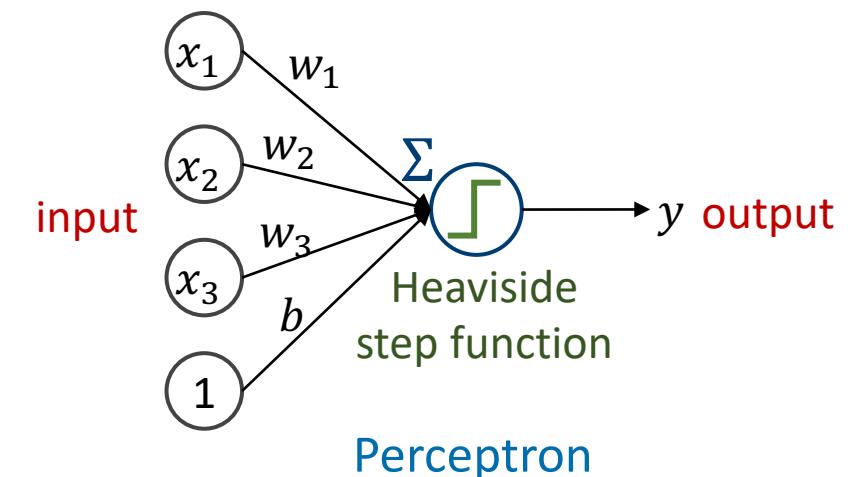
From biological neuron to artificial neuron

- **Biology**
 - Neurons are inter-connected.
 - Signal (electrical, chemical) flows from input at dendrites to output at axon terminals.
- **Artificial neural networks**
 - Inspired by biology.
 - But it is not the exact model of how brain or neuron works.



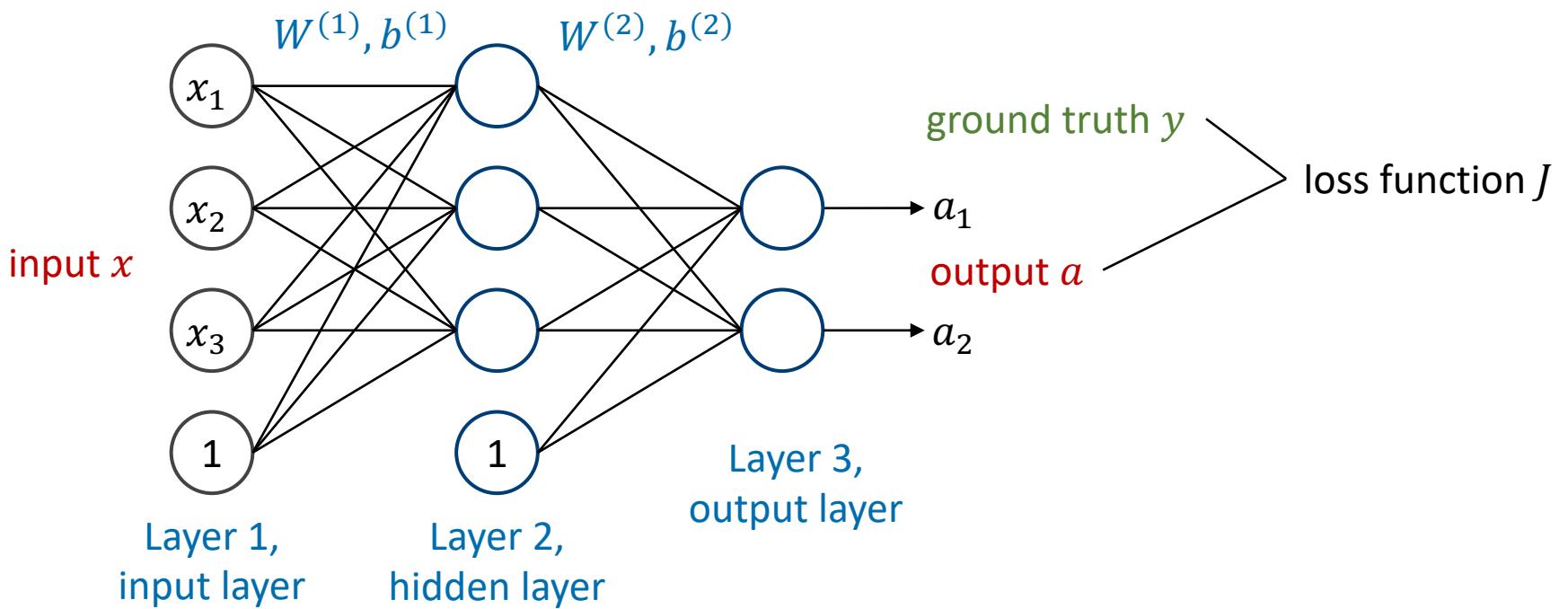
Perceptron

- The simplest form, perceptron, was developed by Frank Rosenblatt in 1957.
 - Perceptron consists of only a single layer and uses the Heaviside step function as activation function.
- We optimise w and b so that y matches ground truth.
- Multi-layer perceptron was developed later.



Multi-layer perceptron

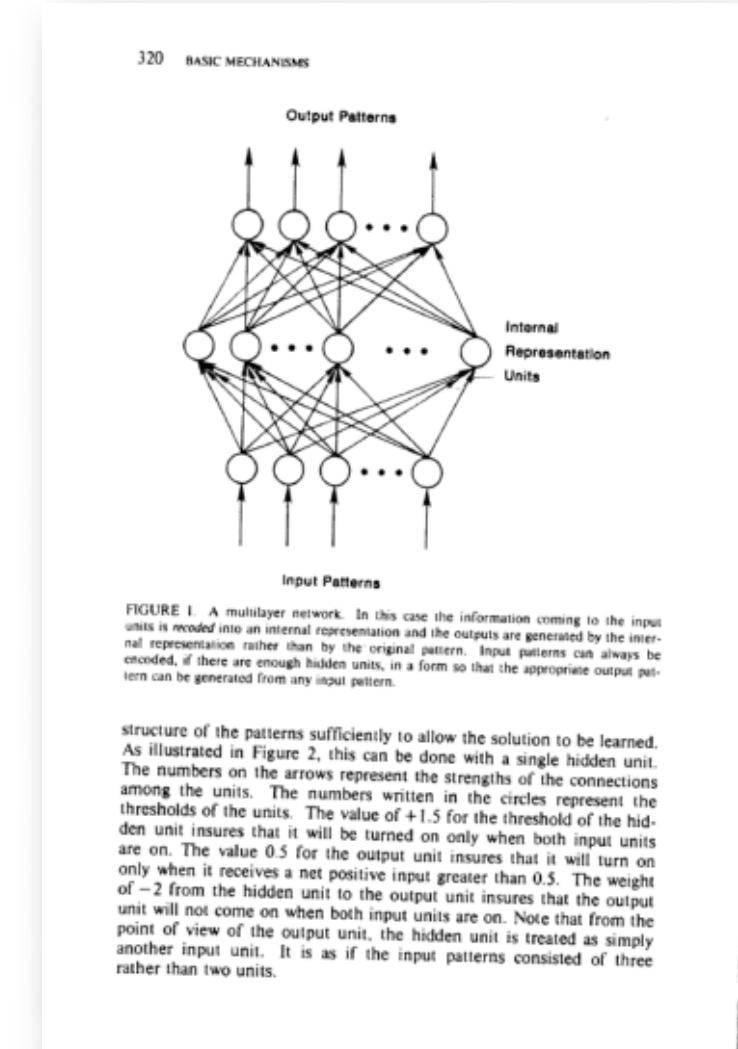
- A multi-layer perceptron (MLP) is formed by putting several layers of neurons into connection, where the output of a neuron can be the input to another.



Multi-layer perceptron (MLP), which is a fully connected multi-layer network.

Neural networks

- The form of neural networks has not changed much since 1980s.
- Its optimisation algorithm, backpropagation, developed by David Rumelhart, Geoffrey Hinton and Ronald J. Williams in 1986, has not changed either.



Neural networks

- So what has changed in the past 40 years?
 - More layers of neural connections (i.e. deeper)
 - Better hardware to enable faster computation (i.e. GPUs)
 - Larger datasets (e.g. ImageNet)
 - Other technical improvements
 - Activation functions, optimisation, data normalisation, data augmentation etc
- Why was it not popular in 1990s and 2000s?
 - Because people were using SVM.
 - Also because many people did not believe deep neural networks would work.

Neural networks



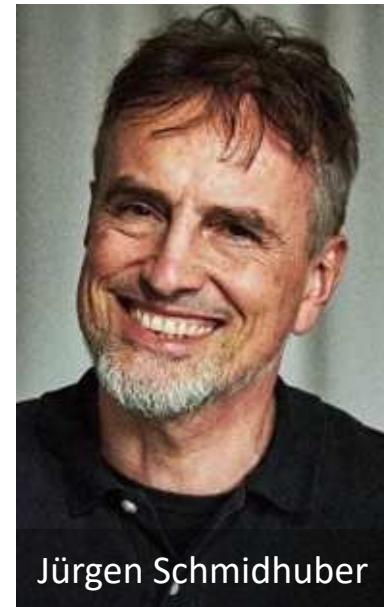
Yoshua Bengio



Geoffrey Hinton



Yann LeCun



Jürgen Schmidhuber

Turing Award 2018

“For conceptual and engineering breakthroughs that have made **deep neural networks** a critical component of computing.”

Contributions to neural networks,
including RNN and LSTM.

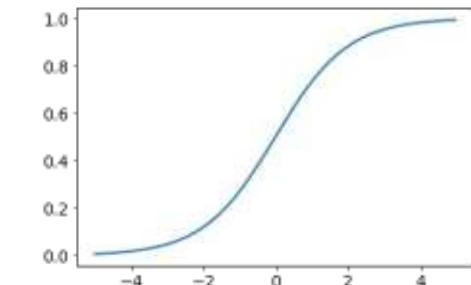
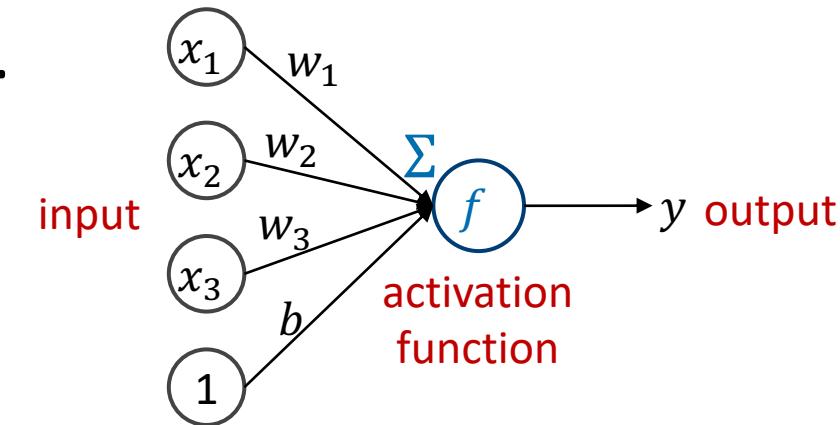
How does a neural network work?

- Let us start from the simplest case, a single neuron.
- The neuron is a computational unit that takes an input, applies an activation function and generates an output.

$$y = f\left(\sum_{i=1}^3 w_i x_i + b\right)$$

- A commonly used activation function is the sigmoid function, also known as the logistic function.

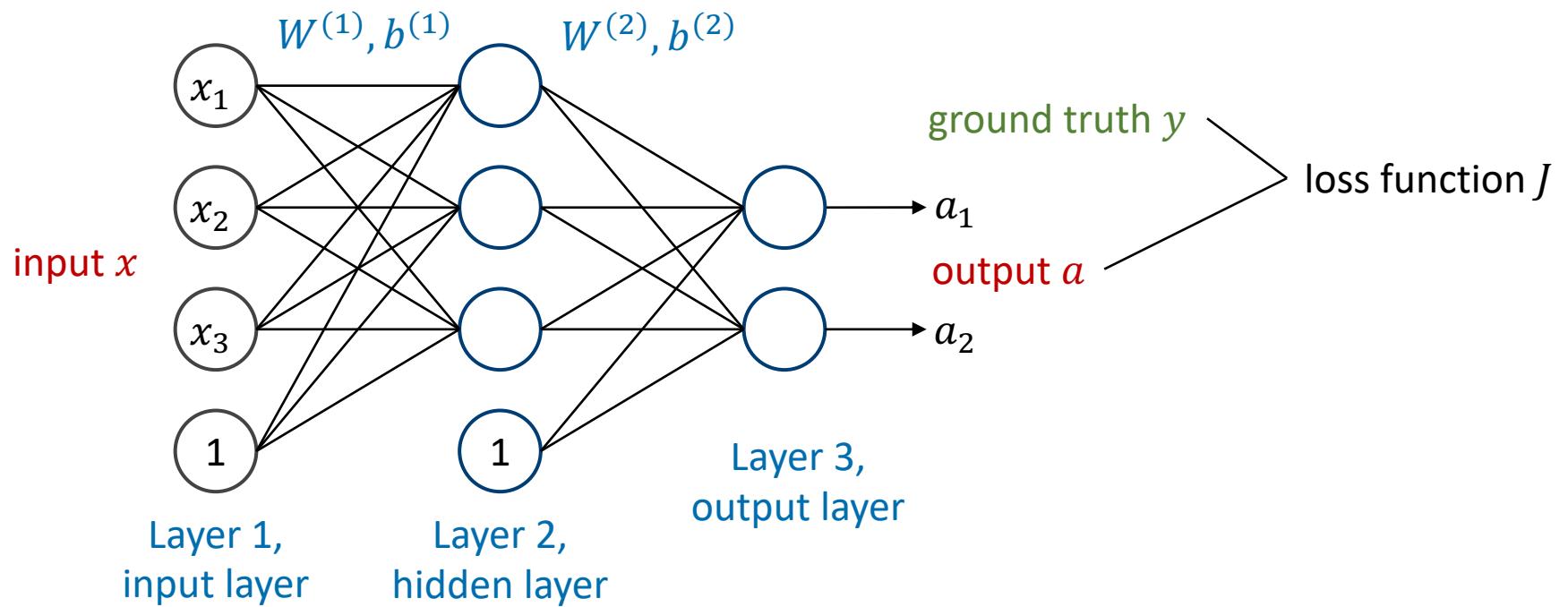
$$f(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid activation function

Multi-layer perceptron

- A multi-layer perceptron (MLP) is formed by putting several layers of neurons into connection, where the output of a neuron can be the input to another.



Multi-layer perceptron (MLP), which is a fully connected multi-layer network.

Training a neural network

- To train a neural network, we need a training set $\{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m), \dots, (x_M, y_M)\}$, which are paired data and ground truth labels.
- We would like to find parameters W and b so that given input x , the output of the network a matches y as close as possible.
- For example, we can define a loss function like this,

$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m - y_m\|^2$$

m-th sample

Training a neural network

- There are two technical questions here.
- Q1: How do we calculate the output of the network a given input x ?
 - A1: We use **forward propagation**.
- Q2: How do we find parameters W and b that minimise the loss function, so that a matching ground truth y as much as possible?
 - A2: We use **backpropagation** to calculate the gradient and then perform stochastic gradient descent for optimisation.
- You can find details in the machine learning module or in the supplementary slides.

Neural networks for image classification

- Suppose we know how a neural network works.
- How do we use it for image classification?
 - We need to define a loss function for image classification.
 - Then train the network, i.e. optimise the loss w.r.t. to network parameters.

Loss function

- Mean squared error

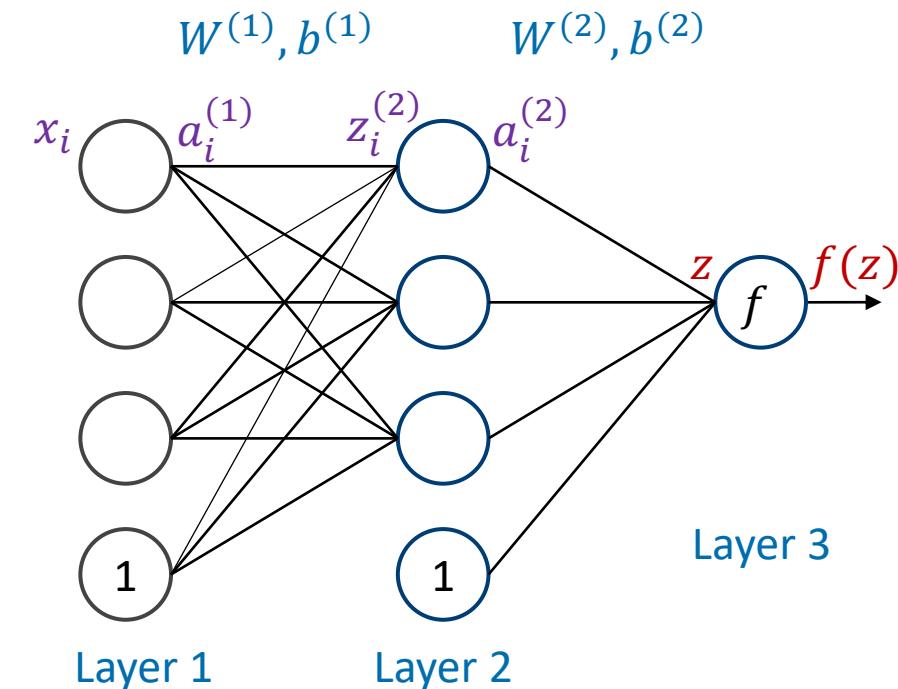
$$J(W, b) = \frac{1}{M} \sum_{m=1}^M \frac{1}{2} \|a_m - y_m\|^2$$

- This works for regression problems, i.e. y is a continuous variable.
- But it may not be optimal for image classification problems, where y is a categorical variable.
 - For binary classification, y can be 0 or 1.
 - For multi-class classification, for example,
 - in MNIST, y is one of the 10 label classes.
 - in ImageNet, y is one of the 1,000 label classes.

Binary classification

- Let us first consider the simplest case, binary classification.
- For binary classification, the output layer only needs 1 neuron.
- We can use the sigmoid activation function for the last layer. Its output is a value between 0 and 1, which is in the range of probability.

$$f(z) = \frac{1}{1 + e^{-z}}$$



Binary classification

- For example, the network predicts the probability to be 0.9 for class 1 and $1 - 0.9 = 0.1$ for class 0.
- Suppose the ground truth is $y = 1$, i.e. class 1. In other words, the probability is 1 for class 1 and 0 for class 0.
- How do we define the loss function?
 - We can define it as the distance between the predicted probability and the true probability.
 - Information theory provides us with a metric for probability distributions, called the cross entropy.

Loss function

- The cross entropy between a true probability distribution p and an estimated probability q is defined as,

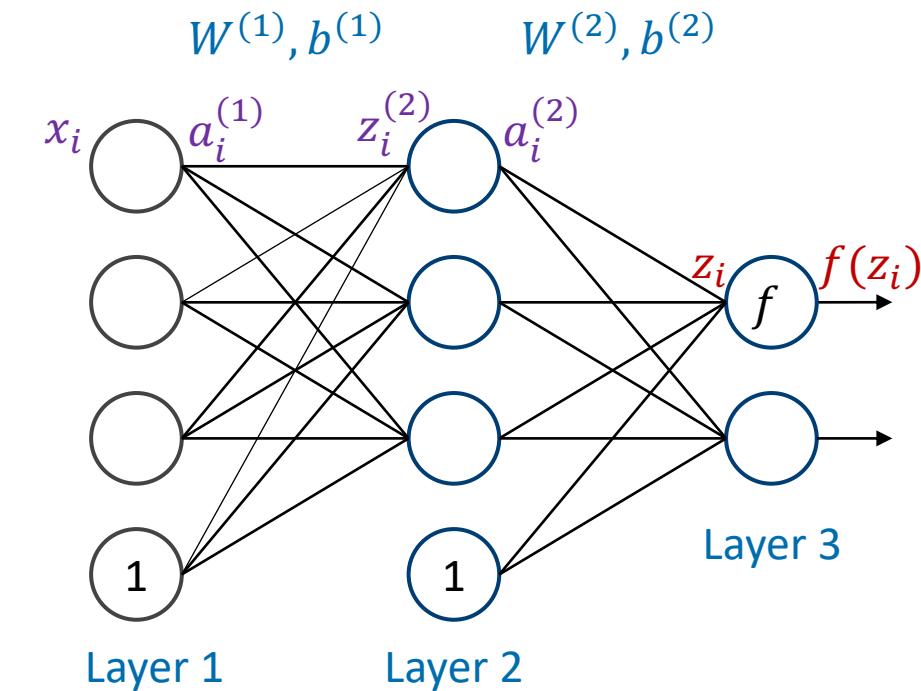
$$H(p, q) = - \sum_{i=1}^{\text{number of classes}} p_i \log(q_i)$$

- In our example (two classes), $p = [1, 0]$, $q = [0.9, 0.1]$,
$$H(p, q) = -(1 * \log(0.9) + 0 * \log(0.1)) = 0.105$$
- For general cases, $p = [y, 1 - y]$, $q = [f(z), 1 - f(z)]$,
$$H(p, q) = -[y \log(f(z)) + (1 - y) \log(1 - f(z))]$$

Multi-class classification

- For K classes, we put K neurons at output layer.
- To make the output of the K neurons to form a probability vector, which have positive values and sum to 1, we use the softmax function.

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$



Loss function

- The true probability $p = [y_1, \dots, y_i, \dots, y_K] = [0, \dots, 1, \dots, 0]$.
 - It is called one-hot encoding or representation.
 - Only one element is 1 (hot), all the others are 0 (cold).
- The estimated probability $q = [f(z_1), \dots, f(z_i), \dots, f(z_K)]$.
- The cross entropy loss is defined,

$$J(W, b; x, y) = H(p, q) = - \sum_{i=1}^K y_i \log(f(z_i))$$

Sigmoid vs softmax

- Sigmoid

- Binary classification
- The probabilities for two classes are

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{e^z + e^0}$$
$$1 - f(z) = \frac{e^0}{e^z + e^0}$$

- Softmax

- Multi-class classification
- The probability for each of the K classes is

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

- It generalises the sigmoid function to multiple classes.

Neural networks for image classification

- Now we know how to define a loss function for multi-class classification.
- Let us try this on MNIST dataset.



28x28 array with 784 pixels

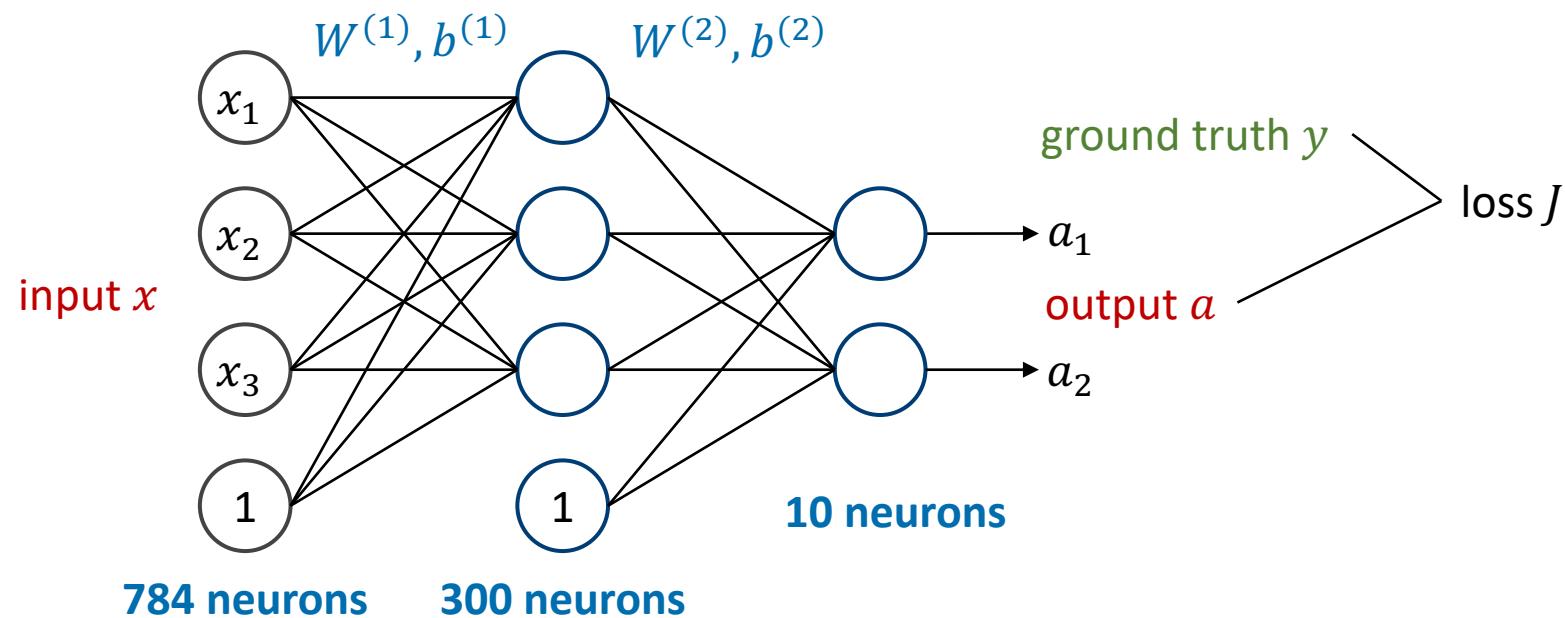


2

10 classes

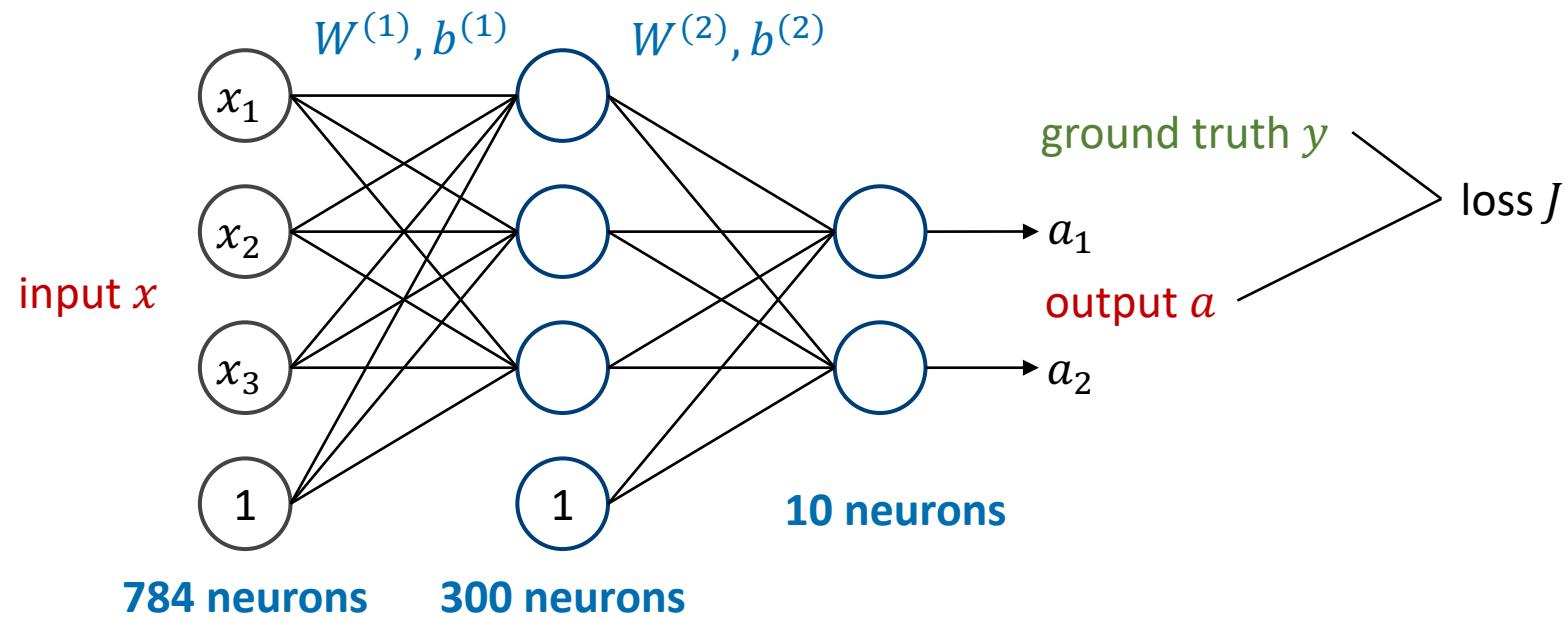
Multi-layer perceptron

- We build a 3-layered network
 - 784-300-10
 - cross entropy loss function



Multi-layer perceptron

- The network parameters can be trained using 60,000 training images.
- Then we can evaluate its performance on 10,000 test images.



Performance on MNIST

- This table reports the error rate.
 - Out of 10,000 test samples, how many have been wrongly classified.
 - Error rate = $100\% - \text{classification accuracy}$

Methods	Error Rate
KNN	5.0%
KNN (deslanted)	2.4%
SVM (polynomial, d=4)	1.1%
MLP (784-300-10)	4.7%

MNIST classification performance

Performance on MNIST

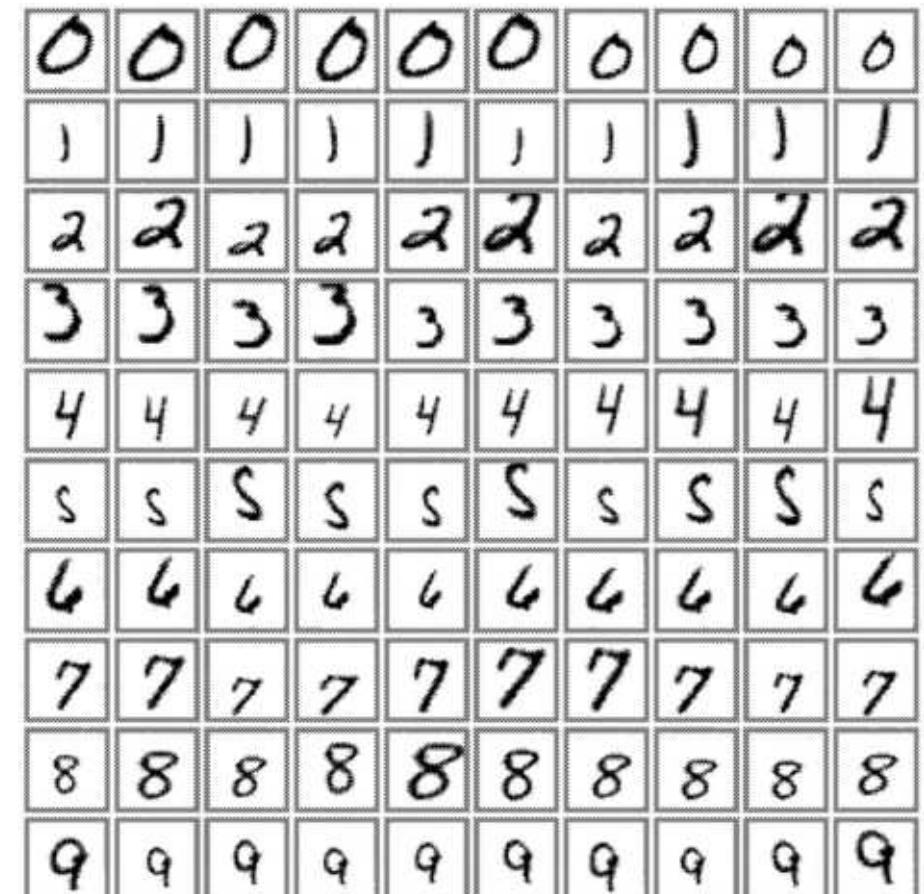
- We can try different network architectures.
 - 784-300-10
 - 784-1000-10
 - 784-500-150-10 (4-layered)

Methods	Error Rate
KNN	5.0%
KNN (deslanted)	2.4%
SVM (polynomial, d=4)	1.1%
MLP (784-300-10)	4.7%
MLP (784-1000-10)	4.5%
MLP (784-500-150-10)	2.95%

MNIST classification performance

Performance on MNIST

- We can perform data augmentation to create more training samples.
 - Apply affine transformation to the original 60,000 training samples
 - Translation
 - Scaling
 - Squeezing
 - Shearing
 - Generate augmented 540,000 samples



Examples of augmented samples.

Performance on MNIST

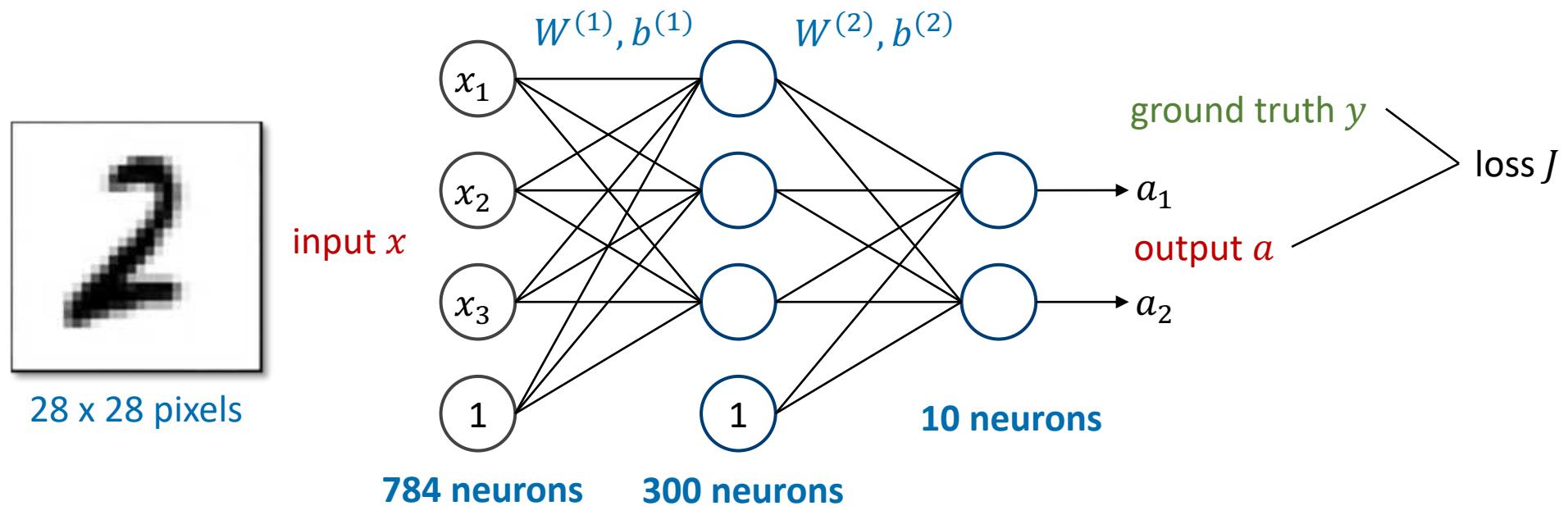
- Data augmentation improves classification performance.
- A 4-layered network with data augmentation can now achieve 2.45% error rate.

Methods	Error Rate
KNN	5.0%
KNN (deslanted)	2.4%
SVM (polynomial, d=4)	1.1%
MLP (784-300-10)	4.7%
MLP (784-300-10, aug.)	3.6%
MLP (784-1000-10)	4.5%
MLP (784-1000-10, aug.)	3.8%
MLP (784-500-150-10)	2.95%
MLP (784-500-150-10, aug.)	2.45%

MNIST classification performance

Limitations with MLP

- MLP uses many parameters. Even for a 3-layered network (784-300-10), without considering the bias parameters, it will use
 - $784 \times 300 + 300 \times 10 = 238,200$ parameters

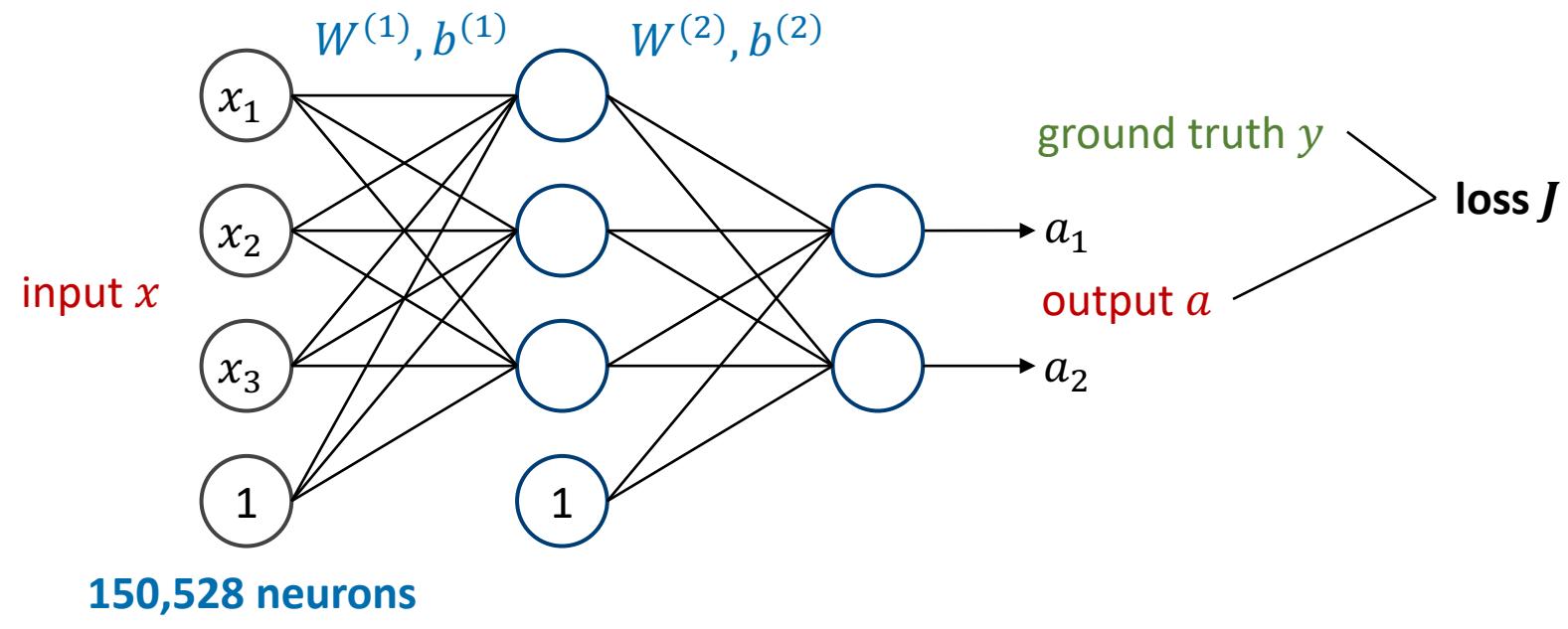


Limitations with MLP

- It may not scale up to bigger images, e.g. for ImageNet images,
 - For a 224×224 RGB images, $224 \times 224 \times 3$ channels = 150,528 neurons for Layer 1.
 - A single neuron in Layer 2 has 150,528 parameters, not to mention all neurons in this layer and following layers.



224 x 224 pixels by 3 channels



Limitations with MLP

- A reason why so many parameters are needed is because a 2D image is considered as a flattened vector, without considering its 2D nature.
- If we could develop an appropriate operator for 2D images, we may be able to have a more efficient network.

References

- Ch. 6, Deep Feedforward Networks. Ian Goodfellow et al. Deep Learning (<https://www.deeplearningbook.org/>).

Image Classification III

Dr Wenjia Bai

Department of Computing & Brain Sciences

Outline

- Convolutional neural networks
 - Building blocks
- Examples
 - LeNet-5 (1998)
 - AlexNet (2012)

Convolutional neural networks

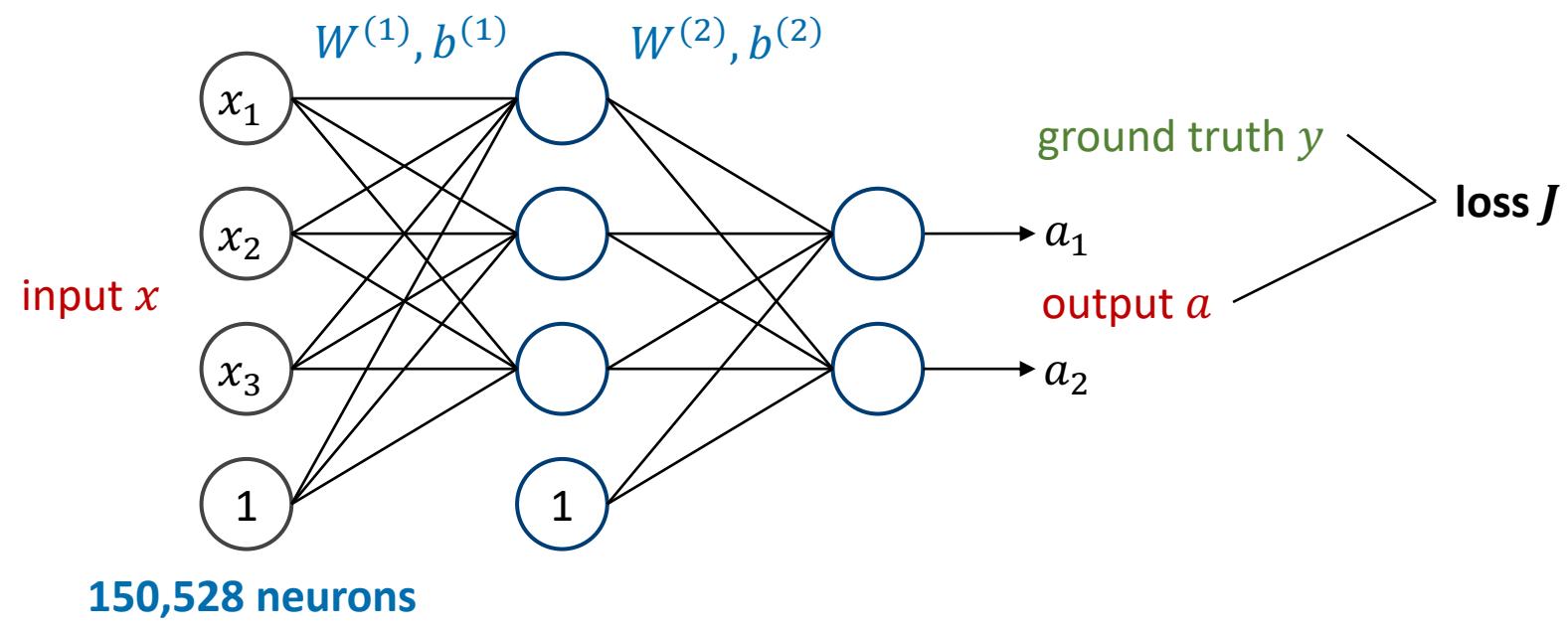
- Convolutional neural networks (CNNs) are similar to the multi-layer perceptron (MLP) in the previous lecture.
- However, CNNs assume that the inputs are images and encode certain properties (local connectivity, weight sharing etc.) into the architecture, which will make the computation more efficient and substantially reduce the number of parameters.

Limitations with MLP

- It may not scale up to bigger images. For example,
 - For a 224×224 RGB images, $224 \times 224 \times 3$ channels = 150,528 neurons for Layer 1.
 - A single neuron in Layer 2 has 150,528 parameters, not to mention all neurons on this layer and following layers.

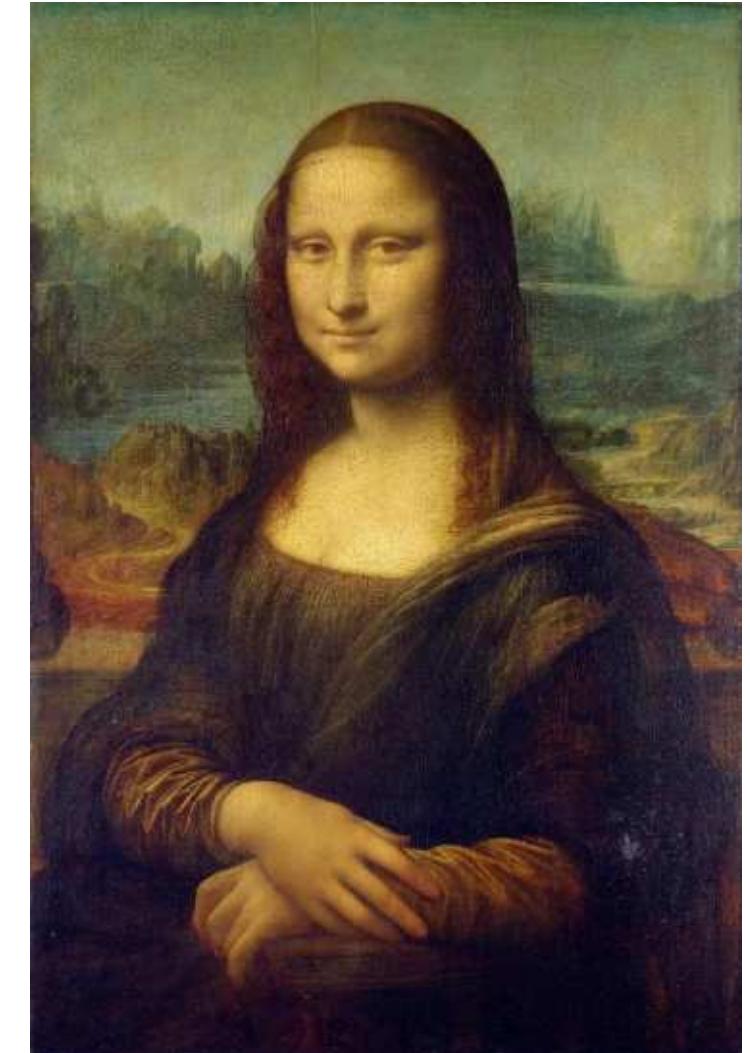


224 x 224 pixels by 3 channels

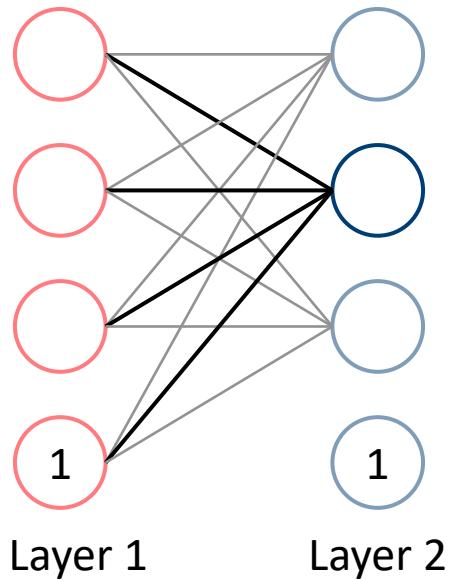


Convolutional neural networks

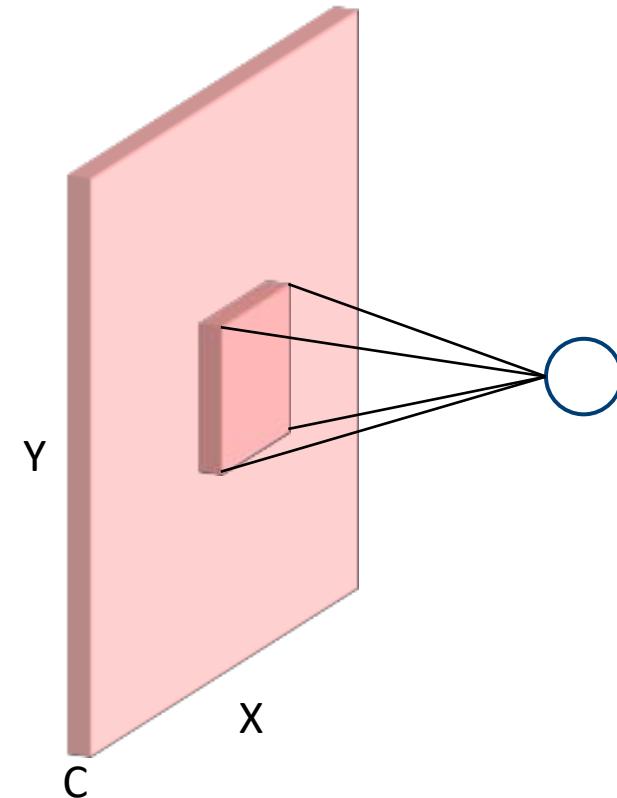
- When we look at images, we look at each local region, process information and combine the local information from different regions to form a global understanding.
- In CNNs, each neuron only see a small local region in the layer before it. The region it sees is called the receptive field.
- The local connectivity substantially reduces the number of parameters.



Convolutional neural networks

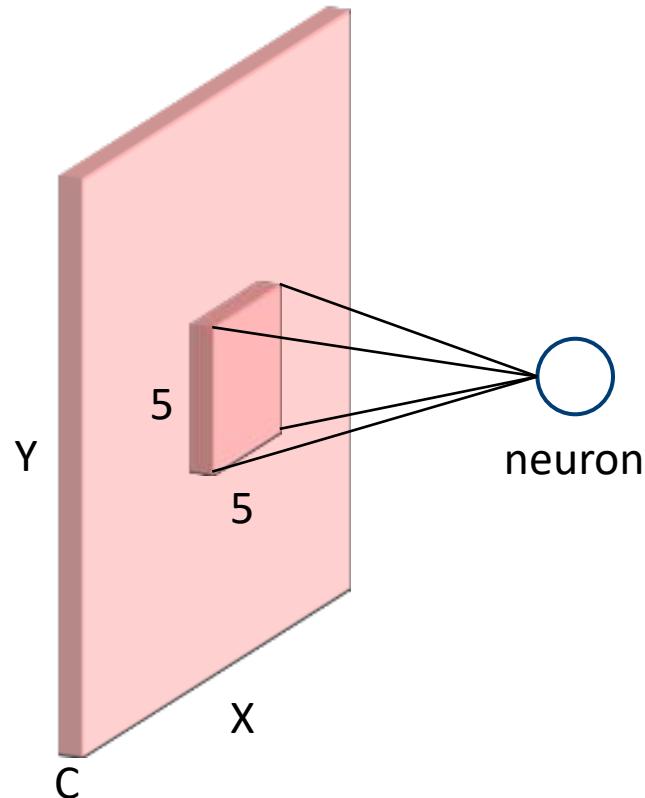


A fully connected layer in MLP.
A neuron in Layer 2 sees all the neurons in the previous layer.



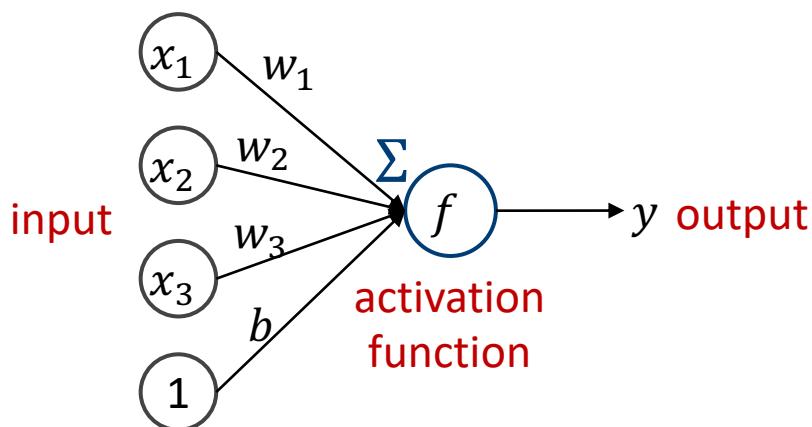
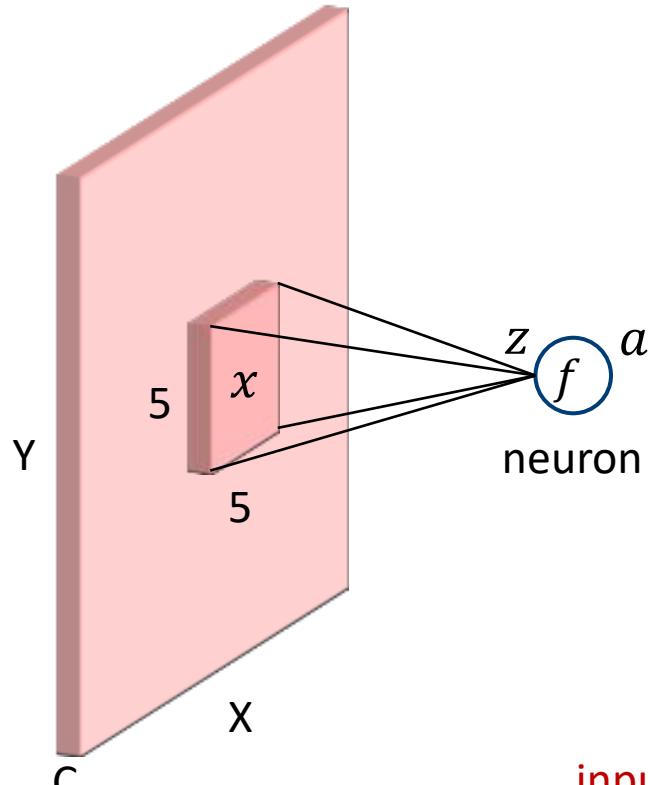
A convolutional layer in CNN.
A neuron in Layer 2 only depends on a small local region in the previous layer.

Convolutional layer



- Input: $X \times Y \times C$
- C : channel.
 - $C = 3$ for RGB image.
 - $C = 1$ for grayscale image.
- This neuron depends a $5 \times 5 \times C$ cube of the input.
- The number of parameters is $5 \times 5 \times C$ for connection weights and 1 for bias.

Convolutional layer



Analogy to a single neuron in MLP

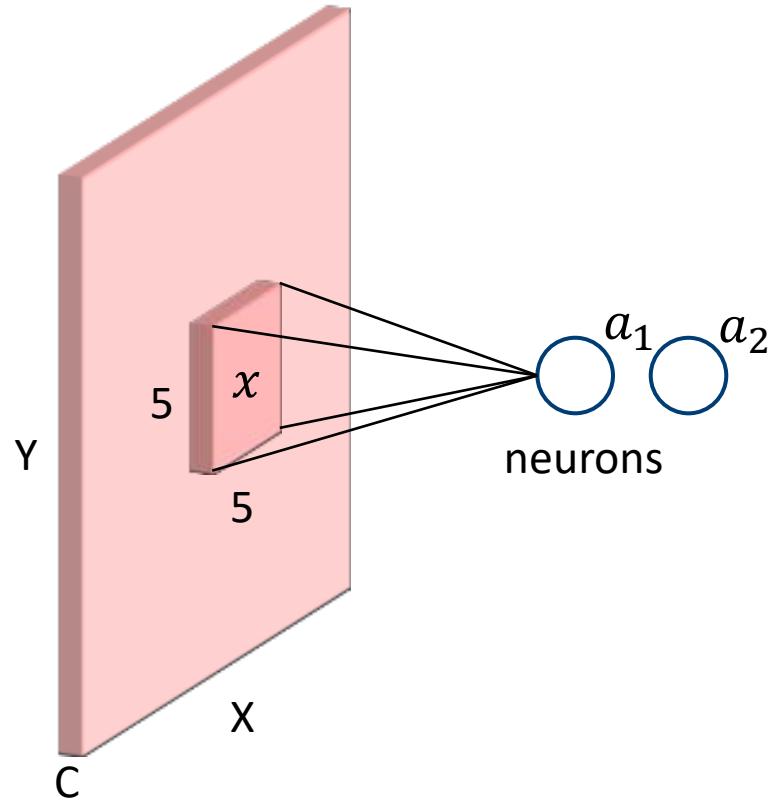
- The input and output of the neuron are defined by,

$$z = \sum_{ijk} \text{weights } W_{ijk} x_{ijk} + \text{bias}$$

sum over 3 axes

$$a = f(z) \quad \text{activation function}$$

Convolutional layer

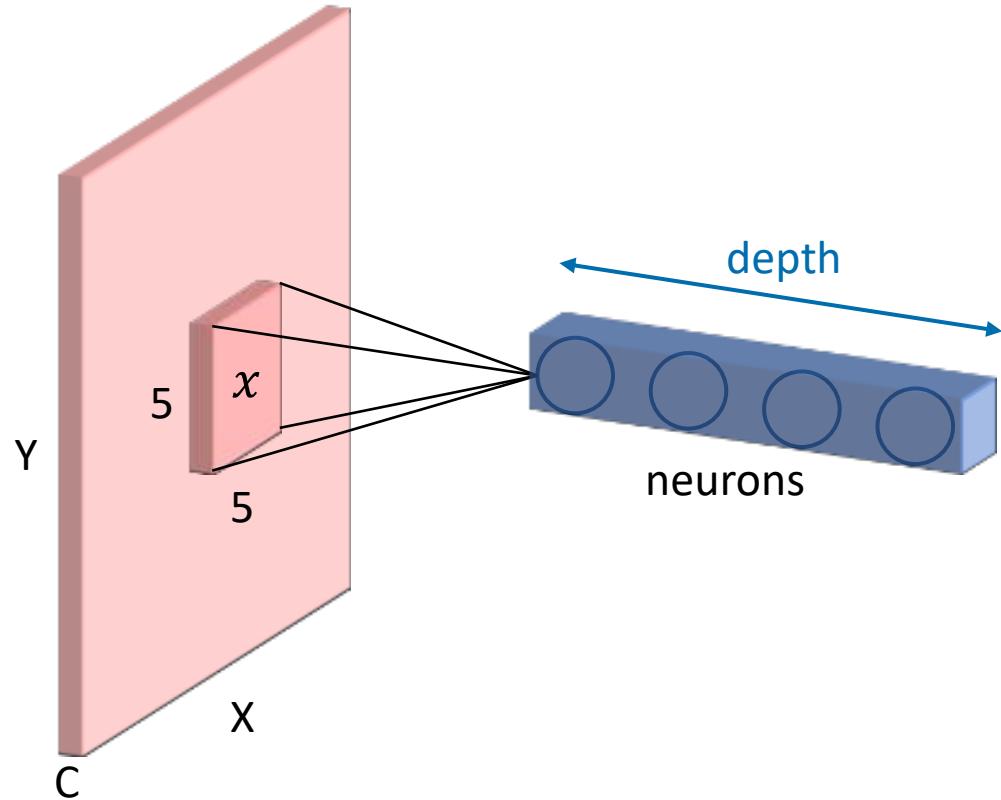


- With 2 neurons, the outputs are

$$a_1 = f\left(\sum_{ijk} W_{1ijk} x_{ijk} + b_1\right)$$

$$a_2 = f\left(\sum_{ijk} W_{2ijk} x_{ijk} + b_2\right)$$

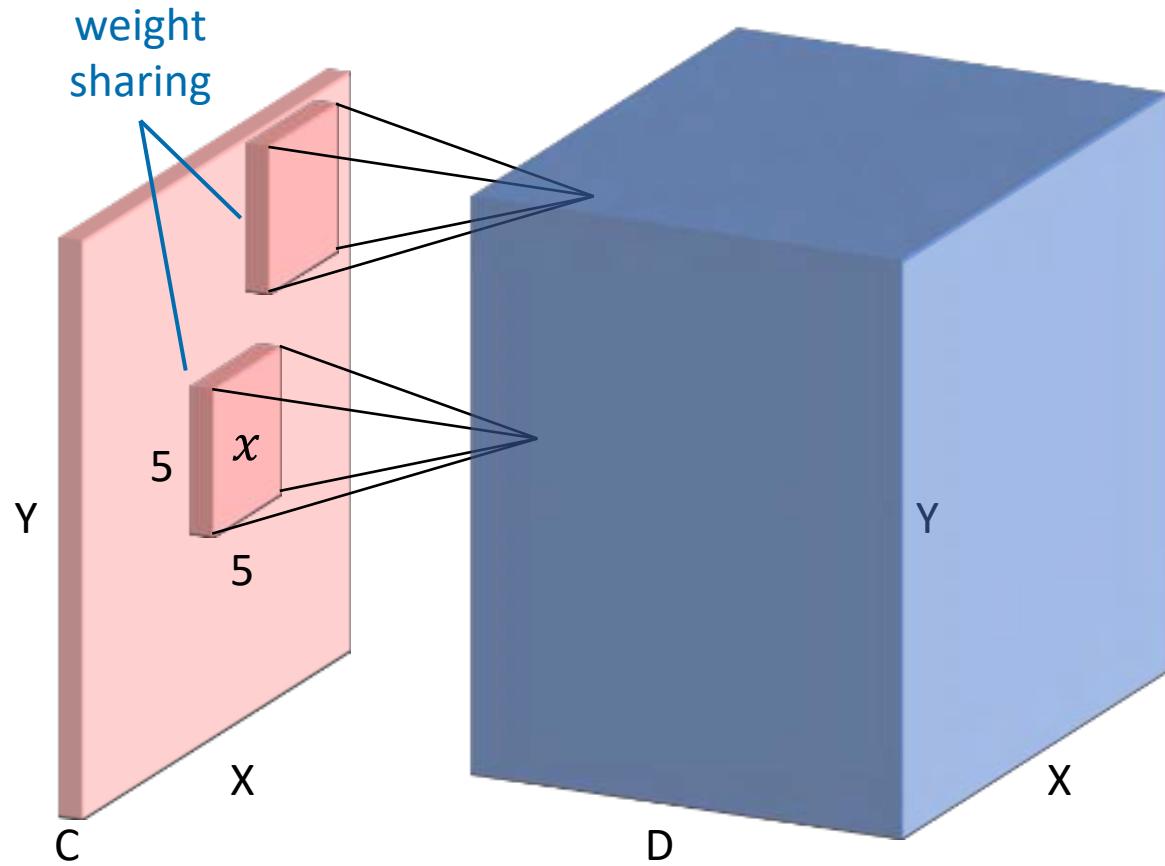
Convolutional layer



- We can add more neurons, which form a $D \times 1 \times 1$ cube of output, where D denotes depth.
- Each of the D neurons has its own weights W and bias b .

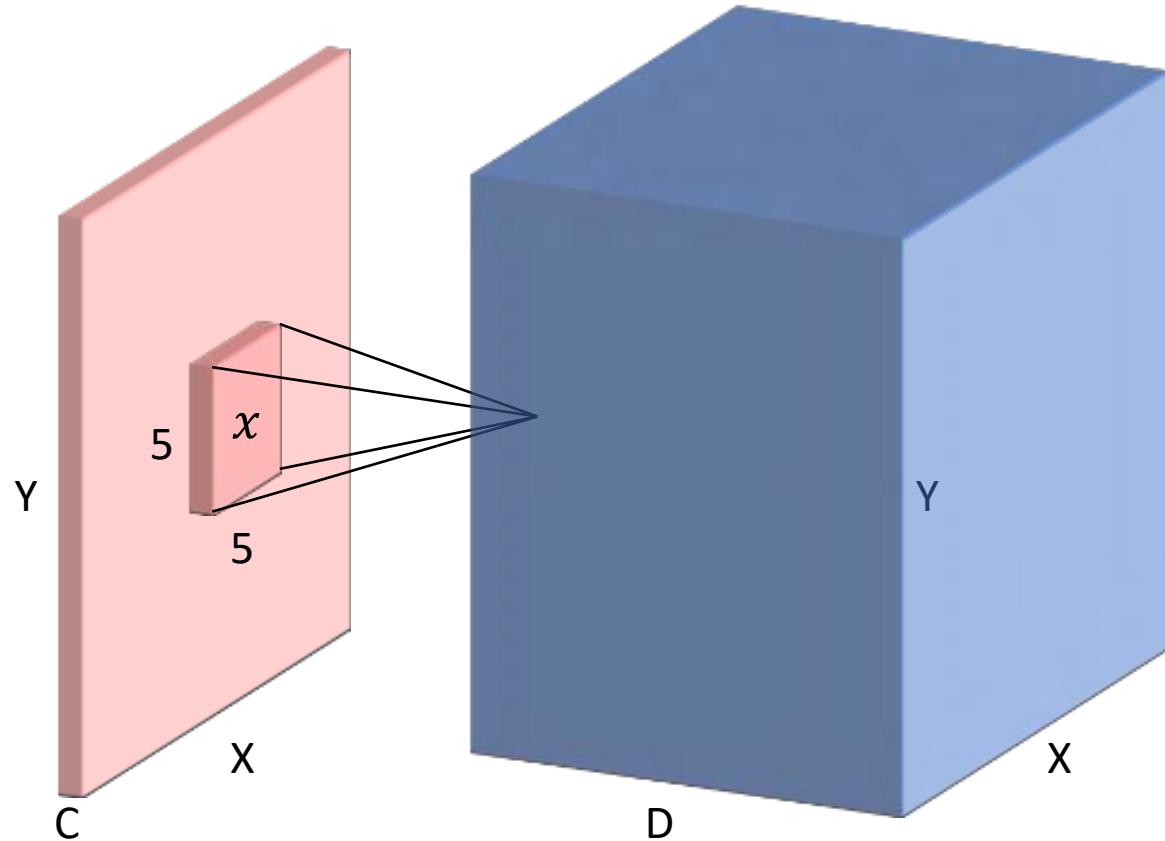
$$a_n = f\left(\sum_{ijk} W_{nijk} x_{ijk} + b_n\right)$$

Convolutional layer



- We can move the small window across the input image and get an output cube of size $X \times Y \times D$.
- Each of the small window shares the same weights (weight sharing).
- Therefore, for this example, the number of parameters is only $5 \times 5 \times C \times D$ for connection weights and D for bias.

Convolutional layer

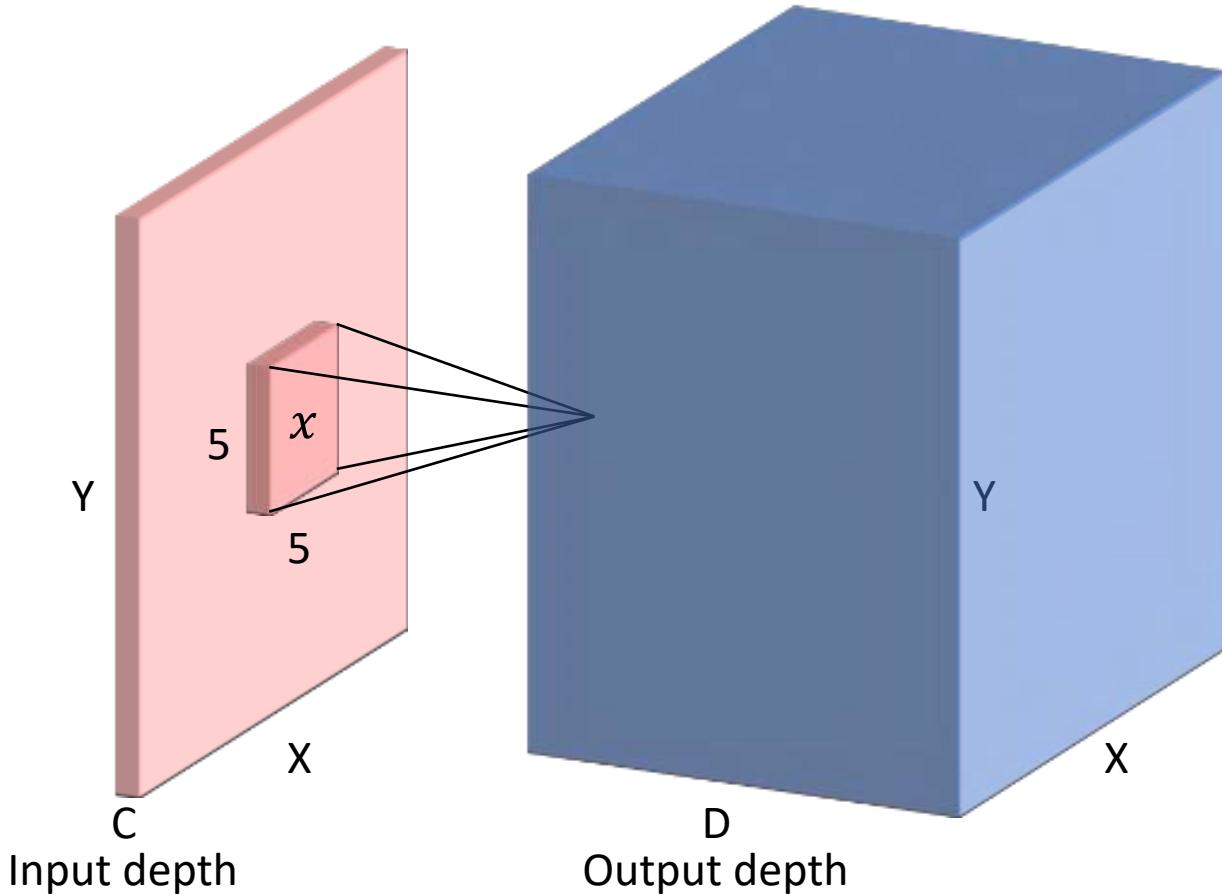


- This is called a convolutional layer.

$$a_d = f\left(\sum_{ijk} W_{dijk} x_{ijk} + b_d\right)$$

convolutional kernel

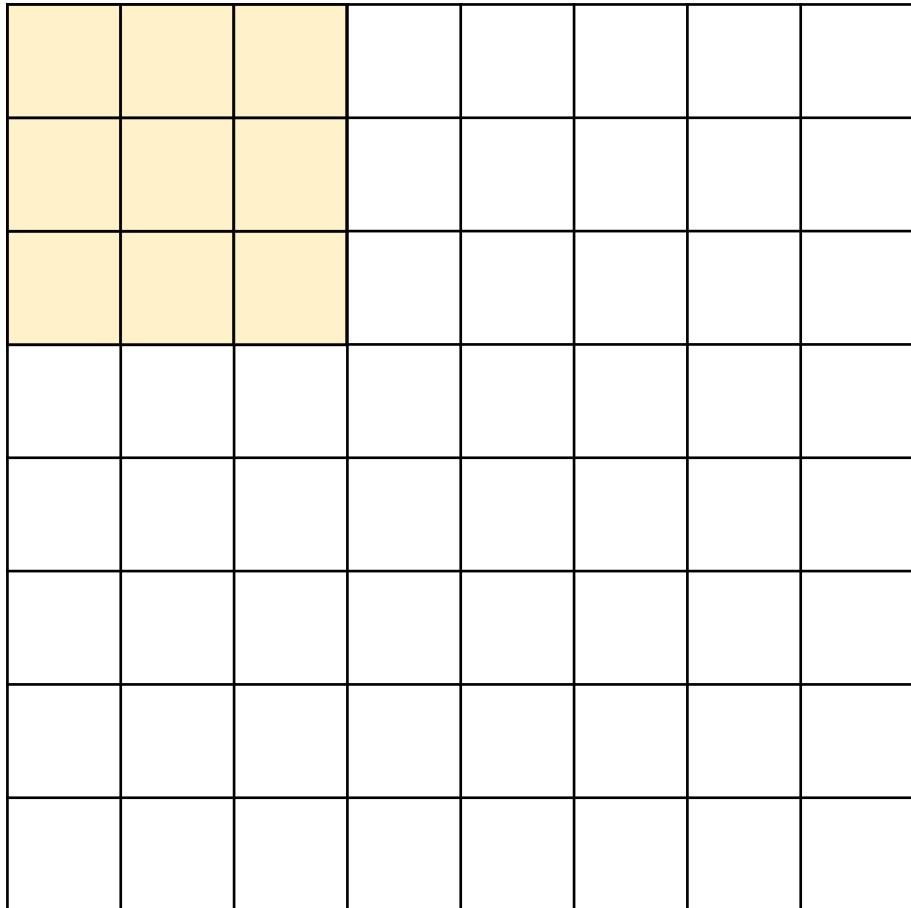
Convolutional layer



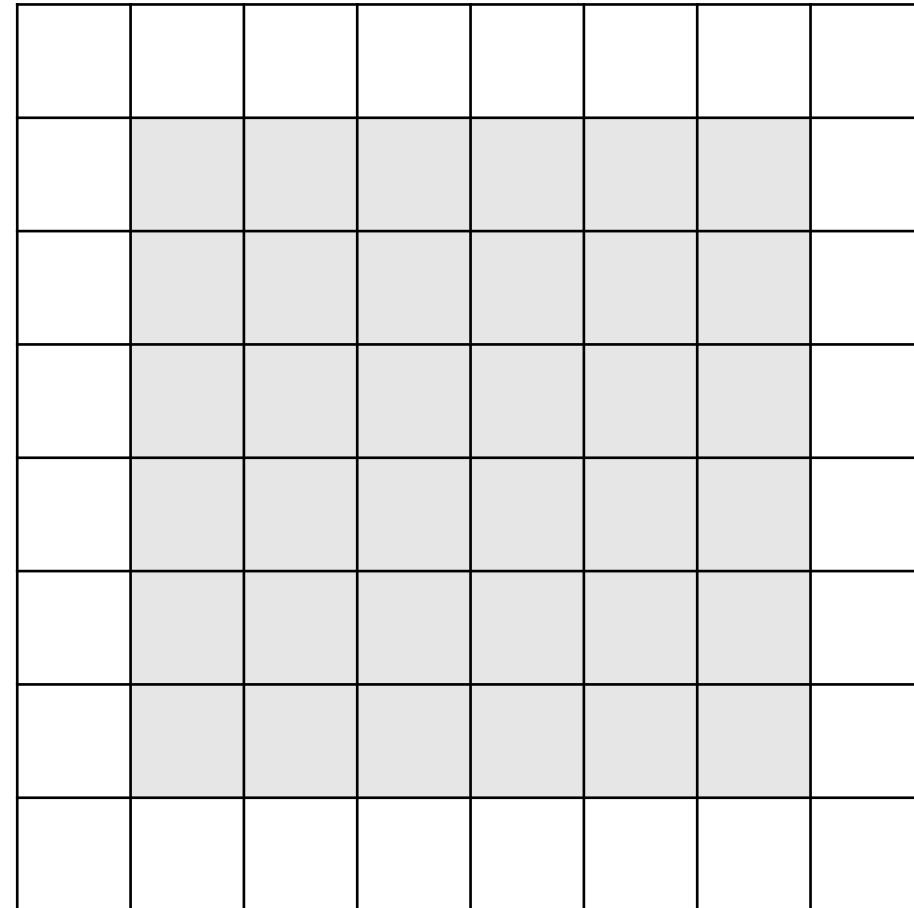
- All together, a 2D convolutional kernel W has four dimensions:
$$a_d = f\left(\sum_{ijk} W_{dijk} x_{ijk} + b_d\right)$$
 - Output depth
 - Kernel width
 - Kernel height
 - Input depth
- The output feature map has three dimensions:
 - Output depth
 - Width
 - Height

Convolutional layer

- Some operations during convolution
 - Padding
 - Stride
 - Dilation



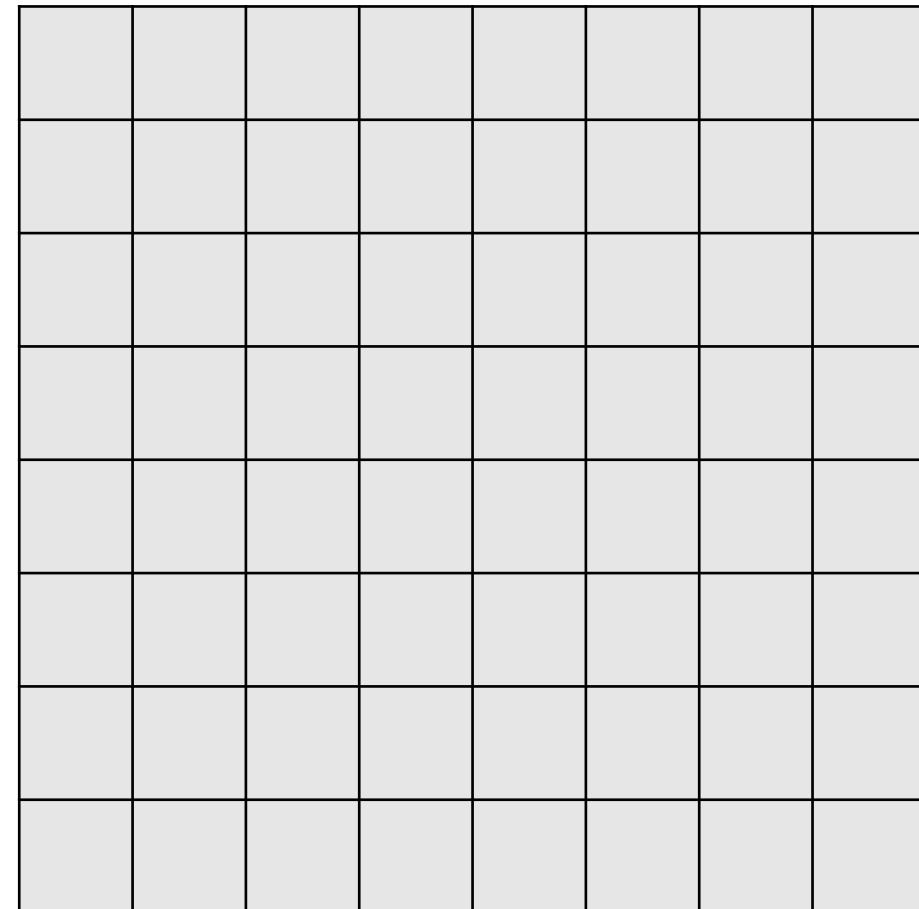
3x3 convolutional kernel applied
to an image of shape $[X, Y]$



output shape = $[X - 2, Y - 2]$

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

3x3 convolutional kernel applied
to the image with padding $p = 1$



output shape = [X, Y]

Stride

- We do not need to move the window across all the pixels.
- We can use a stride, e.g. $s=2$, to move the window faster and look at a downsampled grid.

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

padding p = 1, stride s = 2

output shape = $[X/2, Y/2]$

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

padding p = 1, stride s = 2

output shape = $[X/2, Y/2]$

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

padding p = 1, stride s = 2

output shape = $[X/2, Y/2]$

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

padding p = 1, stride s = 2

output shape = $[X/2, Y/2]$

Dilation

- If we want a neuron to look at a large region (receptive field) in the image, we can increase the kernel size to 5x5 or 7x7. However, this will increase the number of parameters.
- Dilation aims to increase the receptive field on the original image without increasing the number of parameters.

X	X	X						
X	X	X						
X	X	X						

3x3 convolutional kernel,
dilation = 1

X	X	X	X	X				
X	X	X	X	X				
X	X	X	X	X				
X	X	X	X	X				
X	X	X	X	X				

5x5 convolutional kernel,
dilation = 1

X		X		X				
X		X		X				
X		X		X				

3x3 convolutional kernel,
dilation = 2

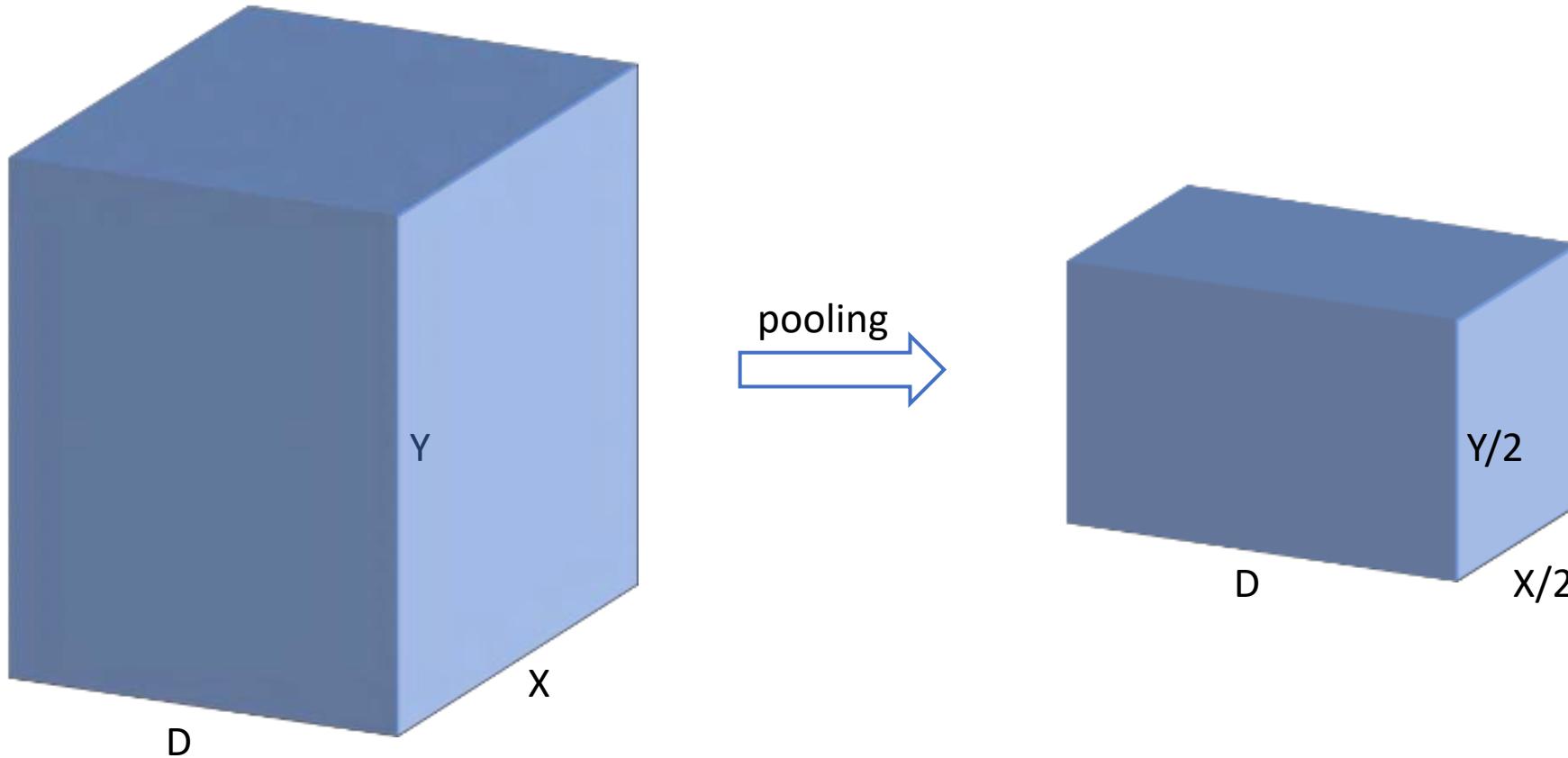
Convolutional layer

- When we perform convolution, we can set these parameters
 - Padding, which affects output size.
 - Stride, which downsamples the image.
 - Dilation, which increases the receptive field.

Pooling layer

- Apart from convolutional layer, pooling layer is another building block for convolutional neural networks.
- It is an operation to make feature maps or representations smaller.
- It is similar to image downsampling.
- After pooling, neurons in the next layer can see a larger region of image.

Pooling layer



Max pooling

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

max pooling
with 2x2 window
and stride=2


6	8
4	3

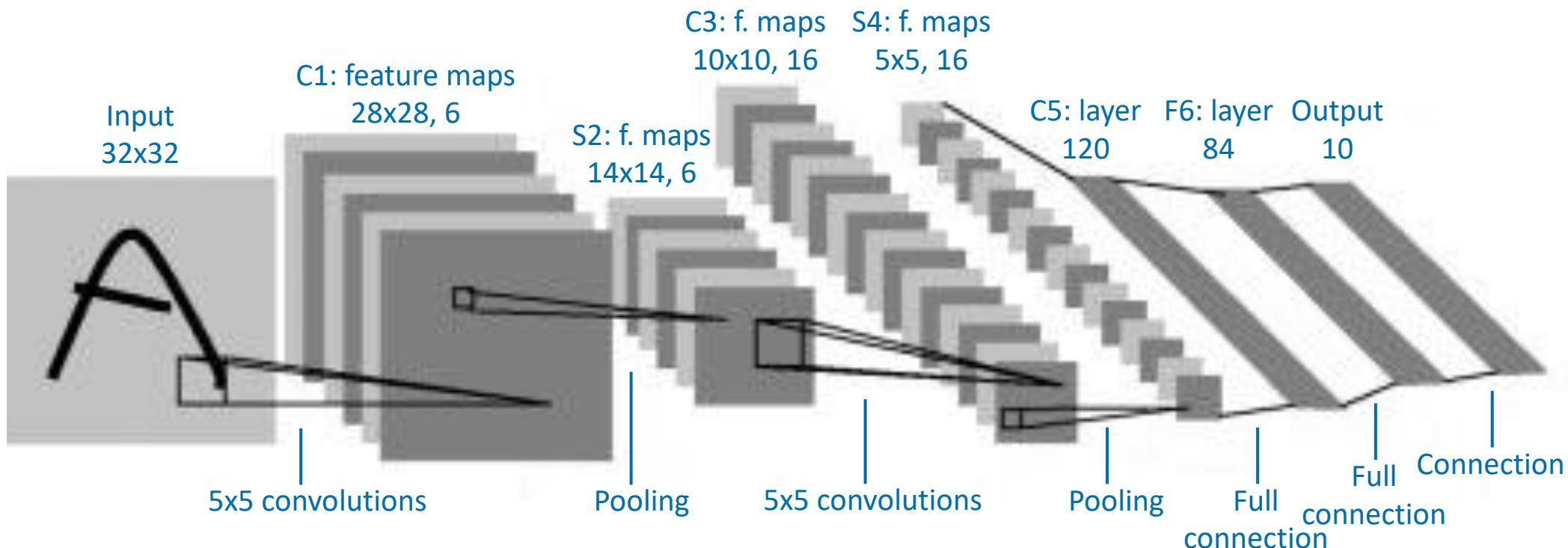
Convolutional neural networks

- After introducing the convolutional layers and basic operations, we can use them to build a convolutional neural network.
- We will show a classic example here.

LeNet-5

- 7 layers in total, not counting the input layer.

- C: convolutional layer
- S: pooling layer (subsampling)
- F: fully connected layer



LeNet-5

Small number of parameters here.

C1: $5 \times 5 \times 6$ (weight) + 6 (bias) = 156 parameters

Large feature map.

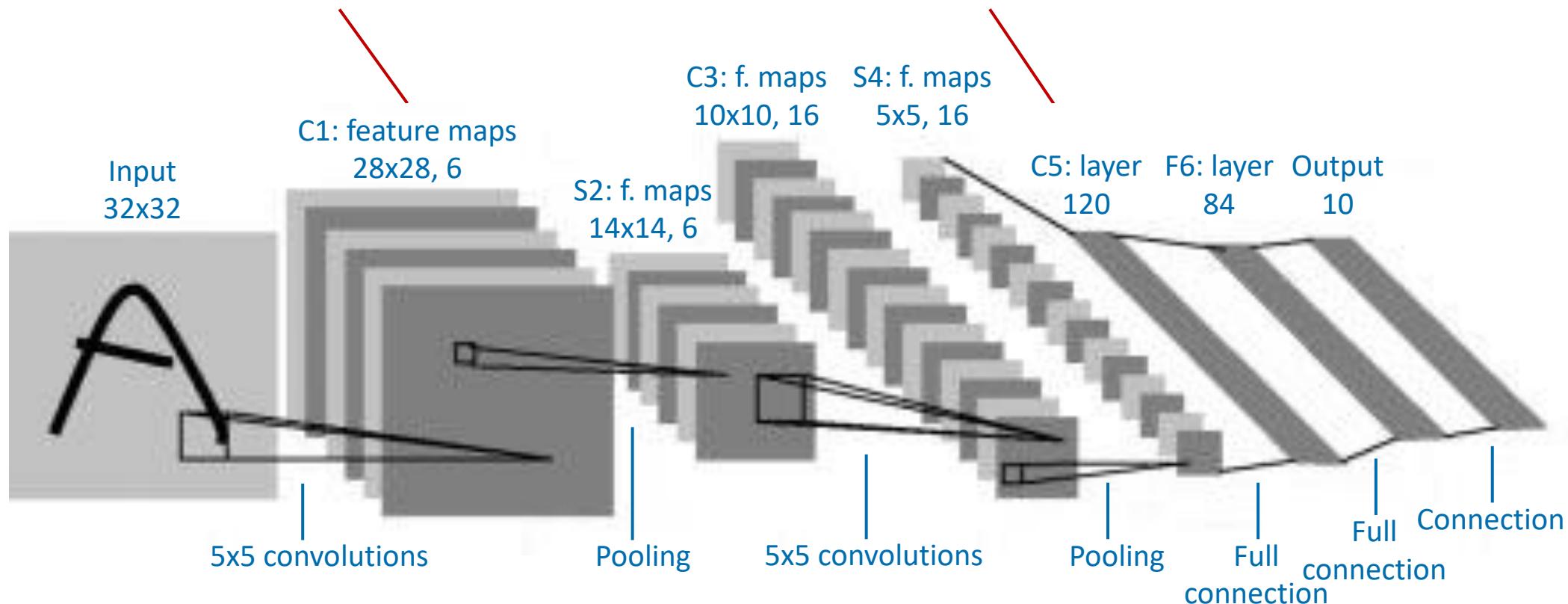
C1: 28×28 pixel $\times 6$ channel = 4,704 floating-point num.

Large number of parameters here.

C5: $5 \times 5 \times 16 \times 120$ (weight) + 120 (bias) = 48120 parameters

Small feature map, abstract representation of the image.

C5: 120 floating-point number



LeNet-5

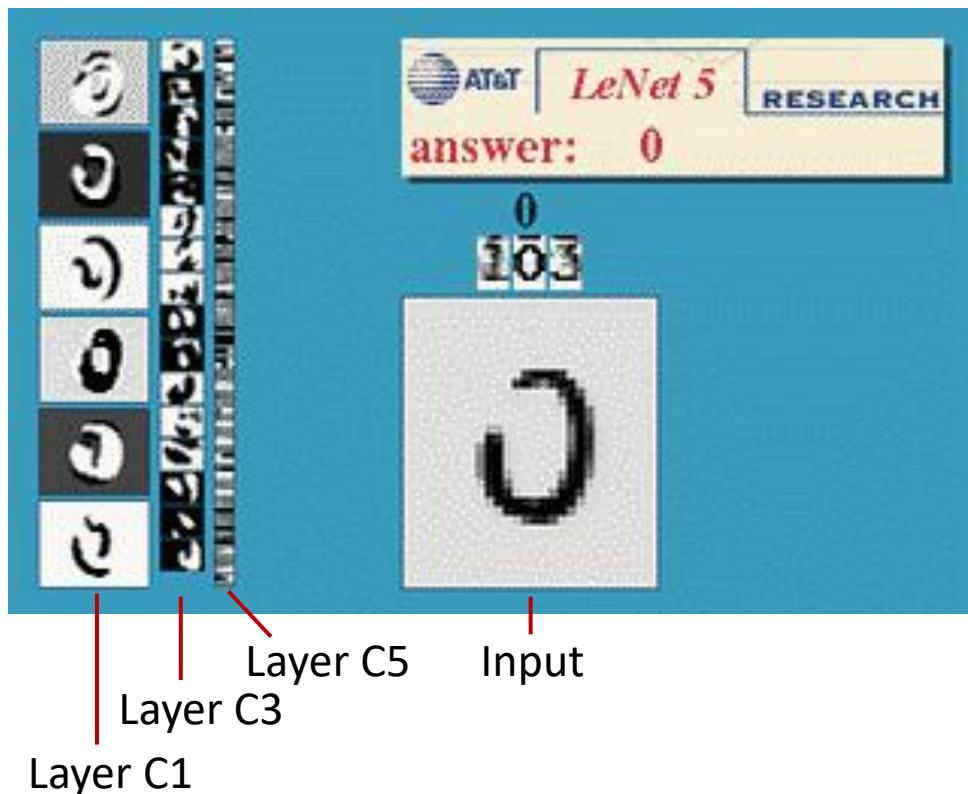
- We can define a loss function, e.g. cross-entropy, for classification.
- Each convolutional layer is mathematically similar to a layer in MLP,

$$z = \sum_{ijk} W_{ijk} x_{ijk} + b$$
$$a = f(z)$$

- The network can be trained similarly as MLP, using backpropagation and stochastic gradient descent.
- After training, we can evaluate its performance on test data.

LeNet-5

- Now performance is comparable to SVM.



Demo from <http://yann.lecun.com/exdb/lenet/>.

Methods	Error Rate
KNN	5.0%
KNN (deslanted)	2.4%
SVM (polynomial, d=4)	1.1%
V-SVM (polynomial, d=9, augment.)	0.8%
MLP (784-300-10)	4.7%
MLP (784-300-10, augment.)	3.6%
MLP (784-1000-10)	4.5%
MLP (784-1000-10, augment.)	3.8%
MLP (784-500-150-10)	2.95%
MLP (784-500-150-10, augment.)	2.45%
LeNet-5	0.95%
LeNet-5 (augment.)	0.8%

MNIST classification performance

Deep neural networks

- In the 1990s and 2000s, it was technically challenging to develop and train a deep neural network.
- Instead, many people used SVM or random forests for classification or regression problems.
- When did deep neural networks become popular again?
 - In ICASSP 2011, a method based on deep belief networks (DBN) increased the speech recognition accuracy by a large margin [1].
 - In NIPS 2012, a deep convolutional neural network substantially increased the image classification accuracy in the ImageNet challenge [2].

[1] G.E. Dahl et al. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. ICASSP 2011.

[2] A. Krizhevsky et al. ImageNet classification with deep convolutional neural networks. NIPS 2012.

Deep neural networks

- Why did it become successful?
 - Hardware
 - Efficient use of graphical processing units (GPUs)
 - Data
 - Large datasets (e.g. ImageNet)
 - Algorithm
 - Improvement of optimisation: ReLU
 - Network architectures: AlexNet, VGG, ResNet, Transformer ...

Hardware

- Graphical processing units (GPU)
 - Developed for computer graphics and games.
 - Ideal for training neural networks, performing convolutional operations in parallel using thousands of cores on a GPU card.



Large datasets

- ImageNet Challenge
 - 256 x 256 x 3 RGB image
 - 1.2 million images
 - 1,000 classes



<http://www.image-net.org>

Improvement in optimisation

- Stochastic gradient descent

- For each batch of samples, we perform gradient descent

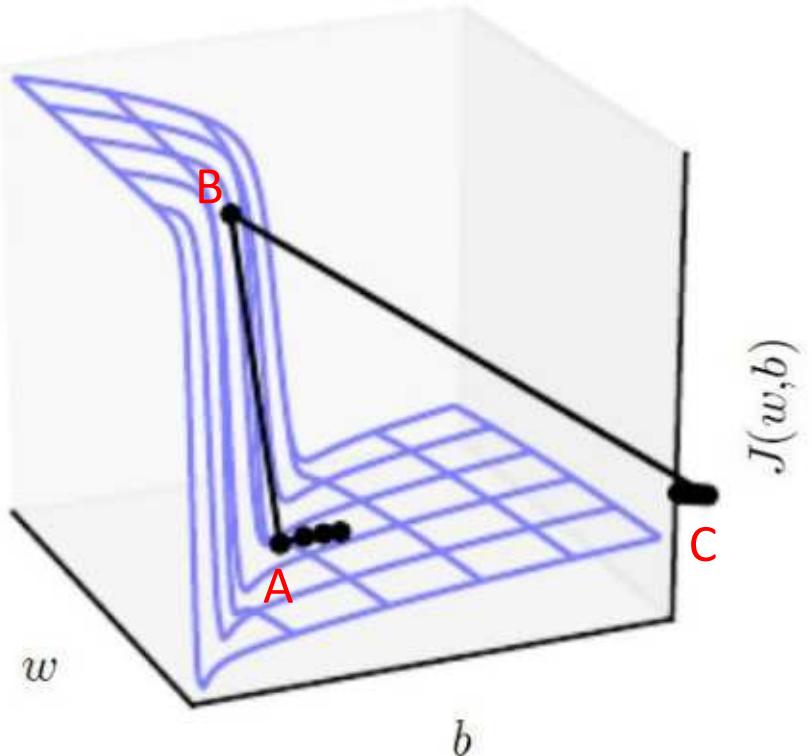
$$W = W - \alpha \frac{\partial L_B}{\partial W}$$
$$b = b - \alpha \frac{\partial L_B}{\partial b}$$

- Problems in optimisation

- Exploding gradient: The gradient becomes very large, preventing the algorithm from converging.
 - Vanishing gradient: The gradient becomes vanishingly small, preventing the weights from changing their values.

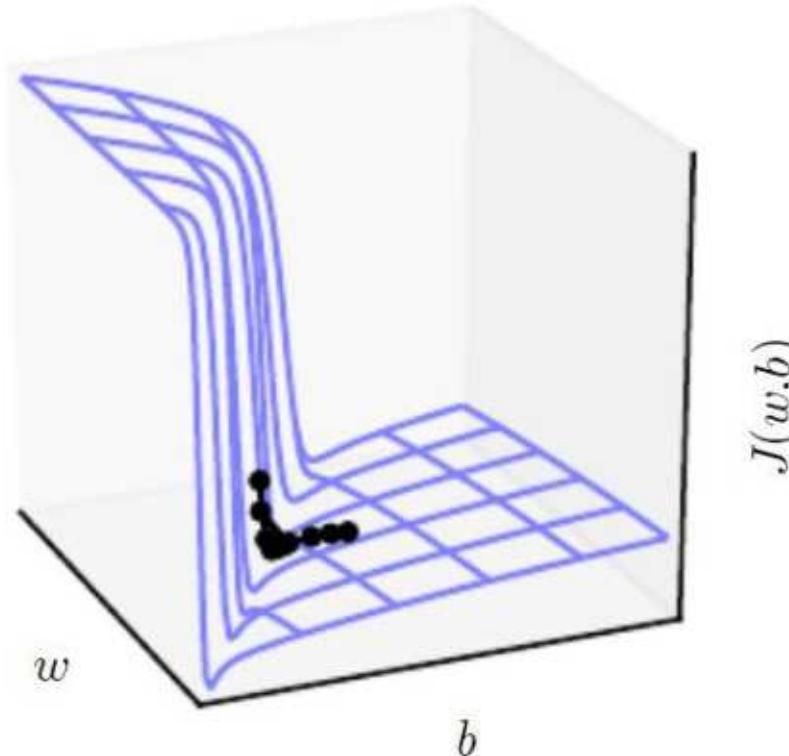
Exploding gradient problem

Without clipping



The gradient becomes very large in the vicinity of the cliff. Gradient descent will overshoot from A to B, then to C. It can not converge.

With clipping



After clipping the gradients, gradient descent converges.

Gradient clipping

- Clip by value

- For each element of the gradient g , clip by a minimal value and a maximal value,

$$g_i = \begin{cases} v_{min}, & \text{if } g_i < v_{min} \\ v_{max}, & \text{if } g_i > v_{max} \\ g_i, & \text{otherwise} \end{cases}$$

- Clip by norm

- Clip by the L2-norm of the gradient g ,

$$g = \begin{cases} \frac{g}{\|g\|} \nu, & \text{if } \|g\| > \nu \\ g, & \text{otherwise} \end{cases}$$

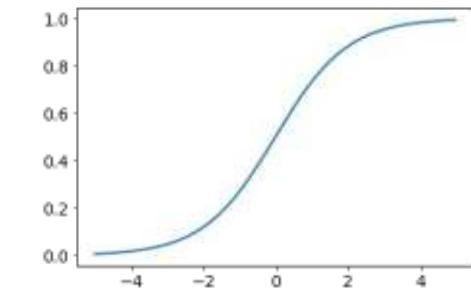
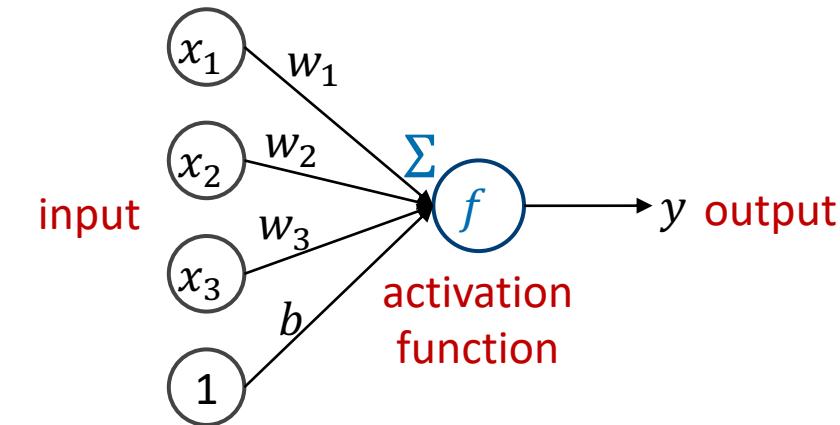
Vanishing gradient problem

- Let us revisit what a single neuron looks like.
- The neuron is a computational unit that takes an input, applies an activation function f and generates an output.

$$y = f\left(\sum_{i=1}^3 w_i x_i + b\right)$$

- For many years, people use the sigmoid function for f .

$$f(z) = \frac{1}{1 + e^{-z}}$$



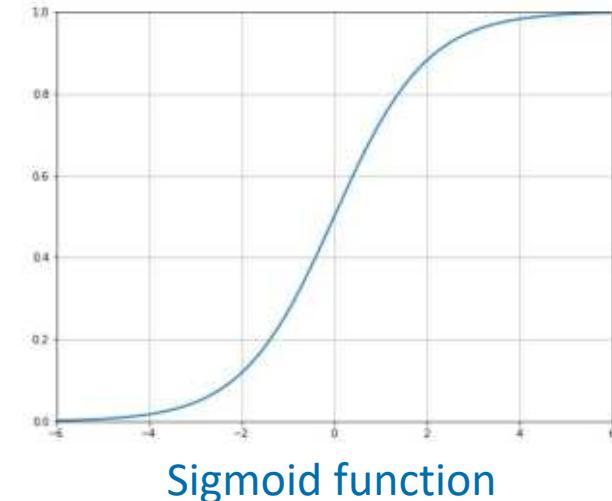
Sigmoid activation function

Vanishing gradient problem

- Sigmoid function

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$f'(z) = \frac{e^{-z}}{(1 + e^{-z})^2} = f(z)(1 - f(z))$$



- If $f(z)$ saturates at 0 or 1, then $f'(z)$ becomes almost 0.
- This causes a problem in backpropagating the gradient.

Backpropagation algorithm (from supplementary slides)

- Neuron inputs $z^{(l)}$, activations $a^{(l)}$.

- At the output layer, calculate the derivative $\frac{\partial J}{\partial z^{(L)}}$,

$$\frac{\partial J}{\partial z^{(L)}} = (a^{(L)} - y) \circ f'(z^{(L)})$$

- For layer $l = L - 1, L - 2, \dots, 1$, calculate the propagated derivatives,

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}} (a^{(l)})^T$$

$$\frac{\partial J}{\partial b^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}}$$

$$\frac{\partial J}{\partial z^{(l)}} = ((W^{(l)})^T \frac{\partial J}{\partial z^{(l+1)}}) \circ f'(z^{(l)})$$

If the sigmoid function saturates,
gradient can not be backpropagated.

Activation function

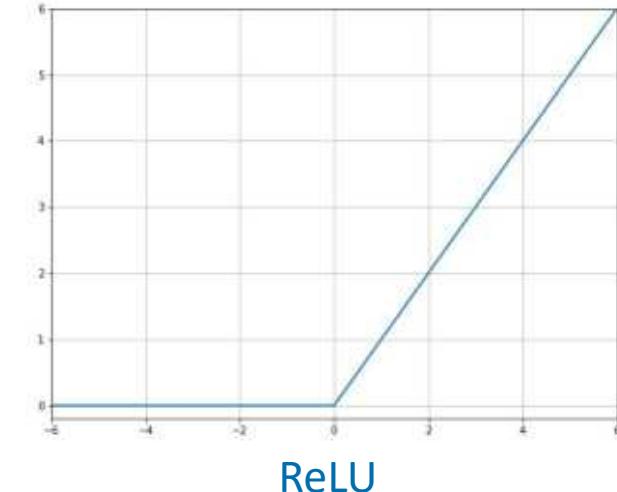
- Rectified linear unit (ReLU)

$$f(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$

- Its subgradient is

$$f'(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

- The gradient will not vanish when z is very large.
- However, the gradient becomes 0 when z becomes negative.
- For deep neural networks, it has been found to outperform the sigmoid activation function and thus it has been widely used.



Activation function

- Leaky ReLU

$$f(z) = \begin{cases} 0.01z, & z < 0 \\ z, & z \geq 0 \end{cases}$$

- Parameteric ReLU (PReLU): similar to leaky ReLU, but learning the parameter a in training.

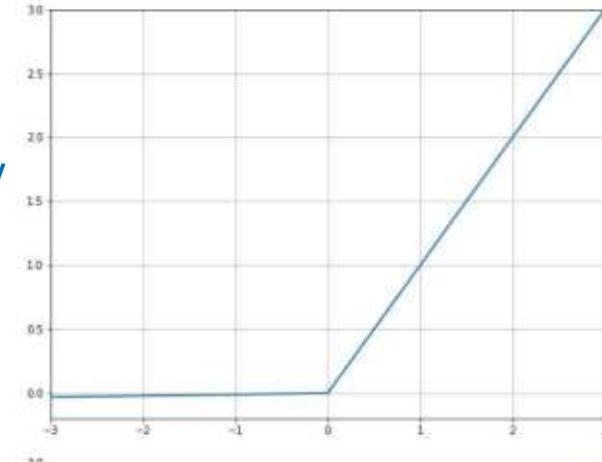
$$f(z) = \begin{cases} az, & z < 0 \\ z, & z \geq 0 \end{cases}$$

- Exponential linear unit (ELU)

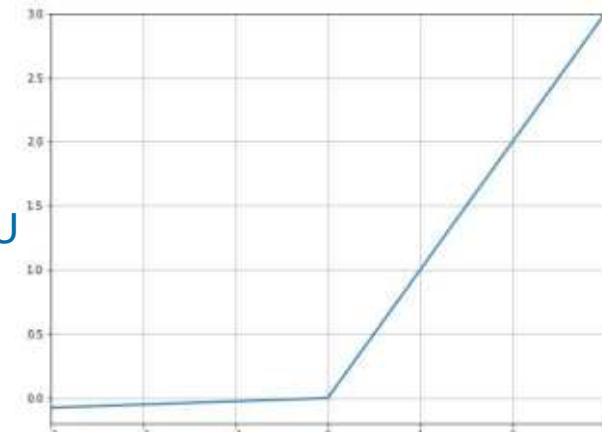
$$f(z) = \begin{cases} a(e^z - 1), & z < 0 \\ z, & z \geq 0 \end{cases}$$

- They are proposed to address vanishing gradient problem.

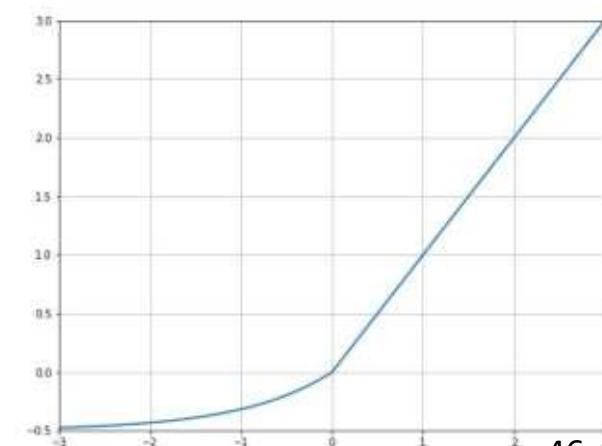
Leaky
ReLU



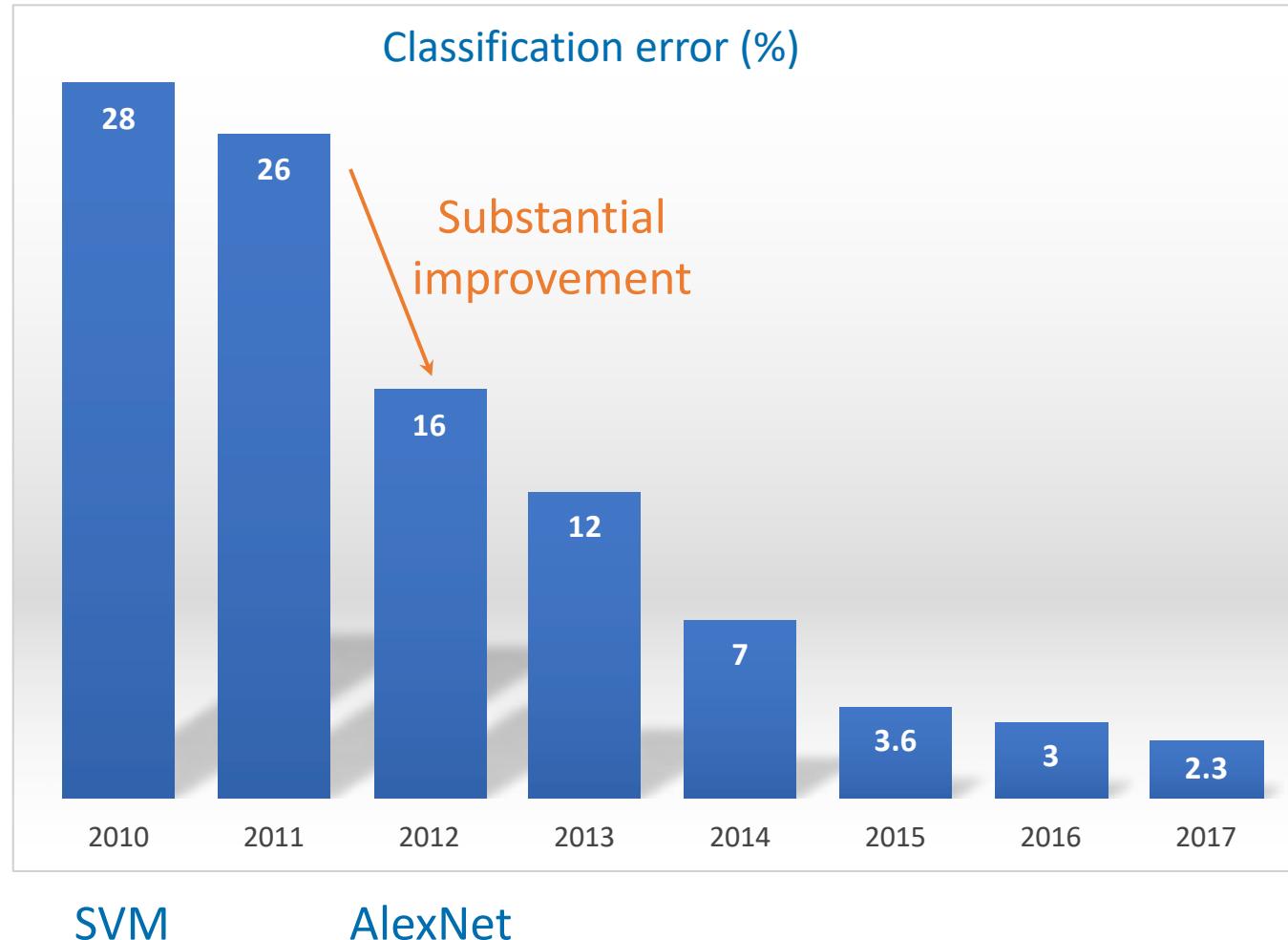
PReLU



ELU



ImageNet classification challenge



Architecture

- AlexNet

- **Input**

- $224 \times 224 \times 3$

- **Layer 1 (Conv 1)**

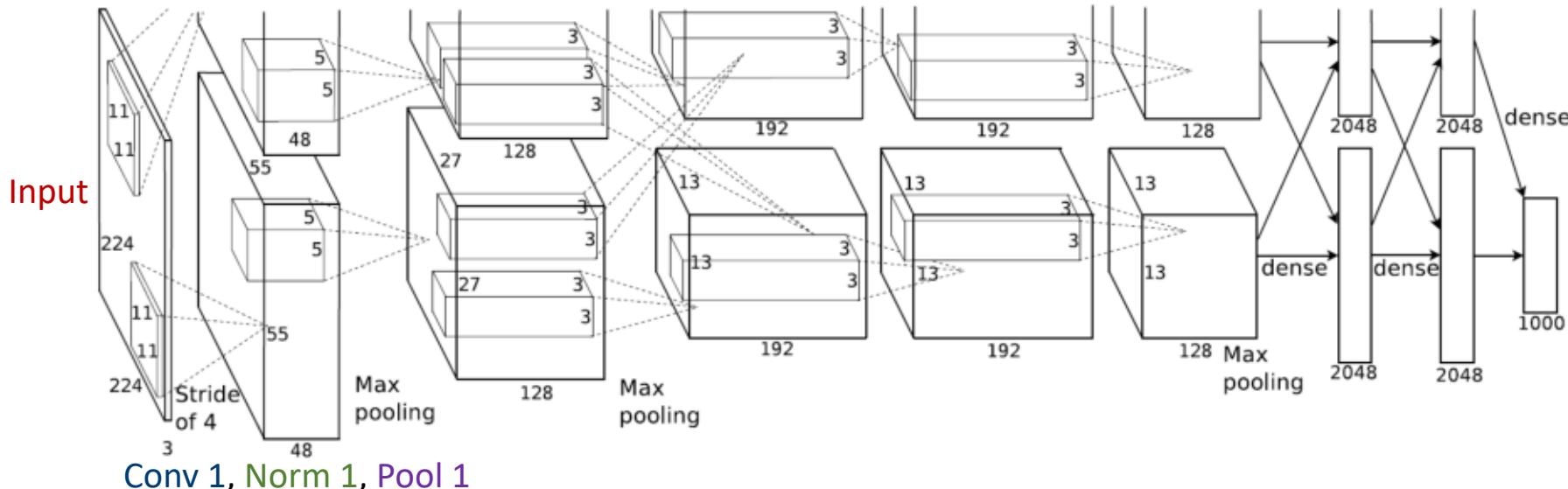
- 11×11 convolution, stride of 4, 96 filters
 - Output feature map: $55 \times 55 \times 96$

- **Layer 2 (Norm 1)**

- Local response normalisation and using ReLU as activation function

- **Layer 3 (Pool 1)**

- 3×3 max pooling window, stride of 2
 - Output feature map: $27 \times 27 \times 96$



AlexNet

[224x224x3] **Input**

[55x55x96] **Conv1**, 11x11, 96, s=4

[55x55x96] **Norm1**

[27x27x96] **Pool1**

[27x27x256] **Conv2**, 5x5, 256

[27x27x256] **Norm2**

[13x13x256] **Pool2**

[13x13x384] **Conv3**, 3x3, 384

[13x13x384] **Conv4**, 3x3, 384

[13x13x256] **Conv5**, 3x3, 256

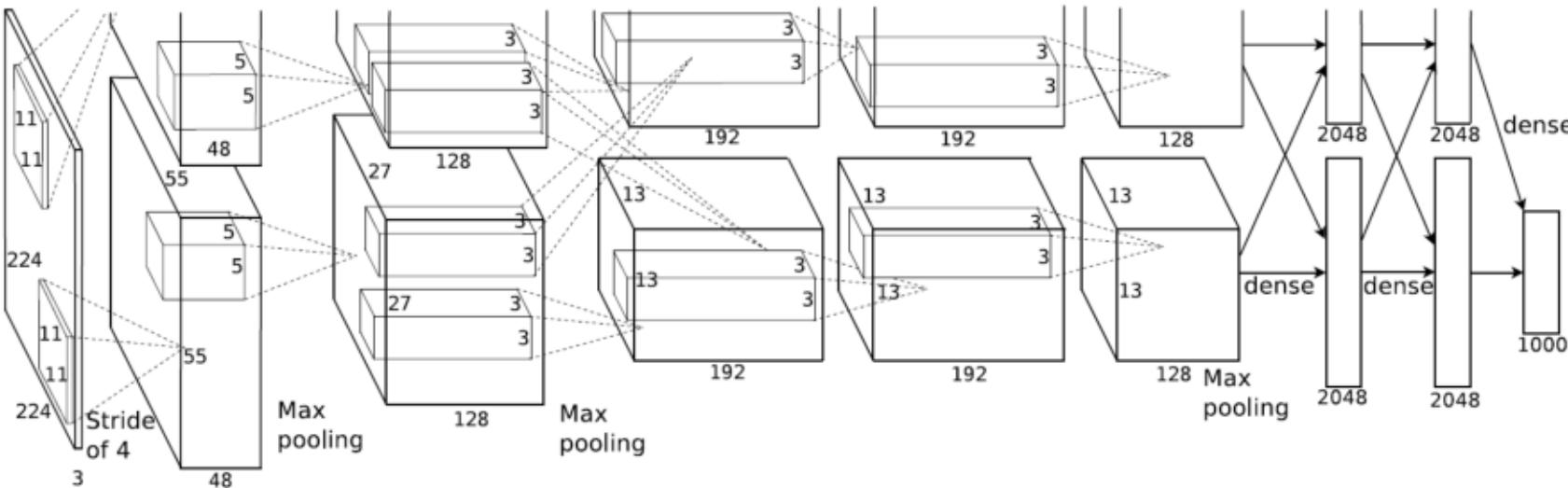
[13x13x256] **Norm3**

[6x6x256] **Pool3**

[4096] **FC1**

[4096] **FC2**

[1000] **FC3** (class score)



AlexNet

- ReLU activation function
 - Previously, most people use sigmoid or tanh functions.
- Data augmentation
 - Randomly crop 224x224 regions from 256x256 original images
 - Horizontal reflection
 - Perturb RGB intensities
- Training on multiple GPUs
 - 2 GTX 580 GPUs, each with only 3GB of memory.
 - Spread the feature maps onto 2 GPUs.

AlexNet

- The main architecture is similar to LeNet.
 - It consists of a number of convolutional layers and pooling layers, followed by fully connected layers at the end.
- But it is deeper.
 - 8 weight layers (with trainable parameters): 5 Conv layers + 3 FC layers
 - Therefore it has more parameters and capacity to learn complicated and non-linear functions.
- It became the first neural network to win the ImageNet classification challenge.
- Research in the following years explores deeper networks and different network architectures.

Summary

- Convolutional neural networks
 - Building blocks
- Examples
 - LeNet-5 (1998)
 - AlexNet (2012)

References

- Ch. 9, Convolutional Networks. Ian Goodfellow et al. Deep Learning (<https://www.deeplearningbook.org/>).

Image Classification IV

Dr Wenjia Bai

Department of Computing & Brain Sciences

Outline

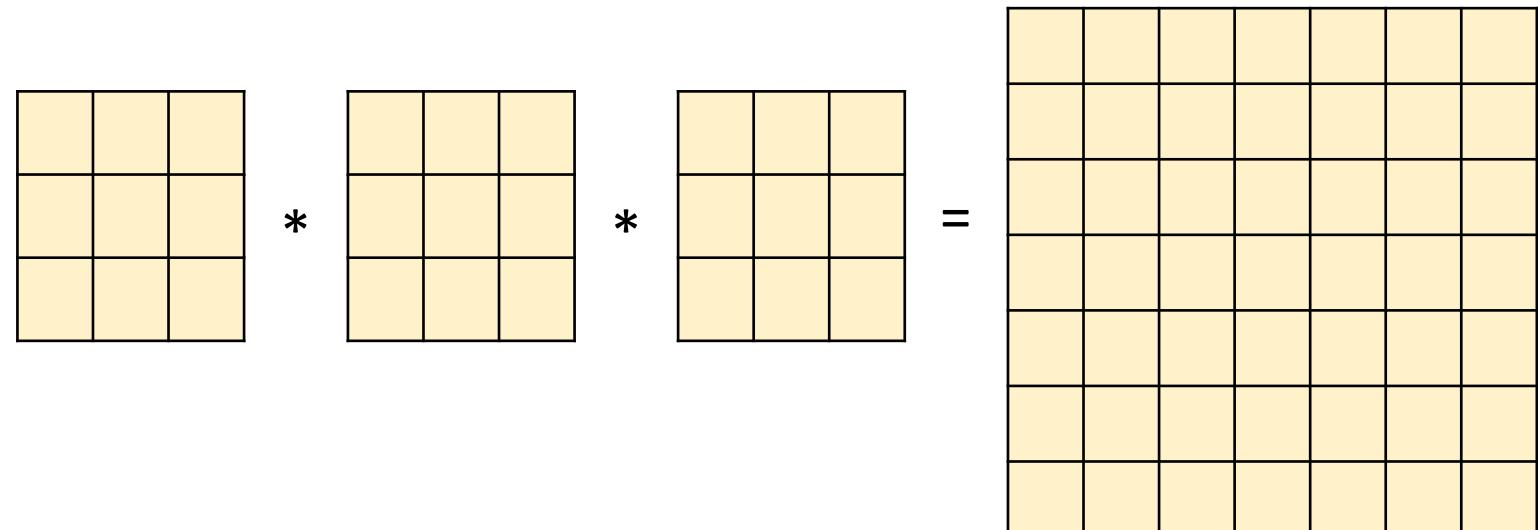
- Recent advances
 - VGG (2014)
 - ResNet (2015)
 - Vision Transformer (2021)
 - ConvNeXt (2022)

VGG

- Visual Geometry Group (VGG) at Oxford
- AlexNet shows the potential of deep neural networks. Why not go deeper?
- The question is how. How do we design the network architecture?

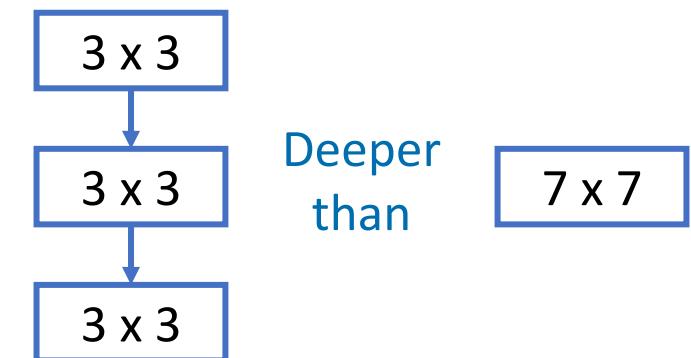
VGG

- While previous research uses 11x11, 7x7 or 5x5 convolutional kernels, VGG only uses 3x3 kernels.
- Convolution of small kernels is equivalent to a large kernel.
 - For example, convolution of three 3x3 kernels is equivalent to a 7x7 kernel.
 - It saves the number of parameters, e.g. $3 \times (3 \times 3) = 27$ vs $7 \times 7 = 49$.



VGG

- While previous research uses 11x11, 7x7 or 5x5 convolutional kernels, VGG only uses 3x3 kernels.
- Convolution of small kernels is equivalent to a large kernel.
 - For example, convolution of three 3x3 kernels is equivalent to a 7x7 kernel.
 - It saves the number of parameters, e.g. $3 \times (3 \times 3) = 27$ vs $7 \times 7 = 49$.
 - It increases the depth of the network and non-linearity.



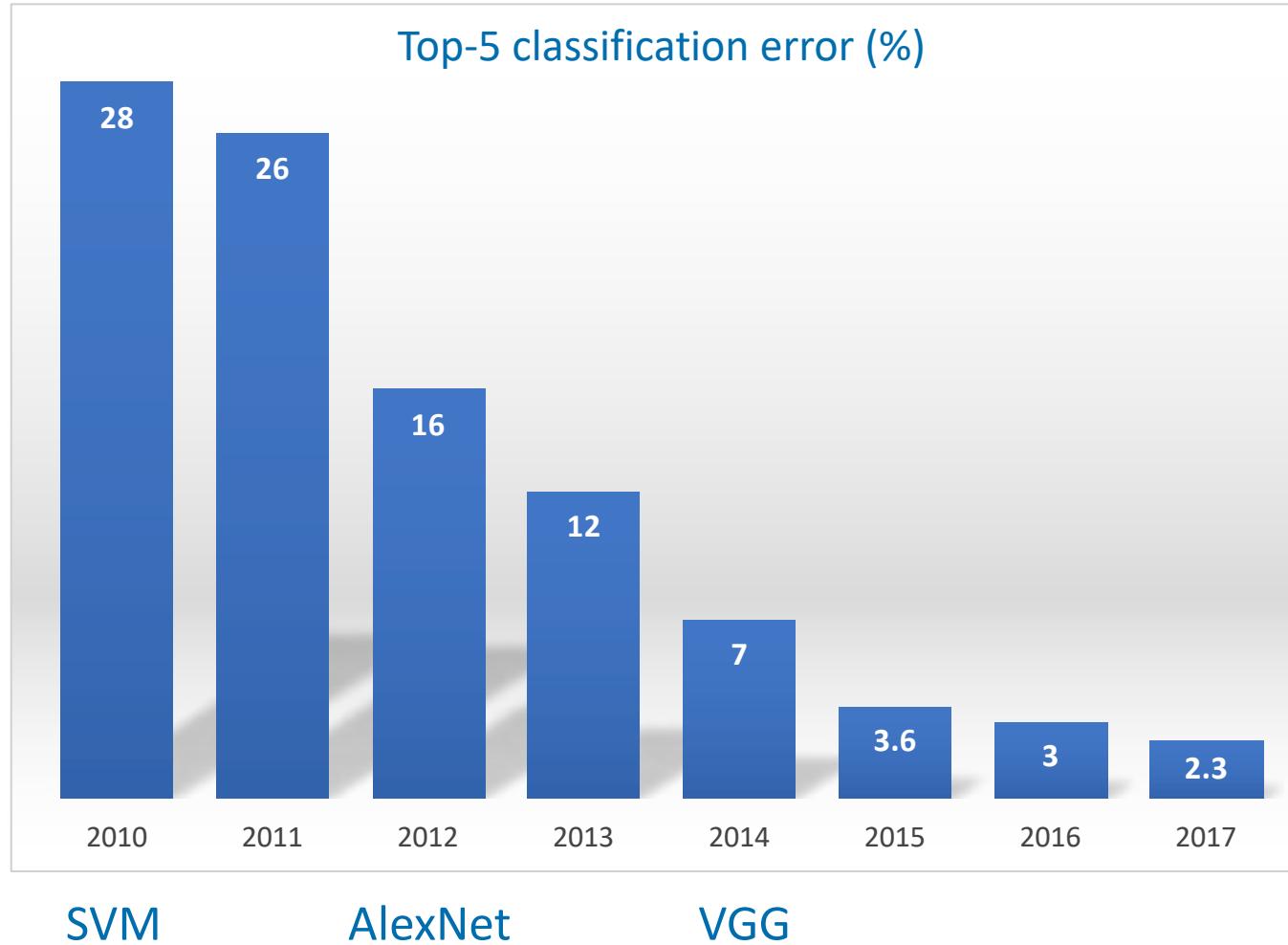
VGG-16

[224x224x3] **Input**
[224x224x64] 3x3 conv1, 64
[224x224x64] 3x3 conv1, 64
[112x112x64] **Pool**
[112x112x128] 3x3 conv2, 128
[112x112x128] 3x3 conv2, 128
[56x56x128] **Pool**
[56x56x256] 3x3 conv3, 256
[56x56x256] 3x3 conv3, 256
[56x56x256] 3x3 conv3, 256
[28x28x256] **Pool**
[28x28x512] 3x3 conv4, 512
[28x28x512] 3x3 conv4, 512
[28x28x512] 3x3 conv4, 512
[14x14x512] **Pool**
[14x14x512] 3x3 conv5, 512
[14x14x512] 3x3 conv5, 512
[14x14x512] 3x3 conv5, 512
[7x7x512] **Pool**
[4096] **FC**, 4096
[4096] **FC**, 4096
[1000] **FC**, 1000

VGG-19

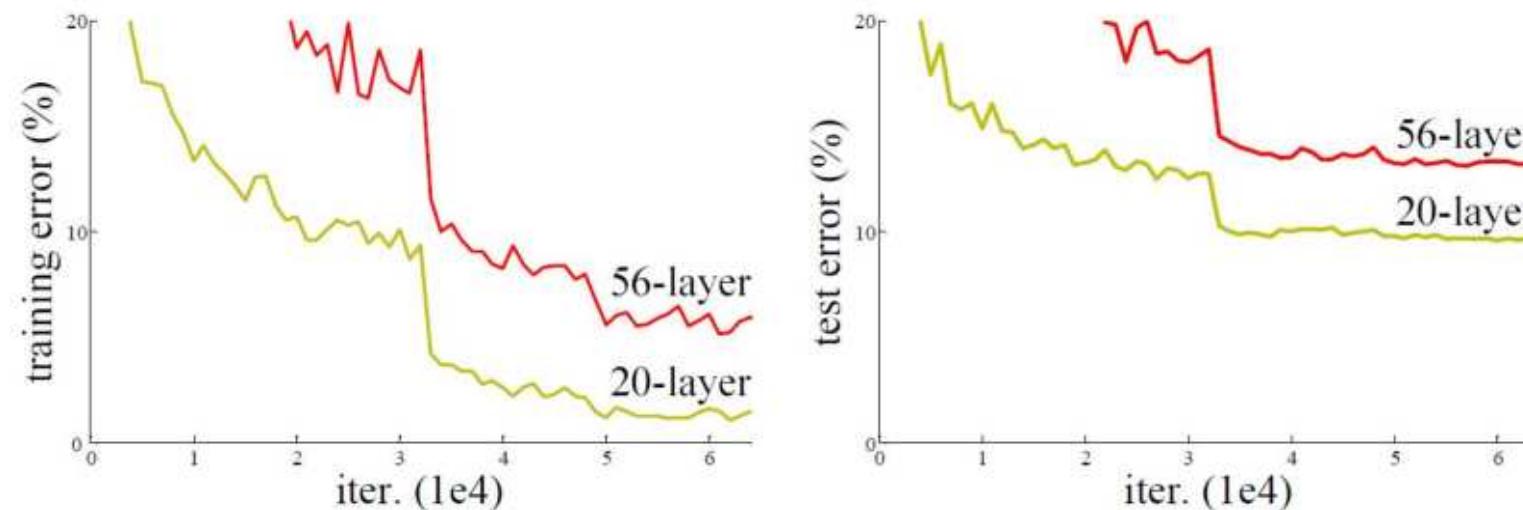
[224x224x3] **Input**
[224x224x64] 3x3 conv1, 64
[224x224x64] 3x3 conv1, 64
[112x112x64] **Pool**
[112x112x128] 3x3 conv2, 128
[112x112x128] 3x3 conv2, 128
[56x56x128] **Pool**
[56x56x256] 3x3 conv3, 256
[56x56x256] 3x3 conv3, 256
[56x56x256] 3x3 conv3, 256
[28x28x256] **Pool**
[28x28x512] 3x3 conv4, 512
[28x28x512] 3x3 conv4, 512
[28x28x512] 3x3 conv4, 512
[14x14x512] **Pool**
[14x14x512] 3x3 conv5, 512
[14x14x512] 3x3 conv5, 512
[14x14x512] 3x3 conv5, 512
[7x7x512] **Pool**
[4096] **FC**, 4096
[4096] **FC**, 4096
[1000] **FC**, 1000

ImageNet classification challenge

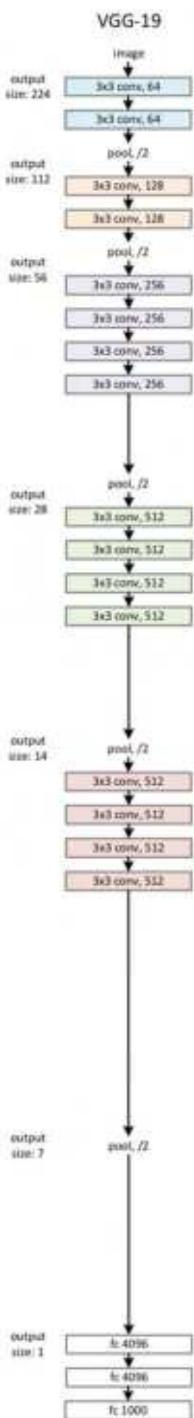


ResNet

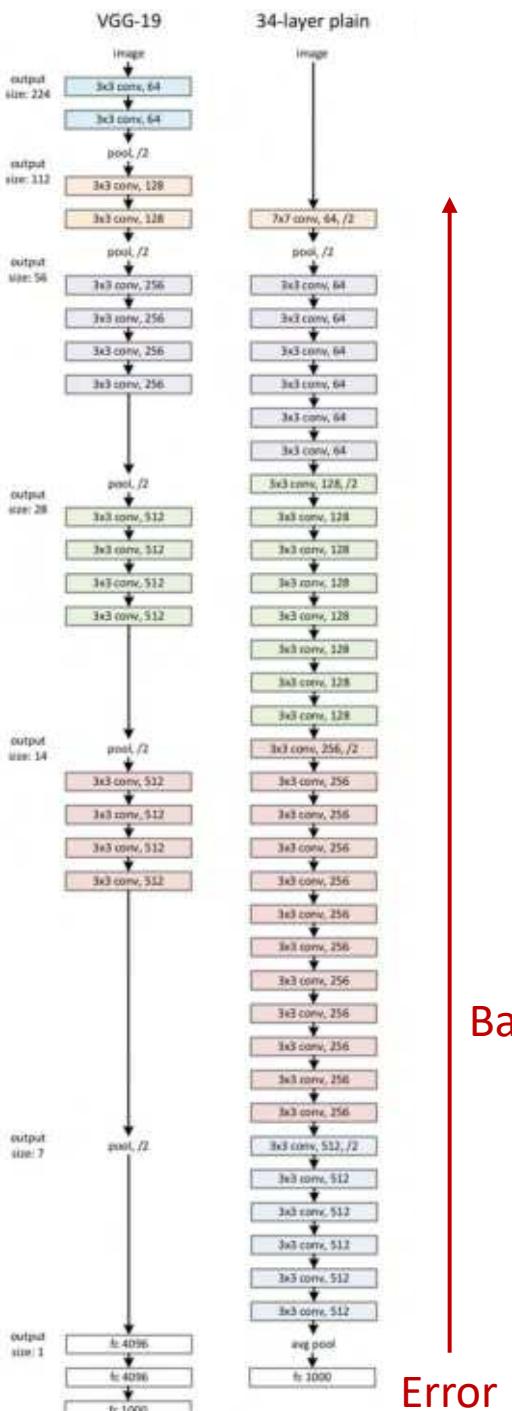
- VGG net outperforms AlexNet due to a deeper network architecture.
- Can we go even deeper?
 - It is not easy. If we simply stack more 3x3 convolutional layers and train the network, it is not performing better.



Training and test errors for a 56-layer network compared to a 20-layer network.



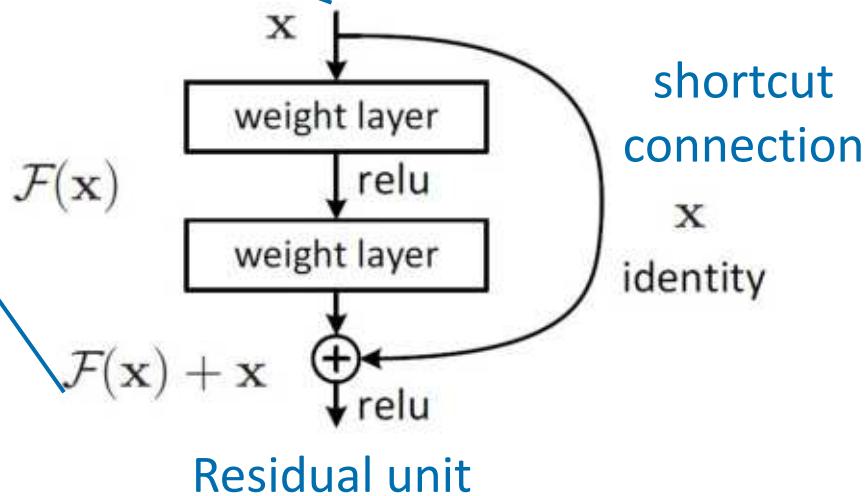
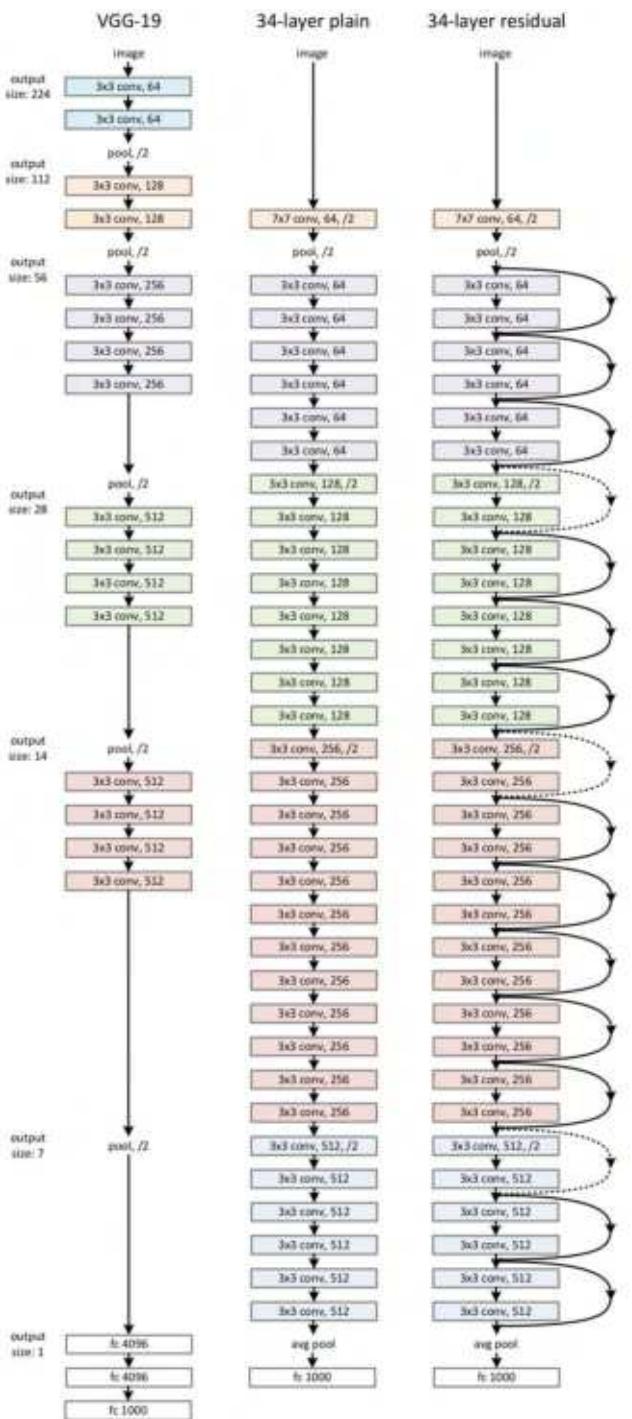
Let us stack more convolutional layers to make this network deeper.



However, training a deeper network becomes difficult. It is not necessarily performing better.

Backpropagation

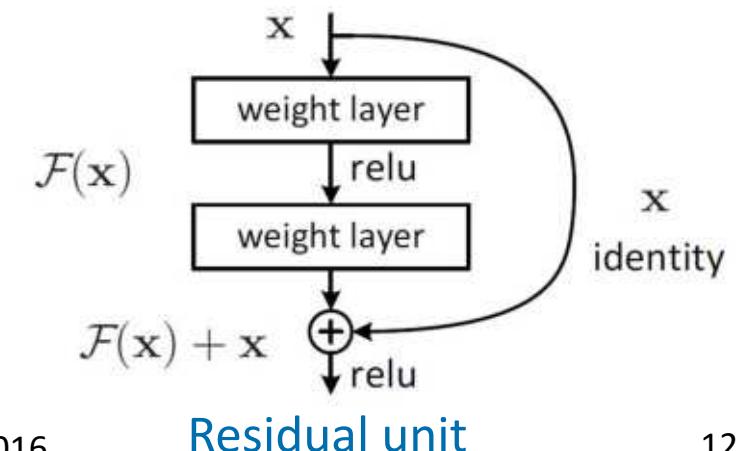
Training is based on **error backpropagation**, layer by layer. It is difficult for error to propagate to layers far away from output and update weights there.



Residual network (ResNet) proposes to add **shortcut connections** to enable the flow of gradients.

Rational behind residual networks

- If weights in the residual unit are zero, it is a identity mapping.
- If we form a deep network by stacking some residual units onto a shallow network, at least it will perform as well as the shallow network, as the residual units can be identity mappings.



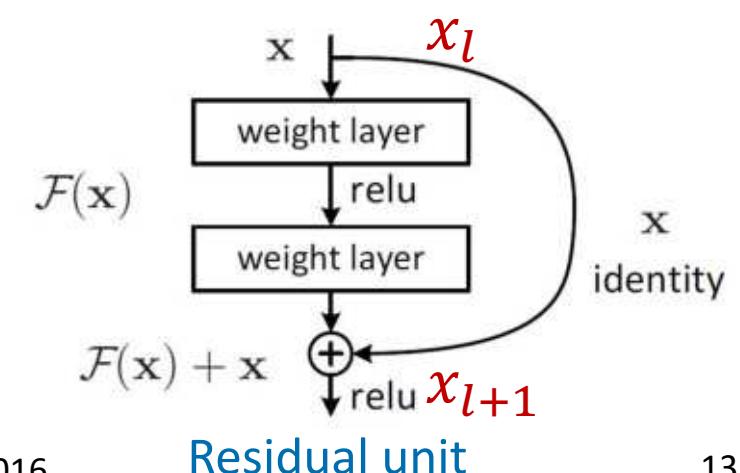
Rational behind residual networks

- Denote the error (loss) as E , the input and output of a residual unit as x_l and x_{l+1} , we have

$$x_{l+1} = x_l + f(x_l, W_l)$$

- When performing backpropagation, we use the chain rule to evaluate the derivative at the previous layer

$$\begin{aligned}\frac{\partial E}{\partial x_l} &= \frac{\partial E}{\partial x_{l+1}} \cdot \frac{\partial x_{l+1}}{\partial x_l} \\ &= \frac{\partial E}{\partial x_{l+1}} \cdot \left(1 + \frac{\partial f(x_l, W_l)}{\partial x_l}\right)\end{aligned}$$



Rational behind residual networks

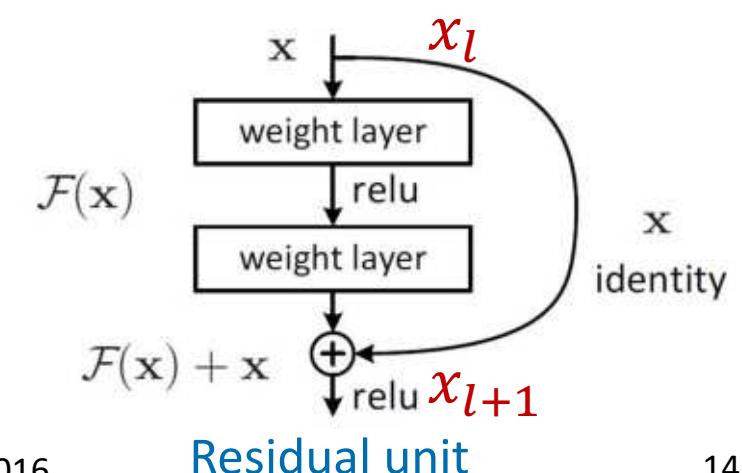
- Denote the error (loss) as E , the input and output of a residual unit as x_l and x_{l+1} , we have

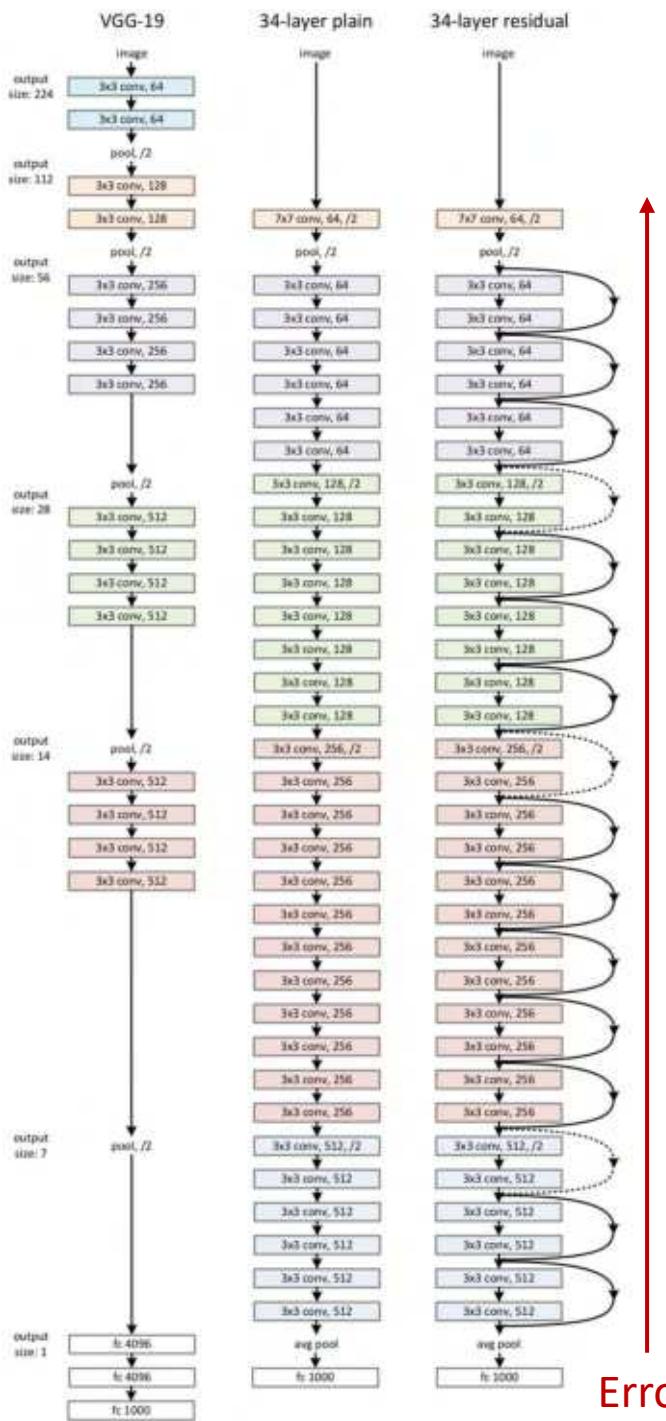
$$x_{l+1} = x_l + f(x_l, W_l)$$

- When performing backpropagation, we use the chain rule to evaluate the derivative at the previous layer

$$\begin{aligned}\frac{\partial E}{\partial x_l} &= \frac{\partial E}{\partial x_{l+1}} \cdot \frac{\partial x_{l+1}}{\partial x_l} \\ &= \frac{\partial E}{\partial x_{l+1}} \cdot \left(1 + \frac{\partial f(x_l, W_l)}{\partial x_l}\right)\end{aligned}$$

Shortcut connection enables the flow of loss gradient.





Error

Error can be backpropagated easily thanks to shortcut connections.

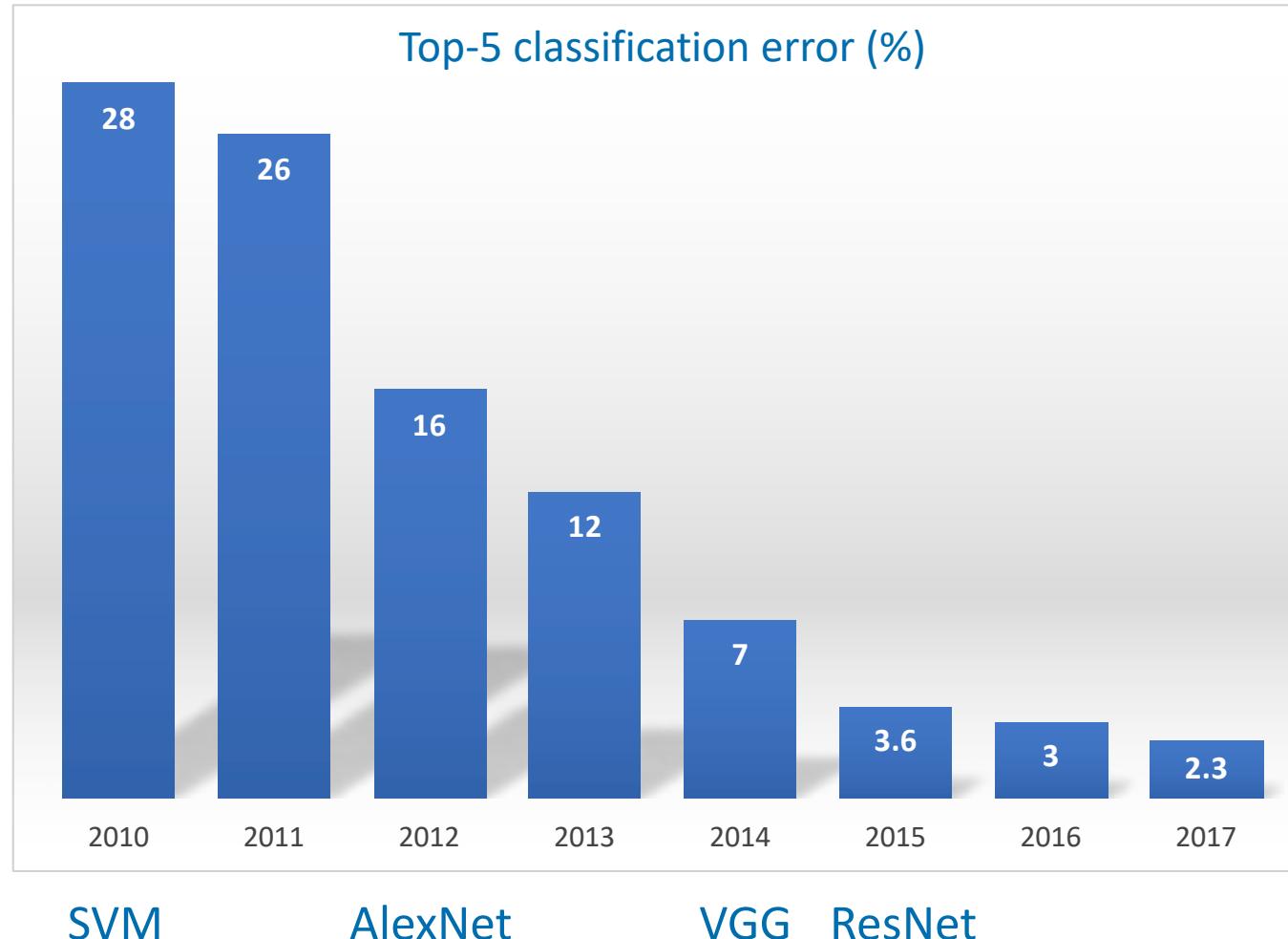
ResNet

- Using the residual units, we can build and train very deep neural networks.

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

Performance on ImageNet validation set.

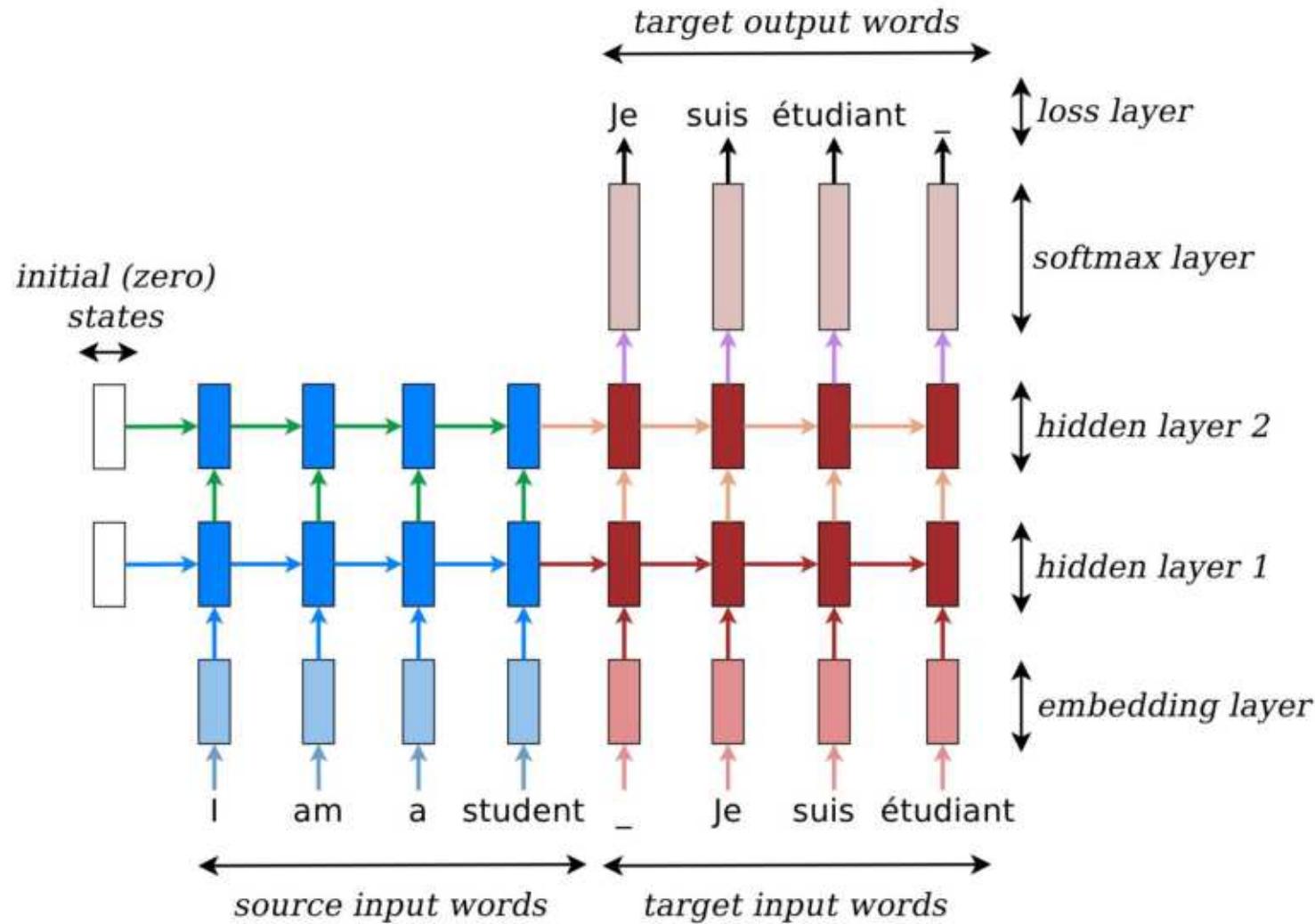
ImageNet classification challenge



Vision Transformer

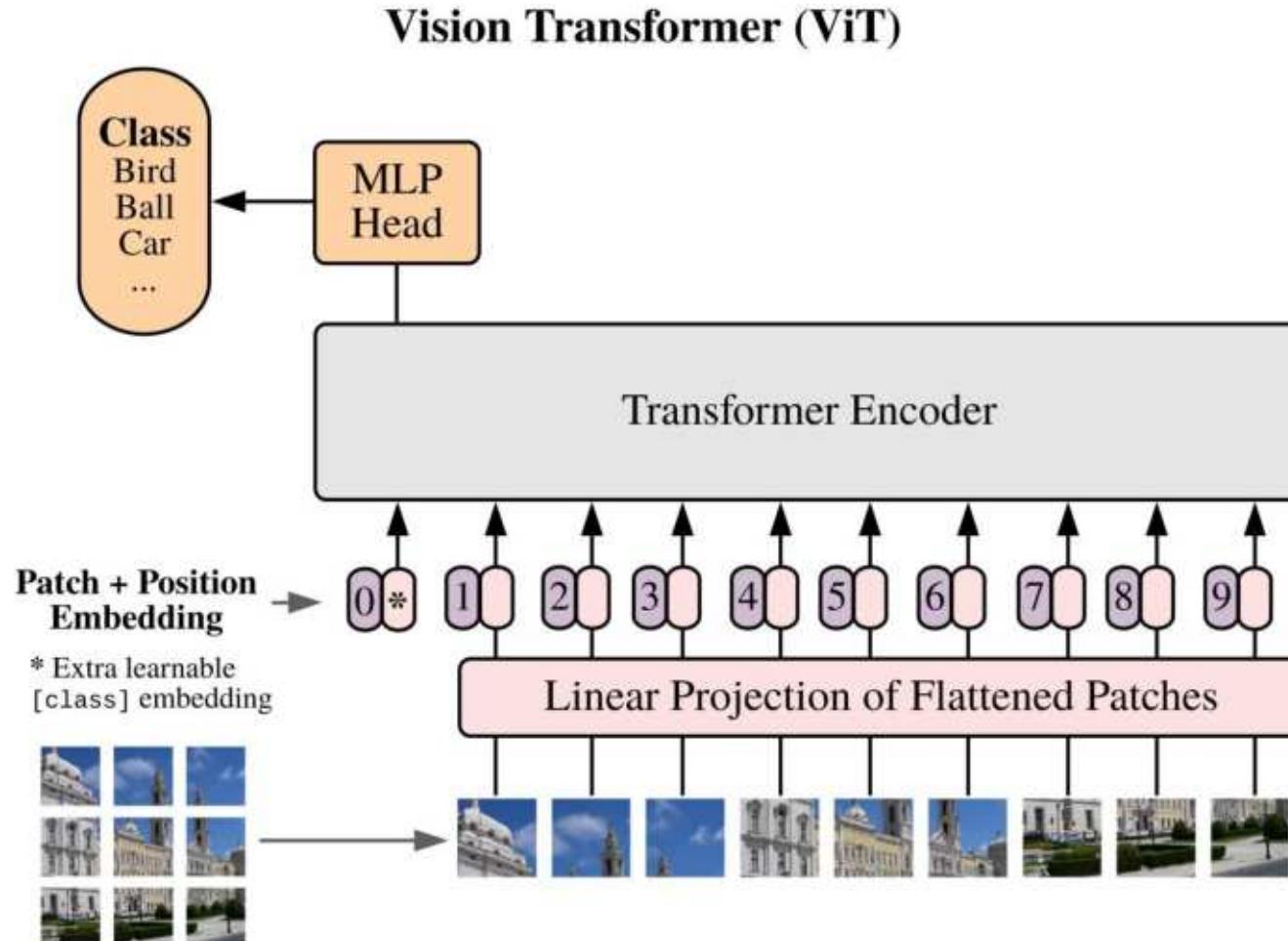
- While deep learning has revolutionised computer vision (CV), it has also transformed natural language processing (NLP).
- The dominant architecture in NLP since 2017 is the Transformer, which is able to model long-range dependencies of words in a sentence.
- Although originally developed for language understanding, it has shown an outstanding performance for image understanding, even images have a quite different format from language (pixels vs words).

Natural language processing



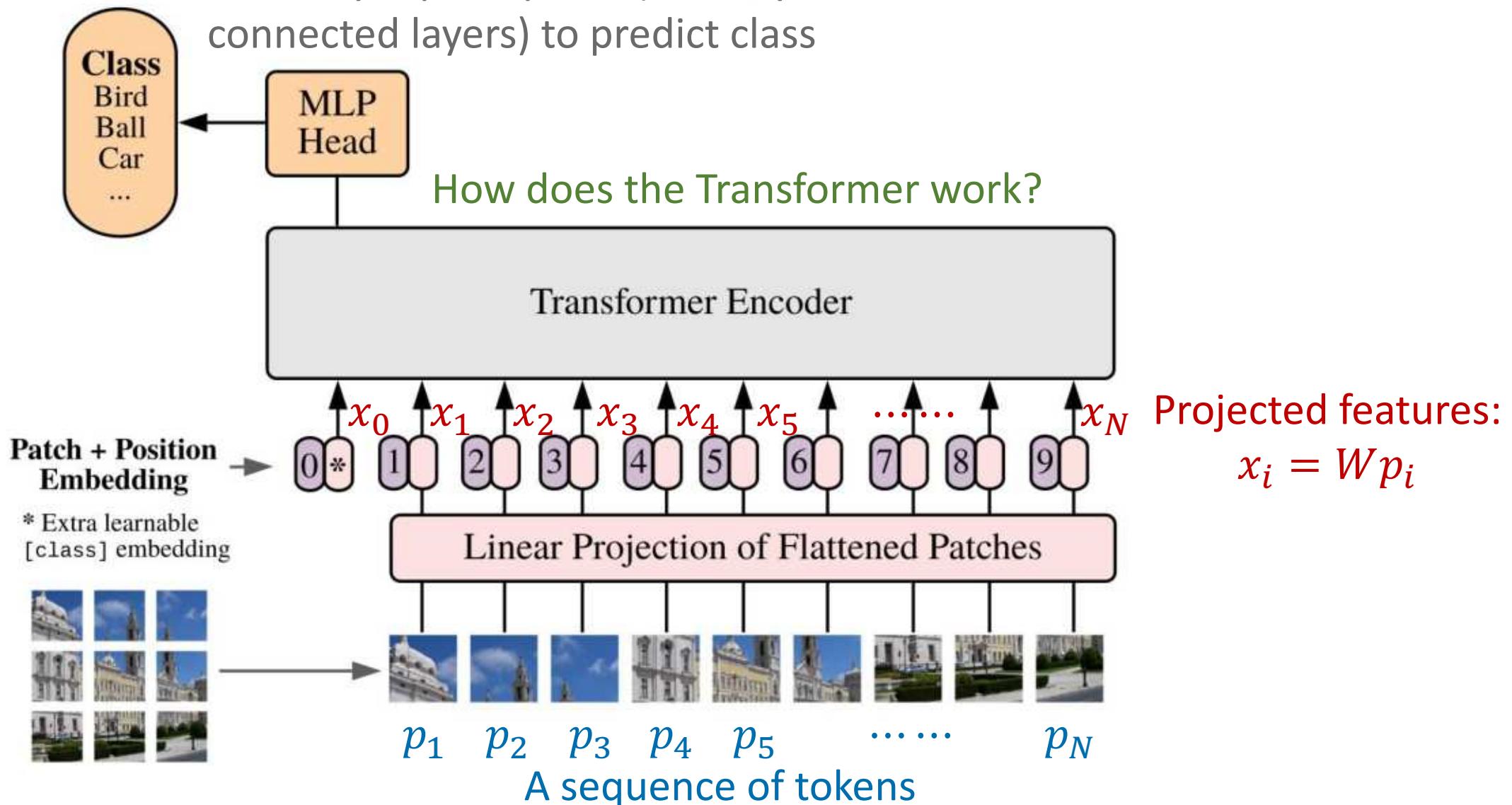
In natural language processing, each word is represented as a vector and input to the network.

Vision Transformer (ViT)



In vision transformer, an image is split into patches.
Each patch is processed similar to a word (token) in NLP.

Multi-layer perceptron (i.e. fully connected layers) to predict class

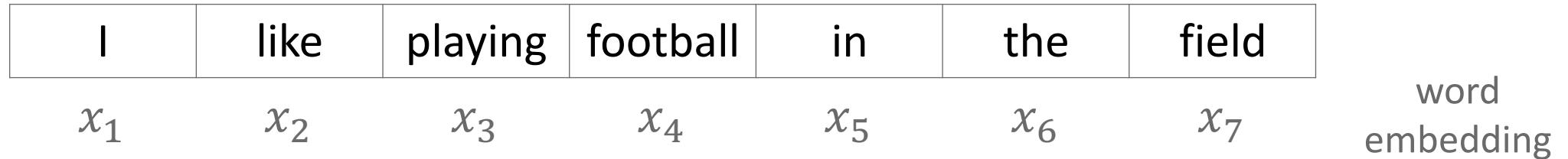


Transformer

- CNN consists of a number of convolutional layers.
- The transformer consists of multi-head self-attention (MSA) layers.
- What is self attention?
 - It is a way to model the dependencies between tokens.

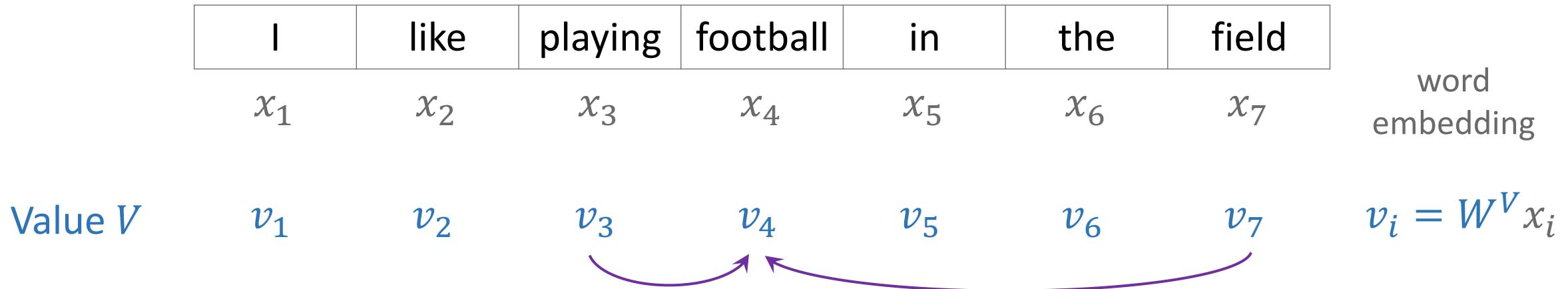
Self-attention

- Suppose we want to understand a sentence,



Self-attention

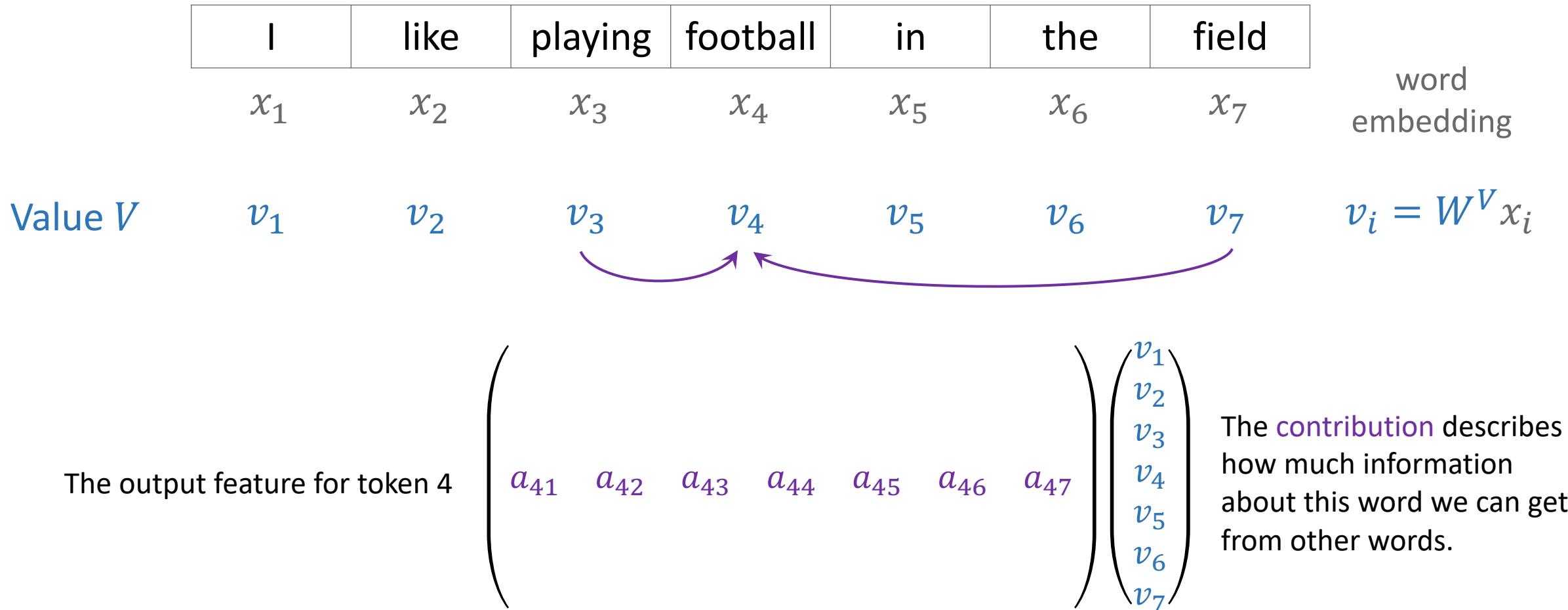
- Let us learn or process features for each token.



These words are not independent. When we analyse “football”, “playing” and “field” can also contribute information.

Self-attention

- Let us learn or process features for each token.



Self-attention

- The relationship between two words is modelled by query and key.

	I	like	playing	football	in	the	field	word embedding
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
Value V	v_1	v_2	v_3	v_4	v_5	v_6	v_7	$v_i = W^V x_i$
Query Q	q_1	q_2	q_3	q_4	q_5	q_6	q_7	$q_i = W^Q x_i$
Key K	k_1	k_2	k_3	k_4	k_5	k_6	k_7	$k_i = W^K x_i$

The inner product between q_i and k_j , $\langle q_i, k_j \rangle$ describes the contribution from j to i .
The contribution is not symmetric, i.e. $\langle q_i, k_j \rangle$ may not equal $\langle q_j, k_i \rangle$.

Self-attention

- The output of the attention function is formulated as,

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q} \mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

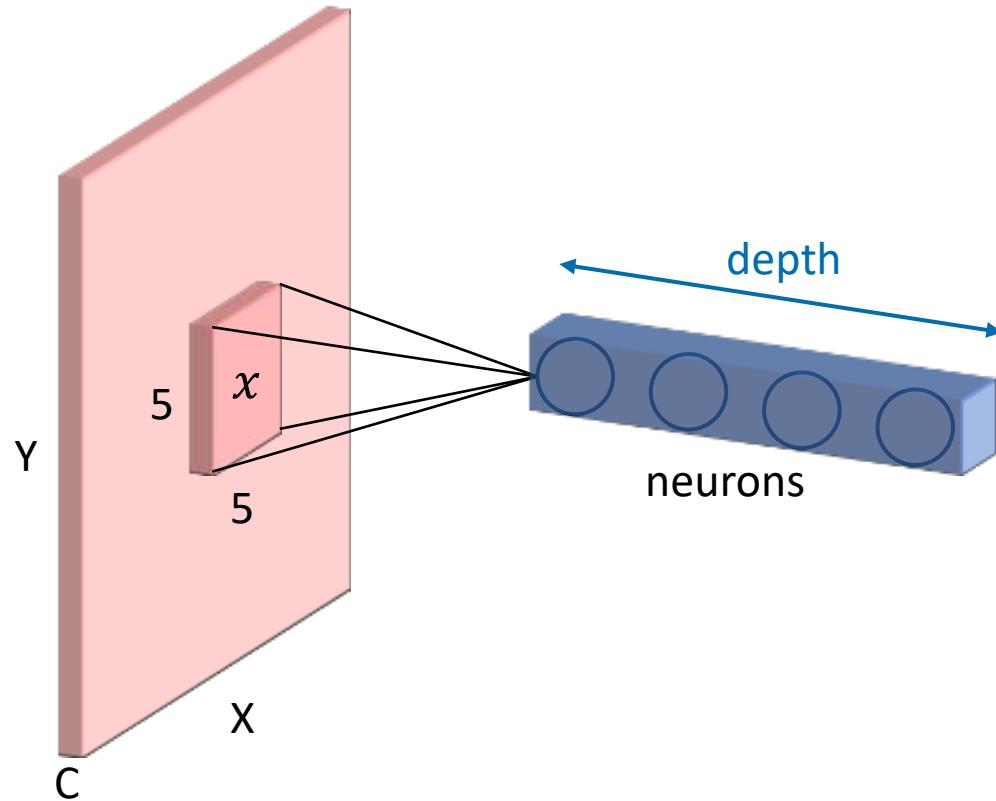
dot product between
query and key

|

|

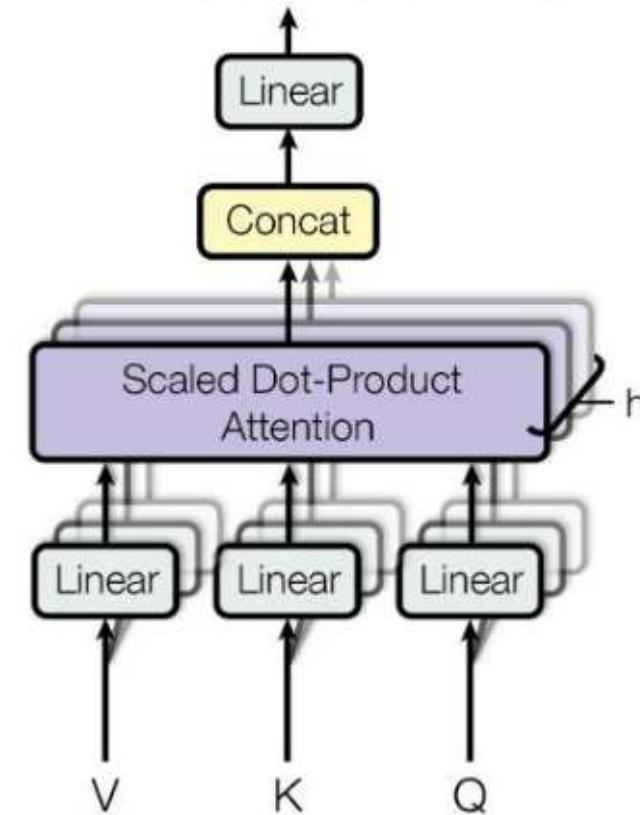
scaling factor, where d_k denotes
the dimension of key vectors k_i

Multi-head self-attention



In CNN, a convolutional layer has multiple neurons, each having its own weights W .

Multi-Head Attention



An attention layer can have multiple heads, each having its own weights W .

Multi-head self-attention

- The multi-head self-attention (MSA) is formulated as,

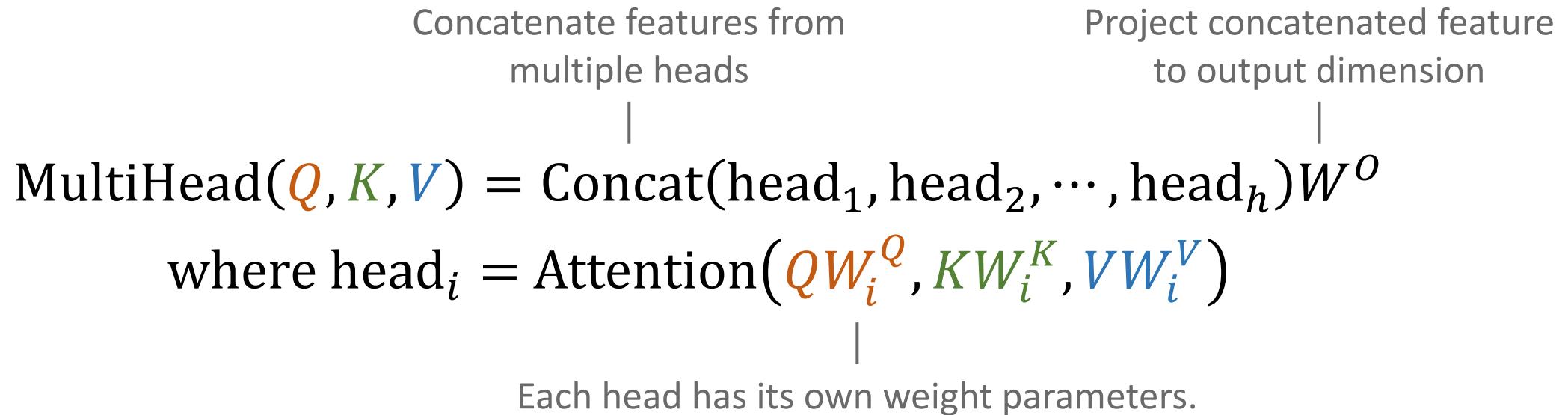
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^O$$

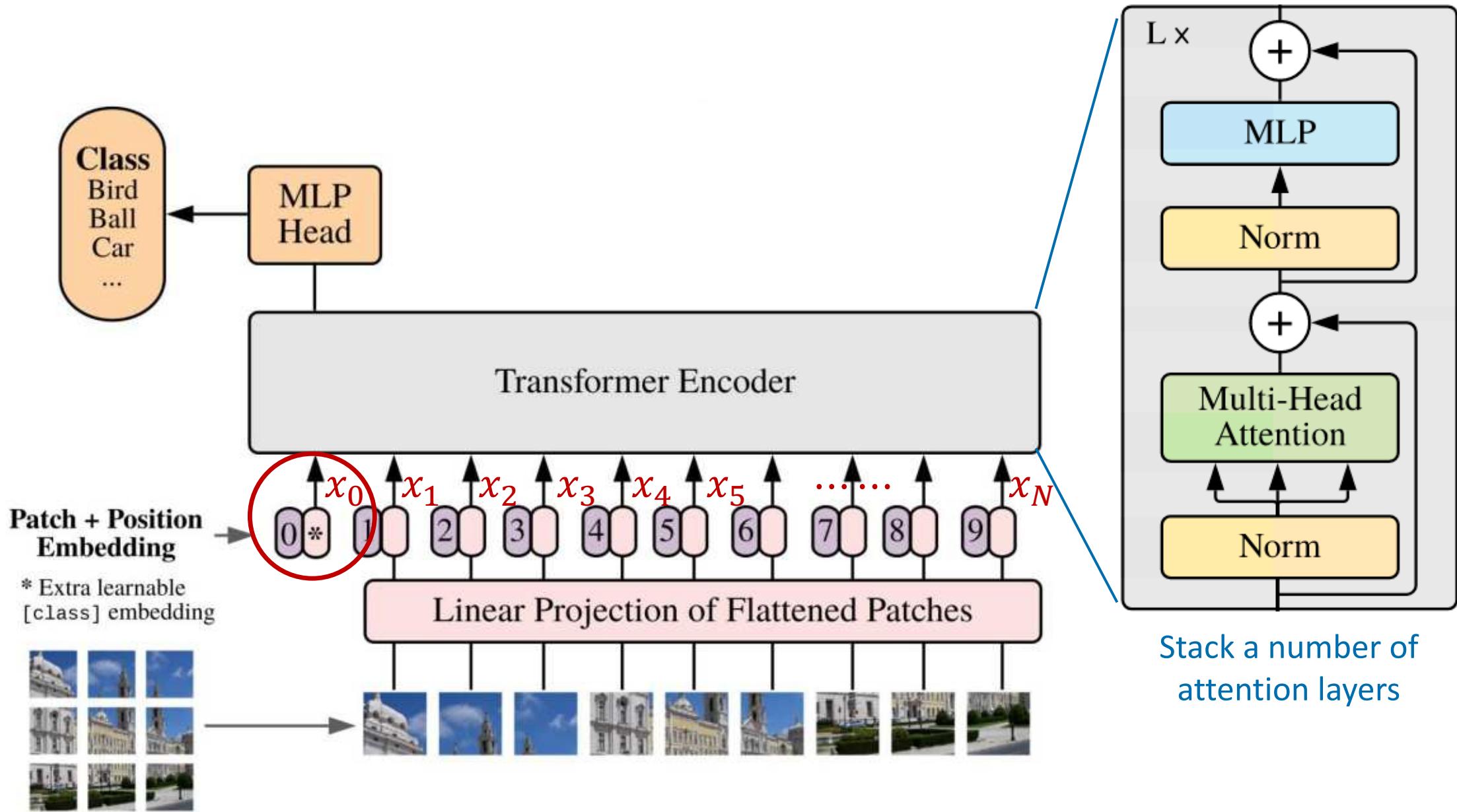
where $\text{head}_i = \text{Attention}(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V)$

Concatenate features from multiple heads

Project concatenated feature to output dimension

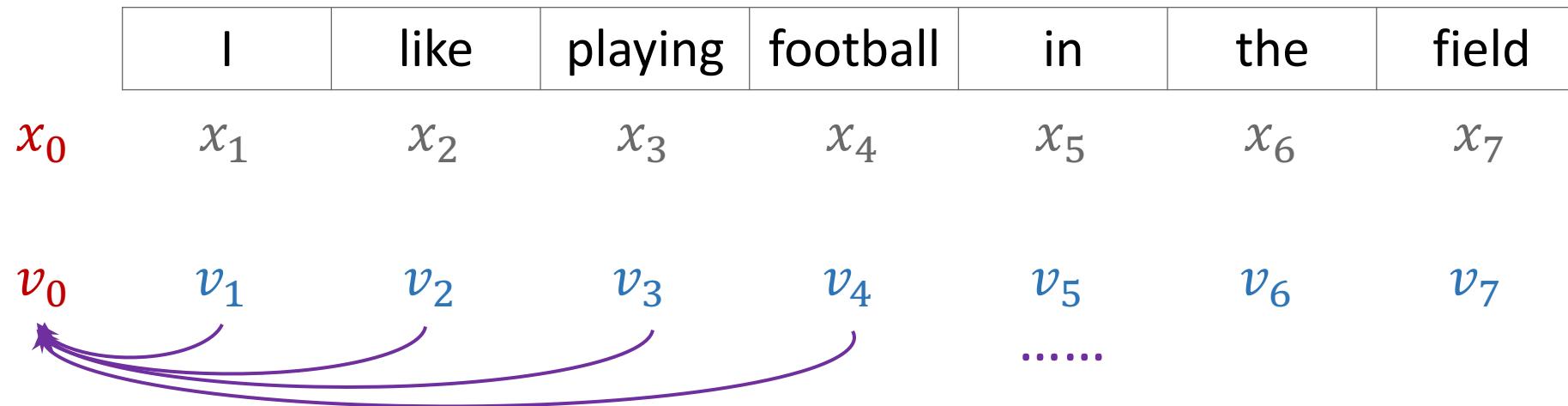
Each head has its own weight parameters.





Self-attention

- An extra token learns the representation for the sentence or image.



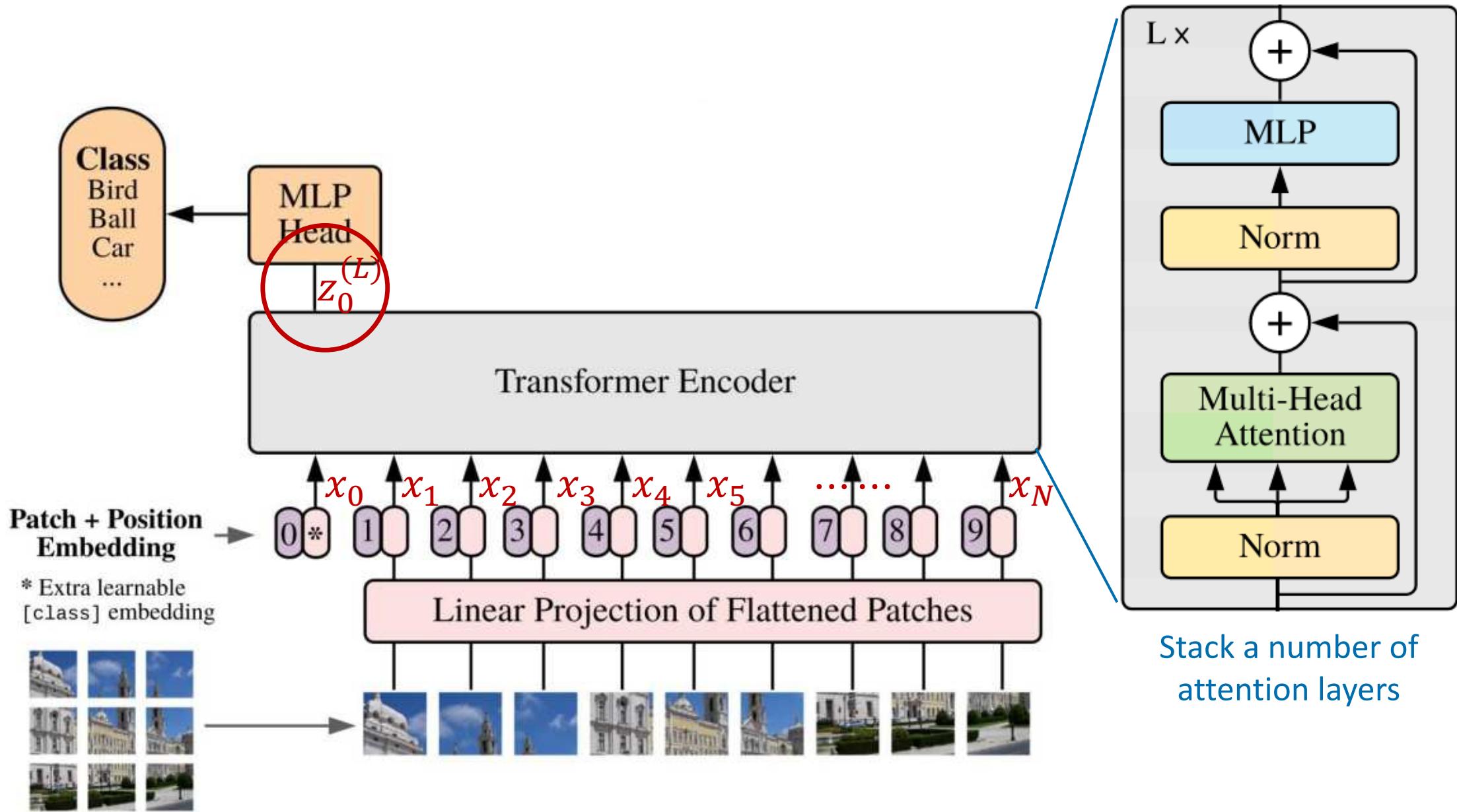


Image classification performance

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.55 \pm 0.04	87.76 \pm 0.03	85.30 \pm 0.02	87.54 \pm 0.02	88.4/88.5*
ImageNet ReaL	90.72 \pm 0.05	90.54 \pm 0.03	88.62 \pm 0.05	90.54	90.55
CIFAR-10	99.50 \pm 0.06	99.42 \pm 0.03	99.15 \pm 0.03	99.37 \pm 0.06	—
CIFAR-100	94.55 \pm 0.04	93.90 \pm 0.05	93.25 \pm 0.05	93.51 \pm 0.08	—
Oxford-IIIT Pets	97.56 \pm 0.03	97.32 \pm 0.11	94.67 \pm 0.15	96.62 \pm 0.23	—
Oxford Flowers-102	99.68 \pm 0.02	99.74 \pm 0.00	99.61 \pm 0.02	99.63 \pm 0.03	—
VTAB (19 tasks)	77.63 \pm 0.23	76.28 \pm 0.46	72.72 \pm 0.21	76.29 \pm 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

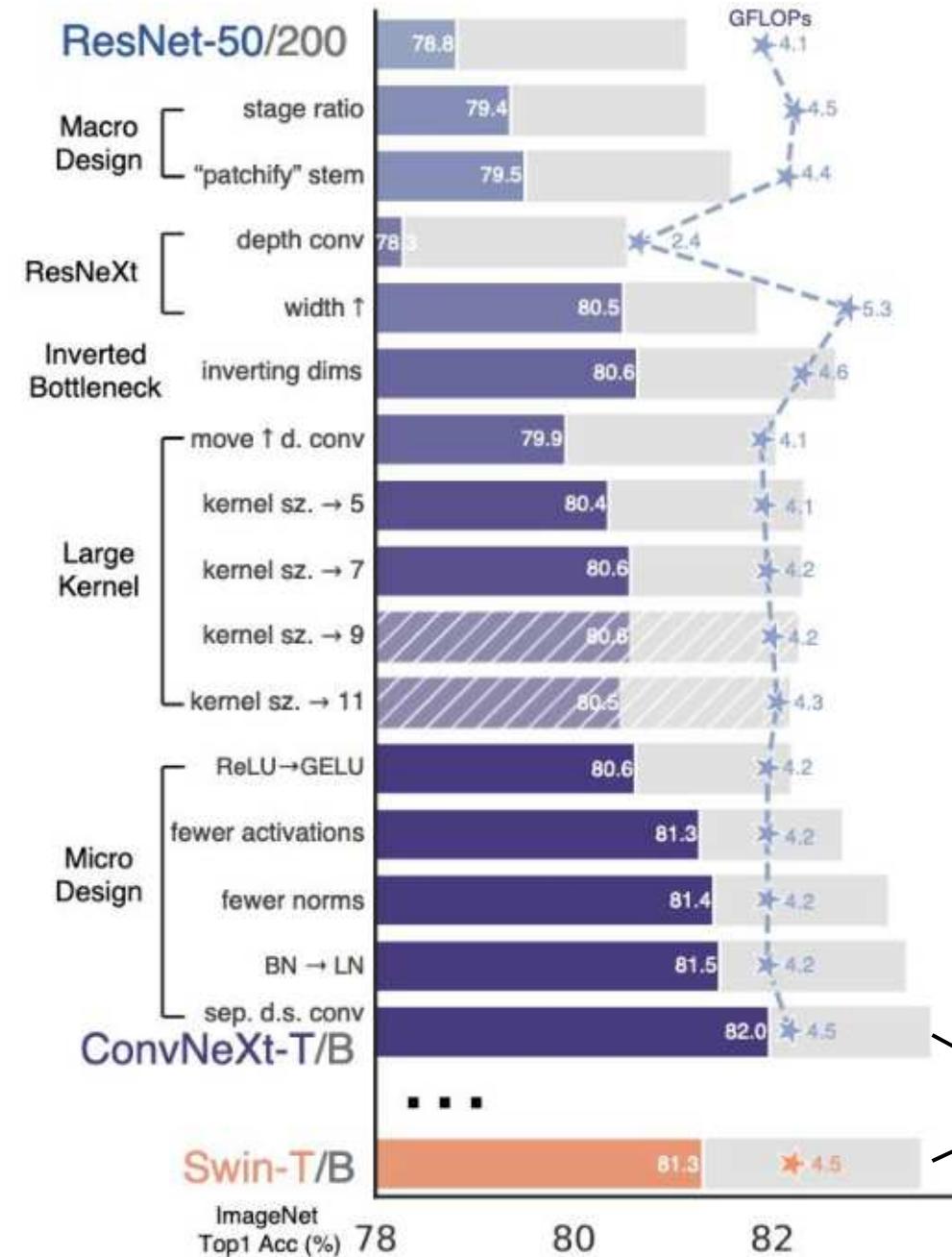
Vision Transformer demonstrates a comparable or better performance, compared to CNNs.
Top-1 classification accuracy is reported here, instead of top-5 error rate.

ConvNeXt

- While Transformers seemed to supercede ConvNet in 2021, would it be possible to innovate and improve ConvNet?
- Yes, it is possible by carefully engineering key components of a ConvNet.
 - Convolution kernel size
 - Number of feature channels
 - Activation function
 - Normalisation layer
 - ...
- The architecture is named as ConvNeXt.

Improve CNN performance
by carefully engineering
each component

Transformer performance



Comparable
performance
at the end.

Some thoughts

- Shall we only teach CNN for image classification?
 - Perhaps not.
 - There can be multiple paradigms achieving good classification performance, as long as they learn good discriminative features for images.
 - It would be helpful to understand the history: SVM, CNN, Transformers...

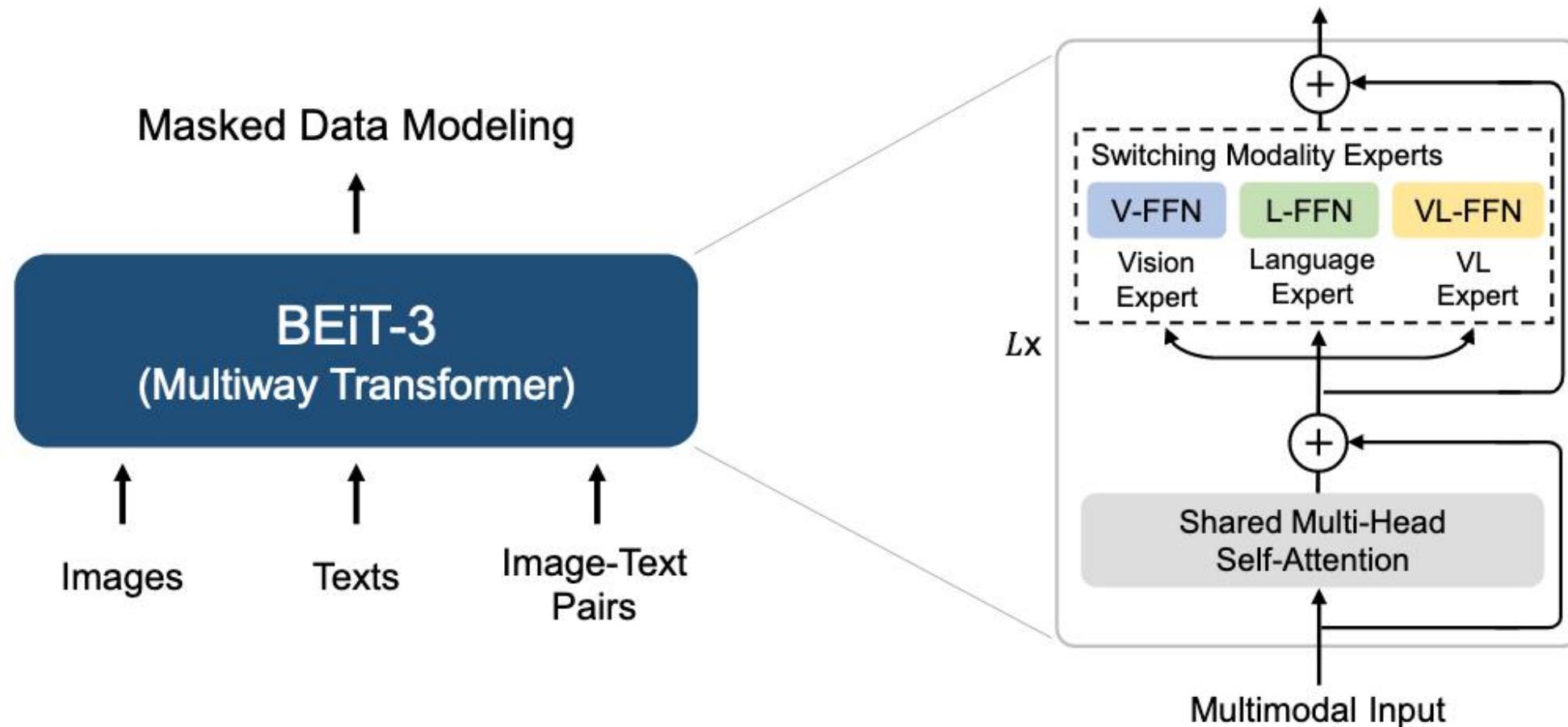
Some thoughts

- Shall we keep on increasing image classification performance in research?
 - Yes and no.
 - Yes. We always want to advance the boundary of human knowledge.
 - No. An almost perfect classification performance does not necessarily mean image understanding. When human understand an image, it is more than classification. We also detect objects, understand their relationships, 3D geometry in the image and conduct reasoning.

Some thoughts

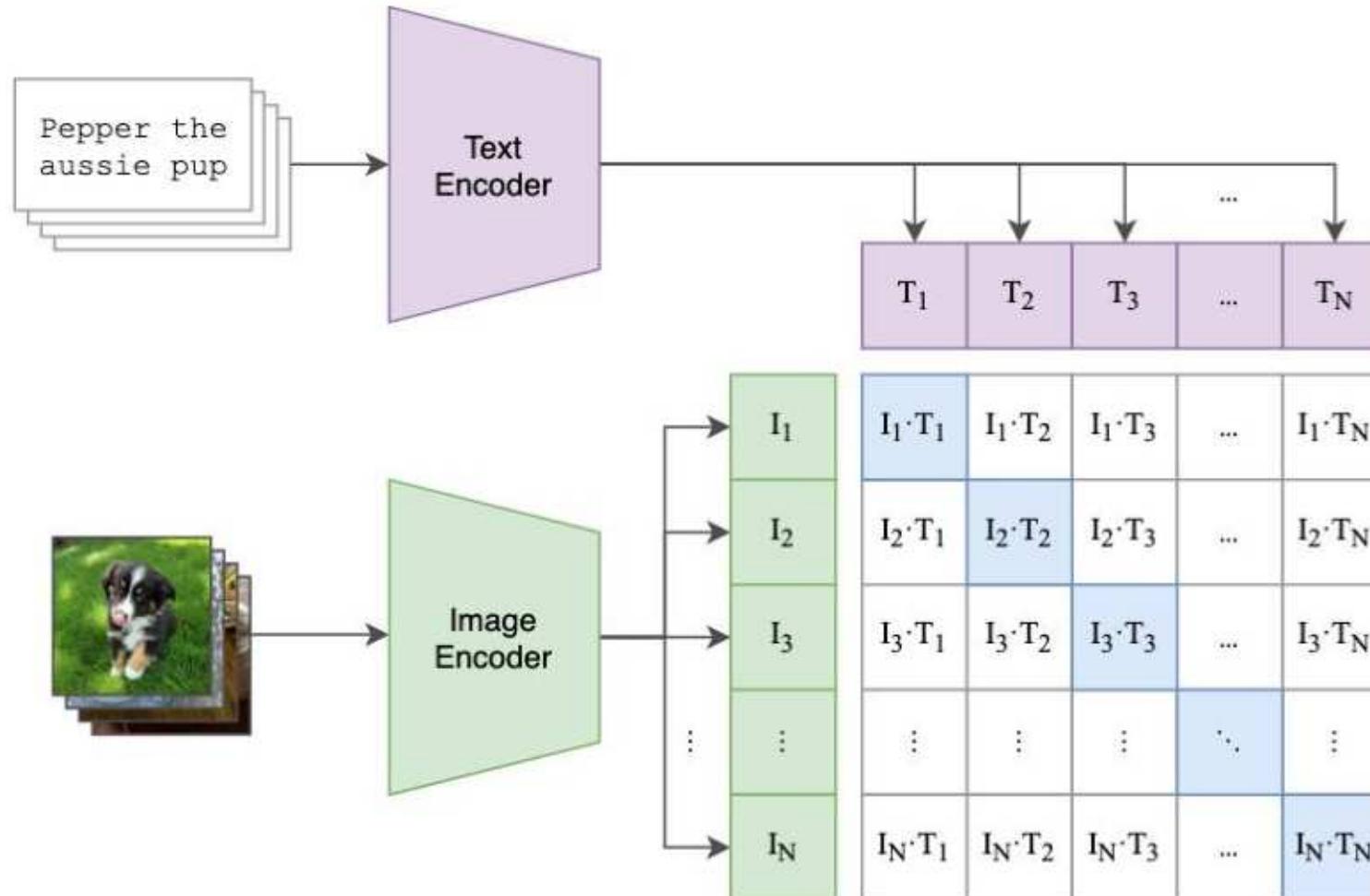
- Why Transformer is interesting?
 - Influence of natural language processing on computer vision.
 - A unified architecture for language, vision and audio.
 - Inspires us to develop new multi-modal learning methods.

Unified learning models



BEiT: A single architecture, the Transformer, can be used for learning from different data modalities.

Vision-language models



Contrastive Language-Image Pre-training (CLIP), which jointly trains an image encoder and a text encoder.

Summary

- Advances in the last decade
 - VGG (2014)
 - ResNet (2015)
 - Vision Transformer (2021)
 - ConvNeXt (2022)

Image Segmentation I

Dr Wenjia Bai

Department of Computing & Brain Sciences

Image segmentation

- Image segmentation is a process of partitioning an image into multiple regions, each region consisting of pixels with similar properties (e.g. intensity, colour, texture) or semantics (e.g. person, car, building).

Image classification



Cat

Probability

Cat: 85%

Tiger: 10%

Dog: 1%

...

Image segmentation



Segmentation
Pixel-wise classification

Cat

Label class for each pixel

1: cat

0: background

Image segmentation

- Segmentation is already a function used in many software, such as in Powerpoint for background removal.



Students enjoying the winter sun by the Queens Tower. ©Imperial College.

Outline

- There are many image segmentation methods.
- We will briefly describe several methods.
- Unsupervised
 - Thresholding
 - K-means clustering
 - Gaussian mixture model
- Supervised
 - Convolutional neural network

Thresholding

- The simplest method for segmentation.
- It converts a grayscale image into a binary label map.
- At each pixel x , the label is defined by,

$$f(x) = \begin{cases} 1, & \text{if } I(x) \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$



Image



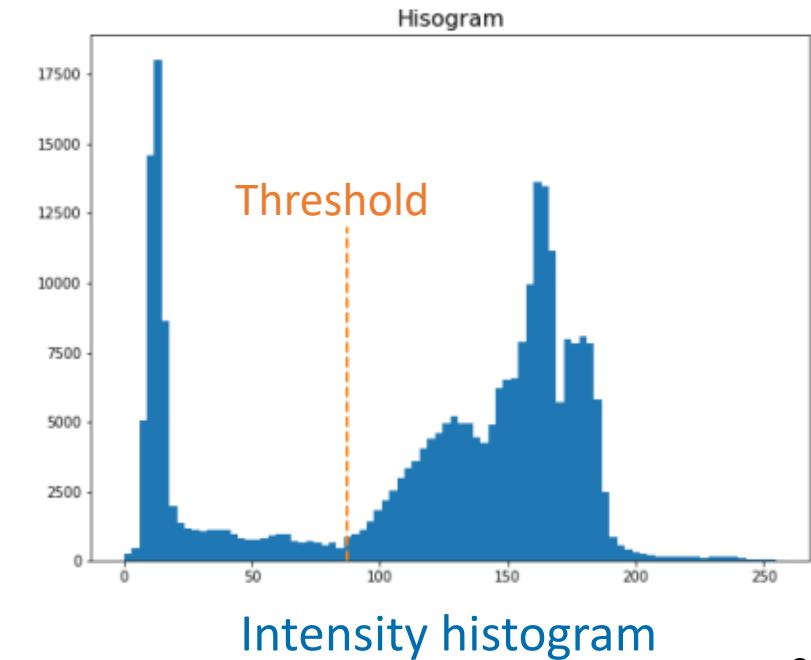
Segmentation

Thresholding

- No training data is involved.
- The only parameter is the threshold.
- It assumes that the pixel intensities can be grouped into two clusters and a threshold can separate the two.

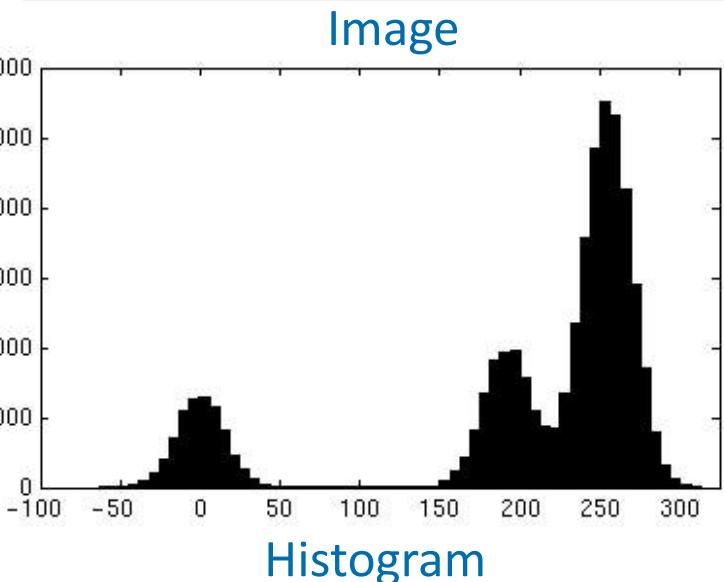
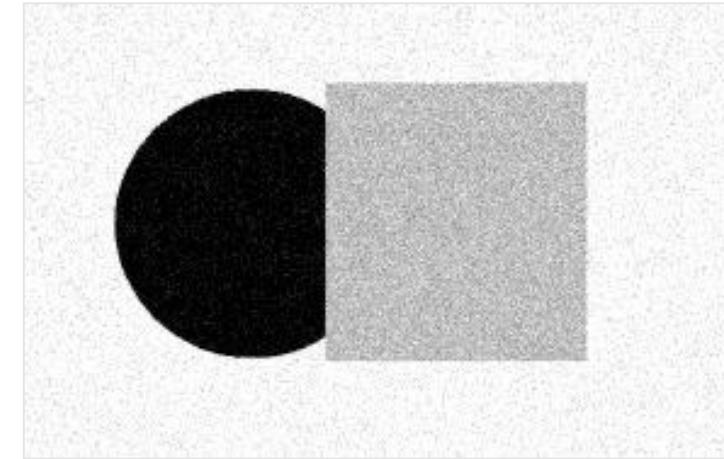


Image



K-means clustering

- In general cases, there can be more than two clusters.
- How do we define these clusters and determine which cluster each pixel belongs to?
- K-means clustering provides a way for cluster analysis, by iteratively estimating cluster parameters and pixel associations.

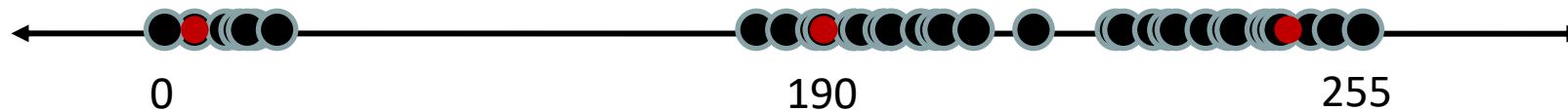


K-means clustering

- Parameters
 - Each cluster is represented by its centre.
- Association
 - Each data point (pixel intensity) is assigned to the nearest cluster centre.
- How to estimate the parameters?
 - The optimal cluster centres are those that minimise the intra-class variance.

$$\min \sum_{k=1}^K \sum_{x \in C_k} (x - \mu_k)^2$$

cluster k point x centre of cluster k



K-means clustering

- It can be re-formulated as,

$$\min \sum_{k=1}^K \sum_x \delta_{x,k} (x - \mu_k)^2$$

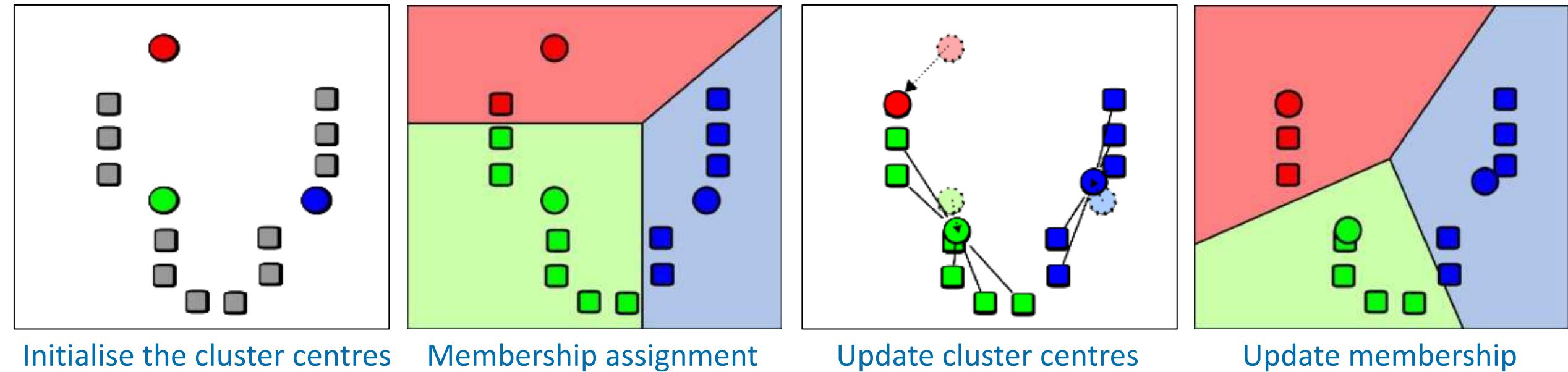
|
whether x is assigned to cluster k

- There are two unknowns, the membership $\delta_{x,k}$ and cluster centre μ_k .
- To know $\delta_{x,k}$ and assign a data point x to the nearest cluster centre μ_k , we need to know where the cluster centre is.
- To calculate the cluster centre μ_k , we need to know $\delta_{x,k}$.
- It is like a chicken and egg problem.

K-means clustering

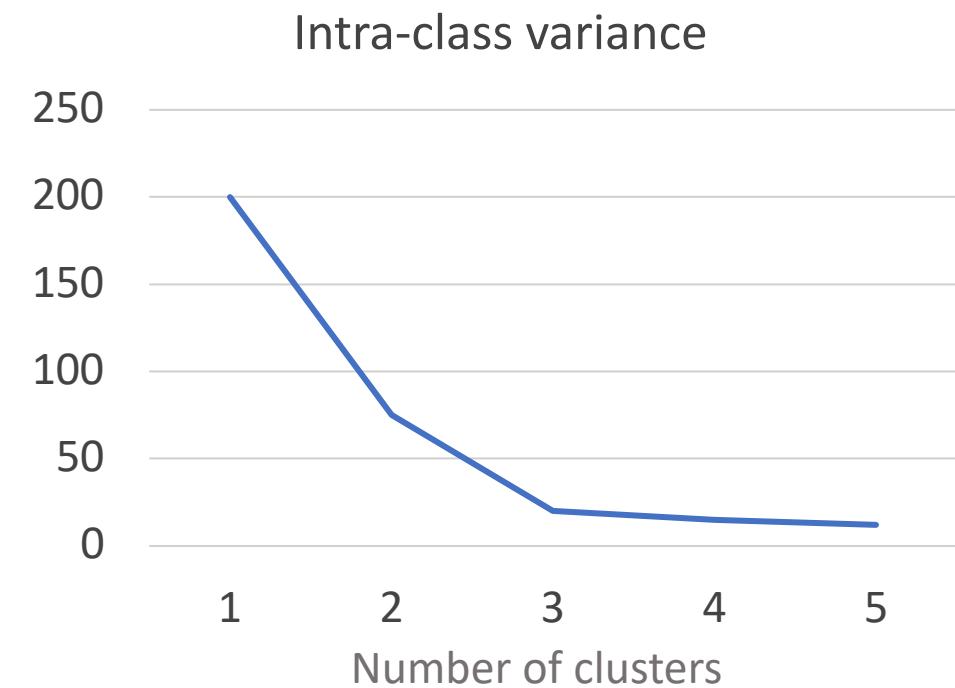
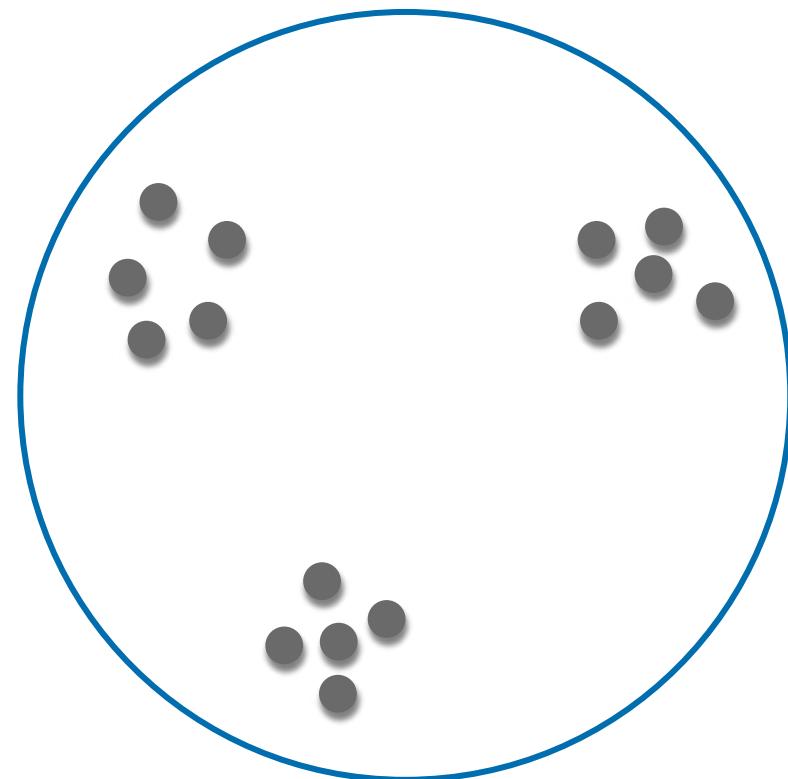
- Initialise the cluster centre $\mu_k, k = 1, 2, \dots, K$.
- For each iteration
 - Compute $\delta_{x,k}$ for each data point, assigning x to the nearest cluster centre μ_k .
 - Update μ_k according to the membership $\delta_{x,k}$.
 - Repeat till $\delta_{x,k}$ no longer changes or the maximum number of iterations is reached.

K-means clustering



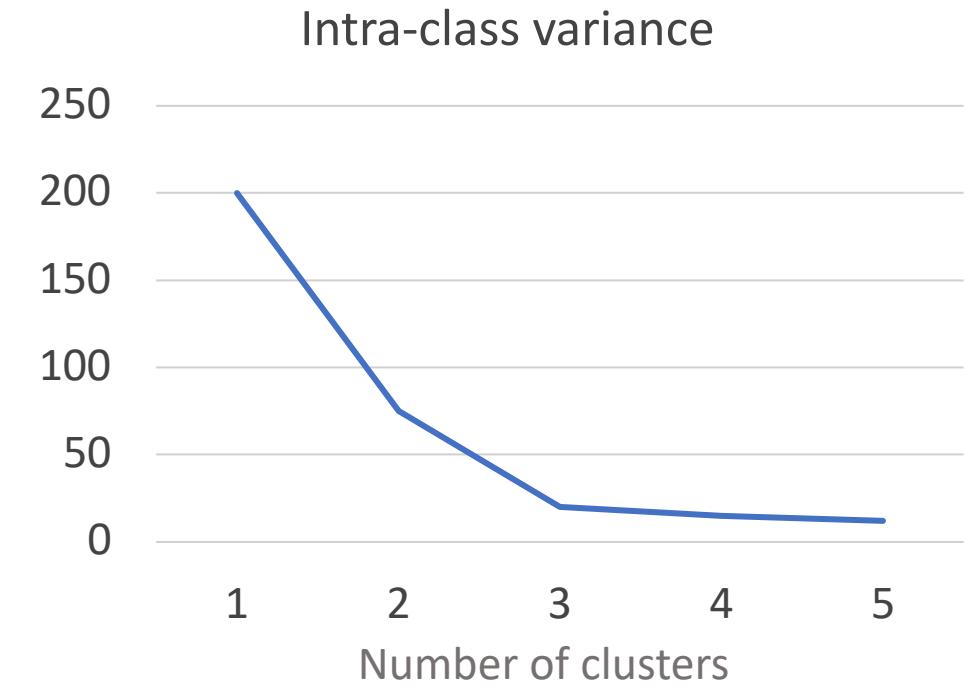
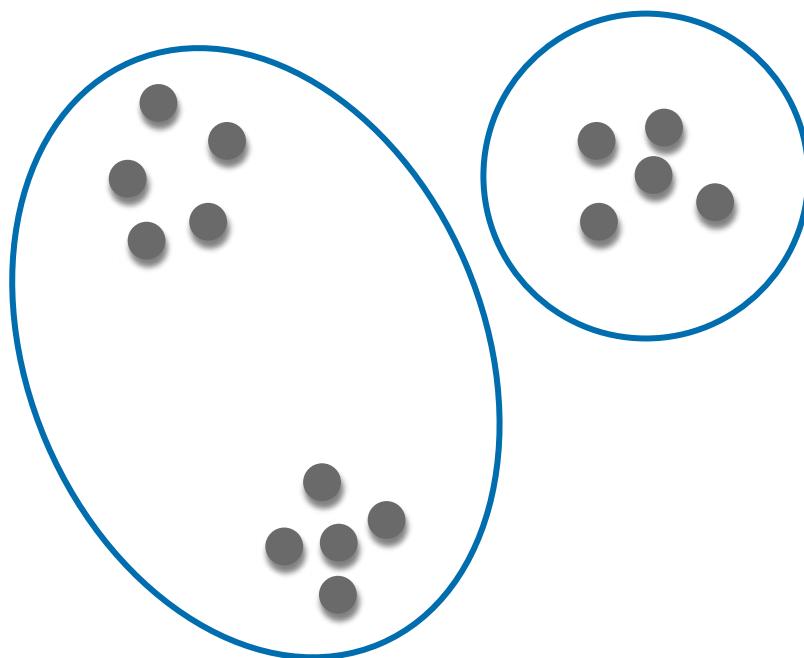
K-means clustering

- How do we determine how many clusters (K) to use?
 - By evaluating the intra-class variance $\sum_{k=1}^K \sum_x \delta_{x,k} (x - \mu_k)^2$.



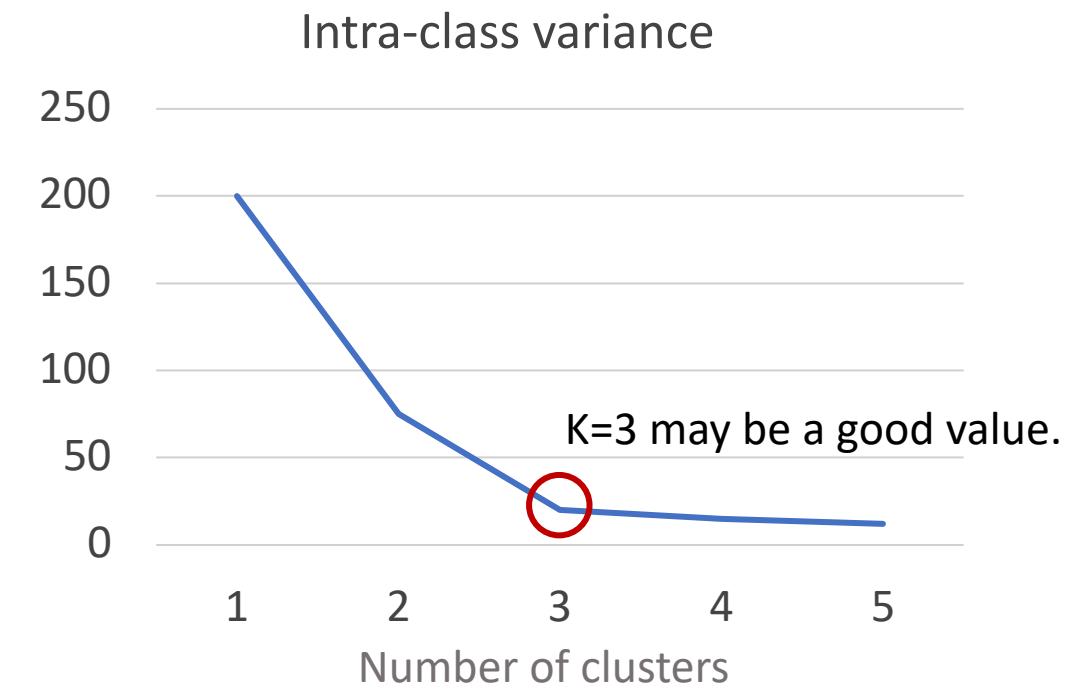
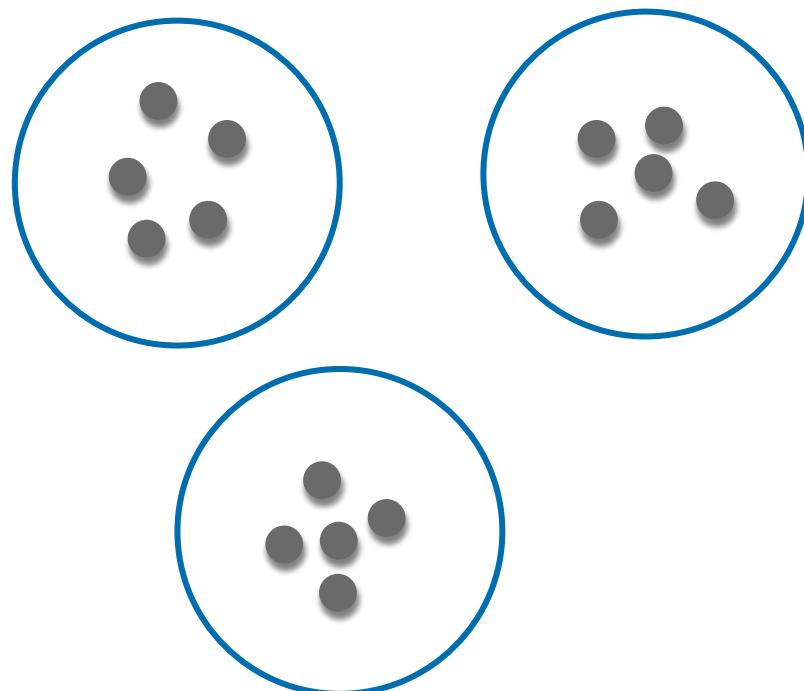
K-means clustering

- How do we determine how many clusters (K) to use?
 - By evaluating the intra-class variance $\sum_{k=1}^K \sum_x \delta_{x,k} (x - \mu_k)^2$.



K-means clustering

- How do we determine how many clusters (K) to use?
 - By evaluating the intra-class variance $\sum_{k=1}^K \sum_x \delta_{x,k} (x - \mu_k)^2$.

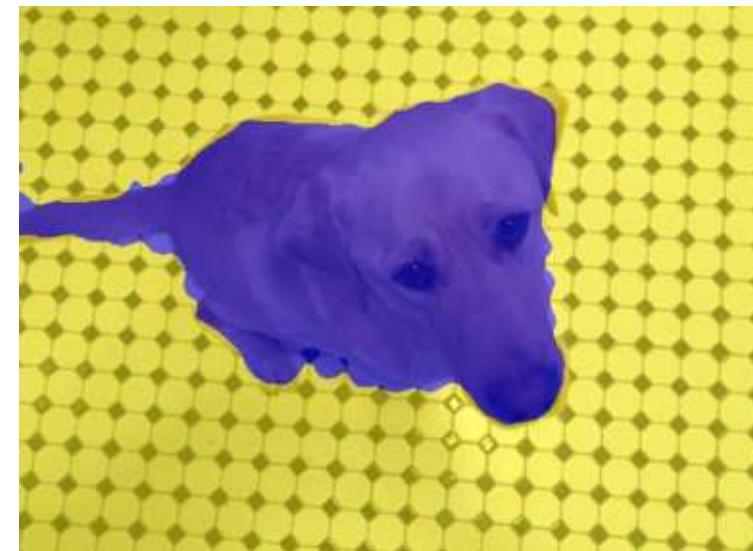


K-means clustering

- The feature can be more than just intensities.
- Clustering can also be performed based on
 - colour similarity,
 - position + colour similarity,
 - other features.
- For these cases, x becomes a feature vector, instead of a scalar.



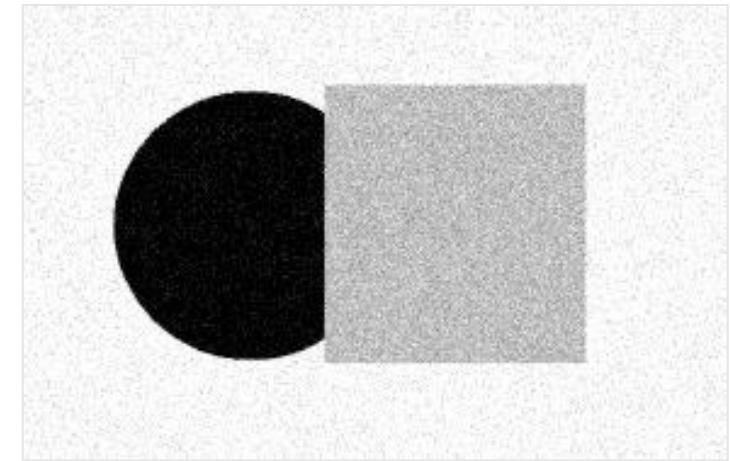
Input image



Clustering ($K = 2$)

Gaussian mixture model (GMM)

- K-means clustering performs a hard assignment of a data point x to cluster k .
 - $\delta_{x,k}$ is either 1 or 0.



Image

- Gaussian mixture model (GMM) performs a soft assignment by assuming a Gaussian distribution for each cluster.

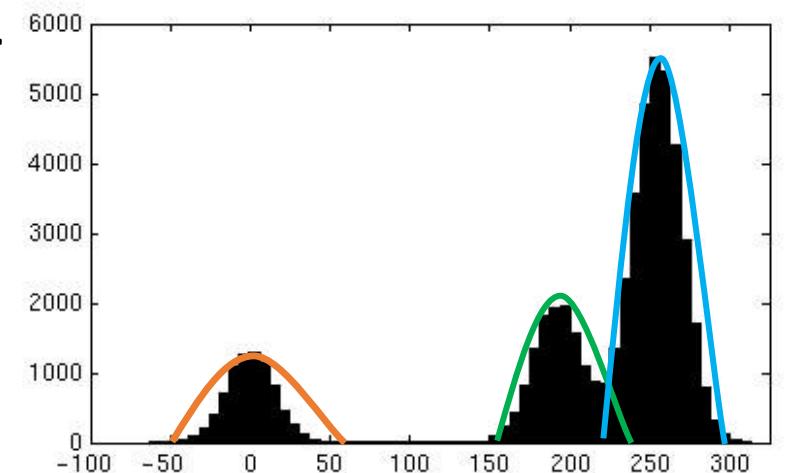
- The probability that x_j belongs to cluster k is

$$P(y_j = k | x_j, \pi_k, \mu_k, \sigma_k) = \pi_k \cdot \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(x - \mu_k)^2}{2\sigma_k^2}\right)$$

x_j : intensity or feature for data point j

j : class for data point j

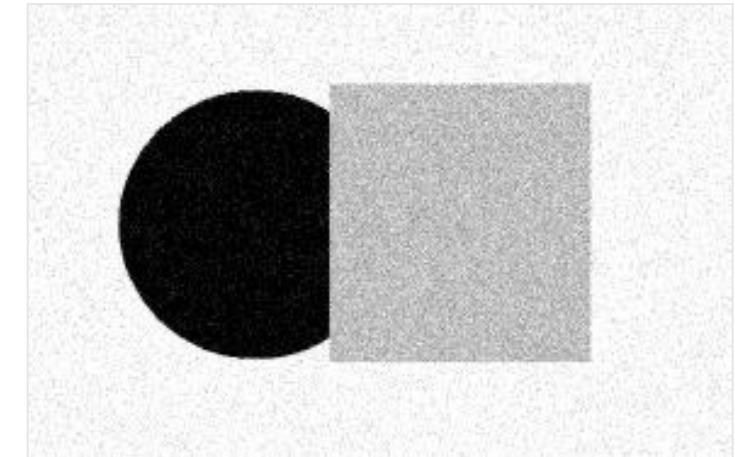
π_k, μ_k, σ_k : parameters for cluster k



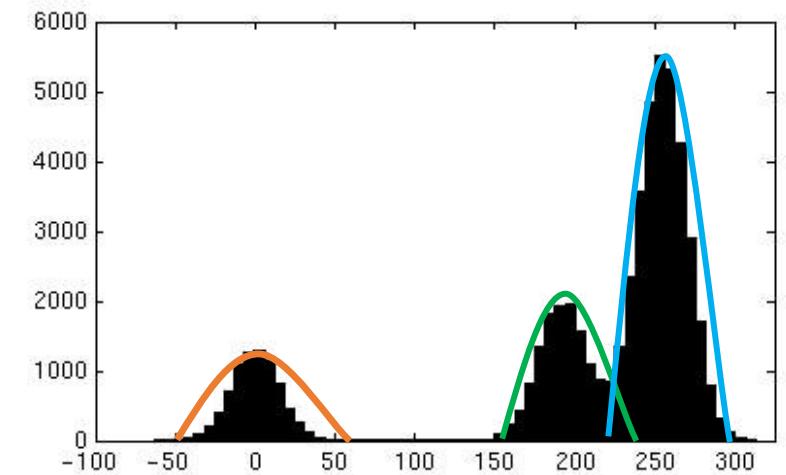
Histogram. Each cluster is modelled by a Gaussian distribution.

Gaussian mixture model (GMM)

- If we know the Gaussian parameters, we can estimate the probability of assigning a data point x to cluster k .
- If we know the assignment of data points, we can estimate the Gaussian parameters.
- Again, this is a chicken and egg problem.
- We can estimate them iteratively.



Image



Histogram. Each cluster is modelled by a Gaussian distribution.

Gaussian mixture model (GMM)

- Initialise the parameters $\pi_k, \mu_k, \sigma_k, k = 1, 2, \dots, K$.

- For each iteration

- Compute the membership for each data point $x_j, j = 1, 2, \dots, N$.

$$P(y_j = k | x_j, \pi_k, \mu_k, \sigma_k) = \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(x_j - \mu_k)^2}{2\sigma_k^2}\right)$$

- Update the Gaussian model parameters

$$\pi_k = \frac{\sum_j P(y_j = k)}{N}$$

π_k : proportion of cluster k

$$\mu_k = \frac{\sum_j P(y_j = k) \cdot x_j}{\sum_j P(y_j = k)}$$

μ_k : mean of cluster k

$$\sigma_k^2 = \frac{\sum_j P(y_j = k) \cdot (x_j - \mu_k)^2}{\sum_j P(y_j = k)}$$

σ_k : standard deviation of cluster k

- Repeat till parameter change is trivial or maximum number of iterations is reached.

Gaussian mixture model (GMM)

- Once we know the probability of data point x_j belonging to class c , we can perform image segmentation.
- We can assign a class to each pixel by maximum probability.

$$c = \operatorname{argmax}_c P(y_j = c | x_j, \pi, \mu, \sigma)$$

Gaussian mixture model (GMM)

- The background removal function in Powerpoint uses the GrabCut algorithm, which iteratively
 - Estimate GMM parameters for two clusters (foreground and background).
 - Perform segmentation with smoothness constraint and user edit constraint (using GraphCut, not explained here).

“GrabCut” — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother*

Vladimir Kolmogorov†

Andrew Blake‡

Microsoft Research Cambridge, UK

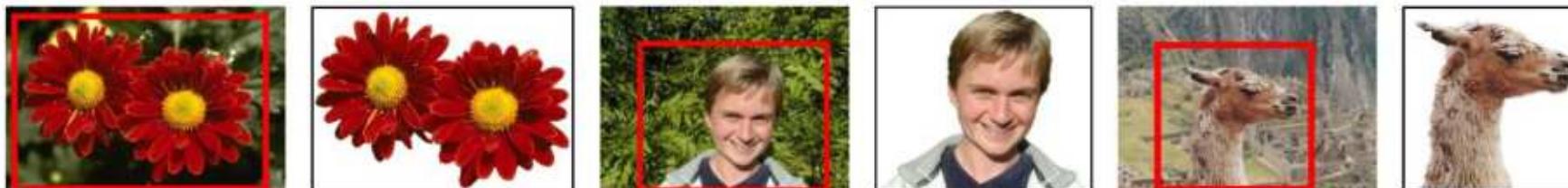


Figure 1: Three examples of GrabCut. The user drags a rectangle loosely around an object. The object is then extracted automatically.

Unsupervised method

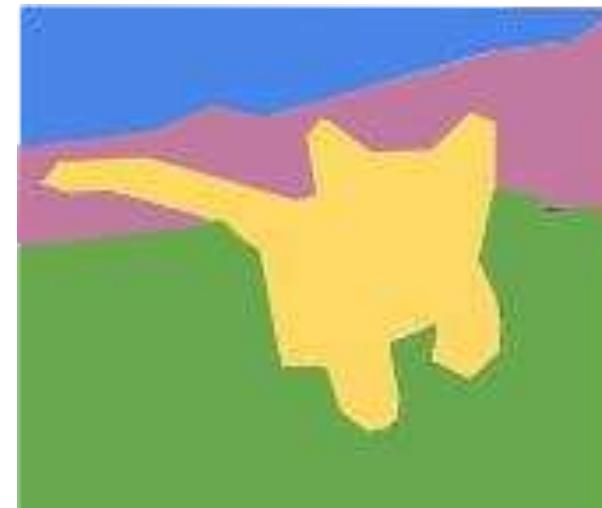
- K-means clustering and GMM are unsupervised methods.
 - No training data needed.
 - Easy to implement and fast.
- Next, we are going to explain supervised methods.
 - It needs training data.
 - In terms of accuracy, supervised methods often performs better than unsupervised methods.

Convolutional neural networks

- Semantic segmentation can be considered as pixel-wise classification.
- We train neural networks with supervisions (image + annotated segmentation map).



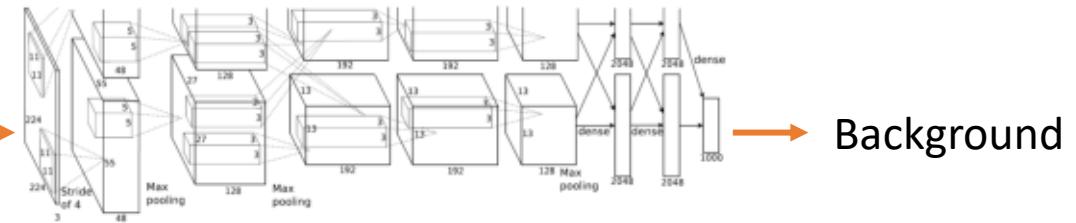
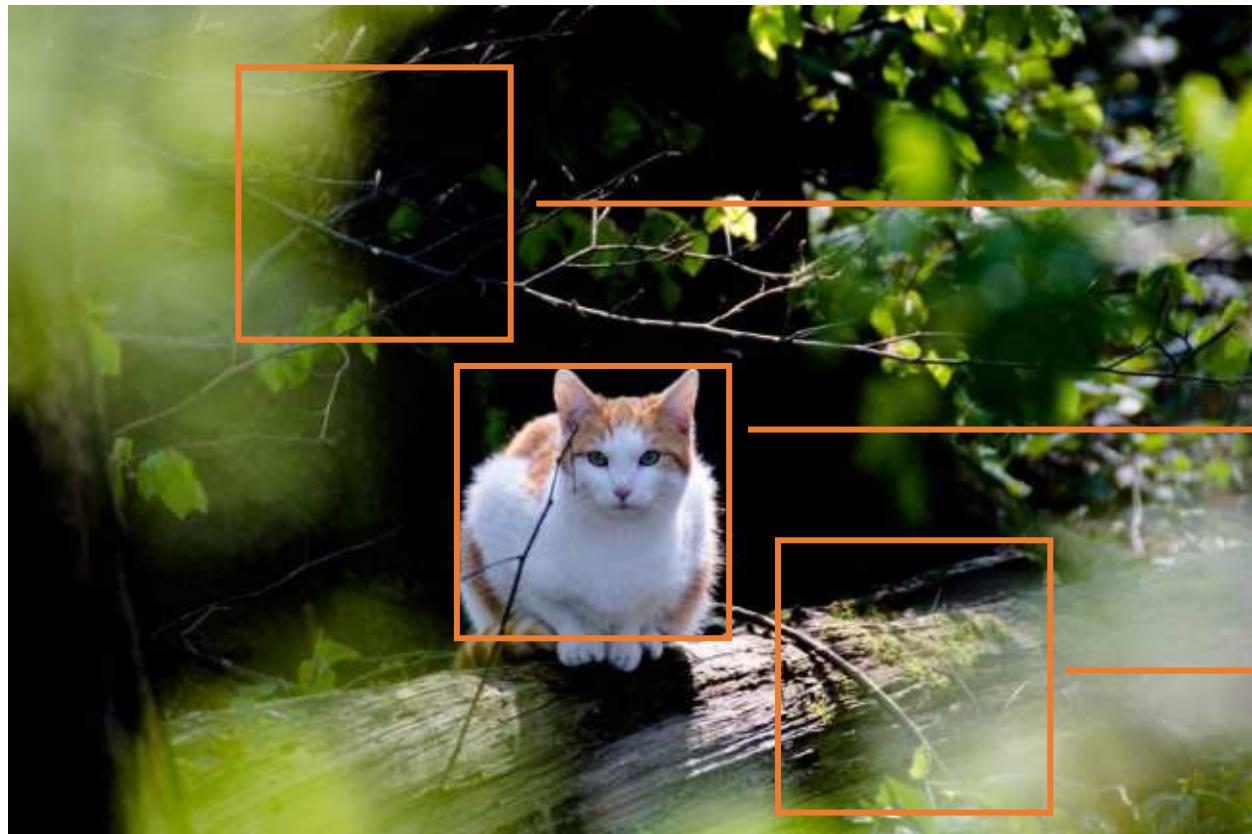
Image



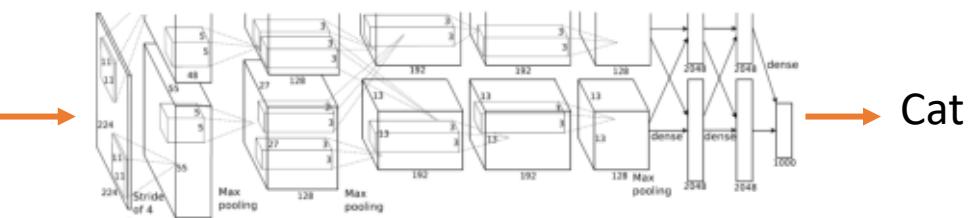
Segmentation
Grass, Cat, Tree, Sky

Segmentation idea

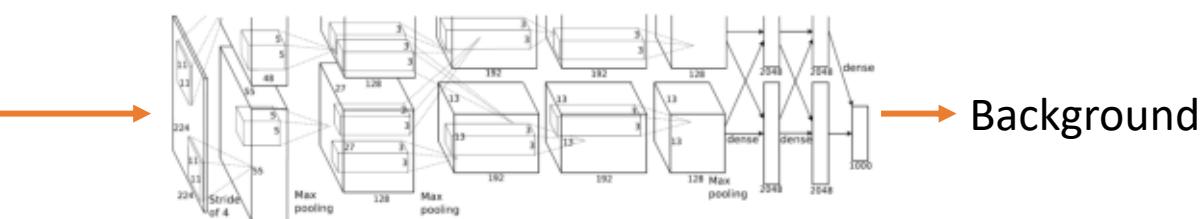
- We already know how to develop a model for image classification.
- Since segmentation is pixel-wise classification, why not applying this classification model to each pixel of the image?



Background



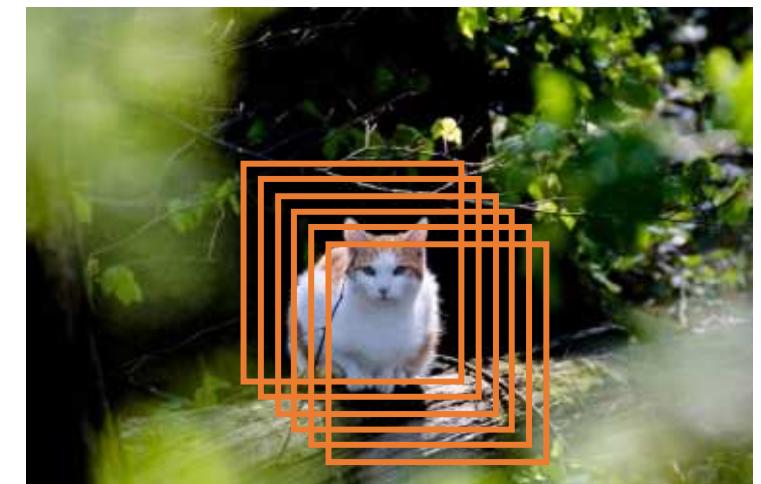
Cat



Background

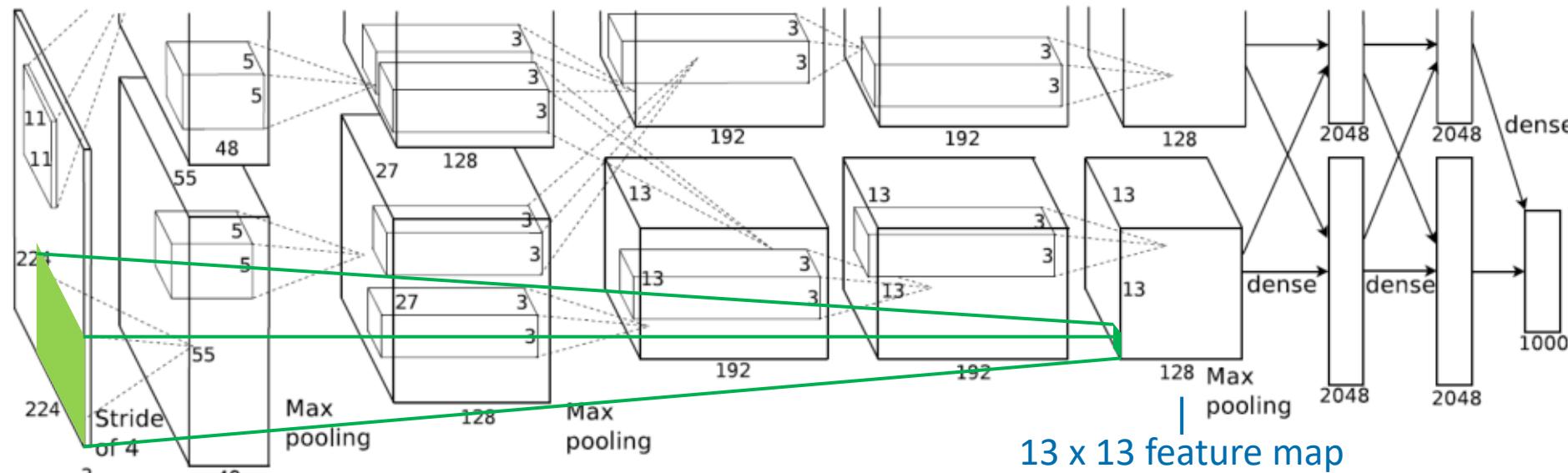
Concerns

- However, it is expensive to apply the network multiple times to all pixels.
- The reason that we need to apply the classification network many times is because that its output is a probability vector for an image.
- Can we develop a network which outputs a pixel-wise probability map, instead of a single probability vector?



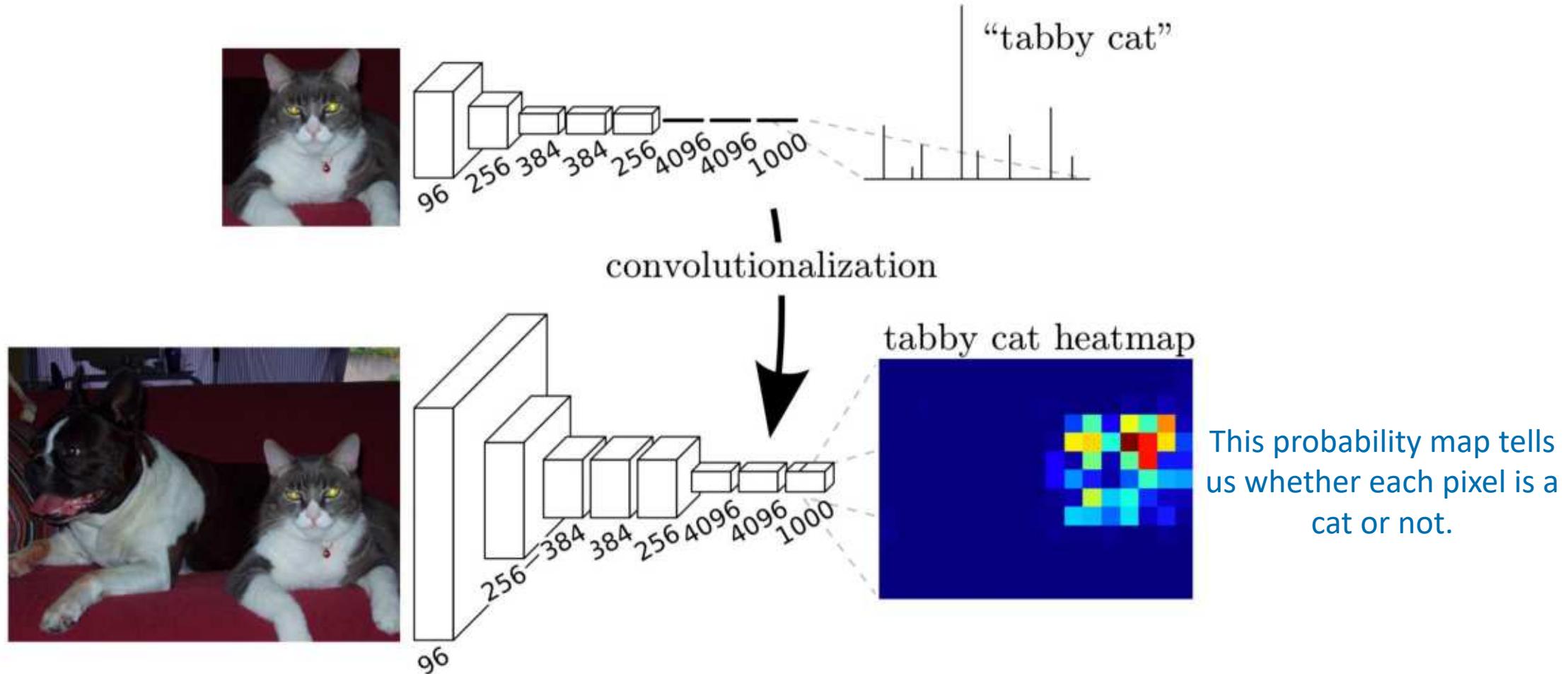
After the last convolutional layer of AlexNet

- The 13×13 feature map is downsampled by a factor of 16 from a 224×224 input image.
 - Each pixel in the feature map encodes information of a 16×16 region in input image.
 - Why not we remove the fully connected layers and directly infer pixel-wise classification results from the feature map?



A pixel here encodes information for a region in input image.

Fully convolutional network

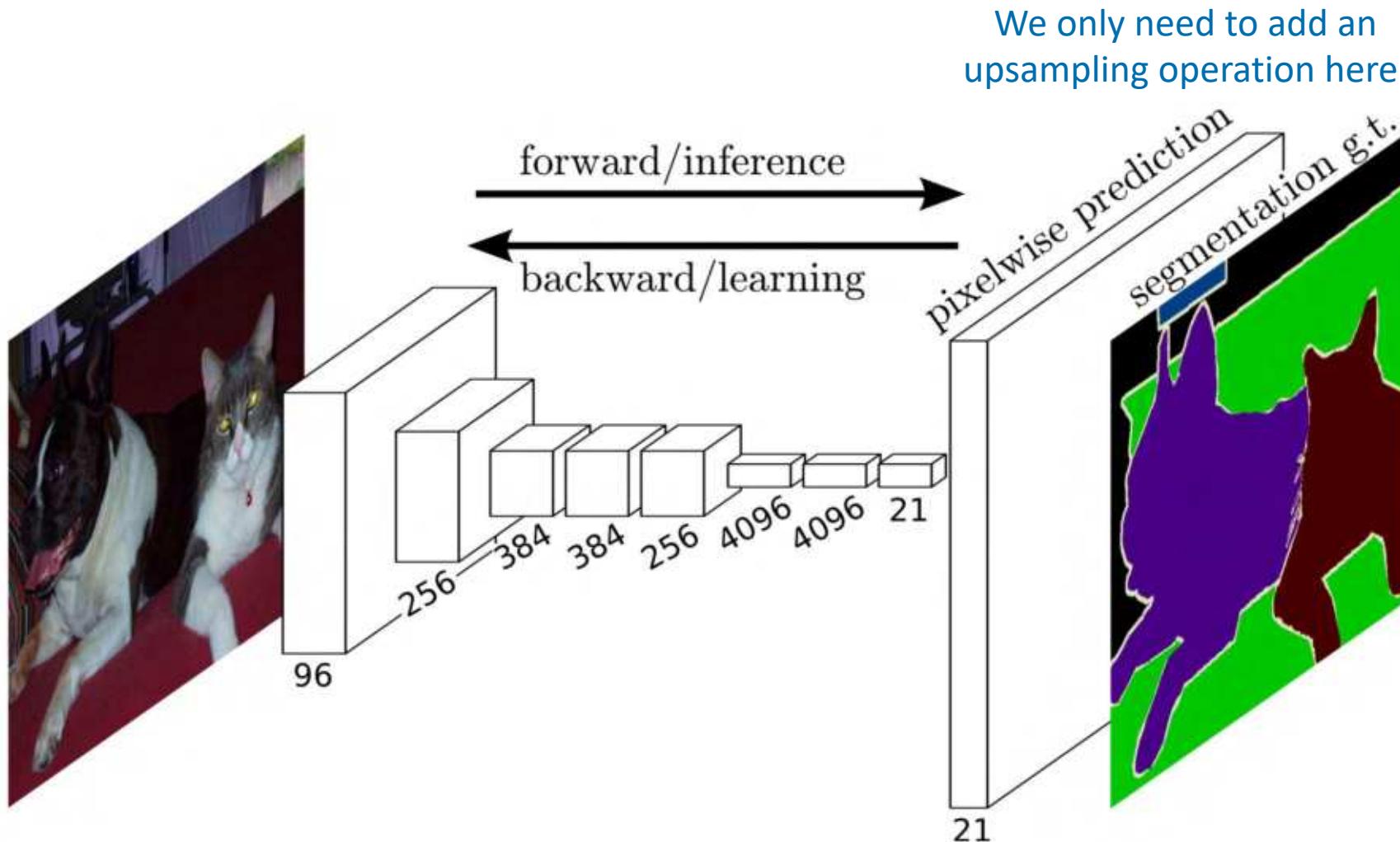


Transforming fully connected layers into convolutional layers enables the network to generate a pixel-wise probability map.

Fully convolutional network

- Since all layers are convolutional layers (no more fully connected layers), it is called a fully convolutional network.
- The only problem is that the pixel-wise probability map has a downsampled size of the original image.
- We can recover it to the original image size by upsampling.

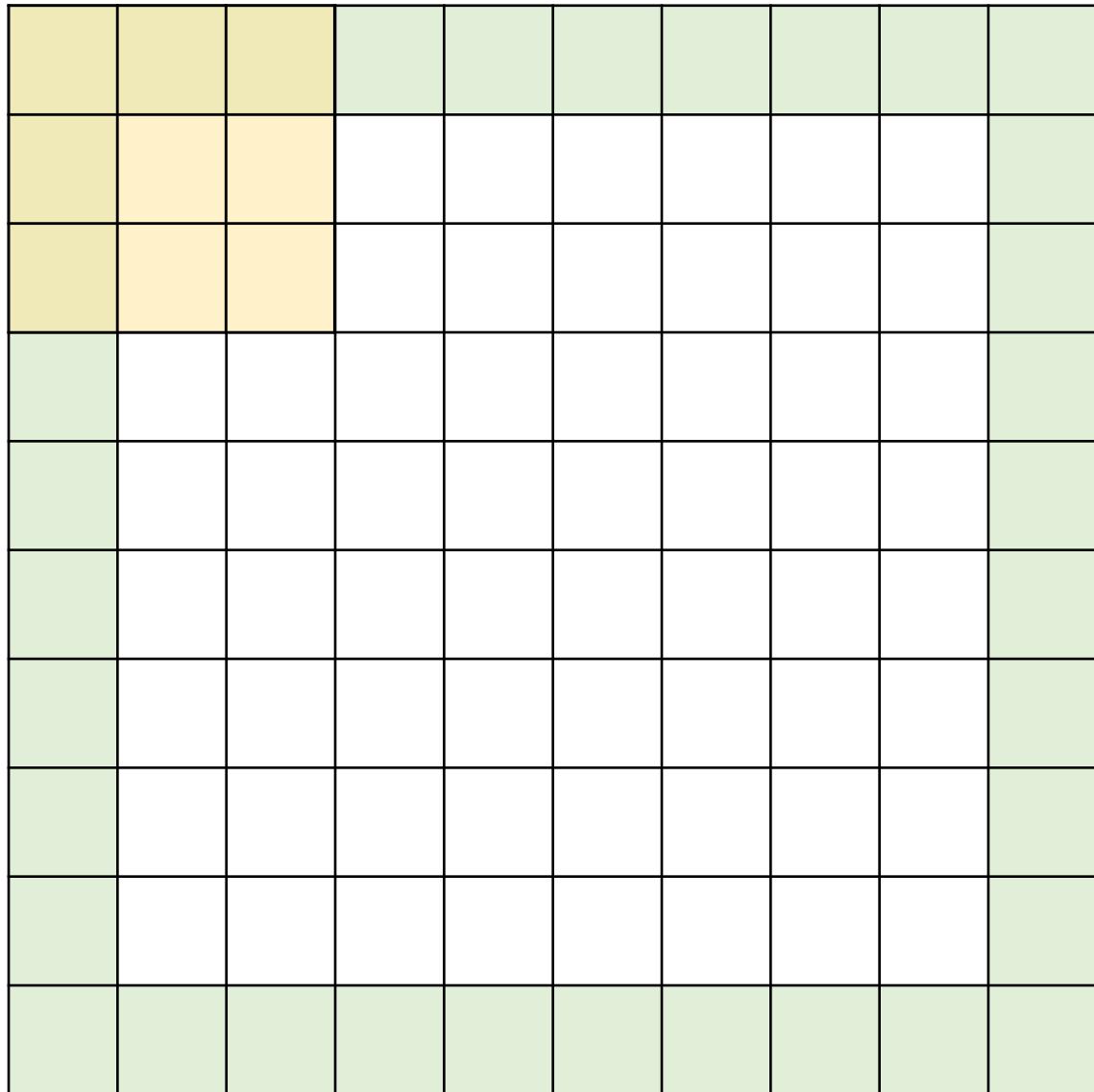
Fully convolutional network



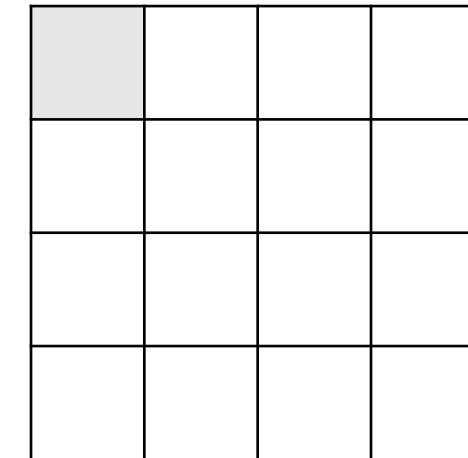
Upsampling

- We know how to perform downsampling in a neural network.
 - Strided convolution
 - Pooling
- How do we perform upsampling in a neural network?
 - Transposed convolution

Convolution (stride = 2)

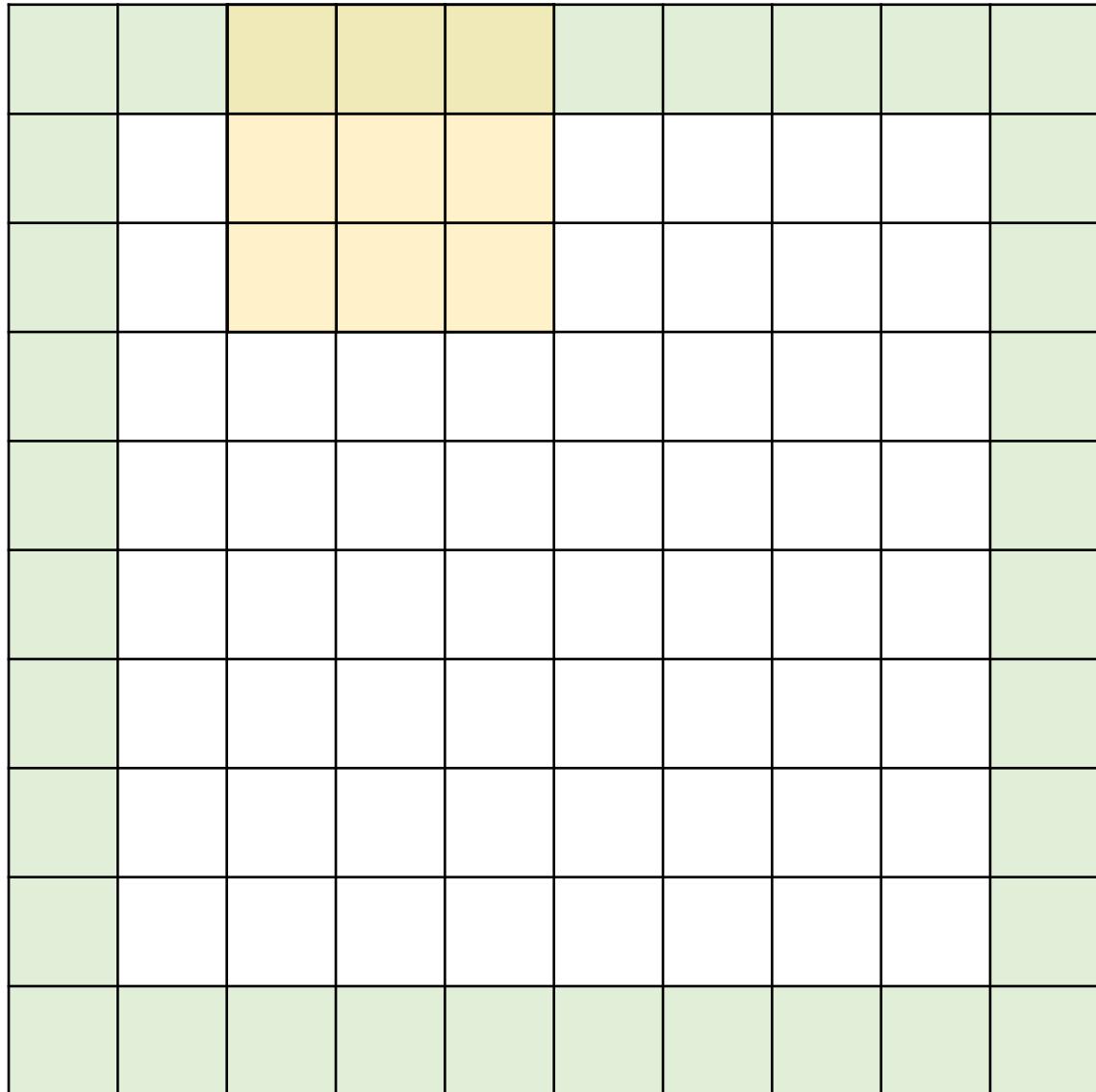


Input shape = [8, 8]

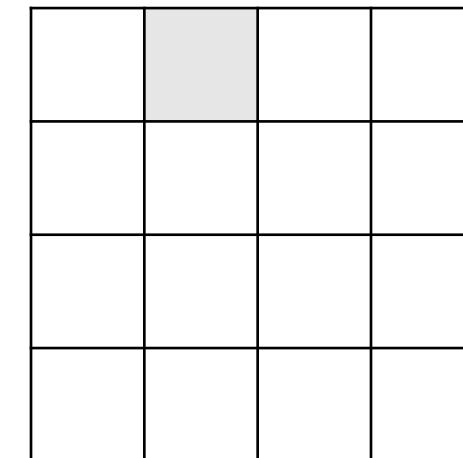


Output shape = [4, 4]

Convolution (stride = 2)

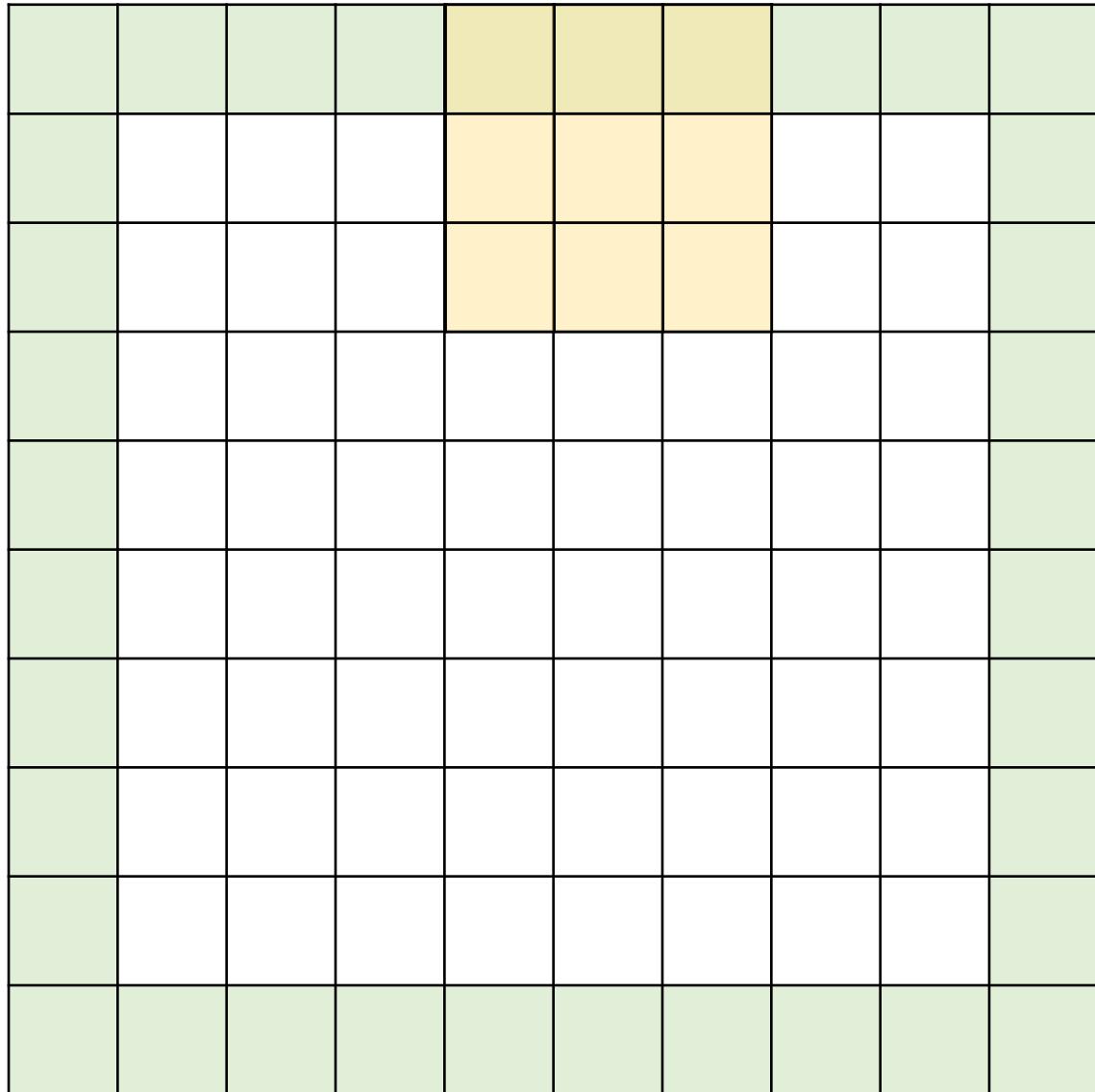


Input shape = [8, 8]

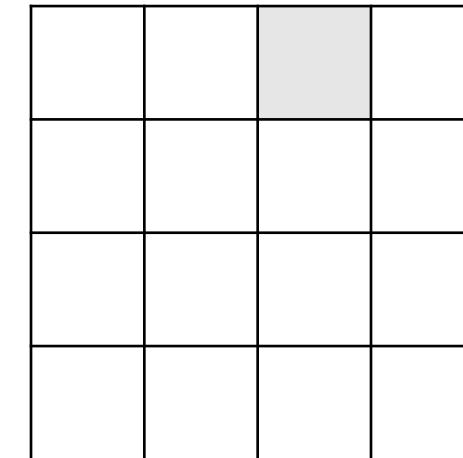


Output shape = [4, 4]

Convolution (stride = 2)

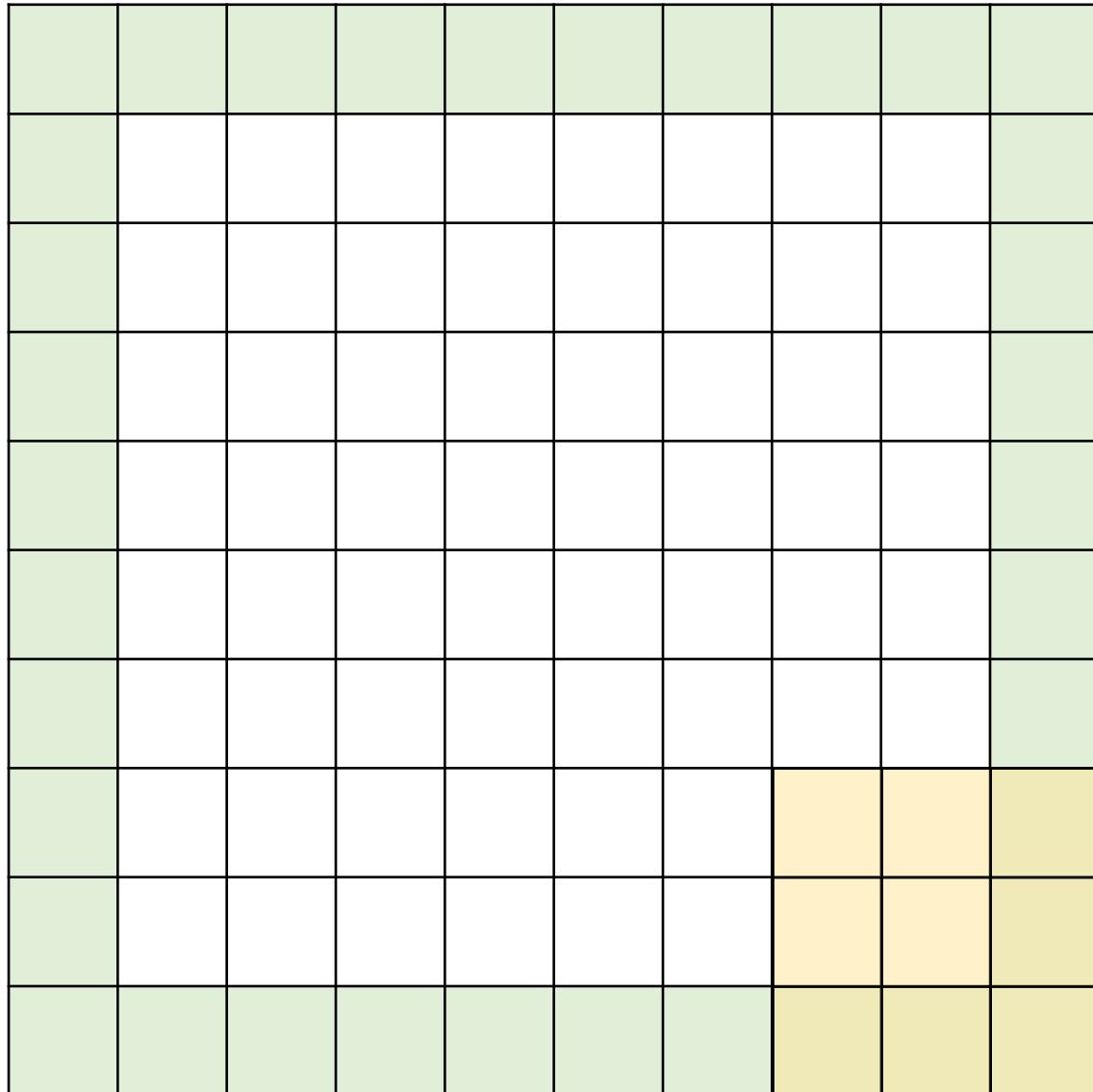


Input shape = [8, 8]

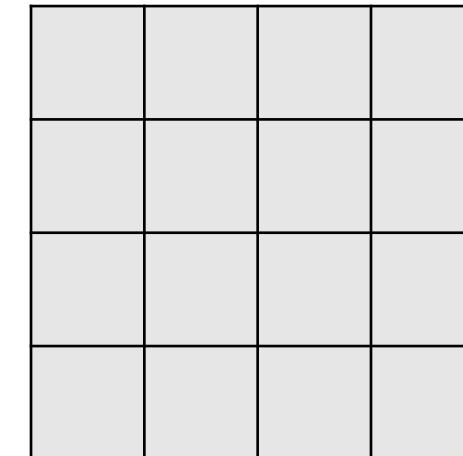


Output shape = [4, 4]

Convolution (stride = 2)



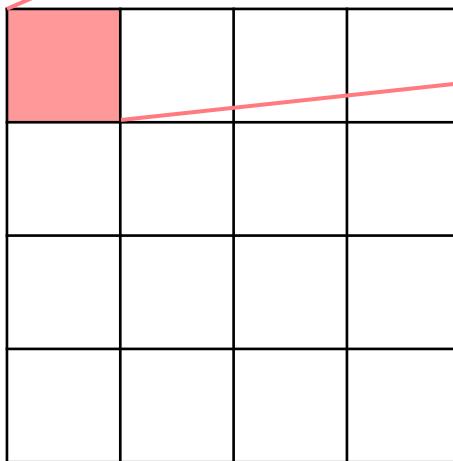
Input shape = [8, 8]



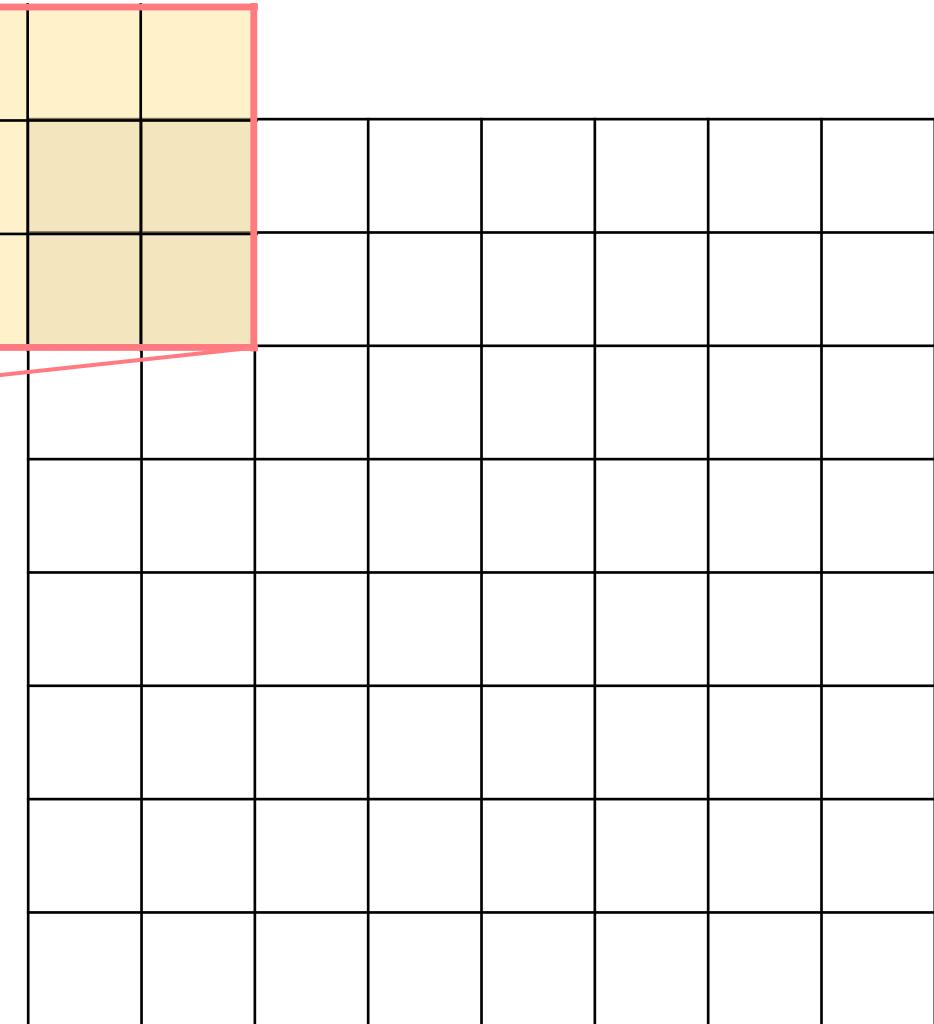
Output shape = [4, 4]

Transposed convolution (stride = 2)

Each pixel in the input contributes to a 3x3 window in the output with some weights.

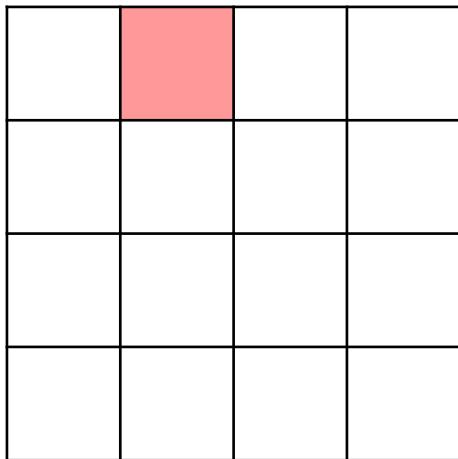


Input shape = [4, 4]

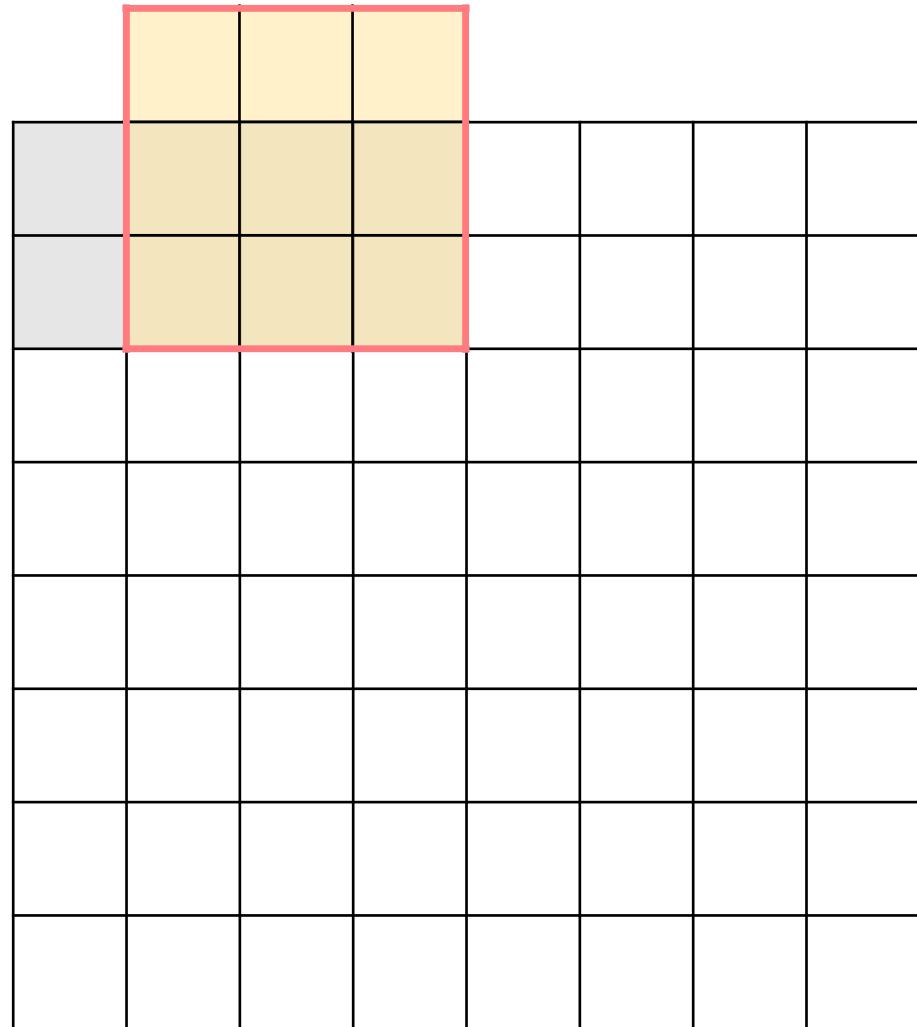


Output shape = [8, 8]

Transposed convolution (stride = 2)

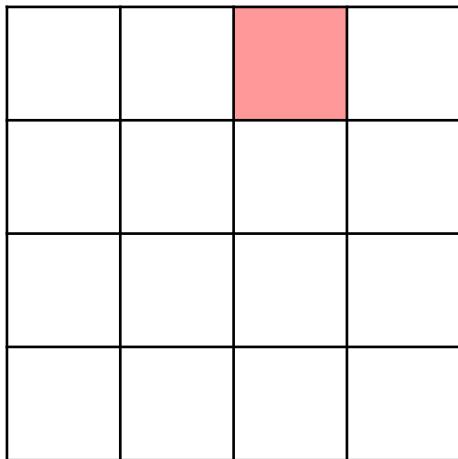


Input shape = [4, 4]

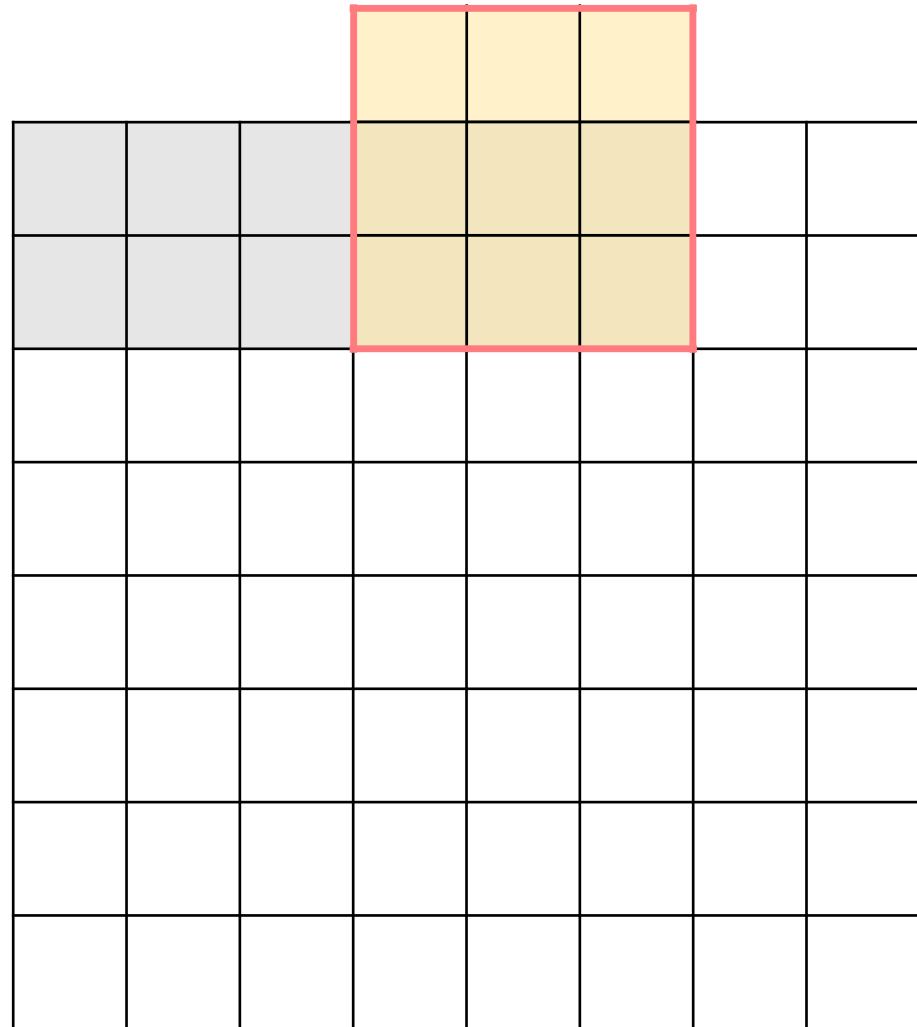


Output shape = [8, 8]

Transposed convolution (stride = 2)

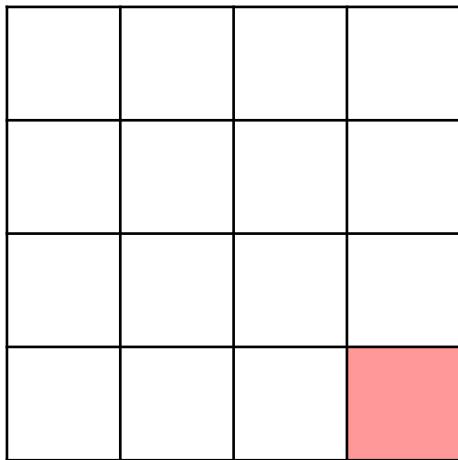


Input shape = [4, 4]

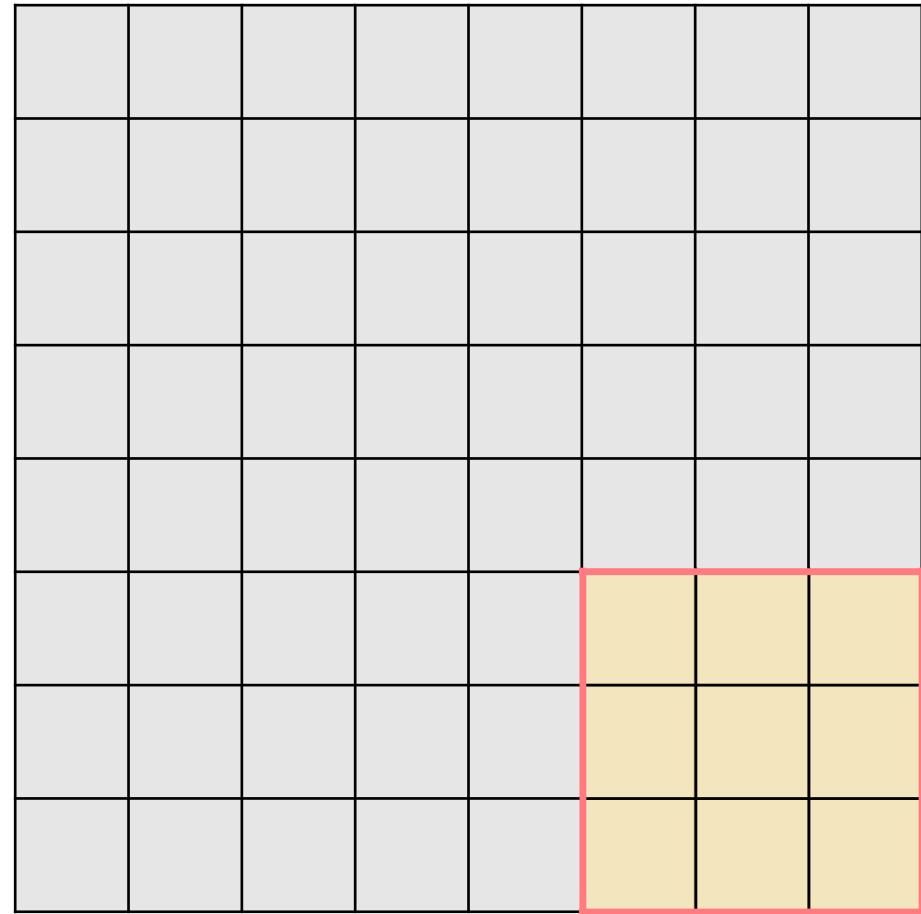


Output shape = [8, 8]

Transposed convolution (stride = 2)



Input shape = [4, 4]



Output shape = [8, 8]

Convolution as a matrix operation

Convolution in 1D

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 \\ 0 & w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

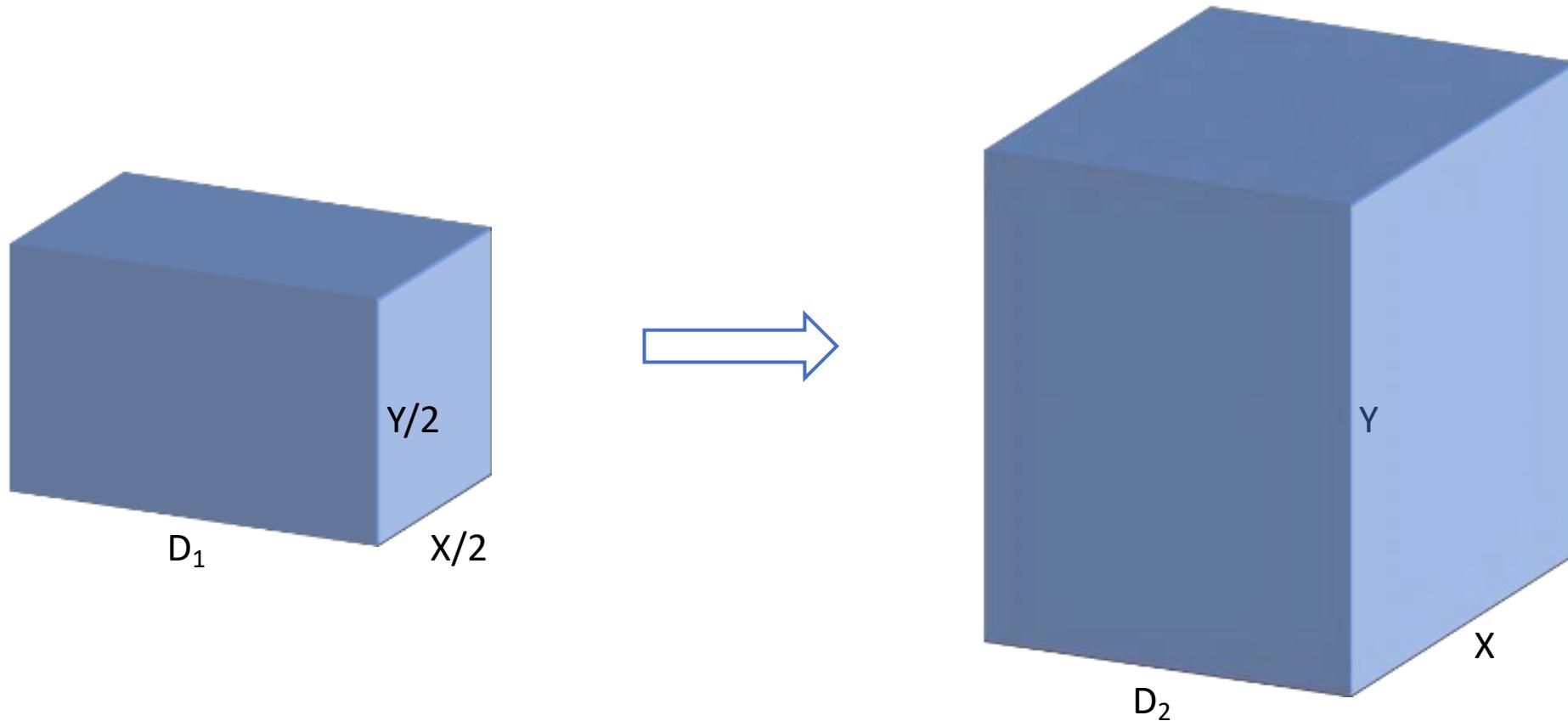
Transposed convolution in 1D

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{trans. conv.} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

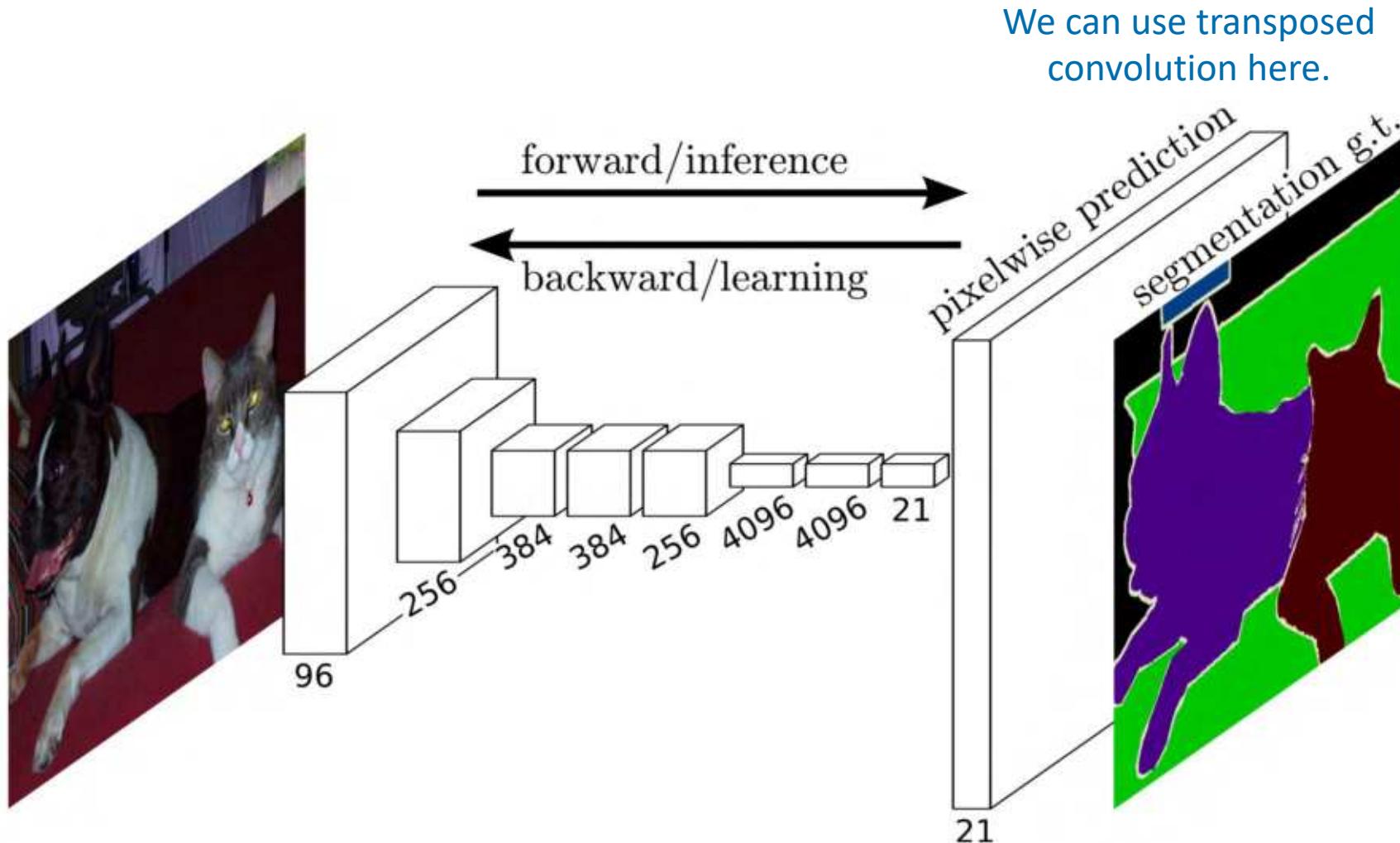
$$\begin{bmatrix} w_1 & 0 \\ w_2 & w_1 \\ w_3 & w_2 \\ 0 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

The weight matrix
is transposed.

Transposed convolutional layer



Fully convolutional network

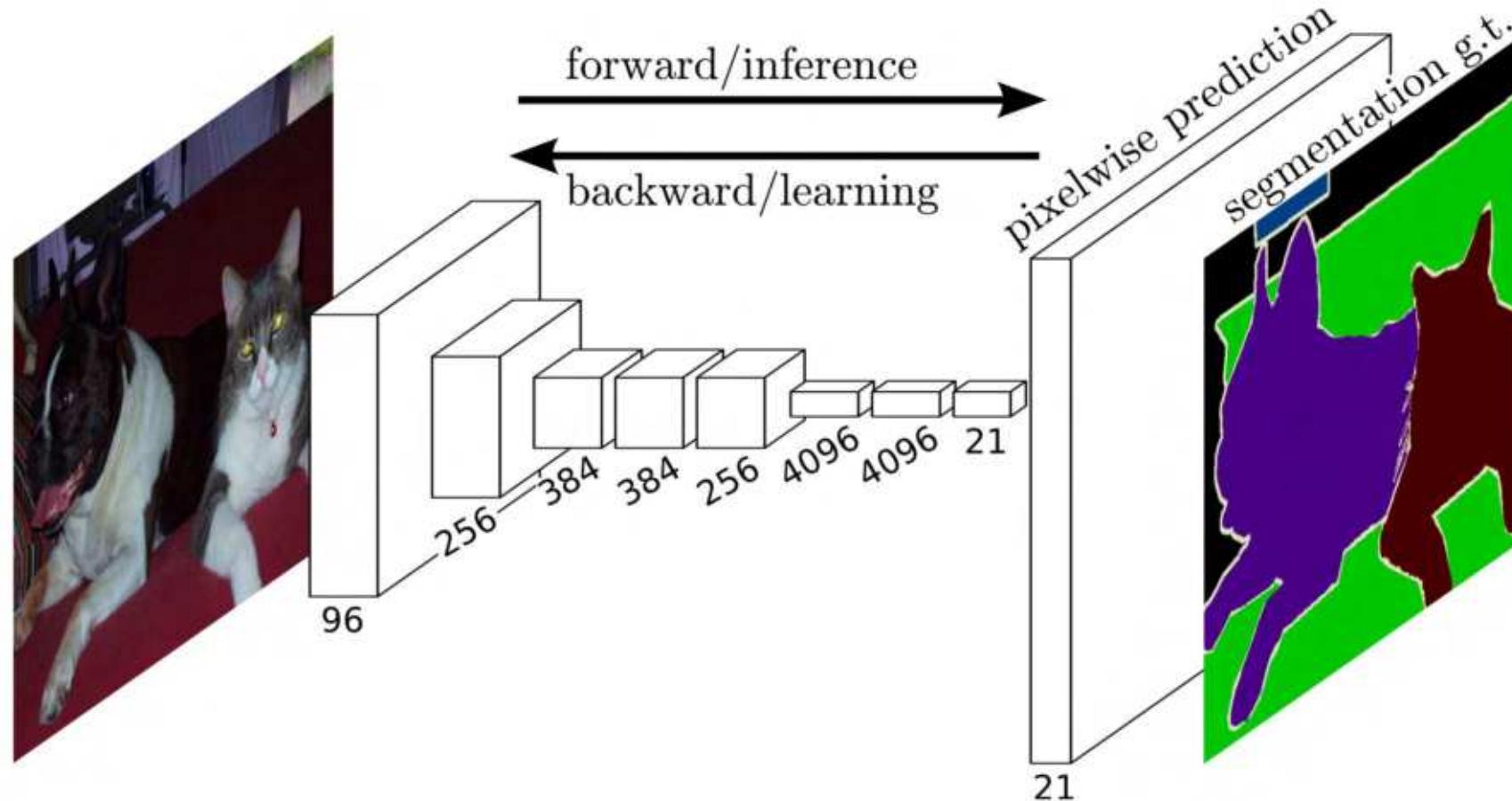


Fully convolutional network

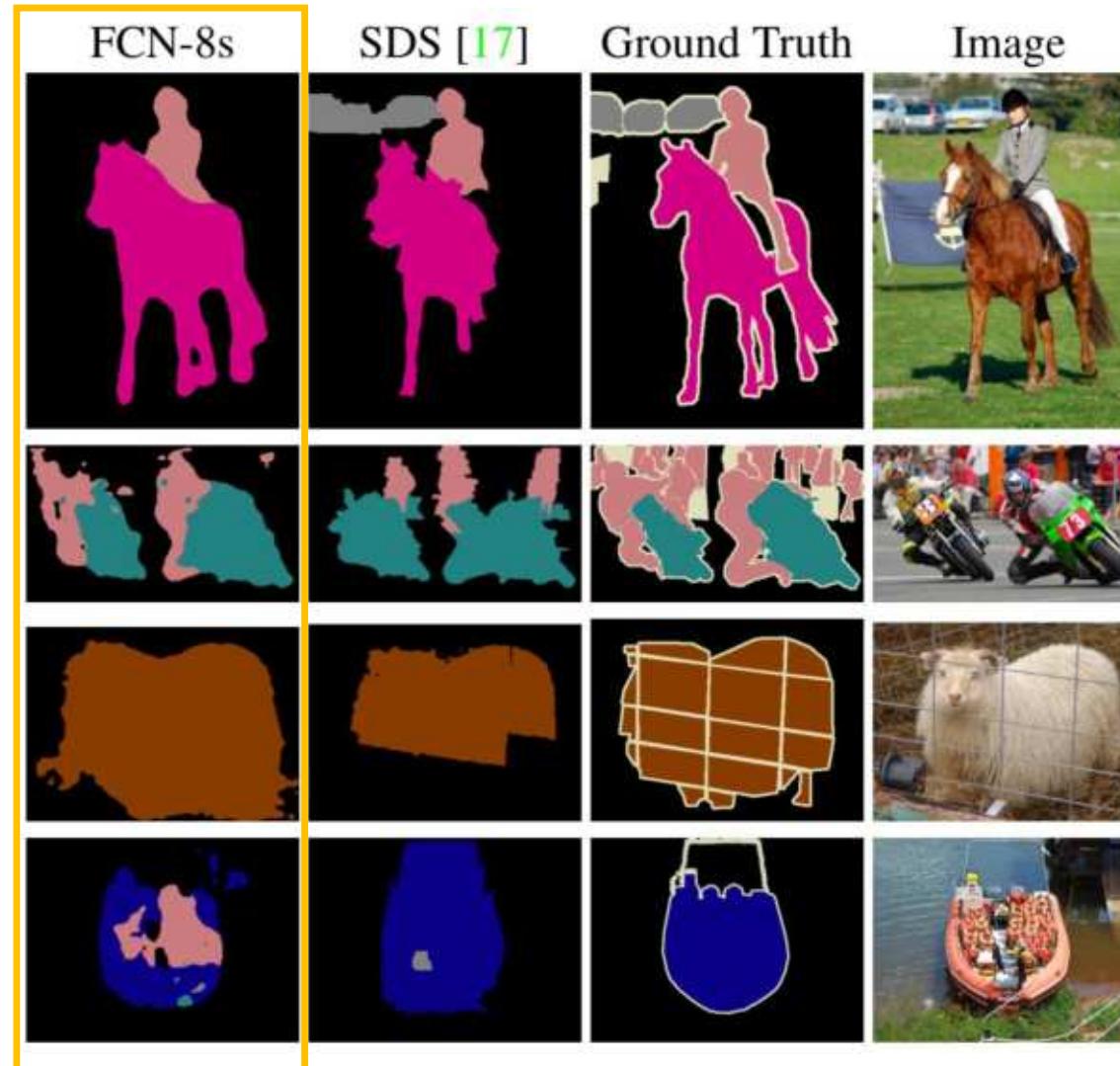
- We have transformed a classification network into a segmentation network, simply by
 - removing the fully connected layers
 - adding convolutional and transposed convolutional layers
- Next question is how to train this segmentation network.
 - Clue: Segmentation is essentially a pixel-wise classification problem.

Fully convolutional network

- At each pixel, we can define a classification loss (e.g. cross entropy).
- The segmentation loss is the average classification loss for all pixels.
- The network can be trained by optimising the segmentation loss.



Performance of FCN



Convolutional neural networks

- There are many recent works that adopt CNNs for image segmentation.
 - They use similar building blocks: convolution, transposed convolution, etc.
 - They may differ in backbone networks, i.e. the network used for extracting features.
 - They may differ in how you connect the convolutional blocks.
- We will explain more in the next lecture.

Summary

- Segmentation is a pixel-wise classification problem.
- Unsupervised segmentation methods
 - Thresholding
 - K-means clustering
 - Gaussian mixture model
- Supervised segmentation methods
 - Convolutional neural networks

References

- Sec. 5.3.1 K-means and mixture of Gaussians; Sec. 5.5 Graph cuts and energy-based methods. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

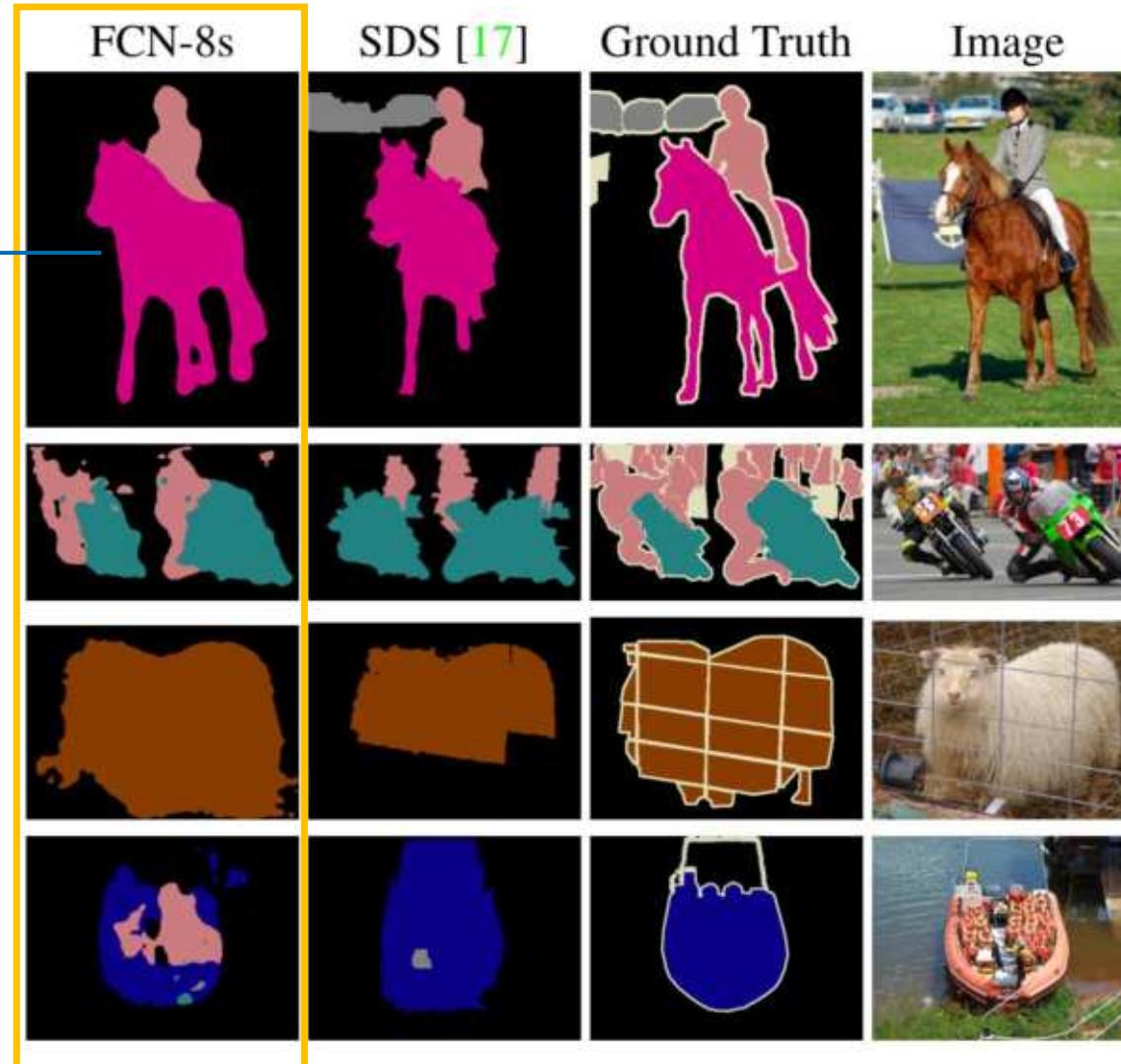
Image Segmentation II

Dr Wenjia Bai

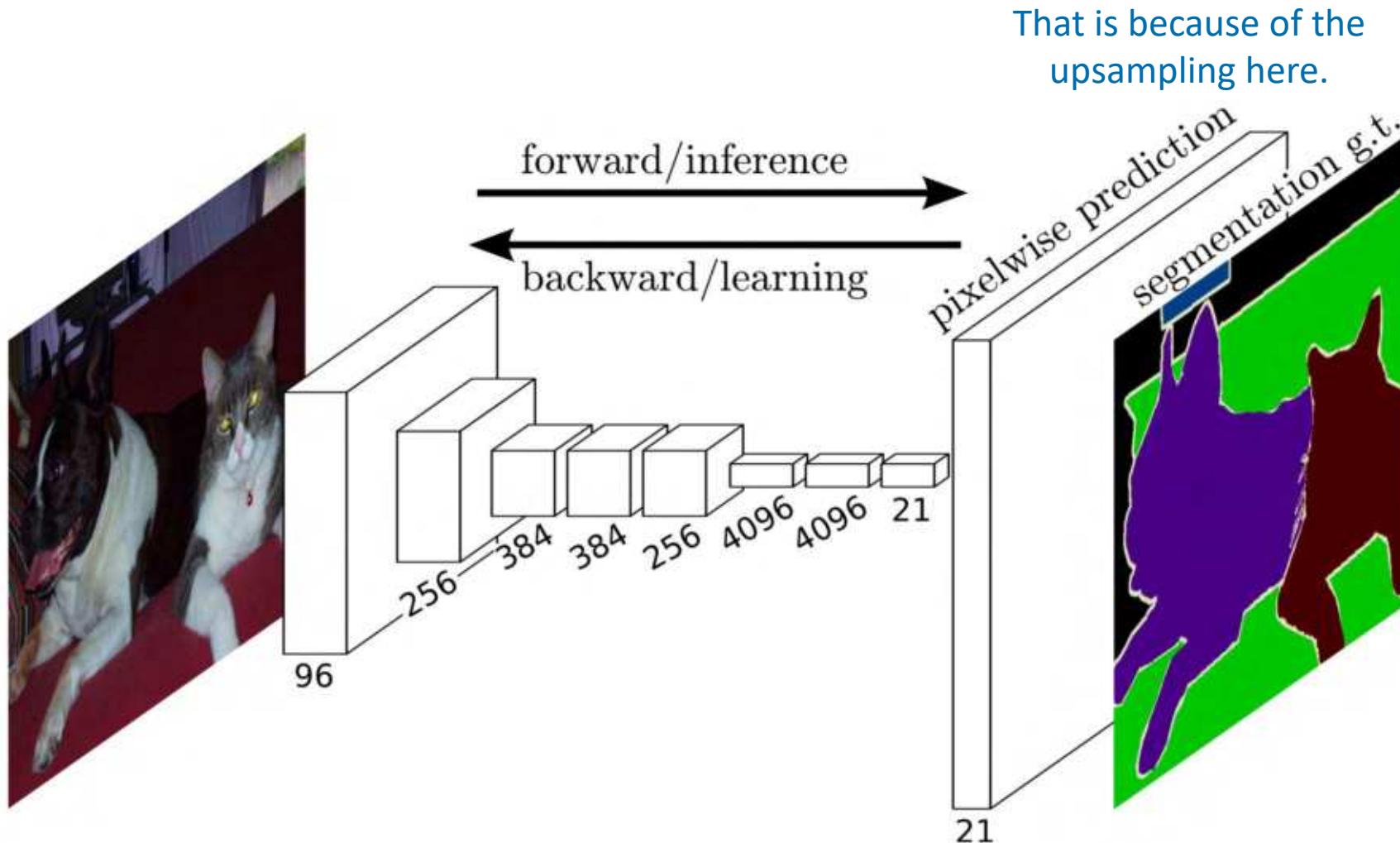
Department of Computing & Brain Sciences

Performance of FCN

Details of the boundary
are not well delineated.



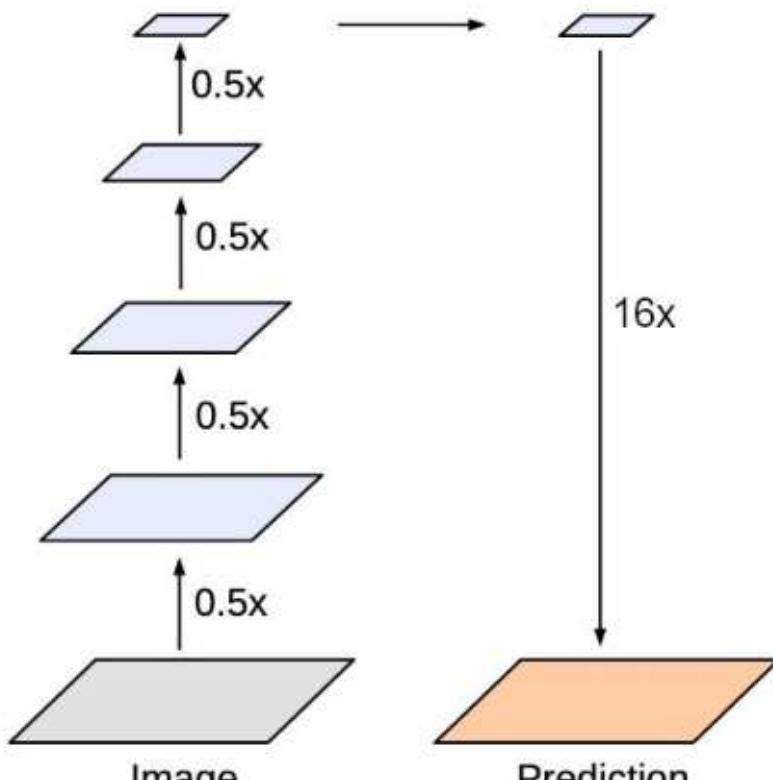
Fully convolutional network



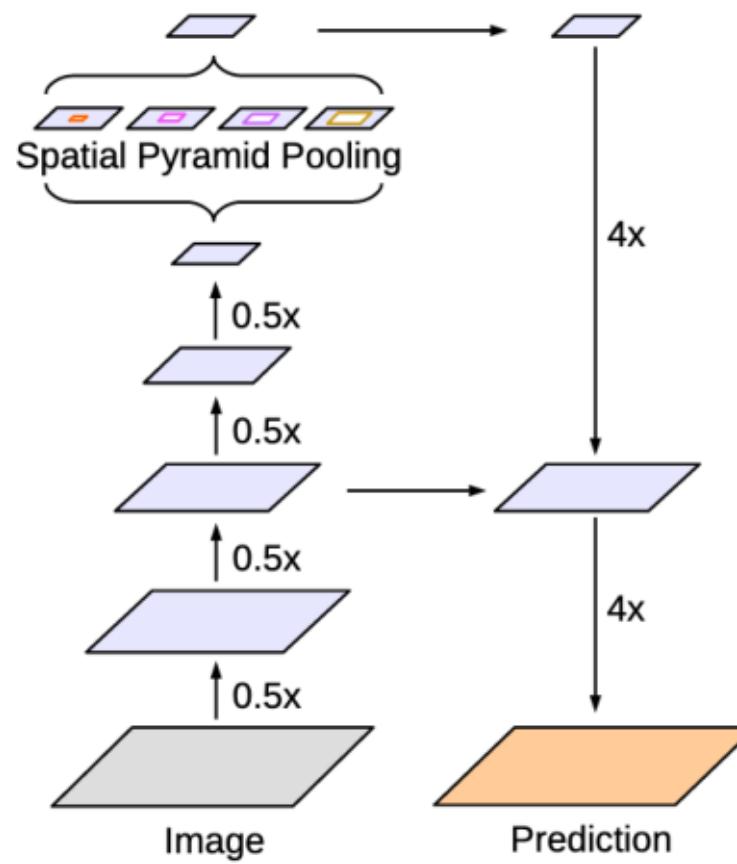
Thoughts

- To accurately delineate the boundary of an object, we need both
 - Local feature (pixel intensities, edges etc)
 - Global feature (what object this is, what it is next to, context)
- The first few layers of a neural network extract local features. Deep layers encode global features.
- We will introduce two examples that integrates multi-scale features.
 - DeepLabv3+
 - U-net

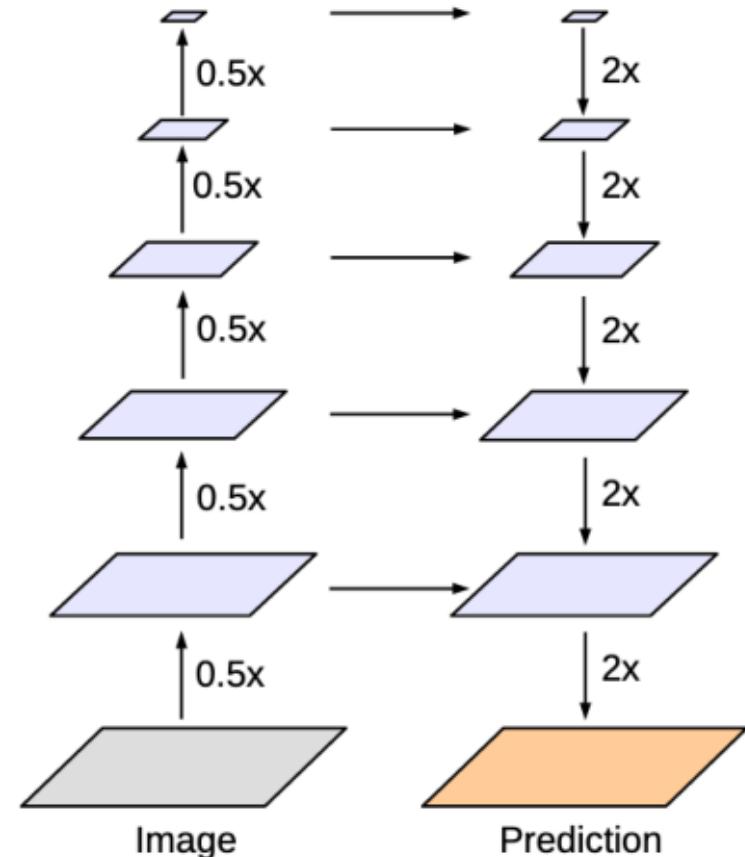
Segmentation networks



FCN



DeepLabv3+



U-net

DeepLabv3+

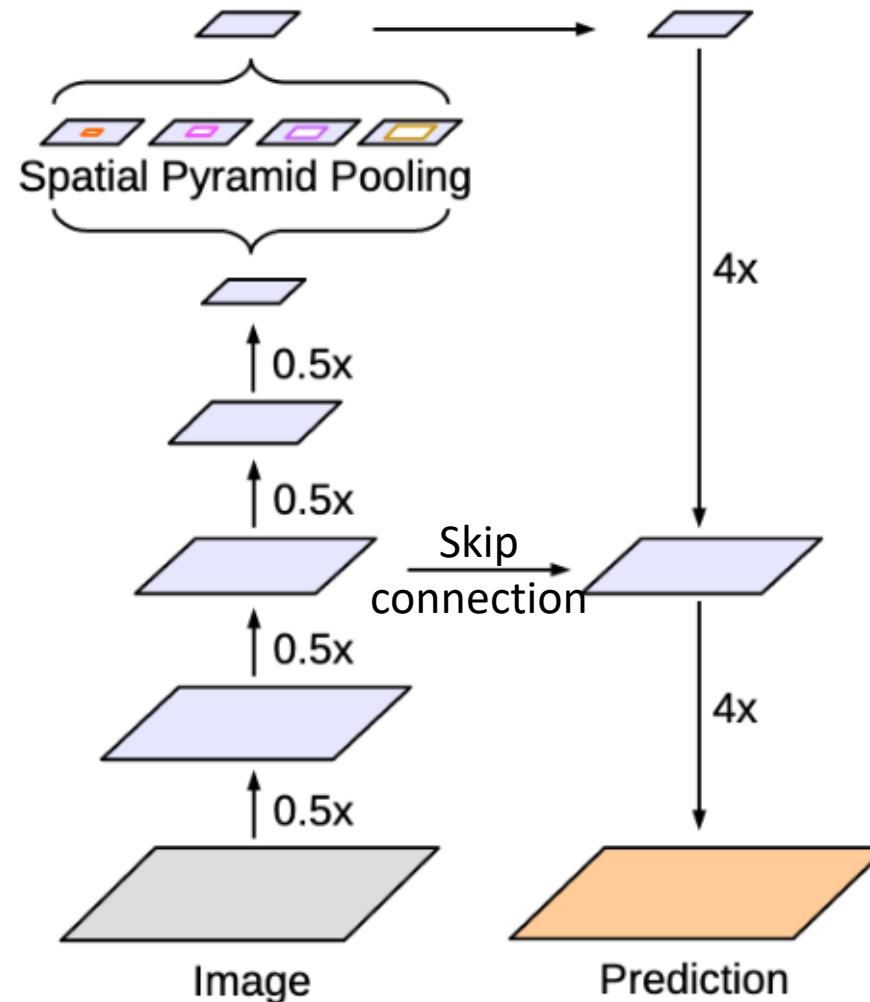
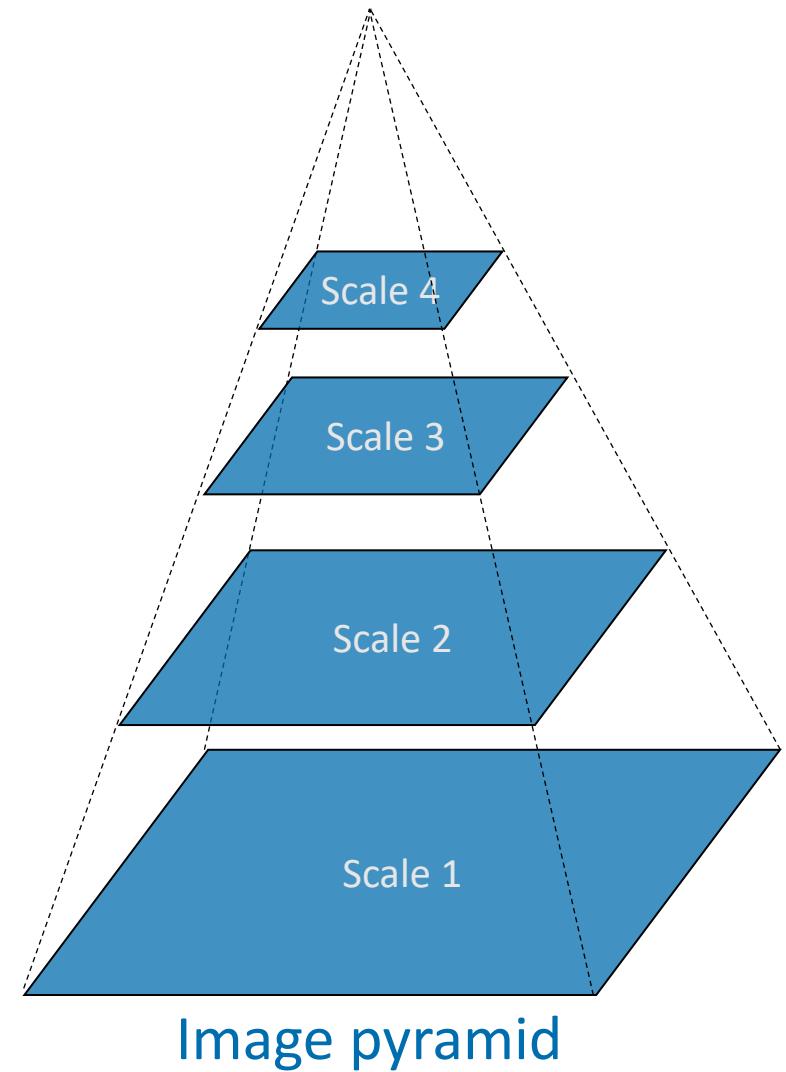
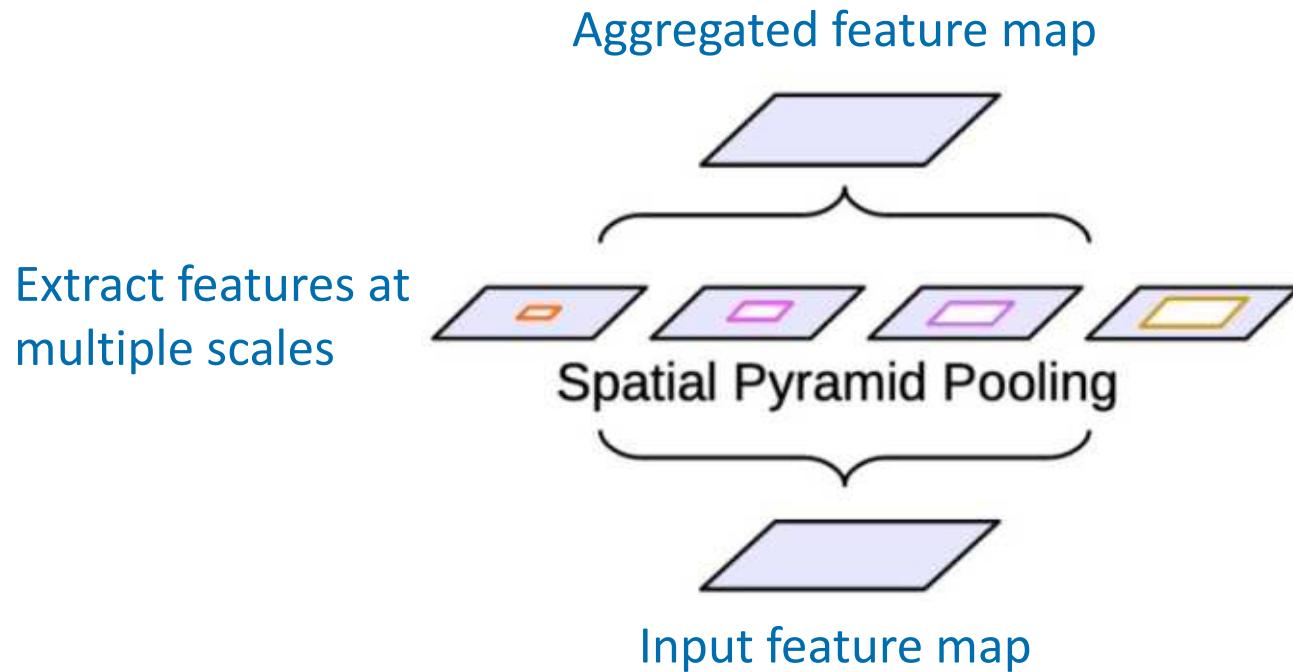


Image pyramid

- Representation of an image at multiple scales.
- In spatial pyramid pooling, features are extracted at multiple scales.



Spatial pyramid pooling



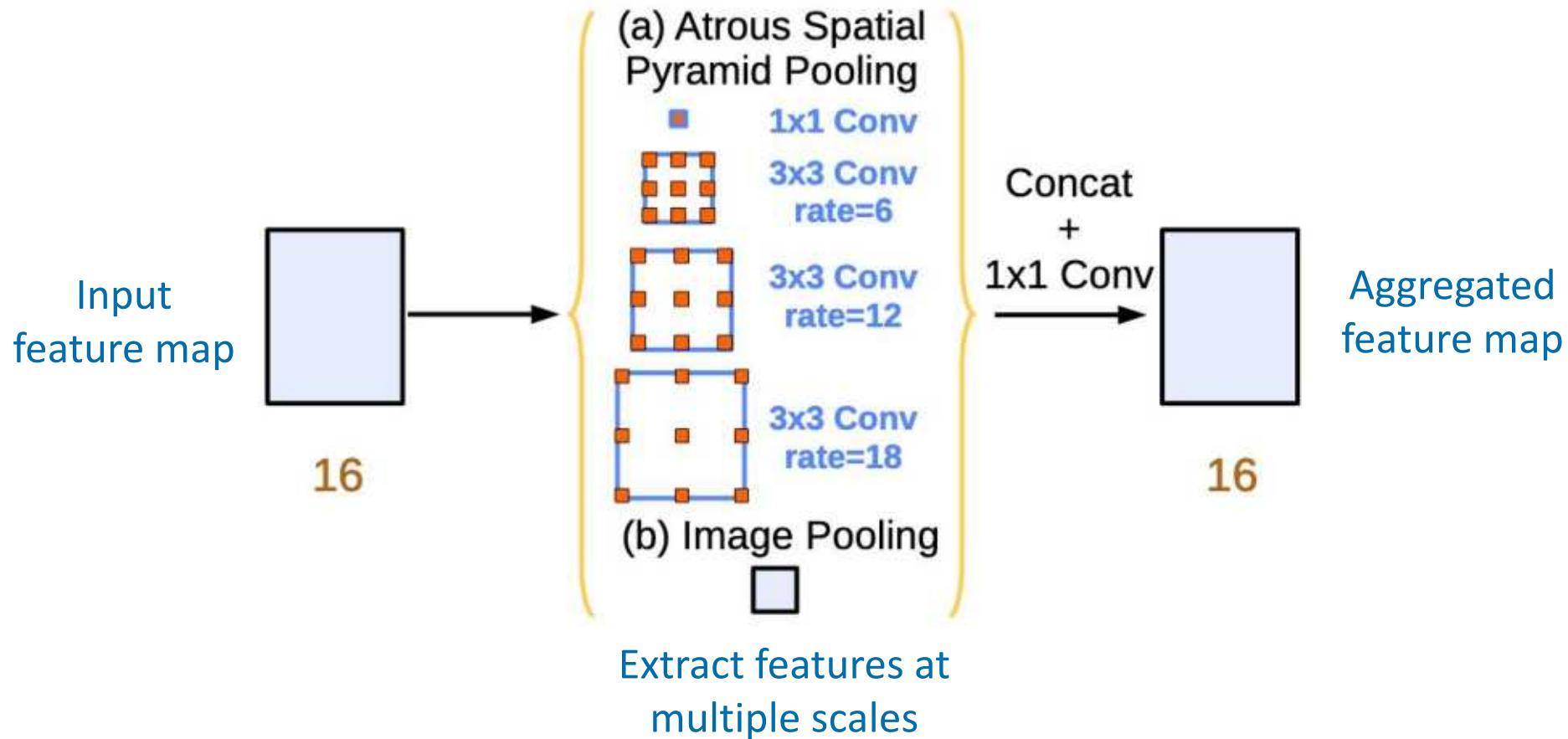
Multi-scale feature extraction

- How to extract features at multiple scales?
 - Convolutional kernels with different sizes
 - Convolutional kernels with different dilations
 - ...

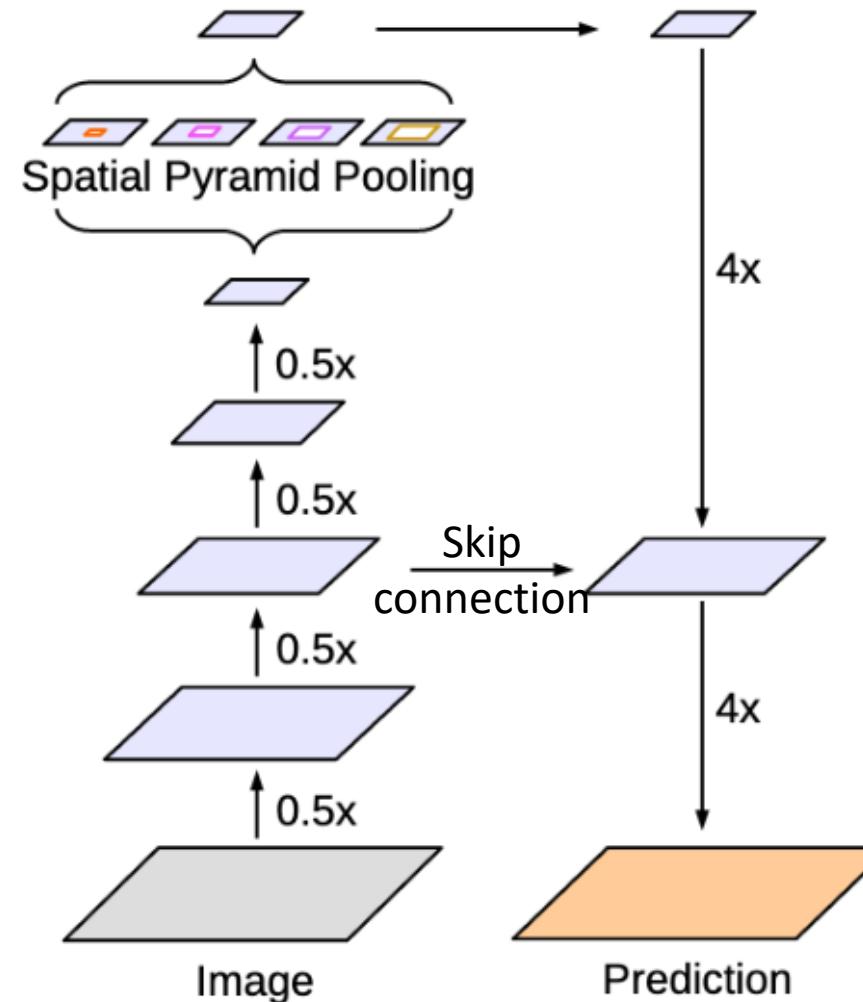
X		X		X				
X		X		X				
X		X		X				

3x3 convolutional kernel,
dilation = 2

Spatial pyramid pooling



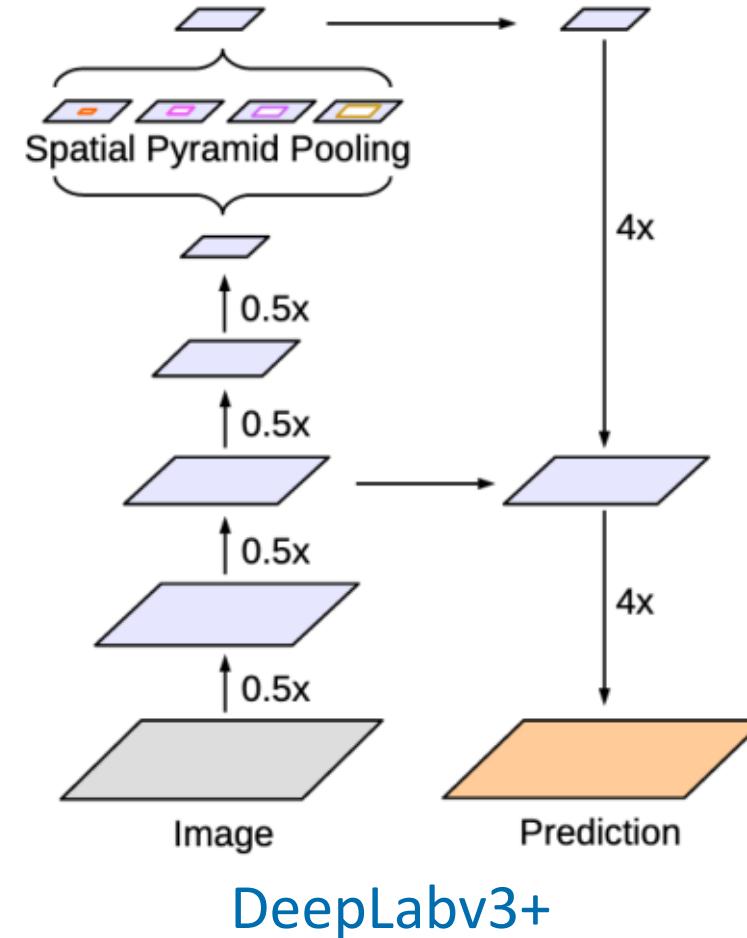
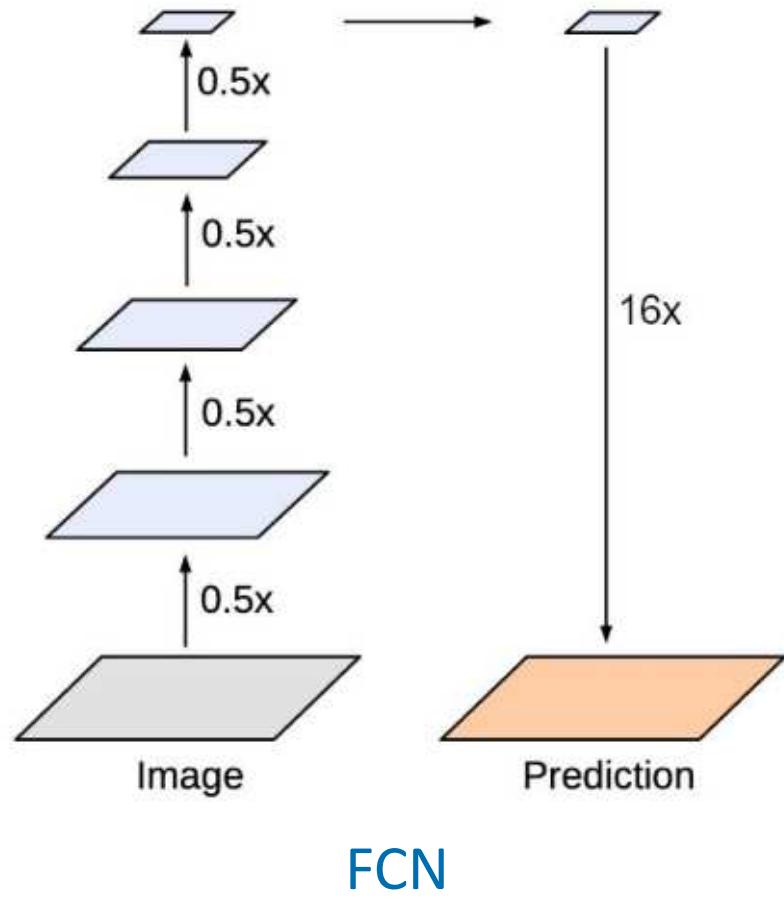
DeepLabv3+



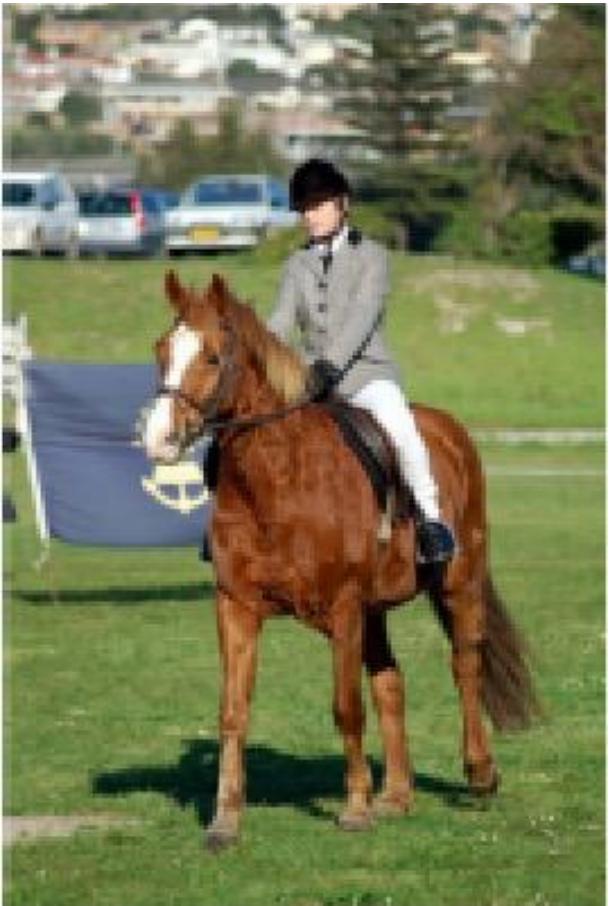
Skip connection concatenates local features with global features.

Prediction is performed on both local and global features.

Segmentation networks



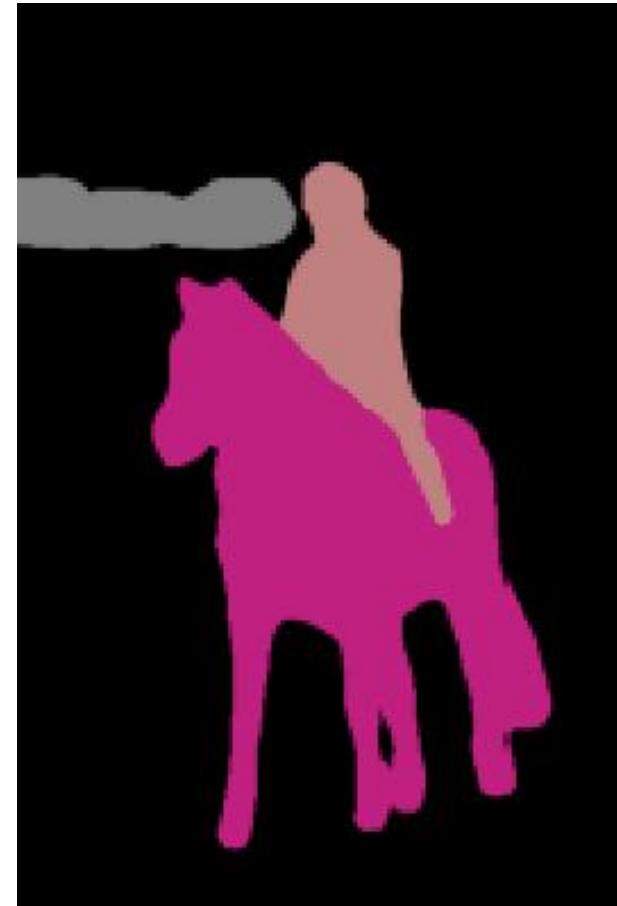
Segmentation results



Input image

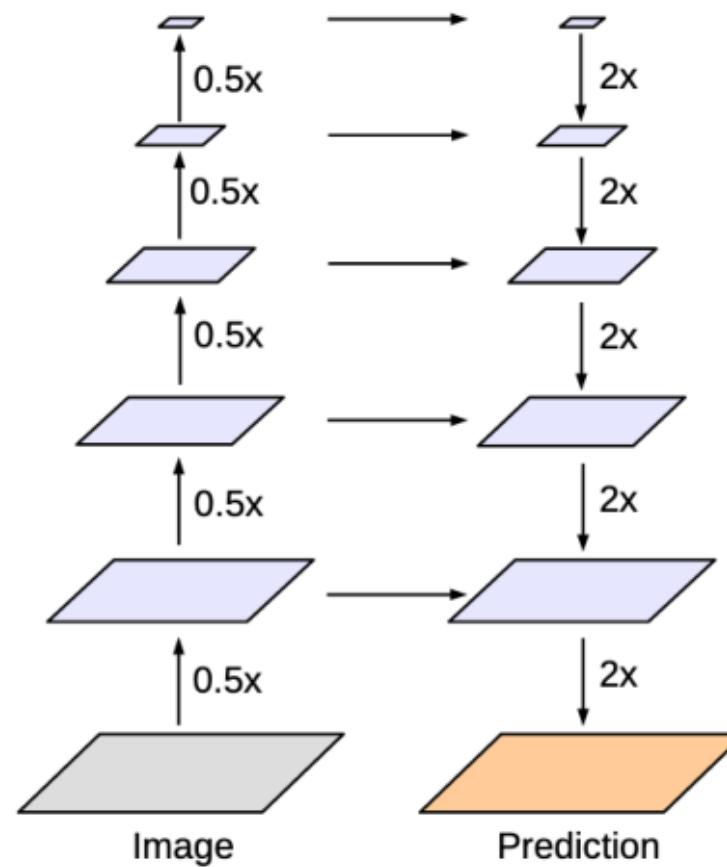


FCN



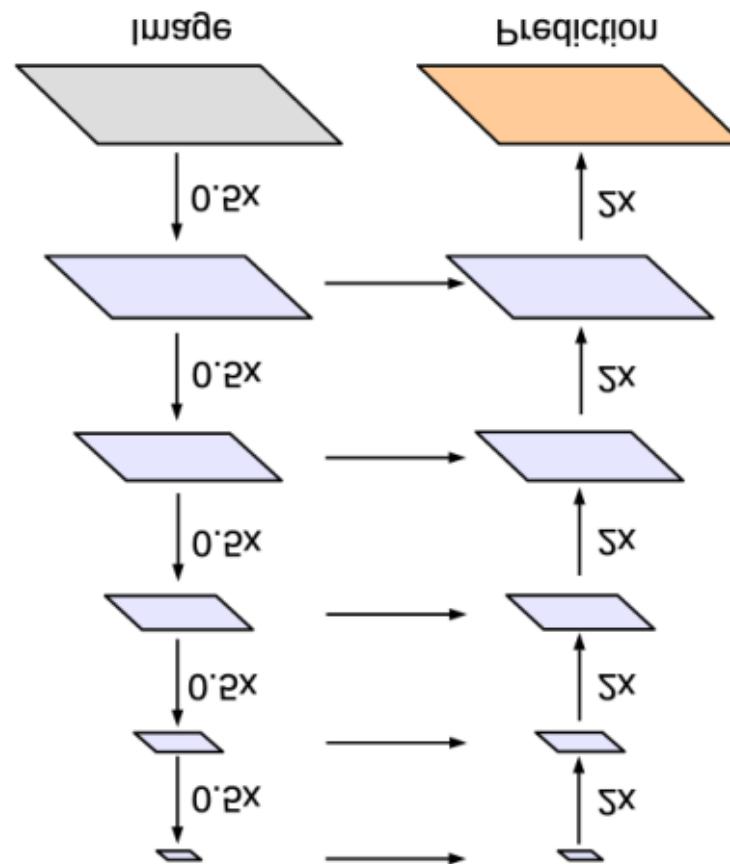
DeepLabv3+

U-net



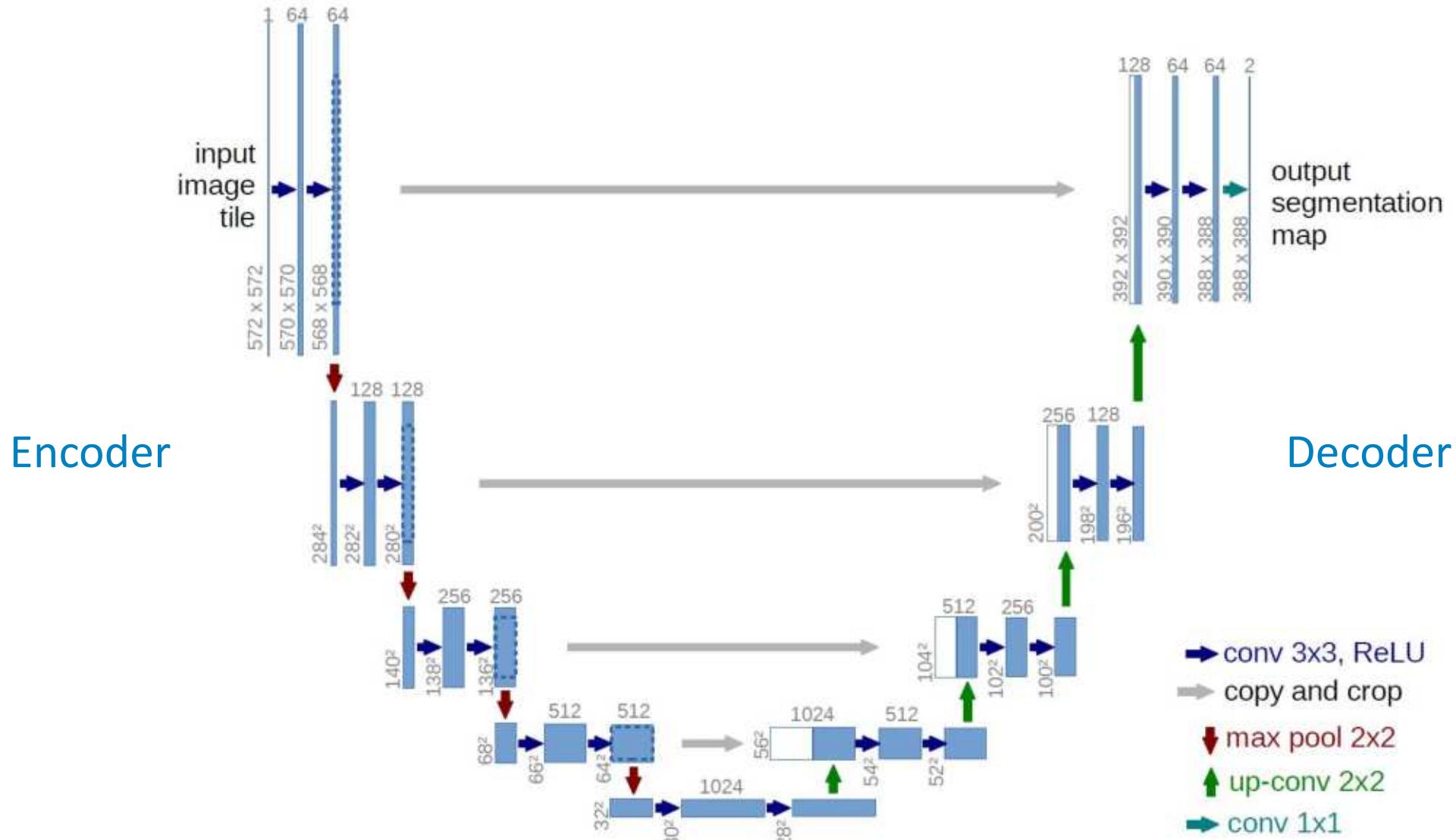
Skip connections at every level.

U-net



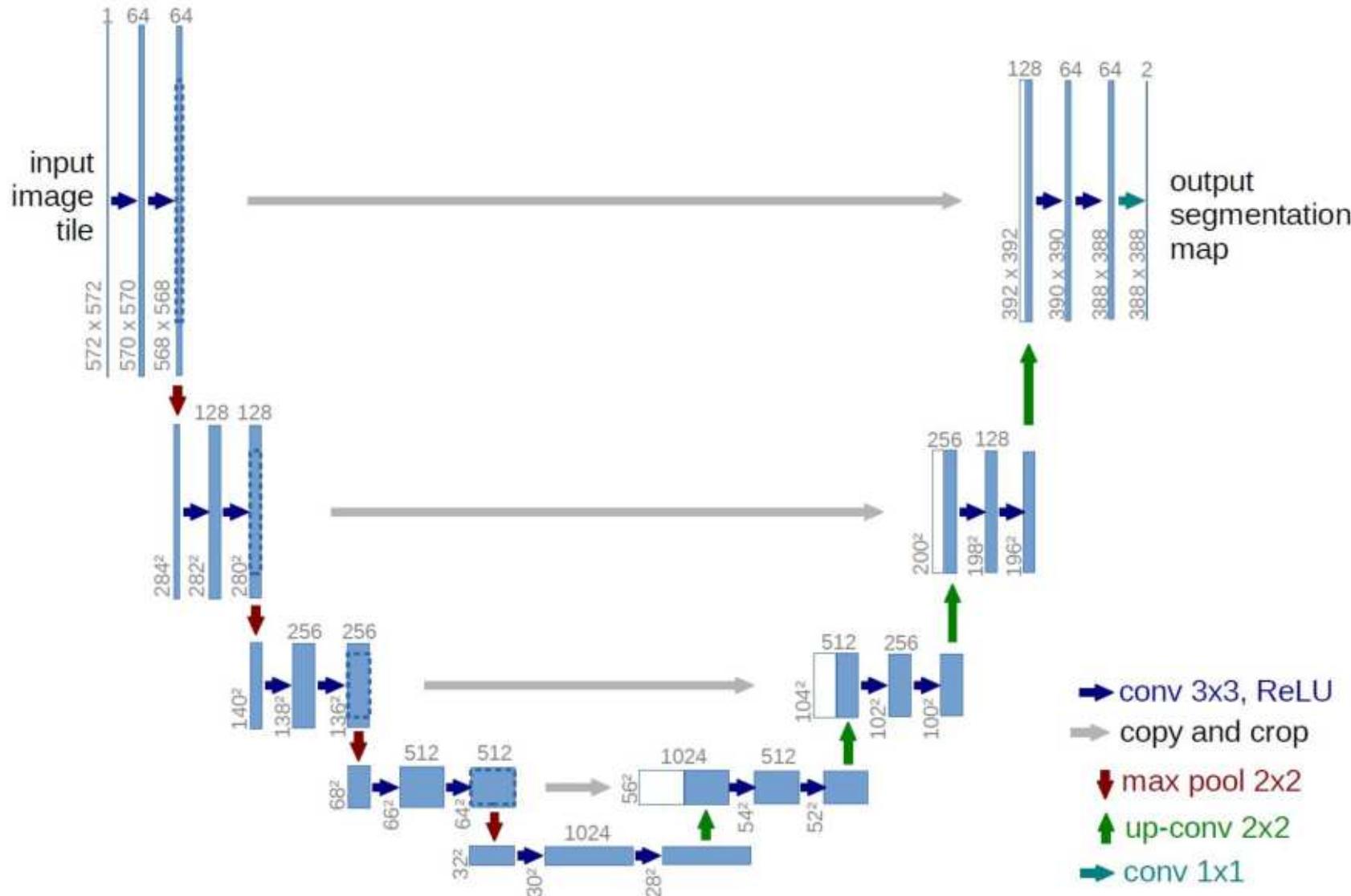
Like a U-shape.

U-net

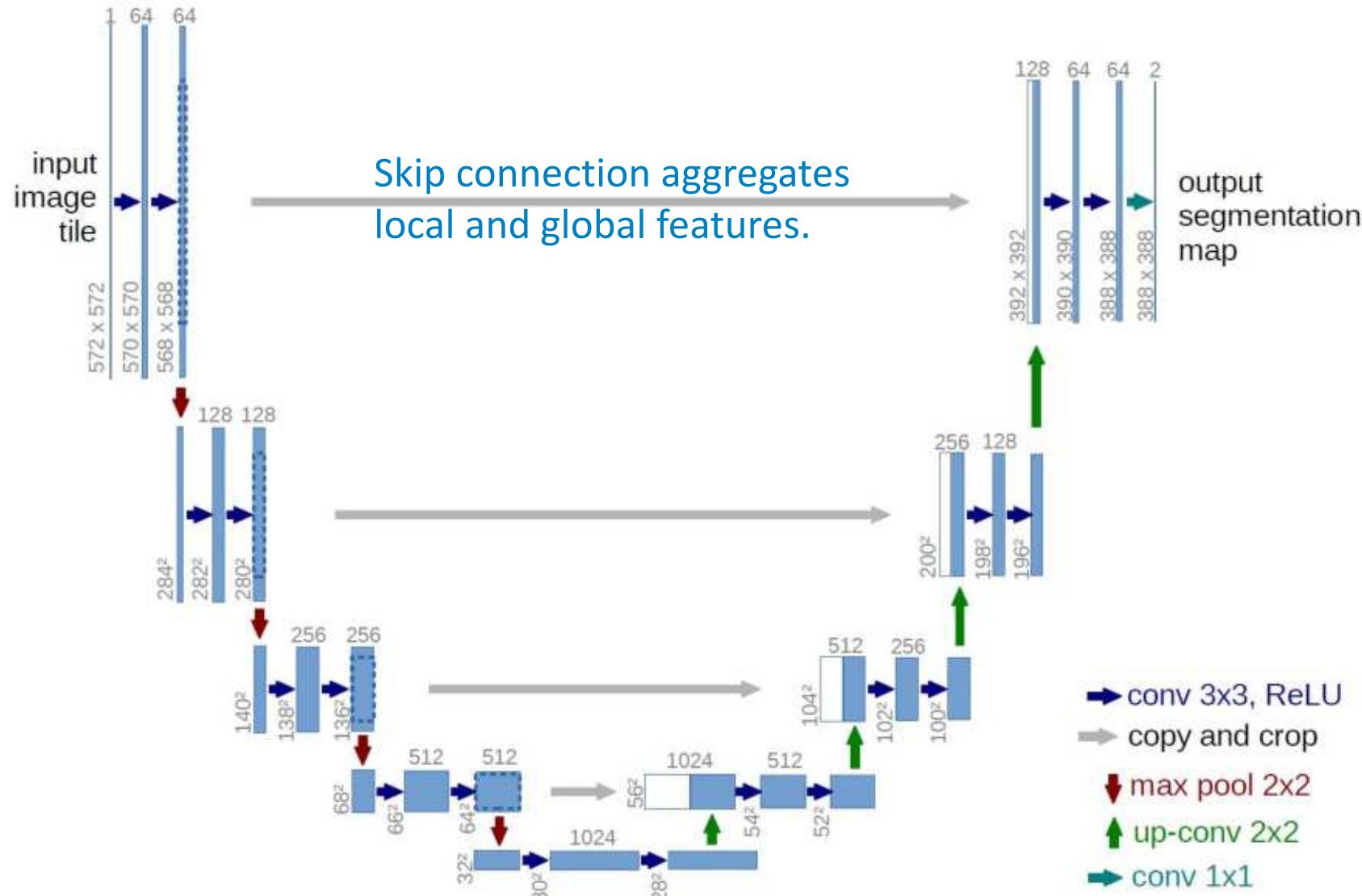


U-net

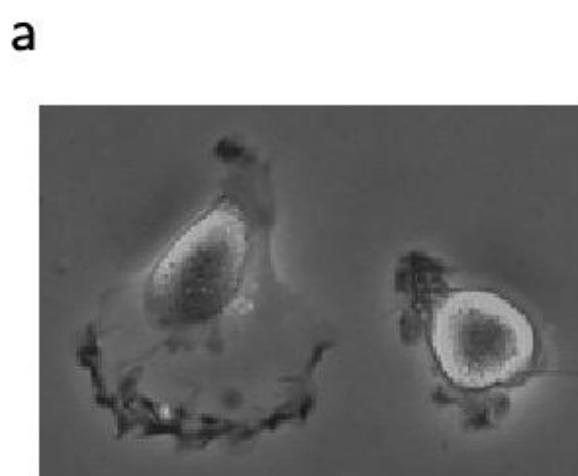
Local
feature



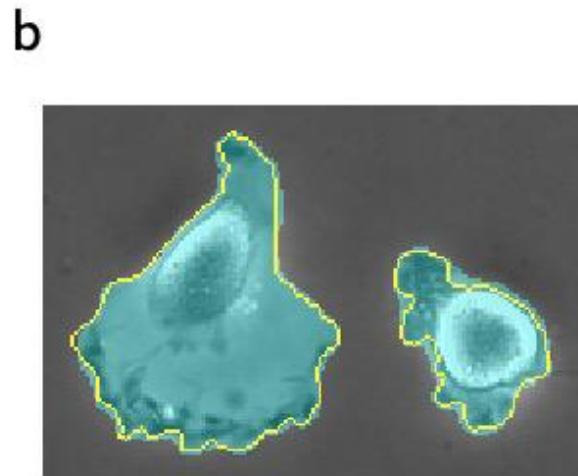
U-net



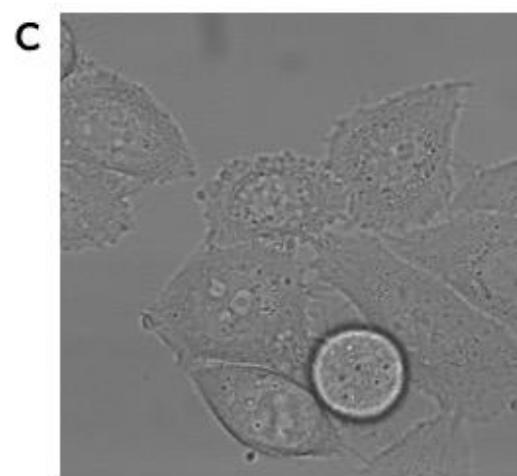
Segmentation results



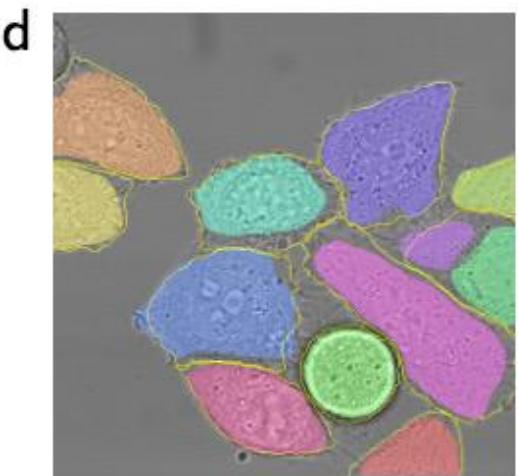
Input image



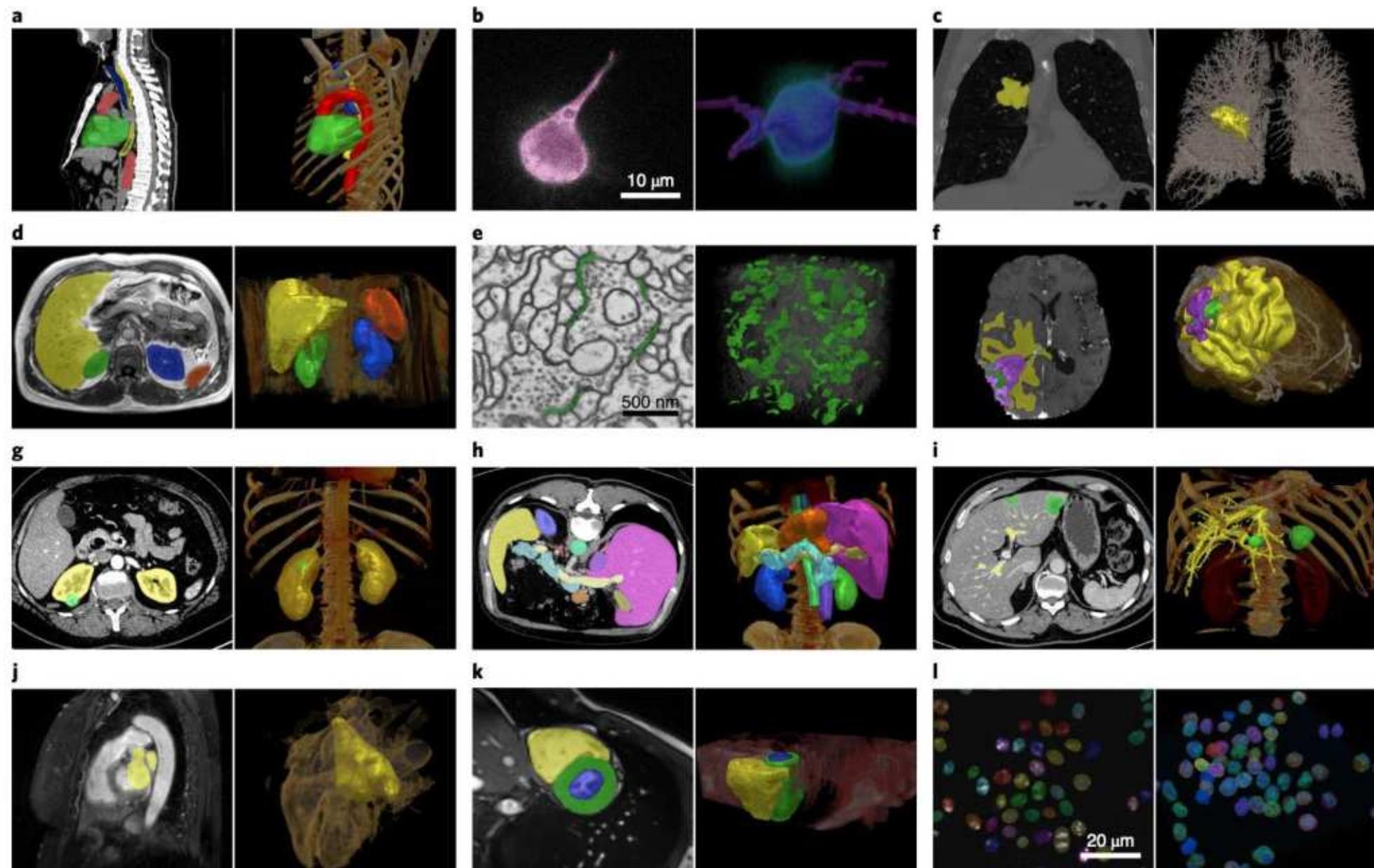
Segmentation compared to
ground truth (yellow contour).



Input image



Segmentation compared to
ground truth (yellow contour).

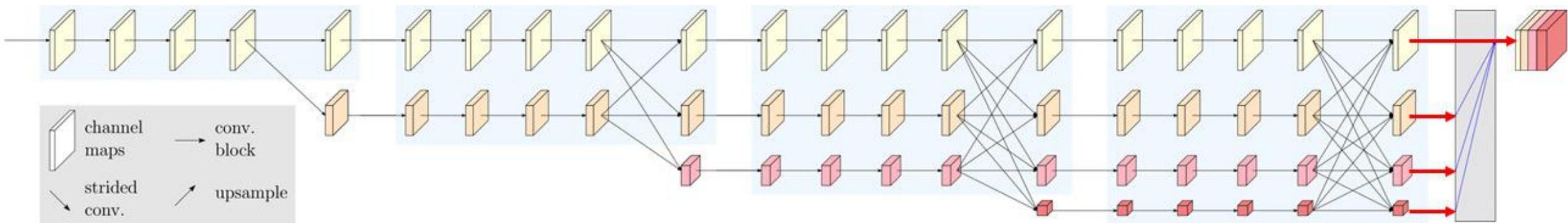


U-net segmentation results for different biomedical images.

Recent advances

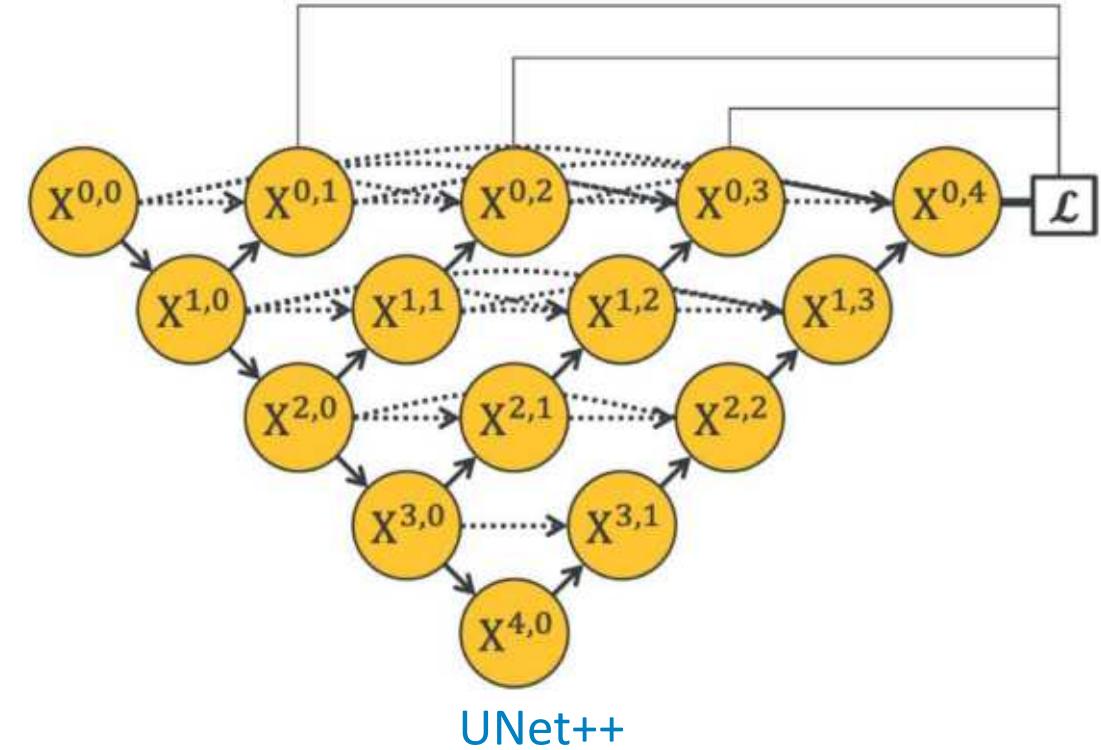
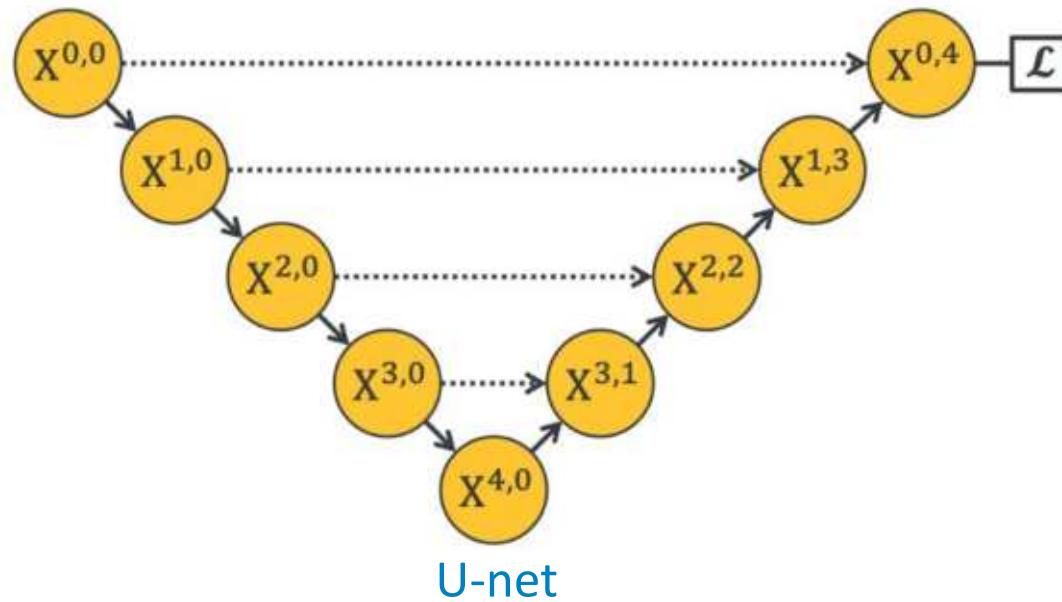
- Recent works on segmentation explores
 - different ways for combining global and local features
 - transformer architecture
 - defining things and stuff in the segmentation

HRNet



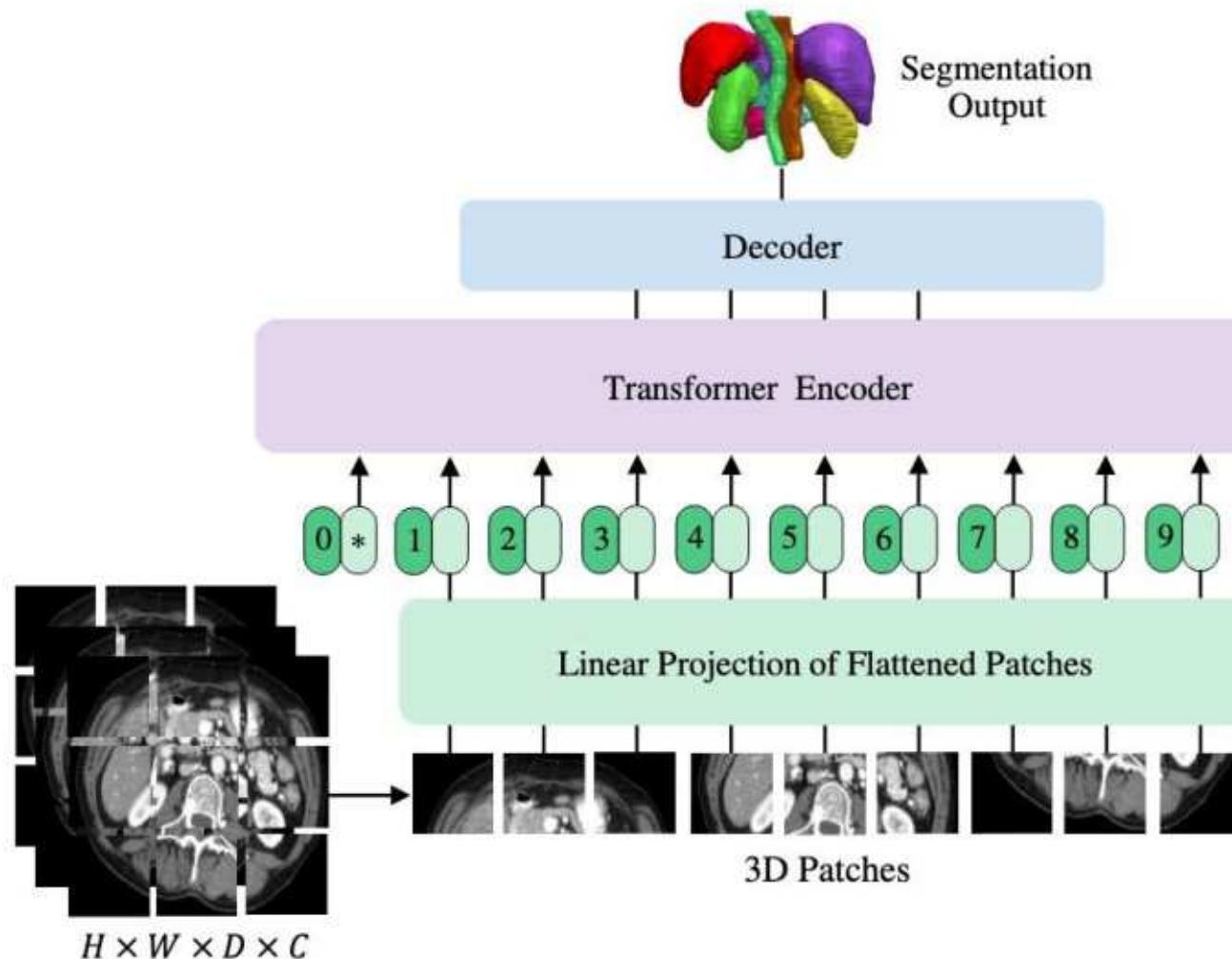
There are four resolution levels of features. High-resolution (local) features contribute to the learning of low-resolution (global) features. And vice versa.

UNet++



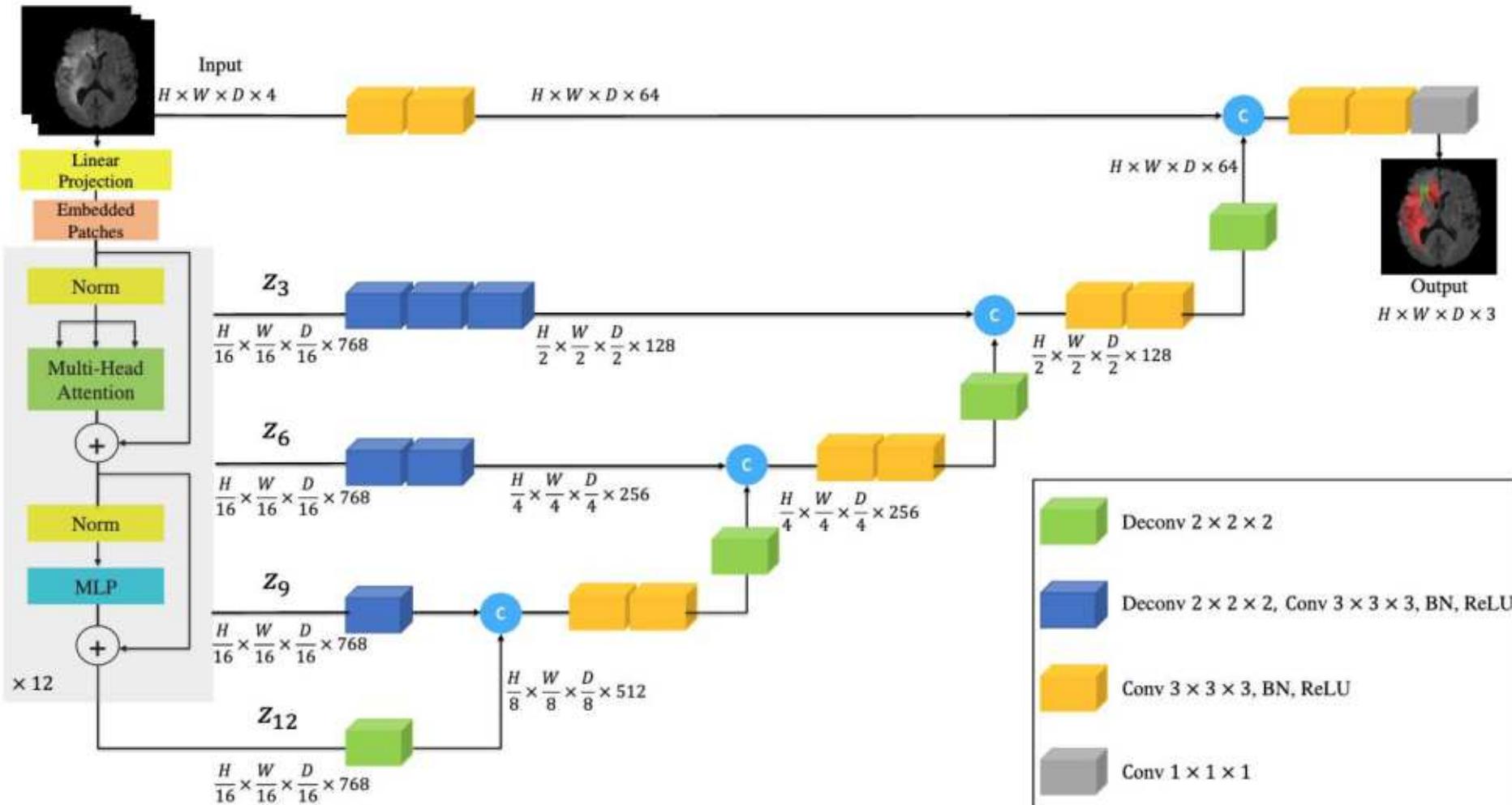
Features of multiple scales are fused multiple times.

UNETR



Vision transformer can learn features for image patches.
However, segmentation relies on pixel-wise features.

UNETR



A hybrid U-net + Transformer architecture can be developed, which aggregates local features learnt by convolutions and global features learnt by transformer.

Defining things

- Semantic segmentation assigns a class label to each pixel. However, it does not care whether these pixels form a single object (instance).
- Instance segmentation aims to define segmentation for each instance.

Instance segmentation



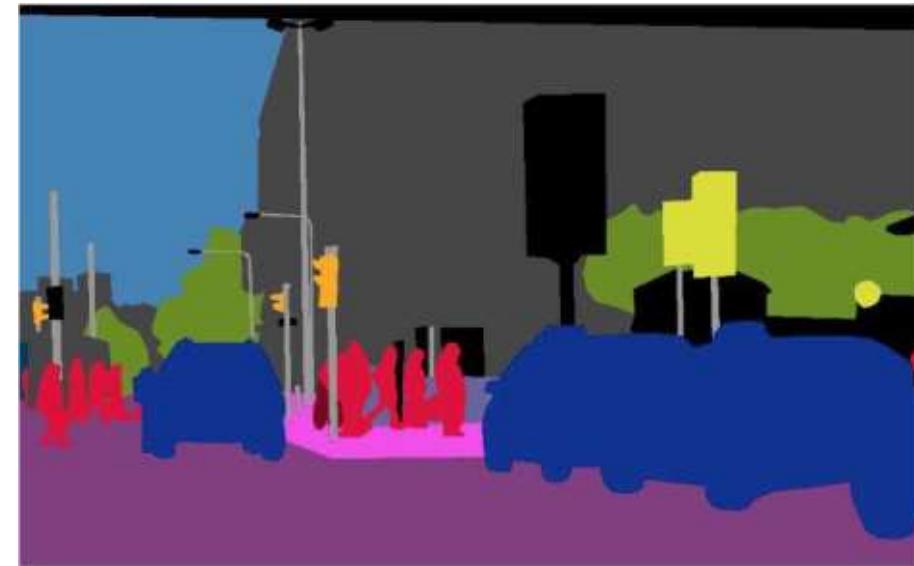
Often relies on object detection, followed by segmentation.

Panoptic segmentation

- Instance segmentation focuses on **countable things**, e.g. human, dogs, cars etc. It is mostly based on object detection, followed by segmentation.
- **Uncountable stuff**, such as sky, road etc , is ignored.
- Panoptic segmentation aims to unify things and stuff, providing segmentation for each thing (instance) and also segmentation for stuff.



Image



Semantic segmentation



Instance segmentation



Panoptic segmentation

Summary

- We explain two methods that fuse multi-scale features for segmentation.
 - DeepLabv3+
 - U-net
- We show some recent advances in the field.
 - Transformer architecture
 - Moving from classifying pixels to understanding what there are in the image, defining things and stuff.

Object Detection

Dr Wenjia Bai

Department of Computing & Brain Sciences

Image classification



Cat

Probability

Cat: 85%

Tiger: 10%

Dog: 1%

...

Object detection



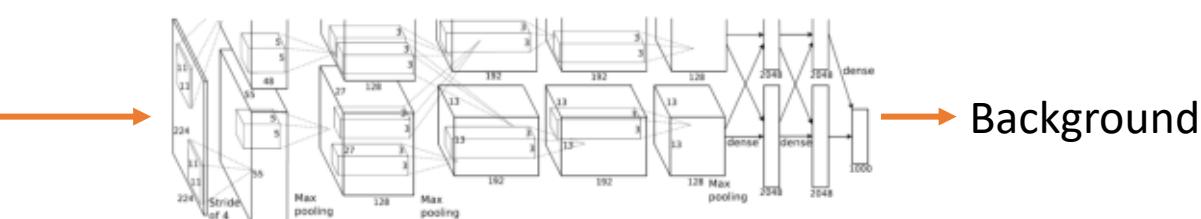
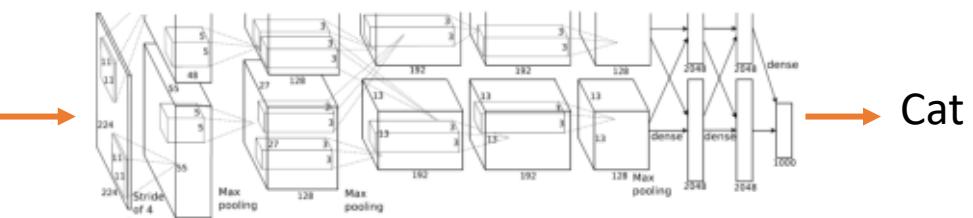
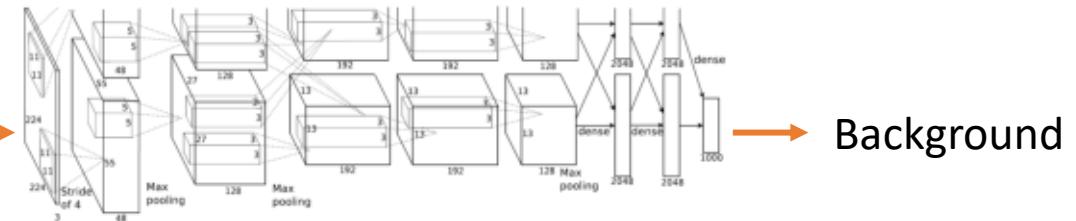
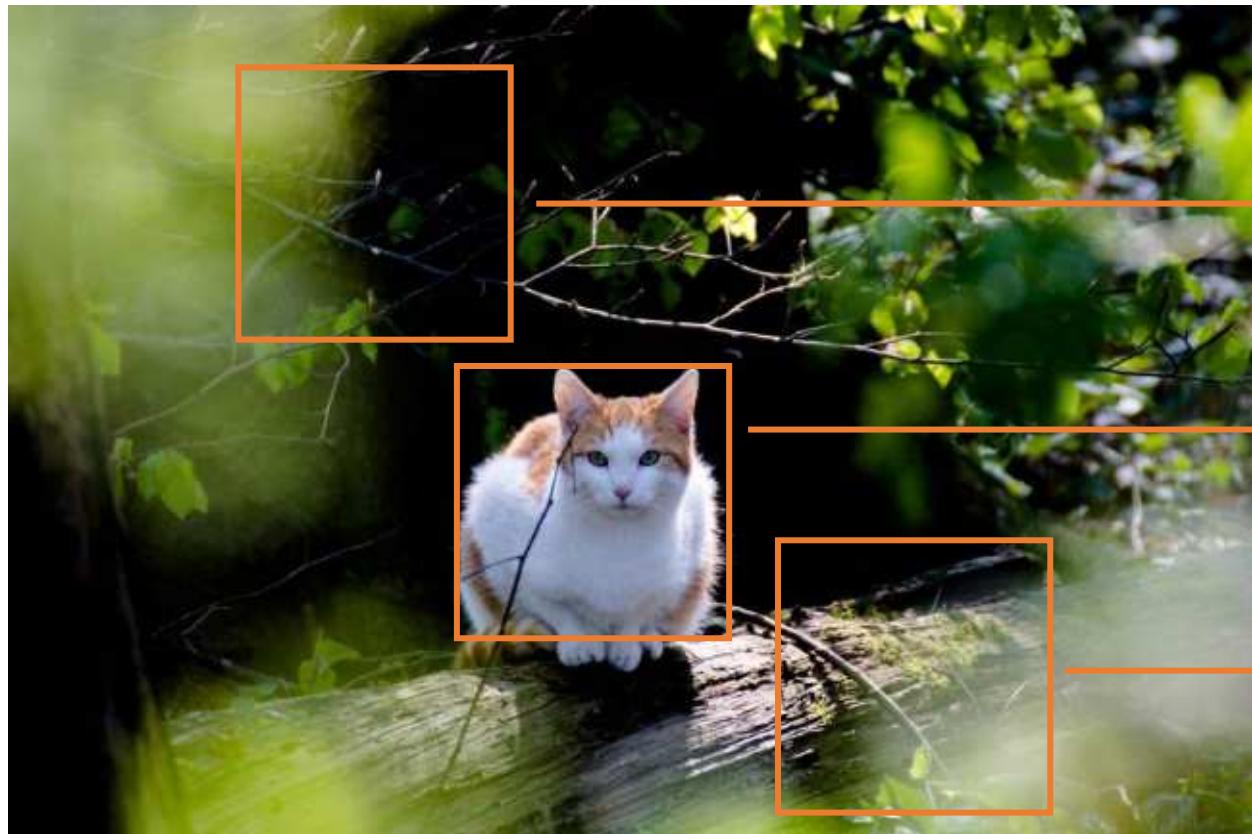
Cat

Bounding box

(x, y, w, h) denoting the top left (or centre) and width, height of the box

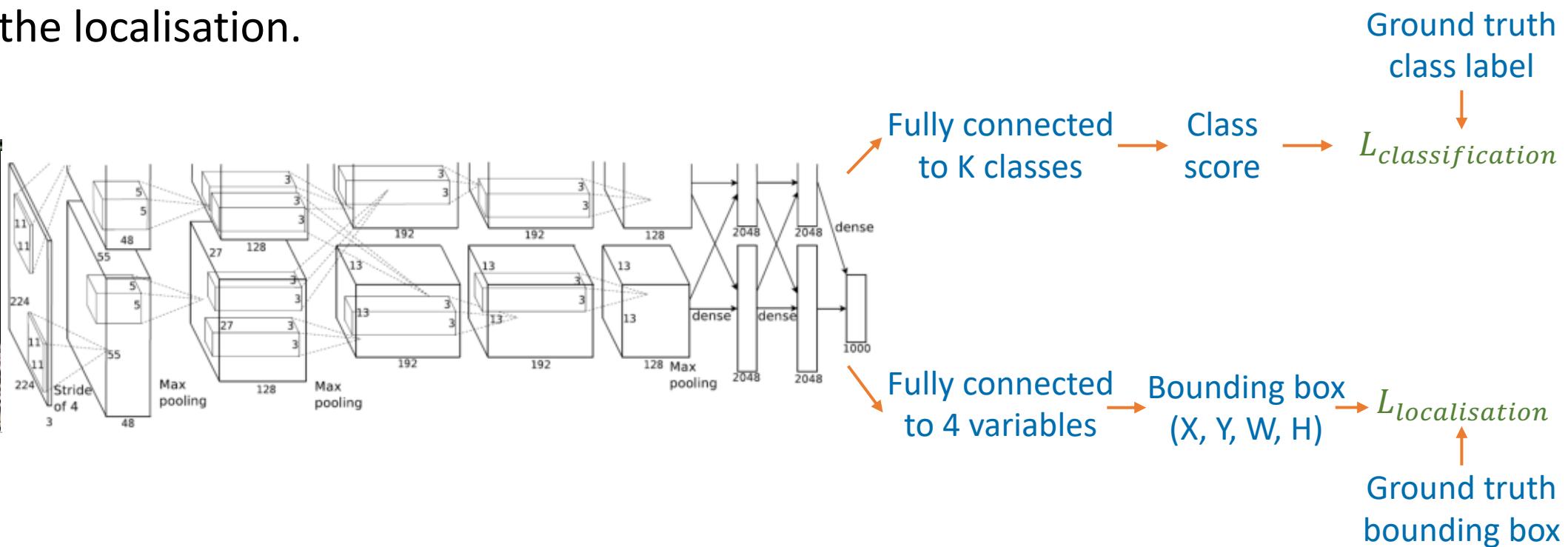
Object detection

- The goal is to predict a set of bounding boxes and labels for each object of interest.
- But how?
 - Same idea.
 - We have already built a model that can perform image classification.
 - How about applying this model to different regions of an image for detection?



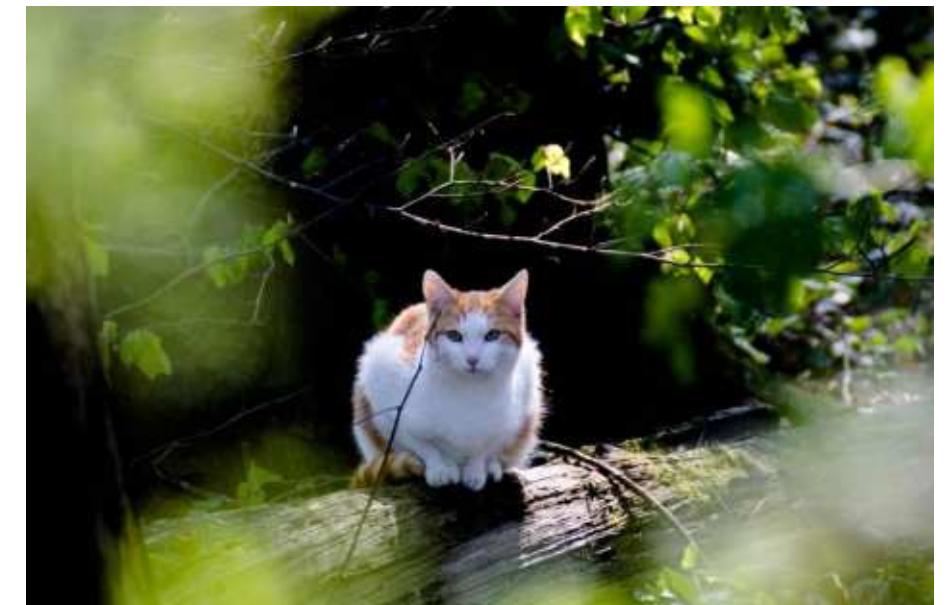
Idea for object detection

- At each location, we perform two tasks:
 - Task 1: classification (whether this is a cat or not)
 - Task 2: localisation (predict bounding box coordinates)
 - The sliding window already provides rough localisation. Using CNN features, we can refine the localisation.



Idea for object detection

- It may be too expensive to apply a convolutional network to each pixel on the image.
- What about having some initial guesses?



Object detection

- Two-stage detection
 - Stage 1: initial guess, propose some possible regions of interest
 - Stage 2: perform classification and localisation for these regions

Two-stage object detection

	Stage 1: Region proposal	Stage 2: Detector
R-CNN, 2014 [1]	Selective search	SVM for classification, linear regression for localization
Fast R-CNN, 2015 [2]	Selective search	CNN
Faster R-CNN, 2015 [3]	CNN	CNN
...		

R-CNN-based methods.

The name R-CNN stands for Regions with CNN features.

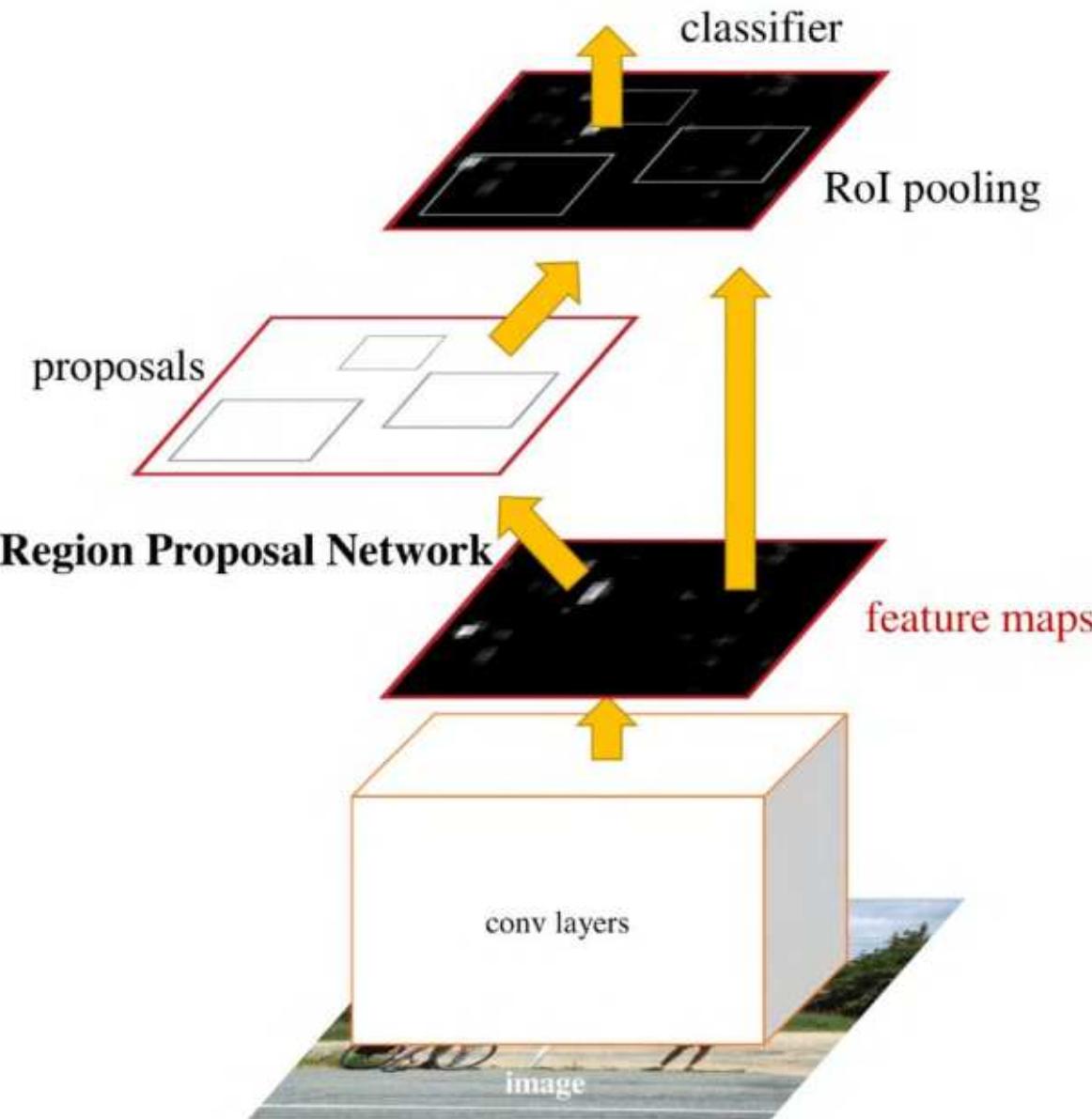
[1] Ross Girshick et al. Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR 2014.

[2] Ross Girshick. Fast R-CNN. ICCV 2015.

[3] Shaoqing Ren et al. Faster R-CNN: Towards real-time object detection with region proposal networks. NIPS 2015.

Two-stage object detection

- We will introduce Faster R-CNN, which employs CNN for both stages.
 - Stage 1: a region proposal network (RPN)
 - Stage 2: a detection network
- It is faster compared to standard R-CNN and Fast R-CNN, due to the efficiency of CNN at inference time.



Stage 1: A region proposal network is applied to the convolutional feature map to propose possible regions of interest.

What is this convolutional feature map?

AlexNet

[224x224x3] **Input**
[55x55x96] **Conv1**, 11x11, 96, s=4
[55x55x96] **Norm1**
[27x27x96] **Pool1**
[27x27x256] **Conv2**, 5x5, 256
[27x27x256] **Norm2**
[13x13x256] **Pool2**
[13x13x384] **Conv3**, 3x3, 384
[13x13x384] **Conv4**, 3x3, 384
[13x13x256] **Conv5**, 3x3, 256 convolutional feature map
[13x13x256] **Norm3**
[6x6x256] **Pool3**
[4096] **FC1**
[4096] **FC2**
[1000] **FC3** (class score)

VGG-16

[224x224x3] **Input**
[224x224x64] **3x3 conv1**, 64
[224x224x64] **3x3 conv1**, 64
[112x112x64] **Pool**
[112x112x128] **3x3 conv2**, 128
[112x112x128] **3x3 conv2**, 128
[56x56x128] **Pool**
[56x56x256] **3x3 conv3**, 256
[56x56x256] **3x3 conv3**, 256
[56x56x256] **3x3 conv3**, 256
[28x28x256] **Pool**
[28x28x512] **3x3 conv4**, 512
[28x28x512] **3x3 conv4**, 512
[28x28x512] **3x3 conv4**, 512
[14x14x512] **Pool**
[14x14x512] **3x3 conv5**, 512
[14x14x512] **3x3 conv5**, 512
[14x14x512] **3x3 conv5**, 512 convolutional feature map
[7x7x512] **Pool**
[4096] **FC**, 4096
[4096] **FC**, 4096
[1000] **FC**, 1000

These are called **backbone networks**, which provide the convolutional feature maps. They are often pre-trained on ImageNet or other datasets.

Convolutional feature map

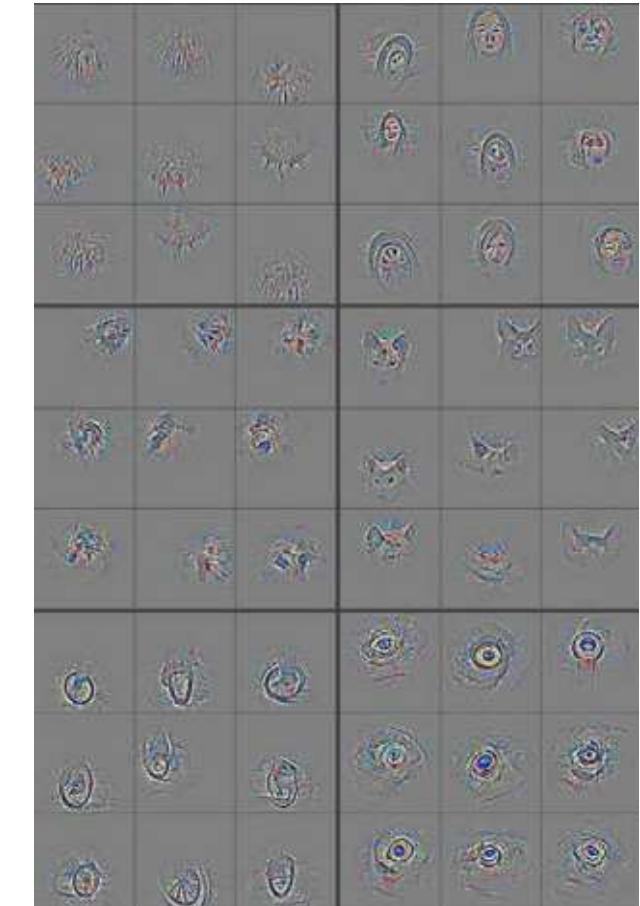
- For VGG net, there are 4 max pooling layers before conv5.
- Therefore, the first two dimensions of the feature map is $X/16 \times Y/16$.
 - If the input image is 224×224 , then conv5 feature map is $14 \times 14 \times D$.
 - D is the depth dimension or the number of filters. For conv5 in VGG net, $D = 512$.

Convolutional feature map

- Each pixel of the feature map is a high-dimensional feature vector, which describes the content of a small region in the input image.
 - This feature vector may tell us whether this is an interesting region or not.



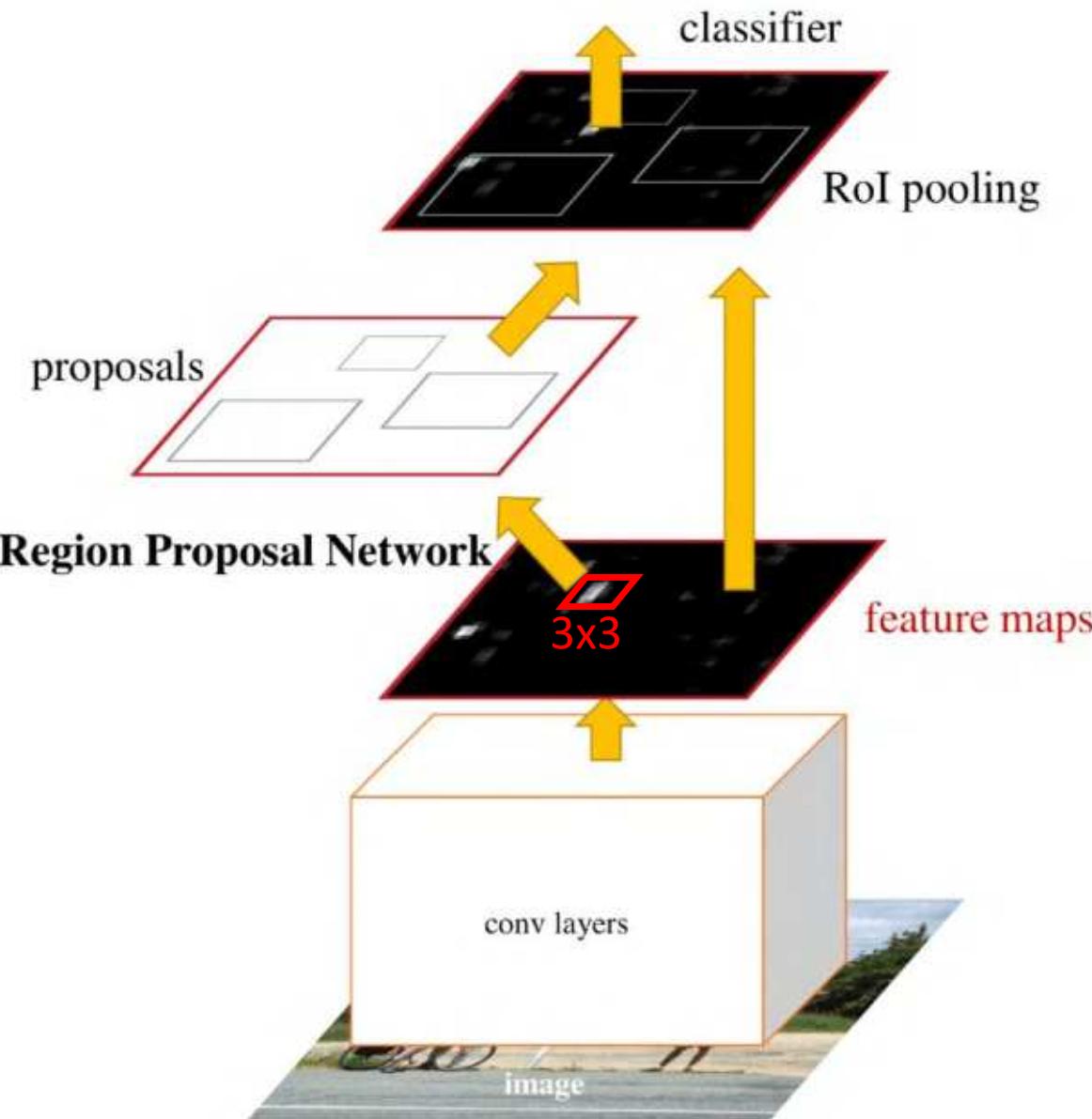
Input images



Visualisation of Conv5 neurons

Region proposal network

- To proposing regions, we go across the convolutional feature map using a 3×3 sliding window.
- Using features at this window, we perform a binary classification.
 - 0: not interesting
 - 1: interesting
- Also, we want to predict the size of this object of interest.

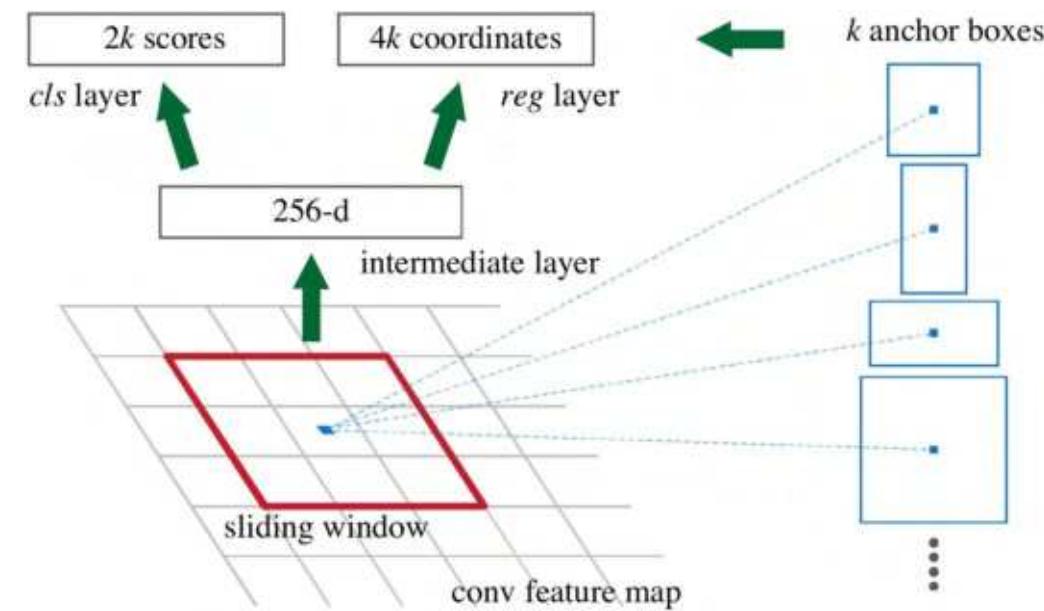


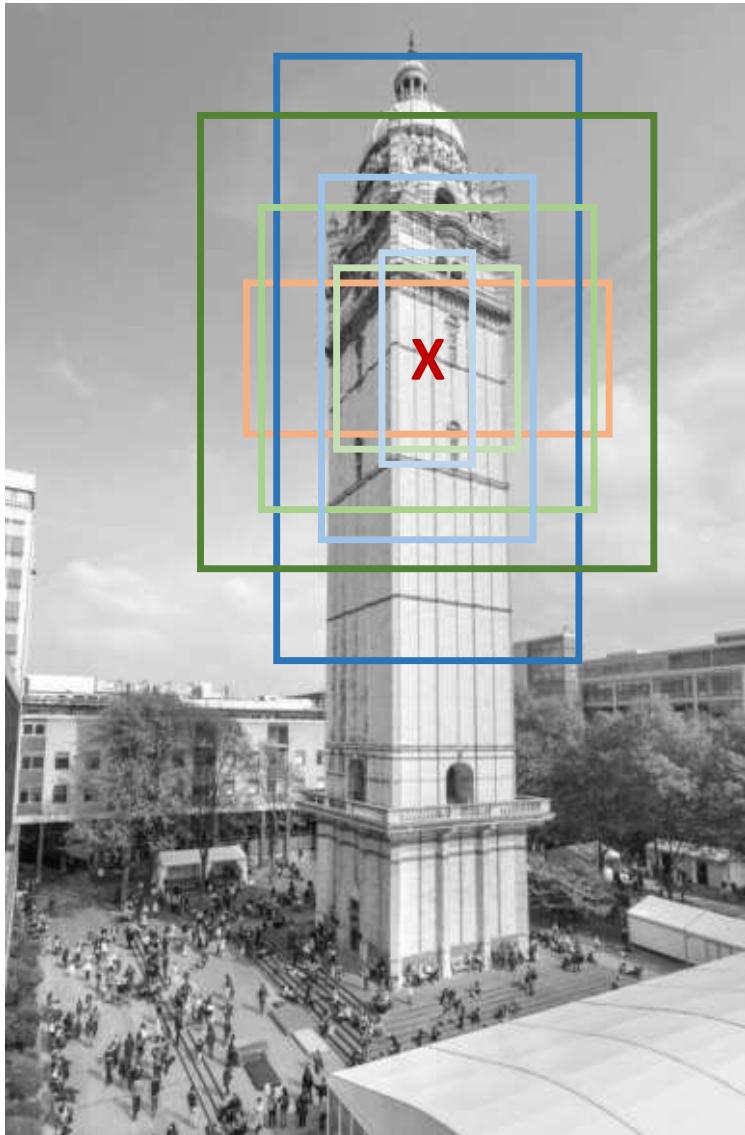


We want to predict the size for objects of interest.

Region proposal network

- At each sliding window, the network makes k predictions, for objects of different scales and aspect ratios.
 - 3 scales: $128^2, 256^2, 512^2$
 - 3 aspect ratios: 1:1, 1:2, 2:1
 - In total, $k = 3 \times 3 = 9$
 - These bounding boxes are called “anchors”.
- Each predictor is trained to tell objects of a specific scale and aspect ratio.





At location **X**, **the first predictor** tells us whether it is a small and long object.

The second predictor tells us whether it is a medium and long object.

The third predictor tells us whether it is a large and long object.

The fourth predictor tells us whether it is a small and square object.

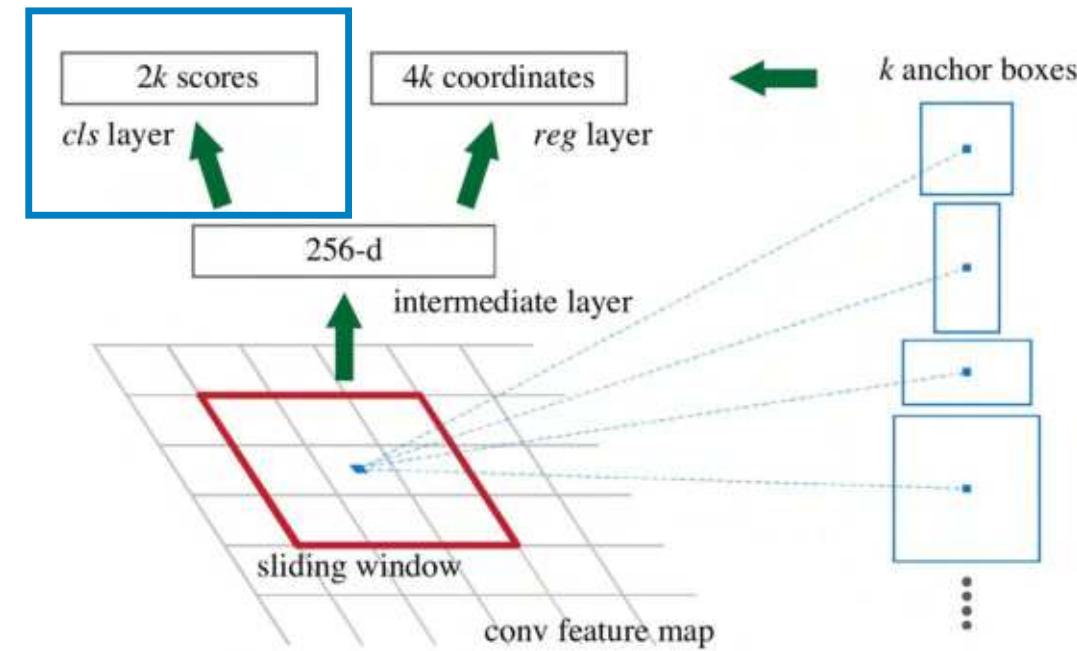
.....

At each location, the network predicts a score for each anchor (e.g. 128x128 anchor, 128x256 anchor, 128x64 anchor, 256x256 anchor ...).

Region proposal network

- At each sliding window, the network makes k predictions, for objects of different scales and aspect ratios.
 - 3 scales: $128^2, 256^2, 512^2$
 - 3 aspect ratios: 1:1, 1:2, 2:1
 - In total, $k = 3 \times 3 = 9$
 - These bounding boxes are called “anchors”.
 - Each predictor is trained to tell objects of a specific scale and aspect ratio.

The predictors are trained here.



Region proposal network

- The loss is defined as

$$L(p, t) = \sum_{i=1}^{n_{anchor}} L_{cls}(p_i, p_i^*) + \lambda \cdot \sum_{i=1}^{n_{anchor}} 1_{y=1} L_{loc}(t_i, t_i^*)$$

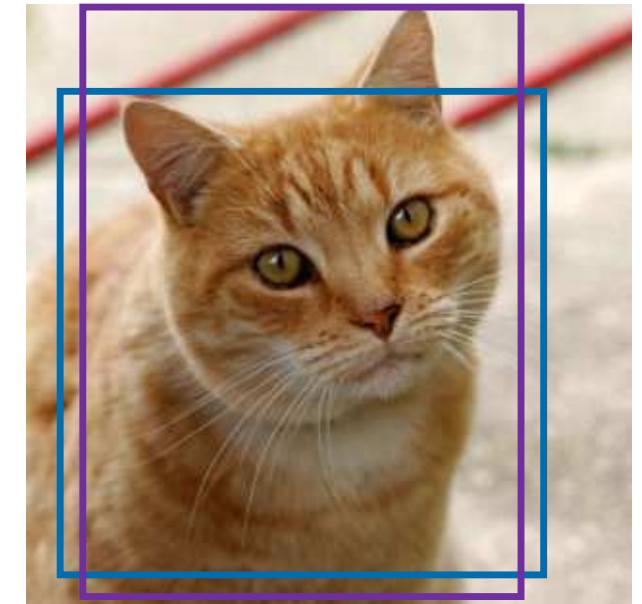
predicted score
for this anchor ground truth,
binary value

Classification loss **Localisation loss**

- Now we can tell whether this region is interesting or not, whether it is small or large. Can we also describe its bounding box?

How do we describe a bounding box?

- A bounding box can be described by its centre (x, y) and size (w, h).
- We can learn to predict these 4 coordinates and compare to ground truth.

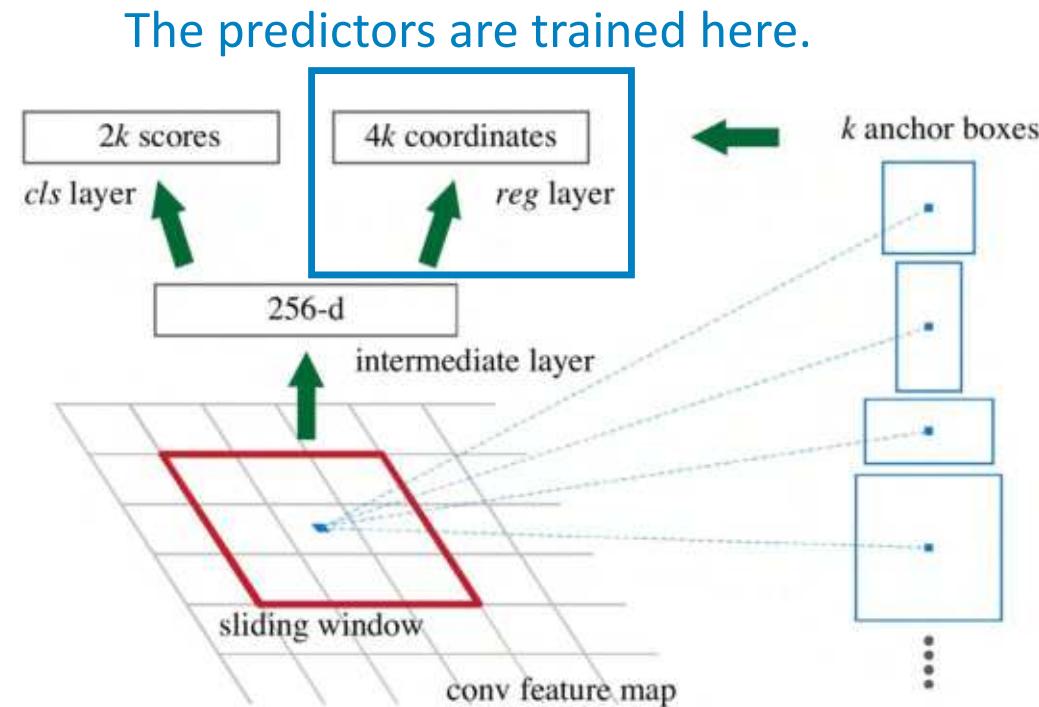


Blue box: prediction

Purple box: ground truth

Region proposal network

- At each sliding window, the network makes k predictions, for objects of different scales and aspect ratios.
 - 3 scales: $128^2, 256^2, 512^2$
 - 3 aspect ratios: 1:1, 1:2, 2:1
 - In total, $k = 3 \times 3 = 9$
 - These bounding boxes are called “anchors”.
- Each predictor is trained to tell objects of a specific scale and aspect ratio.



Region proposal network

- The loss is defined as

$$L(p, t) = \sum_{i=1}^{n_{anchor}} L_{cls}(p_i, p_i^*) + \lambda \cdot \sum_{i=1}^{n_{anchor}} 1_{y=1} L_{loc}(t_i, t_i^*)$$

predicted score
for this anchor ground truth,
binary value

Classification loss

predicted
bounding box ground truth
bounding box

Localisation loss

- In the practical implementation, relative location and size is predicted, instead of the absolute coordinates of the bounding box.

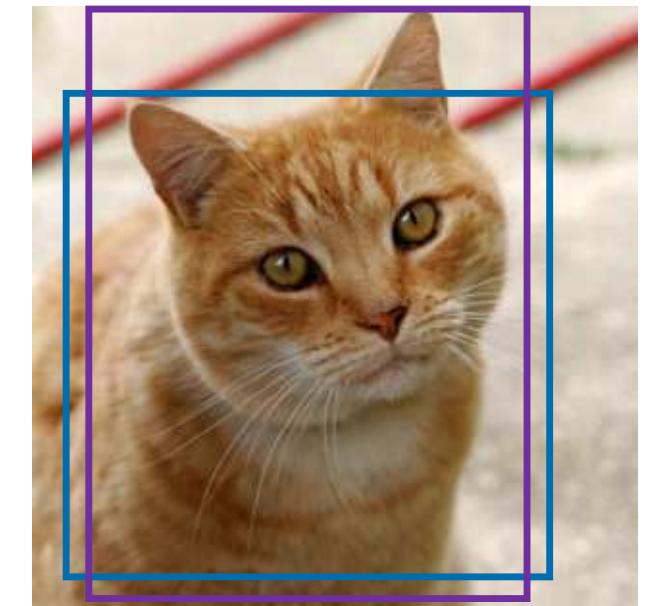
How do we describe a bounding box?

- There are k predictions being made, each for one specific anchor.
- We describe the predicted bounding box by its relative location and size, compared to this anchor.

- Anchor: (x_a, y_a, w_a, h_a)

- Predicted bounding box: (x, y, w, h)

- Predicted transformation t : (t_x, t_y, t_w, t_h)

$$t_x = \frac{x - x_a}{w_a}$$
$$t_y = \frac{y - y_a}{h_a}$$
$$t_w = \log\left(\frac{w}{w_a}\right)$$
$$t_h = \log\left(\frac{h}{h_a}\right)$$


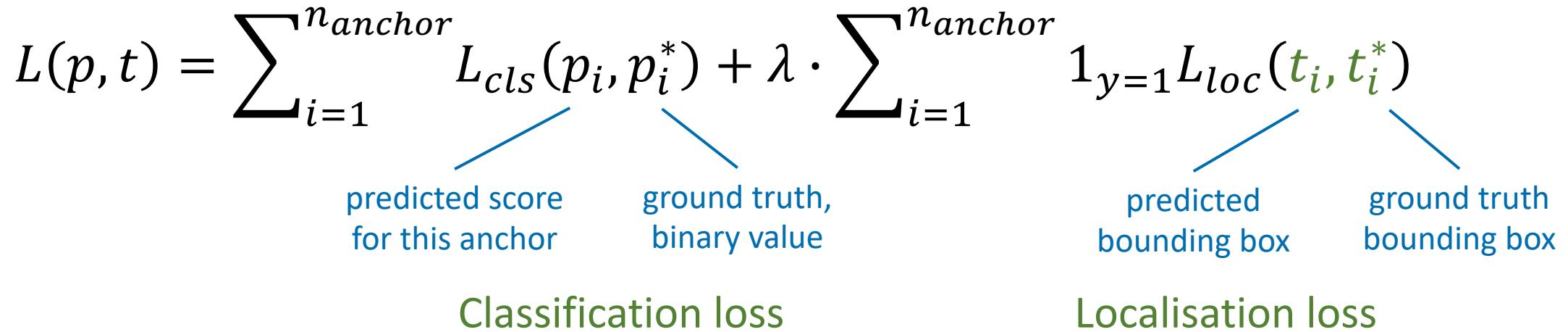
Blue box: prediction

Purple box: ground truth

Region proposal network

- The loss is defined as

$$L(p, t) = \sum_{i=1}^{n_{anchor}} L_{cls}(p_i, p_i^*) + \lambda \cdot \sum_{i=1}^{n_{anchor}} 1_{y=1} L_{loc}(t_i, t_i^*)$$


Classification loss Localisation loss

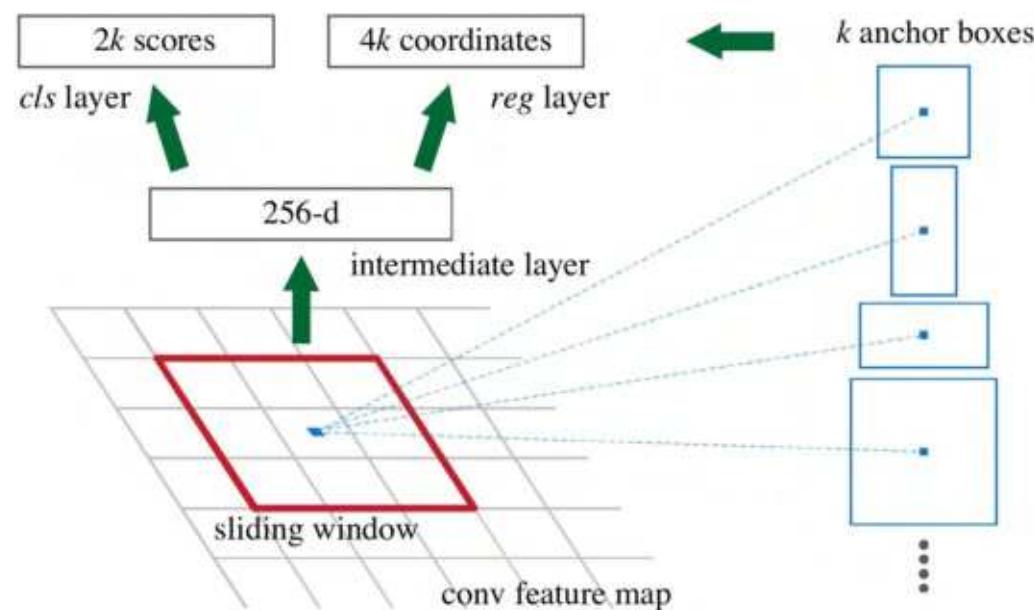
predicted score for this anchor ground truth, binary value

predicted bounding box ground truth bounding box

- In the practical implementation, relative location and size is predicted, instead of the absolute coordinates of the bounding box.

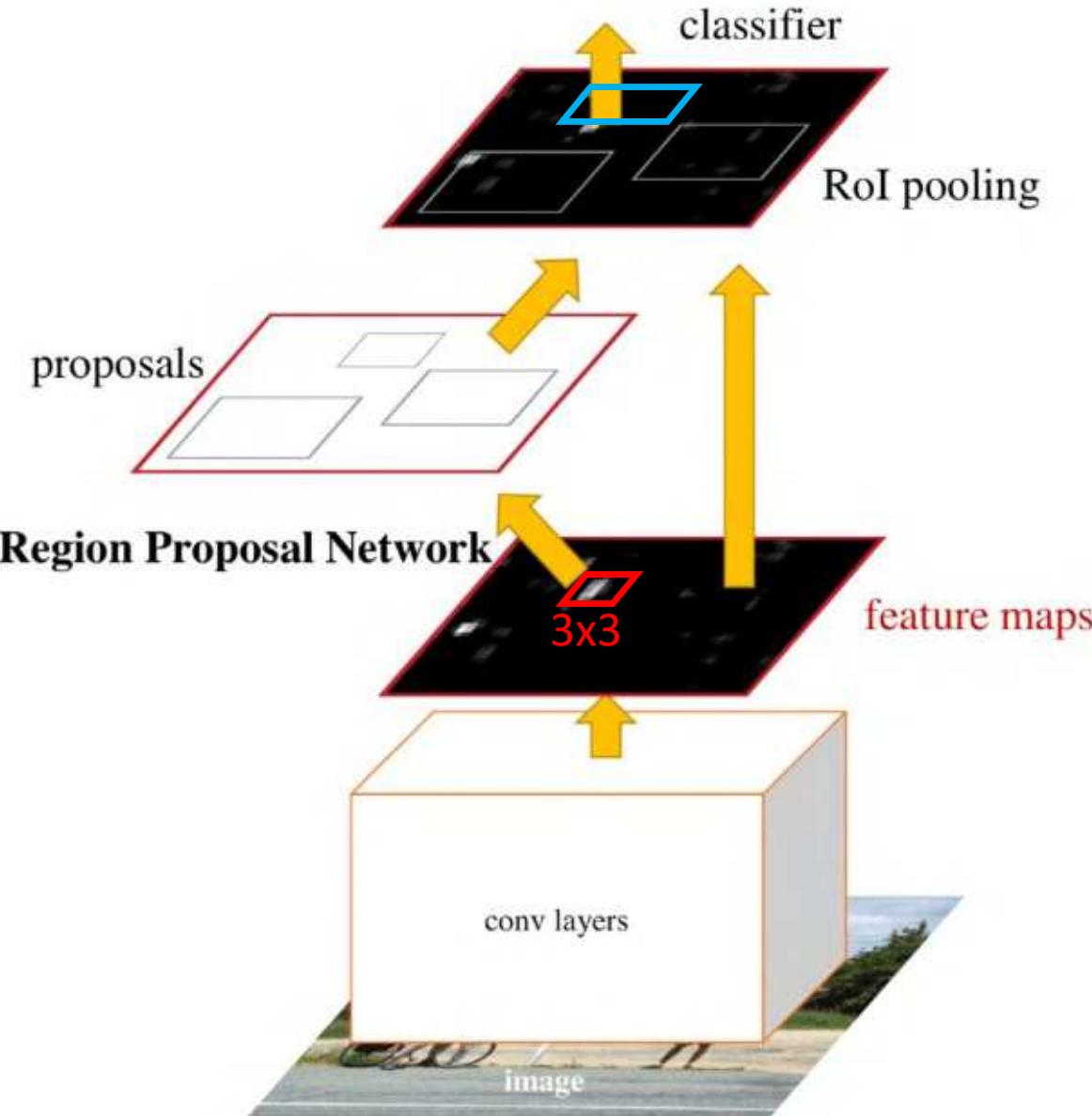
Region proposal network

- In this way, regions of interest are proposed.
 - At each location of the convolution feature map, k predictions are made.
 - Classification: whether this anchor is of interest. However, we only know that this region contain interesting objects, we do not know what they are.
 - Regression: how to transform this anchor to be closer to the object size.
 - In the next stage, we refine both the classification and localisation.



Detection network

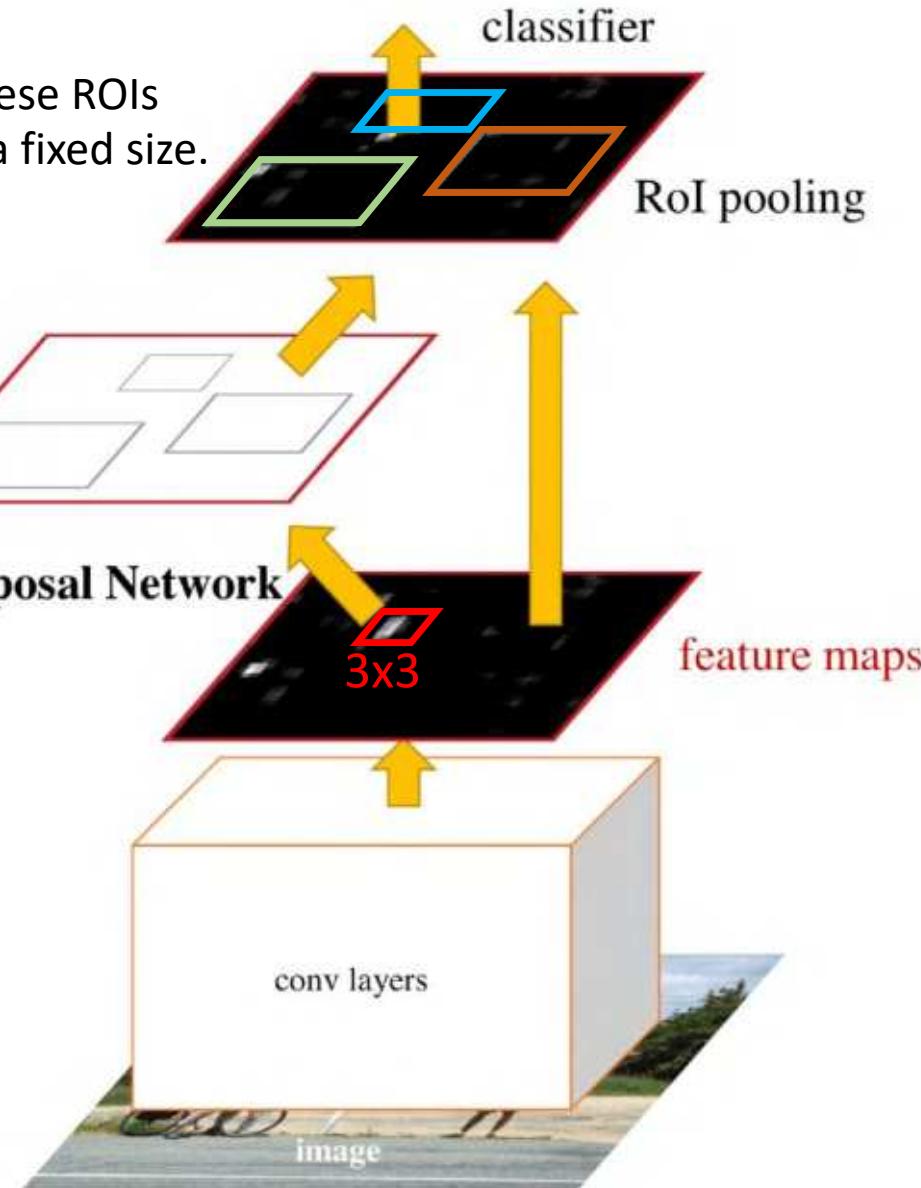
- Previously, we look at **3x3 windows** on the feature map, searching for objects.
- Since we know where they are and their size, why don't we look closer?
 - For each region, we use features from **this region of interest (ROI)**, instead of a fixed **3x3 window**.
 - We perform classification for this region.



ROI pooling

- Each ROI may have different sizes.
- The ROI pooling layer normalises the size, mapping the features within the ROI into a standard fixed size.
- The mapped features are provided to the classifier.

The features for these ROIs are normalised to a fixed size.



Multi-class classifier

- For each ROI, the classifier predicts its label class and refine the bounding box estimate.

$$L(p, t) = L_{cls}(p, y) + \lambda \cdot 1_{y \geq 1} L_{loc}(t, t^*)$$

|
ground truth,
multi-class

Classification loss Localisation loss

RPN vs Detection network

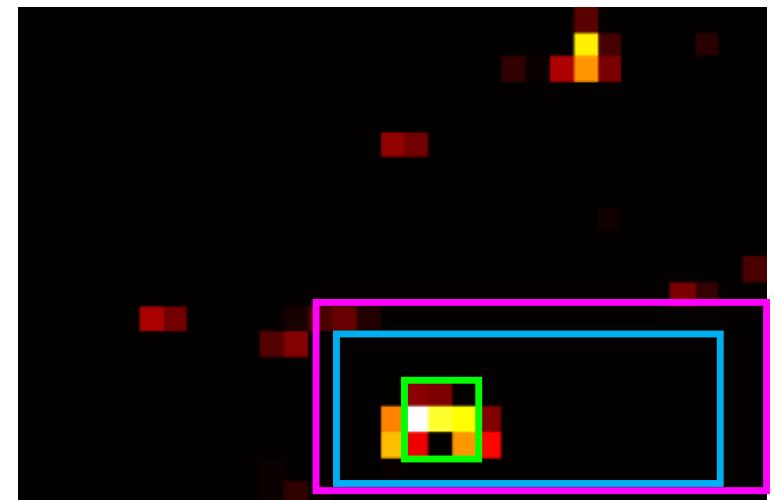
- Looks quite similar. What is the difference between RPN in Stage 1 and detection network in Stage 2?
 - The input to region proposal network is a 3×3 window on convolutional feature map.
 - RPN is class-agnostic. It only checks whether this is a ROI or not (binary).
 - RPN needs to use a lot of anchors, because at Stage 1 we do not yet know what the object looks like.
- The input to detection network is a proposed region, thus contains more accurate features.
- The detection network classifies the region into a number of classes.
- The detection network does not need to use anchors. We already know a rough size from the proposal network.

RPN vs Detection network

- RPN moves the 3×3 sliding window (**green**), classifies it to be something interesting (binary classification) and proposes a region (**blue**).
- The detection network takes the features inside the **blue** region, pools it into a fixed size window, classifies it to be a car (multi-class classification) and refines the bounding box (**purple**).



Input image



Feature map

Object detection

- Faster R-CNN is a **two-stage** object detection method.
 - Stage 1: region proposal.
 - Stage 2: classification and refined localisation.
- Stage 1 already performs localisation task.
- Maybe we can do everything in one go?
 - These are called **one-stage** object detection methods.
 - Examples: YOLO [1], SSD [2].

[1] J. Redmon et al. You only look once: Unified, real-time object detection. CVPR 2016.

[2] W. Liu et al. SSD: Single shot multibox detector. ECCV 2016.

Region proposal network

- In Faster-RCNN, the loss for RPN is defined as

$$L(p, t) = \sum_{i=1}^{n_{anchor}} \mathbf{L}_{cls}(p_i, p_i^*) + \lambda \cdot \sum_{i=1}^{n_{anchor}} 1_{y=1} L_{loc}(t_i, t_i^*)$$

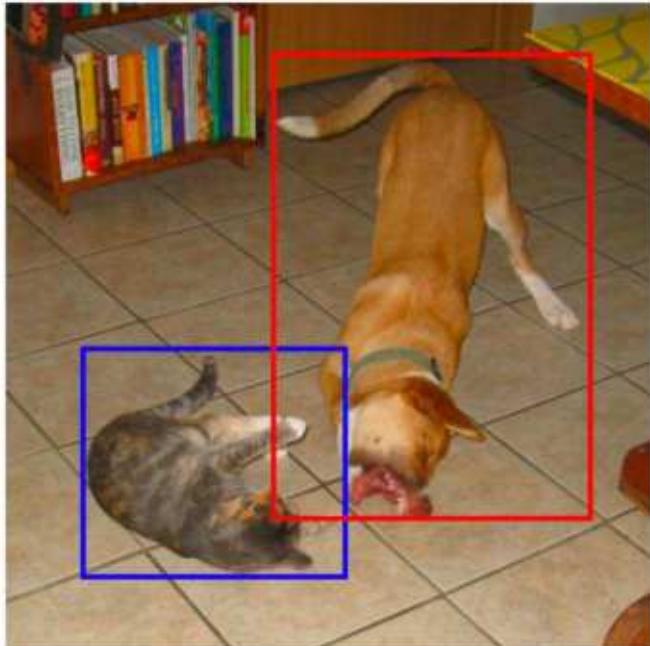
predicted score
for this anchor
 ground truth,
binary value

predicted
bounding box
 ground truth
bounding box

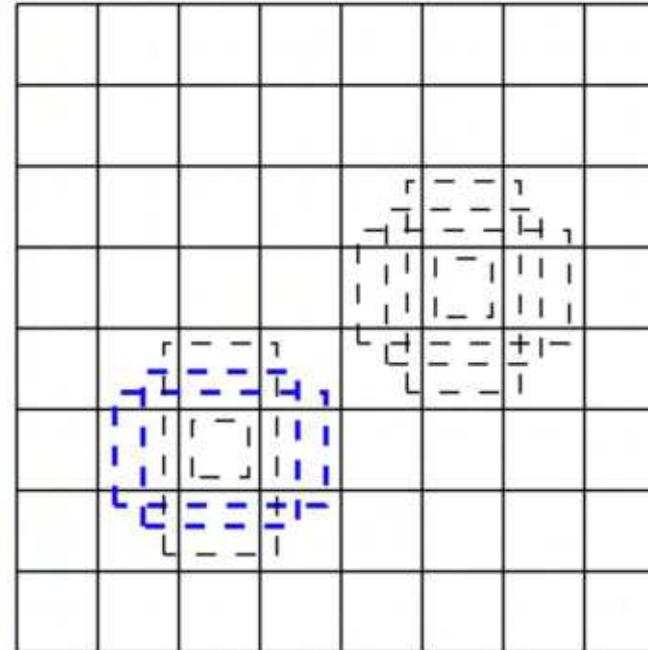
Classification loss
 Localisation loss

- Why don't we change this binary classifier into a multi-class classifier?
 - For each anchor, we directly predict what object it is.
 - SSD is one such example.

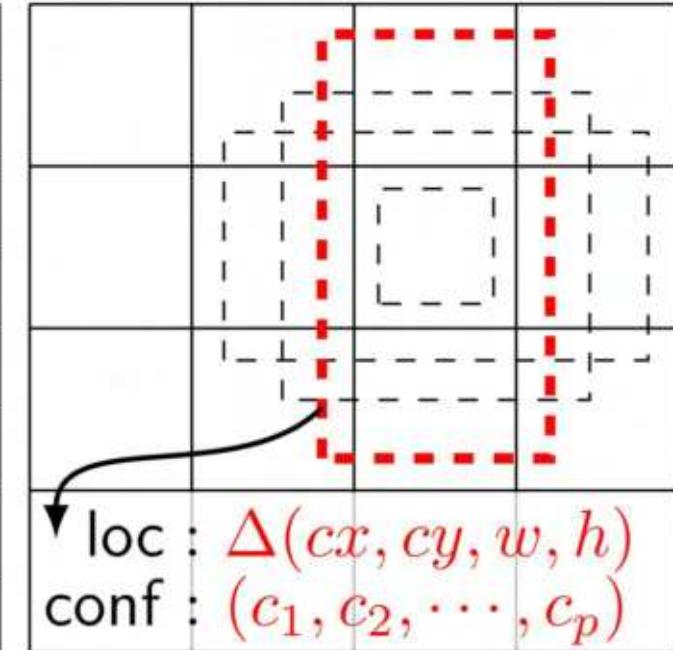
Single shot multibox detector (SSD)



(a) Image with GT boxes



(b) 8×8 feature map



(c) 4×4 feature map

(b) At each location on the feature map, SSD evaluates a set of boxes of different scales and aspect ratios (similar to anchors in RPN).

(c) For each box, SSD performs multi-class classification (different from RPN) and localisation.

One-stage vs Two-stage detection

- Which one performs better?
 - Faster R-CNN is more accurate but slower.
 - SSD is much faster but less accurate.
- Why is Faster R-CNN more accurate?
 - Stage 1: Ah, there is something there. Roughly estimate the region size.
 - Stage 2: Look closely at features in this region to classify and refine.

Summary

- Idea for object detection
 - Slide a window across the feature map.
 - Utilise convolutional features for classification and localisation.
- One-stage vs two-stage detection methods

References

- Sec. 14.1 Object detection. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Motion

Dr Wenjia Bai

Department of Computing & Brain Sciences

Motion

- Previously, we mainly look at understanding static images.
- In today's lecture, we will move onto video understanding.
 - How do we estimate motions in videos?
 - How do we track objects?
 - How do we understand videos and actions?



Object tracking [1].



Action recognition [2].

[1] <http://web.engr.oregonstate.edu/~lif/SegTrack2/dataset.html>

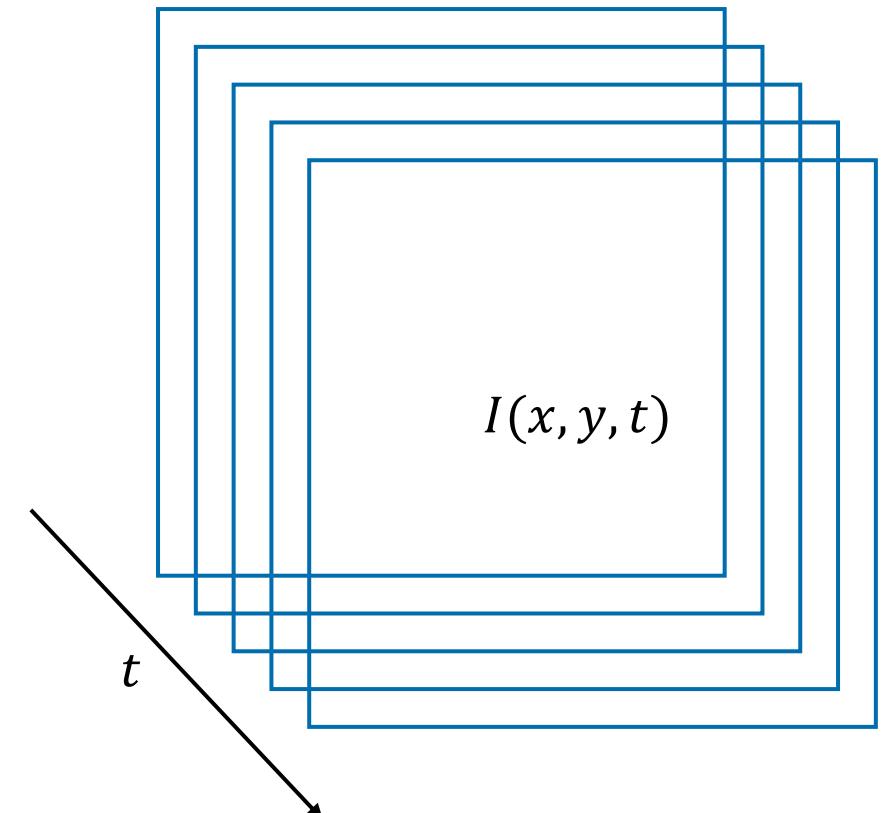
[2] J. Carreira and A. Zisserman. Quo Vadis, action recognition? A new model and the Kinetics dataset. CVPR 2017.

Optic flow method

- What is optic flow?
 - The motion (flow) of brightness patterns (optic) in videos.
 - The output of optic flow methods is a flow field, which describes the displacement vector for each pixel in the image.

Video

- A video is an 2D-t image sequence captured over time. It is a function of both space (x, y) and time t .
- For each point (x, y) at time t , we would like to estimate its corresponding position $(x + u, y + v)$ at time $t + 1$.



Assumptions in optic flow

- Brightness constancy A pixel has constant brightness across time.
- Small motion Between frames, motion is small.
- Spatial coherence Pixels move like their neighbours.



Time t



Time $t + 1$

Optic flow constraint

- Based on the **brightness constancy** assumption, we have

$$I(x + u, y + v, t + 1) = I(x, y, t)$$

- I : intensity
- (x, y, t) : spatial and temporal coordinates
- (u, v) : displacement
- Based on the **small motion** assumption, we perform first-order Taylor expansion for the left-hand term,

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t}$$

- Combining the two equations, we have

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0 \quad \text{Optical flow constraint equation}$$

Optic flow constraint

- If we use I_x, I_y, I_t to denote partial derivatives, it can be written as,

$$I_x u + I_y v + I_t = 0$$

The equation $I_x u + I_y v + I_t = 0$ is shown. Three lines point to the terms: a blue line to $I_x u$ labeled 'spatial gradient', a red line to $I_y v$ labeled 'displacement', and a green line to I_t labeled 'temporal gradient'.

- We have one equation with two unknowns (u, v) , so it is an underdetermined system. We could not solve the system.
- To address this problem, the Lucas-Kanade method introduces the **spatial coherence** assumption.
 - Flow is constant within a small neighbourhood.

Lucas-Kanade method

- At each pixel p , we have the optic flow constraint equation,

$$[I_x(p) \quad I_y(p)] \begin{bmatrix} u \\ v \end{bmatrix} = -I_t(p)$$

- For a small neighbourhood, e.g. a 3×3 window, we have a system of linear equations,

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_N) & I_y(p_N) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_N) \end{bmatrix}$$

- It is in the form of,

$$Ax = b$$

Lucas-Kanade method

- It becomes is an overdetermined system, with more equations than unknowns. The unknowns can be estimated using the least square method.

$$x = \underset{x}{\operatorname{argmin}} \|Ax - b\|^2$$

- The least square solution is given by,

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

Moore-Penrose inverse
or pseudo inverse

where

$$x = \begin{bmatrix} u \\ v \end{bmatrix}, A^T A = \sum_p \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, A^T b = - \sum_p \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix}$$

Lucas-Kanade method

- Have you seen this matrix before?

$$A^T A = \sum_p \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

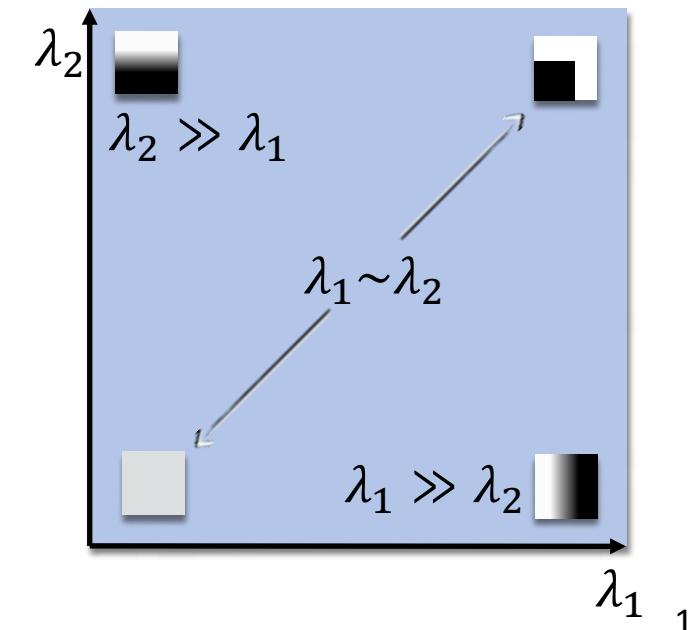
Lucas-Kanade method

- It also appears in the Harris detector.

$$M = \sum w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- In the Harris detector, we calculate eigenvalues λ_1, λ_2 of this matrix and derive a cornerness response.

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$



Lucas-Kanade method

- In the Lucas-Kanade method, we need to calculate the inverse of this matrix to derive the optic flow,

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

where

$$A^T A = \sum_p \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- How is Lucas-Kanade optic flow related to the cornerness response?

Lucas-Kanade method

- The flow is computed by

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

- In linear algebra, we have learnt that the condition number for a matrix is determined by its eigenvalues λ_{max} and λ_{min} .

$$cond(A^T A) = \frac{|\lambda_{max}|}{|\lambda_{min}|}$$

- For flat regions or edges, since λ_{min} is close to 0, we are likely to have a large condition number. Calculating the inverse $(A^T A)^{-1}$ becomes numerically sensitive to small perturbations.

For example,

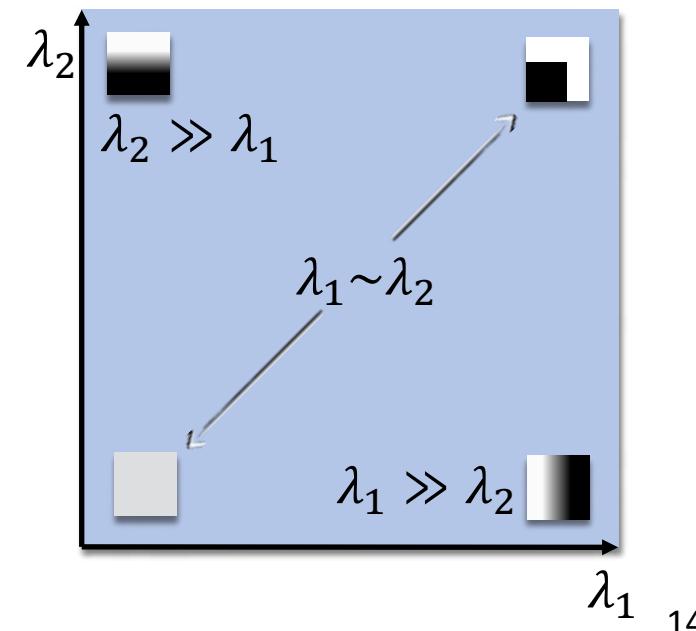
$$\begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix}$$

After adding some noise,

$$\begin{bmatrix} 1.01 & 0 \\ 0 & 0.02 \end{bmatrix}^{-1} = \begin{bmatrix} 0.99 & 0 \\ 0 & 50 \end{bmatrix}$$

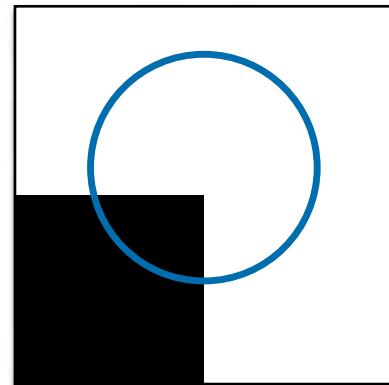
Lucas-Kanade method

- At corners, since λ_{min} are larger, we are likely to have a small condition number.
- This means that the matrix $A^T A$ is well-conditioned. Calculating its inverse $(A^T A)^{-1}$ is numerically more stable and thus flow estimation is more robust.

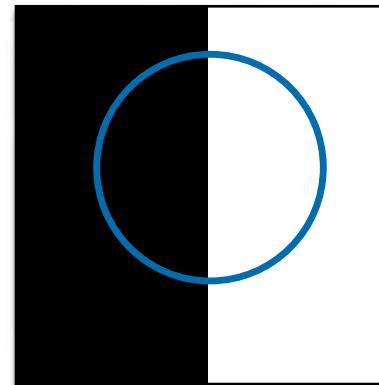


Lucas-Kanade method

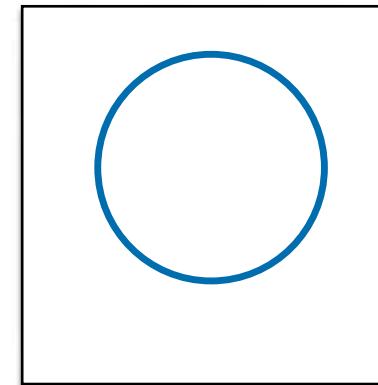
- Intuitively, if we observe motion from a small neighbourhood (the **blue** hole below), the motion at the corner is easier to track compared to an edge or a flat region.



Corner



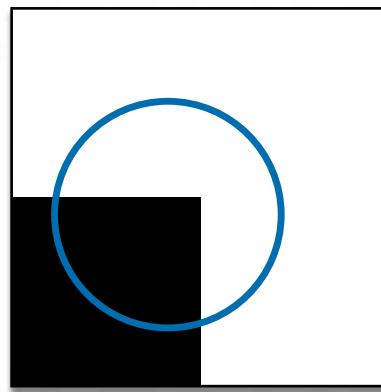
Edge



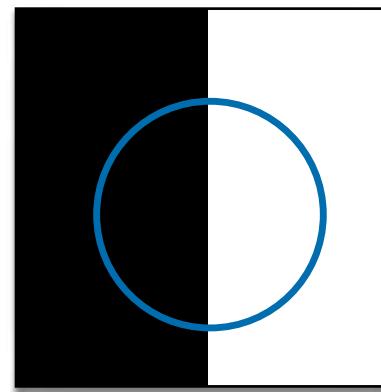
Flat region

Lucas-Kanade method

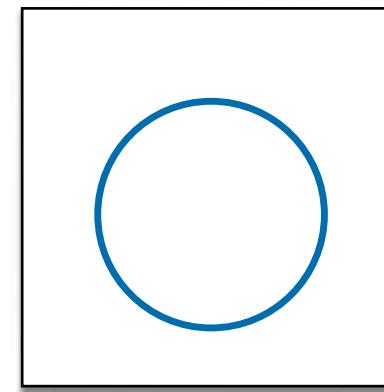
- Intuitively, if we observe motion from a small neighbourhood (the **blue** hole below), the motion at the corner is easier to track compared to an edge or a flat region.



Corner



Edge



Flat region

Aperture problem

- When we look through an aperture (hole), the motion of a line is ambiguous, because the motion component parallel to the line cannot be inferred based on the visual input.



Aperture problem

- When we look through an aperture (hole), the motion of a line is ambiguous, because the motion component parallel to the line cannot be inferred based on the visual input.



Aperture problem

- Motion of a corner is clearer to define, even through the aperture.



Aperture problem

- Motion of a corner is clearer to define, even through the aperture.



Lucas-Kanade method

- Compute the image gradients I_x, I_y (finite difference between neighbouring pixels) and I_t (finite difference between neighbouring time frames).
- For each pixel

- Calculate the following matrix for pixels in its neighbourhood,

$$A^T A = \sum_p \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, A^T b = - \sum_p \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix}$$

- Calculate the optic flow for this pixel,

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

Small motion assumption

- The optic flow constraint is derived based on the **small motion** assumption.
- We perform first-order Taylor expansion for $I(x + u, y + v, t + 1)$,

$$I(x + u, y + v, t + 1) = I(x, y, t)$$
$$I(x + u, y + v, t + 1) \approx I(x, y, t) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t}$$

in which we ignore the high-order terms.

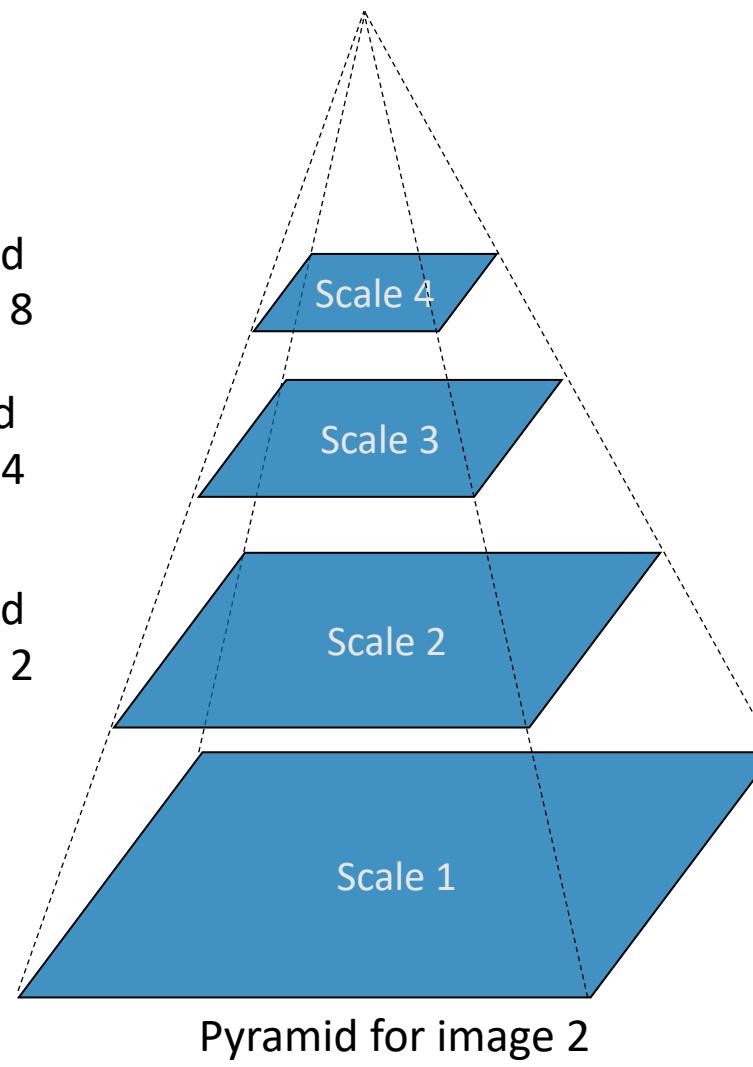
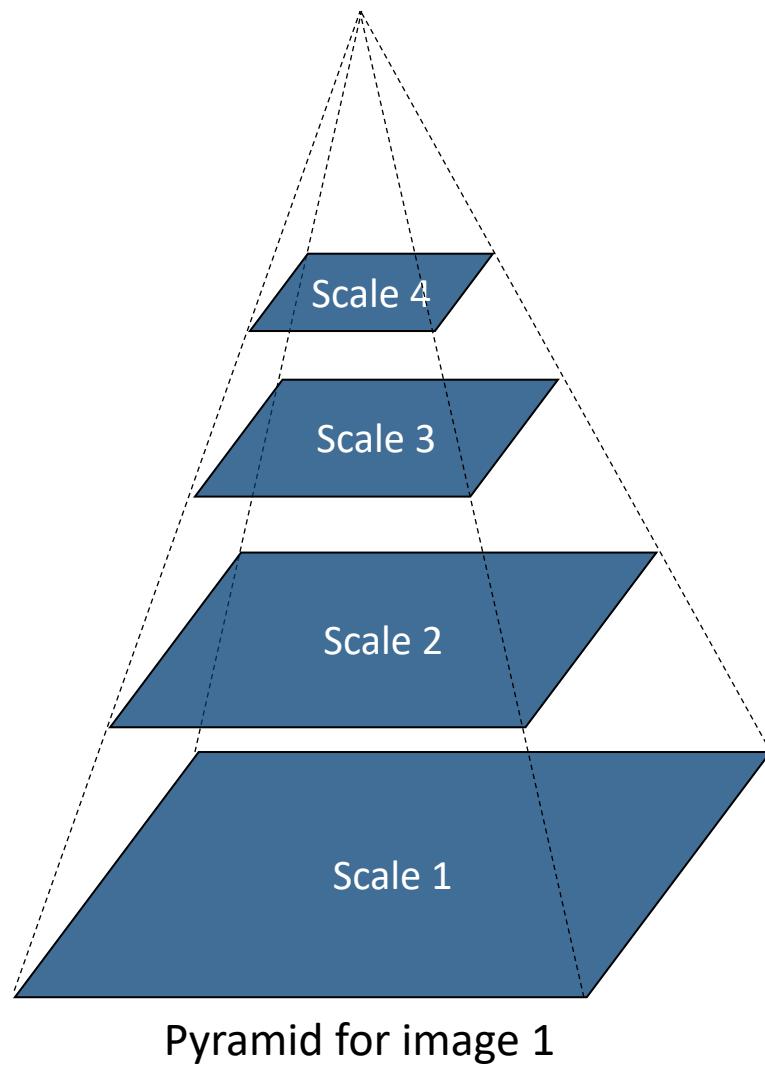
- \approx means “approximately equal to”, not “equal to”.

Large motion

- How do we estimate large motion?
 - For example, fast moving cars, footballs etc.
- We can introduce a multi-scale or multi-resolution framework.
 - Idea: Although the motion is large in the original resolution, it will look small in a downsampled resolution.

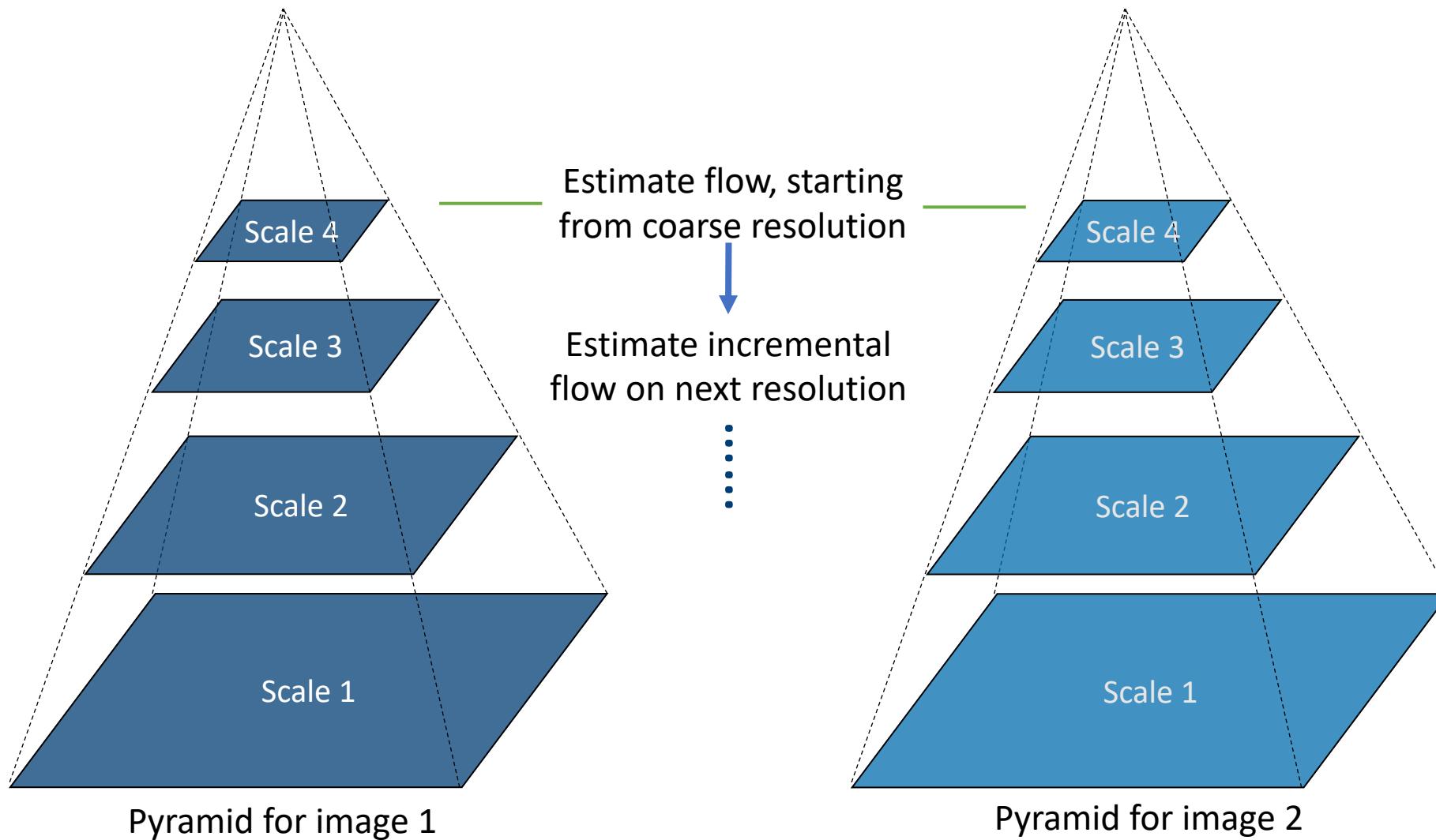
Image pyramid

A displacement of 8 pixels at Scale 1 is only 1 pixel at Scale 4 and becomes small motion.



An image pyramid is a multi-scale representation of an image.

Multi-scale framework



Coarse-to-fine motion estimation. The final flow field is obtained by summing up all incremental flows.

Multi-scale framework

- Suppose we obtain an estimate of the flow field $u^{(4)}, v^{(4)}$ for image I and J at scale 4.
- When we move to scale 3, the flow field becomes $2u^{(4)}, 2v^{(4)}$. This will provide the initial values of the calculation at scale 3.
- We can calculate the incremental flow for the following two images.
 - Image $I(x, y)$
 - Image $J_{warped} = J(x + 2u^{(4)}, y + 2v^{(4)})$
- Then we accumulate the flow estimation.

Multi-scale Lucas-Kanade method

- For scale l from coarse to fine

- Initial guess at scale l by upsampling the flow estimate at previous scale $l + 1$

$$u^{(l)} = 2u^{(l+1)}, v^{(l)} = 2v^{(l+1)}$$

- Compute the warped image

$$J_{warped}^l = J^l(x + u^{(l)}, y + v^{(l)})$$

- For image I^l and J_{warped}^l at this scale, compute the image gradients I_x, I_y and I_t .

- I_x, I_y are calculated from image I^l .

- $I_t = J_{warped}^l(x, y) - I^l(x, y)$.

- Estimate the incremental flow using $\begin{bmatrix} u^\delta \\ v^\delta \end{bmatrix} = (A^T A)^{-1} A^T b$.

- Update the flow at this scale: $u^{(l)} = 2u^{(l+1)} + u^\delta, v^{(l)} = 2v^{(l+1)} + v^\delta$.

Lucas-Kanade method

- We start from the optic flow constraint,

$$I_x u + I_y v + I_t = 0$$

- To solve the underdetermined system, we introduce the spatial coherence assumption, obtain an overdetermined system and solve it,

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_N) & I_y(p_N) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_N) \end{bmatrix}$$

- To address large motions, we introduce the multi-scale framework for motion estimation.

From optic flow to object tracking

- Optic flow methods estimate motion for each pixel in the image and provides a dense motion field.
- How do we perform object tracking, i.e. estimate motion for an object of interest?



Optic flow methods aim to estimate a dense (pixel-wise) motion field.



Object tracking aims to estimate the motion for one or multiple objects in a video.

Object tracking methods

- Lucas-Kanade tracker
- Correlation filter

Lucas-Kanade tracker

- The Lucas-Kanade method is a general framework for both optic flow estimation and object tracking.
- Basic assumptions
 - Constant brightness
 - Small motion

Lucas-Kanade tracker

- Lucas-Kanade aims to estimate the motion from the template image I to the image J in the next time frame.
- With the brightness constancy constraint, we formulate the following optimisation problem,

$$\min_{u,v} E(u, v) = \sum_x \sum_y [I(x, y) - J(x + u, y + v)]^2$$

x, y : pixels in the template image

u, v : motion from template I to image J .
Point (x, y) on I corresponds to Point $(x + u, y + v)$ on J .



Optimisation

- Cost function

$$\min_{u,v} E(u, v) = \sum_x \sum_y [I(x, y) - J(x + u, y + v)]^2$$

- Differentiate E with respect to u, v and let the derivatives be 0,

$$\frac{\partial E}{\partial u} = -2 \sum_x \sum_y [I(x, y) - J(x + u, y + v)] \frac{\partial J}{\partial x} = 0$$

$$\frac{\partial E}{\partial v} = -2 \sum_x \sum_y [I(x, y) - J(x + u, y + v)] \frac{\partial J}{\partial y} = 0$$

Optimisation

- With small motion assumption and Taylor expansion for J , we have

$$\frac{\partial E}{\partial u} = -2 \sum_x \sum_y \left[I(x, y) - J(x, y) - \frac{\partial J}{\partial x} u - \frac{\partial J}{\partial y} v \right] \frac{\partial J}{\partial x} = 0$$

$$\frac{\partial E}{\partial v} = -2 \sum_x \sum_y \left[I(x, y) - J(x, y) - \frac{\partial J}{\partial x} u - \frac{\partial J}{\partial y} v \right] \frac{\partial J}{\partial y} = 0$$

- With small motion assumption, we approximate $\frac{\partial J}{\partial x}$ by $\frac{\partial I}{\partial x}$, $\frac{\partial J}{\partial y}$ by $\frac{\partial I}{\partial y}$.
- We also have $\frac{\partial I}{\partial t} = J(x, y) - I(x, y)$.

Optimisation

- We can rewrite the equations as,

$$\frac{\partial E}{\partial u} = -2 \sum_x \sum_y \left[-\frac{\partial I}{\partial t} - \frac{\partial I}{\partial x} u - \frac{\partial I}{\partial y} v \right] \frac{\partial I}{\partial x} = 0$$

$$\frac{\partial E}{\partial v} = -2 \sum_x \sum_y \left[-\frac{\partial I}{\partial t} - \frac{\partial I}{\partial x} u - \frac{\partial I}{\partial y} v \right] \frac{\partial I}{\partial y} = 0$$

- Or simply as,

$$\frac{\partial E}{\partial u} = -2 \sum_x \sum_y [-I_t - I_x u - I_y v] I_x = 0$$

$$\frac{\partial E}{\partial v} = -2 \sum_x \sum_y [-I_t - I_x u - I_y v] I_y = 0$$

Optimisation

- Rewrite

$$\frac{\partial E}{\partial u} = -2 \sum_x \sum_y [-I_t - I_x u - I_y v] I_x = 0$$

$$\frac{\partial E}{\partial v} = -2 \sum_x \sum_y [-I_t - I_x u - I_y v] I_y = 0$$

in matrix form, we have

$$-\sum_x \sum_y \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix} - \sum_x \sum_y \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = 0$$

Optimisation

- Therefore the motion (u, v) can be solved,

$$\begin{bmatrix} u \\ v \end{bmatrix} = -\left(\sum_x \sum_y \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}\right)^{-1} \sum_x \sum_y \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix}$$

- We get the same solution as the Lucas-Kanade optic flow method, if you check the slides in the beginning of this lecture.
 - Lucas-Kanade optic flow calculates matrix by summing over a small neighbourhood.
 - Lucas-Kanade tracker calculates matrix by summing over pixels within the template image.



Object tracking

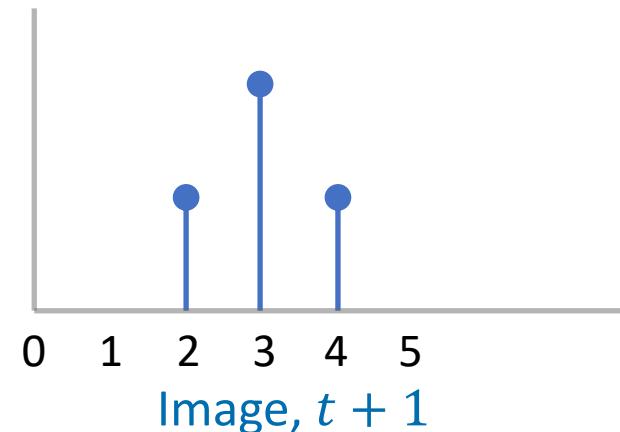
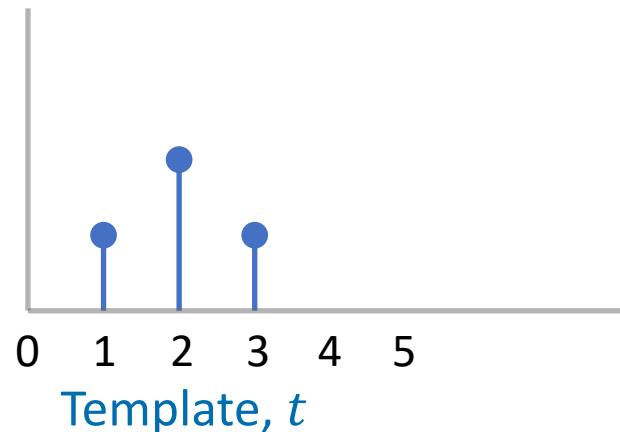
- The Lucas-Kanade method is a classical method.
 - The brightness constancy assumption may not always hold.
 - Lucas-Kanade only uses pixel intensities. It does not learn discriminative features for the template.
- Some recent object tracking methods utilise features extracted by convolutional networks.
 - Correlation filter method

Correlation filter

- We aim to maximise the correlation between template features and image features in the next time frame.

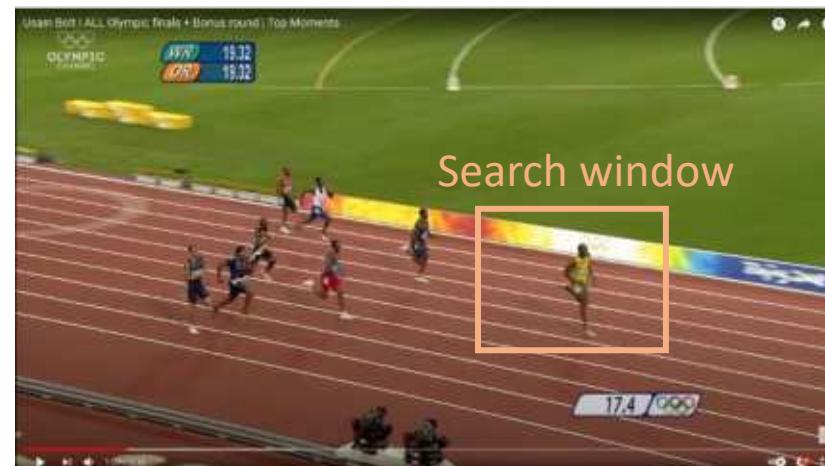
$$(f \star h)[n] = \sum_{m=-\infty}^{\infty} f[m]h[n+m]$$

- Correlation can be more robust to illumination changes than sum of squared difference.



Correlation filter

- For 2D images, we can find where the correlation achieves the maximum between the template features and features in a search window.
- The features can be learnt from CNNs.



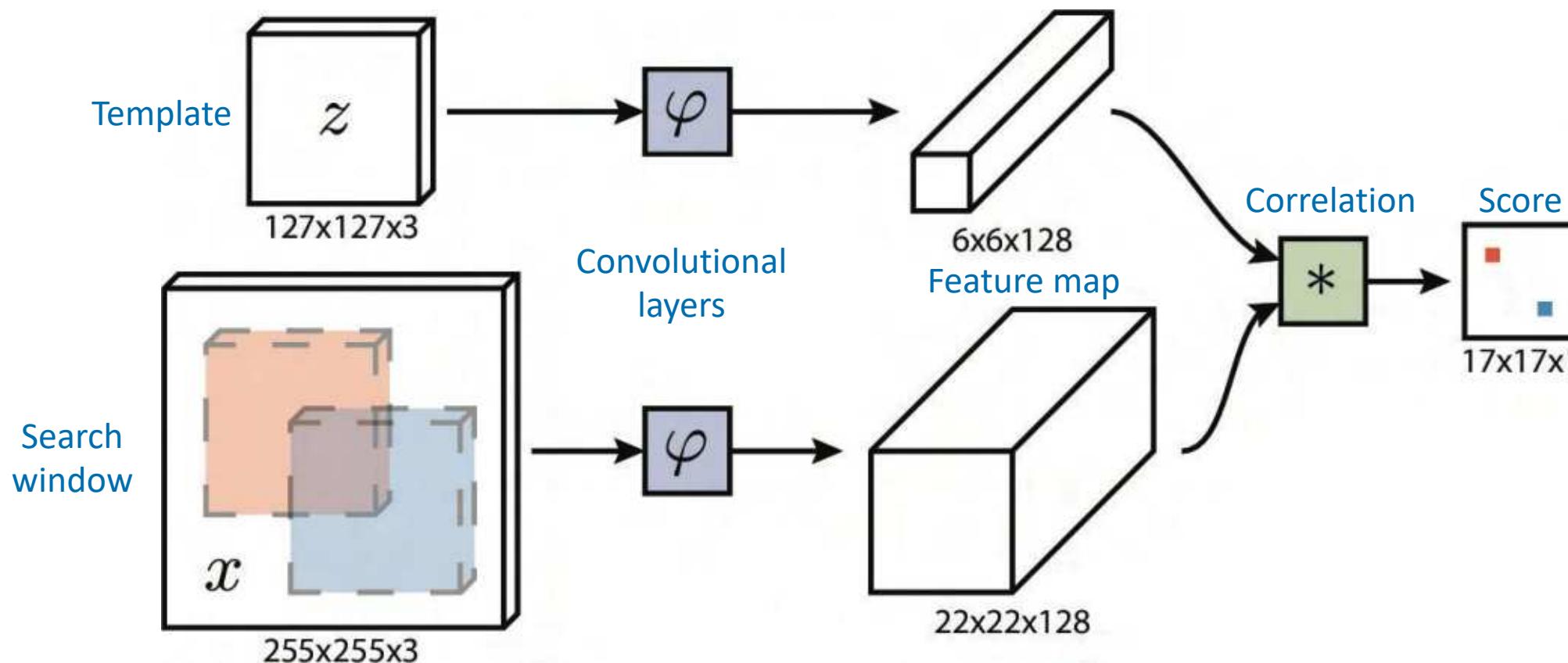
time t

time $t + 1$

time $t + 2$

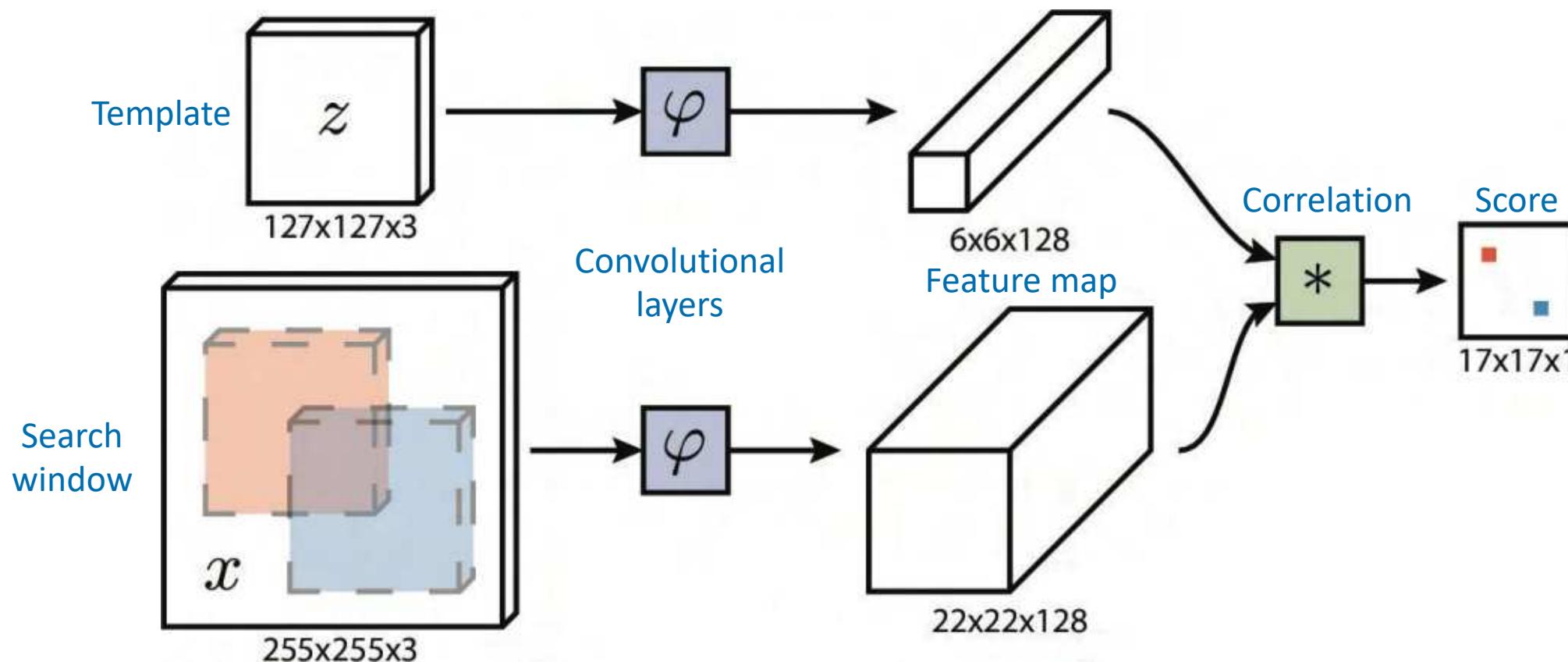
Correlation filter

- A recent approach uses a Siamese network to learn and compare features.



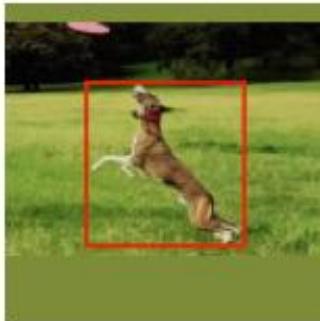
Correlation filter

- The network parameters are trained to predict a ground truth score map.

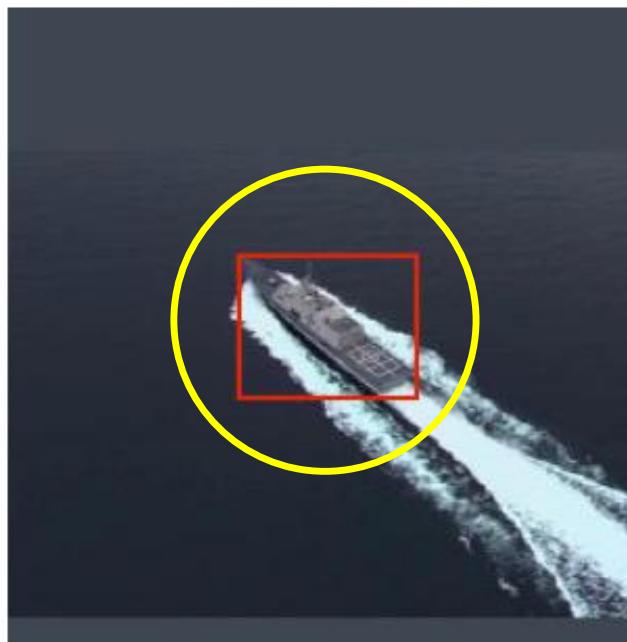


Defining the ground truth score map

Template



Search window



The ground truth score is +1 within the yellow circle and -1 outside.



Example tracking results.

Action recognition

- Apart from object tracking, action recognition is another interesting application that involves videos and motion.



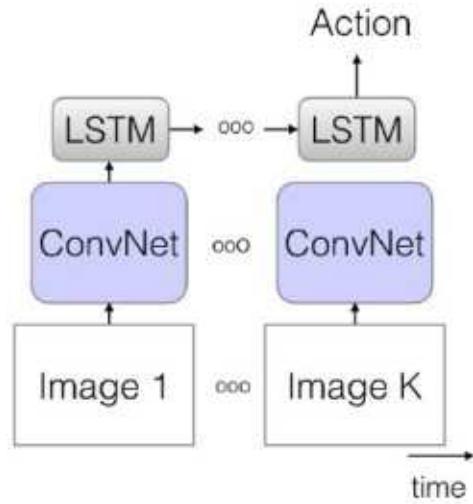
It is challenging to recognise actions from still images.



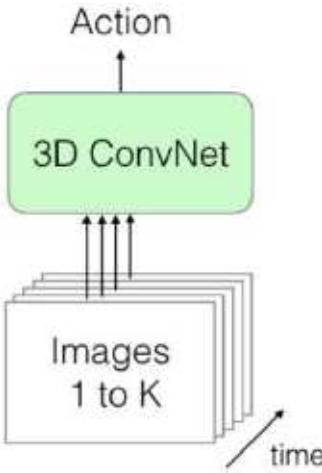
How do we recognise actions?

- We use both image and motion information.
 - Image: static features extracted from a single image
 - Motion: dynamic features extracted from videos, e.g. optic flow fields
- We can combine both to train a classifier for action recognition.

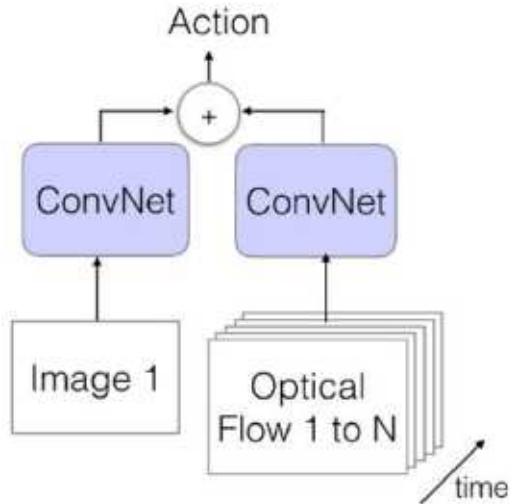
a) LSTM



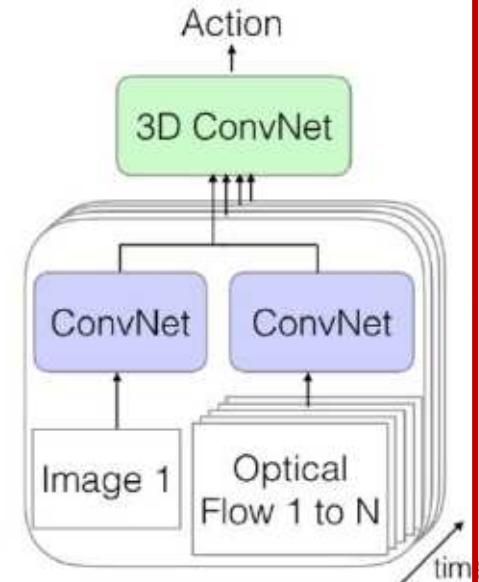
b) 3D-ConvNet



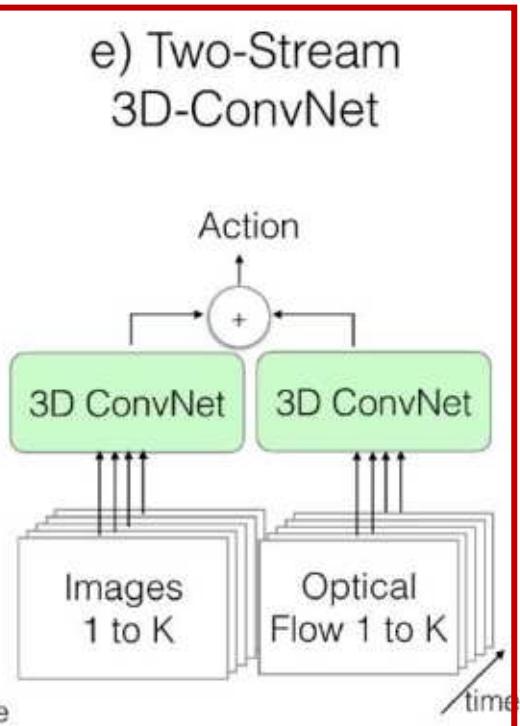
c) Two-Stream



d) 3D-Fused Two-Stream



e) Two-Stream 3D-ConvNet



Network architectures for action recognition, which use both image and motion information.

Architecture	UCF-101			HMDB-51			miniKinetics		
	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow	RGB	Flow	RGB + Flow
(a) LSTM	81.0	–	–	36.0	–	–	69.9	–	–
(b) 3D-ConvNet	51.6	–	–	24.3	–	–	60.0	–	–
(c) Two-Stream	83.6	85.6	91.2	43.2	56.3	58.3	70.1	58.4	72.9
(d) 3D-Fused	83.2	85.8	89.3	49.2	55.5	56.8	71.4	61.0	74.0
(e) Two-Stream I3D	84.5	90.6	93.4	49.8	61.9	66.4	74.1	69.6	78.7

The action recognition accuracy is improved by using both image and motion information, compared to using image information alone.

Summary

- Lucas-Kanade optical flow method
- Lucas-Kanade object tracking method
- Correlation filter method
- Action recognition

References

- Sec. 8.4 Optical flow. Richard Szeliski, Computer Vision: Algorithms and Applications (<http://szeliski.org/Book>).

Camera Model

Dr Wenjia Bai

Department of Computing & Brain Sciences

Object tracking in 3D world

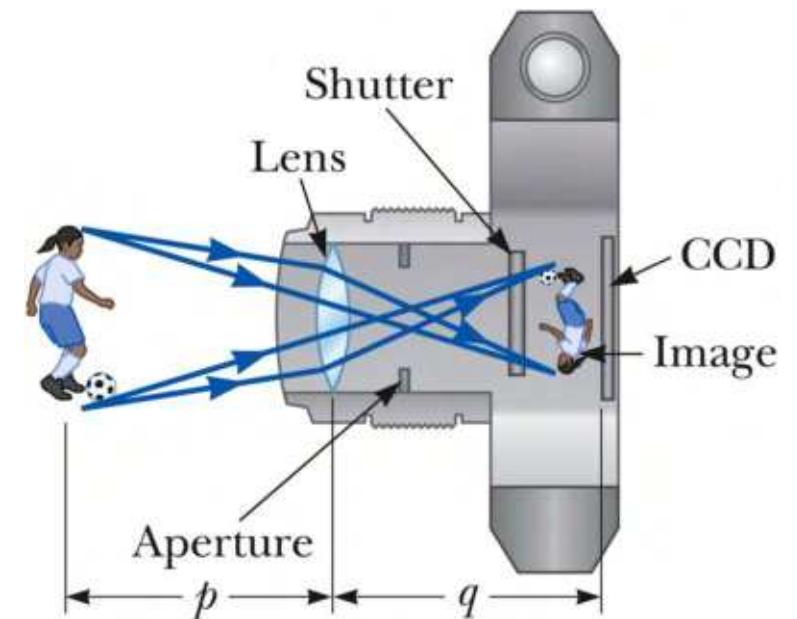


2D to 3D

- The motion that we estimate from videos is in 2D.
- For 3D applications, if we need to estimate the 3D motion of an object, we need to understand
 - how camera works
 - how camera maps 3D coordinates to 2D coordinates.

In this lecture

- Previously, we mainly talk about 2D images (or 2D-t images, videos).
- Today, we will discuss how 2D information is related to the 3D world that we live in.
- The mapping from 3D to 2D is described by the camera model.



Camera model

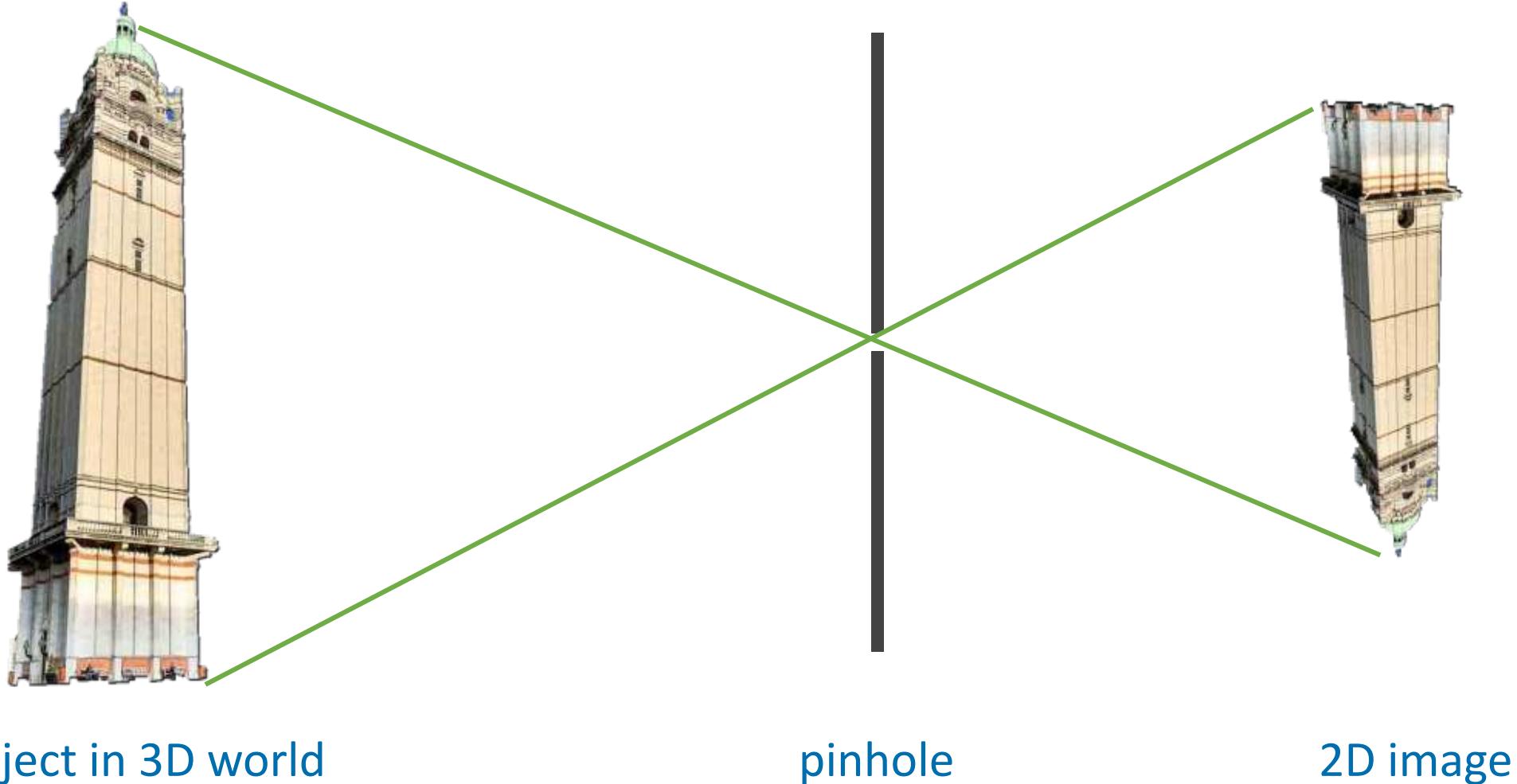
- The camera model describes the mapping from the 3D world to a 2D image.

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

2D camera 3D
coordinate matrix coordinate

- We will start from the pinhole camera.
- The concepts developed here can be generalised to other cameras.

Pinhole camera



Pinhole camera

- Early descriptions dated back to Mozi and Aristotle's writings around 4th and 3rd century BC.
- Studied extensively by Ibn al-Haytham in the early 11th century.



Mozi



Aristotle



Ibn al-Haytham

Pinhole camera

- It is also known as camera obscura.
- Leonardo da Vinci made one and compared it to the working of the eye.



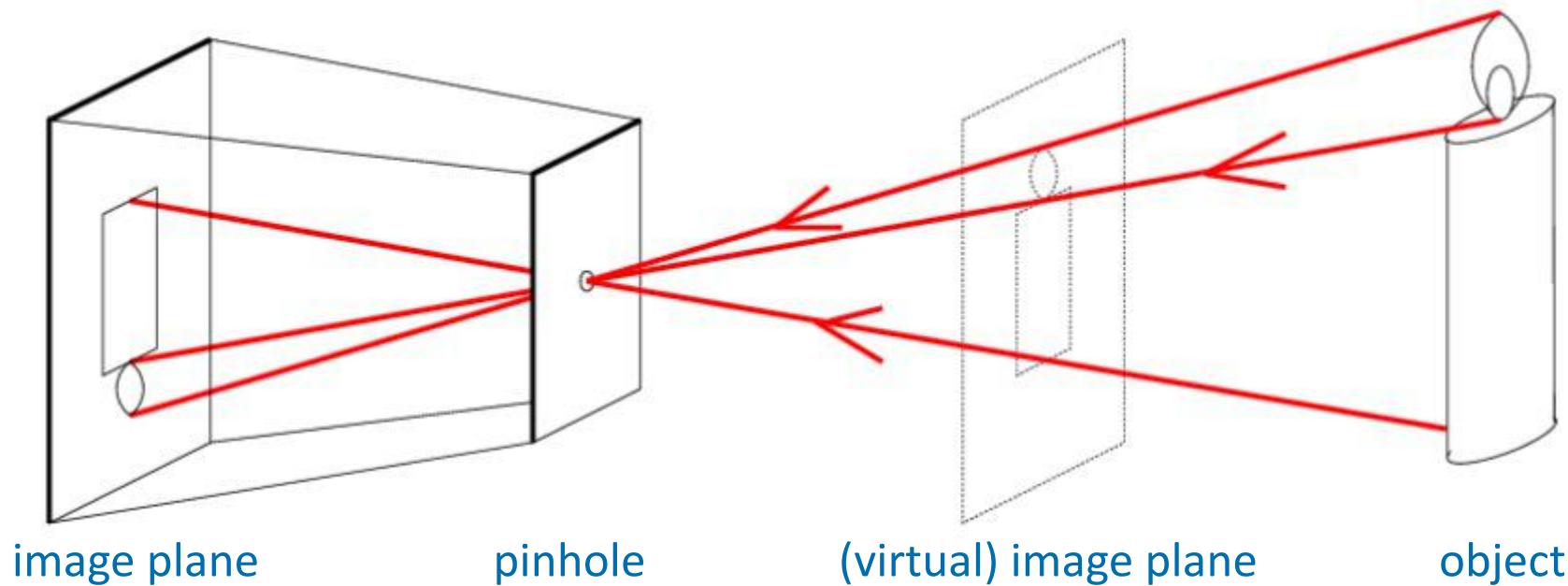
Leonardo da Vinci
(1452/1519)



Esboço do princípio físico da camera obscura
Codex Atlanticus (1515)

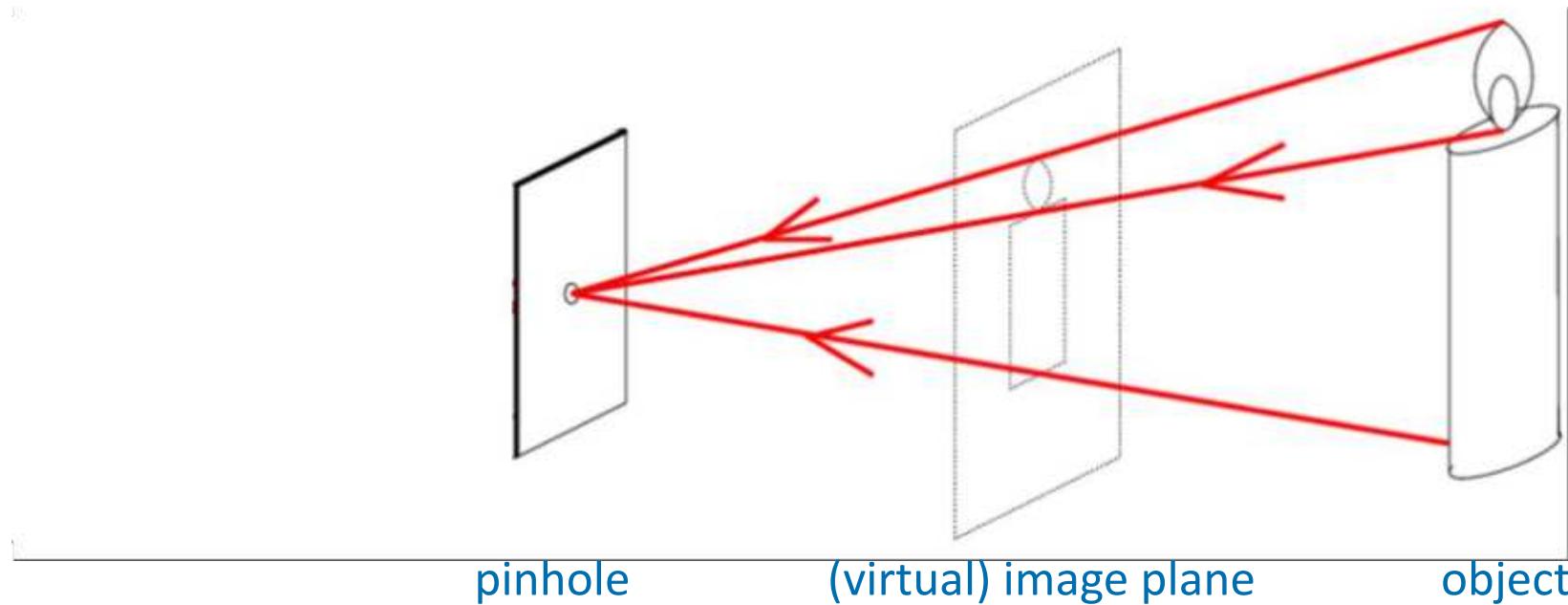
Pinhole camera

- Let us model how a 3D coordinate $X = (X, Y, Z)$ is mapped to $x = (x, y)$ on the image plane.
- For convenience, we rotate the image plane by 180 degrees and put it as a virtual image plane in front of the pinhole.

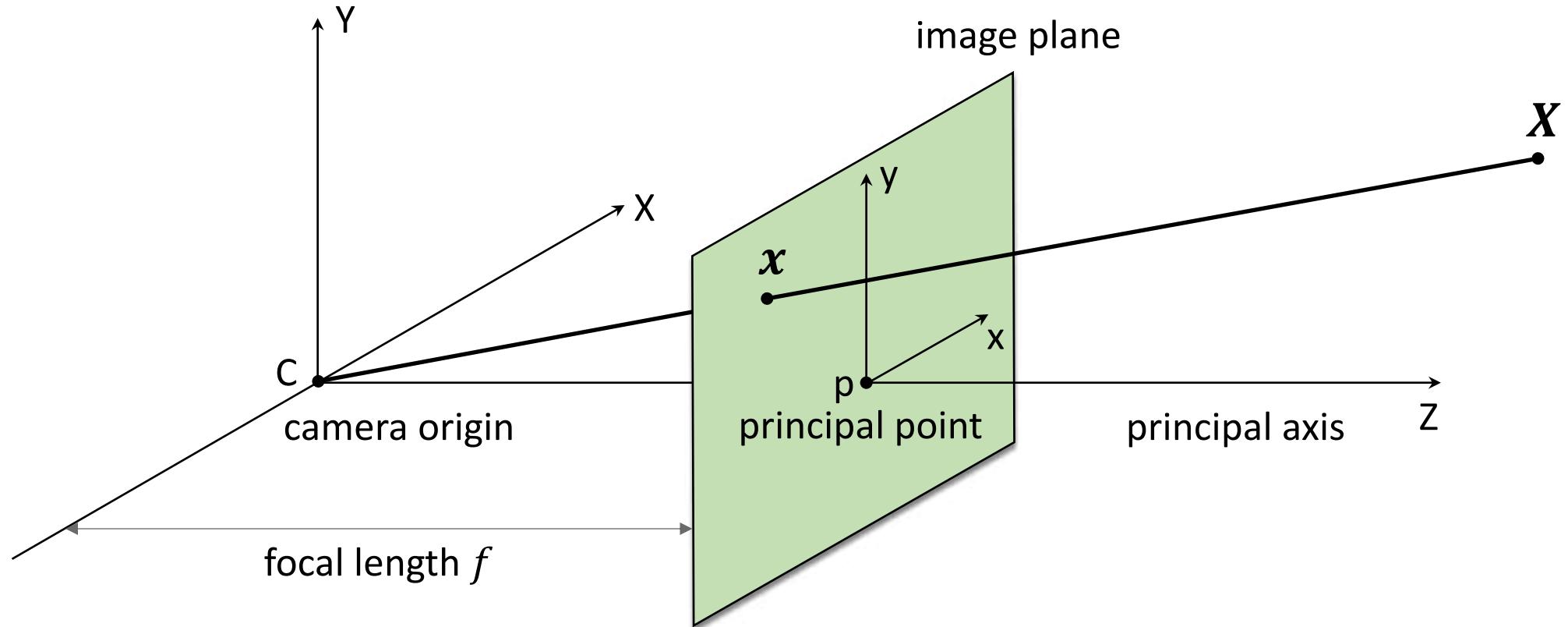


Pinhole camera

- Let us model how a 3D coordinate $X = (X, Y, Z)$ is mapped to $x = (x, y)$ on the image plane.
- For convenience, we rotate the image plane by 180 degrees and put it as a virtual image plane in front of the pinhole.



Pinhole camera



How is a 3D coordinate $\mathbf{X} = (X, Y, Z)$ is mapped to $\mathbf{x} = (x, y)$ on the 2D image plane?

Pinhole camera

- By similar triangles, we can see that

$$(X, Y, Z) \rightarrow (fX/Z, fY/Z)$$

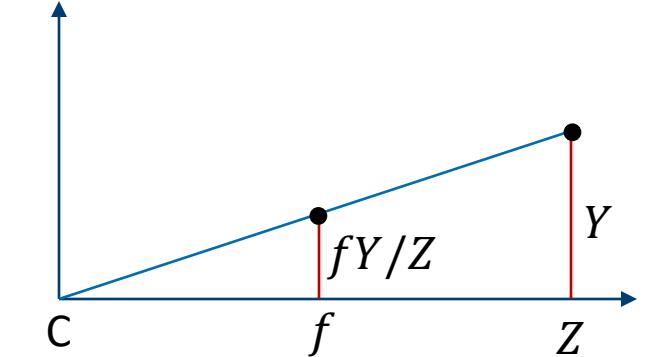
map to

- If we use the homogeneous coordinate, this becomes

$$(X, Y, Z, 1) \rightarrow (\frac{fX}{Z}, \frac{fY}{Z}, 1) \text{ or } (fX, fY, Z)$$

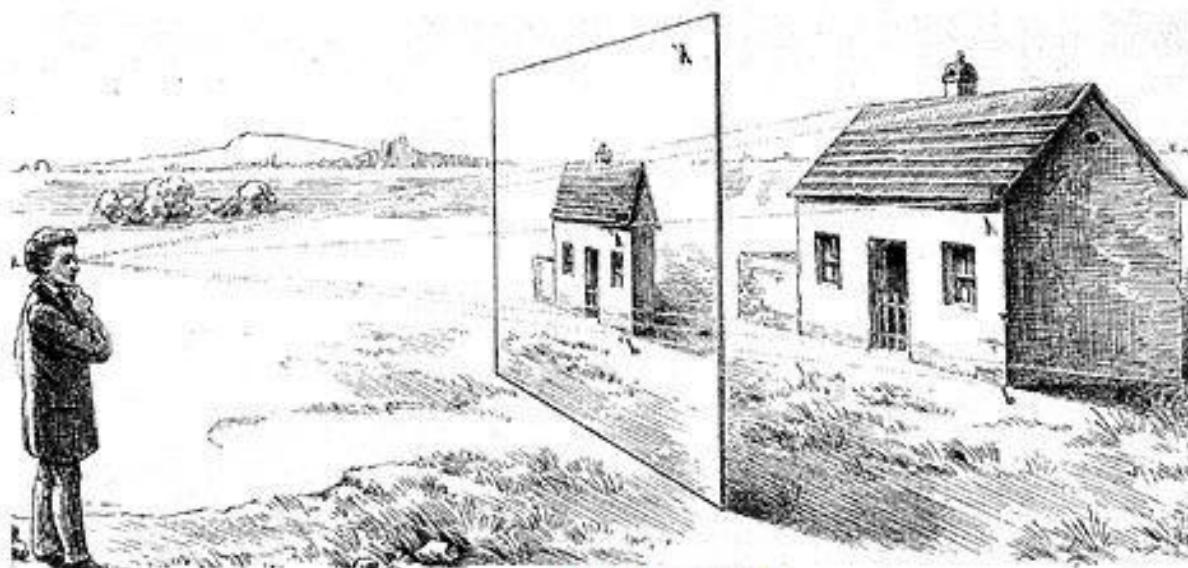
- What is a homogeneous coordinate?

- In a 2D space, $(x, y, 1)$ represent the same point as (kx, ky, k) for any non-zero k .
- Similarly in a 3D space, $(X, Y, Z, 1)$ and (kX, kY, kZ, k) are the same point.
- The homogeneous representation provides convenience in geometry.



Perspective projection

- The mapping for the pinhole camera is a perspective projection.
- The object appear smaller as the distance increases.



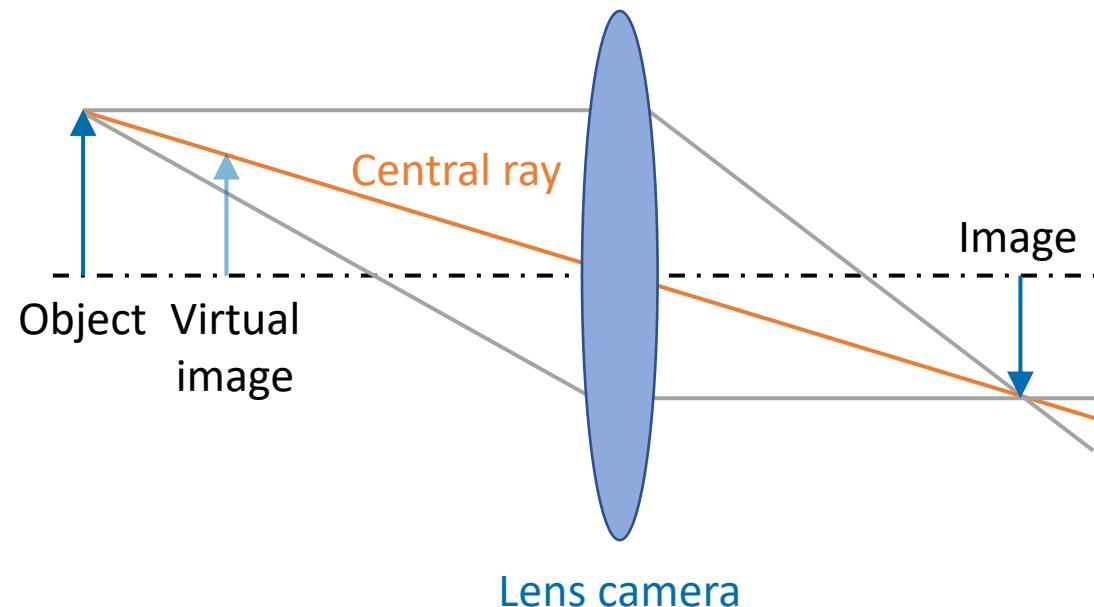
Perspective in painting



Raphael, The School of Athens, 1509-1511
Notable for accurate perspective projection

Lens camera

- For lens camera, when we look along the central ray, we can also find the similar triangles.
- The further the object, the smaller the image. It has a similar camera matrix as the pinhole camera.



Pinhole camera

- Using the matrix notation, we have

$$\begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

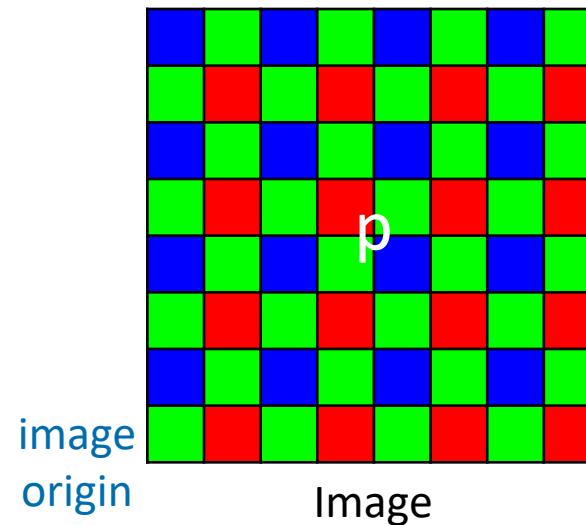
image
coordinate

camera
coordinate

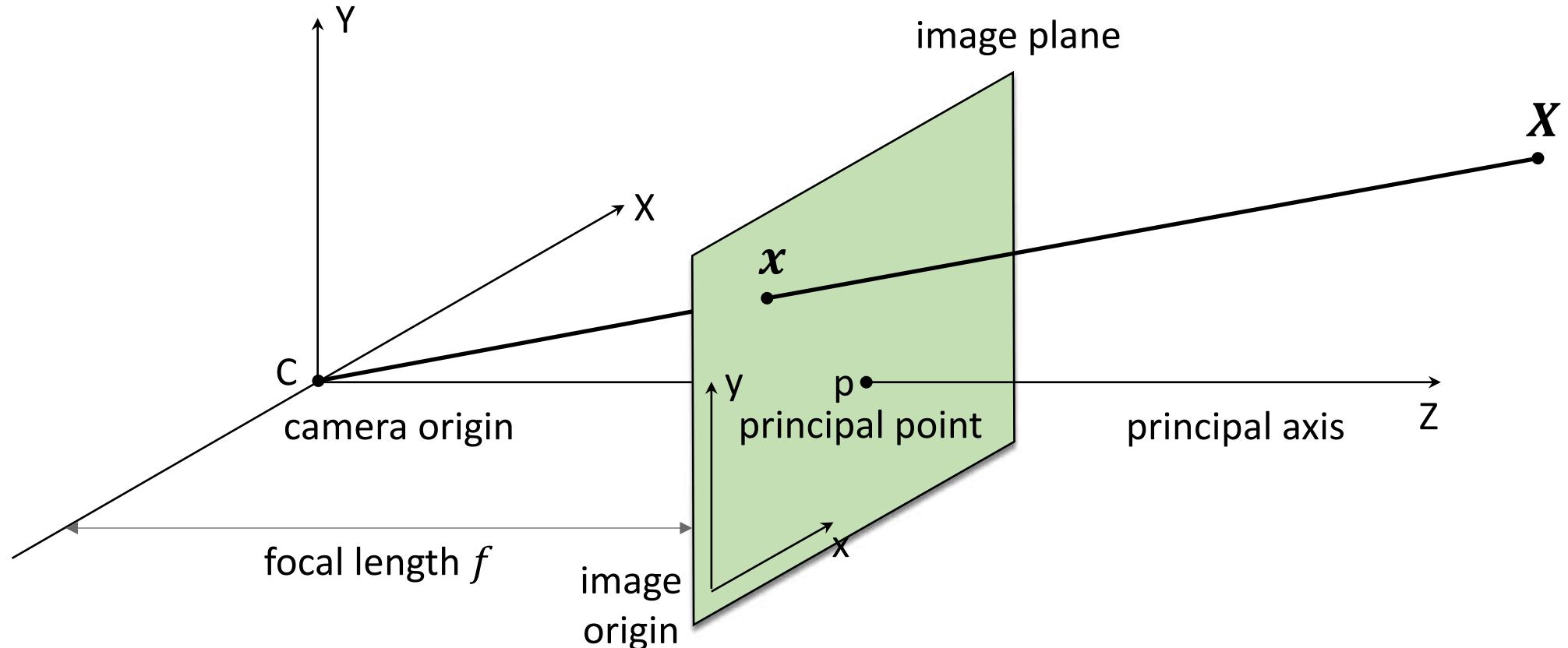
- The matrix $P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ describes the mapping from **3D camera coordinates** to **2D image coordinates**.

Principal point offset

- The image coordinate system may have a different origin from the principal point p .
 - For example, image origin may be defined at the bottom left of the CCD array.



Pinhole camera



The image origin may be different from the principal point p .
We need to account for this shift.

Pinhole camera

- Denote the coordinate of the principal point as $p = (p_x, p_y)$ on the image.
- To account for this shift, $(X, Y, Z, 1)$ is mapped to $\left(\frac{fX}{Z} + p_x, \frac{fY}{Z} + p_y, 1\right)$.
- The mapping from **camera coordinates** to **image coordinates** is formulated as,

$$\begin{bmatrix} fX + p_x Z \\ fY + p_y Z \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Pinhole camera

- The mapping is

$$\begin{bmatrix} fX + p_x Z \\ fY + p_y Z \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Denote

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

we can write a concise form

$$\mathbf{x} = [K \mid 0] \mathbf{X}_{cam}$$

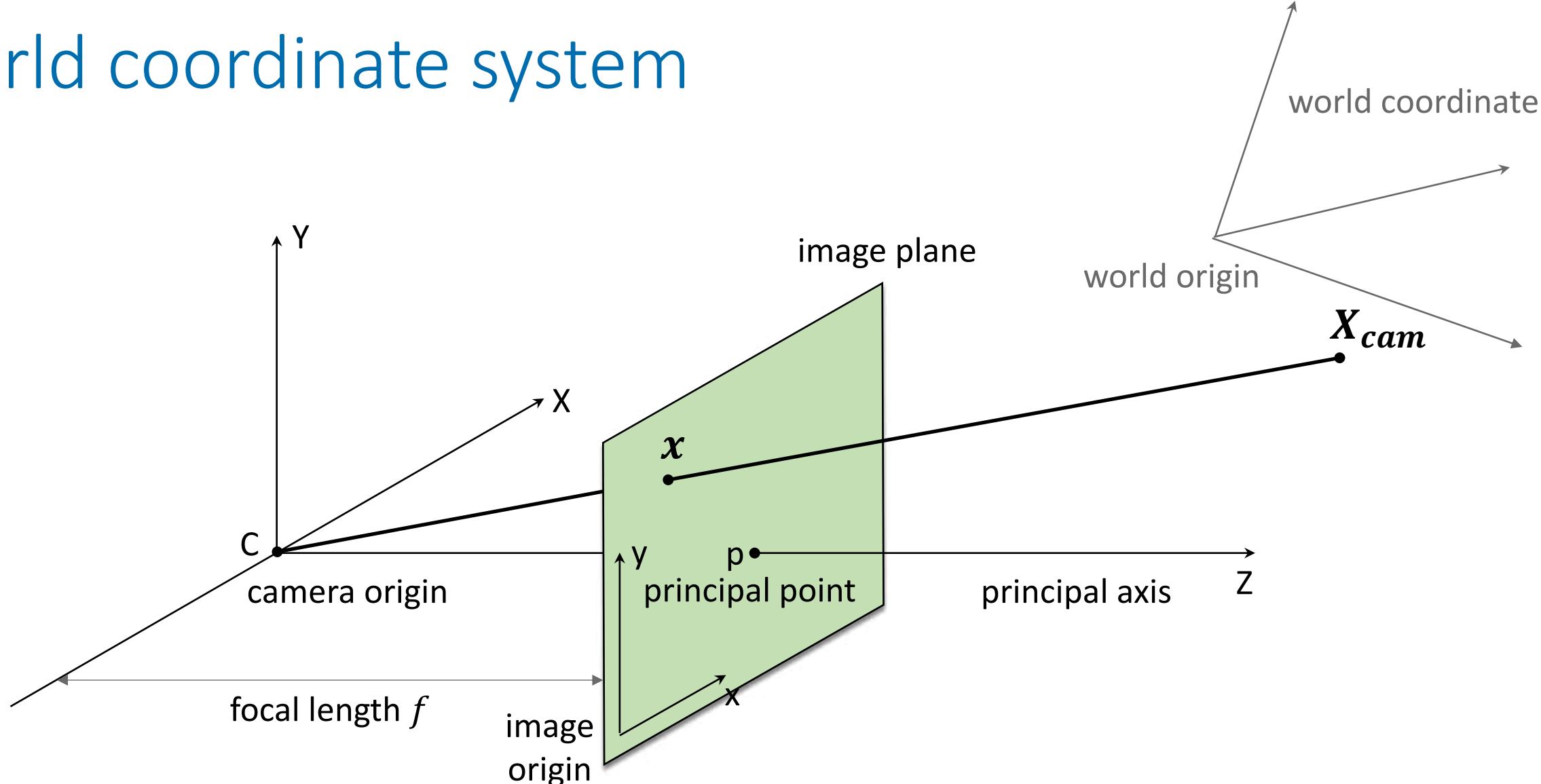
or

$$\mathbf{x} = K[I \mid 0] \mathbf{X}_{cam}$$

image coordinate

camera coordinate

World coordinate system



The camera coordinate system is related to the world coordinate system by 3D rotation and translation.

World-to-camera transformation

- If the world coordinate system differs from the camera coordinate system only by translation, we have

$$X_{cam} = X - C$$

translation

- If there is also rotation involved, we have

$$X_{cam} = R(X - C)$$

3D rotation
matrix

- X_{cam} and X here are inhomogeneous coordinates, i.e. $X = [X, Y, Z]^T$.

World-to-camera transformation

- If we use homogeneous coordinates, i.e. $\mathbf{X} = [X, Y, Z, 1]^T$, the transformation is written as

$$\begin{bmatrix} X_{cam} \\ Y_{cam} \\ Z_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

or

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \mathbf{X}$$

Pinhole camera matrix

- We first map from **world coordinate** to **camera coordinate**, then to **image coordinate**,

$$\mathbf{x} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \cdot \mathbf{X}$$

image coordinate world coordinate

or

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

- The camera matrix is

$$\mathbf{P} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix}$$

Pinhole camera matrix

- The camera matrix is

$$P = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix}$$

or

$$P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & -RC \end{bmatrix}$$

- It can be concisely written as

$$\begin{aligned} P &= K \cdot [R \mid -RC] \\ &= KR[I \mid -C] \end{aligned}$$

Pinhole camera matrix

- The camera matrix is

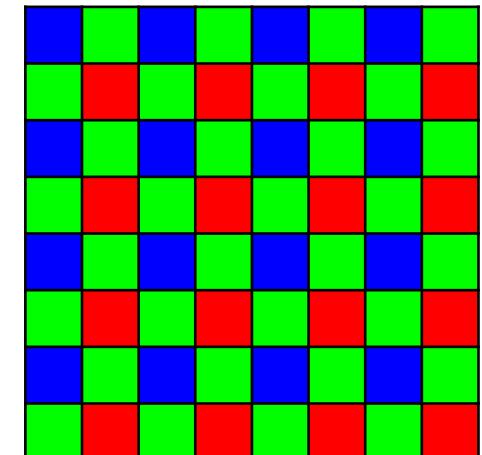
$$P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \cdot [R \quad -RC]$$

intrinsic parameters: extrinsic parameters:
camera internal properties world-to-camera transformation

- It has 9 degrees-of-freedom (DoF).
 - The intrinsic parameters have 3 DoF (f, p_x, p_y).
 - The extrinsic parameters have 6 DoF (3 for 3D rotation R , 3 for 3D translation C).

Pixel on camera sensors

- Till now, we represent the image coordinate in the same unit as camera coordinate, e.g. metre or millimetre.
- On camera sensors, the distance is converted to pixels, depending on the pixel spacing.
 - $\left(\frac{fX}{Z} + p_x, \frac{fY}{Z} + p_y, 1\right)$ is converted to $\left(k_x\left(\frac{fX}{Z} + p_x\right), k_y\left(\frac{fY}{Z} + p_y\right), 1\right)$.
 k_x and k_y are the ratios for conversion (e.g. unit: pixel / millimetre).
 - Or in the form of $(\alpha_x X + x_0, \alpha_y Y + y_0, 1)$.



Pixel on camera sensors

- The camera matrix becomes

$$P = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot [R \quad -RC]$$

- It is possible that the pixels are anisotropic on sensors (α_x, α_y may be unequal).
- In this way, the camera matrix has 10 degrees-of-freedom.

Skew

- With more generality, we can add skew s ,

$$P = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot [R \quad -RC]$$

- Skew means that the x-axis and y-axis are not orthogonal. Skew $s = 0$ for most normal cameras.
- In this way, the camera matrix has 11 degrees-of-freedom.

General projective camera

- We have the following camera matrix.

$$P = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot [R \quad -RC]$$

intrinsic parameters extrinsic parameters

- P is a 3×4 matrix with 12 elements.
- P has 11 DoF (5 from intrinsic parameters, 6 from extrinsic parameters).

Camera calibration

- How do we estimate the camera matrix P ?
- This is known as camera calibration or pose estimation.
- We assume the 3D structure (the world coordinates X of some structures) is already known. Given an input 2D image, we estimate the camera matrix (or the pose of the photographer).

Camera calibration

- Given a set of points $\{X_i, x_i\}$ and the camera model

$$x = P X$$

our aim is to estimate the matrix P .

- This sounds a familiar problem. In image matching, machine learning, motion estimation, we always have such problems. Common solutions include:
 - Analyse the unknowns and equations. Write down an equation system and solve it.
 - Formulate it as an optimisation problem and solve it.

Mapping from world to image

- The mapping is formulated as,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

or

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} \cdot \mathbf{X}$$

- The unknown is a matrix. Let us find a way to transform this into the form of $Ax = b$, where the unknown is a vector.

Mapping from world to image

- We have

$$\begin{aligned}x &= \mathbf{p}_1^T \cdot \mathbf{X} \\y &= \mathbf{p}_2^T \cdot \mathbf{X} \\z &= \mathbf{p}_3^T \cdot \mathbf{X}\end{aligned}$$

- This is the homogeneous image coordinate. Converting it to inhomogeneous image coordinate, we have

$$\begin{aligned}x &= \frac{\mathbf{p}_1^T \cdot \mathbf{X}}{\mathbf{p}_3^T \cdot \mathbf{X}} \\y &= \frac{\mathbf{p}_2^T \cdot \mathbf{X}}{\mathbf{p}_3^T \cdot \mathbf{X}}\end{aligned}$$

Reason for this step: from the image, we can know inhomogeneous image coordinate of some points.

- Rearrange it,

$$\begin{aligned}\mathbf{p}_1^T \cdot \mathbf{X} - \mathbf{p}_3^T \cdot \mathbf{X}x &= 0 \\\mathbf{p}_2^T \cdot \mathbf{X} - \mathbf{p}_3^T \cdot \mathbf{X}y &= 0\end{aligned}$$

Mapping from world to image

$$\begin{aligned}\mathbf{p}_1^T \cdot \mathbf{X} - \mathbf{p}_3^T \cdot \mathbf{X}x &= 0 \\ \mathbf{p}_2^T \cdot \mathbf{X} - \mathbf{p}_3^T \cdot \mathbf{X}y &= 0\end{aligned}$$

- Transpose both sides,

$$\begin{aligned}\mathbf{X}^T \mathbf{p}_1 - \mathbf{X}^T \mathbf{p}_3 x &= 0 \\ \mathbf{X}^T \mathbf{p}_2 - \mathbf{X}^T \mathbf{p}_3 y &= 0\end{aligned}$$

Reason for this step: formulate the unknown as a column vector.

- Write in matrix form,

$$\begin{bmatrix} \mathbf{X}^T & 0 & -\mathbf{X}^T x \\ 0 & \mathbf{X}^T & -\mathbf{X}^T y \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \mathbf{0}$$

2x12 matrix 12x1 vector.

Each \mathbf{p}_i is a 4x1 row vector

Equation system

- For one pair of 3D-2D coordinates $\{X_1, x_1\}$, we have

$$\begin{bmatrix} X_1^T & 0 & -X_1^T x_1 \\ 0 & X_1^T & -X_1^T y_1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = 0$$

- For n pairs of coordinates, we can form an overdetermined system,

$$\begin{bmatrix} X_1^T & 0 & -X_1^T x_1 \\ 0 & X_1^T & -X_1^T y_1 \\ \vdots & \vdots & \vdots \\ X_n^T & 0 & -X_n^T x_n \\ 0 & X_n^T & -X_n^T y_n \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = 0$$

2nx12 matrix

12x1 vector

Equation system

- The equation system is in the form of

$$A\mathbf{p} = 0$$

- The solution \mathbf{p} is the null space of matrix A .

- We can find the solution by formulating the following optimisation problem,

$$\begin{aligned} \mathbf{p} &= \underset{\mathbf{p}}{\operatorname{argmin}} \quad ||A\mathbf{p}||^2 \\ &\text{subject to } \|\mathbf{p}\|^2 = 1 \end{aligned}$$

This constraint is added as we are not interested in trivial solution $\mathbf{p} = \mathbf{0}$.

Solution

$$\begin{aligned} \mathbf{p} &= \underset{\mathbf{p}}{\operatorname{argmin}} \|\mathbf{A}\mathbf{p}\|^2 \\ &\text{subject to } \|\mathbf{p}\|^2 = 1 \end{aligned}$$

- Perform singular value decomposition (SVD) for matrix A .

$$A = U\Sigma V^T$$

The diagram shows the SVD formula $A = U\Sigma V^T$ with arrows pointing from the labels to the corresponding parts of the equation. Below the formula, the dimensions of the matrices are specified: U is $m \times n$, Σ is $m \times m$, and V^T is $n \times n$. Below Σ , it is labeled as a "diagonal matrix". Below U and V^T , it is labeled as "orthogonal matrix".

$m \times n$
matrix

$m \times m$
orthogonal

$m \times n$
diagonal
matrix

$n \times n$
orthogonal
matrix

- The solution \mathbf{p} is the column of V that corresponds to the smallest singular value.

Camera calibration

- After obtaining the solution \mathbf{p} , which is a vector, we re-arrange its elements to form the camera matrix P .
- As a result, we establish the mapping from 3D world coordinates to 2D image coordinates.

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

Camera calibration

- Given a set of points $\{X_i, x_i\}$ and the camera model

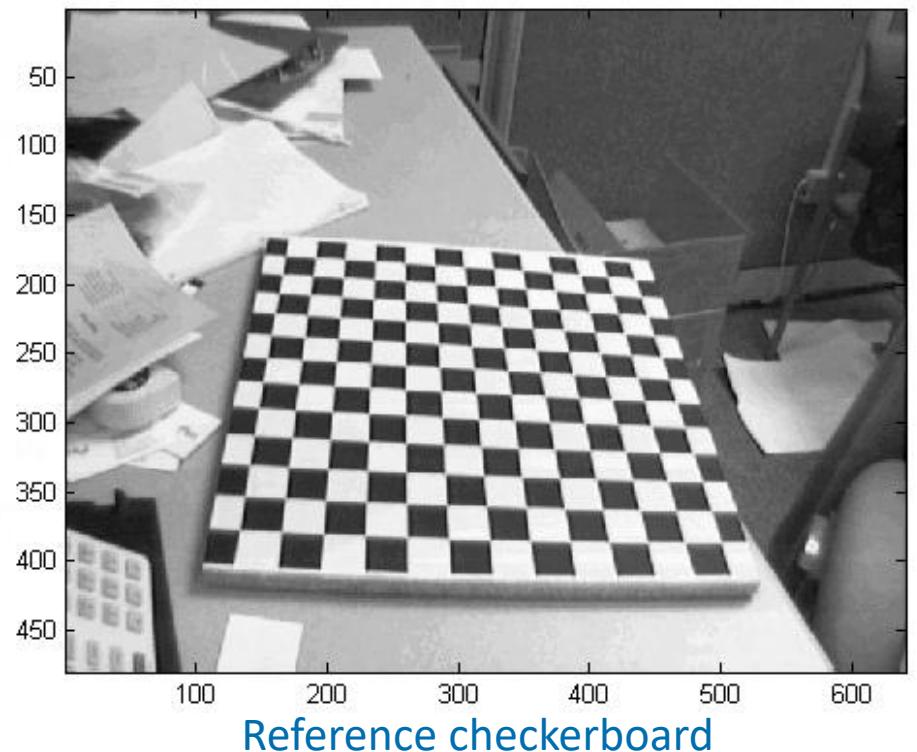
$$x = P X$$

we can estimate the matrix P .

- But where do we get these points $\{X_i, x_i\}$?

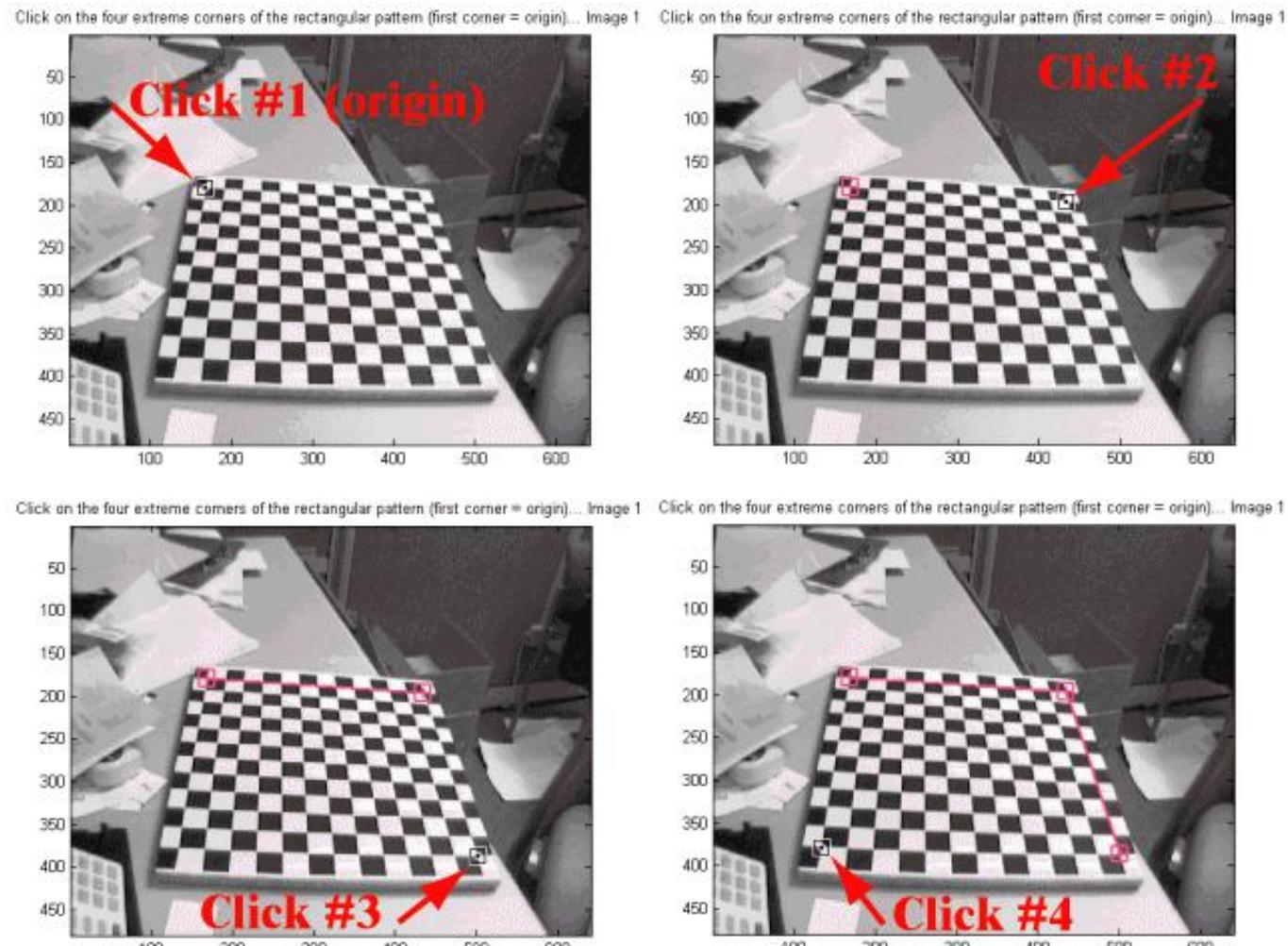
Using a reference object

- Place a reference object in the scene.
- Identify correspondences between points in the image and in the scene.
- Compute the mapping from scene to image.



Using a reference object

- We can click at corners of this checkerboard, defining their 3D coordinates X .
- We also know their image coordinates x .
- In this way, we obtain the coordinates for points $\{X_i, x_i\}$.



Camera model

- With the camera matrix, we can relate the information from 3D world to 2D images.
 - Smartphone apps
 - Object tracking for drones



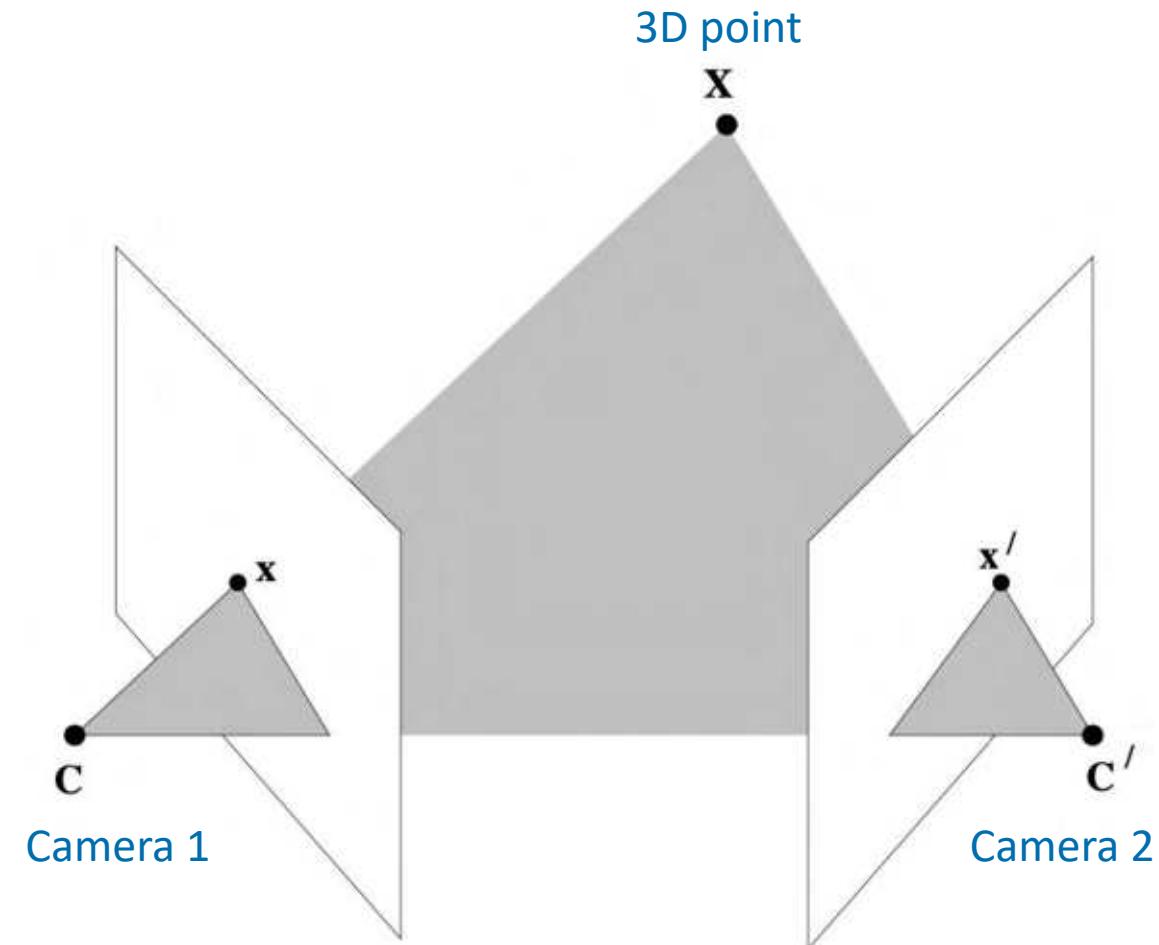
Apple ARKit



Pokemon Go

Multiple view geometry

- Here we only consider a single view of the camera.
- When we move the camera and take images from different perspectives, multiple view geometry is involved.
- Due to time limit, we do not go into detail.



Summary

- A little bit of 3D computer vision.
 - Pinhole camera model
 - Camera calibration

References

- Ch. 6 Camera models. Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*.

Review Lecture

Dr Wenjia Bai

Department of Computing & Brain Sciences

We have learnt

Reflection, optics,
vision system,
image representation

Image formation

Image filtering,
edge detection,
interest point detection,
feature descriptor

Image processing

Image classification,
object detection,
image segmentation,
action recognition

Image understanding

Camera model,
camera calibration

3D vision

Optic flow,
object tracking,
image alignment

Motion

Mathematical foundation: calculus, linear algebra, statistics, geometry

You may want to learn more

Supervised learning,
unsupervised learning,
reinforcement learning,
geometric deep learning

.....

Machine learning

Multiple view geometry,
3D reconstruction,
3D shape analysis

.....

3D vision

Computer vision +
natural language
processing, robotics,
medical imaging

.....

Applications

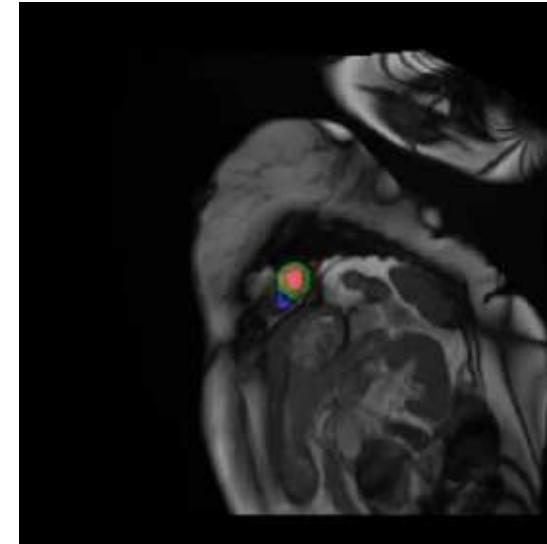
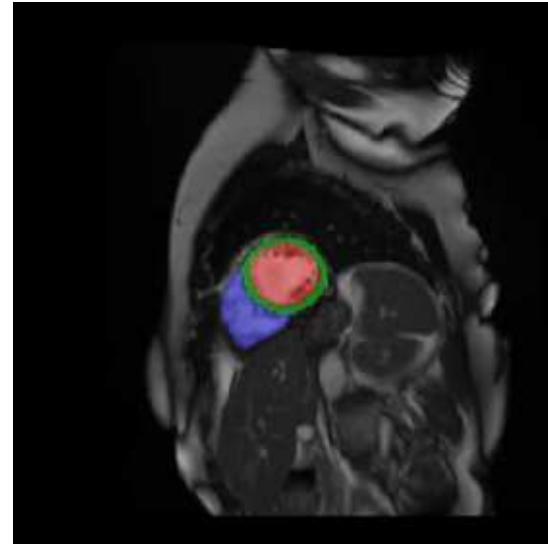
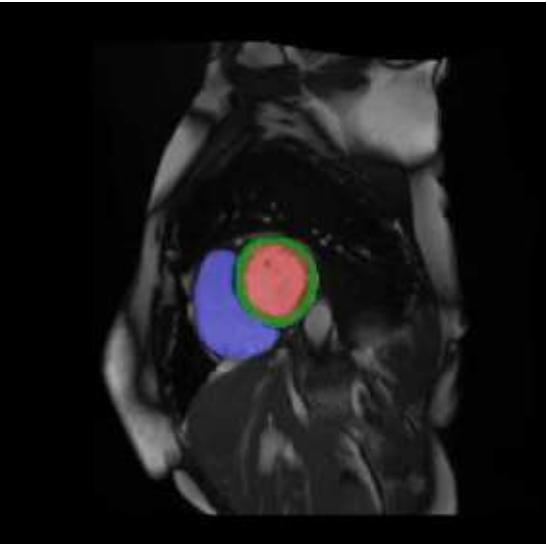
Mathematical foundation: calculus, linear algebra, statistics, geometry

Applications

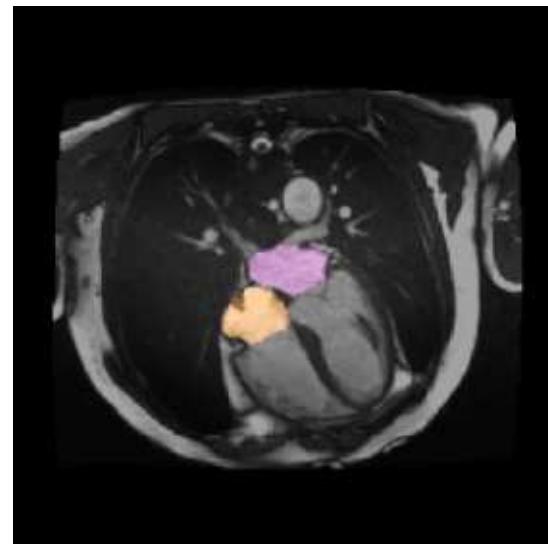
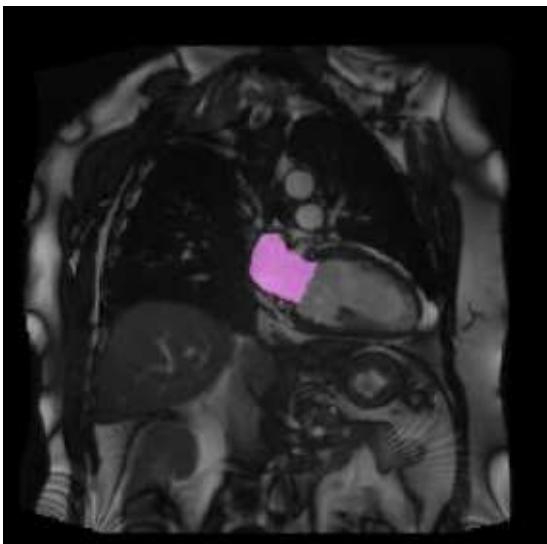
- Medical image computing
 - Computer vision + clinical domain knowledge + medical physics + ...
- Autonomous driving
 - Computer vision + control + hardware + ...
- Robotics
 - Computer vision + control + mechanics + materials + ...

Cardiac image segmentation

Ventricles



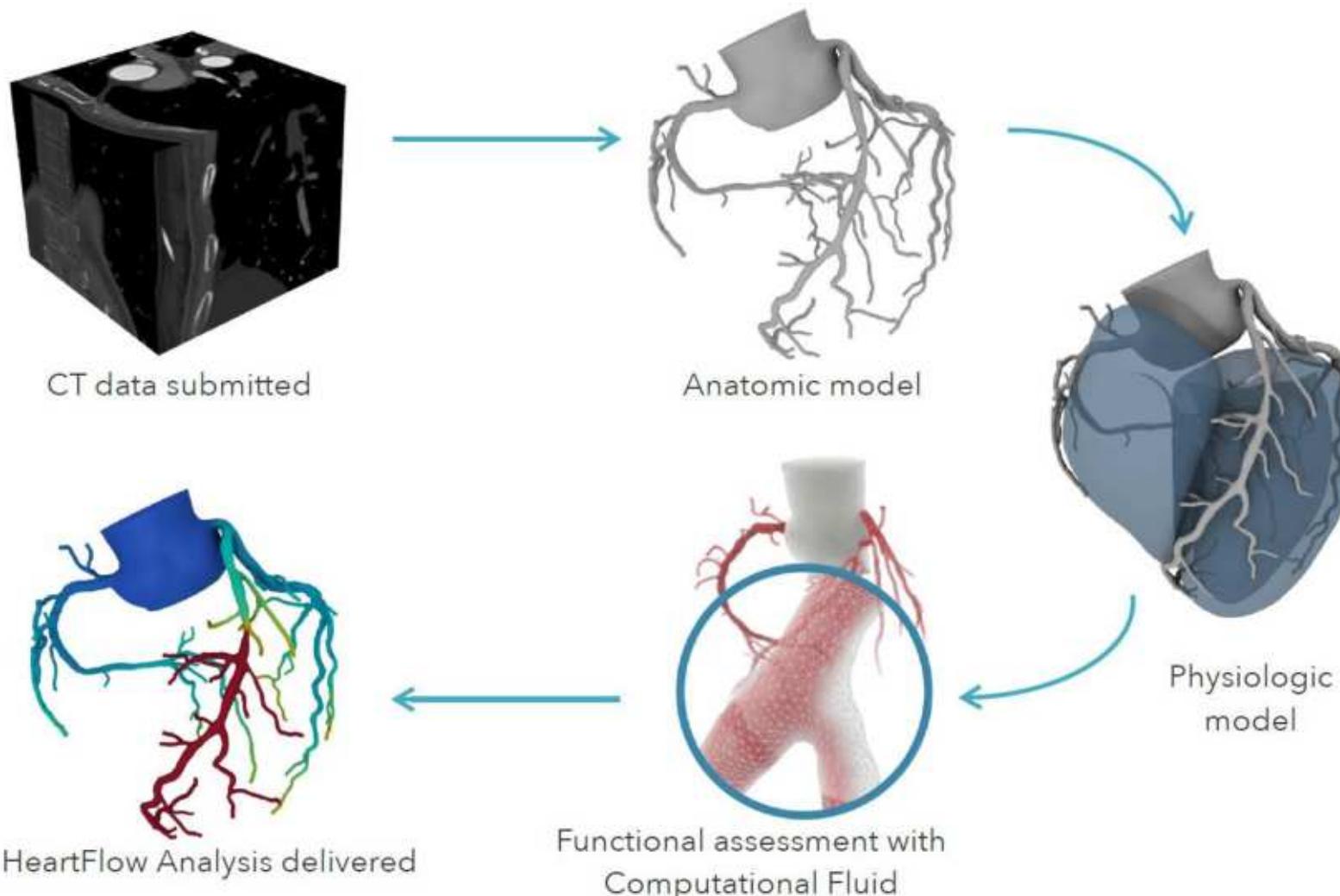
Atria



■ LV cavity ■ RV cavity ■ LA cavity ■ RA cavity
■ LV myocardium

Bai et al. JCMR, 2018.

Coronary artery segmentation



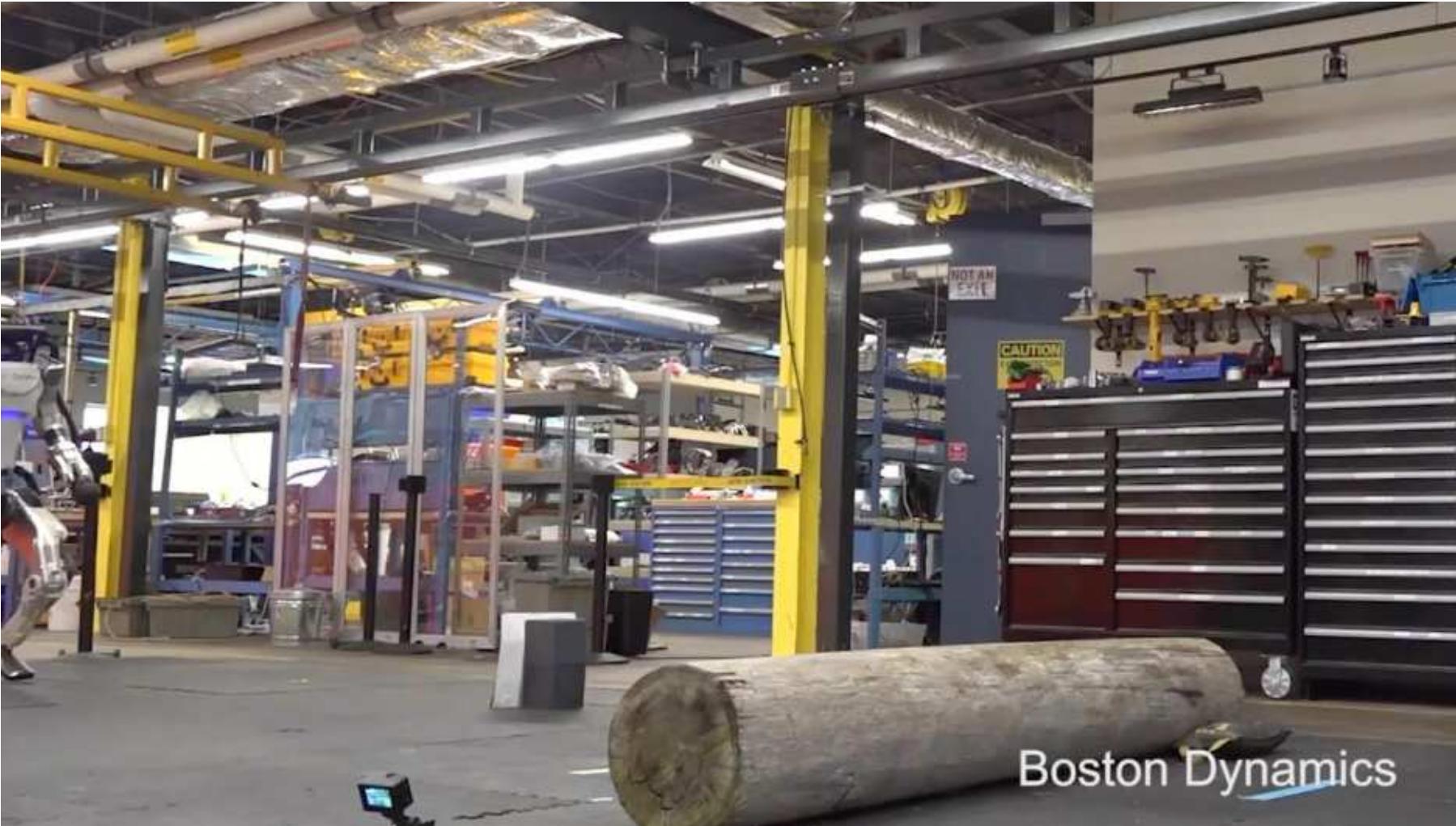
The segmented arteries provide a physiologic model for disease treatment planning. ©HeartFlow

Autonomous driving



Paris streets in the eyes of Tesla Autopilot

Robotics



Atlas uses computer vision to locate itself with respect to visible markers on the approach. @BostonDynamics

Limitations

- Many modern computer vision algorithms are based on deep neural networks, which are very powerful in learning features.
- However, these approaches might have certain limitations:
 - Data hungry.
 - May be over-parameterised, not easy for mobile deployment.
 - May not generalise well, especially on data from an unseen domain.
 - May not be well integrated with prior knowledge, logic and human values.
 - ...

Exam format

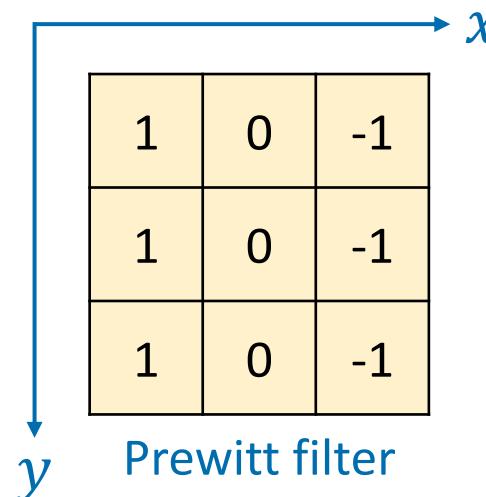
- Timetable: <https://exams.doc.ic.ac.uk/prog/generaltimetable.cgi>

COMP60006=COMP96046	Computer Vision (Term 2)	21-Mar	10:00	120 min
---------------------	--------------------------	--------	-------	---------

- Paper-based exam
 - Similar to past papers, COMP60006: Computer Vision (Term2) on <https://exams.doc.ic.ac.uk/pastpapers>
 - Similar to tutorial questions on Scientia
- You are allowed to bring a A4 sheet of notes.

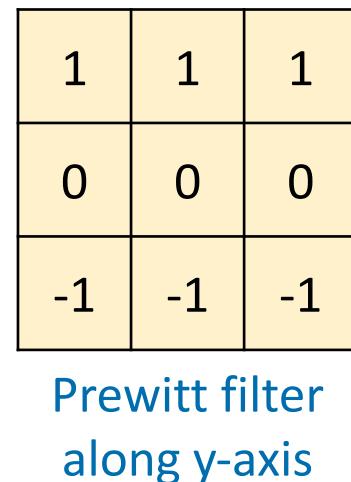
What do I need to know?

- How to perform image filtering using Prewitt or Sobel filters (flip-and-shift)?



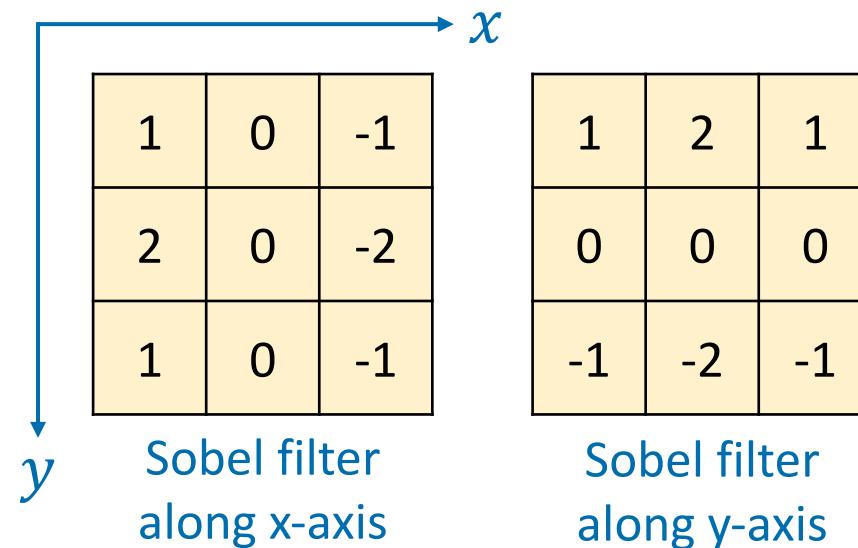
A 3x3 matrix representing a Prewitt filter kernel. The x-axis is indicated by a blue arrow pointing right, and the y-axis is indicated by a blue arrow pointing down. The matrix values are:

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



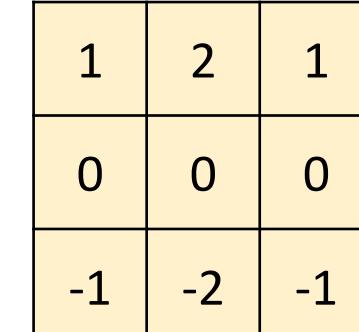
A 3x3 matrix representing a Prewitt filter kernel. The x-axis is indicated by a blue arrow pointing right, and the y-axis is indicated by a blue arrow pointing down. The matrix values are:

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



A 3x3 matrix representing a Sobel filter kernel. The x-axis is indicated by a blue arrow pointing right, and the y-axis is indicated by a blue arrow pointing down. The matrix values are:

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



A 3x3 matrix representing a Sobel filter kernel. The x-axis is indicated by a blue arrow pointing right, and the y-axis is indicated by a blue arrow pointing down. The matrix values are:

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

Image filtering

- Why do we need to perform image filtering?
- What are the commonly used filters?
- How to do differential calculus for some filters?

Feature description

- How to describe features for points and for images?
- Common image feature detectors and descriptors.
- How to perform image matching?

Image classification

- Concepts of image classification, object detection and image segmentation
- Understand several network architectures
 - Multi-layer perceptron (MLP)
 - Convolutional neural network (CNN)
 - Vision Transformer (ViT)

Motion and camera

- Optic flow constraint equation
- Motion estimation based on the optic flow constraint
- Object tracking
- Relationship between world coordinate, camera coordinate and image coordinate

What do I not need to know?

- You do not need to know
 - Derivation of the forward propagation and backpropagation algorithm in the supplementary slides

Any questions?

I hope you enjoy the course.
Good luck!