



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision 2023-24 First Semester, M.Tech (AIML)

Session #1: Introduction to the Course

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

S. P. Vimal | CSIS Department | WILP | BITS – Pilani (vimalsp@wilp.bits-pilani.ac.in)

Every image tells a story



- Goal of computer vision:
perceive the “story” behind
the picture
- Compute properties of the
world
 - 3D shape
 - Names of people or objects
 - What happened?

The goal of computer vision



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

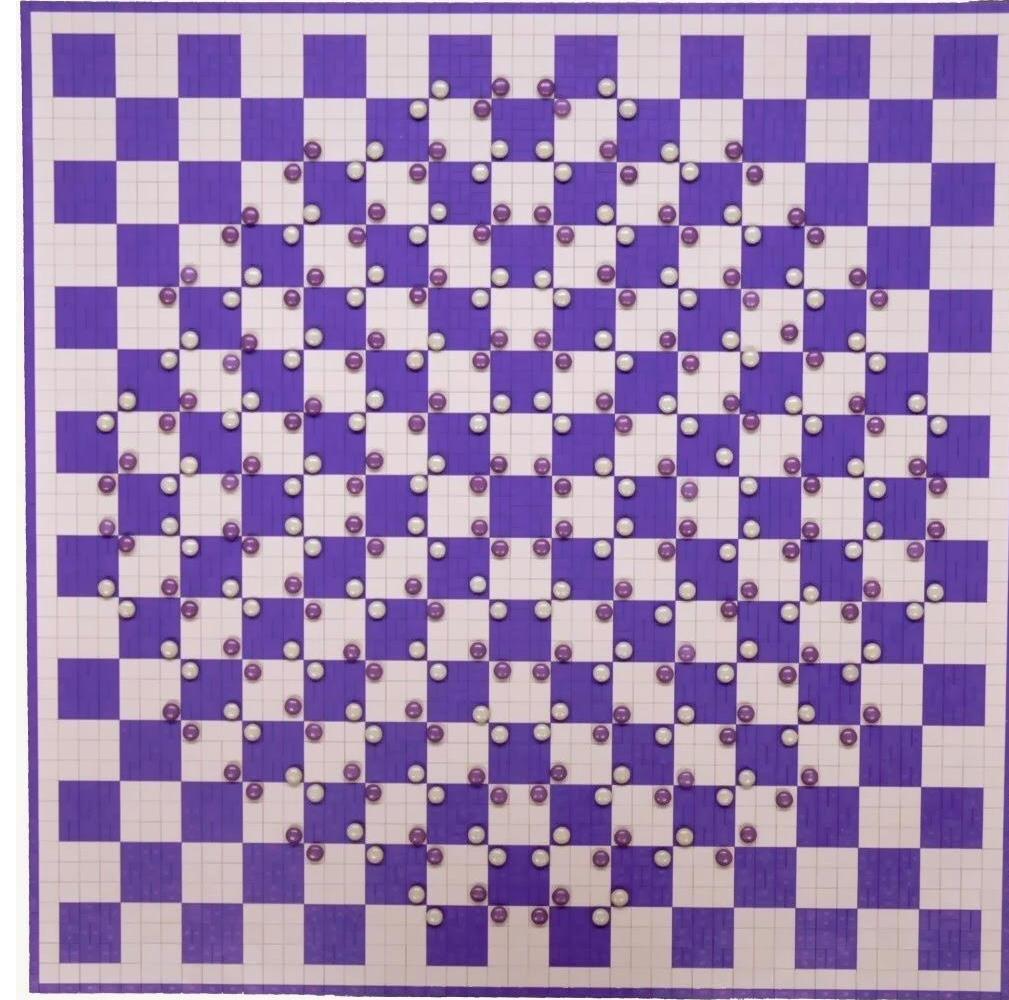
Can computers match human perception?



- Yes and no (mainly no)
 - computers can be better at “easy” things
 - humans are better at “hard” things
- But huge progress
 - Accelerating in the last five years due to deep learning
 - What is considered “hard” keeps changing

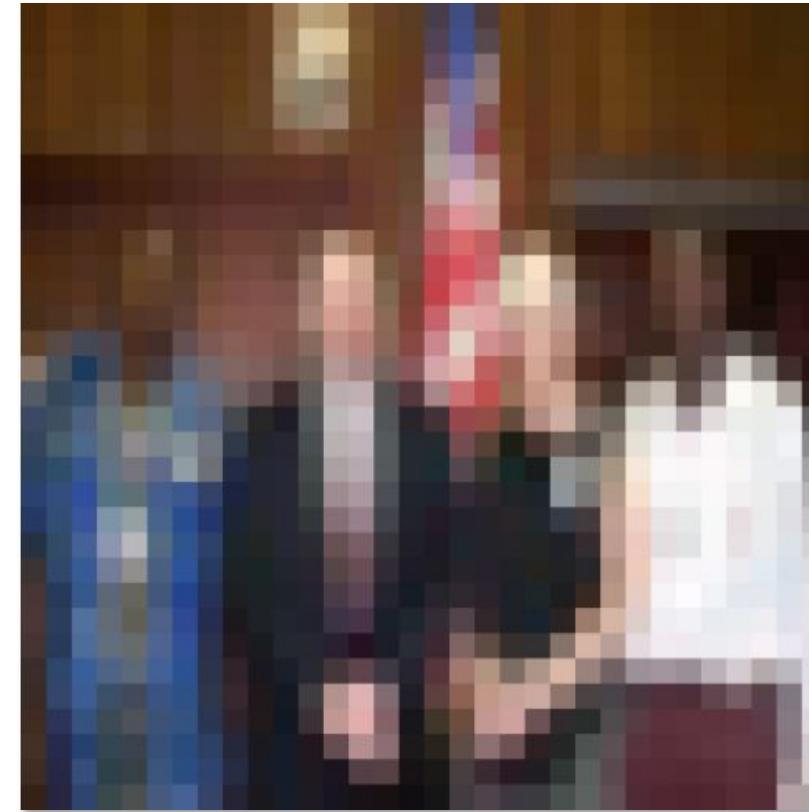


Human perception has its shortcomings



<https://twitter.com/pickover/status/1460275132958662657/>

But humans can tell a lot about a scene from a little information...



Source: "80 million tiny images" by Torralba, et al.





The goal of computer vision

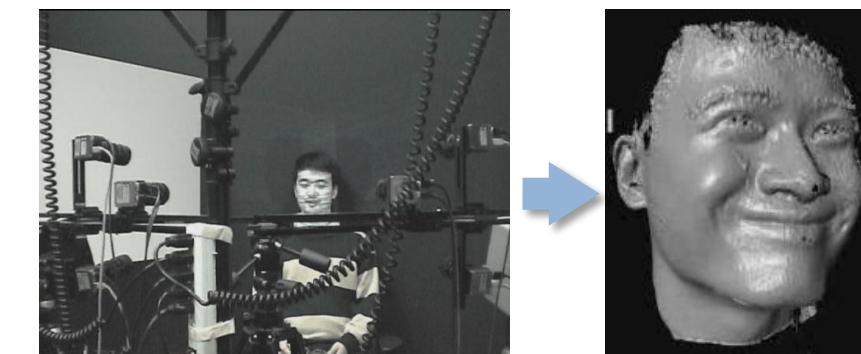
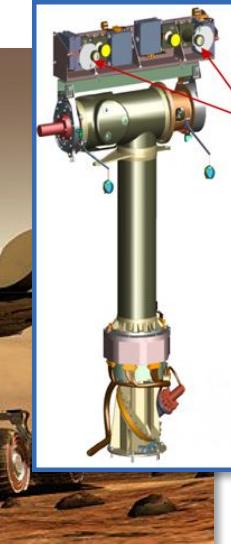
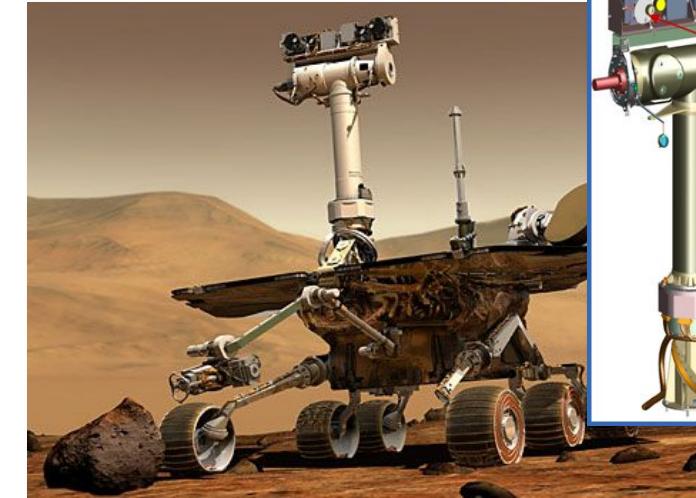
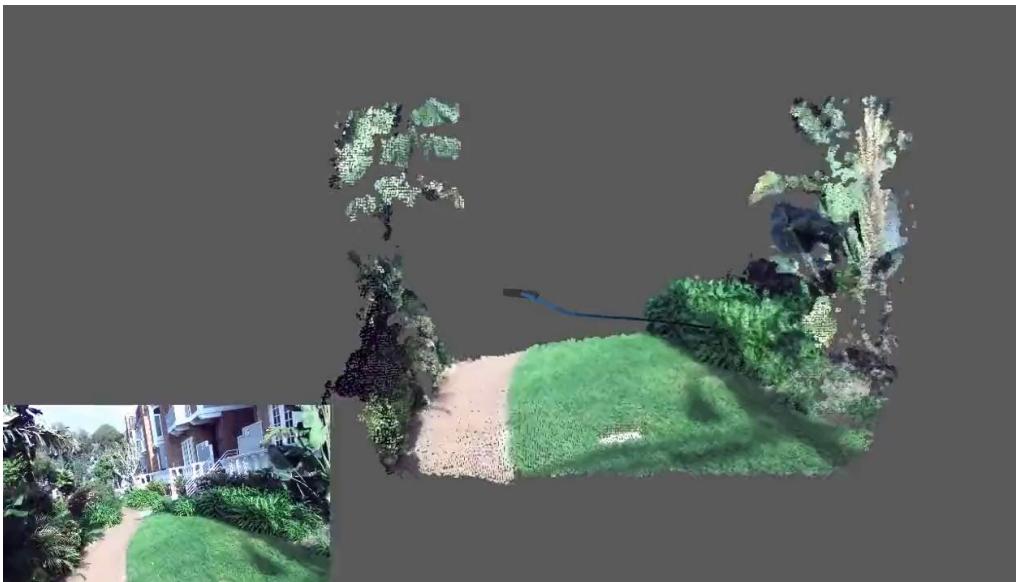


The goal of computer vision

- Compute the 3D shape of the world



ZED 2i Camera



The goal of computer vision

- Recognize objects and people



Terminator 2,
1991





The goal of computer vision

- “Enhance” images



The goal of computer vision

- Improve photos ("Computational Photography")



Super-resolution (source: 2d3)



Low-light photography
(credit: [Hasinoff et al., SIGGRAPH ASIA 2016](#))



Depth of field on cell phone camera
(source: [Google Research Blog](#))



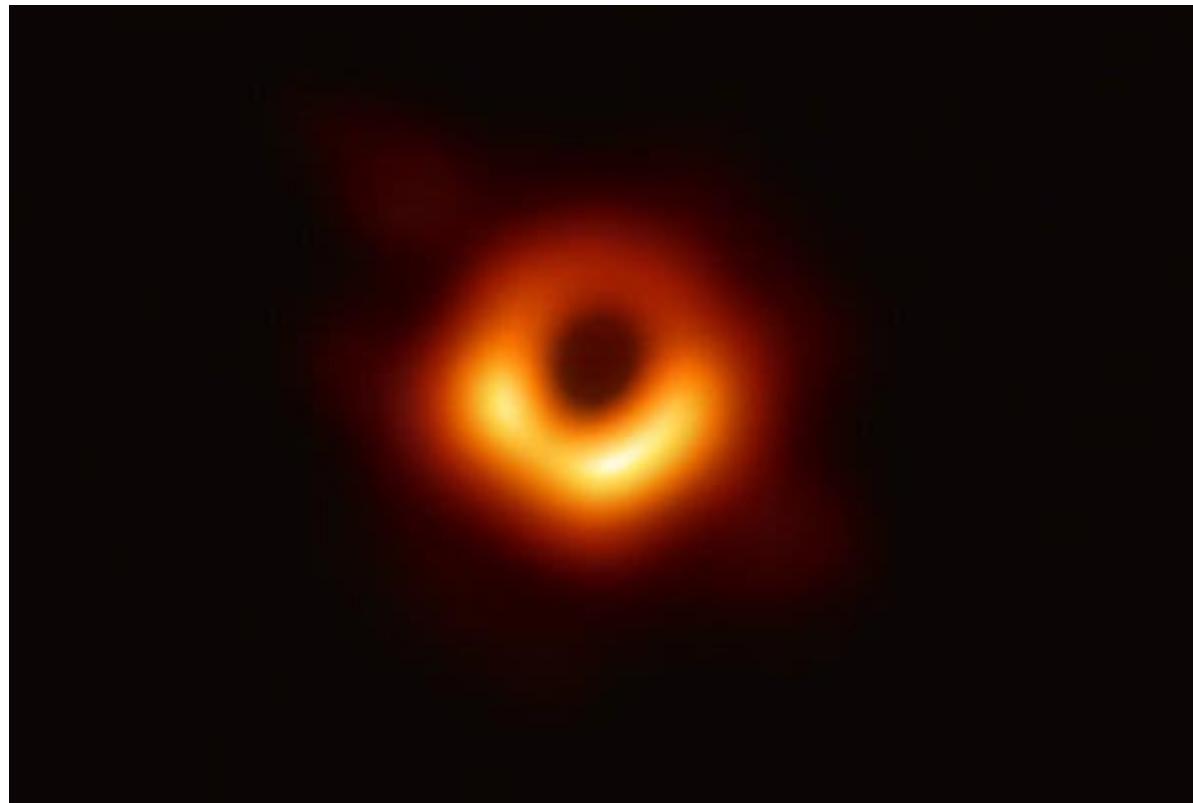
Removing objects
([Google Magic Eraser](#))



Darkness Visible, Finally: Astronomers Capture First Ever Image of a Black Hole

Astronomers at last have captured a picture of one of the most secretive entities in the cosmos.

April 10, 2019

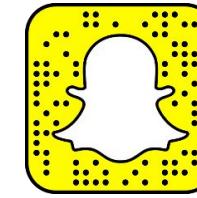


Why study computer vision?

- Billions of images/videos captured per day



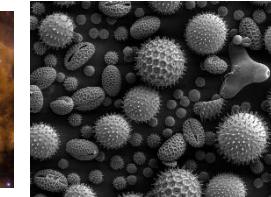
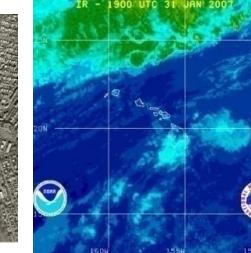
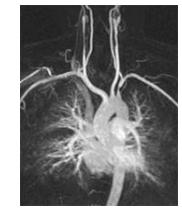
flickr



Google Photos



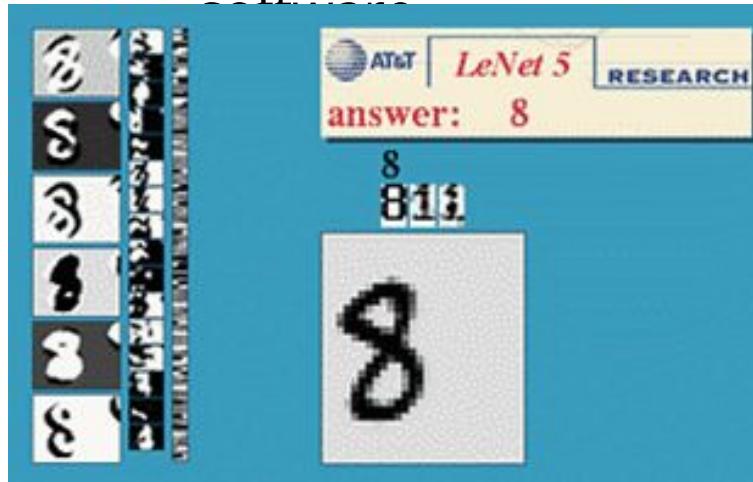
YouTube



- Huge number of potential applications

Optical character recognition (OCR)

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



Digit recognition, AT&T labs
(1990's)

<http://yann.lecun.com/exdb/len>



Automatic check

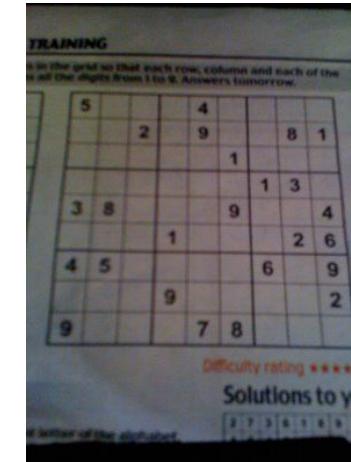
LYCH428

LYCH428

LYCH428

License plate readers

http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

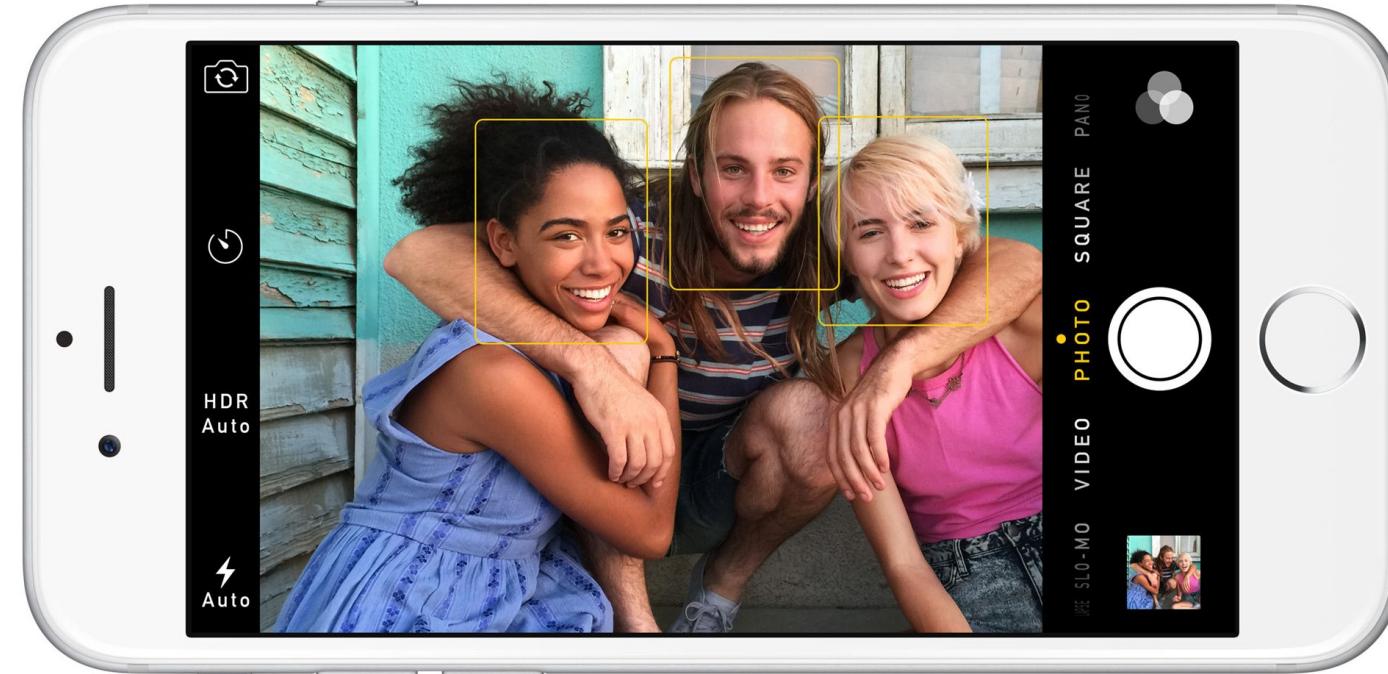


Sudoku grabber

<http://sudokugrab.blogspot.com/>

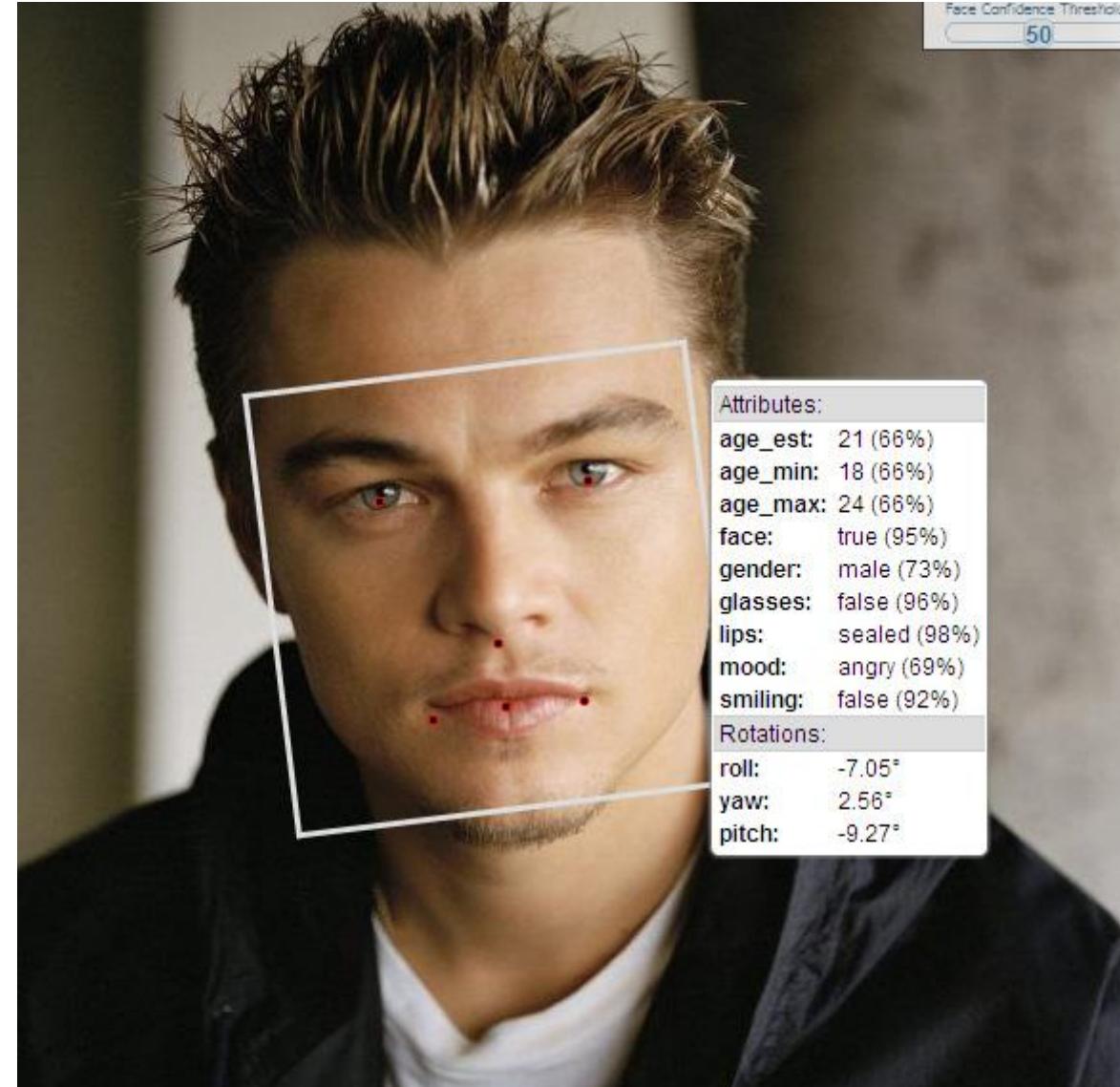


Face detection



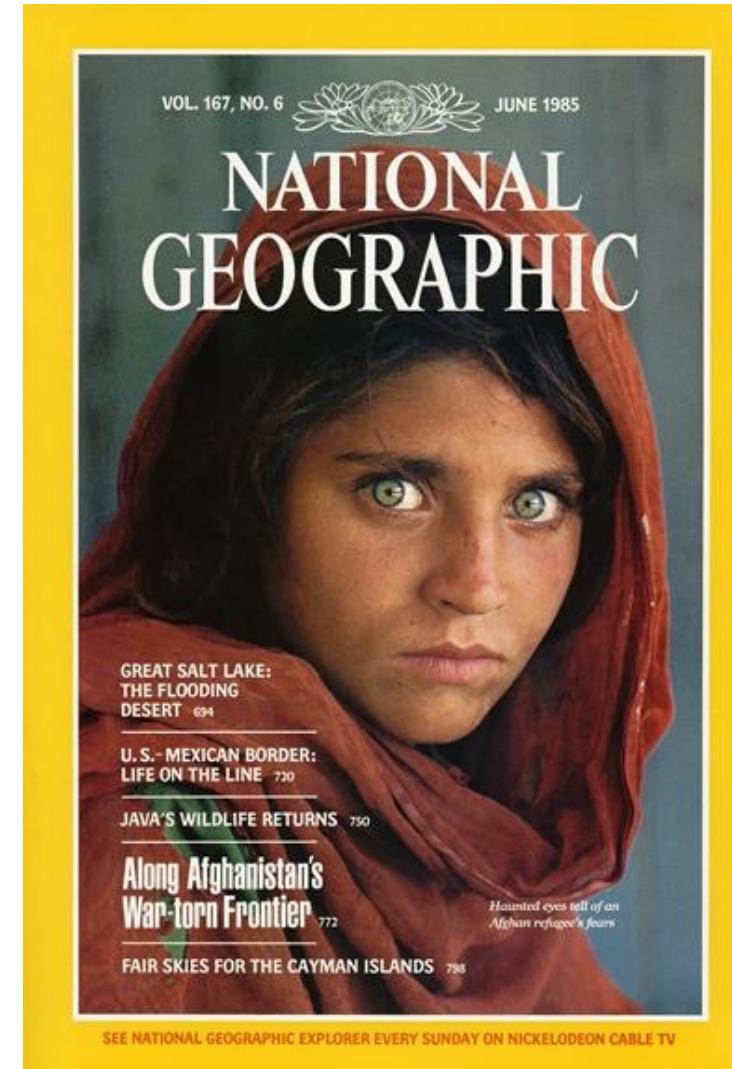
- Nearly all cameras detect faces in real time
 - (Why?)

Face analysis and recognition





Vision-based biometrics



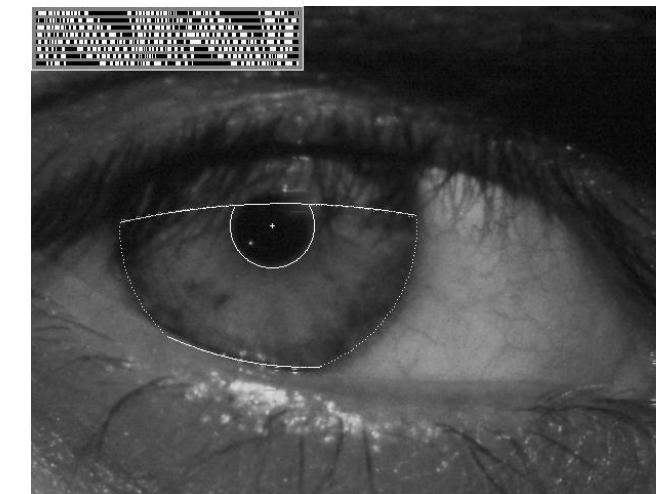
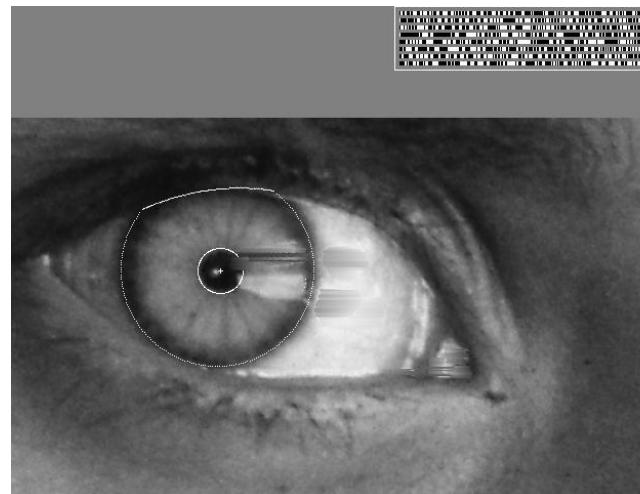
Who is she?

Source: S.
Seitz

Vision-based biometrics

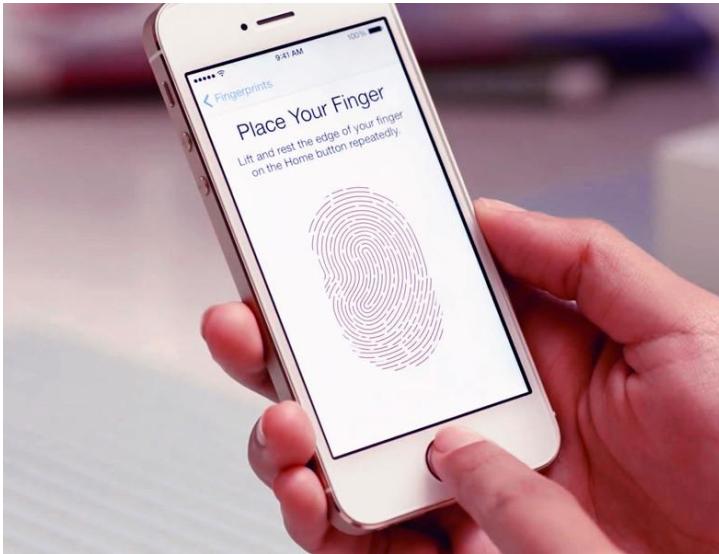


"How the Afghan Girl was Identified by Her Iris Patterns" [[Story](#) , [youtube link](#)]



Source: S.
Seitz

Login without a password



Fingerprint scanners on
many new smartphones
and other devices

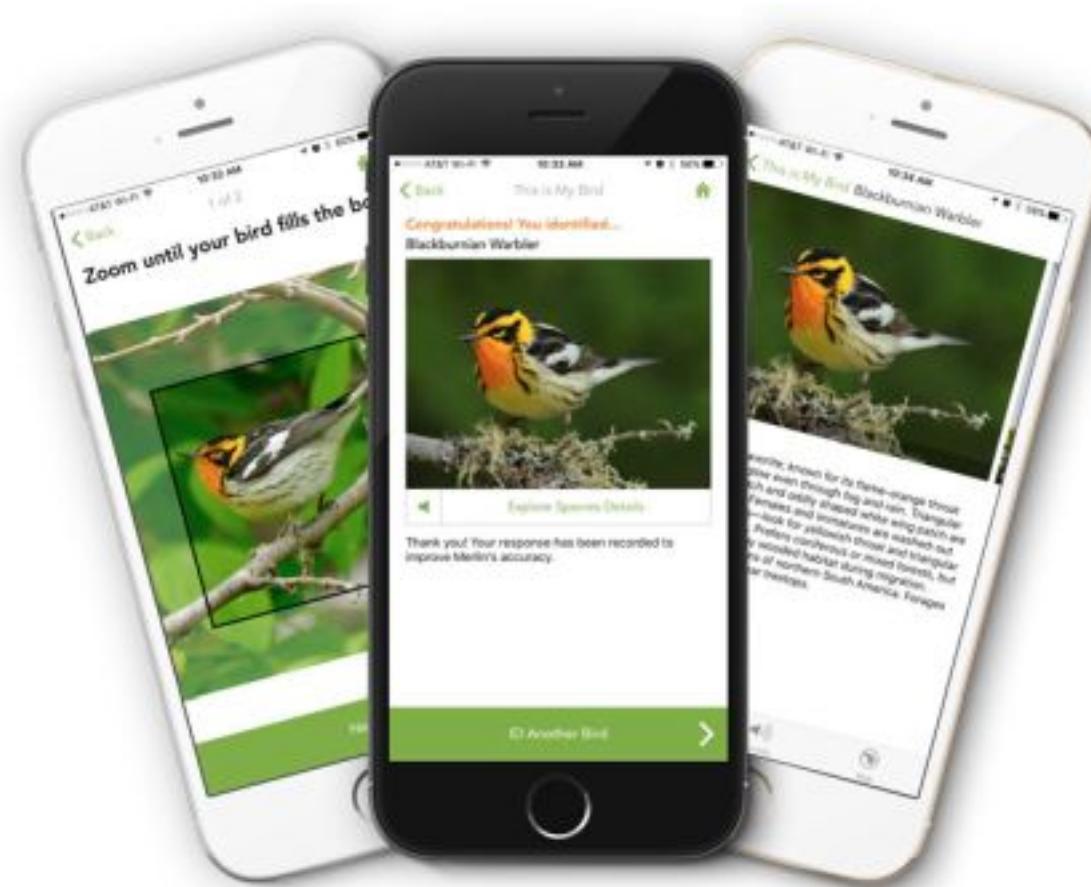


Face unlock on Apple iPhone X
See also

<http://www.sensiblevision.com/>



Bird identification



Merlin Bird ID (based on Cornell Tech technology!)

Special effects: shape capture



The Matrix movies, ESC Entertainment,
XYZRGB, NRC

Source: S.
Seitz

Special effects: motion capture

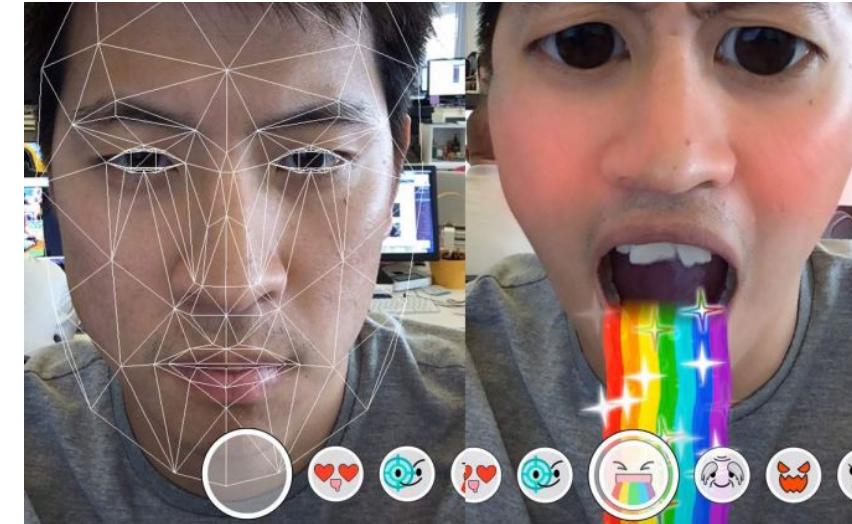


Pirates of the Caribbean, Industrial Light and Magic

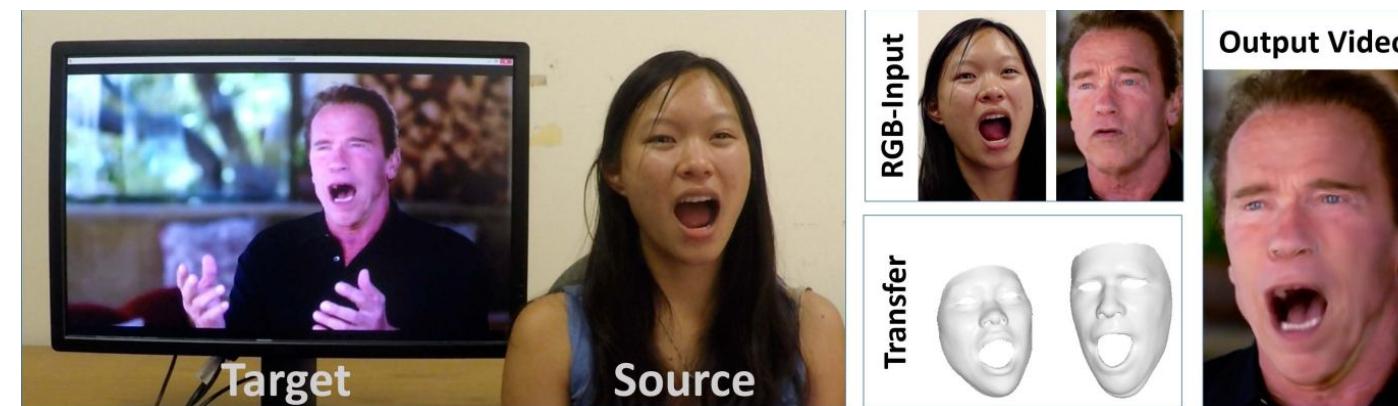
Source: S.
Seitz



3D face tracking w/ consumer cameras



Snapchat Lenses



Face2Face system (Thies et al.)

Image synthesis

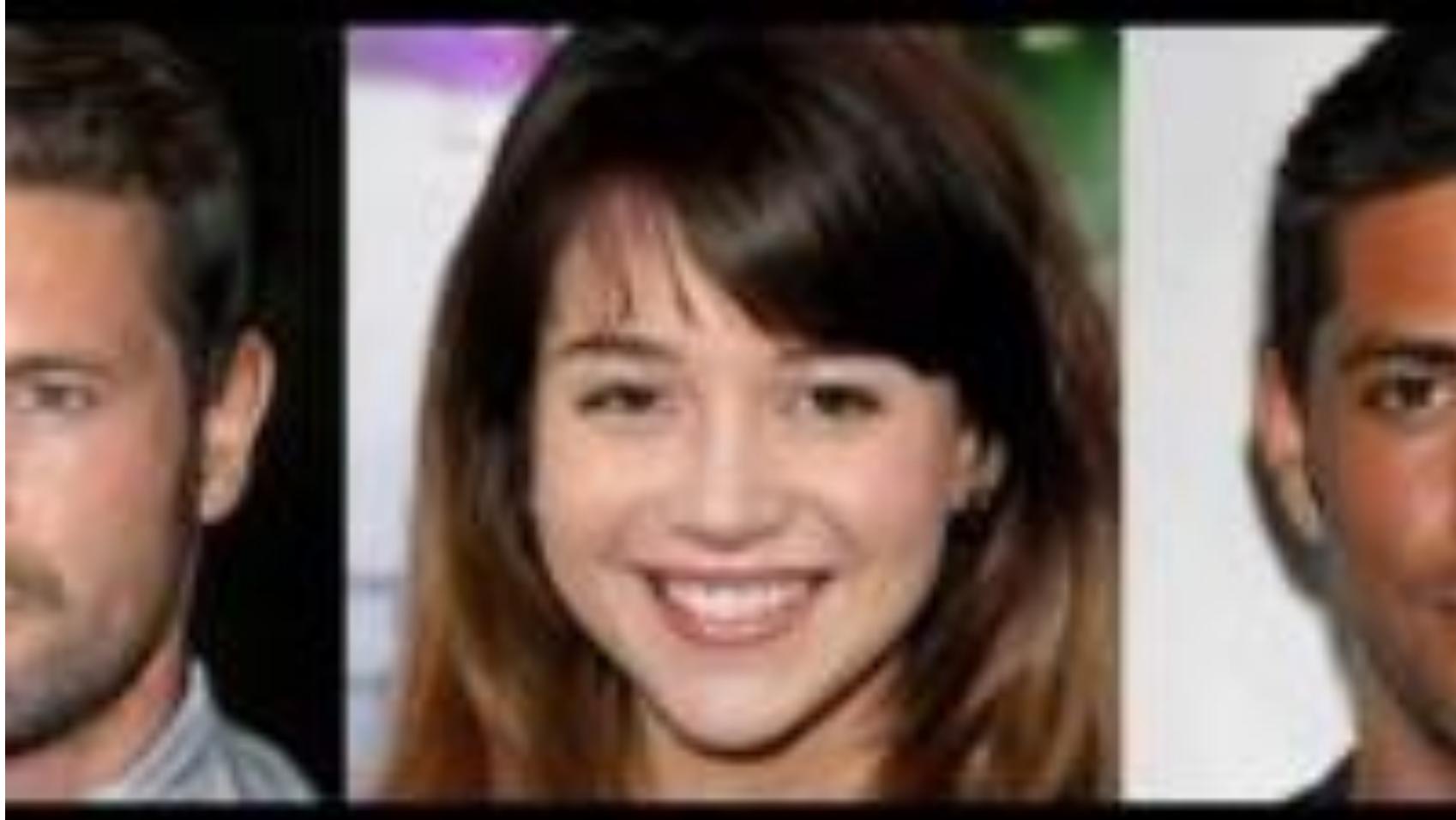


Image synthesis



"An astronaut riding a horse in a photorealistic style" – DALL-E 2

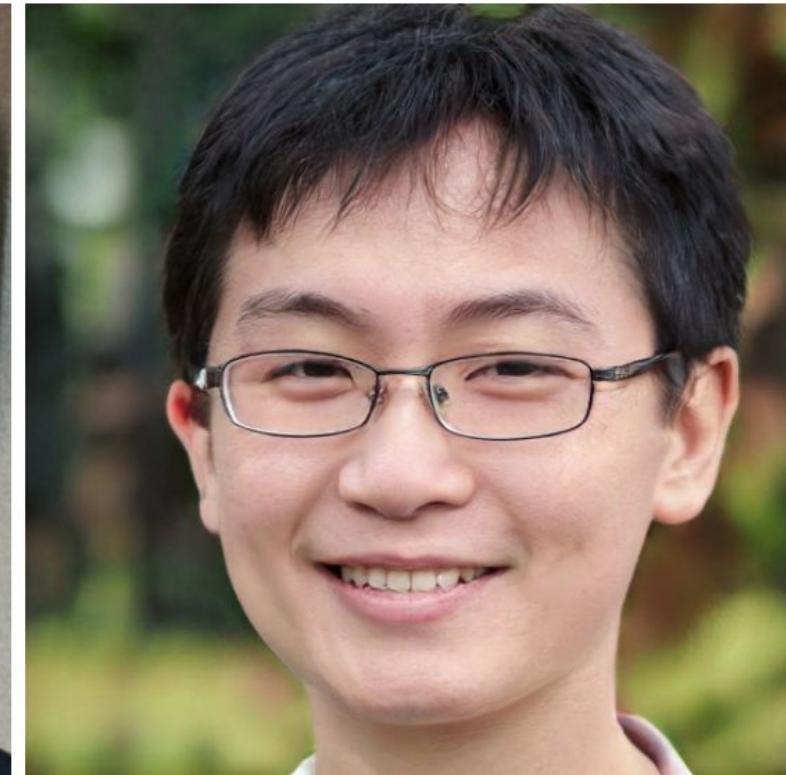


"A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat" – Imagen



Which face is real?

Click on the person who is real.



<https://www.whichfaceisreal.com/>

Smart cars

▷▶ manufacturer products consumer products ◀◀

Our Vision. Your Safety.



rear looking camera forward looking camera
side looking camera

› **EyeQ** Vision on a Chip  [read more](#)

› **Vision Applications**  Road, Vehicle, Pedestrian Protection and more [read more](#)

› **AWS** Advance Warning System  [read more](#)

News

› Mobileye Advanced Technologies Power Volvo Cars World First Collision Warning With Auto Brake System

› Volvo: New Collision Warning with Auto Brake Helps Prevent Rear-end

[all news](#)



Events

› Mobileye at Equip Auto, Paris, France

› Mobileye at SEMA, Las Vegas, NV

[read more](#)

- Mobileye
- Tesla Autopilot
- Safety features in many cars

Robotics



NASA's Mars Curiosity Rover
[https://en.wikipedia.org/wiki/Curiosity_\(rover\)](https://en.wikipedia.org/wiki/Curiosity_(rover))



Amazon Picking Challenge
<http://www.robocup2016.org/en/events/amazon-picking-challenge/>

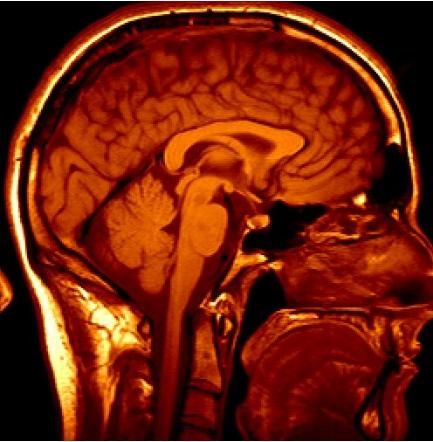


Amazon Prime
Air

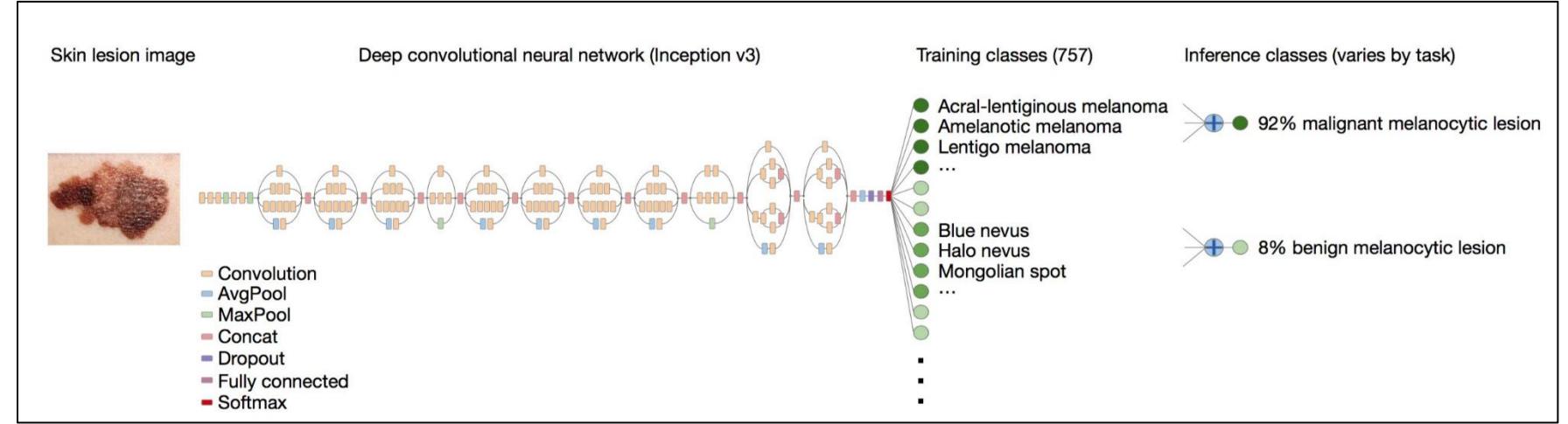


Amazon
Scout

Medical imaging



3D imaging
(MRI, CT)



Skin cancer classification with deep learning
<https://cs.stanford.edu/people/esteva/nature/>



Current state of the art

- You just saw many examples of current systems.
 - Many of these are less than 5 years old
- Computer vision is an active research area, and rapidly changing
 - Many new apps in the next 5 years
 - Deep learning powering many modern applications
- Many startups across a dizzying array of areas
 - Deep learning, robotics, autonomous vehicles, medical imaging, construction, inspection, VR/AR, ...

Why is computer vision difficult?



Viewpoint
variation



Illuminatio
n



Credit: Flickr user michaelpaul

Scal
e

Why is computer vision difficult?



Intra-class
variation



Background
clutter

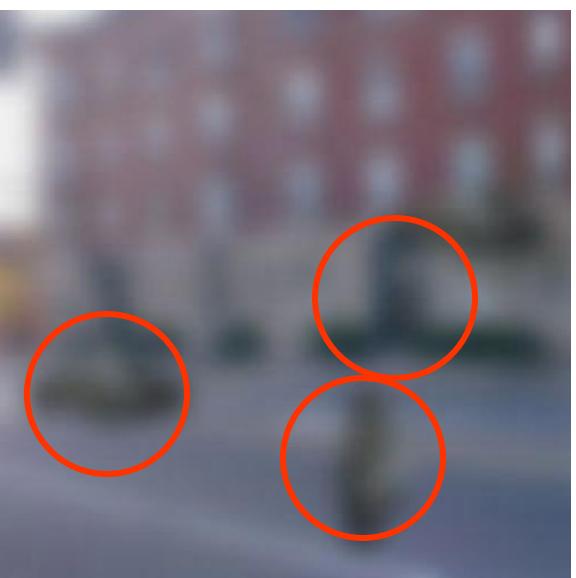
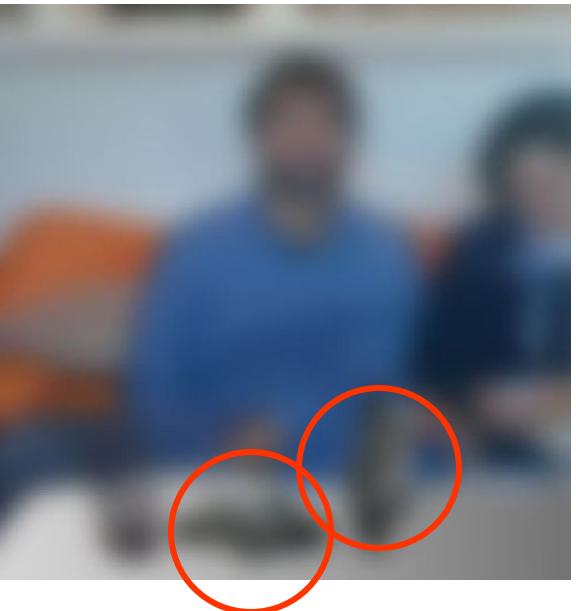


Motion (Source: S.
Lazebnik)

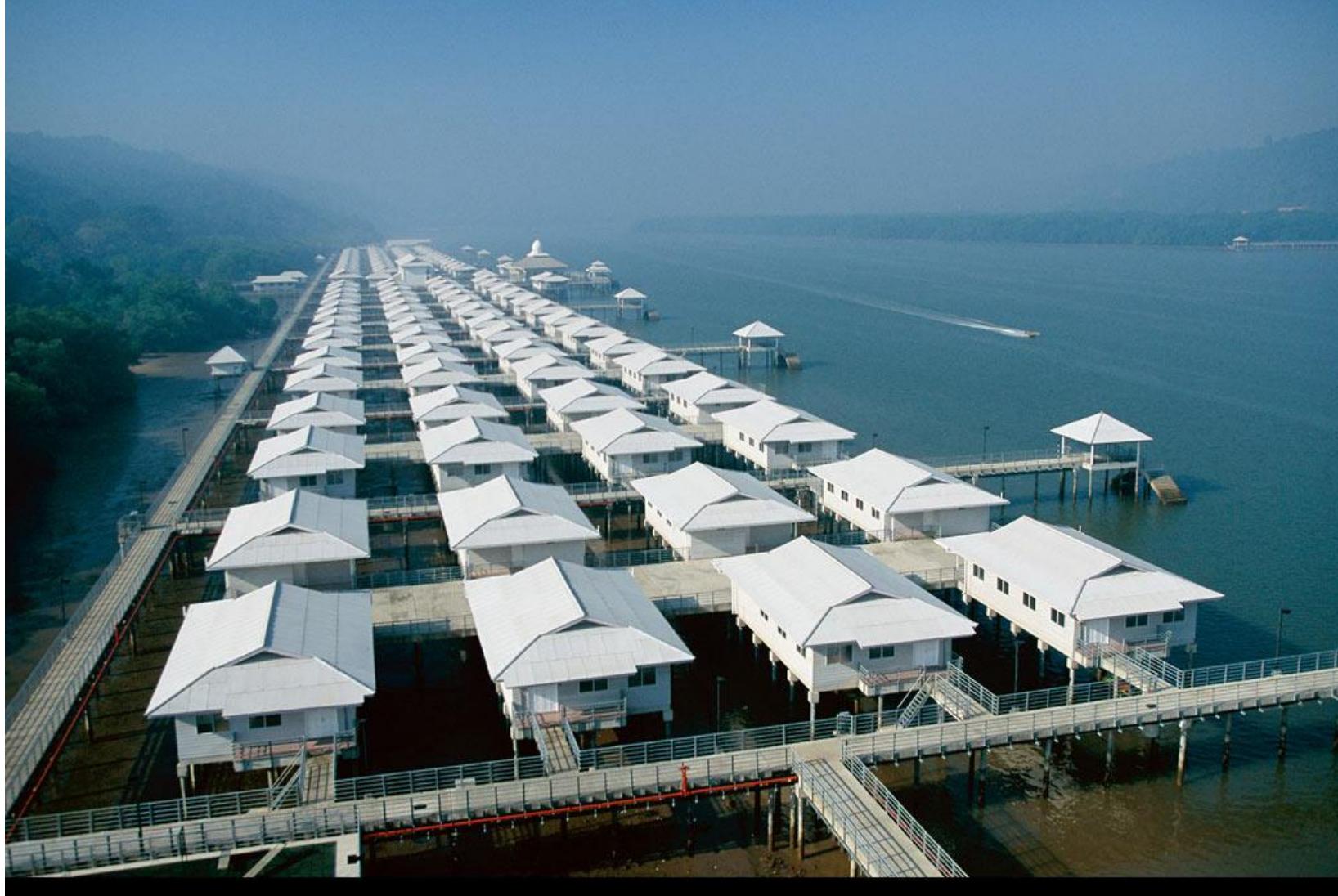


Occlusio
n

Challenges: local ambiguity



But there are lots of visual cues we can use...



Bottom line

- Perception is an inherently ambiguous problem
 - Many different 3D scenes could have given rise to a given 2D image



- We often must use prior knowledge about the world's structure



The state of Computer Vision and AI: we are really, really far.

Oct 22, 2012



The picture above is funny.

But for me it is also one of those examples that make me sad about the outlook for AI and for Computer Vision. What would it take for a computer to understand this image as you or I do? I challenge you to think explicitly of all the pieces of knowledge that have to fall in place for it to make sense. Here is my short attempt:

- You recognize it is an image of a bunch of people and you understand they are in a hallway
- You recognize that there are 3 mirrors in the scene so some of those people are "fake" replicas from different viewpoints.
- You recognize Obama from the few pixels that make up his face. It helps that he is in his suit and that he is surrounded by other people with suits.
- You recognize that there's a person standing on a scale, even though the scale occupies only very few white pixels that blend with the background. But, you've used the person's pose and knowledge of how people interact with objects to figure it out.
- You recognize that Obama has his foot positioned just slightly on top of the scale. Notice the language I'm using: It is in terms of the 3D structure of the scene, not the position of the leg in the 2D coordinate system of the image.
- You know how physics works: Obama is leaning in on the scale, which applies a force on it. Scale measures force that is applied on it, that's how it works => it will over-estimate the weight of the person standing on it.
- The person measuring his weight is not aware of Obama doing this. You derive this because you know his pose, you understand that the field of view of a person is finite, and you understand that he is not very likely to sense the slight push of Obama's foot.
- You understand that people are self-conscious about their weight. You also understand that he is reading off the scale measurement, and that shortly the over-estimated weight will confuse him because it will probably be much higher than what he expects. In other words, you reason about implications of the events that are about to unfold seconds after this photo was taken, and especially about the thoughts and how they will develop inside people's heads. You also reason about what pieces of information are available to people.
- There are people in the back who find the person's imminent confusion funny. In other words you are reasoning about state of mind of people, and their view of the state of mind of another person. That's getting frighteningly meta.
- Finally, the fact that the perpetrator here is the president makes it maybe even a little more funny. You understand what actions are more or less likely to be undertaken by different people based on their status and identity.



The state of Computer Vision and AI: we are really, really far.

Oct 22, 2012



The picture above is funny.



But for me it is also one of those examples that make me sad about the outlook for AI and for Computer Vision. What would it take for a computer to understand this image as you or I do? I challenge you to think explicitly of all the pieces of knowledge that have to fall in place for it to make sense. Here is my short attempt:

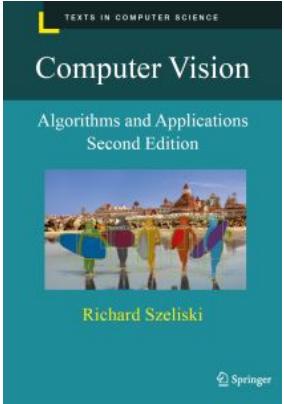
- You recognize it is an image of a bunch of people and you understand they are in a hallway
- You recognize that there are 3 mirrors in the scene so some of those people are "fake" replicas from different viewpoints.
- You recognize Obama from the few pixels that make up his face. It helps that he is in his suit and that he is surrounded by other people with suits.
- You recognize that there's a person standing on a scale, even though the scale occupies only very few white pixels that blend with the background. But, you've used the person's pose and knowledge of how people interact with objects to figure it out.
- You recognize that Obama has his foot positioned just slightly on top of the scale. Notice the language I'm using: It is in terms of the 3D structure of the scene, not the position of the leg in the 2D coordinate system of the image.
- You know how physics works: Obama is leaning in on the scale, which applies a force on it. Scale measures force that is applied on it, that's how it works => it will over-estimate the weight of the person standing on it.
- The person measuring his weight is not aware of Obama doing this. You derive this because you know his pose, you understand that the field of view of a person is finite, and you understand that he is not very likely to sense the slight push of Obama's foot.
- You understand that people are self-conscious about their weight. You also understand that he is reading off the scale measurement, and that shortly the over-estimated weight will confuse him because it will probably be much higher than what he expects. In other words, you reason about implications of the events that are about to unfold seconds after this photo was taken, and especially about the thoughts and how they will develop inside people's heads. You also reason about what pieces of information are available to people.
- There are people in the back who find the person's imminent confusion funny. In other words you are reasoning about state of mind of people, and their view of the state of mind of another person. That's getting frighteningly meta.
- Finally, the fact that the perpetrator here is the president makes it maybe even a little more funnier. You understand what actions are more or less likely to be undertaken by different people based on their status and identity.

Course Objectives

CO1: Students should understand the fundamentals of a camera producing an image, including camera calibration, optical distortions, perspective corrections etc.

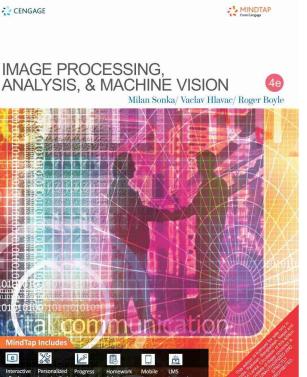
CO2: Students should be familiar with various building block algorithms in Computer Vision, including Image processing and Deep Learning with emphasis on the algorithm building blocks.

CO3: Students should create at least one end-user application.



Textbook-1:

Rick Szeliski, *Computer Vision: Algorithms and Applications* online at:
<http://szeliski.org/Book/>



Textbook-2:

Image Processing, Analysis, and Machine Vision: Milan Sonka, Vaclav Hlavac, Roger Boyle, Fourth edition, Cengage Learning

Evaluation Plan

- Evaluation

Two Quizzes for 5% each; Best one towards final grading; No Makeup; → 5%

Two Assignments – 10 % + 15% = 25%

Mid Term Exam - 30%

Comprehensive Exam - 40%

- Webinars/Tutorials

4 tutorials : 2 before mid-sem & 2 after mid-sem

- Teaching Assistant:



What will you learn in this course?

1. Computer Vision Fundamentals; (~2 Sessions)
2. Selected Topics in Low Level Vision & Mid Level Vision; (~3 Sessions)
3. Review of Deep Learning Approaches for Computer Vision (~ 1 Session)
4. Image Classification – Problem, Deep Learning Architectures, Metrics, Use cases. (~2 Sessions)
5. Classic & Modern Approaches, Applications for
 - (a) Object Detection, Recognition (~2 Sessions)
 - (b) Face Detection, Recognition (~1 Session)
 - (c) Object Tracking (~2 Sessions)
 - (d) Object Segmentation (~2 Sessions)
 - (e) OCR (~ 1 Session)
6. Visual Bag of Words & Semantic Hierarchy (~1 Session)
7. Edge Devices for Computer Vision (~1 Session)



What will you learn in this course?

1. Computer Vision Fundamentals; (~2 Sessions)
2. Selected Topics in Low Level Vision & Mid Level Vision; (~3 Sessions)
3. Review of Deep Learning Approaches for Computer Vision (~ 1 Session) [As Webinar]
4. Image Classification – Problem, Deep Learning Architectures, Metrics, Use cases. (~2 Sessions)
5. Classic & Modern Approaches, Applications for
 - (a) Object Detection, Recognition (~2 Sessions)
 - (b) Face Detection, Recognition (~1 Session) [As Webinar]
 - (c) Object Tracking (~2 Sessions)
 - (d) Object Segmentation (~2 Sessions)
 - (e) OCR (~ 1 Session) [As Webinar]
6. Visual Bag of Words & Semantic Hierarchy (~1 Session)
7. Edge Devices for Computer Vision (~1 Session) [As Webinar]



What will you learn in this course?

- On your learning
 - Reading materials given at the end of each class
 - Most of the topics needs to balance depth and breadth. We will use research articles and other online materials quite frequently.
 - Python demonstrations will be part of regular classes as required □ TA's will join the lectures sessions to support with demonstration.
 - Wherever necessary, stress will be given to related classic topics in Computer Vision.
 - Assignment problems [to be solved in a group of maximum of 4 members] will be given on natural images and remote sensing images & you can make a choice.



- Plagiarism Policy

All submissions for graded components must be the result of your original effort. It is strictly prohibited to copy and paste verbatim from any sources, whether online or from your peers. The use of unauthorized sources or materials, as well as collusion or unauthorized collaboration to gain an unfair advantage, is also strictly prohibited.

Please note that we will not distinguish between the person sharing their resources and the one receiving them for plagiarism, and the consequences will apply to both parties equally.

In cases where suspicious circumstances arise, such as identical verbatim answers or a significant overlap of unreasonable similarities in a set of submissions, will be investigated, and severe punishments will be imposed on all those found guilty of plagiarism.



- Schedule of Quizzes

Sunday, January 14, 2024	7:00:00 PM	Monday, January 15, 2024	7:00:00 PM
Sunday, March 10, 2024	7:00:00 PM	Monday, March 11, 2024	7:00:00 PM

- Schedule of Assignments

Assignment - #1: 02 Jan, 2024	7:00 PM	18 Jan, 2024	11:59 PM
Assignment - #2: 01, Mar 2024	7:00 PM	15, Mar 2024	11:59 PM

- Schedule of Webinars

26-Dec-23
16-Jan-24
27-Feb-24
19-Mar-24



- **Announcements on Lecture Class -03 Rescheduling:**

Class 03, originally scheduled to be held on 9th of December is rescheduled on 8th of December, Friday, Night, from 7:30 PM to 09:30 PM.

Topic Planned for Class -03:

Histogram and Histogram equalization

Gray-scale transformation

Image Smoothing

Connected components in images

Use case: Sharpening, blur, and noise removal using Filtering



- **Readings:**

Reach TB-1, Sections 1.1 and 1.2

- **Topics for Next Class**

Image representation and image analysis tasks (T2 Ch 1.3)

Image digitization - Sampling and resolution (T2 Ch 2.2)

Digital Images (T2 Ch 2.3)

Digital Image types -Binary, Gray-scale and Color (Class Notes)

Color Images (T2 Ch 2.4)

Color spaces: RGB and HSV (T2 Ch 2.4)



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision
2023-24 First Semester, M.Tech (AIML)

Session #2: Digital Image Fundamentals

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

S. P. Vimal | CSIS Department | WILP | BITS – Pilani (vimalsp@wilp.bits-pilani.ac.in)

Topics

- Image Representation
- Image Digitization (Sampling & Quantization)
- Digital Image Properties
- OpenCV/Python Demonstration

Image representation and image analysis tasks, Image digitization - Sampling and resolution, Digital Images, Digital Image types -Binary, Gray-scale and Color, Color Images, Color spaces: RGB and HSV

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

Simple Image Formation Model

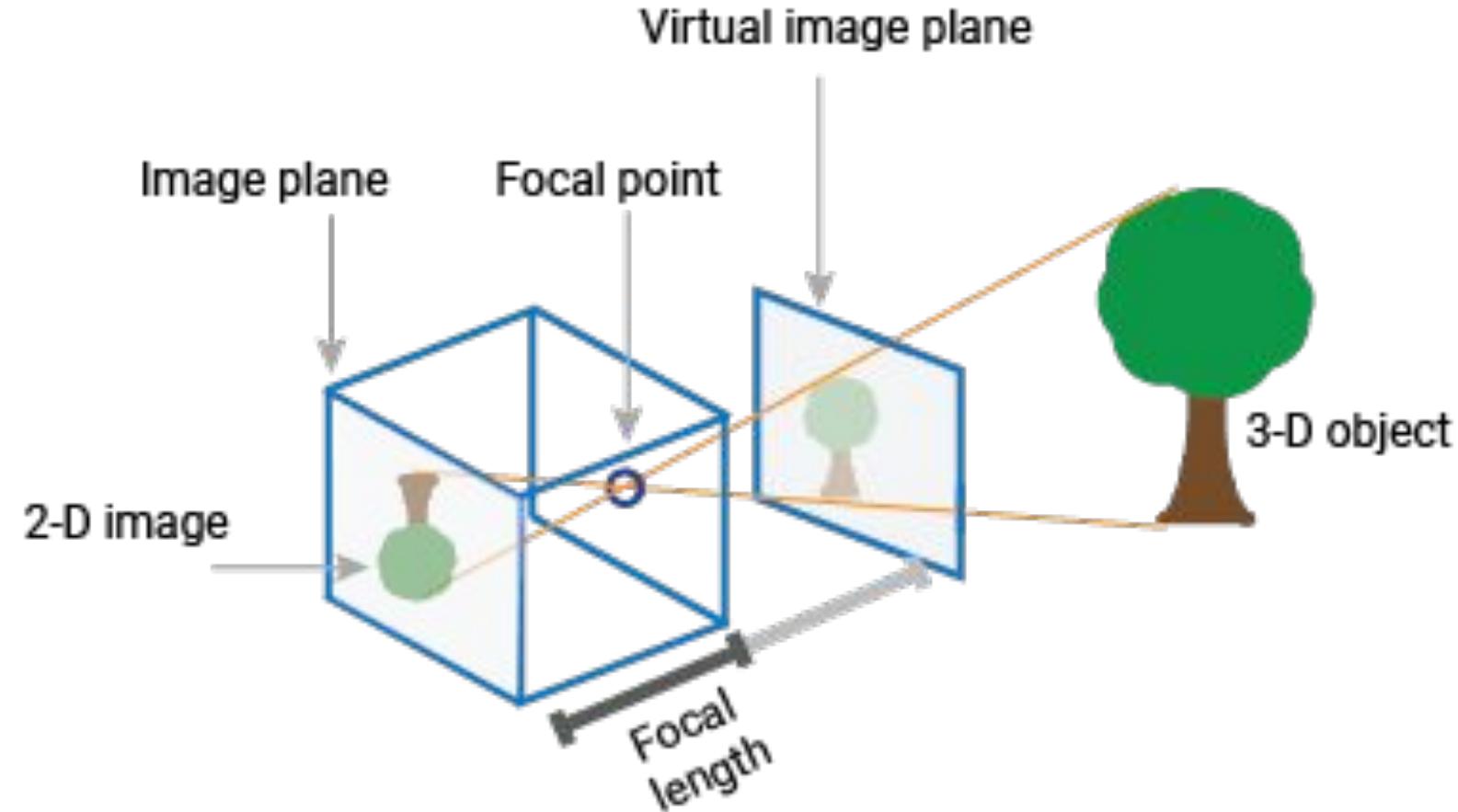
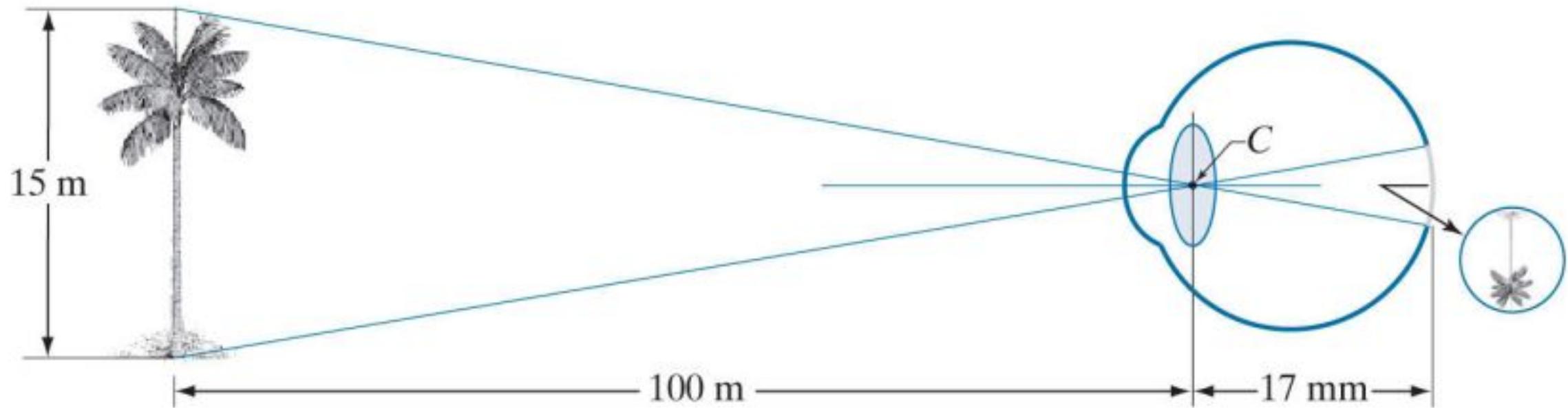
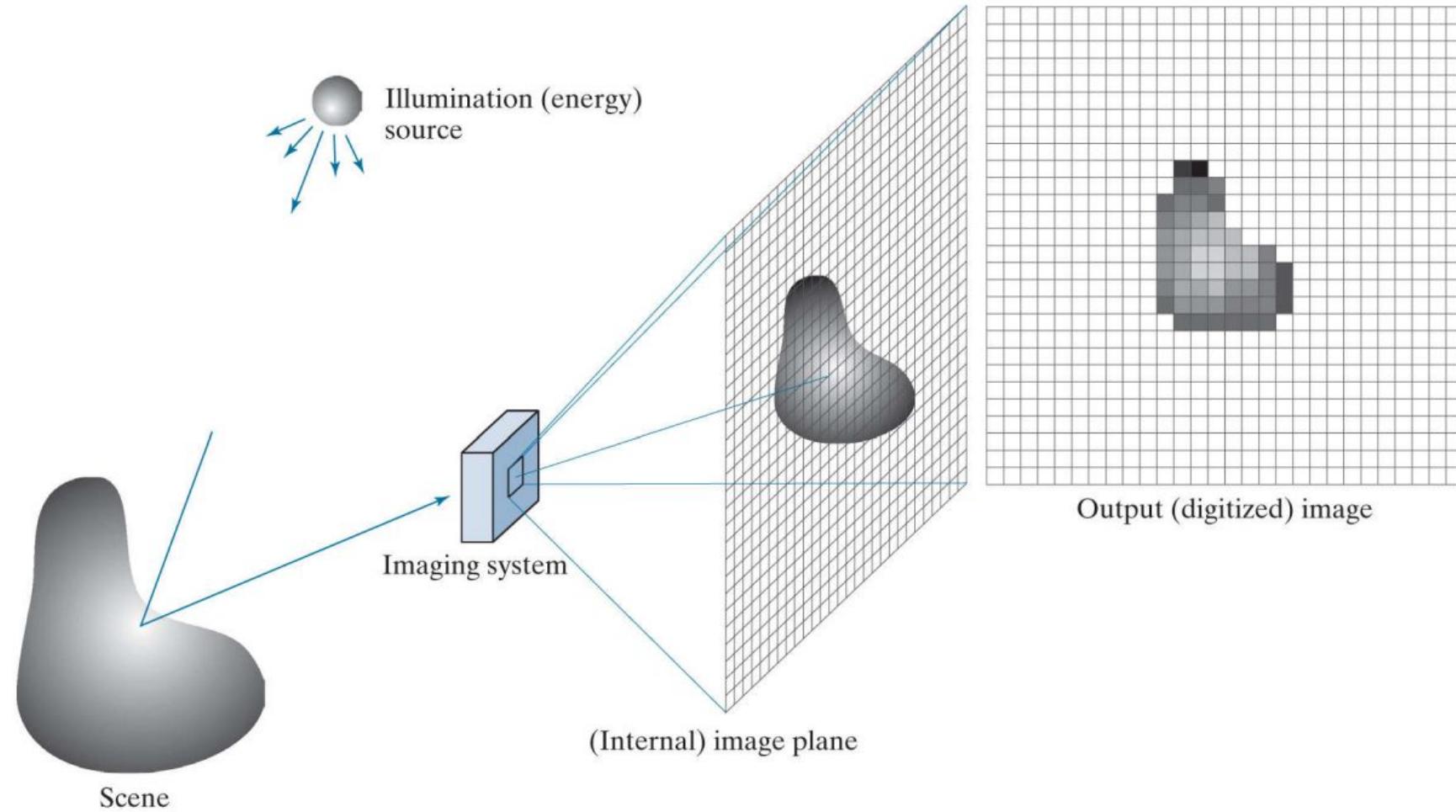


Image Formation in the Eye



Digital Image Acquisition





Simple Image Formation Model

- We denote images by two-dimensional functions of the form $f(x, y)$.
 - Value of f at spatial coordinates (x, y) is a scalar quantity;
 - Physical meaning of $f(x, y)$ is determined by the source of the image, and whose values are proportional to energy radiated by a physical source (e.g., electromagnetic waves);

$$0 \leq f(x, y) < \infty$$

- Function $f(x, y)$ is characterized by two components:
 - Amount of source illumination incident on the scene being viewed, and
 - Amount of illumination reflected by the objects in the scene.

$$f(x, y) = i(x, y)r(x, y)$$

$$0 \leq i(x, y) < \infty$$

$$0 \leq r(x, y) \leq 1$$



Simple Image Formation Model - Grayscale Image

- Let the intensity (gray level) of a monochrome image at any coordinates (x, y) be denoted by

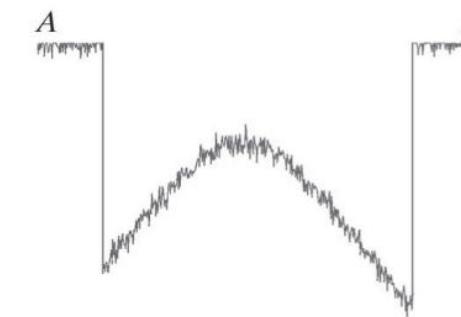
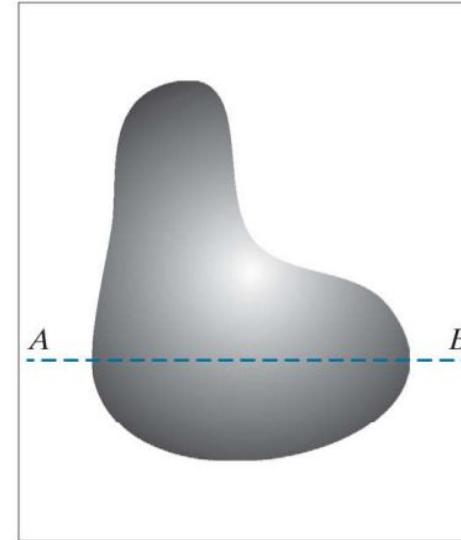
$$\ell = f(x, y) \quad L_{\min} \leq \ell \leq L_{\max}$$

$$L_{\min} = i_{\min} r_{\min} \quad L_{\max} = i_{\max} r_{\max}$$

- For a gray scale, the range is scaled between $[0, L-1]$;
 - All intermediate values from 0 to $L-1$ are shades of gray varying from black to white.

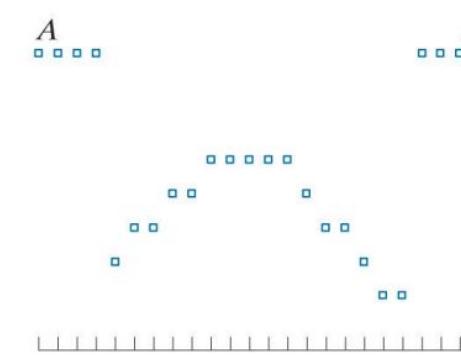
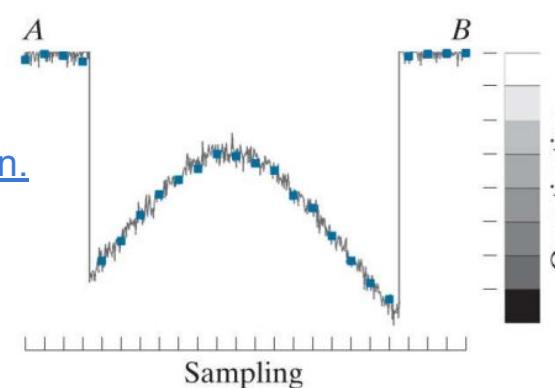
Sampling and Quantization

Continuous image



A scan line showing intensity variations along line AB in the continuous image

Sampling and quantization.

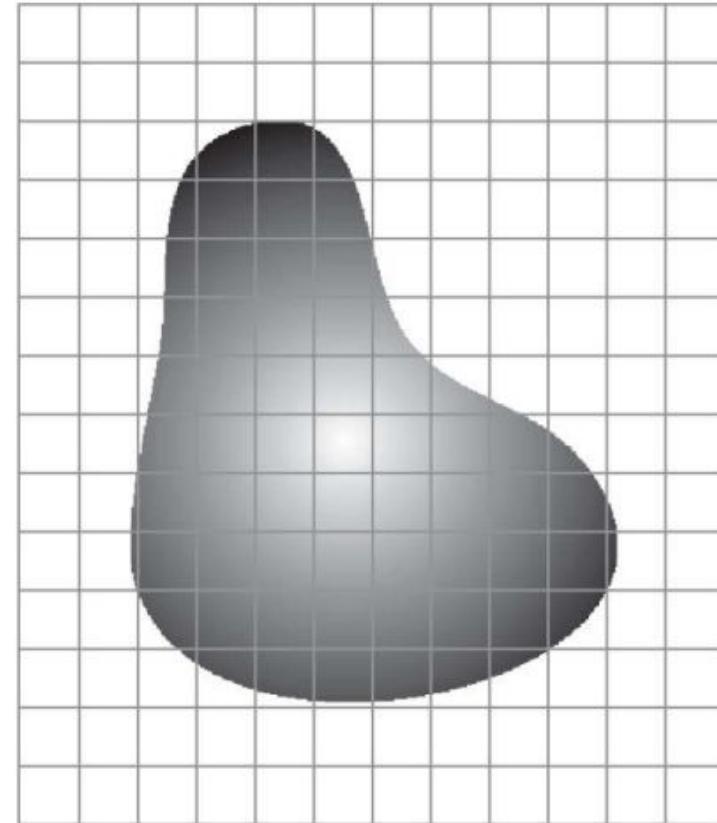


Digital scan line

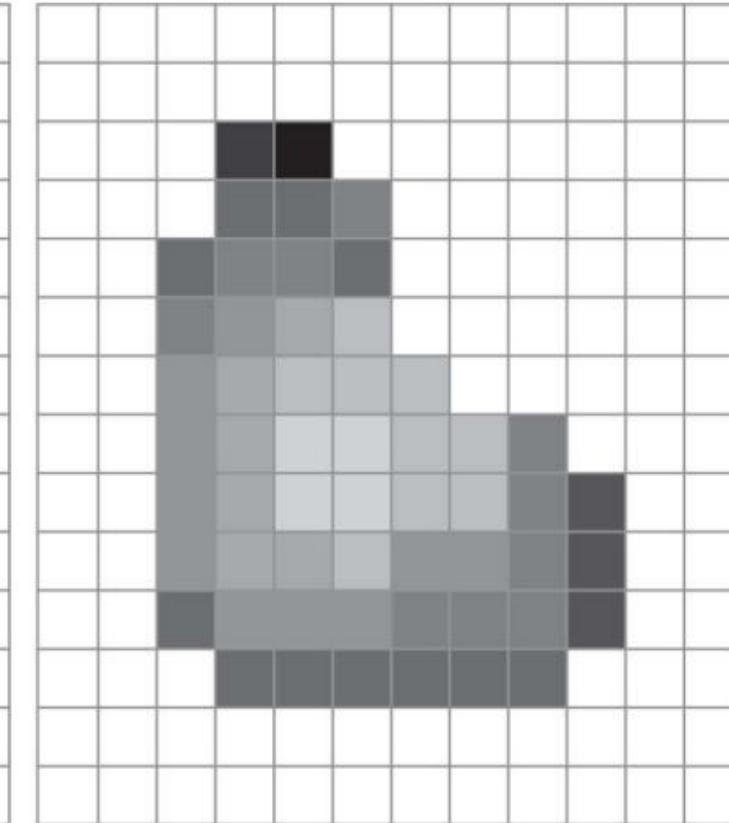
Digitizing the coordinate values is called **sampling** ; Digitizing the amplitude values is called **quantization**.

Sampling and Quantization

Continuous image projected onto a sensor array



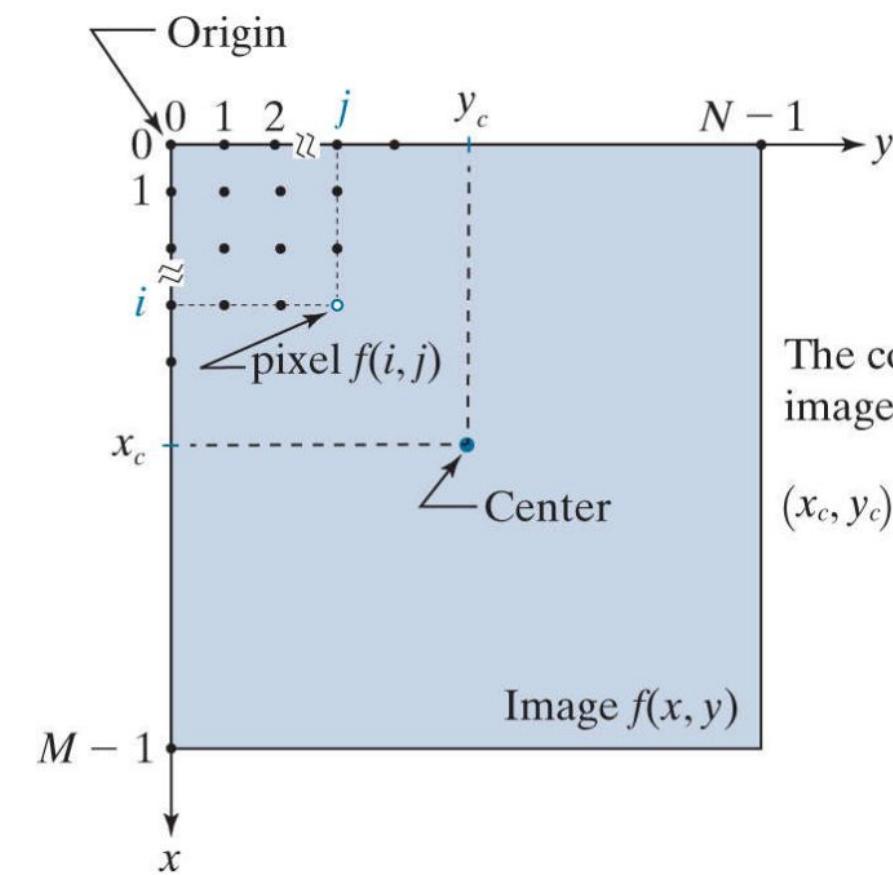
Result of image sampling and quantization



Matrix Representation of an Image

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

•



The coordinates of the image center are

$$(x_c, y_c) = \left(\text{floor}\left(\frac{M}{2}\right), \text{floor}\left(\frac{N}{2}\right) \right)$$

Spatial & Intensity Resolution

- **Spatial resolution** is a measure of the smallest discernible detail in an image;
- **Intensity resolution** similarly refers to the smallest discernible change in intensity level

930 dpi



300 dpi



150 dpi



72 dpi



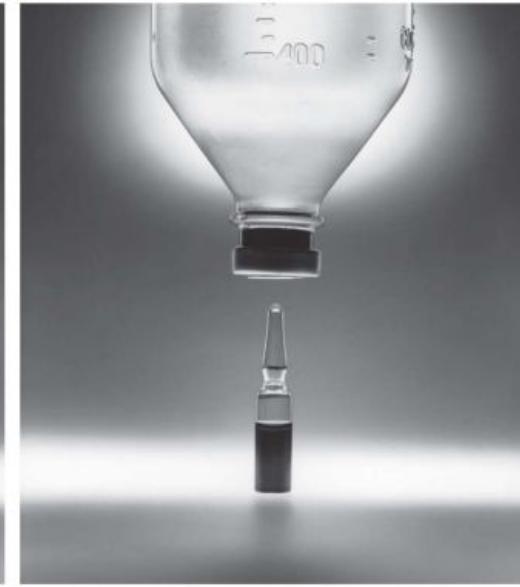
Spatial & Intensity Resolution

- **Spatial resolution** is a measure of the smallest discernible detail in an image;
- **Intensity resolution** similarly refers to the smallest discernible change in intensity level

256-level



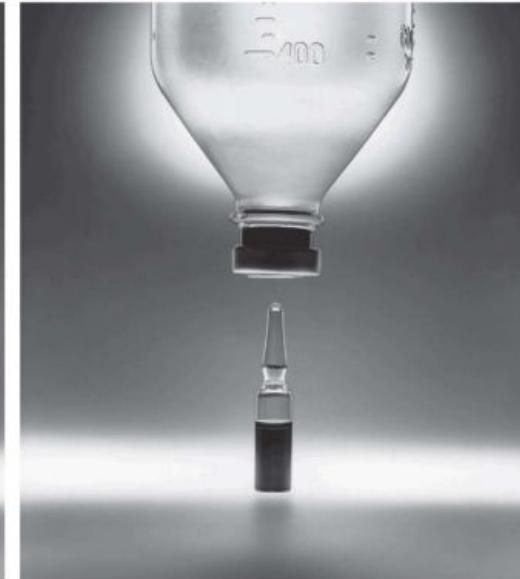
128-level



64-level



32-level



All images have spatial resolution 2022 × 1800

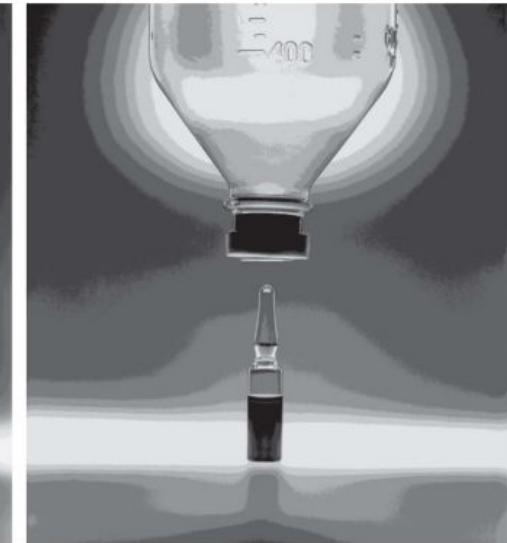
Spatial & Intensity Resolution

- **Spatial resolution** is a measure of the smallest discernible detail in an image;
- **Intensity resolution** similarly refers to the smallest discernible change in intensity level

16-level



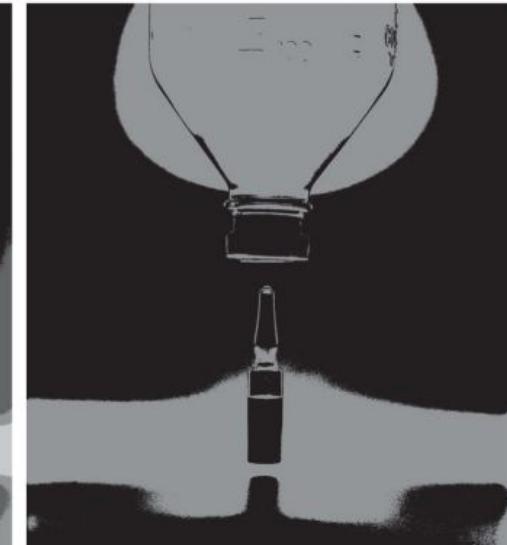
8-level



4-level



2-level



All images have spatial resolution 2022 × 1800



- **Readings:**

Digital Image Processing, Rafael C. Gonzalez & Richard E woods, Third Ed, Chapter 2



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision

2023-24 First Semester, M.Tech (AIML)

Session #3: Low Level Vision

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

S. P. Vimal | CSIS Department | WILP | BITS – Pilani (vimalsp@wilp.bits-pilani.ac.in)

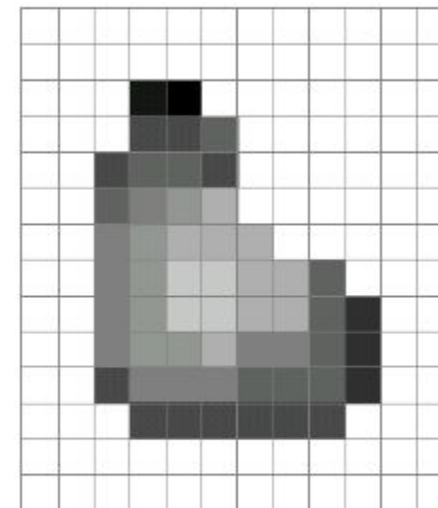
Topics

- Image Histogram & Histogram Equalization
- Basic Intensity Transformations
- Image Smoothing

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

What is an image?

A grid (matrix) of intensity values



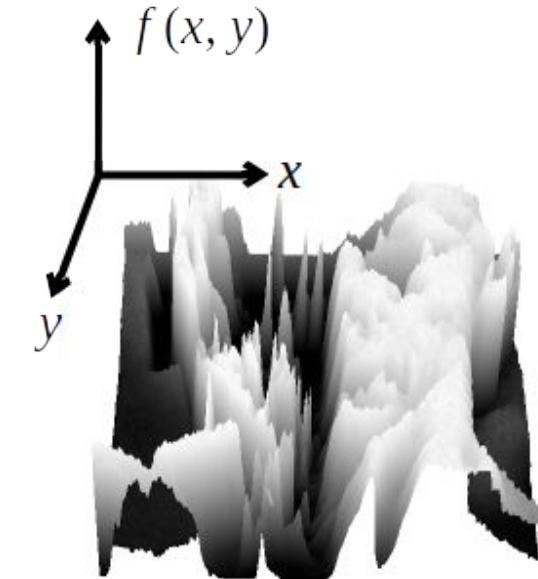
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255	255	255	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255	255	255	255	255	255
255	255	127	145	145	175	127	127	95	47	255	255	255	255	255	255	255
255	255	74	127	127	127	95	95	95	47	255	255	255	255	255	255	255
255	255	255	74	74	74	74	74	74	74	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

What is an image?

Can think of a (grayscale) image as a function f from \mathbb{R}^2 to \mathbb{R} :
 $f(x, y)$ gives the intensity at position (x, y)



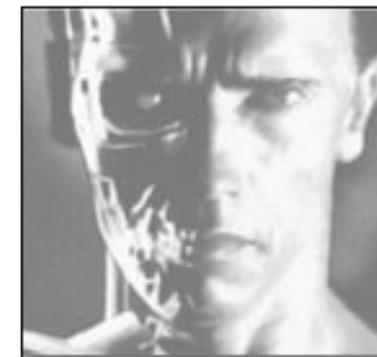
snoop



3D view

Image Transformations

As with any function, we can apply operators to an image



$$g(x,y) = f(x,y) + 20$$



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



Image Histogram

Un-normalized histogram: $h(r_k) = n_k \quad \text{for } k = 0, 1, 2, \dots, L - 1$

Note: n_k is the number of pixels in f with intensity r_k

Normalized histogram: $p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN}$

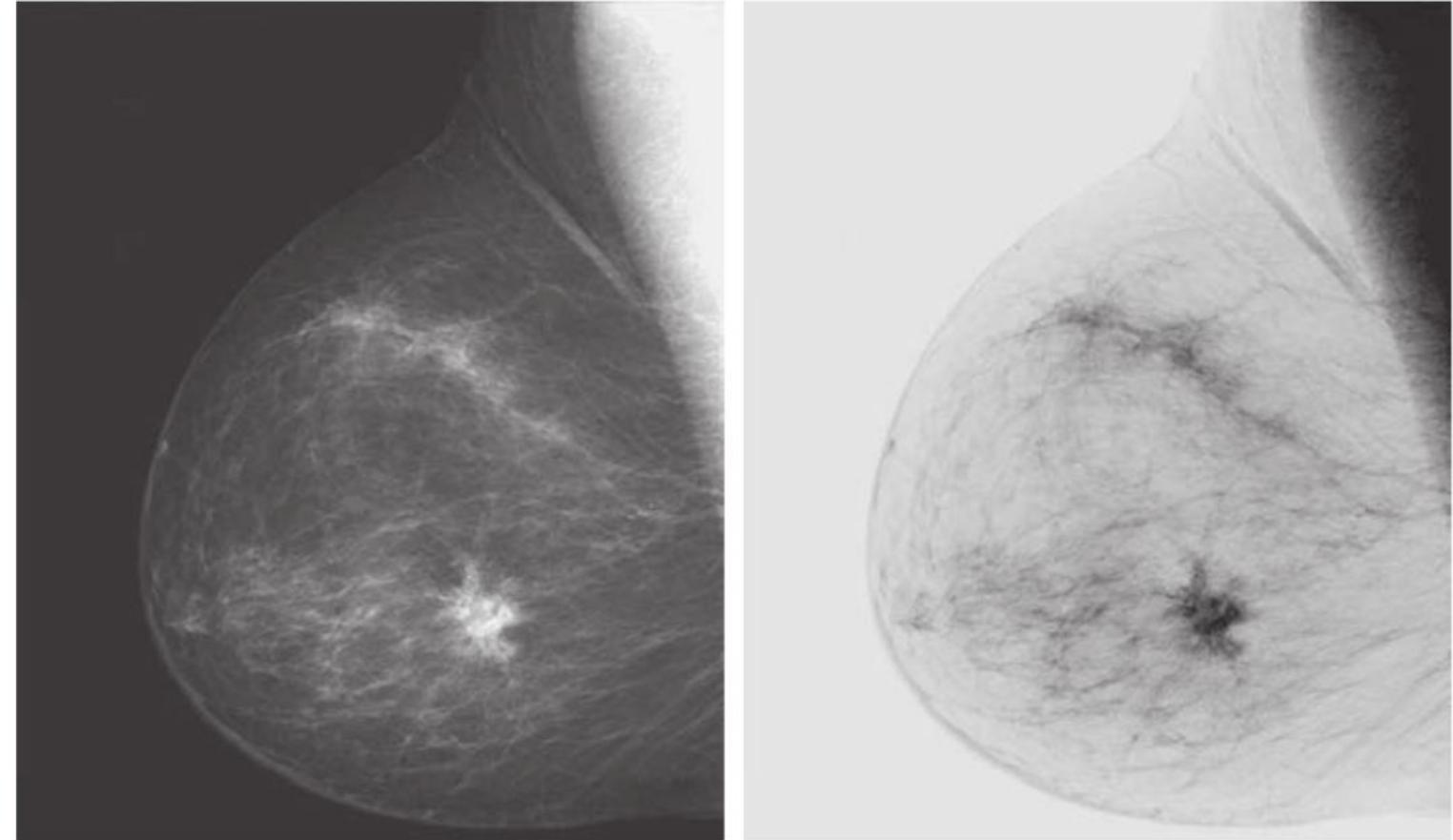
Note: M and N are the number of image rows and columns

The sum of $p(r_k)$ for all values of k is always 1

Mostly, we work with normalized histograms

Elementary Intensity Transformations

$$s = L - 1 - r$$



A digital mammogram and its Negative image

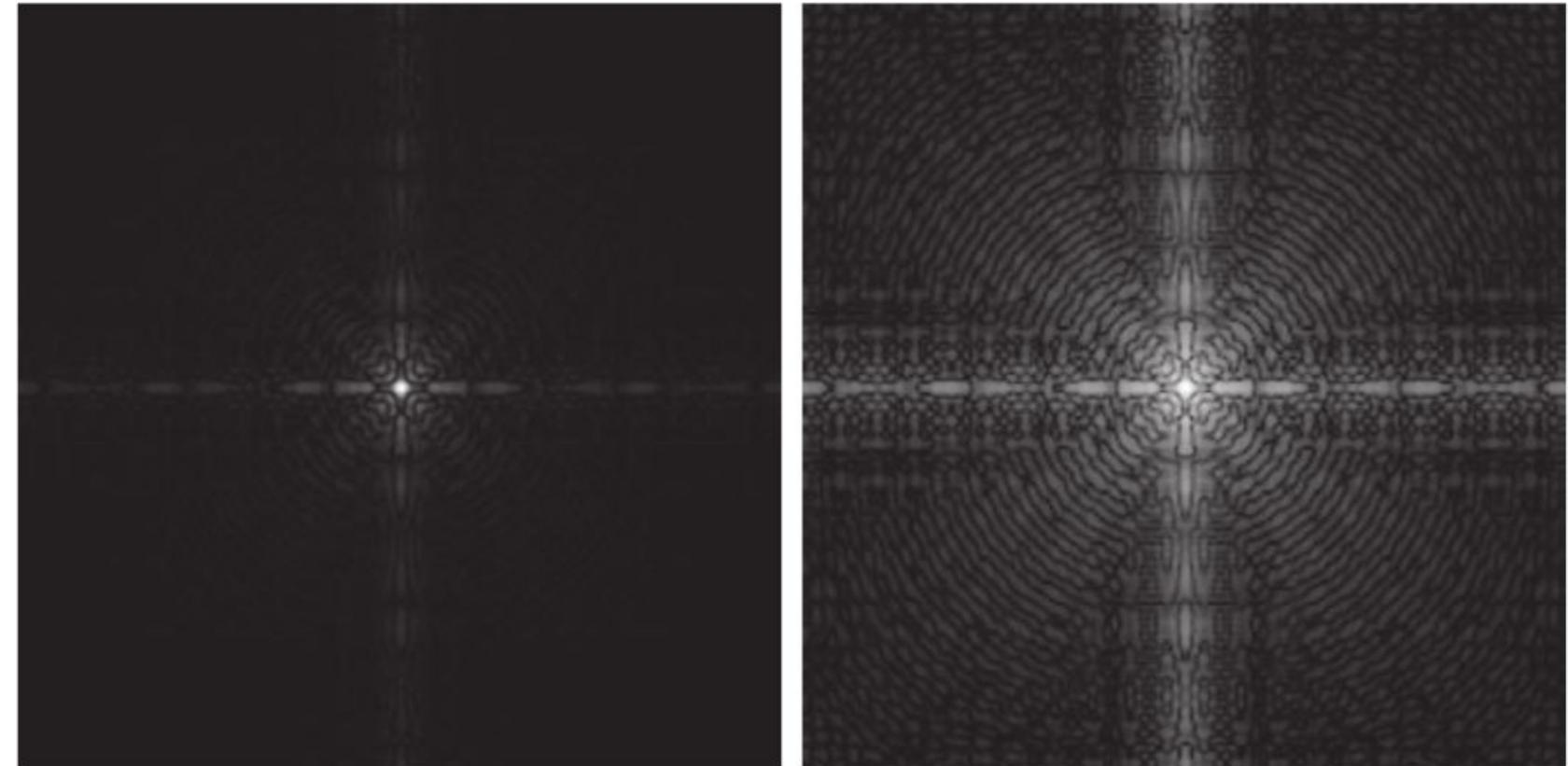
Elementary Intensity Transformations

Log Transformations

$$s = c \log(1 + r)$$

Fourier spectrum displayed as a grayscale image and the result of applying the log transformation in with $c = 1$.

Both images are scaled to the range $[0, 255]$.



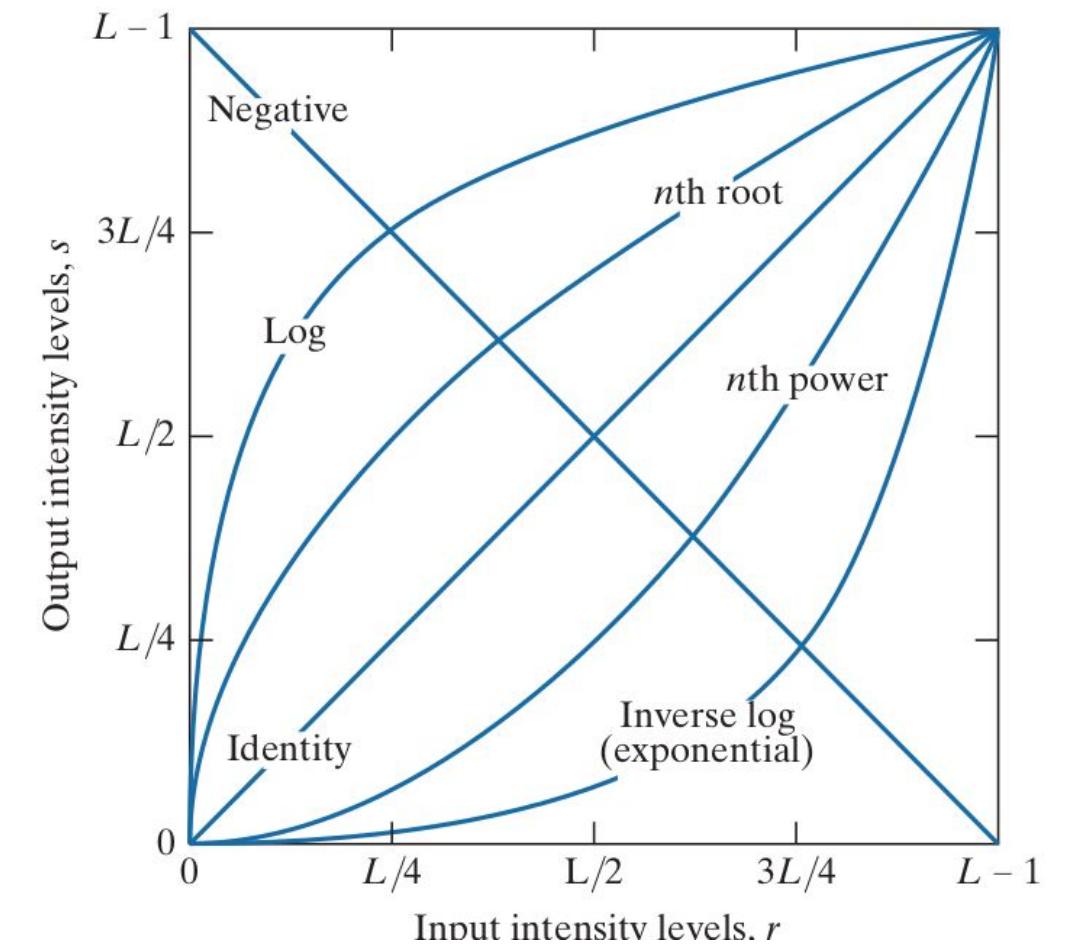
Elementary Intensity Transformations

Log Transformations

$$s = c \log(1 + r)$$

Fourier spectrum displayed as a grayscale image and the result of applying the log transformation in with $c = 1$.

Both images are scaled to the range $[0, 255]$.

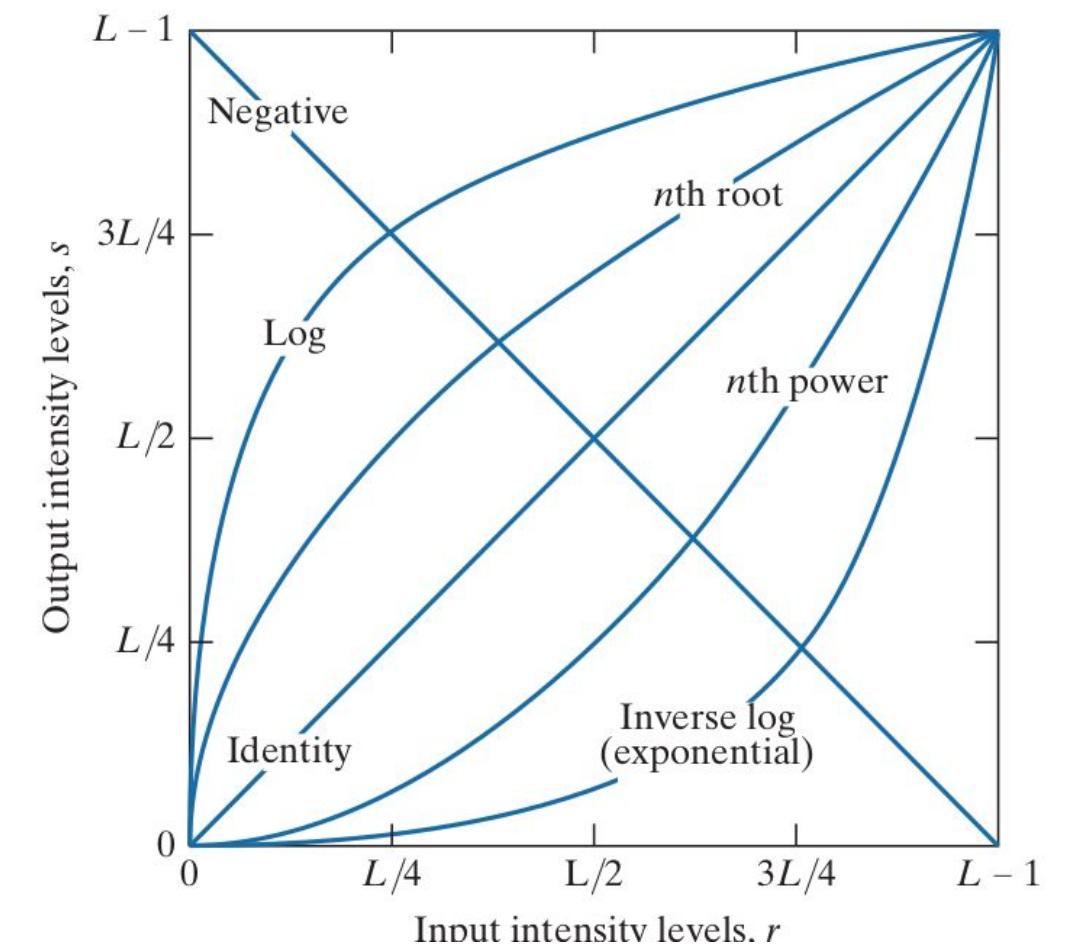


Elementary Intensity Transformations

Power-Law Transformations

$$s = cr^\gamma$$

- c and γ are positive constants

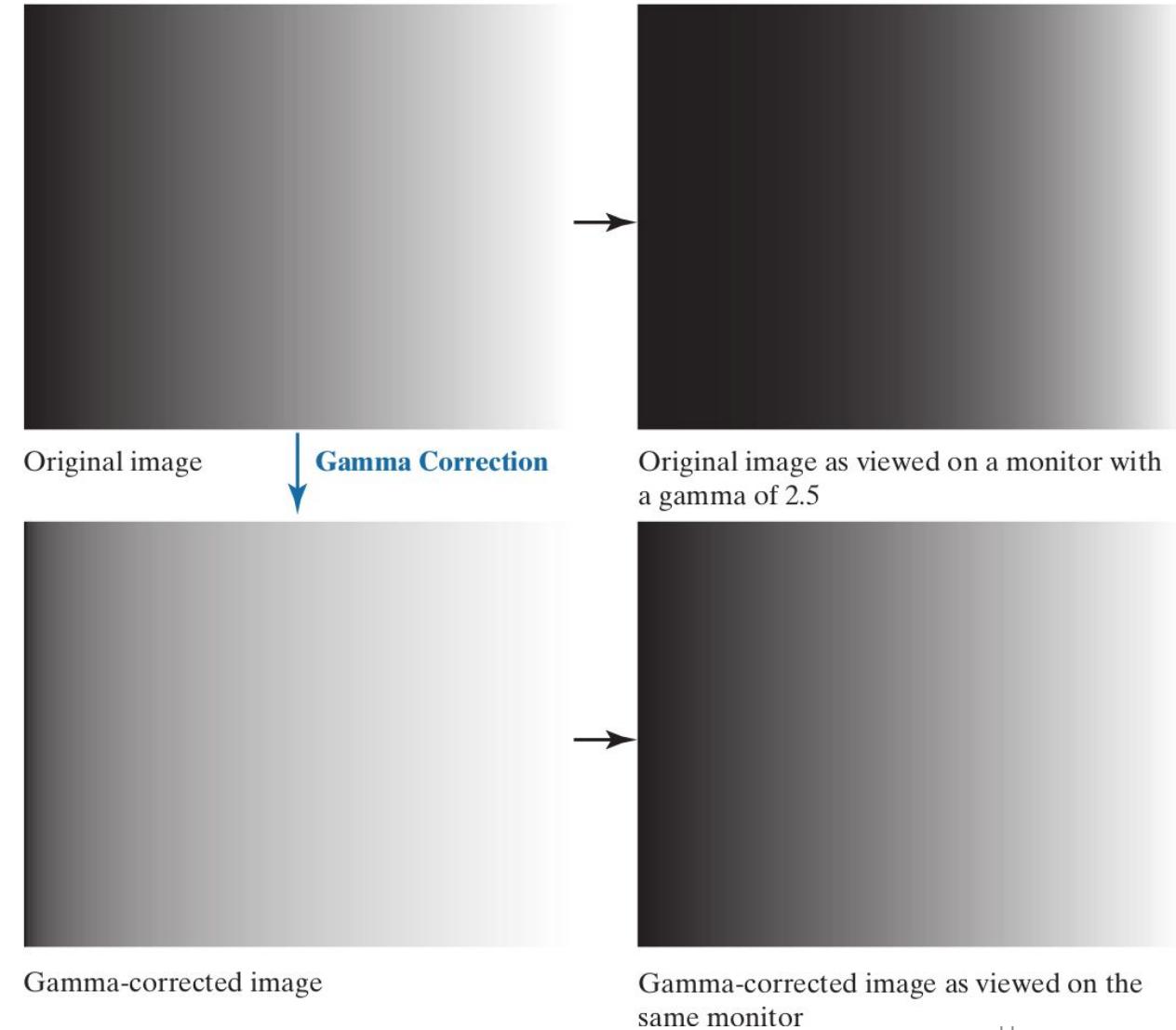


Elementary Intensity Transformations

Power-Law Transformations- Applications

Gamma correction

- Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5.



$$s = cr^\gamma$$

Elementary Intensity Transformations

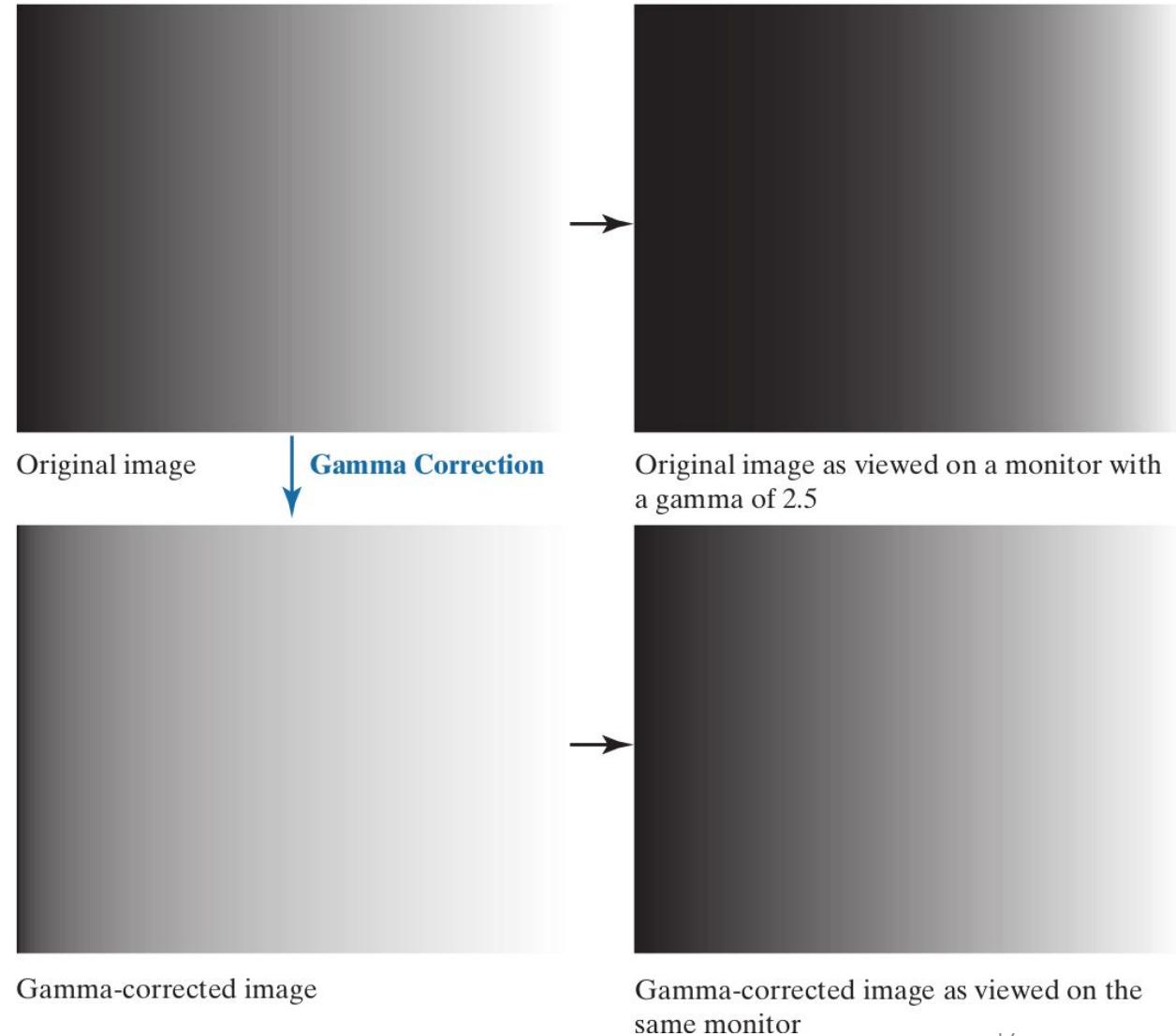
Power-Law Transformations- Applications

Gamma correction

- Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5.
- Gamma correction consists of using the transformation

$$s = r^{1/2.5} = r^{0.4}$$

$$s = cr^\gamma$$



Elementary Intensity Transformations

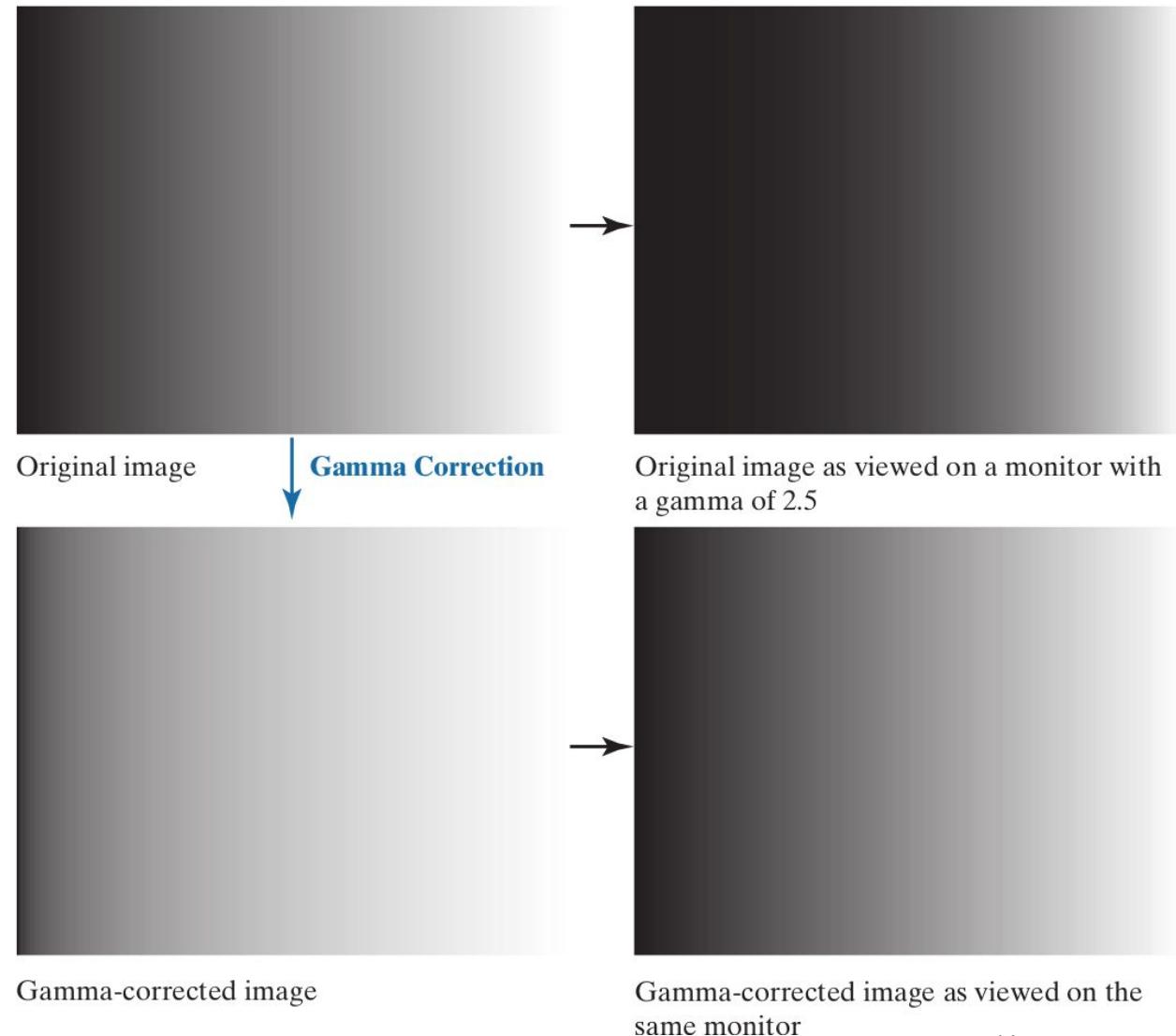
Power-Law Transformations- Applications

Gamma correction

- Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5.
- Gamma correction consists of using the transformation

$$s = r^{1/2.5} = r^{0.4}$$

$$s = cr^\gamma$$



Elementary Intensity Transformations

Power-Law Transformations- Applications

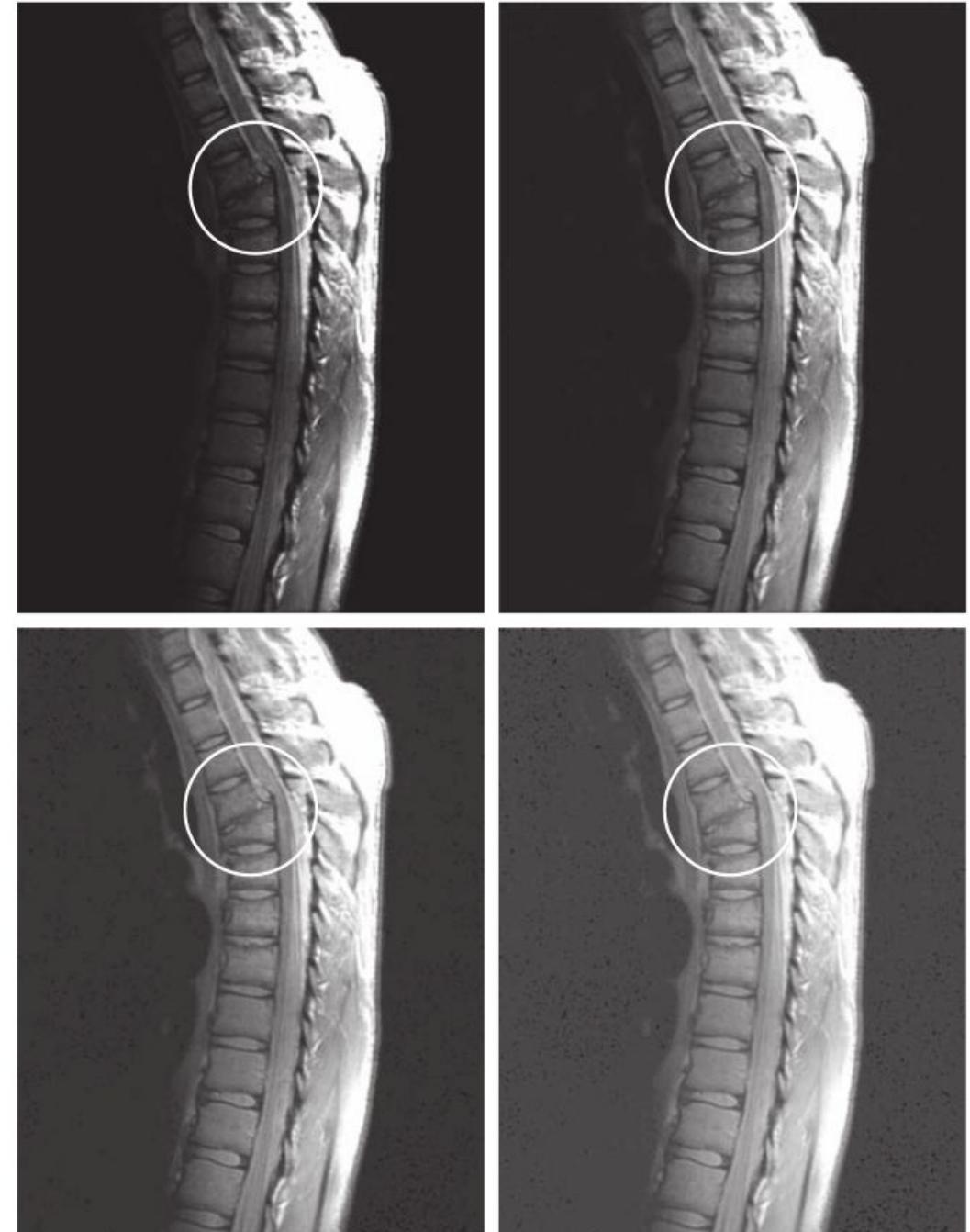
Medical Imaging

(a) Magnetic resonance image (MRI) of a fractured human spine (the region of the fracture is enclosed by the circle).

(b)–(d) Results of applying the transformation with $c = 1$ and $g = 0.6, 0.4$, and 0.3 , respectively.
(Original image courtesy of Dr. David R. Pickens,
Department of Radiology and Radiological
Sciences, Vanderbilt University Medical Center.)

$$s = cr^\gamma$$

a	b
c	d



Elementary Intensity Transformations

Power-Law Transformations- Applications

Aerial Imaging

(a) Aerial image.

(b)–(d) Results of applying the transformation in with $\gamma = 3.0, 4.0$, and 5.0 , respectively. ($c = 1$ in all cases.)

(Original image, courtesy of NASA.)

$$s = cr^\gamma$$

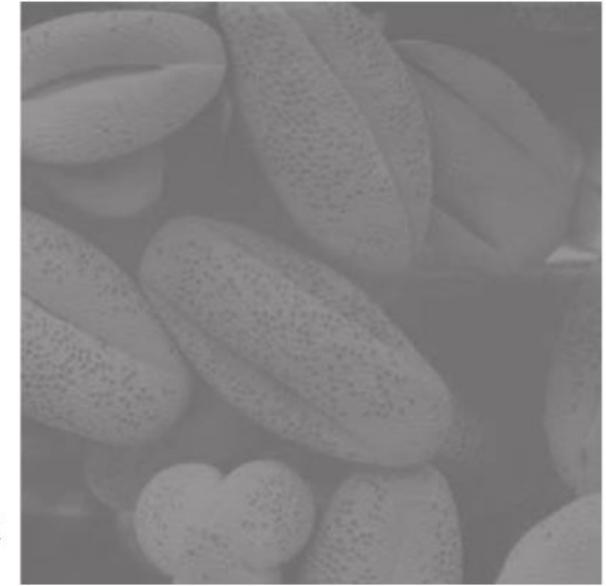
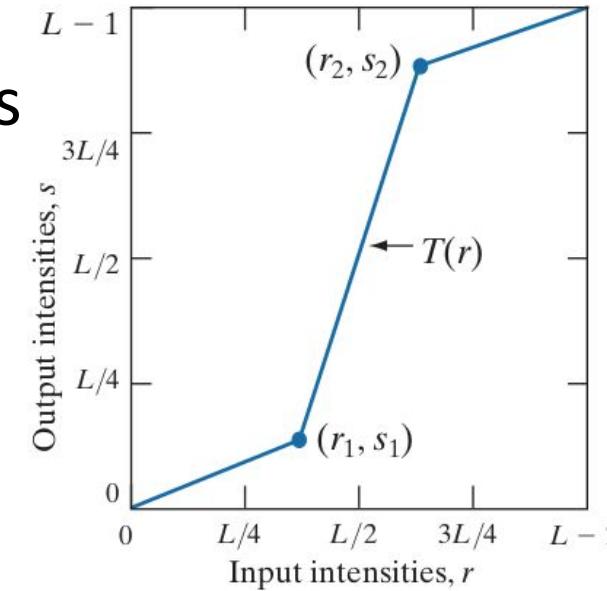


a	b
c	d

Elementary Intensity Transformations

Contrast Stretching; Thresholding

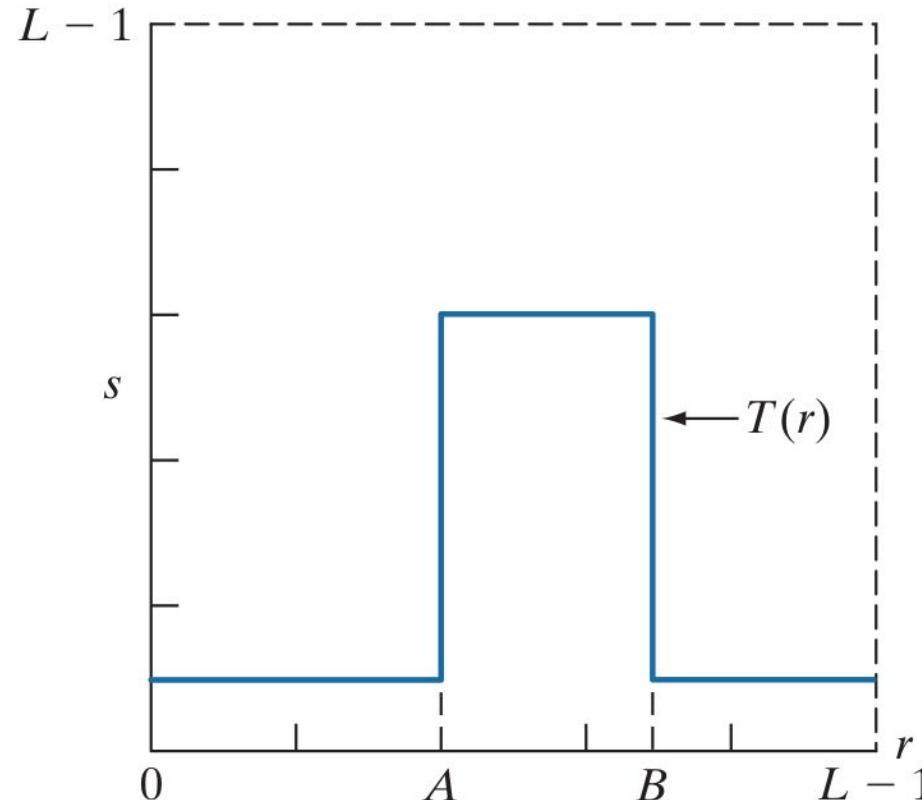
- (a) Piecewise linear transformation function.
- (b) A low-contrast electron microscope image of pollen, magnified 700 times.
- (c) Result of contrast stretching.
- (d) Result of thresholding.



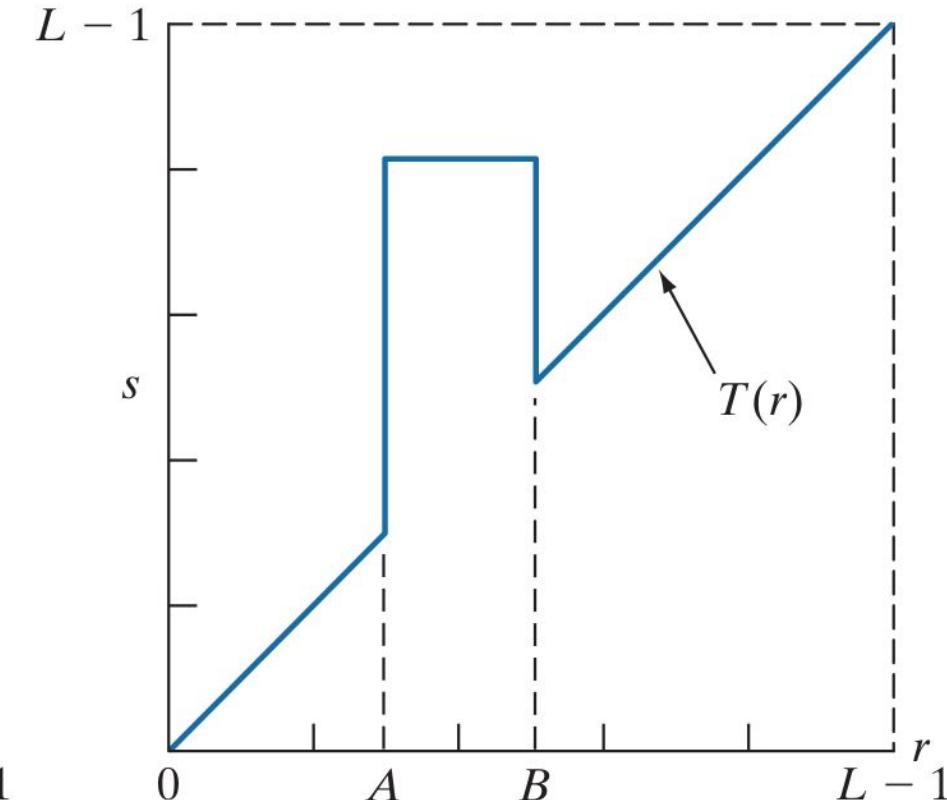
a b
c d

Elementary Intensity Transformations

Intensity Level Slicing

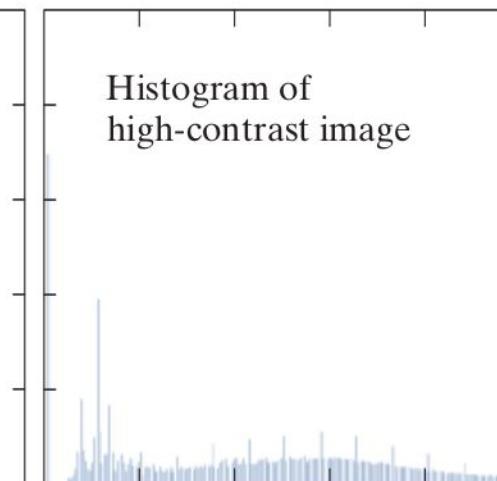
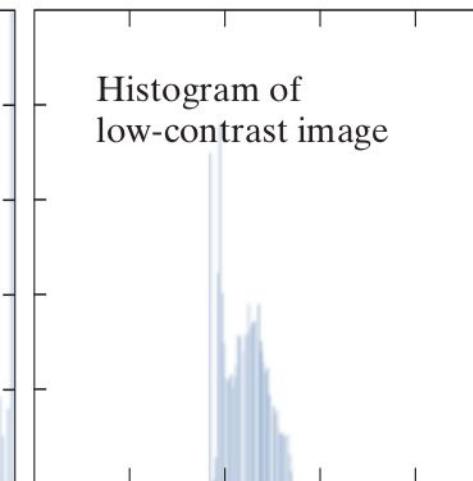
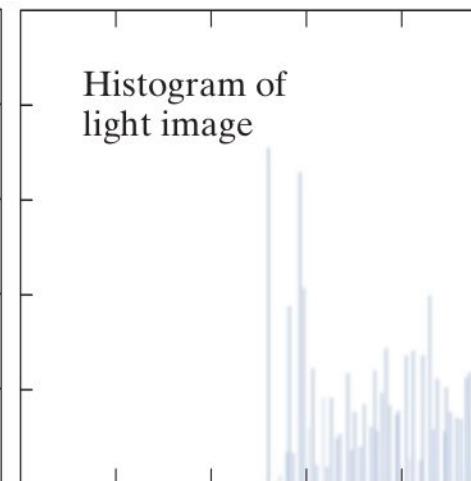
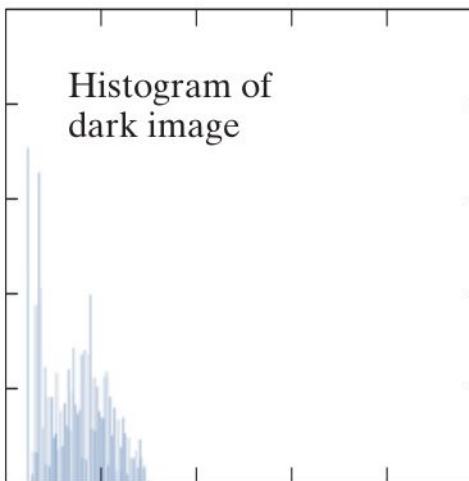


Highlights range [A , B] and reduces all other intensities to a lower level.



This function highlights range [A , B] and leaves other intensities unchanged.

Image Histogram



Histogram shape is related to image appearance ;

Histogram Equalization

Histogram :

$$p(r) = \left(\frac{\text{Number of pixels with intensity } r}{\text{Total number of pixels}} \right)$$

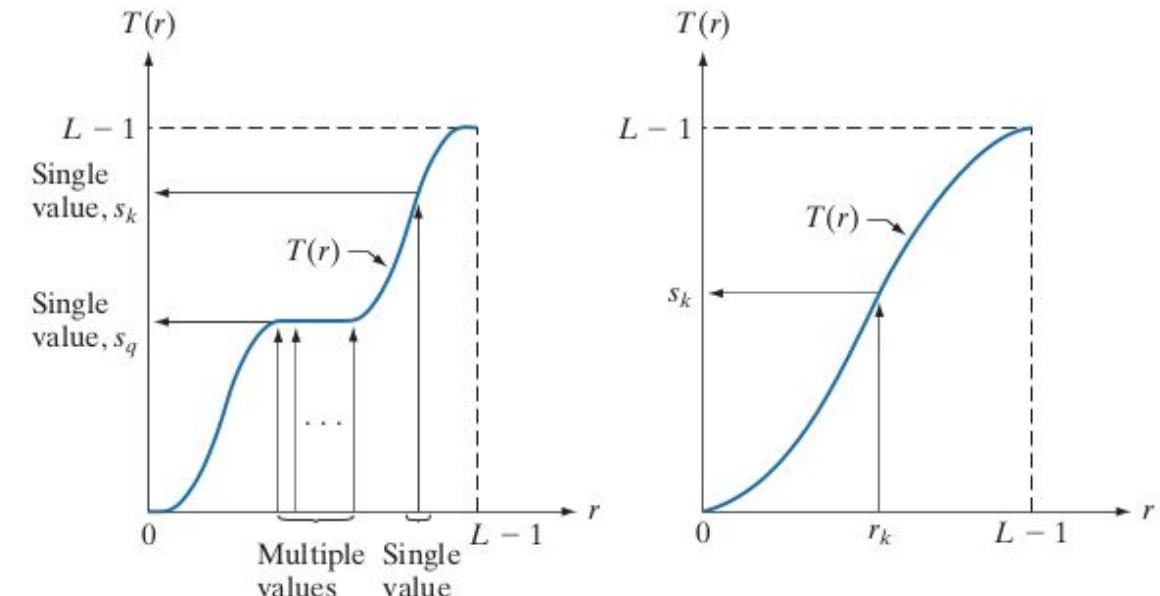
To do :

Intensity mapping

$$s = T(r)$$

Assume

- $T(r)$ is single-valued and monotonically increasing.
- $0 \leq T(r) \leq 1$ and $0 \leq r \leq 1$



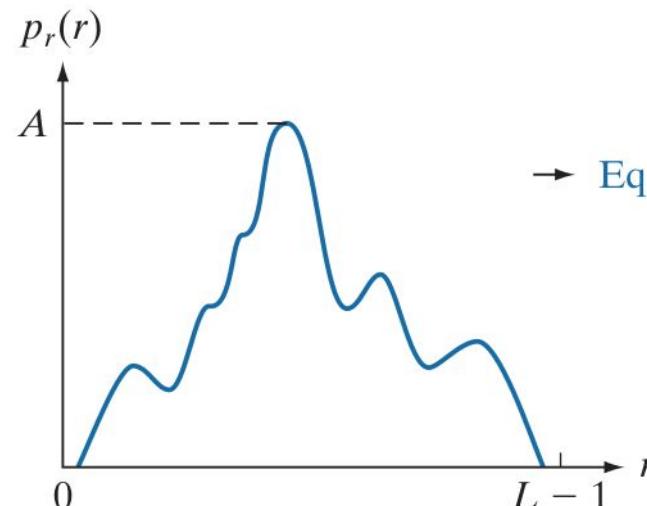
[†]A function $T(r)$ is a *monotonic increasing* function if $T(r_2) \geq T(r_1)$ for $r_2 > r_1$. $T(r)$ is a *strictly monotonic increasing* function if $T(r_2) > T(r_1)$ for $r_2 > r_1$. Similar definitions apply to a monotonic decreasing function.

Histogram Equalization

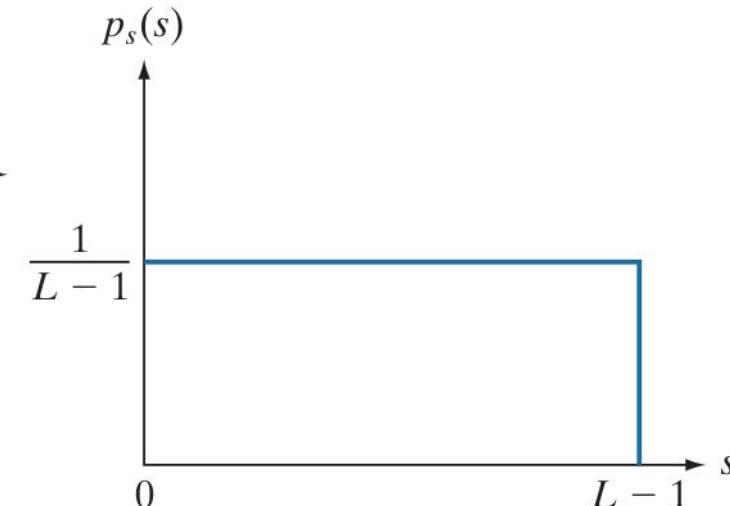
Histogram :

$$p(r) = \left(\frac{\text{Number of pixels with intensity } r}{\text{Total number of pixels}} \right)$$

To do :



→ Eq. (3-11) →



[†] A function $T(r)$ is a *monotonic increasing* function if $T(r_2) \geq T(r_1)$ for $r_2 > r_1$. $T(r)$ is a *strictly monotonic increasing* function if $T(r_2) > T(r_1)$ for $r_2 > r_1$. Similar definitions apply to a monotonic decreasing function.



Histogram Equalization

$$s = T(r) \quad 0 \leq r \leq L - 1$$

- a. $T(r)$ is a strictly monotonically increasing function in the interval $0 \leq r \leq L - 1$;
- b. $0 \leq T(r) \leq L - 1$ for $0 \leq r \leq L - 1$.

If $p_r(r)$ and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed (mapped) variable s can be obtained as

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|$$



Histogram Equalization

If $p_r(r)$ and $T(r)$ are known, and

$T(r)$ is continuous and differentiable over the range of values of interest,
then the PDF of the transformed (mapped) variable s can be obtained as

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|$$

A transformation function of particular importance in image processing is

$$s = T(r) = (L - 1) \int_0^r p_r(w) dw$$

Histogram Equalization

$$p_s(s) = p_r(r) \left| \frac{dr}{ds} \right|$$

trying to compute

$$\frac{ds}{dr} = \frac{dT(r)}{dr}$$

$$= (L-1) \frac{d}{dr} \left[\int_0^r p_r(w) dw \right]$$

$$= (L-1)p_r(r) \quad \text{Leibniz's rule in calculus}$$

$$s = T(r) = (L-1) \int_0^r p_r(w) dw$$

$$\begin{aligned} p_s(s) &= p_r(r) \left| \frac{dr}{ds} \right| \\ &= p_r(r) \left| \frac{1}{(L-1)p_r(r)} \right| \\ &= \frac{1}{L-1} \quad 0 \leq s \leq L-1 \end{aligned}$$

Form of uniform probability density function



Histogram Equalization

T(r) for continuous values:

$$s = T(r) = (L - 1) \int_0^r p_r(w) dw$$

Digital image is discrete:

$$p_r(r_k) = \frac{n_k}{MN}$$

$$s_k = T(r_k) = (L - 1) \sum_{j=0}^k p_r(r_j) \quad k = 0, 1, 2, \dots, L - 1$$

Need to just use this to transform image



Histogram Equalization - Example

Suppose that a 3-bit image ($L=8$) of size 64×64 pixels ($MN = 4096$) has the intensity distribution shown in following table.

r_k	n_k	$p_r(r_k) = n_k / MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Histogram Equalization - Example

r_k	n_k	$p_r(r_k) = n_k / MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

$$s_0 = T(r_0) = 7 \sum_{j=0}^0 p_r(r_j) = 7 \times 0.19 = 1.33 \rightarrow 1$$

$$s_1 = T(r_1) = 7 \sum_{j=0}^1 p_r(r_j) = 7 \times (0.19 + 0.25) = 3.08 \rightarrow 3$$

$$s_2 = 4.55 \rightarrow 5 \qquad s_3 = 5.67 \rightarrow 6$$

$$s_4 = 6.23 \rightarrow 6 \qquad s_5 = 6.65 \rightarrow 7$$

$$s_6 = 6.86 \rightarrow 7 \qquad s_7 = 7.00 \rightarrow 7$$

Histogram Equalization - Example

$$s_0 = 1.33 \rightarrow 1$$

$$s_2 = 4.55 \rightarrow 5$$

$$s_4 = 6.23 \rightarrow 6$$

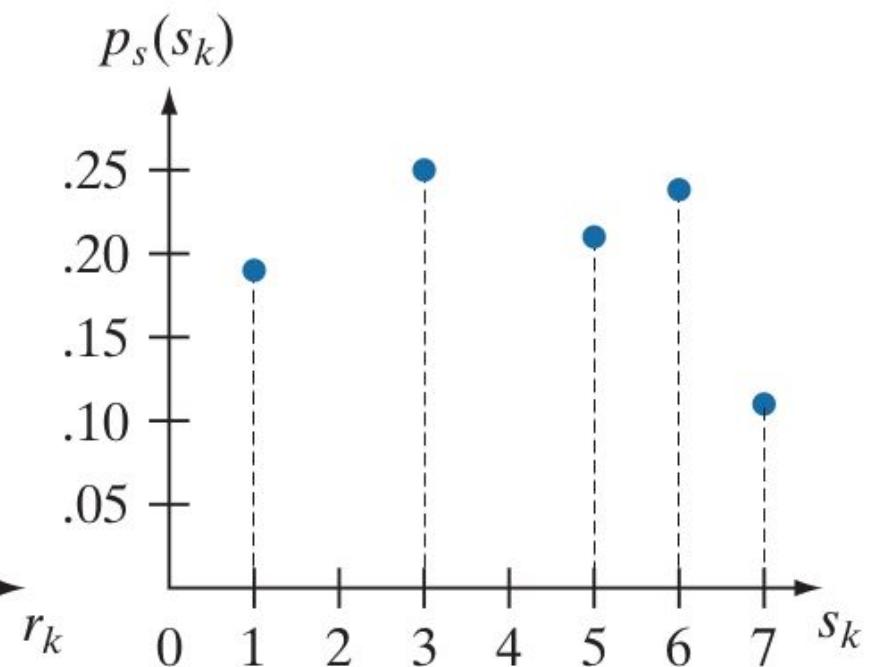
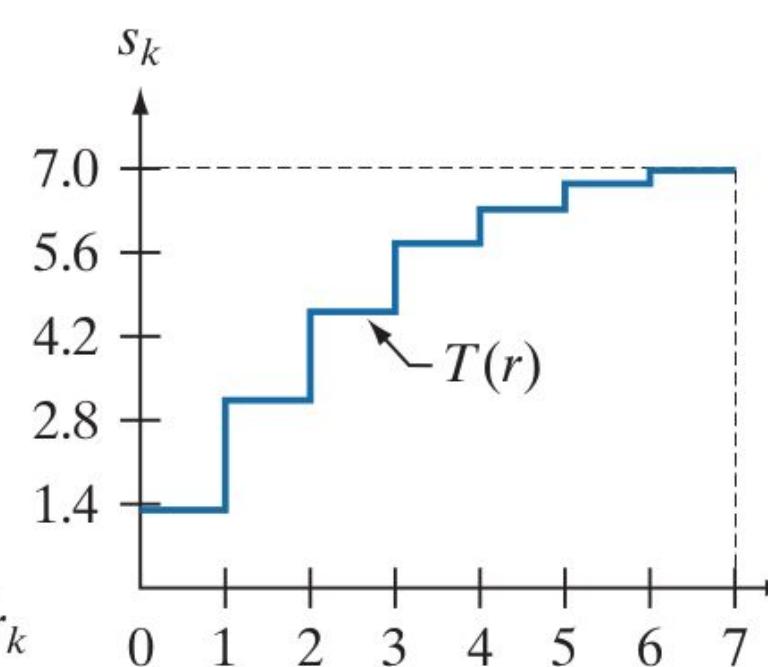
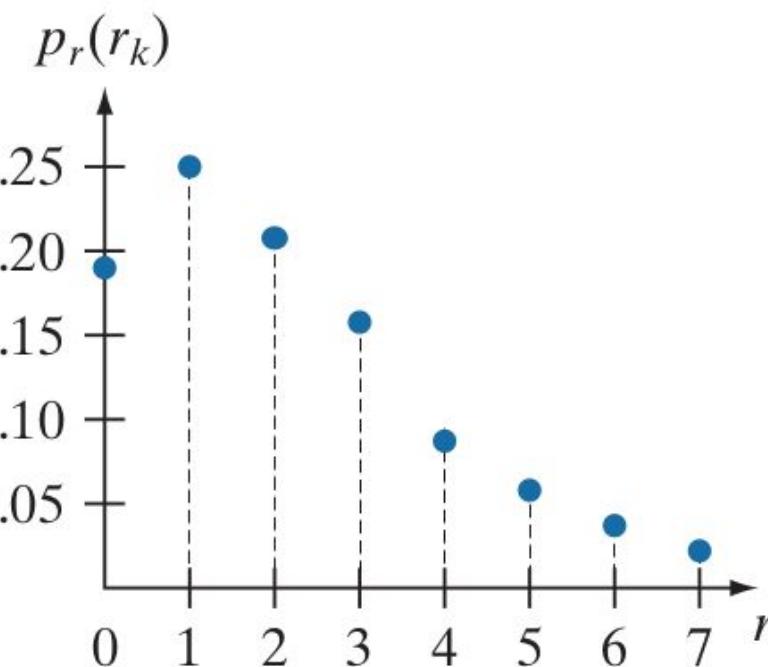
$$s_6 = 6.86 \rightarrow 7$$

$$s_1 = 3.08 \rightarrow 3$$

$$s_3 = 5.67 \rightarrow 6$$

$$s_5 = 6.65 \rightarrow 7$$

$$s_7 = 7.00 \rightarrow 7$$

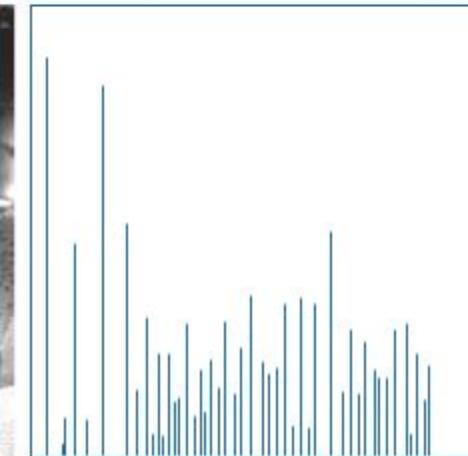
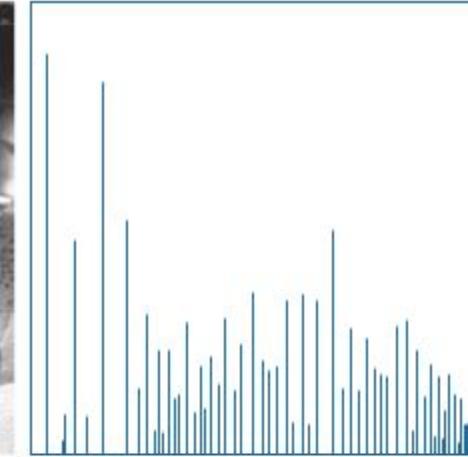


Original histogram.

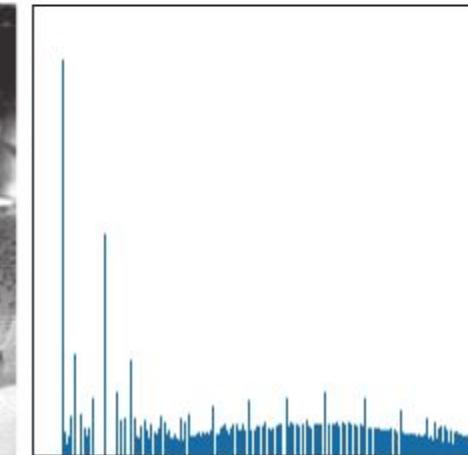
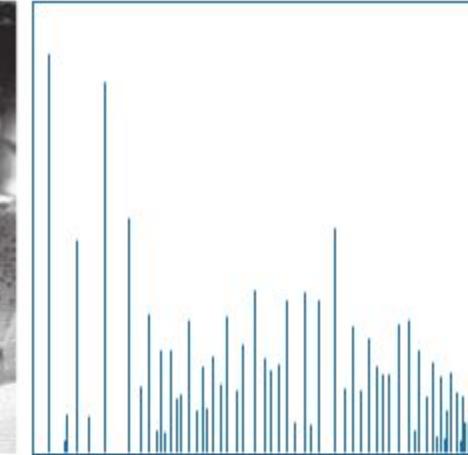
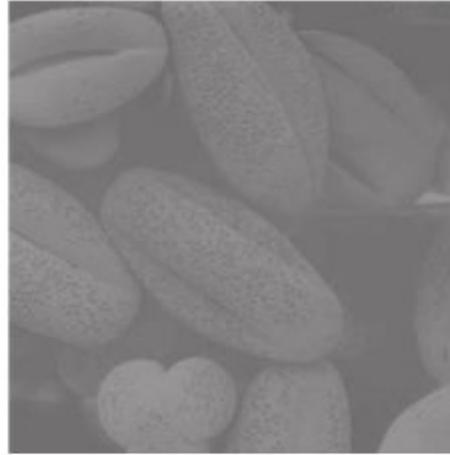
Transformation function.

Equalized histogram.

Histogram Equalization - Example



Histogram Equalization - Example



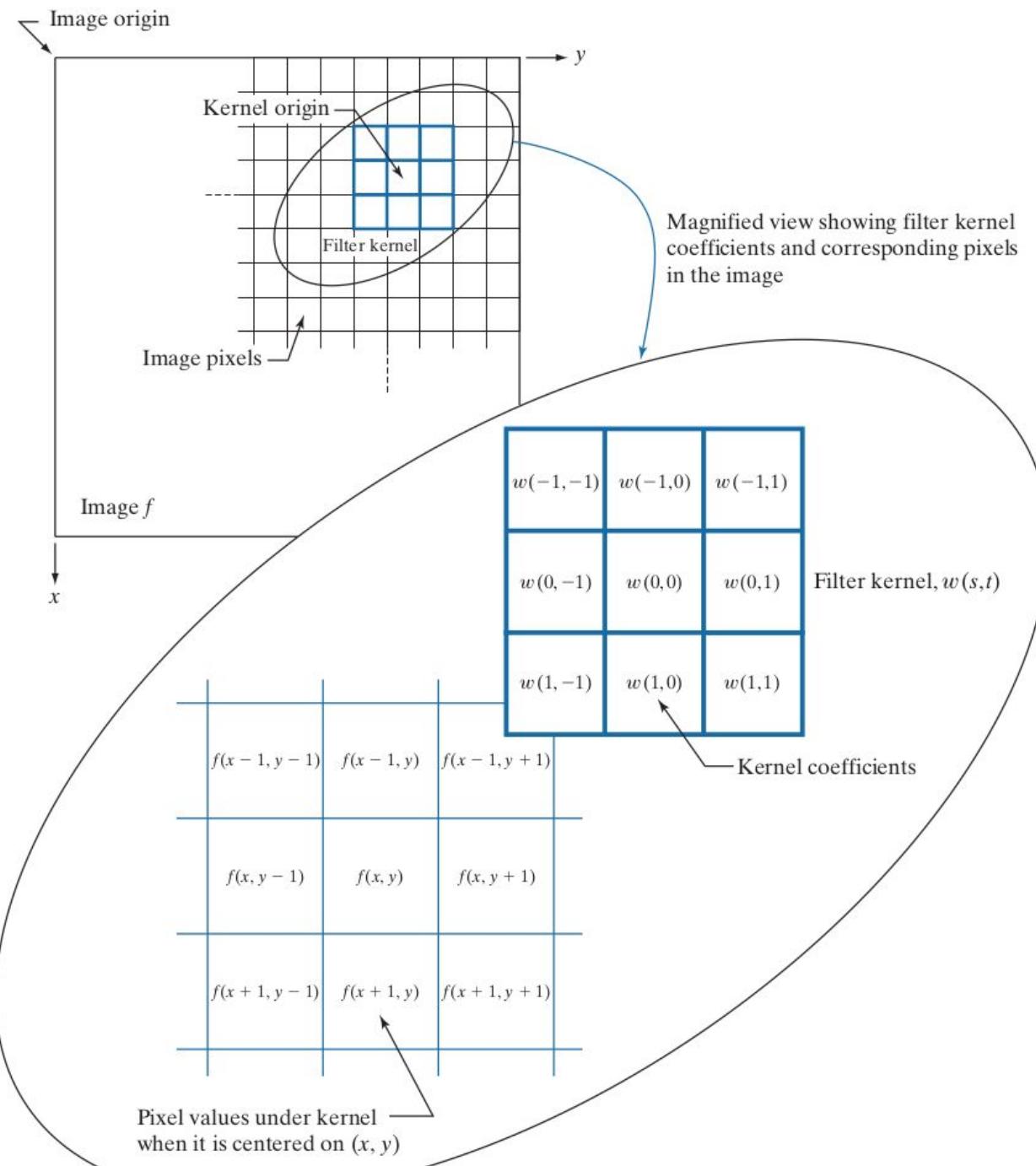


Spatial Filtering

Note:

Origin of the image is at the top left, but the origin of the kernel is at its center.

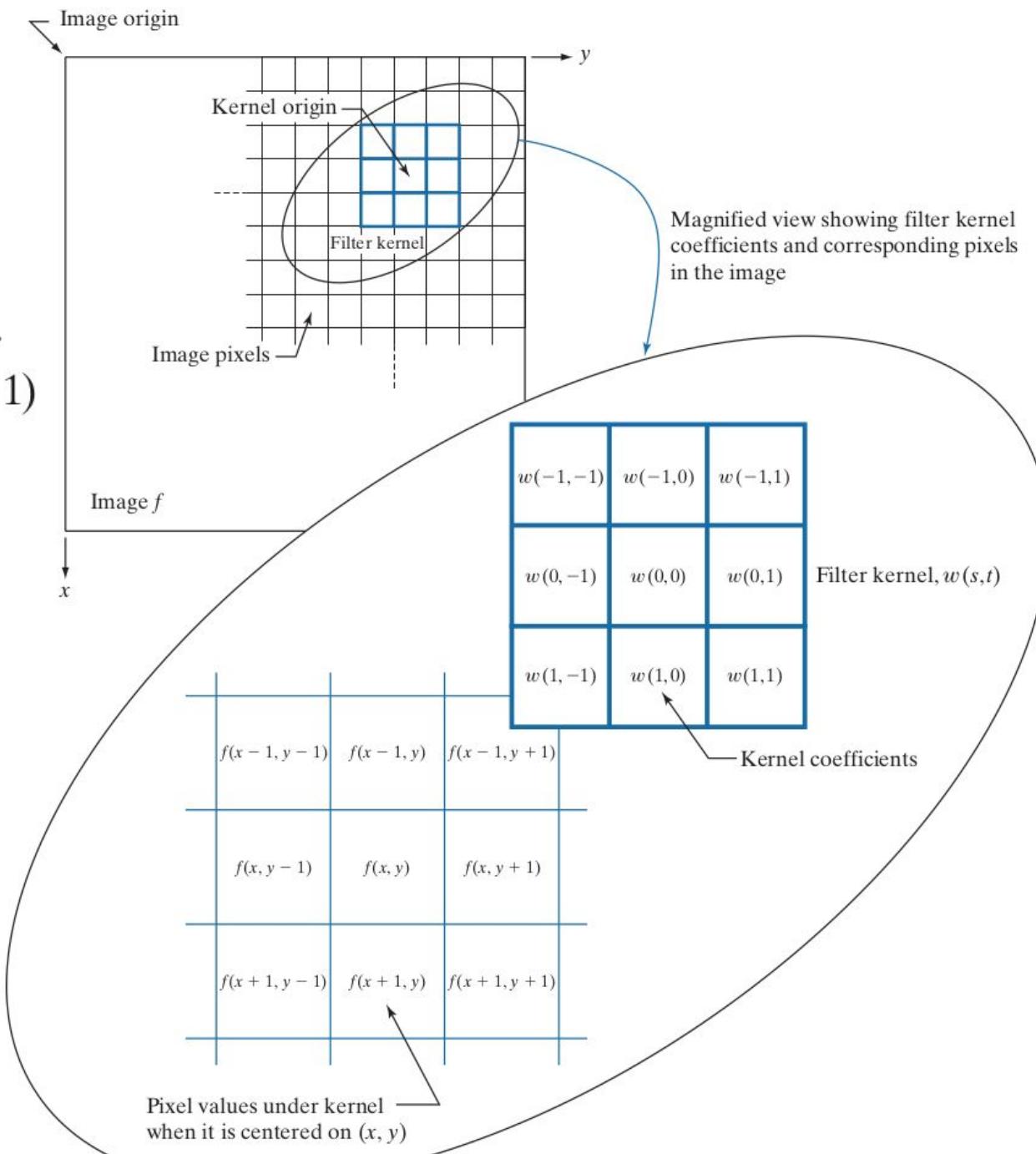
Placing the origin at the center of spatially symmetric kernels simplifies writing expressions for linear filtering.



Spatial Filtering

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

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



Note:

$$m = 2a + 1$$

$$n = 2b + 1$$



Spatial Filtering

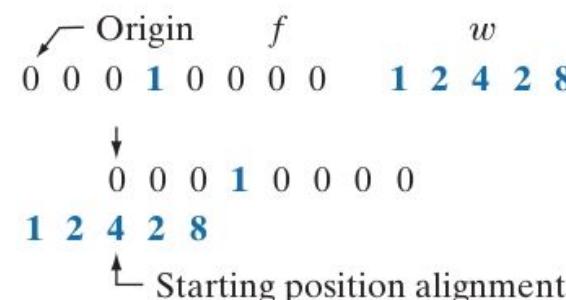
- A spatial filter consists of (a) **a neighborhood**, and (b) **a predefined operation**
- Linear spatial filtering of an image of size $M \times N$ with a filter of size $m \times n$ is given by the expression

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

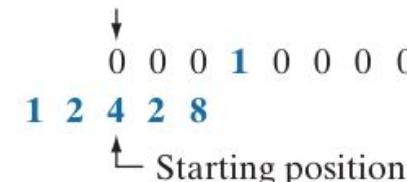
Spatial Filtering

Correlation

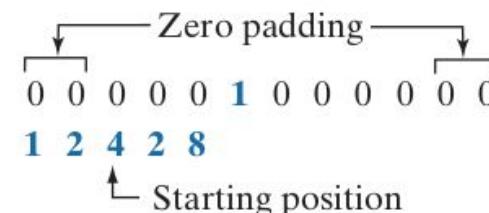
(a)



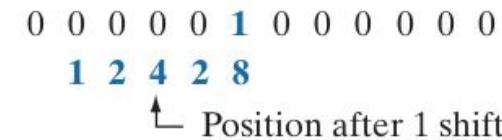
(b)



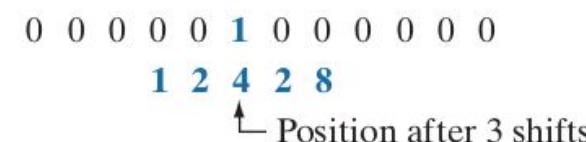
(c)



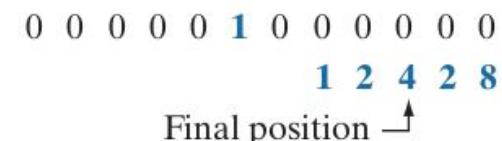
(d)



(e)



(f)

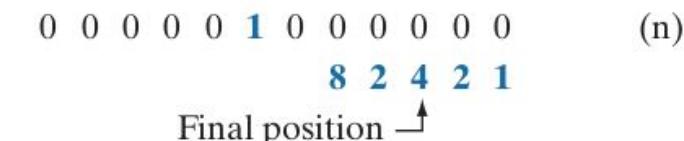
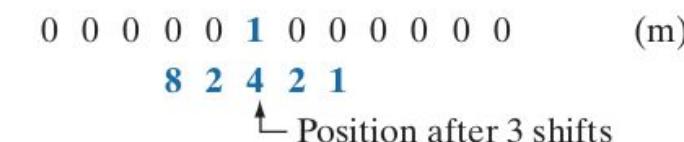
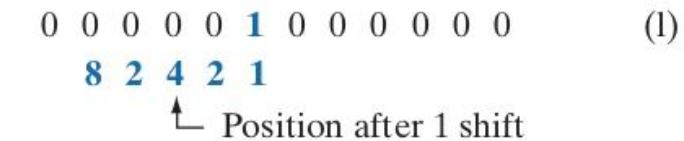
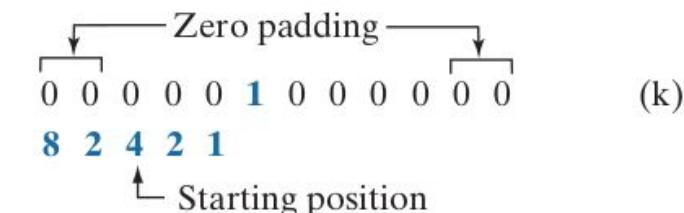
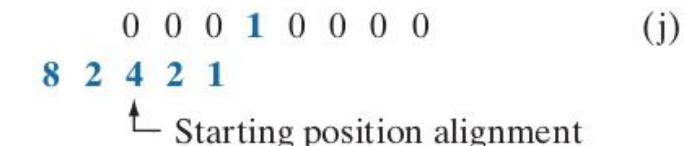
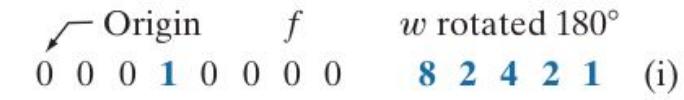


Correlation result

(g)

0 8 2 4 2 1 0 0

Convolution



Convolution result

0 1 2 4 2 8 0 0

(o)

Spatial Filtering

Padded f				
Origin	f	w	w	f
0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
0 0 0 0 0	0 0 0 0 0	1 2 3	1 2 3	0 0 0 1 0 0 0
0 0 1 0 0	0 0 0 0 0	4 5 6	4 5 6	0 0 0 0 0 0 0
0 0 0 0 0	0 0 0 0 0	7 8 9	7 8 9	0 0 0 0 0 0 0

(a) (b)

Correlation result				
Initial position for w				
1 2 3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0
4 5 6	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0
7 8 9	0 0 0 0	0 0 0 0	0 0 0 0	0 9 8 7 0
0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 6 5 4 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 3 2 1 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0

(c) (d)

Convolution result				
Rotated w				
9 8 7	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0
6 5 4	0 0 0 0	0 0 0 0	0 0 0 0	0 1 2 3 0
3 2 1	0 0 0 0	0 0 0 0	0 0 0 0	0 4 5 6 0
0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 7 8 9 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0 0

(f) (g)

Smoothing Spatial Filters

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

Box kernel

$$\frac{1}{4.8976} \times \begin{array}{|c|c|c|}\hline 0.3679 & 0.6065 & 0.3679 \\ \hline 0.6065 & 1.0000 & 0.6065 \\ \hline 0.3679 & 0.6065 & 0.3679 \\ \hline \end{array}$$

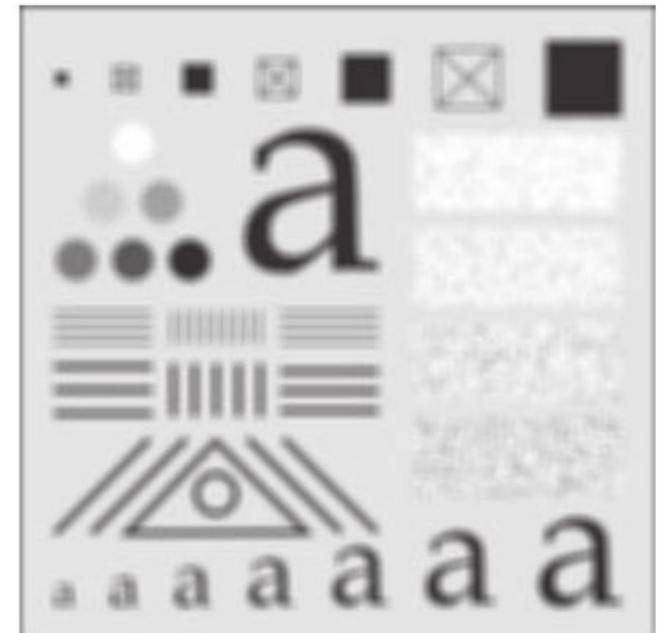
Gaussian kernel.

Smoothing Spatial Filters

(a) Test pattern of size 1024×1024 pixels.

(b)-(d) Results of low pass filtering **with box kernels** of sizes 3×3 , 11×11 , and 21×21 , respectively.

a b
c d

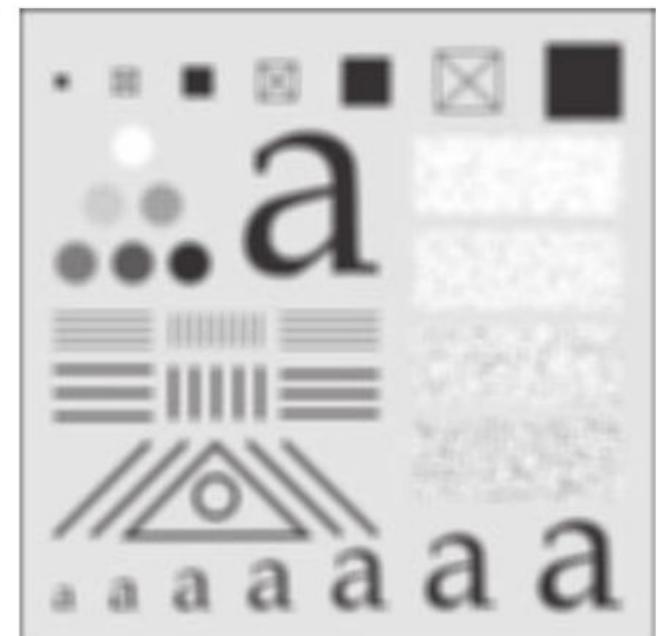


Smoothing Spatial Filters

(a) Test pattern of size 1024×1024 pixels.

(b)-(d) Results of low pass filtering **with box kernels** of sizes 3×3 , 11×11 , and 21×21 , respectively.

a b
c d





Spatial Filtering

- A spatial filter consists of (a) **a neighborhood**, and (b) **a predefined operation**
- Linear spatial filtering of an image of size $M \times N$ with a filter of size $m \times n$ is given by the expression

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

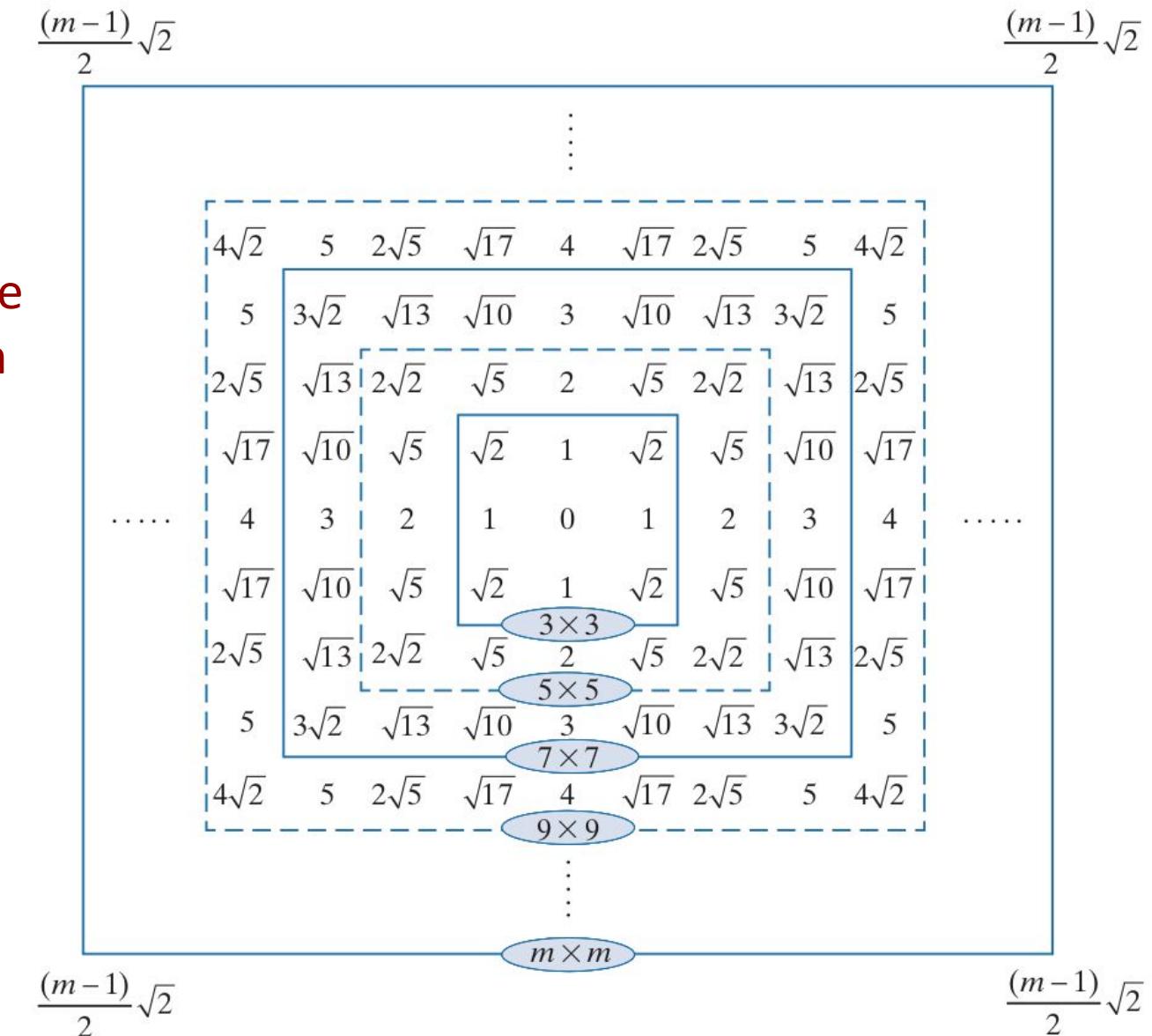
Gaussian Filters

Let s and r be position of a cell in a filter w.r.t the origin; r represent the distance of the cell from the center

$$r = [s^2 + t^2]^{1/2}$$

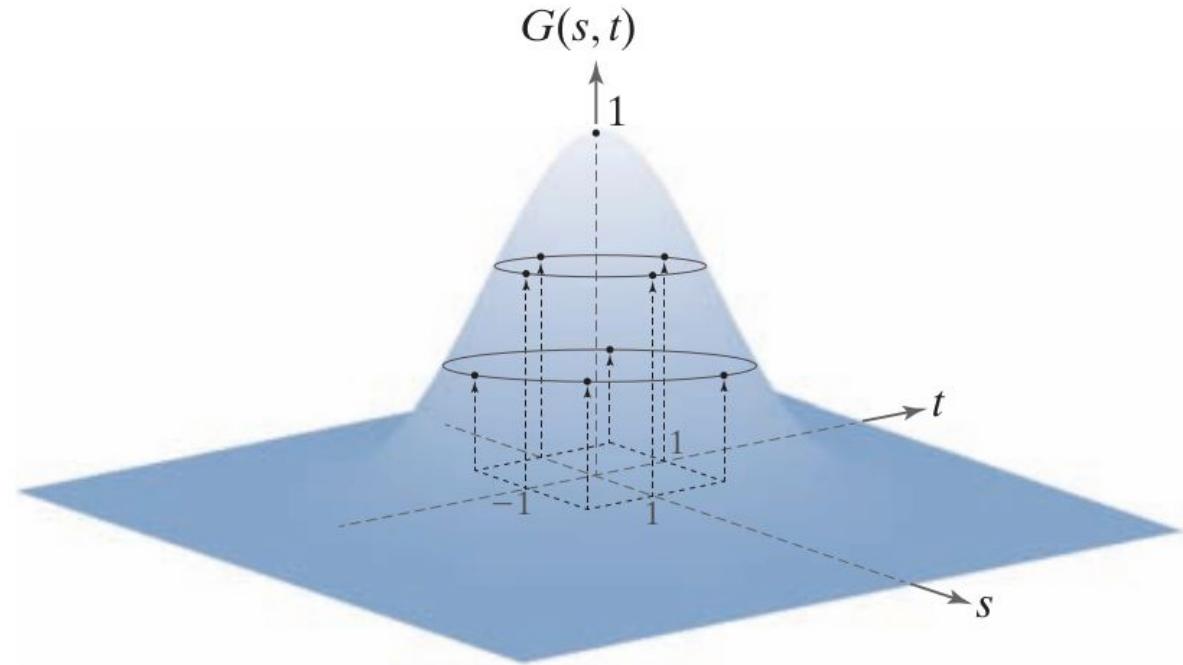
Form of Gaussian kernels is as below:

$$G(r) = Ke^{-\frac{r^2}{2\sigma^2}}$$



Distances from the center for various sizes of square kernels.

Gaussian Filters



Sampling a Gaussian function to obtain a discrete Gaussian kernel. The values shown are for K = 1 and $\sigma = 1$.

$$\frac{1}{4.8976} \times$$

0.3679	0.6065	0.3679
0.6065	1.0000	0.6065
0.3679	0.6065	0.3679

Resulting 3×3 kernel

$$r = [s^2 + t^2]^{1/2}$$

$$G(r) = Ke^{-\frac{r^2}{2\sigma^2}}$$

Gaussian Filters



A test pattern of size 1024×1024



lowpass filtering the pattern with a Gaussian kernel of size 21×21 , with $\sigma = 3.5$, $K=1$



Result of using a kernel of size 43×43 , with $\sigma = 7$, $K=1$

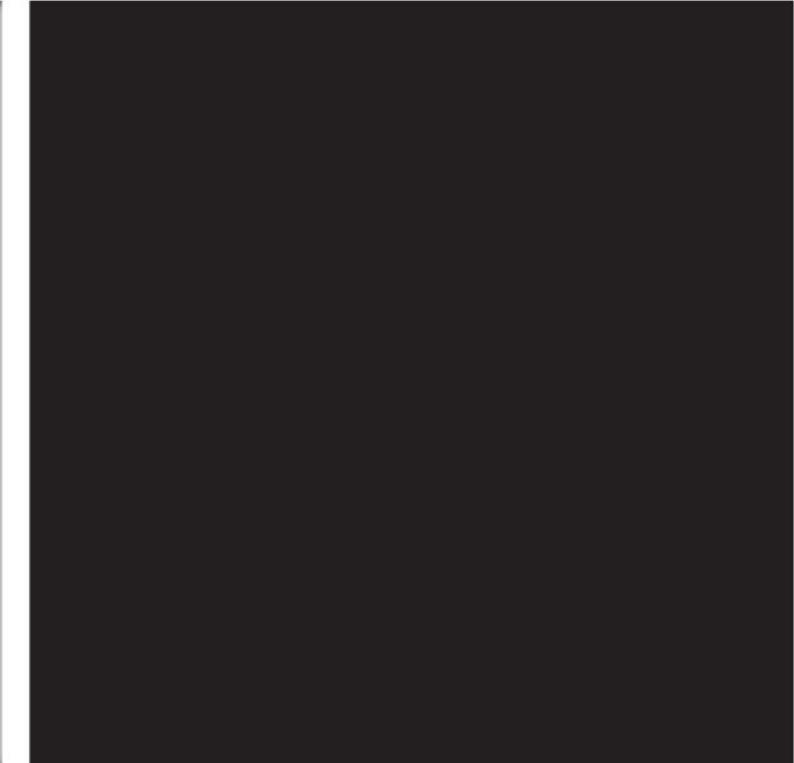
Gaussian Filters



**Gaussian kernels of size 43×43 ,
with $\sigma = 7$**



Kernel of 85×85 , with $\sigma = 7$



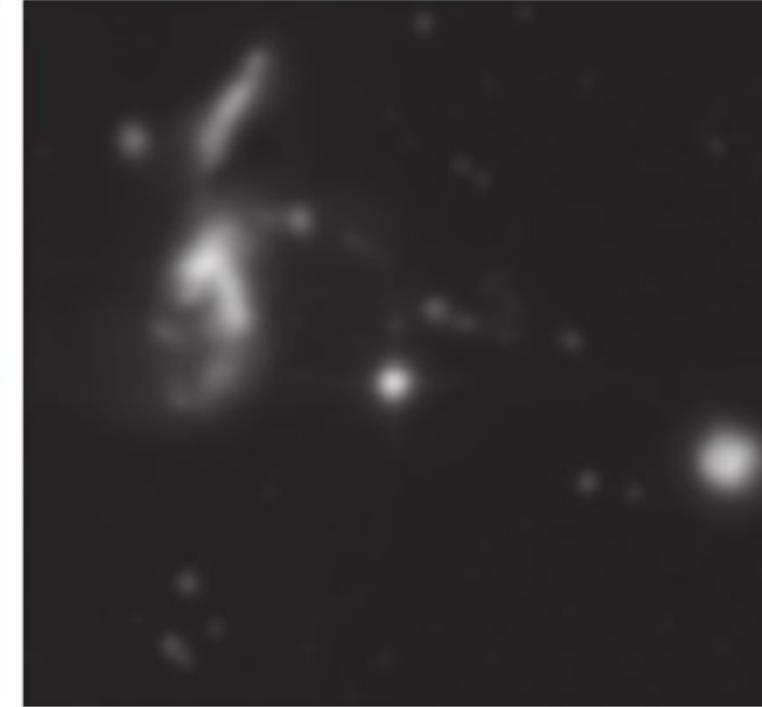
Difference image

Gaussian Filters - Example Usage

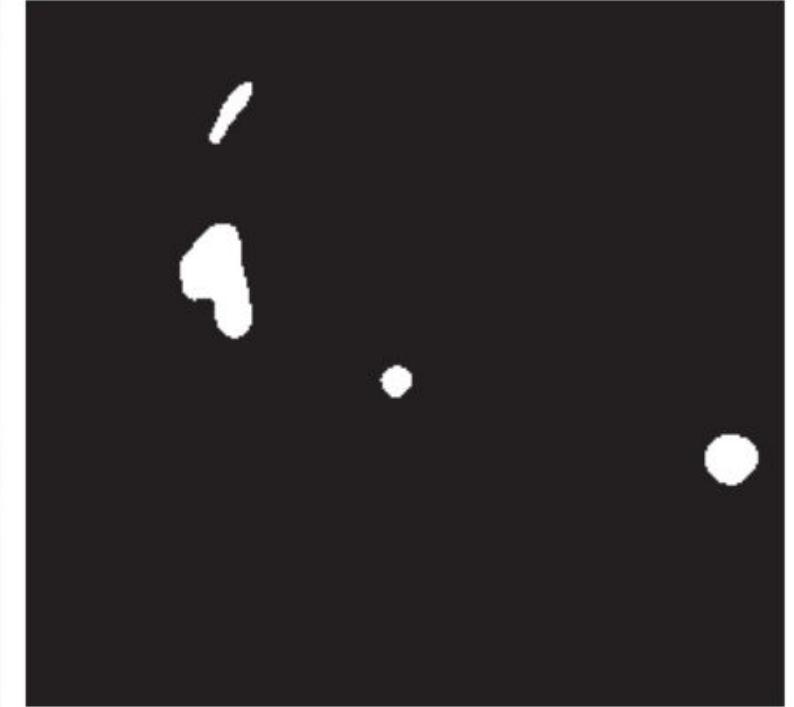


a b c

A 2566×2758 Hubble Telescope image of the Hickson Compact Group



Result of lowpass filtering with a Gaussian kernel.



Result of thresholding the filtered image (intensities were scaled to the range $[0, 1]$)

Gaussian Filters - Example Usage

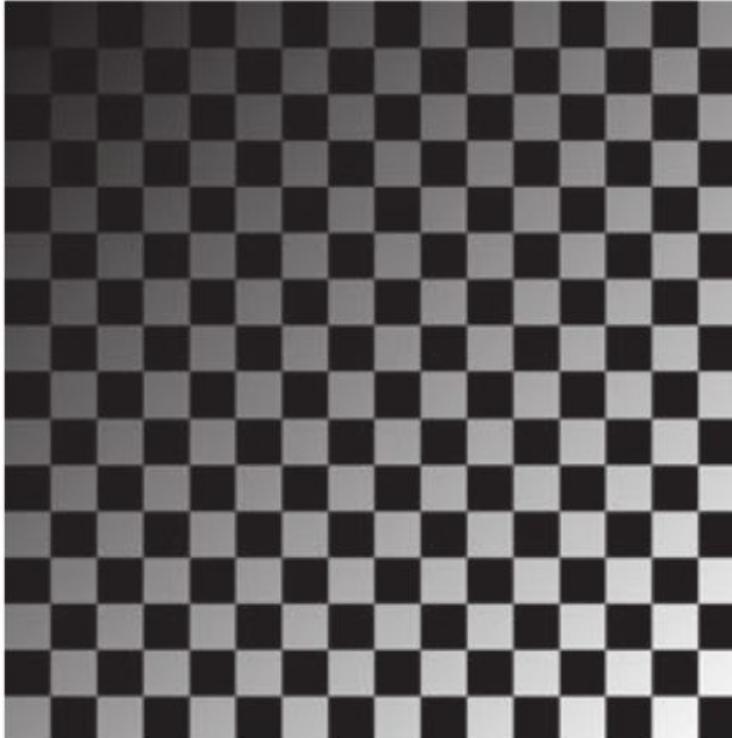
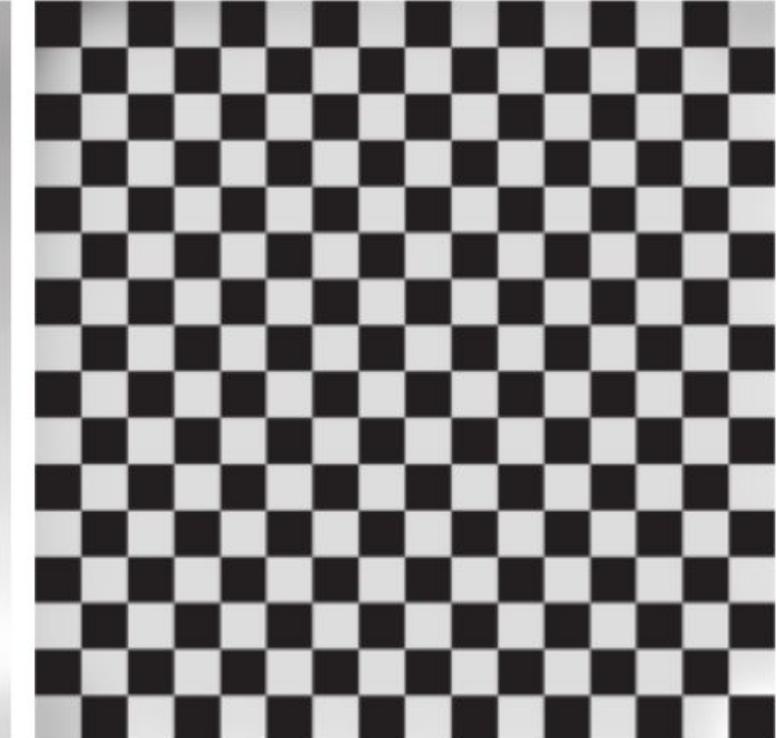


Image shaded by a shading pattern oriented in the -45° direction



Estimate of the shading patterns obtained using lowpass filtering.



Result of dividing (a) by (b). (See Section 9.8 for a morphological approach to shading correction).



- **Readings:**
Digital Image Processing, Rafael C. Gonzalez & Richard E woods, Third Ed, Chapter 3;
- **Next Class:**
Image Sharpening; Linear Filters; Edge Detection using Gradients, Sobel,Canny; Demonstration of Applications in Python
Line detection using Hough transforms; [if possible]



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision
2023-24 First Semester, M.Tech (AIML)

Session #4: Edge Detection & Line Detection

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

S. P. Vimal | CSIS Department | WILP | BITS – Pilani (vimalsp@wilp.bits-pilani.ac.in)

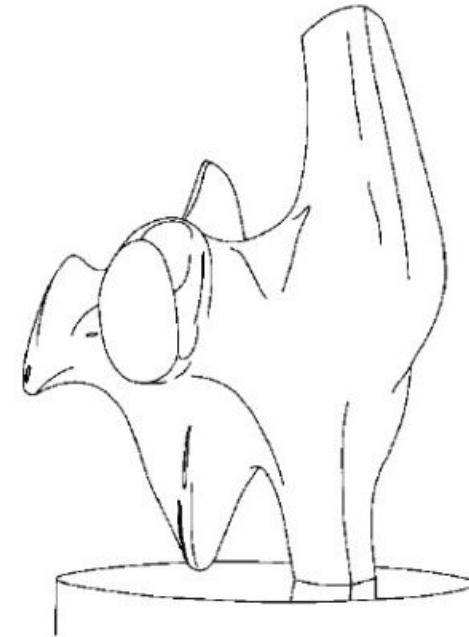
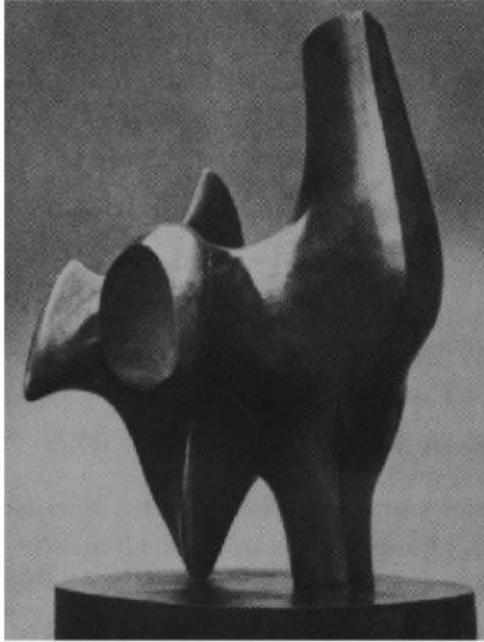
Topics

- Edge Models & Approaches to edge detection
- Hough Transform - Detecting Lines

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely



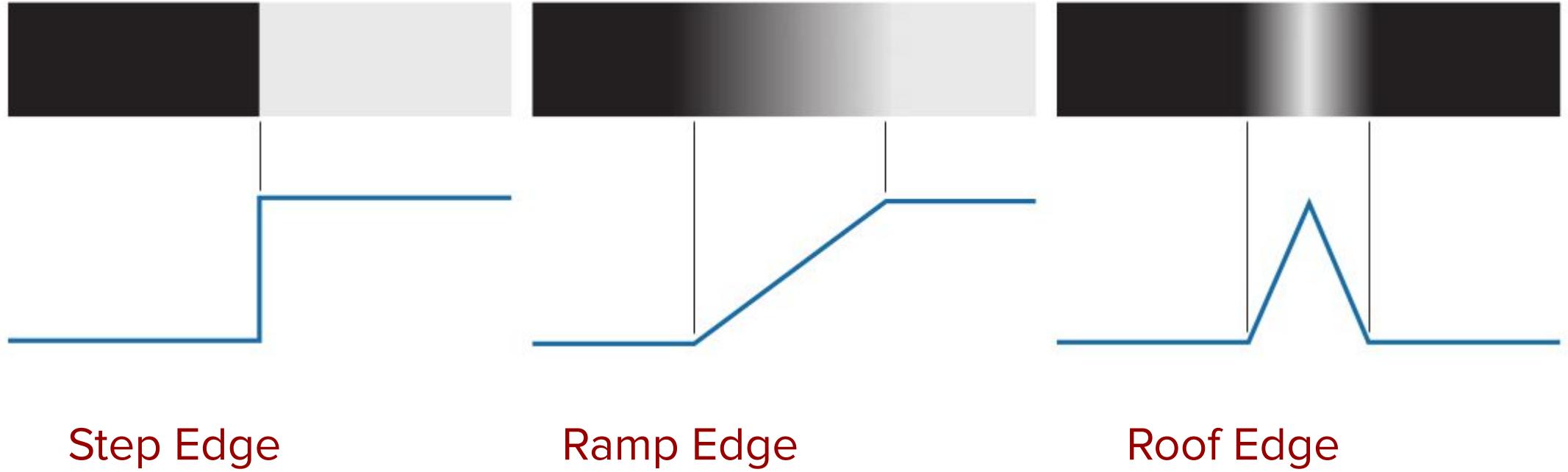
Edge Detection



Task : Convert a 2D image into a set of curves (made of edges)

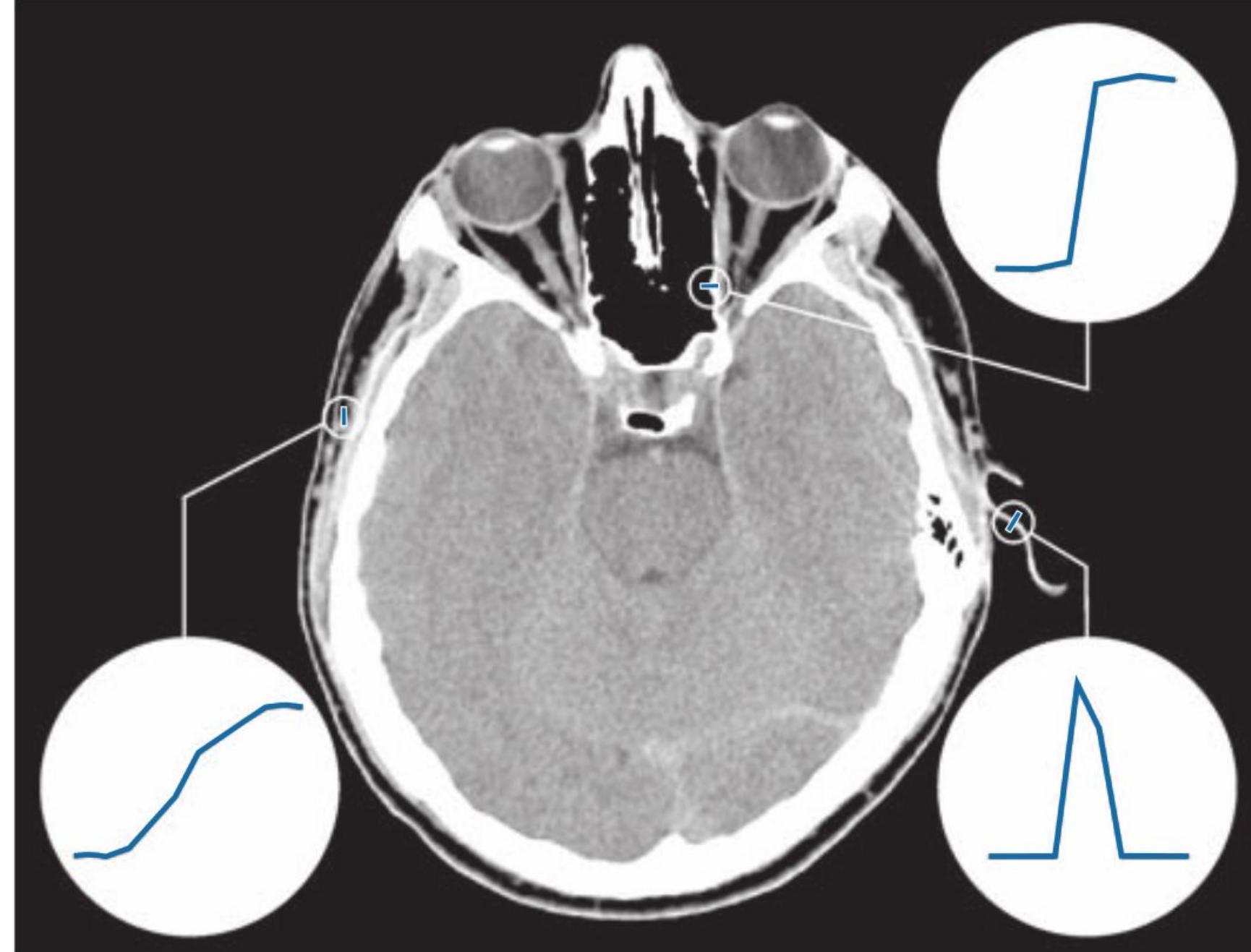


Edge Profiles



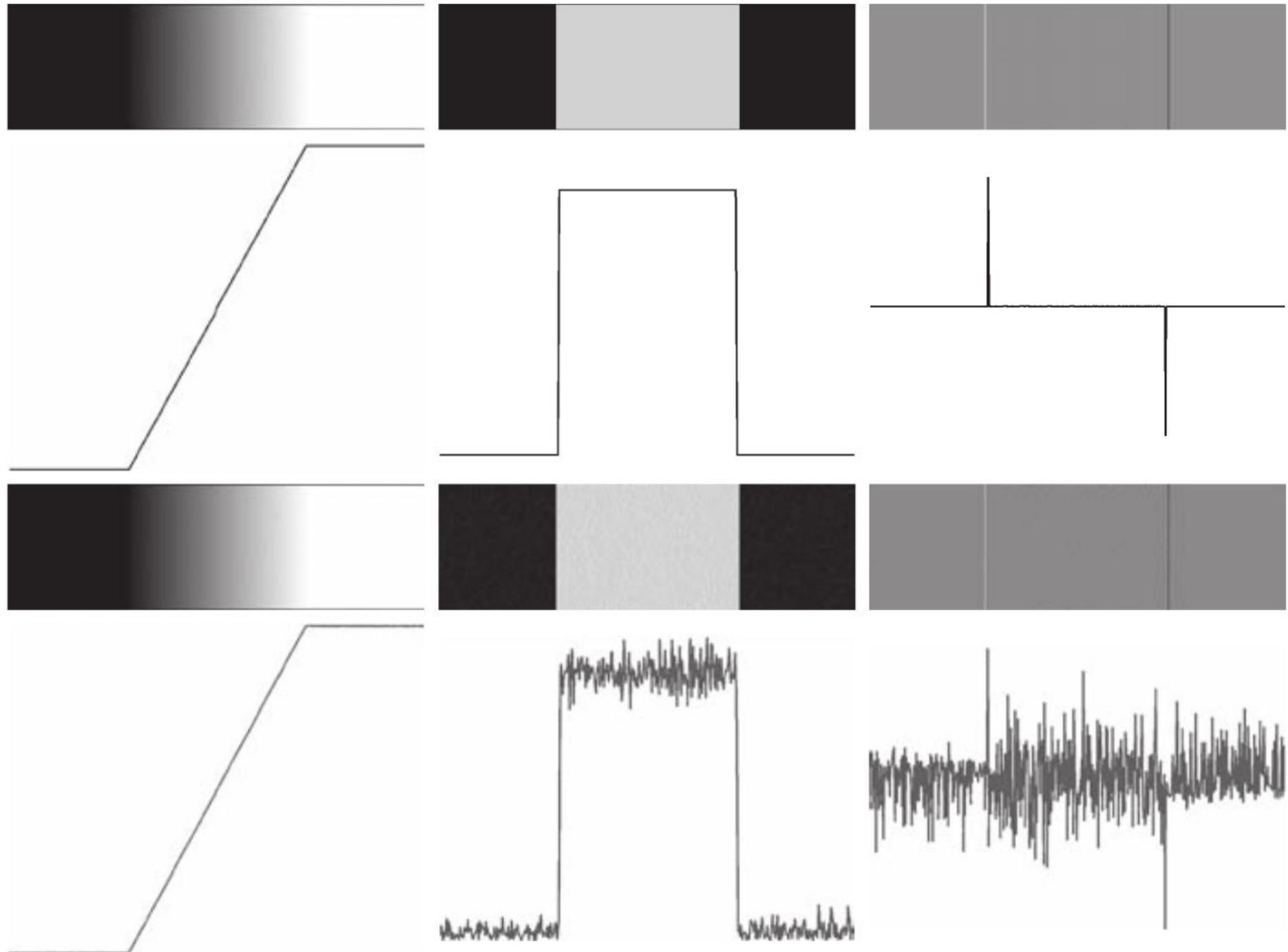


Example





Edge Detection Approach





Edge Detection Approach

1. **Image smoothing** for noise reduction.
2. **Detection of edge points.** As mentioned earlier, this is a local operation that extracts from an image all points that are potential edge-point candidates.
3. **Edge localization.** The objective of this step is to select from the candidate points only the points that are members of the set of points comprising an edge.



Gradient and it's Computation

- How can we differentiate a *digital* image $F[x,y]$?
 - **Option 1:** reconstruct a continuous image, f , then compute the derivative
 - **Option 2:** take discrete derivative (finite difference)

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

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



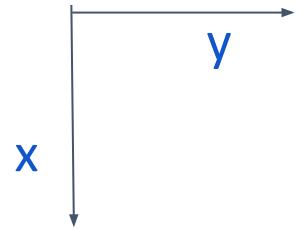
Gradient and it's Computation

- How can we differentiate a *digital* image $F[x,y]$?
 - **Option 1:** reconstruct a continuous image, f , then compute the derivative
 - **Option 2:** take discrete derivative (finite difference)

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

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

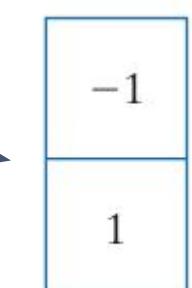
Gradient and it's Computation



- How can we differentiate a *digital* image $F[x,y]$?
 - **Option 1:** reconstruct a continuous image, f , then compute the derivative
 - **Option 2:** take discrete derivative (finite difference)

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


(a)

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


(b)



Gradient and it's Computation

Compute partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at every pixel location in the image

$$\nabla f(x, y) \equiv \text{grad}[f(x, y)] \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Points in the direction of maximum rate of change of f at (x, y)

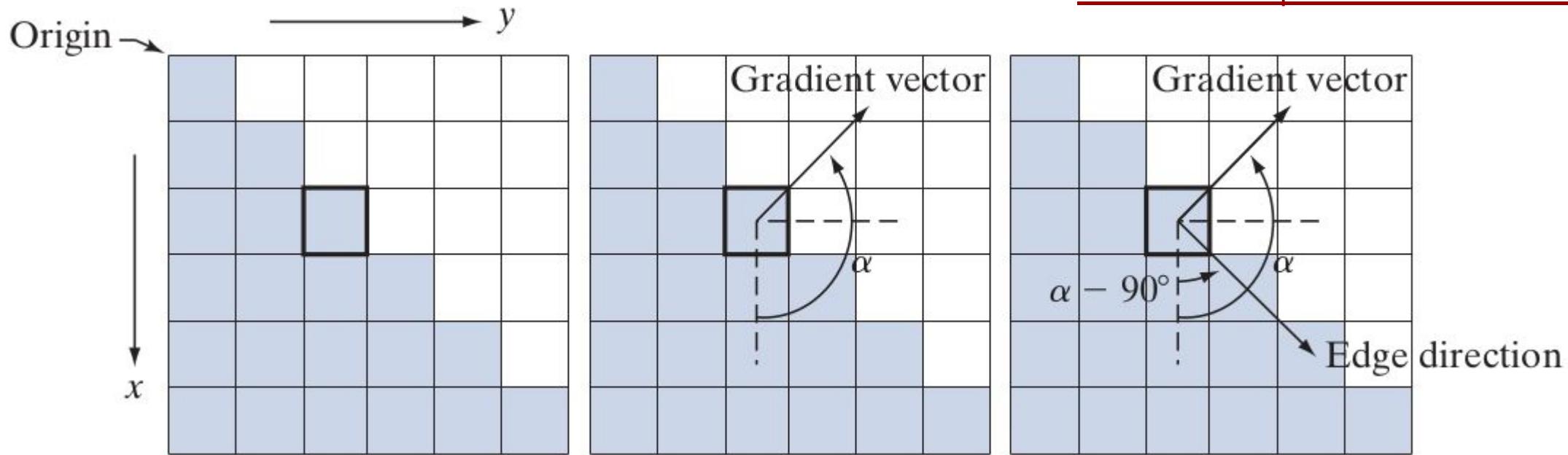
$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right]$$

Direction of gradient vector at (x, y)

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

Value of the rate of change in the direction of the gradient vector at (x, y)

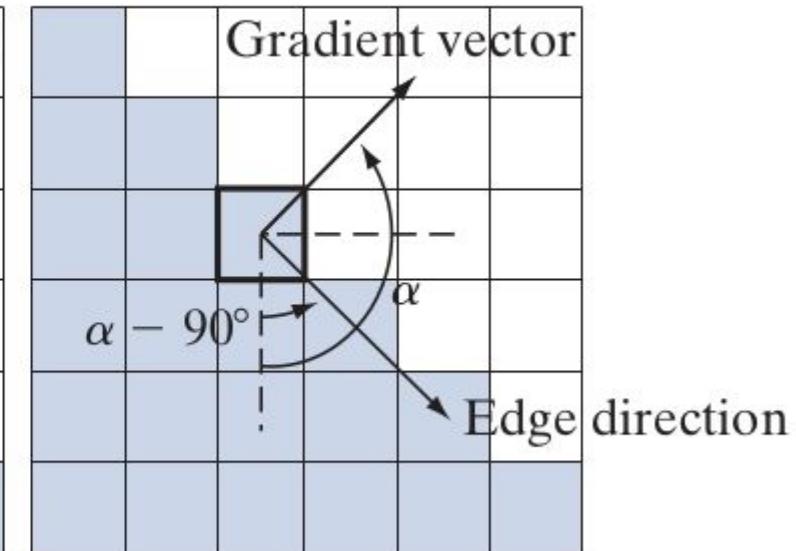
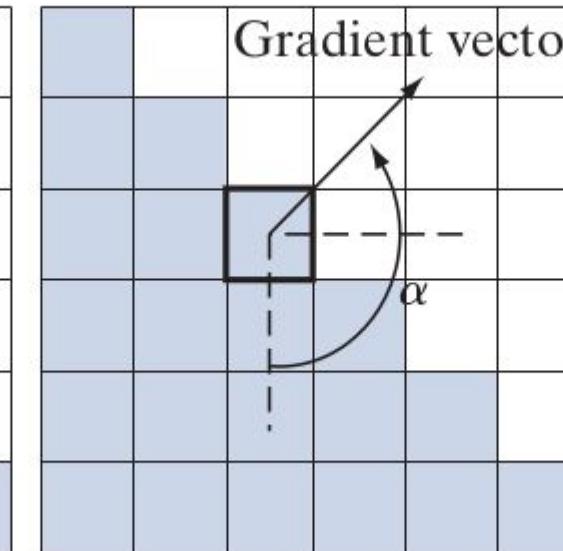
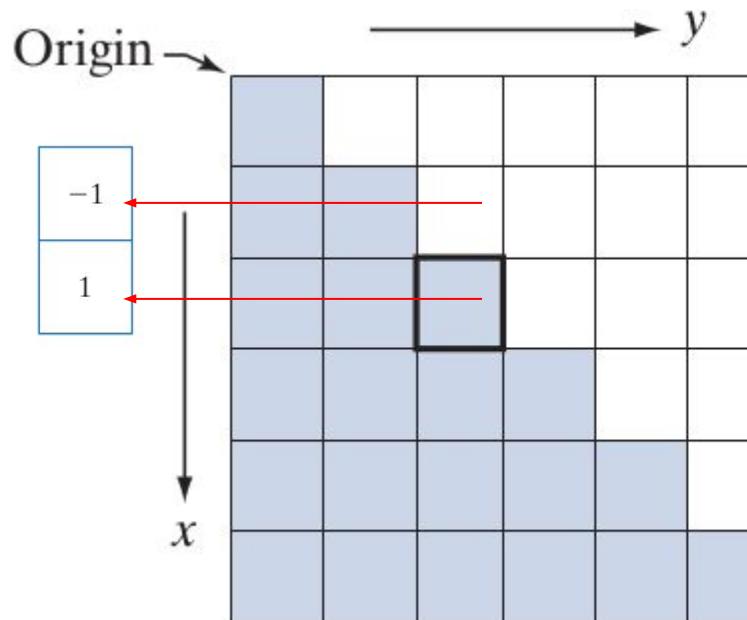
Gradient and it's Computation - Example



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

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

Gradient and it's Computation - Example



The shaded pixels are 0's & white are 2's

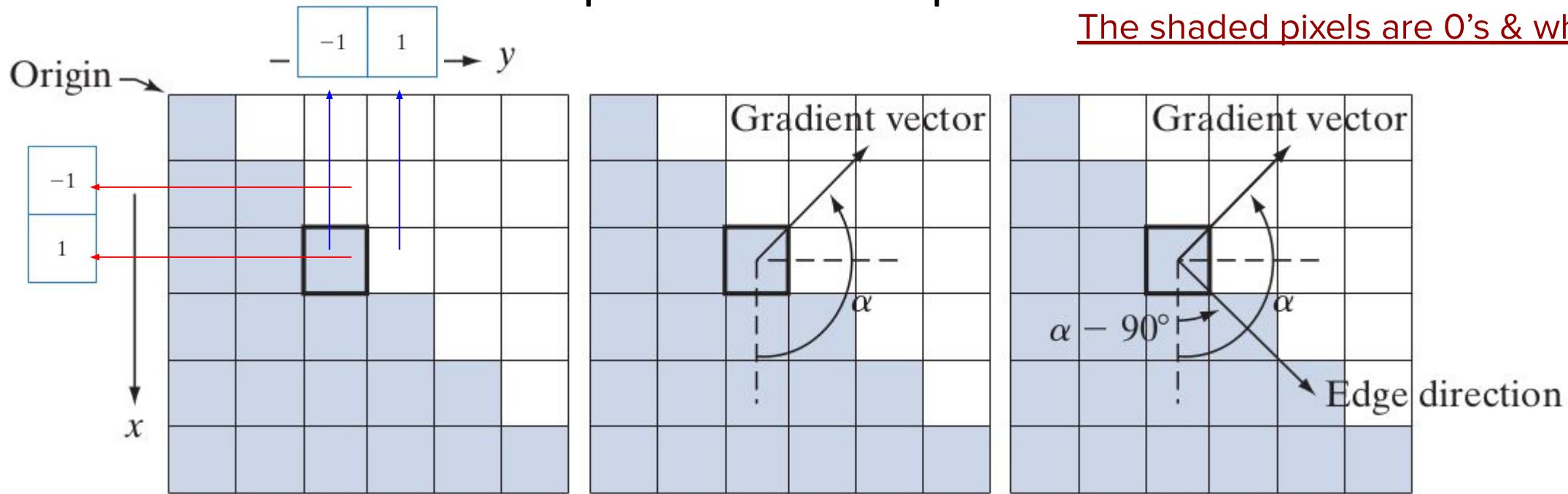
$$g_x(x, y) = \frac{\partial f(x, y)}{\partial x} = f(x + 1, y) - f(x, y)$$

Subtract pixels in the top row of the neighborhood from the pixels in the bottom row → partial derivative in the x-direction

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

Subtract pixels in the left column from the pixels in the right column of the neighborhood → partial derivative in the y-direction

Gradient and it's Computation - Example



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

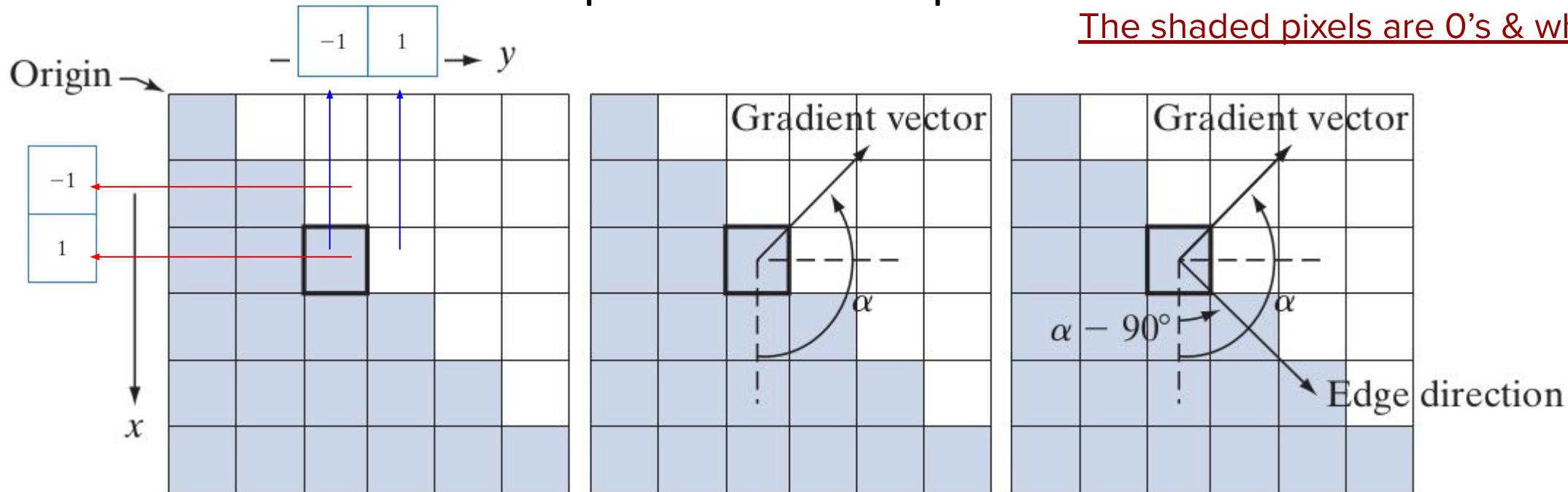
$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} = f(x, y+1) - f(x, y) = 2$$



$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

The shaded pixels are 0's & white are 2's

Gradient and it's Computation - Example



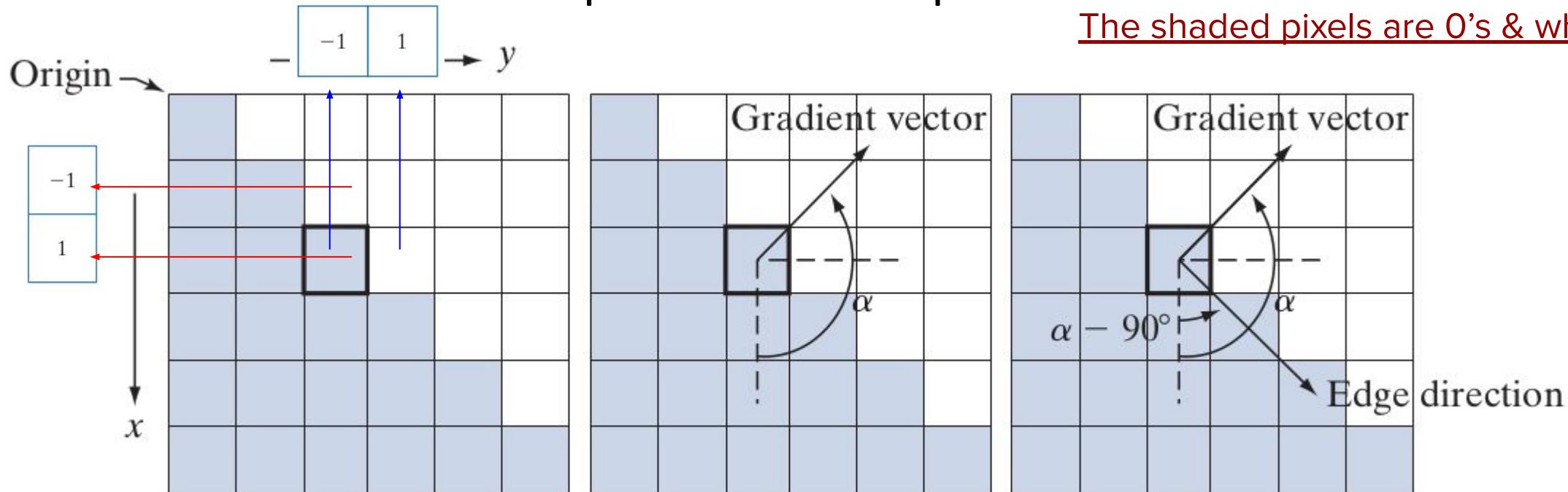
$$\|\nabla f\| = 2\sqrt{2}$$

←

$$\alpha = \tan^{-1}(g_y/g_x) = -45^\circ = 135^\circ$$

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

Gradient and it's Computation - Example



Direction of an edge at a point is orthogonal to the gradient vector at that point \Rightarrow Direction of Edge is

$$\alpha - 90^\circ = 135^\circ - 90^\circ = 45^\circ$$

$$\alpha = \tan^{-1}(g_y/g_x) = -45^\circ = 135^\circ$$



$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$



Gradient and it's Computation - Different Forms

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

A 3×3 region of an image (the z's are intensity values)

To Do: Compute gradient at z_5

Gradient and it's Computation - Different Forms

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Prewitt's operators

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

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

A 3×3 region of an image (the z's are intensity values)

To Do: Compute gradient at z_5

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Gradient and it's Computation - Different Forms

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Sobel's operators

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

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

A 3×3 region of an image (the z's are intensity values)

To Do: Compute gradient at z_5

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$



Gradient and it's Computation - How to use?

- (1) Use any of the gradient operators to obtain g_x and g_y at every pixel location
- (2) Approximate gradient at each of the pixel locations as below:

$$M(x, y) \approx |g_x| + |g_y|$$

Gradient and it's Computation - Example

a	b
c	d

(a) Image of size 834×1114 pixels, with intensity values scaled to $[0, 1]$.

(b) $|g_x|$, the component of gradient obtained using the Sobel kernel

(c) $|g_y|$

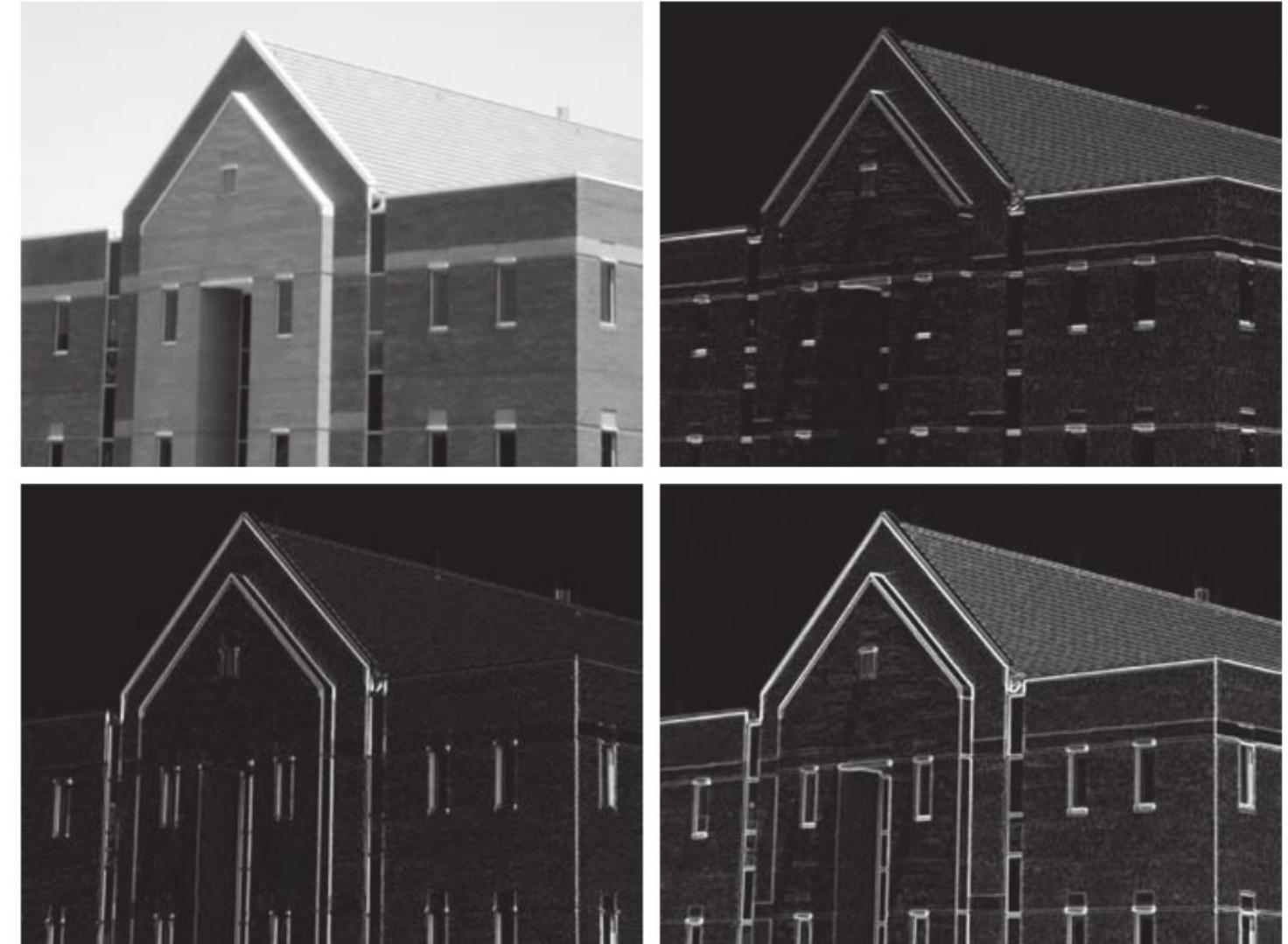
(d) The gradient (using sum of absolute values)



Gradient and it's Computation - Example

a	b
c	d

Same sequence as prev. fig. but with the original image smoothed using a 5×5 averaging kernel prior to edge detection.





Gradient and it's Computation - Kirsch compass kernels

-3	-3	5
-3	0	5
-3	-3	5

-3	5	5
-3	0	5
-3	-3	-3

5	5	5
-3	0	-3
-3	-3	-3

5	5	-3
5	0	-3
-3	-3	-3

N

NW

W

SW

5	-3	-3
5	0	-3
5	-3	-3

-3	-3	-3
5	0	-3
5	5	-3

-3	-3	-3
-3	0	-3
-3	5	5

S

SE

E

NE



Canny Edge Detection

A powerful Edge Detection Approach,
Deal with it in the Next Class

Image Sharpening - Next Class

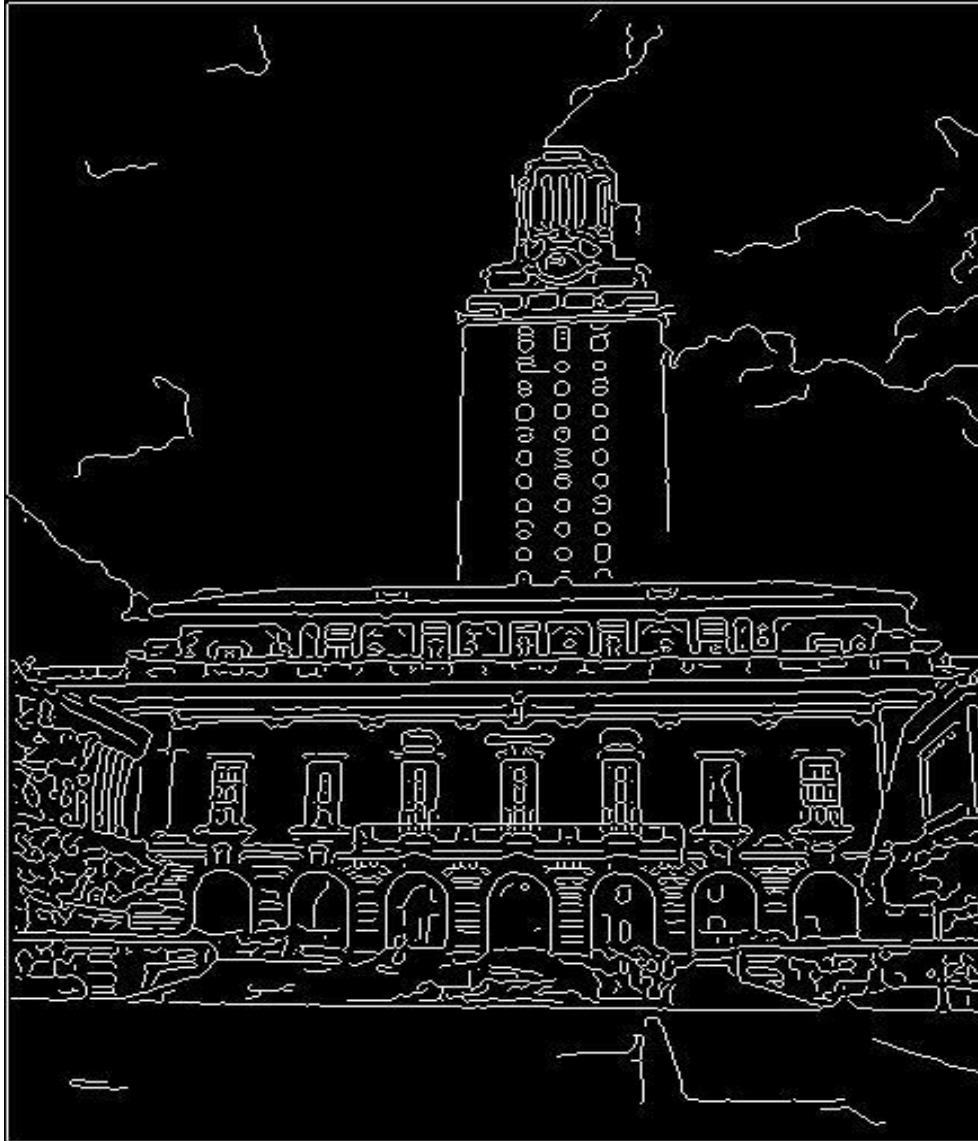
Line Fitting

Many objects characterized by presence of straight lines



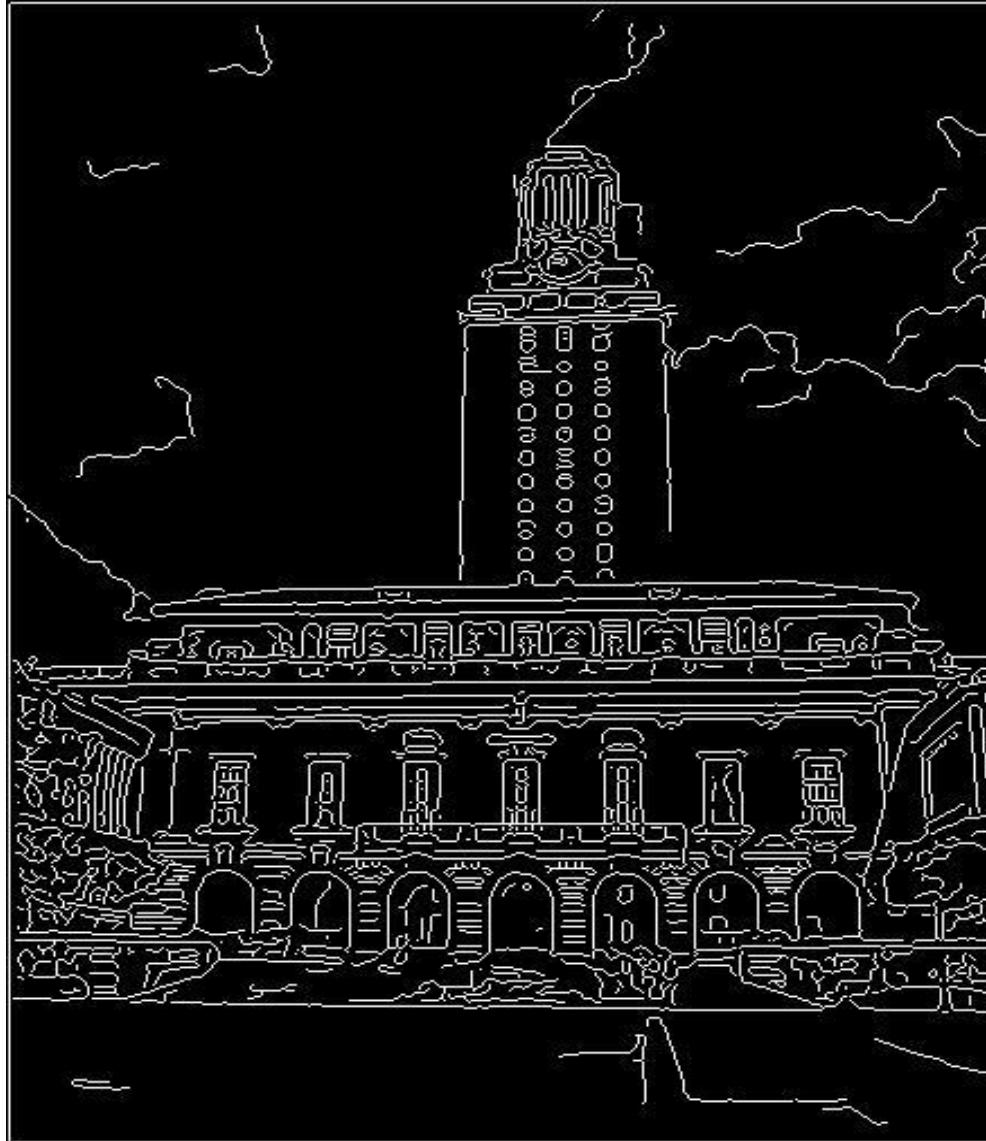
Wait, why aren't we done just by running edge detection?

Difficulty in Line Fitting



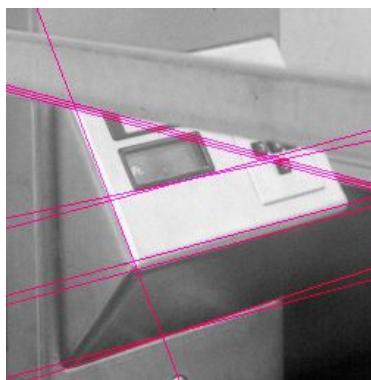
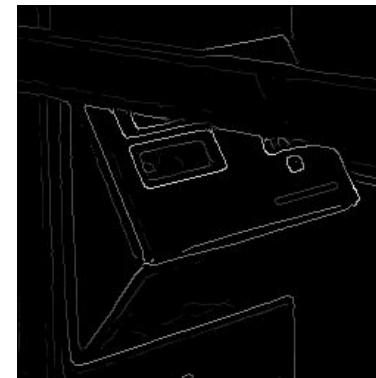
- Extra edge points (clutter), multiple models:
 - which points go with which line, if any?
- Only some parts of each line detected, and some parts are missing:
 - how to find a line that bridges missing evidence?
- Noise in measured edge points, orientations:
 - how to detect true underlying parameters?

Role of Voting in Line Fitting



- It's not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let the features *vote for all models that are compatible with it.*
 - Cycle through features, cast votes for model parameters.
 - Look for model parameters that receive a lot of votes.
- Noise & clutter features will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features.

Hough Transform for Line Fitting

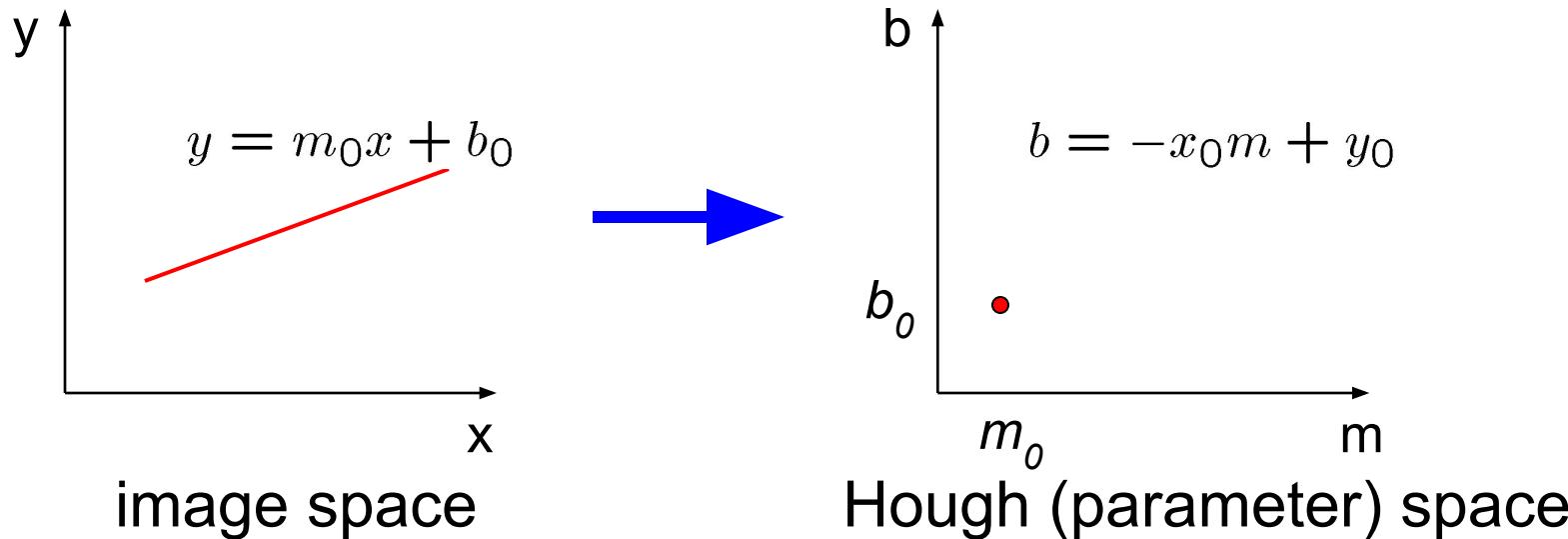


- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?
- Hough Transform is a voting technique that can be used to answer all of these questions.

Main idea:

1. Record vote for each possible line on which each edge point lies.
2. Look for lines that get many votes.

Hough Transform for Line Fitting

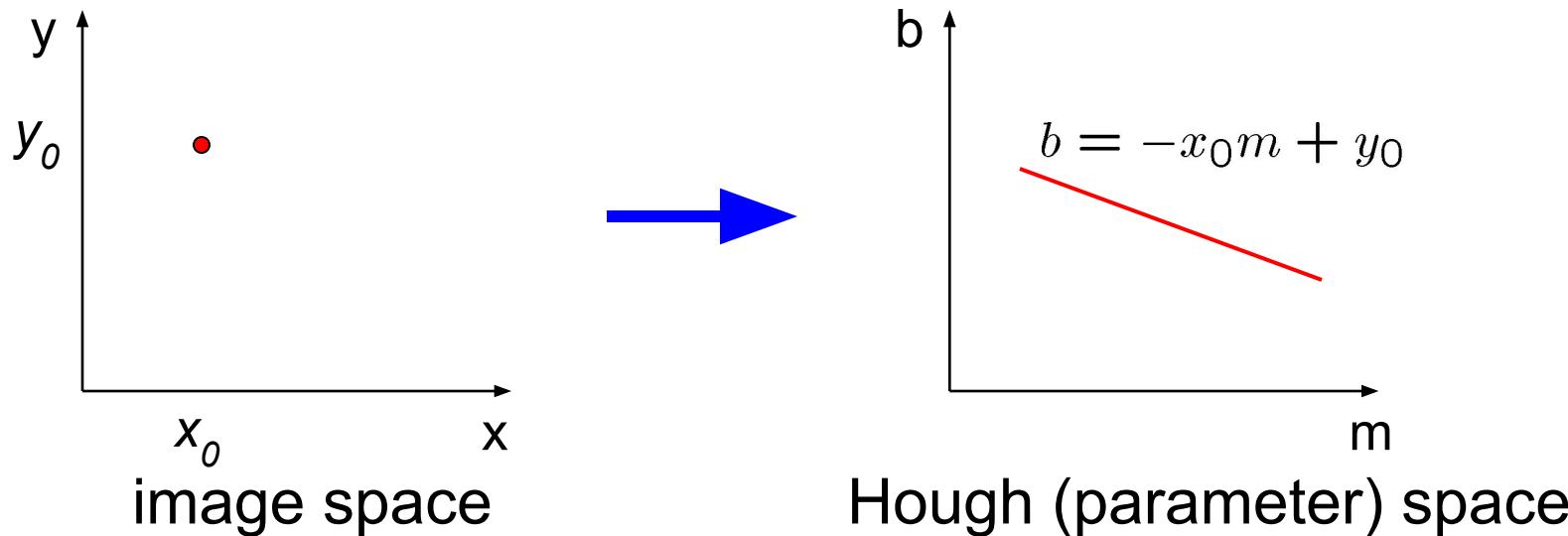


Equation of a line?
 $y = mx + b$

Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
 - given a set of points (x,y) , find all (m,b) such that $y = mx + b$
- What does a point (x_0, y_0) in the image space map to?
 - Answer: the solutions of $b = -x_0m + y_0$
 - this is a line in Hough space

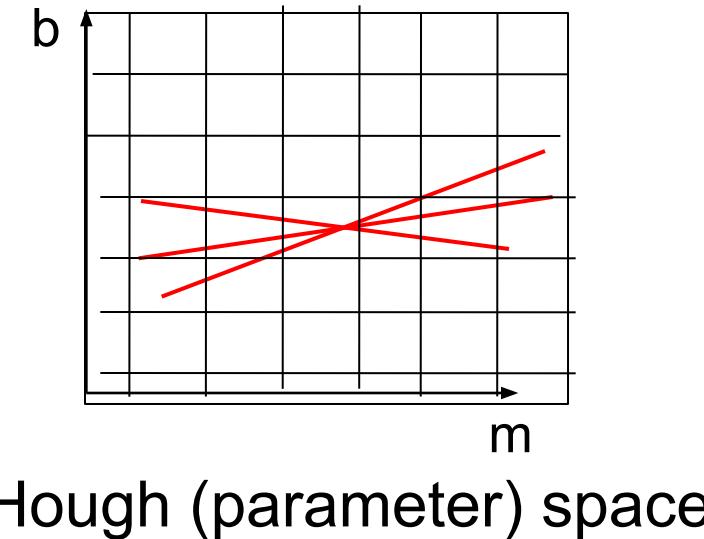
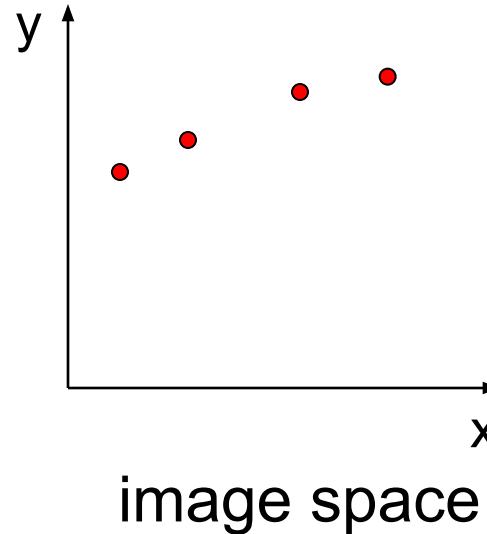
Hough Transform for Line Fitting



What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1) ?

- It is the intersection of the lines $b = -x_0m + y_0$ and $b = -x_1m + y_1$

Hough Transform for Line Fitting



Equation of a line?
 $y = mx + b$

How can we use this to find the most likely parameters (m, b) for the most prominent line in the image space?

- Let each edge point in image space *vote* for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Hough Transform for Line Fitting

Step 1. Quantize parameter space (m, c)

Step 2. Create accumulator array $A(m, c)$

Step 3. Set $A(m, c) = 0$ for all (m, c)

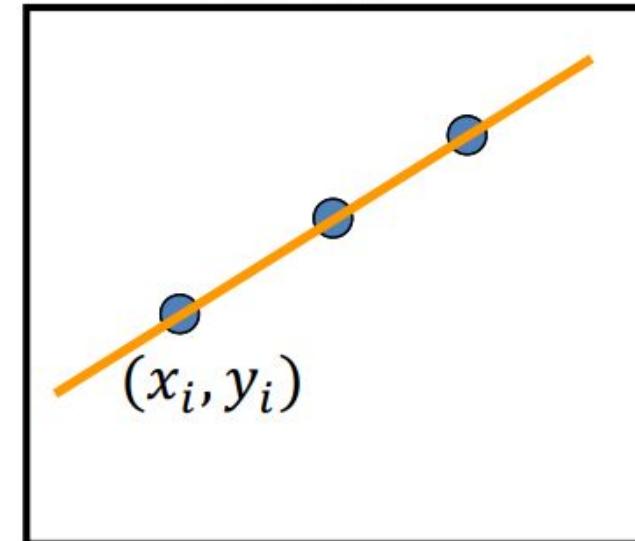
Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

Step 5. Find local maxima in $A(m, c)$

Image



	$A(m, c)$				
c	1	0	0	0	1
0	1	0	1	0	
1	1	3	1	1	
0	1	0	1	0	
1	0	0	0	1	

Hough Transform for Line Fitting

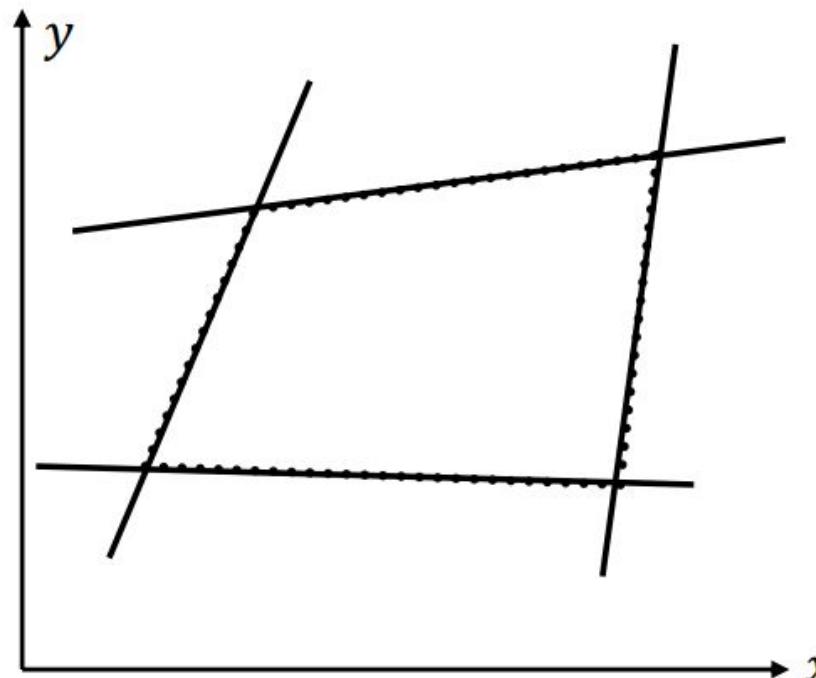
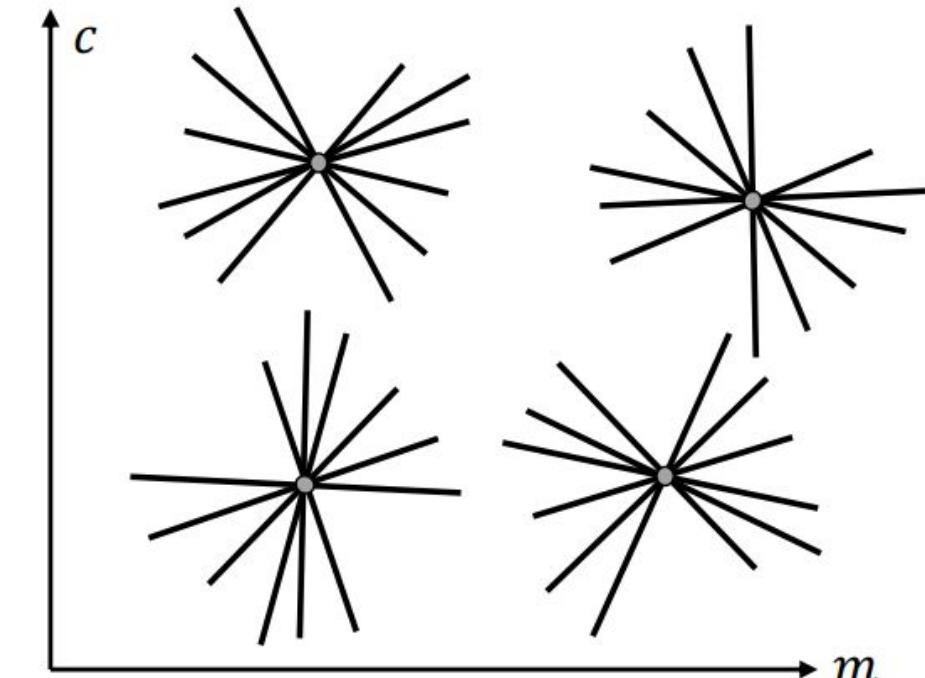


Image Space



Parameter Space



Hough Transform for Line Fitting

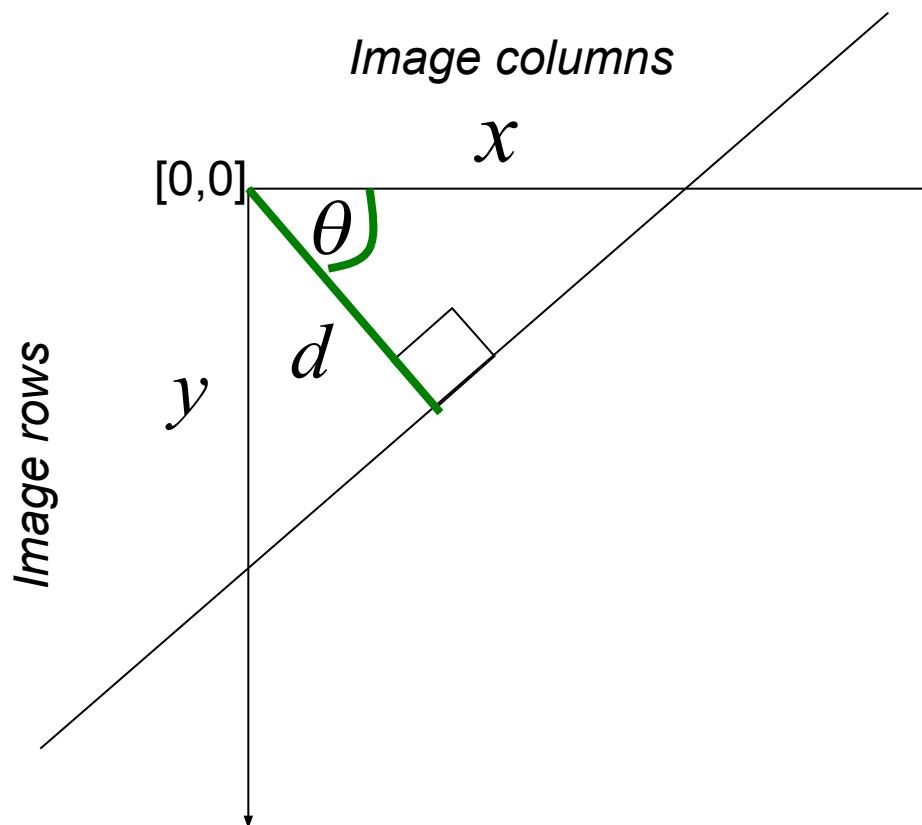
Issue: Slope of the line $-\infty \leq m \leq \infty$

- Large Accumulator
- More Memory and Computation

Solution: Use $x \sin \theta - y \cos \theta + \rho = 0$

- Orientation θ is finite: $0 \leq \theta < \pi$
- Distance ρ is finite

Polar Representation of Lines



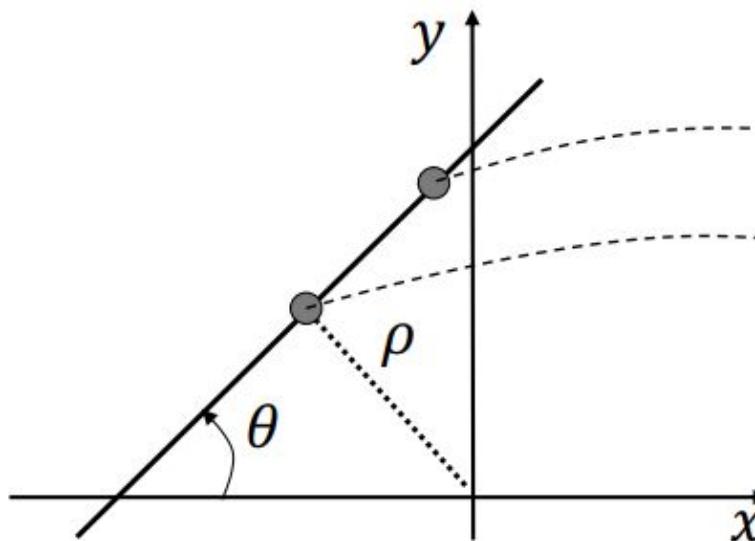
: perpendicular distance from line to origin

: angle the perpendicular makes with the x-axis

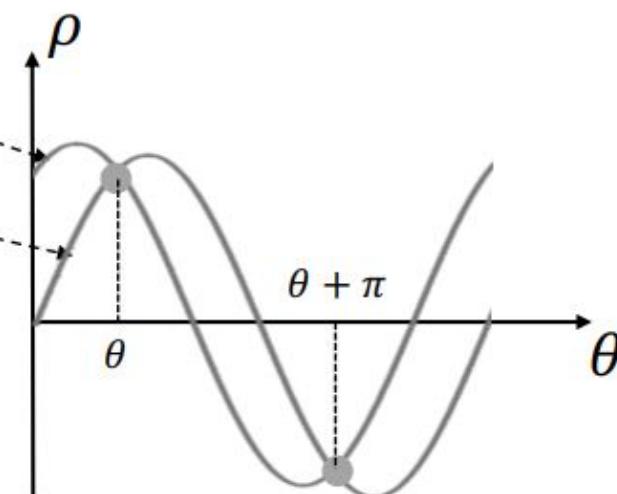
$$x \cos \theta - y \sin \theta = d$$

Alternate formulation using Polar Representation of Lines

Image Space



Parameter Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

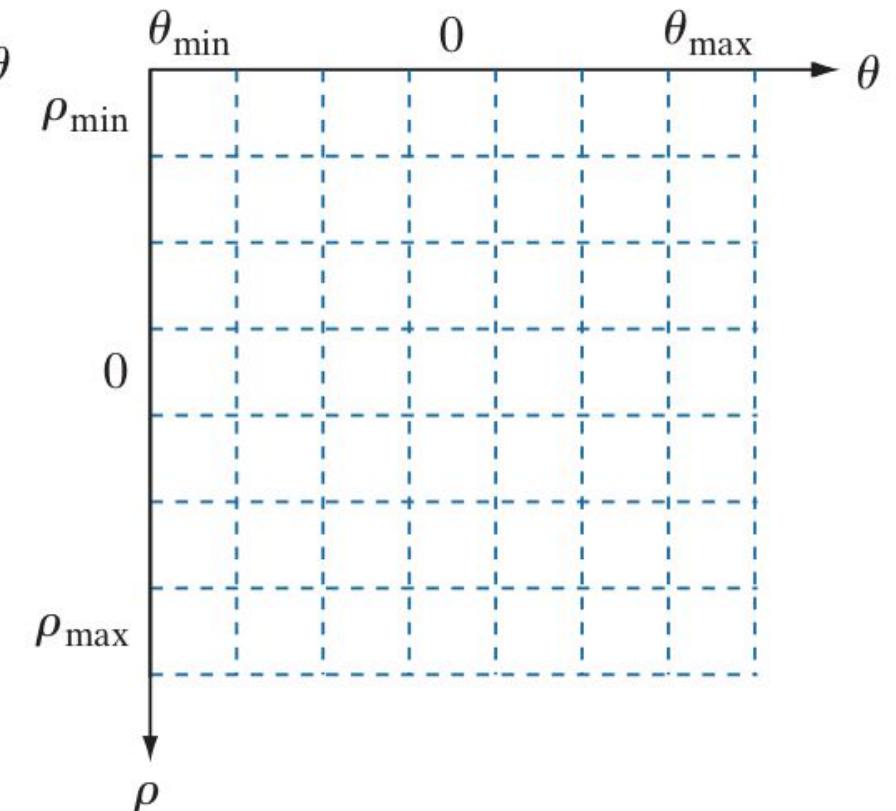
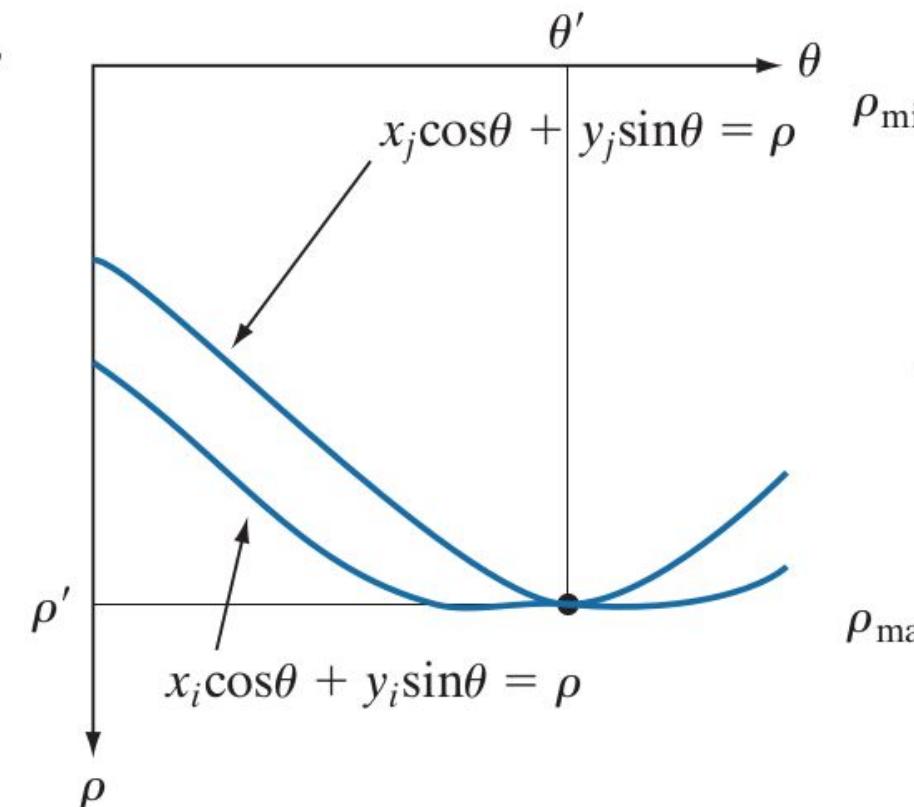
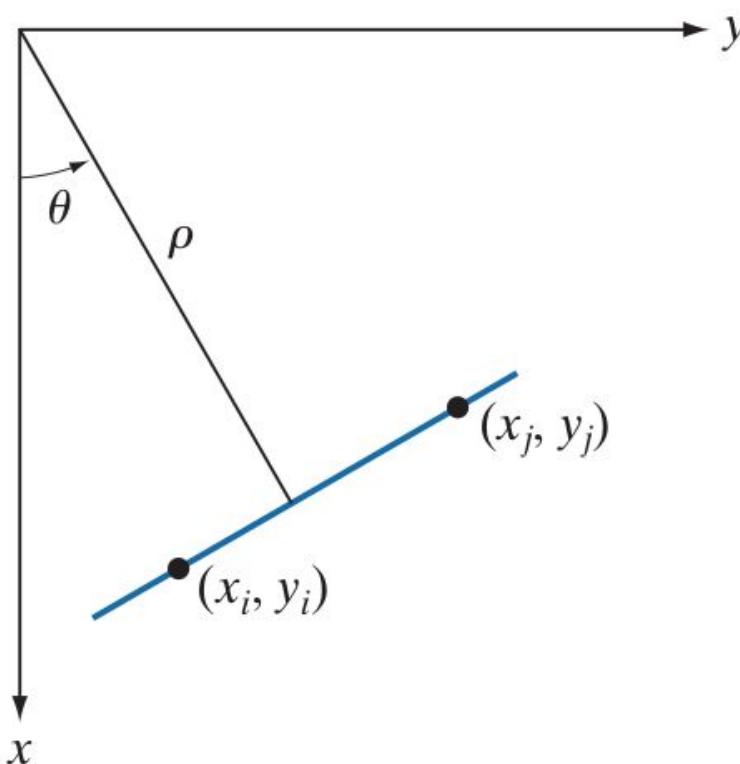
$$x \sin \theta - y \cos \theta + \rho = 0$$

ρ : perpendicular distance from line to origin

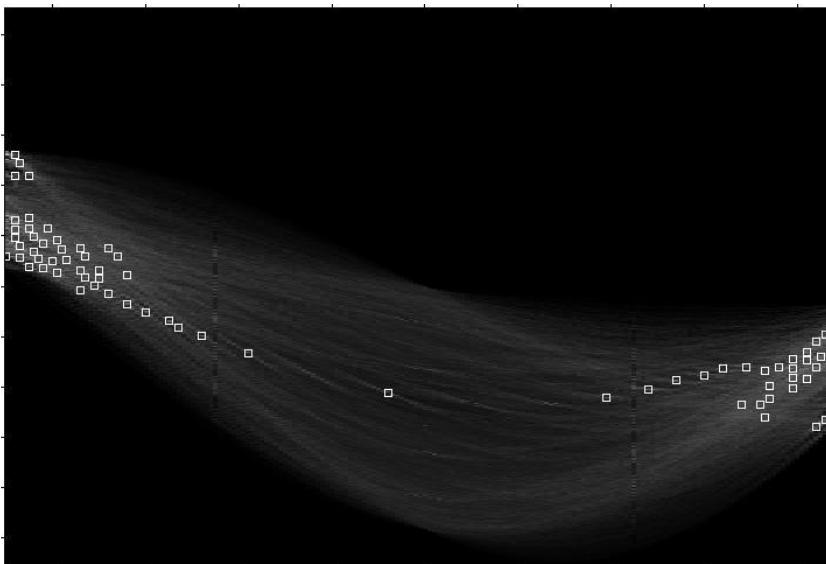
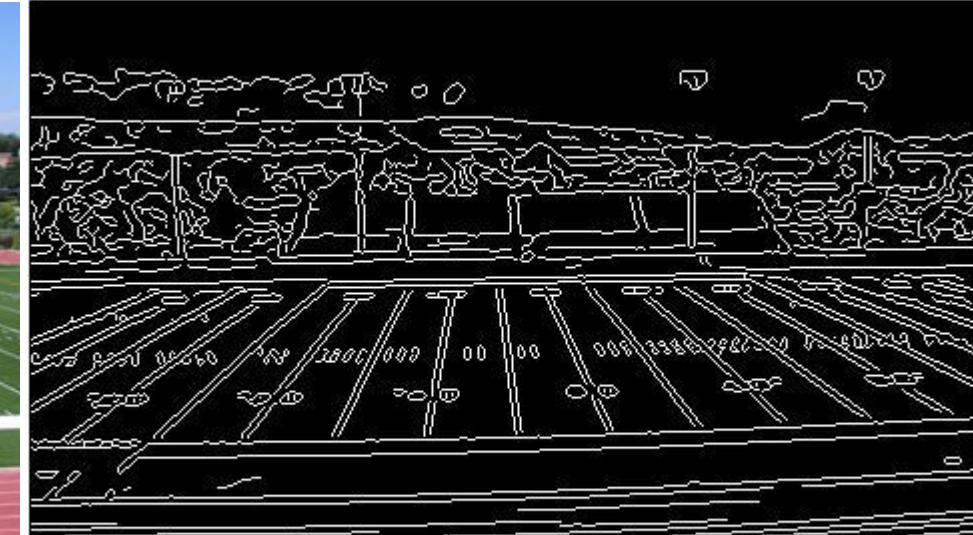
Θ : angle the perpendicular makes with the x-axis

For images: $0 \leq \theta < \pi$ and $|\rho| \leq \text{Image Diagonal}$

Alternate formulation using Polar Representation of Lines



Examples - Hough Transform





- **Readings:**
(1) Hough Transform - Page 737 - 742 - Digital Image Processing, 4th Ed, Rafael C. Gonzalez & Richard E. Woods
- **Next Class:**
Image Sharpening; Linear Filters; Edge Detection using Canny; Line detection using Hough transforms; [if possible]



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision
2023-24 First Semester, M.Tech (AIML)

Session #5: Edge Detection (Canny)

Topics

- Canny Edge Detection
- RANSAC
- Interest Point Detection (Starting Harry's Detector)

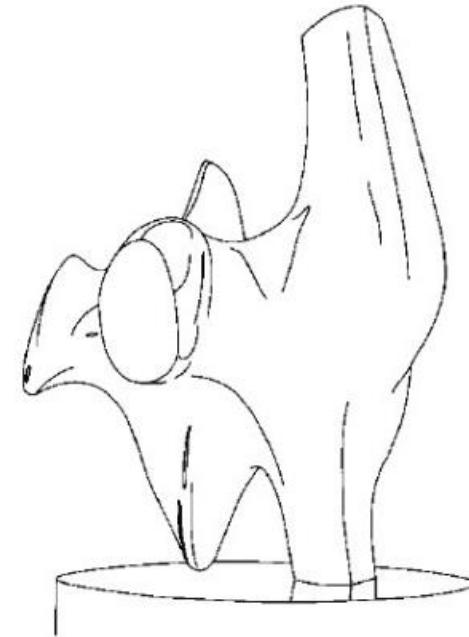
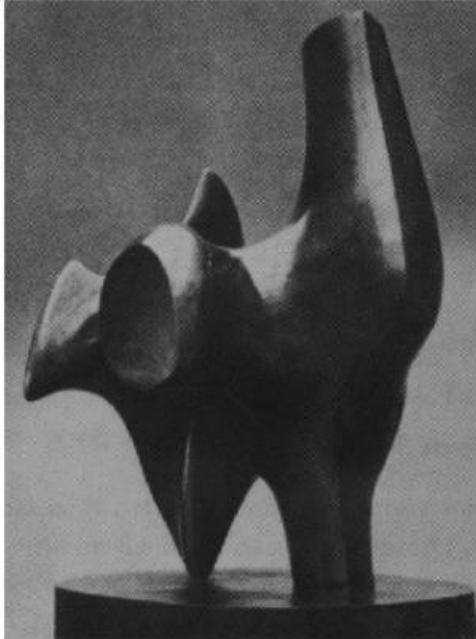
Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely;



Edge Detection

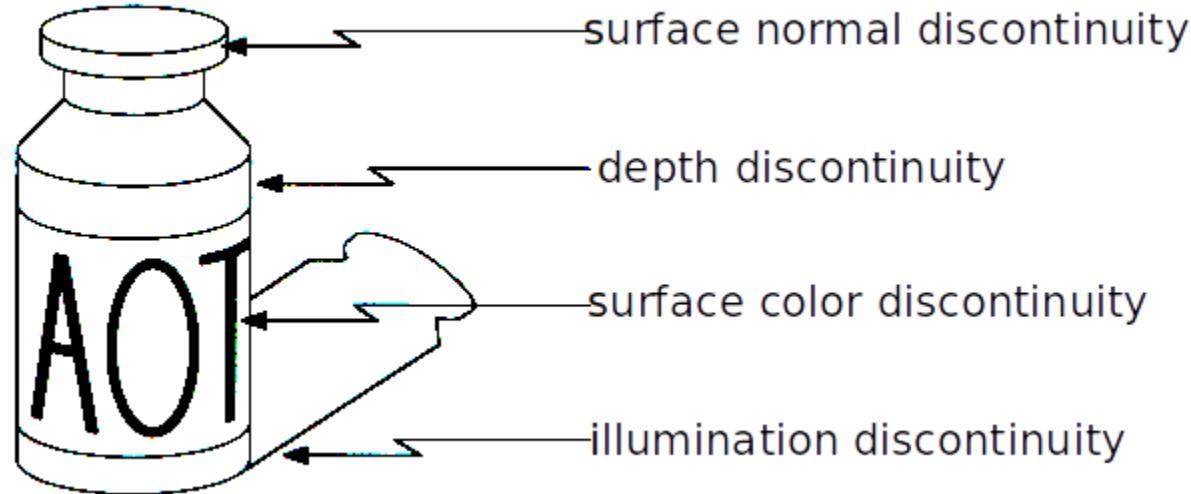
- Edges are pixels where *intensity (brightness)* changes abruptly
- Gradient's (along x and y direction) are used to describe the direction of the largest growth of the image function
 - Derivatives are approximated by differences
- Objectives of Edge Detection
 - **Good Detection.** Filter responds to edge, not noise
 - **Good Localization:** detected edge near true edge.
 - **Single Response:** only one point for each true edge point

Edge Detection



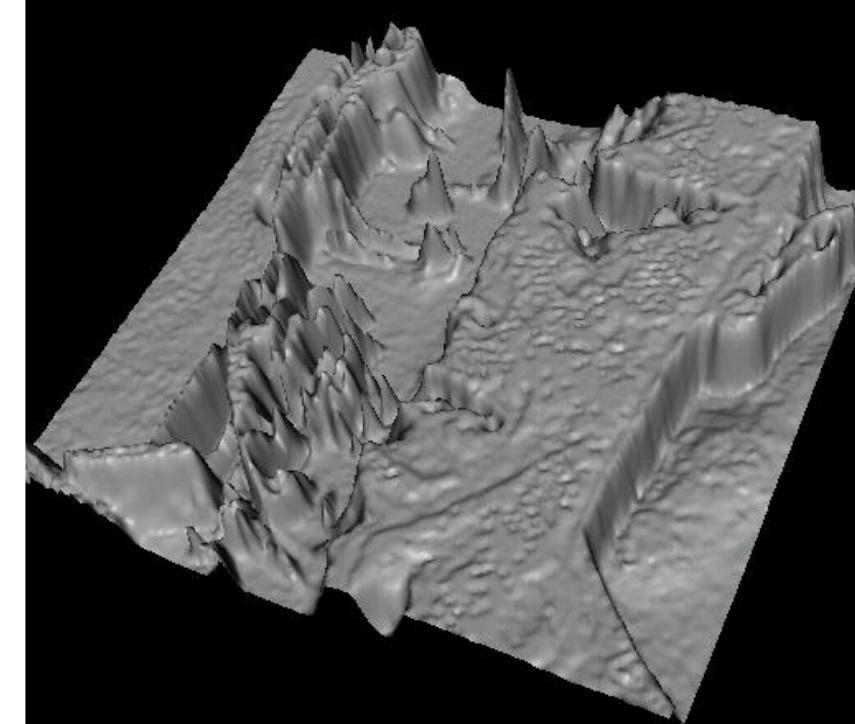
Task : Convert a 2D image into a set of curves (made of edges)

Edge Detection



Edges are caused by a variety of factors

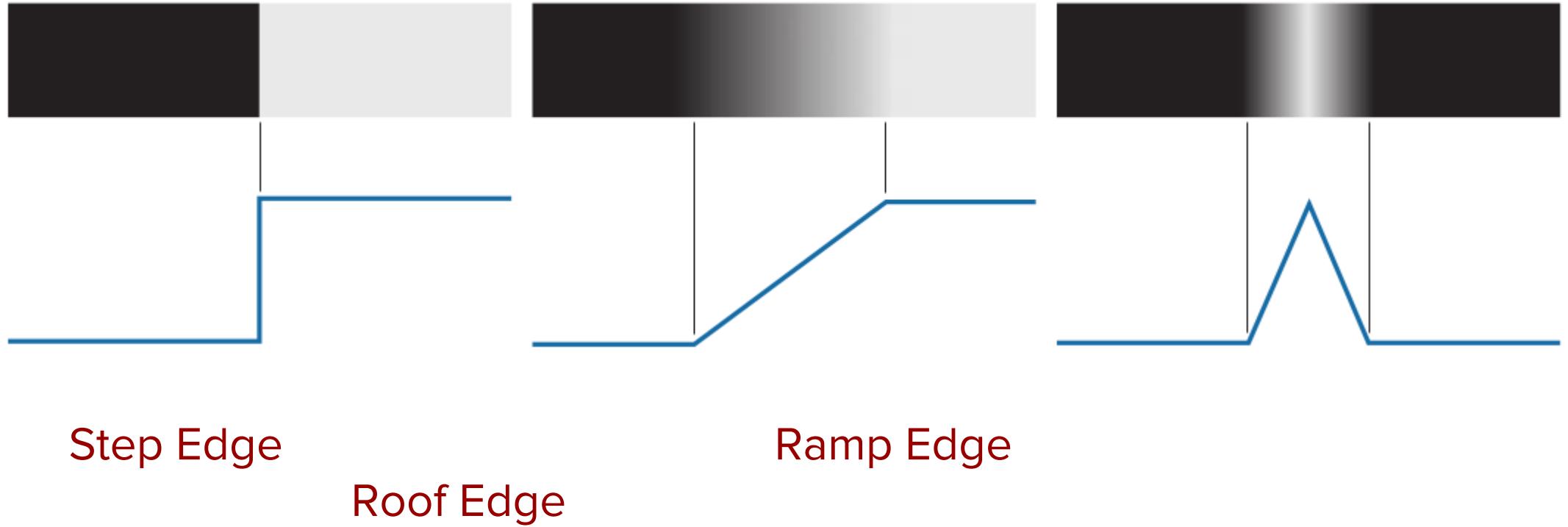
Edge Detection



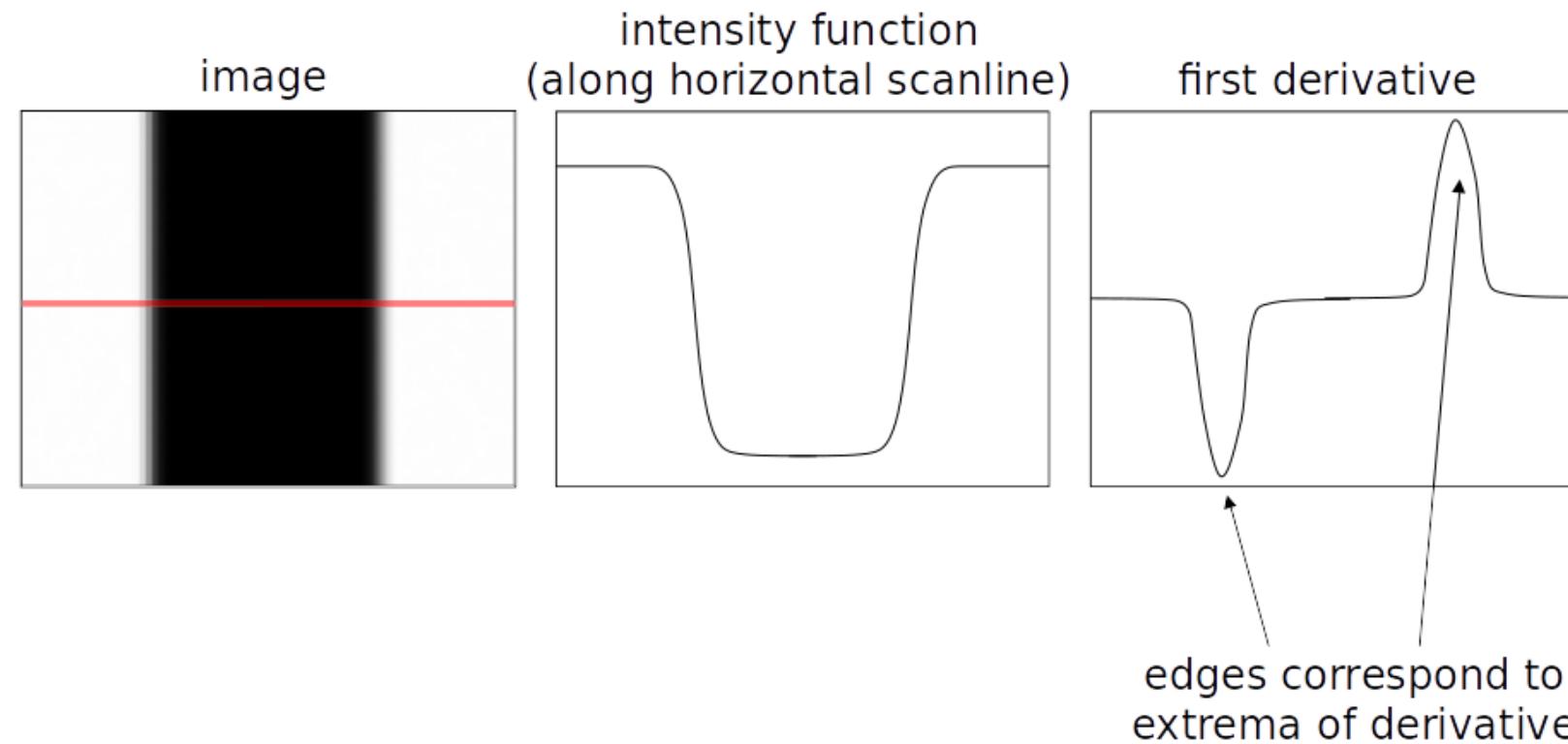
Treating images as functions, edges look like steep cliffs in the visualization



Recall: Edge Profiles



Recall: Edge Profiles



An edge is a place of rapid change in the image intensity function



Recall: Gradient and it's Computation

- How can we differentiate a *digital* image $F[x,y]$?
 - **Option 1:** reconstruct a continuous image, f , then compute the derivative
 - **Option 2:** take discrete derivative (finite difference)

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

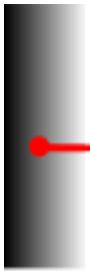
$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y)$$

Recall: Gradient's role in Detecting Edge

- The gradient of an image:

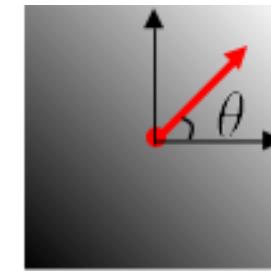
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

- The gradient points in the *direction of most rapid increase in intensity*


$$\nabla f = \left[\frac{\partial f}{\partial x}, 0 \right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y} \right]$$


$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

- The edge strength is given by the gradient magnitude: $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$
- The gradient direction is given by: $\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$



Steps in Canny's Edge Detection

1. Smooth the input image with a Gaussian filter.
2. Compute the gradient magnitude and angle images.
3. Apply non-maxima suppression to the gradient magnitude image.
4. Use double thresholding and connectivity analysis to detect and link edges.

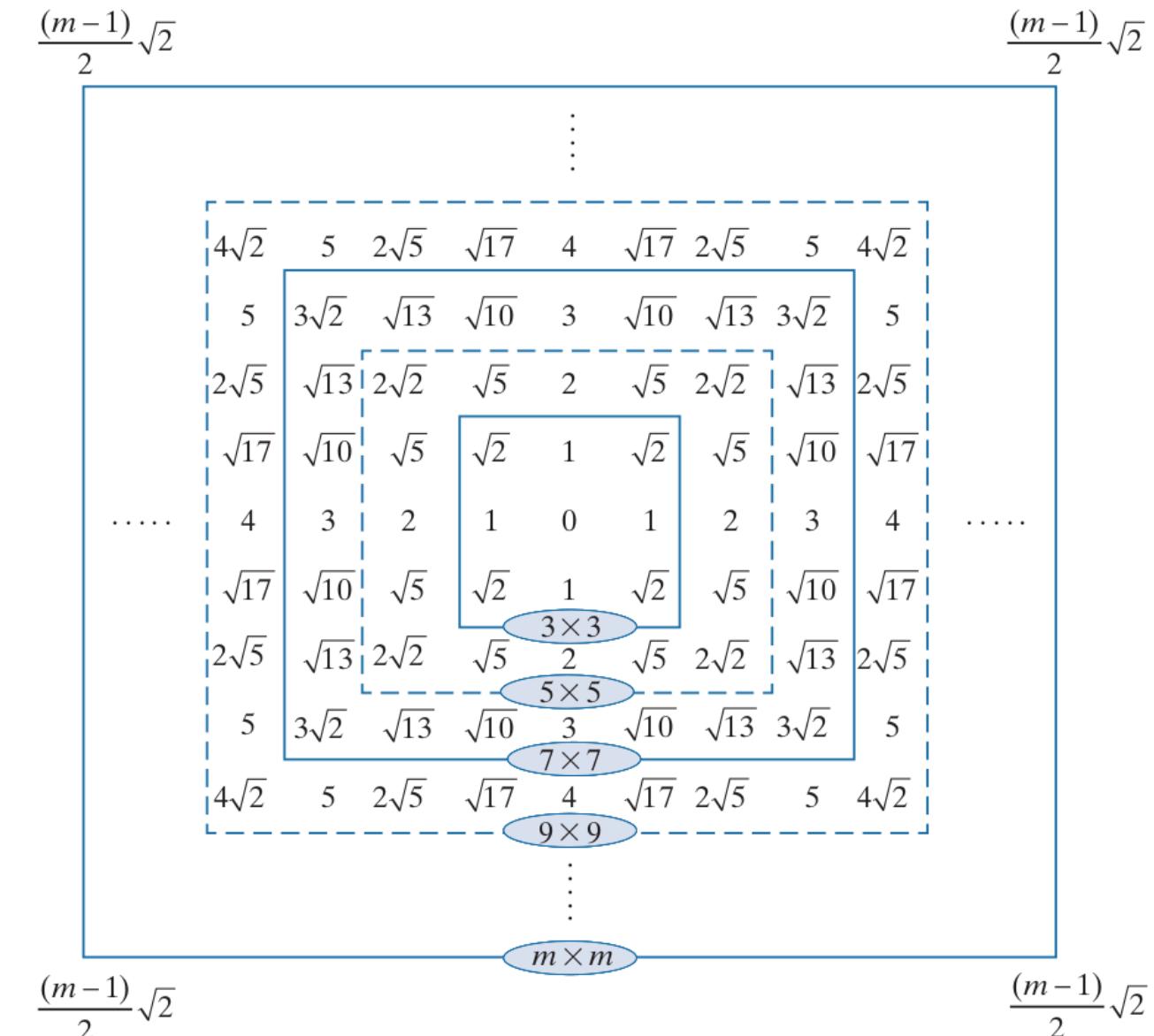
Recall: Gaussian Filters

Let s and r be position of a cell in a filter w.r.t the origin; r represent the distance of the cell from the center

$$r = [s^2 + t^2]^{1/2}$$

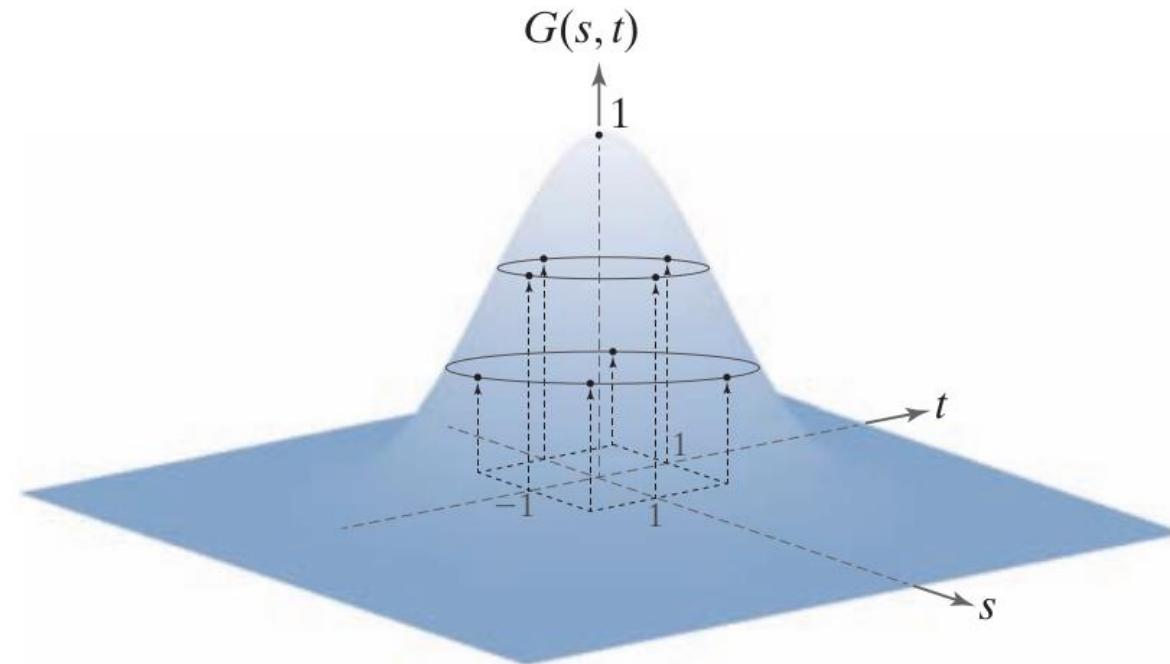
Form of Gaussian kernels is as below:

$$G(r) = K e^{-\frac{r^2}{2\sigma^2}}$$



Distances from the center for various sizes of square kernels.

Recall: Gaussian Filters



$$\frac{1}{4.8976} \times$$

0.3679	0.6065	0.3679
0.6065	1.0000	0.6065
0.3679	0.6065	0.3679

Resulting 3×3 kernel

Sampling a Gaussian function to obtain a discrete Gaussian kernel. The values shown are for $K = 1$ and $\sigma = 1$.

$$r = [s^2 + t^2]^{1/2}$$

$$G(r) = Ke^{-\frac{r^2}{2\sigma^2}}$$

Recall: Gaussian Filters



A test pattern of size 1024×1024



lowpass filtering the pattern with a Gaussian kernel of size 21×21 , with $\sigma = 3.5$, $K=1$



Result of using a kernel of size 43×43 , with $\sigma = 7$, $K=1$

Recall: Gaussian Filters



Gaussian kernels of size 43×43 ,
with $\sigma = 7$



Kernel of 85×85 , with $\sigma = 7$



Difference image

Recall: Gaussian Filters - Example Usage



a b c

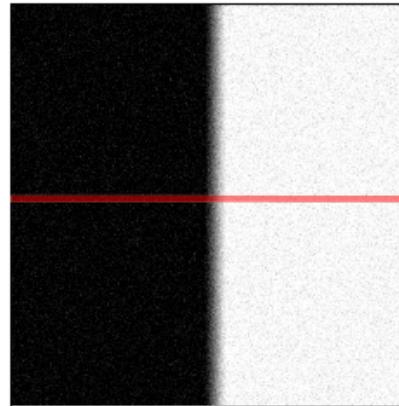
A 2566×2758 Hubble Telescope image of the Hickson Compact Group

Result of lowpass filtering with a Gaussian kernel.

Result of thresholding the filtered image (intensities were scaled to the range $[0, 1]$)

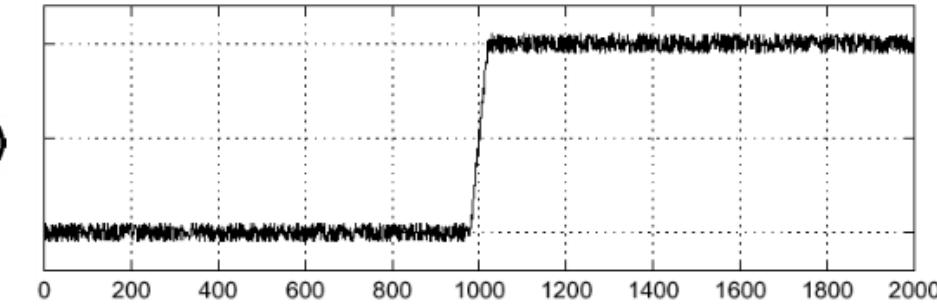
Effect of Noise

- Where is the edge?

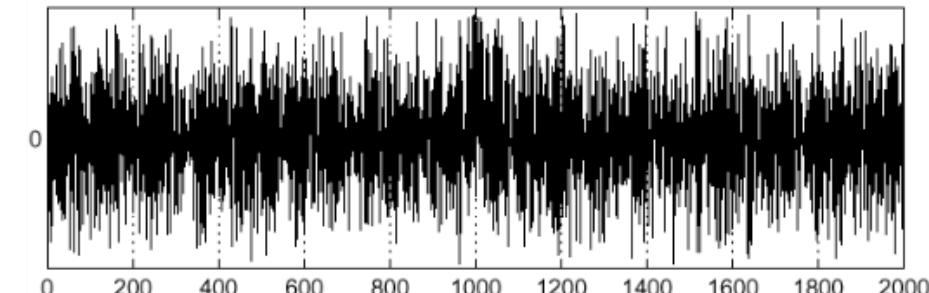


Noisy input image

$$f(x)$$

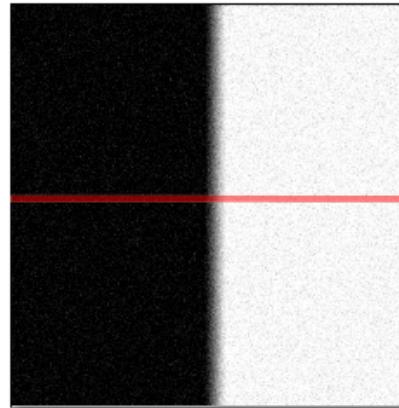


$$\frac{d}{dx}f(x)$$



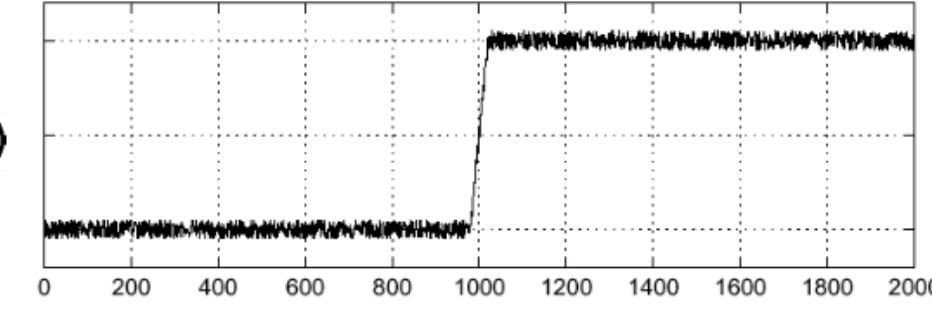
Effect of Noise

- Where is the edge?

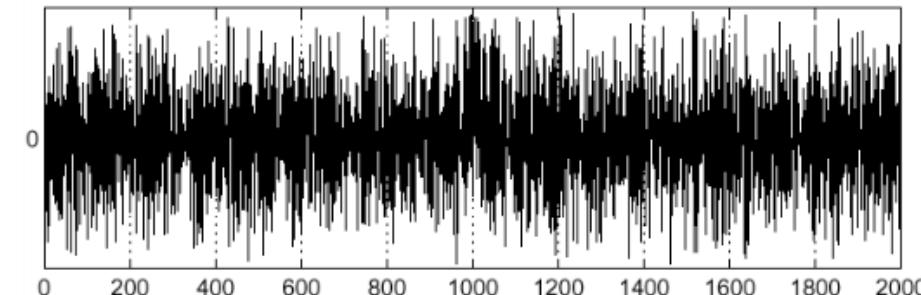


Noisy input image

$$f(x)$$



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



Step -1: Smooth the image with gaussian low-pass filter



Step -2: Computing Gradients (Magnitude and Direction)

Compute partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at every pixel location in the image

$$\nabla f(x, y) \equiv \text{grad}[f(x, y)] \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

Points in the direction of maximum rate of change of f at (x, y)

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right]$$

Direction of gradient vector at (x, y)

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

Value of the rate of change in the direction of the gradient vector at (x, y)

Step -2: Computing Gradients (Magnitude and Direction)

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

A 3×3 region of an image (the z's are intensity values)

To Do: Compute gradient at z_5

Prewitt's operators

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

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

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Step -2: Computing Gradients (Magnitude and Direction)

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Sobel's operators

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

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

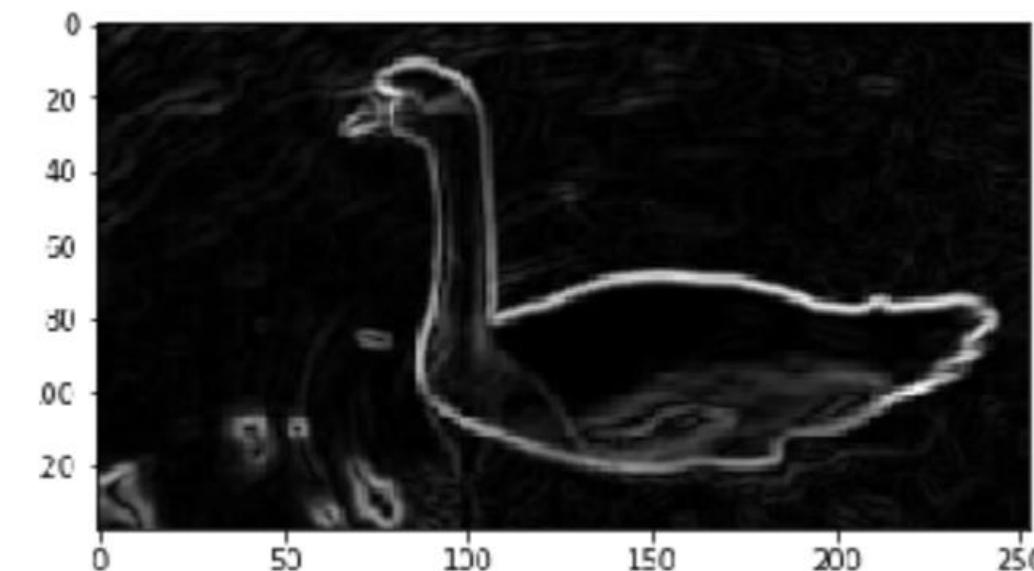
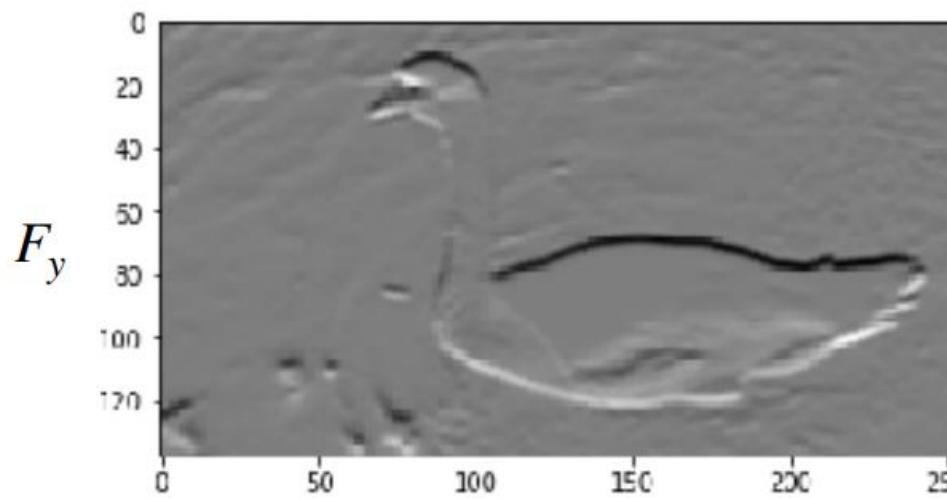
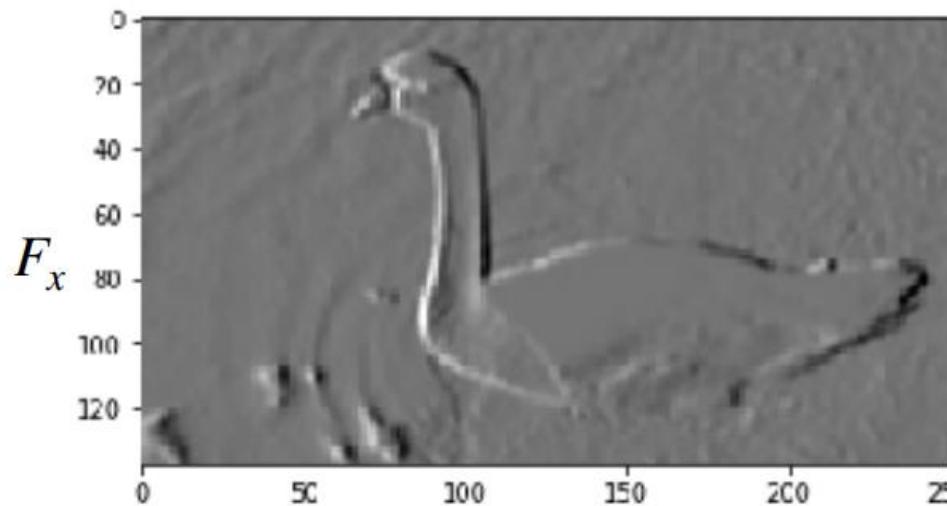
A 3×3 region of an image (the z's are intensity values)

To Do: Compute gradient at z_5

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

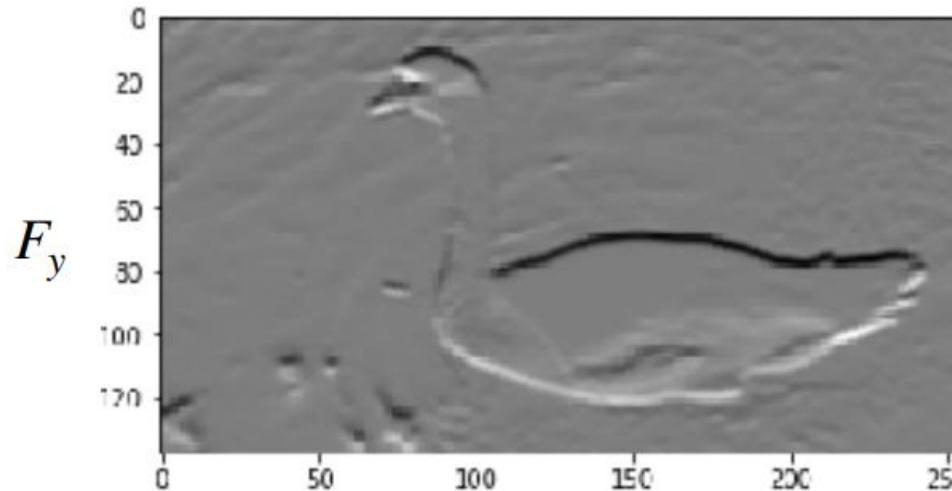
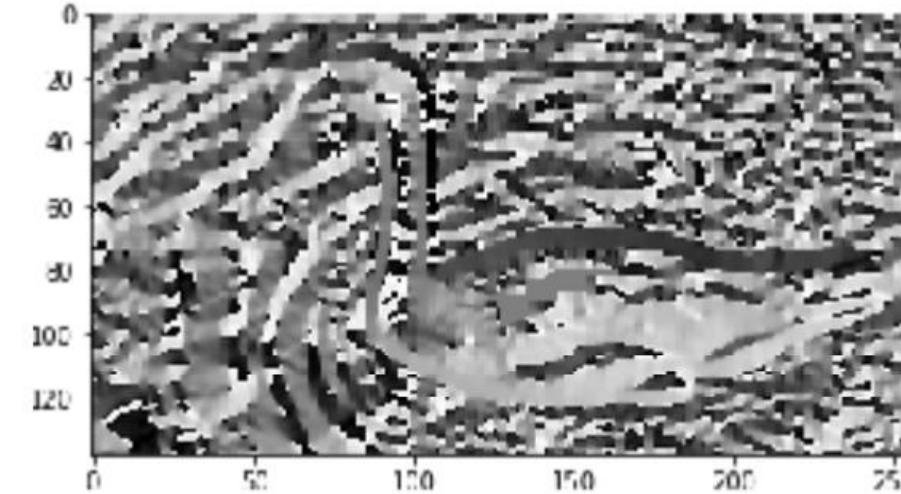
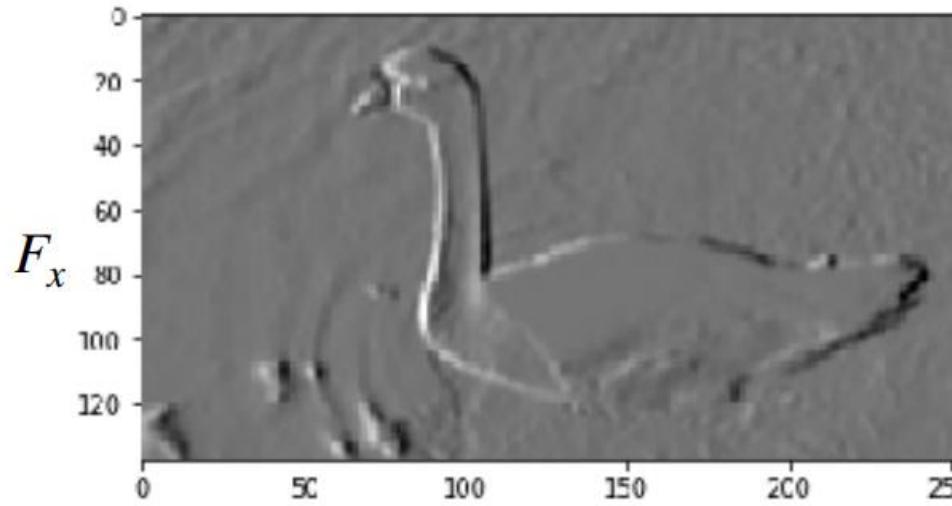
$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

Step -2: Computing Gradients (Magnitude and Direction)



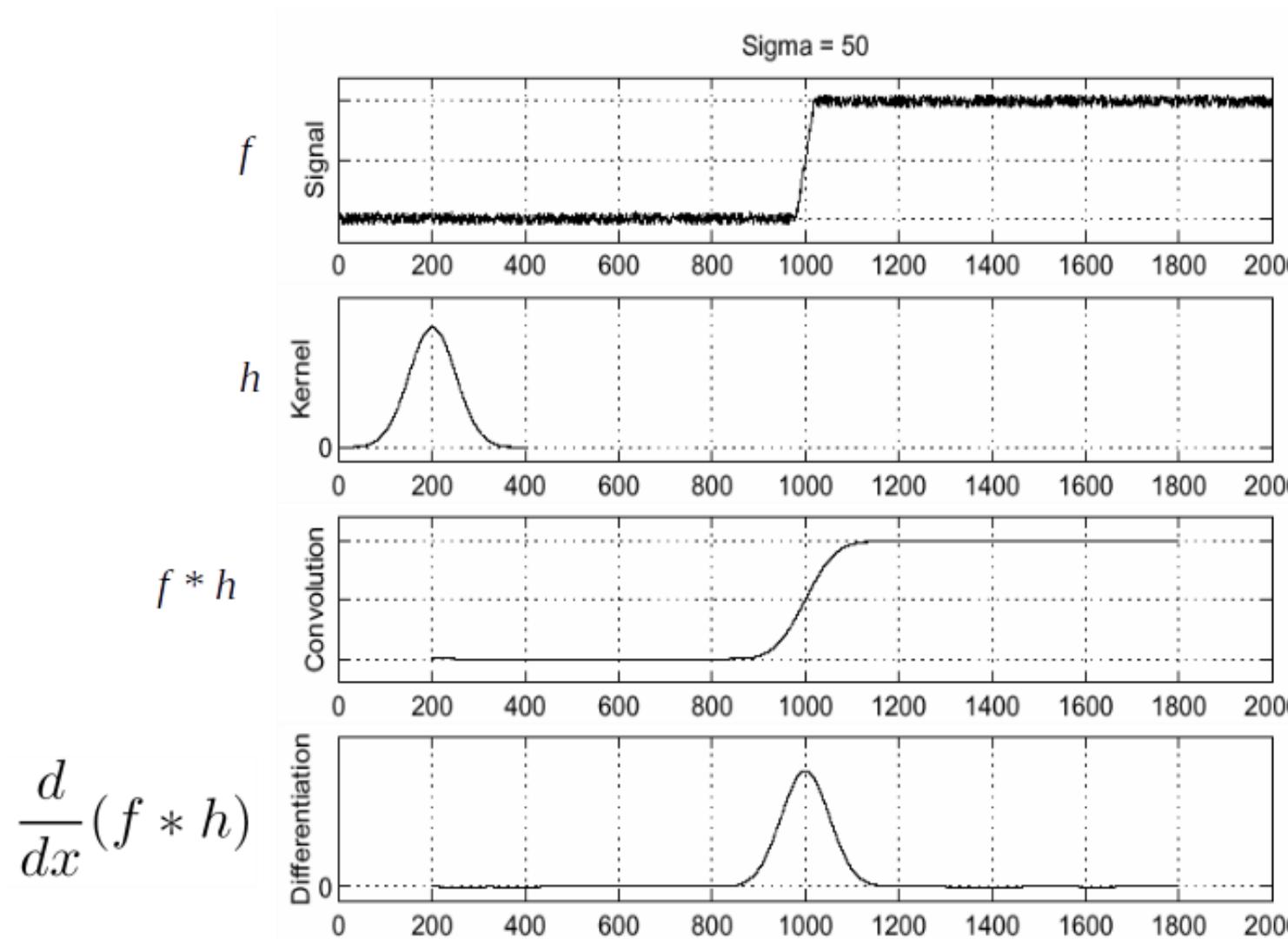
$$G = \sqrt{(F_x^2 + F_y^2)}$$

Step -2: Computing Gradients (Magnitude and Direction)

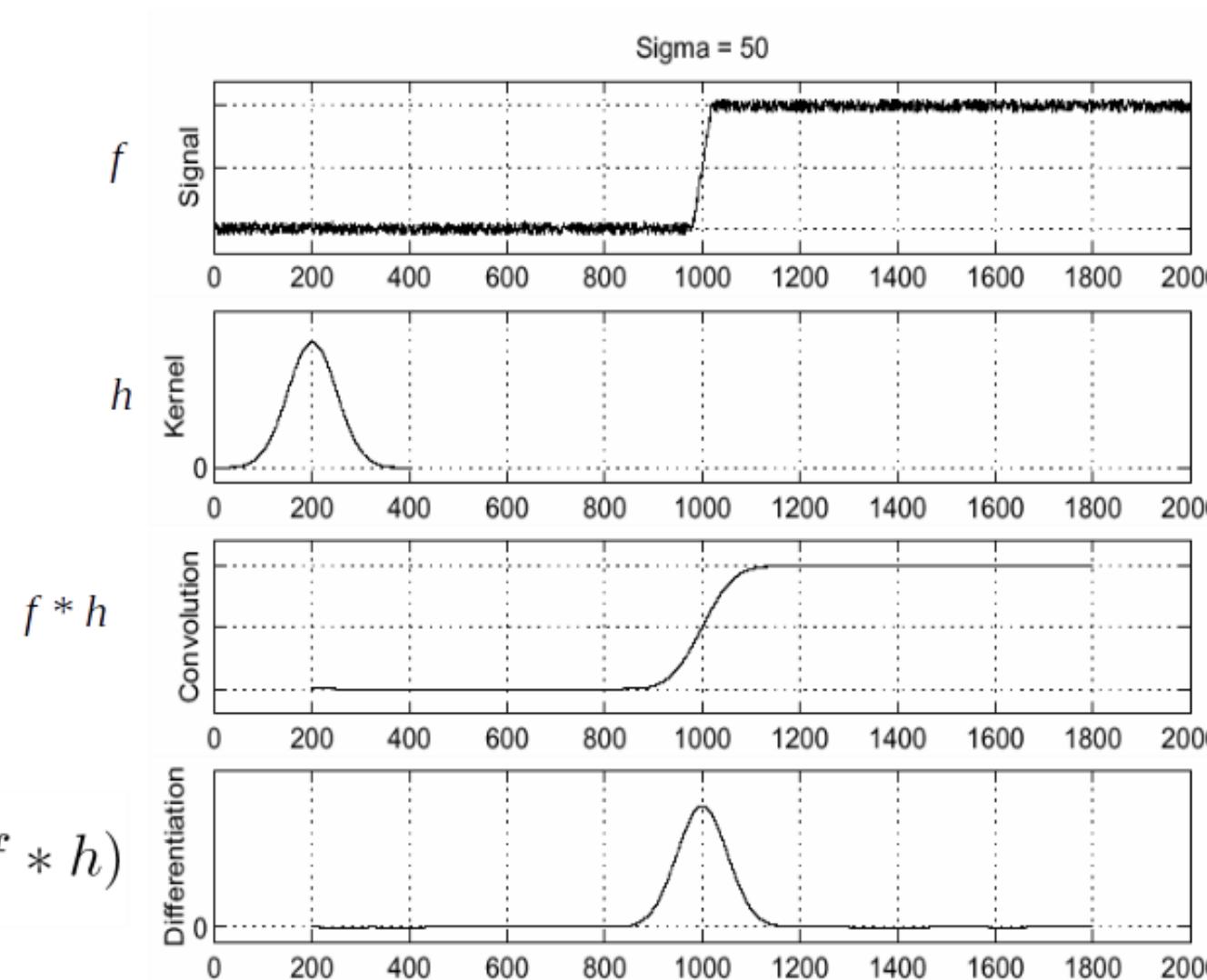


$$\theta = \tan^{-1} \left(\frac{F_y}{F_x} \right)$$

(Aside) Step -2: Efficient Computation of Image Gradients



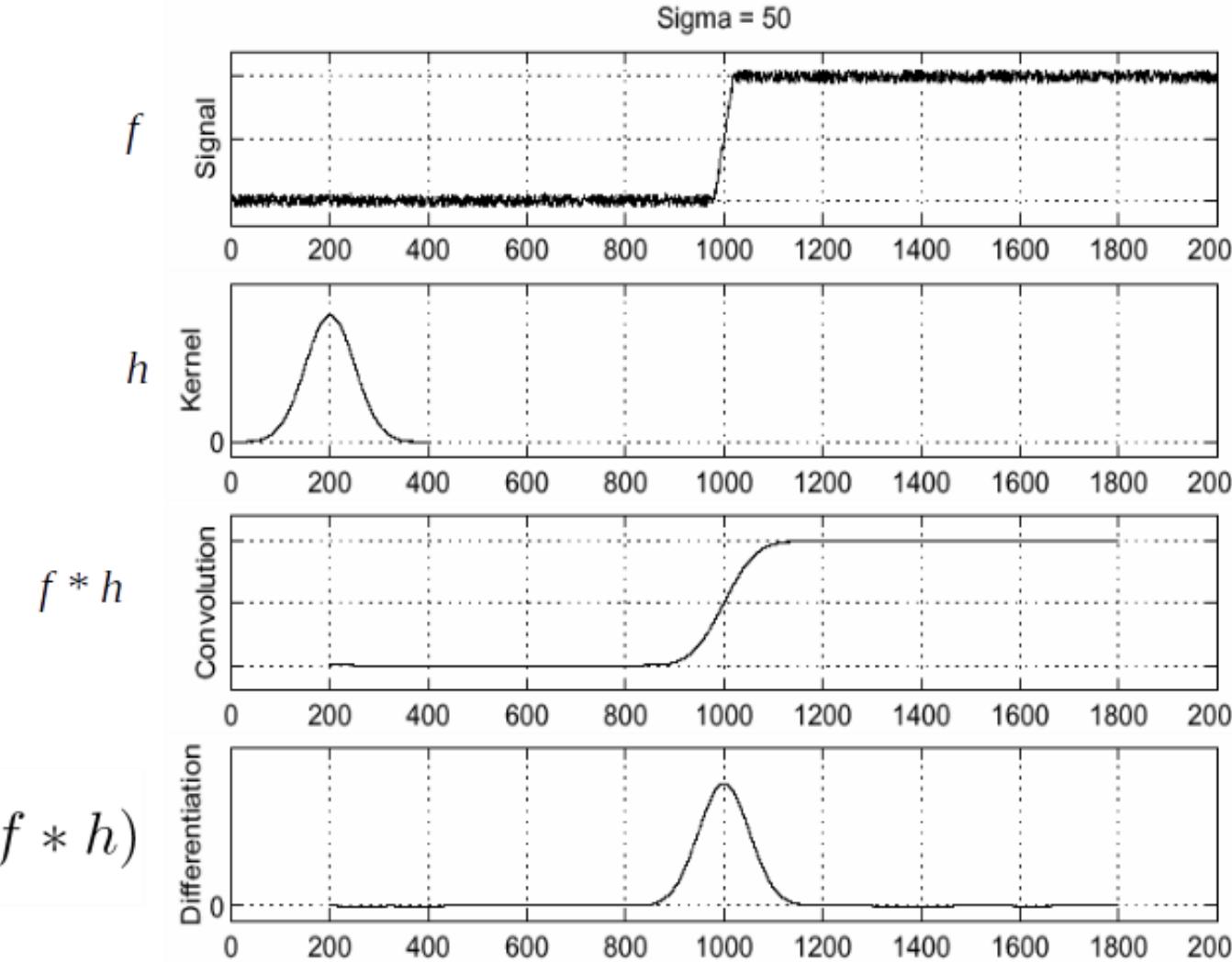
(Aside) Step -2: Efficient Computation of Image Gradients



Objective is to compute

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

(Aside) Step -2: Efficient Computation of Image Gradients



Objective is to compute $\frac{d}{dx}(f * h)$

Using associative property of convolution

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

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

$$G'_\sigma(x) = \frac{d}{dx}G_\sigma(x) = -\frac{1}{\sigma} \left(\frac{x}{\sigma}\right) G_\sigma(x)$$

This saves us one operation



Ex: Step -1 and Step-2



Original Image



Ex: Step -1 and Step-2

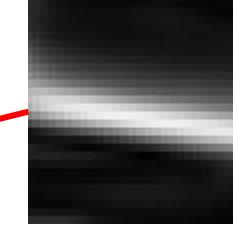


Smoothed Gradient Magnitude



Original Image

Ex: Step -1 and Step-2



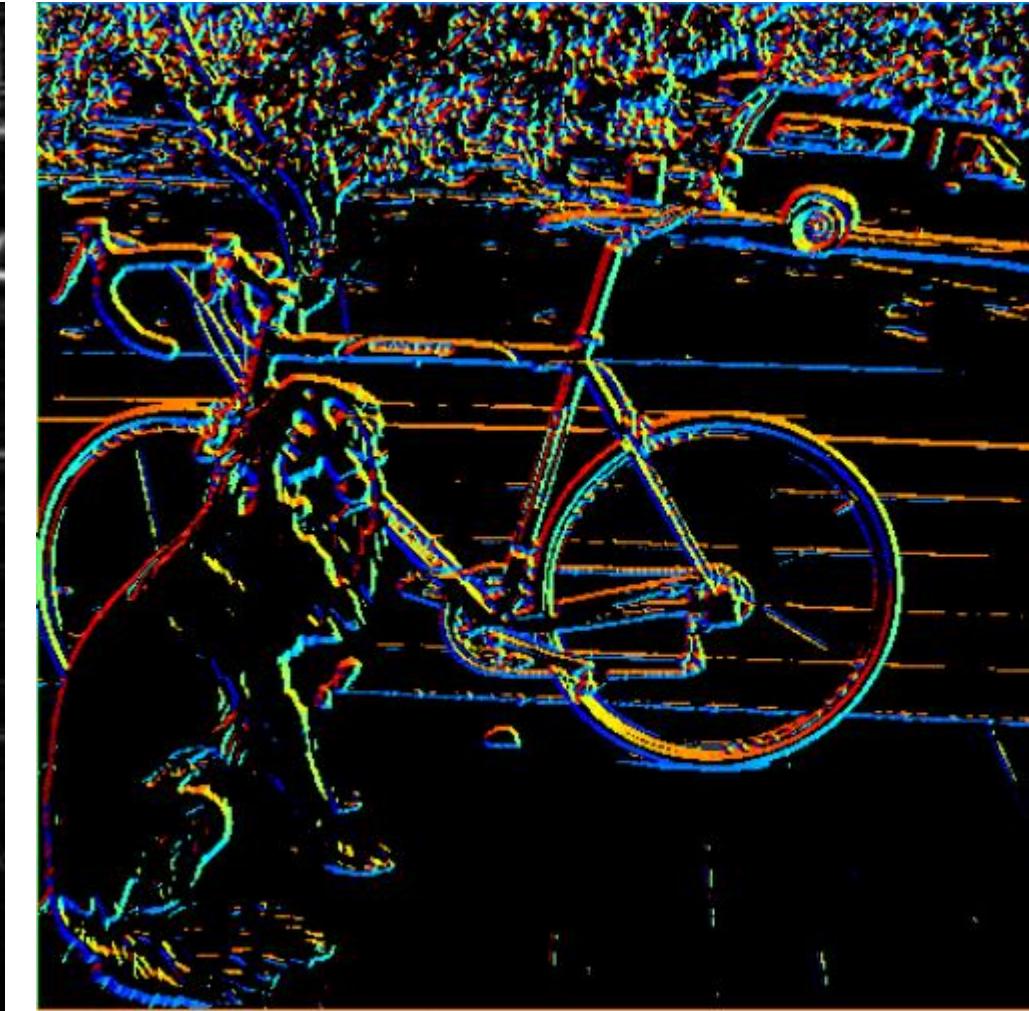
Where is the edge?

Smoothed Gradient Magnitude

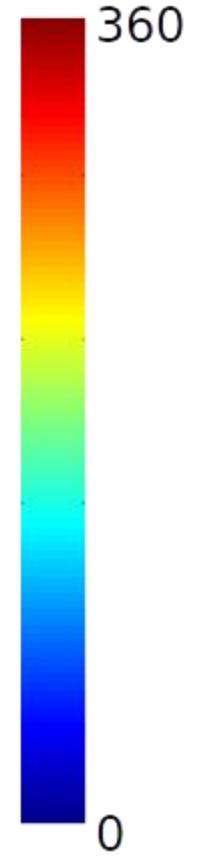
Ex: Step -1 and Step-2



Smoothed Gradient Magnitude

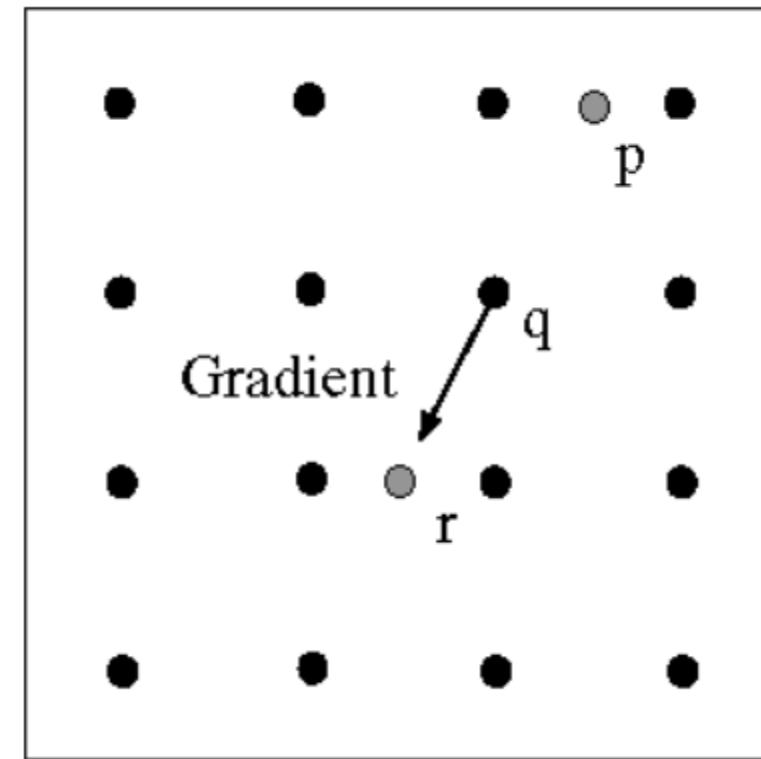
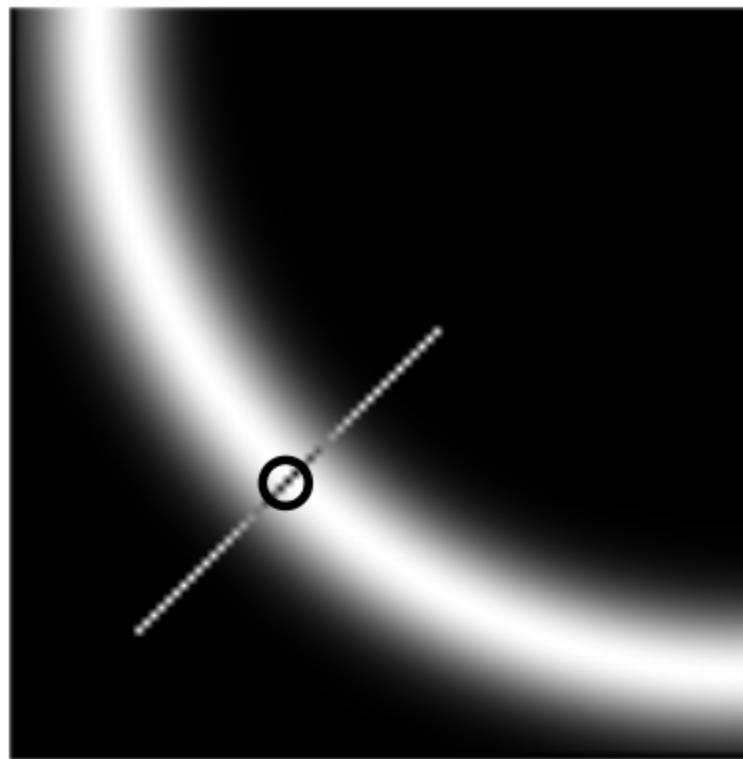


Can we make use of angle to locate the edge better?



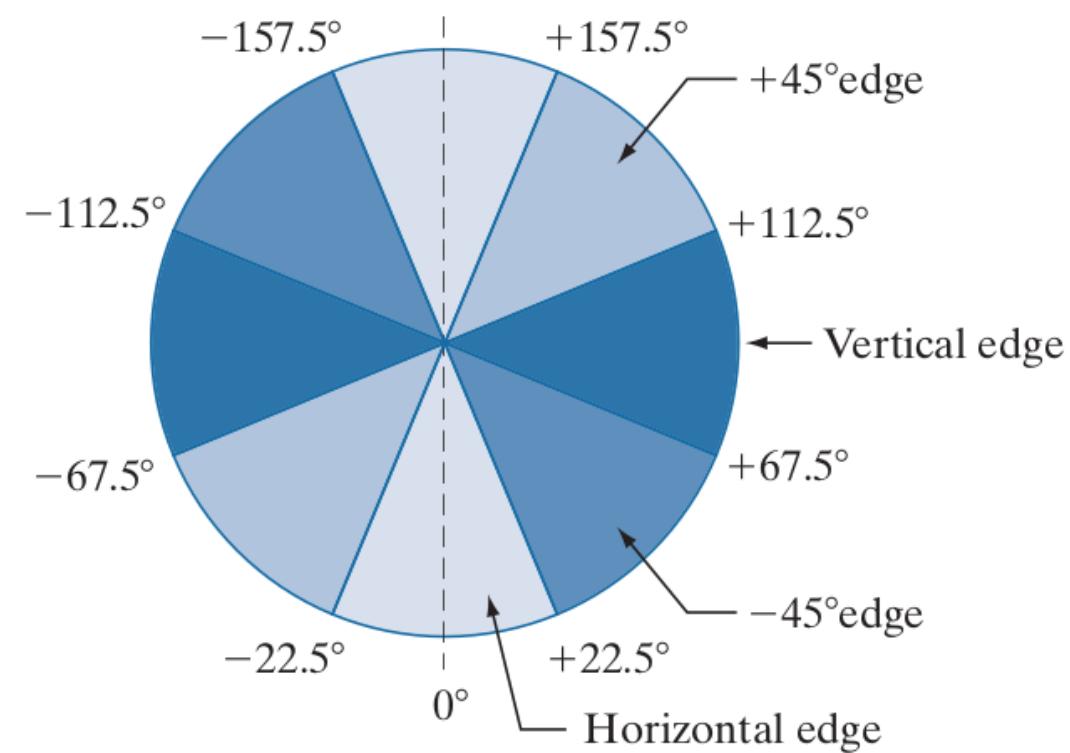
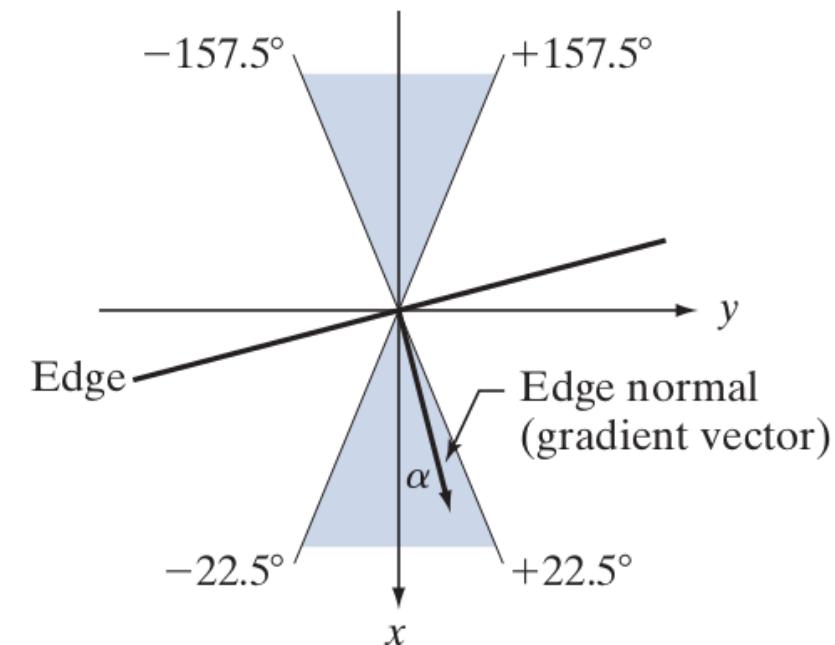
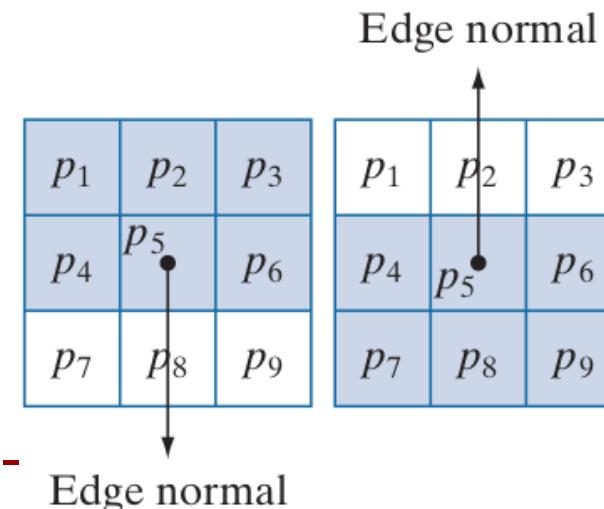
Step -3: Non-Maxima Suppression

Check if pixel is local maximum along gradient direction



Step -3: Non-Maxima Suppression - Computation

Let d_1, d_2, d_3 and d_4 refers to horizontal, vertical, $+45^\circ$ and -45° edges.





Step -3: Non-Maxima Suppression - Computation

Let d_1, d_2, d_3 and d_4 refers to horizontal, vertical, $+45^\circ$ and -45° edges.

Let α be the matrix if gradient orientations

Consider a 3×3 neighborhood around $\alpha(x,y)$
Non-Maxima Scheme around $\alpha(x,y)$ is as below:

1. Find the direction d_k that is closest to $\alpha(x,y)$.
2. Let K denote the value of $\|\nabla f_s\|$ at (x,y) . If K is less than the value of $\|\nabla f_s\|$ at one or both of the neighbors of point (x,y) along d_k , let $g_N(x,y) = 0$ (suppression); otherwise, let $g_N(x,y) = K$.

Repeat (1) and (2) for all values of x and y to get non-maxima suppressed image which is of the same size as original image.

Step -3: Non-Maxima Suppression - Computation



Before Non-Maxima Suppression



After Non-Maxima Suppression

Step -4: Double Thresholding and Edge Linking

Still some noise; Only need strong edges;

Called as *Hysteresis thresholding*

Use two thresholds T_L and T_H which results in two images;

Choose the ratio of high to low threshold in the range of 2:1 to 3:1.



Step -4: Double Thresholding and Edge Linking

<= lower threshold

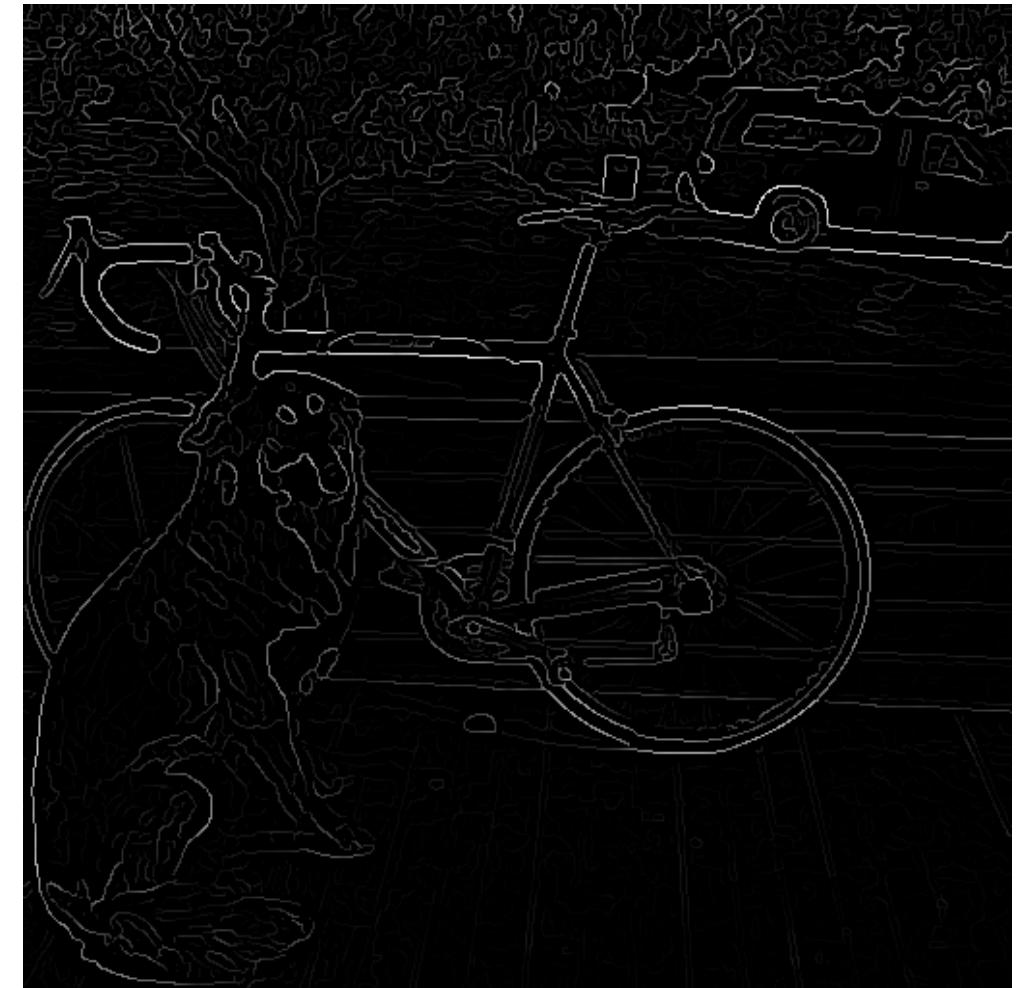
irrelevant = 0

lower threshold
< intensity <
upper threshold

weak = low threshold

>= upper threshold

strong= 255



Step -4: Double Thresholding and Edge Linking

<= lower threshold

irrelevant = 0

lower threshold
< intensity <
upper threshold

weak = low
threshold

>=

upper threshold

strong= 255

$$g_{NH}(x, y) = g_N(x, y) \geq T_H$$

$$g_{NL}(x, y) = g_N(x, y) \geq T_L$$

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y)$$





Step -4: Double Thresholding and Edge Linking

<= lower threshold	lower threshold < intensity < upper threshold	>= upper threshold
irrelevant = 0	weak = low threshold	strong= 255

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad \text{Strong Edge Pixels & Valid Edge Pixels}$$

$$g_{NL}(x, y) = g_N(x, y) \geq T_L$$

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y)$$

Weak Edge Pixels & we need to identify valid edge pixels from this set.



Step -4: Double Thresholding and Edge Linking

<= lower threshold	lower threshold < intensity < upper threshold	>= upper threshold
irrelevant = 0	weak = low threshold	strong= 255

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad \text{Strong Edge Pixels & Valid Edge Pixels}$$

$$g_{NL}(x, y) = g_N(x, y) \geq T_L$$

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y)$$

Weak Edge Pixels & we need to identify valid edge pixels from this set.



Step -4: Double Thresholding and Edge Linking

Approach to Link Edges:

- Strong edges are edges!
- Weak edges are edges iff they connect to strong
- Look in some neighborhood (usually 8 closest)



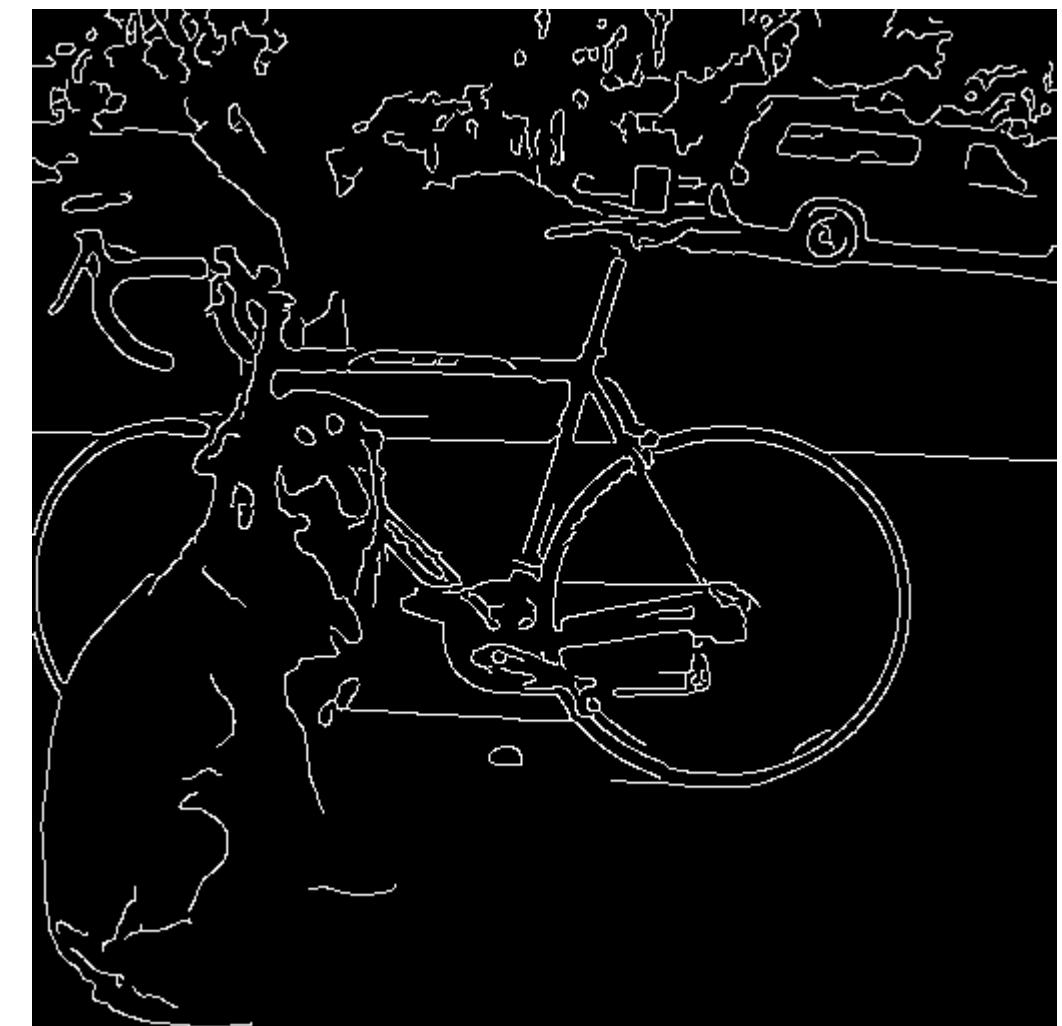
Step -4: Double Thresholding and Edge Linking

- (a) Locate the next unvisited edge pixel, p , in $g_{NH}(x, y)$.
- (b) Mark as valid edge pixels all the weak pixels in $g_{NL}(x, y)$ that are connected to p using, say, 8-connectivity.
- (c) If all nonzero pixels in $g_{NH}(x, y)$ have been visited go to Step (d). Else, return to Step (a).
- (d) Set to zero all pixels in $g_{NL}(x, y)$ that were not marked as valid edge pixels.

Step -4: Double Thresholding and Edge Linking



Before Edge Linking



After Edge Linking

Step -4: Double Thresholding and Edge Linking

Original image



Strong edges only



gap is gone

Strong + connected weak edges



Weak edges

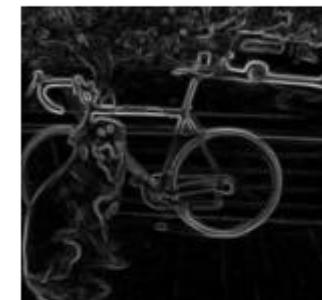


Summary of Canny Edge Detector

(1) Filter image with derivative of Gaussian



(2) Find magnitude and orientation of gradient



(3) Non-maximum suppression



(4) Linking and thresholding (hysteresis):

Define two thresholds: low and high

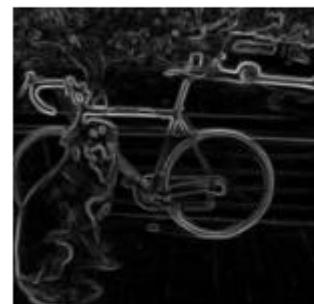
Use the high threshold to start edge curves and
the low threshold to continue them





Summary of Canny Edge Detector

J. Canny, **A Computational Approach To Edge Detection**,
IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.



Our first computer vision pipeline!
Still a widely used edge detector in computer vision





- Readings:
Canny's edge detector- Page 729 - 735 - Digital Image Processing, 4th Ed, Rafael C. Gonzalez & Richard E. Woods



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Vision

2023-24 First Semester, M.Tech (AIML)

Session #6: Harris Corner Detector, HoG & Applications

Topics

- Histogram of Oriented Gradients
- Local Features & Harris Corner detector
- Exam Preparation - Discussion & Advise
- Demonstration of HoG and Harris with a Classification Example.

Histogram of Oriented Gradients

Materials Adopted from CMSC 426 - COMPUTER VISION;
<https://www.cs.umd.edu/class/spring2021/cmsc426-0201/>

Histogram of Oriented Gradients (HOG)

Introduction

- Global features
- Sliding window
- Image subsampled to multiple sizes
- Normalized histograms

Steps

- Find horizontal and vertical gradients
- Gradient Magnitude and orientations
- Use a patch of 64×128
- Divide the image into blocks of 8×8 cells
- Slide over 2×2 block cells
- Quantize the gradient orientation into 9 bins by gradient magnitude
- Concatenate histograms into a feature of : $15 \times 7 \times 4 \times 9 = 3780$ dimensions.

Find horizontal and vertical gradients

Gradient Magnitude and orientations

$$\nabla f(x, y) = [g_x \ g_y] = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \lim_{d \rightarrow 0} \frac{f(x + d) - f(x - d)}{2d}$$

Filter Masks:

-1	0	1	-1
			0
			1

Gradient Magnitude:

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

Gradient Orientation:

$$\theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$

Blocks and Cells

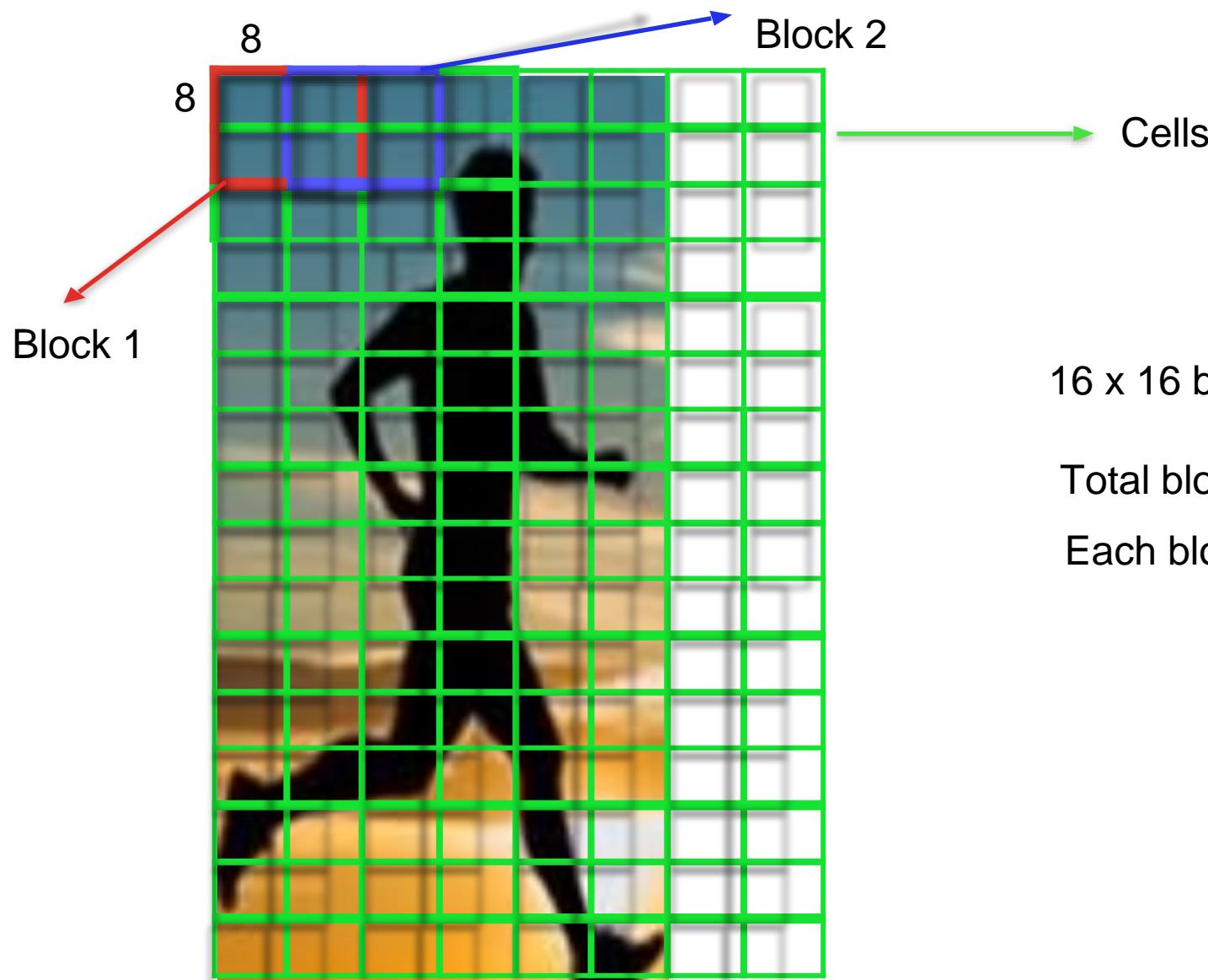
~~Use a patch of 64 x 128~~



128

64

Divide the image into blocks of 8×8 cells Slide over 2×2 block cells



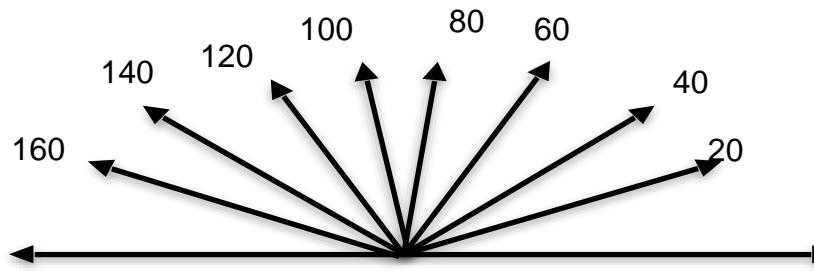
16 x 16 blocks with an overlap

Total blocks: 15×7

Each block is 2×2 of 8×8 cells

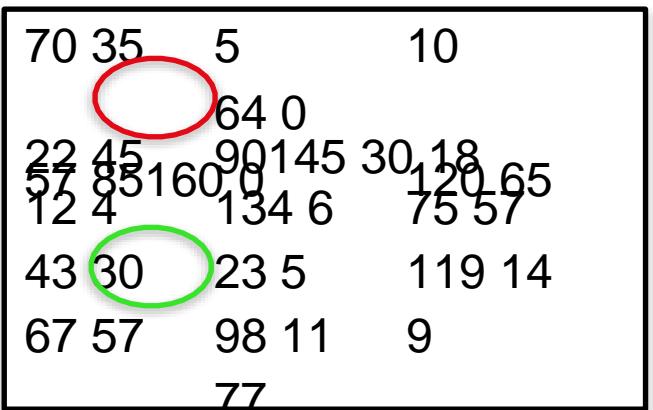
Histogram bins

Quantize the gradient orientation into 9 bins by gradient magnitude



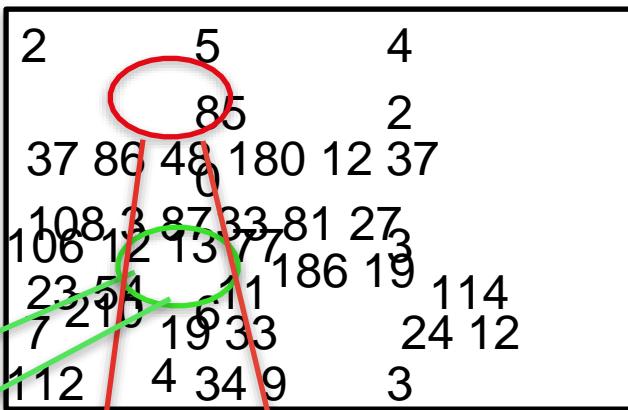
Histogram bins

Quantize gradient orientations into 9 bins



Gradient direction 23 54

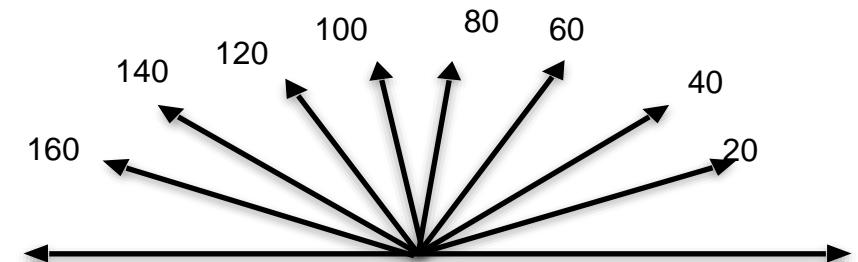
78 90



Gradient Magnitude

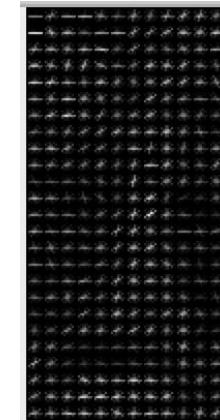
9

3



Concatenate Histograms

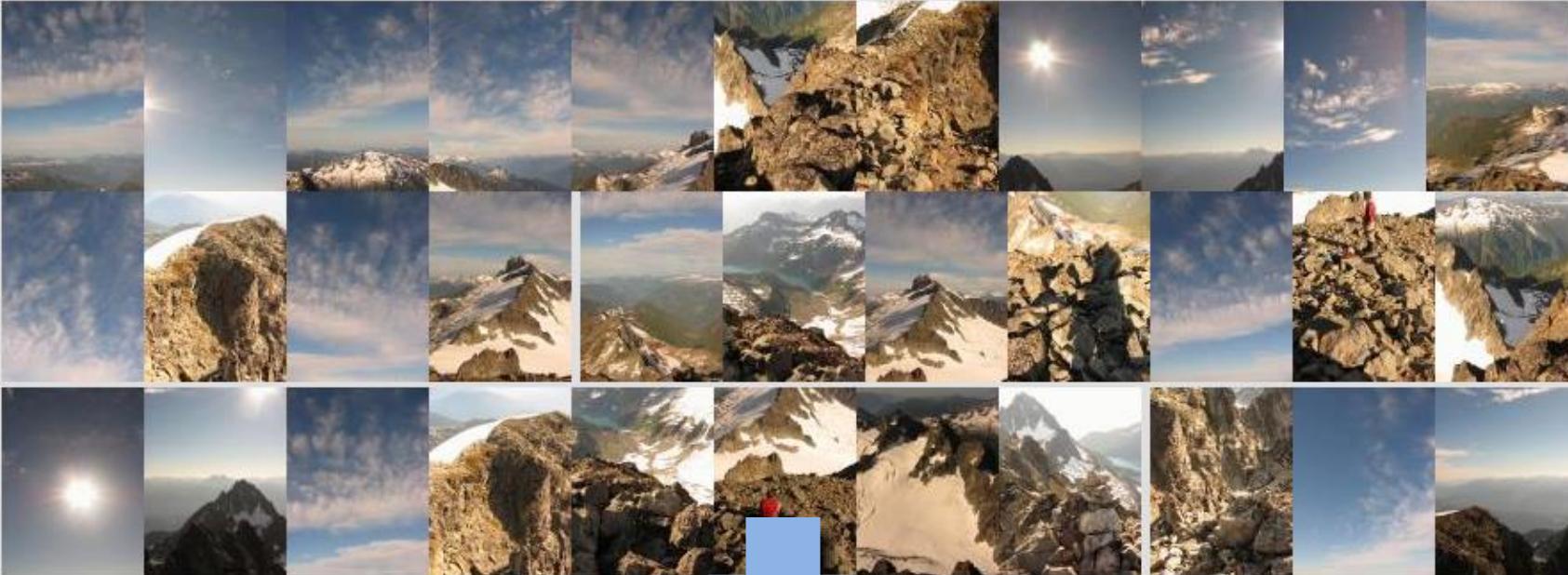
$$15 \times 7 * 9 * 4 = 3780$$



Local Features and Harris Corner Detectors

Adopted from (1) Rick Szeliski's lecture notes, CSE576, Spring 05 (2) CS5670: Computer Vision - Local features & Harris corner detection

Motivation: Automatic panoramas



Credit: Matt

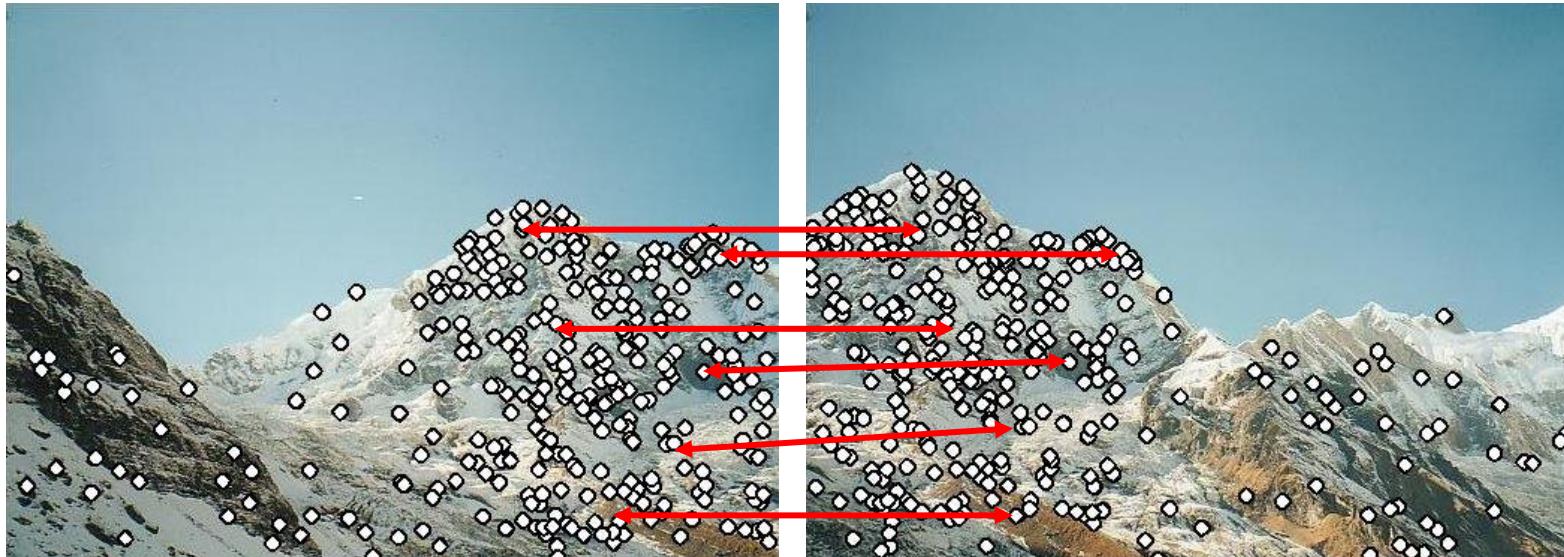
Why extract features?

- Motivation: panorama stitching
 - We have two images – how do we combine them?



Why extract features?

- Motivation: panorama stitching
 - We have two images – how do we combine them?



Step 1: extract features

Step 2: match features

Why extract features?

- Motivation: panorama stitching
 - We have two images – how do we combine them?



Step 1: extract
features

Step 2: match
features

Step 3: align
images

Application: Visual SLAM

- (aka Simultaneous Localization and Mapping)

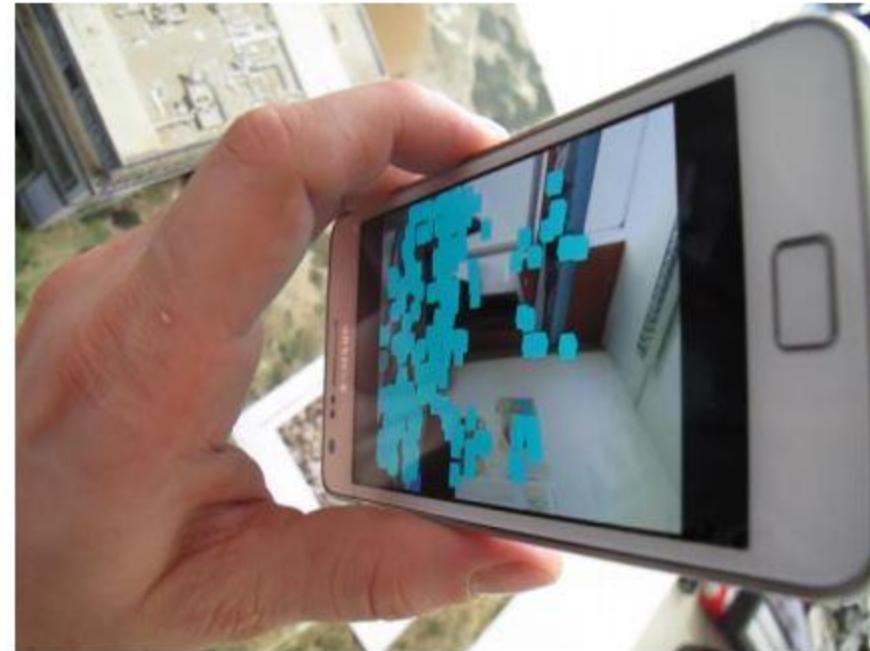


Image matching



by Diva Sian



by swashford

Harder case

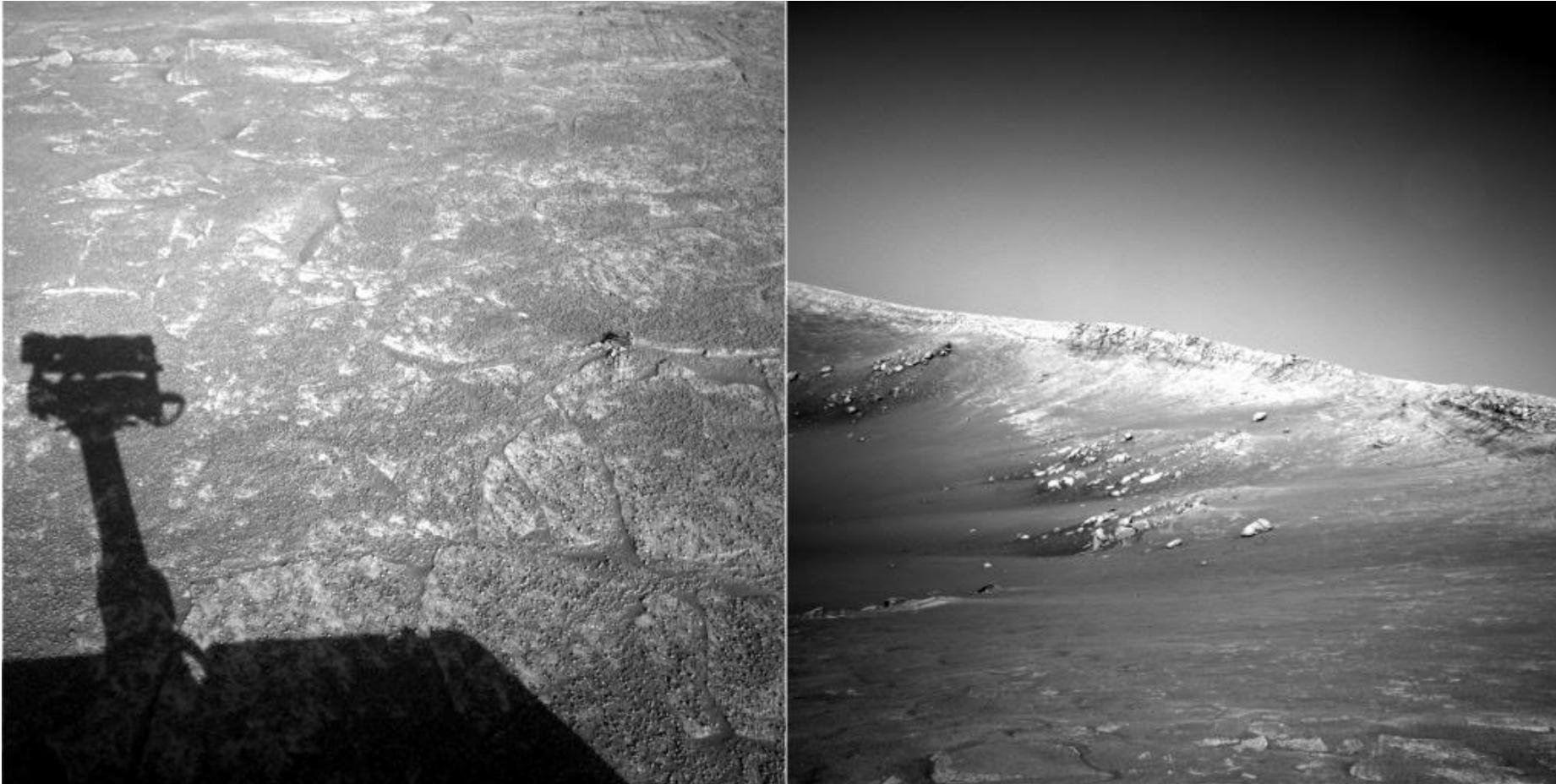


by Diva Sian

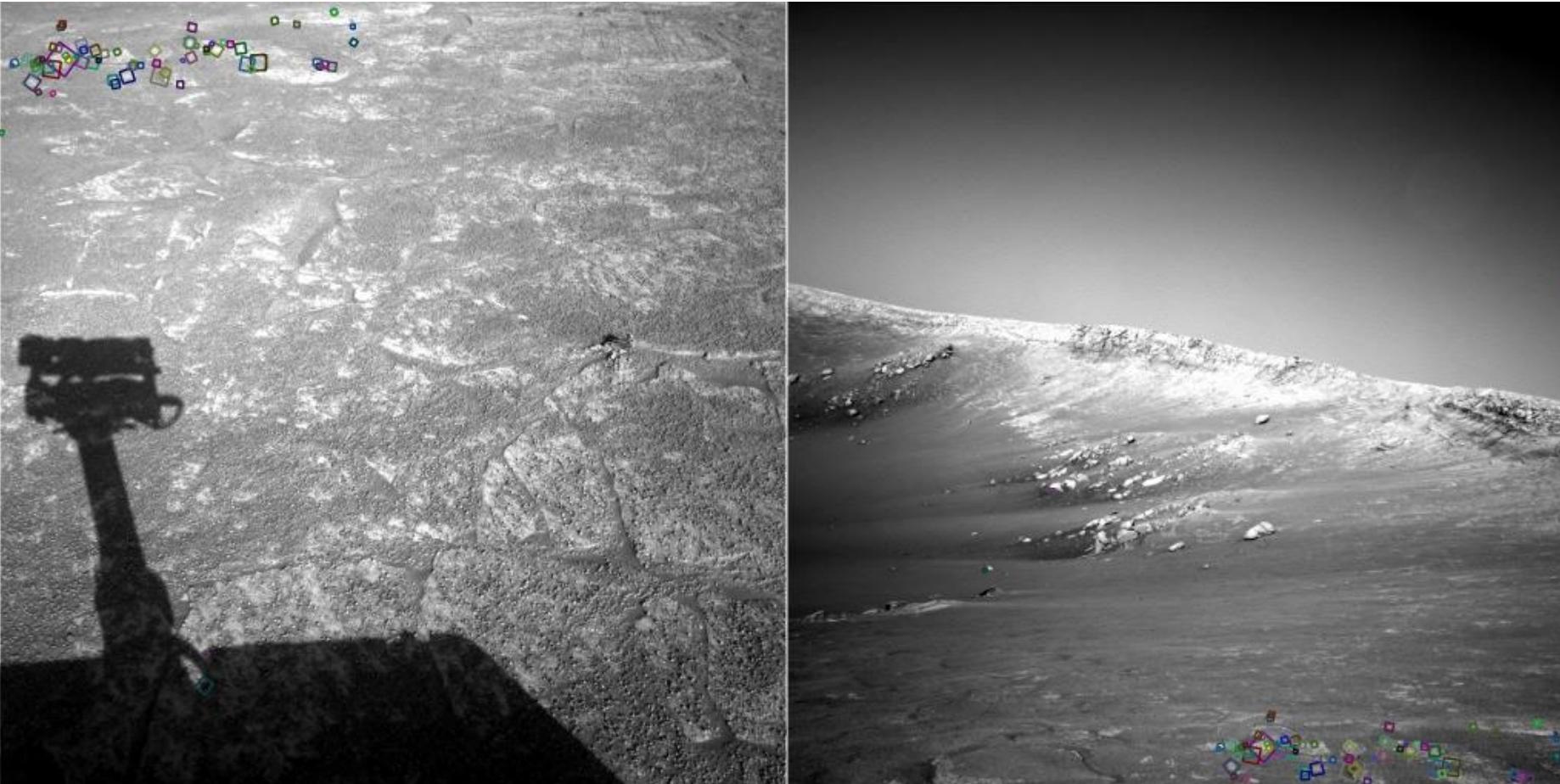


by scgbt

Harder still?

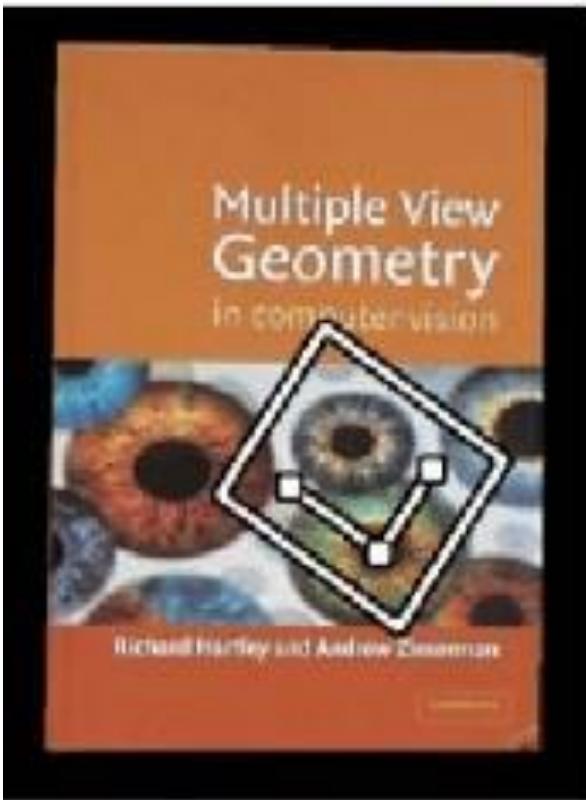


Answer below (look for tiny colored squares...)

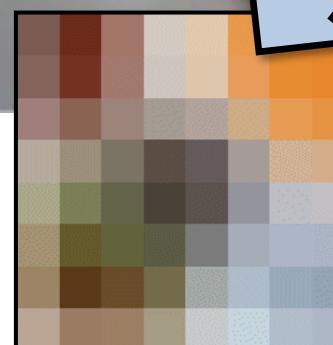
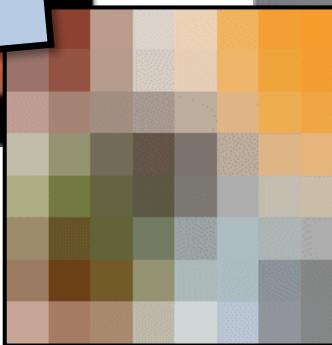
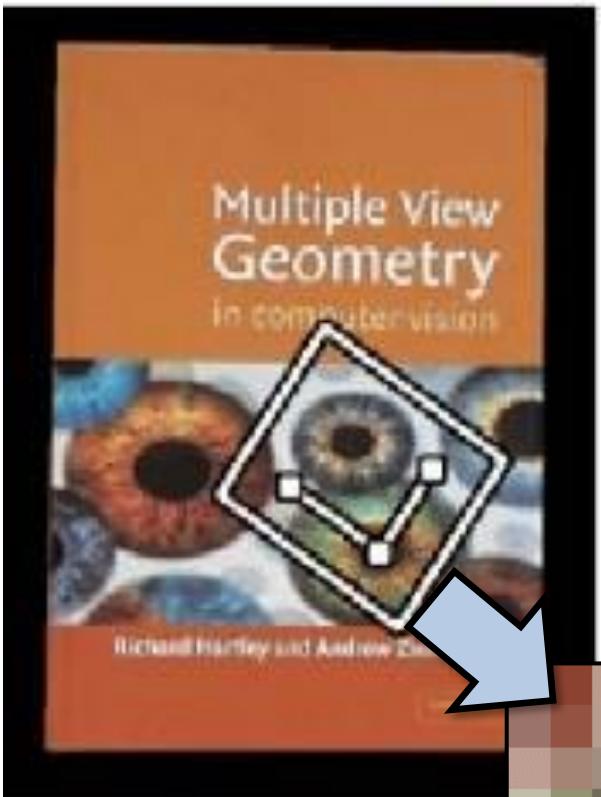


NASA Mars Rover images
with SIFT feature matches

Feature matching for object search



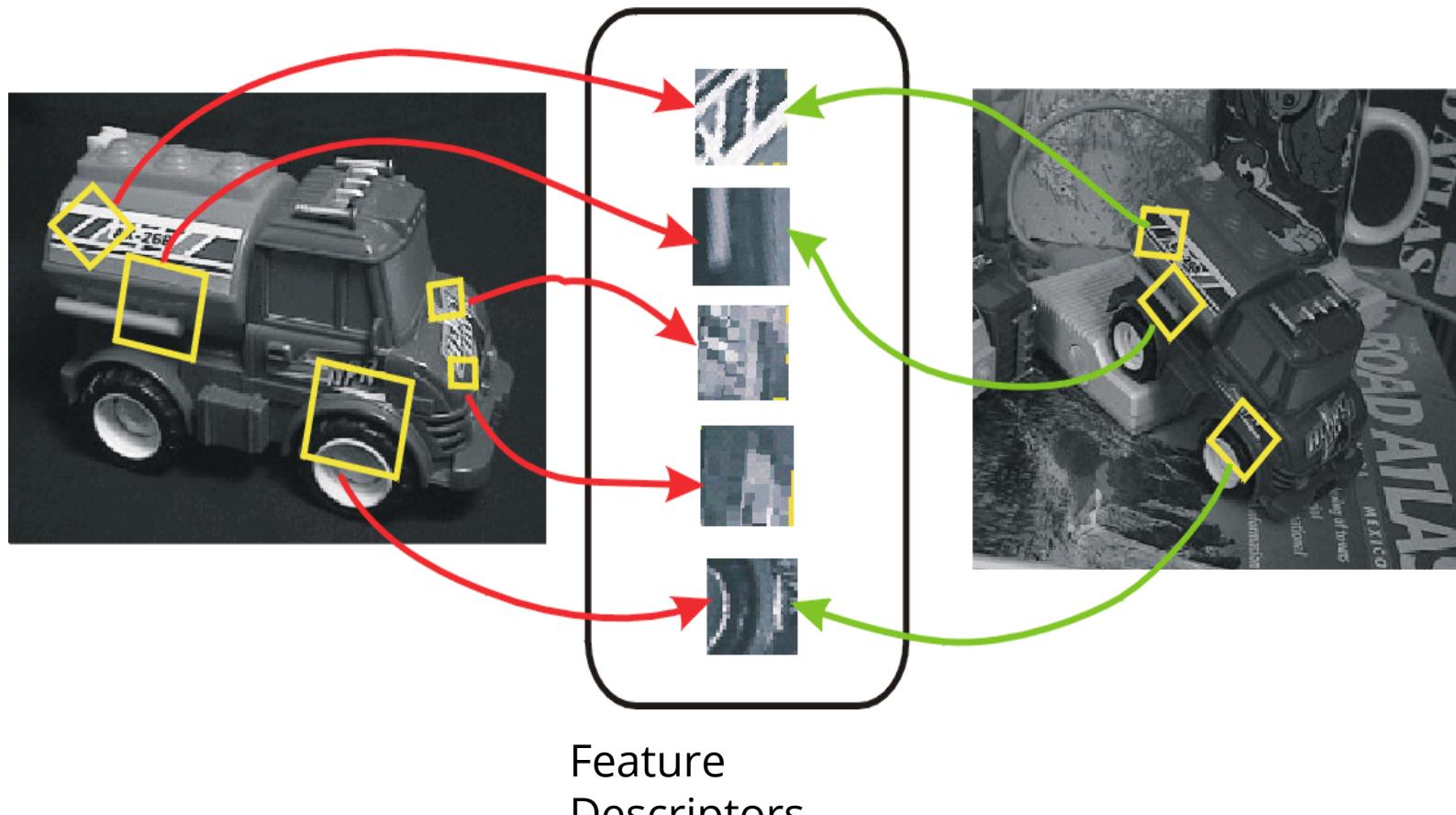
Feature Matching



Invariant local features

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, ...



Advantages of local features

Locality

- features are local, so robust to occlusion and clutter

Quantity

- hundreds or thousands in a single image

Distinctiveness:

- can differentiate a large database of objects

Efficiency

- real-time performance achievable

More motivation...

Feature points are used for:

- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking (e.g. for AR)
- Object recognition
- Image retrieval
- Robot/car navigation
- ... other



Approach

- 1. Feature detection:** find it
- 2. Feature descriptor:** represent it
- 3. Feature matching:** match it

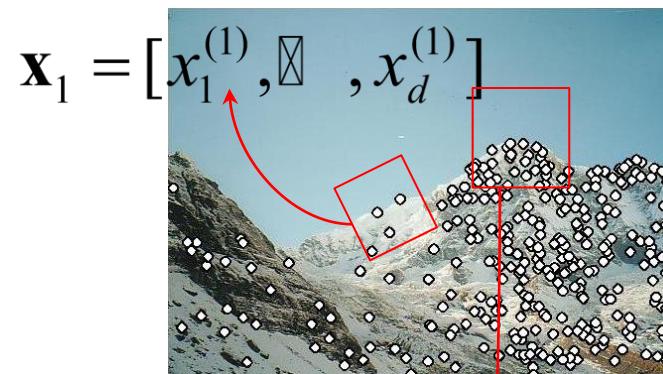
Feature tracking: track it, when motion

Local features: main components

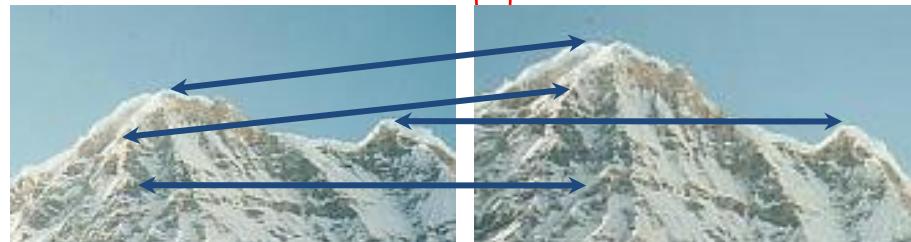
1) **Detection:** Identify the interest points



1) **Description:** Extract vector feature descriptor surrounding each interest point



1) **Matching:** Determine correspondence between descriptors in two views



Credit: Kristen Grauman

Want uniqueness

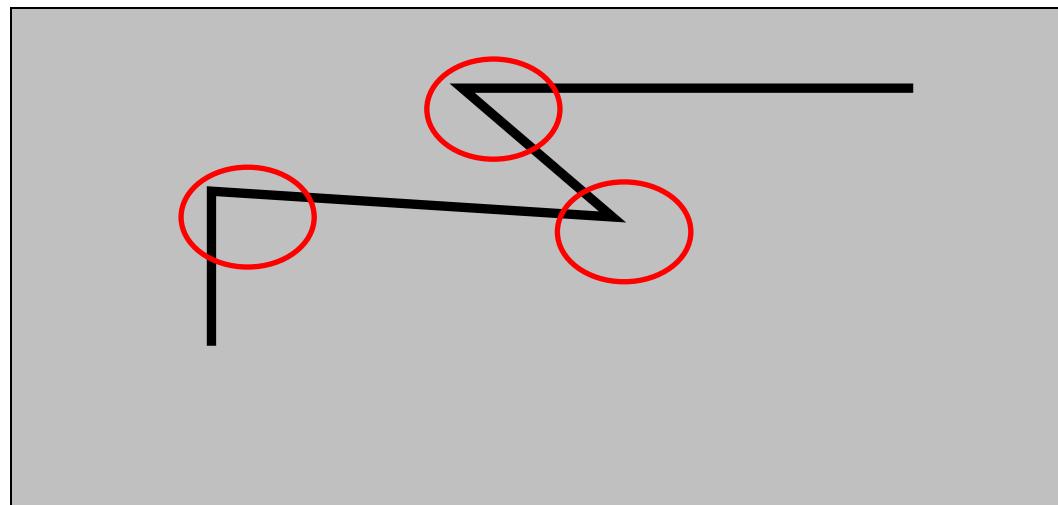
Look for image regions that are unusual

- Lead to unambiguous matches in other images

How to define “unusual”?

Harris corner detector

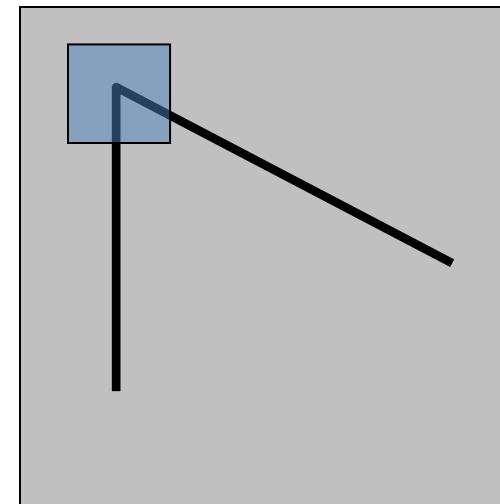
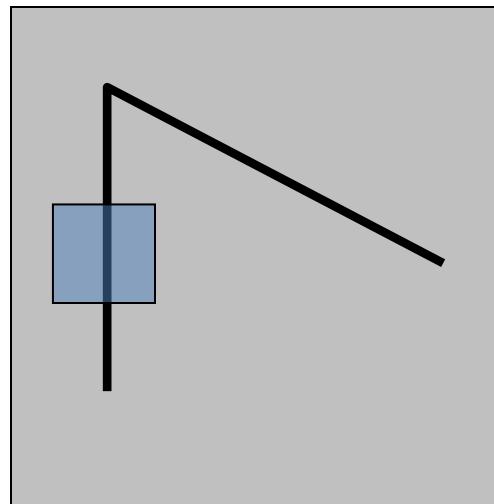
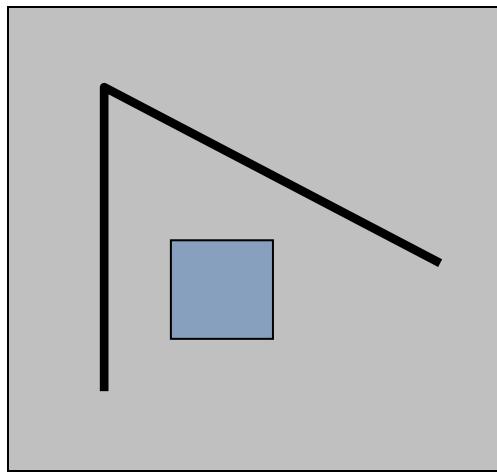
- C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988



Local measures of uniqueness

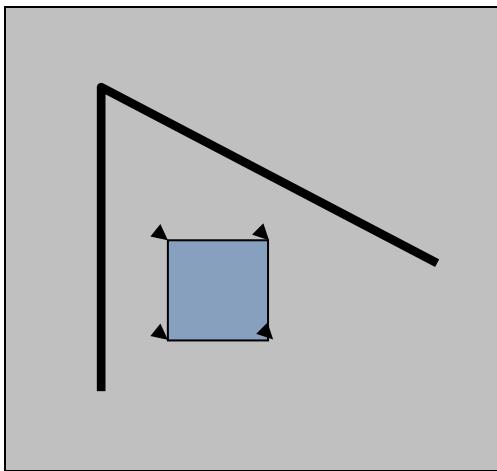
Suppose we only consider a small window of pixels

- What defines whether a feature is a good or bad candidate?

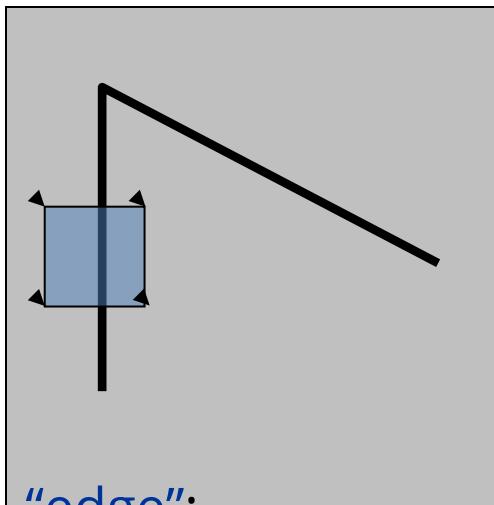


Local measures of uniqueness

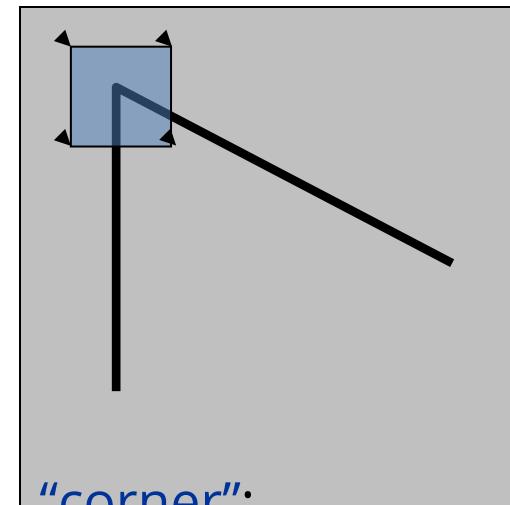
- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



“flat” region:
no change in all
directions



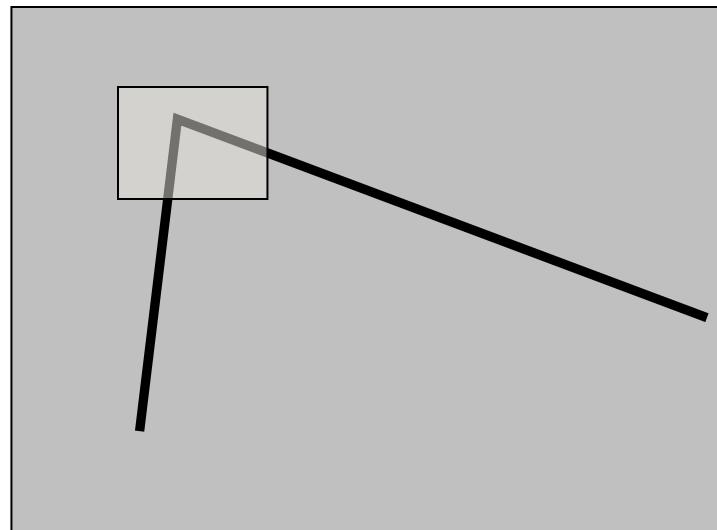
“edge”:
no change along
the edge direction



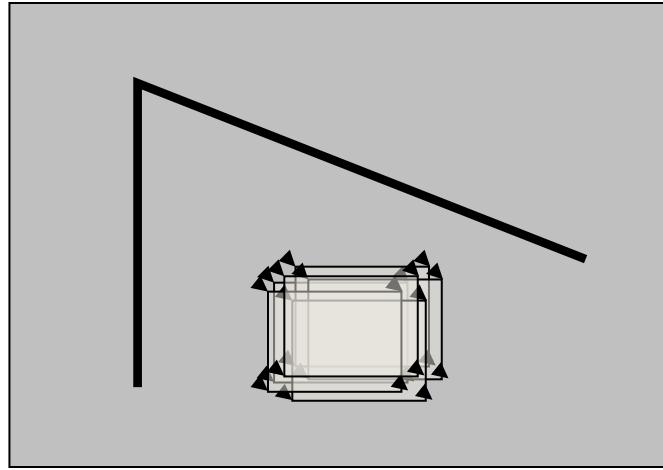
“corner”:
significant change
in all directions

The Basic Idea

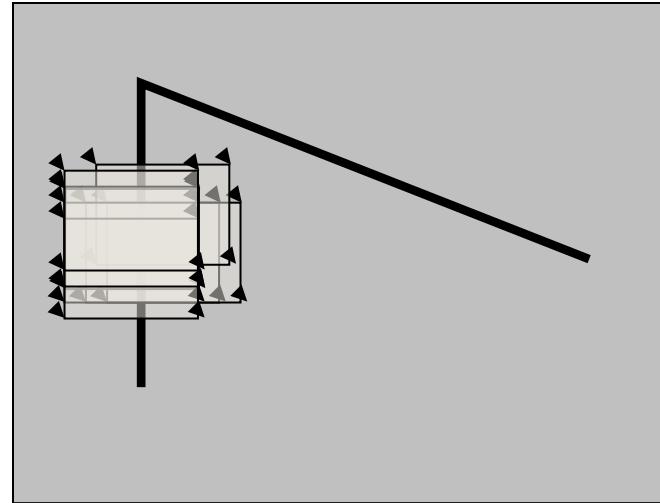
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give a *large change* in intensity



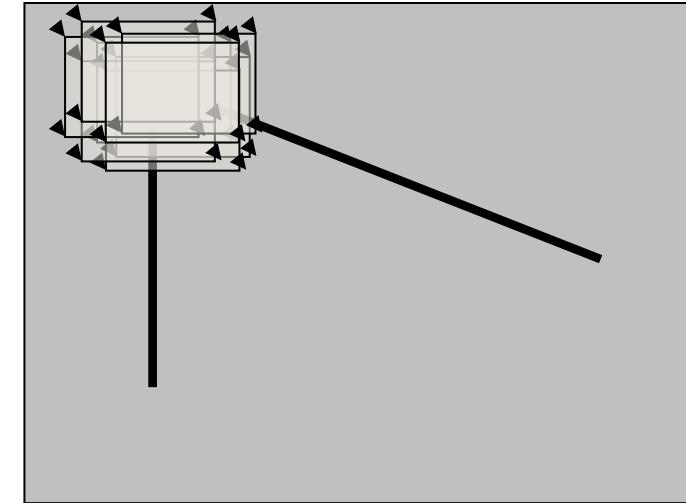
Harris Detector: Basic Idea



“flat” region:
no change in all
directions



“edge”:
no change along the
edge direction

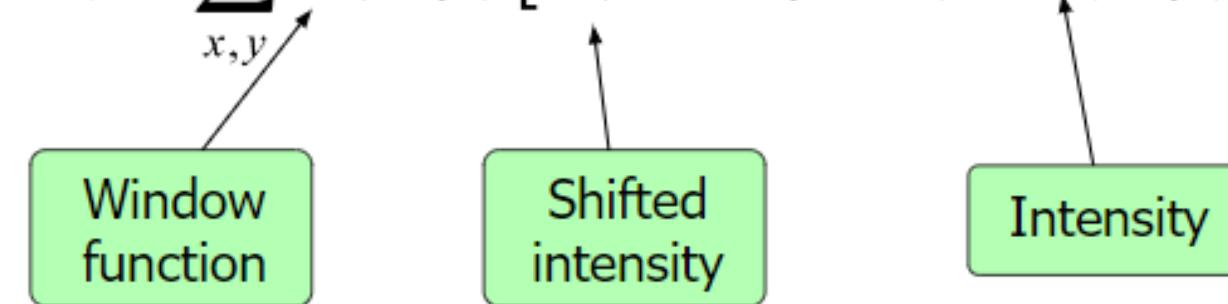


“corner”:
significant change in all
directions

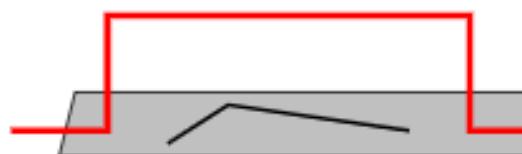
Harris Detector: Mathematics

Change of intensity for the shift $[u, v]$:

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



Window function $w(x, y) =$



1 in window, 0 outside



Gaussian

Harris Detector: Mathematics

Taylor Series expansion of I :

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u, v) is small, then first order approximation is good

$$\begin{aligned} I(x + u, y + v) &\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

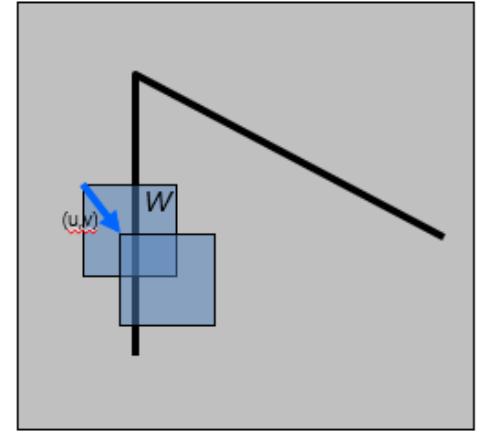
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide...

Harris Detector: Mathematics

Consider shifting the window W by (u, v)

- define an SSD “error” $E(u, v)$:



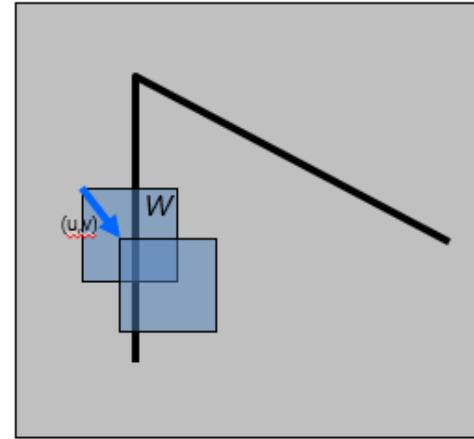
$$\begin{aligned} E(u, v) &= \sum_{(x,y) \in W} [I(x + u, y + v) - I(x, y)]^2 \\ &\approx \sum_{(x,y) \in W} [I(x, y) + I_x u + I_y v - I(x, y)]^2 \\ &\approx \sum_{(x,y) \in W} [I_x u + I_y v]^2 \end{aligned}$$

Harris Detector: Mathematics

Consider shifting the window W by (u, v)

- define an SSD “error” $E(u, v)$:

$$\begin{aligned} E(u, v) &\approx \sum_{(x,y) \in W} [I_x u + I_y v]^2 \\ &\approx A u^2 + 2Buv + C v^2 \end{aligned}$$



$$A = \sum_{(x,y) \in W} I_x^2 \quad B = \sum_{(x,y) \in W} I_x I_y \quad C = \sum_{(x,y) \in W} I_y^2$$

- Thus, $E(u, v)$ is locally approximated as a quadratic error function

Harris Detector: Mathematics

The surface $E(u,v)$ is locally approximated by a quadratic form.

$$E(u, v) \approx Au^2 + 2Buv + Cv^2$$

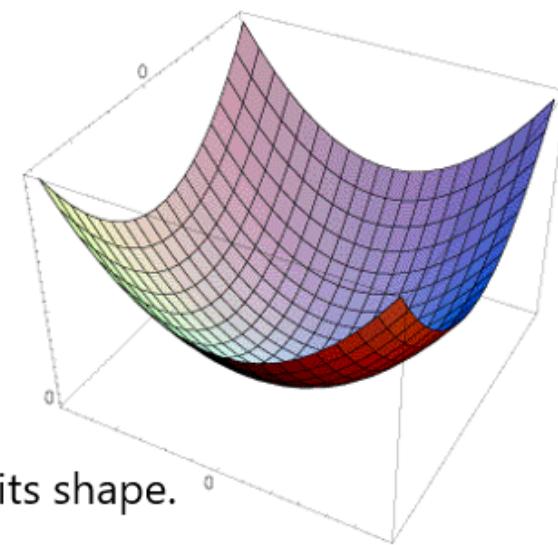
$$\approx [u \ v] \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y) \in W} I_x^2$$

$$B = \sum_{(x,y) \in W} I_x I_y$$

$$C = \sum_{(x,y) \in W} I_y^2$$

$\underbrace{}$
 H



Let's try to understand its shape.

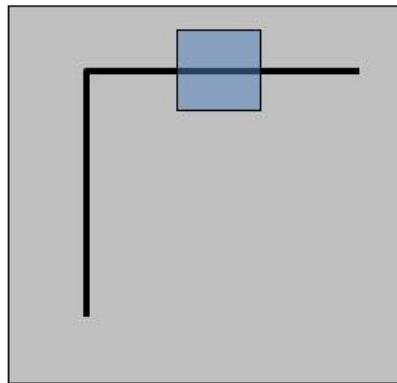
Harris Detector: Mathematics

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_H \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y) \in W} I_x^2$$

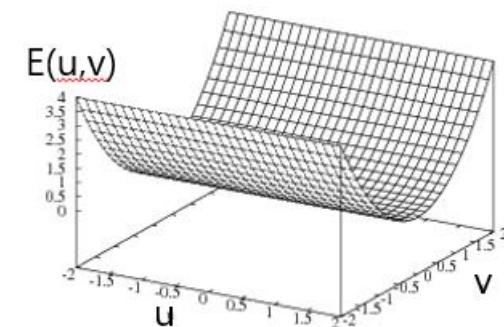
$$B = \sum_{(x,y) \in W} I_x I_y$$

$$C = \sum_{(x,y) \in W} I_y^2$$



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$



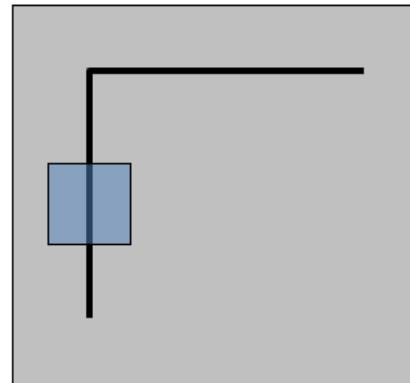
Harris Detector: Mathematics

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_H \begin{bmatrix} u \\ v \end{bmatrix}$$

$$A = \sum_{(x,y) \in W} I_x^2$$

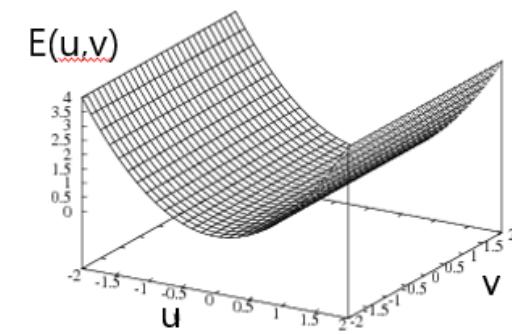
$$B = \sum_{(x,y) \in W} I_x I_y$$

$$C = \sum_{(x,y) \in W} I_y^2$$



Vertical edge: $I_y = 0$

$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$



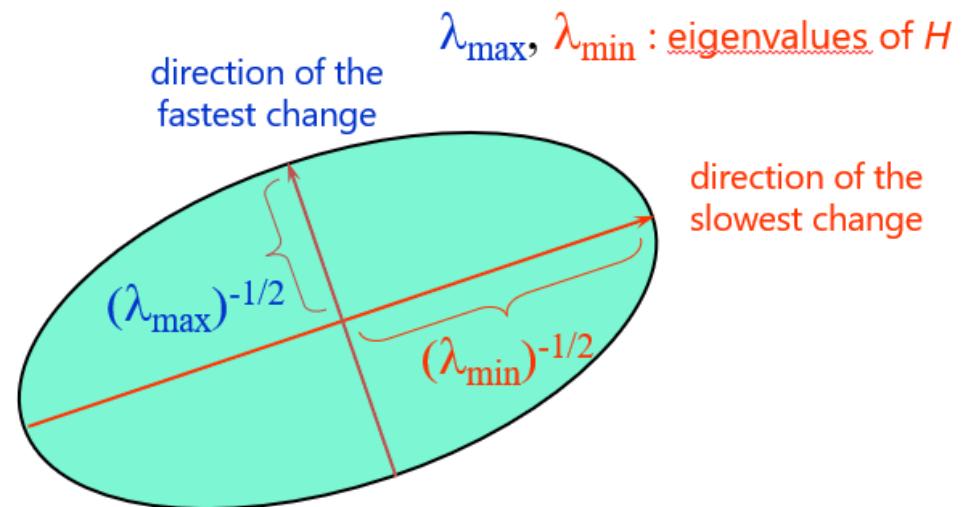
Harris Detector: Mathematics

General case

We can visualize H as an ellipse with axis lengths determined by the *eigen values* of H and orientation determined by the *eigenvectors* of H

Ellipse equation:

$$[u \ v] H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



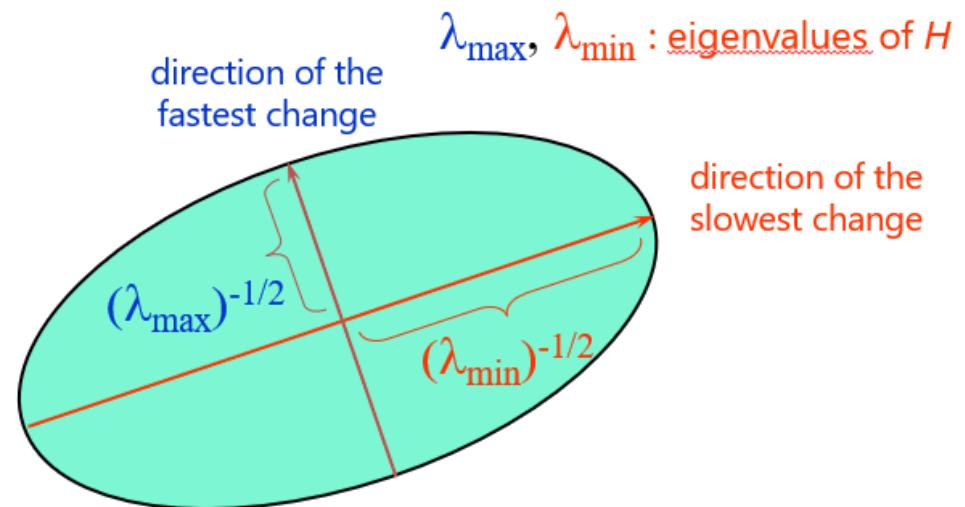
Harris Detector: Mathematics

General case

We can visualize H as an ellipse with axis lengths determined by the *eigen values* of H and orientation determined by the *eigenvectors* of H

Ellipse equation:

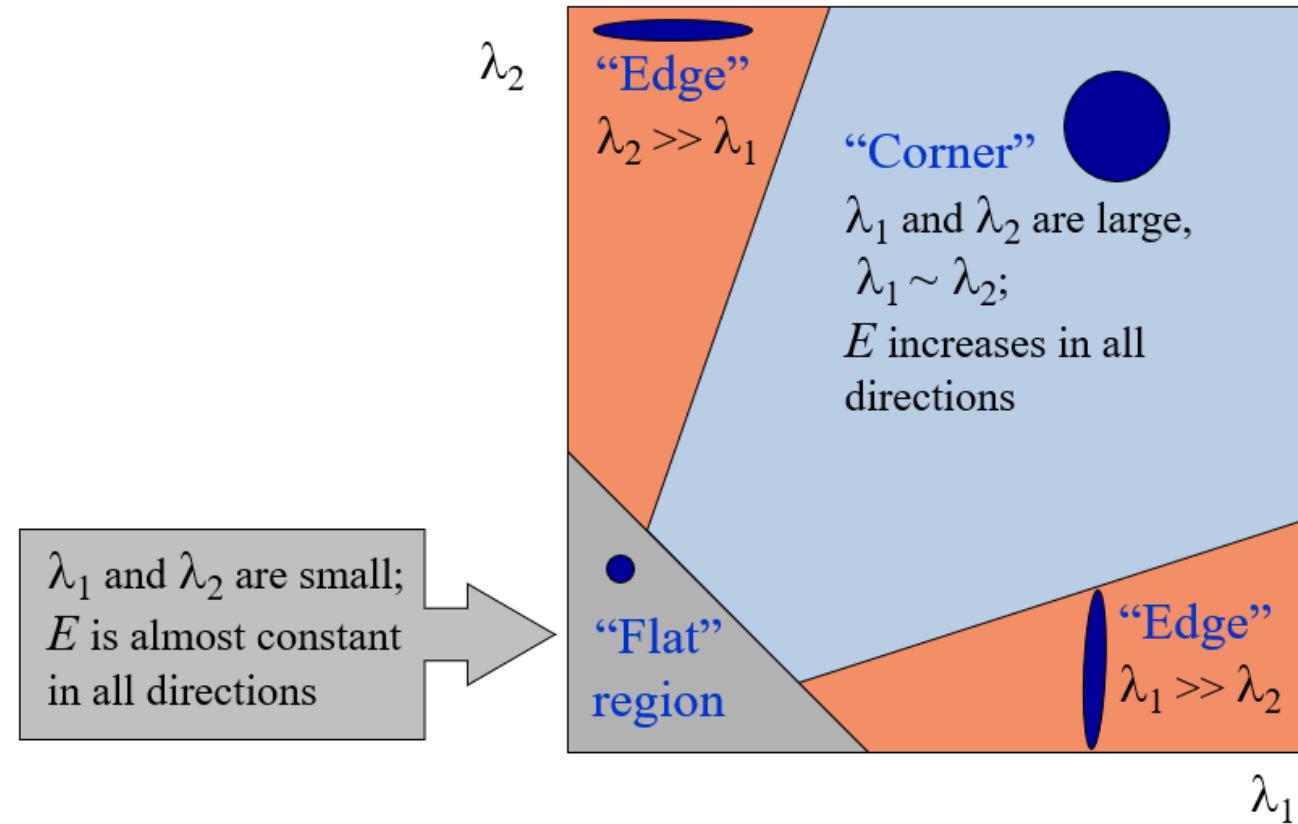
$$[u \ v] H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



Harris Detector: Mathematics

Interpreting the eigenvalues

Classification of image points using eigenvalues of M :



Harris Detector: Mathematics

Corner detection summary

Here's what you do:

- Compute the gradient at each point in the image
- For each pixel:
 - Create the H matrix from the entries in the gradient
 - Compute the eigenvalues.
 - Find points with large response ($\lambda_{\min} >$ threshold)
- Choose those points where λ_{\min} is a local maximum as features

$$H = \sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Harris Detector: Mathematics

Measure of corner response:

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

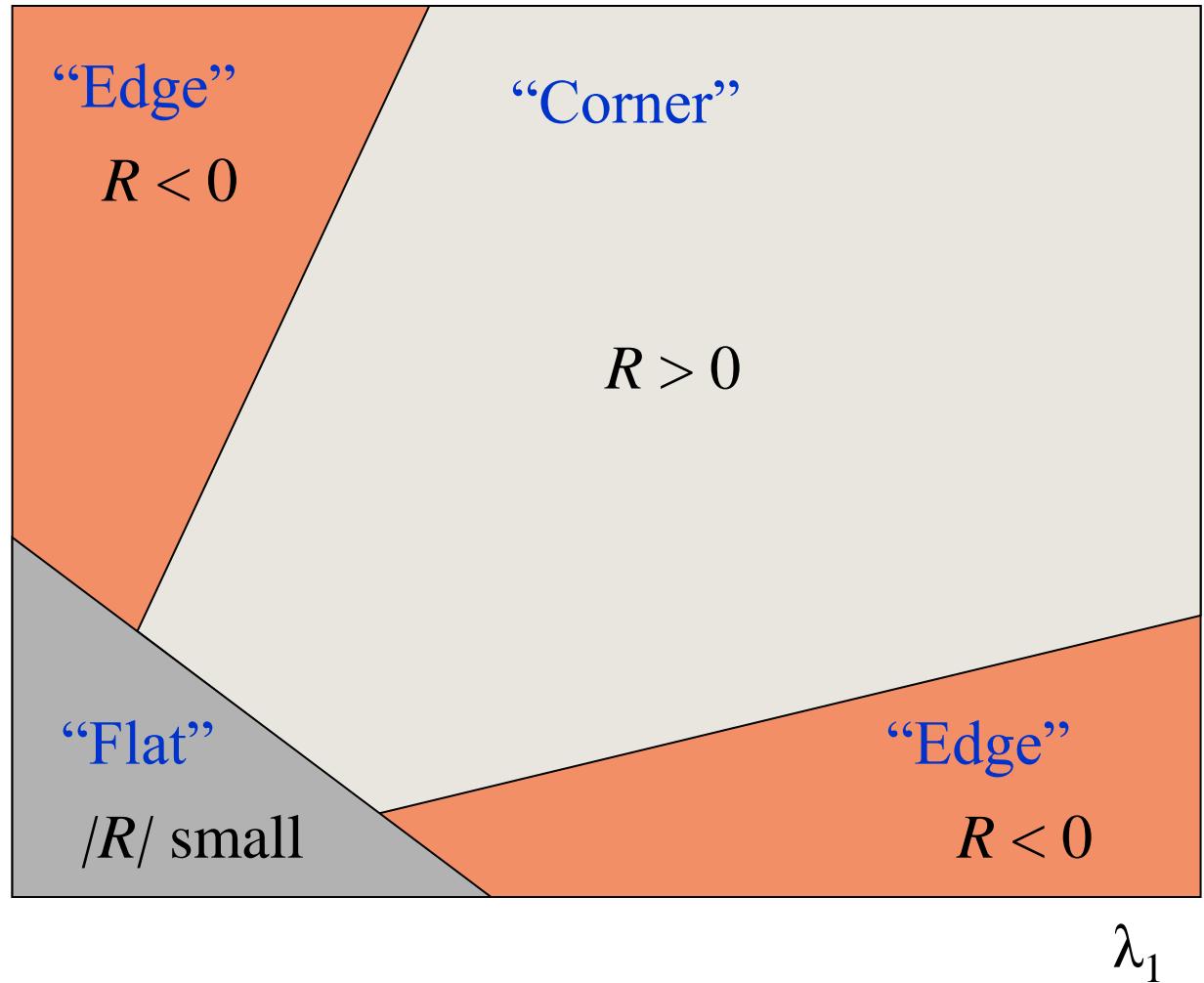
$$\det M = \lambda_1 \lambda_2$$

$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

(k – empirical constant, $k = 0.04\text{-}0.06$)

Harris Detector: Mathematics

- R depends only on eigenvalues of \mathbf{M}
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



Harris Detector

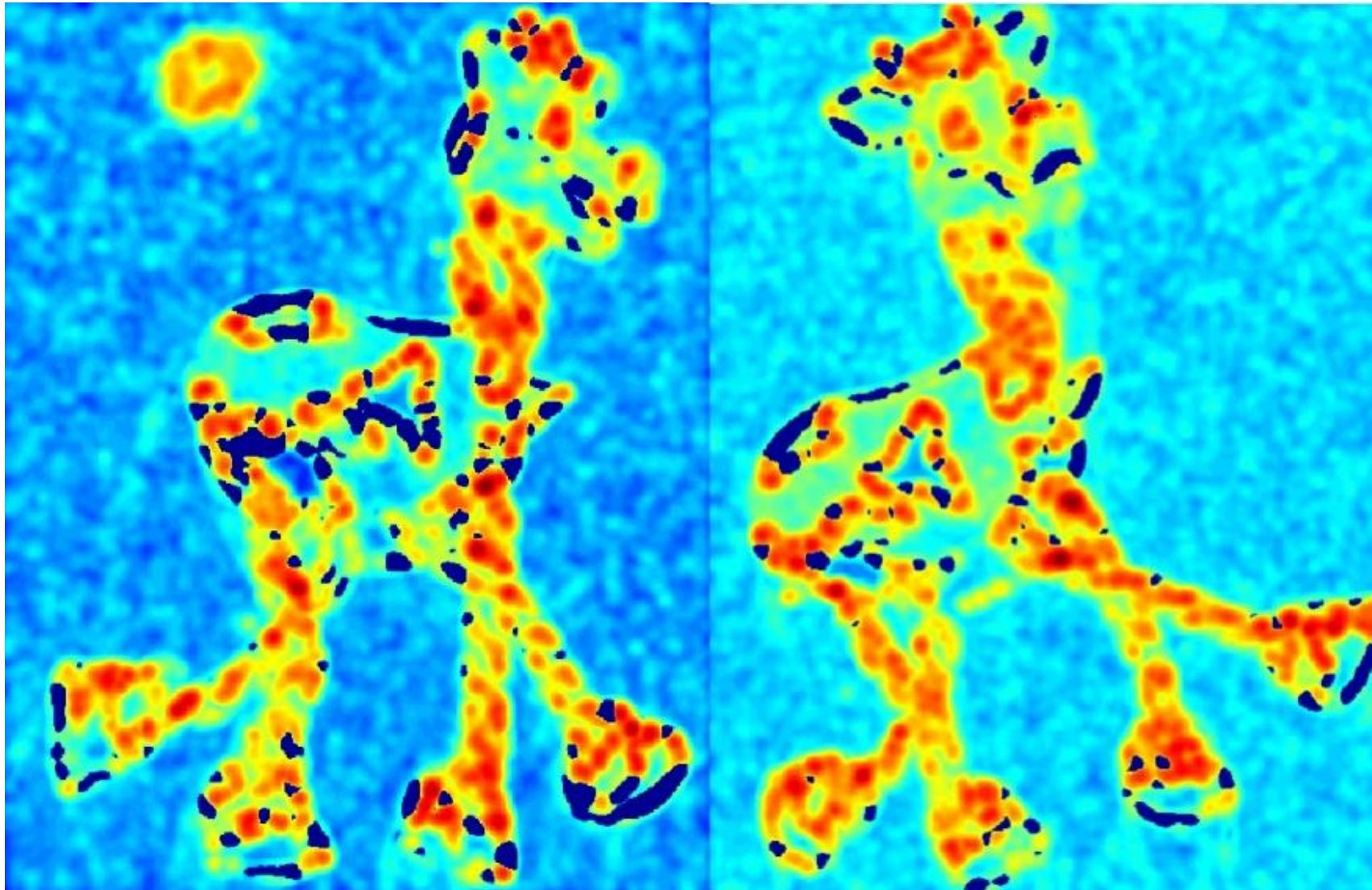
- The Algorithm:
 - Find points with large corner response function R ($R >$ threshold)
 - Take the points of local maxima of R

Harris Detector: Workflow



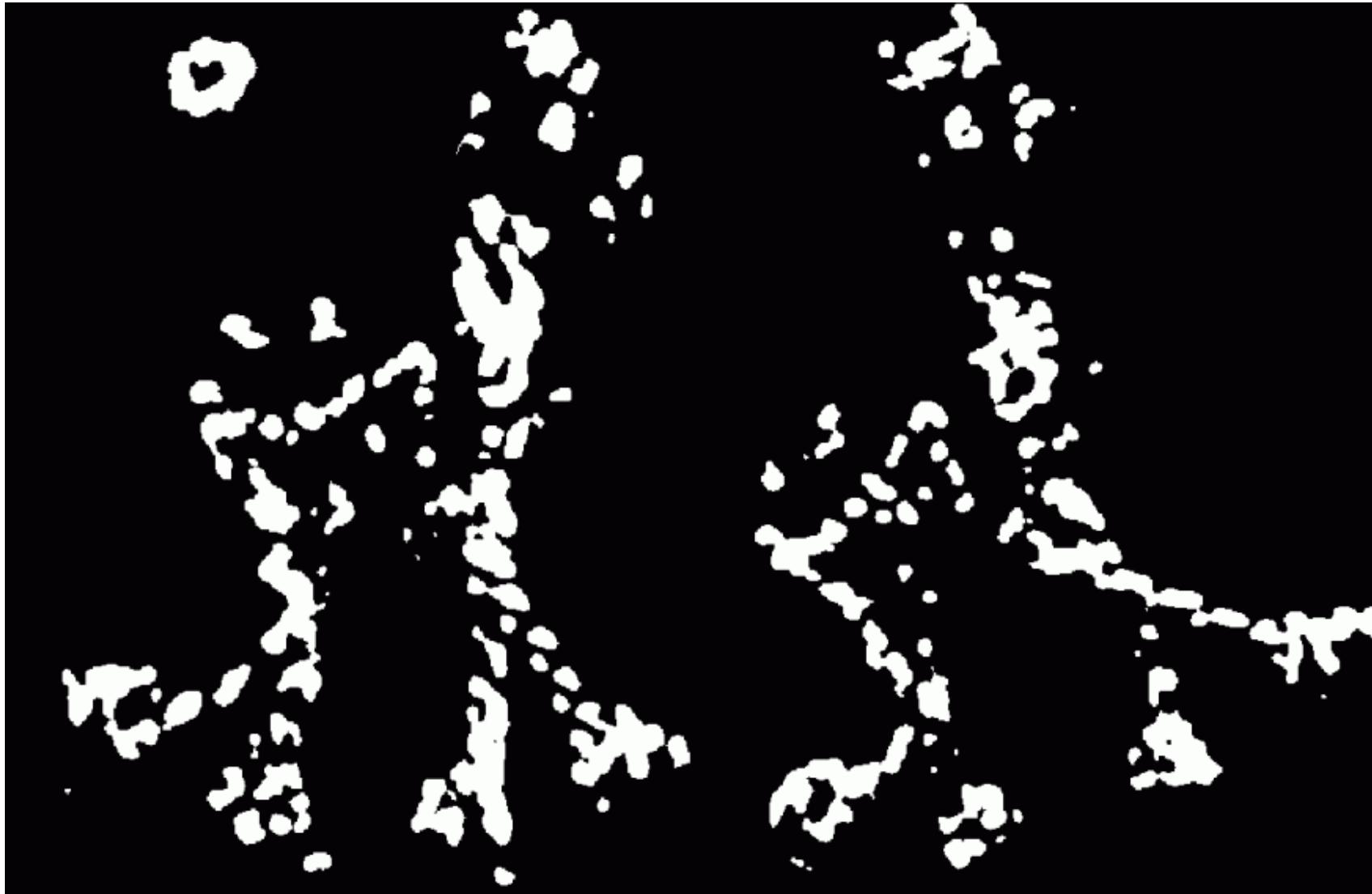
Harris Detector: Workflow

Compute corner response R



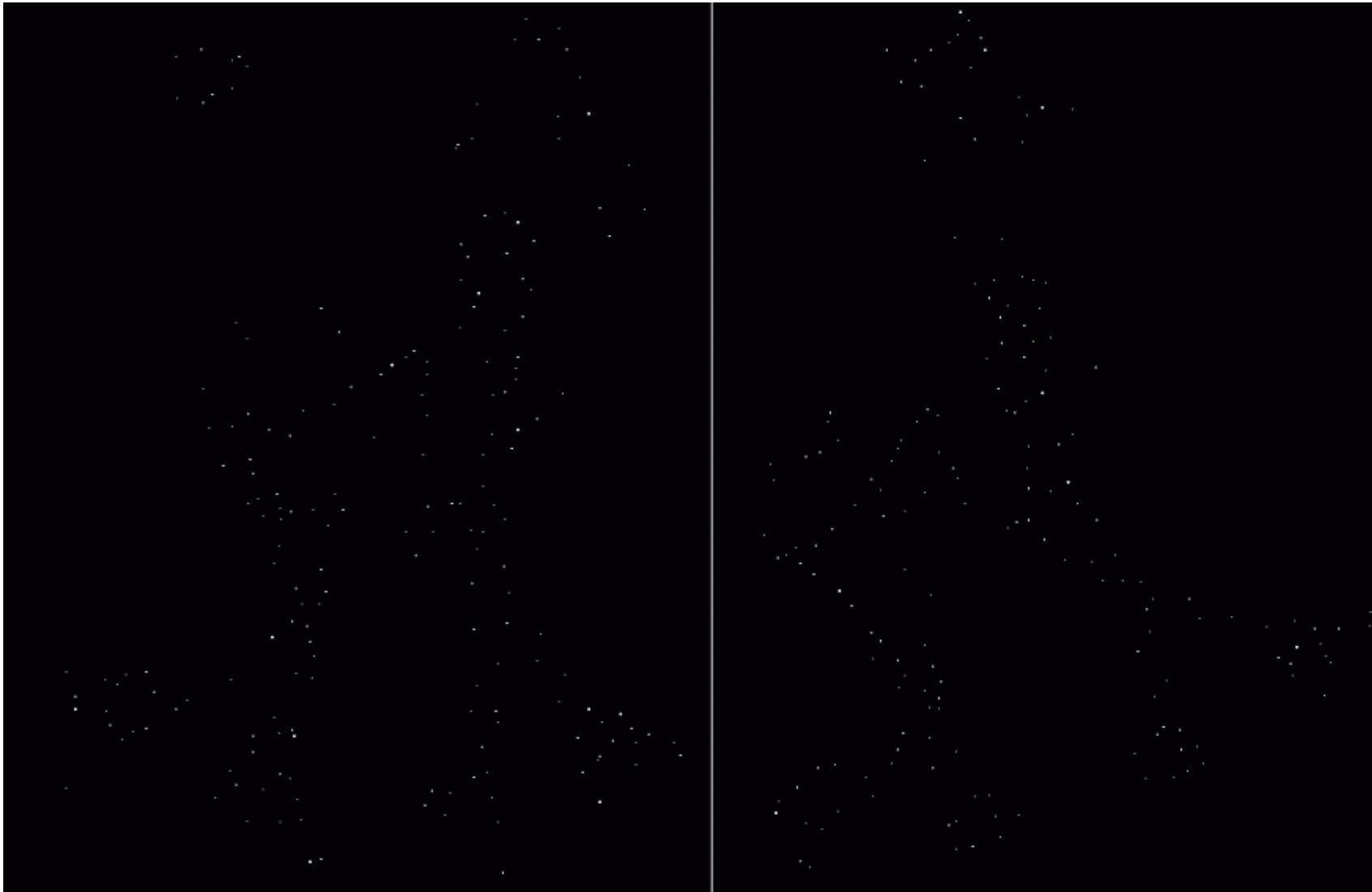
Harris Detector: Workflow

Find points with large corner response: $R > \text{threshold}$



Harris Detector: Workflow

Take only the points of local maxima of R



Harris Detector: Workflow



Algorithm 5.5: Harris corner detector

1. Filter the image with a Gaussian.
2. Estimate intensity gradient in two perpendicular directions for each pixel, $\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial f(x,y)}{\partial y}$. This is performed by twice using a 1D convolution with the kernel approximating the derivative.
3. For each pixel and a given neighborhood window:
 - Calculate the local structure matrix A .
 - Evaluate the response function $R(A)$.
4. Choose the best candidates for corners by selecting a threshold on the response function $R(A)$ and perform non-maximal suppression.

Readings:

Histogram of Oriented Gradients:

- (1) Histogram of Oriented Gradients <https://courses.cs.duke.edu/fall15/cps274/notes/hog.pdf>
- (2) Histograms of Oriented Gradients for Human Detection Navneet Dalal and Bill Triggs
<https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>

Harris Corner Detector

- (1) [Milan Sonka , \(Our Textbook-2\)](#) : Section 5.3.10
[optional, yet recommended reading , Milan Sonka Section 5.3.11]



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

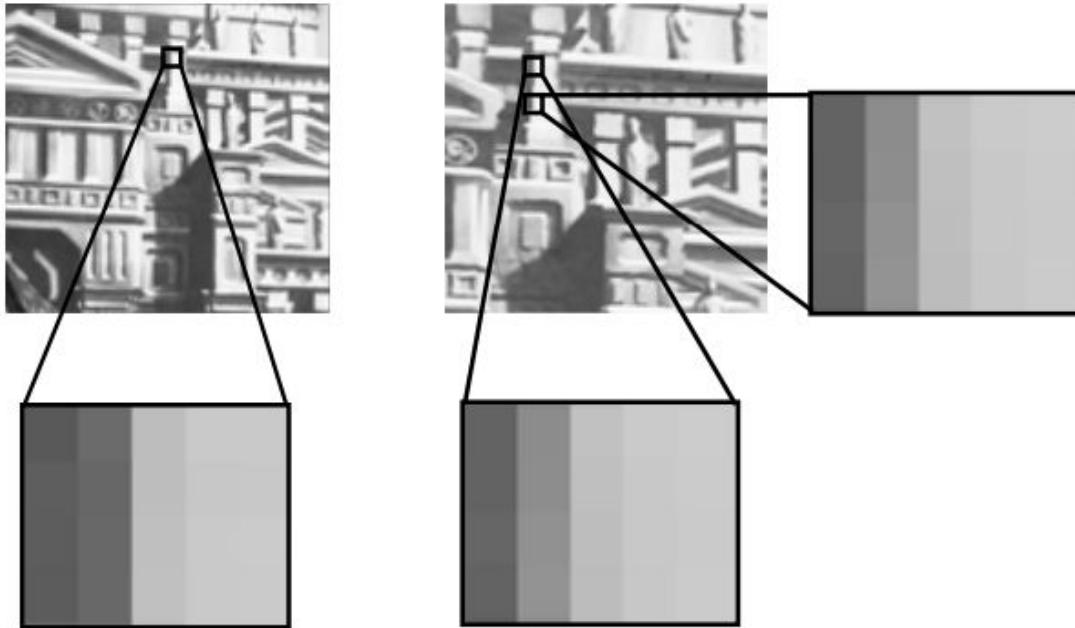
Thank you

SIFT

What is an Interesting Point/Feature?

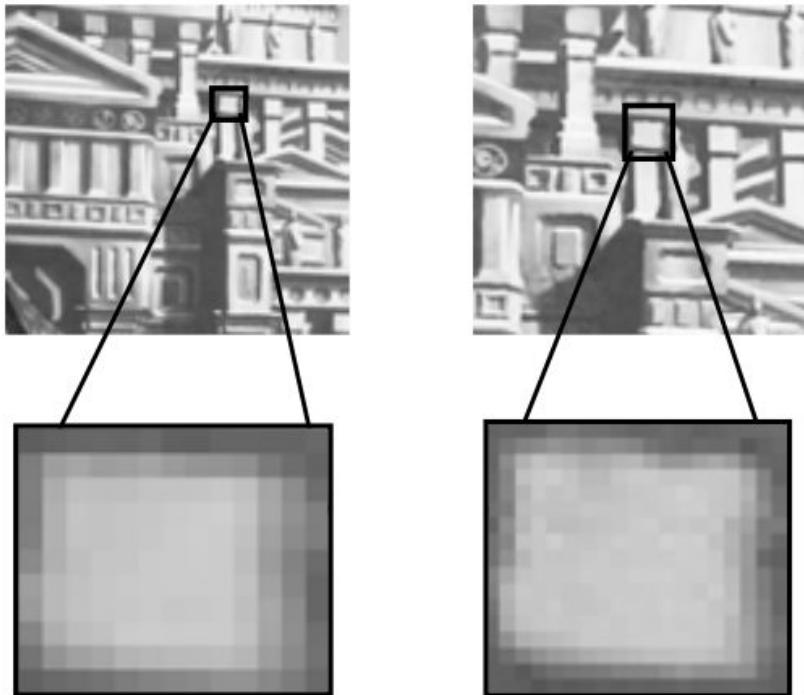
- Has rich image content (brightness variation, color variation, etc.) within the local window
- Has well-defined representation (signature) for matching/comparing with other points
- Has a well-defined position in the image
- Should be invariant to image rotation and scaling
- Should be insensitive to lighting changes

Are Lines/Edges Interesting?



Cannot “Localize” an Edge

Are Blobs Interesting?

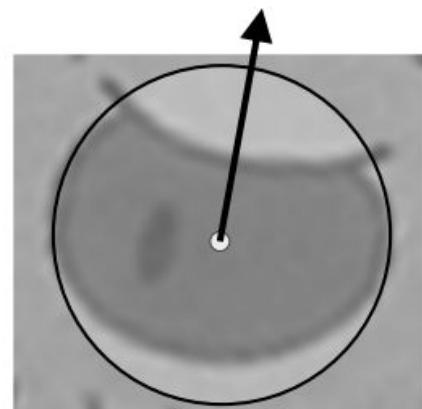


Yes! Blobs have fixed position and definite size.

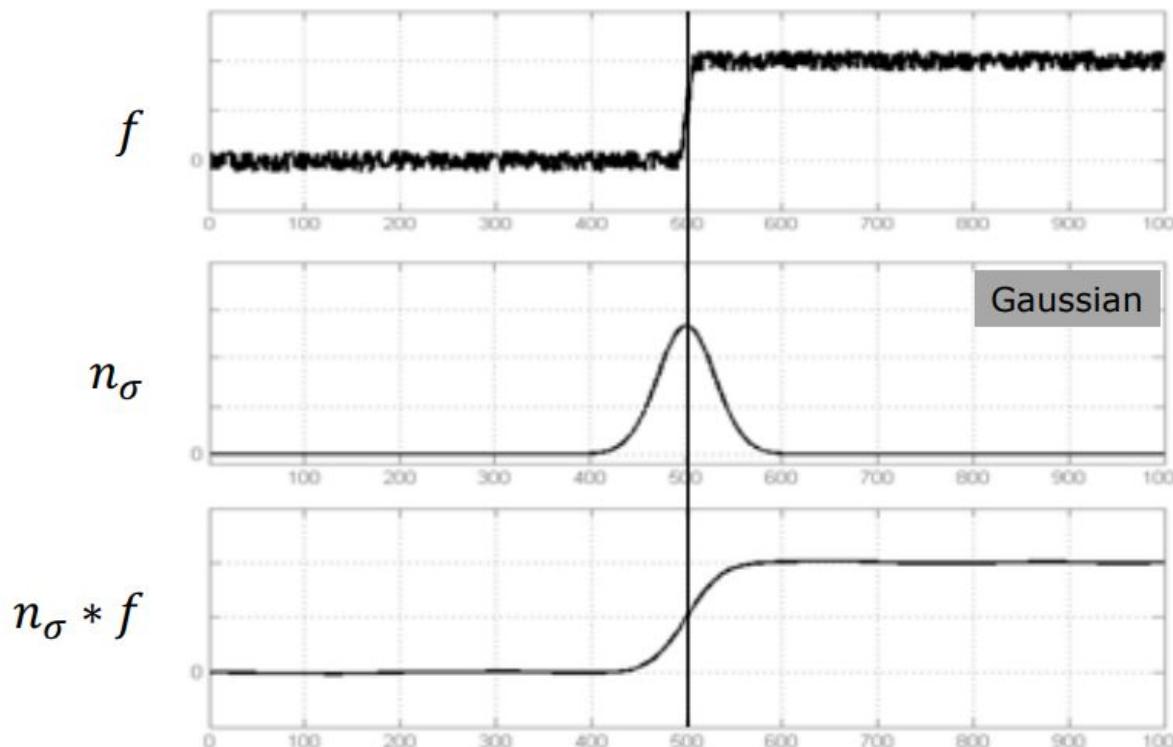
Blobs as Interest Points

For a Blob-like Feature to be useful, we need to:

- Locate the blob
- Determine its size
- Determine its orientation
- Formulate a description or
signature that is independent of
size and orientation

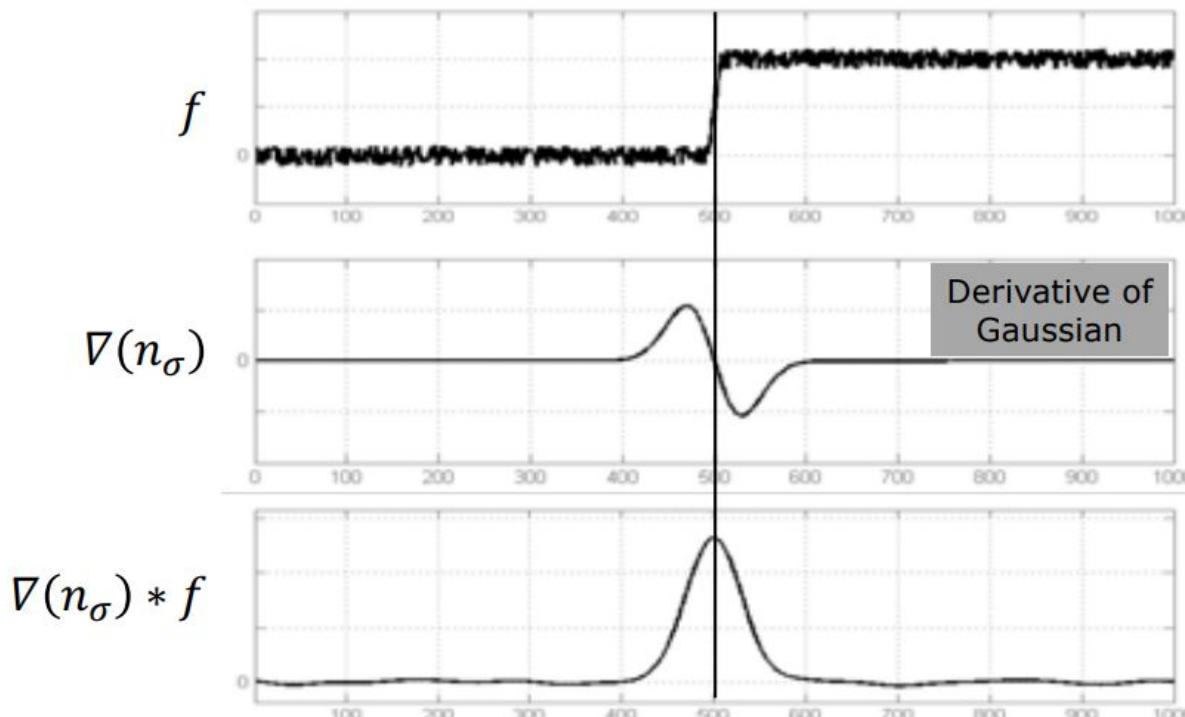


Review: Gaussian Filter



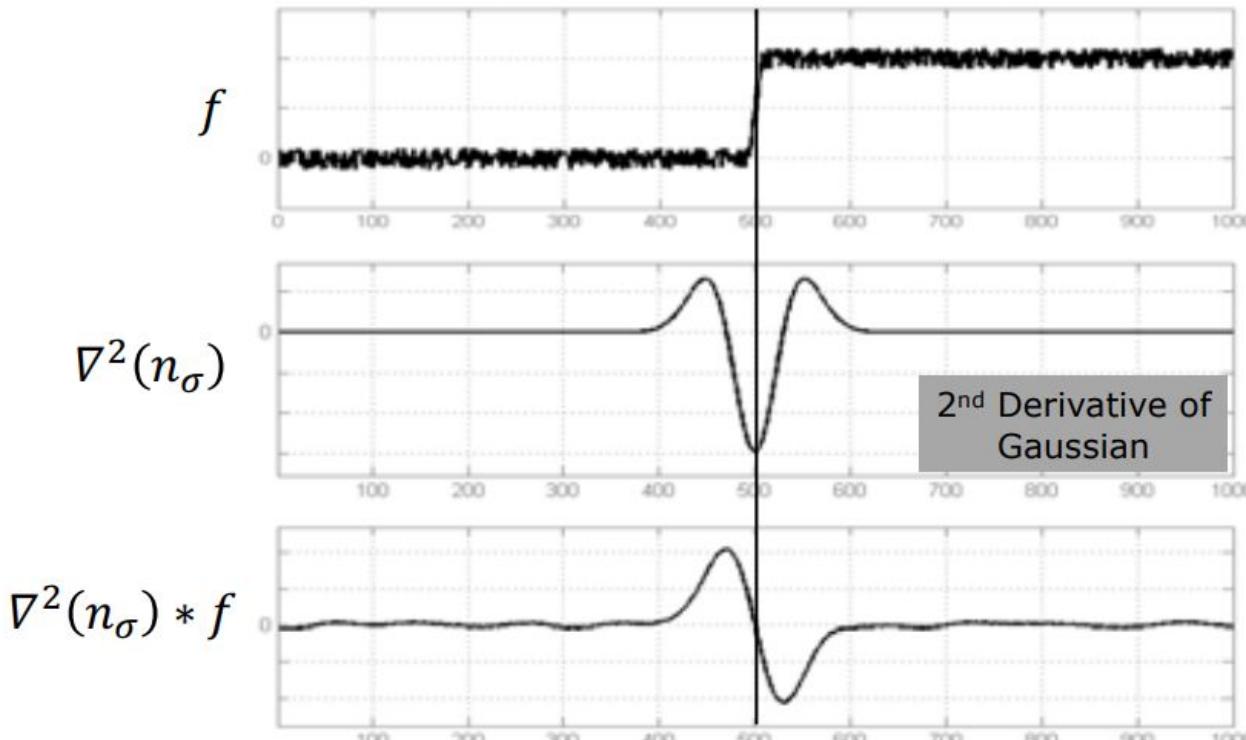
Gaussian Filter is used for removing noise by smoothing

Review: Derivative of Gaussian



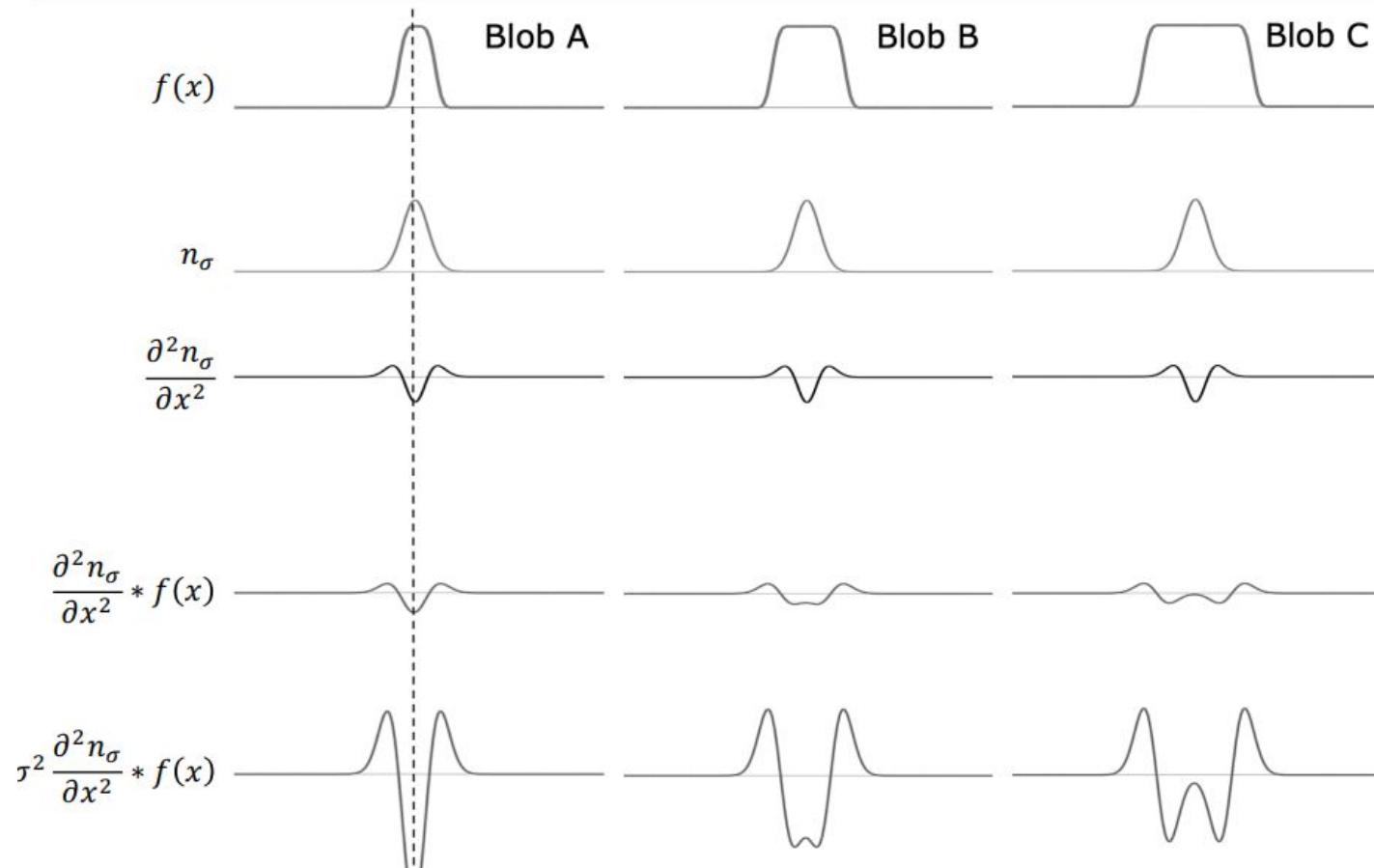
Extremum of Derivative of Gaussian denotes an Edge

Review: 2nd Derivative of Gaussian

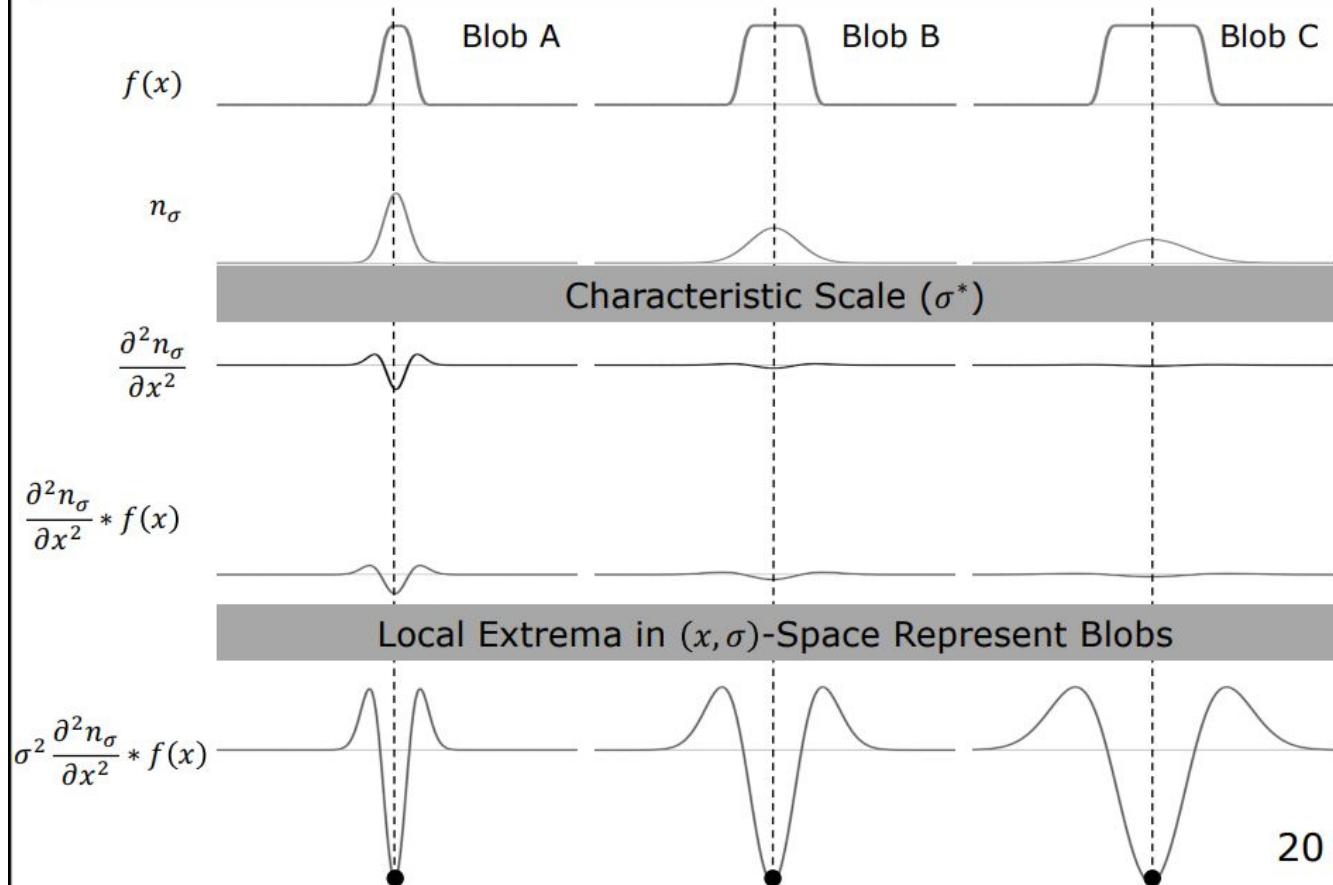


Zero Crossing in 2nd Derivative of Gaussian denote an Edge

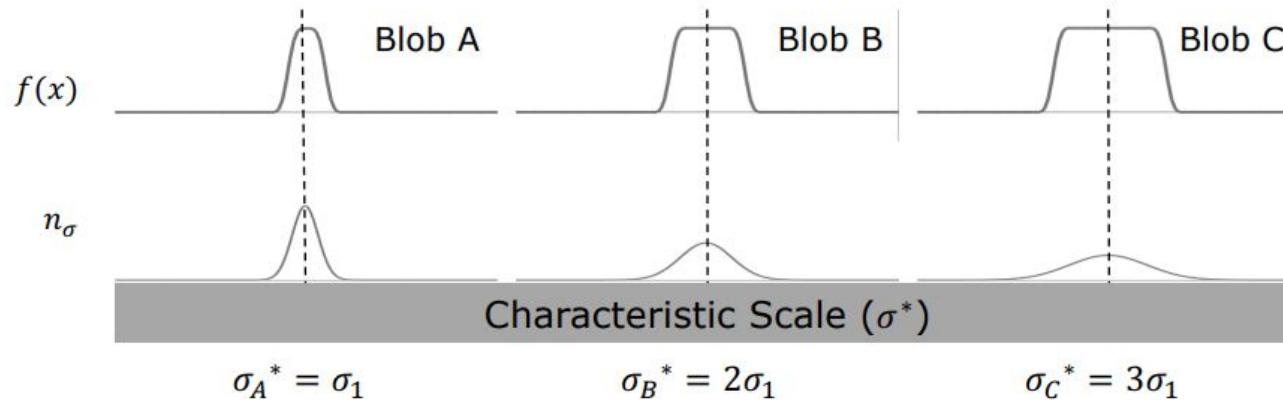
1D Blob and 2nd Derivative of Gaussian



1D Blob and 2nd Derivative of Gaussian



Characteristic Scale and Blob Size



Characteristic Scale: The σ at which σ -normalized 2nd derivative attains its extreme value.

Characteristic Scale \propto Size of Blob

$$\frac{\text{Size of Blob A}}{\text{Size of Blob B}} = \frac{\sigma_A^*}{\sigma_B^*}; \quad \frac{\text{Size of Blob B}}{\text{Size of Blob C}} = \frac{\sigma_B^*}{\sigma_C^*}$$

1D Blob Detection Summary

Given: 1D signal $f(x)$

Compute: $\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$ at many scales $(\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_k)$.

Find:
$$(x^*, \sigma^*) = \arg \max_{(x, \sigma)} \left| \sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x) \right|$$

x^* : Blob Position

σ^* : Characteristic Scale (Blob Size)

2D Blob Detector

Normalized Laplacian of Gaussian (NLoG) is used as the 2D equivalent for Blob Detection.

Laplacian

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

Gaussian



$$n_\sigma$$

LoG



$$\nabla^2 n_\sigma$$

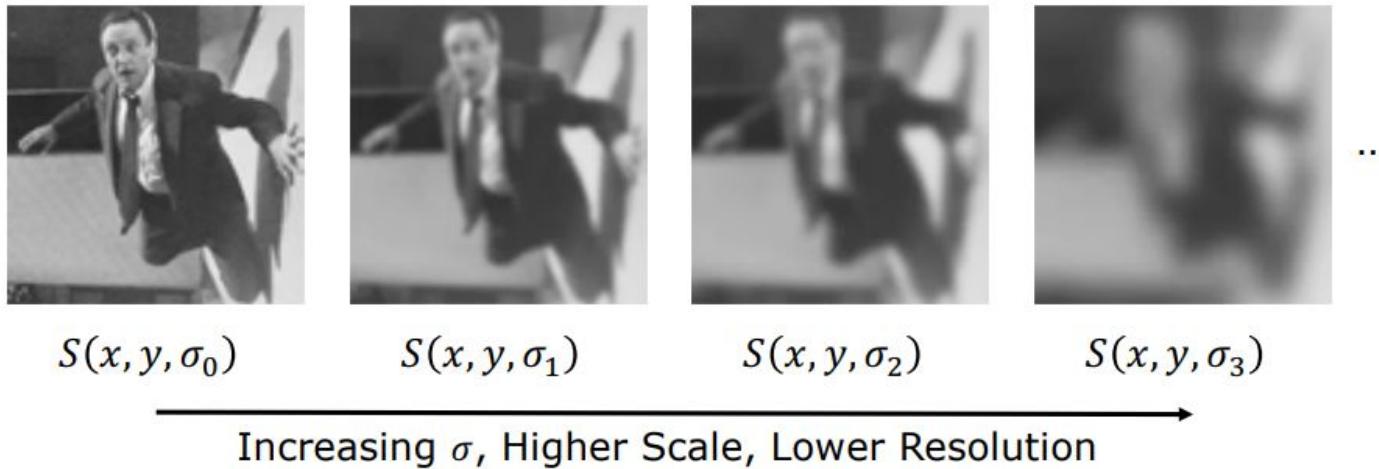
NLoG



$$\sigma^2 \nabla^2 n_\sigma$$

Location of Blobs given by Local Extrema after applying Normalized Laplacian of Gaussian at many scales.

Scale-Space



Scale Space: Stack created by filtering an image with Gaussians of different sigma (σ)

$$S(x, y, \sigma) = n(x, y, \sigma) * I(x, y)$$

Creating Scale-Space



$S(x, y, \sigma_0)$



$S(x, y, \sigma_1)$



$S(x, y, \sigma_2)$



$S(x, y, \sigma_3)$

Increasing σ , Higher Scale, Lower Resolution

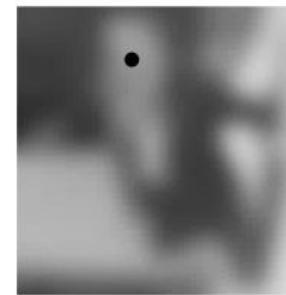
Selecting sigmas to generate the scale-space:

$$\boxed{\sigma_k = \sigma_0 s^k} \quad k = 0, 1, 2, 3, \dots$$

s : Constant multiplier

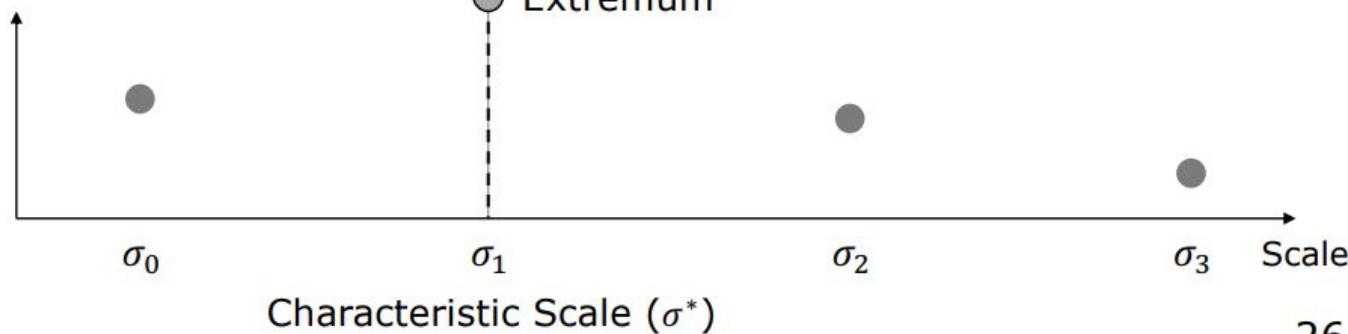
σ_0 : Initial Scale

Blob Detection using Local Extrema

 $S(x, y, \sigma_0)$  $S(x, y, \sigma_1)$  $S(x, y, \sigma_2)$  $S(x, y, \sigma_3)$

...

$$\sigma^2 \nabla^2 S(x, y, \sigma)$$
$$NLoG * I(x, y))$$



Blob Detection using Local Extrema



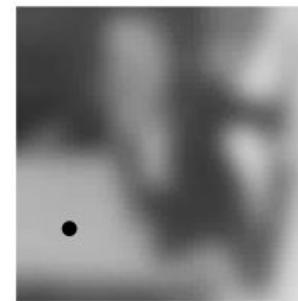
$$S(x, y, \sigma_0)$$



$$S(x, y, \sigma_1)$$



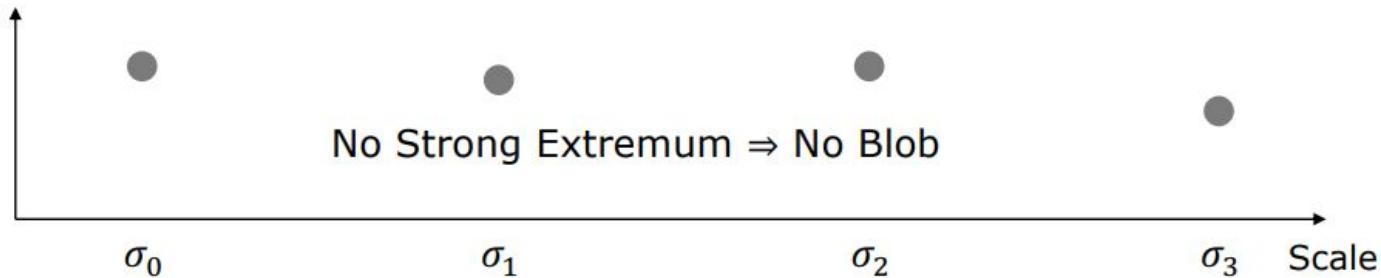
$$S(x, y, \sigma_2)$$



$$S(x, y, \sigma_3)$$

...

$$\begin{aligned} & \sigma^2 \nabla^2 S(x, y, \sigma) \\ & NLoG * I(x, y) \end{aligned}$$



2D Blob Detection Summary

Given an image $I(x, y)$

Convolve the image using NLoG at many scales σ

Find:

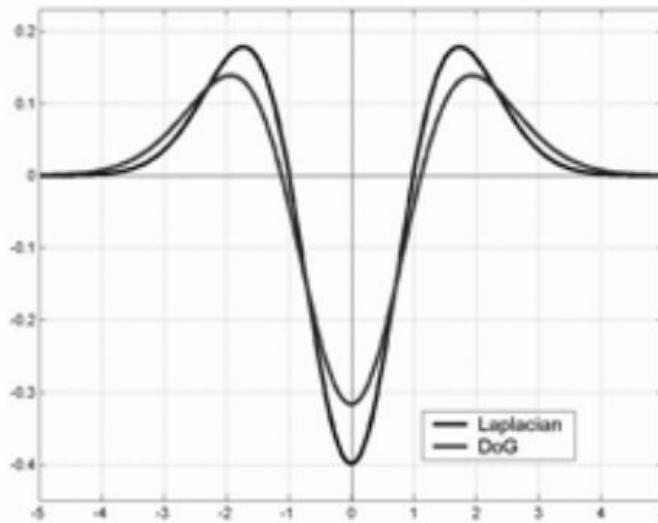
$$(x^*, y^*, \sigma^*) = \arg \max_{(x, y, \sigma)} |\sigma^2 \nabla^2 n_\sigma * I(x, y)|$$

(x^*, y^*) : Position of the blob

σ^* : Size of the blob

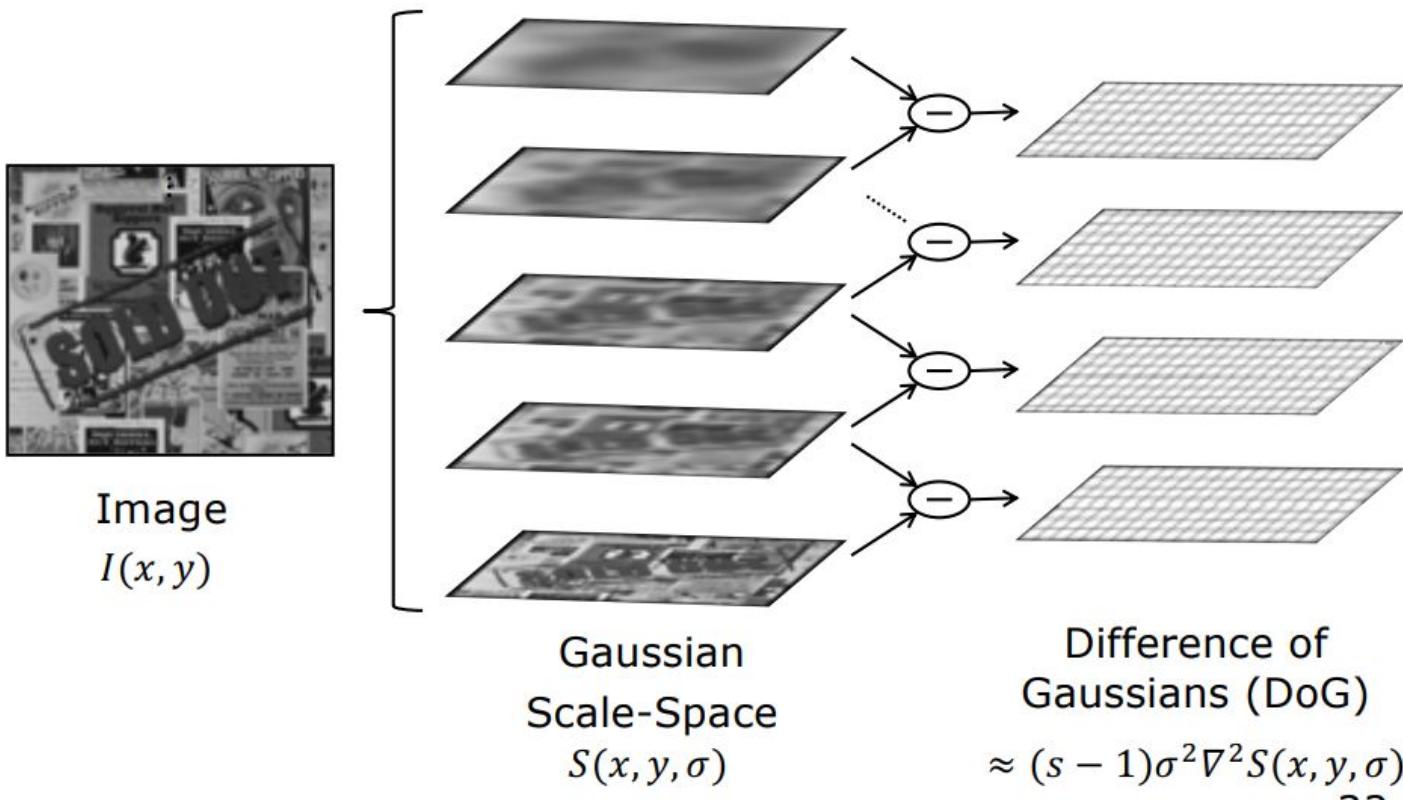
Fast NLoG Approximation: DoG

$$\text{Difference of Gaussian (DoG)} = (n_{s\sigma} - n_\sigma) \approx (s - 1) \underbrace{\sigma^2 \nabla^2 n_\sigma}_{\text{NLoG}}$$

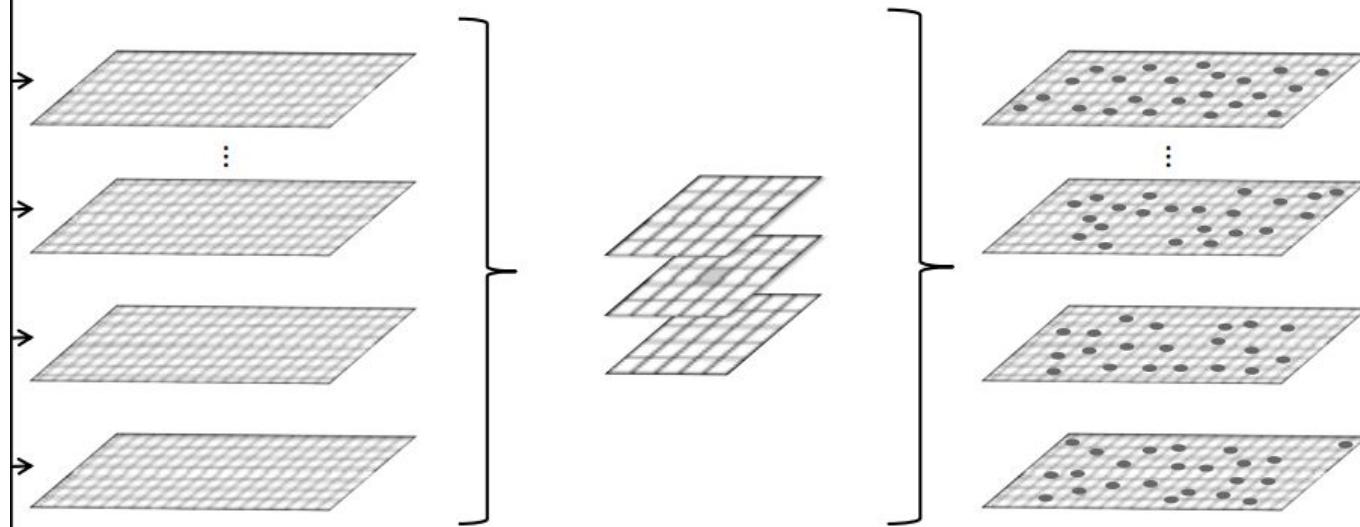


$$\text{DoG} \approx (s - 1) \text{ NLoG}$$

Extracting SIFT Interest Points



Extracting SIFT Interest Points



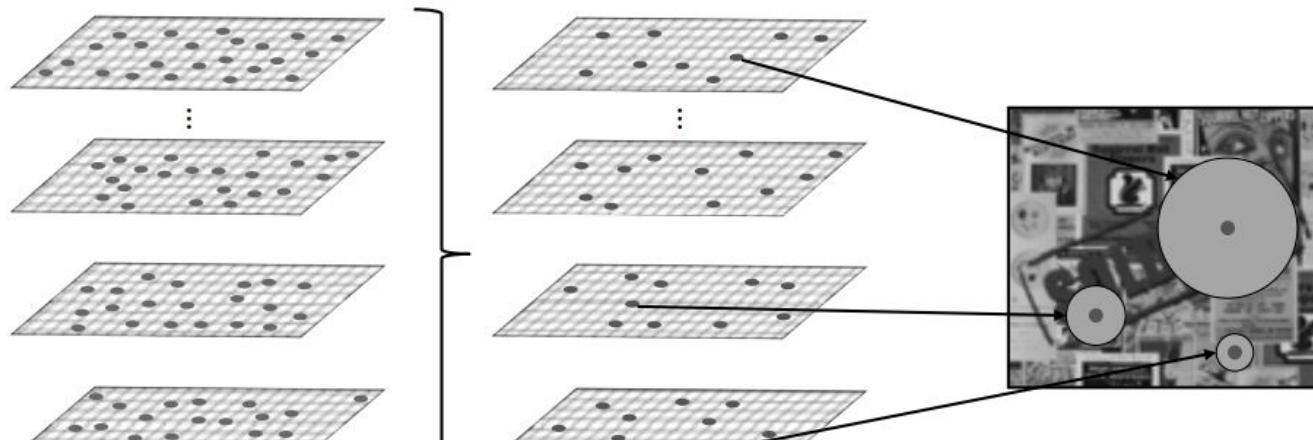
Difference of
Gaussians (DoG)

$$\approx (s - 1)\sigma^2 \nabla^2 S(x, y, \sigma)$$

Find Extremum
in every
3x3x3 grid

Interest Point
Candidates
(includes weak extrema)

Extracting SIFT Interest Points



Interest Point
Candidates
(includes weak extrema)

SIFT
Interest Points
(after removing
weak extrema)

[Lowe 2004]

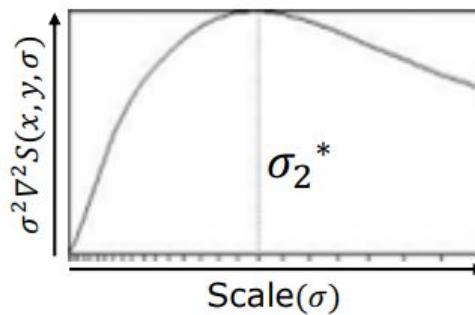
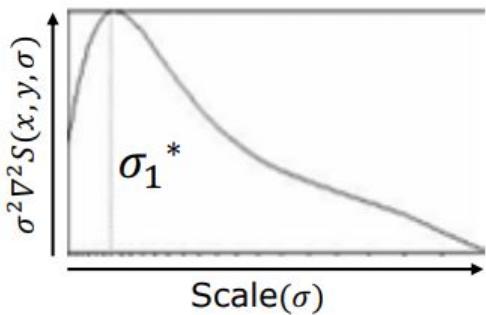
SIFT Detection Examples



SIFT Detection Examples



SIFT Scale Invariance



$$\frac{\sigma_1^*}{\sigma_2^*}$$
: Ratio of Blob Sizes

Computing the Principal Orientation

Use the histogram of gradient directions

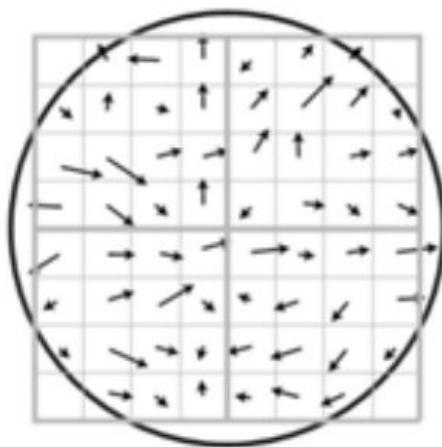
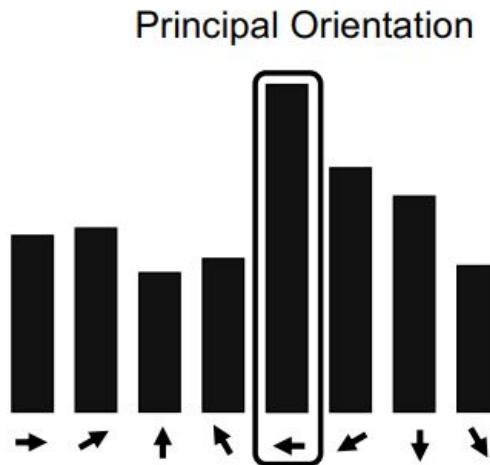


Image gradient directions

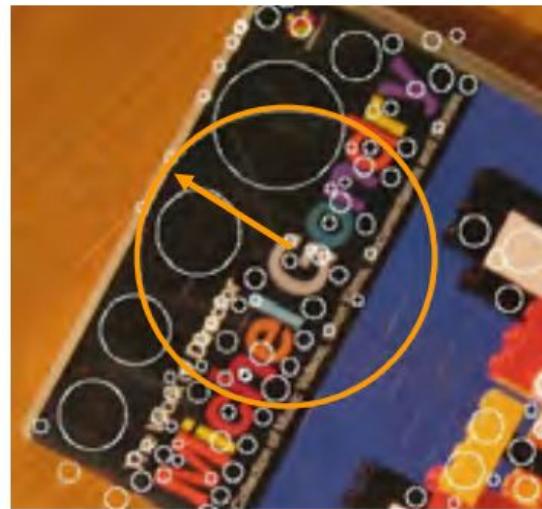
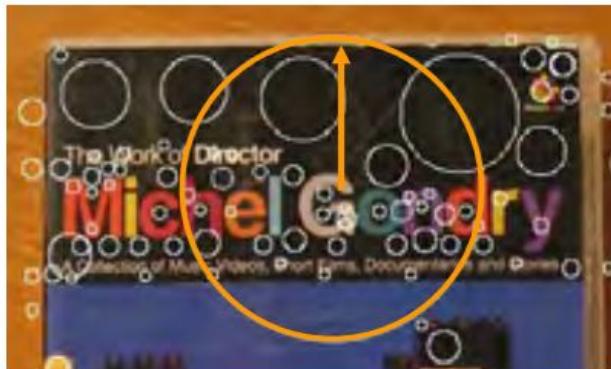
$$\theta = \tan^{-1} \left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x} \right)$$



Choose the most prominent gradient direction

SIFT Rotation Invariance

Use the principal orientation to undo rotation



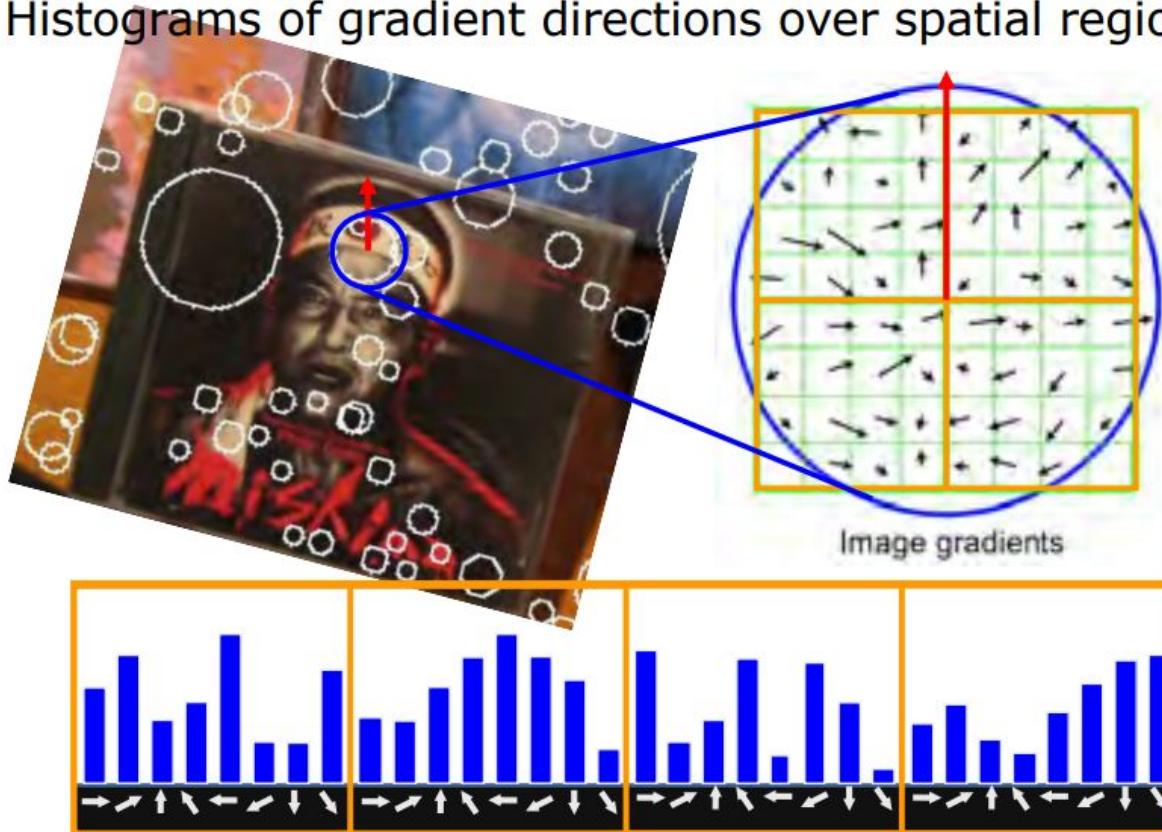
SIFT Descriptor

Histograms of gradient directions over spatial regions



SIFT Descriptor

Histograms of gradient directions over spatial regions



Comparing SIFT Descriptors

Essentially comparing two arrays of data.

Let $H_1(k)$ and $H_2(k)$ be two arrays of data of length N .

L2 Distance:

$$d(H_1, H_2) = \sqrt{\sum_k (H_1(k) - H_2(k))^2}$$

Smaller the distance metric, better the match.

Perfect match when $d(H_1, H_2) = 0$

Comparing SIFT Descriptors

Essentially comparing two arrays of data.

Let $H_1(k)$ and $H_2(k)$ be two arrays of data of length N .

Normalized Correlation:

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \bar{H}_1)(H_2(k) - \bar{H}_2)]}{\sqrt{\sum_k (H_1(k) - \bar{H}_1)^2} \sqrt{\sum_k (H_2(k) - \bar{H}_2)^2}}$$

$$\text{where: } \bar{H}_i = \frac{1}{N} \sum_{k=1}^N H_i(k)$$

Larger the distance metric, better the match.

Perfect match when $d(H_1, H_2) = 1$

Comparing SIFT Descriptors

Essentially comparing two arrays of data.

Let $H_1(k)$ and $H_2(k)$ be two arrays of data of length N .

Intersection:

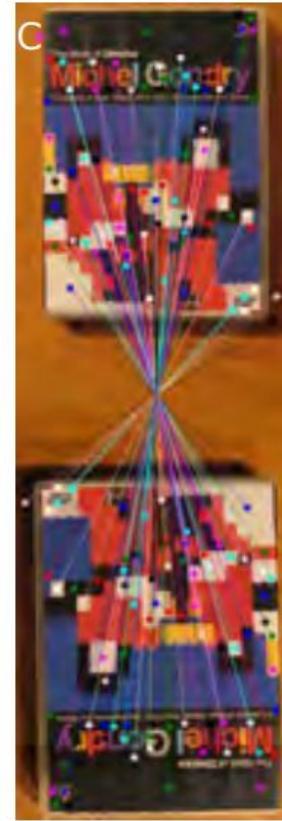
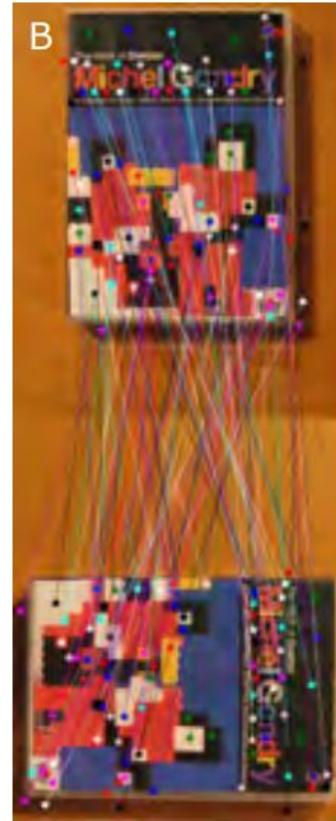
$$d(H_1, H_2) = \sum_k \min(H_1(k), H_2(k))$$

Larger the distance metric, better the match.

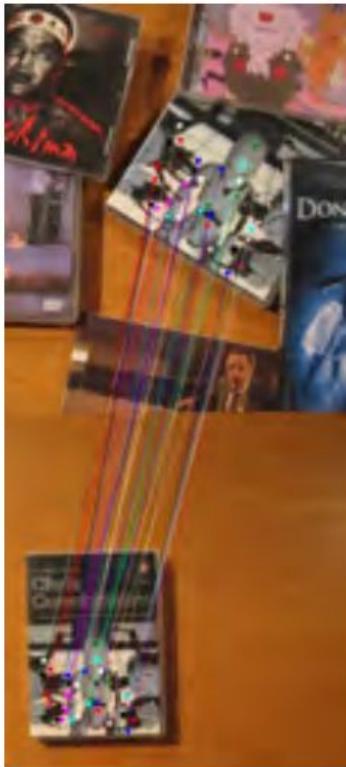
SIFT Results: Scale Invariance



SIFT Results: Rotation Invariance



SIFT Results: Robustness to Clutter



Panorama Stitching using SIFT



Image 1



Image 2



Panorama Stitching using SIFT



Warp and combine images to create a larger image

Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe

Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

January 5, 2004

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

Accepted for publication in the *International Journal of Computer Vision*, 2004.

1 Introduction

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This paper describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. **Scale-space extrema detection:** The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. **Keypoint localization:** At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.
3. **Orientation assignment:** One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.
4. **Keypoint descriptor:** The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features.

An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). The quantity of features is particularly important for object recognition, where the ability to detect small objects in cluttered backgrounds requires that at least 3 features be correctly matched from each object for reliable identification.

For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors. This paper will discuss fast nearest-neighbor algorithms that can perform this computation rapidly against large databases.

The keypoint descriptors are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered

image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. The correct matches can be filtered from the full set of matches by identifying subsets of keypoints that agree on the object and its location, scale, and orientation in the new image. The probability that several features will agree on these parameters by chance is much lower than the probability that any individual feature match will be in error. The determination of these consistent clusters can be performed rapidly by using an efficient hash table implementation of the generalized Hough transform.

Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed verification. First, a least-squared estimate is made for an affine approximation to the object pose. Any other image features consistent with this pose are identified, and outliers are discarded. Finally, a detailed computation is made of the probability that a particular set of features indicates the presence of an object, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

2 Related research

The development of image matching by using a set of local interest points can be traced back to the work of Moravec (1981) on stereo matching using a corner detector. The Moravec detector was improved by Harris and Stephens (1988) to make it more repeatable under small image variations and near edges. Harris also showed its value for efficient motion tracking and 3D structure from motion recovery (Harris, 1992), and the Harris corner detector has since been widely used for many other image matching tasks. While these feature detectors are usually called corner detectors, they are not selecting just corners, but rather any image location that has large gradients in all directions at a predetermined scale.

The initial applications were to stereo and short-range motion tracking, but the approach was later extended to more difficult problems. Zhang *et al.* (1995) showed that it was possible to match Harris corners over a large image range by using a correlation window around each corner to select likely matches. Outliers were then removed by solving for a fundamental matrix describing the geometric constraints between the two views of rigid scene and removing matches that did not agree with the majority solution. At the same time, a similar approach was developed by Torr (1995) for long-range motion matching, in which geometric constraints were used to remove outliers for rigid objects moving within an image.

The ground-breaking work of Schmid and Mohr (1997) showed that invariant local feature matching could be extended to general image recognition problems in which a feature was matched against a large database of images. They also used Harris corners to select interest points, but rather than matching with a correlation window, they used a rotationally invariant descriptor of the local image region. This allowed features to be matched under arbitrary orientation change between the two images. Furthermore, they demonstrated that multiple feature matches could accomplish general recognition under occlusion and clutter by identifying consistent clusters of matched features.

The Harris corner detector is very sensitive to changes in image scale, so it does not provide a good basis for matching images of different sizes. Earlier work by the author (Lowe, 1999) extended the local feature approach to achieve scale invariance. This work also described a new local descriptor that provided more distinctive features while being less

sensitive to local image distortions such as 3D viewpoint change. This current paper provides a more in-depth development and analysis of this earlier work, while also presenting a number of improvements in stability and feature invariance.

There is a considerable body of previous research on identifying representations that are stable under scale change. Some of the first work in this area was by Crowley and Parker (1984), who developed a representation that identified peaks and ridges in scale space and linked these into a tree structure. The tree structure could then be matched between images with arbitrary scale change. More recent work on graph-based matching by Shokoufandeh, Marsic and Dickinson (1999) provides more distinctive feature descriptors using wavelet coefficients. The problem of identifying an appropriate and consistent scale for feature detection has been studied in depth by Lindeberg (1993, 1994). He describes this as a problem of scale selection, and we make use of his results below.

Recently, there has been an impressive body of work on extending local features to be invariant to full affine transformations (Baumberg, 2000; Tuytelaars and Van Gool, 2000; Mikolajczyk and Schmid, 2002; Schaffalitzky and Zisserman, 2002; Brown and Lowe, 2002). This allows for invariant matching to features on a planar surface under changes in orthographic 3D projection, in most cases by resampling the image in a local affine frame. However, none of these approaches are yet fully affine invariant, as they start with initial feature scales and locations selected in a non-affine-invariant manner due to the prohibitive cost of exploring the full affine space. The affine frames are also more sensitive to noise than those of the scale-invariant features, so in practice the affine features have lower repeatability than the scale-invariant features unless the affine distortion is greater than about a 40 degree tilt of a planar surface (Mikolajczyk, 2002). Wider affine invariance may not be important for many applications, as training views are best taken at least every 30 degrees rotation in viewpoint (meaning that recognition is within 15 degrees of the closest training view) in order to capture non-planar changes and occlusion effects for 3D objects.

While the method to be presented in this paper is not fully affine invariant, a different approach is used in which the local descriptor allows relative feature positions to shift significantly with only small changes in the descriptor. This approach not only allows the descriptors to be reliably matched across a considerable range of affine distortion, but it also makes the features more robust against changes in 3D viewpoint for non-planar surfaces. Other advantages include much more efficient feature extraction and the ability to identify larger numbers of features. On the other hand, affine invariance is a valuable property for matching planar surfaces under very large view changes, and further research should be performed on the best ways to combine this with non-planar 3D viewpoint invariance in an efficient and stable manner.

Many other feature types have been proposed for use in recognition, some of which could be used in addition to the features described in this paper to provide further matches under differing circumstances. One class of features are those that make use of image contours or region boundaries, which should make them less likely to be disrupted by cluttered backgrounds near object boundaries. Matas *et al.*, (2002) have shown that their maximally-stable extremal regions can produce large numbers of matching features with good stability. Mikolajczyk *et al.*, (2003) have developed a new descriptor that uses local edges while ignoring unrelated nearby edges, providing the ability to find stable features even near the boundaries of narrow shapes superimposed on background clutter. Nelson and Selinger (1998) have shown good results with local features based on groupings of image contours. Similarly,

Pope and Lowe (2000) used features based on the hierarchical grouping of image contours, which are particularly useful for objects lacking detailed texture.

The history of research on visual recognition contains work on a diverse set of other image properties that can be used as feature measurements. Carneiro and Jepson (2002) describe phase-based local features that represent the phase rather than the magnitude of local spatial frequencies, which is likely to provide improved invariance to illumination. Schiele and Crowley (2000) have proposed the use of multidimensional histograms summarizing the distribution of measurements within image regions. This type of feature may be particularly useful for recognition of textured objects with deformable shapes. Basri and Jacobs (1997) have demonstrated the value of extracting local region boundaries for recognition. Other useful properties to incorporate include color, motion, figure-ground discrimination, region shape descriptors, and stereo depth cues. The local feature approach can easily incorporate novel feature types because extra features contribute to robustness when they provide correct matches, but otherwise do little harm other than their cost of computation. Therefore, future systems are likely to combine many feature types.

3 Detection of scale-space extrema

As described in the introduction, we will detect keypoints using a cascade filtering approach that uses efficient algorithms to identify candidate locations that are then examined in further detail. The first stage of keypoint detection is to identify locations and scales that can be repeatably assigned under differing views of the same object. Detecting locations that are invariant to scale change of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as scale space (Witkin, 1983).

It has been shown by Koenderink (1984) and Lindeberg (1994) that under a variety of reasonable assumptions the only possible scale-space kernel is the Gaussian function. Therefore, the scale space of an image is defined as a function, $L(x, y, \sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with an input image, $I(x, y)$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

where $*$ is the convolution operation in x and y , and

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

To efficiently detect stable keypoint locations in scale space, we have proposed (Lowe, 1999) using scale-space extrema in the difference-of-Gaussian function convolved with the image, $D(x, y, \sigma)$, which can be computed from the difference of two nearby scales separated by a constant multiplicative factor k :

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned} \tag{1}$$

There are a number of reasons for choosing this function. First, it is a particularly efficient function to compute, as the smoothed images, L , need to be computed in any case for scale space feature description, and D can therefore be computed by simple image subtraction.

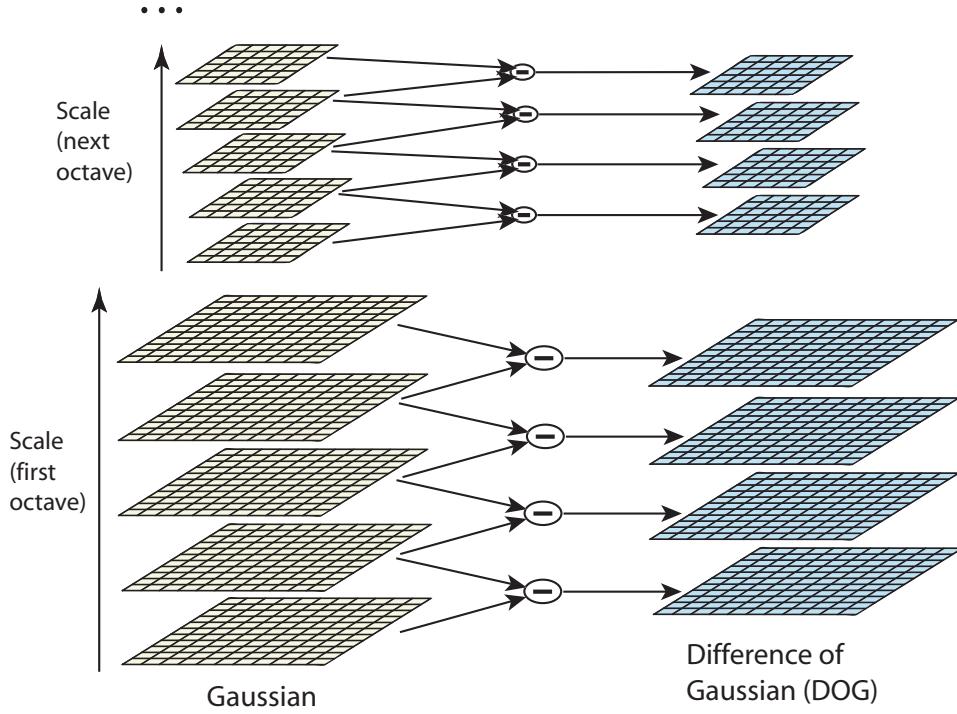


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

In addition, the difference-of-Gaussian function provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2 \nabla^2 G$, as studied by Lindeberg (1994). Lindeberg showed that the normalization of the Laplacian with the factor σ^2 is required for true scale invariance. In detailed experimental comparisons, Mikolajczyk (2002) found that the maxima and minima of $\sigma^2 \nabla^2 G$ produce the most stable image features compared to a range of other possible image functions, such as the gradient, Hessian, or Harris corner function.

The relationship between D and $\sigma^2 \nabla^2 G$ can be understood from the heat diffusion equation (parameterized in terms of σ rather than the more usual $t = \sigma^2$):

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G.$$

From this, we see that $\nabla^2 G$ can be computed from the finite difference approximation to $\partial G / \partial \sigma$, using the difference of nearby scales at $k\sigma$ and σ :

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

and therefore,

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G.$$

This shows that when the difference-of-Gaussian function has scales differing by a constant factor it already incorporates the σ^2 scale normalization required for the scale-invariant

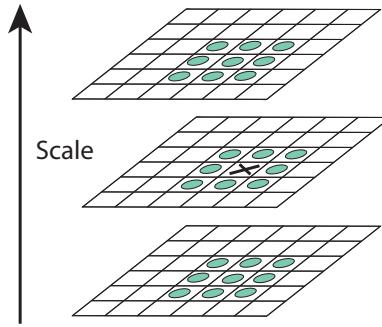


Figure 2: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3×3 regions at the current and adjacent scales (marked with circles).

Laplacian. The factor $(k - 1)$ in the equation is a constant over all scales and therefore does not influence extrema location. The approximation error will go to zero as k goes to 1, but in practice we have found that the approximation has almost no impact on the stability of extrema detection or localization for even significant differences in scale, such as $k = \sqrt{2}$.

An efficient approach to construction of $D(x, y, \sigma)$ is shown in Figure 1. The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s , of intervals, so $k = 2^{1/s}$. We must produce $s + 3$ images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave. Adjacent image scales are subtracted to produce the difference-of-Gaussian images shown on the right. Once a complete octave has been processed, we resample the Gaussian image that has twice the initial value of σ (it will be 2 images from the top of the stack) by taking every second pixel in each row and column. The accuracy of sampling relative to σ is no different than for the start of the previous octave, while computation is greatly reduced.

3.1 Local extrema detection

In order to detect the local maxima and minima of $D(x, y, \sigma)$, each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below (see Figure 2). It is selected only if it is larger than all of these neighbors or smaller than all of them. The cost of this check is reasonably low due to the fact that most sample points will be eliminated following the first few checks.

An important issue is to determine the frequency of sampling in the image and scale domains that is needed to reliably detect the extrema. Unfortunately, it turns out that there is no minimum spacing of samples that will detect all extrema, as the extrema can be arbitrarily close together. This can be seen by considering a white circle on a black background, which will have a single scale space maximum where the circular positive central region of the difference-of-Gaussian function matches the size and location of the circle. For a very elongated ellipse, there will be two maxima near each end of the ellipse. As the locations of maxima are a continuous function of the image, for some ellipse with intermediate elongation there will be a transition from a single maximum to two, with the maxima arbitrarily close to

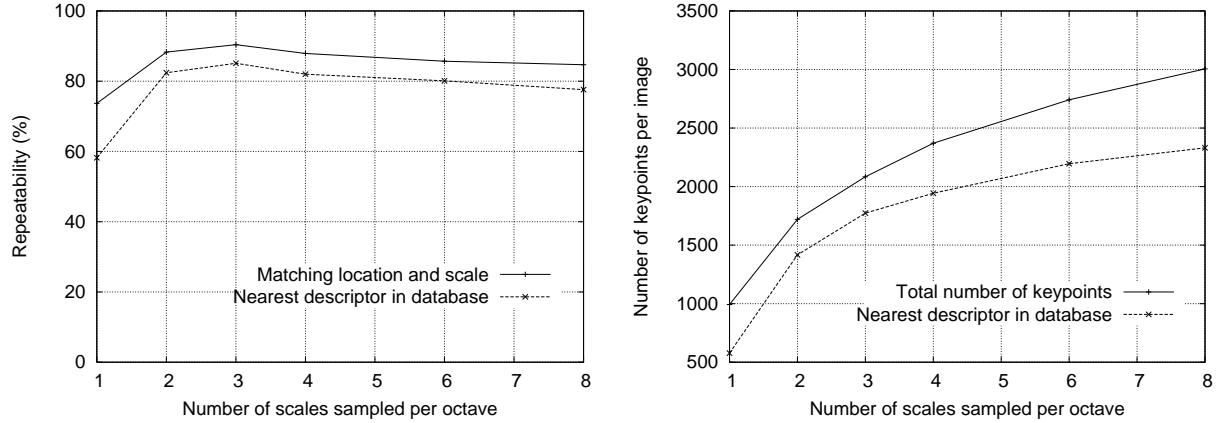


Figure 3: The top line of the first graph shows the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The lower line shows the percent of keypoints that have their descriptors correctly matched to a large database. The second graph shows the total number of keypoints detected in a typical image as a function of the number of scale samples.

each other near the transition.

Therefore, we must settle for a solution that trades off efficiency with completeness. In fact, as might be expected and is confirmed by our experiments, extrema that are close together are quite unstable to small perturbations of the image. We can determine the best choices experimentally by studying a range of sampling frequencies and using those that provide the most reliable results under a realistic simulation of the matching task.

3.2 Frequency of sampling in scale

The experimental determination of sampling frequency that maximizes extrema stability is shown in Figures 3 and 4. These figures (and most other simulations in this paper) are based on a matching task using a collection of 32 real images drawn from a diverse range, including outdoor scenes, human faces, aerial photographs, and industrial images (the image domain was found to have almost no influence on any of the results). Each image was then subject to a range of transformations, including rotation, scaling, affine stretch, change in brightness and contrast, and addition of image noise. Because the changes were synthetic, it was possible to precisely predict where each feature in an original image should appear in the transformed image, allowing for measurement of correct repeatability and positional accuracy for each feature.

Figure 3 shows these simulation results used to examine the effect of varying the number of scales per octave at which the image function is sampled prior to extrema detection. In this case, each image was resampled following rotation by a random angle and scaling by a random amount between 0.2 and 0.9 times the original size. Keypoints from the reduced resolution image were matched against those from the original image so that the scales for all keypoints would be present in the matched image. In addition, 1% image noise was added, meaning that each pixel had a random number added from the uniform interval [-0.01, 0.01] where pixel values are in the range [0,1] (equivalent to providing slightly less than 6 bits of accuracy for image pixels).

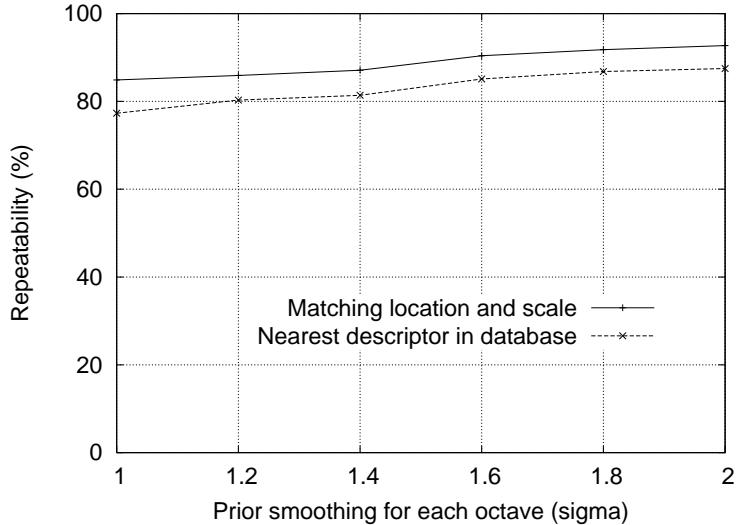


Figure 4: The top line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The lower line shows the percent of descriptors correctly matched against a large database.

The top line in the first graph of Figure 3 shows the percent of keypoints that are detected at a matching location and scale in the transformed image. For all examples in this paper, we define a matching scale as being within a factor of $\sqrt{2}$ of the correct scale, and a matching location as being within σ pixels, where σ is the scale of the keypoint (defined from equation (1) as the standard deviation of the smallest Gaussian used in the difference-of-Gaussian function). The lower line on this graph shows the number of keypoints that are correctly matched to a database of 40,000 keypoints using the nearest-neighbor matching procedure to be described in Section 6 (this shows that once the keypoint is repeatably located, it is likely to be useful for recognition and matching tasks). As this graph shows, the highest repeatability is obtained when sampling 3 scales per octave, and this is the number of scale samples used for all other experiments throughout this paper.

It might seem surprising that the repeatability does not continue to improve as more scales are sampled. The reason is that this results in many more local extrema being detected, but these extrema are on average less stable and therefore are less likely to be detected in the transformed image. This is shown by the second graph in Figure 3, which shows the average number of keypoints detected and correctly matched in each image. The number of keypoints rises with increased sampling of scales and the total number of correct matches also rises. Since the success of object recognition often depends more on the quantity of correctly matched keypoints, as opposed to their percentage correct matching, for many applications it will be optimal to use a larger number of scale samples. However, the cost of computation also rises with this number, so for the experiments in this paper we have chosen to use just 3 scale samples per octave.

To summarize, these experiments show that the scale-space difference-of-Gaussian function has a large number of extrema and that it would be very expensive to detect them all. Fortunately, we can detect the most stable and useful subset even with a coarse sampling of scales.

3.3 Frequency of sampling in the spatial domain

Just as we determined the frequency of sampling per octave of scale space, so we must determine the frequency of sampling in the image domain relative to the scale of smoothing. Given that extrema can be arbitrarily close together, there will be a similar trade-off between sampling frequency and rate of detection. Figure 4 shows an experimental determination of the amount of prior smoothing, σ , that is applied to each image level before building the scale space representation for an octave. Again, the top line is the repeatability of keypoint detection, and the results show that the repeatability continues to increase with σ . However, there is a cost to using a large σ in terms of efficiency, so we have chosen to use $\sigma = 1.6$, which provides close to optimal repeatability. This value is used throughout this paper and was used for the results in Figure 3.

Of course, if we pre-smooth the image before extrema detection, we are effectively discarding the highest spatial frequencies. Therefore, to make full use of the input, the image can be expanded to create more sample points than were present in the original. We double the size of the input image using linear interpolation prior to building the first level of the pyramid. While the equivalent operation could effectively have been performed by using sets of subpixel-offset filters on the original image, the image doubling leads to a more efficient implementation. We assume that the original image has a blur of at least $\sigma = 0.5$ (the minimum needed to prevent significant aliasing), and that therefore the doubled image has $\sigma = 1.0$ relative to its new pixel spacing. This means that little additional smoothing is needed prior to creation of the first octave of scale space. The image doubling increases the number of stable keypoints by almost a factor of 4, but no significant further improvements were found with a larger expansion factor.

4 Accurate keypoint localization

Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

The initial implementation of this approach (Lowe, 1999) simply located keypoints at the location and scale of the central sample point. However, recently Brown has developed a method (Brown and Lowe, 2002) for fitting a 3D quadratic function to the local sample points to determine the interpolated location of the maximum, and his experiments showed that this provides a substantial improvement to matching and stability. His approach uses the Taylor expansion (up to the quadratic terms) of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point:

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

where D and its derivatives are evaluated at the sample point and $\mathbf{x} = (x, y, \sigma)^T$ is the offset from this point. The location of the extremum, $\hat{\mathbf{x}}$, is determined by taking the derivative of this function with respect to \mathbf{x} and setting it to zero, giving

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2} \frac{\partial D}{\partial \mathbf{x}}. \quad (3)$$

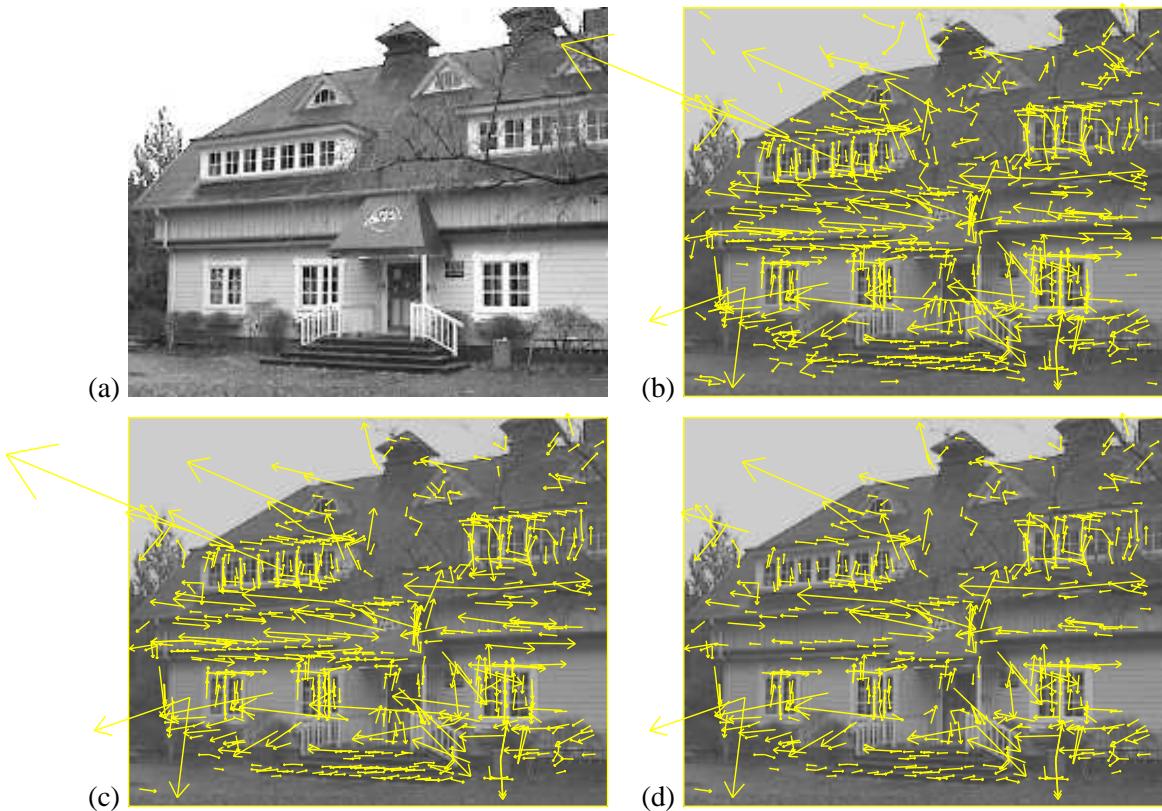


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures.

As suggested by Brown, the Hessian and derivative of D are approximated by using differences of neighboring sample points. The resulting 3x3 linear system can be solved with minimal cost. If the offset $\hat{\mathbf{x}}$ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation performed instead about that point. The final offset $\hat{\mathbf{x}}$ is added to the location of its sample point to get the interpolated estimate for the location of the extremum.

The function value at the extremum, $D(\hat{\mathbf{x}})$, is useful for rejecting unstable extrema with low contrast. This can be obtained by substituting equation (3) into (2), giving

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

For the experiments in this paper, all extrema with a value of $|D(\hat{\mathbf{x}})|$ less than 0.03 were discarded (as before, we assume image pixel values in the range [0,1]).

Figure 5 shows the effects of keypoint selection on a natural image. In order to avoid too much clutter, a low-resolution 233 by 189 pixel image is used and keypoints are shown as vectors giving the location, scale, and orientation of each keypoint (orientation assignment is described below). Figure 5 (a) shows the original image, which is shown at reduced contrast behind the subsequent figures. Figure 5 (b) shows the 832 keypoints at all detected maxima

and minima of the difference-of-Gaussian function, while (c) shows the 729 keypoints that remain following removal of those with a value of $|D(\hat{\mathbf{x}})|$ less than 0.03. Part (d) will be explained in the following section.

4.1 Eliminating edge responses

For stability, it is not sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.

A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction. The principal curvatures can be computed from a 2x2 Hessian matrix, \mathbf{H} , computed at the location and scale of the keypoint:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (4)$$

The derivatives are estimated by taking differences of neighboring sample points.

The eigenvalues of \mathbf{H} are proportional to the principal curvatures of D . Borrowing from the approach used by Harris and Stephens (1988), we can avoid explicitly computing the eigenvalues, as we are only concerned with their ratio. Let α be the eigenvalue with the largest magnitude and β be the smaller one. Then, we can compute the sum of the eigenvalues from the trace of \mathbf{H} and their product from the determinant:

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta, \\ \text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

In the unlikely event that the determinant is negative, the curvatures have different signs so the point is discarded as not being an extremum. Let r be the ratio between the largest magnitude eigenvalue and the smaller one, so that $\alpha = r\beta$. Then,

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

which depends only on the ratio of the eigenvalues rather than their individual values. The quantity $(r+1)^2/r$ is at a minimum when the two eigenvalues are equal and it increases with r . Therefore, to check that the ratio of principal curvatures is below some threshold, r , we only need to check

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}.$$

This is very efficient to compute, with less than 20 floating point operations required to test each keypoint. The experiments in this paper use a value of $r = 10$, which eliminates keypoints that have a ratio between the principal curvatures greater than 10. The transition from Figure 5 (c) to (d) shows the effects of this operation.

5 Orientation assignment

By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation. This approach contrasts with the orientation invariant descriptors of Schmid and Mohr (1997), in which each image property is based on a rotationally invariant measure. The disadvantage of that approach is that it limits the descriptors that can be used and discards image information by not requiring all measures to be based on a consistent rotation.

Following experimentation with a number of approaches to assigning a local orientation, the following approach was found to give the most stable results. The scale of the keypoint is used to select the Gaussian smoothed image, L , with the closest scale, so that all computations are performed in a scale-invariant manner. For each image sample, $L(x, y)$, at this scale, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, is precomputed using pixel differences:

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.

Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations. Only about 15% of points are assigned multiple orientations, but these contribute significantly to the stability of matching. Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.

Figure 6 shows the experimental stability of location, scale, and orientation assignment under differing amounts of image noise. As before the images are rotated and scaled by random amounts. The top line shows the stability of keypoint location and scale assignment. The second line shows the stability of matching when the orientation assignment is also required to be within 15 degrees. As shown by the gap between the top two lines, the orientation assignment remains accurate 95% of the time even after addition of $\pm 10\%$ pixel noise (equivalent to a camera providing less than 3 bits of precision). The measured variance of orientation for the correct matches is about 2.5 degrees, rising to 3.9 degrees for 10% noise. The bottom line in Figure 6 shows the final accuracy of correctly matching a keypoint descriptor to a database of 40,000 keypoints (to be discussed below). As this graph shows, the SIFT features are resistant to even large amounts of pixel noise, and the major cause of error is the initial location and scale detection.

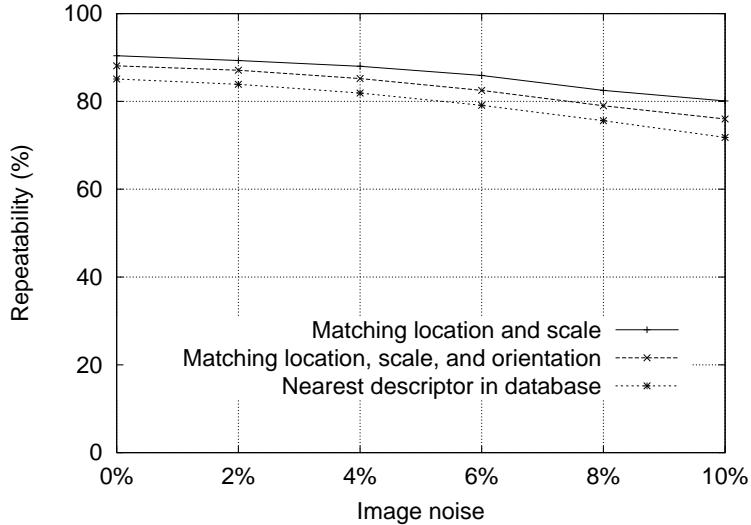


Figure 6: The top line in the graph shows the percent of keypoint locations and scales that are repeatably detected as a function of pixel noise. The second line shows the repeatability after also requiring agreement in orientation. The bottom line shows the final percent of descriptors correctly matched to a large database.

6 The local image descriptor

The previous operations have assigned an image location, scale, and orientation to each key-point. These parameters impose a repeatable local 2D coordinate system in which to describe the local image region, and therefore provide invariance to these parameters. The next step is to compute a descriptor for the local image region that is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

One obvious approach would be to sample the local image intensities around the key-point at the appropriate scale, and to match these using a normalized correlation measure. However, simple correlation of image patches is highly sensitive to changes that cause mis-registration of samples, such as affine or 3D viewpoint change or non-rigid deformations. A better approach has been demonstrated by Edelman, Intrator, and Poggio (1997). Their proposed representation was based upon a model of biological vision, in particular of complex neurons in primary visual cortex. These complex neurons respond to a gradient at a particular orientation and spatial frequency, but the location of the gradient on the retina is allowed to shift over a small receptive field rather than being precisely localized. Edelman *et al.* hypothesized that the function of these complex neurons was to allow for matching and recognition of 3D objects from a range of viewpoints. They have performed detailed experiments using 3D computer models of object and animal shapes which show that matching gradients while allowing for shifts in their position results in much better classification under 3D rotation. For example, recognition accuracy for 3D objects rotated in depth by 20 degrees increased from 35% for correlation of gradients to 94% using the complex cell model. Our implementation described below was inspired by this idea, but allows for positional shift using a different computational mechanism.

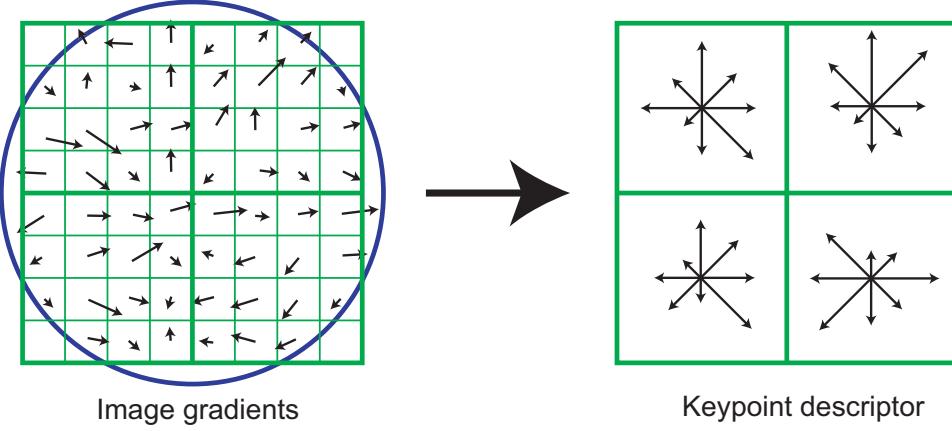


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4×4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2×2 descriptor array computed from an 8×8 set of samples, whereas the experiments in this paper use 4×4 descriptors computed from a 16×16 sample array.

6.1 Descriptor representation

Figure 7 illustrates the computation of the keypoint descriptor. First the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. For efficiency, the gradients are precomputed for all levels of the pyramid as described in Section 5. These are illustrated with small arrows at each sample location on the left side of Figure 7.

A Gaussian weighting function with σ equal to one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point. This is illustrated with a circular window on the left side of Figure 7, although, of course, the weight falls off smoothly. The purpose of this Gaussian window is to avoid sudden changes in the descriptor with small changes in the position of the window, and to give less emphasis to gradients that are far from the center of the descriptor, as these are most affected by misregistration errors.

The keypoint descriptor is shown on the right side of Figure 7. It allows for significant shift in gradient positions by creating orientation histograms over 4×4 sample regions. The figure shows eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of that histogram entry. A gradient sample on the left can shift up to 4 sample positions while still contributing to the same histogram on the right, thereby achieving the objective of allowing for larger local positional shifts.

It is important to avoid all boundary affects in which the descriptor abruptly changes as a sample shifts smoothly from being within one histogram to another or from one orientation to another. Therefore, trilinear interpolation is used to distribute the value of each gradient sample into adjacent histogram bins. In other words, each entry into a bin is multiplied by a weight of $1 - d$ for each dimension, where d is the distance of the sample from the central value of the bin as measured in units of the histogram bin spacing.

The descriptor is formed from a vector containing the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of Figure 7. The figure shows a 2×2 array of orientation histograms, whereas our experiments below show that the best results are achieved with a 4×4 array of histograms with 8 orientation bins in each. Therefore, the experiments in this paper use a $4 \times 4 \times 8 = 128$ element feature vector for each keypoint.

Finally, the feature vector is modified to reduce the effects of illumination change. First, the vector is normalized to unit length. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in relative magnitudes for some gradients, but are less likely to affect the gradient orientations. Therefore, we reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then renormalizing to unit length. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis. The value of 0.2 was determined experimentally using images containing differing illuminations for the same 3D objects.

6.2 Descriptor testing

There are two parameters that can be used to vary the complexity of the descriptor: the number of orientations, r , in the histograms, and the width, n , of the $n \times n$ array of orientation histograms. The size of the resulting descriptor vector is rn^2 . As the complexity of the descriptor grows, it will be able to discriminate better in a large database, but it will also be more sensitive to shape distortions and occlusion.

Figure 8 shows experimental results in which the number of orientations and size of the descriptor were varied. The graph was generated for a viewpoint transformation in which a planar surface is tilted by 50 degrees away from the viewer and 4% image noise is added. This is near the limits of reliable matching, as it is in these more difficult cases that descriptor performance is most important. The results show the percent of keypoints that find a correct match to the single closest neighbor among a database of 40,000 keypoints. The graph shows that a single orientation histogram ($n = 1$) is very poor at discriminating, but the results continue to improve up to a 4×4 array of histograms with 8 orientations. After that, adding more orientations or a larger descriptor can actually hurt matching by making the descriptor more sensitive to distortion. These results were broadly similar for other degrees of viewpoint change and noise, although in some simpler cases discrimination continued to improve (from already high levels) with 5×5 and higher descriptor sizes. Throughout this paper we use a 4×4 descriptor with 8 orientations, resulting in feature vectors with 128 dimensions. While the dimensionality of the descriptor may seem high, we have found that it consistently performs better than lower-dimensional descriptors on a range of matching tasks and that the computational cost of matching remains low when using the approximate nearest-neighbor methods described below.

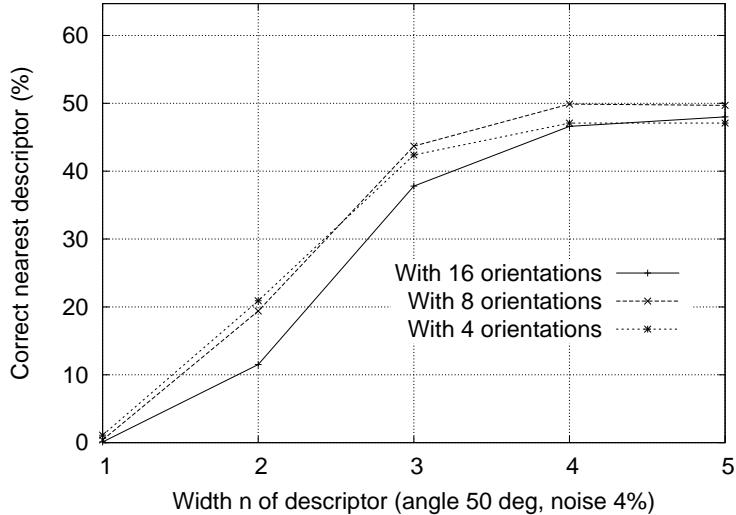


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

6.3 Sensitivity to affine change

The sensitivity of the descriptor to affine change is examined in Figure 9. The graph shows the reliability of keypoint location and scale selection, orientation assignment, and nearest-neighbor matching to a database as a function of rotation in depth of a plane away from a viewer. It can be seen that each stage of computation has reduced repeatability with increasing affine distortion, but that the final matching accuracy remains above 50% out to a 50 degree change in viewpoint.

To achieve reliable matching over a wider viewpoint angle, one of the affine-invariant detectors could be used to select and resample image regions, as discussed in Section 2. As mentioned there, none of these approaches is truly affine-invariant, as they all start from initial feature locations determined in a non-affine-invariant manner. In what appears to be the most affine-invariant method, Mikolajczyk (2002) has proposed and run detailed experiments with the Harris-affine detector. He found that its keypoint repeatability is below that given here out to about a 50 degree viewpoint angle, but that it then retains close to 40% repeatability out to an angle of 70 degrees, which provides better performance for extreme affine changes. The disadvantages are a much higher computational cost, a reduction in the number of keypoints, and poorer stability for small affine changes due to errors in assigning a consistent affine frame under noise. In practice, the allowable range of rotation for 3D objects is considerably less than for planar surfaces, so affine invariance is usually not the limiting factor in the ability to match across viewpoint change. If a wide range of affine invariance is desired, such as for a surface that is known to be planar, then a simple solution is to adopt the approach of Pritchard and Heidrich (2003) in which additional SIFT features are generated from 4 affine-transformed versions of the training image corresponding to 60 degree viewpoint changes. This allows for the use of standard SIFT features with no additional cost when processing the image to be recognized, but results in an increase in the size of the feature database by a factor of 3.

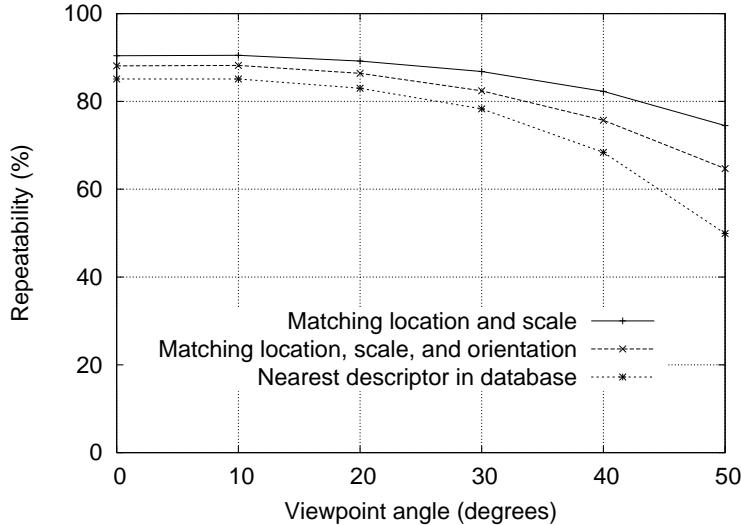


Figure 9: This graph shows the stability of detection for keypoint location, orientation, and final matching to a database as a function of affine distortion. The degree of affine distortion is expressed in terms of the equivalent viewpoint rotation in depth for a planar surface.

6.4 Matching to large databases

An important remaining issue for measuring the distinctiveness of features is how the reliability of matching varies as a function of the number of features in the database being matched. Most of the examples in this paper are generated using a database of 32 images with about 40,000 keypoints. Figure 10 shows how the matching reliability varies as a function of database size. This figure was generated using a larger database of 112 images, with a viewpoint depth rotation of 30 degrees and 2% image noise in addition to the usual random image rotation and scale change.

The dashed line shows the portion of image features for which the nearest neighbor in the database was the correct match, as a function of database size shown on a logarithmic scale. The leftmost point is matching against features from only a single image while the rightmost point is selecting matches from a database of all features from the 112 images. It can be seen that matching reliability does decrease as a function of the number of distractors, yet all indications are that many correct matches will continue to be found out to very large database sizes.

The solid line is the percentage of keypoints that were identified at the correct matching location and orientation in the transformed image, so it is only these points that have any chance of having matching descriptors in the database. The reason this line is flat is that the test was run over the full database for each value, while only varying the portion of the database used for distractors. It is of interest that the gap between the two lines is small, indicating that matching failures are due more to issues with initial feature localization and orientation assignment than to problems with feature distinctiveness, even out to large database sizes.

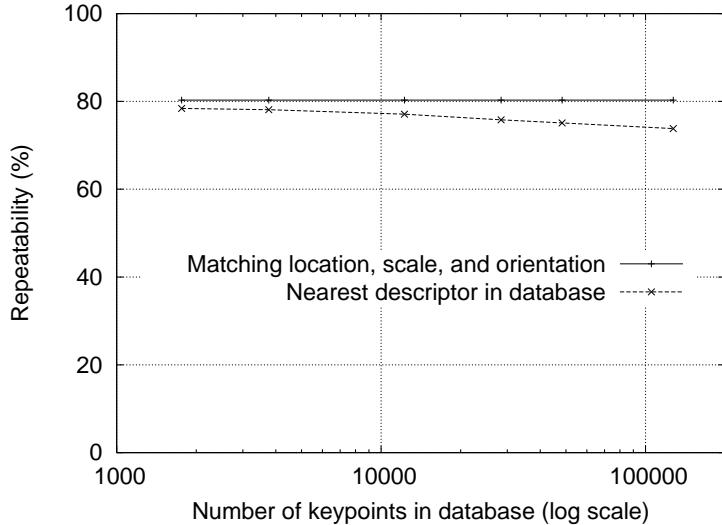


Figure 10: The dashed line shows the percent of keypoints correctly matched to a database as a function of database size (using a logarithmic scale). The solid line shows the percent of keypoints assigned the correct location, scale, and orientation. Images had random scale and rotation changes, an affine transform of 30 degrees, and image noise of 2% added prior to matching.

7 Application to object recognition

The major topic of this paper is the derivation of distinctive invariant keypoints, as described above. To demonstrate their application, we will now give a brief description of their use for object recognition in the presence of clutter and occlusion. More details on applications of these features to recognition are available in other papers (Lowe, 1999; Lowe, 2001; Se, Lowe and Little, 2002).

Object recognition is performed by first matching each keypoint independently to the database of keypoints extracted from training images. Many of these initial matches will be incorrect due to ambiguous features or features that arise from background clutter. Therefore, clusters of at least 3 features are first identified that agree on an object and its pose, as these clusters have a much higher probability of being correct than individual feature matches. Then, each cluster is checked by performing a detailed geometric fit to the model, and the result is used to accept or reject the interpretation.

7.1 Keypoint matching

The best candidate match for each keypoint is found by identifying its nearest neighbor in the database of keypoints from training images. The nearest neighbor is defined as the keypoint with minimum Euclidean distance for the invariant descriptor vector as was described in Section 6.

However, many features from an image will not have any correct match in the training database because they arise from background clutter or were not detected in the training images. Therefore, it would be useful to have a way to discard features that do not have any good match to the database. A global threshold on distance to the closest feature does not perform well, as some descriptors are much more discriminative than others. A more effective measure is obtained by comparing the distance of the closest neighbor to that of the

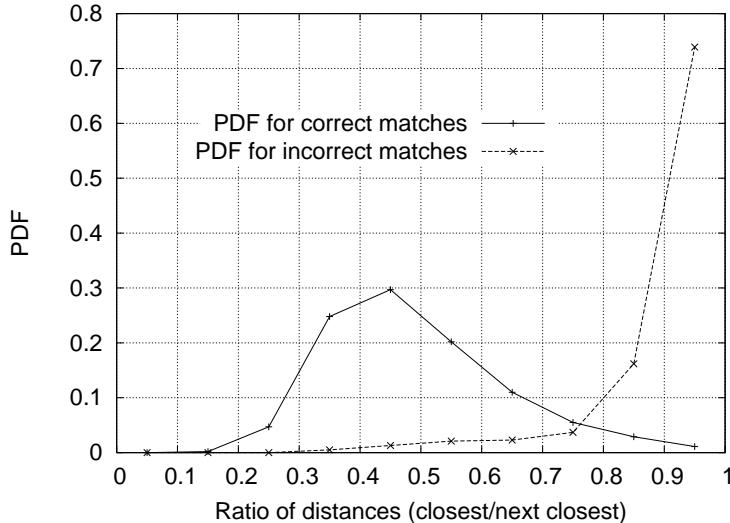


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

second-closest neighbor. If there are multiple training images of the same object, then we define the second-closest neighbor as being the closest neighbor that is known to come from a different object than the first, such as by only using images known to contain different objects. This measure performs well because correct matches need to have the closest neighbor significantly closer than the closest incorrect match to achieve reliable matching. For false matches, there will likely be a number of other false matches within similar distances due to the high dimensionality of the feature space. We can think of the second-closest match as providing an estimate of the density of false matches within this portion of the feature space and at the same time identifying specific instances of feature ambiguity.

Figure 11 shows the value of this measure for real image data. The probability density functions for correct and incorrect matches are shown in terms of the ratio of closest to second-closest neighbors of each keypoint. Matches for which the nearest neighbor was a correct match have a PDF that is centered at a much lower ratio than that for incorrect matches. For our object recognition implementation, we reject all matches in which the distance ratio is greater than 0.8, which eliminates 90% of the false matches while discarding less than 5% of the correct matches. This figure was generated by matching images following random scale and orientation change, a depth rotation of 30 degrees, and addition of 2% image noise, against a database of 40,000 keypoints.

7.2 Efficient nearest neighbor indexing

No algorithms are known that can identify the exact nearest neighbors of points in high dimensional spaces that are any more efficient than exhaustive search. Our keypoint descriptor has a 128-dimensional feature vector, and the best algorithms, such as the k-d tree (Friedman *et al.*, 1977) provide no speedup over exhaustive search for more than about 10 dimensional spaces. Therefore, we have used an approximate algorithm, called the Best-Bin-First (BBF) algorithm (Beis and Lowe, 1997). This is approximate in the sense that it returns the closest

neighbor with high probability.

The BBF algorithm uses a modified search ordering for the k-d tree algorithm so that bins in feature space are searched in the order of their closest distance from the query location. This priority search order was first examined by Arya and Mount (1993), and they provide further study of its computational properties in (Arya *et al.*, 1998). This search order requires the use of a heap-based priority queue for efficient determination of the search order. An approximate answer can be returned with low cost by cutting off further search after a specific number of the nearest bins have been explored. In our implementation, we cut off search after checking the first 200 nearest-neighbor candidates. For a database of 100,000 keypoints, this provides a speedup over exact nearest neighbor search by about 2 orders of magnitude yet results in less than a 5% loss in the number of correct matches. One reason the BBF algorithm works particularly well for this problem is that we only consider matches in which the nearest neighbor is less than 0.8 times the distance to the second-nearest neighbor (as described in the previous section), and therefore there is no need to exactly solve the most difficult cases in which many neighbors are at very similar distances.

7.3 Clustering with the Hough transform

To maximize the performance of object recognition for small or highly occluded objects, we wish to identify objects with the fewest possible number of feature matches. We have found that reliable recognition is possible with as few as 3 features. A typical image contains 2,000 or more features which may come from many different objects as well as background clutter. While the distance ratio test described in Section 7.1 will allow us to discard many of the false matches arising from background clutter, this does not remove matches from other valid objects, and we often still need to identify correct subsets of matches containing less than 1% inliers among 99% outliers. Many well-known robust fitting methods, such as RANSAC or Least Median of Squares, perform poorly when the percent of inliers falls much below 50%. Fortunately, much better performance can be obtained by clustering features in pose space using the Hough transform (Hough, 1962; Ballard, 1981; Grimson 1990).

The Hough transform identifies clusters of features with a consistent interpretation by using each feature to vote for all object poses that are consistent with the feature. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature. Each of our keypoints specifies 4 parameters: 2D location, scale, and orientation, and each matched keypoint in the database has a record of the keypoint's parameters relative to the training image in which it was found. Therefore, we can create a Hough transform entry predicting the model location, orientation, and scale from the match hypothesis. This prediction has large error bounds, as the similarity transform implied by these 4 parameters is only an approximation to the full 6 degree-of-freedom pose space for a 3D object and also does not account for any non-rigid deformations. Therefore, we use broad bin sizes of 30 degrees for orientation, a factor of 2 for scale, and 0.25 times the maximum projected training image dimension (using the predicted scale) for location. To avoid the problem of boundary effects in bin assignment, each keypoint match votes for the 2 closest bins in each dimension, giving a total of 16 entries for each hypothesis and further broadening the pose range.

In most implementations of the Hough transform, a multi-dimensional array is used to represent the bins. However, many of the potential bins will remain empty, and it is difficult to compute the range of possible bin values due to their mutual dependence (for example,

the dependency of location discretization on the selected scale). These problems can be avoided by using a pseudo-random hash function of the bin values to insert votes into a one-dimensional hash table, in which collisions are easily detected.

7.4 Solution for affine parameters

The Hough transform is used to identify all clusters with at least 3 entries in a bin. Each such cluster is then subject to a geometric verification procedure in which a least-squares solution is performed for the best affine projection parameters relating the training image to the new image.

An affine transformation correctly accounts for 3D rotation of a planar surface under orthographic projection, but the approximation can be poor for 3D rotation of non-planar objects. A more general solution would be to solve for the fundamental matrix (Luong and Faugeras, 1996; Hartley and Zisserman, 2000). However, a fundamental matrix solution requires at least 7 point matches as compared to only 3 for the affine solution and in practice requires even more matches for good stability. We would like to perform recognition with as few as 3 feature matches, so the affine solution provides a better starting point and we can account for errors in the affine approximation by allowing for large residual errors. If we imagine placing a sphere around an object, then rotation of the sphere by 30 degrees will move no point within the sphere by more than 0.25 times the projected diameter of the sphere. For the examples of typical 3D objects used in this paper, an affine solution works well given that we allow residual errors up to 0.25 times the maximum projected dimension of the object. A more general approach is given in (Brown and Lowe, 2002), in which the initial solution is based on a similarity transform, which then progresses to solution for the fundamental matrix in those cases in which a sufficient number of matches are found.

The affine transformation of a model point $[x \ y]^T$ to an image point $[u \ v]^T$ can be written as

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

where the model translation is $[t_x \ t_y]^T$ and the affine rotation, scale, and stretch are represented by the m_i parameters.

We wish to solve for the transformation parameters, so the equation above can be rewritten to gather the unknowns into a column vector:

$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \\ \dots & & & & & \\ \dots & & & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u \\ v \\ \vdots \end{bmatrix}$$

This equation shows a single match, but any number of further matches can be added, with each match contributing two more rows to the first and last matrix. At least 3 matches are needed to provide a solution.

We can write this linear system as

$$\mathbf{Ax} = \mathbf{b}$$



Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affine transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

The least-squares solution for the parameters \mathbf{x} can be determined by solving the corresponding normal equations,

$$\mathbf{x} = [\mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{b},$$

which minimizes the sum of the squares of the distances from the projected model locations to the corresponding image locations. This least-squares approach could readily be extended to solving for 3D pose and internal parameters of articulated and flexible objects (Lowe, 1991).

Outliers can now be removed by checking for agreement between each image feature and the model. Given the more accurate least-squares solution, we now require each match to agree within half the error range that was used for the parameters in the Hough transform bins. If fewer than 3 points remain after discarding outliers, then the match is rejected. As outliers are discarded, the least-squares solution is re-solved with the remaining points, and the process iterated. In addition, a top-down matching phase is used to add any further matches that agree with the projected model position. These may have been missed from the Hough transform bin due to the similarity transform approximation or other errors.

The final decision to accept or reject a model hypothesis is based on a detailed probabilistic model given in a previous paper (Lowe, 2001). This method first computes the expected number of false matches to the model pose, given the projected size of the model, the number of features within the region, and the accuracy of the fit. A Bayesian analysis then gives the probability that the object is present based on the actual number of matching features found. We accept a model if the final probability for a correct interpretation is greater than 0.98. For objects that project to small regions of an image, 3 features may be sufficient for reliable recognition. For large objects covering most of a heavily textured image, the expected number of false matches is higher, and as many as 10 feature matches may be necessary.

8 Recognition examples

Figure 12 shows an example of object recognition for a cluttered and occluded image containing 3D objects. The training images of a toy train and a frog are shown on the left.

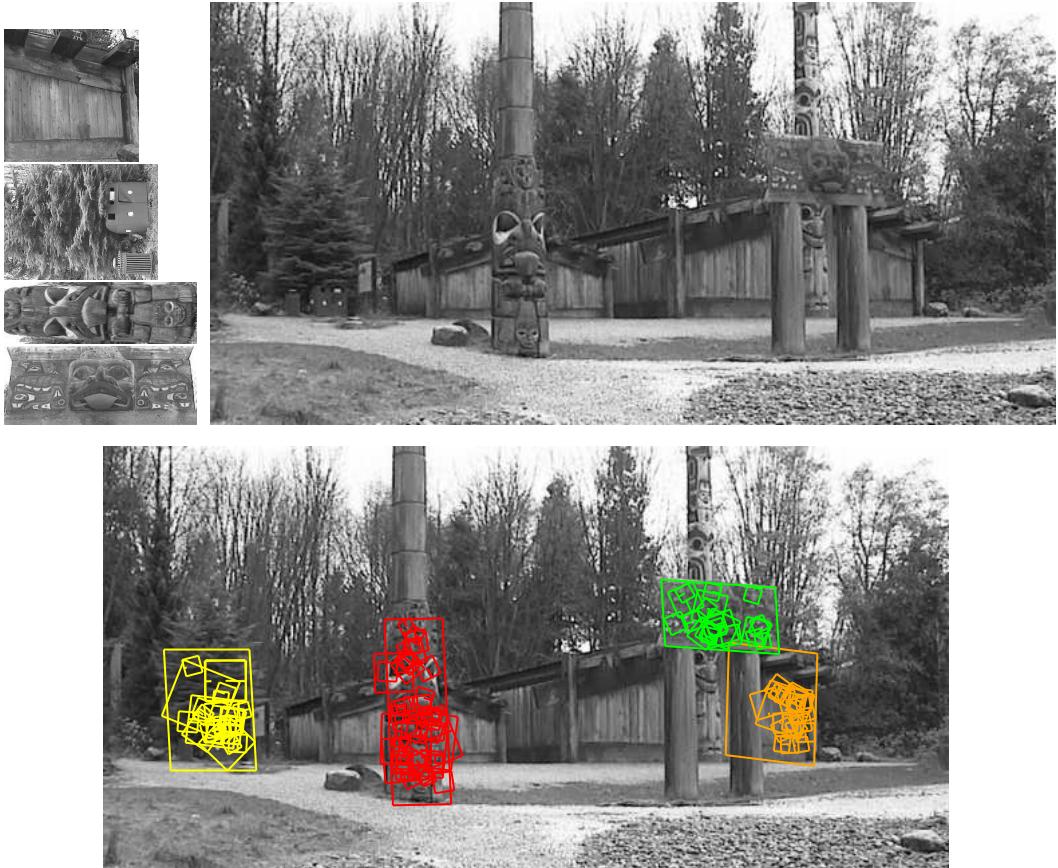


Figure 13: This example shows location recognition within a complex scene. The training images are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affine transform used for recognition.

The middle image (of size 600x480 pixels) contains instances of these objects hidden behind others and with extensive background clutter so that detection of the objects may not be immediate even for human vision. The image on the right shows the final correct identification superimposed on a reduced contrast version of the image. The keypoints that were used for recognition are shown as squares with an extra line to indicate orientation. The sizes of the squares correspond to the image regions used to construct the descriptor. An outer parallelogram is also drawn around each instance of recognition, with its sides corresponding to the boundaries of the training images projected under the final affine transformation determined during recognition.

Another potential application of the approach is to place recognition, in which a mobile device or vehicle could identify its location by recognizing familiar locations. Figure 13 gives an example of this application, in which training images are taken of a number of locations. As shown on the upper left, these can even be of such seemingly non-distinctive items as a wooden wall or a tree with trash bins. The test image (of size 640 by 315 pixels) on the upper right was taken from a viewpoint rotated about 30 degrees around the scene from the original positions, yet the training image locations are easily recognized.

All steps of the recognition process can be implemented efficiently, so the total time to recognize all objects in Figures 12 or 13 is less than 0.3 seconds on a 2GHz Pentium 4 processor. We have implemented these algorithms on a laptop computer with attached video camera, and have tested them extensively over a wide range of conditions. In general, textured planar surfaces can be identified reliably over a rotation in depth of up to 50 degrees in any direction and under almost any illumination conditions that provide sufficient light and do not produce excessive glare. For 3D objects, the range of rotation in depth for reliable recognition is only about 30 degrees in any direction and illumination change is more disruptive. For these reasons, 3D object recognition is best performed by integrating features from multiple views, such as with local feature view clustering (Lowe, 2001).

These keypoints have also been applied to the problem of robot localization and mapping, which has been presented in detail in other papers (Se, Lowe and Little, 2001). In this application, a trinocular stereo system is used to determine 3D estimates for keypoint locations. Keypoints are used only when they appear in all 3 images with consistent disparities, resulting in very few outliers. As the robot moves, it localizes itself using feature matches to the existing 3D map, and then incrementally adds features to the map while updating their 3D positions using a Kalman filter. This provides a robust and accurate solution to the problem of robot localization in unknown environments. This work has also addressed the problem of place recognition, in which a robot can be switched on and recognize its location anywhere within a large map (Se, Lowe and Little, 2002), which is equivalent to a 3D implementation of object recognition.

9 Conclusions

The SIFT keypoints described in this paper are particularly useful due to their distinctiveness, which enables the correct match for a keypoint to be selected from a large database of other keypoints. This distinctiveness is achieved by assembling a high-dimensional vector representing the image gradients within a local region of the image. The keypoints have been shown to be invariant to image rotation and scale and robust across a substantial range of affine distortion, addition of noise, and change in illumination. Large numbers of keypoints can be extracted from typical images, which leads to robustness in extracting small objects among clutter. The fact that keypoints are detected over a complete range of scales means that small local features are available for matching small and highly occluded objects, while large keypoints perform well for images subject to noise and blur. Their computation is efficient, so that several thousand keypoints can be extracted from a typical image with near real-time performance on standard PC hardware.

This paper has also presented methods for using the keypoints for object recognition. The approach we have described uses approximate nearest-neighbor lookup, a Hough transform for identifying clusters that agree on object pose, least-squares pose determination, and final verification. Other potential applications include view matching for 3D reconstruction, motion tracking and segmentation, robot localization, image panorama assembly, epipolar calibration, and any others that require identification of matching locations between images.

There are many directions for further research in deriving invariant and distinctive image features. Systematic testing is needed on data sets with full 3D viewpoint and illumination changes. The features described in this paper use only a monochrome intensity image, so further distinctiveness could be derived from including illumination-invariant color descriptors

(Funt and Finlayson, 1995; Brown and Lowe, 2002). Similarly, local texture measures appear to play an important role in human vision and could be incorporated into feature descriptors in a more general form than the single spatial frequency used by the current descriptors. An attractive aspect of the invariant local feature approach to matching is that there is no need to select just one feature type, and the best results are likely to be obtained by using many different features, all of which can contribute useful matches and improve overall robustness.

Another direction for future research will be to individually learn features that are suited to recognizing particular objects categories. This will be particularly important for generic object classes that must cover a broad range of possible appearances. The research of Weber, Welling, and Perona (2000) and Fergus, Perona, and Zisserman (2003) has shown the potential of this approach by learning small sets of local features that are suited to recognizing generic classes of objects. In the long term, feature sets are likely to contain both prior and learned features that will be used according to the amount of training data that has been available for various object classes.

Acknowledgments

I would particularly like to thank Matthew Brown, who has suggested numerous improvements to both the content and presentation of this paper and whose own work on feature localization and invariance has contributed to this approach. In addition, I would like to thank many others for their valuable suggestions, including Stephen Se, Jim Little, Krystian Mikolajczyk, Cordelia Schmid, Tony Lindeberg, and Andrew Zisserman. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and through the Institute for Robotics and Intelligent Systems (IRIS) Network of Centres of Excellence.

References

- Arya, S., and Mount, D.M. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pp. 271-280.
- Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., and Wu, A.Y. 1998. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891-923.
- Ballard, D.H. 1981. Generalizing the Hough transform to detect arbitrary patterns. *Pattern Recognition*, 13(2):111-122.
- Basri, R., and Jacobs, D.W. 1997. Recognition using region correspondences. *International Journal of Computer Vision*, 25(2):145-166.
- Baumberg, A. 2000. Reliable feature matching across widely separated views. In *Conference on Computer Vision and Pattern Recognition*, Hilton Head, South Carolina, pp. 774-781.
- Beis, J. and Lowe, D.G. 1997. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Conference on Computer Vision and Pattern Recognition*, Puerto Rico, pp. 1000-1006.
- Brown, M. and Lowe, D.G. 2002. Invariant features from interest point groups. In *British Machine Vision Conference*, Cardiff, Wales, pp. 656-665.
- Carneiro, G., and Jepson, A.D. 2002. Phase-based local features. In *European Conference on Computer Vision (ECCV)*, Copenhagen, Denmark, pp. 282-296.
- Crowley, J. L. and Parker, A.C. 1984. A representation for shape based on peaks and ridges in the difference of low-pass transform. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6(2):156-170.

- Edelman, S., Intrator, N. and Poggio, T. 1997. Complex cells and object recognition. Unpublished manuscript: <http://kybele.psych.cornell.edu/~edelman/archive.html>
- Fergus, R., Perona, P., and Zisserman, A. 2003. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, Madison, Wisconsin, pp. 264-271.
- Friedman, J.H., Bentley, J.L. and Finkel, R.A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209-226.
- Funt, B.V. and Finlayson, G.D. 1995. Color constant color indexing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(5):522-529.
- Grimson, E. 1990. *Object Recognition by Computer: The Role of Geometric Constraints*, The MIT Press: Cambridge, MA.
- Harris, C. 1992. Geometry from visual motion. In *Active Vision*, A. Blake and A. Yuille (Eds.), MIT Press, pp. 263-284.
- Harris, C. and Stephens, M. 1988. A combined corner and edge detector. In *Fourth Alvey Vision Conference*, Manchester, UK, pp. 147-151.
- Hartley, R. and Zisserman, A. 2000. *Multiple view geometry in computer vision*, Cambridge University Press: Cambridge, UK.
- Hough, P.V.C. 1962. *Method and means for recognizing complex patterns*. U.S. Patent 3069654.
- Koenderink, J.J. 1984. The structure of images. *Biological Cybernetics*, 50:363-396.
- Lindeberg, T. 1993. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention. *International Journal of Computer Vision*, 11(3): 283-318.
- Lindeberg, T. 1994. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21(2):224-270.
- Lowe, D.G. 1991. Fitting parameterized three-dimensional models to images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(5):441-450.
- Lowe, D.G. 1999. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, Corfu, Greece, pp. 1150-1157.
- Lowe, D.G. 2001. Local feature view clustering for 3D object recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, Kauai, Hawaii, pp. 682-688.
- Luong, Q.T., and Faugeras, O.D. 1996. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision*, 17(1):43-76.
- Matas, J., Chum, O., Urban, M., and Pajdla, T. 2002. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, Cardiff, Wales, pp. 384-393.
- Mikolajczyk, K. 2002. *Detection of local features invariant to affine transformations*, Ph.D. thesis, Institut National Polytechnique de Grenoble, France.
- Mikolajczyk, K., and Schmid, C. 2002. An affine invariant interest point detector. In *European Conference on Computer Vision (ECCV)*, Copenhagen, Denmark, pp. 128-142.
- Mikolajczyk, K., Zisserman, A., and Schmid, C. 2003. Shape recognition with edge-based features. In *Proceedings of the British Machine Vision Conference*, Norwich, U.K.
- Moravec, H. 1981. Rover visual obstacle avoidance. In *International Joint Conference on Artificial Intelligence*, Vancouver, Canada, pp. 785-790.
- Nelson, R.C., and Selinger, A. 1998. Large-scale tests of a keyed, appearance-based 3-D object recognition system. *Vision Research*, 38(15):2469-88.
- Pope, A.R., and Lowe, D.G. 2000. Probabilistic models of appearance for 3-D object recognition. *International Journal of Computer Vision*, 40(2):149-167.

- Pritchard, D., and Heidrich, W. 2003. Cloth motion capture. *Computer Graphics Forum (Eurographics 2003)*, 22(3):263-271.
- Schaffalitzky, F., and Zisserman, A. 2002. Multi-view matching for unordered image sets, or ‘How do I organize my holiday snaps?’’’ In *European Conference on Computer Vision*, Copenhagen, Denmark, pp. 414-431.
- Schiele, B., and Crowley, J.L. 2000. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*, 36(1):31-50.
- Schmid, C., and Mohr, R. 1997. Local grayvalue invariants for image retrieval. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(5):530-534.
- Se, S., Lowe, D.G., and Little, J. 2001. Vision-based mobile robot localization and mapping using scale-invariant features. In *International Conference on Robotics and Automation*, Seoul, Korea, pp. 2051-58.
- Se, S., Lowe, D.G., and Little, J. 2002. Global localization using distinctive visual features. In *International Conference on Intelligent Robots and Systems, IROS 2002*, Lausanne, Switzerland, pp. 226-231.
- Shokoufandeh, A., Marsic, I., and Dickinson, S.J. 1999. View-based object recognition using saliency maps. *Image and Vision Computing*, 17:445-460.
- Torr, P. 1995. *Motion Segmentation and Outlier Detection*, Ph.D. Thesis, Dept. of Engineering Science, University of Oxford, UK.
- Tuytelaars, T., and Van Gool, L. 2000. Wide baseline stereo based on local, affinely invariant regions. In *British Machine Vision Conference*, Bristol, UK, pp. 412-422.
- Weber, M., Welling, M. and Perona, P. 2000. Unsupervised learning of models for recognition. In *European Conference on Computer Vision*, Dublin, Ireland, pp. 18-32.
- Witkin, A.P. 1983. Scale-space filtering. In *International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, pp. 1019-1022.
- Zhang, Z., Deriche, R., Faugeras, O., and Luong, Q.T. 1995. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78:87-119.

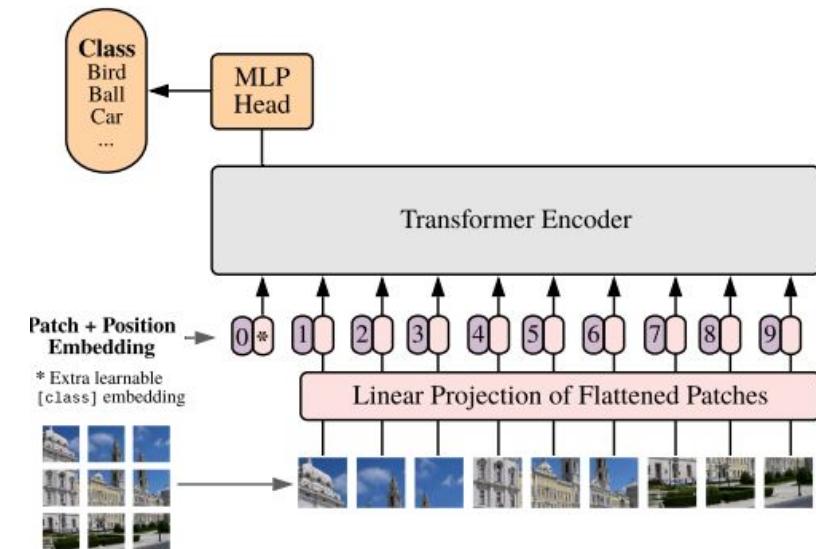


Computer Vision

2023-24 First Semester, M.Tech (AIML)

Session #8-9:

Attention, Transformers & Vision Transformers (ViT)

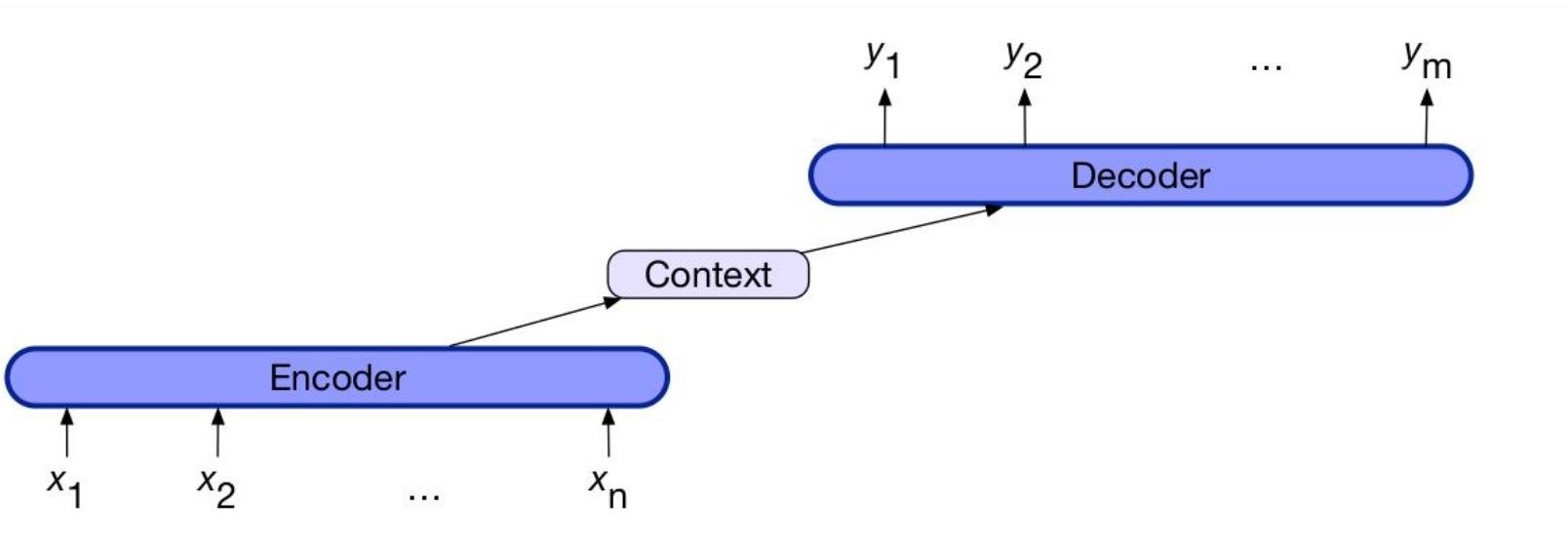


Topics

- Attention (Review)
- Transformers (Review)
- Vision Transformer (ViT)
- Applications of ViT in Computer Vision
- Popular ViT Variants

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

(Review) Encoder - Decoder Arch for Seq. to Seq. Translation

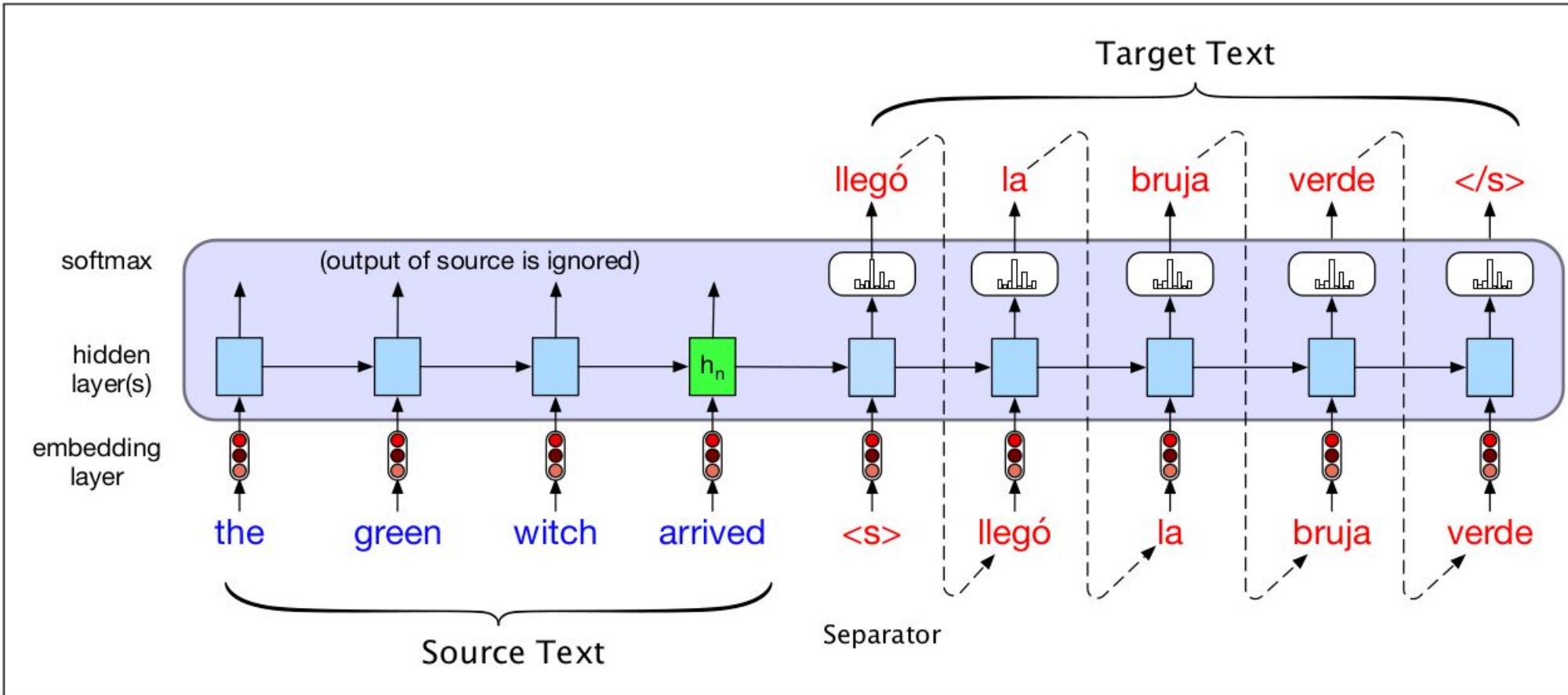


Components:

1. Encoder
2. Context Vector, c
3. Decoder

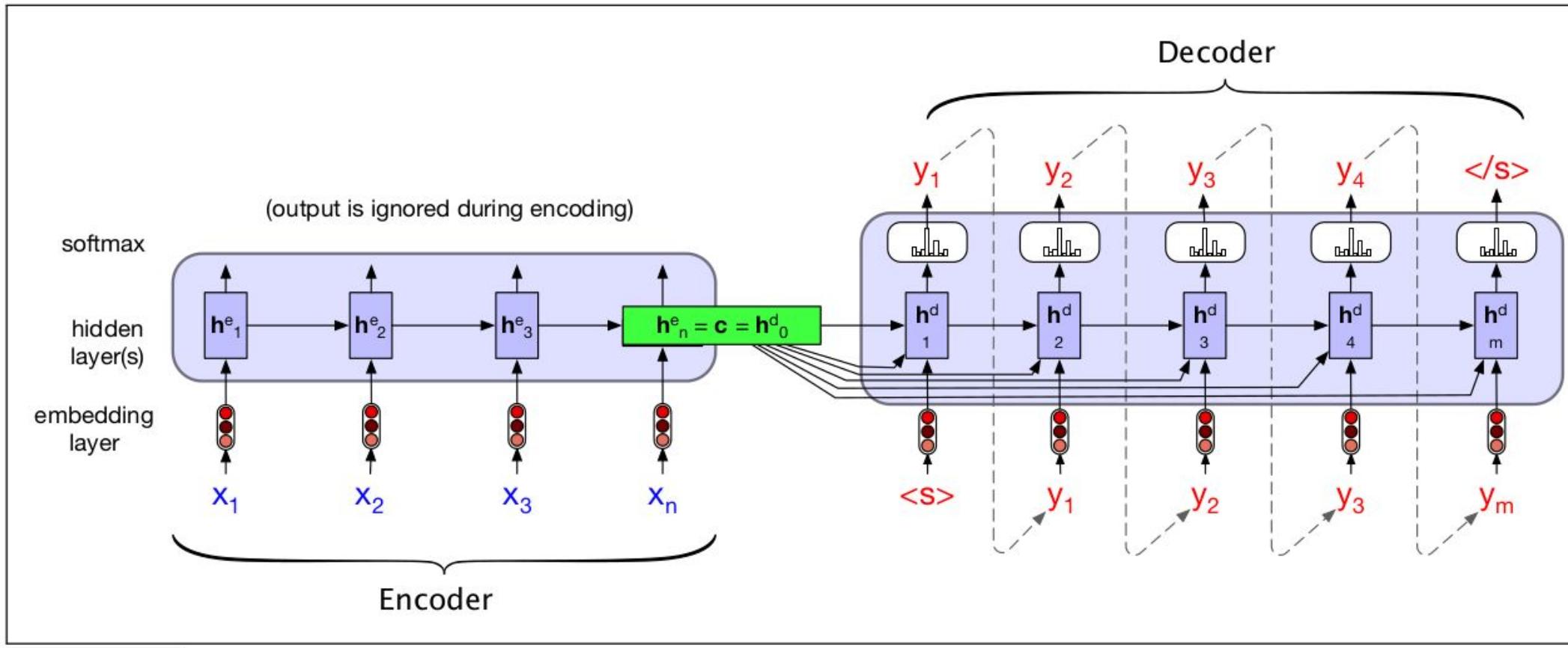
Note: The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways

(Review) Encoder - Decoder Arch for Seq. to Seq. Translation



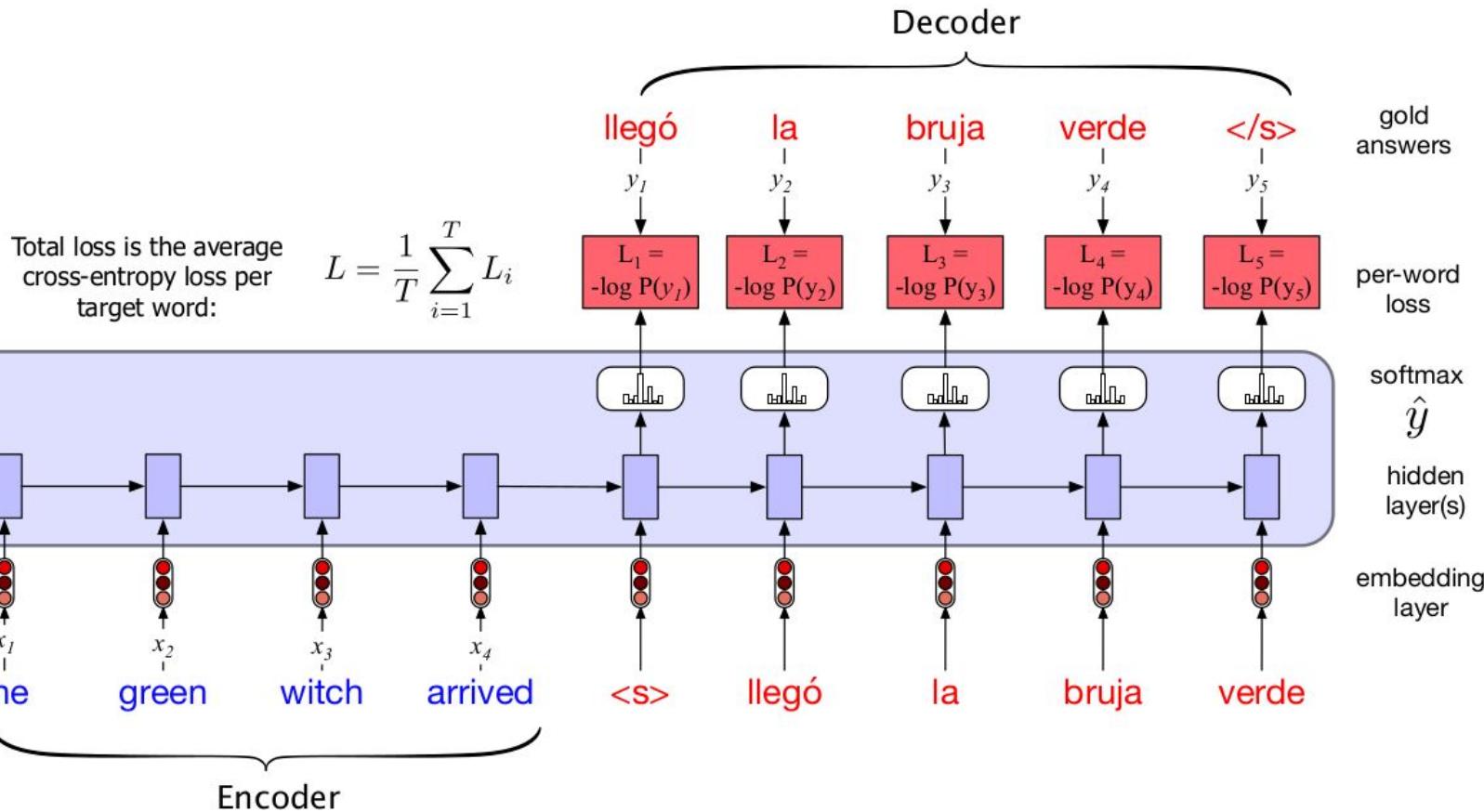
We make the context vector c available to more than just the first decoder hidden state, to ensure that the influence of the context vector, c , doesn't wane as the output sequence is generated.

(Review) Encoder - Decoder Arch for Seq. to Seq. Translation



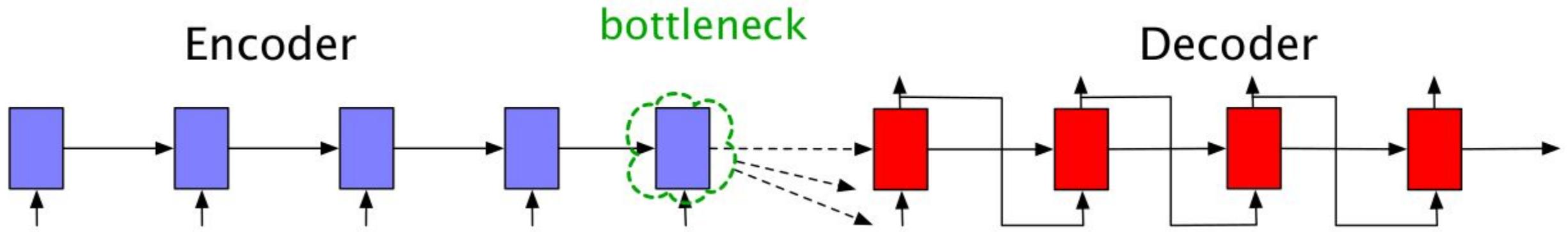
Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

(Review) Encoder - Decoder Arch for Seq. to Seq. Translation



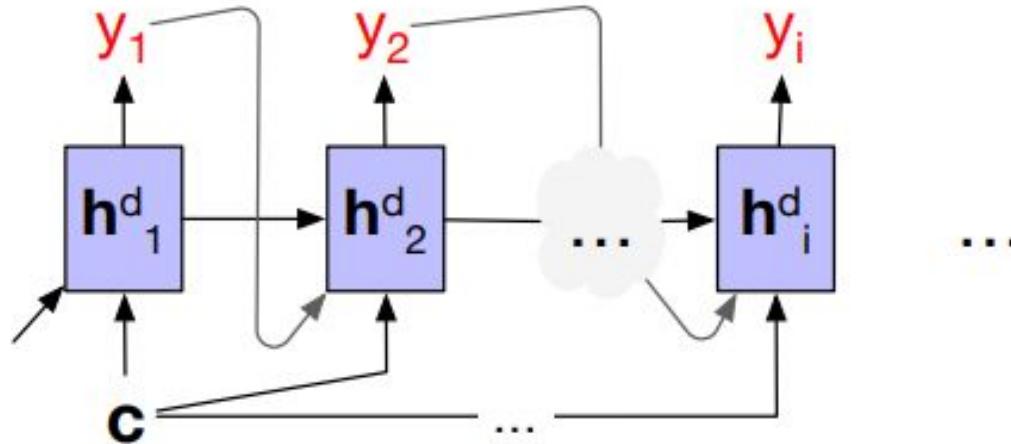
In the decoder we usually don't propagate the model's softmax outputs \hat{y}_t , but use **teacher forcing**. We compute the softmax output distribution over \hat{y} in the decoder, which can then be averaged to compute a loss for the sentence.

(Review) Encoder - Decoder Arch for Seq. to Seq. Translation



Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

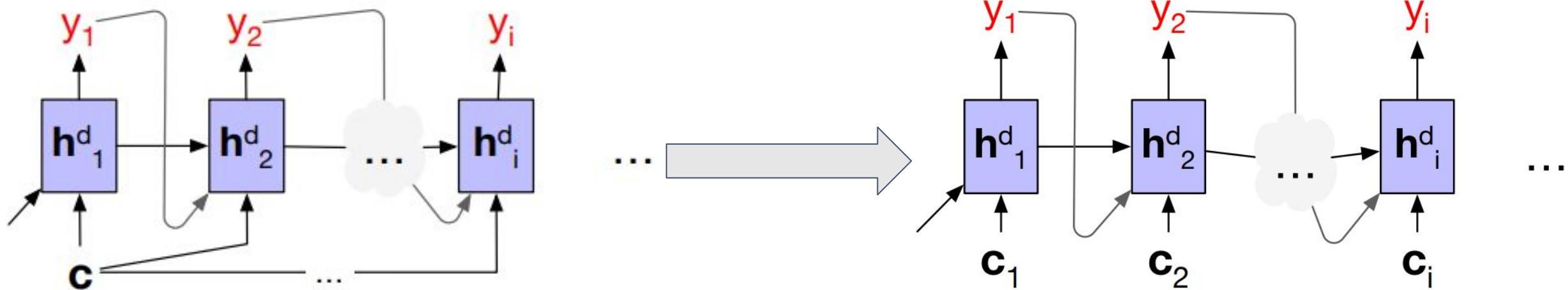
(Review) Attention !



Without attention, a decoder sees the same context vector ,
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

(Review) Attention !



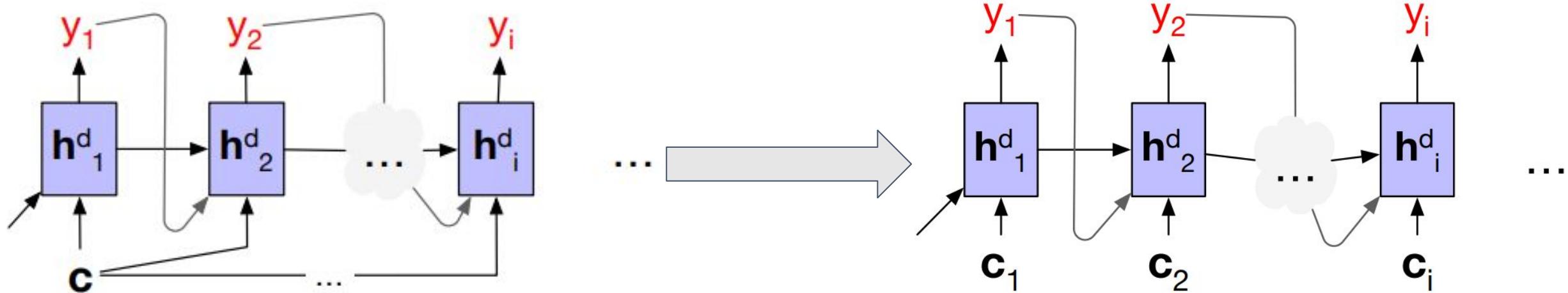
Without attention, a decoder sees the same context vector ,
which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder to sees a different, dynamic,
context, which is a function of all the encoder hidden states

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

(Review) Attention !



Without attention, a decoder sees the same context vector , which is a static function of all the encoder hidden states

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

With attention, decoder gets information from all the hidden states of the encoder, not just the last hidden state of the encoder Each context vector is obtained by taking a weighted sum of all the encoder hidden states.

The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing

With attention, decoder sees a different, dynamic, context, which is a function of all the encoder hidden states

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$

(Review) Attention !

Step -1 : Find out how relevant each encoder state is to the present decoder state \mathbf{h}_{i-1}^d

Compute a score of similarity between \mathbf{h}_{i-1}^d and all the encoder states : $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e)$

Dot Product Attention : $score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$

Step -2 : Normalize all the scores with softmax to create a vector of weights, $\alpha_{i,j}$

$\alpha_{i,j}$ indicates the proportional relevance of each encoder hidden state j to the prior hidden decoder state, \mathbf{h}_{i-1}^d

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) \quad \forall j \in e) \\ &= \frac{\exp(score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))}{\sum_k \exp(score(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))}\end{aligned}$$



(Review) Attention !

Step -3 : Given the distribution in α , compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

(Review) Attention !

Step -3 : Given the distribution in α , compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

Plus : In step-1, we can get a more powerful scoring function by parameterizing the score with its own set of weights, \mathbf{W}_s :

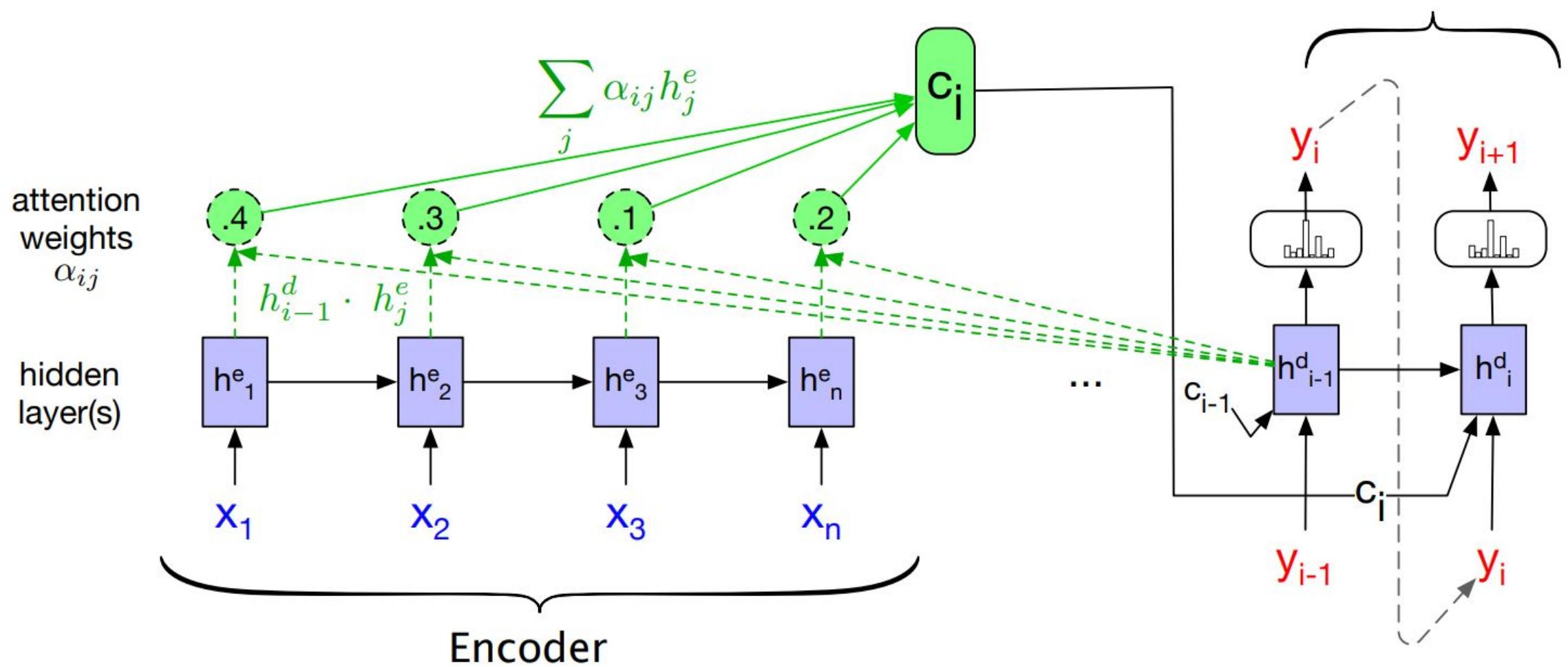
$$\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{t-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

\mathbf{W}_s , is trained during normal end-to-end training,

\mathbf{W}_s , gives the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

(Review) Attention !

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$





(Review) Transformers

- 2017, NIPS, Vaswani et. al., **Attention Is All You Need !!!**
- Transformers **map sequences of input vectors (x_1, \dots, x_n) to sequences of output vectors (y_1, \dots, y_n) of the same length**
- Made up of **transformer blocks** in which the key component is **self-attention layers**

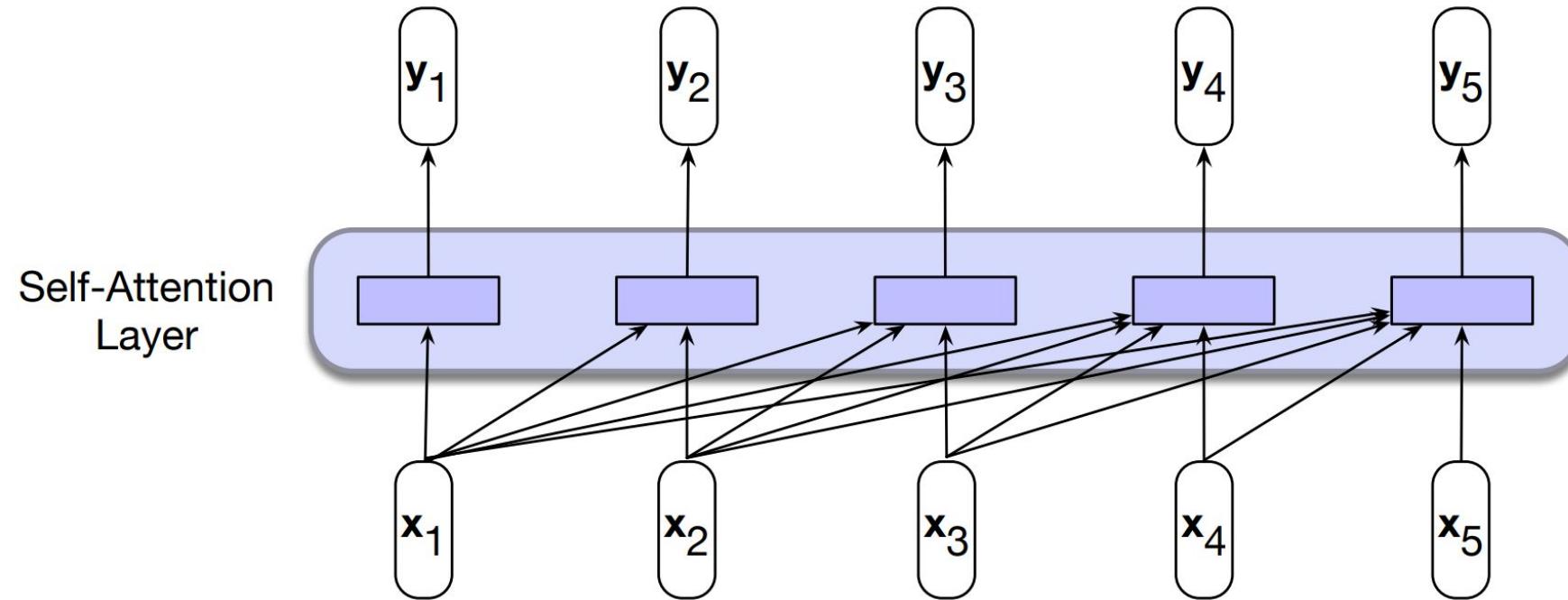
[Self-attention allows a network to directly extract and use information from arbitrarily large contexts directly !!!]
- Transformers are **not based on recurrent connections** \Rightarrow Parallel implementations possible \Rightarrow Efficient to scale (comparing LSTM)



(Review) Self-Attention | Transformers

- Attention \Rightarrow Ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context.
- Self-attention \Rightarrow
 - > Set of *comparisons* are to other elements *within a given sequence*
 - > Use these comparisons to compute an output for the current input

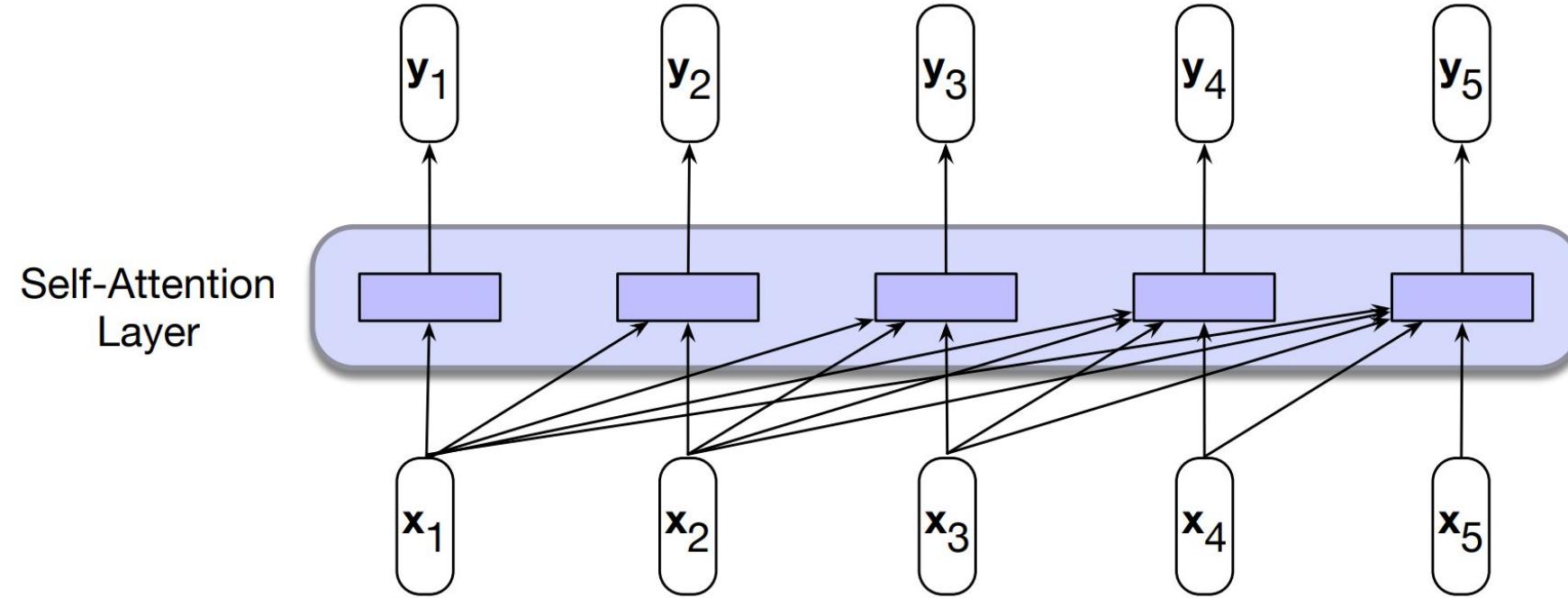
(Review) Self-Attention | Transformers



In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one.

Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

(Review) Self-Attention | Transformers



$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i \end{aligned}$$



(Review) Self-Attention | Transformers

- Let us understand how transformers uses self-attention !



(Review) Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

Query, Q

As the current focus of attention when being compared to all of the other preceding inputs.

Key, K

In its role as a preceding input being compared to the current focus of attention.

Value, V

As a value used to compute the output for the current focus of attention

Three different roles each \mathbf{x}_i (input embedding), in the computation of self attention

(Review) Self-Attention | Transformers

$$\mathbf{W}^V, \mathbf{W}^K, \mathbf{W}^Q \in \mathbb{R}^{d \times d}$$

In Vaswani et al., 2017, d was 1024.

- Let us understand how transformers uses self-attention !

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \quad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

The simple dot product can be an arbitrarily large;
scaled dot-product is used in transformers;

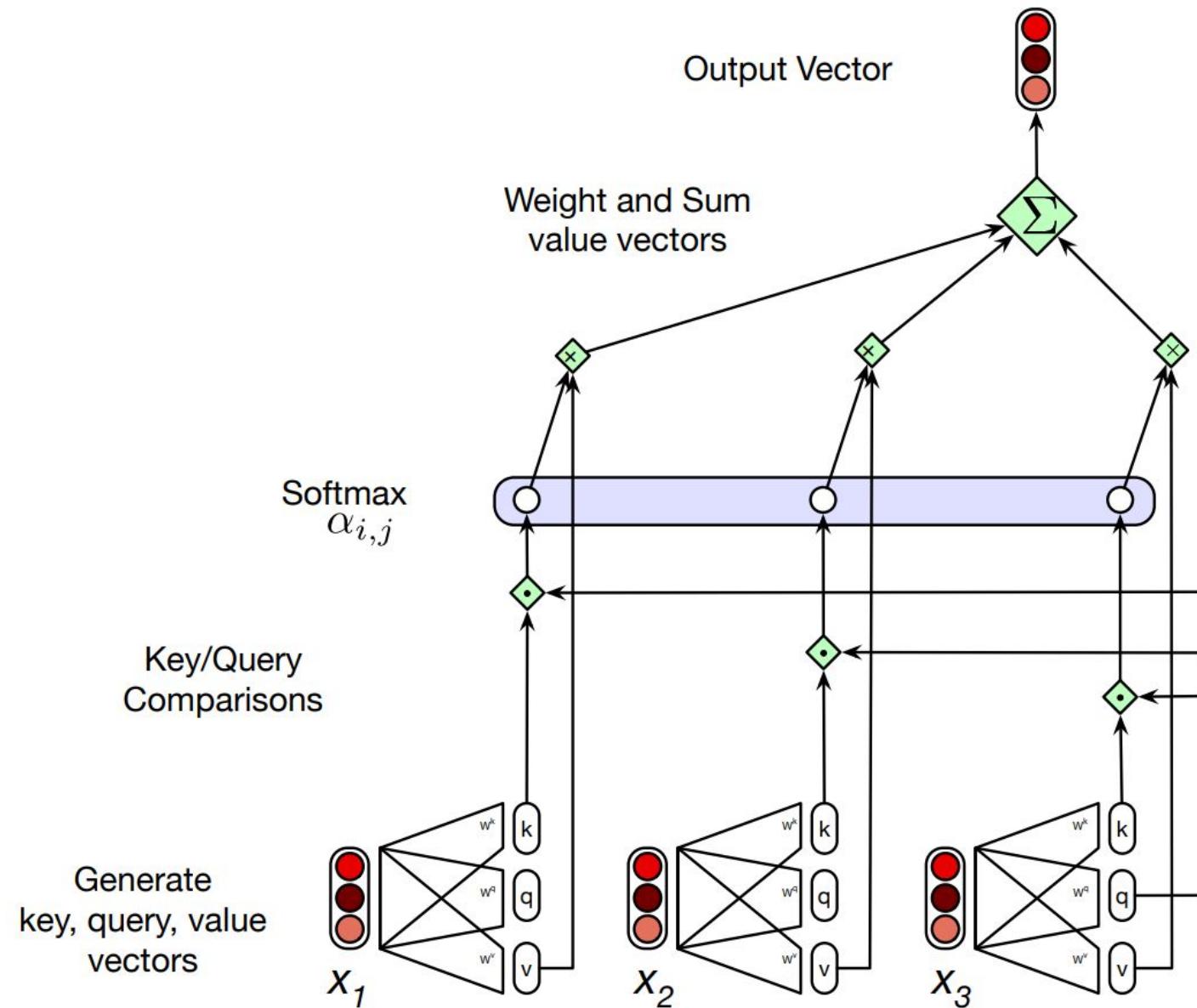
$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i\end{aligned}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

(Review) Self-Attention | Transformers

- Each output, y_i , is computed independently
- Entire process can be parallelized

Calculating the value of y_3 , the third element of a sequence using causal (left-to-right) self-attention





(Review) Self-Attention | Transformers

- Pack the input embeddings of the N input tokens into a single matrix

$$\mathbf{X} \in \mathbb{R}^{N \times d}$$

> Each row of \mathbf{X} is the embedding of one token of the input

- Multiply \mathbf{X} by the key, query, and value ($d \times d$) matrices

$$\mathbf{Q} = \mathbf{XW}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{XW}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{XW}^{\mathbf{V}}$$



(Review) Self-Attention | Transformers

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}$$

QK^T Matrix N

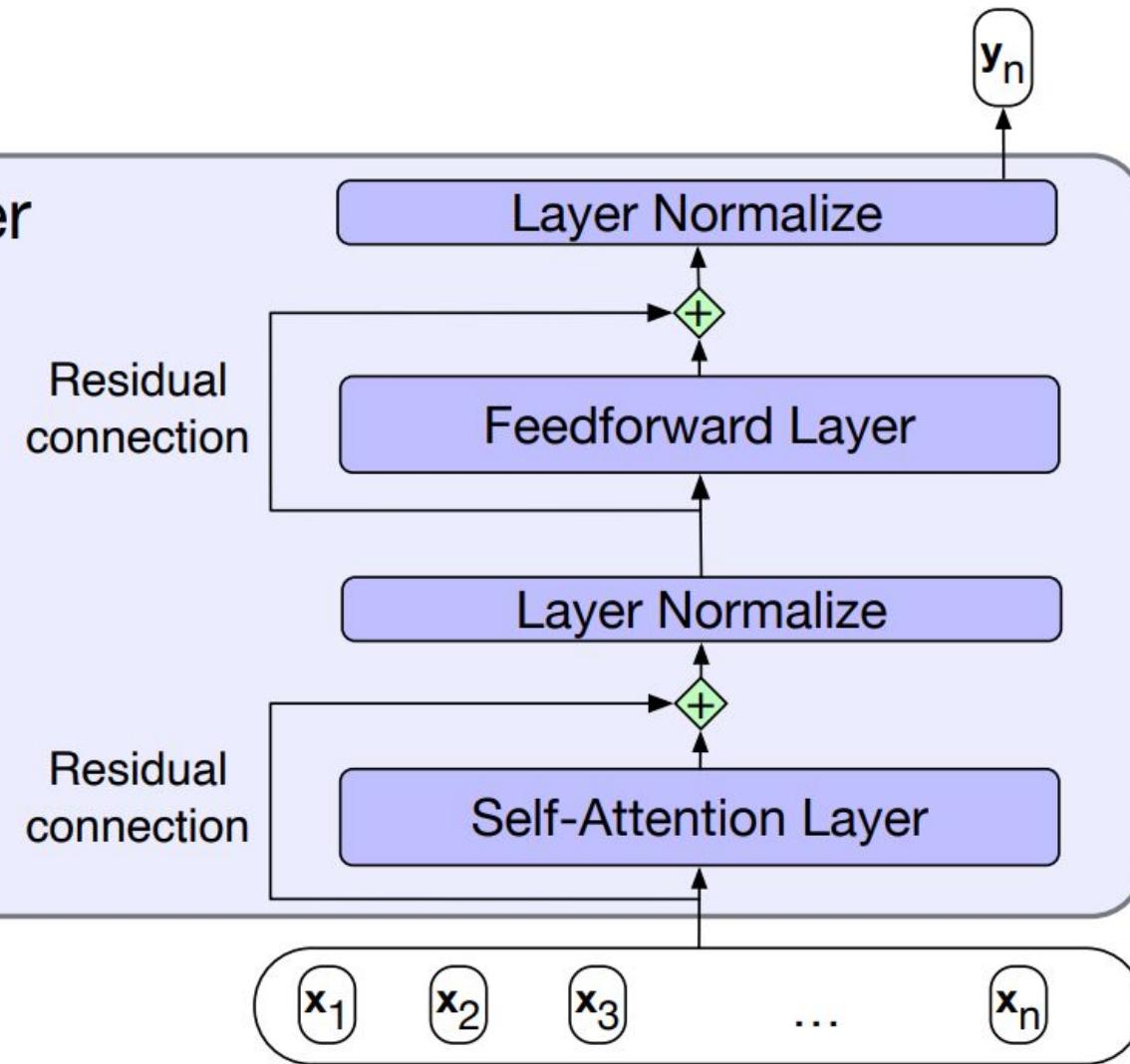
q1•k1	-∞	-∞	-∞	-∞
q2•k1	q2•k2	-∞	-∞	-∞
q3•k1	q3•k2	q3•k3	-∞	-∞
q4•k1	q4•k2	q4•k3	q4•k4	-∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

Note: Upper-triangle portion of the comparisons matrix zeroed out (set to -∞, which the softmax will turn to zero)

(Review) Transformer Blocks | Transformers

Transformer Block

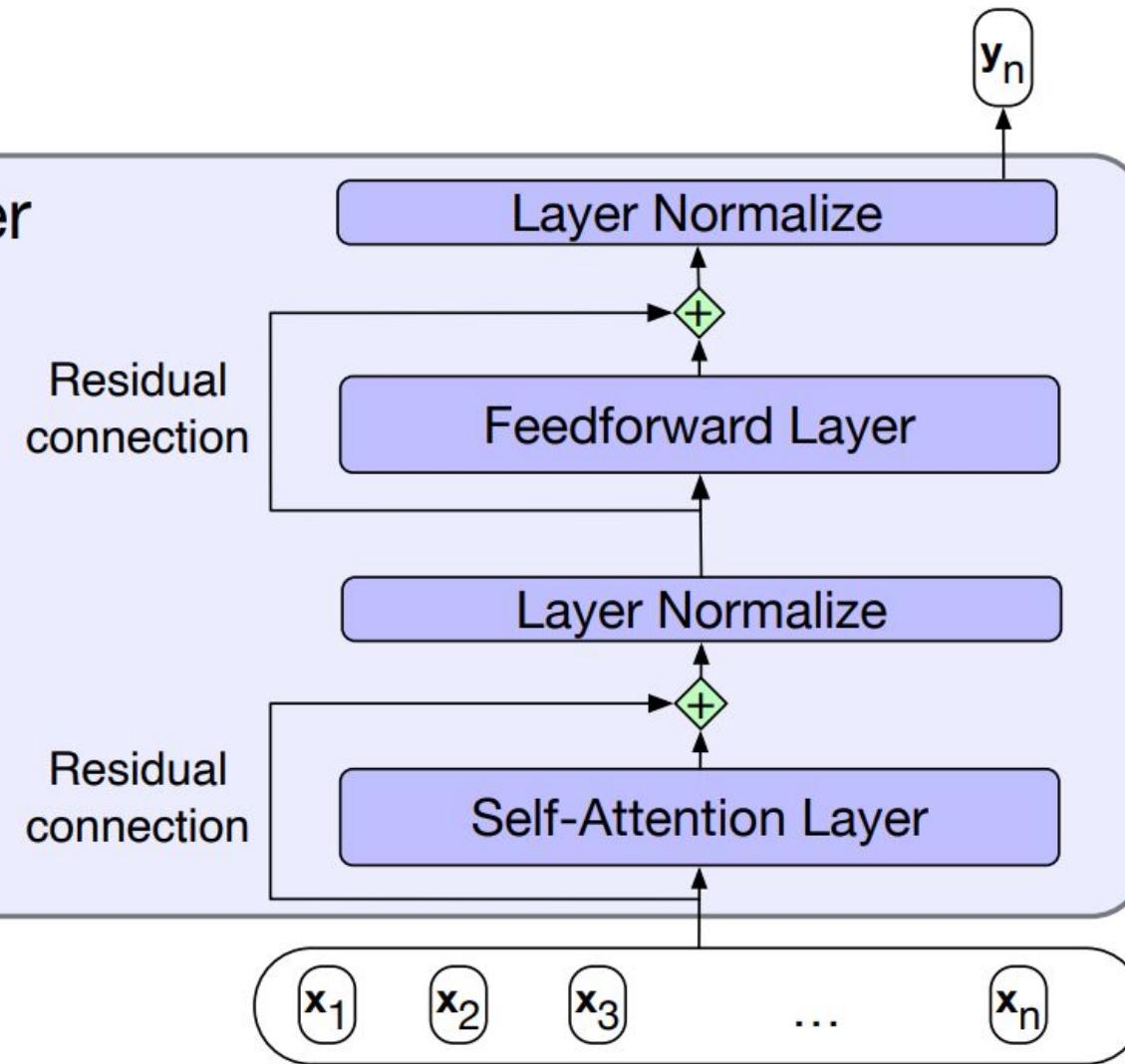


$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x}))$$

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z}))$$

(Review) Transformer Blocks | Transformers

Transformer Block



$$\begin{aligned} \mathbf{z} &= \text{LayerNorm}(\mathbf{x} + \text{SelfAttention}(\mathbf{x})) \\ \mathbf{y} &= \text{LayerNorm}(\mathbf{z} + \text{FFN}(\mathbf{z})) \end{aligned}$$

LayerNorm:

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\text{LayerNorm} = \gamma \hat{\mathbf{x}} + \beta$$



(Review) Multihead-Attention | Transformers

- Different words in a sentence can relate to each other in many different ways simultaneously
 - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
 - >> **Heads** ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
 - >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

(Review) Multihead-Attention | Transformers

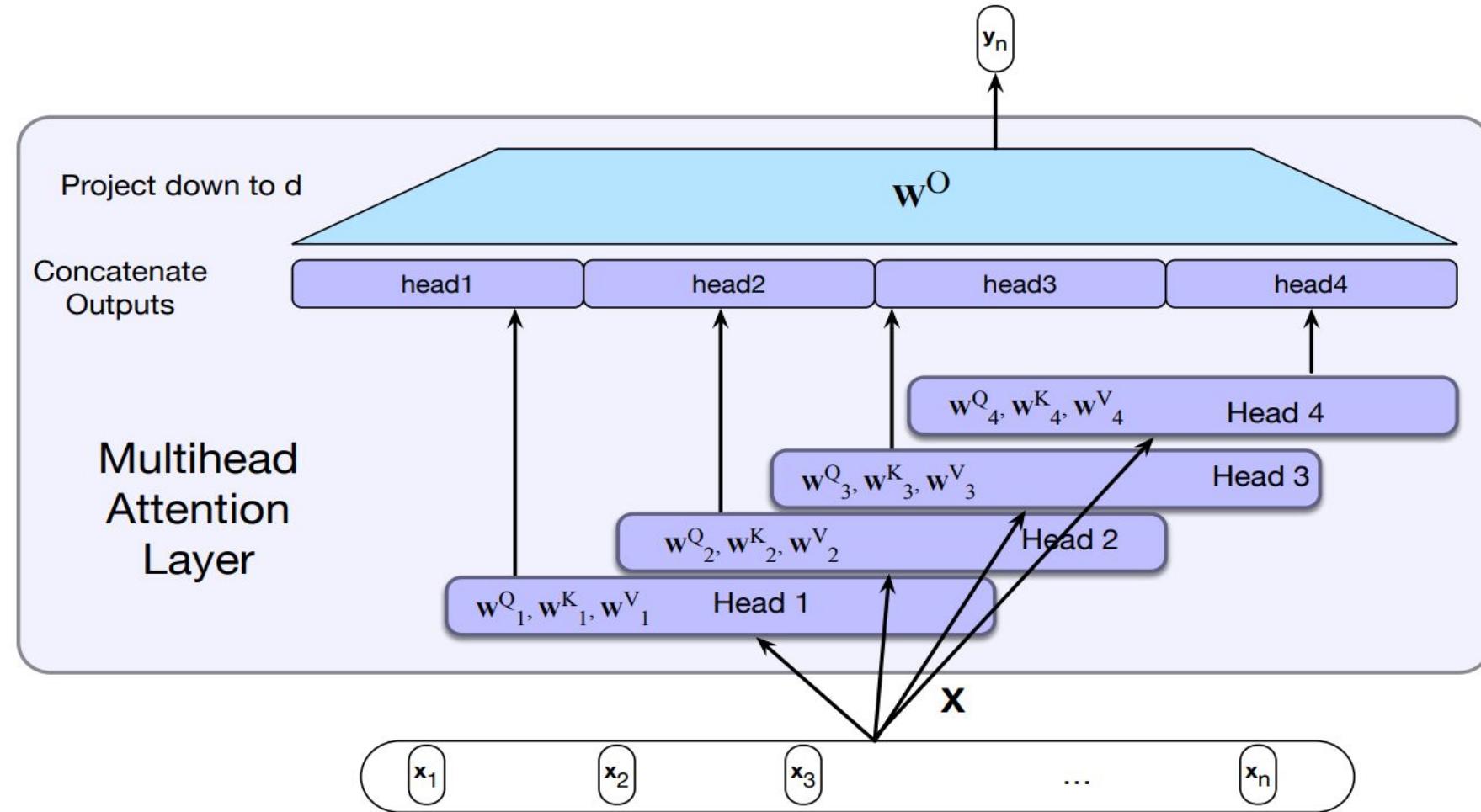
- Different words in a sentence can relate to each other in many different ways simultaneously
 - >> A single transformer block to learn to capture all of the different kinds of parallel relations among its inputs is inadequate.
- **Multihead** self-attention layers
 - >> **Heads** ⇒ sets of self-attention layers, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
 - >> Each head learn different aspects of the relationships that exist among inputs at the same level of abstraction

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X} \mathbf{W}_i^Q ; \mathbf{K} = \mathbf{X} \mathbf{W}_i^K ; \mathbf{V} = \mathbf{X} \mathbf{W}_i^V$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

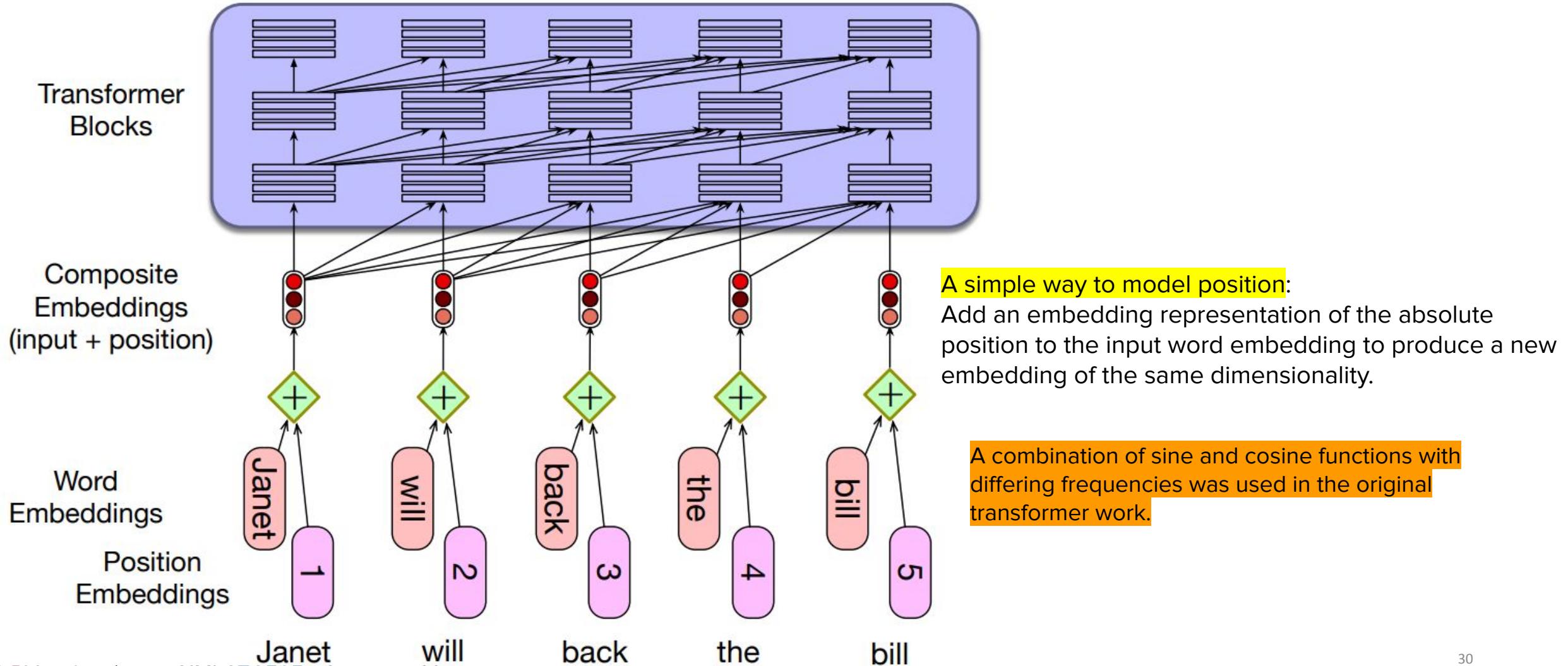
(Review) Multihead-Attention | Transformers



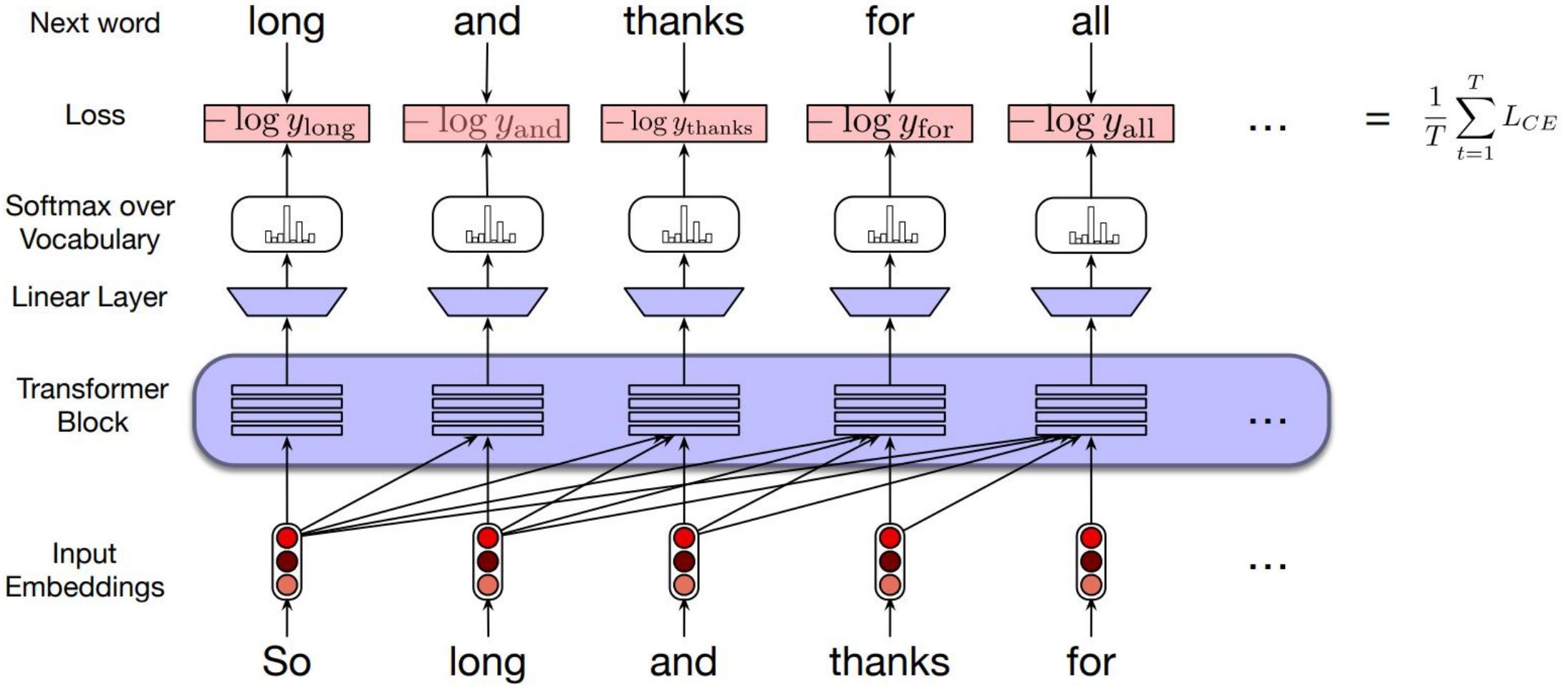
Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices.

The outputs from each of the layers are concatenated and then projected down to d , thus producing an output of the same size as the input so layers can be stacked.

(Review) Positional Embeddings | Transformers



(Review) Transformers as Language Models





Vision Transformers



Vision Transformer (ViT)

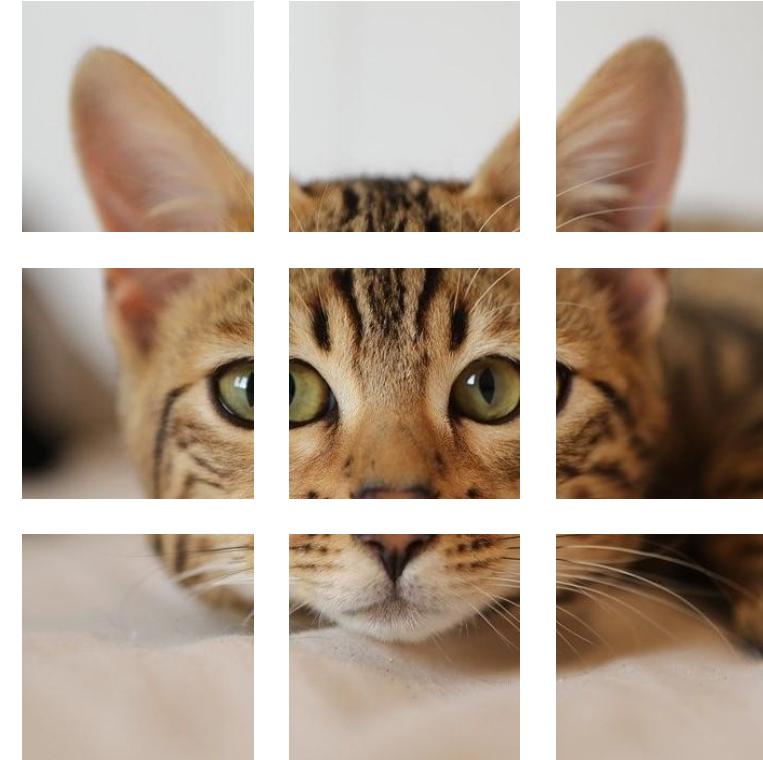


Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

[Cat image](#) is free for commercial use under a [Pixabay license](#)



Vision Transformer (ViT)



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

[Cat image](#) is free for commercial use under a [Pixabay license](#)



Vision Transformer (ViT)

N input patches, each
of shape 3x16x16

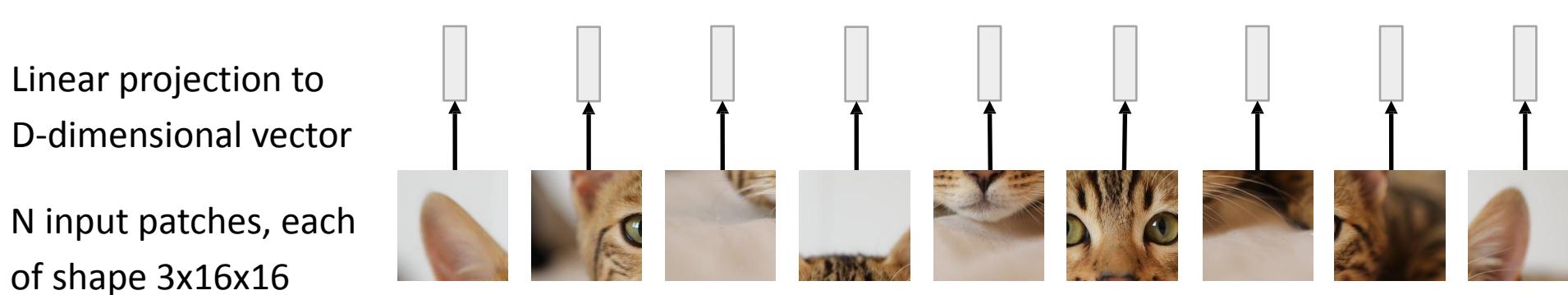


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
use under a [Pixabay license](#)



Vision Transformer (ViT)

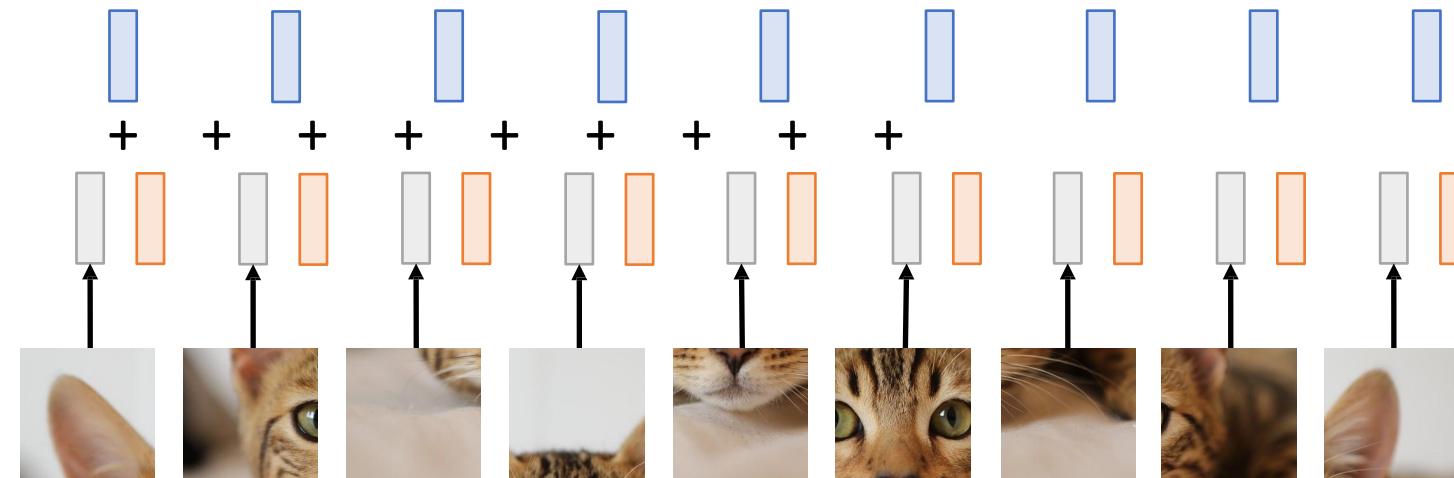


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial use under a [Pixabay license](#)

Vision Transformer (ViT)

Add positional embedding: learned D-dim vector per position



Linear projection to D-dimensional vector

N input patches, each of shape 3x16x16

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial use under a [Pixabay license](#)

Vision Transformer (ViT)

Output vectors



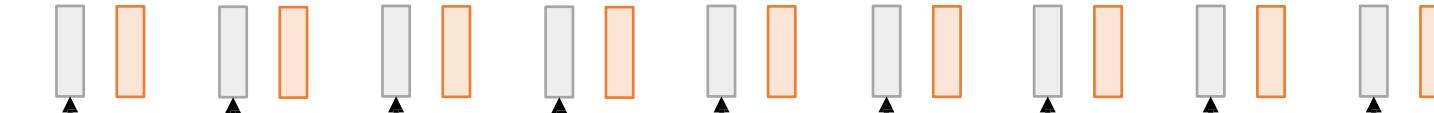
Exact same as
NLP

Transformer

Transformer!
Add positional
embedding: learned D-
dim vector per position



Linear projection to
D-dimensional vector



N input patches, each
of shape 3x16x16



[Cat image](#) is free for commercial
use under a [Pixabay license](#)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021



Vision Transformer (ViT)

Output vectors



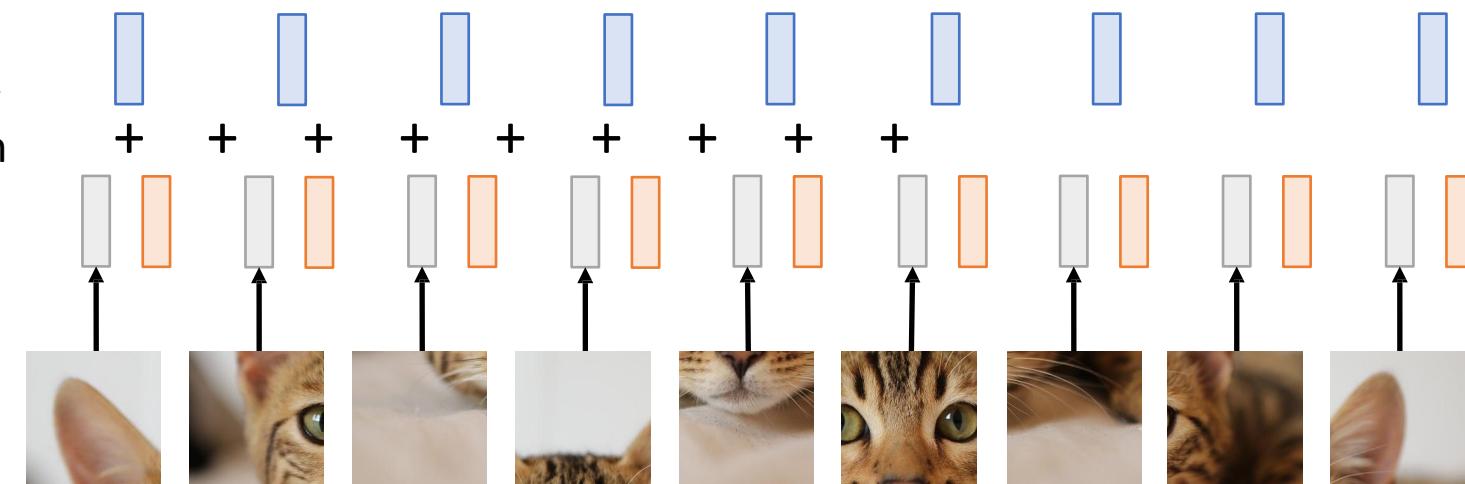
Exact same as
NLP

Transformer

Transformer!
Add positional
embedding: learned D-
dim vector per position

Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16

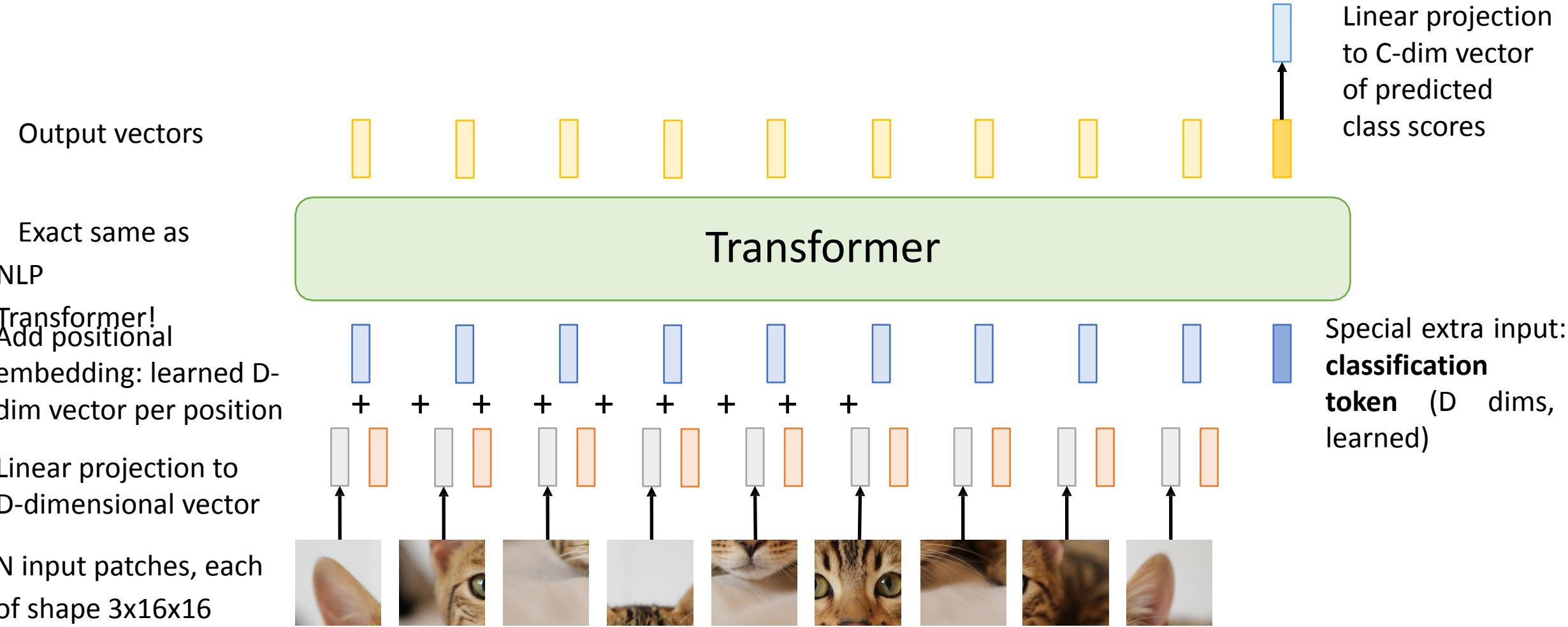


Special extra input:
**classification
token** (D dims,
learned)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
use under a [Pixabay license](#)

Vision Transformer (ViT)



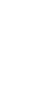
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial use under a [Pixabay license](#)

Vision Transformer (ViT)

Computer vision model
with no convolutions!

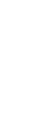
Output vectors



Exact same as
NLP

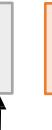
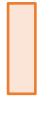
Transformer

Transformer!
Add positional
embedding: learned D-
dim vector per position

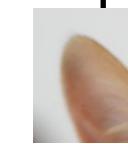


Special extra input:
**classification
token** (D dims,
learned)

Linear projection to
D-dimensional vector



N input patches, each
of shape 3x16x16



Cat image is free for commercial
use under a [Pixabay license](#)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT)

Computer vision model
with no convolutions!

Not quite: With patch size p , first
layer is Conv2D($p \times p$, 3-> D , stride= p)

Output vectors



Linear projection
to C -dim vector
of predicted
class scores

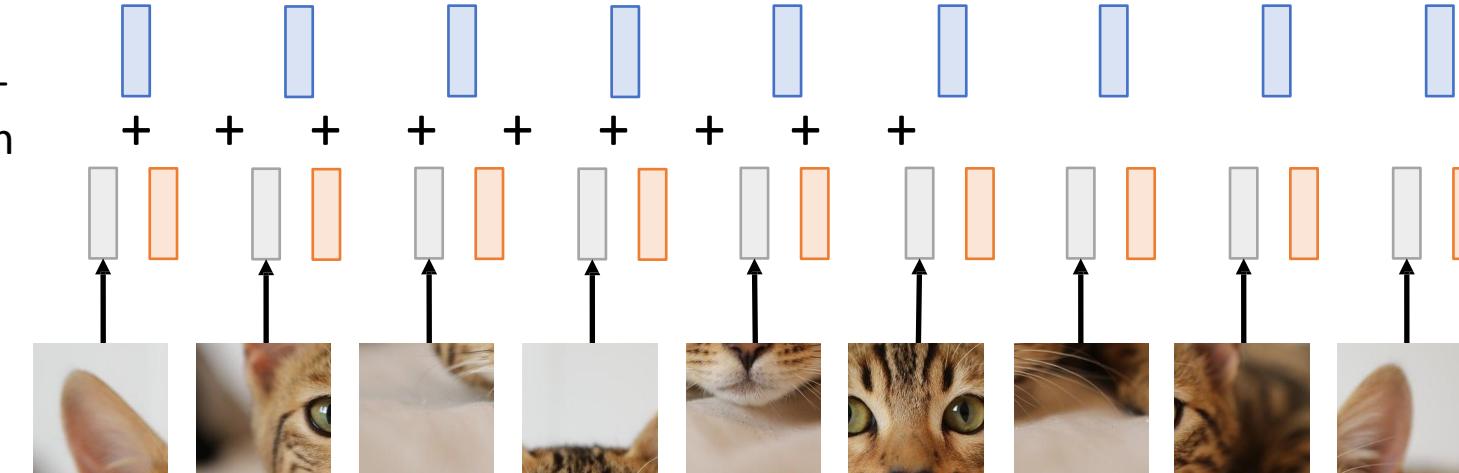
Exact same as
NLP

Transformer!
Add positional
embedding: learned D -
dim vector per position

Transformer

Linear projection to
 D -dimensional vector

N input patches, each
of shape $3 \times 16 \times 16$



Special extra input:
**classification
token** (D dims,
learned)

[Cat image](#) is free for commercial
use under a [Pixabay license](#)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT)

Computer vision model
with no convolutions!

Not quite: MLPs in
Transformer are stacks of 1x1
convolution

Output vectors



Linear projection
to C-dim vector
of predicted
class scores

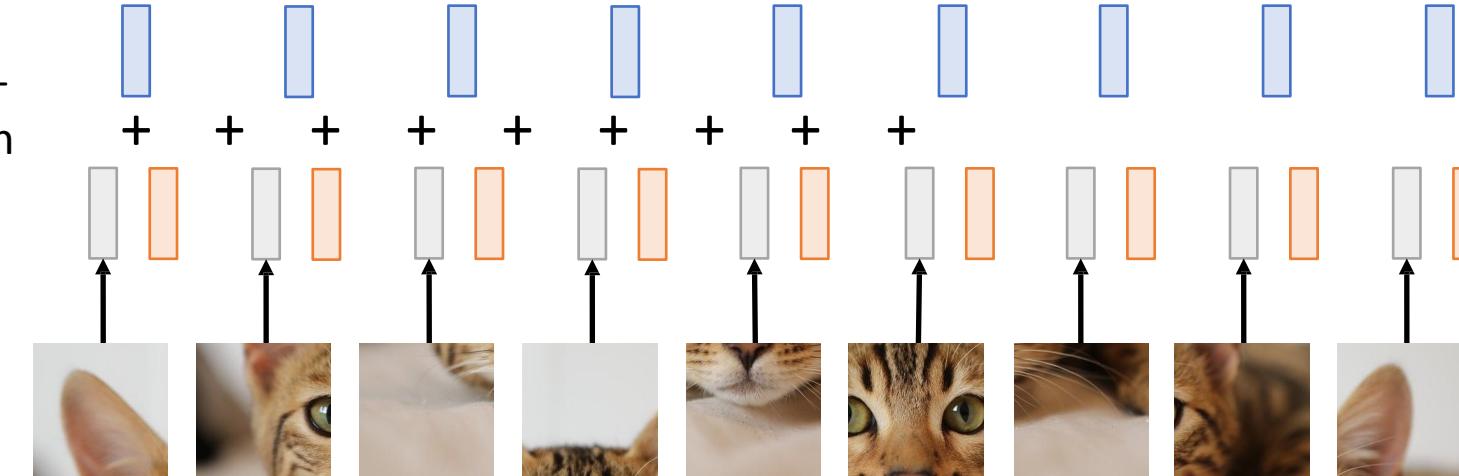
Exact same as
NLP

Transformer!
Add positional
embedding: learned D-
dim vector per position

Transformer

Linear projection to
D-dimensional vector

N input patches, each
of shape 3x16x16



Special extra input:
**classification
token** (D dims,
learned)

[Cat image](#) is free for commercial
use under a [Pixabay license](#)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer

In practice: take 224x224 input image,
(ViT)
 divide into 14x14 grid of 16x16 pixel
 patches (or 16x16 grid of 14x14 patches)

Output vectors

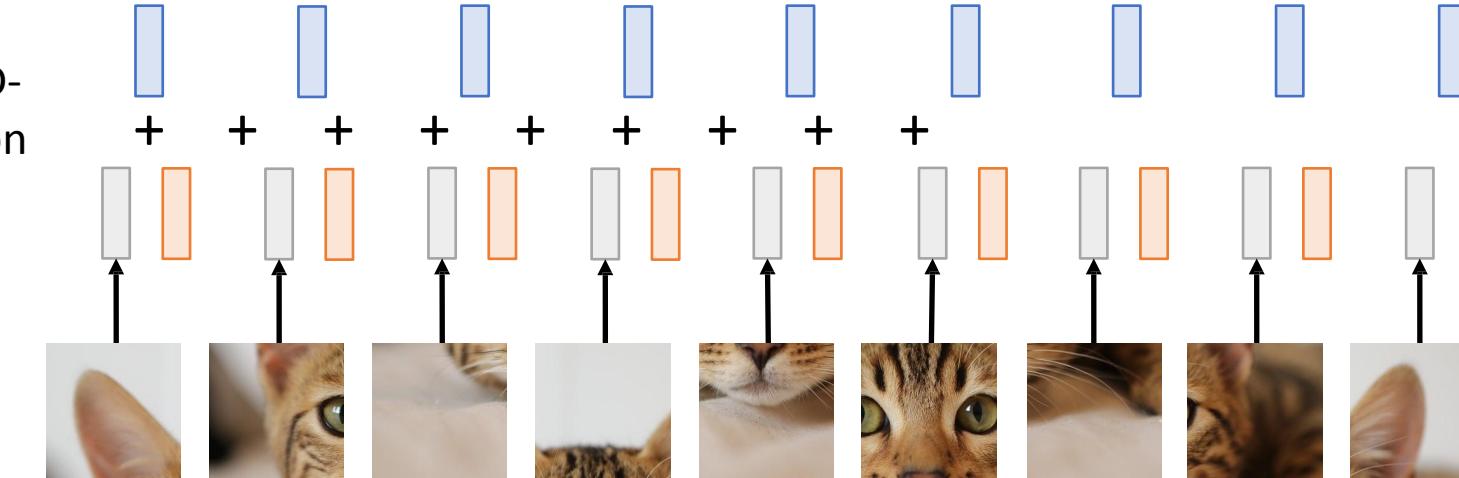


Exact same as
 NLP

Transformer!
 Add positional
 embedding: learned D-
 dim vector per position

Linear projection to
 D-dimensional vector

N input patches, each
 of shape 3x16x16



Each attention matrix has $14^4 = 38,416$
 entries, takes 150 KB
 (or 65,536 entries, takes 256 KB)

Linear projection
 to C-dim vector
 of predicted
 class scores

Special extra input:
**classification
 token** (D dims,
 learned)

Transformer

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
 use under a [Pixabay license](#)

Vision Transformer

In practice: take 224x224 input image,
(ViT)
 divide into 14x14 grid of 16x16 pixel
 patches (or 16x16 grid of 14x14 patches)

Output vectors



With 48 layers, 16 heads per
 layer, all attention matrices
 take 112 MB (or 192MB)

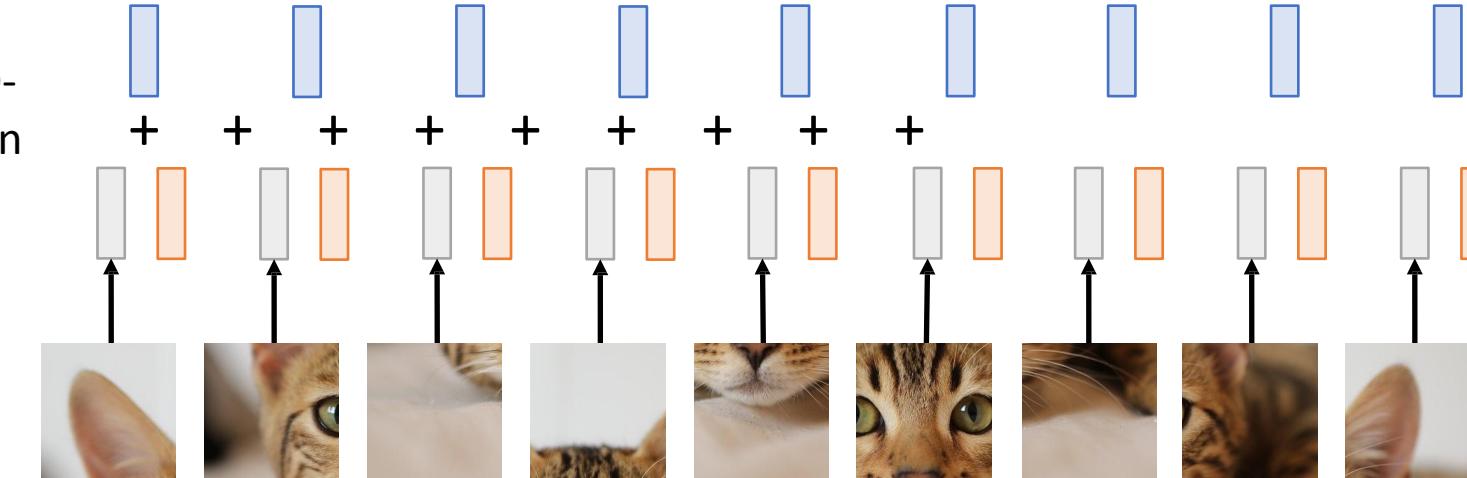
Linear projection
 to C-dim vector
 of predicted
 class scores

Exact same as
 NLP

Transformer!
 Add positional
 embedding: learned D-
 dim vector per position

Transformer

Linear projection to
 D-dimensional vector



N input patches, each
 of shape 3x16x16

Special extra input:
**classification
 token** (D dims,
 learned)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
 use under a [Pixabay license](#)

Vision Transformer

In practice: take 224x224 input image,
(ViT)
 divide into 14x14 grid of 16x16 pixel
 patches (or 16x16 grid of 14x14 patches)

Output vectors



With 48 layers, 16 heads per
 layer, all attention matrices
 take 112 MB (or 192MB)

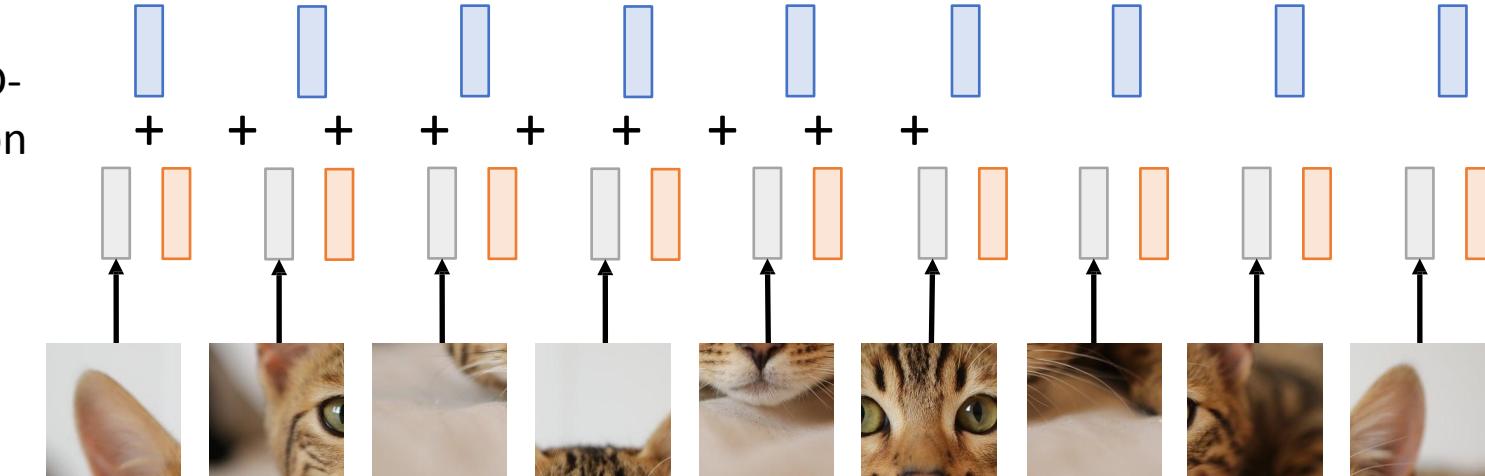
Linear projection
 to C-dim vector
 of predicted
 class scores

Exact same as
 NLP

Transformer!
 Add positional
 embedding: learned D-
 dim vector per position

Transformer

Linear projection to
 D-dimensional vector



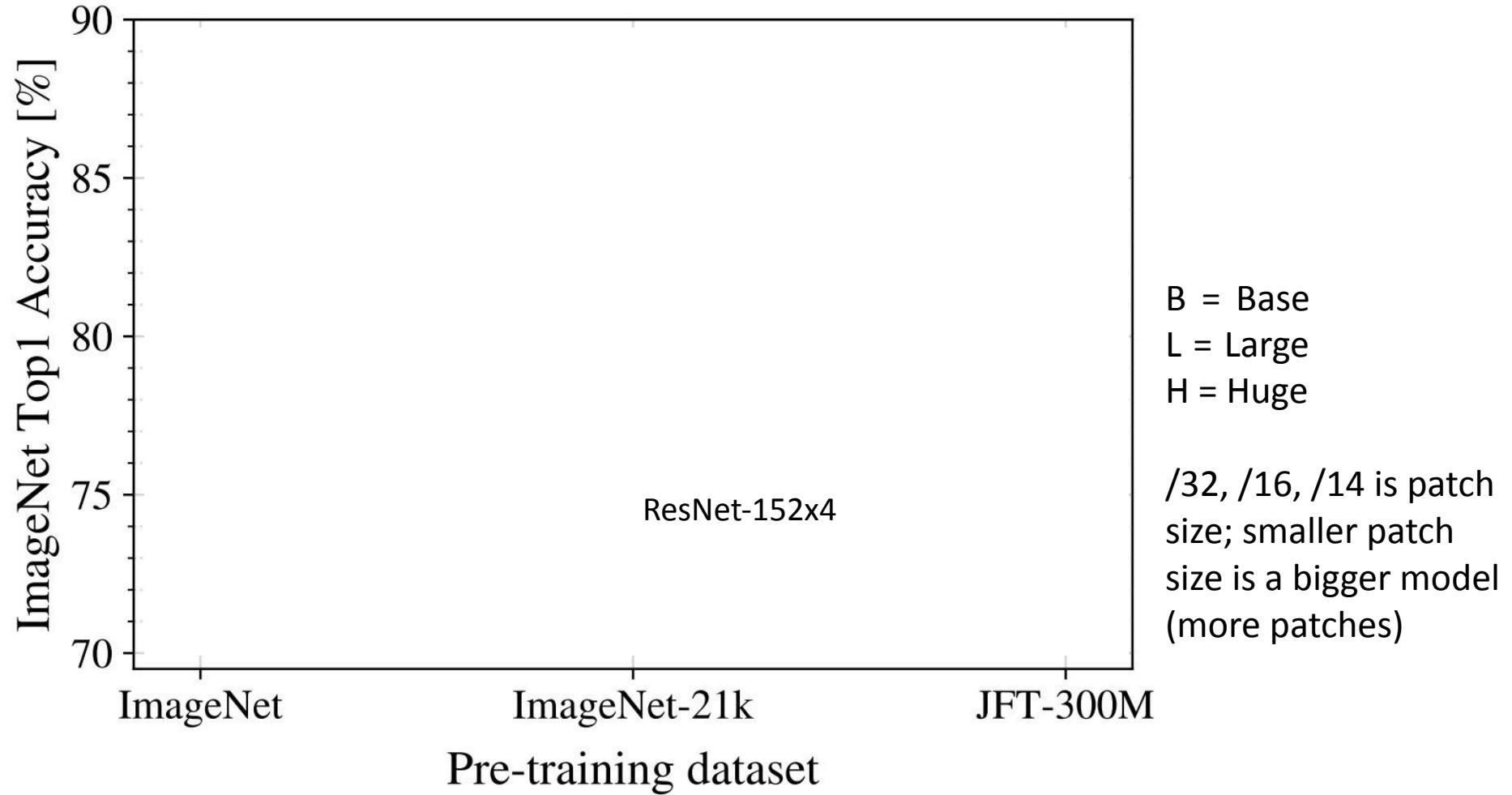
N input patches, each
 of shape 3x16x16

Special extra input:
**classification
 token** (D dims,
 learned)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

[Cat image](#) is free for commercial
 use under a [Pixabay license](#)

Vision Transformer (ViT) vs ResNets

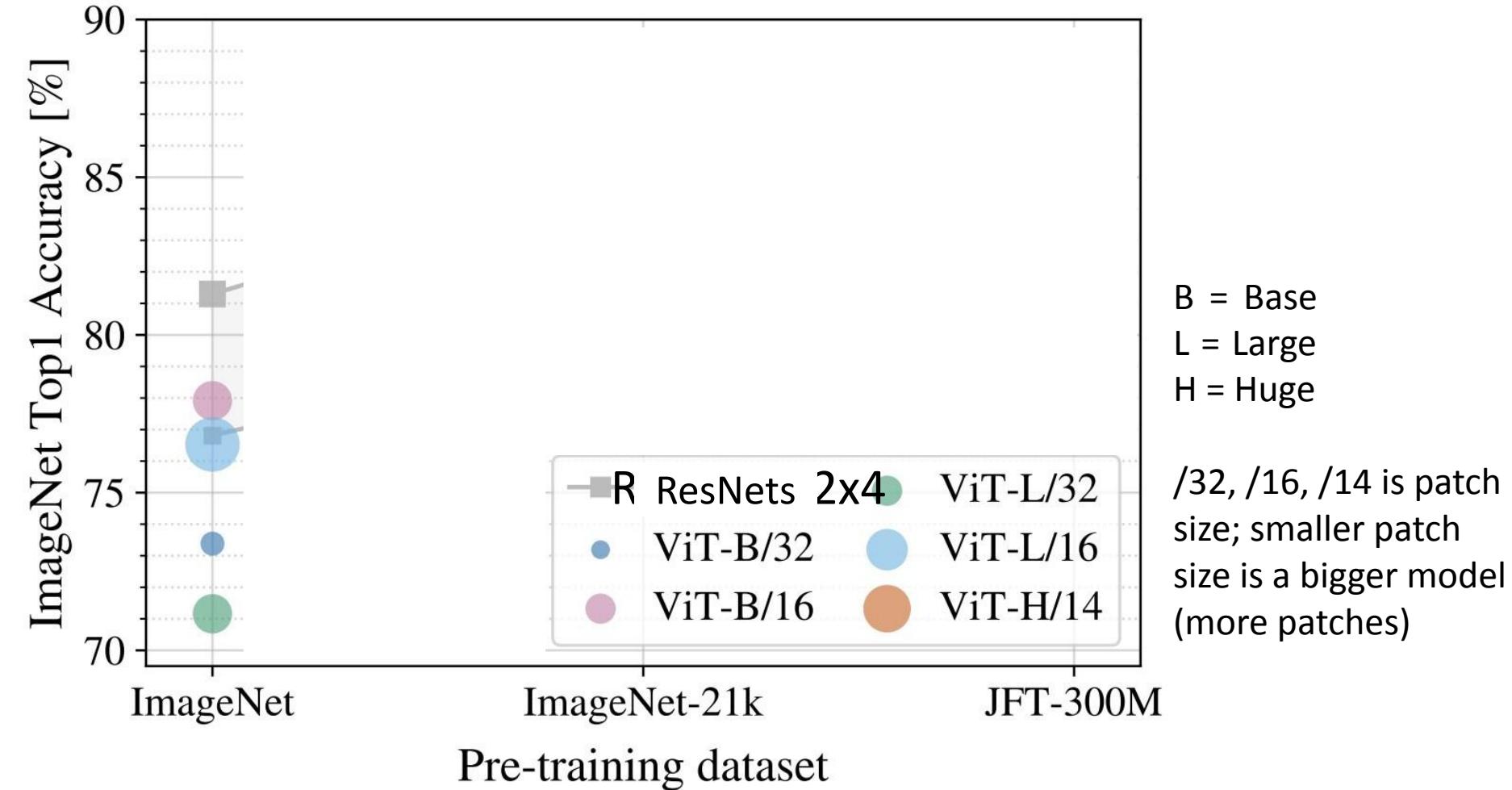


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

Recall: ImageNet dataset has 1k categories, 1.2M images

When trained on ImageNet, ViT models perform worse than ResNets

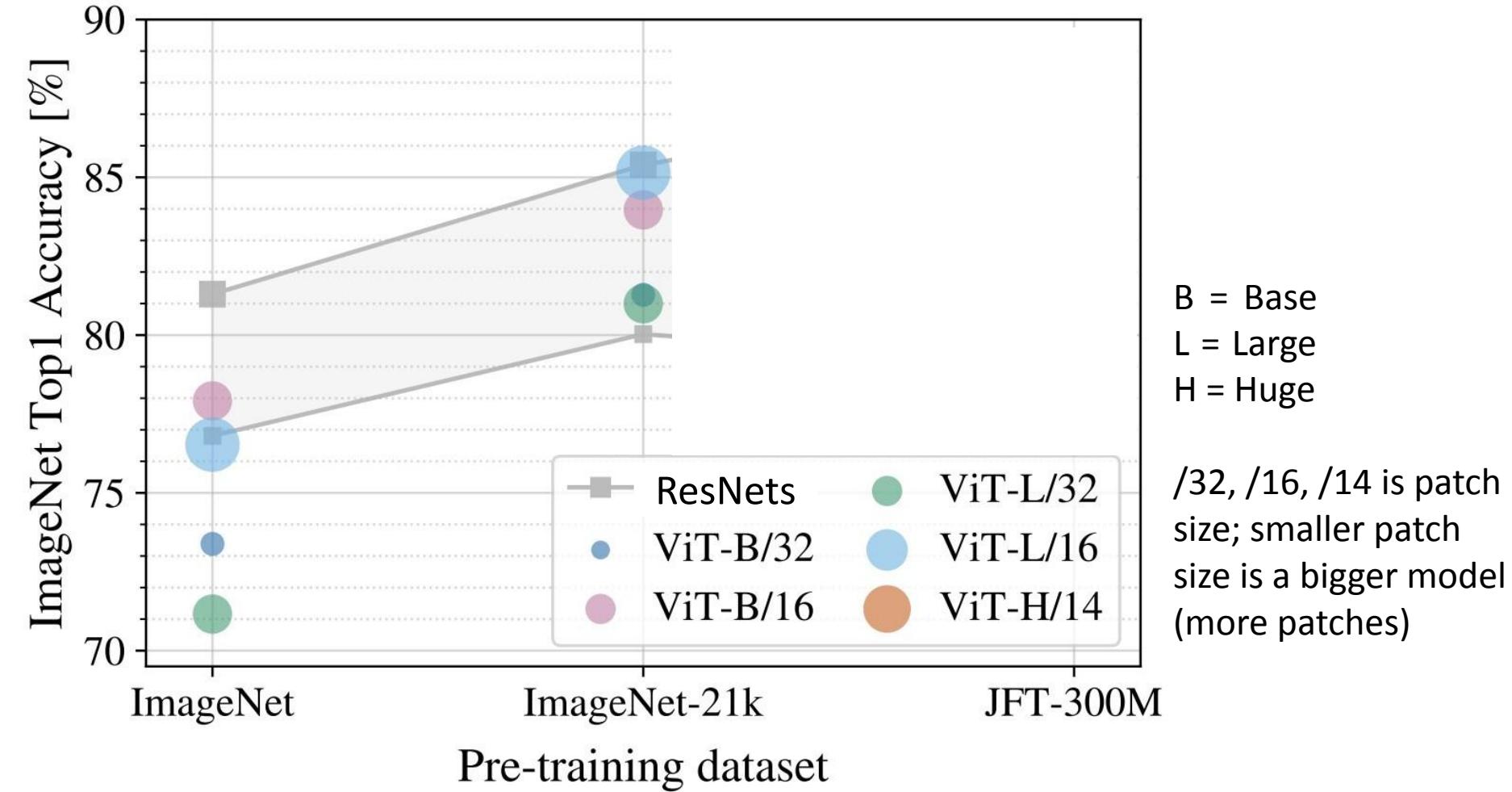


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

ImageNet-21k has 14M images with 21k categories

If you pretrain on ImageNet-21k and fine-tune on ImageNet, ViT does better: big ViTs match big ResNets

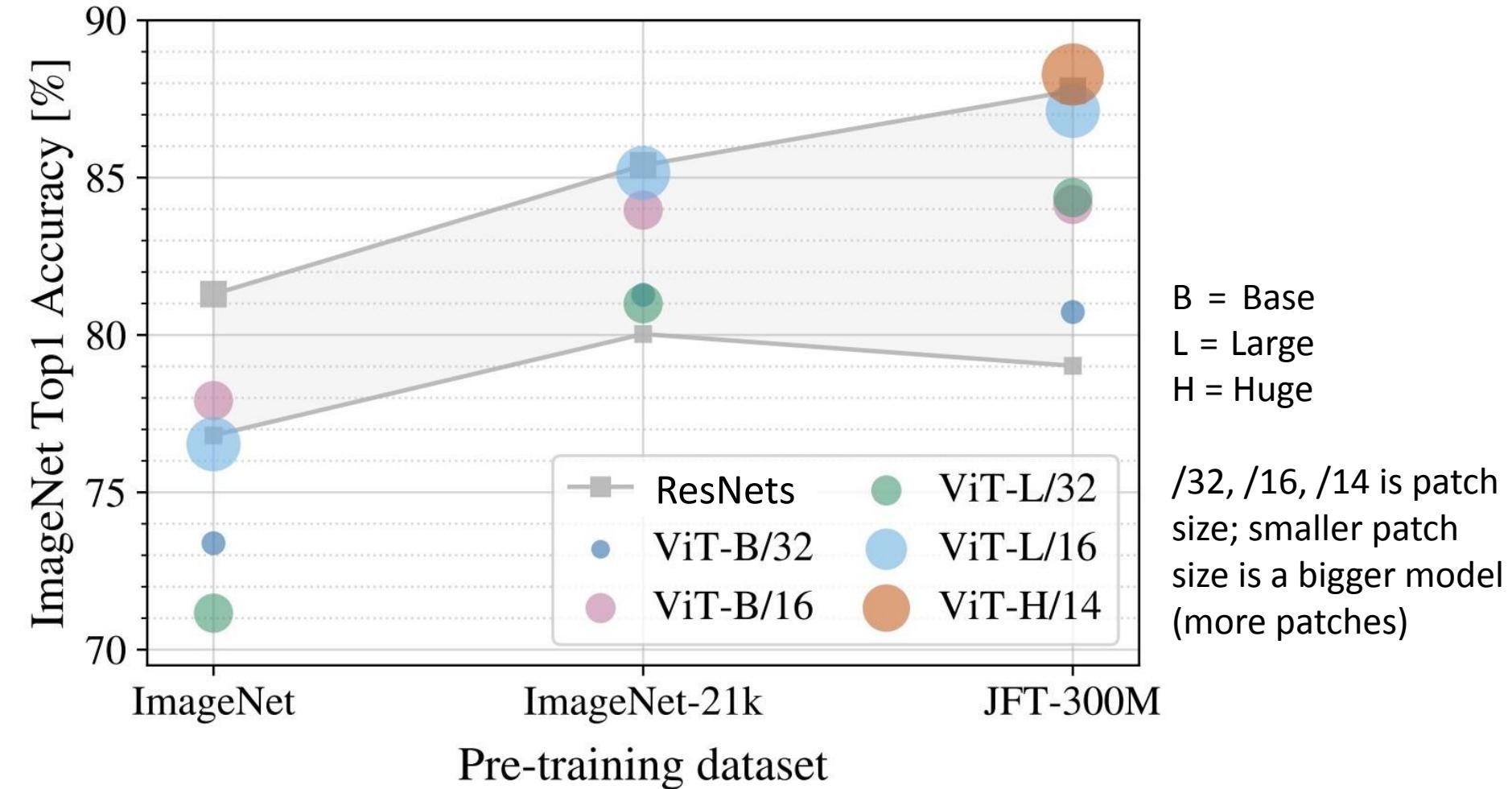


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets

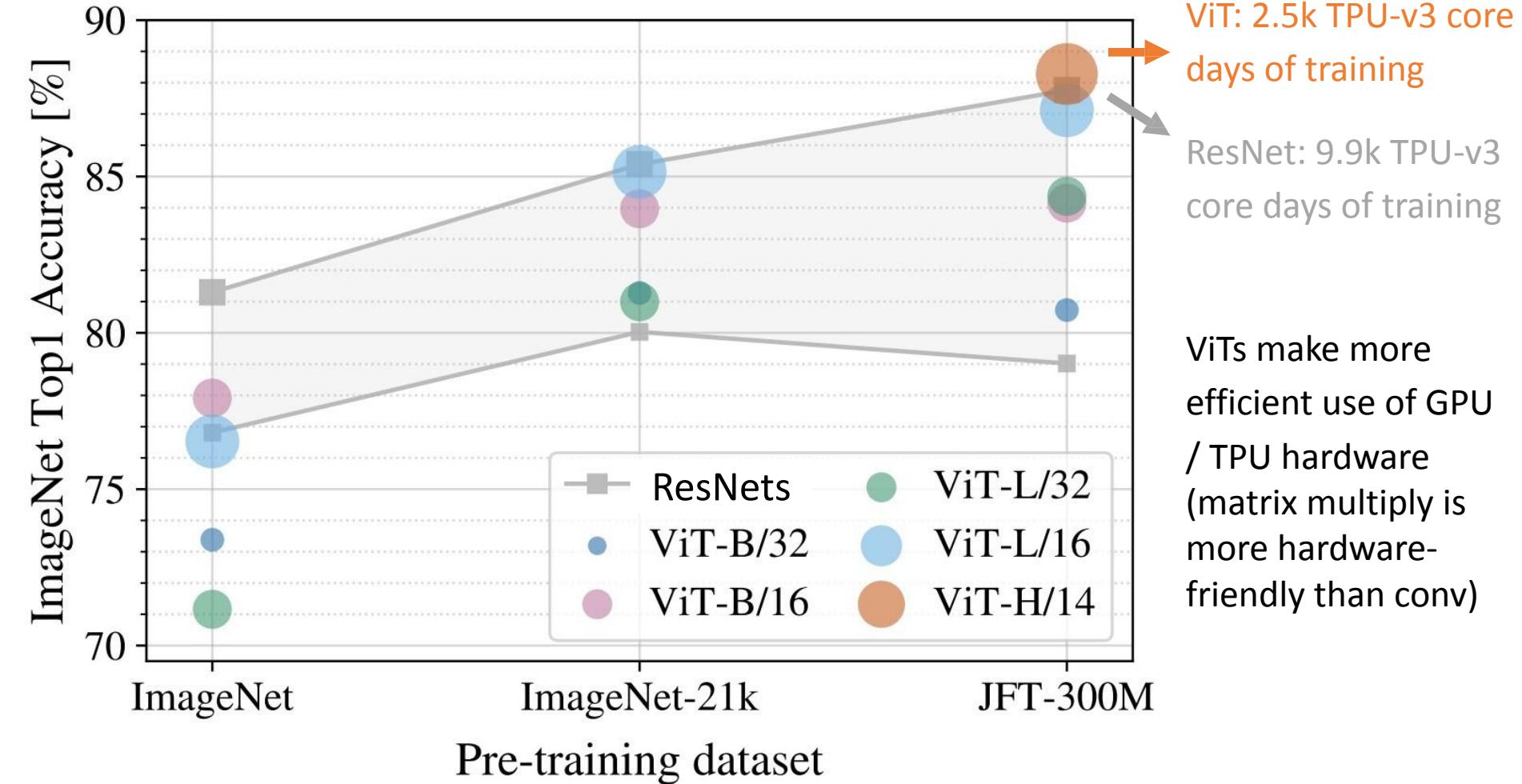


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



ViT: 2.5k TPU-v3 core days of training

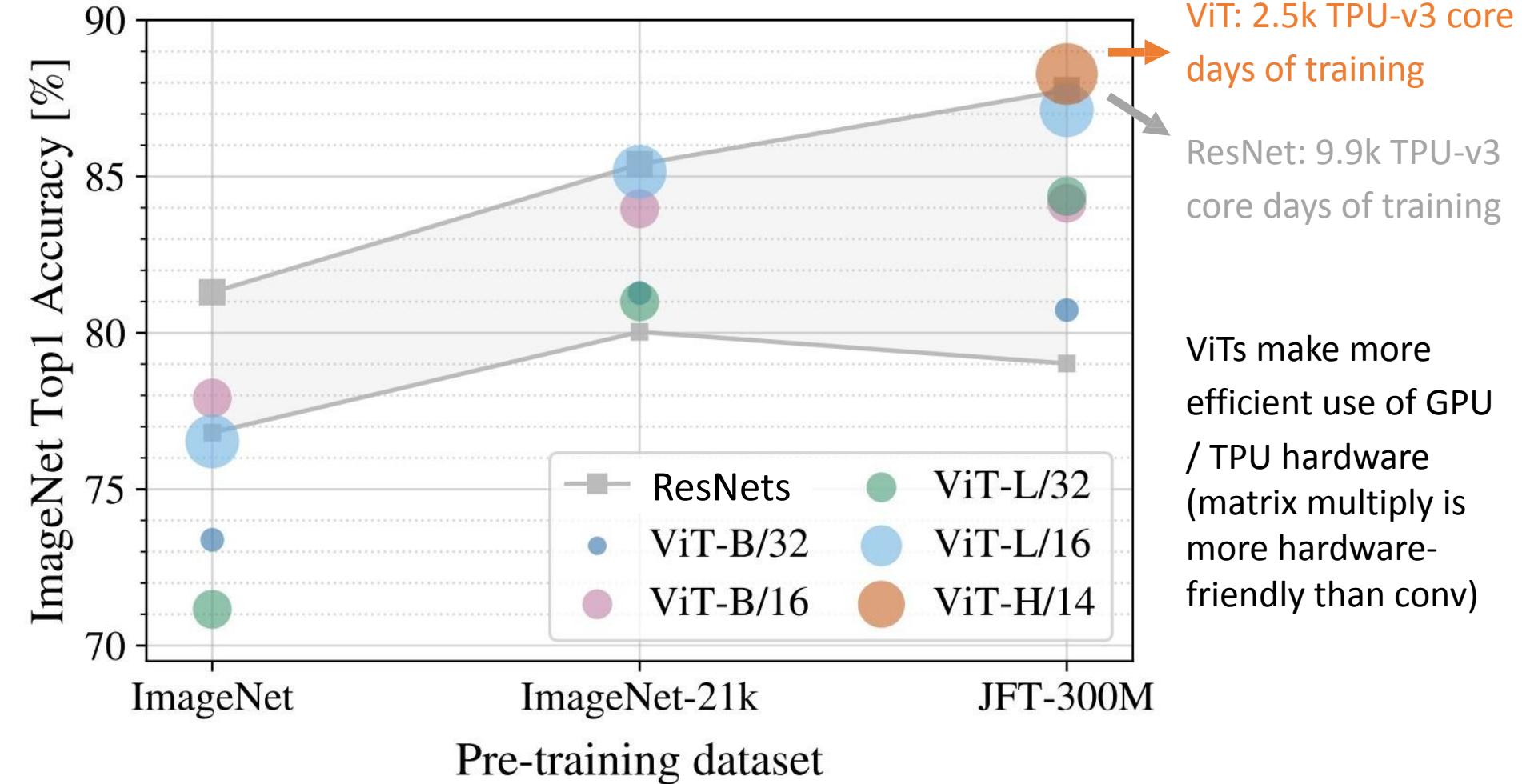
ResNet: 9.9k TPU-v3 core days of training

ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

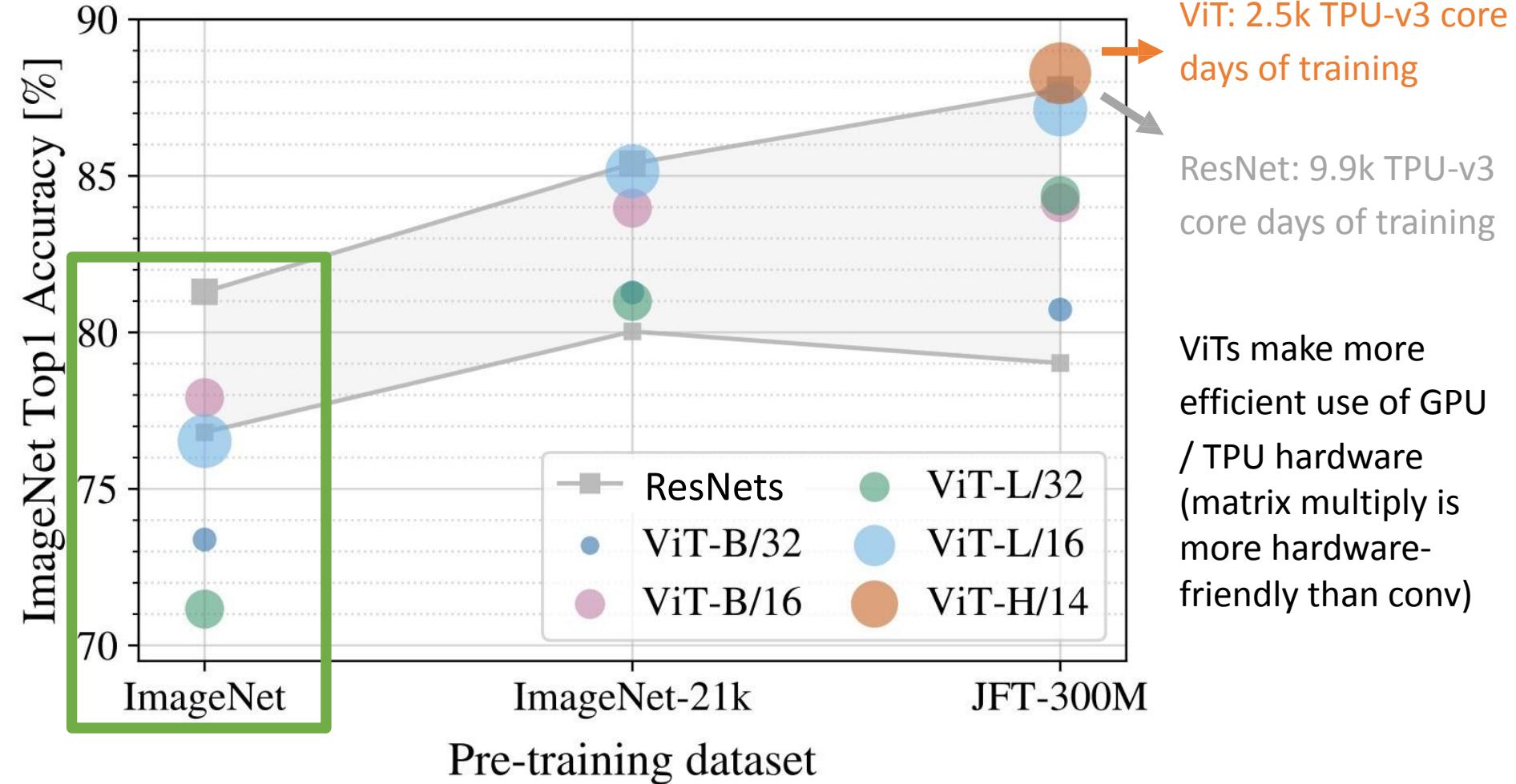
Claim: ViT models have “less inductive bias” than ResNets, so need more pretraining data to learn good features
(Not sure I buy this explanation: “inductive bias” is not a well-defined concept we can measure!)



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Vision Transformer (ViT) vs ResNets

How can we improve the performance of ViT models on ImageNet?



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021



- *Additional Readings:*
 - [Recommended] [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#) - Alexey Dosovitskiy, 2020, CVPR



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

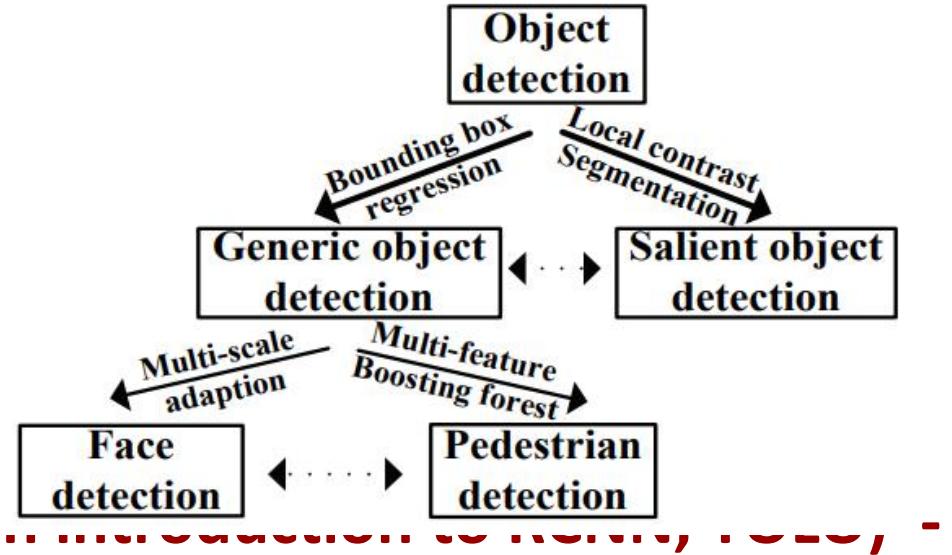
Thank you



Computer Vision

2023-24 First Semester, M.Tech (AIML)

Session #9-10: **Object Detection (wi... Incomplete Deck**



Topics

- Object Detection & Challenges
- RCNN & Faster RCNN
- Yolo & Its Variants
- Demonstration

Acknowledgement: Slide Materials adopted from - Intro to Computer Vision (Cornell Tech); Noah Snavely

An Approach

Examine every sub-window and determine if it is a tight box around an object



Yes



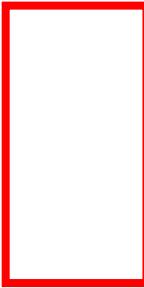
No?
Hold this thought



No

Sliding Window Classification

Let's assume we're looking for pedestrians in a box with a fixed aspect ratio.



Sliding Window

Key idea – just try all the subwindows in the image at all positions.



Generating hypotheses

Key idea – just try all the subwindows in the image at all positions and scales.



Note – Template did not change size

Each window classified separately



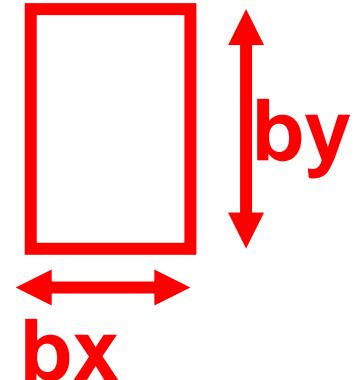
Slide credit: J. Hays

How Many Boxes Are There?

Given a $H \times W$ image and a “template” of size by by bx .

**Q. How many sub-boxes are there of size
(by, bx)?**

A. $(H/by)^*(W/bx)$



This is before considering adding:

- scales (by^*s, bx^*s)
- aspect ratios (by^*sy, bx^*sx)

Challenges of Object Detection

- Have to evaluate *tons* of boxes
- Positive instances of objects are *extremely* rare



How many ways can we get the box wrong?

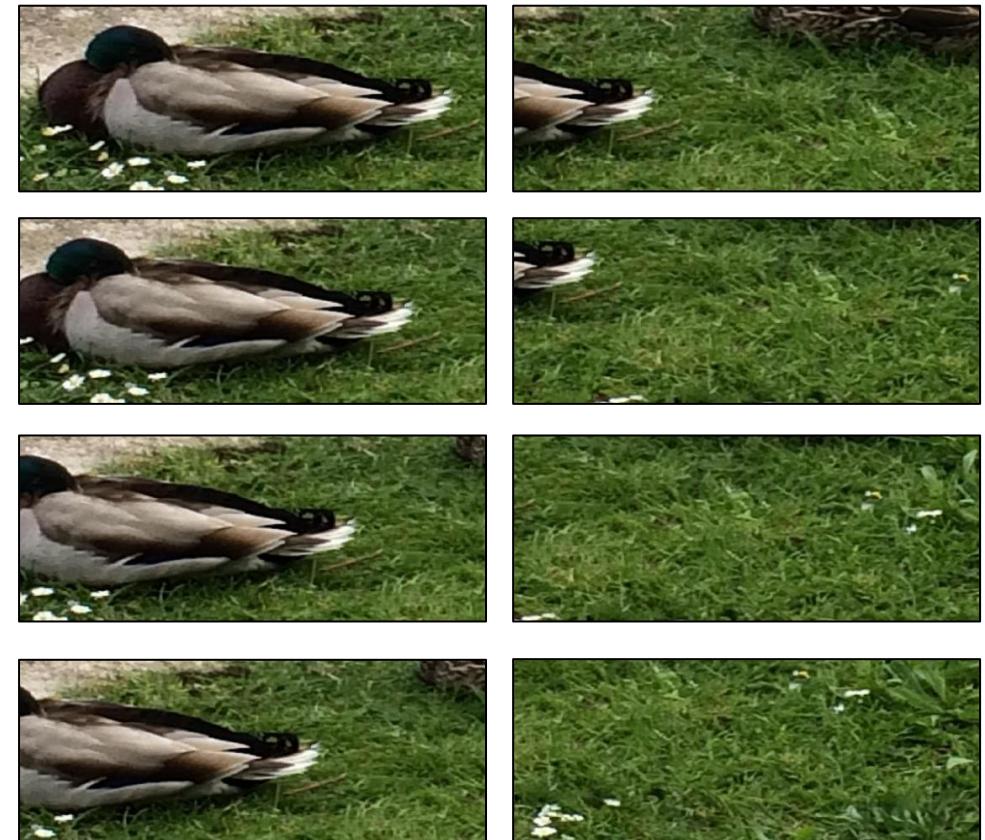
1. Wrong left x
2. Wrong right x
3. Wrong top y
4. Wrong bottom y

Are You Smarter than a Random Number Generator?

- **Prob. of guessing 1k-way classification?**
 - 1/1,000
- **Prob. of guessing all 4 bounding box corners within 10% of image size?**
 - $(1/10)*(1/10)*(1/10)*(1/10)=1/10,000$
 - Probability of guessing both: 1/10,000,000
 - Detection is **hard** (via guessing and in general)
 - Should *always compare* against guessing or picking most likely output label

Evaluating – Bounding Boxes

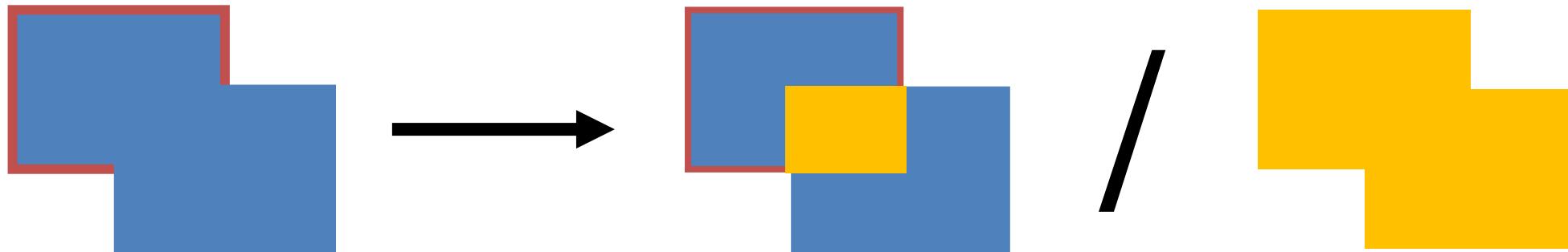
Raise your hand when you think the detection stops being correct.



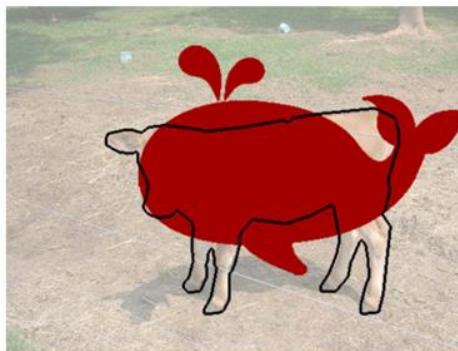
Evaluating – Bounding Boxes

Standard metric for two boxes:

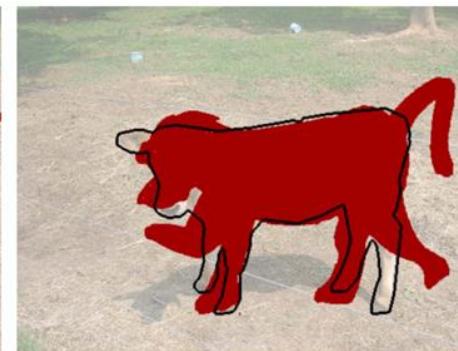
Intersection over union/IoU/Jaccard coefficient



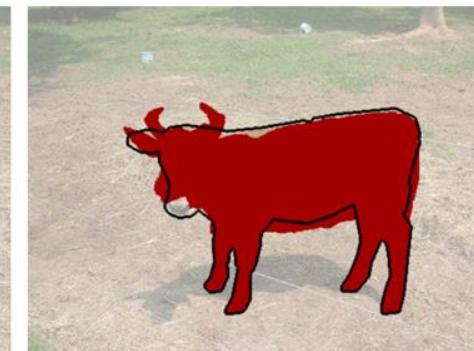
(a) Ground truth



(b) $\mathcal{J} = 0.554$



(c) $\mathcal{J} = 0.703$

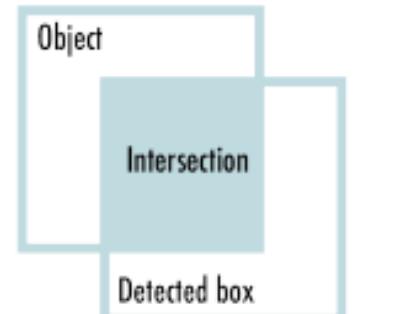


(d) $\mathcal{J} = 0.910$

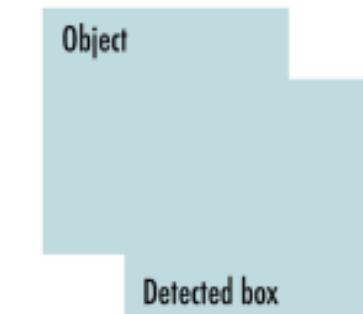
Evaluating – Bounding Boxes

IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box. It measures the overlap between the ground truth and predicted bounding boxes.

a)

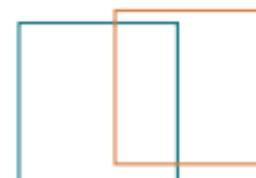


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



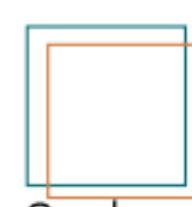
b)

IoU= 0.35



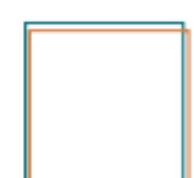
Poor

IoU= 0.74



Good

IoU= 0.93



Excellent

a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; b) examples of three different IoU values for different box locations.

Evaluating Performance

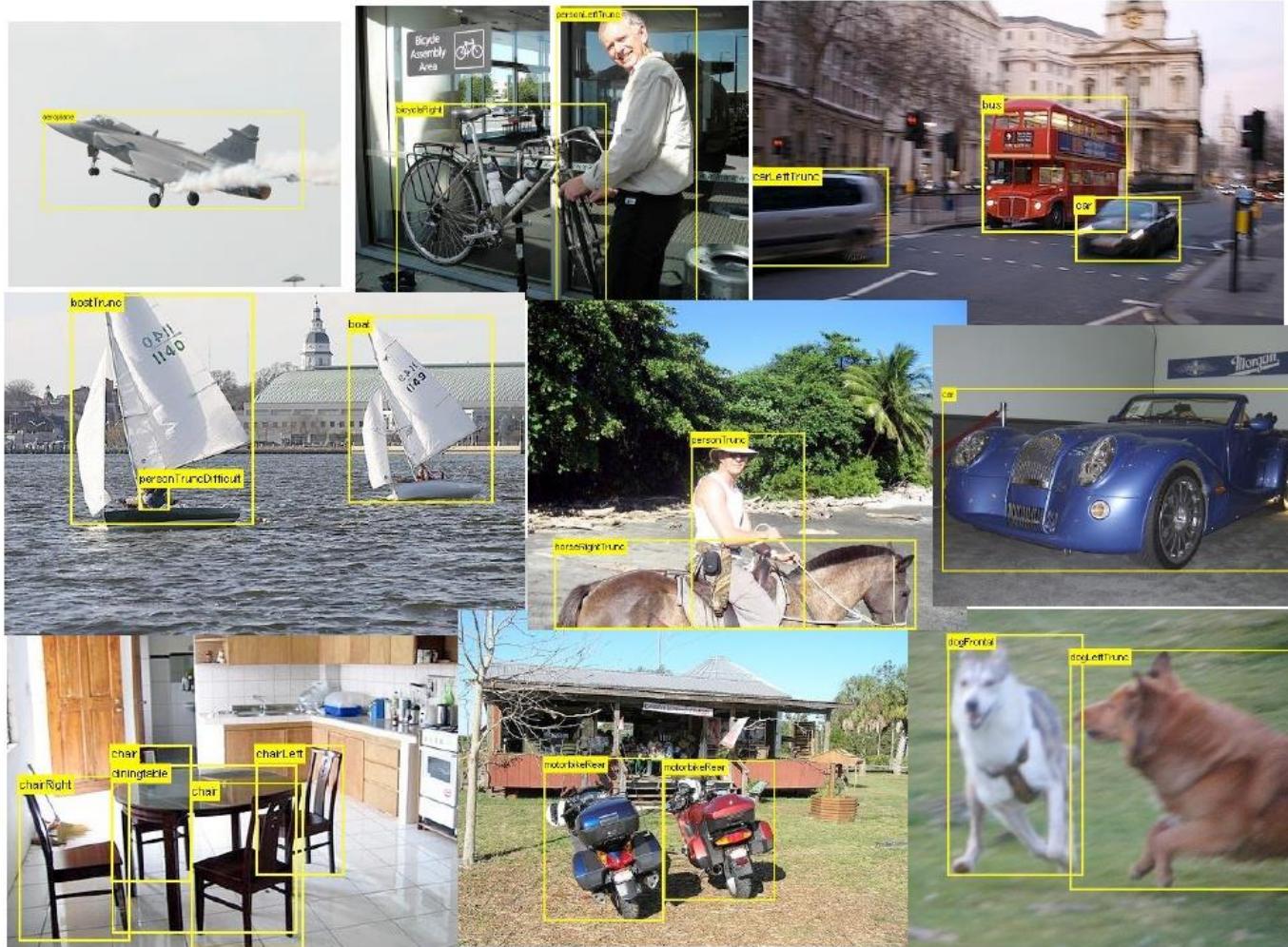
- Remember: accuracy = average of whether prediction is correct
- Suppose I have a system that gets 99% accuracy in person detection.
- **What's wrong?**
- I can get that by just saying no object everywhere!

Evaluating Performance

- True detection aka true positive: high IoU
- Precision: #true detections / #detections by detector
- Recall: #true detections / #ground truth positives



Generic object detection



Histograms of oriented gradients (HOG)

Partition image into blocks and compute histogram of gradient orientations in each block

$H \times W \times 3$ Image



$H' \times W' \times C'$ Image

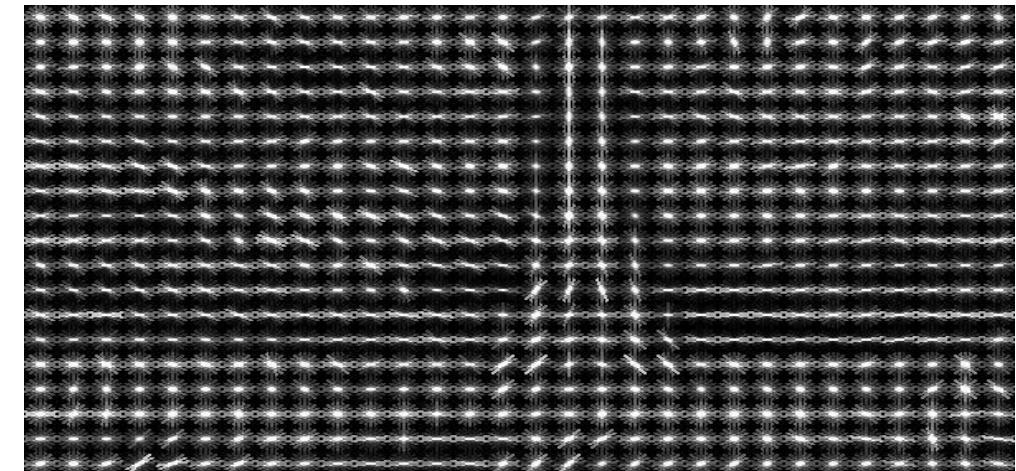


Image credit: N. Snavely

N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Slide Credit: S. Lazebnik

Pedestrian detection with HOG

- Train a pedestrian template using a linear support vector machine

positive training examples



negative training examples



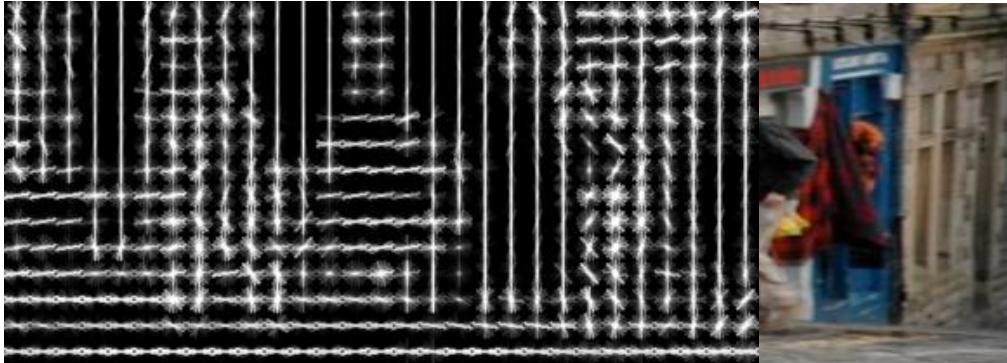
N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Slide Credit: S. Lazebnik

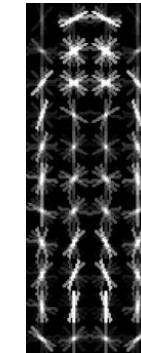
Pedestrian detection with HOG

- Train pedestrian “template” using a linear svm
- At test time, convolve feature map with template
- Find local maxima of response
- For multi-scale detection, repeat over multiple levels of a HOG *pyramid*

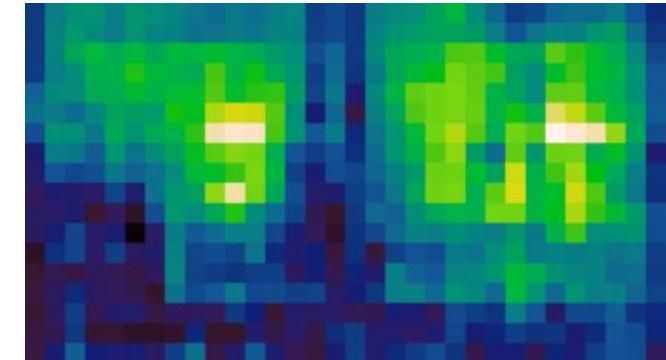
HOG feature map



Template



Detector response map



N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

Slide Credit: S. Lazebnik

Example detections



[Dalal and Triggs, CVPR 2005]

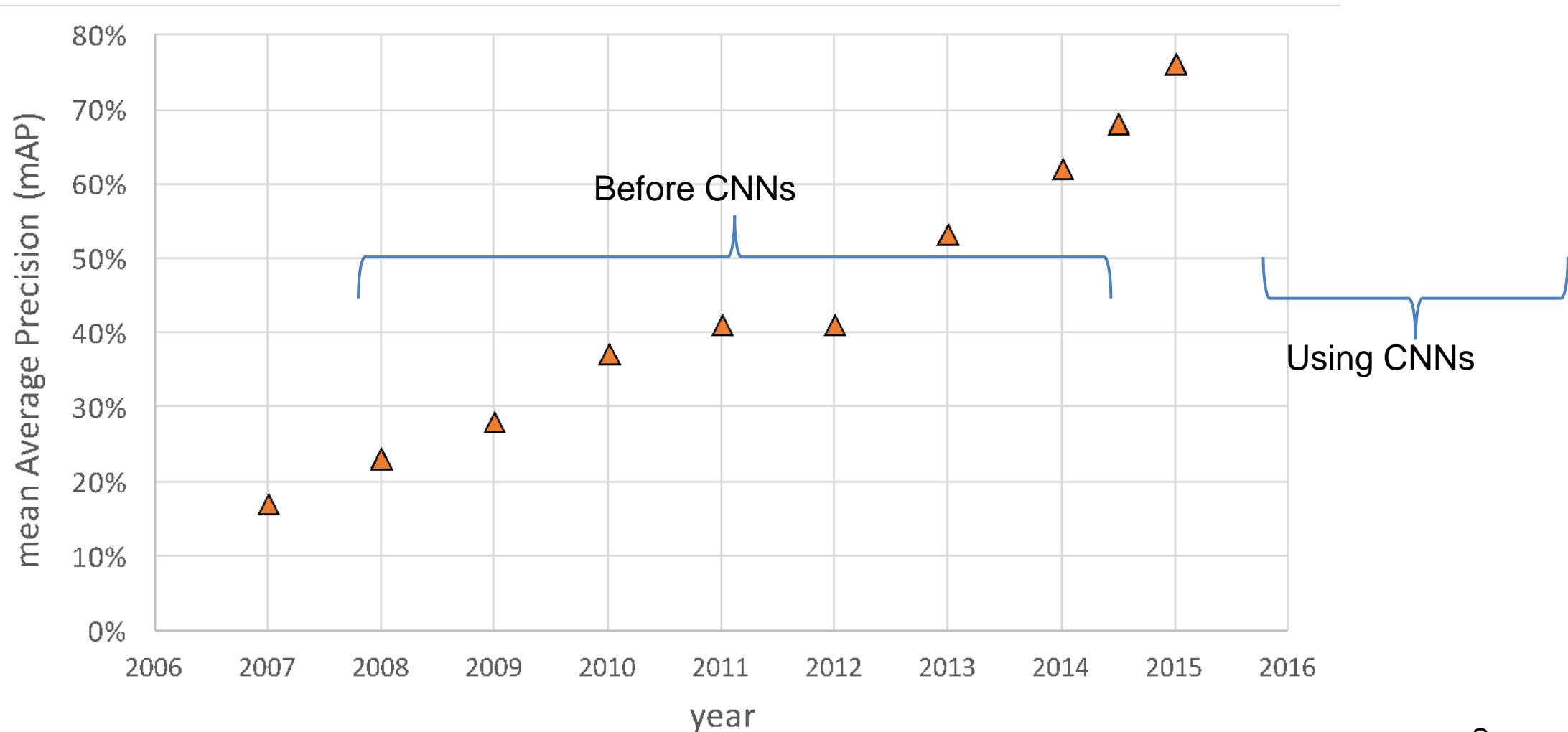
PASCAL VOC Challenge (2005-2012)



- 20 challenge classes:
- *Person*
- *Animals*: bird, cat, cow, dog, horse, sheep
- *Vehicles*: aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor*: bottle, chair, dining table, potted plant, sofa, tv/monitor
- Dataset size (by 2012): 11.5K training/validation images, 27K bounding boxes, 7K segmentations

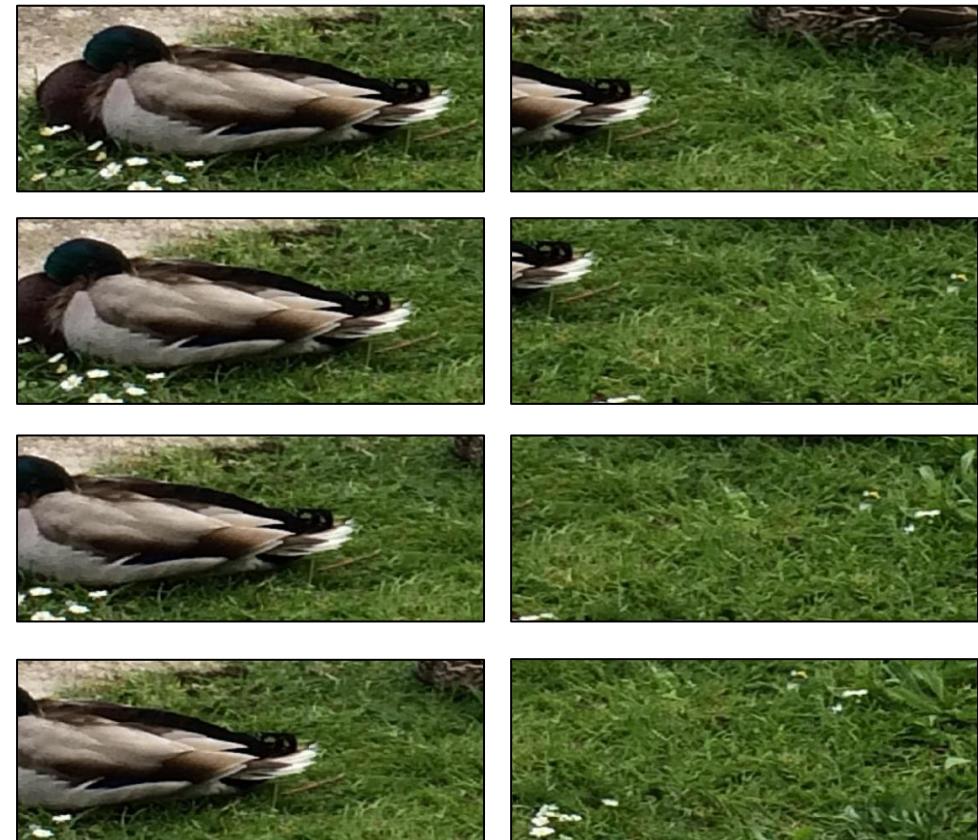
Object detection progress

PASCAL VOC

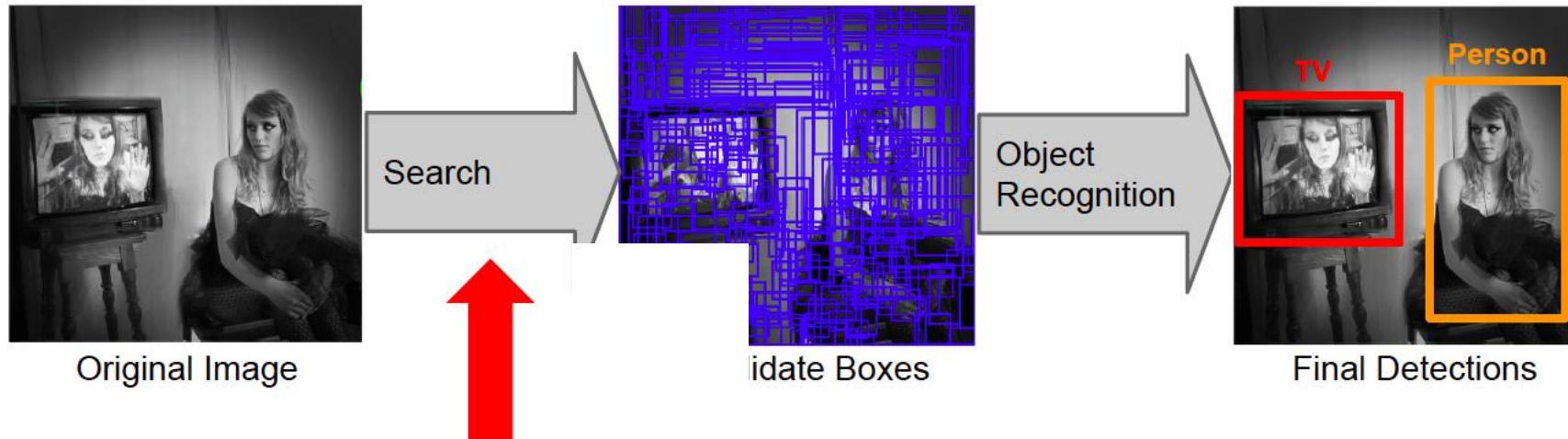


Region Proposals

Do I need to spend a lot of time filtering all the boxes covering grass?



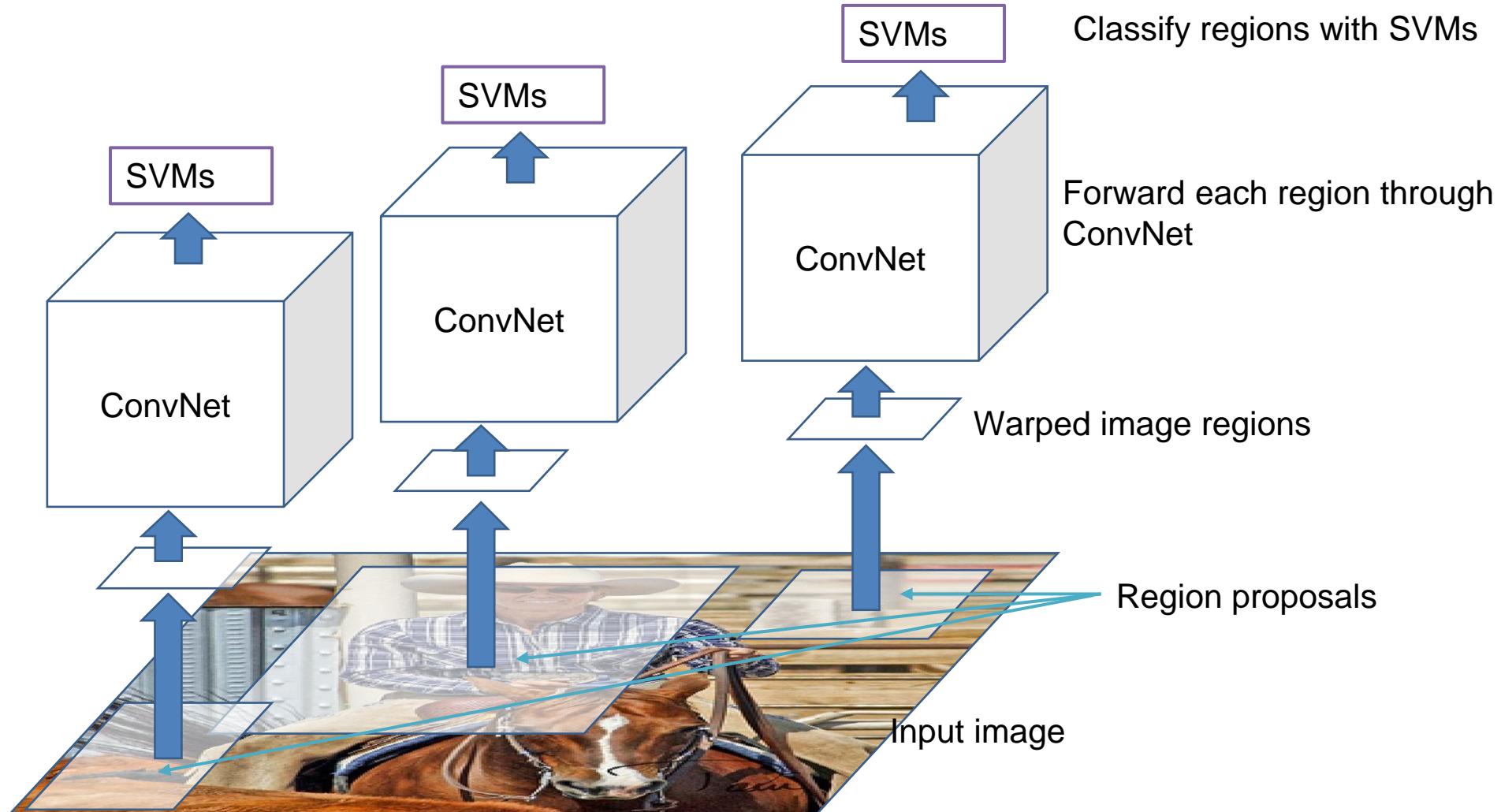
Region Proposals



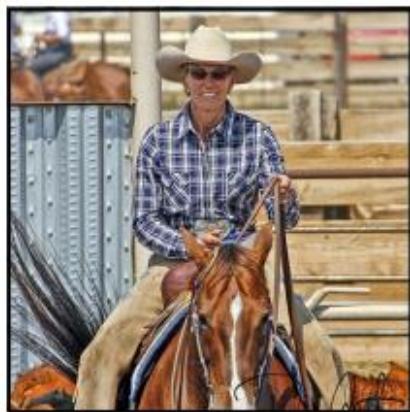
- As an alternative to sliding window search, evaluate a few hundred *region proposals*
 - Can use slower but more powerful features and classifiers
 - Proposal mechanism can be category-independent
 - Proposal mechanism can be trained

R-CNN: Region proposals + CNN features

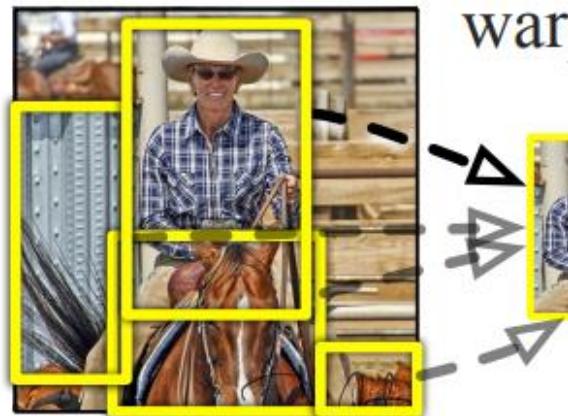
Source: R. Girshick



R-CNN: *Regions with CNN features*

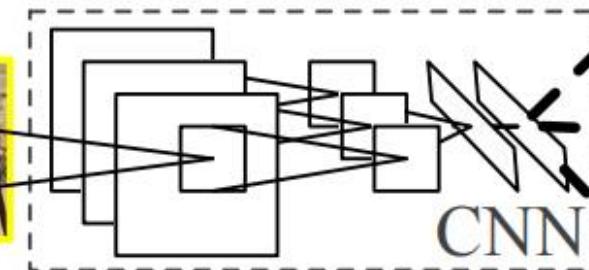


1. Input
image



2. Extract region
proposals (~2k)

warped region



CNN

aeroplane? no.

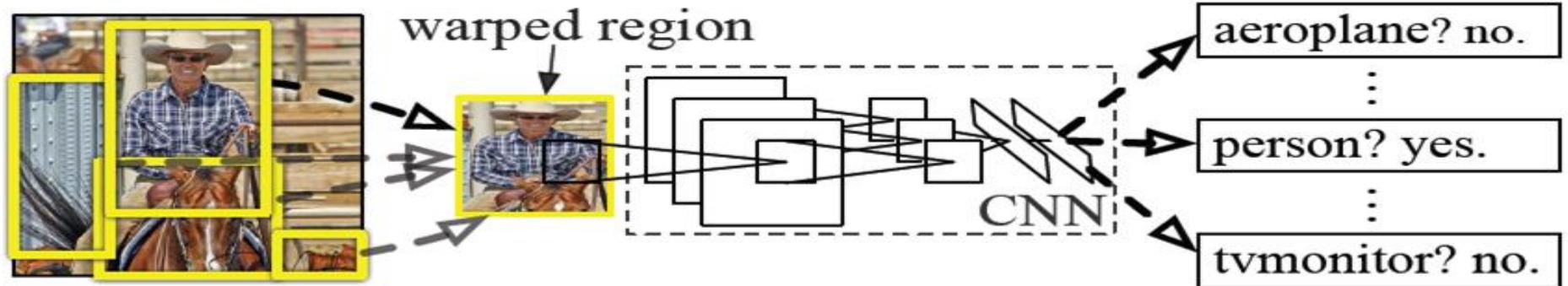
person? yes.

tvmonitor? no.

3. Compute
CNN features

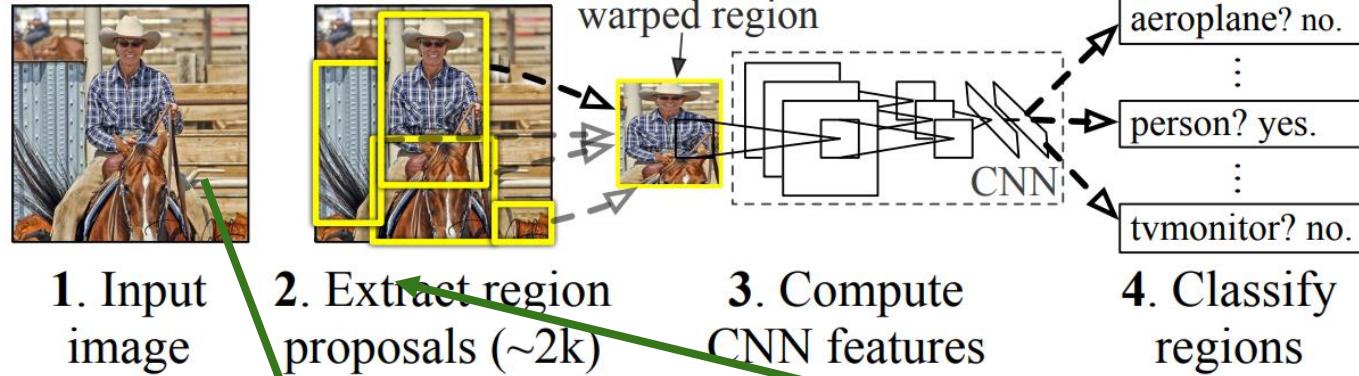
4. Classify
regions

R-CNN details



- **Regions:** ~2000 Selective Search proposals
- **Network:** AlexNet *pre-trained* on ImageNet (1000 classes), *fine-tuned* on PASCAL (21 classes)
- **Final detector:** warp proposal regions, extract fc7 network activations (4096 dimensions), classify with linear SVM
- **Bounding box regression** to refine box locations
- **Performance:** mAP of 53.7% on PASCAL 2010 (vs. 35.1% for Selective Search and 33.4% for DPM).

R-CNN: *Regions with CNN features*

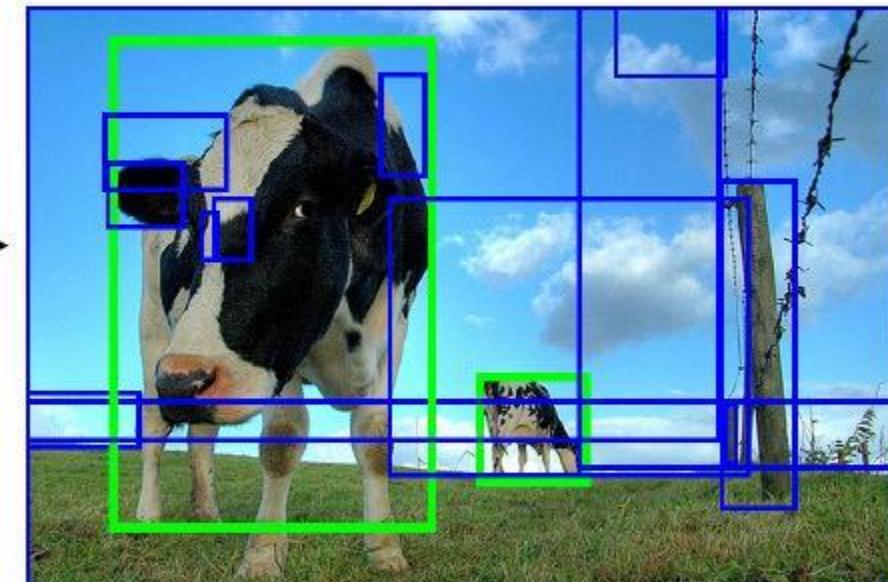
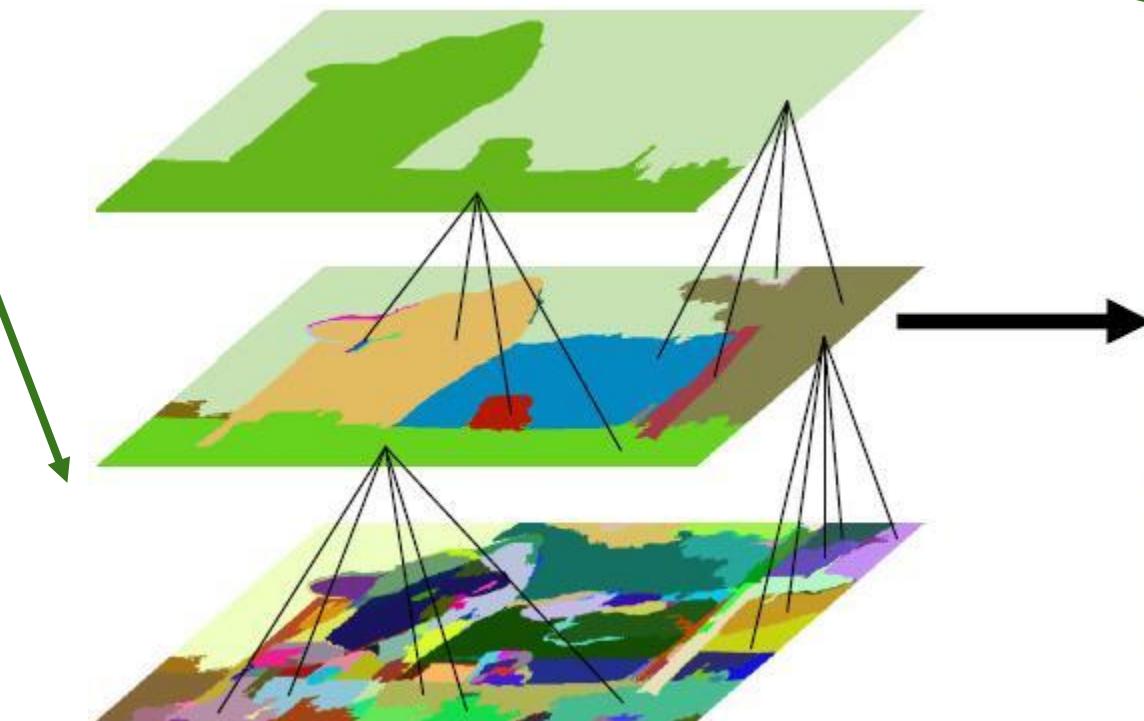


Ref:

Selective Search for Object Recognition

J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders
In International Journal of Computer Vision 2013.

Selective Search at all scales and return region proposals



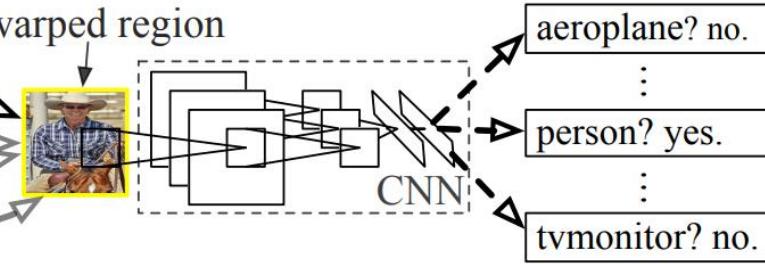
R-CNN: *Regions with CNN features*



1. Input image



2. Extract region proposals (~2k)



3. Compute CNN features

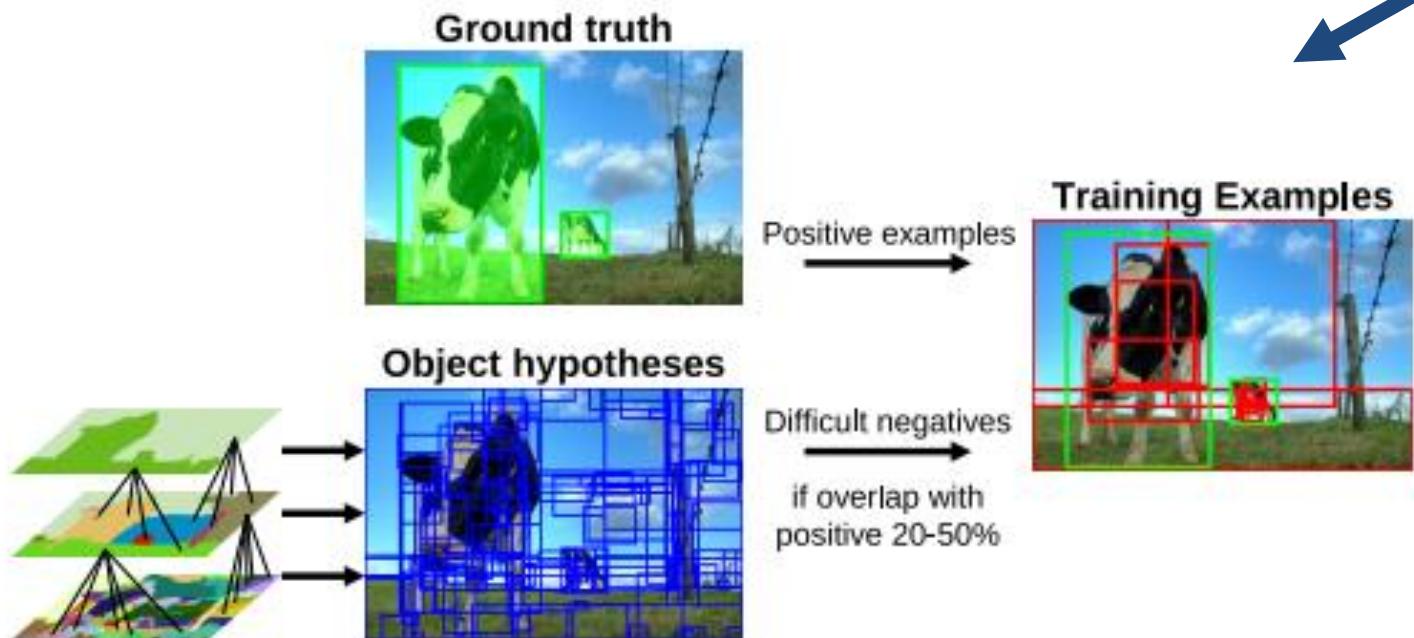
4. Classify regions

Ref:

Selective Search for Object Recognition

J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders
In International Journal of Computer Vision 2013.

Optional



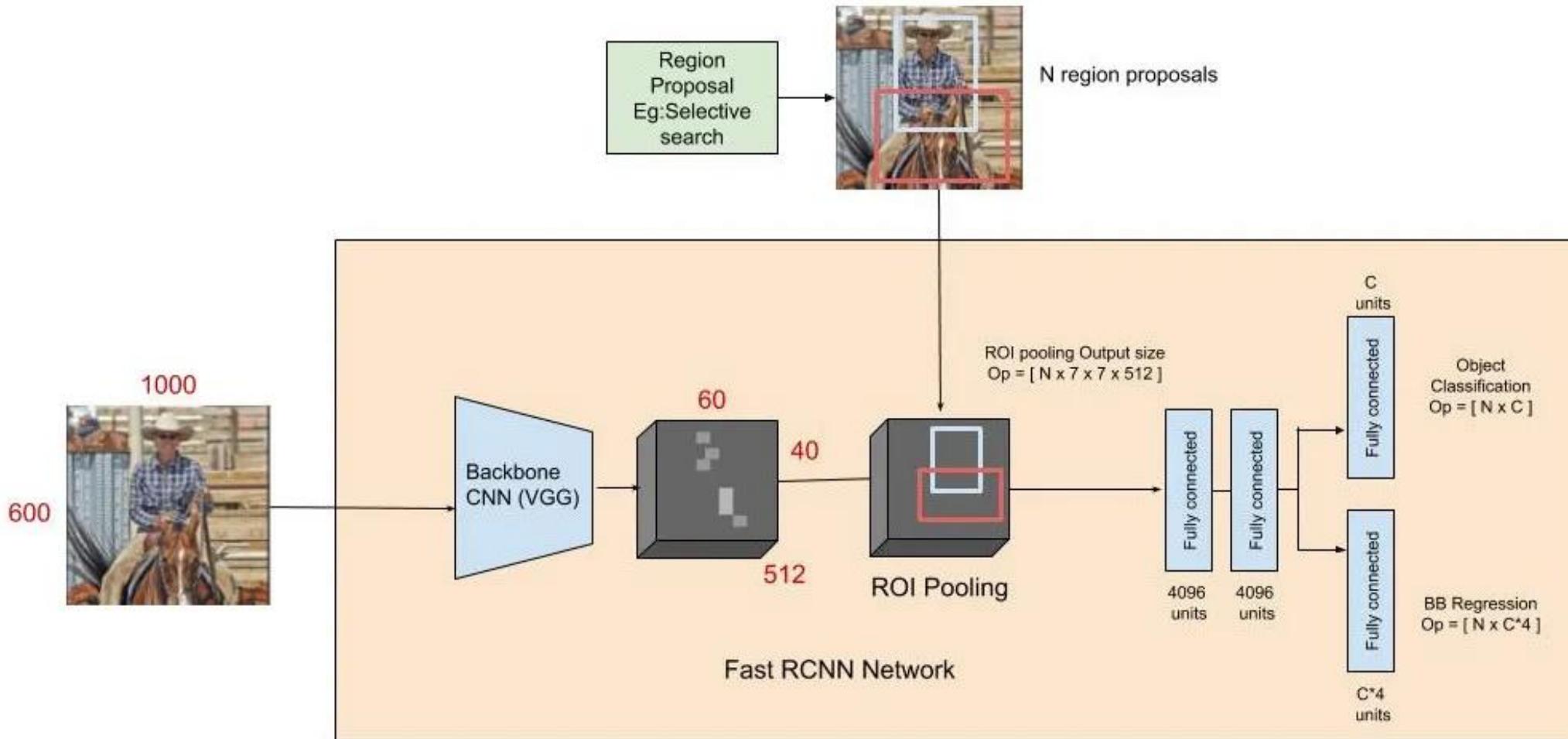
The training procedure of object recognition pipeline.

As positive learning examples we use the ground truth. As negatives we use examples that have a 20-50% overlap with the positive examples. We iteratively add hard negatives using a retraining phase.

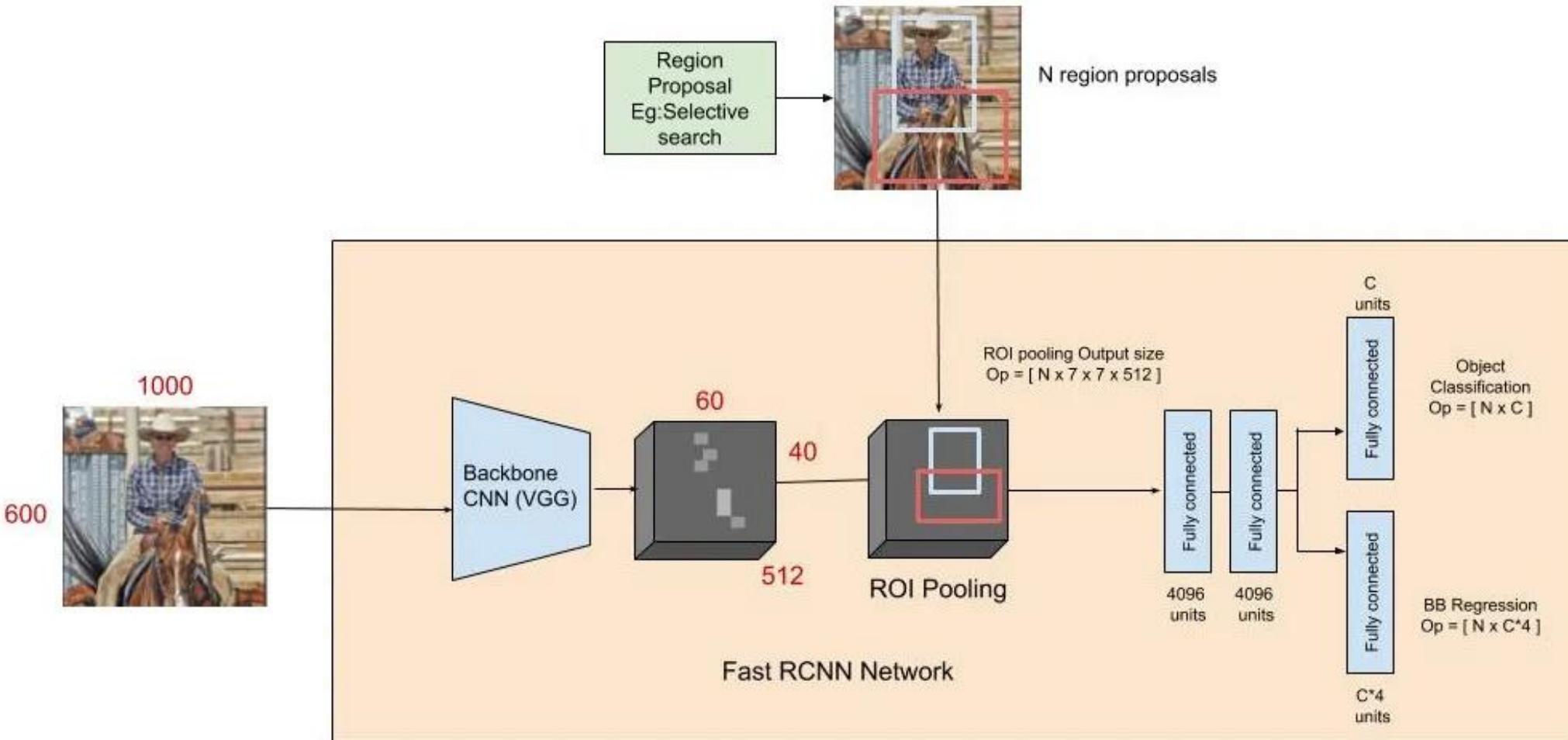
R-CNN pros and cons

- **Pros**
 - Accurate!
 - Any deep architecture can immediately be “plugged in”
- **Cons**
 - Ad hoc training objectives
 - Fine-tune network with softmax classifier (log loss)
 - Train post-hoc linear SVMs (hinge loss)
 - Train post-hoc bounding-box regressions (least squares)
 - Training is slow (84h), takes a lot of disk space
 - 2000 CNN passes per image
 - Inference (detection) is slow (47s / image with VGG16)

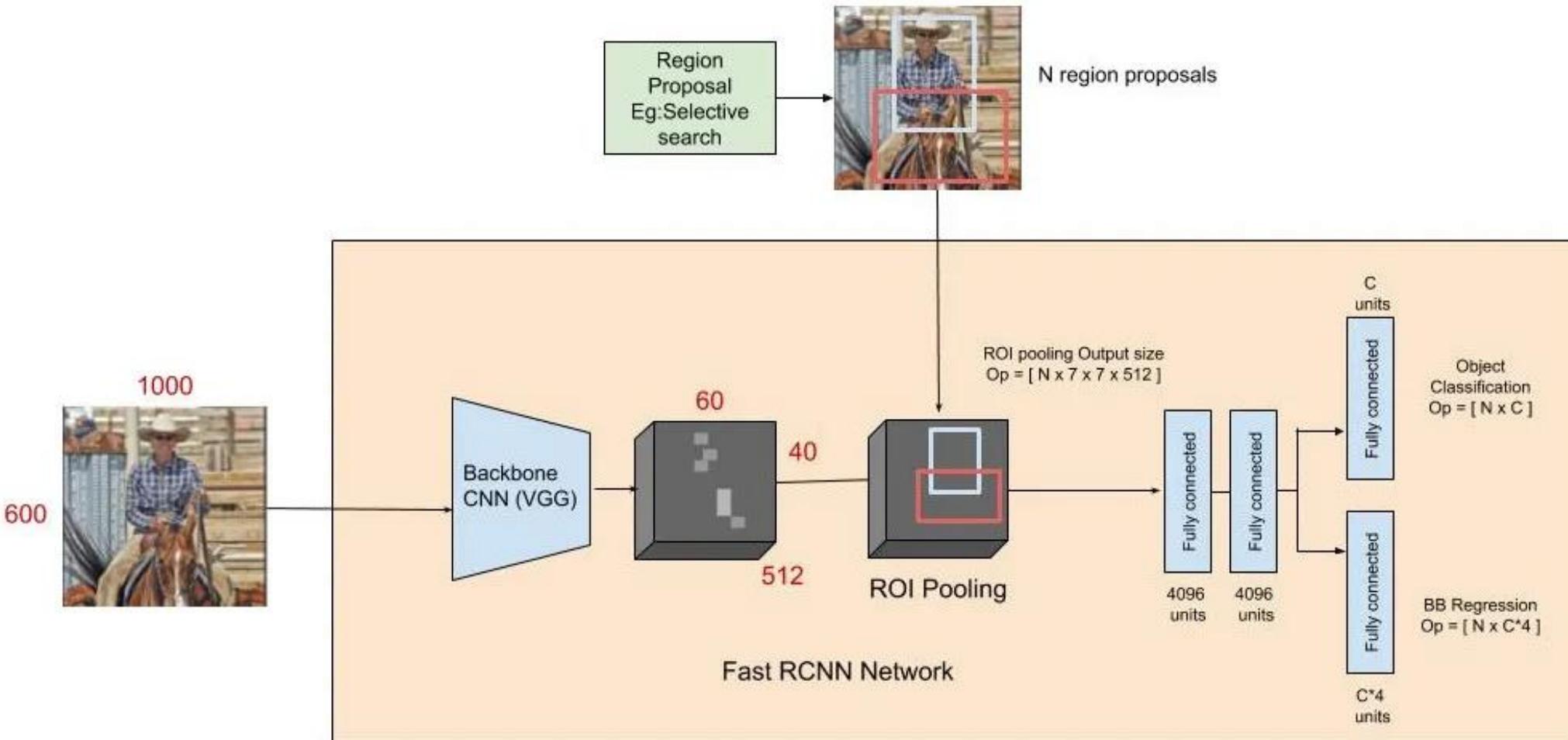
Fast R-CNN



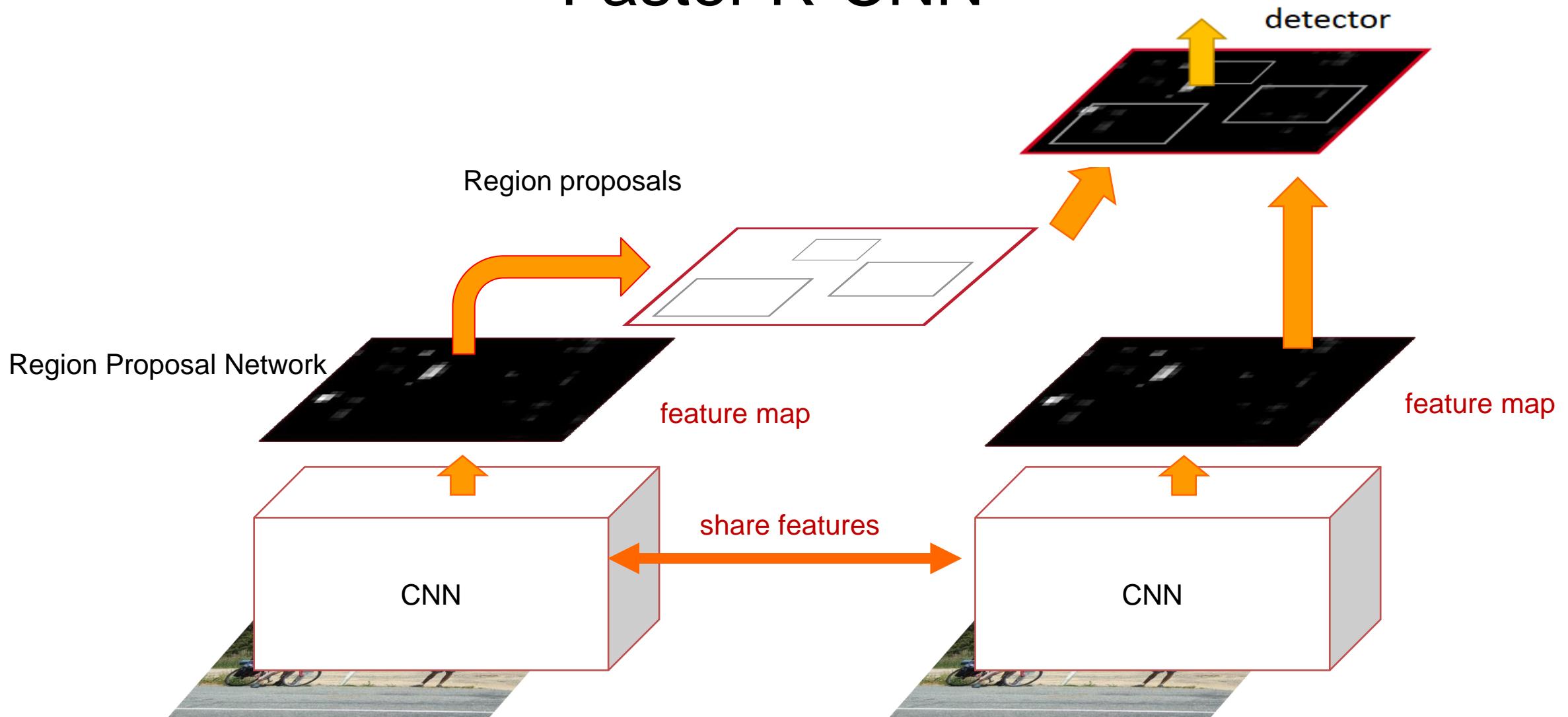
Fast R-CNN



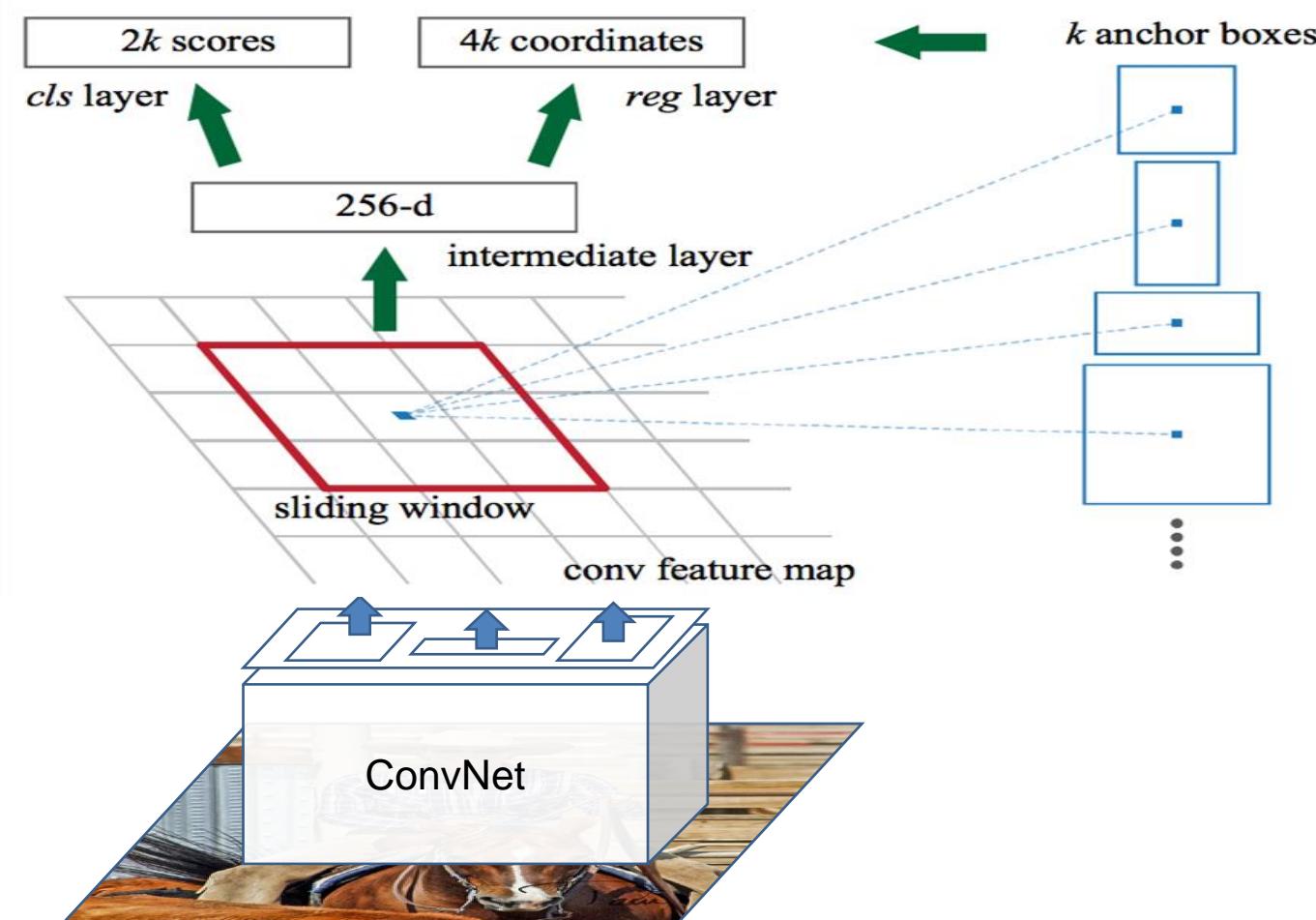
Fast R-CNN



Faster R-CNN



Region Proposal Network (RPN)



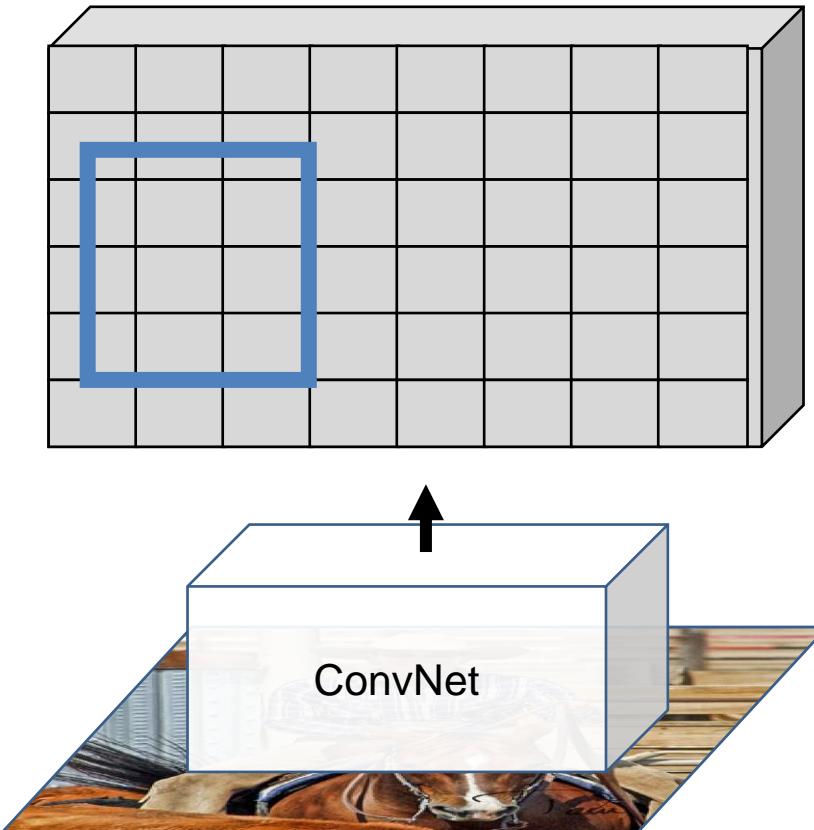
Small network applied to conv5 feature map.

Predicts:

- good box or not (classification),
- how to modify box (regression) for k “anchors” or boxes relative to the position in feature map.

ROI Pooling/Align

Feature Map
(e.g., 6x8x256)

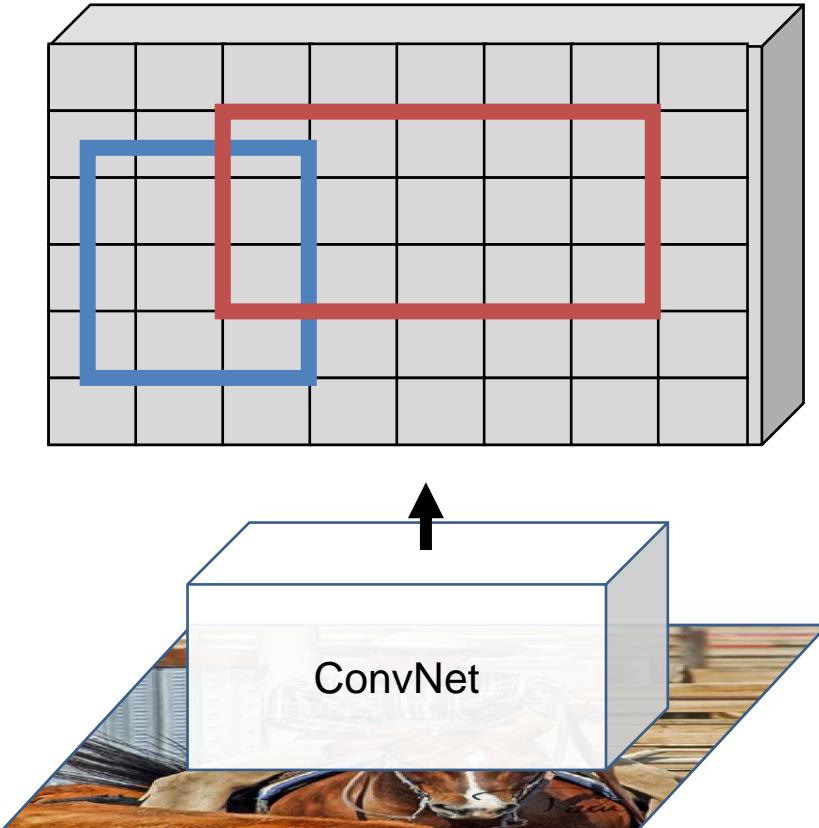


Given box in original image, calculate where the box goes.

- Example: $H=600, W=800$
- Feature map is $H'=6, W'=8$
- Box: left $x=50$, top $y=150$, width=250, height=350
- Feature map box: left $x=0.5$, $y = 1.5$, width=2.5, height=3.5

ROI Pooling/Align

Feature Map
(e.g., 6x8x256)

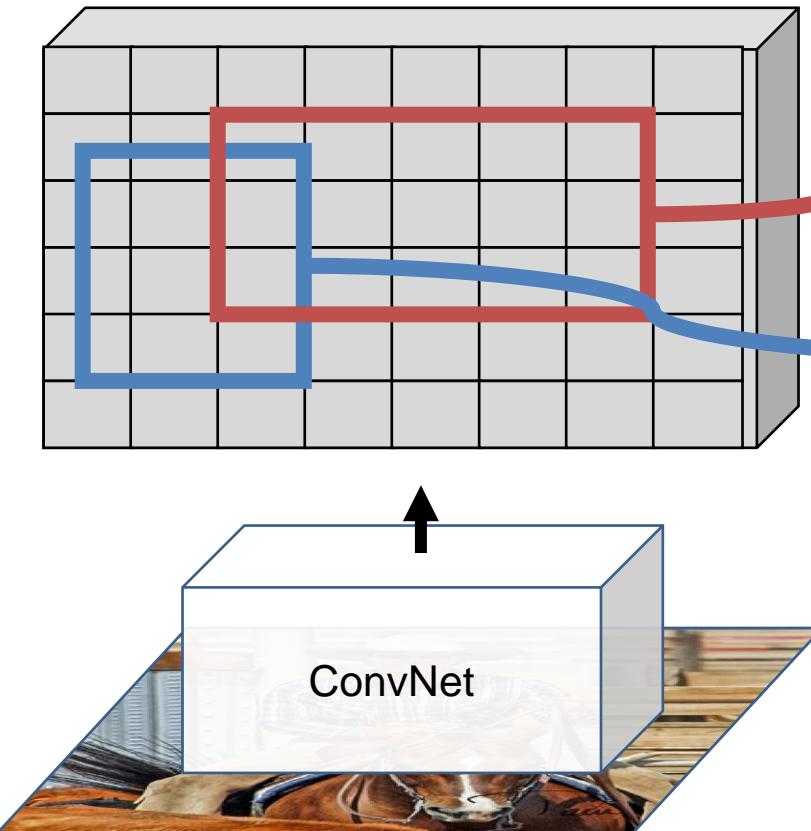


Given box in original image, calculate where the box goes.

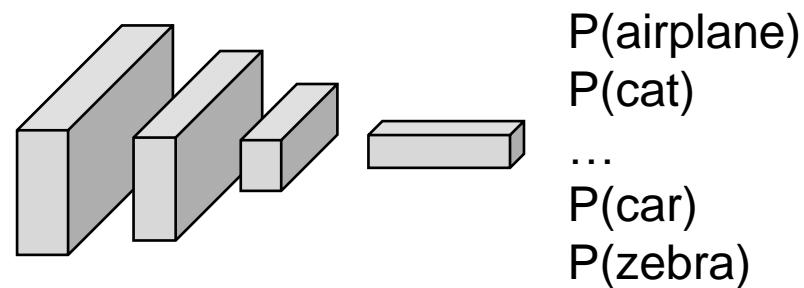
- Example: $H=600, W=800$
- Feature map is $H'=6, W'=8$
- Box: left $x=50$, top $y=150$, width=250, height=350
- Feature map box: left $x=0.5$, $y = 1.5$, width=2.5, height=3.5
- Other feature map box: left $x=2$, top $y = 1$, width=5, height=3

ROI Pooling/Align

Feature Map
(e.g., 6x8x256)



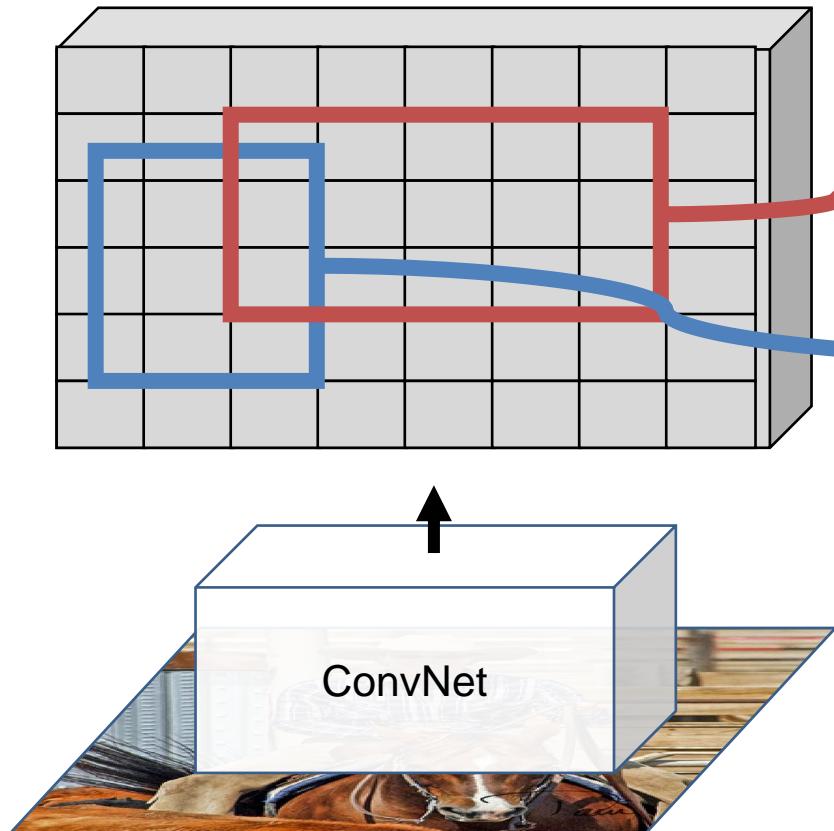
Resize to fixed size (e.g., 7x7)
Details critical, but beyond scope of class.



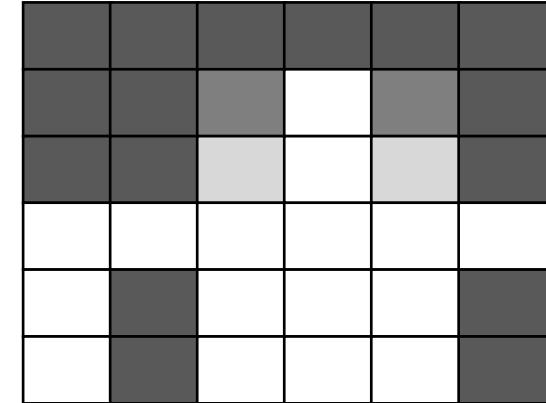
Afterwards, can add a small neural network that classifies the box and is applied to each window.

Mask RCNN

Feature Map
(e.g., 6x8x256)



Resize to fixed size (e.g., 7x7)
Details critical, but beyond scope of class.



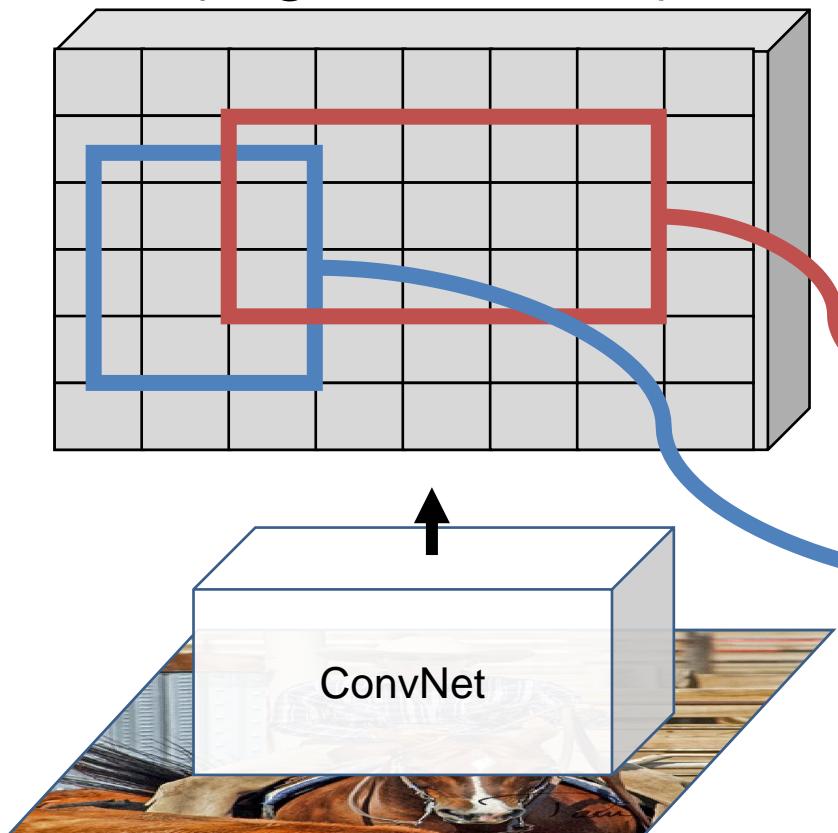
Can also predict a mask!
Everything is learned together
Simple and effective; details critical

MaskRCNN – Results



Extending Object Detection

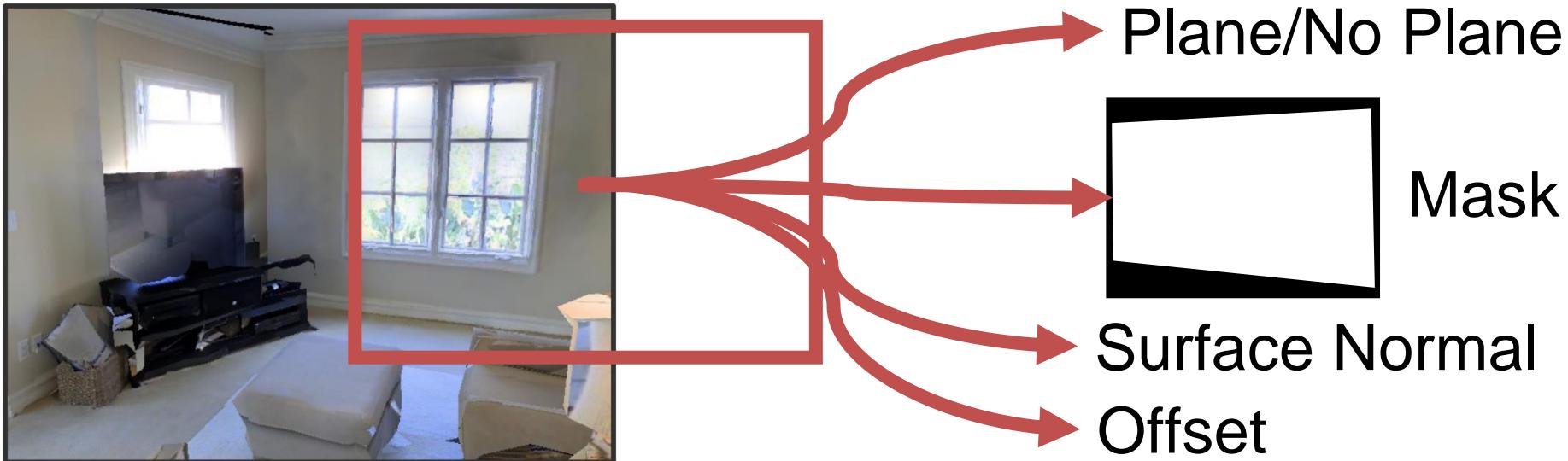
Feature Map
(e.g., 6x8x256)



- Can ask the network to predict *nearly* anything.
- If you can associate it with a bounding box and can get data for it, you can train the model

Extending Object Detection

Example: RGB image input, detect planar surfaces



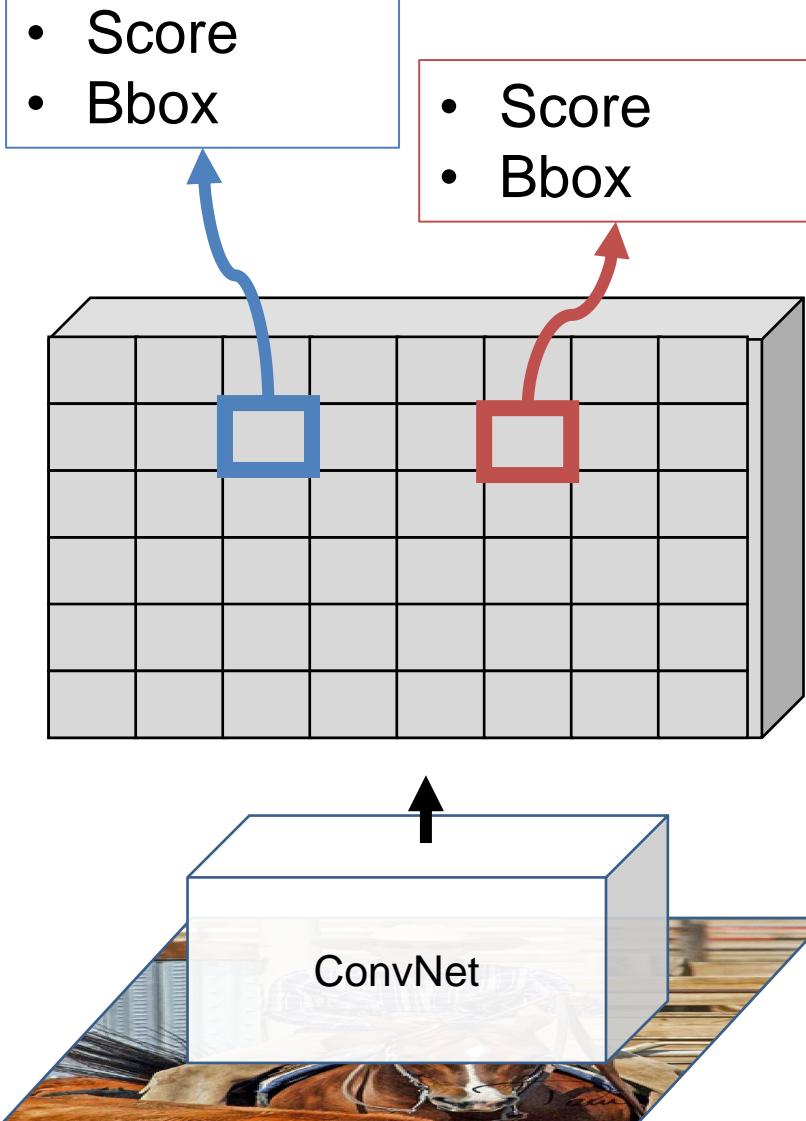
Extending Object Detection



Core building block is
detecting plane in
image.



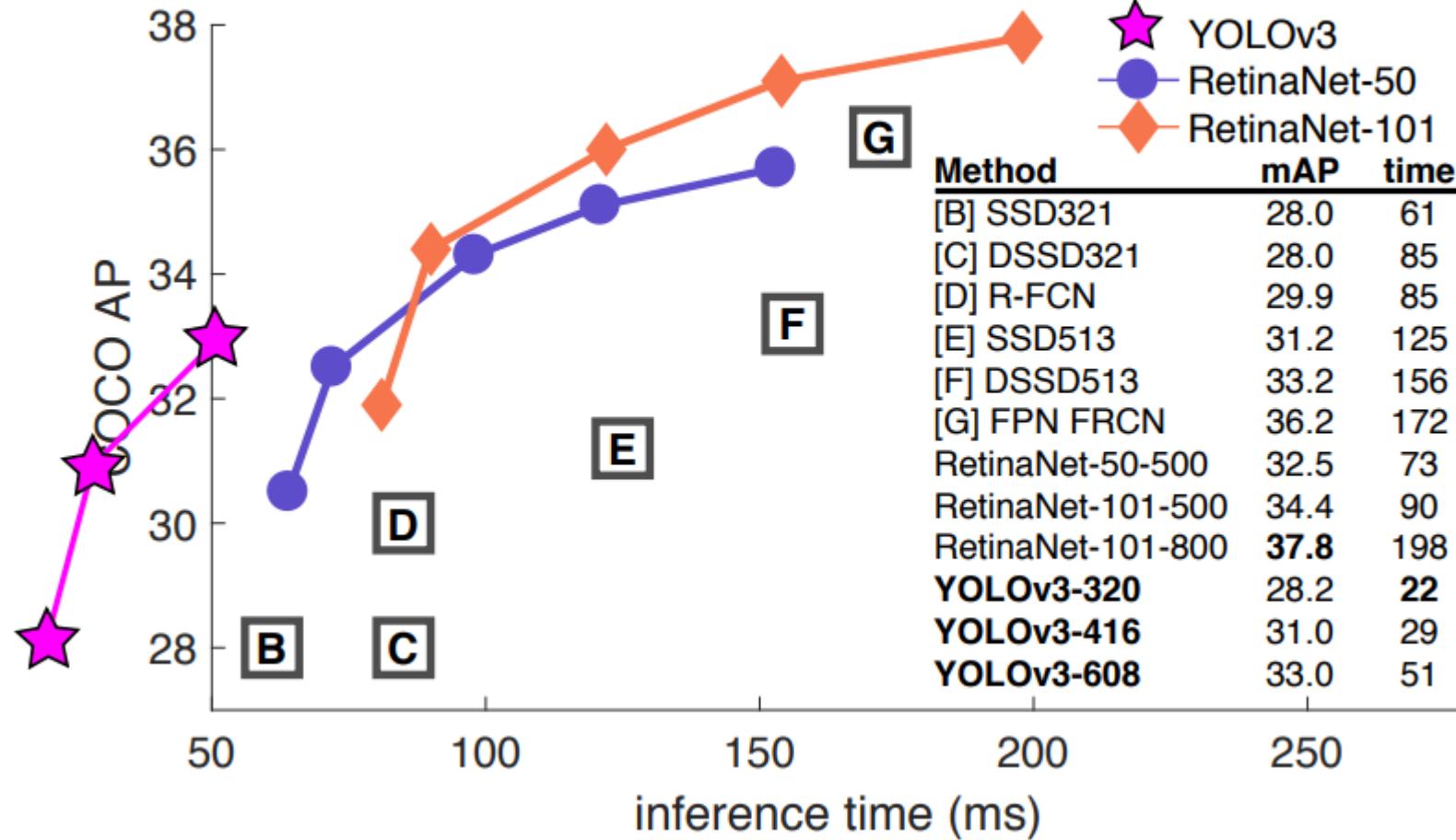
YOLO



- No proposals
- Predict at each location in 7×7 feature map, score for each class + 2 bboxes
- 7x faster than Faster-RCNN, but worse accuracy, precision
- Immensely popular in robotics
- Loads of similar methods (YOLOv2, YOLOv3)

YOLO

- Score



n in 7x7 feature
ass + 2 bboxes
RCNN, but worse

robotics
ods (YOLOv2,

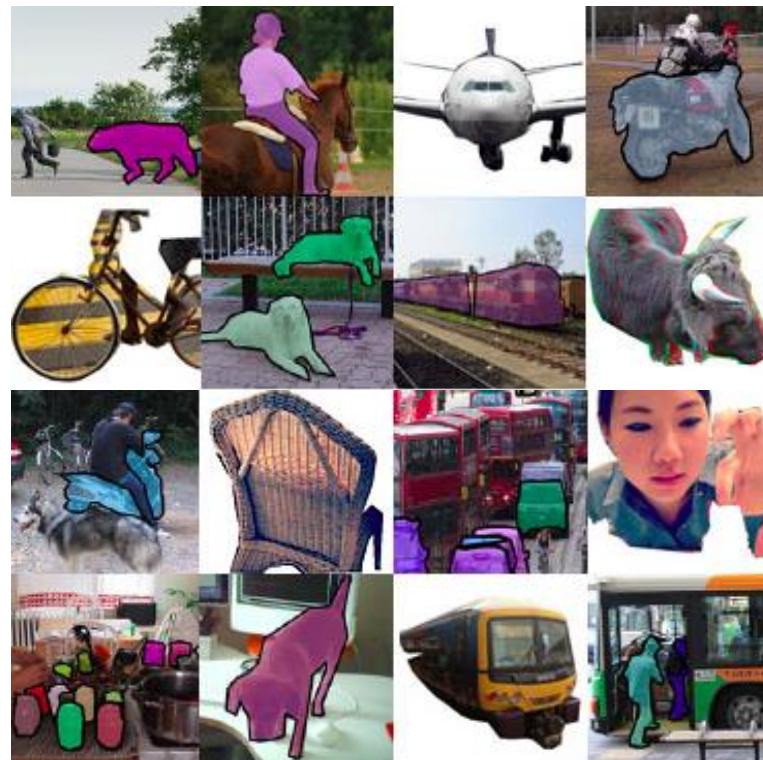
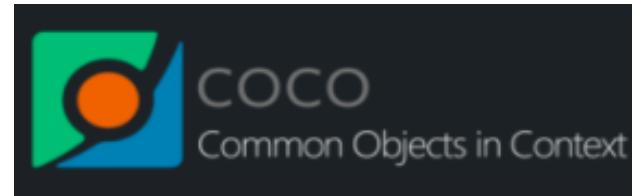
New detection benchmark: COCO (2014)

- 80 categories instead of PASCAL's 20
- Current best mAP: 52%



COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- ✓ Object segmentation
- ✓ Recognition in context
- ✓ Superpixel stuff segmentation
- ✓ 330K Images (>200K labeled)
- ✓ 1.5 million object instances
- ✓ 80 object categories
- ✓ 91 stuff categories
- ✓ 5 captions per image
- ✓ 250,000 people with keypoints



A Few Caveats

- Flickr images come from a really weird process
- Step 1: user takes a picture
- Step 2: user decides to upload it
- Step 3: user decides to write something like “refrigerator” somewhere in the description
- Step 4: a vision person stumbles on it while searching Flickr for refrigerators for a dataset



Who takes photos of open refrigerators ?????

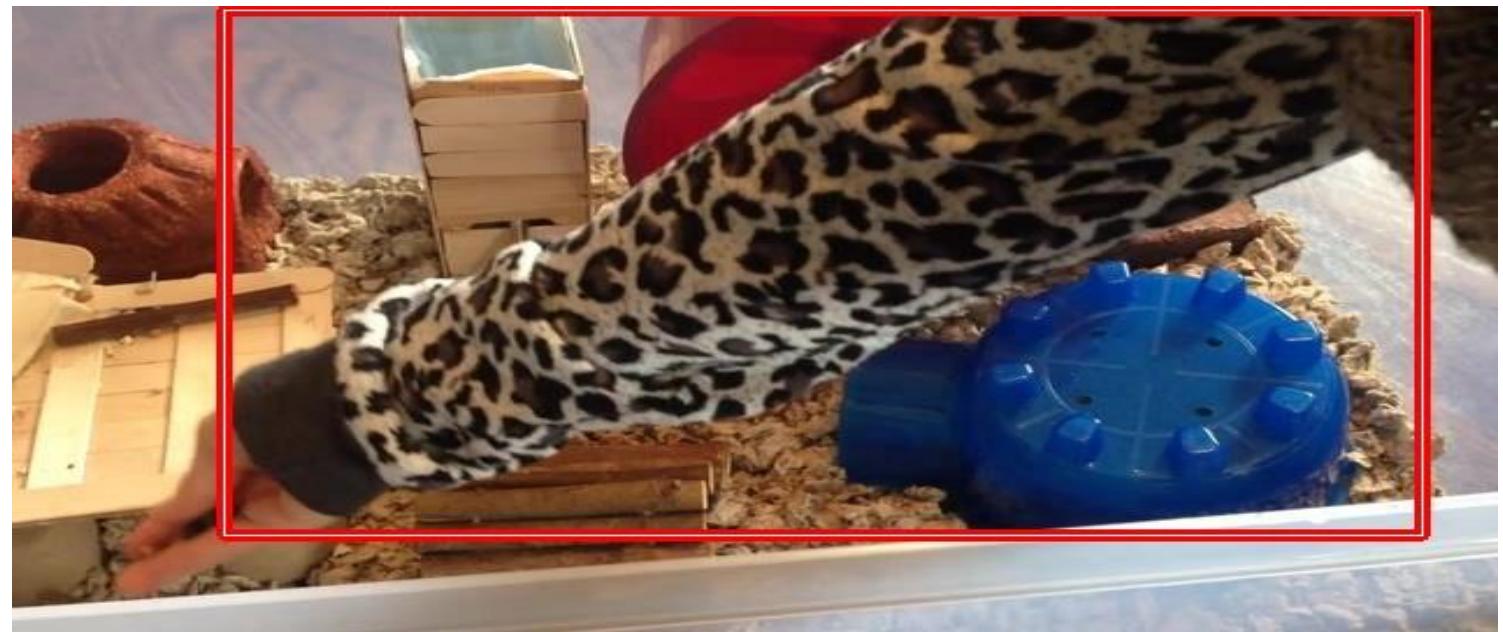


Guess the category!

These were detected with >90% confidence, corresponding to 99% precision on original dataset



(1) Person

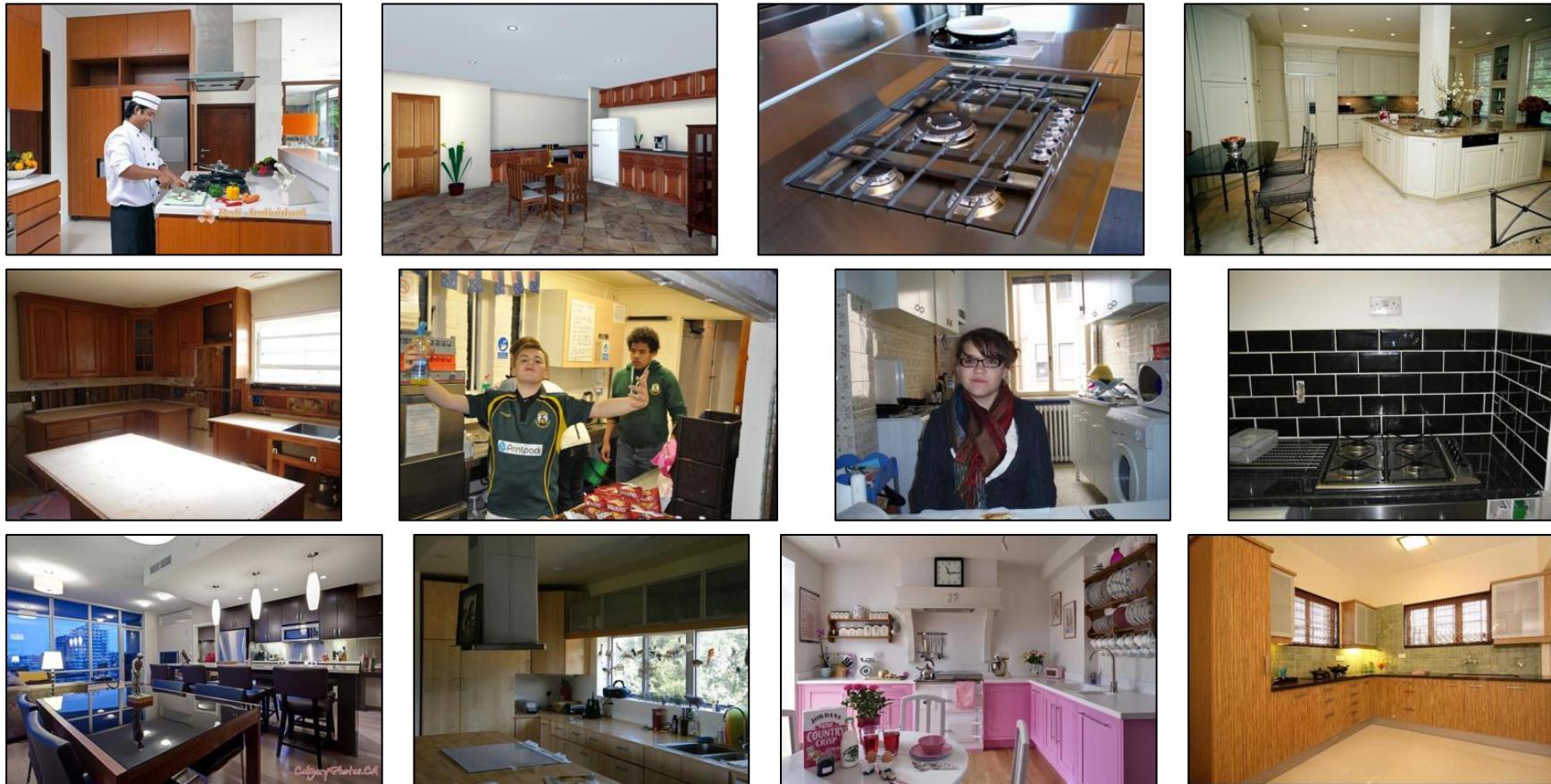


(2) Giraffe



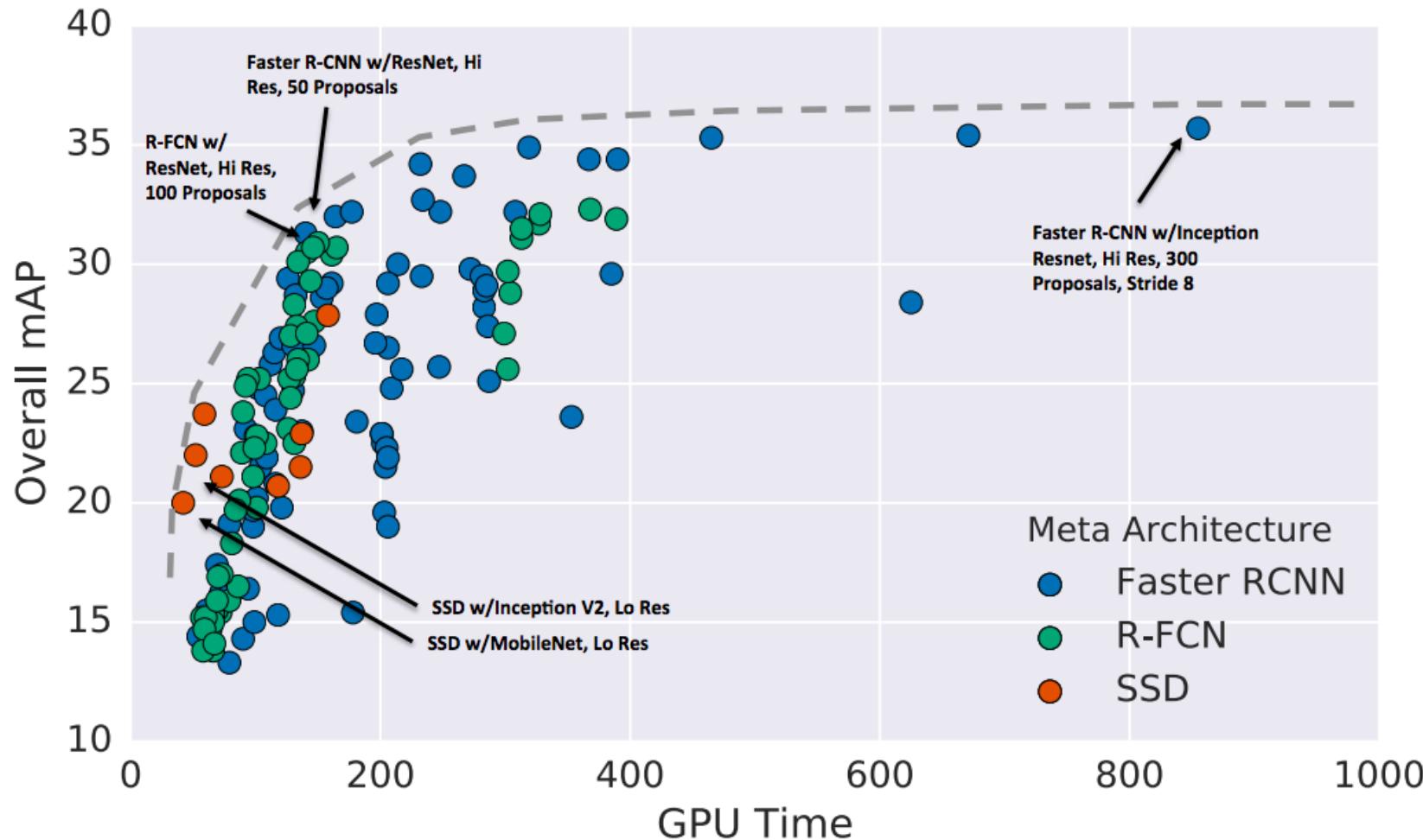
(3) Bicycle

Kitchens from Googling



Places 365 Dataset, Zhou et al. '17

New detection benchmark: COCO (2014)

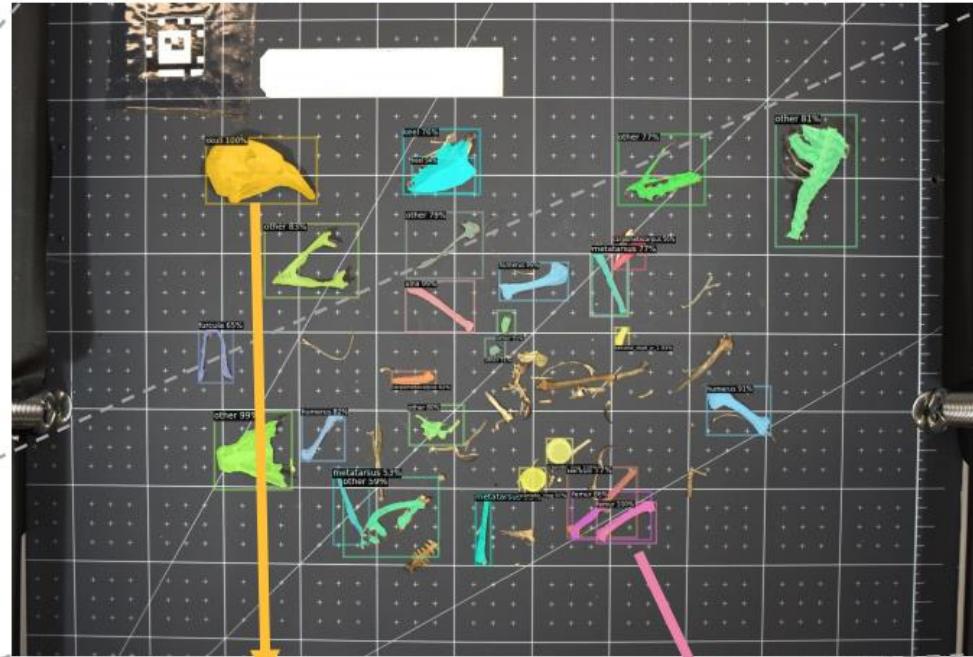
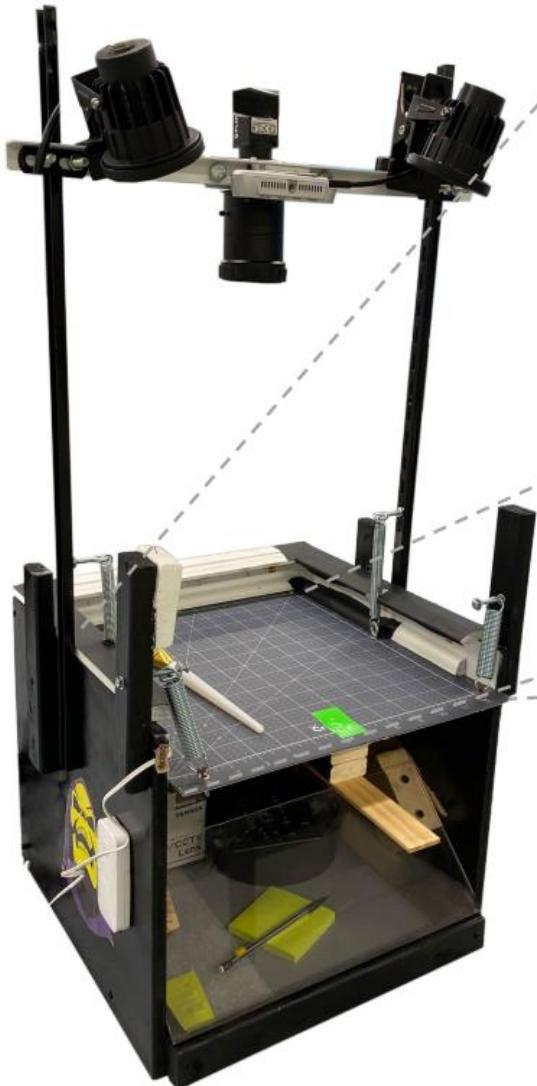


J. Huang et al., Speed/accuracy trade-offs for modern convolutional object detectors,
CVPR 2017

What's It Good For?

Instance Segmentation

Imaging Setup



Skull



36.08 mm

Femur



18.97 mm

Z. Zhou, G. Hassena, B.C. Weeks, D.F. Fouhey. *Quantifying Bird Skeletons*. CVPR Workshops, 2021.

B.C. Weeks, Z. Zhou, B.K. O'Brien, R. Darling, M. Dean, T. Dias, G. Hassena, M. Zhang, D.F. Fouhey. *A deep neural network for high throughput measurement of functional traits on museum skeletal specimens*. Methods in Ecology and Evolution, 2022.

What's It Good For?

Carpometacarpus



Humerus



Skull



- UM has 25K bird skeletons but measuring bird bones by hand is hard and tedious.
- New solution: dump the bones, take a picture, use a deep network.
- Now enabling testing hypotheses about birds at huge scale



- (1) Computer Vision: Algorithms and Applications, 2nd ed (Text Book-1) , Richard Szeliski ; Section 6.3
- (2) [Object Detection with Deep Learning: A Review \[IEEE Transactions on NN & LS\)](#)
- (3) [Chapter -10 – Object Detection, Yolo and their Implementation](#)



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you

Object Detection with Deep Learning: A Review

Zhong-Qiu Zhao, *Member, IEEE*, Peng Zheng,
Shou-tao Xu, and Xindong Wu, *Fellow, IEEE*

Abstract—Due to object detection’s close relationship with video analysis and image understanding, it has attracted much research attention in recent years. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance easily stagnates by constructing complex ensembles which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures. These models behave differently in network architecture, training strategy and optimization function, etc. In this paper, we provide a review on deep learning based object detection frameworks. Our review begins with a brief introduction on the history of deep learning and its representative tool, namely Convolutional Neural Network (CNN). Then we focus on typical generic object detection architectures along with some modifications and useful tricks to improve detection performance further. As distinct specific detection tasks exhibit different characteristics, we also briefly survey several specific tasks, including salient object detection, face detection and pedestrian detection. Experimental analyses are also provided to compare various methods and draw some meaningful conclusions. Finally, several promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network based learning systems.

Index Terms—deep learning, object detection, neural network

I. INTRODUCTION

To gain a complete image understanding, we should not only concentrate on classifying different images, but also try to precisely estimate the concepts and locations of objects contained in each image. This task is referred as object detection [1][S1], which usually consists of different subtasks such as face detection [2][S2], pedestrian detection [3][S2] and skeleton detection [4][S3]. As one of the fundamental computer vision problems, object detection is able to provide valuable information for semantic understanding of images and videos, and is related to many applications, including image classification [5], [6], human behavior analysis [7][S4], face recognition [8][S5] and autonomous driving [9], [10]. Meanwhile, Inheriting from neural networks and related learning systems, the progress in these fields will develop neural network algorithms, and will also have great impacts on object detection techniques which can be considered as learning systems. [11]–[14][S6]. However, due to large variations in viewpoints, poses, occlusions and lighting conditions, it’s difficult to perfectly accomplish object detection with an additional

Zhong-Qiu Zhao, Peng Zheng and Shou-Tao Xu are with the College of Computer Science and Information Engineering, Hefei University of Technology, China. Xindong Wu is with the School of Computing and Informatics, University of Louisiana at Lafayette, USA.

Manuscript received August xx, 2017; revised xx xx, 2017.

object localization task. So much attention has been attracted to this field in recent years [15]–[18].

The problem definition of object detection is to determine where objects are located in a given image (object localization) and which category each object belongs to (object classification). So the pipeline of traditional object detection models can be mainly divided into three stages: informative region selection, feature extraction and classification.

Informative region selection. As different objects may appear in any positions of the image and have different aspect ratios or sizes, it is a natural choice to scan the whole image with a multi-scale sliding window. Although this exhaustive strategy can find out all possible positions of the objects, its shortcomings are also obvious. Due to a large number of candidate windows, it is computationally expensive and produces too many redundant windows. However, if only a fixed number of sliding window templates are applied, unsatisfactory regions may be produced.

Feature extraction. To recognize different objects, we need to extract visual features which can provide a semantic and robust representation. SIFT [19], HOG [20] and Haar-like [21] features are the representative ones. This is due to the fact that these features can produce representations associated with complex cells in human brain [19]. However, due to the diversity of appearances, illumination conditions and backgrounds, it’s difficult to manually design a robust feature descriptor to perfectly describe all kinds of objects.

Classification. Besides, a classifier is needed to distinguish a target object from all the other categories and to make the representations more hierarchical, semantic and informative for visual recognition. Usually, the Supported Vector Machine (SVM) [22], AdaBoost [23] and Deformable Part-based Model (DPM) [24] are good choices. Among these classifiers, the DPM is a flexible model by combining object parts with deformation cost to handle severe deformations. In DPM, with the aid of a graphical model, carefully designed low-level features and kinematically inspired part decompositions are combined. And discriminative learning of graphical models allows for building high-precision part-based models for a variety of object classes.

Based on these discriminant local feature descriptors and shallow learnable architectures, state of the art results have been obtained on PASCAL VOC object detection competition [25] and real-time embedded systems have been obtained with a low burden on hardware. However, small gains are obtained during 2010-2012 by only building ensemble systems and employing minor variants of successful methods [15]. This fact is due to the following reasons: 1) The generation of candidate bounding boxes with a sliding window strategy is redundant, inefficient and inaccurate. 2) The semantic gap cannot be

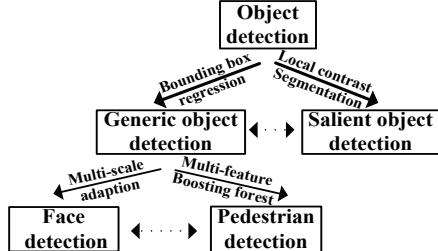


Fig. 1. The application domains of object detection.

bridged by the combination of manually engineered low-level descriptors and discriminatively-trained shallow models.

Thanks to the emergence of Deep Neural Networks (DNNs) [6][S7], a more significant gain is obtained with the introduction of Regions with CNN features (R-CNN) [15]. DNNs, or the most representative CNNs, act in a quite different way from traditional approaches. They have deeper architectures with the capacity to learn more complex features than the shallow ones. Also the expressivity and robust training algorithms allow to learn informative object representations without the need to design features manually [26].

Since the proposal of R-CNN, a great deal of improved models have been suggested, including Fast R-CNN which jointly optimizes classification and bounding box regression tasks [16], Faster R-CNN which takes an additional sub-network to generate region proposals [18] and YOLO which accomplishes object detection via a fixed-grid regression [17]. All of them bring different degrees of detection performance improvements over the primary R-CNN and make real-time and accurate object detection become more achievable.

In this paper, a systematic review is provided to summarise representative models and their different characteristics in several application domains, **including** generic object detection [15], [16], [18], salient object detection [27], [28], face detection [29]–[31] and pedestrian detection [32], [33]. Their relationships are depicted in Figure 1. Based on basic CNN architectures, generic object detection is achieved with bounding box regression, while salient object detection is accomplished with local contrast enhancement and pixel-level segmentation. Face detection and pedestrian detection are closely related to generic object detection and mainly accomplished with multi-scale adaption and multi-feature fusion/boosting forest, respectively. The dotted lines indicate that the corresponding domains are associated with each other under certain conditions. It should be noticed that the covered domains are diversified. Pedestrian and face images have regular structures, while general objects and scene images have more complex variations in geometric structures and layouts. Therefore, different deep models are required by various images.

There has been a relevant pioneer effort [34] which mainly focuses on relevant software tools to implement deep learning techniques for image classification and object detection, but pays little attention on detailing specific algorithms. Different from it, our work not only reviews deep learning based object detection models and algorithms covering different application domains in detail, but also provides their corresponding experimental comparisons and meaningful analyses.

The rest of this paper is organized as follows. In Section

2, a brief introduction on the history of deep learning and the basic architecture of CNN is provided. Generic object detection architectures are presented in Section 3. Then reviews of CNN applied in several specific tasks, including salient object detection, face detection and pedestrian detection, are exhibited in Section 4-6, respectively. Several promising future directions are proposed in Section 7. At last, some concluding remarks are presented in Section 8.

II. A BRIEF OVERVIEW OF DEEP LEARNING

Prior to overview on deep learning based object detection approaches, we provide a review on the history of deep learning along with an introduction on the basic architecture and advantages of CNN.

A. The History: Birth, Decline and Prosperity

Deep models can be referred to as neural networks with deep structures. The history of neural networks can date back to 1940s [35], and the original intention was to simulate the human brain system to solve general learning problems in a principled way. It was popular in 1980s and 1990s with the proposal of back-propagation algorithm by Hinton et al. [36]. However, due to the overfitting of training, lack of large scale training data, limited computation power and insignificance in performance compared with other machine learning tools, neural networks fell out of fashion in early 2000s.

Deep learning has become popular since 2006 [37][S7] with a break through in speech recognition [38]. The recovery of deep learning can be attributed to the following factors.

- The emergence of large scale annotated training data, such as ImageNet [39], to fully exhibit its very large learning capacity;
- Fast development of high performance parallel computing systems, such as GPU clusters;
- Significant advances in the design of network structures and training strategies. With unsupervised and layerwise pre-training guided by Auto-Encoder (AE) [40] or Restricted Boltzmann Machine (RBM) [41], a good initialization is provided. With dropout and data augmentation, the overfitting problem in training has been relieved [6], [42]. With batch normalization (BN), the training of very deep neural networks becomes quite efficient [43]. Meanwhile, various network structures, such as AlexNet [6], Overfeat [44], GoogLeNet [45], VGG [46] and ResNet [47], have been extensively studied to improve the performance.

What prompts deep learning to have a huge impact on the entire academic community? It may owe to the contribution of Hinton's group, whose continuous efforts have demonstrated that deep learning would bring a revolutionary breakthrough on grand challenges rather than just obvious improvements on small datasets. Their success results from training a large CNN on 1.2 million labeled images together with a few techniques [6] (e.g., ReLU operation [48] and ‘dropout’ regularization).

B. Architecture and Advantages of CNN

CNN is the most representative model of deep learning [26]. A typical CNN architecture, which is referred to as VGG16,

can be found in Fig. S1. Each layer of CNN is known as a feature map. The feature map of the input layer is a 3D matrix of pixel intensities for different color channels (e.g. RGB). The feature map of any internal layer is an induced multi-channel image, whose ‘pixel’ can be viewed as a specific feature. Every neuron is connected with a small portion of adjacent neurons from the previous layer (receptive field). Different types of transformations [6], [49], [50] can be conducted on feature maps, such as filtering and pooling. Filtering (convolution) operation convolves a filter matrix (learned weights) with the values of a receptive field of neurons and takes a non-linear function (such as sigmoid [51], ReLU) to obtain final responses. Pooling operation, such as max pooling, average pooling, L2-pooling and local contrast normalization [52], summarizes the responses of a receptive field into one value to produce more robust feature descriptions.

With an interleave between convolution and pooling, an initial feature hierarchy is constructed, which can be fine-tuned in a supervised manner by adding several fully connected (FC) layers to adapt to different visual tasks. According to the tasks involved, the final layer with different activation functions [6] is added to get a specific conditional probability for each output neuron. And the whole network can be optimized on an objective function (e.g. mean squared error or cross-entropy loss) via the stochastic gradient descent (SGD) method. The typical VGG16 has totally 13 convolutional (conv) layers, 3 fully connected layers, 3 max-pooling layers and a softmax classification layer. The conv feature maps are produced by convoluting 3×3 filter windows, and feature map resolutions are reduced with 2 stride max-pooling layers. An arbitrary test image of the same size as training samples can be processed with the trained network. Re-scaling or cropping operations may be needed if different sizes are provided [6].

The advantages of CNN against traditional methods can be summarised as follows.

- Hierarchical feature representation, which is the multi-level representations from pixel to high-level semantic features learned by a hierarchical multi-stage structure [15], [53], can be learned from data automatically and hidden factors of input data can be disentangled through multi-level nonlinear mappings.
- Compared with traditional shallow models, a deeper architecture provides an exponentially increased expressive capability.
- The architecture of CNN provides an opportunity to jointly optimize several related tasks together (e.g. Fast R-CNN combines classification and bounding box regression into a multi-task leaning manner).
- Benefiting from the large learning capacity of deep CNNs, some classical computer vision challenges can be recast as high-dimensional data transform problems and solved from a different viewpoint.

Due to these advantages, CNN has been widely applied into many research fields, such as image super-resolution reconstruction [54], [55], image classification [5], [56], image retrieval [57], [58], face recognition [8][S5], pedestrian detection [59]–[61] and video analysis [62], [63].

III. GENERIC OBJECT DETECTION

Generic object detection aims at locating and classifying existing objects in any one image, and labeling them with rectangular bounding boxes to show the confidences of existence. The frameworks of generic object detection methods can mainly be categorized into two types (see Figure 2). One follows traditional object detection pipeline, generating region proposals at first and then classifying each proposal into different object categories. The other regards object detection as a regression or classification problem, adopting a unified framework to achieve final results (categories and locations) directly. The region proposal based methods mainly include R-CNN [15], SPP-net [64], Fast R-CNN [16], Faster R-CNN [18], R-FCN [65], FPN [66] and Mask R-CNN [67], some of which are correlated with each other (e.g. SPP-net modifies R-CNN with a SPP layer). The regression/classification based methods mainly includes MultiBox [68], AttentionNet [69], G-CNN [70], YOLO [17], SSD [71], YOLOv2 [72], DSSD [73] and DSOD [74]. The correlations between these two pipelines are bridged by the anchors introduced in Faster R-CNN. Details of these methods are as follows.

A. Region Proposal Based Framework

The region proposal based framework, a two-step process, matches the attentional mechanism of human brain to some extent, which gives a coarse scan of the whole scenario firstly and then focuses on regions of interest. Among the pre-related works [44], [75], [76], the most representative one is Overfeat [44]. This model inserts CNN into sliding window method, which predicts bounding boxes directly from locations of the topmost feature map after obtaining the confidences of underlying object categories.

1) R-CNN: It is of significance to improve the quality of candidate bounding boxes and to take a deep architecture to extract high-level features. To solve these problems, R-CNN [15] was proposed by Ross Girshick in 2014 and obtained a mean average precision (mAP) of 53.3% with more than 30% improvement over the previous best result (DPM HSC [77]) on PASCAL VOC 2012. Figure 3 shows the flowchart of R-CNN, which can be divided into three stages as follows.

Region proposal generation. The R-CNN adopts selective search [78] to generate about 2k region proposals for each image. The selective search method relies on simple bottom-up grouping and saliency cues to provide more accurate candidate boxes of arbitrary sizes quickly and to reduce the searching space in object detection [24], [39].

CNN based deep feature extraction. In this stage, each region proposal is warped or cropped into a fixed resolution and the CNN module in [6] is utilized to extract a 4096-dimensional feature as the final representation. Due to large learning capacity, dominant expressive power and hierarchical structure of CNNs, a high-level, semantic and robust feature representation for each region proposal can be obtained.

Classification and localization. With pre-trained category-specific linear SVMs for multiple classes, different region proposals are scored on a set of positive regions and background (negative) regions. The scored regions are then adjusted with

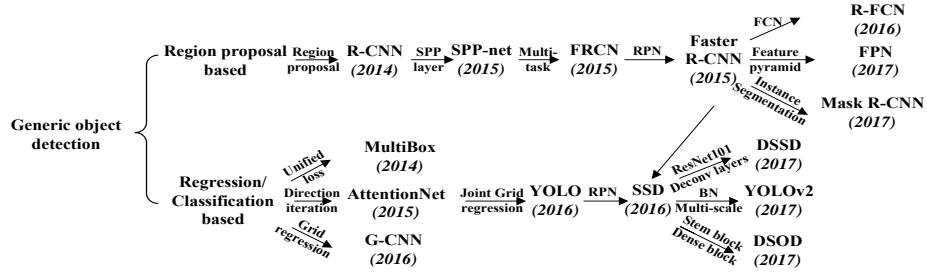


Fig. 2. Two types of frameworks: region proposal based and regression/classification based. SPP: Spatial Pyramid Pooling [64], FRCN: Faster R-CNN [16], RPN: Region Proposal Network [18], FCN: Fully Convolutional Network [65], BN: Batch Normalization [43], Deconv layers: Deconvolution layers [54]

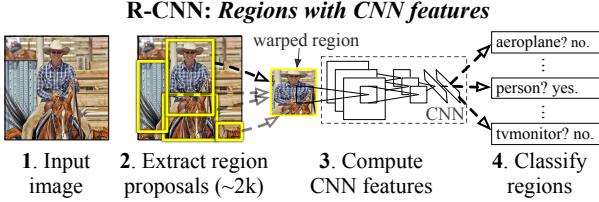


Fig. 3. The flowchart of R-CNN [15], which consists of 3 stages: (1) extracts bottom-up region proposals, (2) computes features for each proposal using a CNN, and then (3) classifies each region with class-specific linear SVMs.

bounding box regression and filtered with a greedy non-maximum suppression (NMS) to produce final bounding boxes for preserved object locations.

When there are scarce or insufficient labeled data, pre-training is usually conducted. Instead of unsupervised pre-training [79], R-CNN firstly conducts supervised pre-training on ILSVRC, a very large auxiliary dataset, and then takes a domain-specific fine-tuning. This scheme has been adopted by most of subsequent approaches [16], [18].

In spite of its improvements over traditional methods and significance in bringing CNN into practical object detection, there are still some disadvantages.

- Due to the existence of FC layers, the CNN requires a fixed-size (e.g., 227×227) input image, which directly leads to the re-computation of the whole CNN for each evaluated region, taking a great deal of time in the testing period.

- Training of R-CNN is a multi-stage pipeline. At first, a convolutional network (ConvNet) on object proposals is fine-tuned. Then the softmax classifier learned by fine-tuning is replaced by SVMs to fit in with ConvNet features. Finally, bounding-box regressors are trained.

- Training is expensive in space and time. Features are extracted from different region proposals and stored on the disk. It will take a long time to process a relatively small training set with very deep networks, such as VGG16. At the same time, the storage memory required by these features should also be a matter of concern.

- Although selective search can generate region proposals with relatively high recalls, the obtained region proposals are still redundant and this procedure is time-consuming (around 2 seconds to extract 2k region proposals).

To solve these problems, many methods have been proposed. GOP [80] takes a much faster geodesic based segmentation to replace traditional graph cuts. MCG [81] searches different scales of the image for multiple hierarchical segmentations and combinatorially groups different regions to produce

proposals. Instead of extracting visually distinct segments, the edge boxes method [82] adopts the idea that objects are more likely to exist in bounding boxes with fewer contours straggling their boundaries. Also some researches tried to re-rank or refine pre-extracted region proposals to remove unnecessary ones and obtained a limited number of valuable ones, such as DeepBox [83] and SharpMask [84].

In addition, there are some improvements to solve the problem of inaccurate localization. Zhang et al. [85] utilized a bayesian optimization based search algorithm to guide the regressions of different bounding boxes sequentially, and trained class-specific CNN classifiers with a structured loss to penalize the localization inaccuracy explicitly. Saurabh Gupta et al. improved object detection for RGB-D images with semantically rich image and depth features [86], and learned a new geocentric embedding for depth images to encode each pixel. The combination of object detectors and superpixel classification framework gains a promising result on semantic scene segmentation task. Ouyang et al. proposed a deformable deep CNN (DeepID-Net) [87] which introduces a novel deformation constrained pooling (def-pooling) layer to impose geometric penalty on the deformation of various object parts and makes an ensemble of models with different settings. Lenc et al. [88] provided an analysis on the role of proposal generation in CNN-based detectors and tried to replace this stage with a constant and trivial region generation scheme. The goal is achieved by biasing sampling to match the statistics of the ground truth bounding boxes with K-means clustering. However, more candidate boxes are required to achieve comparable results to those of R-CNN.

2) *SPP-net*: FC layers must take a fixed-size input. That's why R-CNN chooses to warp or crop each region proposal into the same size. However, the object may exist partly in the cropped region and unwanted geometric distortion may be produced due to the warping operation. These content losses or distortions will reduce recognition accuracy, especially when the scales of objects vary.

To solve this problem, He et al. took the theory of spatial pyramid matching (SPM) [89], [90] into consideration and proposed a novel CNN architecture named SPP-net [64]. SPM takes several finer to coarser scales to partition the image into a number of divisions and aggregates quantized local features into mid-level representations.

The architecture of SPP-net for object detection can be found in Figure 4. Different from R-CNN, SPP-net reuses feature maps of the 5-th conv layer (conv5) to project region

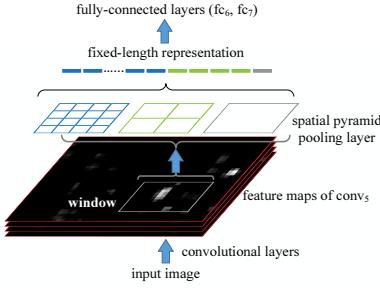


Fig. 4. The architecture of SPP-net for object detection [64].

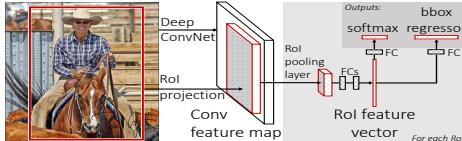


Fig. 5. The architecture of Fast R-CNN [16].

proposals of arbitrary sizes to fixed-length feature vectors. The feasibility of the reusability of these feature maps is due to the fact that the feature maps not only involve the strength of local responses, but also have relationships with their spatial positions [64]. The layer after the final conv layer is referred to as spatial pyramid pooling layer (SPP layer). If the number of feature maps in conv5 is 256, taking a 3-level pyramid, the final feature vector for each region proposal obtained after SPP layer has a dimension of $256 \times (1^2 + 2^2 + 4^2) = 5376$.

SPP-net not only gains better results with correct estimation of different region proposals in their corresponding scales, but also improves detection efficiency in testing period with the sharing of computation cost before SPP layer among different proposals.

3) *Fast R-CNN*: Although SPP-net has achieved impressive improvements in both accuracy and efficiency over R-CNN, it still has some notable drawbacks. SPP-net takes almost the same multi-stage pipeline as R-CNN, including feature extraction, network fine-tuning, SVM training and bounding-box regressor fitting. So an additional expense on storage space is still required. Additionally, the conv layers preceding the SPP layer cannot be updated with the fine-tuning algorithm introduced in [64]. As a result, an accuracy drop of very deep networks is unsurprising. To this end, Girshick [16] introduced a multi-task loss on classification and bounding box regression and proposed a novel CNN architecture named Fast R-CNN.

The architecture of Fast R-CNN is exhibited in Figure 5. Similar to SPP-net, the whole image is processed with conv layers to produce feature maps. Then, a fixed-length feature vector is extracted from each region proposal with a region of interest (RoI) pooling layer. The RoI pooling layer is a special case of the SPP layer, which has only one pyramid level. Each feature vector is then fed into a sequence of FC layers before finally branching into two sibling output layers. One output layer is responsible for producing softmax probabilities for all $C + 1$ categories (C object classes plus one ‘background’ class) and the other output layer encodes refined bounding-box positions with four real-valued numbers. All parameters in these procedures (except the generation of region proposals) are optimized via a multi-task loss in an end-to-end way.

The multi-tasks loss L is defined as below to jointly train

classification and bounding-box regression,

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \quad (1)$$

where $L_{cls}(p, u) = -\log p_u$ calculates the log loss for ground truth class u and p_u is driven from the discrete probability distribution $p = (p_0, \dots, p_C)$ over the $C + 1$ outputs from the last FC layer. $L_{loc}(t^u, v)$ is defined over the predicted offsets $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ and ground-truth bounding-box regression targets $v = (v_x, v_y, v_w, v_h)$, where x, y, w, h denote the two coordinates of the box center, width, and height, respectively. Each t^u adopts the parameter settings in [15] to specify an object proposal with a log-space height/width shift and scale-invariant translation. The Iverson bracket indicator function $[u \geq 1]$ is employed to omit all background RoIs. To provide more robustness against outliers and eliminate the sensitivity in exploding gradients, a smooth L_1 loss is adopted to fit bounding-box regressors as below

$$L_{loc}(t^u, v) = \sum_{i \in x, y, w, h} \text{smooth}_{L_1}(t_i^u - v_i) \quad (2)$$

where

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \quad (3)$$

To accelerate the pipeline of Fast R-CNN, another two tricks are of necessity. On one hand, if training samples (i.e. RoIs) come from different images, back-propagation through the SPP layer becomes highly inefficient. Fast R-CNN samples mini-batches hierarchically, namely N images sampled randomly at first and then R/N RoIs sampled in each image, where R represents the number of RoIs. Critically, computation and memory are shared by RoIs from the same image in the forward and backward pass. On the other hand, much time is spent in computing the FC layers during the forward pass [16]. The truncated Singular Value Decomposition (SVD) [91] can be utilized to compress large FC layers and to accelerate the testing procedure.

In the Fast R-CNN, regardless of region proposal generation, the training of all network layers can be processed in a single-stage with a multi-task loss. It saves the additional expense on storage space, and improves both accuracy and efficiency with more reasonable training schemes.

4) *Faster R-CNN*: Despite the attempt to generate candidate boxes with biased sampling [88], state-of-the-art object detection networks mainly rely on additional methods, such as selective search and Edgebox, to generate a candidate pool of isolated region proposals. Region proposal computation is also a bottleneck in improving efficiency. To solve this problem, Ren et al. introduced an additional Region Proposal Network (RPN) [18], [92], which acts in a nearly cost-free way by sharing full-image conv features with detection network.

RPN is achieved with a fully-convolutional network, which has the ability to predict object bounds and scores at each position simultaneously. Similar to [78], RPN takes an image of arbitrary size to generate a set of rectangular object proposals. RPN operates on a specific conv layer with the preceding layers shared with the object detection network.

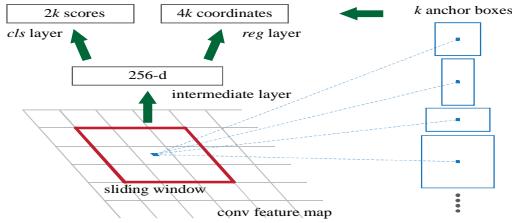


Fig. 6. The RPN in Faster R-CNN [18]. K predefined anchor boxes are convoluted with each sliding window to produce fixed-length vectors which are taken by cls and reg layer to obtain corresponding outputs.

The architecture of RPN is shown in Figure 6. The network slides over the conv feature map and fully connects to an $n \times n$ spatial window. A low dimensional vector (512-d for VGG16) is obtained in each sliding window and fed into two sibling FC layers, namely box-classification layer (cls) and box-regression layer (reg). This architecture is implemented with an $n \times n$ conv layer followed by two sibling 1×1 conv layers. To increase non-linearity, ReLU is applied to the output of the $n \times n$ conv layer.

The regressions towards true bounding boxes are achieved by comparing proposals relative to reference boxes (anchors). In the Faster R-CNN, anchors of 3 scales and 3 aspect ratios are adopted. The loss function is similar to (1).

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (4)$$

where p_i shows the predicted probability of the i -th anchor being an object. The ground truth label p_i^* is 1 if the anchor is positive, otherwise 0. t_i stores 4 parameterized coordinates of the predicted bounding box while t_i^* is related to the ground-truth box overlapping with a positive anchor. L_{cls} is a binary log loss and L_{reg} is a smoothed L_1 loss similar to (2). These two terms are normalized with the mini-batch size (N_{cls}) and the number of anchor locations (N_{reg}), respectively. In the form of fully-convolutional networks, Faster R-CNN can be trained end-to-end by back-propagation and SGD in an alternate training manner.

With the proposal of Faster R-CNN, region proposal based CNN architectures for object detection can really be trained in an end-to-end way. Also a frame rate of 5 FPS (Frame Per Second) on a GPU is achieved with state-of-the-art object detection accuracy on PASCAL VOC 2007 and 2012. However, the alternate training algorithm is very time-consuming and RPN produces object-like regions (including backgrounds) instead of object instances and is not skilled in dealing with objects with extreme scales or shapes.

5) *R-FCN*: Divided by the RoI pooling layer, a prevalent family [16], [18] of deep networks for object detection are composed of two subnetworks: a shared fully convolutional subnetwork (independent of RoIs) and an unshared RoI-wise subnetwork. This decomposition originates from pioneering classification architectures (e.g. AlexNet [6] and VGG16 [46]) which consist of a convolutional subnetwork and several FC layers separated by a specific spatial pooling layer.

Recent state-of-the-art image classification networks, such as Residual Nets (ResNets) [47] and GoogLeNets [45], [93], are fully convolutional. To adapt to these architectures, it's

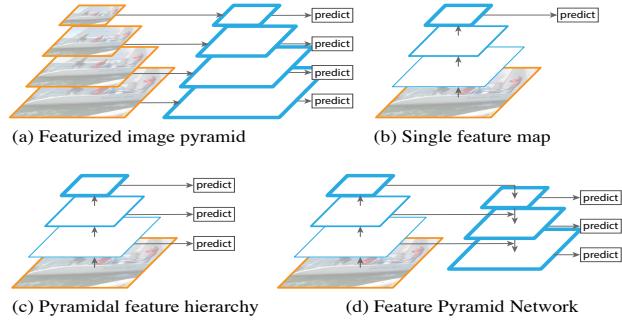


Fig. 7. The main concern of FPN [66]. (a) It is slow to use an image pyramid to build a feature pyramid. (b) Only single scale features is adopted for faster detection. (c) An alternative to the featurized image pyramid is to reuse the pyramidal feature hierarchy computed by a ConvNet. (d) FPN integrates both (b) and (c). Blue outlines indicate feature maps and thicker outlines denote semantically stronger features.

natural to construct a fully convolutional object detection network without ROI-wise subnetwork. However, it turns out to be inferior with such a naive solution [47]. This inconsistency is due to the dilemma of respecting translation variance in object detection compared with increasing translation invariance in image classification. In other words, shifting an object inside an image should be indiscriminative in image classification while any translation of an object in a bounding box may be meaningful in object detection. A manual insertion of the ROI pooling layer into convolutions can break down translation invariance at the expense of additional unshared region-wise layers. So Li et al. [65] proposed a region-based fully convolutional networks (R-FCN, Fig. S2).

Different from Faster R-CNN, for each category, the last conv layer of R-FCN produces a total of k^2 position-sensitive score maps with a fixed grid of $k \times k$ firstly and a position-sensitive ROI pooling layer is then appended to aggregate the responses from these score maps. Finally, in each ROI, k^2 position-sensitive scores are averaged to produce a $C + 1$ -d vector and softmax responses across categories are computed. Another $4k^2$ -d conv layer is appended to obtain class-agnostic bounding boxes.

With R-FCN, more powerful classification networks can be adopted to accomplish object detection in a fully-convolutional architecture by sharing nearly all the layers, and state-of-the-art results are obtained on both PASCAL VOC and Microsoft COCO [94] datasets at a test speed of 170ms per image.

6) *FPN*: Feature pyramids built upon image pyramids (featurized image pyramids) have been widely applied in many object detection systems to improve scale invariance [24], [64] (Figure 7(a)). However, training time and memory consumption increase rapidly. To this end, some techniques take only a single input scale to represent high-level semantics and increase the robustness to scale changes (Figure 7(b)), and image pyramids are built at test time which results in an inconsistency between train/test-time inferences [16], [18]. The in-network feature hierarchy in a deep ConvNet produces feature maps of different spatial resolutions while introduces large semantic gaps caused by different depths (Figure 7(c)). To avoid using low-level features, pioneer works [71], [95] usually build the pyramid starting from middle layers or just sum transformed feature responses, missing the higher-

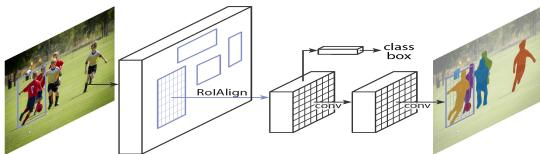


Fig. 8. The Mask R-CNN framework for instance segmentation [67].

resolution maps of the feature hierarchy.

Different from these approaches, FPN [66] holds an architecture with a bottom-up pathway, a top-down pathway and several lateral connections to combine low-resolution and semantically strong features with high-resolution and semantically weak features (Figure 7(d)). The bottom-up pathway, which is the basic forward backbone ConvNet, produces a feature hierarchy by downsampling the corresponding feature maps with a stride of 2. The layers owning the same size of output maps are grouped into the same network stage and the output of the last layer of each stage is chosen as the reference set of feature maps to build the following top-down pathway.

To build the top-down pathway, feature maps from higher network stages are upsampled at first and then enhanced with those of the same spatial size from the bottom-up pathway via lateral connections. A 1×1 conv layer is appended to the upsampled map to reduce channel dimensions and the mergence is achieved by element-wise addition. Finally, a 3×3 convolution is also appended to each merged map to reduce the aliasing effect of upsampling and the final feature map is generated. This process is iterated until the finest resolution map is generated.

As feature pyramid can extract rich semantics from all levels and be trained end-to-end with all scales, state-of-the-art representation can be obtained without sacrificing speed and memory. Meanwhile, FPN is independent of the backbone CNN architectures and can be applied to different stages of object detection (e.g. region proposal generation) and to many other computer vision tasks (e.g. instance segmentation).

7) *Mask R-CNN*: Instance segmentation [96] is a challenging task which requires detecting all objects in an image and segmenting each instance (semantic segmentation [97]). These two tasks are usually regarded as two independent processes. And the multi-task scheme will create spurious edge and exhibit systematic errors on overlapping instances [98]. To solve this problem, parallel to the existing branches in Faster R-CNN for classification and bounding box regression, the Mask R-CNN [67] adds a branch to predict segmentation masks in a pixel-to-pixel manner (Figure 8).

Different from the other two branches which are inevitably collapsed into short output vectors by FC layers, the segmentation mask branch encodes an $m \times m$ mask to maintain the explicit object spatial layout. This kind of fully convolutional representation requires fewer parameters but is more accurate than that of [97]. Formally, besides the two losses in (1) for classification and bounding box regression, an additional loss for segmentation mask branch is defined to reach a multi-task loss. An this loss is only associated with ground-truth class and relies on the classification branch to predict the category.

Because RoI pooling, the core operation in Faster R-CNN, performs a coarse spatial quantization for feature extraction,

misalignment is introduced between the ROI and the features. It affects classification little because of its robustness to small translations. However, it has a large negative effect on pixel-to-pixel mask prediction. To solve this problem, Mask R-CNN adopts a simple and quantization-free layer, namely RoIAlign, to preserve the explicit per-pixel spatial correspondence faithfully. RoIAlign is achieved by replacing the harsh quantization of ROI pooling with bilinear interpolation [99], computing the exact values of the input features at four regularly sampled locations in each ROI bin. In spite of its simplicity, this seemingly minor change improves mask accuracy greatly, especially under strict localization metrics.

Given the Faster R-CNN framework, the mask branch only adds a small computational burden and its cooperation with other tasks provides complementary information for object detection. As a result, Mask R-CNN is simple to implement with promising instance segmentation and object detection results. In a word, Mask R-CNN is a flexible and efficient framework for instance-level recognition, which can be easily generalized to other tasks (e.g. human pose estimation [7][S4]) with minimal modification.

8) *Multi-task Learning, Multi-scale Representation and Contextual Modelling*: Although the Faster R-CNN gets promising results with several hundred proposals, it still struggles in small-size object detection and localization, mainly due to the coarseness of its feature maps and limited information provided in particular candidate boxes. The phenomenon is more obvious on the Microsoft COCO dataset which consists of objects at a broad range of scales, less prototypical images, and requires more precise localization. To tackle these problems, it is of necessity to accomplish object detection with multi-task learning [100], multi-scale representation [95] and context modelling [101] to combine complementary information from multiple sources.

Multi-task Learning learns a useful representation for multiple correlated tasks from the same input [102], [103]. Brahmbhatt et al. introduced conv features trained for object segmentation and ‘stuff’ (amorphous categories such as ground and water) to guide accurate object detection of small objects (StuffNet) [100]. Dai et al. [97] presented Multitask Network Cascades of three networks, namely class-agnostic region proposal generation, pixel-level instance segmentation and regional instance classification. Li et al. incorporated the weakly-supervised object segmentation cues and region-based object detection into a multi-stage architecture to fully exploit the learned segmentation features [104].

Multi-scale Representation combines activations from multiple layers with skip-layer connections to provide semantic information of different spatial resolutions [66]. Cai et al. proposed the MS-CNN [105] to ease the inconsistency between the sizes of objects and receptive fields with multiple scale-independent output layers. Yang et al. investigated two strategies, namely scale-dependent pooling (SDP) and layer-wise cascaded rejection classifiers (CRC), to exploit appropriate scale-dependent conv features [33]. Kong et al. proposed the HyperNet to calculate the shared features between RPN and object detection network by aggregating and compressing hierarchical feature maps from different resolutions into a

uniform space [101].

Contextual Modelling improves detection performance by exploiting features from or around RoIs of different support regions and resolutions to deal with occlusions and local similarities [95]. Zhu et al. proposed the SegDeepM to exploit object segmentation which reduces the dependency on initial candidate boxes with Markov Random Field [106]. Moysset et al. took advantage of 4 directional 2D-LSTMs [107] to convey global context between different local regions and reduced trainable parameters with local parameter-sharing [108]. Zeng et al. proposed a novel GBD-Net by introducing gated functions to control message transmission between different support regions [109].

The Combination incorporates different components above into the same model to improve detection performance further. Gidaris et al. proposed the Multi-Region CNN (MR-CNN) model [110] to capture different aspects of an object, the distinct appearances of various object parts and semantic segmentation-aware features. To obtain contextual and multi-scale representations, Bell et al. proposed the Inside-Outside Net (ION) by exploiting information both inside and outside the RoI [95] with spatial recurrent neural networks [111] and skip pooling [101]. Zagoruyko et al. proposed the MultiPath architecture by introducing three modifications to the Fast R-CNN [112], including multi-scale skip connections [95], a modified foveal structure [110] and a novel loss function summing different IoU losses.

9) *Thinking in Deep Learning based Object Detection:* Apart from the above approaches, there are still many important factors for continued progress.

There is a large imbalance between the number of annotated objects and background examples. To address this problem, Shrivastava et al. proposed an effective online mining algorithm (OHEM) [113] for automatic selection of the hard examples, which leads to a more effective and efficient training.

Instead of concentrating on feature extraction, Ren et al. made a detailed analysis on object classifiers [114], and found that it is of particular importance for object detection to construct a deep and convolutional per-region classifier carefully, especially for ResNets [47] and GoogLeNets [45].

Traditional CNN framework for object detection is not skilled in handling significant scale variation, occlusion or truncation, especially when only 2D object detection is involved. To address this problem, Xiang et al. proposed a novel subcategory-aware region proposal network [60], which guides the generation of region proposals with subcategory information related to object poses and jointly optimize object detection and subcategory classification.

Ouyang et al. found that the samples from different classes follow a longtailed distribution [115], which indicates that different classes with distinct numbers of samples have different degrees of impacts on feature learning. To this end, objects are firstly clustered into visually similar class groups, and then a hierarchical feature learning scheme is adopted to learn deep representations for each group separately.

In order to minimize computational cost and achieve the state-of-the-art performance, with the ‘deep and thin’ design principle and following the pipeline of Fast R-CNN, Hong et

al. proposed the architecture of PVANET [116], which adopts some building blocks including concatenated ReLU [117], Inception [45], and HyperNet [101] to reduce the expense on multi-scale feature extraction and trains the network with batch normalization [43], residual connections [47], and learning rate scheduling based on plateau detection [47]. The PVANET achieves the state-of-the-art performance and can be processed in real time on Titan X GPU (21 FPS).

B. Regression/Classification Based Framework

Region proposal based frameworks are composed of several correlated stages, including region proposal generation, feature extraction with CNN, classification and bounding box regression, which are usually trained separately. Even in recent end-to-end module Faster R-CNN, an alternative training is still required to obtain shared convolution parameters between RPN and detection network. As a result, the time spent in handling different components becomes the bottleneck in real-time application.

One-step frameworks based on global regression/classification, mapping straightly from image pixels to bounding box coordinates and class probabilities, can reduce time expense. We firstly reviews some pioneer CNN models, and then focus on two significant frameworks, namely You only look once (YOLO) [17] and Single Shot MultiBox Detector (SSD) [71].

1) *Pioneer Works:* Previous to YOLO and SSD, many researchers have already tried to model object detection as a regression or classification task.

Szegedy et al. formulated object detection task as a DNN-based regression [118], generating a binary mask for the test image and extracting detections with a simple bounding box inference. However, the model has difficulty in handling overlapping objects, and bounding boxes generated by direct upsampling is far from perfect.

Pinheiro et al. proposed a CNN model with two branches: one generates class agnostic segmentation masks and the other predicts the likelihood of a given patch centered on an object [119]. Inference is efficient since class scores and segmentation can be obtained in a single model with most of the CNN operations shared.

Erhan et al. proposed regression based MultiBox to produce scored class-agnostic region proposals [68], [120]. A unified loss was introduced to bias both localization and confidences of multiple components to predict the coordinates of class-agnostic bounding boxes. However, a large quantity of additional parameters are introduced to the final layer.

Yoo et al. adopted an iterative classification approach to handle object detection and proposed an impressive end-to-end CNN architecture named AttentionNet [69]. Starting from the top-left (TL) and bottom-right (BR) corner of an image, AttentionNet points to a target object by generating quantized weak directions and converges to an accurate object boundary box with an ensemble of iterative predictions. However, the model becomes quite inefficient when handling multiple categories with a progressive two-step procedure.

Najibi et al. proposed a proposal-free iterative grid based object detector (G-CNN), which models object detection as

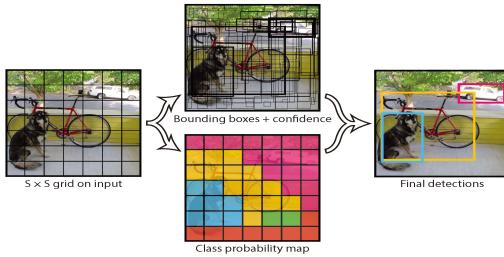


Fig. 9. Main idea of YOLO [17].

finding a path from a fixed grid to boxes tightly surrounding the objects [70]. Starting with a fixed multi-scale bounding box grid, G-CNN trains a regressor to move and scale elements of the grid towards objects iteratively. However, G-CNN has a difficulty in dealing with small or highly overlapping objects.

2) *YOLO*: Redmon et al. [17] proposed a novel framework called YOLO, which makes use of the whole topmost feature map to predict both confidences for multiple categories and bounding boxes. The basic idea of YOLO is exhibited in Figure 9. YOLO divides the input image into an $S \times S$ grid and each grid cell is responsible for predicting the object centered in that grid cell. Each grid cell predicts B bounding boxes and their corresponding confidence scores. Formally, confidence scores are defined as $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$, which indicates how likely there exist objects ($\Pr(\text{Object}) \geq 0$) and shows confidences of its prediction ($\text{IOU}_{\text{pred}}^{\text{truth}}$). At the same time, regardless of the number of boxes, C conditional class probabilities ($\Pr(\text{Class}_i | \text{Object})$) should also be predicted in each grid cell. It should be noticed that only the contribution from the grid cell containing an object is calculated.

At test time, class-specific confidence scores for each box are achieved by multiplying the individual box confidence predictions with the conditional class probabilities as follows.

$$\begin{aligned} & \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} * \Pr(\text{Class}_i | \text{Object}) \\ &= \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \end{aligned} \quad (5)$$

where the existing probability of class-specific objects in the box and the fitness between the predicted box and the object are both taken into consideration.

During training, the following loss function is optimized,

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ &+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ &+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\ &+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\ &+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \quad (6)$$

In a certain cell i , (x_i, y_i) denote the center of the box relative to the bounds of the grid cell, (w_i, h_i) are the normalized width and height relative to the image size, C_i represents confidence scores, $\mathbb{1}_i^{\text{obj}}$ indicates the existence of objects and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the prediction is conducted by the j th bounding box predictor. Note that only when an object is present in that grid cell, the loss function penalizes classification errors. Similarly, when the predictor is ‘responsible’ for the ground truth box (i.e. the highest IoU of any predictor in that grid cell is achieved), bounding box coordinate errors are penalized.

The YOLO consists of 24 conv layers and 2 FC layers, of which some conv layers construct ensembles of inception modules with 1×1 reduction layers followed by 3×3 conv layers. The network can process images in real-time at 45 FPS and a simplified version Fast YOLO can reach 155 FPS with better results than other real-time detectors. Furthermore, YOLO produces fewer false positives on background, which makes the cooperation with Fast R-CNN become possible. An improved version, YOLOv2, was later proposed in [72], which adopts several impressive strategies, such as BN, anchor boxes, dimension cluster and multi-scale training.

3) *SSD*: YOLO has a difficulty in dealing with small objects in groups, which is caused by strong spatial constraints imposed on bounding box predictions [17]. Meanwhile, YOLO struggles to generalize to objects in new/unusual aspect ratios/configurations and produces relatively coarse features due to multiple downsampling operations.

Aiming at these problems, Liu et al. proposed a Single Shot MultiBox Detector (SSD) [71], which was inspired by the anchors adopted in MultiBox [68], RPN [18] and multi-scale representation [95]. Given a specific feature map, instead of fixed grids adopted in YOLO, the SSD takes advantage of a set of default anchor boxes with different aspect ratios and scales to discretize the output space of bounding boxes. To handle objects with various sizes, the network fuses predictions from multiple feature maps with different resolutions .

The architecture of SSD is demonstrated in Figure 10. Given the VGG16 backbone architecture, SSD adds several feature layers to the end of the network, which are responsible for predicting the offsets to default boxes with different scales and aspect ratios and their associated confidences. The network is trained with a weighted sum of localization loss (e.g. Smooth L1) and confidence loss (e.g. Softmax), which is similar to (1). Final detection results are obtained by conducting NMS on multi-scale refined bounding boxes.

Integrating with hard negative mining, data augmentation and a larger number of carefully chosen default anchors, SSD significantly outperforms the Faster R-CNN in terms of accuracy on PASCAL VOC and COCO, while being three times faster. The SSD300 (input image size is 300×300) runs at 59 FPS, which is more accurate and efficient than YOLO. However, SSD is not skilled at dealing with small objects, which can be relieved by adopting better feature extractor backbone (e.g. ResNet101), adding deconvolution layers with skip connections to introduce additional large-scale context [73] and designing better network structure (e.g. Stem Block and Dense Block) [74].

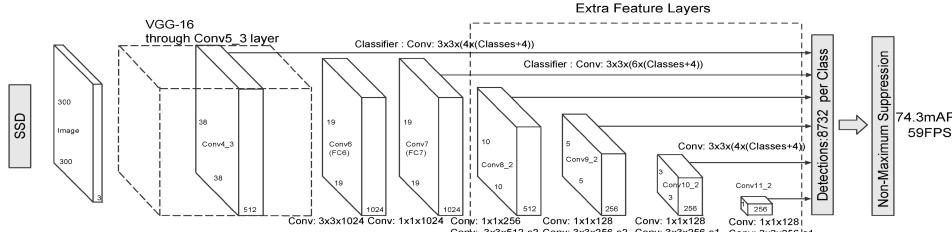


Fig. 10. The architecture of SSD 300 [71]. SSD adds several feature layers to the end of VGG16 backbone network to predict the offsets to default anchor boxes and their associated confidences. Final detection results are obtained by conducting NMS on multi-scale refined bounding boxes.

C. Experimental Evaluation

We compare various object detection methods on three benchmark datasets, including PASCAL VOC 2007 [25], PASCAL VOC 2012 [121] and Microsoft COCO [94]. The evaluated approaches include R-CNN [15], SPP-net [64], Fast R-CNN [16], NOC [114], Bayes [85], MR-CNN&S-CNN [105], Faster R-CNN [18], HyperNet [101], ION [95], MS-GR [104], StuffNet [100], SSD300 [71], SSD512 [71], OHEM [113], SDP+CRC [33], GCNN [70], SubCNN [60], GBD-Net [109], PVANET [116], YOLO [17], YOLOv2 [72], R-FCN [65], FPN [66], Mask R-CNN [67], DSSD [73] and DSOD [74]. If no specific instructions for the adopted framework are provided, the utilized model is a VGG16 [46] pretrained on 1000-way ImageNet classification task [39]. Due to the limitation of paper length, we only provide an overview, including proposal, learning method, loss function, programming language and platform, of the prominent architectures in Table I. Detailed experimental settings, which can be found in the original papers, are missed. In addition to the comparisons of detection accuracy, another comparison is provided to evaluate their test consumption on PASCAL VOC 2007.

1) *PASCAL VOC 2007/2012*: PASCAL VOC 2007 and 2012 datasets consist of 20 categories. The evaluation terms are Average Precision (AP) in each single category and mean Average Precision (mAP) across all the 20 categories. Comparative results are exhibited in Table II and III, from which the following remarks can be obtained.

- If incorporated with a proper way, more powerful backbone CNN models can definitely improve object detection performance (the comparison among R-CNN with AlexNet, R-CNN with VGG16 and SPP-net with ZF-Net [122]).
- With the introduction of SPP layer (SPP-net), end-to-end multi-task architecture (FRCN) and RPN (Faster R-CNN), object detection performance is improved gradually and apparently.
- Due to large quantities of trainable parameters, in order to obtain multi-level robust features, data augmentation is very important for deep learning based models (Faster R-CNN with ‘07’, ‘07+12’ and ‘07+12+coco’).
- Apart from basic models, there are still many other factors affecting object detection performance, such as multi-scale and multi-region feature extraction (e.g. MR-CNN), modified classification networks (e.g. NOC), additional information from other correlated tasks (e.g. StuffNet, HyperNet), multi-scale representation (e.g. ION) and mining of hard negative samples (e.g. OHEM).
- As YOLO is not skilled in producing object localizations

of high IoU, it obtains a very poor result on VOC 2012. However, with the complementary information from Fast R-CNN (YOLO+FRCN) and the aid of other strategies, such as anchor boxes, BN and fine grained features, the localization errors are corrected (YOLOv2).

- By combining many recent tricks and modelling the whole network as a fully convolutional one, R-FCN achieves a more obvious improvement of detection performance over other approaches.

2) *Microsoft COCO*: Microsoft COCO is composed of 300,000 fully segmented images, in which each image has an average of 7 object instances from a total of 80 categories. As there are a lot of less iconic objects with a broad range of scales and a stricter requirement on object localization, this dataset is more challenging than PASCAL 2012. Object detection performance is evaluated by AP computed under different degrees of IoUs and on different object sizes. The results are shown in Table IV.

Besides similar remarks to those of PASCAL VOC, some other conclusions can be drawn as follows from Table IV.

- Multi-scale training and test are beneficial in improving object detection performance, which provide additional information in different resolutions (R-FCN). FPN and DSSD provide some better ways to build feature pyramids to achieve multi-scale representation. The complementary information from other related tasks is also helpful for accurate object localization (Mask R-CNN with instance segmentation task).
- Overall, region proposal based methods, such as Faster R-CNN and R-FCN, perform better than regression/classification based approaches, namely YOLO and SSD, due to the fact that quite a lot of localization errors are produced by regression/classification based approaches.
- Context modelling is helpful to locate small objects, which provides additional information by consulting nearby objects and surroundings (GBD-Net and multi-path).
- Due to the existence of a large number of nonstandard small objects, the results on this dataset are much worse than those of VOC 2007/2012. With the introduction of other powerful frameworks (e.g. ResNeXt [123]) and useful strategies (e.g. multi-task learning [67], [124]), the performance can be improved.
- The success of DSOD in training from scratch stresses the importance of network design to release the requirements for perfect pre-trained classifiers on relevant tasks and large numbers of annotated samples.
- 3) *Timing Analysis*: Timing analysis (Table V) is conducted on Intel i7-6700K CPU with a single core and NVIDIA Titan

TABLE I
AN OVERVIEW OF PROMINENT GENERIC OBJECT DETECTION ARCHITECTURES.

Framework	Proposal	Multi-scale Input	Learning Method	Loss Function	Softmax Layer	End-to-end Train	Platform	Language
R-CNN [15]	Selective Search	-	SGD,BP	Hinge loss (classification),Bounding box regression	+	-	Caffe	Matlab
SPP-net [64]	EdgeBoxes	+	SGD	Hinge loss (classification),Bounding box regression	+	-	Caffe	Matlab
Fast RCNN [16]	Selective Search	+	SGD	Class Log loss+bounding box regression	+	-	Caffe	Python
Faster R-CNN [18]	RPN	+	SGD	Class Log loss+bounding box regression	+	+	Caffe	Python/Matlab
R-FCN [65]	RPN	+	SGD	Class Log loss+bounding box regression	-	+	Caffe	Matlab
Mask R-CNN [67]	RPN	+	SGD	Class Log loss+bounding box regression +Semantic sigmoid loss	+	+	TensorFlow/Keras	Python
FPN [66]	RPN	+	Synchronized SGD	Class Log loss+bounding box regression	+	+	TensorFlow	Python
YOLO [17]	-	-	SGD	Class sum-squared error loss+bounding box regression +object confidence+background confidence	+	+	Darknet	C
SSD [71]	-	-	SGD	Class softmax loss+bounding box regression	-	+	Caffe	C++
YOLOv2 [72]	-	-	SGD	Class sum-squared error loss+bounding box regression +object confidence+background confidence	+	+	Darknet	C

* '+' denotes that corresponding techniques are employed while '-' denotes that this technique is not considered. It should be noticed that R-CNN and SPP-net can not be trained end-to-end with a multi-task loss while the other architectures are based on multi-task joint training. As most of these architectures are re-implemented on different platforms with various programming languages, we only list the information associated with the versions by the referenced authors.

TABLE II
COMPARATIVE RESULTS ON VOC 2007 TEST SET (%).

Methods	Trained on	area	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN (Alex) [15]	07	68.1	72.8	56.8	43.0	36.8	66.3	74.2	67.6	34.4	63.5	54.5	61.2	69.1	68.6	58.7	33.4	62.9	51.1	62.5	68.6	58.5
R-CNN(VGG16) [15]	07	73.4	77.0	63.4	45.4	44.6	79.8	70.5	73.7	62.2	79.4	78.1	73.1	64.2	35.6	66.8	67.2	70.4	71.1	66.0	71.1	66.0
SPP-net [64]	07	68.5	71.7	58.7	41.9	42.5	67.7	72.1	73.8	34.7	67.0	63.4	66.0	72.5	71.3	58.9	32.8	60.9	56.1	67.9	68.8	60.9
GCNet [10]	07	68.3	77.3	55.4	52.4	38.6	70.3	71.0	81.0	47.7	73.6	64.7	77.2	80.5	75.8	54.7	34.7	64.4	75.6	69.4	66.8	73.4
GCNet* [85]	07	74.1	83.8	67.0	50.8	51.8	72.6	81.4	84.8	78.1	78.6	75.4	77.1	70.1	41.4	69.7	60.8	73.2	73.0	73.2	73.8	72.5
Faster R-CNN [16]	07+12	77.0	81.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	80.9	84.7	76.4	69.9	31.8	74.8	80.4	70.4	70.0	73.8	68.9
SubCNN [60]	07	70.2	80.5	69.5	60.3	47.9	79.0	78.7	84.2	48.5	73.9	63.0	82.7	80.6	76.0	70.2	38.2	62.4	67.7	77.7	60.5	68.5
StuffNet30 [100]	07	72.6	81.7	70.6	60.5	53.0	81.5	83.7	83.9	52.2	78.9	70.7	85.0	85.7	77.0	78.7	42.2	73.6	69.2	79.2	73.8	72.7
NOC [114]	07+12	76.3	81.4	74.4	61.7	60.8	84.7	78.2	89.9	53.0	79.2	69.2	83.2	83.2	78.5	68.0	45.0	71.6	76.7	82.2	75.7	73.3
MR-CNN&S-CNN [110]	07+12	80.3	84.1	78.5	70.8	68.5	88.0	85.9	87.8	60.3	85.2	73.7	87.2	86.5	85.0	76.4	48.5	76.3	75.5	85.0	81.0	78.2
HyperNet [101]	07+12	77.4	83.3	75.0	69.1	62.4	83.1	87.4	87.4	57.1	79.8	71.4	85.1	85.1	80.0	79.1	51.2	79.1	75.7	80.8	76.5	76.3
MS-GR [104]	07+12	80.0	81.0	77.4	72.1	64.3	88.2	88.4	89.4	64.4	85.4	73.1	87.3	87.4	85.1	79.6	50.1	78.4	79.5	86.9	75.5	78.6
OHEM+Fast R-CNN [113]	07+12	80.6	85.7	79.8	69.9	60.8	88.3	87.9	89.7	59.7	85.1	76.5	87.1	87.3	82.4	78.8	53.7	80.5	78.7	84.5	80.7	78.9
ION [95]	07+12+S	80.2	85.2	78.8	70.9	62.6	86.6	86.9	89.8	57.7	86.9	76.5	88.4	87.5	83.4	74.2	52.4	78.1	77.2	86.9	83.5	79.2
Faster R-CNN [18]	07+12	84.9	87.4	80.7	69.1	44.4	49.4	70.3	71.3	38.4	86.6	80.7	82.4	82.4	75.2	76.3	38.8	68.3	67.2	81.4	76.9	69.9
Faster R-CNN [18]	07+12+COCO	76.5	79.9	70.9	65.5	52.5	83.1	84.7	84.4	52.0	81.9	65.7	84.8	84.6	75.5	76.7	38.8	73.9	73.0	82.6	72.6	72.6
Faster R-CNN [18]	07+12+COCO	84.3	82.0	77.6	68.9	65.7	88.1	88.4	89.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9	78.8
SSD300 [71]	07+12+COCO	80.9	86.3	79.0	76.2																	
SSD512 [71]	07+12+COCO	86.6	88.3	82.4	76.0	66.3	88.6	88.6	86.0	60.5	85.4	76.7	89.2	84.5	81.9	81.5	85.9	87.9	79.9	79.6	80.4	82.2
SSD512 [71]	07+12+COCO	86.6	88.3	82.4	76.0	66.3	88.6	89.1	65.1	88.4	73.6	86.5	88.9	85.3	84.6	59.1	85.0	80.4	87.4	81.2	81.6	

* '07': VOC2007 trainval, '07+12': union of VOC2007 and VOC2012 trainval, '07+12+COCO': trained on COCO trainval35k at first and then fine-tuned on 07+12. The S in ION '07+12+S' denotes SBD segmentation labels.

TABLE III
COMPARATIVE RESULTS ON VOC 2012 TEST SET (%).

Methods	Trained on	area	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP	
R-CNN(Alex) [15]	12	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1	53.3	
R-CNN(VGG16) [15]	12	79.6	72.1	61.6	41.2	46.1	65.9	66.4	84.0	38.5	64.7	46.8	84.7	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	66.4	
Fast R-CNN [16]	07+12	82.9	86.1	64.1	44.4	49.4	70.3	71.3	84.6	47.7	68.6	55.8	82.7	77.1	77.7	66.7	41.4	68.0	60.0	70.0	66.2	66.4	
StuffNet30 [100]	12	83.0	76.9	71.2	51.6	50.1	76.4	75.7	87.8	48.3	74.8	55.7	85.7	81.2	80.3	79.5	72.2	44.2	71.8	61.0	78.5	65.4	70.0
NOC [114]	07+12	82.8	79.0	71.6	52.3	53.7	71.6	71.6	89.3	44.2	73.0	55.0	87.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2	68.4	68.8	
MR-CNN&S-CNN [110]	07+12	85.5	82.9	76.6	57.8	62.7	79.4	72.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.3	74.0	73.9	
HyperNet [101]	07+12	84.2	87.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.8	81.8	84.6	48.6	73.5	59.4	79.9	65.7	71.4	
OHEM+Fast R-CNN [113]	07+12+coco	90.1	87.4	79.9	65.8	66.3	86.1	85.0	92.9	62.4	83.4	69.5	90.6	88.9	88.9	83.6	59.0	82.0	74.7	88.2	77.3	80.1	
ION [95]	07+12+S	87.5	84.7	76.8	63.8	58.3	82.6	79.0	90.9	57.8	82.0	64.7	88.9	84.5	84.7	82.3	51.4	78.2	69.2	85.2	73.5	76.4	
Faster R-CNN [18]	07+12	84.9	79.8	74.3	53.9	49.8	77.5	75.9	85.8	45.6	77.1	55.3	86.9	81.7	80.9	79.4	40.1	72.6	60.9	81.2	61.5	70.4	
Faster R-CNN [18]	07+12+coco	87.4	83.6	76.8	62.9	59.4	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2	75.9	
YOLO [17]	07+12	77.0	67.2	57.7	38.3	22.7	68.3	55.9	51.4	36.2	60.8	48.5	72.3	72.3	72.3	73.5	32.0	68.8	59.3	70.7	57.9	68.8	
YOLO [17]	07+12+coco	83.8	87.0	78.5	73.5	55.8	43.4	79.7	89.4	49.4	75.7	57.0	87.5	87.5	87.5	81.0	74.7	41.8	73.5	68.5	88.5	76.8	
SSD300 [71]	07+12+coco	91.0	86.0	78.0	65.0	55.4	84.9	84.0	93.4	61.4	82.1	83.6	67.3	91.3	88.9	88.6	54.7	83.8	77.3	88.3	76.5	79.3	
SSD512 [71]	07+12+coco	91.4	88.6	82.6	71.4	63.1	87.4	88.1	93.9	66.9	86.6	66.3	92.0	91.7	90.8	88.5	60.9	87.0	75.4	90.2	80.4	82.2	
F-RCN (ResNet101) [16]	07+12+coco	92.3	89.9	86.7	74.7	75.2	86.7	89.0	95.8	70.2	90.4	66.5	95.0	93.2	92.1	91.1	71.0	89.7	76.0	92.0	83.4	85.0	

* FRCN*: Fast R-CNN with multi-scale training, R-FCN*: R-FCN with multi-scale training, R-FCN**: R-FCN with multi-scale training and testing, Mask: Mask R-CNN.

X GPU. Except for 'SS' which is processed with CPU, the other procedures related to CNN are all evaluated on GPU. From Table V, we can draw some conclusions as follows.

- By computing CNN features on shared feature maps (SPP-net), test consumption is reduced largely. Test time is further reduced with the unified multi-task learning (FRCN) and removal of additional region proposal generation stage (Faster R-CNN). It's also helpful to compress the parameters of FC layers with SVD [91] (PAVNET and FRCN).

TABLE V
COMPARISON OF TESTING CONSUMPTION ON VOC 07 TEST SET.

Methods	Trained on	mAP(%)	Test time(sec/img)	Rate(FPS)
SS+R-CNN [15]	07	66.0	32.84	0.03
SS+SPP-net [64]	07	63.1	2.3	0.44
SS+FRCN [16]	07+12	66.9	1.72	0.6
SS+CR-CNN [33]	07	68.9	0.47	2.1
SS+HyperNet* [101]	07+12	76.3	0.20	5
MR-CNN&S-CNN [110]	07+12	78.2	30	0.03
ION [95]</				

time at the cost of a drop in accuracy compared with region proposal based models. Also, region proposal based models can be modified into real-time systems with the introduction of other tricks [116] (PVANET), such as BN [43], residual connections [123].

IV. SALIENT OBJECT DETECTION

Visual saliency detection, one of the most important and challenging tasks in computer vision, aims to highlight the most dominant object regions in an image. Numerous applications incorporate the visual saliency to improve their performance, such as image cropping [125] and segmentation [126], image retrieval [57] and object detection [66].

Broadly, there are two branches of approaches in salient object detection, namely bottom-up (BU) [127] and top-down (TD) [128]. Local feature contrast plays the central role in BU salient object detection, regardless of the semantic contents of the scene. To learn local feature contrast, various local and global features are extracted from pixels, e.g. edges [129], spatial information [130]. However, high-level and multi-scale semantic information cannot be explored with these low-level features. As a result, low contrast salient maps instead of salient objects are obtained. TD salient object detection is task-oriented and takes prior knowledge about object categories to guide the generation of salient maps. Taking semantic segmentation as an example, a saliency map is generated in the segmentation to assign pixels to particular object categories via a TD approach [131]. In a word, TD saliency can be viewed as a focus-of-attention mechanism, which prunes BU salient points that are unlikely to be parts of the object [132].

A. Deep learning in Salient Object Detection

Due to the significance for providing high-level and multi-scale feature representation and the successful applications in many correlated computer vision tasks, such as semantic segmentation [131], edge detection [133] and generic object detection [16], it is feasible and necessary to extend CNN to salient object detection.

The early work by Eleonora Vig et al. [28] follows a completely automatic data-driven approach to perform a large-scale search for optimal features, namely an ensemble of deep networks with different layers and parameters. To address the problem of limited training data, Kummerer et al. proposed the Deep Gaze [134] by transferring from the AlexNet to generate a high dimensional feature space and create a saliency map. A similar architecture was proposed by Huang et al. to integrate saliency prediction into pre-trained object recognition DNNs [135]. The transfer is accomplished by fine-tuning DNNs' weights with an objective function based on the saliency evaluation metrics, such as Similarity, KL-Divergence and Normalized Scanpath Saliency.

Some works combined local and global visual clues to improve salient object detection performance. Wang et al. trained two independent deep CNNs (DNN-L and DNN-G) to capture local information and global contrast and predicted saliency maps by integrating both local estimation and global search [136]. Cholakkal et al. proposed a weakly supervised saliency detection framework to combine visual saliency from

bottom-up and top-down saliency maps, and refined the results with a multi-scale superpixel-averaging [137]. Zhao et al. proposed a multi-context deep learning framework, which utilizes a unified learning framework to model global and local context jointly with the aid of superpixel segmentation [138]. To predict saliency in videos, Bak et al. fused two static saliency models, namely spatial stream net and temporal stream net, into a two-stream framework with a novel empirically grounded data augmentation technique [139].

Complementary information from semantic segmentation and context modeling is beneficial. To learn internal representations of saliency efficiently, He et al. proposed a novel superpixelwise CNN approach called SuperCNN [140], in which salient object detection is formulated as a binary labeling problem. Based on a fully convolutional neural network, Li et al. proposed a multi-task deep saliency model, in which intrinsic correlations between saliency detection and semantic segmentation are set up [141]. However, due to the conv layers with large receptive fields and pooling layers, blurry object boundaries and coarse saliency maps are produced. Tang et al. proposed a novel saliency detection framework (CRPSD) [142], which combines region-level saliency estimation and pixel-level saliency prediction together with three closely related CNNs. Li et al. proposed a deep contrast network to combine segment-wise spatial pooling and pixel-level fully convolutional streams [143].

The proper integration of multi-scale feature maps is also of significance for improving detection performance. Based on Fast R-CNN, Wang et al. proposed the RegionNet by performing salient object detection with end-to-end edge preserving and multi-scale contextual modelling [144]. Liu et al. [27] proposed a multi-resolution convolutional neural network (Mr-CNN) to predict eye fixations, which is achieved by learning both bottom-up visual saliency and top-down visual factors from raw image data simultaneously. Cornia et al. proposed an architecture which combines features extracted at different levels of the CNN [145]. Li et al. proposed a multi-scale deep CNN framework to extract three scales of deep contrast features [146], namely the mean-subtracted region, the bounding box of its immediate neighboring regions and the masked entire image, from each candidate region.

It is efficient and accurate to train a direct pixel-wise CNN architecture to predict salient objects with the aids of RNNs and deconvolution networks. Pan et al. formulated saliency prediction as a minimization optimization on the Euclidean distance between the predicted saliency map and the ground truth and proposed two kinds of architectures [147]: a shallow one trained from scratch and a deeper one adapted from deconvoluted VGG network. As convolutional-deconvolution networks are not expert in recognizing objects of multiple scales, Kuen et al. proposed a recurrent attentional convolutional-deconvolution network (RACDNN) with several spatial transformer and recurrent network units to conquer this problem [148]. To fuse local, global and contextual information of salient objects, Tang et al. developed a deeply-supervised recurrent convolutional neural network (DSRCNN) to perform a full image-to-image saliency detection [149].

B. Experimental Evaluation

Four representative datasets, including ECSSD [156], HKU-IS [146], PASCALS [157], and SOD [158], are used to evaluate several state-of-the-art methods. ECSSD consists of 1000 structurally complex but semantically meaningful natural images. HKU-IS is a large-scale dataset containing over 4000 challenging images. Most of these images have more than one salient object and own low contrast. PASCALS is a subset chosen from the validation set of PASCAL VOC 2010 segmentation dataset and is composed of 850 natural images. The SOD dataset possesses 300 images containing multiple salient objects. The training and validation sets for different datasets are kept the same as those in [152].

Two standard metrics, namely F-measure and the mean absolute error (MAE), are utilized to evaluate the quality of a saliency map. Given precision and recall values pre-computed on the union of generated binary mask B and ground truth Z , F-measure is defined as below

$$F_\beta = \frac{(1 + \beta^2) \text{Precision} \times \text{Recall}}{\beta^2 \text{Precision} + \text{Recall}} \quad (7)$$

where β^2 is set to 0.3 in order to stress the importance of the precision value.

The MAE score is computed with the following equation

$$\text{MAE} = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W |\hat{S}(i, j) - \hat{Z}(i, j)| \quad (8)$$

where \hat{Z} and \hat{S} represent the ground truth and the continuous saliency map, respectively. W and H are the width and height of the salient area, respectively. This score stresses the importance of successfully detected salient objects over detected non-salient pixels [159].

The following approaches are evaluated: CHM [150], RC [151], DRFI [152], MC [138], MDF [146], LEGS [136], DSR [149], MTDNN [141], CRPSD [142], DCL [143], ELD [153], NLDF [154] and DSSC [155]. Among these methods, CHM, RC and DRFI are classical ones with the best performance [159], while the other methods are all associated with CNN. F-measure and MAE scores are shown in Table VI.

From Table VI, we can find that CNN based methods perform better than classic methods. MC and MDF combine the information from local and global context to reach a more accurate saliency. ELD refers to low-level handcrafted features for complementary information. LEGS adopts generic region proposals to provide initial salient regions, which may be insufficient for salient detection. DSR and MT act in different ways by introducing recurrent network and semantic segmentation, which provide insights for future improvements. CPRSD, DCL, NLDF and DSSC are all based on multi-scale representations and superpixel segmentation, which provide robust salient regions and smooth boundaries. DCL, NLDF and DSSC perform the best on these four datasets. DSSC earns the best performance by modelling scale-to-scale short-connections.

Overall, as CNN mainly provides salient information in local regions, most of CNN based methods need to model

visual saliency along region boundaries with the aid of superpixel segmentation. Meanwhile, the extraction of multi-scale deep CNN features is of significance for measuring local conspicuity. Finally, it's necessary to strengthen local connections between different CNN layers and as well to utilize complementary information from local and global context.

V. FACE DETECTION

Face detection is essential to many face applications and acts as an important pre-processing procedure to face recognition [160]–[162], face synthesis [163], [164] and facial expression analysis [165]. Different from generic object detection, this task is to recognize and locate face regions covering a very large range of scales (30-300 pts vs. 10-1000 pts). At the same time, faces have their unique object structural configurations (e.g. the distribution of different face parts) and characteristics (e.g. skin color). All these differences lead to special attention to this task. However, large visual variations of faces, such as occlusions, pose variations and illumination changes, impose great challenges for this task in real applications.

The most famous face detector proposed by Viola and Jones [166] trains cascaded classifiers with Haar-Like features and AdaBoost, achieving good performance with real-time efficiency. However, this detector may degrade significantly in real-world applications due to larger visual variations of human faces. Different from this cascade structure, Felzenszwalb et al. proposed a deformable part model (DPM) for face detection [24]. However, for these traditional face detection methods, high computational expenses and large quantities of annotations are required to achieve a reasonable result. Besides, their performance is greatly restricted by manually designed features and shallow architecture.

A. Deep learning in Face Detection

Recently, some CNN based face detection approaches have been proposed [167]–[169]. As less accurate localization results from independent regressions of object coordinates, Yu et al. [167] proposed a novel IoU loss function for predicting the four bounds of box jointly. Farfade et al. [168] proposed a Deep Dense Face Detector (DDFD) to conduct multi-view face detection, which is able to detect faces in a wide range of orientations without requirement of pose/landmark annotations. Yang et al. proposed a novel deep learning based face detection framework [169], which collects the responses from local facial parts (e.g. eyes, nose and mouths) to address face detection under severe occlusions and unconstrained pose variations. Yang et al. [170] proposed a scale-friendly detection network named ScaleFace, which splits a large range of target scales into smaller sub-ranges. Different specialized sub-networks are constructed on these sub-scales and combined into a single one to conduct end-to-end optimization. Hao et al. designed an efficient CNN to predict the scale distribution histogram of the faces and took this histogram to guide the zoom-in and zoom-out of the image [171]. Since the faces are approximately in uniform scale after zoom, compared with other state-of-the-art baselines, better performance is achieved with less computation cost. Besides, some generic detection frameworks

TABLE VI
COMPARISON BETWEEN STATE OF THE ART METHODS.

Dataset	Metrics	CHM [150]	RC [151]	DRFI [152]	MC [158]	MDF [146]	LEGS [136]	DSR [149]	MTDNN [141]	CRPSD [142]	DCL [143]	ELD [153]	NLDF [154]	DSSC [155]
PASCAL-S	wF_β MAE	0.631 0.222	0.640 0.225	0.679 0.211	0.721 0.147	0.764 0.145	0.756 0.157	0.697 0.128	0.818 0.170	0.776 0.063	0.822 0.108	0.767 0.121	0.831 0.099	0.830 0.080
ECSSD	wF_β MAE	0.722 0.195	0.741 0.187	0.787 0.166	0.822 0.107	0.833 0.108	0.827 0.118	0.872 0.037	0.810 0.160	0.849 0.046	0.898 0.071	0.865 0.098	0.905 0.063	0.915 0.052
HKU-IS	wF_β MAE	0.728 0.158	0.726 0.165	0.783 0.143	0.781 0.098	0.860 0.129	0.770 0.118	0.833 0.040	-	0.821 0.043	0.907 0.048	0.844 0.071	0.902 0.048	0.913 0.039
SOD	wF_β MAE	0.655 0.249	0.657 0.242	0.712 0.215	0.708 0.184	0.785 0.155	0.707 0.205	-	0.781 0.150	-	0.832 0.126	0.760 0.154	0.810 0.143	0.842 0.118

* The bigger wF_β or the smaller MAE is, the better the performance is.

are extended to face detection with different modifications, e.g. Faster R-CNN [29], [172], [173].

Some authors trained CNNs with other complementary tasks, such as 3D modelling and face landmarks, in a multi-task learning manner. Huang et al. proposed a unified end-to-end FCN framework called DenseBox to jointly conduct face detection and landmark localization [174]. Li et al. [175] proposed a multi-task discriminative learning framework which integrates a ConvNet with a fixed 3D mean face model in an end-to-end manner. In the framework, two issues are addressed to transfer from generic object detection to face detection, namely eliminating predefined anchor boxes by a 3D mean face model and replacing RoI pooling layer with a configuration pooling layer. Zhang et al. [176] proposed a deep cascaded multi-task framework named MTCNN which exploits the inherent correlations between face detection and alignment in unconstrained environment to boost up detection performance in a coarse-to-fine manner.

Reducing computational expenses is of necessity in real applications. To achieve real-time detection on mobile platform, Kalinovskii and Spitsyn proposed a new solution of frontal face detection based on compact CNN cascades [177]. This method takes a cascade of three simple CNNs to generate, classify and refine candidate object positions progressively. To reduce the effects of large pose variations, Chen et al. proposed a cascaded CNN denoted by Supervised Transformer Network [31]. This network takes a multi-task RPN to predict candidate face regions along with associated facial landmarks simultaneously, and adopts a generic R-CNN to verify the existence of valid faces. Yang et al. proposed a three-stage cascade structure based on FCNs [8], while in each stage, a multi-scale FCN is utilized to refine the positions of possible faces. Qin et al. proposed a unified framework which achieves better results with the complementary information from different jointly trained CNNs [178].

B. Experimental Evaluation

The FDDB [179] dataset has a total of 2,845 pictures in which 5,171 faces are annotated with elliptical shape. Two types of evaluations are used: the discrete score and continuous score. By varying the threshold of the decision rule, the ROC curve for the discrete scores can reflect the dependence of the detected face fractions on the number of false alarms. Compared with annotations, any detection with an IoU ratio exceeding 0.5 is treated as positive. Each annotation is only associated with one detection. The ROC curve for the continuous scores is the reflection of face localization quality.

The evaluated models cover DDFD [168], CascadeCNN [180], ACF-multiscale [181], Pico [182], HeadHunter [183],

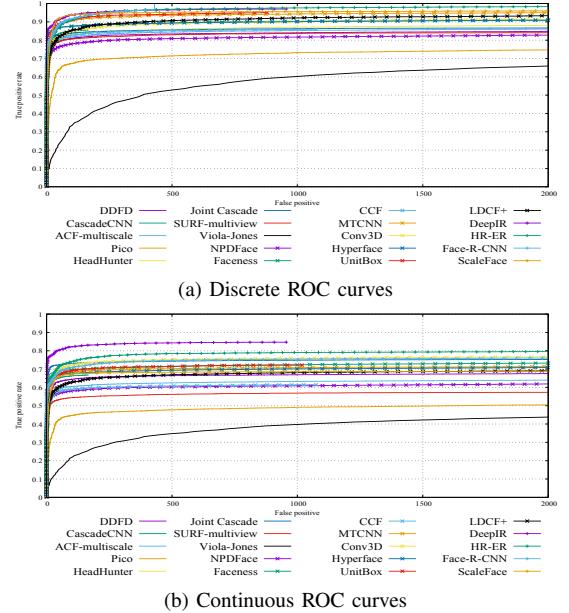


Fig. 11. The ROC curves of state-of-the-art methods on FDDB.

Joint Cascade [30], SURF-multiview [184], Viola-Jones [166], NPDFace [185], Faceness [169], CCF [186], MTCNN [176], Conv3D [175], Hyperface [187], UnitBox [167], LDCF+ [S2], DeepIR [173], HR-ER [188], Face-R-CNN [172] and ScaleFace [170]. ACF-multiscale, Pico, HeadHunter, Joint Cascade, SURF-multiview, Viola-Jones, NPDFace and LDCF+ are built on classic hand-crafted features while the rest methods are based on deep CNN features. The ROC curves are shown in Figure 11.

From Figure 11(a), in spite of relatively competitive results produced by LDCF+, it can be observed that most of classic methods perform with similar results and are outperformed by CNN based methods by a significant margin. From Figure 11(b), it can be observed that most of CNN based methods earn similar true positive rates between 60% and 70% while DeepIR and HR-ER perform much better than them. Among classic methods, Joint Cascade is still competitive. As earlier works, DDFD and CCF directly make use of generated feature maps and obtain relatively poor results. CascadeCNN builds cascaded CNNs to locate face regions, which is efficient but inaccurate. Faceness combines the decisions from different part detectors, resulting in precise face localizations while being time-consuming. The outstanding performance of MTCNN, Conv3D and Hyperface proves the effectiveness of multi-task learning. HR-ER and ScaleFace adaptively detect faces of different scales, and make a balance between accuracy and efficiency. DeepIR and Face-R-CNN are two extensions of the

Faster R-CNN architecture to face detection, which validate the significance and effectiveness of Faster R-CNN. Unitbox provides an alternative choice for performance improvements by carefully designing optimization loss.

From these results, we can draw the conclusion that CNN based methods are in the leading position. The performance can be improved by the following strategies: designing novel optimization loss, modifying generic detection pipelines, building meaningful network cascades, adapting scale-aware detection and learning multi-task shared CNN features.

VI. PEDESTRIAN DETECTION

Recently, pedestrian detection has been intensively studied, which has a close relationship to pedestrian tracking [189], [190], person re-identification [191], [192] and robot navigation [193], [194]. Prior to the recent progress in DCNN based methods [195], [196], some researchers combined boosted decision forests with hand-crafted features to obtain pedestrian detectors [197]–[199]. At the same time, to explicitly model the deformation and occlusion, part-based models [200] and explicit occlusion handling [201], [202] are of concern.

As there are many pedestrian instances of small sizes in typical scenarios of pedestrian detection (e.g. automatic driving and intelligent surveillance), the application of ROI pooling layer in generic object detection pipeline may result in ‘plain’ features due to collapsing bins. In the meantime, the main source of false predictions in pedestrian detection is the confusion of hard background instances, which is in contrast to the interference from multiple categories in generic object detection. As a result, different configurations and components are required to accomplish accurate pedestrian detection.

A. Deep learning in Pedestrian Detection

Although DCNNs have obtained excellent performance on generic object detection [16], [72], none of these approaches have achieved better results than the best hand-crafted feature based method [198] for a long time, even when part-based information and occlusion handling are incorporated [202]. Thereby, some researches have been conducted to analyze the reasons. Zhang et al. attempted to adapt generic Faster R-CNN [18] to pedestrian detection [203]. They modified the downstream classifier by adding boosted forests to shared, high-resolution conv feature maps and taking a RPN to handle small instances and hard negative examples. To deal with complex occlusions existing in pedestrian images, inspired by DPM [24], Tian et al. proposed a deep learning framework called DeepParts [204], which makes decisions based an ensemble of extensive part detectors. DeepParts has advantages in dealing with weakly labeled data, low IoU positive proposals and partial occlusion.

Other researchers also tried to combine complementary information from multiple data sources. CompACT-Deep adopts a complexity-aware cascade to combine hand-crafted features and fine-tuned DCNNs [195]. Based on Faster R-CNN, Liu et al. proposed multi-spectral deep neural networks for pedestrian detection to combine complementary information from color and thermal images [205]. Tian et al. [206] proposed a task-assistant CNN (TA-CNN) to jointly learn multiple tasks with

TABLE VII
DETAILED BREAKDOWN PERFORMANCE COMPARISONS OF STATE-OF-THE-ART MODELS ON CALTECH PEDESTRIAN DATASET. ALL NUMBERS ARE REPORTED IN L-AMR.

Method	Reasonable	All	Far	Medium	Near	none	partial	heavy
Checkerboards+ [198]	17.1	68.4	100	58.3	5.1	15.6	31.4	78.4
LDCF++[S2]	15.2	67.1	100	58.4	5.4	13.3	33.3	76.2
SCF+AlexNet [210]	23.3	70.3	100	62.3	10.2	20.0	48.5	74.7
SA-FastRCNN [211]	9.7	62.6	100	51.8	0	7.7	24.8	64.3
MS-CNN [105]	10.0	61.0	97.2	49.1	2.6	8.2	19.2	60.0
DeepParts [204]	11.9	64.8	100	56.4	4.8	10.6	19.9	60.4
CompACT-Deep [195]	11.8	64.4	100	53.2	4.0	9.6	25.1	65.8
RPN+BF [203]	9.6	64.7	100	53.9	2.3	7.7	24.2	74.2
F-DNN+SS [207]	8.2	50.3	77.5	33.2	2.8	6.7	15.1	53.4

multiple data sources and to combine pedestrian attributes with semantic scene attributes together. Du et al. proposed a deep neural network fusion architecture for fast and robust pedestrian detection [207]. Based on the candidate bounding boxes generated with SSD detectors [71], multiple binary classifiers are processed parallelly to conduct soft-rejection based network fusion (SNF) by consulting their aggregated degree of confidences.

However, most of these approaches are much more sophisticated than the standard R-CNN framework. CompACT-Deep consists of a variety of hand-crafted features, a small CNN model and a large VGG16 model [195]. DeepParts contains 45 fine-tuned DCNN models, and a set of strategies, including bounding box shifting handling and part selection, are required to arrive at the reported results [204]. So the modification and simplification is of significance to reduce the burden on both software and hardware to satisfy real-time detection demand. Tome et al. proposed a novel solution to adapt generic object detection pipeline to pedestrian detection by optimizing most of its stages [59]. Hu et al. [208] trained an ensemble of boosted decision models by reusing the conv feature maps, and a further improvement was gained with simple pixel labelling and additional complementary hand-crafted features. Tome et al. [209] proposed a reduced memory region based deep CNN architecture, which fuses regional responses from both ACF detectors and SVM classifiers into R-CNN. Ribeiro et al. addressed the problem of Human-Aware Navigation [32] and proposed a vision-based person tracking system guided by multiple camera sensors.

B. Experimental Evaluation

The evaluation is conducted on the most popular Caltech Pedestrian dataset [3]. The dataset was collected from the videos of a vehicle driving through an urban environment and consists of 250,000 frames with about 2300 unique pedestrians and 350,000 annotated bounding boxes (BBs). Three kinds of labels, namely ‘Person (clear identifications)’, ‘Person? (unclear identifications)’ and ‘People (large group of individuals)’, are assigned to different BBs. The performance is measured with the log-average miss rate (L-AMR) which is computed evenly spaced in log-space in the range 10^{-2} to 1 by averaging miss rate at the rate of nine false positives per image (FPPI) [3]. According to the differences in the height and visible part of the BBs, a total of 9 popular settings are adopted to evaluate different properties of these models. Details of these settings are as [3].

Evaluated methods include Checkerboards+ [198], LDCF++[S2], SCF+AlexNet [210], SA-FastRCNN [211], MS-CNN

[105], DeepParts [204], CompACT-Deep [195], RPN+BF [203] and F-DNN+SS [207]. The first two methods are based on hand-crafted features while the rest ones rely on deep CNN features. All results are exhibited in Table VII. From this table, we observe that different from other tasks, classic handcrafted features can still earn competitive results with boosted decision forests [203], ACF [197] and HOG+LUV channels [S2]. As an early attempt to adapt CNN to pedestrian detection, the features generated by SCF+AlexNet are not so discriminant and produce relatively poor results. Based on multiple CNNs, DeepParts and CompACT-Deep accomplish detection tasks via different strategies, namely local part integration and cascade network. The responses from different local part detectors make DeepParts robust to partial occlusions. However, due to complexity, it is too time-consuming to achieve real-time detection. The multi-scale representation of MS-CNN improves accuracy of pedestrian locations. SA-FastRCNN extends Fast R-CNN to automatically detecting pedestrians according to their different scales, which has trouble when there are partial occlusions. RPN+BF combines the detectors produced by Faster R-CNN with boosting decision forest to accurately locate different pedestrians. F-DNN+SS, which is composed of multiple parallel classifiers with soft rejections, performs the best followed by RPN+BF, SA-FastRCNN and MS-CNN.

In short, CNN based methods can provide more accurate candidate boxes and multi-level semantic information for identifying and locating pedestrians. Meanwhile, handcrafted features are complementary and can be combined with CNN to achieve better results. The improvements over existing CNN methods can be obtained by carefully designing the framework and classifiers, extracting multi-scale and part based semantic information and searching for complementary information from other related tasks, such as segmentation.

VII. PROMISING FUTURE DIRECTIONS AND TASKS

In spite of rapid development and achieved promising progress of object detection, there are still many open issues for future work.

The first one is small object detection such as occurring in COCO dataset and in face detection task. To improve localization accuracy on small objects under partial occlusions, it is necessary to modify network architectures from the following aspects.

- **Multi-task joint optimization and multi-modal information fusion.** Due to the correlations between different tasks within and outside object detection, multi-task joint optimization has already been studied by many researchers [16] [18]. However, apart from the tasks mentioned in Subs. III-A8, it is desirable to think over the characteristics of different sub-tasks of object detection (e.g. superpixel semantic segmentation in salient object detection) and extend multi-task optimization to other applications such as instance segmentation [66], multi-object tracking [202] and multi-person pose estimation [S4]. Besides, given a specific application, the information from different modalities, such as text [212], thermal data [205] and images [65], can be fused together to achieve a more discriminant network.

- **Scale adaption.** Objects usually exist in different scales, which is more apparent in face detection and pedestrian detection. To increase the robustness to scale changes, it is demanded to train scale-invariant, multi-scale or scale-adaptive detectors. For scale-invariant detectors, more powerful backbone architectures (e.g. ResNext [123]), negative sample mining [113], reverse connection [213] and sub-category modelling [60] are all beneficial. For multi-scale detectors, both the FPN [66] which produces multi-scale feature maps and Generative Adversarial Network [214] which narrows representation differences between small objects and the large ones with a low-cost architecture provide insights into generating meaningful feature pyramid. For scale-adaptive detectors, it is useful to combine knowledge graph [215], attentional mechanism [216], cascade network [180] and scale distribution estimation [171] to detect objects adaptively.

- **Spatial correlations and contextual modelling.** Spatial distribution plays an important role in object detection. So region proposal generation and grid regression are taken to obtain probable object locations. However, the correlations between multiple proposals and object categories are ignored. Besides, the global structure information is abandoned by the position-sensitive score maps in R-FCN. To solve these problems, we can refer to diverse subset selection [217] and sequential reasoning tasks [218] for possible solutions. It is also meaningful to mask salient parts and couple them with the global structure in a joint-learning manner [219].

The second one is to release the burden on manual labor and accomplish real-time object detection, with the emergence of large-scale image and video data. The following three aspects can be taken into account.

- **Cascade network.** In a cascade network, a cascade of detectors are built in different stages or layers [180], [220]. And easily distinguishable examples are rejected at shallow layers so that features and classifiers at latter stages can handle more difficult samples with the aid of the decisions from previous stages. However, current cascades are built in a greedy manner, where previous stages in cascade are fixed when training a new stage. So the optimizations of different CNNs are isolated, which stresses the necessity of end-to-end optimization for CNN cascade. At the same time, it is also a matter of concern to build contextual associated cascade networks with existing layers.

- **Unsupervised and weakly supervised learning.** It's very time consuming to manually draw large quantities of bounding boxes. To release this burden, semantic prior [55], unsupervised object discovery [221], multiple instance learning [222] and deep neural network prediction [47] can be integrated to make best use of image-level supervision to assign object category tags to corresponding object regions and refine object boundaries. Furthermore, weakly annotations (e.g. center-click annotations [223]) are also helpful for achieving high-quality detectors with modest annotation efforts, especially aided by the mobile platform.

- **Network optimization.** Given specific applications and platforms, it is significant to make a balance among speed,

memory and accuracy by selecting an optimal detection architecture [116], [224]. However, despite that detection accuracy is reduced, it is more meaningful to learn compact models with fewer number of parameters [209]. And this situation can be relieved by introducing better pre-training schemes [225], knowledge distillation [226] and hint learning [227]. DSOD also provides a promising guideline to train from scratch to bridge the gap between different image sources and tasks [74].

The third one is to extend typical methods for 2D object detection to adapt 3D object detection and video object detection, with the requirements from autonomous driving, intelligent transportation and intelligent surveillance.

- **3D object detection.** With the applications of 3D sensors (e.g. LIDAR and camera), additional depth information can be utilized to better understand the images in 2D and extend the image-level knowledge to the real world. However, seldom of these 3D-aware techniques aim to place correct 3D bounding boxes around detected objects. To achieve better bounding results, multi-view representation [181] and 3D proposal network [228] may provide some guidelines to encode depth information with the aid of inertial sensors (accelerometer and gyrometer) [229].

- **Video object detection.** Temporal information across different frames play an important role in understanding the behaviors of different objects. However, the accuracy suffers from degenerated object appearances (e.g., motion blur and video defocus) in videos and the network is usually not trained end-to-end. To this end, spatiotemporal tubelets [230], optical flow [199] and LSTM [107] should be considered to fundamentally model object associations between consecutive frames.

VIII. CONCLUSION

Due to its powerful learning ability and advantages in dealing with occlusion, scale transformation and background switches, deep learning based object detection has been a research hotspot in recent years. This paper provides a detailed review on deep learning based object detection frameworks which handle different sub-problems, such as occlusion, clutter and low resolution, with different degrees of modifications on R-CNN. The review starts on generic object detection pipelines which provide base architectures for other related tasks. Then, three other common tasks, namely salient object detection, face detection and pedestrian detection, are also briefly reviewed. Finally, we propose several promising future directions to gain a thorough understanding of the object detection landscape. This review is also meaningful for the developments in neural networks and related learning systems, which provides valuable insights and guidelines for future progress.

ACKNOWLEDGMENTS

This research was supported by the National Natural Science Foundation of China (No.61672203 & 61375047 & 91746209), the National Key Research and Development Program of China (2016YFB1000901), and Anhui Natural Science Funds for Distinguished Young Scholar (No.170808J08).

REFERENCES

- [1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, p. 1627, 2010.
- [2] K. K. Sung and T. Poggio, “Example-based learning for view-based human face detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 1, pp. 39–51, 2002.
- [3] C. Wojek, P. Dollar, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, p. 743, 2012.
- [4] H. Kobatake and Y. Yoshinaga, “Detection of spicules on mammogram based on skeleton analysis,” *IEEE Trans. Med. Imag.*, vol. 15, no. 3, pp. 235–245, 1996.
- [5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *ACM MM*, 2014.
- [6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, 2012.
- [7] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime multi-person 2d pose estimation using part affinity fields,” in *CVPR*, 2017.
- [8] Z. Yang and R. Nevatia, “A multi-scale cascade fully convolutional network face detector,” in *ICPR*, 2016.
- [9] C. Chen, A. Seff, A. L. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *ICCV*, 2015.
- [10] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, “Multi-view 3d object detection network for autonomous driving,” in *CVPR*, 2017.
- [11] A. Dundar, J. Jin, B. Martini, and E. Culurciello, “Embedded streaming deep neural networks accelerator with applications,” *IEEE Trans. Neural Netw. & Learning Syst.*, vol. 28, no. 7, pp. 1572–1583, 2017.
- [12] R. J. Cintra, S. Duffner, C. Garcia, and A. Leite, “Low-complexity approximate convolutional neural networks,” *IEEE Trans. Neural Netw. & Learning Syst.*, vol. PP, no. 99, pp. 1–12, 2018.
- [13] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, “Cost-sensitive learning of deep feature representations from imbalanced data,” *IEEE Trans. Neural Netw. & Learning Syst.*, vol. PP, no. 99, pp. 1–15, 2017.
- [14] A. Stuhlsatz, J. Lippel, and T. Zielke, “Feature extraction with deep neural networks by a generalized discriminant analysis,” *IEEE Trans. Neural Netw. & Learning Syst.*, vol. 23, no. 4, pp. 596–608, 2012.
- [15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *CVPR*, 2014.
- [16] R. Girshick, “Fast r-cnn,” in *ICCV*, 2015.
- [17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *CVPR*, 2016.
- [18] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *NIPS*, 2015, pp. 91–99.
- [19] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. of Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [20] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *CVPR*, 2005.
- [21] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *ICIP*, 2002.
- [22] C. Cortes and V. Vapnik, “Support vector machine,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [23] Y. Freund and R. E. Schapire, “A desicion-theoretic generalization of on-line learning and an application to boosting,” *J. of Comput. & Sys. Sci.*, vol. 13, no. 5, pp. 663–671, 1997.
- [24] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 1627–1645, 2010.
- [25] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge 2007 (voc 2007) results (2007),” 2008.
- [26] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [27] N. Liu, J. Han, D. Zhang, S. Wen, and T. Liu, “Predicting eye fixations using convolutional neural networks,” in *CVPR*, 2015.
- [28] E. Vig, M. Dorr, and D. Cox, “Large-scale optimization of hierarchical features for saliency prediction in natural images,” in *CVPR*, 2014.
- [29] H. Jiang and E. Learned-Miller, “Face detection with the faster r-cnn,” in *FG*, 2017.
- [30] D. Chen, S. Ren, Y. Wei, X. Cao, and J. Sun, “Joint cascade face detection and alignment,” in *ECCV*, 2014.

- [31] D. Chen, G. Hua, F. Wen, and J. Sun, "Supervised transformer network for efficient face detection," in *ECCV*, 2016.
- [32] D. Ribeiro, A. Mateus, J. C. Nascimento, and P. Miraldo, "A real-time pedestrian detector using deep learning for human-aware navigation," *arXiv:1607.04441*, 2016.
- [33] F. Yang, W. Choi, and Y. Lin, "Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers," in *CVPR*, 2016.
- [34] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognition and Image Anal.*, vol. 26, no. 1, p. 9, 2016.
- [35] W. Pitts and W. S. McCulloch, "How we know universals the perception of auditory and visual forms," *The Bulletin of Mathematical Biophysics*, vol. 9, no. 3, pp. 127–147, 1947.
- [36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by back-propagation of errors," *Nature*, vol. 323, no. 323, pp. 533–536, 1986.
- [37] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Sci.*, vol. 313, pp. 504–507, 2006.
- [38] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.
- [39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
- [40] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *INTERSPEECH*, 2010.
- [41] G. Dahl, A.-r. Mohamed, G. E. Hinton *et al.*, "Phone recognition with the mean-covariance restricted boltzmann machine," in *NIPS*, 2010.
- [42] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580*, 2012.
- [43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.
- [44] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv:1312.6229*, 2013.
- [45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.
- [46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
- [47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
- [48] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.
- [49] M. Oquab, L. Bottou, I. Laptev, J. Sivic *et al.*, "Weakly supervised object recognition with convolutional neural networks," in *NIPS*, 2014.
- [50] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *CVPR*, 2014.
- [51] F. M. Wadley, "Probit analysis: a statistical treatment of the sigmoid response curve," *Annals of the Entomological Soc. of America*, vol. 67, no. 4, pp. 549–553, 1947.
- [52] K. Kavukcuoglu, R. Fergus, Y. LeCun *et al.*, "Learning invariant features through topographic filter maps," in *CVPR*, 2009.
- [53] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun, "Learning convolutional feature hierarchies for visual recognition," in *NIPS*, 2010.
- [54] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *CVPR*, 2010.
- [55] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *ICCV*, 2015.
- [56] Z.-Q. Zhao, B.-J. Xie, Y.-m. Cheung, and X. Wu, "Plant leaf identification via a growing convolutional neural network with progressive sample learning," in *ACCV*, 2014.
- [57] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *ECCV*, 2014.
- [58] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li, "Deep learning for content-based image retrieval: A comprehensive study," in *ACM MM*, 2014.
- [59] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, "Deep convolutional neural networks for pedestrian detection," *Signal Process.: Image Commun.*, vol. 47, pp. 482–489, 2016.
- [60] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, "Subcategory-aware convolutional neural networks for object proposals and detection," in *WACV*, 2017.
- [61] Z.-Q. Zhao, H. Bian, D. Hu, W. Cheng, and H. Glotin, "Pedestrian detection based on fast r-cnn and batch normalization," in *ICIC*, 2017.
- [62] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *ICML*, 2011.
- [63] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue, "Modeling spatial-temporal clues in a hybrid deep learning framework for video classification," in *ACM MM*, 2015.
- [64] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015.
- [65] Y. Li, K. He, J. Sun *et al.*, "R-fcn: Object detection via region-based fully convolutional networks," in *NIPS*, 2016, pp. 379–387.
- [66] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017.
- [67] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask r-cnn," in *ICCV*, 2017.
- [68] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *CVPR*, 2014.
- [69] D. Yoo, S. Park, J.-Y. Lee, A. S. Paek, and I. So Kweon, "Attentionnet: Aggregating weak directions for accurate object detection," in *CVPR*, 2015.
- [70] M. Najibi, M. Rastegari, and L. S. Davis, "G-cnn: an iterative grid based object detector," in *CVPR*, 2016.
- [71] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Dssd: Single shot multibox detector," in *ECCV*, 2016.
- [72] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv:1612.08242*, 2016.
- [73] C. Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv:1701.06659*, 2017.
- [74] Z. Shen, Z. Liu, J. Li, Y. G. Jiang, Y. Chen, and X. Xue, "Dsod: Learning deeply supervised object detectors from scratch," in *ICCV*, 2017.
- [75] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *ICANN*, 2011.
- [76] G. W. Taylor, I. Spiro, C. Bregler, and R. Fergus, "Learning invariance through imitation," in *CVPR*, 2011.
- [77] X. Ren and D. Ramanan, "Histograms of sparse codes for object detection," in *CVPR*, 2013.
- [78] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *Int. J. of Comput. Vision*, vol. 104, no. 2, pp. 154–171, 2013.
- [79] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun, "Pedestrian detection with unsupervised multi-stage feature learning," in *CVPR*, 2013.
- [80] P. Krähenbühl and V. Koltun, "Geodesic object proposals," in *ECCV*, 2014.
- [81] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, "Multiscale combinatorial grouping," in *CVPR*, 2014.
- [82] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *ECCV*, 2014.
- [83] W. Kuo, B. Hariharan, and J. Malik, "Deepbox: Learning objectness with convolutional networks," in *ICCV*, 2015.
- [84] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár, "Learning to refine object segments," in *ECCV*, 2016.
- [85] Y. Zhang, K. Sohn, R. Villegas, G. Pan, and H. Lee, "Improving object detection with deep convolutional networks via bayesian optimization and structured prediction," in *CVPR*, 2015.
- [86] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgbd images for object detection and segmentation," in *ECCV*, 2014.
- [87] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy *et al.*, "Deepid-net: Deformable deep convolutional neural networks for object detection," in *CVPR*, 2015.
- [88] K. Lenc and A. Vedaldi, "R-cnn minus r," *arXiv:1506.06981*, 2015.
- [89] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, 2006.
- [90] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *ECCV*, 2010.
- [91] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Interspeech*, 2013.

- [92] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.
- [93] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.
- [94] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014.
- [95] S. Bell, C. Lawrence Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *CVPR*, 2016.
- [96] A. Arnab and P. H. S. Torr, "Pixelwise instance segmentation with a dynamically instantiated network," in *CVPR*, 2017.
- [97] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," in *CVPR*, 2016.
- [98] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," in *CVPR*, 2017.
- [99] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *CVPR*, 2015.
- [100] S. Brahmbhatt, H. I. Christensen, and J. Hays, "Stuffnet: Using stuffto improve object detection," in *WACV*, 2017.
- [101] T. Kong, A. Yao, Y. Chen, and F. Sun, "Hypernet: Towards accurate region proposal generation and joint object detection," in *CVPR*, 2016.
- [102] A. Pentina, V. Sharmanska, and C. H. Lampert, "Curriculum learning of multiple tasks," in *CVPR*, 2015.
- [103] J. Yim, H. Jung, B. Yoo, C. Choi, D. Park, and J. Kim, "Rotating your face using multi-task deep neural network," in *CVPR*, 2015.
- [104] J. Li, X. Liang, J. Li, T. Xu, J. Feng, and S. Yan, "Multi-stage object detection with group recursive learning," *arXiv:1608.05159*, 2016.
- [105] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *ECCV*, 2016.
- [106] Y. Zhu, R. Urtasun, R. Salakhutdinov, and S. Fidler, "segdeephm: Exploiting segmentation and context in deep neural networks for object detection," in *CVPR*, 2015.
- [107] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *CVPR*, 2015.
- [108] B. Moysset, C. Kermorvant, and C. Wolf, "Learning to detect and localize many objects from few examples," *arXiv:1611.05664*, 2016.
- [109] X. Zeng, W. Ouyang, B. Yang, J. Yan, and X. Wang, "Gated bi-directional cnn for object detection," in *ECCV*, 2016.
- [110] S. Gidaris and N. Komodakis, "Object detection via a multi-region and semantic segmentation-aware cnn model," in *CVPR*, 2015.
- [111] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, pp. 2673–2681, 1997.
- [112] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár, "A multipath network for object detection," *arXiv:1604.02135*, 2016.
- [113] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," in *CVPR*, 2016.
- [114] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun, "Object detection networks on convolutional feature maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1476–1481, 2017.
- [115] W. Ouyang, X. Wang, C. Zhang, and X. Yang, "Factors in finetuning deep model for object detection with long-tail distribution," in *CVPR*, 2016.
- [116] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park, "Pvanet: Lightweight deep neural networks for real-time object detection," *arXiv:1611.08588*, 2016.
- [117] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *ICML*, 2016.
- [118] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *NIPS*, 2013.
- [119] P. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to segment object candidates," in *NIPS*, 2015.
- [120] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, high-quality object detection," *arXiv:1412.1441*, 2014.
- [121] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge 2012 (voc2012) results (2012)," in <http://www.pascal-network.org/challenges/VOC/voc2011/index.html>, 2011.
- [122] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.
- [123] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.
- [124] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," *arXiv:1703.06211*, 2017.
- [125] C. Rother, L. Bordeau, Y. Hamadi, and A. Blake, "Autocollage," *ACM Trans. on Graphics*, vol. 25, no. 3, pp. 847–852, 2006.
- [126] C. Jung and C. Kim, "A unified spectral-domain approach for saliency detection and its application to automatic object segmentation," *IEEE Trans. Image Process.*, vol. 21, no. 3, pp. 1272–1283, 2012.
- [127] W.-C. Tu, S. He, Q. Yang, and S.-Y. Chien, "Real-time salient object detection with a minimum spanning tree," in *CVPR*, 2016.
- [128] J. Yang and M.-H. Yang, "Top-down visual saliency via joint crf and dictionary learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 3, pp. 576–588, 2017.
- [129] P. L. Rosin, "A simple method for detecting salient regions," *Pattern Recognition*, vol. 42, no. 11, pp. 2363–2371, 2009.
- [130] T. Liu, Z. Yuan, J. Sun, J. Wang, N. Zheng, X. Tang, and H.-Y. Shum, "Learning to detect a salient object," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 2, pp. 353–367, 2011.
- [131] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.
- [132] D. Gao, S. Han, and N. Vasconcelos, "Discriminant saliency, the detection of suspicious coincidences, and applications to visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, pp. 989–1005, 2009.
- [133] S. Xie and Z. Tu, "Holistically-nested edge detection," in *ICCV*, 2015.
- [134] M. Kümmeler, L. Theis, and M. Bethge, "Deep gaze i: Boosting saliency prediction with feature maps trained on imagenet," *arXiv:1411.1045*, 2014.
- [135] X. Huang, C. Shen, X. Boix, and Q. Zhao, "Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks," in *ICCV*, 2015.
- [136] L. Wang, H. Lu, X. Ruan, and M.-H. Yang, "Deep networks for saliency detection via local estimation and global search," in *CVPR*, 2015.
- [137] H. Cholakkal, J. Johnson, and D. Rajan, "Weakly supervised top-down salient object detection," *arXiv:1611.05345*, 2016.
- [138] R. Zhao, W. Ouyang, H. Li, and X. Wang, "Saliency detection by multi-context deep learning," in *CVPR*, 2015.
- [139] Ç. Bak, A. Erdem, and E. Erdem, "Two-stream convolutional networks for dynamic saliency prediction," *arXiv:1607.04730*, 2016.
- [140] S. He, R. W. Lau, W. Liu, Z. Huang, and Q. Yang, "Supercnn: A superpixelwise convolutional neural network for salient object detection," *Int. J. of Comput. Vision*, vol. 115, no. 3, pp. 330–344, 2015.
- [141] X. Li, L. Zhao, L. Wei, M.-H. Yang, F. Wu, Y. Zhuang, H. Ling, and J. Wang, "Deepsaliency: Multi-task deep neural network model for salient object detection," *IEEE Trans. Image Process.*, vol. 25, no. 8, pp. 3919–3930, 2016.
- [142] Y. Tang and X. Wu, "Saliency detection via combining region-level and pixel-level predictions with cnns," in *ECCV*, 2016.
- [143] G. Li and Y. Yu, "Deep contrast learning for salient object detection," in *CVPR*, 2016.
- [144] X. Wang, H. Ma, S. You, and X. Chen, "Edge preserving and multi-scale contextual neural network for salient object detection," *arXiv:1608.08029*, 2016.
- [145] M. Cornia, L. Baraldi, G. Serra, and R. Cucchiara, "A deep multi-level network for saliency prediction," in *ICPR*, 2016.
- [146] G. Li and Y. Yu, "Visual saliency detection based on multiscale deep cnn features," *IEEE Trans. Image Process.*, vol. 25, no. 11, pp. 5012–5024, 2016.
- [147] J. Pan, E. Sayrol, X. Giro-i Nieto, K. McGuinness, and N. E. O'Connor, "Shallow and deep convolutional networks for saliency prediction," in *CVPR*, 2016.
- [148] J. Kuen, Z. Wang, and G. Wang, "Recurrent attentional networks for saliency detection," in *CVPR*, 2016.
- [149] Y. Tang, X. Wu, and W. Bu, "Deeply-supervised recurrent convolutional neural network for saliency detection," in *ACM MM*, 2016.
- [150] X. Li, Y. Li, C. Shen, A. Dick, and A. Van Den Hengel, "Contextual hypergraph modeling for salient object detection," in *ICCV*, 2013.
- [151] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. Torr, and S.-M. Hu, "Global contrast based salient region detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 569–582, 2015.
- [152] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li, "Salient object detection: A discriminative regional feature integration approach," in *CVPR*, 2013.
- [153] G. Lee, Y.-W. Tai, and J. Kim, "Deep saliency with encoded low level distance map and high level features," in *CVPR*, 2016.
- [154] Z. Luo, A. Mishra, A. Achkar, J. Eichel, S. Li, and P.-M. Jodoin, "Non-local deep features for salient object detection," in *CVPR*, 2017.

- [155] Q. Hou, M.-M. Cheng, X.-W. Hu, A. Borji, Z. Tu, and P. Torr, “Deeply supervised salient object detection with short connections,” *arXiv:1611.04849*, 2016.
- [156] Q. Yan, L. Xu, J. Shi, and J. Jia, “Hierarchical saliency detection,” in *CVPR*, 2013.
- [157] Y. Li, X. Hou, C. Koch, J. M. Rehg, and A. L. Yuille, “The secrets of salient object segmentation,” in *CVPR*, 2014.
- [158] V. Movahedi and J. H. Elder, “Design and perceptual validation of performance measures for salient object segmentation,” in *CVPRW*, 2010.
- [159] A. Borji, M.-M. Cheng, H. Jiang, and J. Li, “Salient object detection: A benchmark,” *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5706–5722, 2015.
- [160] C. Peng, X. Gao, N. Wang, and J. Li, “Graphical representation for heterogeneous face recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 2, pp. 301–312, 2015.
- [161] C. Peng, N. Wang, X. Gao, and J. Li, “Face recognition from multiple stylistic sketches: Scenarios, datasets, and evaluation,” in *ECCV*, 2016.
- [162] X. Gao, N. Wang, D. Tao, and X. Li, “Face sketchphoto synthesis and retrieval using sparse representation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 8, pp. 1213–1226, 2012.
- [163] N. Wang, D. Tao, X. Gao, X. Li, and J. Li, “A comprehensive survey to face hallucination,” *Int. J. of Comput. Vision*, vol. 106, no. 1, pp. 9–30, 2014.
- [164] C. Peng, X. Gao, N. Wang, D. Tao, X. Li, and J. Li, “Multiple representations-based face sketch-photo synthesis,” *IEEE Trans. Neural Netw. & Learning Syst.*, vol. 27, no. 11, pp. 2201–2215, 2016.
- [165] A. Majumder, L. Behera, and V. K. Subramanian, “Automatic facial expression recognition system using deep network-based data fusion,” *IEEE Trans. Cybern.*, vol. 48, pp. 103–114, 2018.
- [166] P. Viola and M. Jones, “Robust real-time face detection,” *Int. J. of Comput. Vision*, vol. 57, no. 2, pp. 137–154, 2004.
- [167] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang, “Unitbox: An advanced object detection network,” in *ACM MM*, 2016.
- [168] S. S. Farfade, M. J. Saberian, and L.-J. Li, “Multi-view face detection using deep convolutional neural networks,” in *ICMR*, 2015.
- [169] S. Yang, P. Luo, C.-C. Loy, and X. Tang, “From facial parts responses to face detection: A deep learning approach,” in *ICCV*, 2015.
- [170] S. Yang, Y. Xiong, C. C. Loy, and X. Tang, “Face detection through scale-friendly deep convolutional networks,” in *CVPR*, 2017.
- [171] Z. Hao, Y. Liu, H. Qin, J. Yan, X. Li, and X. Hu, “Scale-aware face detection,” in *CVPR*, 2017.
- [172] H. Wang, Z. Li, X. Ji, and Y. Wang, “Face r-cnn,” *arXiv:1706.01061*, 2017.
- [173] X. Sun, P. Wu, and S. C. Hoi, “Face detection using deep learning: An improved faster rcnn approach,” *arXiv:1701.08289*, 2017.
- [174] L. Huang, Y. Yang, Y. Deng, and Y. Yu, “Densebox: Unifying landmark localization with end to end object detection,” *arXiv:1509.04874*, 2015.
- [175] Y. Li, B. Sun, T. Wu, and Y. Wang, “face detection with end-to-end integration of a convnet and a 3d model,” in *ECCV*, 2016.
- [176] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, “Joint face detection and alignment using multitask cascaded convolutional networks,” *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, 2016.
- [177] I. A. Kalinovsky and V. G. Spitsyn, “Compact convolutional neural network cascadefor face detection,” in *CEUR Workshop*, 2016.
- [178] H. Qin, J. Yan, X. Li, and X. Hu, “Joint training of cascaded cnn for face detection,” in *CVPR*, 2016.
- [179] V. Jain and E. Learned-Miller, “Fddb: A benchmark for face detection in unconstrained settings,” Tech. Rep., 2010.
- [180] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection,” in *CVPR*, 2015.
- [181] B. Yang, J. Yan, Z. Lei, and S. Z. Li, “Aggregate channel features for multi-view face detection,” in *IJCB*, 2014.
- [182] N. Markuš, M. Frljak, I. S. Pandžić, J. Ahlberg, and R. Forchheimer, “Object detection with pixel intensity comparisons organized in decision trees,” *arXiv:1305.4537*, 2013.
- [183] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool, “Face detection without bells and whistles,” in *ECCV*, 2014.
- [184] J. Li and Y. Zhang, “Learning surf cascade for fast and accurate object detection,” in *CVPR*, 2013.
- [185] S. Liao, A. K. Jain, and S. Z. Li, “A fast and accurate unconstrained face detector,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 211–223, 2016.
- [186] B. Yang, J. Yan, Z. Lei, and S. Z. Li, “Convolutional channel features,” in *ICCV*, 2015.
- [187] R. Ranjan, V. M. Patel, and R. Chellappa, “Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition,” *arXiv:1603.01249*, 2016.
- [188] P. Hu and D. Ramanan, “Finding tiny faces,” in *CVPR*, 2017.
- [189] Z. Jiang and D. Q. Huynh, “Multiple pedestrian tracking from monocular videos in an interacting multiple model framework,” *IEEE Trans. Image Process.*, vol. 27, pp. 1361–1375, 2018.
- [190] D. Gavrila and S. Munder, “Multi-cue pedestrian detection and tracking from a moving vehicle,” *Int. J. of Comput. Vision*, vol. 73, pp. 41–59, 2006.
- [191] S. Xu, Y. Cheng, K. Gu, Y. Yang, S. Chang, and P. Zhou, “Jointly attentive spatial-temporal pooling networks for video-based person re-identification,” in *ICCV*, 2017.
- [192] Z. Liu, D. Wang, and H. Lu, “Stepwise metric promotion for unsupervised video person re-identification,” in *ICCV*, 2017.
- [193] A. Khan, B. Rinner, and A. Cavallaro, “Cooperative robots to observe moving targets: Review,” *IEEE Trans. Cybern.*, vol. 48, pp. 187–198, 2018.
- [194] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *Int. J. of Robotics Res.*, vol. 32, pp. 1231–1237, 2013.
- [195] Z. Cai, M. Saberian, and N. Vasconcelos, “Learning complexity-aware cascades for deep pedestrian detection,” in *ICCV*, 2015.
- [196] Y. Tian, P. Luo, X. Wang, and X. Tang, “Deep learning strong parts for pedestrian detection,” in *CVPR*, 2015.
- [197] P. Dollár, R. Appel, S. Belongie, and P. Perona, “Fast feature pyramids for object detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, 2014.
- [198] S. Zhang, R. Benenson, and B. Schiele, “Filtered channel features for pedestrian detection,” in *CVPR*, 2015.
- [199] S. Paisitkriangkrai, C. Shen, and A. van den Hengel, “Pedestrian detection with spatially pooled features and structured ensemble learning,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, pp. 1243–1257, 2016.
- [200] L. Lin, X. Wang, W. Yang, and J.-H. Lai, “Discriminatively trained and-or graph models for object shape detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 5, pp. 959–972, 2015.
- [201] M. Mathias, R. Benenson, R. Timofte, and L. Van Gool, “Handling occlusions with franken-classifiers,” in *ICCV*, 2013.
- [202] S. Tang, M. Andriluka, and B. Schiele, “Detection and tracking of occluded people,” *Int. J. of Comput. Vision*, vol. 110, pp. 58–69, 2014.
- [203] L. Zhang, L. Lin, X. Liang, and K. He, “Is faster r-cnn doing well for pedestrian detection?” in *ECCV*, 2016.
- [204] Y. Tian, P. Luo, X. Wang, and X. Tang, “Deep learning strong parts for pedestrian detection,” in *ICCV*, 2015.
- [205] J. Liu, S. Zhang, S. Wang, and D. N. Metaxas, “Multispectral deep neural networks for pedestrian detection,” *arXiv:1611.02644*, 2016.
- [206] Y. Tian, P. Luo, X. Wang, and X. Tang, “Pedestrian detection aided by deep learning semantic tasks,” in *CVPR*, 2015.
- [207] X. Du, M. El-Khamy, J. Lee, and L. Davis, “Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection,” in *WACV*, 2017.
- [208] Q. Hu, P. Wang, C. Shen, A. van den Hengel, and F. Porikli, “Pushing the limits of deep cnns for pedestrian detection,” *IEEE Trans. Circuits Syst. Video Technol.*, 2017.
- [209] D. Tomé, L. Bondi, L. Baroffio, S. Tubaro, E. Plebani, and D. Pau, “Reduced memory region based deep convolutional neural network detection,” in *ICCE-Berlin*, 2016.
- [210] J. Hosang, M. Omran, R. Benenson, and B. Schiele, “Taking a deeper look at pedestrians,” in *CVPR*, 2015.
- [211] J. Li, X. Liang, S. Shen, T. Xu, J. Feng, and S. Yan, “Scale-aware fast r-cnn for pedestrian detection,” *arXiv:1510.08160*, 2015.
- [212] Y. Gao, M. Wang, Z.-J. Zha, J. Shen, X. Li, and X. Wu, “Visual-textual joint relevance learning for tag-based social image search,” *IEEE Trans. Image Process.*, vol. 22, no. 1, pp. 363–376, 2013.
- [213] T. Kong, F. Sun, A. Yao, H. Liu, M. Lv, and Y. Chen, “Ron: Reverse connection with objectness prior networks for object detection,” in *CVPR*, 2017.
- [214] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, “Generative adversarial nets,” in *NIPS*, 2014.
- [215] Y. Fang, K. Kuan, J. Lin, C. Tan, and V. Chandrasekhar, “Object detection meets knowledge graphs,” in *IJCAI*, 2017.
- [216] S. Wellock, J. Mao, K. Cho, and Z. Zhang, “Saliency-based sequential image attention with multiset prediction,” in *NIPS*, 2017.
- [217] S. Azadi, J. Feng, and T. Darrell, “Learning detection with diverse proposals,” in *CVPR*, 2017.

- [218] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *NIPS*, 2015.
- [219] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *NIPS*, 2017.
- [220] B. Yang, J. Yan, Z. Lei, and S. Z. Li, "Craft objects from images," in *CVPR*, 2016.
- [221] I. Croitoru, S.-V. Bogolin, and M. Leordeanu, "Unsupervised learning from video to detect foreground objects in single images," in *ICCV*, 2017.
- [222] C. Wang, W. Ren, K. Huang, and T. Tan, "Weakly supervised object localization with latent category learning," in *ECCV*, 2014.
- [223] D. P. Papadopoulos, J. R. R. Uijlings, F. Keller, and V. Ferrari, "Training object class detectors with click supervision," in *CVPR*, 2017.
- [224] J. Huang, V. Rathod, C. Sun, M. Zhu, A. K. Balan, A. Fathi, I. Fischer, Z. Wojna, Y. S. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017.
- [225] Q. Li, S. Jin, and J. Yan, "Mimicking very efficient network for object detection," in *CVPR*, 2017.
- [226] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Comput. Sci.*, vol. 14, no. 7, pp. 38–39, 2015.
- [227] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *Comput. Sci.*, 2014.
- [228] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *NIPS*, 2015.
- [229] J. Dong, X. Fei, and S. Soatto, "Visual-inertial-semantic scene representation for 3d object detection," in *CVPR*, 2017.
- [230] K. Kang, H. Li, T. Xiao, W. Ouyang, J. Yan, X. Liu, and X. Wang, "Object detection in videos with tubelet proposal networks," in *CVPR*, 2017.



Xindong Wu is an Alfred and Helen Lamson Endowed Professor in Computer Science, University of Louisiana at Lafayette (USA), and a Fellow of the IEEE and the AAAS. He received his Ph.D. degree in Artificial Intelligence from the University of Edinburgh, Britain. His research interests include data mining, knowledge-based systems, and Web information exploration. He is the Steering Committee Chair of the IEEE International Conference on Data Mining (ICDM), the Editor-in-Chief of Knowledge and Information Systems (KAIS, by Springer), and a Series Editor of the Springer Book Series on Advanced Information and Knowledge Processing (AI&KP). He was the Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering (TKDE, by the IEEE Computer Society) between 2005 and 2008.



Zhong-Qiu Zhao is a professor at Hefei University of Technology, China. He obtained the Ph.D. degree in Pattern Recognition & Intelligent System at University of Science and Technology, China, in 2007. From April 2008 to November 2009, he held a postdoctoral position in image processing in CNRS UMR6168 Lab Sciences de l'Information et des Systèmes, France. From January 2013 to December 2014, he held a research fellow position in image processing at the Department of Computer Science of Hongkong Baptist University, Hongkong, China.

His research is about pattern recognition, image processing, and computer vision.



Peng Zheng is a Ph.D. candidate at Hefei University of Technology since 2010. He received his Bachelor's degree in 2010 from Hefei University of Technology. His interests cover pattern recognition, image processing and computer vision.



Shou-tao Xu is a Master student at Hefei University of Technology. His research interests cover pattern recognition, image processing, deep learning and computer vision.

Chapter 10

Object Detection – YOLO

In the ever-evolving landscape of deep learning, object detection stands out as a crucial task with wide-ranging applications, from autonomous vehicles to surveillance systems. In this chapter, we delve into the intricacies of one of the most influential and efficient object detection algorithms—You Only Look Once (YOLO). YOLO revolutionized the field by introducing a real-time, one-pass approach to detecting objects in images, streamlining the detection process significantly. This chapter explores the underlying principles of YOLO, detailing its unique architecture and the principles that make it stand apart. From its inception (YOLO v1) to a few improved versions (YOLO v2 and YOLO v3), we will guide you through the evolution of YOLO, providing insights into its strengths, limitations, and practical implementations.

After completing this chapter, one should be able to

- Understand the basic concepts of object detection in computer vision.
- Know the details about the architectures of YOLO versions (YOLO1, YOLO2, YOLO3)
- Understand the loss function of YOLO.
- Implement a YOLO model from scratch in PyTorch.
- Train a YOLO model.

10.1 Introduction

The task of image classification is to predict an image as one of the pre-defined categories, for instance, to classify the picture as “car” or “non-car”. The task of object detection is to deal with the situation where multiple objects, possibly in different categories (e.g., cars, pedestrians, etc.), may be present in one image. The goal of the detection is to find objects of interest in the image. As a result of object detection, each detected object is labeled with a bounding box. An example is shown in Fig.10.1.

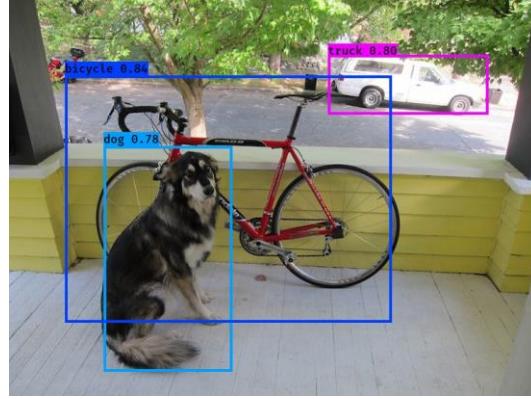


Fig.10.1 An example of object detection (generated by YOLO v2)

A *bounding box* is a rectangle that surrounds an object, that specifies its location, class (e.g., dog, bicycle) and confidence (how likely the object is present in the box). For example, the bounding box for the dog in Fig.10.1 shows the location of the dog and a confidence of 0.78. We can numerically specify the rectangle either by 1) the coordinates of its top-left corner and bottom right corner; or 2) the coordinate of its center and its width and height. In our context, we adopt the second one: $[x, y, h, w]$, where the center of the bounding box is at (x, y) , and the height and width of bounding box are h and w , respectively. The coordinate of the up-left corner of the image is $(0,0)$ while the coordinate of the bottom right corner is $(1,1)$.

10.2 YOLO (v1)

YOLO (You Only Look Once) (v1) was proposed by (Redmon, J. *et al.*, 2015), which achieved object detection with a speed of 150 FPS (frames per second) in video streaming and a good mean average precision (mAP) (63%) on PASCAL VOC 2007 dataset (see the definition of mAP in Section 10.6). Later, YOLO (v2) and YOLO 9000 were proposed by (Redmon, J. *et al.*, 2016), which at 67 FPS gave mAP of 76.8% on VOC 2007 dataset. Two years later, Furthermore, in 2018, YOLO (v3) (Redmon, J. *et al.*, 2018) was released with further improved object detection accuracy and speed. YOLO (v3) introduced a new backbone architecture, called Darknet-53, which improved feature extraction and added additional anchor boxes to better detect objects at different scales. After YOLO (v3), a few groups developed different versions of YOLO for further improvements, up to YOLO (v8) in 2023. In this chapter, we will explore the details of YOLO v1 to v3.

The generic architecture of different YOLO versions can be illustrated in Fig.10.2. In general, a YOLO architecture consists of three parts: backbone, neck, and head. The backbone is a deep convolutional neural network that acts as a feature extractor. The backbone is usually pre-trained on a classification image dataset (e.g., ImageNet). The neck is a feature collector that collects feature maps from different stages of the backbone. The purpose of the neck is to enhance the semantic representation and richness of features extracted for objects of various shapes and sizes. The head is used for loss calculation during training and prediction during inference. Non-max suppression is an algorithm to select the best bounding box for an object and reject redundant bounding boxes.

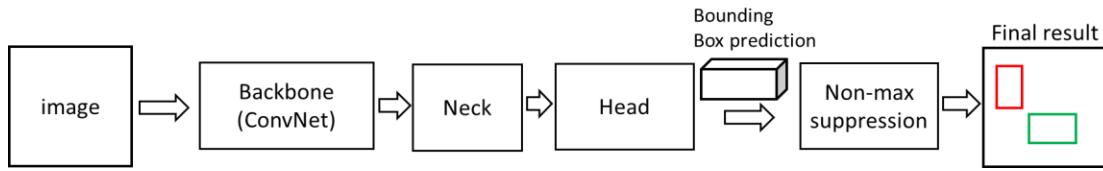


Fig.10.2 the generic architecture of YOLO

Table 10.1 YOLO (v1) architecture (input image size: $448 \times 448 \times 3$)

Layers	filters	Size/stride	output	notes
Conv	64	$7 \times 7/2$	$224 \times 224 \times 64$	backbone
Maxpool		$2 \times 2/2$	$112 \times 112 \times 64$	
Conv	192	$3 \times 3/1$	$112 \times 112 \times 192$	
Maxpool		$2 \times 2/2$	$56 \times 56 \times 192$	
Conv	128	$1 \times 1/1$	$56 \times 56 \times 128$	
Conv	256	$3 \times 3/1$	$56 \times 56 \times 256$	
Conv	256	$1 \times 1/1$	$56 \times 56 \times 256$	
Conv	512	$3 \times 3/1$	$56 \times 56 \times 512$	
Maxpool		$2 \times 2/2$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	512	$1 \times 1/1$	$28 \times 28 \times 512$	
Conv	1024	$3 \times 3/1$	$28 \times 28 \times 1024$	
Maxpool		$2 \times 2/2$	$14 \times 14 \times 1024$	
Conv	512	$1 \times 1/1$	$14 \times 14 \times 512$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	512	$1 \times 1/1$	$14 \times 14 \times 512$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	1024	$3 \times 3/2$	$7 \times 7 \times 1024$	
Conv	1024	$3 \times 3/1$	$7 \times 7 \times 1024$	neck
Conv	1024	$3 \times 3/1$	$7 \times 7 \times 1024$	
FC			4096	
FC			1470	head
reshape			$7 \times 7 \times 30$	

10.2.1 Architecture of YOLO v1

YOLO v1 has 24 convolutional layers followed by 2 fully connected layers (FC), as specified in Table 10.1. Leaky ReLU activation is used for all layers except the final layer that uses a linear activation function. The last convolutional layer generates a tensor with a shape $(7, 7, 1024)$, shown in Fig.10.3. This tensor is then flattened to feed the subsequent fully connected layer. The last fully connected layer generates the final prediction tensor with a shape $(7, 7, 30)$, i.e., 2 bounding box predictions per location, given 20 classes in the dataset, shown in Fig.10.4.

The backbone is composed of 24 convolutional layers and 4 Maxpooling layers. It generates a feature map tensor with a shape of $(7,7,1024)$, from an input image of $(448,448,3)$. Each cell in the feature map tensor, with a shape of $(1,1,1024)$, is the extracted feature vector corresponding to one grid cell in the image, illustrated in Fig.10.3.

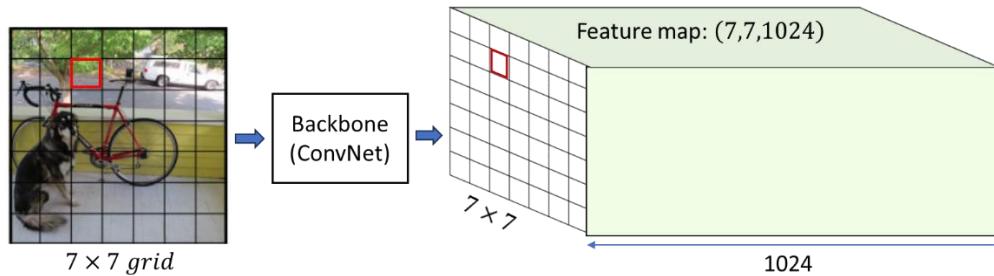


Fig.10.3 the backbone extracts one feature vector for each cell in the image grid (7×7).

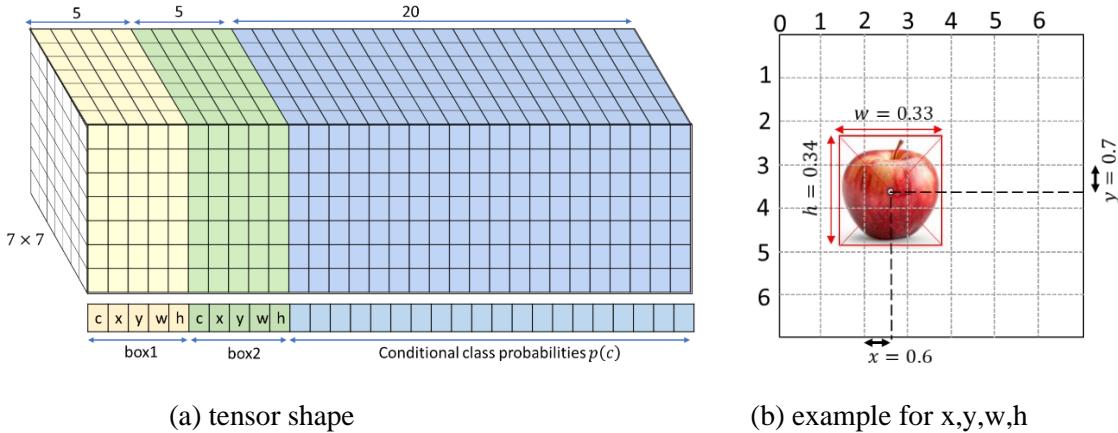


Fig.10.4 the prediction tensor of YOLO v1.

The feature map, output by the backbone convolutional neural network, is then passed through two fully connected layers, which delivers the bounding box prediction on each grid cell. The prediction is a tensor of $(7,7,30)$, illustrated in Fig.10.4 (a). The prediction tensor can be viewed as 49 vectors, and each vector has 30 elements. Each vector is responsible for predicting two bounding boxes for one grid cell of the image. Each bounding box is specified by five parameters: confidence (c) (or objectness score), box center (x,y) and its size (w,h). The confidence score reflects how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. If no object exists in the cell, the confidence should be zero. If an object is predicted,

the confidence represents the IOU between the predicted box and any ground truth box. The coordinates (x, y) represent the center of the box relative to the bounds of the grid cell. The width (w) and height (h) are normalized with respect to the whole image. Thus, $x, y, w, h \in [0,1]$. Fig.10.4 (b) illustrates an example in which an apple is detected in the grid cell (2,3) (i.e., column 2 and row 3).

Each cell also predicts a set of class probabilities, conditioned on the grid cell containing an object, $p(c|object)$, $c=1,2,\dots,20$, where 20 is the total number of classes.

At test time, the class-specific confidence scores for each box are the products of the conditional class probability and the box confidence. Note that YOLO v1 predicts only one object per grid cell.

For a convenience of further discussions on different YOLO versions, we denote grid size as $S \times S$, B as the number of bounding boxes per grid cell, and C as the number of classes. Thus, the shape of prediction tensor for YOLO v1 is $S \times S \times (5 \times B + C) = 7 \times 7 \times (5 \times 2 + 20)$.

10.2.2 Training and loss function

Before training the entire architecture of YOLO v1, the authors pretrained the first 20 convolutional layers followed by an average-pooling layer and a fully connected layer, for classification on ImageNet 1000 classes, with input size of 224×224 , achieving a top-5 accuracy of 88%.

Then, they convert the pretrained architecture for object detection by discarding the average pool and the fully connected layer and adding four convolutional layers and two fully connected layers with randomly initialized weights. Thus, the resulting architecture is specified in Table 10.1, with the first 20 convolutional layers pretrained.

The model is further trained for object detection on datasets from PASCAL VOC 2007 and 2012, based on the loss function (discussed below), with the input resolution increased to 448×448 . It is essential to understand the loss function used for YOLO training. Object detection can be treated as a regression problem of target area prediction and category prediction. Specifically, the loss function of YOLO v1 can be divided into three parts: localization loss, confidence loss, and classification loss.

YOLO v1 predicts two bounding boxes per grid cell. For training purposes, we only want one bounding box predictor to be responsible for each object. We assign one box to be “**responsible**” for predicting an object based on which box has the highest current IOU (i.e., Intersection Over Union) with the ground truth. The IOU between two boxes is defined as the ratio of their intersection area to their union area. IOU with a value “1” implies a perfect match while a small value close to zero means a small overlap. Thus, IOU can be used to measure how close the predicted bounding box is to the ground truth bounding box, and also can be applied to detect the redundant bounding boxes for one object in non-max suppression (discussed later).

$$\text{IOU} = \frac{\text{Intersection Area}}{\text{Union Area}} \quad (10.1)$$

A) Localization loss

The localization loss measures the errors in the predicted boundary box locations and sizes. It only penalizes the bounding box localization error if that box is “responsible” for the ground truth box (i.e., the box has the highest IOU with the ground truth box in that grid cell).

$$\begin{aligned} \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} & \left[\left(x_i - \hat{x}_i \right)^2 + \left(y_i - \hat{y}_i \right)^2 \right] \\ + \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} & \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \end{aligned} \quad (10.2)$$

where

$\mathbb{I}_{ij}^{obj} = 1$ if the j -th bounding box in cell i is responsible for detecting the object, otherwise 0. In other words, localization error is added to loss only for the responsible boxes.

The hat over a variable indicates that the variable is a prediction (e.g., \hat{x}_i). A variable without a hat is a ground truth (e.g., x_i).

λ_{coord} (Default is set to 5) increases the weight for the loss in bounding box coordinates.

Sum-squared error equally weights errors in large boxes and small boxes. The desired loss should reflect that the same deviation in large boxes matters less than in small boxes. To partially address this, we predict the square root of the bounding box width and height instead of the width and height directly.

B) Confidence loss

For the boxes that are responsible for detecting an object, the confidence loss is

$$\sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 \quad (10.3a)$$

where

\hat{C}_i is the box confidence score for the box j in cell i . Note that the ground truth box confidence score C_i for the box with an object is defined to be IOU_{pred}^{truth} by the original paper. However, the ground truth confidence score is simply set to 1 in many implementation examples.

For the boxes that are not responsible for detecting an object, the confidence loss is

$$\lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \quad (10.3b)$$

where

\mathbb{I}_{ij}^{noobj} is the complement of \mathbb{I}_{ij}^{obj} , i.e., that the box j in grid cell i is not responsible for detection.

\hat{C}_i is the box confidence score for the box j in cell i . Note that the ground truth box confidence score C_i for the box without an object is defined to be 0.

λ_{noobj} (=0.5 by default) decreases the loss from confidence predictions for boxes that don't contain objects. Since most boxes do not contain any objects, we train the model to detect background more frequently than detecting objects. To remedy this imbalance, λ_{noobj} is used to limit the loss from background detections.

C) Classification loss

If an object appears in cell i , the sum-squared error of the predicted class conditional probabilities in cell i is counted for the loss function,

$$\sum_{i=1}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (10.4)$$

where

$\mathbb{I}_i^{obj}=1$ if an object appears in cell i . This implies that $p_i(c)$ is conditional.

$\hat{p}_i(c)$ denotes the predicted conditional class probability for class c in cell i . Given that the ground truth object belongs to a class, the ground truth probability is 1.0 for that class, and 0.0 for all others.

D) Total loss

The total loss function is given by the sum of (10.2), (10.3a), (10.3b) and (10.4), i.e.,

$$\begin{aligned} Loss = & \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} \left[\left(x_i - \hat{x}_i \right)^2 + \left(y_i - \hat{y}_i \right)^2 \right] \\ & + \lambda_{coord} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{noobj} \sum_{i=1}^{S^2} \sum_{j=1}^B \mathbb{I}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \\ & + \sum_{i=1}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2 \end{aligned} \quad (10.5)$$

Note that the loss function penalizes classification error only if an object is present in that grid cell. It also only penalizes bounding box coordinate error if that box is responsible for the ground truth box. It penalizes the confidence score errors for both object-present boxes and non-object-present boxes, but with different weights (non-object-present boxes have less weight).

The model, with the first 20 convolutional layer pretrained, was further trained for object detection on datasets PASCAL VOC 2007 and 2012, based on the loss function (10.5). The detailed training settings and tricks can be found in the original paper ([Redmon, J. et al., 2015](#)).

10.2.3 Inference and non-maximal suppression (NMS)

In inference, the YOLO v1 network predicts confidence score, center coordinates, width and height for each bounding box and conditional class probabilities for each grid cell, as shown in Fig.10.4. There are totally 98 bounding boxes. Finally, non-max suppression (NMS) is performed for each class separately to discard the redundant detected boxes while keeping the best one for a detected object.

Consider an example in Fig.10.5. We assume that only one class (e.g., car) is considered in our discussion for simplicity. In this example five bounding boxes have confidence scores above a pre-defined threshold (e.g., 0.6). Obviously, three of them are associated with one car while the other two bounding boxes with another car, as shown in Fig.10.5 (left). After non-max suppression, only two best bounding boxes, one per car, should be kept, as shown in Fig.10.5 (right). The NMS algorithm rejects those bounding boxes which are too close to a high confident bounding box.

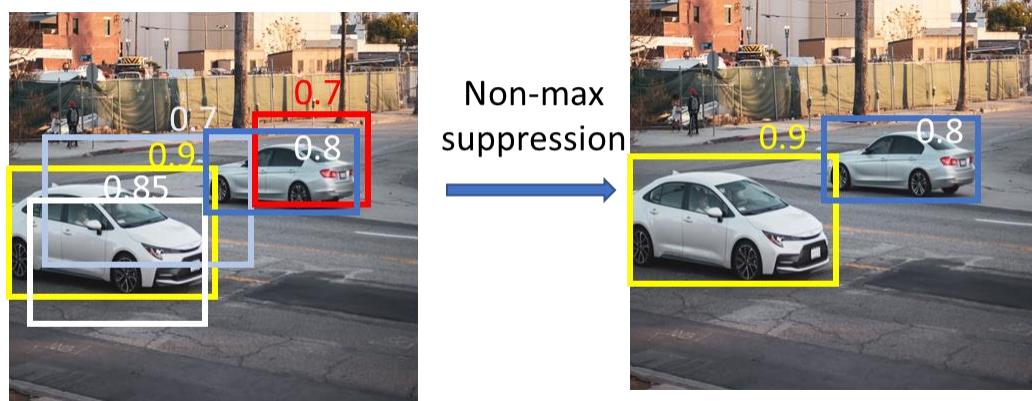


Fig.10.5 Non-max suppression (the number associated with each box is confidence score).

The non-max suppression algorithm can be described below.

Algorithm: non-max suppression

Given: the output tensor (i.e., a list of bounding boxes), a pre-defined threshold for confidence score, a pre-defined threshold for IOU.

Output: a final list that consists of qualified bounding boxes.

Step 1: Remove any bounding box with confidence less than a threshold (e.g., 0.6). Put the remaining boxes into a list, called *remain_list*, in descending order of confidence score.

Step 2: Do the following by looping over *remain_list* until *remain_list* is empty:

1. From *remain_list*, move the first box (i.e., one with the highest confidence), b_{max} , to *final_list*.
2. From *remain_list*, delete any remaining box (for the same class as b_{max}) with $IOU \geq$ a threshold (e.g. 0.5) with the output box, b_{max} .

10.3 YOLO (v2)

YOLO v2 (or YOLO9000) is the second version of the YOLO with the objective of improving the accuracy significantly while making it faster. Compared to YOLO v1, YOLO v2 makes the improvements through the following aspects:

- Batch normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization (e.g., dropout). Batch normalization is added to all convolutional layers in YOLO v2, leading to 2% mAP improvement.
- Higher resolution classifier. YOLO v1 was pre-trained for classification network by images with resolution 224×224 , and then for detection by images with resolution 448×448 . In YOLO v2, the classifier is fine-tuned by the resolution 448×448 , which leads to 4% mAP improvement.
- Architecture for fine-grained features and more bounding boxes per grid cell.
- Anchor boxes. YOLO v2 doesn't predict the bounding box parameters directly, instead it predicts the pre-defined anchor boxes.
- Multi-scale training.

YOLO v2 gives state-of-the-art detection accuracy on PASCAL VOC and COCO. It can run on varying sizes offering a tradeoff between speed and accuracy. At 67 FPS, YOLO v2 can give an accuracy of 76.8 mAP while at 40 FPS the detector gives an accuracy of 78.6 mAP, on VOC dataset.

10.3.3 Architecture of YOLO v2

The overall architecture of YOLO v2 is described in Table 10.2. YOLO v2 adopts a model, called Darknet-19, for its backbone network.

Darknet-19 has 19 convolutional layers and 5 maxpooling layers. Inspired by GoogleNet and Network-in-Network, we use 3×3 filters and 1×1 filters alternatively. The 1×1 filters compress the feature representation between 3×3 convolutional layers. The number of channels is doubled while the feature map size is halved, by each maxpooling layer. This results in an output feature map size 13×13 for an input image 416×416 after 5 maxpooling layers (note: $\frac{416}{2^5} = 13$).

In Darknet-19, the last convolutional layer $1 \times 1 \times 1000$ and the avgpool layer are only used to train the backbone on ImageNet 1000 dataset. Please note that it was initially trained on images 224×224 , and then fine-tuned on images 448×448 .

To construct YOLO v2 network for object detection, we modify the pretrained Darknet-19 by removing the last convolutional layer and the avgpool layer, and adding 3 convolutional layers with filter 3×3 and 1024 channels per layer and a final 1×1 convolutional layer with 125 channels, as described in Table 10.2. The 125 channels are responsible for 5 bounding boxes. In addition, the output from the final $3 \times 3 \times 512$ layer is reshaped to $13 \times 13 \times 2048$, and concatenated with the output of the third to the last convolutional layer so that the model can use fine grain features, as shown in Fig.10.6.

Table 10.2 YOLO v2 architecture (input image size: $416 \times 416 \times 3$)

Layers	filters	Size/stride	Output				
Conv	32	3×3	416×416				
Maxpool		$2 \times 2/2$	208×208				
Conv	64	3×3	208×208				
Maxpool		$2 \times 2/2$	104×104				
Conv	128	3×3	104×104				
Conv	64	1×1	104×104				
Conv	128	3×3	104×104				
Maxpool		$2 \times 2/2$	52×52				
Conv	256	3×3	52×52				
Conv	128	1×1	52×52				
Conv	256	3×3	52×52				
Maxpool		$2 \times 2/2$	26×26				
Conv	512	3×3	26×26				
Conv	256	1×1	26×26				
Conv	512	3×3	26×26				
Conv	256	1×1	26×26				
Conv	512	3×3	26×26				
Maxpool		$2 \times 2/2$	13×13				
Conv	1024	3×3	13×13				
Conv	512	1×1	13×13				
Conv	1024	3×3	13×13				
Conv	512	1×1	13×13				
Conv	1024	3×3	13×13				
For backbone pre-training				For detection			
Layers	Filters	Size/stride	output	layers	filters	Size/stride	output
Conv	1000	1×1	13×13	Conv	1024	3×3	13×13
Avgpool		Global	1000	Conv	1024	3×3	13×13
Softmax				Concat	3072		13×13
				Conv	1024	3×3	13×13
				Conv	125	1×1	13×13

The YOLO v2 model was trained on COCO and VOC datasets. For an input image 416×416 , YOLO v2 divides it into an 13×13 grid and predicts 5 bounding boxes per grid cell. Each bounding box prediction includes 25 elements: 1 for confidence score, 4 for coordinates, and 20 for conditional class probabilities of 20 classes. Thus, the prediction tensor of YOLO v2 has a shape of $13 \times 13 \times 125$, as illustrated in Fig.10.7. In general, the YOLO v2 prediction tensor has a shape of $13 \times 13 \times (B \times (5 + C))$, where B is the number of bounding boxes per grid cell, C is the total number of classes.

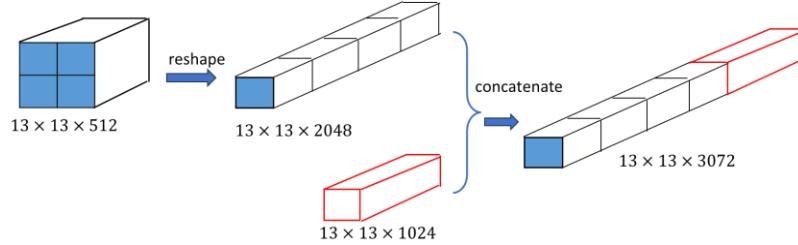


Fig.10.6 Concatenation for the passthrough layer

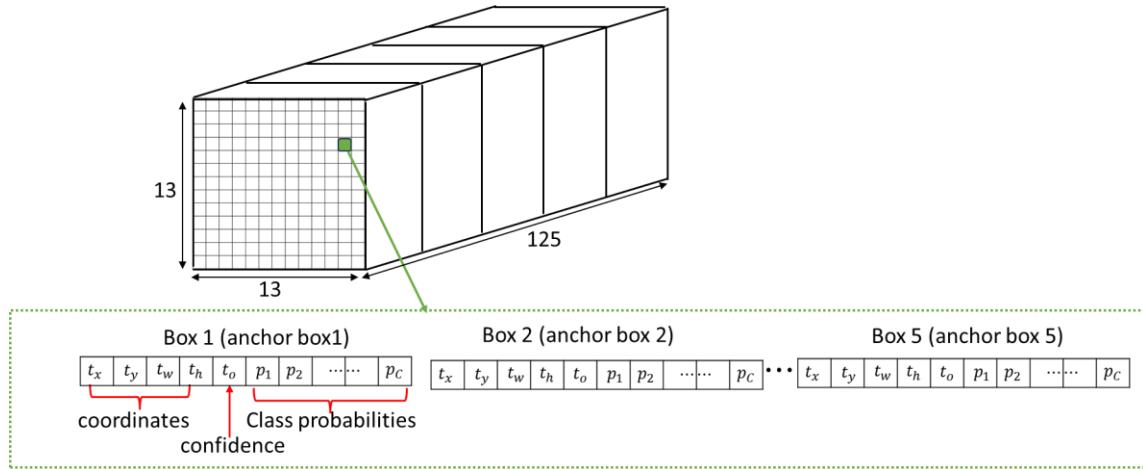


Fig.10.7 Prediction tensor of YOLO v2

10.3.2 Anchor boxes

An important ingredient of YOLO v2 is the concept of anchor boxes, which improves performance and has been applied in the subsequent YOLO models. YOLO v1 predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Instead of predicting coordinates directly, YOLO v2 predicts a bounding box relative to a pre-defined bounding box, called *anchor box*. In other words, the model uses the anchor boxes as the initial values of bounding boxes, and then learns to adjust the bounding box for fitting the detected object closely. This idea makes it easier for the model to learn. Furthermore, the use of a set of anchor boxes enables a model to detect multiple objects, objects of different scales, and overlapping objects.

The number of anchor boxes and their aspect ratios can be determined by a statistical analysis on the target datasets. For example, k -means clustering on the dimensions of objects in VOC and COCO datasets suggests that $k = 5$ (i.e., 5 different shapes for anchor boxes) give a good tradeoff between the recall and the complexity. As the result, a widely used set of anchor boxes (p_w, p_h) is given in `yolov2.cfg` (provided by YOLO v2 paper authors) as $\{(0.57273, 0.677385), (1.87446, 2.06253), (3.33843, 5.47434), (7.88282, 3.52778), (9.77052, 9.16828)\}$, drawn in Fig.10.8. Note that p_w, p_h are the width and the height, respectively, and they are normalized by the grid cell size. Alternatively, the size of bounding box or anchor box is usually normalized by the image size in YOLO algorithms. Thus, divided by 13, the above set of

anchor boxes can be specified, relative to the image size, as anchors={ (0.04405615, 0.05210654), (0.14418923, 0.15865615), (0.25680231, 0.42110308), (0.60637077, 0.27136769), (0.75157846, 0.70525231) }.

The geometric relationship between an anchor box and the corresponding bounding box is defined by eq (10.6) and illustrated in Fig.10.9.

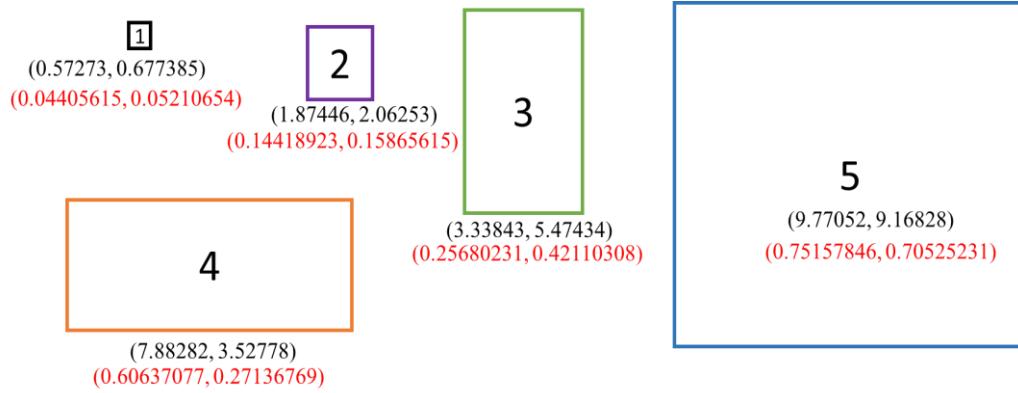


Fig.10.8 Five anchor boxes in YOLO v2

(normalized by the grid cell size in black [yolov2.cfg by YOLO v2 authors])

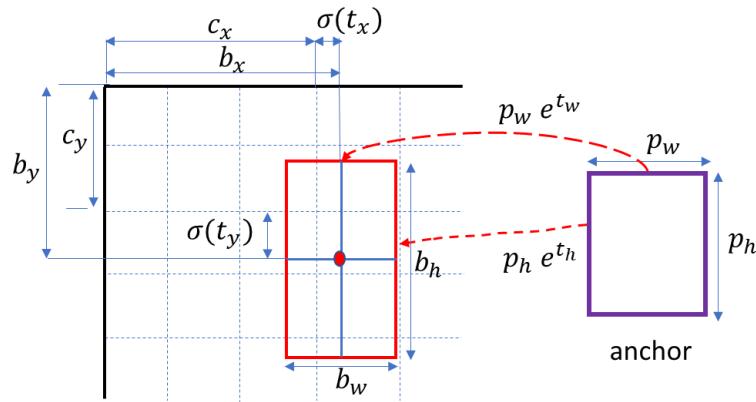


Fig.10.9 Bounding box prediction from the output tensor of YOLO v2

10.3.3 Predictions from YOLO v2

The prediction tensor ($13 \times 13 \times 125$) generated by YOLO v2 model is illustrated in Fig.10.7. To understand the predictions, we imagine that the image is divided into a 13×13 grid. For each grid cell, there is a prediction 125-element vector consisting of 5 bounding boxes, which corresponds to 5 anchor boxes, respectively. The prediction of each bounding box includes box coordinates (4 numbers), confidence (1 number) and conditional class probabilities (20 numbers).

Now let's focus on the bounding box prediction, corresponding to an anchor box (p_w, p_h) in a specific grid cell (c_x, c_y) , where $0 \leq c_x \leq 12, 0 \leq c_y \leq 12$ are the index of the grid cell, for

instance, cell(3,2) is the grid cell where the bounding box center point is located, as illustrated in Fig.10.9.

Suppose t_x, t_y, t_w, t_h, t_o are the data from the prediction tensor of YOLO v2. Then, the actual parameters of the bounding box can be obtained by the following transformation:

$$b_x = \sigma(t_x) + c_x \quad (10.6 \text{ a})$$

$$b_y = \sigma(t_y) + c_y \quad (10.6 \text{ b})$$

$$b_w = p_w e^{t_w} \quad (10.6 \text{ c})$$

$$b_h = p_h e^{t_h} \quad (10.6 \text{ d})$$

$$\text{object score} = P_r(\text{object}) \times \text{IOU}(b, \text{object}) = \sigma(t_o) \quad (10.6 \text{ e})$$

where

$\sigma(\cdot)$ is the sigmoid function.

b_x, b_y are the coordinates of the bounding box center point.

b_w, b_h are the width and the height of the bounding box.

Note that $\sigma(t_x), \sigma(t_y) \in (0,1)$ are the offsets of the bounding box center point relative to the left-top corner of the grid cell. t_w, t_h are used to scale the anchor box to get the bounding box size. Object score represents the probability that an object is contained inside a bounding box, and it is obtained by passing t_o through a sigmoid function. Note that the box position parameters such as $c_x, c_y, b_x, b_y, b_w, b_h, p_w, p_h$ are numerically normalized relative to the grid size (width or height), i.e., their units are the grid size.

The loss function defined in equation (10.5) for YOLO v1 can be applied to YOLO v2, in general. Since anchor boxes are used for bounding box prediction, the ground truth parameters used in the loss function can be obtained by the inverse transform of (10.6) based on the label bounding boxes.

10.4 YOLO (v3)

YOLO (v3) introduced a new backbone architecture, called Darknet-53, which improved feature extraction and added additional anchor boxes to better detect objects at different scales. This model features multi-scale detection, a stronger feature extraction network, and a few changes in the loss function.

10.4.1 Architecture of YOLO v3

The neural network of YOLO v3 consists of three parts: Darknet-53, neck, and detection head, as shown in Fig.10.10. Darknet-53 is used as the backbone to extract features from the input images. The combination of Darknet-53 and the up-sample neck network results in a feature pyramid network in which a feature map gradually decreases in spatial dimension but increases later again and is concatenated with previous feature maps with corresponding sizes. The different sized feature maps are then fed to a distinct detection head. Each head is implemented by a few convolutional layers and generates predictions at a different grid scale.

Our presentation is based on the following settings for YOLO v3: three scale predictions (13x13, 26x26, 52x52 for an input image 416x416, three anchor boxes ($B=3$) for each scale prediction, and the number of classes C . The prediction for each grid cell is a vector with M elements, where $M = B \times (5 + C)$. Thus, $M=75$ for $C=20$ while $M=255$ for $C=80$.

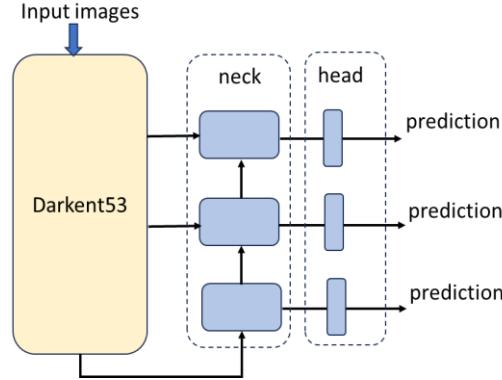


Fig.10.10 Overall architecture of YOLO v3

A) Backbone: Darknet-53

The backbone network, Darknet-53, is illustrated in Fig.10.11. Darknet-53 has 53 trainable layers (52 convolutional layers and one fully connected layer). The last three layers (avgpool, full connected, softmax) are designed for pre-training, and thus being removed when it is adopted for YOLO feature extraction. Darknet-53 was pre-trained on ImageNet dataset with the input size 256×256 . In Fig.10.11, the shape of the tensor from each block is specified as [channel, w, h] while a convolutional layer is specified as **Conv(channel, kernel size, kernel size, stride, padding)**. Note that all convolutional layers are followed by batch normalization (BN) and leakyReLU activation (L).

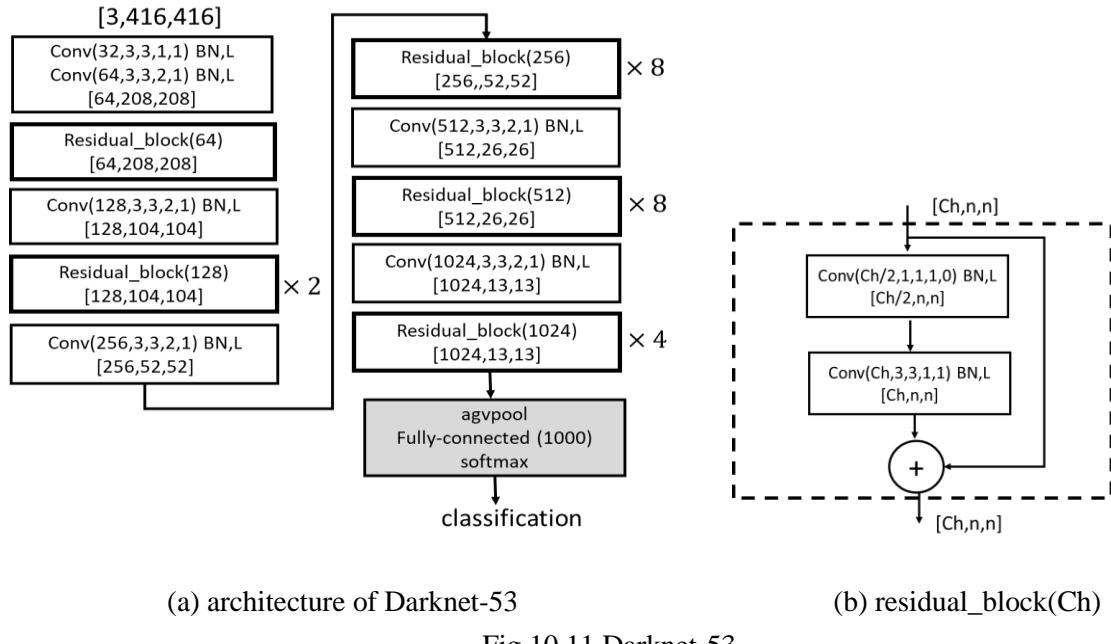


Fig.10.11 Darknet-53

Darknet-53 has 23 residual blocks. Each residual block contains one 1×1 and one 3×3 convolutional layer. At the end of each residual unit, an element-wise addition is performed between the input tensor and the output tensor from the second convolutional layer. The shape of the output tensor of a residual block is the same as the input tensor.

The down-sample step is performed by five separate convolutional layers with a stride of 2.

B) Neck: up-sample network

To detect objects at different scales, YOLO v3 adopts a Feature Pyramid Network (FPN) to extract the feature maps at different grid scales. The architecture of YOLO v3 is detailed by Fig.10.12 where we list all convolutional layers (or blocks) and the corresponding output tensor shapes, assuming that the input is a batch of images, i.e., $[N, 3, 416, 416]$. All convolutional layers, except the last Conv2d layer in each detection head, are followed by a batch normalization and leakyReLU. We count any one of the following items as one layer: convolutional layer, shortcut pass, input route from a distant layer, upsample (), concatenate (after upsample), and yolo detection. Thus, a residual block counts for 3 layers. The entire YOLO model consists of 107 layers. All layers are labeled in an order from layer 0 to layer 106.

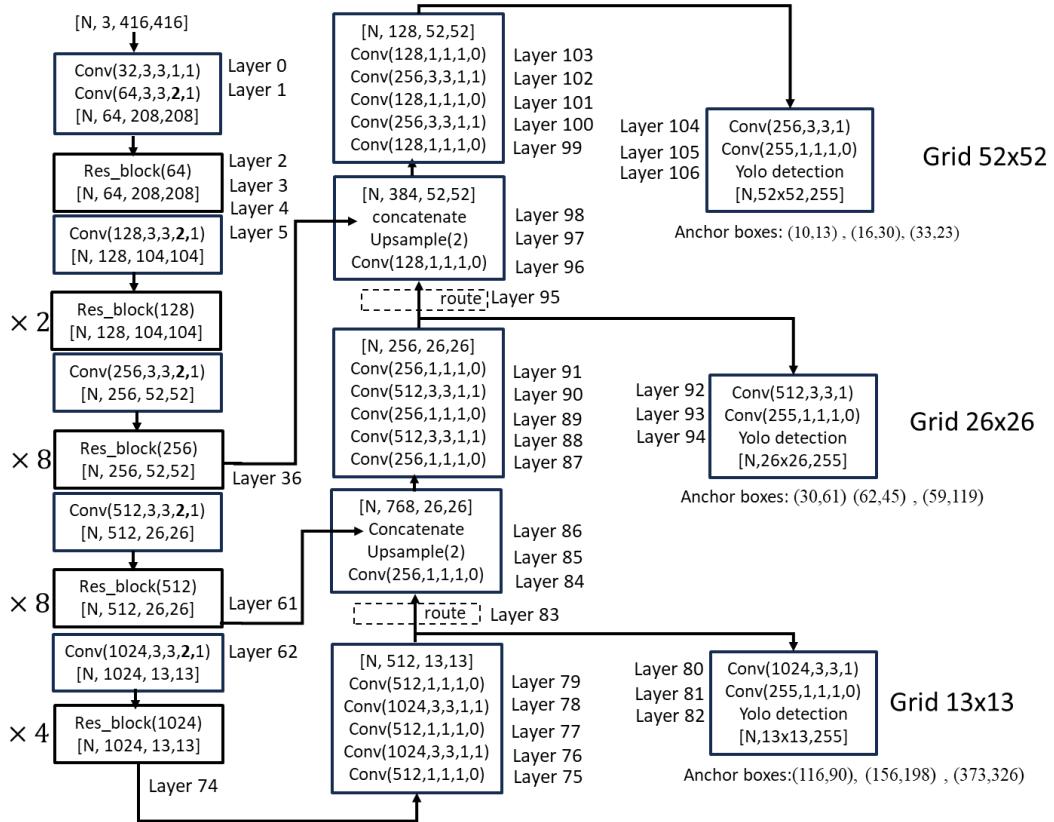


Fig.10.12 Architecture of YOLO v3 (all conv layers are followed by BN and leakyReLU, except that the last conv layer in each detection head. We define the first conv layer as layer 0 and count three layers for each Res_block).

The entire Darknet-53 down-samples the input image 416×416 by a factor of 32, and thus generates feature maps at the grid scale 13×13 . A set of convolutional layers (layers 75 to layers 82: 1×1 and 3×3 alternatively) process the feature map and generate the prediction $[N, 13, 13, M]$ at grid scale 13×13 . Note that $M=255$ when $C=80$, for instance.

Layers 83 to 94 are responsible to generate the prediction at grid scale 26×26 . First, we concatenate the feature map $[N, 512, 26, 26]$ from the last residual block (512) in Darknet-53 and the up-sampled (with a factor of 2) feature map $[N, 256, 26, 26]$ from the neck of scale 13×13 . Then the merged feature map $[N, 768, 26, 26]$ will be passed through a series of convolutional layers, which are similar to those convolutional layers used for the scale 13×13 , but the number of channels is halved. The prediction is a tensor $[N, 26, 26, M]$. The up-sample operation is illustrated in Fig.10.13.

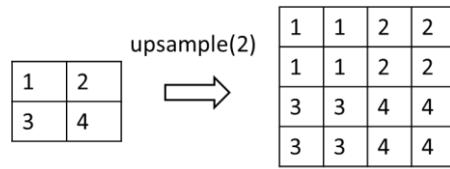


Fig.10.13 Up-sample by a factor of 2 using “nearest mode (default)”

Similarly, layers 95 to 106 are designed to predict at grid scale 52×52 . Specifically, we concatenate the feature map $[N, 256, 52, 52]$ from the last residual block (256) in Darknet-53 and the up-sampled (with a factor of 2) feature map $[N, 128, 52, 52]$ from the neck of scale 26×26 . Then the merged feature map $[N, 384, 52, 52]$ will be passed through a set of convolutional layers to generate the prediction $[N, 52, 52, M]$.

C) Understand predictions at three grid scales.

YOLO v3 generates a prediction tensor at each grid scale. The tensor shapes are $[N, 13, 13, M]$, $[N, 26, 26, M]$, and $[N, 52, 52, M]$ for three grid scales, respectively, where N is the batch size, $M = B \times (5 + C)$, B is the number of anchor boxes for one grid cell, C is the number of classes. YOLO v3 predicts three bounding boxes at each grid cell across all different scales (i.e., $B=3$). As an example, Fig.10.14 shows the prediction tensor at the scale 13×13 with $C=80$.

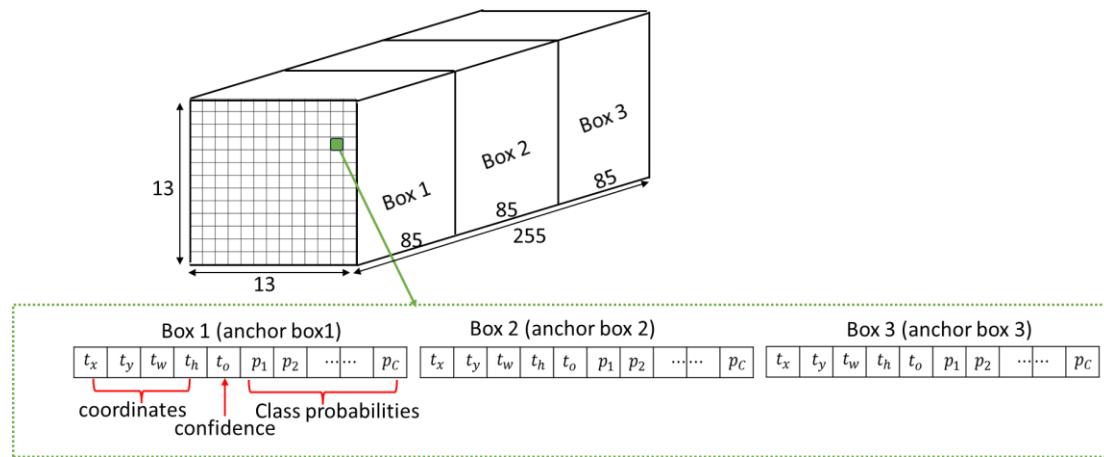


Fig.10.14 Prediction tensor at scale 13×13 for $C=80$.

To determine the anchor boxes, YOLO v3 applies k -means clustering on the COCO dataset and select 9 clusters. As a result, the width and height of the nine anchor boxes are: (10×13) , (16×30) , (33×23) , (30×61) , (62×45) , (59×119) , (116×90) , (156×198) , (373×326) . These 9 anchor boxes are grouped into 3 different groups according to their size. The three smallest anchor boxes are assigned to the finer grid scale (e.g., 52×52), The three middle-sized anchor boxes are used for the middle fine grid scale (e.g., 26×26), and the three largest anchor boxes are used for the coarse grid scale (e.g., 13×13). Note that the anchor boxes can be normalized to the input image by dividing 416 or normalized to the grid size by dividing the corresponding stride (e.g., 32 for 13×13 grid scale).

With the information of anchor boxes, the final bounding box parameters and confidence score can be obtained by the transformation defined in equation (10.6). The relevant data from the last conv layers (i.e., layer 81, 93, 105 in Fig.10.12) need to pass a sigmoid function to form the class probabilities.

Like YOLO v2, the final step in the inference is to apply a non-max suppression on the prediction tensor to eliminate unqualified bounding boxes.

10.4.2 Loss function of YOLO v3

It is essential to understand the loss function and its implementation for training a YOLO model. Pre-trained YOLO v3 models are available in open resources for detecting pre-defined general 80 classes from COCO dataset. However, in many applications, it is required to train the model based on a customized dataset. In practice, we compute the loss at each grid scale independently, and then obtain the total loss by adding the losses at three grid scales.

A) Prediction

Suppose the prediction at a grid scale s is represented as a tensor with a shape $(3, s, s, 85)$, where s is the grid size (e.g., 13, 26, 52), denoted as $\text{pred}(3, s, s, 85)$. Thus, one predicted bounding box, corresponding to anchor box k ($k=0,1,2$), at grid cell (i, j) , is $\text{pred}[k, i, j, 0: 85]$ that has a format:

t_x	t_y	t_w	t_h	t_o	d_1	d_2	$\dots\dots$	d_c
↑ confidence								

Note that all numbers above are generated by the last Conv2d layer, without passing through any activation function or a sigmoid function. To match the format of the target (discussed later), it is convenient to update the prediction tensor, $\text{pred}(3, s, s, 85)$ to the following format, by applying a sigmoid function to t_x, t_y , tentatively,

$\sigma(t_x)$	$\sigma(t_y)$	t_w	t_h	t_o	d_1	d_2	$\dots\dots$	d_c
↑ confidence								

The meanings of the first four items were illustrated in Fig.10.9.

B) Target

To compute the loss of a prediction for an input image, we need to generate the target (i.e., ground truth) based on the label bounding boxes associated with the image. The label bounding

boxes are given in a format per object: $[class, bx, by, bw, bh]$, where bx and by are the center of the box, bw, bh are the width and height of the box, and all of them are relative to the image, $class$ is the index of the object class. The target can be organized as a tensor with a shape $(3, s, s, 6)$, illustrated in Fig.10.15 for $s=13$.

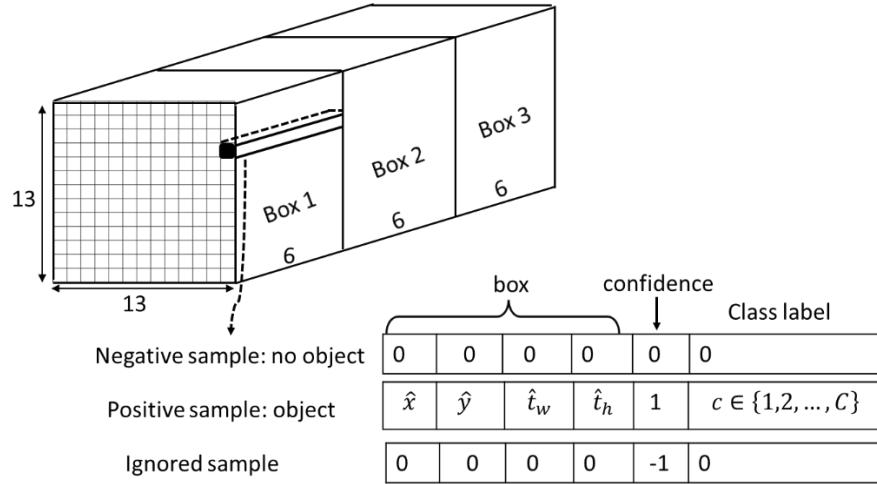


Fig.10.15 Target at grid scale 13

At each grid scale, we need to pick an anchor box responsible for the ground truth bounding box. The selected anchor box (positive sample) should be in the grid cell where the ground truth box is located, and also has the highest IOU with the ground truth box among the three anchor boxes in that cell. The confidence of the picked anchor box is assigned to 1. If any of the remaining two anchor boxes has a high (but not the highest) IOU with the ground truth box (e.g, more than an ignore_threshold), then they will be labeled as ignored box by assigning the confidence as -1, so that the corresponding prediction will be ignored, i.e., will not contribute any loss. All other target boxes (negative samples) are labeled as “no object” by assigning the confidence as 0.

In Fig.10.15, the elements in the target for positive samples are calculated as follows:

- 1) Ground truth box location offset to the up-left corner of the cell, relative to the grid cell size,

$$\begin{cases} \hat{x} = bx \cdot s - \text{int}(s \cdot bx) \\ \hat{y} = by \cdot s - \text{int}(s \cdot by) \end{cases} \quad (10.7)$$

- 2) Scaling parameters for width and height by the inverse function of eq. (10.6 c and 10.6 d)

$$\begin{cases} \hat{t}_w = \log\left(\frac{bw}{p_w}\right) \\ \hat{t}_h = \log\left(\frac{bh}{p_h}\right) \end{cases} \quad (10.8)$$

where p_w, p_h are the anchor box width and height relative to the image size, respectively.

Therefore, we generate the target tensor, denoted as $Target(3,s,s,6)$, shown in Fig.10.15 at each grid scale. There are three types of target vectors (each vector has 6 elements): negative sample (no object), positive sample (object), ignored sample. In the target tensor, each 6-element vector corresponds to a predicted 85-element vector at a grid cell and a particular anchor box.

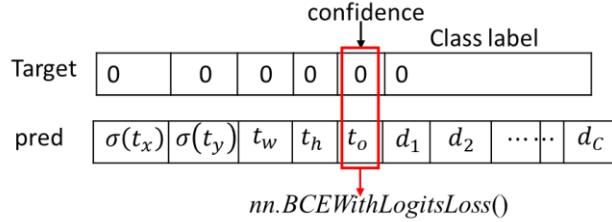
The loss function is defined to measure the mismatch between the target vectors and predicted vectors.

C) Loss function

Although Equation (10.5) was proposed for the loss function in YOLO v1, it defines the framework of loss function for the subsequent YOLO versions. There are many variants for the loss function and its implementation for YOLO v3. We introduce one of them below. The loss is composed of four parts: “no object” confidence loss, “object” confidence loss, “object” box loss, and “object” classification loss.

- 1) “no object” confidence loss. If the target has a confidence 0 (i.e., no object at the cell for the anchor box), the corresponding prediction only results in a confidence loss, specified as the binary cross-entropy loss,

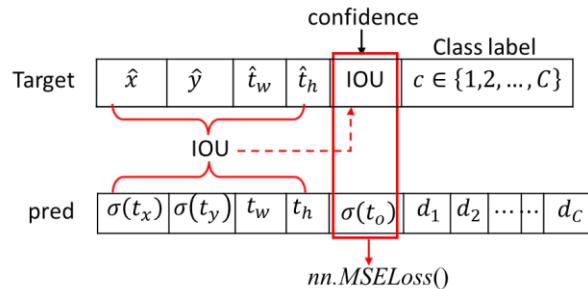
$$loss_{noobj} = -\lambda_{noobj} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{noobj} \log(1 - \sigma(t_o)) \quad (10.9)$$



In PyTorch, we can call `nn.BCEWithLogitsLoss()` to compute $loss_{noobj}$ using the confidence in the target (actually 0) and the prediction t_o , for all negative samples.

- 2) “object” confidence loss. For a positive sample, the confidence “1” in the target should be updated by the IOU between the target box and the predicted box, and then we compute the confidence loss as the mean squared error between the IOU and $\sigma(t_o)$,

$$loss_{obj} = \lambda_{obj} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} (IOU - \sigma(t_o))^2 \quad (10.10)$$

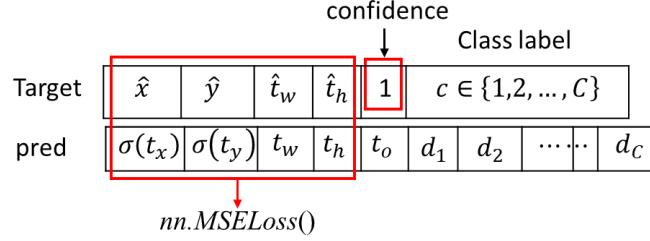


In PyTorch, we can call `nn.MSELoss()` to compute $loss_{obj}$, using the updated confidence scores (shown above) for all positive samples.

- 3) “object” box loss. For a positive sample, the box loss defines the location mismatch between the ground truth bounding boxes and the predicted bounding boxes. The box loss is specified as MSE loss,

$$loss_{box} = \lambda_{coord} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} \left[(\hat{x} - \sigma(t_x))^2 + (\hat{y} - \sigma(t_y))^2 + (\hat{t}_w - t_w)^2 + (\hat{t}_h - t_h)^2 \right]$$

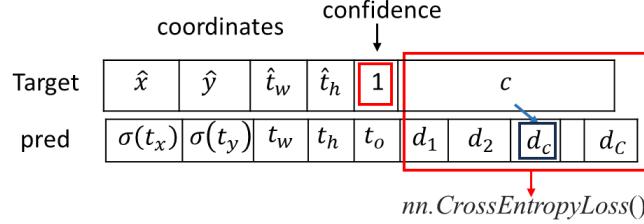
(10.11)



In PyTorch, we call `nn.MSELoss()` to compute $loss_{box}$, based on the coordinates field shown above.

- 4) “object” classification loss. For a positive sample, the classification loss is specified as cross entropy loss,

$$loss_{class} = -\lambda_{class} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} \log \left(\frac{\exp(d_c)}{\sum_{k=1}^C \exp(d_k)} \right) \quad (10.12)$$



Note that $d_k, k = 1, 2, \dots, C$, are the unnormalized logits from the last Conv2d layer for the grid scale prediction in YOLO v3 model. Thus, in PyTorch we can call `nn.CrossEntropyLoss()` to calculate $loss_{class}$, because `nn.CrossEntropyLoss()` combines the `nn.LogSoftmax` and `nn.NLLLoss` functions to compute the loss.

The total loss at the grid s is the sum of all above loss components (10.9), (10.10), (10.11), (10.12) as

$$\begin{aligned} Loss(s) &= loss_{noobj} + loss_{obj} + loss_{box} + loss_{class} \\ &= \lambda_{coord} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} \left[(\hat{x} - \sigma(t_x))^2 + (\hat{y} - \sigma(t_y))^2 + (\hat{t}_w - t_w)^2 + (\hat{t}_h - t_h)^2 \right] \\ &\quad - \lambda_{noobj} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{noobj} \log(1 - \sigma(t_o)) \\ &\quad + \lambda_{obj} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} (IOU - \sigma(t_o))^2 \\ &\quad - \lambda_{class} \sum_{i=0}^{s \times s} \sum_{j=0}^3 1_{ij}^{obj} \log \left(\frac{\exp(d_c)}{\sum_{k=1}^C \exp(d_k)} \right) \end{aligned} \quad (10.13)$$

where $s = 13, 26, 52$.

The final loss function of YOLO v3 model is the sum of losses at three grid scales,

$$LOSS = Loss(13) + Loss(26) + Loss(52) \quad (10.14)$$

Note that $\lambda_{noobj}, \lambda_{obj}, \lambda_{coord}, \lambda_{class}$ are the weights for the loss components, which balance the contribution of each loss component to the final loss. For example, the number of negative

samples in the target is much larger than that of positive samples, λ_{noobj} is set to a smaller number (e.g., 0.5) than the value of λ_{obj} (e.g., 5). The mask element, for grid cell i and anchor box j , is defined as

$$1_{ij}^{obj} = \begin{cases} 1 & \text{positive sample (i.e., confidence = 1)} \\ 0 & \text{negative sample (i.e., confidence = 0)} \end{cases} \quad (10.15)$$

$$1_{ij}^{noobj} = \begin{cases} 0 & \text{positive sample (i.e., confidence = 1)} \\ 1 & \text{negative sample (i.e., confidence = 0)} \end{cases} \quad (10.16)$$

10.5 Implementation of YOLO v3 Using Pre-trained Model

This section gives a comprehensive tutorial of YOLO v3 implementation based on the pre-trained model by Joseph Redmon (the inventor of YOLO). Three files are downloaded for this section: yolov3.cfg, yolov3.weights, and coco.names.

yolov3.cfg: <https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg>, a configuration file that defines the structure of YOLO v3 model.

yolov3.weights: <https://pjreddie.com/media/files/yolov3.weights>, a binary file that stores the weights in a float data type.

coco.names: <https://github.com/pjreddie/darknet/tree/master/data>, a text file that lists the name of classes in COCO dataset.

Fig.10.16 illustrates the flowchart of YOLO v3 implementation. The configuration file, *yolov3.cfg*, defines the architecture of YOLO v3 model. We use a function *parse_cfg()* to parse the cfg file, and eventually generate *yolonet()* as an nn.Module. The model *yolonet()* takes images as its input, and delivers all bounding boxes for all grid cells. The function *non_max_suppression()* generates the final detected bounding boxes by eliminating the low confident bounding boxes and redundant bounding boxes via the non-max suppression algorithm. We will explore the details of the implementation in the following subsections.

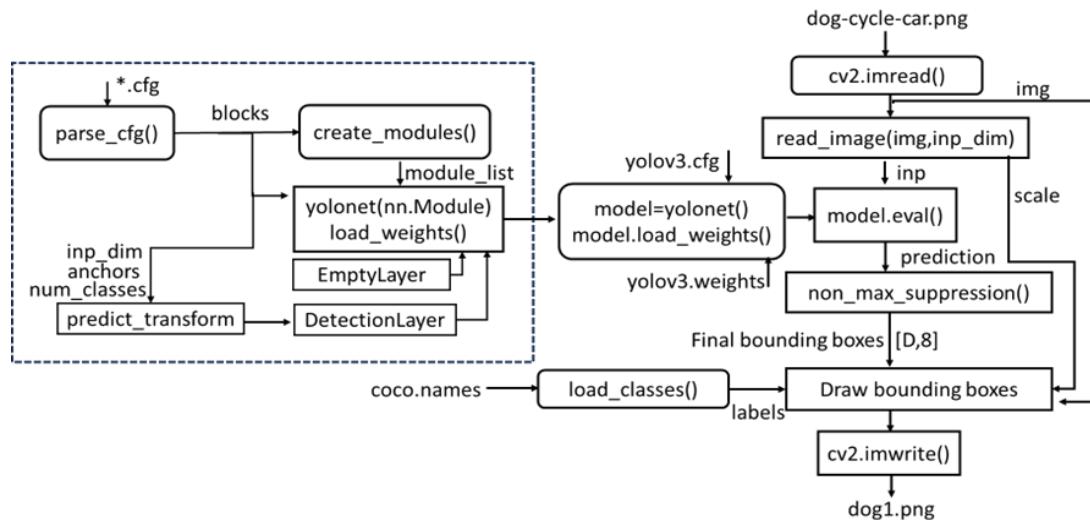


Fig.10.16 Flowchart of YOLO v3 implementation

10.5.2 Model architecture specified by a configuration file: yolov3.cfg

It is common practice to specify the architecture of a deep neural network by a configuration (or config) file, and then convert it to a neural network module described by a standard framework (e.g. *nn.Module* in PyTorch). A config file (*.cfg) is a text file that consists of a sequence of blocks.

In yolov3.cfg, there are six types of blocks: **[net]**, **[convolutional]**, **[shortcut]**, **[route]**, **[upsample]** and **[yolo]**. Each block, except the first block [net], specifies a layer. The first block in yolov3.cfg, called **[net]**, describes the information on the network input and training/testing parameters. Each of the subsequent blocks belongs to one of the other five types: [convolutional], [shortcut], [route], [upsample], [yolo], and specifies a layer. A block **[convolutional]** defines the structure of a conv layer, including the number of filters, filter size, stride, zero-padding, batch normalization and activation function. The number of input channels for a layer is defined by the number of output channels in the previous layer. A **[shortcut]** block specifies a jump connection. A **[route]** block has an attribute “*layers*” which can have either one, or two values. When *layers* attribute has only one value, it outputs the feature maps of the layer indexed by the value. When *layers* has two values, it returns the concatenated feature maps of the layers indexed by the two values. A block **[upsample]** up-samples the feature map from the previous layer by a factor of stride. A **[yolo]** block provides the parameters required for detection, and it corresponds to the detection layer, which processes the output feature map tensor for detection.

As examples, Table 10.3 lists some blocks in yolov3.cfg. The **[convolutional]** block specifies a convolutional layer with batch normalization and leakyReLU activation. The **[shortcut]** block describes a jump connection *from* layer -3, which means the output of the shortcut layer is obtained by adding feature maps from the previous (i.e. the first layer backwards) and the 3rd layer backwards from the shortcut layer. The block “[**route**] layers = -4” specifies a layer that outputs the feature map from the 4th layer backwards. The block “[**route**] layers = -1, 61” specifies a layer that outputs a concatenated (along with depth dimension) feature map from the previous layer (-1) and the 61st layer. The **[upsample]** block defines a layer that up-samples the feature map from the previous layer by a factor of stride 2 using a default mode – nearest mode. The **[yolo]** block gives anchor boxes and other parameters for detection at the corresponding scale. The *anchors* give all anchors while *mask* specifies the corresponding three anchors for this grid scale.

Table 10.3 Examples of blocks in yolov3.cfg

[convolutional] batch_normalize=1 filters=32 size=3 stride=1 pad=1 activation=leaky	[yolo] mask = 6,7,8 anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326 classes=80 num=9 jitter=.3 ignore_thresh = .5 truth_thresh = 1 random=1		
[shortcut] from=-3 activation=linear	[route] layers = -4	[route] layers = -1, 61	[upsample] stride=2

10.5.2 Create the model and load the weights

In this section, we will explore how to create the neural network model and load the pre-trained weights to the model, from two files – yolov3.cfg and yolov3.weights. All Python codes can be run through Jupyter notebook.

We start by importing the basic packages.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import numpy as np
import cv2
```

The function *parse_cfg* is to parse the cfg file and store every block as a dictionary data type. The attributes of each block and their values are stored as key-value pairs in the dictionary. As we parse through the cfg, we keep appending these dictionaries, denoted by the variable *block* in the code, to a list *blocks*. The function returns this list *blocks*.

```
def parse_cfg(cfgfile):
    """
    input: a configuration file, eg. yolov3.cfg

    Returns: blocks, which is a list of blocks.
    -- each block corresponds to a block in cfg file
    -- each block is represented as a dictionary
    """

    file = open(cfgfile, 'r')
    #lines = file.read().split('\n')                      # store the lines in a list
    lines = file.read().splitlines()                      # store the lines in a list
    lines = [x for x in lines if len(x) > 0]           # remove the empty lines
    lines = [x for x in lines if x[0] != '#']          # remove the comment lines
    lines = [x.rstrip().lstrip() for x in lines]
    # remove leading and trailing whitespaces
    #print(lines)
    # lines: a list of strings, each string is an effective line in cfg

    block = {}                                         # initial a dict
    blocks = []                                         # initial list

    for line in lines:
        if line[0] == "[":
            # line is a string, line[0] is the first character in the line
            if len(block) != 0:
                # If block is not empty,
                # the "block" is for the previous block
                # and ready to store to "blocks"
                blocks.append(block)      # add it the blocks list
                block = {}               # re-init the block
                block["type"] = line[1:-1].rstrip()
                # store the block "type" for current block
            else:
                pass
```

```

        key,value = line.split(">")
        block[key.rstrip()] = value.lstrip() #store "value" to block[key]
blocks.append(block) # store the last "block" to the list

return blocks

```

The content of “blocks” is printed below. The “blocks” include 108 blocks. The first block is “net” about the information of the model, The remaining 107 blocks correspond to 107 layers in Fig.10.12.

```
print(blocks)
```

```
[{'type': 'net', 'batch': '1', 'subdivisions': '1', 'width': '416', 'height': '416', 'channels': '3', 'momentum': '0.9', 'decay': '0.0005', 'angle': '0', 'saturation': '1.5', 'exposure': '1.5', 'hue': '.1', 'learning_rate': '0.001', 'burn_in': '1000', 'max_batches': '500200', 'policy': 'steps', 'steps': '400000,450000', 'scales': '.1,.1'}, {'type': 'convolutional', 'batch_normalize': '1', 'filters': '32', 'size': '3', 'stride': '1', 'pad': '1', 'activation': 'leaky'}, ...]
```

The *create_modules* function takes the list *blocks* generated by the *parse_cfg* function, and returns *net_info* and *module_list*. The *net_info* is a dictionary that stores information about the network. The *module_list* is a list *nn.ModuleList()* that contains all 107 layers in YOLO v3, including the index of each layer. The *EmptyLayer()* is defined for route and shortcut layers while *DetectionLayer()* is defined for the YOLO detection head. Each element in *module_list* corresponds to a layer in Fig.10.12.

```

class EmptyLayer(nn.Module):
    def __init__(self):
        super(EmptyLayer, self).__init__()

class DetectionLayer(nn.Module):
    def __init__(self, anchors):
        super(DetectionLayer, self).__init__()
        self.anchors = anchors

def create_modules(blocks):
    """
    input: a list -- blocks, each element is a dict -- block
    returns: net_info -- the first block about the net
             module_list: a list of nn.modules
    """
    net_info = blocks[0] # net info is a dict for the net information
    module_list = nn.ModuleList() # init a list containing nn.modules
    prev_filters = 3 # initial prev_filters
    output_filters = [] # store

    for index, x in enumerate(blocks[1:]):
        module = nn.Sequential() # initial module for current block

        #start with blocks[1]

```

```

#check the type of block
#create a new module for the block
#append to module_list

#if a convolutional layer
if (x["type"] == "convolutional"):
    #Get the info about the layer
    activation = x["activation"]
    try:
        batch_normalize = int(x["batch_normalize"])
        bias = False
    except:
        batch_normalize = 0
        bias = True

    filters= int(x["filters"])
    padding = int(x["pad"])           # whether zero padding
    kernel_size = int(x["size"])
    stride = int(x["stride"])

    if padding:
        pad = (kernel_size - 1) // 2 #pad:the zero pad in nn.Conv2d()
    else:
        pad = 0

    #Add the convolutional layer
    conv = nn.Conv2d(prev_filters, filters, kernel_size, stride, pad,
bias = bias)
    module.add_module("conv_{0}".format(index), conv)

    #Add the Batch Norm Layer
    if batch_normalize:
        bn = nn.BatchNorm2d(filters)
        module.add_module("batch_norm_{0}".format(index), bn)

    #Check the activation.
    if activation == "leaky":
        activn = nn.LeakyReLU(0.1, inplace = True)
        module.add_module("leaky_{0}".format(index), activn)

    # If an upsampling layer
    # nearest mode, factor = stride
    elif (x["type"] == "upsample"):
        stride = int(x["stride"])
        upsample = nn.Upsample(scale_factor = stride, mode = "nearest")
        module.add_module("upsample_{0}".format(index), upsample)
        # filters is the same as the previous layer, no need to update

    #If a route layer
    elif (x["type"] == "route"):

        route = EmptyLayer()
        module.add_module("route_{0}".format(index), route)

        x["layers"] = x["layers"].split(',')
        #Start of a route
        start = int(x["layers"][0])    # the first integer e.g -1, or -4

```

```

#end, if there is a second value for layers, e.g, 36, or 61.
try:
    end = int(x["layers"][1]) # second value
except:
    end = 0      # no second value

if start > 0:
    start = start - index
if end > 0:
    end = end - index

if end == 0: # only one value (start)
    filters = output_filters[index + start]
else: # two values (start, end)
    filters=output_filters[index+start]+output_filters[index+end]

#shortcut corresponds to skip connection
elif x["type"] == "shortcut":
    shortcut = EmptyLayer()
    module.add_module("shortcut_{}".format(index), shortcut)
    # no need to update filters

#Yolo is the detection layer
elif x["type"] == "yolo":
    mask = x["mask"].split(",")
    mask = [int(x) for x in mask]

    anchors = x["anchors"].split ","
    anchors = [int(a) for a in anchors]
    anchors=[(anchors[i],anchors[i+1]) for i in
range(0,len(anchors),2)]
    anchors = [anchors[i] for i in mask]

    detection = DetectionLayer(anchors)
    module.add_module("Detection_{}".format(index), detection)
    # no need to update filters

    module_list.append(module)      # add module to the module_list
    prev_filters = filters
    # will be used for the input channels if the next layer is conv2d

    output_filters.append(filters)
    # save the channels for layers to be used in layer "route"

return (net_info, module_list)

```

module_list can be printed as below, and its length is 107.

```

print(create_modules(blocks))

({'type': 'net', 'batch': '1', 'subdivisions': '1', 'width': '416', 'height':
'416', 'channels': '3', 'momentum': '0.9', 'decay': '0.0005', 'angle': '0',
'saturation': '1.5', 'exposure': '1.5', 'hue': '.1', 'learning_rate':

```

```

'0.001', 'burn_in': '1000', 'max_batches': '500200', 'policy': 'steps',
'steps': '400000,450000', 'scales': '.1,.1'}, ModuleList(
    (0): Sequential(
        (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (leaky_0): LeakyReLU(negative_slope=0.1, inplace=True)
    )
    (1): Sequential(
        (conv_1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (batch_norm_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (leaky_1): LeakyReLU(negative_slope=0.1, inplace=True)
    )
...
)

```

Based on the functions `parse_cfg()` and `create_modules()`, we can build the entire neural network for YOLO v3, called `Yolonet()`. In addition to the basic layers in PyTorch, such as `nn.Conv2d()`, `nn.BatchNorm2d()`, `nn.LeakyReLU()`, an important function, called `predict_transform()`, has been defined for the operation in `DetectionLayer()`, which is illustrated in Fig.10.17.

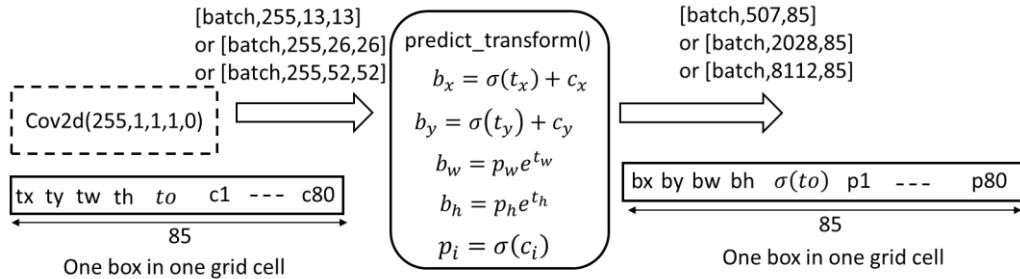


Fig.10.17 the operation of `predict_transform()`, c_x, c_y are the coordinates of the grid cell.

```

def predict_transform(prediction, inp_dim, anchors, num_classes):
    # this function is used in yolo DetectionLayer()
    # generates all predictions for one grid scale
    # 1) re-organize the tensor
    # 2) scale the anchors
    # 3) coordinate transform
    # 4) apply sigmoid for class probabilities

    """
    inputs:
        -- prediction: output from the last conv2d
            shape [batch, 255, grid_size, grid_size] (e.g. [b, 255, 13, 13])
        -- inp_dim: input size, integer, e.g. 416
        -- anchors: a list of 3 anchors for this grid scale,
            [(*,*), (*,*), (*,*)]
        -- num_classes: the number of classes, integer, e.g. 80
    returns:
        -- prediction: tensor shape [batch, grid_size x grid_size x 3, 85]
    each box: prediction[i,j,:]=(bx,by,bw,bh,sigmoid(to), p1, ..., p80)

```

```

"""
batch_size = prediction.size(0)
stride = inp_dim // prediction.size(2)
grid_size = prediction.size(2)
bbox_attrs = 5 + num_classes
num_anchors = len(anchors)

prediction = prediction.view(batch_size, bbox_attrs*num_anchors,
grid_size*grid_size)
#example: [b, 255,13,13]--> [b, 255, 169]

prediction = prediction.transpose(1,2).contiguous()
#example: [b,255,169] --> [b,169,255]

prediction = prediction.view(batch_size, grid_size*grid_size*num_anchors,
bbox_attrs)
#example: [b,169,255] --> [b,169x3,85]=[b,507,85]

anchors = [(a[0]/stride, a[1]/stride) for a in anchors]
# scaled to unit grid size

#Sigmoid the centre_X, centre_Y. and object confidence
prediction[:, :, 0] = torch.sigmoid(prediction[:, :, 0])
prediction[:, :, 1] = torch.sigmoid(prediction[:, :, 1])
prediction[:, :, 4] = torch.sigmoid(prediction[:, :, 4])

#Add the center offsets
grid = np.arange(grid_size)
a,b = np.meshgrid(grid, grid)    # a: (13,13), b: (13,13)

x_offset = torch.FloatTensor(a).view(-1,1)  #shape(169,1)
y_offset = torch.FloatTensor(b).view(-1,1)  #shape(169,1)

x_y_offset = torch.cat((x_offset, y_offset),
1).repeat(1,num_anchors).view(-1,2).unsqueeze(0)
# x_y_offset, shape (1, 507, 2), grid coordinates repeated 3 time.
# [[0.,0.], [0., 0.], [0.,0.],
# [1.,0.], [1.,0.], [1.,0.],
# [2.,0.], [2.,0.], [2.,0.],
# ....
# [12.,12.], [12.,12.], [12.,12.]]

prediction[:, :, :2] += x_y_offset
# bounding box: bx, by

#log space transform height and the width
anchors = torch.FloatTensor(anchors)

anchors = anchors.repeat(grid_size*grid_size, 1).unsqueeze(0)
# shape: (1, 507, 2) to match prediction (1, 507, 85)

prediction[:, :, 2:4] = torch.exp(prediction[:, :, 2:4])*anchors
# bounding box: bw, bh

```

```

    prediction[:, :, 5: 5 + num_classes] = torch.sigmoid((prediction[:, :, 5 : 5
+ num_classes]))

    prediction[:, :, :, 4] *= stride
    # scaled back to unit of pixel, shape is [batch, 13x13x3, 85]
    # one bounding box prediction[1,1,: ] is (bx,by,bw,bh,sigmoid(to), p1, p2,
..., p80)

    return prediction

```

In class `Yolonet()`, we construct the forward path of YOLO v3, and also define `load_weights(self, weightfile)` to add an attribute to `Yolonet()` for loading the pre-trained weights.

```

class Yolonet(nn.Module):
    def __init__(self, cfgfile):
        super(Yolonet, self).__init__()
        self.blocks = parse_cfg(cfgfile)
        self.net_info, self.module_list = create_modules(self.blocks)

    def forward(self, x):
        modules = self.blocks[1:]
        outputs = {}      # We save the outputs for the route or shortcut layer,
                         # key is layer index, value is the output of the layer

        first = 1
        for i, module in enumerate(modules):
            module_type = (module["type"])

            if module_type == "convolutional" or module_type == "upsample":
                x = self.module_list[i](x)

            elif module_type == "route":
                layers = module["layers"]
                layers = [int(a) for a in layers]

                if (layers[0]) > 0:
                    layers[0] = layers[0] - i

                if len(layers) == 1:
                    x = outputs[i + (layers[0])]

            else:
                if (layers[1]) > 0:
                    layers[1] = layers[1] - i

                map1 = outputs[i + layers[0]]
                map2 = outputs[i + layers[1]]
                x = torch.cat((map1, map2), 1)

            elif module_type == "shortcut":
                from_ = int(module["from"])
                x = outputs[i-1] + outputs[i+from_]

            elif module_type == 'yolo':

```

```

        anchors = self.module_list[i][0].anchors
        #Get the input dimensions
        inp_dim = int (self.net_info["height"])

        #Get the number of classes
        num_classes = int (module["classes"])

        #Transform
        x = x.data
        x = predict_transform(x, inp_dim, anchors, num_classes)
        if first:                      #if the first yolo detectio head.
            detections = x
            first = 0

        else:           # if the 2nd and 3rd head
            detections = torch.cat((detections, x), 1)

        outputs[i] = x

    return detections

def load_weights(self, weightfile):
    # Open weightfile
    fp = open(weightfile, "rb")

    #The first 5 values are header information
    header = np.fromfile(fp, dtype = np.int32, count = 5)
    self.header = torch.from_numpy(header)
    self.seen = self.header[3]

    weights = np.fromfile(fp, dtype = np.float32)

    ptr = 0
    for i in range(len(self.module_list)):
        module_type = self.blocks[i + 1]["type"]

        # If module_type is convolutional load weights
        # Otherwise do nothing.

        if module_type == "convolutional":
            model = self.module_list[i]
            try:
                batch_normalize=int(self.blocks[i+1]["batch_normalize"])
            except:
                batch_normalize = 0

            conv = model[0]

            if (batch_normalize):   # if BN applied, load BN parameters
                bn = model[1]

                #Get the number of weights of Batch Norm Layer
                num_bn_biases = bn.bias numel()

                #Load the weights

```

```

        bn_biases = torch.from_numpy(weights[ptr:ptr +
num_bn_biases])

        ptr += num_bn_biases

        bn_weights = torch.from_numpy(weights[ptr: ptr +
num_bn_biases])
        ptr += num_bn_biases

        bn_running_mean = torch.from_numpy(weights[ptr: ptr +
num_bn_biases])
        ptr += num_bn_biases

        bn_running_var = torch.from_numpy(weights[ptr: ptr +
num_bn_biases])
        ptr += num_bn_biases

#Cast the loaded weights into dims of model weights.
bn_biases = bn_biases.view_as(bn.bias.data)
bn_weights = bn_weights.view_as(bn.weight.data)
bn_running_mean = bn_running_mean.view_as(bn.running_mean)
bn_running_var = bn_running_var.view_as(bn.running_var)

#Copy the data to model
bn.bias.data.copy_(bn_biases)
bn.weight.data.copy_(bn_weights)
bn.running_mean.copy_(bn_running_mean)
bn.running_var.copy_(bn_running_var)

else: # if no BN, load conv layer biases
#Number of biases
num_biases = conv.bias.numel()

#Load the weights
conv_biases = torch.from_numpy(weights[ptr: ptr +
num_biases])
ptr = ptr + num_biases

#reshape the loaded weights according to the dims of the
model weights
conv_biases = conv_biases.view_as(conv.bias.data)

#Finally copy the data
conv.bias.data.copy_(conv_biases)

#load the weights for the Convolutional layers
num_weights = conv.weight.numel()
conv_weights = torch.from_numpy(weights[ptr:ptr+num_weights])
ptr = ptr + num_weights

conv_weights = conv_weights.view_as(conv.weight.data)
conv.weight.data.copy_(conv_weights)

```

We can instantiate *Yolonet()* and load the weights as below.

```

model = Yolonet("cfg/yolov3.cfg")
model.load_weights("yolov3.weights")

```

Given a batch of images, to get the output from model, we just run: $pred = model(inp)$. The input, inp , is a tensor with shape [Batch_size, Channel, Height, Width] assuming all images have the same Height and Width. The output, $pred$, is a tensor that contains the predicted bounding boxes for all images, and its shape is [Batch_size, number of boxes, 85], where

$$\text{number of boxes} = (13 \times 13 + 26 \times 26 + 52 \times 52) \times 3 = 10647$$

when image size is $(3 \times 416 \times 416)$ and the number of classes is 80. The prediction for image i is $pred[i,:,:]$, shown in Fig.10.18. There are three bounding boxes for one grid cell. The prediction for each bounding box includes 85 numbers: 4 for box center and width and height, 1 for object confidence score, 80 for class probabilities.

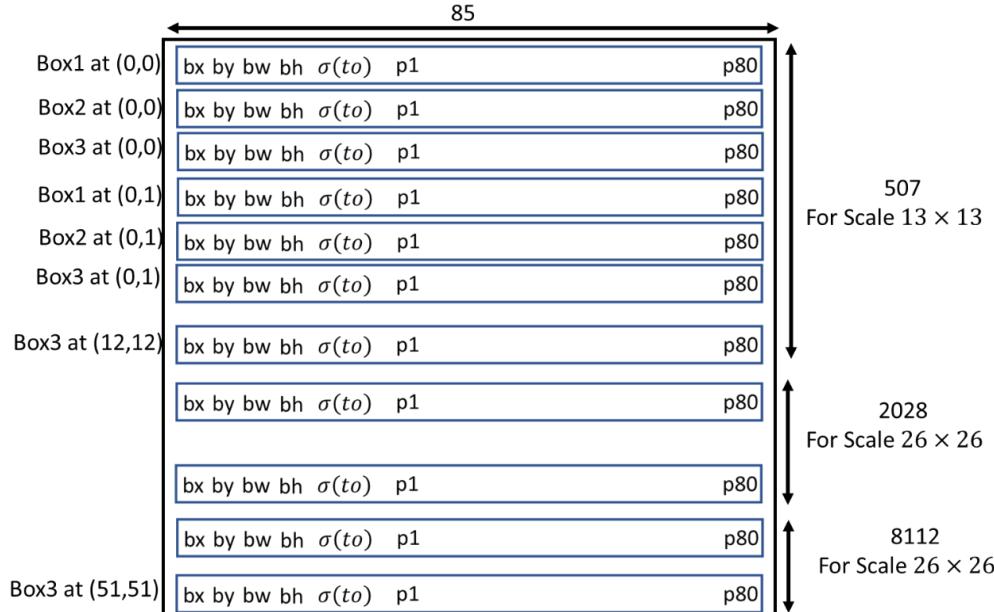


Fig.10.18 The output of *Yolonet()* for one image i , $pred[i,:,:]$. Note that the unit of bx,by,bw,bh is a pixel, and that p1,..., p80 are class probabilities after sigmoid function.

```
inp_random = torch.rand(2, 3, 416, 416)
pred_random = model(inp_random)
pred_random.shape

torch.Size([2, 10647, 85])
```

10.5.3 Non-max suppression

The model *Yoloknet* delivers all bounding boxes for a batch of images, with 10647 bounding boxes per image, as shown in Fig.10.18. Since most of the bounding boxes with low confidence scores are not responsible for any object or some of them points to the same object, we need to eliminate the low confident bounding boxes (i.e., $\sigma(to) < threshold$) and the redundant bounding boxes for one object, so that each detected object is framed by only one bounding box. This is done by the algorithm of non-max suppression.

First, we define a function, `bbox_iou(box1,box2)`, to calculate the intersection over union between box1 and box2. Then the function, `non_max_suppression(prediction, confidence, num_classes,`

nms_conf), is defined to convert the output of *Yolonet, prediction*, to the finally detected bounding boxes, *output*, through non-max suppression. The tensor *prediction* has a shape of [Batch_size, 10647, 85] while the tensor *output* has a shape of [D,8], where D is the total number of detected bounding boxes for the batch. The format of each bounding box in prediction or output is illustrated in Fig.10.19.

Specifically, *non_max_supresion ()* performs the following operations in order: 1) zero out all bounding boxes whose object confidence score $\sigma(to)$ is less than the threshold *confidence*; 2) convert center coordinates to corner coordinates for each bounding box; 3) on each image, perform non-max suppression class-wisely; 4) attach the image index to the corresponding bounding boxes, and keep the maximum of class probabilities and the corresponding class index, while discarding other class probabilities. As the result, each detected bounding box has a format of an 8-element vector: (image_index, x1, y1, x2, y2, $\sigma(to)$, max(pc), argmax(pc)).

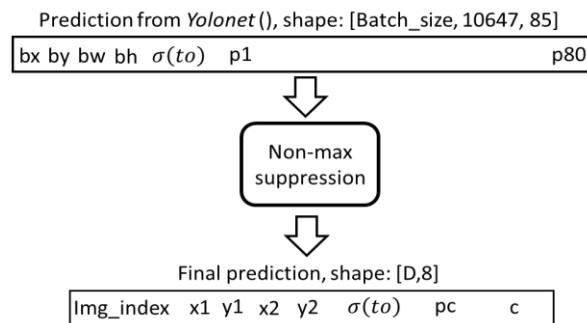


Fig.10.19 Input and output of non-max suppression. D is the total number of final bounding boxes in the batch.

```

def bbox_iou(box1, box2):
    """
    inputs:
        --box1: tensor shape [N1,4], given as top-left and bottom-right
        --box2: tensor shape [N2,4], given as top-left and bottom-right
            box1 and box2 should include the same number of boxes (N1=N2),
            or one of them includes only one box and the other includes
            multiple boxes (N1=1, or N2=1)

    Returns
        -- if N1 =N2, returns one-to-one IOUs
        -- if N1=1 or N2=1, returns one-to-many IOUs
    """
    #Get the coordinates of bounding boxes
    b1_x1, b1_y1, b1_x2, b1_y2 = box1[:,0], box1[:,1], box1[:,2], box1[:,3]
    b2_x1, b2_y1, b2_x2, b2_y2 = box2[:,0], box2[:,1], box2[:,2], box2[:,3]

    #get the corrdinates of the intersection rectangle
    inter_rect_x1 = torch.max(b1_x1, b2_x1)
    inter_rect_y1 = torch.max(b1_y1, b2_y1)
    inter_rect_x2 = torch.min(b1_x2, b2_x2)
    inter_rect_y2 = torch.min(b1_y2, b2_y2)

```

```

#Intersection
inter_area = torch.clamp(inter_rect_x2 - inter_rect_x1 + 1, min=0) *
torch.clamp(inter_rect_y2 - inter_rect_y1 + 1, min=0)

#Union
b1_area = (b1_x2 - b1_x1 + 1)*(b1_y2 - b1_y1 + 1)
b2_area = (b2_x2 - b2_x1 + 1)*(b2_y2 - b2_y1 + 1)

#IOU
iou = inter_area / (b1_area + b2_area - inter_area)

return iou

```



```

def non_max_suppression(prediction, confidence, num_classes, nms_conf = 0.4):
    """
    delivers all final bounding boxes, which are ready to draw on original
    images.

    inputs:
        -- prediction, from predict_transform() in Darknet model, includes all
           bounding boxes shape is [batch, (13x13+26x26+52x52)x3, 85]
           =[batch, 10647, 85], 10647 bounding boxes per batch
           prediction[0,0,:]:
           bx,by,bw,bh,sigmoid(t0),p1=sigmoid(), p2=sigmoid()..., p80=sigmoid()
        -- confidence, object confidence threshold (e.g. 0.5) for sigmoid(t0) in
           each bounding box
        -- num_classes, integer, the number of class (e.g. 80)
        -- nms_conf, the threshold of IOU for non-max suppression
           outputs:
        -- output, a tensor [N, 8], N is the total number of predicted boxes for
           the batch
           each box:
           (image_index, x1,y1,x2,y2,sigmoid(to), class probability, class index)
    """

    conf_mask = (prediction[:,:,:4] > confidence).float().unsqueeze(2)
    #.unsqueeze(2) keep the dimension 2 to match the prediction dimension
    #[batch, 10647,85]
    # conf_mask shape is [batch, 10647,1]

    prediction = prediction*conf_mask
    # clear all low confident bounding boxes to zero-vectors
    # maintain all other bounding boxes

    # convert (center,w,h) format to (top-left, bottom-right) format
    box_corner = prediction.new(prediction.shape) #create a new tensor
    box_corner[:,:,:0] = (prediction[:,:,:0] - prediction[:,:,:2]/2)
    box_corner[:,:,:1] = (prediction[:,:,:1] - prediction[:,:,:3]/2)
    box_corner[:,:,:2] = (prediction[:,:,:0] + prediction[:,:,:2]/2)
    box_corner[:,:,:3] = (prediction[:,:,:1] + prediction[:,:,:3]/2)
    prediction[:,:,:4] = box_corner[:,:,:4]
    # now, prediction in a corner format:
    #[x1, y1, x1, y2, sigmoid(t0), sigmoid().....]

batch_size = prediction.size(0)

```

```

#initial
first = True

for ind in range(batch_size):
    image_pred = prediction[ind]
    # image_pred [10647,85]: prediction for one image
    #confidence threshholding
    #NMS

    max_conf, max_conf_score=torch.max(image_pred[:,5:5+num_classes], 1)
    max_conf = max_conf.float().unsqueeze(1)
    max_conf_score = max_conf_score.float().unsqueeze(1)
    # max_conf: max value, shape [10647,1]
    # max_conf_score: max index, shape [10647,1]

    seq = (image_pred[:, :5], max_conf, max_conf_score)
    image_pred = torch.cat(seq, 1)
    #shape [10647,7], (x1,y1,x2,y2,sigmoid(to),pc_max,class_max)

    non_zero_ind = (torch.nonzero(image_pred[:,4]))
    try:
        image_pred_ = image_pred[non_zero_ind.squeeze(),:].view(-1,7)
    except:
        continue

    if image_pred_.shape[0] == 0:
        continue
    # image_pred_, includes only detected bounding boxes, shape [M1, 7]

    #Get the various classes detected in the image
    img_classes = torch.unique(image_pred_[:, -1])
    # keep one element for each detected class.
    # img_class shape [M2], M2 < or = M1

    for cls in img_classes:
        #perform NMS for each class

        #get the detections with one particular class
        cls_mask=image_pred_* (image_pred_[:, -1]==cls).float().unsqueeze(1)
        class_mask_ind = torch.nonzero(cls_mask[:, -2]).squeeze()
        image_pred_class = image_pred_[class_mask_ind].view(-1,7)
        # image_pred_class [M3,7]: detected bounding boxes for class cls

        # sort the detections such that the bounding box
        # with the maximum objectness confidence is at the beginning
        conf_sort_index=torch.sort(image_pred_class[:,4],descending=True
) [1]

        image_pred_class = image_pred_class[conf_sort_index]
        idx = image_pred_class.size(0)
        #Number of detected bounding boxes for class cls

        for i in range(idx):
            #Get the IOUs of all boxes that come after the one we are looking
            # at in the loop
            try:

```

```

        ious = bbox_iou(image_pred_class[i].unsqueeze(0),
image_pred_class[i+1:])
        # compute IOUs between box[i] and all the rest boxes
    except ValueError:
        break

    except IndexError:
        break

    #Zero out all the detections that have IoU > threshold
    iou_mask = (ious < nms_conf).float().unsqueeze(1)
    image_pred_class[i+1:] *= iou_mask

    #Remove the bounding boxes whose IOU with box[i]
    #is greater than nms_conf
    non_zero_ind = torch.nonzero(image_pred_class[:,4]).squeeze()
    image_pred_class = image_pred_class[non_zero_ind].view(-1,7)

    batch_ind = image_pred_class.new(image_pred_class.size(0),
1).fill_(ind)

    seq = (batch_ind, image_pred_class)
    # attach the batch index to the bounding boxes for class cls

    if first:
        output = torch.cat(seq,1)
        first = False
    else:
        out = torch.cat(seq,1)
        output = torch.cat((output,out))
    # output shape [M4, 8], M4 is the number of valid bounding boxes
    # up to current batch, image, class cls

    # output shape [M5, 8], M5 is the number of valid BBs up to
    # current image

try:
    return output
except:
    return 0
# the returned output [D,8], includes D bounding boxes for the batch
# each box:
# (image_index, x1,y1,x2,y2,sigmoid(to), class probability, class index)

```

10.5.4 Put all together

Finally, we apply the model *Yolonet* with pre-trained weights and non-max suppression for an object detection task. The final detected bounding boxes are plotted on the testing image.

Since the pre-trained model was trained on COCO dataset, a list of class names in the dataset is created, and saved as one line per name in a text file, named *coco.names*. We define a function, *load_classes ()*, to read the names from the file, and return a list of classes. With this list, we can display the name of detected object in the image.

```
def load_classes(namesfile):
```

```

fp = open(namesfile, "r")
names = fp.read().split("\n")[:-1]
return names

```

We use CV2 to load the images. Since CV2 loads an image as a numpy array [H,W,Channel], with BGR as the order of the color channels, and PyTorch neural network input format is [Batch, Channel, inp_dim, inp_dim], with the channel order of RGB, we need to write a function *read_image* to transform the numpy array into PyTorch input format. First, *read_image()* scales the original image ($H \times W$) to fit the frame ($inp_dim \times inp_dim$) as large as possible while keeping the aspect ratio unchanged, and pads the left out areas with the color (128,128,128), illustrated in Fig.10.20. Then *read_image()* convert the data format to [Batch, Channel, inp_dim, inp_dim]. The scale factor is also returned to be used for drawing the detected bounding boxes on the original image.

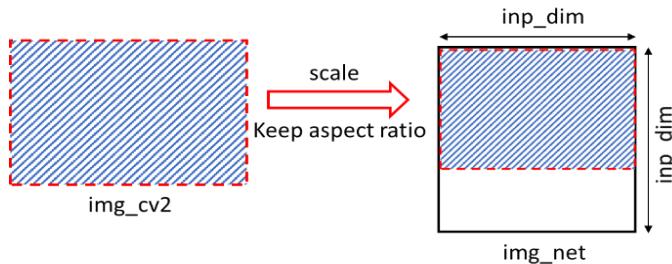


Fig.10.20 Scale the image to fit the input size $inp_dim \times inp_dim$

```

def read_image(img_cv2, inp_dim):
    """
    converts and scale the image read by cv2 to
    a tensor [batch,channel,inp_dim,inp_dim] for yolonet

    inputs:
        -- img_cv2: shape is [h,w,channel]: the orignal image from
    cv2.imread
        -- inp_dim: integer, input size to yolo nueral network, e.g.
416
    Returns
        -- img_net: tensor shape is [batch, channel, height, width]
        -- scale_factor
    """
    #img, scale_factor = (scale_image(img, (inp_dim, inp_dim)))

    img_cv2_h, img_cv2_w = img_cv2.shape[0], img_cv2.shape[1]
    w = inp_dim
    h = inp_dim
    scale_factor = min(w/img_cv2_w, h/img_cv2_h)
    new_w = int(img_cv2_w * scale_factor)
    new_h = int(img_cv2_h * scale_factor)
    resized_image = cv2.resize(img_cv2, (new_w,new_h), interpolation =
cv2.INTER_CUBIC)
    # cv2.resize(img, (width, height), interpolation =...)

    canvas = np.full((inp_dim, inp_dim, 3), 128)

```

```

    canvas[0:new_h, 0:new_w, :] = resized_image
    # align the resized image to the top-left of canvas

    img_net = canvas[:, :, ::-1].transpose((2, 0, 1)).copy()
    # canvas[:, :, ::-1] reverse the number in color dimension BGR--> RGB
    # In CV2, channel order is [blue, green, red],
    # but in PyTorch [red, green, blue]
    # transpose((2,0,1)): convert [H,W,C] to [C,H,W] for PyTorch

    img_net = torch.from_numpy(img_net).float().div(255.0).unsqueeze(0)
    # add batch dimension
    # img_net: [1, channel, height, width]
    return img_net, scale_factor

```

By putting all together, the following codes perform the object detection on the benchmark image "dog-cycle-car.png". If non-max-suppression is not applied (i.e., nms_thresh=1.0), there are multiple bounding boxes for one object, as shown in Fig.10.21(a). After non-max-suppression (e.g., nms_thresh=0.4) is applied, each detected object is frame by one bounding box, as shown in Fig.10.21(b).

```

import os

if not os.path.exists("./results"):
    os.makedirs("./results")

inp_dim = 416
batch_size = 1
confidence = 0.5 # threshold for confidence score, default 0.5
nms_thresh = 0.4 # threshold for nms iou, default 0.4
num_classes = 80
classes = load_classes("data/coco.names")

model = Yolonet("cfg/yolov3.cfg")
model.load_weights("yolov3.weights")
model.eval()

img = cv2.imread("dog-cycle-car.png")
inp, scale_factor = read_image(img, inp_dim)
pred = model(inp)
prediction = non_max_suppression(pred, confidence, num_classes,
nms_conf = nms_thresh)

prediction[:,1:5] /= scale_factor

for i in range(prediction.shape[0]):
    x = prediction[i]

    c1 = (x[1:3].int()).tolist()
    c2 = (x[3:5].int()).tolist()
    #img = results[int(x[0])]
    cls = int(x[-1])
    color = (5*cls,10*cls,100*cls)

```

```

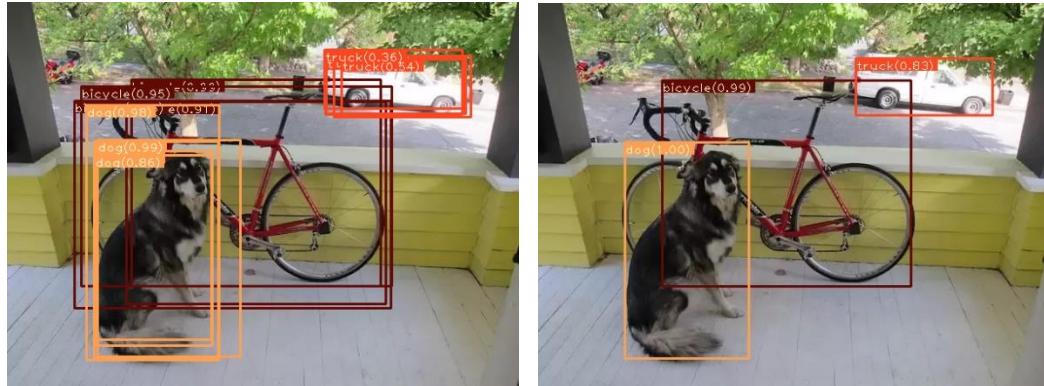
label = "{0}".format(classes[cls])
cv2.rectangle(img, c1, c2,color, 2)
t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1 , 1)[0]
c2 = c1[0] + t_size[0] + 50, c1[1] + t_size[1] + 4
cv2.rectangle(img, c1, c2,color, -1)
prob="(% .2f)" % round(float(x[5]*x[6]), 2)
cv2.putText(img, label+str(prob), (c1[0], c1[1] + t_size[1] + 4),
cv2.FONT_HERSHEY_PLAIN, 1, [225,255,255], 1);

cv2.imwrite("./results/dog1.png", img)

print(prediction.shape)

tensor([[0.0000,146.6964, 91.9886,439.0550,334.3987,0.9917,0.9982,1.0000],
       [0.0000,374.5365, 65.5042,535.1949,132.5022,0.9943,0.8332,7.0000],
       [0.0000,102.2761,164.2881,248.8867,419.3336,1.0000,0.9996,16.0000]])

```



(a) without nms (i.e., nms_thresh=1)

(b) with nms (nms_thresh=0.4)

Fig.10.21 YOLO v3 results on an image.

10.6 A Metric for Object Detection: mAP

Mean average precision (mAP) is a popular metric to measure the performance of an object detection model.

10.6.1 Precision and recall in object detection

The definition of mAP is built upon the concepts of precision and recall. The metrics, *precision* and *recall*, are typically used to measure the performance of a classification model. Precision is defined as the ratio between the predicted true positives and the total predicted positives, and thus it indicates the purity of predicted positives. On the other hand, recall is defined as the ratio between the predicted true positives and the total true positives, and thus it tells us what percentage of positive examples are correctly classified as positive.

$$Precision = \frac{TP}{TP+FP} \quad (10.17)$$

$$Recall = \frac{TP}{TP+FN} \quad (10.18)$$

Since the objective of object detection is not only to correctly classify the object (or objects) in the image but to also find where in the image it is located, we need to consider both the class and location of the objects in formulating precision and recall.

The computation of the overall mAP is based on a test dataset and the predictions of the model on the dataset. The test dataset consists of multiple images along with the ground truth bounding boxes. The predictions of the model on each image are the predicted bounding boxes, each of which includes the box location, confidence score and object class.

In object detection, precision and recall are calculated class-wisely. Let us consider one class in an image. A predicted box is counted as a *true positive* only if it meets the following two requirements: 1) there exists a ground truth box, whose IOU with the predicted box is greater than a predefined IOU threshold; and 2) no other predicted box has a larger IOU with this ground truth box than this predicted box. Requirement 1 guarantees the location of predicted box is close enough to the ground truth box while requirement 2 makes sure that each object can be predicted by no more than one box. If a predicted box is not a true positive, then it is a *false positive*. The total number of *false negatives* (FN) is the difference between the total number of ground truth boxes and the total number of true positives (TP).

As an example, Fig.10.22 illustrates the predicted bounding boxes (dashed line) versus ground truth boxes (solid line) in an image for a particular class, given a confidence threshold. In the image there are 6 objects (GT1, GT2, ..., GT6) to be detected, but the model delivers all predicted boxes (B1, B2, ..., B8) whose confidence scores exceeds the confidence threshold (e.g., 0.3). With an IOU threshold 0.5 (a default value), four predicted boxes B1, B4, B5 and B7 are considered as true positive. The rest of the predicted boxes (B2, B3, B6, B8) are considered as false positive. Note that B6 is considered as a false positive because the GT4 has been matched by a better predicted box B5. In this image, we have TP=4, FP=4, and FN=2 (two objects GT2 and GT6 are not correctly detected), and thus the precision and recall are equal to 4/8 and 4/6.

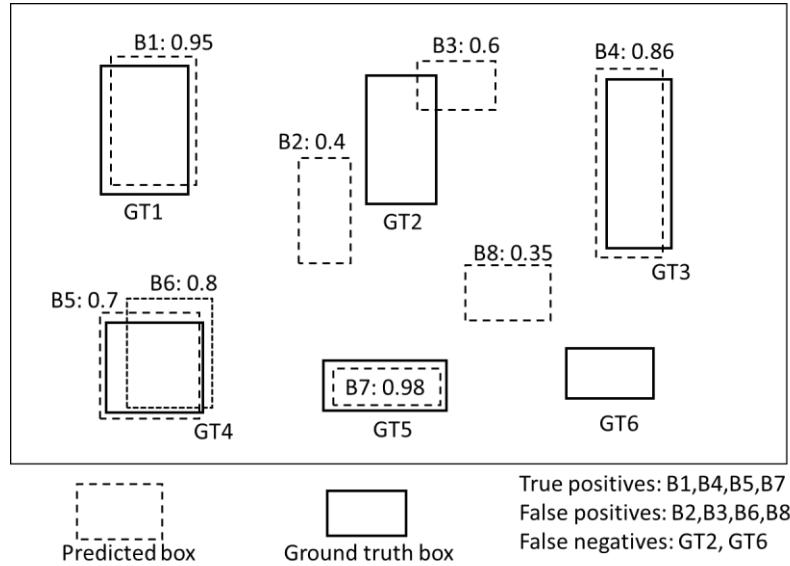


Fig.10.22 Examples of predicted boxes versus ground truth boxes in an image

10.6.2 Calculate mean average precision

If we change the confidence threshold, the number of predicted boxes is expected to change in general, and thus the precision and the recall will change accordingly. A metric, called *average precision* (AP) for a given class, is defined as the average of its precisions over recalls. To calculate

AP, we plot the curve of precision versus recall by changing the confidence threshold, and compute the area under the curve (AUC) as AP.

We use an example illustrated in Fig.10.23 to demonstrate the steps for calculating the AP for a given class. Suppose the test dataset is composed of two images (more images in real situations) and there are 12 ground truth boxes (solid boxes) and 13 predicted bounding boxes (dashed boxes, labeled as A, B, ...) for a given class. Each predicted bounding box is attached by a confidence score.

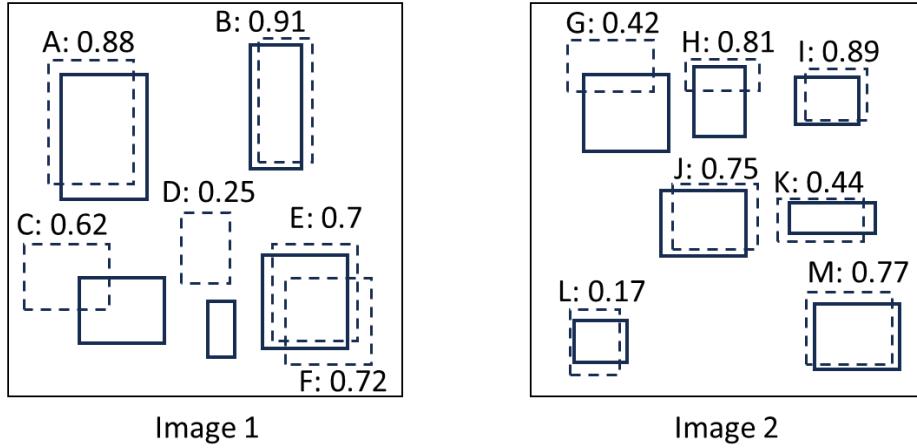


Fig.10.23 ground truth and predictions for a 2-image dataset

A) Plot the precision-recall curve.

To plot the precision-recall curve, we rank all predicted boxes by the confidence score, as shown in Table 10.4. At a particular index, precision is the proportion of all predicted boxes up to the index which are true positive, while recall is the proportion of all ground truth boxes which have been correctly detected up to the index. To calculate precision and recall, we label each box as either TP (true positive) or FP (false positive) by two separate columns and compute the accumulated TP and FP for each index. At each index, the precision is equal to the accumulated TP divided by the index. The recall is equal to the accumulated TP divided by the total number of ground truth boxes (e.g., 12 in this case). The curve of precision-recall is plotted in Fig.10.24.

Table 10.4 Calculate precision and recall

index	image	prediction	confidence	TP	acc TP	FP	acc FP	precision	recall
1	1 B		0.91	1	1	0	0	1	0.083333
2	2 I		0.89	1	2	0	0	1	0.166667
3	1 A		0.88	1	3	0	0	1	0.25
4	2 H		0.81	0	3	1	1	0.75	0.25
5	2 M		0.77	1	4	0	1	0.8	0.333333
6	2 J		0.75	1	5	0	1	0.833333	0.416667
7	1 F		0.72	0	5	1	2	0.714286	0.416667
8	1 E		0.7	1	6	0	2	0.75	0.5
9	1 C		0.62	0	6	1	3	0.666667	0.5
10	2 K		0.44	1	7	0	3	0.7	0.583333
11	2 G		0.42	0	7	1	4	0.636364	0.583333
12	1 D		0.25	0	7	1	5	0.583333	0.583333
13	2 L		0.17	1	8	0	5	0.615385	0.666667

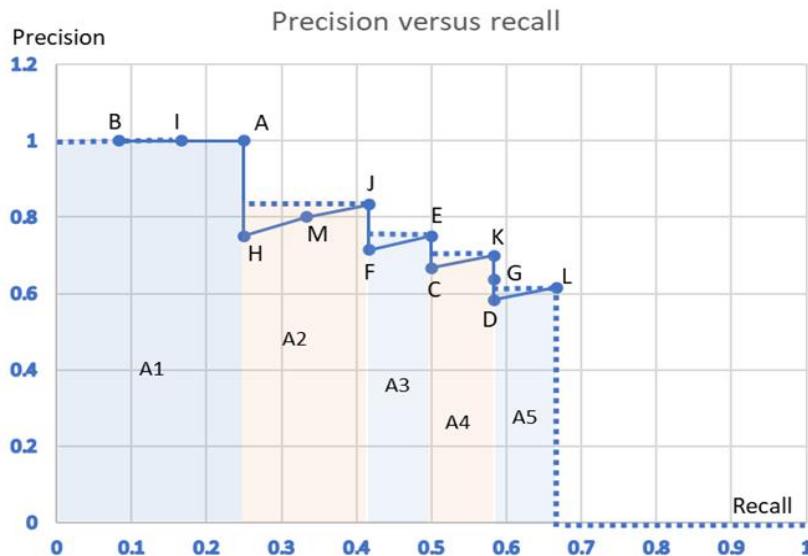


Fig.10.24 Precision-recall curve

B) Interpolate the precision-recall curve.

To simplify the computation of the area under the precision-recall curve, we approximate the curve by the maximum interpolation,

$$P_{interp}(r) = \max_{r':r \geq r} P(\tilde{r}) \quad (10.19)$$

where $P(\tilde{r})$ is the precision measured at recall \tilde{r} . The interpolated precision at recall r takes the maximum precision measured for which the recall \tilde{r} is greater than r . In other words, we start from the last precision value (at the highest recall) and keep moving toward lower recall, as soon as a higher precision value is found update the precision value with the higher precision. The corresponding interpolated precision in the example is plotted as the dotted line in Fig.10.24.

C) Calculate the average precision.

With the interpolated precision-recall curve, we can calculate the AP by summing up the areas of rectangles under the curve. In Fig.10.24, the AP is calculated as

$$AP = A1 + A2 + A3 + A4 + A5 = 1 \times 0.25 + (0.417 - 0.25) \times 0.833 + (0.5 - 0.417) \times 0.75 + (0.583 - 0.5) \times 0.7 + (0.667 - 0.583) \times 0.615 = 0.561$$

An alternative way to calculate the AP is to average the sample points of the interpolated precision at 11 recall points: 0, 0.1, 0.2, 0.3, ..., 1.0,

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 0.9, 1.0\}} P_{interp}(r) \quad (10.20)$$

For the interpolated precision in Fig.10.24, the 11-point AP is calculated as

$$AP = \frac{1}{11} (1 + 1 + 1 + 0.833 + 0.833 + 0.75 + 0.75) = 0.5605$$

In fact, the 11-point average method was introduced in the 2007 PASCAL VOC challenge, and is widely adopted to calculate the AP.

D) Calculate mAP.

The mean average precision (*mAP*) is defined as the average of APs of all classes,

$$mAP = \frac{1}{N} \sum_{k=1}^N AP_k \quad (10.21)$$

where N is the number of classes, AP_k is the average precision of class k . Note that the value of *mAP* depends on the IOU threshold that was used to determine whether a prediction is true positive or false positive. A major metric for PASCAL VOC challenge is the *mAP* at the IOU threshold 0.5, denoted as $mAP@IOU=0.5$.

Since averaging over IoUs rewards models with better localization, the primary metric for COCO challenge is extended to multiple IOU thresholds, and defined as the average mAP across all 10 IoU thresholds 0.50: 0.05: 0.95 (starting with 0.50, ending at 0.95 with step size 0.05),

$$mAP@IOU=[0.5:0.05:0.95] = \frac{1}{10} \sum_{i=0}^9 mAP@IOU=0.5+0.05\times i \quad (10.22)$$

As an example, YOLO v3 achieves the following performance on COCO dataset: $mAP@IOU=[0.5:0.05:0.95] = 0.33$, $mAP@IOU=0.5 = 0.579$, $mAP@IOU=0.75 = 0.344$.

Summary

This chapter describes a popular object detection algorithm – YOLO v1 and its two subsequent versions (v2 and v3). Their architectures have been presented in an evolved order so that we can understand how the performance has been improved through different versions. The key idea of YOLO algorithm is to apply a deep ConvNet (e.g. Darknet) as the backbone network to extract the image feature at a certain grid resolution, and then use a few Conv layers to predict the bounding box and class.

It is essential to understand the loss function of YOLO for training. The total loss consists of three types of losses: confidence loss, box location loss, and probability loss. However, a negative ground truth sample (no object for an anchor box in a grid cell) only results in confidence loss while the prediction for a positive sample generates all three types of losses. The exact definition and implementation of the loss function is very empirical. In general, the box location loss is usually defined as the mean squared error (MSE), and the probability-related loss is defined as entropy loss.

The implementation of YOLO v3 is detailed in Section 10.5. The exploration of the implementation provides a deep understanding of all concepts presented in the preceding sections. In the end, the metric mAP is presented.

Files: C:\Users\weido\ch10_object\yolov3_weidong.ipynb

Further Reading

Original papers for Yolo v1 – v3

The authors highly recommend the original papers that provide the origin of YOLO in comprehensive perspectives: motivations, architectures, training tricks, and performance. These papers include YOLO (v1) (Redmon, J., et al., 2015), YOLO9000 (v2) (Redmon, J., et al., 2016), YOLO(v3) (Redmon, J., et al., 2018). The author's website (Redmon, J. website) and GitHub (Redmon, J., GitHub) include rich resources related to his original work, such as papers, source codes, and pre-trained models.

Hands-on Implementations

The blog posts (Kathuria, A. 2017a) and the GitHub (Kathuria, A. 2017b) present the detailed implementation of YOLO v3 using pre-trained model. The website (Geeksforgeeks website) provides an implementation of YOLO v3 from scratch by PyTorch, including loss function and training.

Advance Yolo versions

To explore advanced YOLO architectures up to YOLO v8, one can read a review paper (Terven, J., R., et al., 2023) and then narrow down to the original paper of a particular YOLO architecture. The advanced YOLO versions include Yolov4 (Bochkovskiy, A. 2020), Yolov5 (Jocher, G. 2020), YoloR (Wang, C.-Y. 2021), YoloX (Ge, Z. 2021), Yolov6 (Li, C. 2022), Yolov7 (Wang, C.-Y. 2022), Yolov8 (Jocher, G. 2023). AlexeyAB repository (Bochkovskiy, A., GitHub) provides yolov4 source code.

References

- Bochkovskiy, A. (GitHub), <https://github.com/AlexeyAB/darknet>
- Bochkovskiy, A., Wang, C.-Y. and Liao, H.-Y. M. (2020), Yolov4: Optimal speed and accuracy of object detection, [arXiv:2004.10934](https://arxiv.org/abs/2004.10934) [cs.CV]
- Ge, Z., Liu, S., Wang, F., Li, Z. and Sun, J. (2021), Yolox: Exceeding yolo series in 2021, [arXiv:2107.08430](https://arxiv.org/abs/2107.08430) [cs.CV]
- Geeksforgeeks website, <https://www.geeksforgeeks.org/yolov3-from-scratch-using-pytorch/>
- Jocher, G. (2020), YOLOv5 by Ultralytics, <https://github.com/ultralytics/yolov5>
- Jocher, G., Chaurasia, A., and Qiu, J. (2023), YOLO by Ultralytics, <https://github.com/ultralytics/ultralytics>
- Kathuria, A. (2017a), How to implement a YOLO (v3) object detector from scratch in PyTorch. <https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/>
- Kathuria, A. (2017b), GitHub, <https://github.com/ayoshkathuria/pytorch-yolo-v3>
- Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W. et al. (2022), Yolov6: A single-stage, object detection framework for industrial applications, [arXiv:2209.02976](https://arxiv.org/abs/2209.02976) [cs.CV]
- Redmon, J., GitHub, <https://github.com/pjreddie>
- Redmon, J., Website, <https://pjreddie.com/>

- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015) ‘You only look once: Unified, real-time object detection’, [arXiv:1506.02640](https://arxiv.org/abs/1506.02640) [cs.CV]
- Redmon, J. and Farhadi, A. (2016) ‘Yolo9000: Better, faster, stronger’, [arXiv:1612.08242](https://arxiv.org/abs/1612.08242) [cs.CV]
- Redmon, J. and Farhadi, A. (2018) ‘YOLOv3: An Incremental Improvement’, [arXiv:1804.02767](https://arxiv.org/abs/1804.02767) [cs.CV]
- Terven, J., R., and Cordova-Esparza, D., M. (2023), A comprehensive review of YOLO: from YOLOv1 and beyond. <https://arxiv.org/pdf/2304.00501.pdf>
- Wang, C.-Y., Yeh, I.-H. and Liao, H.-Y. M. (2021), You only learn one representation: Unified network for multiple tasks, [arXiv:2105.04206](https://arxiv.org/abs/2105.04206) [cs.CV]
- Wang, C.-Y., Bochkovskiy, A., and H.-Y. M. Liao, H.-Y.M. (2022), Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, [arXiv:2207.02696](https://arxiv.org/abs/2207.02696) [cs.CV]

Exercises

10.1 Download and explore the datasets Pascal VOC 2012 and COCO 2017. Write a report about these two datasets.

10.2 Estimate the number of learnable parameters for models yolo v1, v2, and v3. Using the Python program in Section 10.5, verify your estimate of model yolo v3.

10.3 Suppose an image includes two relevant objects: person and dog, with a ground truth label file as [class x y w h]

```
0 0.3 0.6 0.4 0.8
16 0.5 0.6 0.5 0.3
```

- 1) Manually draw the ground truth boxes in a square that represents the image.
- 2) To compute the loss function, we need to generate a tuple targets = (targets[0], targets[1], targets[3]), where targets[0], targets[1], targets[2] are three tensors for grid scale 13, 26, 52, respectively. Each tensor targets[i] has a shape of ([3, s, s, 6]), as shown in Fig.10.15 for scale s=13. Calculate the value of the targets for this image, i.e., targets[i][anchor_idx, m, n,:], where i=0,1,2, is the grid scale index for scale 13, 26, 52, respectively, anchor_idx =0,1,2 is the anchor box index at grid scale i, (m,n) is the grid cell index.

Anchor box assignment

Grid scale index	Anchor index		
	0	1	2
0 (for 13x13)	(0.28, 0.22)	(0.38, 0.48)	(0.9, 0.78)
1 (for 26x26)	(0.07, 0.15)	(0.15, 0.11)	(0.14, 0.29)
2 (for 52x52)	(0.02, 0.03)	(0.04, 0.07)	(0.08, 0.06)

10.4 Given two images: 000001.jpg and 000017.jpg, and their label files 000001.txt and 000017.txt. (available at the book website, originally downloaded from https://www.kaggle.com/datasets/aladdinpersson/pascal-voc-dataset-used-in-yolov3-video/?select=PASCAL_VOC).

000001.txt:

```
16 0.34135977337110485 0.609 0.4164305949008499 0.262  
0 0.5070821529745043 0.508 0.9745042492917847 0.972
```

000017.txt:

```
0 0.48125 0.3557692307692308 0.1958333333333333 0.3763736263736264  
17 0.5114583333333333 0.565934065934066 0.6520833333333333 0.7087912087912088
```

- 1) Run the program provided in Section 10.5, and detect the objects in two images.
- 2) Add Python code to compute the loss of YOLO v3 model on the two images for a) random weight model and b) pre-trained model. The pre-trained model should have a much smaller loss than the random weight model. (hint: the loss is defined in Section 10.4.2. Since you may need to scale the image to fit the model input size, you need to scale the ground truth boxes as well. For Python implementation, please refer to Geeksforgeeks website, <https://www.geeksforgeeks.org/yolov3-from-scratch-using-pytorch/>).

10.5 Suppose the ground truth bounding boxes for a batch are given as a tensor gtbox with a shape of ([N,6]), where N is the total number of ground truth boxes for the batch, and the format of each ground truth box gtbox[i,:] is

Img_index	x1	y1	x2	y2	Class_index
-----------	----	----	----	----	-------------

The predicted bounding boxes are stored in a tensor bbox with a shape of ([M,7]), where M is the total number of predicted boxes for the batch, and the format of each predicted box bbox[j,:] is

Img_index	x1	y1	x2	y2	Class_index	confidence
-----------	----	----	----	----	-------------	------------

Write a Python program to compute the mAP for a particular IOU threshold.

10.6 project 1. Train yolo v3 on Pascal VOC dataset from scratch.

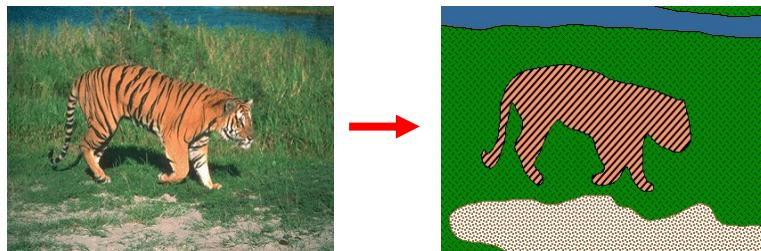
10.7 project 2. Train yolo v3 on a custom dataset.

Materials for Image Segmentation

Image Segmentation



From images to objects



What Defines an Object?

- Subjective problem, but has been well-studied
- Gestalt Laws seek to formalize this
 - proximity, similarity, continuation, closure, common fate
 - see [notes](#) by Steve Joordens, U. Toronto

Image Segmentation

We will consider different methods

Already covered:

- Intelligent Scissors (contour-based)
- Hough transform (model-based)

This week:

- K-means clustering (color-based)
- EM
- Mean-shift
- Normalized Cuts (region-based)

Image histograms

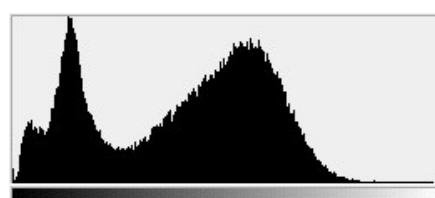
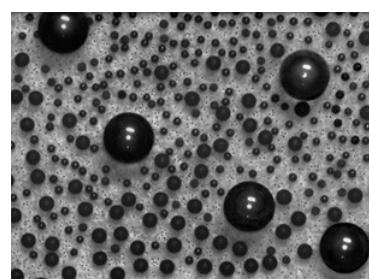


How many “orange” pixels are in this image?

- This type of question answered by looking at the *histogram*
- A histogram counts the number of occurrences of each color
 - Given an image $f[x, y] \rightarrow RGB$
 - The histogram is defined to be

$$H_f[c] = |\{(x, y) \mid f[x, y] = c\}|$$

What do histograms look like?



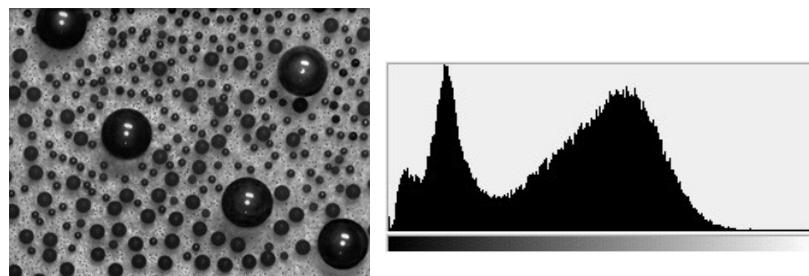
How Many Modes Are There?

- Easy to see, hard to compute

Histogram-based segmentation

Goal

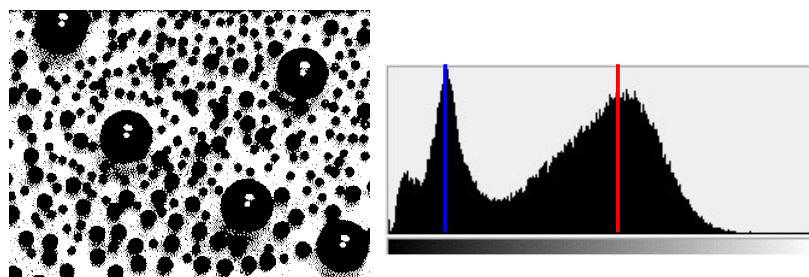
- Break the image into K regions (segments)
- Solve this by reducing the number of colors to K and mapping each pixel to the closest color
 - photoshop demo



Histogram-based segmentation

Goal

- Break the image into K regions (segments)
- Solve this by reducing the number of colors to K and mapping each pixel to the closest color
 - photoshop demo

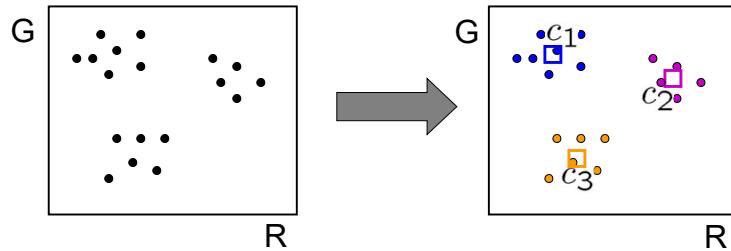


Here's what it looks like if we use two colors

Clustering

How to choose the representative colors?

- This is a clustering problem!



Objective

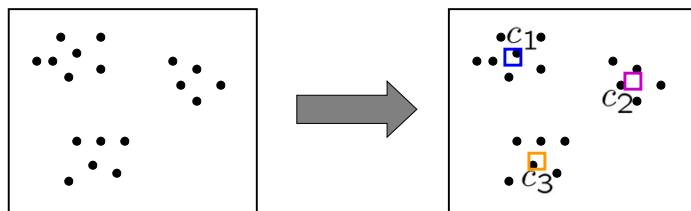
- Each point should be as close as possible to a cluster center
 - Minimize sum squared distance of each point to closest center

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$

Break it down into subproblems

Suppose I tell you the cluster centers c_i

- Q: how to determine which points to associate with each c_i ?
- A: for each point p , choose closest c_i



Suppose I tell you the points in each cluster

- Q: how to determine the cluster centers?
- A: choose c_i to be the mean of all points in the cluster

K-means clustering

K-means clustering algorithm

1. Randomly initialize the cluster centers, c_1, \dots, c_K
2. Given cluster centers, determine points in each cluster
 - For each point p , find the closest c_i . Put p into cluster i
3. Given points in each cluster, solve for c_i
 - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2

Java demo: <http://www.cs.mcgill.ca/~bonnef/project.html>

Properties

- Will always converge to *some* solution
- Can be a “local minimum”
 - does not always find the global minimum of objective function:

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$

Probabilistic clustering

Basic questions

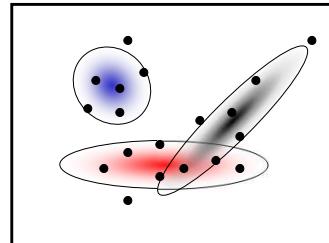
- what's the probability that a point x is in cluster m ?
- what's the shape of each cluster?

K-means doesn't answer these questions

Basic idea

- instead of treating the data as a bunch of points, assume that they are all generated by sampling a continuous function
- This function is called a **generative model**
 - defined by a vector of parameters θ

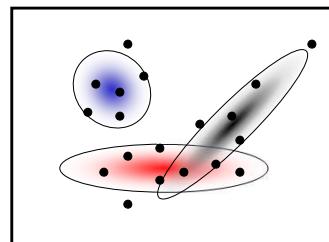
Mixture of Gaussians



One generative model is a mixture of Gaussians (MOG)

- K Gaussian blobs with means μ_b , covariance matrices V_b , dimension d
 - blob b defined by: $P(x|\mu_b, V_b) = \frac{1}{\sqrt{(2\pi)^d |V_b|}} e^{-\frac{1}{2}(x-\mu_b)^T V_b^{-1} (x-\mu_b)}$
- blob b is selected with probability α_b
- the likelihood of observing \mathbf{x} is a weighted mixture of Gaussians
$$P(x|\theta) = \sum_{b=1}^K \alpha_b P(x|\theta_b)$$
- where $\theta = [\mu_1, \dots, \mu_n, V_1, \dots, V_n]$

Expectation maximization (EM)



Goal

- find blob parameters θ that maximize the likelihood function:

$$P(\text{data}|\theta) = \prod_x P(x|\theta)$$

Approach:

1. E step: given current guess of blobs, compute ownership of each point
2. M step: given ownership probabilities, update blobs to maximize likelihood function
3. repeat until convergence

EM details

E-step

- compute probability that point \mathbf{x} is in blob i, given current guess of $\boldsymbol{\theta}$

$$P(b|x, \mu_b, V_b) = \frac{\alpha_b P(x|\mu_b, V_b)}{\sum_{i=1}^K \alpha_i P(x|\mu_i, V_i)}$$

M-step

- compute probability that blob b is selected

$$\alpha_b^{new} = \frac{1}{N} \sum_{i=1}^N P(b|x_i, \mu_b, V_b) \quad N \text{ data points}$$

- mean of blob b

$$\mu_b^{new} = \frac{\sum_{i=1}^N x_i P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^N P(b|x_i, \mu_b, V_b)}$$

- covariance of blob b

$$V_b^{new} = \frac{\sum_{i=1}^N (x_i - \mu_b^{new})(x_i - \mu_b^{new})^T P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^N P(b|x_i, \mu_b, V_b)}$$

EM demos

<http://www.cs.ucsd.edu/users/ibayrakt/java/em/>

<http://www.dreier.cc/index.php?topic=downloads&sub=em>

Applications of EM

Turns out this is useful for all sorts of problems

- any clustering problem
- any model estimation problem
- missing data problems
- finding outliers
- segmentation problems
 - segmentation based on color
 - segmentation based on motion
 - foreground/background separation
- ...

Problems with EM

Local minima

Need to know number of segments

Need to choose generative model

Mean-shift for image segmentation

Useful to take into account spatial information

- instead of (R, G, B), run in (R, G, B, x, y) space
- D. Comaniciu, P. Meer, Mean shift analysis and applications, *7th International Conference on Computer Vision*, Kerkyra, Greece, September 1999, 1197-1203.
 - <http://www.caip.rutgers.edu/~rui/research/papers/pdf/spatmsft.pdf>

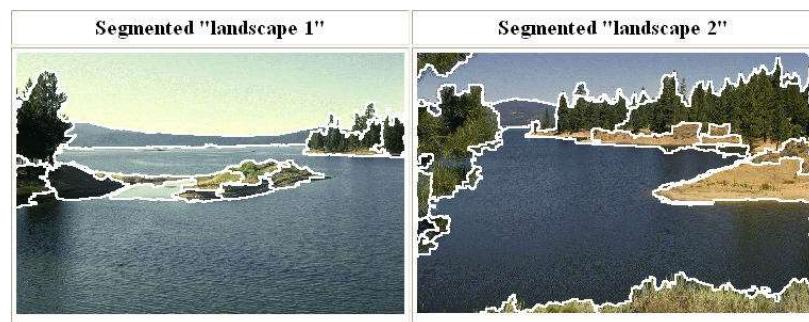


More Examples: http://www.caip.rutgers.edu/~comanici/segm_images.html

Mean shift segmentation

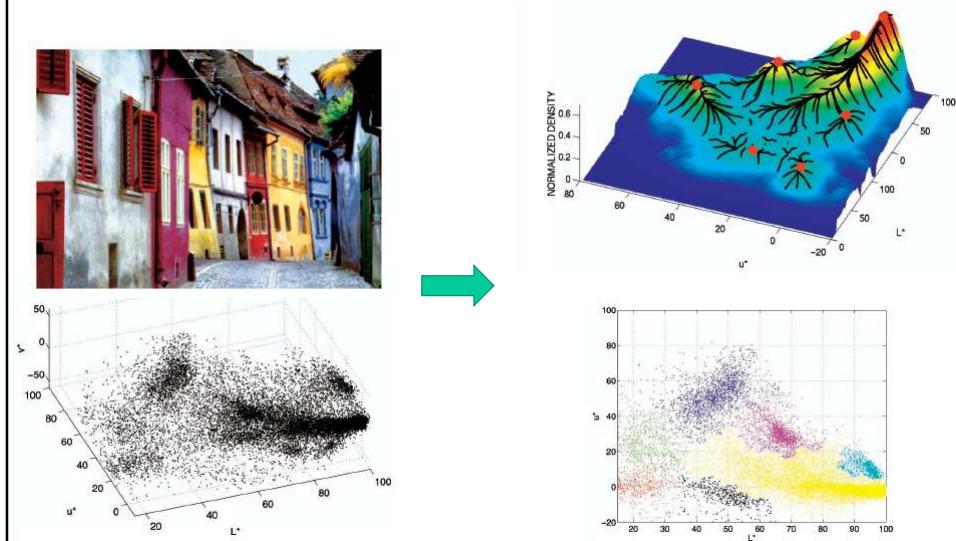
D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

Versatile technique for clustering-based segmentation



Mean shift algorithm

Try to find *modes* of this non-parametric density



Kernel density estimation

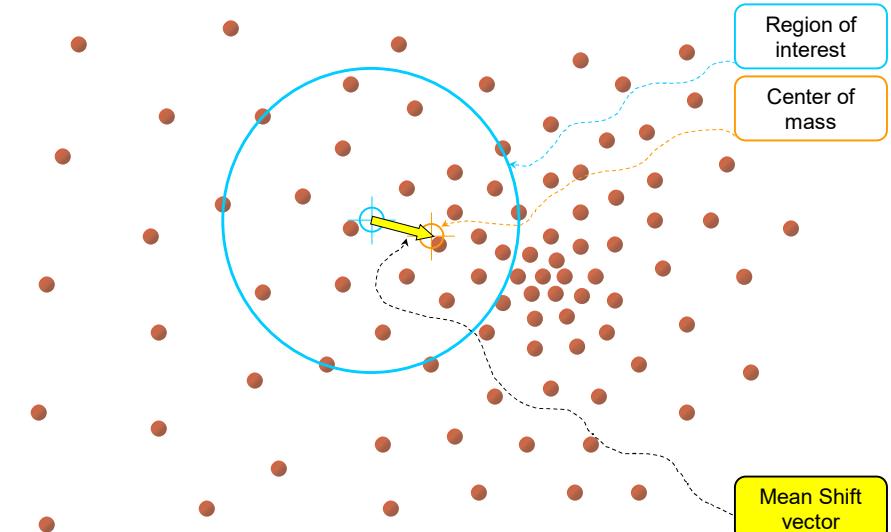
Kernel density estimation function

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$$

Gaussian kernel

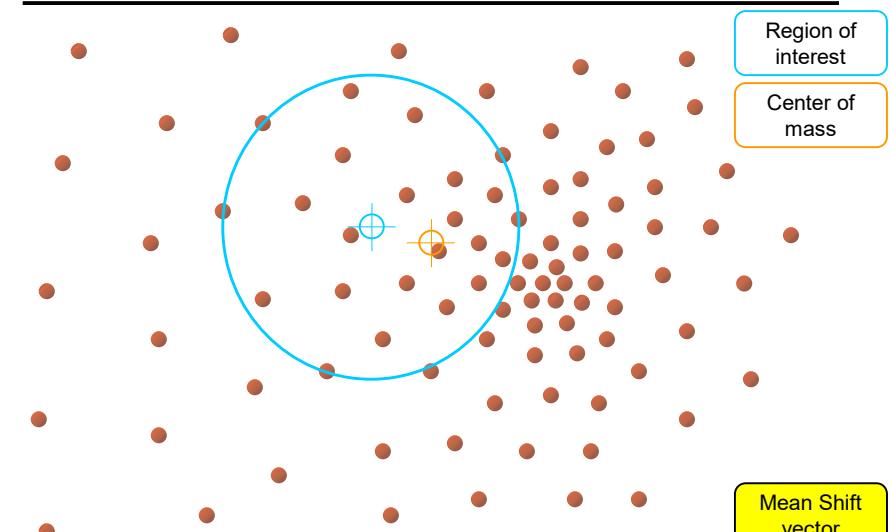
$$K\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}.$$

Mean shift



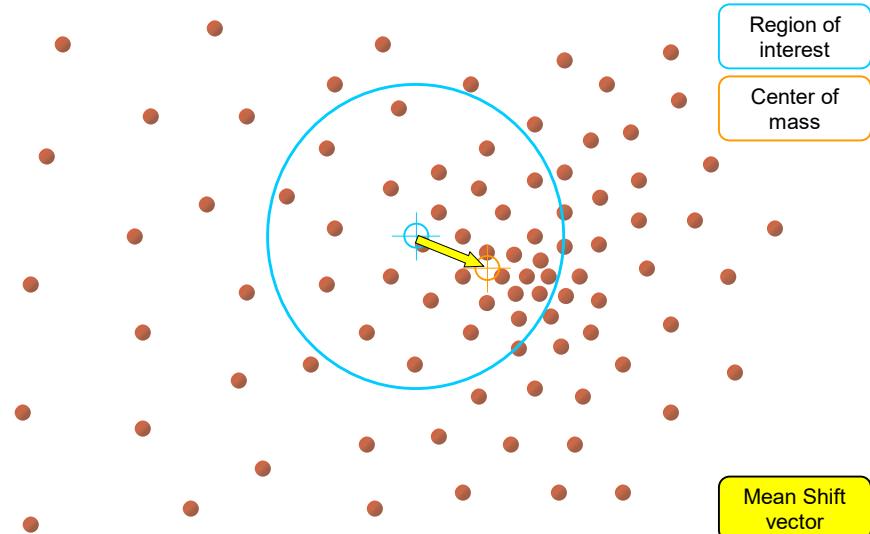
Slide by Y. Ukrainitz & B. Sarel

Mean shift



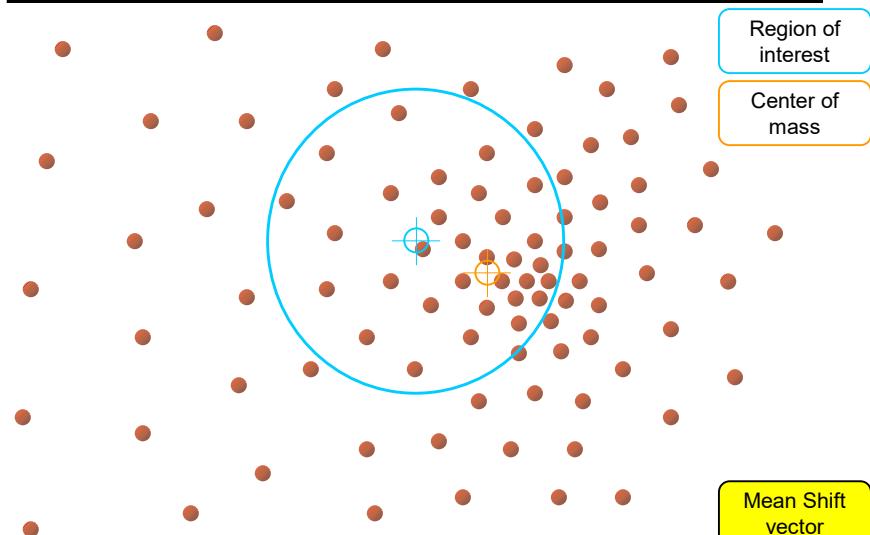
Slide by Y. Ukrainitz & B. Sarel

Mean shift



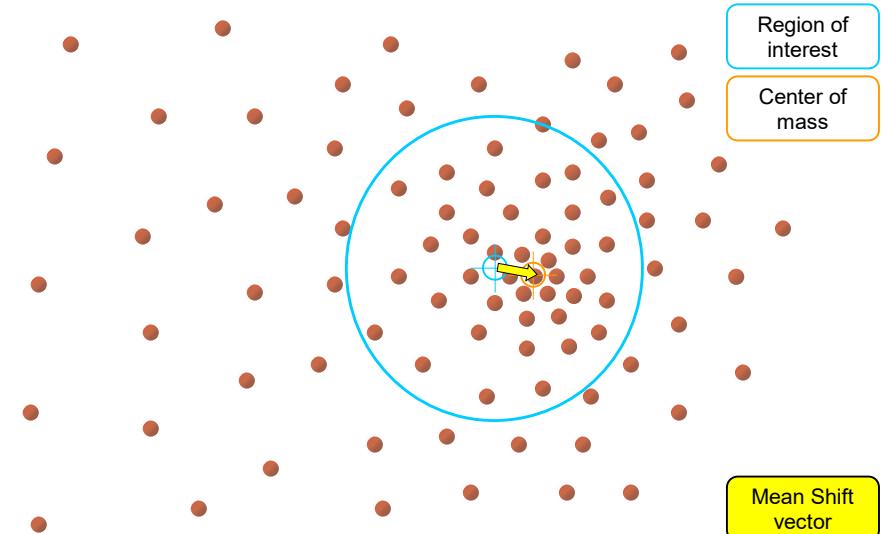
Slide by Y. Ukrainitz & B. Sarel

Mean shift



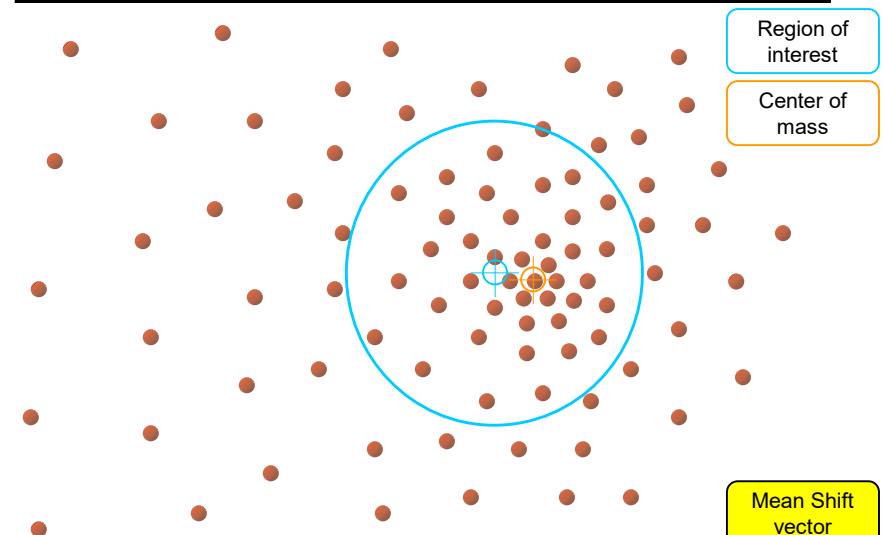
Slide by Y. Ukrainitz & B. Sarel

Mean shift



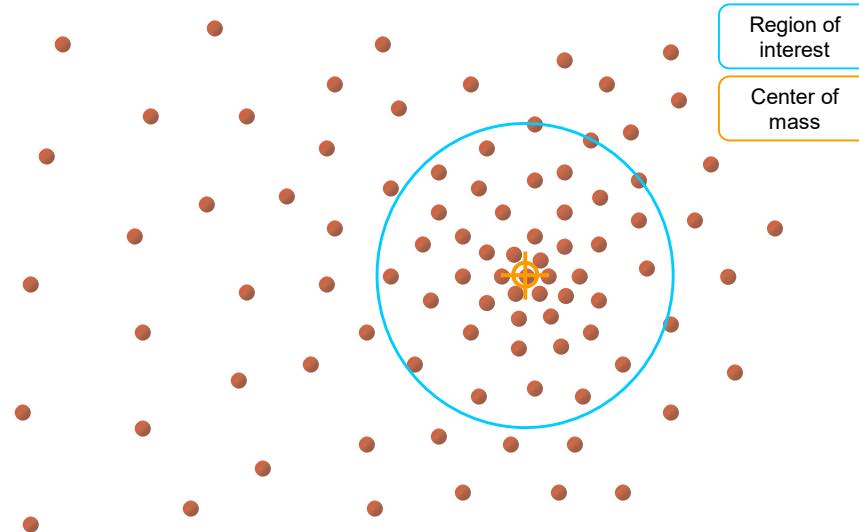
Slide by Y. Ukrainitz & B. Sarel

Mean shift



Slide by Y. Ukrainitz & B. Sarel

Mean shift



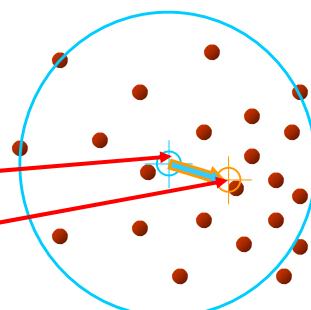
Slide by Y. Ukrainitz & B. Sarel

Computing the Mean Shift

Simple Mean Shift procedure:

- Compute mean shift vector
- Translate the Kernel window by $\mathbf{m}(\mathbf{x})$

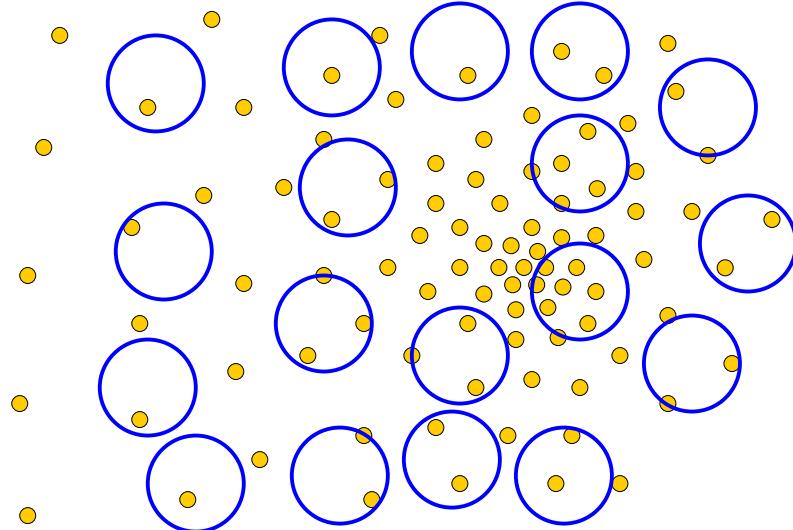
$$\mathbf{m}(\mathbf{x}) = \begin{bmatrix} \sum_{i=1}^n \mathbf{x}_i g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h}\right) \\ \sum_{i=1}^n g\left(\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h}\right) \end{bmatrix}$$



$$\mathbf{g}(\mathbf{x}) = -k'(\mathbf{x})$$

Slide by Y. Ukrainitz & B. Sarel

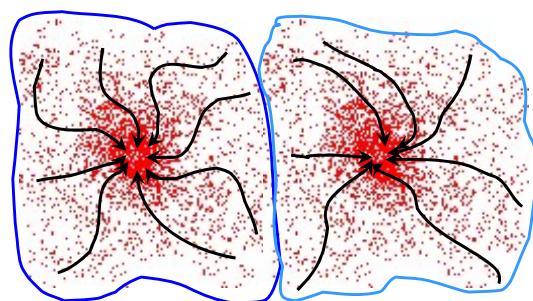
Real Modality Analysis



Attraction basin

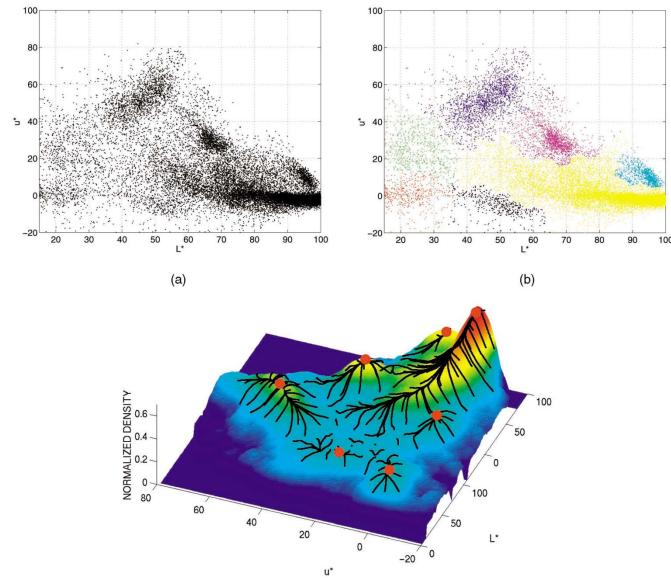
Attraction basin: the region for which all trajectories lead to the same mode

Cluster: all data points in the attraction basin of a mode

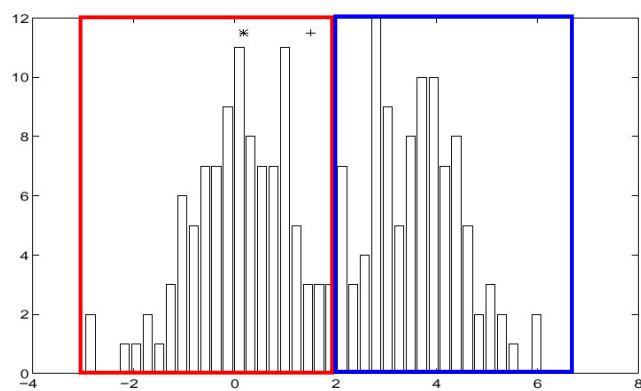


Slide by Y. Ukrainitz & B. Sarel

Attraction basin



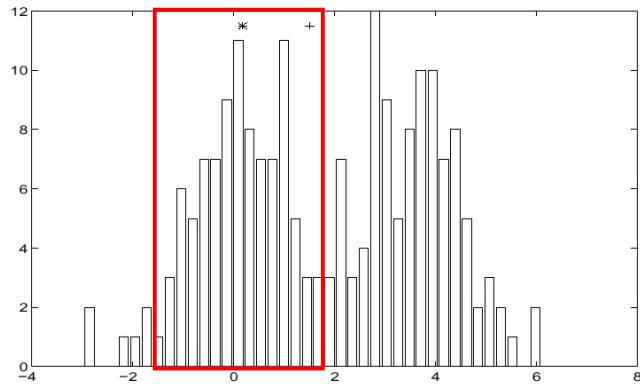
Finding Modes in a Histogram



How Many Modes Are There?

- Easy to see, hard to compute

Mean Shift [Comaniciu & Meer]



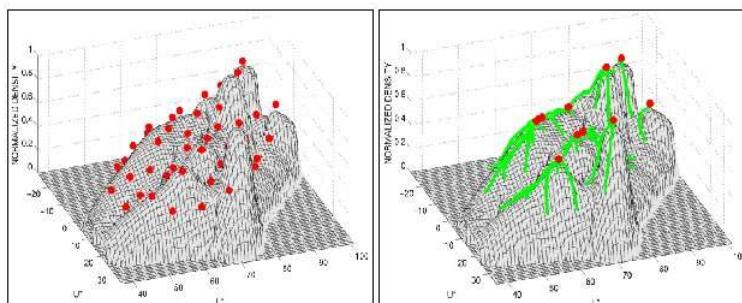
Iterative Mode Search

1. Initialize random seed, and window W
2. Calculate center of gravity (the “mean”) of W : $\sum_{x \in W} xH(x)$
3. Translate the search window to the mean
4. Repeat Step 2 until convergence

Mean-Shift

Approach

- Initialize a window around each point
- See where it shifts—this determines which segment it’s in
- Multiple points will shift to the same segment



Mean shift trajectories

Mean shift clustering

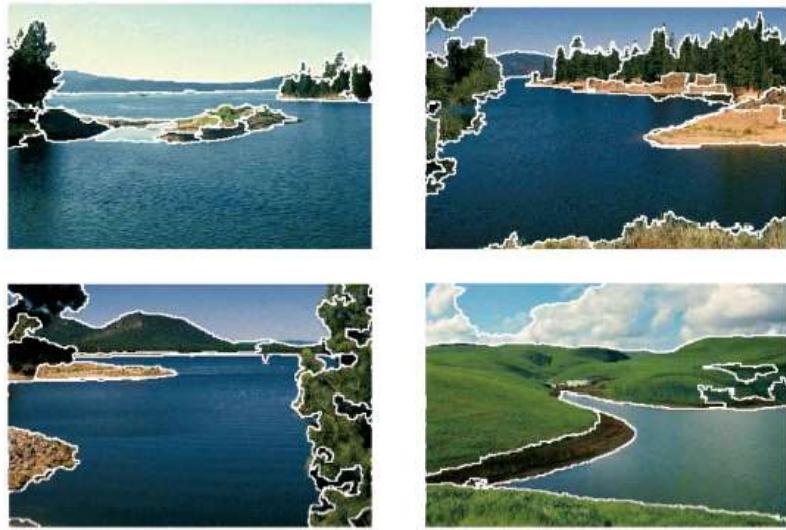
The mean shift algorithm seeks *modes* of the given set of points

1. Choose kernel and bandwidth
2. For each point:
 - a) Center a window on that point
 - b) Compute the mean of the data in the search window
 - c) Center the search window at the new mean location
 - d) Repeat (b,c) until convergence
3. Assign points that lead to nearby modes to the same cluster

Mean shift segmentation results



<http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html>



<http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html>

Mean-shift: other issues

Speedups

- Binned estimation
- Fast search of neighbors
- Update each window in each iteration (faster convergence)

Other tricks

- Use kNN to determine window sizes adaptively

Lots of theoretical support

D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

Mean shift pros and cons

Pros

- Good general-practice segmentation
- Flexible in number and shape of regions
- Robust to outliers

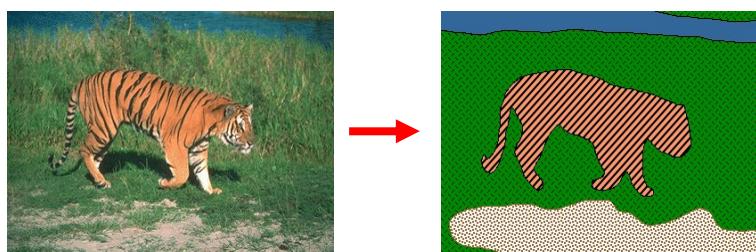
Cons

- Have to choose kernel size in advance
- Not suitable for high-dimensional features

When to use it

- Oversegmentation
- Multiple segmentations
- Tracking, clustering, filtering applications

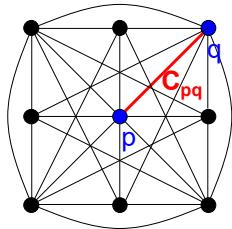
Region-based segmentation



Color histograms don't take into account spatial info

- Gestalt laws point out importance of spatial grouping
 - proximity, similarity, continuation, closure, common fate
- Suggests that *regions* are important

Images as graphs



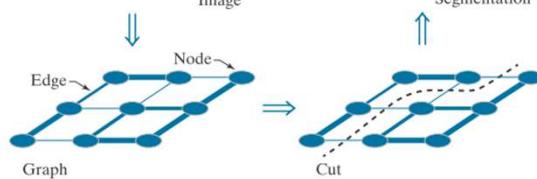
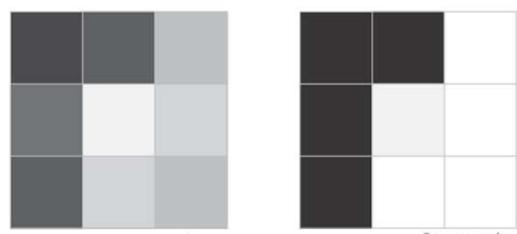
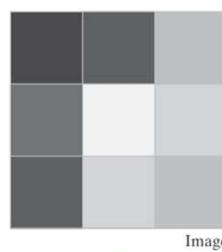
Fully-connected graph

- node for every pixel
- link between **every** pair of pixels, p, q
- cost C_{pq} for each link
 - C_{pq} measures *similarity*
 - » similarity is *inversely proportional* to difference in color and position
 - » this is different than the costs for intelligent scissors

Images as graphs

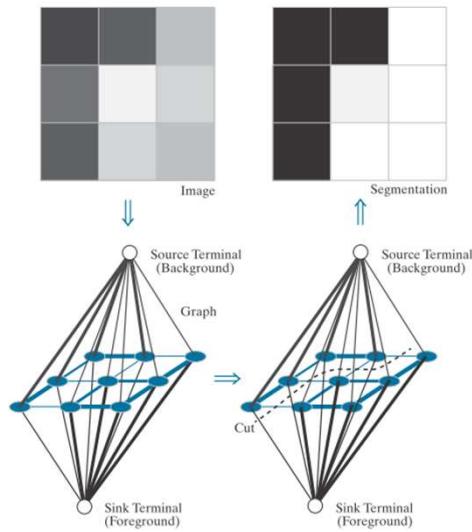
a
b
c
d

FIGURE 10.53
(a) A 3×3 image.
(c) A corresponding graph.
(d) Graph cut.
(e) Segmented image.

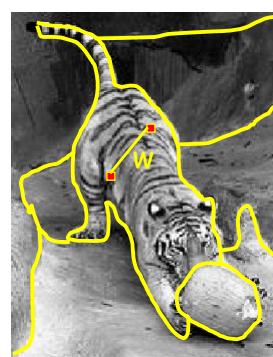
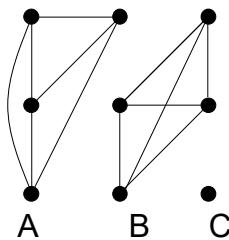


Images as graphs

a b
c d
FIGURE 10.54
(a) Same image
as in Fig. 10.53(a).
(c) Corresponding
graph and terminal
nodes. (d) Graph
cut. (b) Segmented
image.



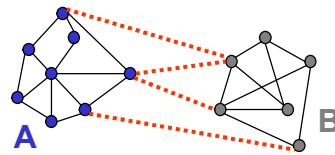
Segmentation by Graph Cuts



Break Graph into Segments

- Delete links that cross between segments
- Easiest to break links that have high cost
 - similar pixels should be in the same segments
 - dissimilar pixels should be in different segments

Cuts in a graph



Link Cut

- set of links whose removal makes a graph disconnected
- cost of a cut:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

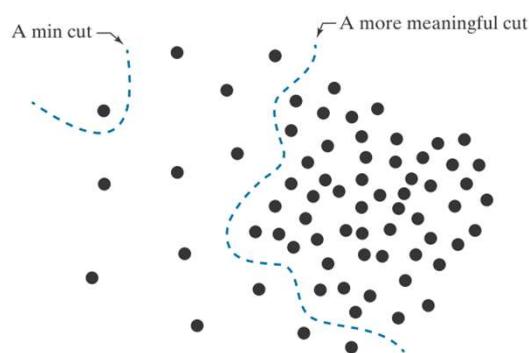
Find minimum cut

- gives you a segmentation
- fast algorithms exist for doing this

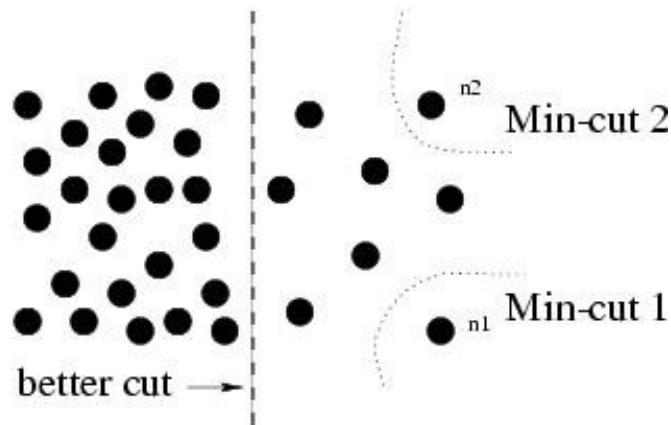
But min cut is not always the best cut...

FIGURE 10.55

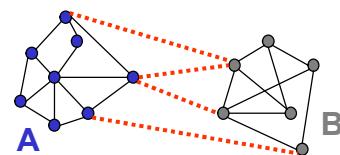
An example showing how a min cut can lead to a meaningless segmentation. In this example, the similarity between pixels is defined as their spatial proximity, which results in two distinct regions.



But min cut is not always the best cut...



Cuts in a graph



Normalized Cut

- a cut penalizes large segments
- fix by normalizing for size of segments

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

$$assoc(A, V) = \sum_{u \in A, z \in V} w(u, z) \quad | \quad assoc(B, V) = \sum_{v \in B, z \in V} w(v, z)$$

Cuts in a graph

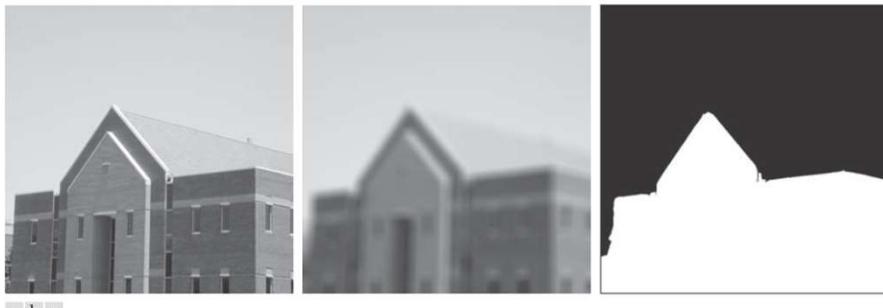
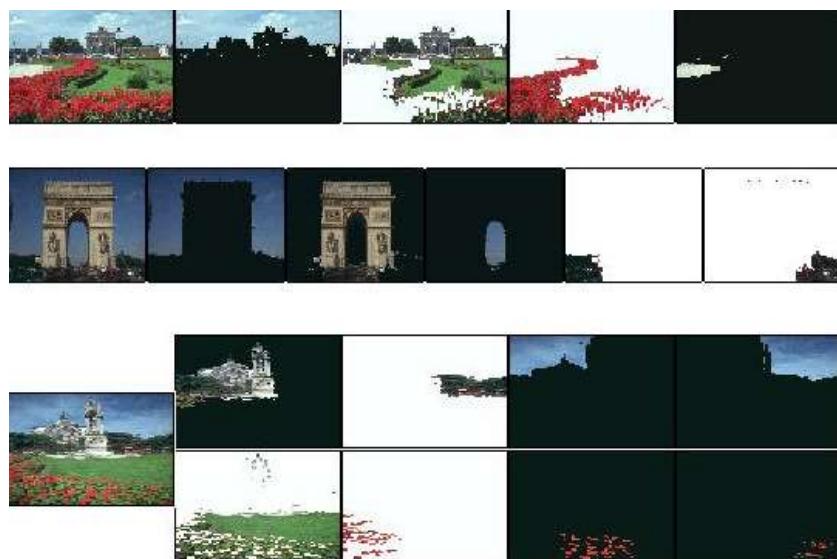


FIGURE 10.56 (a) Image of size 600×600 pixels. (b) Image smoothed with a 25×25 box kernel. (c) Graph cut segmentation obtained by specifying two regions.

Color Image Segmentation



Normalize Cut in Matrix Form

\mathbf{W} is the cost matrix : $\mathbf{W}(i, j) = c_{i,j}$;

\mathbf{D} is the sum of costs from node i : $\mathbf{D}(i, i) = \sum_j \mathbf{W}(i, j)$; $\mathbf{D}(i, j) = 0$

Can write normalized cut as:

$$Ncut(A, B) = \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T \mathbf{D} \mathbf{y}}, \text{ with } \mathbf{y}_i \in \{1, -b\}, \mathbf{y}^T \mathbf{D} \mathbf{1} = 0.$$

- Solution given by “generalized” eigenvalue problem:

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$$

- Solved by converting to standard eigenvalue problem:

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{z} = \lambda \mathbf{z}, \text{ where } \mathbf{z} = \mathbf{D}^{\frac{1}{2}}\mathbf{y}$$

- optimal solution corresponds to second smallest eigenvector

- **for more details, see**

- J. Shi and J. Malik, [Normalized Cuts and Image Segmentation, IEEE Conf.](#)

- Computer Vision and Pattern Recognition(CVPR), 1997

- <http://www.cs.washington.edu/education/courses/455/03wi/readings/Ncut.pdf>

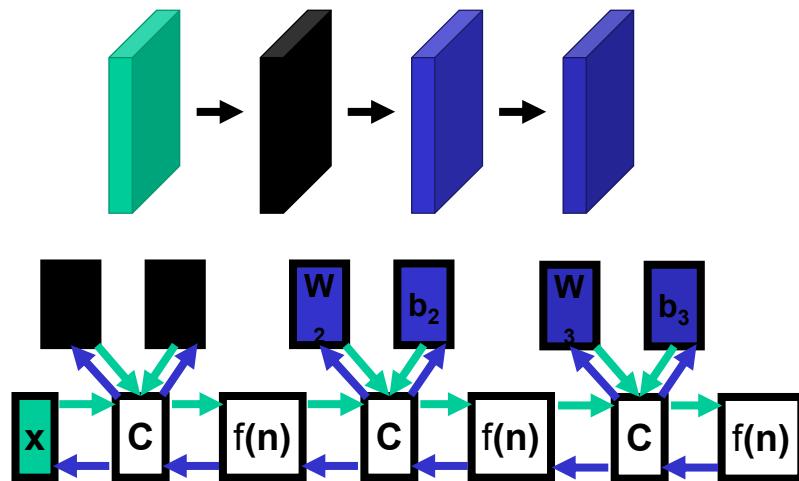
Image Segmentation -2

Slide Source from

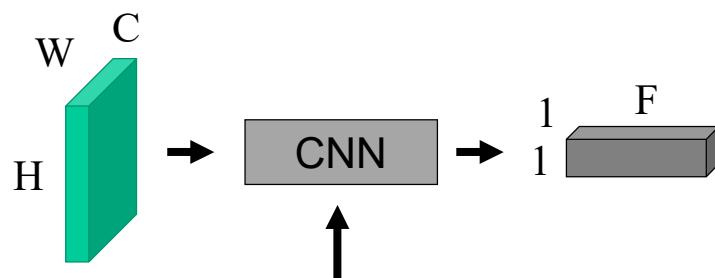
EECS 442 – David Fouhey

Winter 2023, University of Michigan

Convolutional Neural Network (CNN)

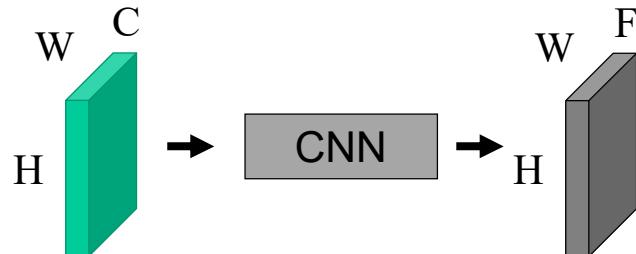


CNN



Function of the image that is parameterized by the convolutional filter weights and biases. We design the form of the function and fit the parameters to data.

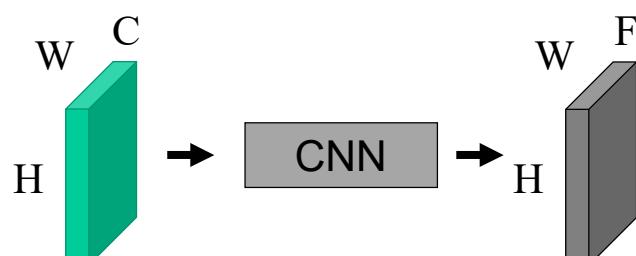
Now



Convert $H \times W$ image into a F -dimensional vector

Which pixels in this image are a cat?
How far is each pixel away from the camera?
Which pixels of this image are fake?

Semantic Segmentation



Today's Running Example

- Predict F -dimensional vector representing probability of each of F classes at every pixel
 - Loss computed/backprop'd at *every* pixel.

Semantic Segmentation

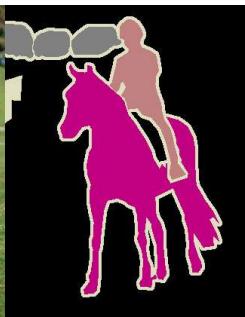
Each pixel has label
Usually visualized by colors.

Note: don't distinguish between object *instances*

Input



Label



Input



Label



Image Credit: Everingham et al. Pascal VOC 2012.

Semantic Segmentation

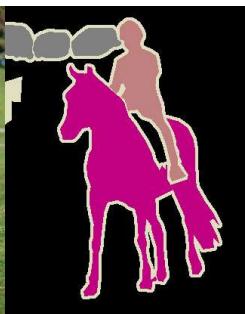
“Semantic”: a usually meaningless word and an indication that someone is trying to trick you.

Meant to indicate here that we’re **naming** things.

Input



Label



Input



Label



Semantic Segmentation

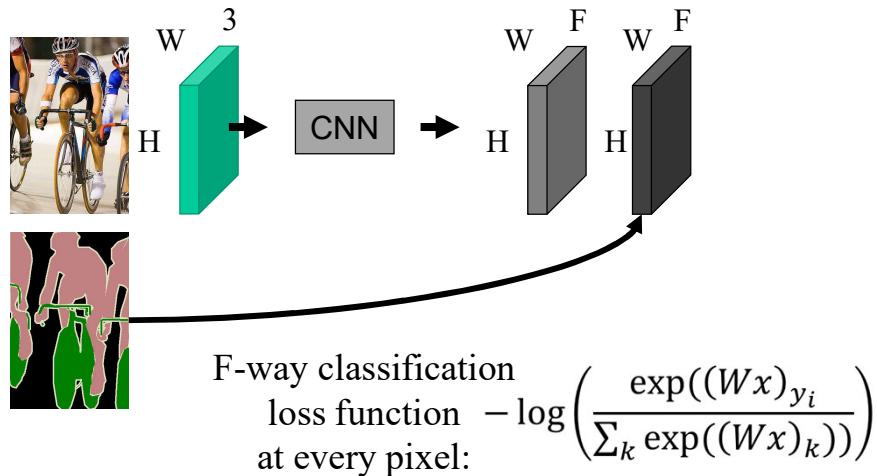


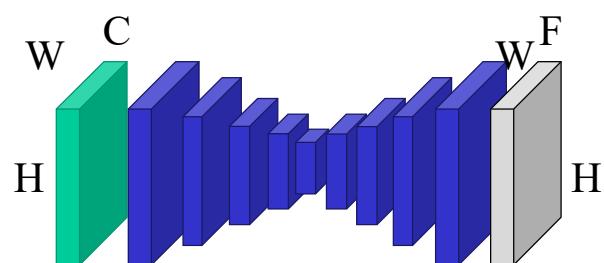
Image Credit: Everingham et al. Pascal VOC 2012.

Encoder-Decoder

Key idea: First **downsample** towards middle of network. Then **upsample** from middle.

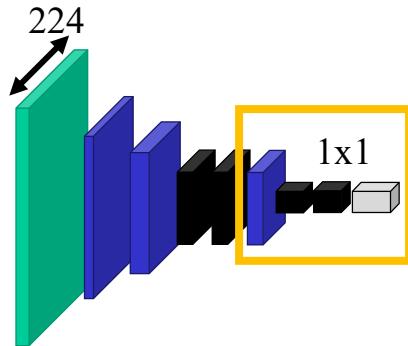
How do we downsample?

Convolutions, pooling



Where Do We Get Parameters?

Convnet that maps
images to vectors



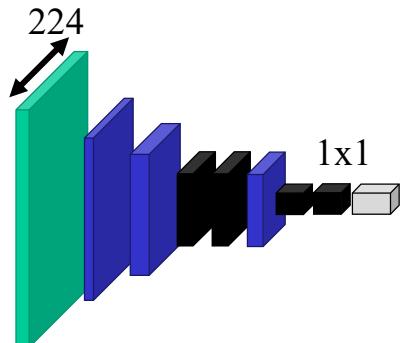
$$\blacksquare * \square \rightarrow \blacksquare$$

Recall that we can rewrite
any vector-vector
operations via 1x1
convolutions

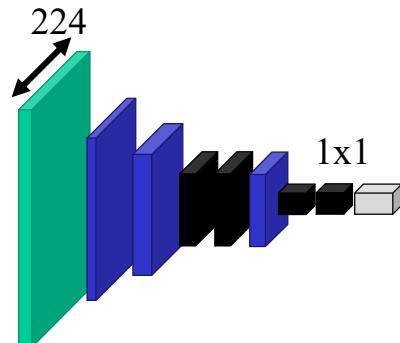
Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Where Do We Get Parameters?

Convnet that maps
images to vectors

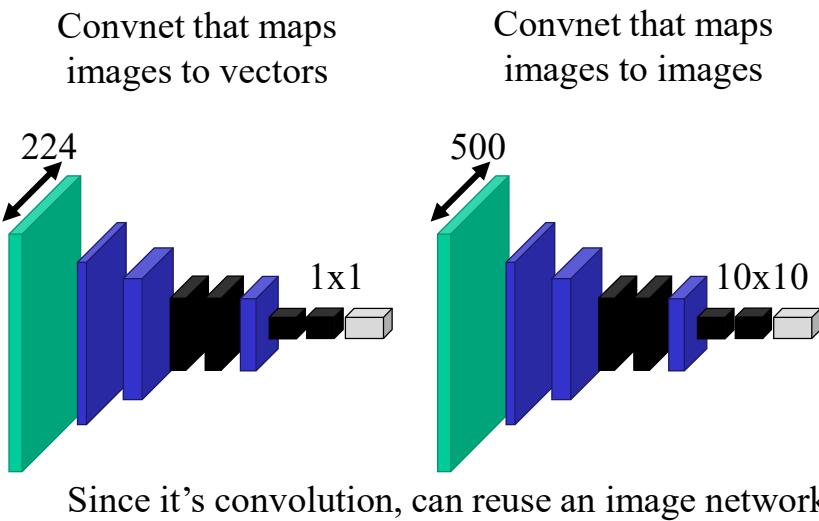


Convnet that maps
images to images

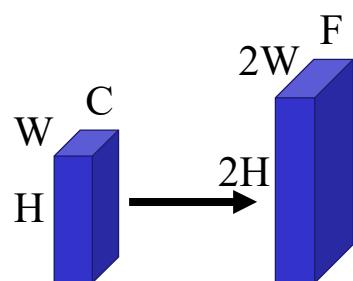


What if we make the input bigger?

Where Do We Get Parameters?

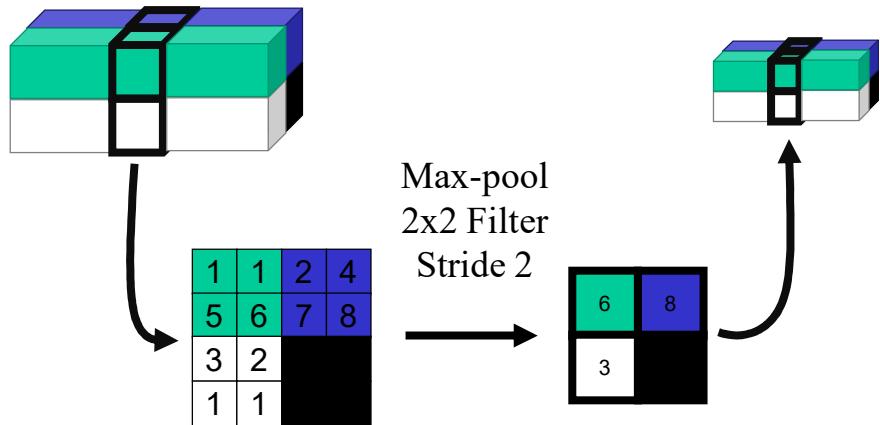


How Do We Upsample?



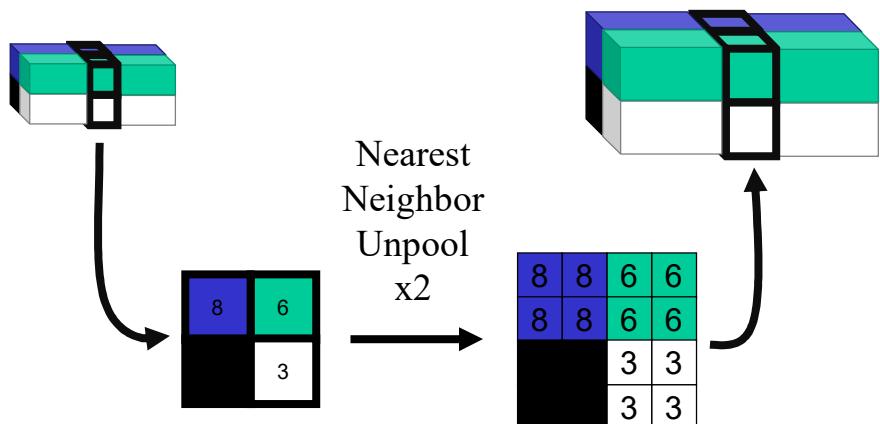
Do the opposite of how we downsample:
1. Pooling → “Unpooling”
2. Convolution → “Transpose Convolution”

Recall: Pooling



Max-pool
2x2 Filter
Stride 2

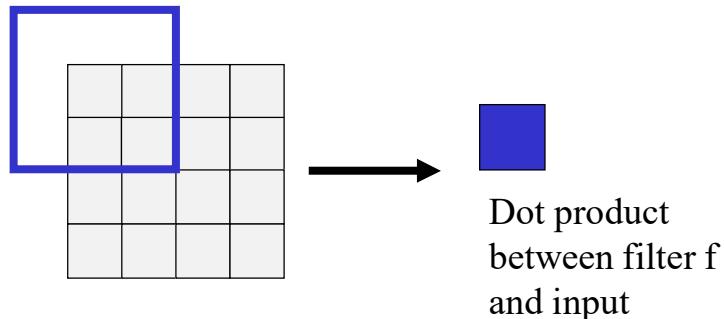
Now: Unpooling



Nearest
Neighbor
Unpool
x2

Recall: Convolution

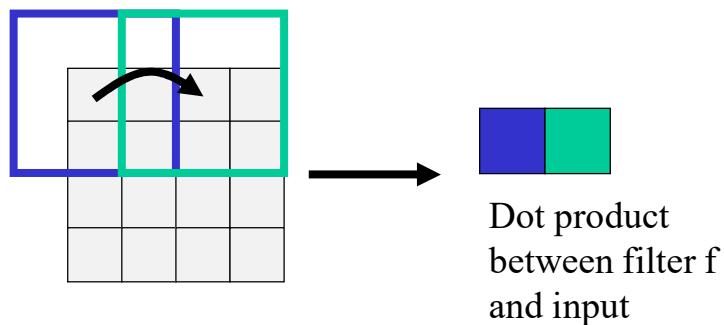
3x3 Convolution, Stride 2, Pad 1



Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Recall: Convolution

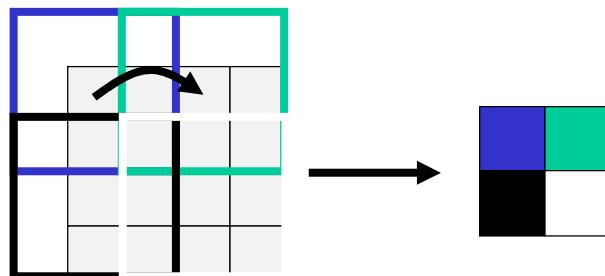
3x3 Convolution, Stride 2, Pad 1



Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Recall: Convolution

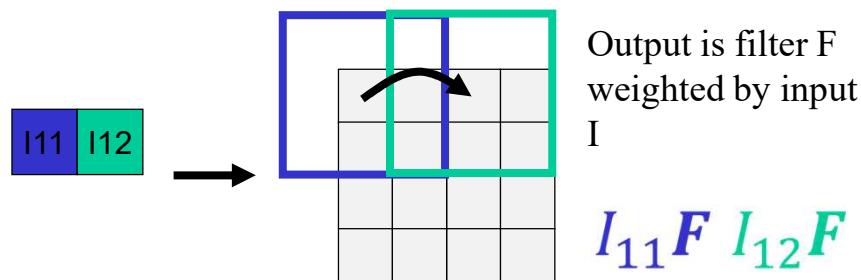
3x3 Convolution, Stride 2, Pad 1



Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Transpose Convolution

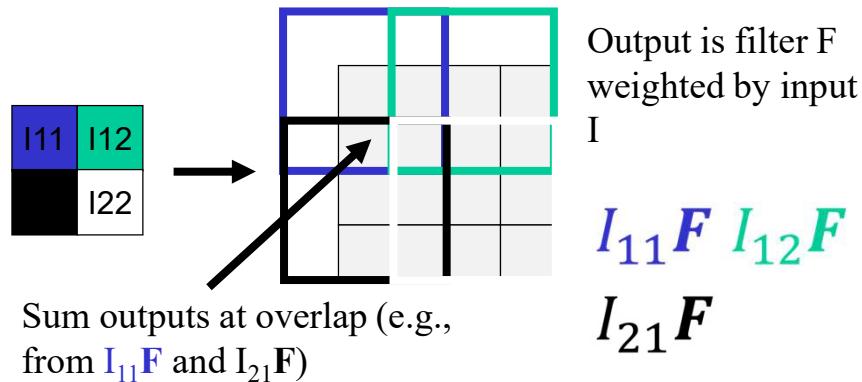
3x3 Transpose Convolution, Stride 2, Pad 1



Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Transpose Convolution

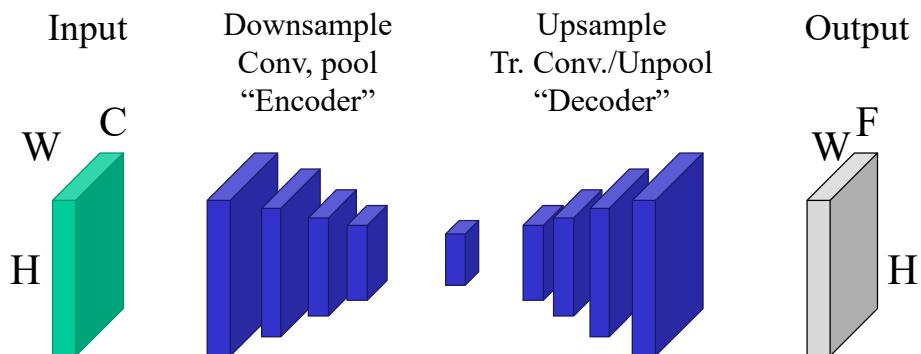
3x3 Transpose Convolution, Stride 2, Pad 1



Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Putting it Together

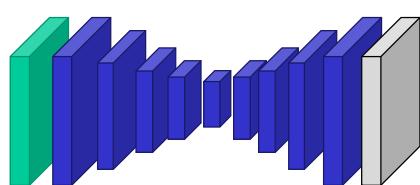
Convolutions + pooling downsample/compress/encode
Transpose convs./unpoolings upsample/uncompress/decode



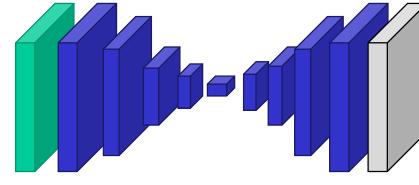
Putting It Together – Block Sizes

- Networks come in lots of forms
- **Don't take any block sizes literally.**
- Often (not always) keep some spatial resolution

Encode to spatially smaller tensor, then decode.

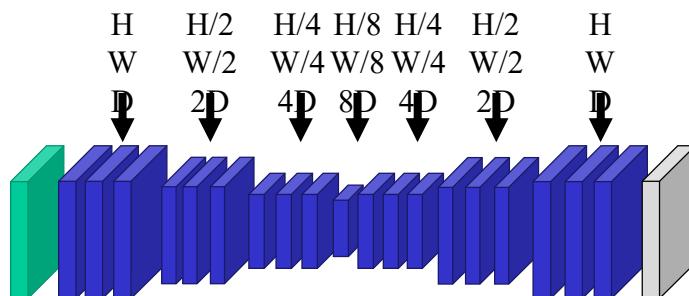


Encode to 1D vector then decode



Putting It Together – Block Sizes

- Often multiple layers at each spatial resolution.
- Often halve spatial resolution and double feature depth every few layers



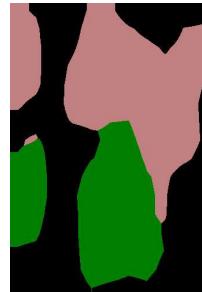
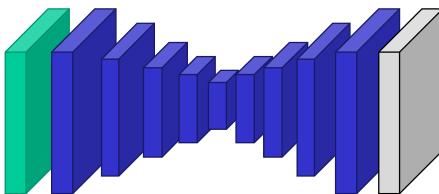
Missing Details

While the output *is* $H \times W$, just upsampling often produces results without details/not aligned with the image.

Why?



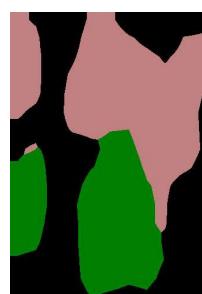
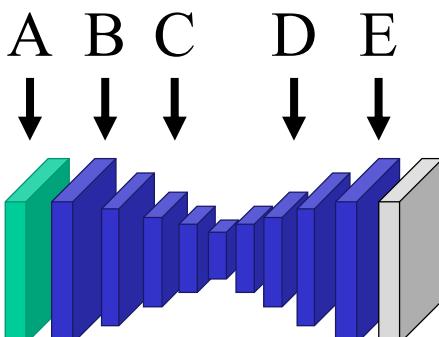
Information about details lost when downsampling!



Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Missing Details

Where is the useful information about the high-frequency details of the image?



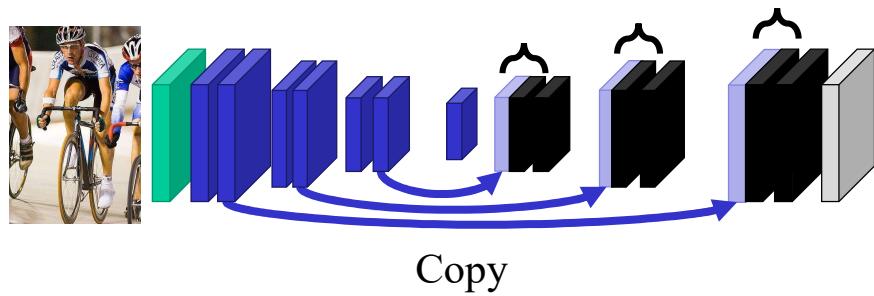
Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

Missing Details

How do you send details forward in the network?

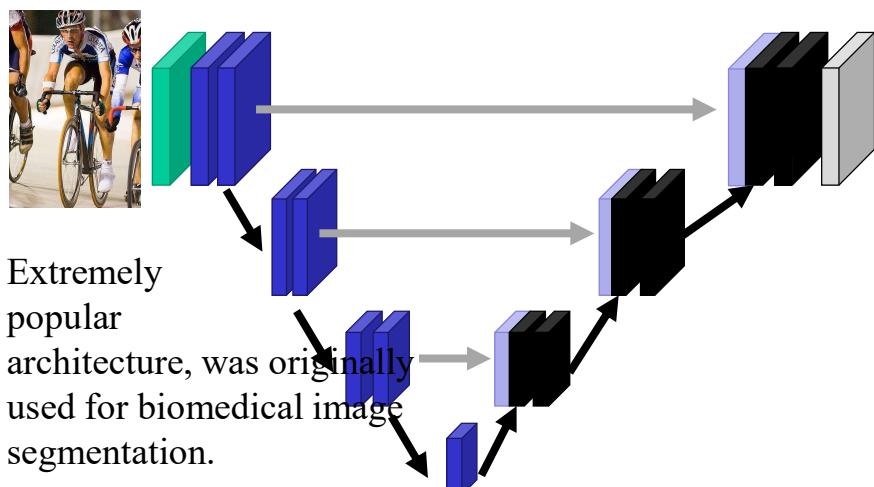
You copy the activations forward.

Subsequent layers at the same resolution figure out how to fuse things.



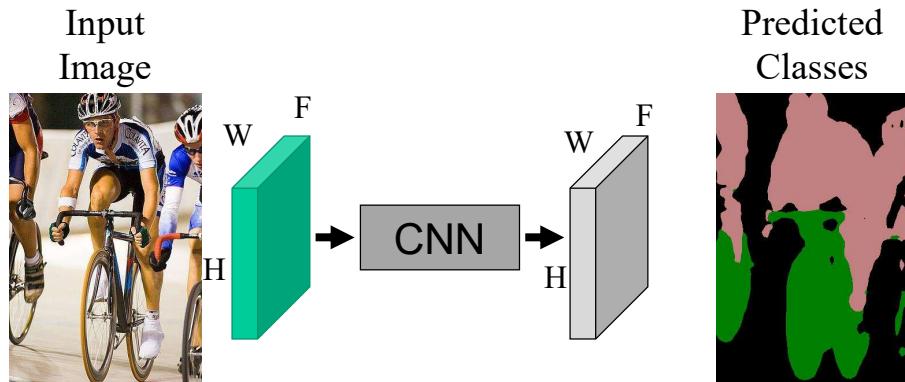
Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

U-Net



Ronneberger et al. "U-Net: Convolutional Networks for Biomedical Image Segmentation". MICCAI 2015

Evaluating Pixel Labels

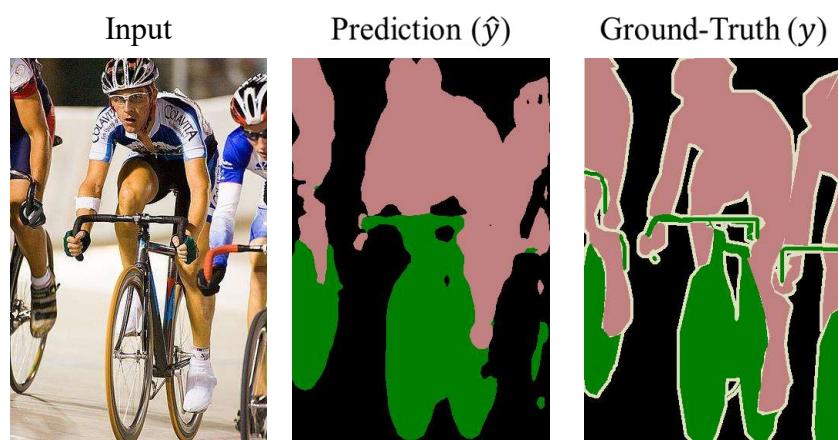


How do we convert final $H \times W \times F$ into labels?

argmax over labels

Evaluating Semantic Segmentation

Given predictions, how well did we do?

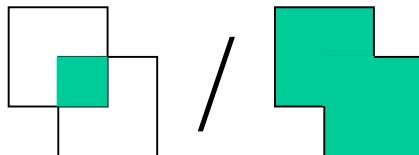


Evaluating Semantic Segmentation

Prediction and ground-truth are images where each pixel is one of F classes.

Accuracy: $\text{mean}(\hat{y} = y)$

Intersection over union,
averaged over classes



Prediction
 (\hat{y})



Ground-Truth
 (y)



Aside.

Why “Transpose Convolution”?

Can write convolution as matrix-multiply
Input: 4, Filter: 3, Stride: 1, Pad: 1

$$\begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} * \begin{matrix} x & y & z \end{matrix} \\ \\ \begin{matrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{matrix} X \begin{matrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{matrix} = \begin{matrix} ay+bz \\ ax+by+cz \\ bx+cy+dz \\ cx+dy \end{matrix} \end{array}$$

Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

Why “Transpose Convolution”?

Transpose convolution is convolution transposed

$$\begin{array}{c} \begin{matrix} a & b & c & d \end{matrix} *^T \begin{matrix} x & y & z \end{matrix} \\ \\ \begin{matrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{matrix} X \begin{matrix} a \\ b \\ c \\ d \end{matrix} = \begin{matrix} ax \\ ay+bx \\ az+by+cx \\ bz+cy+dx \\ cz+dy \\ dz \end{matrix} \quad \begin{matrix} ax \\ ay+bx \\ az+by+cx \\ bz+cy+dx \\ cz+dy \\ dz \end{matrix} \dots \end{array}$$

Example Credit: L. Fei-Fei, J. Johnson, S. Yeung

4 Performance evaluation parameters

The performance evaluation of a method is necessary to assure the validity of a method. This section lists out various performance measures that researchers have found useful for the quantitative evaluation of an image segmentation method [51]. Table 3 tabulates the formulation of various performance parameters.

1. Confusion matrix (CM): It is a widely used representation for the assessing the efficacy of a classification method [66]. However, it can be used to analyze the results of a clustering method too. The confusion matrix (CM) of size NxN represents that there are N classes (or clusters). In context of clustering, the number of correctly clustered patterns (true positive (TP), true negative (TN)) and wrongly clustered patterns (false positive (FP), false negative (FN)) can be easily identified from the confusion matrix. Table 4 illustrates an example of a typical confusion matrix for 2 clusters, i.e. positive and negative. In the table, TP and TN depict the number of data items which are correctly predicted in positive and negative clusters respectively. FP and FN correspond to the number of data items which are wrongly predicted as positive and negative clusters respectively. Based on Table 4, precision, recall and accuracy can also be computed using (18) – (20).

$$Precision = \frac{TP}{TP + FP} \quad (18)$$

$$Recall = \frac{TP}{TP + FN} \quad (19)$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (20)$$

2. Intersection of Union (IoU): Intersection of Union is the ratio of the number of common pixels between X and Y to the total number of pixels in X and Y. Here, X and Y

Table 3 Various performance parameters for the quantitative evaluation of an image segmentation method [51]

Parameters	Formulation
Boundary Displacement Error (BDE)	$BDE = \begin{cases} \frac{u-v}{L-1}, & 0 < (u - v) \\ 0, & (u - v) < 0 \end{cases}$
Probability Rand Index (PRI)	$PRI = \frac{a+b}{a+b+c+d} = \frac{a+b}{L}$
Variation of Information (VoI)	$VoI = H(S_1) + H(S_2) - 2I(S_1, S_2)$
Global Consistency Error (GCE)	$GCE = \frac{1}{n} \left\{ \sum_i E(S_1, S_2, p_i), \sum_i E(S_2, S_1, p_i) \right\}$
Structural Similarity Index (SSIM)	$SSIM = \frac{(2 \times \bar{X} \times \bar{Y} + c_1)(2 \times \sigma_{XY} + c_2)}{(\sigma_X^2 + \sigma_Y^2 + c_2) \times ((\bar{X})^2 + (\bar{Y})^2 + c_1)}$
Feature Similarity Index (FSIM)	$FSIM = \frac{\sum_{x \in Q} S_L(x).PC_m(x)}{\sum_{x \in Q} PC_m(x)}$
Root Mean Squared Error (RMSE)	$RMSE = \sqrt{MSE(\hat{\theta})} = \sqrt{E((\hat{\theta} - \theta)^2)}$
Peak Signal to Noise Ratio (PSNR)	$PSNR = 10 \log_{10} \frac{(2^b - 1)^2}{\sqrt{MSE}}$
Normalized Cross-Correlation (NCC)	$NCC = \frac{1}{n} \sum_{x,y} \frac{1}{\sigma_X \sigma_Y} (X(x, y) - \bar{X})(Y(x, y) - \bar{Y})$
Average Difference (AD)	$AD = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N X(x, y) - Y(x, y) $
Maximum Difference (MD)	$MD = \max X(x, y) - Y(x, y) $
Normalized Absolute Error (NAE)	$NAE = \frac{\sum_{x=1}^M \sum_{y=1}^N X(x, y) - Y(x, y) }{\sum_{x=1}^M \sum_{y=1}^N X(x, y) }$

Table 4 Confusion matrix

	True Positive	True Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

correspond to the segmented image and ground truth respectively. The formulation of IoU is depicted in (21).

$$IoU = \frac{|X \cap Y|}{|X| + |Y|} \quad (21)$$

3. Dice-Coefficient (DC): Dice-Coefficient is defined as twice the number of common pixels divided by total number of pixels in X and Y, where X corresponds to the segmented image and Y is the ground truth. DC is mathematically defined as (22).

$$DC = \frac{2|X \cap Y|}{|X| + |Y|} \quad (22)$$

4. Boundary Displacement Error (BDE): This parameter computes the average boundary pixels displacement error between two segmented images as depicted in (23). The error of one boundary pixel is defined as it's distance from the closest pixel in the other boundary image.

$$\mu_{LA}(u, v) = \begin{cases} \frac{u-v}{L-1} & 0 < u - v \\ 0 & u - v < 0 \end{cases} \quad (23)$$

5. Probability Rand Index (PRI): Probability Rand Index finds labelling consistency between the segmented image and its ground truth. It counts such fraction of pairs of pixels and average the result across all ground truths of a given image as shown in (24).

$$R = \frac{a + b}{a + b + c + d} = \frac{a + b}{n} \quad (24)$$

6. Variation of Information (VOI): Variation of Information as shown in (25) computes the randomness in one segmentation in terms of the distance from given segmentation.

$$VOI(X; Y) = H(X) + H(Y) - 2I(X, Y) \quad (25)$$

7. Global Consistency Error (GCE): A refinement in one segmentation over the other is depicted by the value of GCE as given in (26). If two segmentations are related this way then they are considered as consistent, i.e. both can represent the same natural image segmentation at different scales.

$$GCE = \frac{1}{n} \left\{ \sum_i E(s_1, s_2, p_i), \sum_i E(s_2, s_1, p_i) \right\} \quad (26)$$

8. Structural Similarity Index (SSIM): Structural Similarity Index measures the similarity between two images by taking initial uncompressed or distortion-free image as the reference and computed as (27). It incorporates important perceptual phenomena such as luminance masking and contrast masking.

$$SSIM = \frac{(2 \times \bar{x} \times \bar{y} + c_1)(2 \times \sigma_{xy} + c_2)}{(\sigma_x^2 + \sigma_y^2) \times ((\bar{x})^2 + (\bar{y})^2 + c_1)} \quad (27)$$

9. Feature Similarity Index (FSIM): Feature Similarity Index is a quality score that uses the phase congruency (PC), which is a dimensionless measure and shows the significance of a local structure. It is calculated by (28).

$$FSIM = \frac{\sum_{x \in \Omega} S_L(x) \cdot PC_m(x)}{\sum_{x \in \Omega} PC_m(x)} \quad (28)$$

10. Root Mean squared error (RMSE): The root-mean-squared error computes the difference between sample value predicted by a model or an estimator and actual value. The formulation is shown in (29).

$$RMSE(\hat{\theta}) = \sqrt{MSE(\hat{\theta})} = \sqrt{E((\hat{\theta} - \theta)^2)} \quad (29)$$

11. Peak Signal to Noise Ratio (PSNR in dB): Peak signal-to-noise ratio is defined as the ratio between the maximum possible power of a signal and the power of corrupting noise and is calculated using (30). In general, a higher value of PSNR represents high quality reconstruction and is defined via MSE.

$$PSNR = 10 \log_{10} \frac{(2^n - 1)^2}{\sqrt{MSE}} \quad (30)$$

12. Normalized Cross-Correlation (NCC): Normalized cross-correlation is used for template matching where images are first normalized due to lighting and exposure conditions. It is calculated by subtracting the mean of original and segmented images from the corresponding images, and divided by their standard deviations. Let $t(x, y)$ is the segmented image of $f(x, y)$, then NCC is calculated by (31).

$$\frac{1}{n} \sum_{x,y} \frac{1}{\sigma_f \sigma_t} (f(x, y) - \bar{f})(t(x, y) - \bar{t}) \quad (31)$$

13. Average Difference (AD): This parameter represents the average difference between the pixel values and is computed by (32).

$$AD = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (x(i, j) - y(i, j)) \quad (32)$$

14. Maximum Difference (MD): This parameter finds the maximum of error signal by taking the difference between original image and segmented image and defined in (33).

$$MD = \max |x(i, j) - y(i, j)| \quad (33)$$

15. Normalized Absolute Error (NAE): The normalized absolute difference between the original and corresponding segmented image gives NAE and is calculated by (34).

$$NAE = \frac{\sum_{i=1}^M \sum_{j=1}^N |x(i, j) - y(i, j)|}{\sum_{i=1}^M \sum_{j=1}^N |x(i, j)|} \quad (34)$$

In the above mentioned parameters, IoU, DC, SSIM, FSIM, PRI, PSNR, and NCC show better segmentation on high values. A high value indicates that the segmented image is more close to the ground truth. To measure the same, these parameters compute values by considering the region of intersection between the segmented image and the ground truth which corresponds to matching the number of similar pixels. On the contrary, other indices prefer lower values for better segmentation as these measures compute error between the segmented image and the ground truth and aim at reducing the difference between them.

5 Image segmentation benchmark datasets

The considered dataset is the key for fair analysis of an image segmentation method. The validation of segmentation methods against benchmark datasets tests its performance against the challenges posed by the considered dataset. Generally, a benchmark dataset consists of a variety of images that varies in number of different aspects. Some of the common challenges which segmentation methods need to handle are illumination variation, intra-class variation, and background complexity. This paper lists some of the popular benchmark datasets in Table 5 which are publicly available for image segmentation task and are briefed below.

- **Aberystwyth Leaf Evaluation Dataset:** It is dataset of timelapse images of *Arabidopsis thaliana* (*Arabidopsis*) plant to perform leaf-level segmentation. It consists of original *arabidopsis* plant images with 56 annotated ground truth images [1].
- **ADE20K:** It is a scene parsing segmentation dataset with around 22K hierarchically segmented images. For each image, there is a mask to segment objects and different parts of the objects [3].

Table 5 Benchmark datasets for image segmentation [23]

S.No.	Dataset	Purpose	Links
1	Aberystwyth Leaf Evaluation Dataset	Segmentation of Leaf	[1]
2	ADE20K Dataset	Segmentation of object and different parts of the objects	[3]
3	Berkeley Segmentation Dataset and Benchmark (BSDS)	Segmentation of objects	[74, 78]
4	Brain MRI Dataset	Segmentation of FLAIR abnormality	[14]
5	CAD120 Affordance Dataset	Segmentation of Affordance of objects	[15]
6	CoVID-19 CT-images Segmentation Dataset	Segmentation of infected region	[21]
7	Crack Detection Dataset	Segmentation of cracks	[5]
8	Daimler Pedestrian Segmentation Benchmark	Segmentation of pedestrian	[24]
9	Epithelium Segmentation Dataset	Segmentation of epithelium region	[80]
10	EVIMO Dataset	Motion segmentation	[30]
11	Liver Tumor Segmentation Dataset	Segmentation of Liver and Liver Tumor	[39]
12	Materials in Context Dataset	Material segmentation	[57]
13	Nuclei Segmentation Dataset	Segmentation of breast cancer nuclei	[79]
14	Objects with Thin and Elongated parts	Segmentation of objects with thin and elongated parts	[20]
15	OpenSurfaces Dataset	Surface segmentation	[56]
16	Oxford-IIIT Pet	Segmentation of pet animals	[82]
17	PetroSurf3D Dataset	Segmentation of petroglyphs	[35]
18	Segmentation Evaluation Database	Segmentation of single or two objects	[67]
19	Sky Dataset	Segmentation of sky	[71]
20	TB-roses-v1 Dataset	Segmentation of rose stems	[26]
21	Tsinghua Road Markings Dataset	Segmentation of road markings	[63]

- **Berkeley Segmentation Dataset and Benchmark (BSDS):** This is a benchmark dataset for evaluation of image segmentation method. It is a collection of 12K manually-labelled segmentations performed on 1K images from the Corel dataset by 30 human subjects. There are two versions of this dataset. BSDS300 [74] and BSDS500 [78] consist of 300 and 500 number of images respectively.
- **Brain MRI dataset:** It is segmentation dataset of MRI images along with manual fluid-attenuated inversion recovery (FLAIR) abnormality segmentation masks [14].
- **CAD120 Affordance Dataset:** This segmentation dataset provides binary masks for each the affordance in the RGB image captured from the Cornell Activity Dataset (CAD) 120. The affordance annotation is performed in the context of human [15].
- **CoVID-19 CT-images Segmentation Dataset:** This dataset consists of 100 CT-scan images of more than 40 CoVID-19 patients along with segmented ground truths. The segmented images were annotated by an expert radiologist [21].
- **Crack Detection Dataset:** It consists of five datasets where each dataset consists of pavement images and associated segmented images for crack detection [5].
- **Daimler Pedestrian Segmentation Benchmark:** This dataset contains pedestrian images along with corresponding segmented ground truth to perform pedestrian segmentation [24].
- **Epithelium Segmentation dataset:** There are 42 ER+ breast cancer images scanned at the magnification of 20x. The ground truth contains manually annotated epithelium region by an expert pathologist [80].
- **EVIMO Dataset:** This is a motion segmentation dataset along with egomotion estimation and tracking captured through event camera. It provides pixel-level masks for the motion segmentation [30].
- **Liver Tumor Segmentation (LITS):** This is a public available benchmark dataset provided by CodeLab for segmentation of liver tumor. It contains 3D-CT scans of liver along with segmented liver and liver tumor [39].
- **Materials in Context (MINC):** This dataset is intended for the segmentation and recognition of materials from the image. It consists of point annotations for 23 different types of materials [57].
- **Nuclei Segmentation dataset:** This dataset contains 143 ER+ breast cancer images scanned at 40x. In total, there are around 12,000 nuclei segmented manually. [79]
- **Objects with Thin and Elongated Parts:** It is a database of three datasets where each image consists of objects having thin and elongated parts. In total, there are 280 images of birds and insects with corresponding ground truth images [20].
- **OpenSurfaces:** This dataset includes tens of thousands of segmented surfaces. The dataset is constructed from interior photographs and annotated with information like material type, texture, and contextual [56].
- **Oxford-IIIT Pet Dataset:** This dataset consists of pet images with their pixel-level trimap segmentation. There are 37 categories of pets and each category contains 200 images [82].
- **PetroSurf3D Dataset:** This is a dataset which contributes in the area of rock art. There are total 26 high-resolution 3D-scans images with pixel-wise labelling for the segmentation of petroglyphs [35].
- **Segmentation Evaluation Database:** This dataset contains 200 gray-level images along with manually generated ground-truth images by three human subjects. Each image consists of either one or two objects in the foreground and the ground truth images are annotated into two or three classes [67].



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

AIMLC ZG525

Computer Vision

Tracking Algorithms



Agenda for the class

→ Kalman filtering

Scenarios

Imagine:

- Viewing a small bird flying through a forest
- Tracking a missile given a blip every few seconds
- Tracking planets, given intermittent observations

In each case:

- The observations are noisy
- But we can formulate an expectation about the trajectory

Scenarios

Imagine:

- Viewing a small bird flying through a forest
- Tracking a missile given a blip every few seconds
- Tracking planets, given intermittent observations

In each case:

- The observations are noisy
- But we can formulate an expectation about the trajectory

Goal:

- We are trying to infer the state, X , of a dynamic system, given only noisy measurements, Z , over time

Scenarios

Let

X - coordinate of a bird being tracked (ignoring its Y and Z coordinates for brevity)

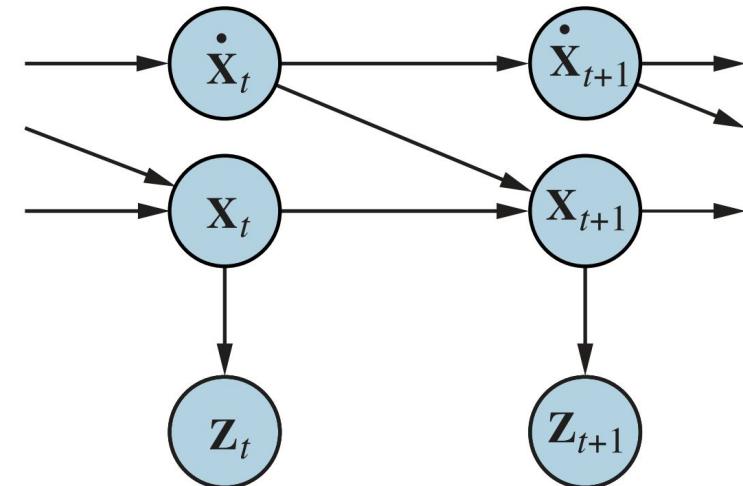
\dot{X} - velocity (assumed to be constant) [X dot]

Δ - time interval between measurements

$$X_{t+\Delta} = X_t + \dot{X} \Delta$$

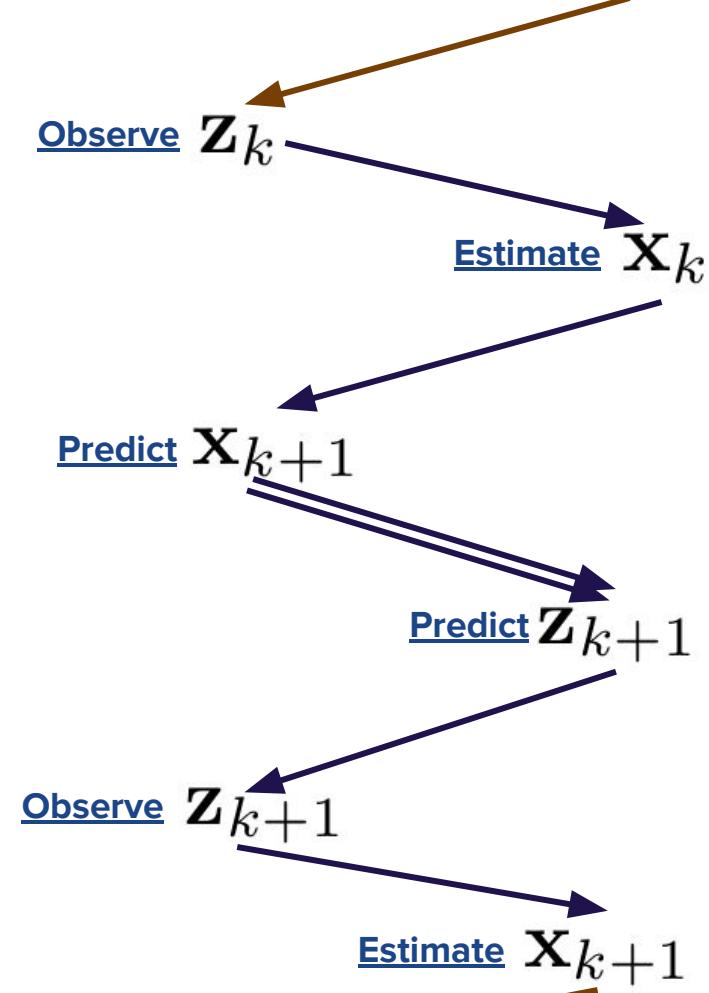
Adding **Gaussian noise** (to account for wind variation, etc.), we get a linear-Gaussian transition model:

$$P(X_{t+\Delta} = x_{t+\Delta} | X_t = x_t, \dot{X}_t = \dot{x}_t) = \mathcal{N}(x_{t+\Delta}; x_t + \dot{x}_t \Delta, \sigma^2)$$

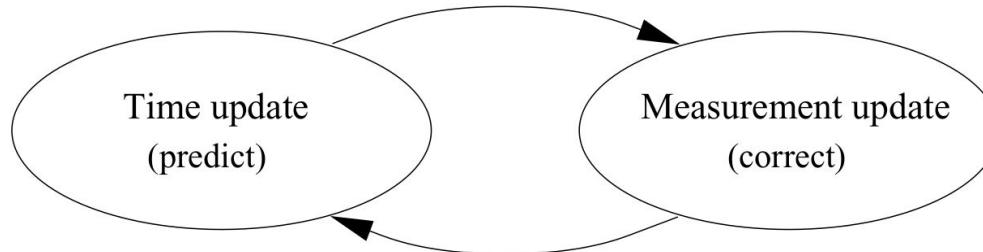




Modelling Motion



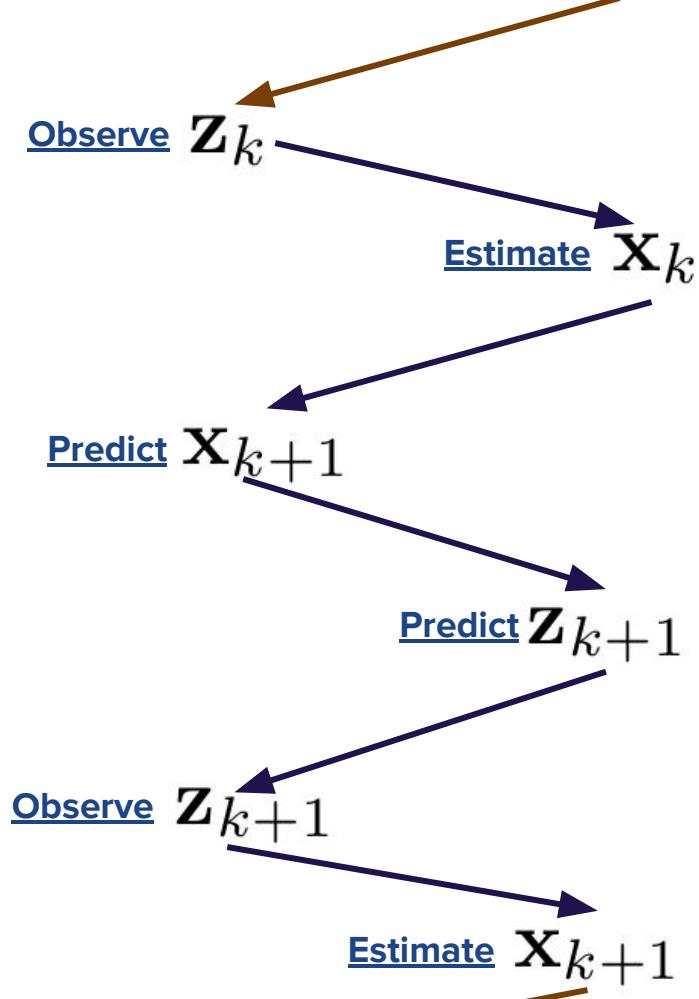
Modelling Motion



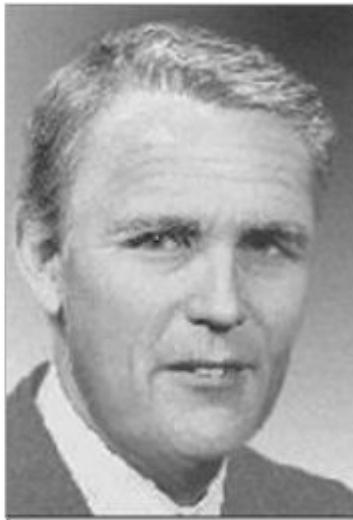
The **predictor-corrector iterative cycle**:

Time update predicts events at the next step;

Measurement update adjusts estimates in the light of observation.

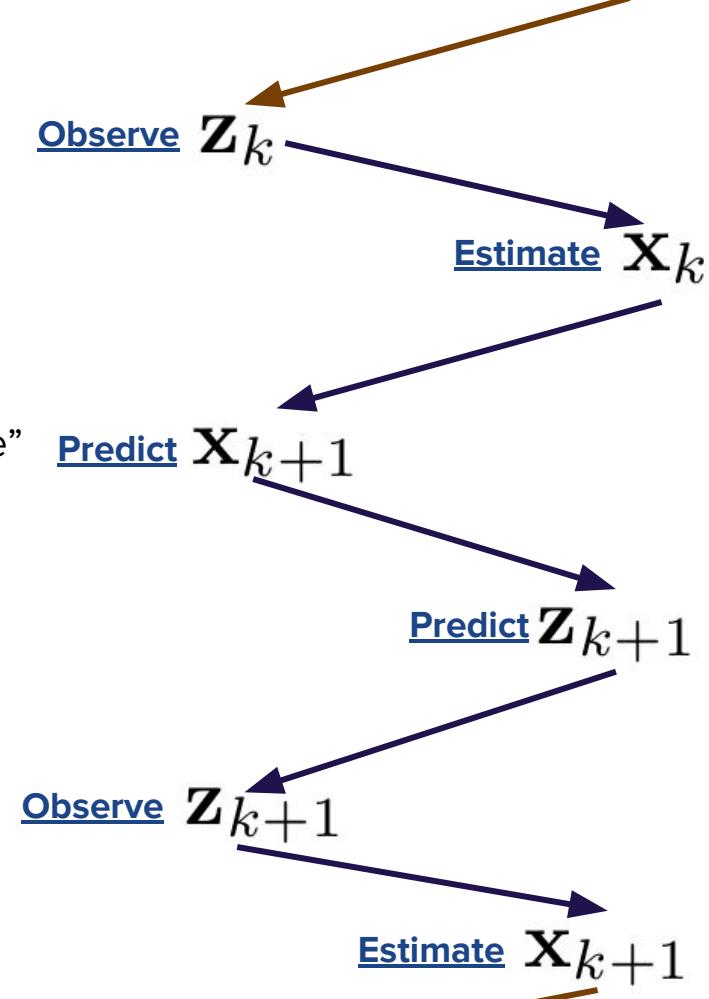


Kalman Filters



Rudolf Emil Kalman

- Assume that results of experiment are noisy measurements of “system state”
- Model of how system evolves
- Optimal combination of system model and observations
- Prediction / correction framework
- System is linear

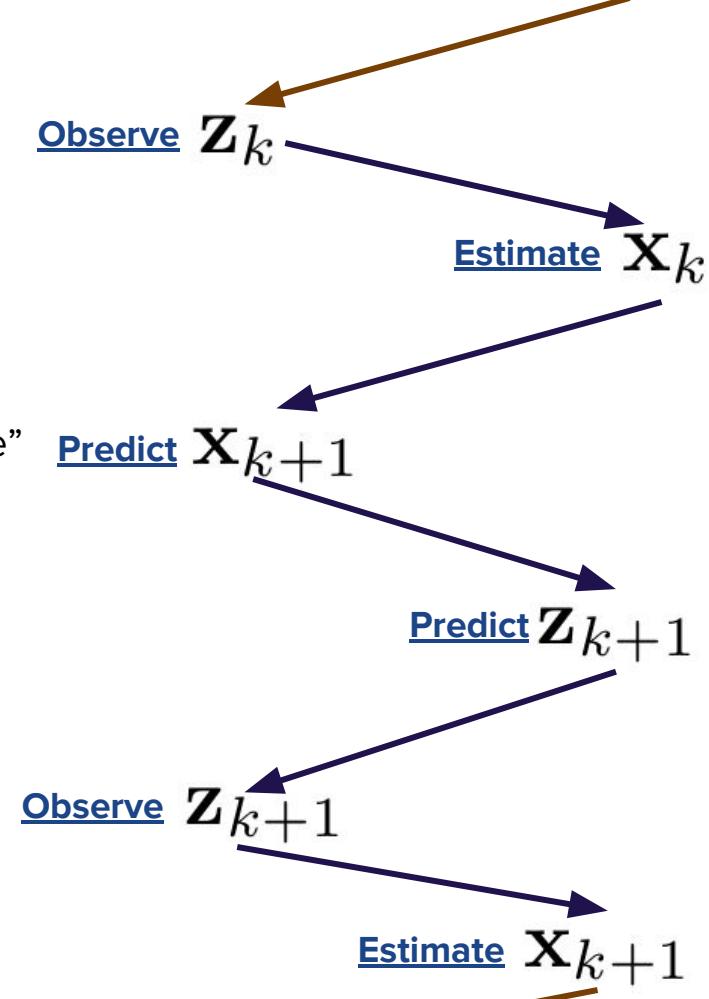


Kalman Filters



Rudolf Emil Kalman

- Assume that results of experiment are noisy measurements of “system state”
- Model of how system evolves
- Optimal combination of system model and observations
- Prediction / correction framework
- System is linear





Kalman Filters

$$z_k = \begin{bmatrix} x_{m_k} \\ y_{m_k} \end{bmatrix} \quad R_k = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} \\ \sigma_{xy_m} & \sigma_{y_m}^2 \end{bmatrix} \quad t_k = t_{m_k}$$

The subscript m denotes the measurement parameters.
The k subscript denotes the order of the measurement.

x	state variable	n x 1 column vector	Output
P	state covariance matrix	n x n matrix	Output
z	measurement	m x 1 column vector	Input
A	state transition matrix	n x n matrix	System Model
H	state-to-measurement matrix	m x n matrix	System Model
R	measurement covariance matrix	m x m matrix	Input
Q	process noise covariance matrix	n x n matrix	System Model
K	Kalman Gain	n x m	Internal

Kalman Filter Algorithm Reference Terms



Kalman Filters

$$z_k = \begin{bmatrix} x_{m_k} \\ y_{m_k} \end{bmatrix} \quad R_k = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} \\ \sigma_{xy_m} & \sigma_{y_m}^2 \end{bmatrix} \quad t_k = t_{m_k}$$

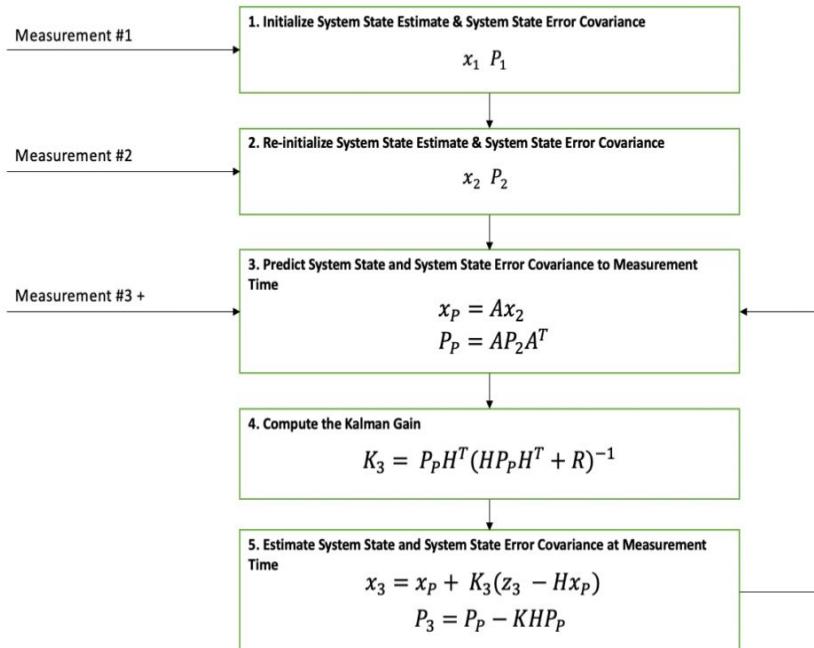
The subscript m denotes the measurement parameters.
The k subscript denotes the order of the measurement.

$$x_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \quad P_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{x\dot{x}} & \sigma_{y\dot{x}} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{x\dot{y}} & \sigma_{y\dot{y}} & \sigma_{\dot{x}\dot{y}} & \sigma_{\dot{y}}^2 \end{bmatrix} \quad T_k$$

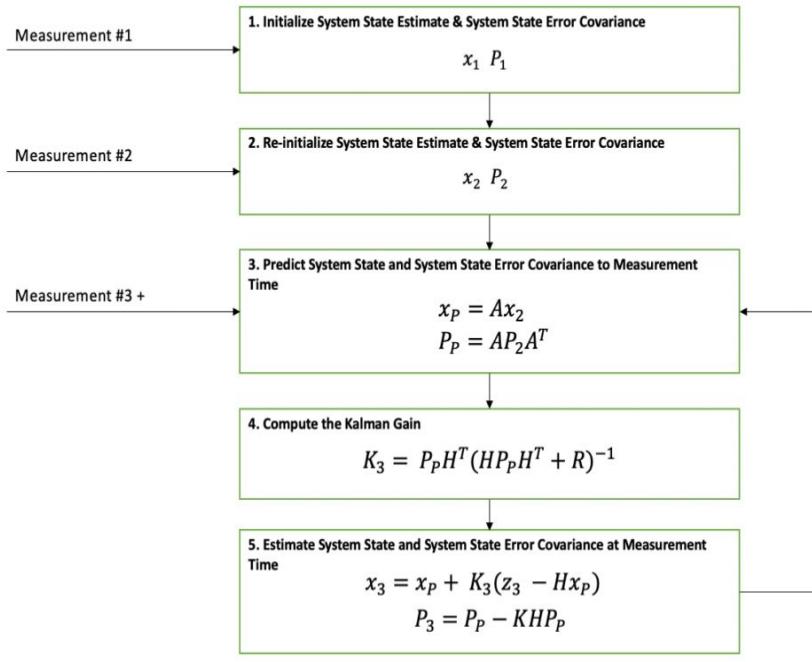
x	state variable	n x 1 column vector	Output
P	state covariance matrix	n x n matrix	Output
z	measurement	m x 1 column vector	Input
A	state transition matrix	n x n matrix	System Model
H	state-to-measurement matrix	m x n matrix	System Model
R	measurement covariance matrix	m x m matrix	Input
Q	process noise covariance matrix	n x n matrix	System Model
K	Kalman Gain	n x m	Internal

Kalman Filter Algorithm Reference Terms

Kalman Filters



Kalman Filters

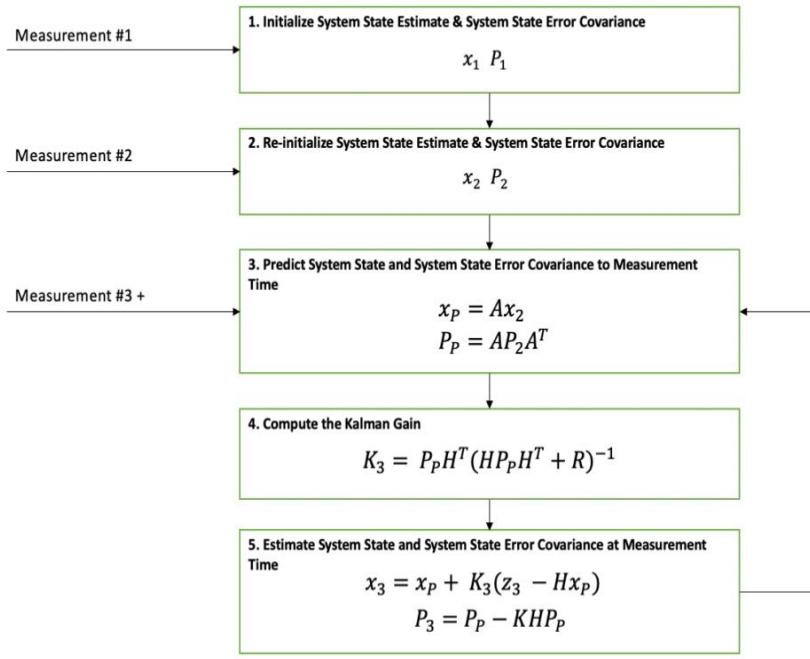


$$z_1 = \begin{bmatrix} x_{m_1} \\ y_{m_1} \end{bmatrix} \quad R_1 = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} \\ \sigma_{xy_m} & \sigma_{y_m}^2 \end{bmatrix} \quad t_1 = t_{m_1}$$

$$x_1 = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x_{m_1} \\ y_{m_1} \\ 0 \\ 0 \end{bmatrix} \quad T_1 = t_{m_1}$$

$$P_1 = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{x\dot{x}} & \sigma_{y\dot{x}} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{x\dot{y}} & \sigma_{y\dot{y}} & \sigma_{\dot{x}\dot{y}} & \sigma_{\dot{y}}^2 \end{bmatrix} = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} & 0 & 0 \\ \sigma_{xy_m} & \sigma_{y_m}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Kalman Filters



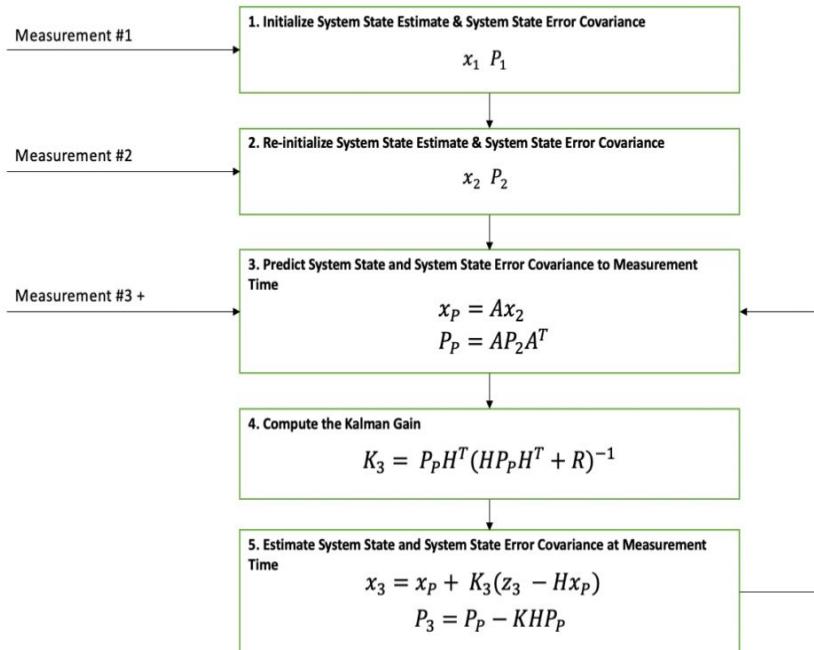
$$z_2 = \begin{bmatrix} x_{m_2} \\ y_{m_2} \end{bmatrix} \quad R_2 = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} \\ \sigma_{xy_m} & \sigma_{y_m}^2 \end{bmatrix} \quad t_2 = t_{m_2}$$

$$x_2 = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x_{m_2} \\ y_{m_2} \\ \frac{x_{m_2} - x_{m_1}}{\Delta T} \\ \frac{y_{m_2} - y_{m_1}}{\Delta T} \end{bmatrix} \quad T_2 = t_{m_2} \quad \Delta T = T_2 - T_1$$

$$P_2 = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\dot{x}} & \sigma_{x\dot{y}} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\dot{x}} & \sigma_{y\dot{y}} \\ \sigma_{x\dot{x}} & \sigma_{y\dot{x}} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x}\dot{y}} \\ \sigma_{x\dot{y}} & \sigma_{y\dot{y}} & \sigma_{\dot{x}\dot{y}} & \sigma_{\dot{y}}^2 \end{bmatrix} = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} & 0 & 0 \\ \sigma_{xy_m} & \sigma_{y_m}^2 & 0 & 0 \\ 0 & 0 & 10^4 & 0 \\ 0 & 0 & 0 & 10^4 \end{bmatrix}$$

Step-3

Kalman Filters



$$z_3 = \begin{bmatrix} x_{m_3} \\ y_{m_3} \end{bmatrix} \quad R_2 = \begin{bmatrix} \sigma_{x_m}^2 & \sigma_{xy_m} \\ \sigma_{xy_m} & \sigma_{y_m}^2 \end{bmatrix} \quad t_3 = t_{m_3}$$

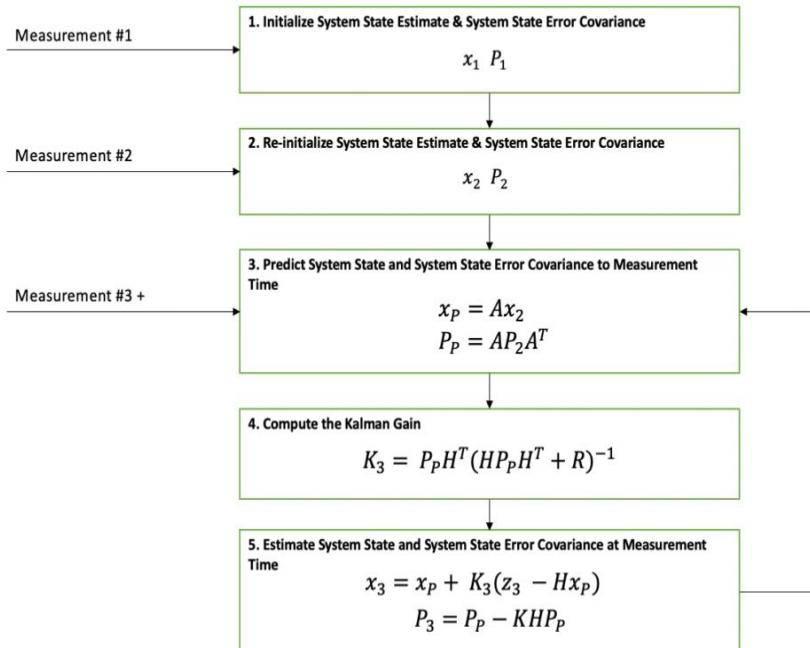
$$T_3 = t_{m_3} \quad \Delta T = T_3 - T_2$$

$$A = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 25 \end{bmatrix}$$

$$x_p = Ax_2 = \begin{bmatrix} x + \dot{x}\Delta T \\ y + \dot{y}\Delta T \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

$$P_P = AP_2A^T + Q = \begin{bmatrix} \sigma_x^2 + 10000 \cdot \Delta T^2 + 10 & \sigma_{xy} & 10000 \cdot \Delta T & 0 \\ \sigma_{xy} & \sigma_y^2 + 10000 \cdot \Delta T^2 + 10 & 0 & 10025 \\ 10000 \cdot \Delta T & 0 & 10000 \cdot \Delta T & 0 \\ 0 & 10000 \cdot \Delta T & 0 & 10025 \end{bmatrix}$$

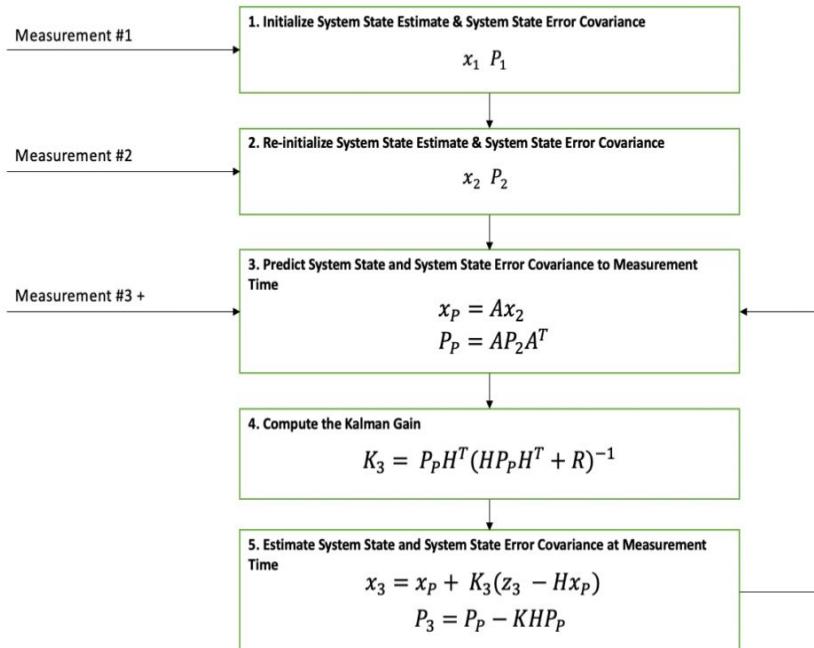
Kalman Filters



Compute Kalman Gain

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

Kalman Filters



$$x_k = x_p + K(z_k - Hx_p)$$

$$P_k = P_p - K H P_p$$



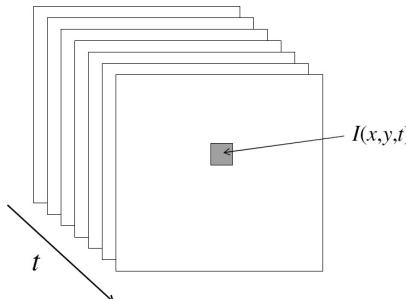
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you

Optical flow and tracking

From images to videos

- A video is a sequence of frames captured over time
- Now our image data is a function of space (x, y) and time (t)



Uses of motion

- Improving video quality
 - Motion stabilization
 - Super resolution
- Segmenting objects based on motion cues
- Tracking objects
- Recognizing events and activities

Super-resolution

- Irani, M.; Peleg, S. (June 1990). "Super Resolution From Image Sequences". International Conference on Pattern Recognition
- Fast and Robust Multiframe Super Resolution, Sina Farsiu, M. Dirk Robinson, Michael Elad, and Peyman Milanfar, EEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 13, NO. 10, OCTOBER 2004

Example: A set of low quality images

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Most of the test data o
couple of exceptions. 1
low-temperature solder
investigated (or some
manufacturing technol
nonwetting of 40fl40S
microstructural coarse
mal cycling of 58Bi42S

Super-resolution

Each of these images looks like this:

Most of the test data or couple of exceptions. low-temperature solder investigated (or some manufacturing technology) nonwetting of 40In40Sb microstructural coarse thermal cycling of 58Bi42Si

5

Super-resolution

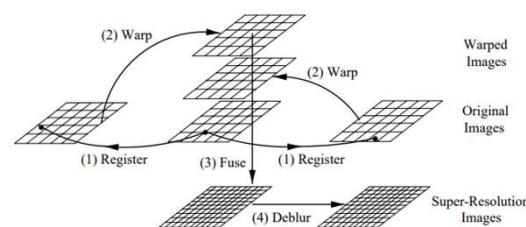
The recovery result:

Most of the test data or couple of exceptions. low-temperature solder investigated (or some manufacturing technology) nonwetting of 40In40Sb microstructural coarse thermal cycling of 58Bi42Si

6

Super-resolution

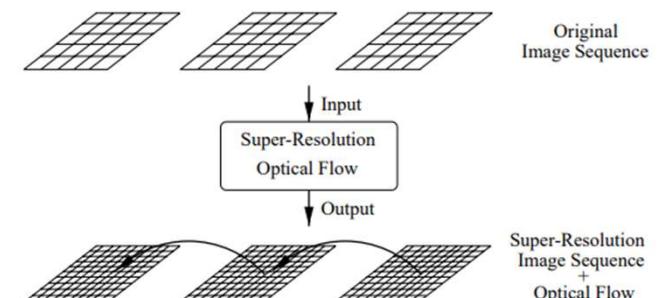
[A peep into the process](#) ← Click for details



The four steps of super-resolution: (1) the input images are registered to find corresponding pixels, (2) the original images are warped into the coordinate frame of one image, (3) the warped images are fused to form a super-resolution image, and (4) the super-resolution image is deblurred (optional)

6

Super-resolution



6

Visual SLAM

Simultaneous Localization And Mapping

Courtesy of Jean-Yves Bouguet – Vision Lab, California Institute of Technology

Visual SLAM

Simultaneous Localization And Mapping

White dots - Landmarks point cloud
Blue arrow - latest visual odometry pose
Green line - odometry (accumulated)

Colorful dots - Observations point cloud
Red tiles - Loop Closure point cloud

Courtesy of Jean-Yves Bouguet – Vision Lab, California Institute of Technology

Segmenting objects based on motion cues

- Background subtraction
 - A static camera is observing a scene
 - Goal: separate the static *background* from the moving *foreground*

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

Suhak Kwak, Taegyu Lim, Woonyun Nam, Bohyung Han, Joon Hee Han: Generalized background subtraction based on hybrid inference by belief propagation and Bayesian filtering. ICCV 2011

Segmenting objects based on motion cues

Hopkins-People2 sequence

Ours

Motion segmentation

Ours without motion inference

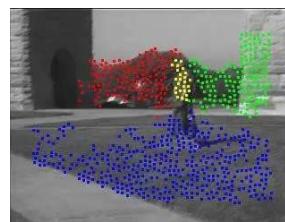
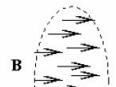
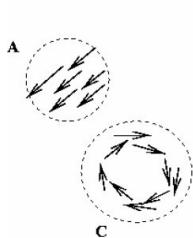
Sheikh et al. ICCV'09

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

Suhak Kwak, Taegyu Lim, Woonyun Nam, Bohyung Han, Joon Hee Han: Generalized background subtraction based on hybrid inference by belief propagation and Bayesian filtering. ICCV 2011

Segmenting objects based on motion cues

- Motion segmentation
 - Segment the video into multiple *coherently* moving objects



S. J. Pundlik and S. T. Birchfield, Motion Segmentation at Any Speed, Proceedings of the British Machine Vision Conference '06

Tracking objects

- Facing tracking on openCV



OpenCV's face tracker uses an algorithm called Camshift (based on the meanshift algorithm)

http://www.youtube.com/watch?v=HTk_UwAYzVk

Tracking objects

Object Tracking by Oversampling Local Features. Del Bimbo, and F. Pernici, IEEE Transaction On Pattern Analysis And Machine Intelligence, 2014



Double Loop

- Use Scale Invariant Feature Transform (SIFT) when applied to (flat) objects

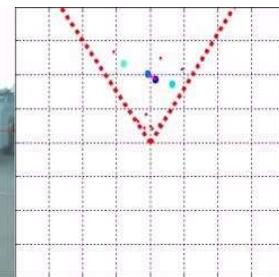
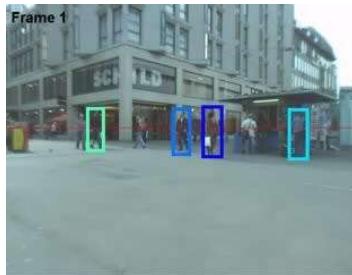
<http://www.micc.unifi.it/pernici/#alien>

DOWNLOAD <http://www.micc.unifi.it/pernici/>

Tracking objects



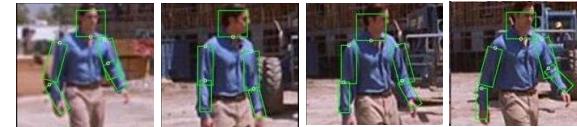
Joint tracking and 3D localization



W. Choi & K. Shahid & S. Savarese WMC 2009
W. Choi & S. Savarese , ECCV, 2010

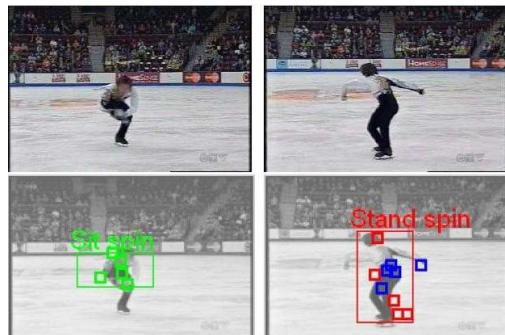
Tracking body parts

Cascaded Models for Articulated Pose Estimation, B Sapp, A Toshev, B Taskar, Computer Vision-ECCV 2010, 406-420



Courtesy of Benjamin Sapp

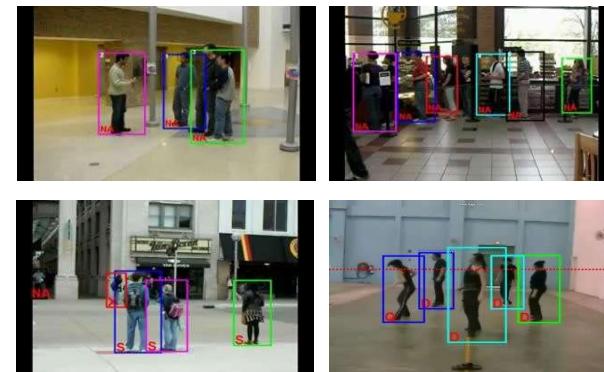
Recognizing events and activities



Juan Carlos Niebles, Hongcheng Wang and Li Fei-Fei, **Unsupervised Learning of Human Action Categories Using Spatial-Temporal Words**, ([BMVC](#)), Edinburgh, 2006.

Recognizing group activities

Crossing – Talking – Queuing – Dancing – jogging



Choi & Savarese, CVPR 11
Choi & Savarese, ECCV 2012

X: Crossing, S: Waiting, Q: Queuing,
W: Walking, T: Talking, D: Dancing

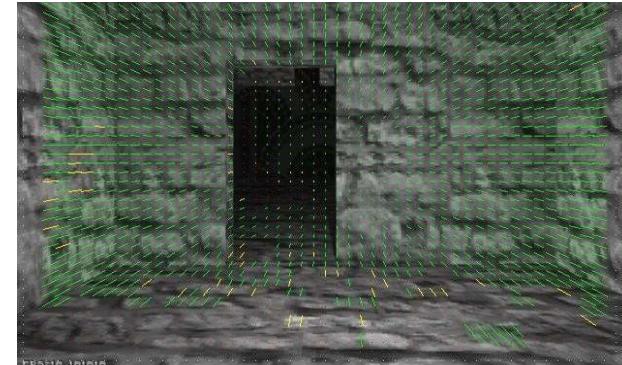
Motion estimation techniques

- Optical flow
 - Recover image motion at each pixel from spatio-temporal image brightness variations (optical flow)
- Feature-tracking
 - Extract visual features (corners, textured areas) and "track" them over multiple frames



Optical flow

Vector field function of the spatio-temporal image brightness variations



Picture courtesy of Selim Temizer - Learning and Intelligent Systems (LIS) Group, MIT

Optical flow

Vector field function of the spatio-temporal image brightness variations

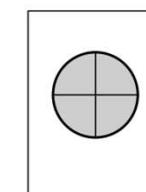


<http://www.youtube.com/watch?v=JILkkom6tWw>

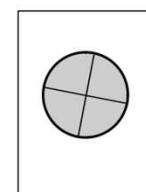
Optical flow

Definition: optical flow is the *apparent* motion of brightness patterns in the image

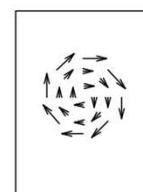
GOAL: Recover image motion at each pixel by optical flow



Time t1



Time t2



Optical flow

Optical flow

Definition: optical flow is the **apparent** motion of brightness patterns in the image

GOAL: Recover image motion at each pixel by optical flow

Note: apparent motion can be caused by lighting changes without any actual motion



The brightness constancy constraint

$$\begin{array}{c} (x, y) \quad v \\ u \\ \text{displacement} \end{array} = (u, v) \quad \begin{array}{c} (x + u, y + v) \\ I(x, y, t-1) \end{array} \quad \begin{array}{c} I(x, y, t) \end{array}$$

Brightness Constancy Equation:

$$I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t)$$

Linearizing the right side using Taylor expansion:

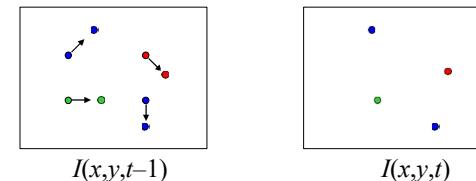
$$I(x + u, y + v, t) \approx I(x, y, t-1) + I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t$$

$$I(x + u, y + v, t) - I(x, y, t-1) = I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t$$

Hence, $I_x \cdot u + I_y \cdot v + I_t \approx 0 \rightarrow \nabla I \cdot [u \ v]^T + I_t = 0$

Optical flow constraint equation; I_x, I_y, I_t can be computed from 2 successive frames

Estimating optical flow



Given two subsequent frames, estimate the apparent motion field $u(x, y), v(x, y)$ between them

- Key assumptions

- **Brightness constancy:** projection of the same point looks the same in every frame
- **Small motion:** points do not move very far
- **Spatial coherence:** points move like their neighbors

The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$I_x \cdot u + I_y \cdot v + I_t \approx 0$$

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

How many equations and unknowns per pixel?

- One equation (this is a scalar equation!), two unknowns (u, v)

Adding constraints....

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674-679, 1981.

How to get more equations for a pixel?

Spatial coherence constraint:

Assume the pixel's neighbors have the same (u, v)

- If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [u \ v] \quad p_i = (x_i, y_i)$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

Lucas-Kanade flow

Overconstrained linear system:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix} \quad A \ d = b$$

Lucas-Kanade flow

Overconstrained linear system

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix} \quad A \ d = b$$

Least squares solution for d given by $(A^T A) d = A^T b$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$A^T A \quad A^T b$$

The summations are over all pixels in the K x K window

Conditions for solvability

- Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$A^T A \quad A^T b$$

When is this solvable?

- $A^T A$ should be invertible
- Eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

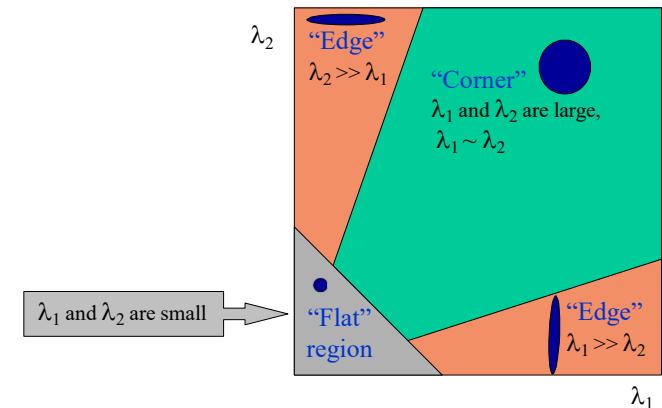
$M = A^T A$ is the *second moment matrix*!
(Harris corner detector...)

$$M = A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

- Eigenvectors and eigenvalues of $A^T A$ relate to edge direction and magnitude
 - The eigenvector associated with the larger eigenvalue points in the direction of fastest intensity change
 - The other eigenvector is orthogonal to it

Interpreting the eigenvalues

Classification of image points using eigenvalues of the second moment matrix:



Low-texture region



$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

- gradients have small magnitude
- small λ_1 , small λ_2

Edge



$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

- gradients very large or very small
- large λ_1 , small λ_2

High-texture region



$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

- gradients are different, large magnitudes
- large λ_1 , large λ_2

What are good features to track?

J. Shi and C. Tomasi (June 1994). [Good Features to Track](#). 9th IEEE Conference on Computer Vision and Pattern Recognition. Springer.

Can we measure “quality” of features from just a single image

Good features to track:

- Harris corners (guarantee small error sensitivity)

Bad features to track:

- Image points when either λ_1 or λ_2 (or both) is small (i.e., edges or uniform textured regions)

The barber pole illusion



http://en.wikipedia.org/wiki/Barberpole_illusion

Motion estimation techniques

Optical flow

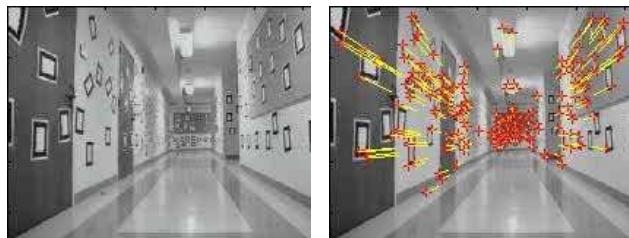
- Recover image motion at each pixel from spatio-temporal image brightness variations (optical flow)



Feature-tracking

- Extract visual features (corners, textured areas) and “track” them over multiple frames
 - Shi-Tomasi feature tracker
 - Tracking with dynamics
- Implemented in Open CV

Tracking features



Courtesy of Jean-Yves Bouguet – Vision Lab, California Institute of Technology

Recap

- Key assumptions (Errors in Lucas-Kanade)

- **Small motion:** points do not move very far
- **Brightness constancy:** projection of the same point looks the same in every frame
- **Spatial coherence:** points move like their neighbors

Revisiting the small motion assumption

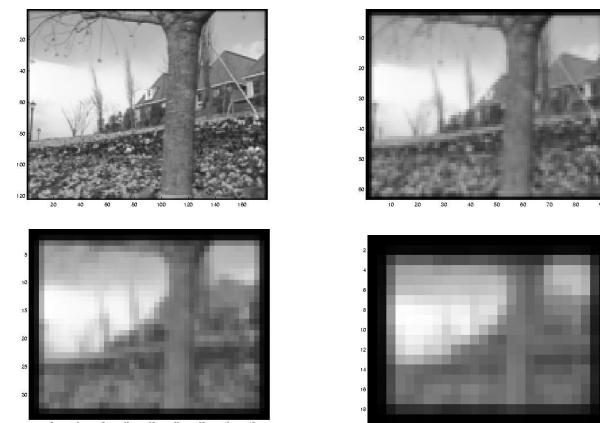


Is this motion small enough?

- Probably not—it's much larger than one pixel (2nd order terms dominate)
- How might we solve this problem?

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

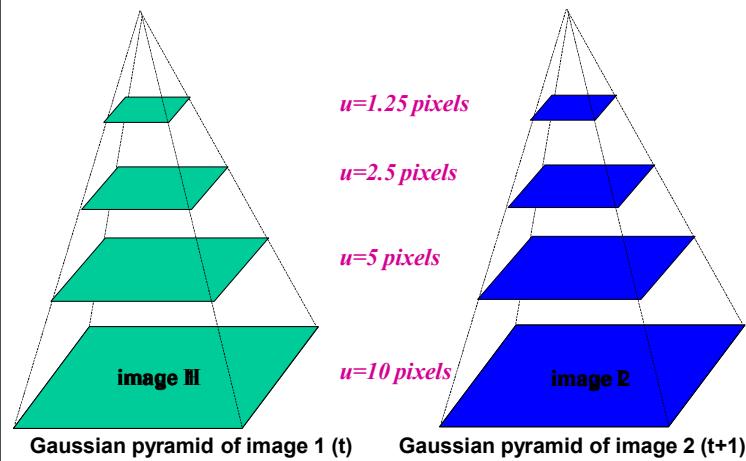
Reduce the resolution!



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

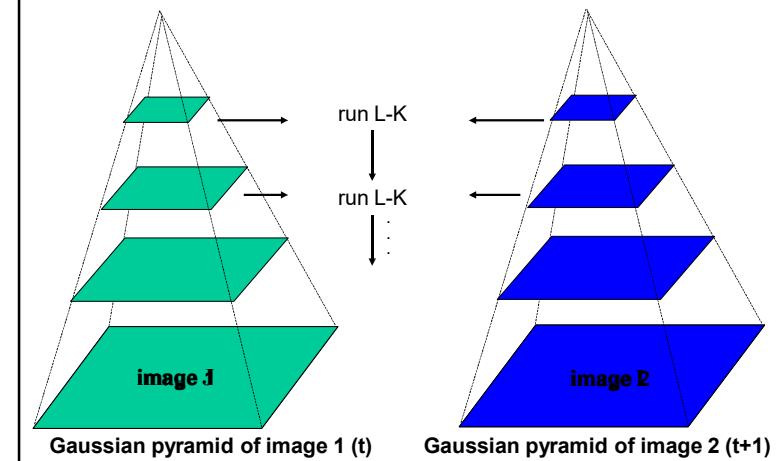
Coarse-to-fine optical flow estimation

JY Bouguet, Pyramidal Implementation of the Lucas Kanade Feature Tracker
Description of the algorithm,
Tech. Report: http://robots.stanford.edu/cs223b04/algo_tracking.pdf

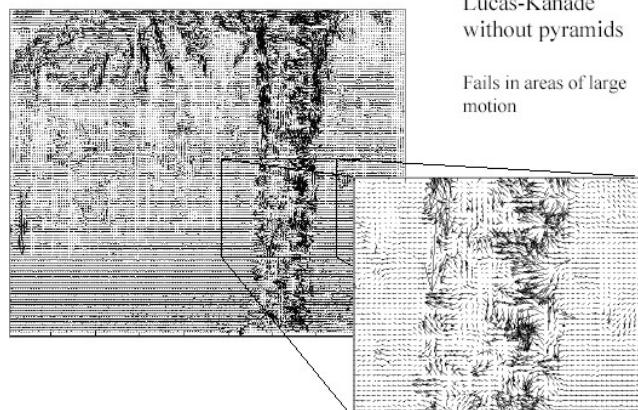


Coarse-to-fine optical flow estimation

JY Bouguet, Pyramidal Implementation of the Lucas Kanade Feature Tracker
Description of the algorithm,
Tech. Report: http://robots.stanford.edu/cs223b04/algo_tracking.pdf

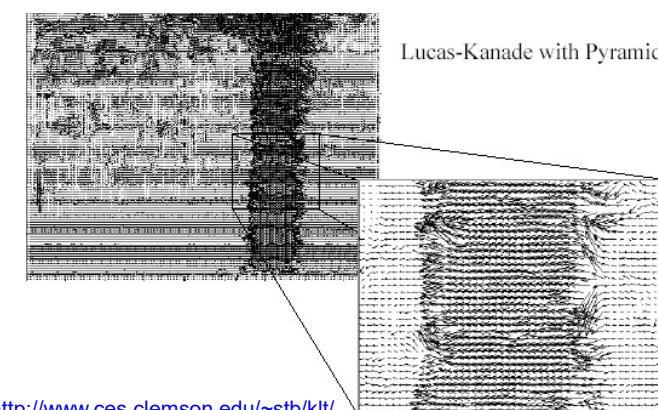


Optical Flow Results



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Optical Flow Results



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

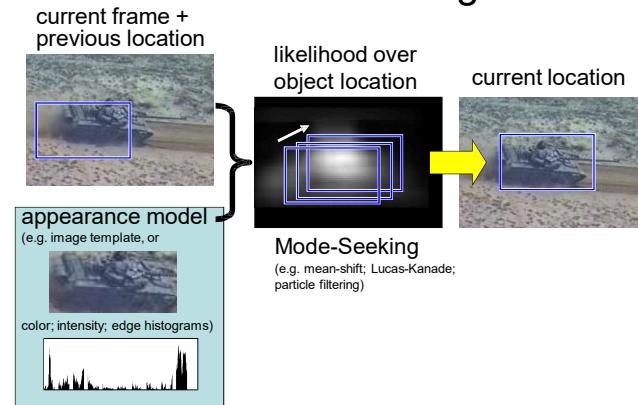
Recap

- Key assumptions (Errors in Lucas-Kanade)
 - **Small motion:** points do not move very far
 - **Brightness constancy:** projection of the same point looks the same in every frame
 - **Spatial coherence:** points move like their neighbors

Tracking (2) –Mean Shift for Tracking

• Adopted from Source: R.Collins, CSE, PSU CSE598G Spring 2006 & Yaron Ukrainitz & Bernard Sarel

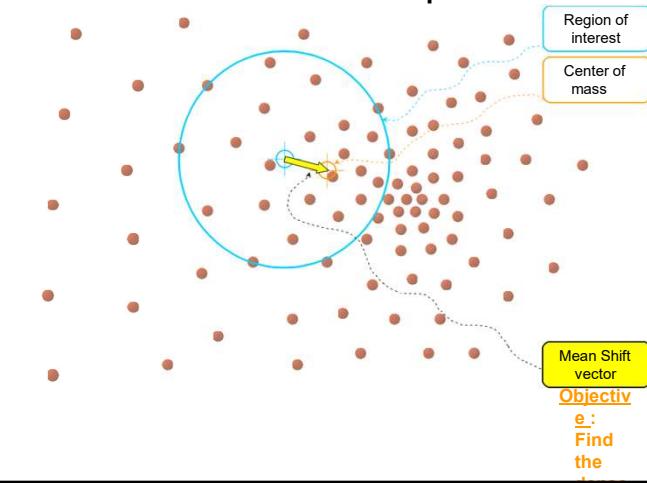
Appearance-Based Tracking



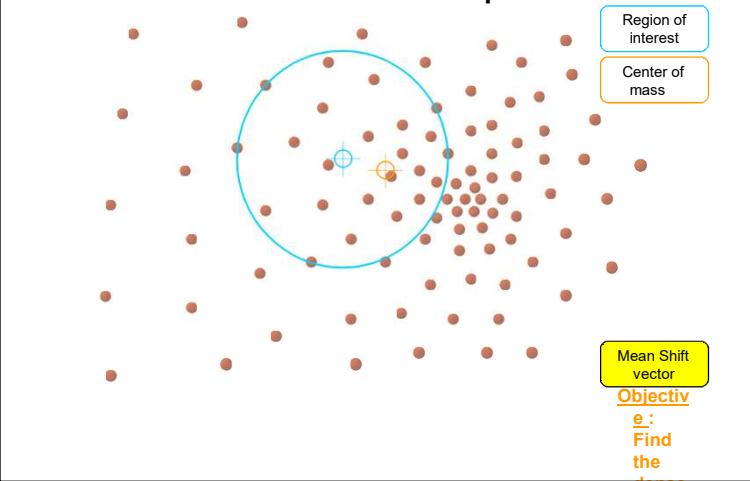
Motivation

- Motivation – to track non-rigid objects, (like a walking person), it is hard to specify
- an explicit 2D parametric motion model.
- Appearances of non-rigid objects can sometimes be modeled with color distributions

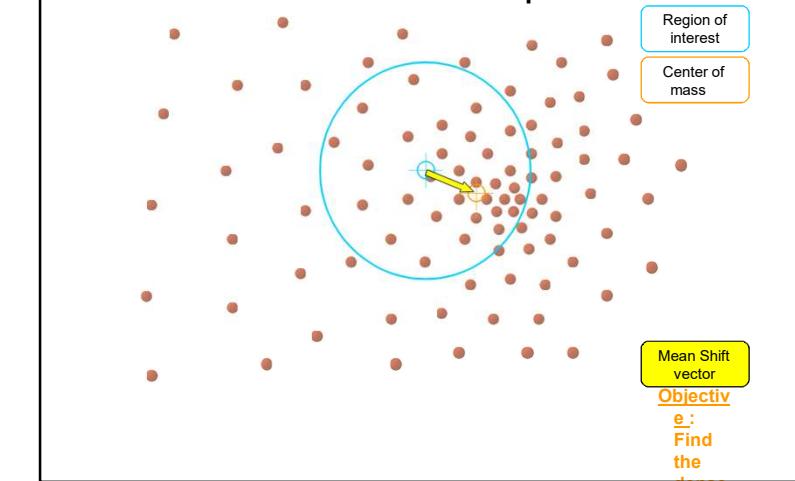
Intuitive Description



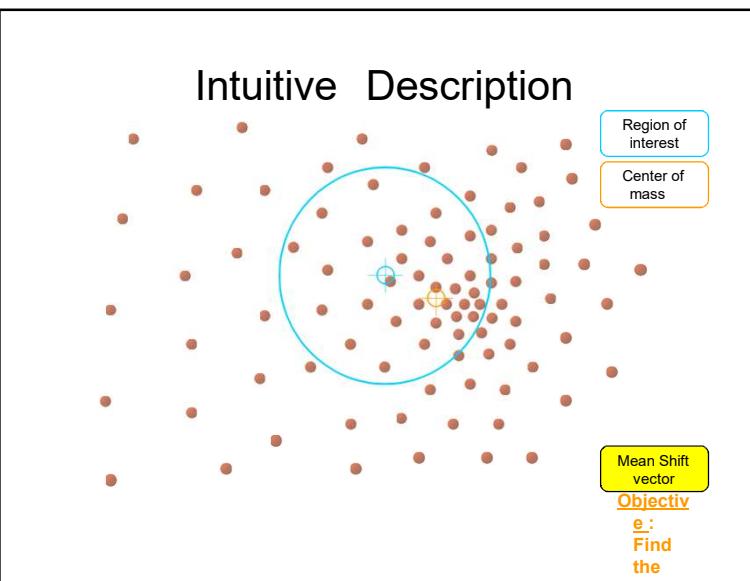
Intuitive Description



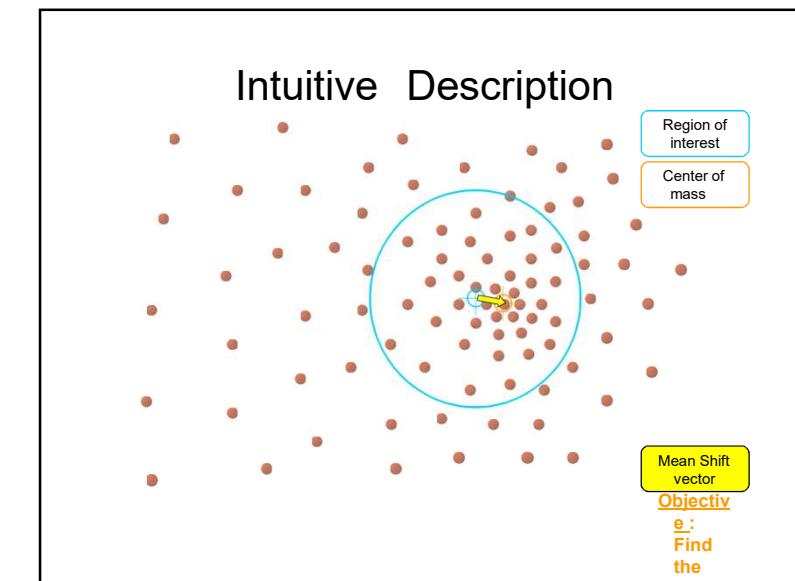
Intuitive Description



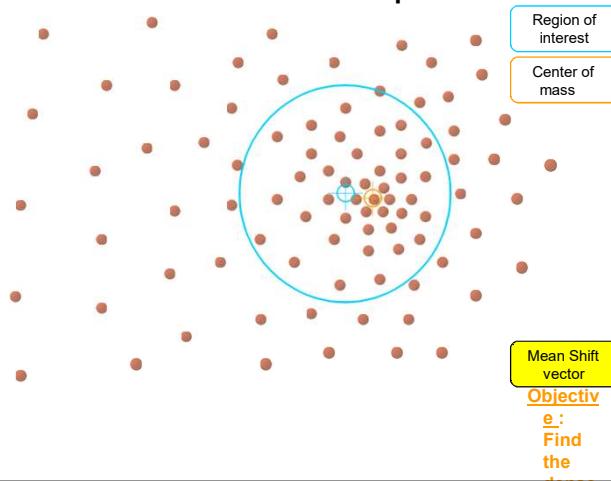
Intuitive Description



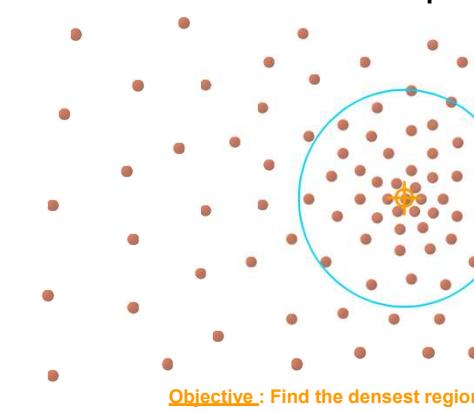
Intuitive Description



Intuitive Description

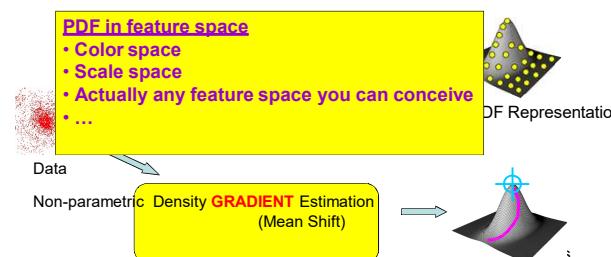


Intuitive Description



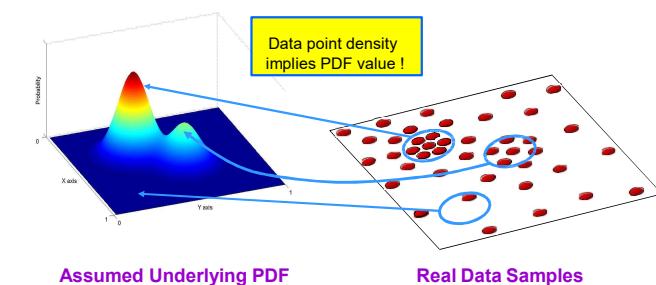
What is Mean Shift ?

A tool for:
Finding modes in a set of data samples, manifesting an underlying probability density function (PDF) in \mathbb{R}^n

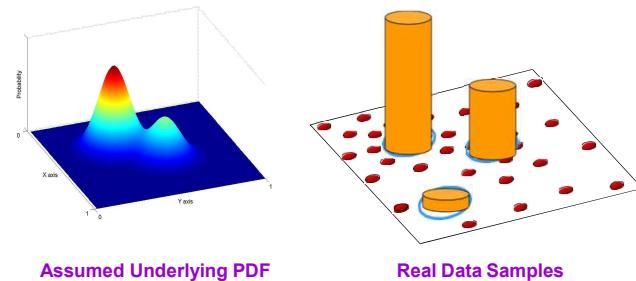


Non-Parametric Density Estimation

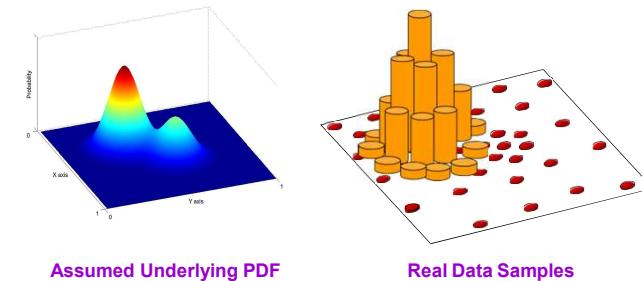
Assumption : The data points are sampled from an underlying PDF



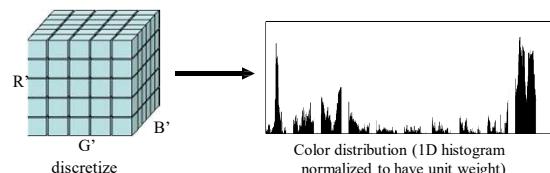
Non-Parametric Density Estimation



Non-Parametric Density Estimation



Appearance via Color Histograms

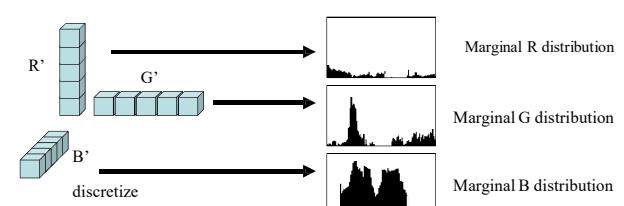


$R' = R \lll (8 - \text{nbits})$
 $G' = G \lll (8 - \text{nbits})$
 $B' = B \lll (8 - \text{nbits})$

Total histogram size is $(2^{(8-\text{nbits})})^3$
example, 4-bit encoding of R,G and B channels
yields a histogram of size $16*16*16 = 4096$.

Smaller Color Histograms

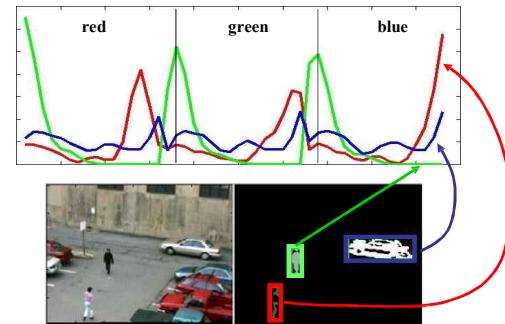
Histogram information can be much much smaller if we are willing to accept a loss in color resolvability.



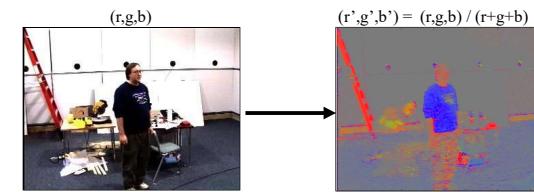
$R' = R \lll (8 - \text{nbits})$
 $G' = G \lll (8 - \text{nbits})$
 $B' = B \lll (8 - \text{nbits})$

Total histogram size is $3*(2^{(8-\text{nbits})})$
example, 4-bit encoding of R,G and B channels
yields a histogram of size $3*16 = 48$.

Color Histogram Example

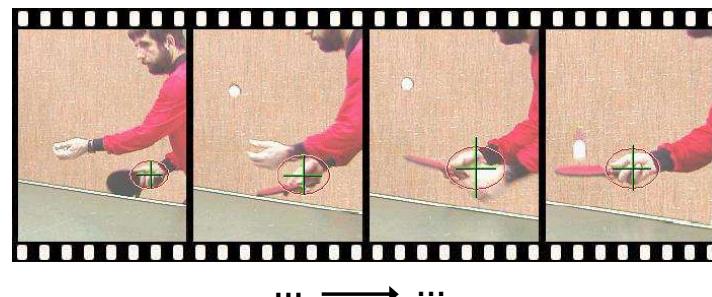


Normalized Color



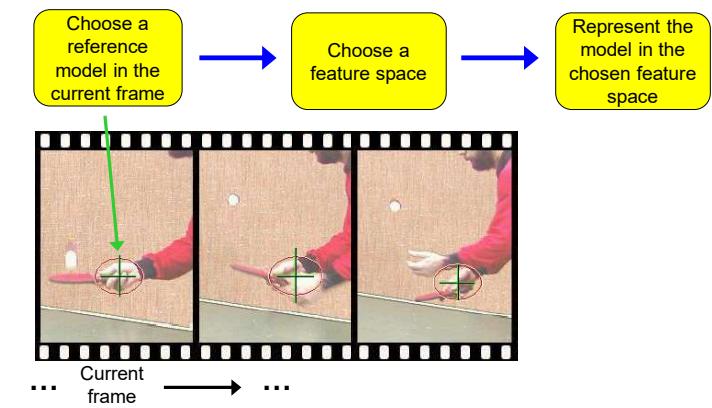
Normalized color divides out pixel luminance (brightness), leaving behind only chromaticity (color) information. The result is less sensitive to variations due to illumination/shading.

Non-Rigid Object Tracking



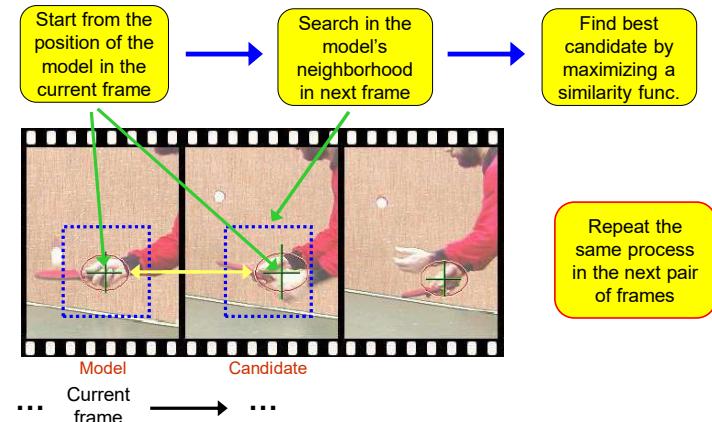
Mean-Shift Object Tracking

General Framework: Target Representation



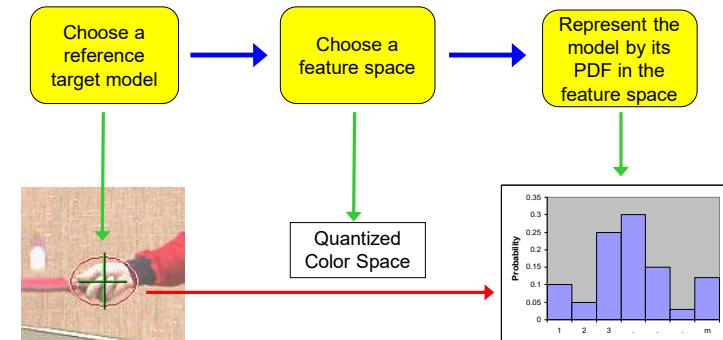
Mean-Shift Object Tracking

General Framework: Target Localization



Mean-Shift Object Tracking

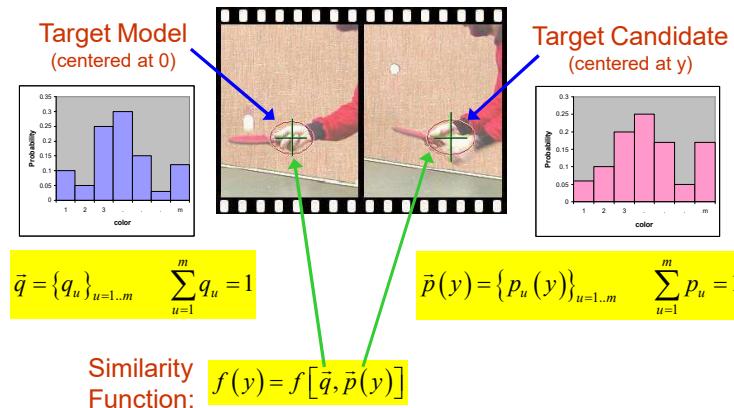
Target Representation



Kernel Based Object Tracking, by Comaniciu, Ramesh, Meer

Mean-Shift Object Tracking

PDF Representation

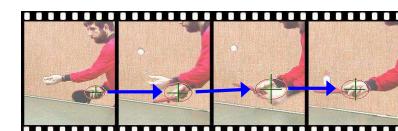


Mean-Shift Object Tracking

Adaptive Scale

Problem:

- The scale of the target changes in time
- The scale (h) of the kernel must be adapted



Solution:

- Run localization 3 times with different h
- Choose h that achieves maximum similarity



Mean-Shift Object Tracking Results

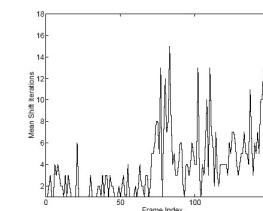


Feature space: 16×16×16 quantized RGB
Target: manually selected on 1st frame
Average mean-shift iterations: 4

Mean-Shift Object Tracking Results



Partial occlusion Distraction Motion blur



Computer Vision

Topics:
BoW Model, Scene Labelling, Learning
Image Similarity

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

1

Agenda for the sessions

- Bag-of-Visual Words
- Similarity Learning

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

2

Image classification

Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

- Image classification: assigning a class label to the image



- Object localization: define the location and the category



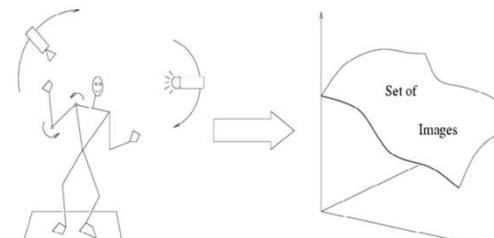
Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

3

Difficulties: within object variations

Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble



Variability: Camera position, Illumination, Internal parameters

Within-object variations

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

4



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Difficulties: within class variations



Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

5



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Image classification

- Given

Positive training images containing an object class



Negative training images that don't



- Classify

A test image as to whether it contains the object class or not



Bag-of-Visual-Words

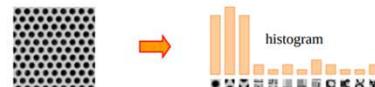
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

6



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features – Origin: texture recognition



Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

7



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag of Words Model

The last duel

After quarrelling over a bank loan, two men took part in the last fatal duel staged on Scottish soil. BBC News's James Landale retraces the steps of his ancestor, who died in that final challenge.

	Loan	Bank	Land	Water	Farmer
Document	1	0	0	0	0
Farmer	0	0	0	0	1

West Bank water row

Palestinians have accused Israel of diverting water away from their towns in order to keep Jewish settlements in the occupied territories fully supplied. Israel denies the charge, saying Palestinian farmers are to blame for using illegal connections to irrigate their fields.

	Bank	Land	Water	Farmer
Document	0.5	0	0.5	0.5
Farmer	0	0	0	0.2

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

8



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag of Visual Words model



11

Bag-of-Visual-Words

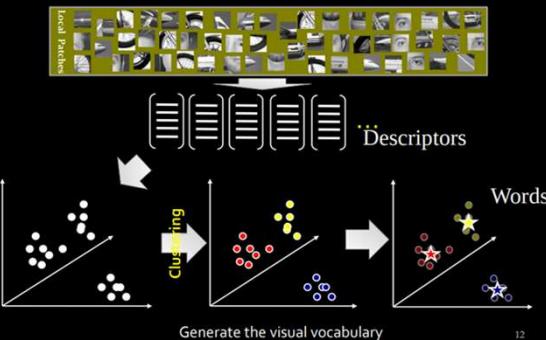
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

9



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag of Visual Words model



12

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

10



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag of Visual Words model



Represent an image as a
histogram or bag of words

13

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

11



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features – Origin: bag-of-words (text)

- Orderless document representation: frequencies of words from a dictionary
- Classification to determine document categories

d1	d2	d3	d4
common people	sculpture	sculpture common	common
people		sculpture	common
common		sculpture	people
people		common	common

Bag-of-words

Common	2	0	1	3
People	3	0	0	2
Sculpture	0	1	3	0
...

Bag-of-Visual-Words

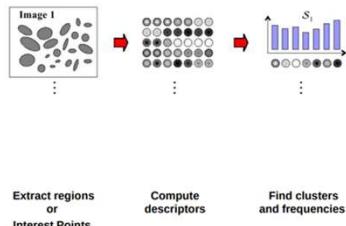
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

12



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features for image classification



Extract regions or
Interest Points Compute
descriptors Find clusters
and frequencies

Bag-of-Visual-Words

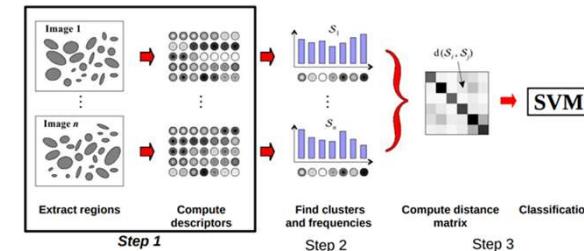
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

13



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features for image classification



Extract regions Compute
descriptors Find clusters
and frequencies Compute distance
matrix Classification

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

14



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Step 1: feature extraction

- Detect Interest Points
 - SIFT
 - Harris
 - Dense (take every nth pixel as interest point)
- Compute Descriptor around each interest point
 - SIFT
 - HOG

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

15



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Dense features



Bag-of-Visual-Words

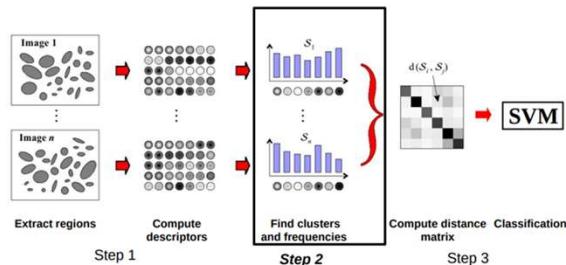
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

16



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features for image classification

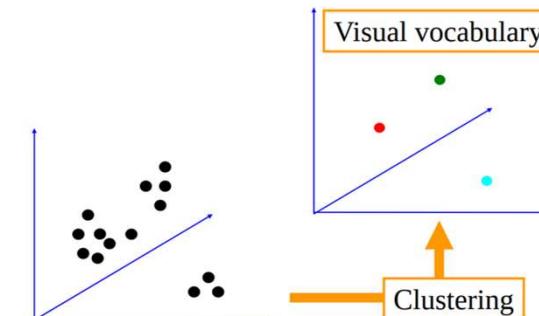


Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

17

Step 2: Quantization



Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

18



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Step 2: Quantization

- Cluster descriptors
 - K-means
- Assign each visual word to a cluster
- Build frequency histogram

Bag-of-Visual-Words

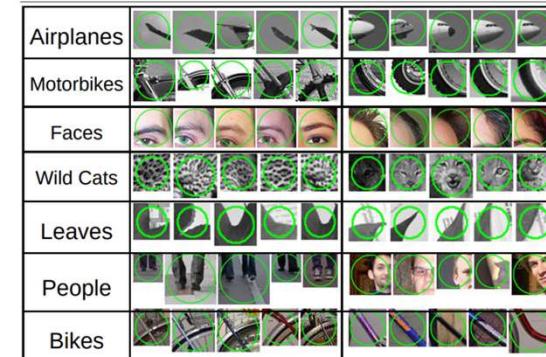
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

19



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Examples for visual words



Bag-of-Visual-Words

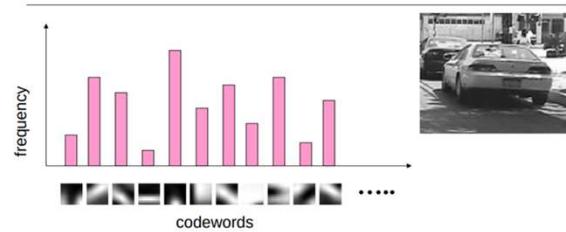
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

20



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Image representation



- each image is represented by a vector, typically 1000-4000 dimension,

Bag-of-Visual-Words

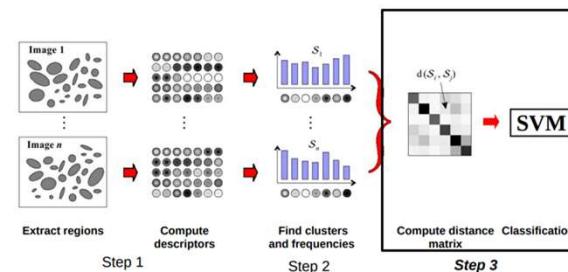
S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

21



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Bag-of-features for image classification



Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

22



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Step 3: Classification

- Learn a decision rule (classifier) assigning bag-of-features representations of images to different classes

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

23



Slides Credit: Cordelia Schmid LEAR – INRIA Grenoble

Training data

Vectors are histograms, one from each training image



Train classifier, e.g. SVM

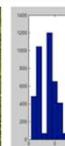
Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

24



All of these images have the same color histogram!



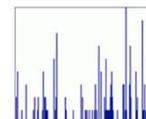
How can we encode the spatial layout?

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

25

Spatial Pyramid representation



Lazebnik, Schmid & Ponce (CVPR 2006)

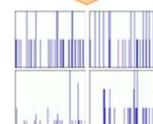
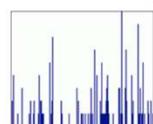
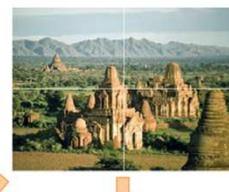
Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

26



Spatial Pyramid representation



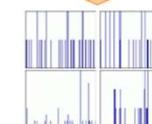
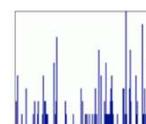
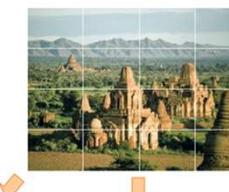
Lazebnik, Schmid & Ponce (CVPR 2006)

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

27

Spatial Pyramid representation



Lazebnik, Schmid & Ponce (CVPR 2006)

Bag-of-Visual-Words

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

28

Word Image Retrieval using Bag of Visual Words

Ravi Shekhar and C.V. Jawahar
Center for Visual Information Technology, IIT Hyderabad, India
Email: ravi.shekhar@research.iit.ac.in, jawahar@iit.ac.in

Figure 1. Bag of Visual Words Representation. Left: Input Images. Middle: Feature Extraction. Right Upper: Code Book Generation. Right Lower: Histogram Computation.

Bag-of-Visual-Words → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 29

Word Image Retrieval using Bag of Visual Words

Ravi Shekhar and C.V. Jawahar
Center for Visual Information Technology, IIT Hyderabad, India
Email: ravi.shekhar@research.iit.ac.in, jawahar@iit.ac.in

Figure 2. Overview of the indexing and retrieval.

Bag-of-Visual-Words → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 30

Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

- Learn a similarity function

Introduction → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 31

Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

$d(A, B) > \tau$ Not the same person

Introduction → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 32

Source: Prof. Leal-Taixé and Prof. Niessner

Learning Similarity

Same person $d(A, B) < \tau$

A 

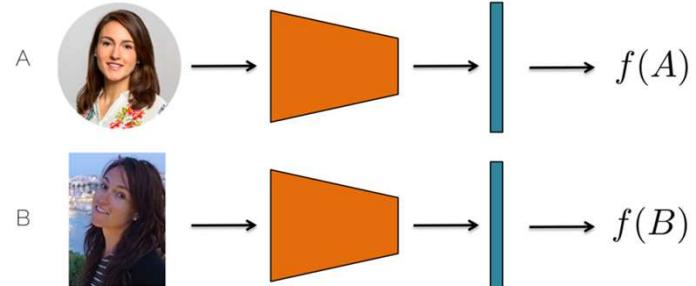
B 

Introduction  S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 33

Source: Prof. Leal-Taixé and Prof. Niessner

Learning Similarity

- How do we train a network to learn similarity?



A 

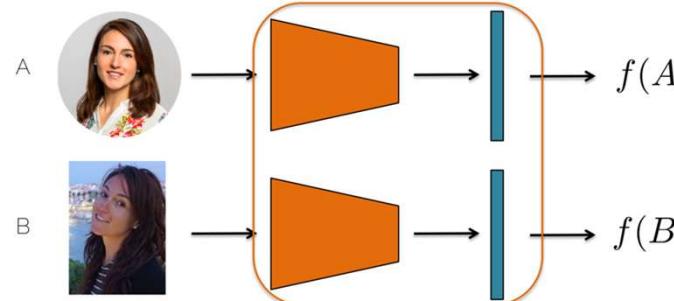
B 

Introduction  S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 34

Source: Prof. Leal-Taixé and Prof. Niessner

Learning Similarity

- Siamese network = shared weights



A 

B 

Introduction  S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 35

Source: Prof. Leal-Taixé and Prof. Niessner

Learning Similarity

- Siamese network = shared weights
- We use the same network to obtain an encoding of the image $f(A)$
- To be done: compare the encodings

Introduction  S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 36



Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

- Distance function $d(A, B) = \|f(A) - f(B)\|^2$
- Training: learn the parameter such that
 - If A and B depict the same person, $d(A, B)$ is small
 - If A and B depict a different person, $d(A, B)$ is large

Introduction

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

37



Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

- Loss function for a positive pair:
 - If A and B depict the same person, $d(A, B)$ is small

$$\mathcal{L}(A, B) = \|f(A) - f(B)\|^2$$

Contrastive Loss

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

38



Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

- Loss function for a negative pair:
 - If A and B depict a different person, $d(A, B)$ is large
 - Better use a Hinge loss:

$$\mathcal{L}(A, B) = \max(0, m^2 - \|f(A) - f(B)\|^2)$$

If two elements are already far away, do not spend energy in pulling them even further away

Contrastive Loss

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

39



Learning Similarity

Source: Prof. Leal-Taixé and Prof. Niessner

- Contrastive loss:

$$\mathcal{L}(A, B) = y^* \|f(A) - f(B)\|^2 + (1 - y^*) \max(0, m^2 - \|f(A) - f(B)\|^2)$$

Positive pair,
reduce the distance
between the
elements

Negative pair,
brings the elements
further apart up to a
margin

Contrastive Loss

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

40



Learning Similarity

- Training the siamese networks
 - You can update the weights for each channel independently and then average them
- This loss function allows us to learn to bring positive pairs together and negative pairs apart

Contrastive Loss

Source: Prof. Leal-Taixé and Prof. Niessner

41

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity

- Triplet loss allows us to learn a ranking



Anchor (A)



Positive (P)



Negative (N)

We want: $\|f(A) - f(P)\|^2 < \|f(A) - f(N)\|^2$

Triplet Loss

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

42



Learning Similarity

- Triplet loss allows us to learn a ranking

$$\|f(A) - f(P)\|^2 < \|f(A) - f(N)\|^2$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 < 0$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m < 0$$



Schroff et al., 'FaceNet: a unified embedding for face recognition and clustering'. CVPR 2015

Triplet Loss

Source: Prof. Leal-Taixé and Prof. Niessner

43

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity

- Triplet loss allows us to learn a ranking

$$\|f(A) - f(P)\|^2 < \|f(A) - f(N)\|^2$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 < 0$$

$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m < 0$$

$$\mathcal{L}(A, P, N) = \max(0, \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m)$$

Schroff et al., 'FaceNet: a unified embedding for face recognition and clustering'. CVPR 2015

Triplet Loss

Source: Prof. Leal-Taixé and Prof. Niessner

44

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity

- Hard negative mining: training with hard cases

$$\mathcal{L}(A, P, N) = \max(0, \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m)$$

- Train for a few epochs
- Choose the hard cases where $d(A, P) \approx d(A, N)$
- Train with those to refine the distance learned

Triplet Loss

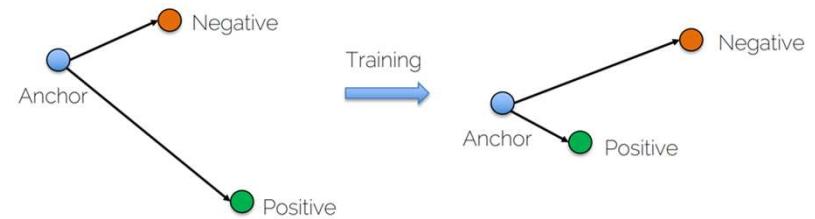
Source: Prof. Leal-Taixé and Prof. Niessner

45

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity



Triplet Loss

Source: Prof. Leal-Taixé and Prof. Niessner

46

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity

- Just do nearest neighbor search!



Triplet Loss

Source: Prof. Leal-Taixé and Prof. Niessner

47

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)



Learning Similarity

- Random sampling does not work - the number of possible triplets is $O(n^3)$ so the network would need to be trained for a very long time.
- Even with hard negative mining, there is the risk of being stuck in local minima.

Challenges in Triplet Loss

Source: Prof. Leal-Taixé and Prof. Niessner

48

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

 Source: Prof. Leal-Taixé and Prof. Niessner

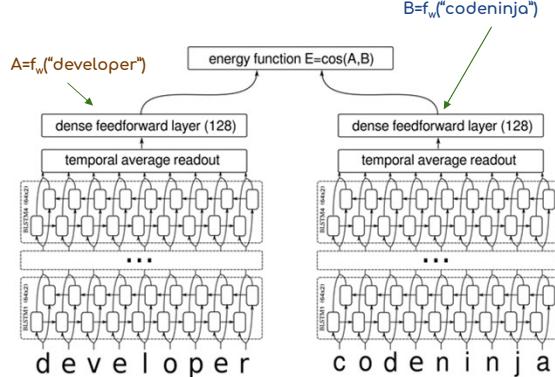
Learning Similarity

- Loss:
 - Contrastive vs. triplet loss
- Sampling:
 - Choosing the best triplets to train with, sample the space wisely
 - diversity of classes + hard cases
- Ensembles:
 - Why not using several networks, each of them trained with a subset of triplets?
- Can we use a classification loss for similarity learning?

Challenges in Triplet Loss → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 49

 From: Learning Text Similarity with Siamese Recurrent Networks Paul Neculoiu, Maarten Versteegh and Mihai Rotaru

Similarity on Text



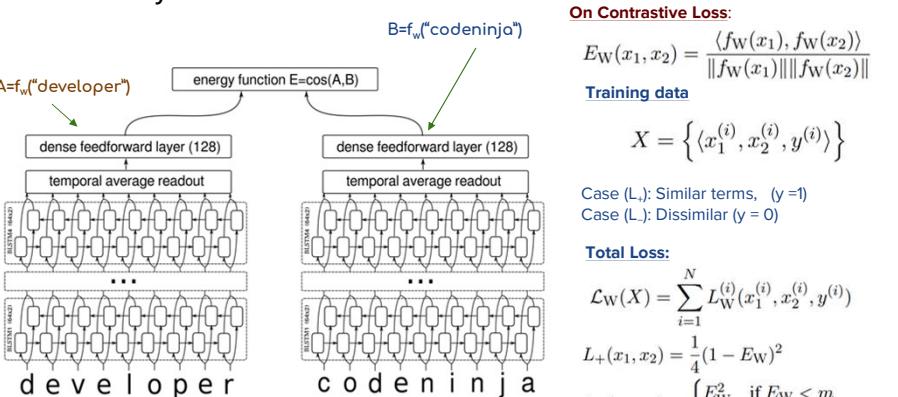
About the Network:

- Four layers of Bidirectional LSTM nodes.
- Activations at each timestep of the final BLSTM layer are averaged to produce a fixed-dimensional output
- Output is projected through a single densely connected feedforward layer

Challenges in Triplet Loss → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 50

 From: Learning Text Similarity with Siamese Recurrent Networks Paul Neculoiu, Maarten Versteegh and Mihai Rotaru

Similarity on Text



On Contrastive Loss:

$$E_W(x_1, x_2) = \frac{\langle f_W(x_1), f_W(x_2) \rangle}{\|f_W(x_1)\| \|f_W(x_2)\|}$$

Training data:

$$X = \left\{ \langle x_1^{(i)}, x_2^{(i)}, y^{(i)} \rangle \right\}$$

Case (L_y): Similar terms, (y = 1)
Case (L_y): Dissimilar (y = 0)

Total Loss:

$$\mathcal{L}_W(X) = \sum_{i=1}^N L_W^{(i)}(x_1^{(i)}, x_2^{(i)}, y^{(i)})$$

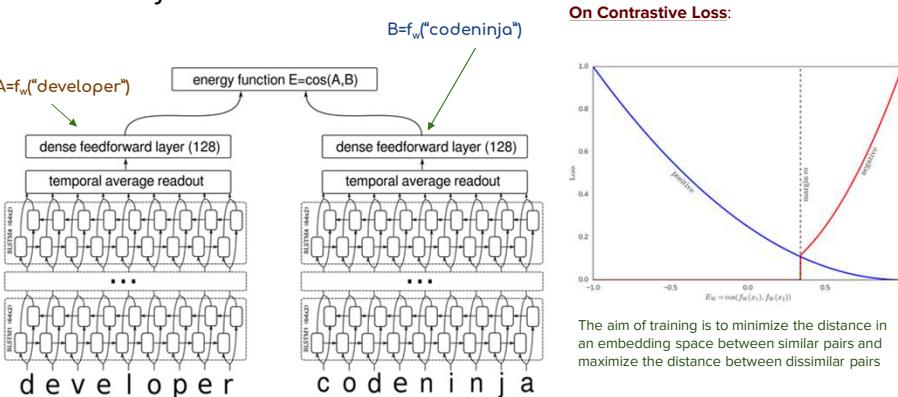
$$L_+(x_1, x_2) = \frac{1}{4}(1 - E_W)^2$$

$$L_-(x_1, x_2) = \begin{cases} E_W^2 & \text{if } E_W < m \\ 0 & \text{otherwise} \end{cases}$$

Challenges in Triplet Loss → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 51

 From: Learning Text Similarity with Siamese Recurrent Networks Paul Neculoiu, Maarten Versteegh and Mihai Rotaru

Similarity on Text



On Contrastive Loss:

The aim of training is to minimize the distance in an embedding space between similar pairs and maximize the distance between dissimilar pairs

Challenges in Triplet Loss → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 52

Similarity on Text

From: Learning Text Similarity with Siamese Recurrent Networks Paul Neculoiu, Maarten Versteegh and Mihai Rotaru

On Contrastive Loss:

The aim of training is to minimize the distance in an embedding space between similar pairs and maximize the distance between dissimilar pairs

Challenges in Triplet Loss

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

53

Introduction

Some Applications

Hot topic detection
Personalized recommendation
Multi-modal data retrieval

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

54

Introduction

Sample Training Data →

In southeastern Washington, a stretch of the river passes through the Hanford Site, established in 1943 as part of the Manhattan Project. The site served as a plutonium production complex, with nine nuclear reactors and related facilities located on the banks of the river. From 1944 to 1971, pump systems drew cooling water from the river and, after treating this water for use by the reactors, returned it to the river. Before being released back into the river, the used water was held in large tanks known as retention basins for up to six hours. Longer-lived isotopes were not affected by this retention.(....)

Sample Query & Result →

Until April, the Polish forces had been slowly but steadily advancing eastward. The new Latvian government requested and obtained Polish help in capturing Daugavpils. The city fell after heavy fighting at the Battle of Daugavpils in January and was handed over to the Latvians. By March, Polish forces had driven a wedge between Soviet forces to the north (Belarusia) and south (Ukraine). On April 24, Poland began its main offensive, "Operation Kiev". Its stated goal was the creation of an independent Ukraine that would become part of Piłsudski's project of a "Miedzymorze" Federation.(....)

Cross Model IR

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

55

Introduction

Query →

Multimedia Database →

Results →

Cross Model IR

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

56

Cross Model Retrieval

TextRS: Deep Bidirectional Triplet Network for Matching Text to Remote Sensing Images

Taghreed Abdullah¹, Yakoub Bazi^{2,*}, Mohamad M. Al Rahhal³, Mohamed L. Mekhalfi⁴, Lalitha Rangarajan¹ and Mansour Zuair²

- 1. A semi crowded airport with aircraft loading
2. An airport radar with green surroundings
3. Many aircrafts parked near the loading area
4. Aircrafts approaching the loading area
5. An airport radar located next to the terminal
- 1. A large bridge crossing a canal
2. Cars appearing at the end of the bridge
3. Vehicles appearing on the bridge
4. A bridge supported by two base pillars
5. The bridge has one white line in the middle
- 1. A river crossing a desert
2. A rocky appearance near the river at the desert
3. Green trees near the river in a desert
4. Patterns appearing on the sands near the river
5. A plain sandy area nearby a river in the desert
- 1. A line of trees edging a port
2. Boats of multiple sizes parked at the port
3. A boat outside the boat parking area at a port
4. Buildings appearing nearby the port
5. A view of calm water at the port
- 1. Bridges leading to a railway station
2. A view of city railway station
3. Buildings are located near a railway station
4. A river crossed by tracks at a railway station
5. A main road passing by a railway station
- 1. Multilane freeway in a populated area
2. Green grass surrounding the freeway
3. Buildings nearby the freeway lanes
4. Vehicles on the lanes of a freeway
5. Cars parked beside buildings near the freeway
- 1. A beach with a wave moving along
2. A dry sandy area near the beach
3. Dry bushes on the sands
4. Vehicles on the sands at the beach
5. A road path on the beach sands
- 1. A curved road in a golf course
2. An open dry area next to the road
3. A line of grass edging the road
4. Sandy areas beside the road in a golf course
5. A vehicle on the road in a golf course

TextRS, A Case of CM-IR → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 57

Cross Model Retrieval

TextRS: Deep Bidirectional Triplet Network for Matching Text to Remote Sensing Images

Taghreed Abdullah¹, Yakoub Bazi^{2,*}, Mohamad M. Al Rahhal³, Mohamed L. Mekhalfi⁴, Lalitha Rangarajan¹ and Mansour Zuair²

Text as anchor

TextRS, A Case of CM-IR → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 58

Cross Model Retrieval

TextRS: Deep Bidirectional Triplet Network for Matching Text to Remote Sensing Images

Taghreed Abdullah¹, Yakoub Bazi^{2,*}, Mohamad M. Al Rahhal³, Mohamed L. Mekhalfi⁴, Lalitha Rangarajan¹ and Mansour Zuair²

Text as anchor

Image as anchor

TextRS, A Case of CM-IR → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 59

Cross Model Retrieval

TextRS: Deep Bidirectional Triplet Network for Matching Text to Remote Sensing Images

Taghreed Abdullah¹, Yakoub Bazi^{2,*}, Mohamad M. Al Rahhal³, Mohamed L. Mekhalfi⁴, Lalitha Rangarajan¹ and Mansour Zuair²

Image Encoding

EfficientNet-B2:
Take output of Swish4

Conv (3x3)+BN+Swish

670 → 128 → 128

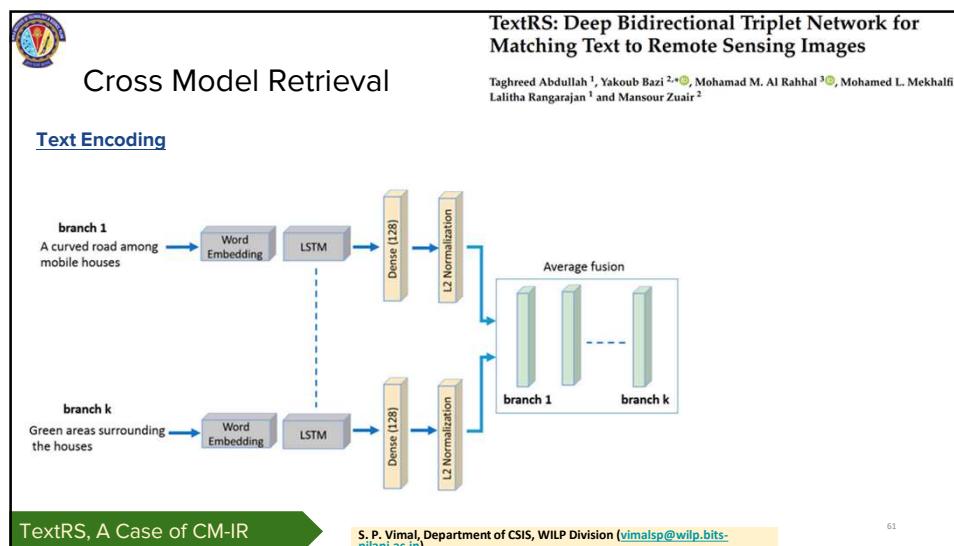
Channel attention using SE

Dense (128)

L2 Normalization

V: 28×28×128
GAP
 $W_1 + \text{ReLU}$
 $W_2 + \text{Sigmoid}$
Scale
28×28×128
 V_{SE}

TextRS, A Case of CM-IR → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in) 60



Cross Model Retrieval

TextRS: Deep Bidirectional Triplet Network for Matching Text to Remote Sensing Images

Taghreed Abdullah¹, Yakoub Bazi^{2,*}, Mohamad M. Al Rahhal³, Mohamed L. Mekhalfi⁴, Lalitha Rangarajan¹ and Mansour Zuair²

Parameter Learning

$$l_{DBTN} = \lambda_1 L_1 + \lambda_2 L_2$$

$$L_1 = \sum_{i=1}^N \left[\|g(T_i^a) - f(X_i^p)\|_2^2 - \|g(T_i^a) - f(X_i^n)\|_2^2 + \alpha \right]_+$$

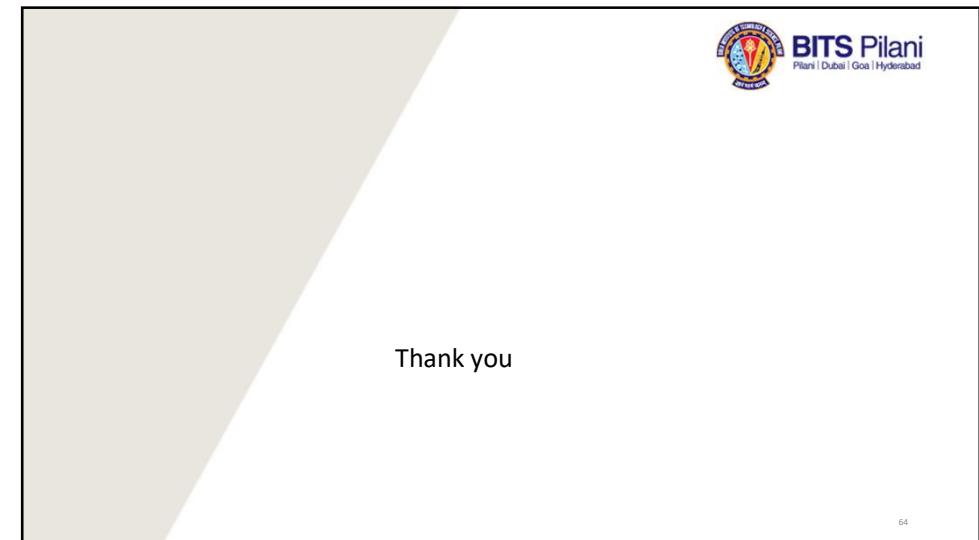
$$L_2 = \sum_{i=1}^N \left[\|f(X_i^a)\| - g(T_i^p)\|_2^2 - \|f(X_i^a) - g(T_i^n)\|_2^2 + \alpha \right]_+$$

$$|z|_+ = \max(z, 0)$$

TextRS, A Case of CM-IR → S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

62

- Recommended Readings & References**
1. Neural IR Tutorials / Surveys
 1. [Neural information retrieval: at the end of the early years](#) (Information Retrieval Journal), 2017 - A survey paper. [your first read]
 2. [An Introduction to Neural Information Retrieval](#), Bhaskar Mitra, Nick Croswell (Microsoft)
 2. Some Cross Model Retrieval Papers Used in this presentation
 1. [A Comprehensive Survey on Cross-modal Retrieval](#) Kaiye Wang, Qiyue Yin, Wei Wang, Shu Wu, Liang Wang
 2. [DRSL: Deep Relational Similarity Learning for Cross-modal Retrieval](#) - Xu Wang, Peng Hu, Liangli Zhen, Dezhong Peng
 3. [HCMSL: Hybrid Cross-Modal Similarity Learning for Cross-Modal Retrieval](#); CHENGYUAN ZHANG, JIAYU SONG, XIAOFENG ZHU, LEI ZHU, SHICHAO ZHANG
 3. hi
- 63





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

AIMLC ZG525

Computer Vision

Note on Performance Metrics & Course Summary



Agenda for the class

- Mean Average Precision (mAP)
- ROC

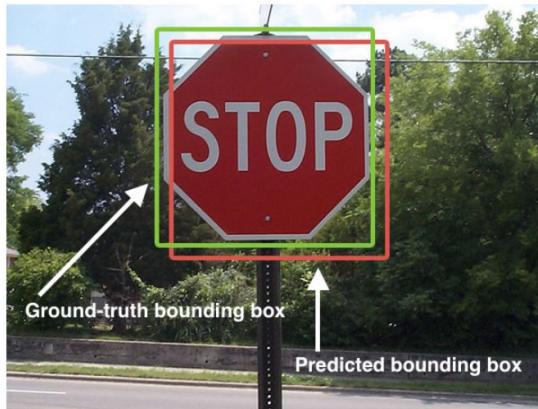
Confusion Matrix (revisiting)

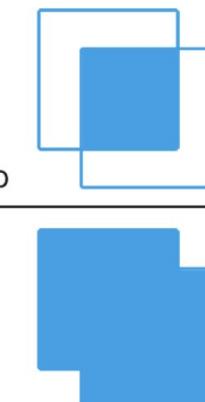
- **Precision:** fraction of retrieved docs that are relevant
 - $P(\text{relevant}|\text{retrieved})$
- **Recall:** fraction of relevant docs that are retrieved
 - $P(\text{retrieved}|\text{relevant})$

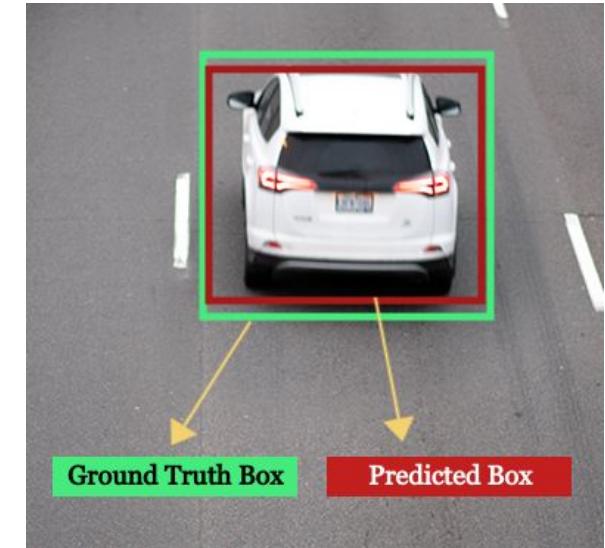
	Relevant	Non-Relevant
Retrieved / Predicted Positive	tp	fp
Not Retrieved / Predicted Negative	fn	tn

- Precision $P = tp/(tp + fp)$
- Recall $R = tp/(tp + fn)$

Intersection over Union - IoU (revisiting)



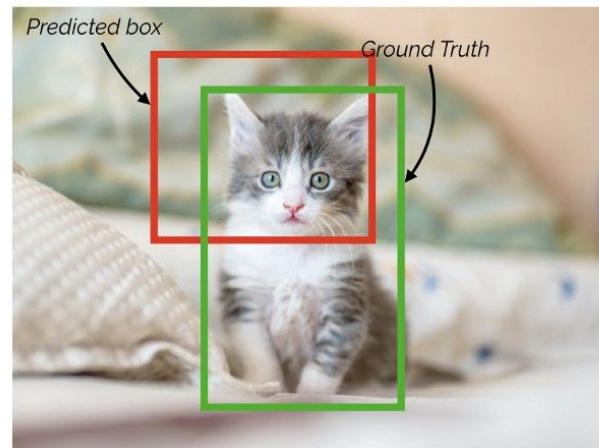
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$




Intersection over Union - IoU (revisiting)

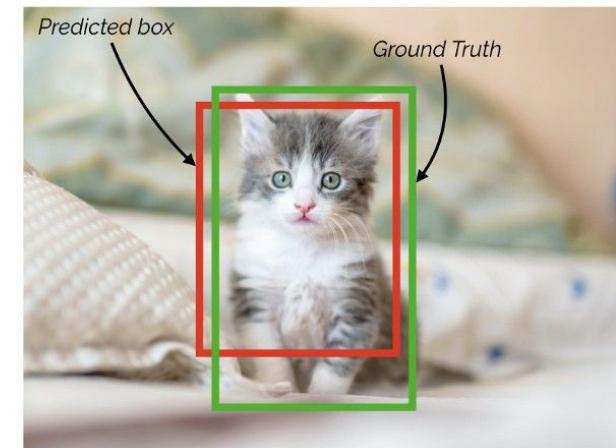
If IoU threshold = 0.5

False Positive (FP)



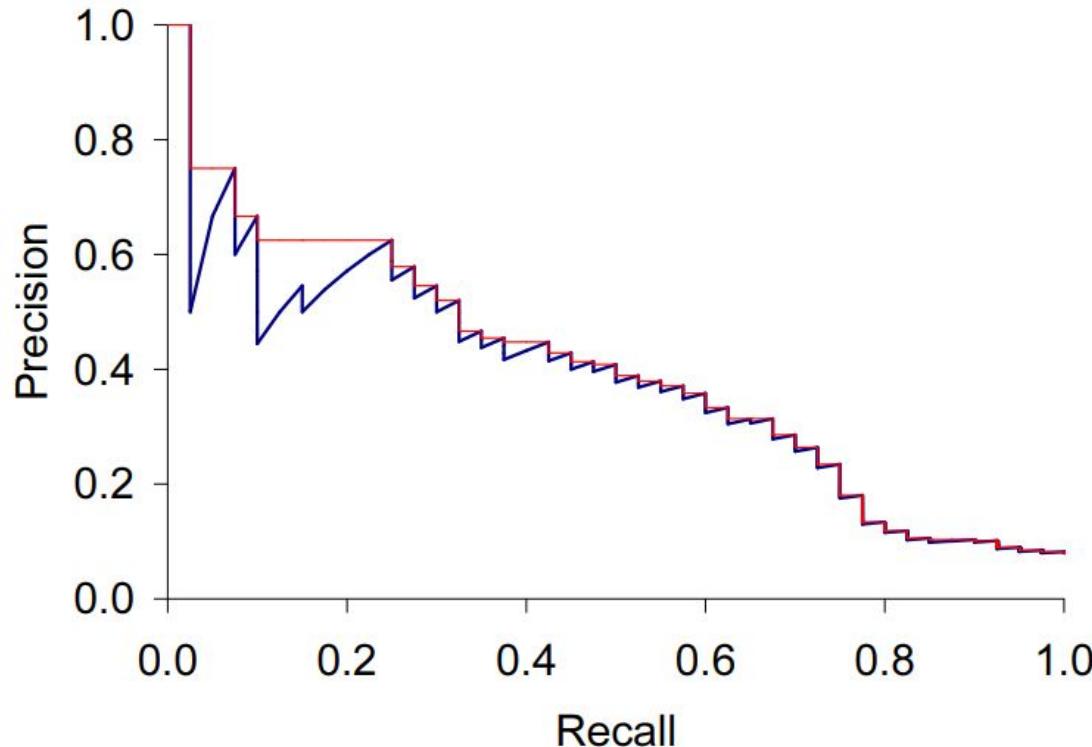
$$IoU = \sim 0.3$$

True Positive (TP)



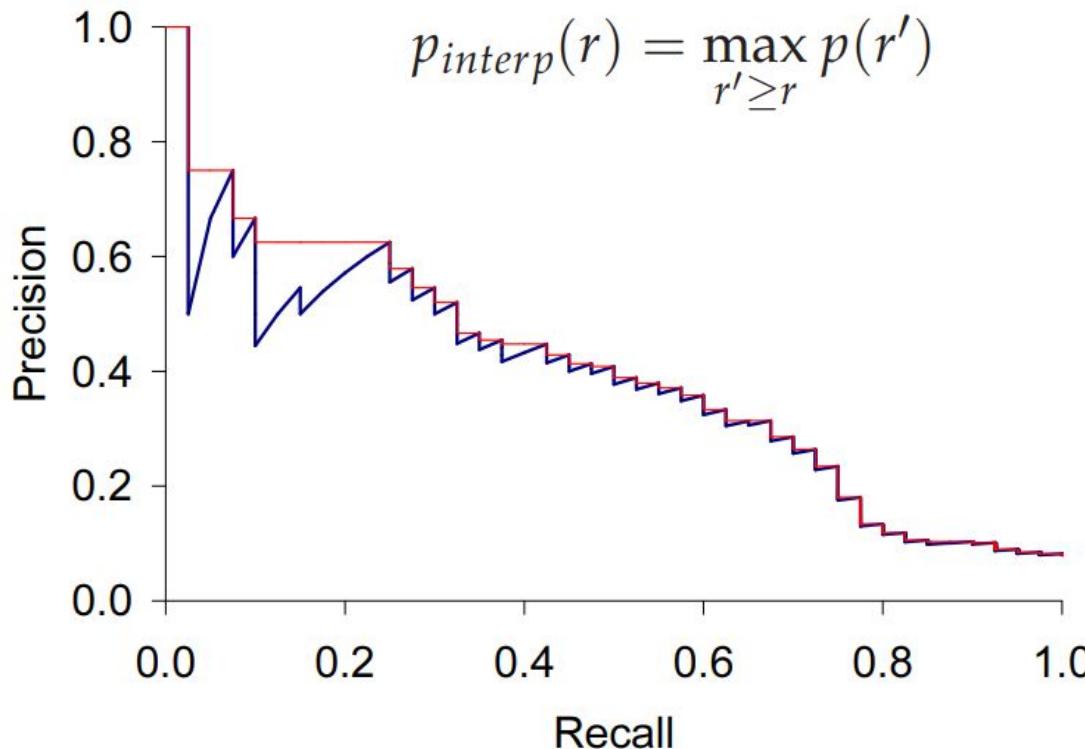
$$IoU = \sim 0.7$$

Precision Recall Curve & purpose

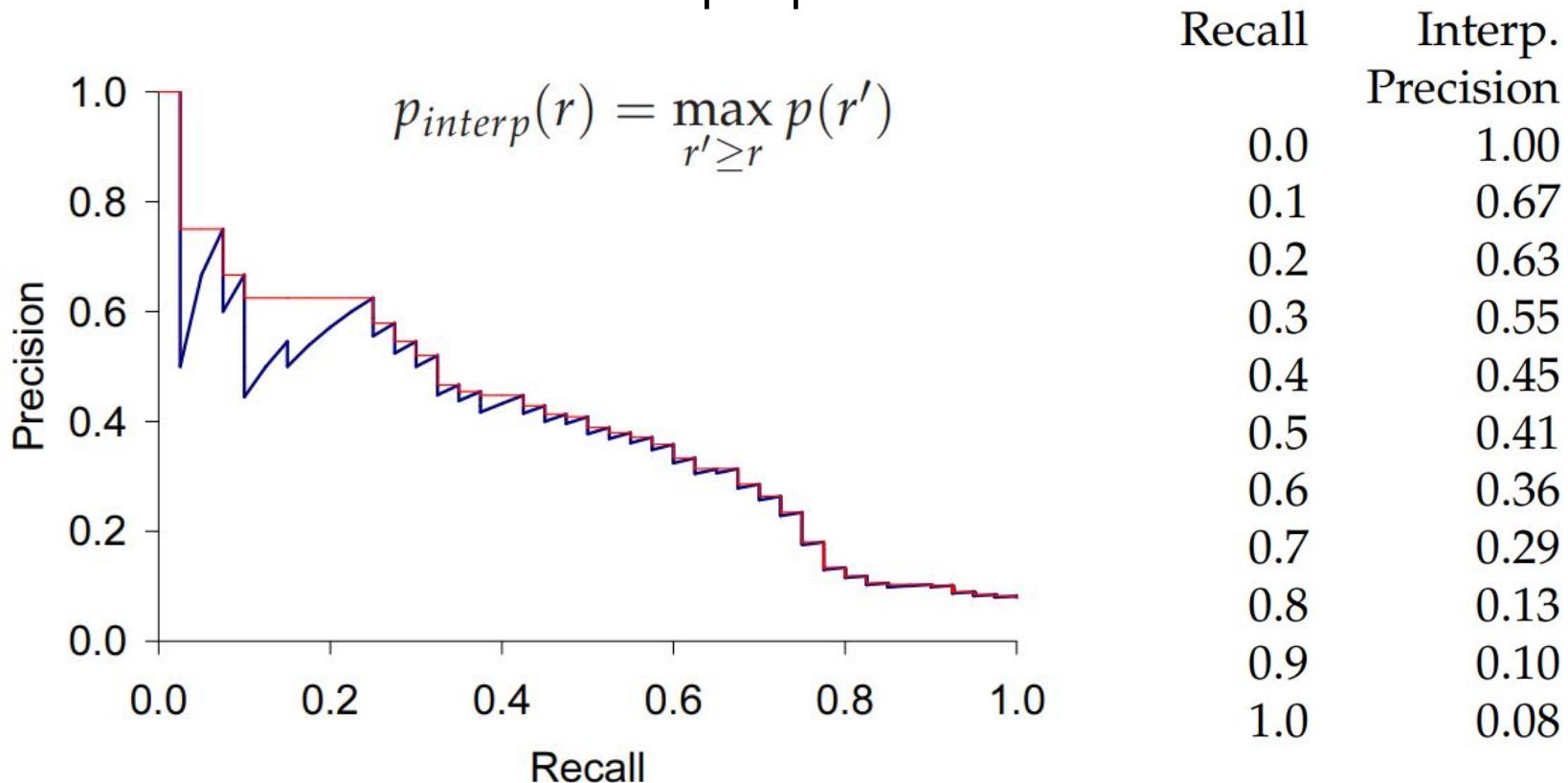


$$P = tp / (tp + fp)$$
$$R = tp / (tp + fn)$$

Precision Recall Curve & purpose

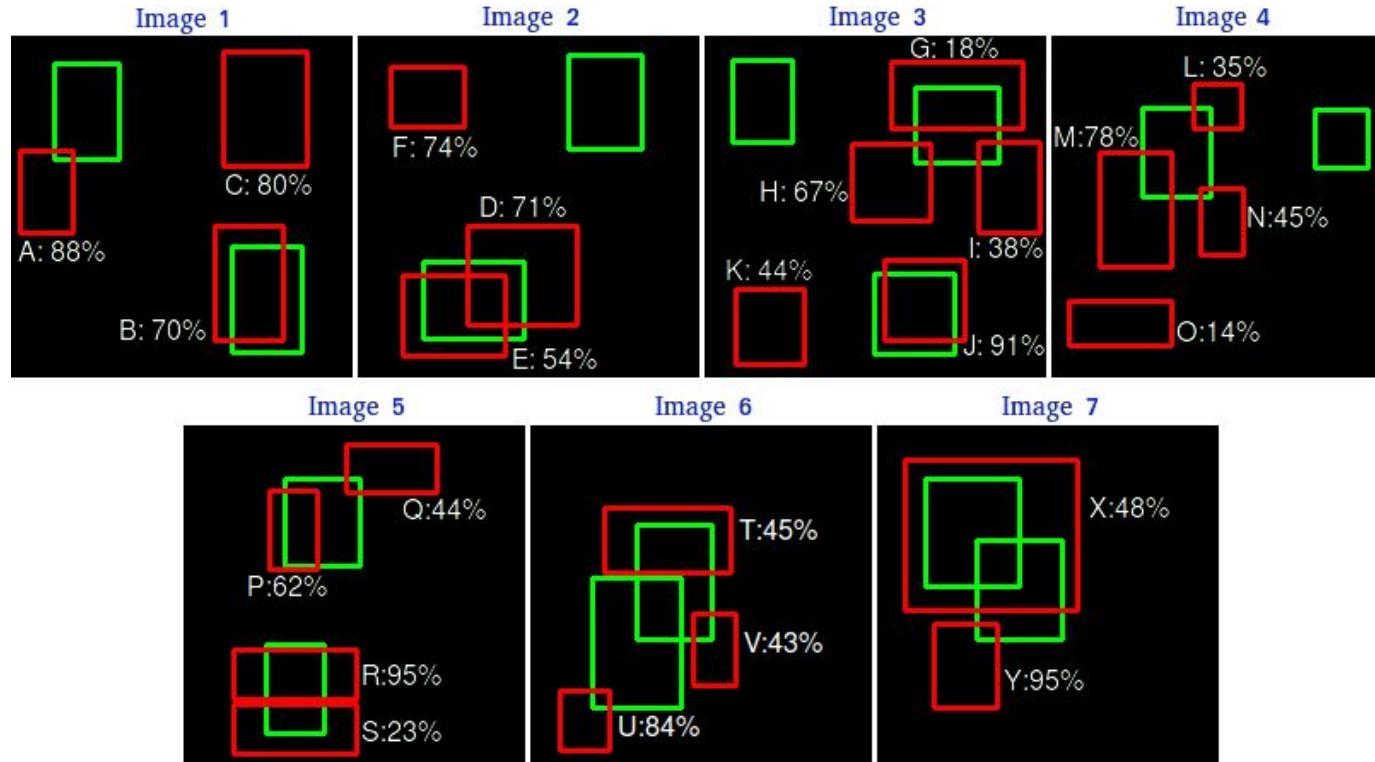


Precision Recall Curve & purpose



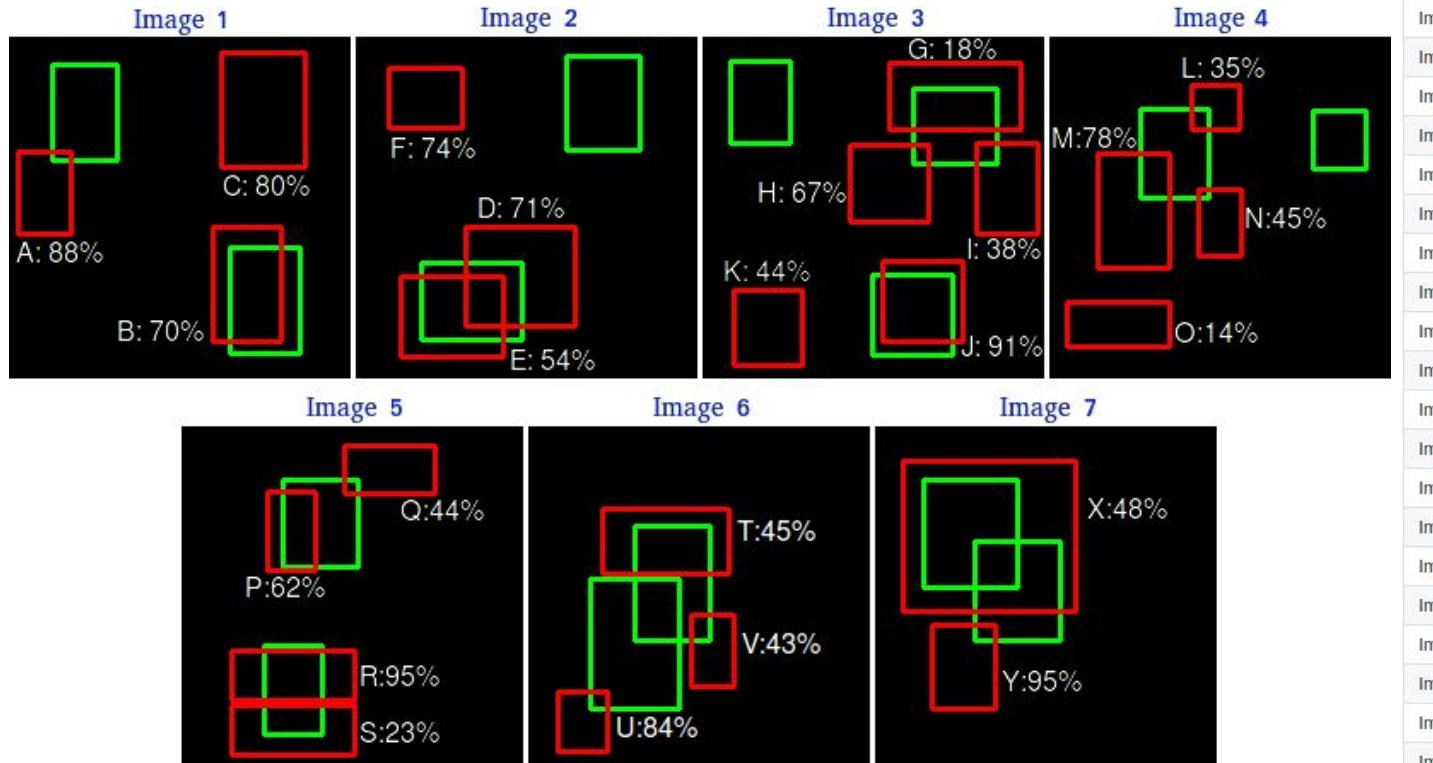
ROI Proposals = 24
Ground Truths = 15

Example:



Example:

ROI Proposals = 24
Ground Truths = 15





Example:

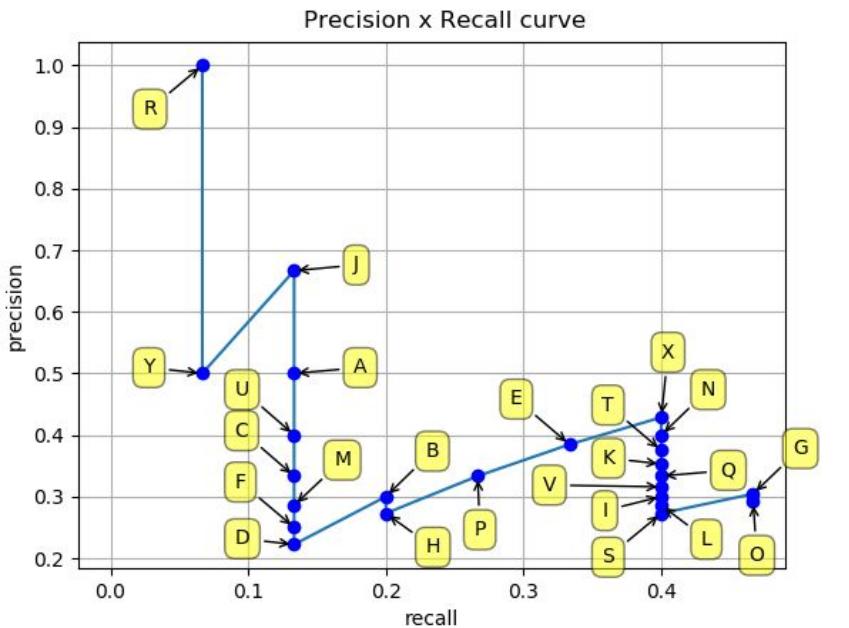
Ground Truths = 15

Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
		14%	0	1	7	17	0.2916	0.4666



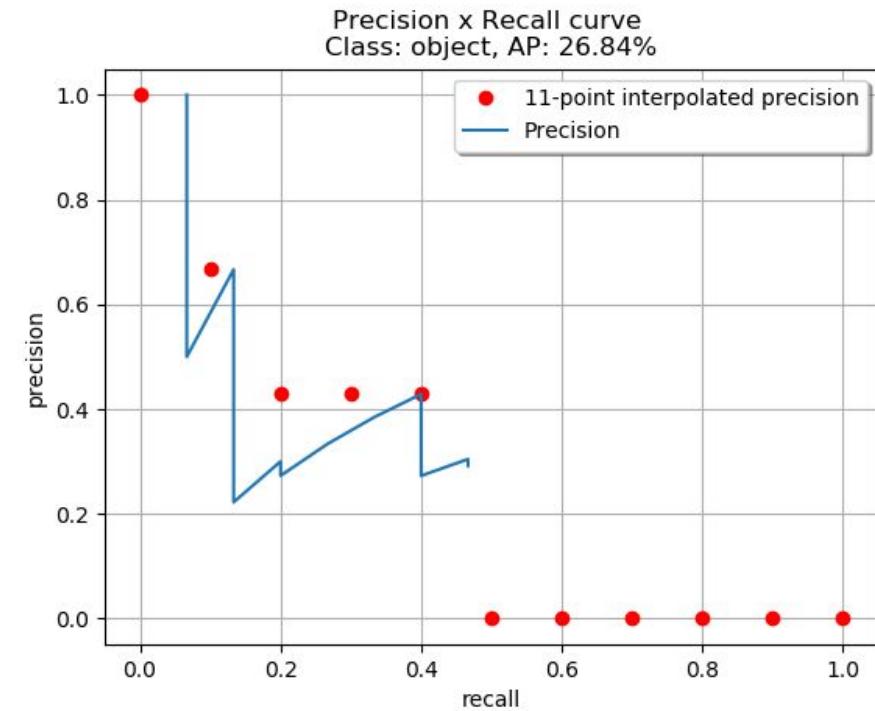
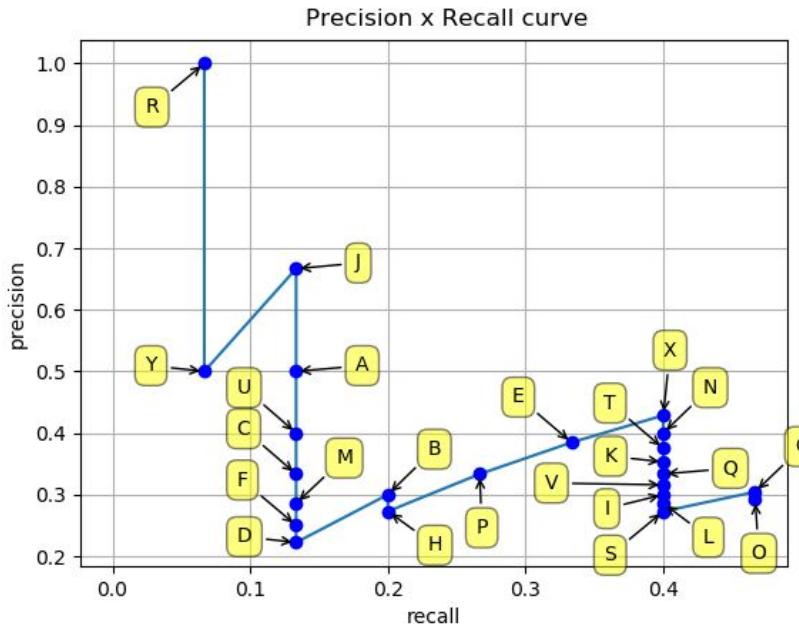
Images	Detections	Confidences	TP or FP
Image 1	A	88%	FP
Image 1	B	70%	TP
Image 1	C	80%	FP
Image 2	D	71%	FP
Image 2	E	54%	TP
Image 2	F	74%	FP
Image 3	G	18%	TP
Image 3	H	67%	FP
Image 3	I	38%	FP
Image 3	J	91%	TP
Image 3	K	44%	FP
Image 4	L	35%	FP
Image 4	M	78%	FP
Image 4	N	45%	FP
Image 4	O	14%	FP
Image 5	P	62%	TP
Image 5	Q	44%	FP
Image 5	R	95%	TP
Image 5	S	23%	FP
Image 6	T	45%	FP
Image 6	U	84%	FP
Image 6	V	43%	FP
Image 7	X	48%	TP
Image 7	Y	95%	FP

Example:

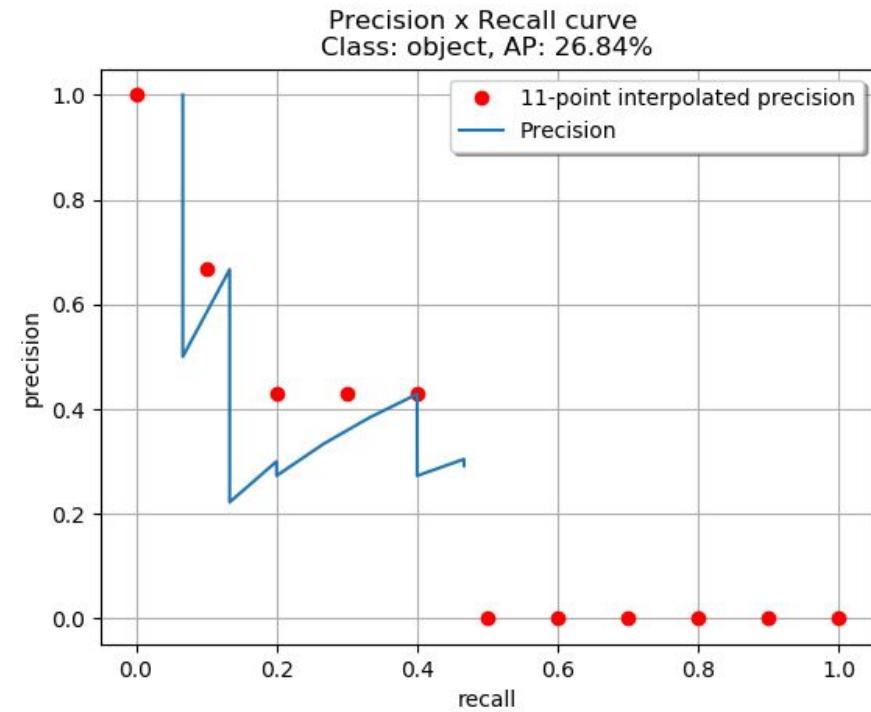
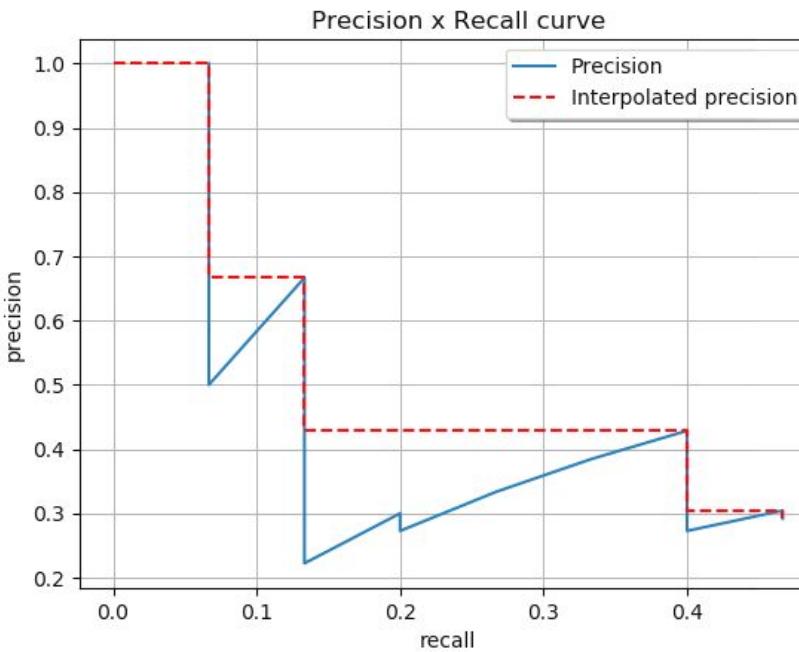


Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
Image 4	O	14%	0	1	7	17	0.2916	0.4666

Example:



Example:

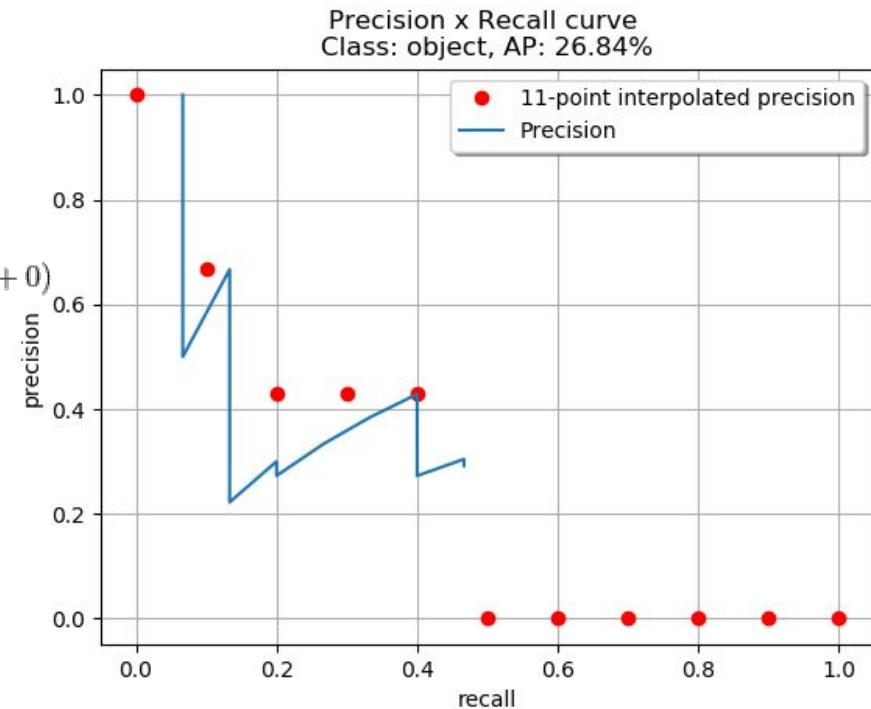


Example:

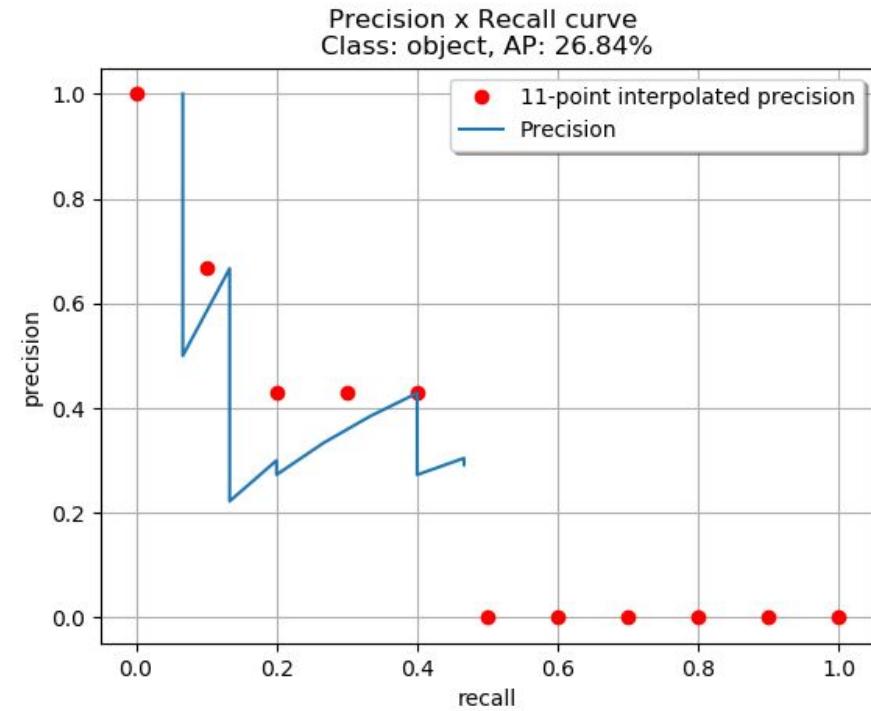
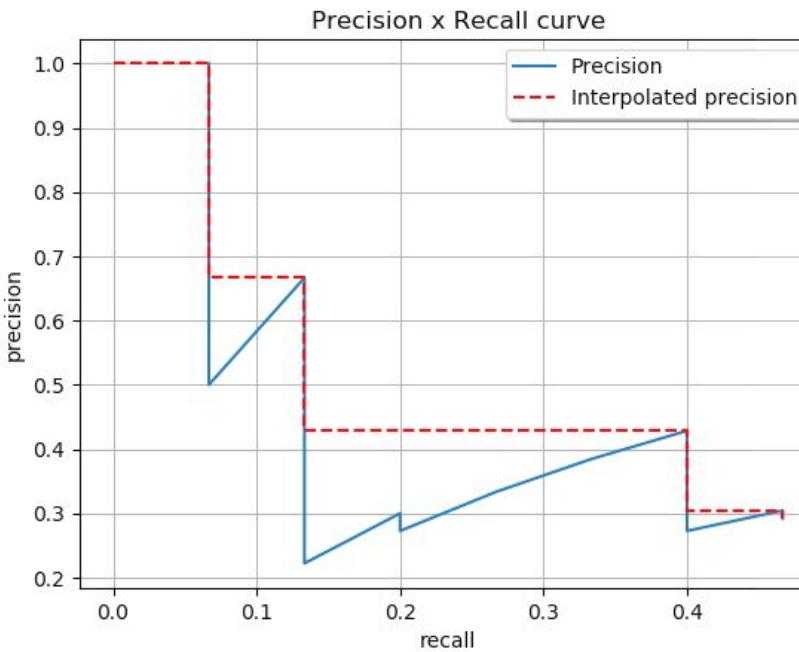
$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} \rho_{\text{interp}}(r)$$

$$AP = \frac{1}{11} (1 + 0.6666 + 0.4285 + 0.4285 + 0.4285 + 0 + 0 + 0 + 0 + 0 + 0)$$

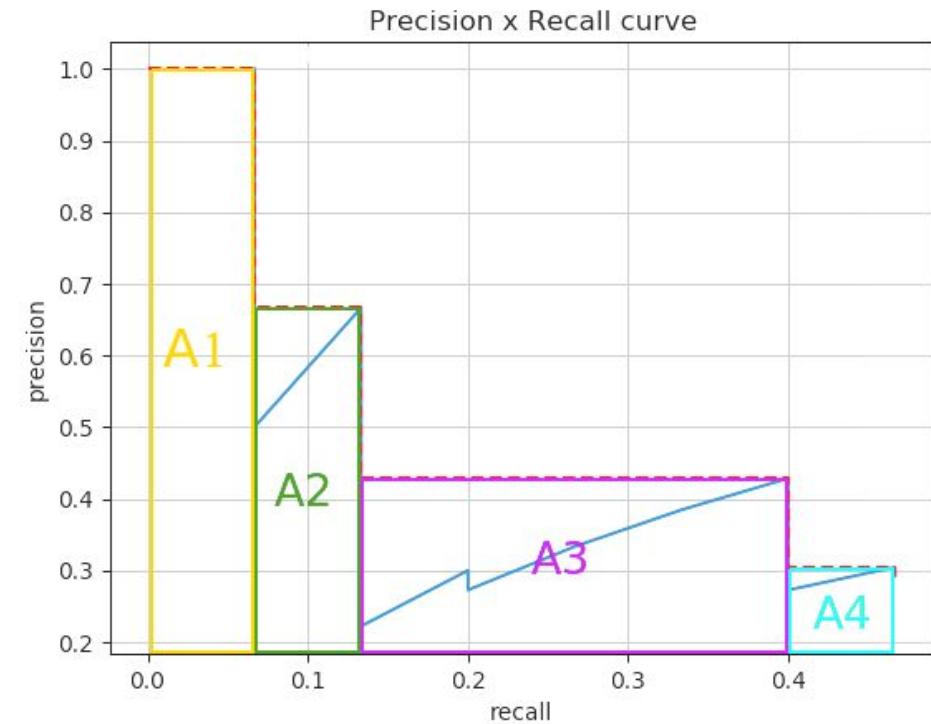
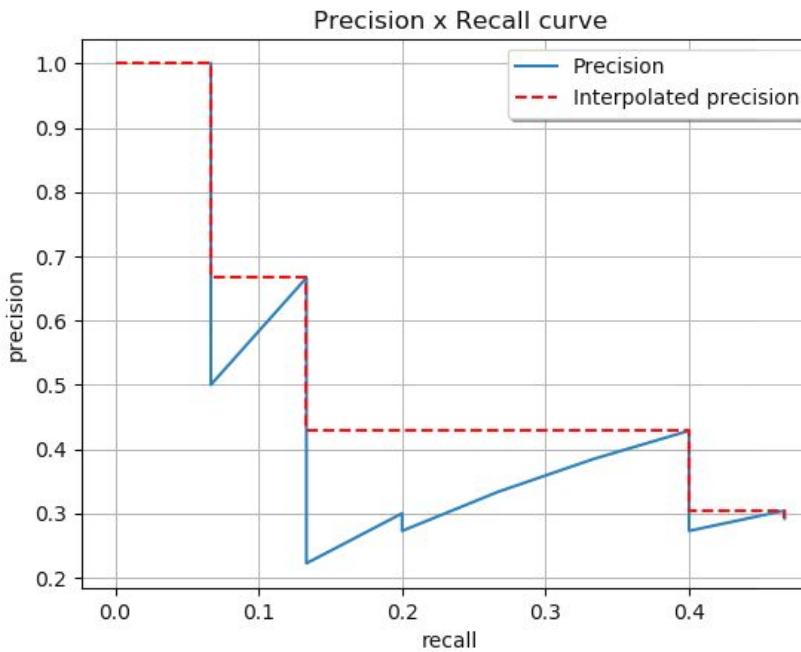
$$AP = 26.84\%$$



Example:



Example:





Example:

	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Dining table	Dog	Horse	Motorbike	Person	Potted plant	Sheep	Sofa	Train	TV/Monitor
CVC_CLS	45.4	49.8	15.7	16.0	26.3	54.6	44.8	35.1	16.8	31.3	23.6	26.0	45.6	49.6	42.2	14.5	30.5	28.5	45.7	40.0
MISSOURI	51.4	53.6	18.3	15.6	31.6	56.5	47.1	38.6	19.5	31.9	22.1	25.0	50.3	51.9	44.9	11.9	37.7	30.6	50.8	39.2
NEC	65.0	46.8	25.0	24.6	16.0	51.0	44.9	51.5	13.0	26.6	<i>31.0</i>	40.2	39.7	51.5	32.8	12.6	35.7	33.5	48.0	44.8
OLB_R5	47.5	51.6	14.2	12.6	27.3	51.8	44.2	25.3	17.8	30.2	18.1	16.9	46.9	50.9	43.0	09.5	31.2	23.6	44.3	22.1
SYSU_DYNAMIC	50.1	47.0	07.9	03.8	24.8	47.2	42.8	31.1	17.5	24.2	10.0	21.3	43.5	46.4	37.5	07.9	26.4	21.5	43.1	36.7
OXFORD	59.6	54.5	21.9	21.6	32.1	52.5	49.3	40.8	<i>19.1</i>	35.1	28.9	37.2	50.9	49.9	46.1	15.6	<i>39.3</i>	<i>35.6</i>	48.9	42.8
UVA_HYBRID	61.8	52.0	24.6	24.8	20.2	57.1	44.5	53.6	17.4	<i>33.0</i>	38.2	42.8	48.8	59.4	35.7	22.8	40.3	39.5	51.1	49.5
UVA_MERGED	47.2	50.2	18.3	21.4	25.2	53.3	46.3	46.3	17.5	27.8	30.3	35.0	41.6	52.1	43.2	18.0	35.2	31.1	45.4	44.4

Methods trained on VOC2012 data. For each object class and submission, the AP score (%) is shown. Bold entries in each column denote the maximum AP for the corresponding class, and italic entries denote the results ranked in second place

Source: <http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham15.pdf>



ROC Analysis

Contents are adopted from :

An introduction to ROC analysis

Tom Fawcett

Institute for the Study of Learning and Expertise, 2164 Staunton Court, Palo Alto, CA 94306, USA

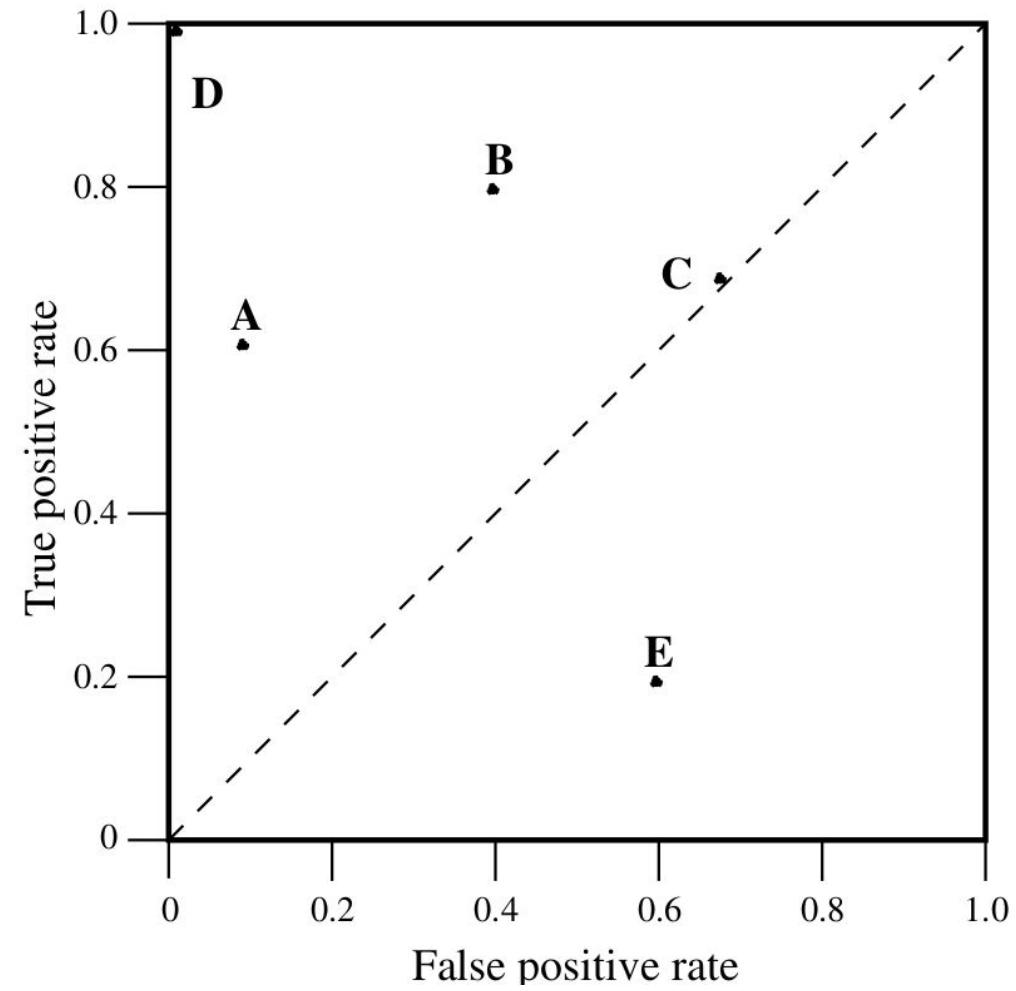
Available online 19 December 2005

Available from <https://people.inf.elte.hu/kiss/13dwhdm/roc.pdf>

Confusion Matrix (Again)

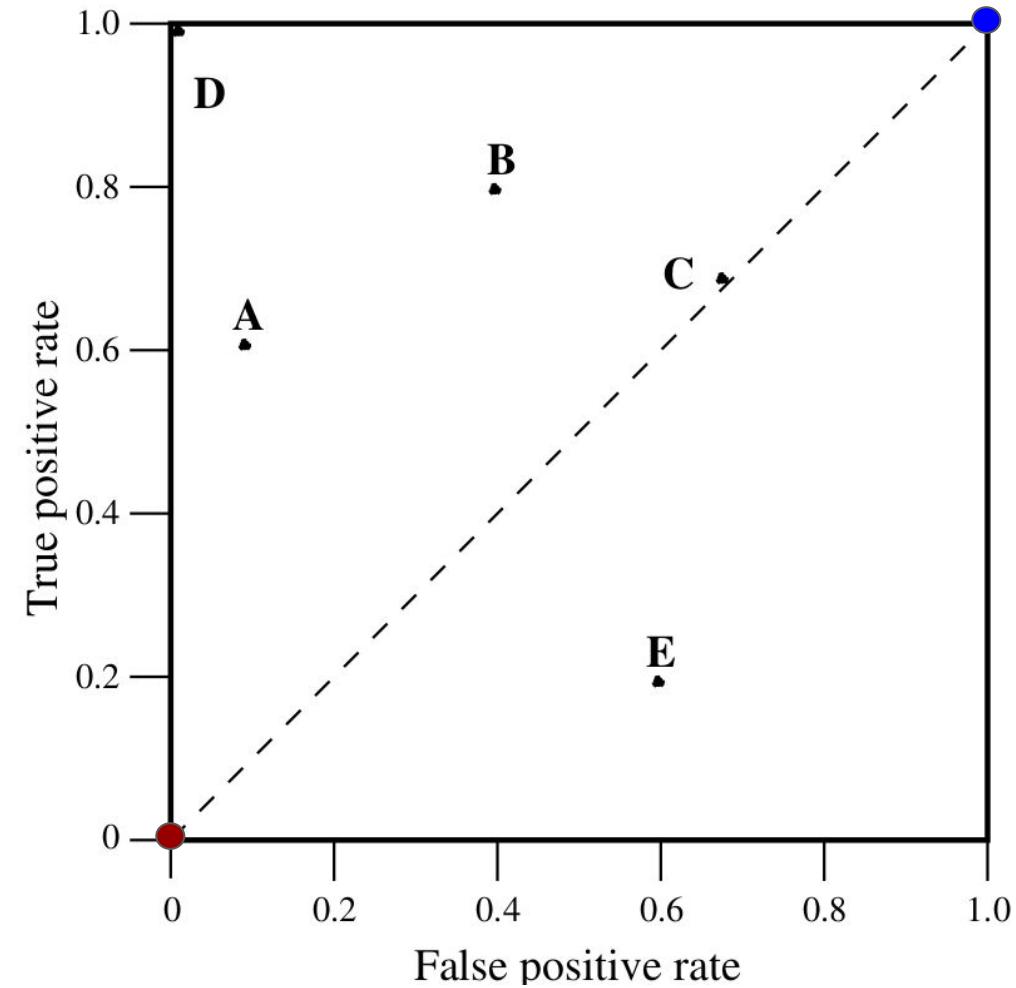
		<u>True class</u>	
		p	n
<u>Hypothesized class</u>	Y	True Positives	False Positives
	N	False Negatives	True Negatives
Column totals:	P	N	$\text{precision} = \frac{TP}{TP+FP}$ $\text{recall} = \frac{TP}{P}$
			$\text{accuracy} = \frac{TP+TN}{P+N}$
			$\text{F-measure} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$

ROC Space



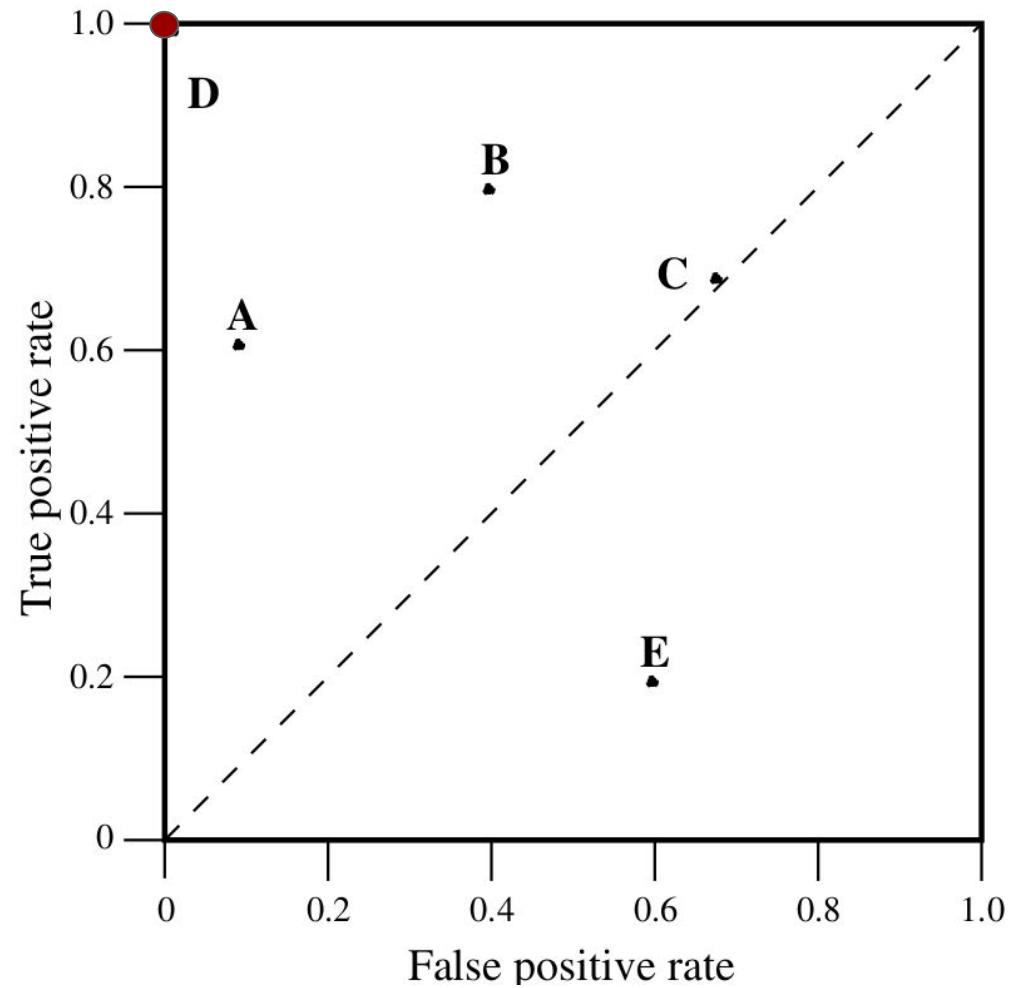
Points in ROC Space

- Lower left point $(0, 0)$ represents the strategy of never issuing a positive classification;
 - Such a classifier commits no false positive errors but also gains no true positives.
- Upper right corner $(1, 1)$ represents the opposite strategy, of unconditionally issuing positive classifications.



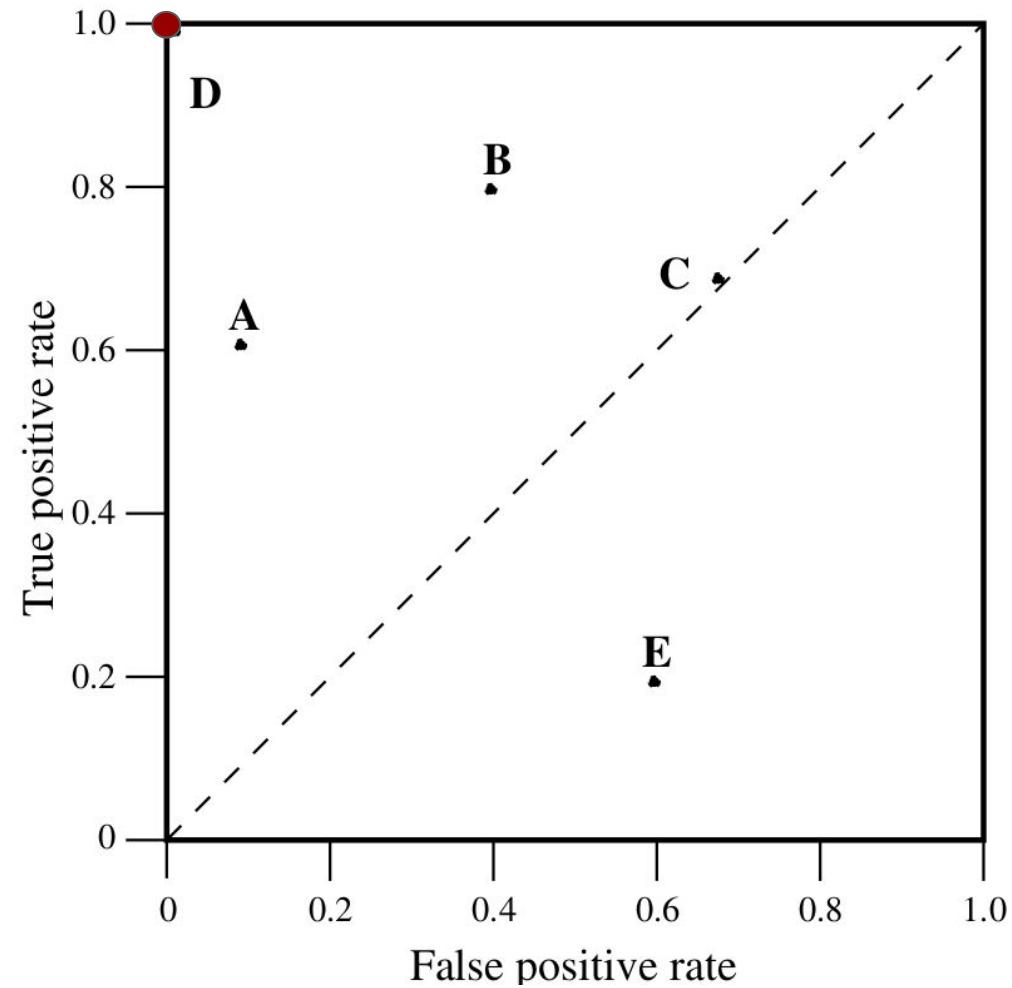
Points in ROC Space

- Point $(0, 1)$ represents perfect classification.
 - D's performance is perfect as shown.
- Informally, one point in ROC space is better than another if it is to the northwest of the first
 - tp rate is higher, fp rate is lower, or both.



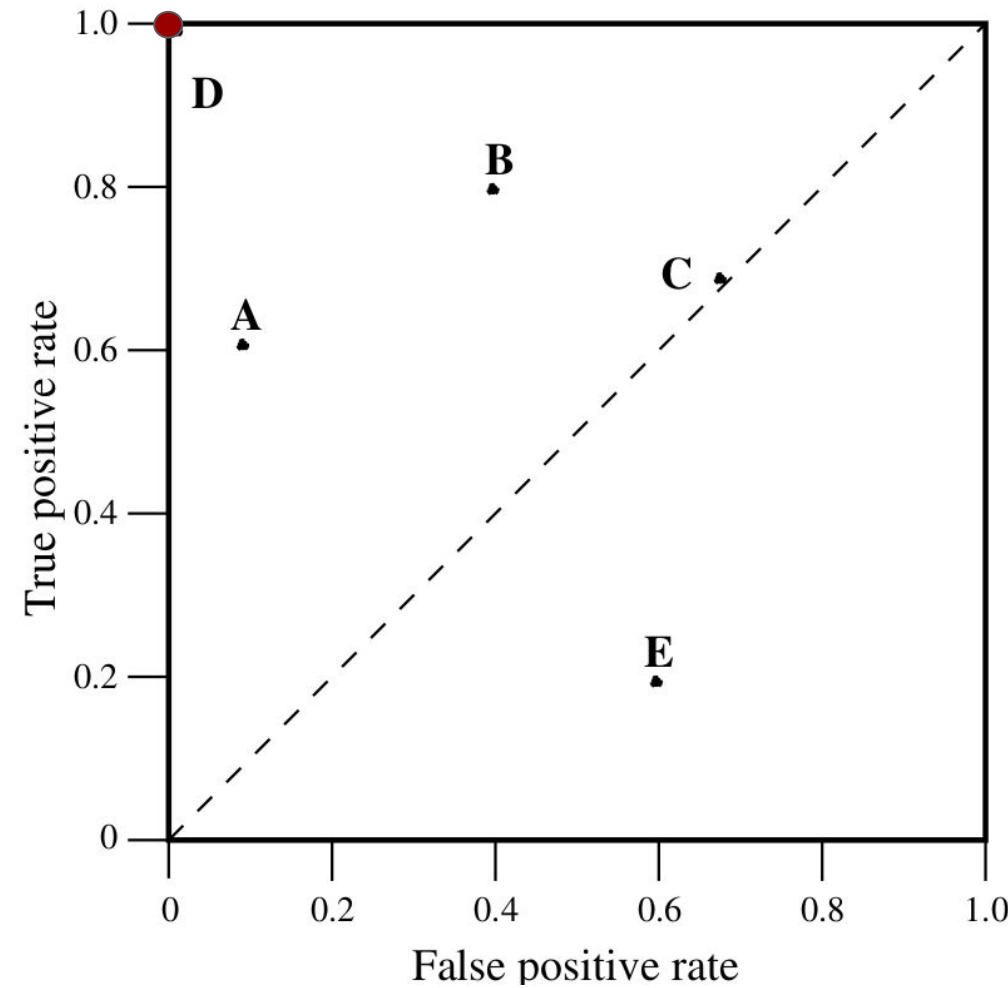
Conservative Vs. Liberal

- Classifiers appearing on the left hand-side of an ROC graph, near the X axis, may be thought of as “conservative”
 - they make positive classifications only with strong evidence so they make few false positive errors,
 - but they often have low true positive rates as well.



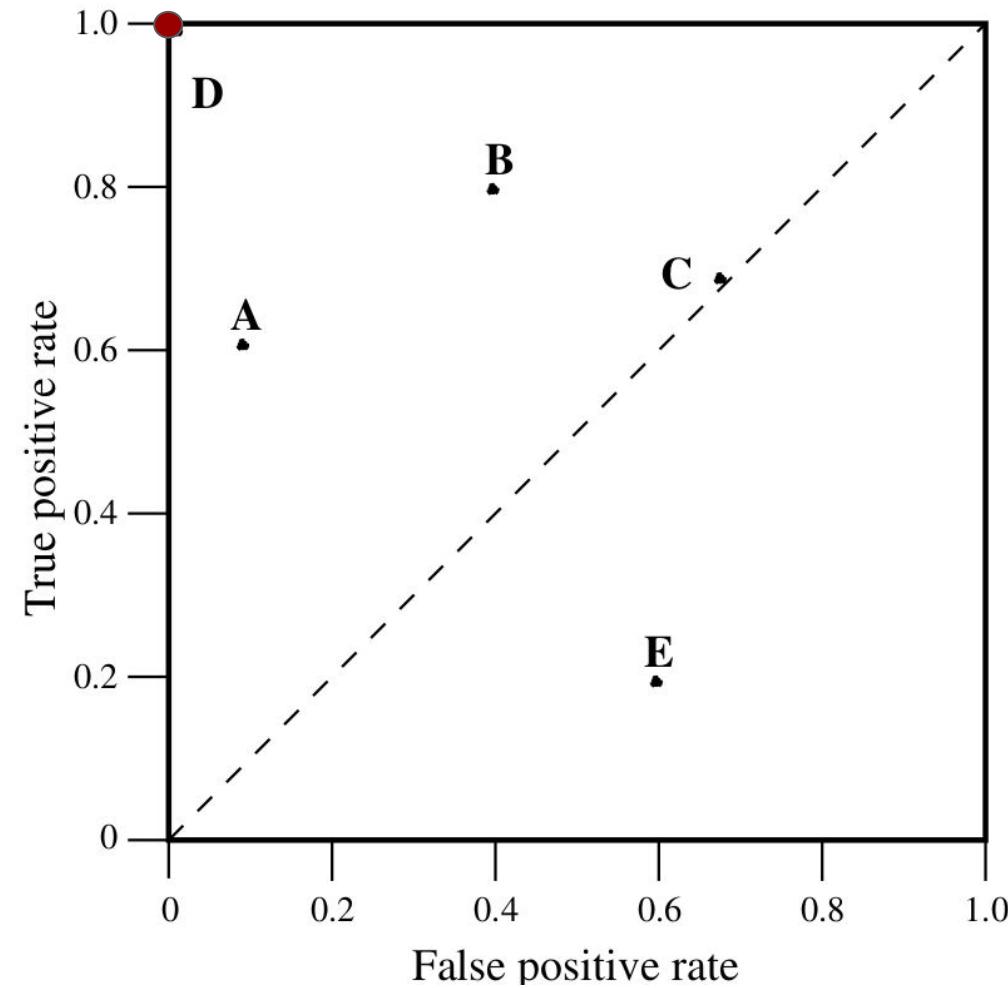
Conservative Vs. Liberal

- Classifiers on the upper right-hand side of an ROC graph may be thought of as “liberal”
 - they make positive classifications with weak evidence so they classify nearly all positives correctly,
 - but they often have high false positive rates.
- In figure, A is more conservative than B.



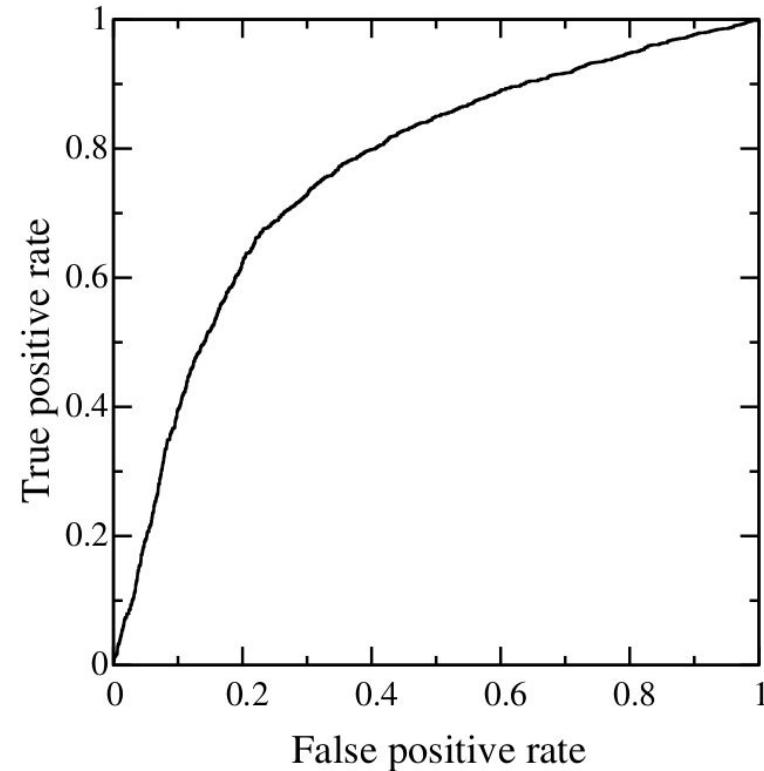
Random Performance

- Classifiers on the upper right-hand side of an ROC graph may be thought of as “liberal”
 - they make positive classifications with weak evidence so they classify nearly all positives correctly,
 - but they often have high false positive rates.
- In figure, A is more conservative than B.



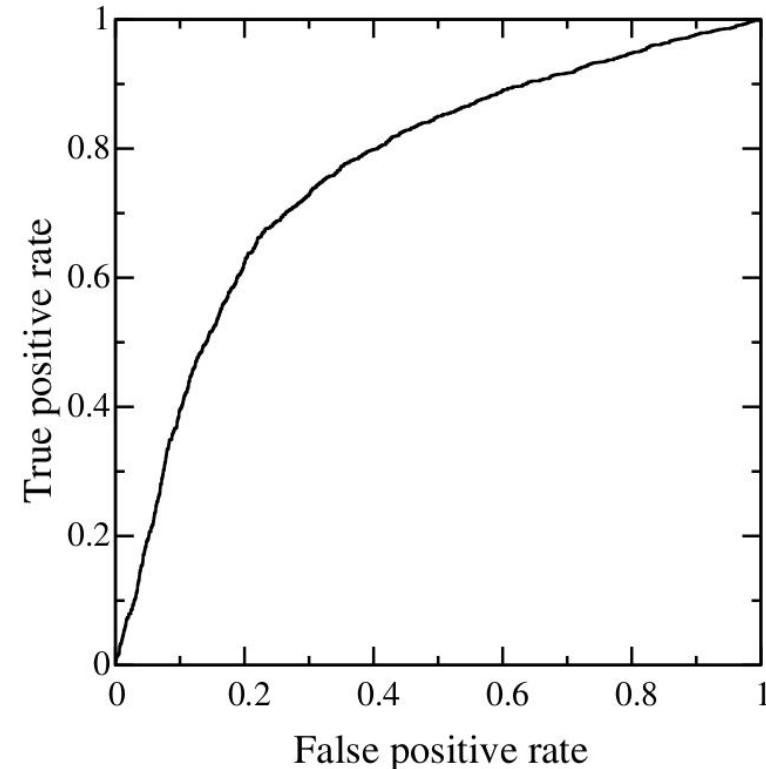
Curves in ROC space

- Classifiers, such as decision trees or rule sets, are designed to produce only a class decision, i.e., a Y or N on each instance.
 - When such a **discrete classifier** is applied to a test set, **it yields a single confusion matrix**, which in turn corresponds to one ROC point.
 - Thus, a discrete classifier produces only a single point in ROC space.



Curves in ROC space

- Classifiers, such as a Naive Bayes classifier, yield an instance probability or score.
 - Such a ranking or scoring classifier can be used with a threshold to produce a discrete (binary) classifier:
 - if the classifier output is above the threshold, the classifier produces a Y,
 - else a N.
- Each threshold value produces a different point in ROC space (corresponding to a different confusion matrix).
- Conceptually, we may imagine varying a threshold from –infinity to + infinity and tracing a curve through ROC space.



Algorithm

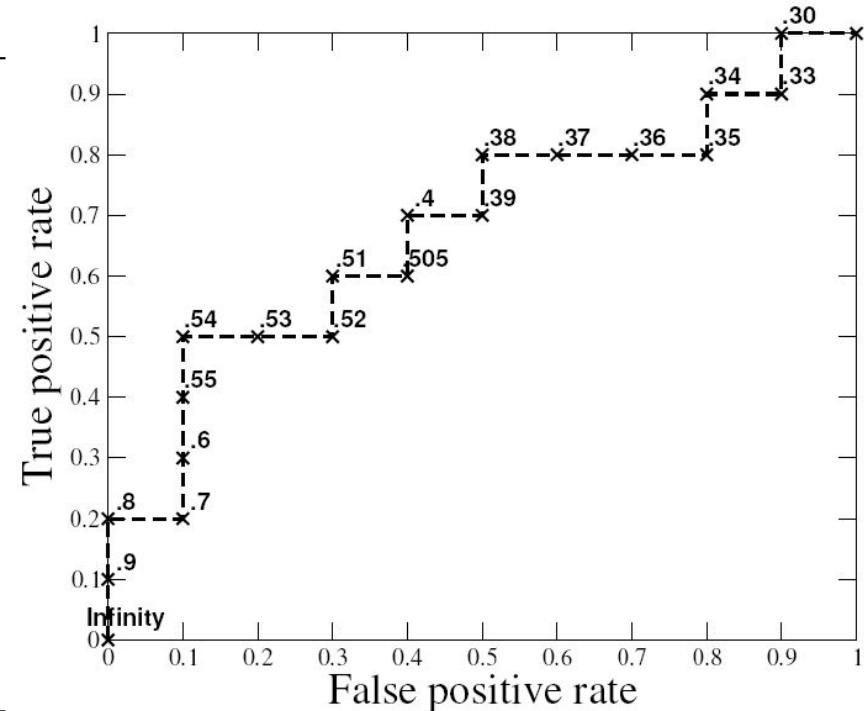
- Exploit monotonicity of thresholded classifications:
 - Any instance that is classified positive with respect to a given threshold will be classified positive for all lower thresholds as well.
- Therefore, we can simply:
 - sort the test instances decreasing by their scores and
 - move down the list, processing one instance at a time and
 - update TP and FP as we go.
- In this way, an ROC graph can be created from a linear scan

Example

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

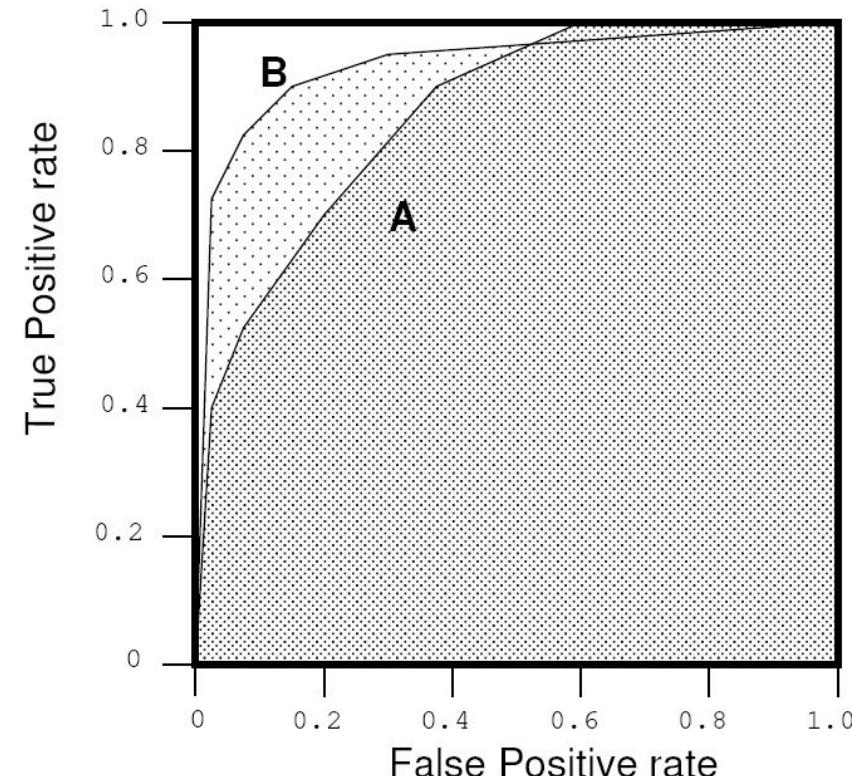
Example

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1



Area under ROC

- AUC has an important statistical property:
 - The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.
- Often used to compare classifiers:
- The bigger AUC the better
- AUC can be computed by a slight modification to the algorithm for constructing ROC curves.





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you