

Basics of Computer Vision

By: Dr. Muhammad Fahim

Contact Information

- Instructor:
 - Muhammad Fahim
 - Telegram: @Fahim17
- Teaching Assistants
 - Batul Yaseen
 - Karam Almaghout
- Course materials will be sent to you through moodle

Grading Criteria

- Group Project (3 students per group) 20%
 - Homework 20%
 - Mid-term Exam + Lab Exam 20% + 10%
 - Final Exam + Lab Exam 20% + 10%
-
- A. 90-100
 - B. 75-89
 - C. 60-74
 - D. 0-59 (*Fail*)

Attendance is mandatory in the course

Goals of this Course

- This course is designed for undergraduate students to provide comprehensive introduction and practical knowledge of computer vision.
- Student will learn to implement different computer vision methods in Python programming environment.
- The end of the day they will be able to apply computer vision technique to solve real-world problems.

Prerequisites Courses

- Prerequisites—*these are essential!*
 - Data structures
 - Linear algebra
 - Vector calculus
 - A good working knowledge of Python programming
- Course does ***not*** assume prior imaging experience
 - Image processing, graphics, etc.

Learning Outcome

- After this course, you will be able to:
 - Understand how machine can visualize the objects and understand images/videos
 - Significant exposure to real-world implementations
 - To develop research interest in the theory and application of computer vision

Materials

- **Textbook**
 - *Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, Wiley, 2014.*
- **Reference Materials**
 - *Szeliski, R. Computer Vision: Algorithms and Applications, Springer, 2010.*
 - *Trucco, E. and Verri, A. Introductory Techniques for 3-D Computer Vision, Prentice-Hall, 1998.*
 - *Vernon, D. Machine Vision: Automated Visual Inspection and Robot Vision, Prentice-Hall, 1991*
- **My slides and shared materials through moodle**

Tentative Plan

- Week 1: Introduction to Computer Vision, Image Acquisition, Basic Image Processing
- Week 2: Image Filtering, Noise, Convolutions, Kernels, Smoothing and Blurring
- Week 3: Thresholding, Histograms, Image Gradients, Edge Detection and Morphological operations
- Week 4: Object Detection, Template Matching, Image Pyramids, HOG with SVM and Contours (**Homework Out**)
- Week 5: Image Features: Interest Points, Descriptors and Matching
- Week 6: Bag of Visual Words, Image Classification (**Homework Submission**)
- Week 7: Midterm Exam
- Week 8: Deep Learning for Object Detection and Localization (YOLO) (**Group Project Out**)
- Week 9: Generative Adversarial Networks (GAN) for Computer vision
- Week 10: Face Detection, Viola Jones, Haar-like Features, Integral Image, Adaboost, Classifier Cascade
- Week 11: Hough Transform
- Week 12: Semantic Segmentation
- Week 13: Video Processing and Tracking (**Group Project Submission**)
- Week 14: Project Presentations
- Week 15:Final Exam

Ready to go😊

Contents

- Introduction to computer vision
- Computer vision in action
- The human vision system
- Image sensing and acquisition
- Sampling and quantization
- Fundamentals of color images
- Converting images from one color space to another space
- Summary

Source of the material

- This lecture is based on the following resources
 - Lecture slides of *Prof. Adil Khan, Innopolis University.*
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe, lecture slides of David Vernon and his book.
 - Computer vision: Foundation and applications by Ranjhay Krishna.
 - Found material over the internet to aligned the subject according to the need of the students.

Introduction

- Computer vision is inspired by the capabilities of the human vision system and initially addressed in the 1960s and 1970s
- Definition of Computer Vision
 - Computer vision is the automatic analysis of images or videos by computers in order to gain understanding of the world

Introduction – Every image tells a story



Goal of computer vision: perceive the “story” behind the picture

Introduction

- <https://www.youtube.com/watch?v=NrmMk1Myrxc>

Introduction

- Computer Vision is about understanding images. These images can be
 - Black and white, greyscale, colour or **multi-spectral**
 - Snapshots or video sequences
 - Taken with a static or moving camera
 - Taken of a stationary or dynamic scene
 - Taken with a **calibrated** or **un-calibrated camera**
- What computer vision aims to do is to extract some **useful information** from these images for...
 - Inspection purposes
 - Analysis purposes
 - Control purposes

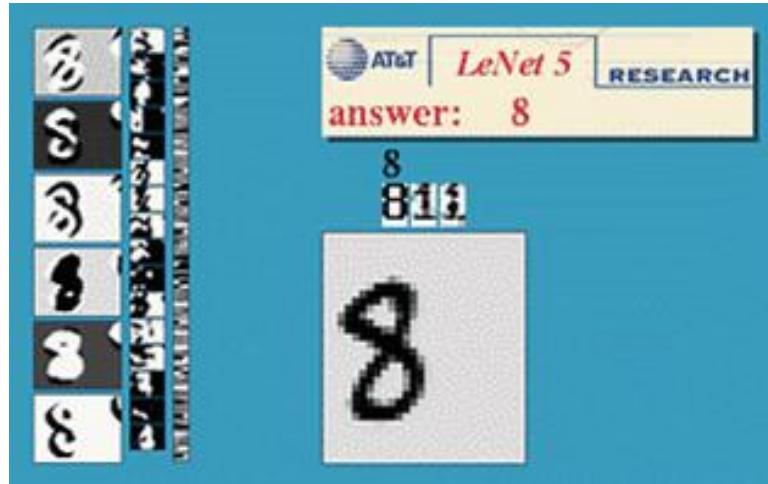
Why study computer vision?

- Billions of images/videos captured per day



- Huge number of useful applications
- The next couple of slides show the computer vision in action

Computer Vision in Action!!



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

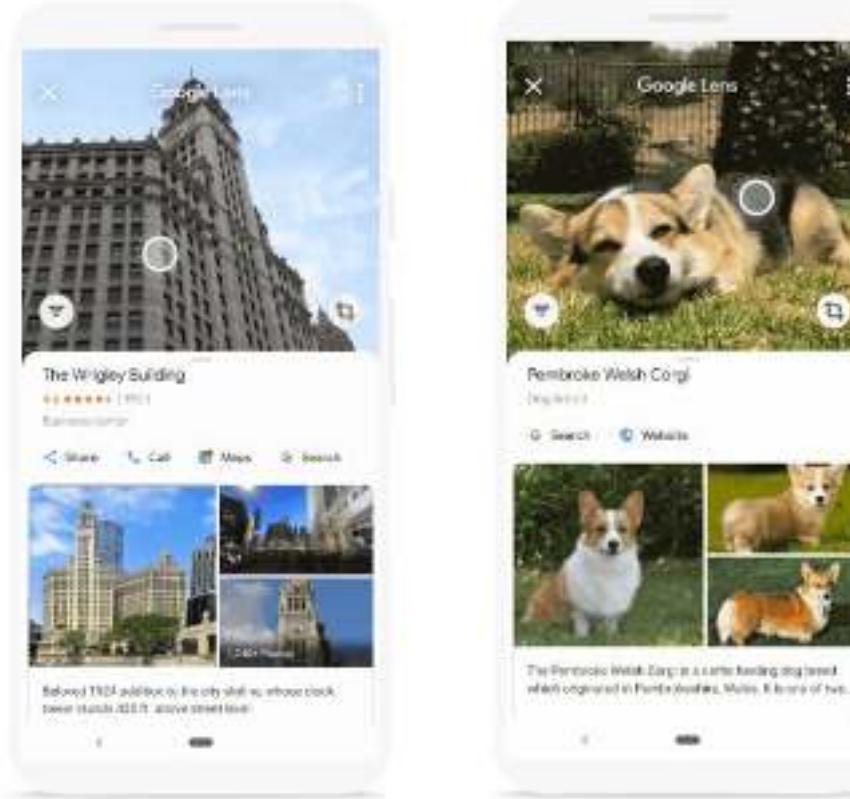
<http://yann.lecun.com/exdb/lenet/>



Computer Vision in Action!!



Google Lens
Search what you see



<https://lens.google.com/>

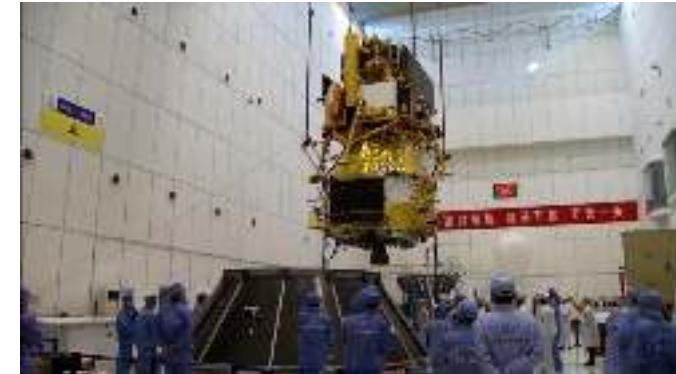
Computer Vision in Action!!

- Special Effects: Motion Capture



Computer Vision in Action!!

It is a robotic spacecraft mission of Chinese Lunar Exploration Program



Change 5: It
launched on 23
November 2020

The first panorama from the far side of the Moon



Computer Vision in Action!!



NASA's Mars Curiosity Rover



Amazon Prime Air

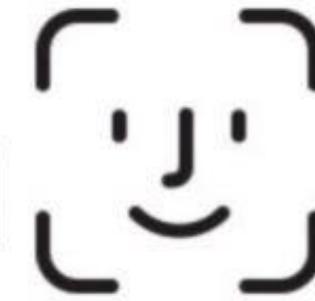
Computer Vision in Action!!



Fingerprint scanners on many new smartphones and other devices



Login without a password

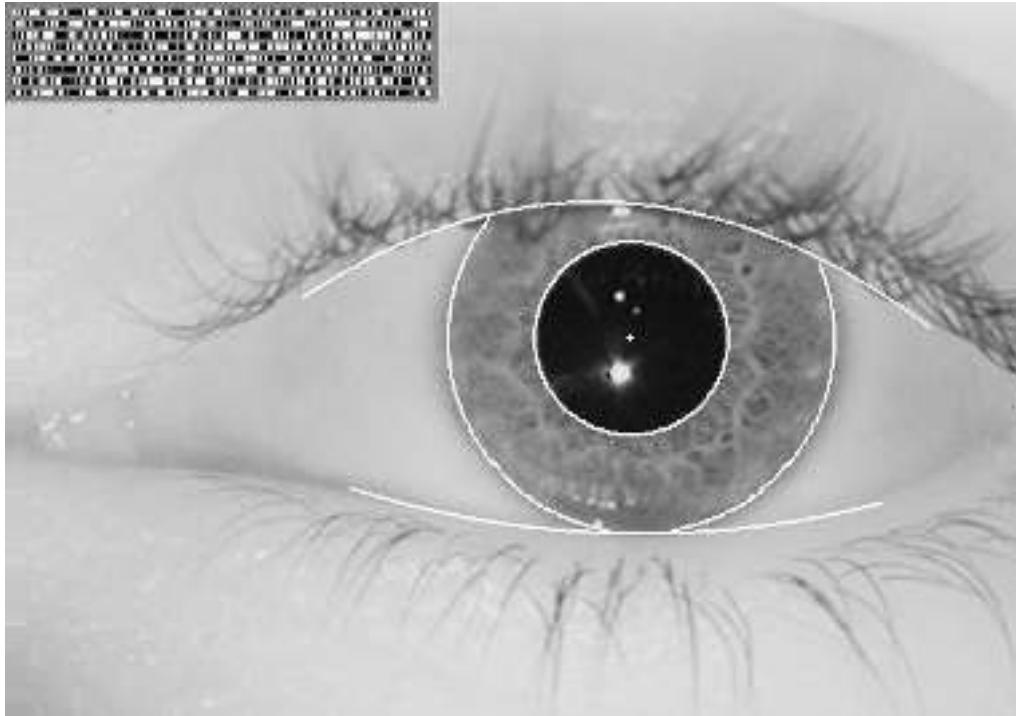


Face ID



Computer Vision in Action!!

Iris Recognition



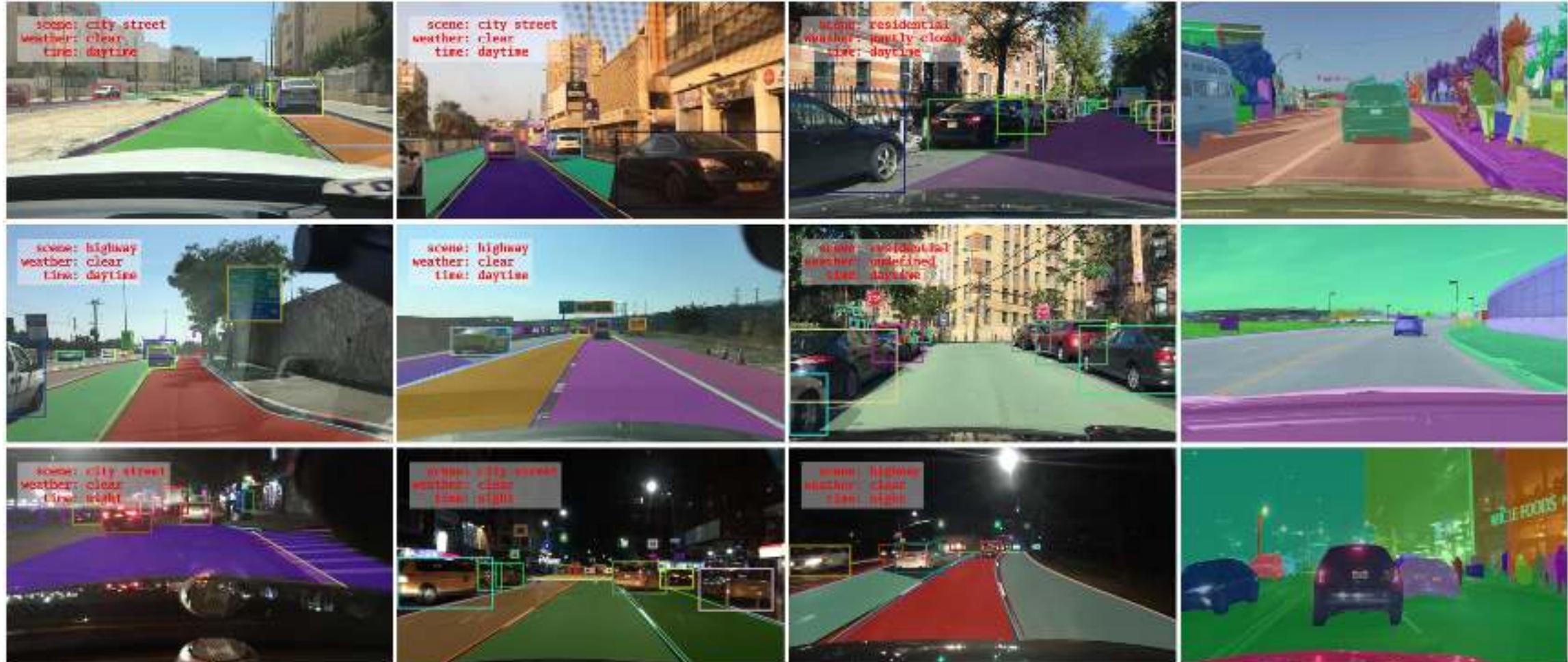
[John Daugman: How Iris Recognition Works, 2004]
Genetically identical eyes also have different iris.



Access control,
e.g., border

Computer Vision in Action!!

Berkeley Deep Drive (Driverless cars)



<https://bdd-data.berkeley.edu/wad-2018.html>

Computer Vision in Action!!

Image Super Resolution



Source: [Goodfellow, 2016]

<https://arxiv.org/abs/1701.00160>

Computer Vision in Action!!

Image Inpainting



Ugur Demir and Gozde Unal. Patch-Based Image Inpainting with Generative Adversarial Networks. *arXiv preprint*, 2018. URL <http://arxiv.org/abs/1803.07422>.

Computer Vision in Action!!

Create Art



Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms. *arXiv preprint*, (Iccv):1–22, 2017. doi: 10.1089/cyber.2017.29084.csi. URL <http://arxiv.org/abs/1706.07068>.

Computer Vision in Action!!

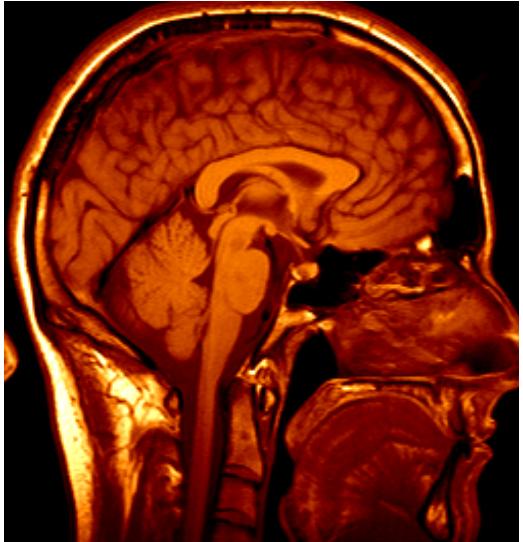
Sports



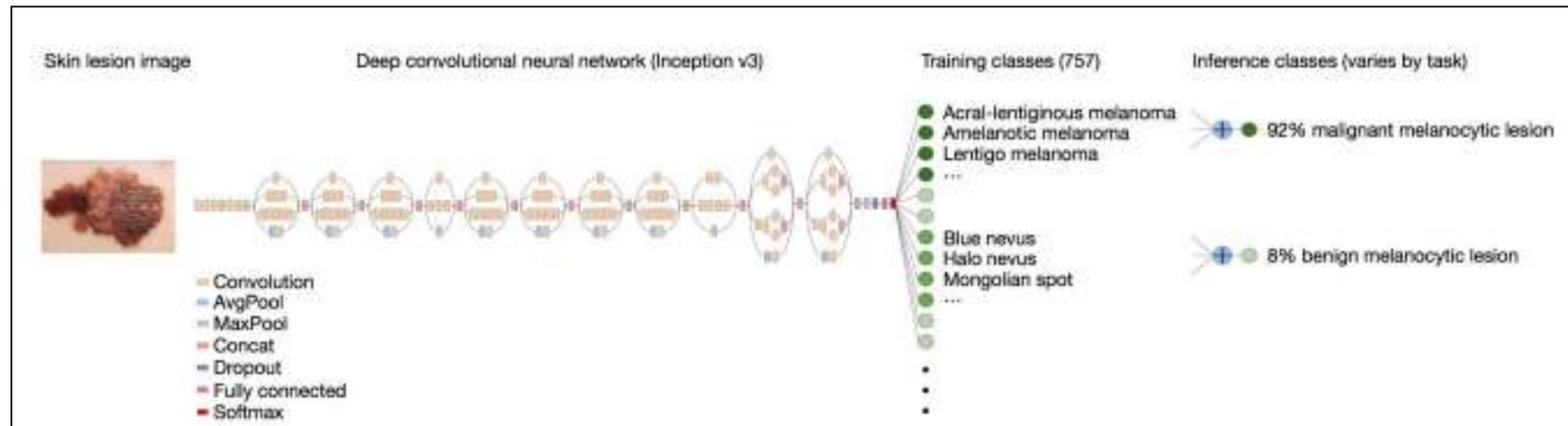
Check the video: <https://www.smt.com/>

Computer Vision in Action!!

Medical Imaging



3D imaging (MRI)



Skin cancer classification with deep learning (Published: 25 January 2017)
<https://cs.stanford.edu/people/esteva/nature/>

Computer Vision – A difficult Problem



Motion



Viewpoint variation



Intra-class variation



Illumination



Background clutter



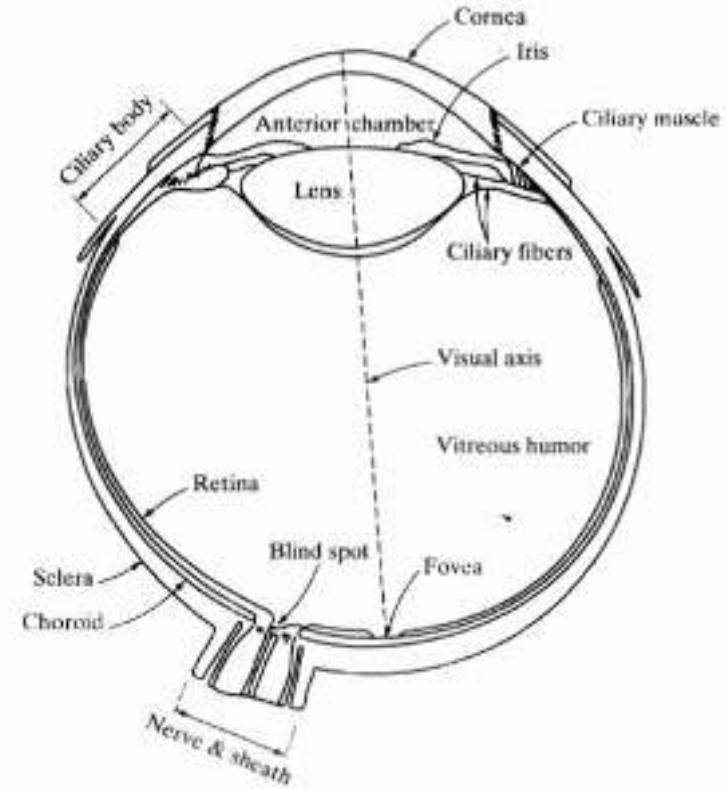
Occlusion

The Human Vision System

- If we could **duplicate the human visual system**, then the **problem of developing a computer vision system** would be **solved**.
 - **So why can't we?**
 - The main difficulty is that we **do not understand what the human vision system is doing most of the time.**

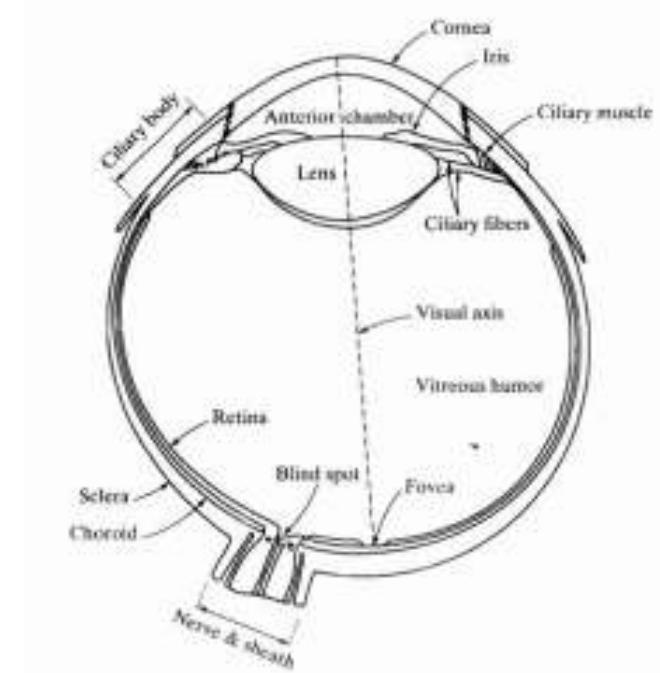
Structure of Human Eye

- Eye is nearly a sphere, with an **average diameter** of approximately **20 mm**.
- Three membranes enclose the eye:
 - **Cornea** (Tough and Transparent tissue) and **sclera** outer cover
 - **Choroid** (i.e., Membrane contains a network of blood vessels and major source of nutrition)
 - **Retina**.

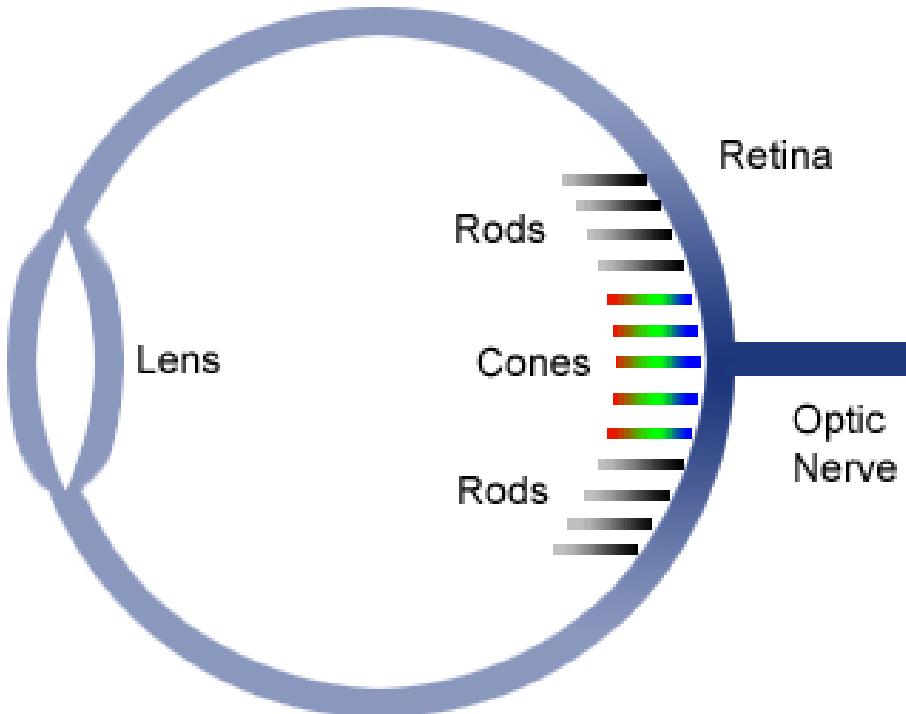


Structure of Human Eye

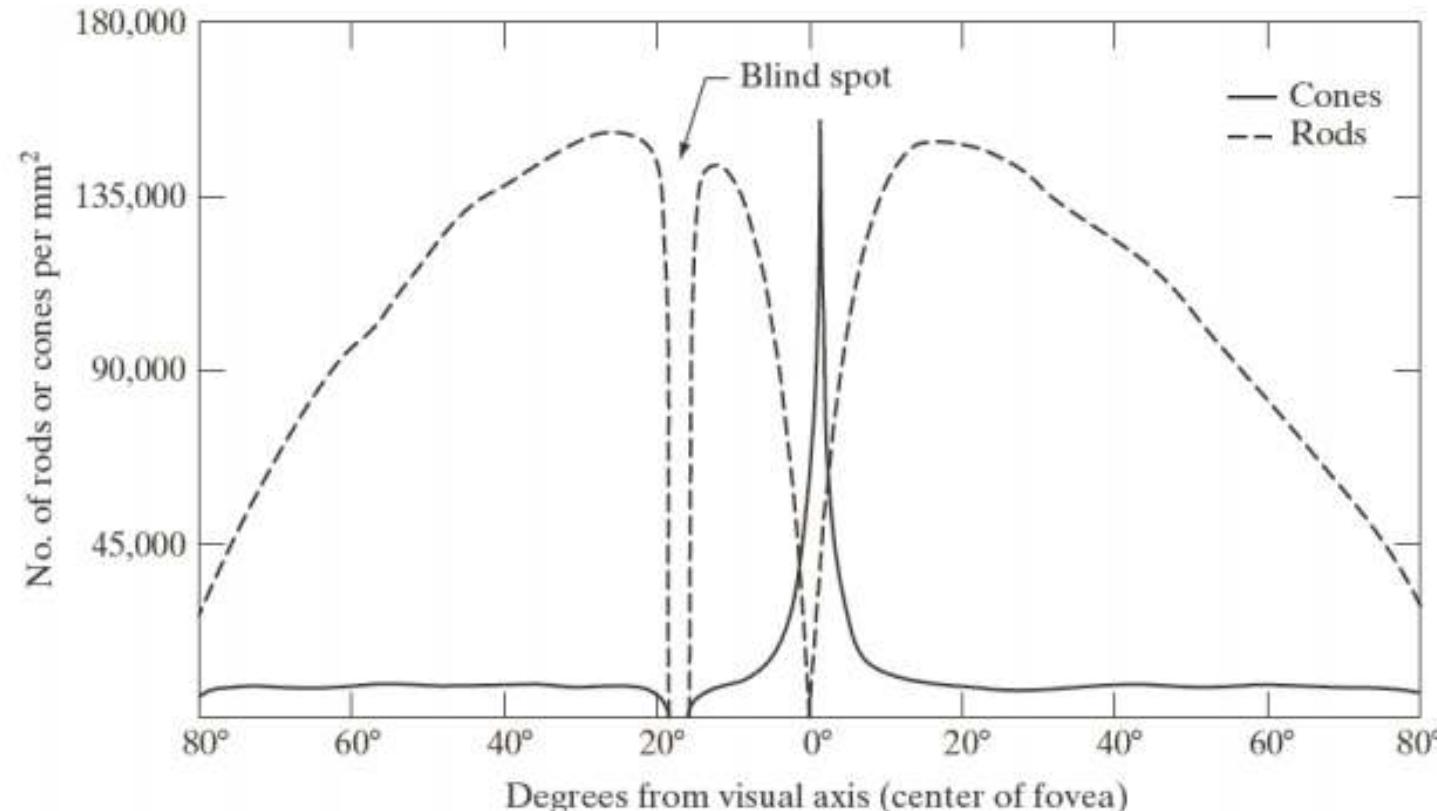
- When the eye is properly focused, light from an object outside the eye is **imaged on the retina**
- Two classes
 - **Cones**
 - 6-7 million located primarily near the center of the retina (the fovea)
 - highly **sensitive to color**
 - can **resolve fine details** because each is attached to a **single nerve ending**
 - Cone vision is called photopic or **bright-light vision**
 - **Rods**
 - 75-150 million distributed over the retinal surface
 - multiple rods connected to a **single nerve ending**
 - give a general overall picture of the **field of illumination**
 - **not color sensitive** but are sensitive to low levels of illumination
 - Rod vision is called scotopic or **dim-light vision**
- Around the region of the emergence of the optic nerve, there is **no receptors** and results in the so-called **blind spot**



Cones and Rods



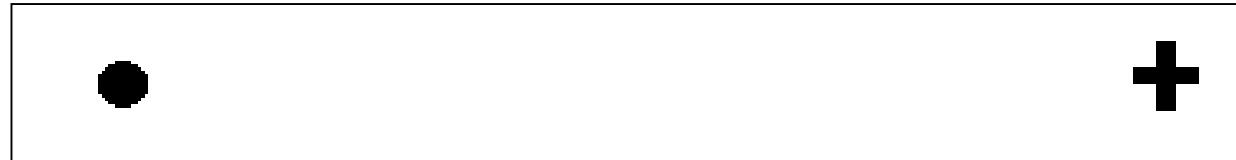
Structure of Human Eye



Distribution of rods and cones in the retina

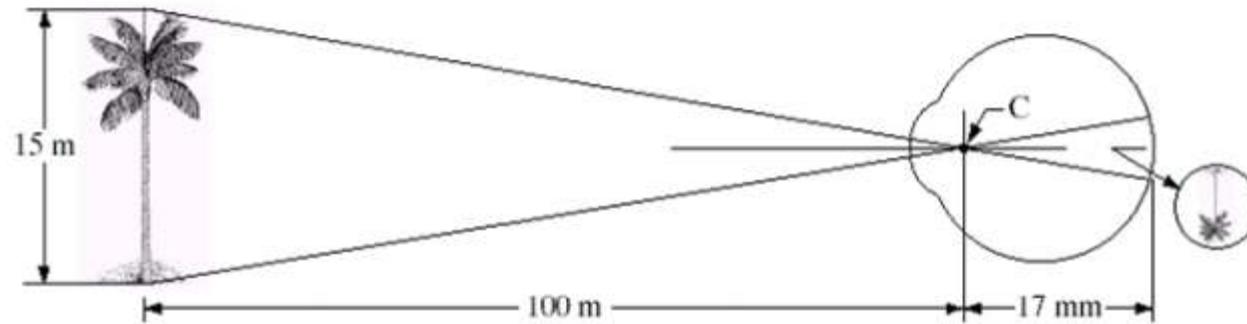
Blind-Spot Experiment

- Draw an image similar to that below on a piece of paper (the dot and cross are about 6 inches apart)



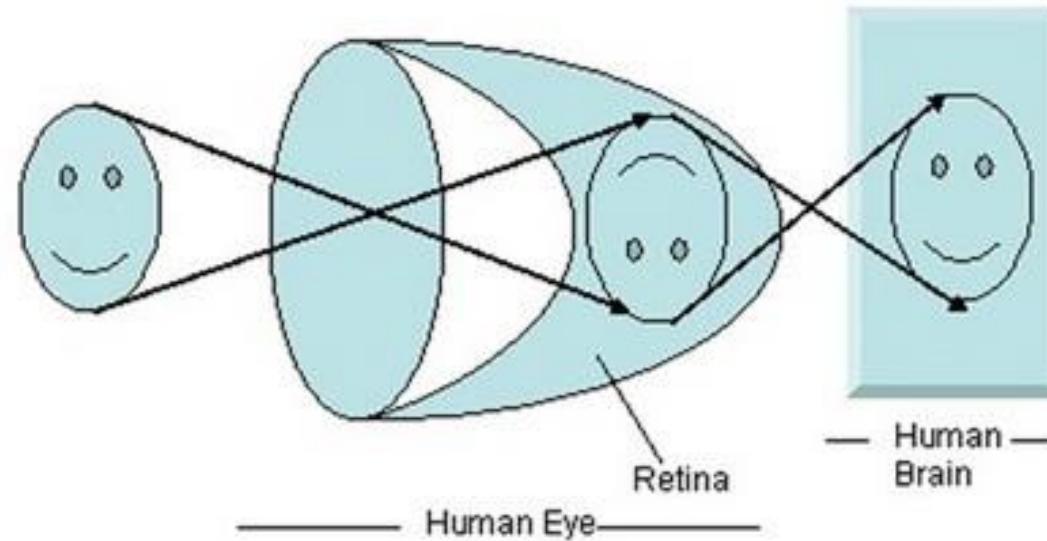
- Close your right eye and focus on the cross with your left eye
- Hold the image about 20 inches away from your face and move it slowly towards you
- The dot should disappear!

Image Formation in the Eye

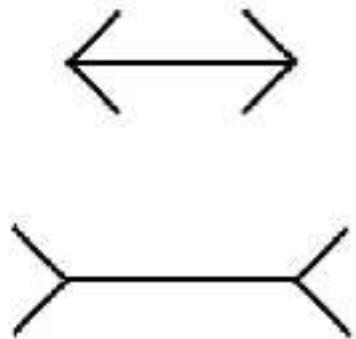


- Suppose a person is looking at a palm tree 15 m high at a distance of 100m.
 - $15/100 = h/17$ or approximately 2.55 mm

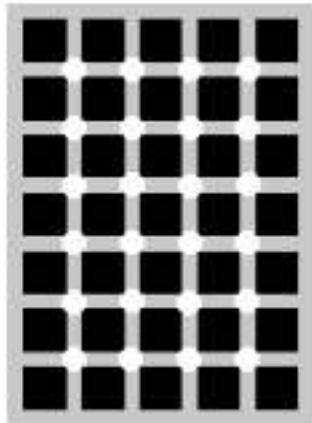
Image Formation



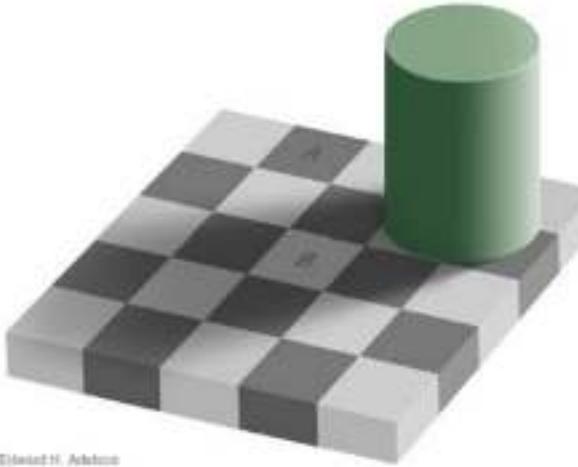
Optical Illusions



(a)



(c)



© David H. Al�am

(b)

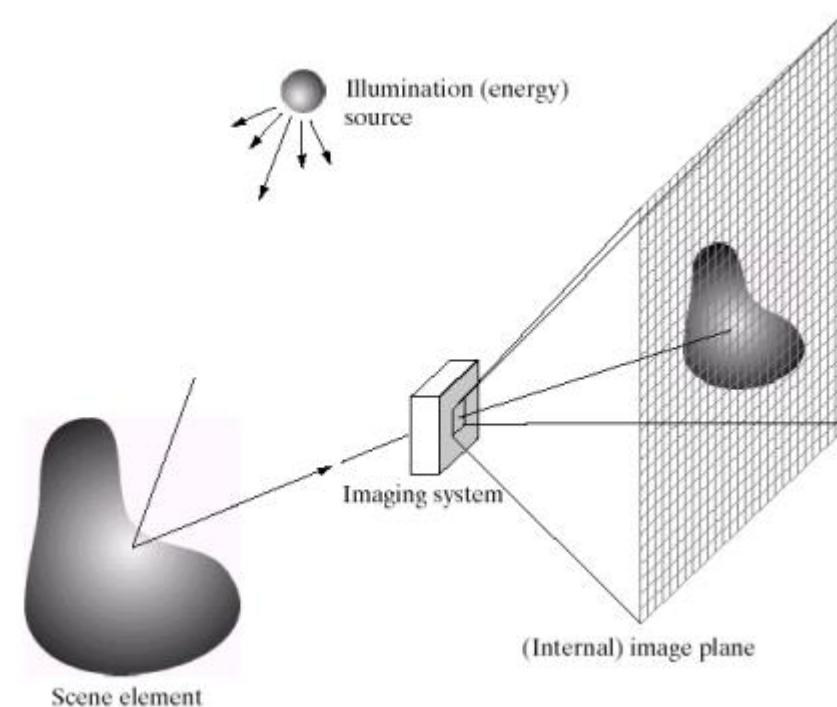
X X X X X X X X	O X O X O X X
X X X X X X X X	X O X X X O X
X X X X X X X X	O X X O X X O
X X X X X X X X	X X O X O O X
X X X X X X X X	O X X O X X X
X X X X X X X X	X O X X X O X
X X X X X X X X	O X X O X X O
X X X X X X X X	X O X X X O X
X X X X X X X X	X X X O O X X
X X X X X X X X	X O X X X X O X

(d)

Image Sensing and Acquisition

- Images are generated by the combination of
 - an “illumination” source and
 - the reflection or absorption of energy from that source

by the elements of the “scene” being imaged.

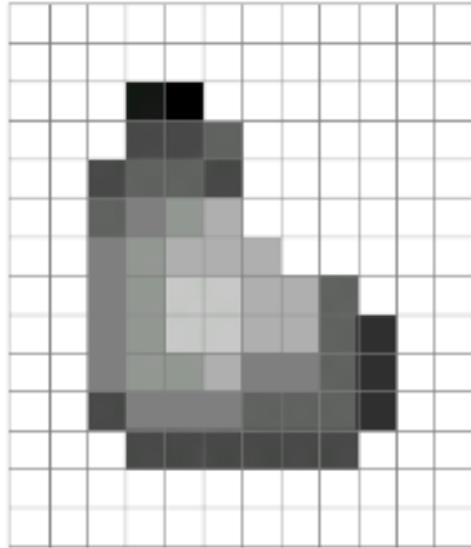


An example of the digital image acquisition process

Digital images represent the reflectance function of a scene, but they do so in a sampled and quantized form

What is an Image?

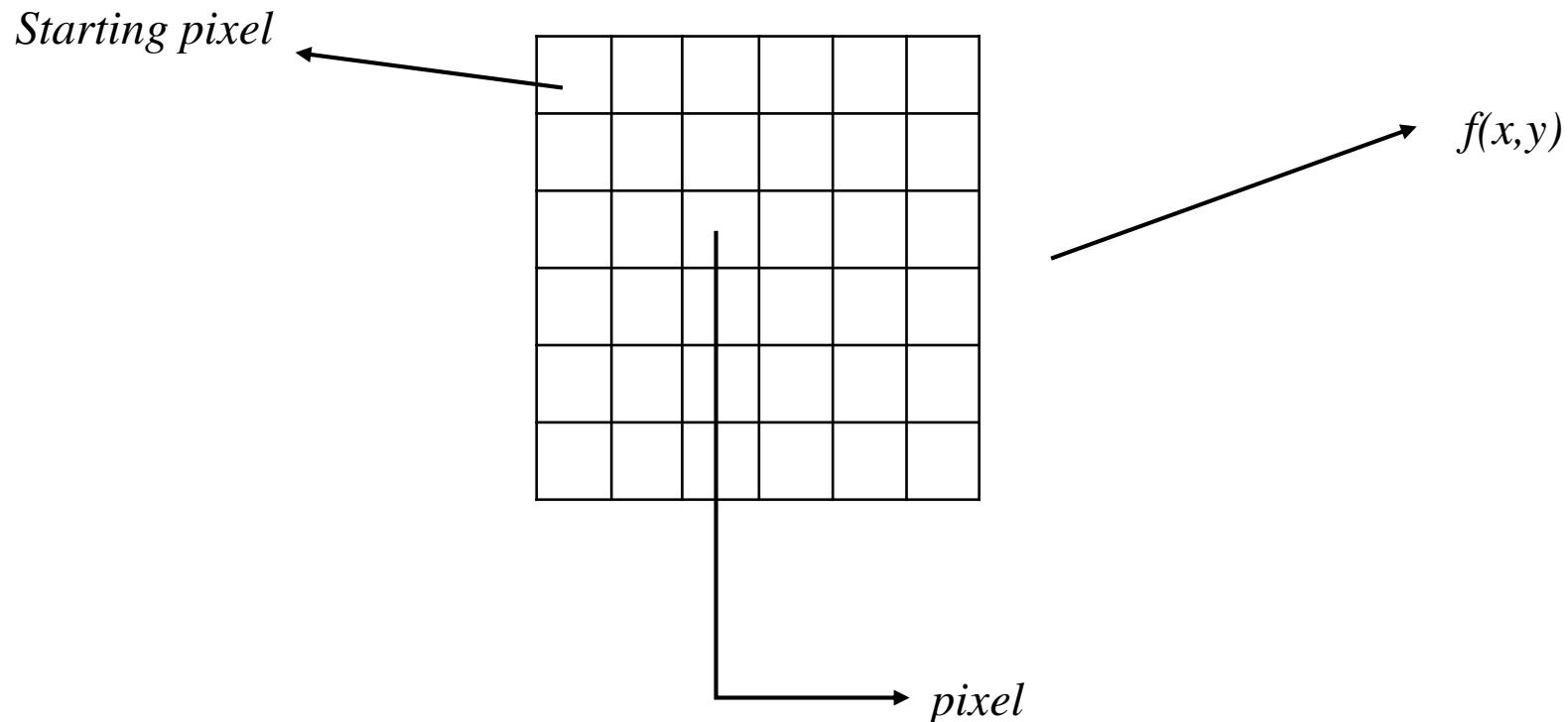
A grid (matrix) of intensity values



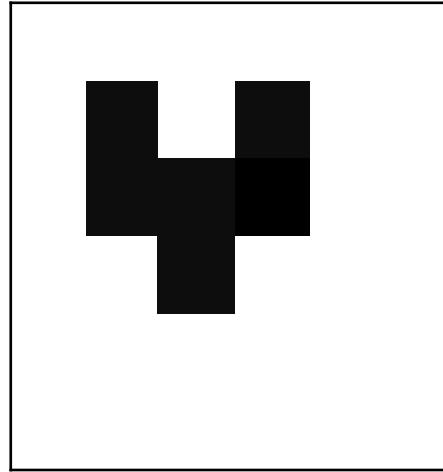
=

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

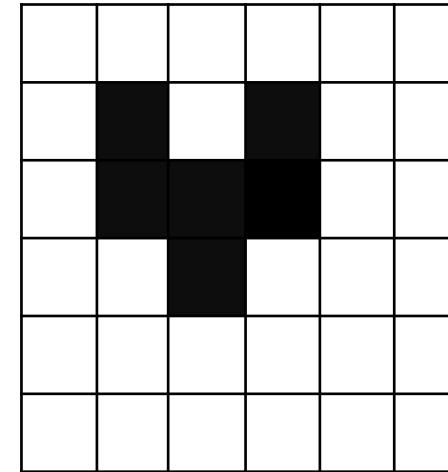
Processing the Images



Processing the Images



Image



$f(x, y)$

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

*In Computer
(Digitized Image)*

Digitizing an Image – Sampling



Four different samplings of the same image; top left 256x192, top right 128x96, bottom left 64x48 and bottom right 32x24

Digitizing an Image – Quantization

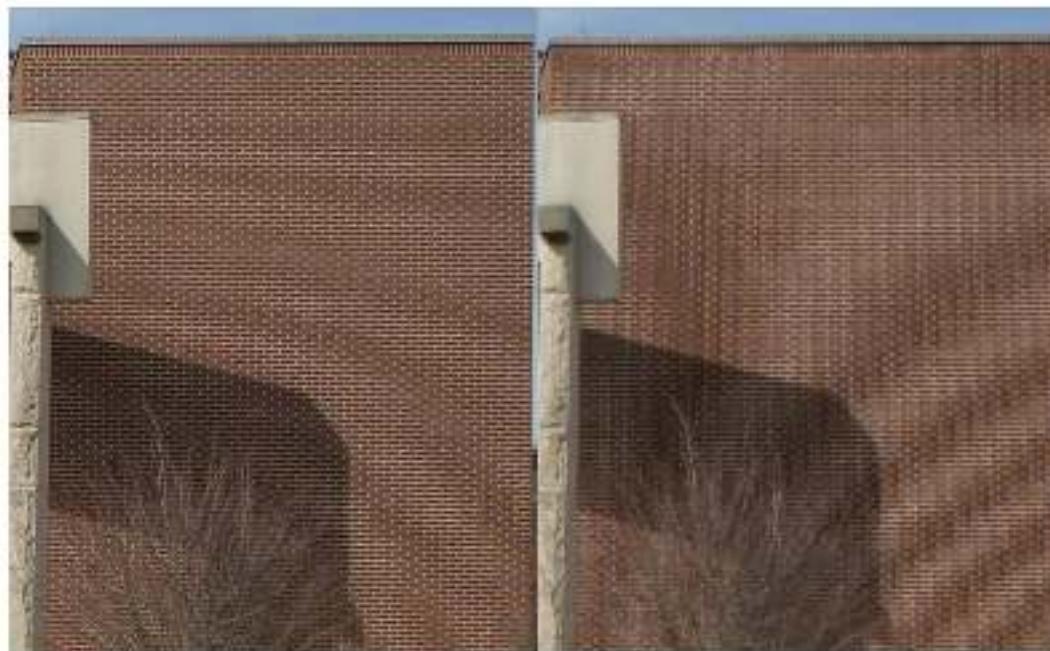
- Each quantized integer value is known as a pixel
- It is the smallest discrete accessible sub-section of a digital image.



Four different quantization's of the same grey-scale image; top left **8 bits**, top right **6 bits**, bottom left **4 bits** and bottom right **2 bits**

Sampling Rate – Issue

- If the sample rate is not sufficient, the sampled image will be **aliased**



This is also known as
“moire pattern”

Sampling Rate – Solution

- **Nyquist Shannon's Sampling Theorem**
 - The **minimum sampling frequency** required to reconstruct a signal from its instantaneous samples must be at least twice the **highest frequency**

$$f_s \geq 2f_{\max}$$

Image Representation

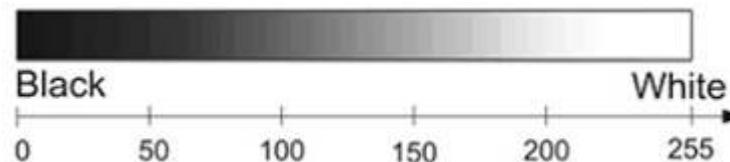
- Pixel **content** depends on the image type
 - Gray level pictorial digital images: **intensity**
 - Color pictorial digital images: color (modeled as **triplets**, e.g., RGB)
 - Range images: **depth information**
 - Medical images: **radiation absorbance level**
 - Thermal images: **heat**

Check Point!!

- Black and White image contains only two levels



- Gray image represent by black and white shades or combination of levels
 - How many colors in gray scale images?
 - How many bits are required for gray scale image?
 - **8-bit gray image** means total 2^8 levels form black to white. Where 0 = black and 255 is White



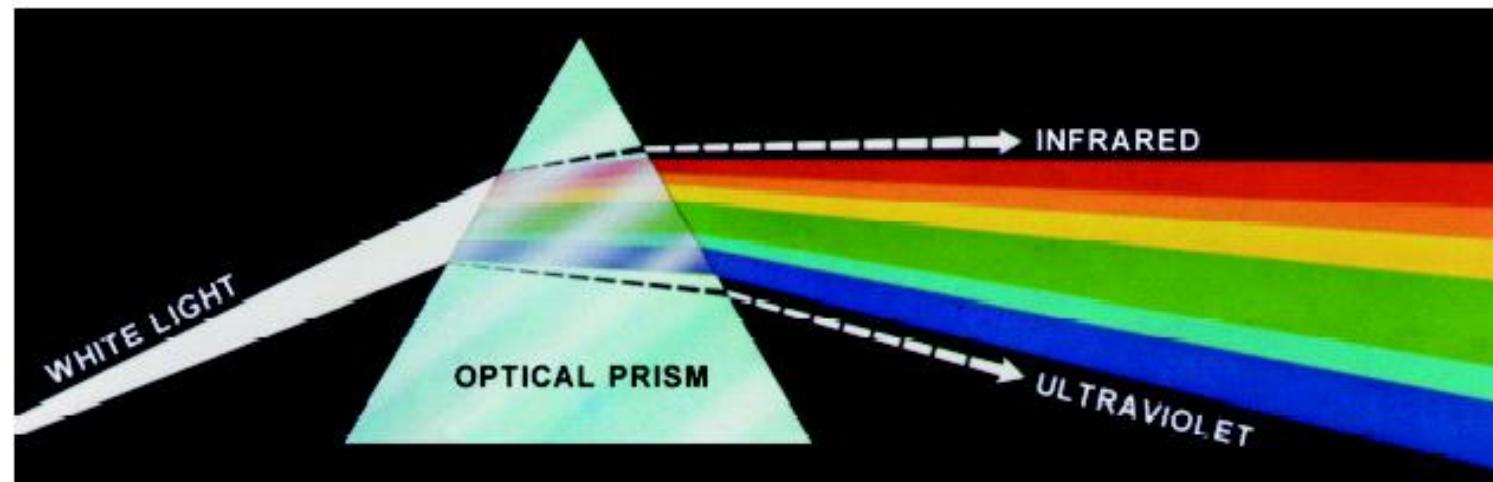
Color Images – Motivation

- Color is a **powerful descriptor** that often **simplifies object identification** and extraction from a scene.
- Human can **distinguish thousands of color** shades and intensities, **compared to about only two dozen shades of gray**.

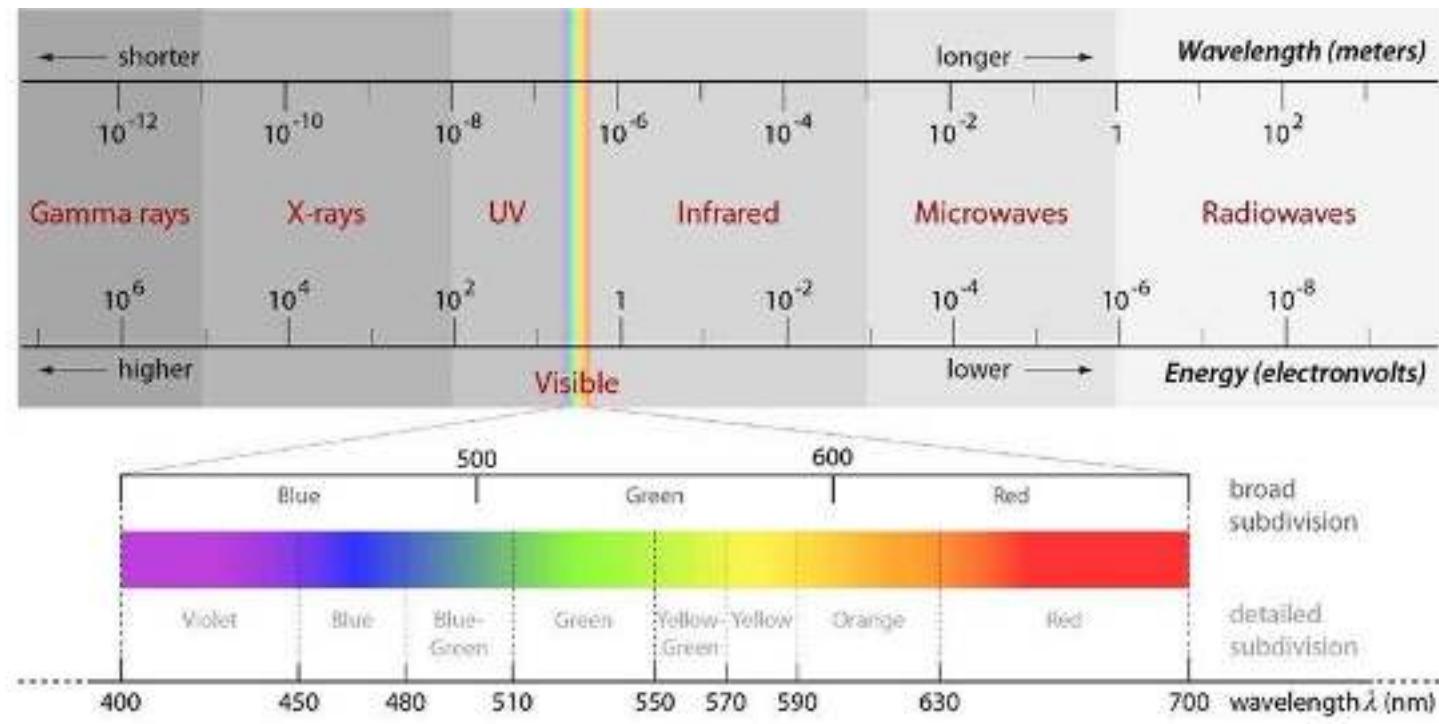


Colour Fundamentals

- In 1666 Sir Isaac Newton discovered that when a beam of sunlight passes through a glass prism, the emerging beam is split into a spectrum of colors



Electromagnetic Spectrum

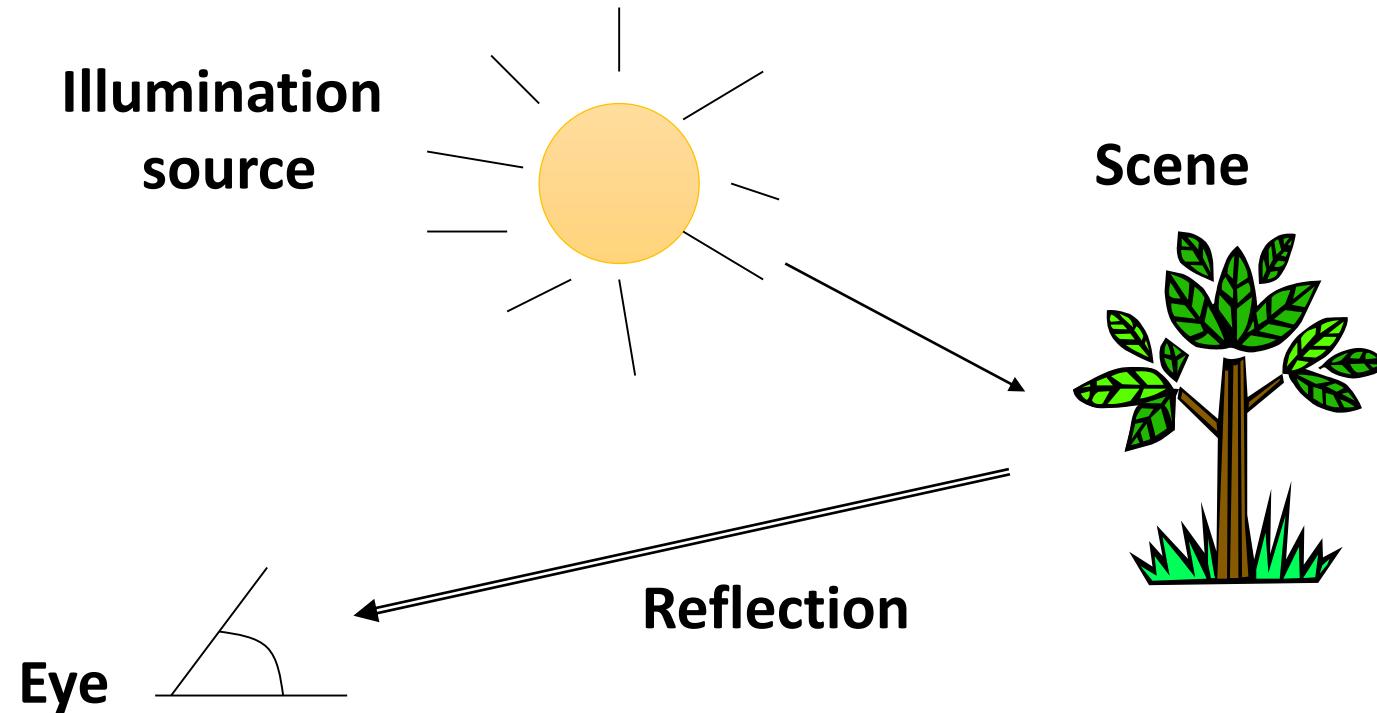


- **Visible light wavelength:** from around 400 to 700 nm

Image Source: https://www.researchgate.net/figure/The-complete-electromagnetic-spectrum-with-the-spectral-subdivisions-of-the-visible_fig2_320616988

Color Fundamentals

- The color that human perceive in an object = the light **reflected** from the object



Primary and Secondary Colors

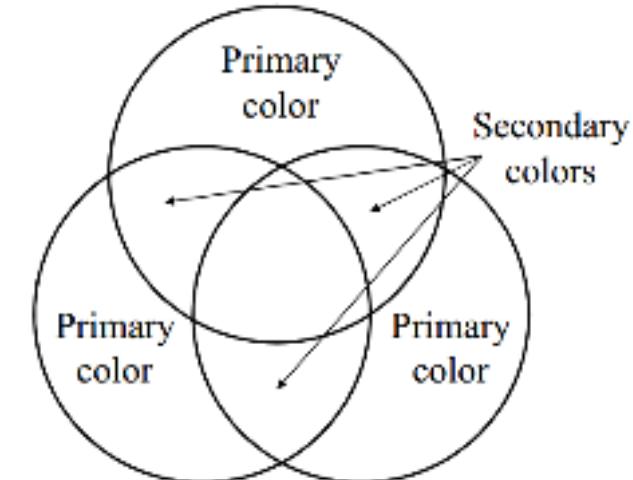
- **Primary colors:** In 1931, CIE (International Commission on Illumination) defines specific wavelength values to the primary colors

- Blue = 435.8 nm
- Green = 546.1 nm
- Red = 700 nm

16.8 million colors can be reproduced by mixing an appropriate set of three primary colors

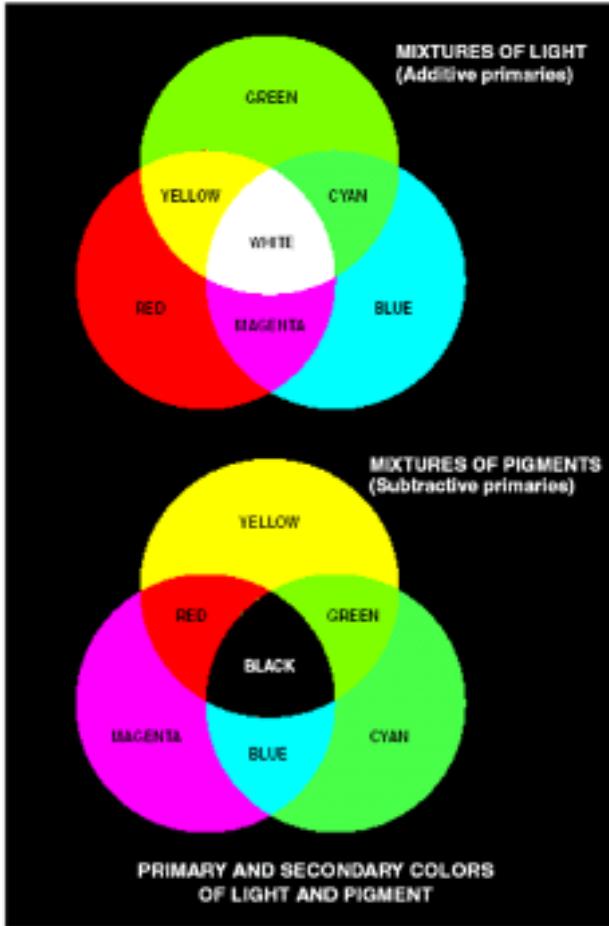
- **Secondary colors:**

- G+B=Cyan
- R+G=Yellow
- R+B=Magenta



*CIE = Commission Internationale de l'Eclairage

Primary and Secondary Colors



Additive Primary Colors: RGB use in the case of light sources such as [color monitors](#)

RGB add together to get [White](#)

Subtractive Primary Colors: CMY use in the case of pigments in printing devices

White subtracted by CMY to get [Black](#)

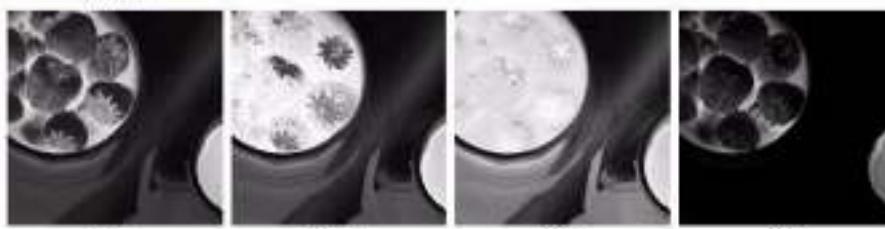
Colour Spaces

- There are many colour representations
 - RGB Red, Green, Blue -- monitor, video camera
 - CMY Cyan, Magenta, Yellow -- model for color printing
 - YUV Luminance (Y), Blue minus Luminance (U), Red minus Luminance (V)
 - YCrCb Scaled version of YUV
 - CIE XYZ Standard reference colour space based on the response of human eye
 - CIE L^{*}u^{*}V^{*} Perceptually uniform colour space
 - CIE L^{*}a^{*}b^{*} Device independent colour space (all colours perceived by humans)
 - HSV Hue, Saturation, Value
 - HLS Hue, Luminance, Saturation
 - HSI Hue, Saturation, Intensity. -- close to human interpret color

Example: Full-Color Image and Various Color Space Components



Color image



CMYK components



RGB components

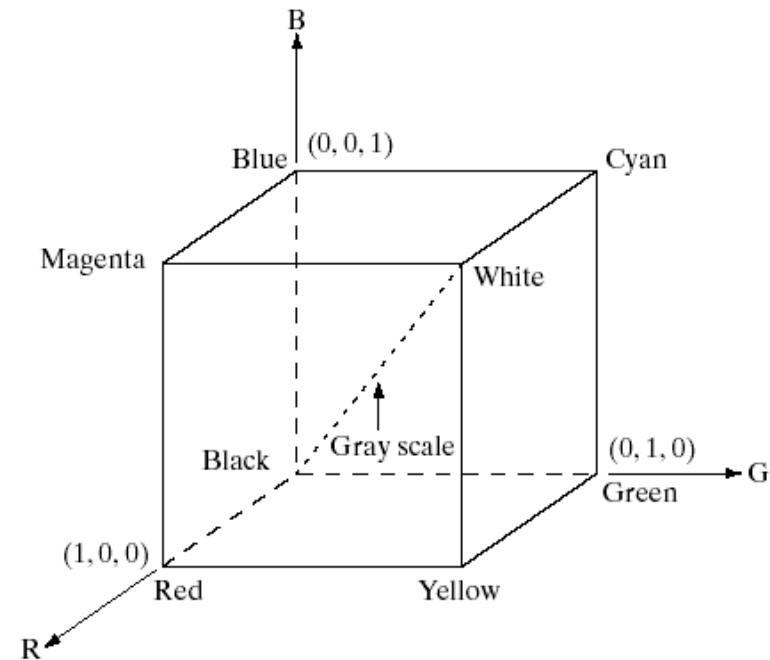
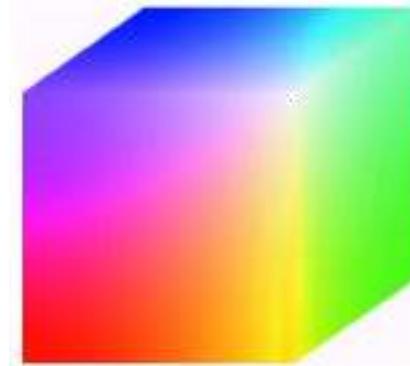


HSI components

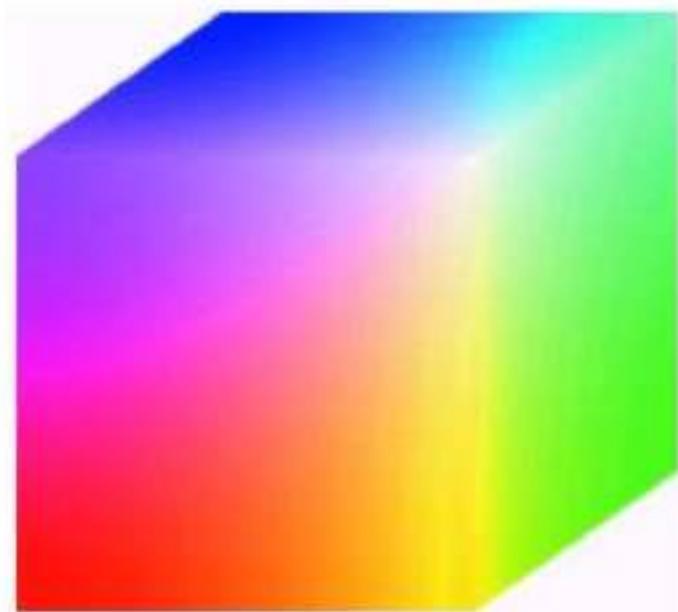
(Images from Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, 2nd Edition.)

RGB

- In the RGB model each color appears in its **primary spectral components of red, green and blue**
- **Purpose of color models:** To facilitate the specification of colors in some standard

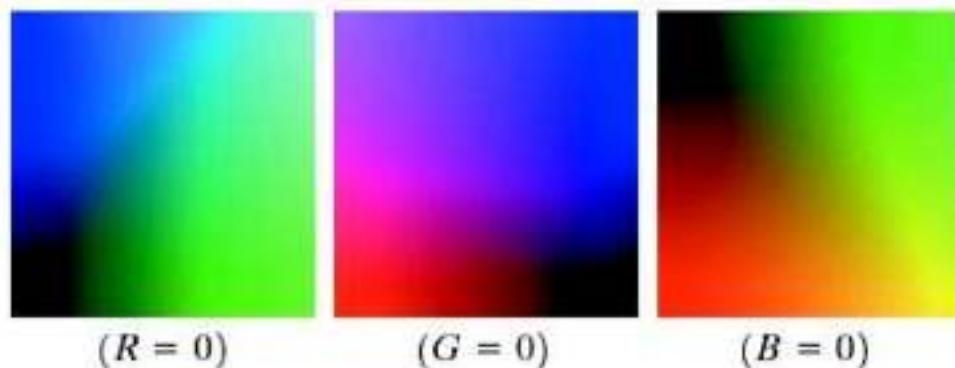


RGB



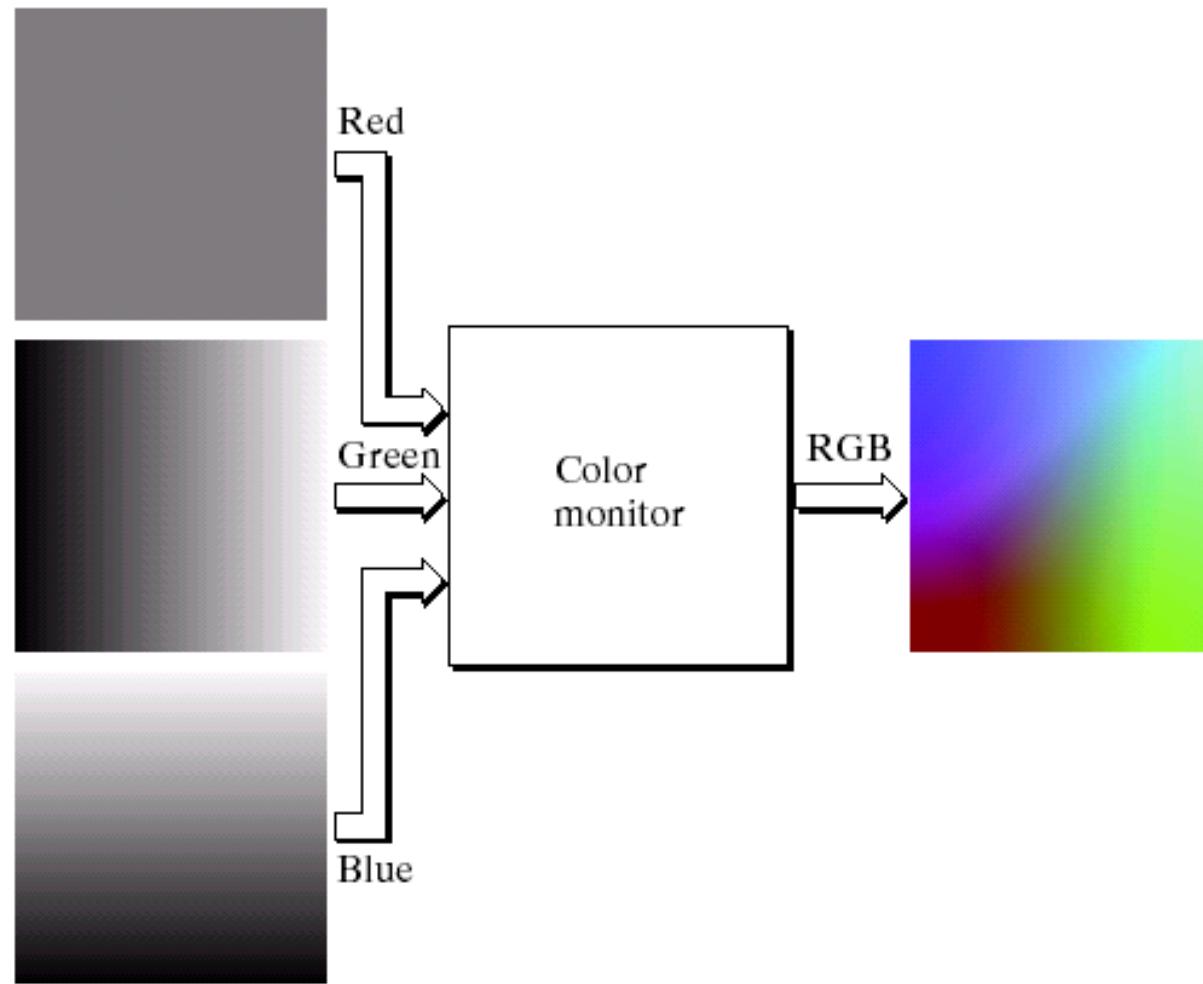
R = 8 bits
G = 8 bits
B = 8 bits

Color depth 24 bits
= 16777216 colors



Hidden faces
of the cube

RGB



The HSI Color Model

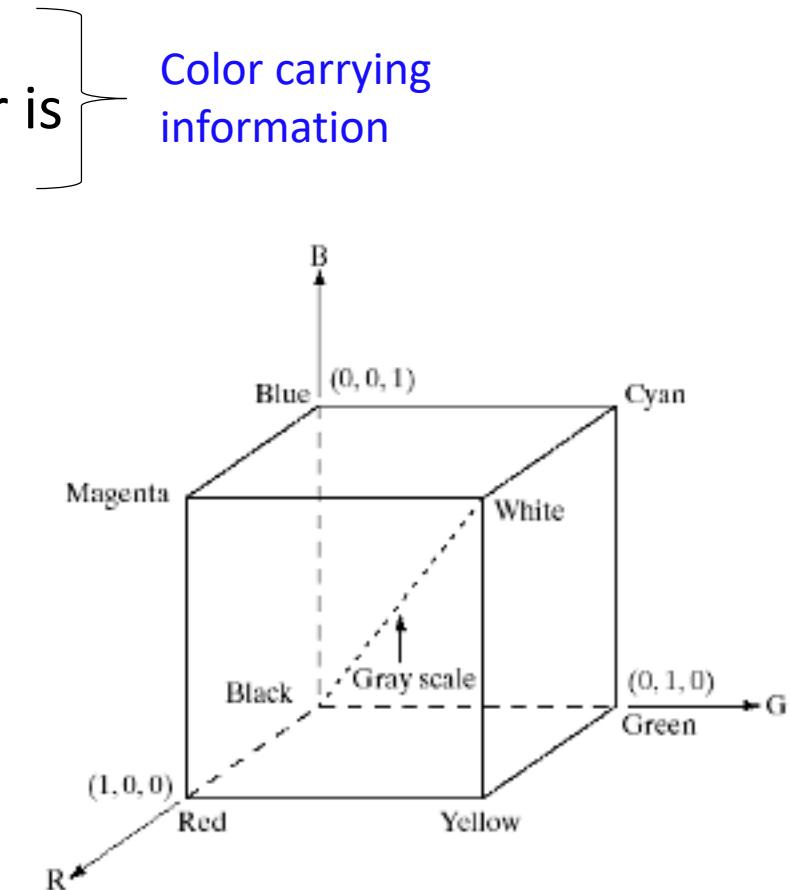
- RGB is useful for hardware implementations and is luckily related to the way in which the human visual system works
- However, RGB is not a particularly intuitive way in which to describe colors
- Rather when people describe colors they tend to use hue, saturation and brightness

RGB is great for color generation, but HSI
is great for color description

The HSI Color Model

- The HSI model uses three measures to describe colours:

- **Hue**: Dominant color
- **Saturation**: Gives a measure of how much a pure color is diluted with white light
- **Intensity**: Brightness



- **Intensity** can be extracted from RGB images
- Remember the **diagonal** on the RGB color cube ran from **black to white**

Converting From RGB To HSI

- Given a color as R, G, and B its H, S, and I values are calculated as follows:

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B > G \end{cases}$$

$$\theta = \cos^{-1} \left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2 + (R-B)(G-B)]^{\frac{1}{2}}} \right\}$$

$$S = 1 - \frac{3}{(R+G+B)} [\min(R, G, B)]$$

$$I = \frac{1}{3}(R+G+B)$$

Converting From HSI To RGB

- Given a color as H, S, and I it's R, G, and B values are calculated as follows:
- RG sector ($0 \leq H < 120^\circ$)**

$$R = I \left[1 + \frac{S \cos H}{\cos(60 - H)} \right]$$

$$G = 3I - (R + B)$$

$$B = I(1 - S)$$

- GB sector ($120^\circ \leq H < 240^\circ$)**

$$R = I(1 - S)$$

$$G = I \left[1 + \frac{S \cos(H - 120)}{\cos(H - 60)} \right]$$

$$B = 3I - (R + G)$$

- BR sector ($240^\circ \leq H < 360^\circ$)**

$$R = 3I - (G + B)$$

$$G = I(1 - S)$$

$$B = I \left[1 + \frac{S \cos(H - 240)}{\cos(H - 180)} \right]$$

Converting RGB to Greyscale

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

Some Colour Applications – Skin Detection

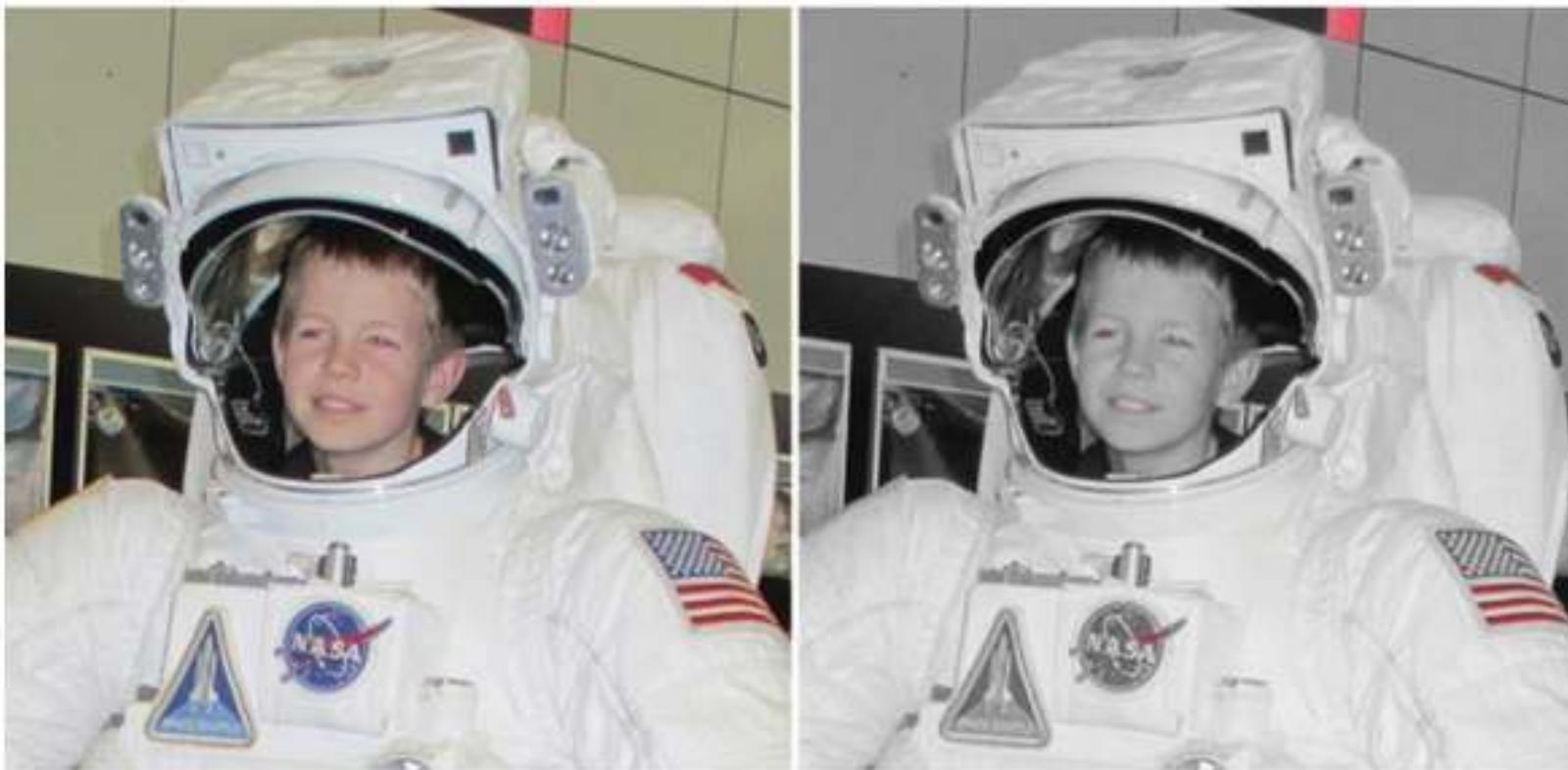


Colour image (left), selection of skin pixels by selecting all points with
 $(\text{Saturation} \geq 0.2) \text{ AND } (0.5 < \text{Luminance/Saturation} \leq 28^\circ \text{ OR } \text{Hue} \geq 330^\circ)$ (right)

How these numbers are found?

Through simple experimentation it was found

Some Colour Applications – Red Eye Detection



(Luminance ≥ 0.25) AND (Saturation ≥ 0.4) AND
($0.5 < \text{Luminance}/\text{Saturation} < 1.5$) AND (Hue $\leq 14^\circ$ OR Hue $\geq 324^\circ$)

Reading

Kenneth, A Practical Introduction to Computer Vision with OpenCV

Chapter 1

R. Szeliski, *Computer Vision: Algorithms and Applications*

[Section 1.1 What is computer vision?](#)

[Section 1.2 Brief history of computer vision](#)

Q & A

Image Filtering

By: Dr. Muhammad Fahim

Contents

- Image processing
- Point operations
 - Background Subtraction
 - Contrast Enhancement
- Neighborhood operations
 - Linear spatial filters
 - Non-linear spatial filters
- Geometric operations
 - Affine transformations
- Summary

Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - Found material over the internet to aligned the subject according to the need of the students.

Recap!!

- We can think of an **image** as a function f

$f(x, y)$ gives the **intensity** at position (x, y)

- A **color image** is just three functions pasted together. We can write this as a “vector-valued” function:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Image Processing

The image processing phase should:

- facilitate the extraction of information
- compensate for non-uniform illumination
- re-adjust the image to compensate for distortions introduced by the imaging system

Image Processing

There are **three classes** of operations

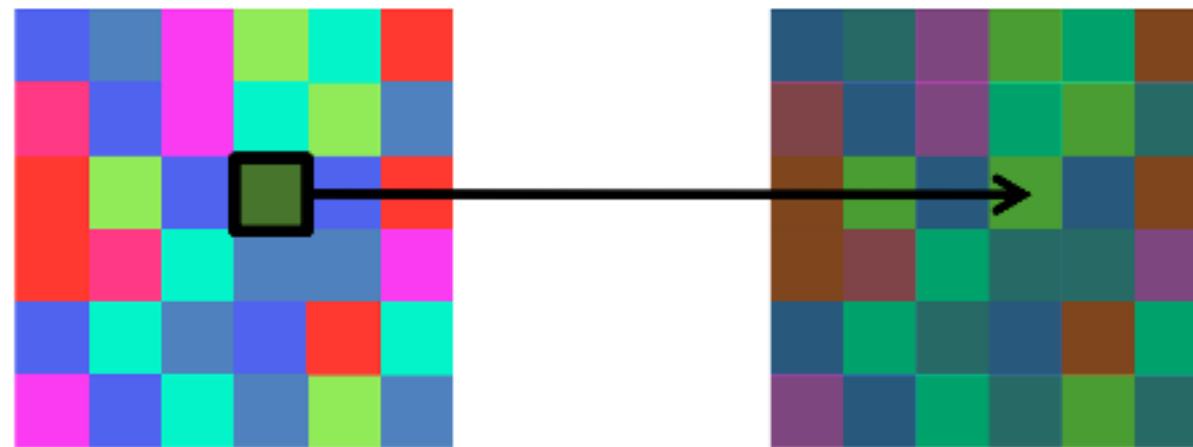
1. Point Operations
2. Neighbourhood Operations
3. Geometric Operations

1. Point Operations

- It deals with pixel intensity values individually.
- The intensity values are altered using particular transformation techniques as per the requirement.

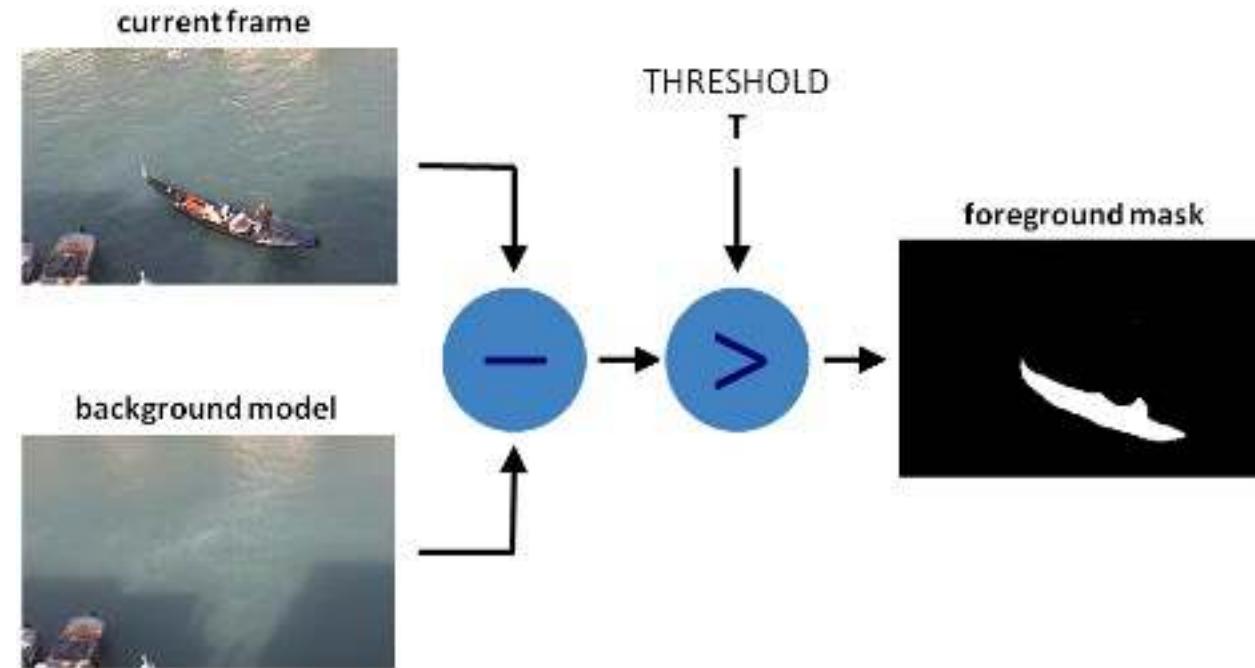
- **Examples**

- Background subtraction
- Thresholding
- Contrast enhancement
- Brightness enhancement



1. Point Operations – Background Subtraction

- Pixel values of two images are subtracted on a point-by-point basis.



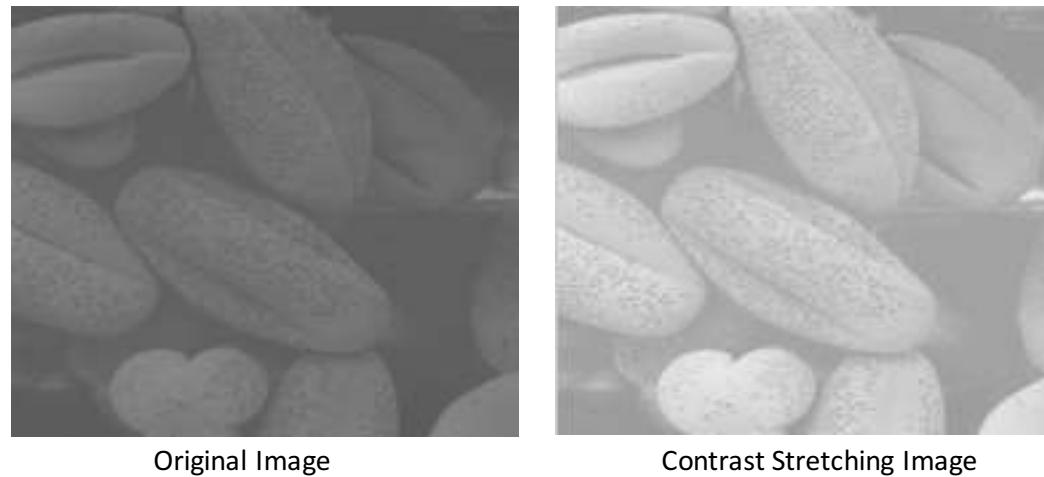
- Motion Detection:** Stationary objects cancel each other out while moving objects are highlighted

https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

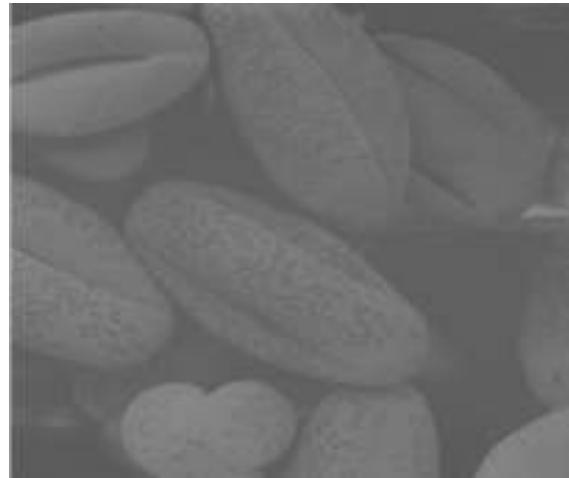
1. Point Operations – Contrast Enhancement

- Contrast basically the difference between the intensity values of darker and brighter pixels.
 - For Example: Multiplying each input pixel intensity value with a constant scalar.

$$S = 2 * r \quad (r \text{ is original image})$$



1. Point Operations – Contrast Enhancement



Original Image

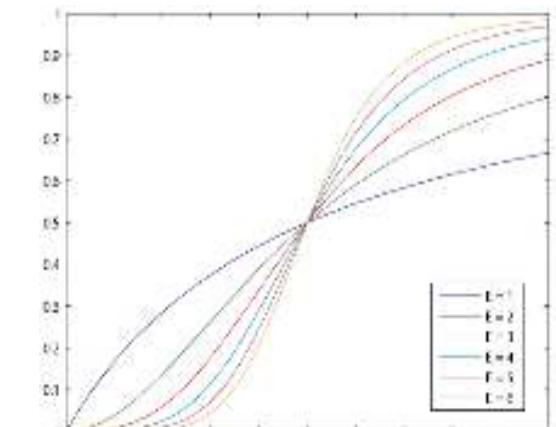


Contrast Stretching Image

- Most Commonly used method is:

$$S = \frac{1}{1 + (m/r)^E}$$

- E controls the slope of the function and m is the mid-line/mean where you want to switch from dark values to light values



Plot of contrast stretching Transformation with changing E

2. Neighbourhood Operations

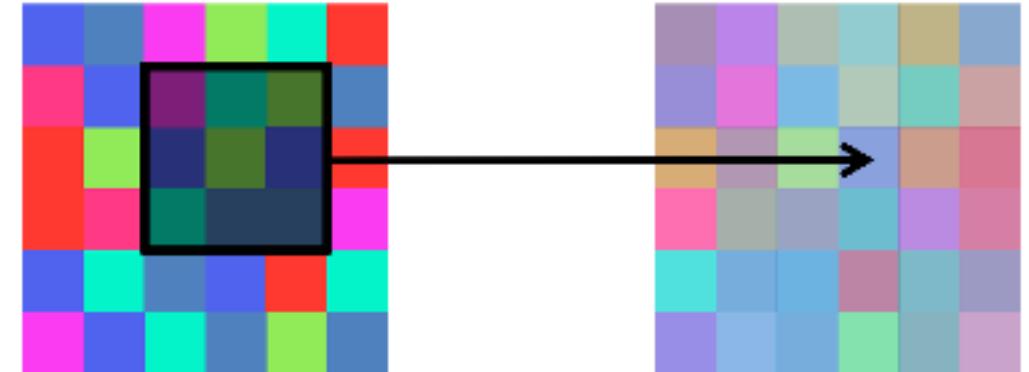
- Generate an **output** pixel on the basis of the pixel at the **corresponding position** in the input image *and on the basis of its neighbouring pixels*

- The size of the neighbourhood may vary:

$3 * 3$

$5 * 5$

$63 * 63$ pixels and so on...



- Often referred to as **Filtering Operations**

2. Neighbourhood Operations

- Applying some logical test based on the presence or absence of object pixels in a local neighbourhood surrounding
- It is used for:
 - Removal of noise
 - Object Thinning (or Skeletonising)
 - Erosion and Dilation (contract/expand an object)

2. Neighbourhood Operations – Filter/Kernel

- **Convolution** of an image f with a filter/kernel/mask h

$$g = f * h$$

- The function h is normally referred to as the **filter**
 - It dictates what elements of the input image are **allowed to pass** to the output image
 - By choosing an **appropriate filter**, we can **enhance certain aspects** of the output and **attenuate others**

2. Neighbourhood Operations – Image Filtering

- In the discrete domain of digital images, the convolution operation is given by:

$$g(i, j) = f * h = \sum_m \sum_n f(i - m, j - n)h(m, n)$$

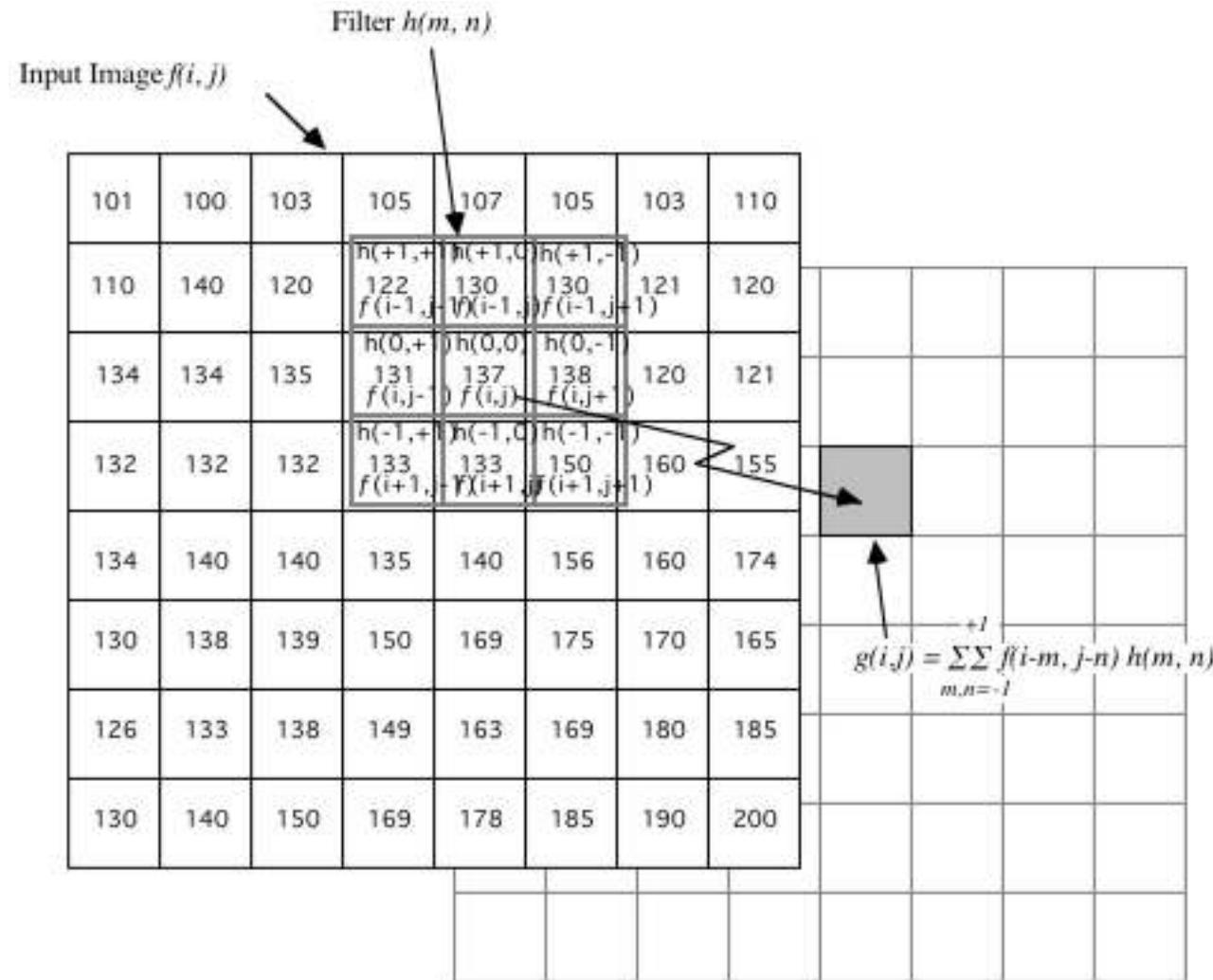
- The summation is taken only over the area where $(i-m, j-n)$ is defined, i.e. over the area where f and h overlap.

Convolutions Simplified

$h(-1, -1)$	$h(-1, 0)$	$h(-1, +1)$
$h(0, -1)$	$h(0, 0)$	$h(0, +1)$
$h(+1, -1)$	$h(+1, 0)$	$h(+1, +1)$

$3 * 3$ Convolution Filter h

Convolutions Simplified



Convolutions Simplified – Example

Here is an example of convolving (which is normally denoted mathematically as the \star operator) a 3×3 region of an image with a 3×3 kernel:

$$O_{i,j} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \star \begin{bmatrix} 93 & 139 & 101 \\ 26 & 252 & 196 \\ 135 & 230 & 18 \end{bmatrix} = \begin{bmatrix} -1 \times 93 & 0 \times 139 & 1 \times 101 \\ -2 \times 26 & 0 \times 252 & 2 \times 196 \\ -1 \times 135 & 0 \times 230 & 1 \times 18 \end{bmatrix}$$

Therefore,

$$O_{i,j} = \sum \begin{bmatrix} -93 & 0 & 101 \\ -52 & 0 & 392 \\ -135 & 0 & 18 \end{bmatrix} = 231$$

After applying this convolution, we would set the pixel located at the coordinate (i, j) of the output image O to $O_{i,j} = 231$.

2. Neighbourhood Operations – Noise Removal

- What is noise?
 - Any **unwanted contamination** of an image
- Why we have noise in images?
 - The sources of noise in digital images arise during image acquisition and transmission
 - Imaging sensors can be affected by **ambient conditions**
 - Interference can be added to an image **during transmission**
 - Old photographs
 - and so on...

Common Type of Noise – Impulse Noise

- This type of noise can be caused by **analog-to-digital converter errors, bit errors in transmission, etc.**

Color Image



Grey-scale Image



- It contains **random occurrences of black and white pixels**

It is also known as **salt-and-pepper noise**.

Common Type of Noise – Gaussian Noise

- Principal sources of Gaussian noise in digital images arise [during acquisition](#).
- The sensor has inherent noise due to the [level of illumination](#) and its own temperature
- The electronic circuits connected to the sensor inject their own [share of electronic circuit noise](#)

Color Image



Grey-scale Image



Noise Models

- Noise must be joined with the image data in some way.
- The way in which we model this depends upon whether the noise is data independent or data dependent.
- Additive noise models – Data independent noise
- Multiplicative noise models – Data dependent noise

Additive Noise Model

- Additive noise models are **data independent** (*i.e., noise where the amount of noise is not related to the image data itself*)

$$f(i,j) = g(i,j) + v(i,j)$$

Where,

$g(i,j)$ is the ideal image

$v(i,j)$ is the noise

$f(i,j)$ is the actual image

Multiplicative Noise Model

- It is data dependent noise (*i.e., noise where the amount of noise is related to the image data itself*)

$$f(i,j) = g(i,j) + g(i,j).v(i,j)$$

Where,

$g(i,j)$ is the ideal image

$v(i,j)$ is the noise

$f(i,j)$ is the actual image

Noise Reduction Techniques

- Linear Spatial Filters
 - Averaging/Smoothing Filter
 - Gaussian Filter
- Non-linear Spatial Filters
 - Order-statistics Filter

NOTE: Noise reduction techniques are known as **denoising**.

Averaging/Smoothing Filter

- An average filter does exactly what you think it might do — takes an area of pixels surrounding a central pixel, averages all these pixels together, and replaces the central pixel with the average.
- By taking the average of the region surrounding a pixel, we are smoothing it and replacing it with the value of its local neighborhood.
- This allows us to reduce noise and the level of detail, simply by relying on the average.

Averaging/Smoothing Filter

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



- Also known as **box-filter**
- It replaces pixel **with local average**
- It has **smoothing (blurring) effect**

Two 3x3 Smoothing Linear Filters

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

Standard average

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

Weighted average

Check Point!!

$$\frac{1}{?} \times$$

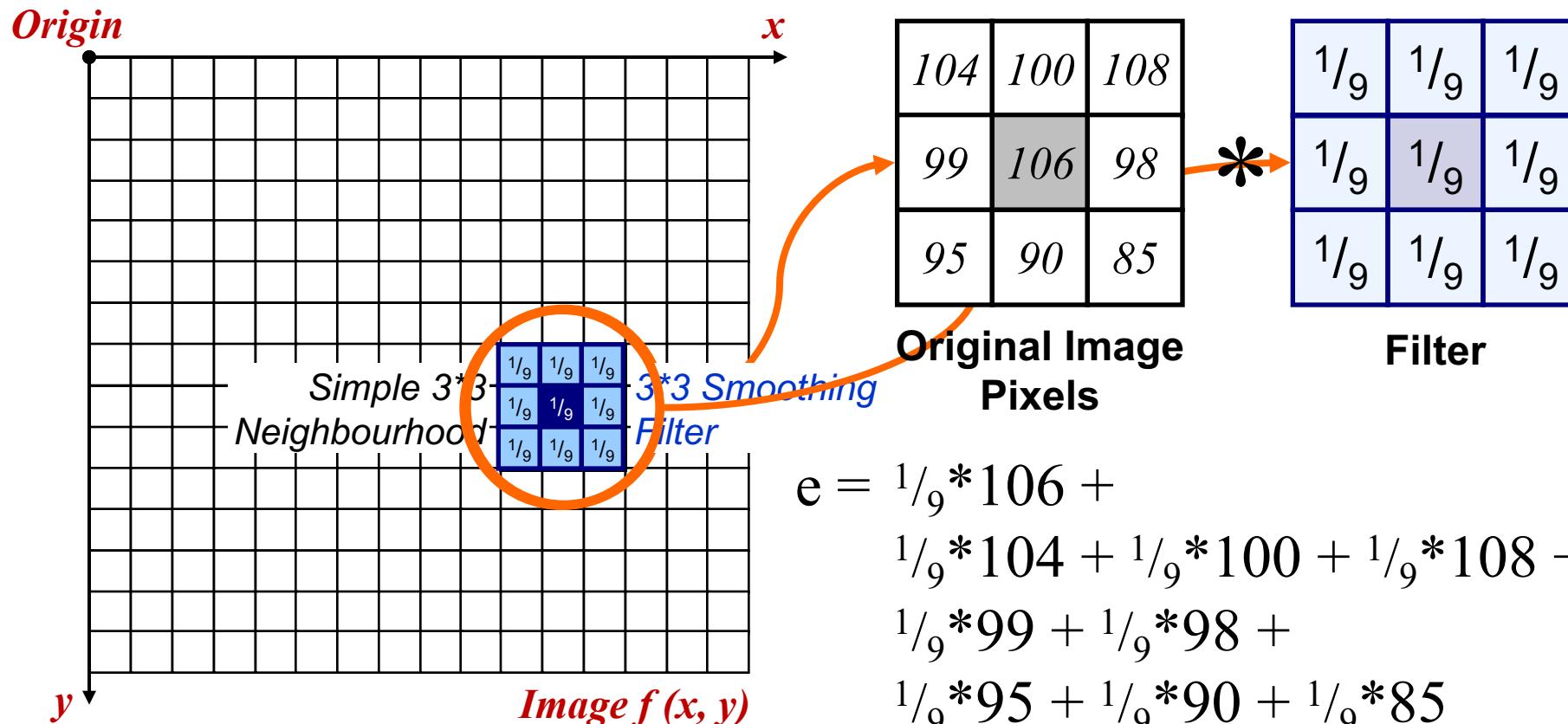
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

5x5 Smoothing Linear Filters

$$\frac{1}{25} \times$$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

3x3 Smoothing Linear Filters (Averaging)



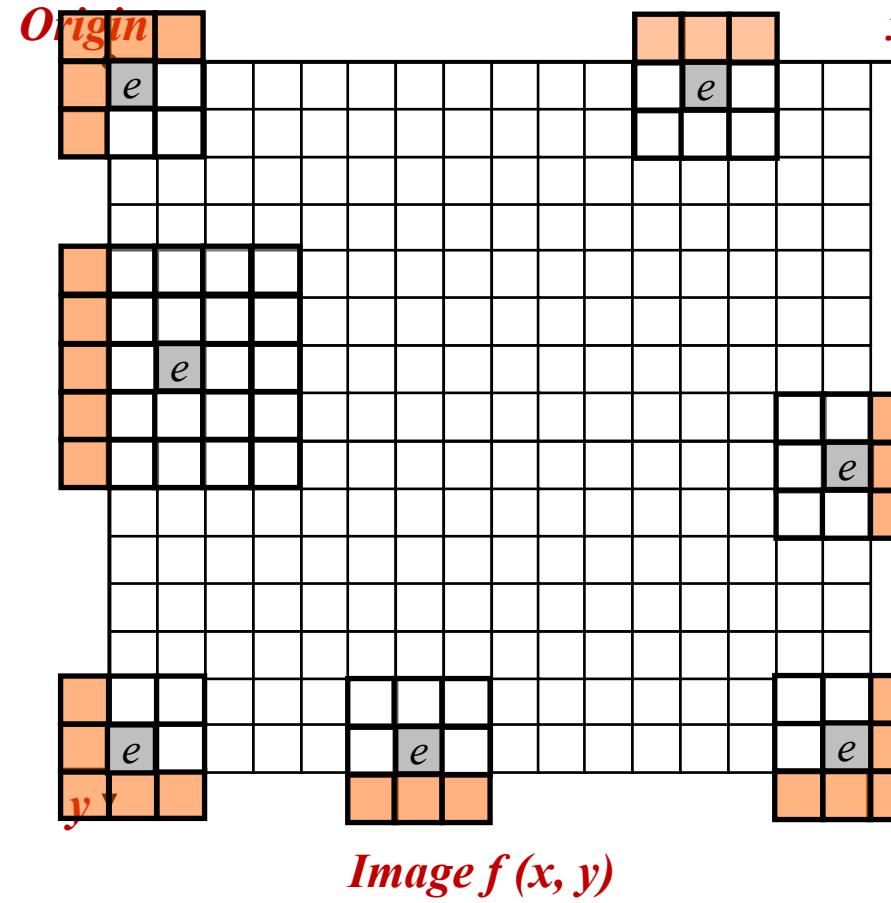
The above process is repeated for every pixel in the original image

Dealing with Edges

- There are a few approaches to deal with edge pixels
 - Omit missing pixels
 - Only works with some filters
 - Can add extra code and slow down processing
 - Pad the image
 - Typically with either all white or all black pixels
 - Replicate border pixels
 - Truncate the image
 - Or some other....

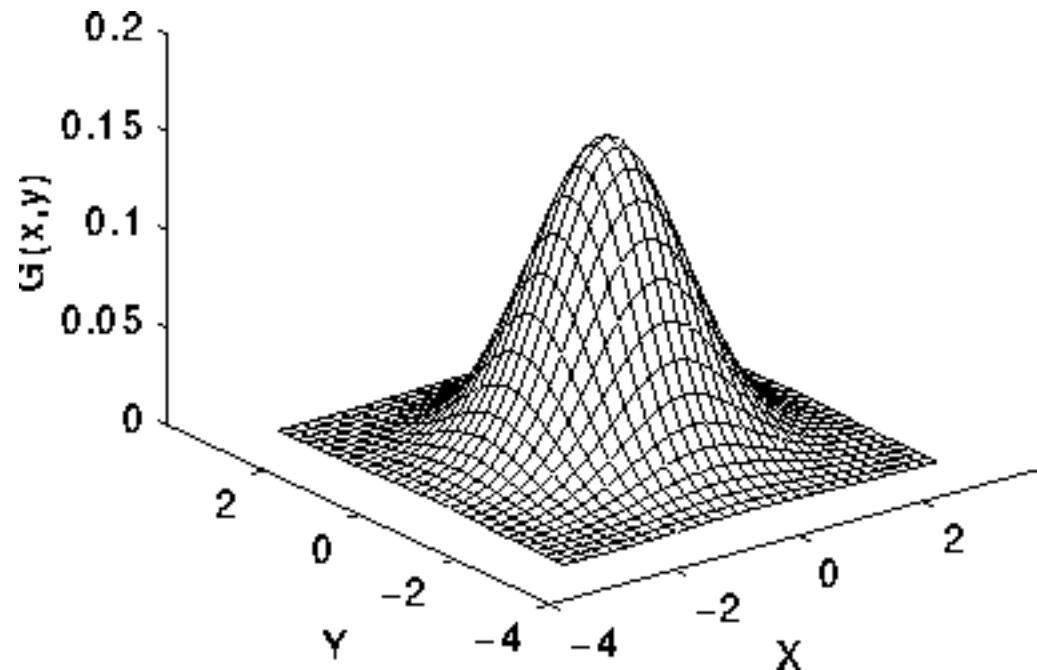
Dealing with Edges

- At the edges of an image we are missing pixels to form a neighbourhood



Gaussian Filter

- It is another noise suppression techniques

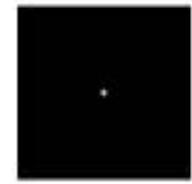
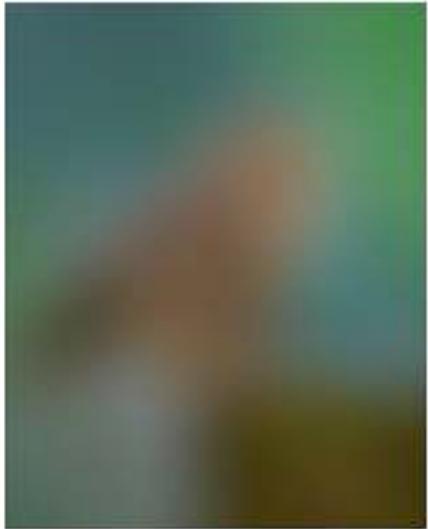
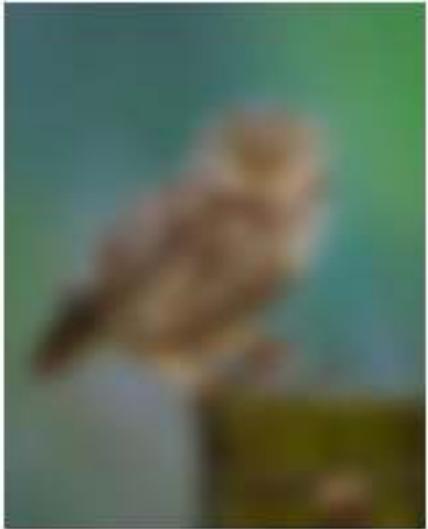
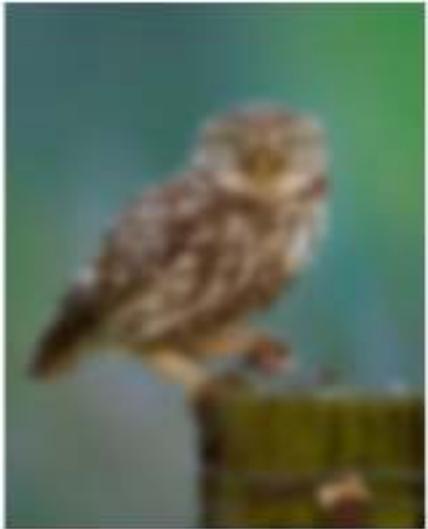


$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

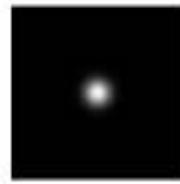


- σ defines the effective spread of the function

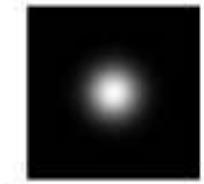
Gaussian Filter



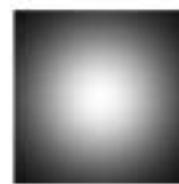
$\sigma = 1$ pixel



$\sigma = 5$ pixels

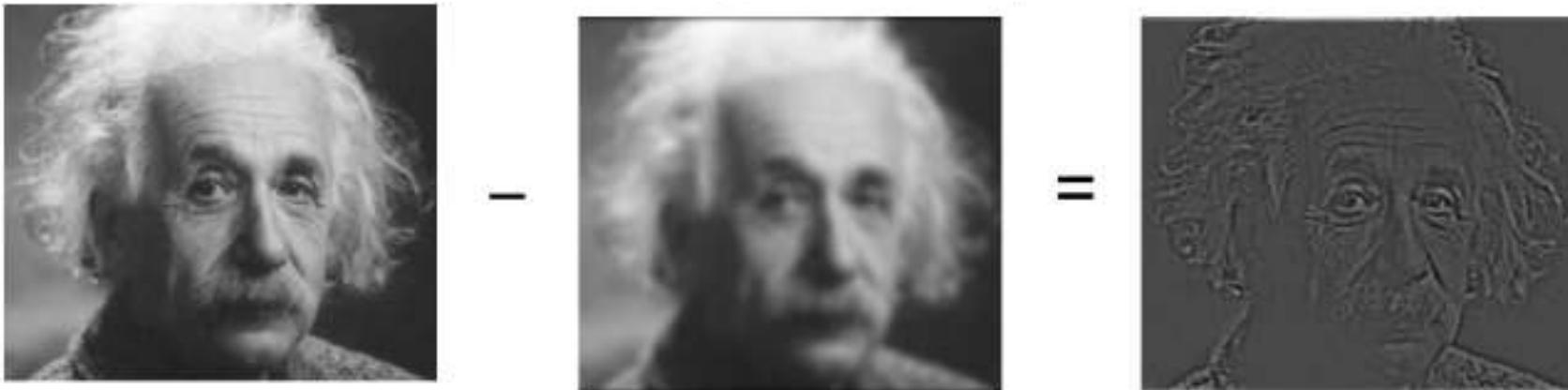


$\sigma = 10$ pixels

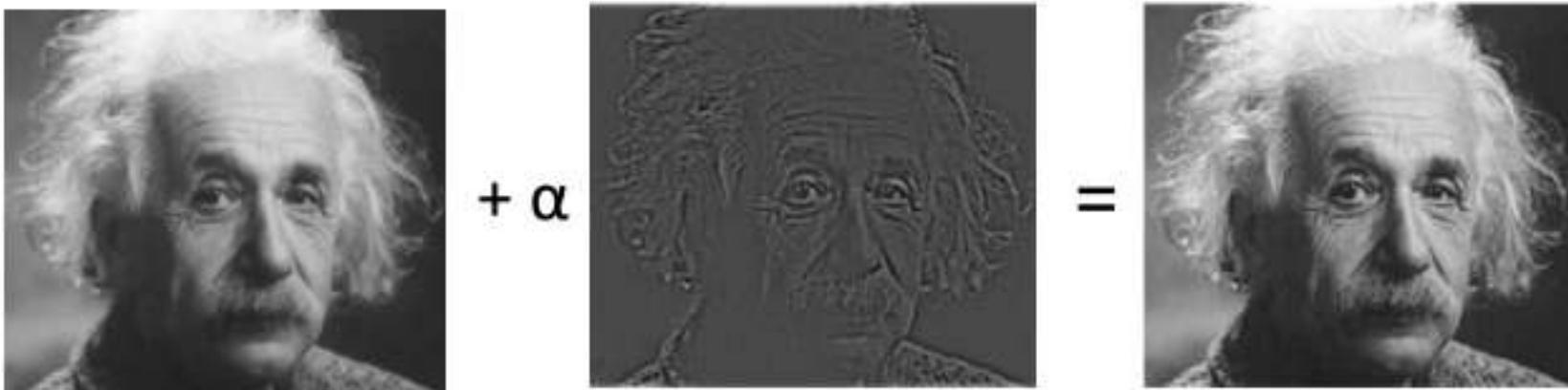


$\sigma = 30$ pixels

What does blurring take away?



Let's add it back:



Sharpening

Order-Statistics Filters

- Order-statistics filters are **nonlinear spatial filters**
- Response is based on **ordering (ranking) the pixels contained in the image area encompassed by the filter**, and then **replacing the value of the center pixel with the value determined by the ranking result.**
- Best-known “median filter”

Process of Median Filter

	10	15	20	
	20	100	20	
	20	20	25	

10, 15, 20, 20, 20, 20, 20, 25, 100

↑
5th

- Crop the region of neighborhood
- Sort the values of the pixel in our region
- In the $M \times N$ mask the median is $(M \times N \text{ div } 2) + 1$

Result of Median Filter



Noise from Glass effect



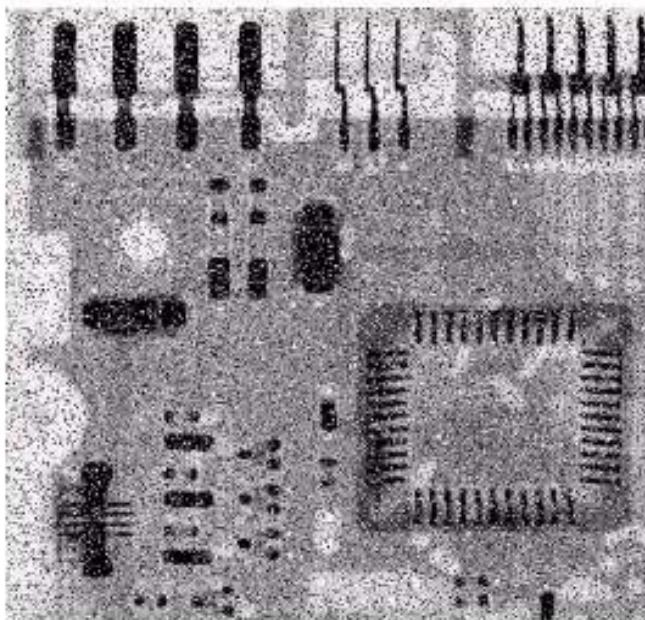
Remove noise by median filter



Noise Removed by
median filter

Comparative Analysis

- Sometimes a median filter works better than an averaging filter



Original Image
With Noise

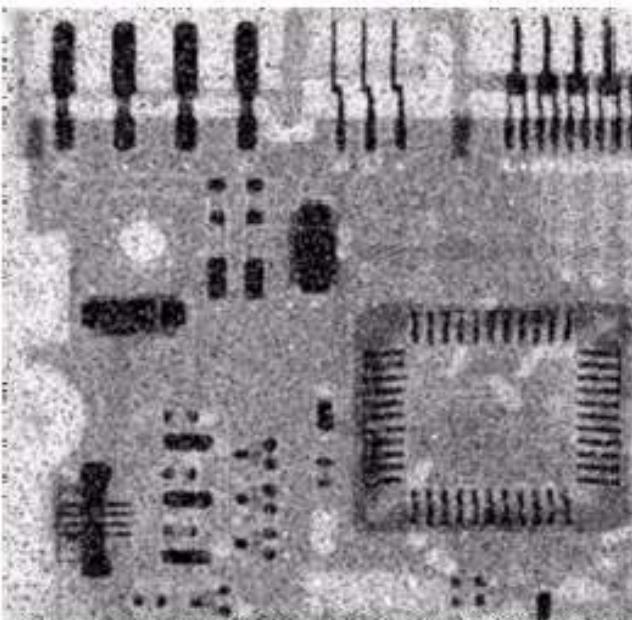


Image After
Averaging Filter

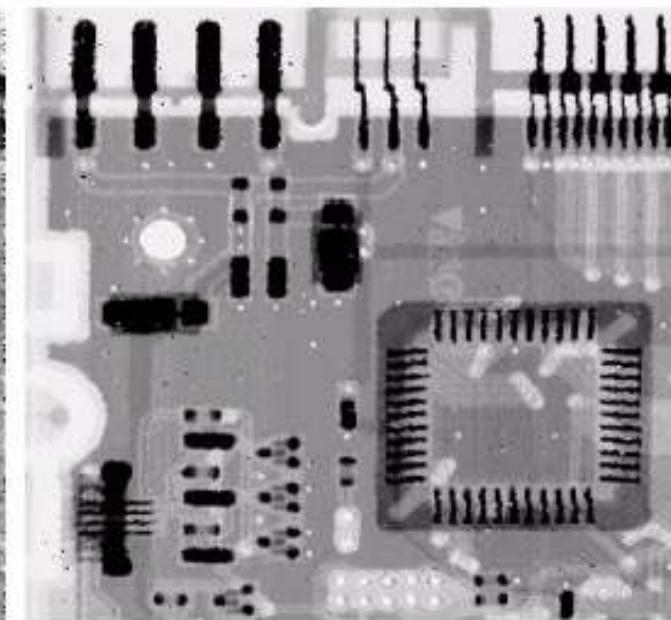
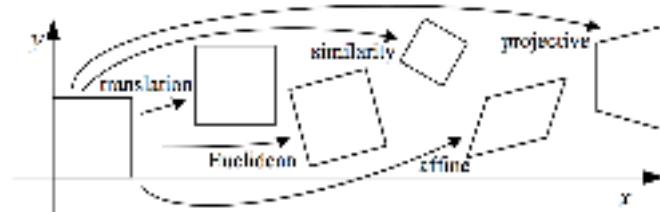
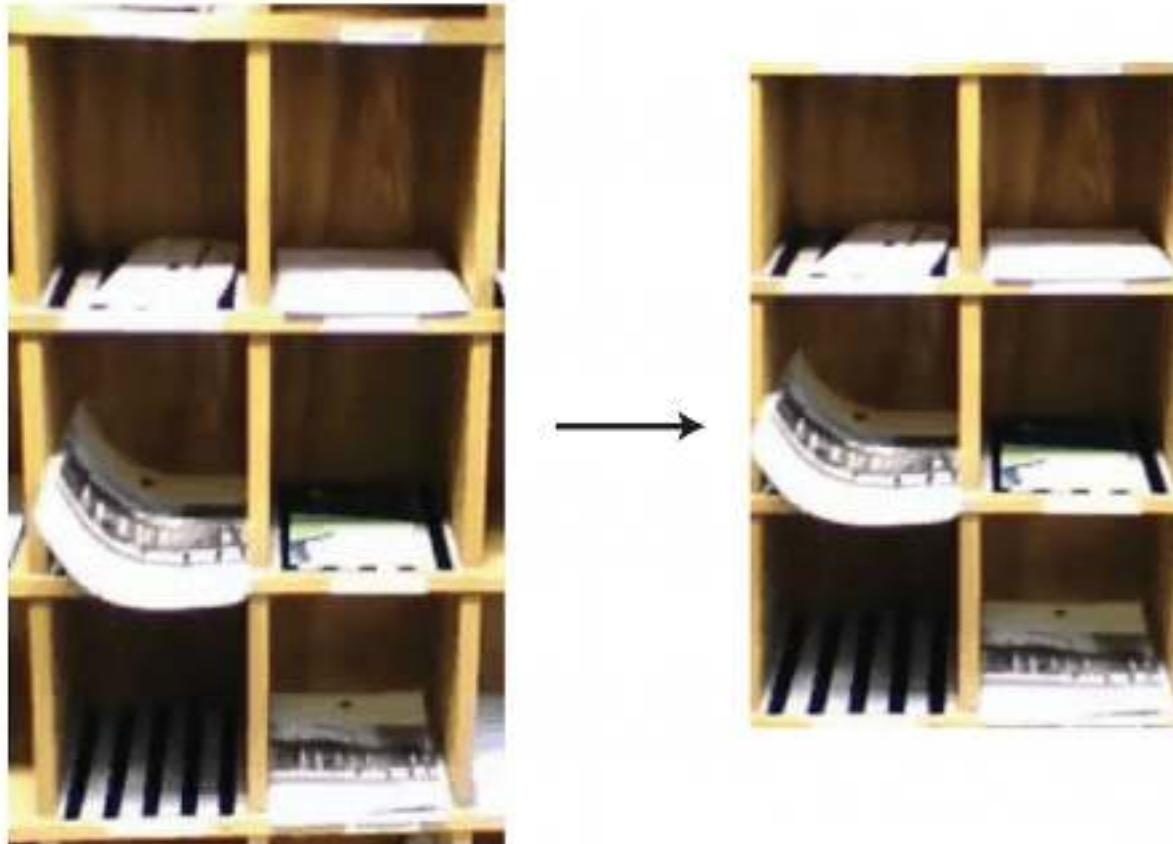


Image After
Median Filter

Geometric Operations



- Change the **spatial relationship** between objects in an image



A geometric transformation which **corrects the perspective distortion** introduced when viewing a **planar object**

Why we need Geometric Transformations?

- Correct effects of camera orientation
- Correct images for lens distortion
- Align images that were taken with different sensors
- Image morphing or other special effects

Affine Transformation

- Many common transformations can be described using the [affine transform](#), which is defined as follows:

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

Known Affine Transformation – Translation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & 0 & m \\ 0 & 1 & n \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Translation by m along the horizontal axis and n along the vertical axis

Known Affine Transformation – Translation

Derive the unified expression on whiteboard!!

Change of Scale (Expand/Shrink)

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Change of scale a in the horizontal axis and b in the vertical axis.



Check Point!!

- Let either a or b go negative. What will happen?

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Rotation by angle ϕ about the origin.
- This is sometimes required to align an image with the axes to simplify further processing.



Skewing

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 1 & \tan \phi & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$

- Skewing by angle ϕ .
- Often needed to **remove nonlinear viewing effects** such as those generated by a line scan camera.



Panoramic Distortion

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i' \\ j' \\ 1 \end{bmatrix}$$



- Panoramic distortion is in effect an **incorrect aspect ratio**.
- It appears in **line scanners** when the **mirror rotates at an incorrect speed**.



Rotation, followed by skewing, followed by panoramic distortion of an image

Summary

- We talked about image filtering
- Point operations
 - Background Subtraction
 - Contrast Enhancement
- Neighborhood operations
 - Linear spatial filters
 - Non-linear spatial filters
- Geometric operations
 - Affine transformations

Reading

- Kenneth: Chapter 2, Section 4.4, Chapter 5
- Szeliski: Chapter 3.1-3.2

Thank You 😊

Binary Vision

(Segmentation, Image Gradients, Morphology)

By: Dr. Muhammad Fahim

Contents

- What is Image Segmentation?
- Segmentation Methods
 - Thresholding
 - Image Gradients – Edge Detection
- Mathematical Morphology
 - Erosion
 - Dilation
 - Compound Operations
- Summary

Slide Credits and Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - Volker Krüger & Rune Andersen, University of Aalborg, Esbjerg
 - *Lecture Material is based on Gonzalez, Rafael C. Digital image processing. Pearson Education India, 2009.*
 - <https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm>
 - <https://theailearner.com/2019/05/11/understanding-image-gradients/>
 - Found material over the internet to aligned the subject according to the need of the students.

What is Image Segmentation?

- It is a process of **partitioning** a digital image into **sets of pixels** which are similar with respect to some **characteristic or computed property**, such as **color, intensity, or texture**.



Image Binarization

- Binary images often arise as a **result** of certain operations such as **segmentation**.
- Image binarization applies often just one **global threshold T** for mapping a scalar image I into a binary image.

$$S(x, y) = \begin{cases} 0 & \text{if } I(x, y) < T \\ 1 & \text{otherwise} \end{cases}$$

Segmentation Methods

- Thresholding
 - Bimodal Histogram
 - Otsu's Method
 - Band Thresholding
 - Semi-Thresholding
 - Adaptive Thresholding
- Image Gradients – Edge Detection
 - Gradient Filters
 - Sobel
 - Prewitt
 - Roberts
 - Scharr
 - Laplacian of Gaussian (Marr-Hildreth)
 - Canny Edge Detector

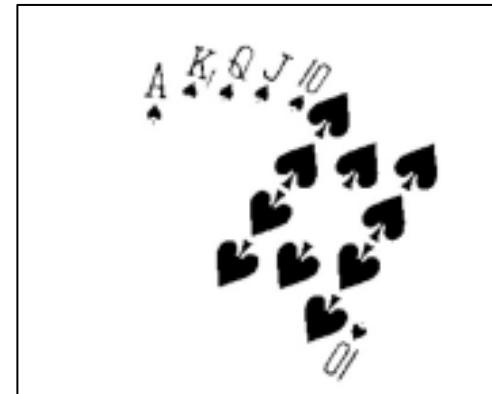
Many more....

Segmentation Method – Thresholding

- The thresholding operation is generally used in order to **separate some objects of interest** from the **background**.



Original Image



Thresholded Image



Original Image



Thresholded Image

- Thresholds are determined using a **variety of different ways**.

Segmentation Method – Simple Thresholding

- The most efficient means of implementing this operation (which can often be done in hardware) is to use a lookup table (LUT)

For all pixels (i, j)

$$\begin{aligned}f'(i, j) &= 1 \text{ where } f(i, j) \geq T \\&= 0 \text{ where } f(i, j) < T\end{aligned}$$

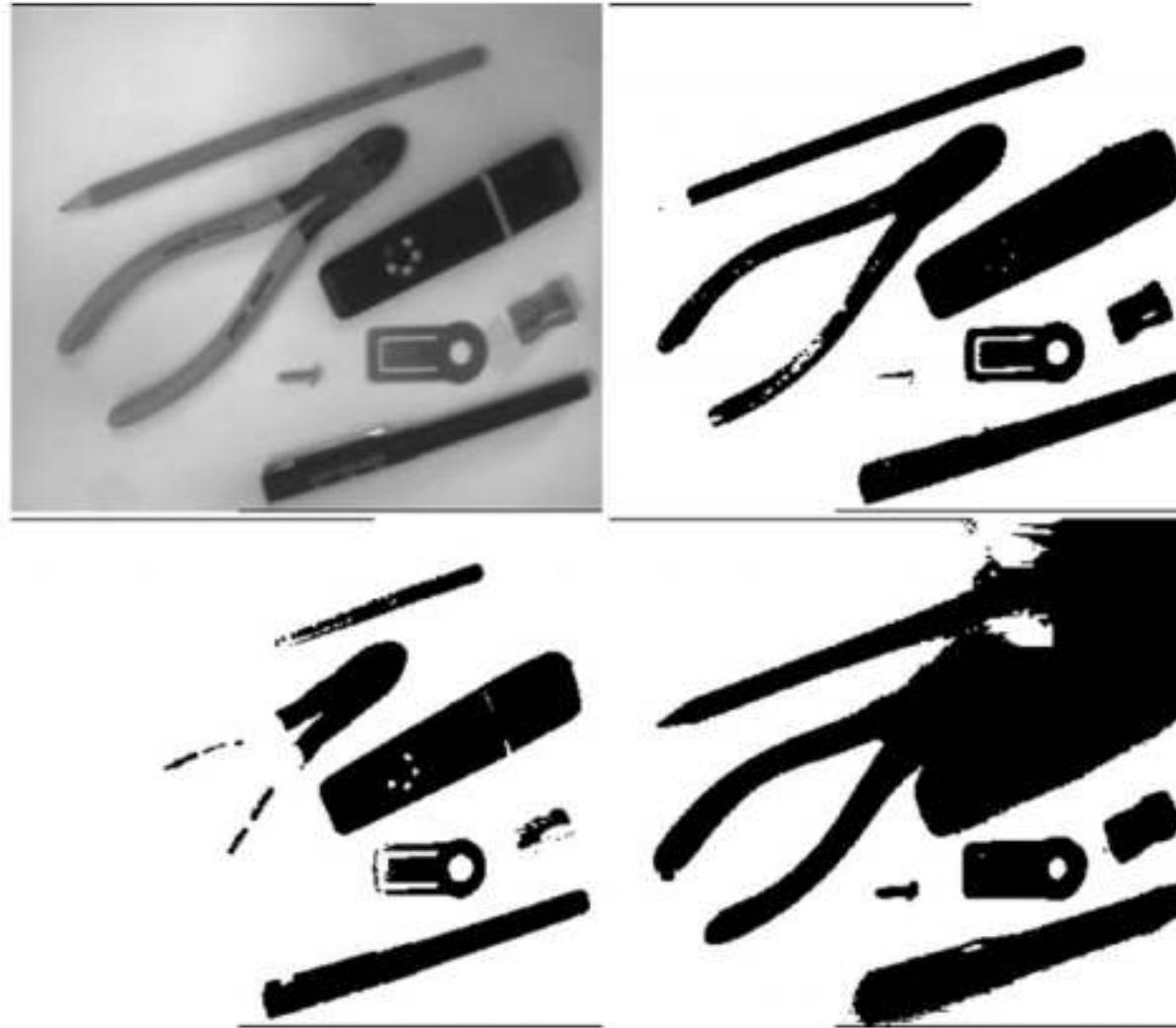
$$\begin{aligned}\text{For all grey levels } k = 0 \dots 255 \\ \text{LUT}(k) &= 1 \text{ where } k \geq T \\&= 0 \text{ where } k < T\end{aligned}$$

For all pixels (i, j)

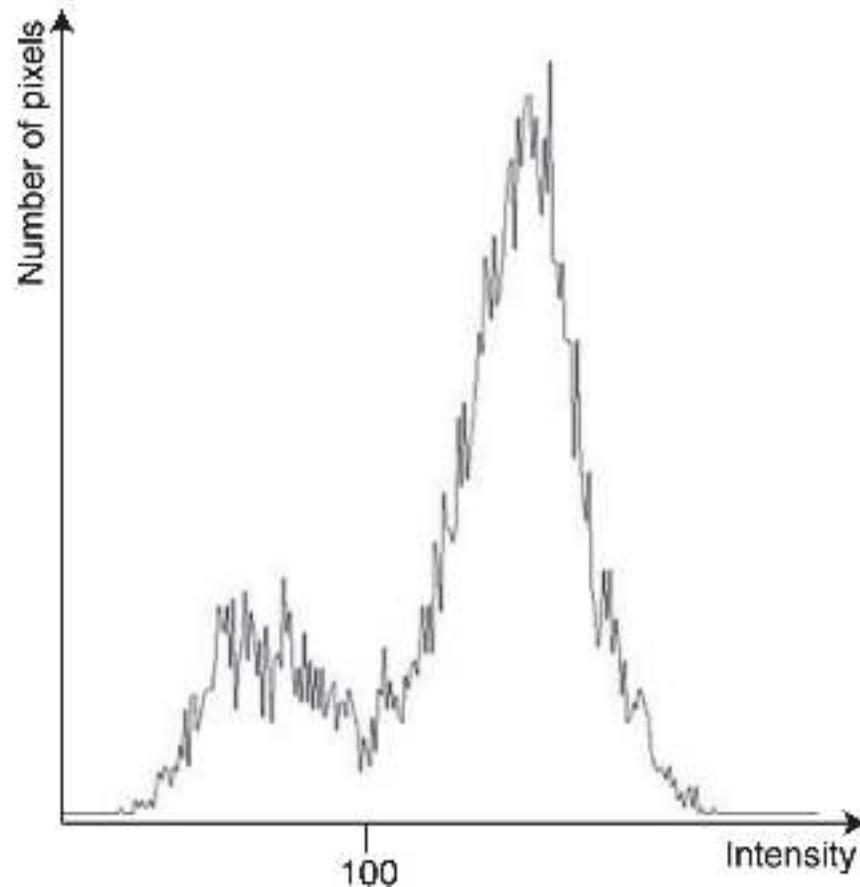
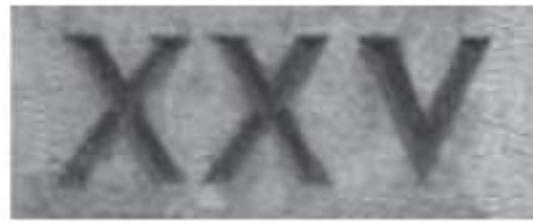
$$g(i, j) = \text{LUT}(f(i, j))$$

NOTE: Often grey-level 255 is used instead of binary 1 (so that the resulting image can be represented using a 8-bit format and displayed/processed in the same manner as the original grey-scale image).

Thresholding is not Easy!



Segmentation Method – Bimodal Histogram

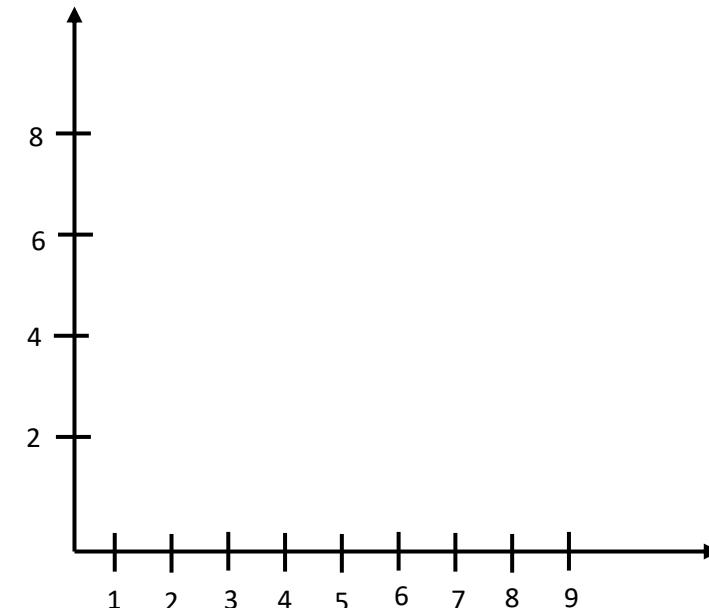


Histogram

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

$f(x,y)$

No	Freq.
1	
2	
3	
4	
5	
6	
7	
8	
9	



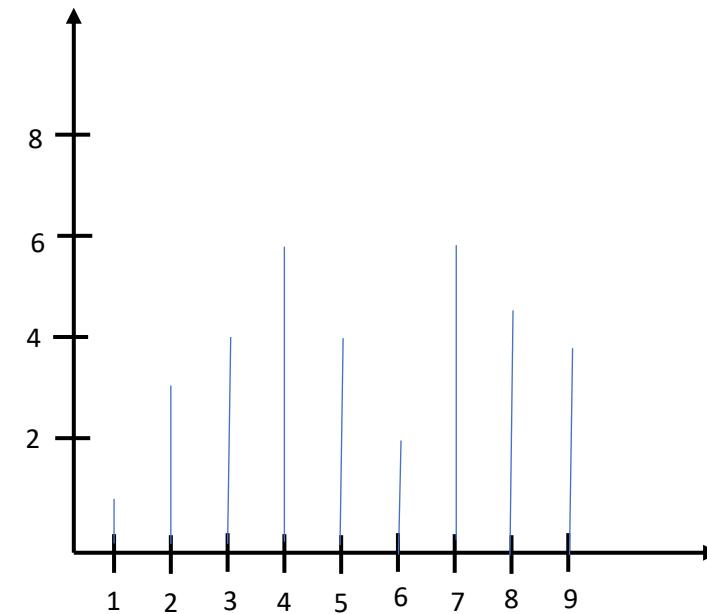
Histogram

Histogram

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

$f(x,y)$

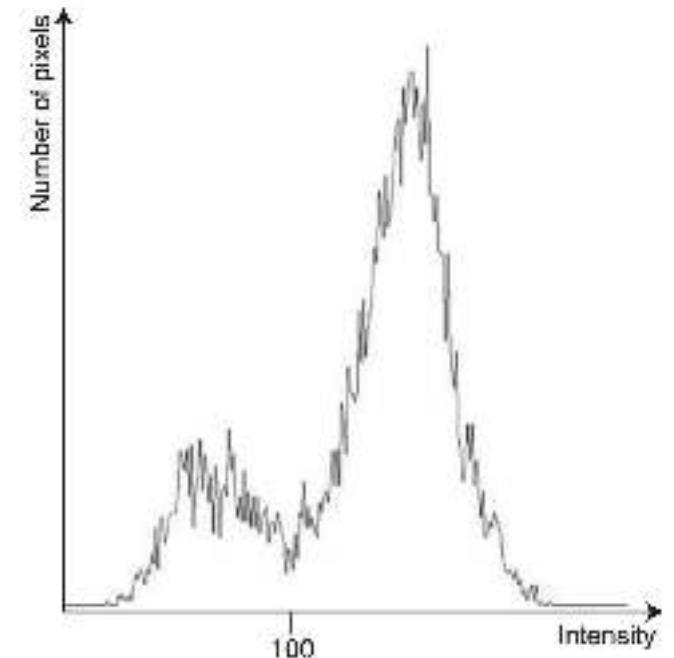
No	Freq.
1	1
2	3
3	4
4	6
5	4
6	2
7	7
8	5
9	4



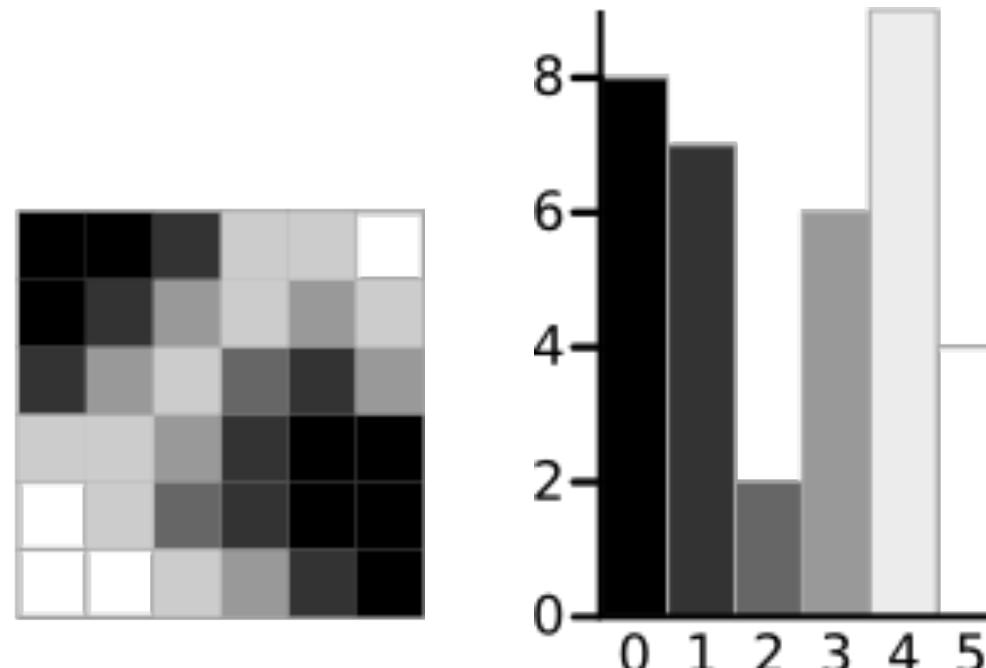
Segmentation Method – Otsu’s Method

- Simple thresholding requires **manually selecting** the threshold value
- Otsu’s method **automates** this selection
- **Assumptions:**
 1. Assumes that our **image contains two classes** of pixels: the ***background*** and the ***foreground***.
 2. Assumes that the grayscale histogram of our pixel intensities of our image is ***bi-modal***, which simply means that the histogram is **two peaks**.

- Compute the threshold that achieves
 - Smallest within-class (intra-class) variance
 - Largest between-class (inter-class) variance



Otsu's Method – Explanation

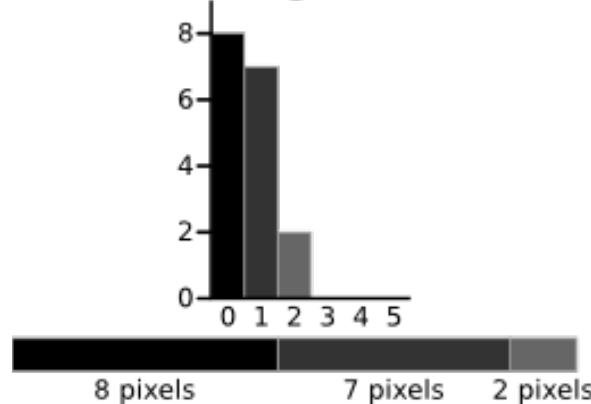


A 6-level greyscale image and its histogram

Example is taken from: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>

Otsu's Method – Explanation

Background

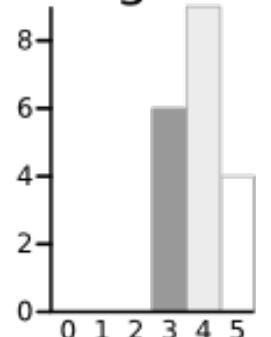


$$\text{Weight } W_b = \frac{8 + 7 + 2}{36} = 0.4722$$

$$\text{Mean } \mu_b = \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471$$

$$\begin{aligned}\text{Variance } \sigma_b^2 &= \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7) + ((2 - 0.6471)^2 \times 2)}{17} \\ &= \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17} \\ &= 0.4637\end{aligned}$$

Foreground



$$\text{Weight } W_f = \frac{6 + 9 + 4}{36} = 0.5278$$

$$\text{Mean } \mu_f = \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947$$

$$\begin{aligned}\text{Variance } \sigma_f^2 &= \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9) + ((5 - 3.8947)^2 \times 4)}{19} \\ &= \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 4)}{19} \\ &= 0.5152\end{aligned}$$

Example is taken from: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>

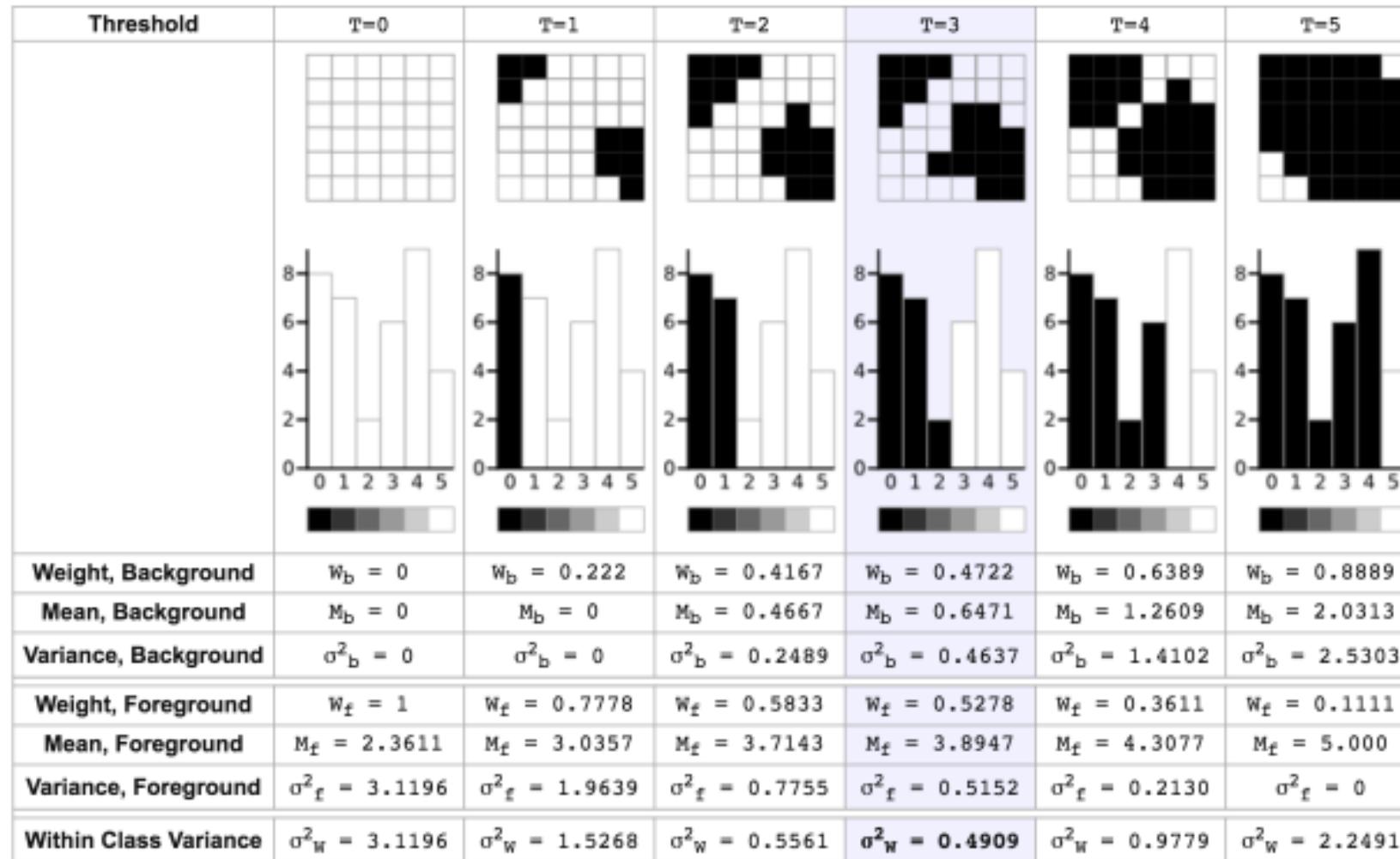
Otsu's Method – Explanation

- The next step is to calculate the '**Within (Intra)-class variance**'.
- **Within-class variance** = sum of the two variances multiplied by their associated weights.

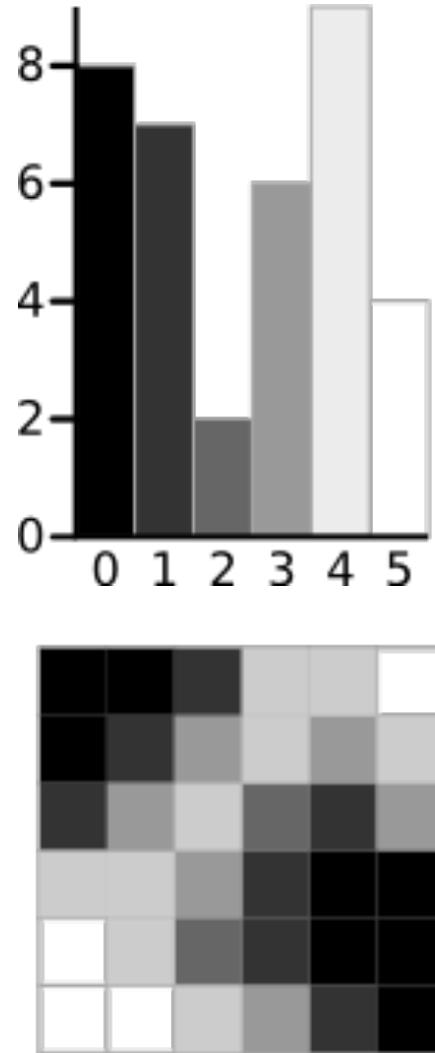
$$\text{Within Class Variance} \quad \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 * 0.4637 + 0.5278 * 0.5152 \\ = 0.4909$$

Example is taken from: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>

Otsu's Method – Explanation



Example is taken from: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>



Otsu's Method – Explanation

$$\text{Within Class Variance } \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2$$

$$\text{Between Class Variance } \sigma_B^2 = W_b W_f (\mu_b - \mu_f)^2$$

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
Within Class Variance	$\sigma_W^2 = 3.1196$	$\sigma_W^2 = 1.5268$	$\sigma_W^2 = 0.5561$	$\sigma_W^2 = 0.4909$	$\sigma_W^2 = 0.9779$	$\sigma_W^2 = 2.2491$
Between Class Variance	$\sigma_B^2 = 0$	$\sigma_B^2 = 1.5928$	$\sigma_B^2 = 2.5635$	$\sigma_B^2 = 2.6287$	$\sigma_B^2 = 2.1417$	$\sigma_B^2 = 0.8705$

Example is taken from: <http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html>

Otsu's Method – Example

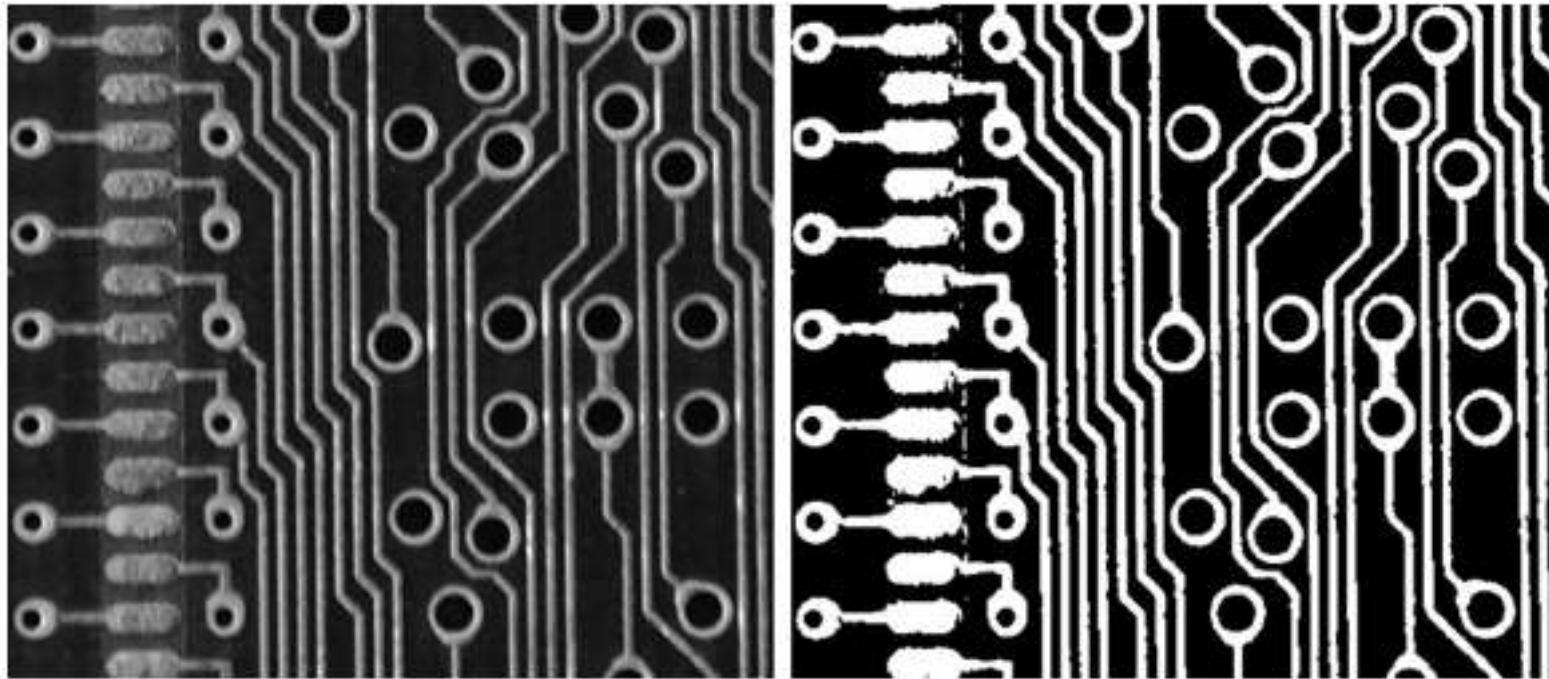


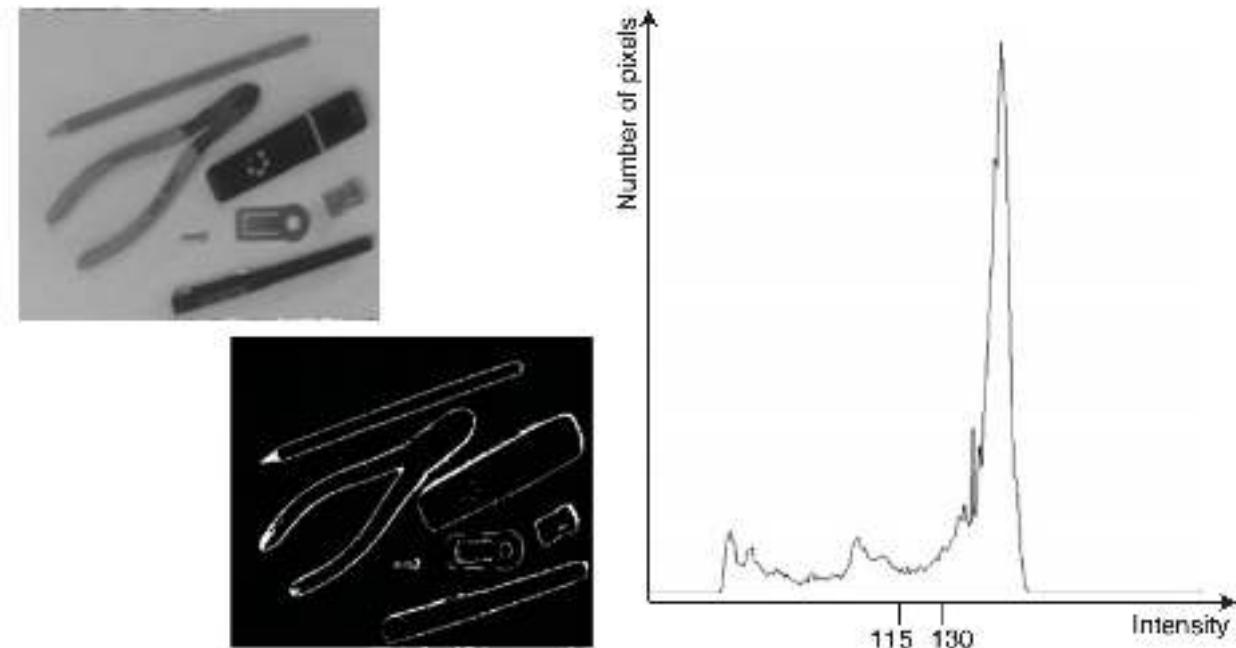
Figure 4.6 Otsu thresholding of a grey-scale image. Note that some points from the background of the pads (on the left-hand side) are being erroneously classified as foreground, and some of the points on the pads themselves are being erroneously classified as background. These issues can be dealt with by using very specific lighting configurations

Segmentation Method – Band Thresholding

- In band thresholding **two thresholds** are used, one below and one above the object pixels:

For all pixels (i, j)

$$\begin{aligned}f'(i, j) &= 1 \text{ for } f(i, j) \geq T_1 \text{ and } f(i, j) \leq T_2 \\&= 0 \text{ otherwise}\end{aligned}$$



Segmentation Method – Semi-Thresholding

- Semi thresholding is where **object pixels** retain their original grey-scale and background pixels are set to black.

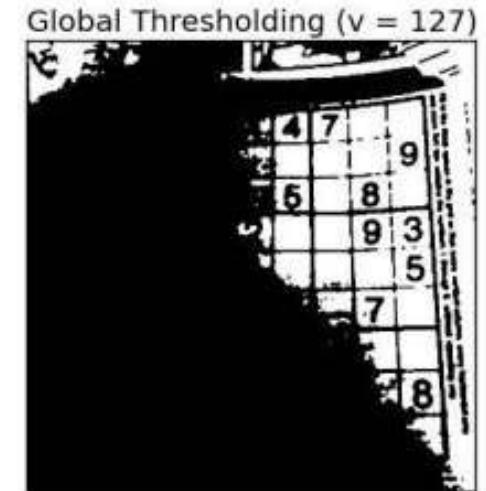
For all pixels (i,j)

$$f'(i,j) = f(i,j) \text{ for } f(i,j) \geq T \\ = 0 \text{ otherwise}$$



Segmentation Method – Adaptive Thresholding

- Having one or two value of T for the entire image is not enough
- Adaptive thresholding, calculate the threshold for a **small regions** of the image.
- So, we get **different thresholds** for **different regions** of the same image, and it gives us **better results** for images with varying illumination.



https://docs.opencv.org/3.4.0/d7/d1b/group_imgproc_misc.html#gaa42a3e6ef26247da787bf34030ed772caf262a01e7a3f112bbab4e8d8e28182dd

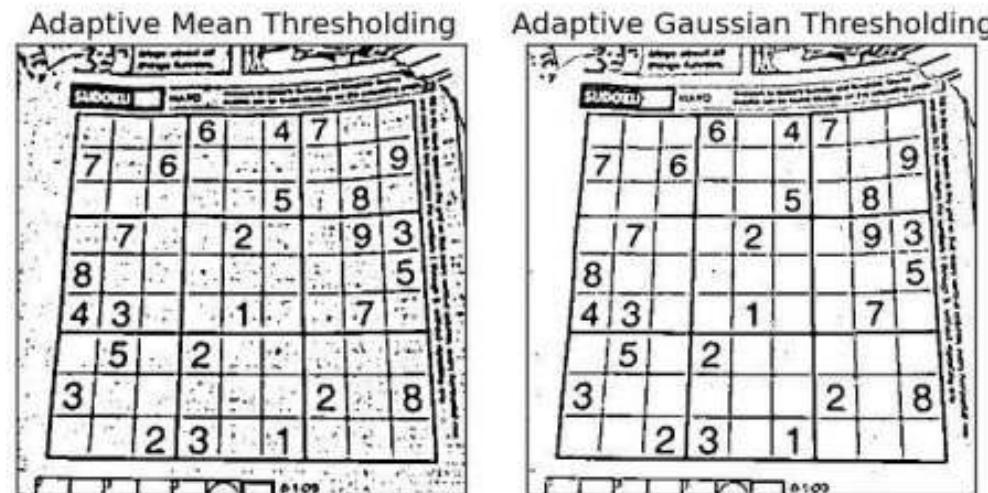
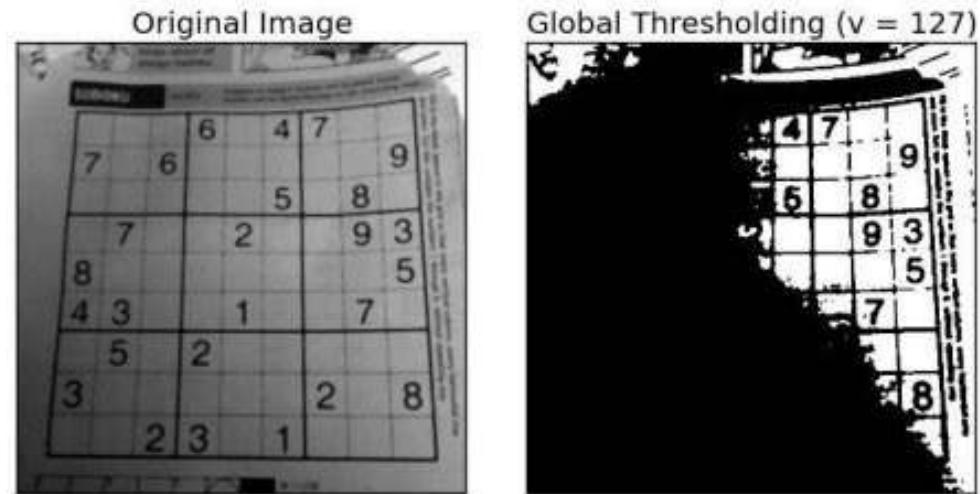
Adaptive Thresholding

- **Assumption**

- Smaller regions of an image are more likely to have approximately uniform illumination.
- This implies that local regions of an image will have similar lighting, as opposed to the image as a whole, which may have dramatically different lighting for each region.

- **What about the size of neighborhood?**

- Choosing the size of the pixel neighborhood for local thresholding is **absolutely crucial**.
- The neighborhood must be large enough to cover sufficient background and foreground pixels, otherwise the value of T will be more or less irrelevant.
- But not too large, then we completely violate the assumption that local regions of an image will have approximately uniform illumination.



Is it too much to worry about neighborhood size?

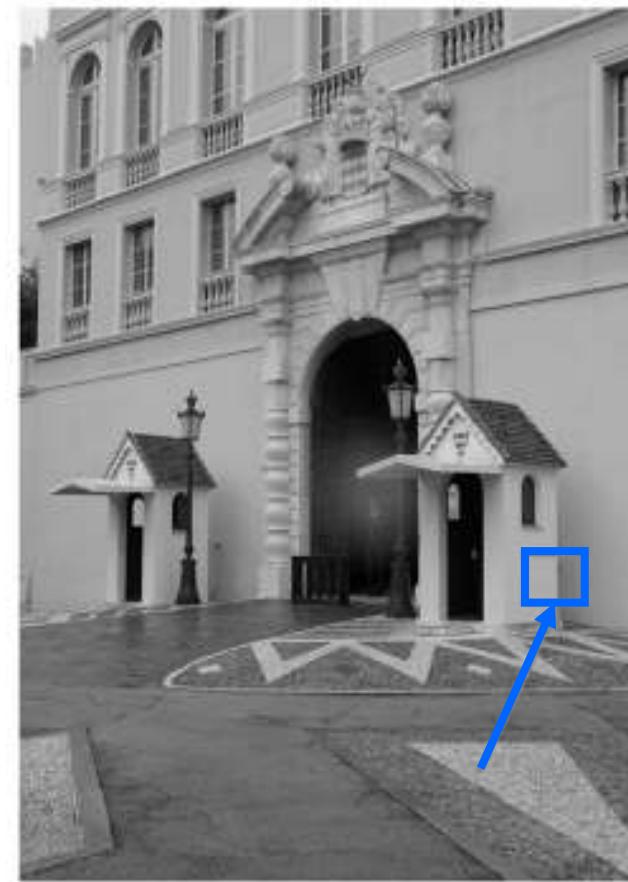
- In practice, **tuning** the neighborhood size is (usually) **not that hard** of a problem.
- You'll often find that there **is a broad range of neighborhood sizes** that provide you with **adequate** results
- It's **easier than finding an optimal value of T**, which if it is not correct, could make or break your thresholding output.

Image Gradients

Edge Detection

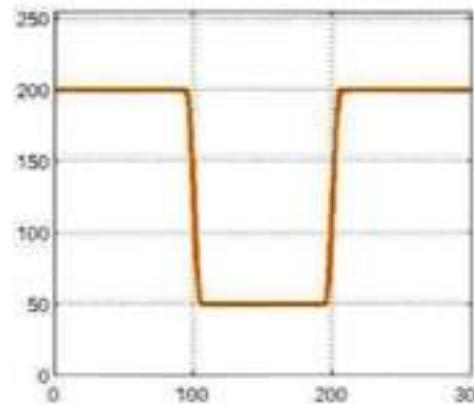
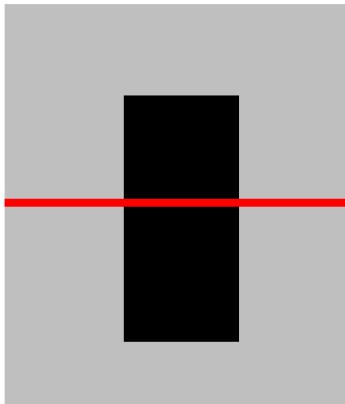
Edge Detection

- The aim is to find the **boundaries** of objects within images.
- Define a **local edge** in an image to be a **transition between two regions** of significantly different intensities.
- Edge detection is a **crucial tool** particularly in the areas of **feature extraction** of image segmentation.

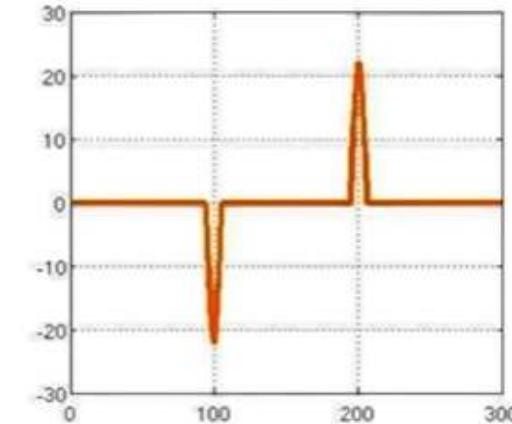


Characterizing Edges

- An edge is a place of **rapid change** in the image intensity function
- The usage of **derivatives** is a good tool for the **detection of rapid changes**.



Intensity along
horizontal line red



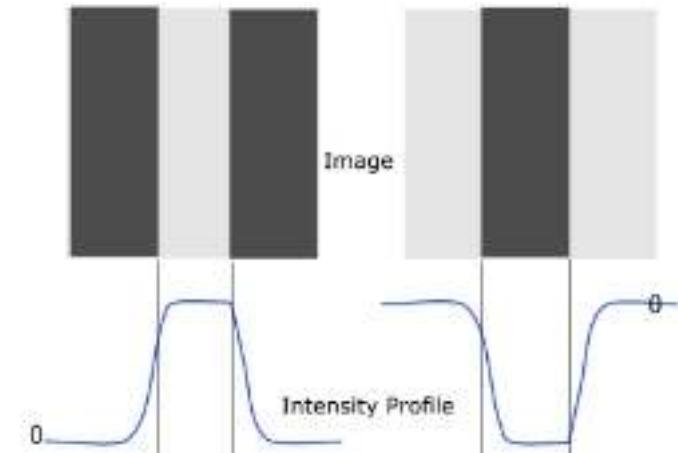
1st derivative tells us
where is an edge

Derivative measures rate of change

- High when there is a sudden change
- When there is no change it is zero

Characterizing Edges

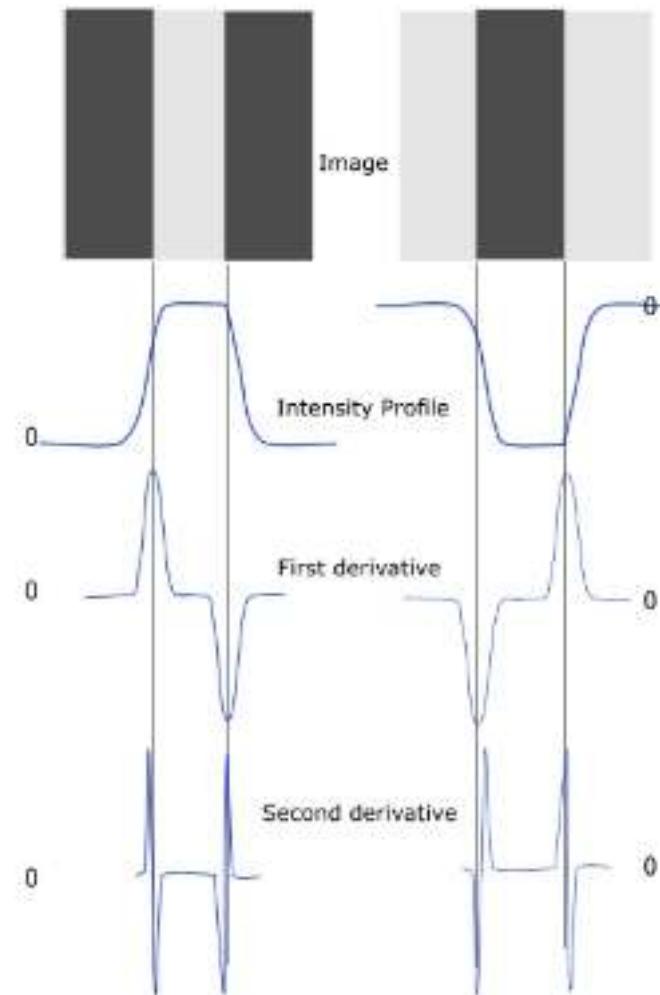
- An edge can be detected by
 1. First derivative
 2. Second derivative



How the curve look like after 1st and
2nd derivative ?

Characterizing Edges – Answer

- An edge can be detected by
 - First derivative (local maximum or minimum)
 - Second derivative (zero crossings)



Understanding Image Gradients

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

- Gradients are defined only for **continuous functions** and Image is a **2-d discrete function**.
- We need to **approximate the gradients** and we do this using **finite differences**.

Finite Difference Approximation

$$f'(x) \approx \frac{f(x + h) - f(x)}{h}$$

Understanding Image Gradients

- Three forms of finite differences are commonly used:

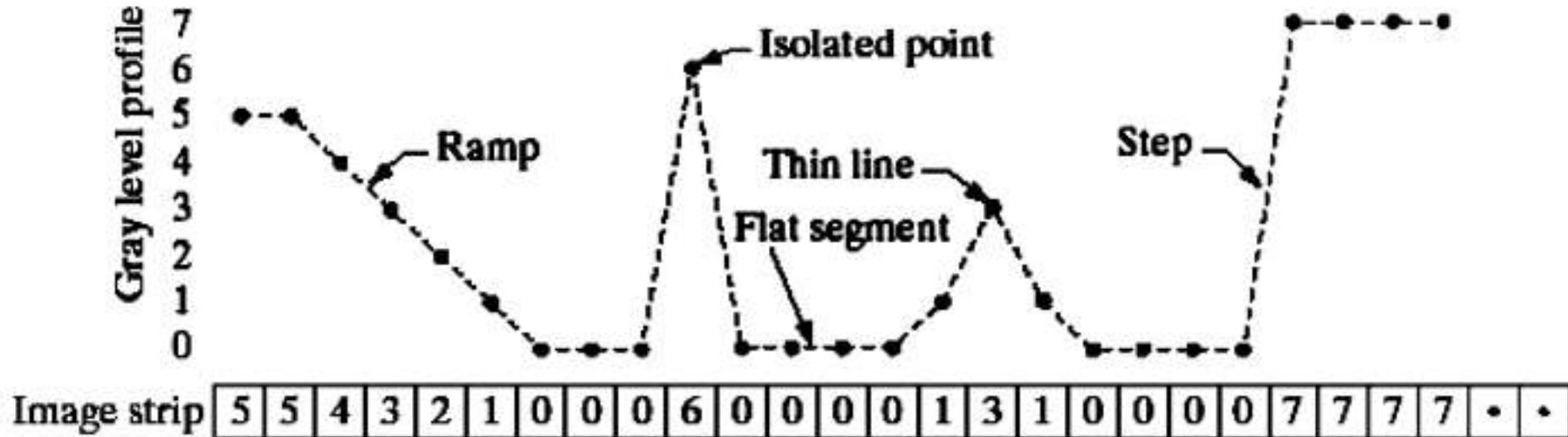
$$f'(x) \approx \frac{f(x + h) - f(x)}{h} \quad (\text{forward})$$

$$f'(x) \approx \frac{f(x) - f(x - h)}{h} \quad (\text{backward})$$

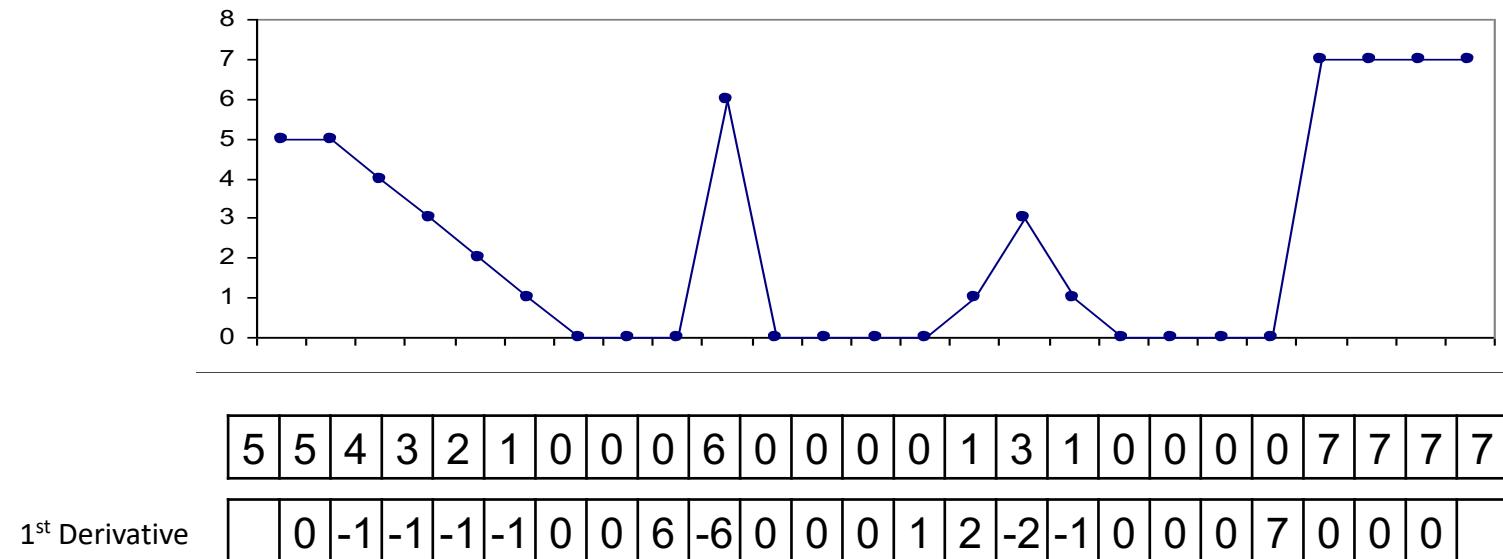
$$f'(x) \approx \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \quad (\text{central})$$

- For calculating Image Gradients, we use the central difference to approximate gradients in x and y directions.

Understanding Image Gradients



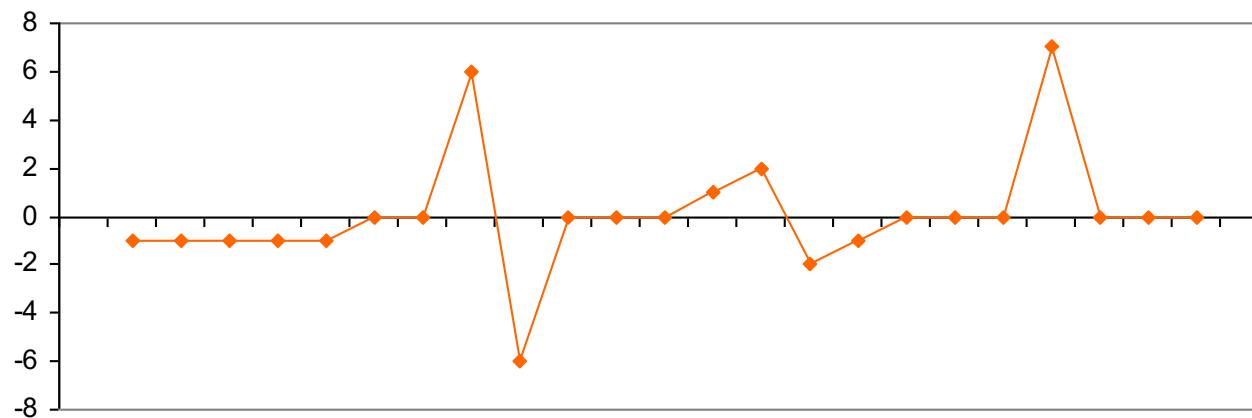
First Derivative



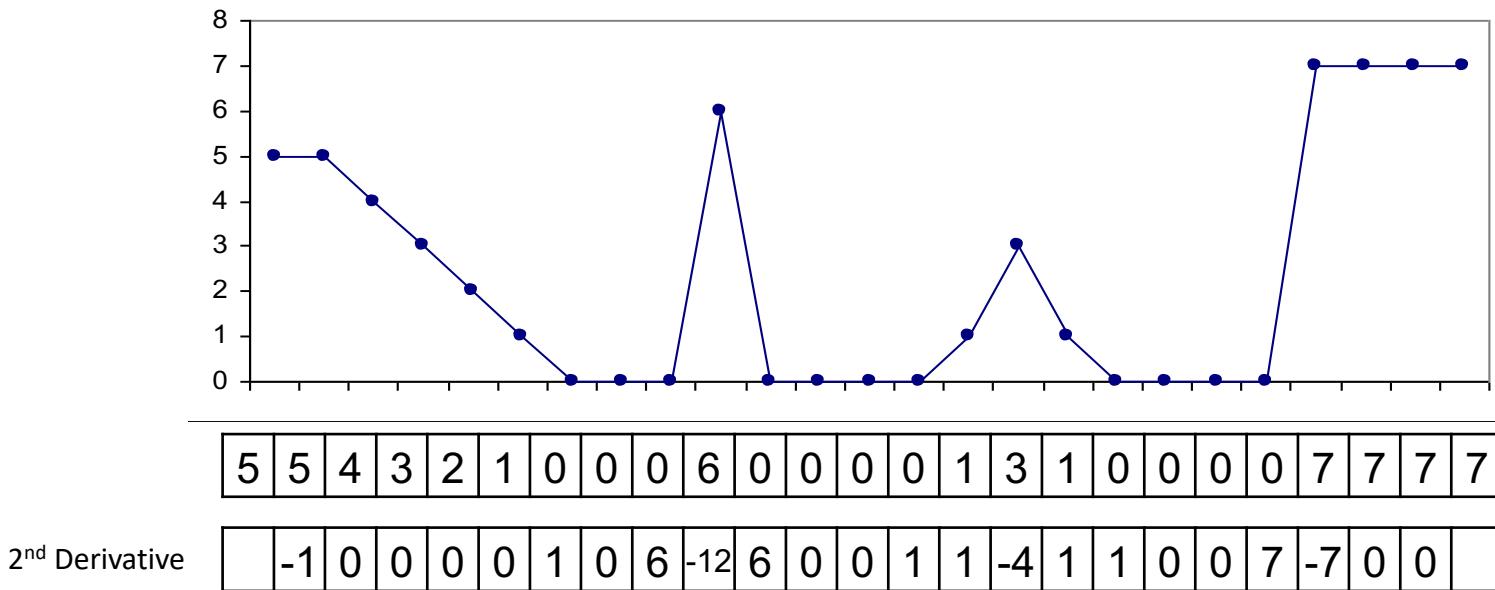
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Approximation by finite differences ($h = 1$)

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



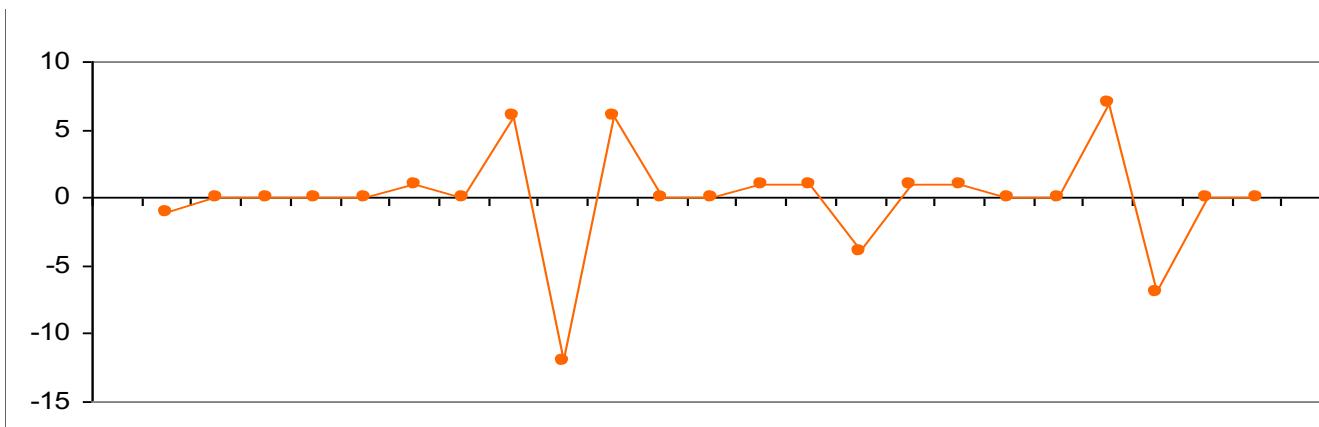
Second Derivative



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

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

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



Understanding Image Gradients

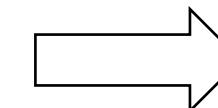
- For edge detection, we will prefer the central difference

$$f'(x) \approx \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

For discrete signals
set $h = 2$

- We can obtain the derivative filter in x and y directions

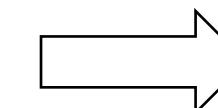
$$\frac{\partial f}{\partial x} = \frac{f(x + 1, y) - f(x - 1, y)}{2}$$



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

x-derivative

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



1
0
-1

y-derivative

The Sobel Filter

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

Sobel filter

a derivative filter
(with some smoothing)

$$= \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} *$$

What filter
is this?

Blurring/Smoothing

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

1D derivative
filter

The Sobel Filter

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

=

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

*

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

Horizontal

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

=

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

*

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

Vertical

Computing Image Gradients

- Select your favorite derivative filters

$$S_x = \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad S_y = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

- Convolve with the image to compute derivatives

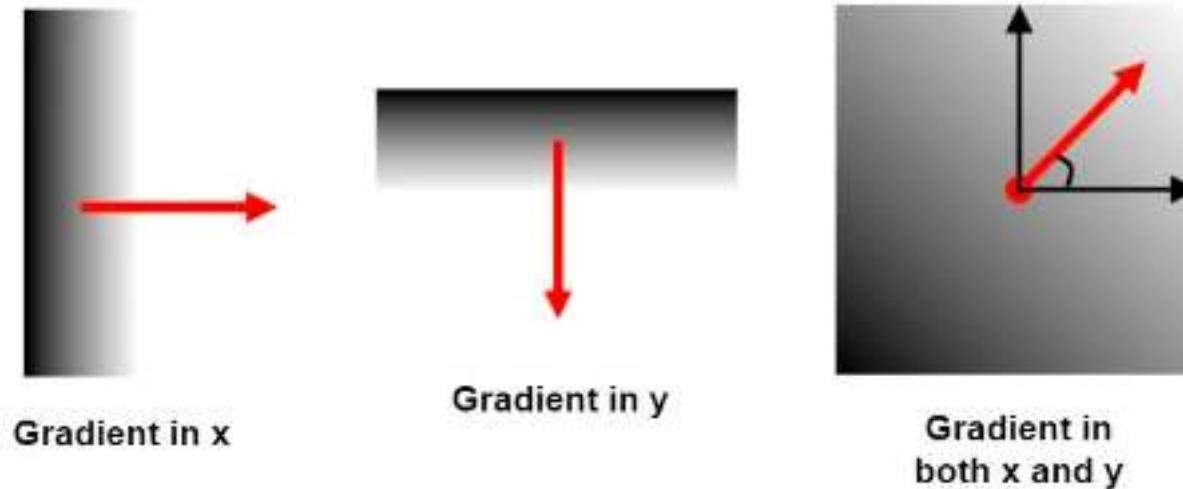
$$\frac{\partial f}{\partial x} = S_x \otimes f \quad \frac{\partial f}{\partial y} = S_y \otimes f$$

- Form the image gradient and compute its direction and amplitude.

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \quad \theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \quad \| \nabla f \| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$

Computing Image Gradients

- Image gradient in **x**-direction measures the horizontal change in intensity
- Image gradient in **y** measures the vertical change in intensity.



- We can infer the edge direction or the **angle** will be positive for the transition from dark to white and negative otherwise.

Common Edge Detectors

- Given a 3*3 region of an image the following edge detection filters can be used

Sobel

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

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

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Prewitt

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

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

Roberts

0	1
-1	0

1	0
0	-1

Edge Detection – Derivatives and Noise

- Derivative based edge detectors are **extremely sensitive to noise**

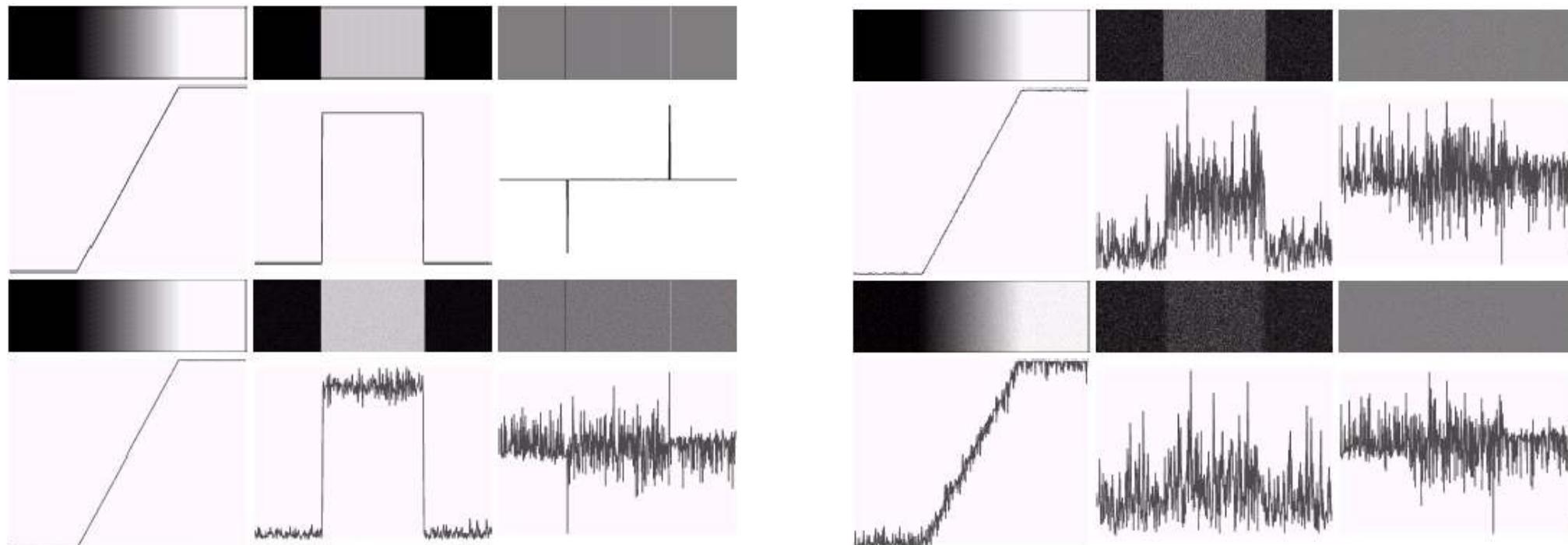
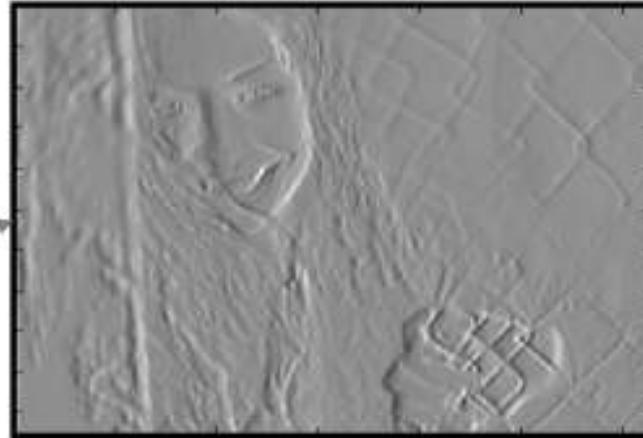


Image Gradient – Example



$$\frac{d}{dx} I$$



$$\frac{d}{dy} I$$

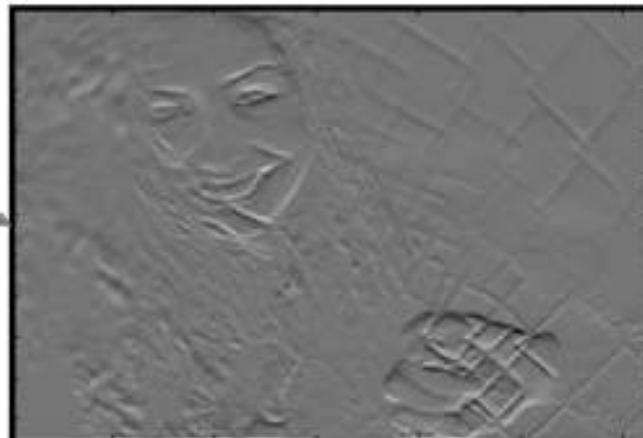
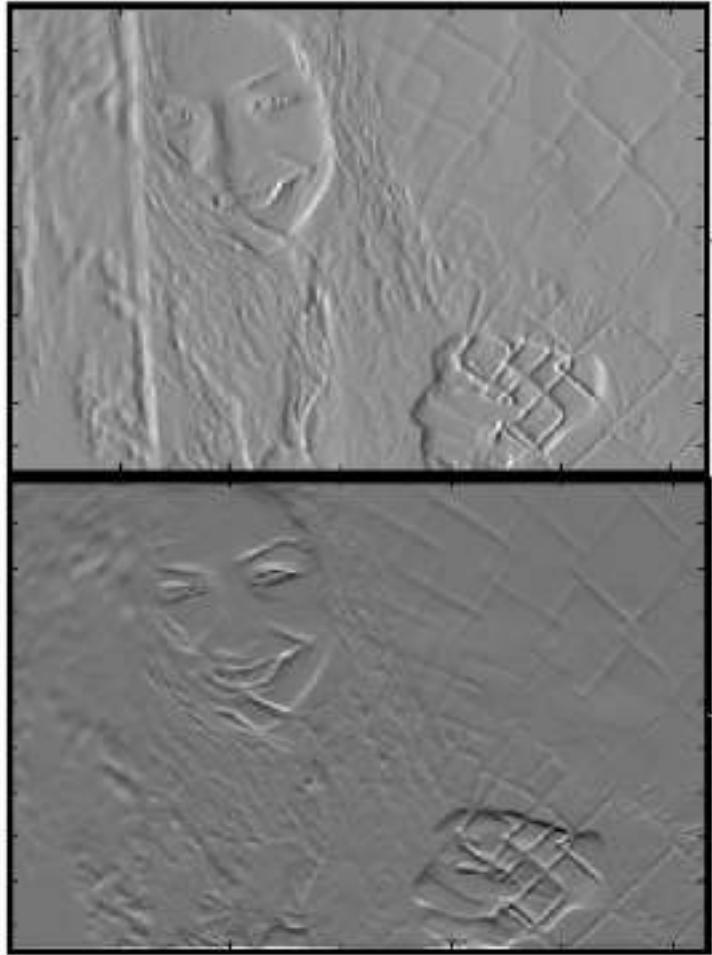


Image Gradient – Example



$$\Delta = \sqrt{\left(\frac{d}{dx} I\right)^2 + \left(\frac{d}{dy} I\right)^2}$$



Image Gradient – Example



$\Delta \geq Threshold = 100$

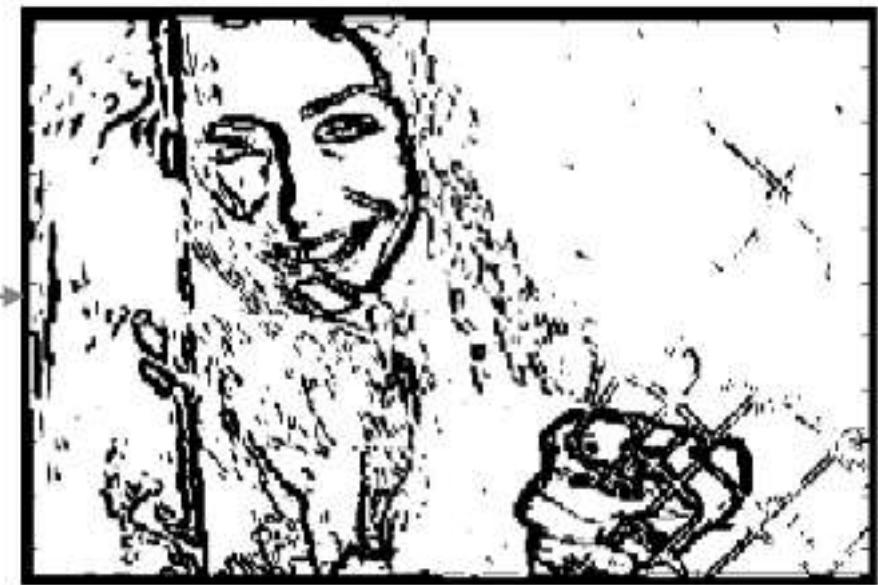


Image Gradient – Example

Final Result:



Original Image

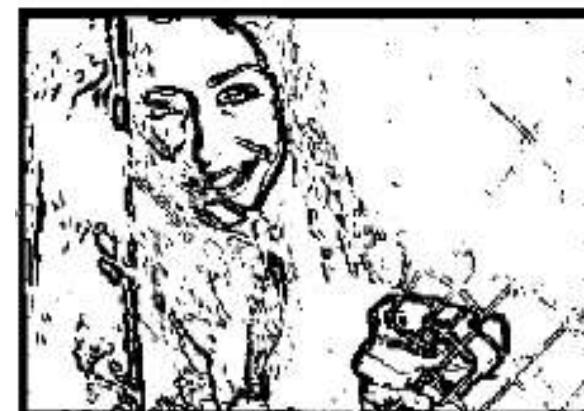


Image Gradient – Edge detection

The Laplacian Filter

- The simplest isotropic derivative operator is Laplacian and defined as:

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

- Where the partial derivative in the x direction is defined as follows:

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

- and in the y direction as follows:

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

Isotropic: Invariant with respect to direction

The Laplacian Filter

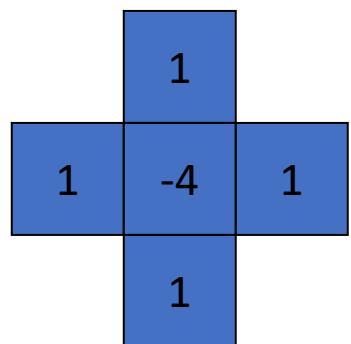
- The digital implementation of the 2-Dimensional Laplacian is obtained by summing two components

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

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

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

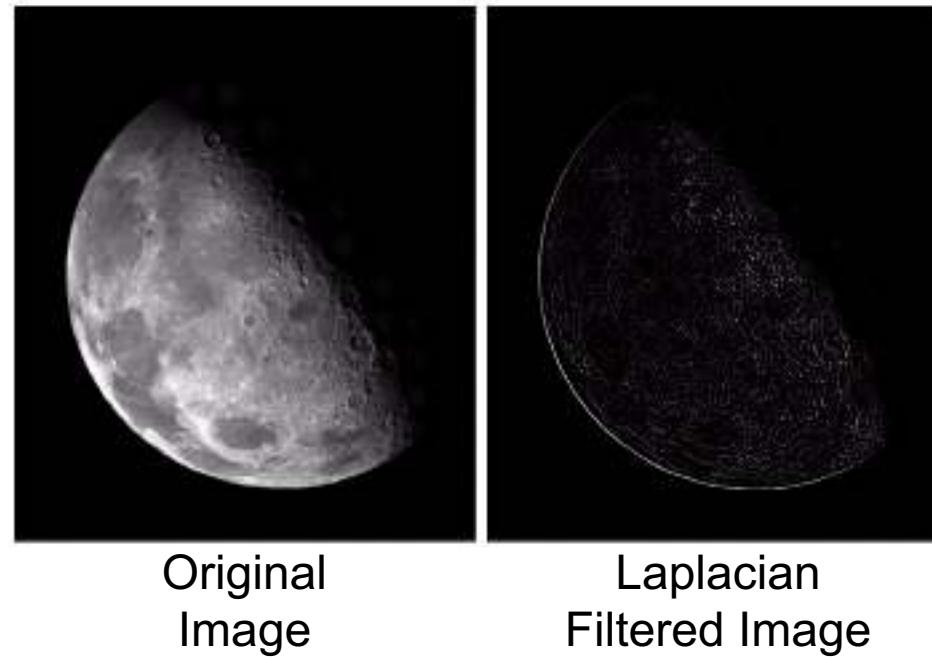
$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$



0	1	0
1	-4	1
0	1	0

The Laplacian Filter

- Applying the Laplacian to an image we get a new image that **highlights edges** and other **discontinuities**

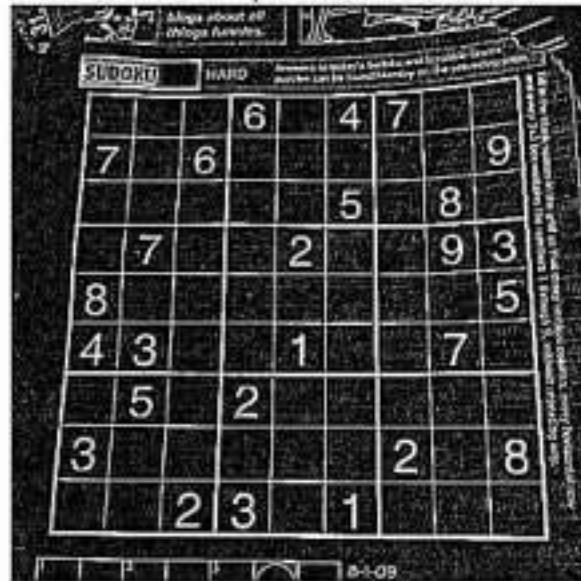


Laplacian vs. Sobel

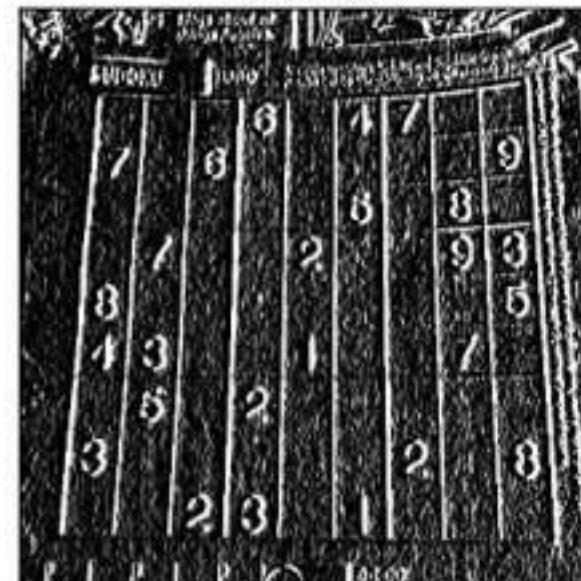
Original



Laplacian



Sobel X



Sobel Y



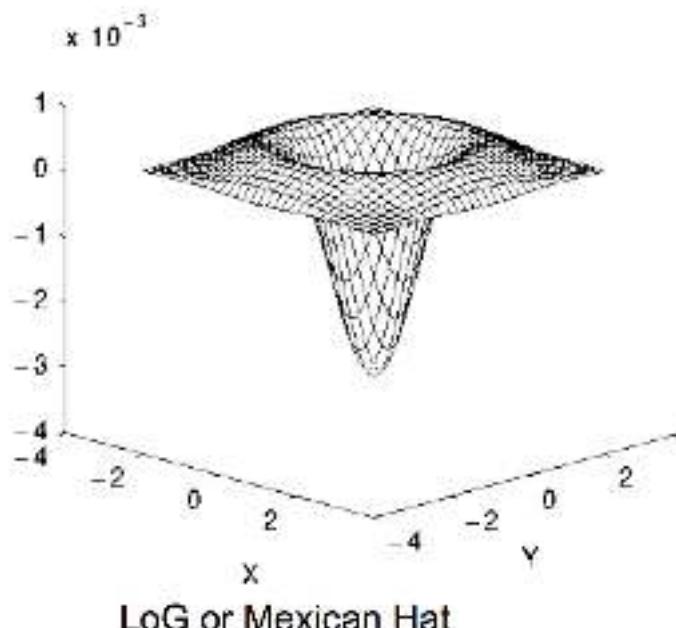
Laplacian Of Gaussian (LoG)



Laplacian Of Gaussian (LoG)

- The Laplacian of Gaussian (or Mexican hat) filter uses the Gaussian for noise removal and the Laplacian for edge detection

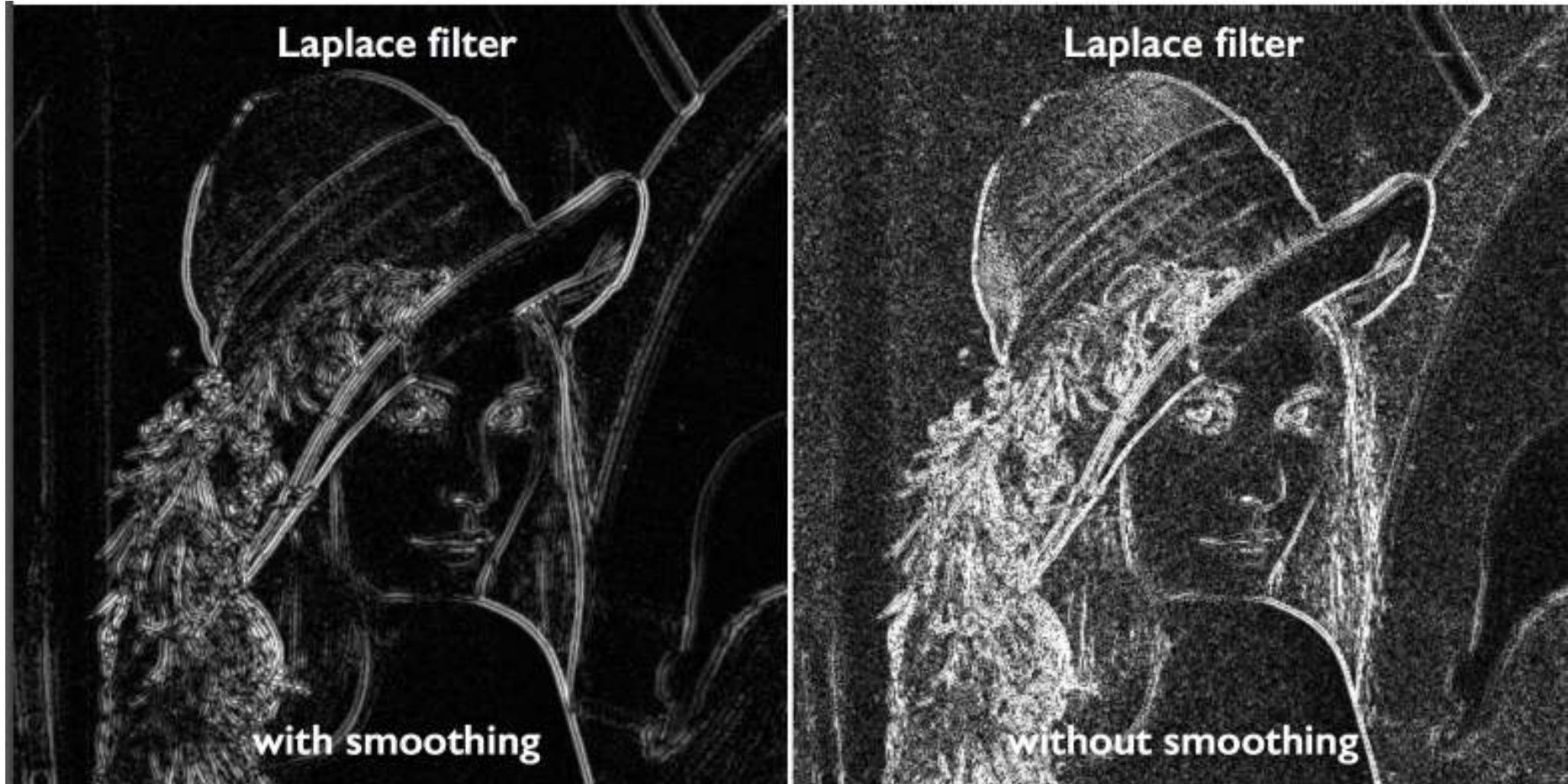
$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$



0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	9	5	4	1
2	6	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	9	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Example filter 9x9, $\sigma = 1.4$

Laplacian Of Gaussian (LoG)



Canny Edge Detector

- The Canny operator is a **multi-stage algorithm** designed to be an optimal edge detector.
- Its steps are:
 1. Gaussian convolution **smoothing** to remove noise
 2. Find the intensity gradients of the image: An edge in an image may point in a variety of directions, so the Canny algorithm uses **four filters** to detect **horizontal, vertical and diagonal edges**
 3. Edges thinning through **non-maximum suppression**.
 4. Thresholding and linking (hysteresis):
 - Define two thresholds: low and high – Use the high threshold to start edge curves and the low threshold to continue them

Canny Edge Detector



What is Morphology?



Image after segmentation



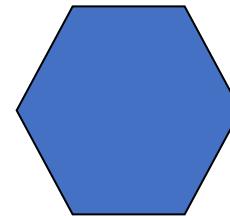
Image after segmentation and
morphological processing

What is Morphology?

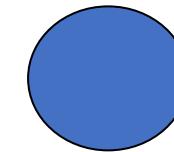
- Morphological operations are typically applied **to remove imperfections** introduced during segmentation
- It has two basic operations: **Dilation**, **Erosion**, two derived operations: **Opening**, **Closing** and a **structuring element** whose size and shape may vary
- Morphological image processing (or *morphology*) describes a range of image processing techniques that **deal with the shape of features** in an image



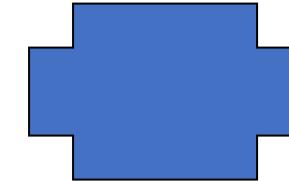
box



hexagon



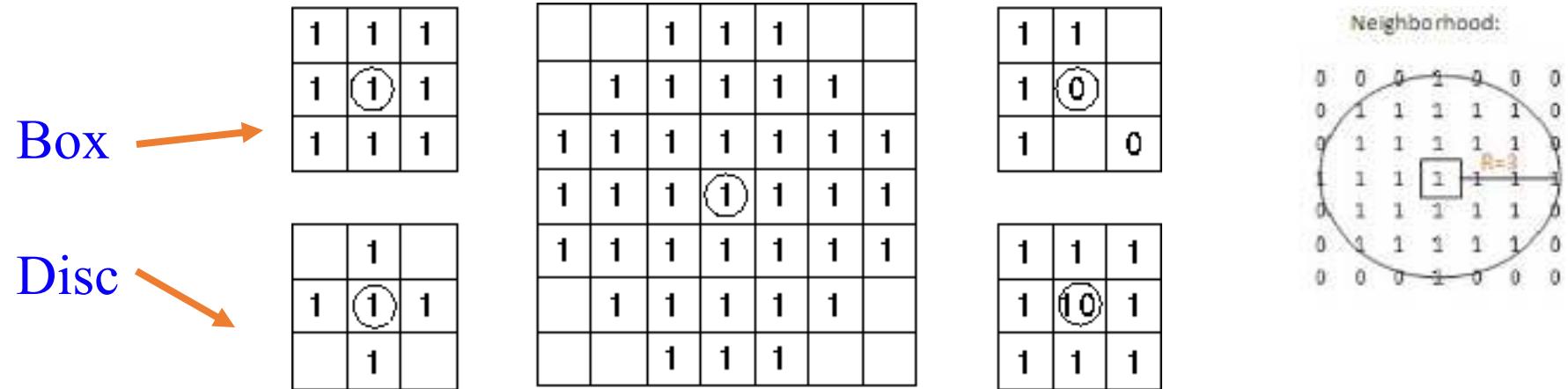
disk



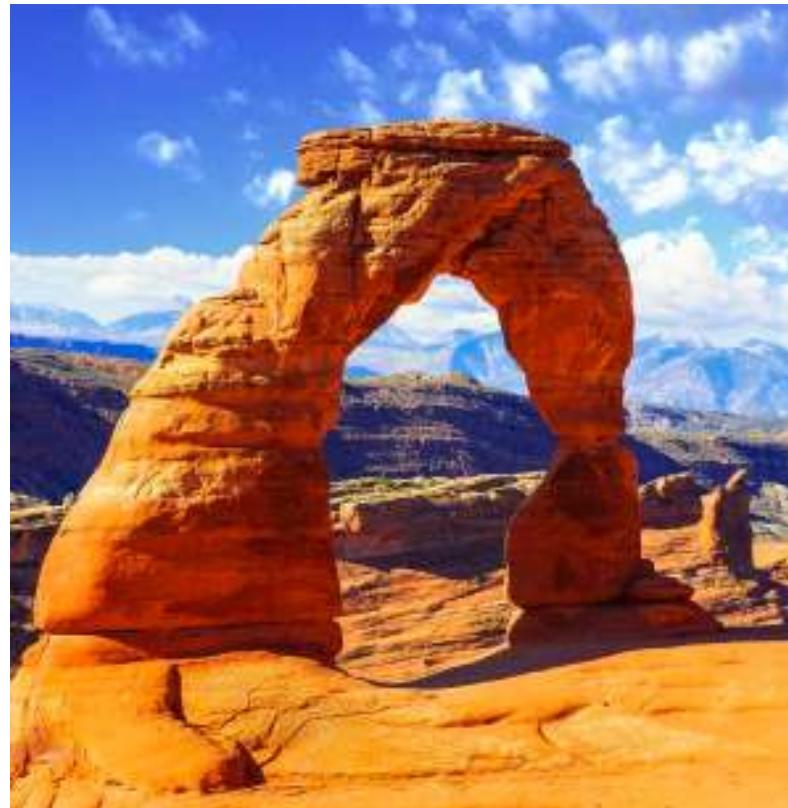
something

Structuring Element (Kernel)

- It has **varying sizes** and origin
- Element values are
 - 0,1 and none – other values are possible
 - Empty spots are *don't care's!*



Erosion



Erosion and Dilation

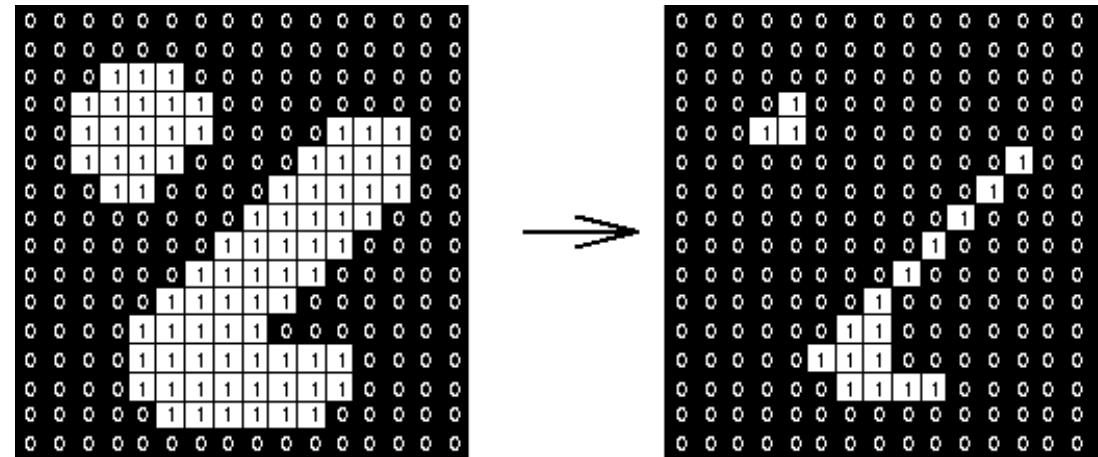
- **Erosion:** (It shrinks the object)

- Apply the *structuring element* on every pixel (i,j) of the image f
- (i,j) is the center of the *structuring element*
- If the whole structuring element is included in the region, then $f[i,j] = 1$
- Otherwise, $f[i,j] = 0$

- **Dilation:** (It expands the region)

- If at least one pixel of the structuring element is inside the region, then $f[i,j] = 1$

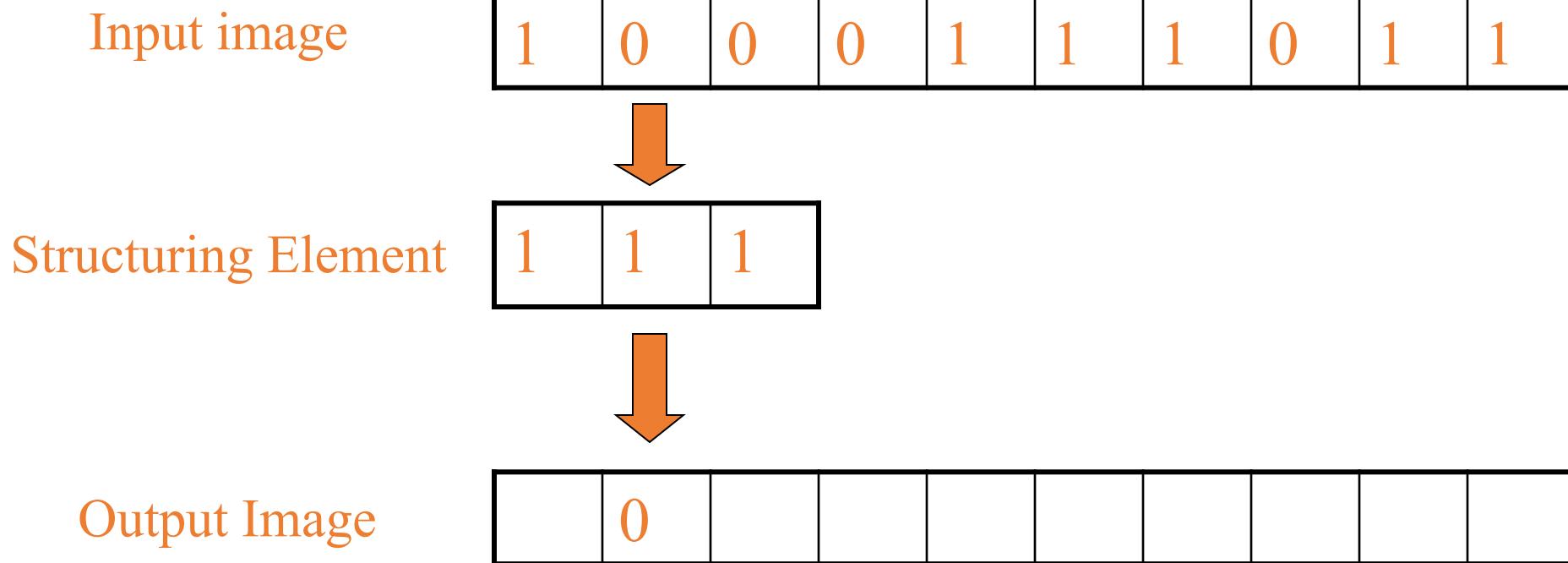
Erosion Example



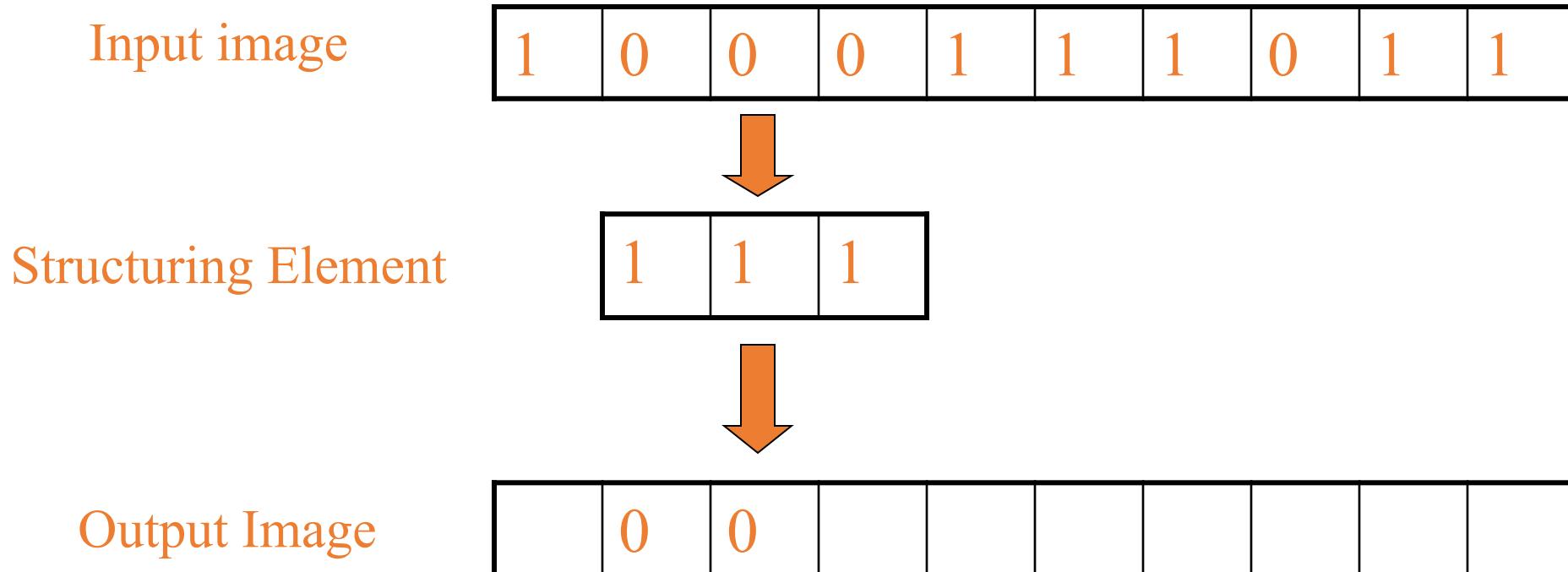
- Applied structuring element:

1	1	1
1	1	1
1	1	1

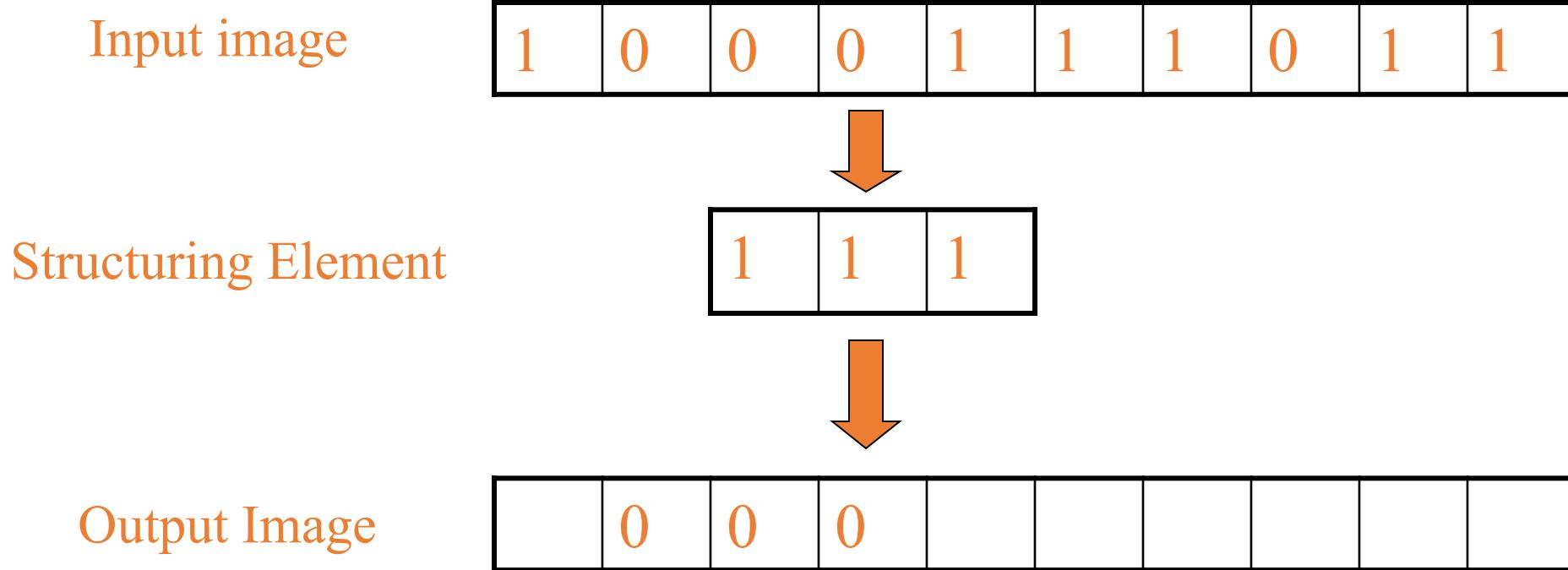
Erosion Simplified



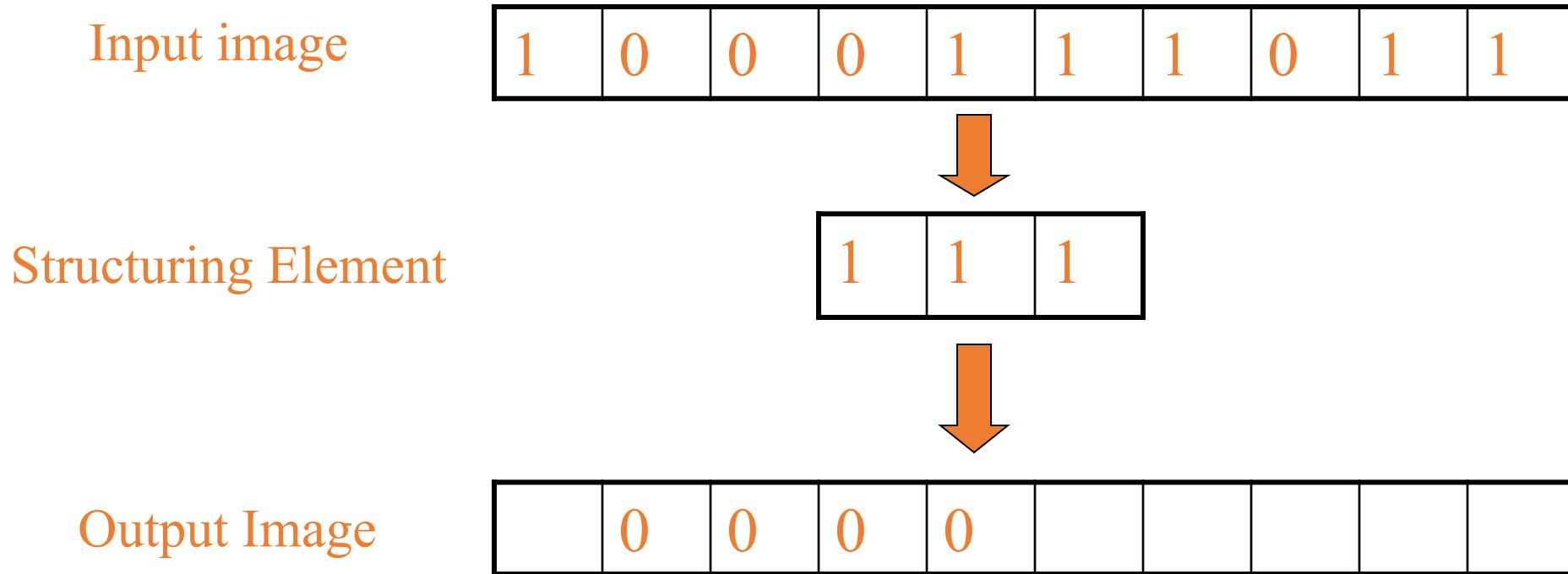
Erosion Simplified



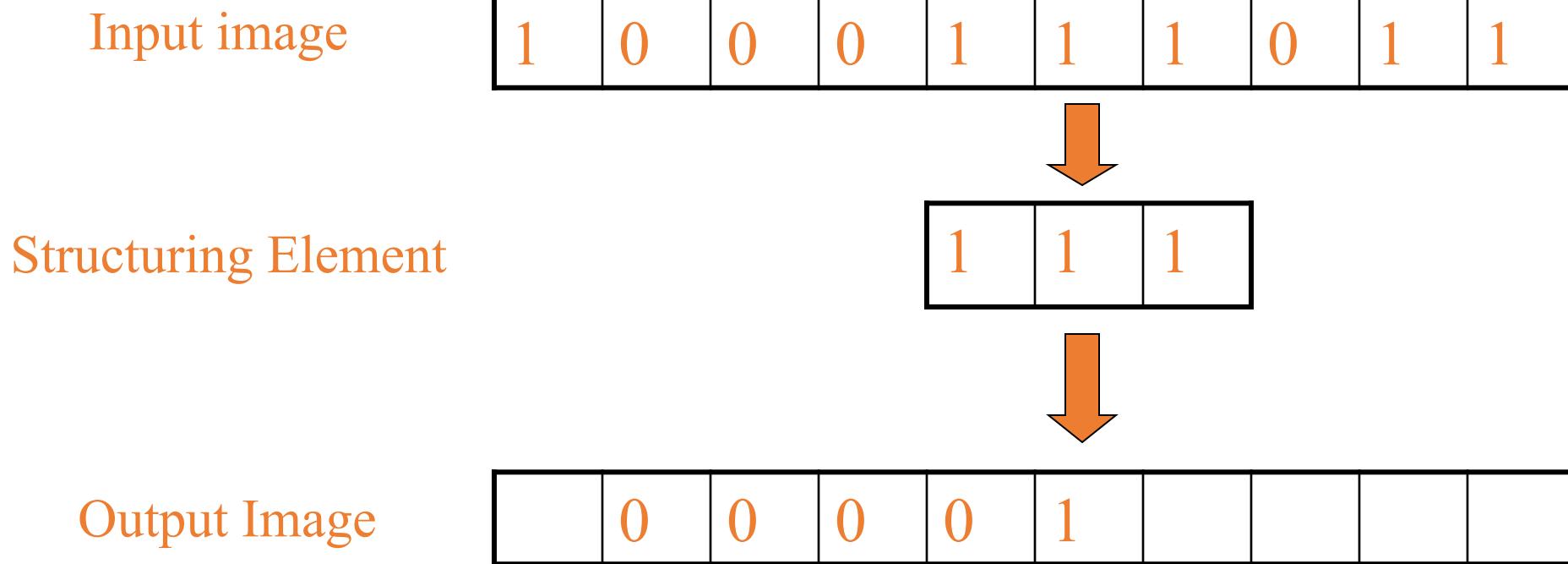
Erosion Simplified



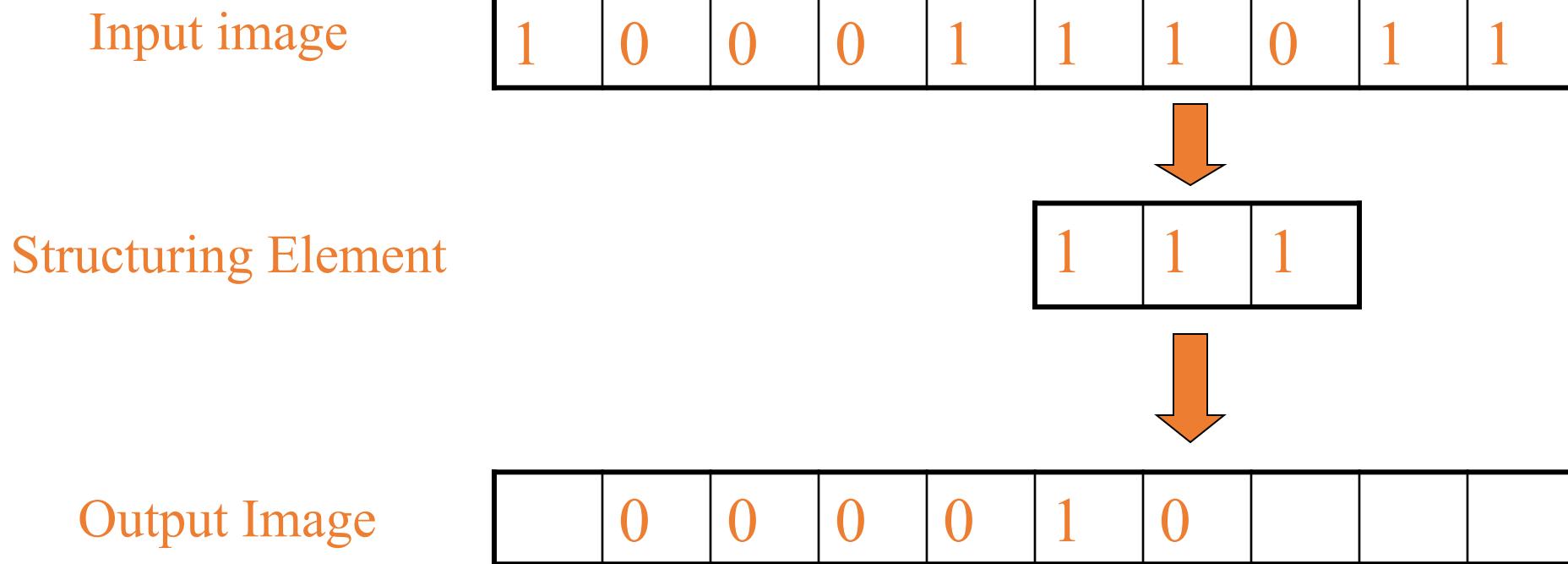
Erosion Simplified



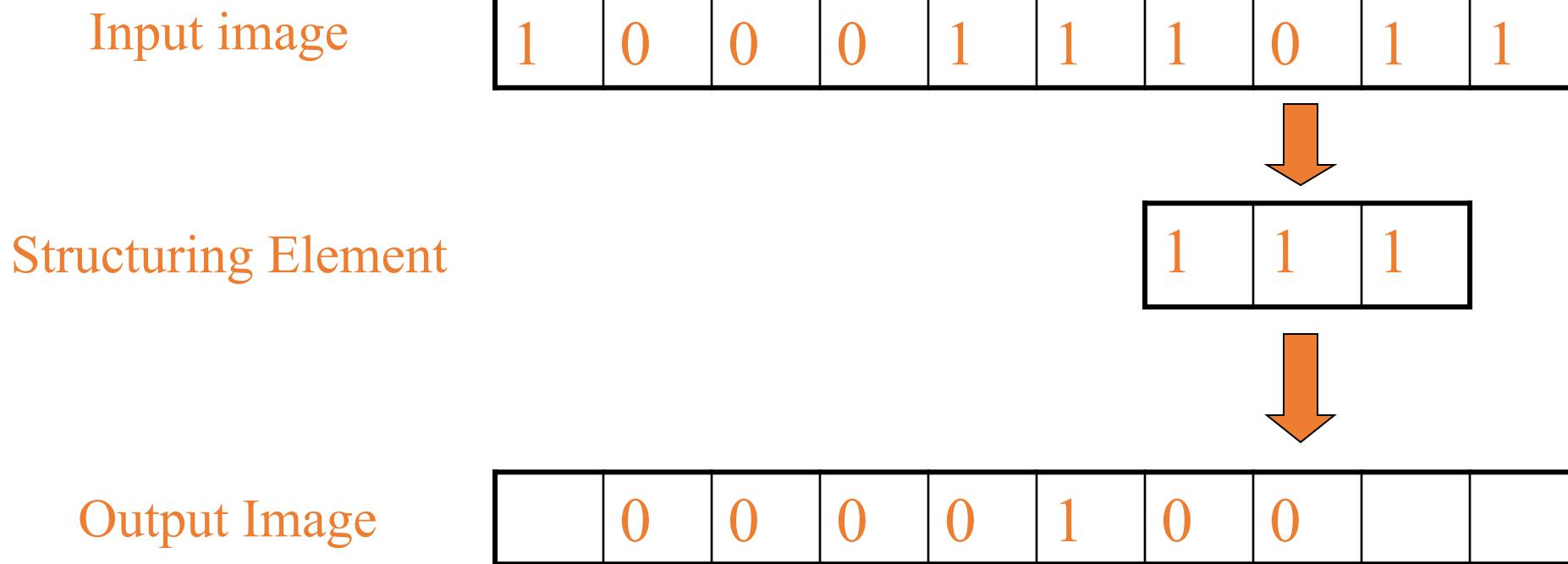
Erosion Simplified



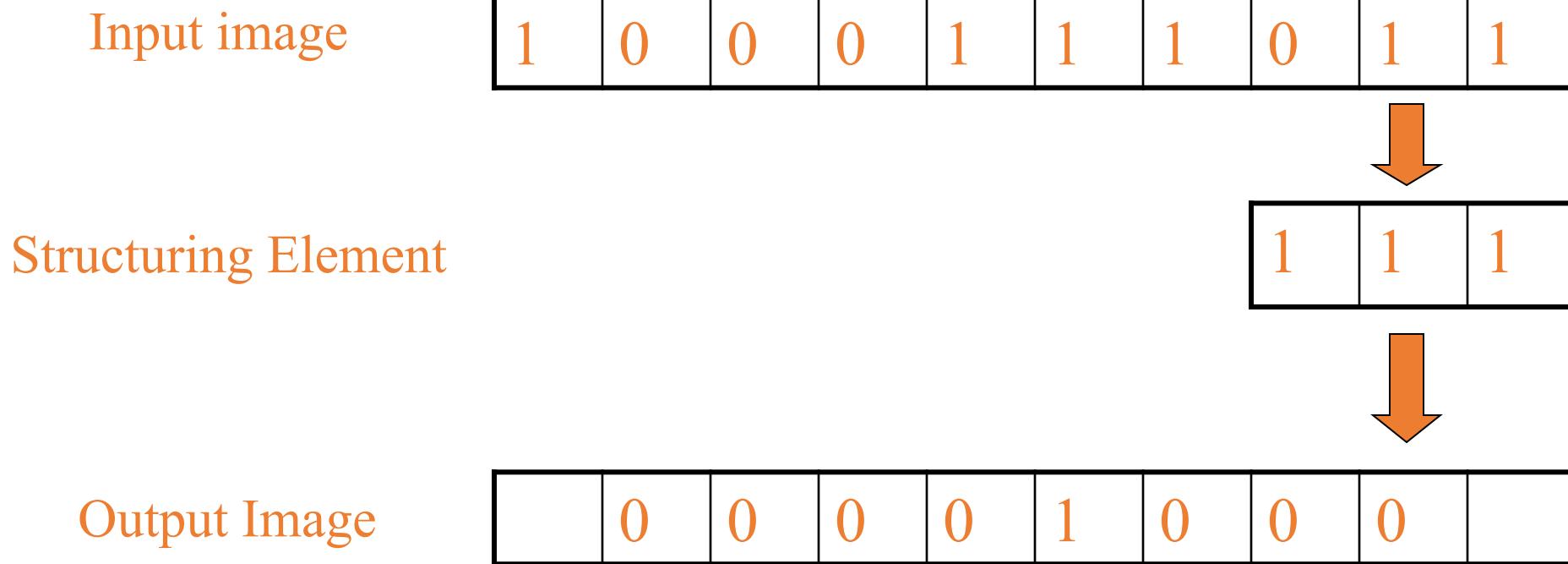
Erosion Simplified



Erosion Simplified

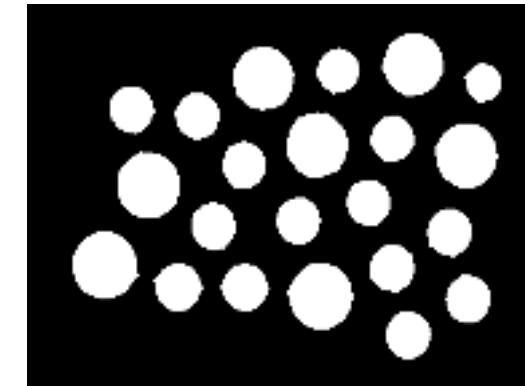
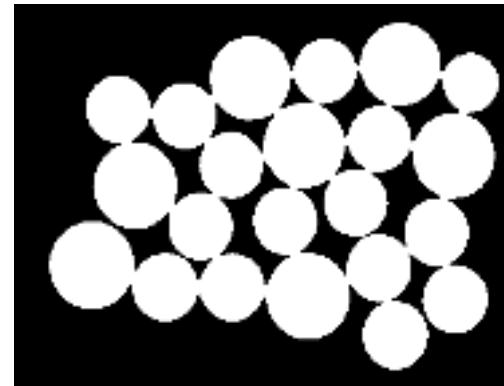
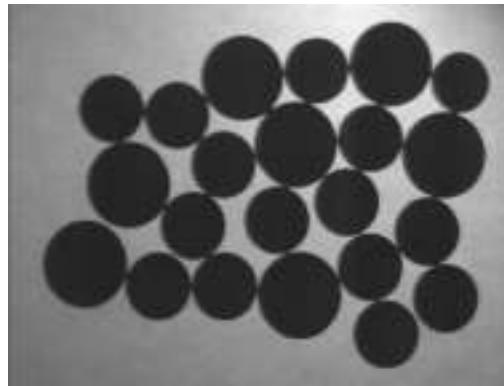


Erosion Simplified

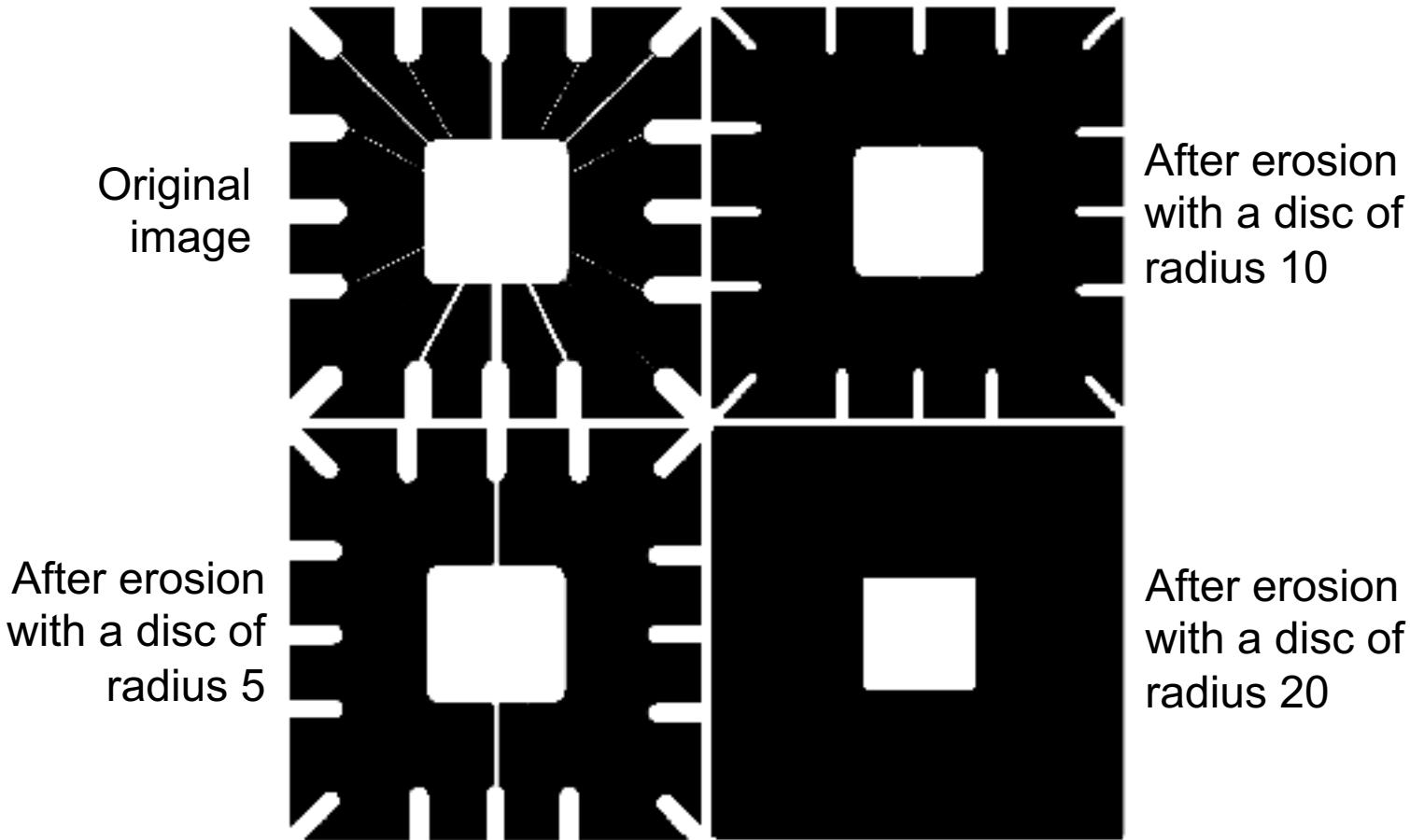


Erosion – Counting Coins

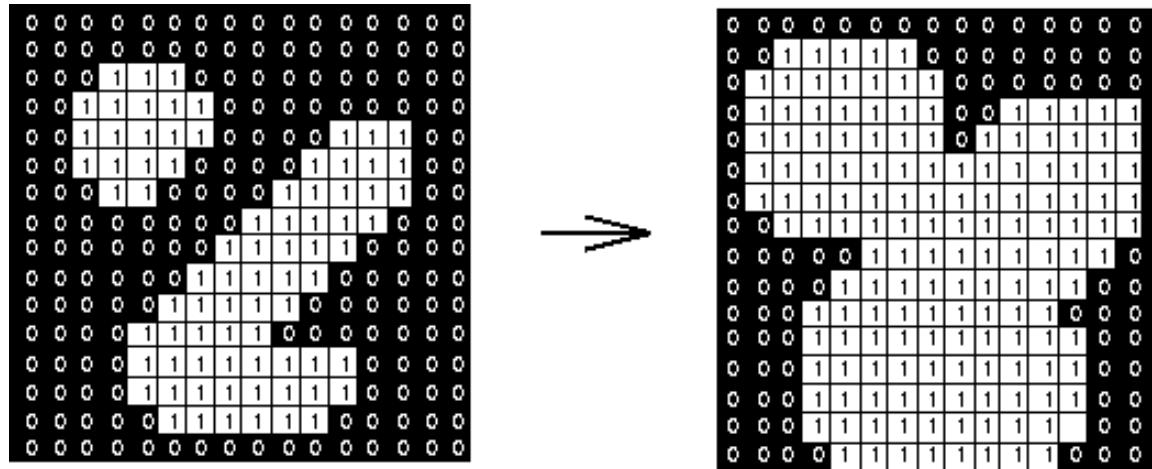
- Counting coins is difficult because they touch each other!
- **Solution:** Binarization and Erosion separates them!



Erosion Example II



Dilation Example

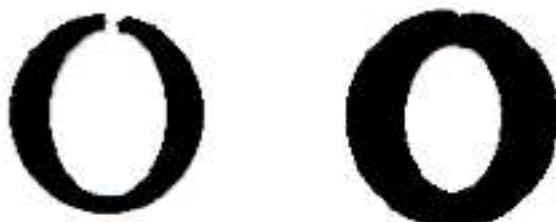


- Applied **structuring element**:

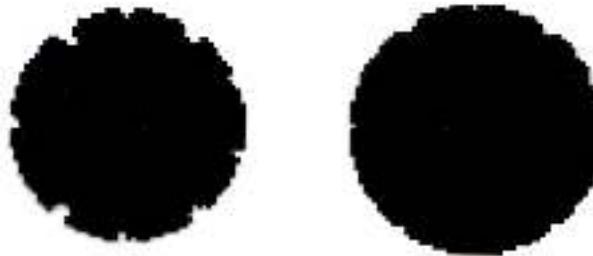
1	1	1
1	1	1
1	1	1

What is Dilation For?

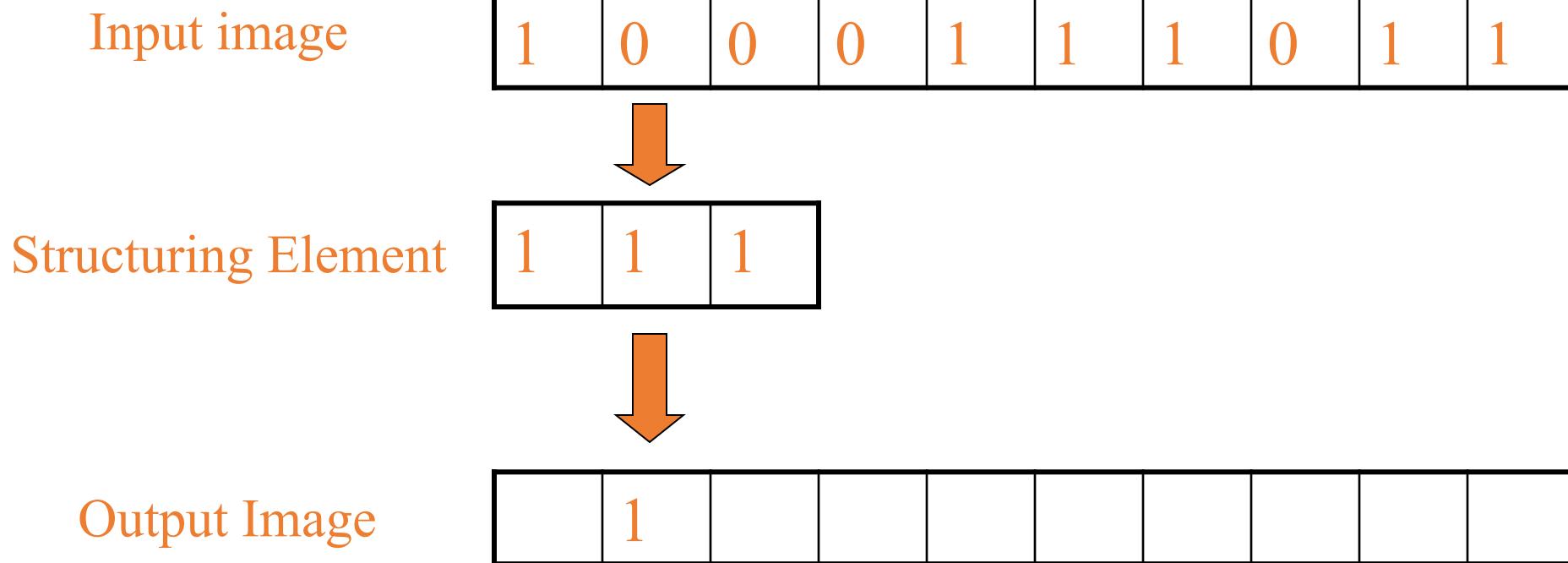
- Dilation can repair breaks



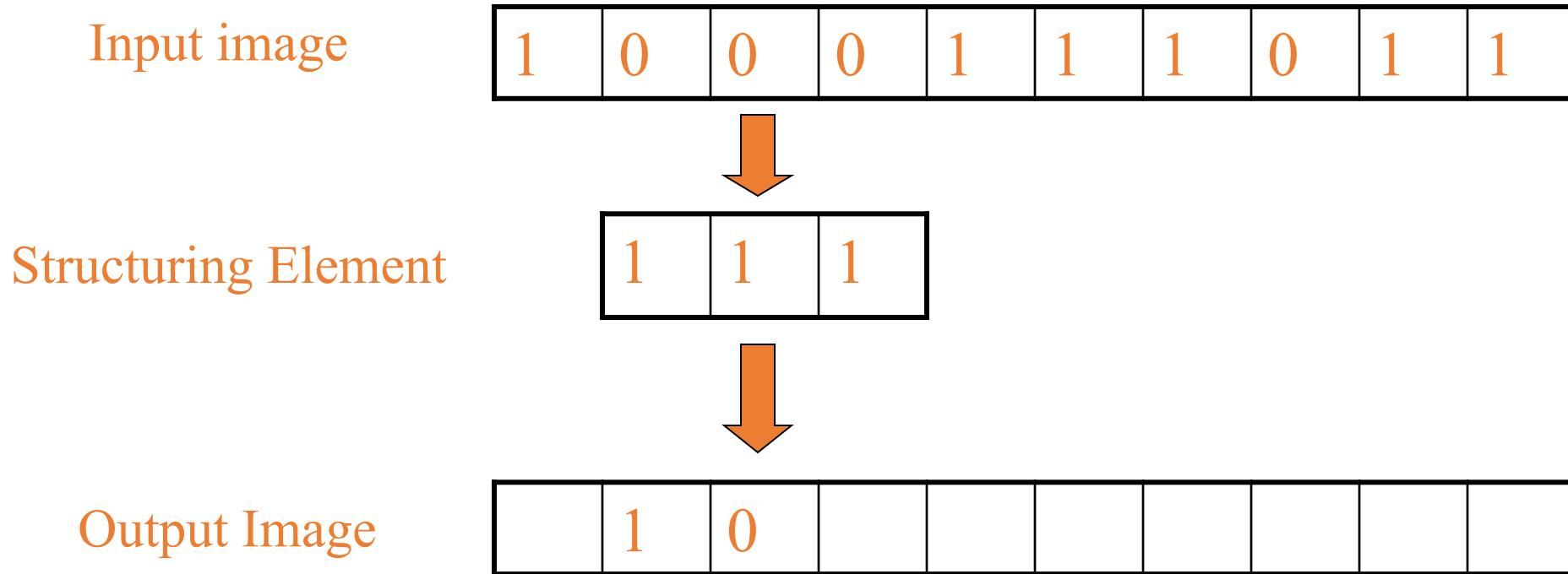
- Dilation can repair intrusions



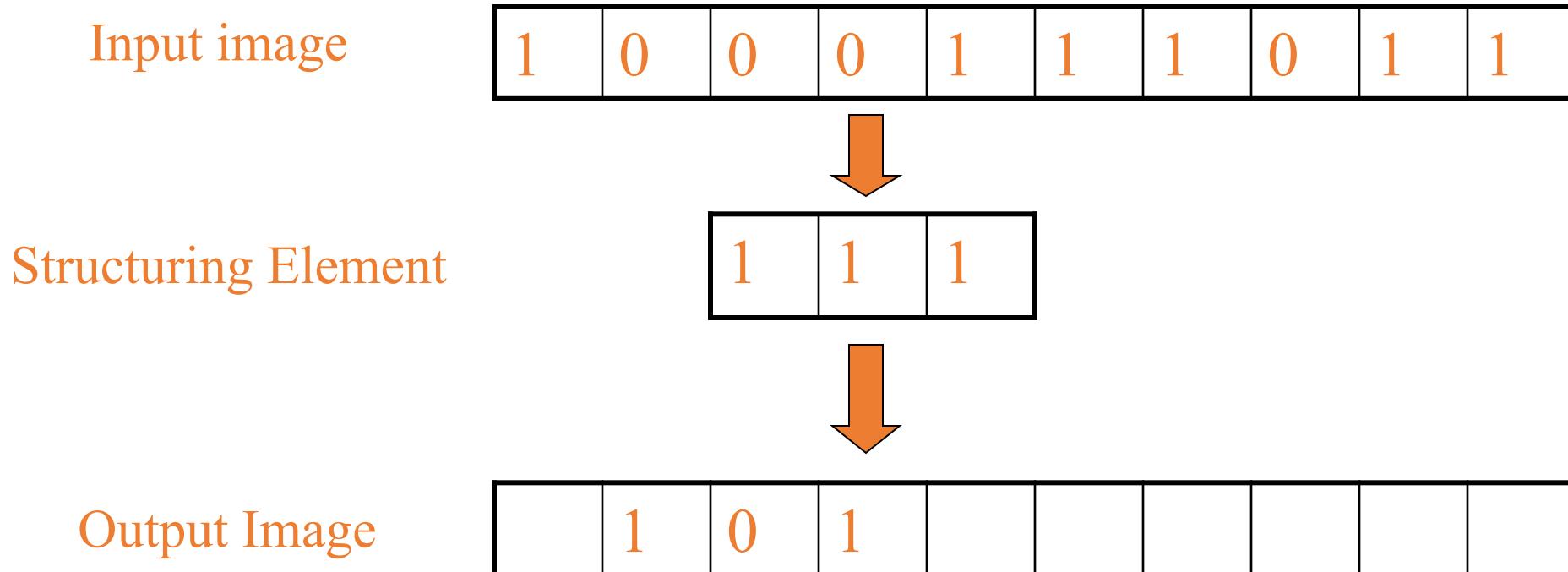
Dilation Simplified



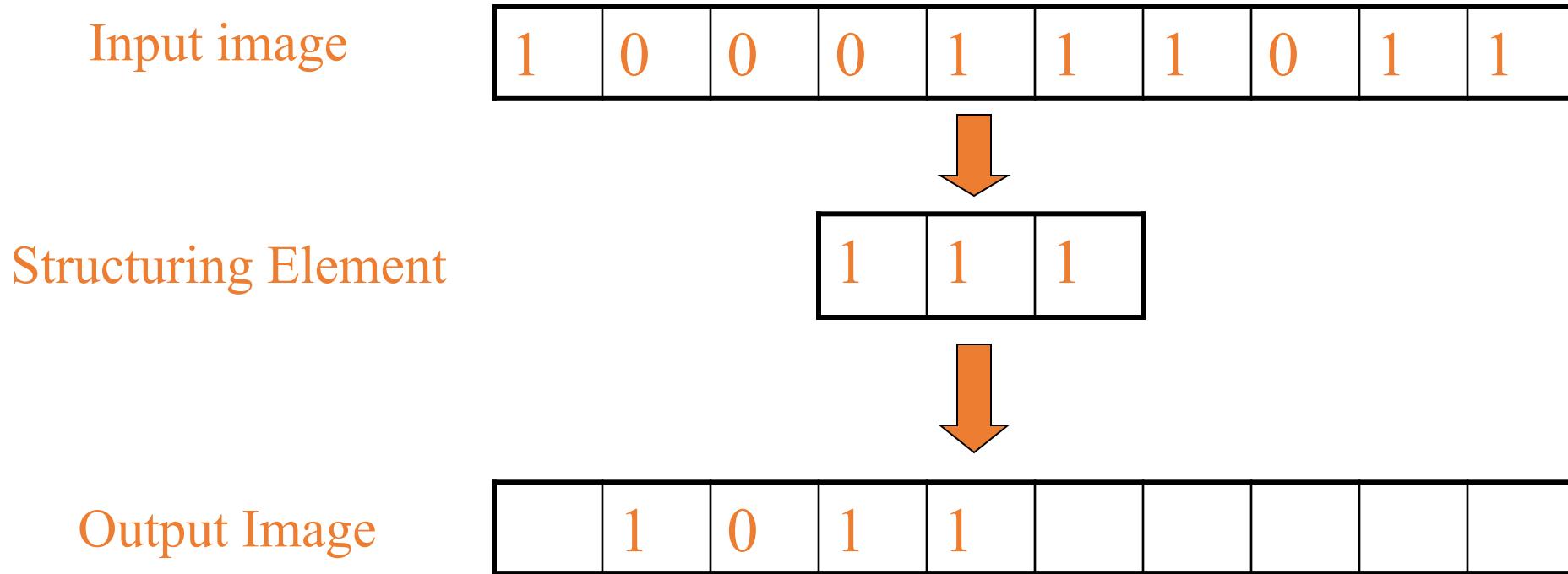
Dilation Simplified



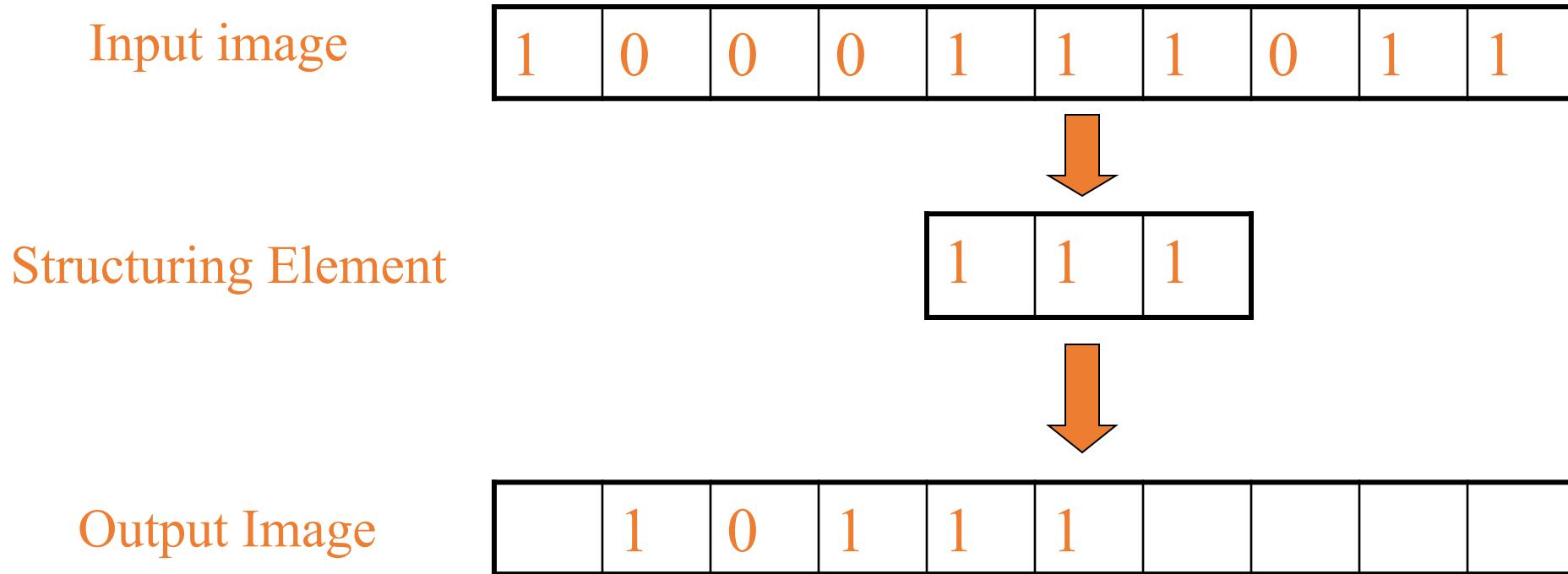
Dilation Simplified



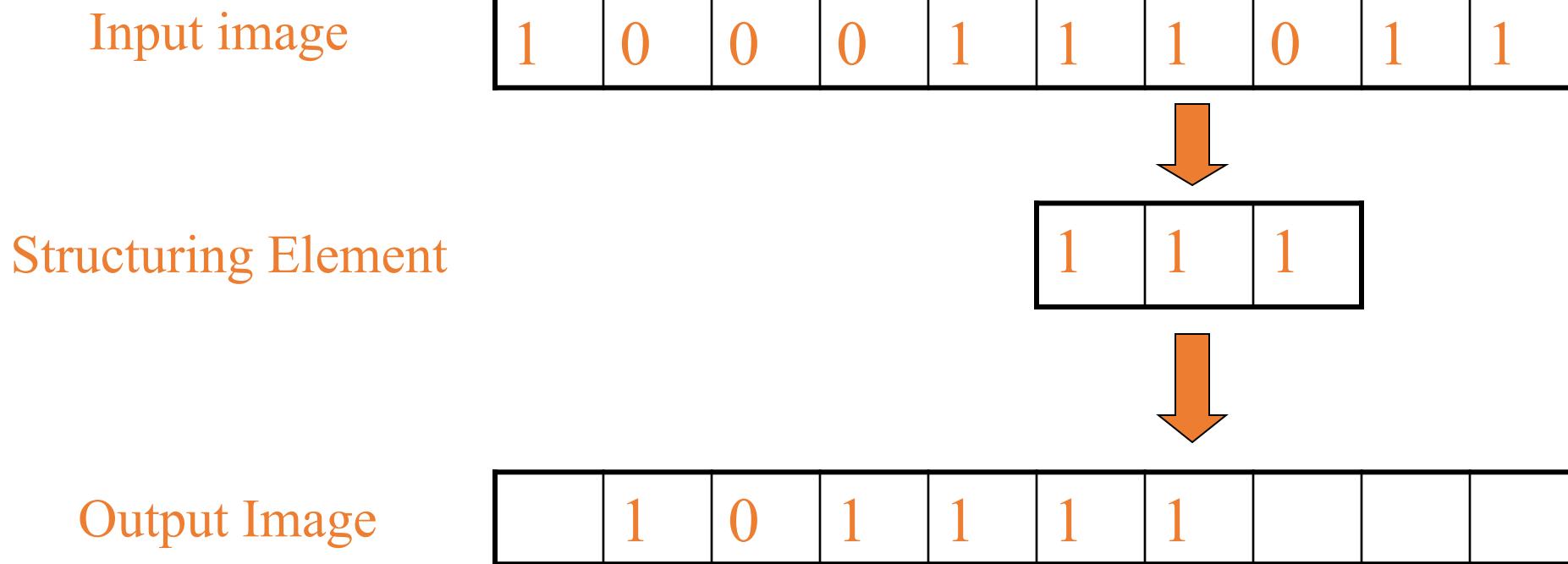
Dilation Simplified



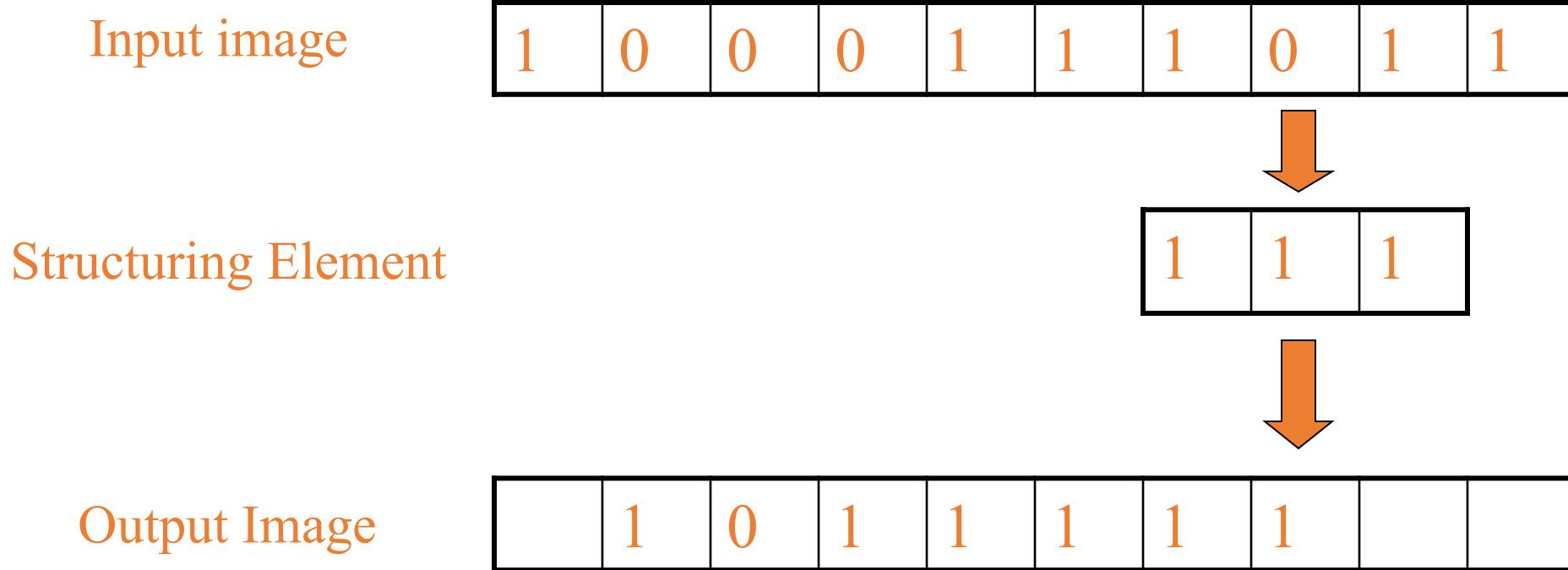
Dilation Simplified



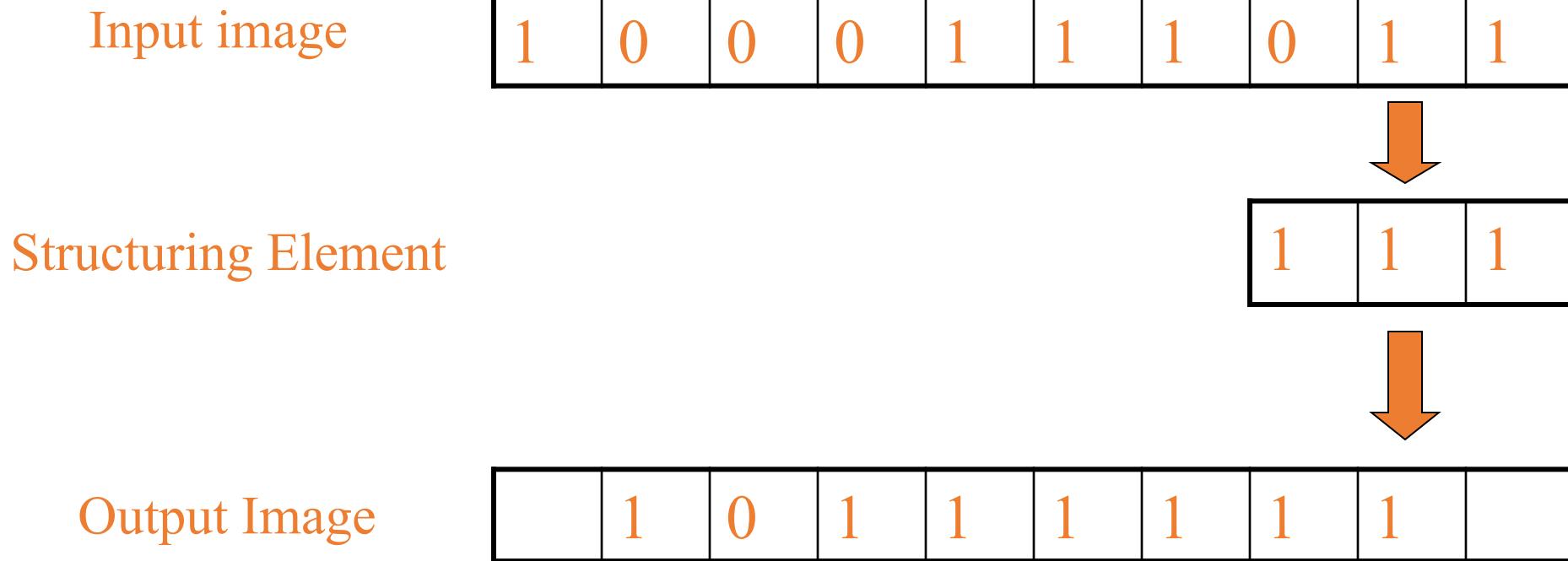
Example for Dilation



Dilation Simplified



Dilation Simplified



Dilation Example

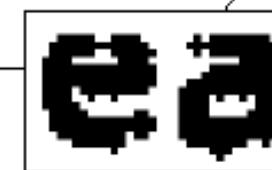
Original image

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



After dilation

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



0	1	0
1	1	1
0	1	0

Structuring element

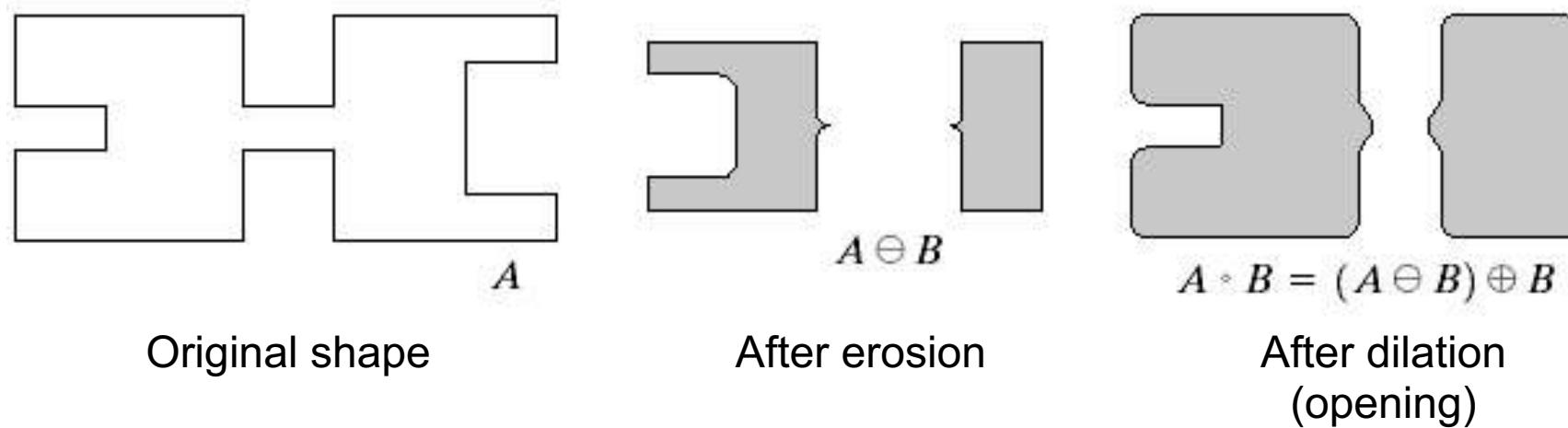
Compound Operations

- More interesting morphological operations can be performed by performing **combinations of erosions and dilations**
- The most widely used of these *compound operations* are:
 - Opening
 - Closing

Opening

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

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



Note: A disc shaped structuring element is used

Opening

Original
Image



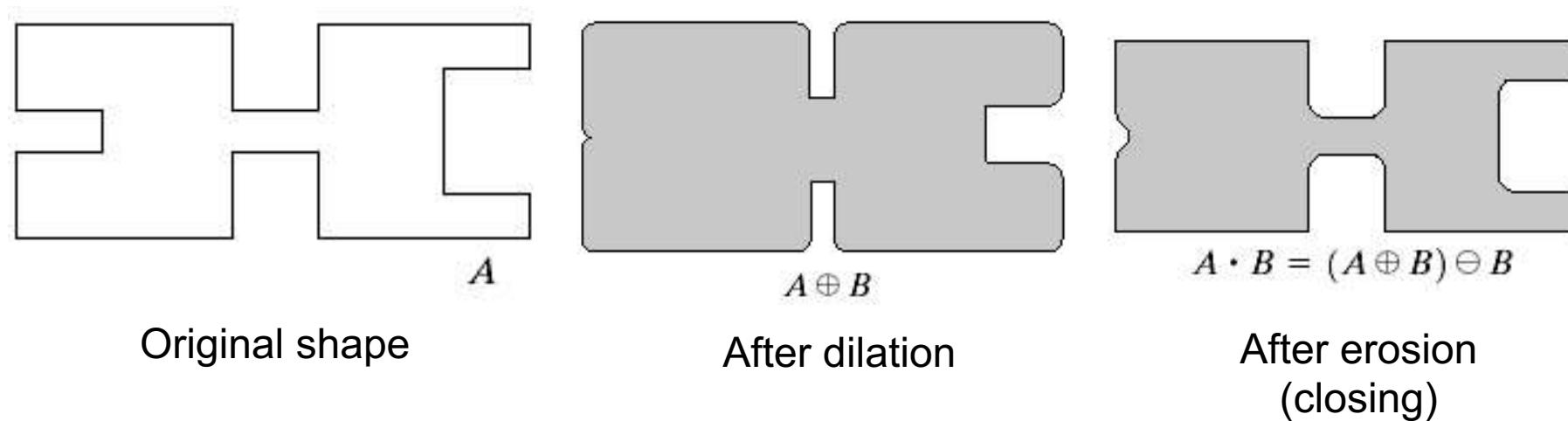
Image
After
Opening



Closing

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

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



Closing Example

Original
Image

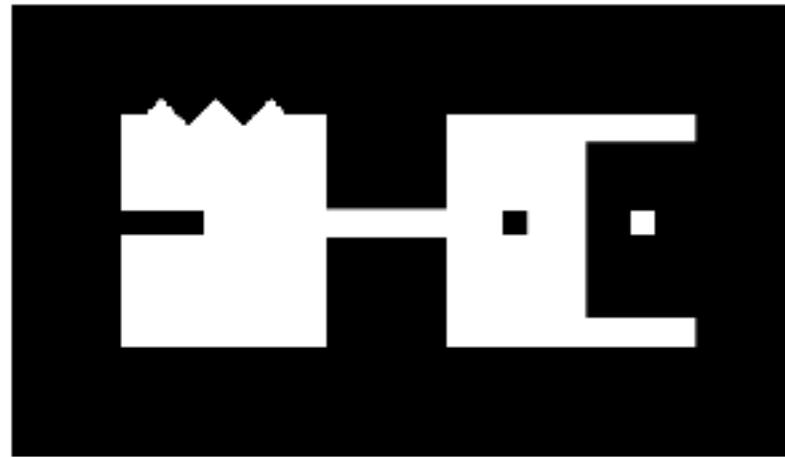


Image
After
Closing



Closing Example

1. Threshold
2. Closing with disc of size 20

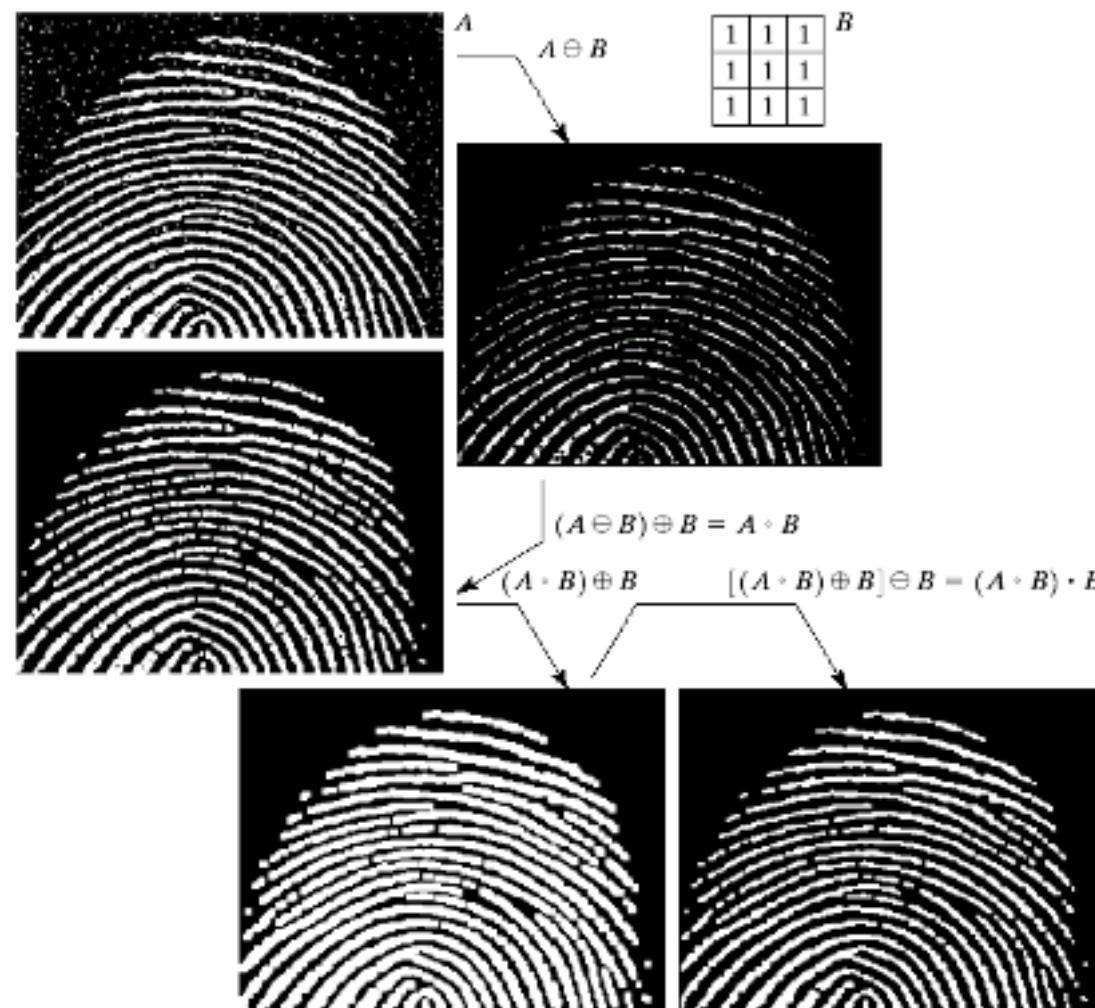


Thresholded



closed

Morphological Processing Example

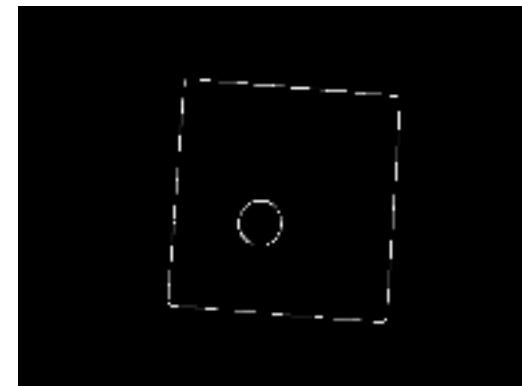
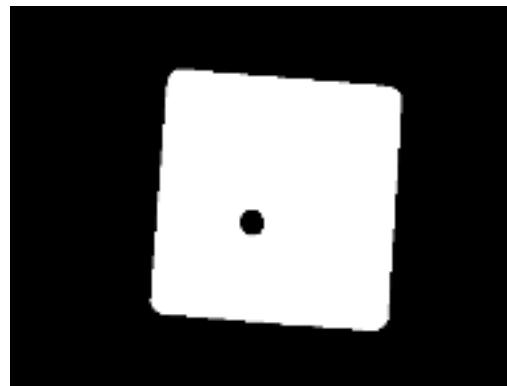
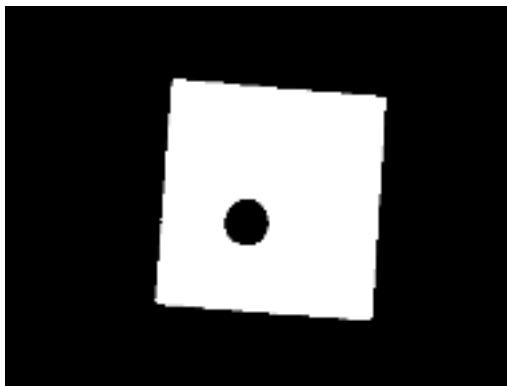


Morphology for Boundary Extraction

- The *boundary* of a set A , denoted as $\beta(A)$, can be obtained by first eroding A by B and then performing the set difference between A and its erosion as follows:

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

- Where B is a suitable structuring element.



Reading

- Kenneth: Chapter 3 and 4
- Szeliski: Section 5.5 (Graph cuts and energy-based methods) (**Optional Reading**)

Thank You 😊

Object Recognition and Detection

(Connected components, Template Matching, HOG, Active Contours)

By: Dr. Muhammad Fahim

Contents

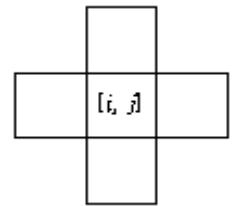
- Connected Components
- Connected Components Labelling (CCL)
- Object Recognition and Detection
 - Template Matching
 - Image Pyramids
 - Histogram of Oriented Gradients (HOG)
- Active Contour Models
- Summary

Slide Credits and Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - <https://slideplayer.com/slide/5215601/>
 - <https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/>
 - Found material over the internet to aligned the subject according to the need of the students.

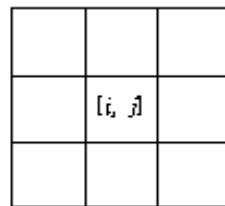
What is Connectivity?

- Connectivity is defined by **relationships between pixels**
 - 4-neighbors (4-connected)



$[i+1, j], [i-1, j], [i, j+1], [i, j-1]$

- 8-neighbors (8-connected)

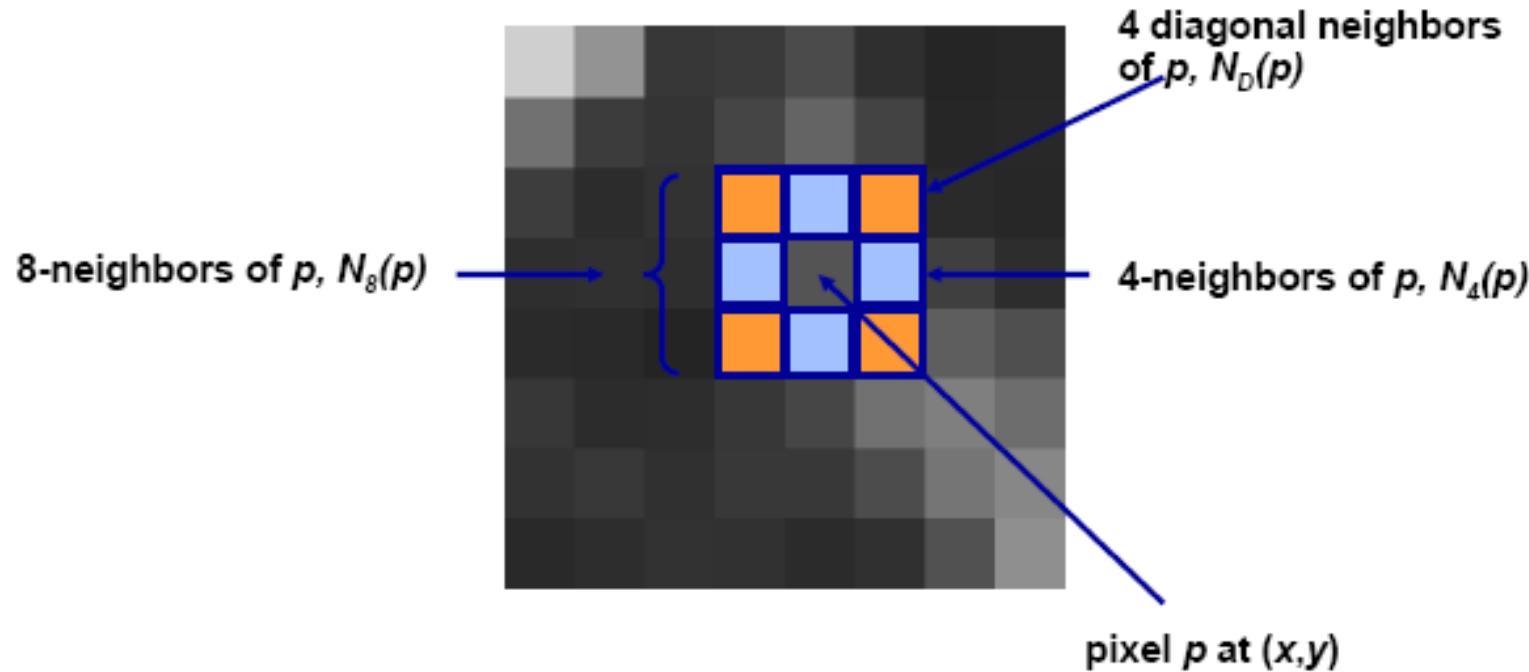


$[i+1, j], [i-1, j], [i, j+1], [i, j-1]$

$[i+1, j+1], [i+1, j-1], [i-1, j+1], [i-1, j-1]$

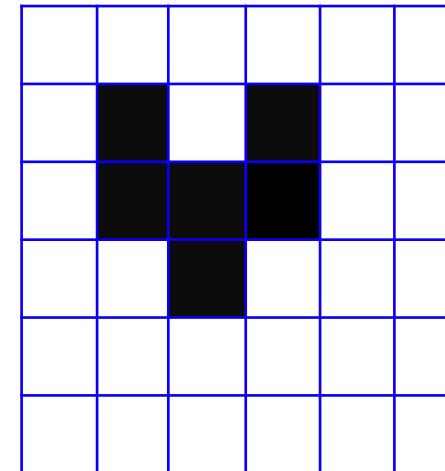
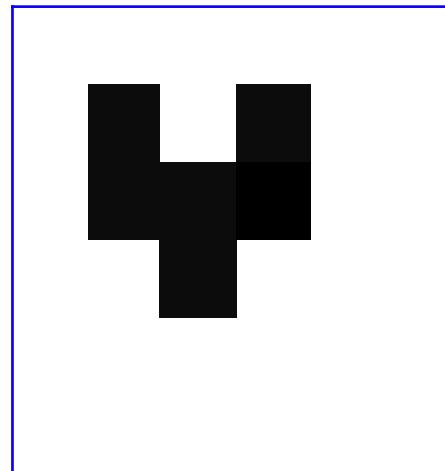
- **Connected components**
 - A set of pixels in which **each pixel** is connected to all other pixels.

Connectivity



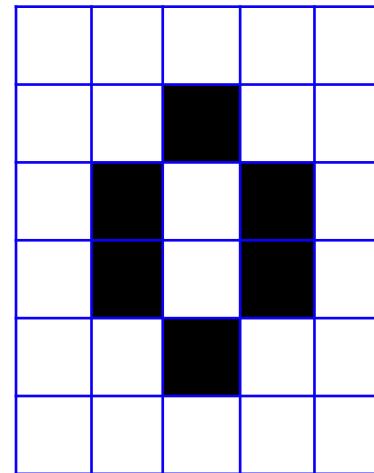
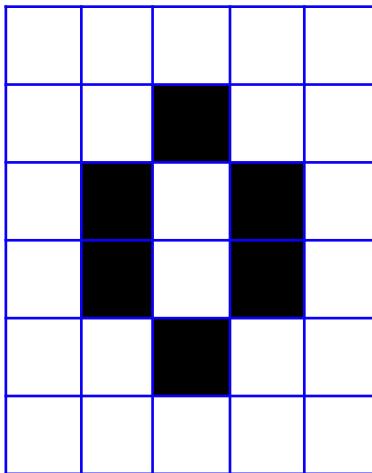
Connected Components

- How many objects?
- How many connected components?

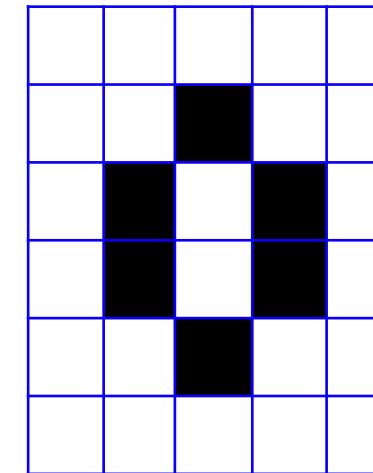


Connected Components

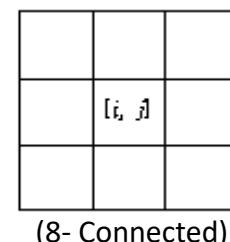
- How many connected components are there in the object?



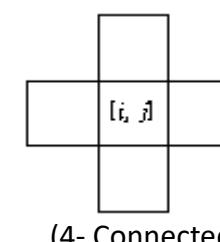
8-connected object



4-connected object



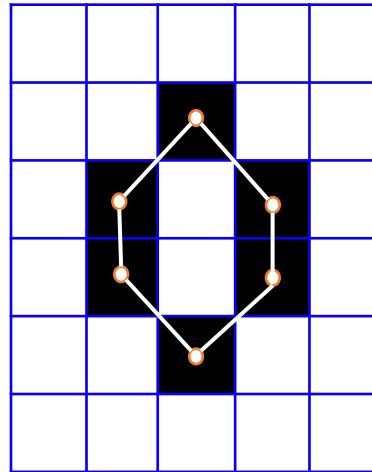
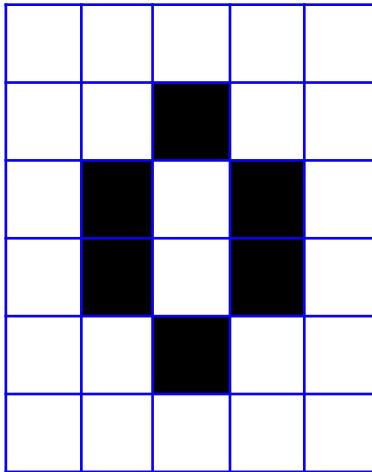
(8- Connected)



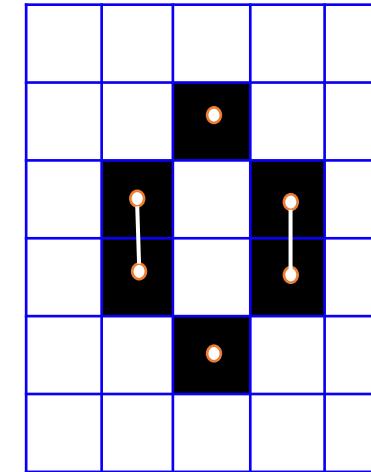
(4- Connected)

Connected Components

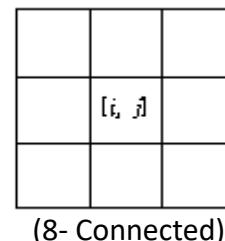
- How many connected components are there in the object?



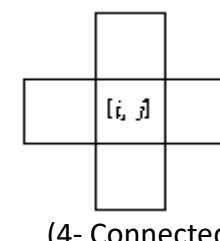
8-connected object
(1 component)



4-connected object
(4 components)



(8- Connected)



(4- Connected)

Connected Component Labeling (CLL)

- CCL scans an image and groups its pixels into components based on pixel connectivity
- Once all groups have been determined, each pixel is labeled with greyscale or color (labels) according to the component it was assigned to.



Figure 4.22 A frame from a surveillance image (left) and a labelled version of the cleaned binary image from Figure 4.17 (right). The original image (left) is reproduced by permission of Dr. James Ferryman, University of Reading

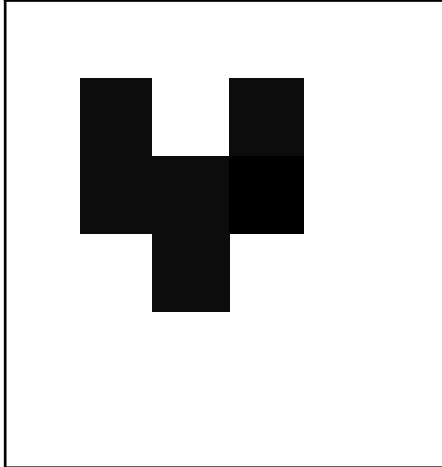
CCL – Important!

- It's important to note that we only apply connected-component analysis to ***binary*** or ***thresholded*** images.
- Given an RGB or grayscale image, threshold it based on some criterion in a manner that can segment the background from the foreground
- Once we have obtained the binary version of the image, we can proceed to analyze the components.

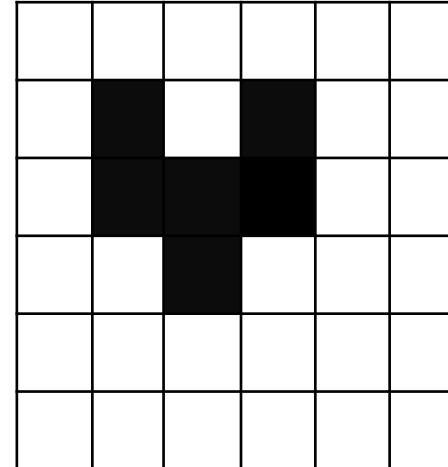
CCL: A Classical Approach (Rosenfeld and Pfaltz Algorithm)

- A clever use of **graph theory** to **analyze connected-components** in an image is very efficient and still **heavily used today**.
- Consists of two passes
 1. **First Pass:** The algorithm loops over **each individual pixel**, to determine its **connectivity** and **generate labels** for it.
 2. **Second Pass:** The connected-component analysis algorithm loops over the labels generated from the first pass and **merges any regions together that are connected but have different labels**

Processing the Images



Image



$f(x,y)$

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

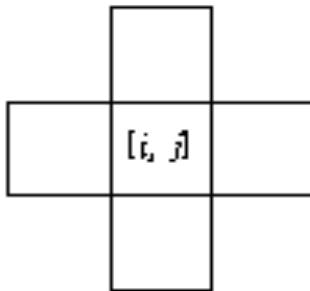
In Matrix

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

Easy to interpret

Finding the number of objects

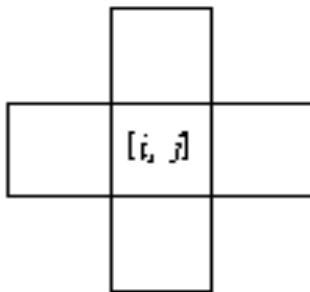
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

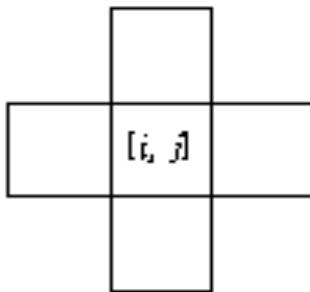
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

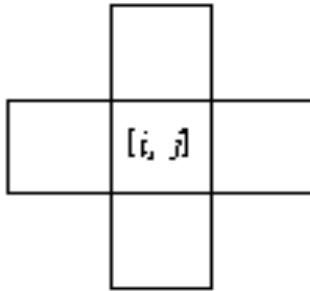
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

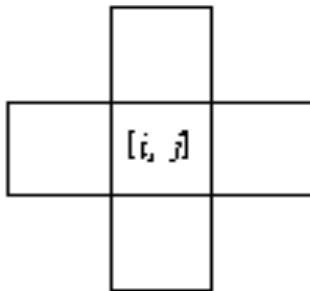
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

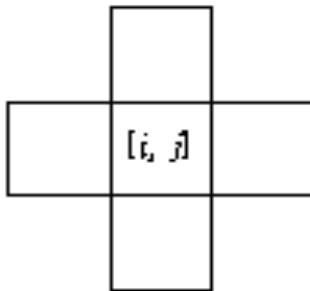
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

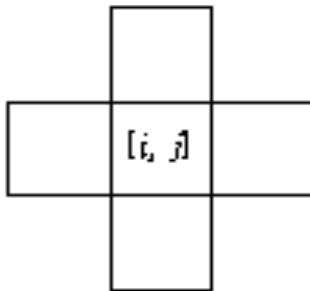
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

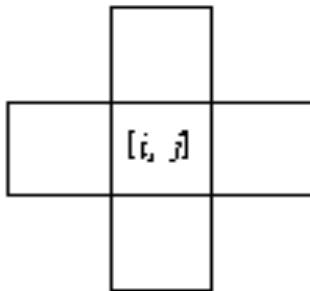
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

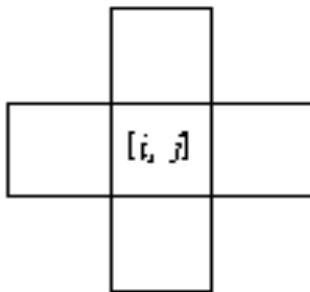
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

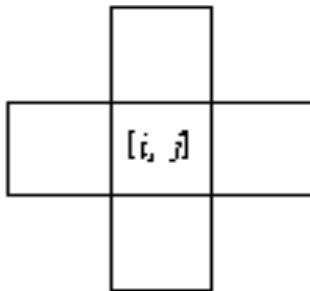
- **Considered Connectivity:** 4-connectivity



0	0	0	0	0	0
0	2	0	1	0	0
0	1	1	1	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Finding the number of objects

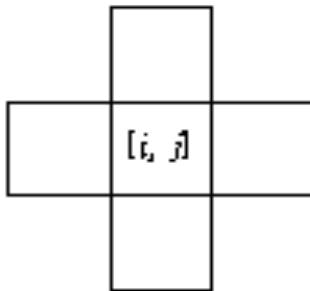
- **Considered Connectivity:** 4-connectivity



0	0	0	0	0	0
0	2	0	1	0	0
0	1	1	1	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Finding the number of objects

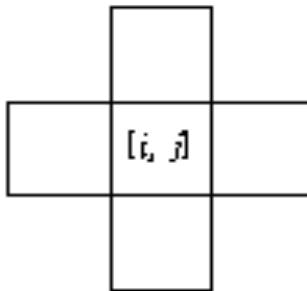
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

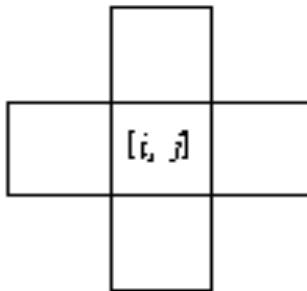
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

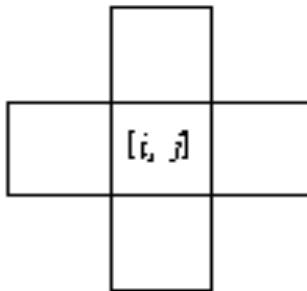
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

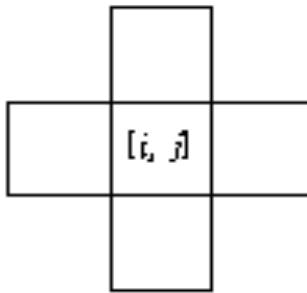
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

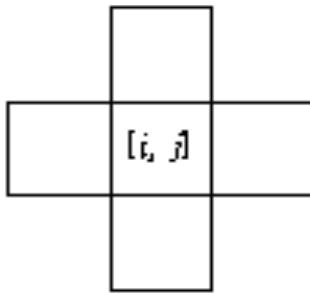
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

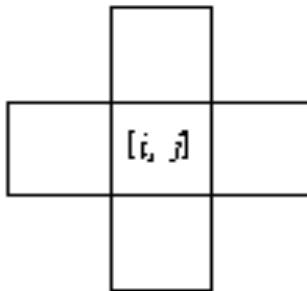
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

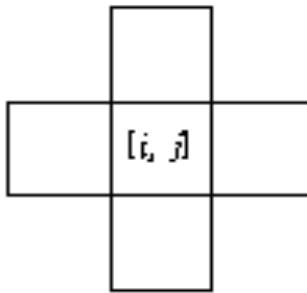
- **Considered Connectivity:** 4-connectivity



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

Finding the number of objects

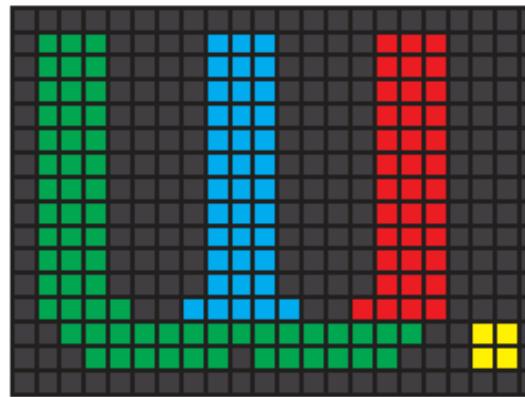
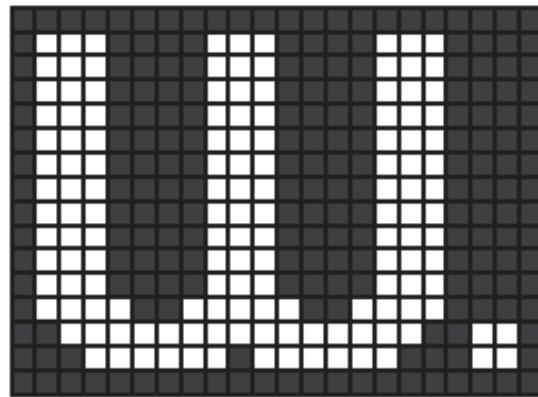
- **Considered Connectivity:** 4-connectivity



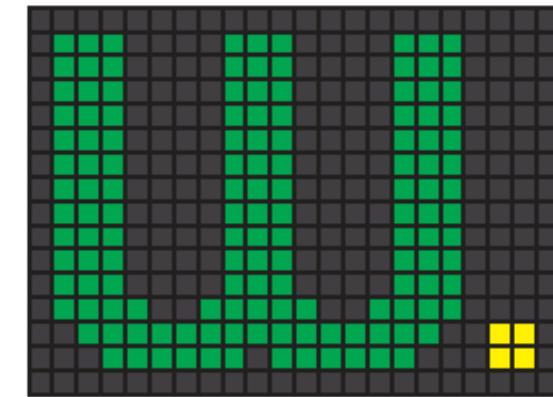
0	0	0	0	0	0
0	2	0	3	0	0
0	2	2	2	0	0
0	0	2	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

0	0	0	0	0	0
0	2	0	2	0	0
0	2	2	2	0	0
0	0	2	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Another Example



$$\begin{array}{c} \textcolor{blue}{\blacksquare} = \blacksquare \\ \textcolor{red}{\blacksquare} = \blacksquare \end{array}$$



Object Detection

- What is **object detection**?
- How is it **different** from simple **image classification**?
- What can be an “**object**?”
- Why is object detection **hard**?

Object Detection



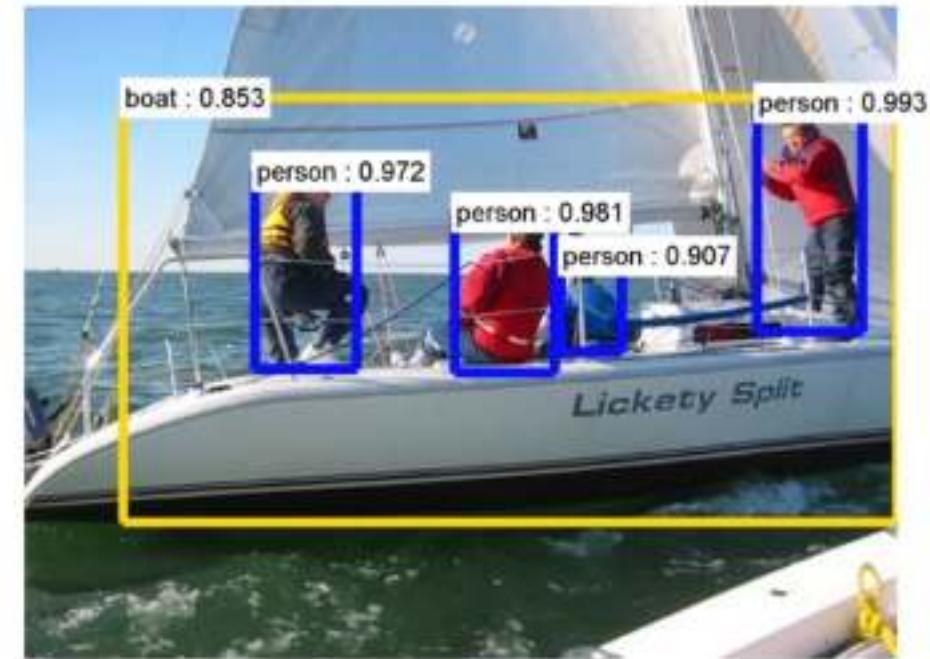
Object detection



Object detection



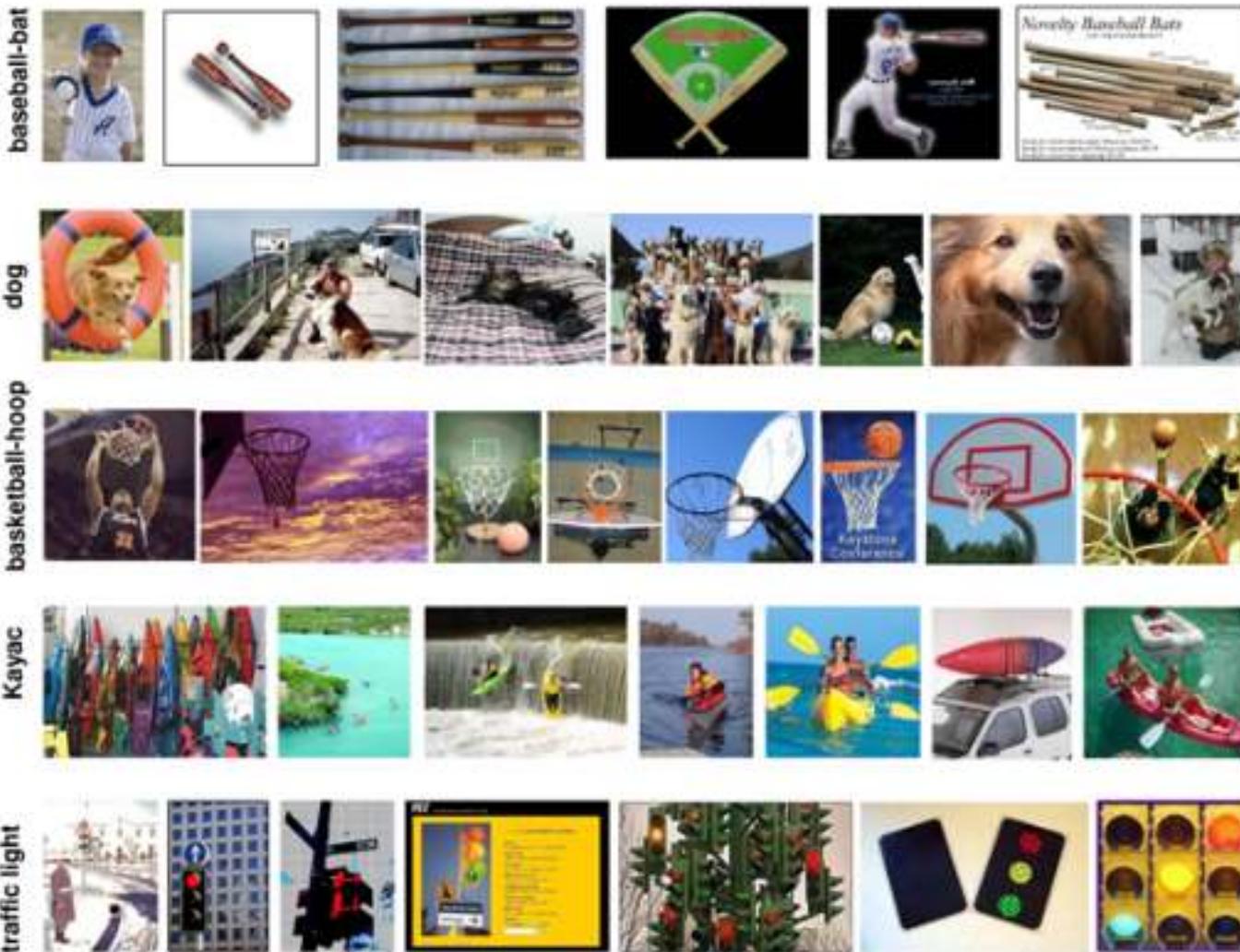
Image classification
(what? I don't care where)



Object detection
(what + where?)

Source: http://deeplearning.csail.mit.edu/instance_ross.pdf

Examples of objects in images

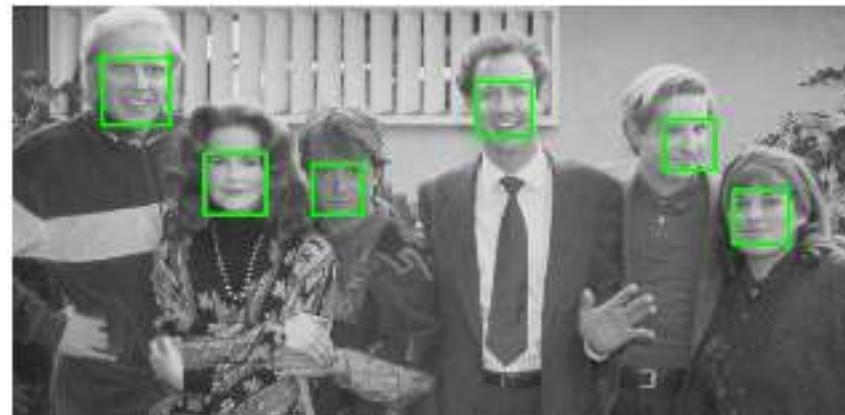
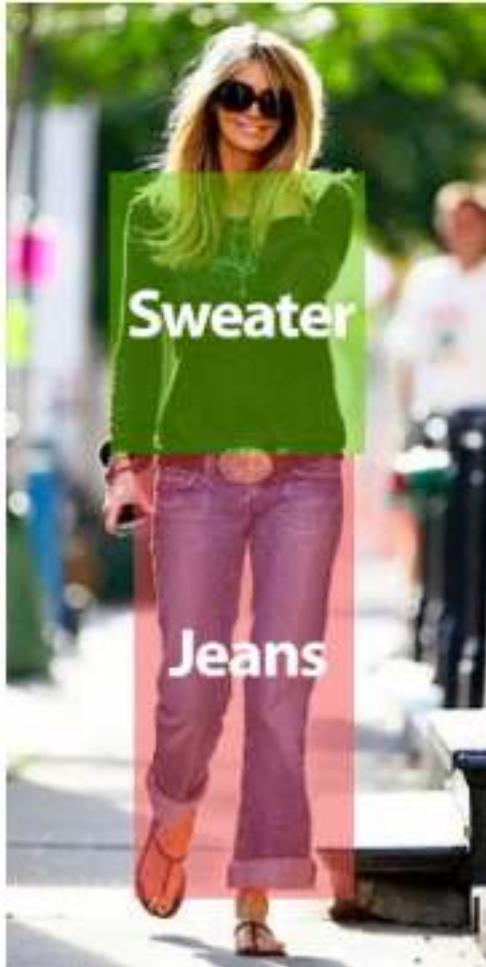


Why is object detection hard?

- Objects in the real-world can exhibit **substantial variations** in **viewpoint**, **scale**, **deformation**, **occlusion**, **illumination**, **background clutter**, and **intra-class variation**



Object detection – Applications



Approaches to Object Recognition

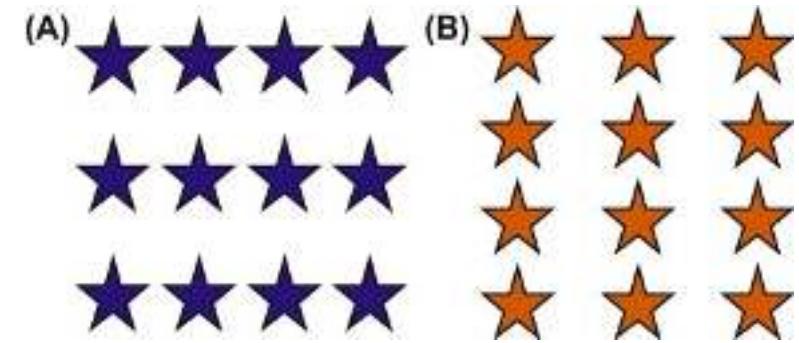
- **Generic Gestalt Principles**

- The world is structured, extract features
- Perceptual grouping



- **Model based**

- CAD model of object
- Geometric features
- Locate features and their arrangement



- (A) The stars are closer together horizontally than they are vertically, so we see three rows of stars
- (B) They are closer together vertically than they are horizontally, so we see three columns.

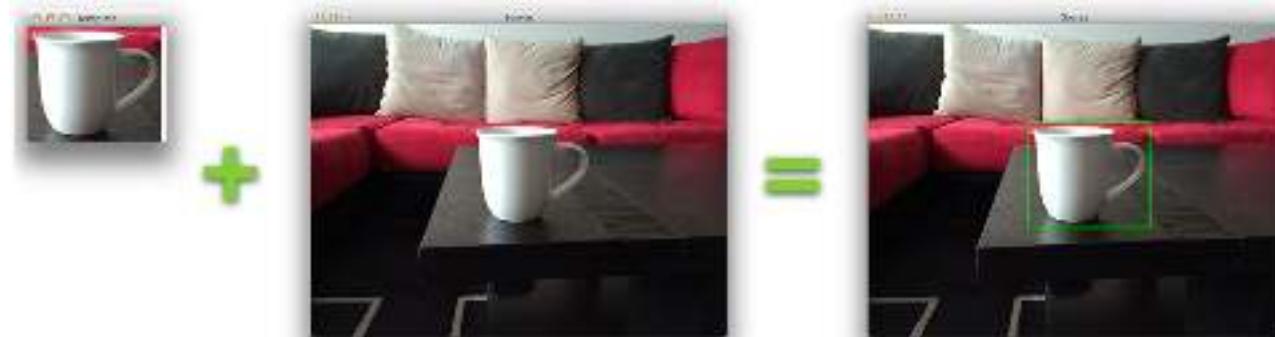
<https://www.sciencedirect.com/topics/computer-science/gestalt-principle>

- **Appearance based (Template Matching)**

- Interest points/point features
- or “whole” object

Template Matching

- Template matching is a technique for **finding areas of an image** that match (are similar) to a **template image** (patch)
- We need two primary components:
 - **Source image (I)**
 - The image in which we expect to find a match to the template image
 - **Template image (T):**
 - The patch image which will be compared to the template image
- Our goal is to detect the highest matching area



https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

Template Matching

- It is based on **similarity measures**
 - Some based on the summation of differences between the image and template
 - Other based on cross-correlation techniques

$$D_{\text{SquareDifferences}}(i, j) = \sum_{(m,n)} (f(i + m, j + n) - t(m, n))^2$$

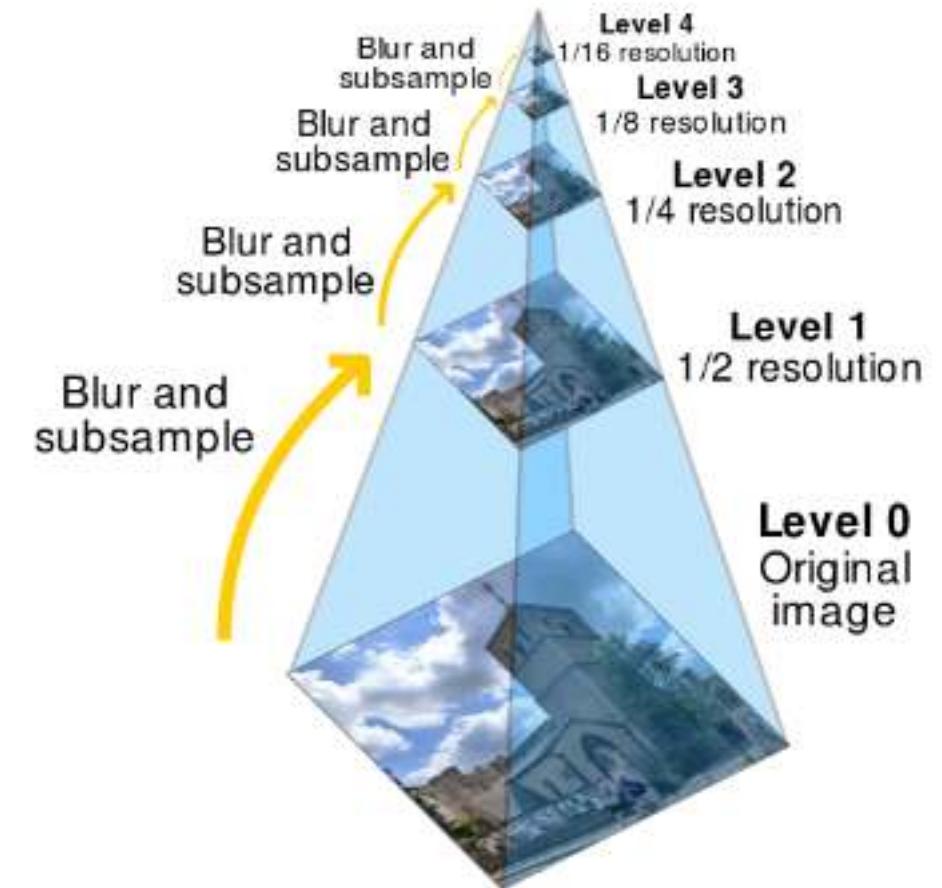
$$D_{\text{NormalisedSquareDifferences}}(i, j) = \frac{\sum_{(m,n)} (f(i + m, j + n) - t(m, n))^2}{\sqrt{\sum_{(m,n)} f(i + m, j + n)^2 \sum_{(m,n)} t(m, n)^2}}$$

$$D_{\text{CrossCorrelation}}(i, j) = \sum_{(m,n)} f(i + m, j + n) \cdot t(m, n)$$

$$D_{\text{NormalisedCrossCorrelation}}(i, j) = \frac{\sum_{(m,n)} f(i + m, j + n) \cdot t(m, n)}{\sqrt{\sum_{(m,n)} f(i + m, j + n)^2 \sum_{(m,n)} t(m, n)^2}}$$

Template Matching – Image Pyramids

- Normally, we used to work with an image of **constant size**.
- But in some occasions, we need to work with images of **different resolution of the same image**.
- **For example:** while searching for something in an image, **like face**, we are not sure at what size the object will be present in the image.
- In that case, we will need to create **a set of images** with different resolution and search for object in all the images.



Why blur and sub-sampled?



1/2



1/4 (2x zoom)



1/8 (4x zoom)



Gaussian 1/2

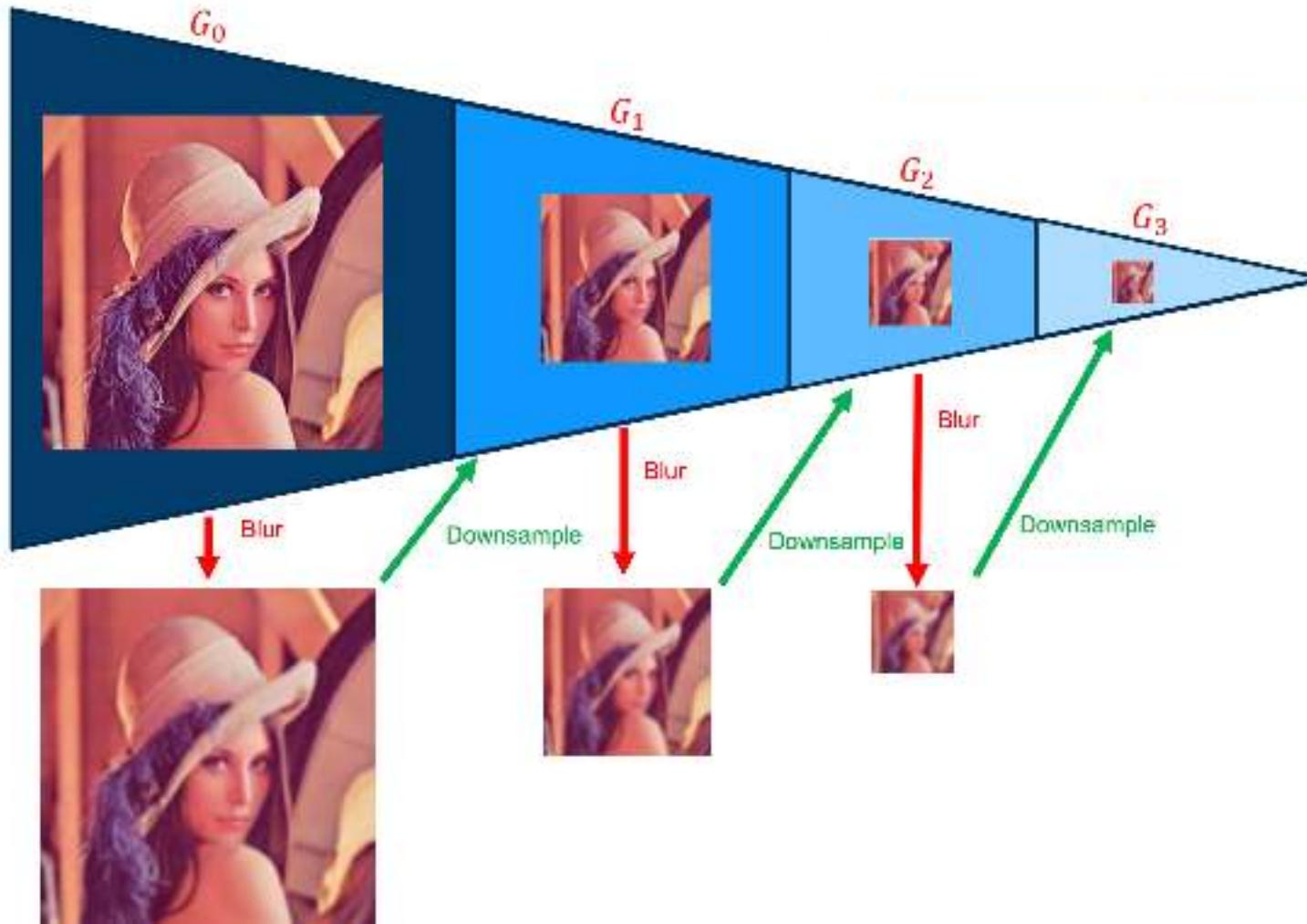


Gaussian 1/4



Gaussian 1/8

Template Matching – Gaussian Pyramids



Template Matching with Image Pyramids

- **Input:** Image, Template

1. Match template at current scale
2. Downsample image
3. Repeat 1-2 until image is very small
4. Take responses above some threshold, perhaps with **non-maxima suppression.**

More robust object detection

- We will learn [a better way](#) to detect objects in images
- More specifically, we will need
 - An image descriptor ([HOG](#))
 - A classifier ([SVM](#))
 - Methods to overcome “[scale](#)” and “[location](#)” problems

Histogram of Oriented Gradients (HOG)

- First introduced by *Dalal and Triggs* in their CVPR 2005 paper, *Histogram of Oriented Gradients for Human Detection*
- HOG uses a global feature to describe a person **rather than a collection of local features**
- The entire **person** is represented by a **single feature vector** (not by many feature vectors representing smaller parts of the person)
- A **Support Vector Machine (SVM)** was trained to classify HOG descriptors of people
 - Positive examples (images with people)
 - Negative examples (images without people)

NOTE: In order to understand the concept, I'll keep the original paper description (i.e., HOG person detector)

Histogram of Oriented Gradients (HOG)

- HOG is a **feature descriptor** that is often used to **extract features** from image data.
- It is widely used in computer vision tasks for **object detection**
- **Steps**
 1. Preprocessing the data
 2. Calculate gradients histogram
 3. Normalized gradients
 4. Feature descriptor

1. Preprocess the Data

- We need to preprocess the image and bring down the width to height ratio 64×128
- **Why 64×128 ?** This is the exact value used in the original paper
- Below are some of the **original images used to train the detector**, cropped in to the 64×128 window



64x128



64x128



64x128



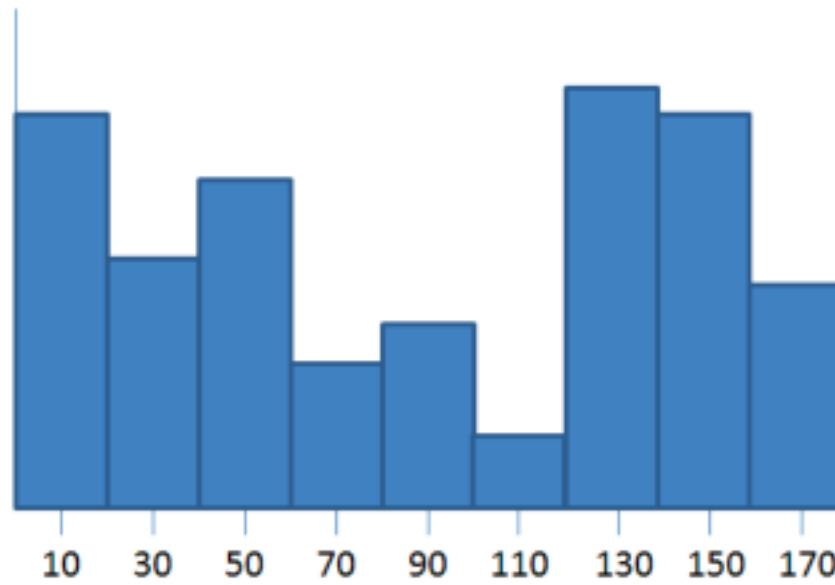
64x128



64x128

2. Calculate Gradient Histograms

- The HOG descriptor is constructed using 8x8 pixel cells
- Within each cell, compute the gradient vector at each pixel
- Generate a 9-bin histogram from the 64 gradient vectors
- The histogram ranges from 0 to 180 degrees (20 degrees per bin)



NOTE: Dalal and Triggs used “unsigned gradients” so that orientations only ranged from 0 to 180 degrees instead of 0 to 360

2. Calculate Gradient Histograms – Simplified

- First calculate the horizontal and vertical gradients

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$



- We can also achieve the same results, by using **Sobel**

$$\frac{\partial f}{\partial x} = S_x \otimes f$$

$$\frac{\partial f}{\partial y} = S_y \otimes f$$

2. Calculate Gradient Histograms – Simplified

- Calculate the gradient magnitude and orientation:

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

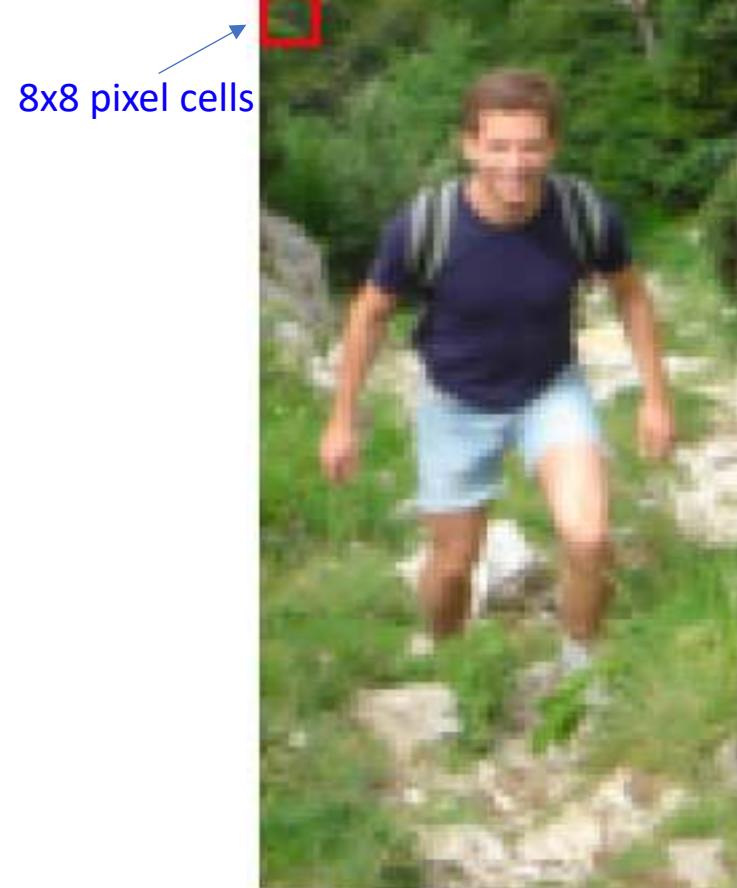
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2}$$

80	36	5	10	0	84	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

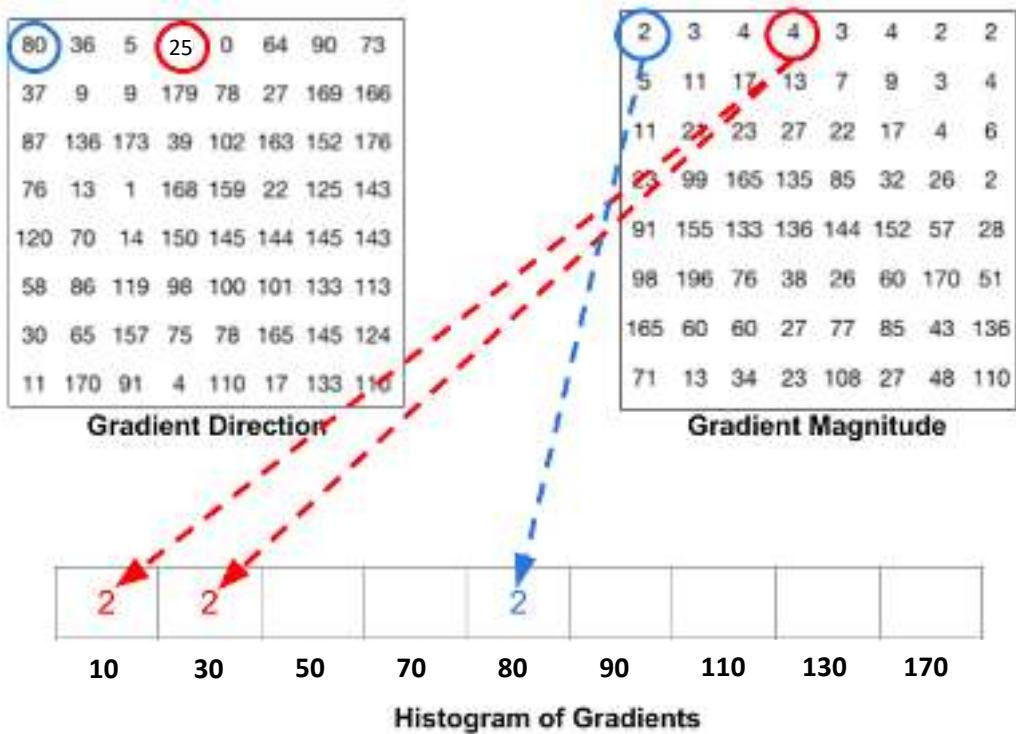
2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude



- The next step is to create a histogram of gradients in these 8×8 cells.
- The histogram contains 9 bins corresponding to angles 0, 20, 40 ... 160.

2. Calculate Gradient Histograms – Simplified



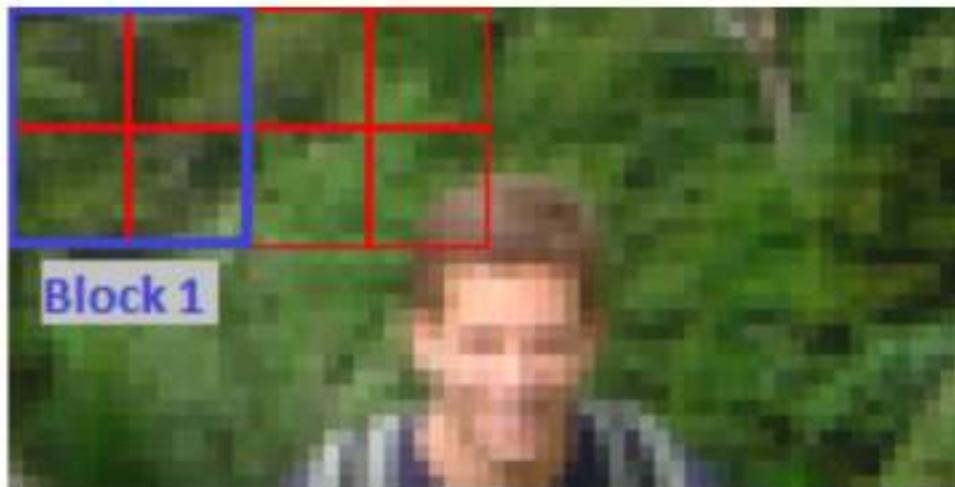
64x128

Special Case: The contribution is split between the two closest bins

- For example: if a gradient vector has an angle of 25 degrees
 - Add 1/2 of its magnitude to the bin centered at 10 degrees
 - Add 1/2 of its magnitude to the bin centered at 30

3. Normalized Gradients

- Normalize the histogram so they are not affected by **lighting variations**
- Rather than normalize each histogram individually, the cells are first grouped into **blocks of 2x2 cells** and normalized based on **all histograms in the block**
- The blocks have 50% overlap



3. Normalized Gradients

- Mathematically, for a given vector h :

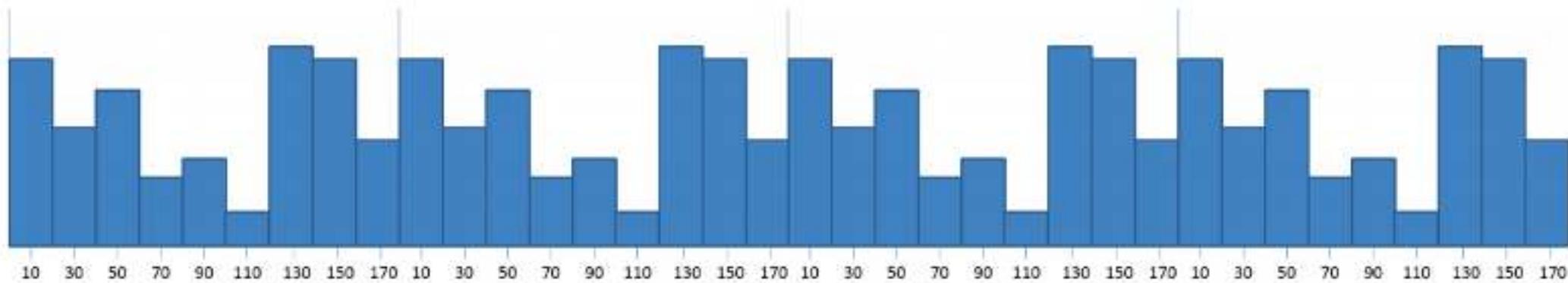
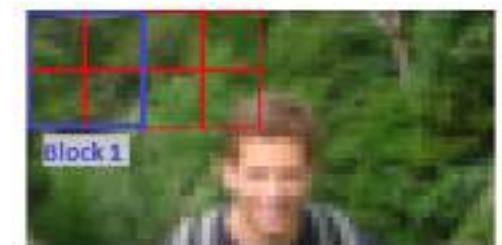
$$h = [m1, m2, m3, \dots, m36]$$

$$k = \sqrt{(m1)^2 + (m2)^2 + (m3)^2 + \dots + (m36)^2}$$

$$\text{Normalized Gradient} = \left[\frac{m1}{k}, \frac{m2}{k}, \frac{m3}{k}, \dots, \frac{m36}{k} \right]$$

3. Normalized Gradients

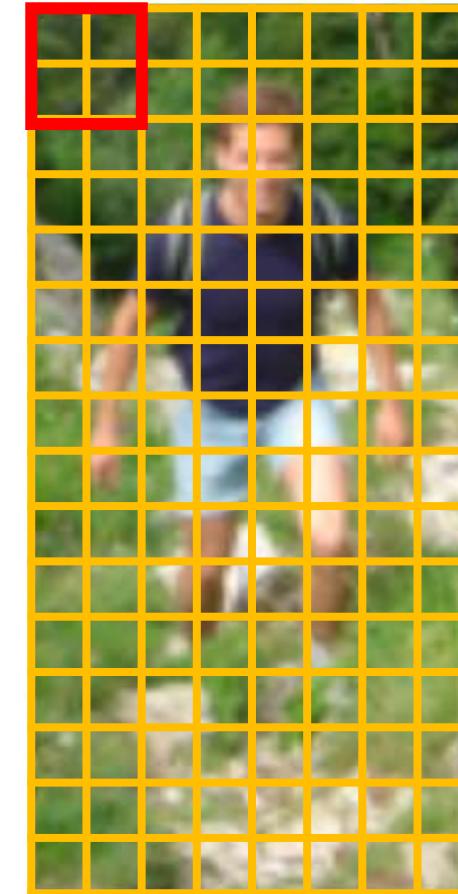
- Block normalization is performed by concatenating the **histograms of the four cells** within the block into a **vector with 36 components**



(4 histograms x 9 bins per histogram)

4. Feature for Complete Image (64 x 128)

- We would have 105 (7×15) blocks
- Each of these 105 blocks has a vector of 36×1 as features
- Hence, the total features for the single image
 $105 \times 36 \times 1 = 3780$ features

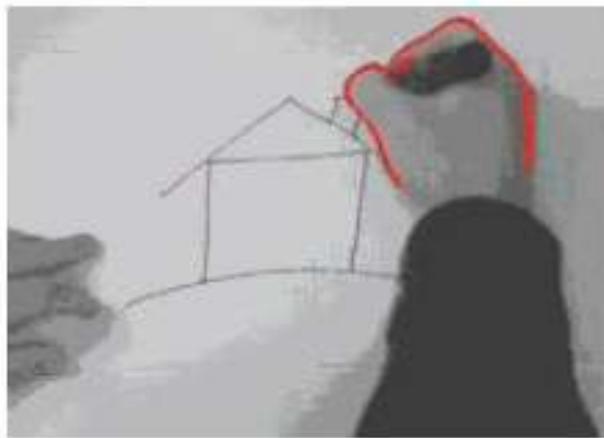


Support Vector Machine (SVM) classification

- Reference: Introduction to Machine Learning course by *Prof. Adil Khan*

Contour

- Contours can be explained simply as a **curve joining all the continuous points (along the boundary), having same color or intensity.**
- The contours are a useful tool for **shape analysis, object detection and recognition**



Human Computer Interaction



Surveillance/Speed control



Face Recognition

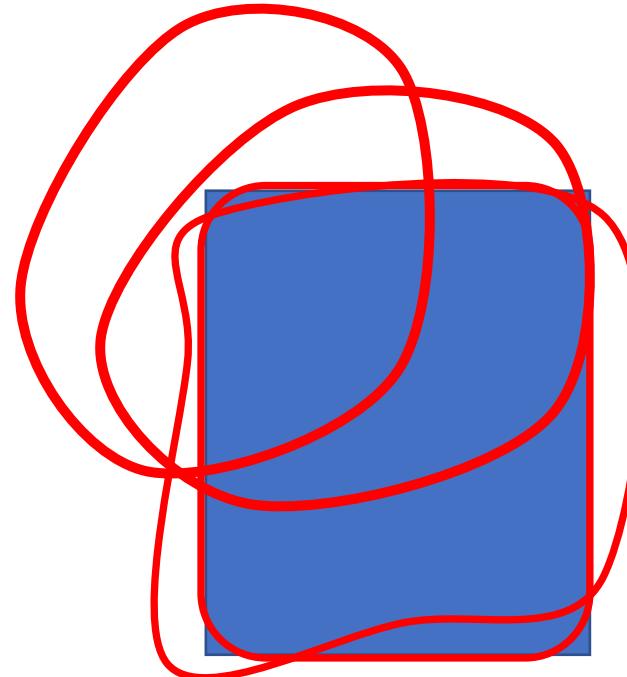
NOTE: It is also known as Active Contour

Active Contour Models

- Two types of model
 - Parametric active contour model
 - Snake
 - Balloon model
 - Gradient Vector Flow(GVF) snake model
 - Geometric active contour model
 - Level set

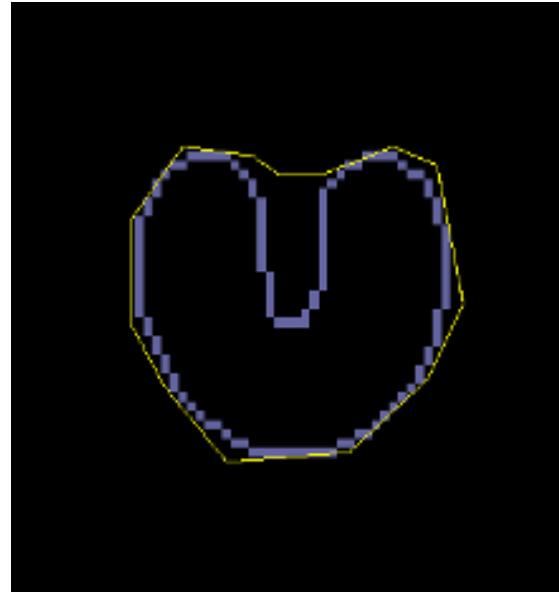
Framework for Snakes

- A higher-level process or a user initializes any curve *close to the object boundary*.
- The snake then starts *deforming* and moving towards the desired object boundary.
- In the end it completely "shrink-wraps" around the object.



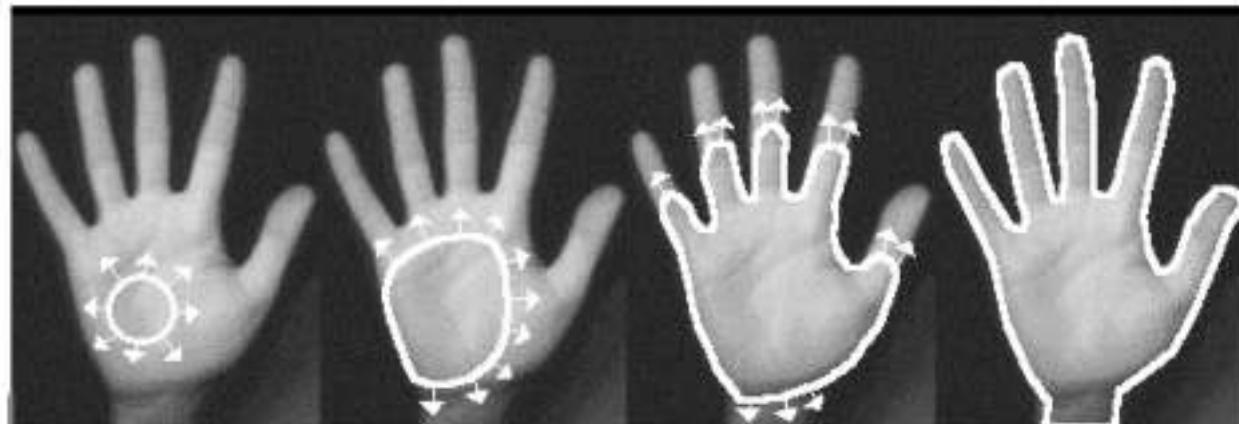
Weakness of traditional snakes (Kass model)

- Fails to detect **concave boundaries**. External force can't pull control points into boundary concavity.
- The **initial contour must**, in general, be **close to the true boundary** or else it will likely converge to the wrong result.



Balloon Model

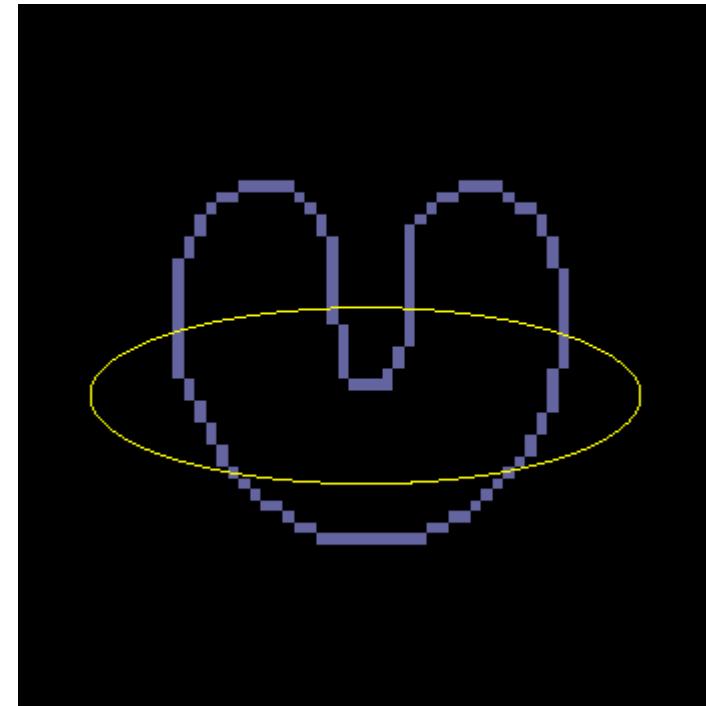
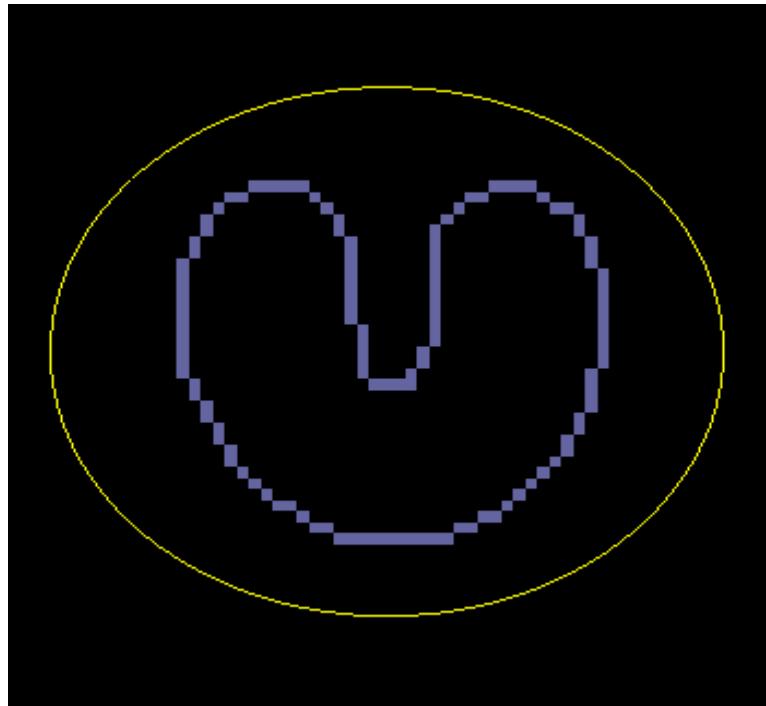
- Works on the same principles as the snake.
- Where the snake would shrink while balloon model expands.
- Introduces makes it possible to find the contours of an object by placing the initial snake inside the object instead of outside, while also providing more stable results.



<http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/bmvc97/paper.html>

Gradient Vector Flow (GVF)

- A new external force for snakes. A snake with GVF external forces moves into the concave boundary region.
- The contour can also be initialized across the boundary of object!! Something not possible with traditional snakes.



Reading

- Kenneth: Chapter 8
- Szeliski: Section 3.5, Section 14.1.2 Pedestrian detection
- D. Vernon, Machine Vision: Automated Visual Inspection and Robot Vision, Prentice-Hall, 1991 – Section 6.2 Template matching

Q & A

Image Features (Part I)

(Feature Detectors, Descriptor and Matching)

By: Dr. Muhammad Fahim

Contents

- Introduction
- Feature Detectors
 - Harris Detector
 - Shi-Tomasi Detector
 - Lowe Detector (SIFT)
- Feature Descriptor
 - Lowe Descriptor (SIFT)
- Feature Matching
- Application Areas

Slides Credit and Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - http://www.cs.cornell.edu/courses/cs5670/2019sp/lectures/lec04_harris.pdf
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - 16-385 Computer Vision -- Spring 2019 at CMU.
 - Found material over the internet to aligned the subject according to the need of the students.

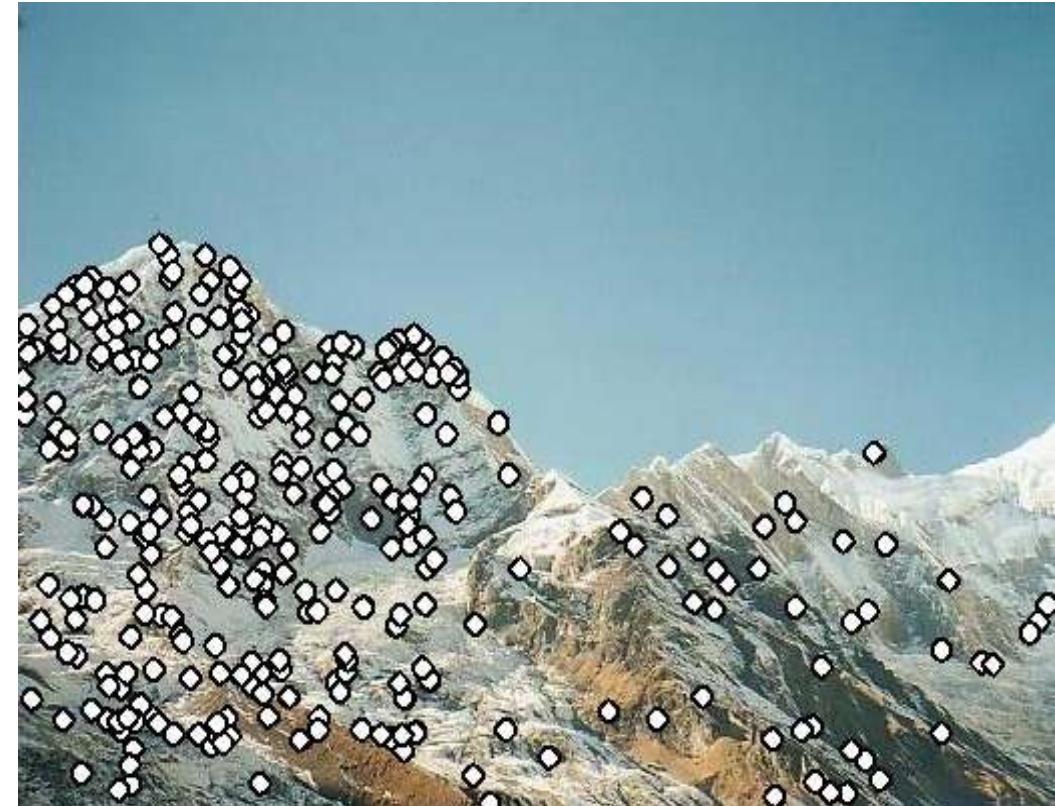
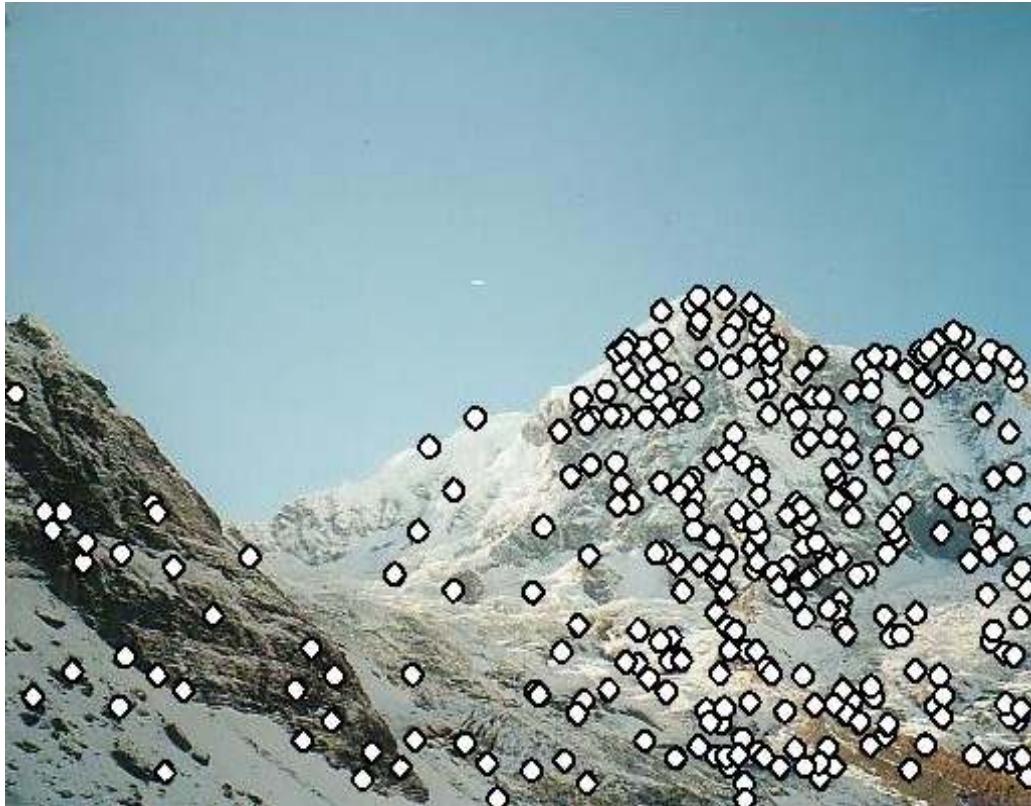
Feature Extraction

- We have two images – how do we combine them?



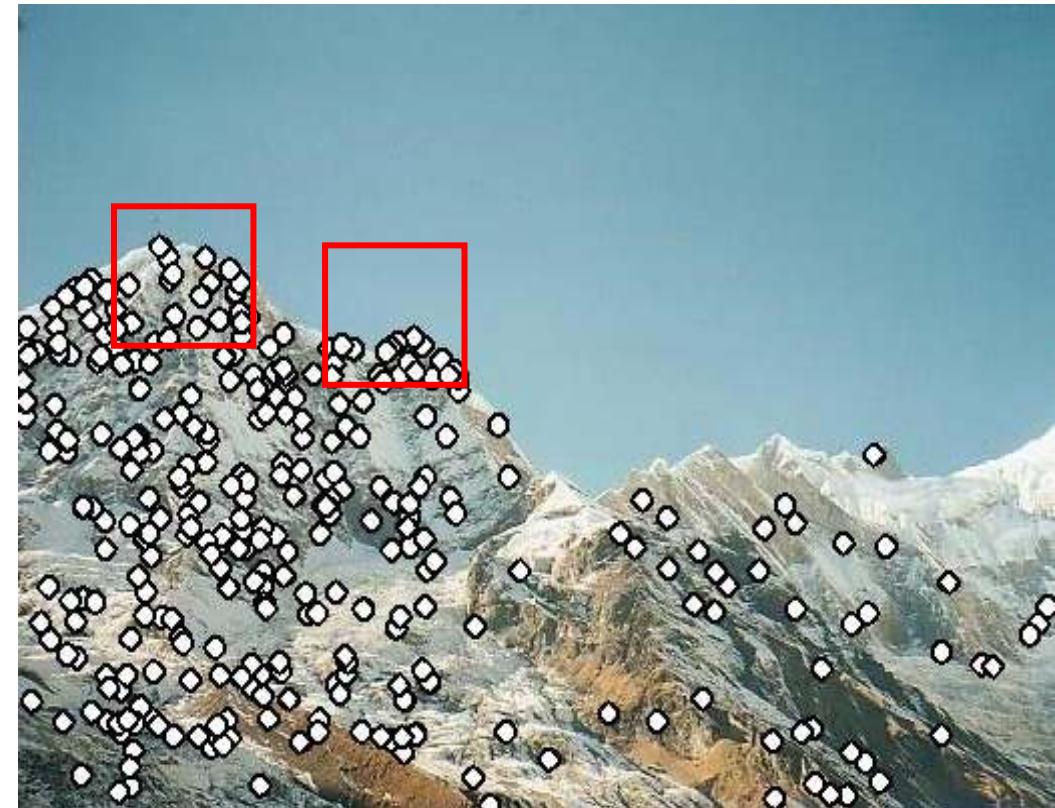
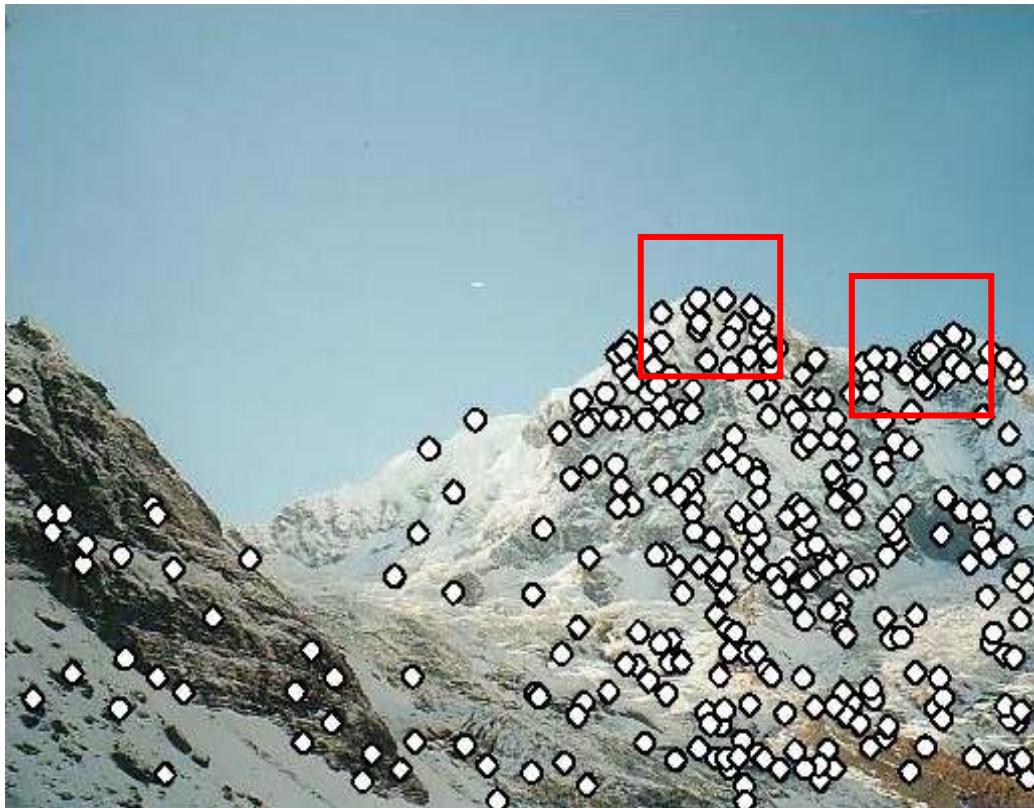
1. Identify Interest Points

- Detect interest points(features) in both images



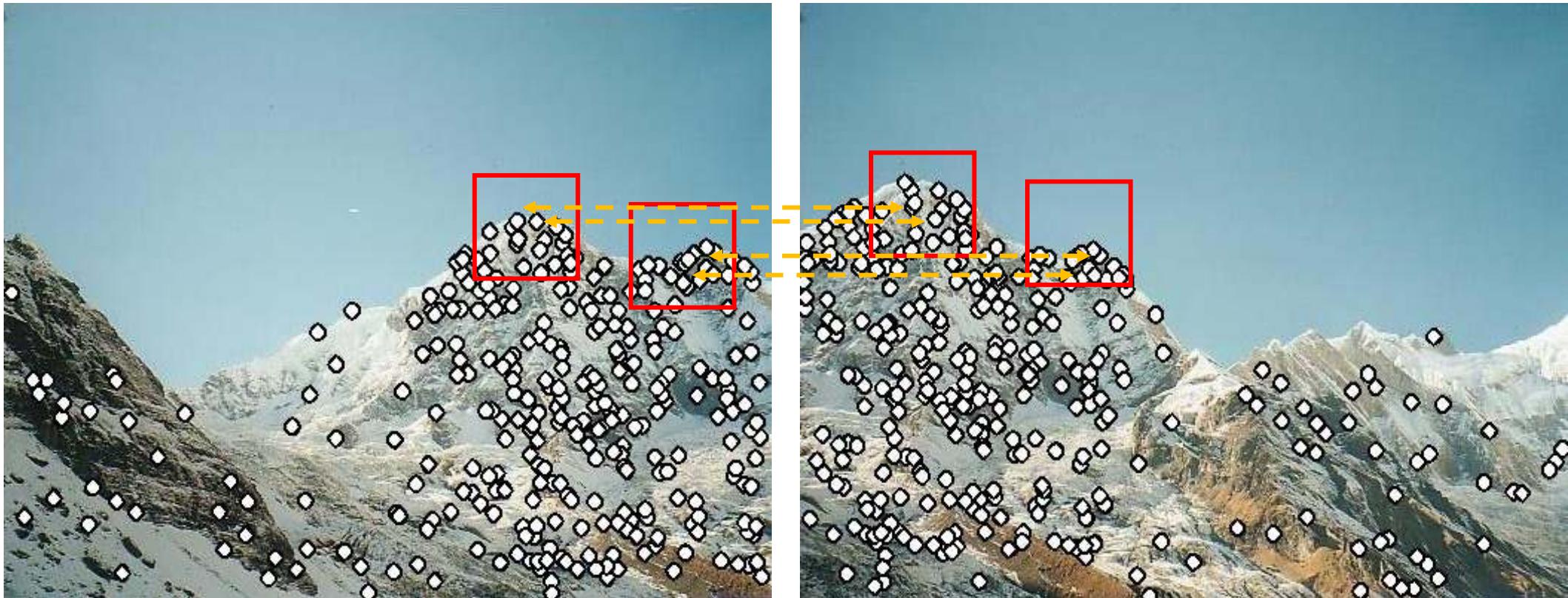
2. Feature Description

- Extract vector feature descriptor surrounding each interest point.



3. Feature Matching

- Determine correspondence between descriptors in two views



Feature Extraction – Panorama stitching



Visual Slam



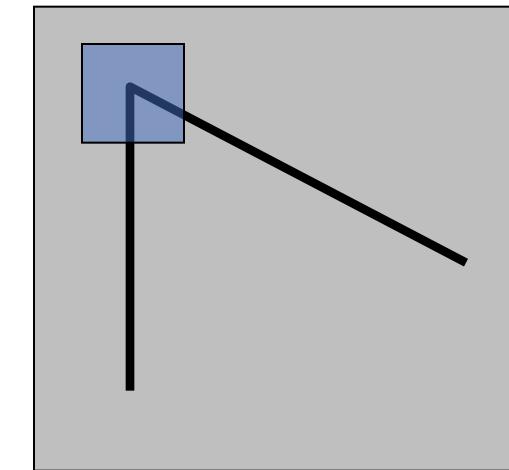
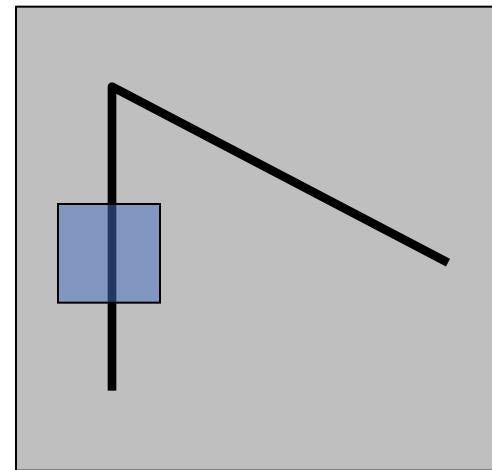
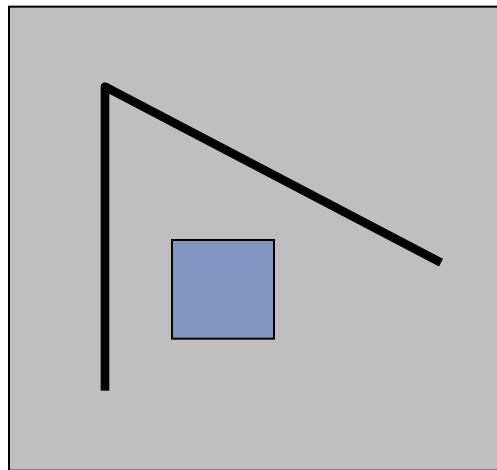
- See the video: <https://www.youtube.com/watch?v=TR8BMZj-Udc>

Feature Detector

- Harris Detector
- SIFT

What makes a good feature/interest points?

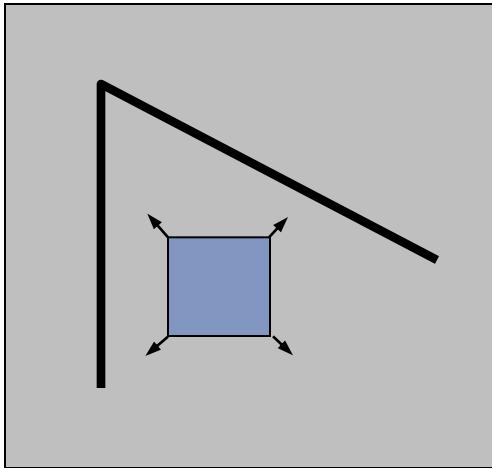
- Suppose we only consider a small window of pixels
- What defines whether a feature is a **good or bad candidate?**



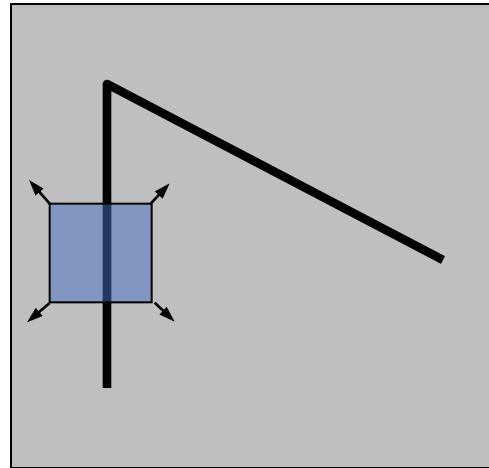
Credit: S. Seitz, D. Frolova, D. Simakov

What makes a good feature?

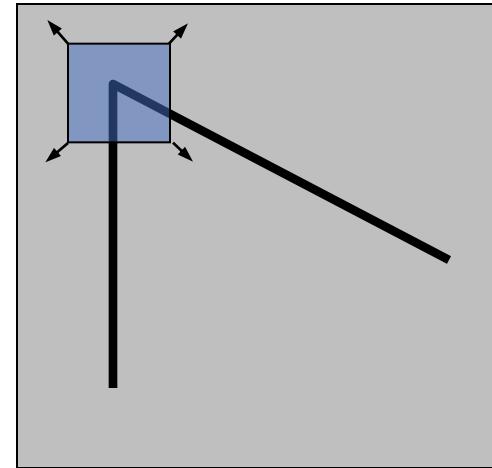
- How does the **window** change when you shift it?



“flat” region:
no change in all
directions



“edge”:
no change along the
edge direction

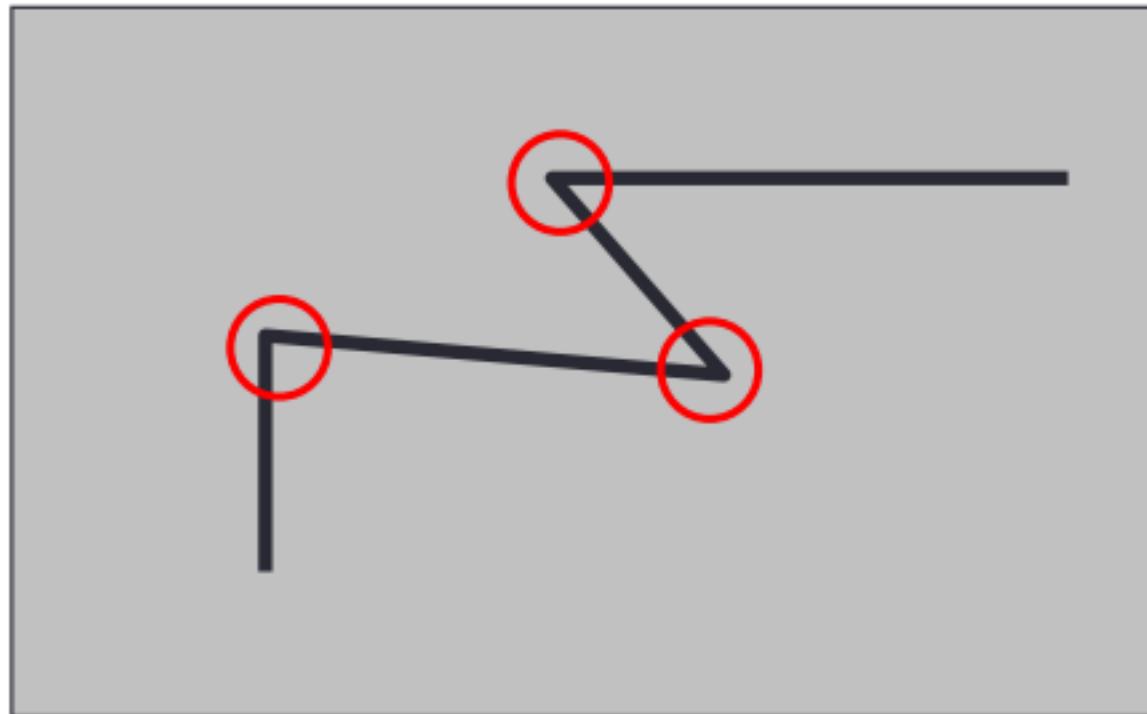


“corner”:
significant change in
all directions

Harris corner detection (based on the corner)

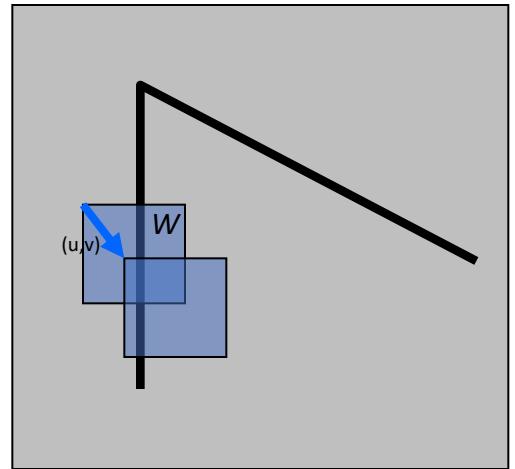
Credit: S. Seitz, D. Frolova, D. Simakov

Feature Detector: Harris Corner Detector



Mathematics of Harris Corner Detection

- Consider shifting the window W by (u, v)
 - How do the pixels in W change?



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

Limitation: Slow to compute exactly for each pixel and each offset (u, v)

Mathematics of Harris Corner Detection

- 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**

$$I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

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

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

Mathematics of Harris Corner Detection

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

Mathematics of Harris Corner Detection

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

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

Mathematics of Harris Corner Detection

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

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

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

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

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

$$\begin{bmatrix} u & v \end{bmatrix} H \begin{bmatrix} u \\ v \end{bmatrix}$$

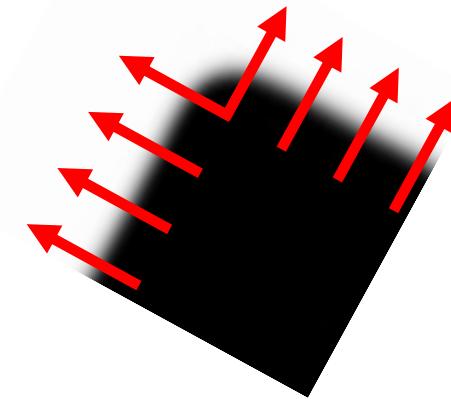
Ellipse Equation

Mathematics of Harris Corner Detection

We have,

$$H = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

Since H is symmetric



- **What does the matrix H reveal?**

The *eigenvalues of H* reveal the **amount of intensity change** in the two principal **orthogonal gradient directions** in the window.

Review!! Eigen Vectors and Eigen Values

- The eigen vector, x , of a matrix A is **special vector**, with property:

$$Ax = \lambda x$$

- **Eigen values** of a matrix A:

$$\det(A - \lambda I) = 0$$

- Find the **eigen vector** for corresponding eigen value:

$$(A - \lambda I)x = 0$$

Review!! Example

$$A = \begin{bmatrix} -1 & 2 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 7 \end{bmatrix}$$

Eigen Values: $\lambda_1 = 7$
 $\lambda_2 = 3$
 $\lambda_3 = -1$

Eigen Vectors: $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 4 \\ 4 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

Source: Dr. Mubarak Shah

Review!! Example – Eigen Values

- **Finding Eigen Values**

$$\det(A - \lambda I) = 0$$

$$\det\begin{bmatrix} -1 & 2 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 7 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 0$$

$$\det\begin{bmatrix} -1-\lambda & 2 & 0 \\ 0 & 3-\lambda & 4 \\ 0 & 0 & 7-\lambda \end{bmatrix} = 0$$

$$(-1-\lambda)((3-\lambda)(7-\lambda)-0) = 0$$

$$(-1-\lambda)(3-\lambda)(7-\lambda) = 0$$

$$\lambda = -1, \quad \lambda = 3, \quad \lambda = 7$$

Source: Dr. Mubarak Shah

Review!! Example – Eigen Vectors

$$A = \begin{bmatrix} -1 & 2 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 7 \end{bmatrix}$$

$$\lambda = -1$$

$$(A - \lambda I)x = 0$$



$$\begin{bmatrix} -1 & 2 & 0 \\ 0 & 3 & 4 \\ 0 & 0 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 & 0 \\ 0 & 4 & 4 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$0+2x_2+0=0$$

$$0+4x_2+4x_3=0$$

$$0+0+8x_3=0$$

$$x_1=1, \quad x_2=0, \quad x_3=0$$

Detail:

[https://matrixcalc.org/vectors.html#eigenvectors\(%7B%7B-1,2,0%7D,%7B0,3,4%7D,%7B0,0,7%7D%7D\)](https://matrixcalc.org/vectors.html#eigenvectors(%7B%7B-1,2,0%7D,%7B0,3,4%7D,%7B0,0,7%7D%7D))

Source: Dr. Mubarak Shah

Mathematics of Harris Corner Detection

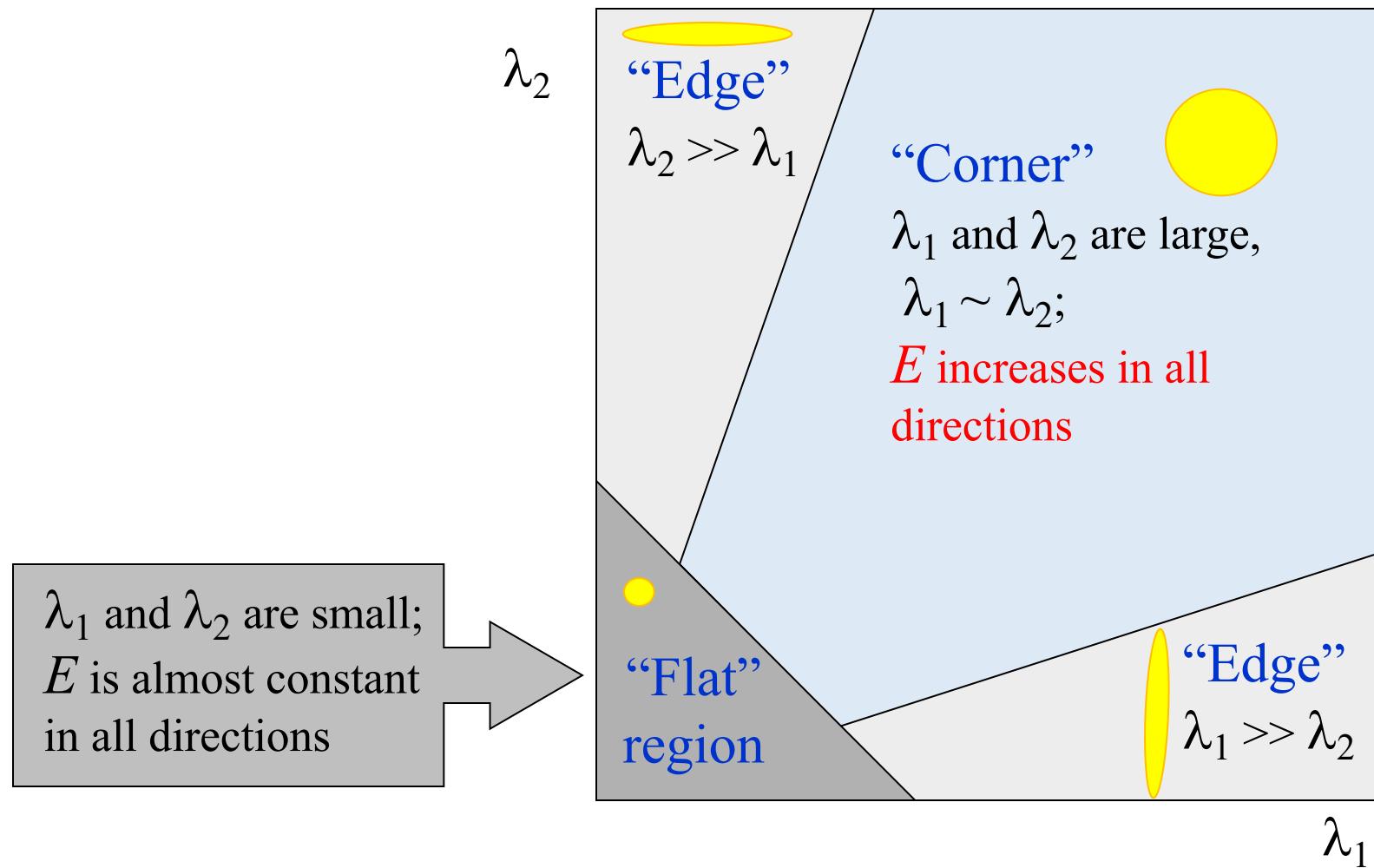
$$[u \ v] \ H \begin{bmatrix} u \\ v \end{bmatrix}$$

Ellipse Equation

- We can visualize H as an ellipse with
 - axis lengths determined by the eigenvalues of H and
 - orientation determined by the eigenvectors of H .
- Let λ_1, λ_2 are eigen values of H

Interpretation of Eigenvalues

- Classification of image points using eigenvalues of H :



Harris Corner Detection

- Measure of corner response (R):

$$R = \det(H) - \alpha \operatorname{trace}(H)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

$$\begin{array}{l} \det H = \lambda_1 \lambda_2 \\ \operatorname{trace} H = \lambda_1 + \lambda_2 \end{array}$$

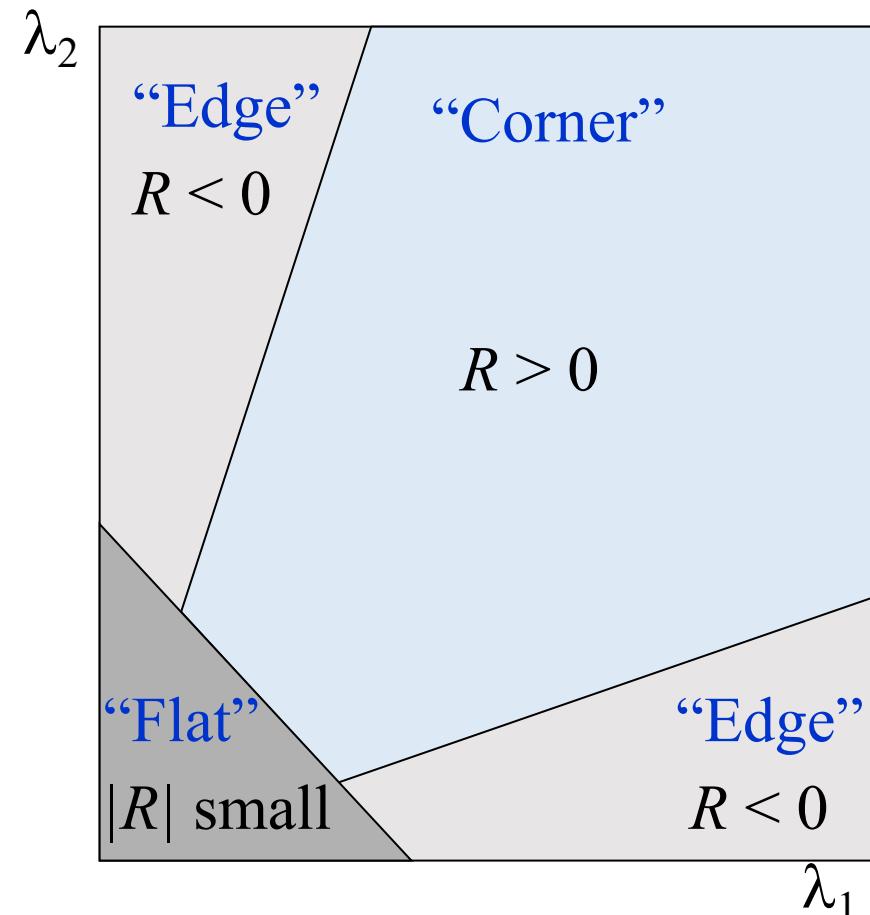
$$H = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

Because H is symmetric

$0 < \alpha < 0.25$ is a empirical constant (value ~ 0.05)

- R depends only on eigenvalues of H

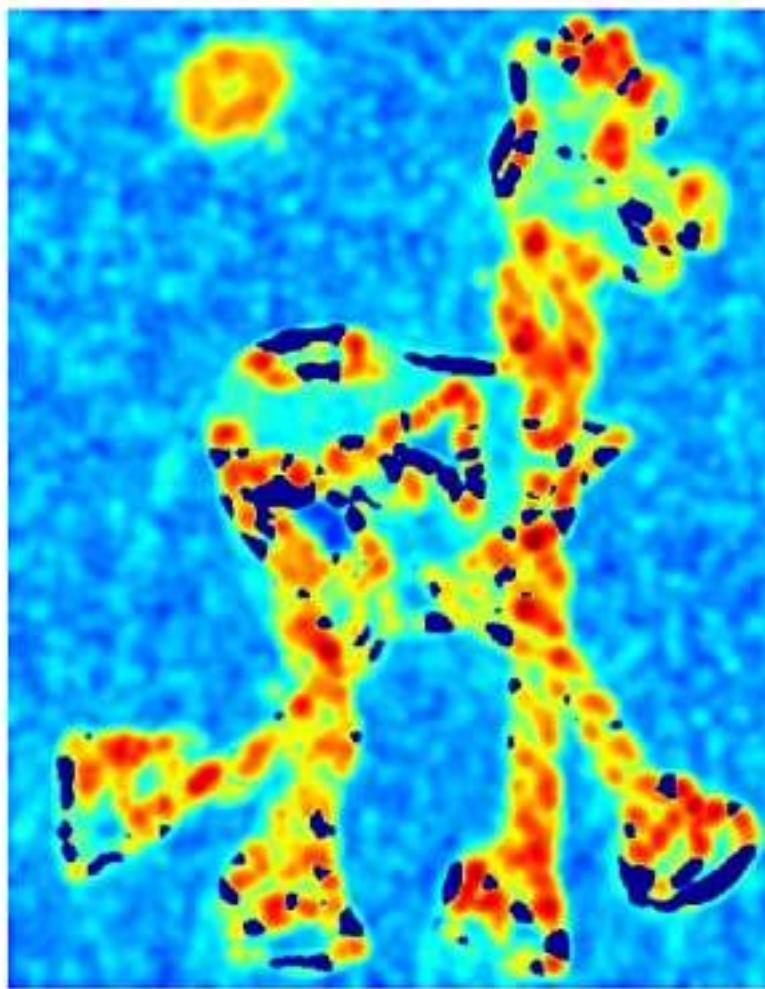
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



Harris Detector Example



Harris corner response (red high, blue low)



Thresholded corner response ($R > \text{Threshold}$)



Compute non-maximal suppression



Harris Features



Weighting the derivatives

- In practice, using a simple window W doesn't work too well

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

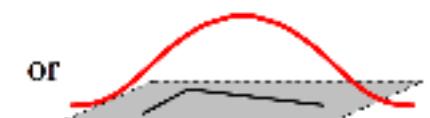
- Instead, we'll weight each derivative value based on its distance from the center pixel

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

$$w(x,y) =$$



1 in window, 0 outside

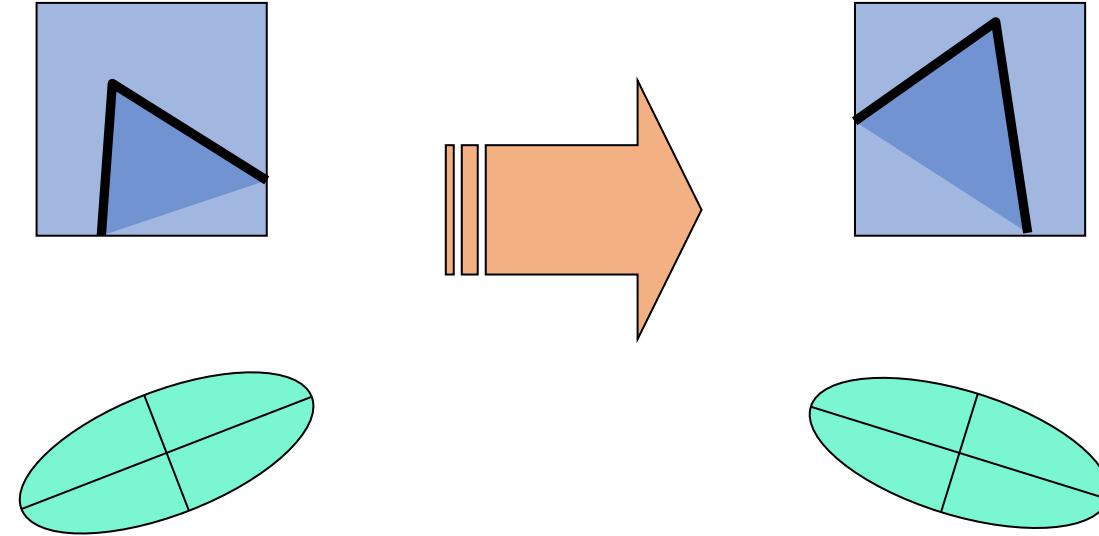


Gaussian

Harris Corner Detection: Summary

1. Compute x and y derivatives of image
2. Compute products of derivatives at every pixel
3. Compute the sums of the products of derivatives at each pixel
4. Define the matrix at H each pixel
5. Compute the response of the detector at each pixel (R)
6. Threshold on value of R, compute non-max suppression
7. For each pixel that meets the criteria in 6, compute a feature descriptor (Speak about descriptor a little later).

Harris corner response is rotation invariant

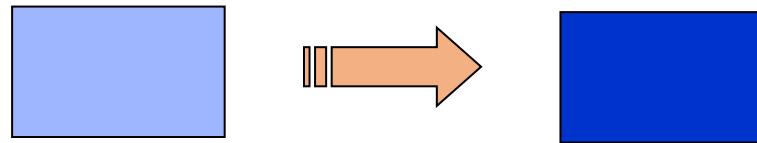


Ellipse rotates but its shape
(eigenvalues) remains the same

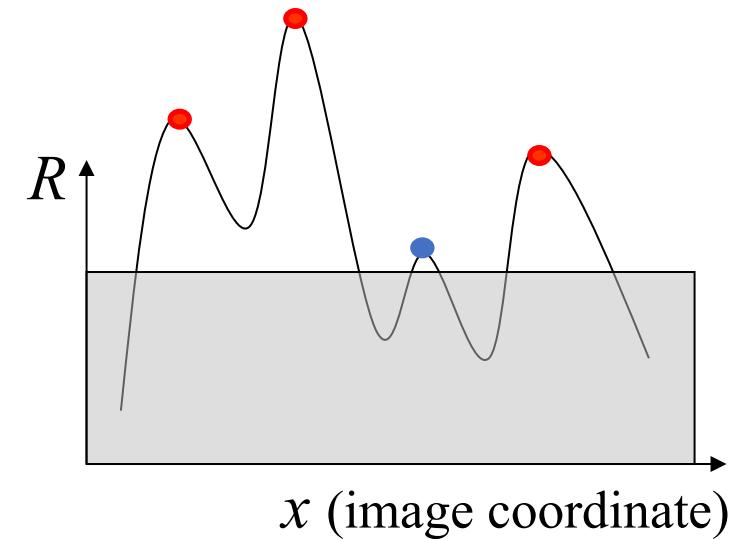
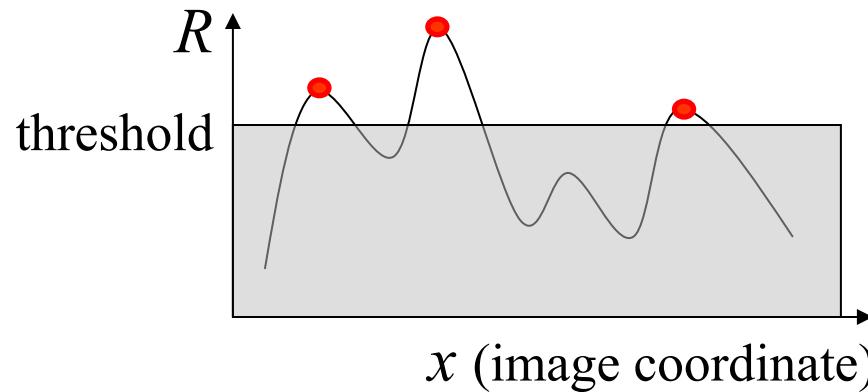
Corner response \mathbf{R} is invariant to image rotation

Intensity changes

Partially invariance to affine intensity change



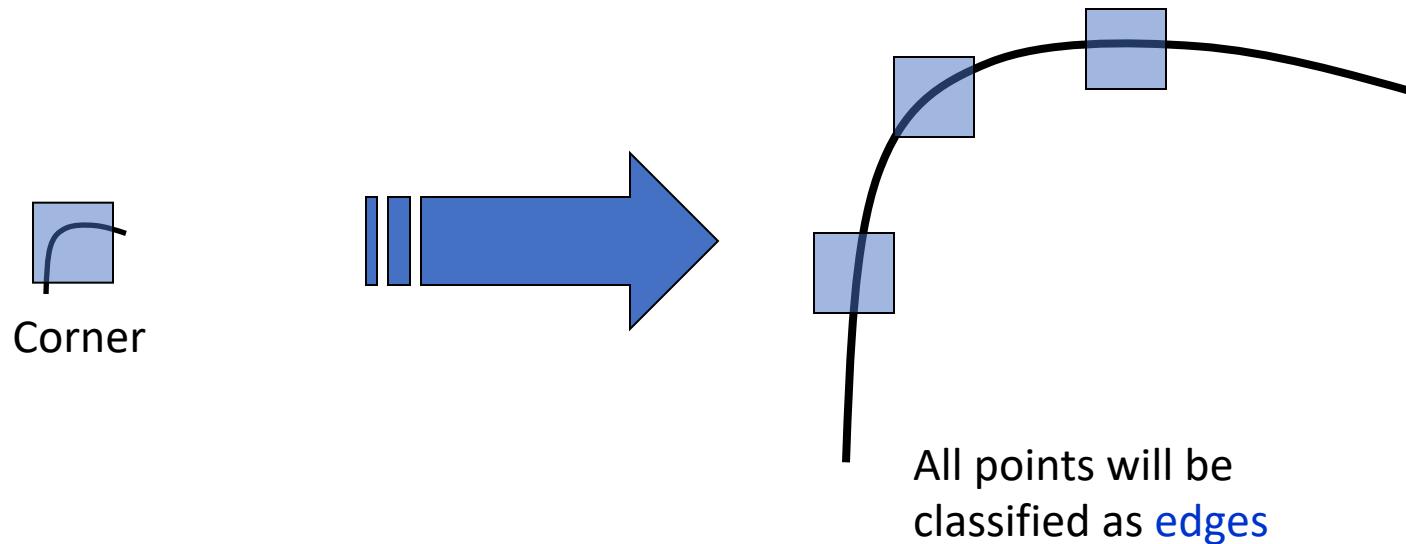
- Only derivatives are used => invariance to intensity shift ($I \rightarrow I+b$)
- Intensity scaling: $I \rightarrow a I$



Harris Corner Detection



Scaling



*Harris detector **not invariant** to changes in scaling*

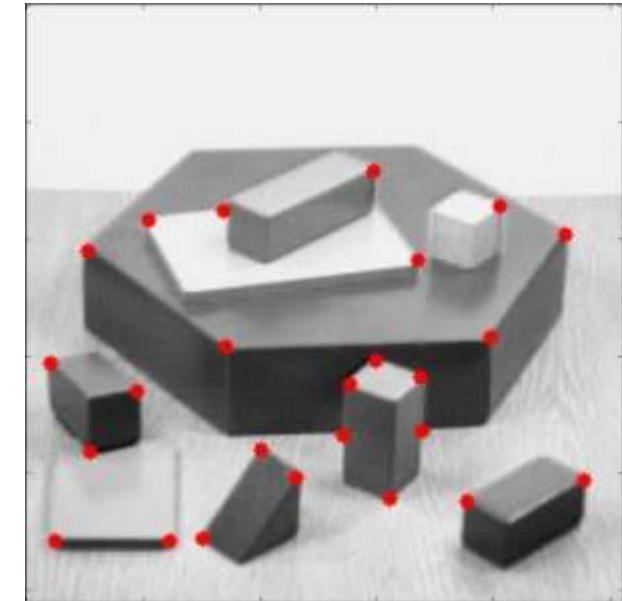
Shi-Tomasi Corner Detector

- Harris Corner Detector was given by:

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

- A small modification to it in their paper **Good Features to Track** which shows **better results** compared to Harris Corner Detector

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

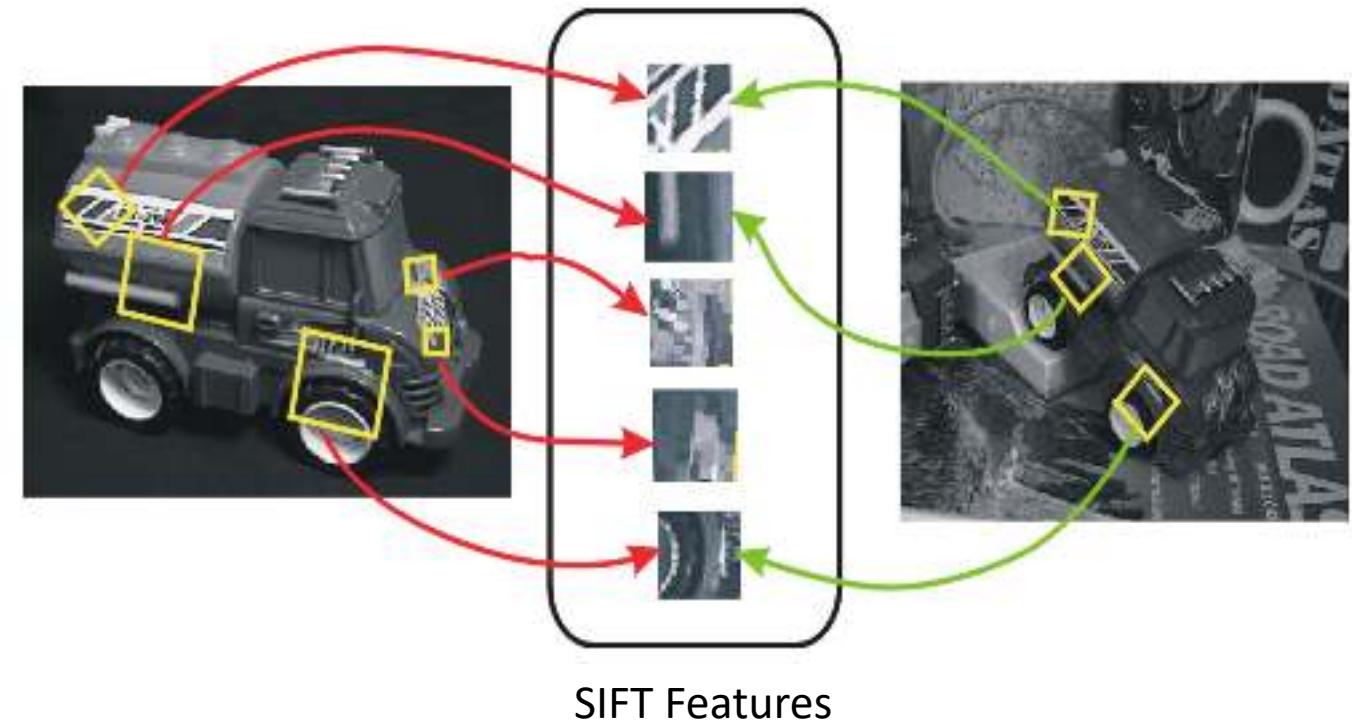


If it is greater than a threshold value, it is considered as a corner

Scale Invariant Feature Transform (SIFT) – David Lowe (ICCV 1999)

Scale Invariant Feature Transform (SIFT)

- SIFT is local feature **detector** and **descriptors**
- It is **reasonably invariant** to changes in
 - illumination
 - image noise
 - rotation
 - scaling
 - small changes in viewpoint



Journal + conference versions: 60,000+ citations
[Lowe, ICCV 1999]

Patented: University of British Columbia (Canada).

Detection Stages for SIFT Features

1. Scale-space extrema detection

- Potential locations for finding features

2. Key point localization

- Accurately locating the feature keypoints

3. Orientation assignment

- Assigning orientation to the keypoints

Detector

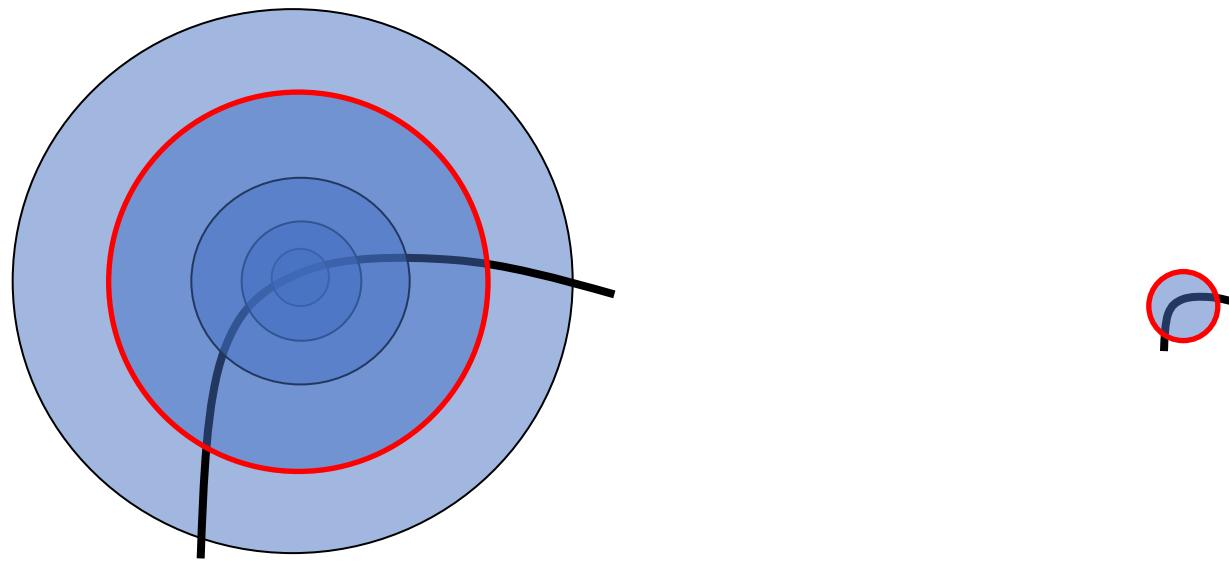
4. Keypoint descriptor

- Describing the keypoints as a high dimensional vector.

Descriptor

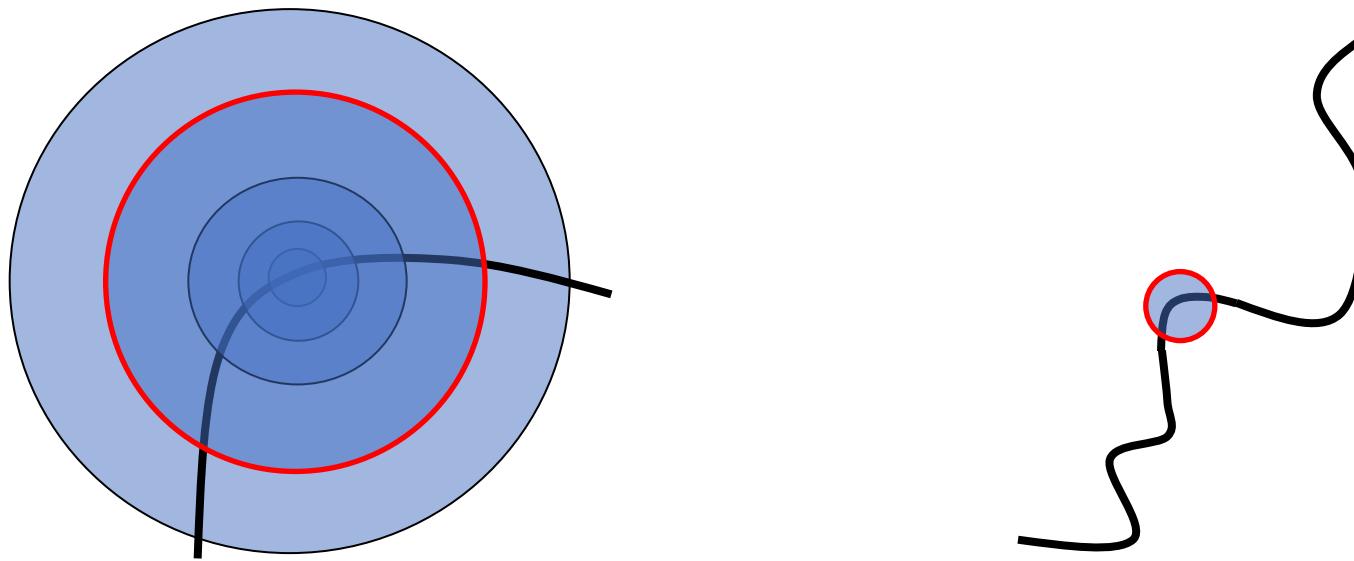
1. Scale-space Extrema Detection

Scale Invariant Detection



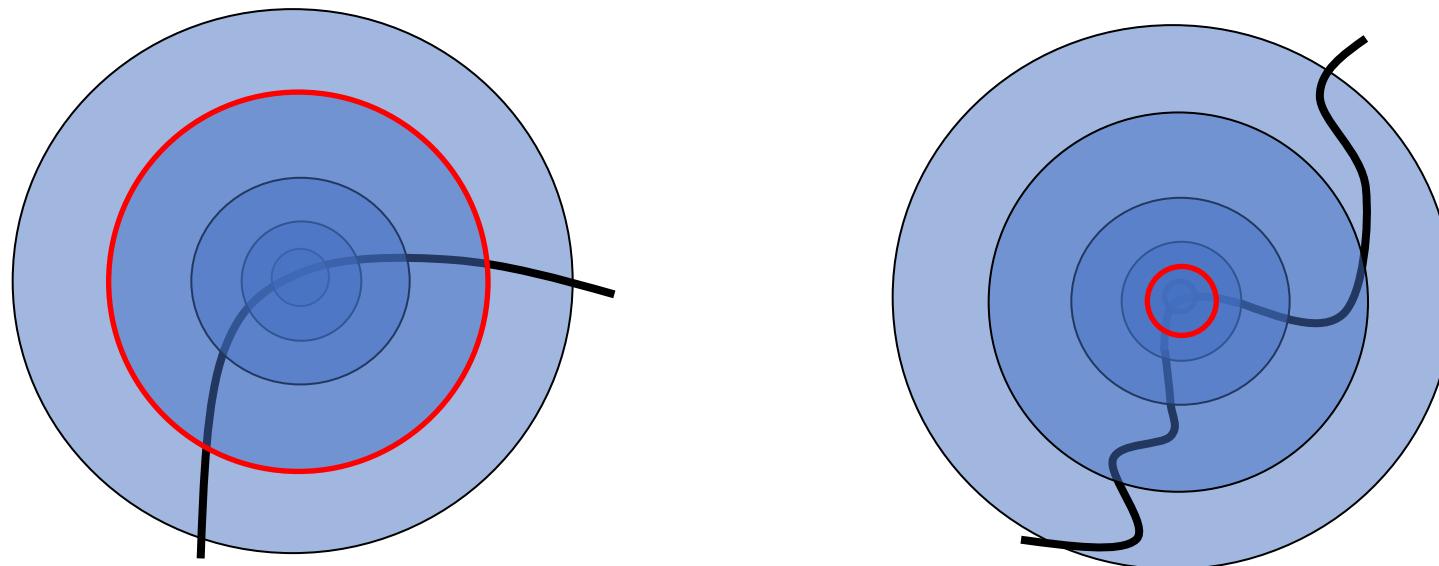
1. Scale-space Extrema Detection

Scale Invariant Detection



1. Scale-space Extrema Detection

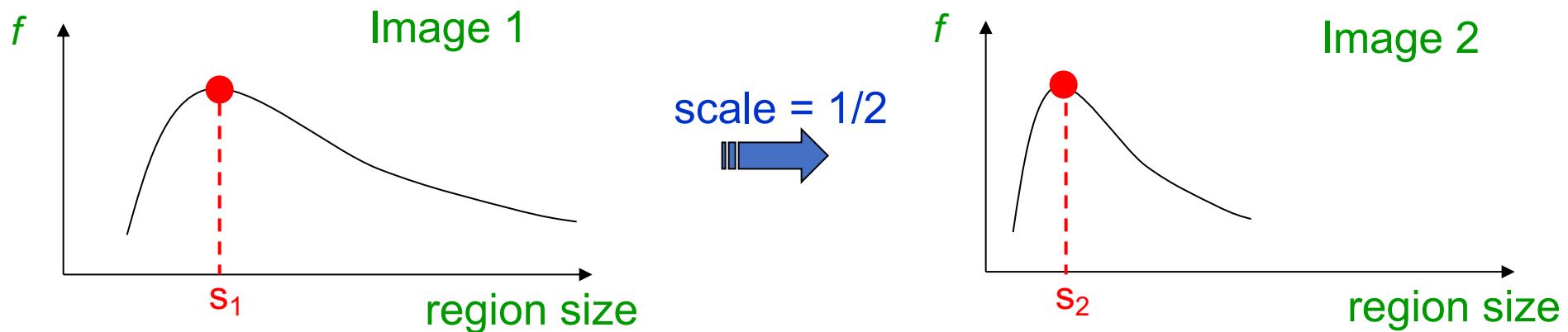
Scale Invariant Detection



- How do we choose corresponding windows **independently** in each image?
- Do objects have a characteristic scale that we can identify?

1. Scale-space Extrema Detection

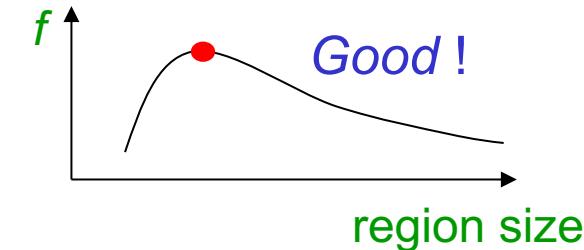
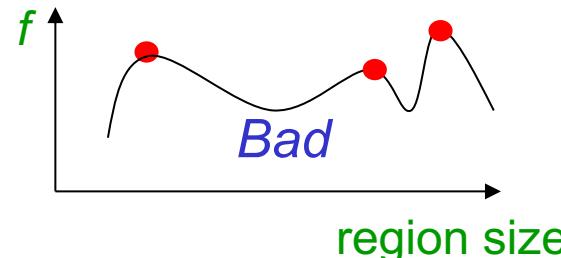
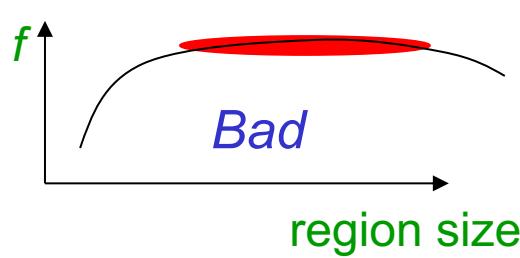
- **Solution:**
 - Design a **function** on the region which has the **same shape** even if the image is resized
 - Take a **local maximum** of this function



Source: A. Torralba

1. Scale-space Extrema Detection

- A “good” function for scale detection has one stable sharp peak



1. Scale-space Extrema Detection

- We need to identify those locations and scales that are identifiable from different views of the same object.
- This can be efficiently achieved using a "scale space" function.
- **Reasonable assumption:** it must be based on the Gaussian function and define as:

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

- Where:
 - * is the convolution operator
 - $G(x, y, \sigma)$ is a variable-scale Gaussian and
 - $I(x, y)$ is the input image.

1. Scale-space Extrema Detection

- Laplacian of Gaussians is one such technique to locate scale-space extrema
- Calculation costly...
- What is the solution?
 - Approximate LoG

1. Scale-space Extrema Detection

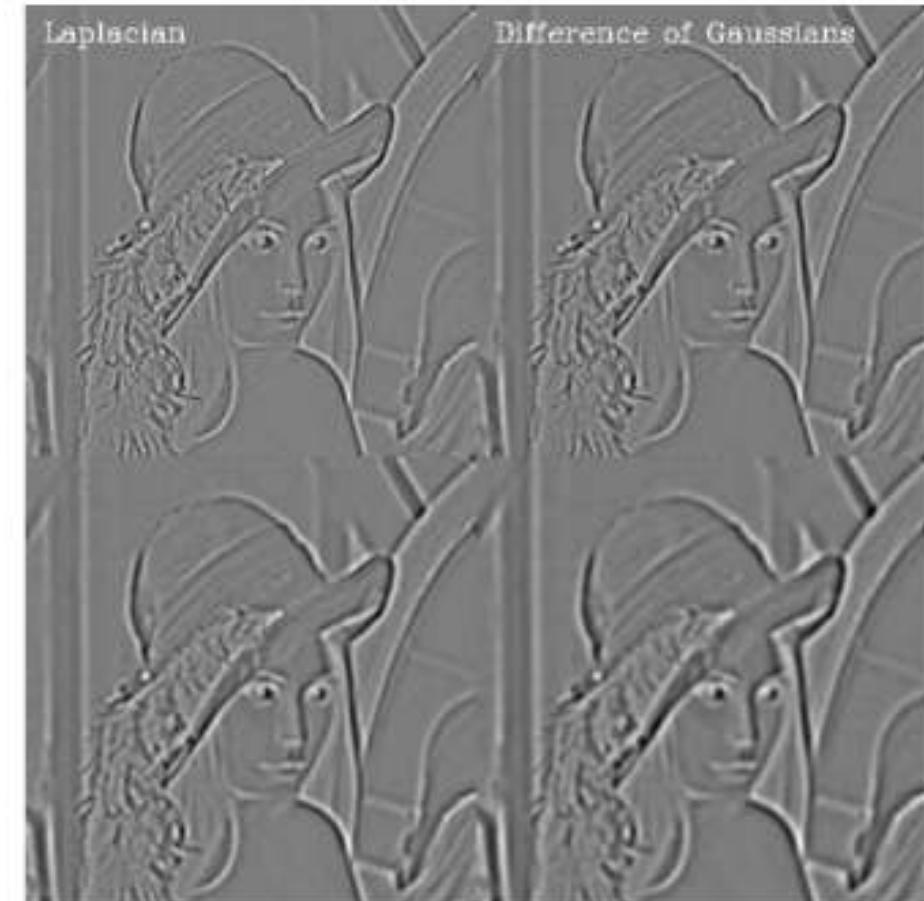
- **Approximation of Laplacian of Gaussians**

- Difference of Gaussians (DoG):

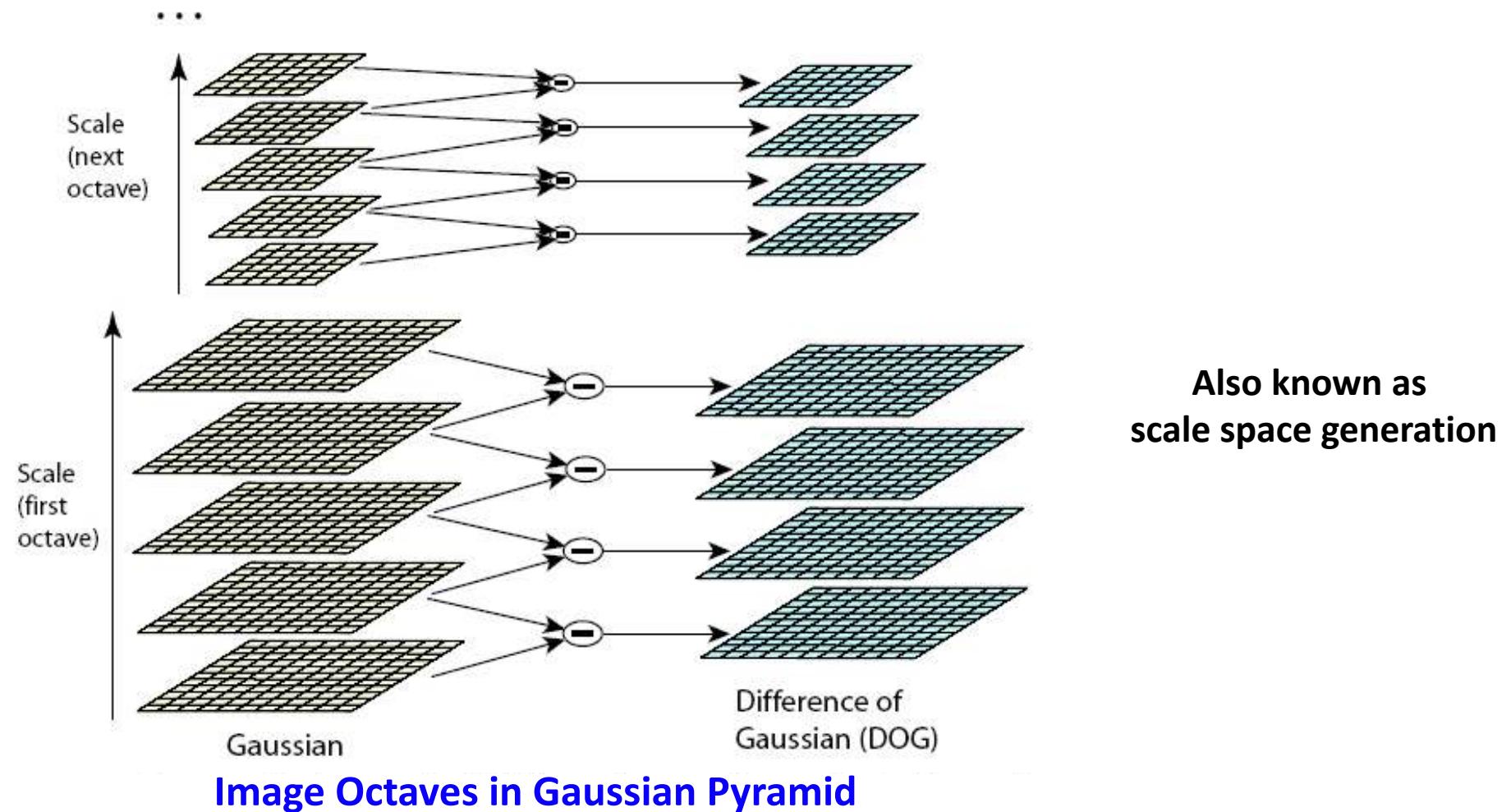
$$D(x, y, \sigma) = L(x, y, k\sigma) * L(x, y, \sigma)$$

Where,

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



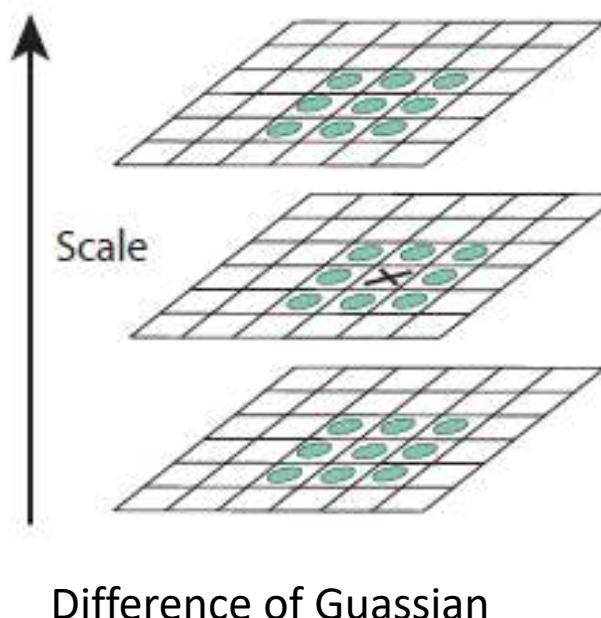
1. Scale-space Extrema Detection



SIFT suggests that 4 octaves and 5 blur levels are ideal for the algorithm.

1. Scale-space Extrema Detection

- Images are searched for **local extrema** over scale and space.



- One pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a **local extrema**, it is a **potential keypoint**.

2. Keypoint Localization

- Now we have much less points than pixels.
- However, still lots of points (~1000s)...



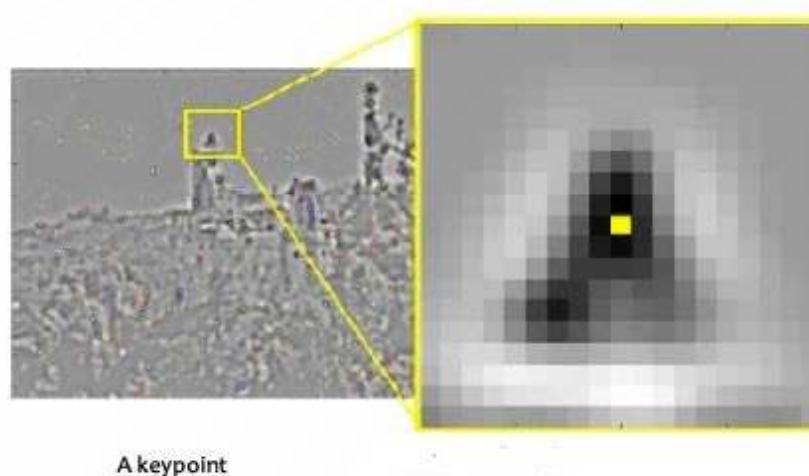
Source: Brown & Lowe 2002

2. Keypoint Localization

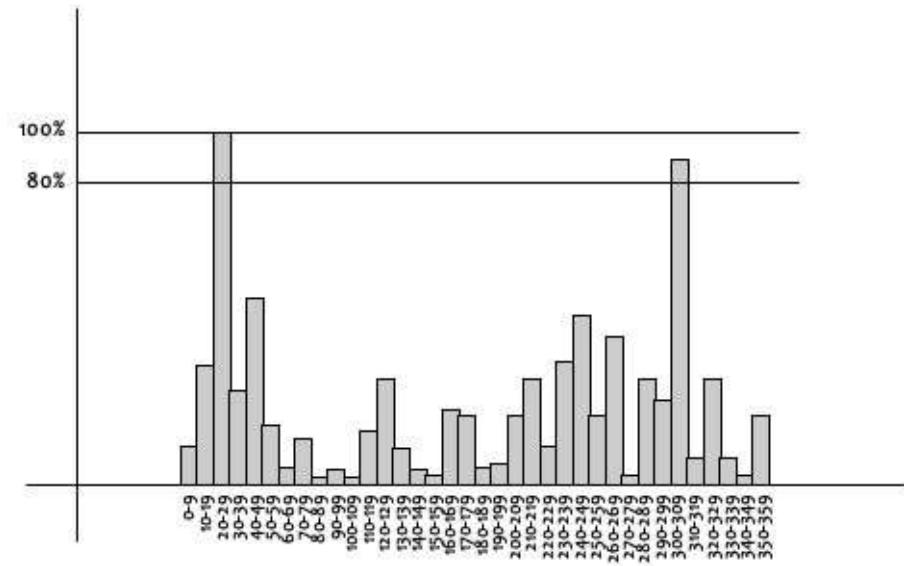
- SIFT also used Taylor series expansion of scale space to get more accurate location of extrema
 - If the intensity at extrema < assigned value then rejected (**region flat**)
- DoG has higher response for edges (**need to be removed**)
 - A concept similar to Harris corner detector is used – Hessian matrix (H) to compute the principal curvature.
 - We know from Harris corner detector that for edges – One eigen value is larger than the other.
 - If this ratio is greater than a **threshold** that keypoint is discarded

3. Orientation Assignment

- After localization, we have stable keypoints
- An orientation is assigned to each keypoint.
- A neighborhood is taken around the key point location depending on the scale, and the gradient magnitude and direction is calculated in that region.
- An orientation histogram with 36 bins covering 360 degrees is created.



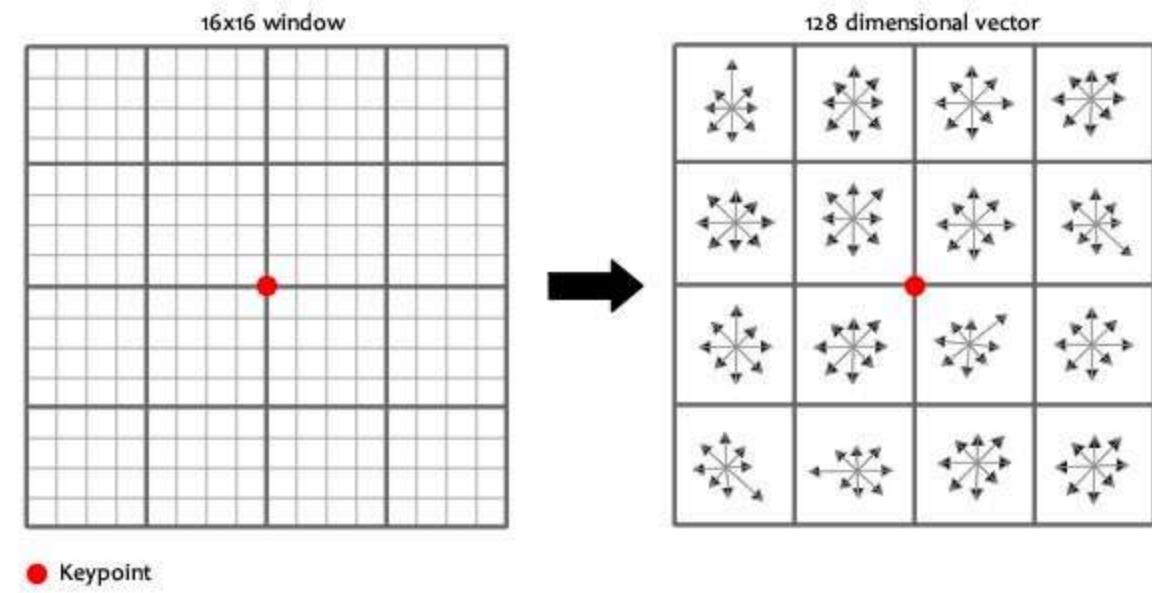
Source: Deepanshu Tyagi



4. Keypoint Descriptor

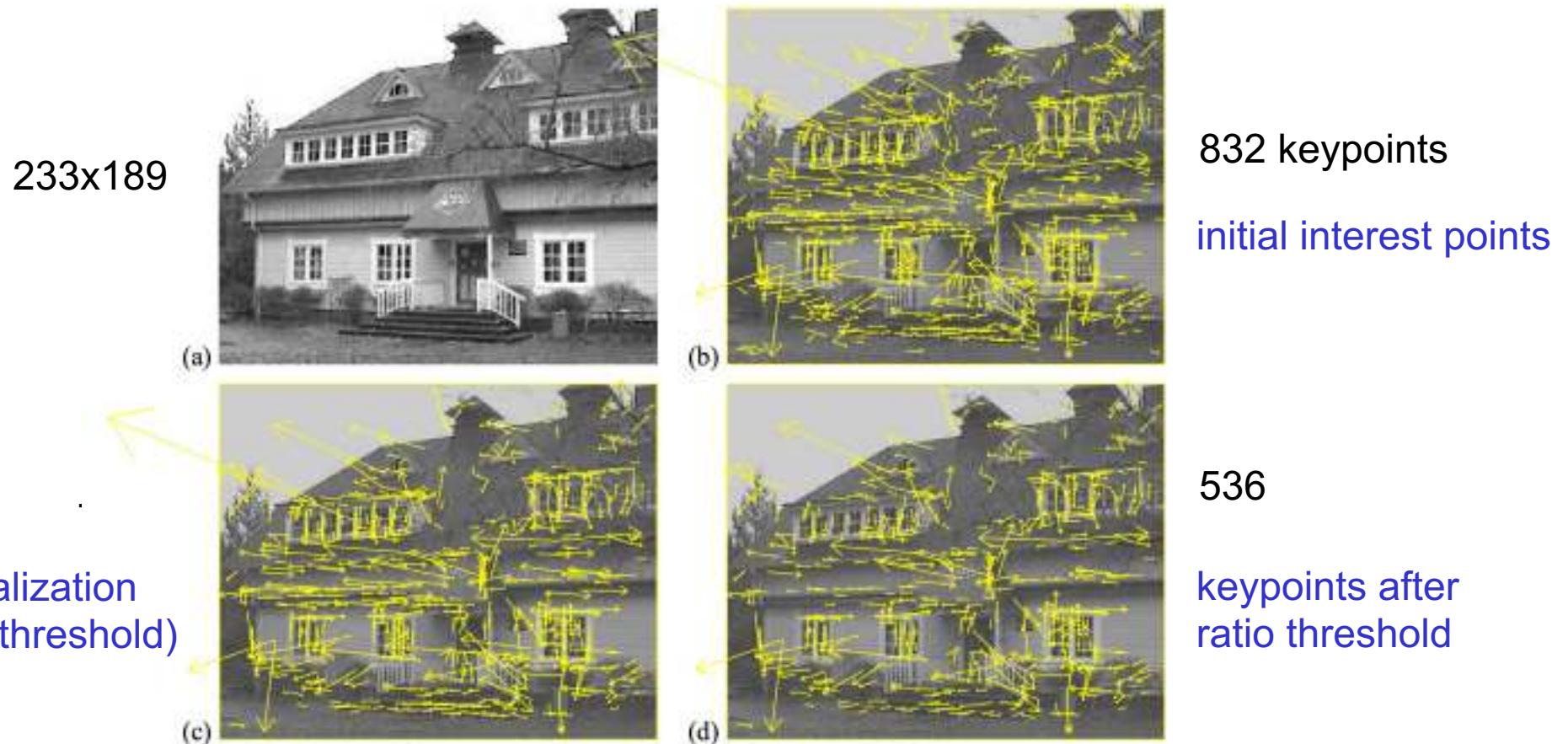
- At this point, each **keypoint** has a location, scale, orientation
- Now we can **compute a descriptor** for the local image region about each key point
- **16x16 window** around the keypoint is taken. It is divided into **16 sub-blocks** of **4x4** size

- Based on **16*16** patches
- **4*4** sub regions
- 8 bins in each sub region
- $4*4*8=128$ dimensions in total



Source: Deepanshu Tyagi

Stages of keypoint selection



SIFT is Robust

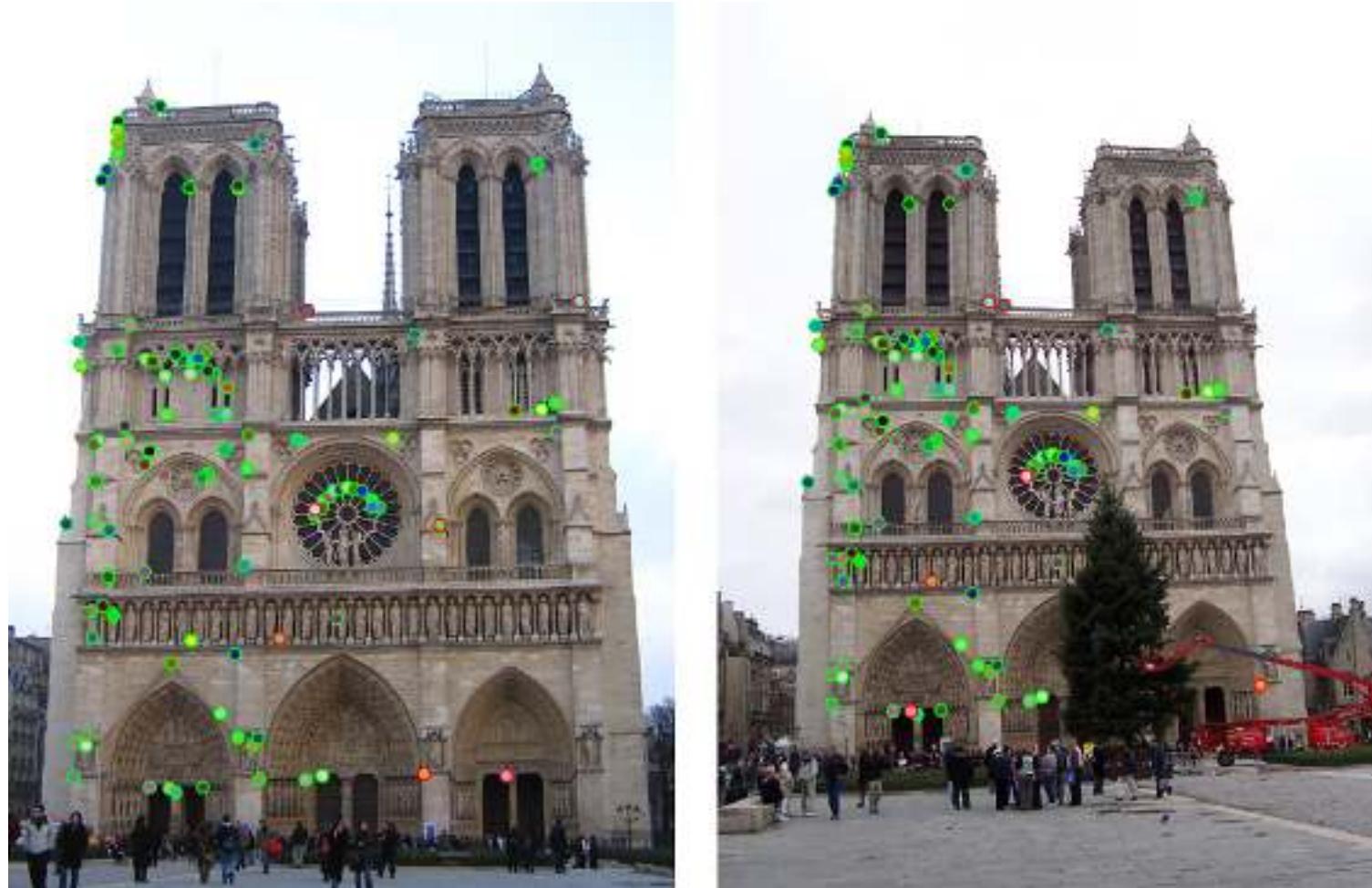
- Extraordinarily robust matching technique
- Can handle changes in viewpoint
 - Up to about 60 degree
- Can handle significant changes in illumination
 - Sometimes even day vs. night
- Fast and efficient
 - Can run in real time

Other Detectors and Descriptors

- SURF
 - Approximate SIFT
 - Works almost equally well
 - Very fast
- HOG: Histogram of Gradients (HOG)
 - Sliding window, pedestrian detection
- FREAK: Fast Retina Keypoint
 - Perceptually motivated
 - Used in Visual SLAM
- LIFT: Learned Invariant Feature Transform
 - Learned via deep learning

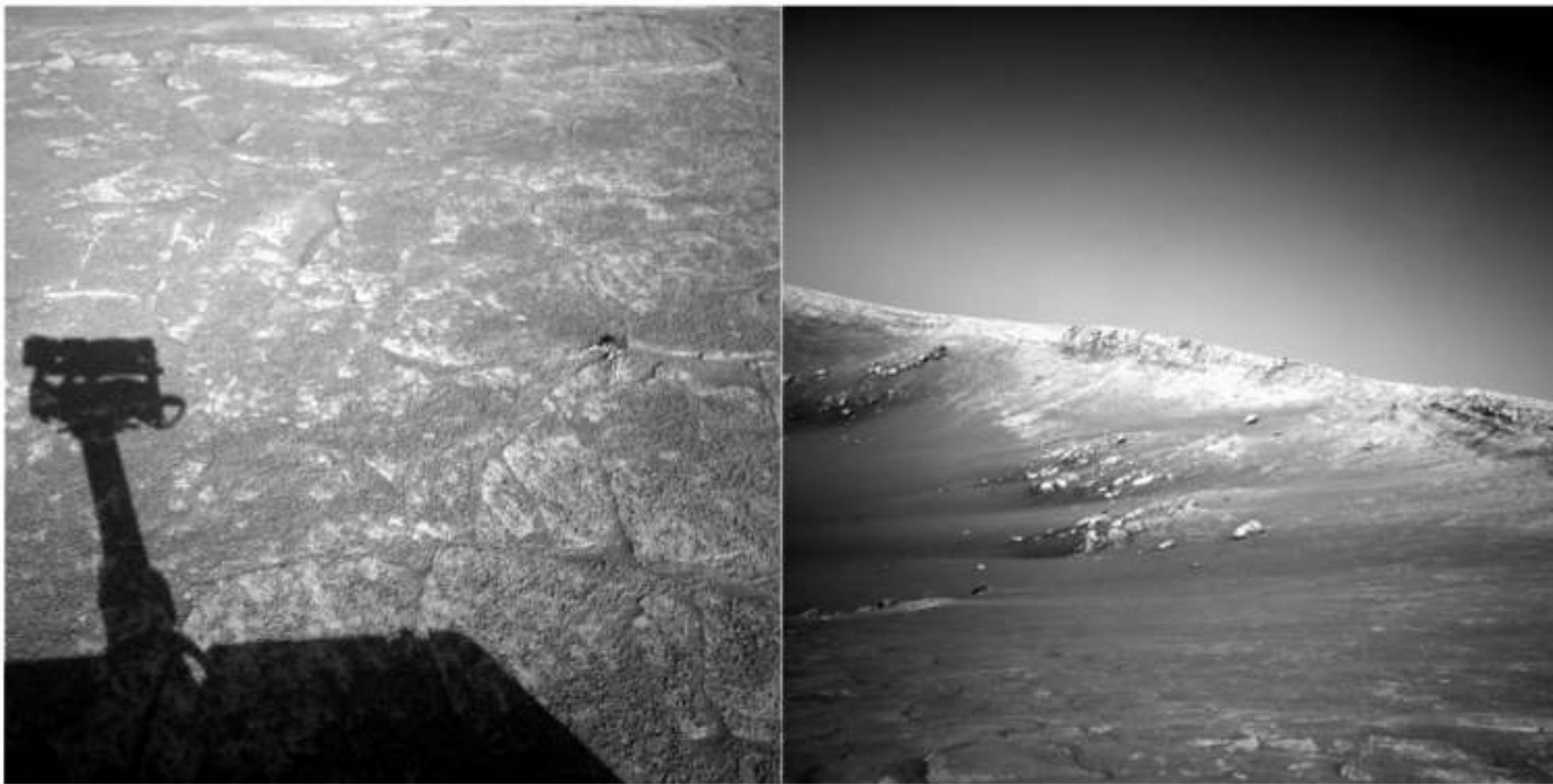
Feature Matching

Which Features Match?



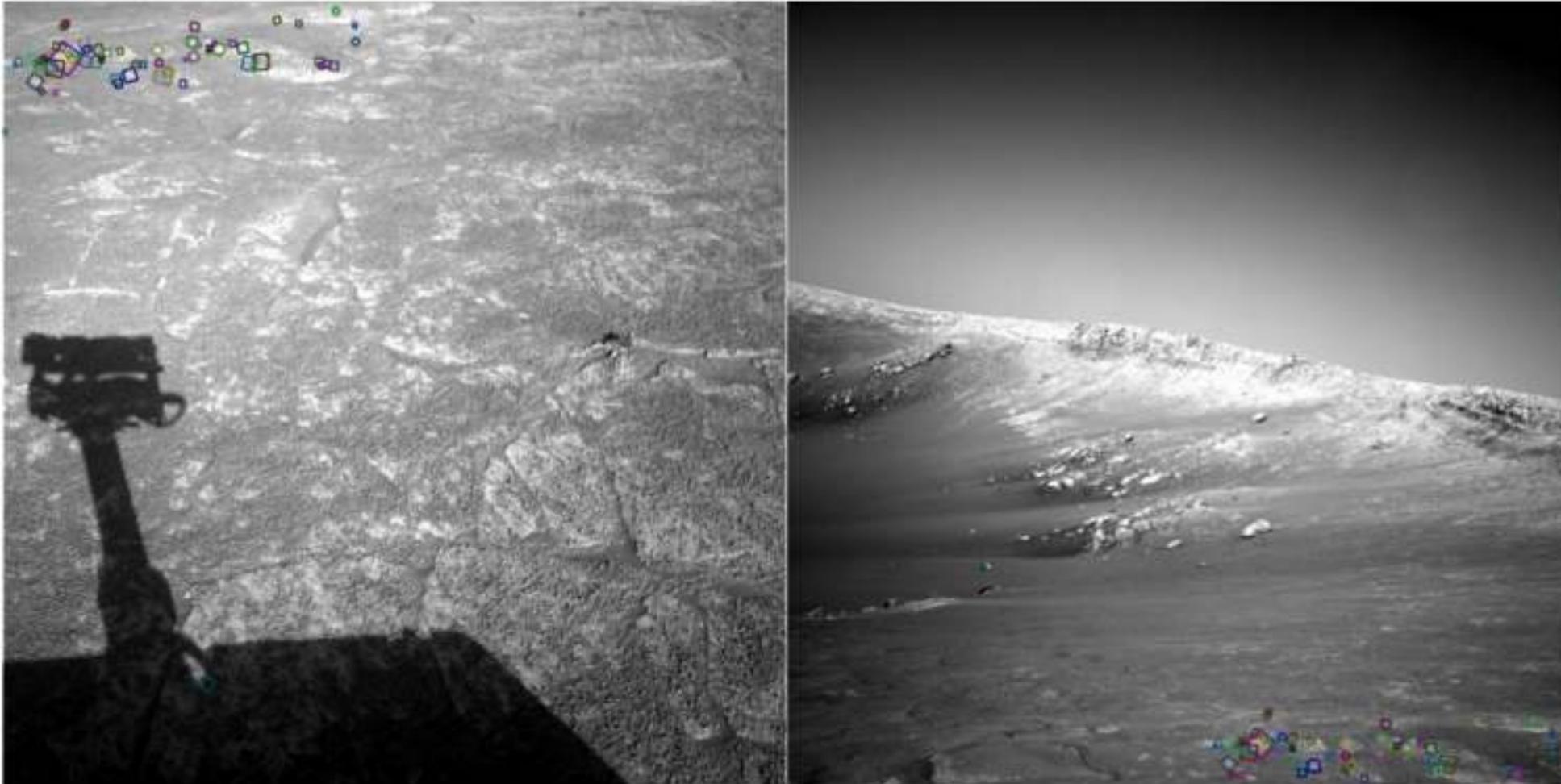
Source: Noah Snavely

Harder Case



Source: Noah Snavely

Harder Case



Source: Noah Snavely

NASA Mars Rover images with SIFT feature matches

Feature Matching

- Keypoints between two images are matched using Nearest Neighbor algorithm based on L_2 distance
- How to discard bad matches?
 - Threshold on $L_2 \Rightarrow$ bad performance
 - **Solution:** threshold on ratio
 - The second closest-match may be very near to the first.
 - In this case, ratio of closest-distance to second-closest distance is taken.

Feature Matching

Given a feature in I_1 , how to find the best match in I_2 ?

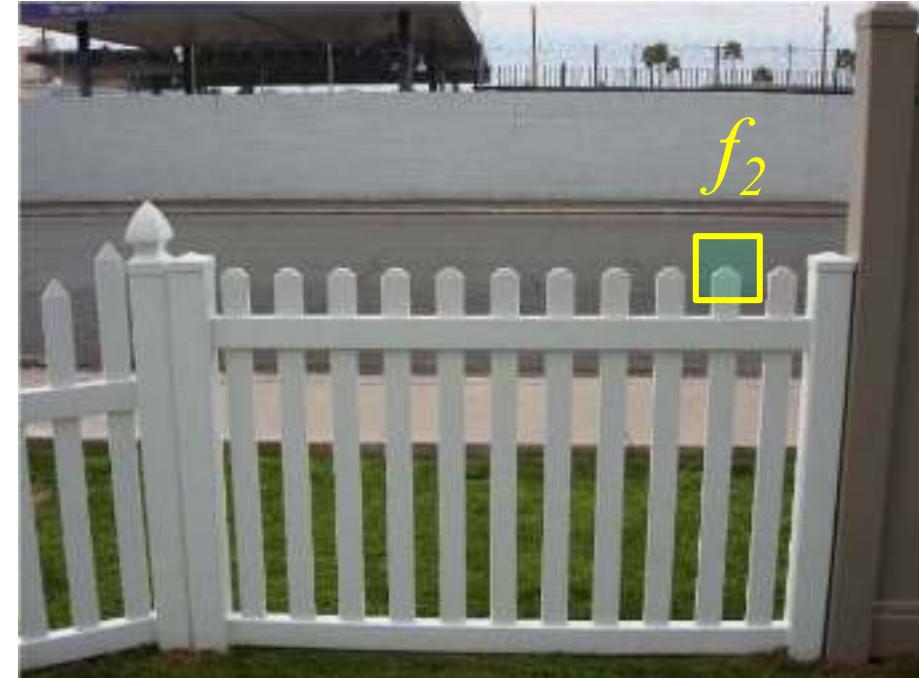
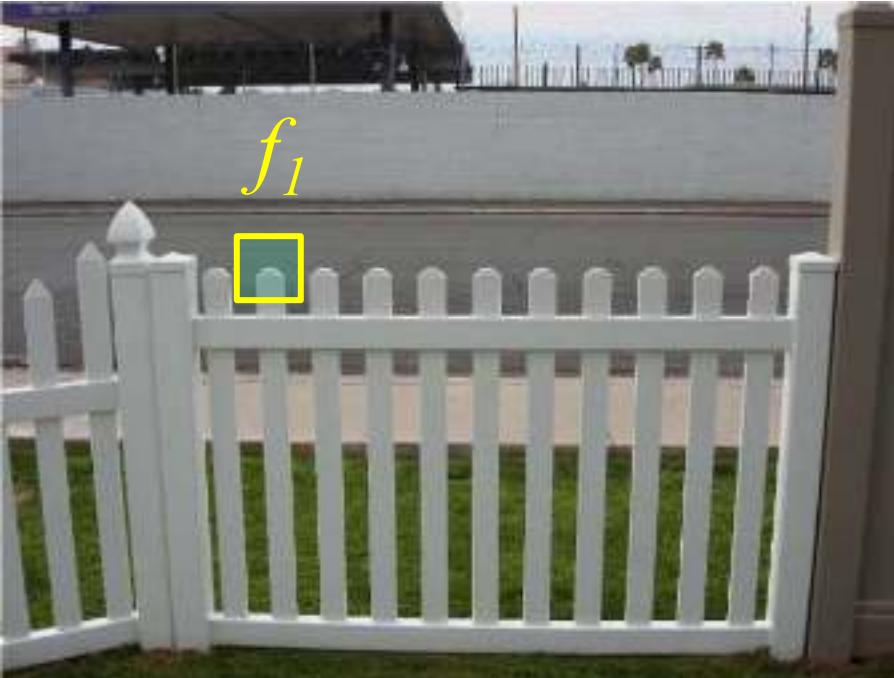
1. Define **distance function** that compares **two descriptors**
2. Test all the features in I_2
3. Find the one with **min distance**

Feature Distance

How to define the difference between two features f_1, f_2 ?

- **Simple Approach**

- L_2 distance = $\|f_1 - f_2\|$
- It can give small distances for ambiguous (incorrect) matches



Source: Noah Snavely

I_1

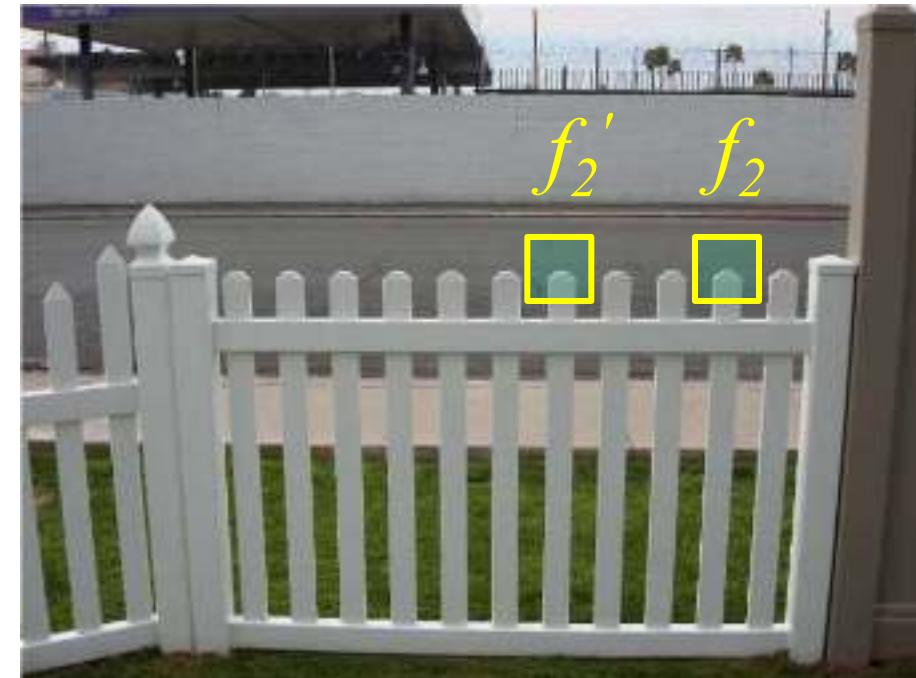
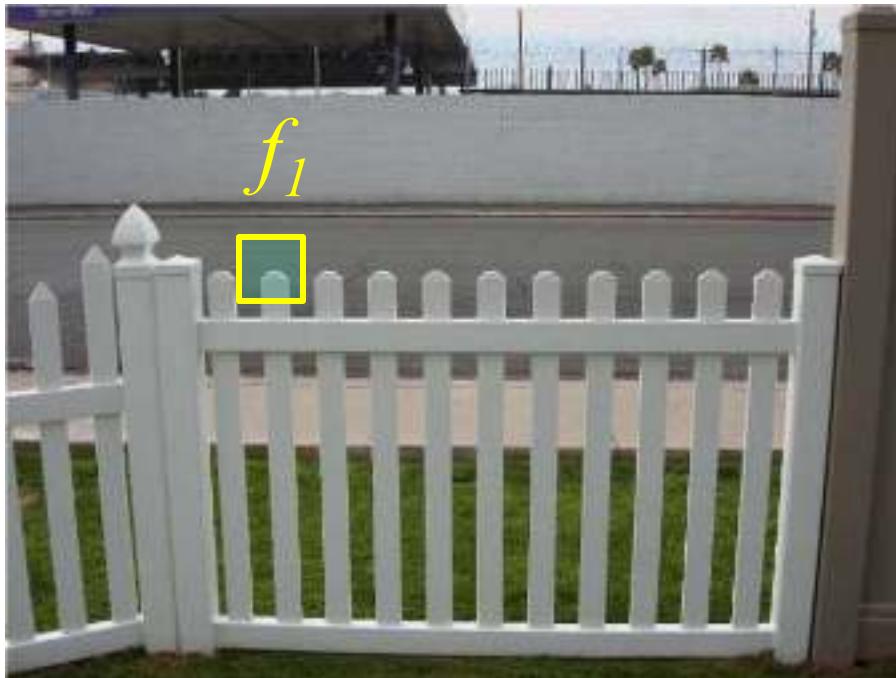
I_2

Feature Distance

How to define the difference between two features f_1, f_2 ?

- **Better Approach**

- Ratio Distance = $\|f_1 - f_2\| / \|f_1 - f'_2\|$
- f_2 is best match to f_1 in I_2
- f_2' is 2nd best match to f_1 in I_2
- Gives large values for ambiguous matches

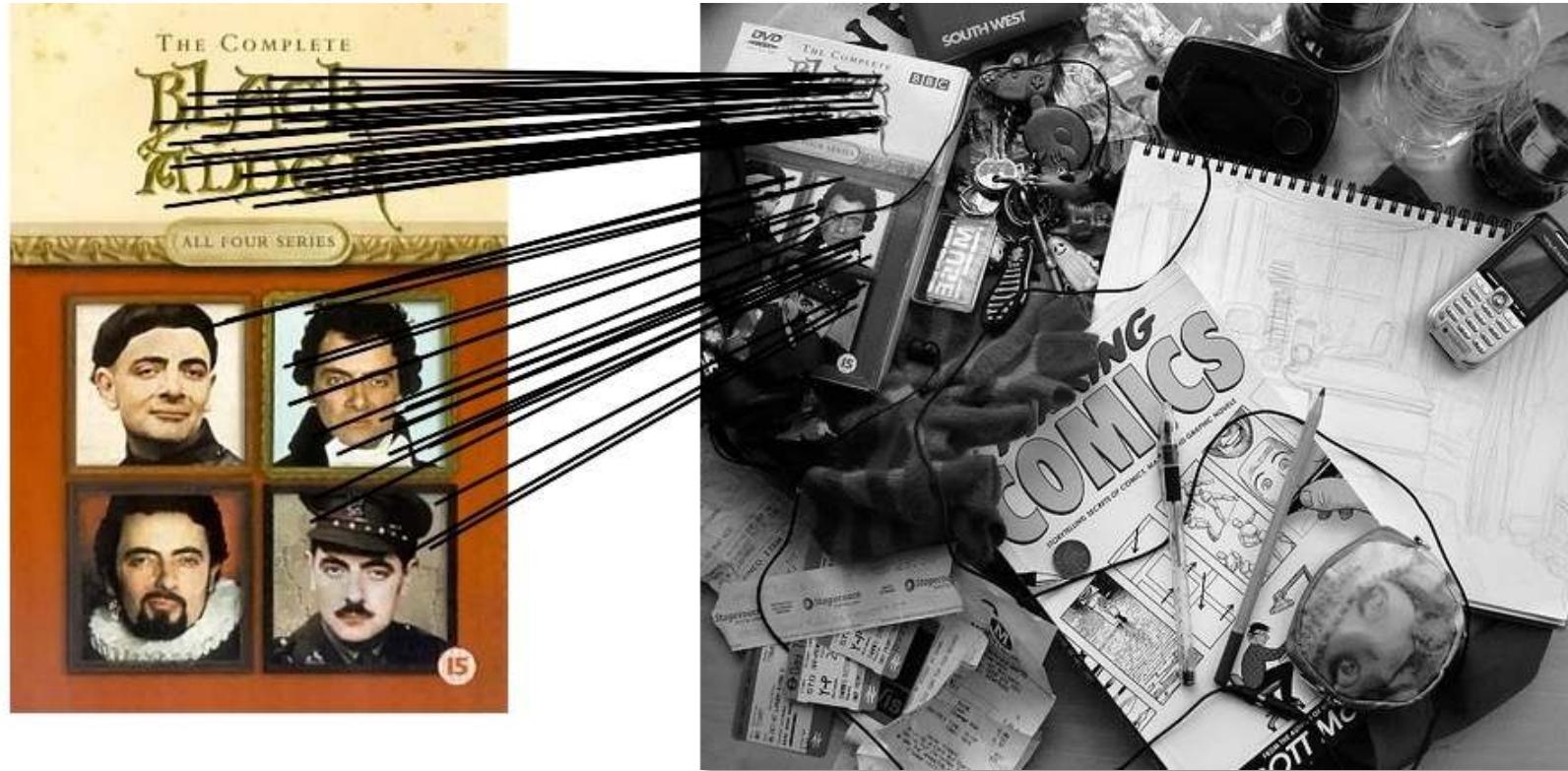


Source: Noah Snavely

I_1

I_2

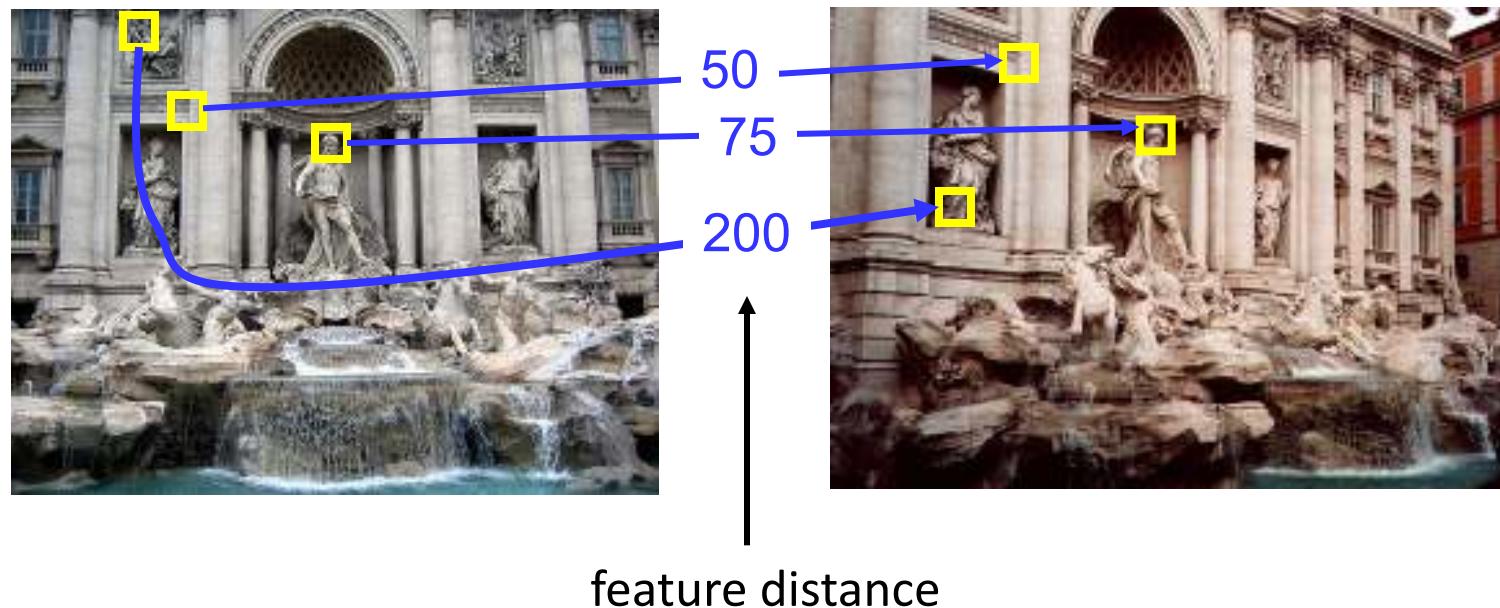
Feature Matching Example



Source: Noah Snavely

Evaluating the Results

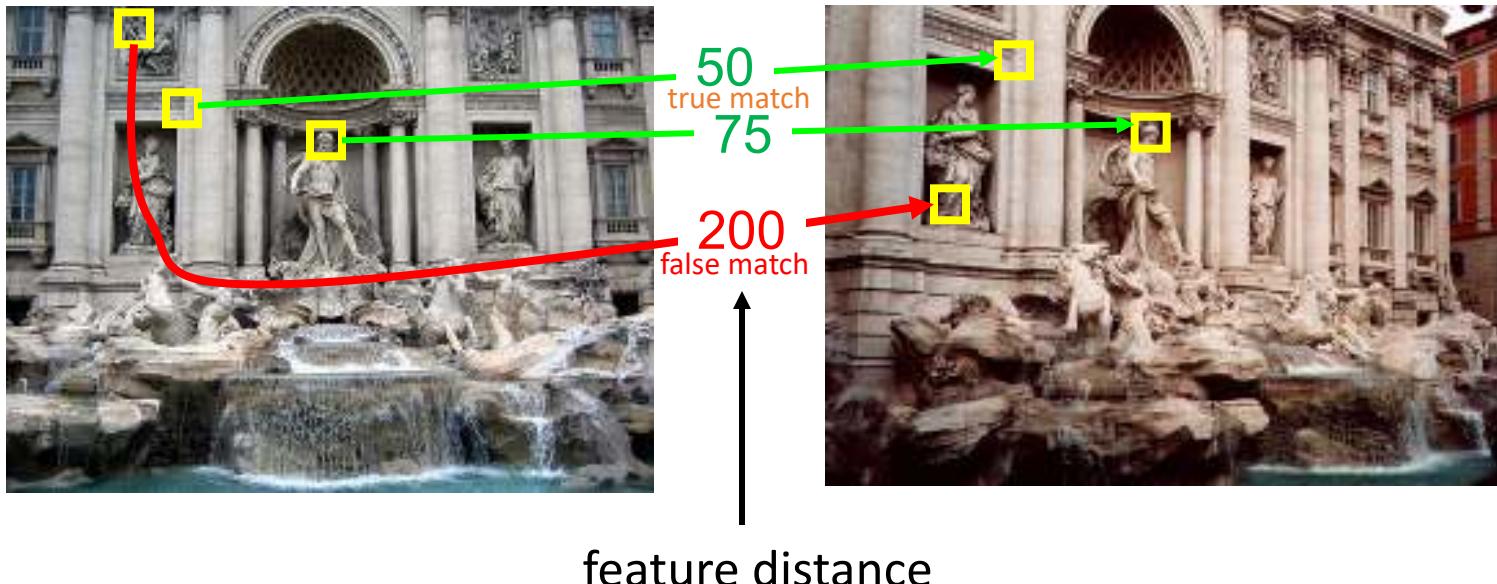
- How can we measure the performance of a feature matcher?



Source: Noah Snavely

True/False Positives

- How can we measure the performance of a feature matcher?



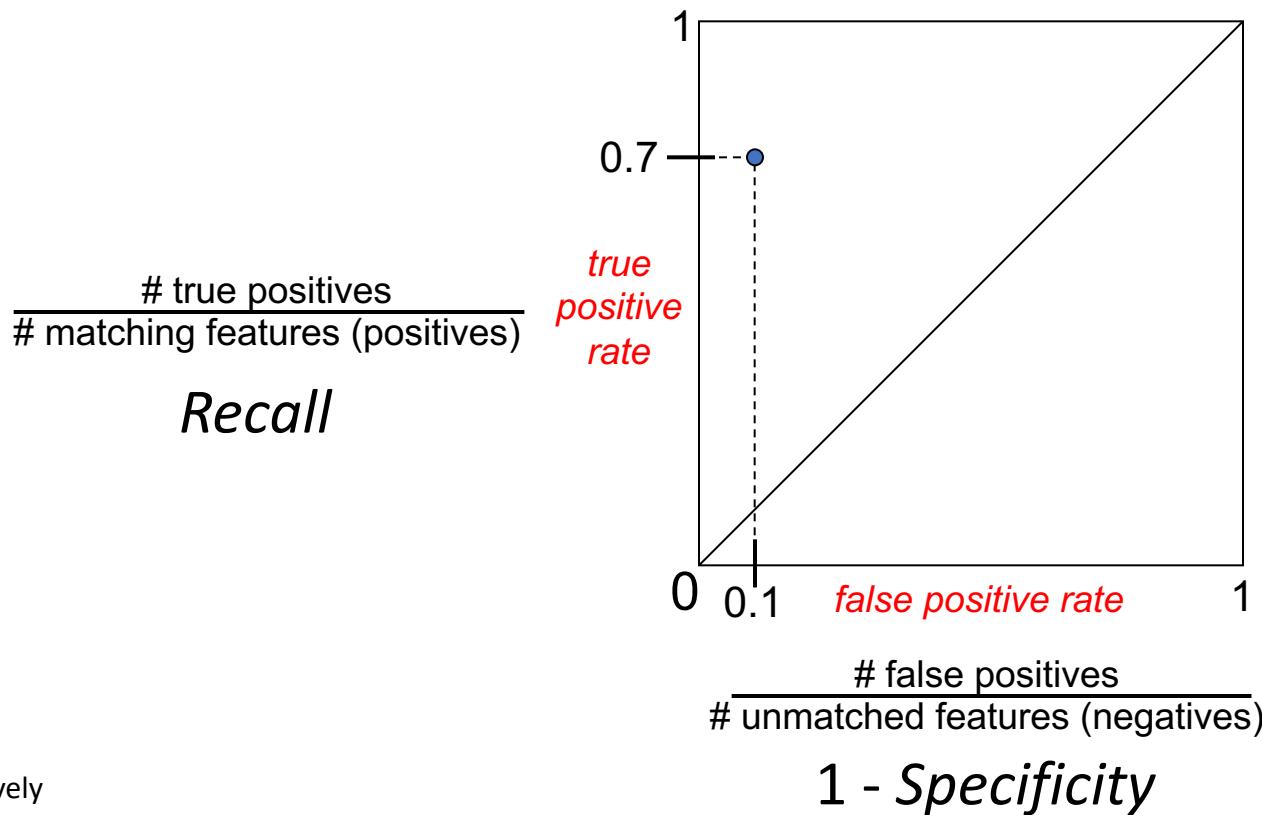
The distance threshold affects performance

- True positives = # of detected matches that are correct
 - Suppose we want to maximize these
- False positives = # of detected matches that are incorrect
 - Suppose we want to minimize these

Source: Noah Snavely

Evaluating the results

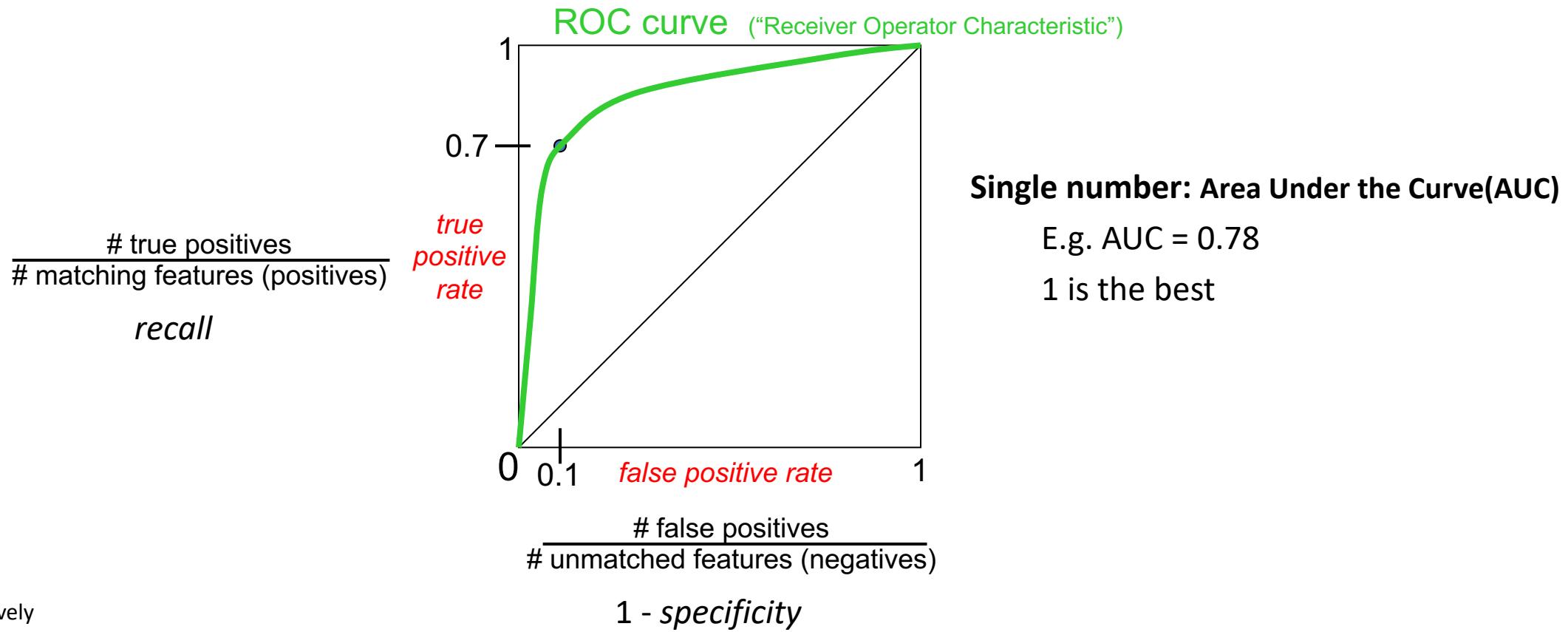
- How can we measure the performance of a feature matcher?



Source: Noah Snavely

Evaluating the results

- How can we measure the performance of a feature matcher?



Source: Noah Snavely

Lots of Applications

Features are used for:

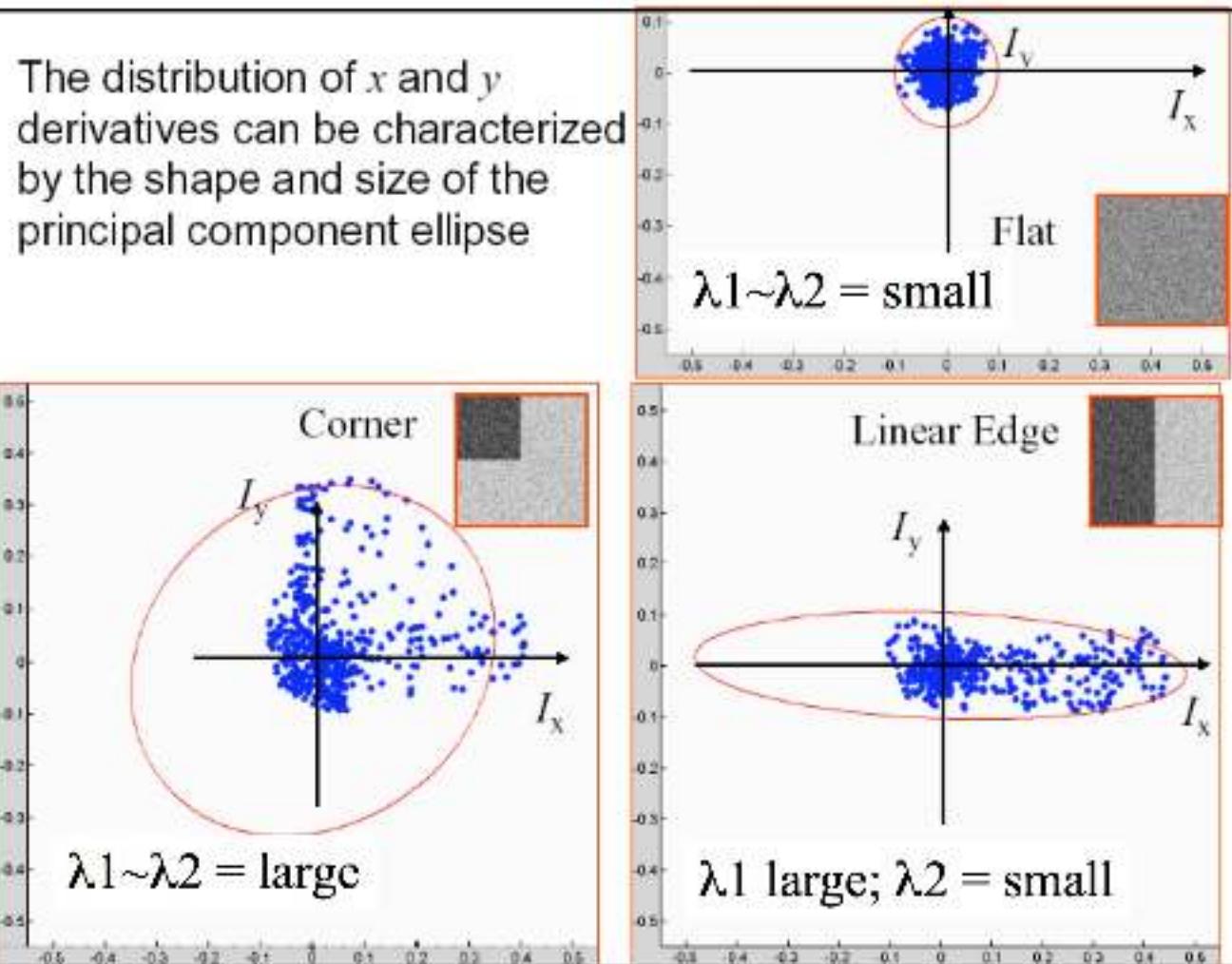
- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- ... other

Reading

- Kenneth: Section 7.2 and 7.4

Q & A

Fitting Ellipse to each Set of Points



Source Credit: Robert Collins

Feature Extraction Part II

Bag of Visual Words Model

(Recognizing object categories)

By: Dr. Muhammad Fahim

Contents

- Origin of Bag of Words
- Bag of Visual Words Model
 - Extract Features
 - Codebook Generation
 - Quantization
- Image classification
- Summary

Source of the material

- This lecture is based on the following resources
 - Lecture slides of *Prof. Adil Khan*, Innopolis University.
 - Adapted from slides by Rob Fergus and Svetlana Lazebnik
 - Computer Vision Carnegie Mellon University (Kris Kitani)
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - <https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb>
 - Found material over the internet to aligned the subject according to the need of the students.

Problem: Image Classification

- **Given:**

- Positive training images containing an object class, and



- Negative training images that don't



- **Classify:**

- A test image as to whether it contains the object class or not



?

Weakly-supervised Learning

- Learn model from a **set of training images** containing object instances



- Know if image **contains object** or not
- But **no segmentation** of object

Three stages

1. Represent each training image by a vector
 - Use a bag of visual words representation ([Today's topic](#))
2. Train a classifier to discriminate vectors corresponding to positive and negative training images
3. Apply the trained classifier to the test image

Bag of Visual Words (BOVW)

Origin 1: Texture

- Texture depicts spatially repeating patterns
- Many natural phenomena are textures



Radishes



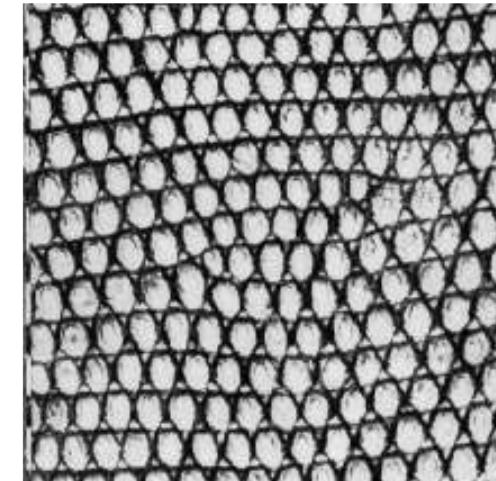
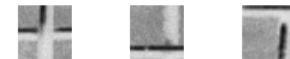
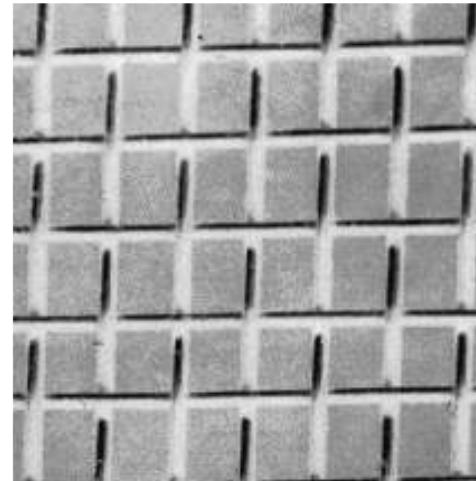
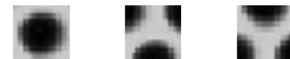
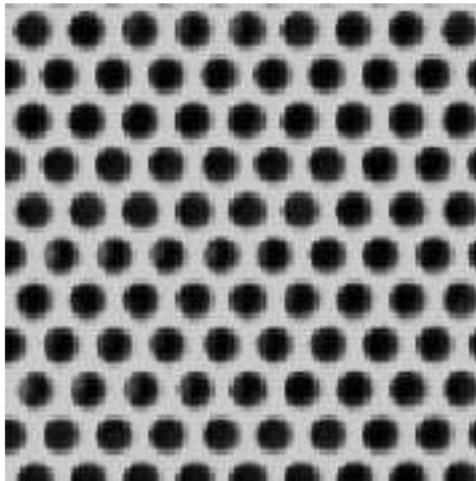
Rocks



Yogurt

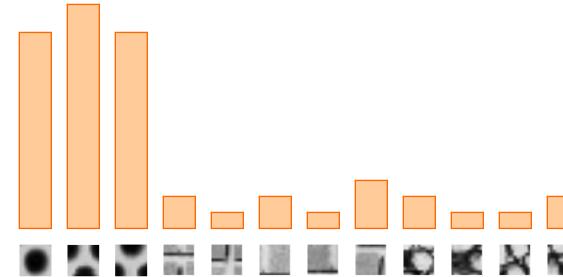
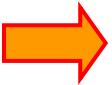
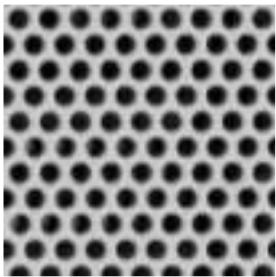
Origin 1: Texture Recognition

- Texture is characterized by the repetition of basic elements or *textons*
- For stochastic textures, it is the identity of the textons, not their spatial arrangement, that matters

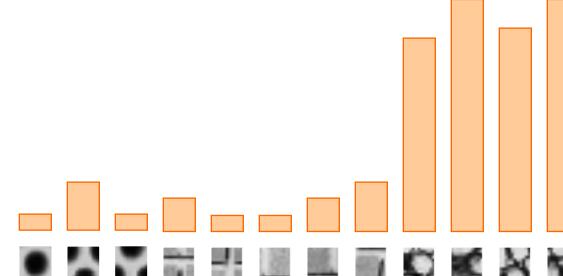
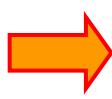
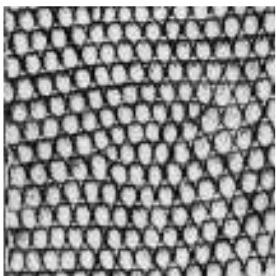
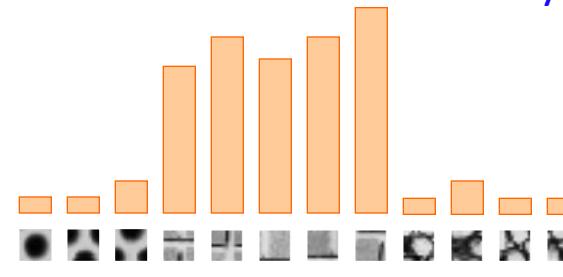
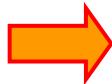
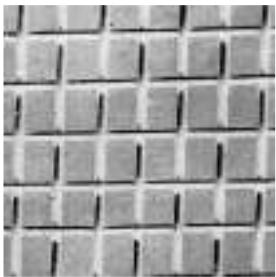


Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

Origin 1: Texture Recognition



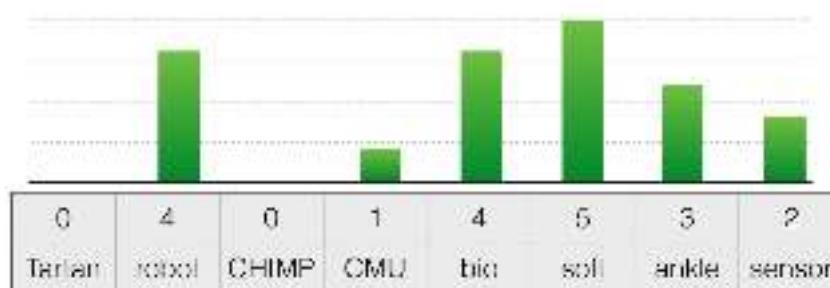
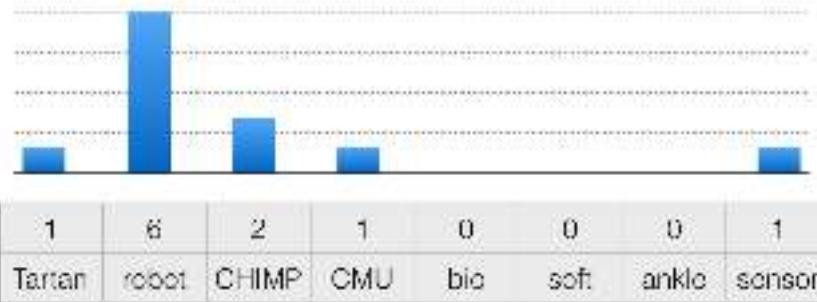
Universal texton dictionary



Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

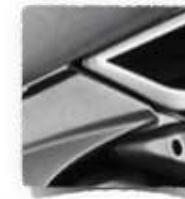
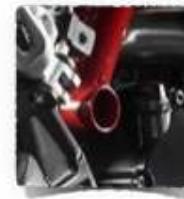
Origin 2: Bag-of-words models

- **Orderless document representation:** frequencies of words from a dictionary



Vector Space Model

What object do these parts belong to?



What object do these parts belong to?

- Some local feature are very informative



Source: Kris Kitani

Bag of Visual Words Model

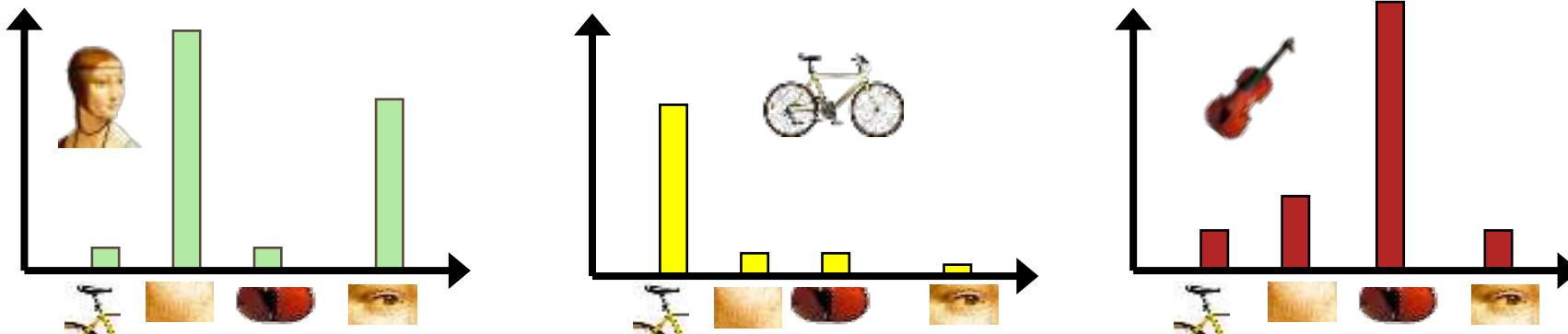
- One of the **most important** concepts in computer vision
- Used to build **accurate system** for:
 - Image classification
 - Content based image retrieval
 - Many others...

Bag of Visual Words (BOVW)



Source: Svetlana Lazebnik

Bag of Visual Words (BOVW)



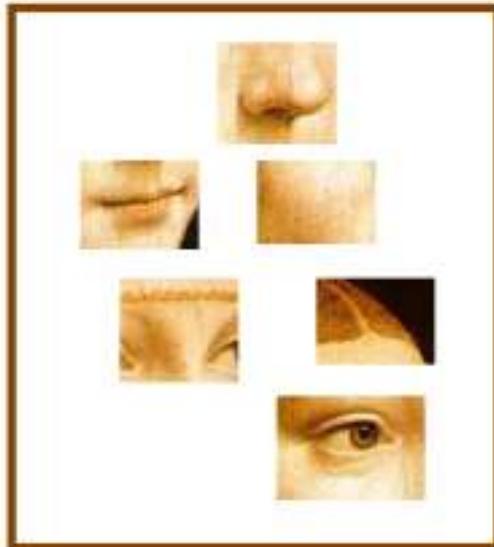
Source: Svetlana Lazebnik

Standard BOVW Pipeline

1. Extract features
2. Cluster features to form a “visual vocabulary” (codebook generation)
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”

Standard BOVW Pipeline

1. Extract features



Standard BOVW Pipeline

1. Extract features
2. Learn “visual vocabulary” (codebook generation)



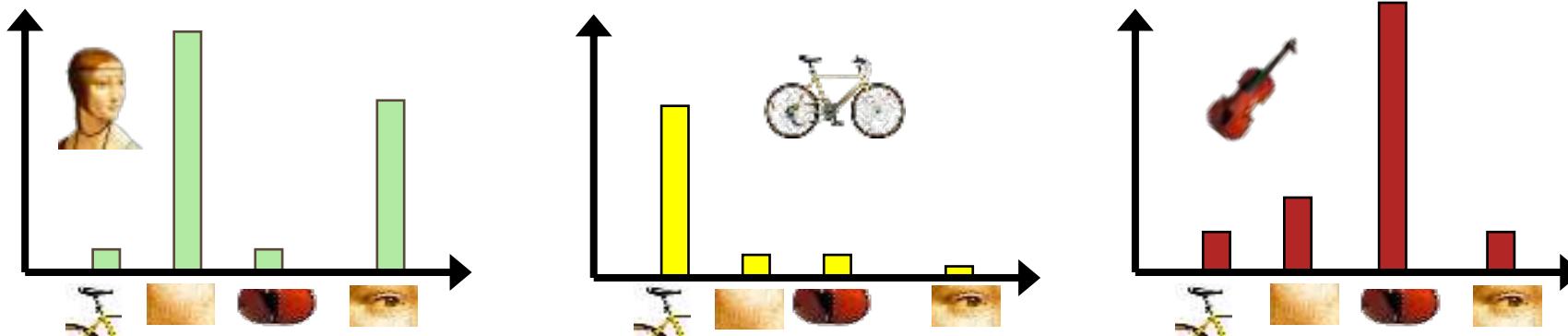
Standard BOVW Pipeline

1. Extract features
2. Learn “visual vocabulary” (codebook generation)
3. Quantize features using visual vocabulary



Standard BOVW Pipeline

1. Extract features
2. Learn “visual vocabulary” (codebook generation)
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”



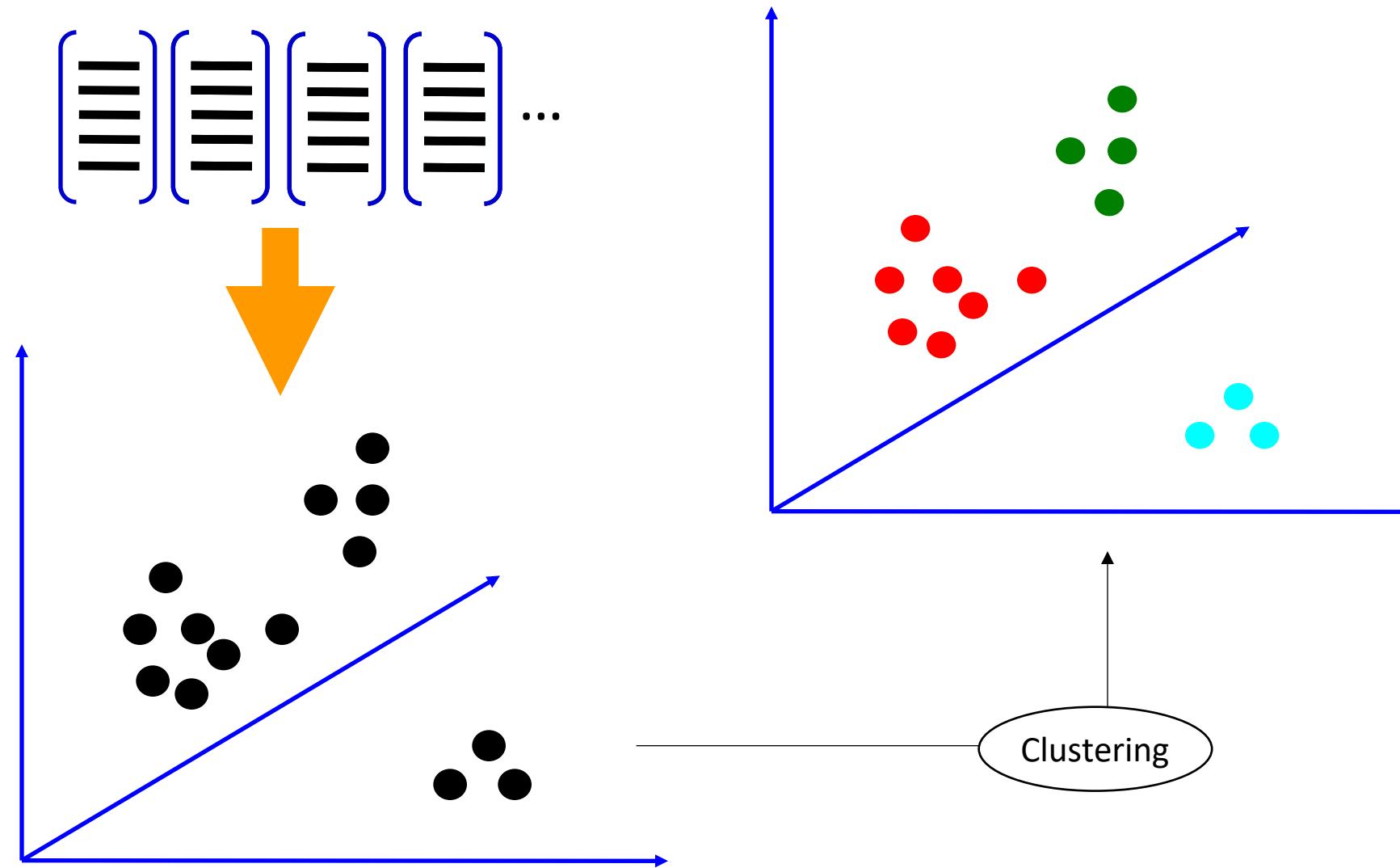
Step 1: Feature Extraction

- **Feature extraction methods**
 - Harris
 - SIFT
 - Dense
 - Any other ...

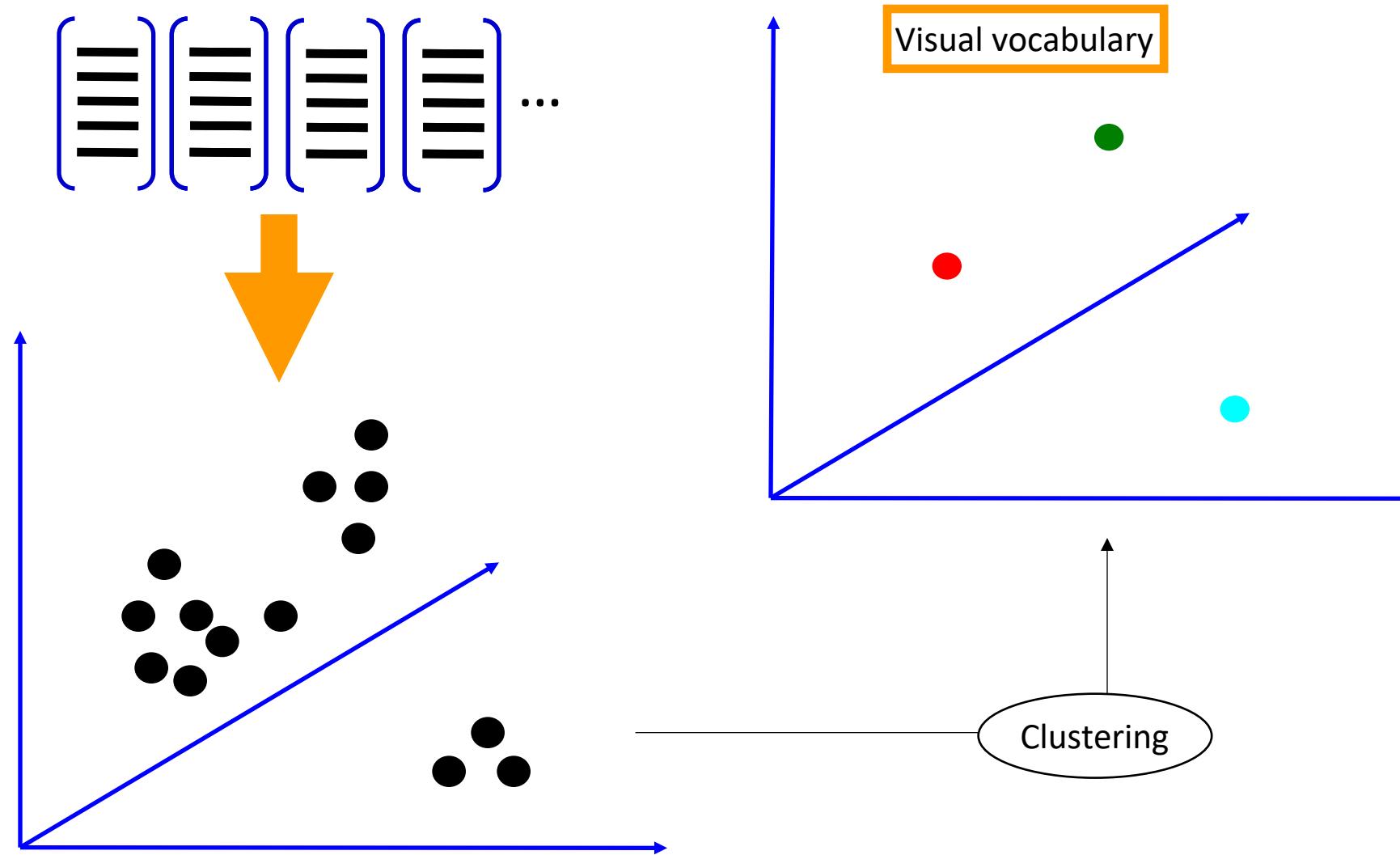
Step 2: Codebook generation

- We extracted feature vectors from each image in our dataset, we need to construct our **vocabulary of possible visual words**.
- Vocabulary construction is normally accomplished via the **k-mean algorithm** where we cluster the feature vectors obtained from **Step #1**.
- The resulting **cluster centers** (i.e., centroids) are treated as our ***dictionary*** of visual words.

Vocabulary construction using k-mean Clustering



Vocabulary construction using k- Mean Clustering



K-means Clustering - Steps

Given k:

1. Select initial centroids at random.
2. Assign each object to the cluster with the nearest centroid.
3. Compute each centroid as the mean of the objects assigned to it.
4. Repeat previous 2 steps until no change.

K-means Clustering

- One of the **most used** clustering algorithm is ***k-means***.
- Let's imagine we have **5 objects** (say 5 people) and for each of them we know two features (height and weight). We want to group them into ***k=2* clusters**.

	Height(H)	Weight(W)
Person 1	167	55
Person 2	120	32
Person 3	113	33
Person 4	175	76
Person 5	108	25

Source: <https://amva4newphysics.wordpress.com/2016/10/26/into-the-world-of-clustering-algorithms-k-means-k-modes-and-k-prototypes/comment-page-1/>

K-means Clustering

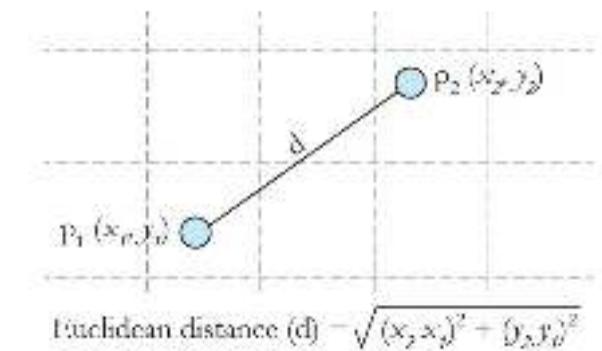
- Initialize the value of the centroids (randomly)

- For instance:

- Person 2 – $c1=(120,32)$
- Person 3 – $c2=(113,33)$

Compute the *Euclidian distance* between each of the two centroids and each point in the data.

	Distance of object from C1	Distance of object from C2
Person 1	52.3	58.3
Person 2	0	7.1
Person 3	7.1	0
Person 4	70.4	75.4
Person 5	13.9	9.4



$$\text{Euclidean distance } (d) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

At this point, we will assign each object to the cluster it is closer to (that is taking the minimum between the two computed distances for each object).

K-means Clustering

- We can then arrange the points as follows:

Person 1 → cluster 1

Person 2 → cluster 1

Person 3 → cluster 2

Person 4 → cluster 1

Person 5 → cluster 2

	Height(H)	Weight(W)
Person 1	167	55
Person 2	120	32
Person 3	113	33
Person 4	175	76
Person 5	108	25

	Distance of object from C1	Distance of object from C2
Person 1	52.3	58.3
Person 2	0	7.1
Person 3	7.1	0
Person 4	70.4	75.4
Person 5	13.9	9.4

Let's iterate, which means to **redefine the centroids** by calculating the mean of the members of each of the two clusters.

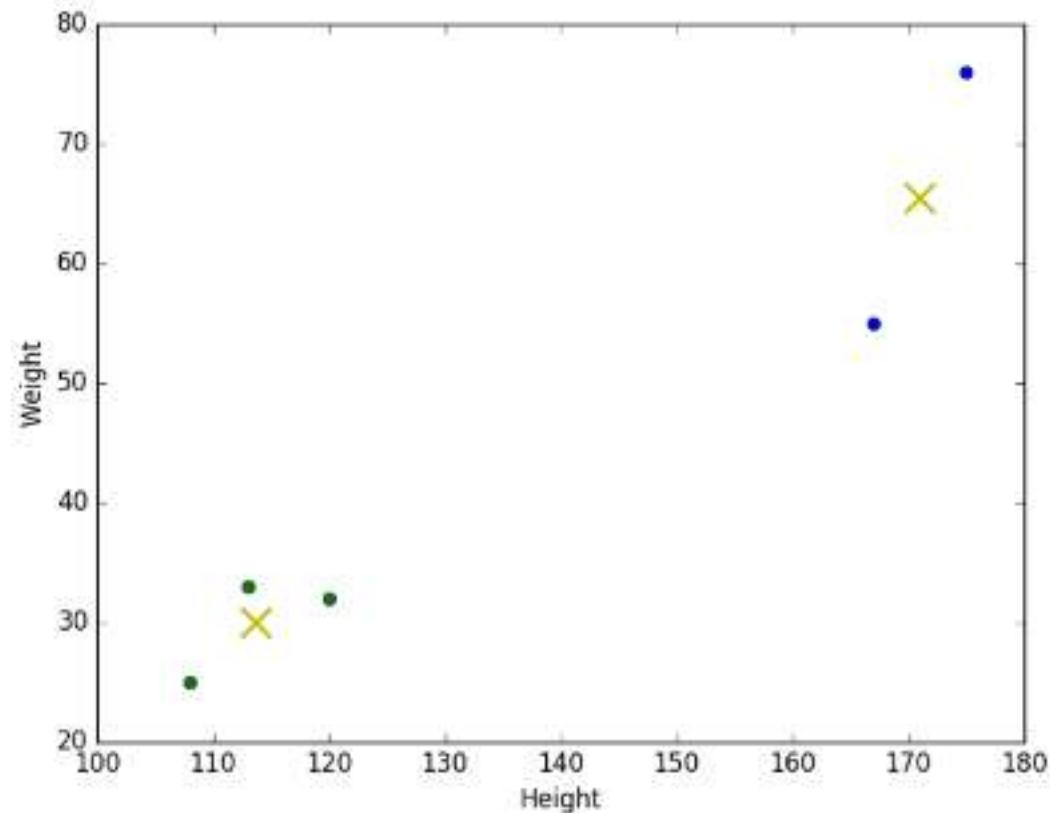
So, $c'1 = ((167+120+175)/3, (55+32+76)/3) = (154, 54.3)$ and $c'2 = ((113+108)/2, (33+25)/2) = (110.5, 29)$

Then, we calculate the **distances again** and **re-assign the points** to the new centroids.

We repeat this process until the centroids don't move anymore

K-means Clustering

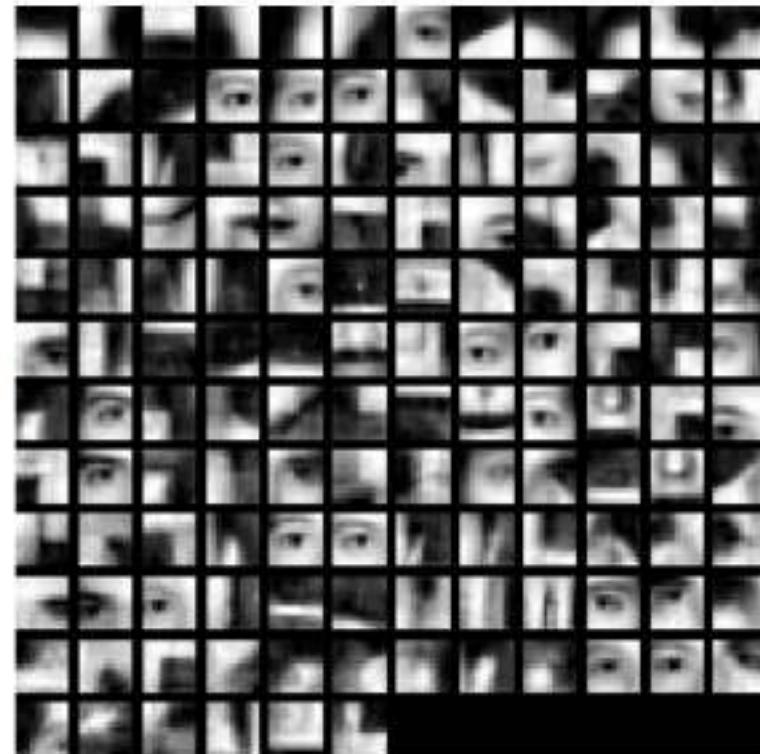
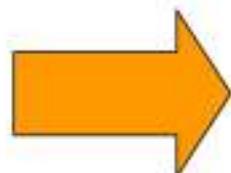
	Height(H)	Weight(W)
Person 1	167	55
Person 2	120	32
Person 3	113	33
Person 4	175	76
Person 5	108	25



Example of visual words learnt by clustering faces

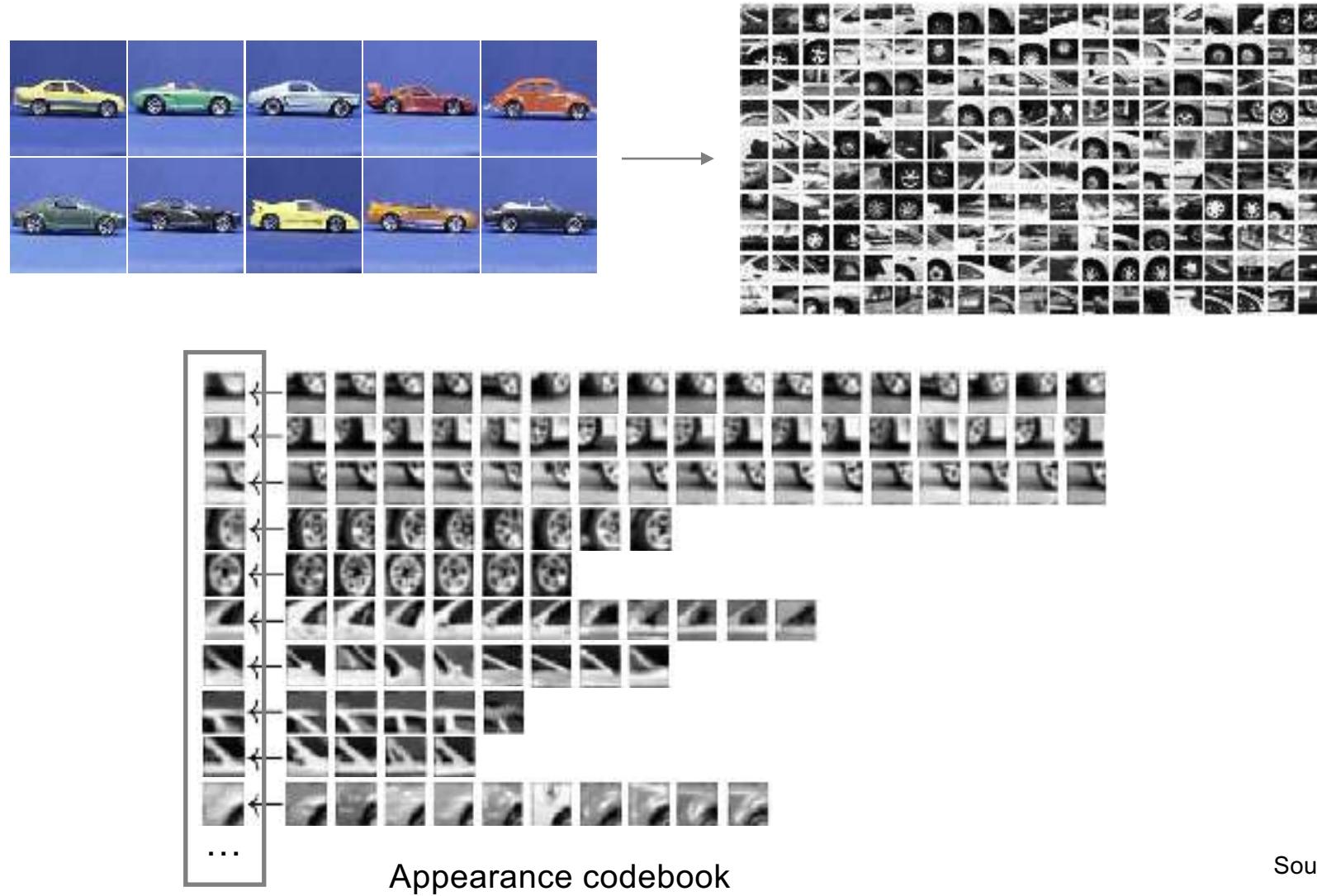


100-1000 images



~100 visual words

Example: Codebook



Source: B. Leibe

Another Codebook



Appearance codebook

Source: B. Leibe

Step 3: Vector Quantization

Given an arbitrary image, we can **quantify and abstractly represent the image** using bag of visual words model

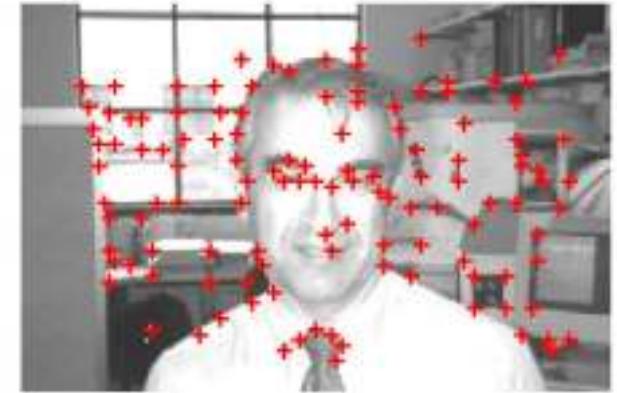
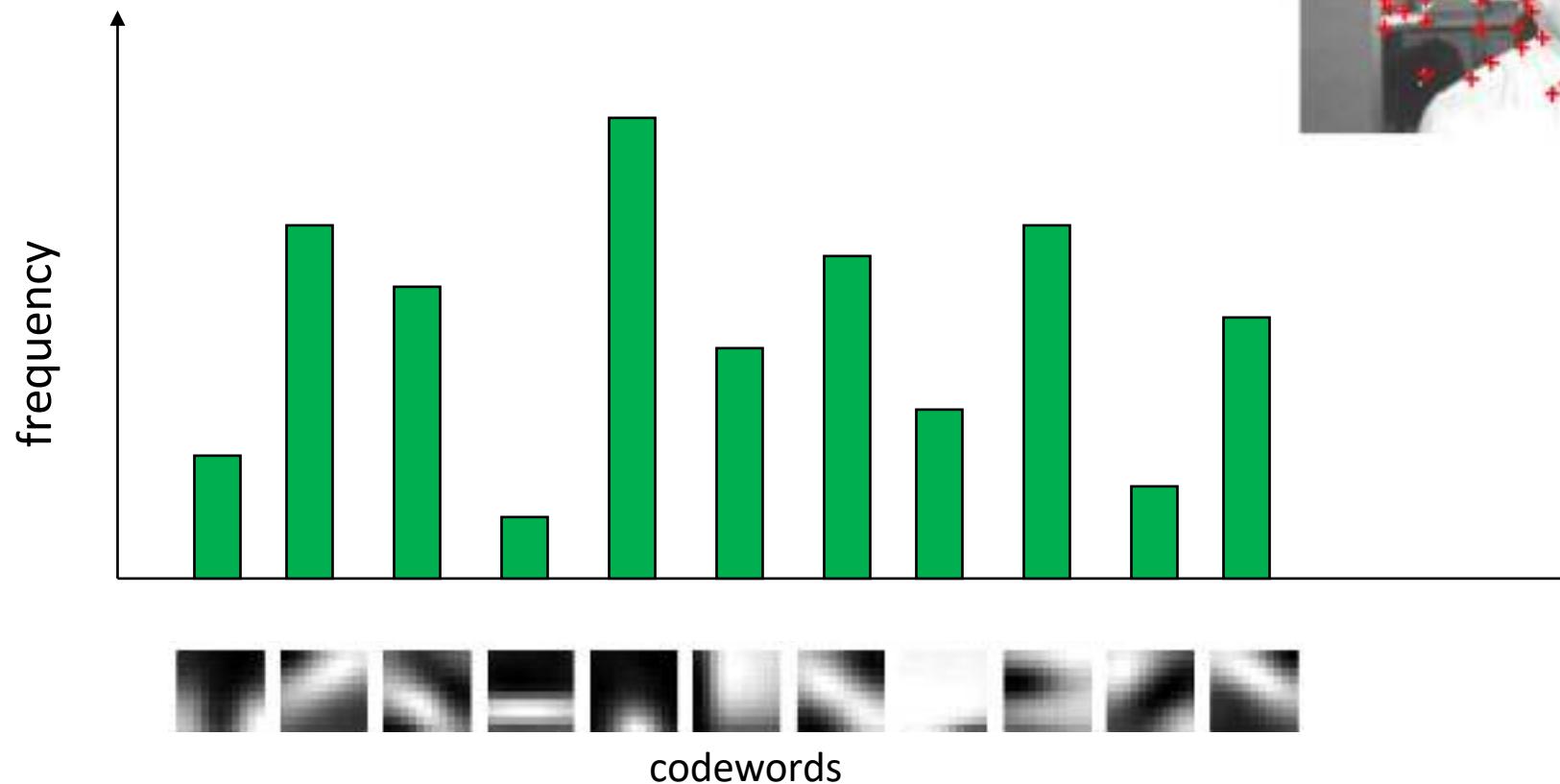
Step 3: Vector Quantization

- Extract feature vectors from the image (**Step #1**)
- For each extracted feature vector, compute its nearest neighbor in the dictionary created in **Step #2** — this is normally accomplished using the Euclidean Distance.
- Take the set of nearest neighbor labels and build **a histogram of length k** (the number of clusters generated from k-means)

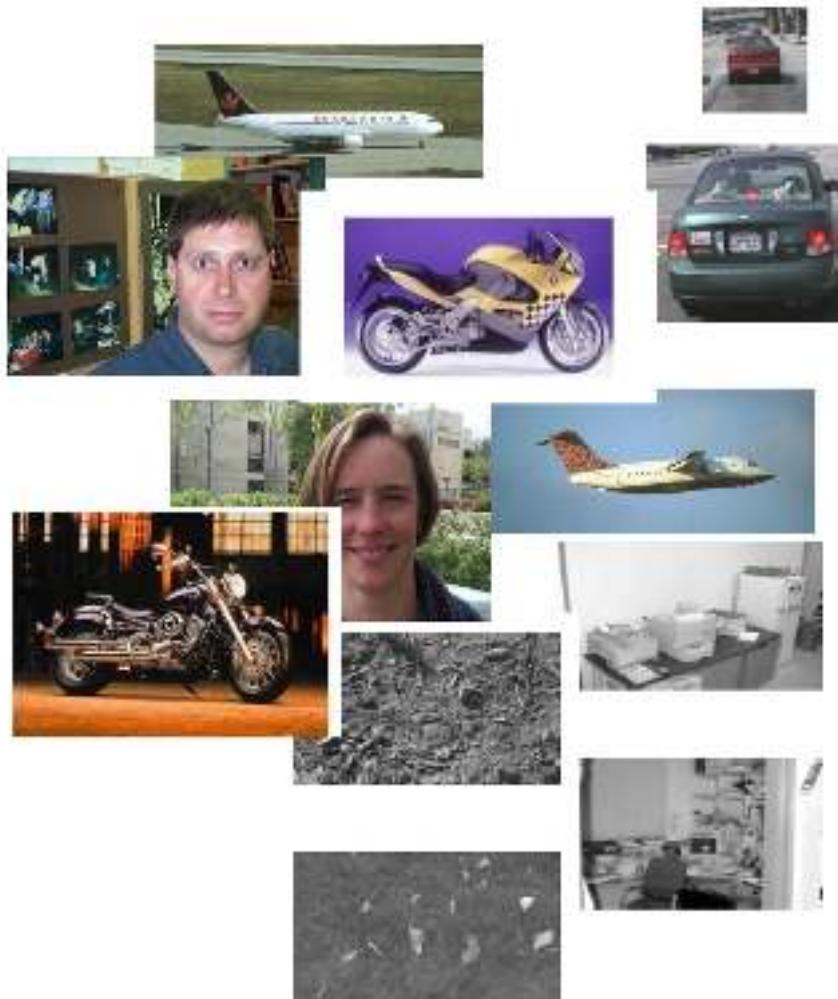
This process commonly called vector quantization.

Image Representation

- Detect interest point features
- Find closest visual word to region around detected points
- Record number of occurrences, but not position



Example: Four object classes + Background

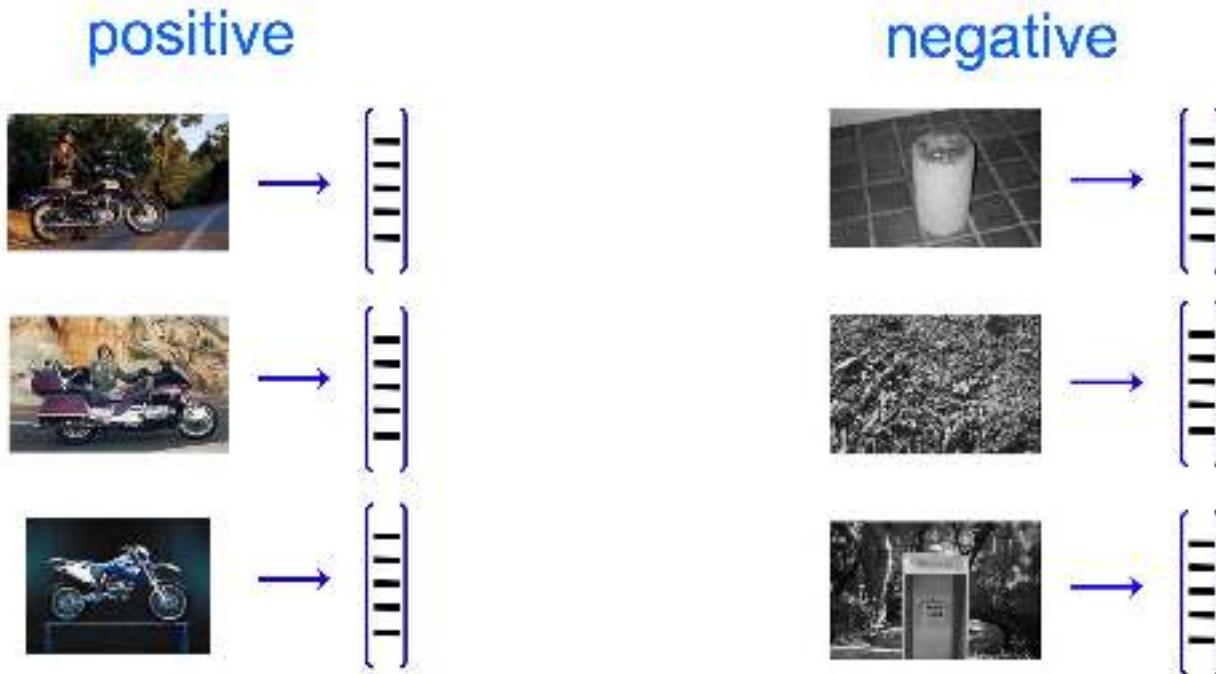


Dataset: Caltech 5

Faces	435
Motorbikes	800
Airplanes	800
Cars (rear)	1155
Background	900
Total:	4090

Training

- Training data: vectors are histograms, one from each training image



Train classifier, e.g. SVM

Results: Weak Supervision

Training

- 50% images
- No identification of object within image

Motorbikes



Airplanes



Frontal Faces



Testing

- 50% images
- Simple object present/absent test

Cars (Rear)



Background



Learning

- SVM classifier
- Gaussian kernel

Result

- Between 98.3 – 100% correct, depending on class

Csurka et al 2004
Zhang et al 2005

Bag of visual words: Summary

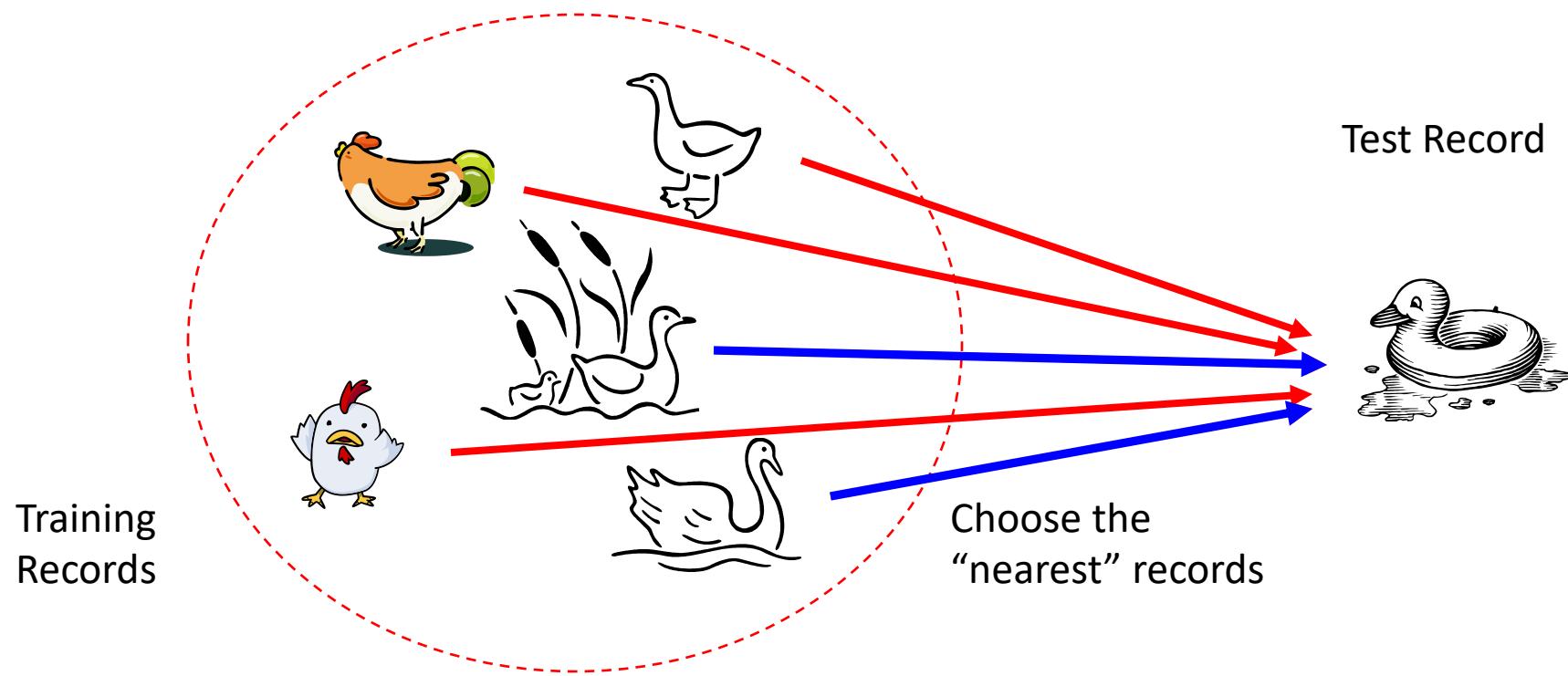
- **Advantages:**
 - Largely unaffected by position and orientation of object in image
 - Fixed length vector irrespective of number of detections
 - Very successful in classifying images according to the objects they contain
 - Still requires further testing for large changes in scale and viewpoint
- **Disadvantages:**
 - No explicit use of configuration of visual word positions
 - Poor at localizing objects within an image

Thank You 😊

Appendix

k-Nearest Neighbor Classifiers – Intuition

- Basic idea:
 - If it walks like a duck, quacks like a duck, then it's probably a duck



Nearest-neighbor classification

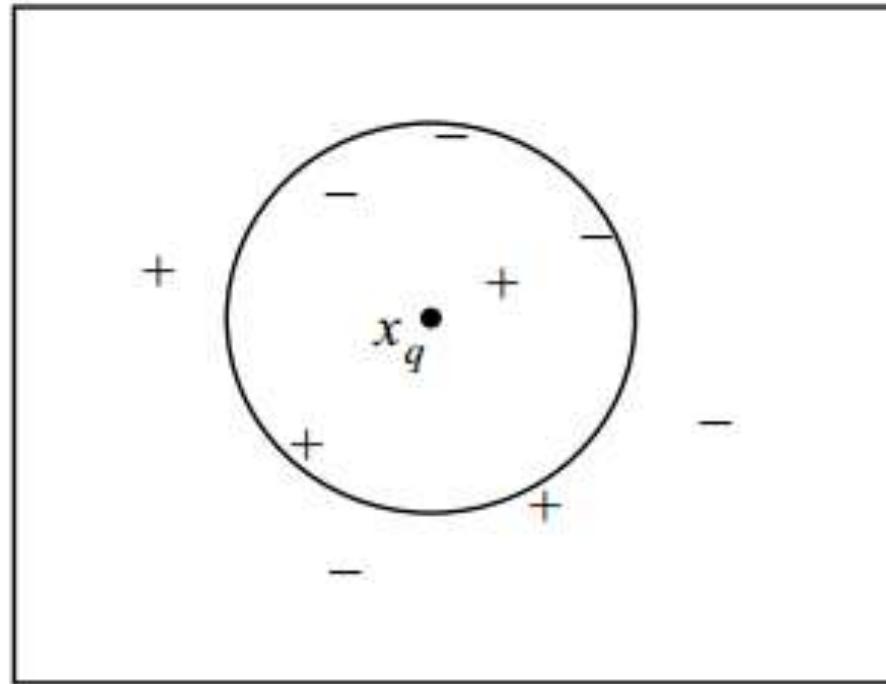
- **Learning task**

- Given a training set $(x_1, y_1) \dots (x_n, y_n)$, do nothing (That's why sometime called a **lazy learner**)

- **Classification task**

- **Given:** an instance x_q to classify
- Find the training-set instance x_i that is most similar to x_q
- Return the class value y_i

k-Nearest Neighbor Learning



Note: that 1-nearest neighbor classifies x_q as positive,
whereas 5-nearest neighbor classifies it as negative

Distance Metrics

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_N - y_N)^2} \quad \text{Euclidean}$$

$$D(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{x_1 y_1 + \cdots + x_N y_N}{\sqrt{\sum_n x_n^2} \sqrt{\sum_n y_n^2}} \quad \text{Cosine}$$

$$D(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \sum_n \frac{(x_n - y_n)^2}{(x_n + y_n)} \quad \text{Chi-squared}$$

Nearest Neighbor is Competitive

40281508803277064455529284686500876/71127400776386420140578274711366
507111676796641431122410876340063017113109975414895351982339901029
8468482467933943144705960444612336459685656641865284554770782237018
76953465013828357808571101378507110114527623028596972136418240510226
9377714906484272810078333137613160547598249916501320348220251514889
82049962335648092836757294912860709116759919592504108908989425798980
3551721691995516228671460403322368985385452056328399579467131366090/
94568160413174951001162198403649071654525185470670258104571851900607
88573898868239756292881688791801720751902098623938021111429725112199
1485343477507488153959969036398212868553949251514414435912233029009
931909754920105149336152520266012030255795508950326908845884548549
69285457999216340783939456239260061287982047750564674307507420819404
12845278113035703193631773084826529739099642972116747598821445161325
90666367728608302983253880019513960141712379749939284718091017796499
21010452828351781129784050768477858498138031745516574935471208160734
28308784084458566309376893495891288681379011970817457121130621280766
41992780136134111560707232522949918161274000822922799275134941856283

MNIST Digit Recognition

- Handwritten digits
- 28x28 pixel images: $d = 784$
- 60,000 training samples
- 10,000 test samples

Test Error Rate (%)	
K-nearest-neighbors (Euclidean)	5.0

You Only Look Once (YOLO)

Real-Time Object Detection

Instructor: Dr. Muhammad Fahim

Contents

- Motivation for Neural Learning Models
- Deep Learning models for Object Detection
- Transfer Learning Mechanism
- You Only Look Once: Unified, Real-Time Object Detection
- Summary

Source of the material

- This lecture is based on the following resources
 - Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.
 - Taegyun Jeon slides
 - Other material found over the internet.

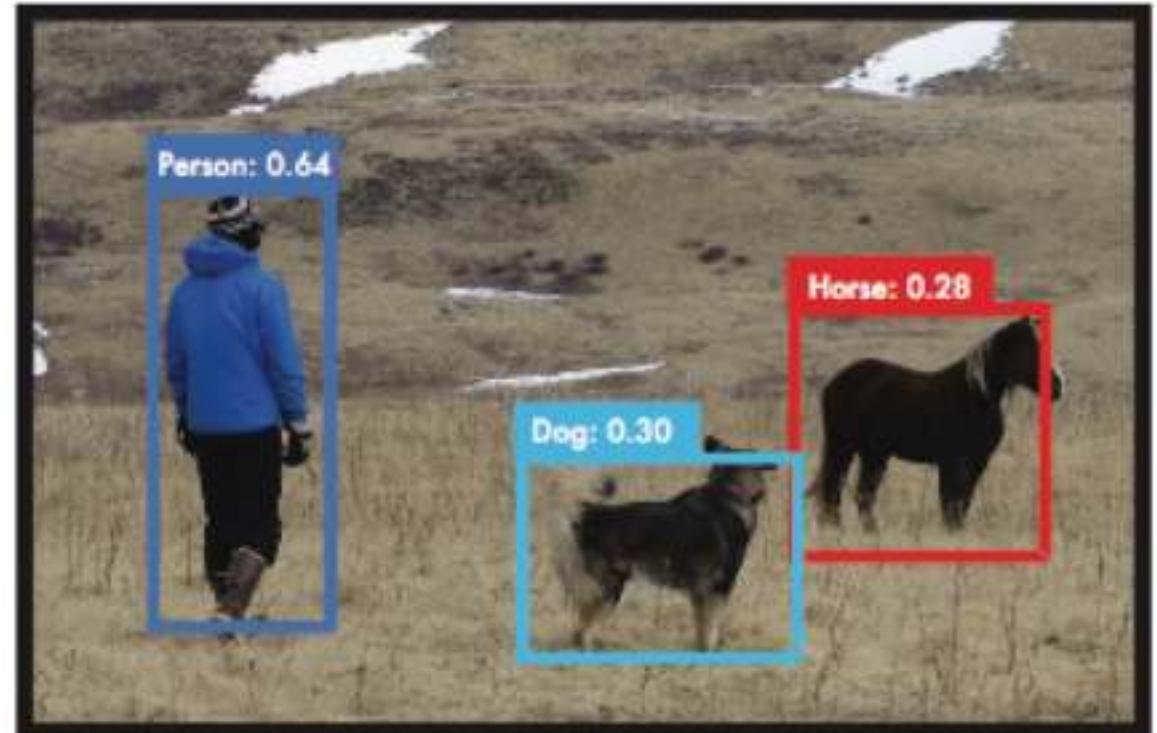
Motivation for Neural Learning Models

- We can apply any **existing machine learning/deep learning techniques** for classification problem.
- Some of which are:
 - SVM on image features
 - kNN on image features
 - and so on...

Convolutional Neural Networks on raw images

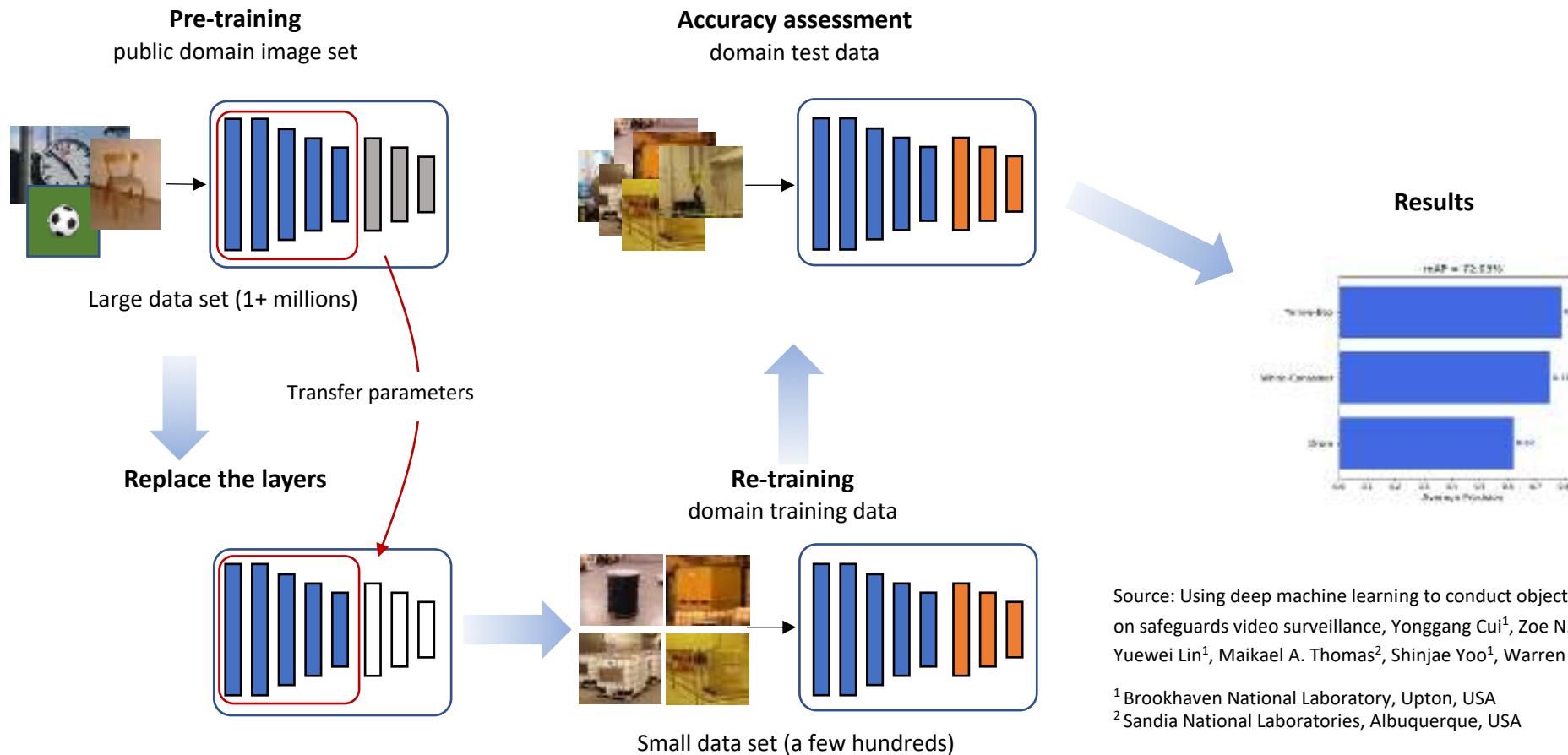
Object Detection Task

- The object detection task consists
 1. Location of the objects
 2. Classification of those objects



Transfer Learning Mechanism

- Solves the issue of insufficient domain training data
- Re-training requires much less computational resource



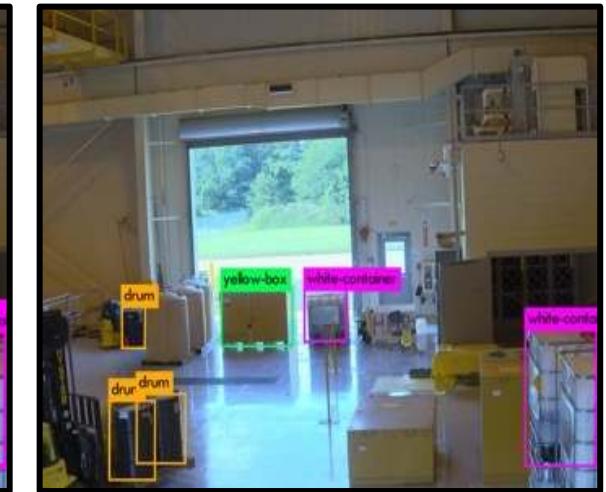
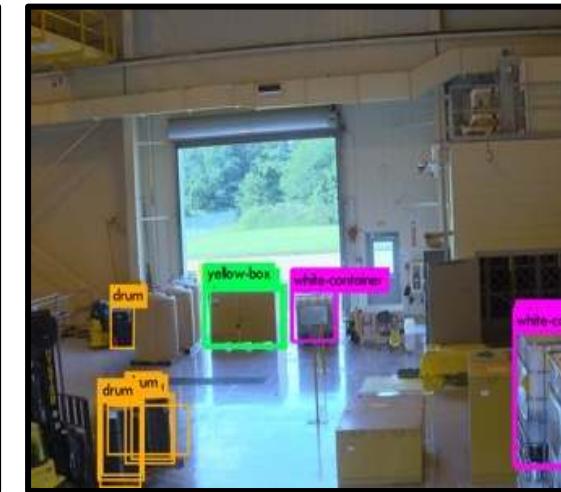
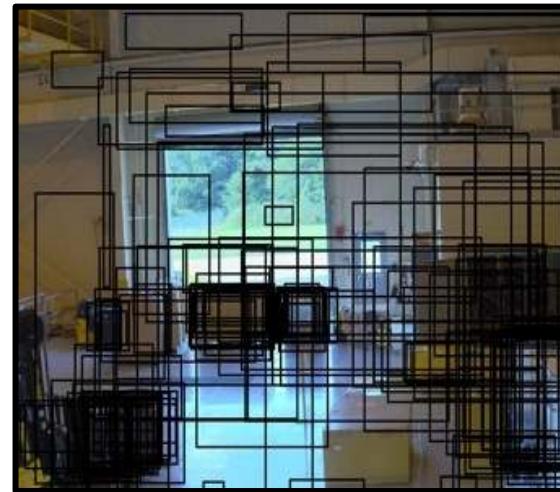
Deep Learning Models for Object Detection

R-CNN OverFeat DetectorNet
DeepMultibox SPP-net Fast R-
CNN MR-CNN SSD YOLO YOLOv2
G-CNN AttractioNet Mask R-CNN
R-FCN RPN FPN Faster R-CNN ...

and many more words

YOLO: Object Detection

- YOLO (You Only Look Once) is a deep neural network
- Image-based algorithm **suitable** for safeguards **surveillance camera data**



Basic Concept

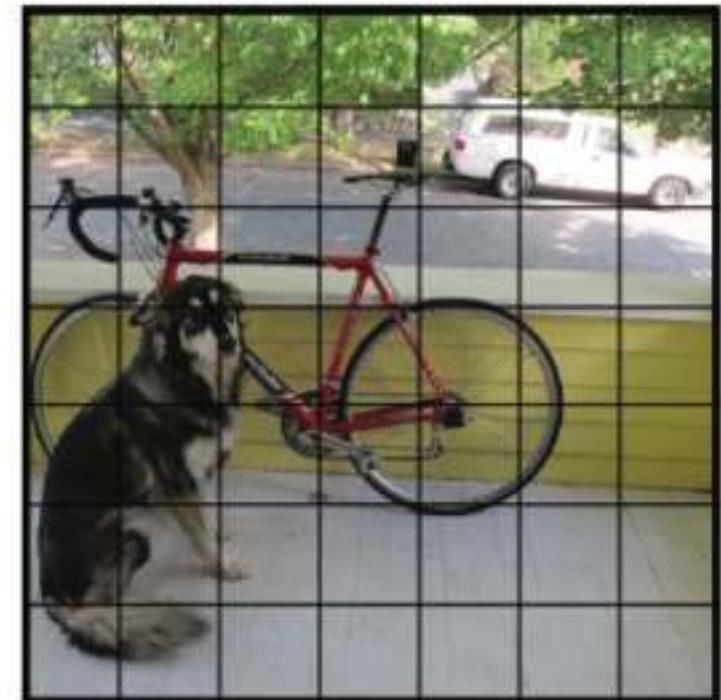
- **Object Detection**
 - Regression problem
- **YOLO**
 - Only one feedforward NN
 - Global context
- **Unified (Real-time detection)**
 - YOLO: 45 FPS
 - Fast YOLO: 155 FPS
- **General Representation**
 - Robust on various background
 - Other domain

Basic idea

Take an image as input, pass it through a neural network that looks similar to a normal CNN, and you get a vector of bounding boxes and class predictions in the output

The Predictions Vector

- The first step to understanding YOLO is how it encodes its output.
- The input image is divided into a $S \times S$ grid of cells.
- For each object that is present on the image, one grid cell is said to be “responsible” for predicting it. (i.e., the cell where the center of the object falls into).



$S \times S$ grid on input

The Predictions Vector

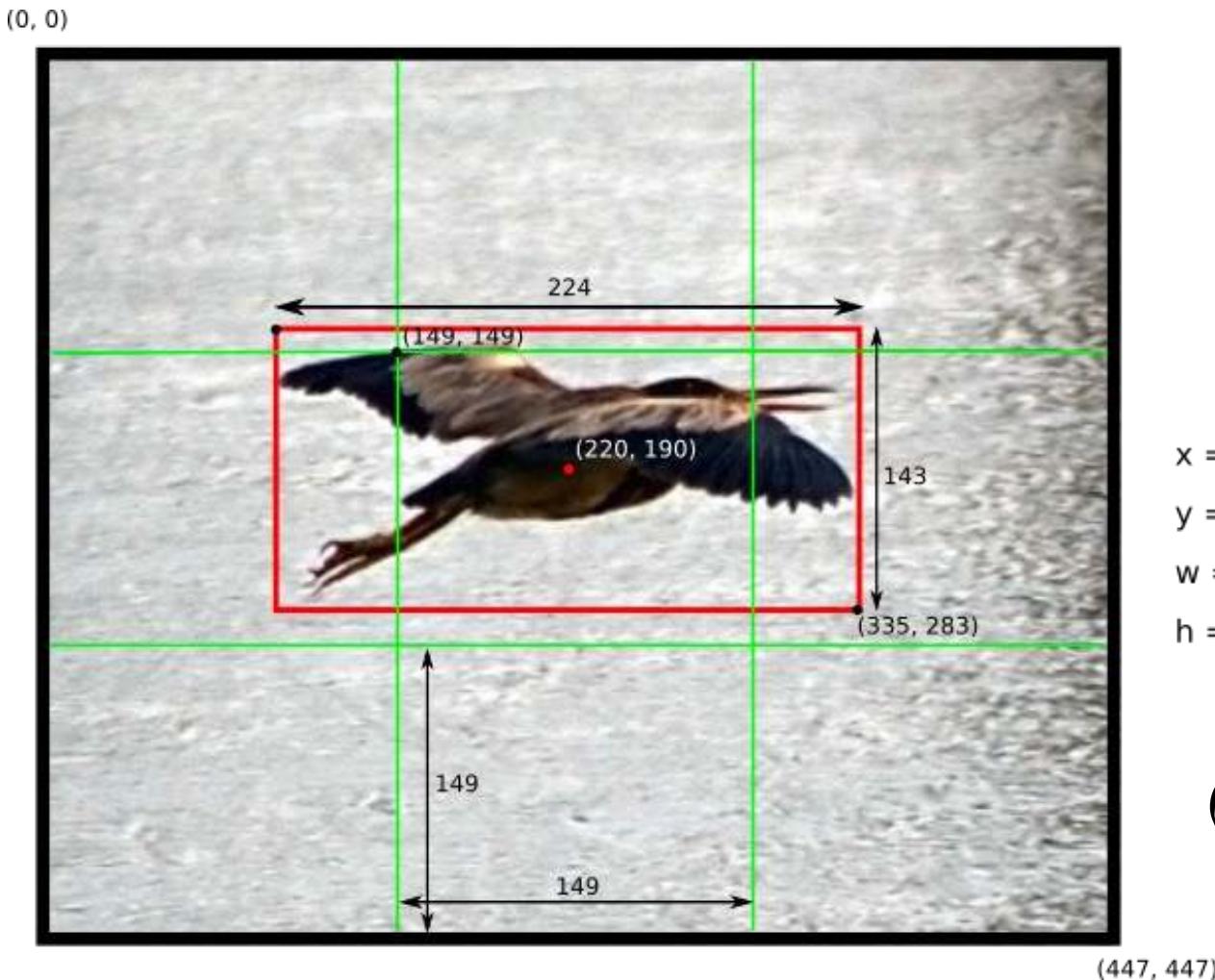
- Each grid cell predicts B bounding boxes as well as C class probabilities.
- The bounding box prediction has 5 components:

$$(x, y, w, h, \text{confidence})$$


- **Normalization**
 - The (x, y) coordinates are **normalized** to fall between 0 and 1.
 - The (w, h) box dimensions are also **normalized** to $[0, 1]$, relative to the image size.

The Predictions Vector

Example:



$$x = (220 - 149) / 149 = 0.48$$

$$y = (190 - 149) / 149 = 0.28$$

$$w = 224 / 448 = 0.50$$

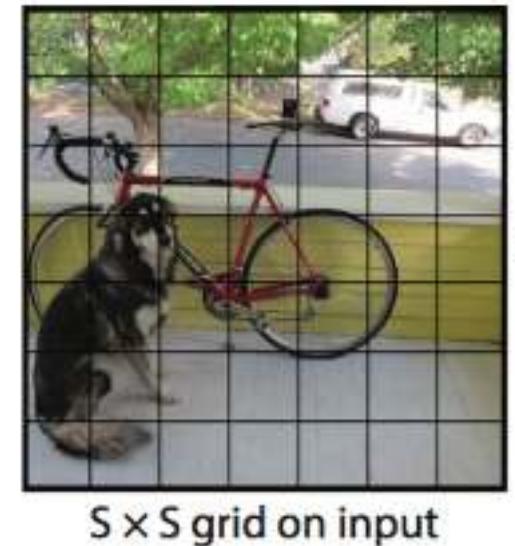
$$h = 143 / 448 = 0.32$$

$(x, y, w, h, \text{confidence})$

The Predictions Vector: Confidence Score

- Confidence score reflects the **presence or absence** of an object of *any class*.
- Confidence is defined as:

$$Pr(\text{Object}) * IOU(\text{pred}, \text{truth})$$

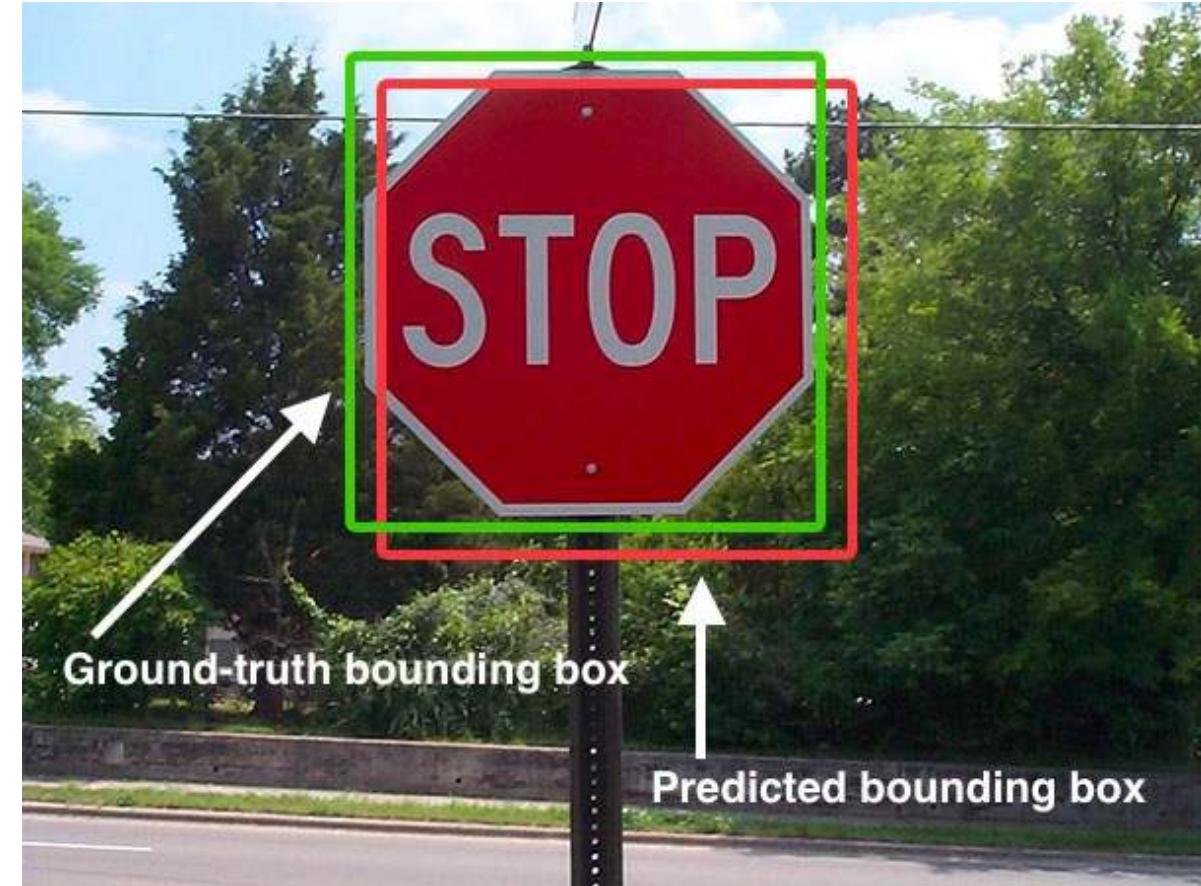


- If no object exists in that cell, the confidence score should be zero.

Intersection over Union (IoU)

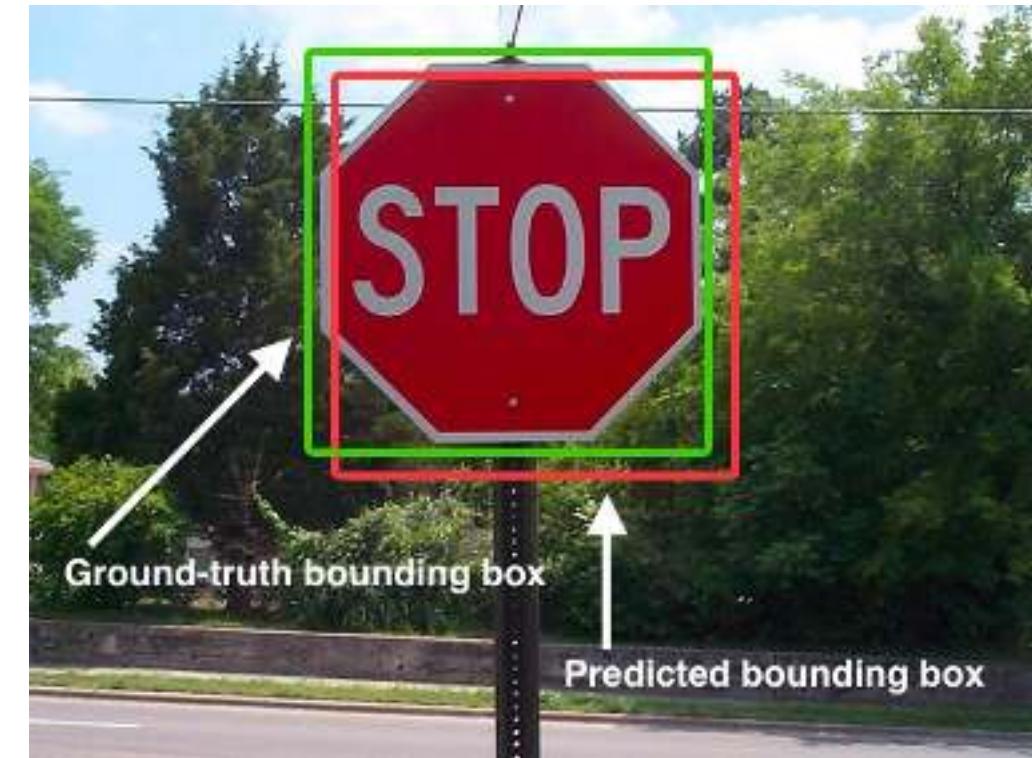
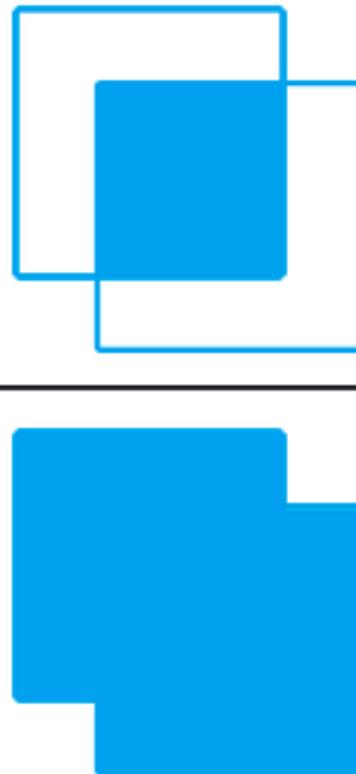
Our goal is to compute the **Intersection of Union** between these bounding box.

- An example of detecting a stop sign in an image.
 - The predicted bounding box is drawn in red.
 - The ground-truth bounding box is drawn in green.



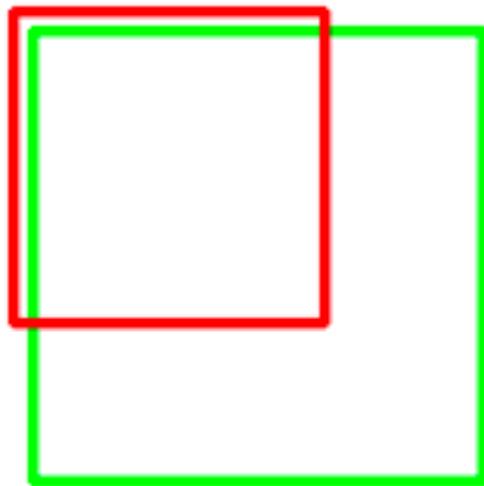
Computing the Intersection over Union

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



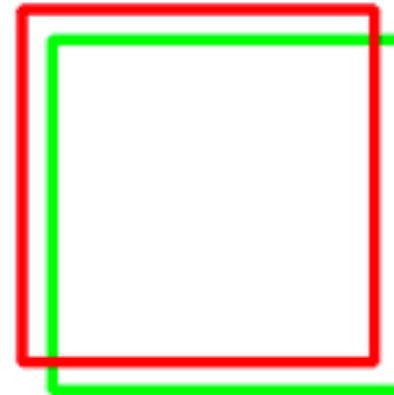
Computing the Intersection over Union

IoU: 0.4034



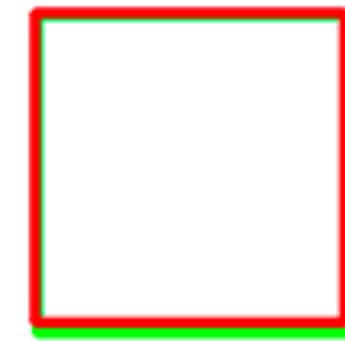
Poor

IoU: 0.7330



Good

IoU: 0.9264



Excellent

So far...

- We understand the 5 components of the box prediction
 $(x, y, w, h, \text{confidence})$

- Each grid cell makes **B** of those predictions
- So, there are total **$S \times S \times B * 5$** outputs

- It is also necessary to predict the class probabilities

$$Pr(\text{Class}(i) \mid \text{Object})$$



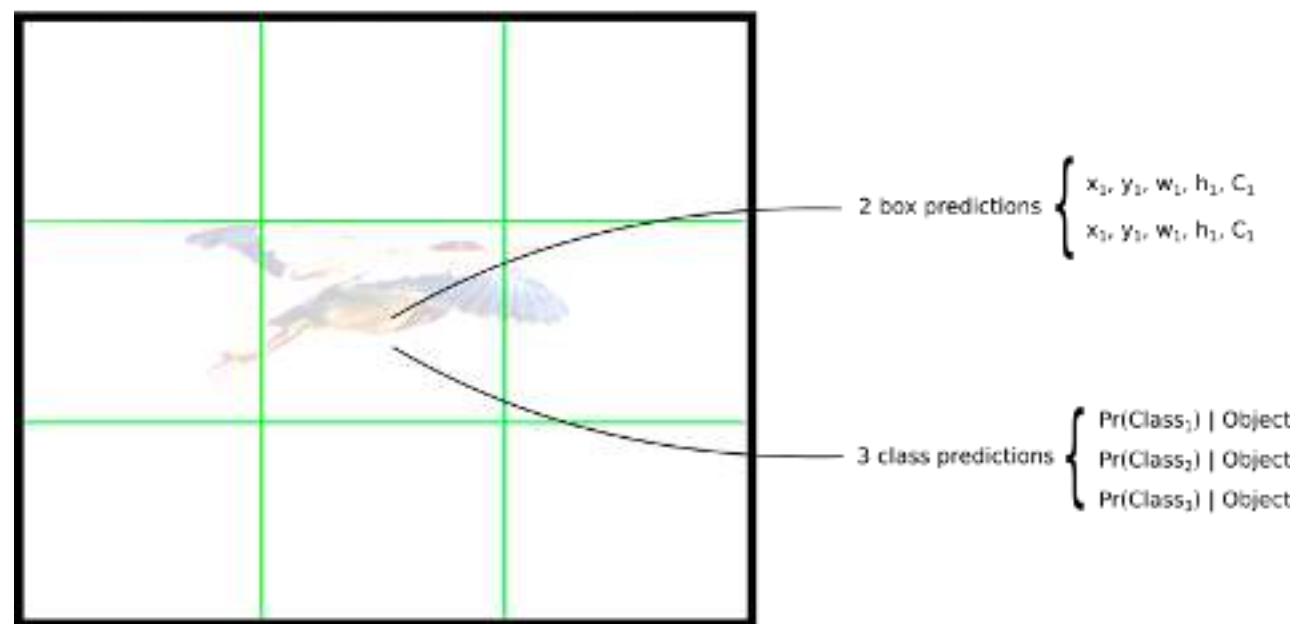
5×5 grid on input

- This probability is conditioned on the grid cell containing one object

Adding class predictions

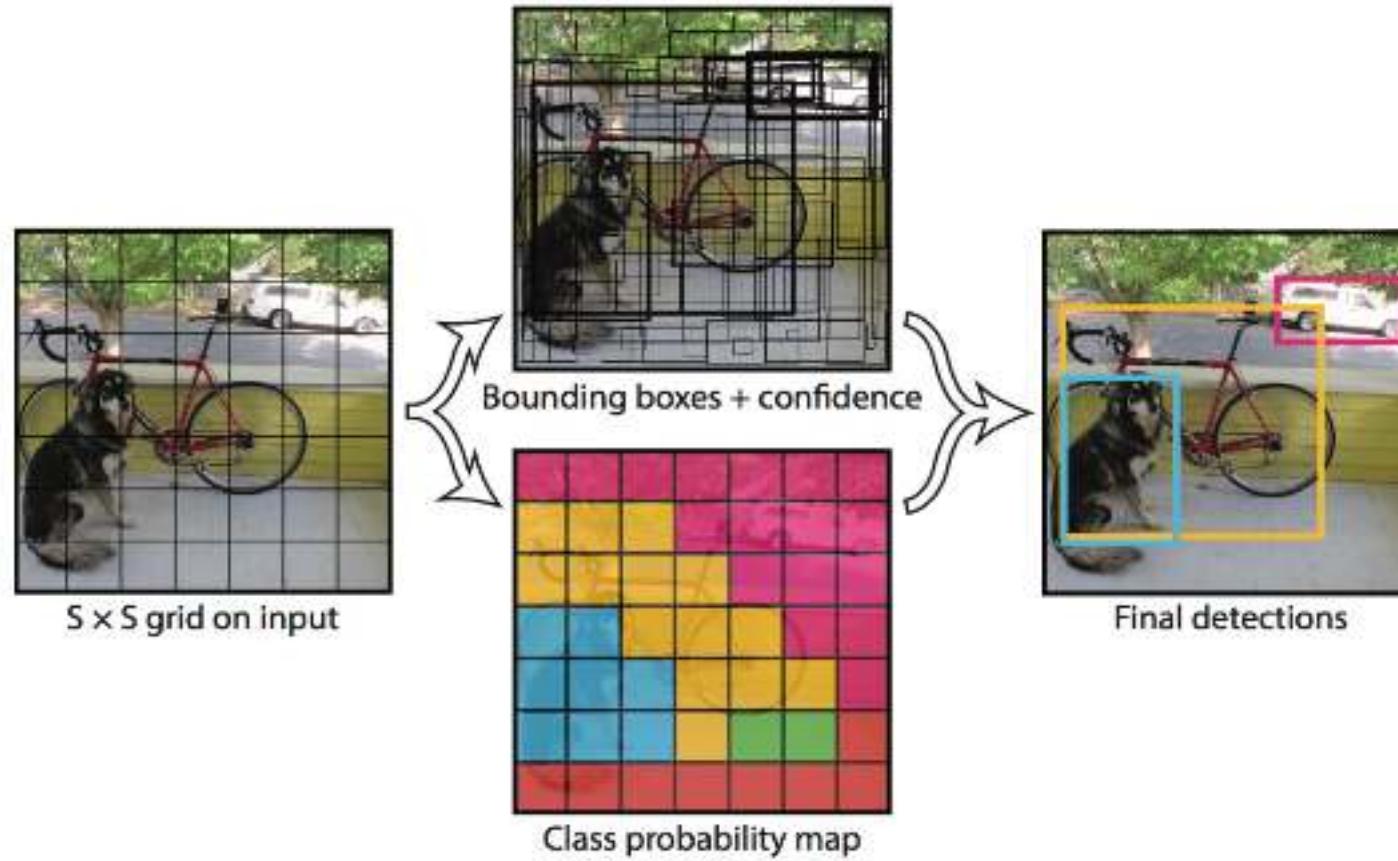
- Adding the class predictions to the output vector

$$S \times S \times (B * 5 + C)$$



Each grid cell makes B bounding box predictions and C class predictions
($S=3$, $B=2$ and $C=3$ in this example)

Final Model looks like!!



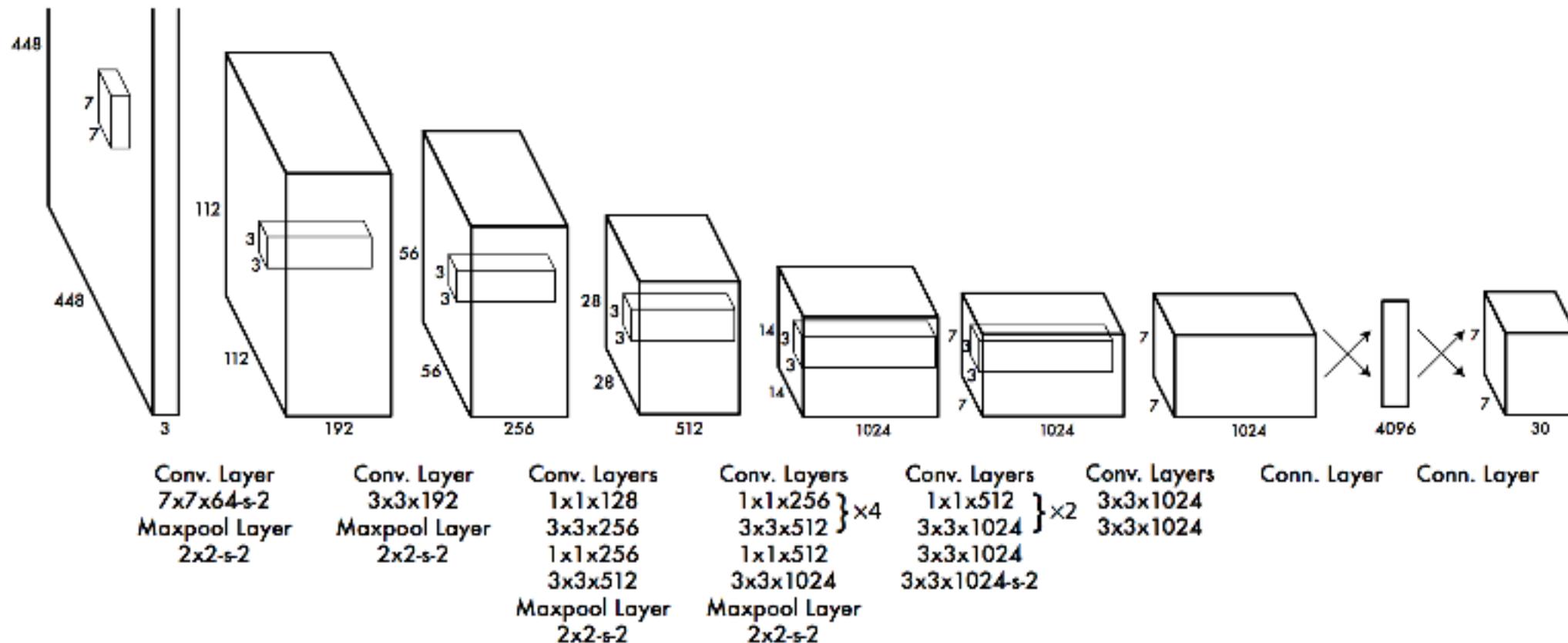
Network Design

Network Design

- Modified GoogLeNet

- 1x1 reduction layer (“Network in Network”)

S=7, B=2 and C=20



Network Design Summary (Another view)

Name	Filters	Output Dimension
Conv 1	7 x 7 x 64, stride=2	224 x 224 x 64
Max Pool 1	2 x 2, stride=2	112 x 112 x 64
Conv 2	3 x 3 x 192	112 x 112 x 192
Max Pool 2	2 x 2, stride=2	56 x 56 x 192
Conv 3	1 x 1 x 128	56 x 56 x 128
Conv 4	3 x 3 x 256	56 x 56 x 256
Conv 5	1 x 1 x 256	56 x 56 x 256
Conv 6	1 x 1 x 512	56 x 56 x 512
Max Pool 3	2 x 2, stride=2	28 x 28 x 512
Conv 7	1 x 1 x 256	28 x 28 x 256
Conv 8	3 x 3 x 512	28 x 28 x 512
Conv 9	1 x 1 x 256	28 x 28 x 256
Conv 10	3 x 3 x 512	28 x 28 x 512
Conv 11	1 x 1 x 256	28 x 28 x 256
Conv 12	3 x 3 x 512	28 x 28 x 512
Conv 13	1 x 1 x 256	28 x 28 x 256
Conv 14	3 x 3 x 512	28 x 28 x 512
Conv 15	1 x 1 x 512	28 x 28 x 512
Conv 16	3 x 3 x 1024	28 x 28 x 1024
Max Pool 4	2 x 2, stride=2	14 x 14 x 1024
Conv 17	1 x 1 x 512	14 x 14 x 512
Conv 18	3 x 3 x 1024	14 x 14 x 1024
Conv 19	1 x 1 x 512	14 x 14 x 512
Conv 20	3 x 3 x 1024	14 x 14 x 1024
Conv 21	3 x 3 x 1024	14 x 14 x 1024
Conv 22	3 x 3 x 1024, stride=2	7 x 7 x 1024
Conv 23	3 x 3 x 1024	7 x 7 x 1024
Conv 24	3 x 3 x 1024	7 x 7 x 1024
FC 1	-	4096
FC 2	-	7 x 7 x 30 (1470)

Activation Function

- **Final layer**
 - A linear activation function
- **All other layers**
 - leaky rectified linear activation (Leaky Relu)

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

Loss Function (sum-squared error)

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Loss Function (sum-squared error)

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

- This part computes the loss related to the predicted bounding box position (x, y) .
- λ_{coord} : just consider it as a parameter and its value is given as $\lambda_{coord} = 5$
- $\mathbb{1}^{obj}$ is defined as:
 - 1, If an object is present in grid cell i and the j th bounding box predictor is “responsible” for that prediction
 - 0, otherwise

Loss Function (sum-squared error)

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

- This is the loss related to the predicted box width/height

Loss Function (sum-squared error)

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

- Compute the loss associated with the confidence score for each bounding box predictor.
 - $\mathbb{1}^{obj}$ is equal to one when there is an object in the cell, and 0 otherwise.
 - $\mathbb{1}^{noobj}$ is the opposite.
 - λ_{noobj} is a constant and its value is 0.5

Loss Function (sum-squared error)

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

- It looks similar to a normal sum-squared error for classification, except for the $\mathbb{1} obj$ term.

Loss Function (sum-squared error)

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \boxed{\mathbb{1}_{ij}^{\text{obj}}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \boxed{\mathbb{1}_{ij}^{\text{obj}}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$\boxed{\mathbb{1}_{ij}^{\text{obj}}}$

The **jth bbox predictor** in **cell i** is “responsible” for that prediction

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \boxed{\mathbb{1}_{ij}^{\text{obj}}} (C_i - \hat{C}_i)^2$$

$\boxed{\mathbb{1}_{ij}^{\text{noobj}}}$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \boxed{\mathbb{1}_{ij}^{\text{noobj}}} (C_i - \hat{C}_i)^2$$

$\boxed{\mathbb{1}_i^{\text{obj}}}$

If object appears in **cell i**

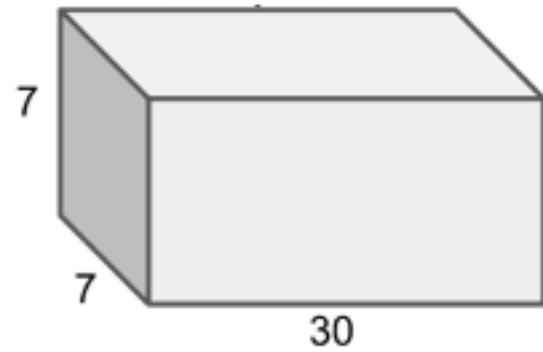
$$+ \sum_{i=0}^{S^2} \boxed{\mathbb{1}_i^{\text{obj}}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Network Training

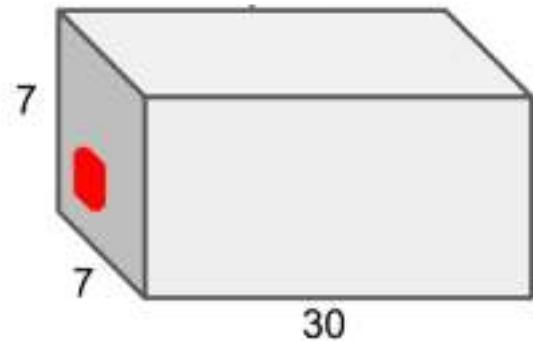
- Pretrain the first 20 convolutional layers using the ImageNet 1000-class competition dataset, using a input size of 224x224
- Double the input resolution (i.e., 448x448)
- **epochs**=135
- **batch_size**=64
- **momentum_a** = 0.9
- **decay**=0.0005
- **lr**=[10^{-3} , 10^{-2} , 10^{-3} , 10^{-4}]
- **dropout_rate**=0.5
- **augmentation**
=[scaling, translation, exposure, saturation]

Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from 10^{-3} to 10^{-2} . If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with 10^{-2} for 75 epochs, then 10^{-3} for 30 epochs, and finally 10^{-4} for 30 epochs.

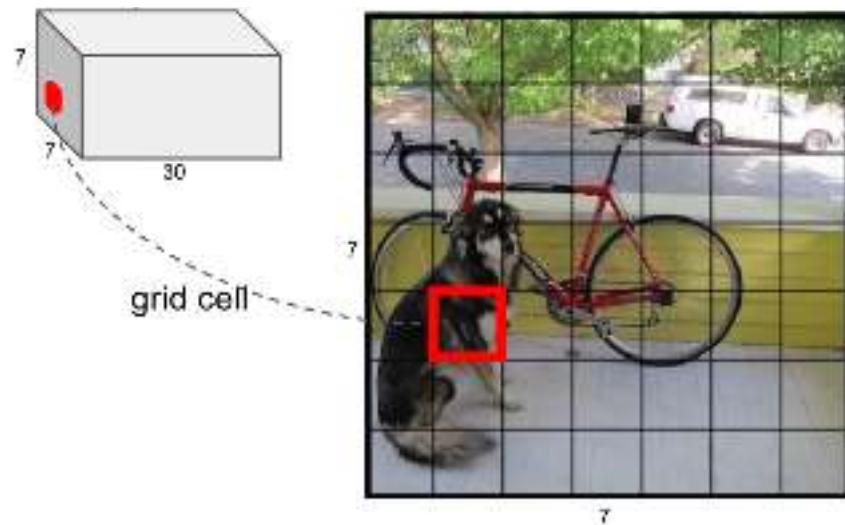
Inference



Inference

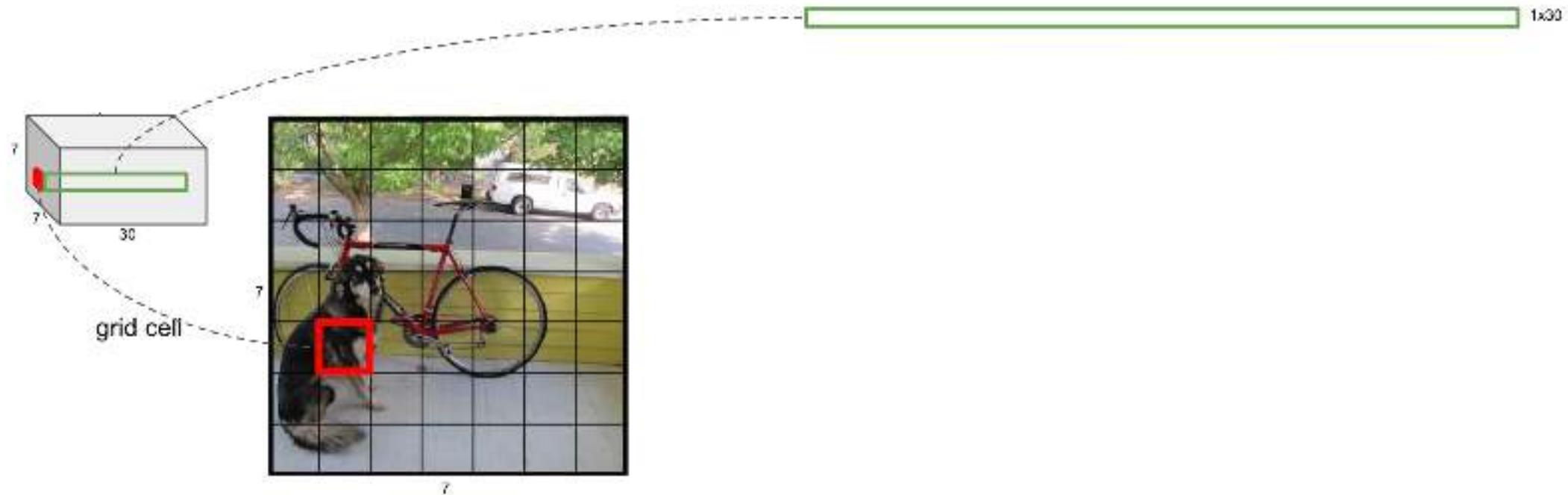


Inference



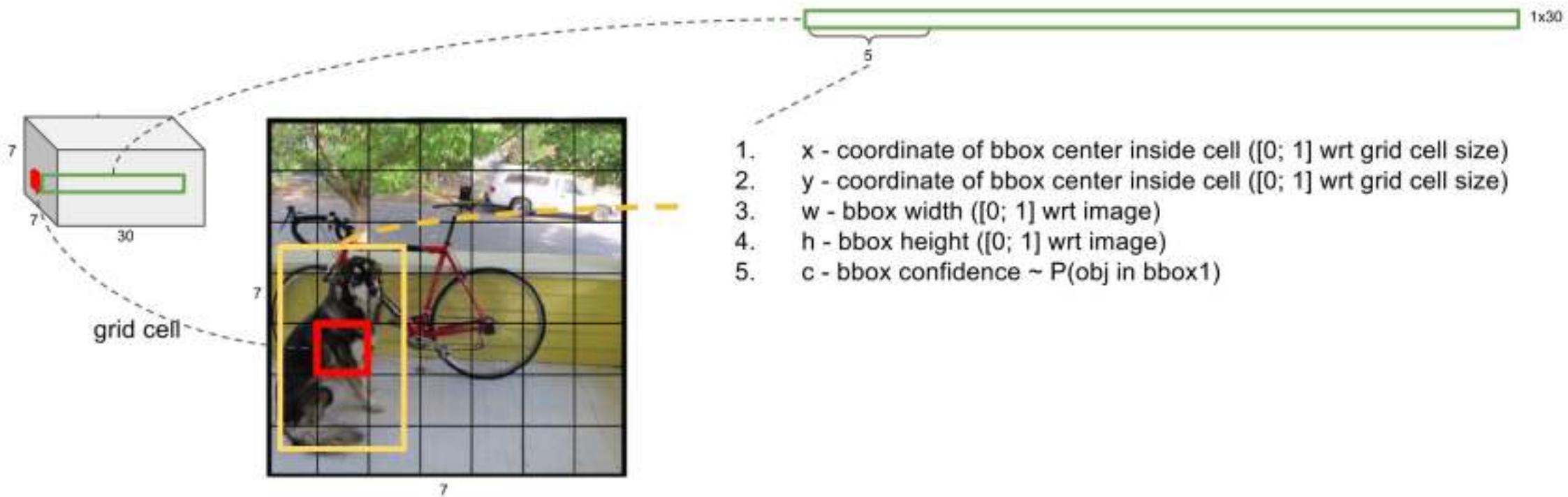
Source: deepsystem.ai

Inference



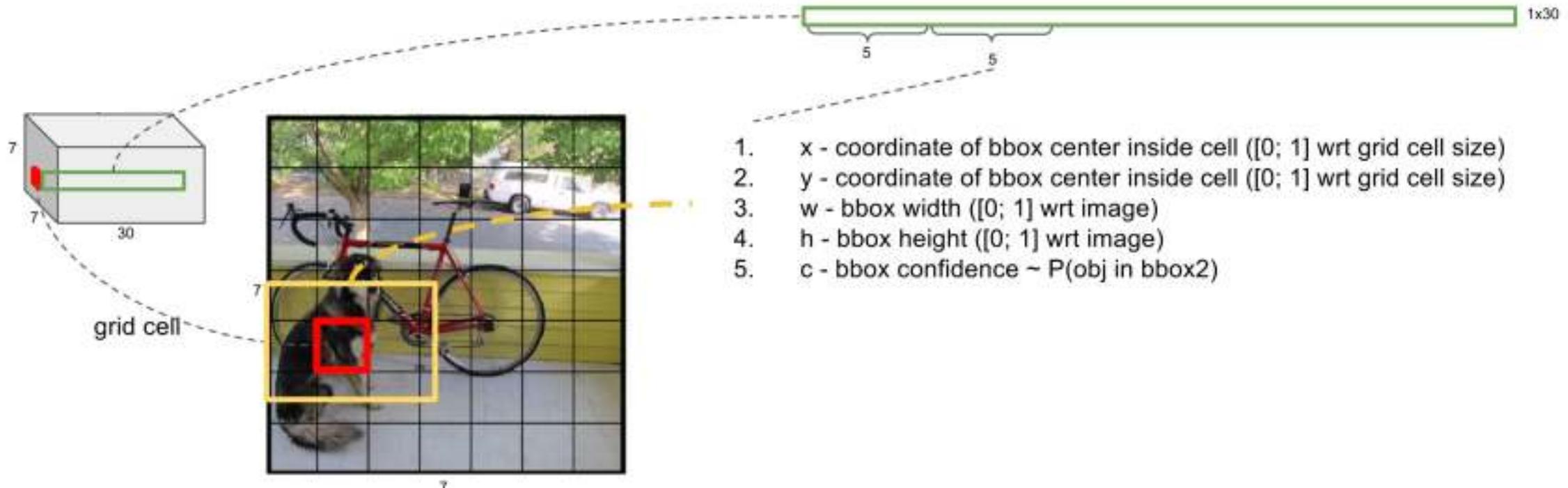
Source: deepsystem.ai

Inference



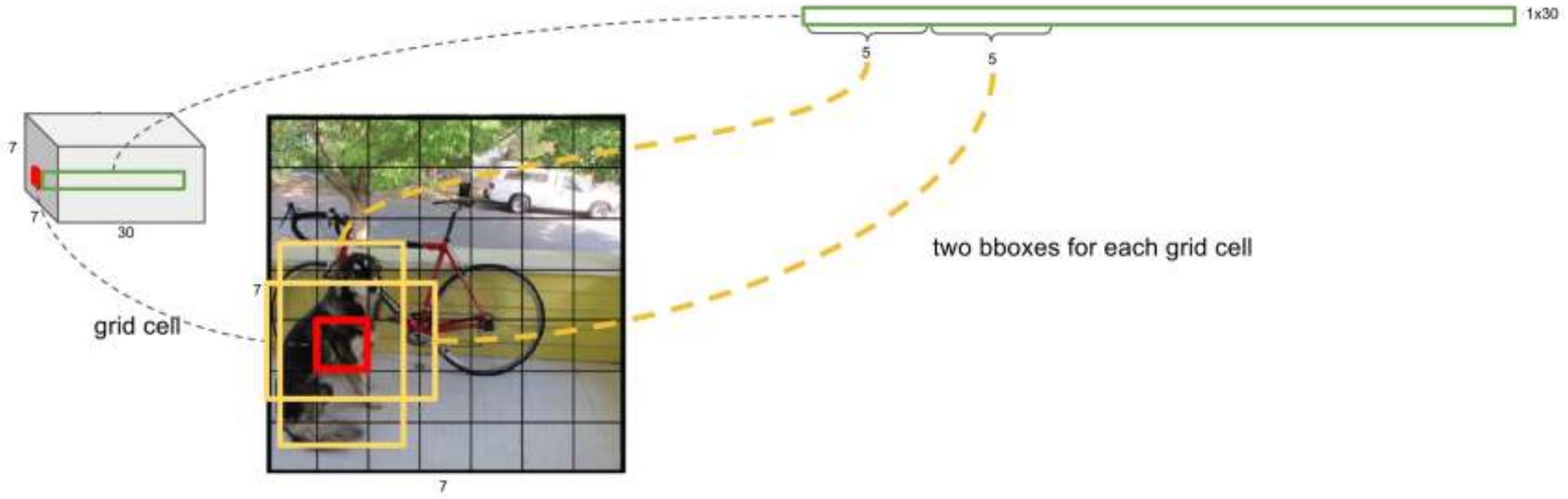
Source: deepsystem.ai

Inference



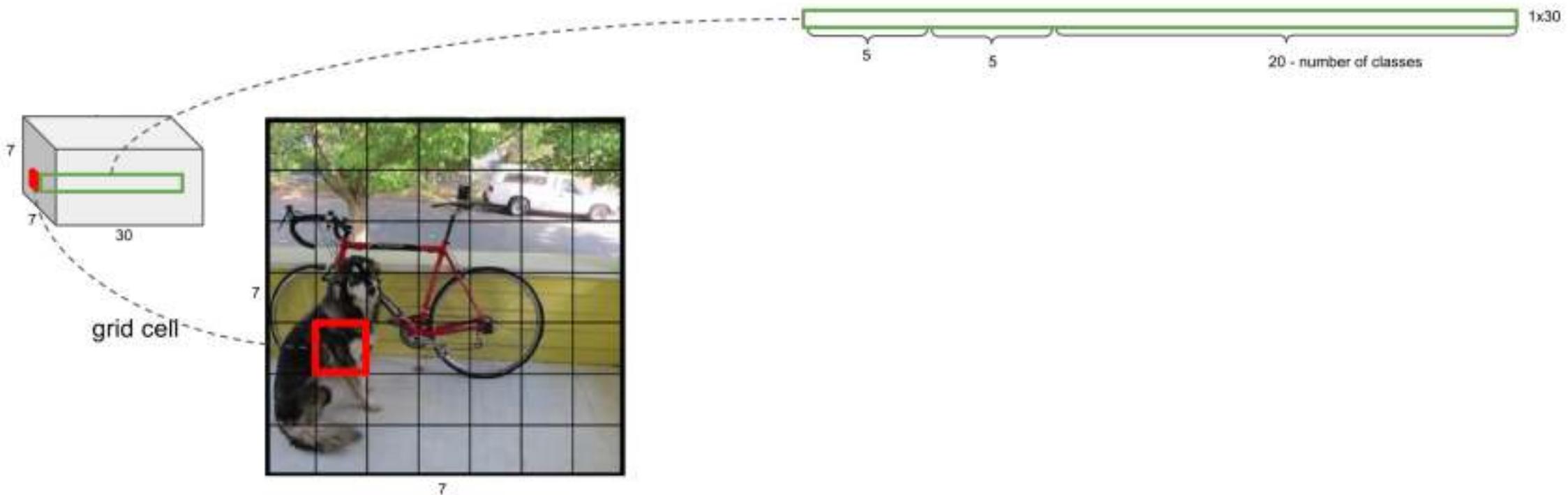
Source: deepsystem.ai

Inference



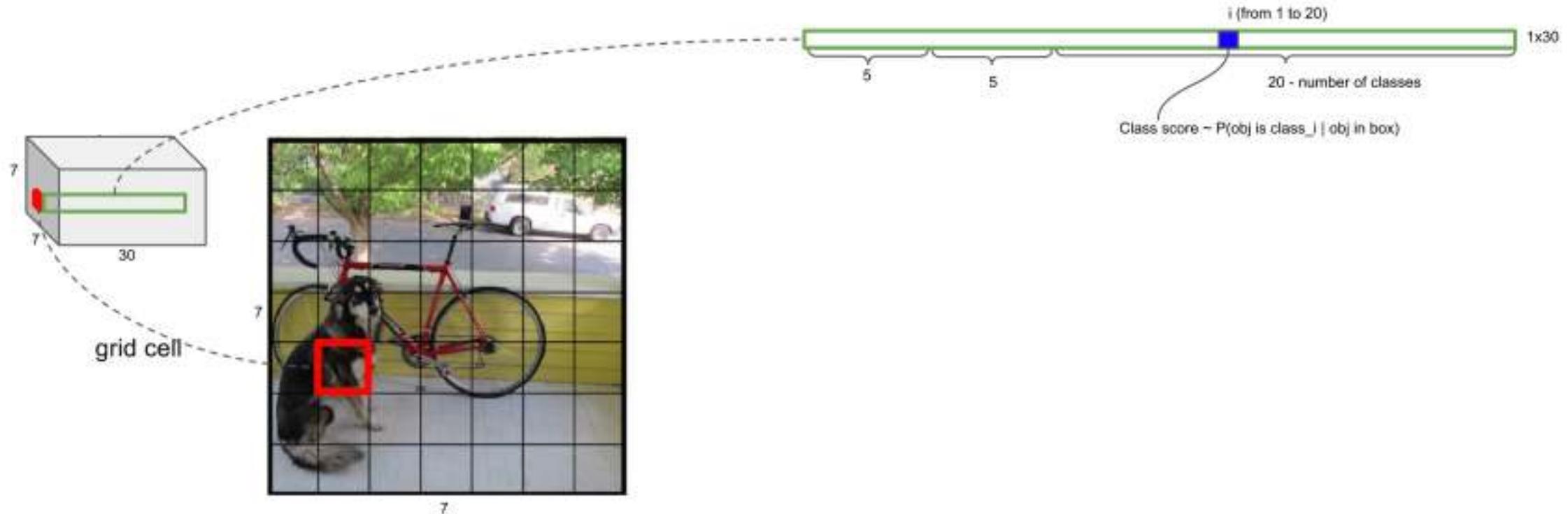
Source: deepsystem.ai

Inference



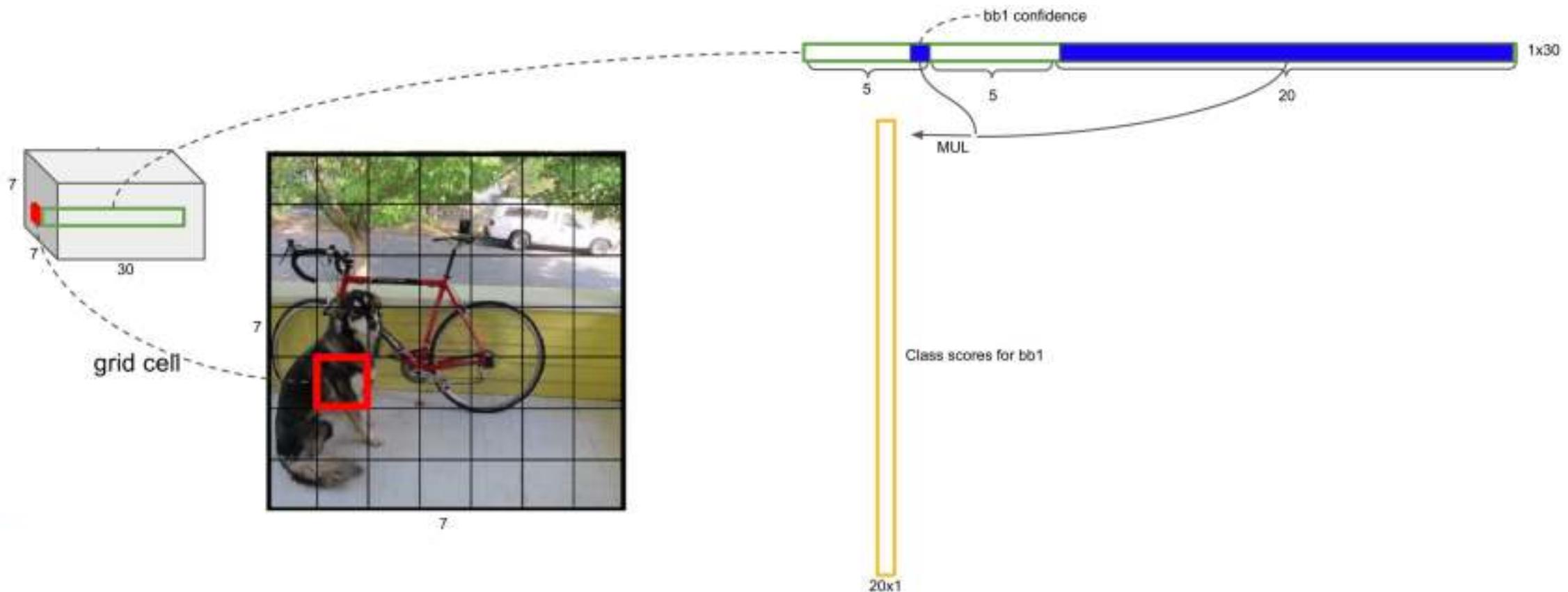
Source: deepsystem.ai

Inference



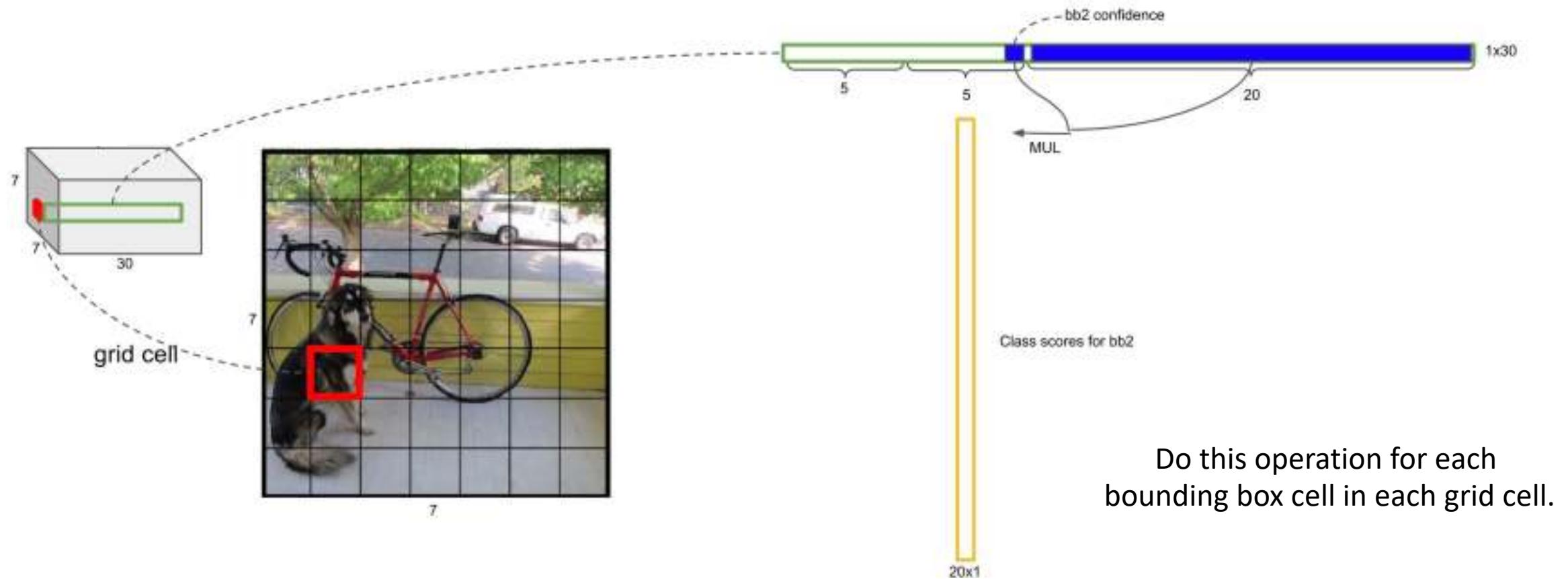
Source: deepsystem.ai

Inference



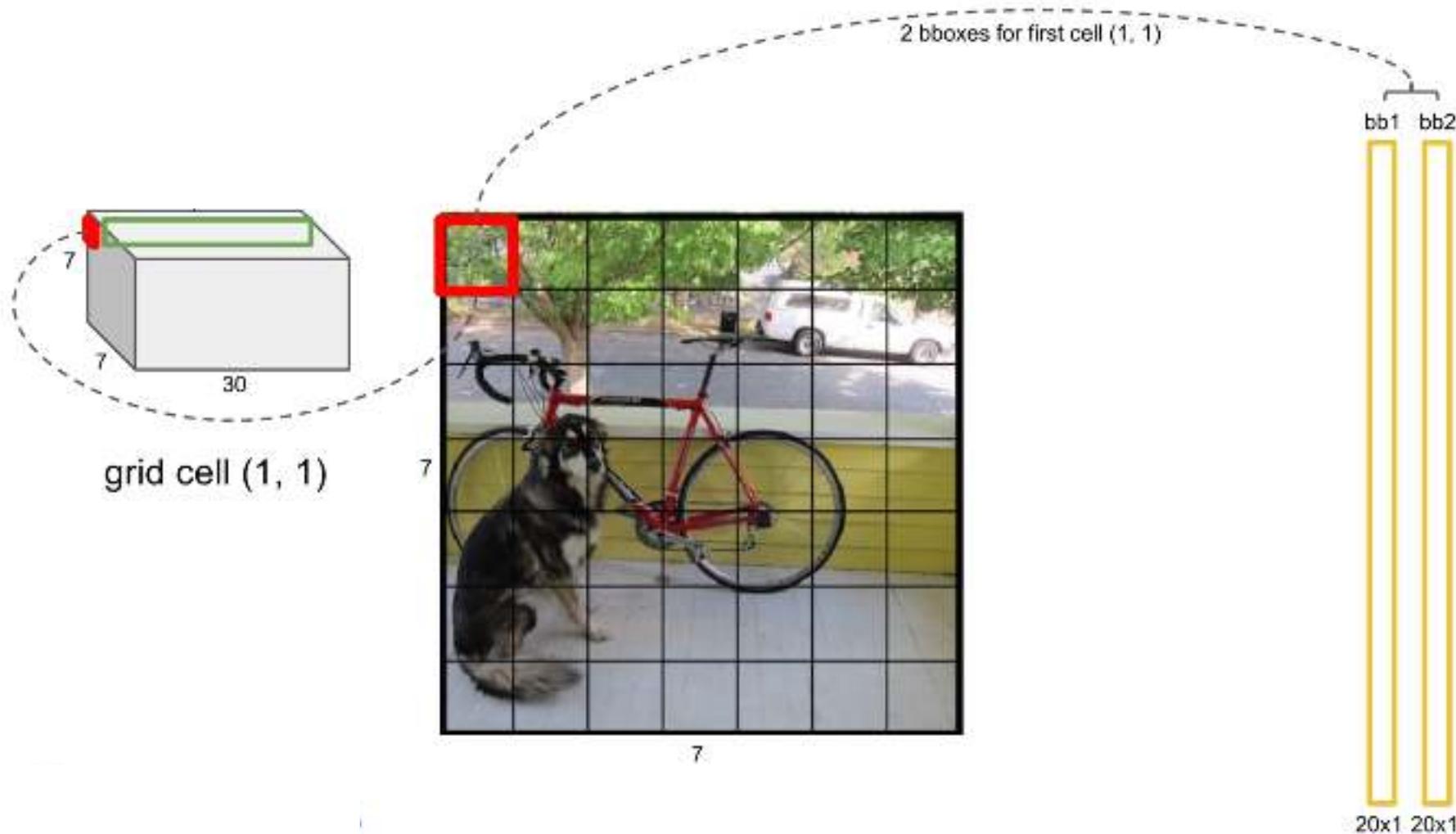
Source: deepsystem.ai

Inference



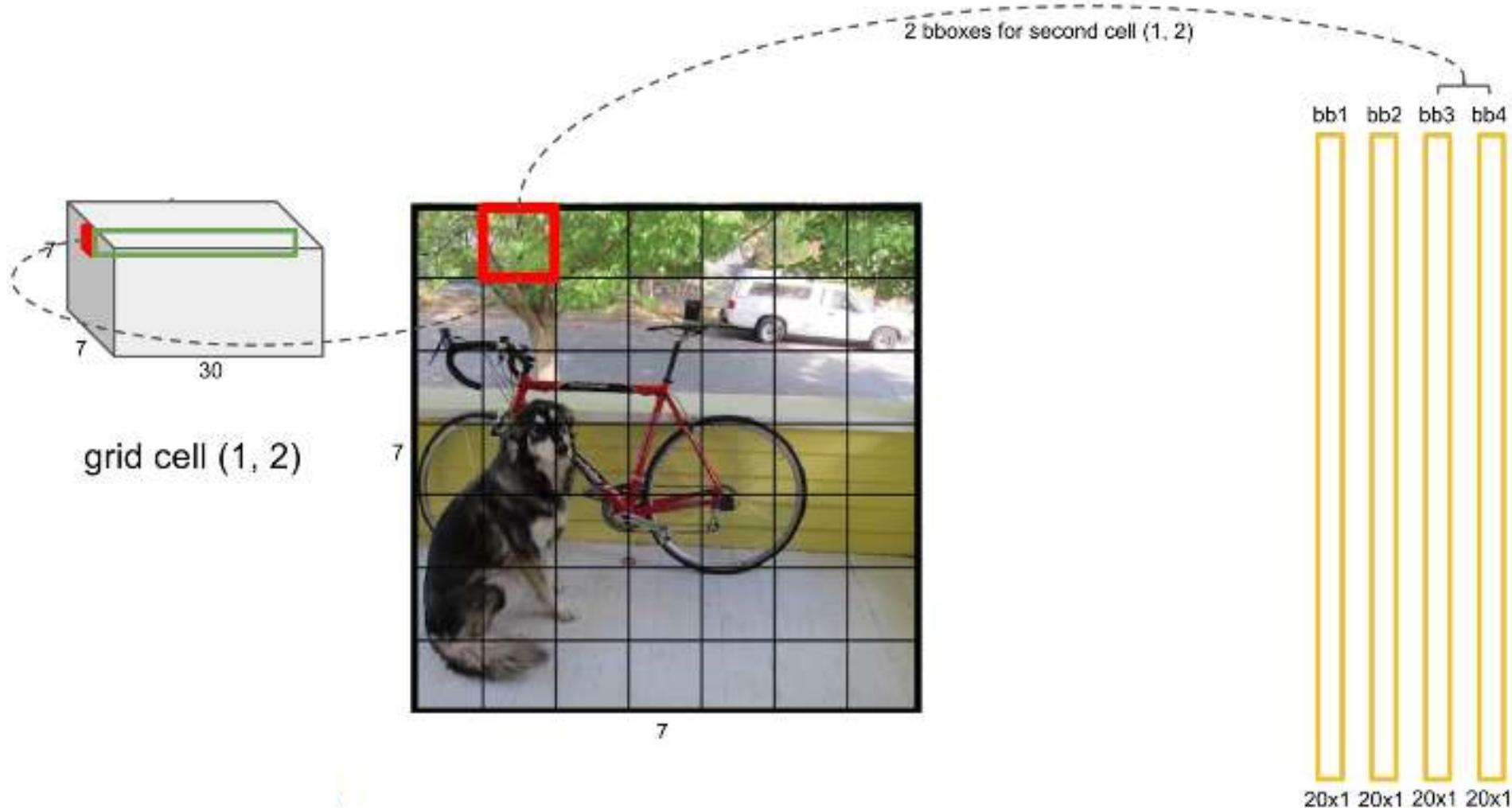
Source: deepsystem.ai

Inference



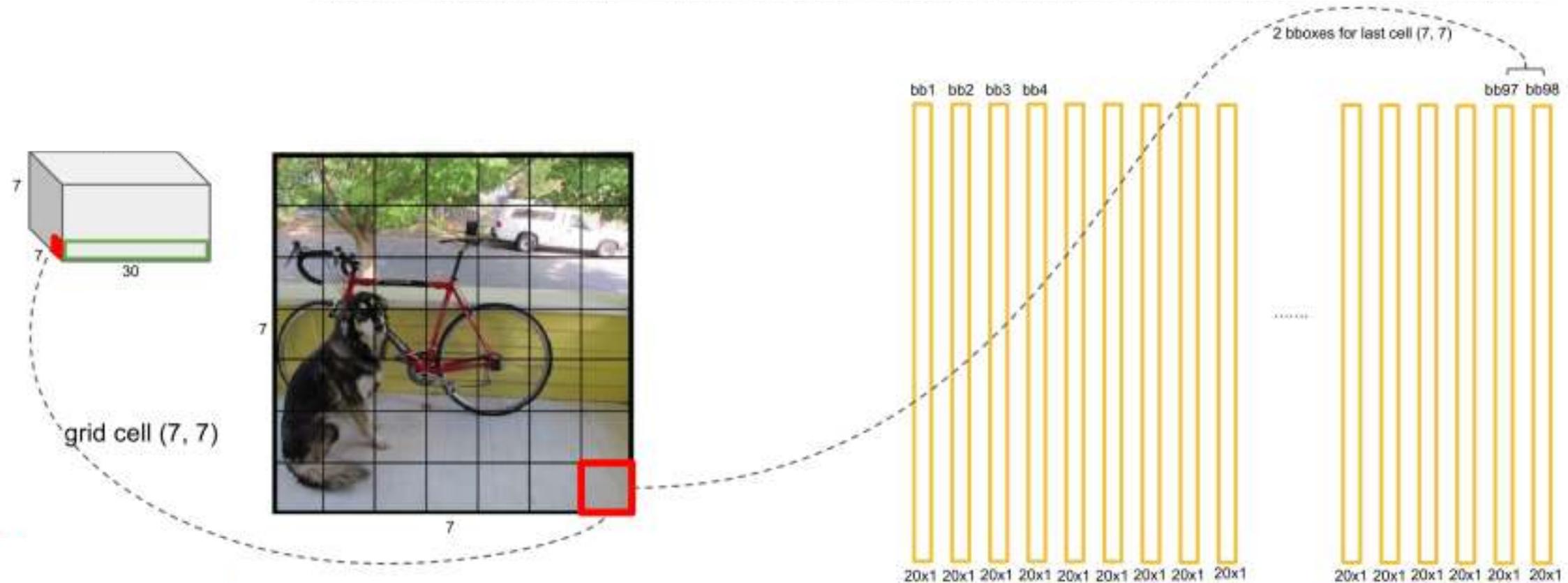
Source: deepsystem.ai

Inference



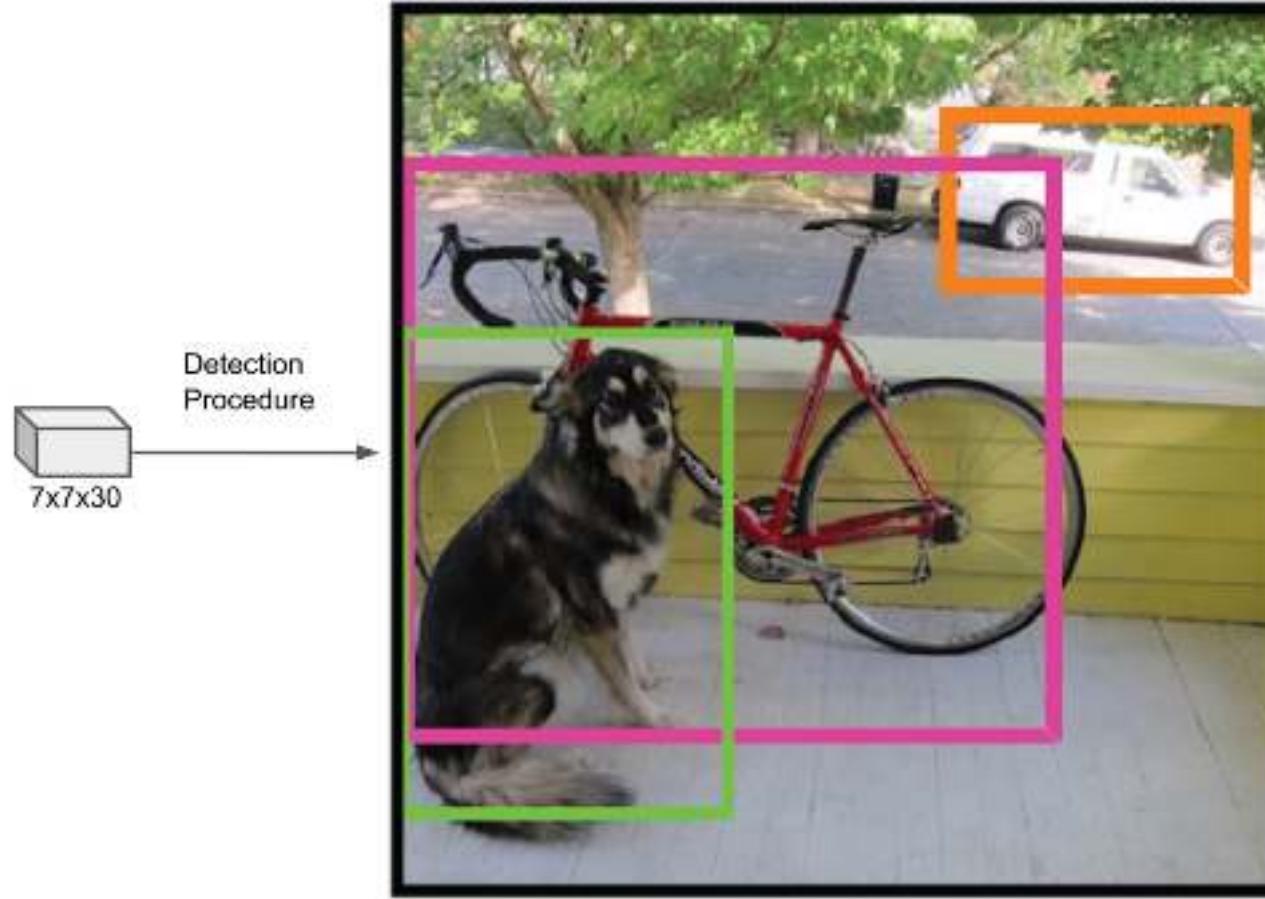
Source: deepsystem.ai

Inference



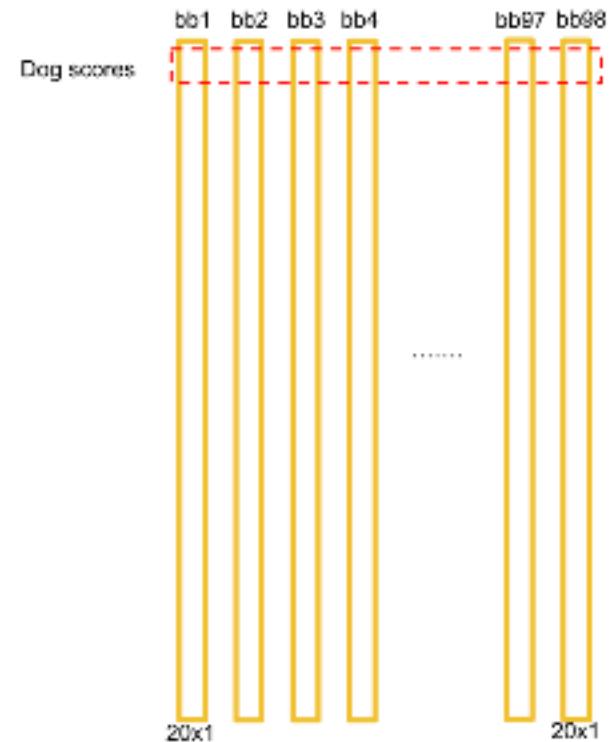
Source: deepsystem.ai

Detection Procedure



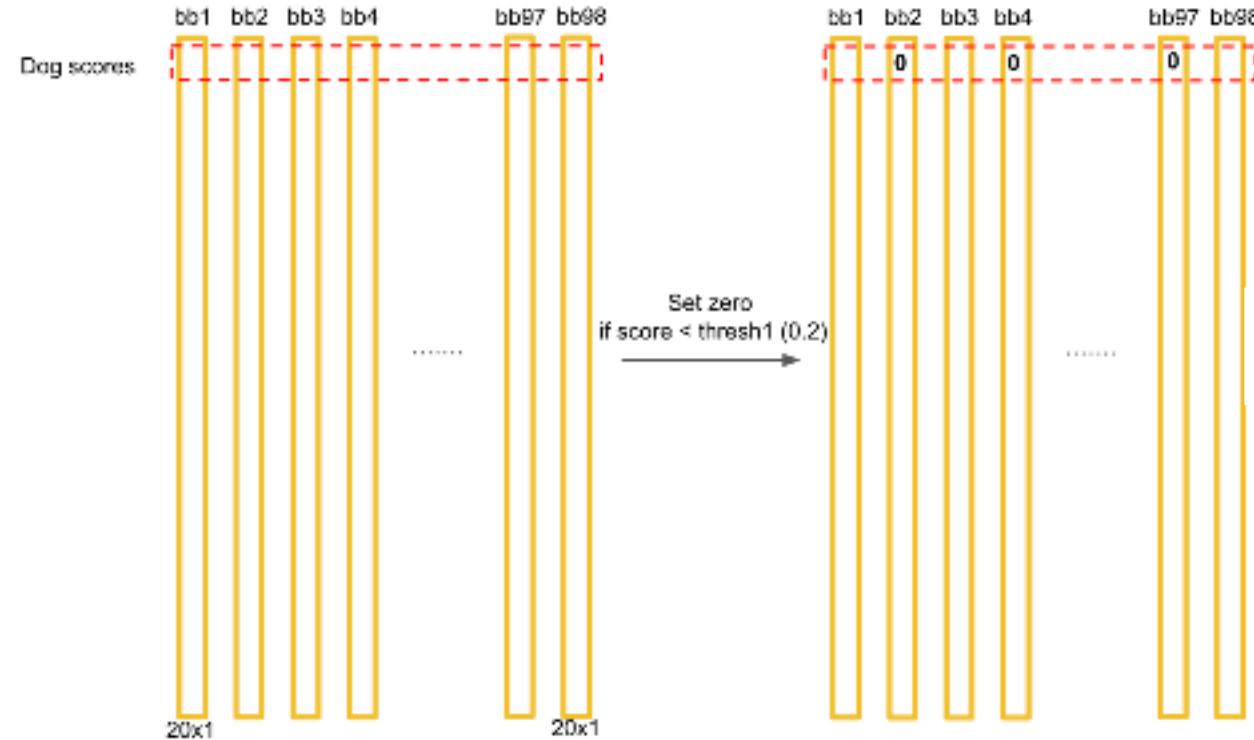
Source: deepsystem.ai

Class score for each boundary box



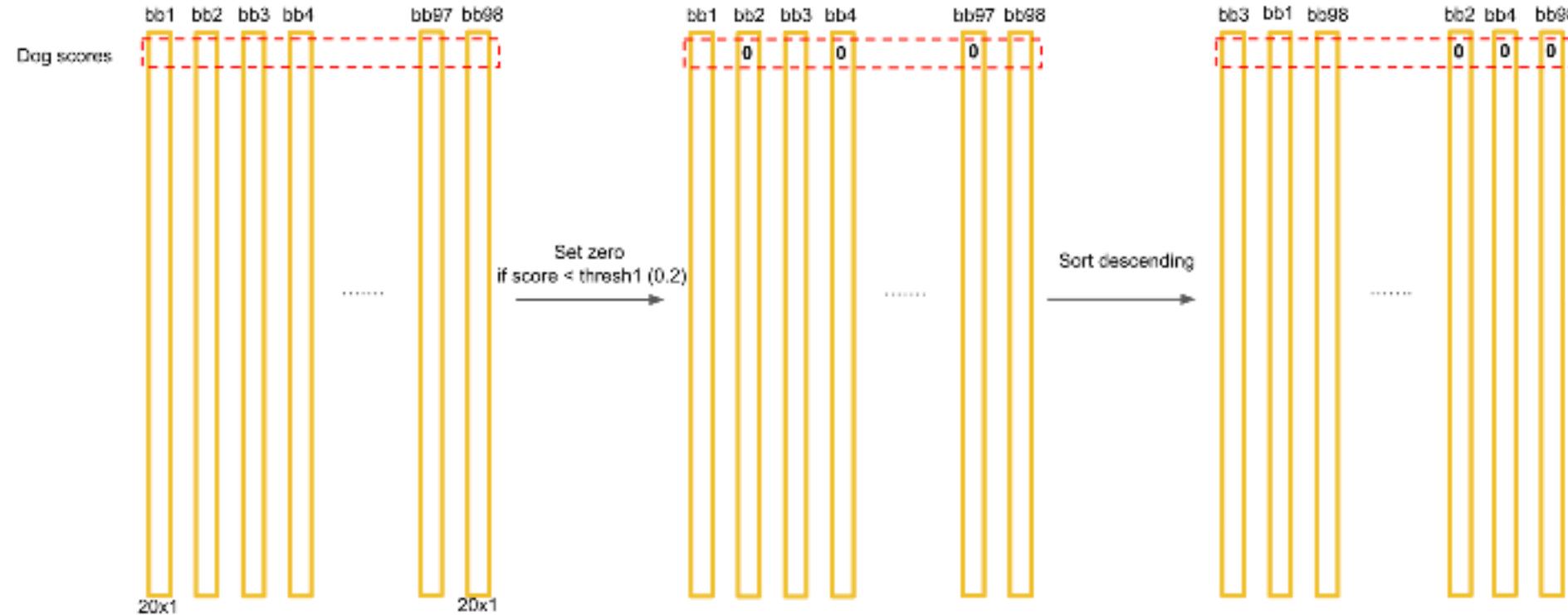
Source: deepsystem.ai

Class score for each boundary box



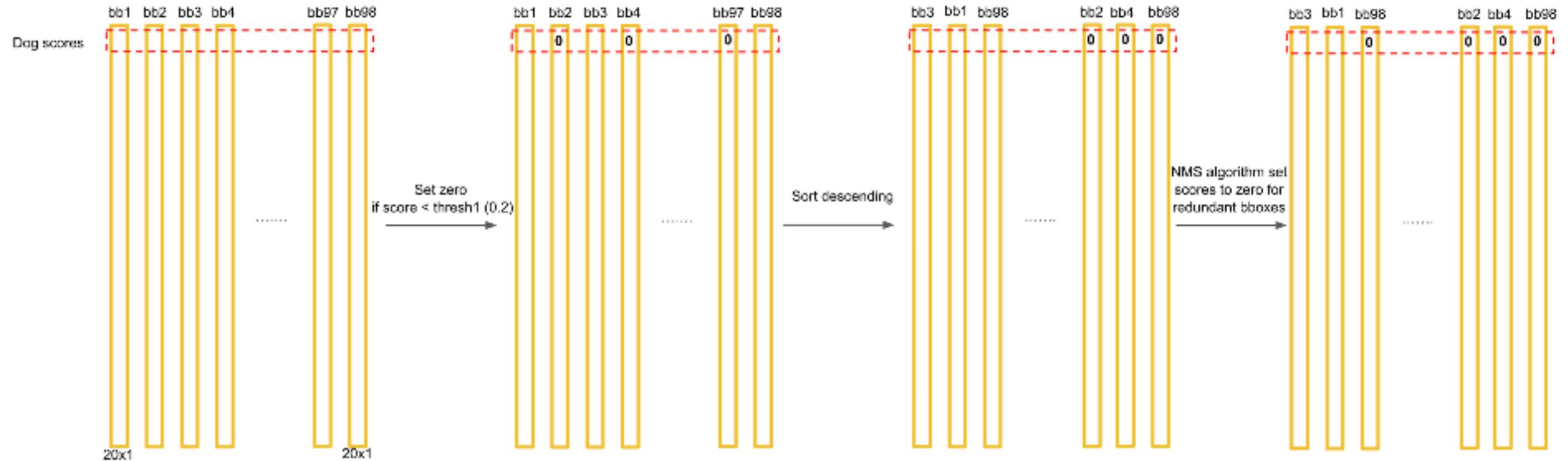
Source: deepsystem.ai

Class score for each boundary box



Source: deepsystem.ai

Class score for each boundary box



Non-maximum Suppression: Intuition



Source: deepsystem.ai

Non-maximum Suppression: Intuition



Source: deepsystem.ai

Non-maximum Suppression: Intuition



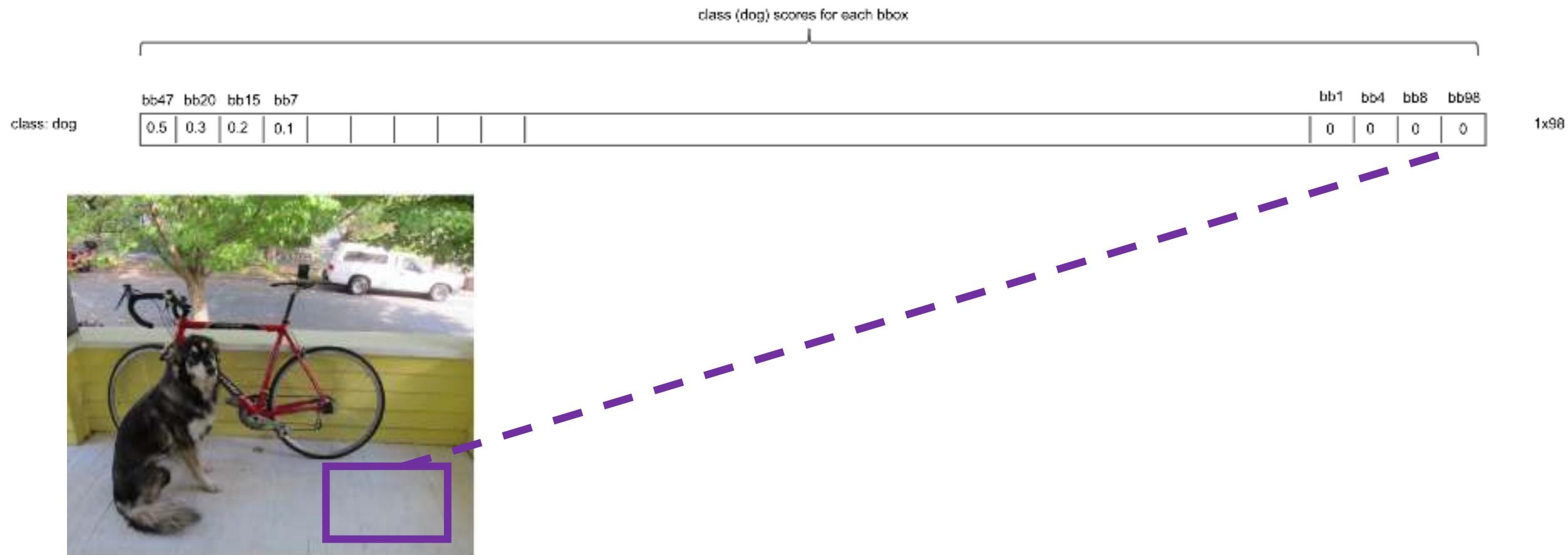
Source: deepsystem.ai

Non-maximum Suppression: Intuition



Source: deepsystem.ai

Non-maximum Suppression: Intuition



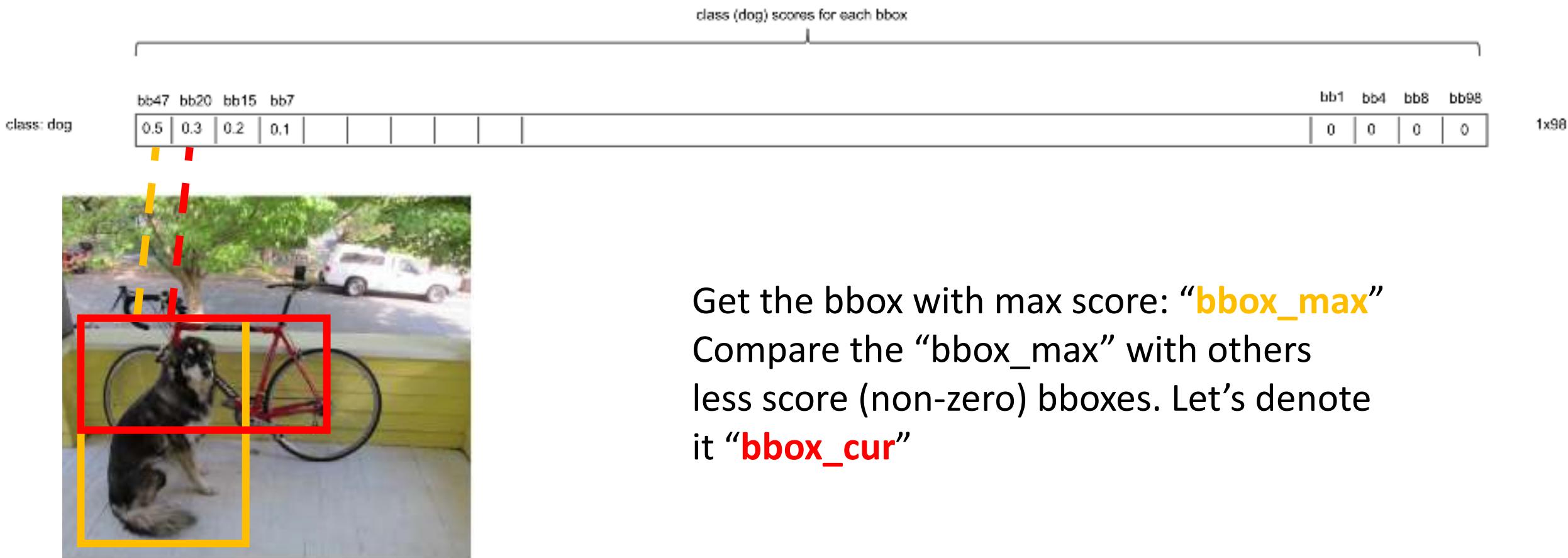
Source: deepsystem.ai

Non-maximum Suppression: Intuition



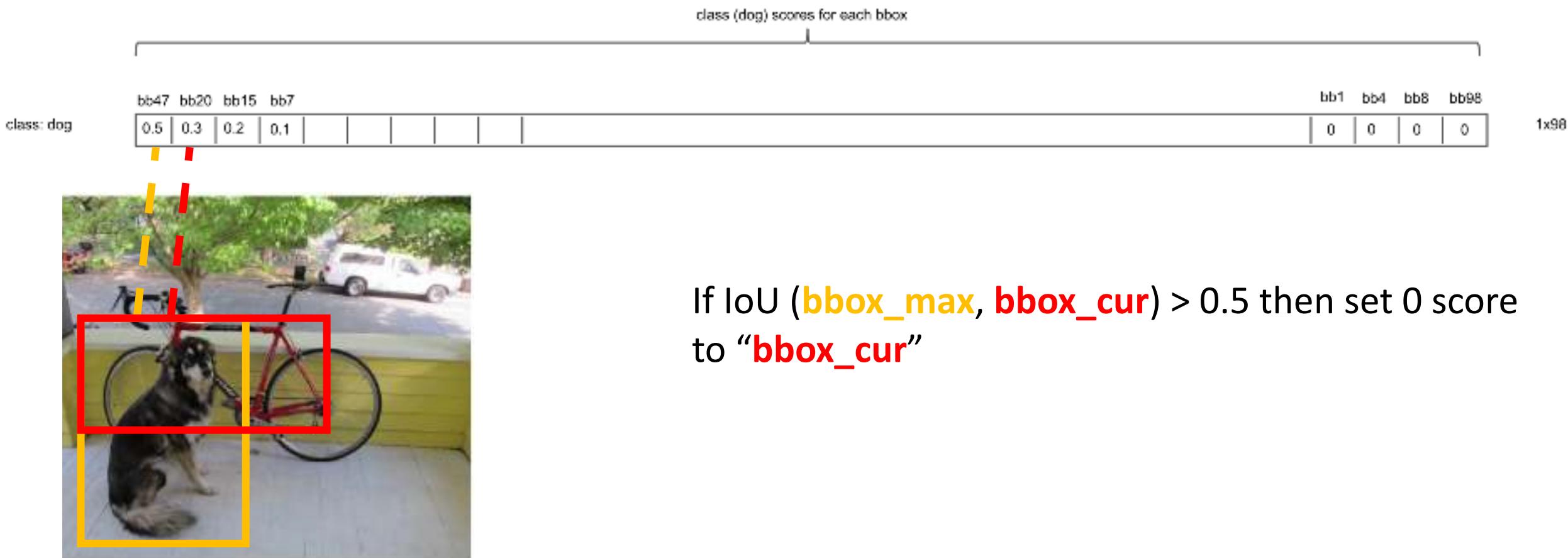
Source: deepsystem.ai

Non-maximum Suppression: Intuition



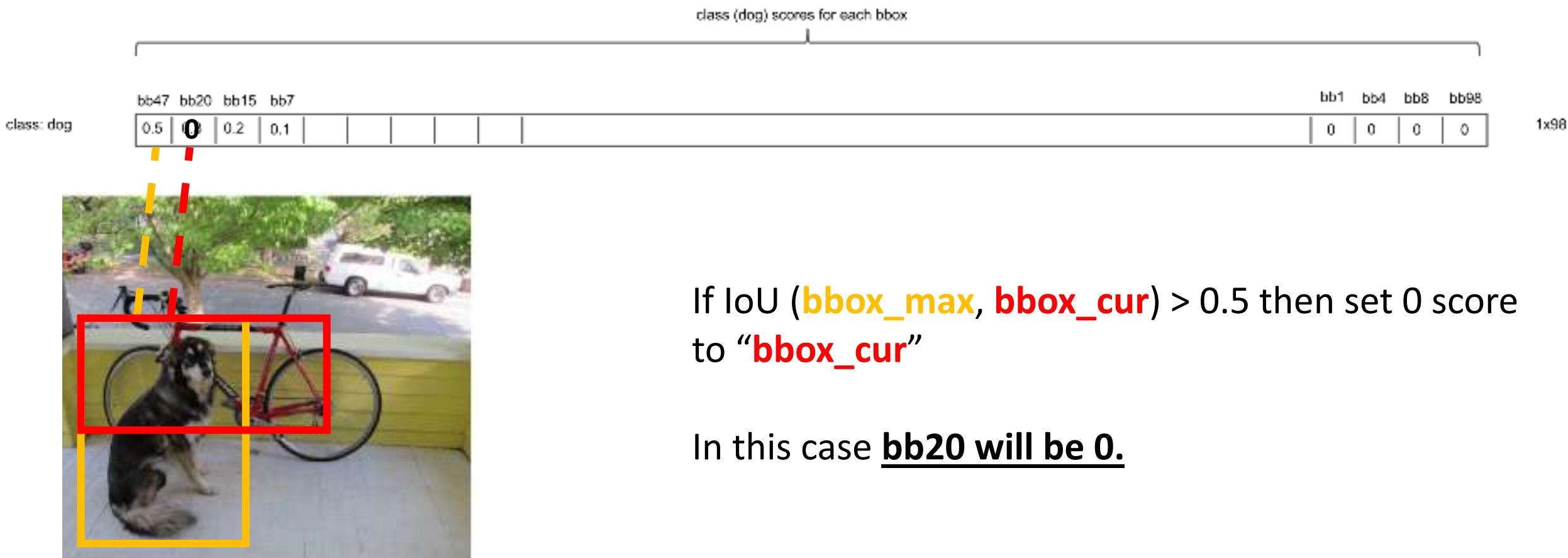
Source: deepsystem.ai

Non-maximum Suppression: Intuition



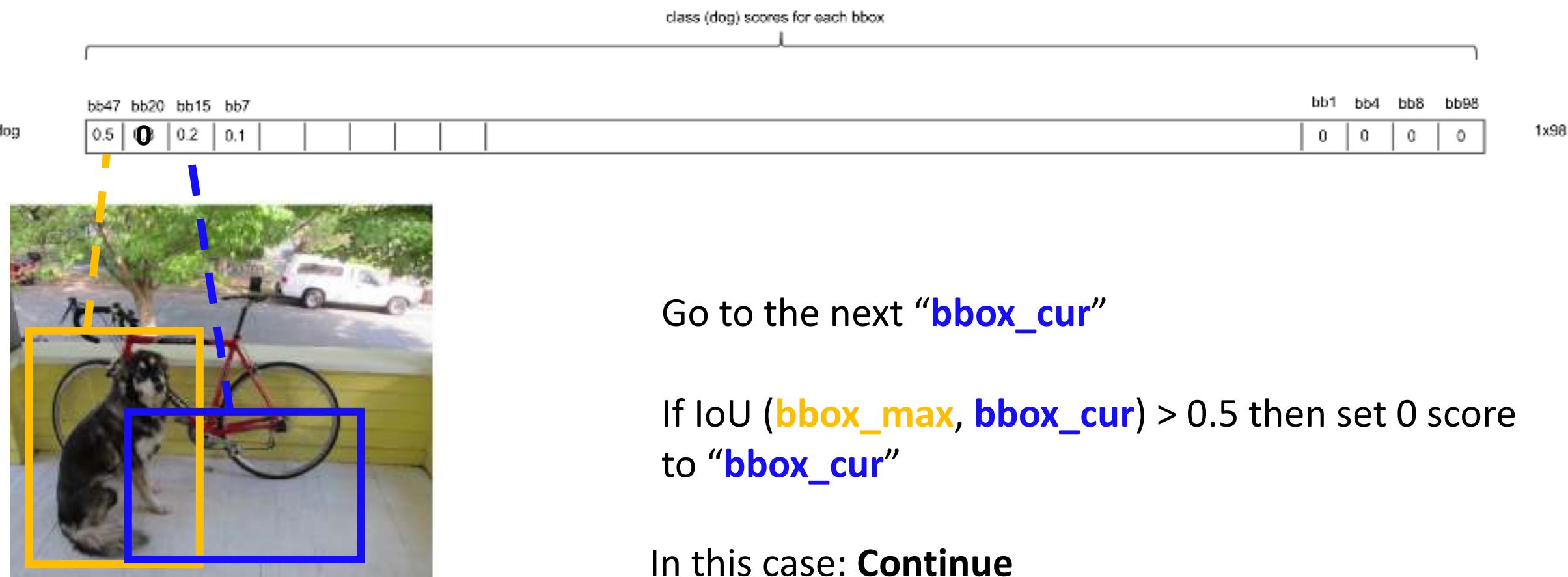
Source: deepsystem.ai

Non-maximum Suppression: Intuition



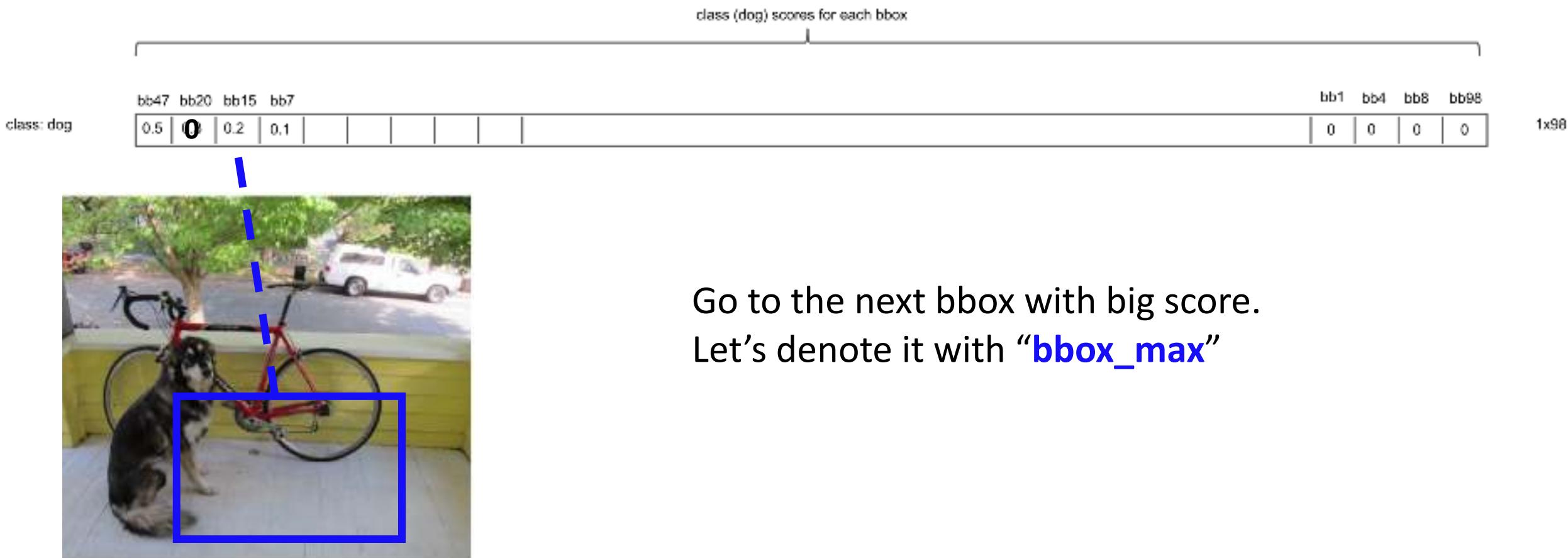
Source: deepsystem.ai

Non-maximum Suppression: Intuition



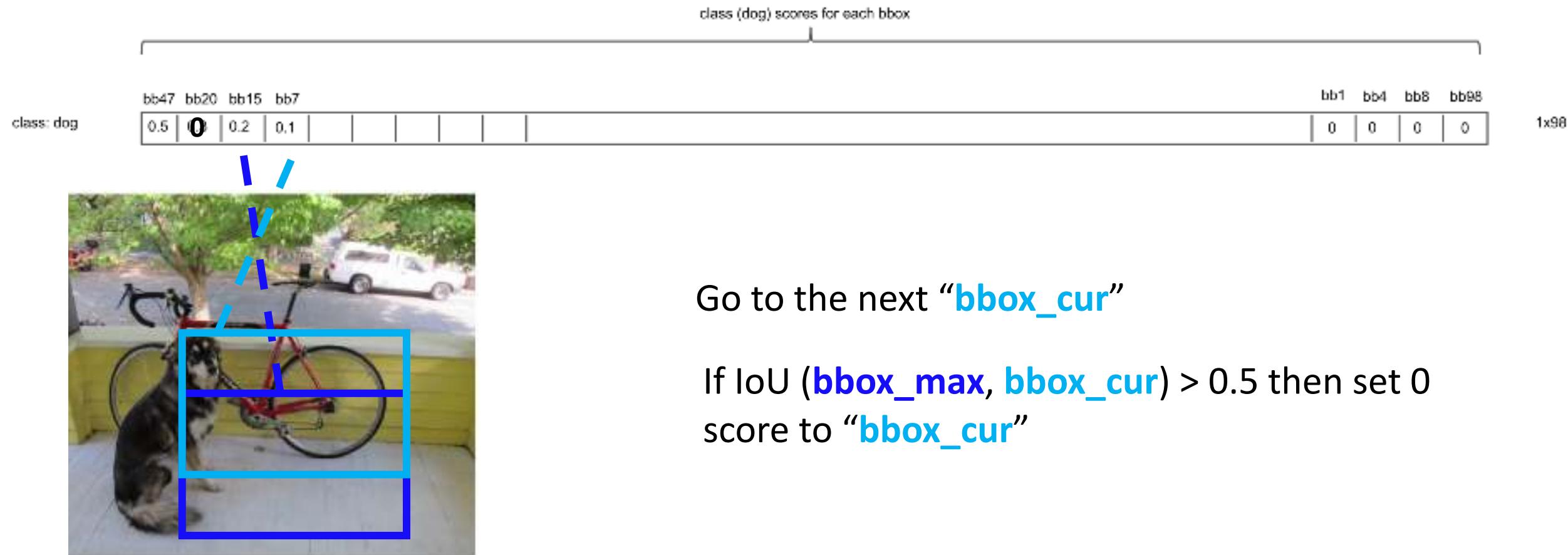
Source: deepsystem.ai

Non-maximum Suppression: Intuition



Source: deepsystem.ai

Non-maximum Suppression: Intuition

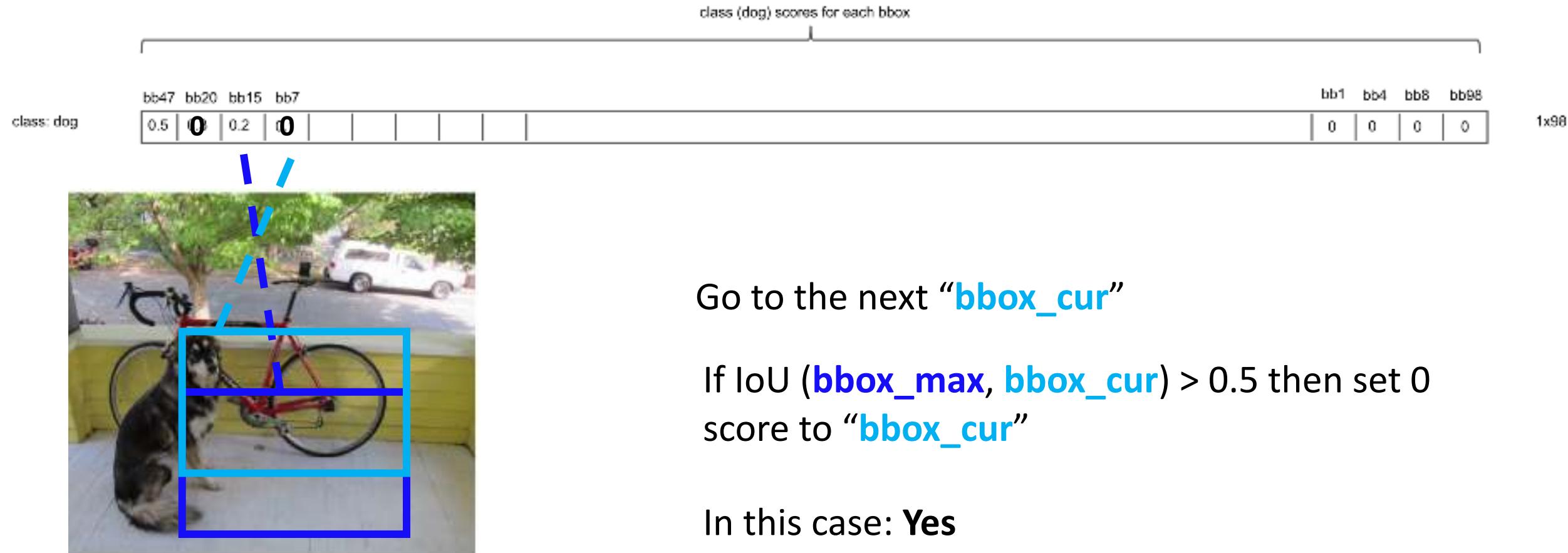


Go to the next “bbox_cur”

If $\text{IoU}(\text{bbox_max}, \text{bbox_cur}) > 0.5$ then set 0 score to “**`bbox_cur`**”

Source: deepsystem.ai

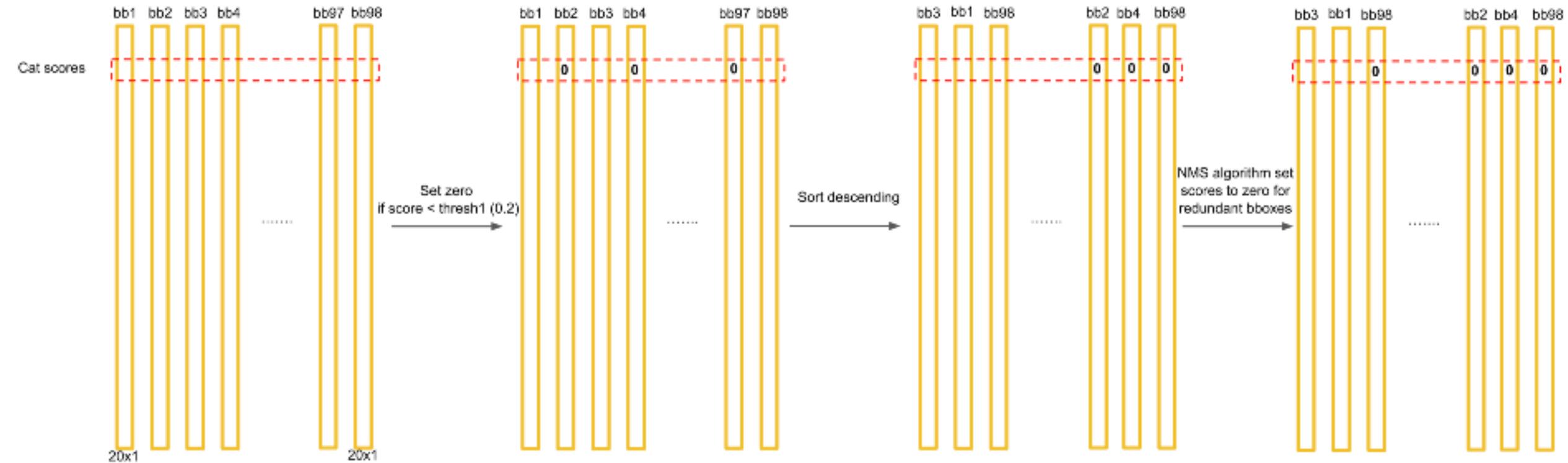
Non-maximum Suppression: Intuition



Do this procedure for other “bbox max” and other corresponding “bbox cur”

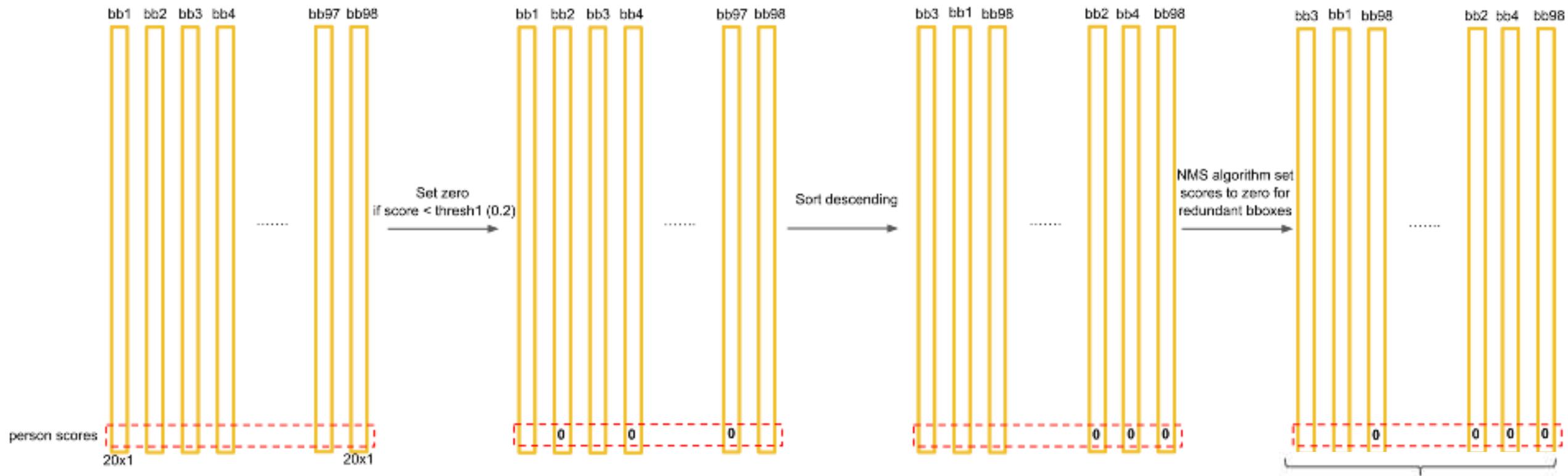
Source: deepsystem.ai

Non-maximum Suppression: Intuition



Source: deepsystem.ai

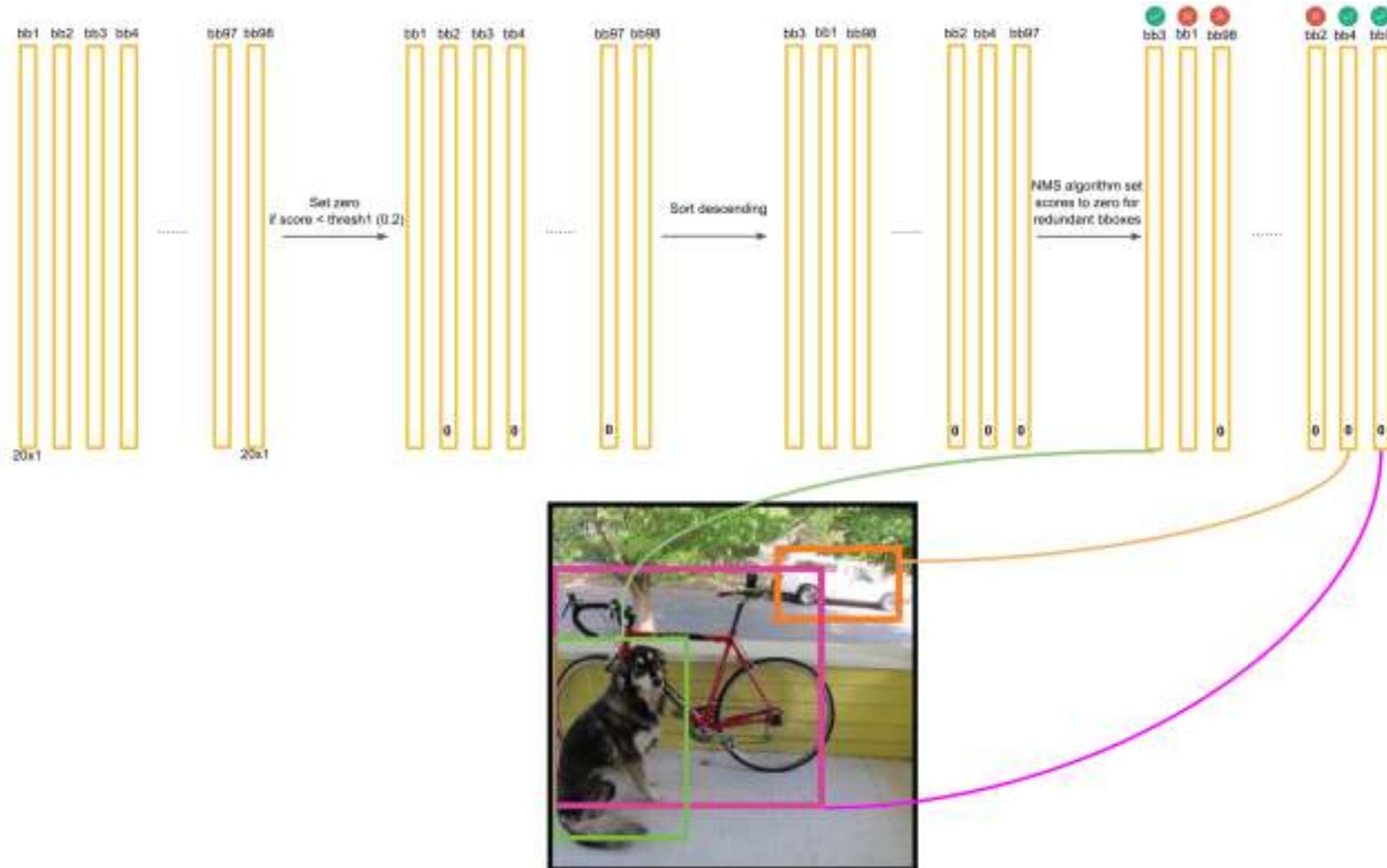
Non-maximum Suppression: Intuition



After this procedure -
a lot of zeros

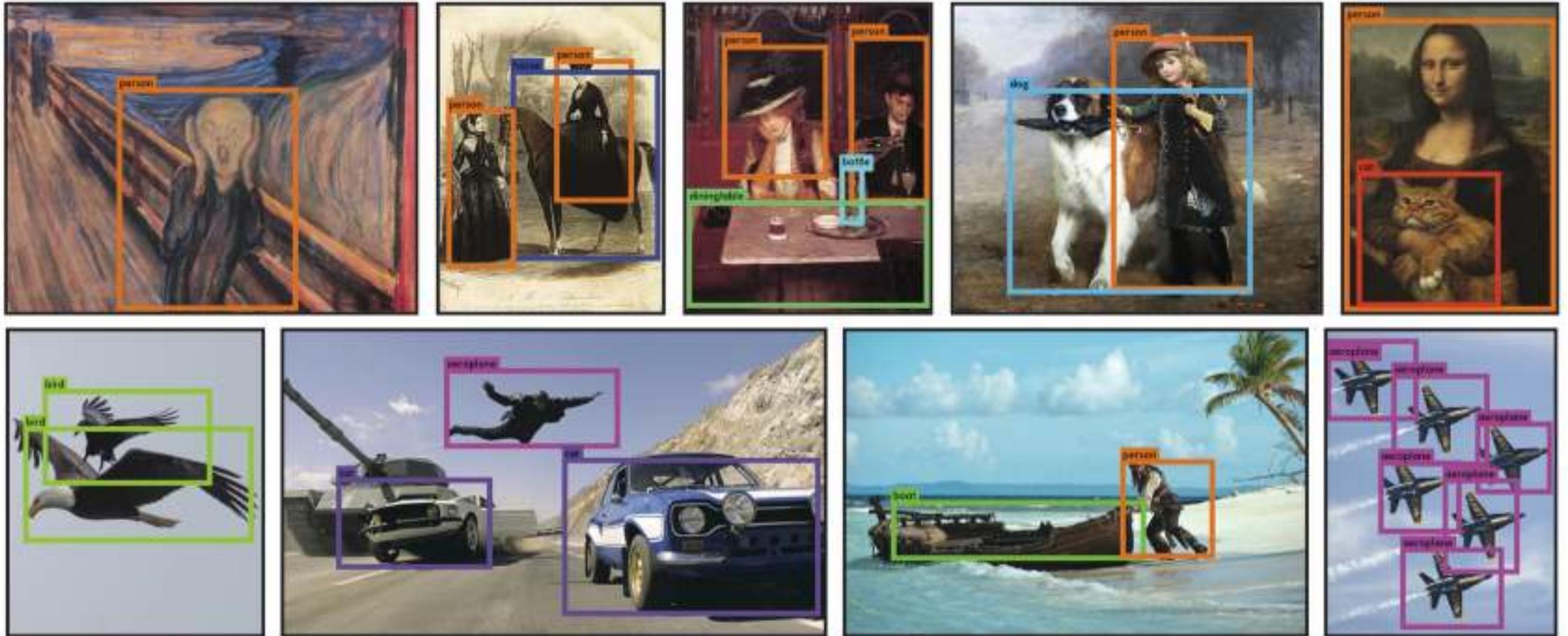
Source: deepsystem.ai

Finally!!!



Source: deepsystem.ai

Generalization: Artwork



Reading Material

- Original Paper: <https://arxiv.org/pdf/1506.02640.pdf>
- Recent work: <https://github.com/WongKinYiu/yolov7>

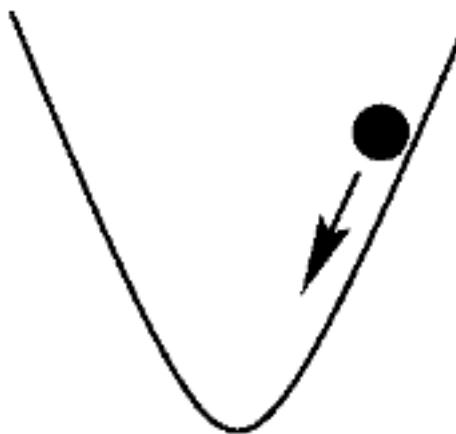
Thank You 😊

Appendix: Receptive Field Calculator

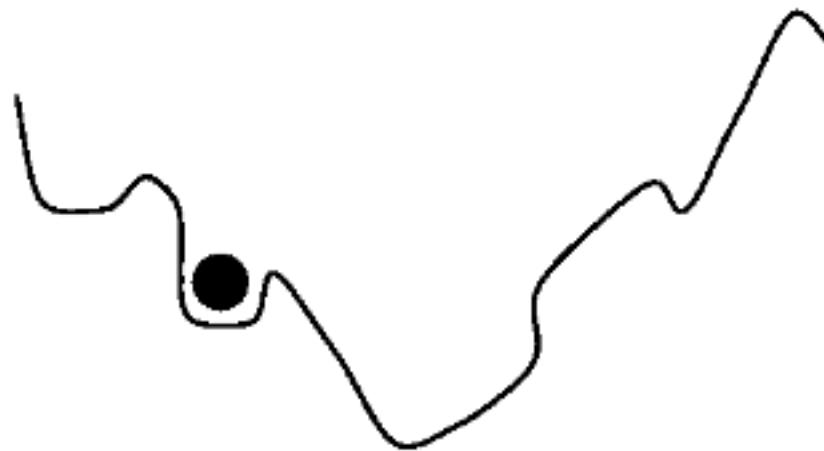
- <https://fomoro.com/research/article/receptive-field-calculator#5,1,1,VALID>

Appendix: Momentum

In neural networks, we use gradient descent optimization algorithm to minimize the error function to reach a global minima. In an ideal world the error function would look like this



So you are guaranteed to find the global optimum because there are no local minimum where your optimization can get stuck. However in real the error surface is more complex, may comprise of several local minima and may look like this



In this case, you can easily get stuck in a local minima and the algorithm may think you reach the global minima leading to sub-optimal results. To avoid this situation, we use a momentum term in the objective function, which is a value between 0 and 1 that increases the size of the steps taken towards the minimum by trying to jump from a local minima. If the momentum term is large then the learning rate should be kept smaller. A large value c

Available Datasets: MNIST

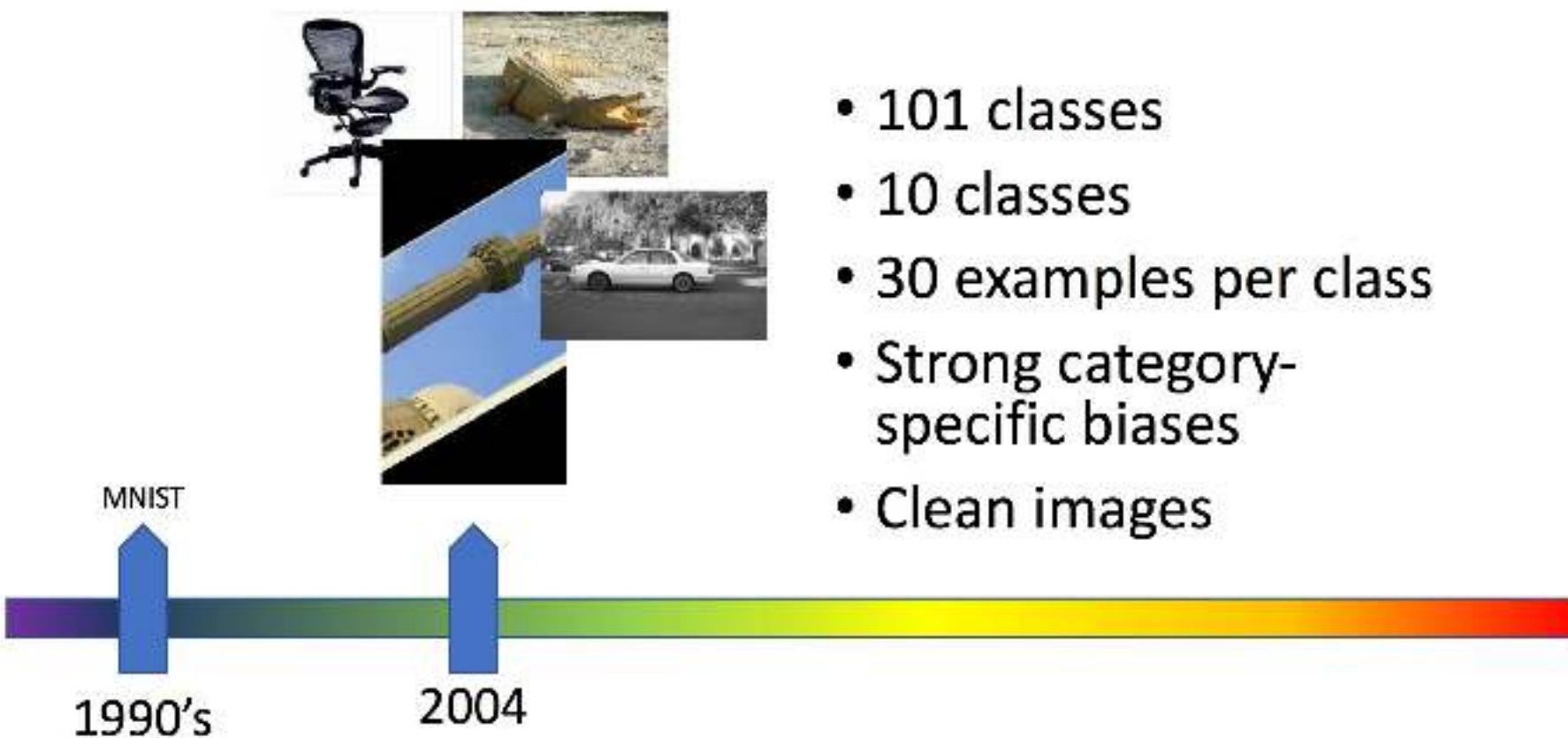


- 2D
- 10 classes
- 6000 examples per class



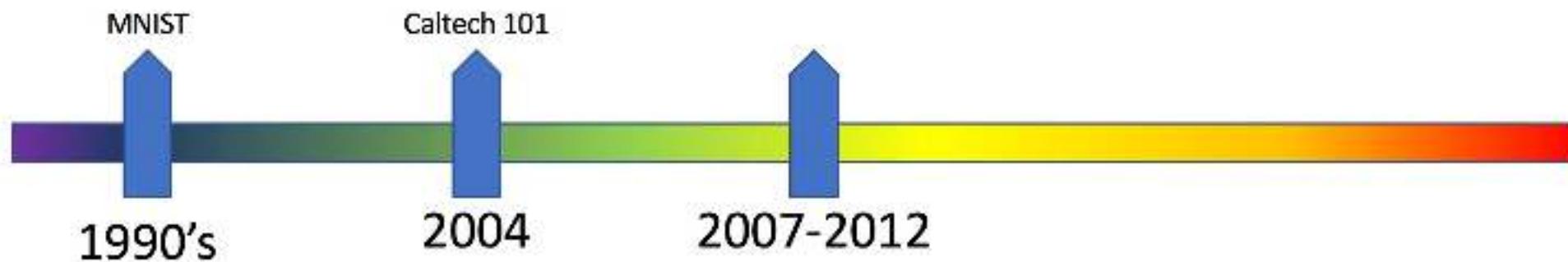
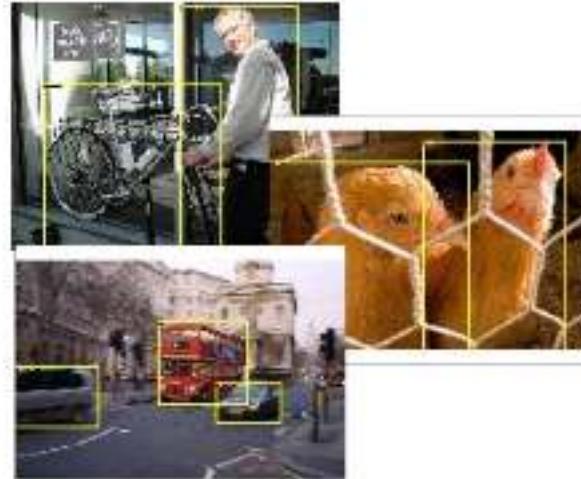
1990's

Available Datasets: Caltech 101



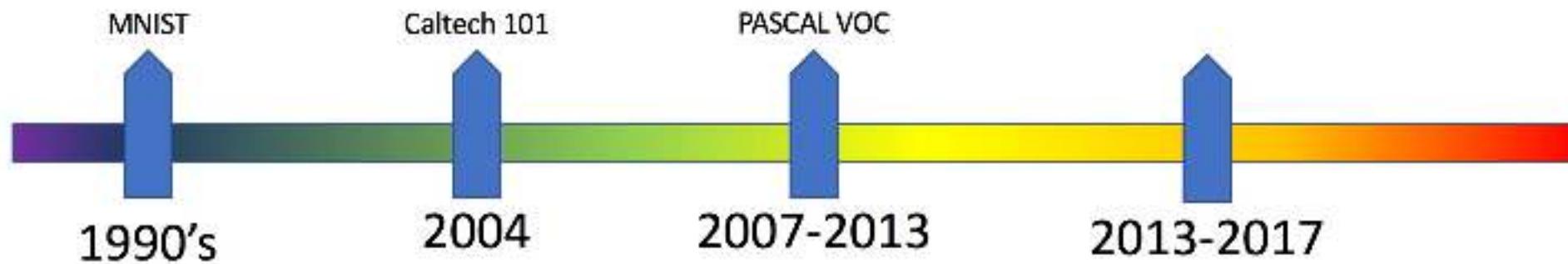
Available Datasets: PASCAL VOC

- 20 classes
 - ~500 examples per class
 - Clutter, occlusion, natural scenes



Available Datasets: ImageNet

- 1000 classes
- ~1000 examples per class
- Mix of cluttered and clean images



Generative Adversarial Networks (GAN) for Computer vision

Instructor: Dr. Muhammad Fahim

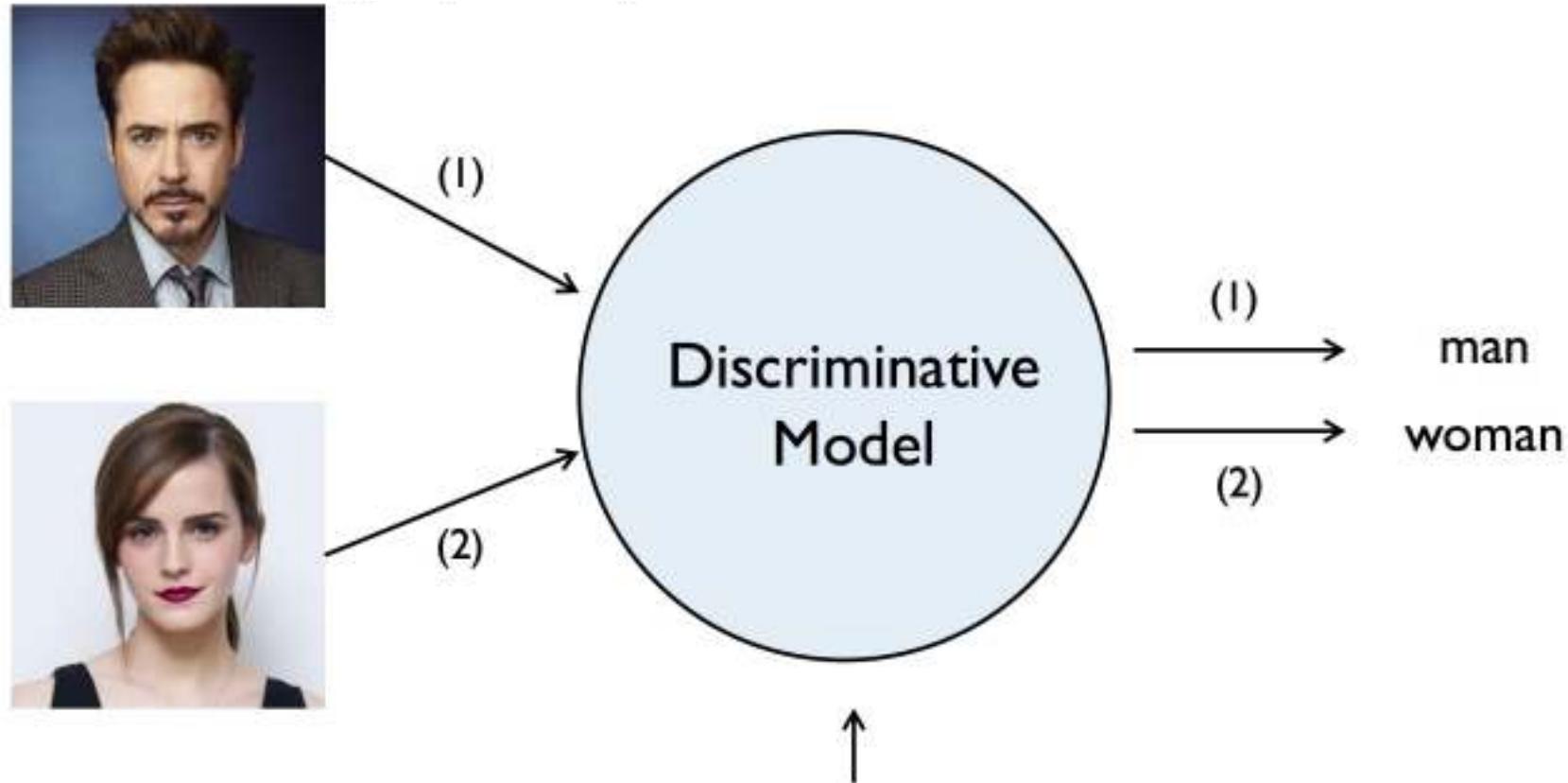
Slide Credit and Source Material

- This lecture is based on the following material and some other resources over the internet
 - https://www.uio.no/studier/emner/matnat/ifi/INF5860/v18/undervisningsmateriale/lectures/slides_inf5860_s18_week14.pdf#page=85&zoom=100,0,108
 - Goodfellow, Ian. "NIPS 2016 Tutorial: Generative Adversarial Networks." arXiv preprint arXiv:1701.00160 (2016).
 - <https://github.com/hwalsuklee/tensorflow-generative-model-collections>
 - <https://medium.com/jungle-book/towards-data-set-augmentation-with-gans-9dd64e9628e6>
 - Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative adversarial nets, NIPS (2014).
 - Radford, A., Metz, L. and Chintala, S., Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434. (2015).
 - Generative models and adversarial training by Kevin McGuinness

Contents

- Introduction
- Generative Adversarial Networks
- Architecture Details
- Loss Function
- Variants of GAN
- Summary

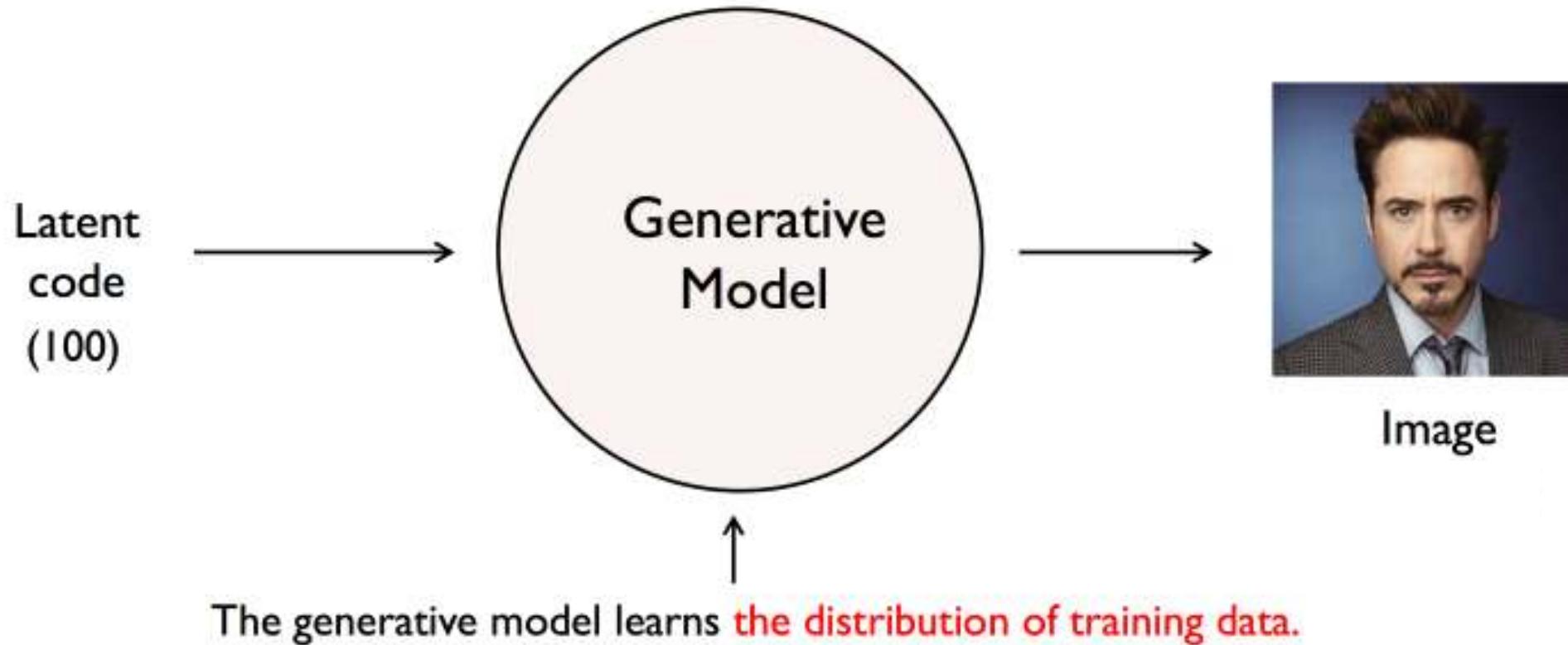
Discriminative Model



The discriminative model learns **how to classify** input to its class.

Source: Yunjey Choi, Korea University

Generative Model



Source: Yunjey Choi, Korea University

Probability Distribution

- What if x is actual images in the training data?
- At this point, x can be represented as a dimensional vector (e.g., 64x64x3)

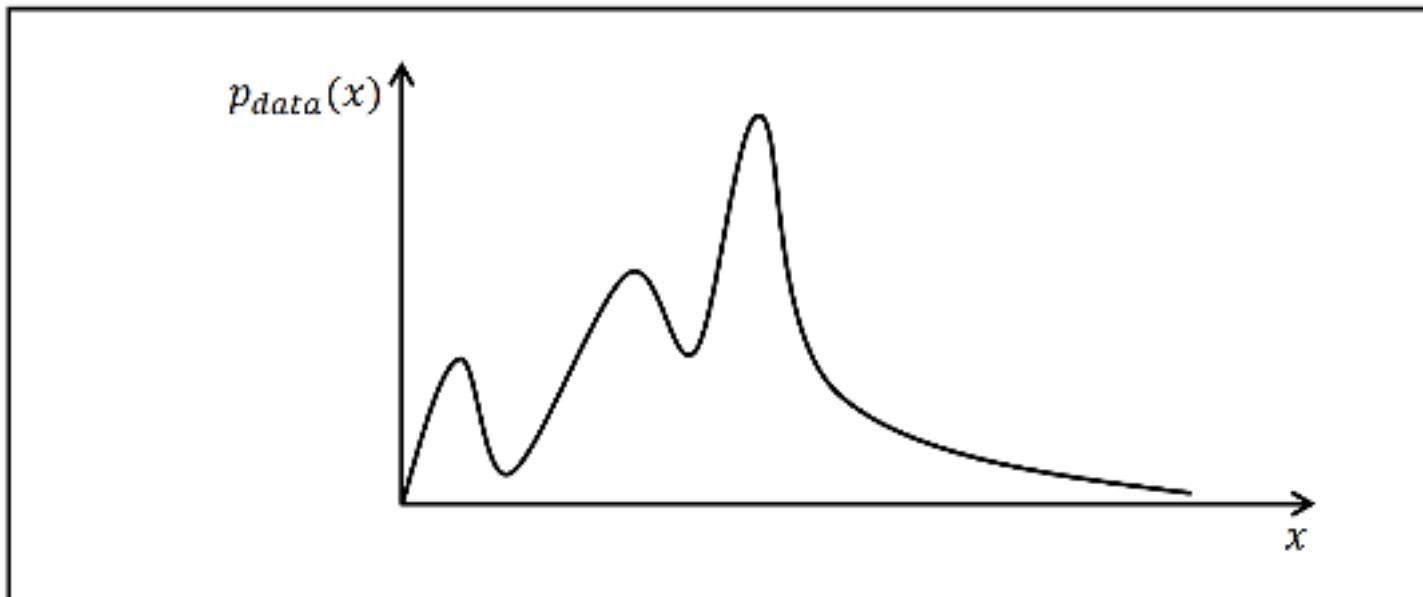


Source: Yunjey Choi, Korea University

Probability Distribution

Probability density function

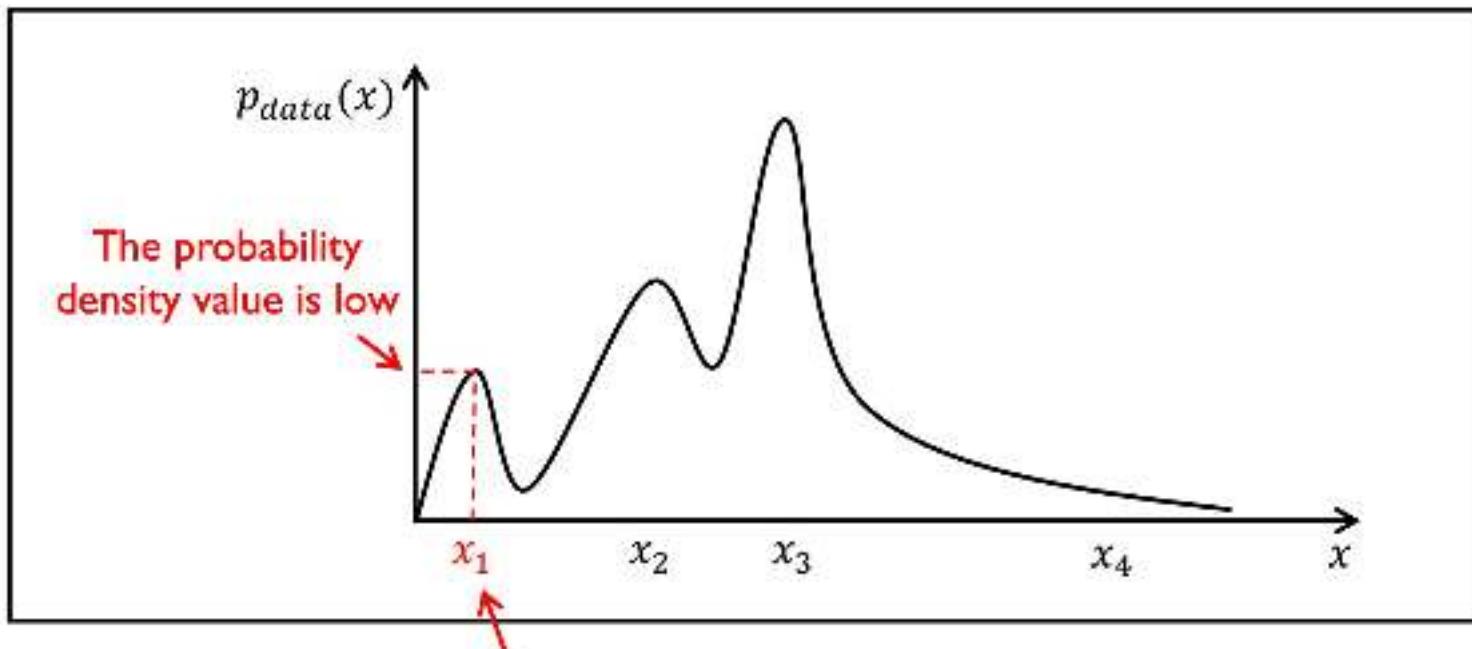
There is a $p_{data}(x)$ that represents the distribution of actual images.



Source: Yunjey Choi, Korea University

Probability Distribution

Let's take an example with human face image dataset.
Our dataset may contain few images of men with glasses.

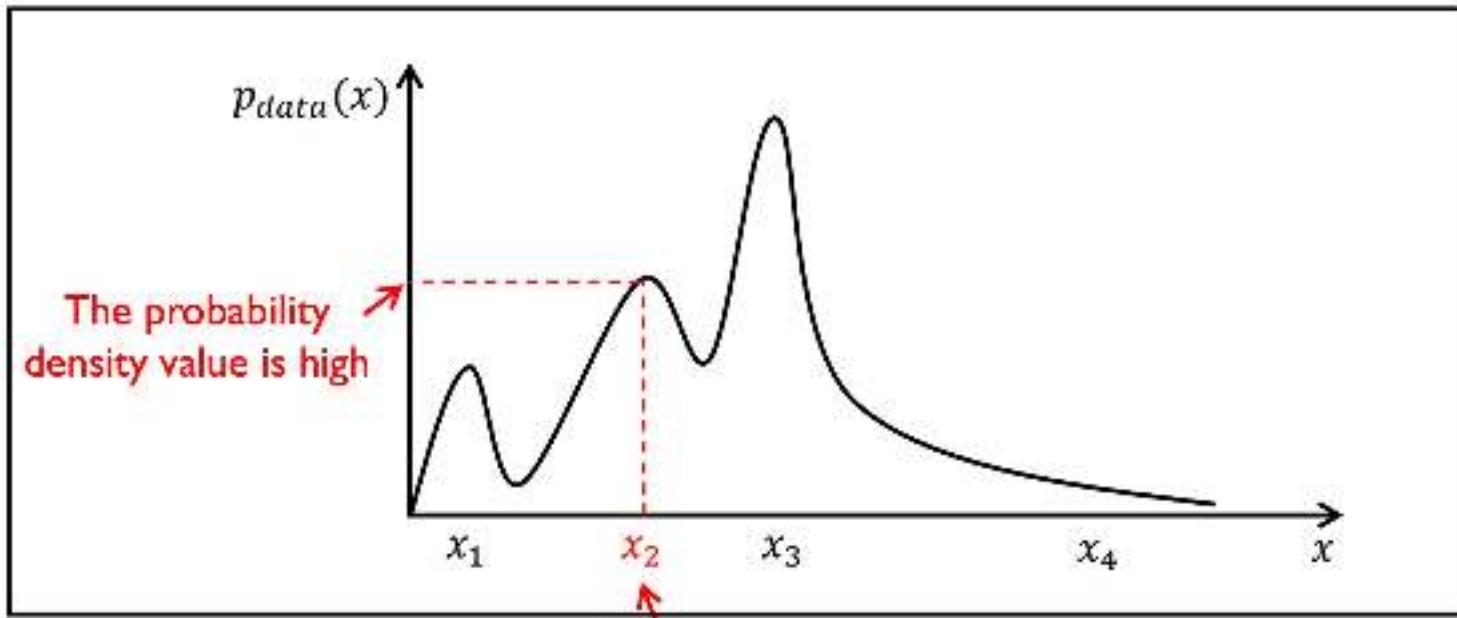


x_1 is a $64 \times 64 \times 3$ high dimensional vector
representing a man with glasses.

Source: Yunjey Choi, Korea University

Probability Distribution

Our dataset may contain many images of women with black hair.

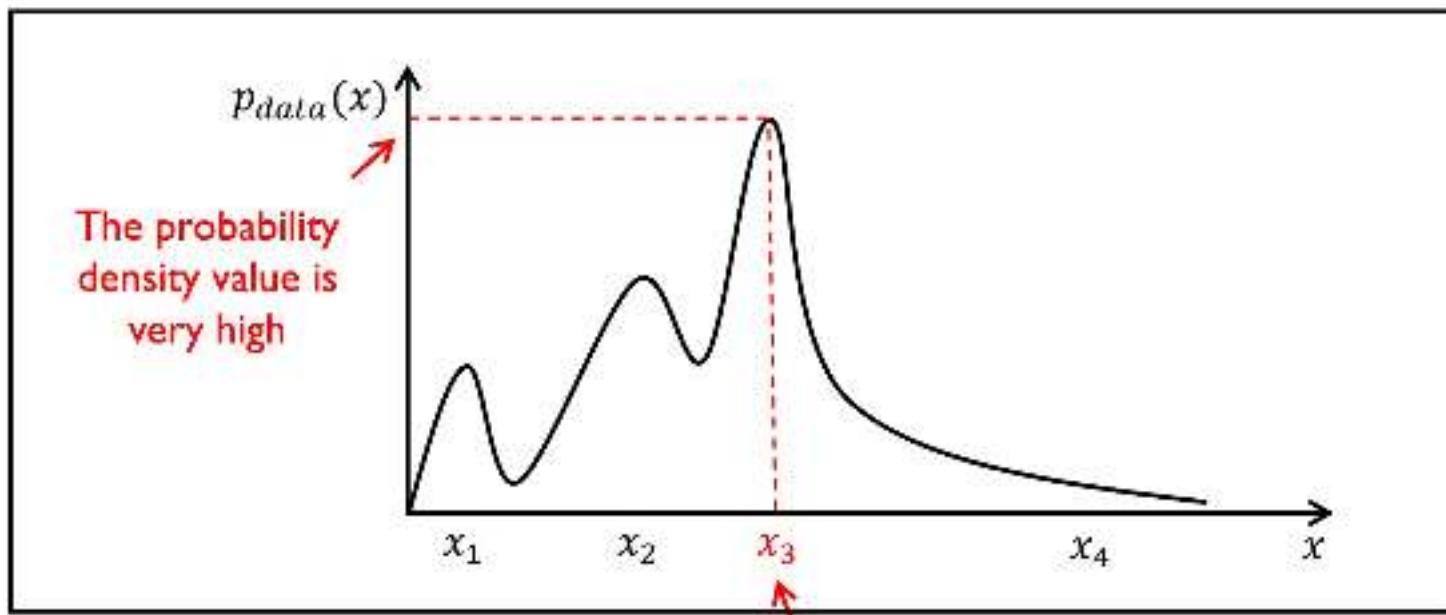


x_2 is a $64 \times 64 \times 3$ high dimensional vector
representing a woman with black hair.

Source: Yunjey Choi, Korea University

Probability Distribution

Our dataset may contain very many images of women with blonde hair.

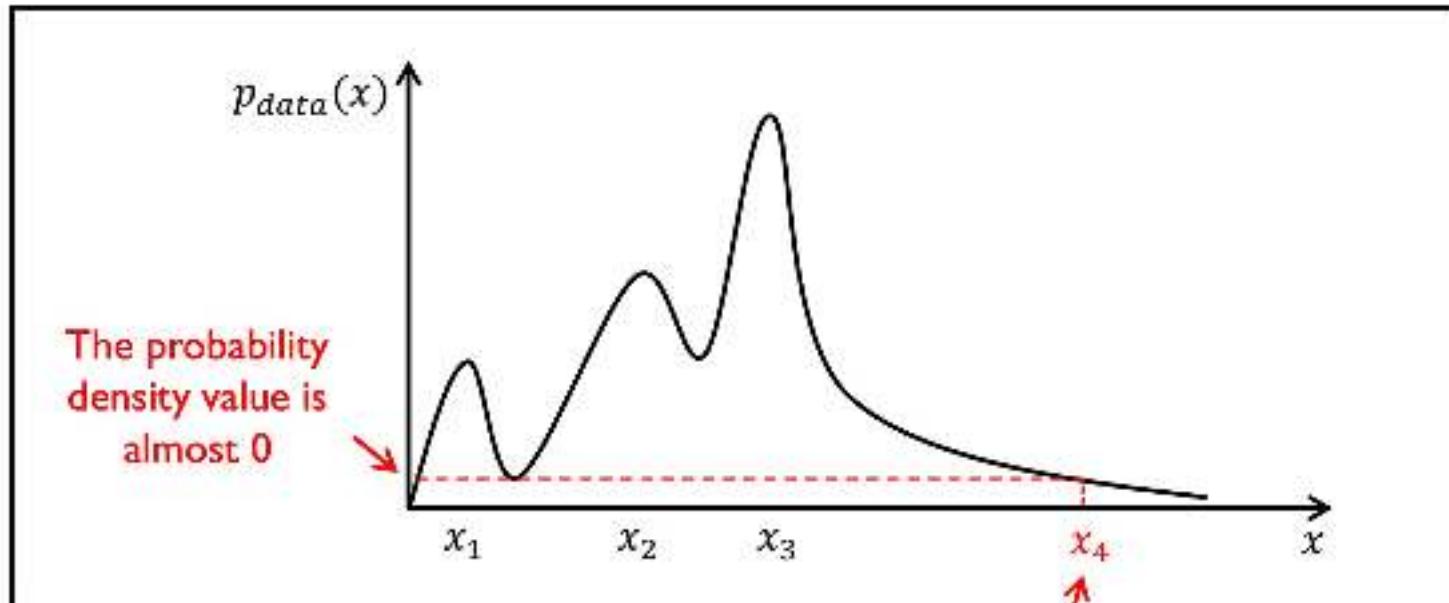


x_3 is a $64 \times 64 \times 3$ high dimensional vector representing a woman with blonde hair.

Source: Yunjey Choi, Korea University

Probability Distribution

Our dataset may not contain **these strange images**.



x_4 is an $64 \times 64 \times 3$ high dimensional vector representing **very strange images**.

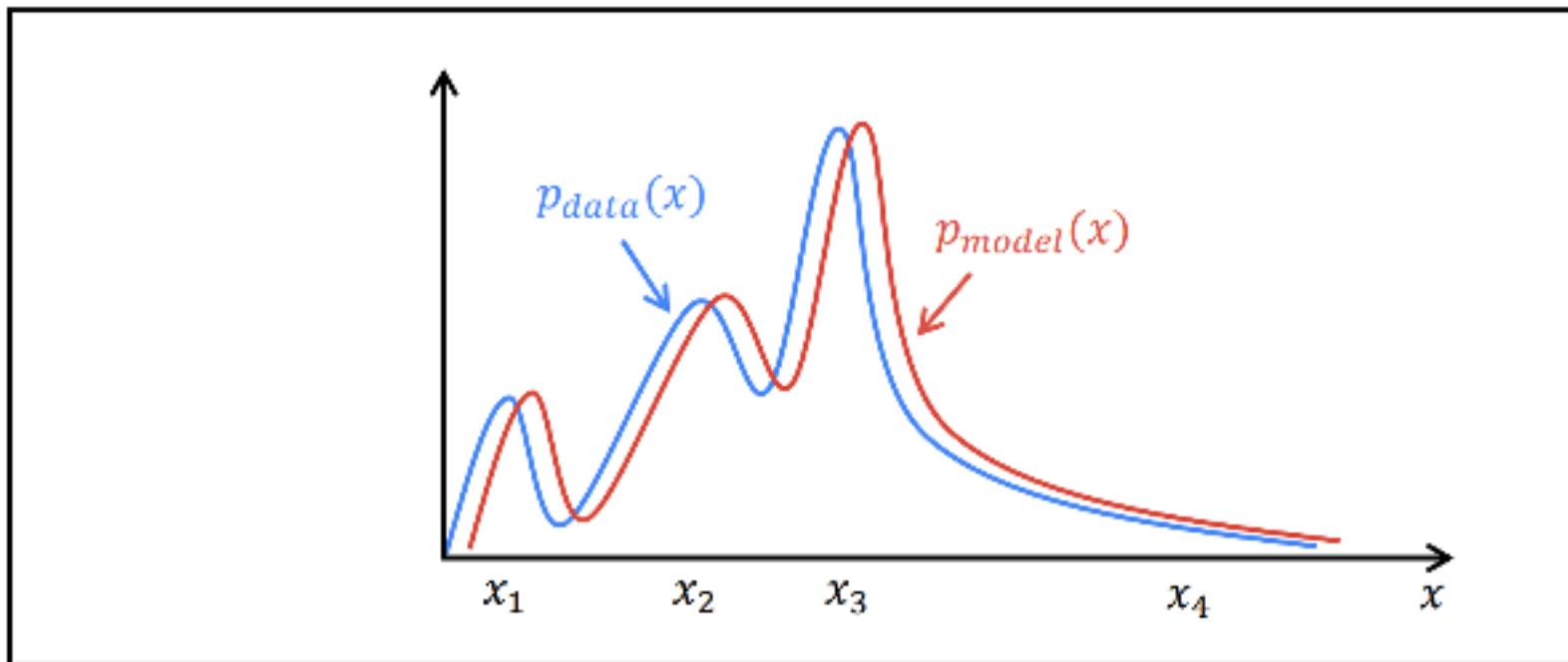
Source: Yunjey Choi, Korea University

Probability Distribution

The goal of the generative model is to find a $p_{model}(x)$ that approximates $p_{data}(x)$ well.

↗ Distribution of images generated by the model

↘ Distribution of actual images



GAN

- Generative
 - Learn a generative model
- Adversarial
 - Trained in an adversarial setting
- Networks
 - Use Deep Neural Networks

Generative Models

- We have **training samples** from an unknown distribution p_{data}
- We **want a model** that can **draw samples** from some distribution p_{model}
- p_{model} should be an estimate of p_{data}
- A model that can sample from this p_{model} is termed **a generative model**

Why Generative Models?

- Analyze our ability to represent and manipulate high-dimensional distributions (e.g. images)
- Can be used as a tool in reinforcement learning
- Can be used in semi-supervised learning where labelled data is scarce
- Sampling of realistic examples from some high-dimensional distribution can have many applications

Image Super Resolution

- Generating high-resolution images from **low-resolution inputs**
- GANs tend to produce perceptually **pleasing** and sharp results



Source: [Goodfellow, 2016]

<https://arxiv.org/abs/1701.00160>

Generative Adversarial Networks



[GAN, Goodfellow et al. 2014]

Generate Images



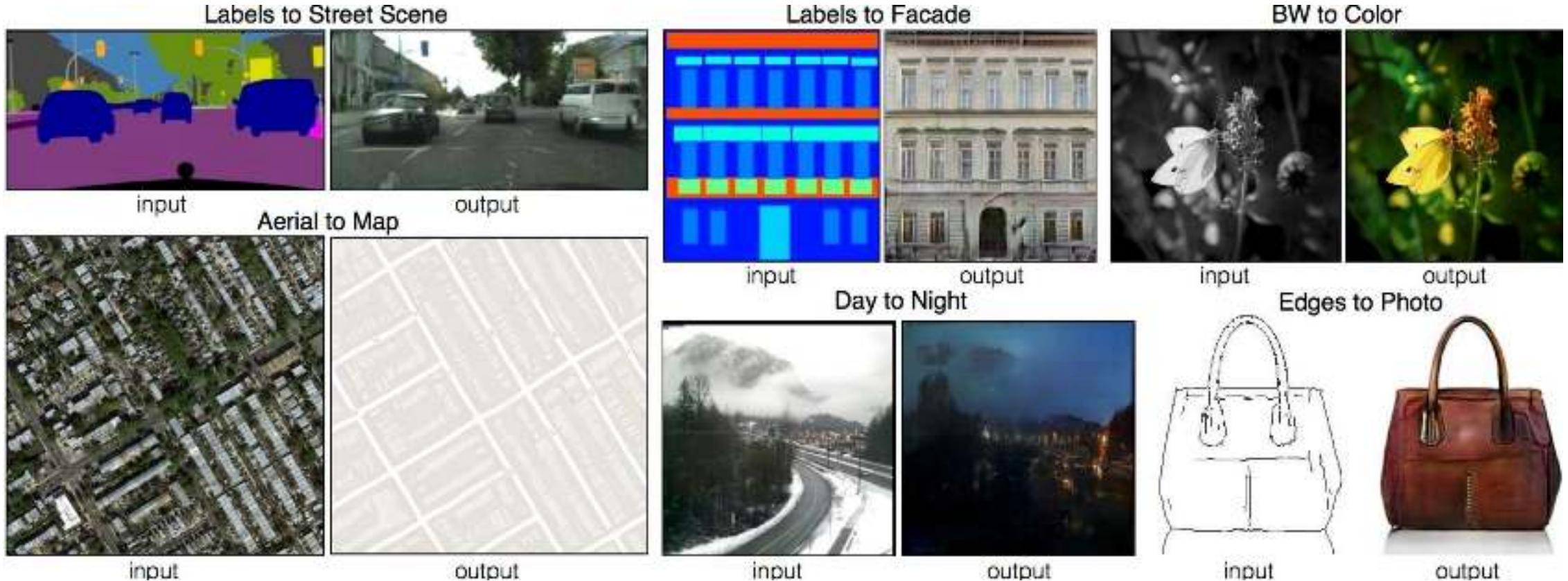
[DCGAN, Radford, Metz, Chintala 2015]

Generate Images



[CycleGAN: Zhu, Park, Isola & Efros, 2017]

Image-to-Image Translation (Conditional GAN)



<https://arxiv.org/pdf/1611.07004.pdf> (2018)

Generate Images



[StyleGAN, Karras, Laine, Aila, 2018]

Generative Adversarial Networks



- Paper Source: <https://arxiv.org/pdf/1809.11096.pdf> (ICLR 2019)
- Source Code: <https://tfhub.dev/s?q=biggan>

Virtual Try-On



VITON-HD (CVPR 2021)

GAN Introduction

Generative adversarial networks are deep neural net architectures comprised of [two nets](#), putting one against the other (thus the “adversarial”).

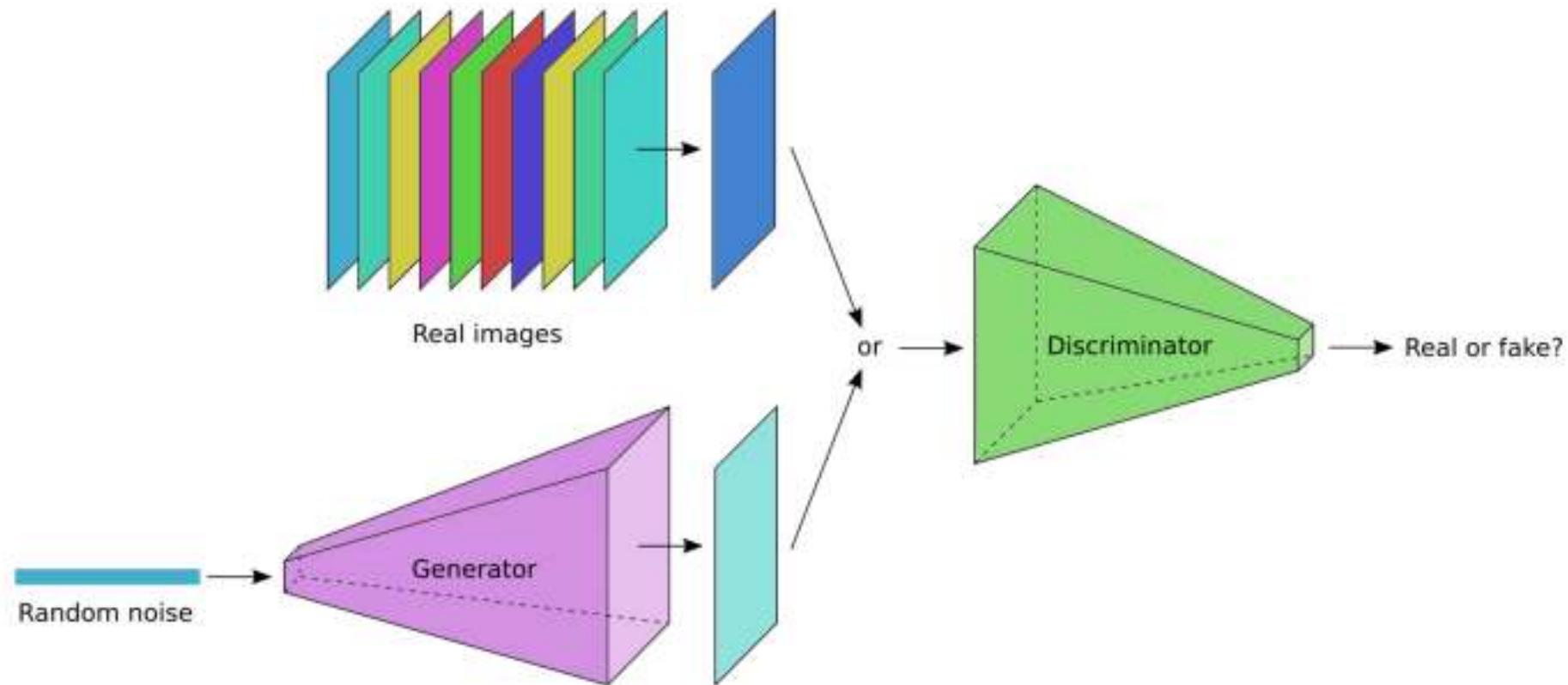


Ian Goodfellow
GAN – 2014

Facebook's AI research director Yann LeCun called adversarial training [“the most interesting idea in the last 10 years in ML.”](#)

GAN Components

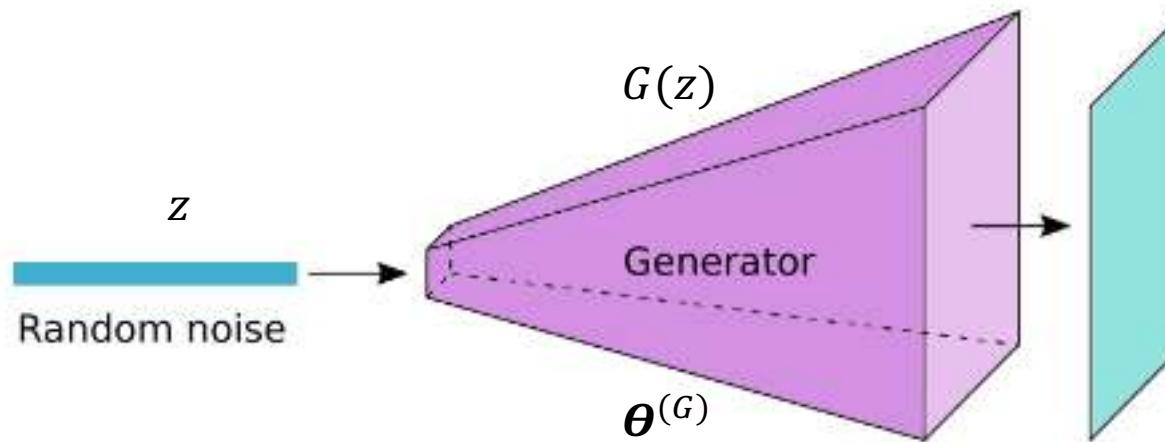
- A generator function that tries to create real-looking examples
- A discriminator function that tries to distinguish real from fake examples
- Functions are updated in a feedback loop, making each better at its task



Like two players in the game: a generator and a discriminator

GAN Components – Generator

- Lets G be the generator and a kind of generative neural network and differentiable function
- The input z is a random vector sampled from some simple prior distribution
- The generator takes as input $z \sim N(0, I)$ and produces $G(z) = x$

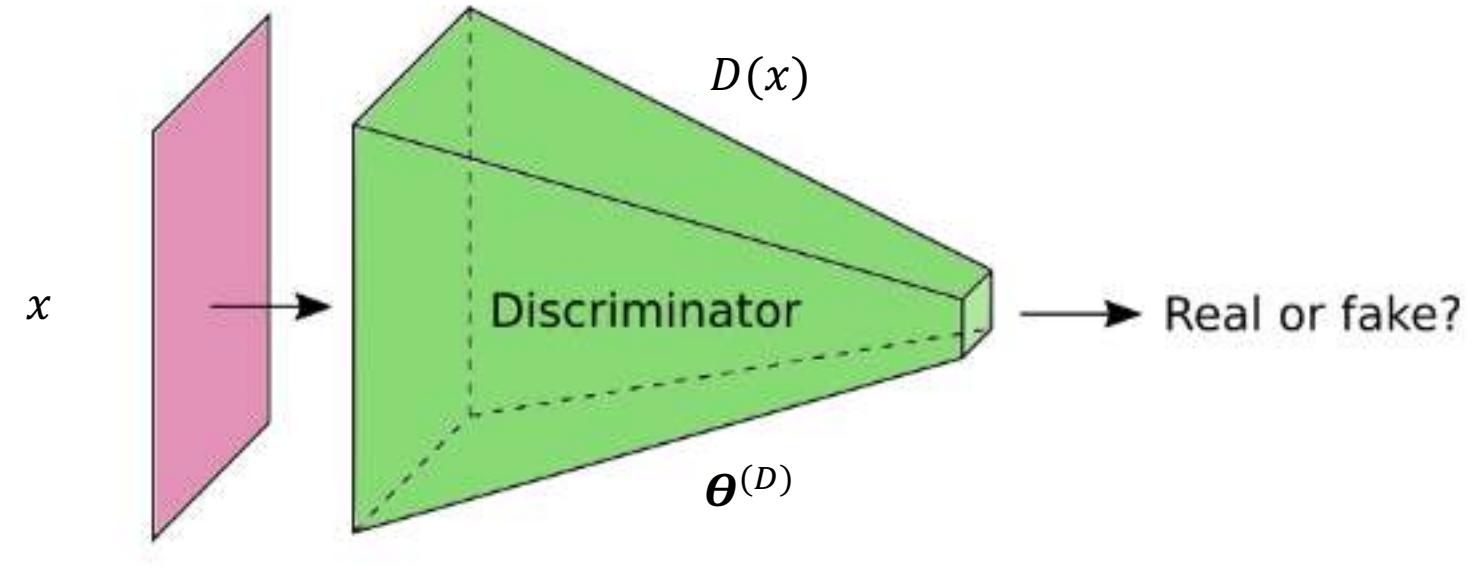


The generator is a function

$$G: z \mapsto G(z; \theta^{(G)})$$

GAN Components – Discriminator

- The discriminator is a standard classification network
- Trained to differentiate between real (x) and fake ($x = G(z)$) images
- Outputs a single number in $[0, 1]$
 - $D(x) = 0 \rightarrow D$ believes x is fake
 - $D(x) = 1 \rightarrow D$ believes x is real



The discriminator is a function

$$D: x \mapsto D(x; \theta^{(D)})$$

NASH Equilibrium

- Instead of treating this as an optimization problem, we treat this as a game between two players. The solution to a game is called a Nash equilibrium.
- For GANs, a Nash equilibrium is a tuple, $(\theta^{(D)}, \theta^{(G)})$ that is:
 - A local minimum of $\text{Loss}(D)$ with respect to $\theta^{(D)}$
 - A local minimum of $\text{Loss}(G)$ with respect to $\theta^{(G)}$
- If $G(z)$ and $D(x)$ have sufficient capacity then the Nash equilibrium corresponds to:
 - The generator draws samples and discriminator cannot discriminate between them, i.e., $D(x) = 1/2$ all x .

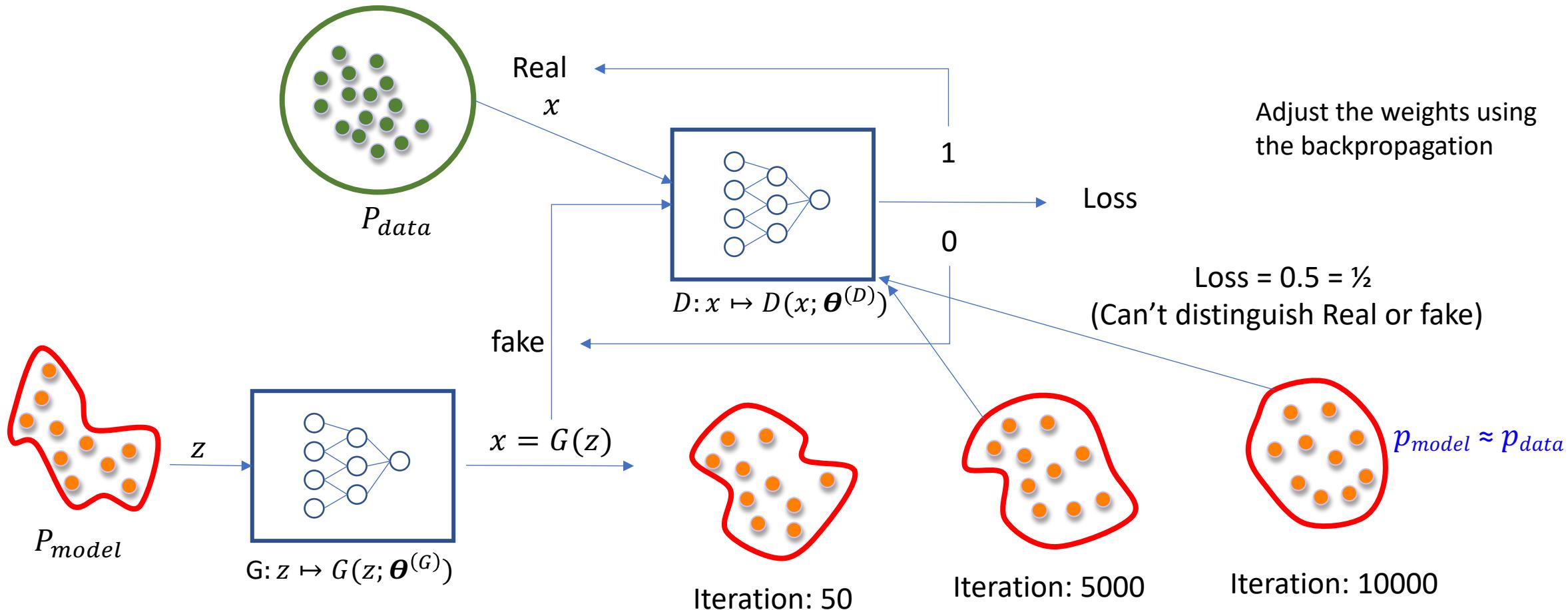
Loss Function of GAN

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

Loss Function of GAN

Discriminator Role: distinguish between the real and fake

Generator Role: Create data in such a way that it can fool the discriminator



Loss Function of GAN

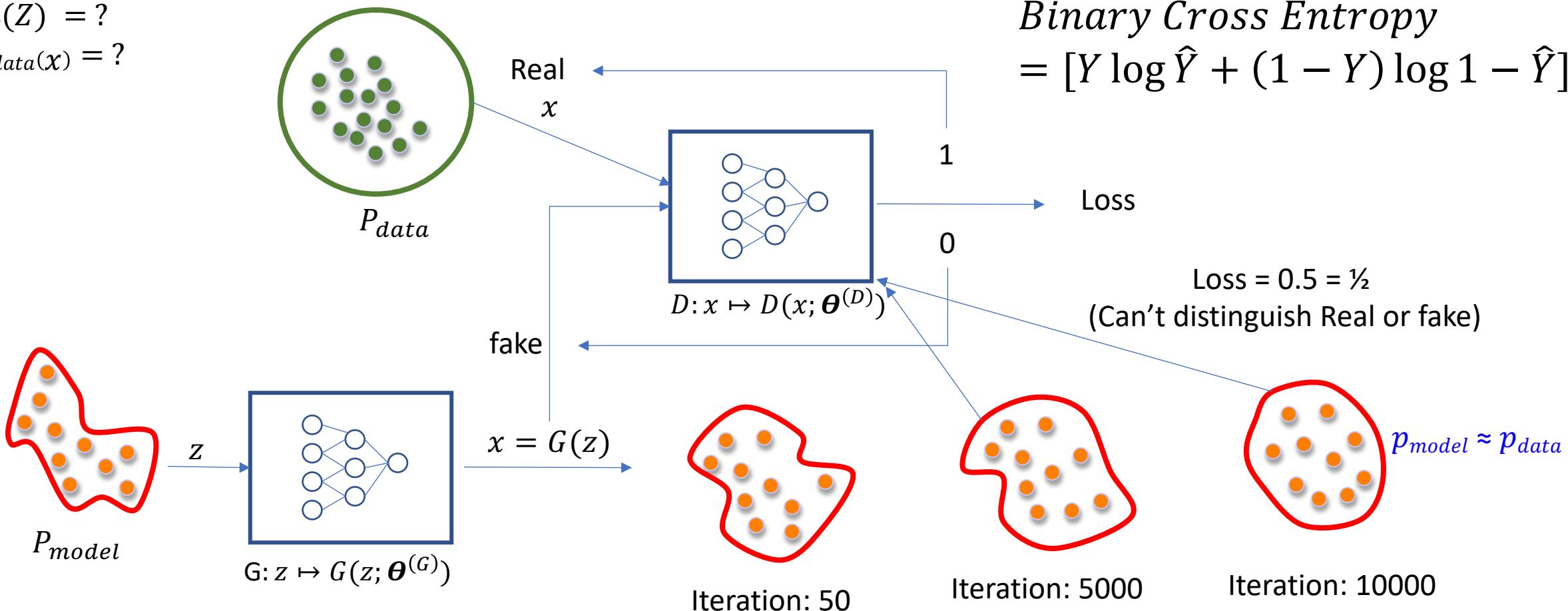
Point out the following over the diagram:

$$D(x) = ?$$

$$D(G(z)) = ?$$

$$p_z(Z) = ?$$

$$p_{data}(x) = ?$$



Cross-Entropy as a Loss Function



Animal	Label
Dog	[1 0 0 0 0]
Fox	[0 1 0 0 0]
Horse	[0 0 1 0 0]
Eagle	[0 0 0 1 0]
Squirrel	[0 0 0 0 1]

We can treat one hot encoding as a probability distribution for each image. Let's see a few examples.

$$\begin{array}{lll} P_1(\text{dog}) & = 1 & P_2(\text{dog}) & = 0 \\ P_1(\text{fox}) & = 0 & P_2(\text{fox}) & = 1 \\ P_1(\text{horse}) & = 0 & P_2(\text{horse}) & = 0 \\ P_1(\text{eagle}) & = 0 & P_2(\text{eagle}) & = 0 \\ P_1(\text{squirrel}) & = 0 & P_2(\text{squirrel}) & = 0 \end{array}$$

Machine Learning Model

$$\begin{aligned} Q_1 &= [0.4 \quad 0.3 \quad 0.05 \quad 0.05 \quad 0.2] \\ P_1 &= [1 \quad 0 \quad 0 \quad 0 \quad 0] \\ H(P_1, Q_1) &= - \sum_i P_1(i) \log Q_1(i) \\ &= -(1 \log 0.4 + 0 \log 0.3 + 0 \log 0.05 + 0 \log 0.05 + 0 \log 0.2) \\ &= -\log 0.4 \\ &\approx 0.916 \end{aligned}$$

$$\begin{aligned} Q_1 &= [0.98 \quad 0.01 \quad 0 \quad 0 \quad 0.01] \\ H(P_1, Q_1) &= - \sum_i P_1(i) \log Q_1(i) \\ &= -(1 \log 0.98 + 0 \log 0.01 + 0 \log 0 + 0 \log 0 + 0 \log 0.01) \\ &= -\log 0.98 \\ &\approx 0.02 \end{aligned}$$

The cross-entropy goes down as the prediction gets more and more accurate. It becomes zero if the prediction is perfect.

Source: <https://towardsdatascience.com/demystifying-cross-entropy-e80e3ad54a8>

Binary Cross-Entropy

- In binary classification: we have two classes only.
 - For example: there are only **dogs** or **cats** in images.
- Thus cross-entropy formula contains only two probabilities:

$$\begin{aligned} H(P, Q) &= - \sum_{i=(\text{cat}, \text{dog})} P(i) \log Q(i) \\ &= -P(\text{cat}) \log Q(\text{cat}) - P(\text{dog}) \log Q(\text{dog}) \end{aligned}$$

- Using the following relationship

$$P(\text{dog}) = (1 - P(\text{cat}))$$

$$H(P, Q) = -P(\text{cat}) \log Q(\text{cat}) - (1 - P(\text{cat})) \log(1 - Q(\text{cat}))$$

Further simplify

$$\text{BinaryCrossEntropy} = -P \log \hat{P} - (1 - P) \log(1 - \hat{P})$$

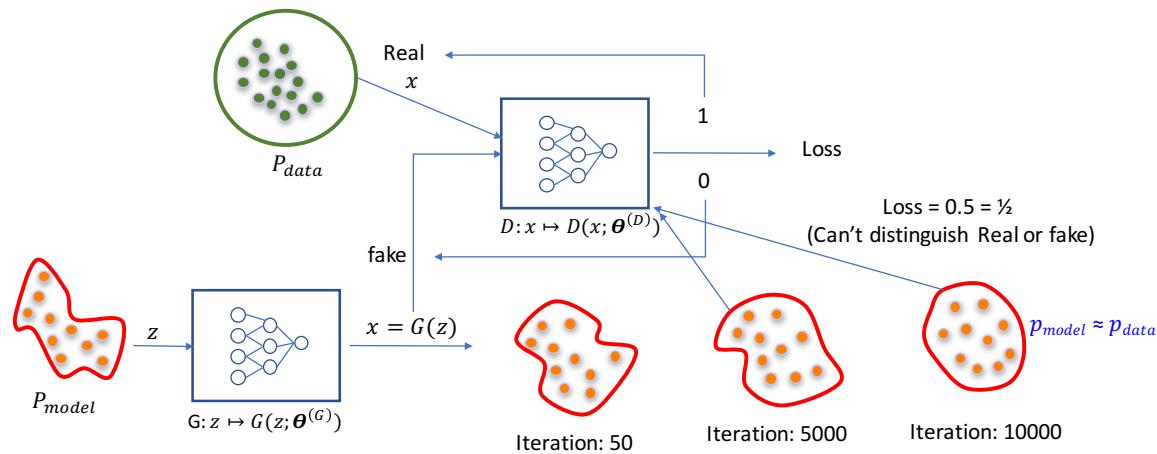
$$P = P(\text{cat})$$

$$\hat{P} = Q(\text{cat})$$

Loss Function of GAN

$$L(Y, \hat{Y}) = [Y \log \hat{Y} + (1 - Y) \log(1 - \hat{Y})]$$

$Y = \text{original}$
 $\hat{Y} = \text{reconstructed}$



The label for the data coming from $P_{data}(x)$ is $y = 1$ and $\hat{y} = D(x)$, so we can get

$$L(1, D(x)) = [\log D(x)]$$

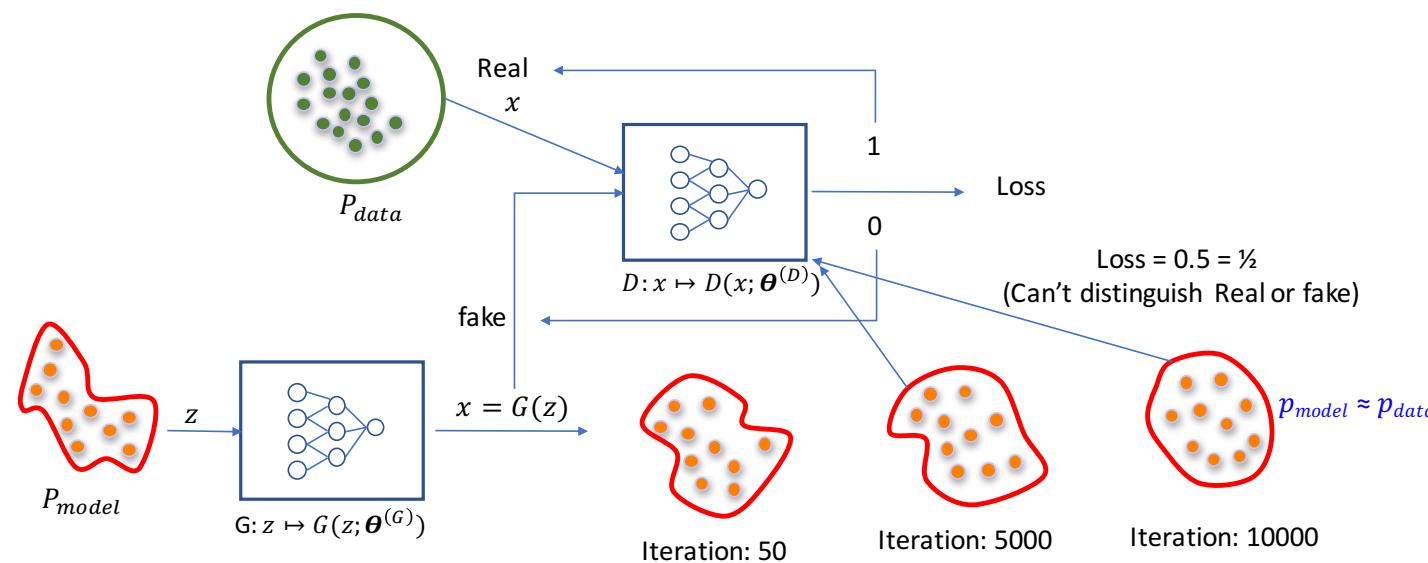
The label for the data coming from $P_{model}(z)$ is $y = 0$ and $\hat{y} = D(G(z))$, so we can get

$$L(0, D(G(z))) = [\log(1 - D(G(z)))]$$

Loss Function of GAN

- Objective of discriminator is to correctly classify the fake vs real. So we need to maximize the previous both cases

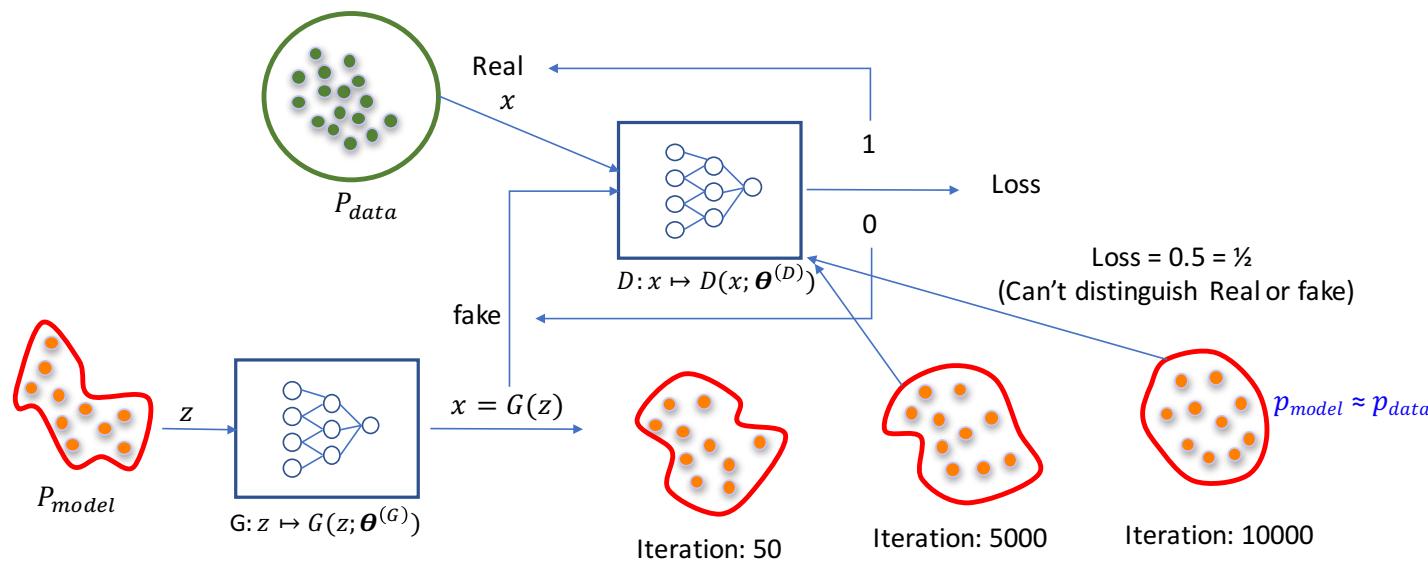
$$\max \quad [\log D(x)] + [\log(1 - D(G(z)))] = D$$



Loss Function of GAN

- In case of generator, we want to maximize D value

$$\max D(G(z)) = 1 \quad \rightarrow \quad \min \log(1 - D(G(z))) = G$$



Loss Function of GAN

$$\max [\log D(x) + \log 1 - D(G(z))] = D$$

$$\min [\log(1 - D(G(z)))] = G$$

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

GAN Training

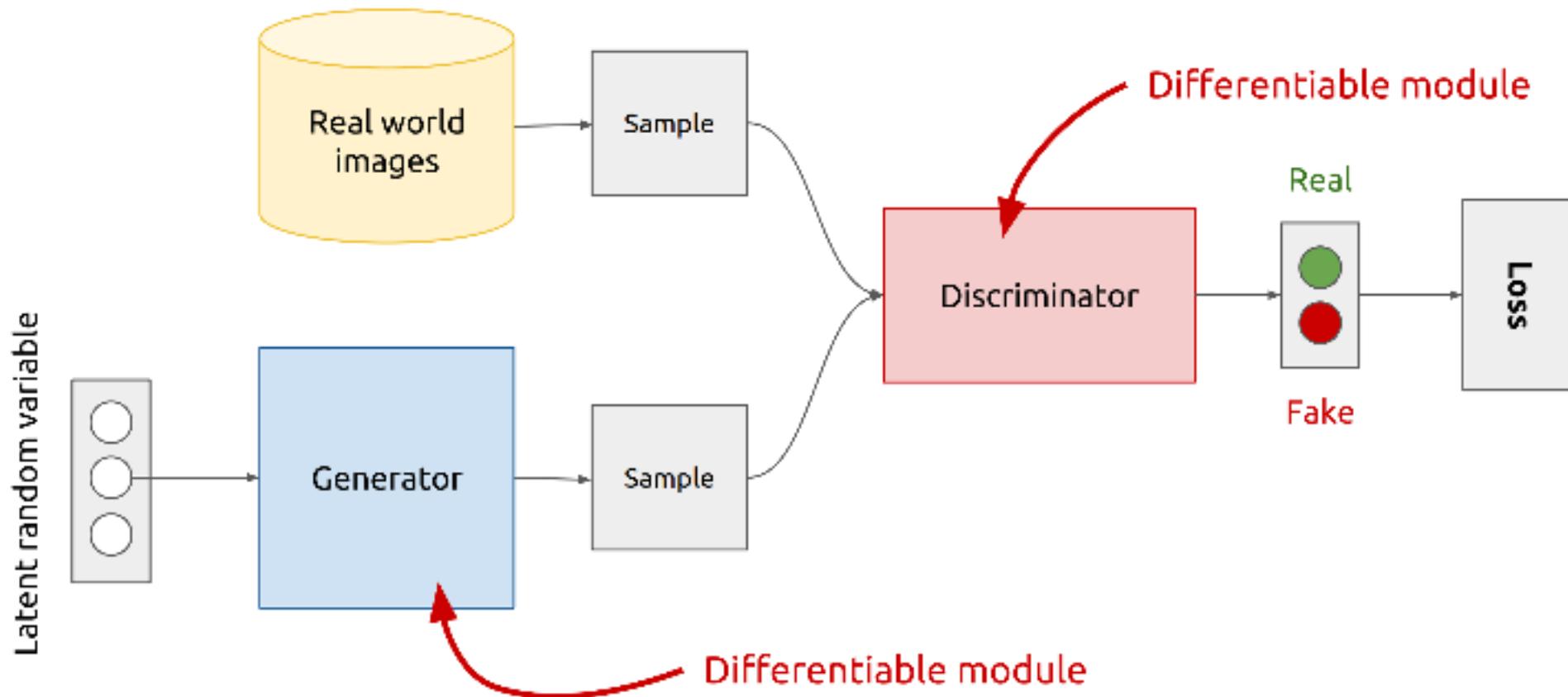
- Because a GAN contains two separately trained networks, its training algorithm must address two complications:
 - GANs must juggle two different kinds of training (generator and discriminator).
 - GAN convergence is hard to identify.

Alternating Training

- The generator and the discriminator have different training processes.
So how do we train the GAN as a whole?
- GAN training proceeds in alternating periods:
 1. The discriminator trains for one or more epochs.
 2. The generator trains for one or more epochs.
 3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

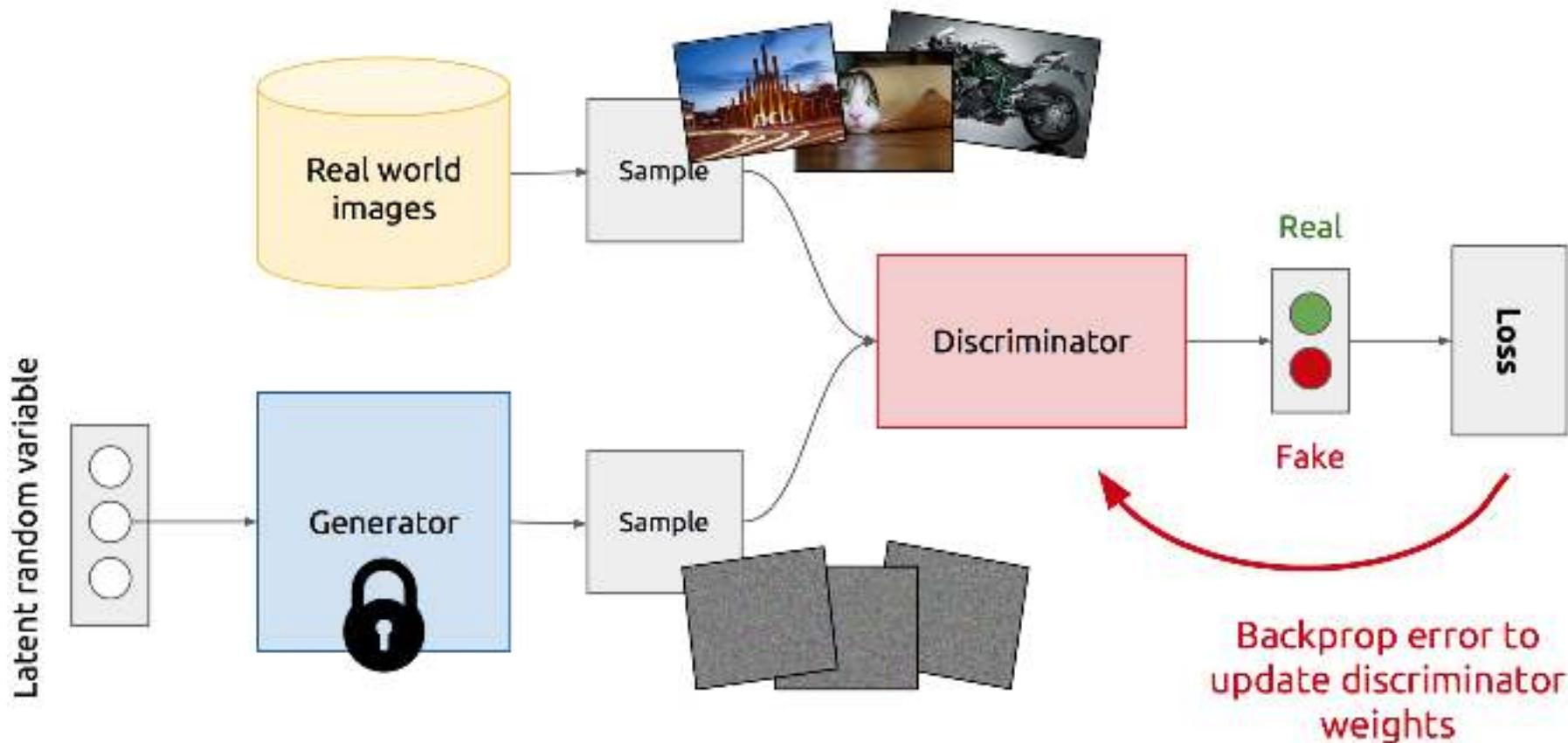
Training GANs

Alternate between training the discriminator and generator



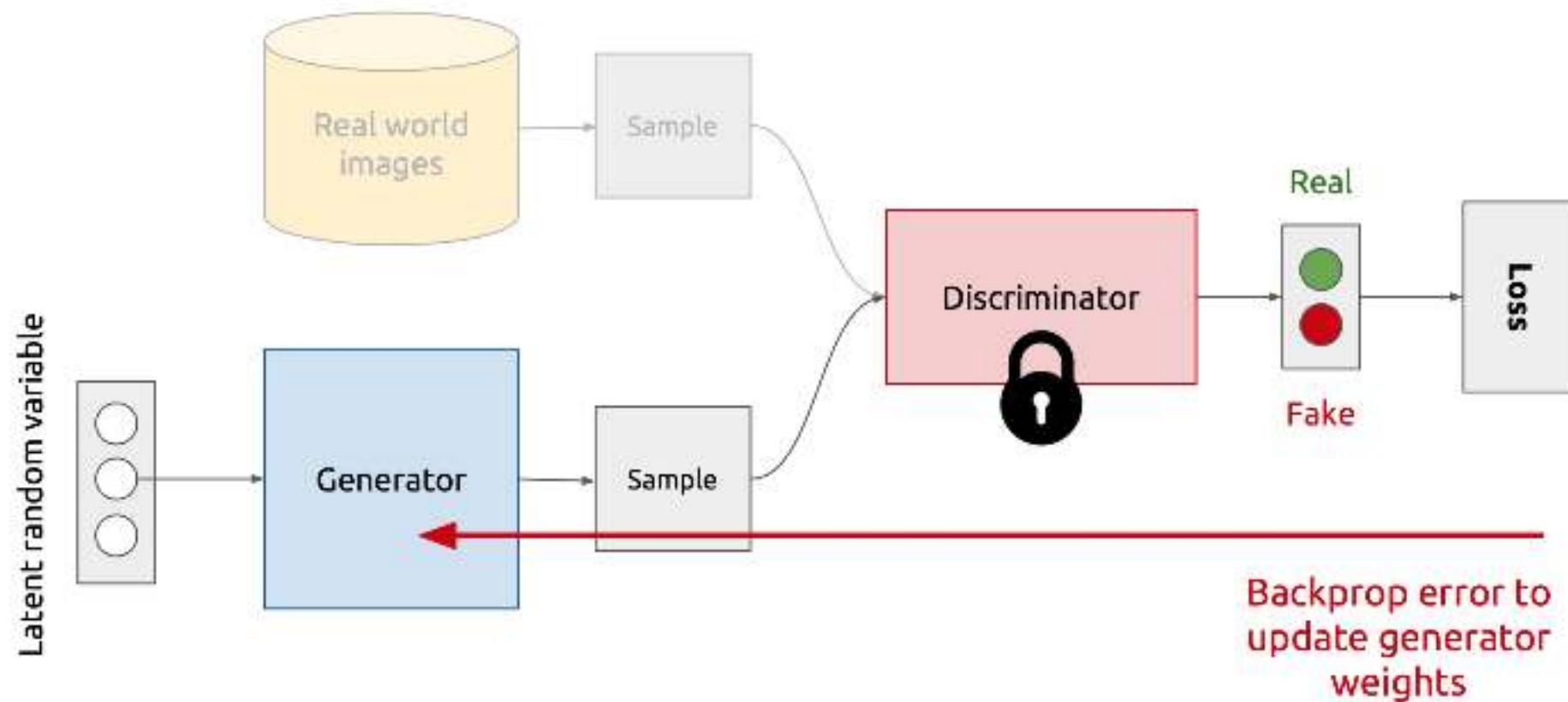
Training Discriminator

1. Fix generator weights, draw samples from both real world and generated images
2. Train discriminator to distinguish between real world and generated images



Training Generator

1. Fix discriminator weights
2. Sample from generator
3. Backprop error through discriminator to update generator weights



Training GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Discriminator
Updates

Generator
Updates

In practice, it does
not work well and
we use a slightly
modified objective

Modification in Generator Updates (Final)

procedure GAN TRAINING

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{data}(\mathbf{x})$
- Update the discriminator by ascnding its stochastic gradient:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m \left[\log D_{\theta} \left(x^{(i)} \right) + \log \left(1 - D_{\theta} \left(G_{\phi} \left(z^{(i)} \right) \right) \right) \right]$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$
- Update the generator by ascending its stochastic gradient

$$\nabla_{\phi} \frac{1}{m} \sum_{i=1}^m \left[\log \left(D_{\theta} \left(G_{\phi} \left(z^{(i)} \right) \right) \right) \right]$$

end for

end procedure

Adversarial Nets

- Adversarial nets have the advantages that **Markov chains** are never needed
- Only **backpropagation** is used to obtain gradients
- **No inference is required** during learning, and a wide variety of factors and interactions can easily be incorporated into the model.

Training Problems

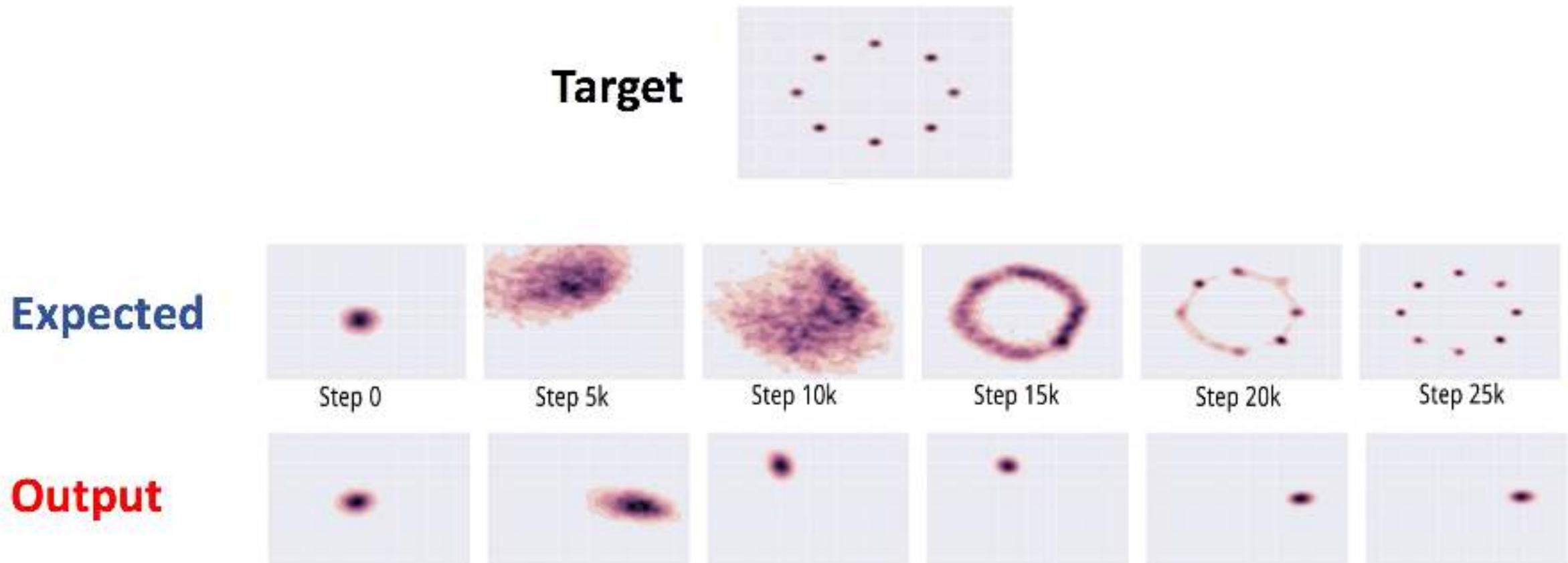
- Non-Convergence
- Mode-Collapse

Non-Convergence

- Deep Learning models (in general) involve a single player
 - The player tries to maximize its reward (minimize its loss).
 - Use SGD (with Backpropagation) to find the optimal parameters.
 - SGD has convergence guarantees (under certain conditions).
 - **Problem:** With non-convexity, we might converge to local optima.
- GANs instead involve two (or more) players
 - Discriminator is trying to maximize its reward.
 - Generator is trying to minimize Discriminator's reward.
- SGD was not designed to find the Nash equilibrium of a game.
- **Problem:** We might not converge to the Nash equilibrium at all.

Mode-Collapse

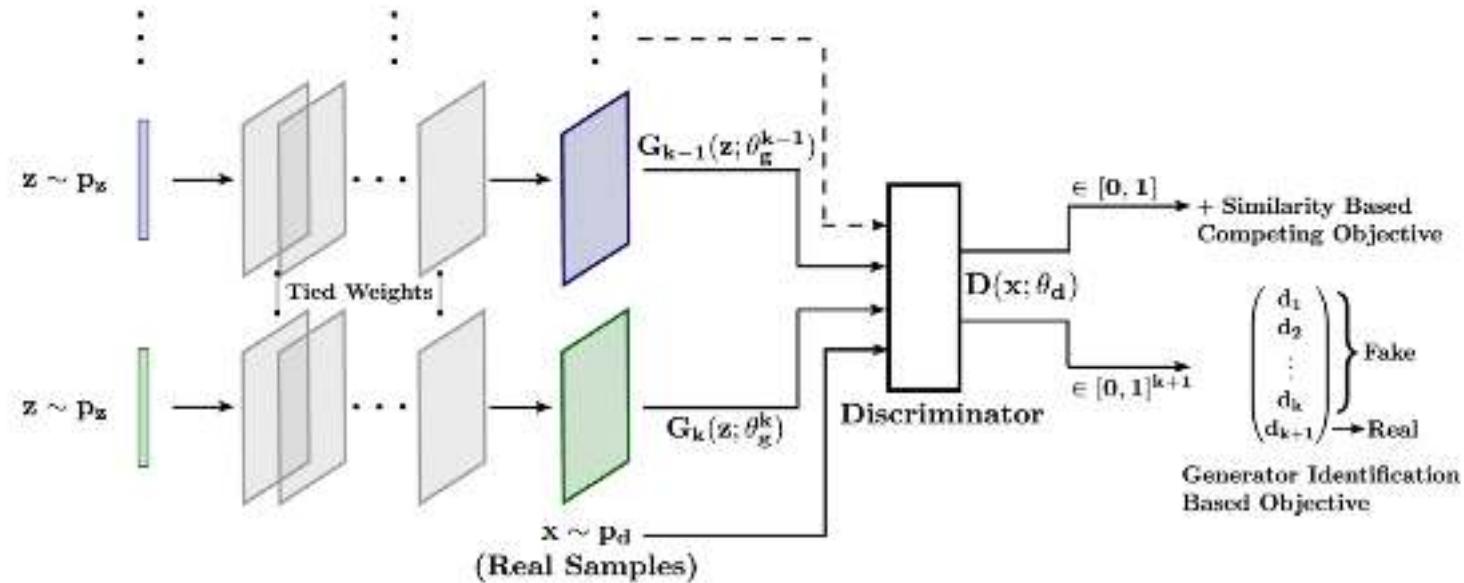
- Generator fails to output diverse samples



Metz, Luke, et al. "Unrolled Generative Adversarial Networks." arXiv preprint arXiv:1611.02163 (2016)

Mode-Collapse

- GANs often seem to collapse to far fewer modes than the model can represent



- multiple parallel generators
- share parameters up to layer
- applies a diversity loss on different generators
- Alternatively, have D predict which generator the fake sample came from!

Some Solutions – Implementation Tips

- Scale the image pixel value between -1 and 1.
- Use `tanh` as the output layer for the generator.
- Experiment sampling `z` with Gaussian distributions.
- Batch normalization often stabilizes training.
- Use PixelShuffle and transpose convolution for upsampling.
- Avoid max pooling for downsampling.
- Use convolution stride.
- Adam optimizer usually works better than other methods.
- Add noise to the real and generated images before feeding them into the discriminator.

Remarks!!

- They currently generate the **sharpest images**
- They are **easy to train** (since no statistical inference is required), and only back-propagation is needed to obtain gradients
- GANs are **difficult to optimize** due to unstable training dynamics.

Tips and Tricks to Train GAN

- More Information: <https://github.com/soumith/ganhacks>

Active Area of Research

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

Why GAN?

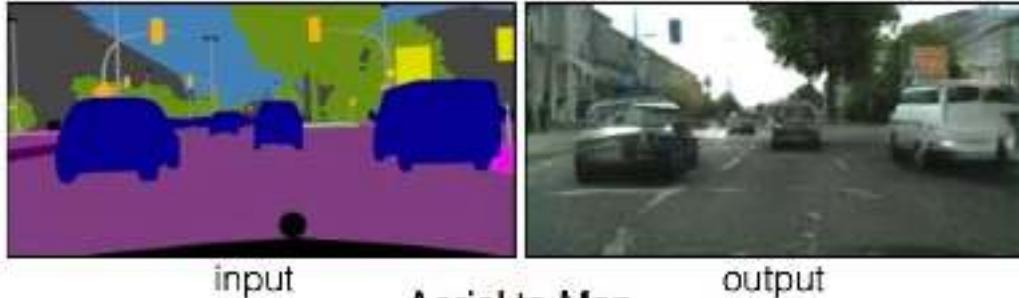
- The GAN is nice for many reasons, some of which are listed here:
 - The design of the generator has very few restrictions. We could incorporate a CNN easily, or an LSTM, etc.
 - The GAN doesn't require a variational lower bound.
 - GANs are asymptotically consistent.
 - GANs are subjectively better at generating samples.
- Sampling (or generation) is straightforward.
- Training doesn't involve Maximum Likelihood estimation.
- Robust to Overfitting since Generator never sees the training data.
- Empirically, GANs are good at capturing the modes of the distribution.

Applications

- Image-to-Image Translation
- Text-to-Image Synthesis
- Face Aging many more...

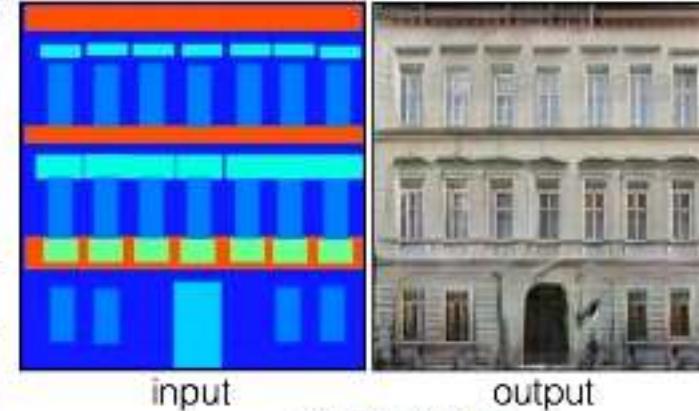
Image-to-Image Translation

Labels to Street Scene



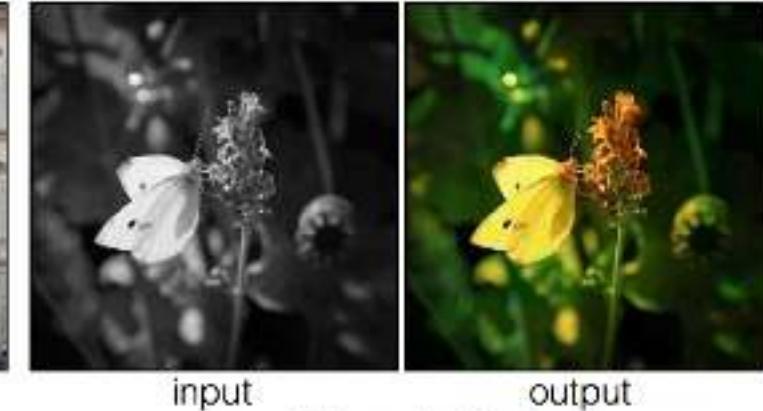
input

Labels to Facade



input

BW to Color



input

output

Aerial to Map



input

output

Day to Night



input

output

Edges to Photo



input

output

Source: <https://arxiv.org/abs/1611.07004>

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "Image-to-image translation with conditional adversarial networks". arXiv preprint arXiv:1611.07004. (2016).

Text-to-Image Synthesis

this small bird has a pink breast and crown, and black primaries and secondaries.



this magnificent fellow is almost all black with a red crest, and white cheek patch.



the flower has petals that are bright pinkish purple with white stigma

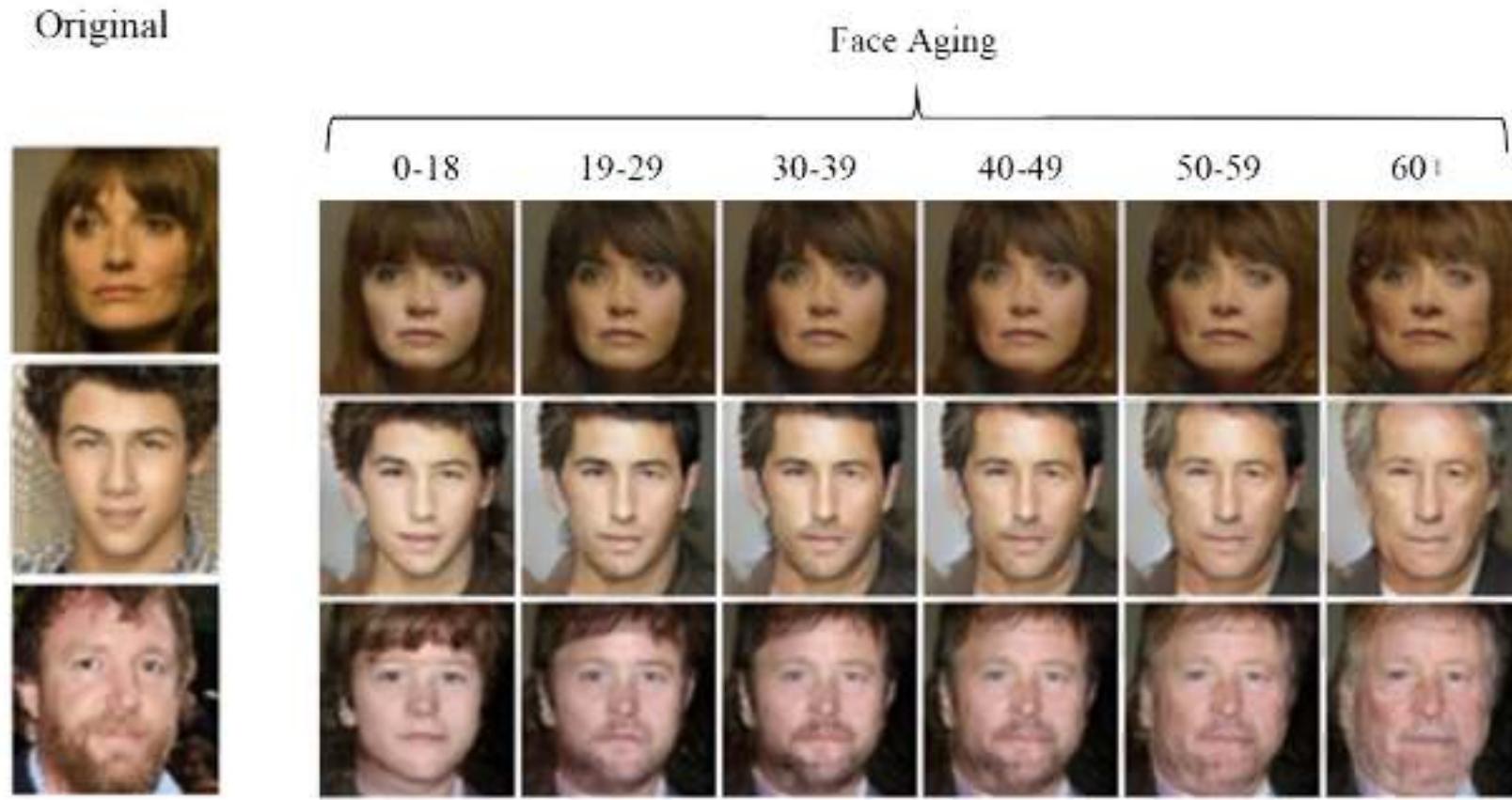


this white and yellow flower have thin white petals and a round yellow stamen



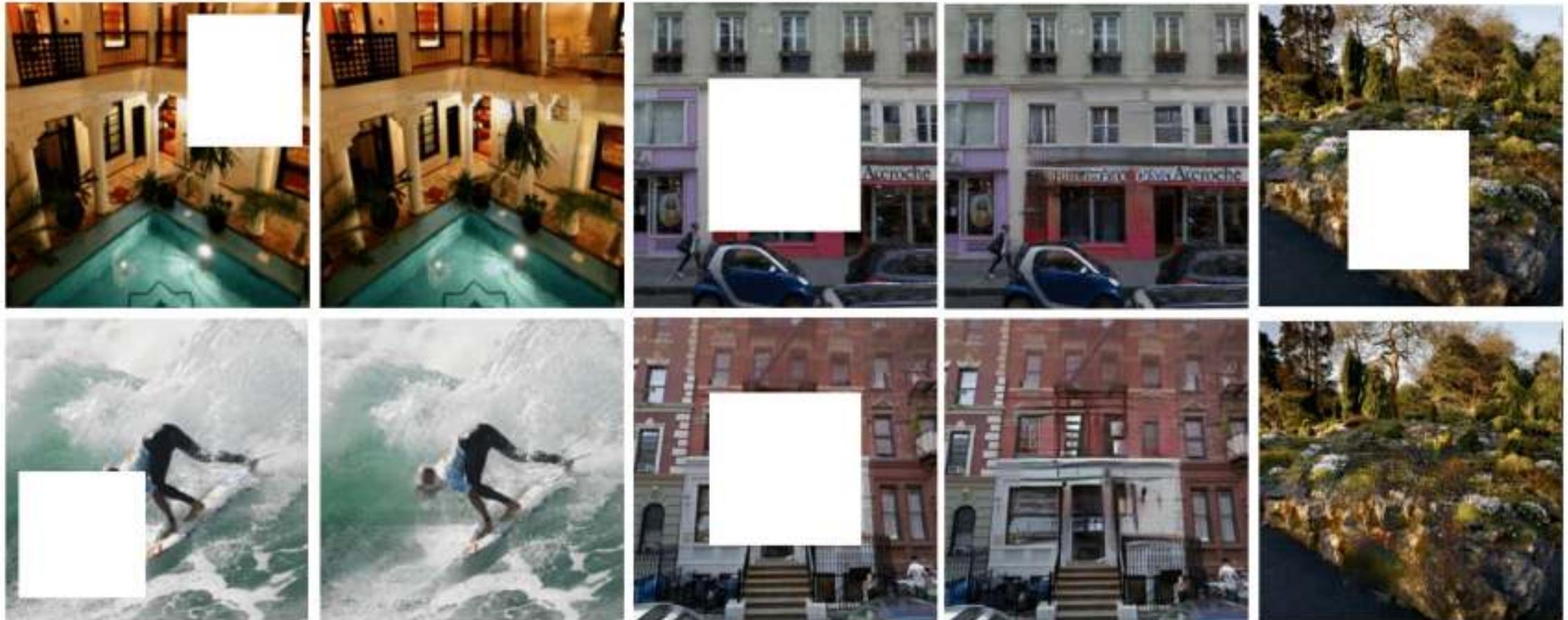
Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. "Generative adversarial text to image synthesis". ICML (2016).

Face Aging



Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "Face Aging With Conditional Generative Adversarial Networks". arXiv preprint arXiv:1702.01983.

Image Inpainting



Ugur Demir and Gozde Unal. Patch-Based Image Inpainting with Generative Adversarial Networks. *arXiv preprint*, 2018. URL <http://arxiv.org/abs/1803.07422>.

Create Art



Ahmed Elgammal, Bingchen Liu, Mohamed Elhoseiny, and Marian Mazzone. CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms. *arXiv preprint*, (Iccv):1–22, 2017. doi: 10.1089/cyber.2017.29084.csi. URL <http://arxiv.org/abs/1706.07068>.

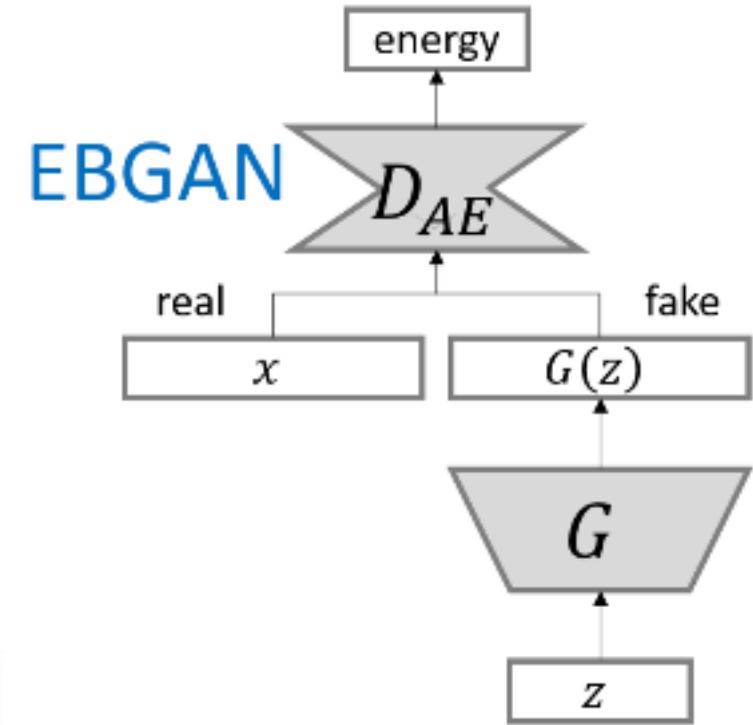
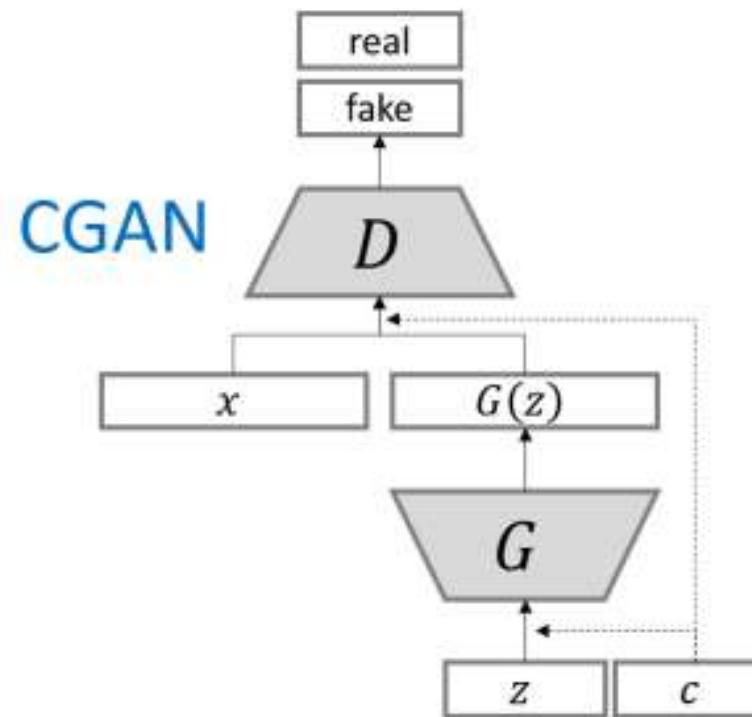
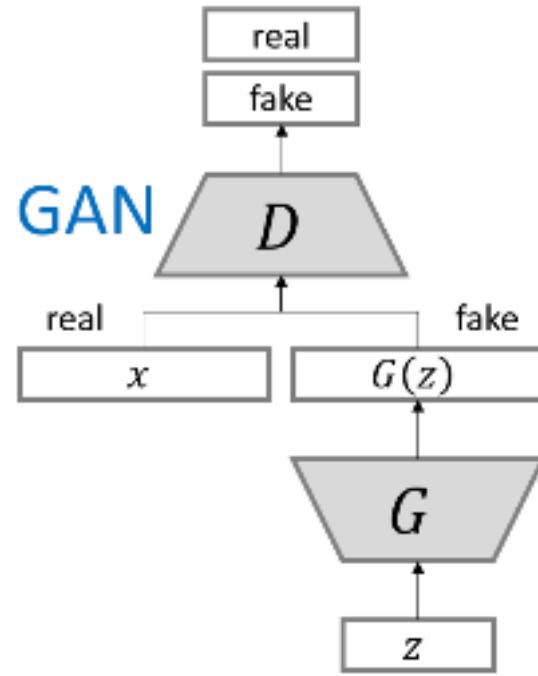
Reading

- Goodfellow, Ian. "NIPS 2016 Tutorial: Generative Adversarial Networks." arXiv preprint arXiv:1701.00160 (2016).

Thank You 😊

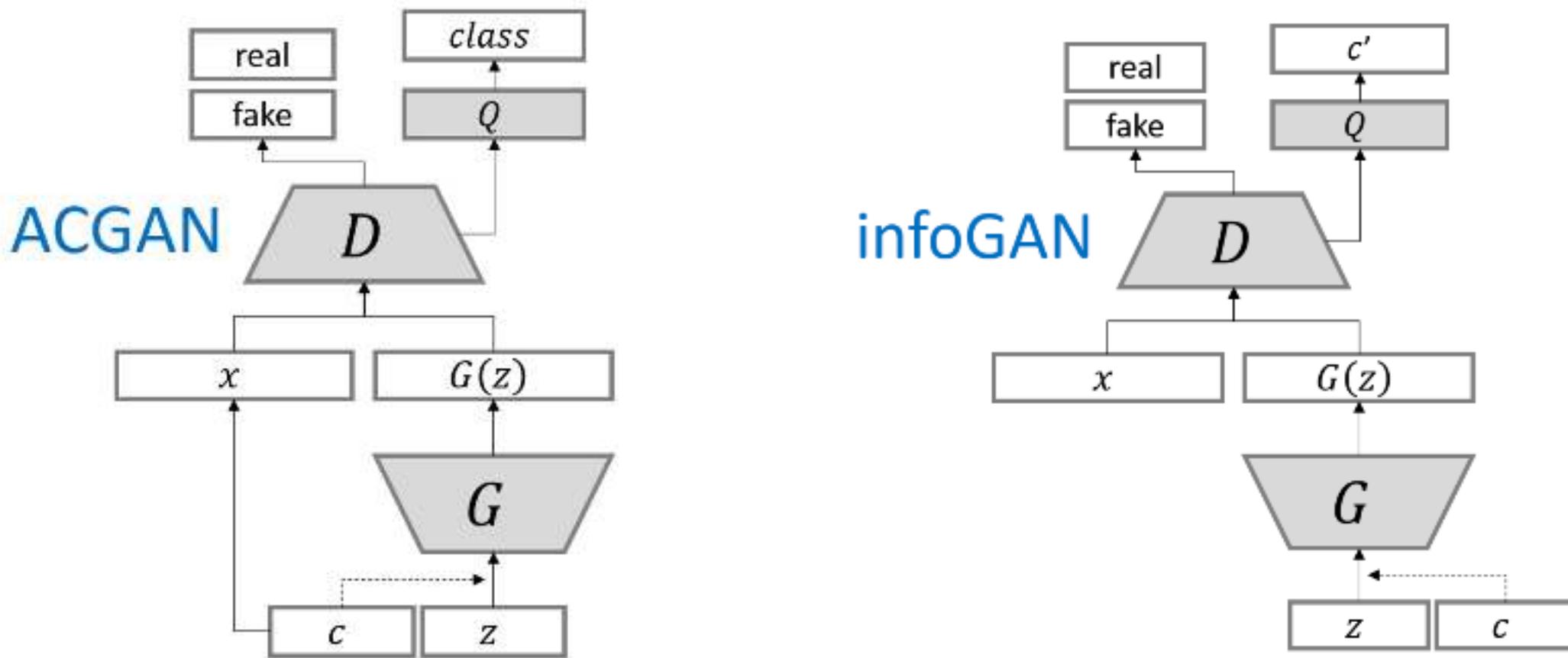
Appendix

Variants of GAN Structure



Source: <https://github.com/hwalsuklee/tensorflow-generative-model-collections>

Variants of GAN Structure



Source: <https://github.com/hwalsuklee/tensorflow-generative-model-collections>

Conditional GAN (cGAN)

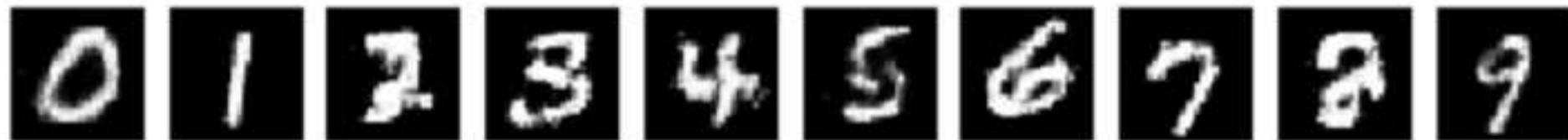
- In an unconditioned generative model, there is no control on modes of the data being generated.
- GANs can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y .
- y could be any kind of auxiliary information, such as class labels or data from other modalities.
- We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer.

Conditional GAN (cGAN)

Conditional Loss Function

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|\mathbf{y})))]$$

- MNIST Example
 - To be able to tell our generator to generate any digit we want, only a very small change in training is needed.
 - For each iteration, the generator takes as input not only z but also a one-hot encoded vector indicating the digit.
 - The discriminator input consists then of not only the real or fake sample but also the same label vector.



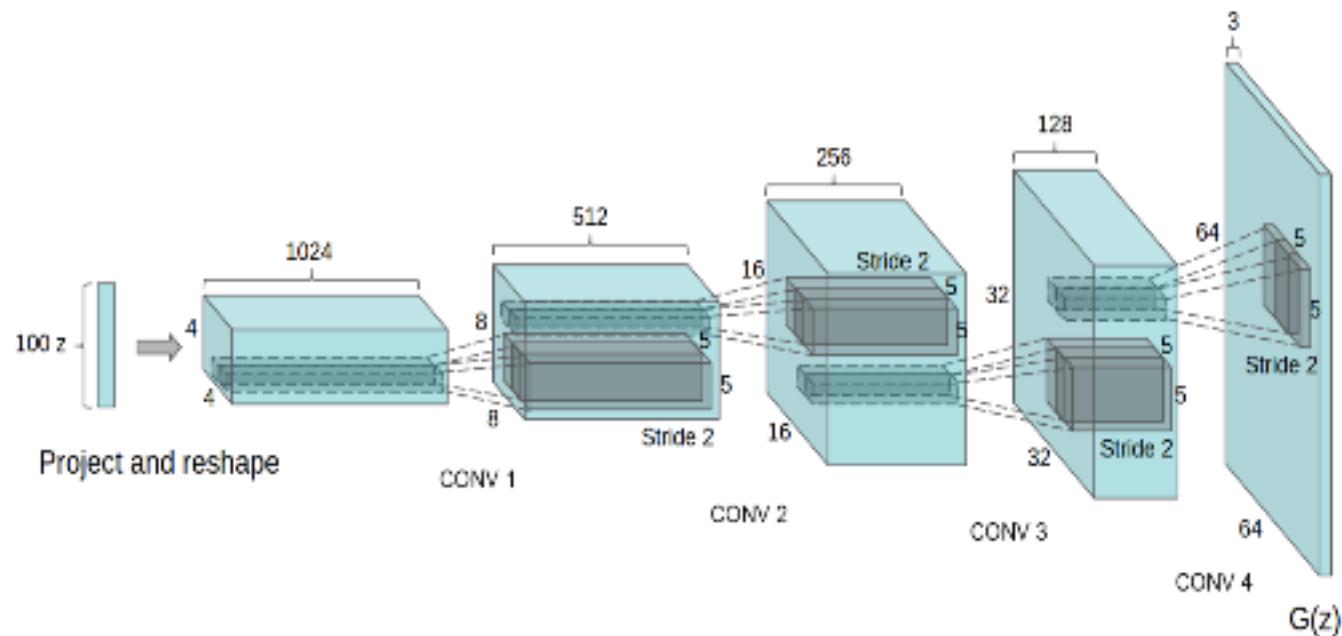
Conditionally generated digit samples

Conditional GAN (Text to Image)

Caption	Image
a pitcher is about to throw the ball to the batter	
a group of people on skis stand in the snow	
a man in a wet suit riding a surfboard on a wave	

Deep Convolutional GANs (DCGANs)

Generator Architecture



Key ideas:

- Replace FC hidden layers with Convolutions
 - **Generator:** Fractional-Strided convolutions
- Use Batch Normalization after each layer
- **Inside Generator**
 - Use ReLU for hidden layers
 - Use Tanh for the output layer

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).

Face Detection and Recognition

Instructor: Dr. Muhammad Fahim

Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - Found material over the internet to aligned the subject according to the need of the students.

Contents

- Computer vision technologies for human faces
- Face detection and its evaluation
 - Viola Jones Algorithm
- Face Recognition
 - Eigenfaces

Computer vision technologies that involve human faces

- Face Detection
- Face Recognition
- Facial Analysis
- Facial Tracking

Today our focus is:
Face Recognition and Detection

Face Detection

- Available in most digital cameras nowadays
- **Challenges**
 - A picture has 0,1 or many faces.
 - Faces are not the same: with spectacles, mustache etc.
 - Sizes of faces vary a lot.
- **The simple method**
 - [Slide a window](#) across the image and detect faces.
 - Too slow, pictures have too many pixels. ($1280 \times 1024 = 1.3M$ pixels)

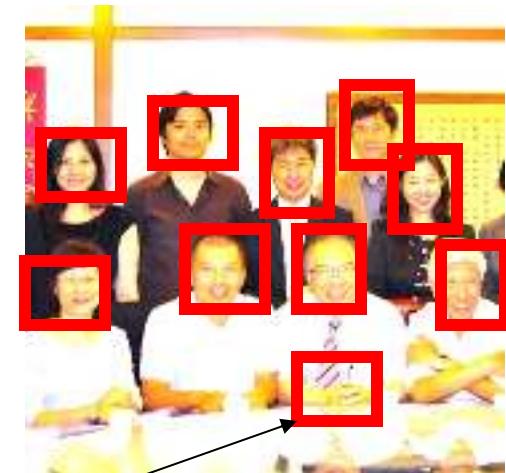
Face detection

- **Basic idea:** slide a window across image and evaluate a face model at every location



Evaluation of face detection

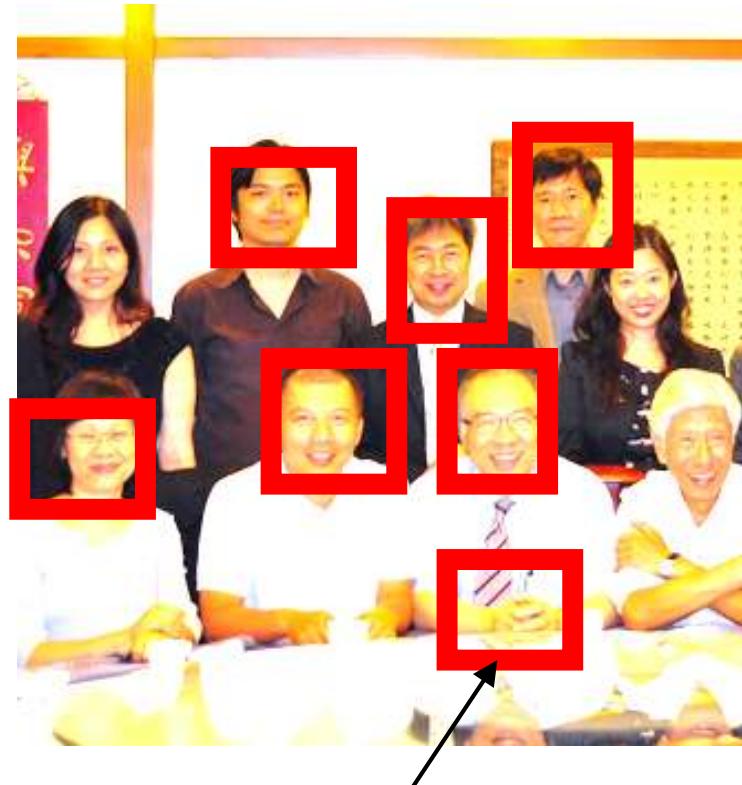
- **Detection rate**
 - Total number of faces that are correctly detected/total number of faces exist in the picture
 - Should be high $> 95\%$.
- **False positive rate**
 - The detector output is positive, but it is false (there is actually no face)
 - Should be low $< 10^{-6}$
- A **good system** has
 - High detection rate
 - Low false positive rate



False positive result

Example

- What are the detection rate and false positive rate here?

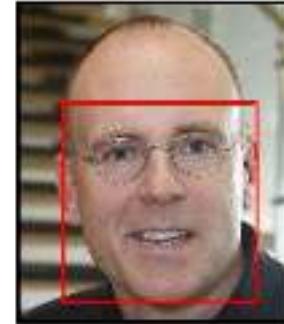


False positive result

Face Detection: Viola Jones Algorithm (VJA)

- The most famous method
 - Training is slow but recognition is very fast

- **Key Steps**
 1. Integral image for fast feature extraction
 2. Boosting for feature selection
 3. Attentional cascade for fast rejection of non-face sub-windows



Paul Viola



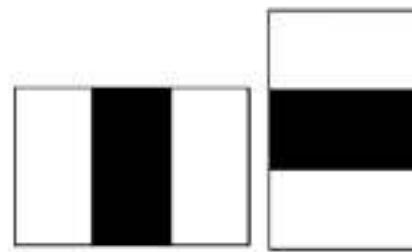
Michael J. Jones

VJA Image Features

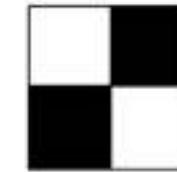
- A very simple feature calculation method (Haar-like Features)



Edge Features



Line Features



Four-rectangle
Features

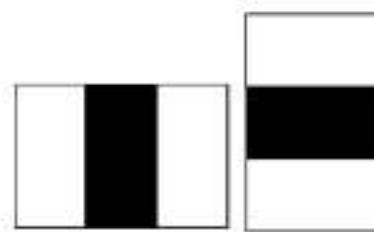


Alfred Haar

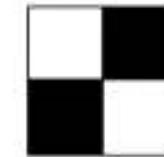
What do these features have to do with faces?



Edge Features



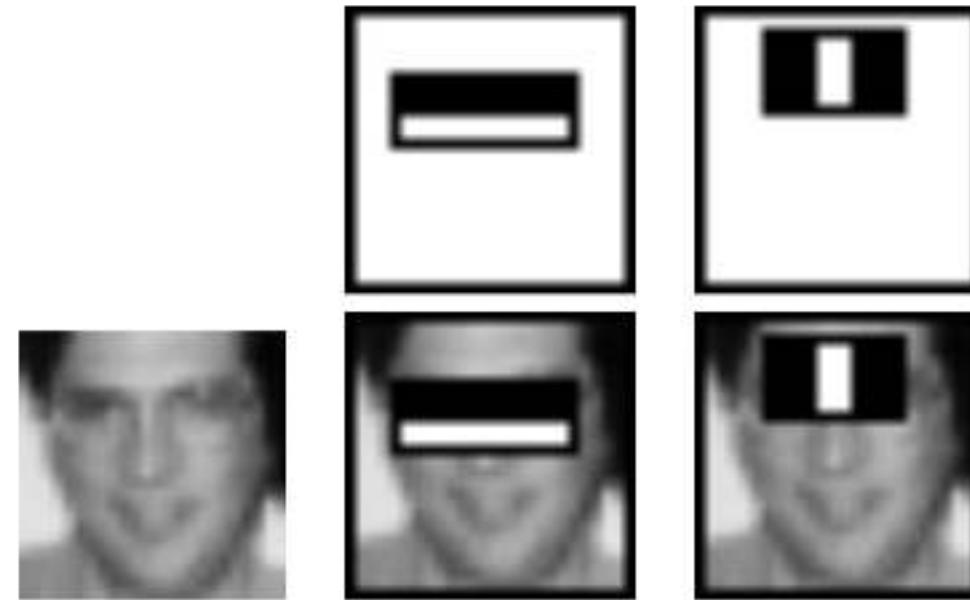
Line Features



Four-rectangle
Features

- These features have real importance in the context of face detection:
 - Eye regions tend to be darker than cheek regions.
 - The nose region is brighter than the eye region.
 - ...

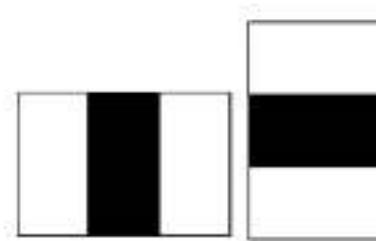
What do these features have to do with faces?



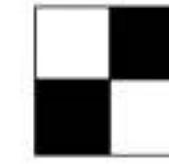
Feature Calculation



Edge Features



Line Features



Four-rectangle
Features

- To obtain features for each of these five rectangular areas, we simply **subtract the sum of pixels under the white region from the sum of pixel from the black region**

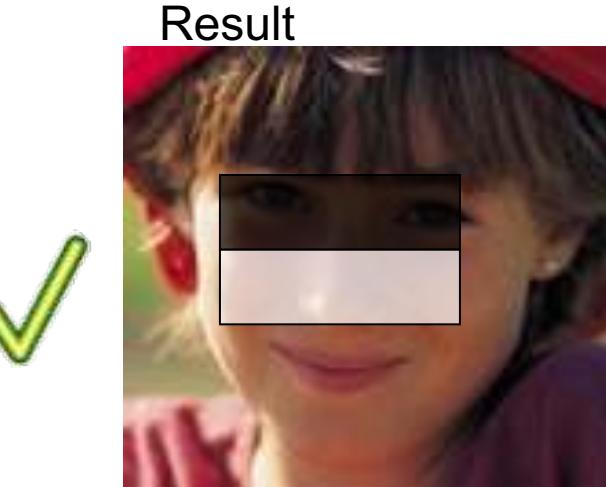
Face vs Non-face

- Example: A simple face detection method using one feature in the middle of image

$$f = \sum(\text{pixels in white area}) - \sum(\text{pixels in shaded area})$$



This is not a face.
Because f is small

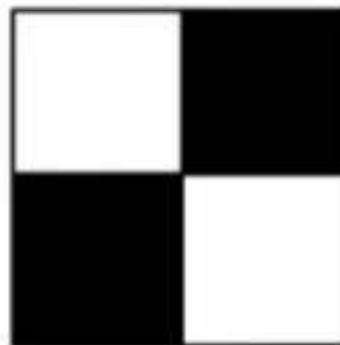


This is a face: The eye-area (shaded area) is dark, the nose-area (white area) is bright. So f is large, hence it is face

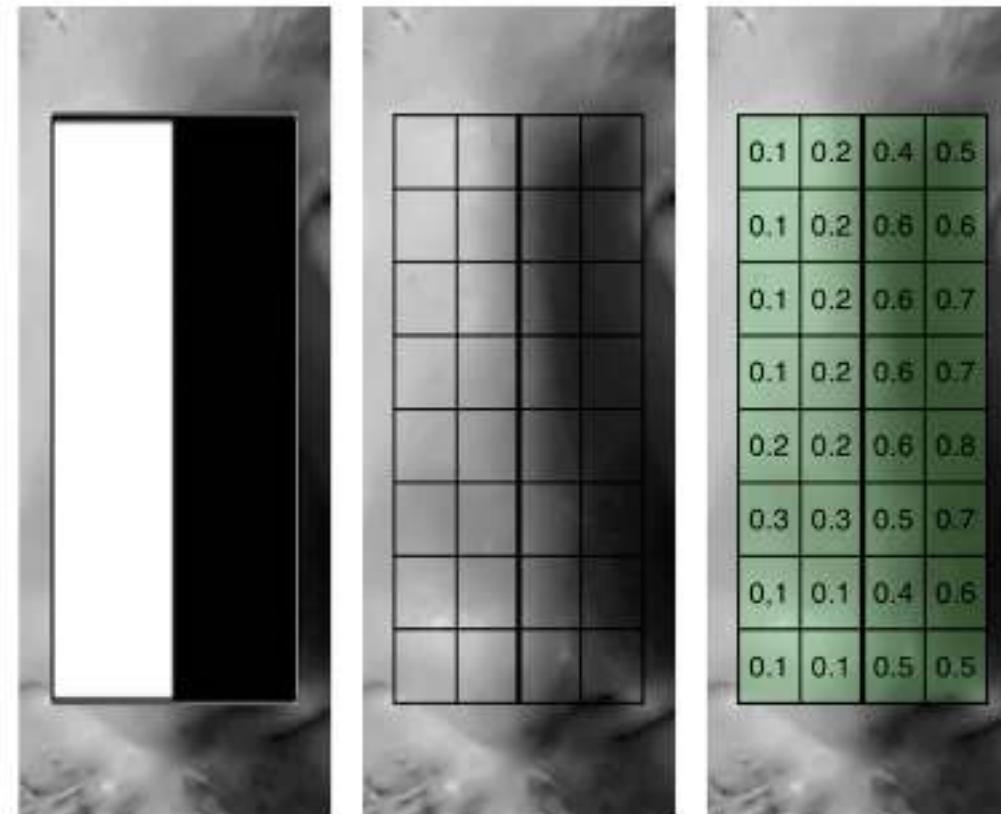
if $(f) > \text{threshold}$, then
face
else
non-face

Check Point!

- How you will calculate this?



Computation can be extremely expensive

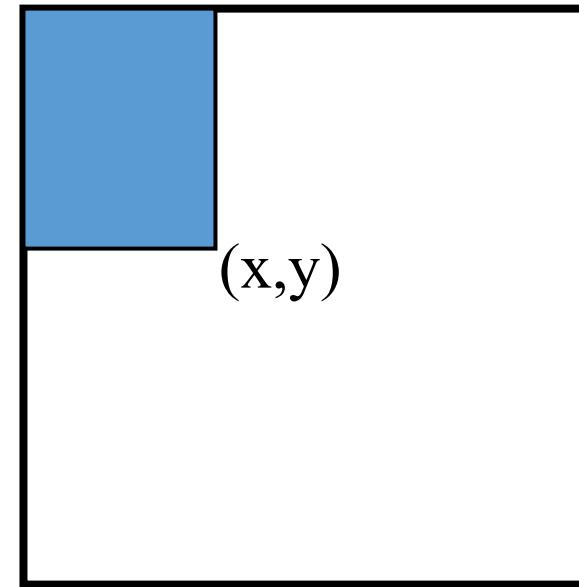


Integral Image comes to rescue!!

Image Source: SuperDataScience

Integral Image – A way to find features faster

integral image = sum of all pixel values
above and to the left of (x,y)



Integral Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Input Image

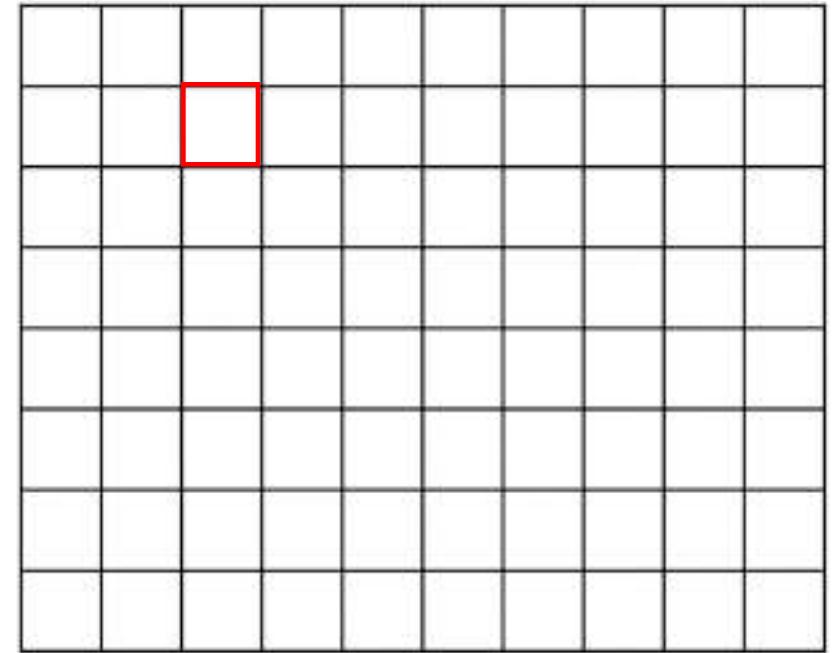
Integral Image

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Integral Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Input Image



Integral Image

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Integral Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Input Image

Integral Image

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Integral Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Input Image

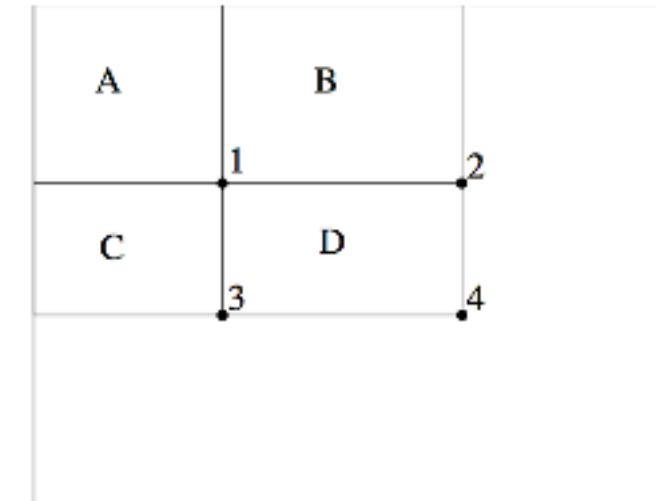
1	3	8	15	17	25	25	31	35	41
10	20	25	36	47	60	70	83	97	106
17	33	48	61	72	95	109	131	155	172
20	44	60	78	93	124	138	169	198	223
29	58	74	93	111	146	161	201	236	262
30	61	82	107	134	178	193	235	274	300
31	64	89	115	148	198	223	269	310	341
36	75	102	138	176	229	263	319	370	403

Integral Image

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

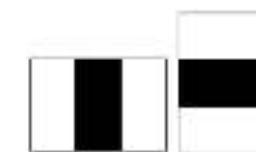
Use of Integral Images in VJA

- The sum of the pixels within rectangle D can be computed with four array references.
- The value of the integral image at location 1 is the sum pixels in rectangle A
- The value at location 2 is A+ B, at location 3 is A + C, and at location 4 is A+B+C+D.
- The sum within D can be computed as $1 + 4 - (3 + 2)$.



Edge Features

6-points



Line Features

8-points



Four-rectangle Features

9-points

Because: $A + (A + B + C + D) - (A + C + A + B)$

Use of Integral Images in VJA

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

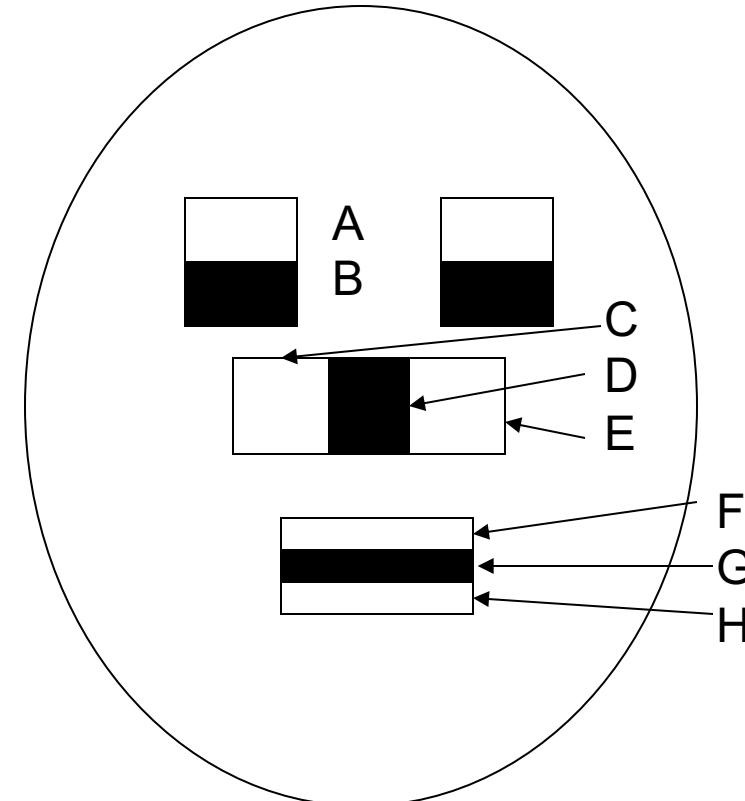
1	3	8	15	17	25	25	31	35	41
10	20	25	36	47	60	70	83	97	106
17	33	48	61	72	95	109	131	155	172
20	44	60	78	93	124	138	169	198	223
29	58	74	93	111	146	161	201	236	262
30	61	82	107	134	178	193	235	274	300
31	64	89	115	148	198	223	269	310	341
36	75	102	138	176	229	263	319	370	403

$$235 - 83 + 47 - 134 = 65$$

Image Source: SuperDataScience

Why do we need to find pixel sum of rectangles?

- You may consider these features as face features
 - Left Eye: $(\text{Area_A} - \text{Area_B})$
 - Nose: $(\text{Area_C} + \text{Area_E} - \text{Area_D})$
 - Mouth: $(\text{Area_F} + \text{Area_H} - \text{Area_G})$
- They can be different sizes and aspect ratios



Basic Types of Features

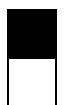
$$f = \sum(\text{pixels in white area}) - \sum(\text{pixels in shaded area})$$

- Type) Rows x columns

- Type 1) 1x2



- Type 2) 2x1



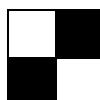
- Type 3) 1x3



- Type 4) 3x1

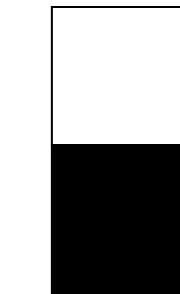
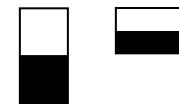


- Type 5) 2x2



- Each basic type can have different sizes and aspect ratios.

- i.e., the following feature windows are of the same type (Type2) even they have different sizes, or aspect ratios



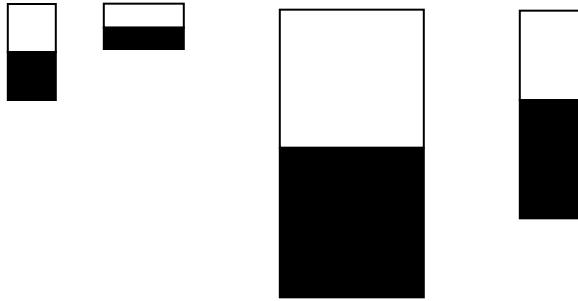
Faces can be any sizes

- **Example:** A face can be big or small, from 24 x24 to 1024x1024
- There are faces with different sizes



So, we need feature windows with different sizes.

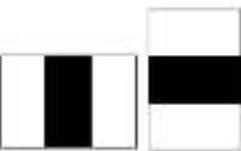
- As long as white/gray areas have the relations
- The followings are Type2 Rectangular Features
 - The white rectangle is above the shaded rectangle
 - White and shaded rectangle are of same dimension



VJA Training



Edge Features



Line Features



Four-rectangle
Features

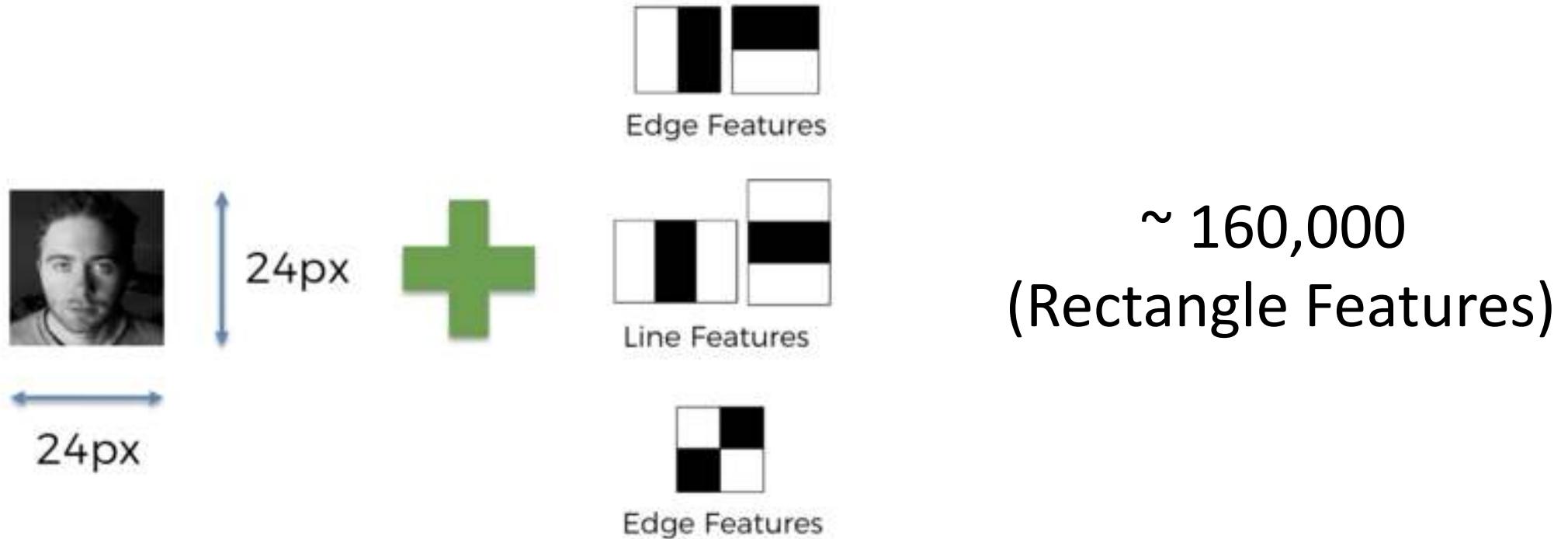


Face Images



Non-Face Images

But there is a problem!



- Even though each feature can be computed very efficiently, computing the complete set is **prohibitively expensive**.
- Viola and Jones found that a **very small number of these features** can be combined to **form an effective classifier**.
The main challenge was to find these features.

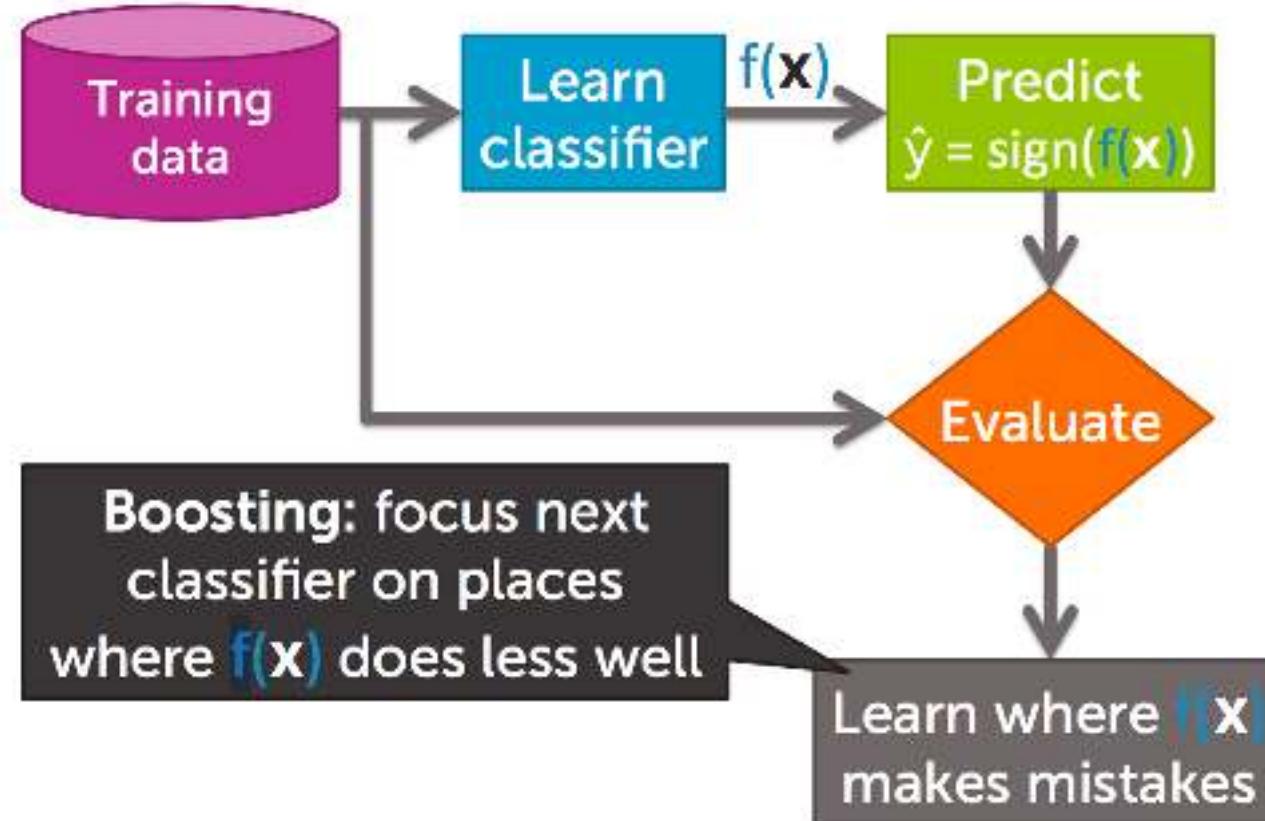
Solution to make it efficient

- The whole 162,336 feature set is too large
 - Solution: select good features to make it more efficient.
 - Use: “Boosting”
- **Boosting**
 - Boosting is a classification scheme that works by combining weak learners into a more accurate ensemble classifier
 - Weak learner: classifier with accuracy that need be only better than chance
 - We can define weak learners based on rectangle features
 - **Training is needed.**

Boosting: Train A Classifier



Boosting: Train Next Classifier by Focusing More on the Hard Points



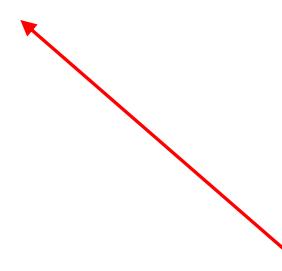
Adaboost for Feature Selection



$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \dots$$



Strong Classifier



Weak Classifiers

Face Detection using Adaboost

- **AdaBoost Training**

- For Example: Collect 5000 faces, and 9400 non-faces.
- Use AdaBoost for training to build a strong classifier.
- Pick suitable features of different scales and positions, pick the best few. (Take months to do , details is in [Viola 2004] paper)

- **Testing**

- Scan through the image (any where), pick a window (any size $\geq 24 \times 24$) and rescale it to 24×24
- Pass it to the strong classifier for detection
- Report face, if the output is positive

Boosting for Face Detection [viola2004]

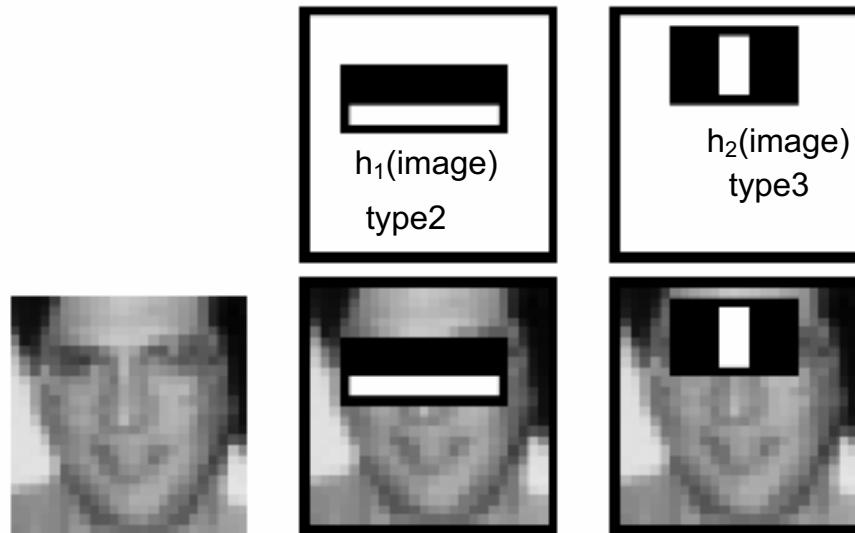
- In the paper it shows that the following two features (obtained after training) in cascaded picked by AdaBoost have 100% detection rate and 50% false positive rate
- But 50% false positive rate is not good enough
- Approach [viola2004]: Attentional cascade

i.e. Strong classifier

$$H(\text{face}) = \text{Sign}\{\alpha_1 h_1(\text{image}) + \alpha_2 h_2(\text{image})\}$$

$H(\text{face}) = +1 \rightarrow \text{face}$

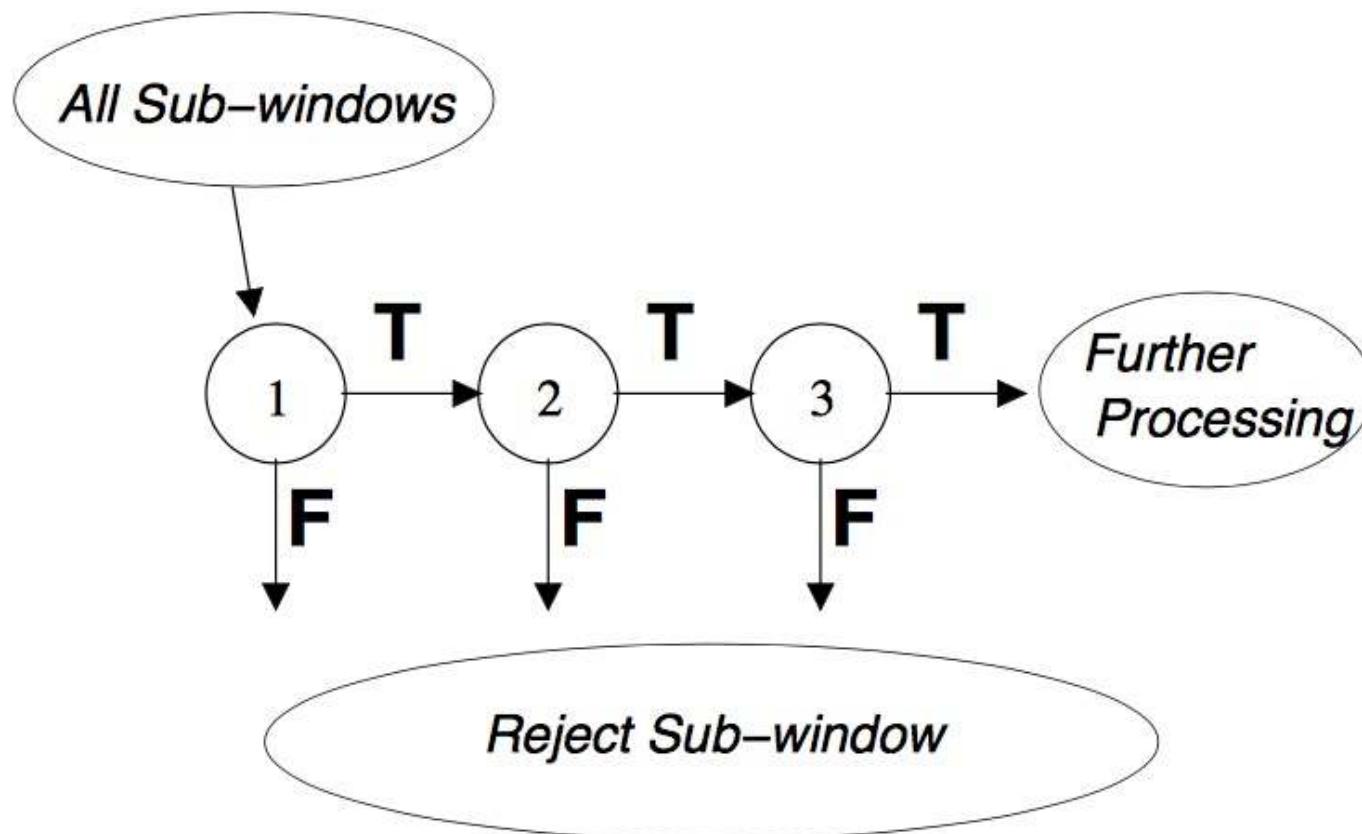
$H(\text{face}) = -1 \rightarrow \text{non-face}$



Standard Types	
1)	
2)	
3)	
4)	
5)	



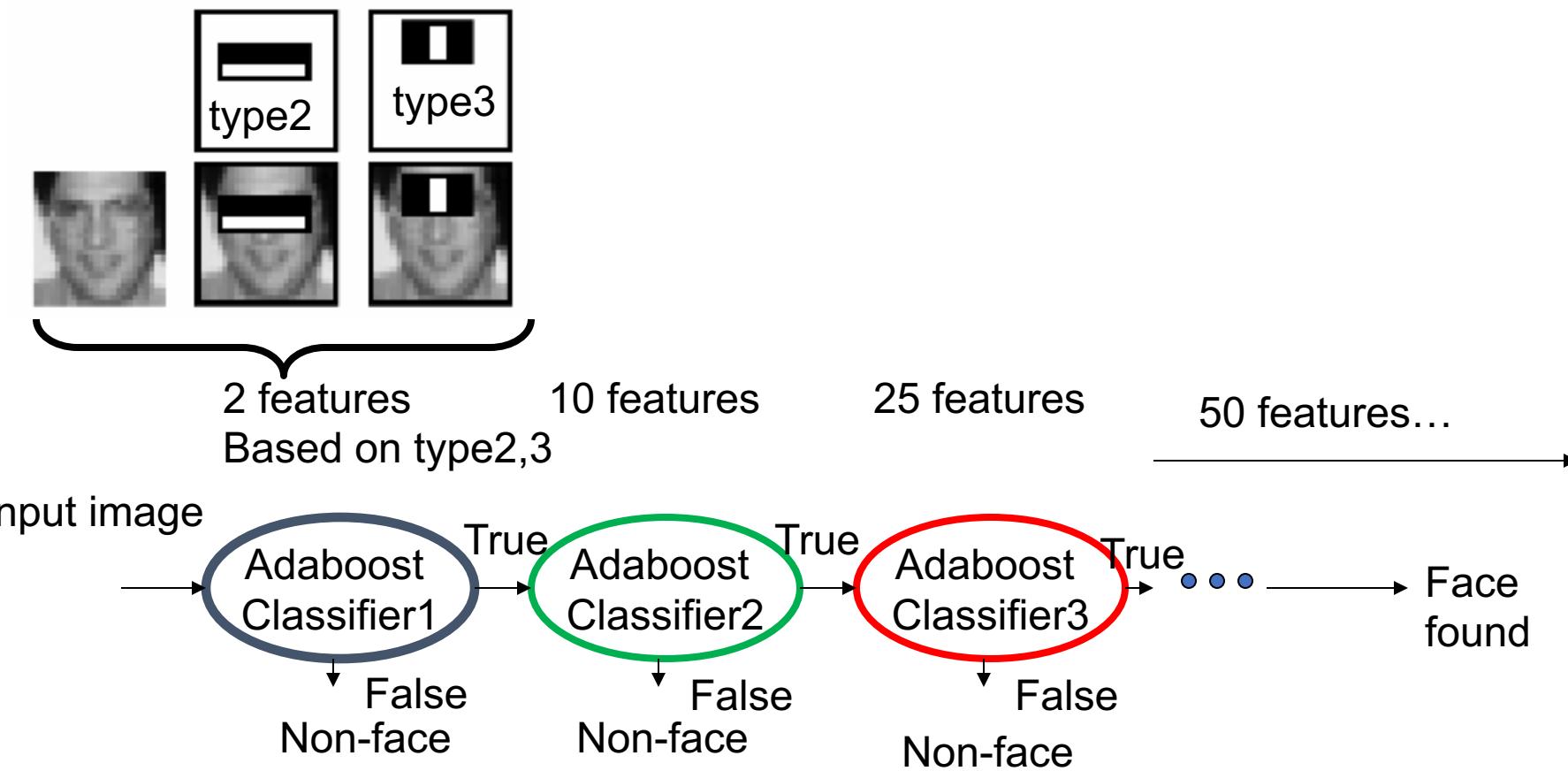
Attentional Cascading



Where each classifier uses a small number of tests (say, a two-term AdaBoost classifier) to reject a large fraction of non-faces while trying to pass through all potential face candidates

An Example

- More features for later stages in the cascade [viola2004]



Face Recognition

- Eigenfaces

Face Recognition Objectives

- What is face recognition?
- Learn about Eigenfaces
- Learn how they can be used for face recognition

Face Recognition

- **Face recognition** is the challenge of classifying whose face is in an input image.
- This is **different than face detection** where the challenge is determining if there is a face in the input image.
- With face recognition, we need an **existing database of faces**. Given a new image of a face, we need to **report the person's name**.

Face Recognition Applications

- Face recognition can be used in a variety of applications:
 - Criminal identification,
 - Identity verification,
 - patient monitoring,
 - etc.

Can we use individual's features?

- Individual features

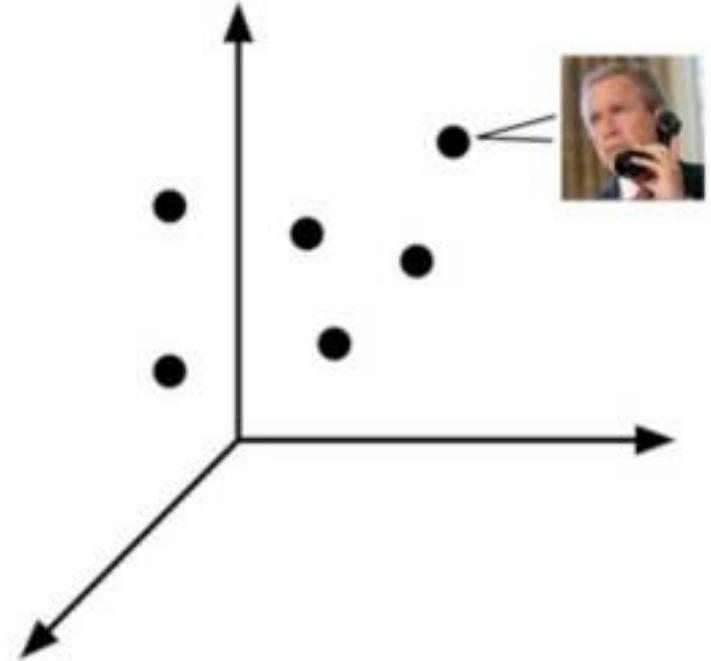
- eyes, nose, mouth, head outline
- position and size relationships



Eigenfaces

- The eigenface approach

- images are points in a vector space
- use PCA to reduce dimensionality
- face space
 - Kirby and Sirovich [1990]



- Compare projections onto face space to recognize faces

Steps

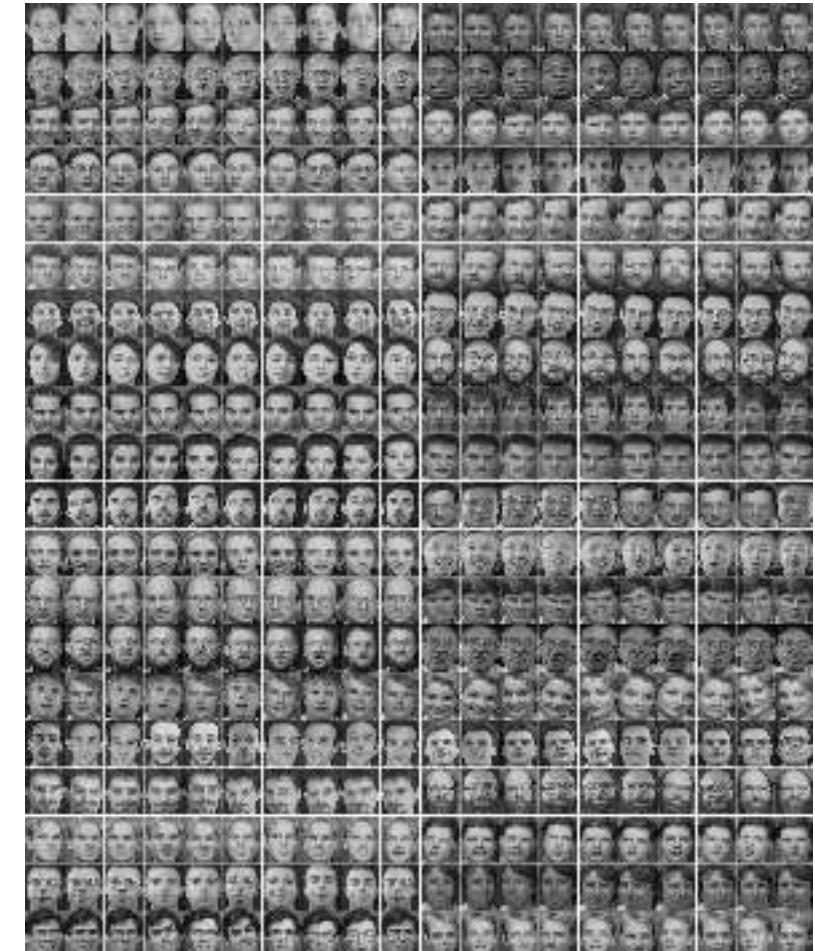
1. Get the images
2. Calculate the eigen faces
3. Projection into face space
4. Report the image if it's known face (face recognition).

Special Case:

- If it's unknown face and you want to add it. In this case you need to calculate the step 2 for all images again.

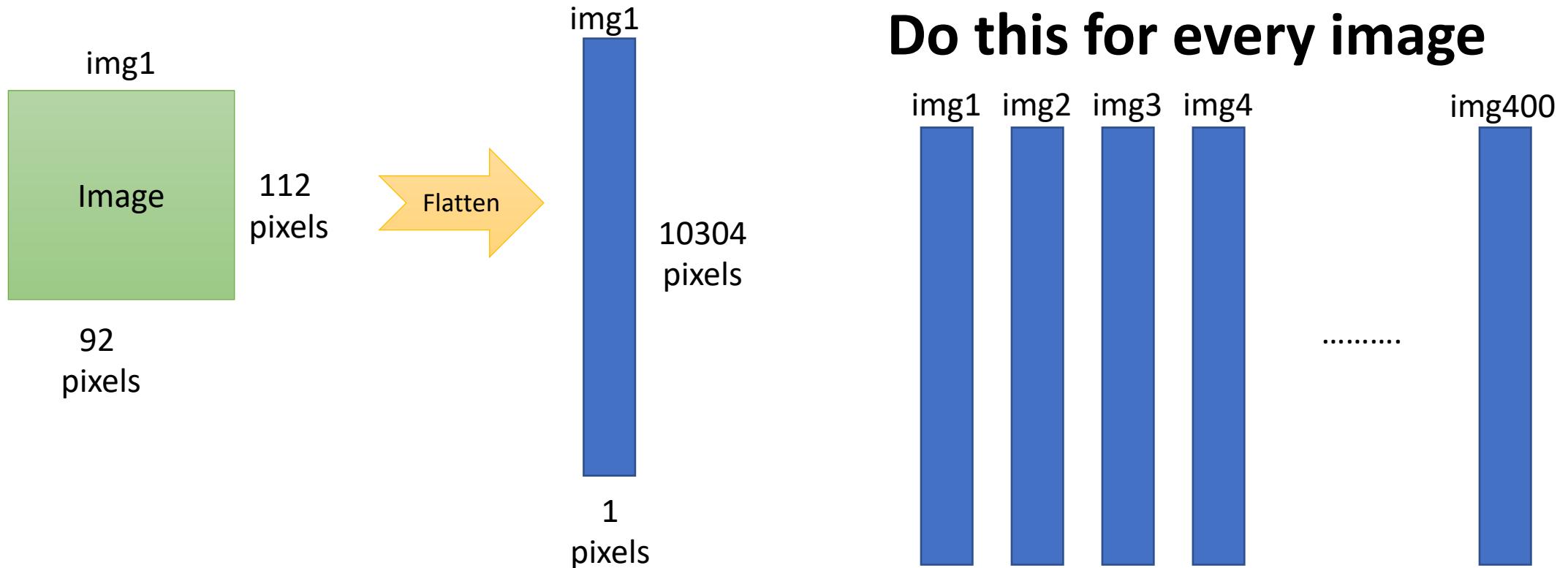
Step 1: Get the images

- **The Database of Faces (AT&T)**
- **Dataset statistics**
 - **Color:** Grey-scale
 - **Image Size:** 92x112
 - **No of Samples:**
10 images of one subject x 40 subjects = 400
- **NOTE:**
 - These 10 images vary in terms of
 - Lightning
 - Facial expression
 - Facial details

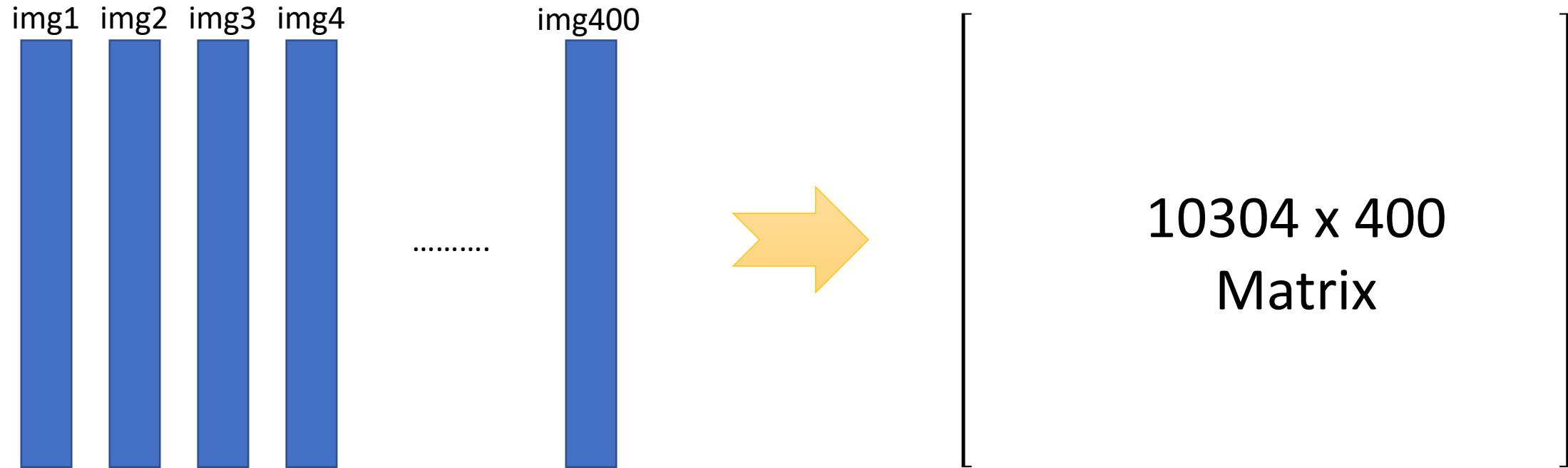


https://git-disl.github.io/GTDLBench/datasets/att_face_dataset/

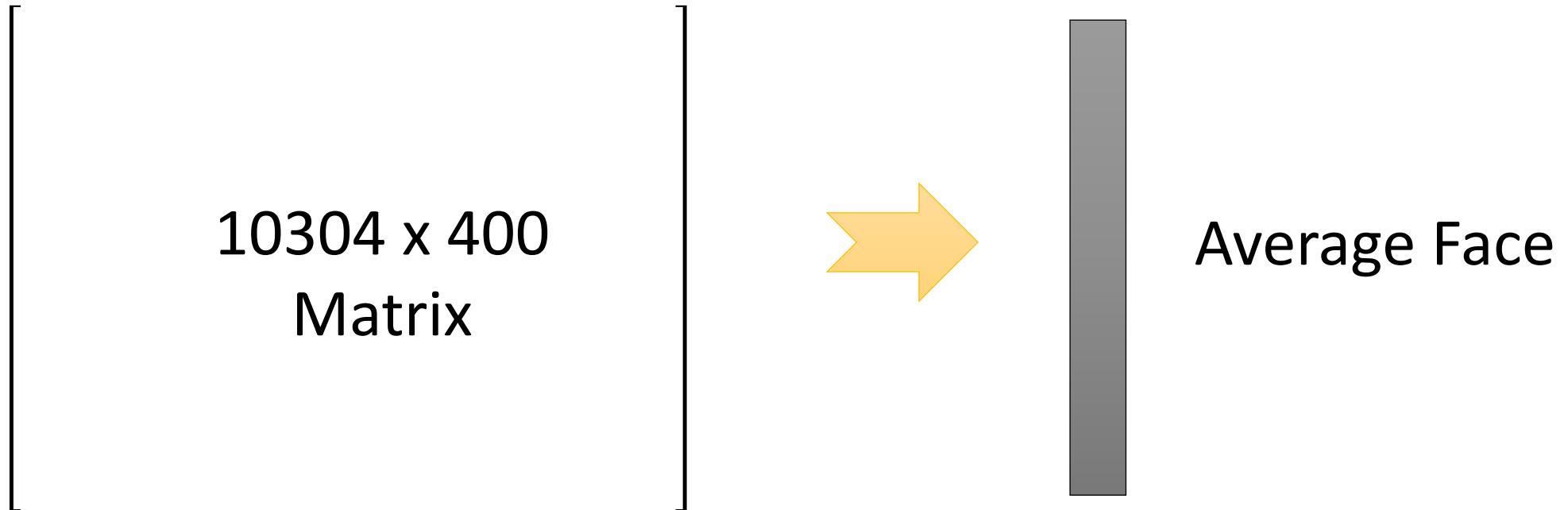
Step 2 – Calculate the Eigen Faces



Step 2 – Calculate the Eigen Faces



Step 2 – Calculate the Eigen Faces (Average Face)



Sum of all images divided by total number of images in the database.

Step 2 – Mathematically

- Images in database

$$\Gamma_1, \Gamma_2, \Gamma_3, \dots \Gamma_M$$

M is total number of images

Average Face

$$\Psi = \frac{1}{M} \sum_{i=1}^M \Gamma_i$$

Step 2 – Calculate the Eigen Faces

Adjusted Images

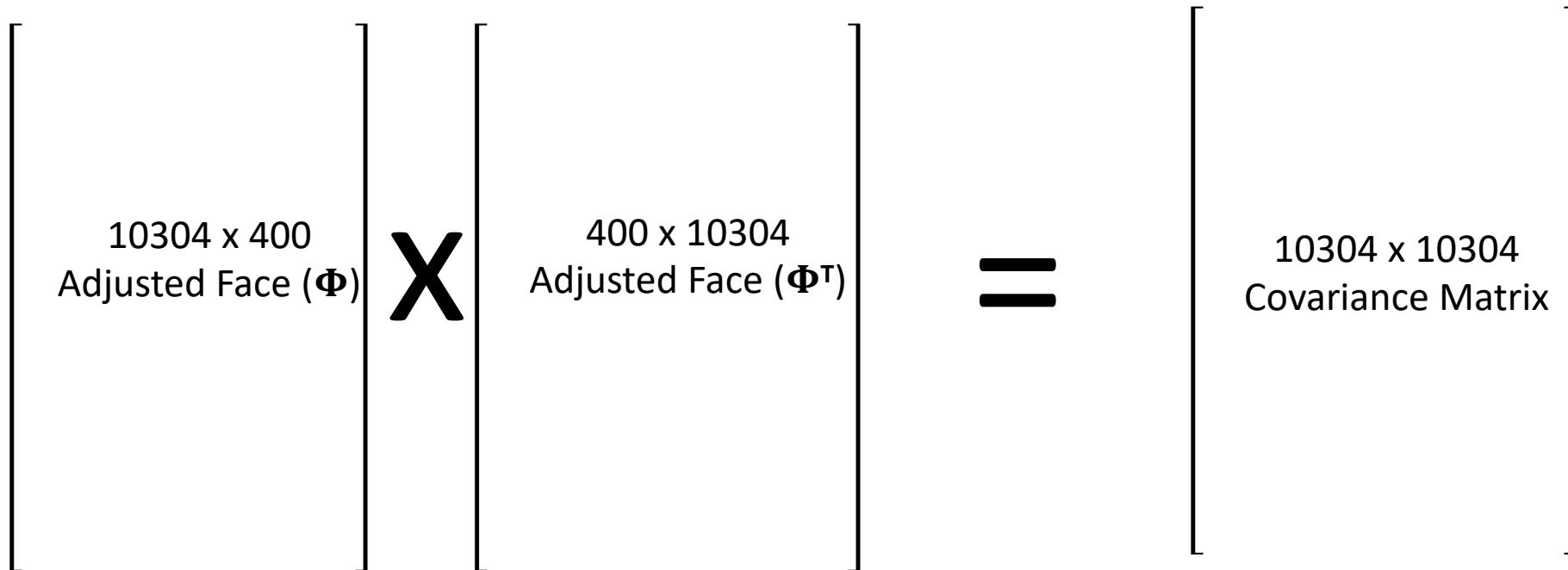
$$\Phi_i = \Gamma_i - \Psi$$

$$\begin{bmatrix} 10304 \times 400 \\ \text{Adjusted Face} \end{bmatrix} = \begin{bmatrix} 10304 \times 400 \\ \text{Images Matrix} \end{bmatrix} - \begin{bmatrix} \text{Average Face} \end{bmatrix}$$

Step 2 – Calculate the Eigen Faces

Compute the Covariance Matrix

$$C = \frac{1}{M} \sum_{n=1}^M \Phi_n \Phi_n^T = AA^T$$



Absolutely intractable task!

Any idea...

Step 2 – Calculate the Eigen Faces

Simple mathematical trick

$$\begin{bmatrix} 400 \times 10304 \\ \text{Adjusted Face } (\Phi) \end{bmatrix} \times \begin{bmatrix} 10304 \times 400 \\ \text{Adjusted Face } (\Phi^T) \end{bmatrix} = \begin{bmatrix} 400 \times 400 \\ \text{Covariance Matrix} \end{bmatrix}$$

NEXT: Compute the PCA to get
eigen vectors

Step 2 – Calculate the Eigen Faces

- We can get matrix of eigenfaces \mathbf{u} as:

$$V \times \Phi \rightarrow U$$

- Finally, we got the **eigenfaces**.

Step 3. Projection into Face Space

- A face image can be projected into this face space by

$$\Omega_k = U^T(\Gamma^k - \Psi)$$

Where k=1,...,M

Step 4. Face Recognition

- The test image Γ is projected into the face space to obtain a vector, Ω :

$$\Omega = U^T(\Gamma - \Psi)$$

- The distance of Ω to each face class is defined by

$$\epsilon_k^2 = ||\Omega - \Omega_k||^2; k = 1, \dots, M$$

- A threshold is set to recognize the face.

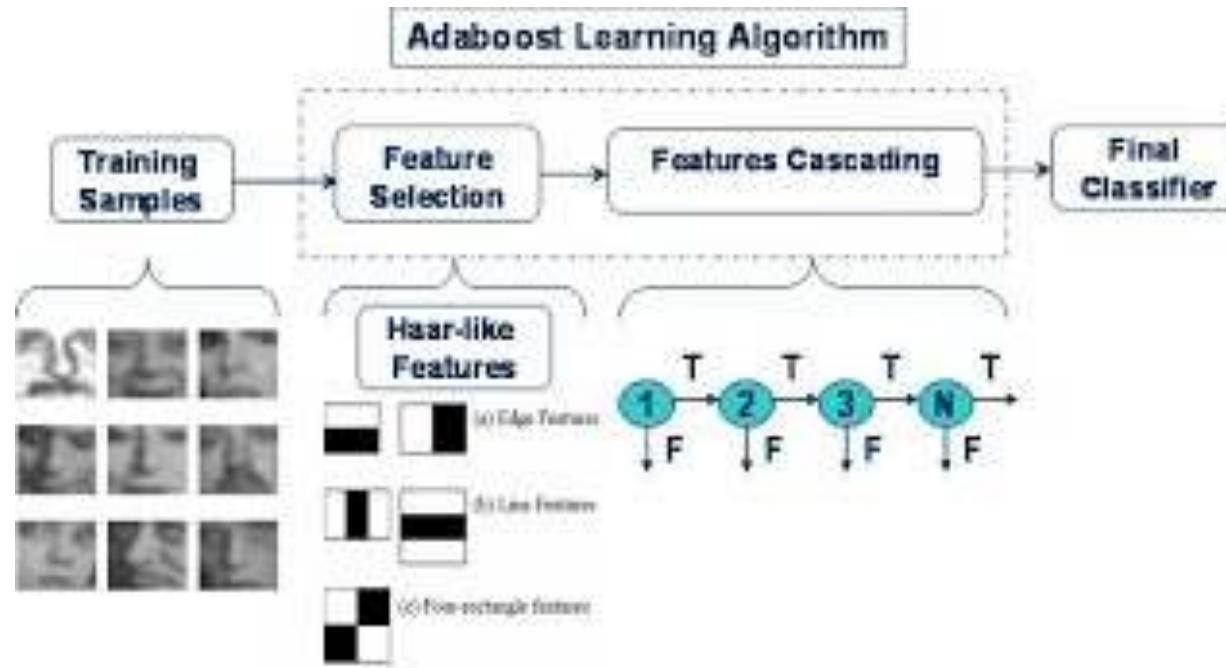
Reading

- DeepFace (*Read yourself*).
 - Rapid object detection using a boosted cascade of simple features (2011)
 - Pick important papers from the reference list (from above paper) and read them
 - M. Turk and A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71-86, 1991
 - Y. Freund and R. Schapire, A short introduction to boosting, *Journal of Japanese Society for Artificial Intelligence*, 14(5):771-780, September 1999.

Thank You 😊

Appendix

Appendix



- A face detection algorithm that could be operated in real-time
 - 95% accuracy at around 17fps.

Covariance matrix to semi-definite matrix.

calculation for the covariance matrix can be also expressed as

$$C = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})^T$$

where our data set is expressed by the matrix $X \in \mathbb{R}^{n \times d}$. Following from this equation, the covariance matrix can be computed for a data set with zero mean with $C = \frac{XX^T}{n-1}$ by using the semi-definite matrix XX^T .

Appendix: Cascading Classifiers



Waterfall



Cascade

- Key difference between the two: in a waterfall, water moves only downward, while in a cascade, it moves both downward and onward, like stairs.

<https://outdoors.stackexchange.com/questions/14367/what-is-the-difference-between-a-cascade-and-a-waterfall>

Practical Implementation

- Details discussed in Viola-Jones paper
- Training time = weeks (with 5k faces and 9.5k non-faces)
- Final detector has 38 layers in the cascade, 6060 features
- 700 Mhz processor:
 - Can process a 384 x 288 image in 0.067 seconds (in 2003 when paper was written)

Hough Transform

Instructor: Dr. Muhammad Fahim

Source of the material

- This lecture is based on the following resources
 - Lecture slides of Prof. Adil Khan, Innopolis University.
 - Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe
 - Found material over the internet to aligned the subject according to the need of the students.

Contents

- Introduction
- Hough Transform
- Hough Space Calculation
- Hough Lines
- Hough Circles
- Pros and Cons of Hough Transform

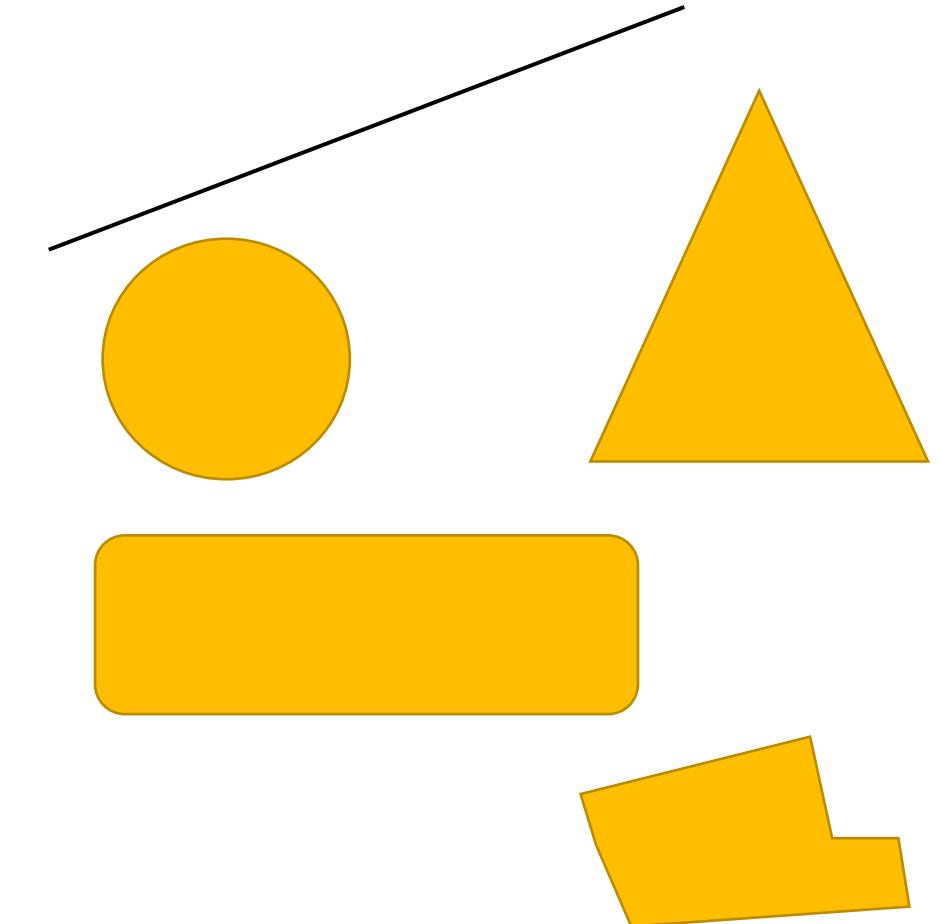
Recap! Image Features

- Edge Detection
 - Sobel, Robert, Prewit
 - Laplacian of Gaussian
 - Canny
- Interest Points
 - Harris
 - SIFT
- Descriptors
 - SIFT
 - HOG

Today! Shape Features

What are the Shape Features?

- Line shape
- Circle shape
- Ellipse shape
- Triangles shape and so on...

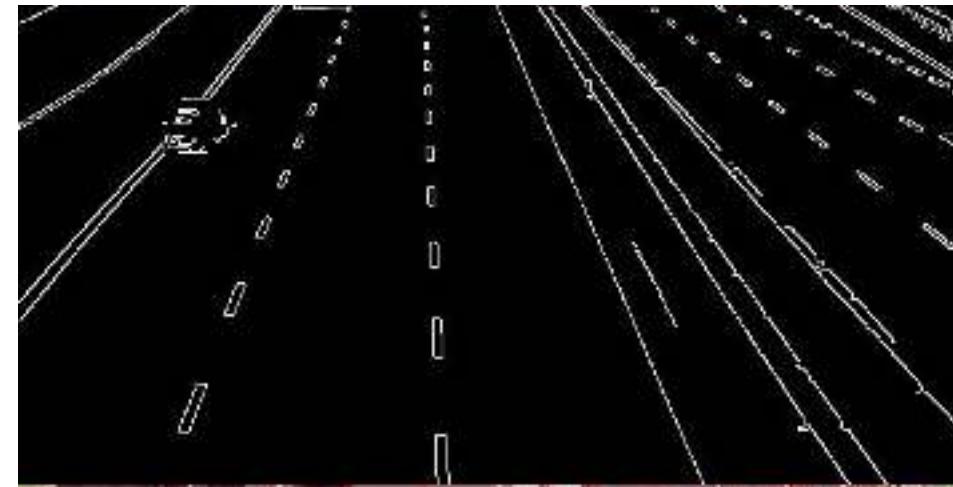


Hough Transform

- The Hough Transform was been invented by **Paul Hough** in 1962 and patented by IBM.
- It has become a **standard tool** in the domain of computer vision for the recognition of straight lines, circles and ellipses.
- The Hough Transform is particularly **robust to missing** and **contaminated data**.

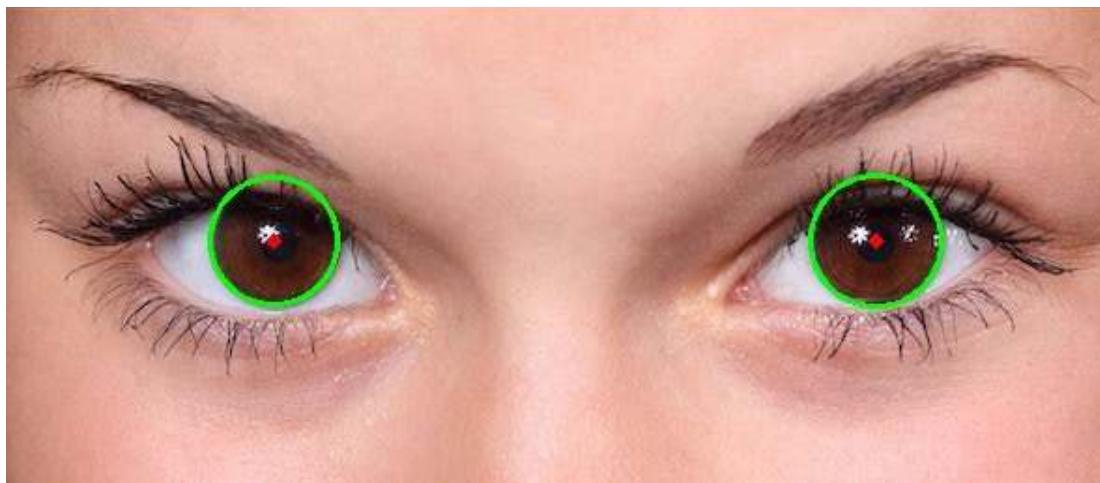
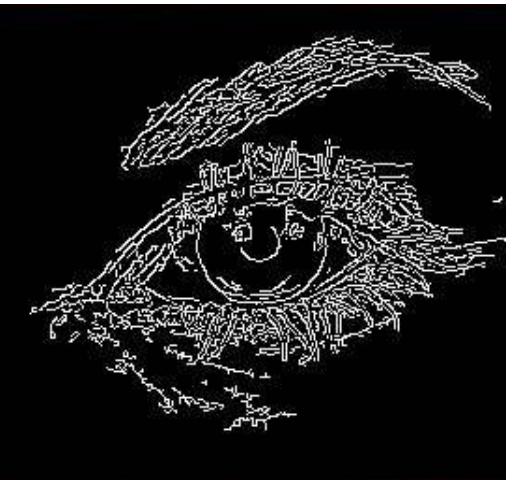
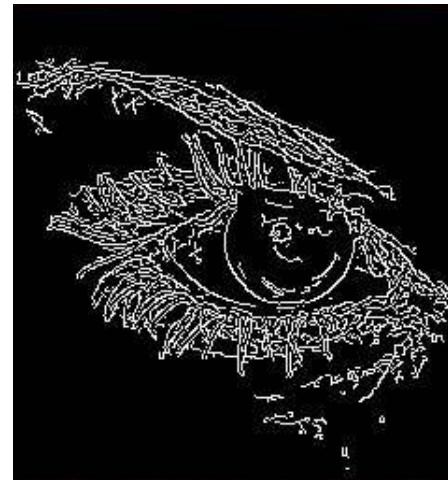
It can conduct a search in parametric space for any number of lines (or other geometric objects that have parametric descriptions).

HoughLines: Line Detection



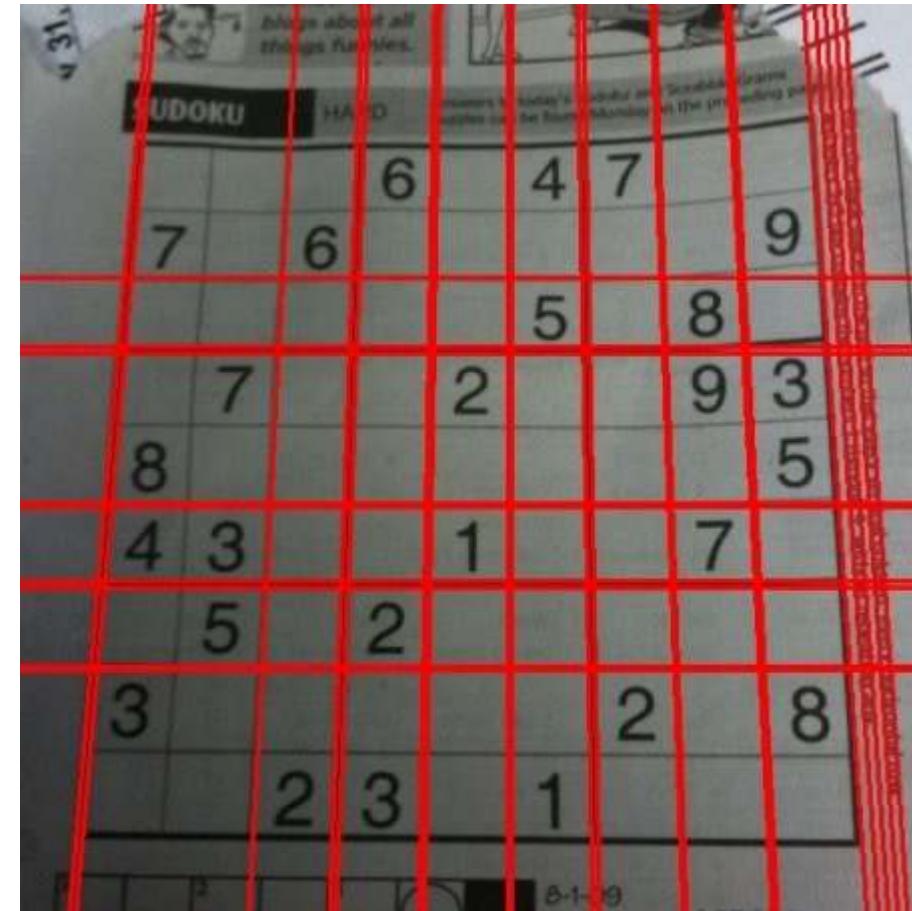
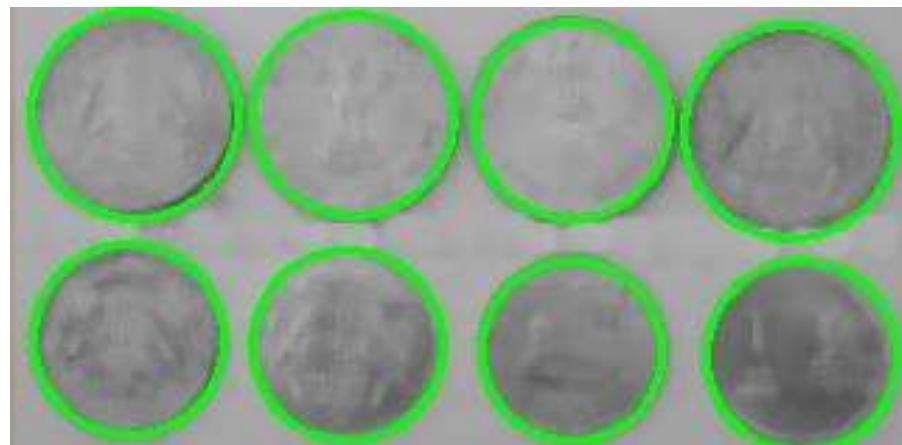
<https://www.learnopencv.com/hough-transform-with-opencv-c-python/>

HoughCircles : Detect circles in an image



<https://www.learnopencv.com/hough-transform-with-opencv-c-python/>

Other Possible Applications



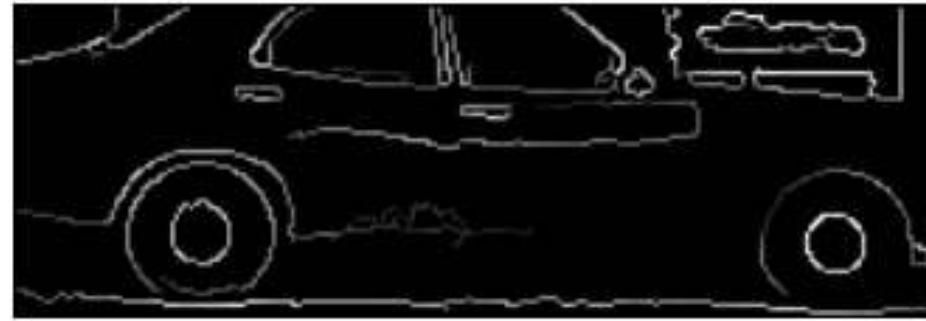
https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

<https://www.learnopencv.com/hough-transform-with-opencv-c-python/>

Other Possible Applications



Gray level image



Canny edges

Hough Transform



Most prominent circle

Hough Transform

- The Hough strategy is to find parametric combination that fit the data.
 - The slope-intercept form of a straight line is

$$y = mx + b$$

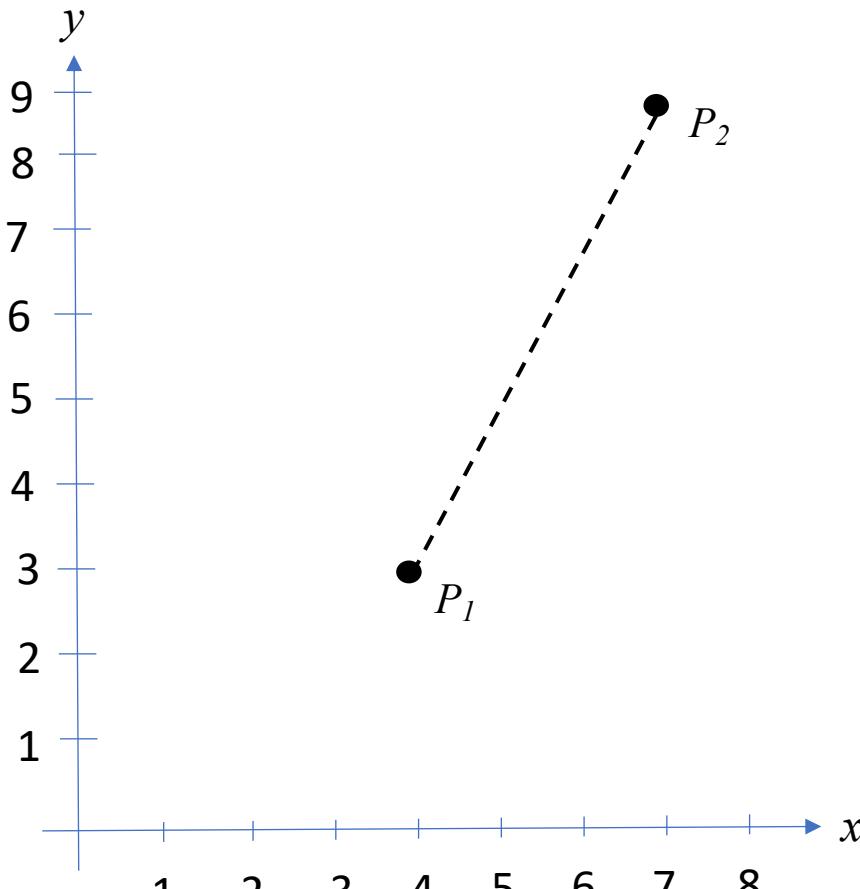
- This can be inverted to express the parameter b in terms of m at each data point.

$$b = -mx + y$$

- This represents a straight line in parametric space for each (x, y) pair.
- Only one parameter combination satisfies all points on a given line.

The parameter-space lines will cross at (m, b) point.

Equation of Line



$$y = 2x - 5$$

- $P_1(4, 3)$
- $P_2(7, 9)$

How to find line?

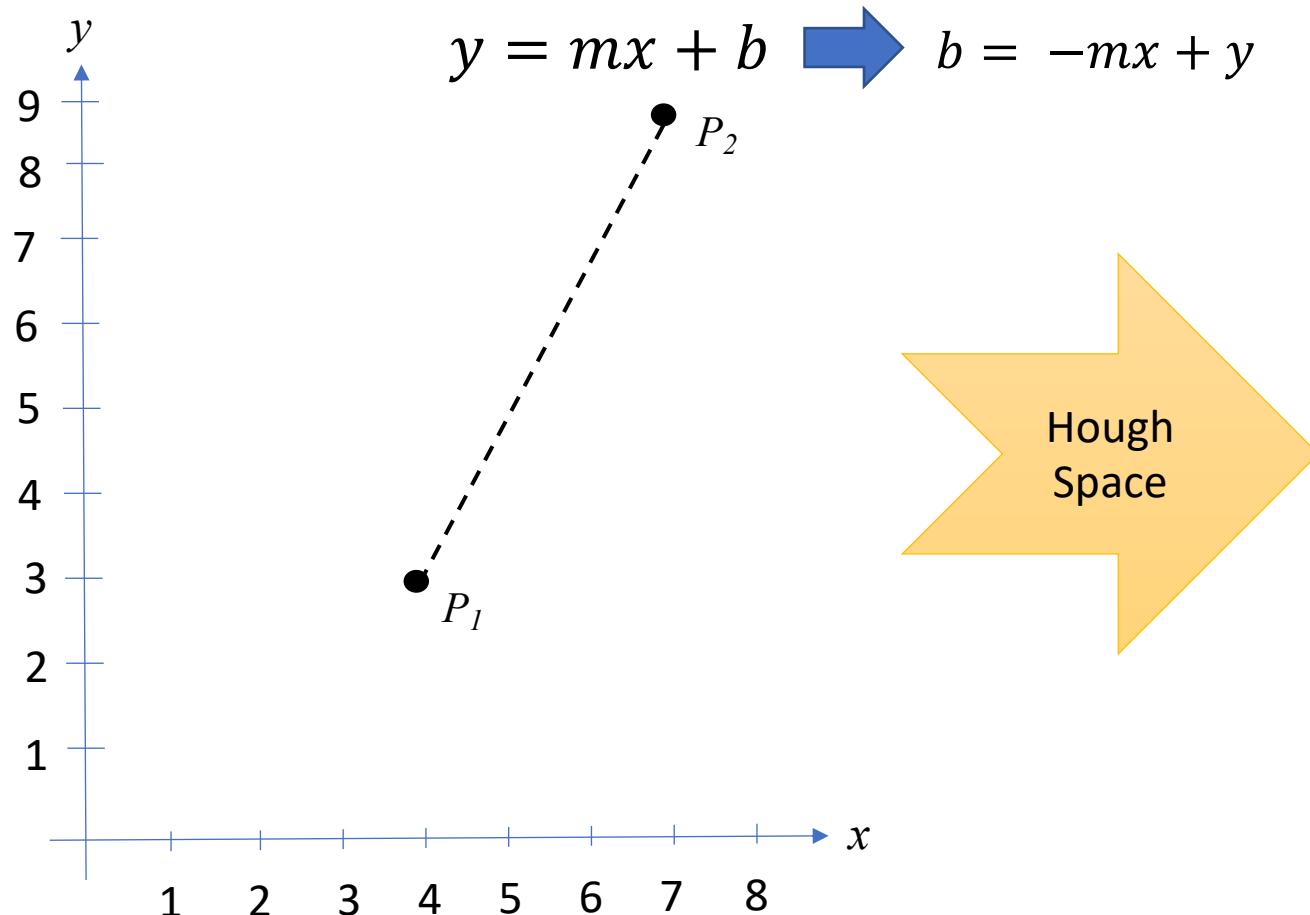
$$y = mx + b$$

$$m = \text{slope/gradient} = \frac{\Delta y}{\Delta x}$$

$$= \frac{y_2 - y_1}{x_2 - x_1}$$

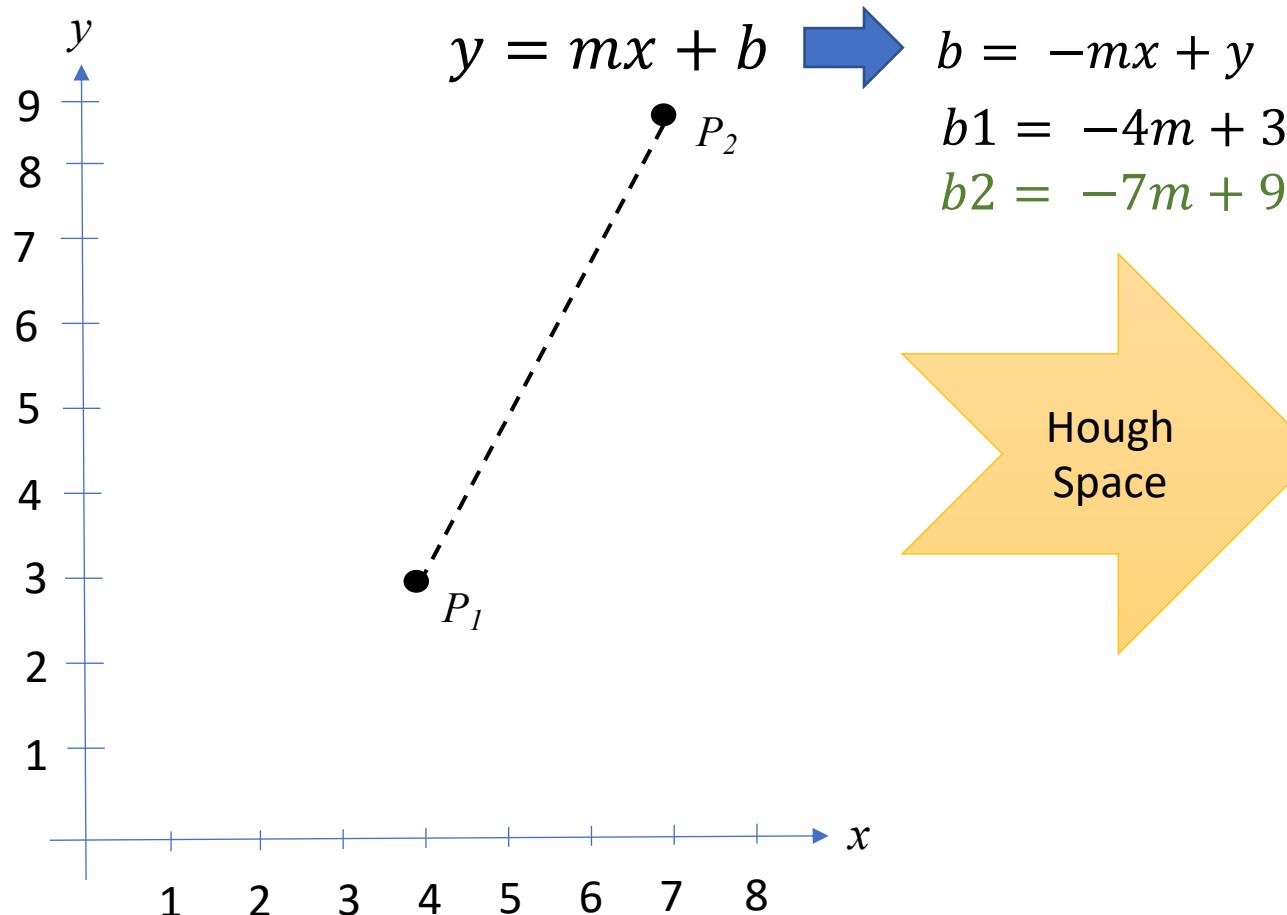
$b = \text{intercept or where } y = 0$

Transformation to Hough Space



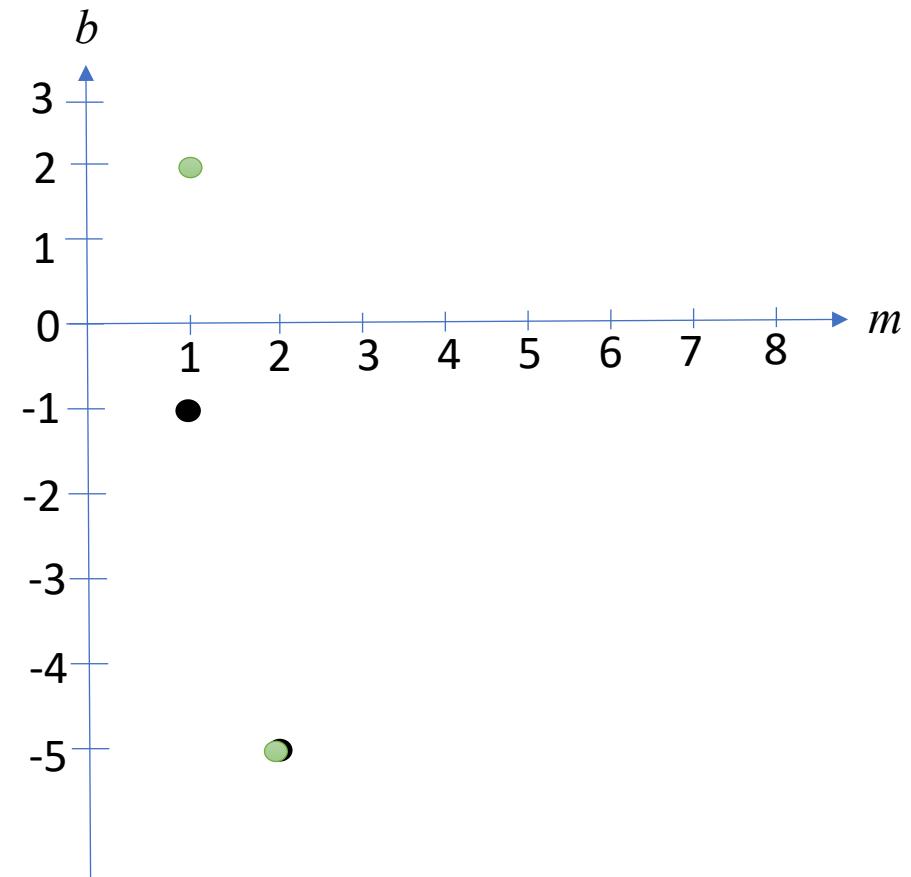
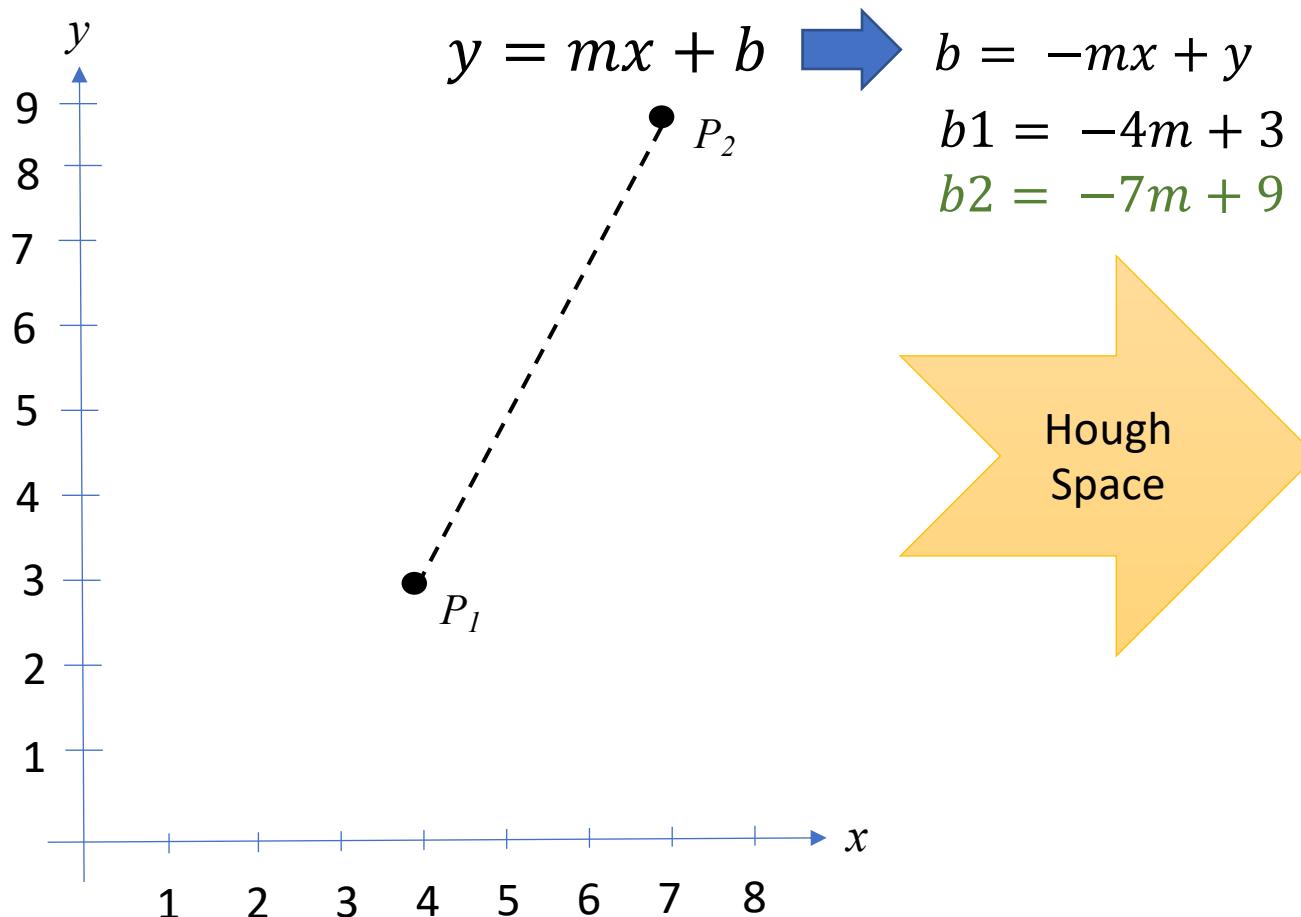
- $P_1(4,3)$ and $P_2(7,9)$

Equation of Line



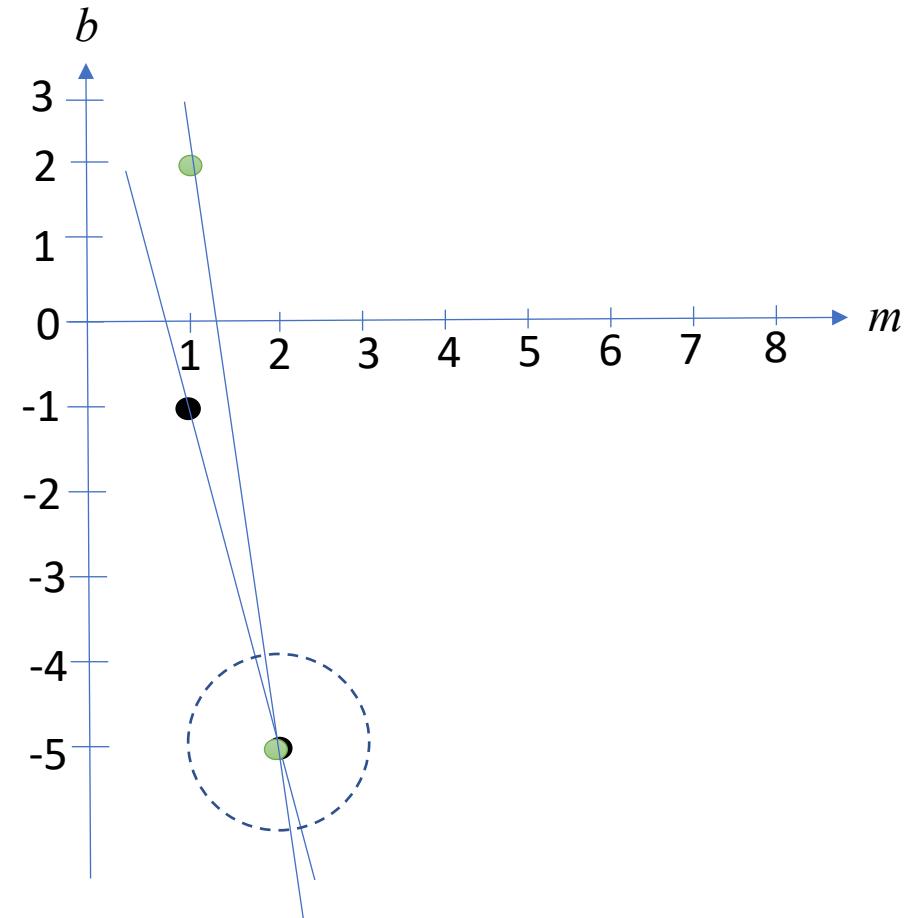
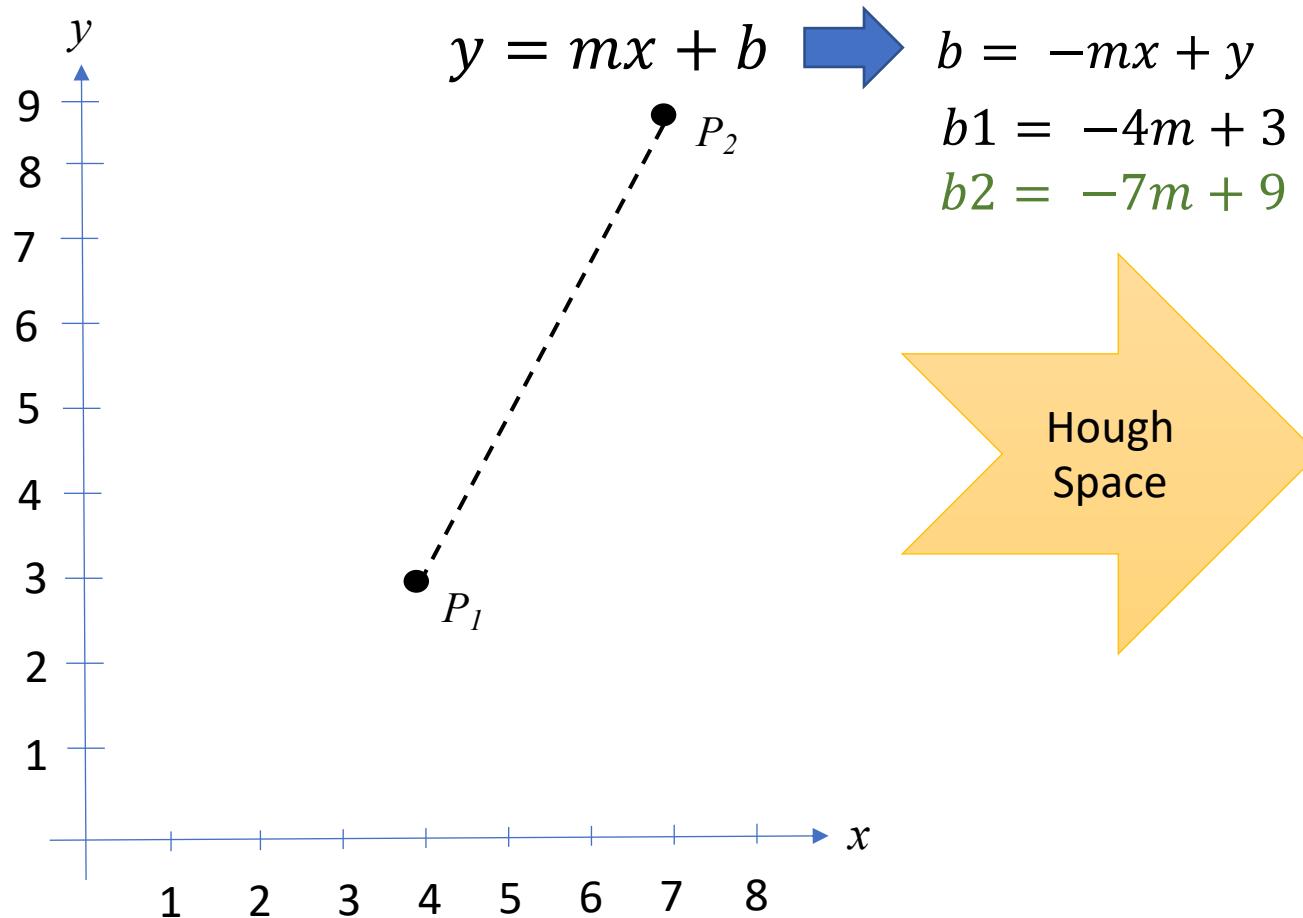
- $P_1(4,3)$ and $P_2(7,9)$

Equation of Line



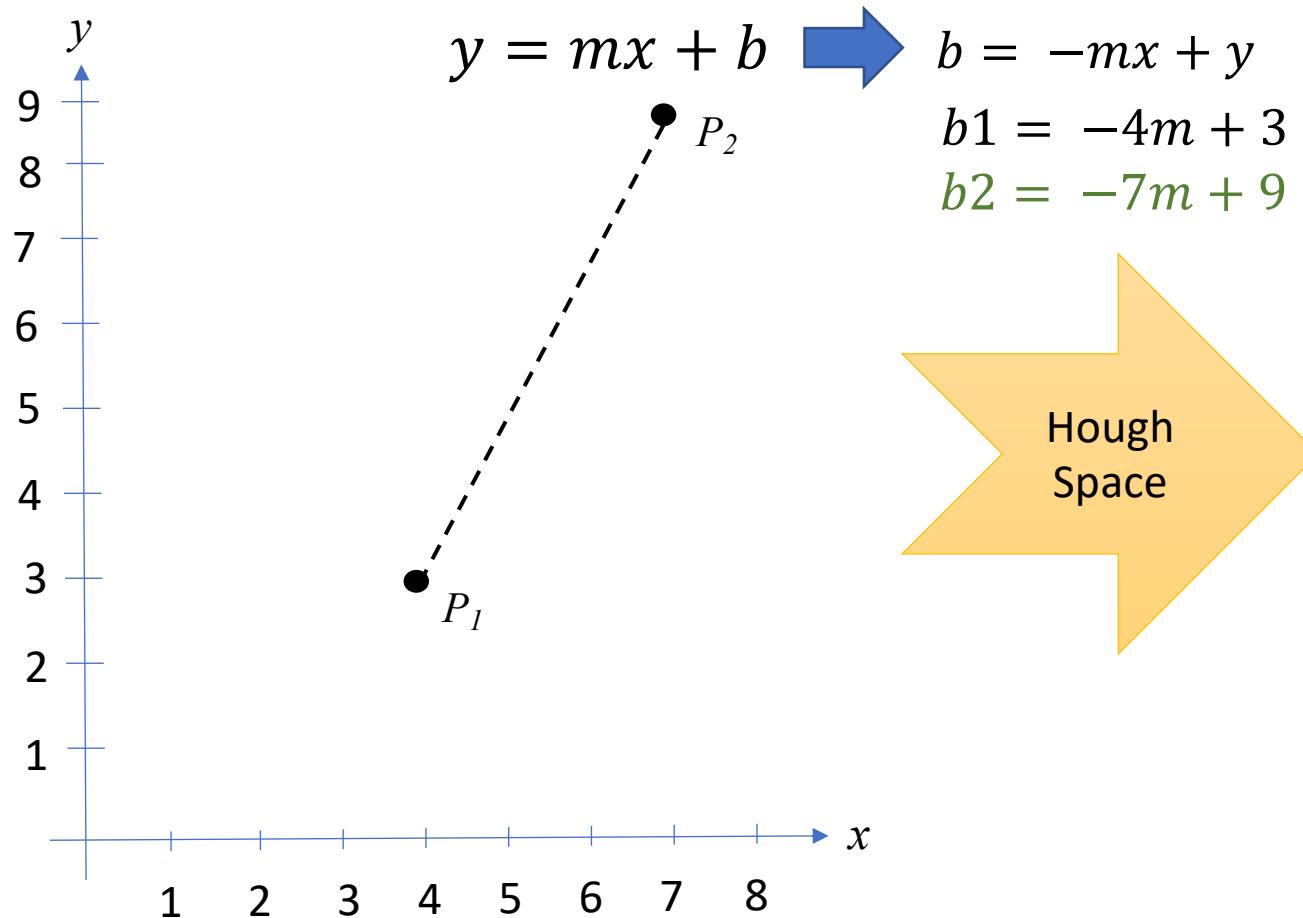
- $P_1(4,3)$ and $P_2(7,9)$

Equation of Line



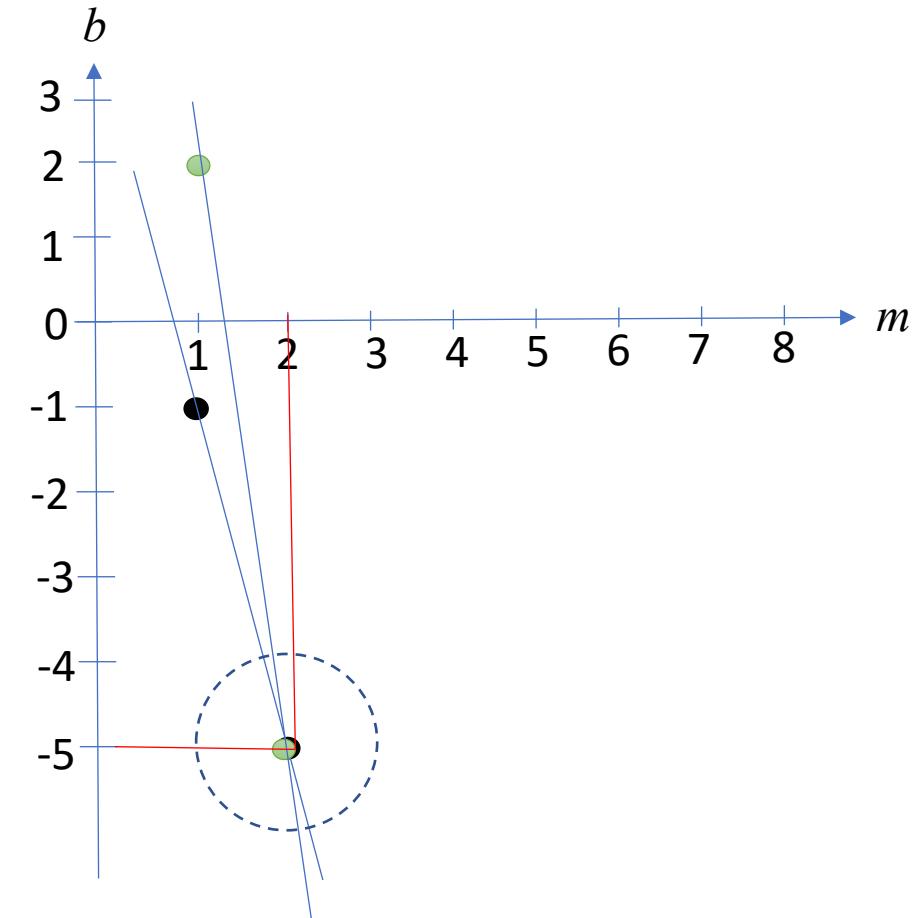
- $P_1(4,3)$ and $P_2(7,9)$

Equation of Line



- $P_1(4,3)$ and $P_2(7,9)$

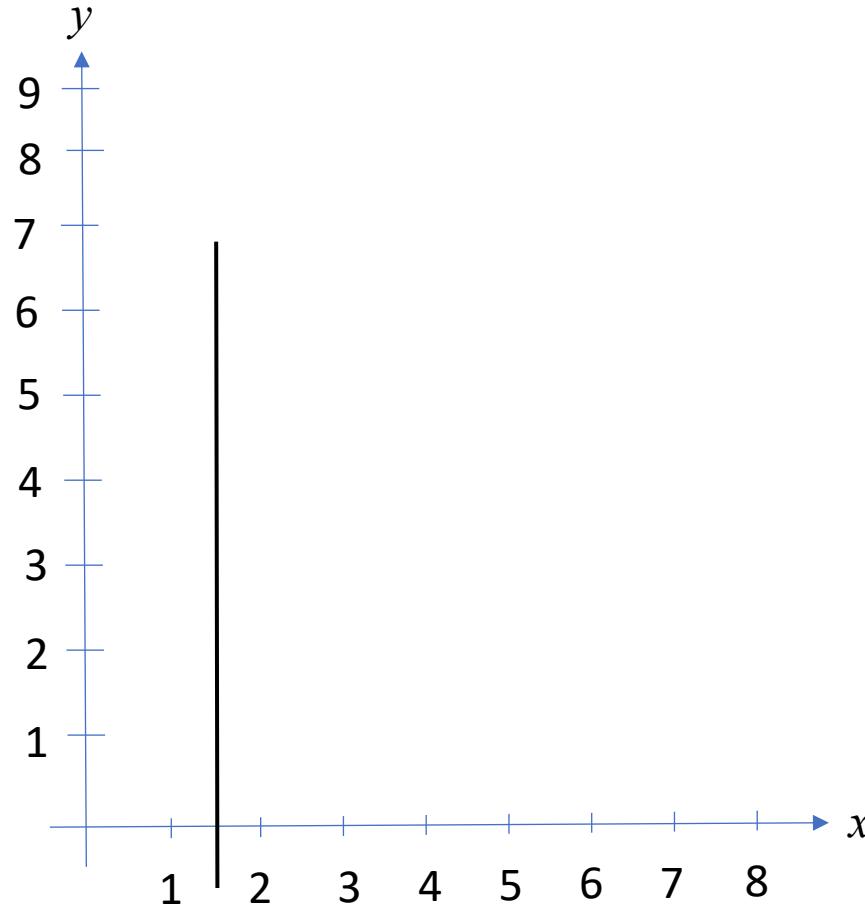
$m = 2$ and $b = -5$



Important!!

Special case: What is the equation for a vertical line?

$$x = 1.5$$



Hough Transform

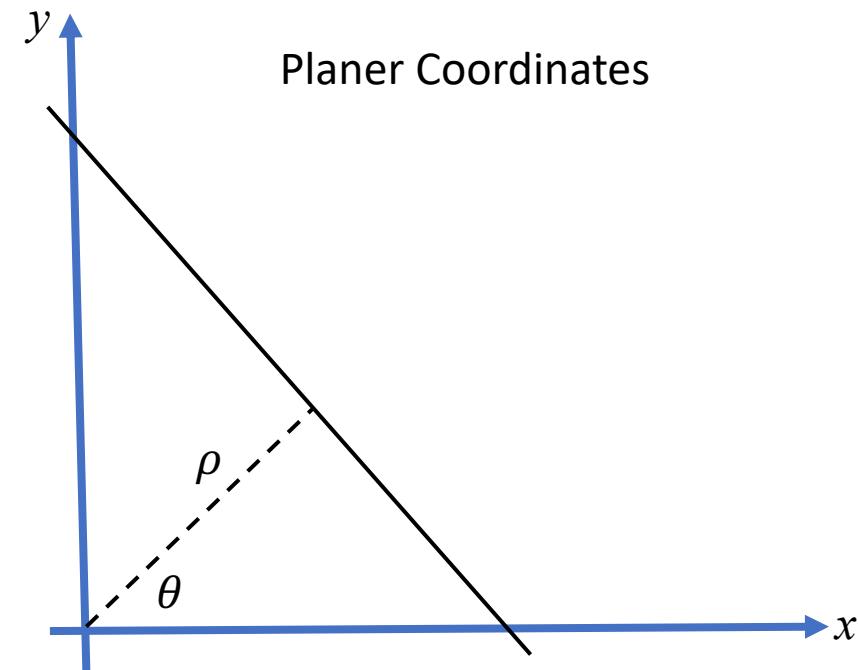
- How to represent a line?

$$y = mx + b$$

- The normal form of a straight line is

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

- ρ is perpendicular distance from origin to the line
- θ is the angle formed by this perpendicular line and horizontal axis

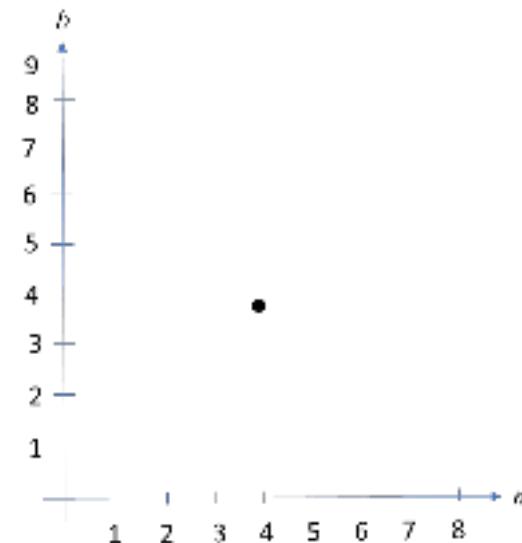


Hough Transform

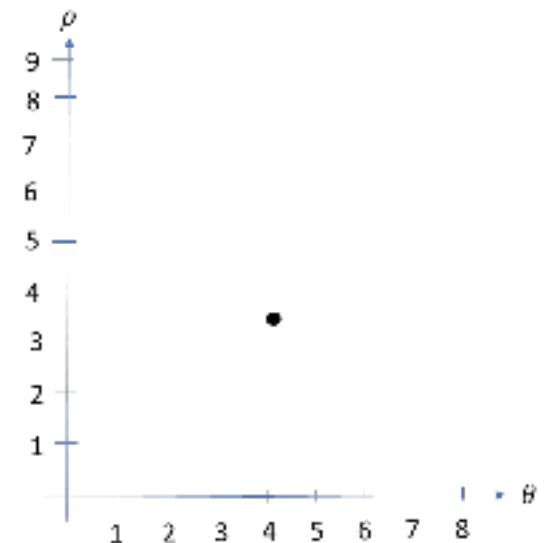
$$y = mx + b \text{ or } \rho = x \cos(\theta) + y \sin(\theta)$$



Image

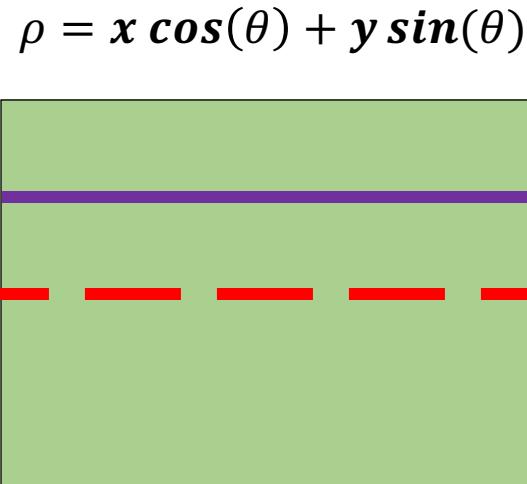


Hough Space



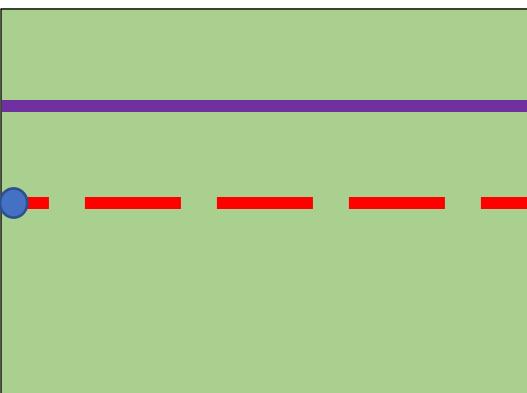
Calculate Hough Space

- Let's see how we can calculate Hough Space in polar coordinates



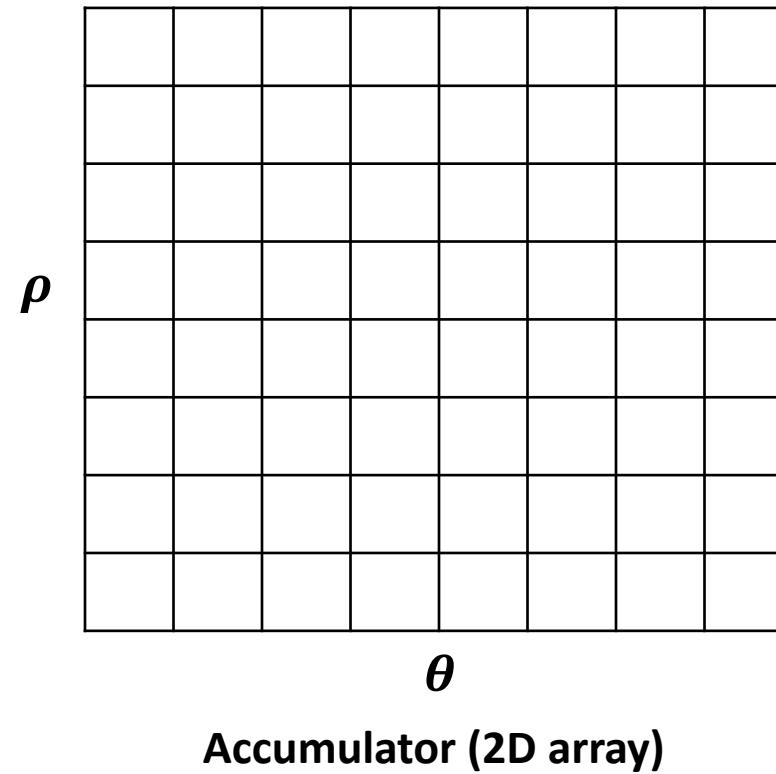
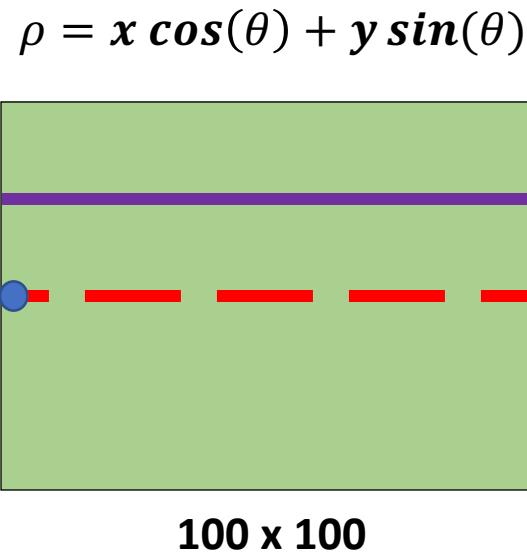
Calculate Hough Space

- Let's see how we can calculate Hough Space in polar coordinates



Calculate Hough Space

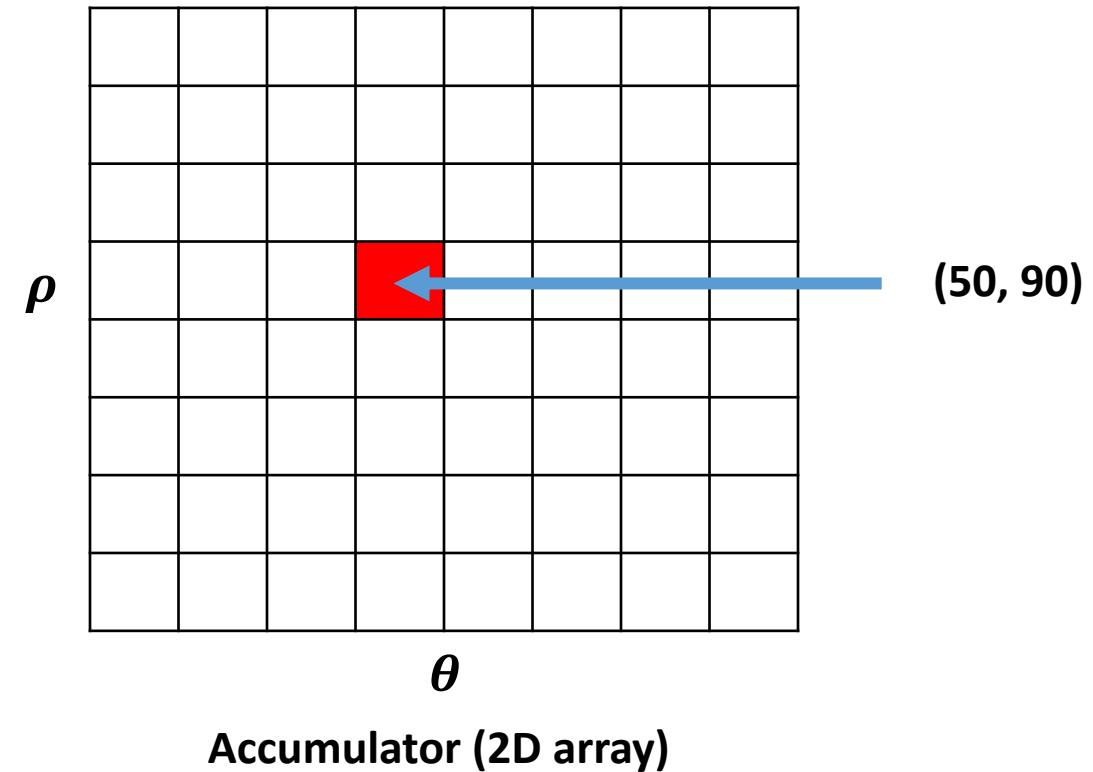
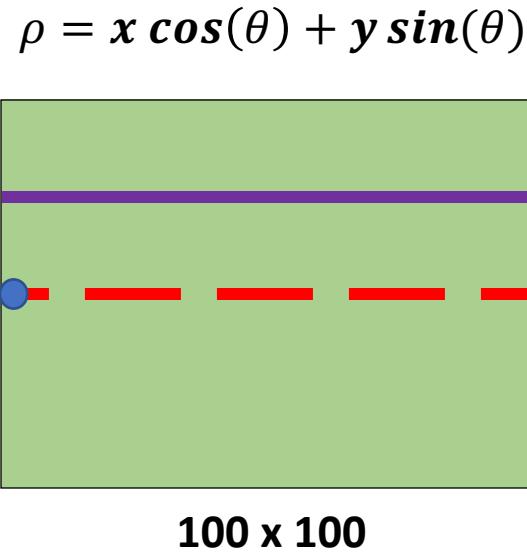
- Let's see how we can calculate Hough Space in polar coordinates



First create an accumulator (i.e, 2D array) to hold two values of the parameters rho and theta. Initialize it with zero.

Calculate Hough Space

- Let's see how we can calculate Hough Space in polar coordinates



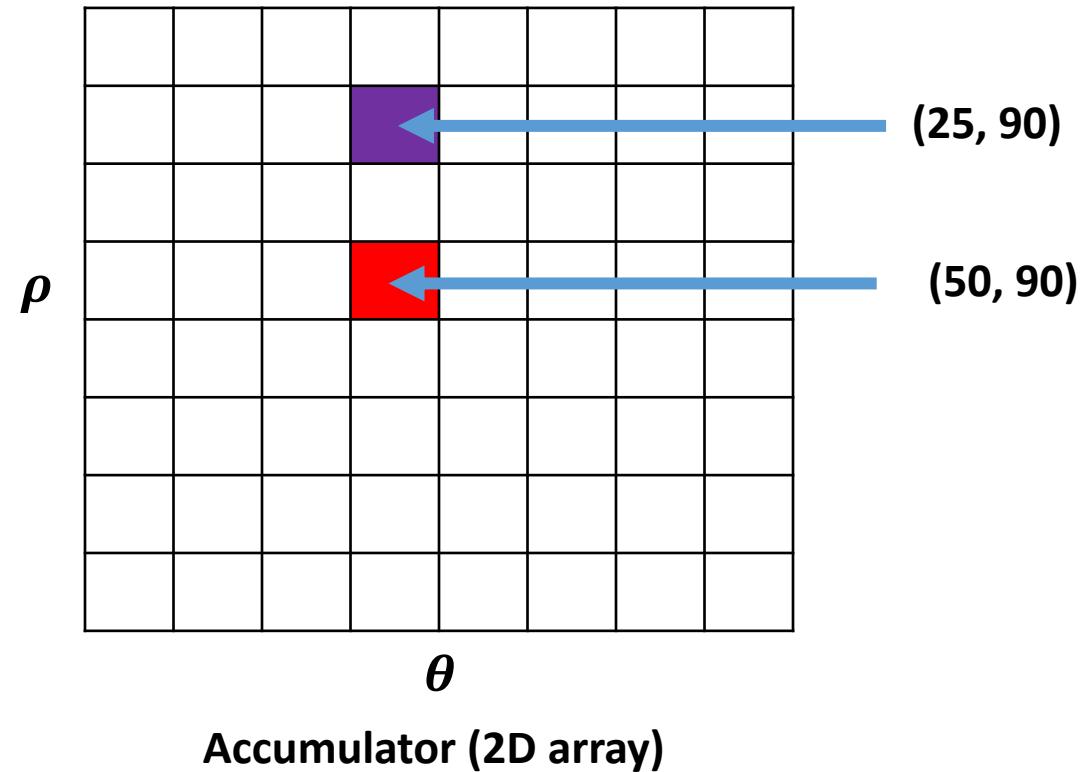
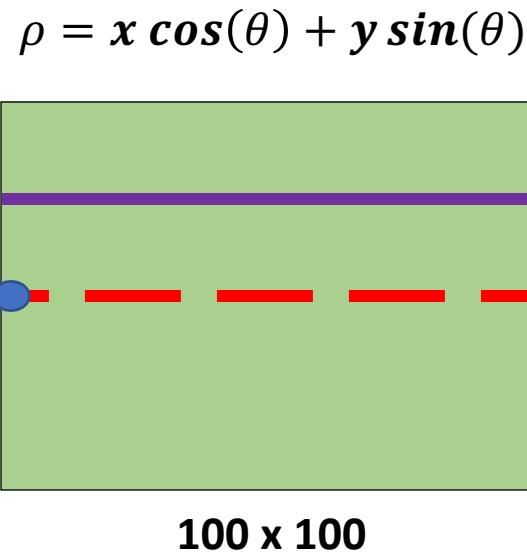
Calculate Hough Space (Repeat the process)

- Now take the second point on the line. Do the same.
- Increment the the values in the cells corresponding to (ρ, θ) you got. This time, the cell $(50,90) = 2$.
- What you actually do is voting the (ρ, θ) values.
- You continue this process for every point on the line. At each point, the cell $(50,90)$ will be incremented or voted up, while other cells may or may not be voted up.
- This way, at the end, the cell $(50,90)$ will have maximum votes.
- So if you search the accumulator for maximum votes, you get the value $(50,90)$ which says, there is a line in this image at distance 50 from origin and at angle 90 degrees

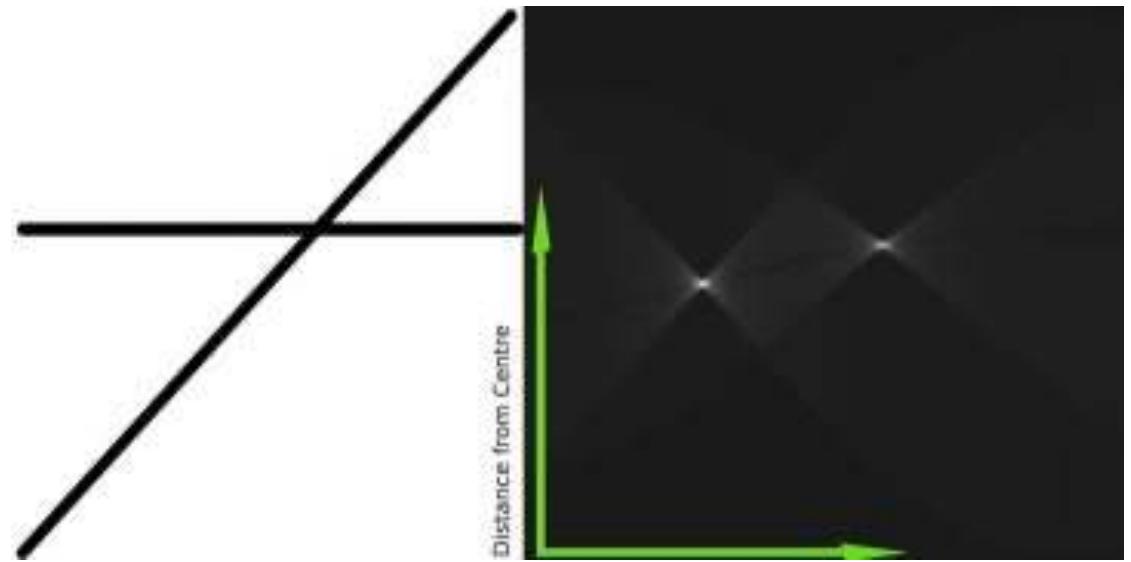
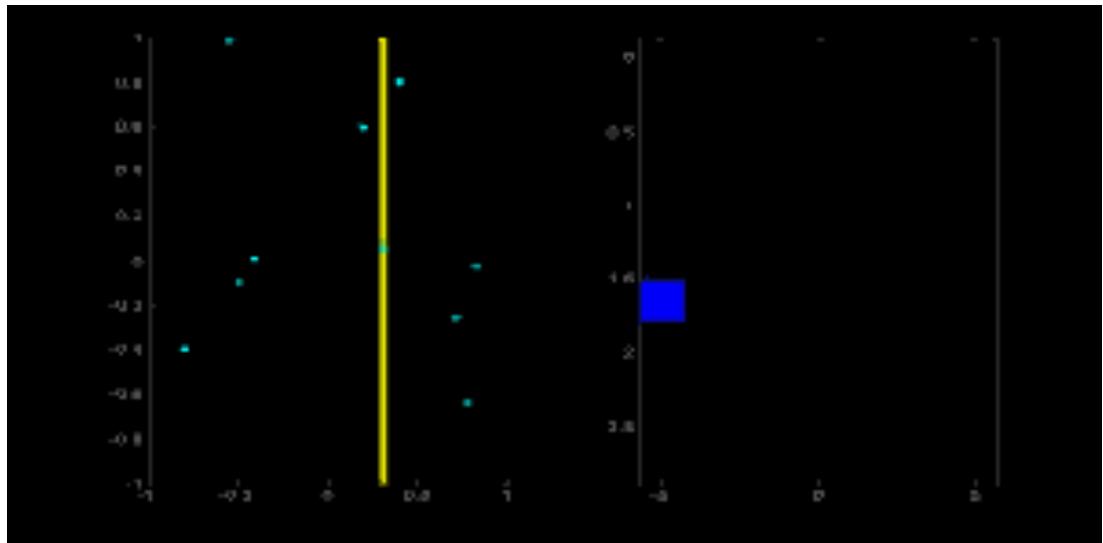
https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Calculate Hough Space

- Let's see how we can calculate Hough Space in polar coordinates



Hough Space Demo



<http://homepages.inf.ed.ac.uk/amos/hough.html>

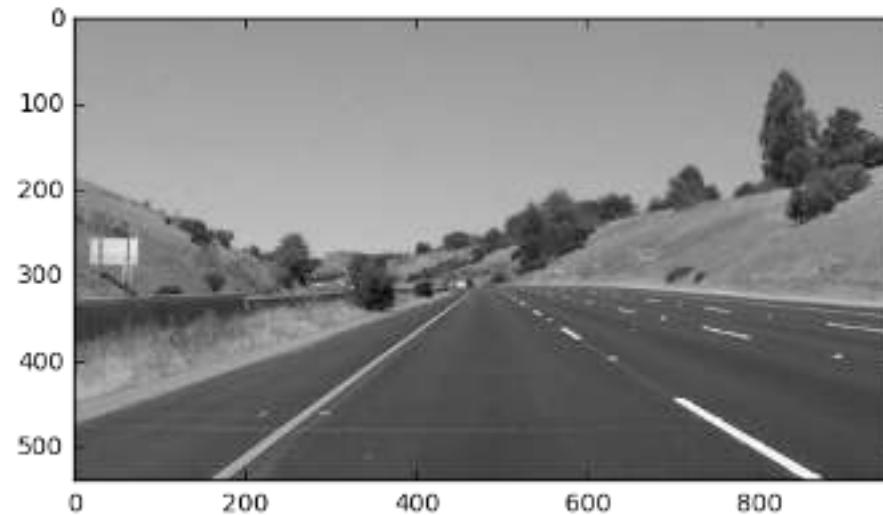
https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Implementation Details!!

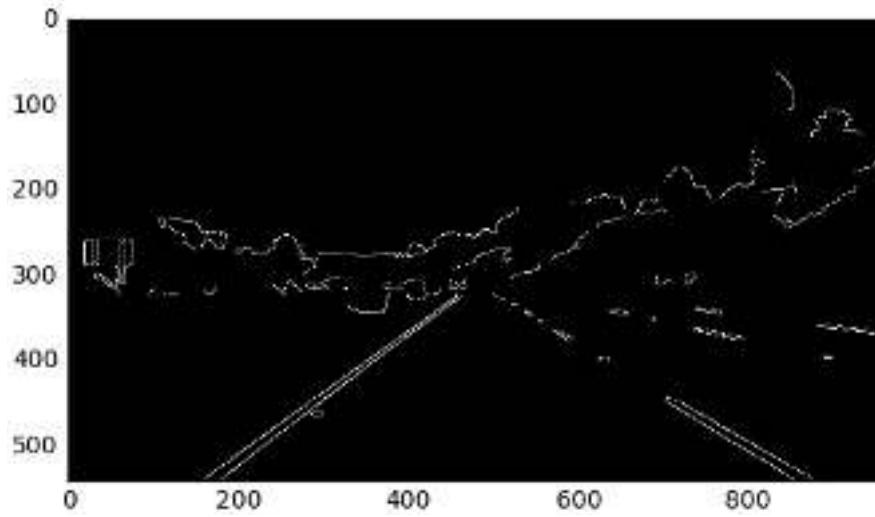
Original
Image



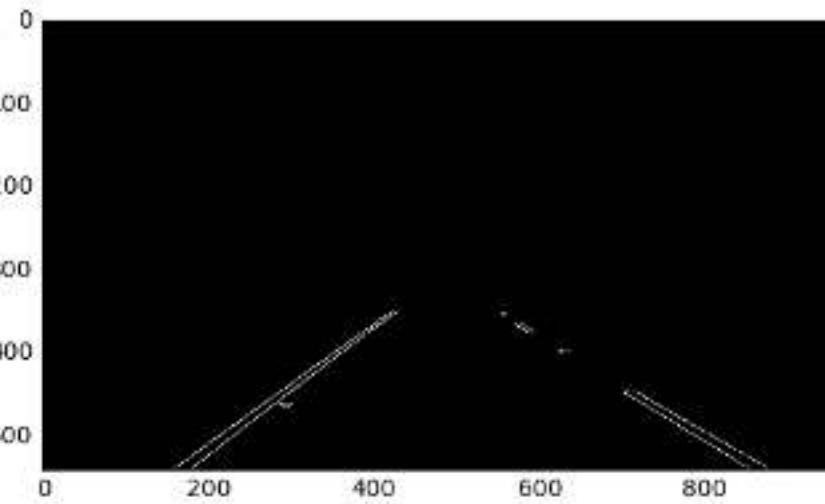
Grayscale
image



Canny Edge
Detection

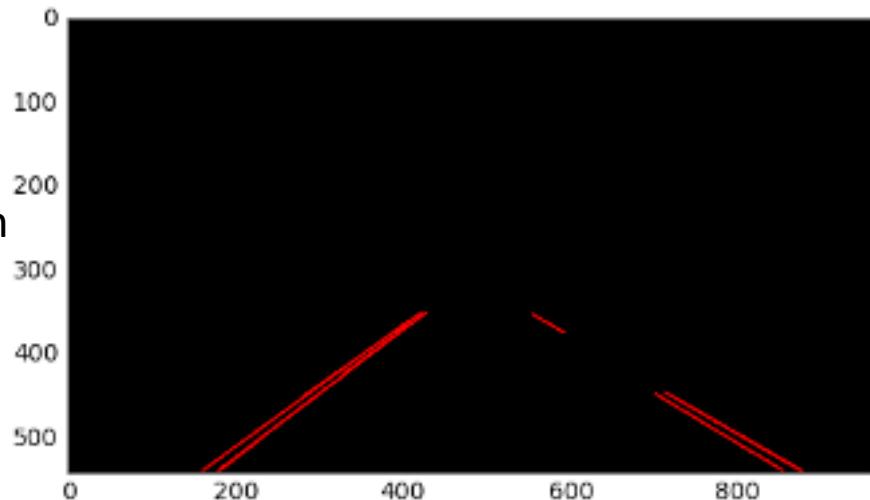


Mask out points that
are not in the region
of interest

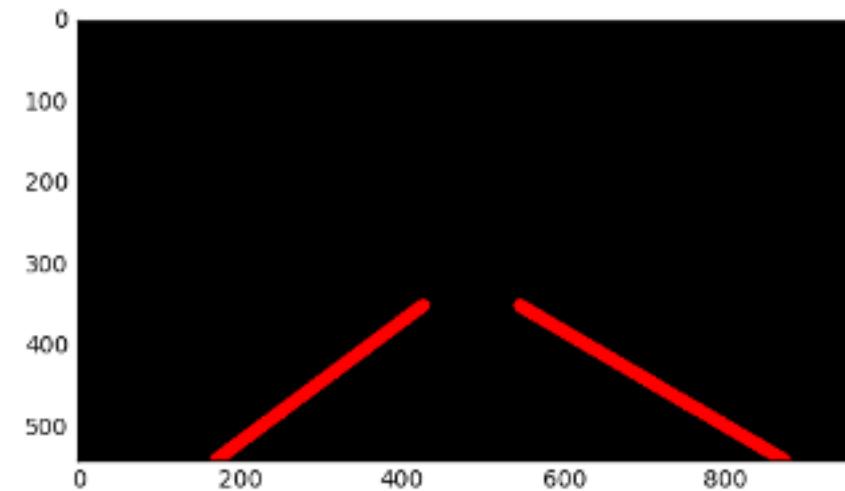


Final Output

Hough Transform output



Extrapolated lane lines



Final!! Line lanes overlaid on the input image

Final Output



Hough Circle Transform

Circle Hough Transform (CHT)

- The Hough Circle Transform works in a *roughly similar* way to the Hough Line Transform explained.
- In the **line detection case**, a line was defined by **two parameters** (ρ, θ) .
- In the **circle case**, we need **three parameters** to define a circle
$$(a, b, r)$$
- Where **(a, b)** is the **center** of the circle and **r** is **radius**



Circle Hough Transform (CHT)

- A circle with radius r and center (a, b) can be described with the parametric equations:

$$x = a + r \cos(\theta)$$

$$y = b + r \sin(\theta)$$

- x and y trace the perimeter of a circle

Challenge!!

- If an image contains many points, some of which fall on perimeters of circles, then the job of the search is to **find parameter triplets (a, b, R)** to describe each circle.
- The fact that the parameter space is 3D that makes a direct implementation of the **Hough technique more expensive** in computer memory and time.

Search with Known Radius

- If the circles in an image are of known radius r , then the search can be reduced to 2D.
- The objective is to find the (a, b) coordinates of the centers.

$$x = a + r \cos(\theta)$$

$$y = b + r \sin(\theta)$$

Search with Fixed Radius

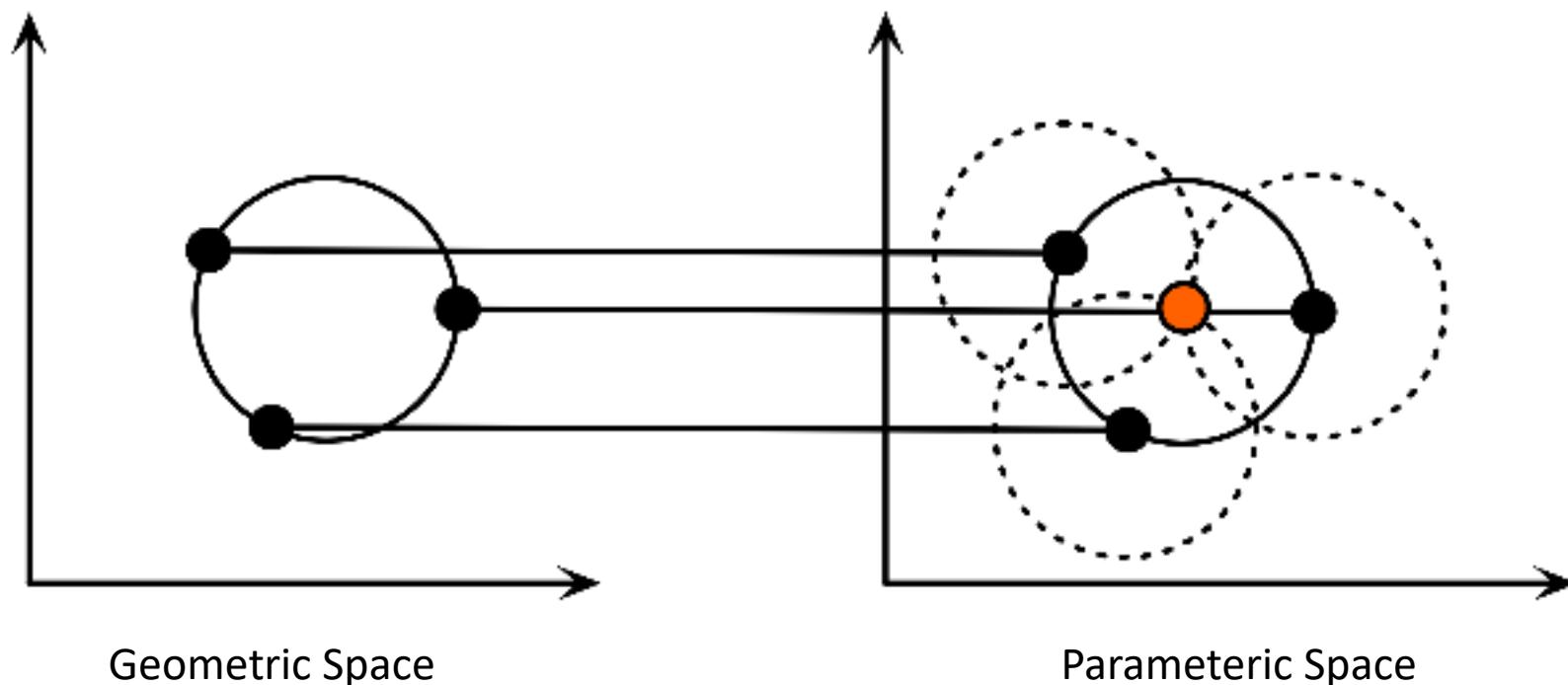
$$x = a + r \cos(\theta)$$

$$y = b + r \sin(\theta)$$

- The locus of (a, b) points in the *parameter space* fall on a circle of radius r centered at (x, y) .
- The true center point will be common to all parameter circles, and can be found with a Hough accumulation array.

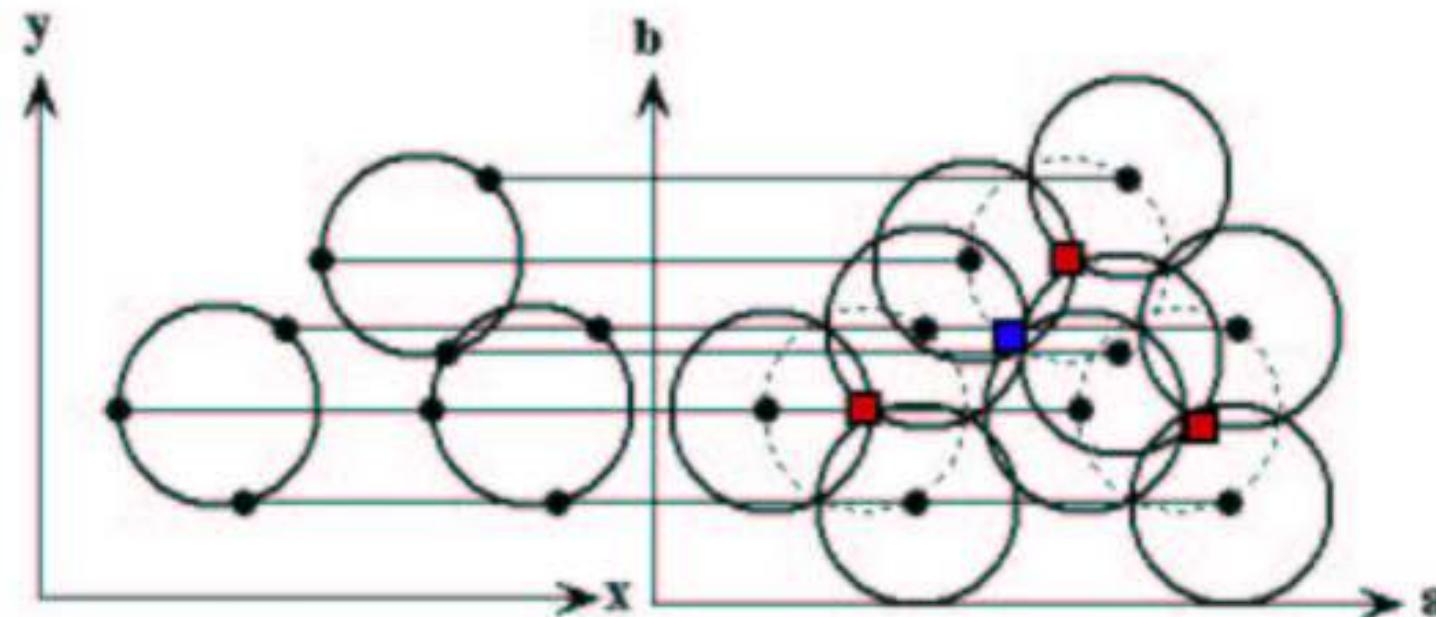
Search with Fixed Radius

- Each point in geometric space generates a circle in parameter space.
- The circles in parameter space **intersect at the (a, b)** that is the center in geometric space.



Multiple Circles with known Radius

- Multiple circles with the same radius can be found with the **same technique**.
- The **center points** are represented as **red cells** in the parameter space drawing.
- Overlap of circles can **cause spurious centers** to also be found, such as at **the blue cell**.



Search with Unknown Radius

- The search for circles with unknown radius can be conducted by using a three dimensional accumulation matrix.

Advantages

- Conceptually simple
- Easy implementation
- Handles missing and occluded data very gracefully
- Can be adapted to many types of forms, not just lines

Disadvantages

- Computationally complex for objects with many parameters
- Looks for only one single type of object
- Can be “fooled” by “apparent lines”
- Co-linear line segments cannot be separated

Reading

- A lot of resources available over the internet
- Open CV documentation

Q & A

Semantic Segmentation

Instructor: Dr. Muhammad Fahim

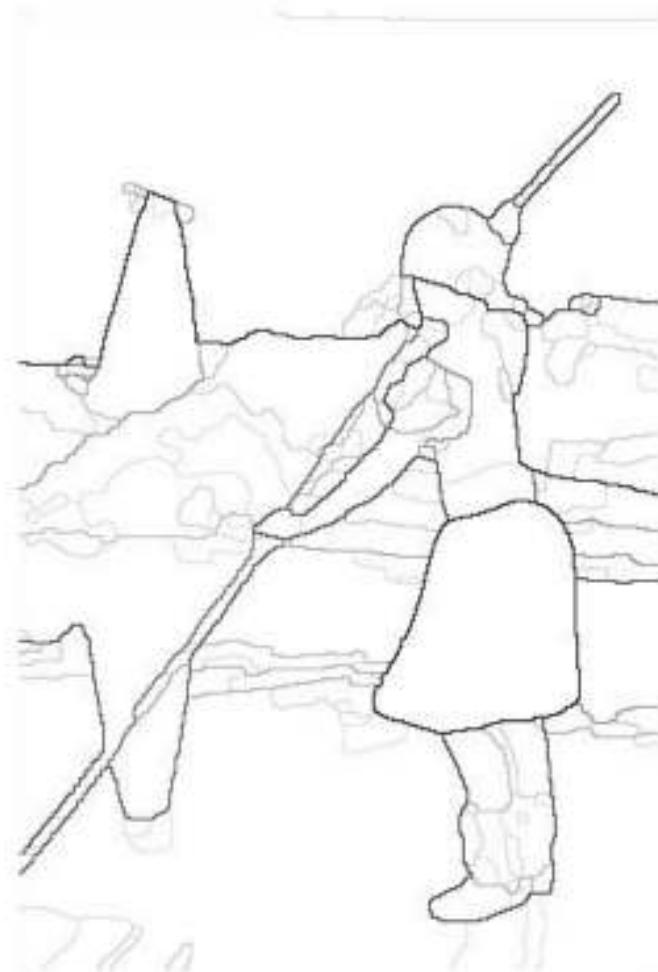
Source of the material

- This lecture is based on the following resources
 - Semantic Segmentation CVPR Paper 2015.
 - Jeremy Jordan: <https://www.jeremyjordan.me/semantic-segmentation/>
 - Other resources found over the internet to make the concept clear.

Contents

- What is semantic segmentation?
- Why we need this?
- Fully Convolutional Neural Network for this task
- Architecture Details
- Obtained Results
- Evaluation Metrics
- Other Architectures for Semantic Segmentation
- Semantic Segmentation vs Instance Segmentation
- Summary

What is segmentation?



Source: Tingwu Wang

What is segmentation?

- Most of the time, we need to "process the image"

- Filters
- Gradient information
- Color information etc.

- That's not quite so human.
- What if we want to understand the image?



Arbelaez, Pablo, et al. "Contour detection and hierarchical image segmentation." IEEE transactions on pattern analysis and machine intelligence, 2011.

Semantic Segmentation

"What's in this image, and where in the image is it located?"

Semantic Segmentation

The goal is to **predict class labels for each pixel** in the image

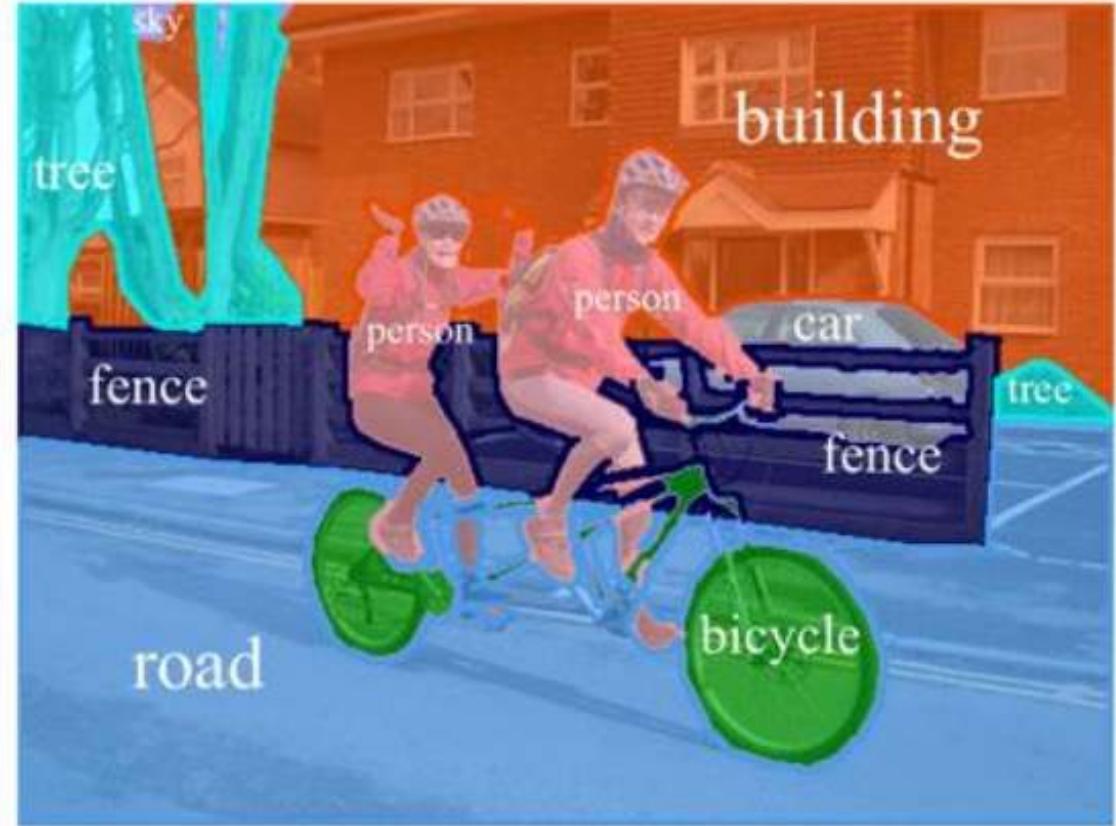
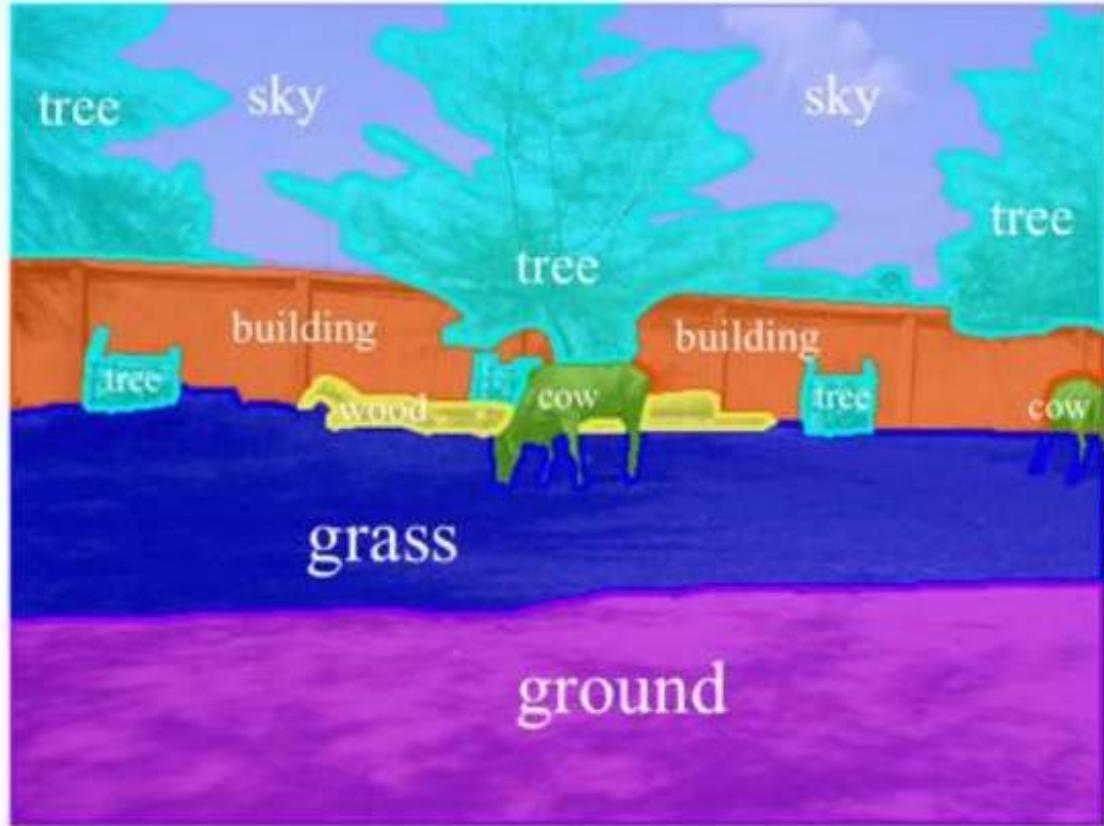


predict →



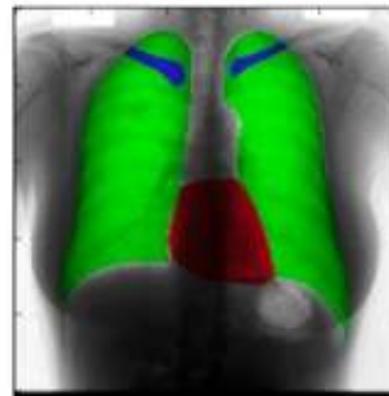
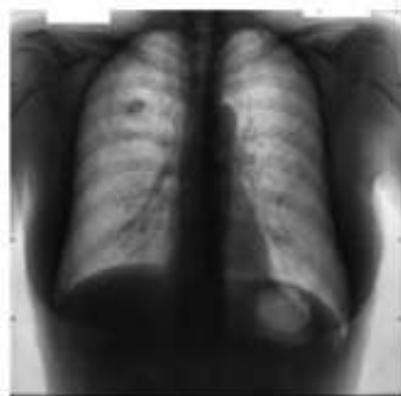
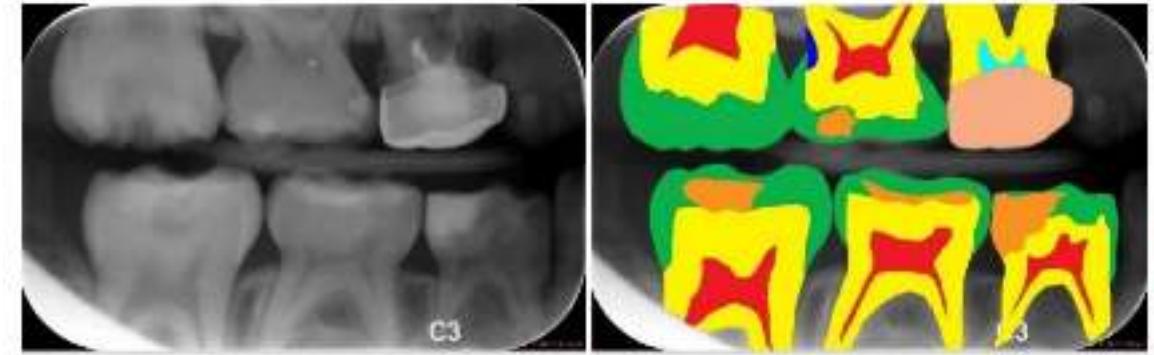
Person
Bicycle
Background

Semantic Segmentation



Why semantic segmentation?

- Robot vision and seen understanding
- Autonomous driving
- Medical imaging segmentation
- Many more...



Input Image

Segmented Image



Watch this: <https://www.youtube.com/watch?v=ATlcEDSPWXY>

Semantic Representation



Input

segmented

- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
5	5	3	3	3	3	3	3	3	3	1	1	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4

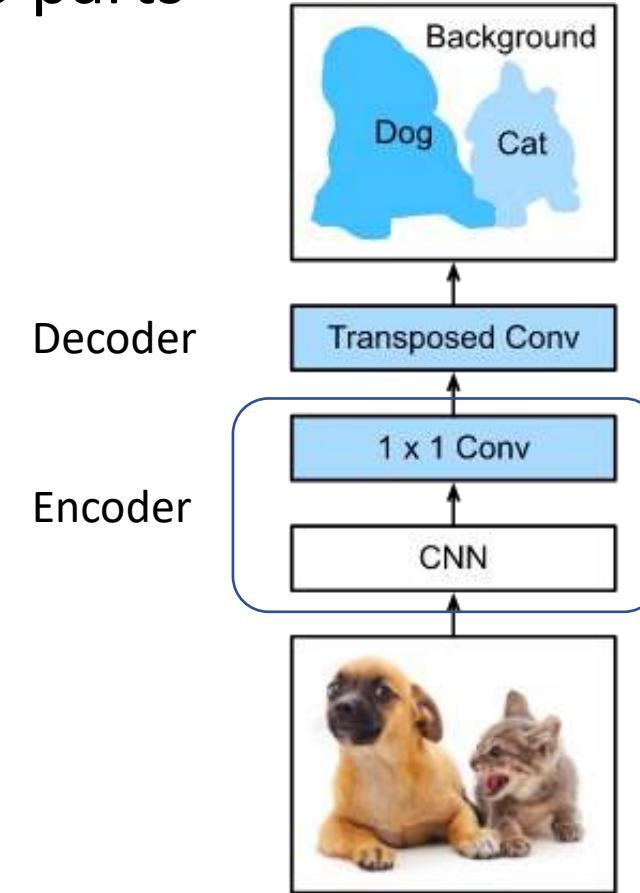
Semantic Labels

Fully Convolutional Networks for Semantic Segmentation

Long Jonathan, Evan Shelhamer, and Trevor Darrell.
UC Berkeley, *CVPR* (2015)

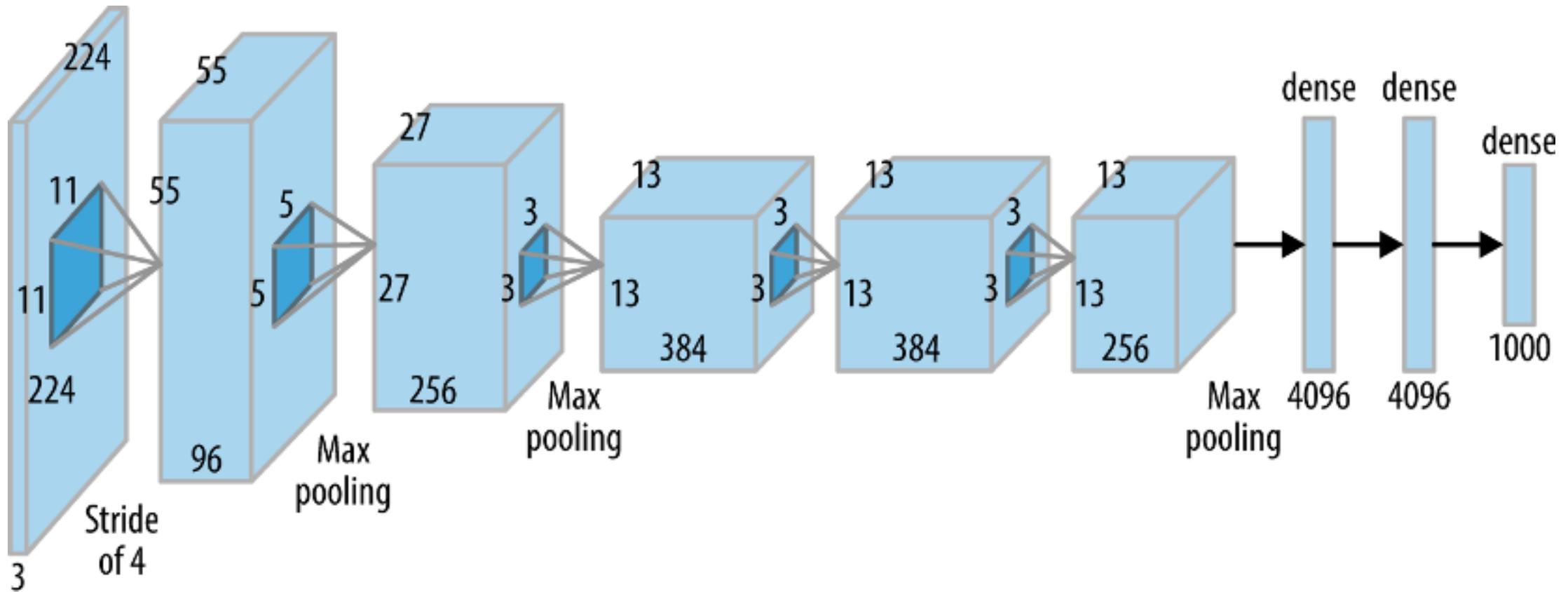
Fully Convolutional Networks for Semantic Segmentation

- Fully Convolutional Networks (FCN) has two parts
 1. Encoder Module (AlexNet/VGG-16)
 2. Decoder Module



FCN – Encoder Module

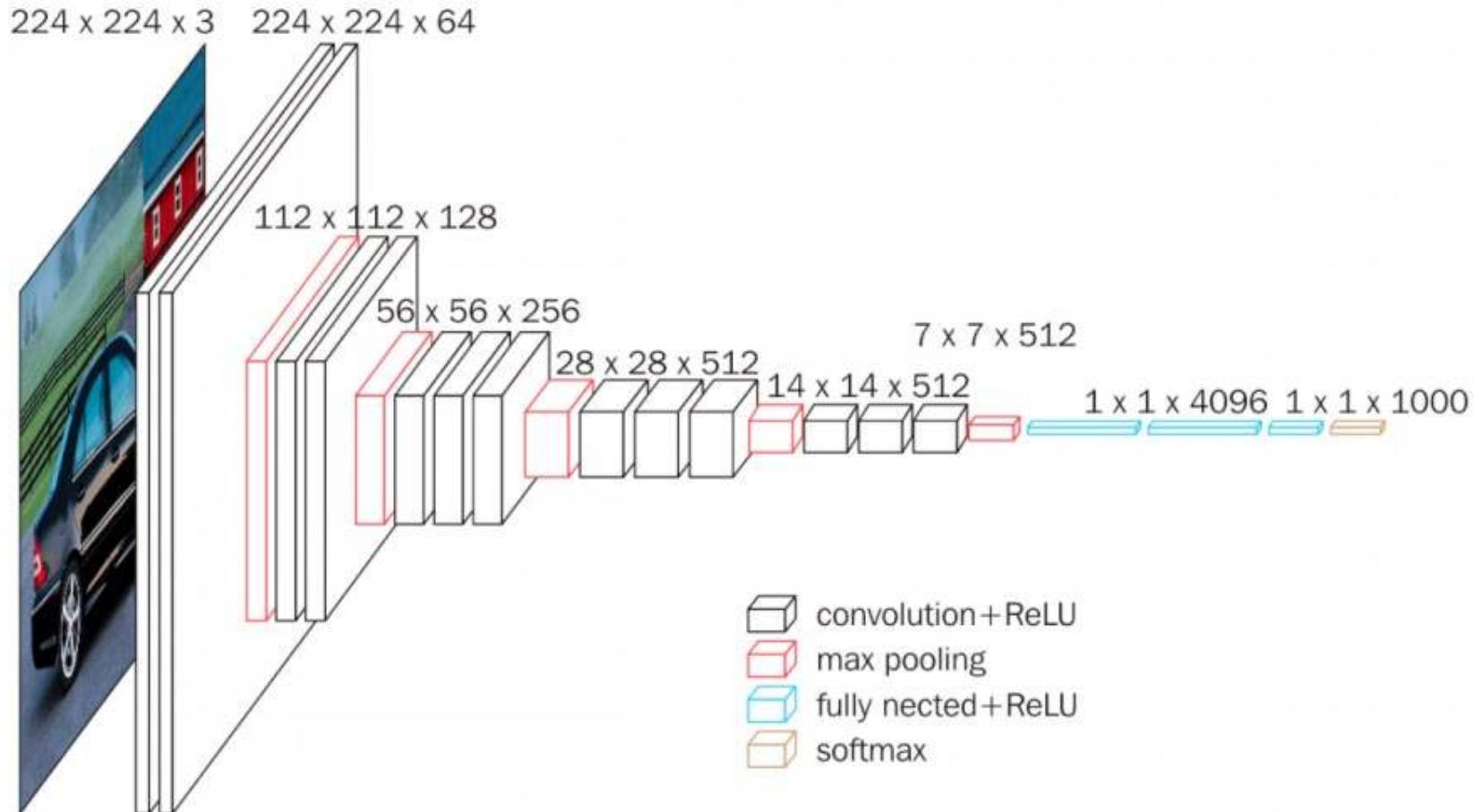
- AlexNet Architecture



The dense layer was designed for classification task

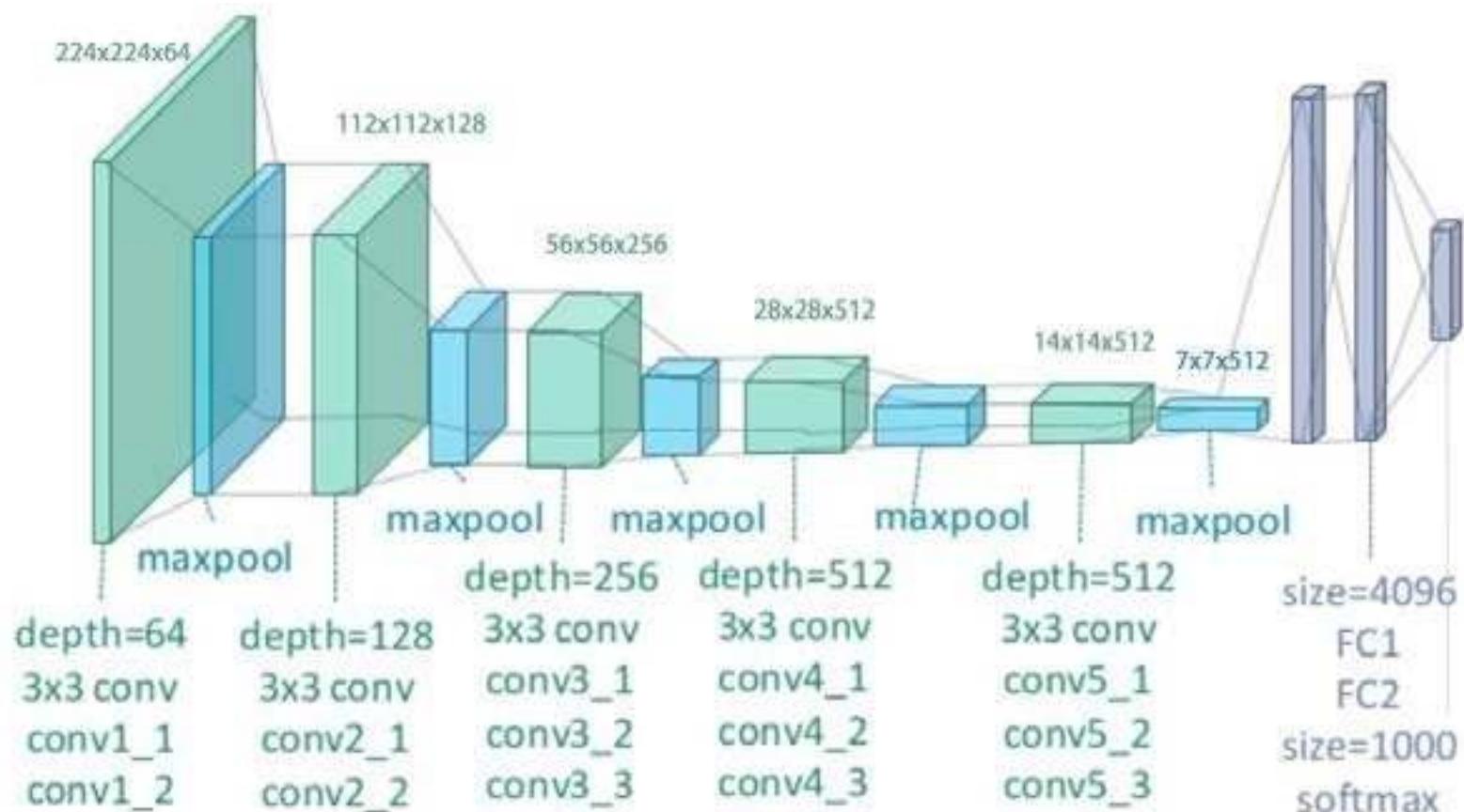
FCN – Encoder Module

VGG-16 Architecture



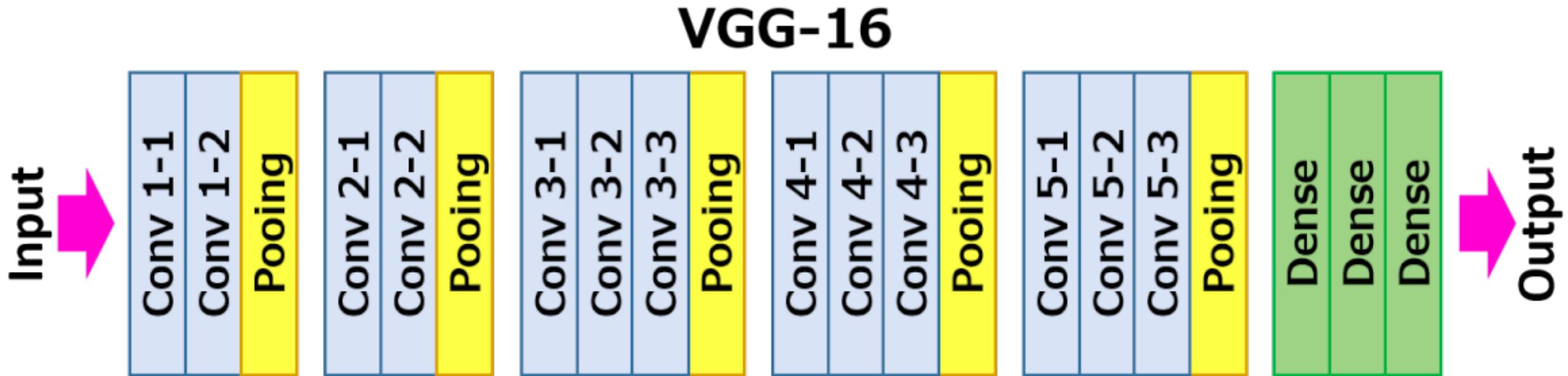
FCN – Encoder Module

VGG-16 Architecture (Another view)

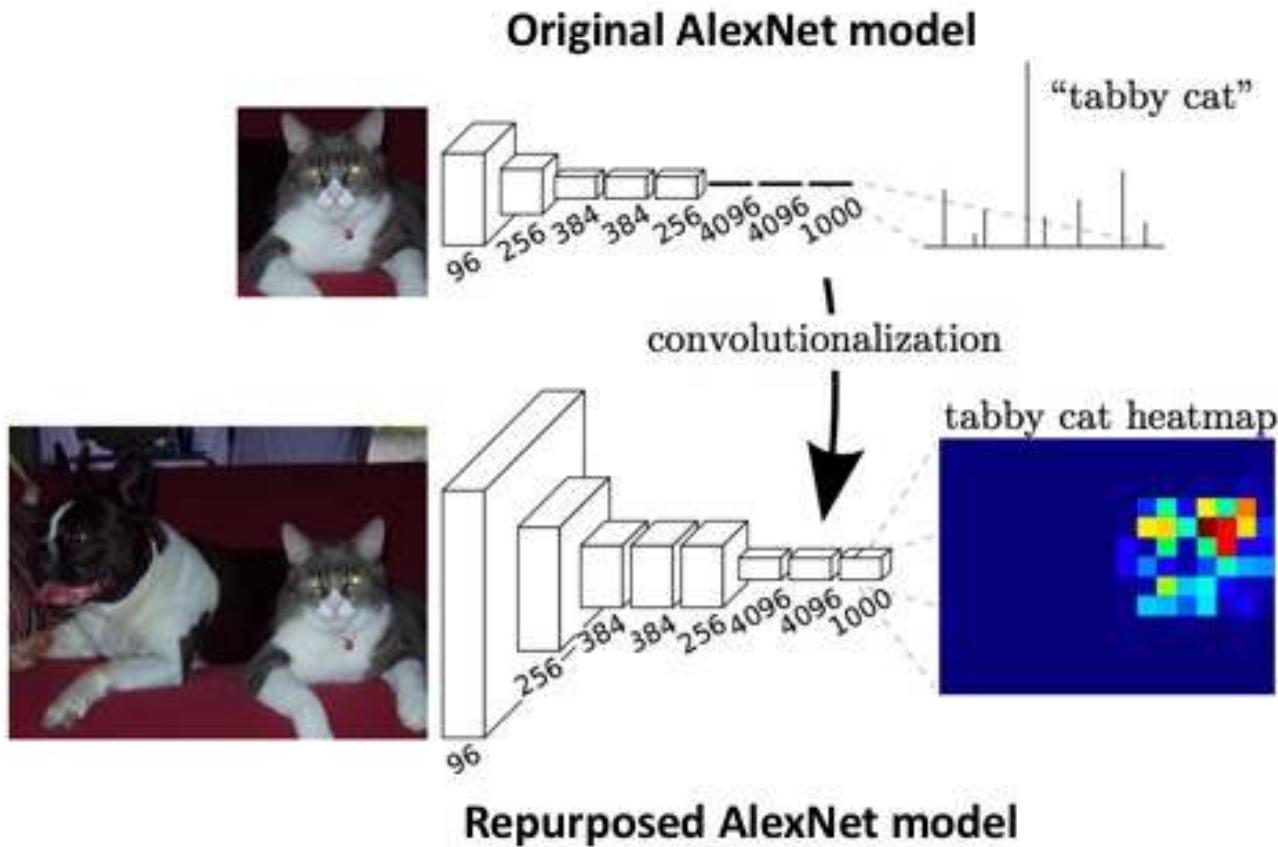


FCN – Encoder Module

VGG-16 Architecture



FCN – Encoder Module

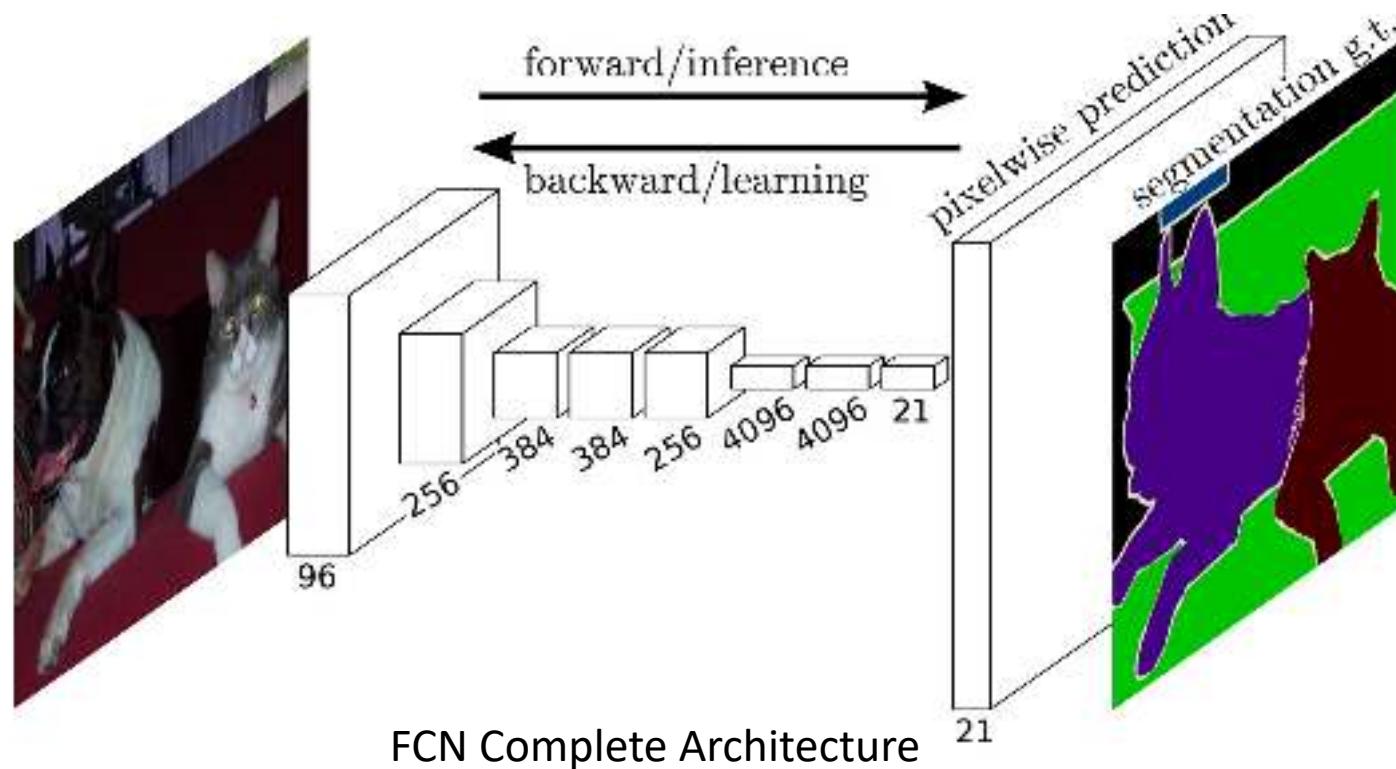


The encoder produces a **coarse** feature map which is then refined by the decoder module.

- The last fully connected layers are replaced by a 1x1 convolution.

FCN – Decoder Module

- A decoder module with transpose convolutional layers to upsample the coarse feature maps into a **full-resolution segmentation map**

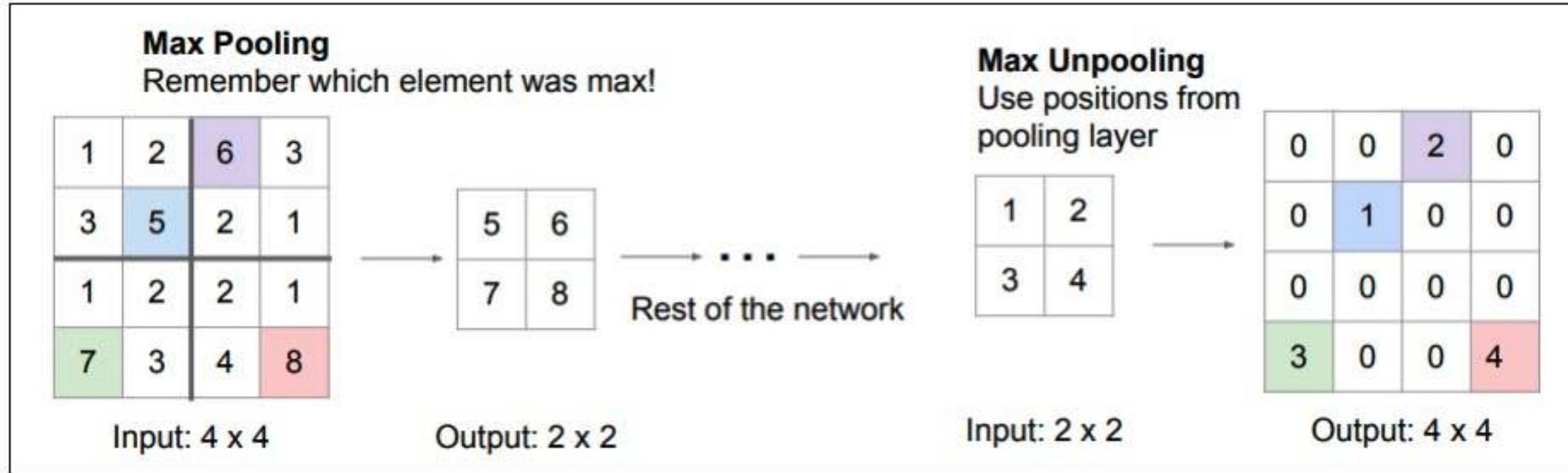


- Furthermore, to fully recover the **fine-grained spatial information** lost in the pooling or downsampling layers, we often use **skip connections**. (More details a little later)

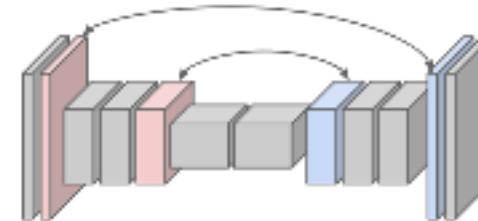
Decoder Module – Upsampling Methods

- There are a few different approaches that we can use to *upsample* the *resolution* of a feature map.
- **Unpooling**
 - Unpooling operations *upsample the resolution* by distributing a *single value* into a *higher resolution*.
- NOTE:
 - **Pooling:** Pooling operations *downsample the resolution* by summarizing a local area with a *single value* (i.e., average or max pooling or many others...)

Decoder Module – Upsampling Methods



Corresponding pairs of downsampling and upsampling layers



Source: Fei-Fei Li (Stanford)

Decoder Module – Upsampling Methods

Nearest Neighbor

1	2
3	4



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

Input: 2 x 2

Output: 4 x 4

Decoder Module – Upsampling Methods

“Bed of Nails”

1	2
3	4



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

Input: 2 x 2

Output: 4 x 4

New Concept!! Learnable upsampling.

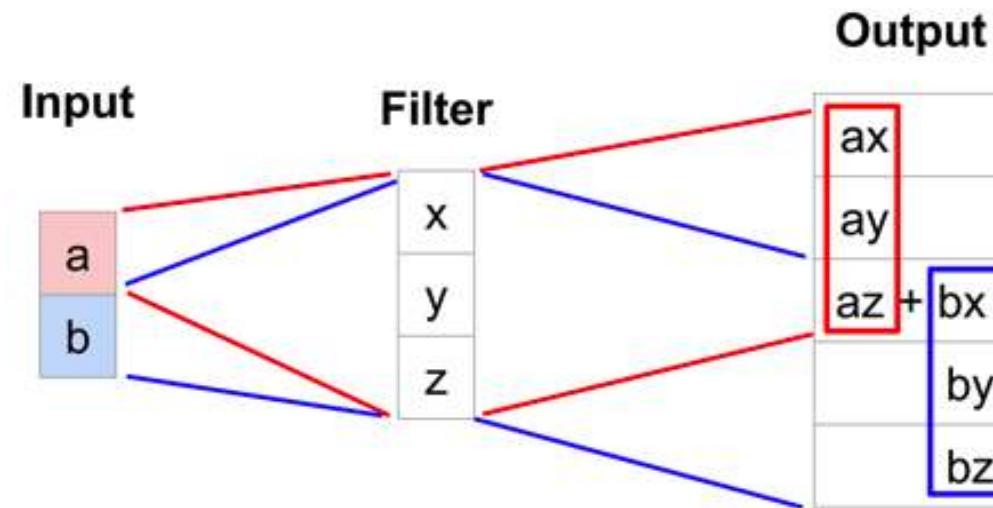
Source: Fei-Fei Li (Stanford)

Decoder Module – Learnable Upsampling Method

Transpose convolutions allow us to develop a learned upsampling.

Decoder Module – Upsampling Methods

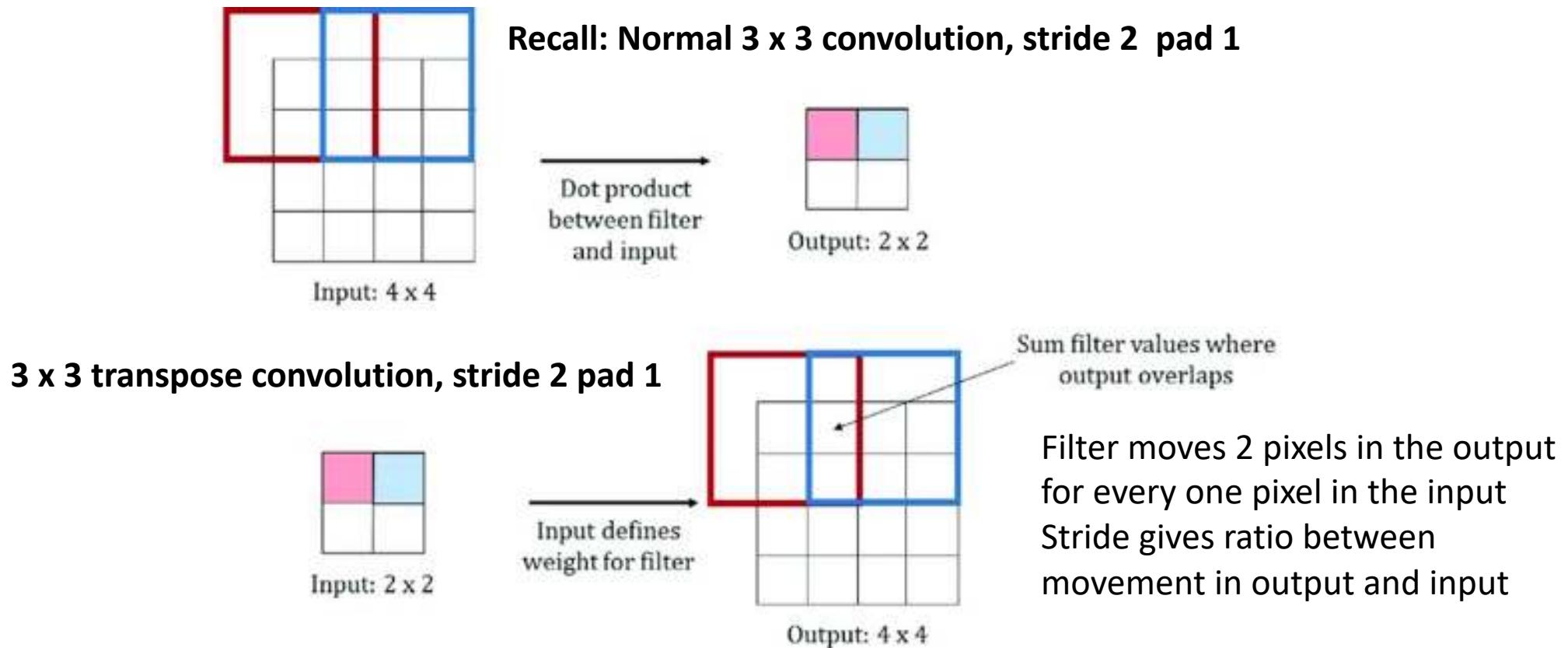
- **Transpose convolution**
 - We take a single value from the **low-resolution feature map** and multiply all of the **weights in our filter** by this value, projecting those weighted values into the **output feature map**.
- A simplified **1D example of upsampling** through a transpose operation.



Source: Fei-Fei Li (Stanford)

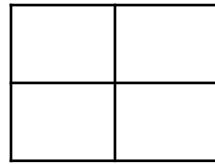
Decoder Module – Learnable Upsampling Method

- 2D Transpose Convolution

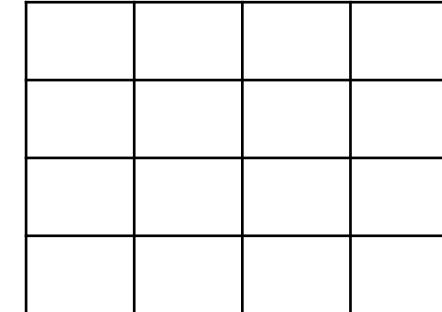


Source: Fei-Fei Li (Stanford)

2D Learnable Upsampling Example



Input: 2 x 2



Output: 4 x 4

Input	Kernel	Output							
		=			+			+	

FCN – Decoder Module

- The decoder module struggles to produce fine-grained segmentations

Ground truth target



Predicted segmentation



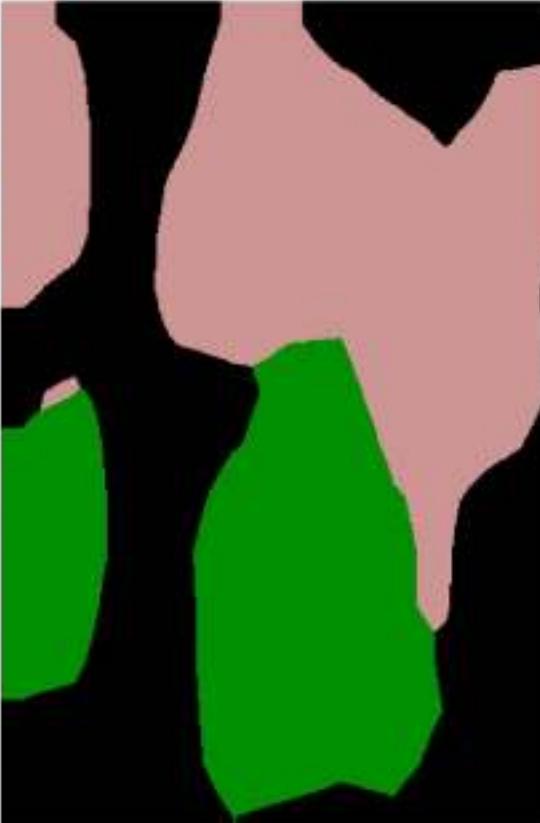
FCN – Decoder Module

The paper's authors comment on this **struggle**:

*Semantic segmentation faces an inherent tension between semantics and location: global information resolves **what** while local information resolves **where**... Combining fine layers and coarse layers lets the model make local predictions that respect global structure. – Long et al.*

Solution: Adding Skip Connections

Skip Connections



without skip

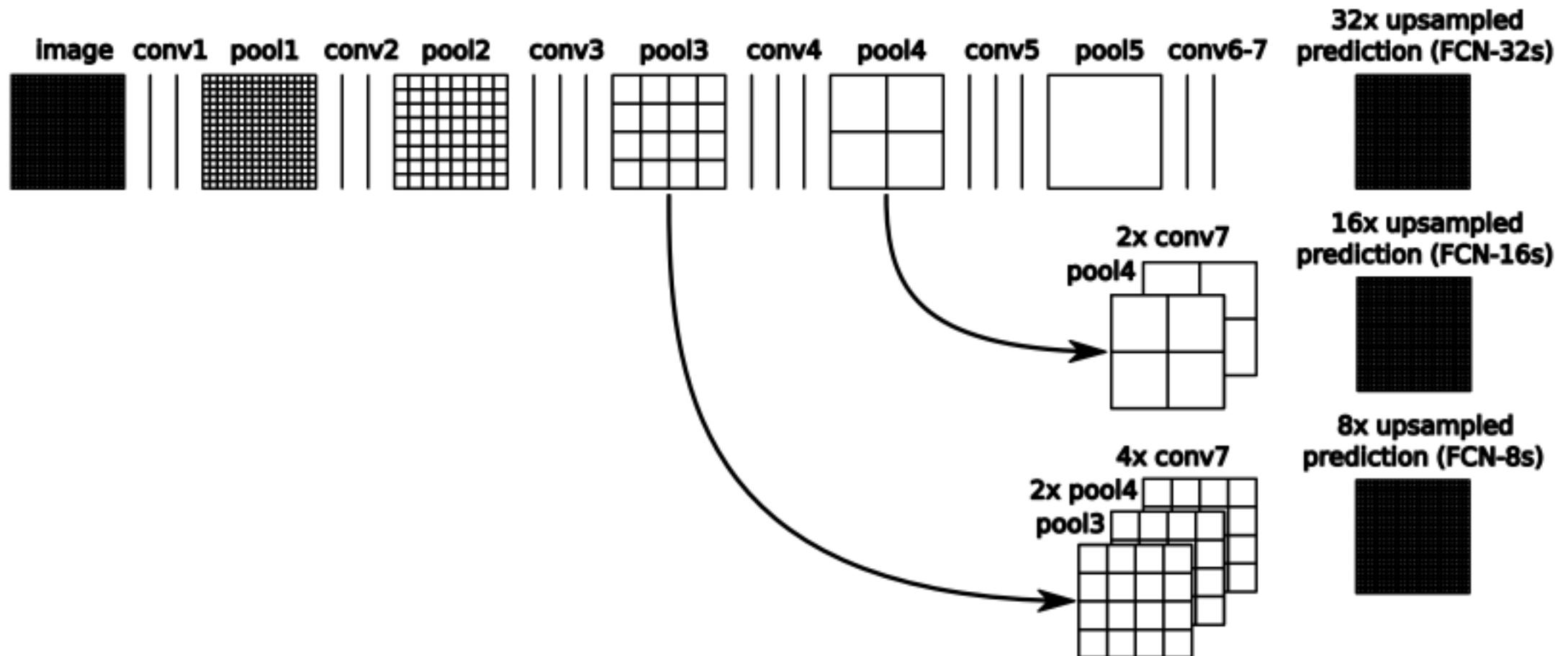


with skip

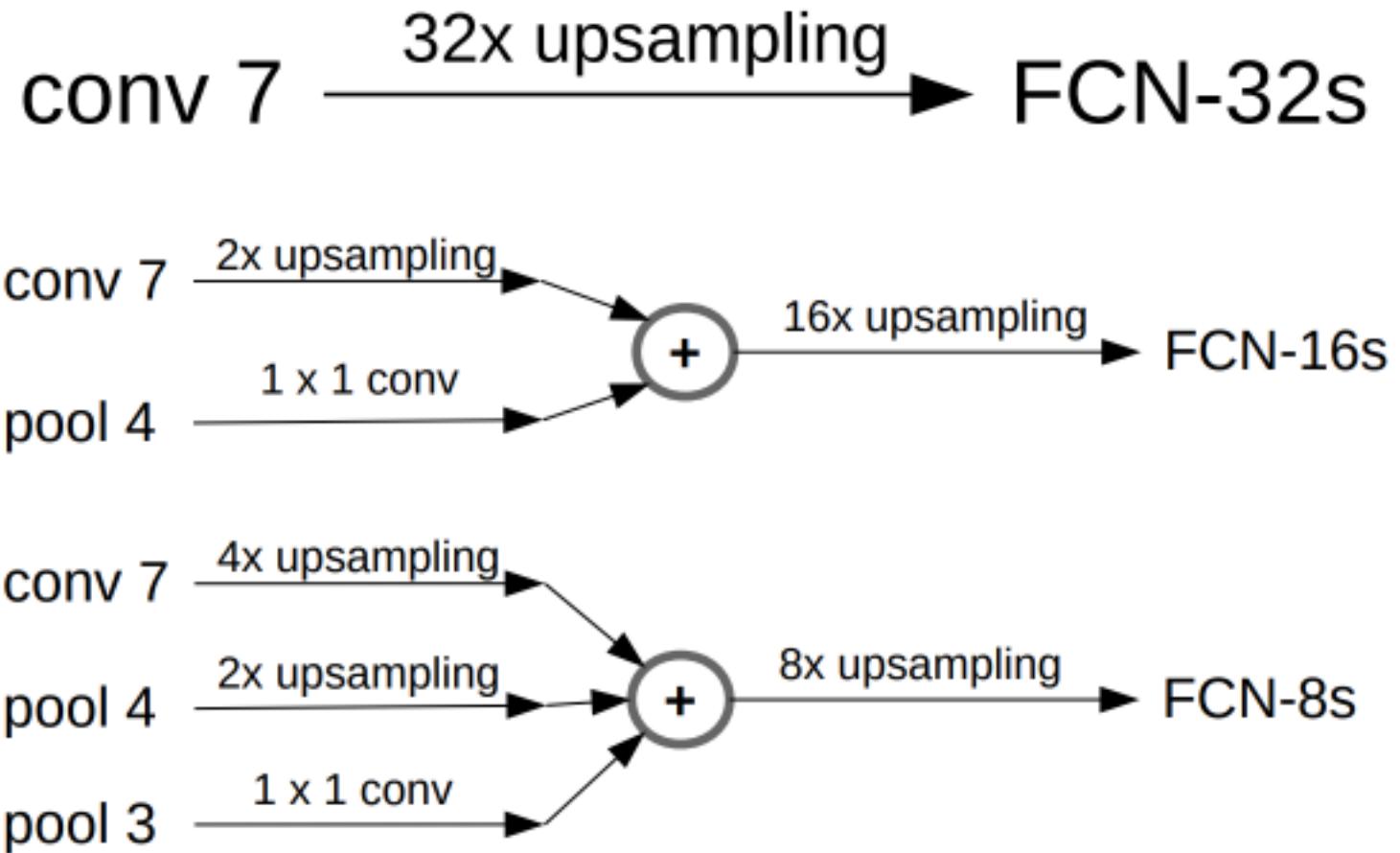
Adding Skip Connections

- A skip connection is a connection that bypasses at least one layer.
- It is often used to transfer local information by concatenating or summing feature maps from the **downsampling path** with feature maps from the **upsampling path**.
- Merging features from various resolution levels helps combining context information with spatial information.

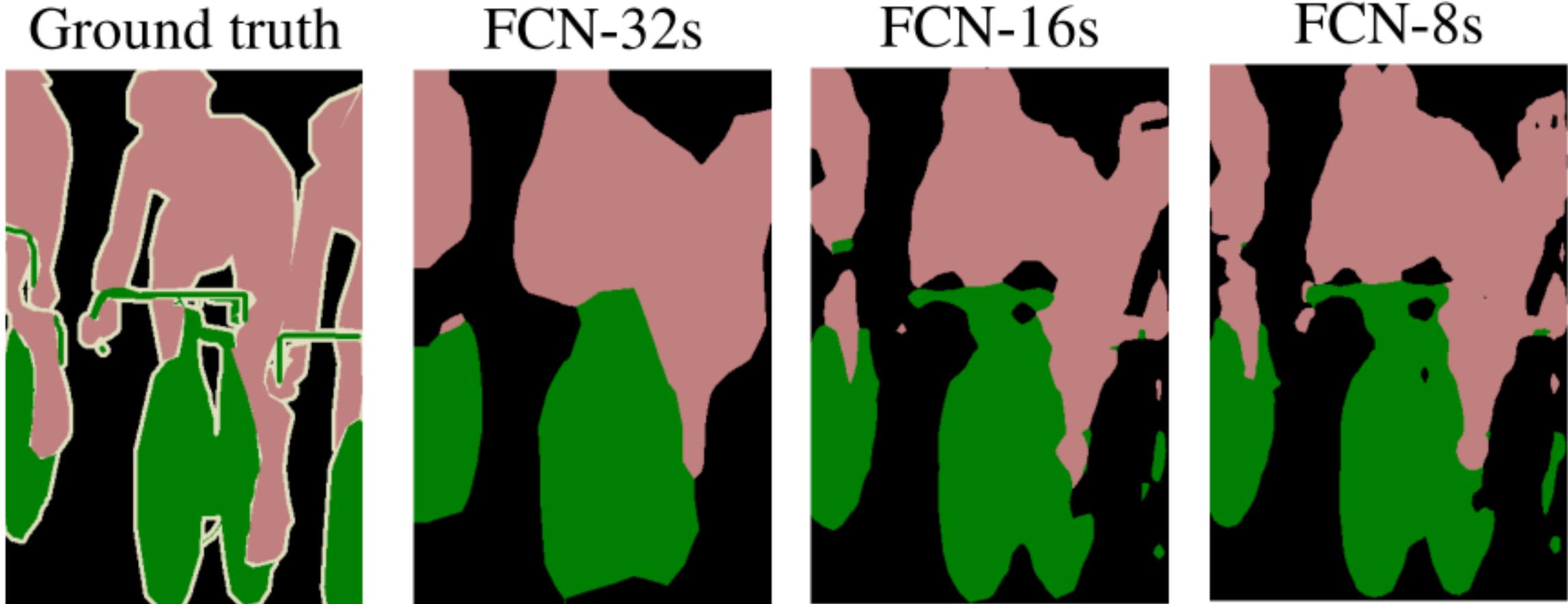
Combine Coarse and Fine Feature Maps



Combine Coarse and Fine Feature Maps

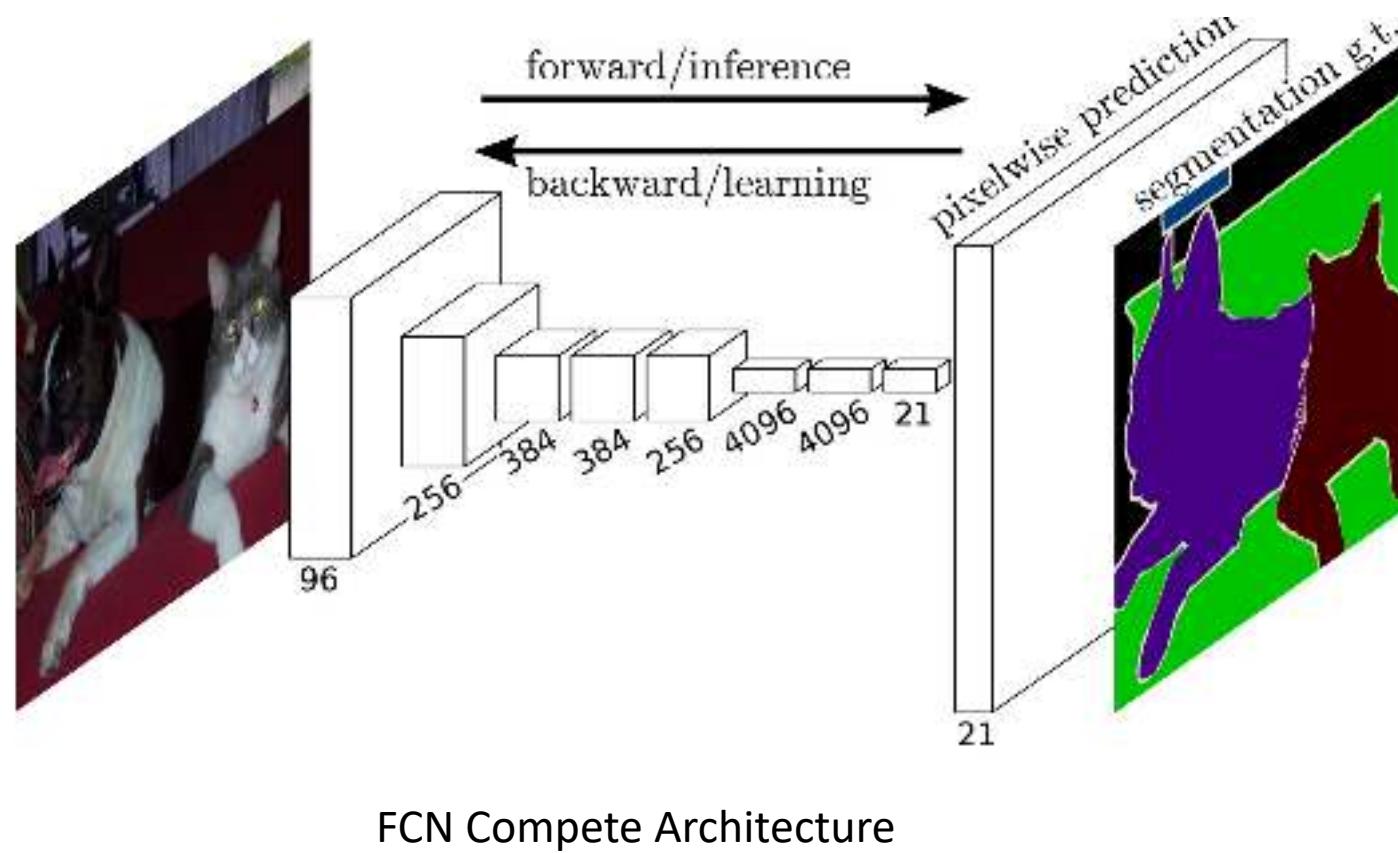


Combine Coarse and Fine Feature Maps



FCN Architecture (Revisit)

Where we started...

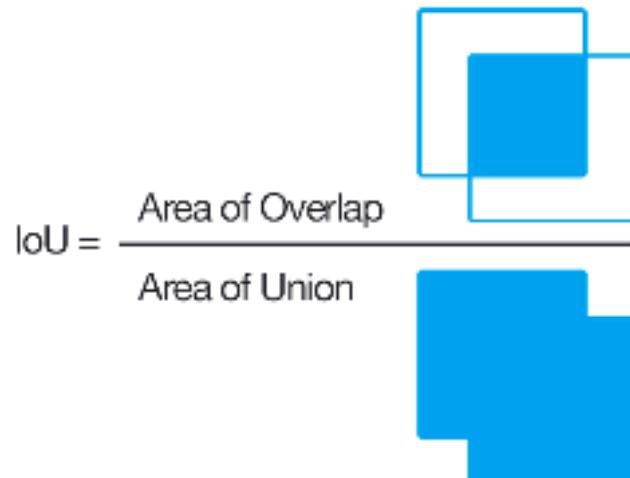


Performance Measures – Metrics

- **Per pixel accuracy**

$$acc(P, GT) = \frac{|\text{pixels correctly predicted}|}{|\text{total nb of pixels}|}$$

- **Intersection over Union (Jaccard index)**



$$jacc(P(\text{class}), GT(\text{class})) = \frac{|P(\text{class}) \cap GT(\text{class})|}{|P(\text{class}) \cup GT(\text{class})|}$$

Results

Dataset: Visual Object Classes Challenge 2011 (VOC2011)

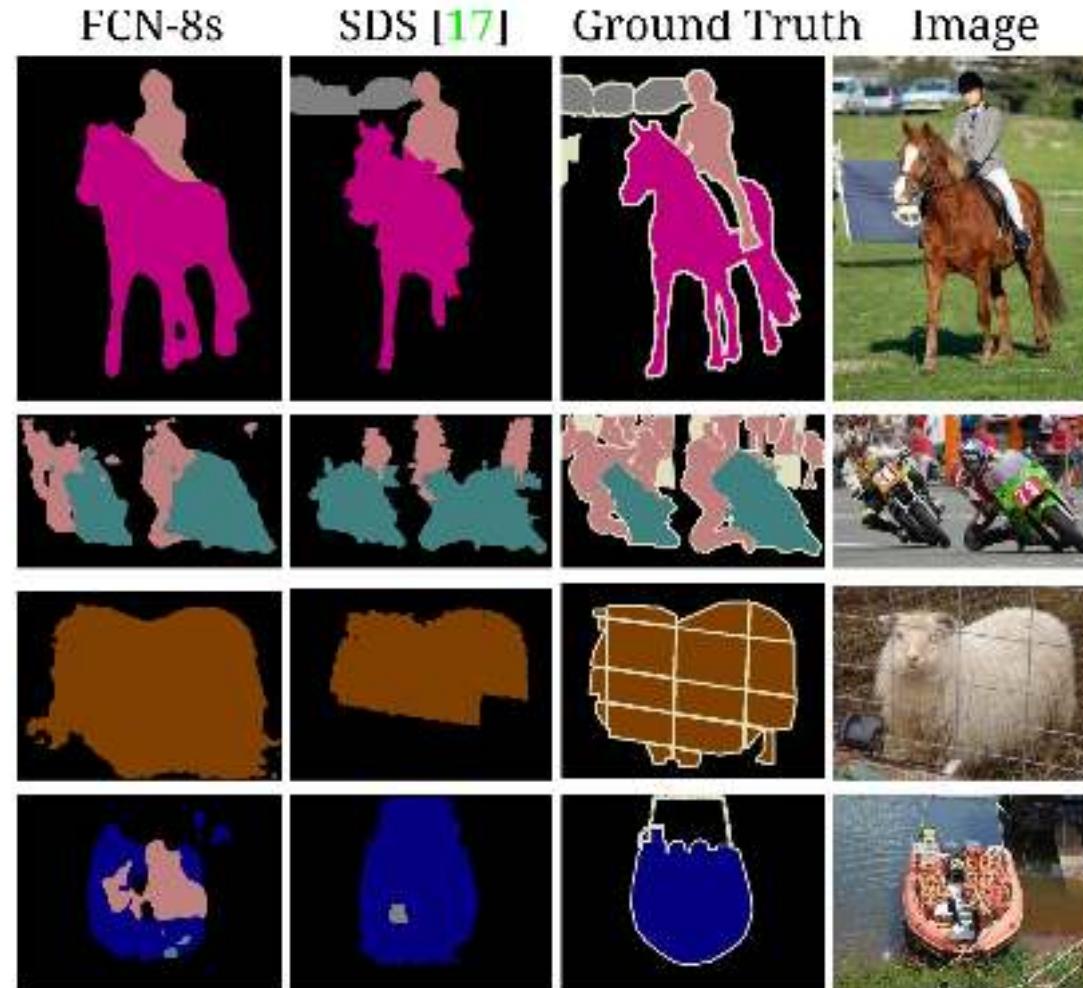
	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	90.3	75.9	62.7	83.2

Dataset Link

<http://host.robots.ox.ac.uk/pascal/VOC/voc2011/index.html>

*frequency weighted

Results



Importance

- FCN for pixel-wise prediction
- Arbitrary-sized inputs
- Learning and inference whole image at a time
- Leverage supervised pre-train model
- Upsampling (Learnable mechanism)

Challenges in Data Collection

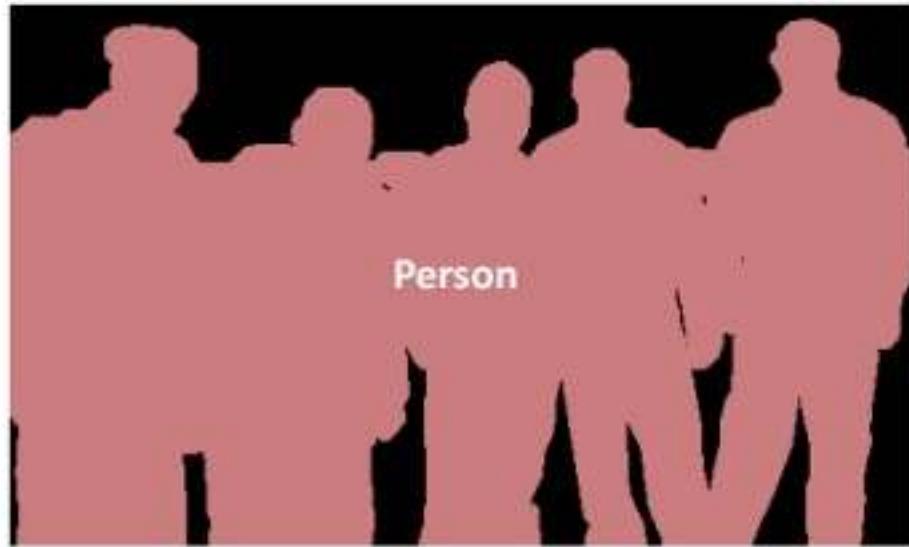
- Precise localization is hard to annotate
- Annotating every pixel leads to heavy tails
- **Common Solution:** annotate few classes (often things), mark rest as “Other”
- **Common Datasets**
 - Visual Objects Context (VOC 2005 to 2012) (**Challenge finished!!**)
 - COCO 2020 is already announced!!

<https://cocodataset.org/#home>

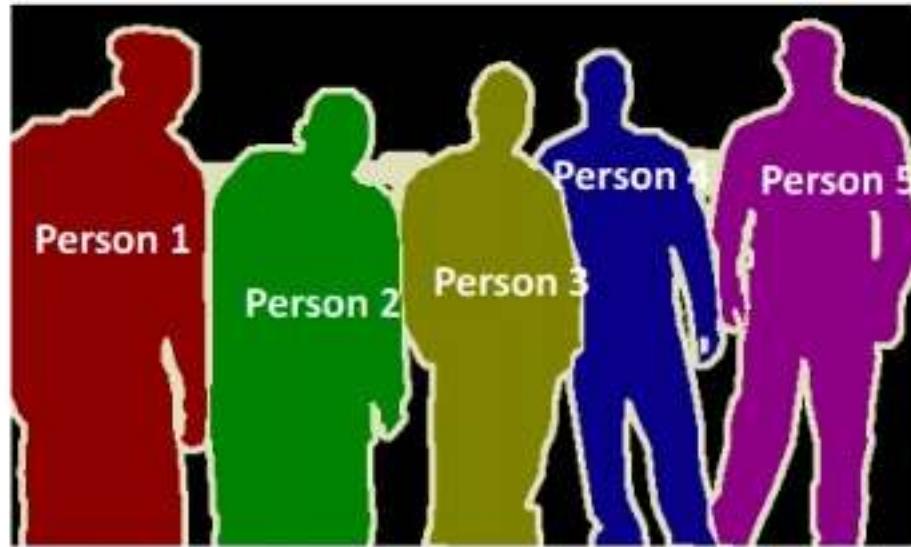
Other Architectures

- SegNet
- DeepLab-CRF
- Dilated Convolutions

Semantic segmentation vs. Instance Segmentation



Semantic Segmentation



Instance Segmentation

Summary

- Segmentation task is different from simple classification task because it requires predicting a class for each pixel of the input image

Reading

- Fully Convolutional Networks for Semantic Segmentation, CVPR (2015)

Thank You 😊

Appendix

- It is also worthy to review some standard deep networks that have made significant contributions to the field of computer vision, as they are often used as the basis of semantic segmentation systems:
- **AlexNet:** Toronto's pioneering deep CNN that won the 2012 ImageNet competition with a test accuracy of 84.6%. It consists of 5 convolutional layers, max-pooling ones, ReLUs as non-linearities, 3 fully-convolutional layers, and dropout.
- **VGG-16:** This Oxford's model won the 2013 ImageNet competition with 92.7% accuracy. It uses a stack of convolution layers with small receptive fields in the first layers instead of few layers with big receptive fields.
- **GoogLeNet:** This Google's network won the 2014 ImageNet competition with accuracy of 93.3%. It is composed by 22 layers and a newly introduced building block called inception module. The module consists of a Network-in-Network layer, a pooling operation, a large-sized convolution layer, and small-sized convolution layer.
- **ResNet:** This Microsoft's model won the 2016 ImageNet competition with 96.4 % accuracy. It is well-known due to its depth (152 layers) and the introduction of residual blocks. The residual blocks address the problem of training a really deep architecture by introducing identity skip connections so that layers can copy their inputs to the next layer.

Video Tracking

MDNets and GOTURN

Instructor: Dr. Muhammad Fahim

Slide credit and Source of the material

- This lecture is based on the following resources
 - EdwardLee: <https://www.jianshu.com/p/960620fce3b4>
 - <https://blog.csdn.net/linolzhang/article/details/72354427>
 - Found material over the internet to aligned the subject according to the need of the students

Contents

- What is Object Tracking?
- Object Tracking Models
- Components of Tracking Algorithm
- Tracking Models
 - Multi Domain Network (MDNet)
 - Generic Object Tracking Using Regression Networks (GOTURN)
- Common Problems
- Summary

Object Tracking

- The goal of object tracking is to **keep track of an object** in a **video sequence**
- Object tracking is an **old and hard problem** of computer vision.
- There are **various techniques** and algorithms which try to solve this problem in **various different ways**.
 - Motion-based model
 - Visual appearance-based model

Motion-based Model

- A motion model is developed that **captures the dynamic behavior** of an object.
- It predicts the **potential position of objects** in the future frames
- ***Drawback***
 - The motion model can **fail in scenarios** where motion is caused by things that are **not in a video** or **abrupt direction** and **speed change**.

Motion-based Model Techniques

- Optical flow
- Kalman filtering
- Kanade-Lucas-Tomashi (KLT) feature tracker
- Meanshift and many more...

Visual Appearance-based Model (VAM)

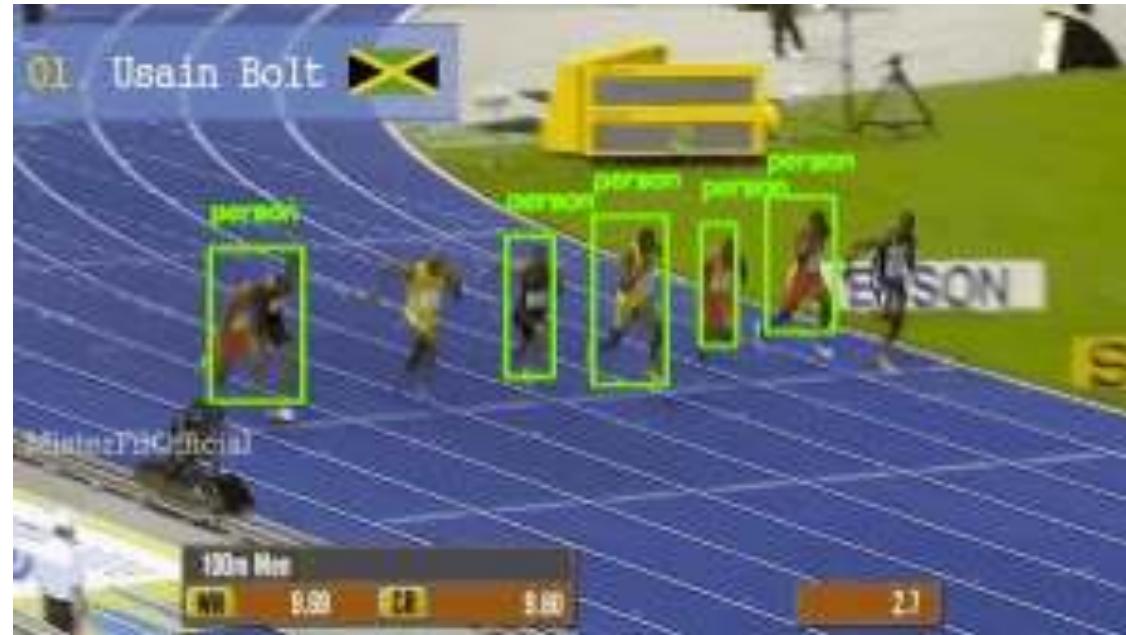
- VAM tries to understand the appearance of the object
- Technically, they learn to discriminate the object from the background
- Visual models can be
 - Single object trackers
 - Multiple-object trackers

Single Object Tracking



<https://nanonets.com/blog/object-tracking-deepsort/>

Multiple Object Tracking



Components of a VAM Tracking Algorithm

1. *Target initialization*

- We need to define the **initial state of the target** by drawing a bounding box around it.
- The idea is to draw bounding box of the target in the initial frame of the video and **tracker has to estimate the target position** in the remaining frames in the video.

2. *Appearance modeling*

- Now, we need to **learn the visual appearance** of the object by using **learning techniques**.
- In this phase, we have to model and **learn about the visual features** of the object while in motion, various view-points, scale, illuminations etc.

Components of a Tracking Algorithm

3. *Motion estimation*

- Objective of motion estimation is the learn to predict a zone where the target is most likely to be present in the subsequent frames.

4. *Target positioning*

- Motion estimation gives us the possible region where the target could be present and we scan that using the visual model to lock down the exact location of the target.

Generally a tracking algorithms don't try to learn all the variations of the object.

CNN Based Online Training Trackers

Multi Domain Network(MDNet)

POSTECH, South Korea (CVPR 2016)

<https://arxiv.org/pdf/1510.07945.pdf>

<https://github.com/hyeonseobnam/MDNet>

MDNet Motivation

1. The Pretrain Problem
2. Multi-Domain Problem
3. The Network Size Problem

1. The Pretrain Problem

- Pre-train CNN for large-scale feature extraction on large-scale data
(not reasonable!)
- **A Realistic Approach**
 - Training the CNN of the tracking model directly on the video tracking data.
 - For tracking targets, their types may be different, but there should be some commonalities including edge gradients, etc., which require network learning.

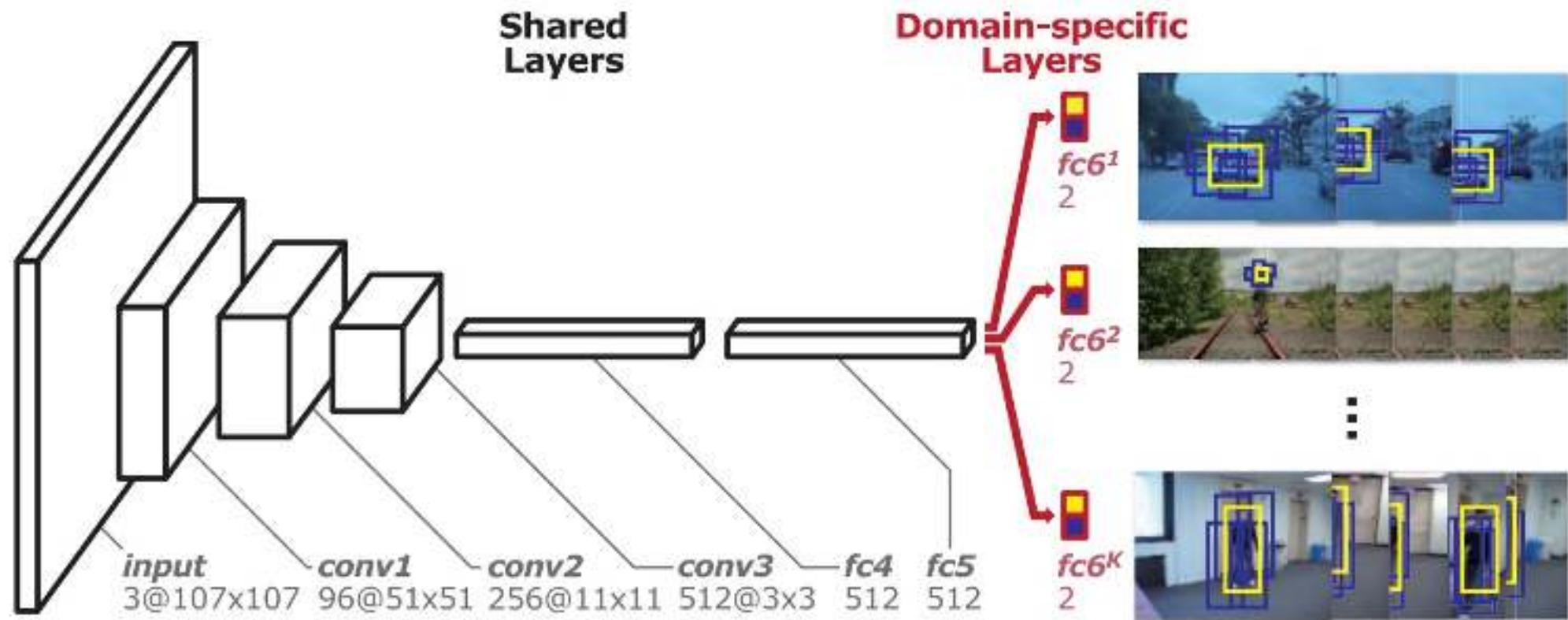
2. Multi-Domain Problem

- It is **more difficult to train CNN directly** with tracking data.
- **Why?** One object is the **target in one video** frame sequence, and may be **background in another** sequence.
- **Additional challenges**
 - Background clutter
 - Occlusions
 - Illumination variations, and so on.

3. The Network Size Problem

- The CNN network in detection, classification, and segmentation is very large!!
- However, in tracking, label has only two categories
 - Target
 - Background

MDNet Architecture



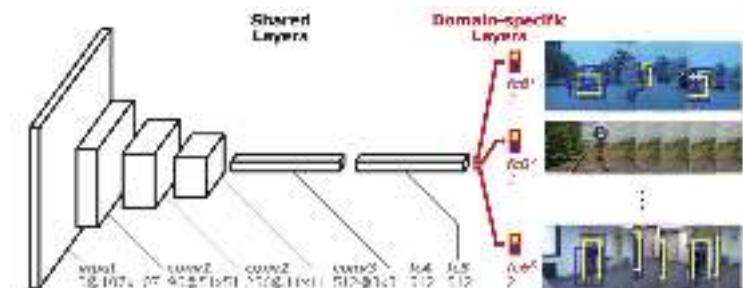
- The **convolutional layers are identical** to the corresponding parts of VGG-M network except that the feature map sizes are adjusted by input size.

VGG-M Network: <https://arxiv.org/pdf/1405.3531.pdf>

Learning Algorithm

Our CNN is trained by the **Stochastic Gradient Descent (SGD)** method, where **each domain** is handled exclusively in each iteration. In the k^{th} iteration, the network is updated based on **a minibatch** that consists of the training samples from the $(k \bmod K)^{\text{th}}$ sequence, where only a single branch $\text{fc6}^{(k \bmod K)}$ is enabled. It is repeated until the network is converged or the predefined number of iterations is reached. Through this learning procedure, domain-independent information is modeled in the **shared layers** from which useful generic feature representations are obtained.

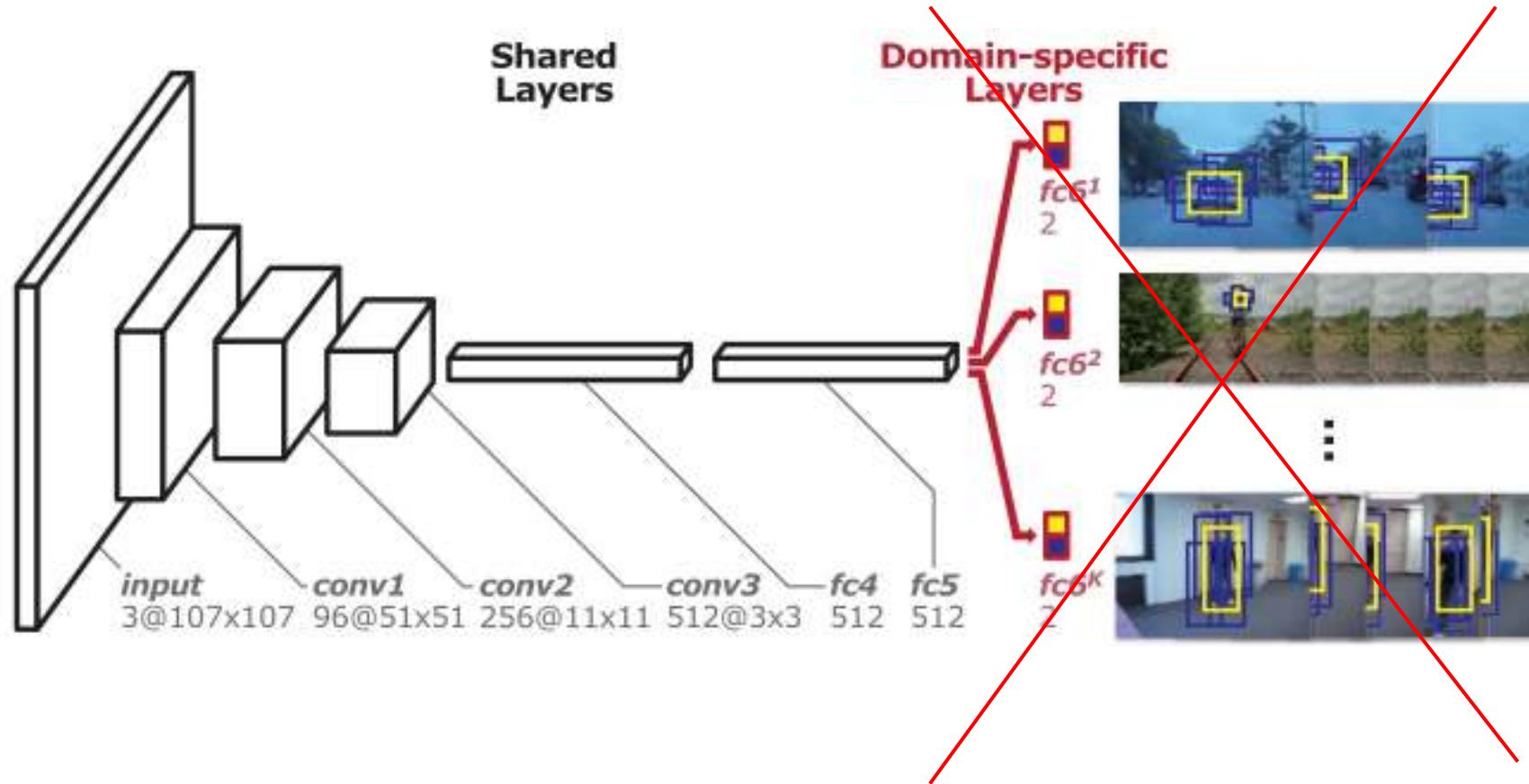
- **What will be the impact of this mechanism?**



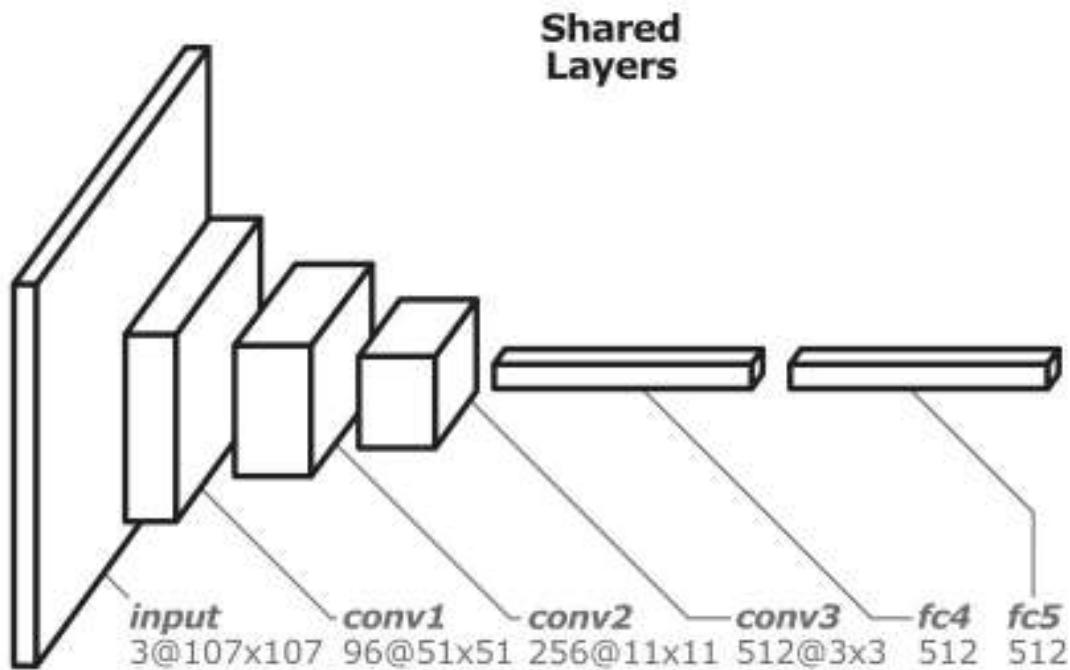
Tracking Part (MDNet)

Online Tracking using MDNet

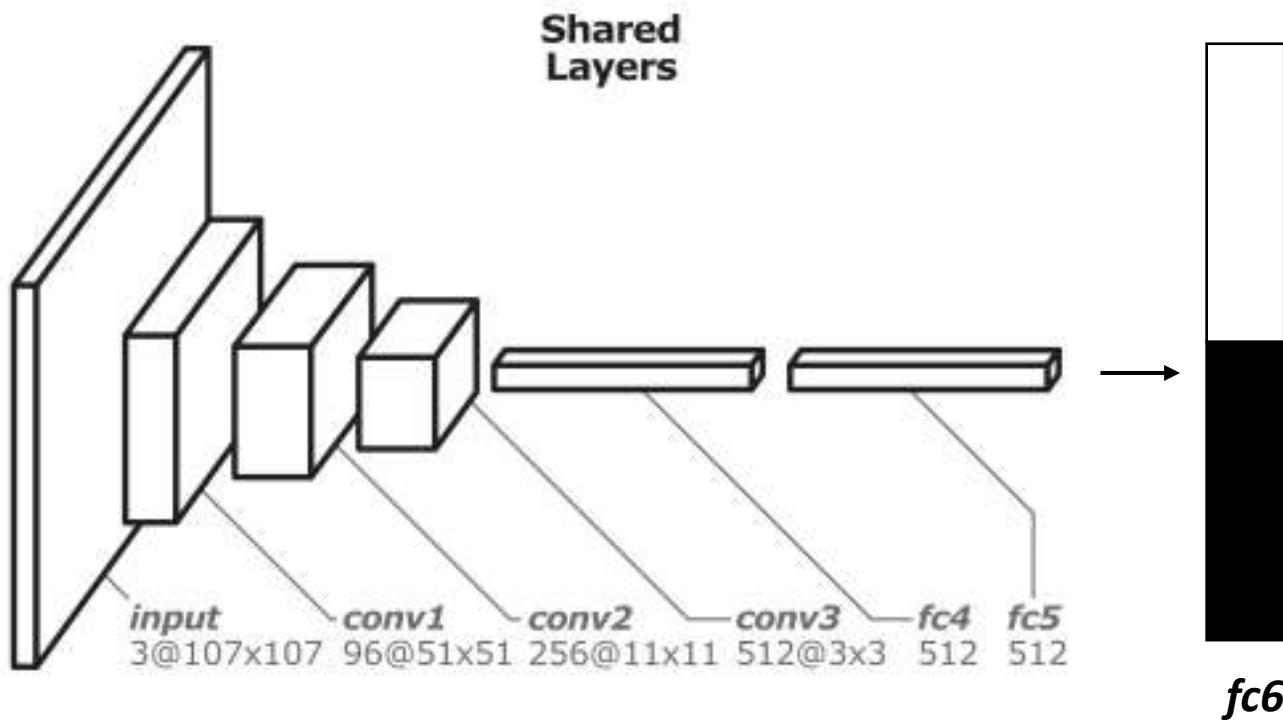
- After completion of the training



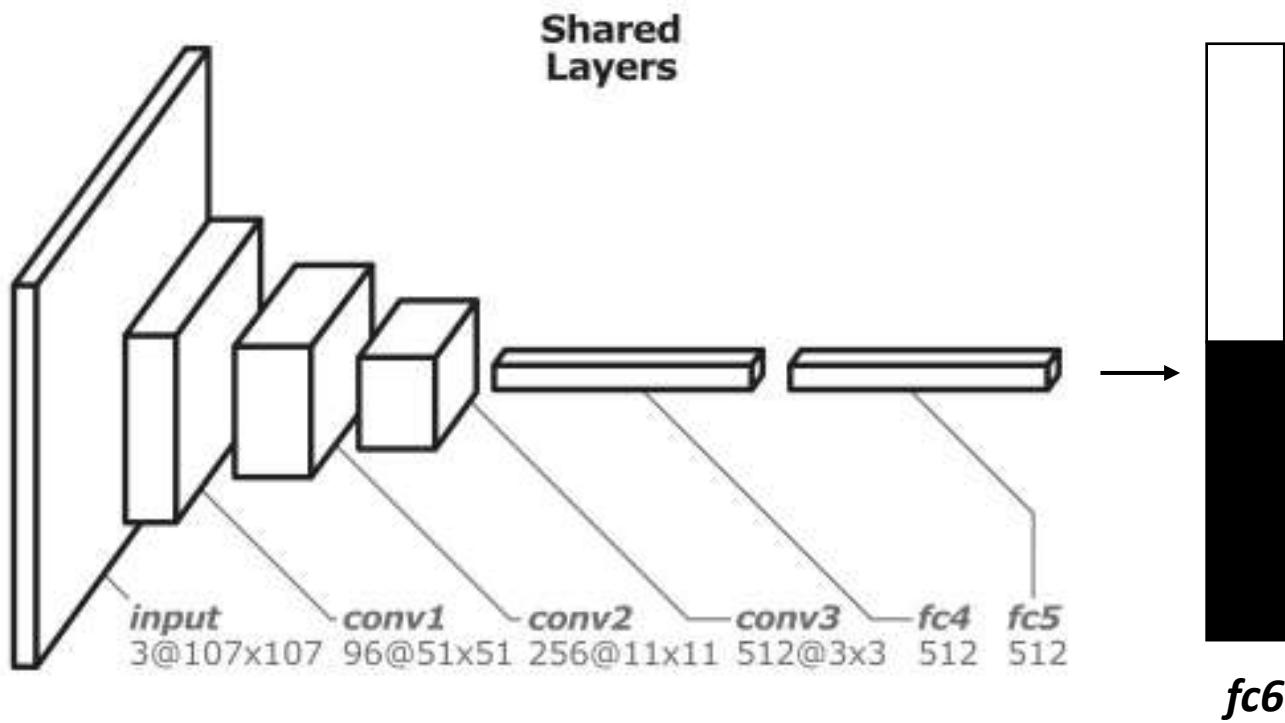
Online Tracking using MDNet



Online Tracking using MDNet



Online Tracking using MDNet



- Fine-tune the new domain specific layer and the fully connected layers in the shared network online at the same time.

Online Tracking using MDNet

- Tracking procedure is explained as:
 1. Tracking Control and Network Update
 2. Hard Minibatch Mining
 3. Bounding Box Regression

1. Tracking Control and Network Update

- Authors consider **two complementary aspects** in visual tracking:
 - Robustness and adaptiveness by long-term and short term updates
- **Long-term updates** are performed in regular intervals using positive samples collected for a long period
- **Short-term updates** are conducted whenever potential tracking failures are detected – when the positive score of the estimated target is less than 0.5

Long-term and Short-term Updates

To estimate the target state in each frame, N target candidates $\mathbf{x}^1, \dots, \mathbf{x}^N$ sampled around the previous target state are evaluated using the network, and we obtain their positive scores $f^+(\mathbf{x}^i)$ and negative scores $f^-(\mathbf{x}^i)$ from the network. The optimal target state \mathbf{x}^* is given by finding the example with the maximum positive score as

$$\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}^i} f^+(\mathbf{x}^i). \quad (1)$$

2. Hard Minibatch Mining

The majority of **negative examples** are typically **trivial or redundant** in tracking-by-detection approaches

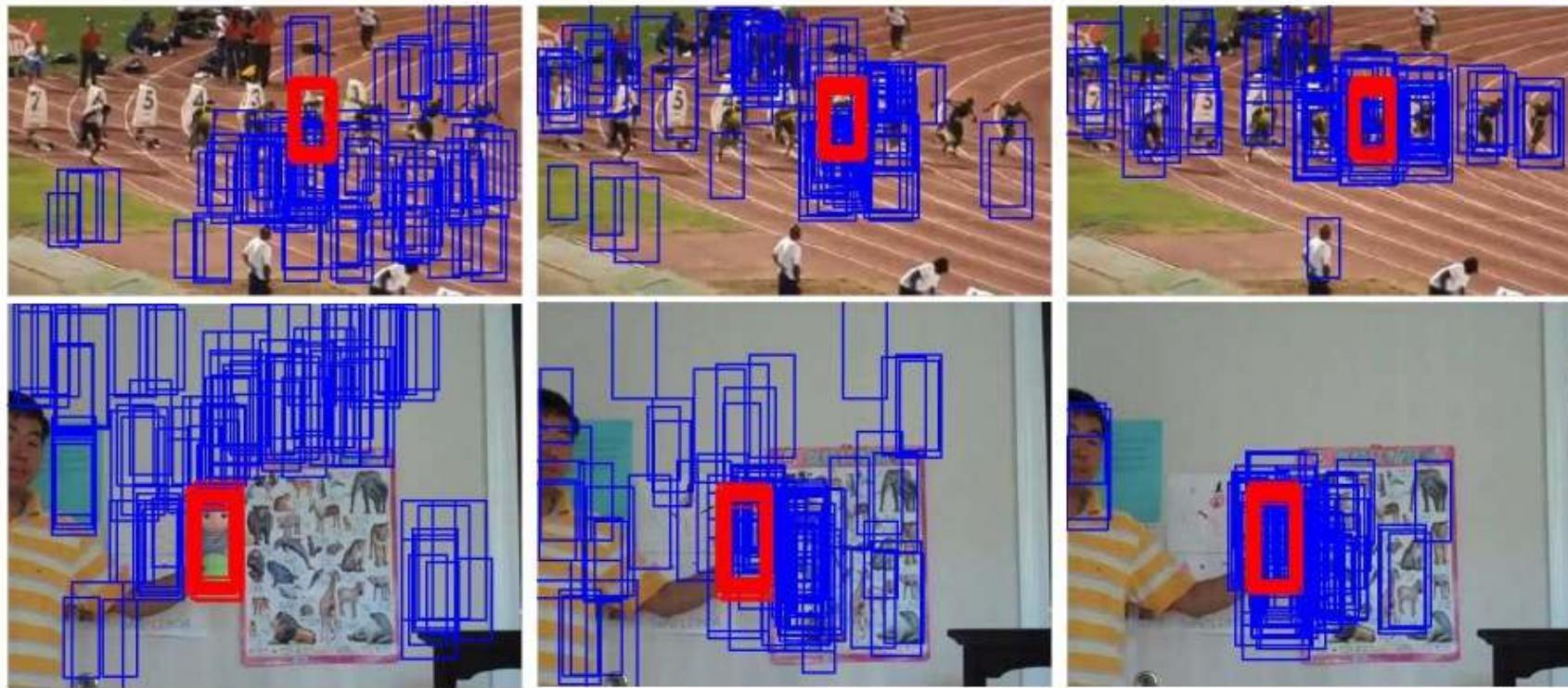
While only a **few negative samples** are **effective to training** a classifier

- Hence, the ordinary SGD method, where the **training samples evenly contribute to learning**, easily suffers from a **drift problem** since the negative examples are considered insufficiently.
- **Solution:** Hard Negative Mining

Similar Task: Have a look
Hard Example Mining with Auxiliary Embeddings, CVPR2018

Hard Negative Mining

- Training and testing procedures are alternated to identify the hard negative examples, typically false positives



(a) 1st minibatch

(b) 5th minibatch

(c) 30th minibatch

3. Bounding Box Regression

- Due to the **high-level abstraction** of CNN-based features and **data augmentation strategy** – which samples multiple positive examples around the target
- MDNet sometimes fails to find **tight bounding boxes** enclosing the target.
- They apply the **bounding box regression technique**, which is popular in object detection, to **improve target localization accuracy**.

3. Bounding Box Regression

Given the first frame of a test sequence, we train a simple linear regression model to predict the precise target location using conv3 features of the samples near the target location. In the subsequent frames, we adjust the target locations estimated from Eq. (1) using the regression model if the estimated targets are reliable (*i.e.* $f^+(x^*) > 0.5$). The bounding box regressor is trained only in the first frame since it is time consuming for online update and incremental learning of the regression model may not be very helpful considering its risk. Refer to [12] for details as we use the same formulation and parameters.

Experiments

- Object Tracking Benchmark (OTB)
- Visual Object Tracking (VOT2014)



CNN Based Offline Training Trackers

Generic Object Tracking Using Regression Networks (GOTURN)

Paper Title: Learning to Track at 100 FPS with Deep Regression Networks, ECCV, 2016

<https://arxiv.org/abs/1604.01802>

<https://github.com/davheld/GOTURN>

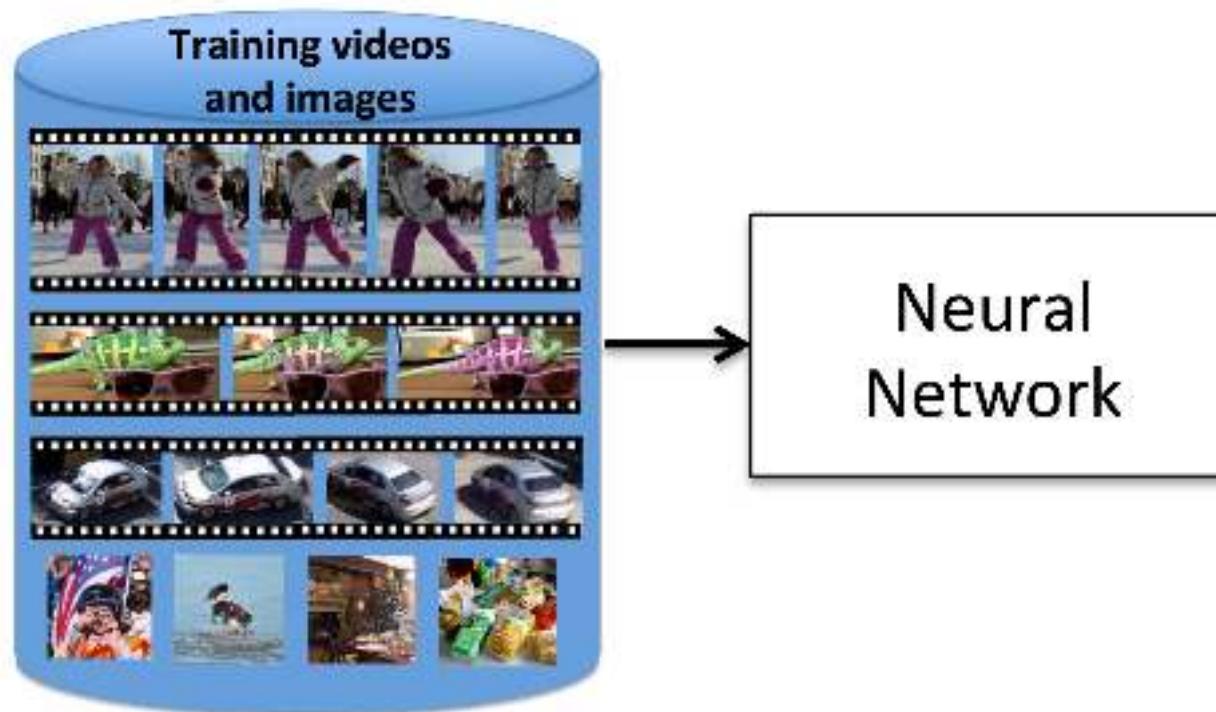
GOTURN

- GOTURN apply the discriminative power of convolutional neural networks to the task of visual object tracking.
- It is offline learning tracker based on CNN
 - The tracker is trained using thousands of videos for general object tracking.
 - This tracker can be used to track most of the objects without any problem even if those objects were not part of the training set

GOTURN can run very fast (i.e., 100fps on GPU powered machine)

GOTURN – Training

Training:

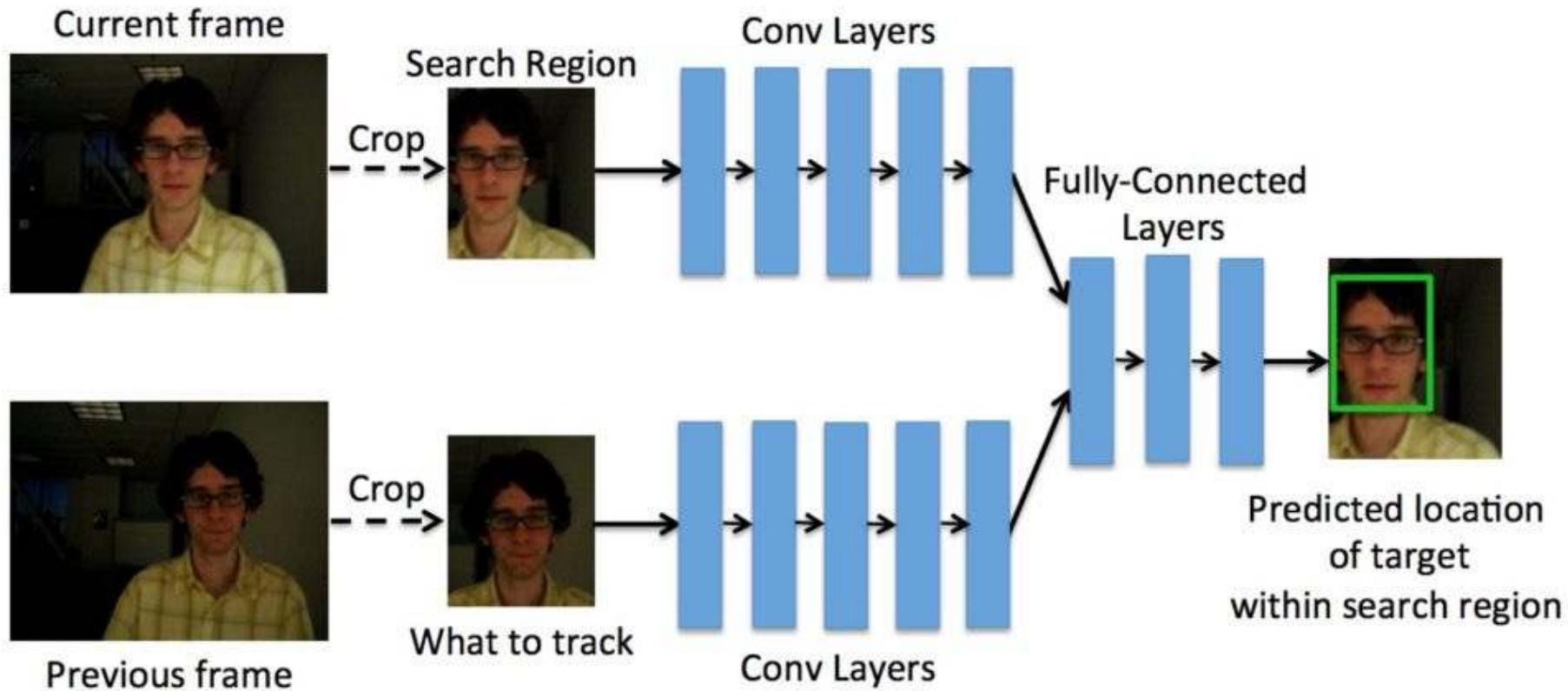


Network learns generic object tracking

GOTURN – Training

- At a high level, we **feed frames of a video into a neural network**
- The network **successively outputs the location** of the tracked object in each frame.
- We train the **tracker entirely offline** with video sequences and images.
- Offline training procedure
 - It learns a **generic relationship between appearance and motion** that can be used to track novel objects at test time with **no online training** required.

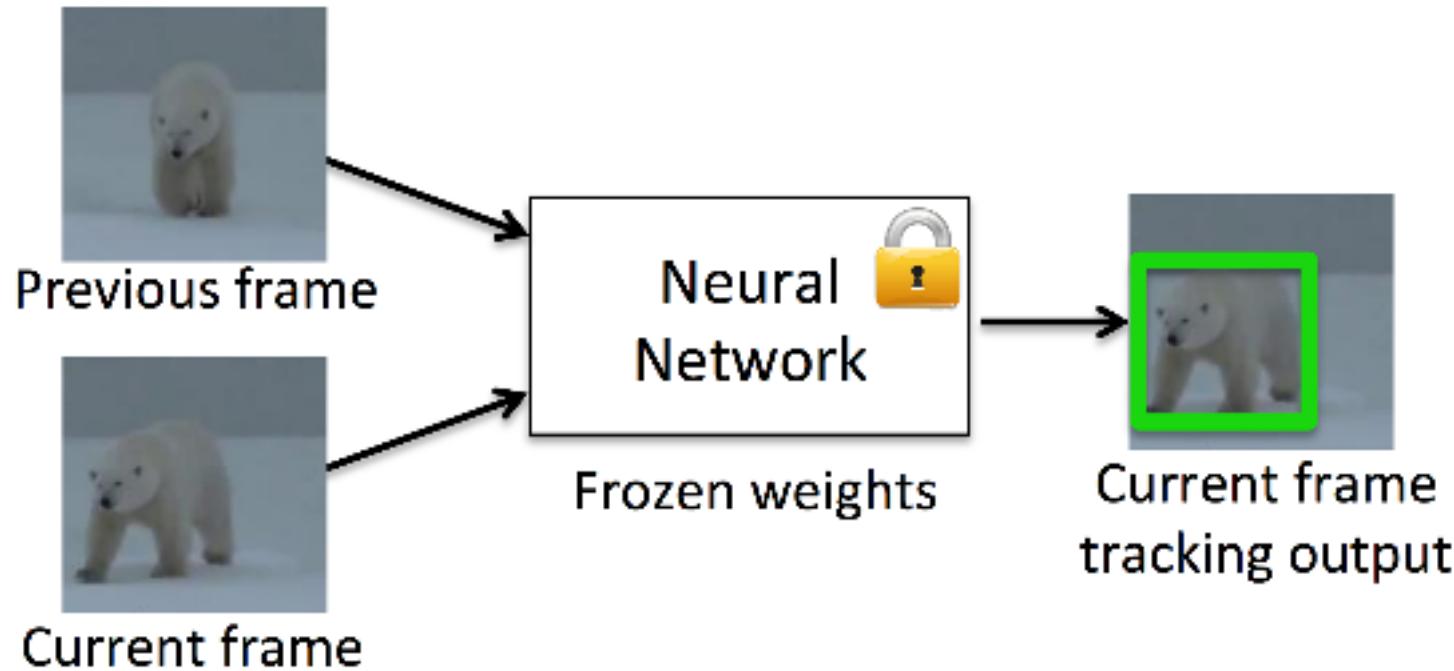
GOTURN – Architecture



- The layers are simply the [first five convolutional layers of the CaffeNet architecture](#).
- The outputs of these convolutional layers are concatenated into a [single vector of length 4096](#).
- The last fully connected layer is finally connected to the output layer containing [4 nodes representing the top and bottom points](#) of the bounding box.

GOTURN – Test

Test:



**Network tracks novel objects
(no finetuning)**

Network Hyperparameters

- Network hyperparameters are taken from the defaults for CaffeNet
- For each fully-connected layer:
 - Dropout and ReLU non-linearities are used.

Training and Loss Function

- Network is trained with a combination of videos and still images.
- The training procedure is described below.
 - In both cases: L1 loss between the predicted bounding box and the ground-truth bounding box.

$$L1LossFunction = \sum_{i=1}^n |y_{true} - y_{predicted}|$$

Input/output Format

- **What to track?** In case there are **multiple objects in the video**
 - The network **must receive** some information about **which object in the video is being tracked**.
 - To achieve this
 - Input an image of the target object into the network
 - The network will track **whatever object is being input in this crop**.

This input allows the network to track novel objects that **it has not seen before**

Input/output Format

- **Where to look?**
 - To find the target object in the current frame, the tracker should know where the object was previously located.
 - Since objects tend to move smoothly through space, the previous location of the object will provide a good guess of where the network should expect to currently find the object.
 - It is achieved by choosing a search region in current frame based on the object's previous location.

Network Output

- The goal of the network is then to regress to the location of the target object within the search region.
- The network outputs the coordinates of the object in the current frame, relative to the search region.
- The network's output consists of the coordinates of the top left and bottom right corners of the bounding box.

Applications

- Object tracking has a wide range of applications in computer vision, such as:
 - Surveillance
 - Human-computer interaction
 - Medical imaging
 - Traffic flow monitoring
 - Human activity recognition, etc.

Common Problems

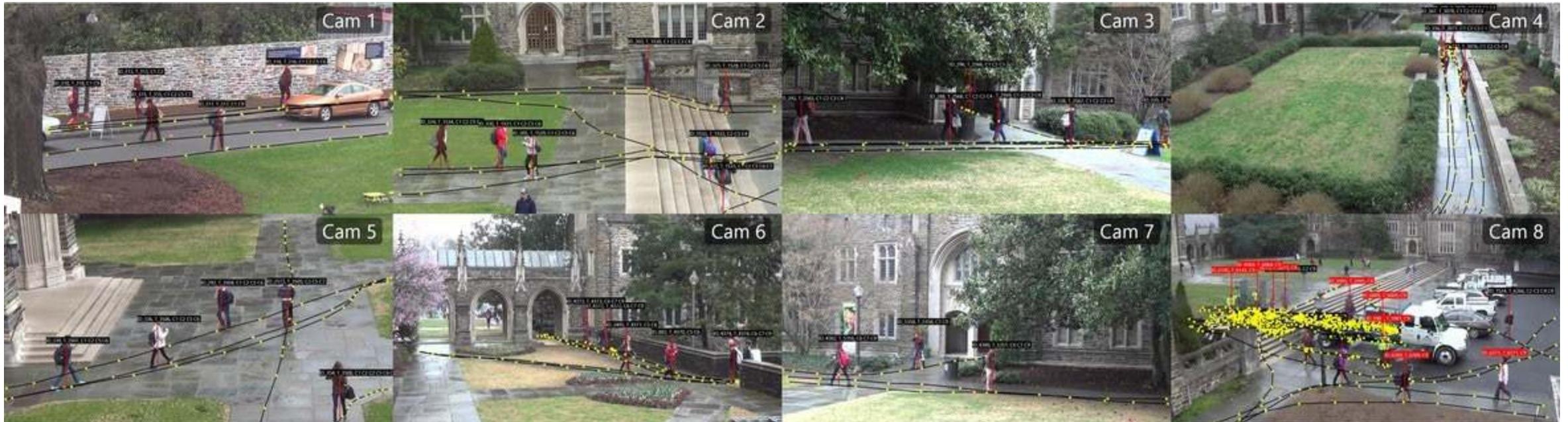
- Tracking an object on a **straight road or a clear area** might sound very easy.
- But **hardly any real world application** is that **straight forward** and free of any challenges.

Occlusion



Common Problems

Variations in view points



Common Problems

Non stationary camera

- When the camera used for tracking a particular object is also in motion **with respect to the object**, it can often lead to unintended consequences.
- A **robust tracker** for this problem can be very helpful in important applications like object tracking **drones, autonomous navigation**.

Common Problems

- **Scale Change:** Huge changes in object scale may cause a failure in detection.
- **Background Clutters:** Background near object has similar color or texture as the object. Hence, it may become harder to separate the object from the background.
- **Illumination Variation:** Illumination near the target object is significantly changed. Hence, it may become harder to visually identify it.
- **Low Resolution:** When the number of pixels inside the ground truth bounding box is very less, it may be too hard to detect the objects visually.
- **Annotating Training Data:** One of the most annoying things about building an object tracker is getting good training data for a particular scenario.

Reading

- If you are interested in this area then explore
 - <https://github.com/abhineet123/Deep-Learning-for-Tracking-and-Detection>
 - <https://cv-tricks.com/object-tracking/quick-guide-mdnet-goturn-rolo/>
 - <https://nanonets.com/blog/object-tracking-deepsort/>

Thank You 😊

Appendix

- OpenCV tracking API that was introduced in OpenCV 3.0.
- Eight different trackers available in OpenCV 3.4.1
 - BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE and CSRT.

Hard Negative Mining

- Let's say I give you a bunch of images that contain one or more people, and I give you bounding boxes for each one. Your classifier will need both positive training examples (person) and negative training examples (not person).
- For each person, you create a positive training example by looking inside that bounding box. But how do you create useful negative examples?
- A good way to start is to generate a bunch of random bounding boxes, and for each that doesn't overlap with any of your positives, keep that new box as a negative.
- Ok, so you have positives and negatives, so you train a classifier, and to test it out, you run it on your training images again with a sliding window. But it turns out that your classifier isn't very good, because it throws a bunch of false positives (people detected where there aren't actually people).
- A hard negative is when you take that falsely detected patch, and explicitly create a negative example out of that patch, and add that negative to your training set. When you retrain your classifier, it should perform better with this extra knowledge, and not make as many false positives.

https://www.reddit.com/r/computervision/comments/2ggc5l/what_is_hard_negative_mining_and_how_is_it/