COM SCI 188

Intro to Robotics

Lecture 7

Yuchen Cui

Winter 2026

Visual art generated by DALLE

Agenda

- Announcements

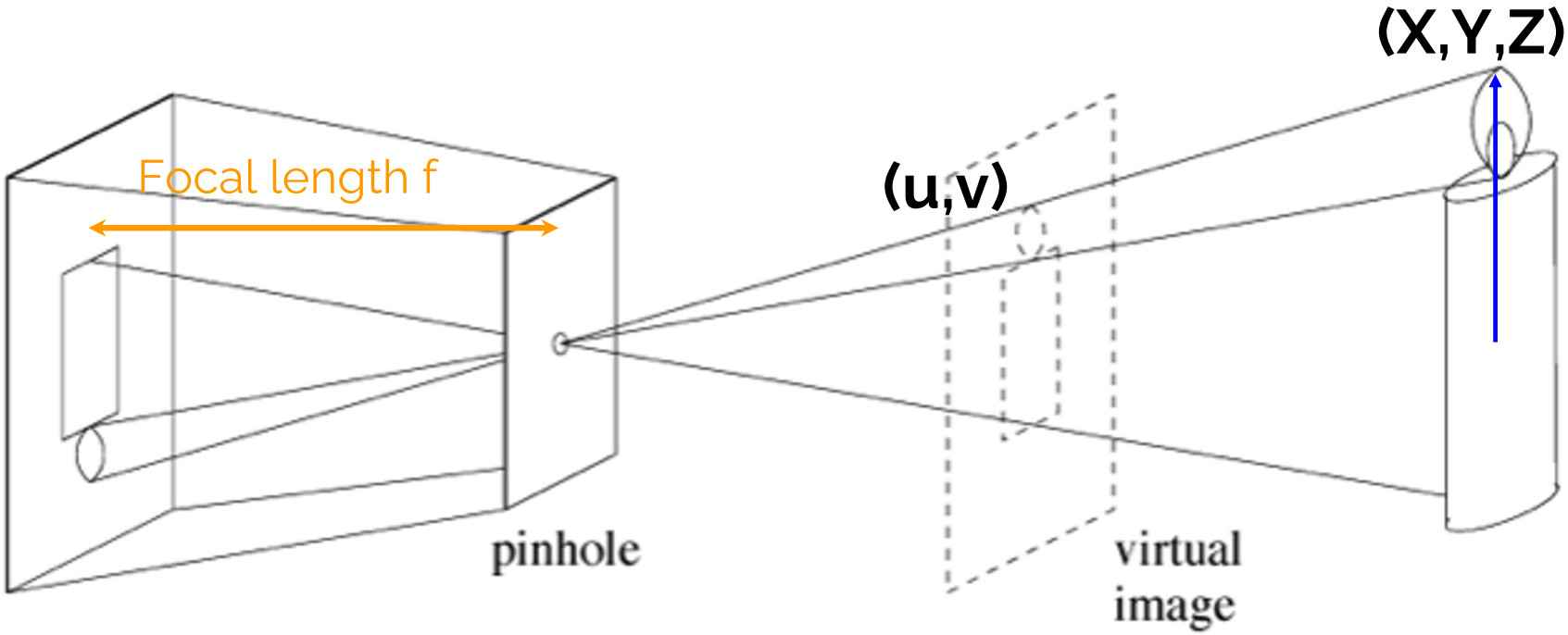
- Camera Model & Camera calibration
- Computer Vision Introduction
- Convolutional Neural Networks

Announcements

- Problem Set 2 due Friday
- Coding Assignment 2 out, due next Friday

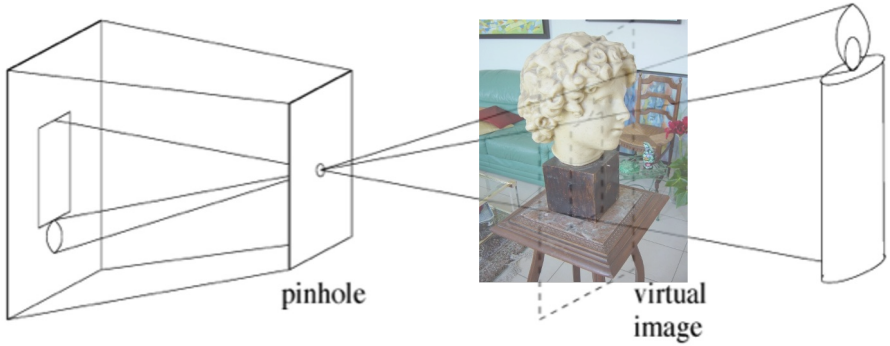
- Midterm Exam (Week 7): Feb 19th

Recap: pinhole camera model



Recap: Camera Projection

Image Unit: Pixels



World Unit:
Meters

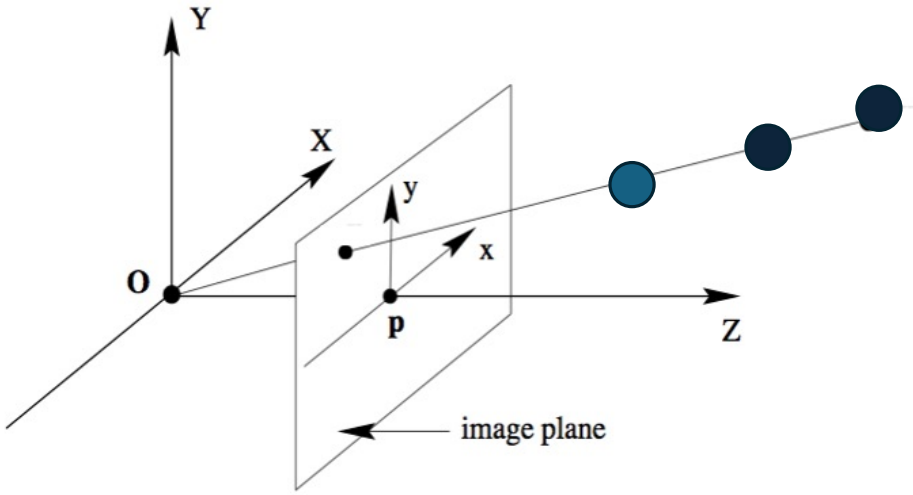
If we let f to take care transform from word unit to image unit.
The unit of f is pixel

$$u = \frac{fX}{Z} \qquad v = \frac{fY}{Z}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

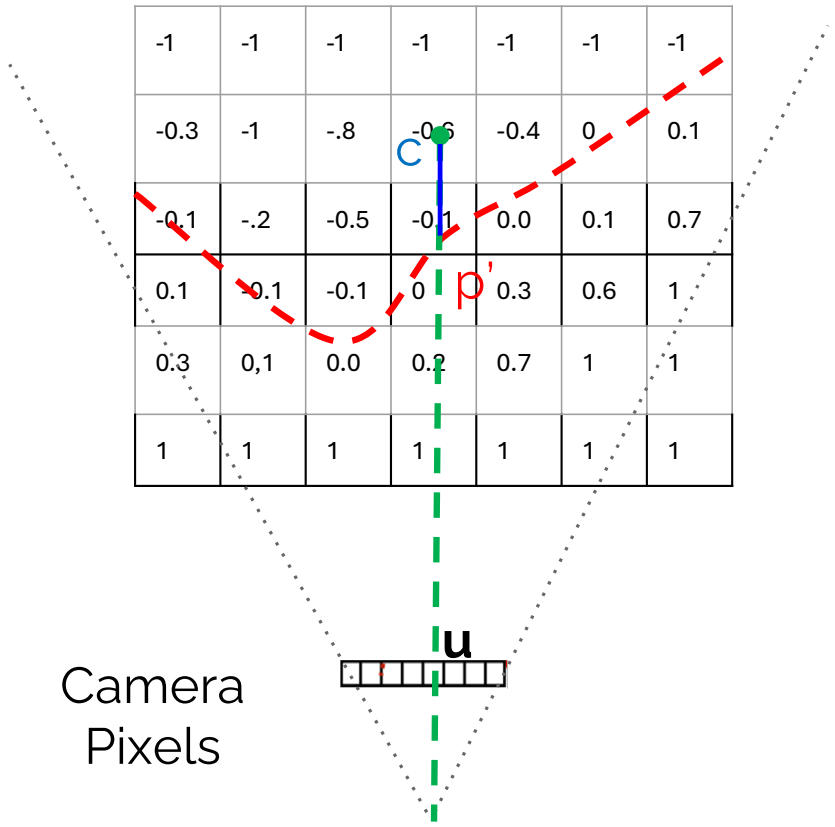
Recap:

$$\lambda \begin{bmatrix} u \\ v \\ f \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



All the points on the same ray, projected to the same pixels [u,v]
Only the point with smallest distance is observed in the image.

Projective Sign Distance Function



Sign indicated occluded/free space with respect to camera

In this example:

Red: surface observed in camera

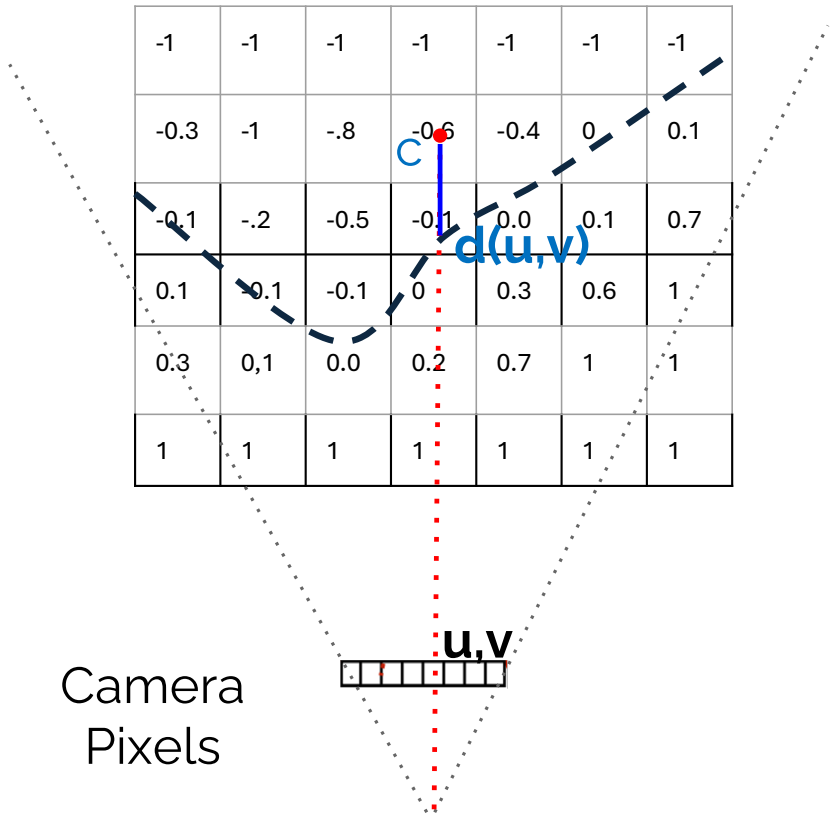
Green: camera projection ray

We take p' as the nearest point.

Sign: c is behind p' -> occluded -> negative

Projective distance: Blue line

Computing projective distance

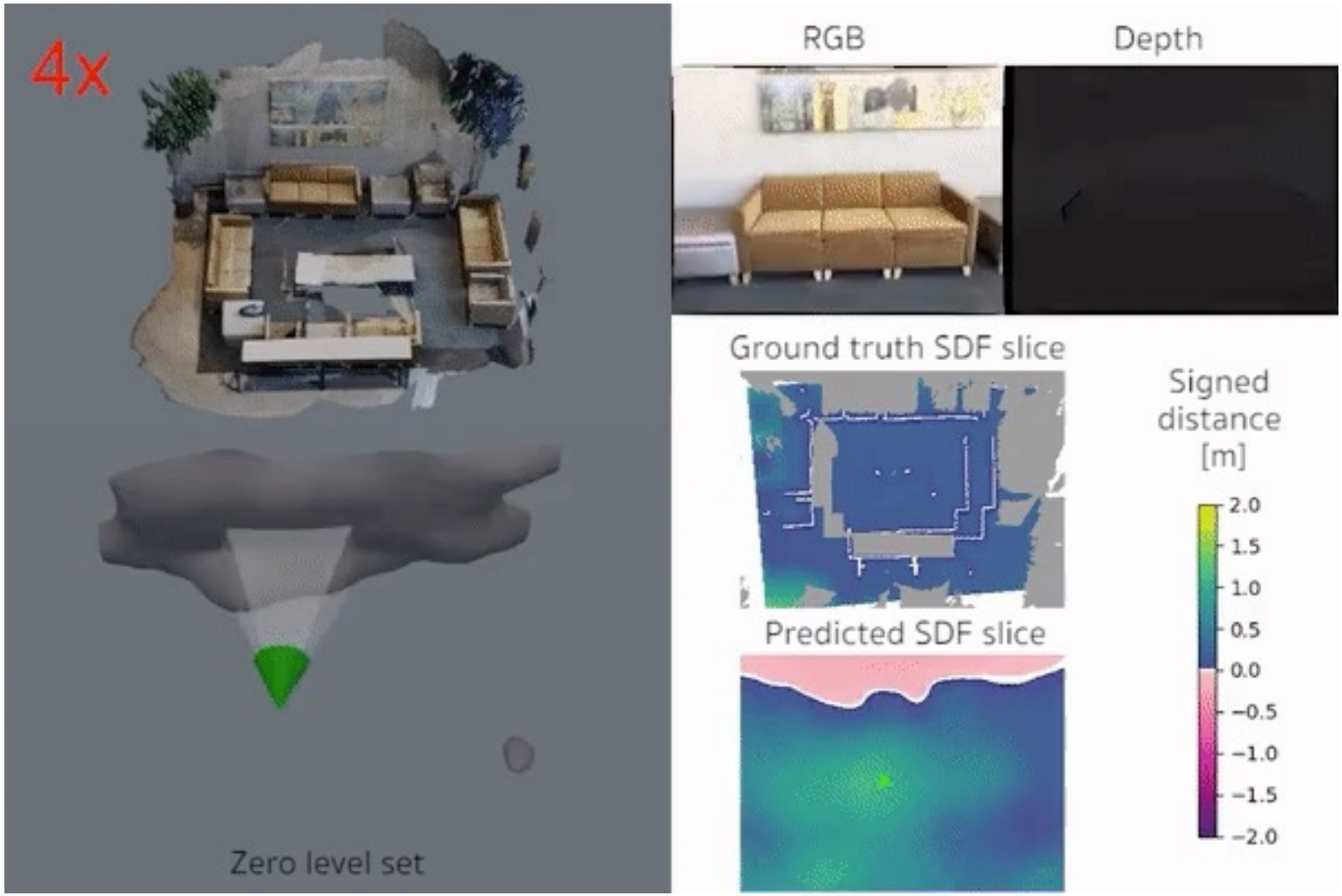


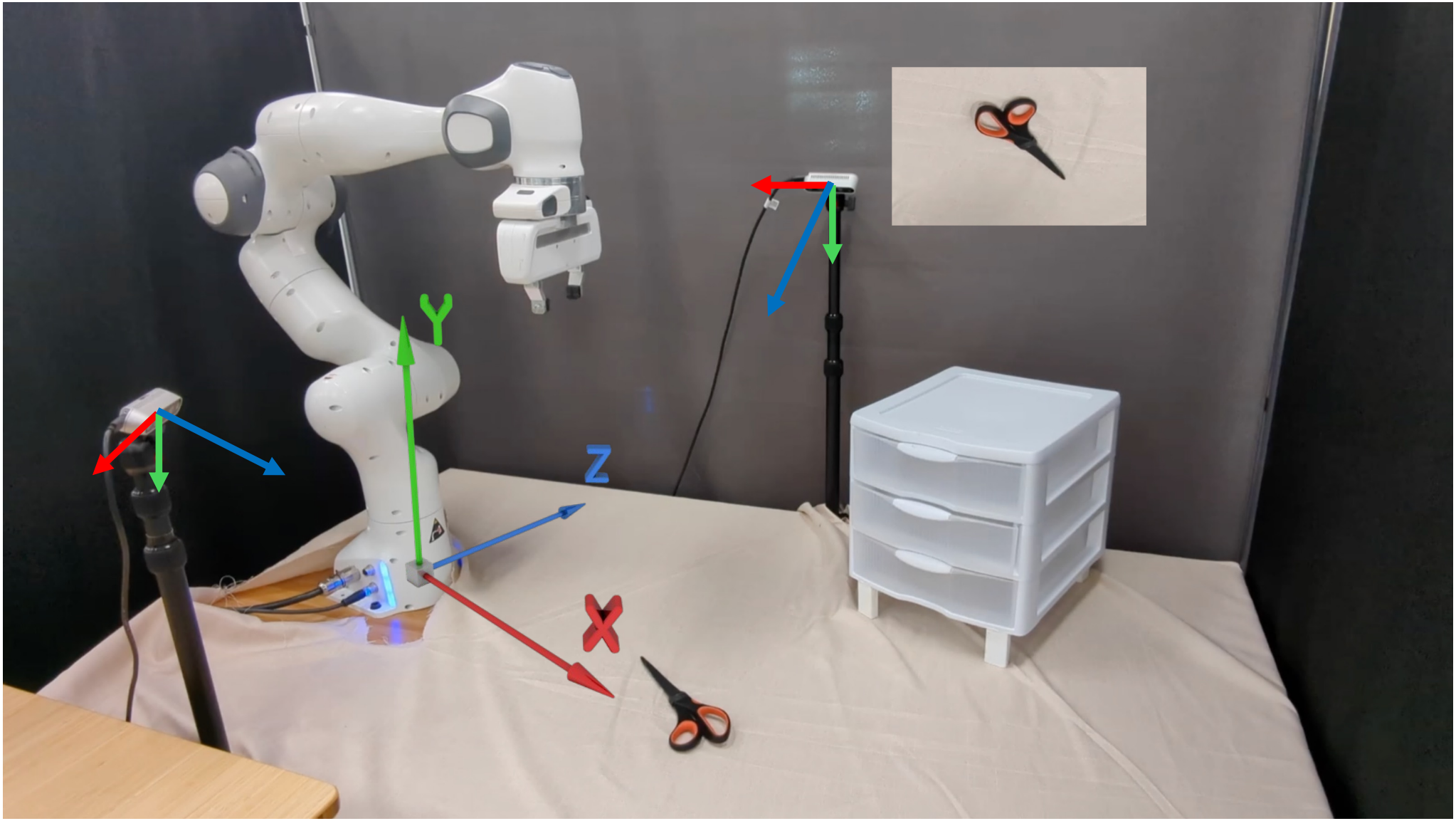
- 3D point location **c<x,y,z>**
- Project **c<x,y,z>** to the 2D image, gives us 2D coordinate **u,v**
- Depth reading on pixel is d(u,v)

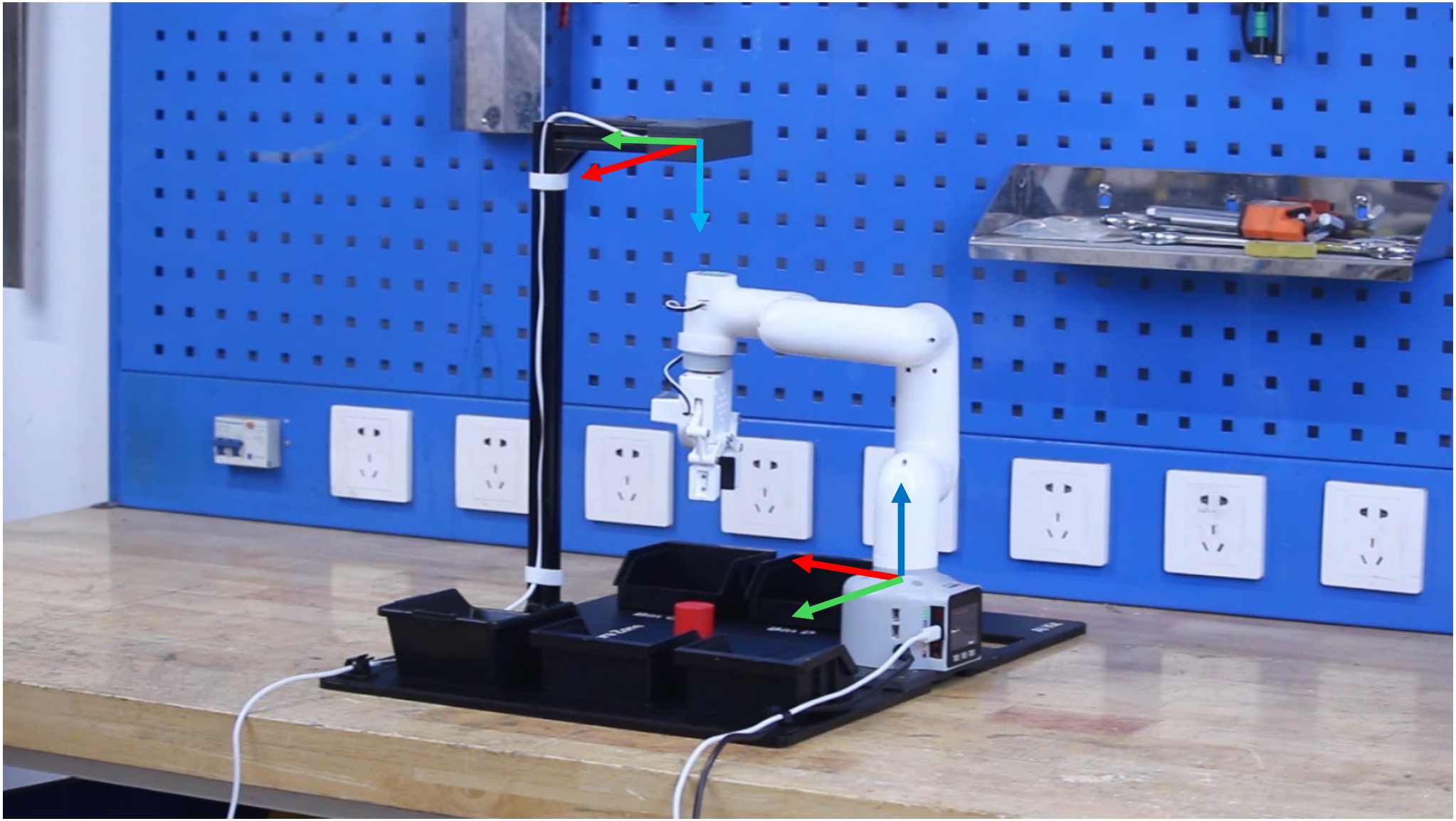
Signed projective distance

$$d_{proj} = d(u,v) - z$$

iSDF: Real-Time Neural Signed Distance Fields for Robot Perception




Y
Z
X



Recap: Camera Projection

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\mathbf{K}}_{\text{Intrinsic Matrix}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \underbrace{\left[\begin{array}{c|c} I & t \\ \hline \mathbf{0} & 1 \end{array}\right]}_{\text{translation}} \times \underbrace{\left[\begin{array}{c|c} R & \mathbf{0} \\ \hline \mathbf{0} & 1 \end{array}\right]}_{\text{rotation}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Camera Extrinsic Matrix [R|t]

Camera Parameter
Camera Projection Matrix

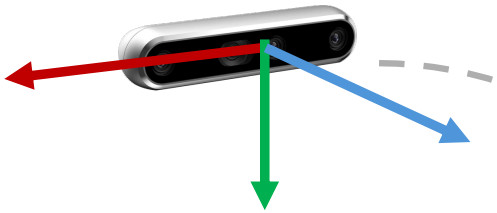
$$\mathbf{x} \overset{\mathbf{P}}{=} \mathbf{PX}$$

Recap: Camera Projection

$$\mathbf{x} = \mathbf{K}\left[\mathbf{R}|\mathbf{t}\right]\mathbf{X}$$

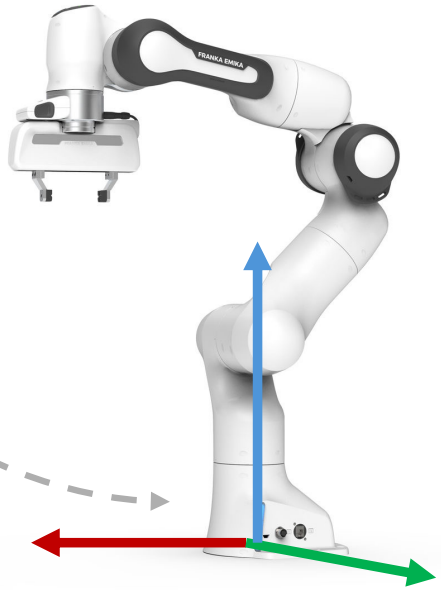
- Map a 3D point X into a 2D coordinate in image x
- How to describe its *pose* in the world? (extrinsic matrix)
- How to describe its internal parameters? (intrinsic matrix)

Camera Calibration



Camera/Image frame:
reference frame for *visual observations*

Robot base frame:
reference frame for *actions*



Hand-Eye Calibration

Camera Calibration

Goal: express pixels in robot frame!

Option 1: solve for camera parameters

– intrinsic (focal length, principle point, pixel size)

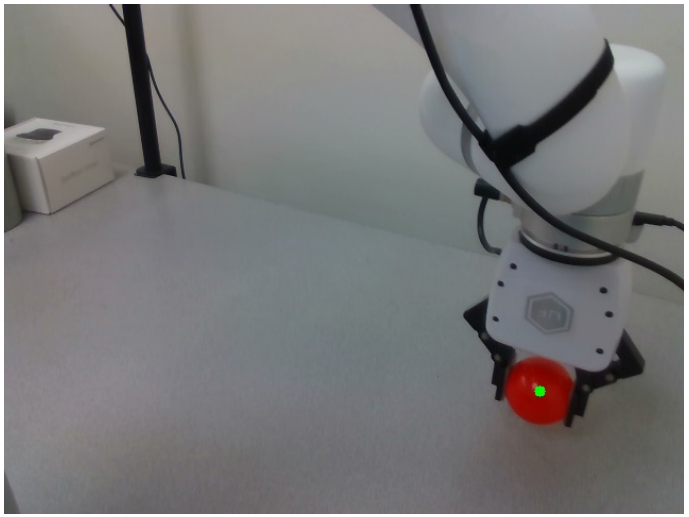
– extrinsic (rotation angles, translation)

Option 2: solve for projection matrix

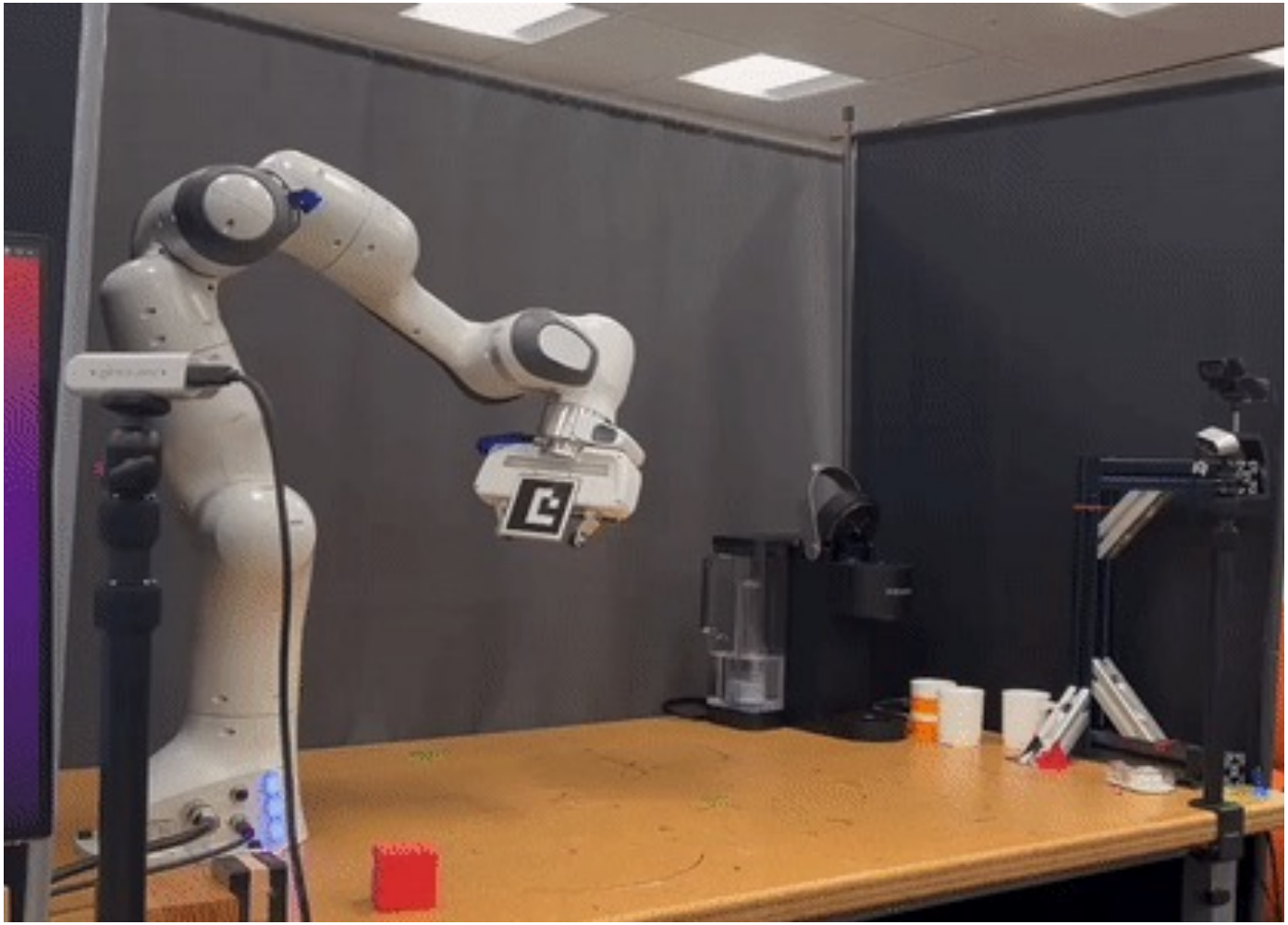
$$\mathbf{X} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{PX}$$

Robot: move_to(x=0.3, y=-0.2, z = 0.1)

Camera:



During the hand-eye calibration process, we move robot to a know position in its base frame, and captures the image with camera and detects the robot's hand's position (approximated with the red ball)



Video credit: Priya Sundaresan

Problem Setup: solve for transformation

You're given:

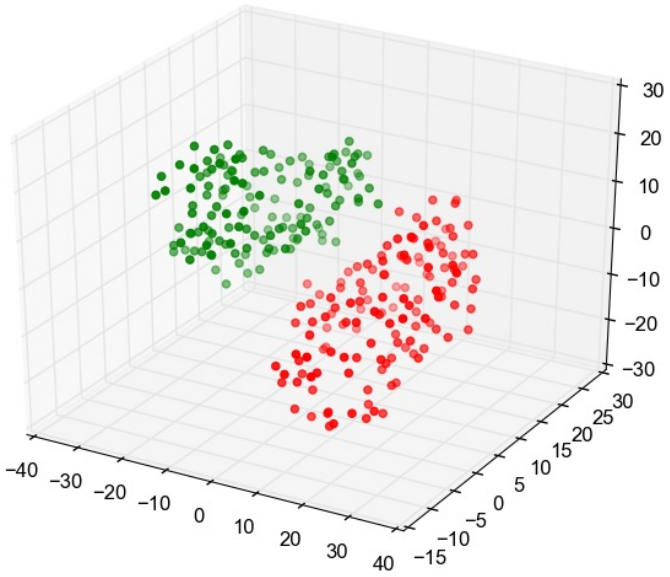
- A set of points in frame **A**: $\{\mathbf{p}_i^A\}$
- Corresponding points in frame **B**: $\{\mathbf{p}_i^B\}$

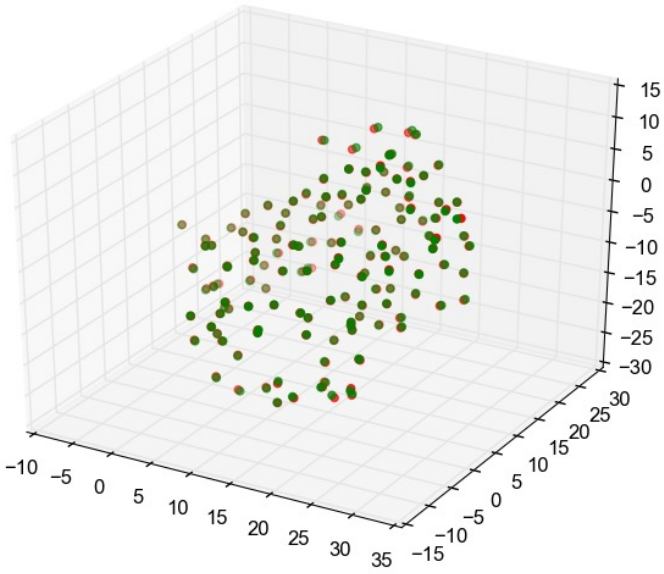
You want to find:

- A **rotation matrix** $R \in SO(3)$
- A **translation vector** $t \in \mathbb{R}^3$

  such that:

$$\mathbf{p}_i^B \approx R\mathbf{p}_i^A + t$$


30
20
10
0
−10
−20
−30
−40 −30 −20 −10 0 10 20 30 40
−15 −10 −5 0 5 10 15 20 25 30


15
10
5
0
−5
−10
−15
−20
−25
−30
−10 −5 0 5 10 15 20 25 30 35
−15 −10 −5 0 5 10 15 20 25 30

The **Kabsch-Umeyama** Algorithm

1.Compute Centroids
2.Center the Points
3.Compute Cross-Covariance Matrix
4.Apply SVD (Singular Value Decomposition)
5.Construct the Rotation Matrix
6.Compute the Translation Vector

Output the Transformation
Combine rotation and translation into a rigid body transformation

The **Kabsch-Umeyama** Algorithm

1. Compute centroids:

$$\bar{\mathbf{p}}^A = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i^A, \quad \bar{\mathbf{p}}^B = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i^B$$

---

2. Center the points:

$$\mathbf{q}_i^A = \mathbf{p}_i^A - \bar{\mathbf{p}}^A, \quad \mathbf{q}_i^B = \mathbf{p}_i^B - \bar{\mathbf{p}}^B$$

The **Kabsch-Umeyama** Algorithm

3. Compute covariance matrix:

$$H = \sum_{i=1}^{N} \mathbf{q}_i^A (\mathbf{q}_i^B)^\top$$

---

4. Singular Value Decomposition (SVD):

$$H = U \Sigma V^\top$$

The **Kabsch-Umeyama** Algorithm

5. Compute rotation matrix R:

$$R = VU^\top$$

If $\det(R) < 0$, correct for reflection:

$$V[:, 3] \leftarrow -V[:, 3] \quad (\text{flip the third column of } V)$$
$$R = VU^\top$$

---

6. Compute translation vector $\mathbf{t}$:

$$\mathbf{t} = \bar{\mathbf{p}}^B - R\bar{\mathbf{p}}^A$$

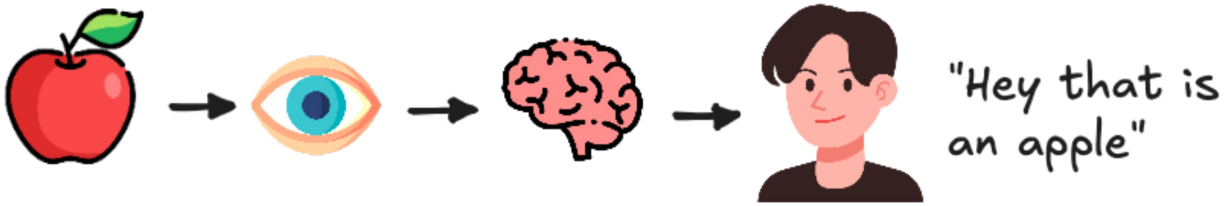
Computer Vision Basics
Image Processing + CNN Survival Kit

Slides Adapted from Stanford EE227/CS227A: Robot Perception

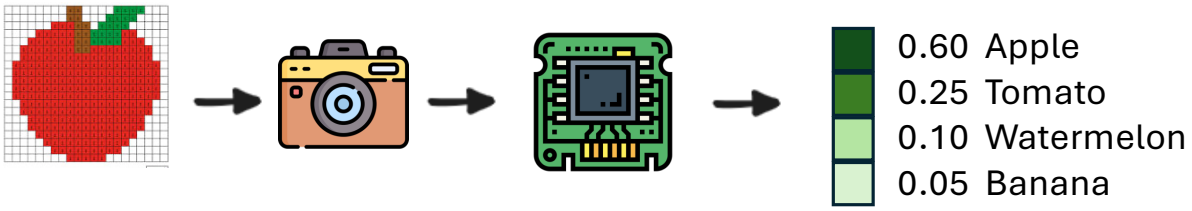
credit: Shuran Song

What is computer vision?

Human Vision

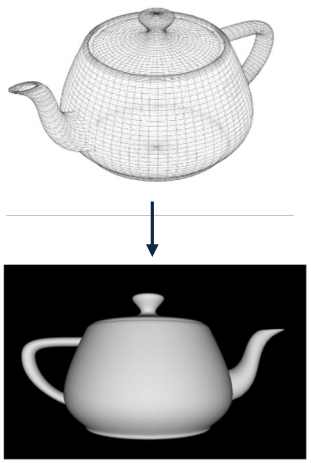


Computer Vision

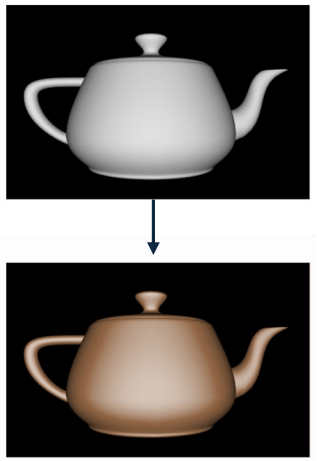


Computer Vision: Quick Overview

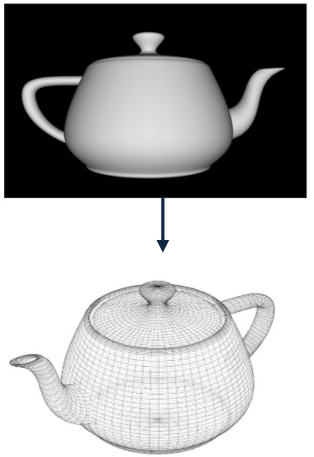
What is computer vision?



Computer Graphics
Models to Images



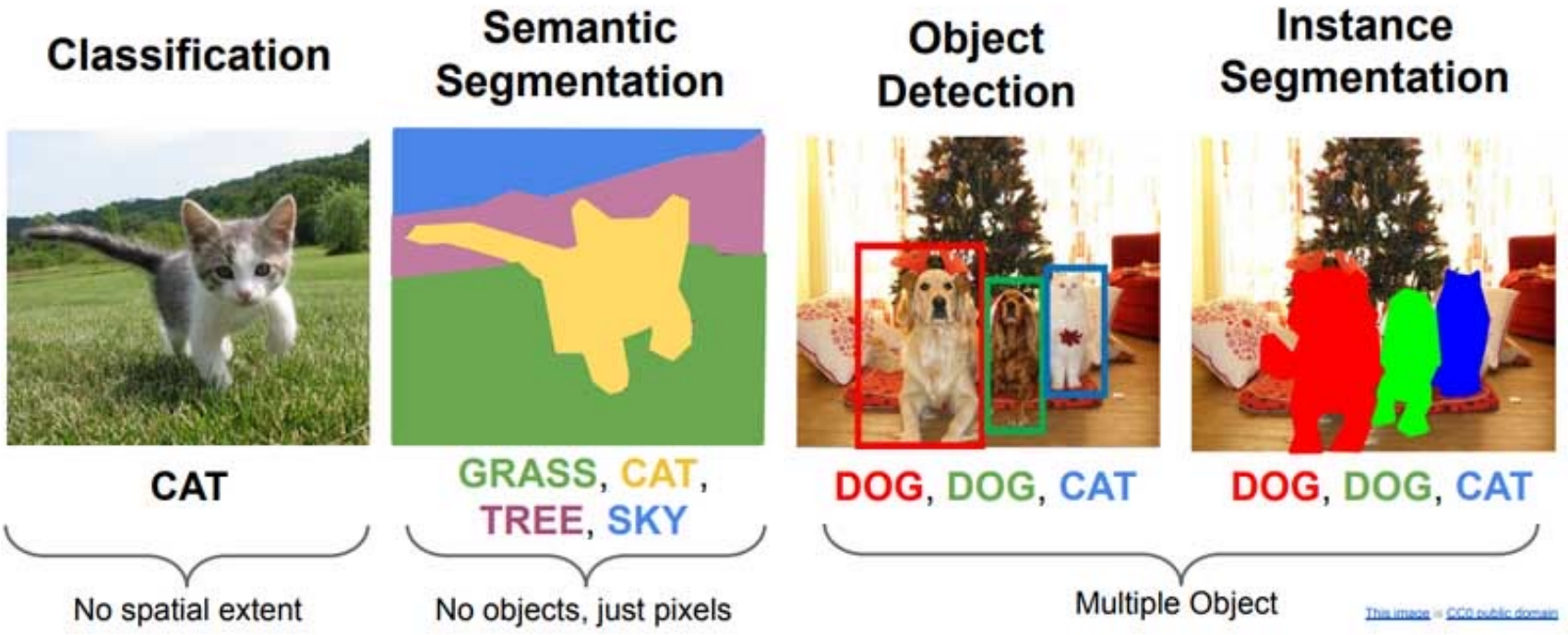
Comp. Photography
Images to Images



Computer Vision
Images to Models

Computer Vision: Quick Overview

Common 2D computer vision tasks



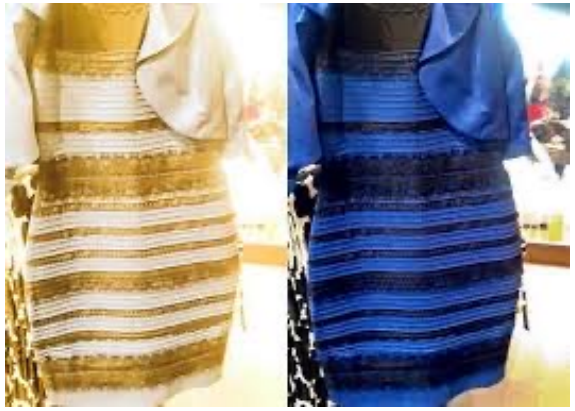
What Makes 2D Computer Vision Hard?

Variation: same cat different poses, view points ...

What Makes 2D Computer Vision Hard?

Variation: different cat with different shape, color, texture...

What Makes 2D Computer Vision Hard?

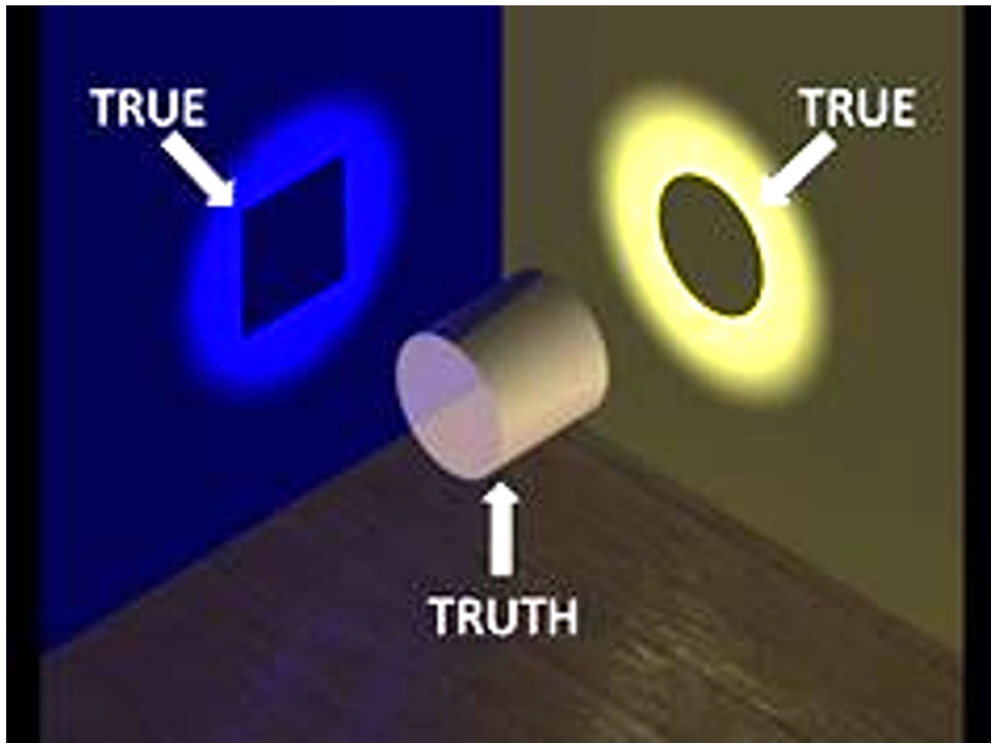


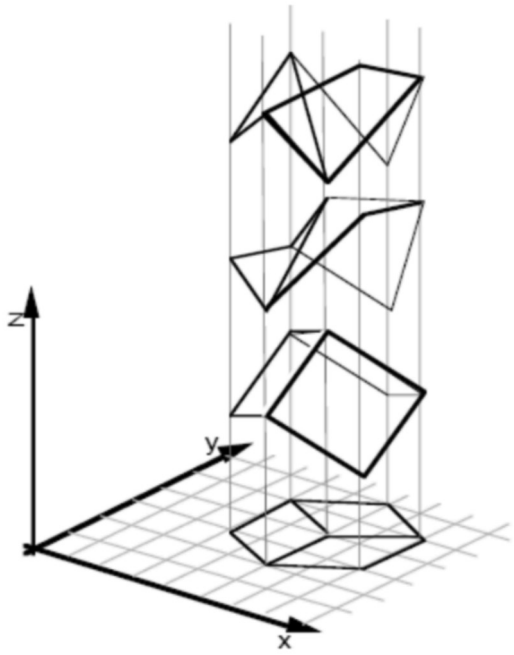
Illumination

What Makes 2D Computer Vision Hard?

Occlusion: partial observation

What Makes 2D Computer Vision Hard?

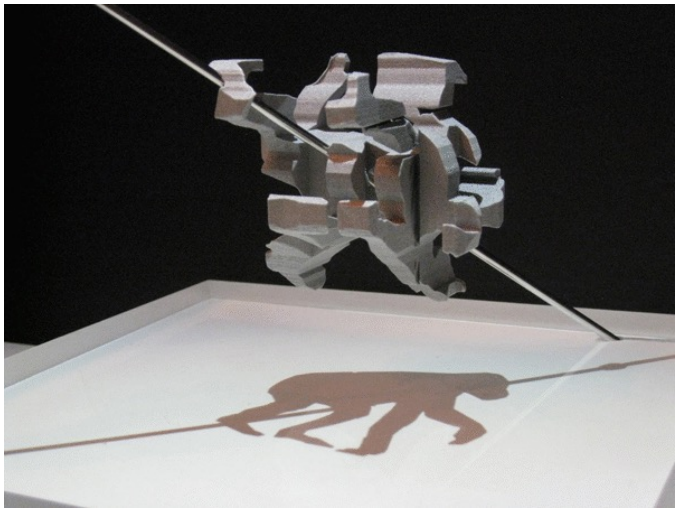




[Sinha and Adelson 1993]

Ambiguity: Ill-posed Problem

What Makes 2D Computer Vision Hard?



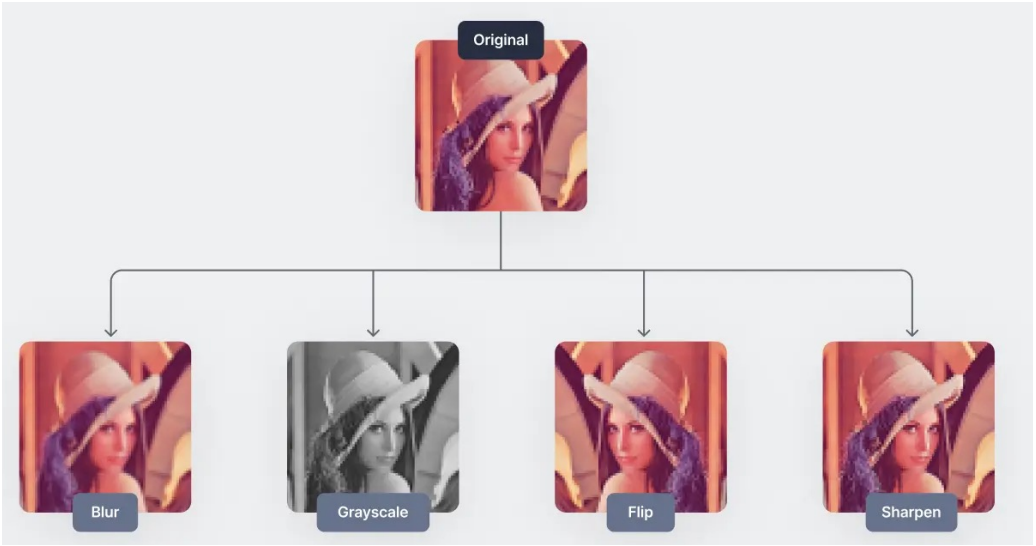
More examples: https://www.jvmuntean.com/gallery-1#photos)

Basic Image Processing

There are many operations we can apply on a image.

Some examples:

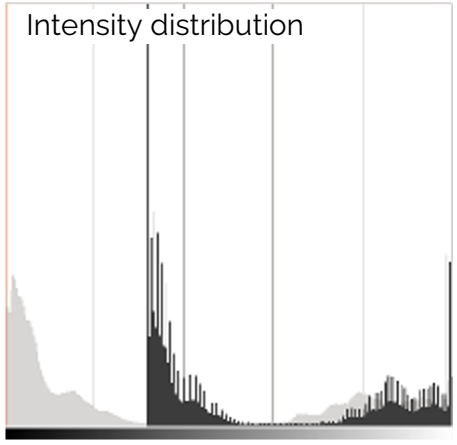
- Resize
- Color manipulation
- Contrast
- Brightness
- Gamma
- Filtering
- Denosing
- ...



Brightness (overall intensity of an image)

Adjust brightness: g(x) = f(x) + β

Increasing (/ decreasing) the β value will add (/ subtract) a constant value to every pixel. Pixel values outside of the [0 ; 1] range will be saturated (i.e. a pixel value higher (/ lesser) than 1 (/ 0) will be clamped to 1 (/ 0)).



In light gray, histogram of the original image, in dark gray when brightness = 0.15

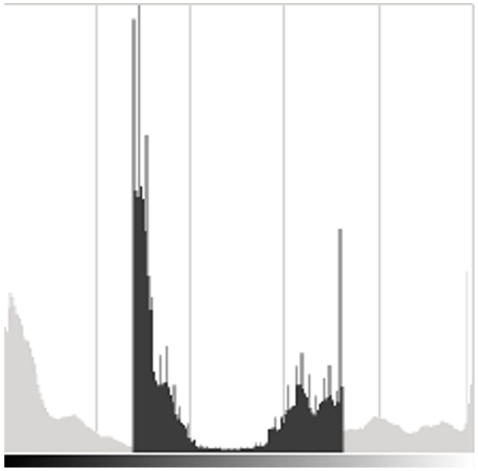
f(x): input image pixel value

g(x): output image pixel value

Contrast

Adjust contrast: $g(x) = \alpha f(x)$

The α parameter will modify how the intensity distribution spread. If $\alpha<1$, the color levels will be compressed and the result will be an image with less contrast.



In light gray, histogram of the original image, in darker gray when contrast a<1

f(x): input image pixel value

g(x): output image pixel value

Note: f(x) is normalize to have zero mean, g(x) will be convert back to (0,1) before visualization

Gamma Correction

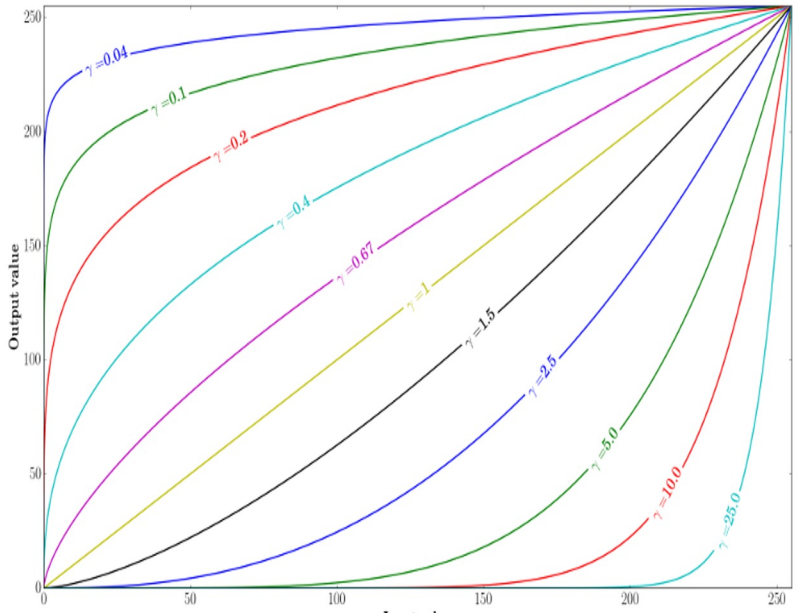
Gamma correction can be used to correct the brightness of an image by using a non linear transformation between the input values and the mapped output values:

$$g(\boldsymbol{x}) = [f(\boldsymbol{x})]^{1/\gamma}$$

As this relation is nonlinear, the effect will not be the same for all the pixels and will depend to their original value.

When γ<1, the original dark regions will be brighter and the histogram will be shifted to the right. (reducing contrast)

Y: output value



X: input value

Gamma Correction

The following image has been corrected with: α=1.3 and β=40. The overall brightness has been improved but the clouds are now greatly saturated due to clipping.

The following image has been corrected with: gamma correction γ=0.4.

Image Filtering

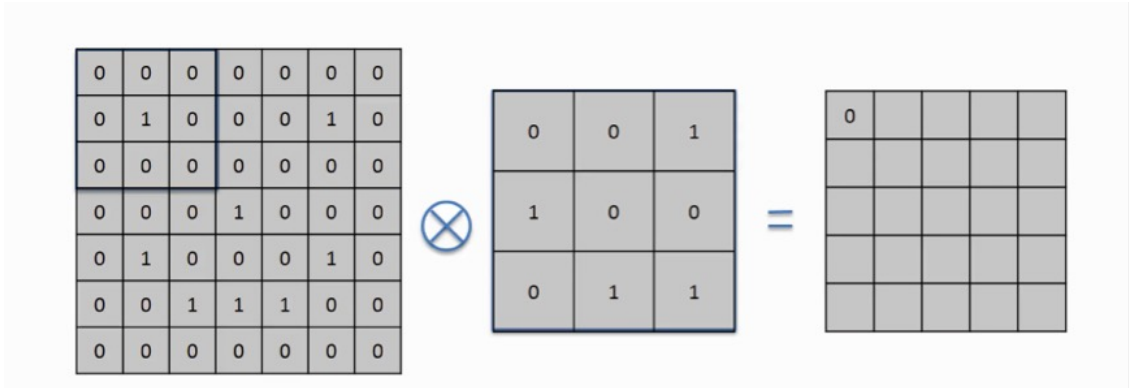
Image filtering: compute function of local neighborhood at each position

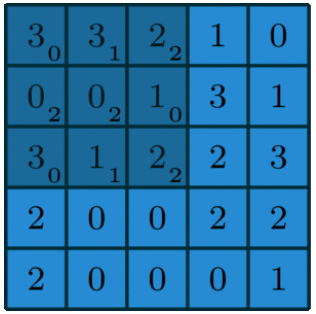
- Filtering can be used for:
  - Enhance images
    - Denoise, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching
  - Deep Convolutional Networks

As seen on Instagram...

Convolution operation

- Sliding the kernel (small 2D matrix) over the 2D image (Big 2D matrix)
- Performing elementwise multiplication with the part of the input it is currently on
- Sum up the result into a single output pixel
- Slide to the next location





Input Image kernel or filter

Padding

What about near the edge?

- the filter window falls off the edge of the image

Boundary padding

- Zero (smaller value around boundary)
- Circular/warp (best for panorama)
- Replicate
- Symmetric

Zero Pad

Circular

Replicate

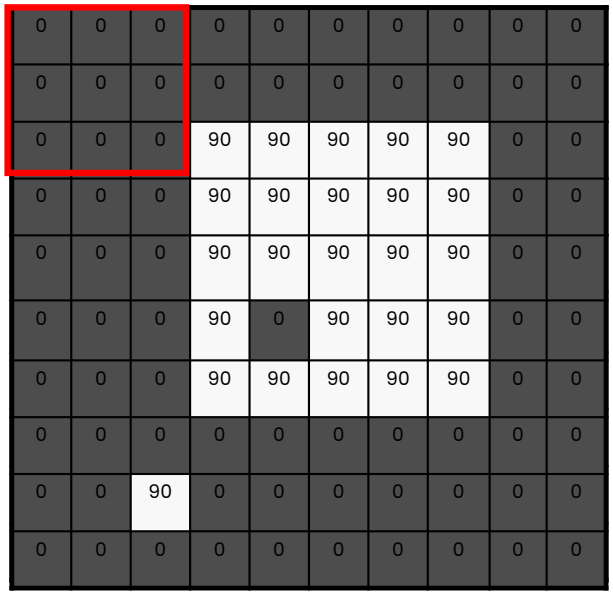
Symmetric

Image Filtering (step by step)

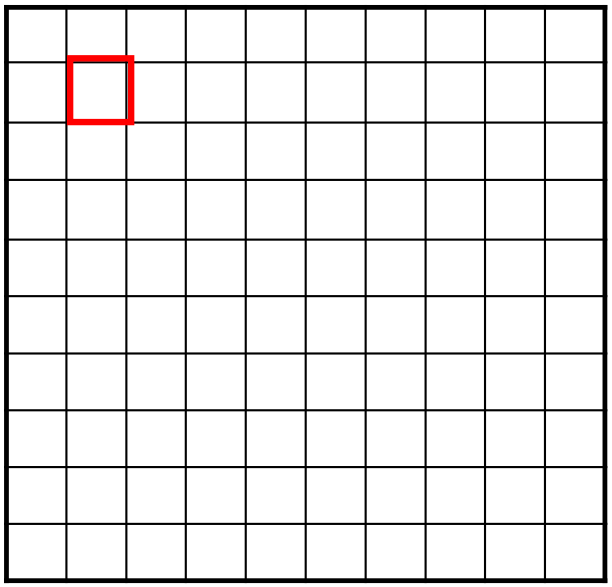
$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$h[.,.]$



Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

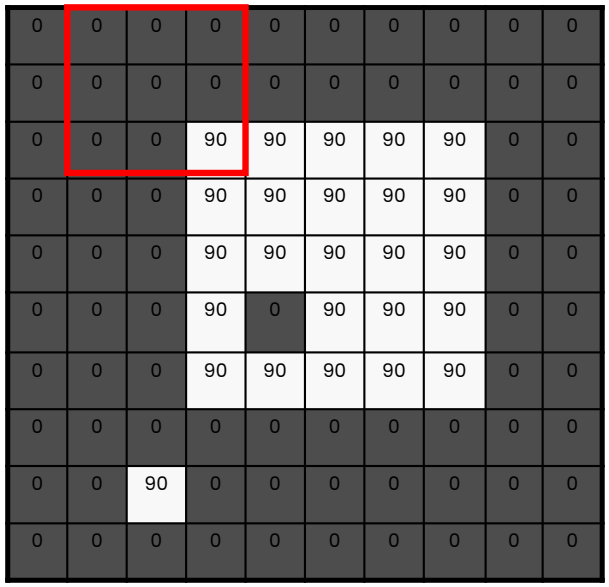
Credit: S. Seitz

Image Filtering

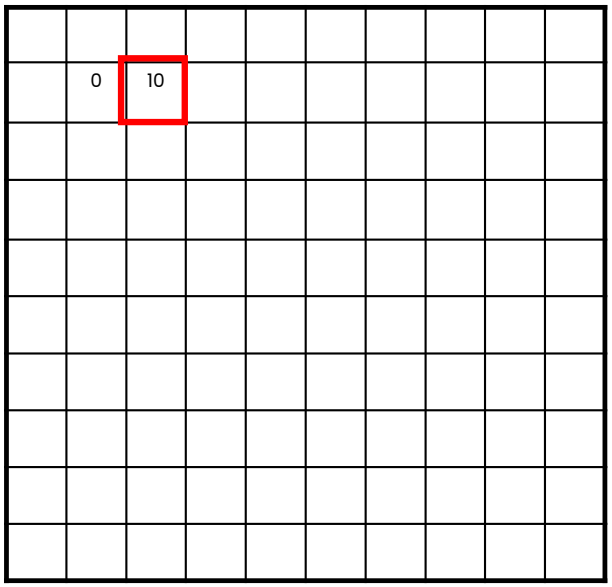
$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$h[.,.]$



Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

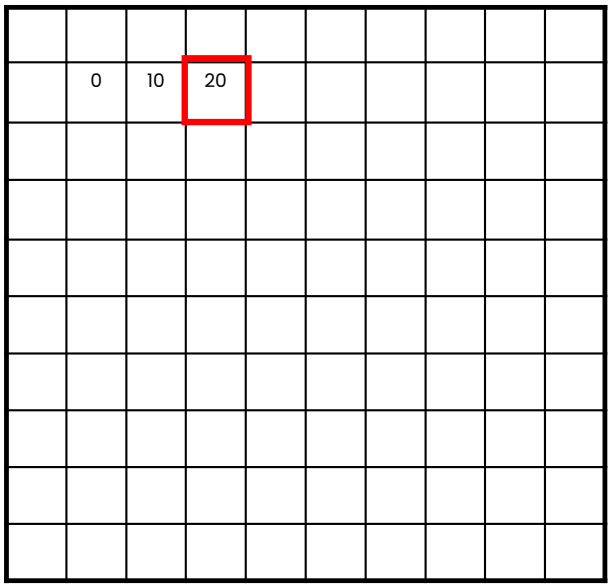
Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$h[.,.]$



Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30					

Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

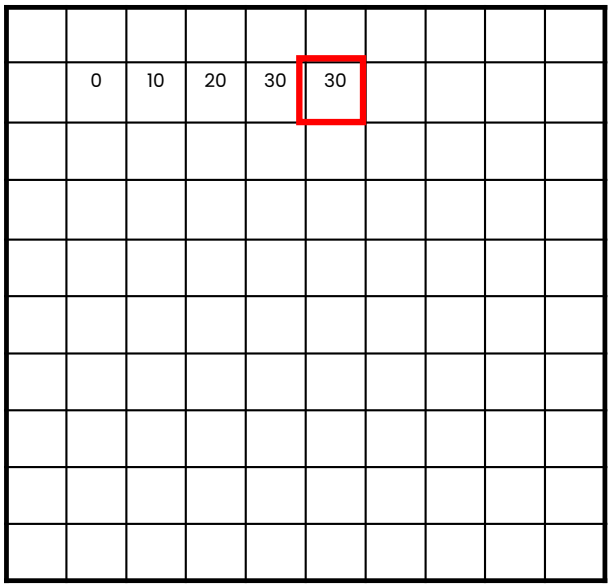
Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$h[.,.]$



Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
				?					

Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

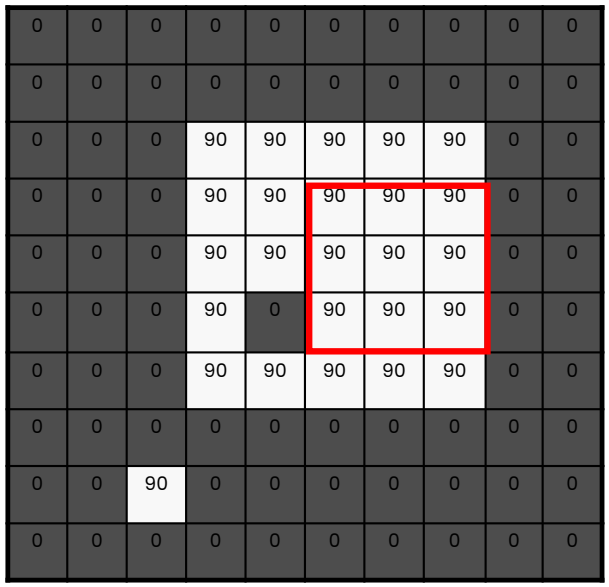
Credit: S. Seitz

Image Filtering

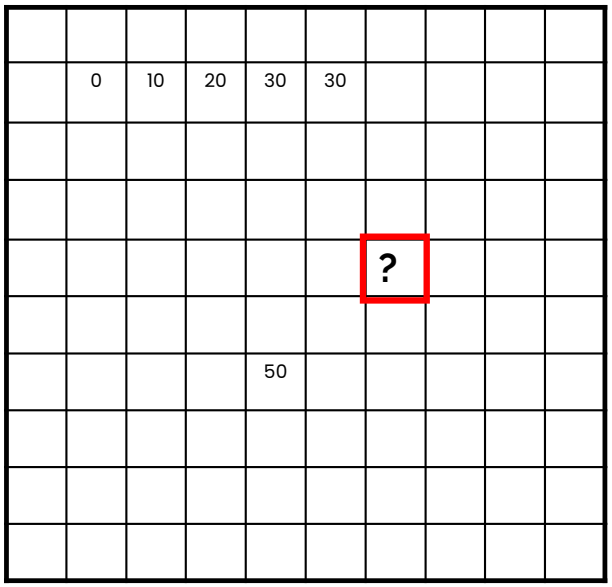
$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$h[.,.]$



Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Dot product at each position

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Credit: S. Seitz

Image Filtering: Box filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

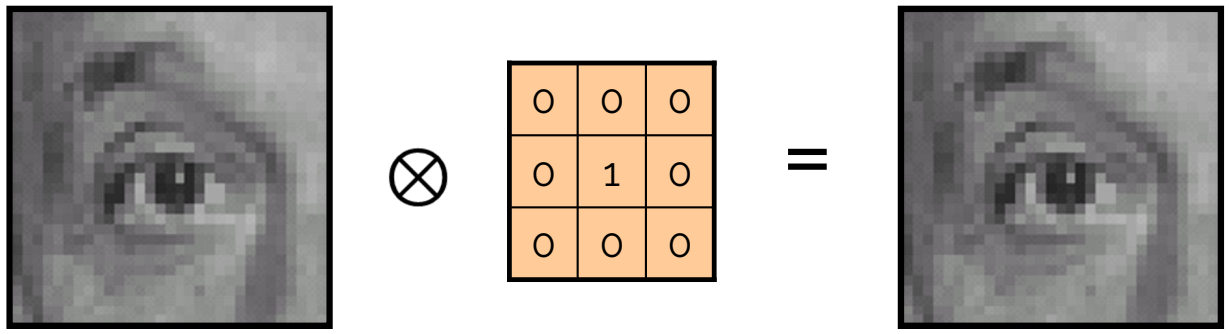
$$g[\cdot,\cdot]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

Smoothing with Box filter

Practice with Linear filter

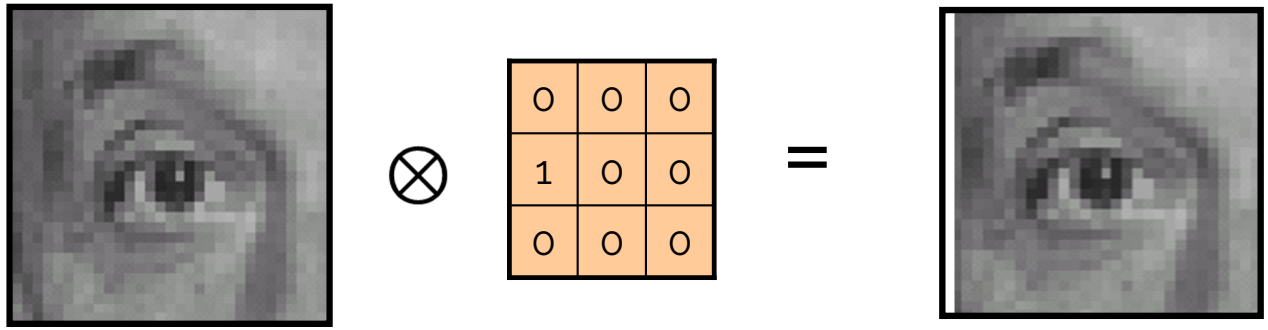


Original

Identical image

Source: D. Lowe

Practice with Linear filter

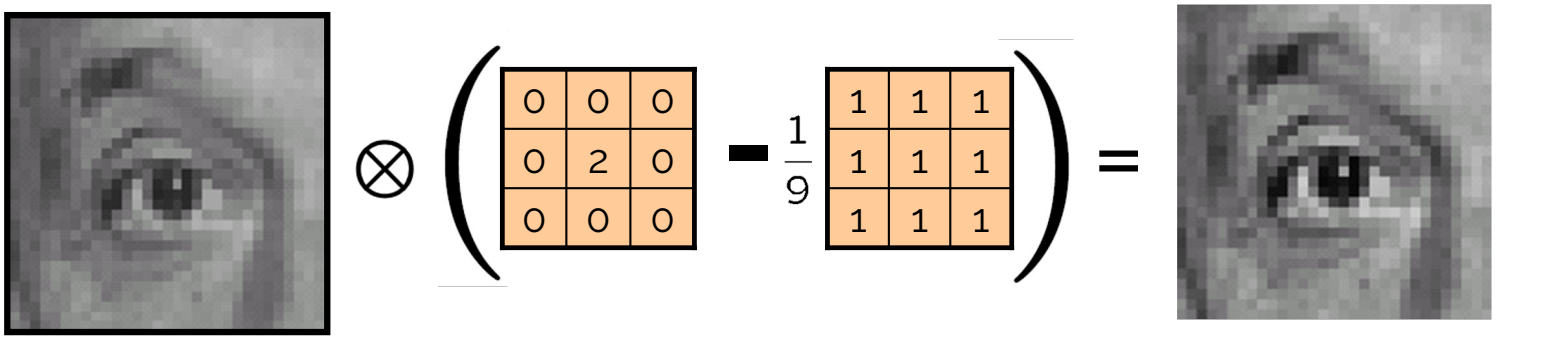


Original

Shifted right
By 1 pixel

Source: D. Lowe

Practice with Linear filter

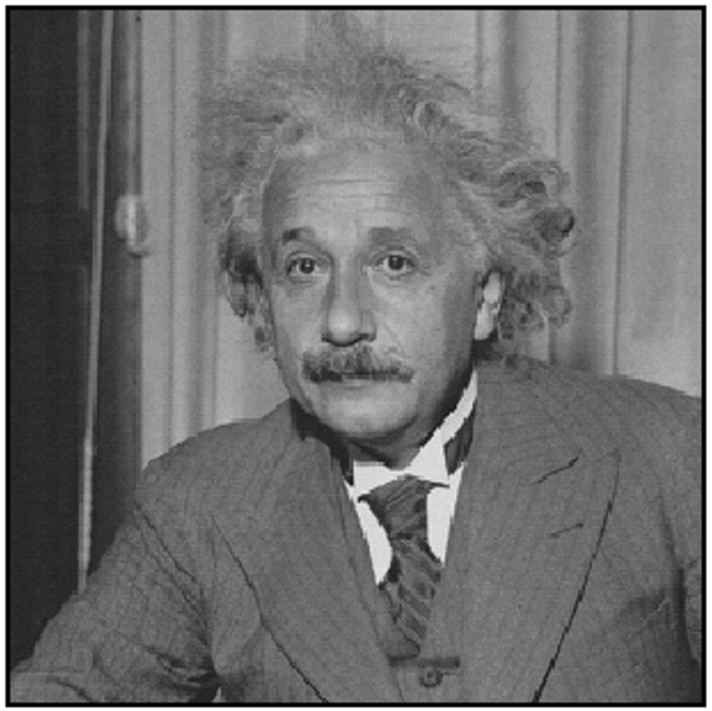


Original

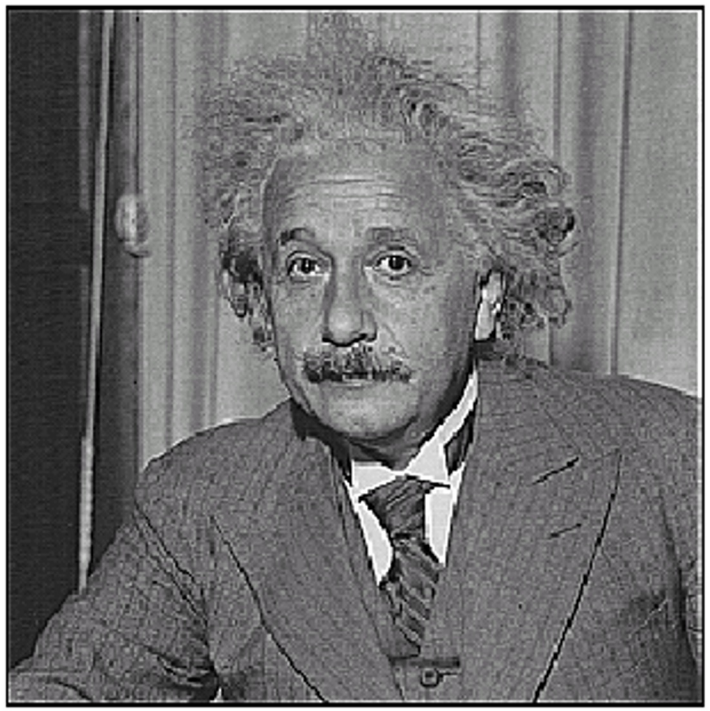
Sharpening filter
(Amplifies the difference)

Source: D. Lowe

Sharpening



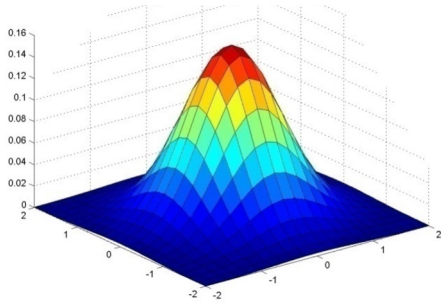
before

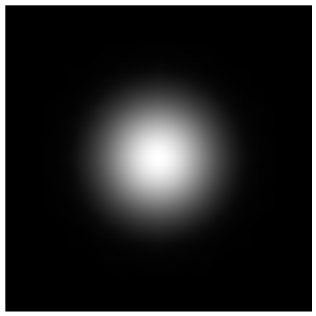


after

Source: D. Lowe

Other Filters: Gaussian Kernel





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

Constant factor at front makes matrix sum to 1

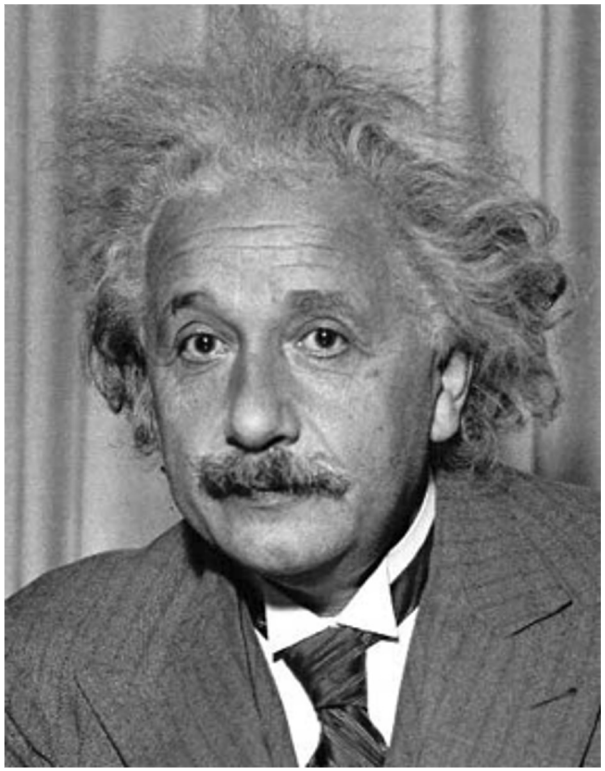
Approximated by:

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

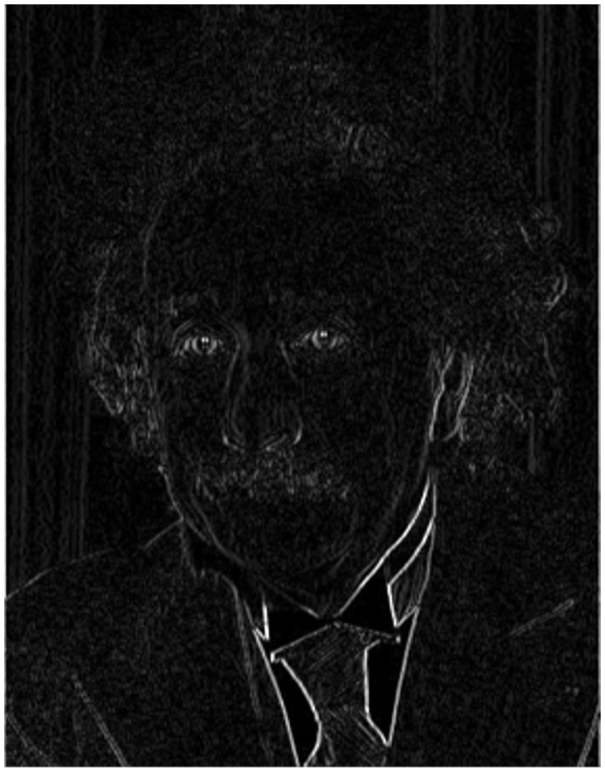
Source: C. Rasmussen

Other Filters: Edge



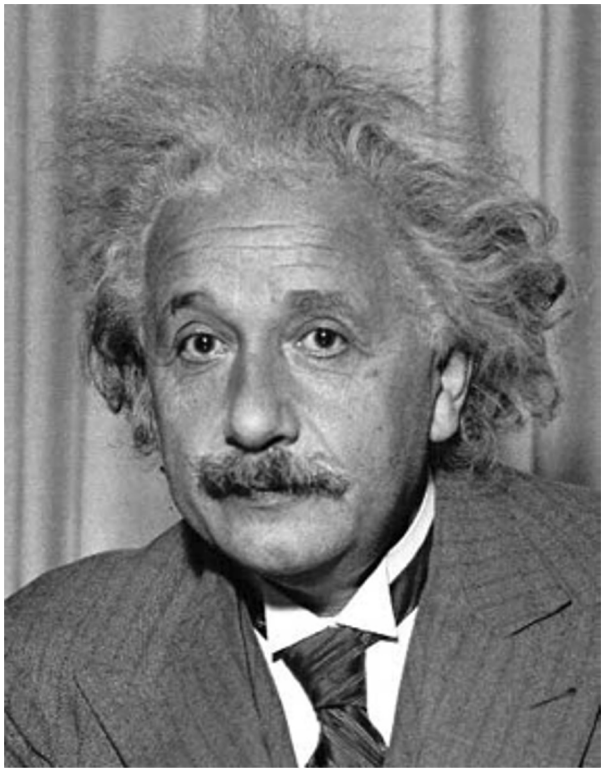
1	0	-1
2	0	-2
1	0	-1

Vertical
Sobel



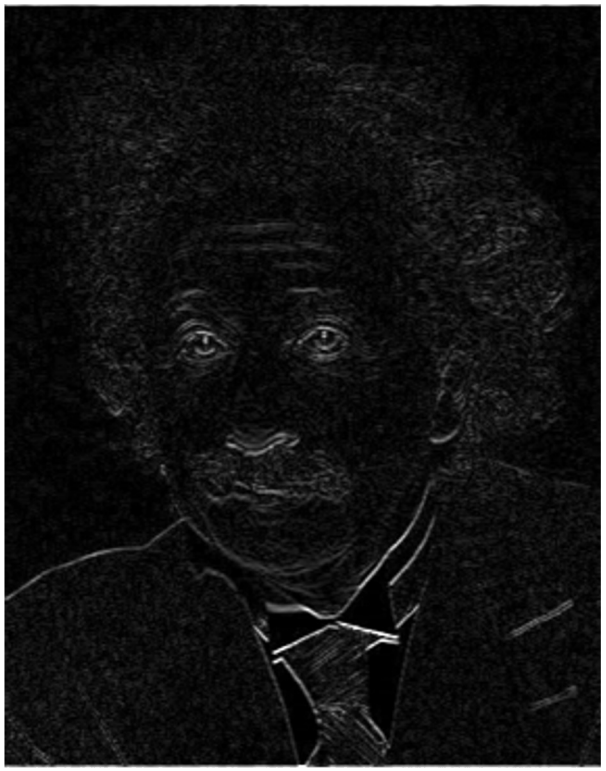
Vertical Edge
(absolute value)

Other Filters: Edge



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

Horizontal Sobel

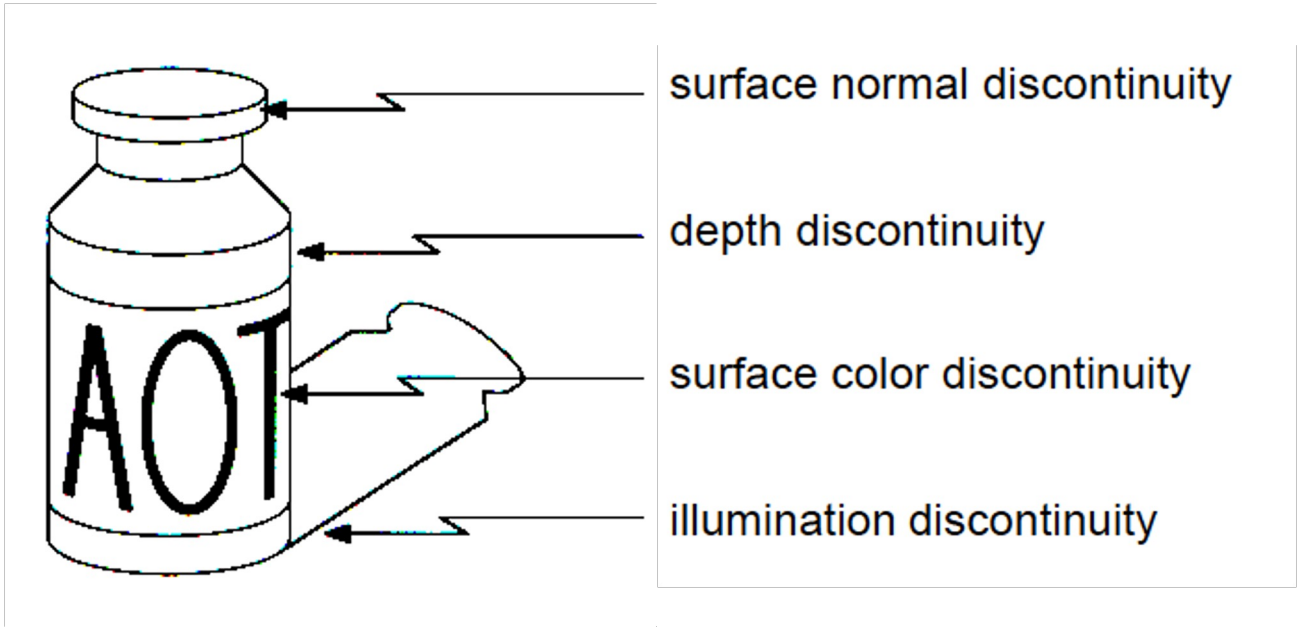


Horizontal Edge (absolute value)

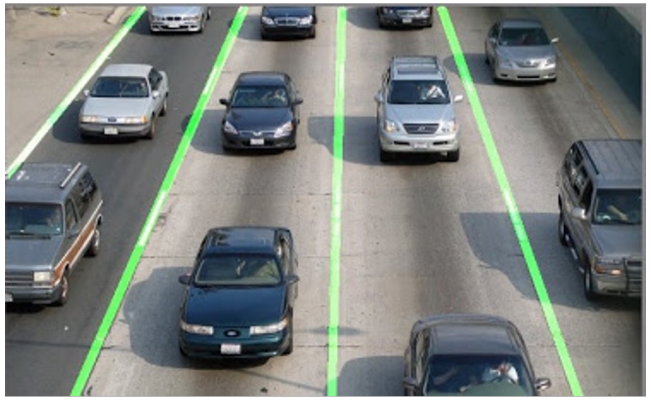
Edge Detection

Goal: Identify sudden changes (discontinuities) in an image

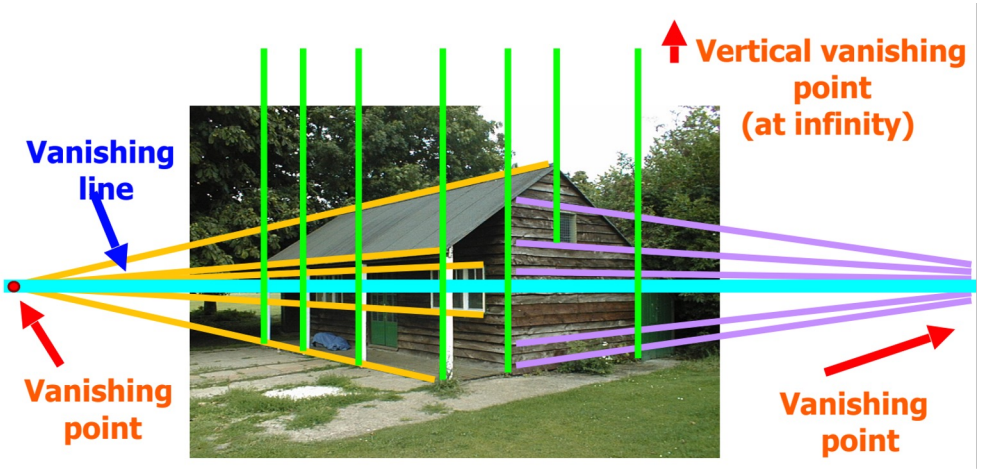
Why? Intuitively, edges carry a lot of the semantic and shape information from the image



Edge Detection in Robotics



Extract information Lane detection



geometry and camera viewpoint

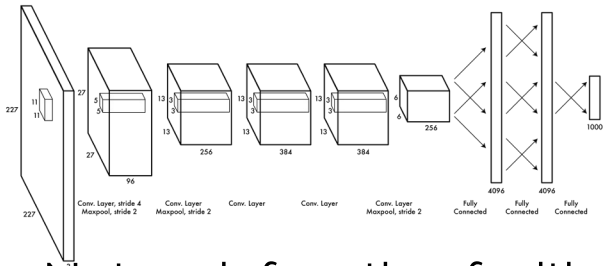
Convolutional Neural Network (CNN)

Can we automatically ***learn*** which filter to apply to get most useful information?

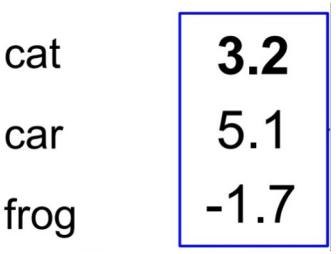
Convnet Building Blocks

Input Image:
x_i

$$s = f(x_i; W)$$



Network function f with weights W.

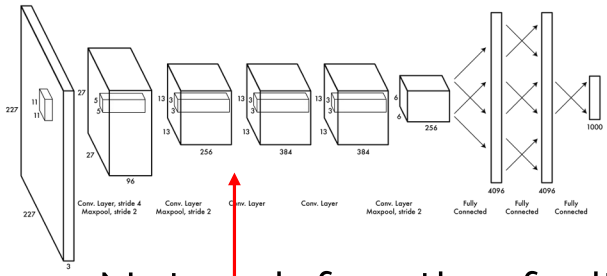


Output: s is a vector of classification scores

Convnet Building Blocks

Input Image:
x_i

$$s = f(x_i; W)$$



Network function f with weights W.

cat	**3.2**	**1**
car	5.1	0
frog	-1.7	0

Groundtruth Label

Network Layers (function approximator):
Convolutional layers
Pooling layers
Fully connected layers

Losses (objectives):
Regression
Classification

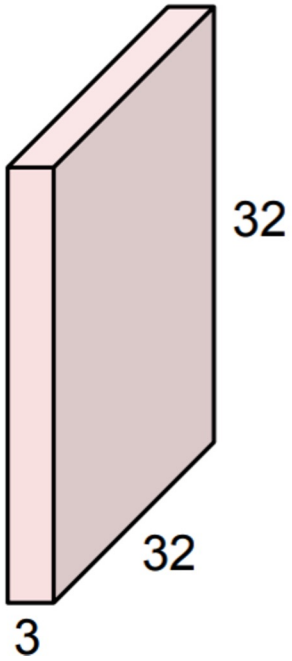
Convolutional Layer

Input: an image
Processing: convolution with **multiple** filters
Output: an image, # channels = # filters, weighted sum of input

torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *padding_mode='zeros'*)

Convolutional Layer

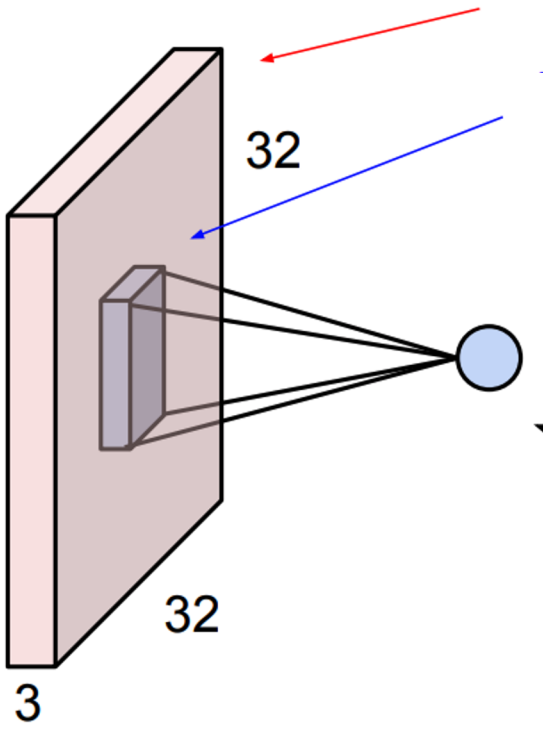
3x 32 x32 image



3x 5 x 5 filter

Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Layer



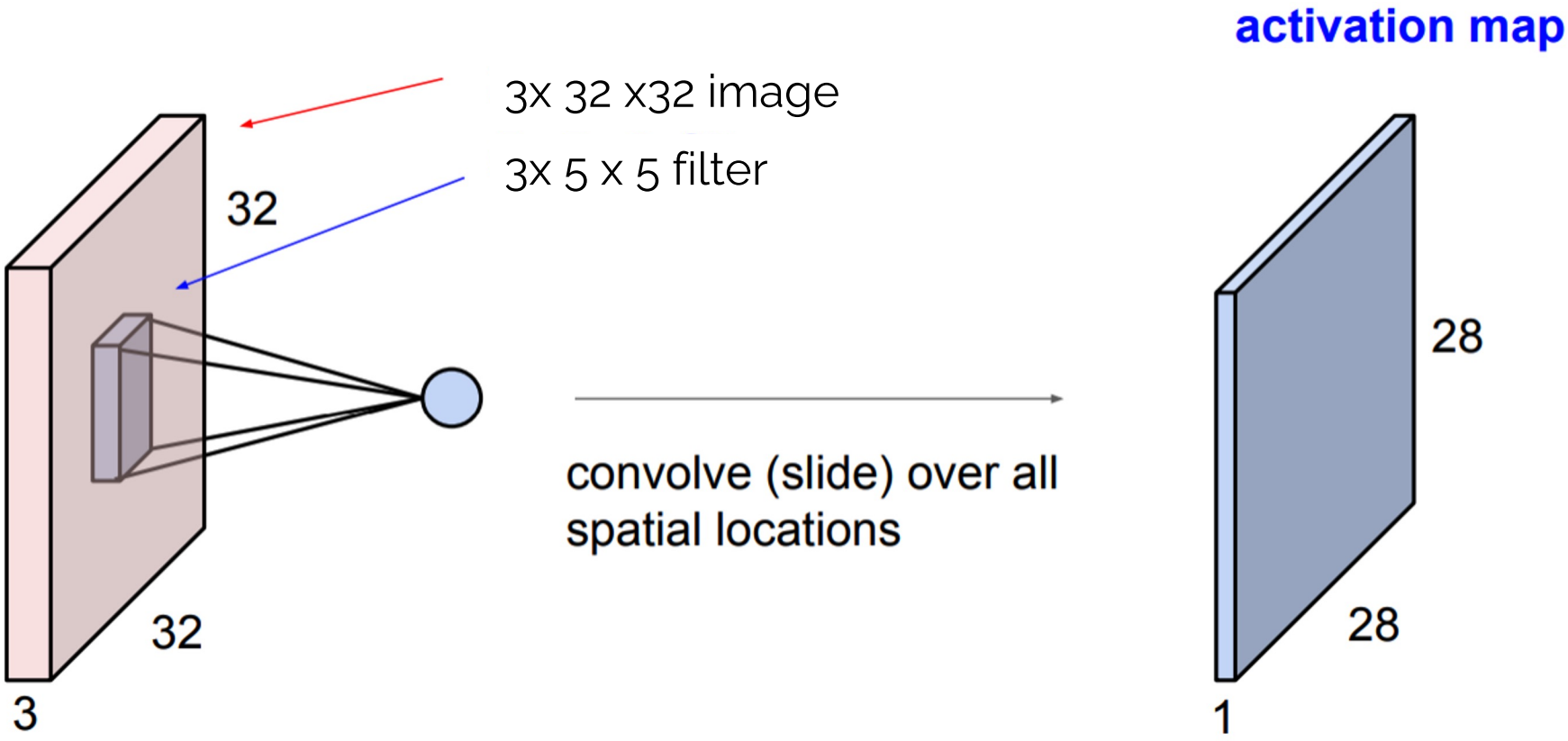
3x 32 x32 image

3x 5 x 5 filter

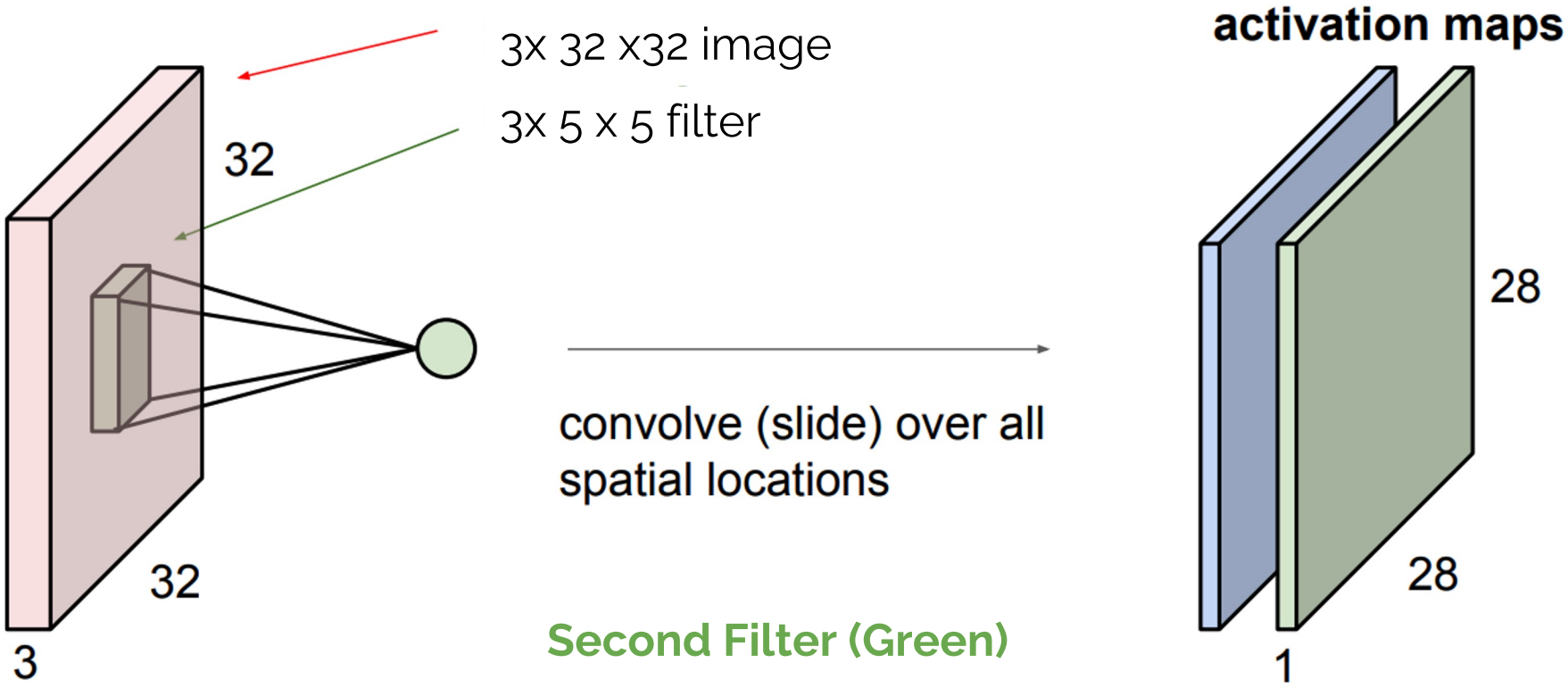
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

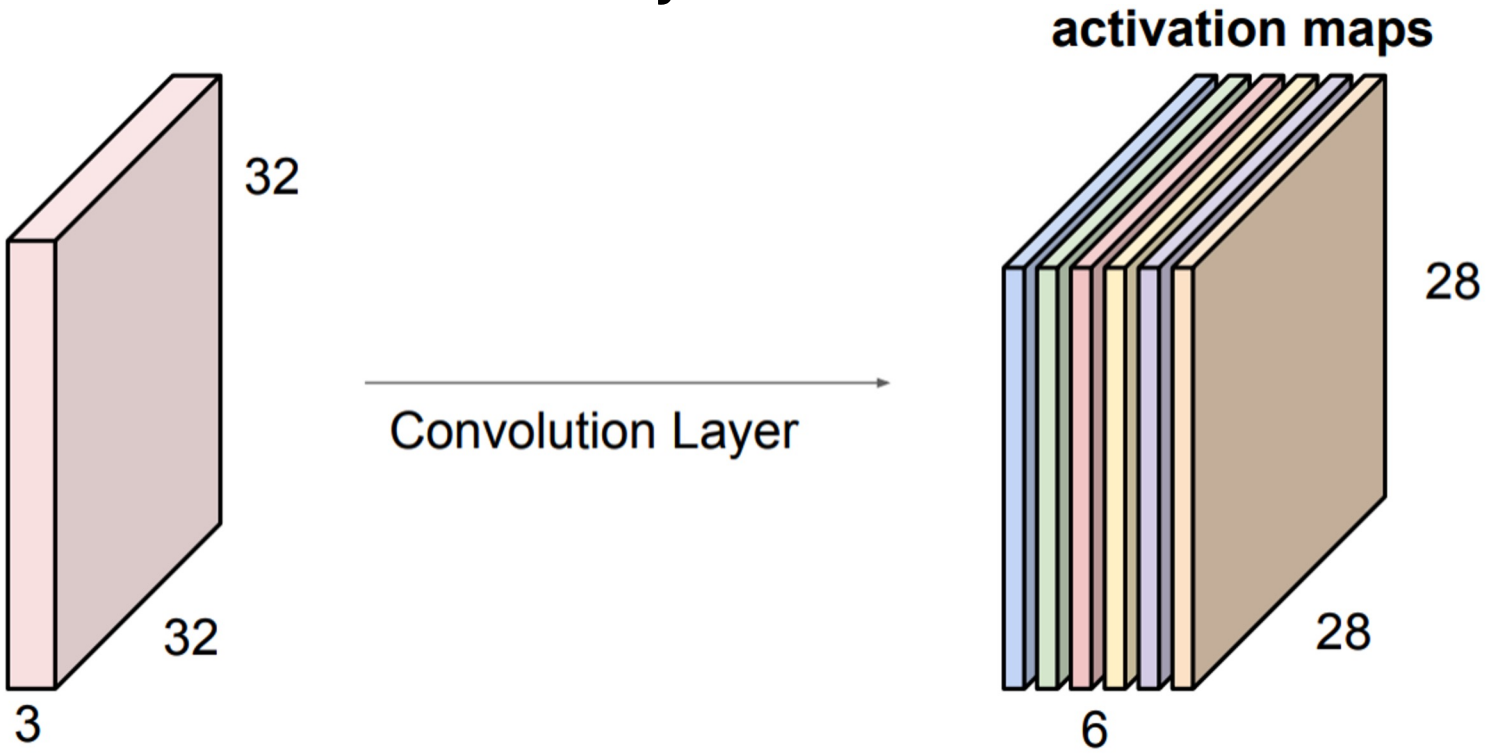
Convolutional Layer



Convolutional Layer



Convolutional Layer



If we had **6** 5x5 filters, we'll get 6 separate activation maps:
We stack these up to get a "new image" of size 28x28x6!

Kernel size

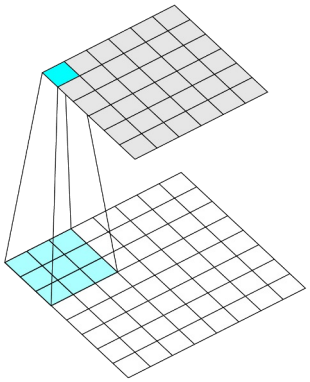
How big the filter for a layer is

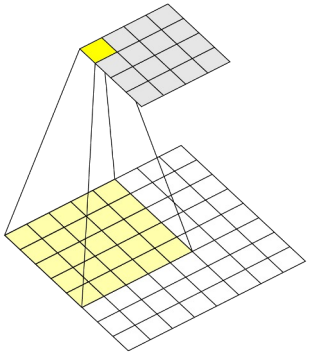
Typically 1 x 1 <-> 7x7

Larger kernel aggregate information from **larger** local region (**bigger receptive field**)

1 x 1 is just linear combination of channels in previous image (no spatial processing)

Filters usually have same number of channels as input image.



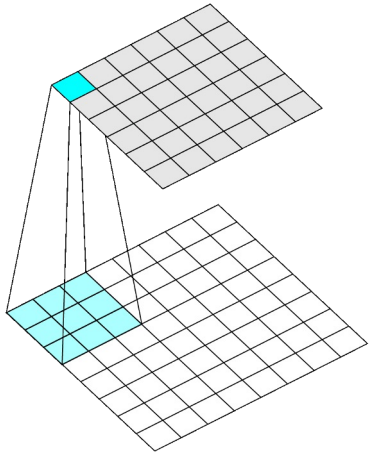


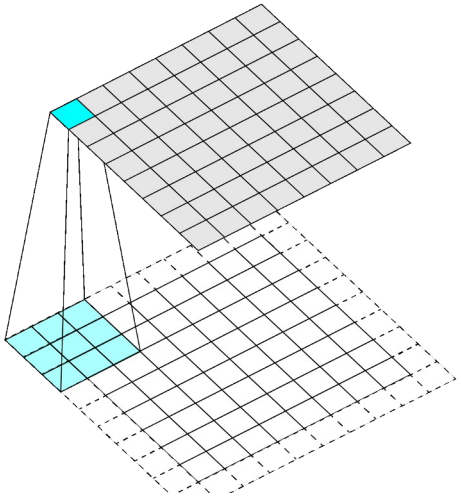
Padding

Convolutions have problems on edges

Do nothing: output a little smaller than input

Pad: add extra pixels on edge



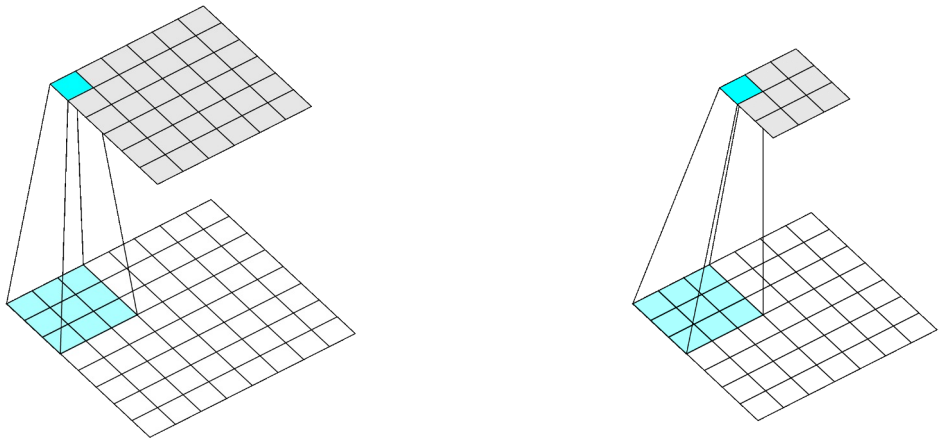


Stride

How far to move filter between applications

We've done stride 1 convolutions up until now, approximately preserves image size

Could move filter further, downsample image



Input - output size

- Input C_{in}xH_{in}xW_{in} image
- Conv2d(K,N,S,P): Number of kernel K, kernel size [NxN], stride: s, padding: p
- What's the output feature size (C_{out},H_{out},W_{out}) ?

$H_{out} = \lfloor \frac{H_{in} + 2 * padding - (N-1) - 1}{stride} + 1 \rfloor$

$W_{out} = \lfloor \frac{W_{in} + 2 * padding - (N-1) - 1}{stride} + 1 \rfloor$

$C_{out} = K$

- Ex: Input: 3x200x300 Conv2d (18,3,1,1) Output size?

Input - output size

- Input C_{in}xH_{in}xW_{in} image
- Conv2d(K,N,S,P): Number of kernel K, kernel size [NxN], stride: s, padding: p
- What's the output feature size (C_{out},H_{out},W_{out}) ?

$$H_{out} = \lfloor \frac{H_{in} + 2 * padding - (N-1) - 1}{stride} + 1 \rfloor$$

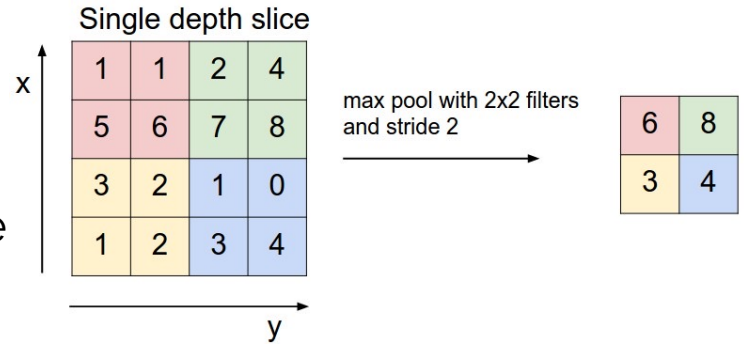
$$W_{out} = \lfloor \frac{W_{in} + 2 * padding - (N-1) - 1}{stride} + 1 \rfloor$$

$$C_{out} = K$$

- Ex: Input: 3x200x300 Conv2d (18,3,1,1) Output size?
- How about Conv2d (18,5,1,2)?

Pooling Layer

Input: an image
Processing: pool pixel values over region
Output: an image, shrunk by a factor of the stride



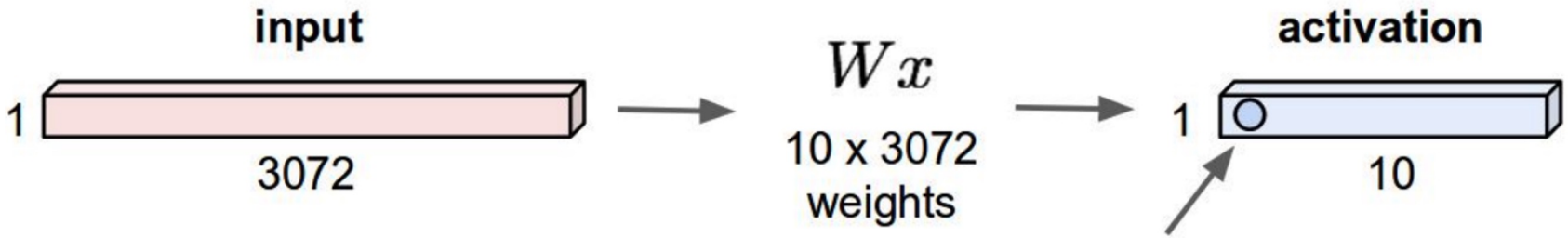
Hyperparameters:

What kind of pooling? Average, mean, max, min
How big of stride? Controls downsampling
How big of region? Usually not bigger than stride

Most common: maxpooling 2x2 stride of 2 or 3x3 stride of 3

Fully Connected Layer (InnerProduct)

Every input neuron connects to every output neuron



Often used to go from image feature map -> final output or map image features to a single vector

Eliminates spatial information

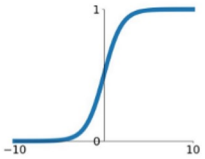
Activation Layer

Used to increase non-linearity of the network without affecting receptive fields of the conv layers.

Most commonly used activation layer: ReLU

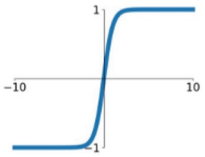
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



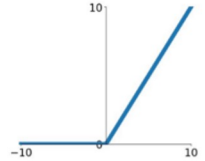
tanh

$\tanh(x)$



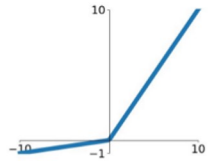
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

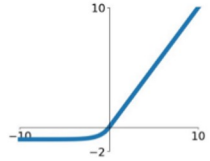


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

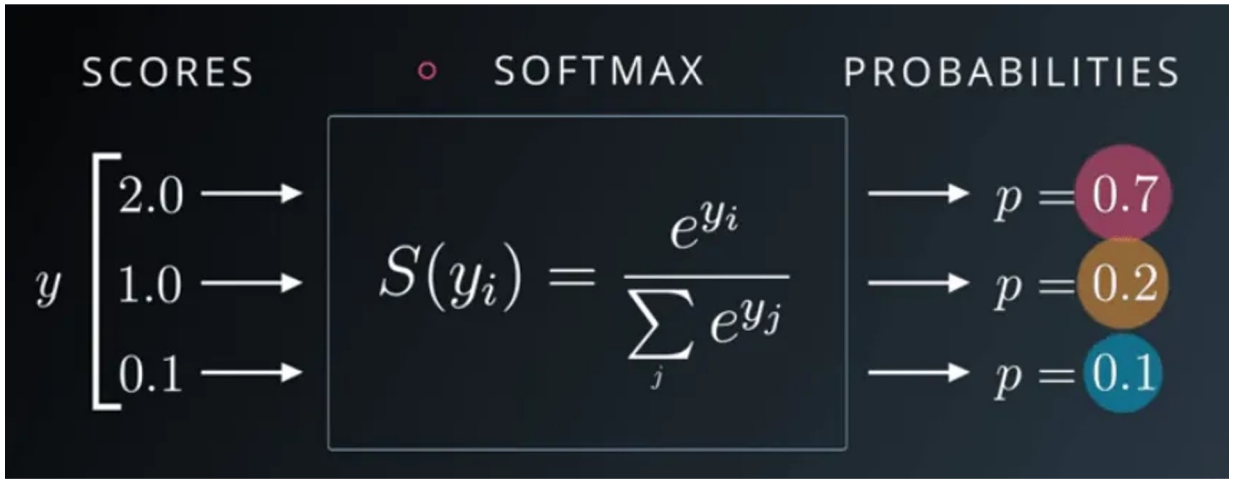
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Layer: Softmax

A special kind of activation layer, usually at the end of fully connected layer and before cross-entropy loss.

Can be viewed as a fancy normalization function that produce a probability distribution vector.



Loss Function

A loss function scores how far off a prediction is from the desired “target” output.

- loss(prediction,target) returns a number called “the loss”
- Big Loss = Bad Error
- Small Loss = Minor Error
- Zero Loss = No Error

Loss Function

Regression

Predict real-valued output

What temperature will it be tomorrow?

Loss function: squared error, L1 error ...

Classification

Predict category output

Will it be sunny tomorrow?

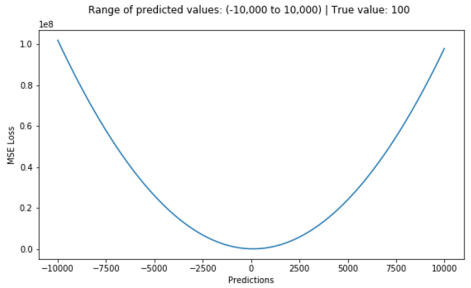
Often predicts continuous values which are probabilities of different labels, use the one with max probabilities as predicted class.

Loss function: cross-entropy

Regression

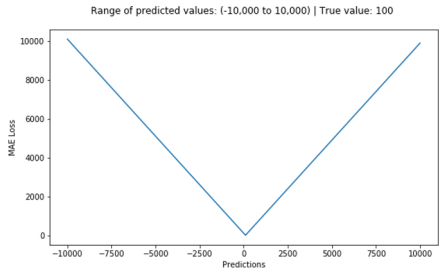
L2 Loss

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$



L1 Loss

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

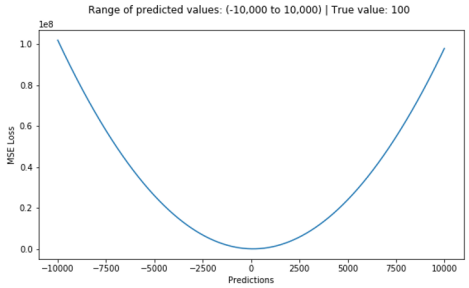


L1 loss is more robust to outliers, but its derivatives around zero are not continuous, making it inefficient to find the solution. L2 loss is sensitive to outliers, but gives a more stable and closed form solution.

Regression

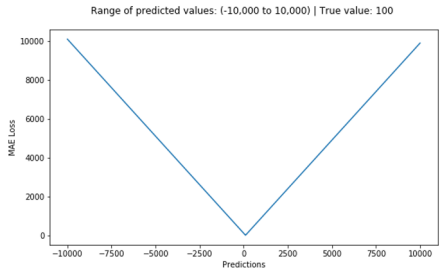
L2 Loss

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$



L1 Loss

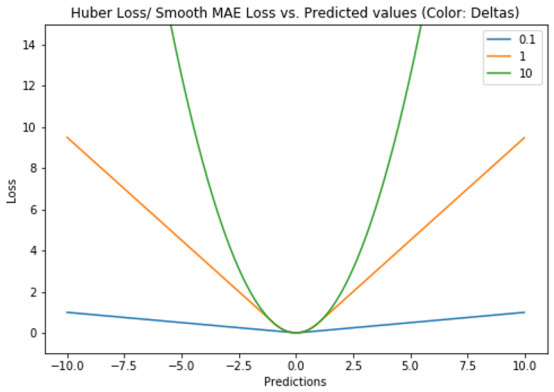
$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$



Smooth L1

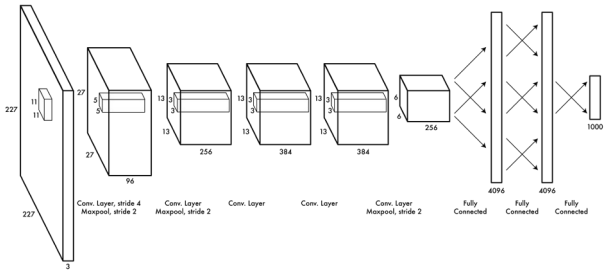
$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

L2 when δ ~ 0 and L1 when δ ~ ∞ (large numbers.)



Classification: CrossEntropy

Input Image:
x_i



Network function f with weights W.

cat	**3.2**
car	5.1
frog	-1.7

s is a vector of classification scores

1
0
0

Groundtruth Label

$$s = f(x_i; W)$$

Classification: CrossEntropy

$$s = f(x_i; W)$$

scores = unnormalized log probabilities of the classes.

cat	**3.2**
car	5.1
frog	-1.7

Network output prediction

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Classification: CrossEntropy

$$s = f(x_i; W)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

cat	**3.2**
car	5.1
frog	-1.7

Network output prediction

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Classification: CrossEntropy

$$s = f(x_i; W)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Probabilities must be >= 0

	Network output prediction	exp	unnormalized probabilities
cat	3.2		24.5
car	5.1		164.0
frog	-1.7		0.18

$$e^{s_i}$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Classification: CrossEntropy

$$s = f(x_i; W)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

	Network output prediction	exp	Probabilities must be >= 0: unnormalized probabilities	normalize	Probabilities must sum to 1: probabilities
cat	3.2		24.5		0.13
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$$\frac{e^{s_i}}{\sum_j e^{s_j}}$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Classification: CrossEntropy

$$s = f(x_i; W)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Probabilities must be >= 0

Probabilities must sum to 1

cat	3.2	24.5	0.13
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp

normalize

Network output prediction

unnormalized probabilities

probabilities

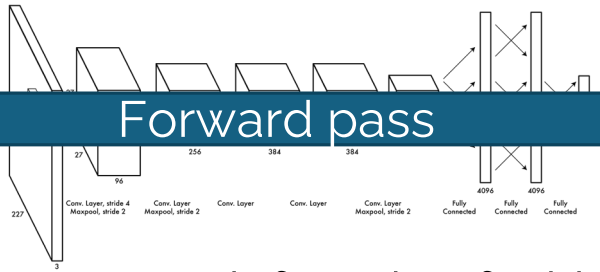
$L_i = -\log(0.13)$
$= \mathbf{2.04}$

Minimize the negative log likelihood = maximize the log likelihood of the correct class

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Training the network

Input Image:
x_i



Network function f with weights W.

$$s = f(x_i; W)$$

cat
frog

3.2
5.1
-1.7

s is a vector of classification sc

1
0
0

Groundtr uth

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Backpropagation

Weight updates through backpropagation:
change the network weight *W* so that the loss *L* is minimized

Backpropagation using gradient decent:
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

Summary of Convnet Building Blocks

Convolutional layers:
- Used convolution operation to extract features
- Convolution with stride > 1 will downsample image

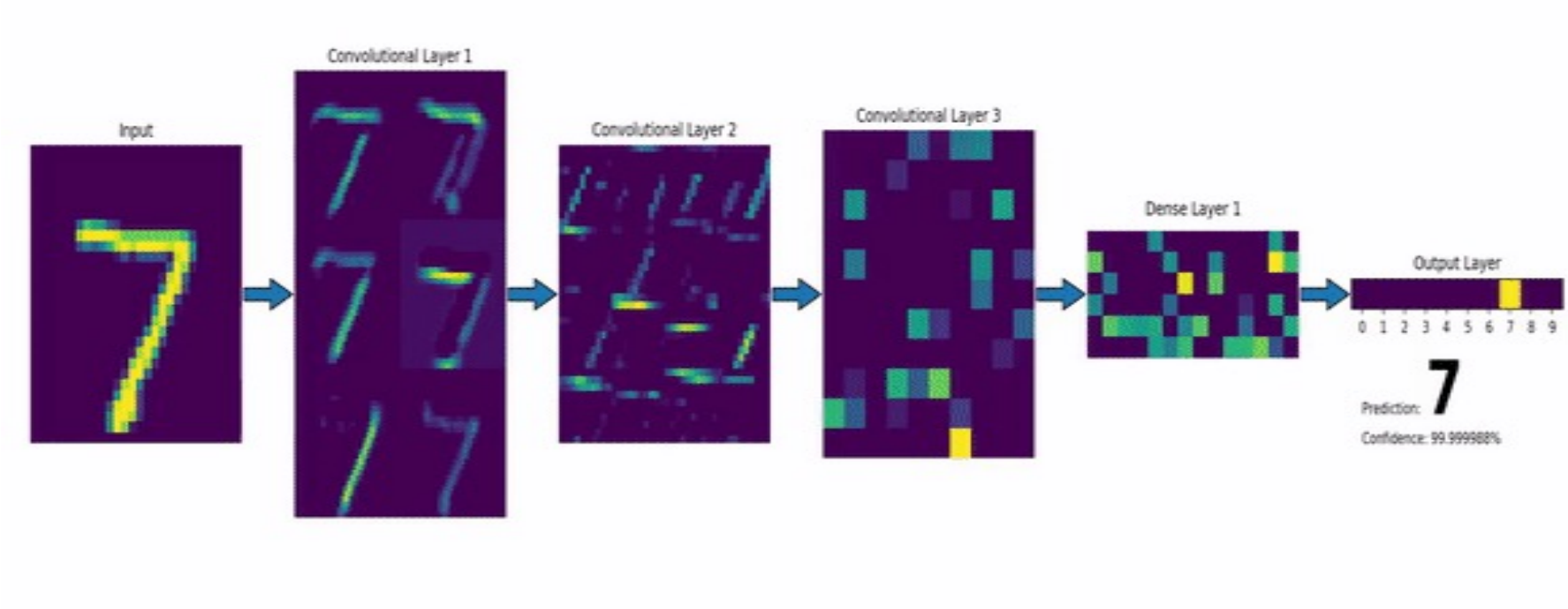
Pooling layers:
- Used to downsample feature maps, make processing more efficient

Fully connected layers:
- Often used as last layer, to map image features -> prediction
- No spatial information
- Inefficient: lots of weights, no weight sharing

Activation layers: increase non-linearity of the network

Loss function: classification and regression

MNIST Example




That’s it for today!
Questions?