

Computer Vision

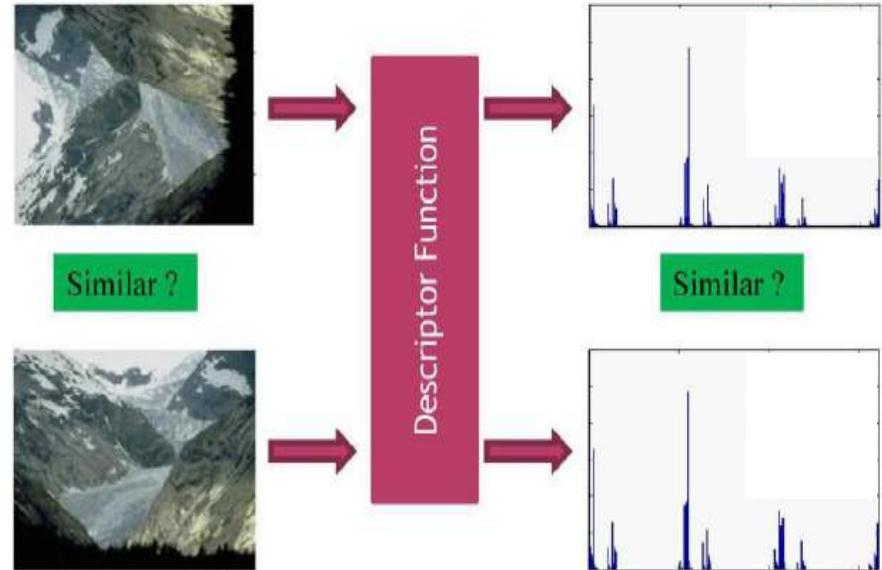
Region Detection & Local Descriptors

Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**

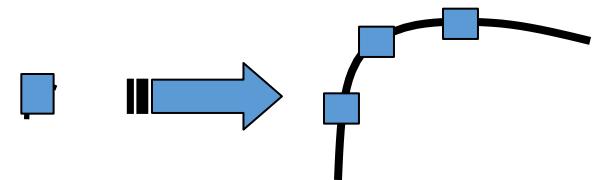


Region Detection & Local Descriptors



Previous Class

- Keypoint detection: repeatable and distinctive
 - Corners, Harris
 - Invariant to scale, rotation, etc.
- Harris Corner Detection
 - Rotation Invariant
 - Partial Intensity Change Invariant
 - *Not Invariant to Scale*

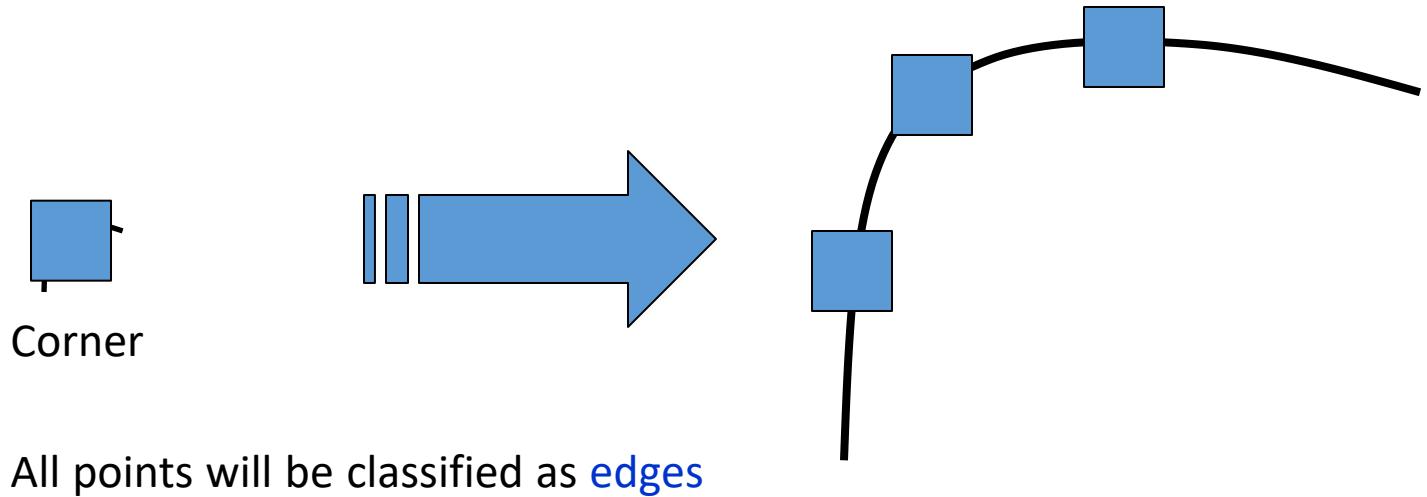


Today's class

- Scale Invariance
- Region Detection
- Local Descriptors
- Image Matching

Harris Detector: Invariance Properties

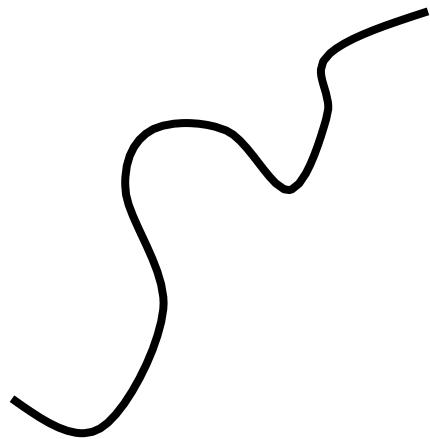
- Scaling



Not invariant to scaling

Scale invariant detection

Suppose you're looking for corners

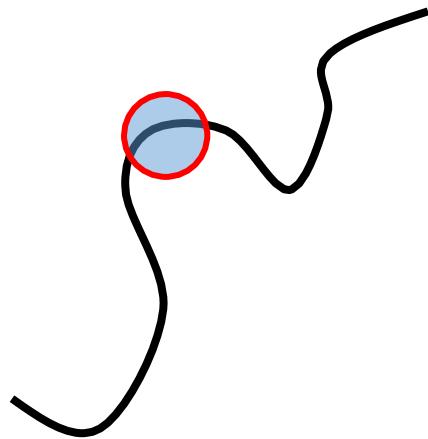


Key idea: find scale that gives local maximum of f

- in both position and scale
- One definition of f : the Harris operator

Scale invariant detection

Suppose you're looking for corners

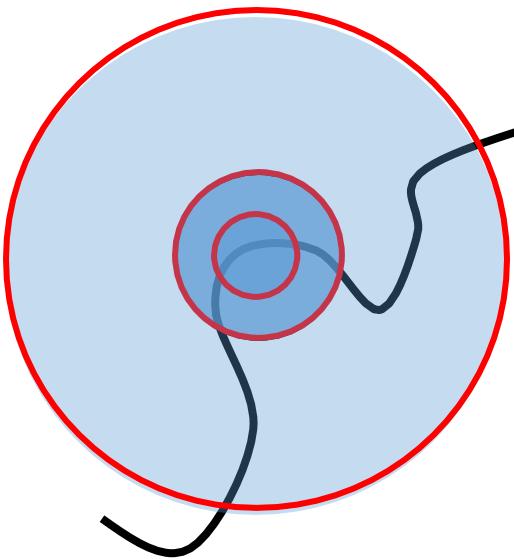


Key idea: find scale that gives local maximum of f

- in both position and scale
- One definition of f : the Harris operator

Scale invariant detection

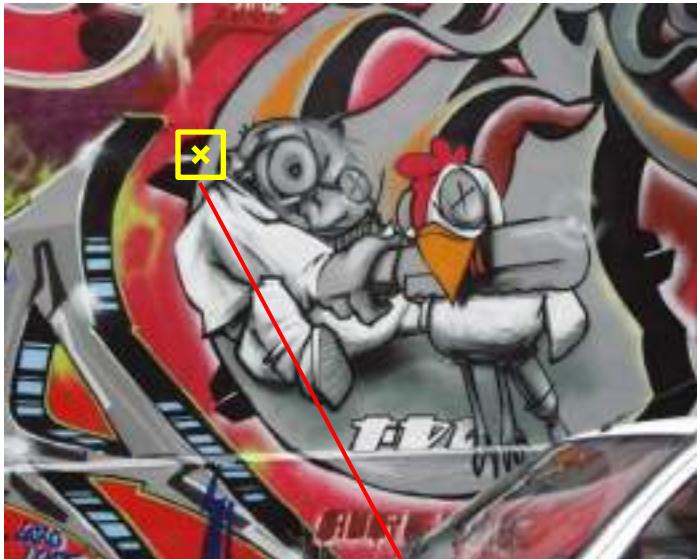
Suppose you're looking for corners



Key idea: find scale that gives local maximum of f

- in both position and scale
- One definition of f : the Harris operator

Automatic Scale Selection

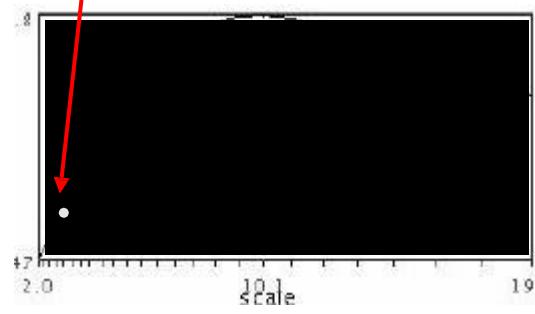
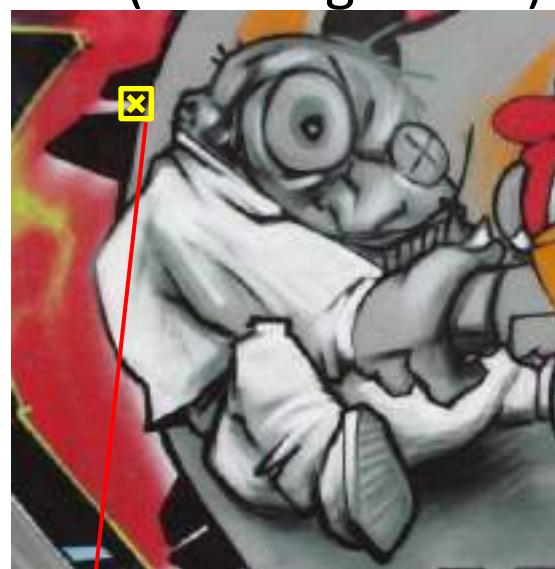
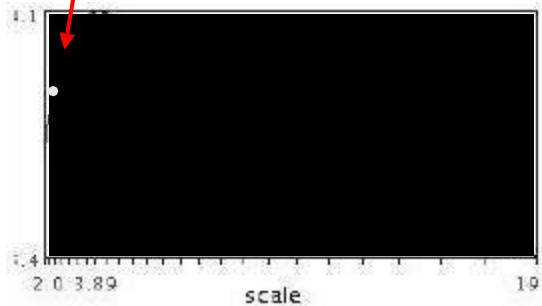


$$f(I_{i_1 \dots i_m}(x, \sigma)) = f(I_{i_1 \dots i_m}(x', \sigma'))$$

How to find corresponding patch sizes?

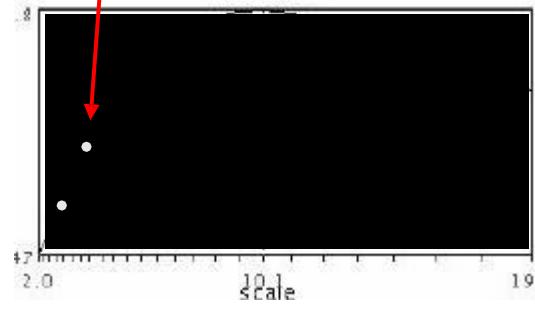
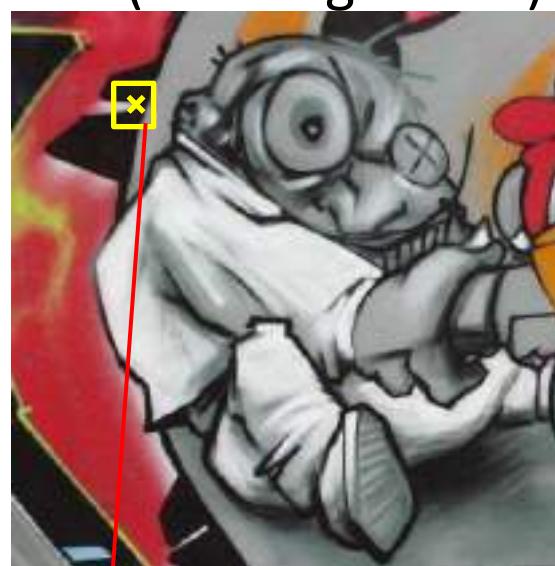
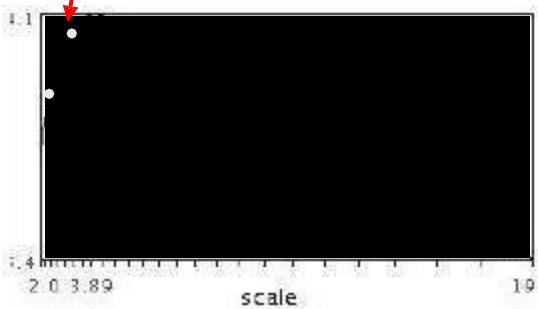
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



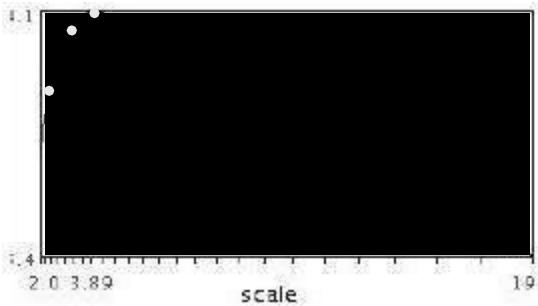
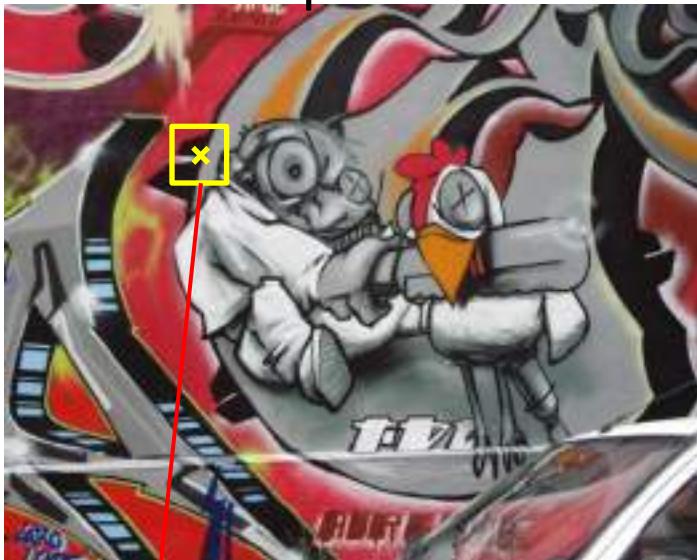
Automatic Scale Selection

- Function responses for increasing scale (scale signature)

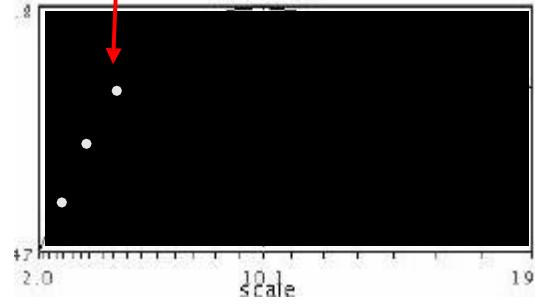
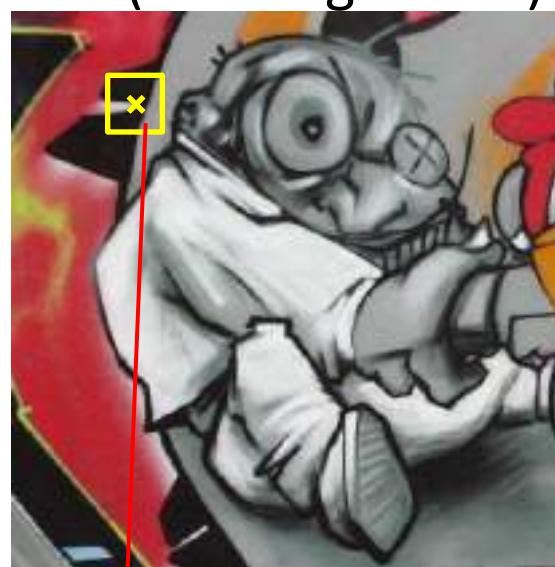


Automatic Scale Selection

- Function responses for increasing scale (scale signature)



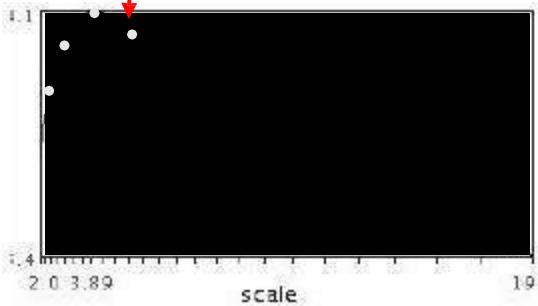
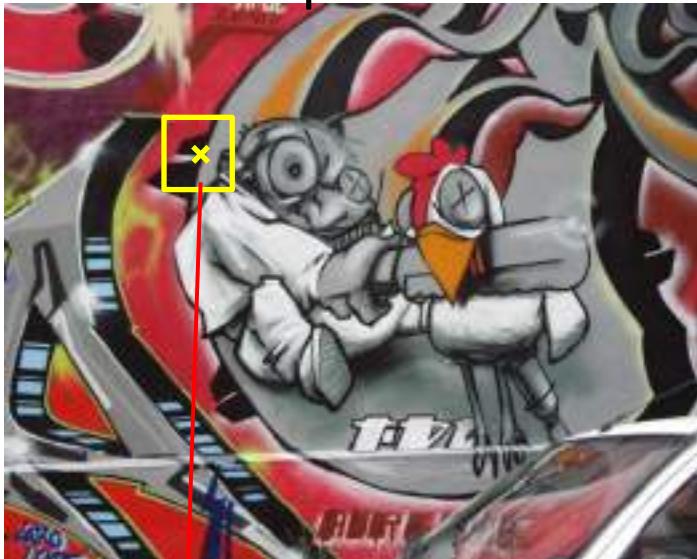
$$f(I_{i_1 \dots i_m}(x, \sigma))$$



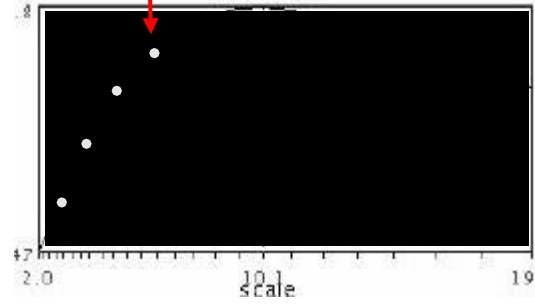
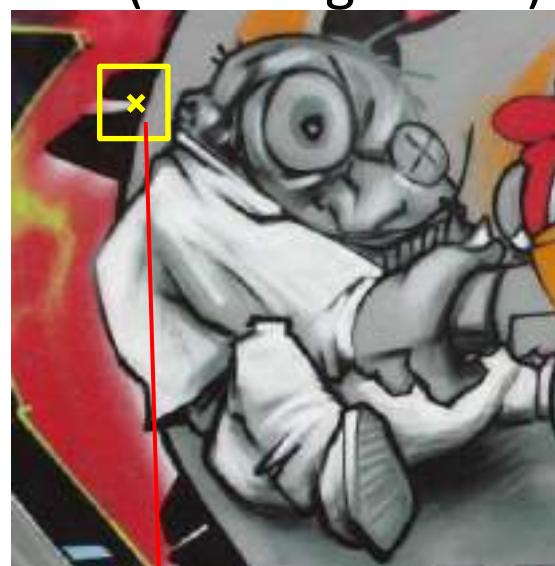
$$f(I_{i_1 \dots i_m}(x', \sigma))$$

Automatic Scale Selection

- Function responses for increasing scale (scale signature)



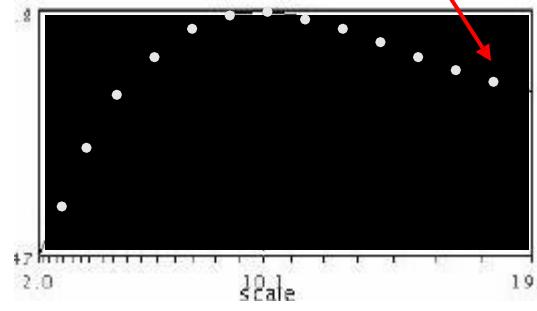
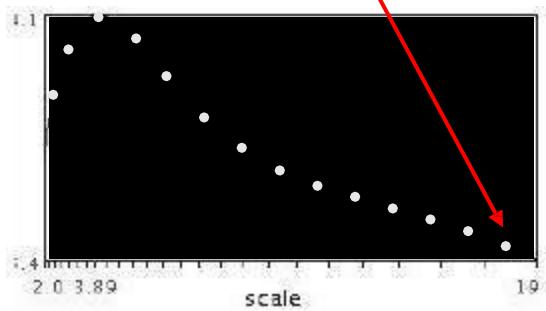
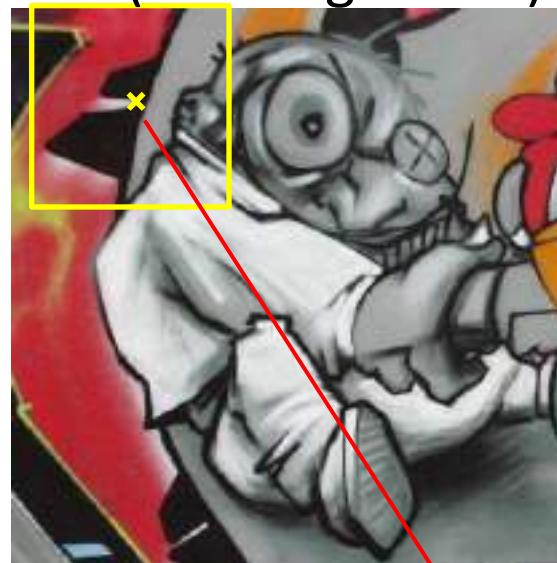
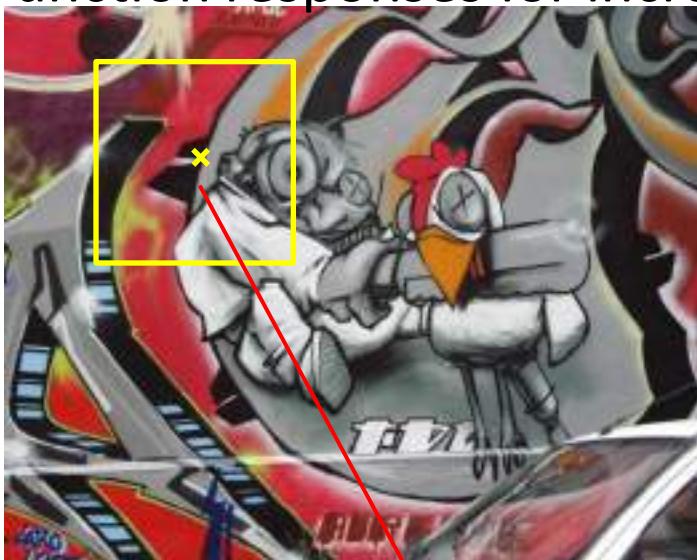
$$f(I_{i_1 \dots i_m}(x, \sigma))$$



$$f(I_{i_1 \dots i_m}(x', \sigma))$$

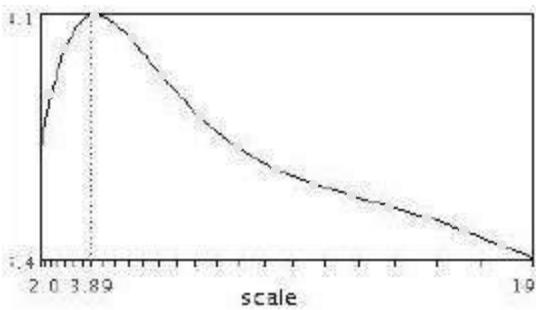
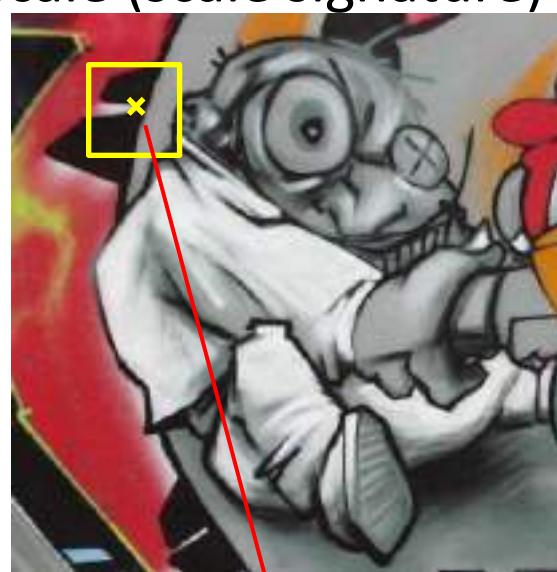
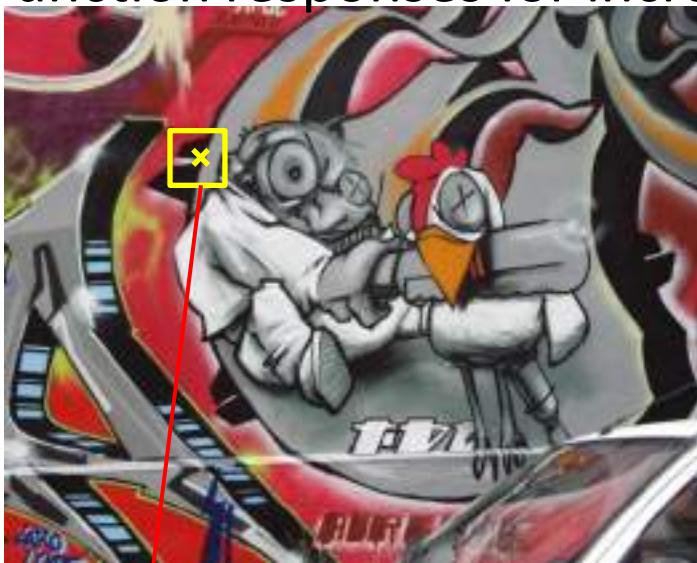
Automatic Scale Selection

- Function responses for increasing scale (scale signature)

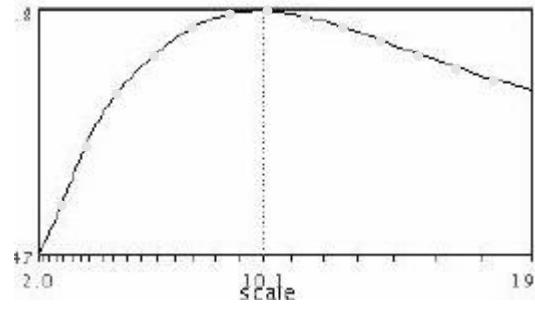


Automatic Scale Selection

- Function responses for increasing scale (scale signature)



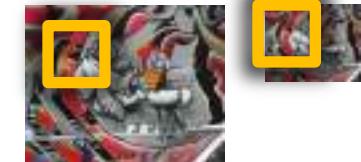
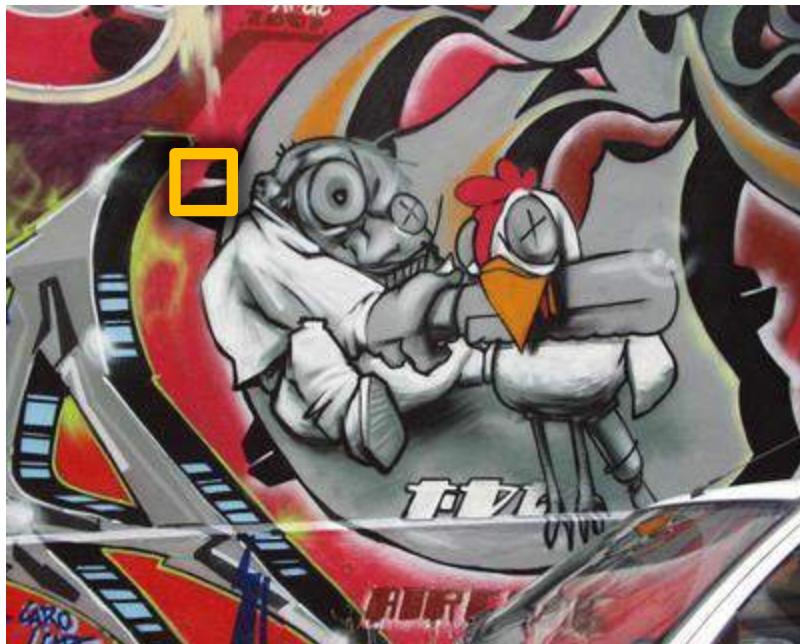
$$f(I_{i_1 \dots i_m}(x, \sigma))$$



$$f(I_{i_1 \dots i_m}(x', \sigma'))$$

Implementation

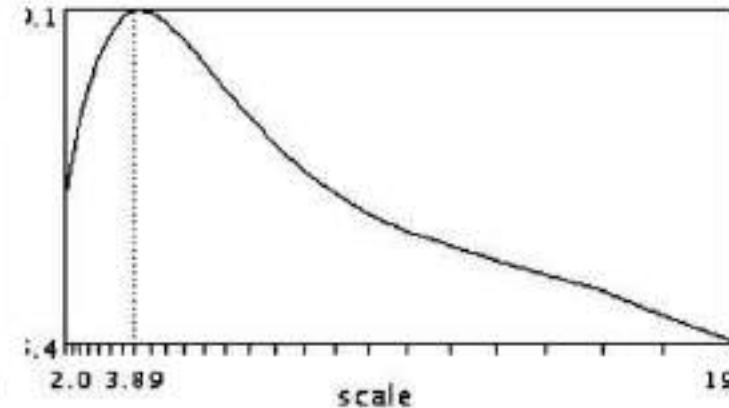
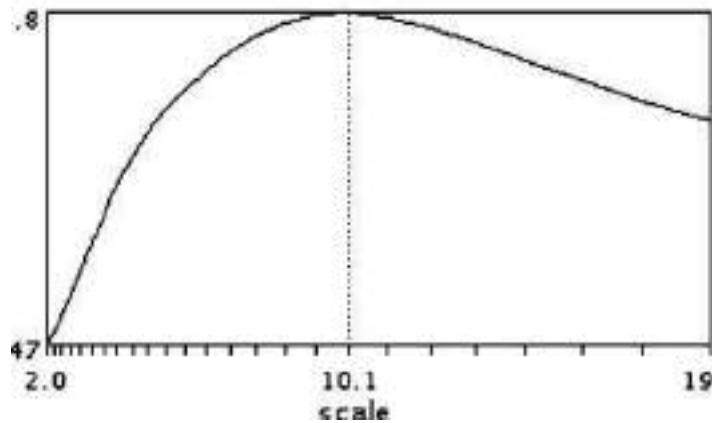
- Instead of computing f for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid



(sometimes need to create in-between levels, e.g. a $\frac{3}{4}$ -size image)

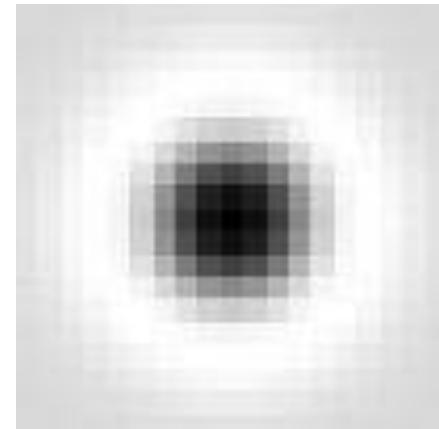
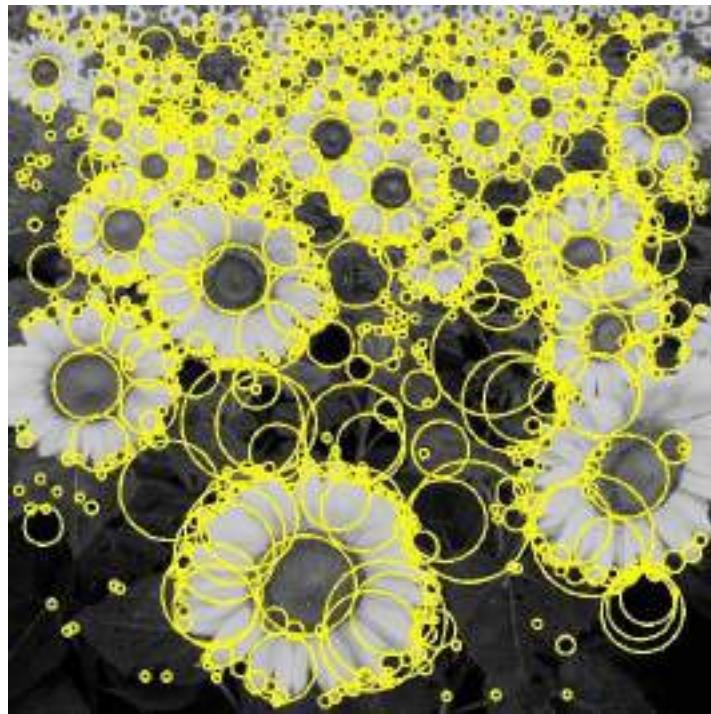
Keypoint detection with scale selection

- We want to extract keypoints with characteristic scale that is *covariant* with the image transformation



Basic idea

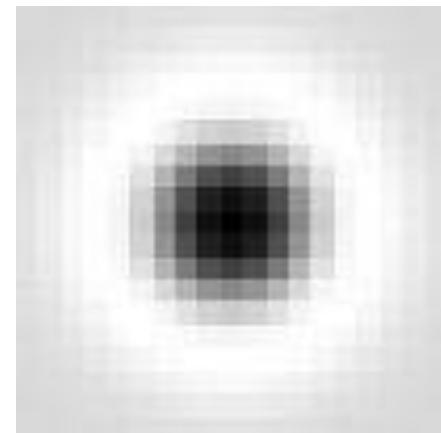
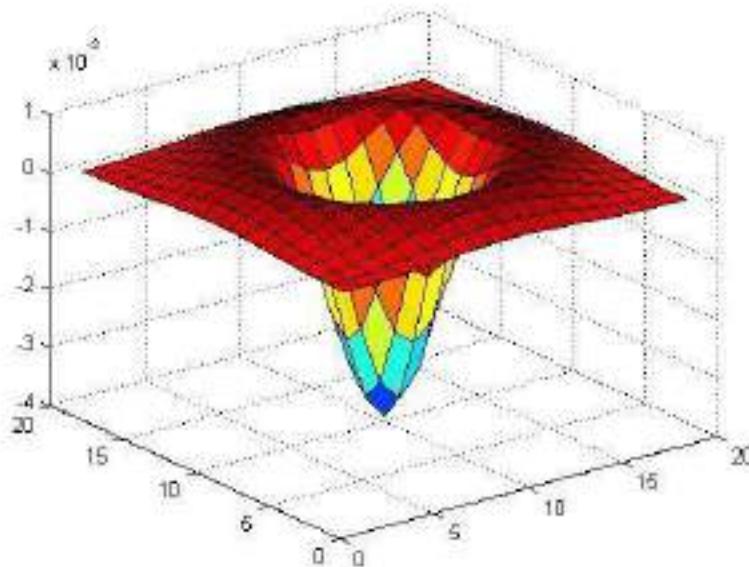
- Convolve the image with a “blob filter” at multiple scales and look for extrema of filter response in the resulting *scale space*



T. Lindeberg. [Feature detection with automatic scale selection.](#)
IJCV 30(2), pp 77-116, 1998.

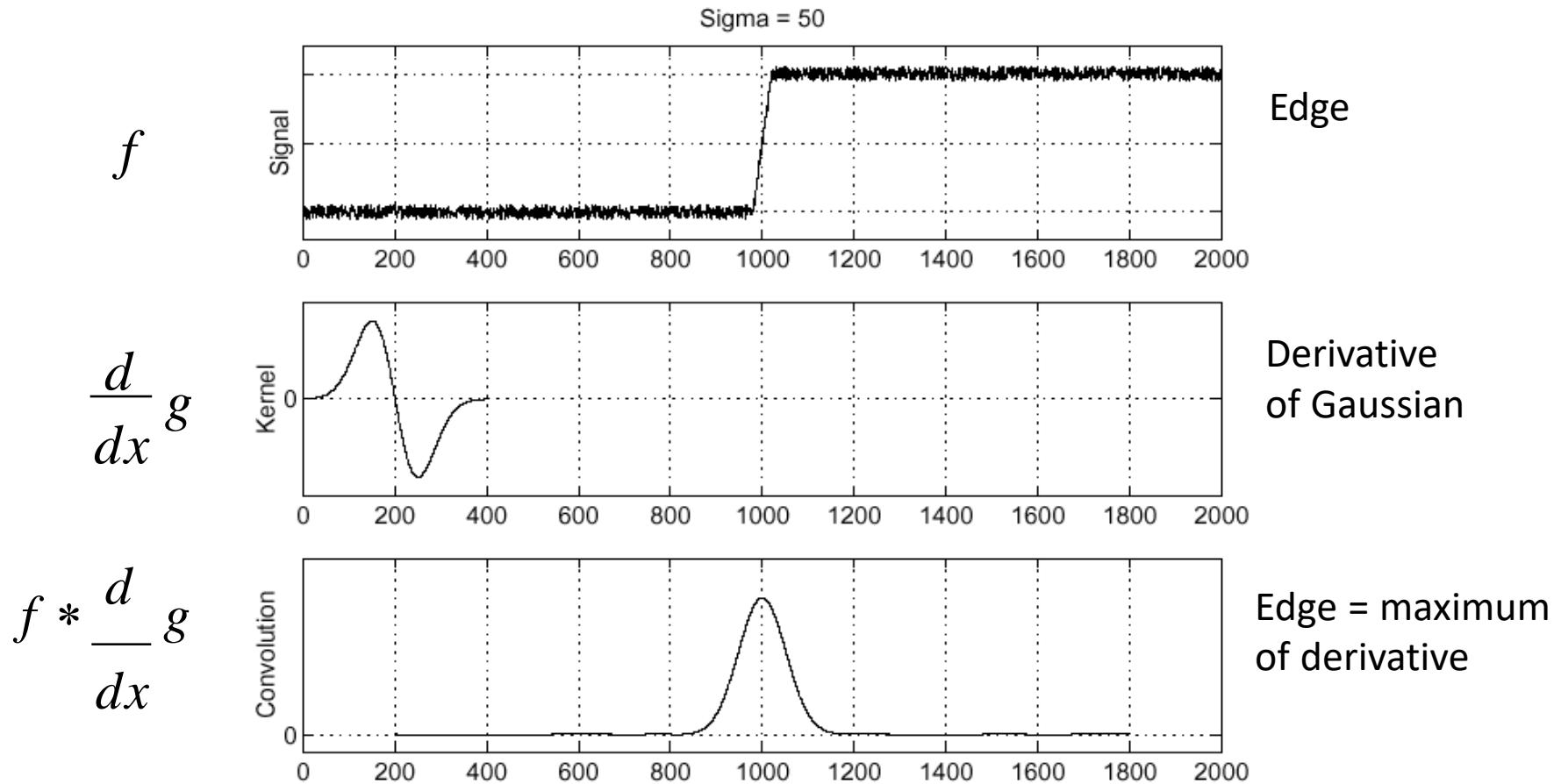
Blob filter

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D

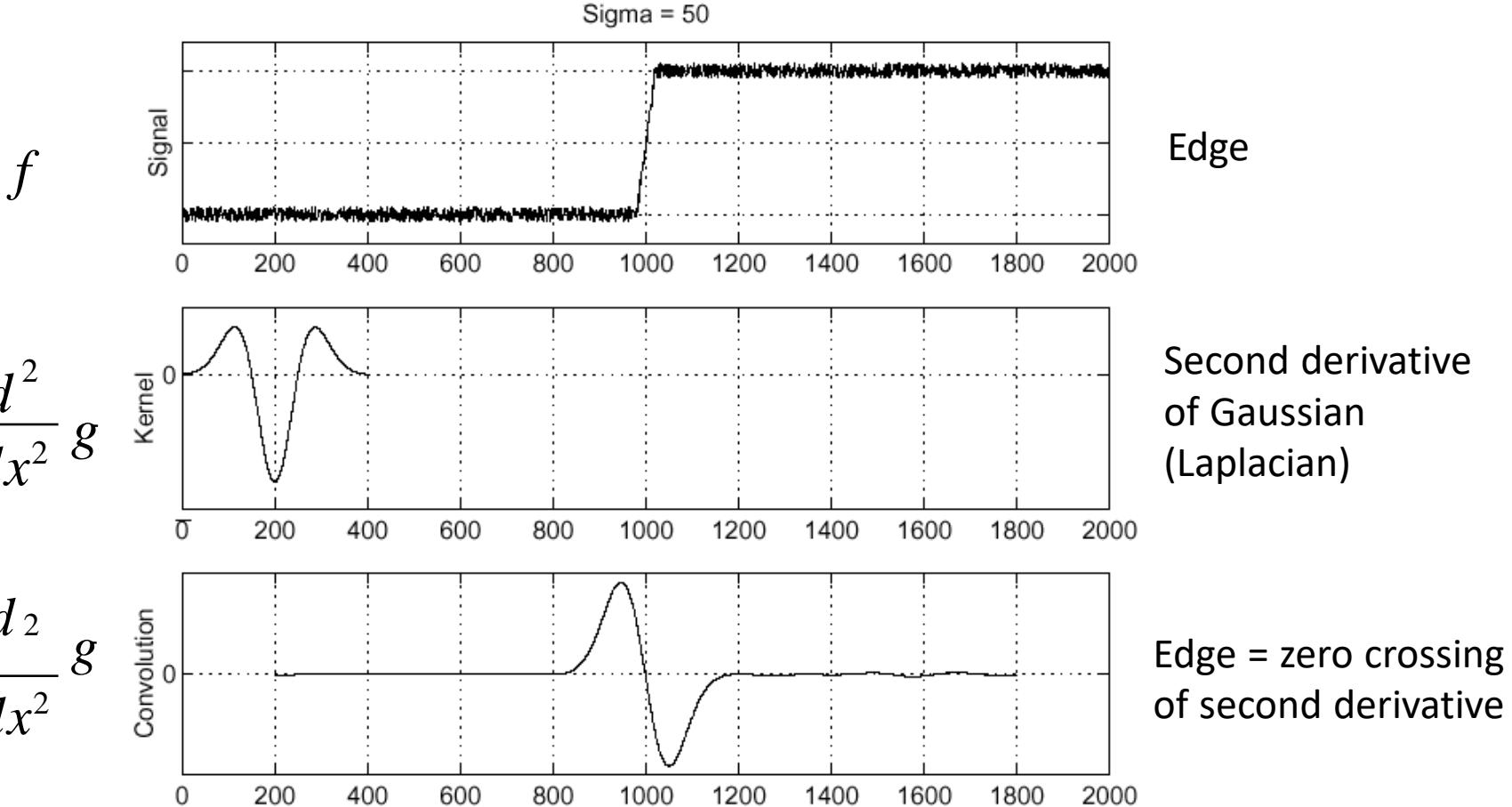


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Recall: Edge detection

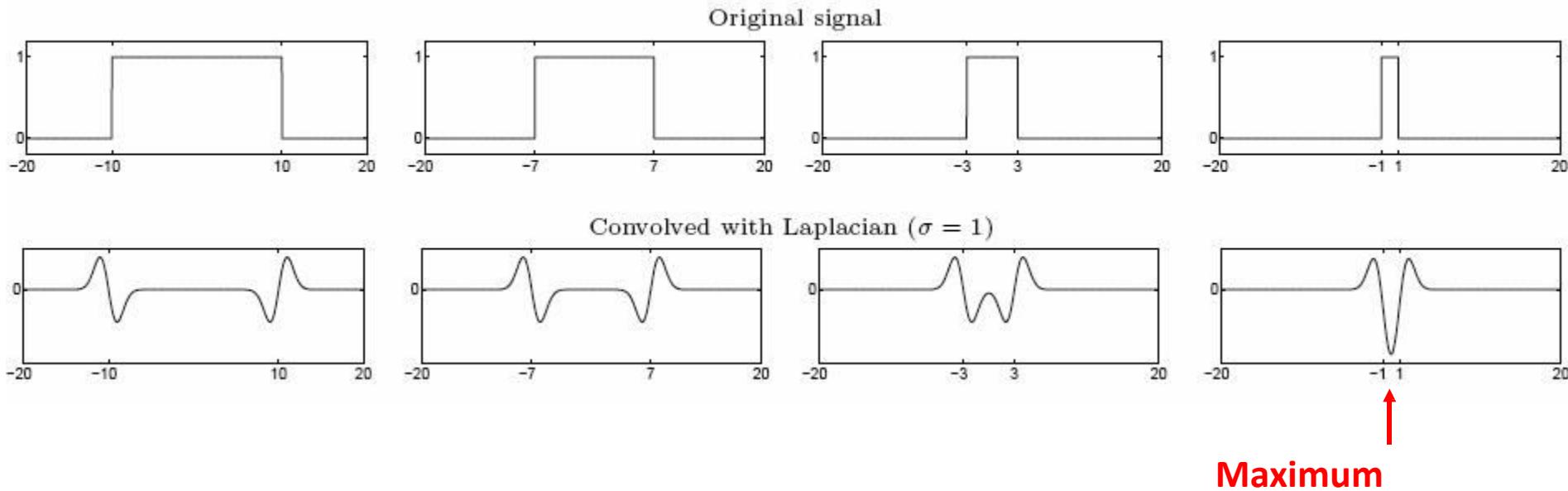


Edge detection, Take 2



From edges to blobs

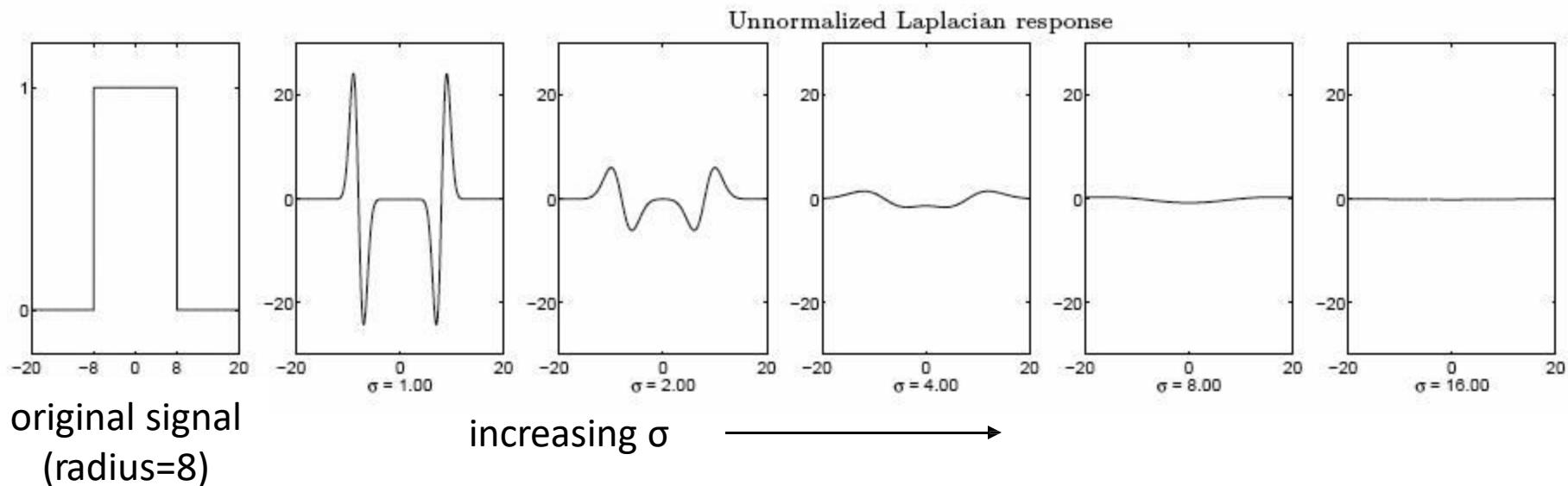
- Edge = ripple
- Blob = superposition of two ripples



Spatial selection: the magnitude of the Laplacian response will achieve a maximum at the center of the blob, provided the scale of the Laplacian is “matched” to the scale of the blob

Scale selection

- We want to find the characteristic scale of the blob by convolving it with Laplacians at several scales and looking for the maximum response
- However, Laplacian response decays as scale increases:

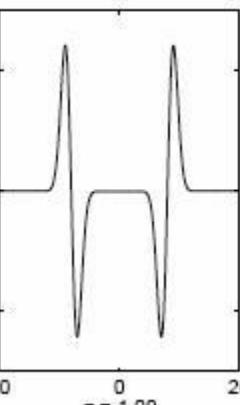
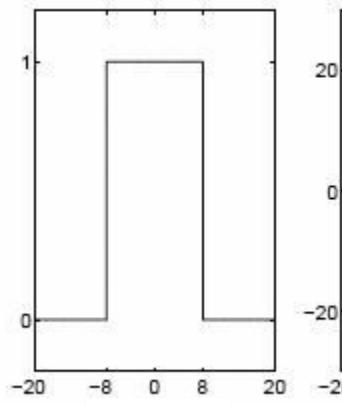


Scale normalization

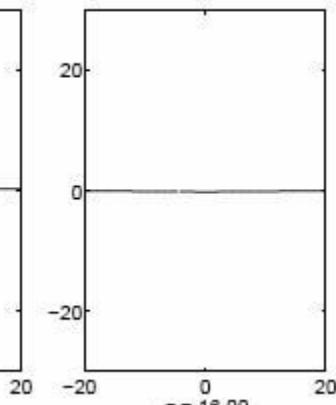
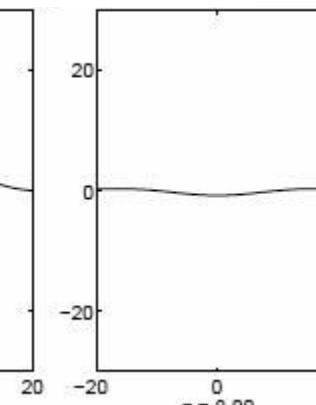
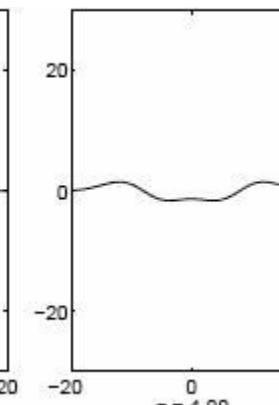
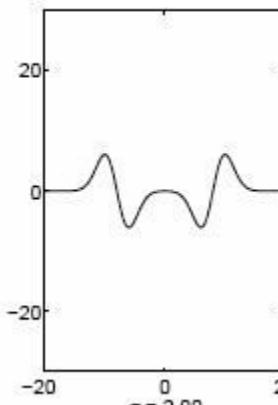
- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases
- To keep response the same (scale-invariant), must multiply Gaussian derivative by σ
- Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2

Effect of scale normalization

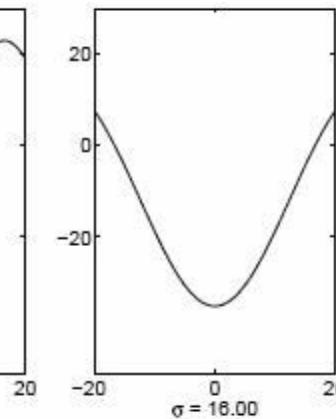
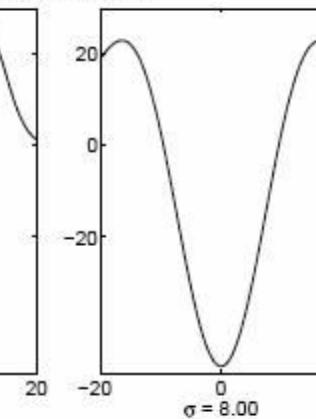
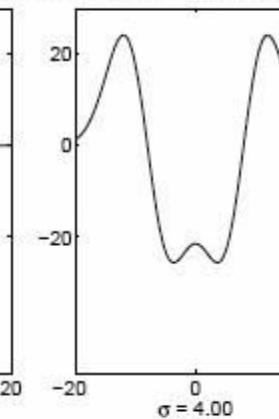
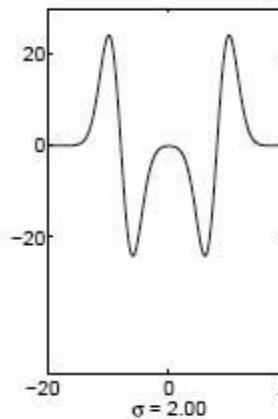
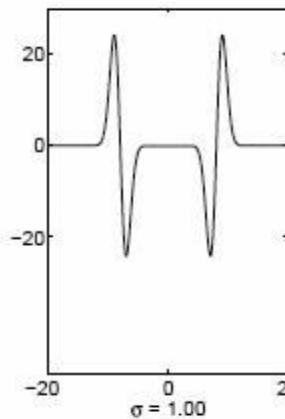
Original signal



Unnormalized Laplacian response



Scale-normalized Laplacian response

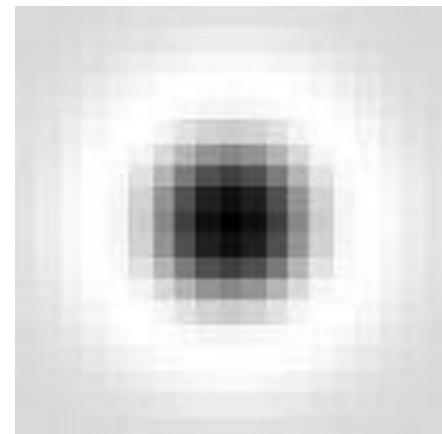
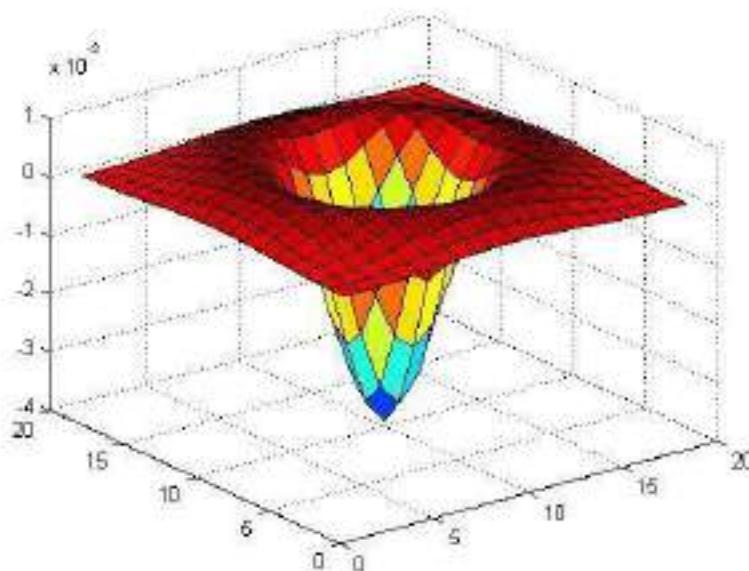


maximum

Blob detection in 2D

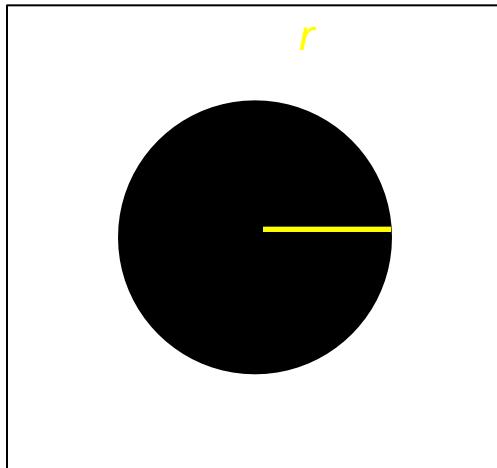
- *Scale-normalized Laplacian of Gaussian:*

$$\nabla_{\text{norm}}^2 g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

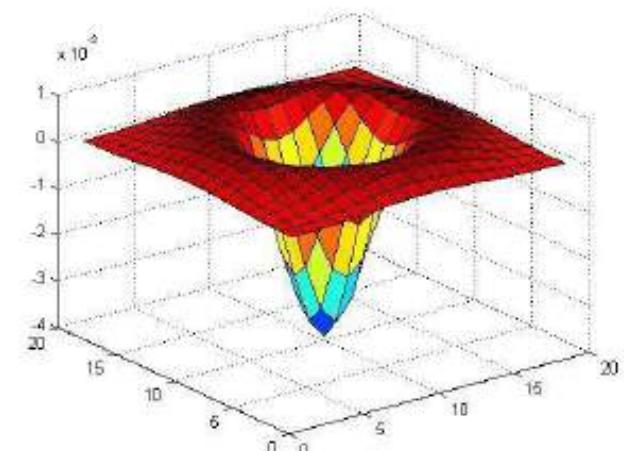
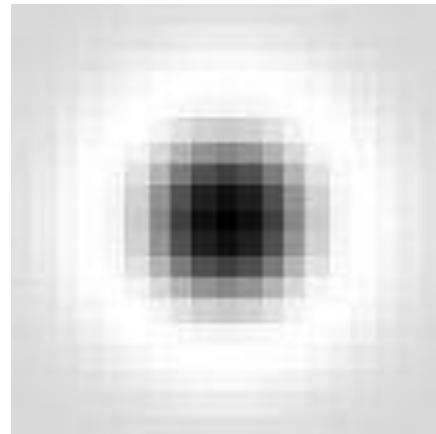


Blob detection in 2D

- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r ?



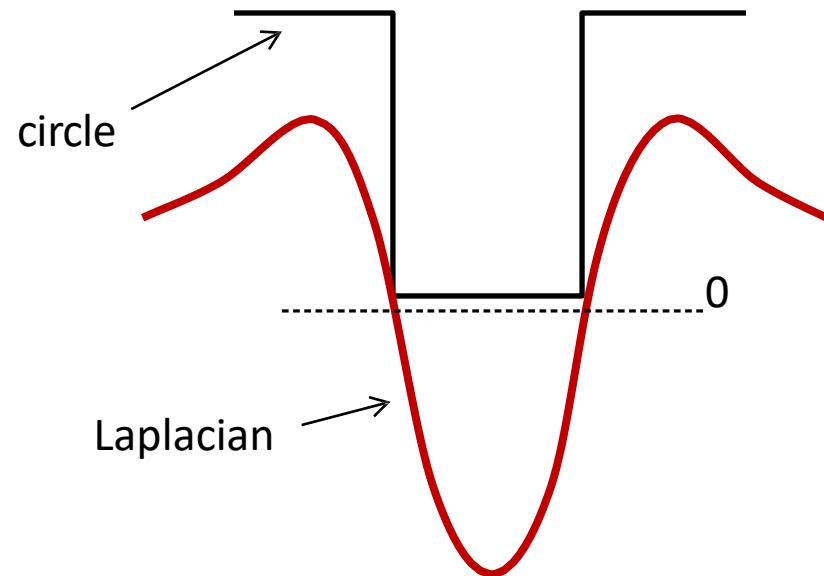
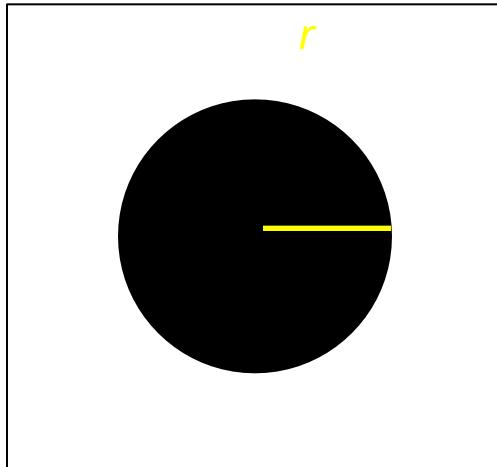
image



Laplacian

Blob detection in 2D

- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r ?
- To get maximum response, the zeros of the Laplacian have to be aligned with the circle

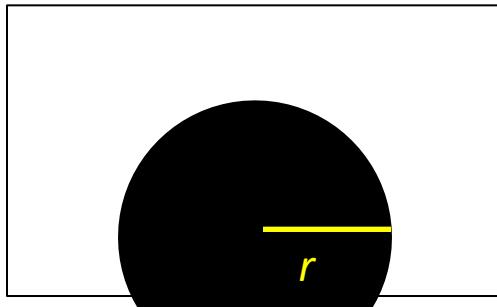


image

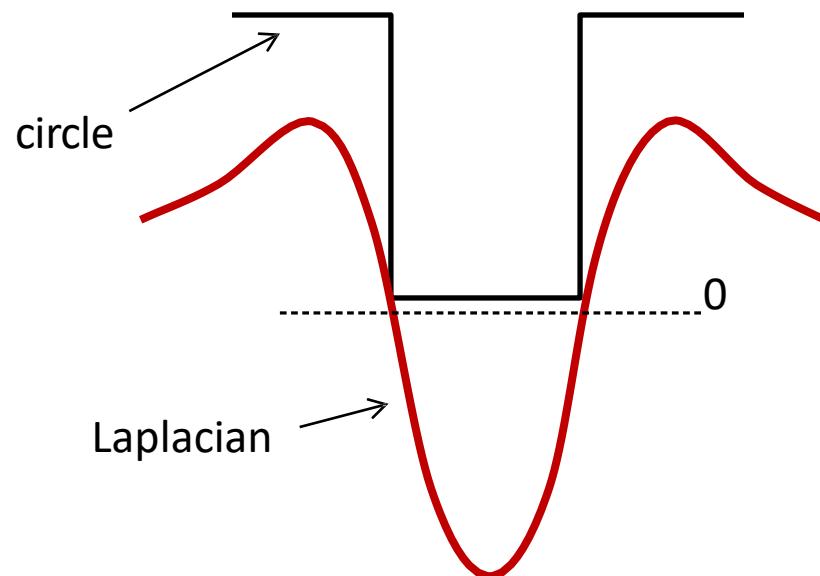
Blob detection in 2D

- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r ?
- To get maximum response, the zeros of the Laplacian have to be aligned with the circle
- The Laplacian is given by (up to scale):

$$(x^2 + y^2 - 2\sigma^2) e^{-(x^2+y^2)/2\sigma^2}$$



image

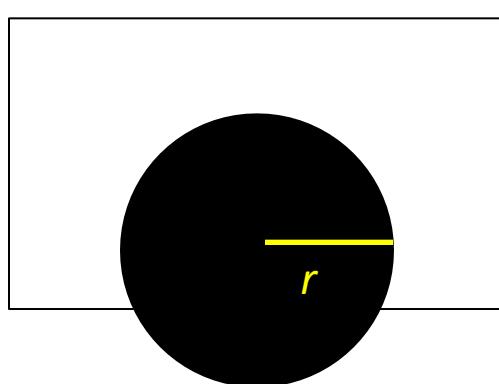


Blob detection in 2D

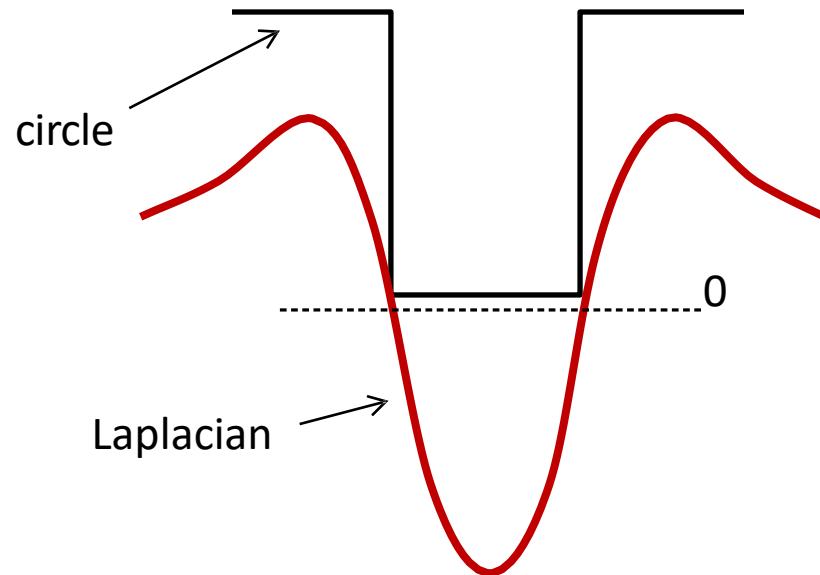
- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r ?
- To get maximum response, the zeros of the Laplacian have to be aligned with the circle
- The Laplacian is given by (up to scale):

$$(x^2 + y^2 - 2\sigma^2) e^{-(x^2+y^2)/2\sigma^2}$$

- Therefore, the maximum response occurs at $\sigma = r / \sqrt{2}$



image



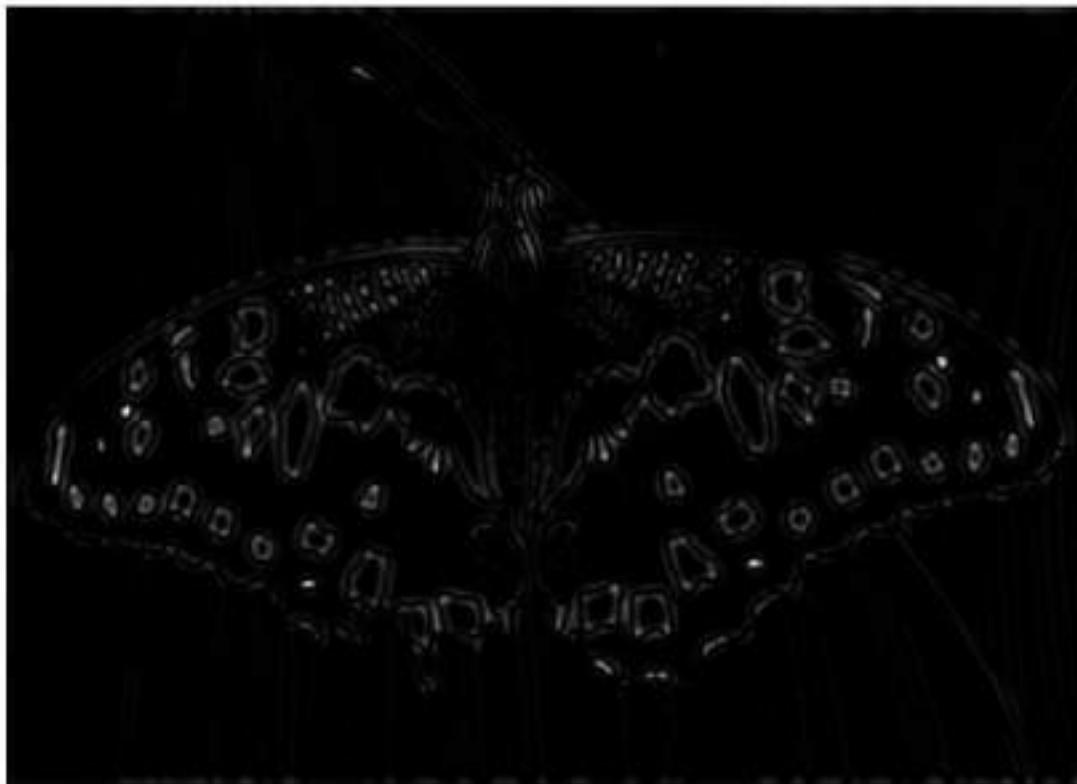
Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales

Scale-space blob detector: Example

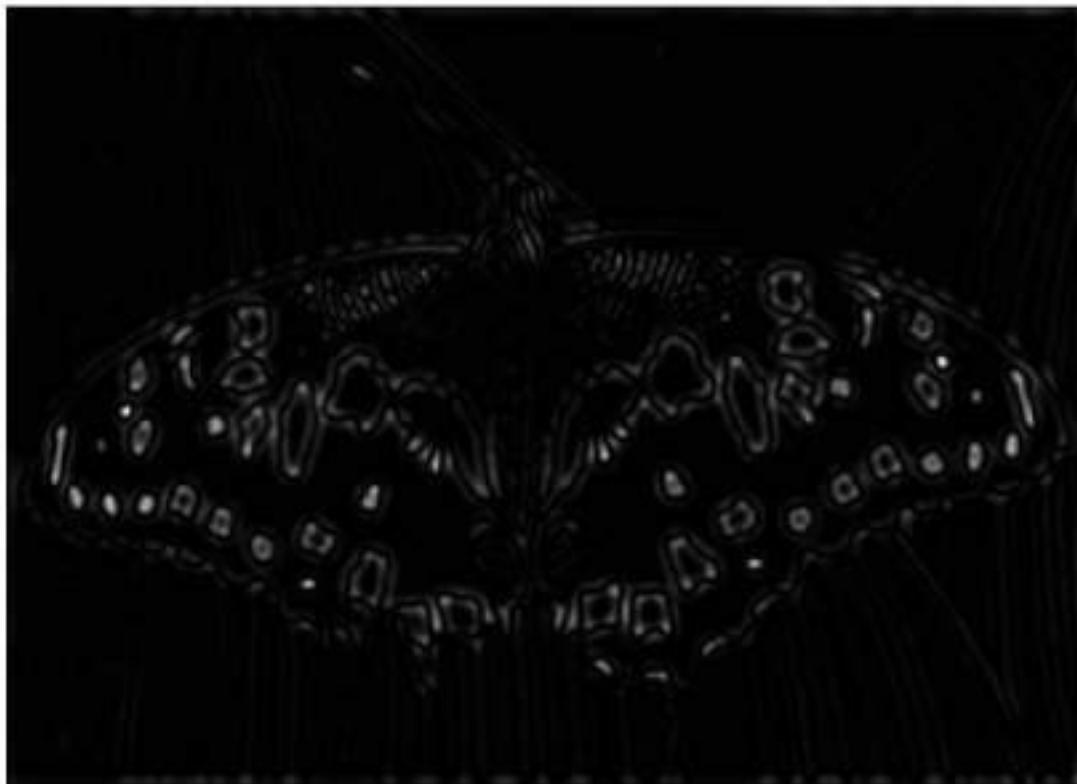


Scale-space blob detector: Example



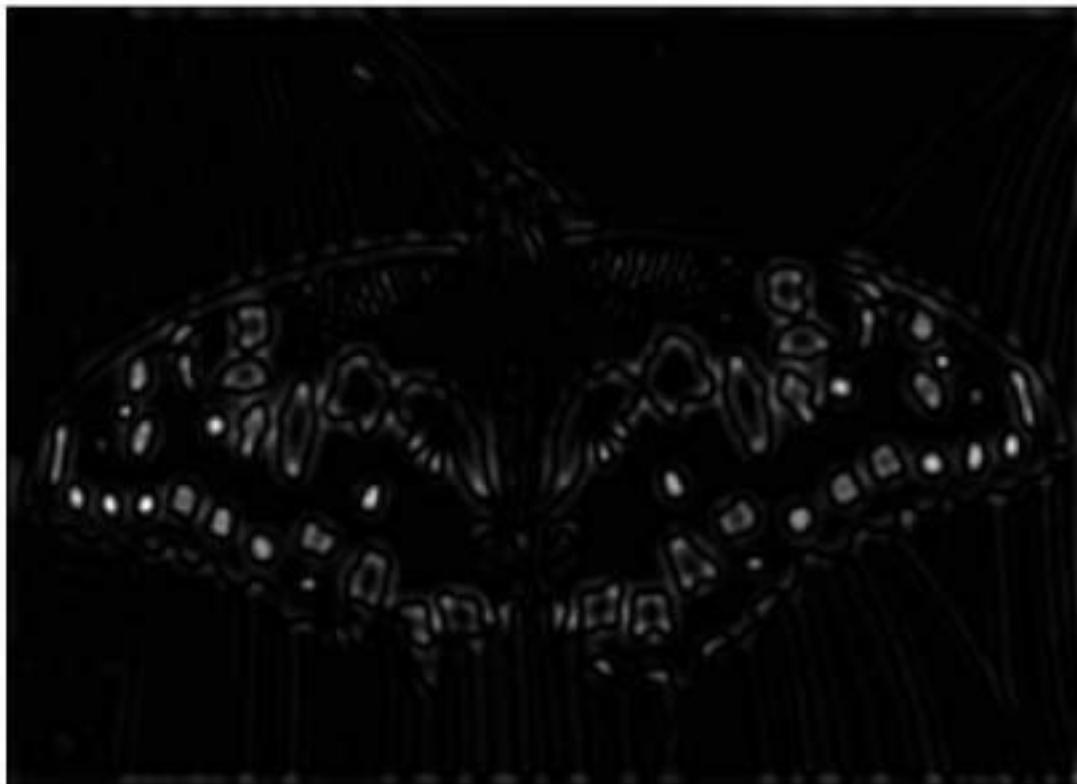
$\sigma = 2$

Scale-space blob detector: Example



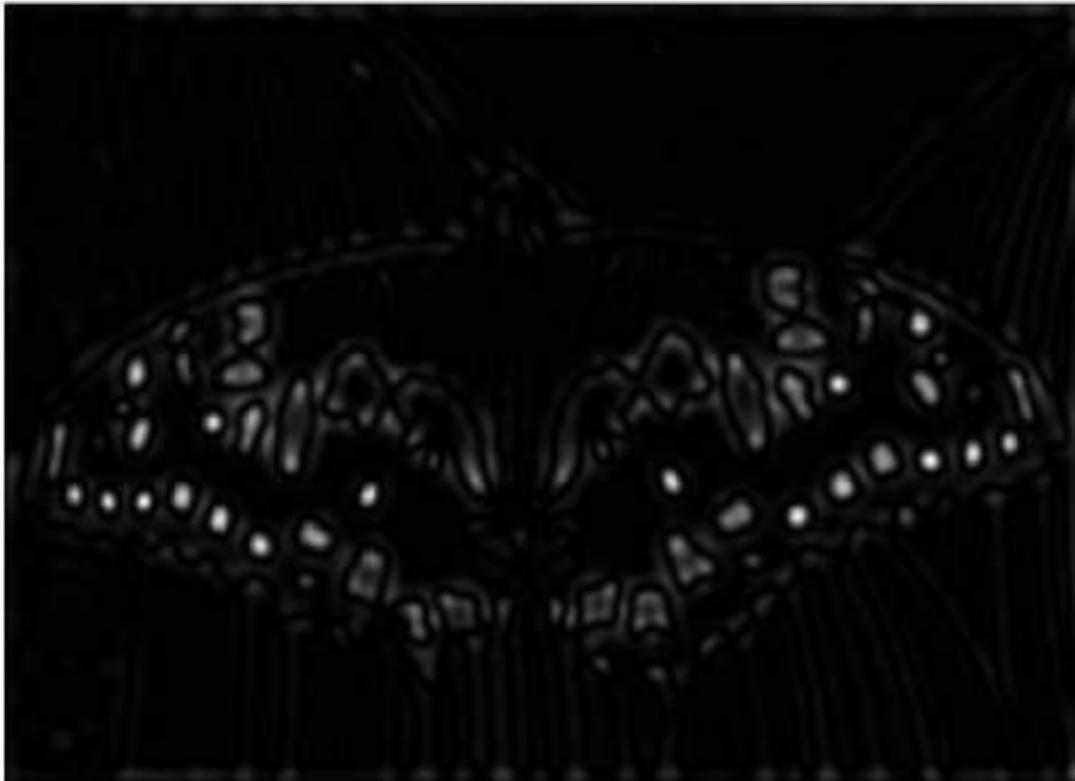
sigma = 2.5018

Scale-space blob detector: Example



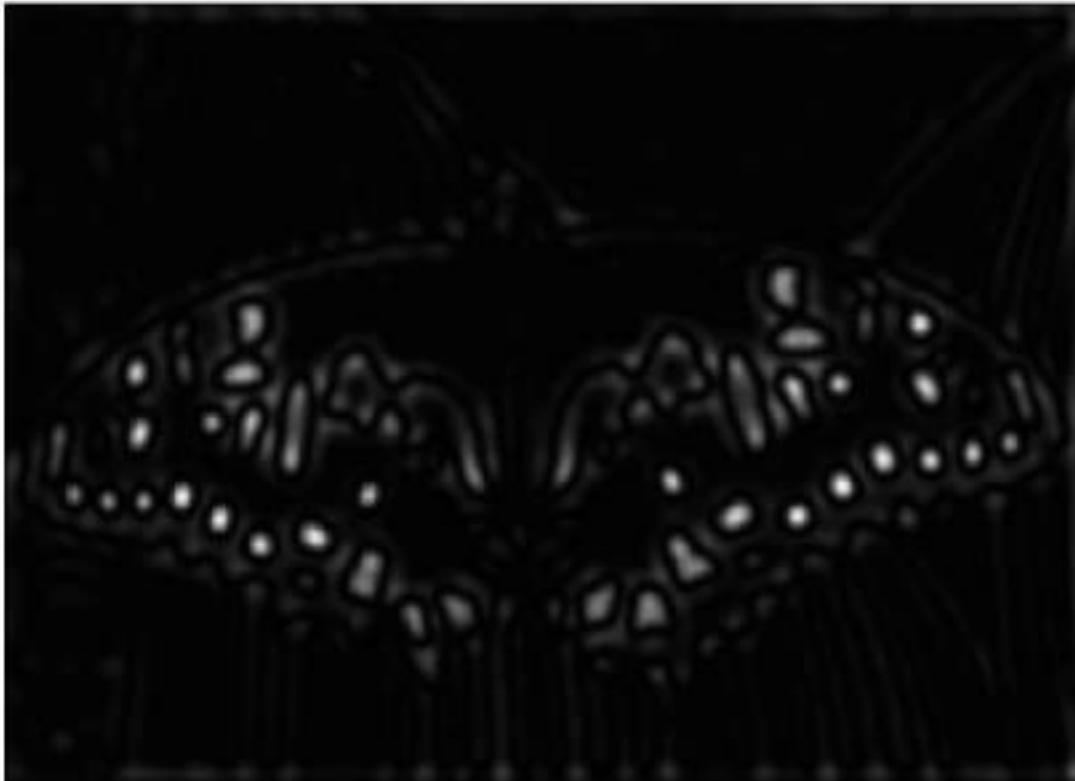
$\sigma = 3.1296$

Scale-space blob detector: Example



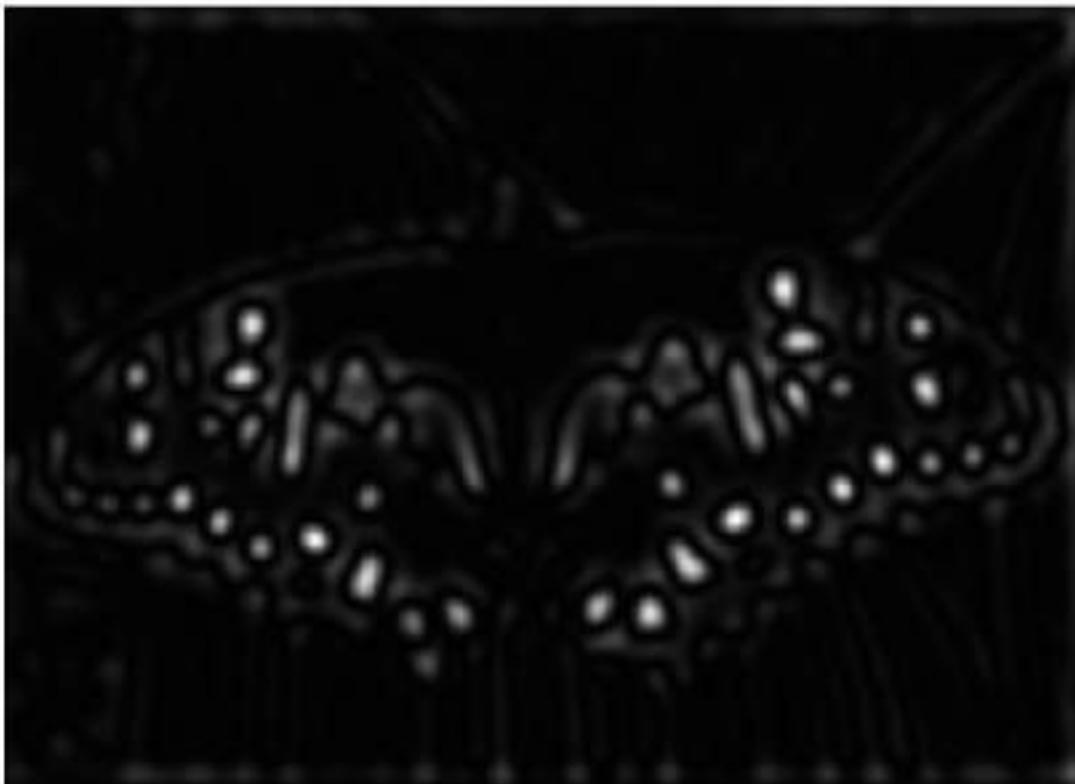
$\sigma = 3.9149$

Scale-space blob detector: Example



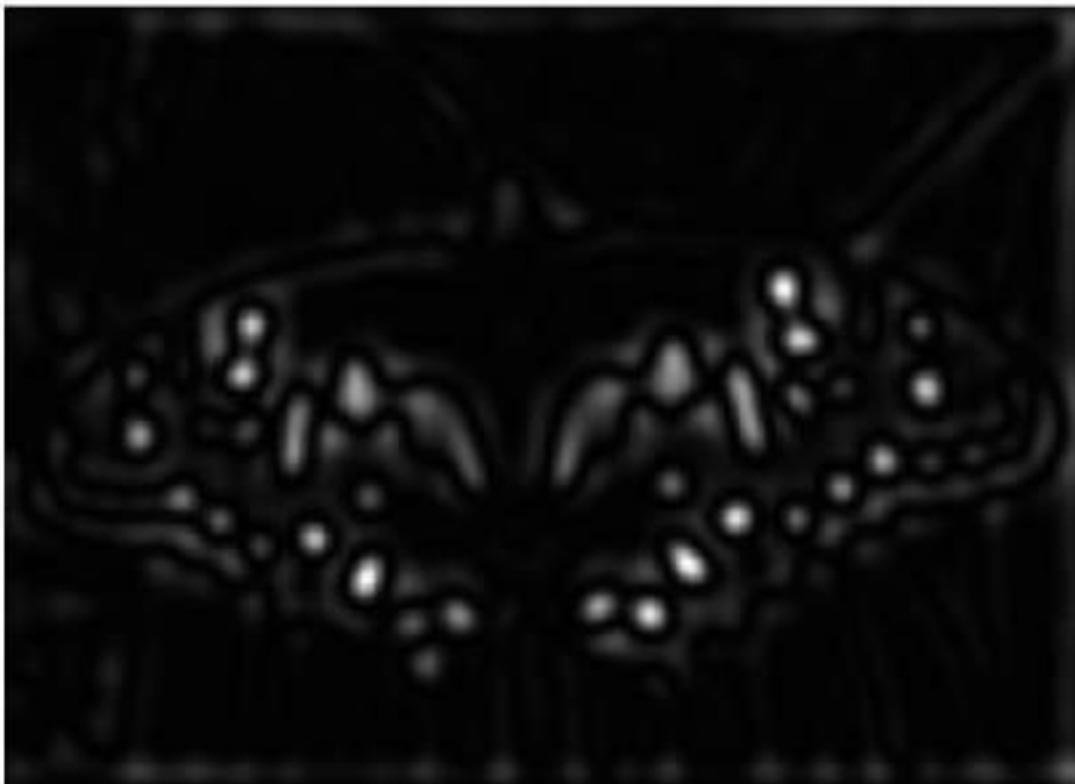
$\sigma = 4.8972$

Scale-space blob detector: Example



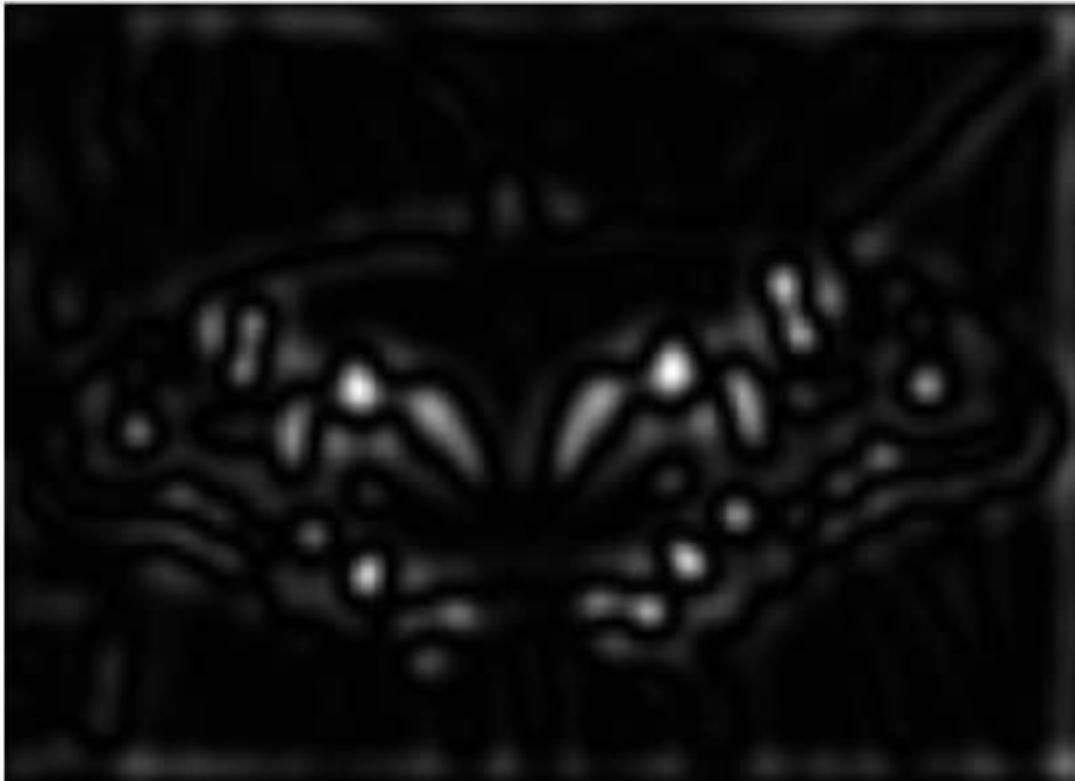
$\sigma = 6.126$

Scale-space blob detector: Example



$\sigma = 7.6631$

Scale-space blob detector: Example



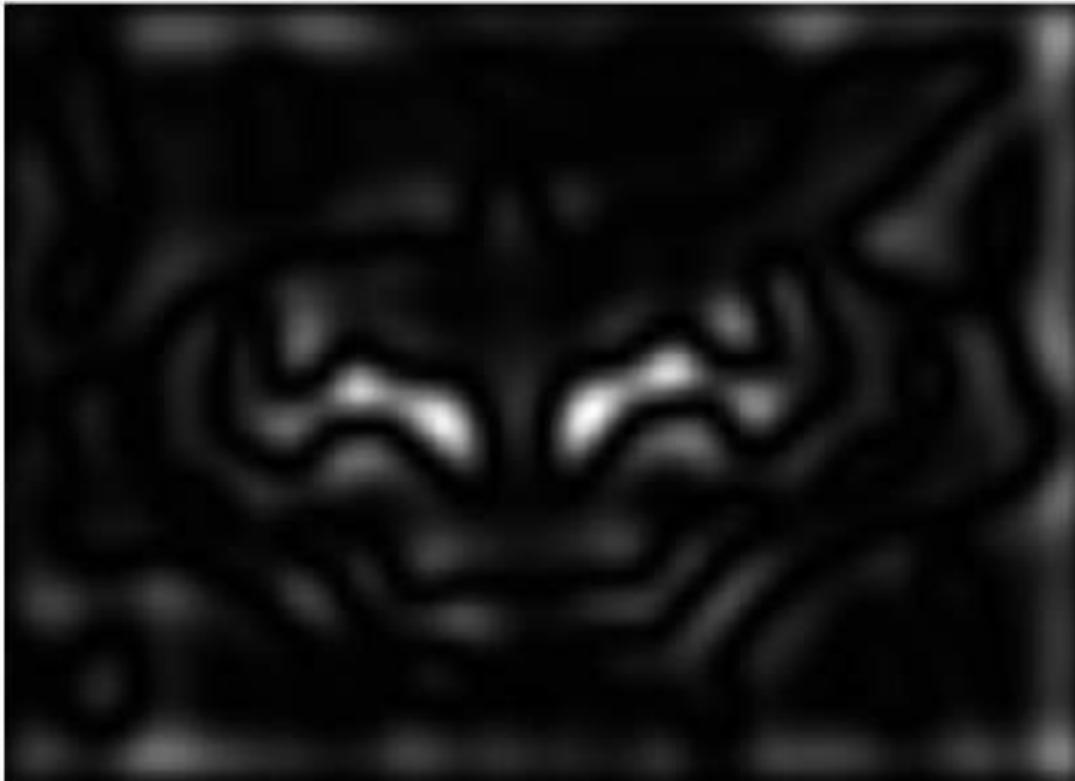
sigma = 9.5859

Scale-space blob detector: Example



$\sigma = 11.9912$

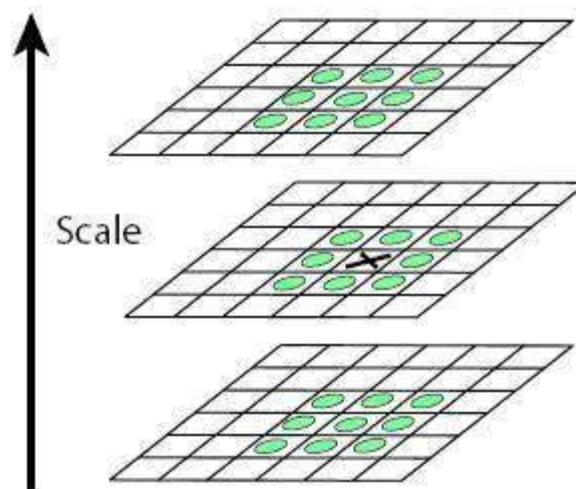
Scale-space blob detector Example



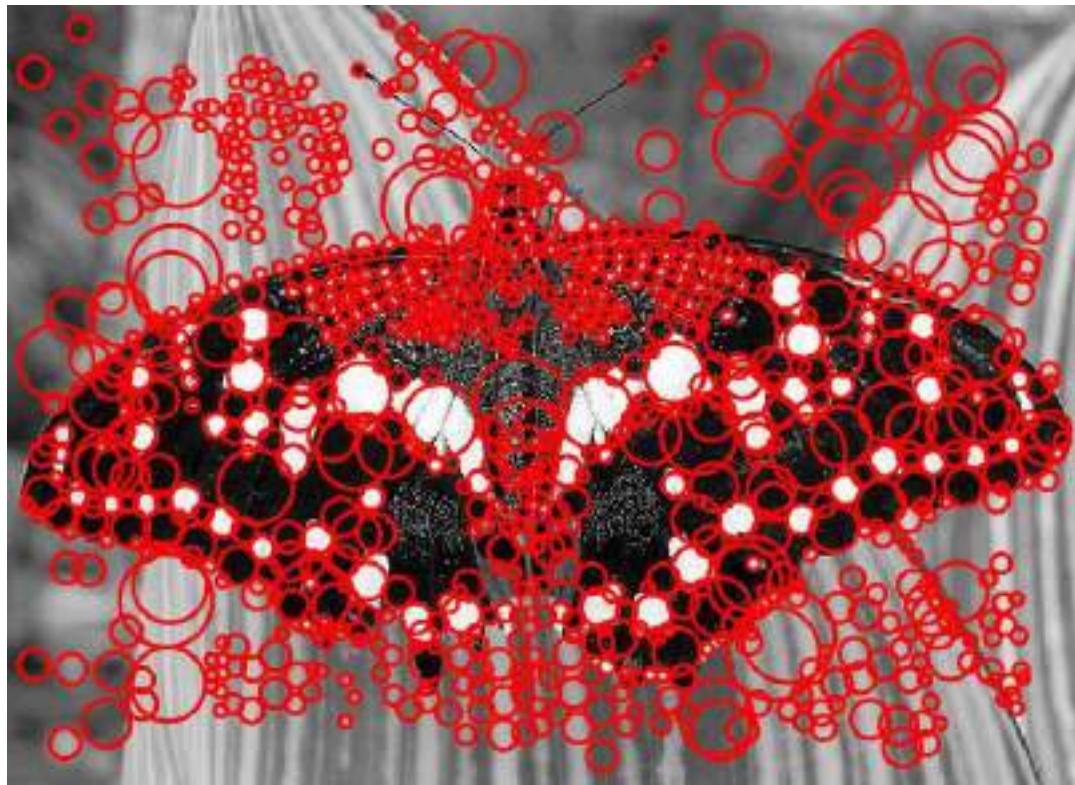
sigma = 15

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space



Scale-space blob detector: Example



Efficient implementation (SIFT)

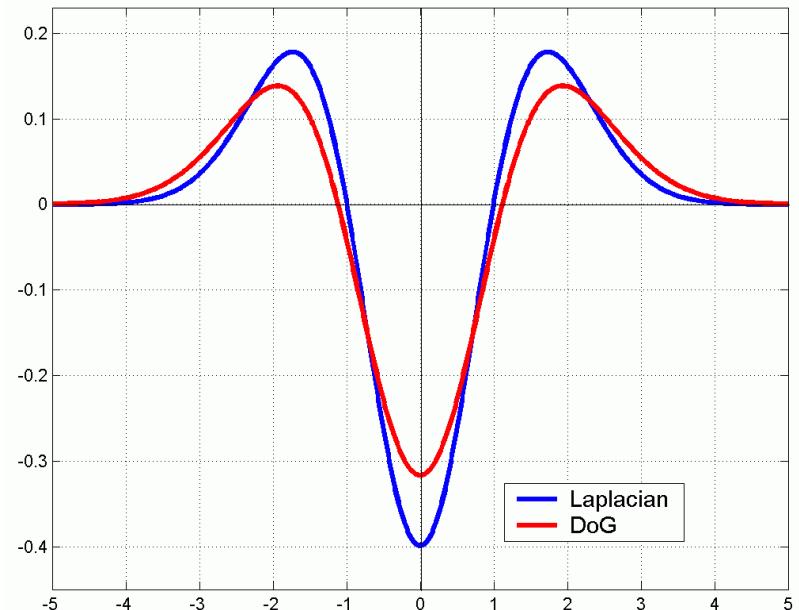
- Approximating the Laplacian with a difference of Gaussians by SIFT detector:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

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

(Difference of Gaussians)



Difference-of-Gaussian (DoG)



-

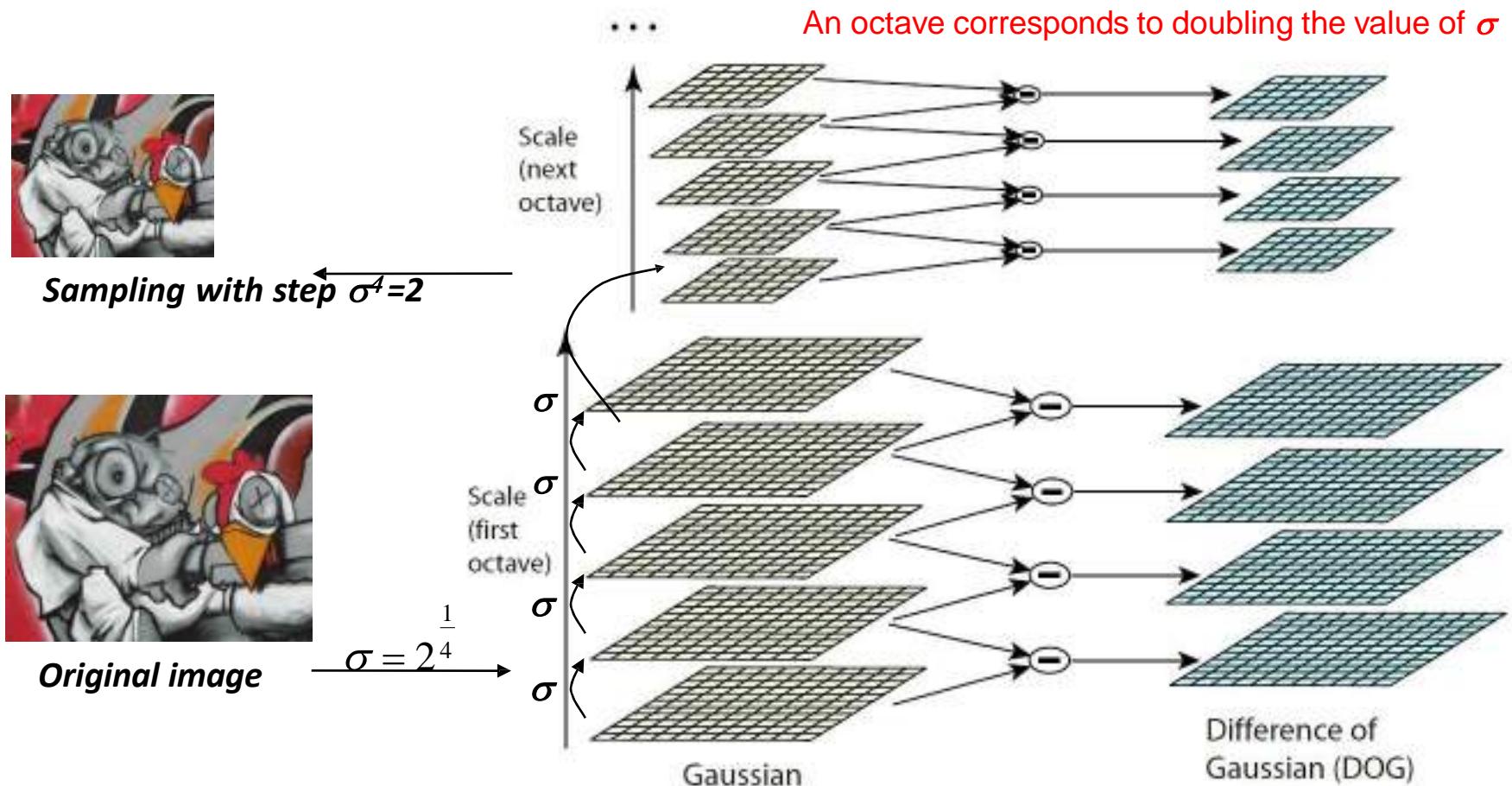


=

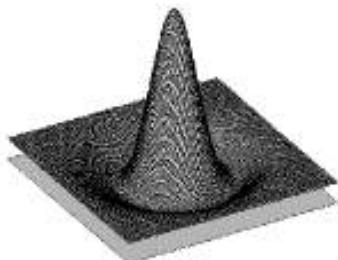


DoG – Efficient Computation

- Computation in Gaussian scale pyramid

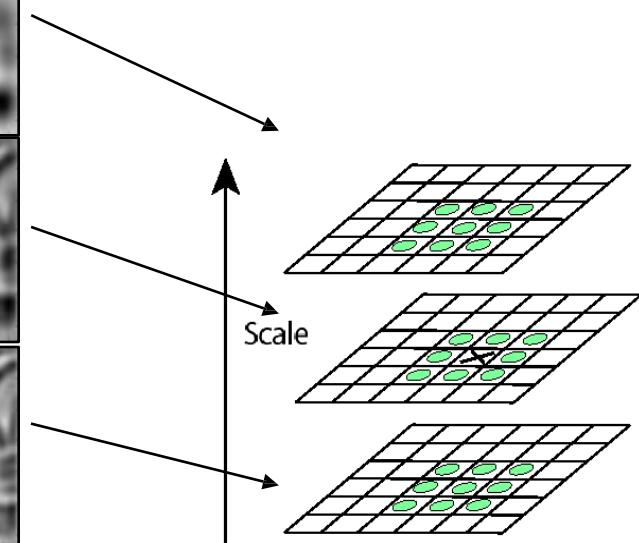
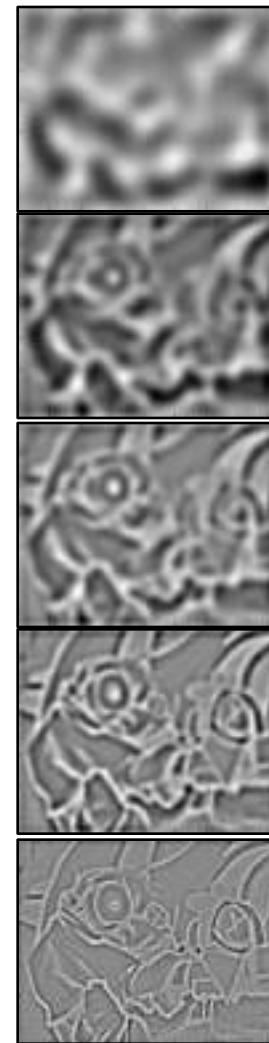


Find local maxima in position-scale space of Difference-of-Gaussian



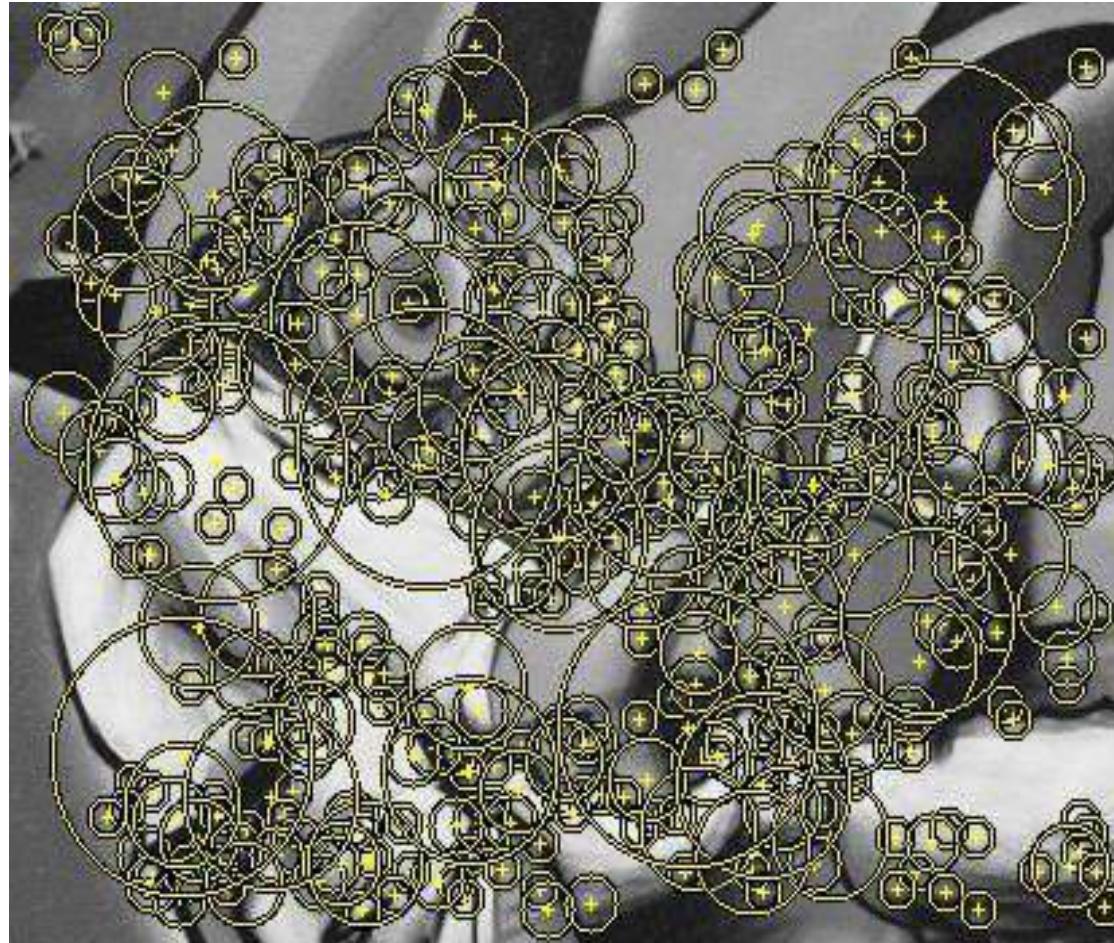
$$L_{xx}(\sigma) + L_{yy}(\sigma) \rightarrow \sigma^3$$

Arrows point from the equation to five vertically stacked images showing increasing levels of scale-space filtering, labeled σ , σ^2 , σ^3 , σ^4 , and σ^5 .



⇒ List of
 (x, y, s)

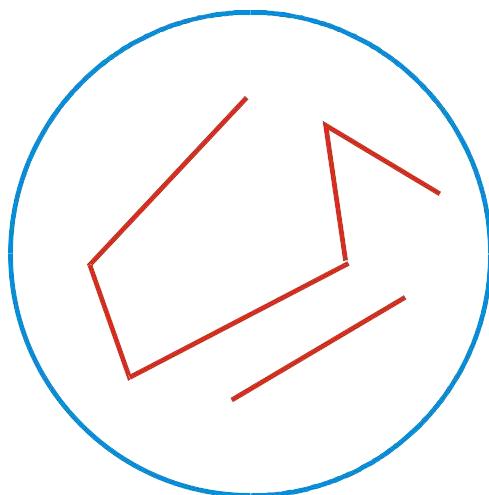
Results: Difference-of-Gaussian



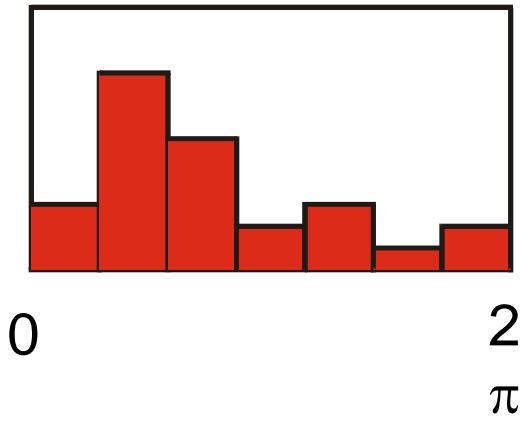
Orientation Normalization

[Lowe, SIFT, 1999]

- Compute orientation histogram



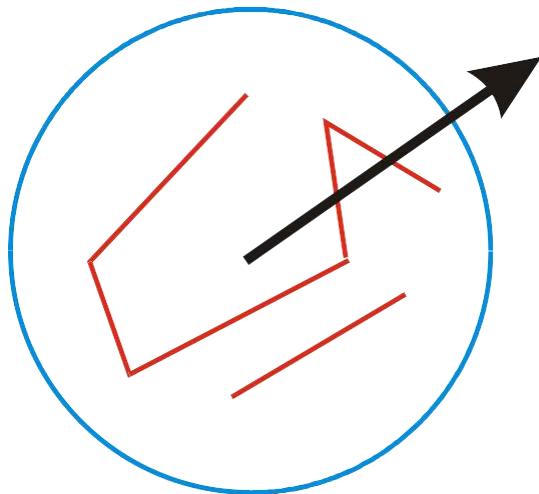
T. Tuytelaars, B. Leibe



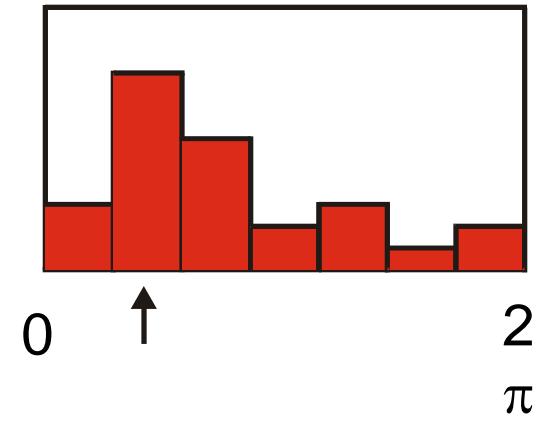
Orientation Normalization

[Lowe, SIFT, 1999]

- Compute orientation histogram
- Select dominant orientation



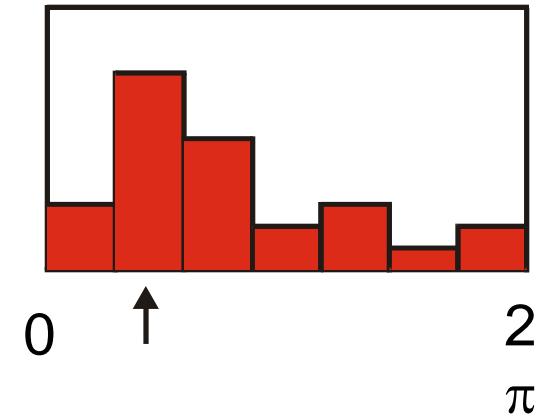
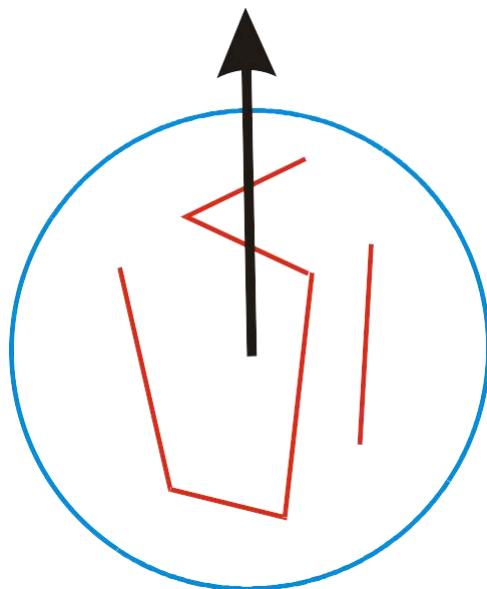
T. Tuytelaars, B. Leibe



Orientation Normalization

[Lowe, SIFT, 1999]

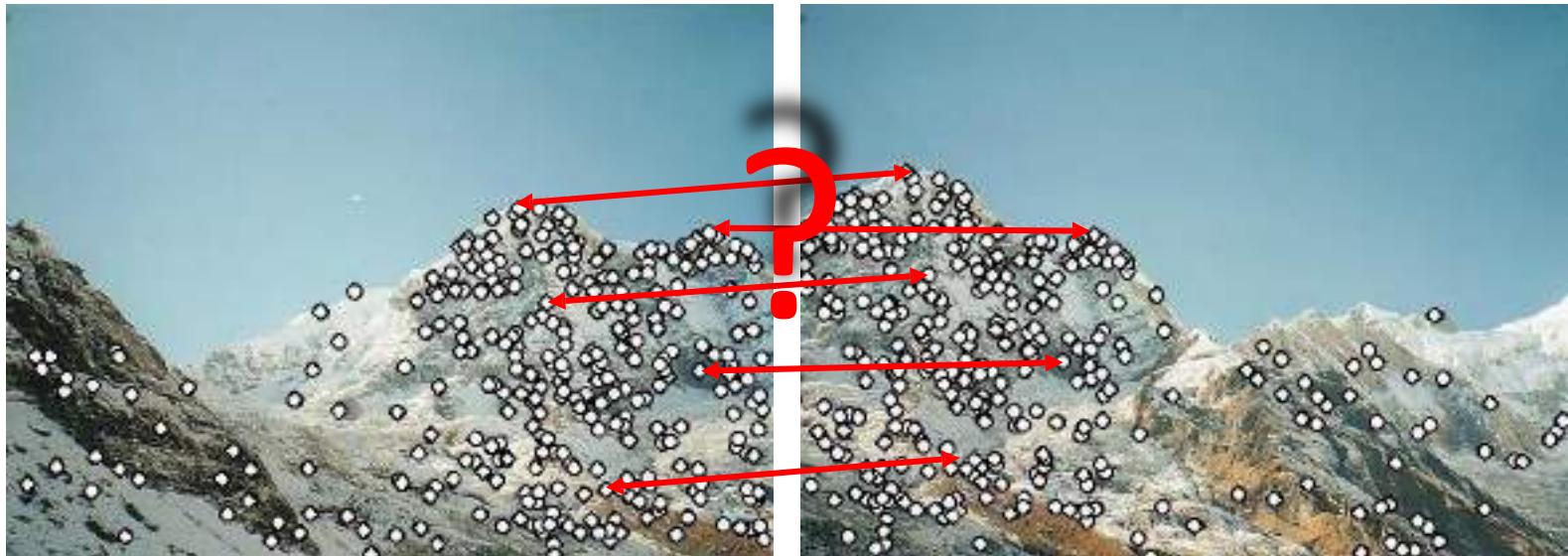
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation



Feature descriptors

We know how to detect good points

Next question: **How to match them?**

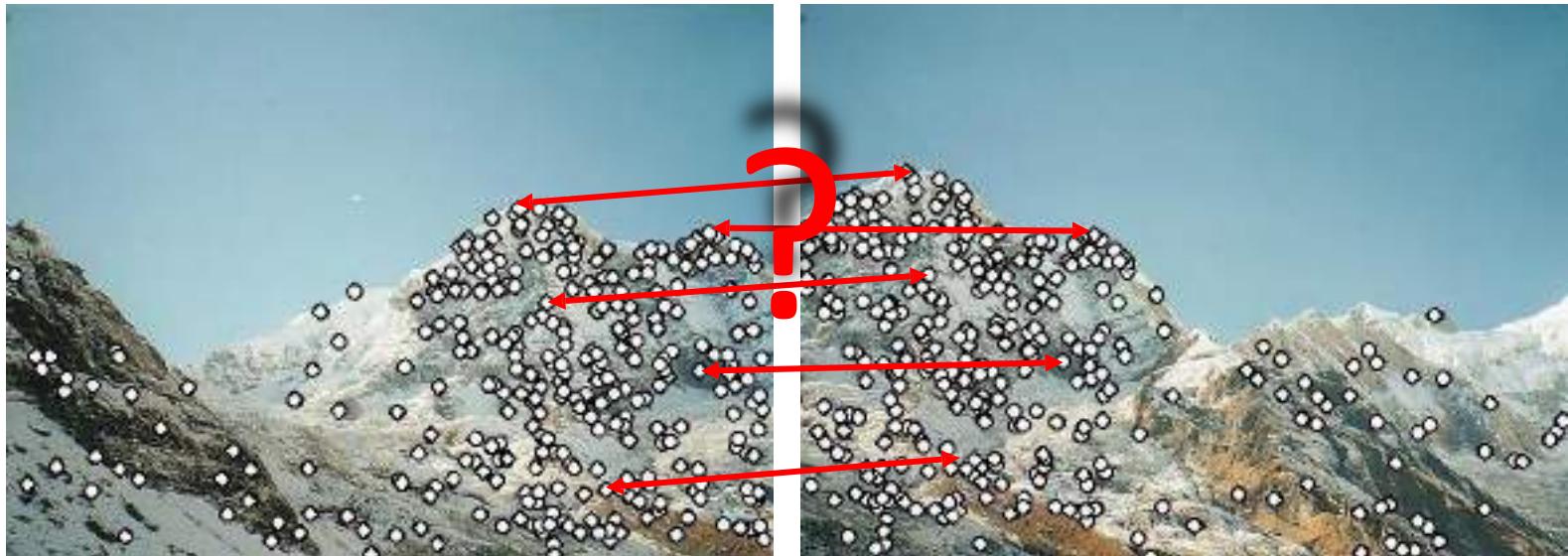


Answer: Come up with a *descriptor* for each point,
find similar descriptors between the two images

Feature descriptors

We know how to detect good points Next question:

How to match them?



Lots of possibilities (this is a popular research area)

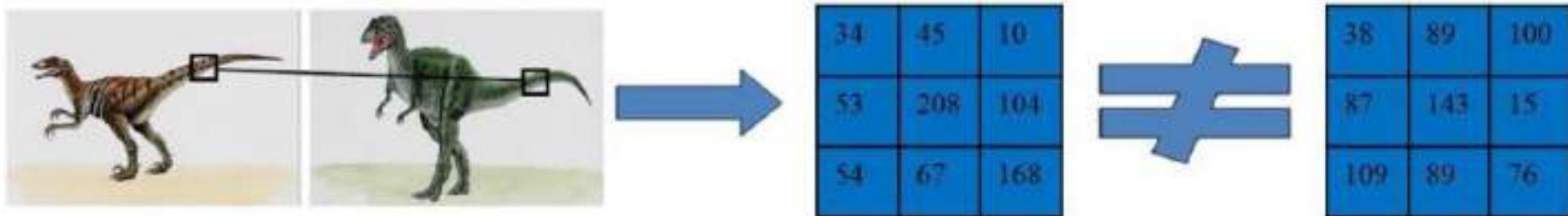
- Simple option: match square windows around the point
- State of the art approach: SIFT
 - David Lowe, UBC <http://www.cs.ubc.ca/~lowe/keypoints/>

Image/Region Matching

- Automatically recognize whether two images/regions contain the similar content.

Image/Region Matching

- Automatically recognize whether two images/regions contain the similar content.
- Comparing the image pixels as they are, will not work.



Image/Region Matching

- Pixel-based distances on high-dimensional data (and images especially) can be very unintuitive.

original



shifted



messed up



darkened



Challenges

Viewpoint variation



Scale variation



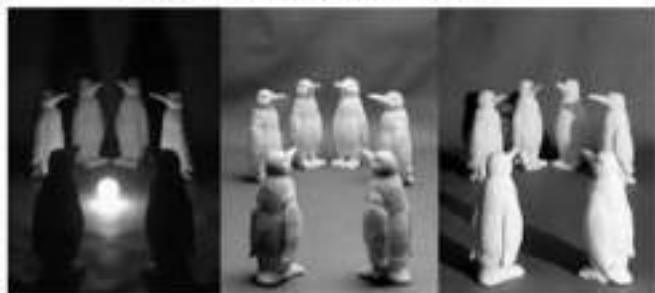
Deformation



Occlusion



Illumination conditions



Background clutter

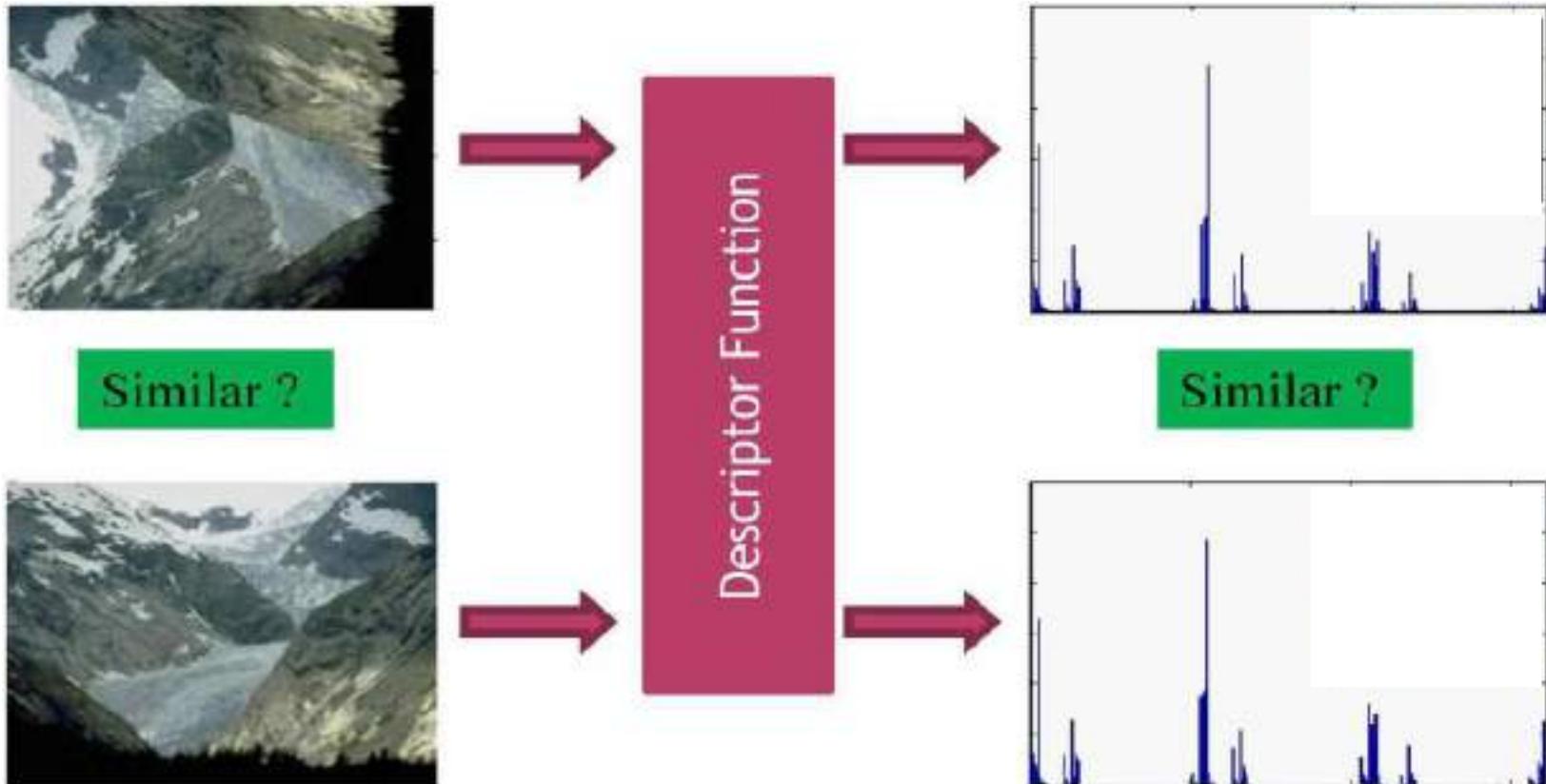


Intra-class variation



Solution

- Descriptors allow certain differences between the images.



Comparing using descriptor function
(images are taken from Corel-database and RSHD descriptor is used)

Applications

- Image Matching
- Image Retrieval
- Biomedical Image Analysis
- Texture Classification
- Image Correspondence
- Face Analysis
- Biometrics
- Building Panorama
- And many more...

Image descriptor

- Descriptions of the visual features
- Described by appearance based characteristics such as color, shape, etc.

Image descriptor

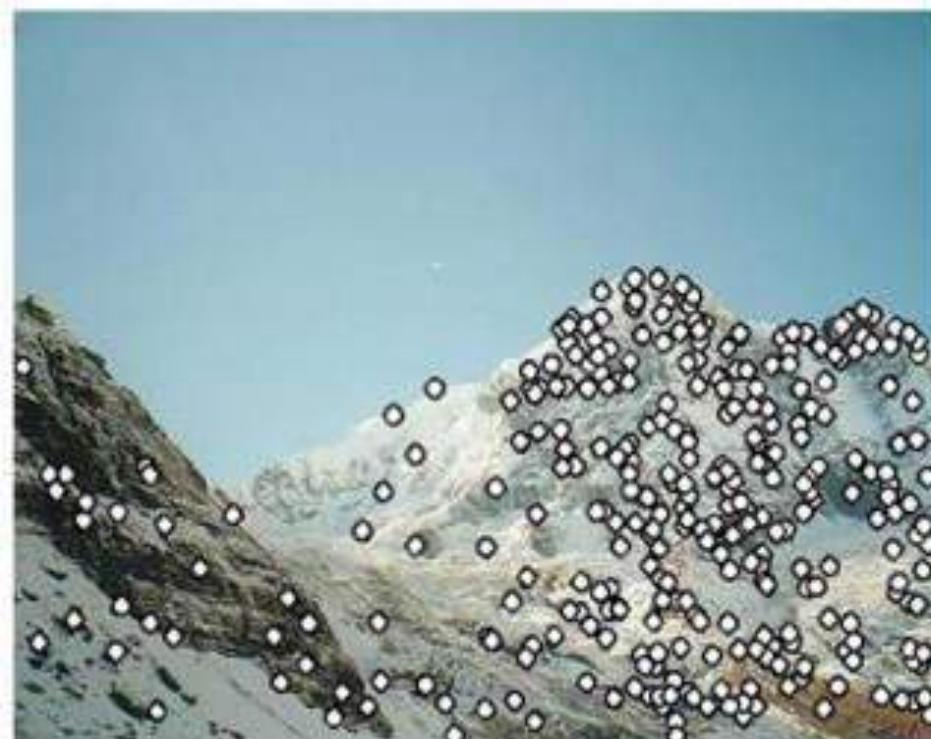
- Descriptions of the visual features
- Described by appearance based characteristics such as color, shape, etc.

A descriptor must be

- Distinctive
- Robust
- Compact
- Low Dimensional

Where to compute the descriptors?

- Over interest regions.
- Interest region may be
 - Key-Points or Global based.



Local Descriptors

- Most available descriptors focus on -
 - Edge/gradient information
 - Capture texture information
 - Exploit local relationship
 - Color also play a vital role
 - Shape features
 - Feature fusion

Widely Used Local Descriptors

- SIFT – Scale Invariant Feature Transform

Distinctive image features from scale-invariant keypoints

DG Lowe - International journal of computer vision, 2004 - Springer

... the assigned orientation, **scale**, and location for each **feature**, thereby providing **invariance** to these ... **Invariant Feature Transform** (SIFT), as it **transforms** image data into **scale-invariant** coordinates relative ... that densely cover the image over the full range of **scales** and locations ...

☆ 99 [Cited by 50252](#) Related articles All 179 versions

- LBP – Local Binary Pattern

Multiresolution gray-scale and rotation invariant texture classification with local binary patterns

T Ojala, M Pietikainen... - ... Transactions on **pattern** ..., 2002 - ieeexplore.ieee.org

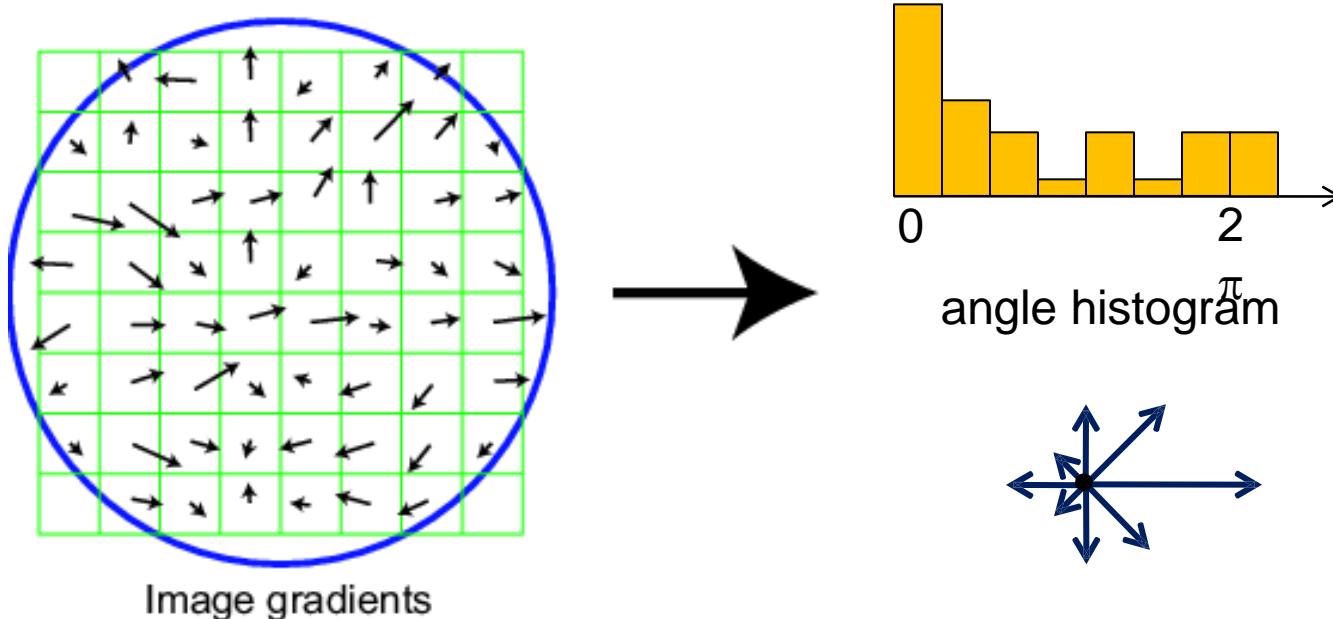
Presents a theoretically very simple, yet efficient, multiresolution approach to gray-scale and rotation invariant texture classification based on **local binary patterns** and nonparametric discrimination of sample and prototype distributions. The method is based on recognizing ...

☆ 99 [Cited by 12251](#) Related articles All 16 versions

Scale Invariant Feature Transform

Basic idea:

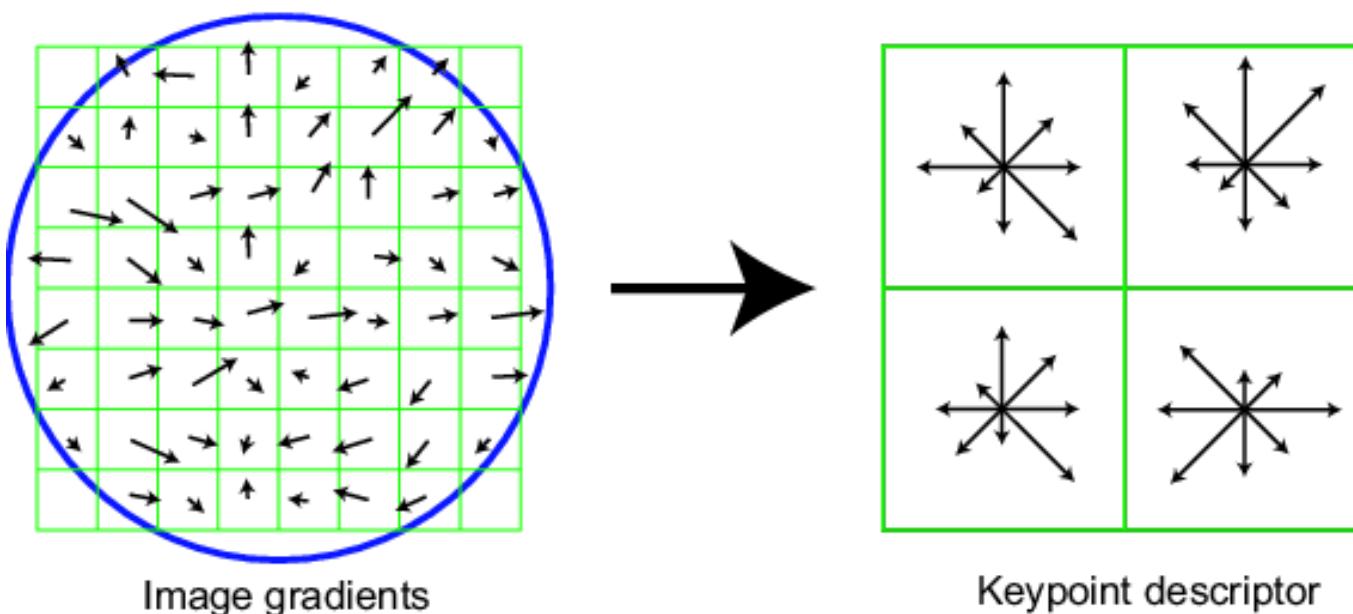
- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations



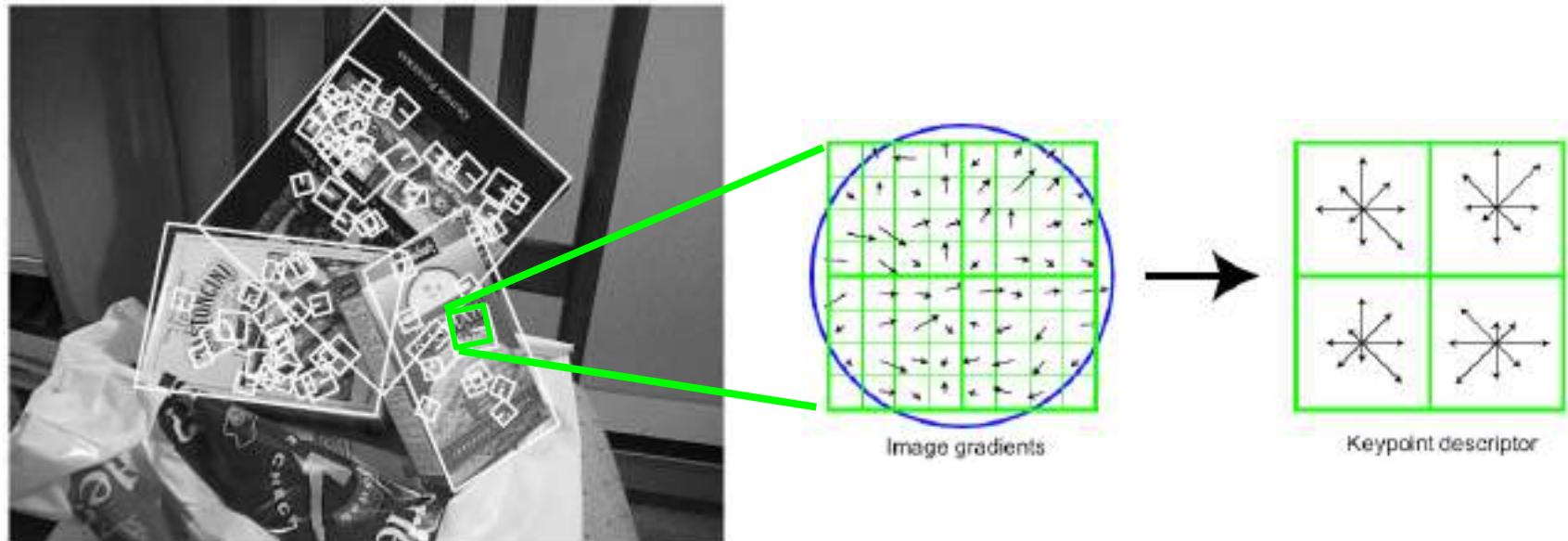
SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Local Descriptors: SIFT Descriptor



**Histogram of oriented
gradients**

- Captures important texture information
- Robust to small translations / affine deformations

[Lowe, ICCV 1999]

Feature matching

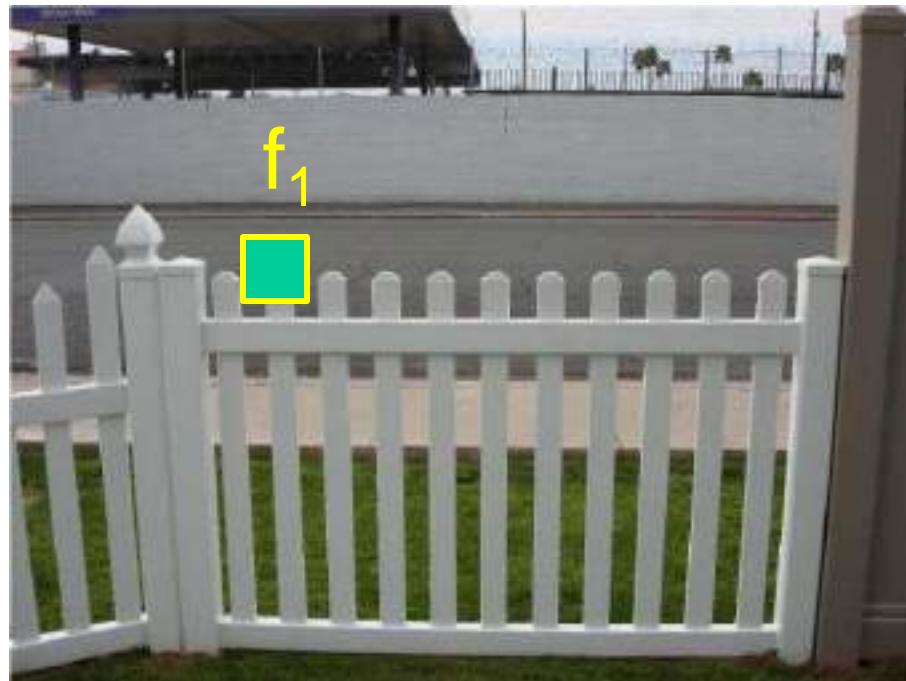
Given a feature in I_1 , how to find the best match in I_2 ?

1. Define distance function that compares two descriptors
2. Test all the features in I_2 , find the one with min distance

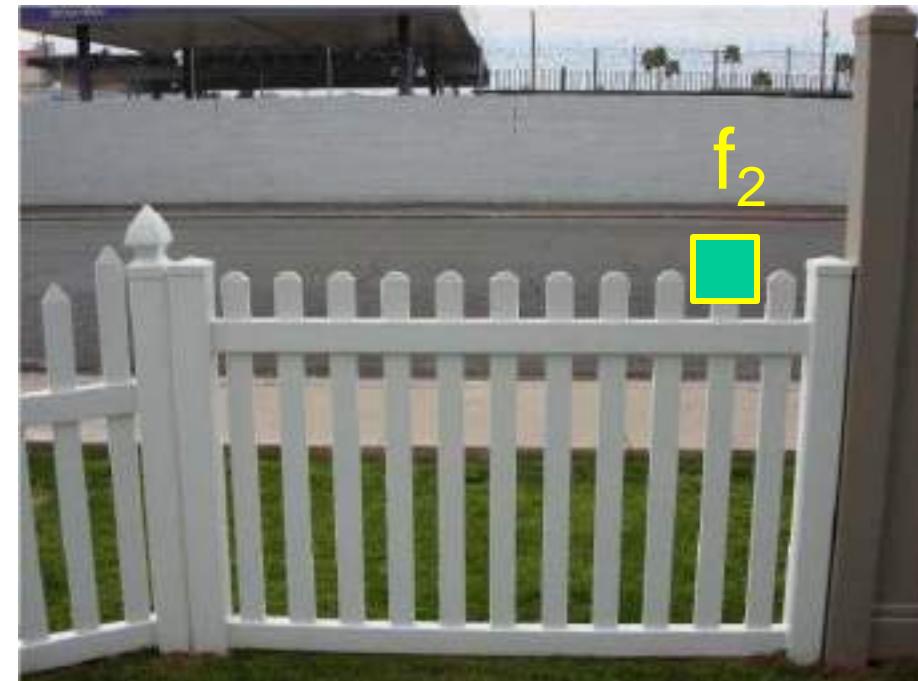
Feature distance

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

- Simple approach: L_2 distance, $\|f_1 - f_2\|$
- can give good scores to ambiguous (incorrect) matches



I_1

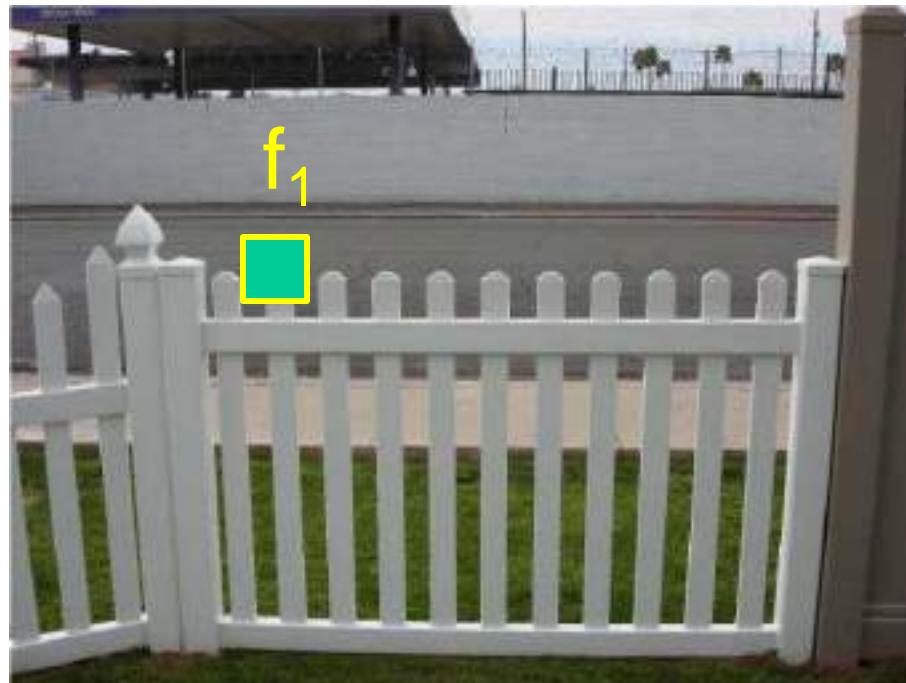


I_2

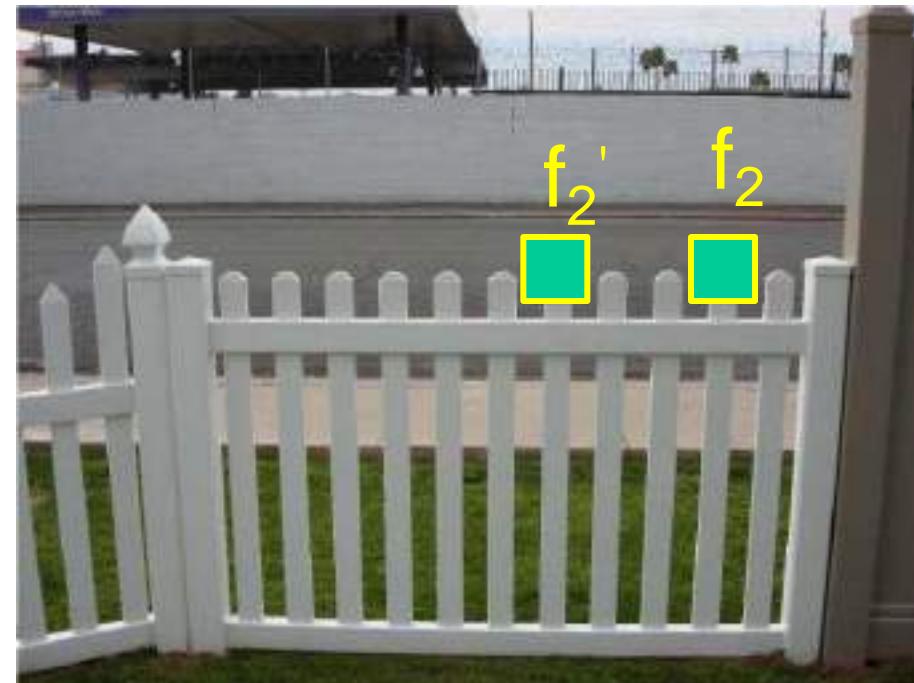
Feature distance

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 SSD match to f_1 in I_2
 - f_2' is 2nd best SSD match to f_1 in I_2
 - gives large values for ambiguous matches



I_1

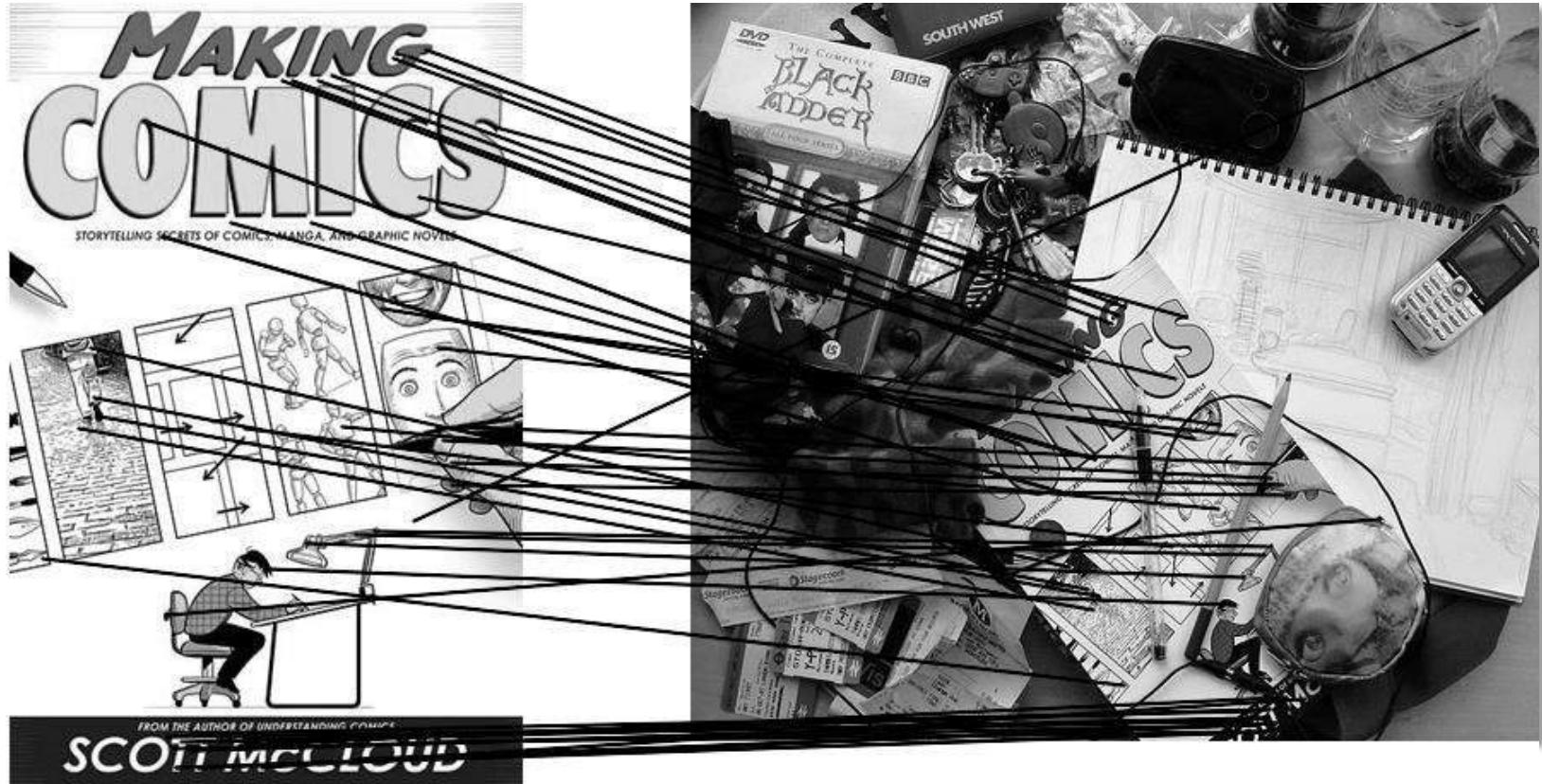


I_2

Feature matching example

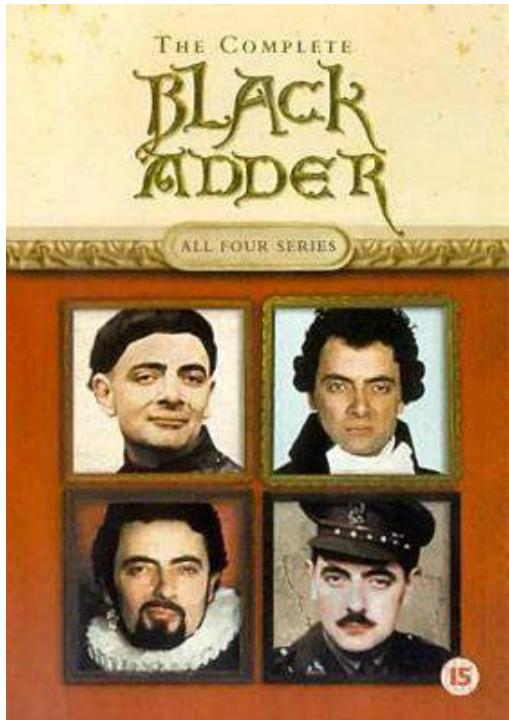


Feature matching example

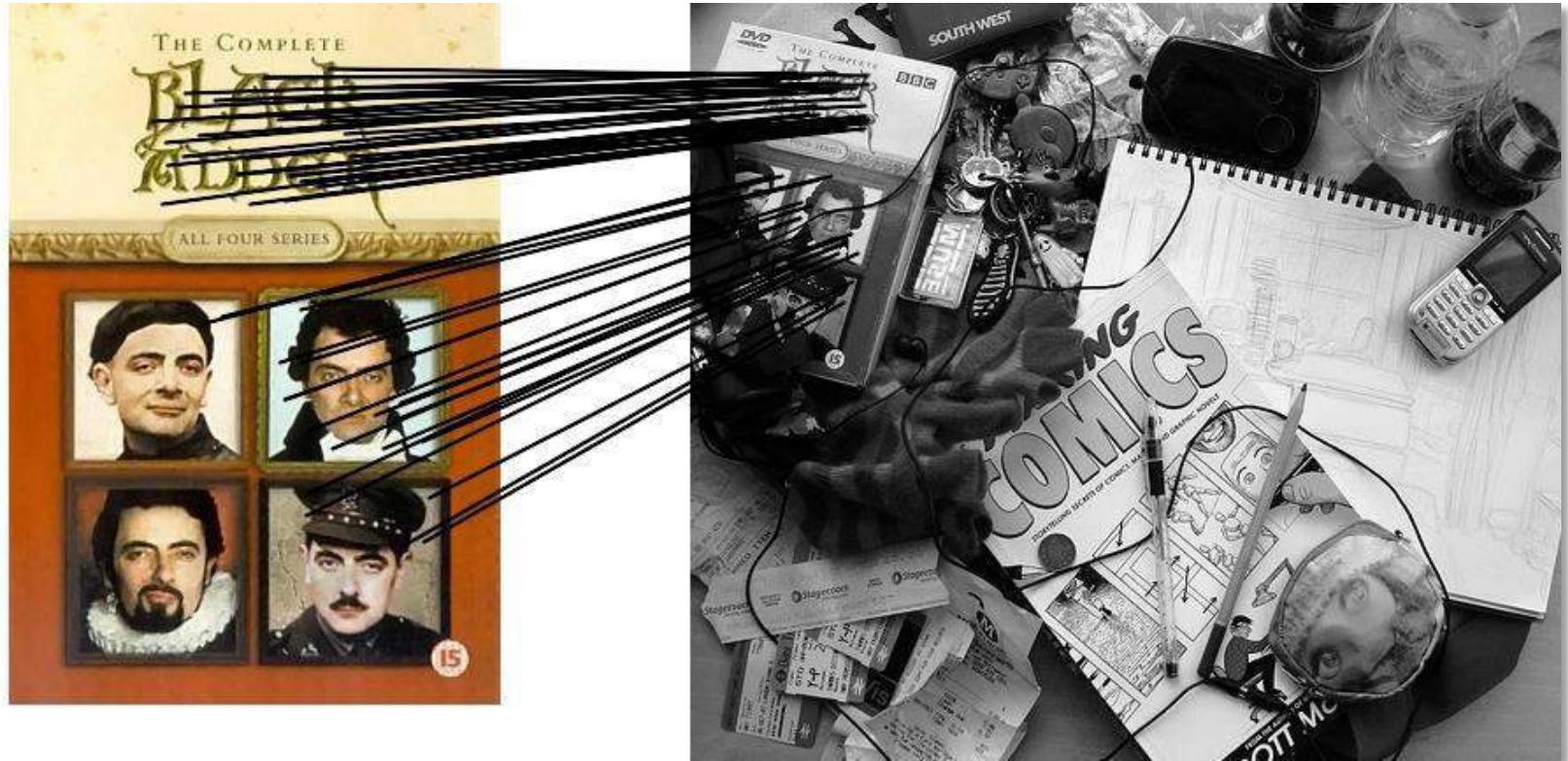


51 matches

Feature matching example



Feature matching example



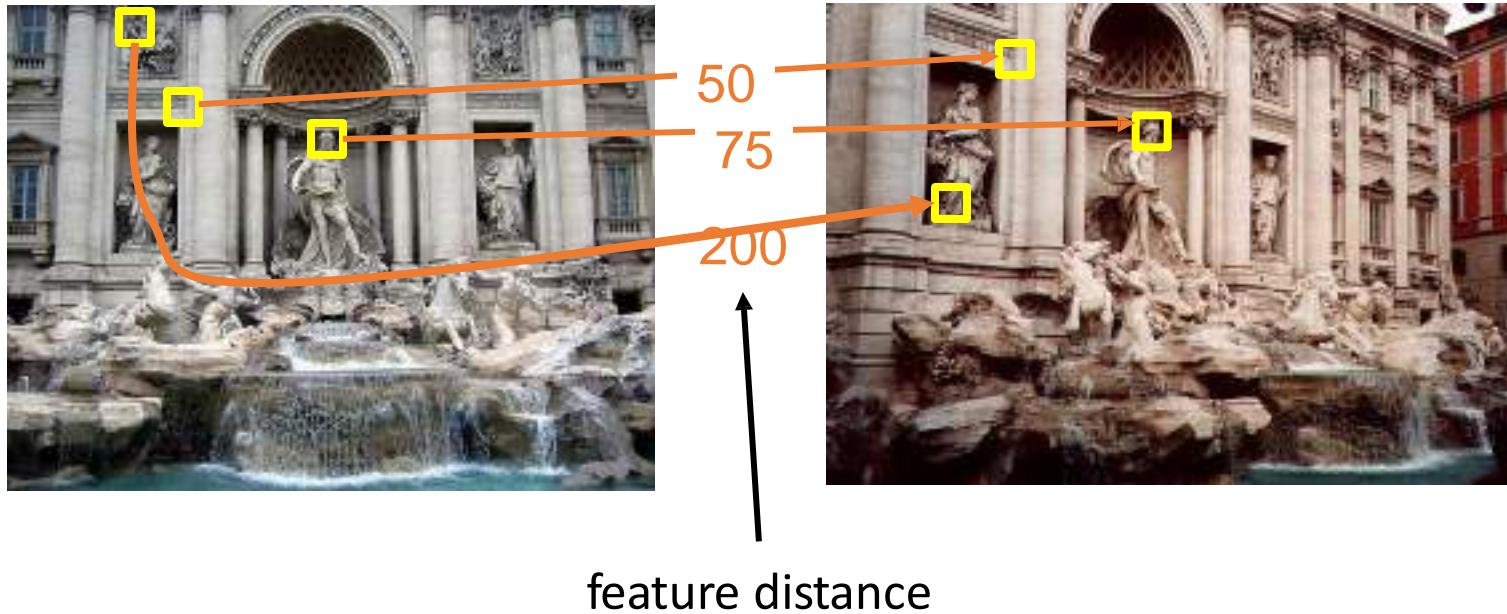
58 matches

Evaluating the results

How can we measure the performance of a feature matcher?

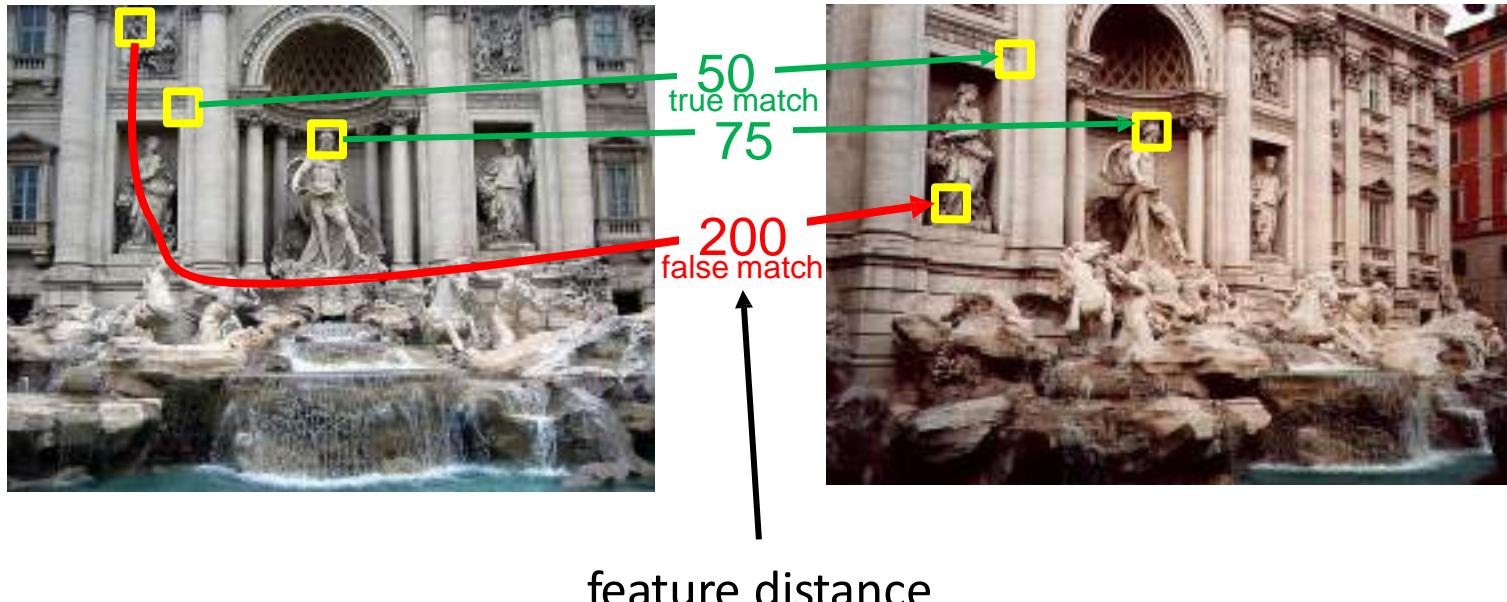
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

How can we measure the performance of a feature matcher?



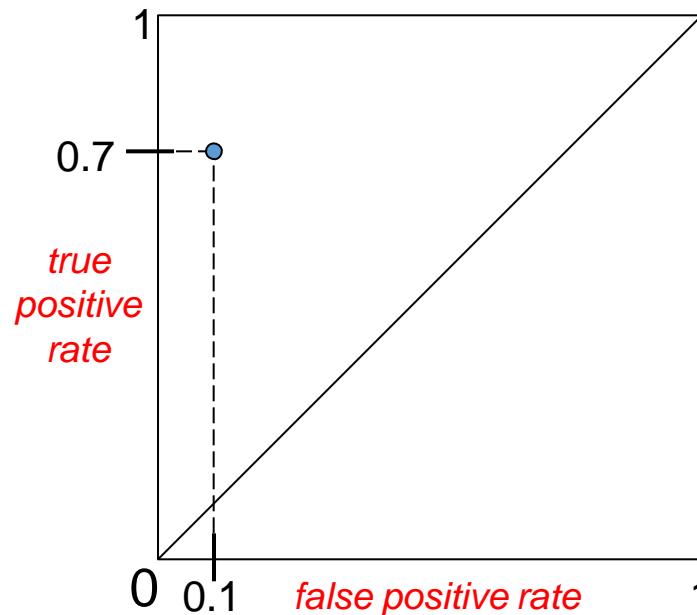
The distance threshold affects performance

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

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\text{\# true positives}}{\text{\# correctly matched features (positives)}} \quad \text{"recall"}$$

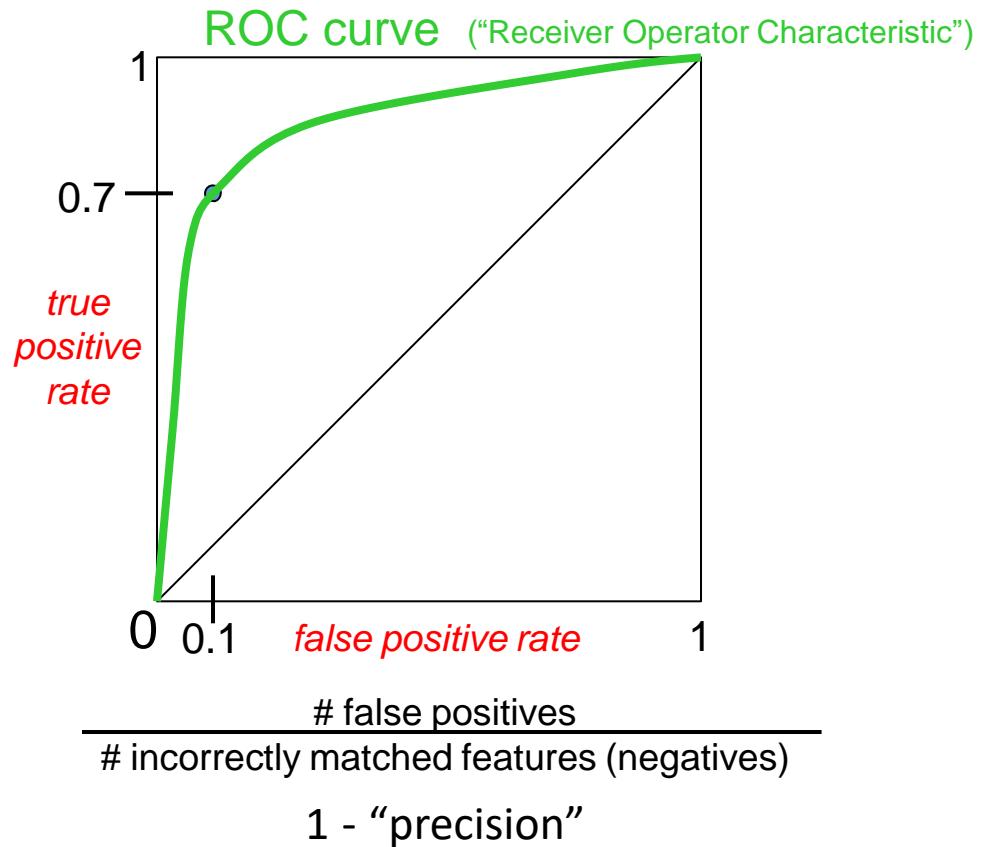


$$\frac{\text{\# false positives}}{\text{\# incorrectly matched features (negatives)}} \quad 1 - \text{"precision"}$$

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\# \text{ true positives}}{\# \text{ correctly matched features} \text{ (positives)}} \text{ "recall"}$$



Evaluating the results

- ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
- Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.
- ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

Local Binary Pattern (LBP)

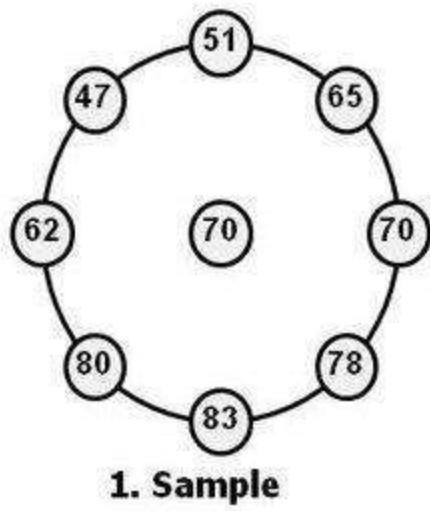


Image Source: scholarpedia

[IEEE TPAMI 2002]

Local Binary Pattern (LBP)

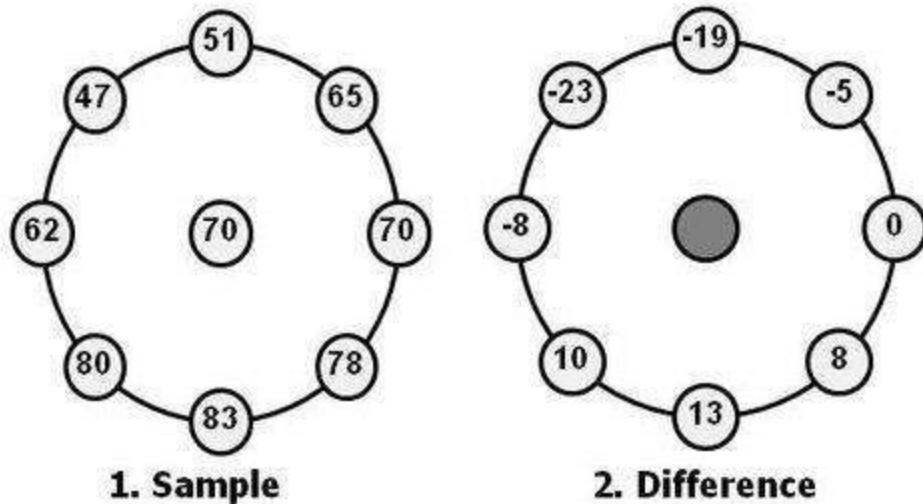


Image Source: scholarpedia

[IEEE TPAMI 2002]

Local Binary Pattern (LBP)

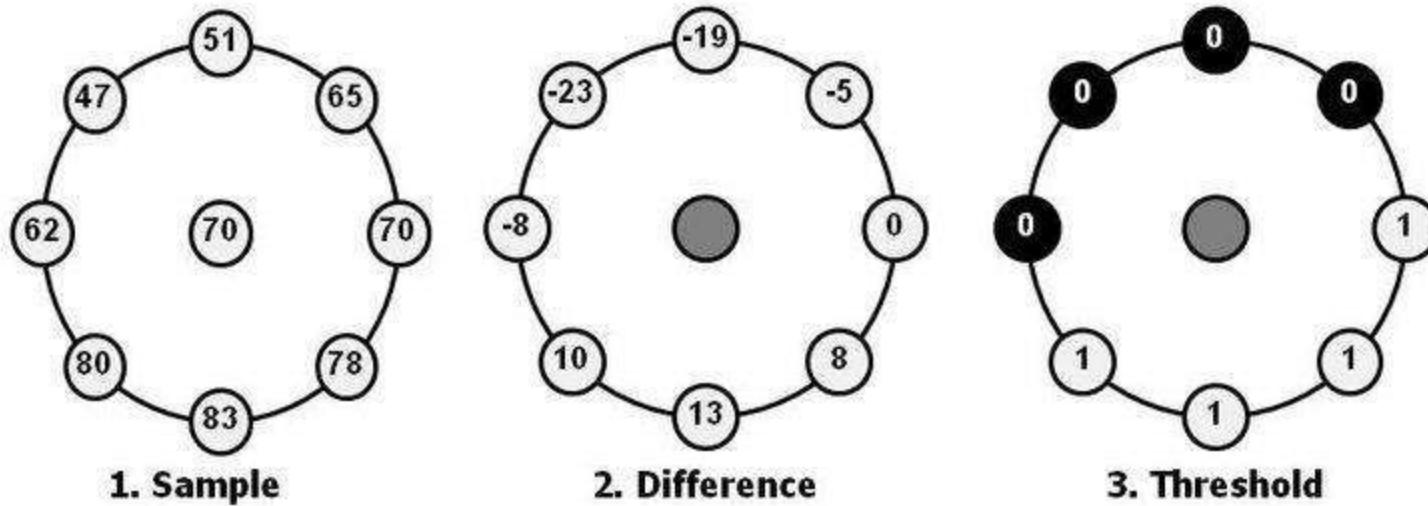
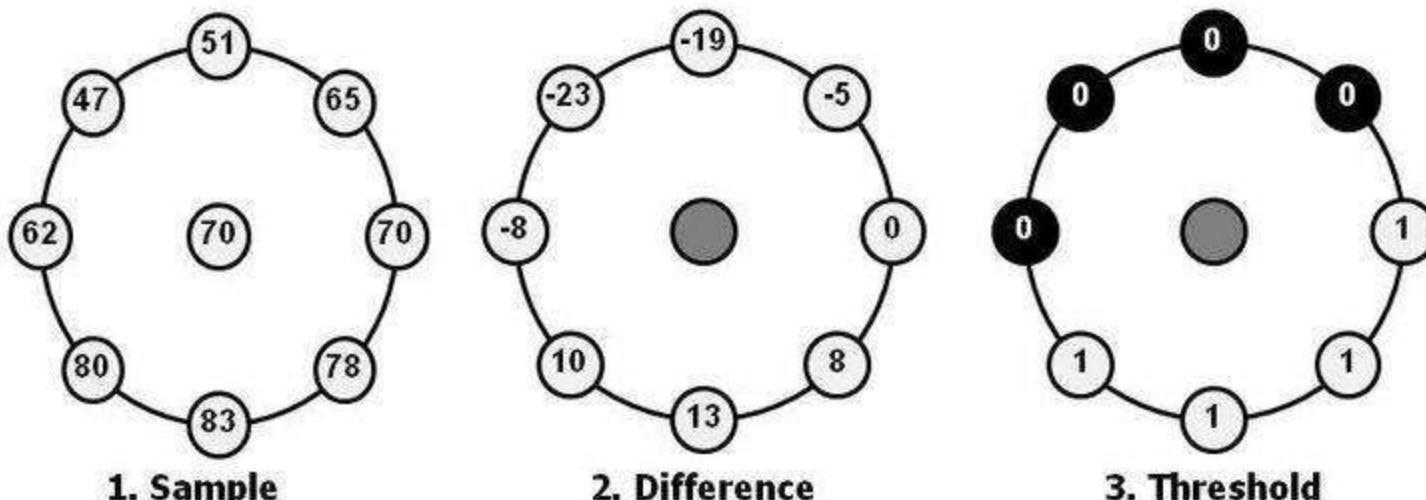


Image Source: scholarpedia

[IEEE TPAMI 2002]

Local Binary Pattern (LBP)



$$1 \cdot 1 + 1 \cdot 2 + 1 \cdot 4 + 1 \cdot 8 + 0 \cdot 16 + 0 \cdot 32 + 0 \cdot 64 + 0 \cdot 128 = 15$$

4. Multiply by powers of two and sum

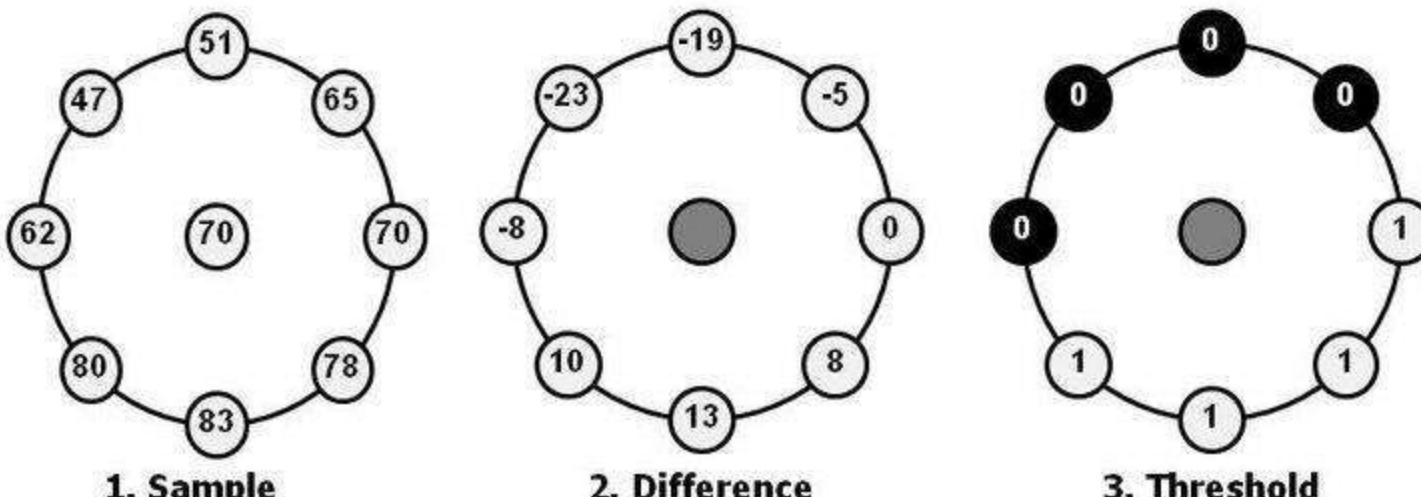
Image Source: scholarpedia

[IEEE TPAMI 2002]

Local Binary Pattern (LBP)

The value of the LBP code of a pixel (x_c, y_c) is given by:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \quad s(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$



$$1*1 + 1*2 + 1*4 + 1*8 + 0*16 + 0*32 + 0*64 + 0*128 = 15$$

4. Multiply by powers of two and sum

Image Source: scholarpedia

[IEEE TPAMI 2002]

Local Binary Pattern (LBP)

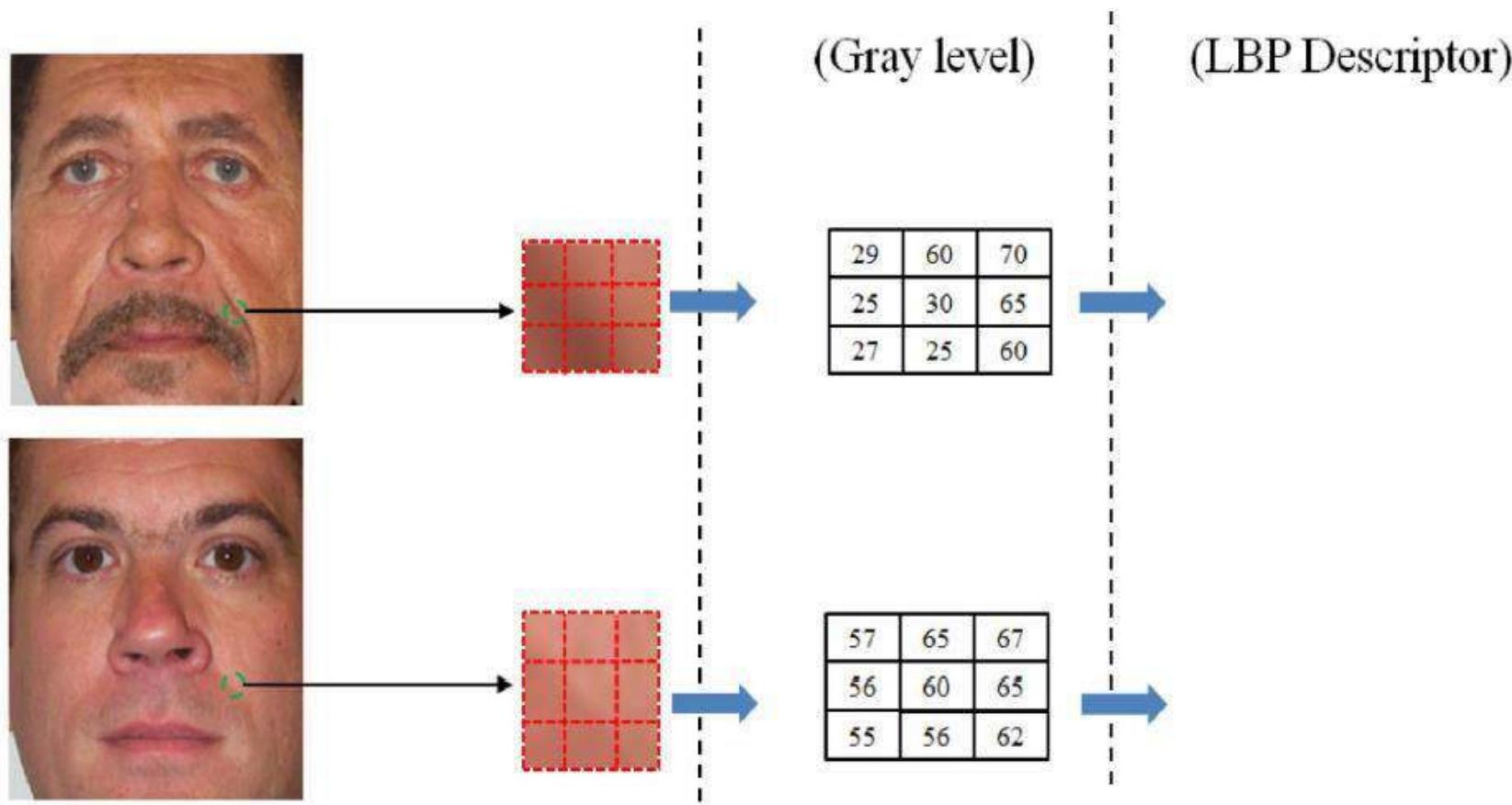
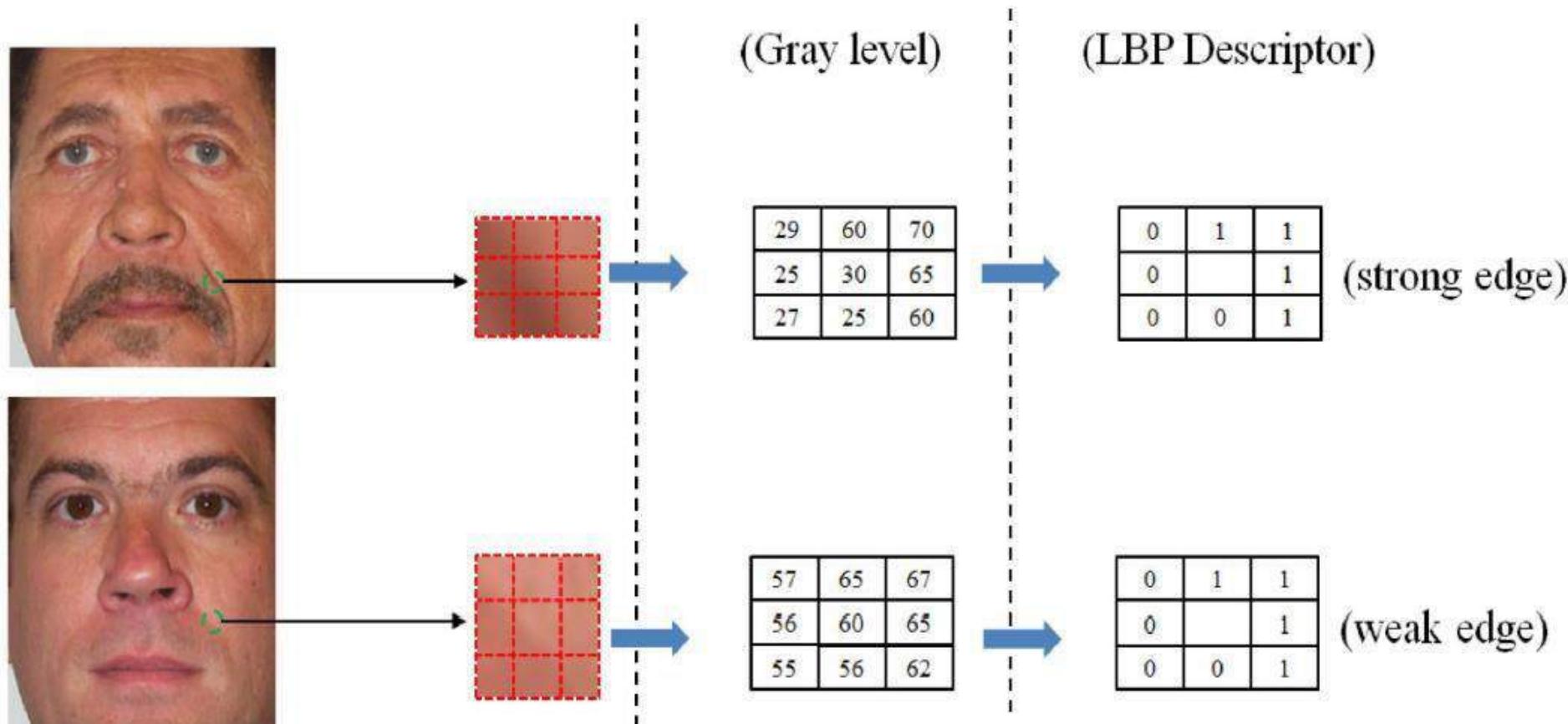


Image Source: Nguyen, D. T., Cho, S. R., & Park, K. R. (2015). Age Estimation-Based Soft Biometrics Considering Optical Blurring Based on Symmetrical Sub-Blocks for MLBP. *Symmetry*, 7(4), 1882-1913.

Local Binary Pattern (LBP)



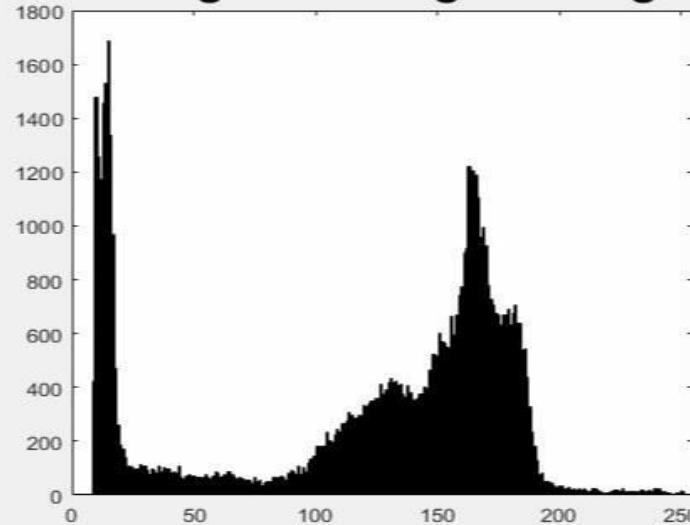
*Image Source: Nguyen, D. T., Cho, S. R., & Park, K. R. (2015). Age Estimation-Based Soft Biometrics Considering Optical Blurring Based on Symmetrical Sub-Blocks for MLBP. *Symmetry*, 7(4), 1882-1913.*

Local Binary Pattern (LBP)

Original Grayscale Image



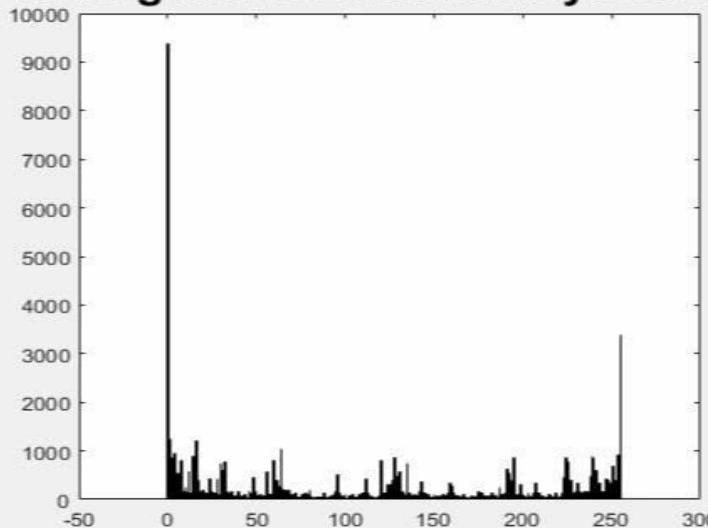
Histogram of original image



Local Binary Pattern



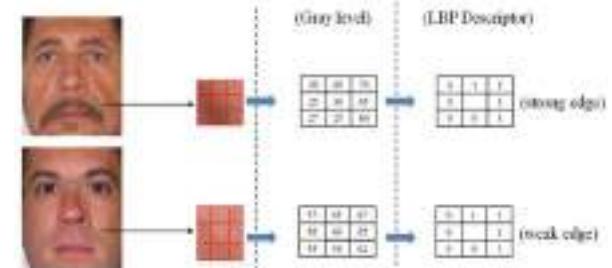
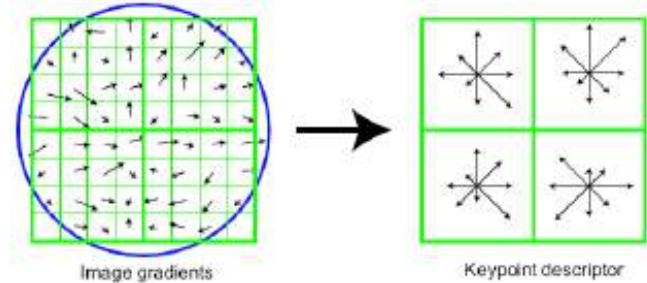
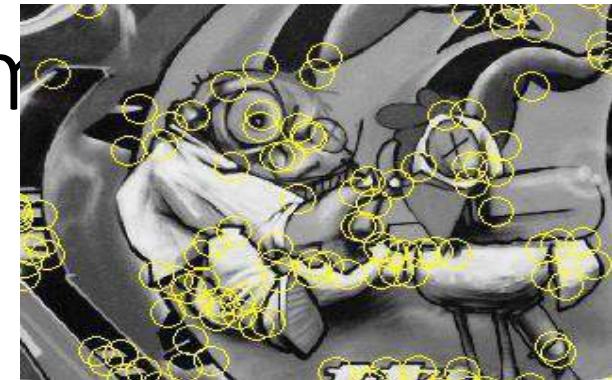
Histogram of Local Binary Pattern



*Image Source:
Mathworks*

Things to remember

- Region detection: repeatable and distinctive
 - Blobs, DoG, stable regions
 - Invariant to affine transformation
- Local Descriptors:
 - Discriminative
 - Robust
 - Compact
 - Efficient
- Scale Invariant Feature Transform
 - Utilizes gradient information
 - Gradient direction binning
- Local Binary Pattern
 - Exploits local relationship
 - Captures edge, spot, corner, etc.

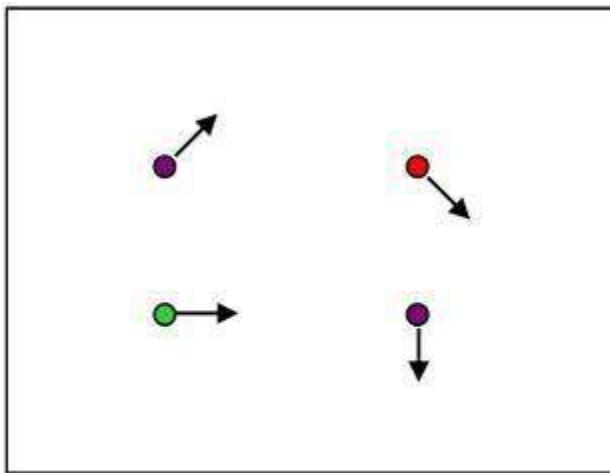


Acknowledgements

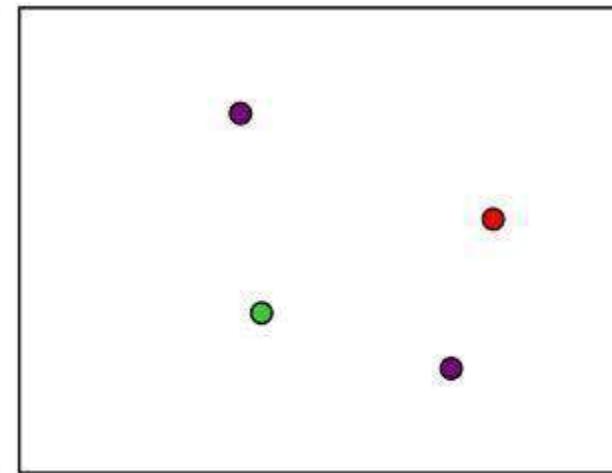
- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Leibe

Thank you

- Next class: feature tracking and optical flow



$$I(x,y,t)$$



$$I(x,y,t+1)$$

Computer Vision

Feature Tracking and

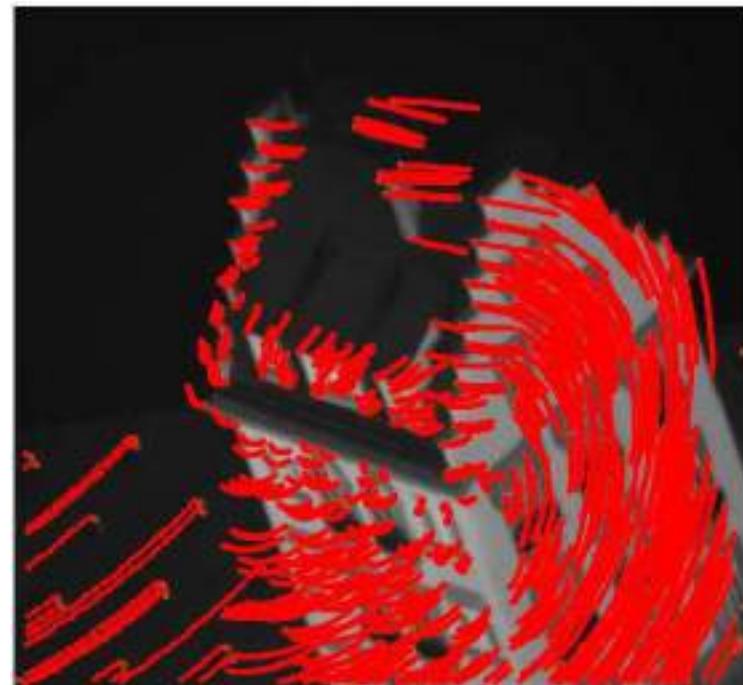
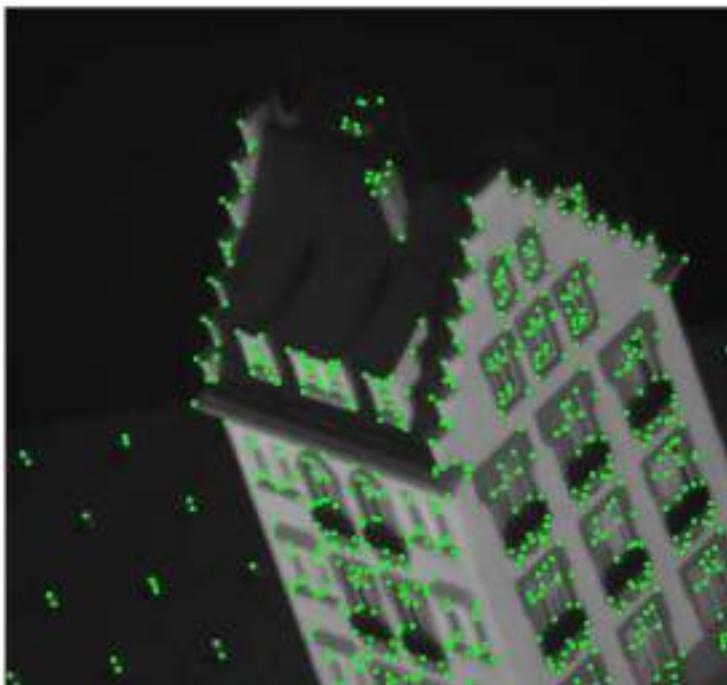
Optical Flow

Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**



Feature Tracking and Optical Flow



This class: recovering motion

- Feature tracking
 - Extract visual features (corners, textured areas) and “track” them over multiple frames
- Optical flow
 - Recover image motion at each pixel from spatio-temporal image brightness variations

B. Lucas and T. Kanade. [An iterative image registration technique with an application to stereo vision.](#) In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.

Feature tracking - Challenges

- Figure out which features can be tracked

Feature tracking - Challenges

- Figure out which features can be tracked
- Efficiently track across frames

Feature tracking - Challenges

- Figure out which features can be tracked
- Efficiently track across frames
- Some points may change appearance over time
(e.g., due to rotation, moving into shadows, etc.)

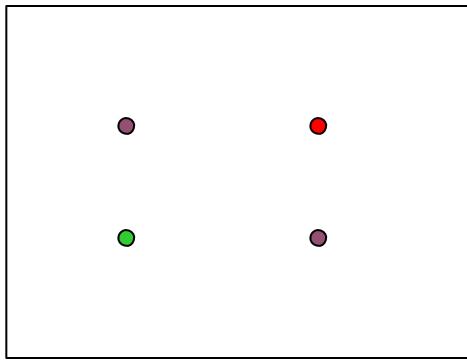
Feature tracking - Challenges

- Figure out which features can be tracked
- Efficiently track across frames
- Some points may change appearance over time
(e.g., due to rotation, moving into shadows, etc.)
- Points may appear or disappear: need to be able to add/delete tracked points

Feature tracking - Challenges

- Figure out which features can be tracked
- Efficiently track across frames
- Some points may change appearance over time
(e.g., due to rotation, moving into shadows, etc.)
- Points may appear or disappear: need to be able to add/delete tracked points
- Drift: small errors can accumulate as appearance model is updated

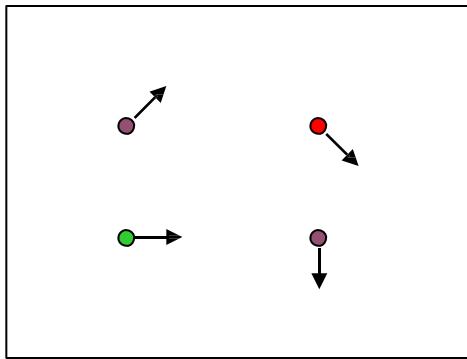
Feature tracking: Lucas-Kanade



$$I(x,y,t)$$

- Given two subsequent frames, estimate the point translation

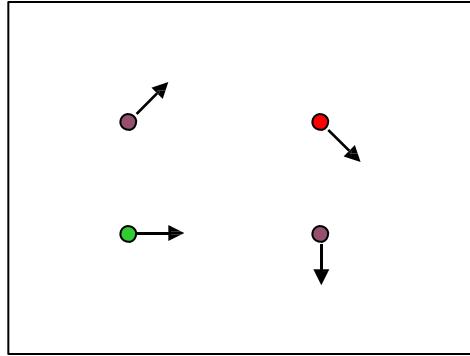
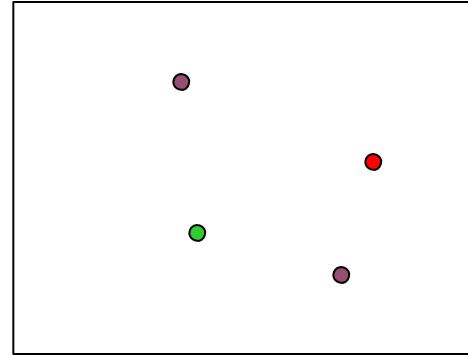
Feature tracking: Lucas-Kanade



$$I(x,y,t)$$

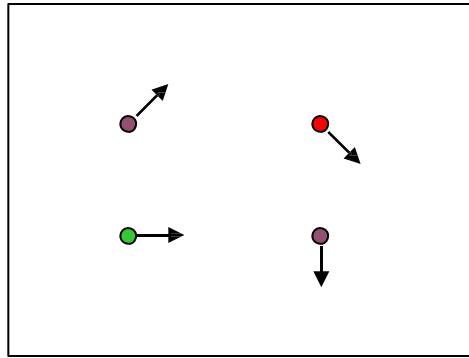
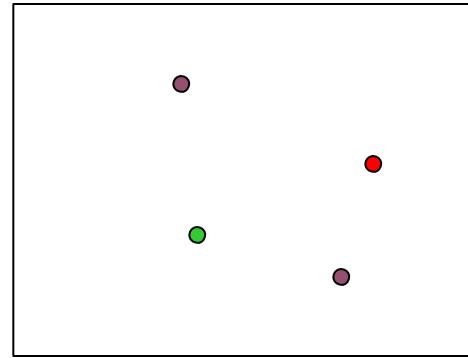
- Given two subsequent frames, estimate the point translation

Feature tracking: Lucas-Kanade

 $I(x,y,t)$  $I(x,y,t+1)$

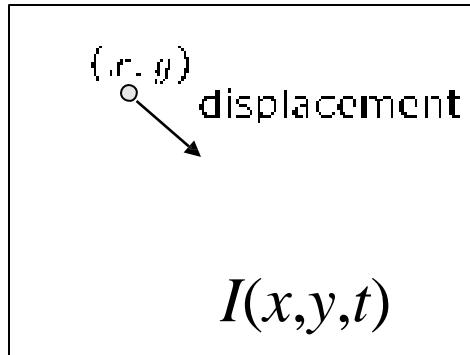
- Given two subsequent frames, estimate the point translation

Feature tracking: Lucas-Kanade

 $I(x,y,t)$  $I(x,y,t+1)$

- Given two subsequent frames, estimate the point translation
- Key assumptions of Lucas-Kanade Tracker
 - **Brightness constancy:** projection of the same point looks the same in every frame
 - **Small motion:** points do not move very far
 - **Spatial coherence:** points move like their neighbors

The brightness constancy constraint

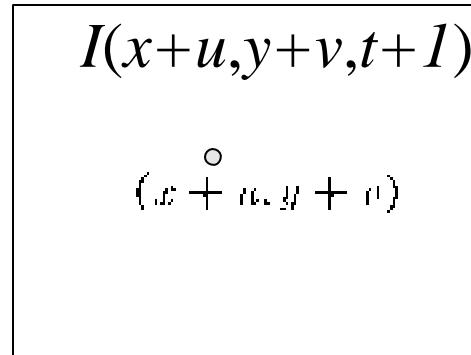
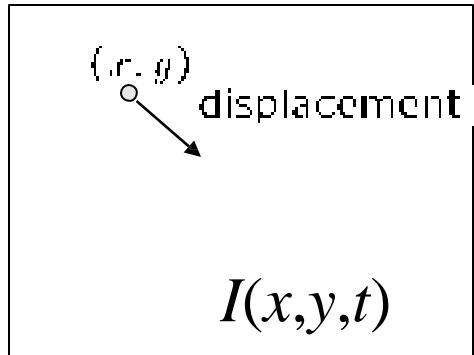


$$I(x+u, y+v, t+1)$$
$$(x + \overset{\circ}{u}, y + v)$$

- Brightness Constancy Equation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

The brightness constancy constraint



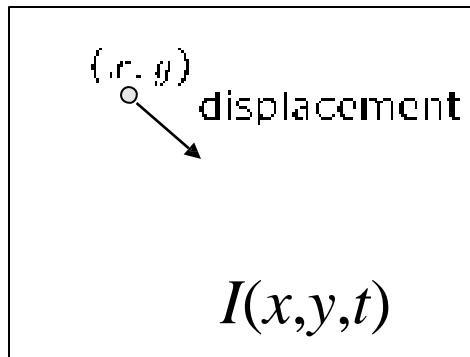
- Brightness Constancy Equation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the rightside:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x \cdot u + I_y \cdot v + I_t$$

The brightness constancy constraint



A diagram enclosed in a box. At the top is the function $I(x+u, y+v, t+1)$. Below it is the resulting coordinate pair $(x + u, y + v)$, also with a small circle.

- Brightness Constancy Equation:

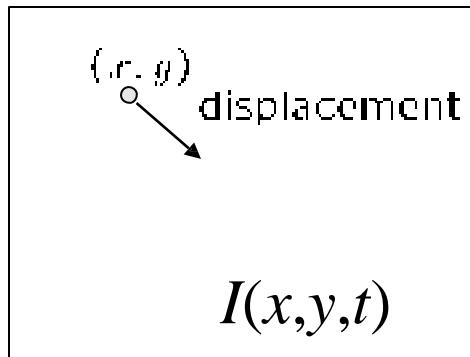
$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the right side:

derivative
along x

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + \boxed{I_x} \cdot u + I_y \cdot v + I_t$$

The brightness constancy constraint



$$I(x+u, y+v, t+1)$$

$\stackrel{\circ}{(x+u, y+v)}$

- Brightness Constancy Equation:

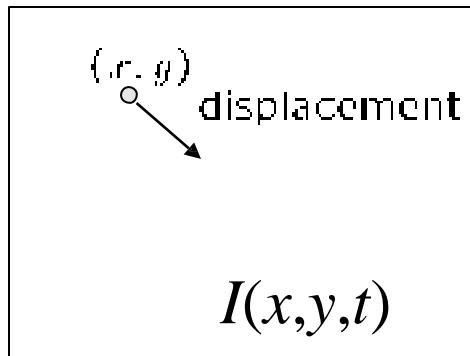
$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the right side:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + \boxed{I_x} \cdot u + \boxed{I_y} \cdot v + I_t$$

derivative along x derivative along y

The brightness constancy constraint



A diagram enclosed in a box. At the top right is the function $I(x+u, y+v, t+1)$. Below it is a coordinate pair $(x + u, y + v)$ with a small circle and a dot above it.

- Brightness Constancy Equation:

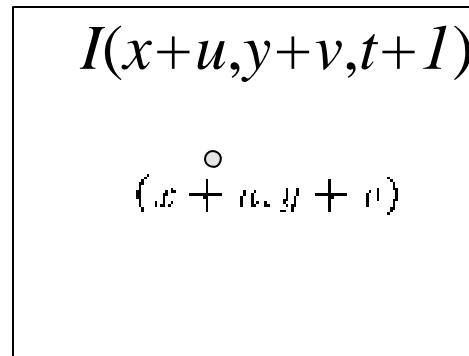
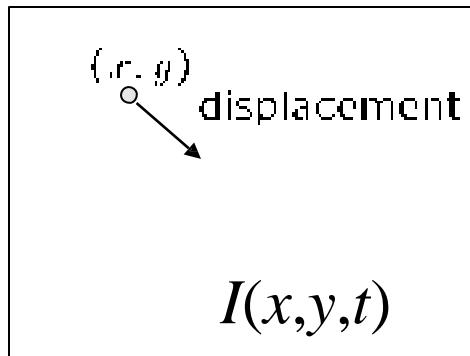
$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the right side:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + [I_x] \cdot u + [I_y] \cdot v + [I_t]$$

derivative along x derivative along y derivative along t
i.e., difference over frames

The brightness constancy constraint



- Brightness Constancy Equation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

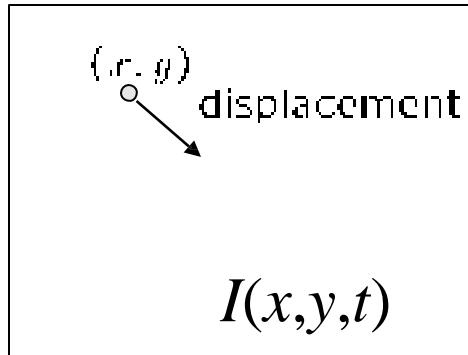
Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the right side:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + \boxed{I_x} \cdot u + \boxed{I_y} \cdot v + \boxed{I_t}$$

derivative along x derivative along y derivative along t
i.e., difference over frames

$$I(x + u, y + v, t + 1) - I(x, y, t) = I_x \cdot u + I_y \cdot v + I_t$$

The brightness constancy constraint



$$I(x+u, y+v, t+1)$$

$\stackrel{\circ}{(x+u, y+v)}$

- Brightness Constancy Equation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x+u, y+v, t+1)$ at (x, y, t) to linearize the right side:

$$I(x+u, y+v, t+1) \approx I(x, y, t) + \boxed{I_x} \cdot u + \boxed{I_y} \cdot v + \boxed{I_t}$$

derivative along x derivative along y derivative along t
i.e., difference over frames

$$I(x+u, y+v, t+1) - I(x, y, t) = I_x \cdot u + I_y \cdot v + I_t$$

So: $I_x \cdot u + I_y \cdot v + I_t \approx 0 \rightarrow \nabla I \cdot [u \ v]^T + I_t = 0$

The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?

The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?
 - One equation (this is a scalar equation!), two unknowns (u, v)

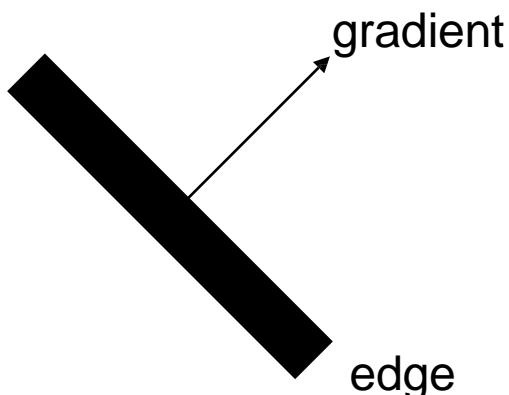
The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?
 - One equation (this is a scalar equation!), two unknowns (u, v)

The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured



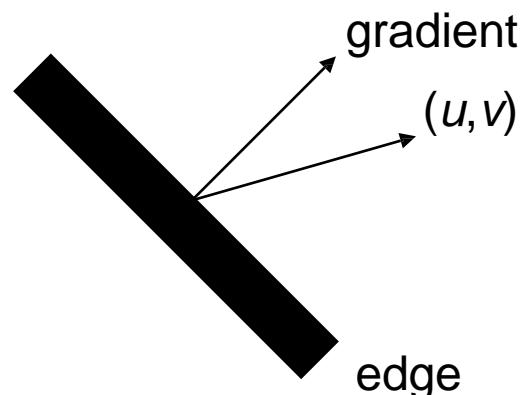
The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?
 - One equation (this is a scalar equation!), two unknowns (u, v)

The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured



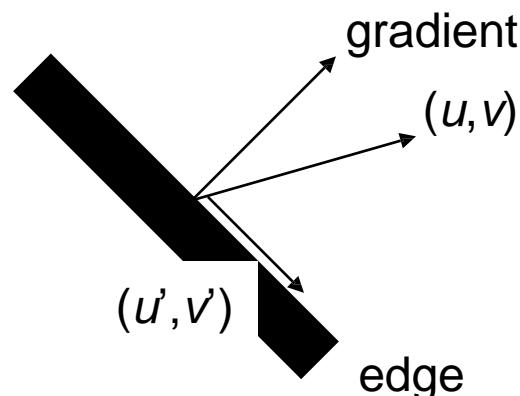
The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?
 - One equation (this is a scalar equation!), two unknowns (u, v)

The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured



The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

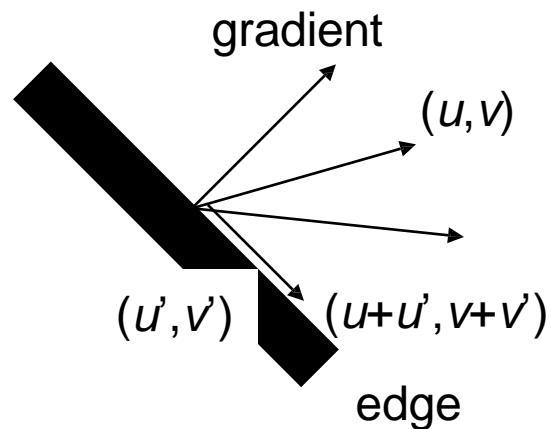
$$\nabla I \cdot [u \ v]^T + I_t = 0$$

- How many equations and unknowns per pixel?
- One equation (this is a scalar equation!), two unknowns (u, v)

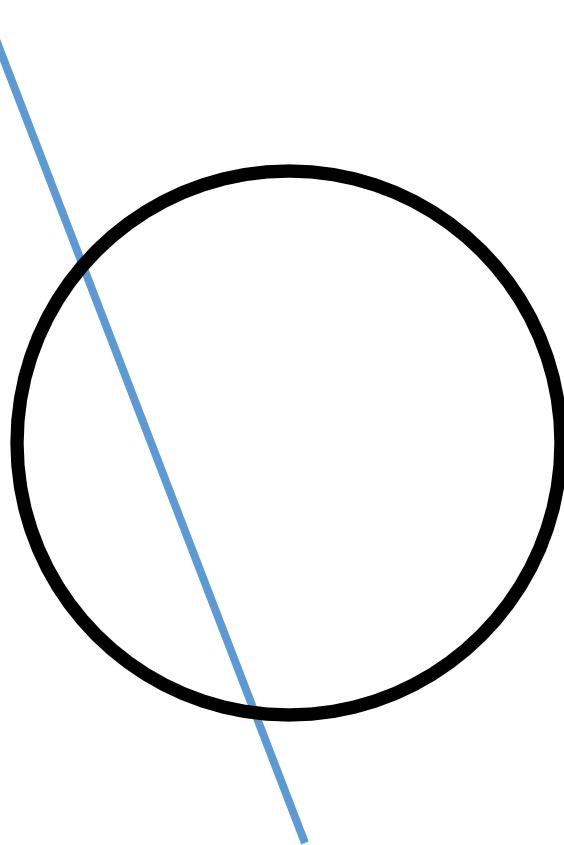
The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured

If (u, v) satisfies the equation, so does $(u+u', v+v')$ if

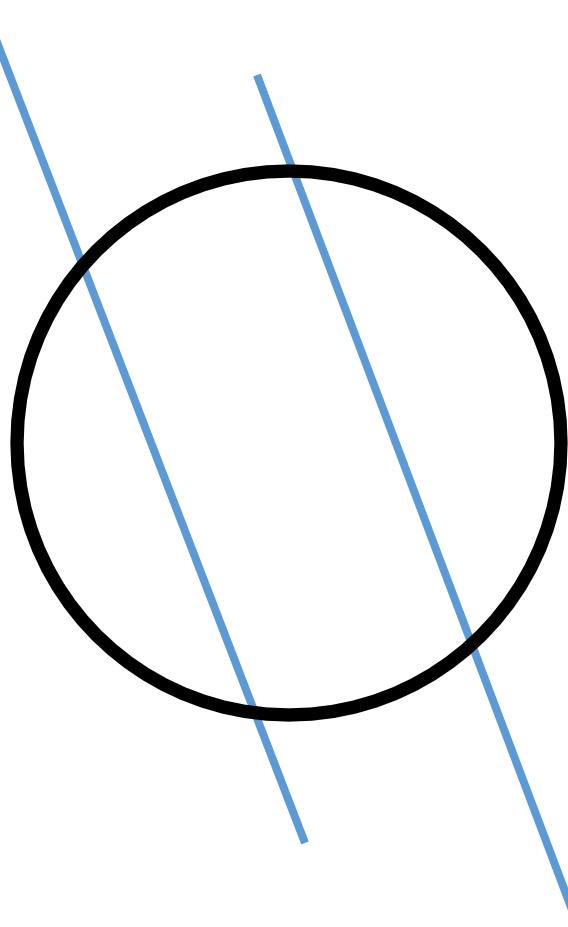
$$\nabla I \cdot [u' \ v']^T = 0$$



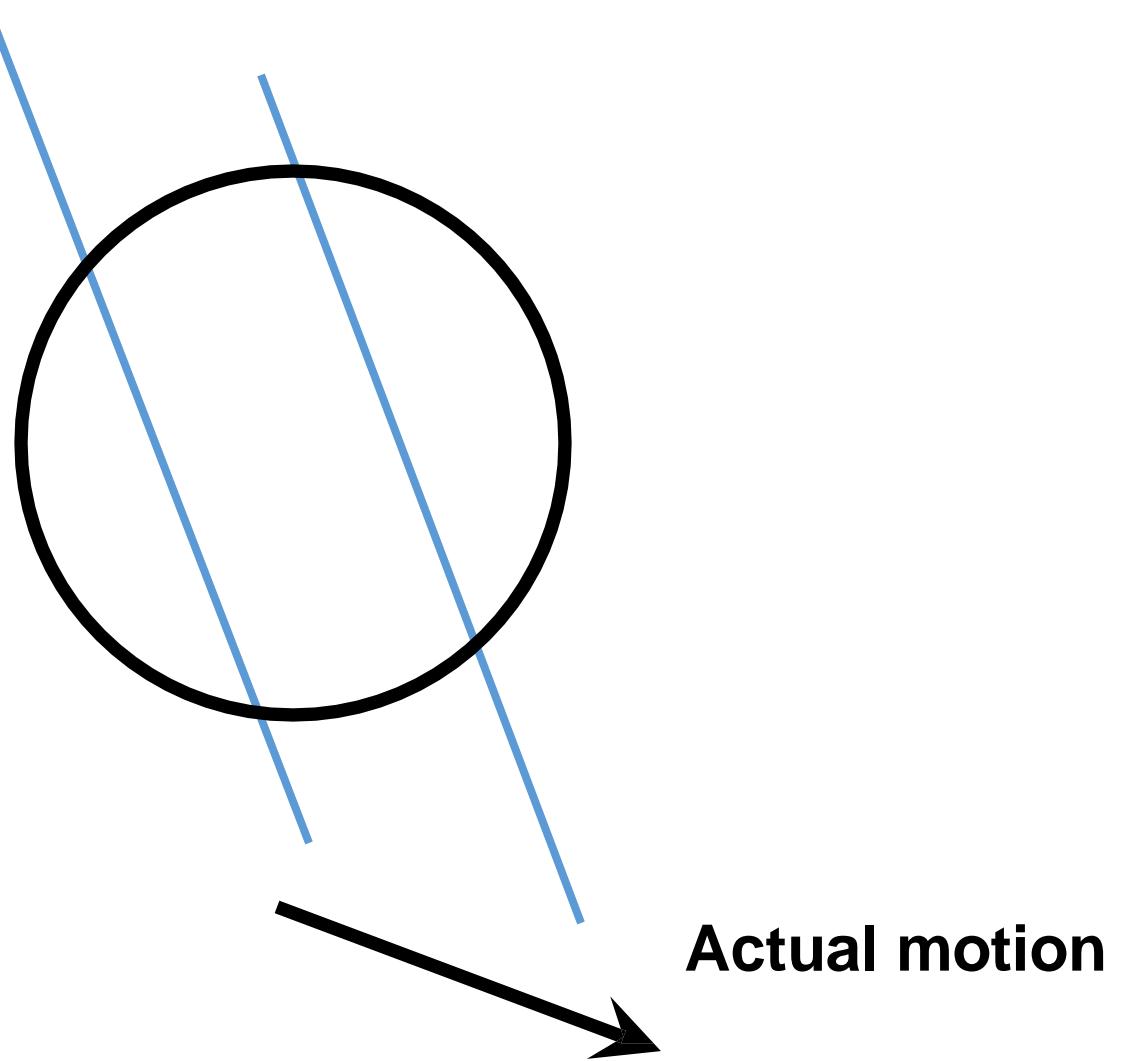
The aperture problem



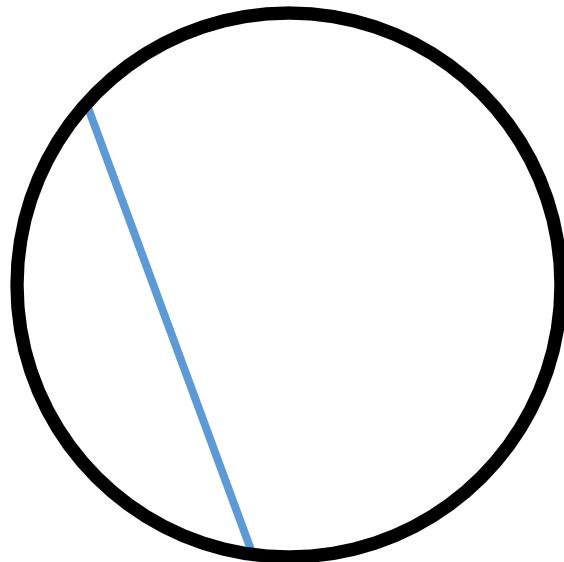
The aperture problem



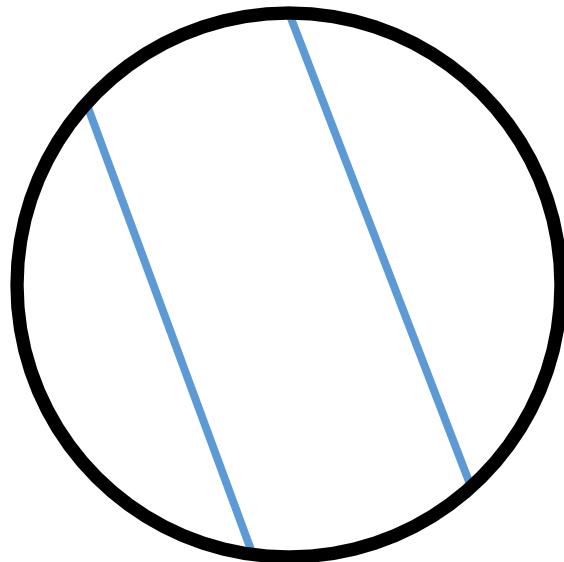
The aperture problem



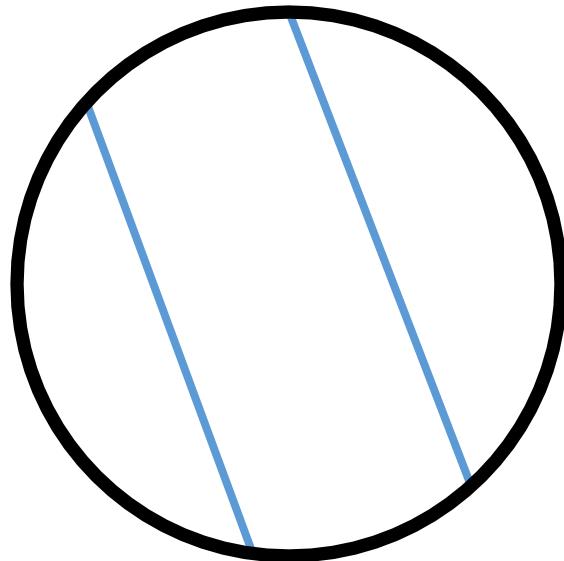
The aperture problem



The aperture problem



The aperture problem



Perceived motion

Solving the ambiguity...

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

Solving the ambiguity...

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

- How to get more equations for a pixel?

Solving the ambiguity...

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

- How to get more equations for a pixel?
- **Spatial coherence constraint**
 - Assume the pixel's neighbors have the same (u,v)
 - If we use a 5×5 window, that gives us 25 equations per pixel

Solving the ambiguity...

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

- How to get more equations for a pixel?
- **Spatial coherence constraint**
 - Assume the pixel's neighbors have the same (u, v)
 - If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p}_i) + \nabla I(\mathbf{p}_i) \cdot [u \ v]$$

Solving the ambiguity...

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

- How to get more equations for a pixel?
- **Spatial coherence constraint**
 - Assume the pixel's neighbors have the same (u, v)
 - If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p}_i) + \nabla I(\mathbf{p}_i) \cdot [u \ v]$$

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

Matching patches across images

- Overconstrained linear system

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$A \quad d = b$
 $25 \times 2 \quad 2 \times 1 \quad 25 \times 1$

Matching patches across images

- Overconstrained linear system

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$A \quad d = b$
 $25 \times 2 \quad 2 \times 1 \quad 25 \times 1$

Least squares solution for d is given by $(A^T A)^{-1} A^T b$

Matching patches across images

- Overconstrained linear system

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

$A \quad d = b$
 $25 \times 2 \quad 2 \times 1 \quad 25 \times 1$

Least squares solution for d is given by $(A^T A)^{-1} A^T b$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A \qquad \qquad \qquad A^T b$

The summations are over all pixels in the $K \times K$ window

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$A^T A \quad A^T b$$

When is this solvable? i.e., what are good points to track?

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$
$$A^T A \quad A^T b$$

When is this solvable? i.e., what are good points to track?

- $A^T A$ should be invertible

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$
$$A^T A \quad A^T b$$

When is this solvable? i.e., what are good points to track?

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
 - eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$A^T A \quad A^T b$$

When is this solvable? i.e., what are good points to track?

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
 - eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $A^T b$

When is this solvable? i.e., what are good points to track?

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
 - eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Does this remind you of anything?

Conditions for solvability

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$A^T A$ $A^T b$

When is this solvable? i.e., what are good points to track?

- $A^T A$ should be invertible
- $A^T A$ should not be too small due to noise
 - eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Does this remind you of anything?

Criteria for Harris corner detector

Low-texture region



$$\sum \nabla I (\nabla I)^T$$

- gradients have small magnitude
- small λ_1 , small λ_2

Edge



$$\sum \nabla I (\nabla I)^T$$

- gradients very large or very small
- large λ_1 , small λ_2

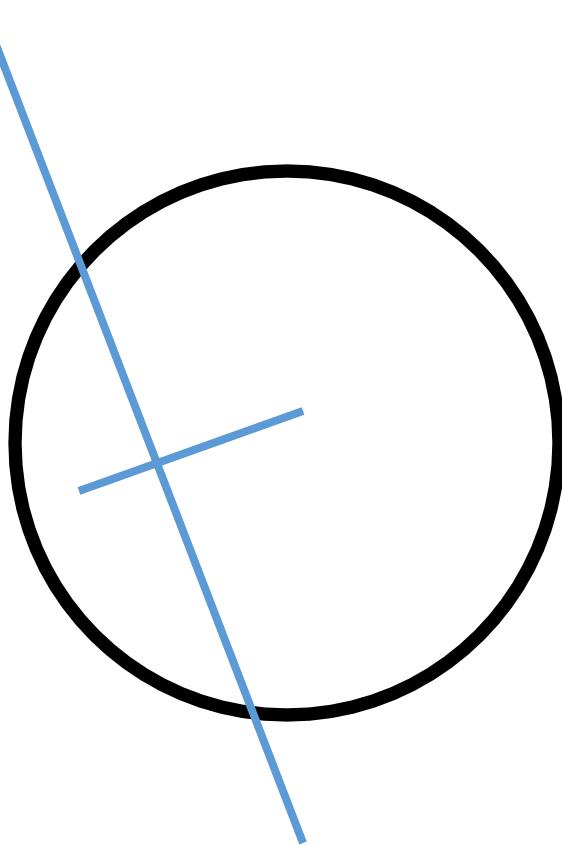
High-texture region



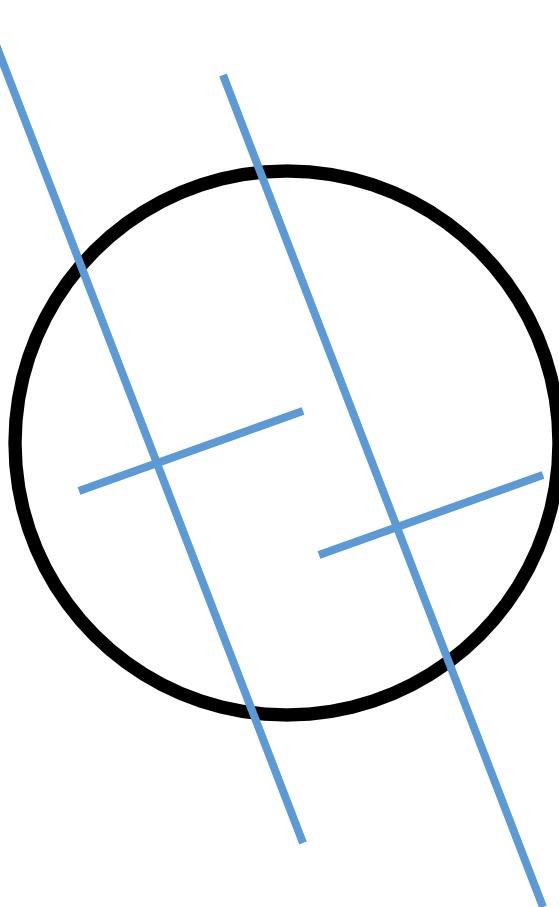
$$\sum \nabla I (\nabla I)^T$$

- gradients are different, large magnitudes
- large λ_1 , large λ_2

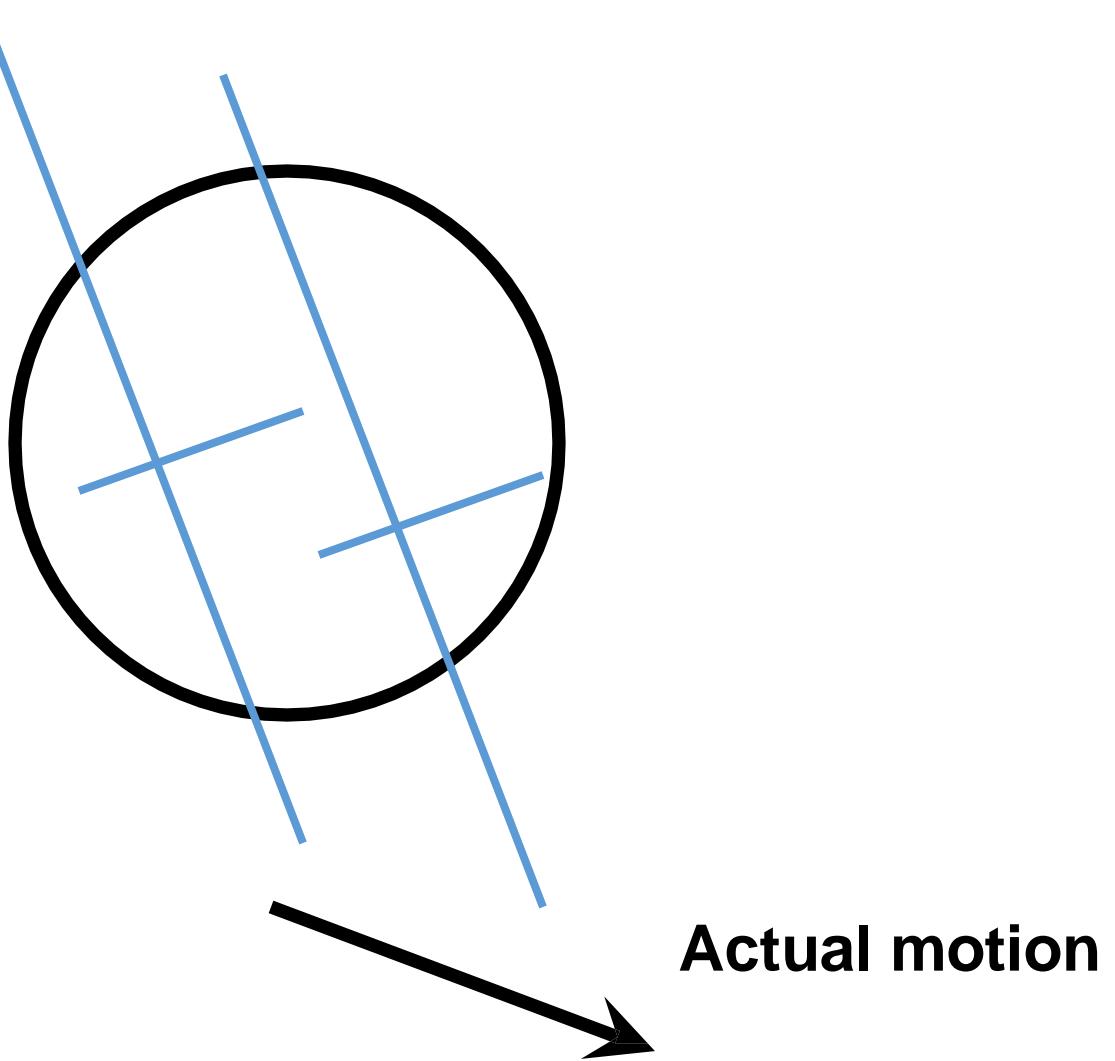
The aperture problem resolved



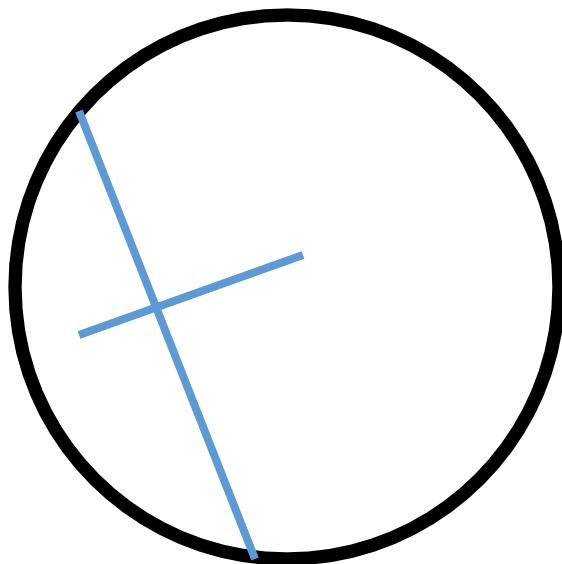
The aperture problem resolved



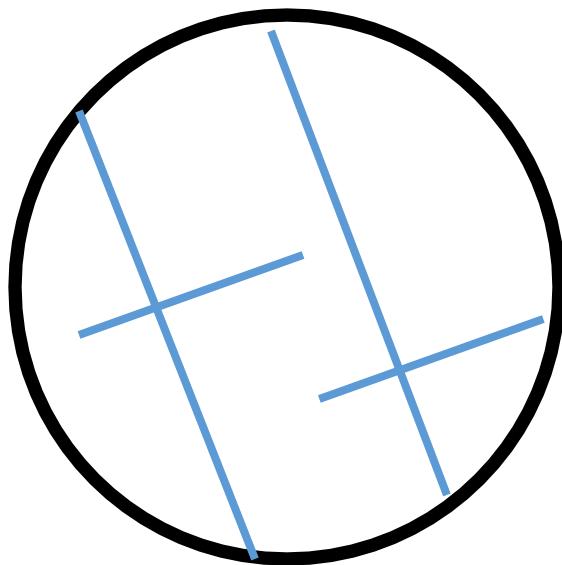
The aperture problem resolved



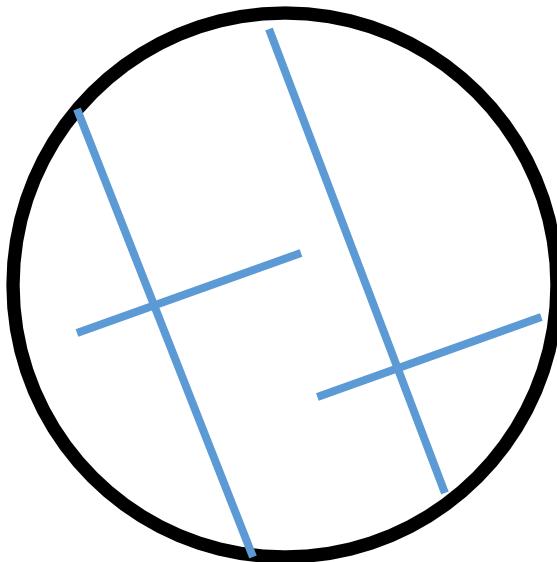
The aperture problem resolved



The aperture problem resolved



The aperture problem resolved



Perceived motion

Dealing with larger movements: Iterative refinement

Original (x,y) position



1. Initialize $(x',y') = (x,y)$

$$I_t = I(x', y', t+1) - I(x, y, t)$$

Dealing with larger movements: Iterative refinement

Original (x,y) position



1. Initialize $(x',y') = (x,y)$

$$I_t = I(x', y', t+1) - I(x, y, t)$$

2. Compute (u,v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature
patch in first image

displacement

Dealing with larger movements: Iterative refinement

1. Initialize $(x', y') = (x, y)$

2. Compute (u, v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature
patch in first image

$$I_t = I(x', y', t+1) - I(x, y, t)$$

displacement

Dealing with larger movements: Iterative refinement

1. Initialize $(x', y') = (x, y)$

2. Compute (u, v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature
patch in first image

displacement

$$I_t = I(x', y', t+1) - I(x, y, t)$$

3. Shift window by (u, v) : $x' = x' + u$; $y' = y' + v$;

Original (x,y) position



Dealing with larger movements: Iterative refinement

1. Initialize $(x', y') = (x, y)$

2. Compute (u, v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature
patch in first image

$$I_t = I(x', y', t+1) - I(x, y, t)$$

displacement

3. Shift window by (u, v) : $x' = x' + u$; $y' = y' + v$;

4. Recalculate I_t

Dealing with larger movements: Iterative refinement

1. Initialize $(x',y') = (x,y)$

$$I_t = I(x', y', t+1) - I(x, y, t)$$

2. Compute (u,v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

2nd moment matrix for feature
patch in first image

displacement

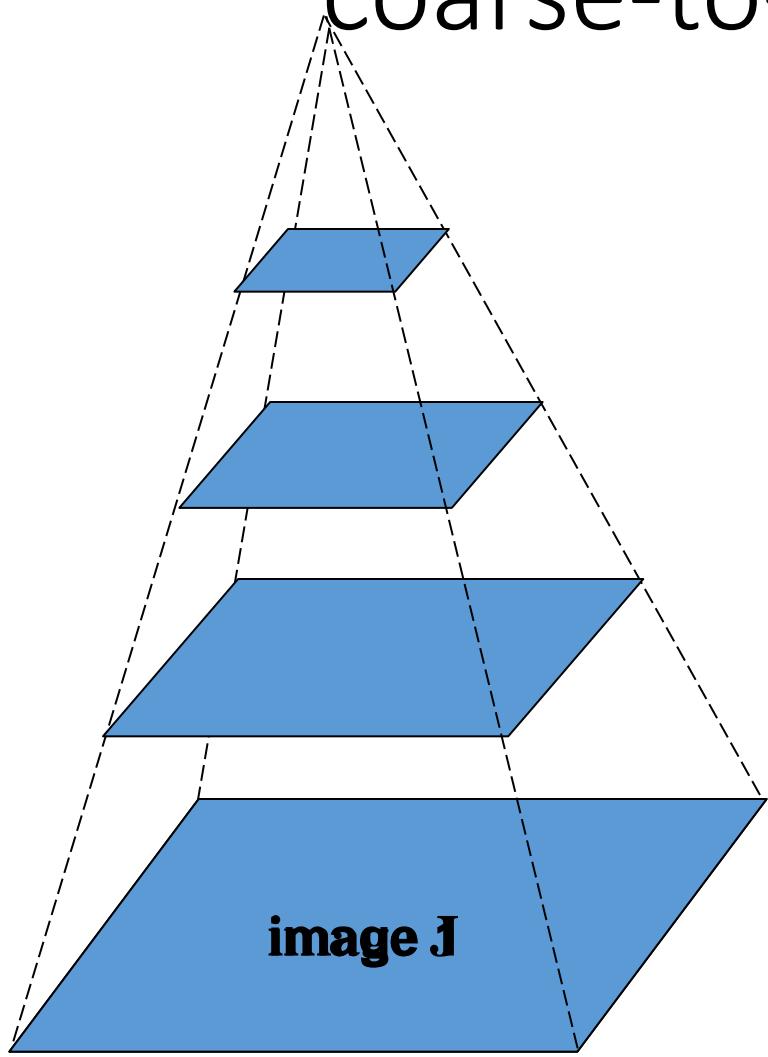
3. Shift window by (u, v) : $x' = x' + u$; $y' = y' + v$;

4. Recalculate I_t

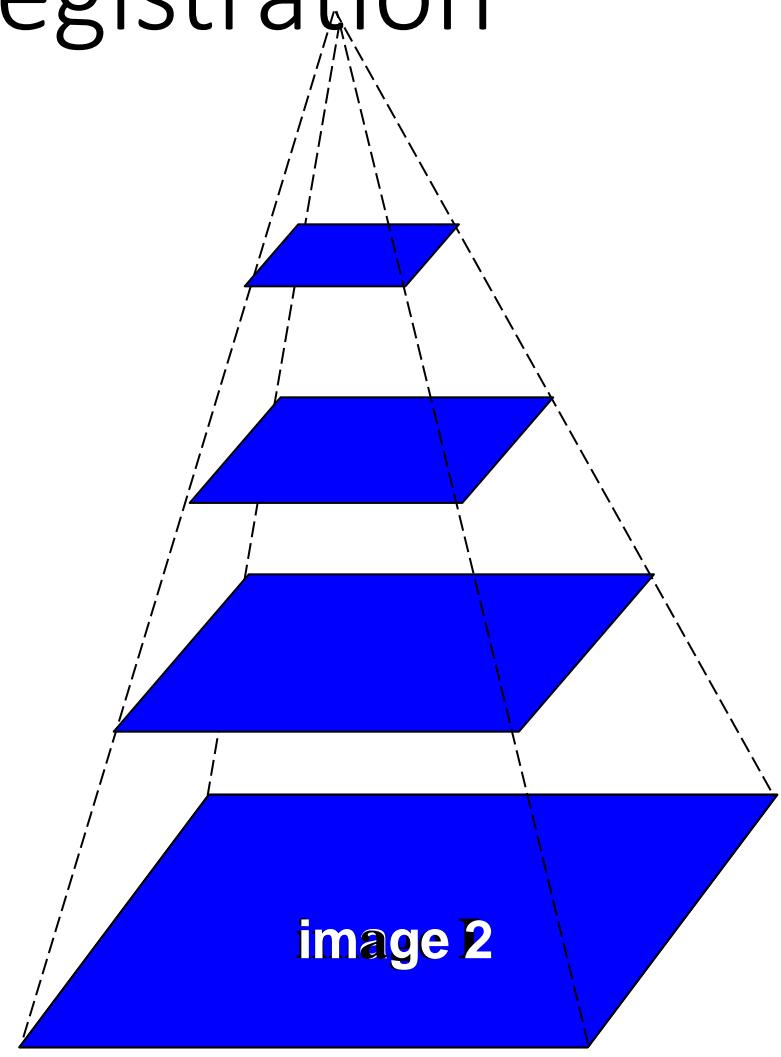
5. Repeat steps 2-4 until small change

- Use interpolation for subpixel values

Dealing with larger movements: coarse-to-fine registration

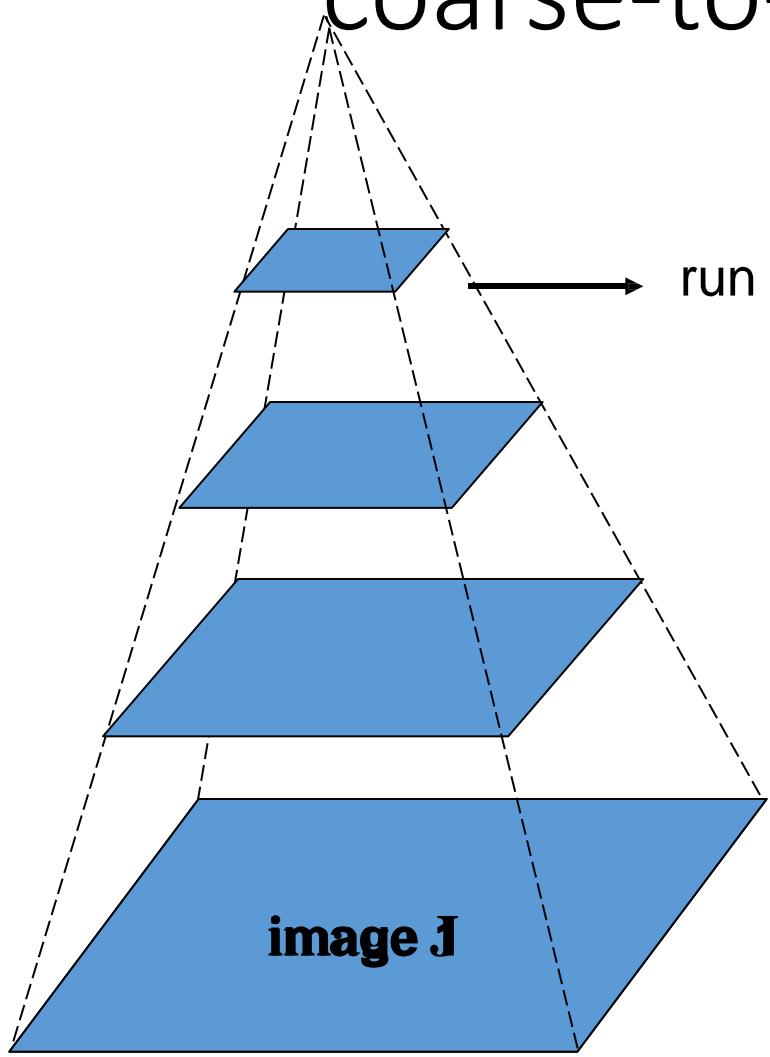


Gaussian pyramid of image 1 (t)

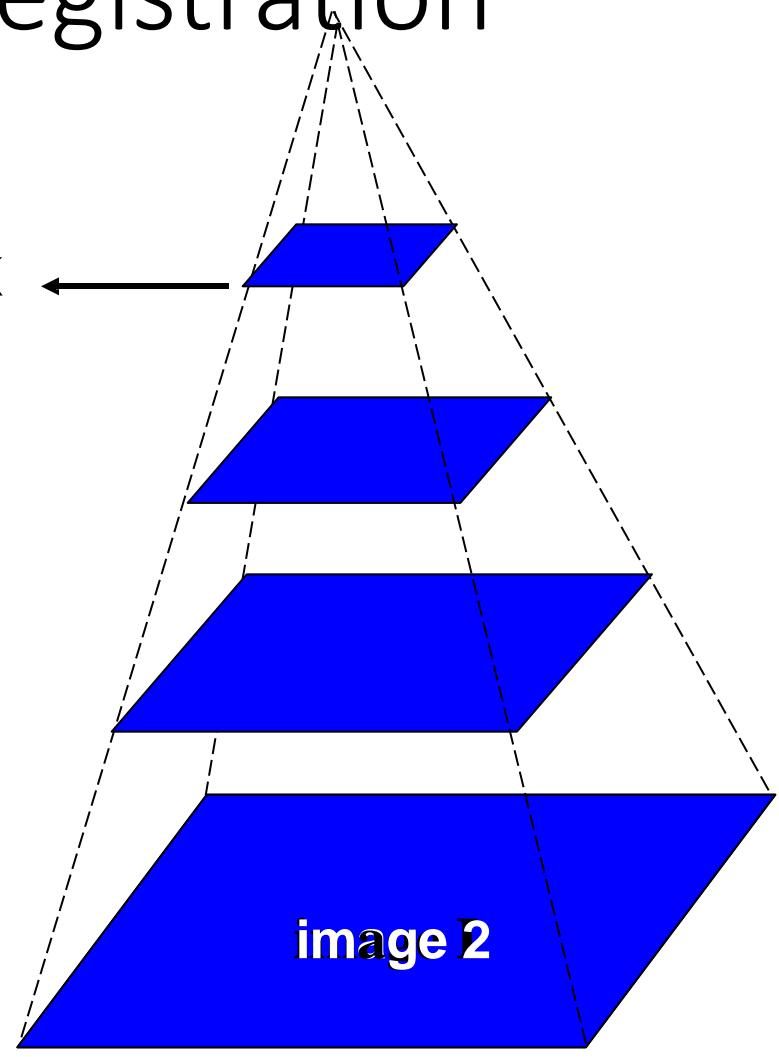


Gaussian pyramid of image 2 (t+1)

Dealing with larger movements: coarse-to-fine registration



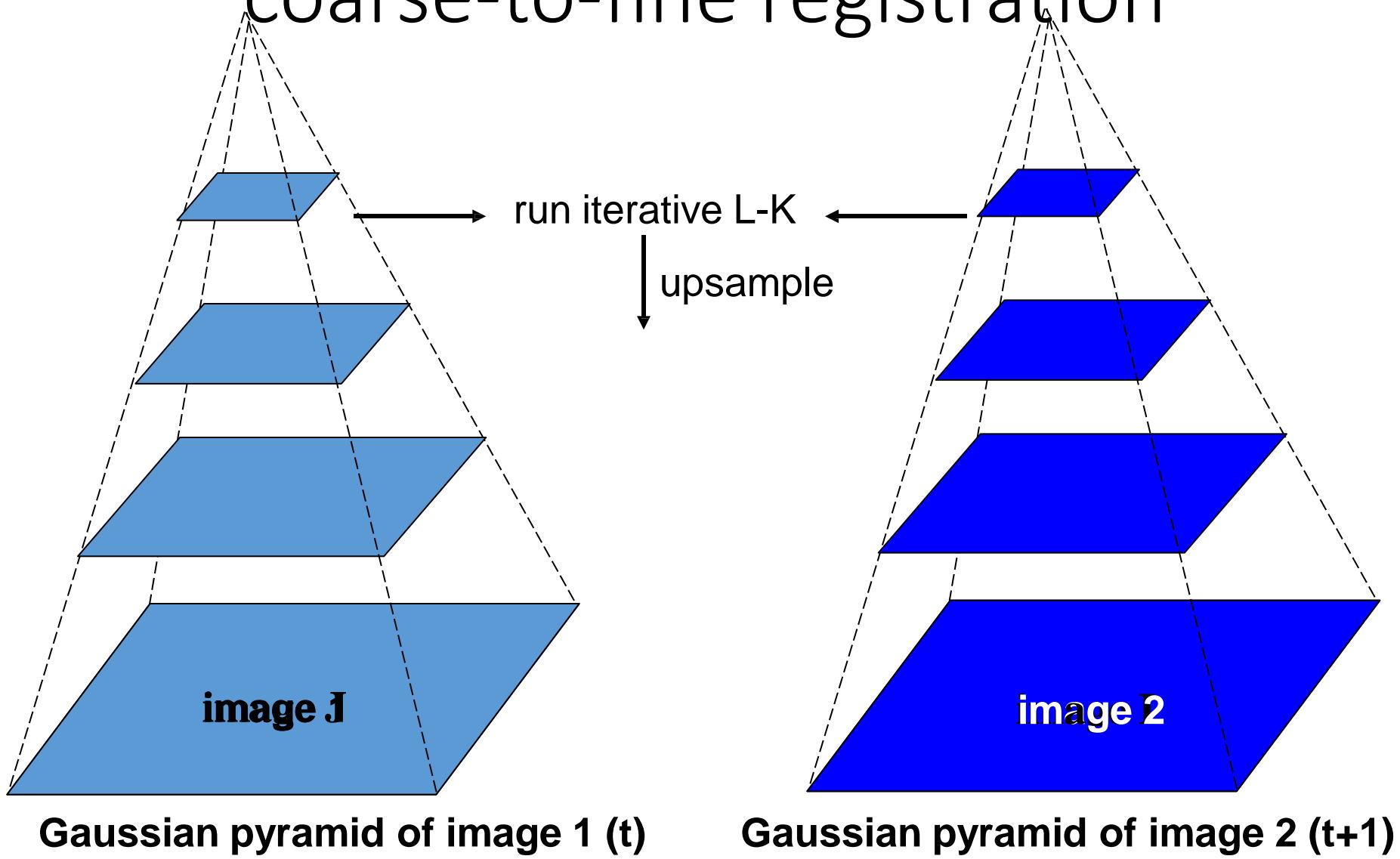
Gaussian pyramid of image 1 (t)



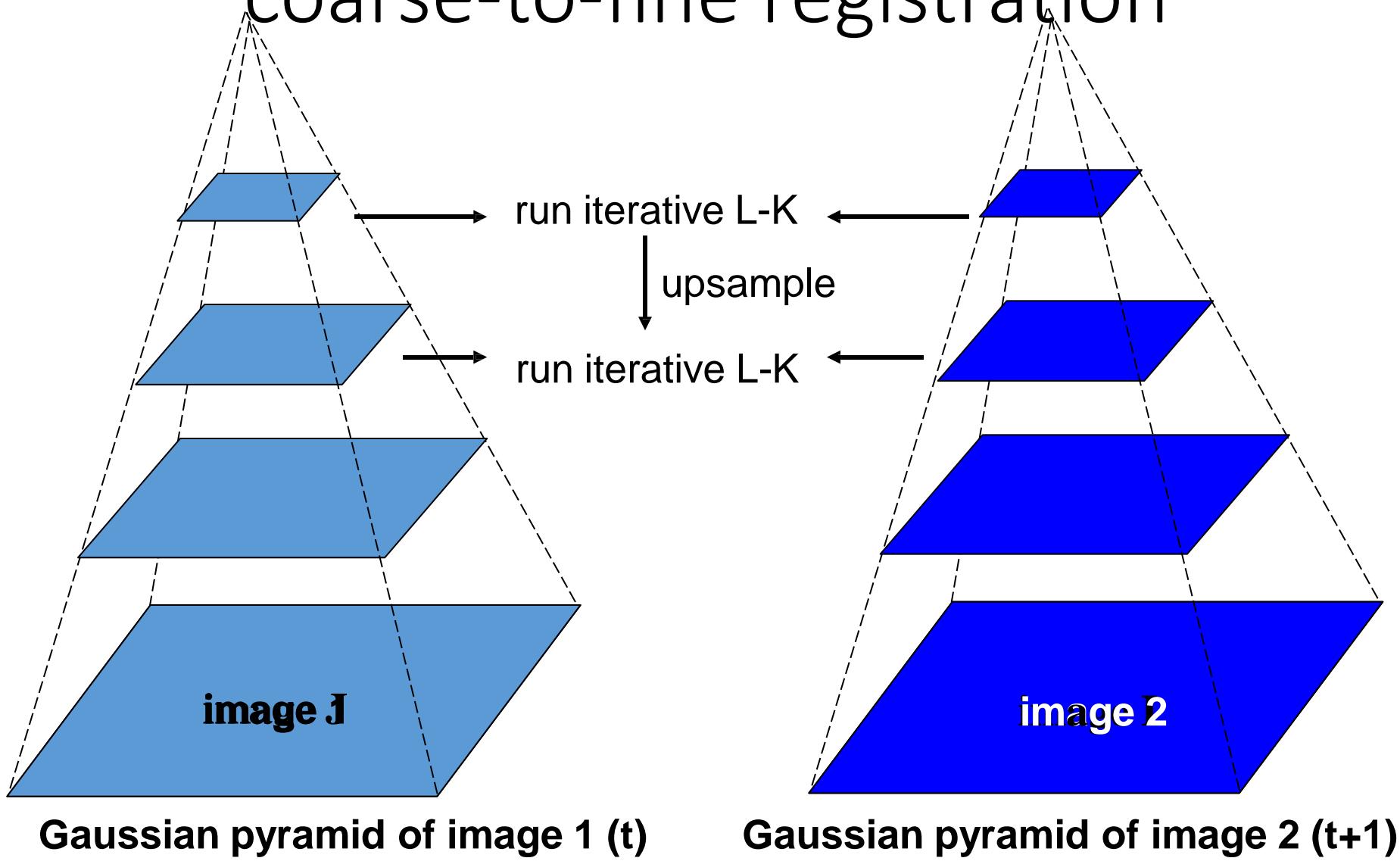
Gaussian pyramid of image 2 (t+1)

run iterative L-K

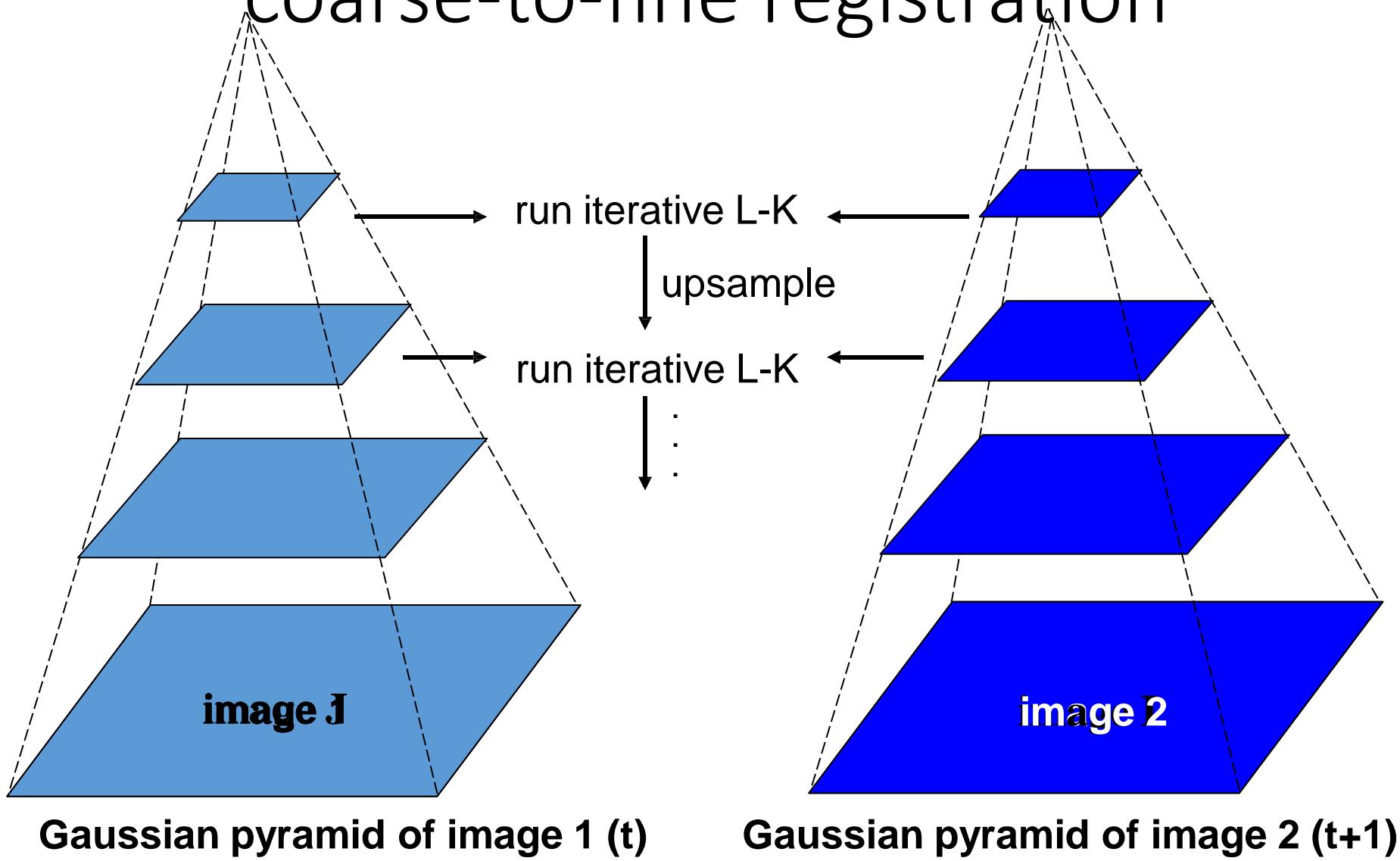
Dealing with larger movements: coarse-to-fine registration



Dealing with larger movements: coarse-to-fine registration



Dealing with larger movements: coarse-to-fine registration

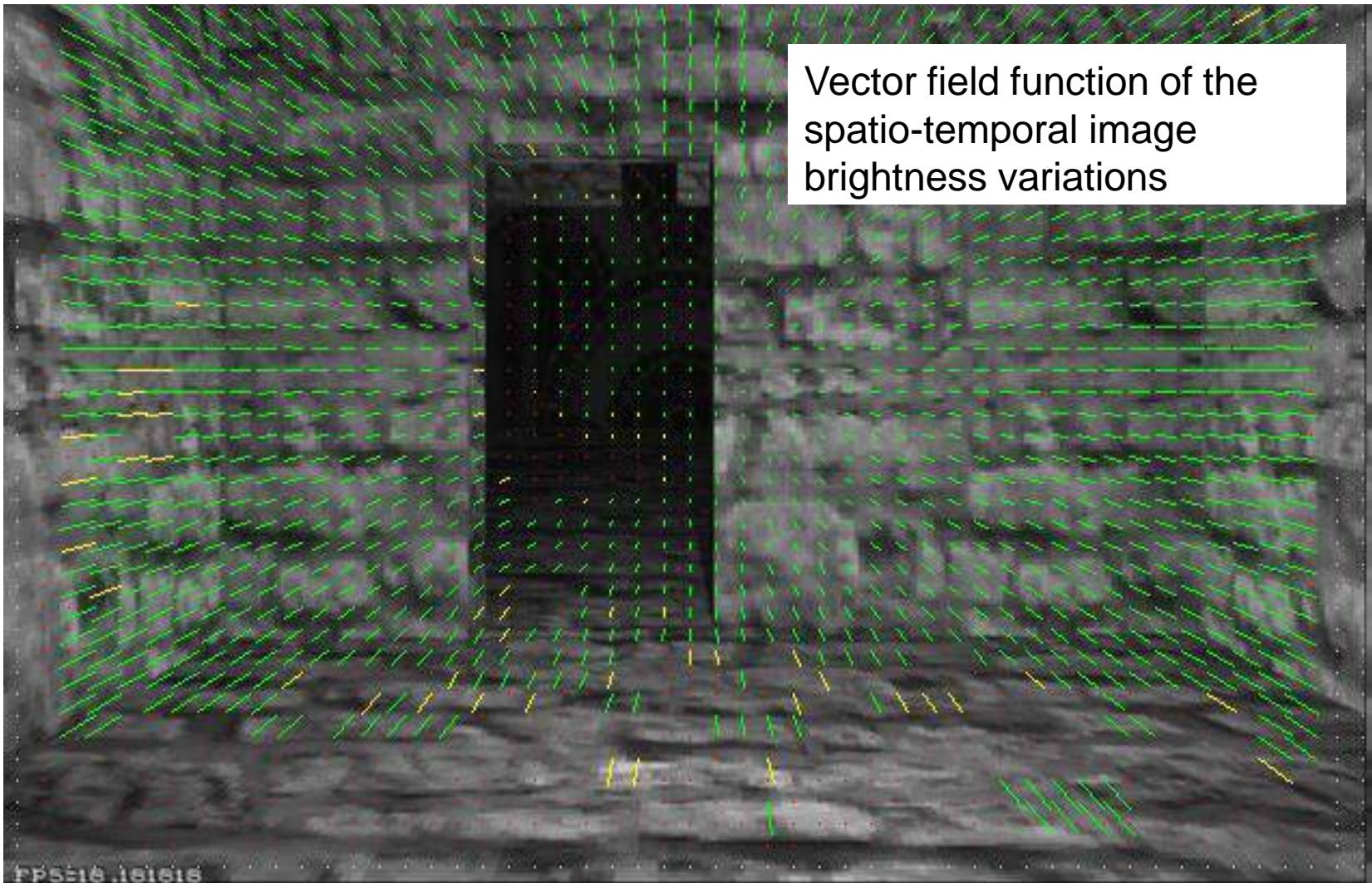


Summary of Lucas & Kanade tracking

- Find a good point to track
- Use intensity second moment matrix and difference across frames to find displacement
- Iterate and use coarse-to-fine search to deal with larger movements
- When creating long tracks, check appearance of registered patch against appearance of initial patch to find points that have drifted

B. Lucas and T. Kanade. [An iterative image registration technique with an application to stereo vision.](#) In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.

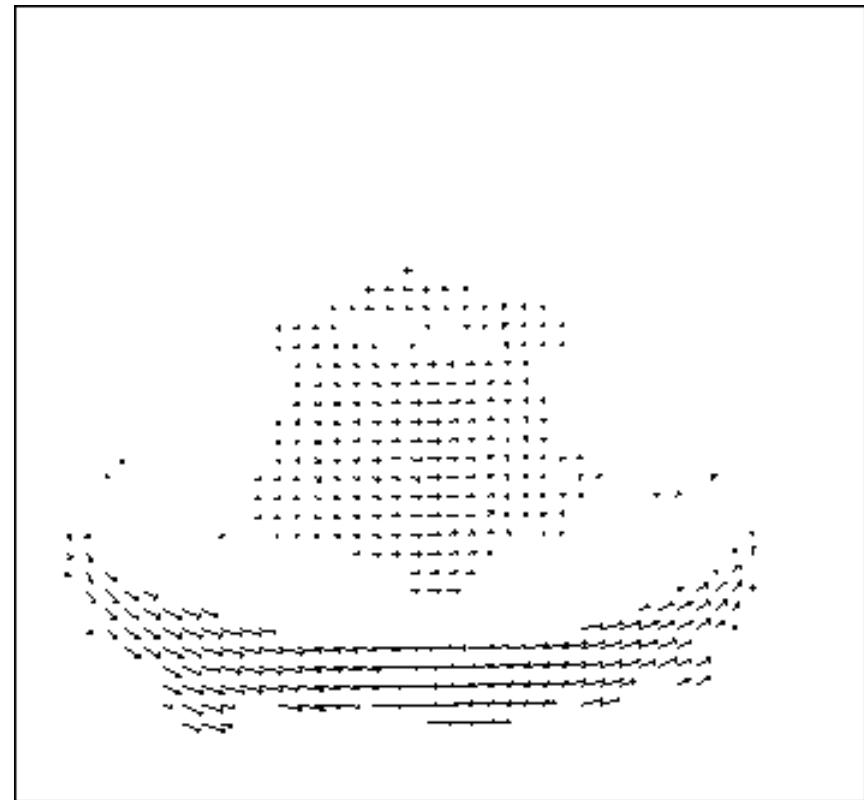
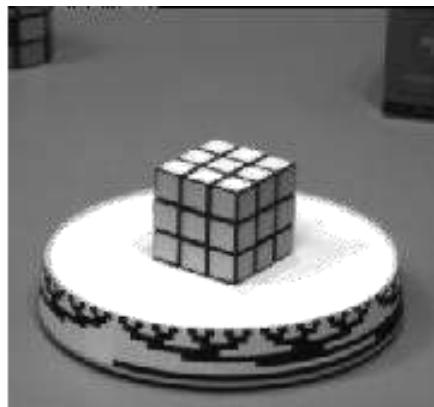
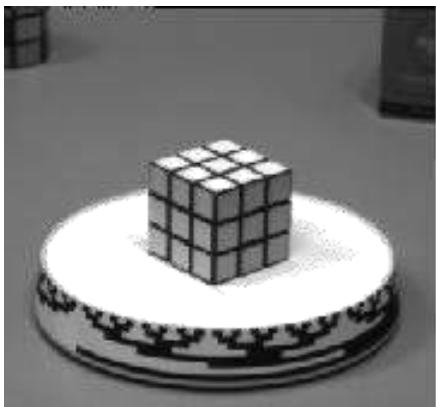
Optical flow



Picture courtesy of Selim Temizer - Learning and Intelligent Systems (LIS) Group, MIT

Motion field

- The motion field is the projection of the 3D scene motion into the image



Optical flow

- Definition: optical flow is the *apparent* motion of brightness patterns in the image

Optical flow

- Definition: optical flow is the *apparent* motion of brightness patterns in the image
- Ideally, optical flow would be the same as the motion field

Optical flow

- Definition: optical flow is the *apparent* motion of brightness patterns in the image
- Ideally, optical flow would be the same as the motion field
- Have to be careful: apparent motion can be caused by lighting changes without any actual motion

Optical flow

- Definition: optical flow is the *apparent* motion of brightness patterns in the image
- Ideally, optical flow would be the same as the motion field
- Have to be careful: apparent motion can be caused by lighting changes without any actual motion
 - Think of a uniform rotating sphere under fixed lighting vs. a stationary sphere under moving illumination

Lucas-Kanade Optical Flow

- Same as Lucas-Kanade feature tracking, but for each pixel
 - As we saw, works better for textured pixels
- Operations can be done one frame at a time, rather than pixel by pixel
 - Efficient

Errors in Lucas-Kanade

- The motion is large

Errors in Lucas-Kanade

- The motion is large
 - Possible Fix: Keypoint matching

Errors in Lucas-Kanade

- The motion is large
 - Possible Fix: Keypoint matching
- A point does not move like its neighbors

Errors in Lucas-Kanade

- The motion is large
 - Possible Fix: Keypoint matching
- A point does not move like its neighbors
 - Possible Fix: Region-based matching

Errors in Lucas-Kanade

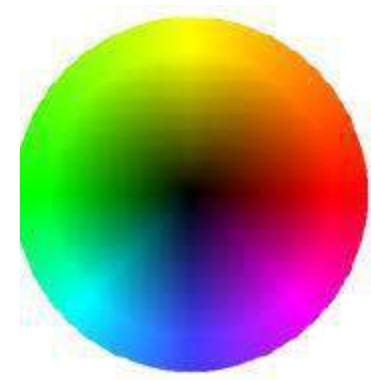
- The motion is large
 - Possible Fix: Keypoint matching
- A point does not move like its neighbors
 - Possible Fix: Region-based matching
- Brightness constancy does not hold

Errors in Lucas-Kanade

- The motion is large
 - Possible Fix: Keypoint matching
- A point does not move like its neighbors
 - Possible Fix: Region-based matching
- Brightness constancy does not hold
 - Possible Fix: Gradient constancy

State-of-the-art optical flow

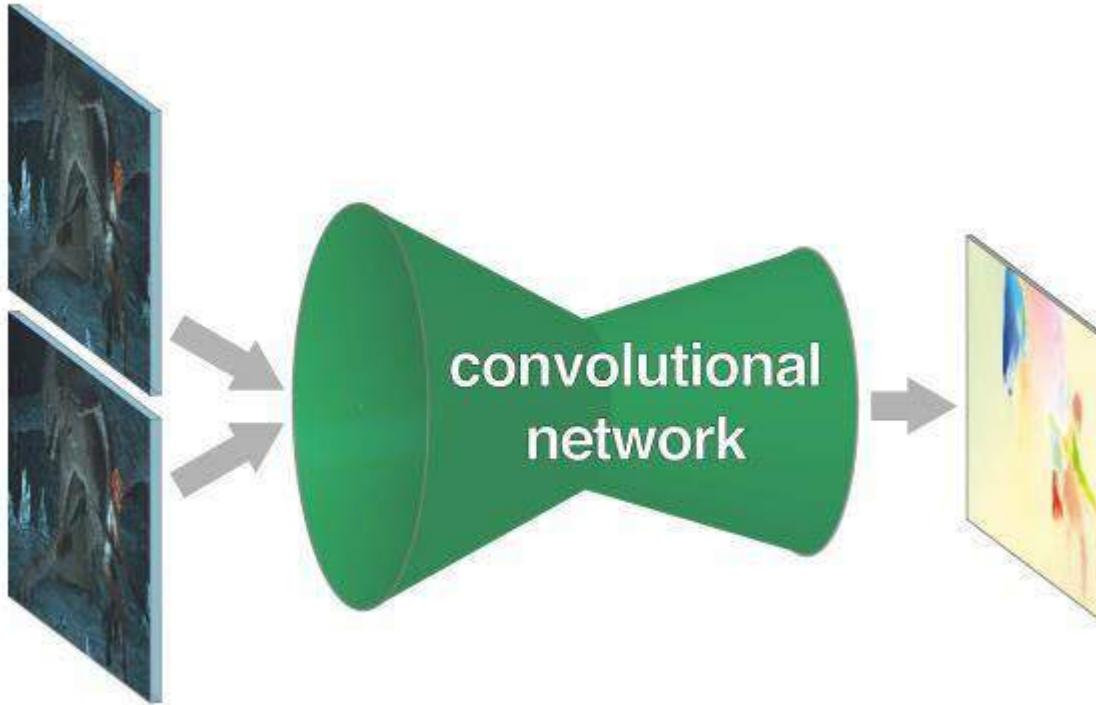
Start with something similar to Lucas-Kanade
+ gradient constancy
+ energy minimization with smoothing term
+ region matching
+ keypoint matching (long-range)



Region-based +Pixel-based +Keypoint-based

[Large displacement optical flow](#), Brox et al., CVPR 2009

Recent Trends



[DeepFlow: Large displacement optical flow with deep matching](#). ICCV 2013

[FlowNet: Learning Optical Flow with Convolutional Networks](#). ICCV 2015

[Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation](#). ICCV 2015

[A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation](#). CVPR 2016

[FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks](#). CVPR 2017

[Optical flow estimation using a spatial pyramid network](#). CVPR 2017

[Unsupervised Deep Learning for Optical Flow Estimation](#). AAAI 2017

[Semi-supervised learning for optical flow with generative adversarial networks](#). NIPS 2017

Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Leibe
 - And many more



Brightness Constancy

16-385 Computer Vision (Kris Kitani)
Carnegie Mellon University

Optical Flow

Problem Definition

Given two consecutive image frames,
estimate the motion of each pixel

Assumptions

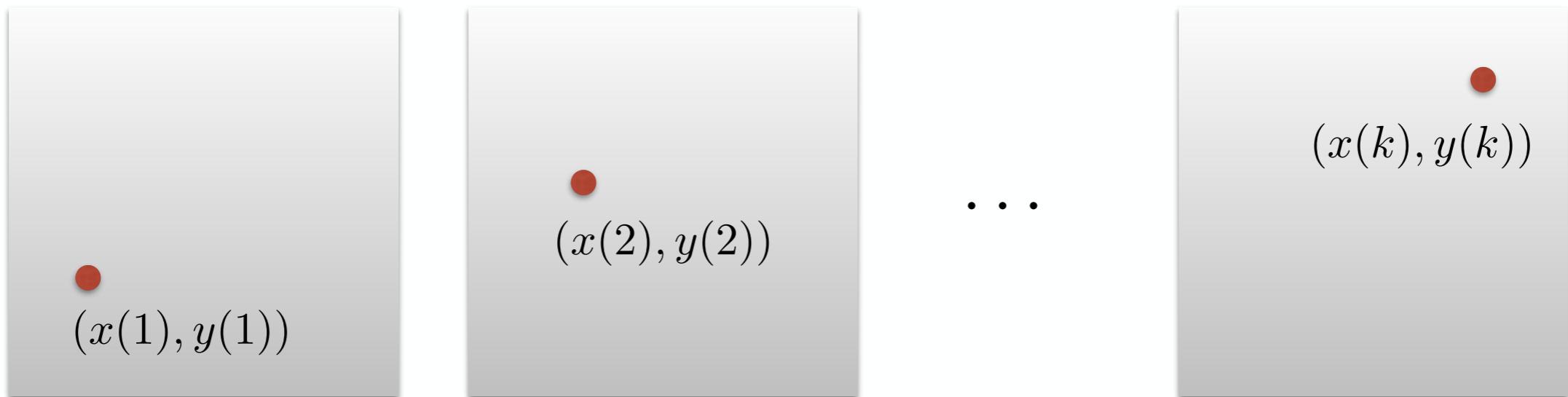
Brightness constancy

Small motion

Assumption 1

Brightness constancy

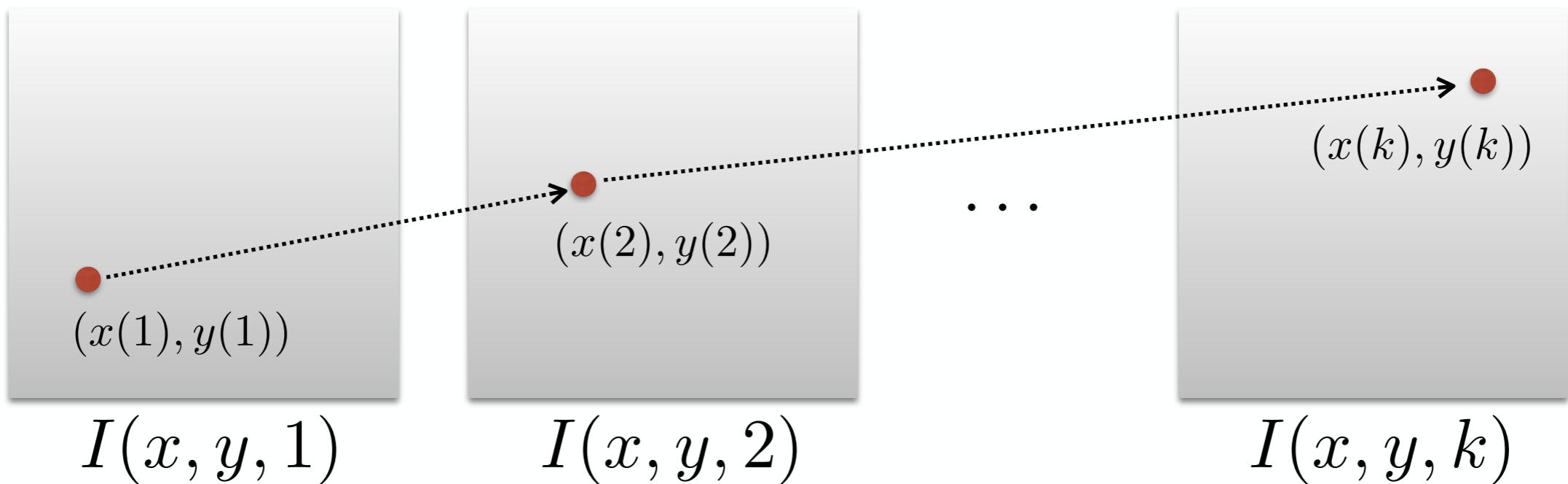
Scene point moving through image sequence



Assumption 1

Brightness constancy

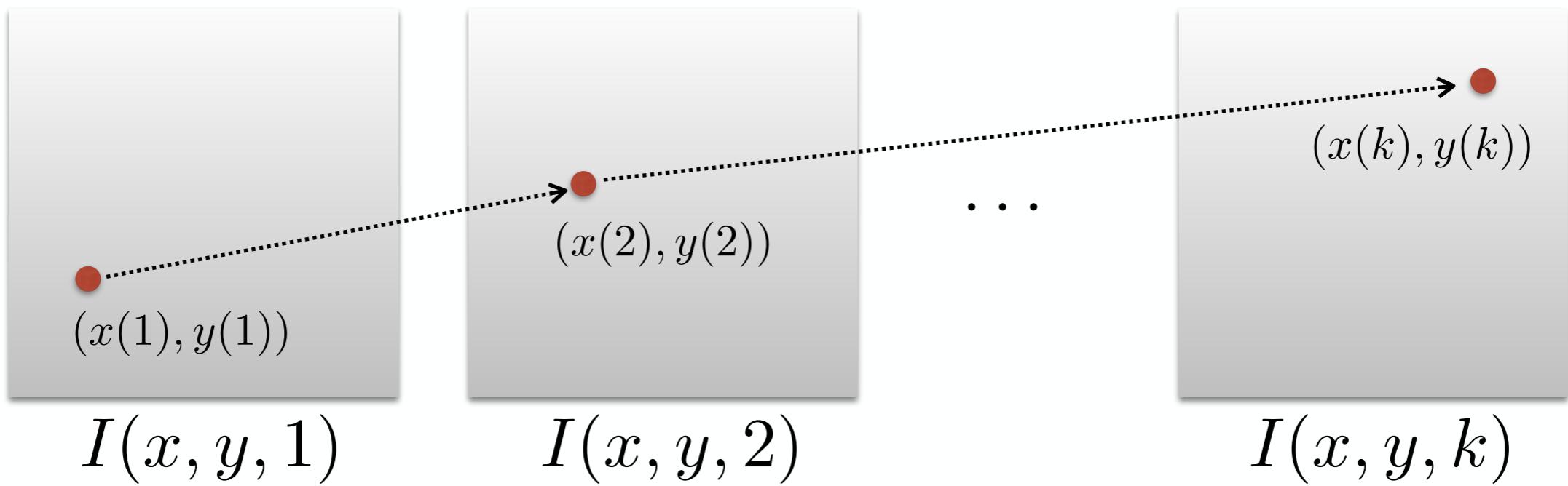
Scene point moving through image sequence



Assumption 1

Brightness constancy

Scene point moving through image sequence

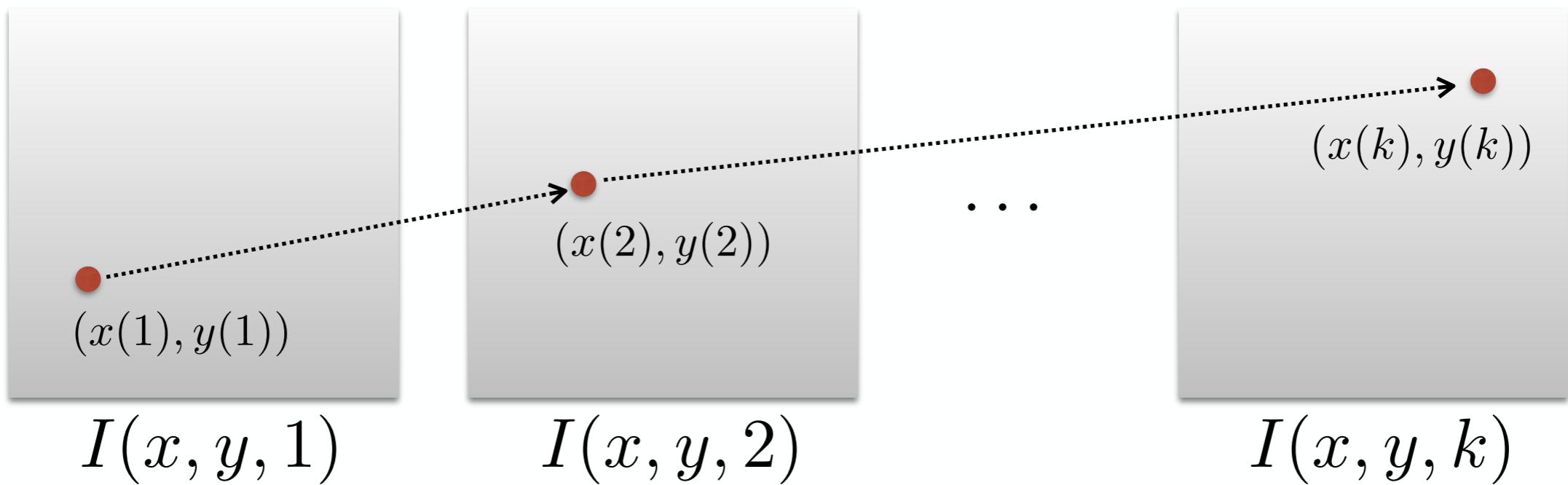


Assumption: Brightness of the point will remain the same

Assumption 1

Brightness constancy

Scene point moving through image sequence



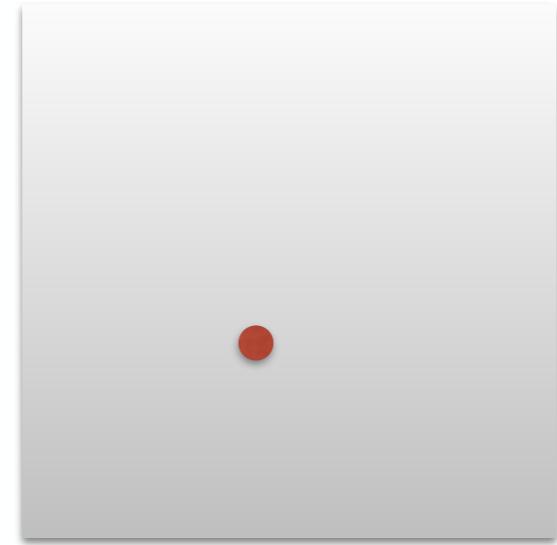
Assumption: Brightness of the point will remain the same

$$I(x(t), y(t), t) = C$$

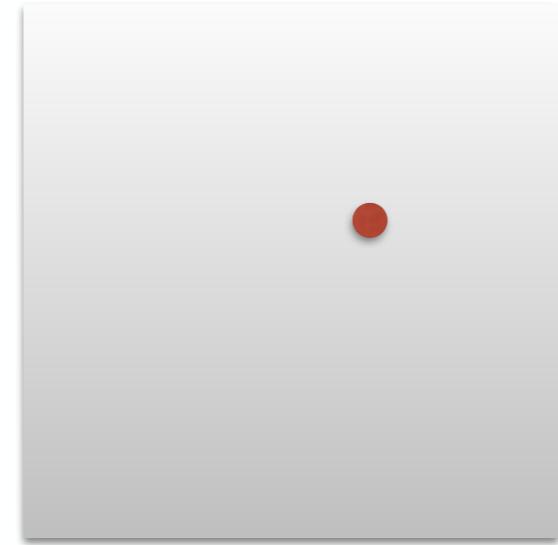
constant

Assumption 2

Small motion



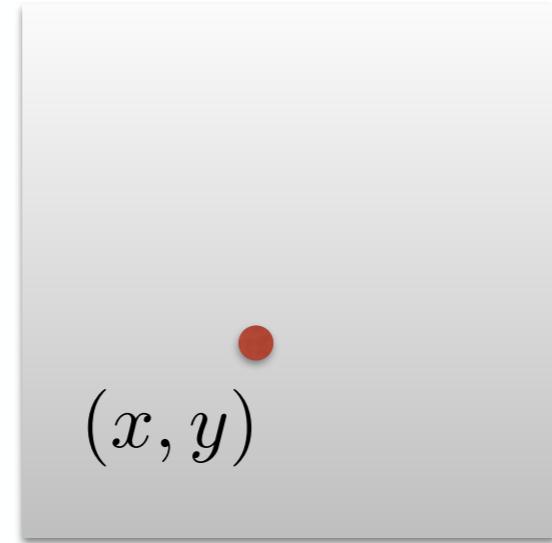
$I(x, y, t)$



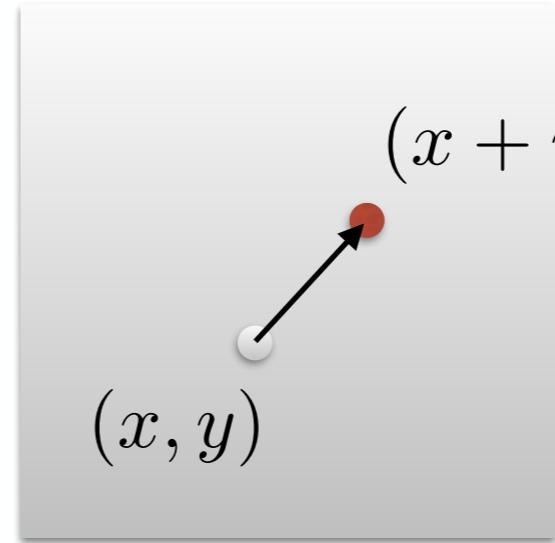
$I(x, y, t + \delta t)$

Assumption 2

Small motion



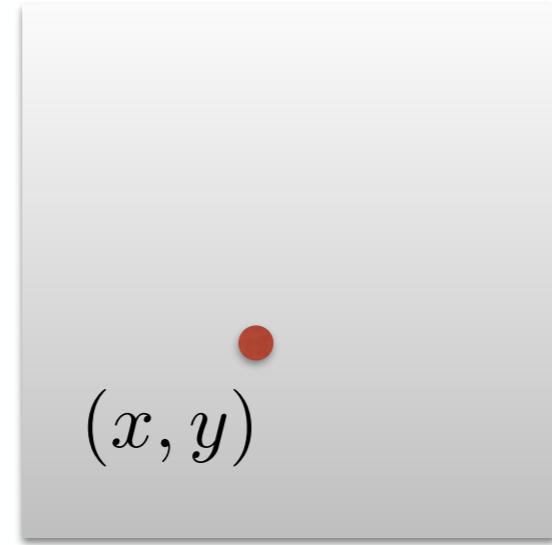
$$I(x, y, t)$$



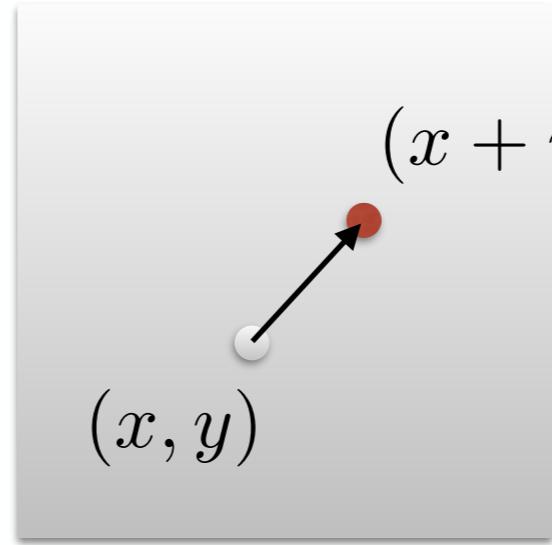
$$I(x, y, t + \delta t)$$

Assumption 2

Small motion



$$I(x, y, t)$$



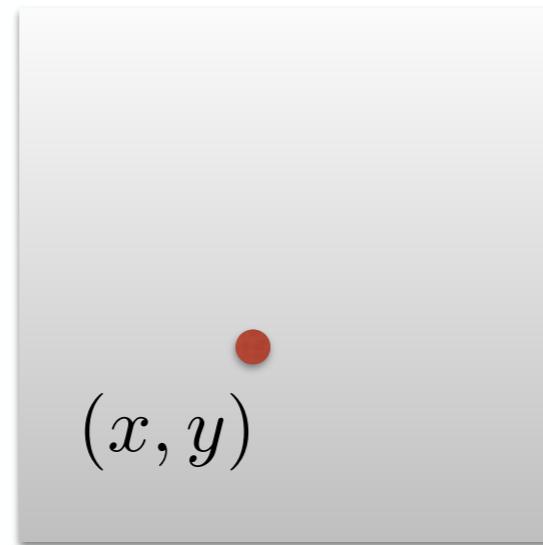
$$I(x, y, t + \delta t)$$

Optical flow (velocities): (u, v)

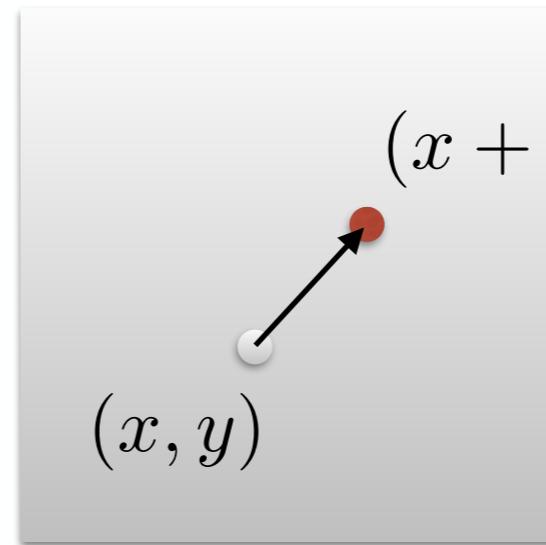
Displacement: $(\delta x, \delta y) = (u\delta t, v\delta t)$

Assumption 2

Small motion



$$I(x, y, t)$$



$$I(x, y, t + \delta t)$$

Optical flow (velocities): (u, v)

Displacement: $(\delta x, \delta y) = (u\delta t, v\delta t)$

For a really small space-time step...

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

... the brightness between two consecutive image frames is the same

These assumptions yield the ...

Brightness Constancy Equation

$$\frac{dI}{dt} = \frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

total derivative

partial derivative

Equality is not obvious. Where does this come from?

These assumptions yield the ...

Brightness Constancy Equation

$$\frac{dI}{dt} = \frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

total derivative

partial derivative

Where does this come from?

proof!

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

For small space-time step, brightness of a point is the same

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

For small space-time step, brightness of a point is the same

Insight:

If the time step is really small,
we can *linearize* the intensity function

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion
(First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion
(First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion (First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

partial derivative

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

cancel terms

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion (First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

$$\frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0 \quad \text{cancel terms}$$

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion (First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

$$\frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0 \quad \begin{array}{l} \text{divide by } \delta t \\ \text{take limit } \delta t \rightarrow 0 \end{array}$$

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion (First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

$$\frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0 \quad \begin{matrix} \text{divide by } \delta t \\ \text{take limit } \delta t \rightarrow 0 \end{matrix}$$

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t)$$

Multivariable Taylor Series Expansion
 (First order approximation, two variables)

$$f(x, y) \approx f(a, b) + f_x(a, b)(x - a) + f_y(a, b)(y - b)$$

$$I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = I(x, y, t) \quad \text{assuming small motion}$$

$$\frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0 \quad \begin{array}{l} \text{divide by } \delta t \\ \text{take limit } \delta t \rightarrow 0 \end{array}$$

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

**Brightness Constancy
Equation**

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

**Brightness
Constancy Equation**

$$I_x u + I_y v + I_t = 0$$

(x-flow) (y-flow)

shorthand notation

$$\nabla I^\top \mathbf{v} + I_t = 0$$

(1 × 2) (2 × 1)

vector form

(putting the math aside for a second...)

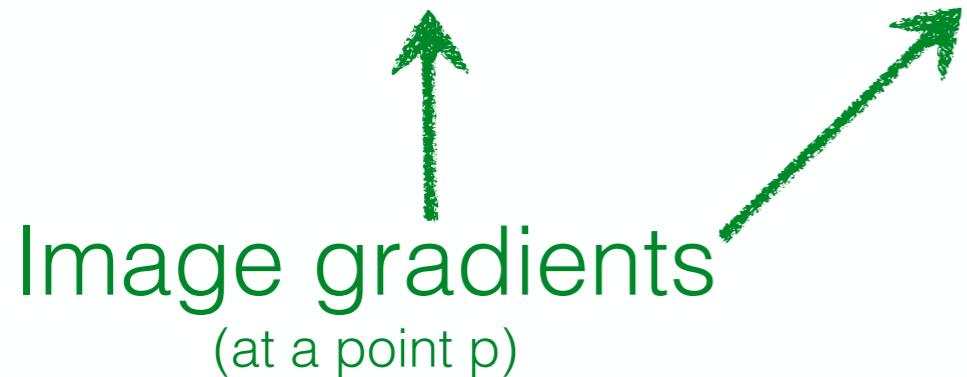
What do the term of the
brightness constancy equation represent?

$$I_x u + I_y v + I_t = 0$$

(putting the math aside for a second...)

What do the term of the
brightness constancy equation represent?

$$I_x u + I_y v + I_t = 0$$



(putting the math aside for a second...)

What do the term of the
brightness constancy equation represent?

flow velocities

$$I_x u + I_y v + I_t = 0$$

Image gradients
(at a point p)

The diagram illustrates the components of the brightness constancy equation. It shows a green arrow labeled "Image gradients (at a point p)" pointing from the origin to a point. Two blue arrows labeled "flow velocities" point downwards from the terms $I_x u$ and $I_y v$ in the equation, representing the spatial derivatives of the image intensity.

(putting the math aside for a second...)

What do the term of the
brightness constancy equation represent?

$$I_x u + I_y v + I_t = 0$$

flow velocities

Image gradients
(at a point p)

temporal gradient

flow velocities

How do you compute these terms?

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

Forward difference

Sobel filter

Scharr filter

...

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference

Sobel filter

Scharr filter

...

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

Forward difference

Sobel filter

Scharr filter

...

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

frame differencing

Frame differencing

t

1	1	1	1	1
1	1	1	1	1
1	10	10	10	10
1	10	10	10	10
1	10	10	10	10

-

$t + 1$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	10	10	10
1	1	10	10	10

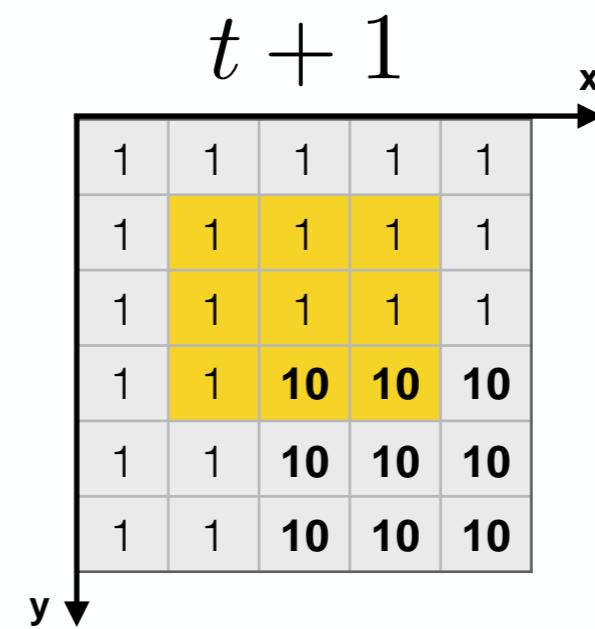
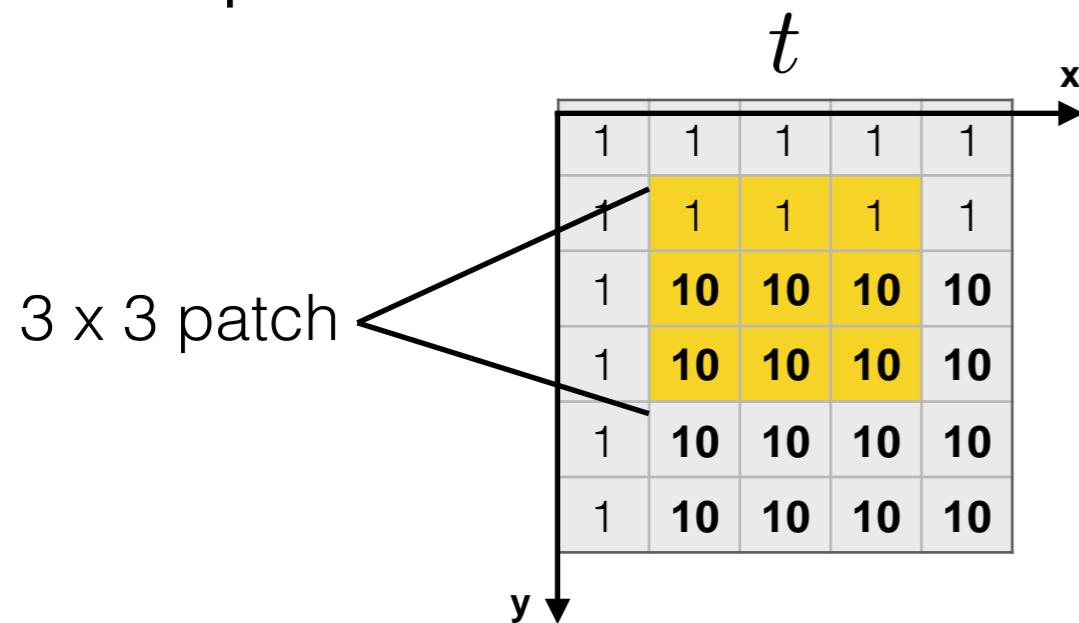
=

0	0	0	0	0
0	0	0	0	0
0	9	9	9	9
0	9	0	0	0
0	9	0	0	0

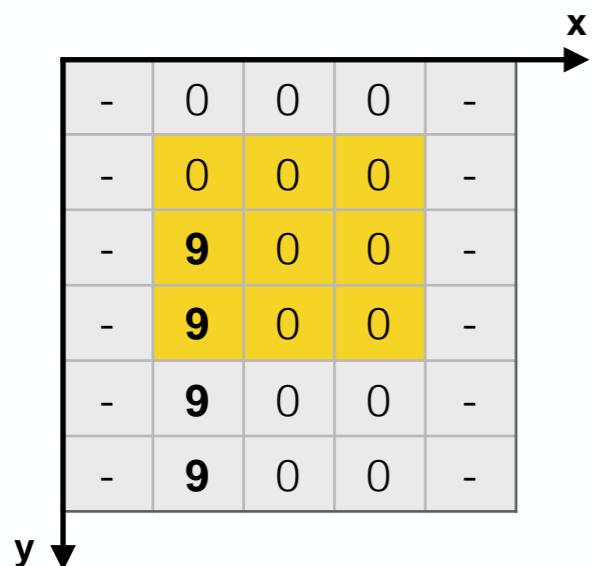
(example of a forward difference)

$$I_t = \frac{\partial I}{\partial t}$$

Example:

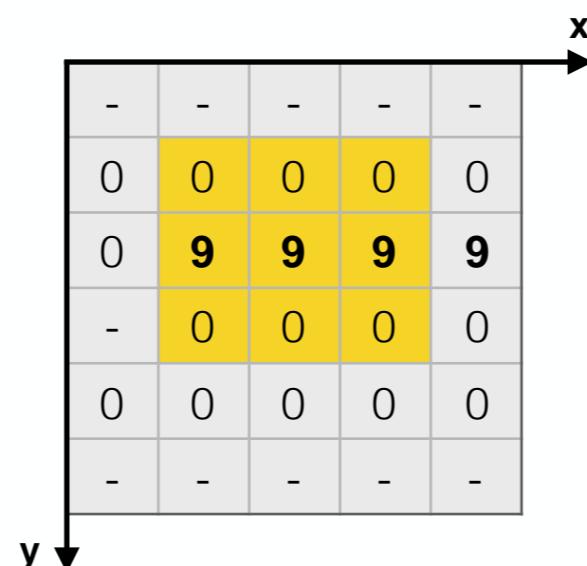


$$I_x = \frac{\partial I}{\partial x}$$



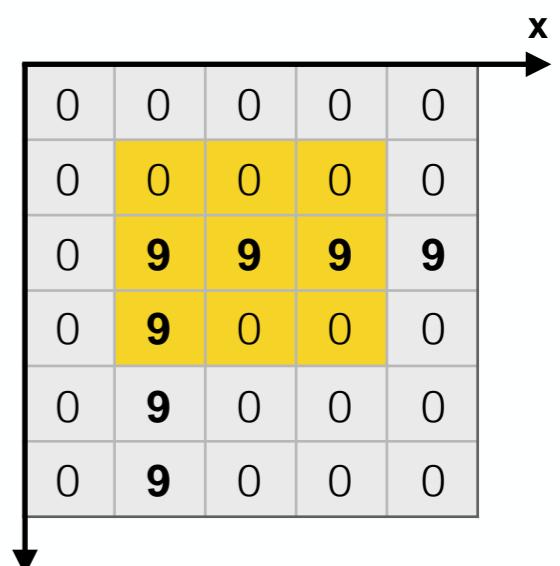
-1 0 1

$$I_y = \frac{\partial I}{\partial y}$$



-1
0
1

$$I_t = \frac{\partial I}{\partial t}$$



$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference

Sobel filter

Scharr filter

...

How do you compute this?

frame differencing

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference

Sobel filter

Scharr filter

...

We need to solve for this!

(this is the unknown in the
optical flow problem)

frame differencing

$$I_x u + I_y v + I_t = 0$$

How do you compute ...

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

Forward difference

Sobel filter

Scharr filter

...

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

(u, v)

Solution lies on a line

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

frame differencing

Cannot be found uniquely
with a single constraint

unknown

$$I_x u + I_y v + I_t = 0$$

The diagram illustrates a linear equation $I_x u + I_y v + I_t = 0$. The terms $I_x u$ and $I_y v$ are circled in green and labeled "unknown". The term I_t is labeled "known". Three arrows point from the "known" label to the term I_t .

known

We need at least _____ equations to solve for 2 unknowns.

unknown

$$I_x u + I_y v + I_t = 0$$

The diagram illustrates a vector equation $I_x u + I_y v + I_t = 0$. The terms $I_x u$ and $I_y v$ are circled in green and labeled "unknown". The term I_t is labeled "known". Three black arrows point from the right side of the equation towards the terms $I_x u$, $I_y v$, and I_t .

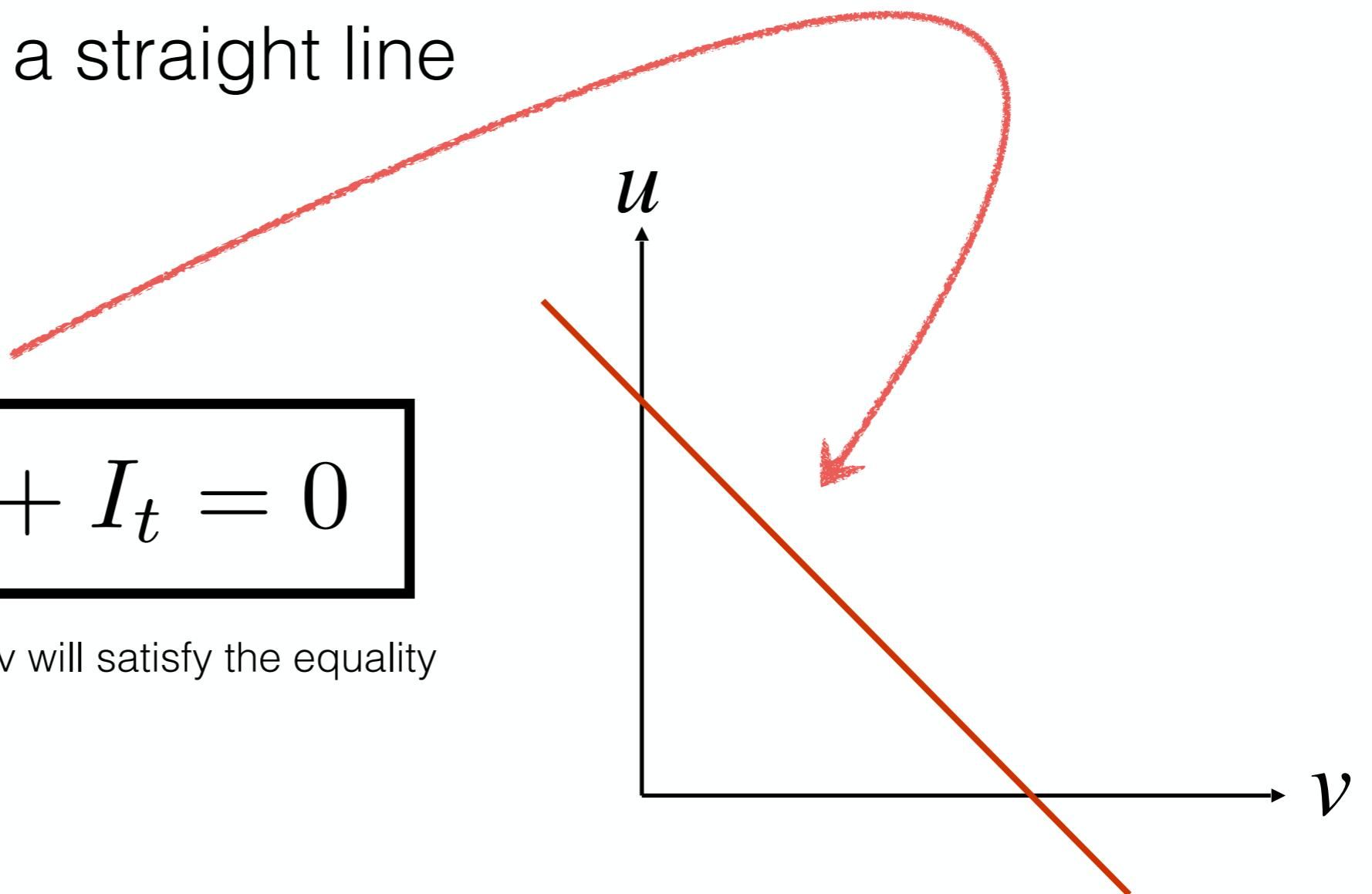
known

Where do we get more equations (constraints)?

Solution lies on a straight line

$$I_x u + I_y v + I_t = 0$$

many combinations of u and v will satisfy the equality



The solution cannot be determined uniquely with a single constraint (a single pixel)

$$I_x u + I_y v + I_t = 0$$

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

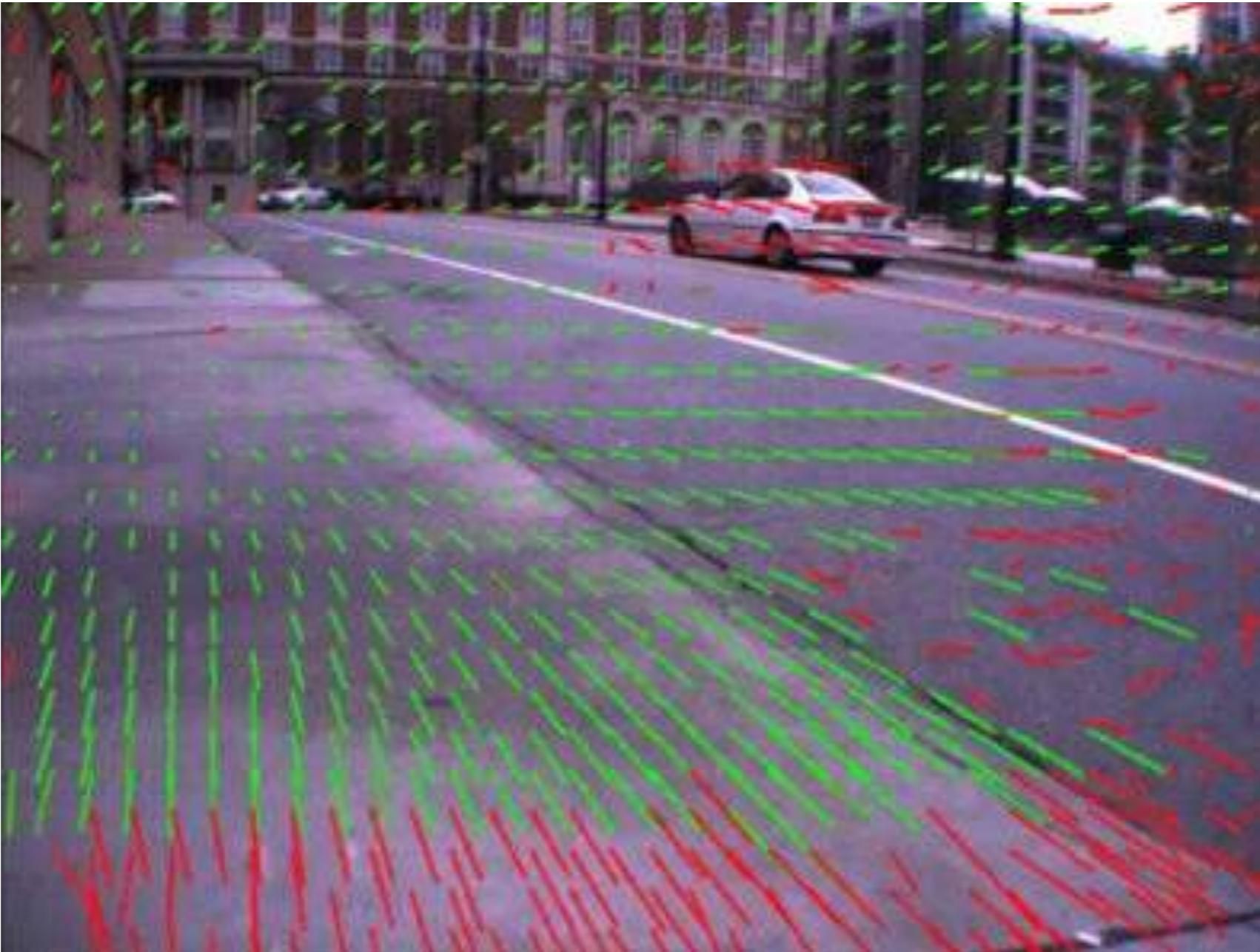
$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

How can we use the brightness constancy equation to estimate the optical flow?



Optical Flow: Constant Flow

Computer Vision 16-385
Carnegie Mellon University (Kris Kitani)

Optical Flow

(a.k.a., Video Stabilization, Tracking, Stereo Matching, Registration)

Given a pair of images

$$\{I_t, I_{t+1}\}$$

Estimate the optical flow field

$$\{v(p_i), u(p_i)\}$$

$$I_x u + I_y v + I_t = 0$$

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

How can we use the brightness constancy equation to estimate the optical flow?

unknown

$$I_x u + I_y v + I_t = 0$$

The diagram shows a central node with three arrows. Two arrows point away from the node, labeled "known" at their tails. One arrow points towards the node, labeled "unknown" at its head.

known

We need at least _____ equations to solve for 2 unknowns.

unknown

$$I_x u + I_y v + I_t = 0$$

The diagram illustrates a vector equation $I_x u + I_y v + I_t = 0$. The terms $I_x u$ and $I_y v$ are circled in green and labeled "unknown". The term I_t is labeled "known". Three black arrows point from the right side of the equation towards the terms $I_x u$, $I_y v$, and I_t .

known

Where do we get more equations (constraints)?

Where do we get more equations (constraints)?

$$I_x u + I_y v + I_t = 0$$

Assume that the surrounding patch (say 5x5) has
'constant flow'

Assumptions:

Flow is locally smooth

Neighboring pixels have same displacement

Using a 5×5 image patch, gives us  equations

Assumptions:

Flow is locally smooth

Neighboring pixels have same displacement

Using a 5×5 image patch, gives us 25 equations

$$I_x(\mathbf{p}_1)u + I_y(\mathbf{p}_1)v = -I_t(\mathbf{p}_1)$$

$$I_x(\mathbf{p}_2)u + I_y(\mathbf{p}_2)v = -I_t(\mathbf{p}_2)$$

⋮

$$I_x(\mathbf{p}_{25})u + I_y(\mathbf{p}_{25})v = -I_t(\mathbf{p}_{25})$$

Assumptions:

Flow is locally smooth

Neighboring pixels have same displacement

Using a 5×5 image patch, gives us 25 equations

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

Matrix form

Assumptions:

Flow is locally smooth

Neighboring pixels have same displacement

Using a 5×5 image patch, gives us 25 equations

$$\begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \vdots & \vdots \\ I_x(\mathbf{p}_{25}) & I_y(\mathbf{p}_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \vdots \\ I_t(\mathbf{p}_{25}) \end{bmatrix}$$

$$A_{25 \times 2}$$

$$x_{2 \times 1}$$

$$b_{25 \times 1}$$

How many equations? How many unknowns? How do we solve this?

Least squares approximation

$\hat{x} = \arg \min_x \|Ax - b\|^2$ is equivalent to solving $A^\top A \hat{x} = A^\top b$

Least squares approximation

$\hat{x} = \arg \min_x \|Ax - b\|^2$ is equivalent to solving $A^\top A \hat{x} = A^\top b$

To obtain the least squares solution solve:

$$\begin{matrix} A^\top A & \hat{x} & A^\top b \\ \left[\begin{array}{cc} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{array} \right] & \left[\begin{array}{c} u \\ v \end{array} \right] & = - \left[\begin{array}{c} \sum_{p \in P} I_x I_t \\ \sum_{p \in P} I_y I_t \end{array} \right] \end{matrix}$$

where the summation is over each pixel p in patch P

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

Least squares approximation

$\hat{x} = \arg \min_x \|Ax - b\|^2$ is equivalent to solving $A^\top A \hat{x} = A^\top b$

To obtain the least squares solution solve:

$$\begin{matrix} A^\top A & \hat{x} & A^\top b \\ \left[\begin{array}{cc} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{array} \right] & \left[\begin{array}{c} u \\ v \end{array} \right] & = - \left[\begin{array}{c} \sum_{p \in P} I_x I_t \\ \sum_{p \in P} I_y I_t \end{array} \right] \end{matrix}$$

where the summation is over each pixel p in patch P

Sometimes called ‘Lucas-Kanade Optical Flow’

(can be interpreted to be a special case of the LK method with a translational warp model)

When is this solvable?

$$A^\top A \hat{x} = A^\top b$$

$A^\top A$ should be invertible

$A^\top A$ should not be too small

λ_1 and λ_2 should not be too small

$A^\top A$ should be well conditioned

λ_1/λ_2 should not be too large (λ_1 =larger eigenvalue)

Where have you seen this before?

$$A^\top A = \begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

Where have you seen this before?

$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

Harris Corner Detector!

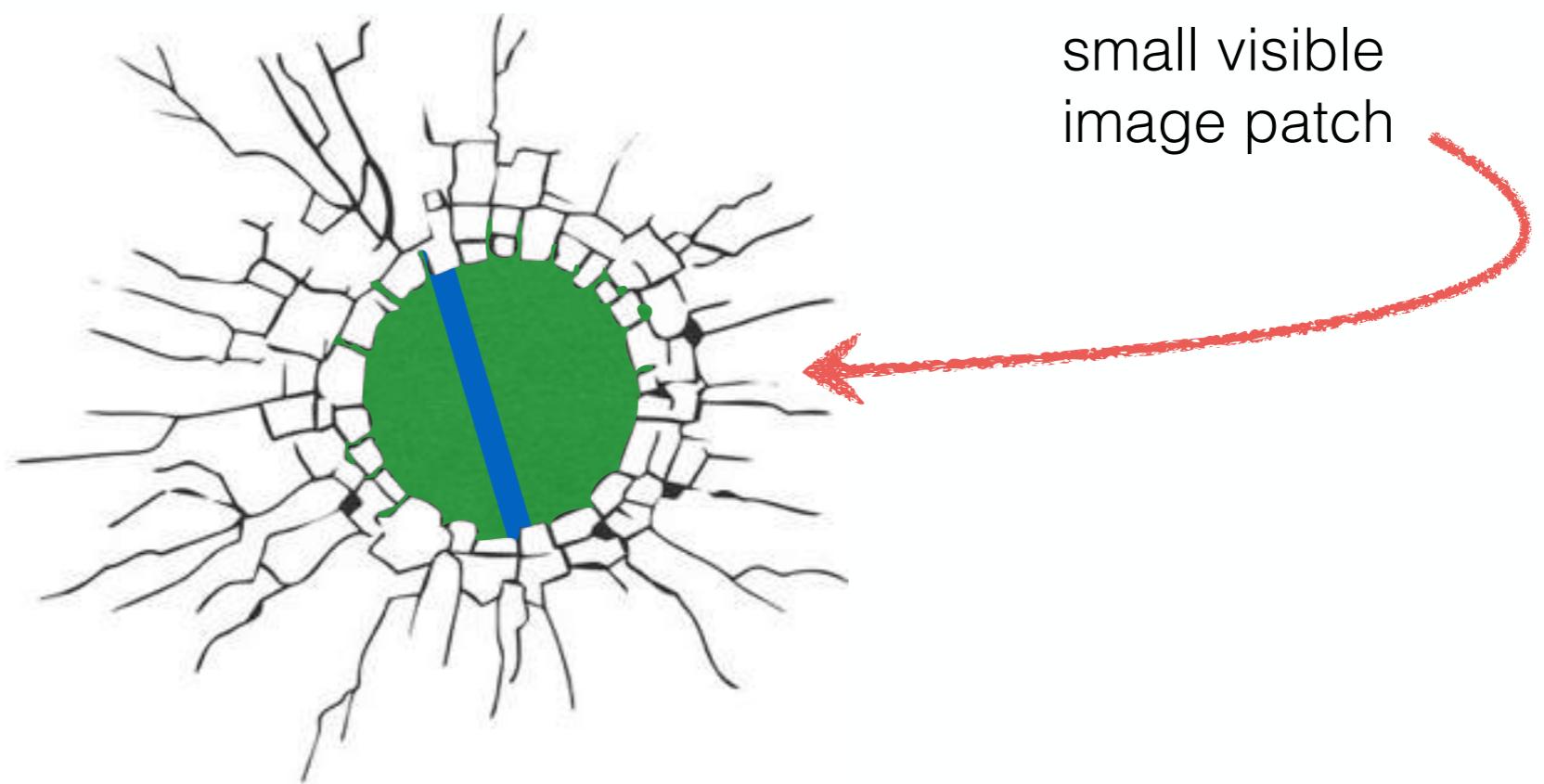
Implications

- Corners are when λ_1, λ_2 are big; this is also when Lucas-Kanade optical flow works best
- Corners are regions with two different directions of gradient (at least)
- Corners are good places to compute flow!

What happens when you have no ‘corners’?

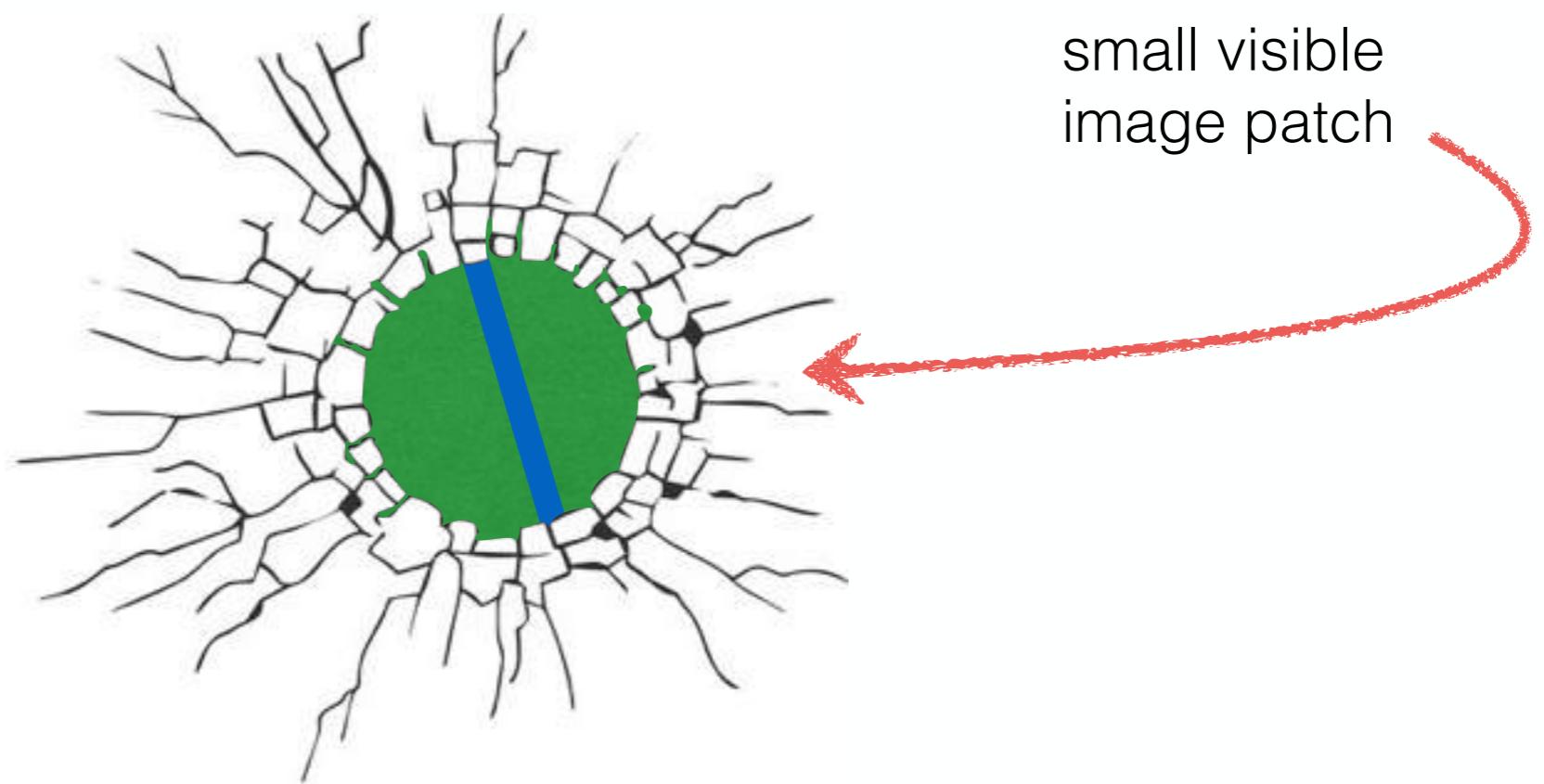
*You want to compute optical flow.
What happens if the image patch contains only a line?*

Aperture Problem



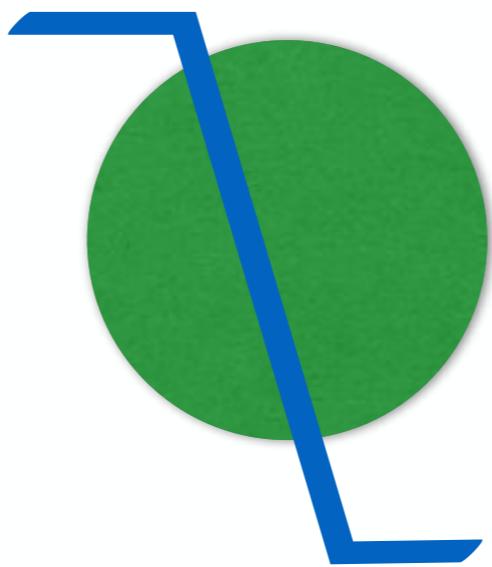
In which direction is the line moving?

Aperture Problem

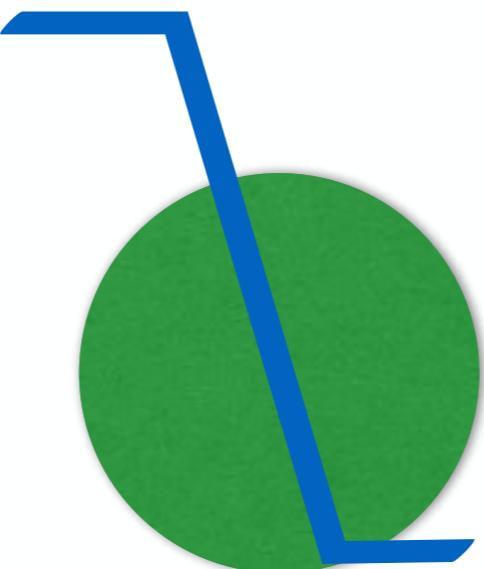


In which direction is the line moving?

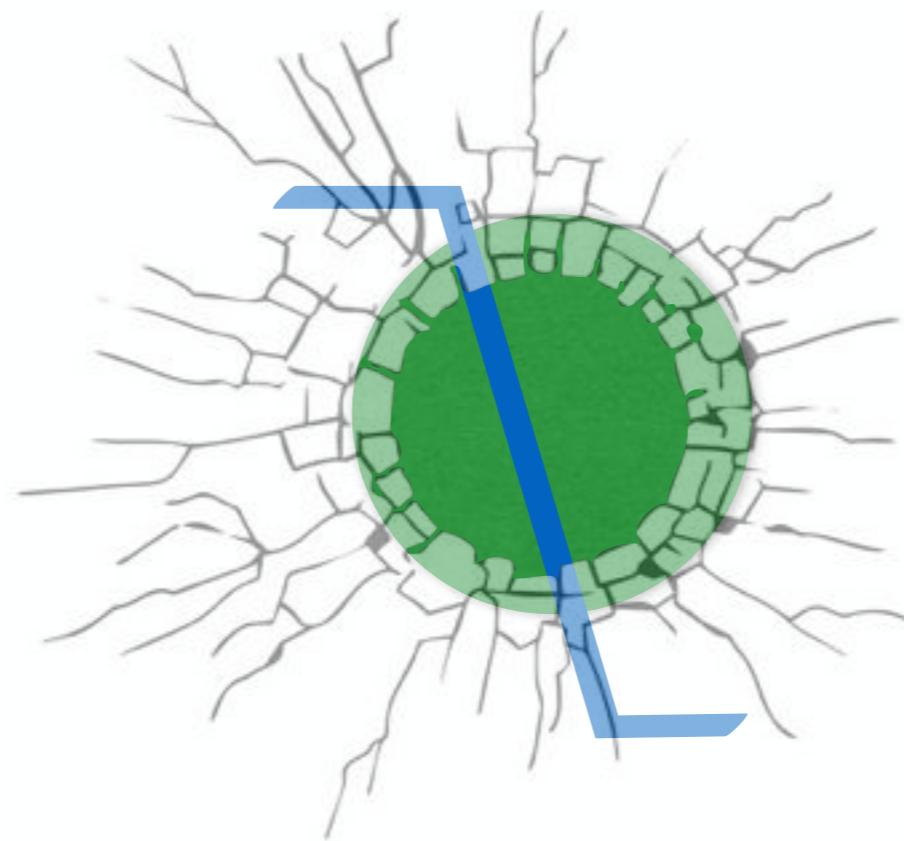
Aperture Problem



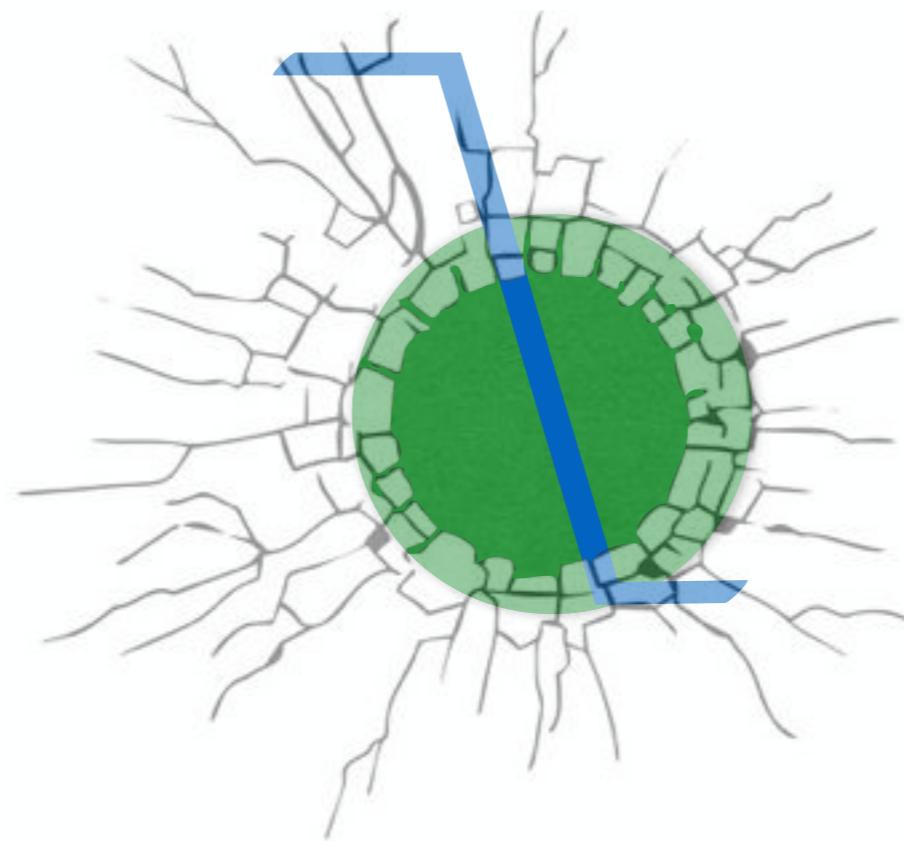
Aperture Problem

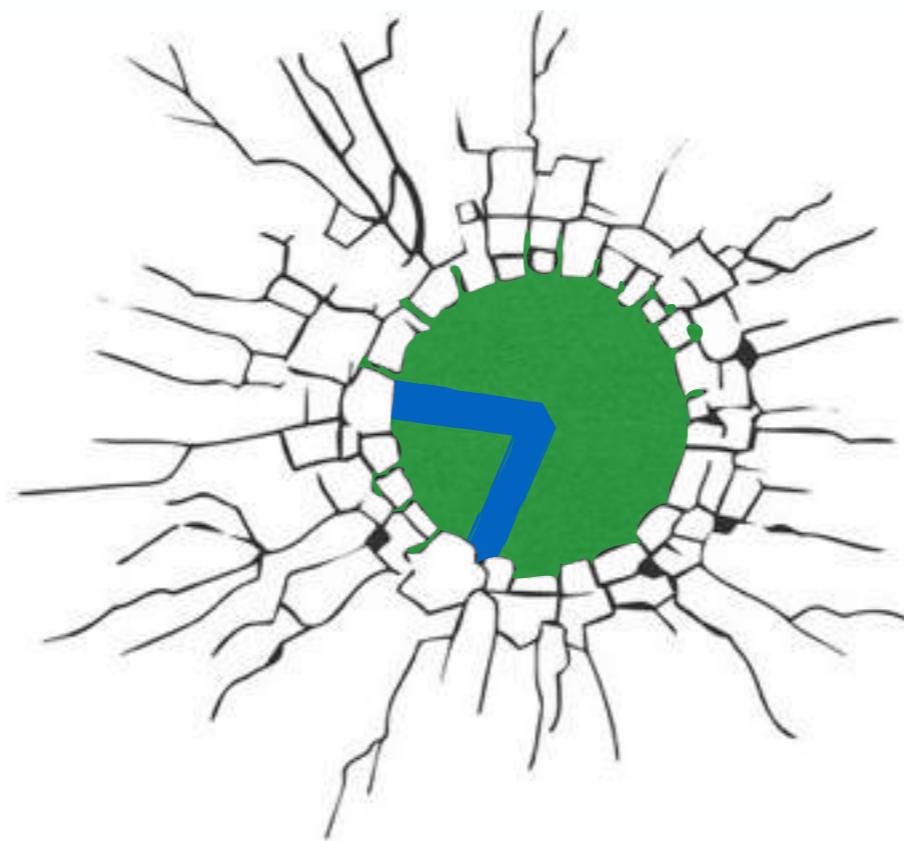


Aperture Problem

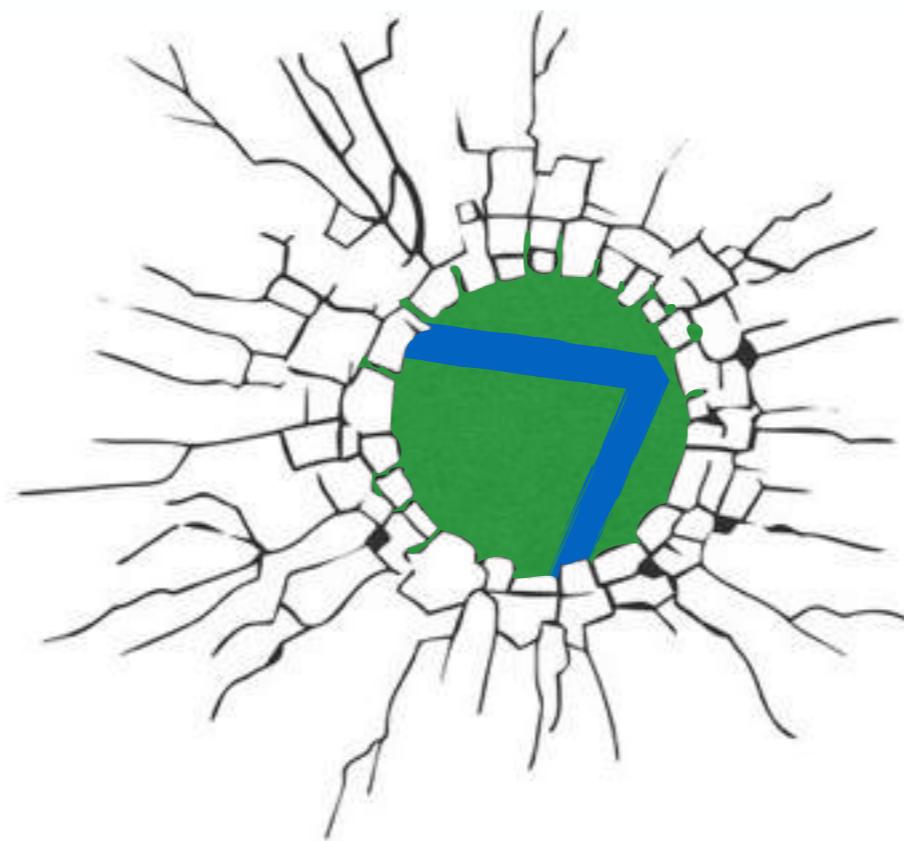


Aperture Problem





Want patches with different gradients to
the avoid aperture problem



Want patches with different gradients to
the avoid aperture problem

$$H(x,y) = y$$

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5

$$I(x,y)$$

-	-	-	-
-	1	1	1
-	2	2	2
-	3	3	3
-	4	4	4

optical flow: (1,1)

$$I_x u + I_y v + I_t = 0$$

Compute gradients

$$I_x(3,3) = 0$$

$$I_y(3,3) = 1$$

$$I_t(3,3) = I(3,3) - H(3,3) = -1$$

Solution:

$$\rightarrow v = 1$$

We recover the v of the optical flow but not the u .

This is the aperture problem.

Computer Vision

Image Categorization

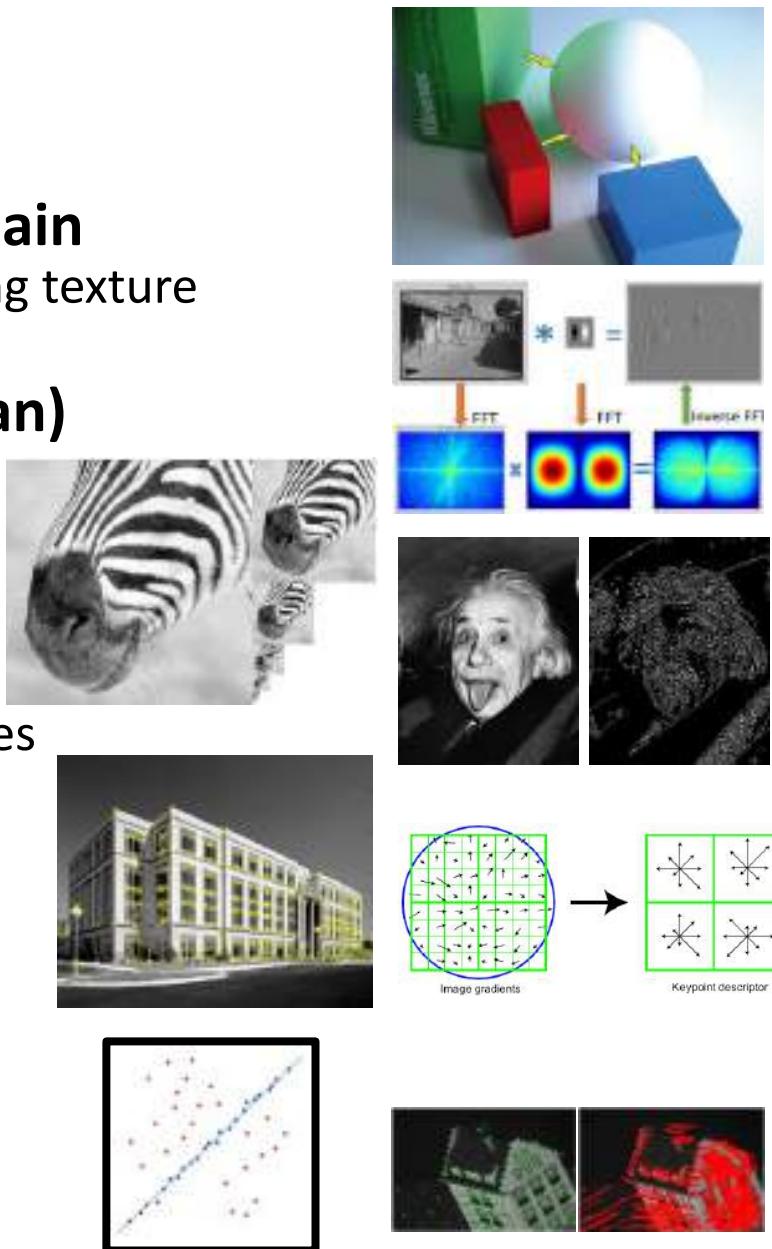
Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**



Topics Covered

- **Light and color**
 - What an image records
- **Filtering in spatial and frequency domain**
 - Filtering, Smoothing, sharpening, measuring texture
 - Denoising, sampling
- **Image pyramid (Gaussian and Laplacian)**
 - Multi-scale analysis
- **Edge detection**
 - Canny edge, Finding straight lines
- **Interest points**
 - Find *distinct* and *repeatable* points in images
 - Harris-> corners, DoG -> blobs
 - SIFT -> feature descriptor
- **Feature tracking and optical flow**
 - Find motion of a keypoint/pixel over time
 - Lucas-Kanade, Handle large motion
- **Fitting and alignment**
 - find the transformation parameters that best align matched points



Topics Covered

- **Camera Models and Projective Geometry**

- Perspective projection
- Vanishing points/lines

- **Projection Matrix and Calibration**

- $\mathbf{x} = \mathbf{K}[\mathbf{R} \ \mathbf{t}] \mathbf{X}$
- Calibration using known 3D object or vanishing points

- **Single-view Metrology and Camera Properties**

- Measuring size using perspective cues
- Focal length, Field of View, etc.

- **Photo stitching**

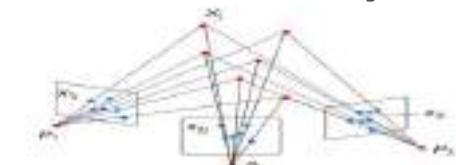
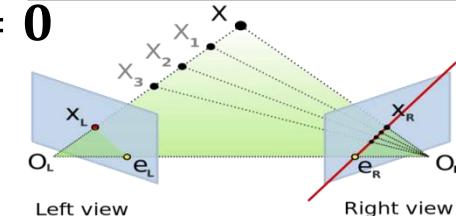
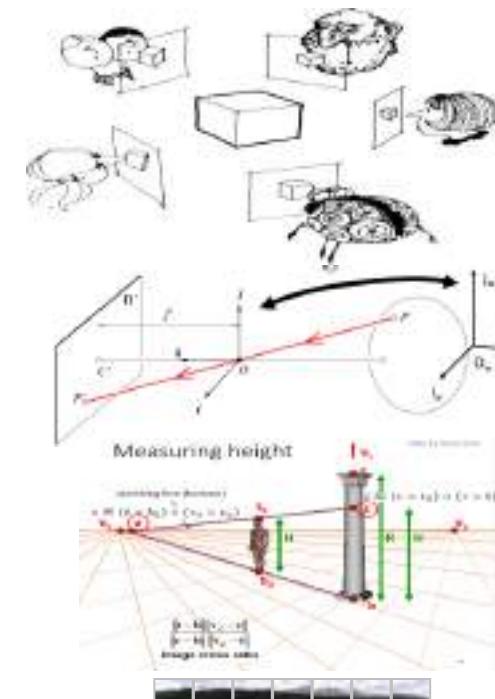
- Homography relates rotating cameras $\mathbf{x}' = \mathbf{Hx}$
- Recover homography using RANSAC

- **Epipolar Geometry and Stereo Vision**

- Fundamental/essential matrix relates two cameras $\mathbf{x}' \mathbf{F} \mathbf{x} = \mathbf{0}$
- Recover \mathbf{F} using RANSAC + normalized 8-point algorithm
- Enforce rank 2 using SVD

- **Structure from motion**

- How can we recover 3D points from multiple images?



Recognition and Learning

- Image Features and Categorization
- Classifiers
- Neural Networks
- Convolutional Neural Networks
- Object Detection
- Segmentation
- Image Generation
- Etc.

Today: Image features and categorization

- General concepts of categorization
 - Why? What? How?
- Image features
 - Color, texture, gradient, shape, interest points
 - Histograms, SIFT, LBP, HoG
 - Bag of Visual Words
 - CNN as feature
- Image and region categorization

What do you see in this image?

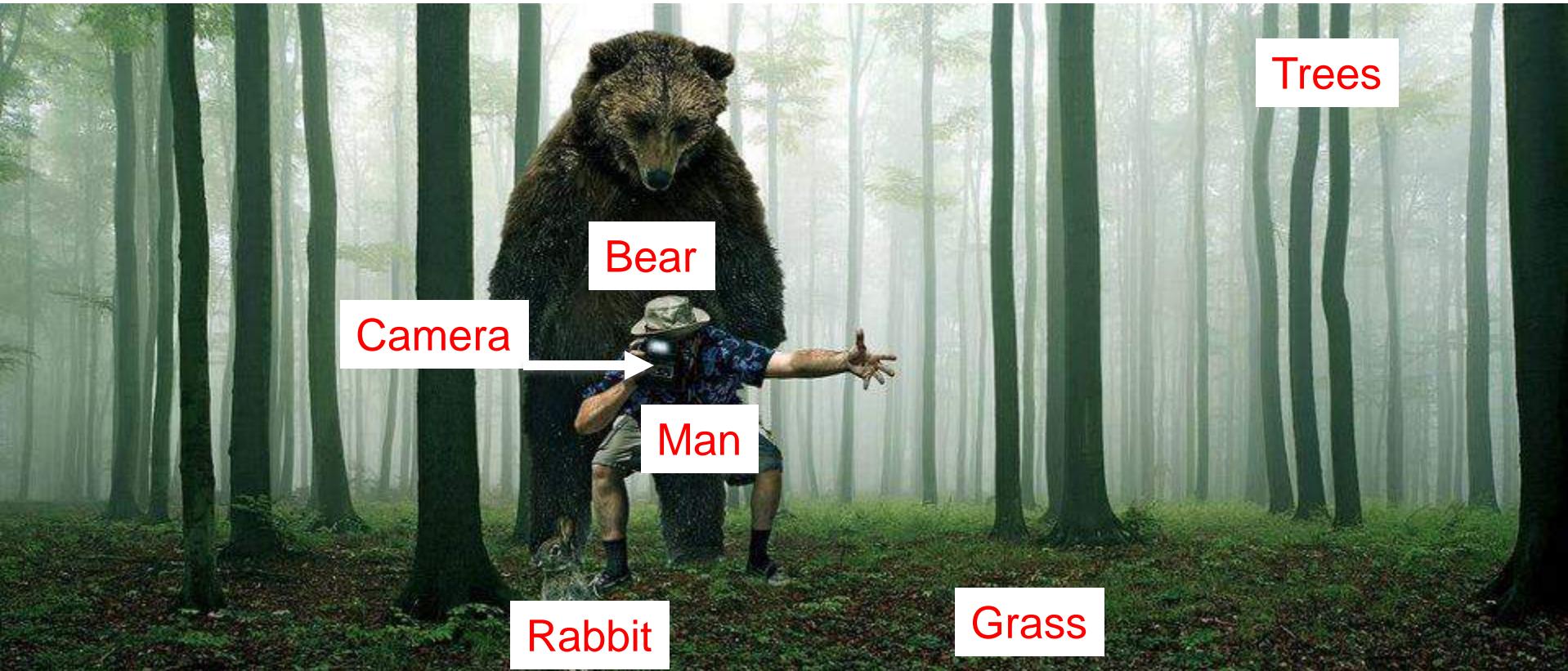


What do you see in this image?



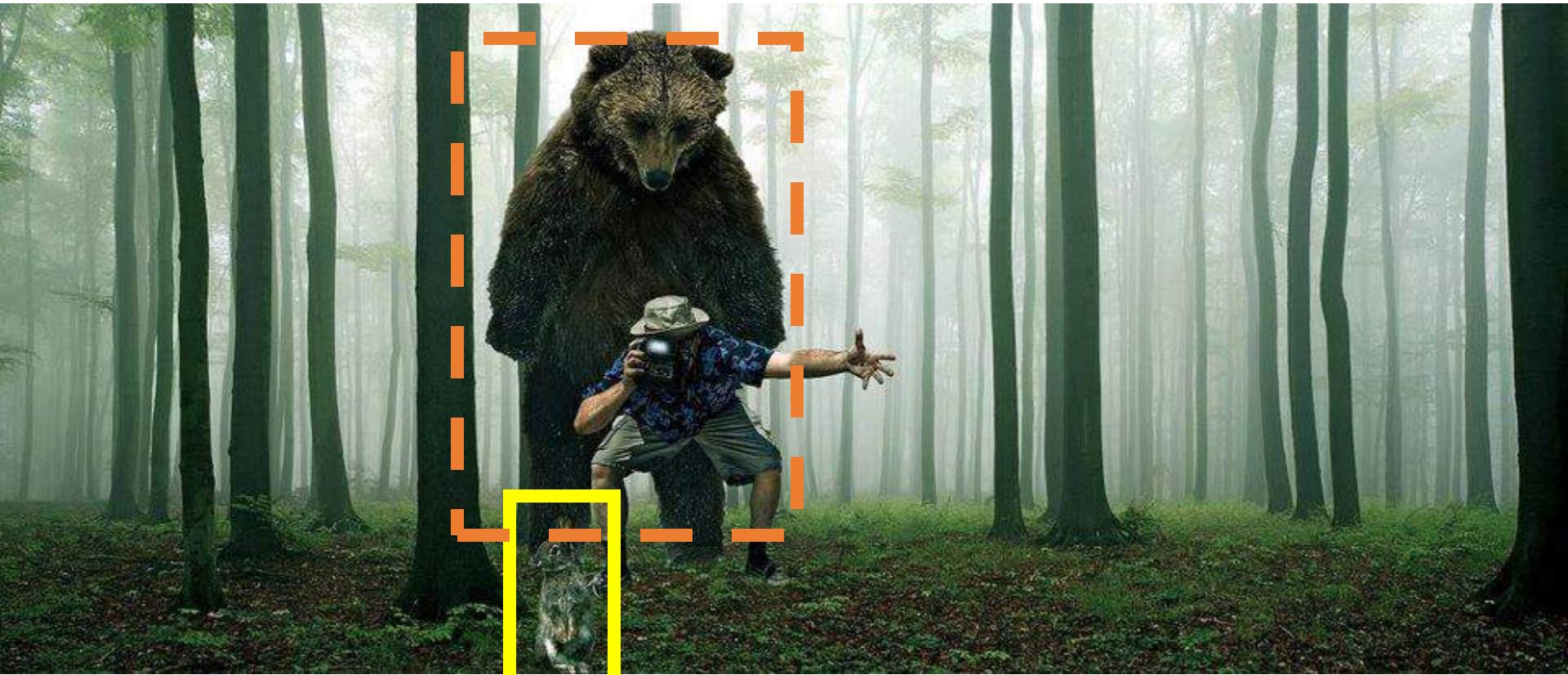
Forest

What do you see in this image?



Forest

Describe, predict, or interact with the object based on visual cues



Is it **dangerous**?

How **fast** does it run?

Is it **alive**?

Does it have a **tail**?

Is it **soft**?

Can I **poke with it**?

Why do we care about categories?

- From an object's category, we can make predictions about its behavior in the future, beyond of what is immediately perceived.
- Pointers to knowledge
 - Help to understand individual cases not previously encountered

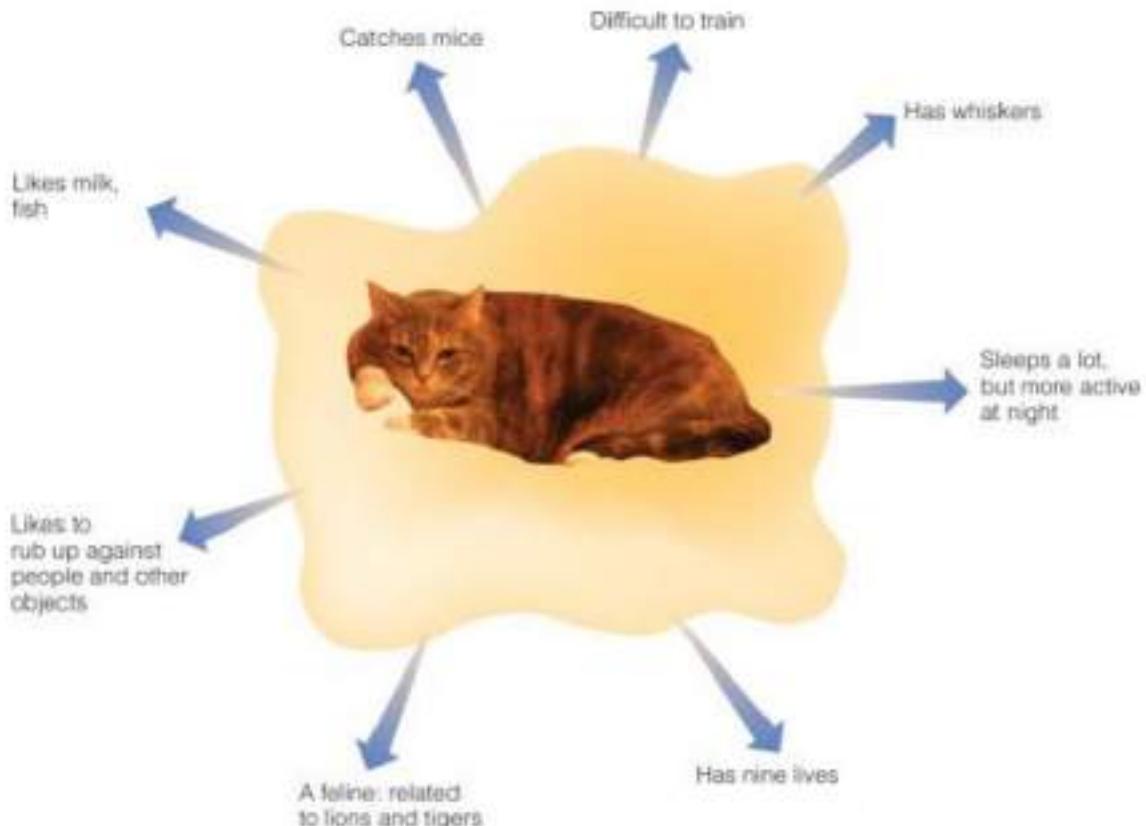


Image categorization

- Cat vs Dog



Image categorization

- Object recognition



Caltech 101 Average Object Images

Image categorization

- Place recognition



spare bedroom

teenage bedroom

romantic bedroom



darkest forest path

wintering forest path

greener forest path



wooded kitchen

messy kitchen

stylish kitchen



rocky coast

misty coast

sunny coast

Places Database [Zhou et al. NIPS 2014]

Image categorization

- Image style recognition



HDR



Macro



Baroque



Rococo



Vintage



Noir



Northern Renaissance



Cubism



Minimal



Hazy



Impressionism



Post-Impressionism



Long Exposure



Romantic



Abs. Expressionism



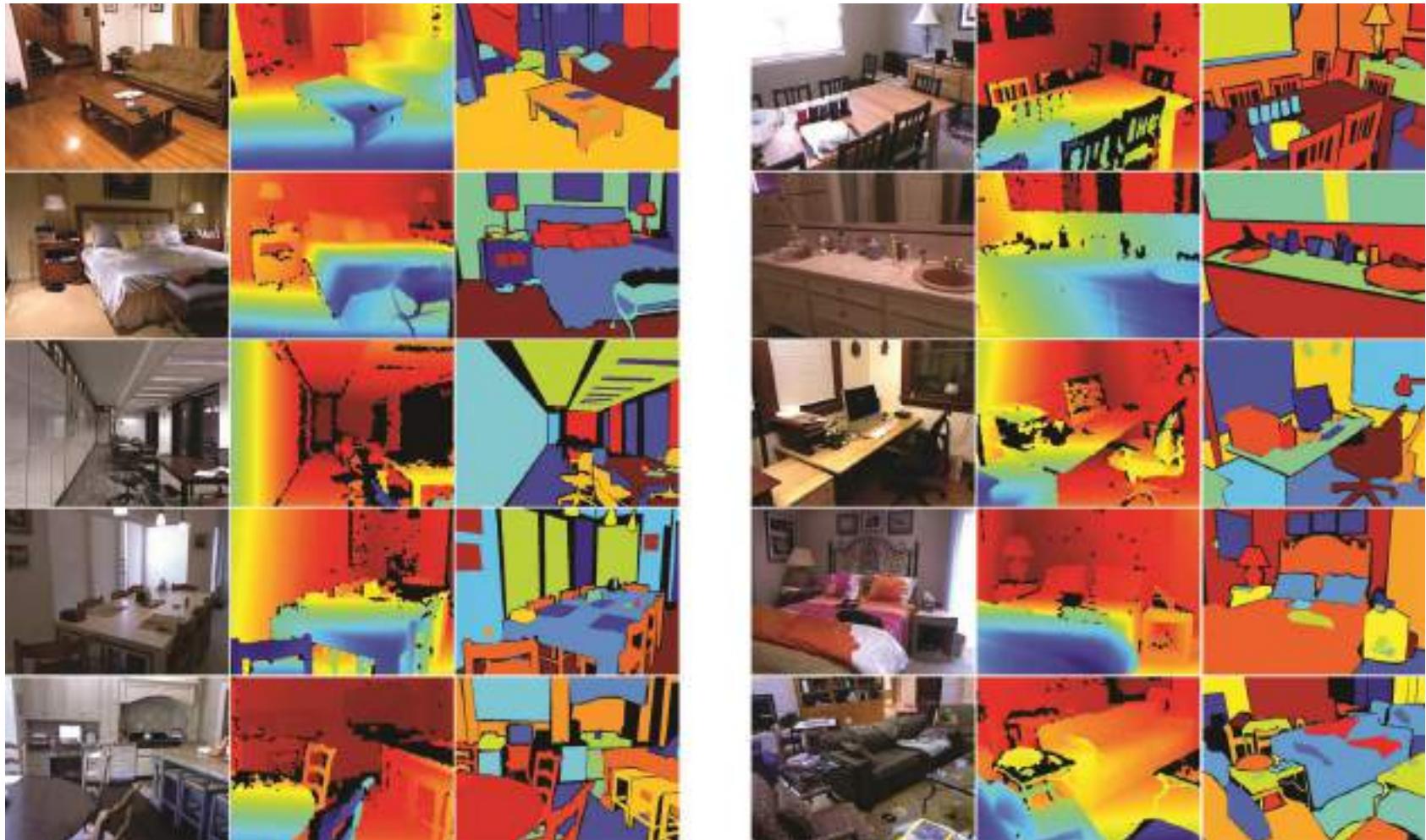
Color Field Painting

Flickr Style: 80K images covering 20 styles.

Wikipaintings: 85K images for 25 art genres.

Region categorization

- Semantic segmentation from RGBD images



[Silberman et al. ECCV 2012]

Region categorization

- Material recognition

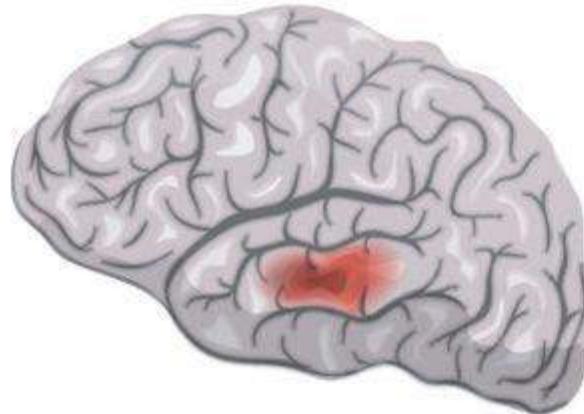


[[Bell et al. CVPR 2015](#)]

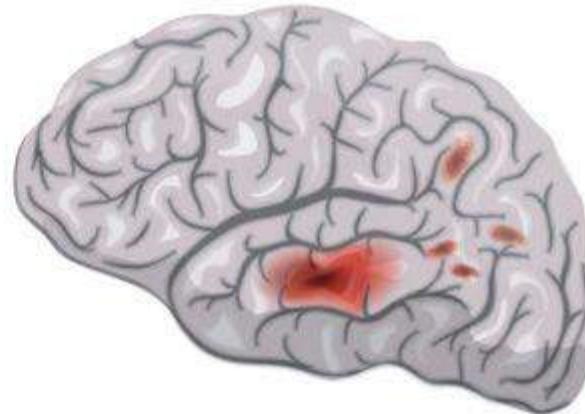
Image categorization

- Primary Tumor vs Metastasis

Brain Cancer



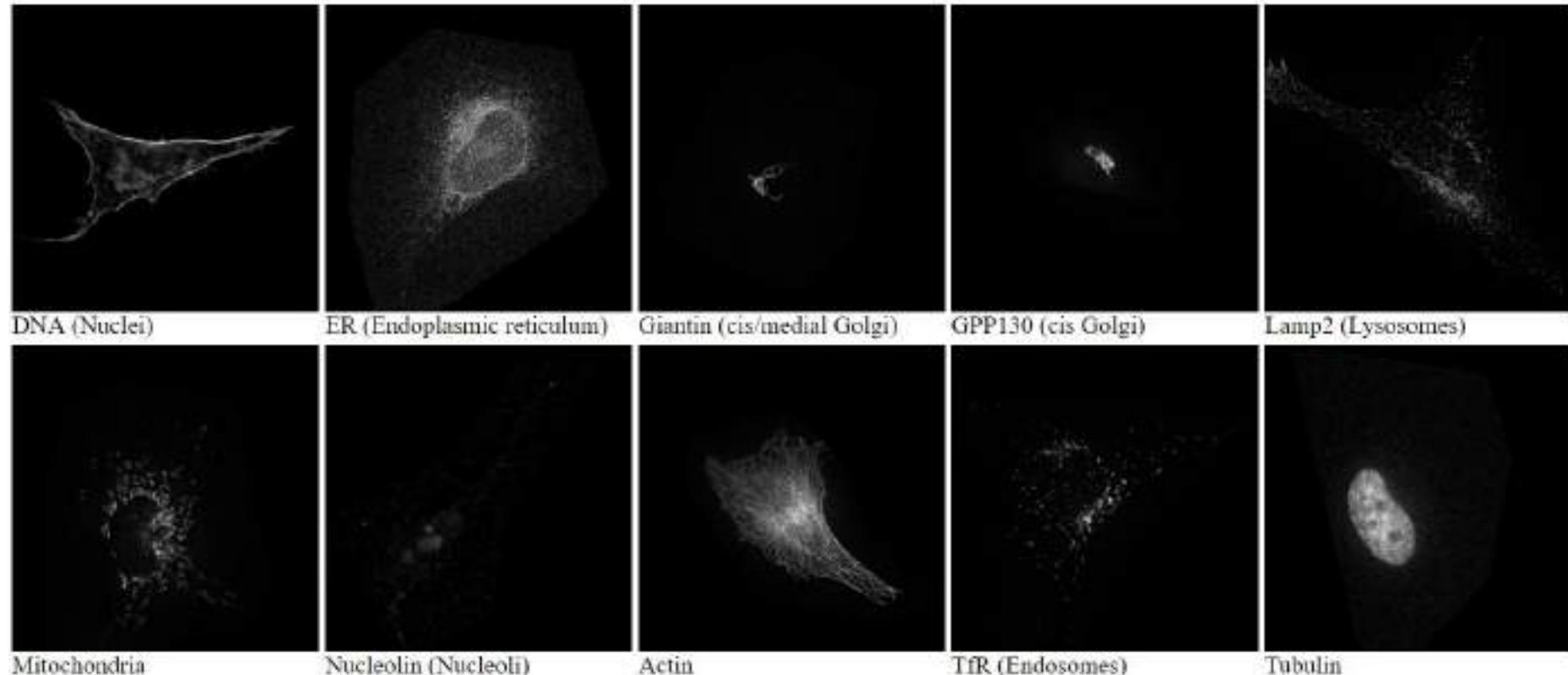
the primary tumor



metastasis

Image categorization

- Identification of sub-cellular organelles

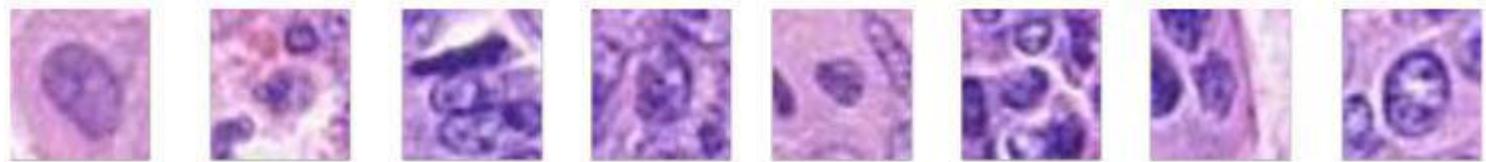


Fluorescence microscopy images of HeLa cells

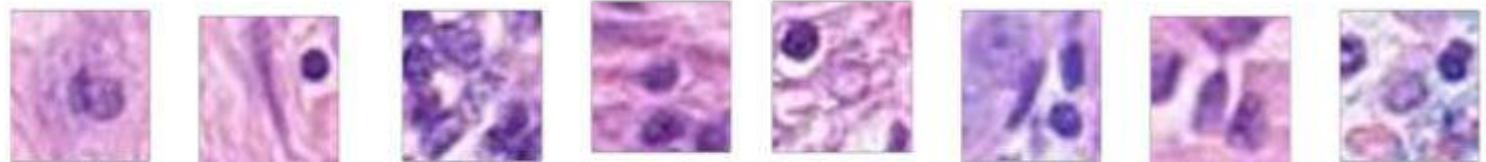
Image categorization

- Colon Cancer Nuclei Classification

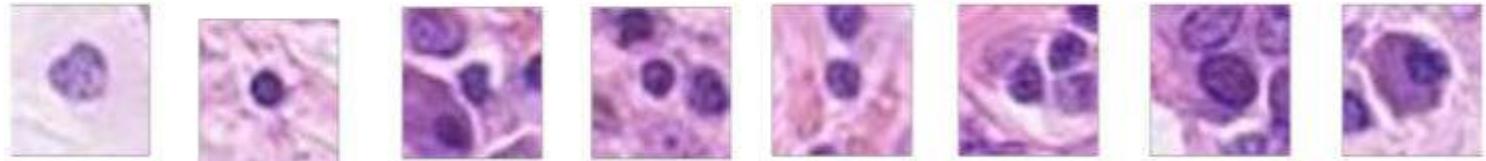
'Epithelial'



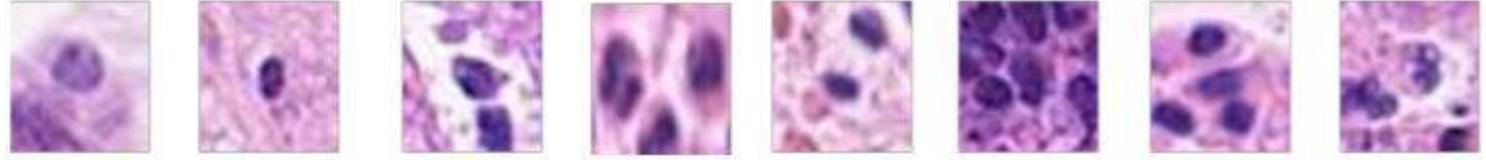
'Fibroblast'



'Inflammatory'

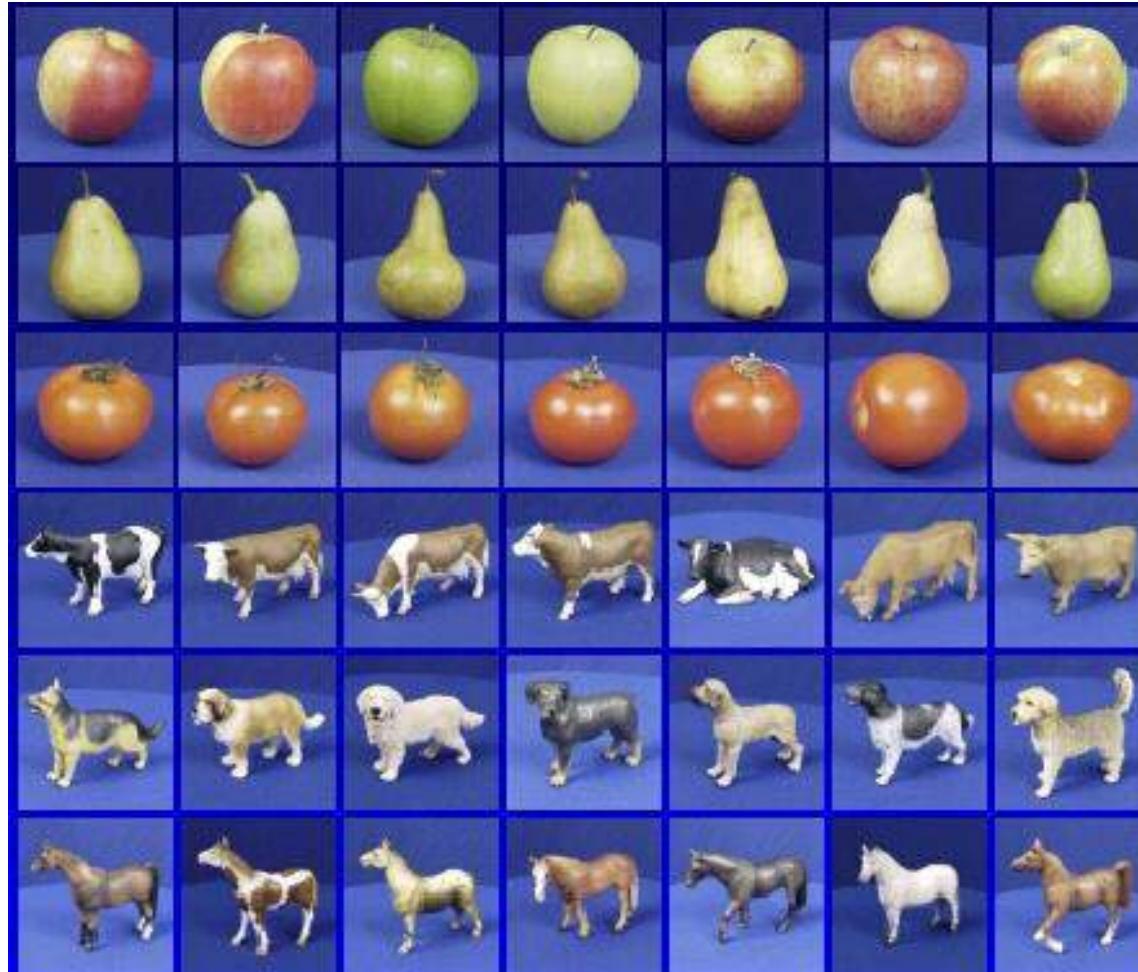


'Miscellaneous'



"CRCHistoPhenotypes" dataset images

Image categorization/ classification



The statistical learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

$$f(\text{apple}) = \text{"apple"}$$
$$f(\text{tomato}) = \text{"tomato"}$$
$$f(\text{cow}) = \text{"cow"}$$

The statistical learning framework

$$y = f(x)$$

A diagram illustrating the statistical learning framework. At the top is the equation $y = f(x)$ in blue. Three red arrows point downwards from this equation to three labels below it: "output" on the left, "prediction function" in the middle, and "Image feature" on the right. The "Image feature" arrow points diagonally upwards and to the left towards the variable x .

output prediction function Image feature

The statistical learning framework

$$y = f(x)$$

A diagram illustrating the components of the equation $y = f(x)$. The equation is written in blue. Three red arrows point from labels below to specific parts of the equation: one arrow points to the variable y and is labeled "output"; another arrow points to the function f and is labeled "prediction function"; a third arrow points to the variable x and is labeled "Image feature".

- **Training:** given a *training set* of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set

The statistical learning framework

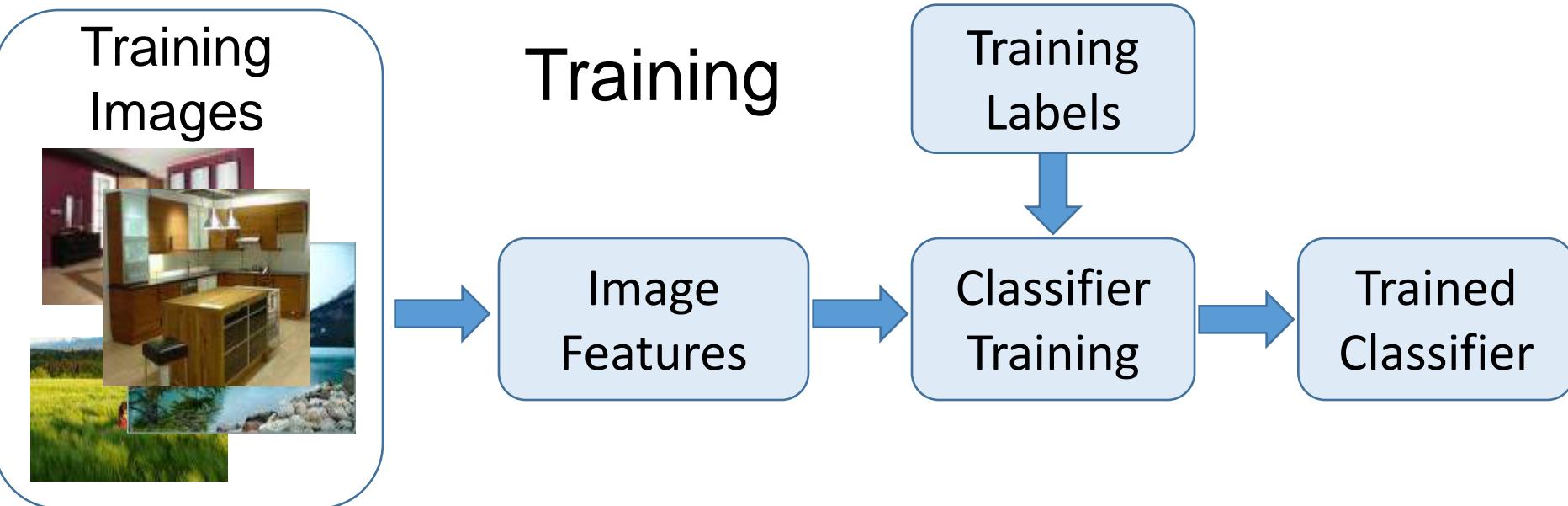
$$y = f(x)$$

A diagram illustrating the components of the prediction function $y = f(x)$. The equation is written in blue. Three red arrows point upwards from below to each part of the equation: one arrow points to the output variable y , another to the prediction function f , and a third to the input feature x .

output prediction function Image feature

- **Training:** given a *training set* of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* x and output the predicted value $y = f(x)$

Training phase



Training phase

Training Images



Training

Image Features

Training Labels

Classifier Training

Trained Classifier

Testing phase

Testing



Image Features

Trained Classifier

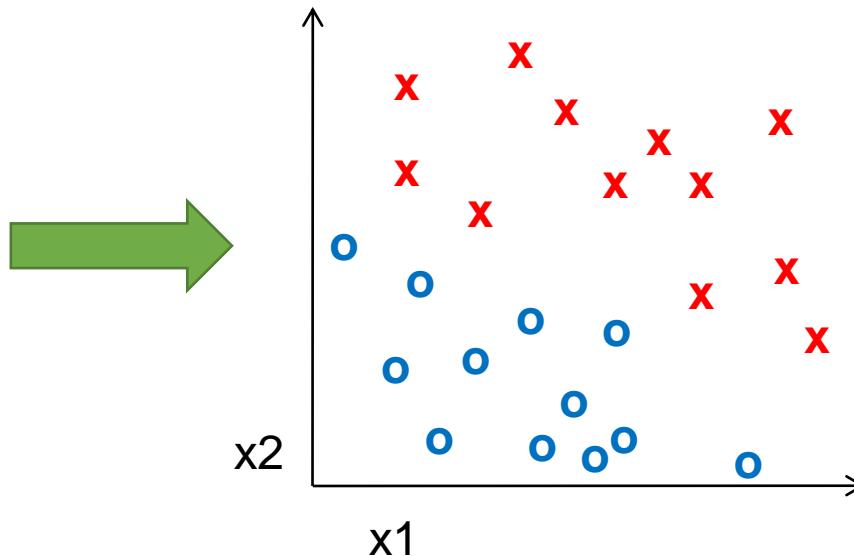
Prediction
Outdoor

Test Image

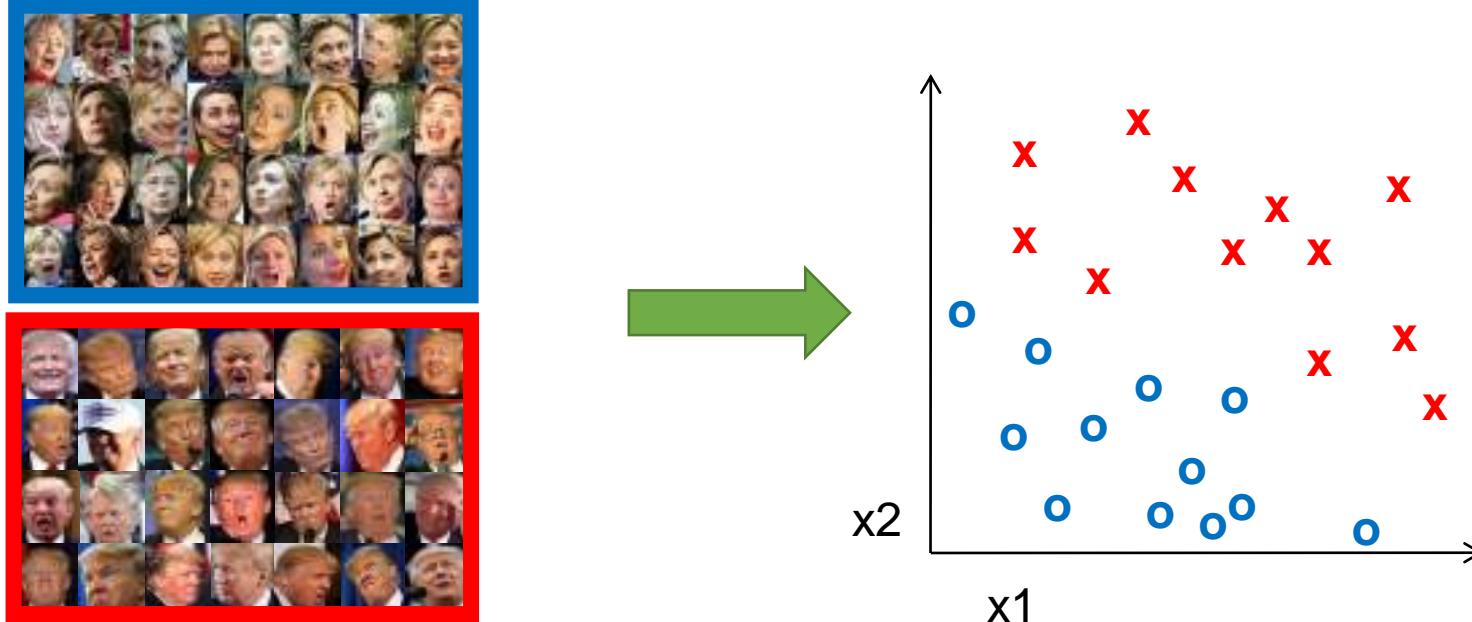
- **Image features:** map images to feature space



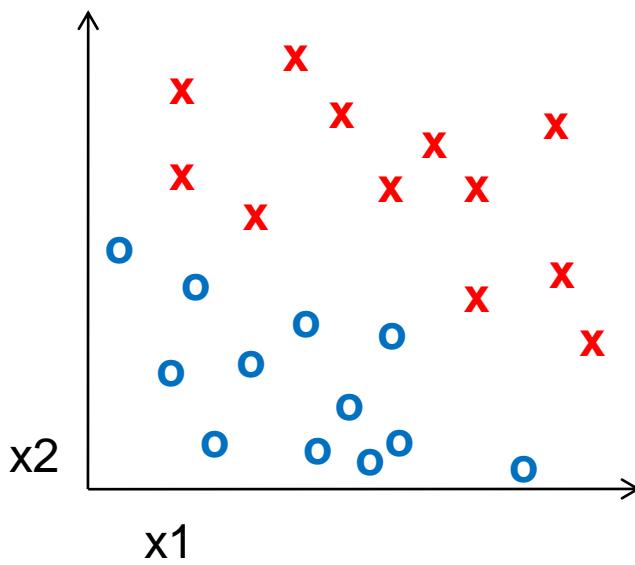
- **Image features:** map images to feature space



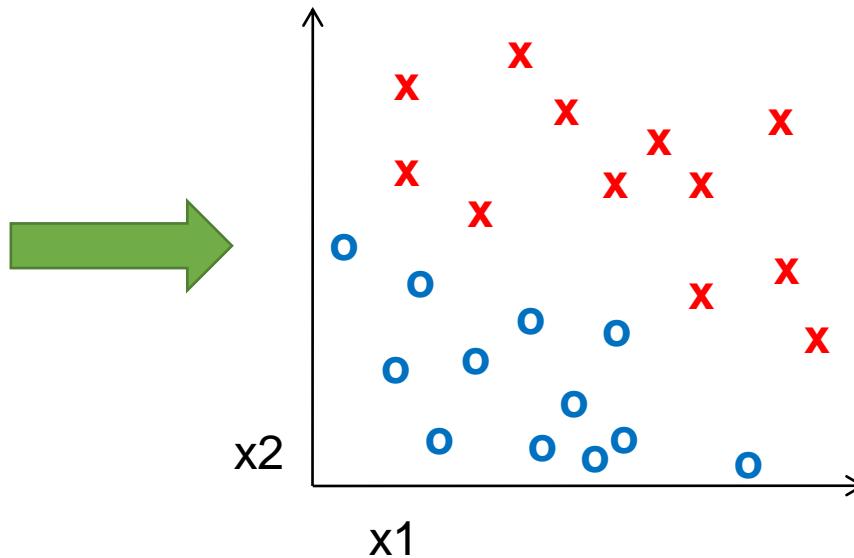
- **Image features:** map images to feature space



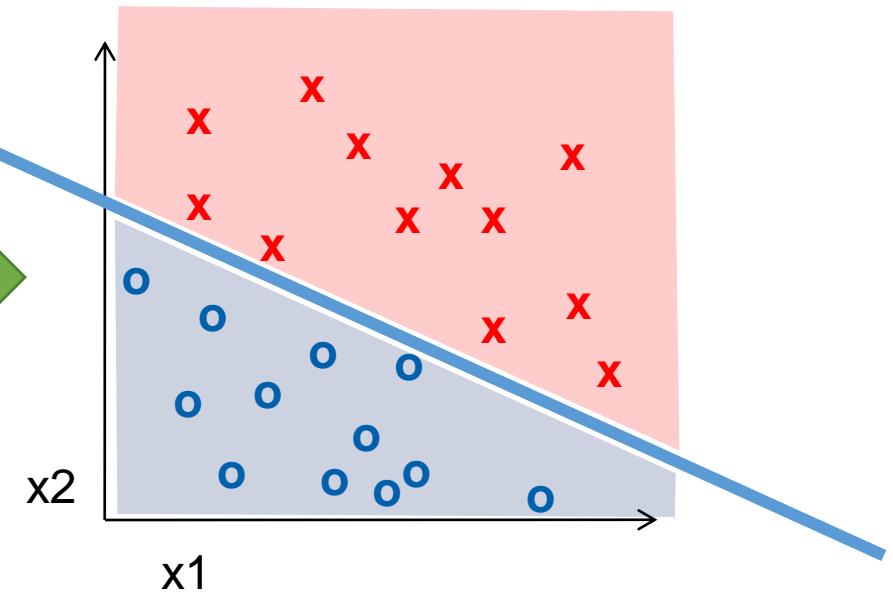
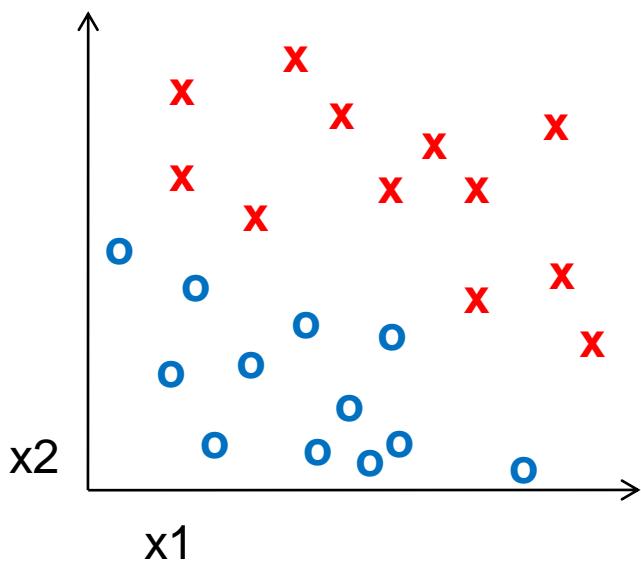
- **Classifiers:** map feature space to label space



- **Image features:** map images to feature space



- **Classifiers:** map feature space to label space



Training phase

Training Images



Training

Image Features

Training Labels

Classifier Training

Trained Classifier

Testing phase

Testing



Image Features

Trained Classifier

Prediction
Outdoor

Test Image

Q: What are good features for...

- recognizing a beach?



Q: What are good features for...

- recognizing cloth fabric?



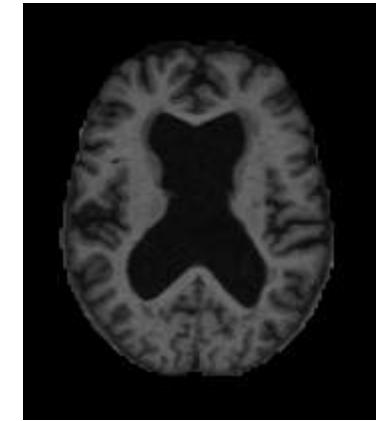
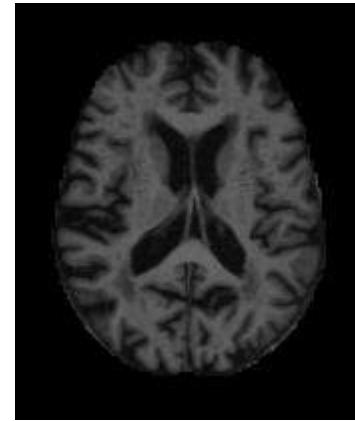
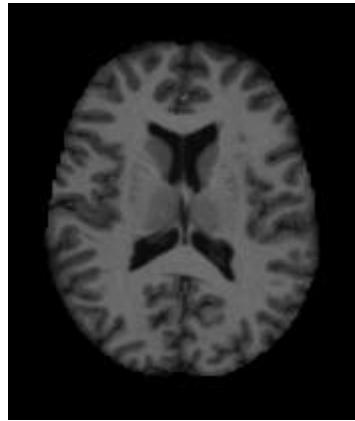
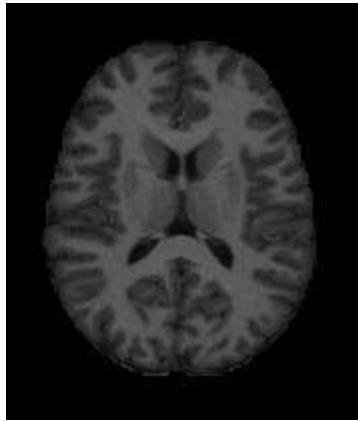
Q: What are good features for...

- recognizing a mug?



Q: What are good features for...

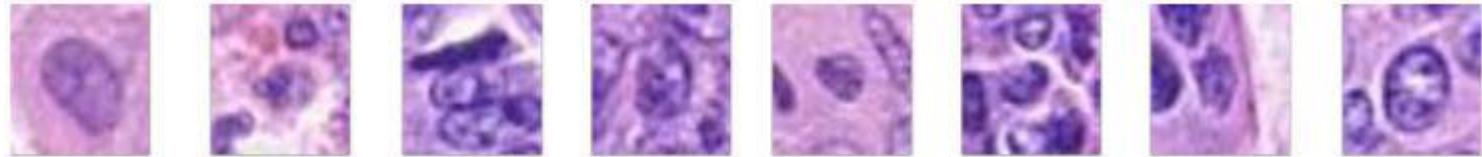
- recognizing the nodule in MRI data?



Q: What are good features for...

- recognizing the type of colon cancer?

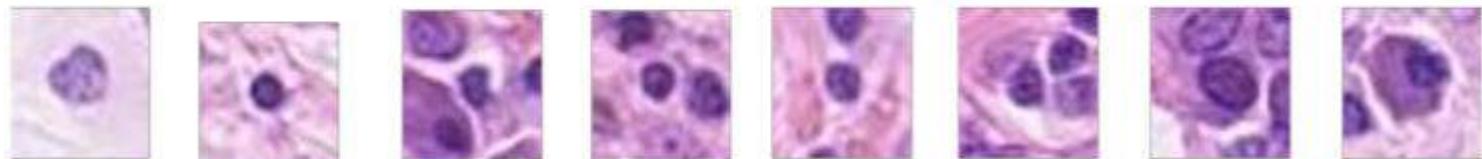
'Epithelial'



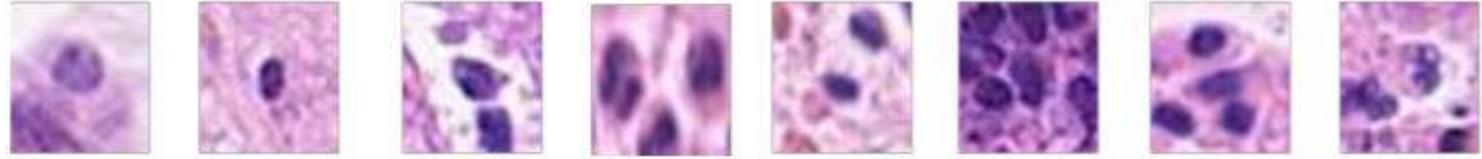
'Fibroblast'



'Inflammatory'



'Miscellaneous'



"CRCHistoPhenotypes" dataset images

What are the right features?

What are the right features?

Depend on what you want to know!

What are the right features?

Depend on what you want to know!

- Object: shape
 - Local shape info, shading, shadows, texture

What are the right features?

Depend on what you want to know!

- Object: shape
 - Local shape info, shading, shadows, texture
- Scene: geometric layout
 - linear perspective, gradients, line segments

What are the right features?

Depend on what you want to know!

- Object: shape
 - Local shape info, shading, shadows, texture
- Scene: geometric layout
 - linear perspective, gradients, line segments
- Material properties: albedo, feel, hardness
 - Color, texture

What are the right features?

Depend on what you want to know!

- Object: shape
 - Local shape info, shading, shadows, texture
- Scene: geometric layout
 - linear perspective, gradients, line segments
- Material properties: albedo, feel, hardness
 - Color, texture
- Action: motion
 - Optical flow, tracked points

Image representations

- Templates
 - Intensity, gradients, etc.



Image
Intensity

Gradient
template

- Histograms
 - Color, texture, SIFT descriptors, etc.

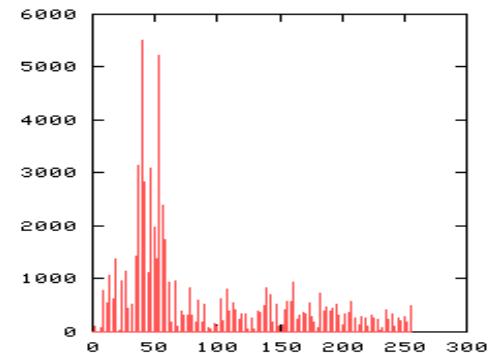
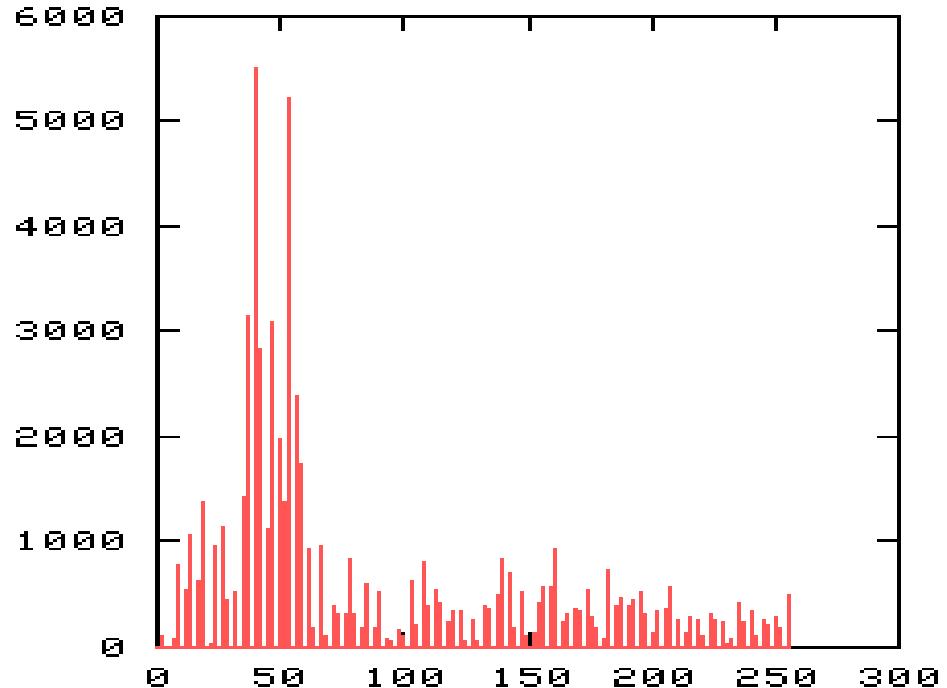
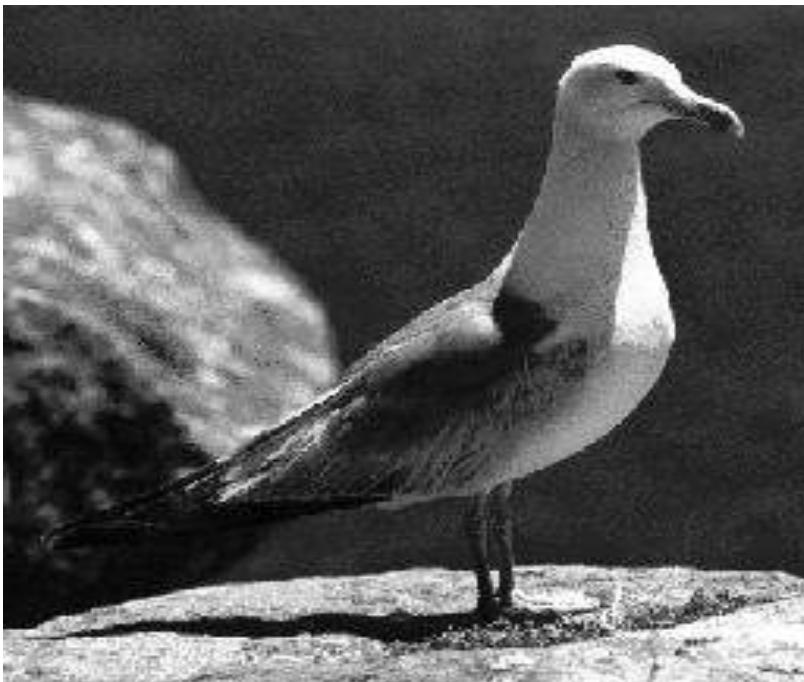


Image representations: histograms



Global histogram

- Represent distribution of features
 - Color, texture, depth, ...

Computing histogram distance

?

Computing histogram distance

- Histogram intersection

$$\text{histint}(h_i, h_j) = 1 - \sum_{m=1}^K \min(h_i(m), h_j(m))$$

- Chi-squared Histogram matching distance

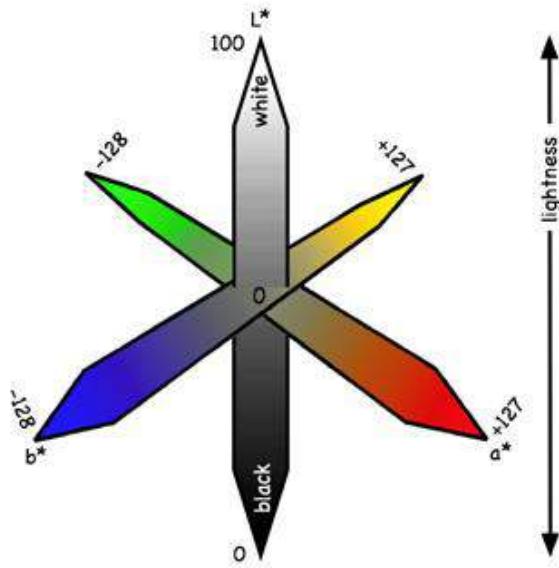
$$\chi^2(h_i, h_j) = \frac{1}{2} \sum_{m=1}^K \frac{[h_i(m) - h_j(m)]^2}{h_i(m) + h_j(m)}$$

- Earth mover's distance
(Cross-bin similarity measure)
 - minimal cost paid to transform one distribution into the other

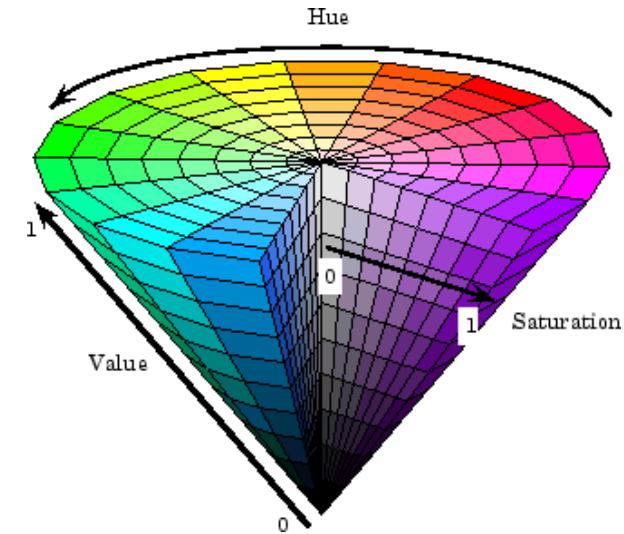
[Rubner et al. [The Earth Mover's Distance as a Metric for Image Retrieval](#), IJCV 2000]

What kind of things do we compute histograms of?

- Color

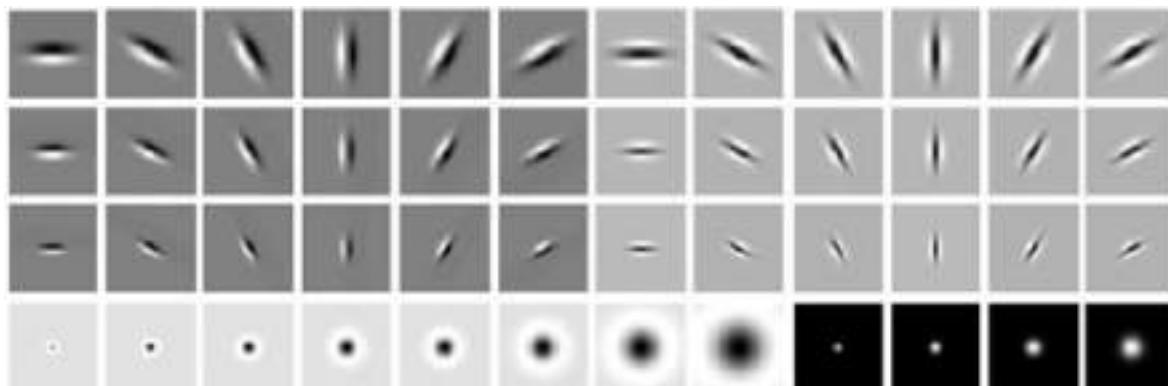


L*a*b* color space



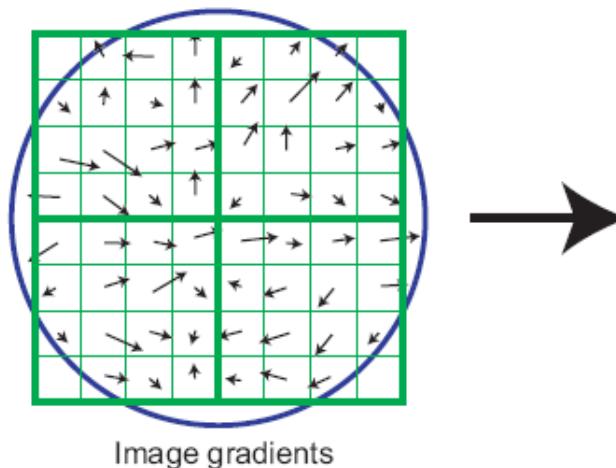
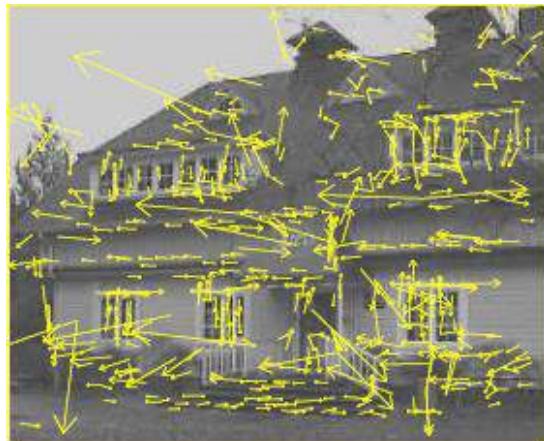
HSV color space

- Texture (filter banks or HOG over regions)



What kind of things do we compute histograms of?

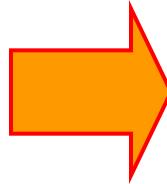
Histograms of descriptors



SIFT – [Lowe IJCV 2004]

What kind of things do we compute histograms of?

Bags of visual words



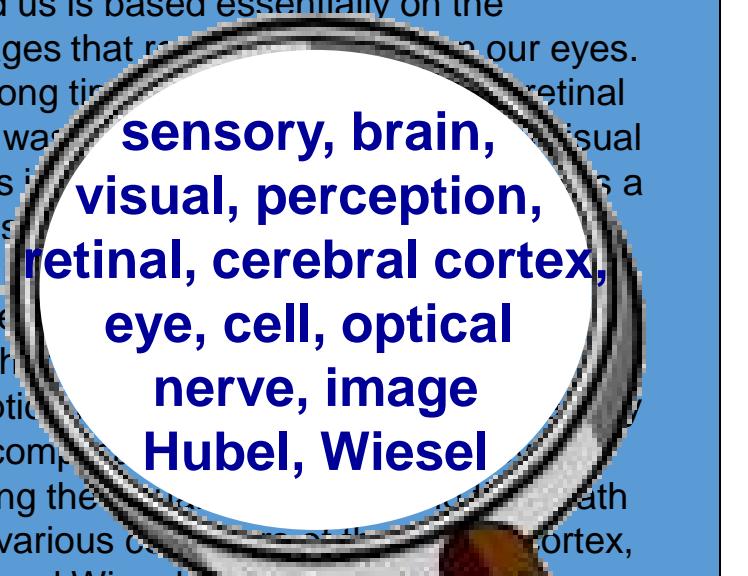
Analogy to documents

Of all the sensory impressions proceeding to the brain, the visual experiences are the dominant ones. Our perception of the world around us is based essentially on the messages that reach the brain from our eyes. For a long time it was thought that the retinal image was transmitted point by point to visual centers in the brain; the cerebral cortex was a movie screen, so to speak, upon which the image in the eye was projected. Through the discoveries of Hubel and Wiesel we now know that behind the origin of the visual perception in the brain there is a considerably more complicated course of events. By following the visual impulses along their path to the various cell layers of the optical cortex, Hubel and Wiesel have been able to demonstrate that the *message about the image falling on the retina undergoes a step-wise analysis in a system of nerve cells stored in columns. In this system each cell has its specific function and is responsible for a specific detail in the pattern of the retinal image.*

China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% jump in exports to \$750bn, compared with a 18% rise in imports to \$660bn. The figures are likely to further annoy the US, which has long argued that China's exports are unfairly helped by a deliberately undervalued yuan. Beijing agrees the surplus is too high, but says the yuan is only one factor. Bank of China governor Zhou Xiaochuan said the country also needed to do more to boost domestic demand so more goods stayed within the country. China increased the value of the yuan against the dollar by 2.1% in July and permitted it to trade within a narrow band, but the US wants the yuan to be allowed to trade freely. However, Beijing has made it clear that it will take its time and tread carefully before allowing the yuan to rise further in value.

Analogy to documents

Of all the sensory impressions proceeding to the brain, the visual experiences are the dominant ones. Our perception of the world around us is based essentially on the messages that reach our brain via our eyes. For a long time it was believed that the retinal image was processed directly in the visual centers in the brain. In 1961, however, a movie showing the flow of information in the visual system was made. It was discovered that the visual system does not know the whole image at once. The visual perception is built up step by step, more complex, more complete, following the path through the optic nerve to the various columns of the cerebral cortex. Hubel and Wiesel have shown that the message about the image falling on the retina undergoes a top-down analysis in a system of nerve cells stored in columns. In this system each column has its specific function and is responsible for a specific detail in the pattern of the retinal image.



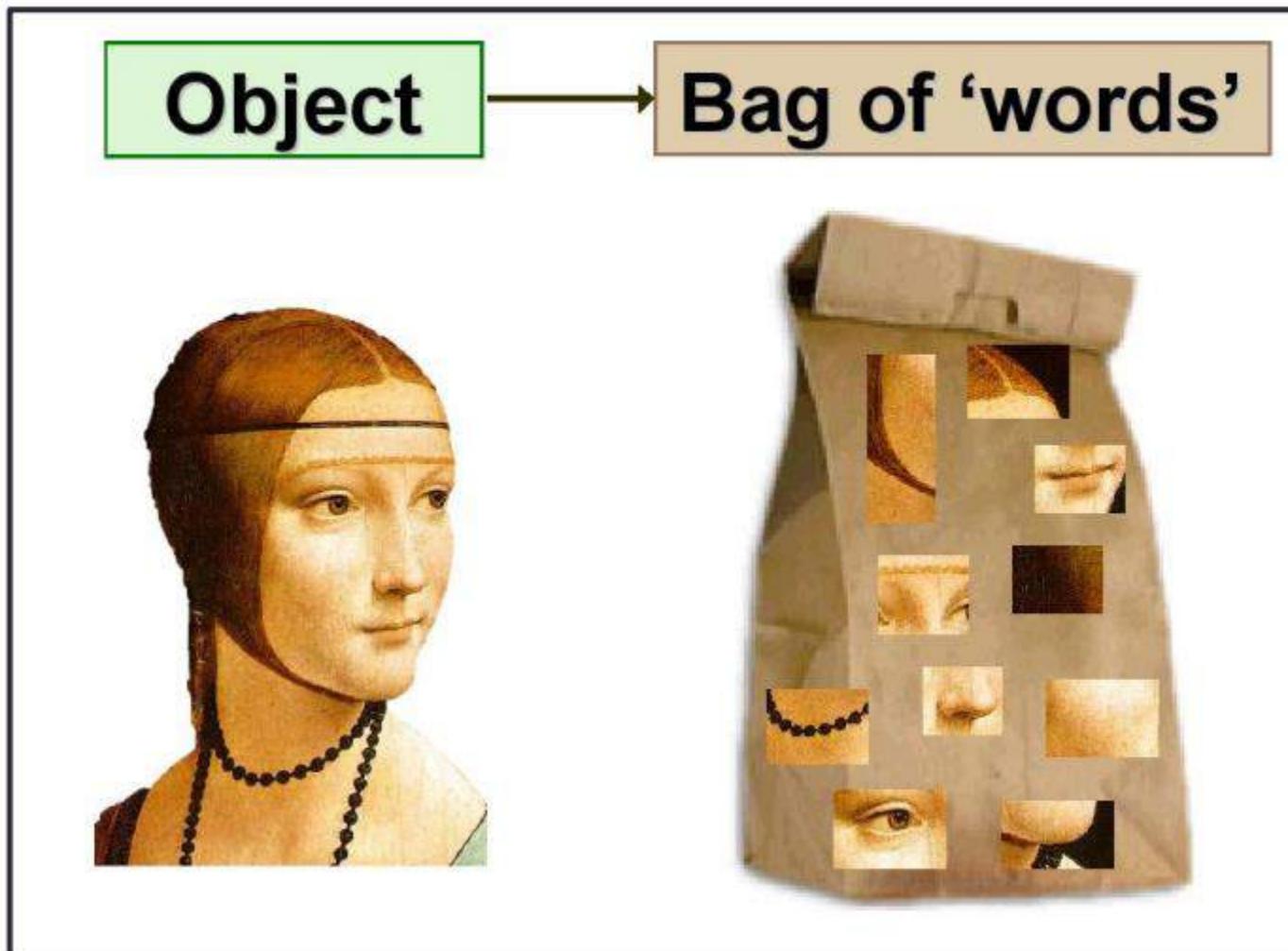
**sensory, brain,
visual, perception,
retinal, cerebral cortex,
eye, cell, optical
nerve, image
Hubel, Wiesel**

China is forecasting a trade surplus of \$90bn (£51bn) to \$100bn this year, a threefold increase on 2004's \$32bn. The Commerce Ministry said the surplus would be created by a predicted 30% increase in exports to \$750bn, compared with \$660bn. That would annoy the US, which China's leaders deliberately agreed to do. The yuan is governed by the Chinese government, which also needs to manage the demand so that it does not damage the country. China has been allowed to let the yuan against the dollar rise slowly and permitted it to trade within a narrow band, but the US wants the yuan to be allowed to trade freely. However, Beijing has made it clear that it will take its time and tread carefully before allowing the yuan to rise further in value.



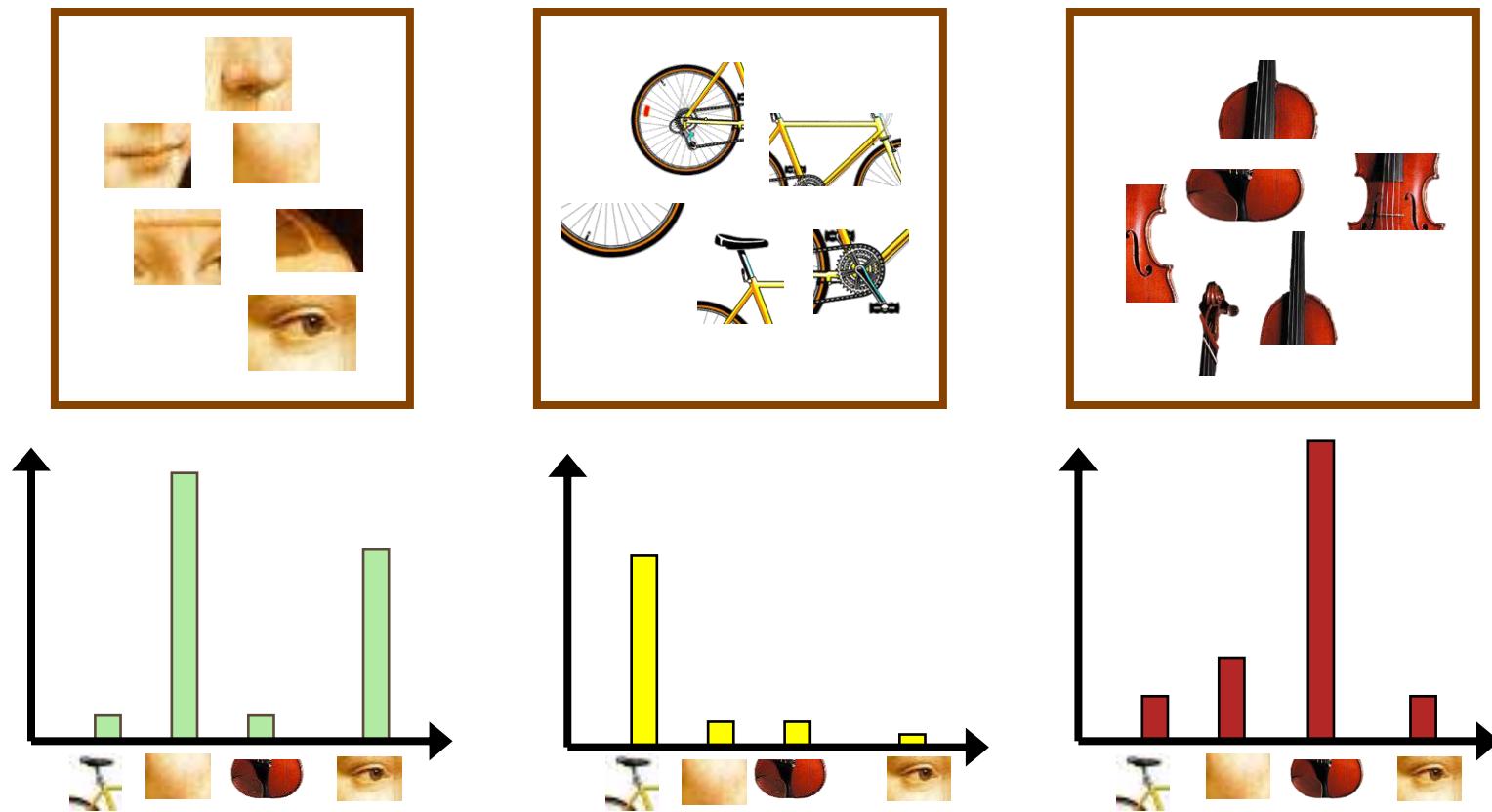
**China, trade,
surplus, commerce,
exports, imports, US,
yuan, bank, domestic,
foreign, increase,
trade, value**

Bags of visual words: Motivation



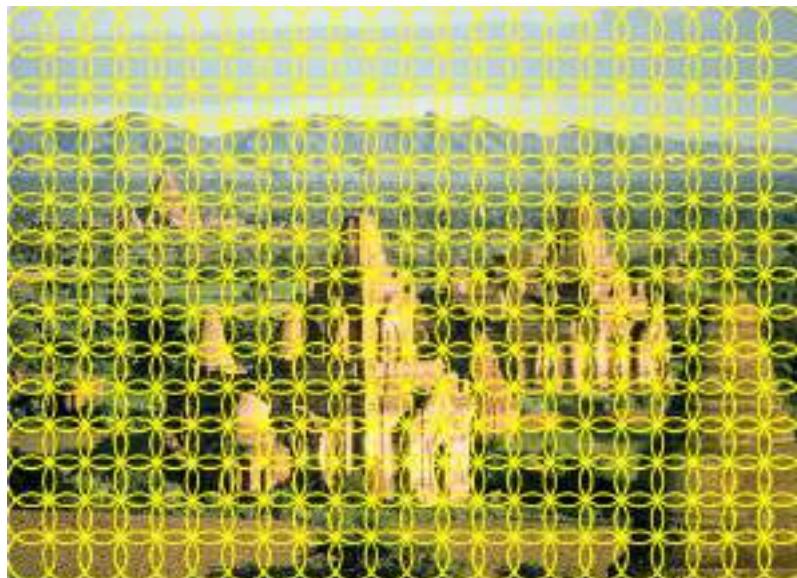
Bags-of-visual-words

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

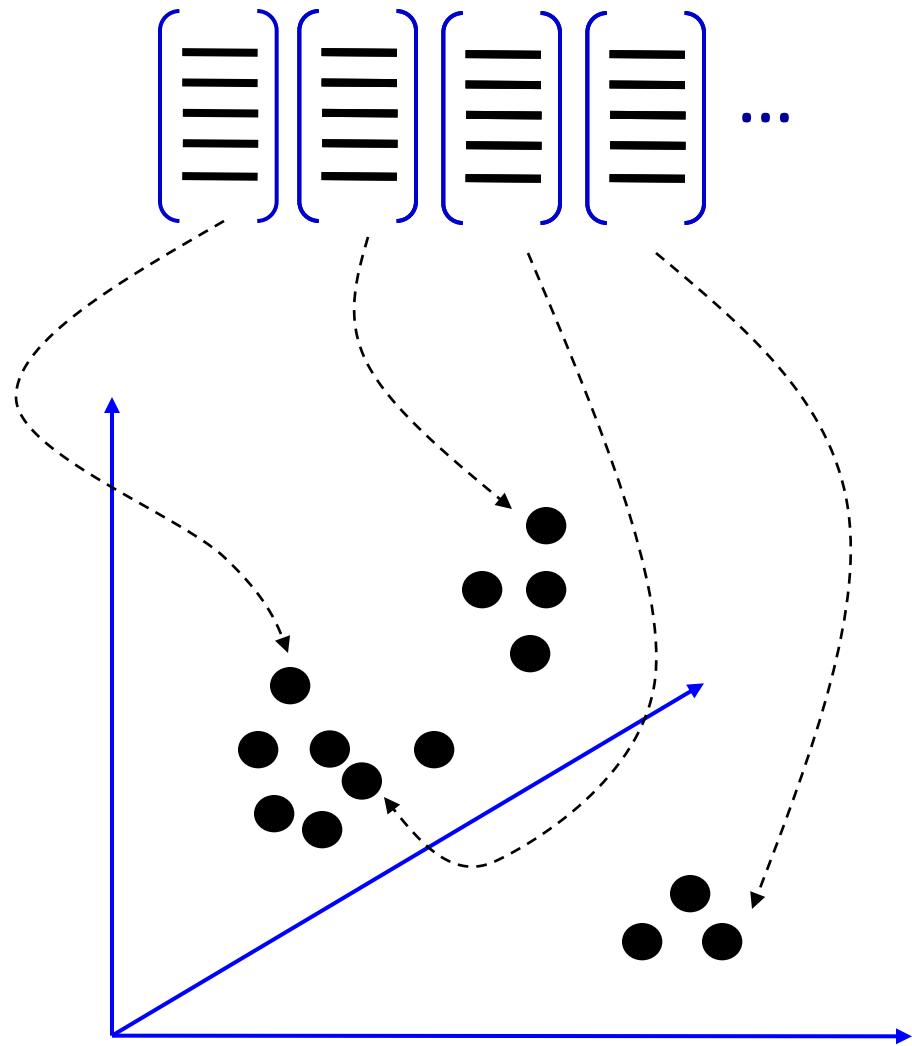


1. Local feature extraction

- Sample patches and extract descriptors

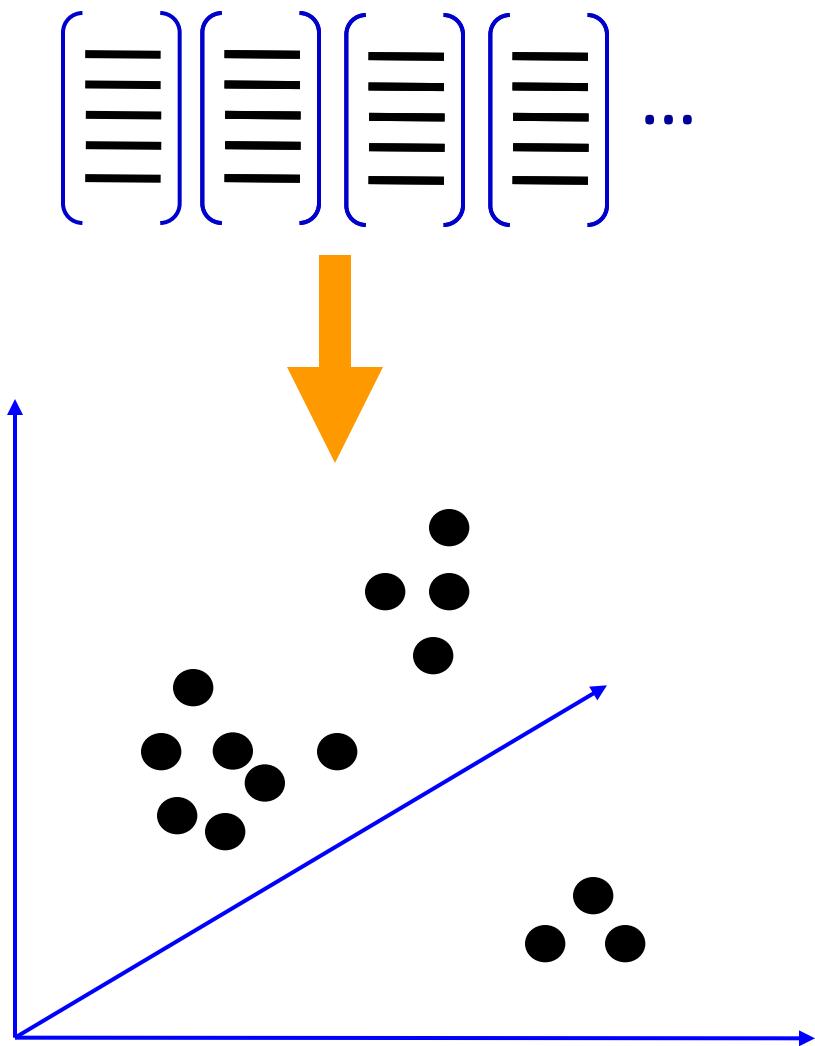


2. Learning the visual vocabulary

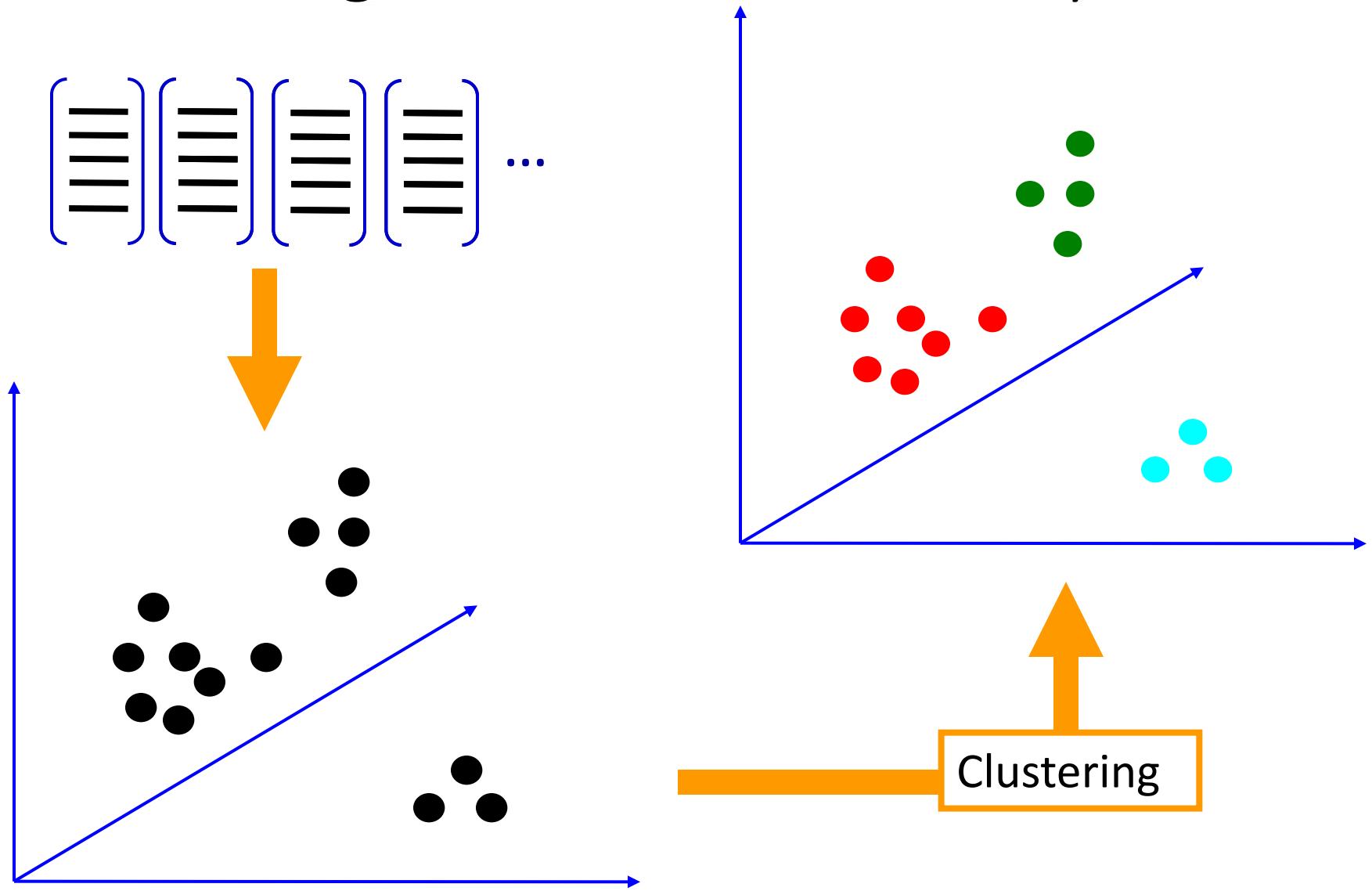


Extracted descriptors
from the training set

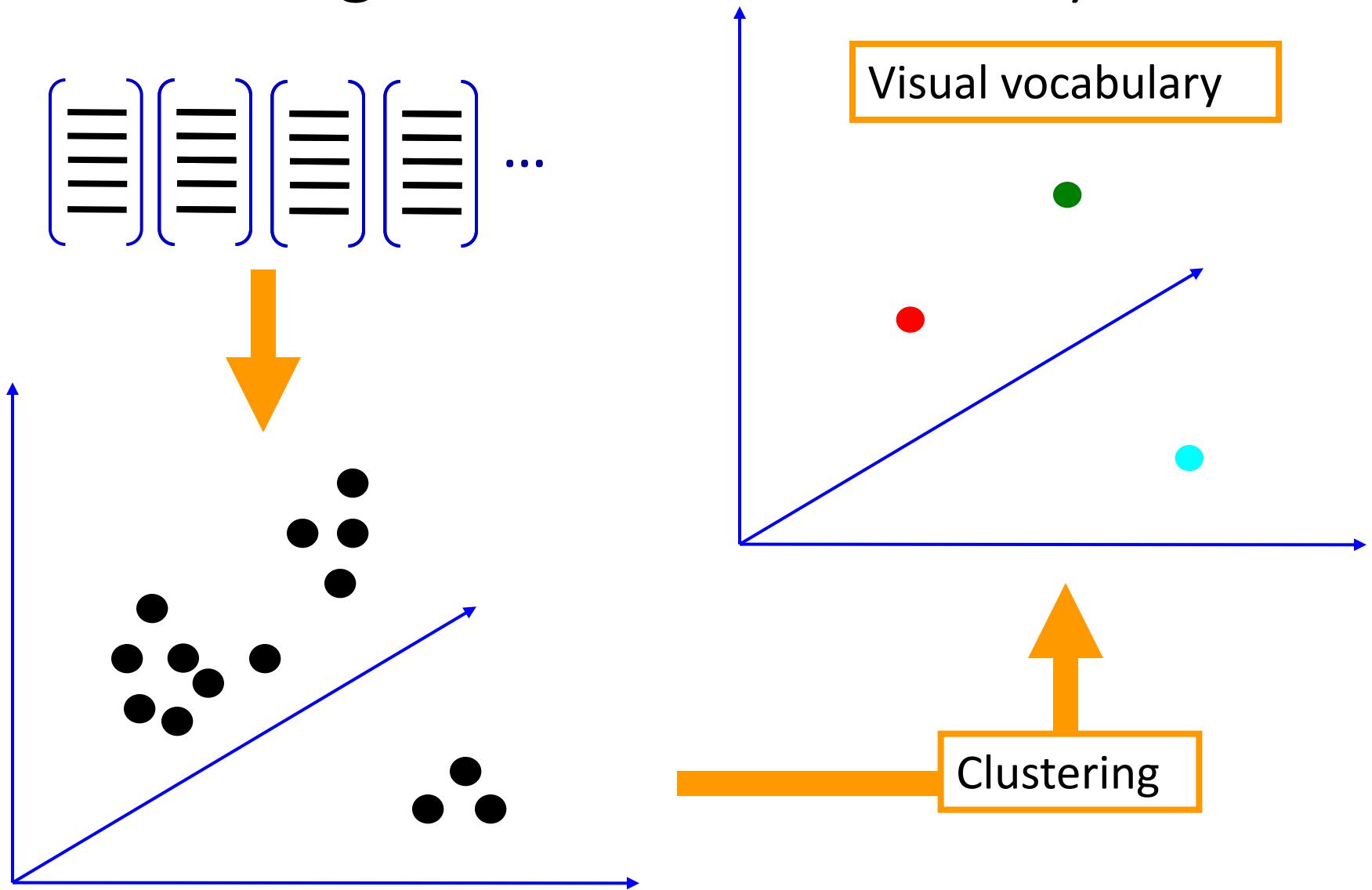
2. Learning the visual vocabulary



2. Learning the visual vocabulary



2. Learning the visual vocabulary



Review: K-means clustering

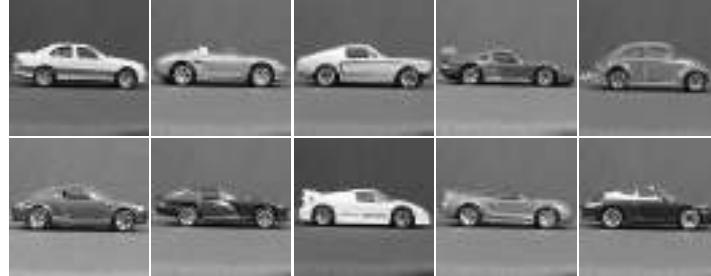
- Want to minimize sum of squared Euclidean distances between features \mathbf{x}_i and their nearest cluster centers \mathbf{m}_k

$$D(X, M) = \sum_{\text{cluster } k} \sum_{\substack{\text{point } i \text{ in} \\ \text{cluster } k}} (\mathbf{x}_i - \mathbf{m}_k)^2$$

Algorithm:

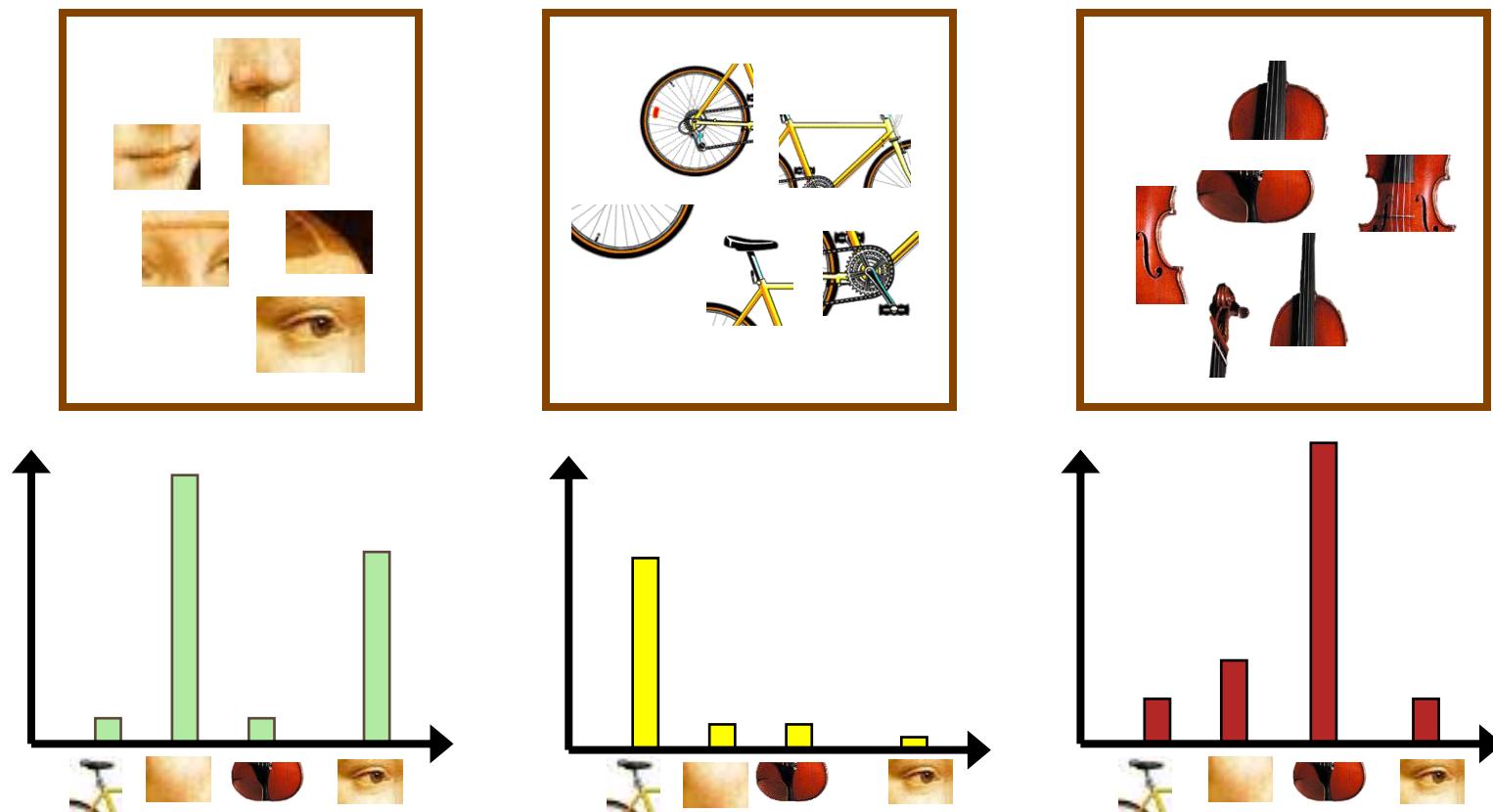
- Randomly initialize K cluster centers
- Iterate until convergence:
 - Assign each feature to the nearest center
 - Recompute each cluster center as the mean of all features assigned to it

Example visual vocabulary



Bag-of-features steps

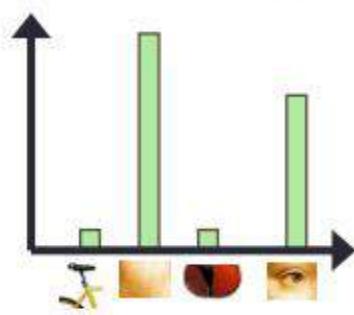
1. Extract local features
2. Learn “visual vocabulary”
3. **Quantize local features using visual vocabulary**
4. Represent images by frequencies of “visual words”



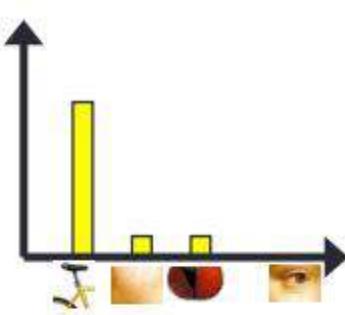
Comparing bags of words

- Rank frames by normalized scalar product between their (possibly weighted) occurrence counts--*nearest neighbor* search for similar images.

[1 8 1 4]



[5 1 1 0]



\vec{d}_j \vec{q}

$$sim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \|q\|}$$

$$= \frac{\sum_{i=1}^V d_j(i) * q(i)}{\sqrt{\sum_{i=1}^V d_j(i)^2} * \sqrt{\sum_{i=1}^V q(i)^2}}$$

for vocabulary of V words

Image categorization with bag of words

Training

1. Extract keypoints and descriptors for all training images
2. Cluster descriptors
3. Quantize descriptors using cluster centers to get “visual words”
4. Represent each image by normalized counts of “visual words”
5. Train classifier on labeled examples using histogram values as features

Image categorization with bag of words

Training

1. Extract keypoints and descriptors for all training images
2. Cluster descriptors
3. Quantize descriptors using cluster centers to get “visual words”
4. Represent each image by normalized counts of “visual words”
5. Train classifier on labeled examples using histogram values as features

Testing

1. Extract keypoints/descriptors and quantize into visual words
2. Compute visual word histogram
3. Compute label or confidence using classifier

Object classification with bag of words

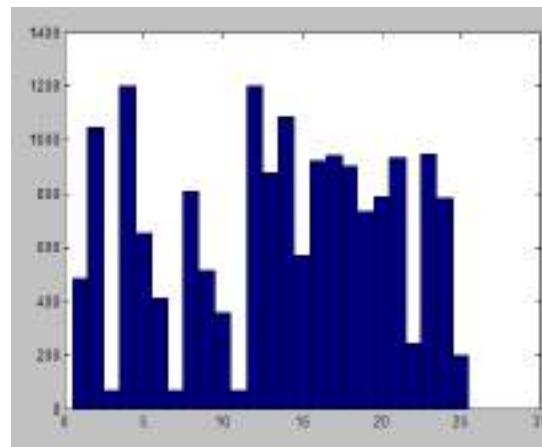
- Performance on Caltech 101 dataset with linear SVM on bag-of-word vectors:



<i>True classes →</i>	<i>faces (frontal)</i>	<i>airplanes (side)</i>	<i>cars (rear)</i>	<i>cars (side)</i>	<i>motorbikes (side)</i>
<i>faces(frontal)</i>	94	0.4	0.7	0	1.4
<i>airplanes (side)</i>	1.5	96.3	0.2	0.1	2.7
<i>cars (rear)</i>	1.9	0.5	97.7	0	0.9
<i>cars(side)</i>	1.7	1.9	0.5	99.6	2.3
<i>motorbikes (side)</i>	0.9	0.9	0.9	0.3	92.7

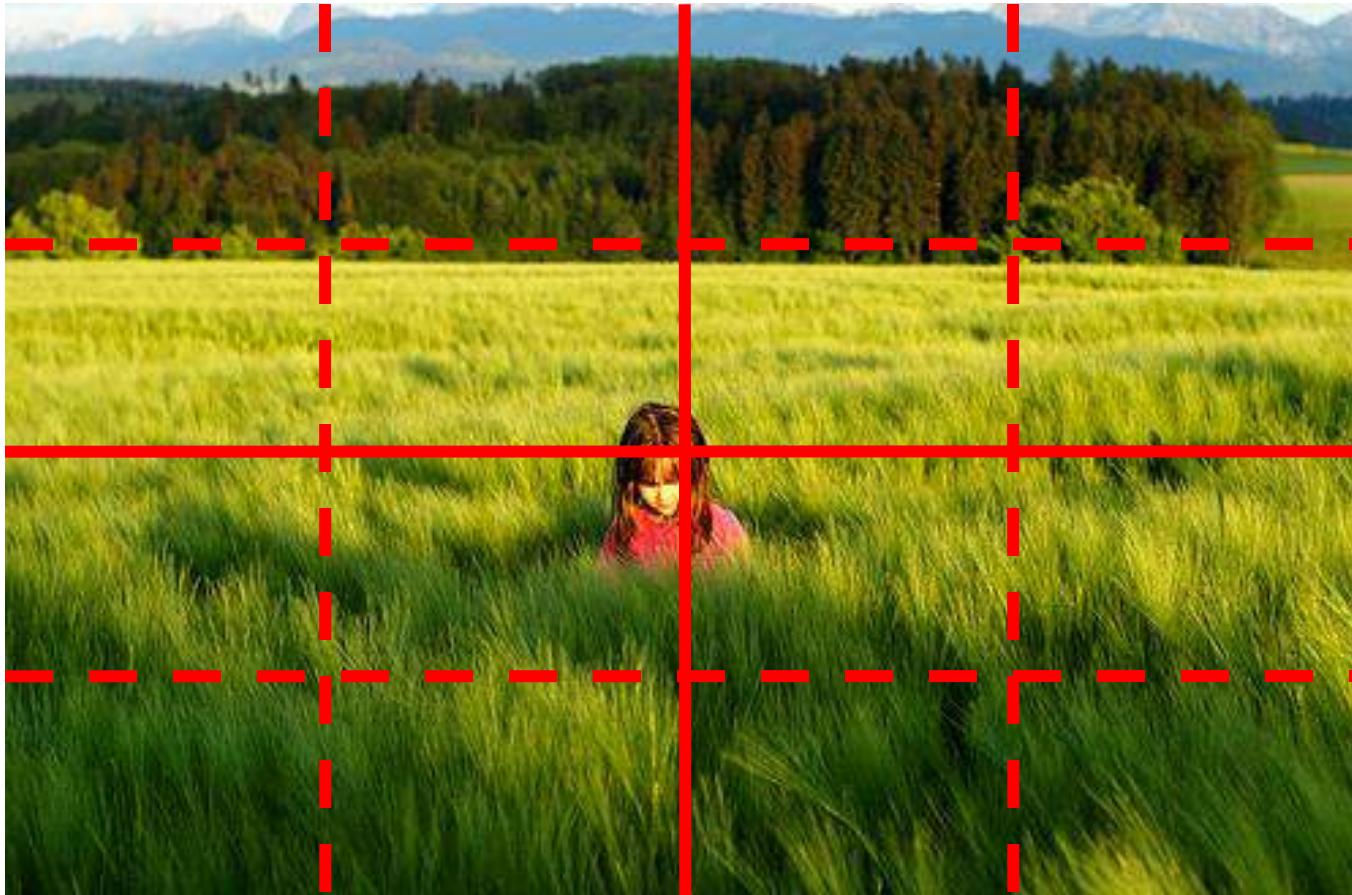
[Csurka et al., '04]

But what about spatial layout?



All of these images have the same color histogram

Spatial pyramid



Compute histogram in each spatial bin

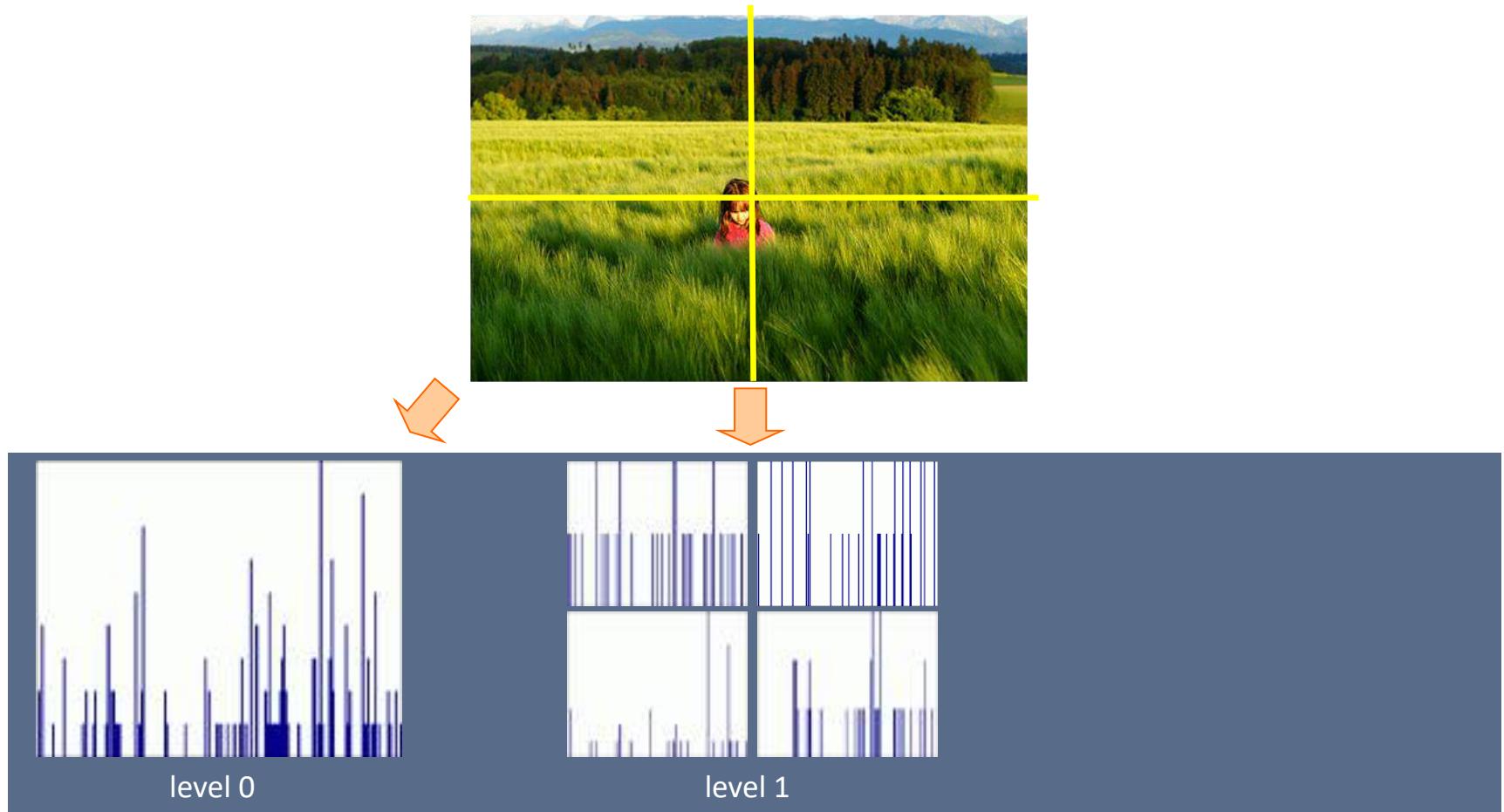
Spatial pyramids



[Lazebnik, Schmid & Ponce \(CVPR 2006\)](#) –

Beyond bags of features: spatial pyramid matching for recognizing natural scene categories

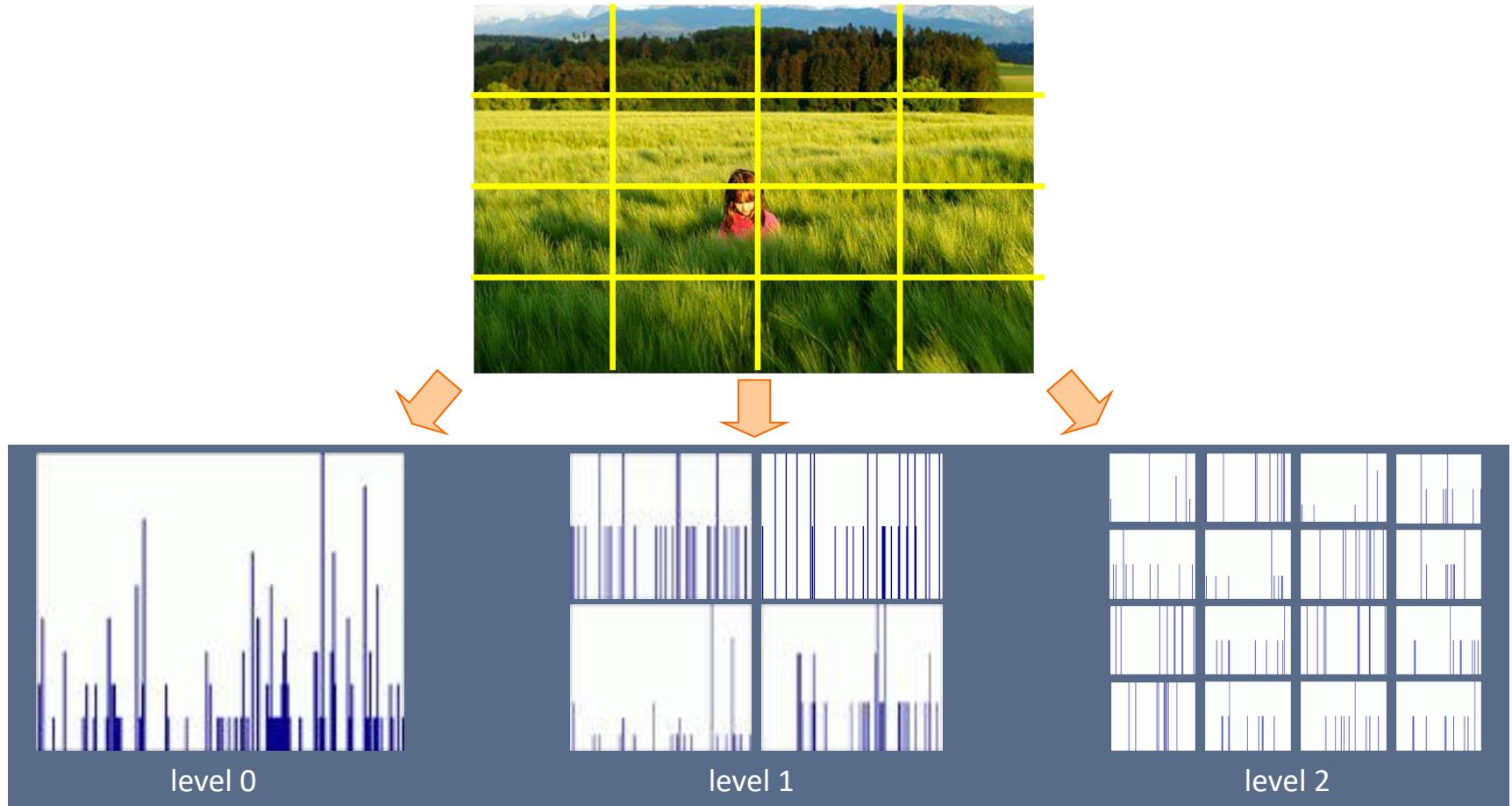
Spatial pyramids



[Lazebnik, Schmid & Ponce \(CVPR 2006\)](#) –

Beyond bags of features: spatial pyramid matching for recognizing natural scene categories

Spatial pyramids



[Lazebnik, Schmid & Ponce \(CVPR 2006\) –](#)

Beyond bags of features: spatial pyramid matching for recognizing natural scene categories

Spatial pyramids: Scene classification results

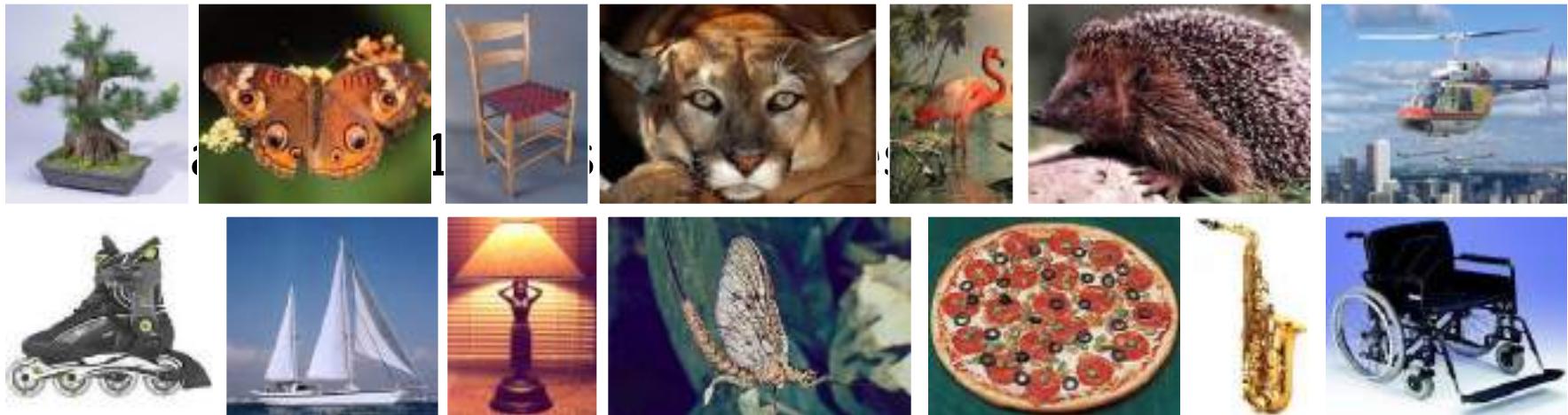


Level	Weak features (vocabulary size: 16)		Strong features (vocabulary size: 200)	
	Single-level	Pyramid	Single-level	Pyramid
0 (1×1)	45.3 ± 0.5		72.2 ± 0.6	
1 (2×2)	53.6 ± 0.3	56.2 ± 0.6	77.9 ± 0.6	79.0 ± 0.5
2 (4×4)	61.7 ± 0.6	64.7 ± 0.7	79.4 ± 0.3	81.1 ± 0.3
3 (8×8)	63.3 ± 0.8	66.8 ± 0.6	77.2 ± 0.4	80.7 ± 0.3

[Lazebnik, Schmid & Ponce \(CVPR 2006\) –](#)

Beyond bags of features: spatial pyramid matching for recognizing natural scene categories

Spatial pyramids: Caltech101 classification results

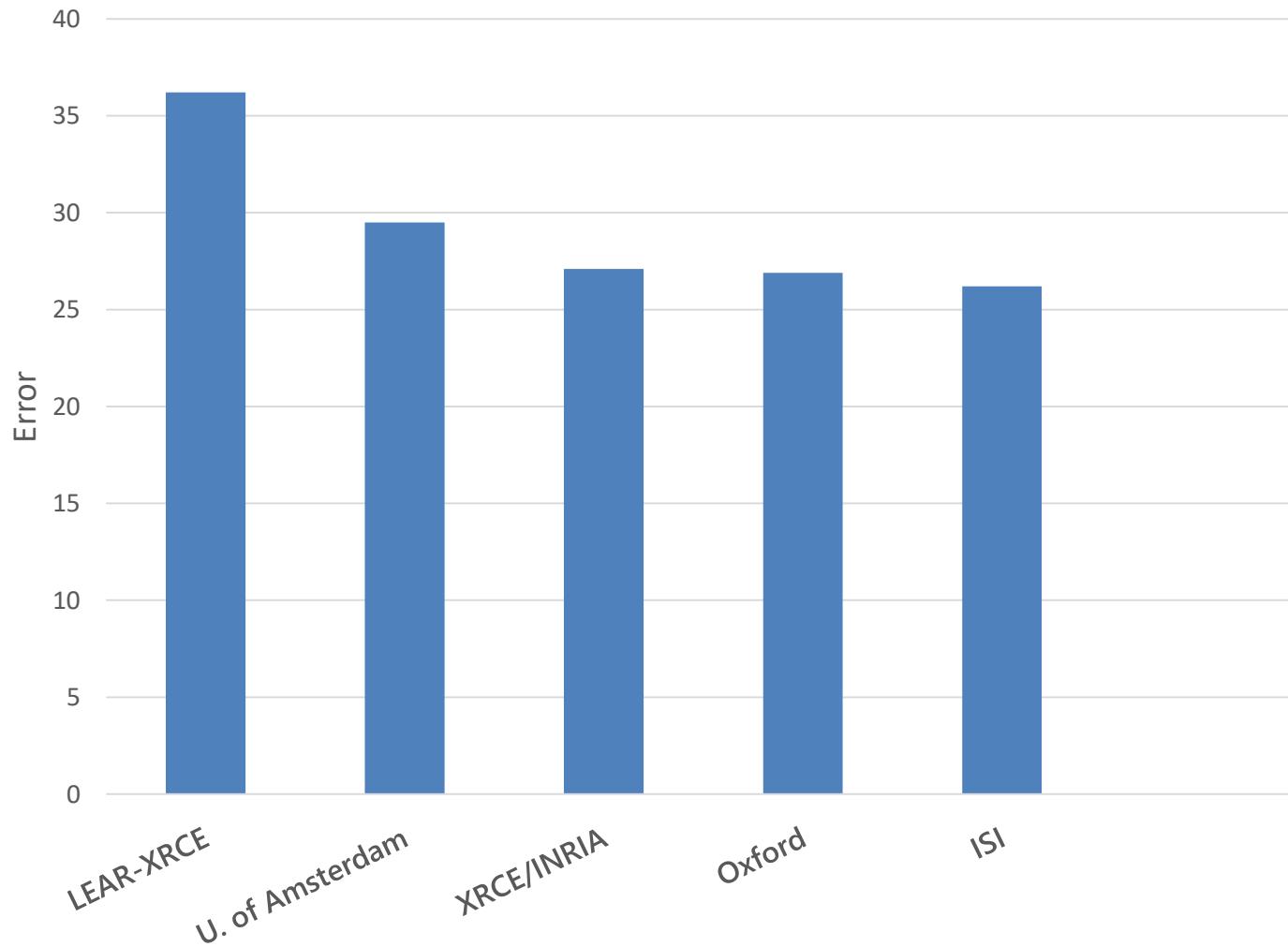


	Weak features (16)		Strong features (200)	
Level	Single-level	Pyramid	Single-level	Pyramid
0	15.5 ± 0.9		41.2 ± 1.2	
1	31.4 ± 1.2	32.8 ± 1.3	55.9 ± 0.9	57.0 ± 0.8
2	47.2 ± 1.1	49.3 ± 1.4	63.6 ± 0.9	64.6 ± 0.8
3	52.2 ± 0.8	54.0 ± 1.1	60.3 ± 0.9	64.6 ± 0.7

[Lazebnik, Schmid & Ponce \(CVPR 2006\)](#) –

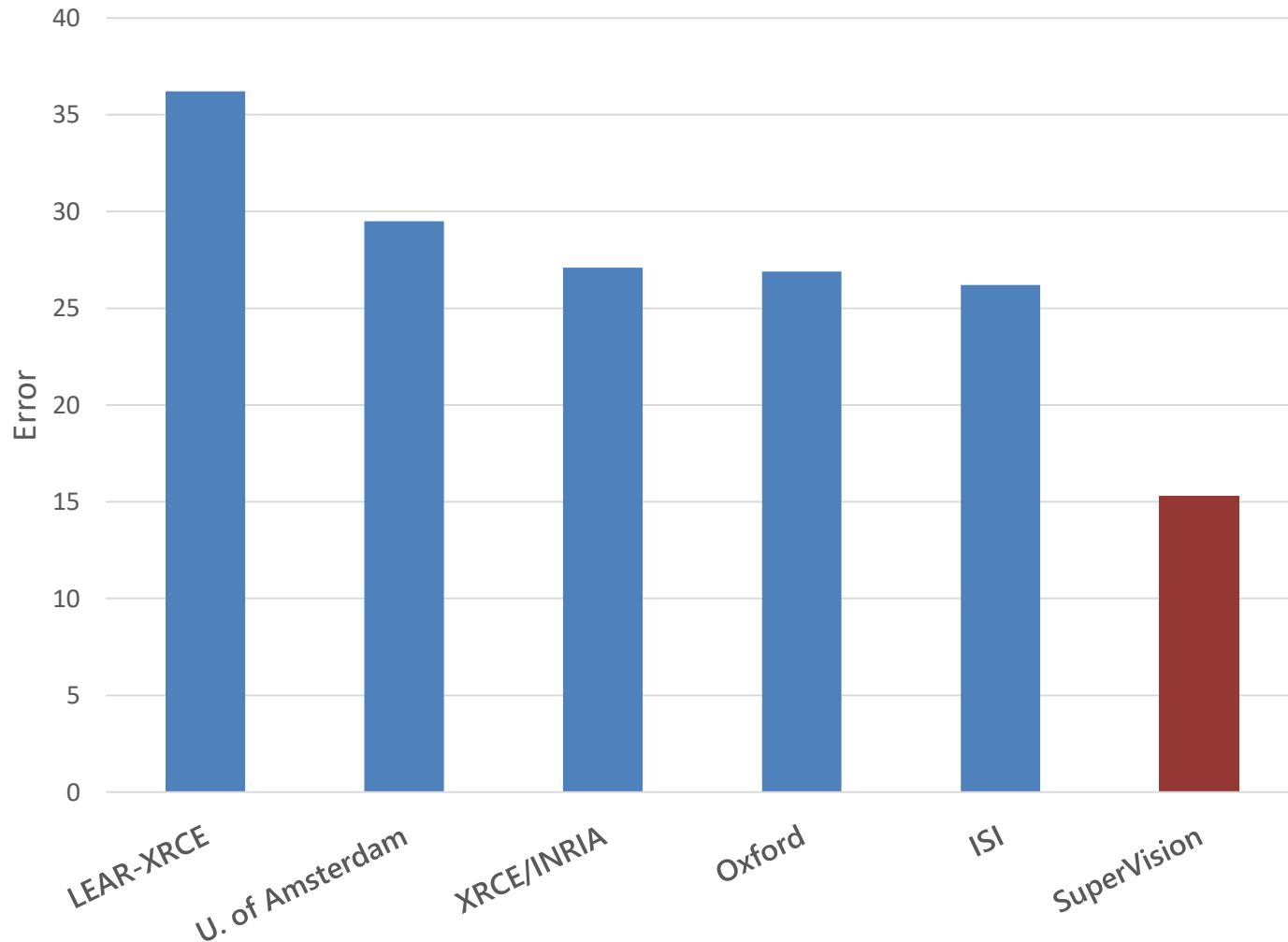
Beyond bags of features: spatial pyramid matching for recognizing natural scene categories

ImageNet 1K



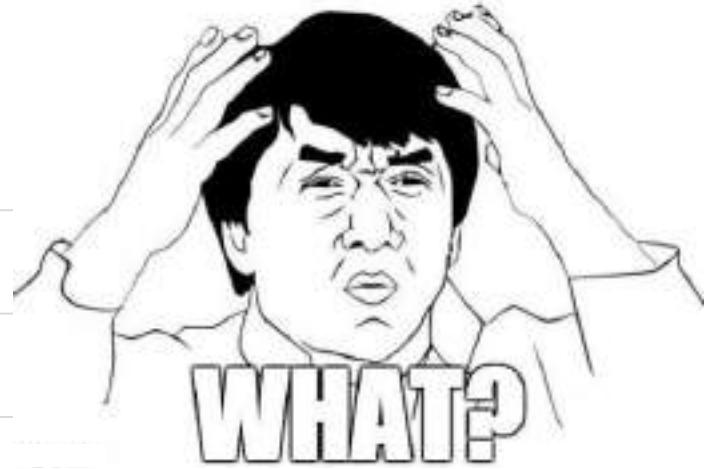
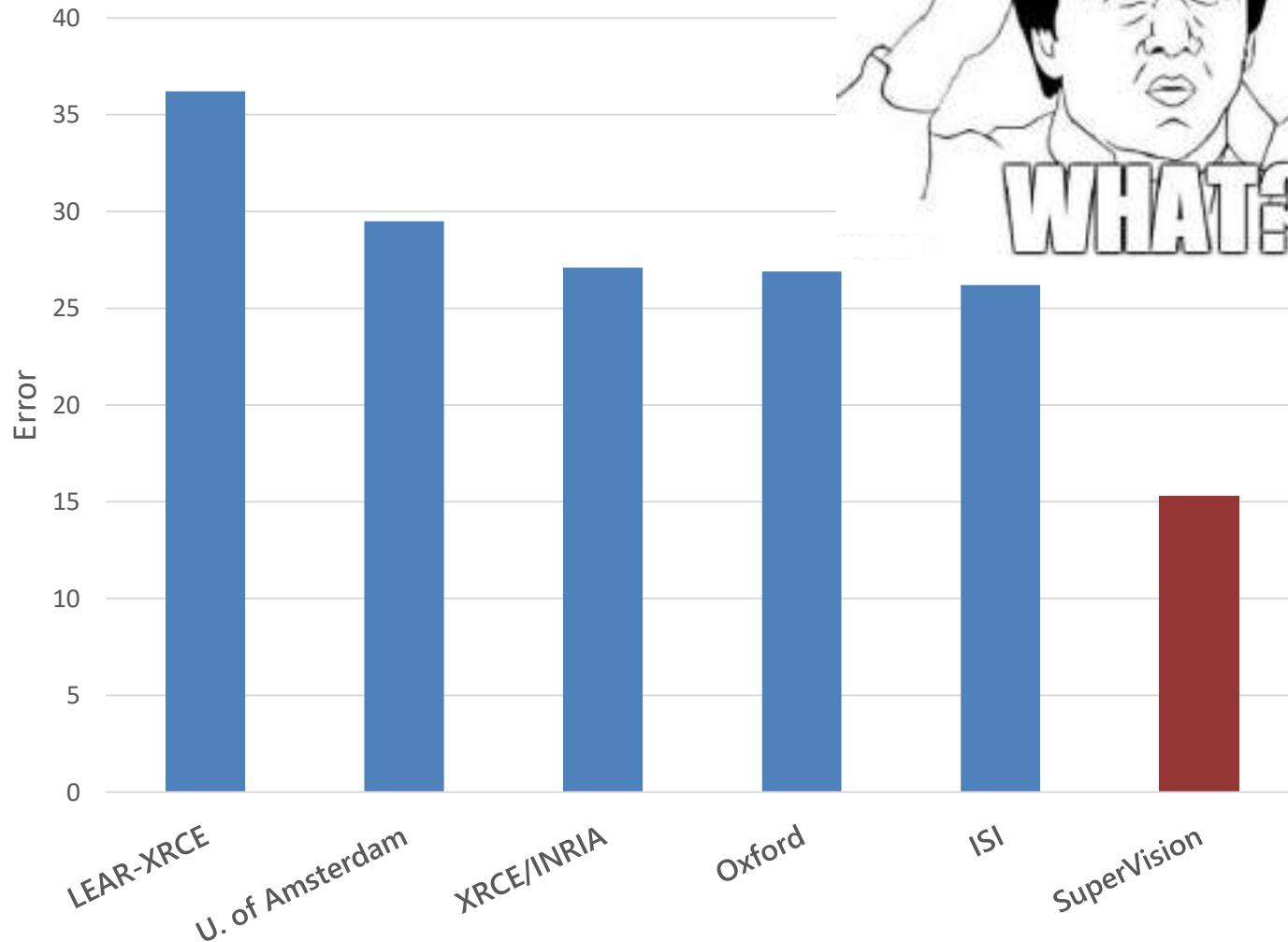
ImageNet 1K

(Fall 2012)



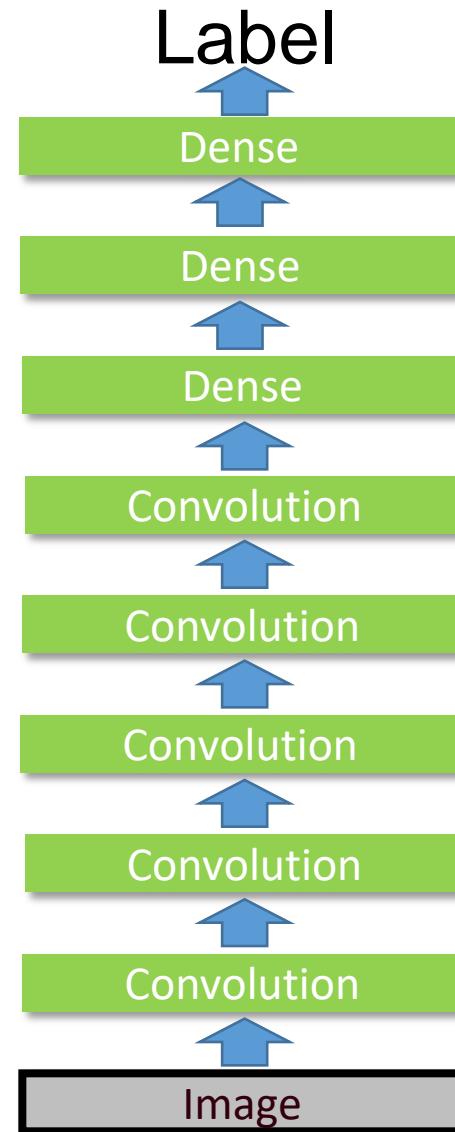
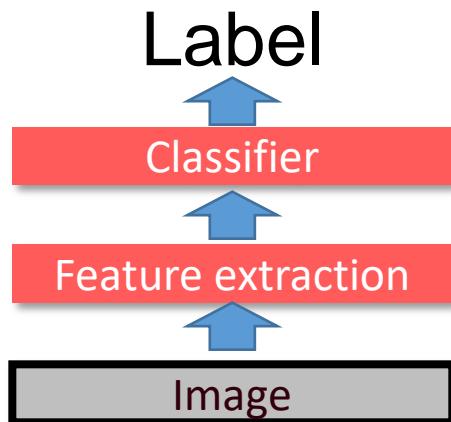
ImageNet 1K

(Fall 2012)

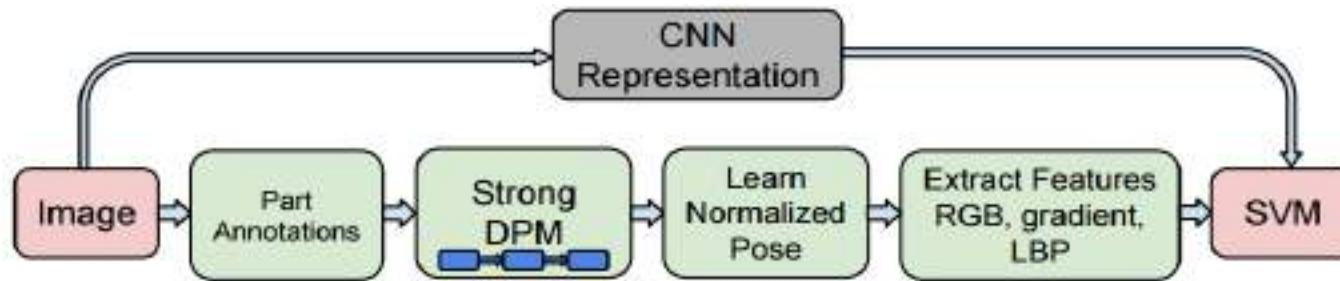


Shallow vs. deep learning

- Engineered vs. learned features

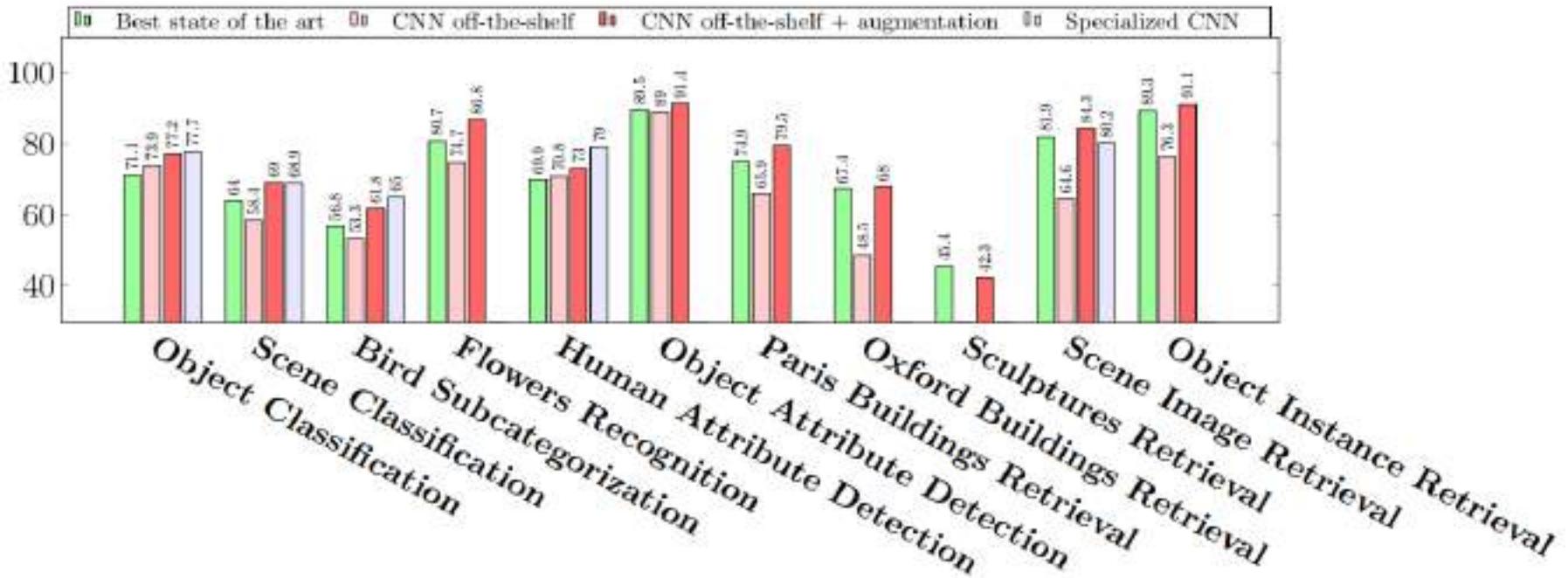
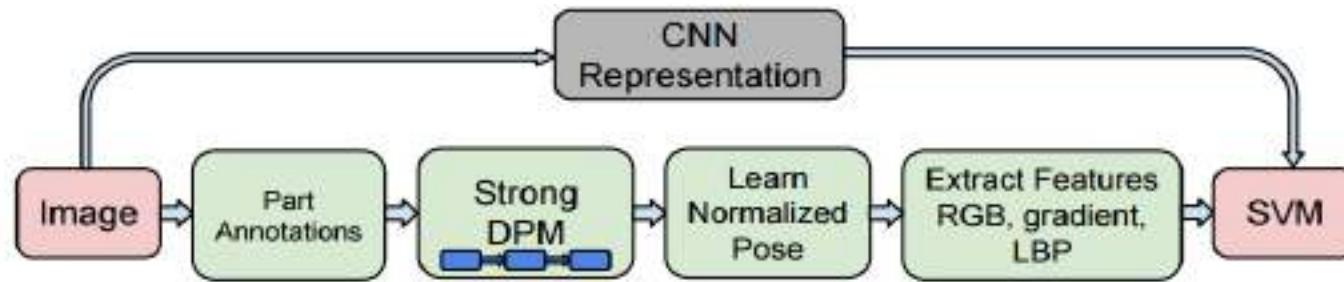


Convolutional activation features



CNN Features off-the-shelf: an Astounding Baseline for Recognition
[[Razavian et al. 2014](#)]

Convolutional activation features



CNN Features off-the-shelf: an Astounding Baseline for Recognition
[Razavian et al. 2014]

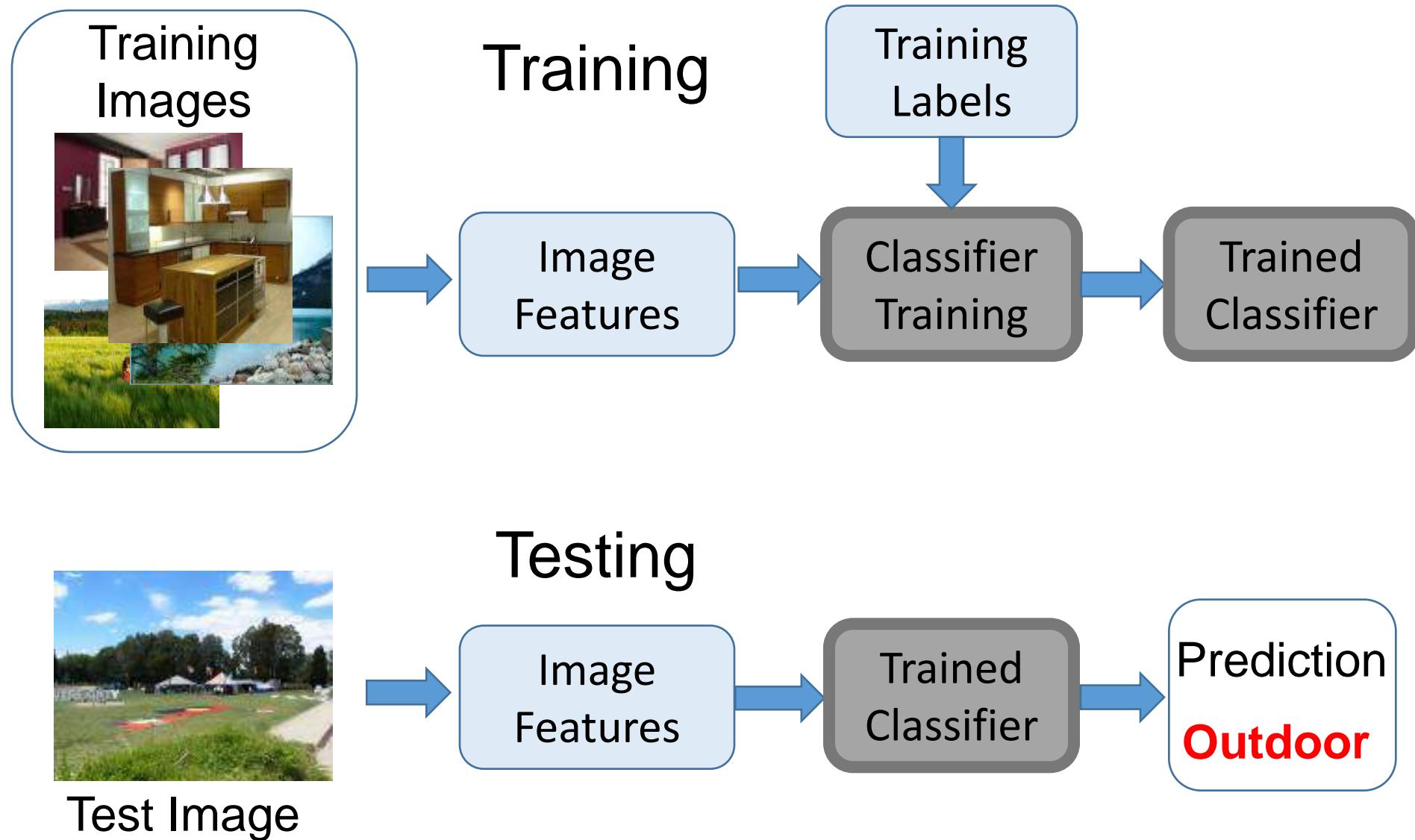
Things to remember

- Visual categorization help transfer knowledge
- Image features
 - Color, gradients, textures, motion
 - Histogram, SIFT, Descriptors
 - Bag-of-visual-words
 - CNN Feature
- Image/region categorization

Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Leibe
 - And many more

Next Lecture - Classifiers



Computer Vision

Introduction to Classifiers

Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**



Previous Class

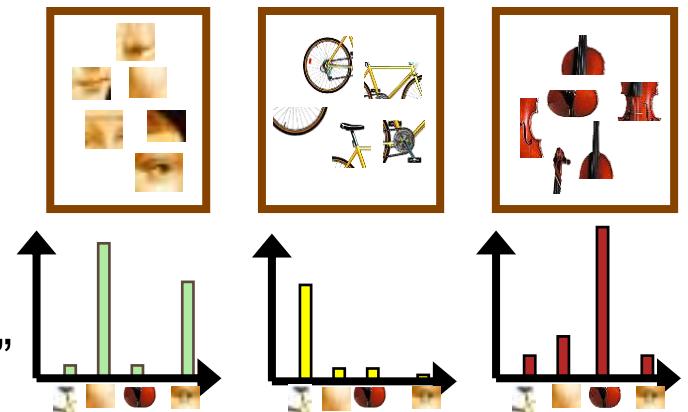
Image features and categorization

Choosing right features
Object, Scene, Action, etc.

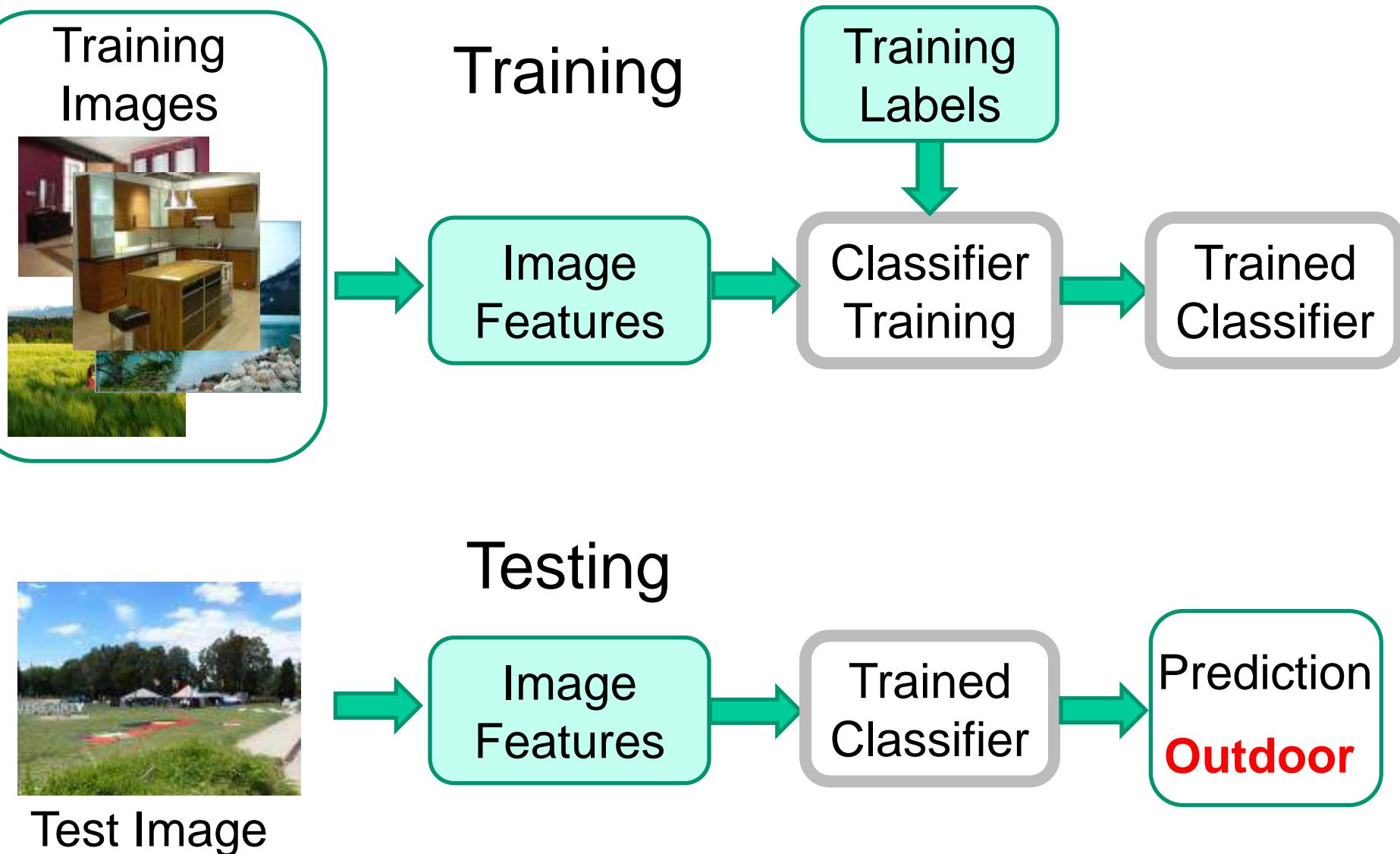


Bag-of-visual-words

Extract local features
Learn “visual vocabulary”
Quantize features using visual vocabulary
Represent by frequencies of “visual words”



Today's Class



Today's Class

Nearest Neighbor Classifier

K - Nearest Neighbor Classifier

Linear Classifier

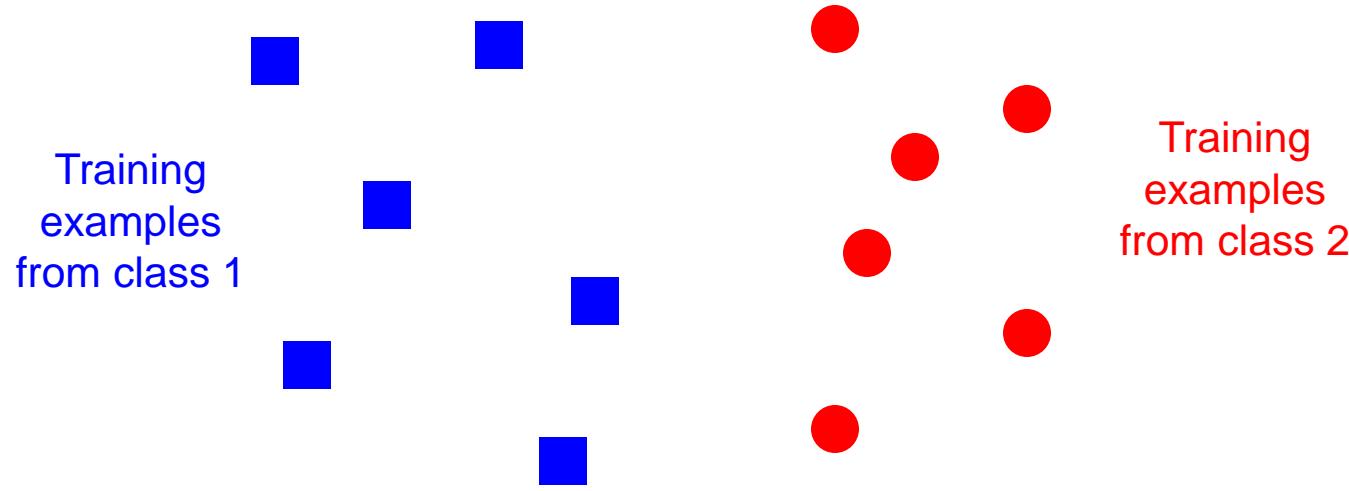
Support Vector Machine

Non-linear SVM

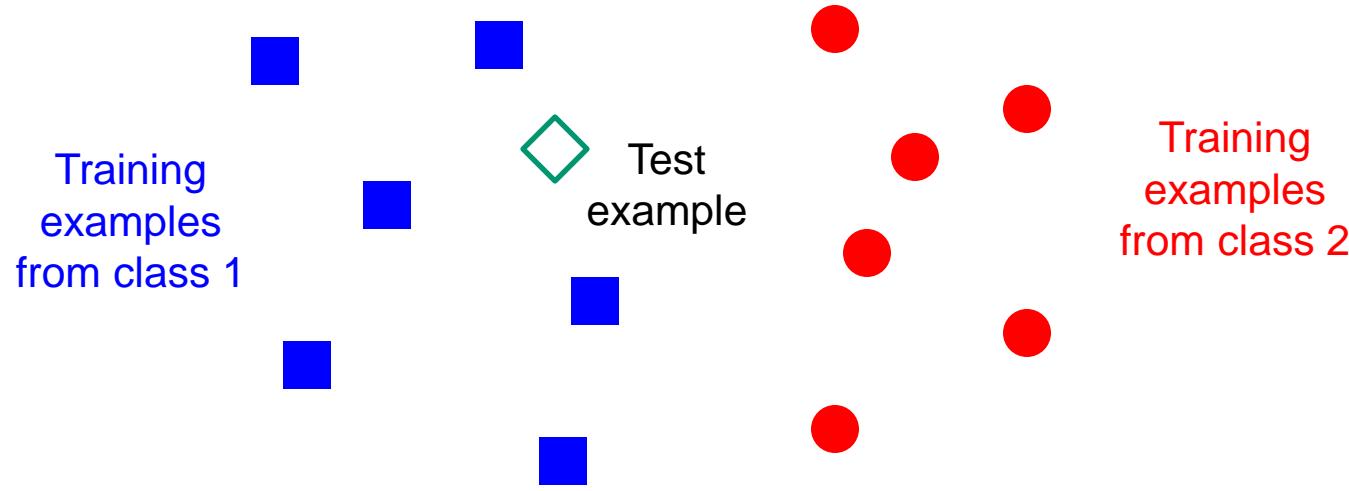
Multi-class SVM

Softmax Classifier

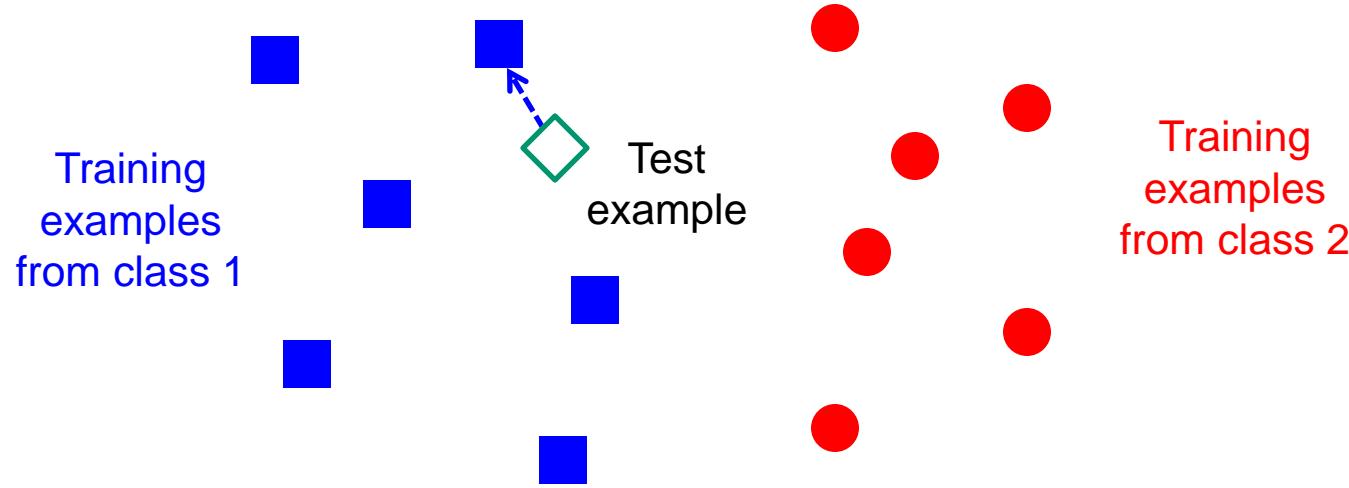
Classifiers: Nearest neighbor



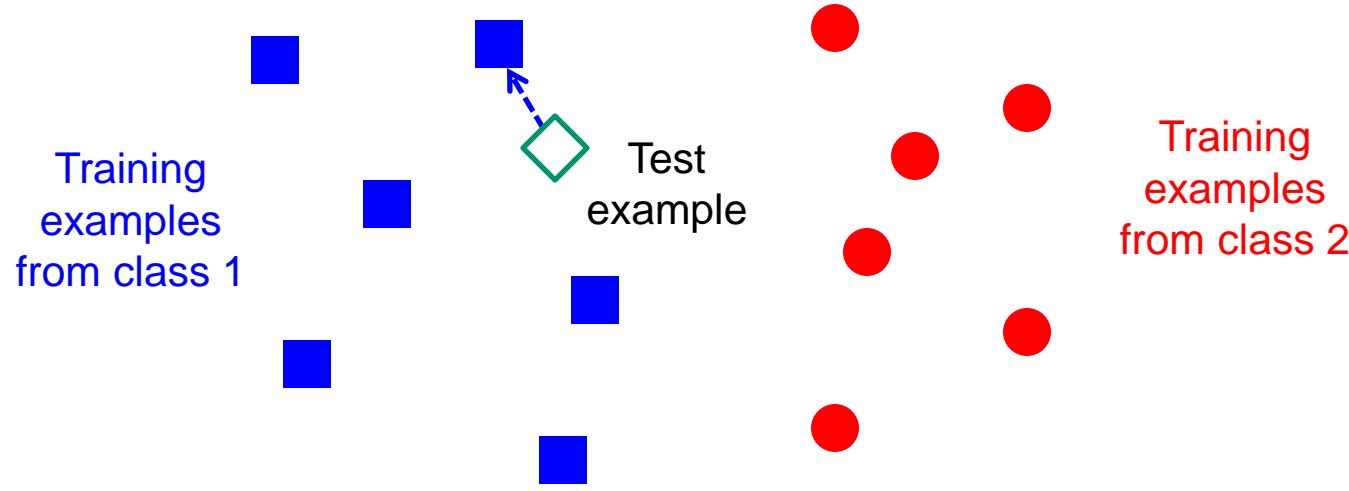
Classifiers: Nearest neighbor



Classifiers: Nearest neighbor



Classifiers: Nearest neighbor



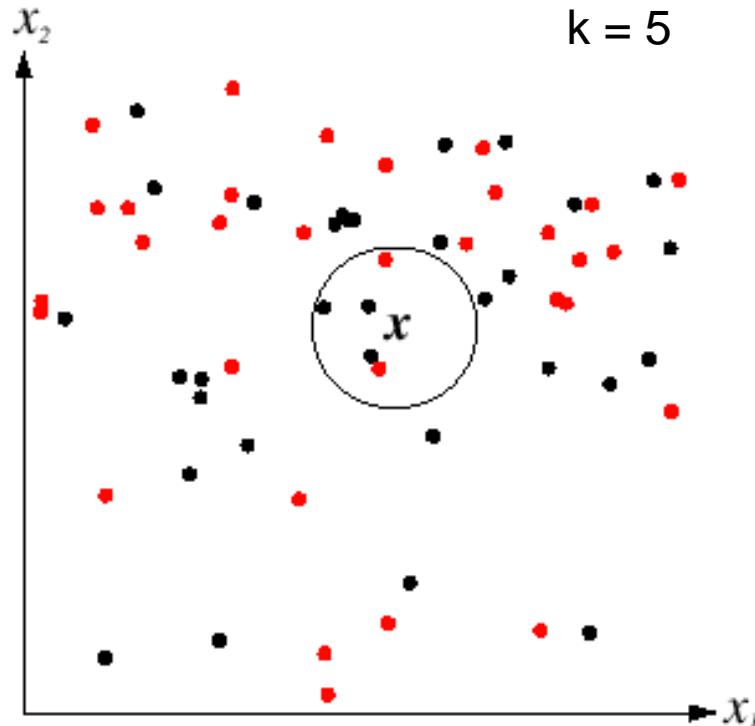
$f(x)$ = label of the training example nearest to x

All we need is a distance function for our inputs

No training required!

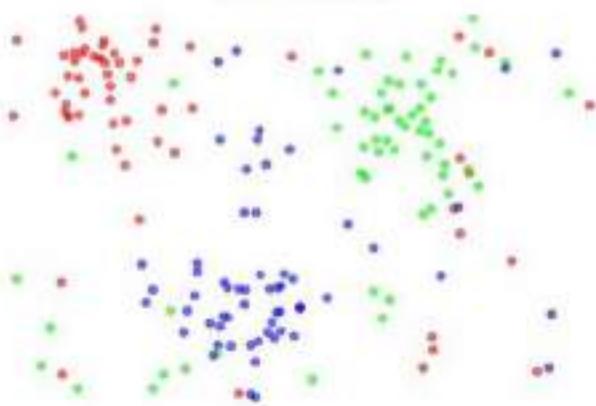
K-nearest neighbor classifier

- For a new point, find the k closest points from training data
- Vote for class label with labels of the k points

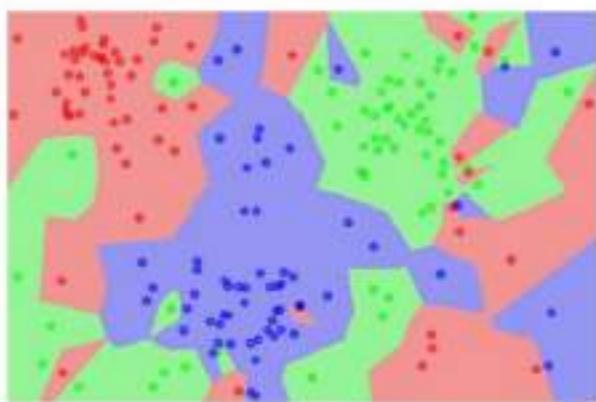


K-nearest neighbor classifier

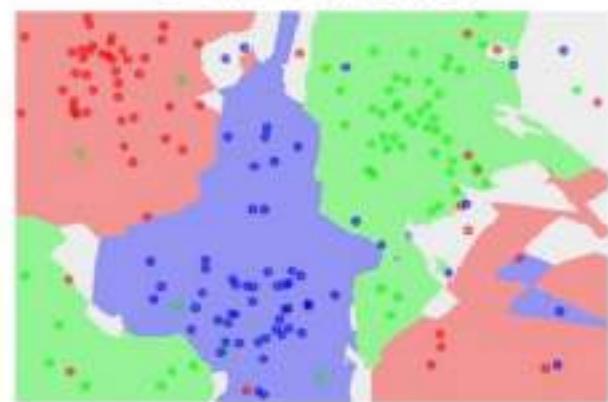
the data



NN classifier

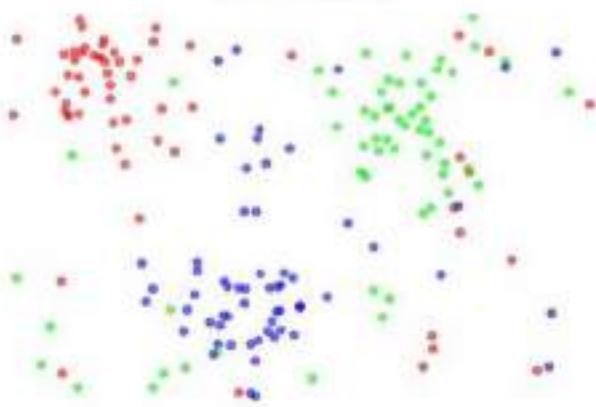


5-NN classifier

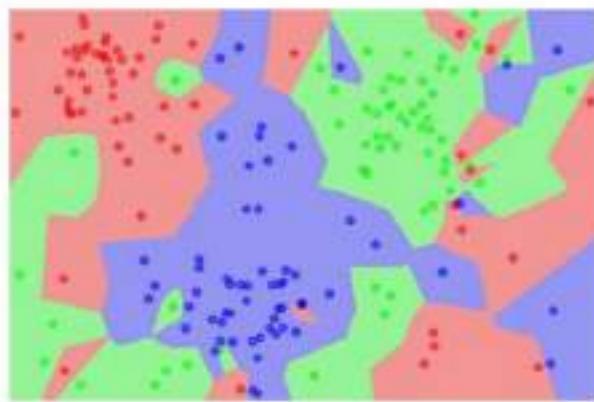


K-nearest neighbor classifier

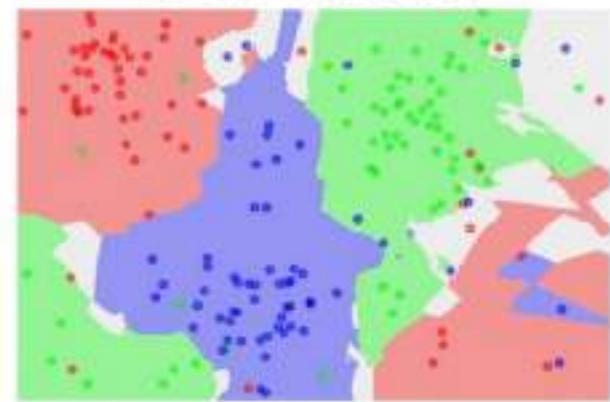
the data



NN classifier

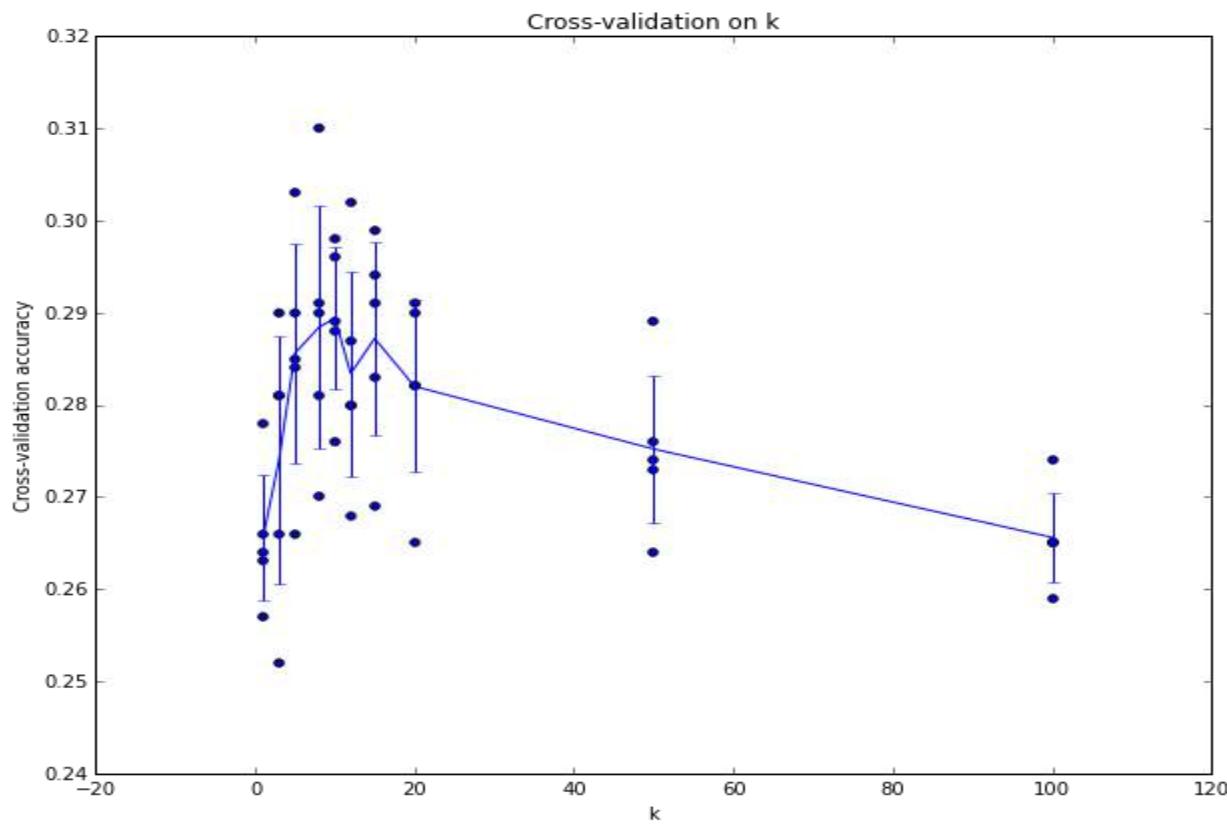


5-NN classifier



Which classifier is more robust to *outliers*?

Choice of K in KNN classifier



Example of a 5-fold cross-validation run for the parameter k . Note that in this particular case, the cross-validation suggests that a value of about $k = 7$ works best on this particular dataset (corresponding to the peak in the plot).

Classifiers: K-Nearest neighbor

“Non-parametric” classifier: the entire training set is essentially the model parameters.

Classifiers: K-Nearest neighbor

“Non-parametric” classifier: the entire training set is essentially the model parameters.

Pros:

- **Very fast at training time**
- **Flexible:** all it requires is a way to compute similarity or distances between pairs of features. Applies to many different kinds of features.
- Works with **any number of classes**.
- Works well in practice for large datasets (but see cons)

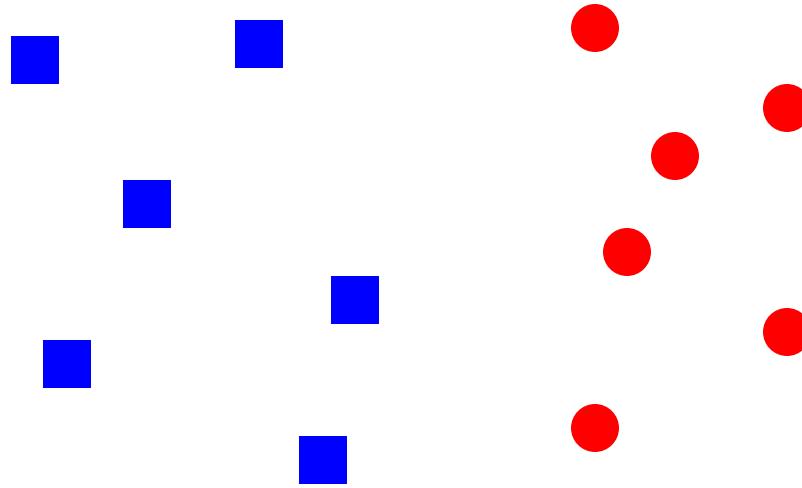
Classifiers: K-Nearest neighbor

“Non-parametric” classifier: the entire training set is essentially the model parameters.

Cons:

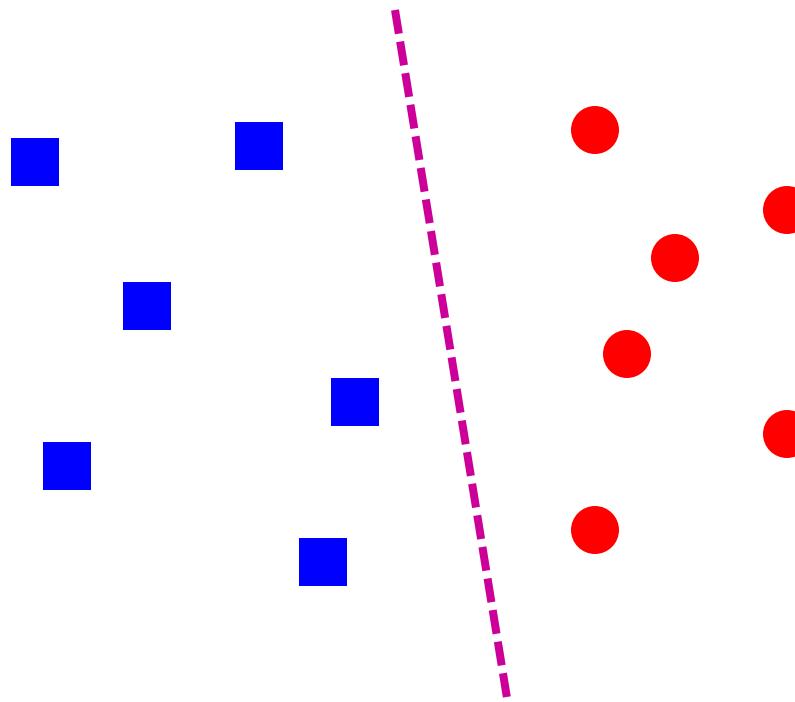
- The classifier must **remember all of the training data** and store it for future comparisons with the test data.
- This is **space inefficient** because datasets may easily be gigabytes in size.
- **Slow at test time** (need to compute distances between test example and every training example)
- Optimum value of **K is not known**.

Linear classifiers – 2 class problem



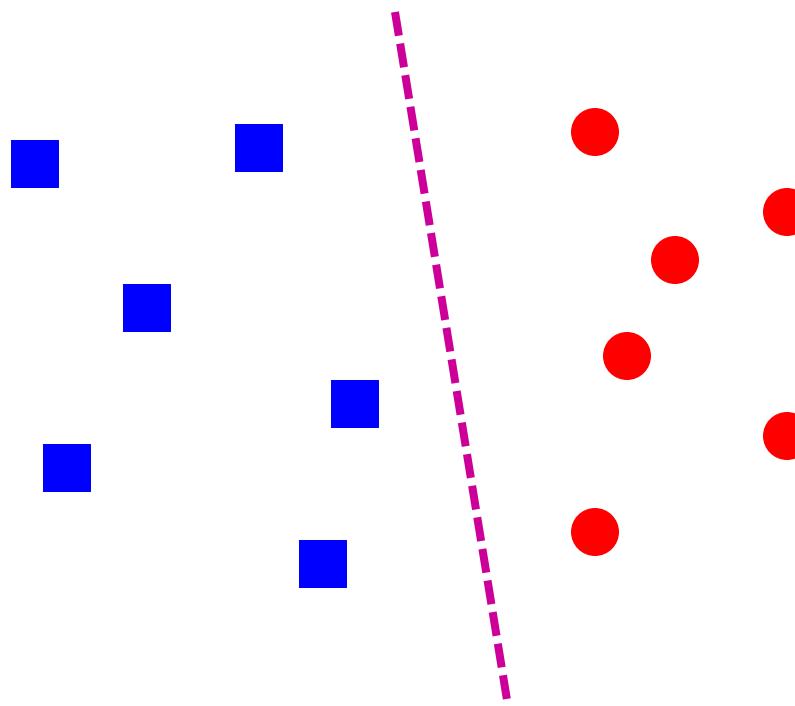
Find a *linear function* to separate the classes:

Linear classifiers – 2 class problem



Find a *linear function* to separate the classes:

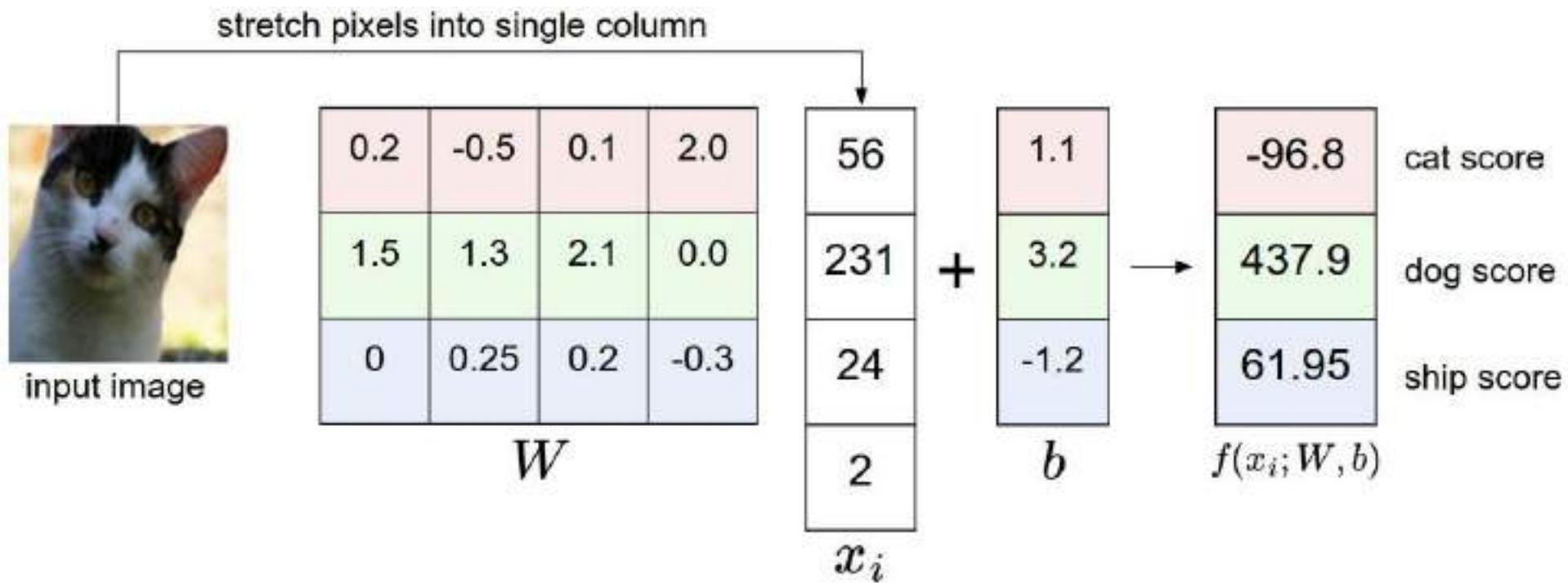
Linear classifiers – 2 class problem



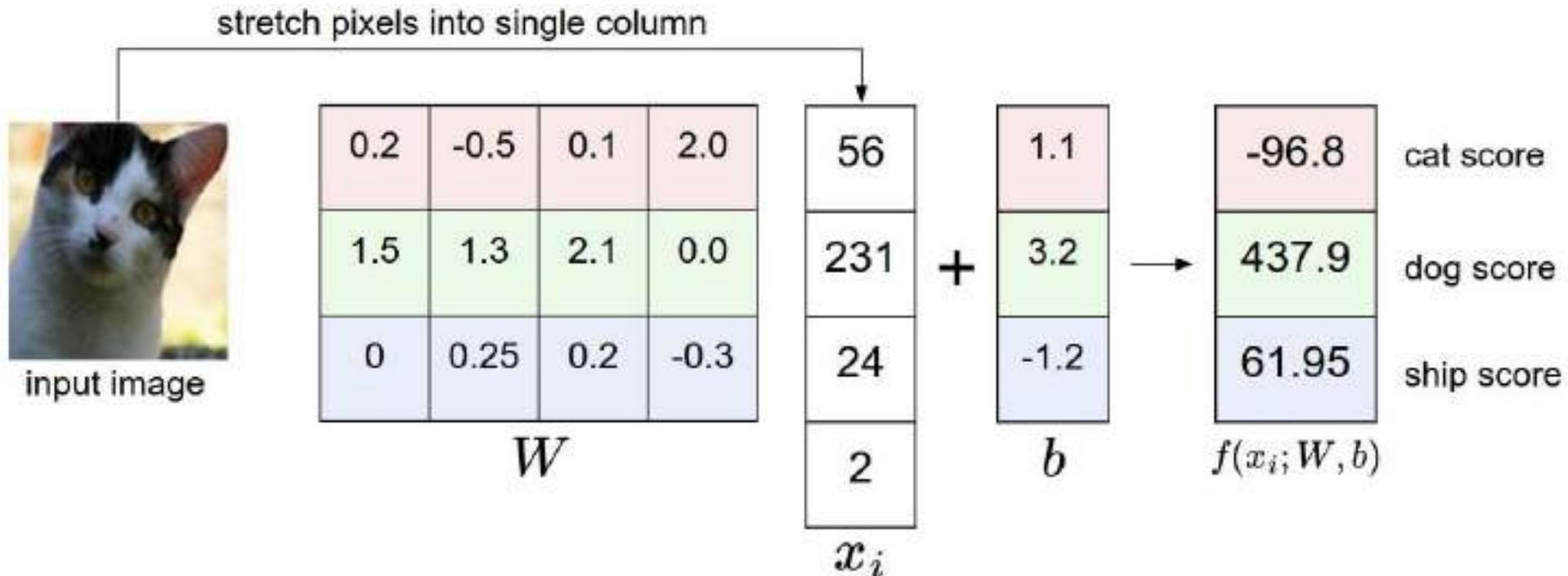
Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

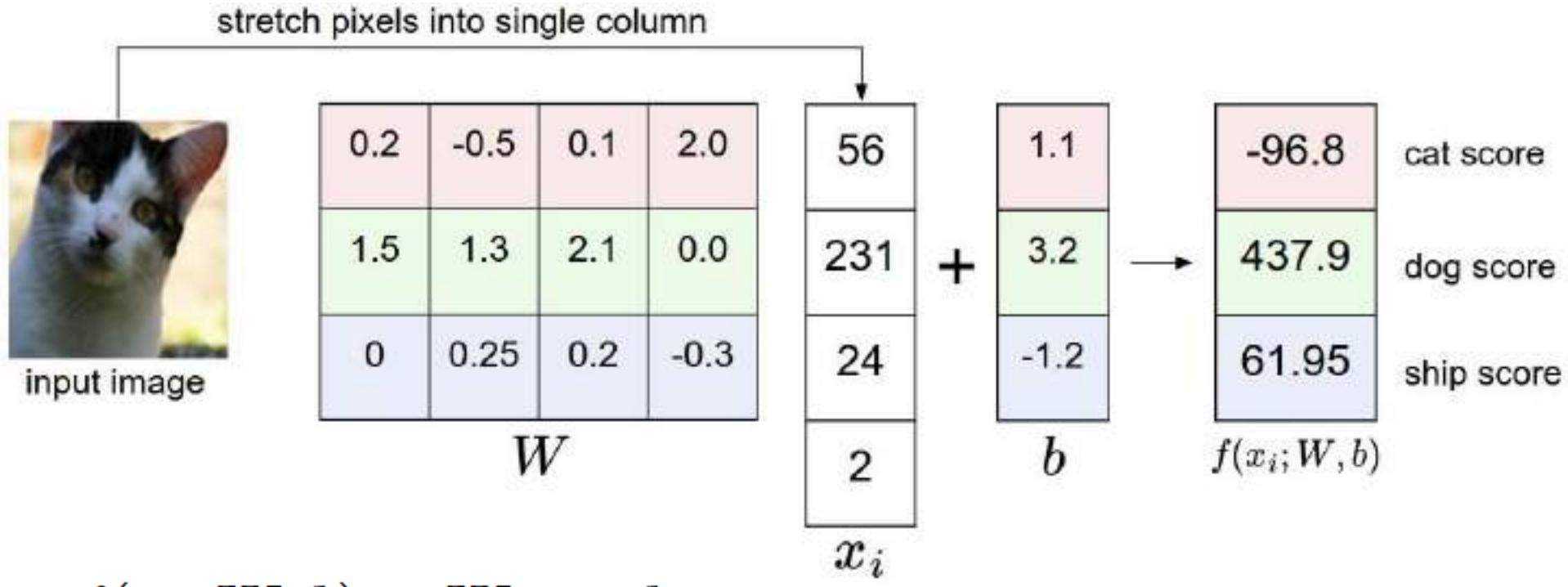
Linear classifiers – more than 2 class



Linear classifiers – more than 2 class



Linear classifiers – more than 2 class



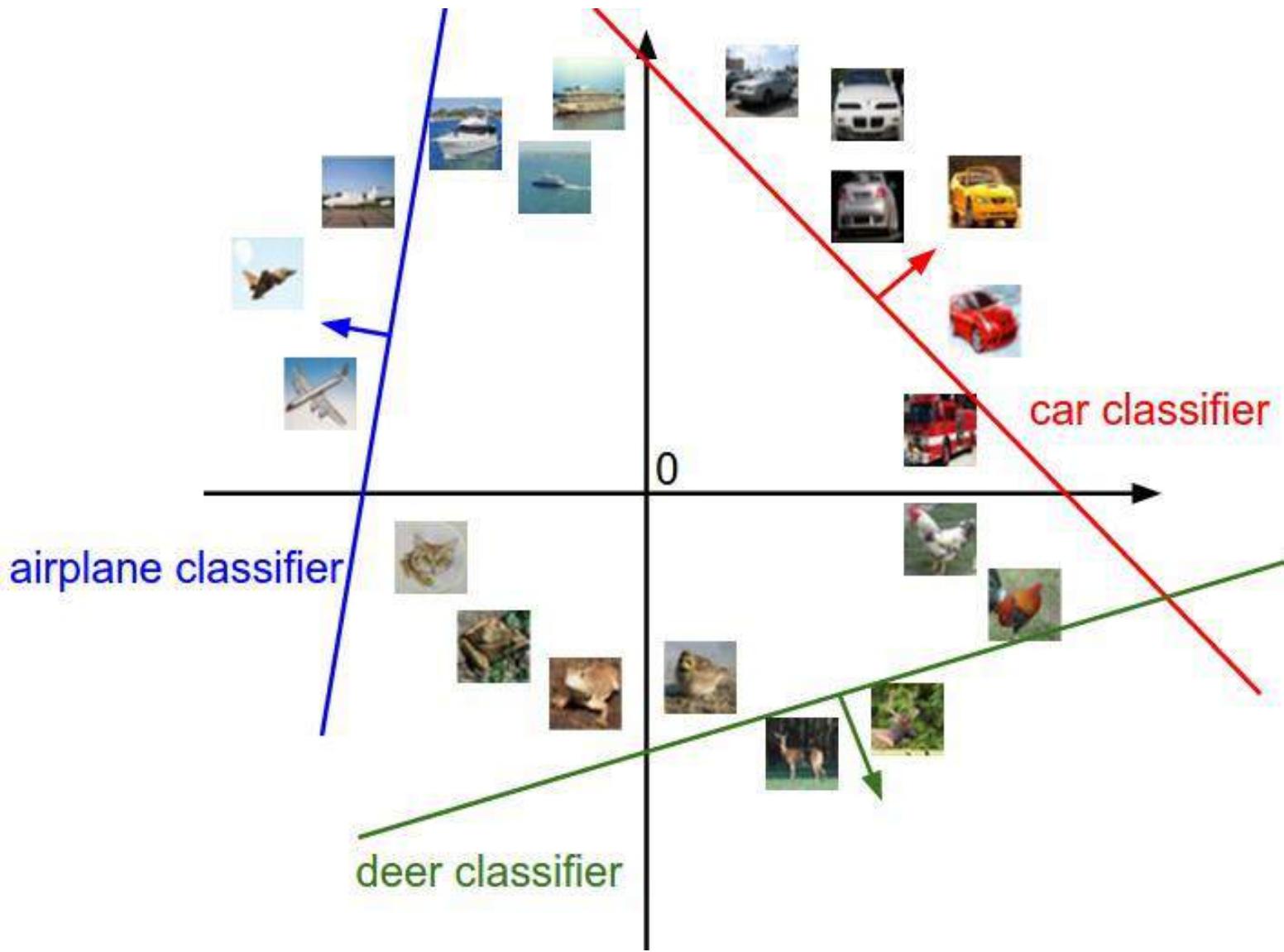
$$f(x_i, W, b) = Wx_i + b$$

Image x_i has all of its pixels flattened out to a single column vector of shape $[D \times 1]$.

Matrix **W** (of size $[K \times D]$), and vector **b** (of size $[K \times 1]$) are the **parameters**.

K is the number of classes.

Analogy of images as high-dimensional points



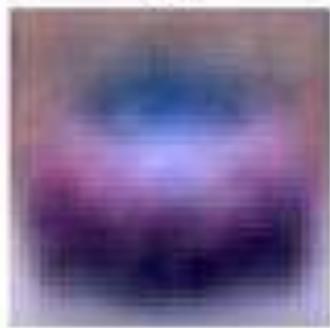
Source: cs231n, <http://cs231n.github.io/linear-classify/>

Interpretation of linear classifiers as template matching

plane



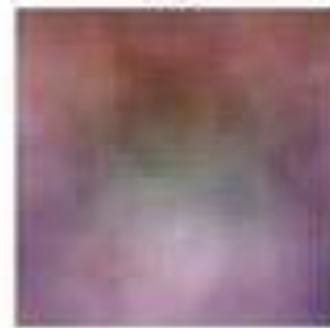
car



bird



cat



deer



dog



frog



horse



ship



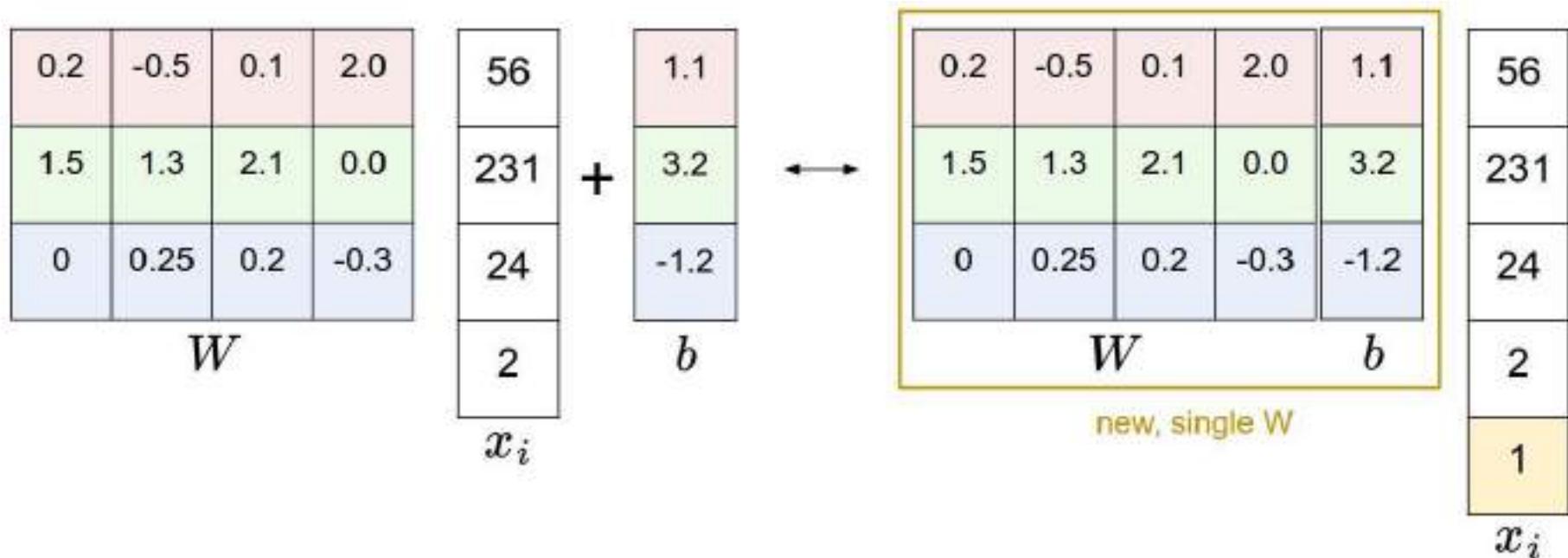
truck



Example learned weights at the end of learning for CIFAR-10. Note that, for example, the ship template contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

Source: cs231n, <http://cs231n.github.io/linear-classify/>

Bias Trick



Linear classifiers

“Parametric” classifier: model defined by a small number of parameters (w , b)

Pros:

- Very fast at test time

Cons:

- Slow at training time: need to estimate the parameters
- Data may not be linearly separable

Nearest neighbor vs. linear classifiers

- NN pros:
 - Simple to implement
 - Decision boundaries not necessarily linear
 - Works for any number of classes
 - *Nonparametric* method

Nearest neighbor vs. linear classifiers

- NN pros:
 - Simple to implement
 - Decision boundaries not necessarily linear
 - Works for any number of classes
 - *Nonparametric* method
- NN cons:
 - Need good distance function
 - Slow at test time, Memory in-efficient

Nearest neighbor vs. linear classifiers

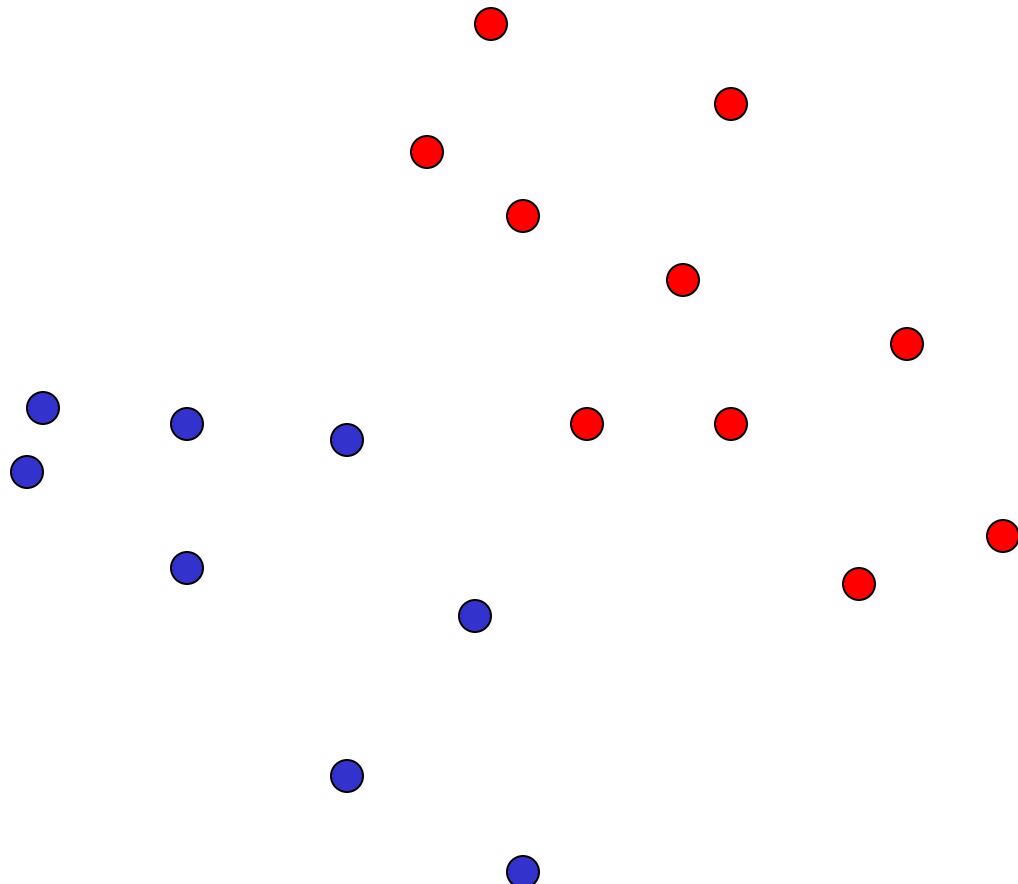
- NN pros:
 - Simple to implement
 - Decision boundaries not necessarily linear
 - Works for any number of classes
 - *Nonparametric* method
- NN cons:
 - Need good distance function
 - Slow at test time, Memory in-efficient
- Linear pros:
 - Low-dimensional *parametric* representation
 - Very fast at test time

Nearest neighbor vs. linear classifiers

- NN pros:
 - Simple to implement
 - Decision boundaries not necessarily linear
 - Works for any number of classes
 - *Nonparametric* method
- NN cons:
 - Need good distance function
 - Slow at test time, Memory in-efficient
- Linear pros:
 - Low-dimensional *parametric* representation
 - Very fast at test time
- Linear cons:
 - How to train the linear function?
 - What if data is not linearly separable?

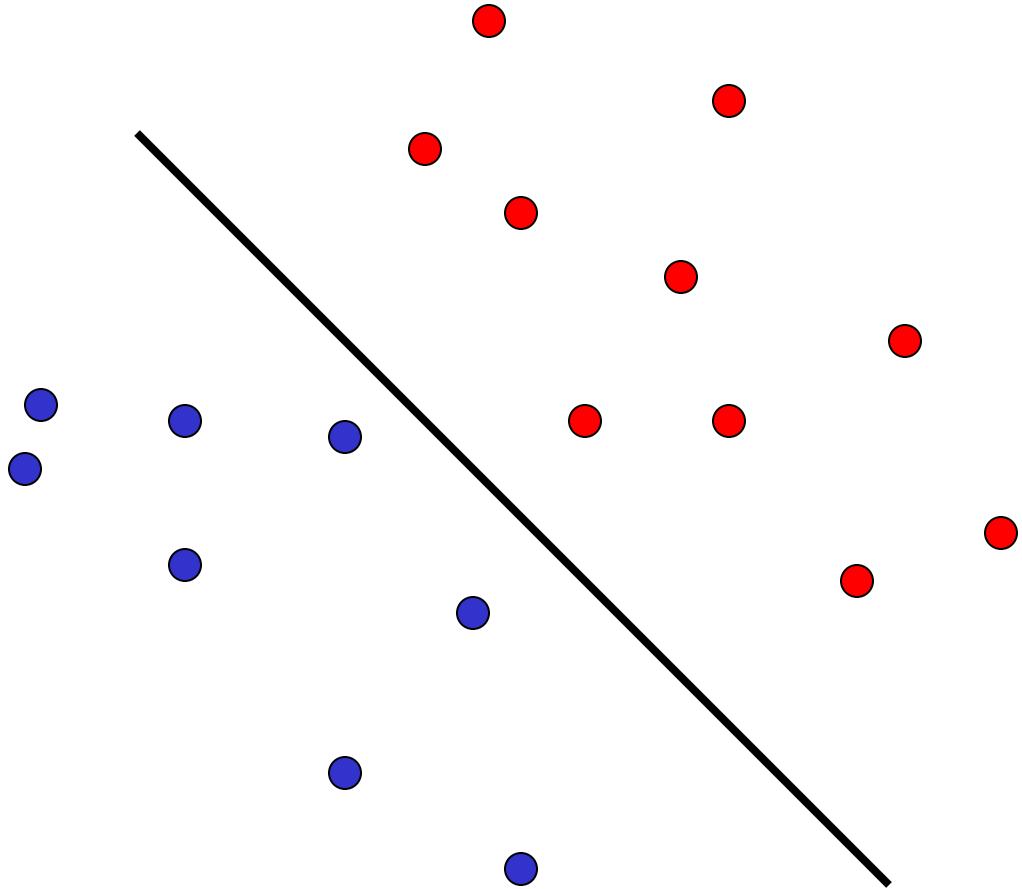
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)



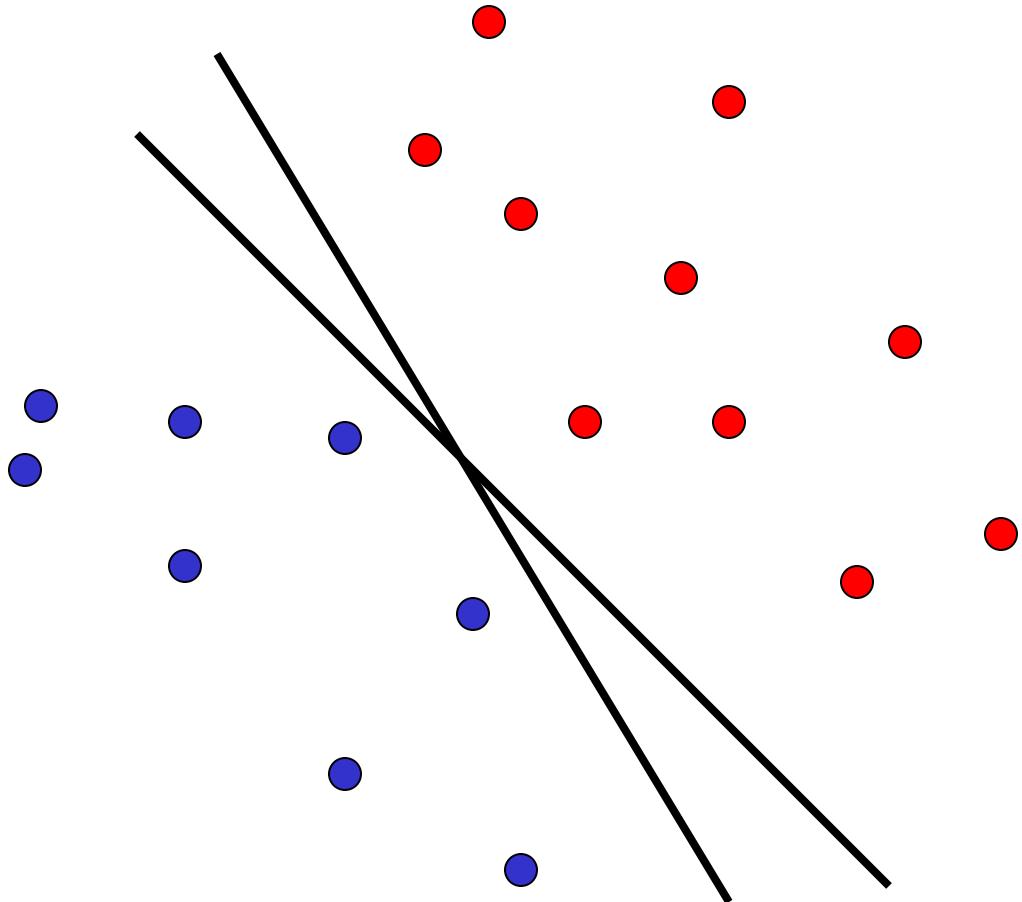
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)



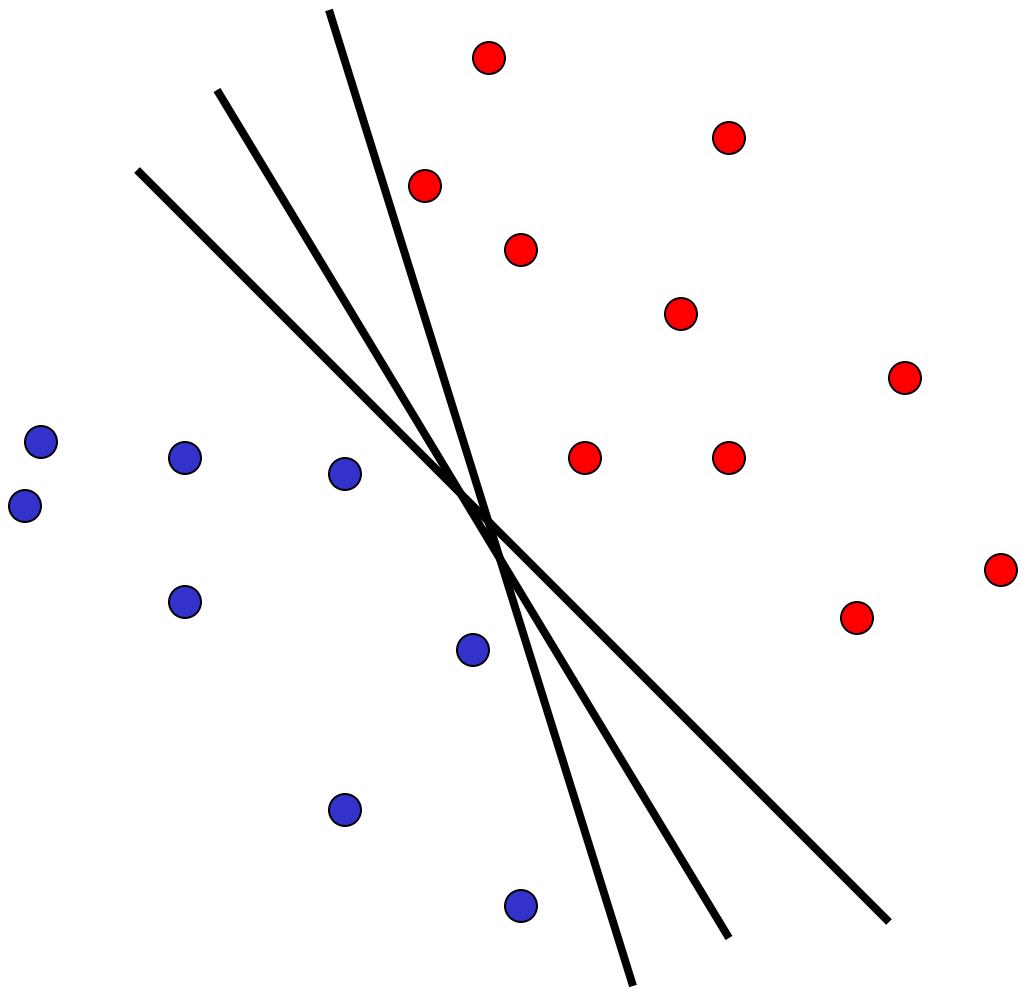
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)



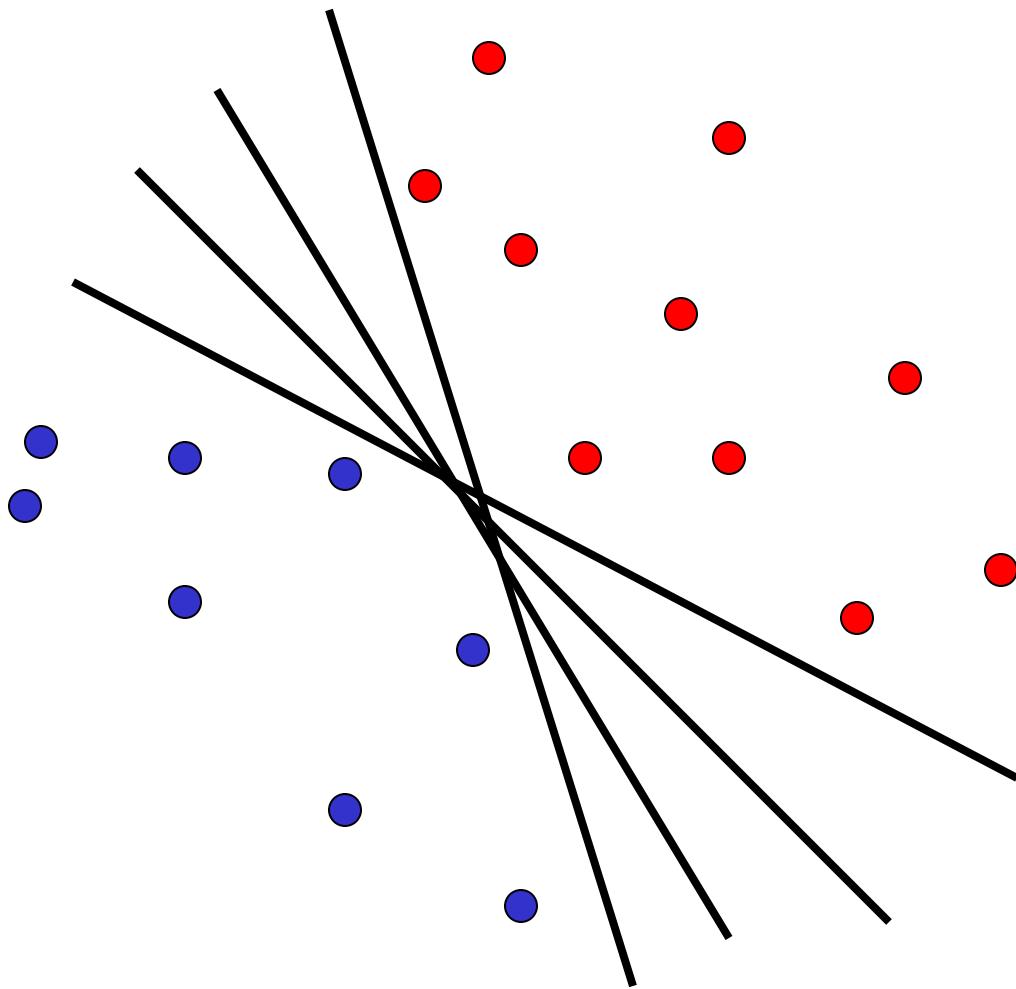
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)



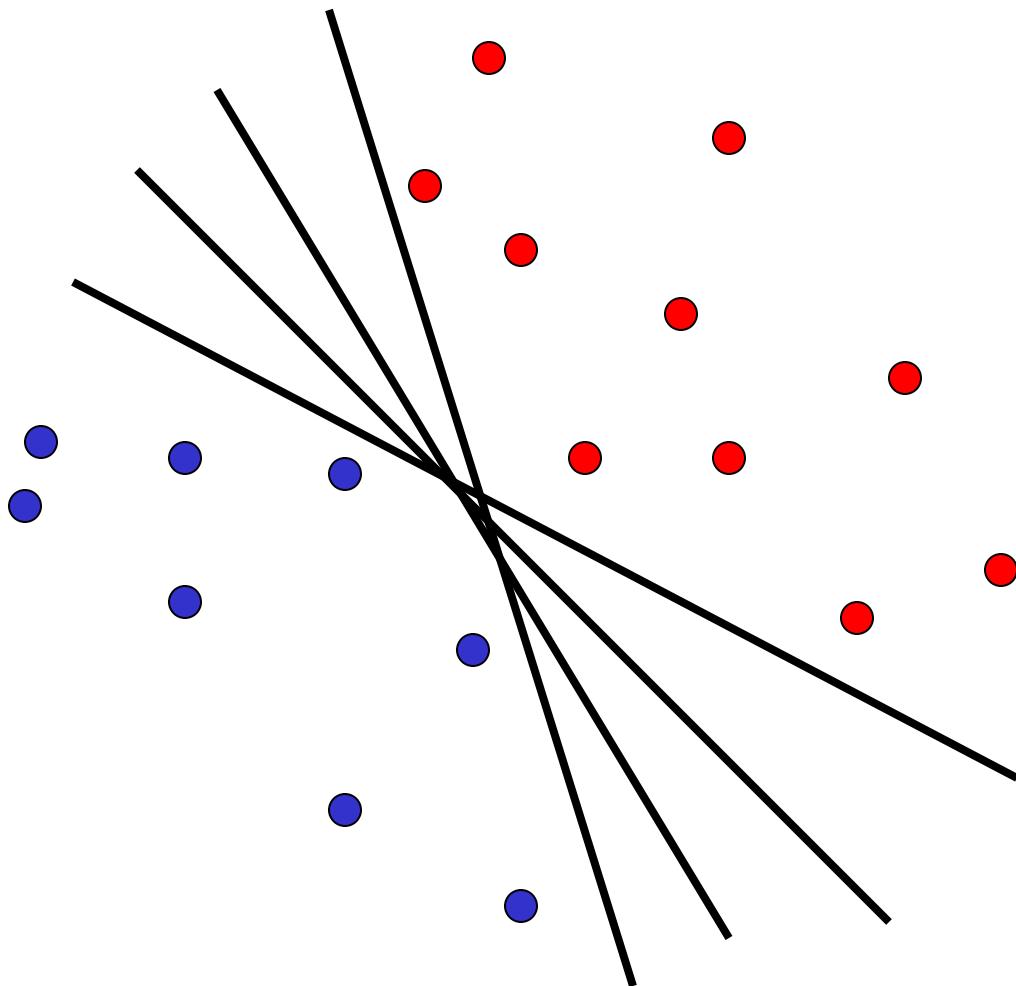
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)



Support vector machines

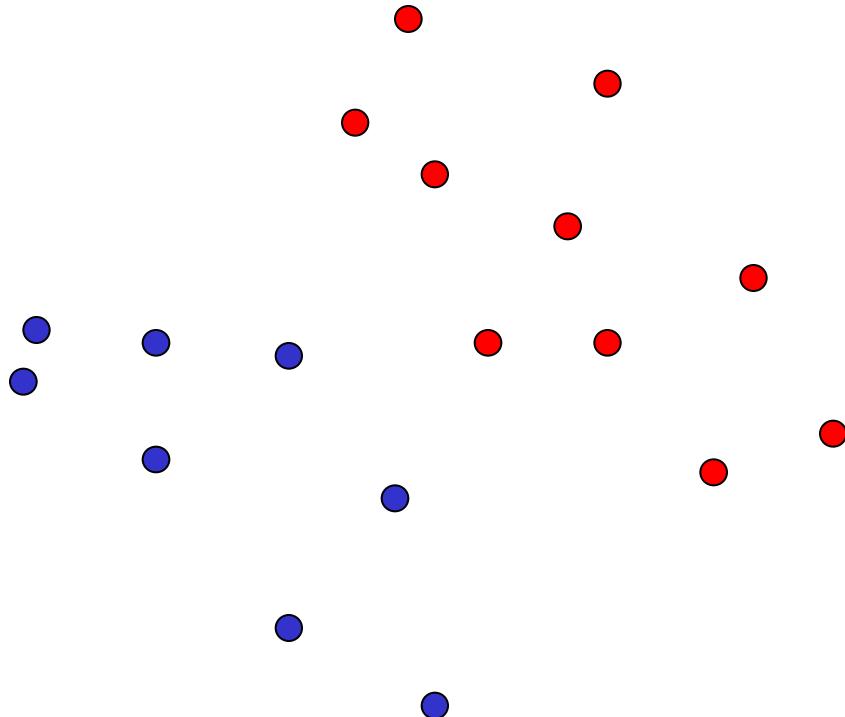
- When the data is linearly separable, there may be more than one separator (hyperplane)



Which separator
is best?

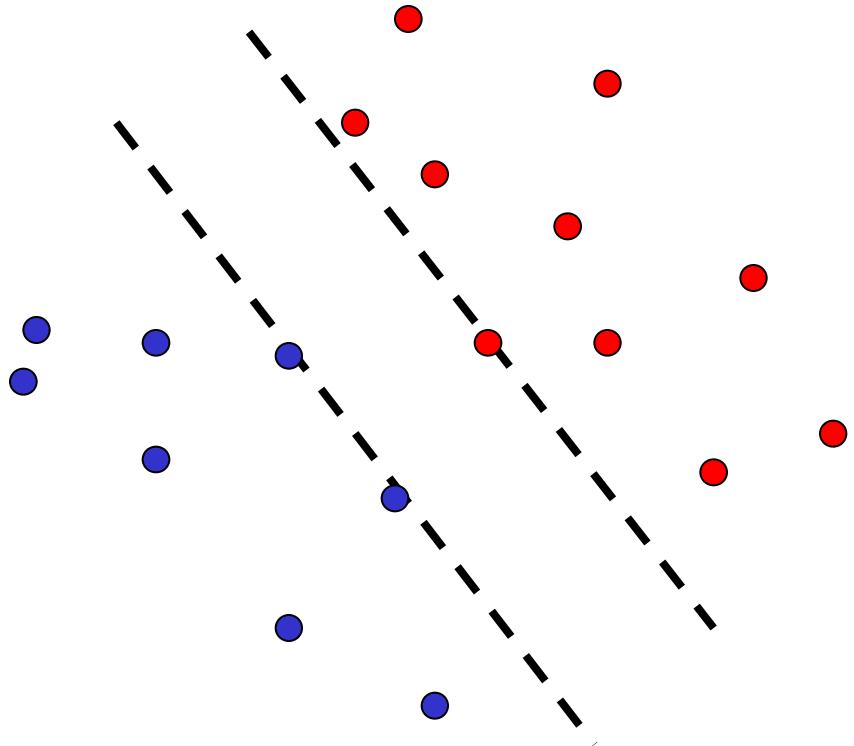
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



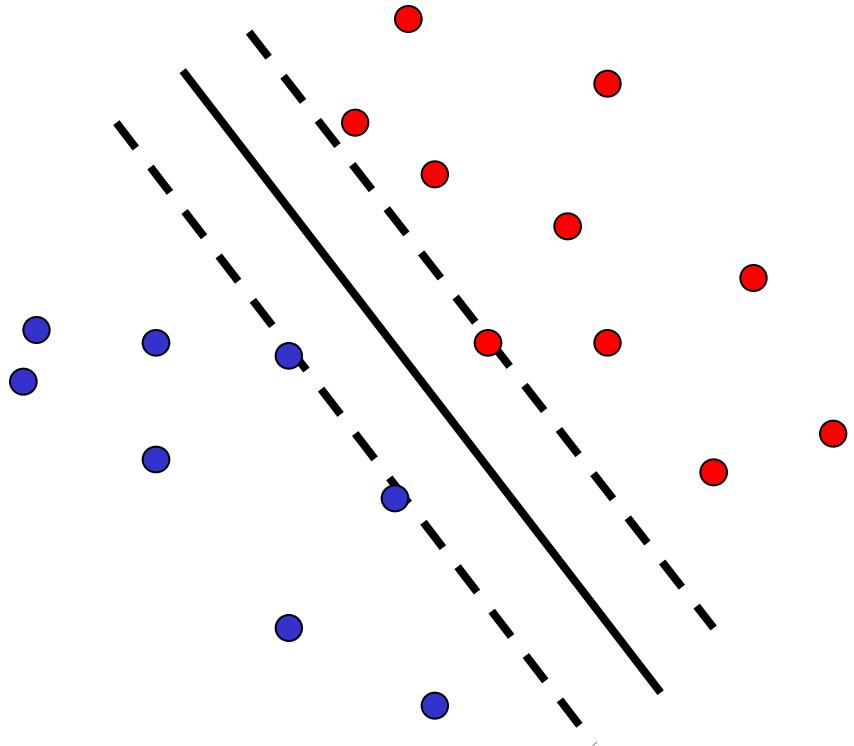
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



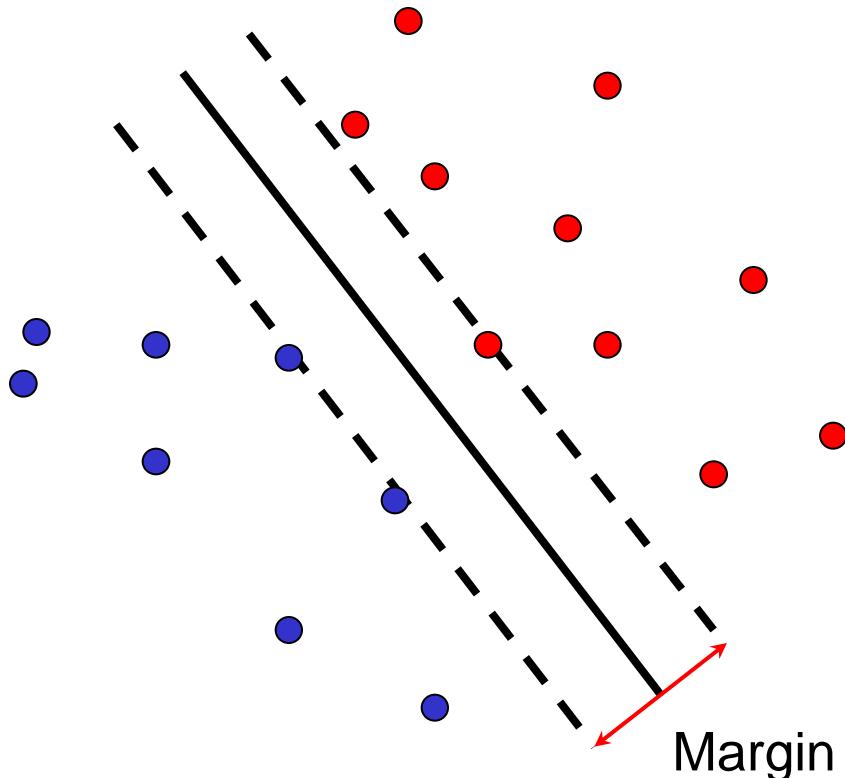
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



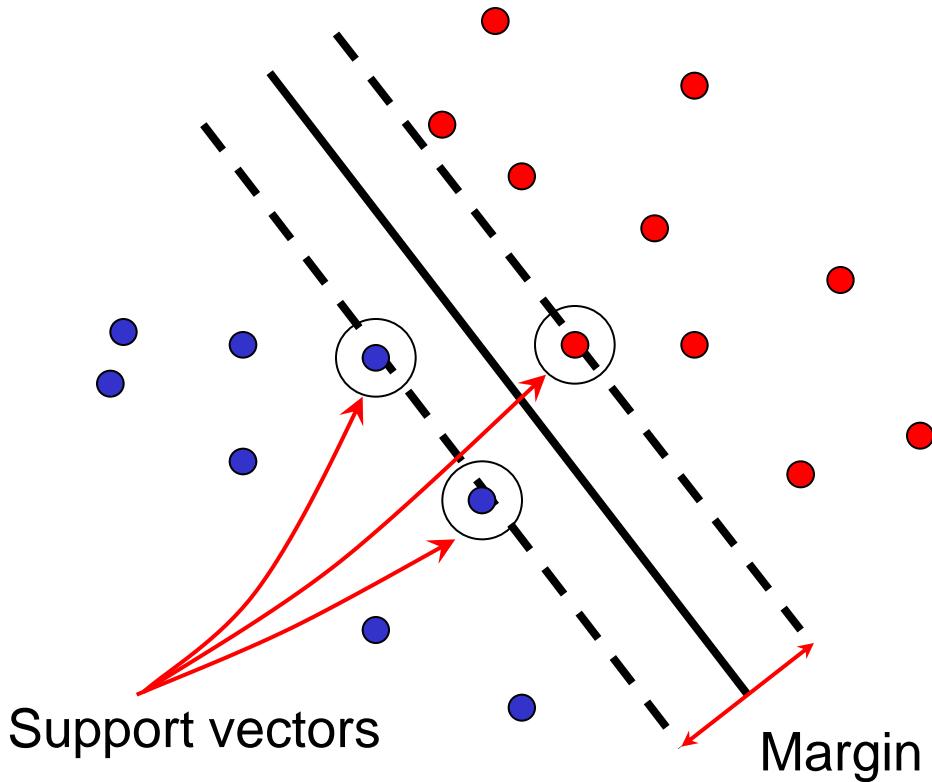
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



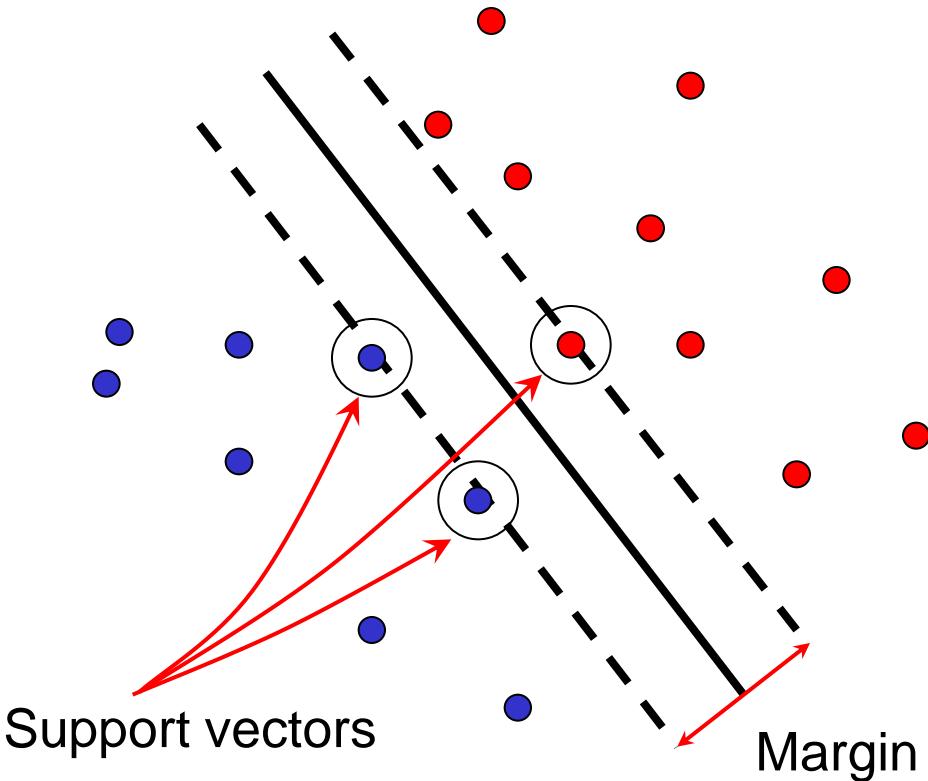
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



Support vector machines

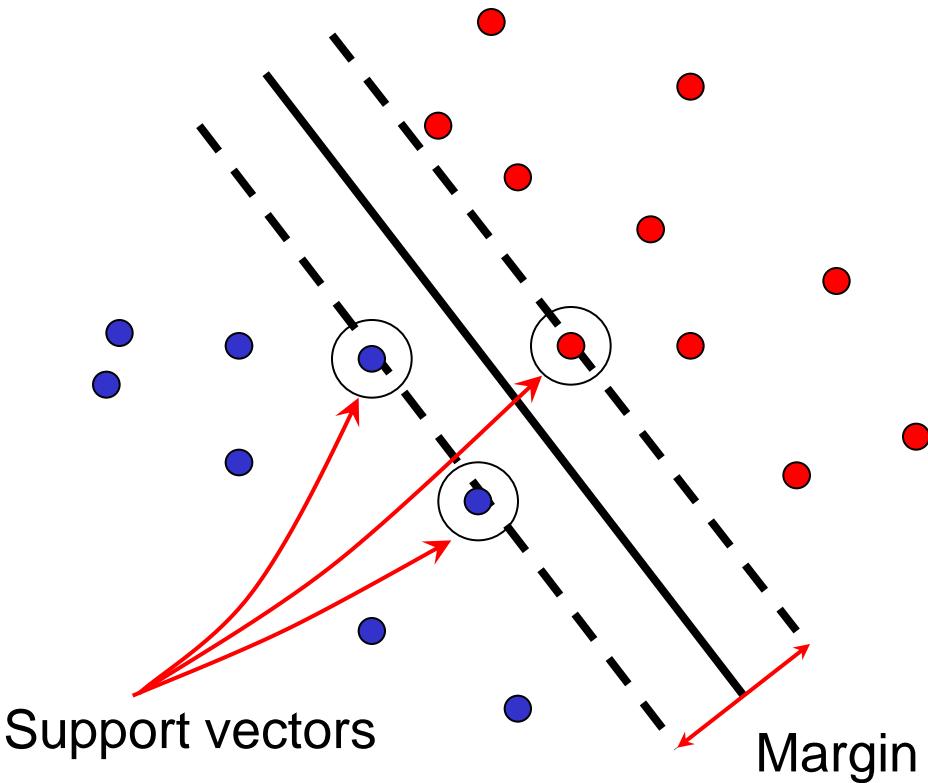
- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



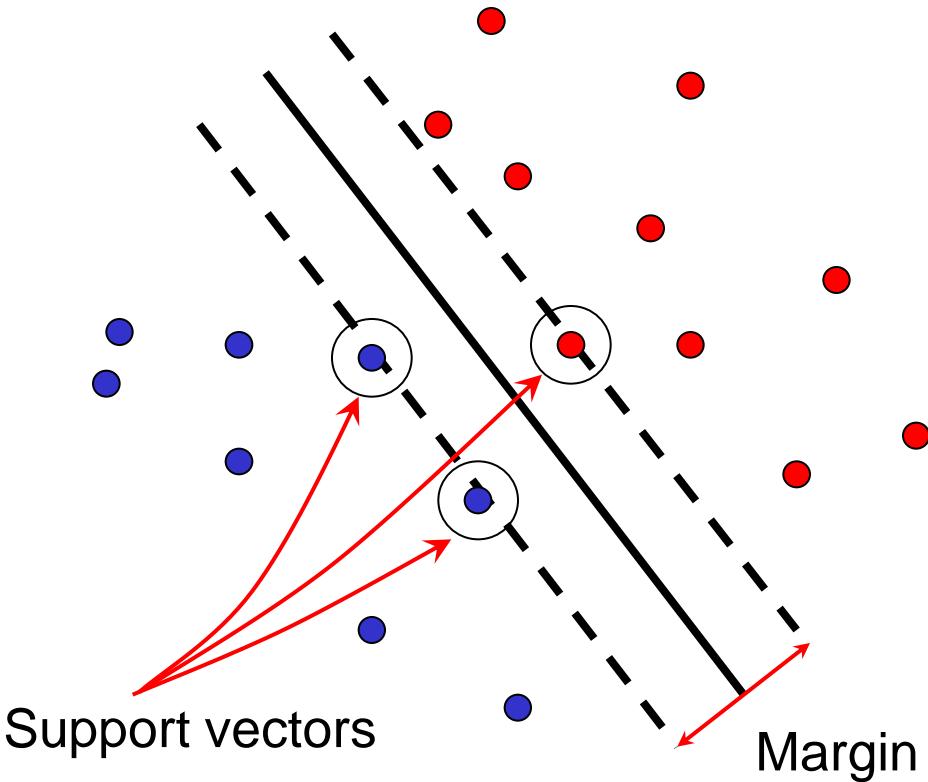
\mathbf{x}_i positive ($y_i = 1$): $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

\mathbf{x}_i negative ($y_i = -1$): $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

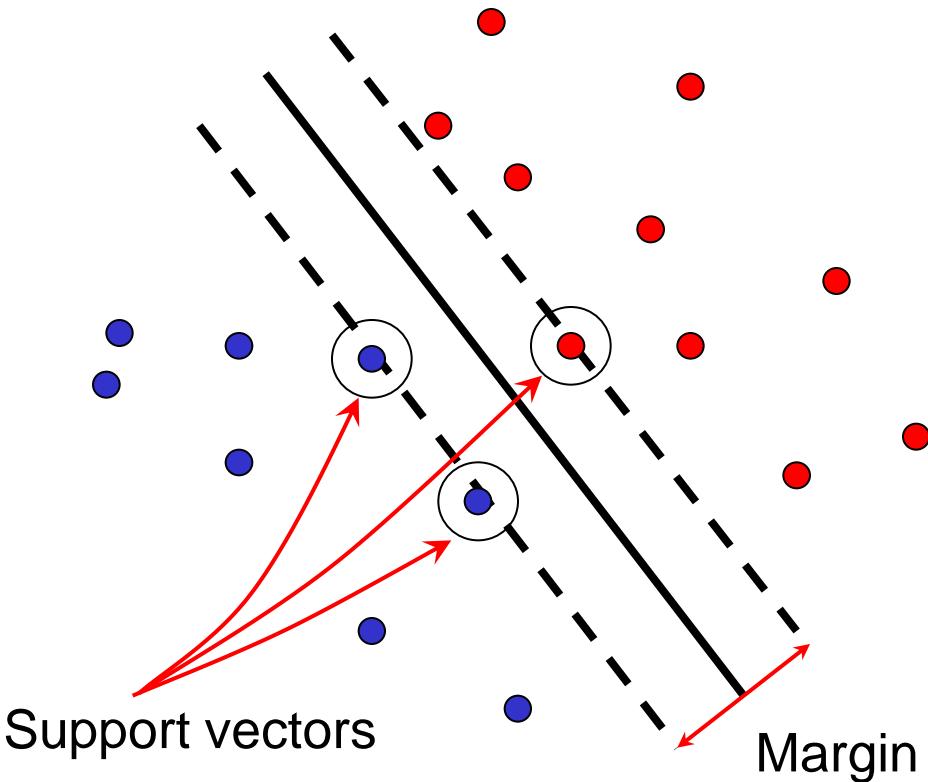
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

- For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

- Distance between point and hyperplane:
$$\frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and hyperplane:
$$\frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

Therefore, the margin is $2 / \|\mathbf{w}\|$

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|\mathbf{w}\|$

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1) : \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

Finding the maximum margin hyperplane

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
-
- The diagram illustrates the decomposition of the learned weight vector \mathbf{w} into a sum of scaled support vectors. A red arrow points from the term $\sum_i \alpha_i y_i$ in the equation to a box labeled "Support vector". Another red arrow points from the term \mathbf{x}_i in the equation to a box labeled "learned weight".

- The weights α_i are non-zero only at support vectors.

Finding the maximum margin hyperplane

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)
 $\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$

Finding the maximum margin hyperplane

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Classification function:

$$f(x) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \quad \begin{matrix} \text{If } f(x) < 0, \text{ classify as negative,} \\ = \text{sign}\left(\sum_i \alpha_i y_i \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right) \quad \begin{matrix} \text{if } f(x) > 0, \text{ classify as positive} \end{matrix} \end{matrix}$$

Dot product only!



SVM parameter learning

- Separable data: $\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$ subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$
- The diagram illustrates the SVM optimization problem. It shows the objective function $\frac{1}{2} \|\mathbf{w}\|^2$ enclosed in a red bracket below it, and the constraint $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ enclosed in a red bracket below the first one. Below these brackets are two red-bordered boxes: the left one contains the text "Maximize margin", and the right one contains the text "Classify training data correctly".
- Maximize
margin
- Classify training data correctly

SVM parameter learning

- Separable data: $\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$ subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$



Maximize margin



Classify training data correctly

- Non-separable data:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$$



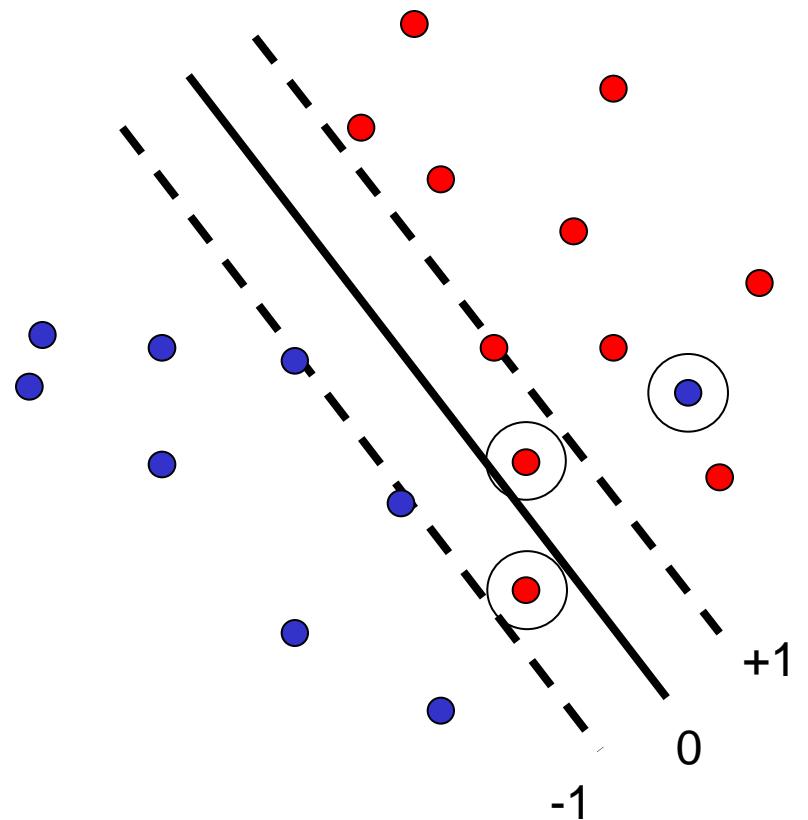
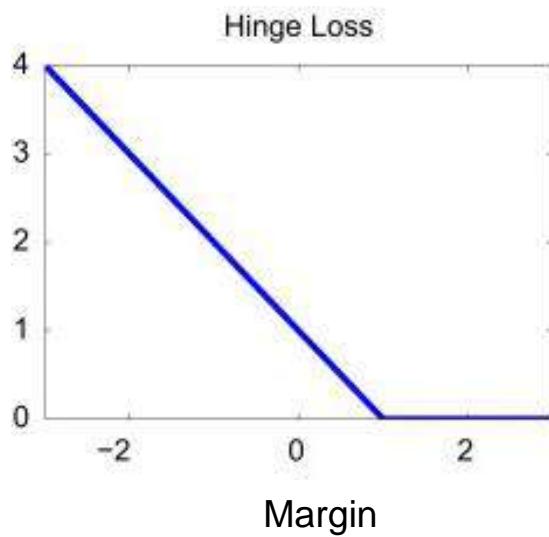
Maximize margin



Minimize classification mistakes

SVM parameter learning

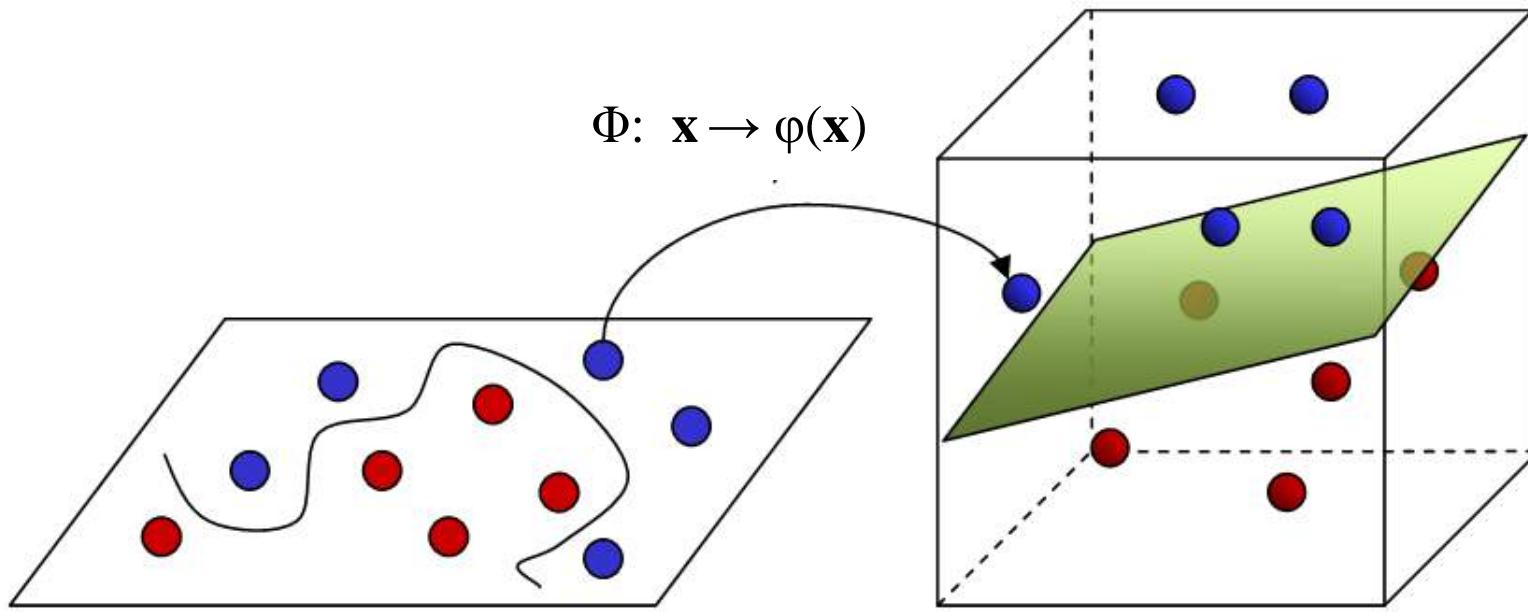
$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$



Demo: <http://cs.stanford.edu/people/karpathy/svmjs/demo>

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable



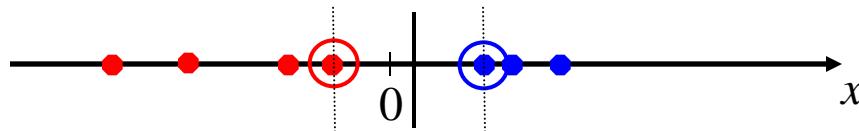
Input Space

Feature Space

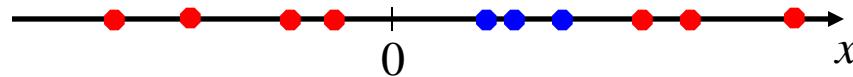
[Image source](#)

Nonlinear SVMs

- Linearly separable dataset in 1D:

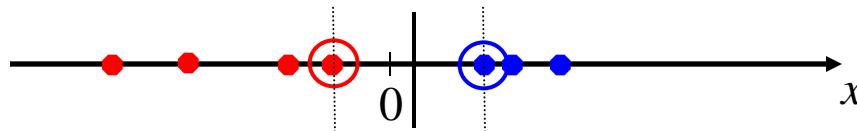


- Non-linearly separable dataset in 1D:

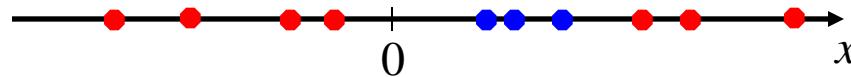


Nonlinear SVMs

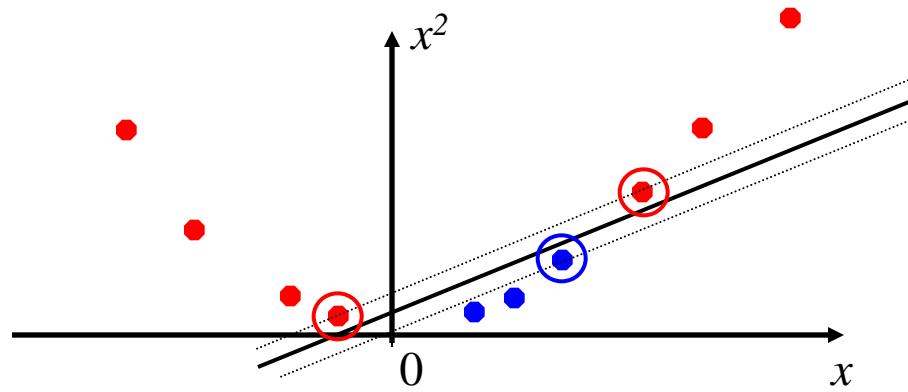
- Linearly separable dataset in 1D:



- Non-linearly separable dataset in 1D:



- We can map the data to a *higher-dimensional space*:



The kernel trick

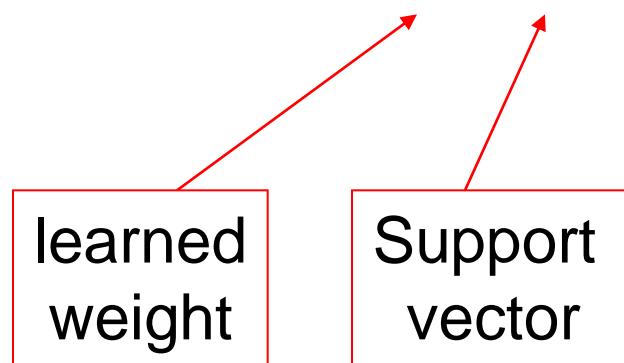
- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable
- **The kernel trick:** instead of explicitly computing the lifting transformation $\varphi(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})$$

The kernel trick

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$



The kernel trick

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Kernel SVM decision function:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- This gives a nonlinear decision boundary in the original feature space

Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$:

Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T$$

$$[1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}]$$

Example

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$;

let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2,$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T$$

$$[1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}]$$

$$= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j),$$

$$\text{where } \phi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2]$$

SVMs: Pros and cons

- **Pros**
 - Kernel-based framework is **very powerful**, flexible
 - Training is convex optimization, **globally optimal solution** can be found
 - SVMs work very well in practice, even with **very small training sample sizes**
- **Cons**
 - No “direct” **multi-class SVM**, must combine two-class SVMs (e.g., with one-vs-others)
 - **Computation, memory** (esp. for nonlinear SVMs)

Multiclass Support Vector Machine loss

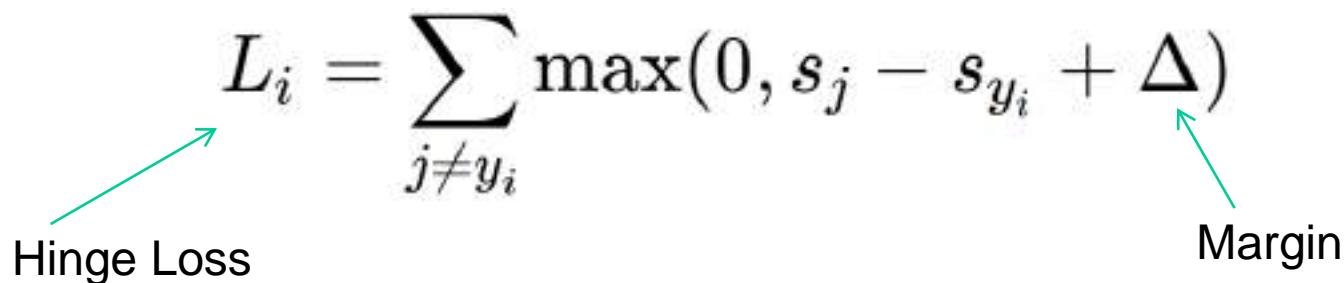
- i^{th} example: image x_i and the label y_i
- Score for the j^{th} class: $s_j = f(x_i, W)_j$

Multiclass Support Vector Machine loss

- i^{th} example: image x_i and the label y_i
- Score for the j^{th} class: $s_j = f(x_i, W)_j$
- The Multiclass SVM loss for the i^{th} example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Hinge Loss Margin

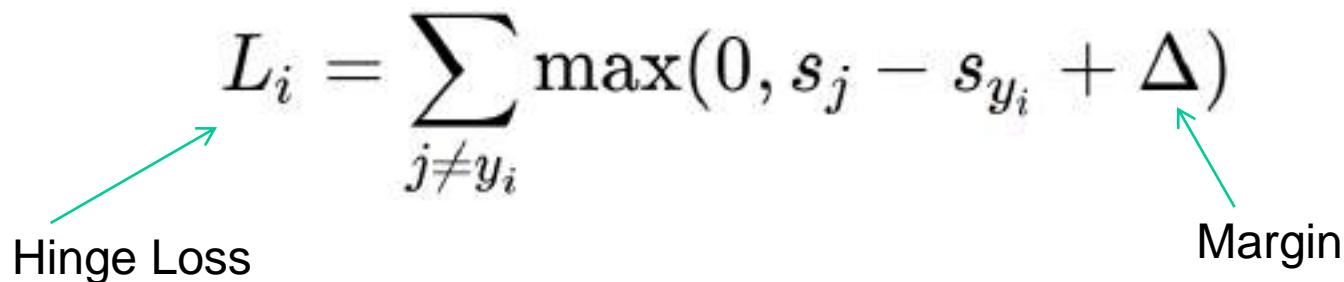


Multiclass Support Vector Machine loss

- i^{th} example: image x_i and the label y_i
- Score for the j^{th} class: $s_j = f(x_i, W)_j$
- The Multiclass SVM loss for the i^{th} example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Hinge Loss Margin



Problem: W is not necessarily unique

Source: cs231n, <http://cs231n.github.io/linear-classify/>

Multiclass Support Vector Machine loss

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Multiclass Support Vector Machine loss

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

- Hinge Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

Multiclass Support Vector Machine loss

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

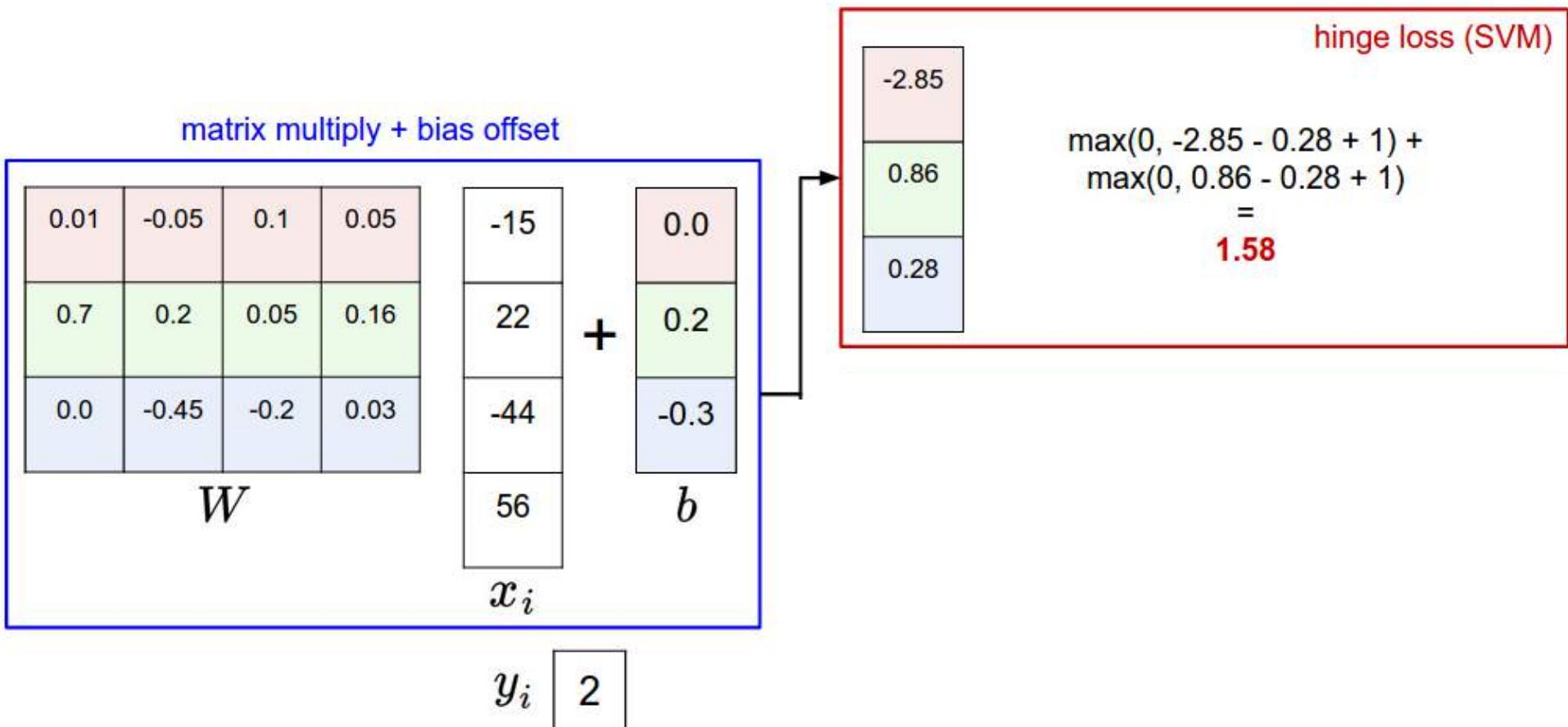
- Hinge Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$



$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

Hinge Loss



Softmax Classifier

- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

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

Softmax Classifier

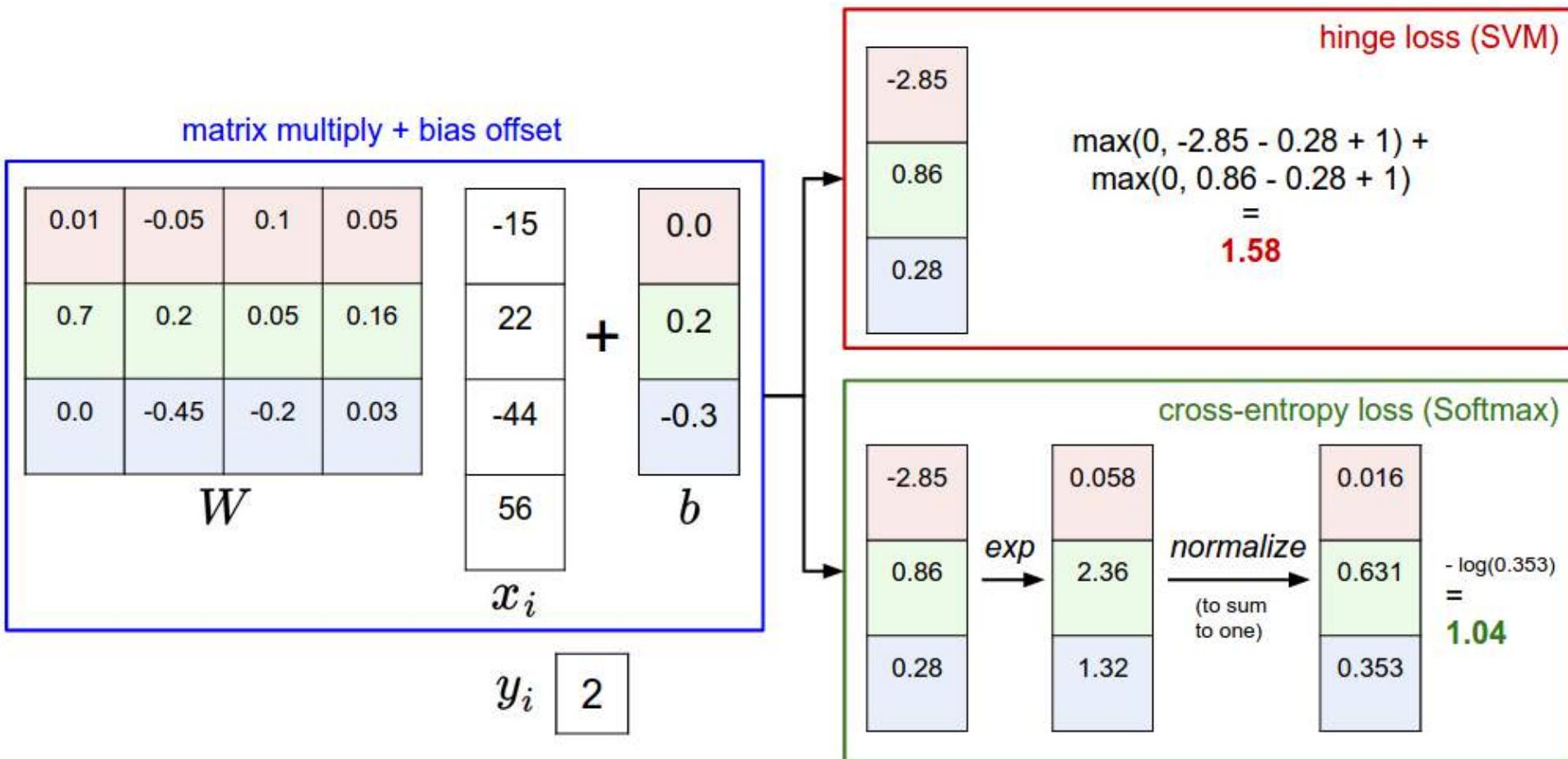
- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

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

- Softmax Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

Hinge vs Cross-entropy Loss



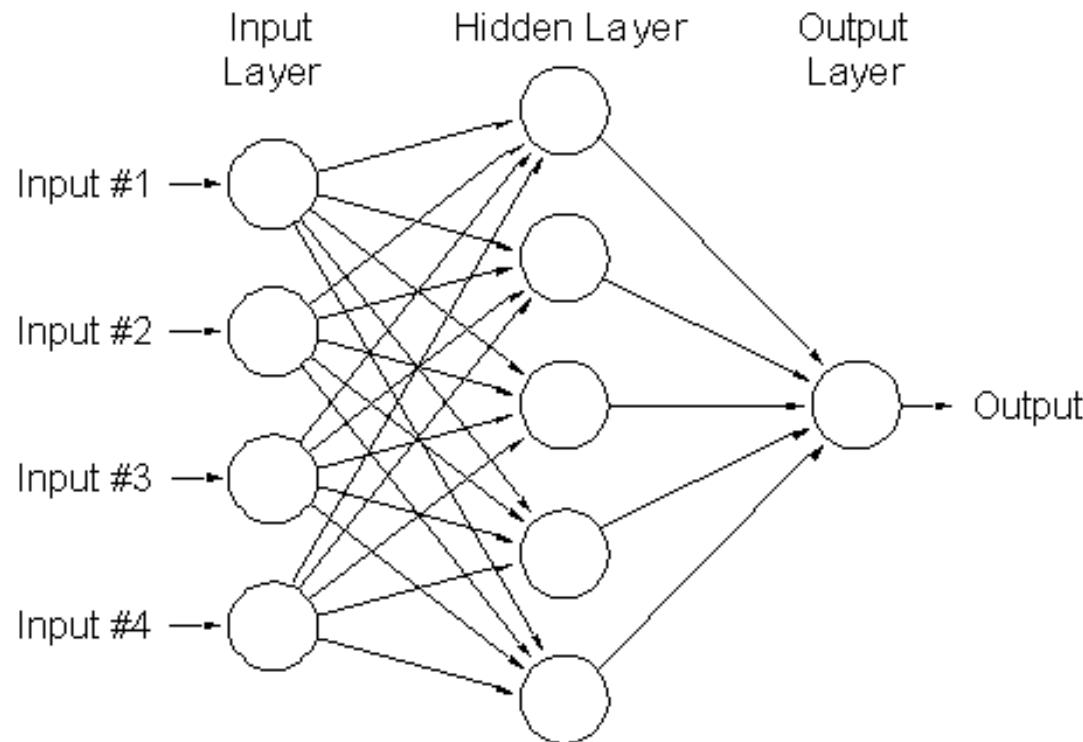
Acknowledgements

Thanks to the following researchers for making their teaching/research material online

- Forsyth
- Steve Seitz
- Noah Snavely
- J.B. Huang
- Derek Hoiem
- D. Lowe
- A. Bobick
- S. Lazebnik
- K. Grauman
- R. Zaleski
- Antonio Torralba
- Rob Fergus
- Leibe
- And many more

Next Lecture

Neural Networks



Computer Vision

Neural Networks

Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**



We have learned so far in this module

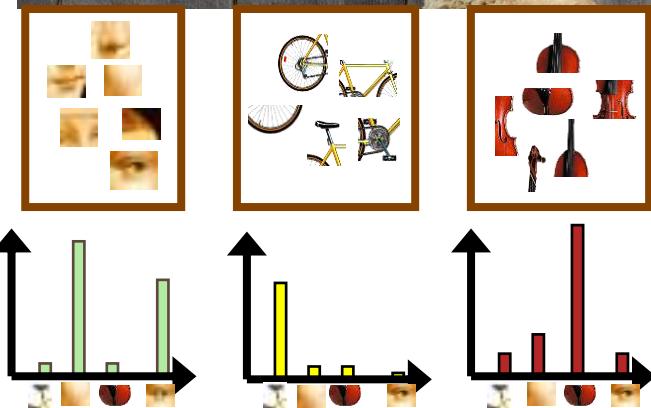
Image features and categorization

Choosing right features
Object, Scene, Action, etc.



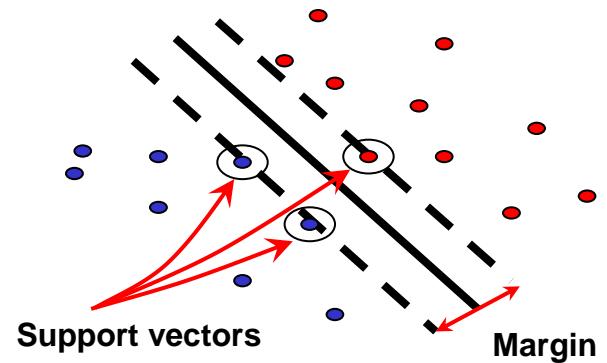
Bag-of-visual-words

Extract local features
Learn “visual vocabulary”
Quantize features using visual vocabulary
Represent by frequencies of “visual words”



Classifiers

Nearest neighbor, KNN, Linear classifier,
SVM, Non-linear SVM, Multi-class SVM,
Softmax classifier



Previous Class

Two key components in context of the image classification

1. A **(parameterized) score function:**

Mapping the raw image pixels/features to class scores
(e.g. a linear function)

Previous Class

Two key components in context of the image classification

1. A (parameterized) score function:

Mapping the raw image pixels/features to class scores
(e.g. a linear function)

2. A loss function:

Measures the goodness of parameter values in terms of how well it performs over the training data
(e.g. Softmax/SVM)

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $R(W) = \sum_k \sum_l W_{k,l}^2$

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $R(W) = \sum_k \sum_l W_{k,l}^2$

SVM Loss:

Hinge Loss

Max-margin loss

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $R(W) = \sum_k \sum_l W_{k,l}^2$

SVM Loss:

Hinge Loss

Max-margin loss

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Softmax Loss:

Cross-entropy loss

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

Today's class

Optimization

Gradient Descent & Back propagation

Perceptron

Update rule

Neural networks

Optimization

Optimization is the process of finding the set of parameters W that minimize the loss function.

Optimization

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1:First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Optimization

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1: First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Strategy #2: Random local search:

Start out with a random W , generate random changes δW to it and if the loss at the changed $W + \delta W$ is lower, we will perform an update.

Optimization

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1: First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Strategy #2: Random local search:

Start out with a random W , generate random changes δW to it and if the loss at the changed $W + \delta W$ is lower, we will perform an update.

Strategy #3: Following the gradients:

There is no need to randomly search for a good direction: this direction is related to the **gradient** of the loss function.

Gradient Descent

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a **parameter update**.

Gradient Descent

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a **parameter update**.

Vanilla (Original) Gradient Descent:

```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Gradient Descent

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a **parameter update**.

Vanilla (Original) Gradient Descent:

```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Mini-batch Gradient Descent (MGD):

```
while True:  
    data_batch = sample_training_data(data, 256) # sample 256 examples  
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Gradient Descent

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a **parameter update**.

Vanilla (Original) Gradient Descent:

```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Mini-batch Gradient Descent (MGD):

```
while True:  
    data_batch = sample_training_data(data, 256) # sample 256 examples  
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Stochastic Gradient Descent (SGD):

Special case of MGD when mini-batch contains only a single example

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = \quad \frac{\partial f}{\partial y} =$$

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = \quad \frac{\partial f}{\partial y} =$$

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

$$f(x, y) = \max(x, y) \quad \rightarrow \quad \frac{\partial f}{\partial x} = \quad \frac{\partial f}{\partial y} =$$

Interpretation of the gradient

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

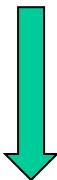
$$f(x, y) = \max(x, y) \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1(x \geq y) \quad \frac{\partial f}{\partial y} = 1(y \geq x)$$

Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

Compound expressions with chain rule

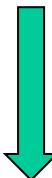
$$f(x, y, z) = (x + y)z$$



$$q = x + y \text{ and } f = qz$$

Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$



$$q = x + y \text{ and } f = qz$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$



$$q = x + y \text{ and } f = qz$$

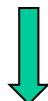
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$

Compound expressions with chain rule

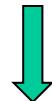
$$f(x, y, z) = (x + y)z \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$q = x + y \text{ and } f = qz \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

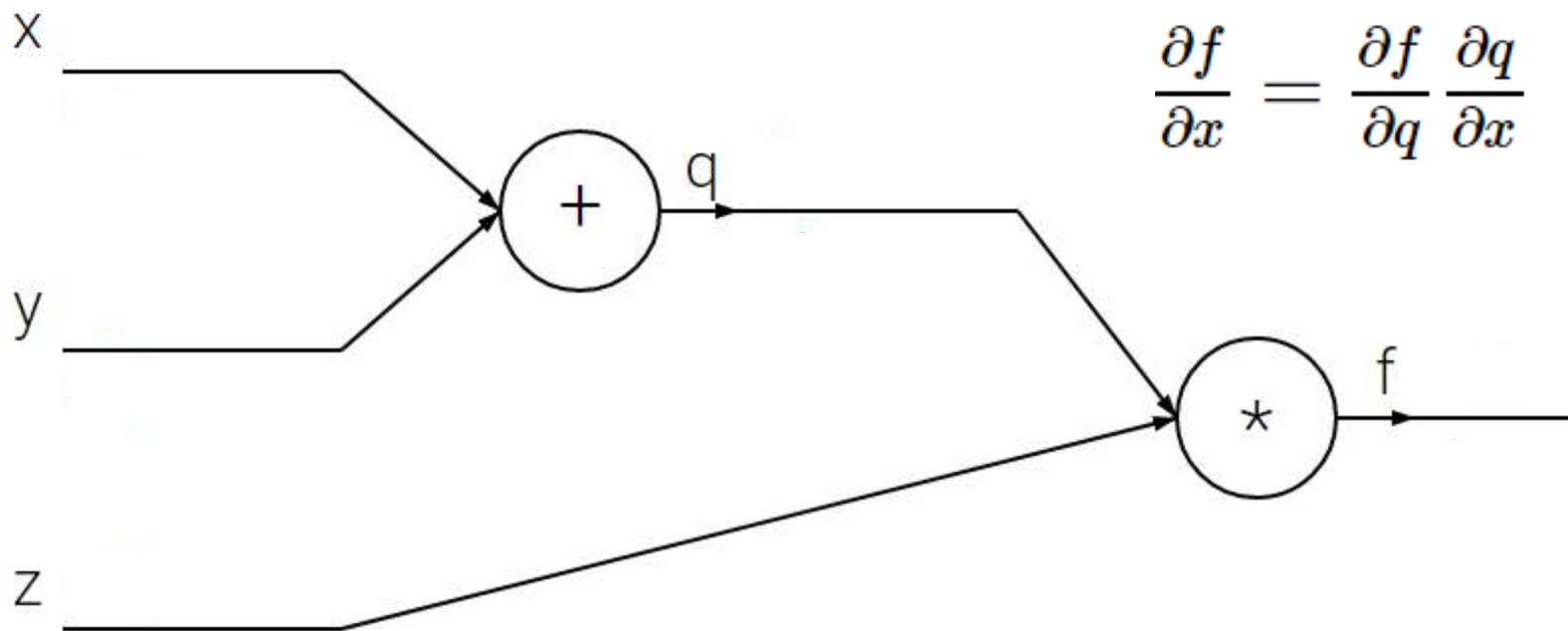


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

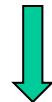
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

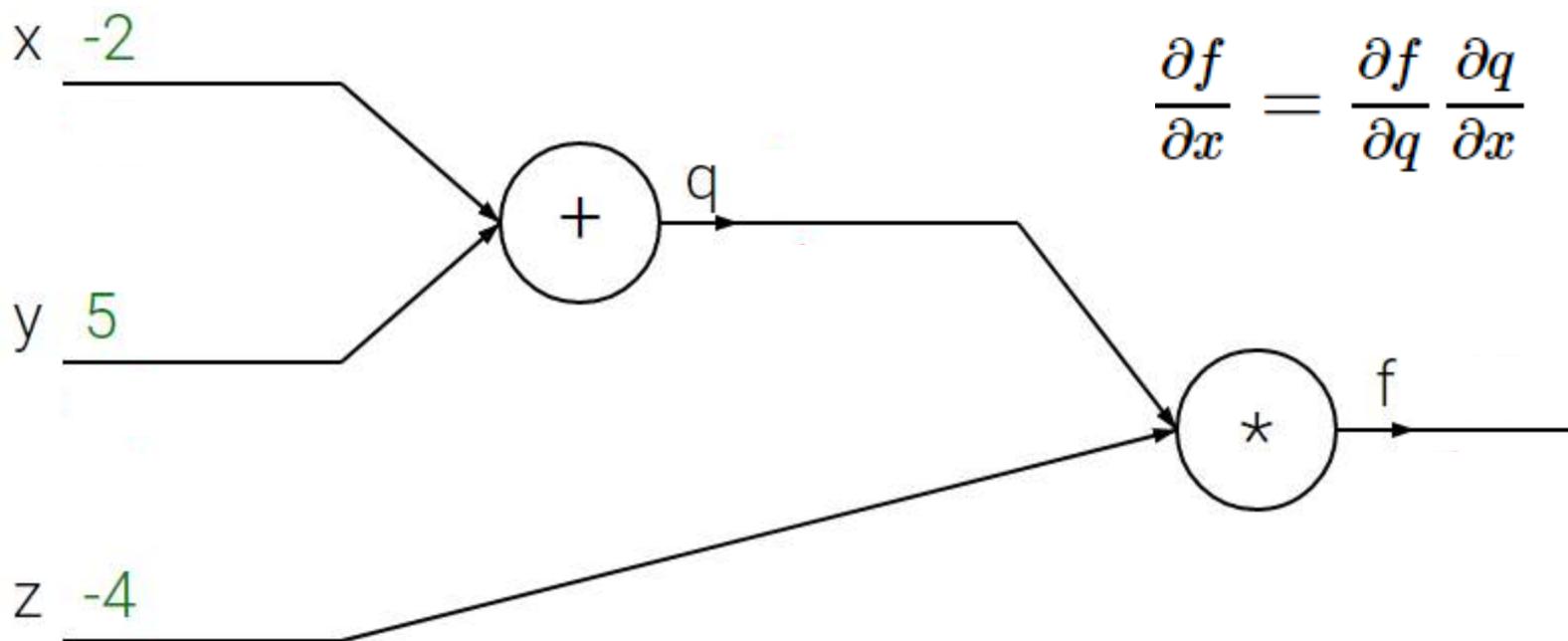


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



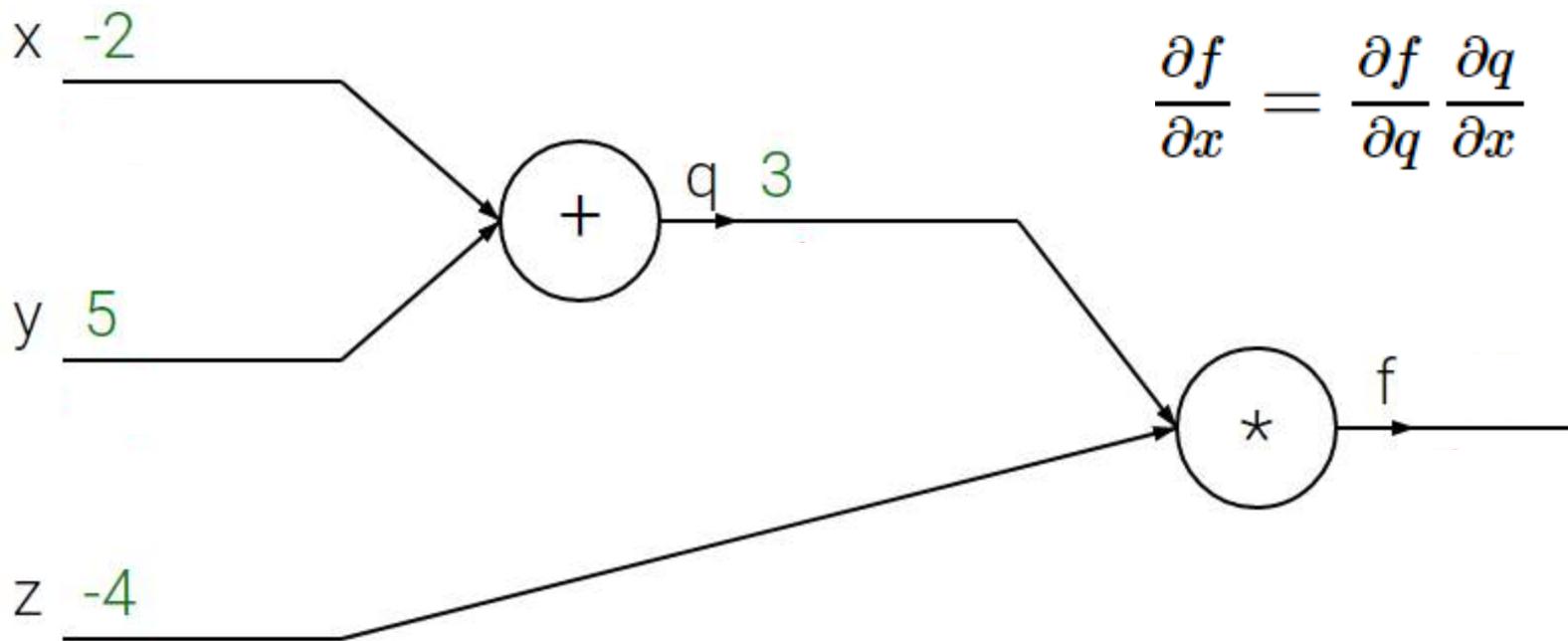
Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

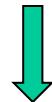
$$q = x + y \text{ and } f = qz$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

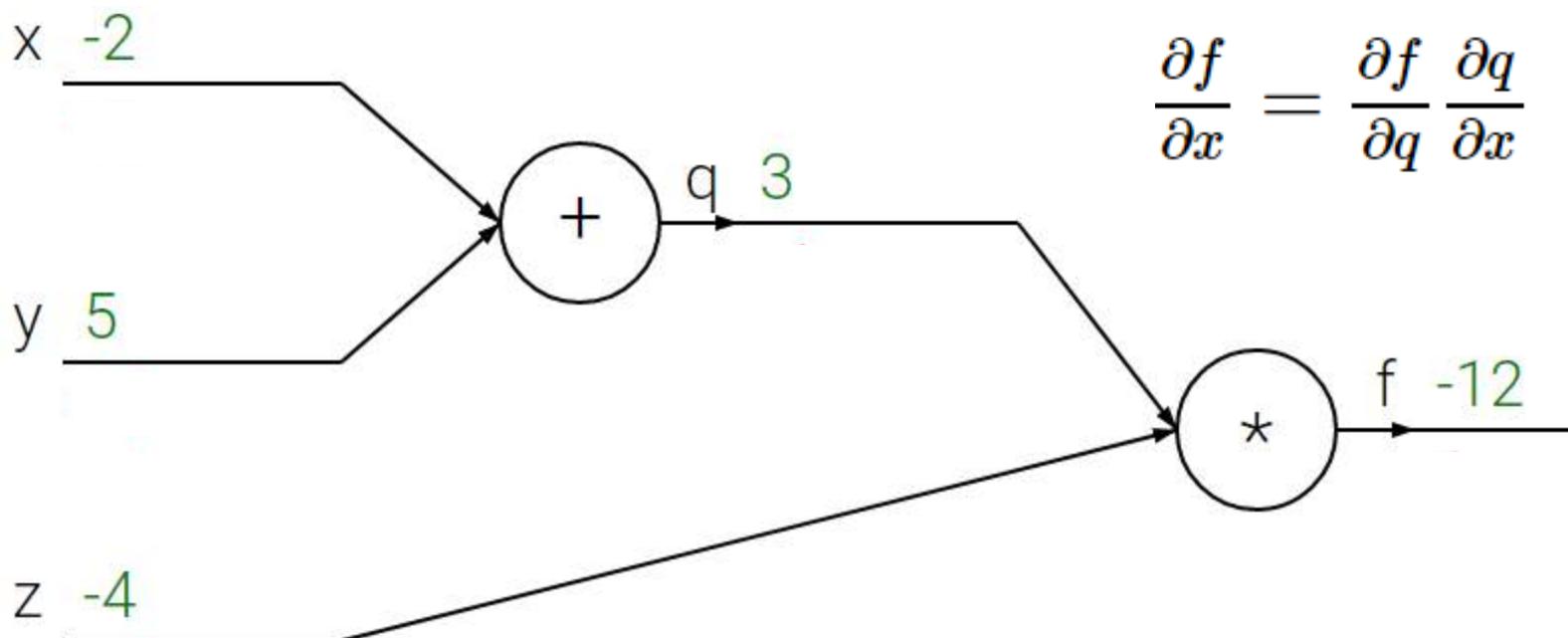


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

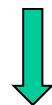
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

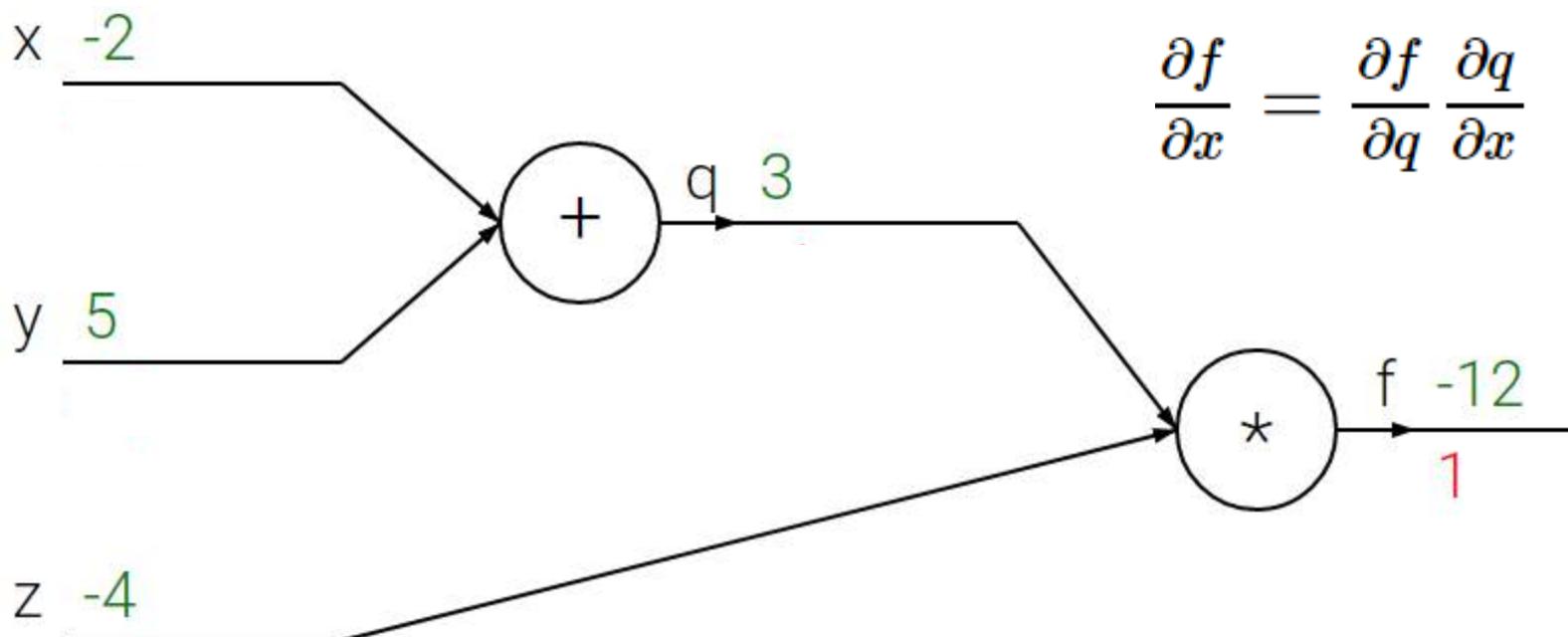


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

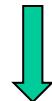
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

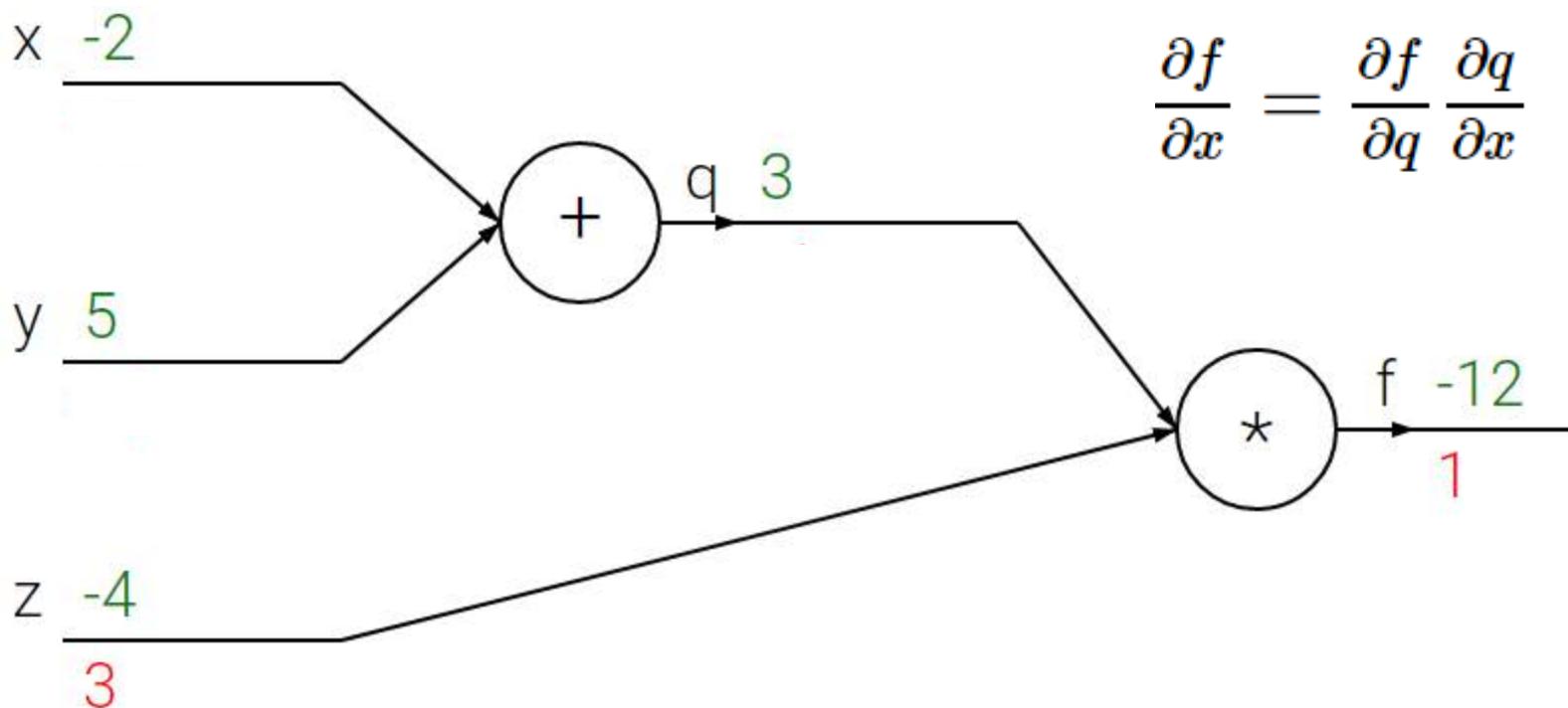


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

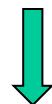
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

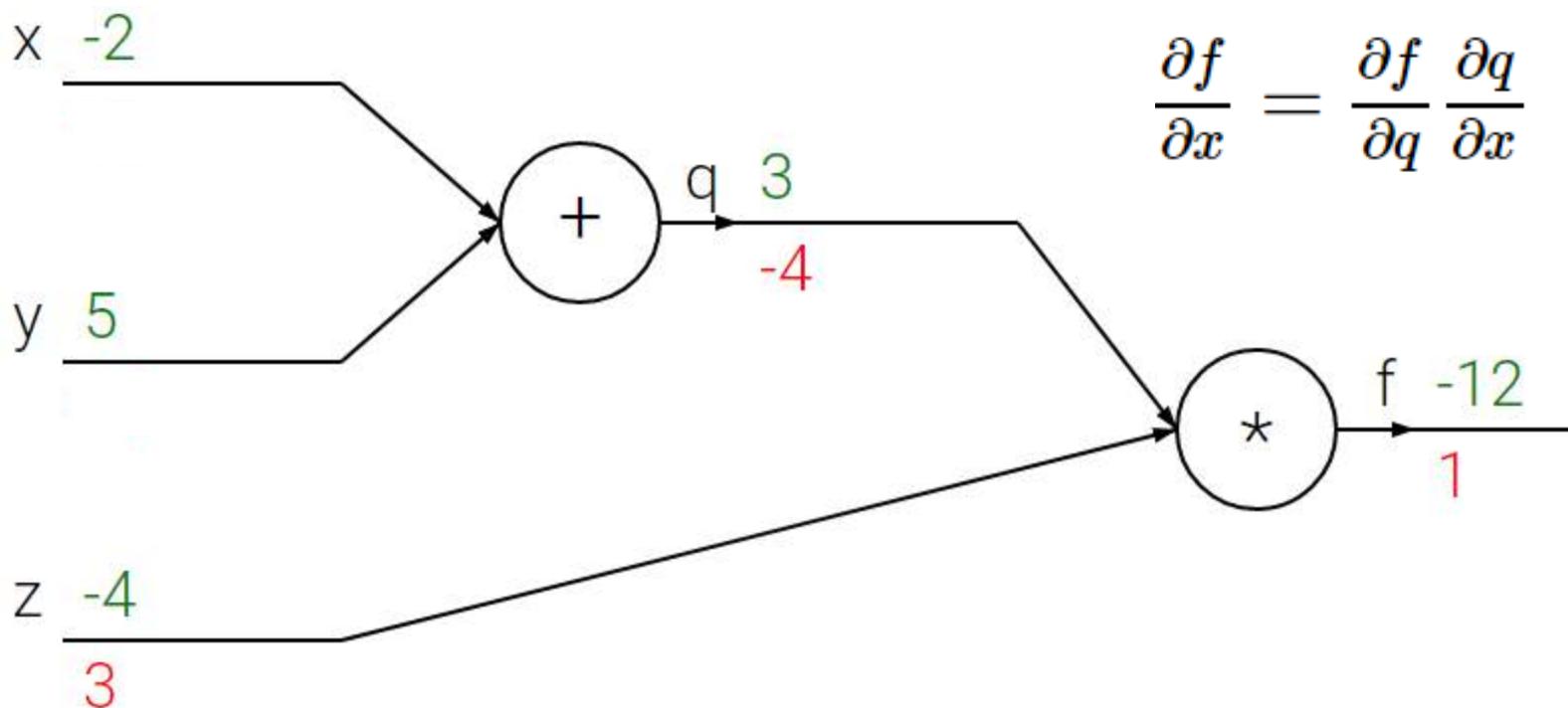


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

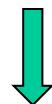
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

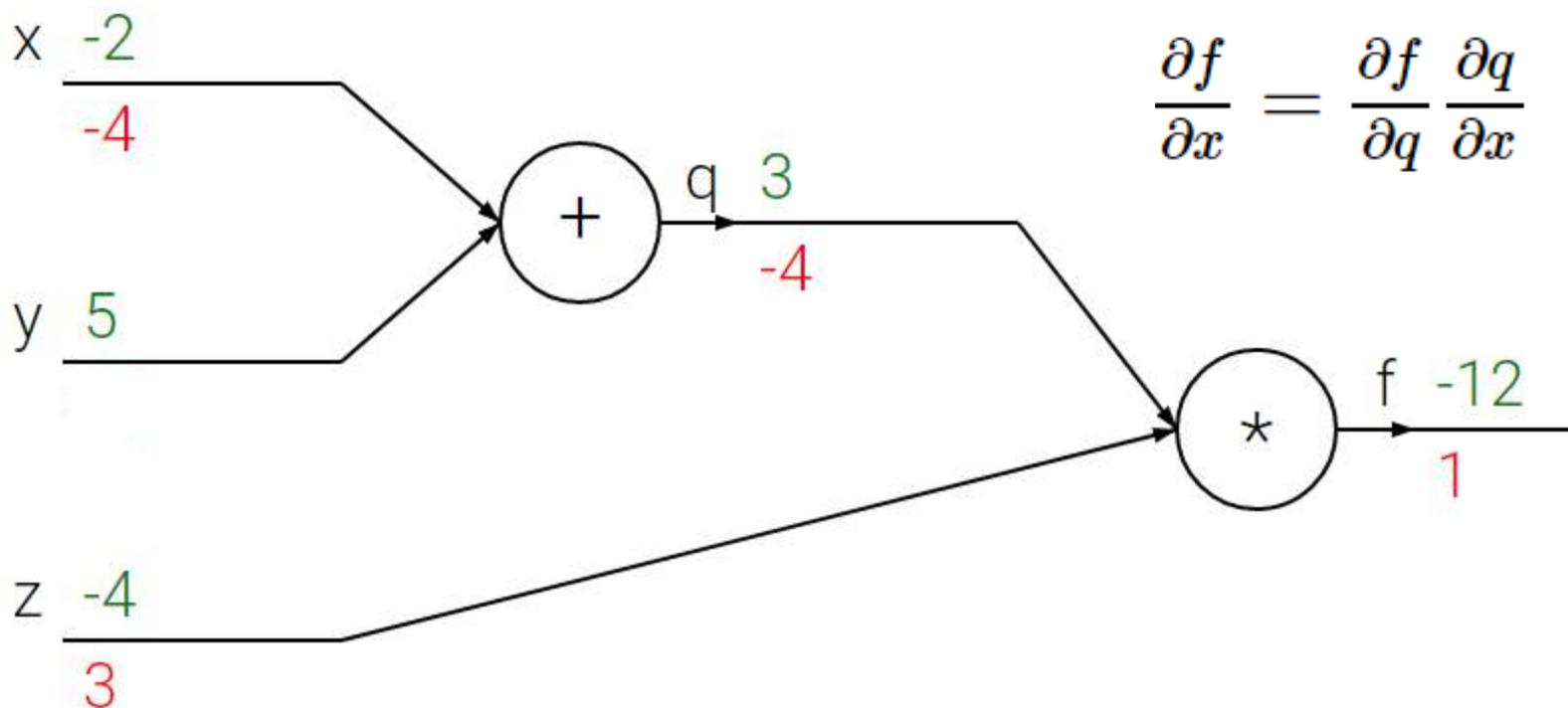


$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



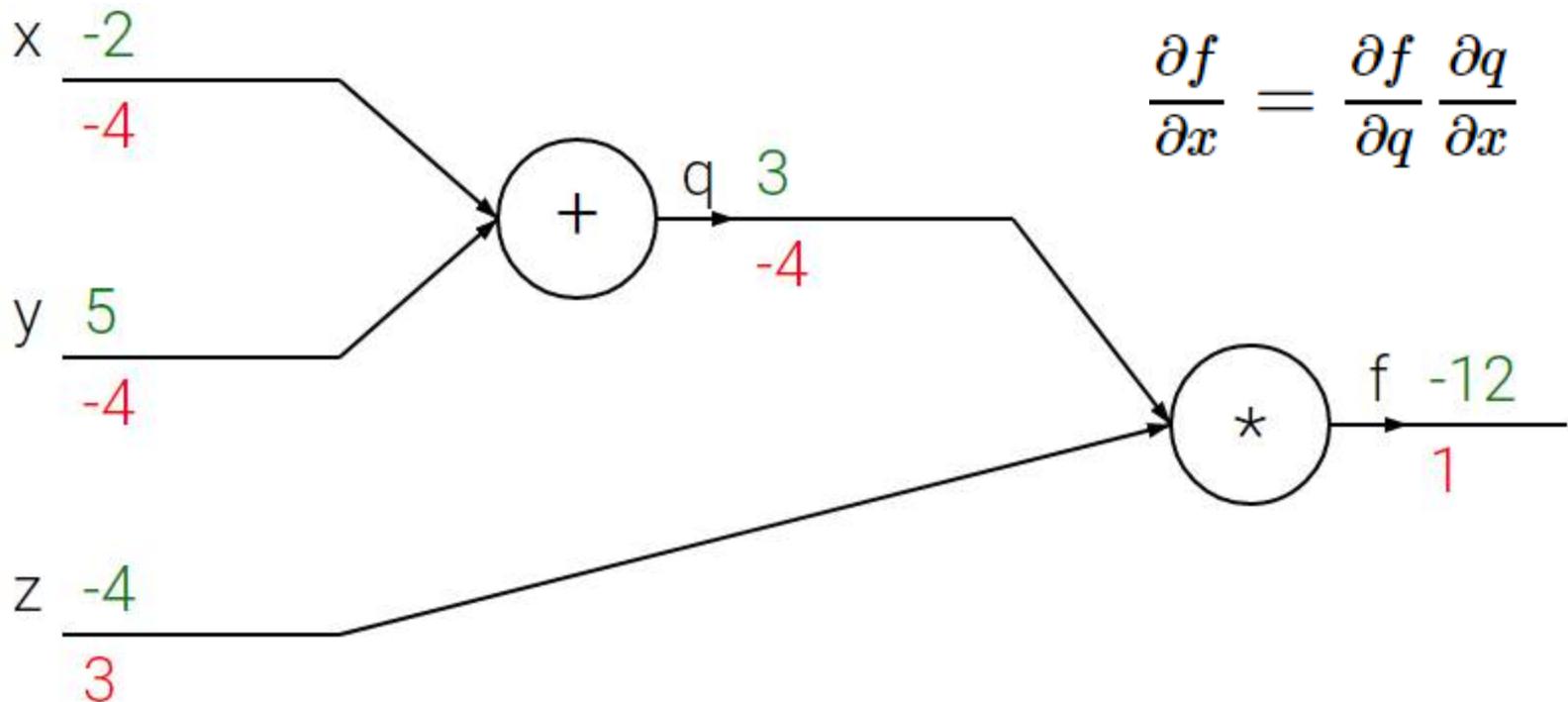
Compound expressions with chain rule

$$f(x, y, z) = (x + y)z$$

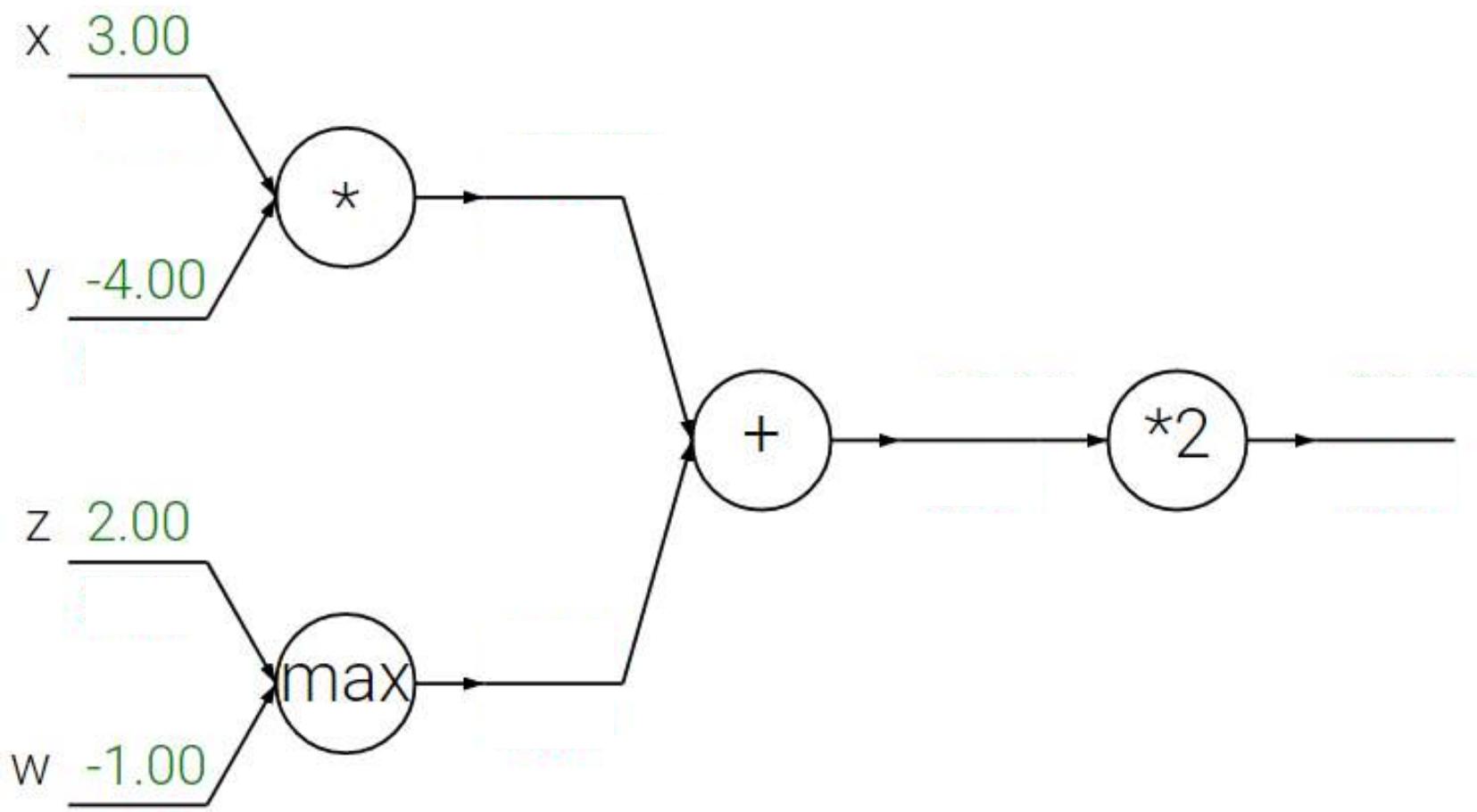
$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$q = x + y \text{ and } f = qz$$

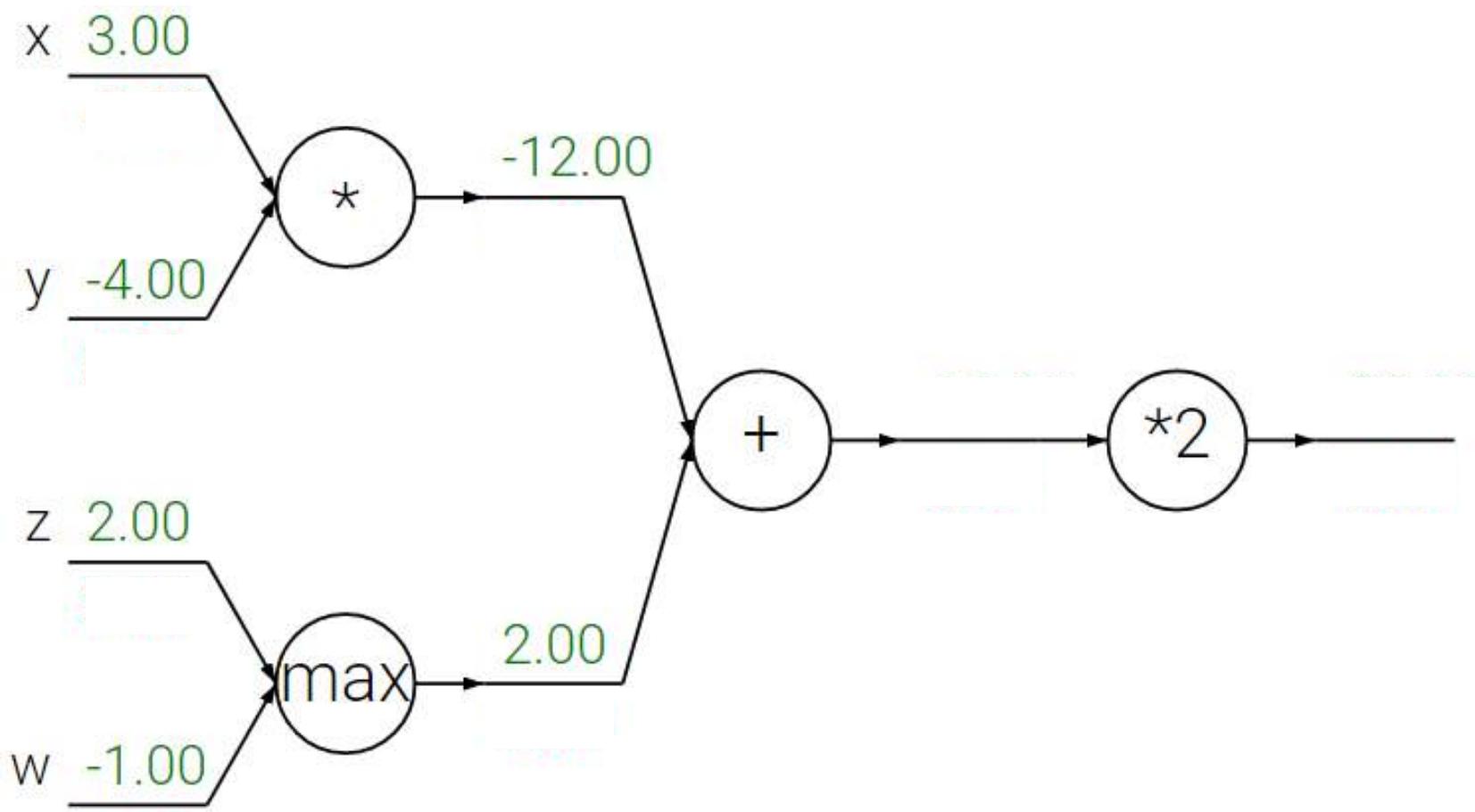
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$



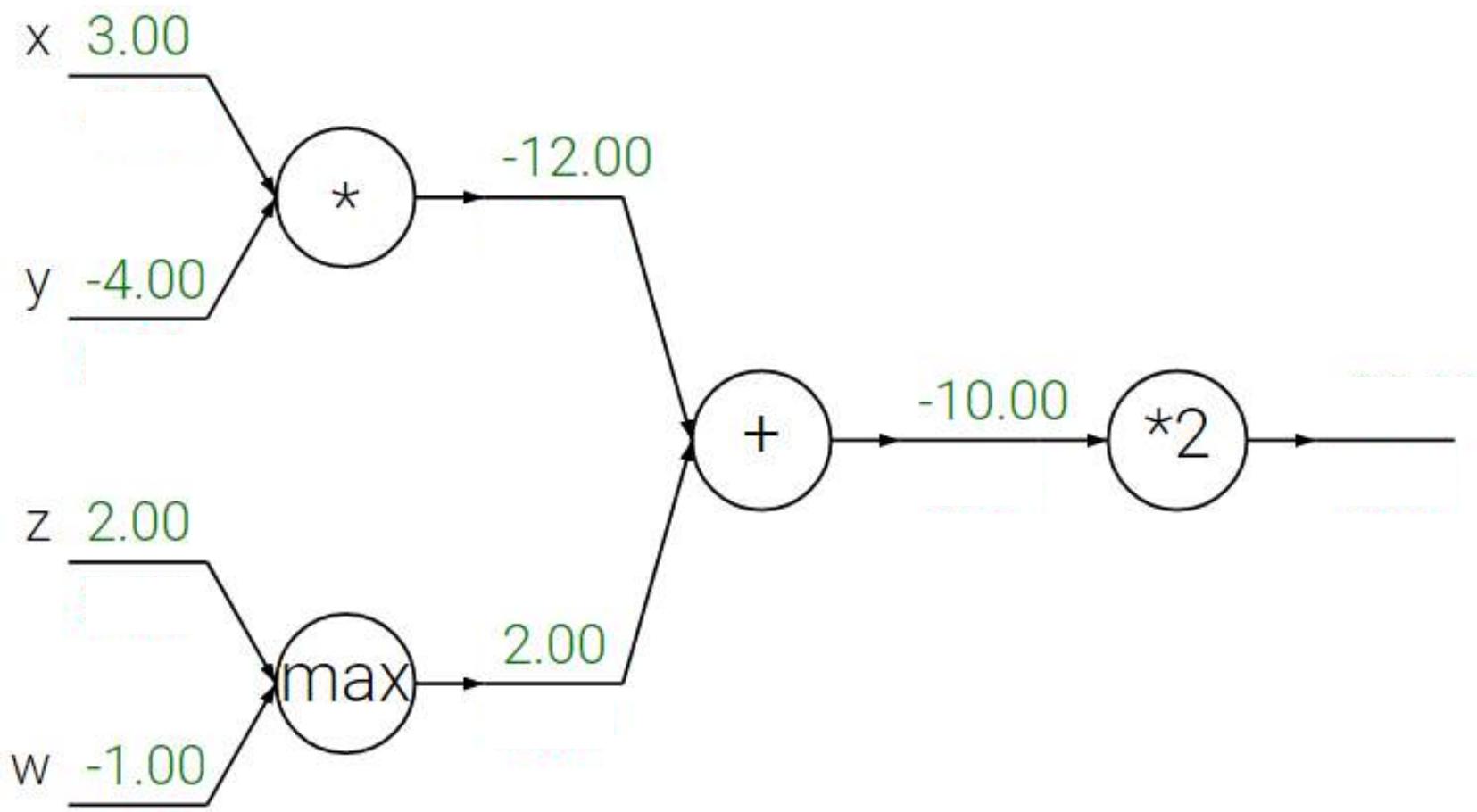
Forward and Backward Pass



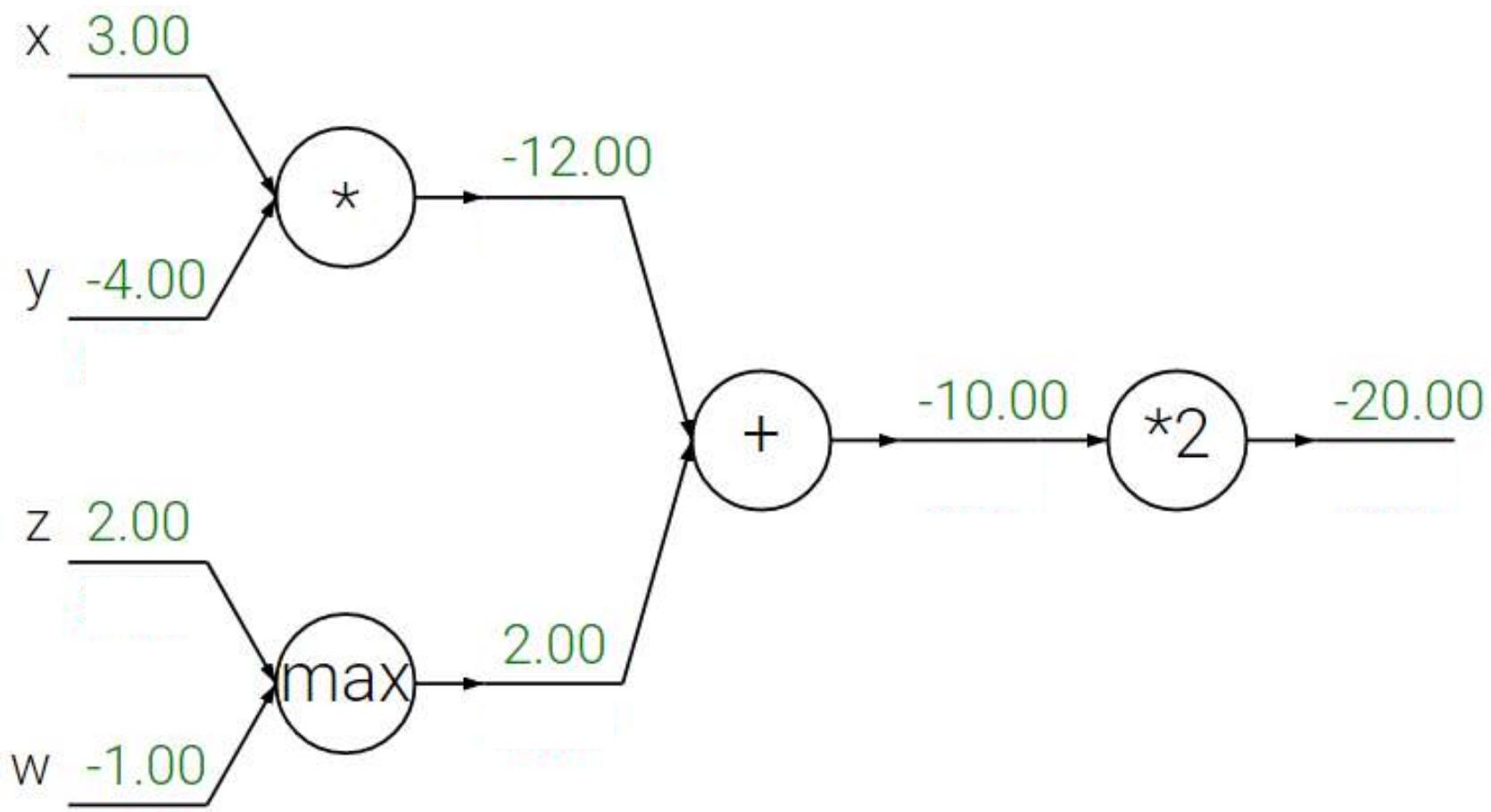
Forward and Backward Pass



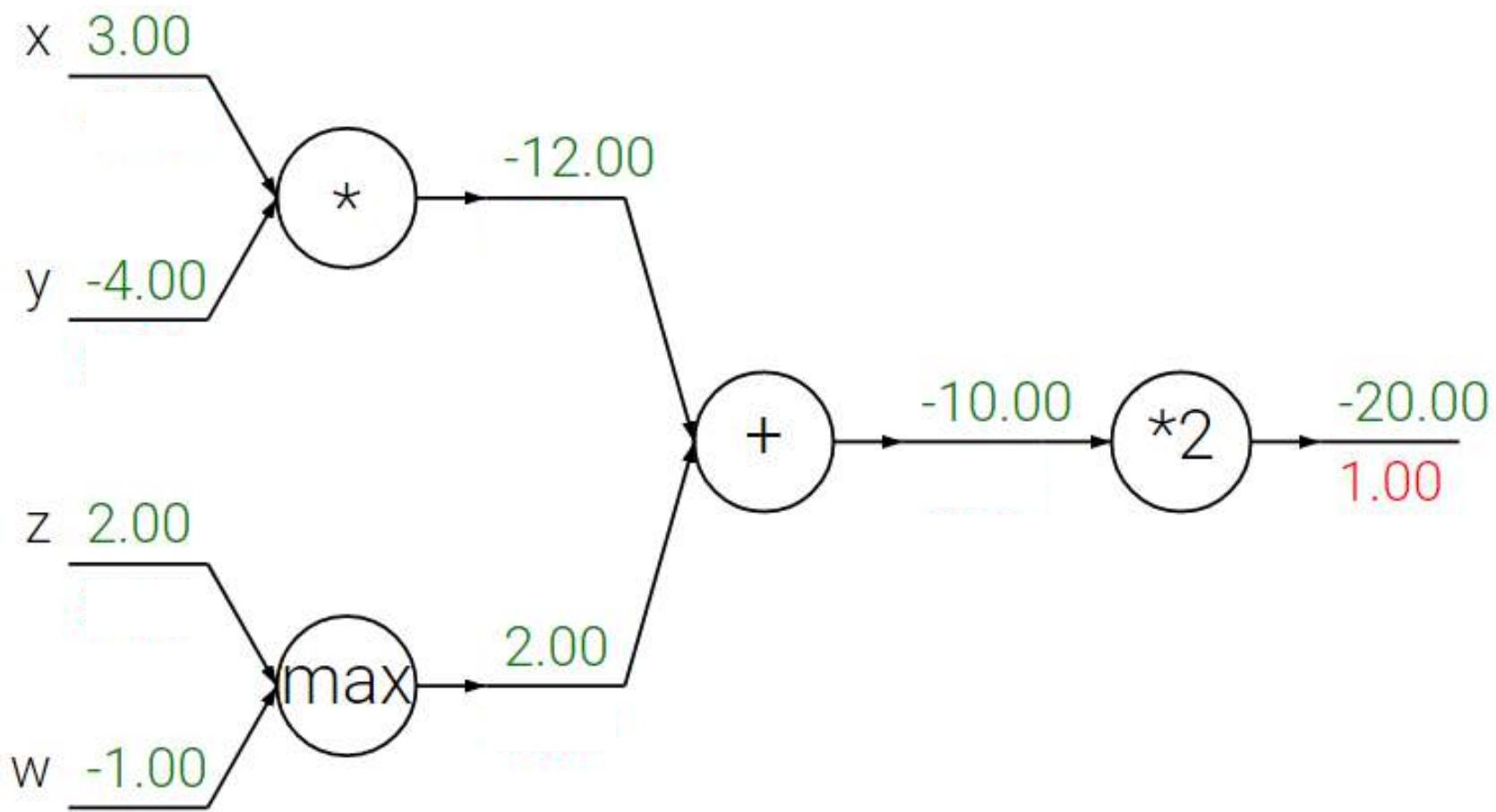
Forward and Backward Pass



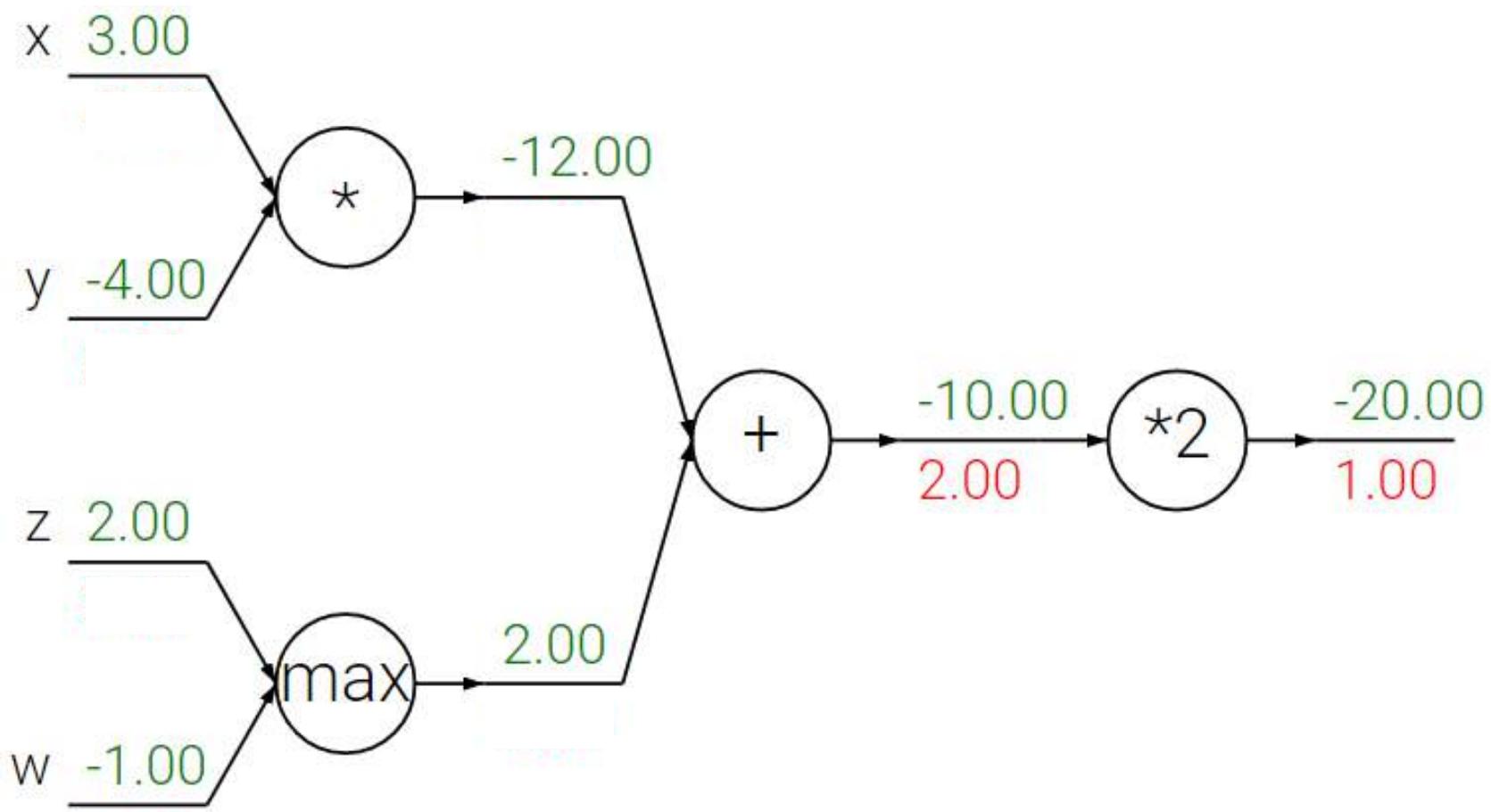
Forward and Backward Pass



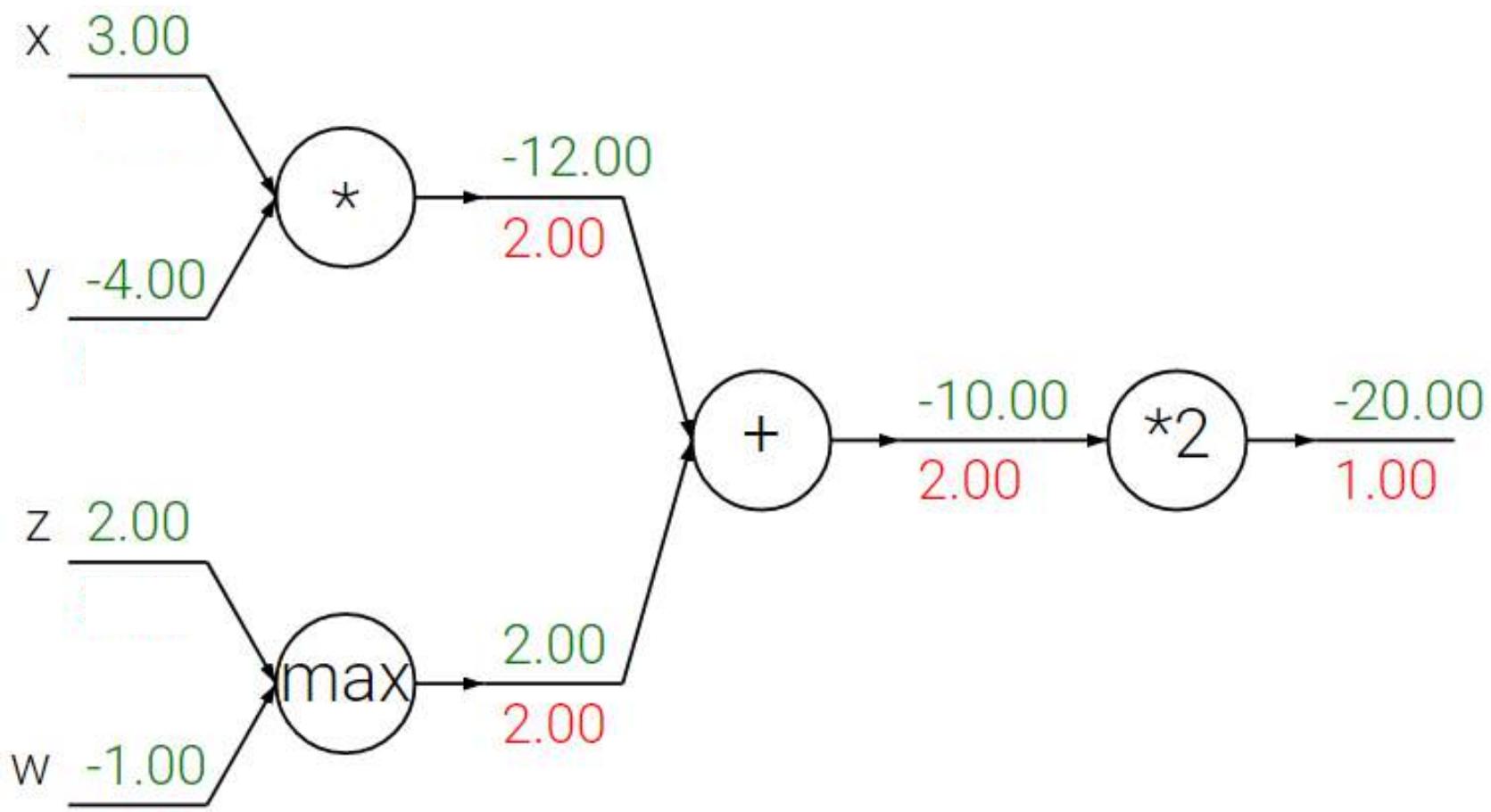
Forward and Backward Pass



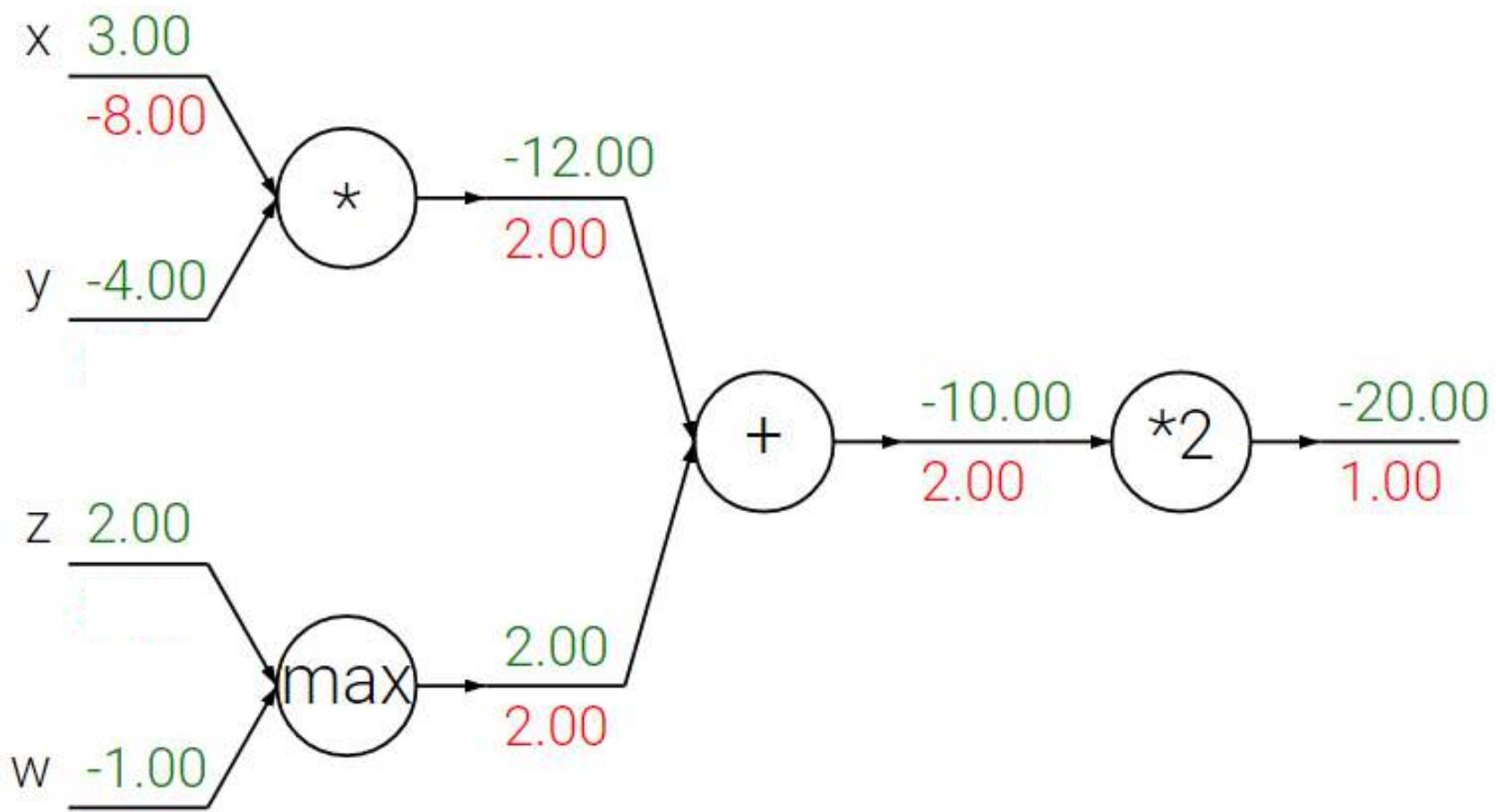
Forward and Backward Pass



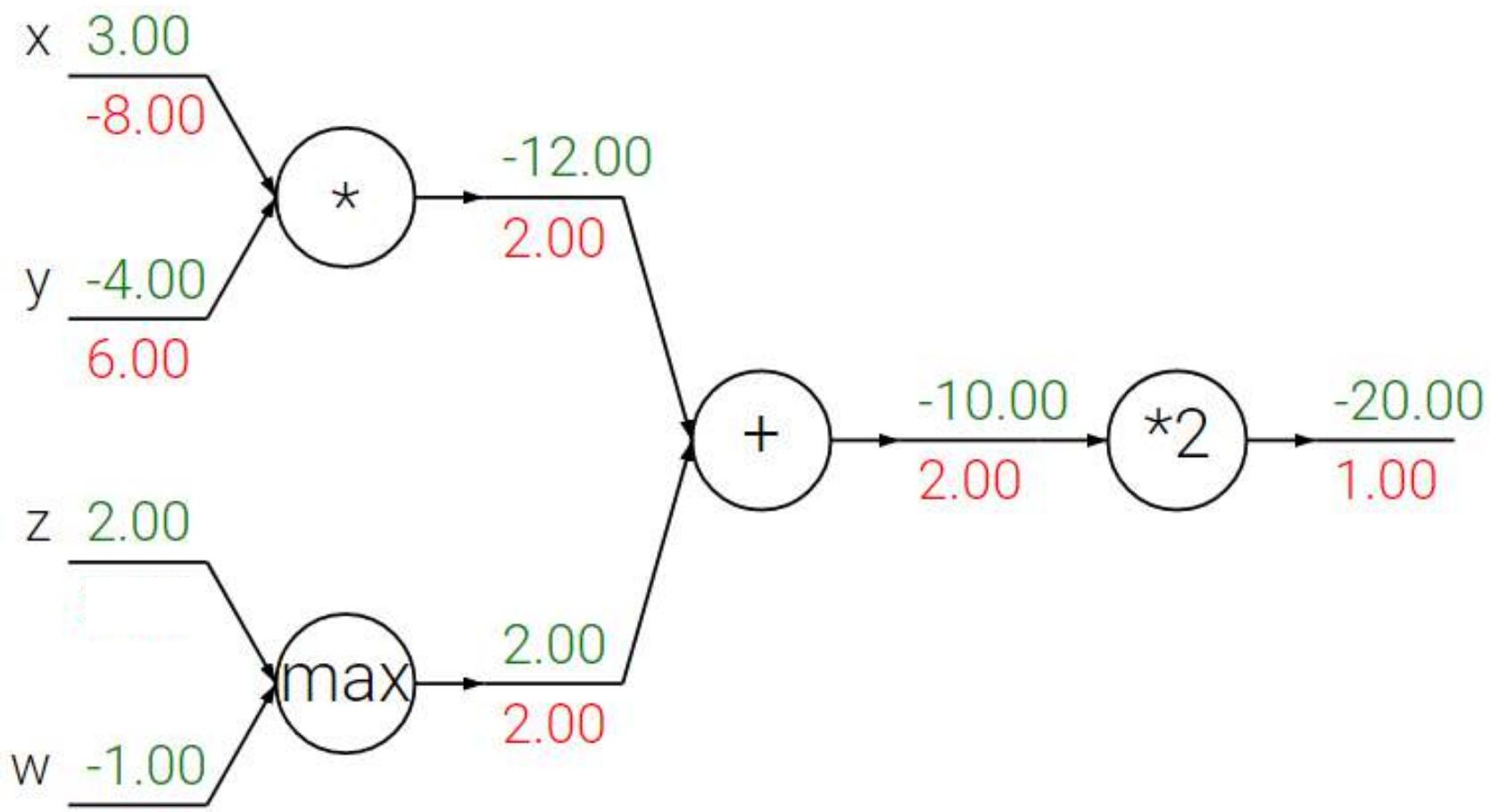
Forward and Backward Pass



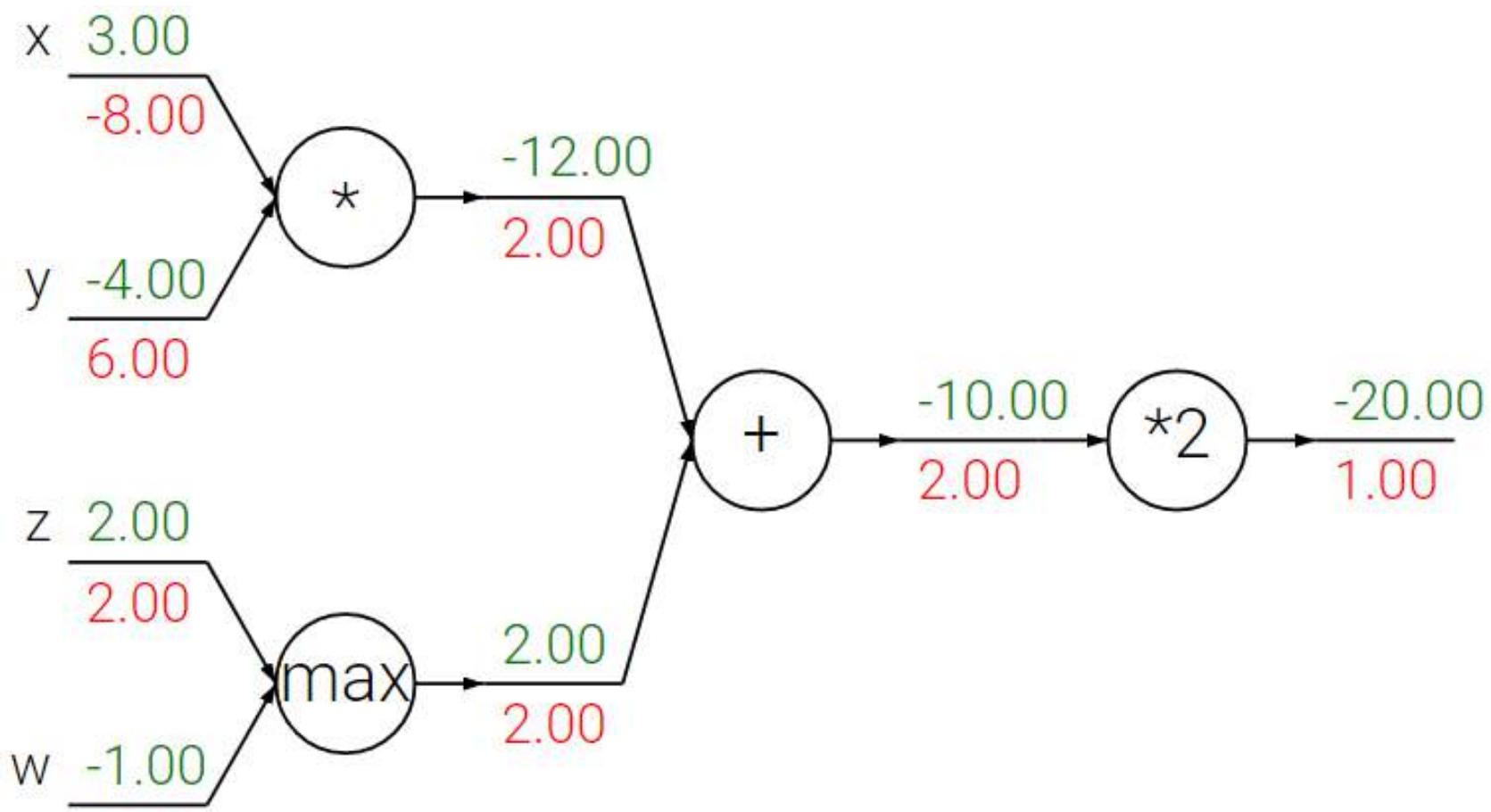
Forward and Backward Pass



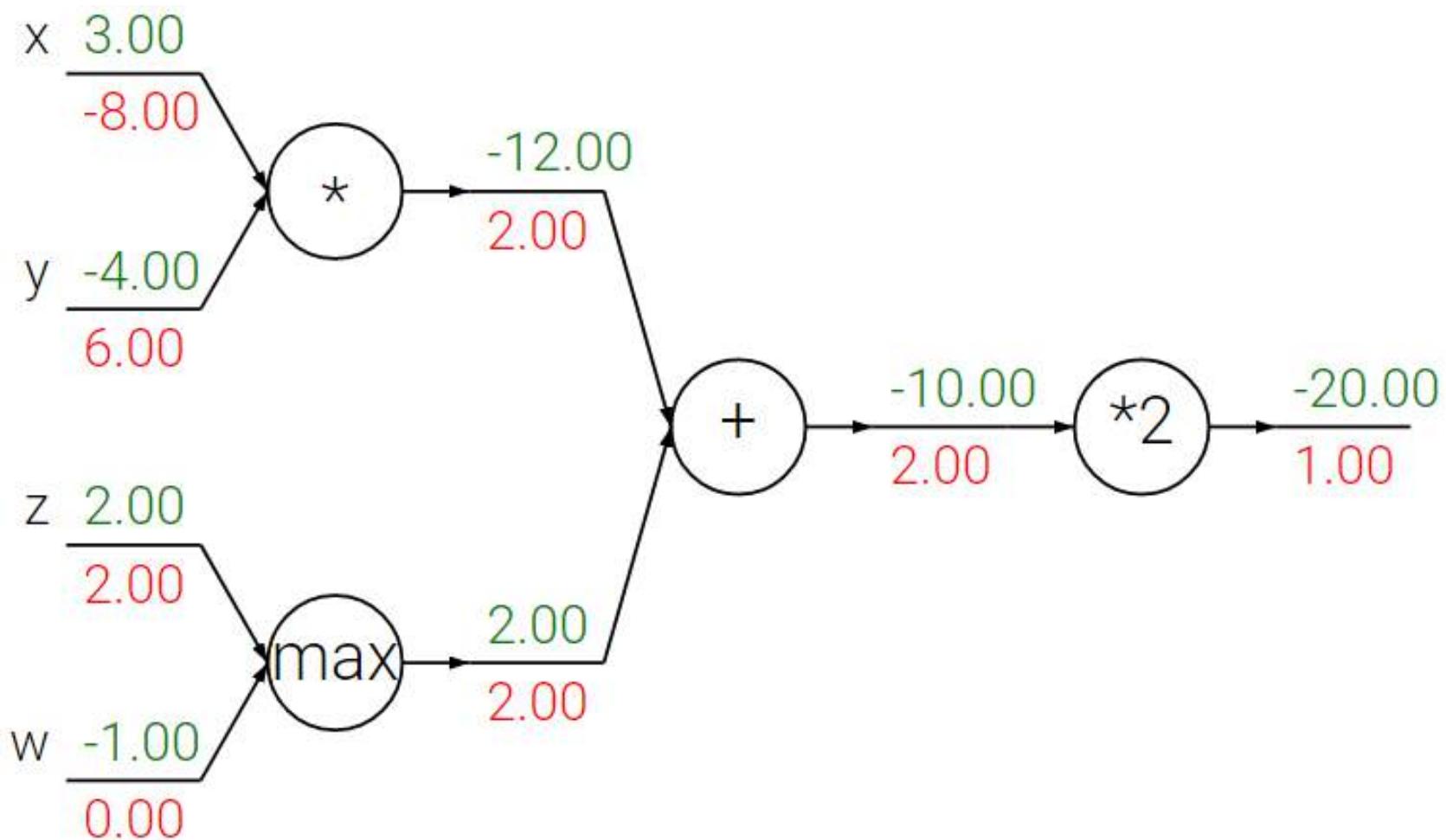
Forward and Backward Pass



Forward and Backward Pass

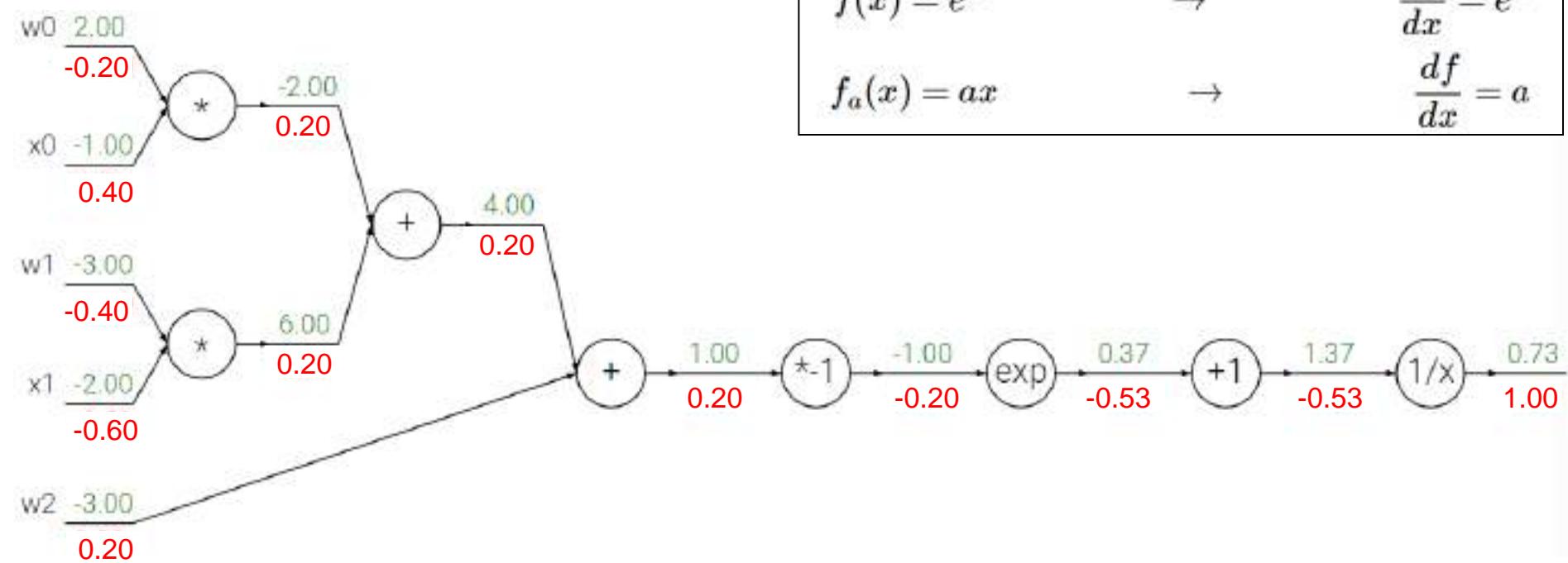


Forward and Backward Pass



Sigmoid example

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$



$f(x) = \frac{1}{x}$	\rightarrow	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	\rightarrow	$\frac{df}{dx} = 1$
$f(x) = e^x$	\rightarrow	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	\rightarrow	$\frac{df}{dx} = a$

SVM Loss: Gradient

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

SVM Loss: Gradient

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

SVM Loss: Gradient

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

$$\nabla_{w_{y_i}} L_i = - \left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

SVM Loss: Gradient

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

$$\nabla_{w_{y_i}} L_i = - \underbrace{\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)}_{\text{Count of the number of classes that didn't meet the desired margin}} x_i$$

SVM Loss: Gradient

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

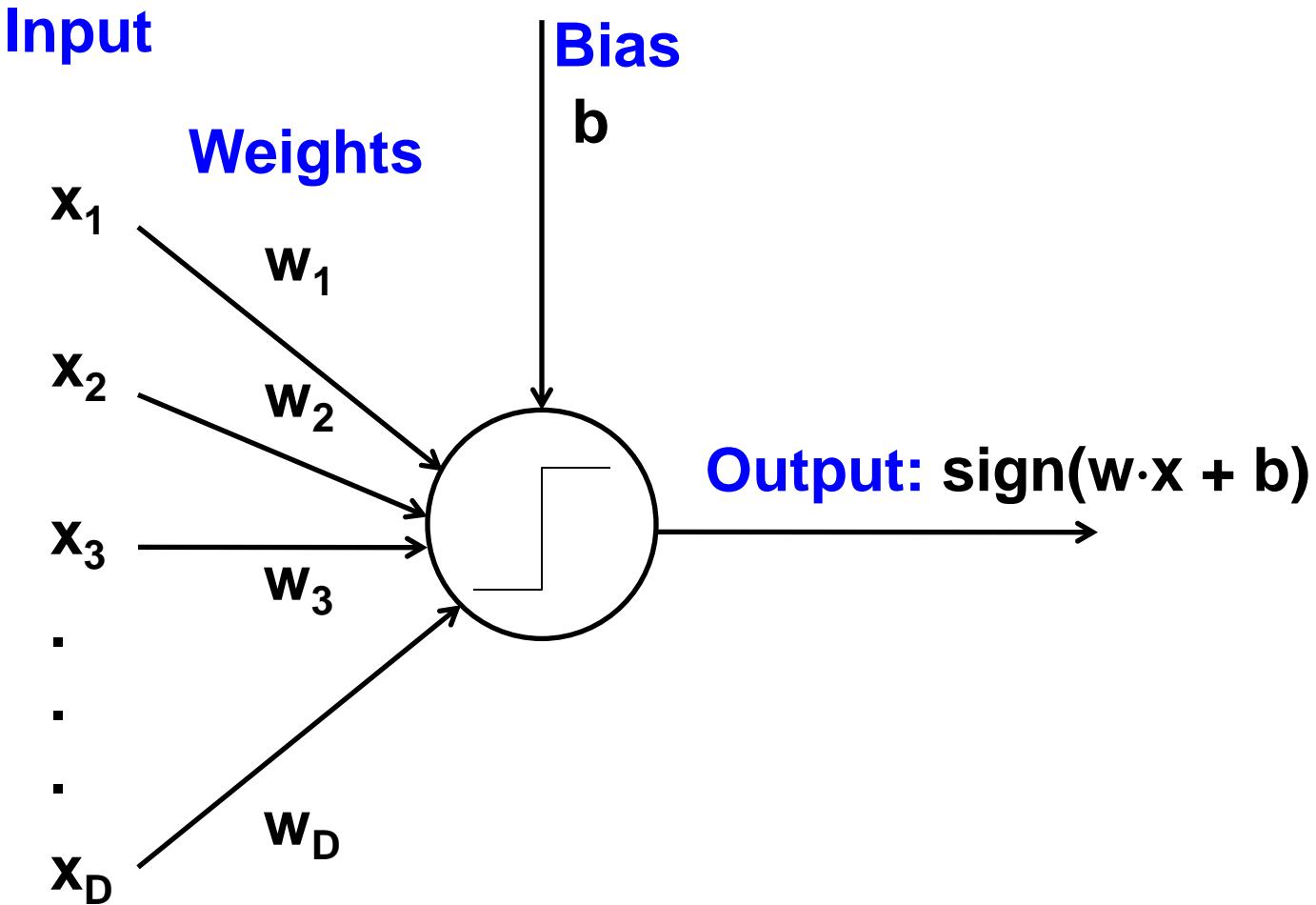
$$\nabla_{w_{y_i}} L_i = - \underbrace{\left(\sum_{j \neq y_i} \mathbf{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)}_{\text{Count of the number of classes that didn't meet the desired margin}} x_i$$

Gradient for the other rows where $j \neq y_i$:

$$\nabla_{w_j} L_i = \mathbf{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

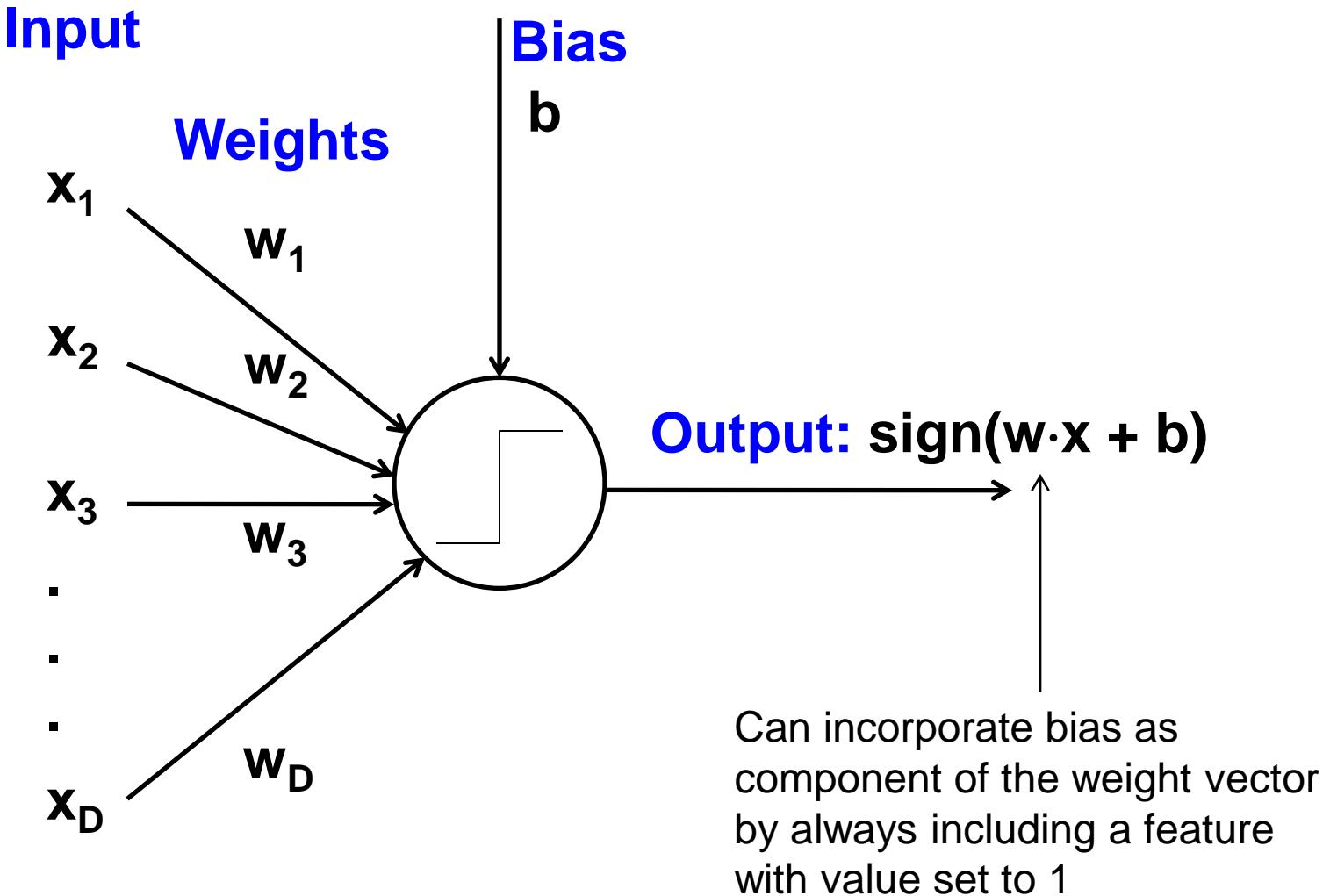
Perceptron

- Supervised learning of binary classifier



Perceptron

- Supervised learning of binary classifier



Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
 - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
 - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$
 - **What happens if y' is correct?**

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
 - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$
 - **What happens if y' is correct?**
 - **Otherwise, if y' is wrong -**

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
 - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$
 - **What happens if y' is correct?**
 - **Otherwise, if y' is wrong** -
 $w_i \leftarrow w_i + \alpha(y - y')x_i$
 - If $y = 1$ and $y' = -1$, w_i will be increased if x_i is positive or decreased if x_i is negative $\rightarrow \mathbf{w} \cdot \mathbf{x}$ will get bigger

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance \mathbf{x} with label y :
 - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
 - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$
 - **What happens if y' is correct?**
 - **Otherwise, if y' is wrong** -
 $w_i \leftarrow w_i + \alpha(y - y')x_i$
 - If $y = 1$ and $y' = -1$, w_i will be increased if x_i is positive or decreased if x_i is negative $\rightarrow \mathbf{w} \cdot \mathbf{x}$ will get bigger
 - If $y = -1$ and $y' = 1$, w_i will be decreased if x_i is positive or increased if x_i is negative $\rightarrow \mathbf{w} \cdot \mathbf{x}$ will get smaller

Single neuron as a linear classifier

Binary Softmax classifier (*Logistic Regression*)

$$\sigma(\sum_i w_i x_i + b)$$

Single neuron as a linear classifier

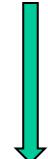
Binary Softmax classifier (*Logistic Regression*)

$$\sigma\left(\sum_i w_i x_i + b\right)$$


Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

Single neuron as a linear classifier

Binary Softmax classifier (*Logistic Regression*)

$$\sigma(\sum_i w_i x_i + b)$$


Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

Probability of the other class would be:

$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

Single neuron as a linear classifier

Binary Softmax classifier (*Logistic Regression*)

$$\sigma\left(\sum_i w_i x_i + b\right)$$


Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

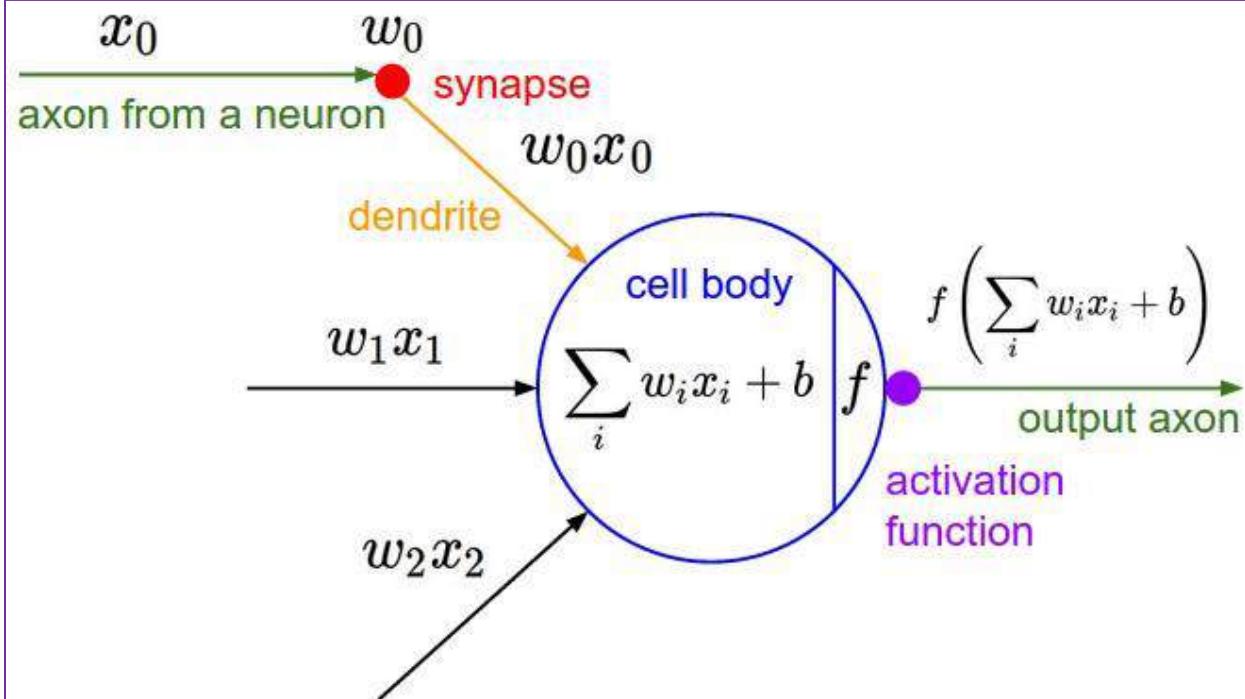
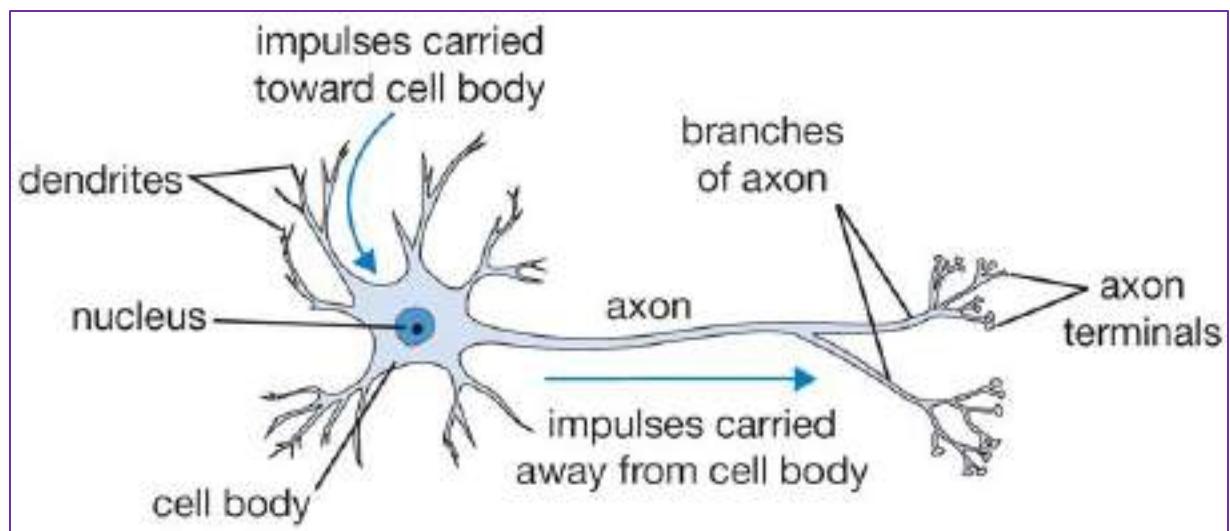
Probability of the other class would be:

$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

Binary SVM classifier.

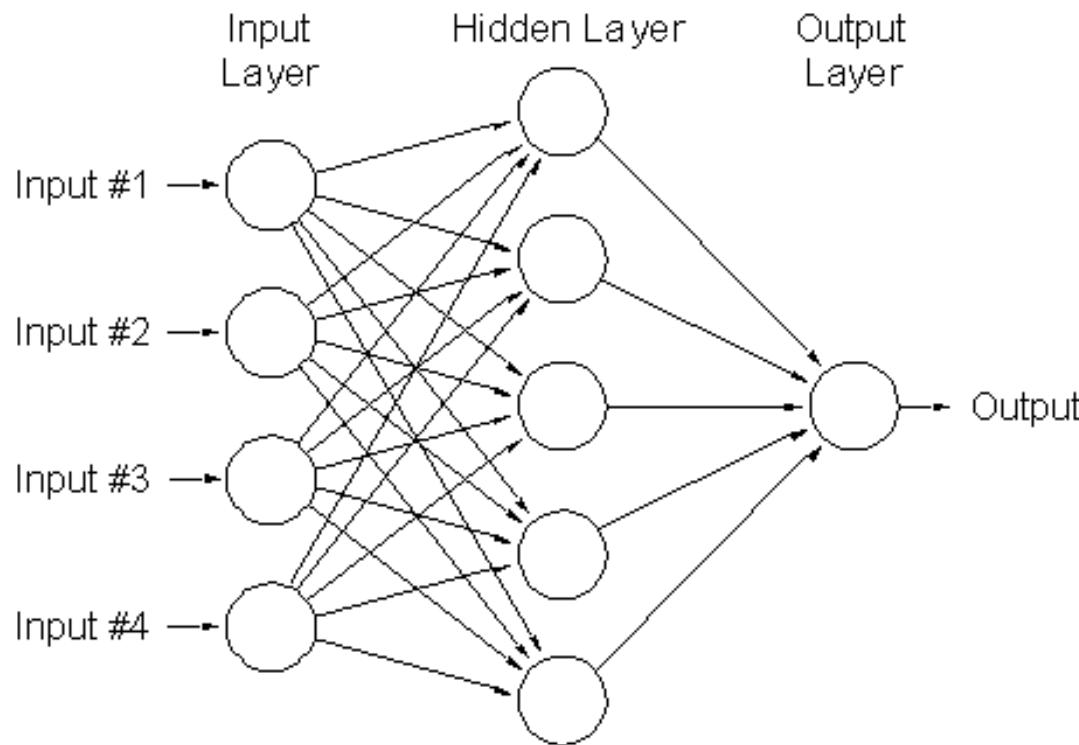
Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Loose inspiration: Human neurons



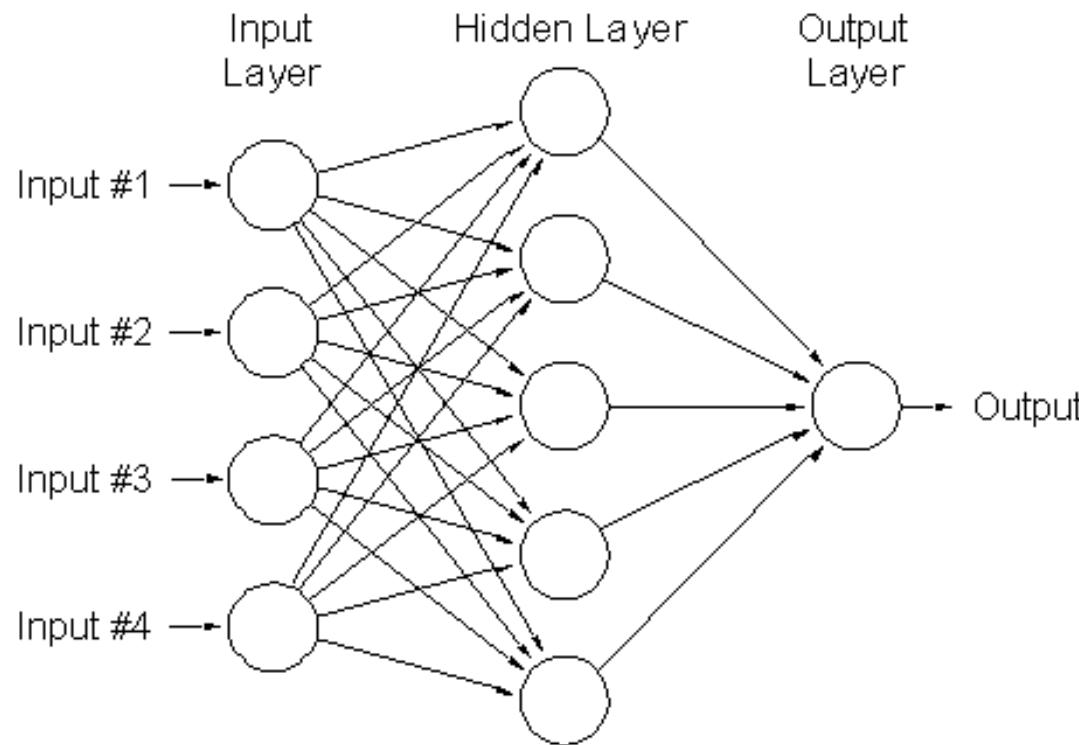
Multi-Layer Neural Networks

- Network with a hidden layer:



Multi-Layer Neural Networks

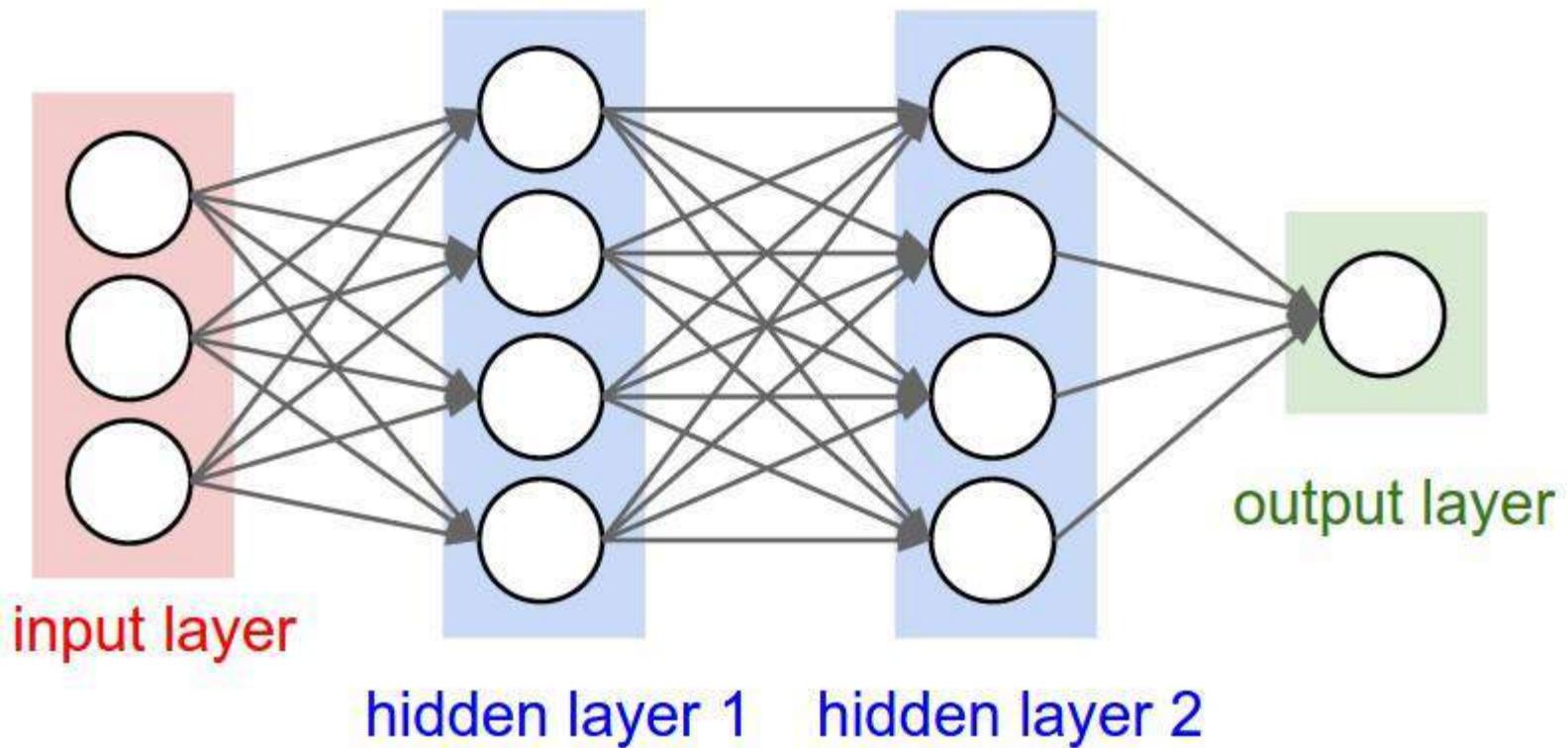
- Network with a hidden layer:



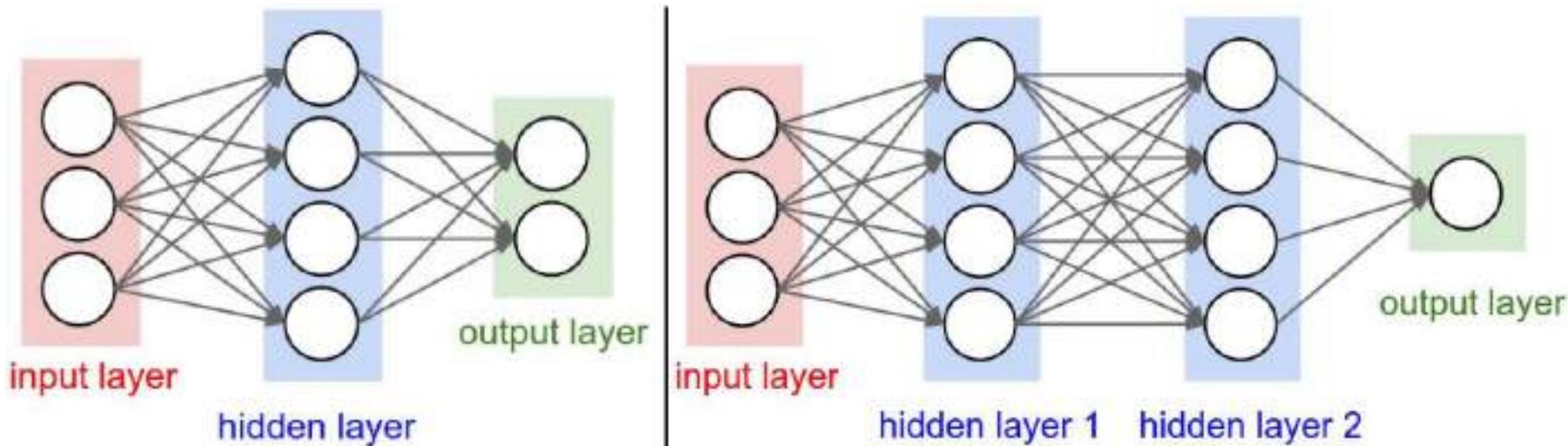
- Can represent nonlinear functions (provided each perceptron has a nonlinearity)

Multi-Layer Neural Networks

- Beyond a single hidden layer:



Sizing neural networks

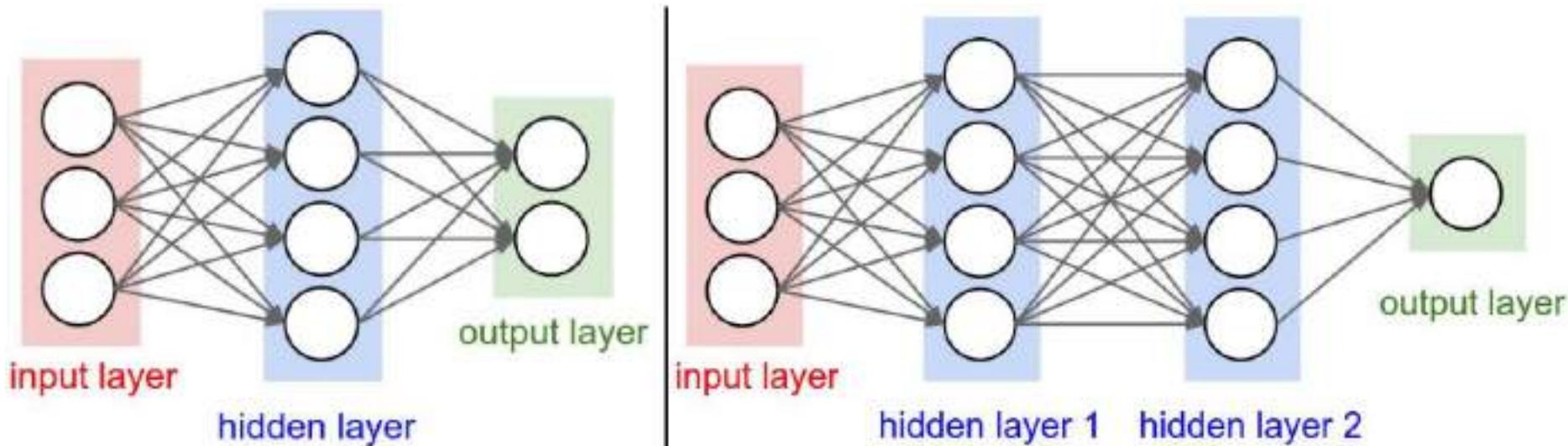


First network (left):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Sizing neural networks

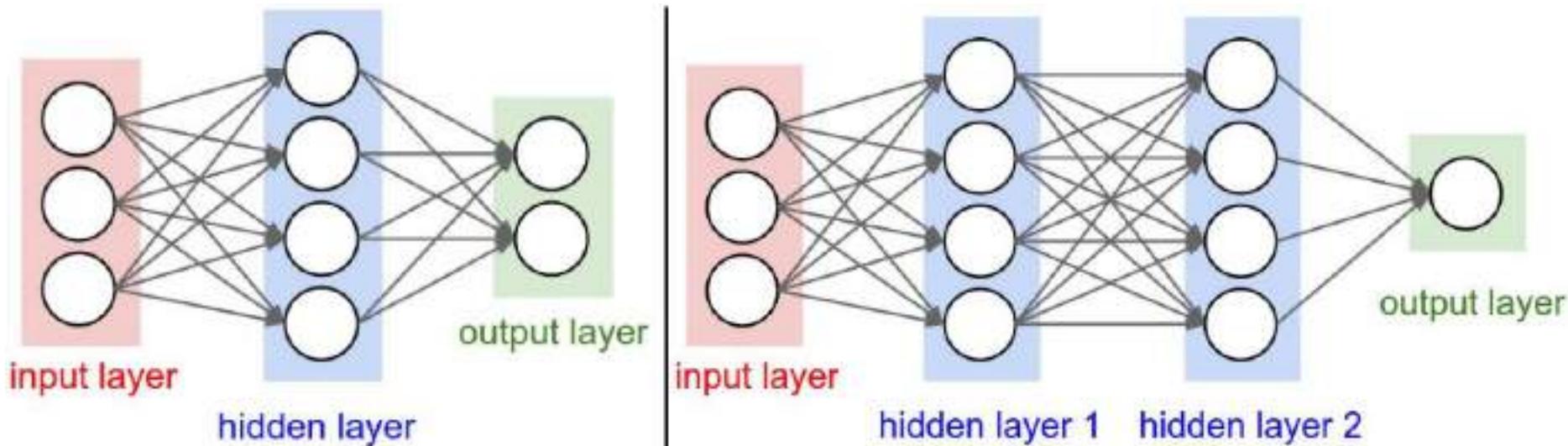


First network (left):

No. of neurons (not counting the inputs): $4 + 2 = 6$

No. of learnable parameters:

Sizing neural networks

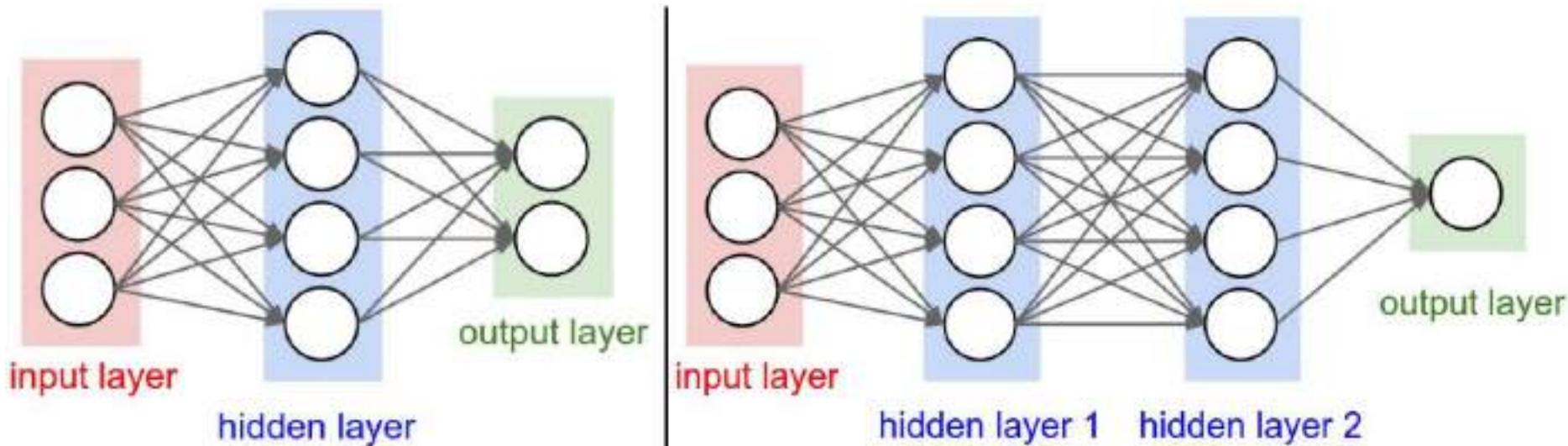


First network (left):

No. of neurons (not counting the inputs): $4 + 2 = 6$

No. of learnable parameters: $[3 \times 4] + [4 \times 2] = 20$ weights +
 $4 + 2 = 6$ biases = 26.

Sizing neural networks



First network (left):

No. of neurons (not counting the inputs): $4 + 2 = 6$

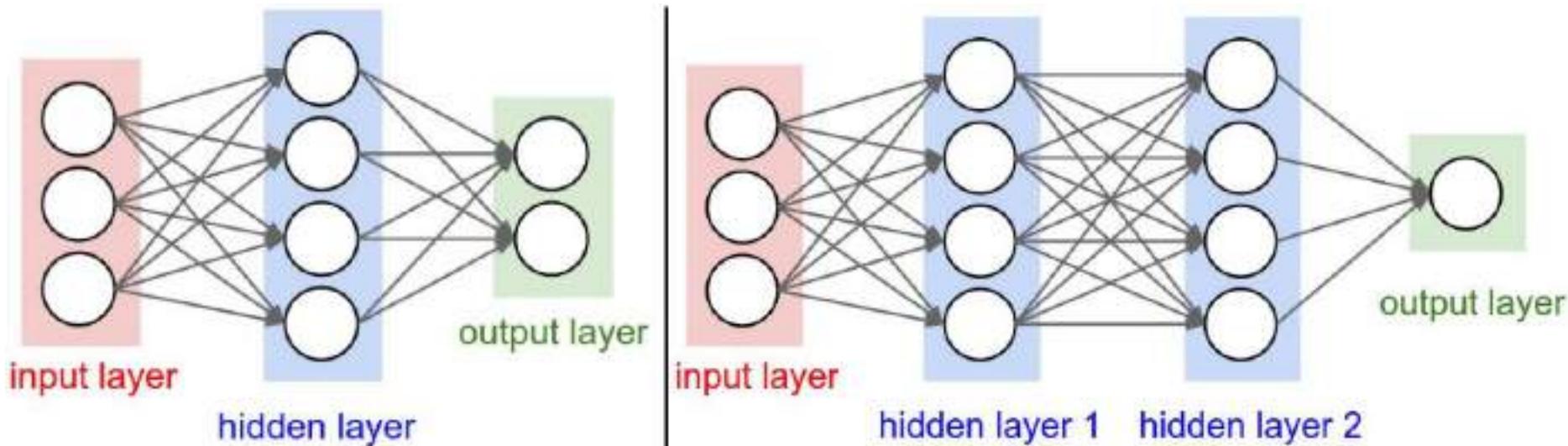
No. of learnable parameters: $[3 \times 4] + [4 \times 2] = 20$ weights +
 $4 + 2 = 6$ biases = 26.

Second network (right):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Sizing neural networks



First network (left):

No. of neurons (not counting the inputs): $4 + 2 = 6$

No. of learnable parameters: $[3 \times 4] + [4 \times 2] = 20$ weights +
 $4 + 2 = 6$ biases = 26.

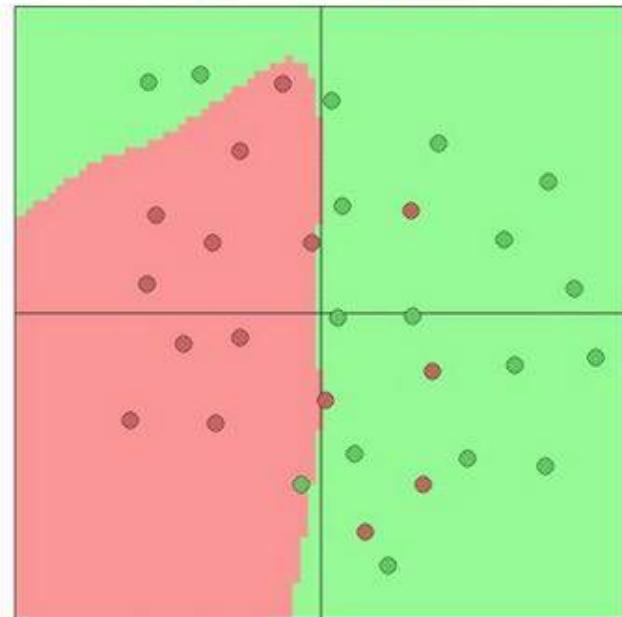
Second network (right):

No. of neurons (not counting the inputs): $4 + 4 + 1 = 9$

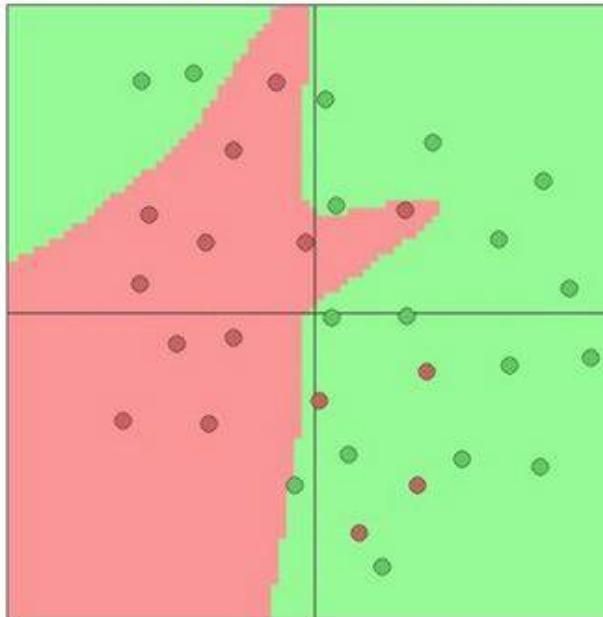
No. of learnable parameters: $[3 \times 4] + [4 \times 4] + [4 \times 1] = 32$ weights +
 $4 + 4 + 1 = 9$ biases = 41.

Multi-Layer Neural Networks

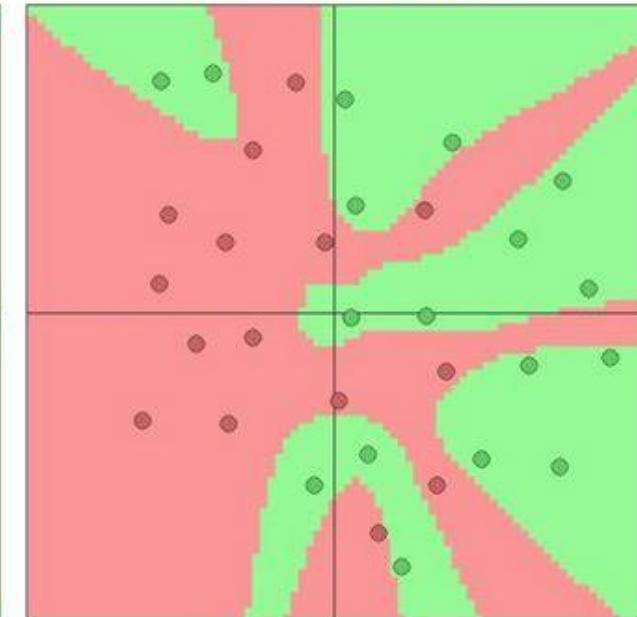
3 hidden neurons



6 hidden neurons



20 hidden neurons



Training of multi-layer networks

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^N (y_j - f_{\mathbf{w}}(\mathbf{x}_j))^2$$

Training of multi-layer networks

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^N (y_j - f_{\mathbf{w}}(\mathbf{x}_j))^2$$

- Update weights by **gradient descent**: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

Training of multi-layer networks

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^N (y_j - f_{\mathbf{w}}(\mathbf{x}_j))^2$$

- Update weights by **gradient descent**: $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$
- **Back-propagation**: gradients are computed in the direction from output to input layers and combined using chain rule

Neural networks: Pros and cons

- **Pros**
 - Flexible and general function approximation framework
 - Can build extremely powerful models by adding more layers

Neural networks: Pros and cons

- **Pros**
 - Flexible and general function approximation framework
 - Can build extremely powerful models by adding more layers
- **Cons**
 - Hard to analyze theoretically (e.g., training is prone to local optima)
 - Huge amount of training data, computing power may be required to get good performance
 - The space of implementation choices are huge (network architectures, parameters)

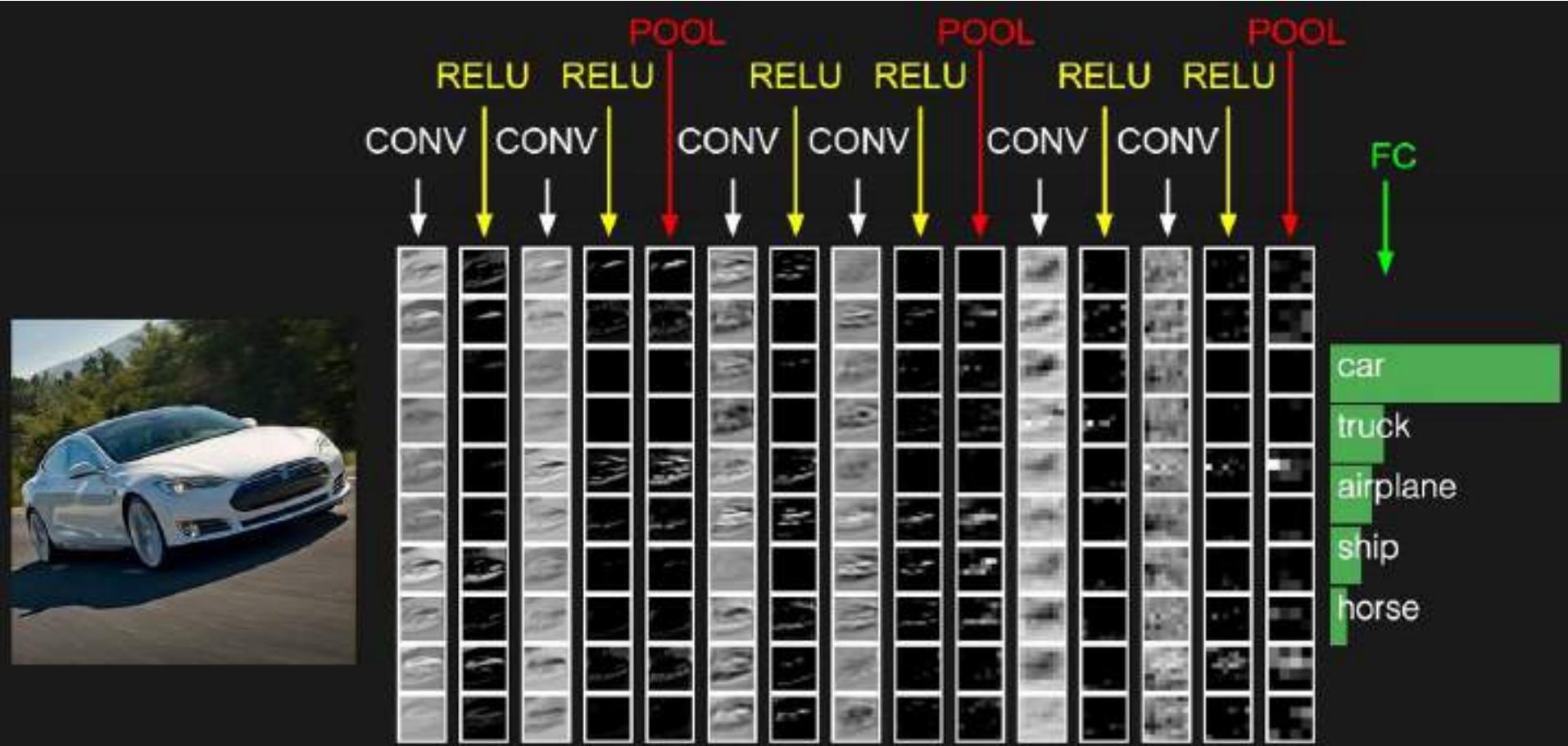
Acknowledgements

Thanks to the following researchers for making their teaching/research material online

- Forsyth
- Steve Seitz
- Noah Snavely
- J.B. Huang
- Derek Hoiem
- D. Lowe
- A. Bobick
- S. Lazebnik
- K. Grauman
- R. Zaleski
- Antonio Torralba
- Rob Fergus
- Leibe
- And many more

Next Lecture

Convolutional Neural Networks



Computer Vision

Convolutional Neural Network

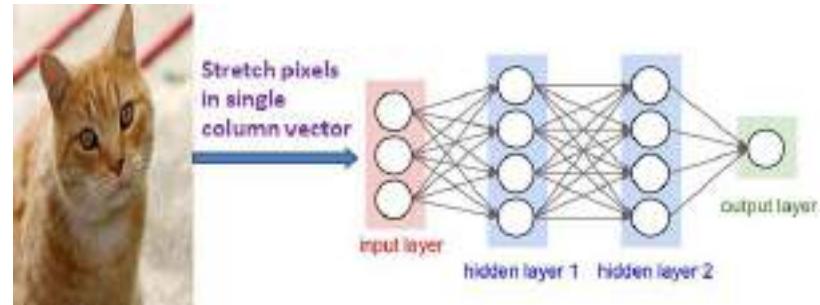
Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**

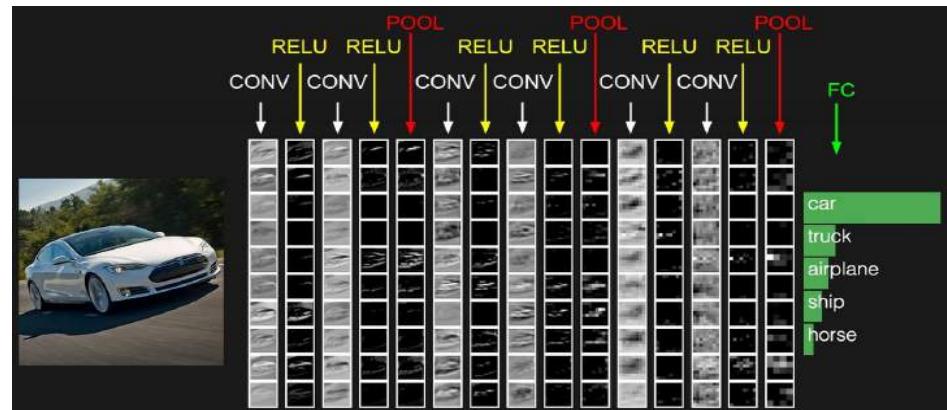


This Class

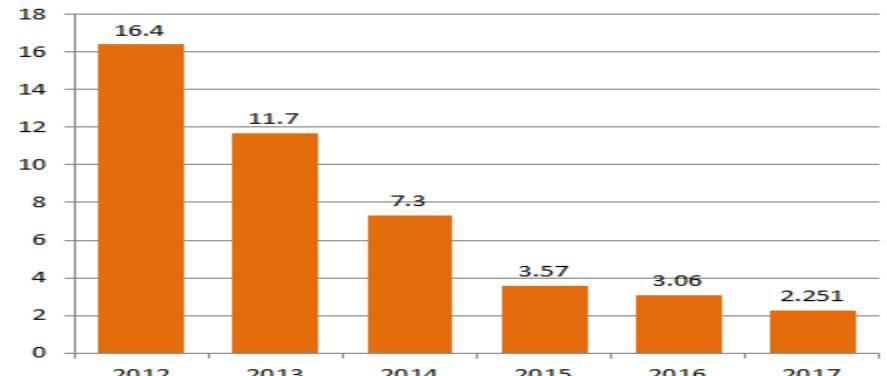
- Neural Network and Image
 - Dimensionality
 - Local relationship



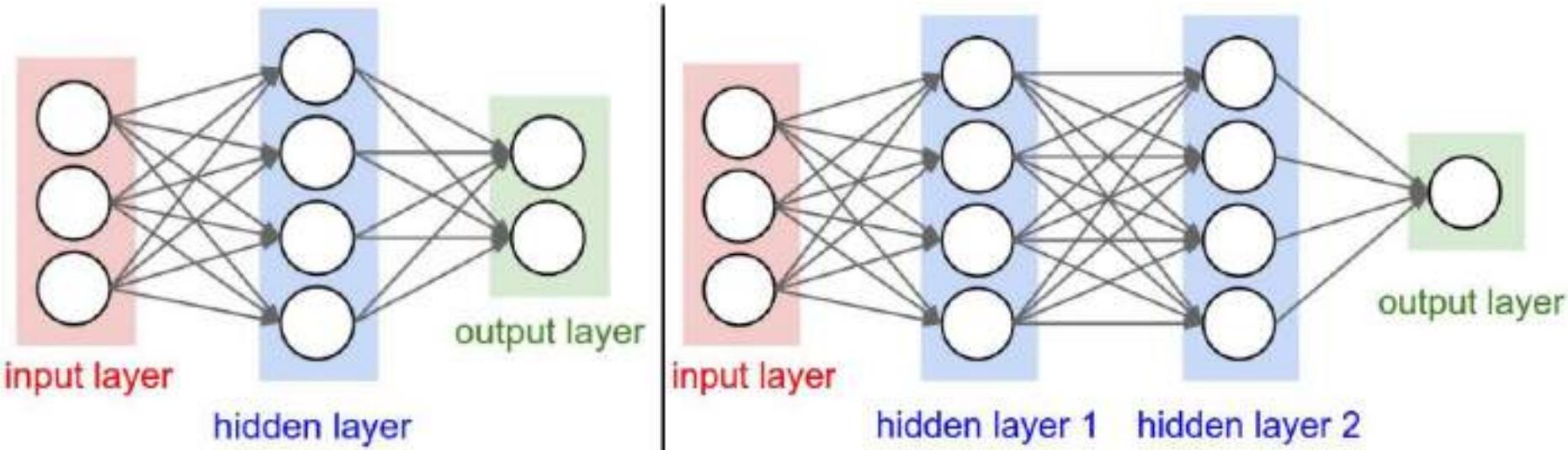
- Convolutional Neural Network (CNN)
 - Convolution Layer
 - Non-linearity Layer
 - Pooling Layer
 - Fully Connected Layer
 - Classification Layer



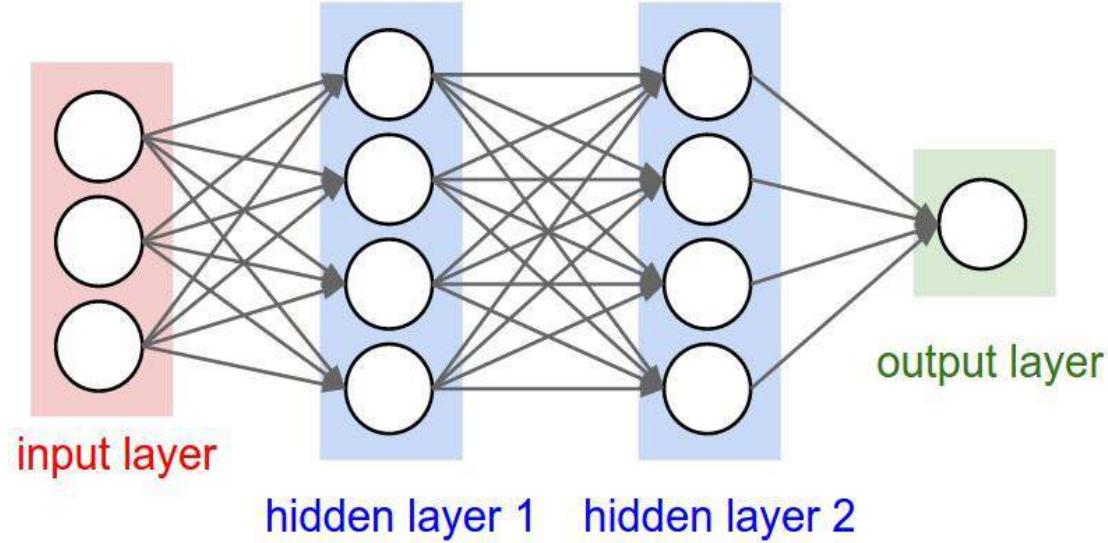
- ImageNet Challenge
 - Progress
 - Human Level Performance



Neural Networks



Multi-layer Neural Network & Image

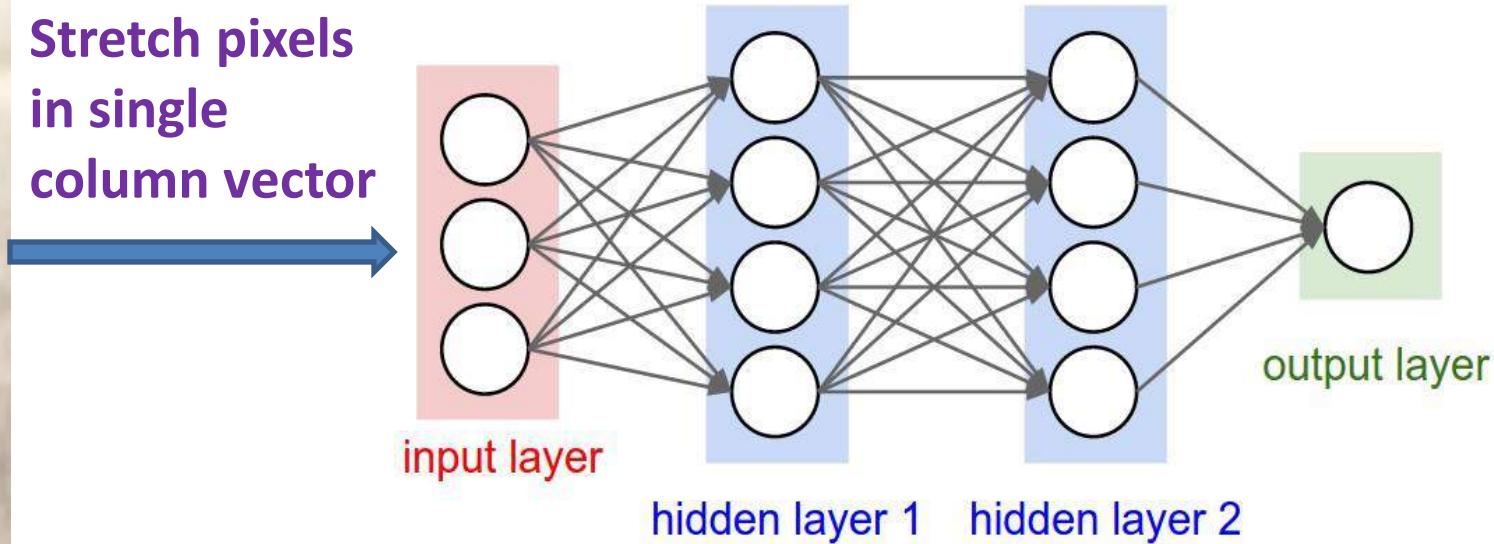


How to apply NN over Image?

Multi-layer Neural Network & Image



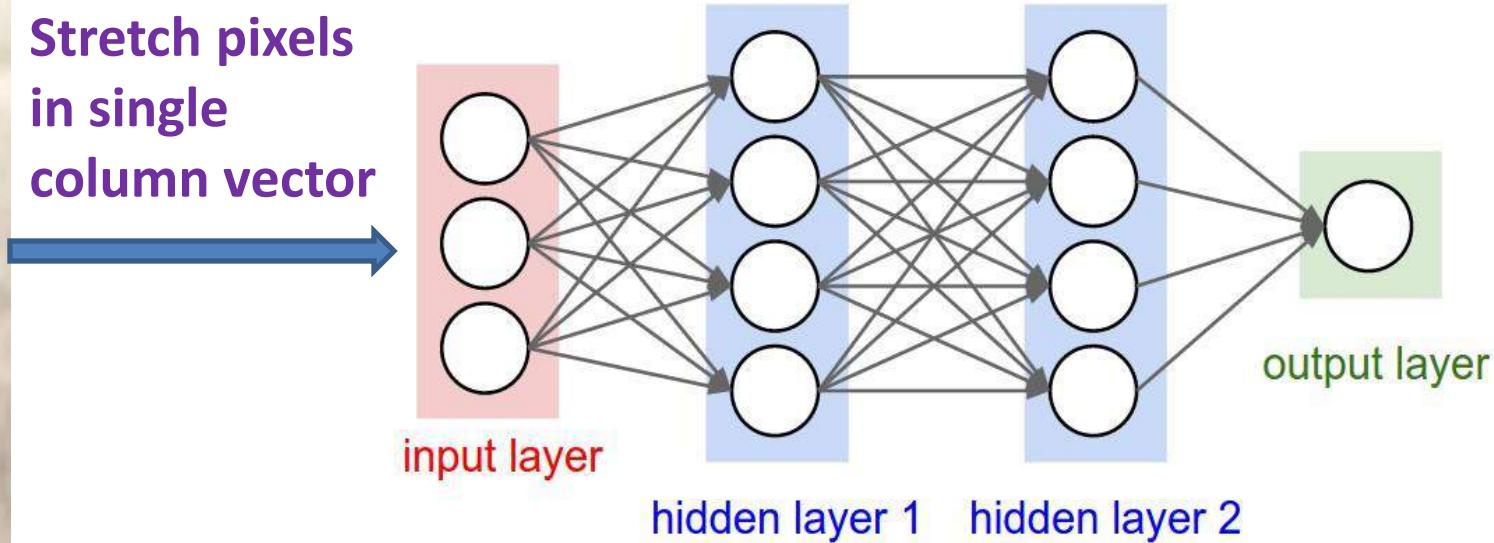
Stretch pixels
in single
column vector



Multi-layer Neural Network & Image



Stretch pixels
in single
column vector

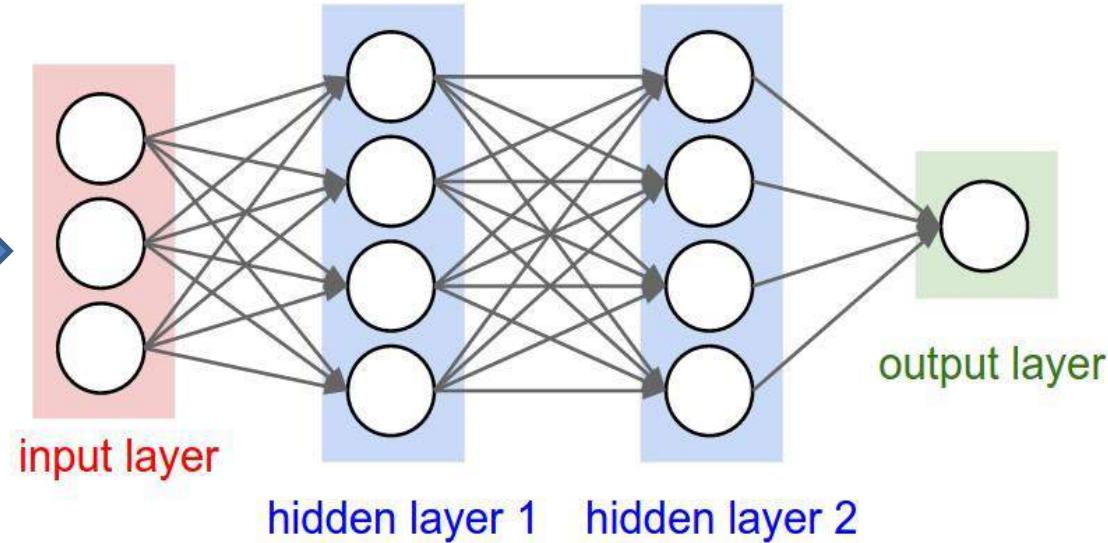


Problems ?

Multi-layer Neural Network & Image



Stretch pixels
in single
column vector



Problems:

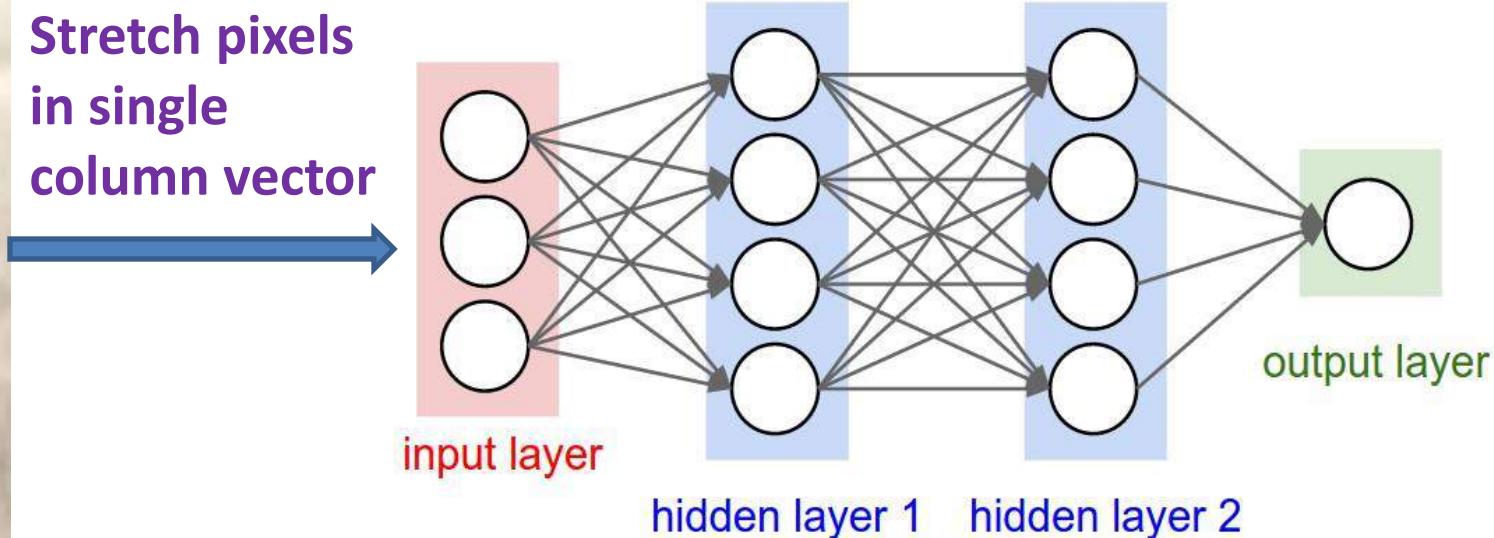
High dimensionality

Local relationship

Multi-layer Neural Network & Image



Stretch pixels
in single
column vector



Problems:

High dimensionality

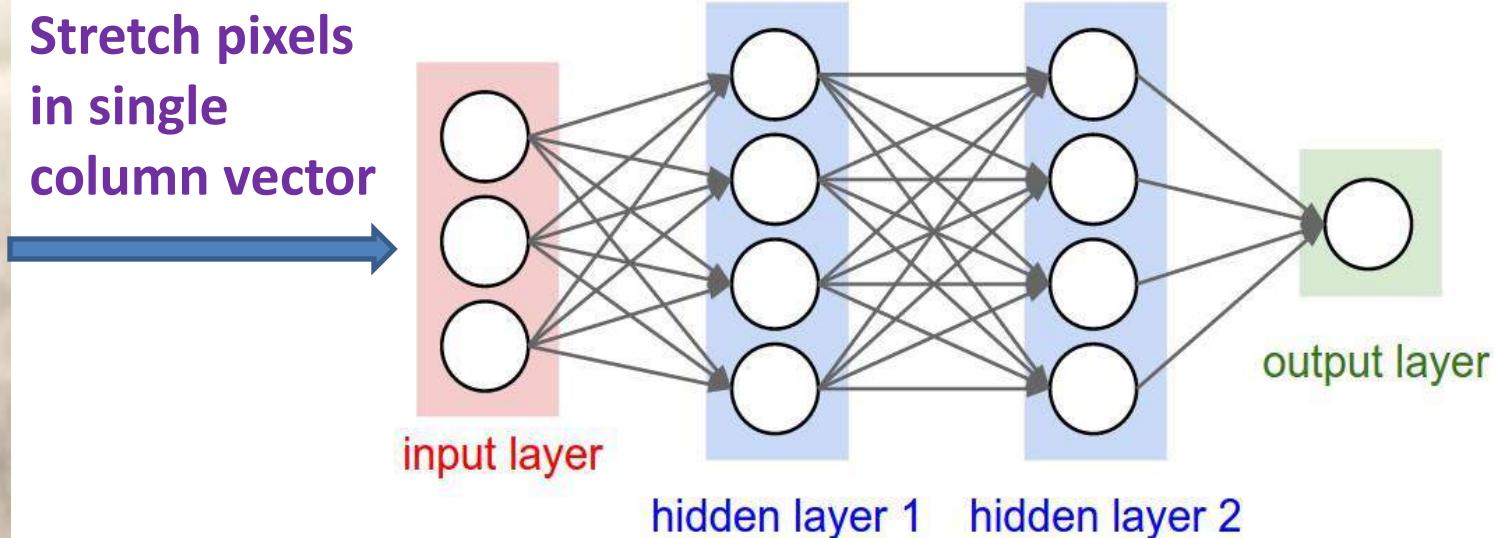
Local relationship

Solution ?

Multi-layer Neural Network & Image



Stretch pixels
in single
column vector



Problems:

High dimensionality

Local relationship

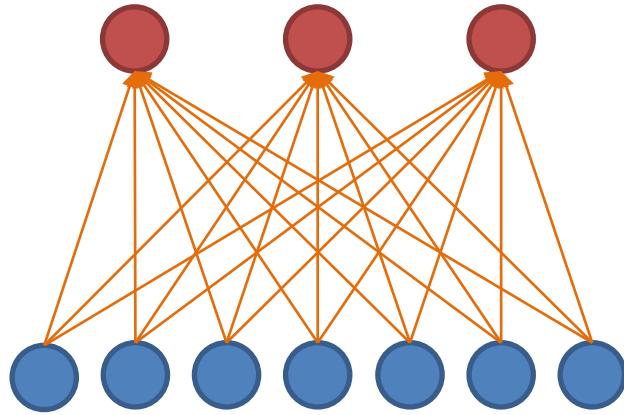
Solution:

Convolutional Neural Network

Convolutional Neural Networks

- Also known as
 - CNN,
 - ConvNet,
 - DCN
- CNN = a multi-layer neural network with
 1. Local connectivity
 2. Weight sharing

CNN: Local Connectivity

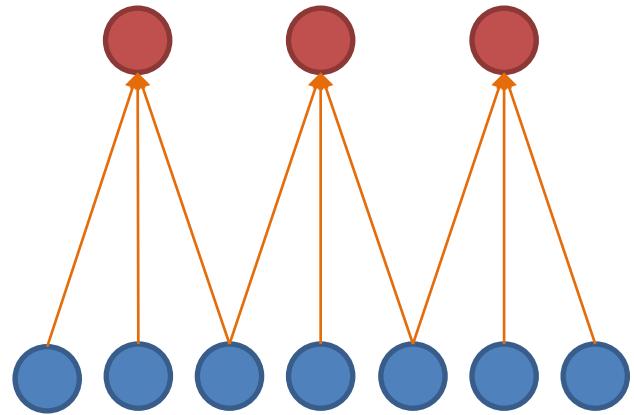


Global connectivity

- # input units (neurons): 7
- # hidden units: 3

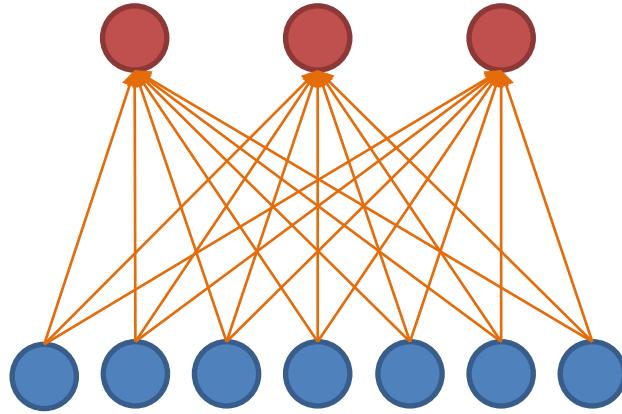
Hidden layer

Input layer



Local connectivity

CNN: Local Connectivity

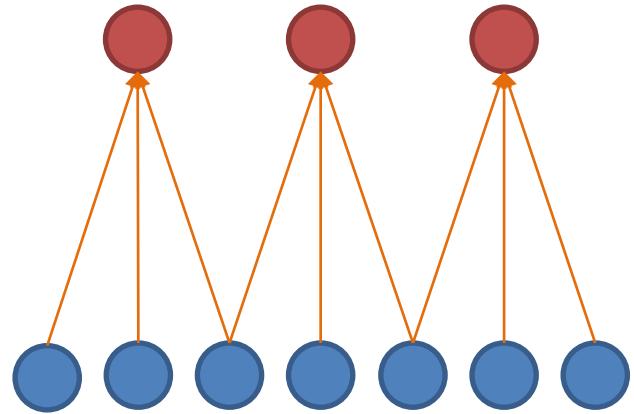


Global connectivity

- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
 - Global connectivity: ?
 - Local connectivity: ?

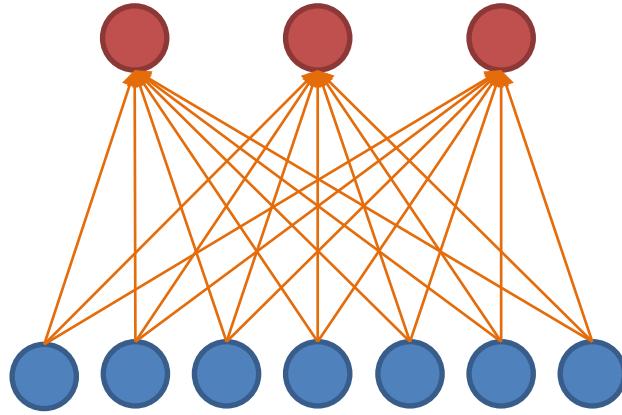
Hidden layer

Input layer



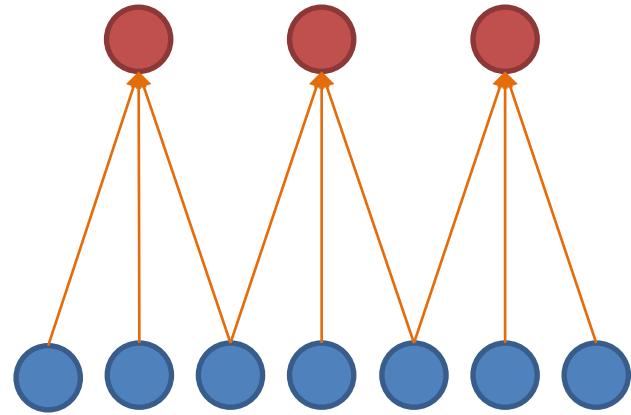
Local connectivity

CNN: Local Connectivity



Hidden layer

Input layer

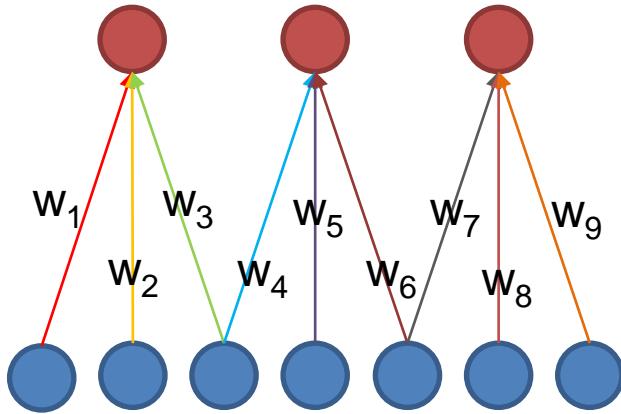


Local connectivity

Global connectivity

- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
 - Global connectivity: $3 \times 7 = 21$
 - Local connectivity: $3 \times 3 = 9$

CNN: Weight Sharing

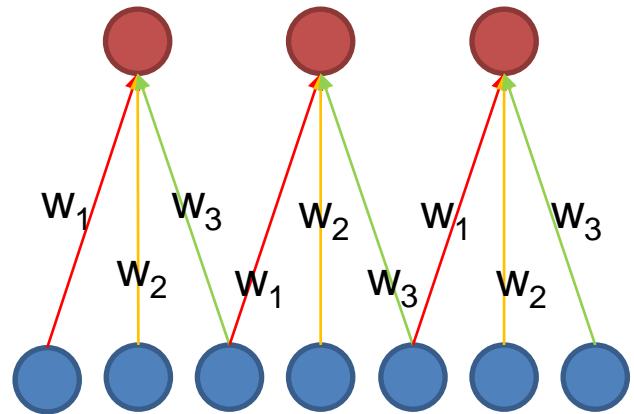


Without weight sharing

- # input units (neurons): 7
- # hidden units: 3

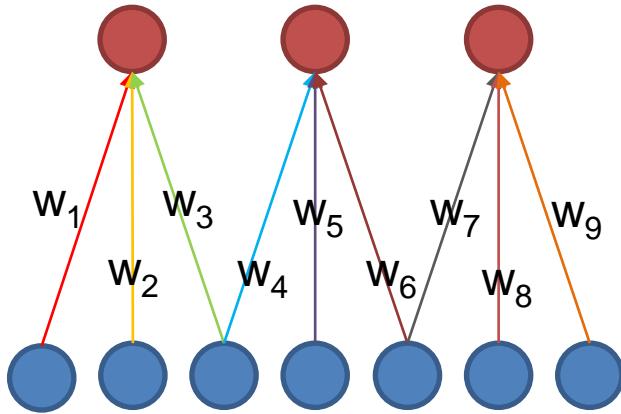
Hidden layer

Input layer



With weight sharing

CNN: Weight Sharing

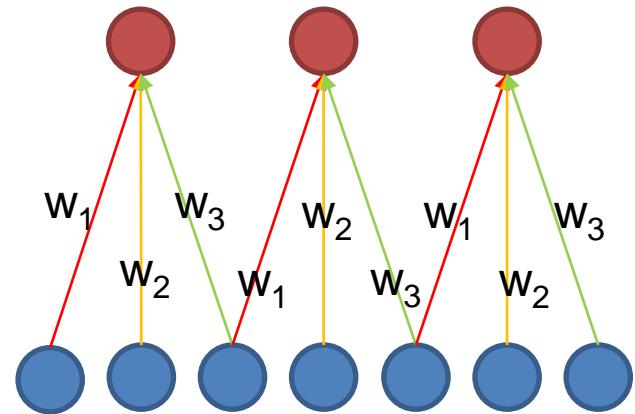


Without weight sharing

- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
 - Without weight sharing: ?
 - With weight sharing : ?

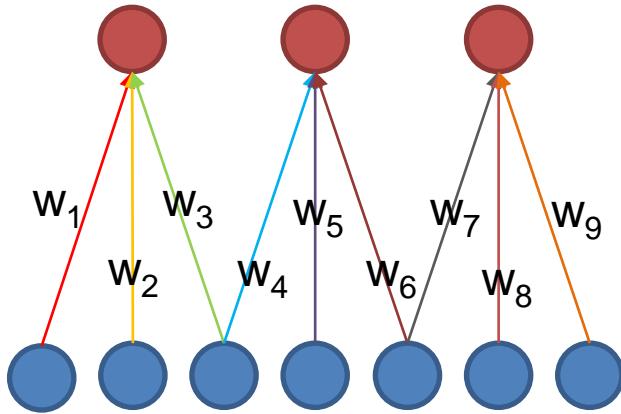
Hidden layer

Input layer



With weight sharing

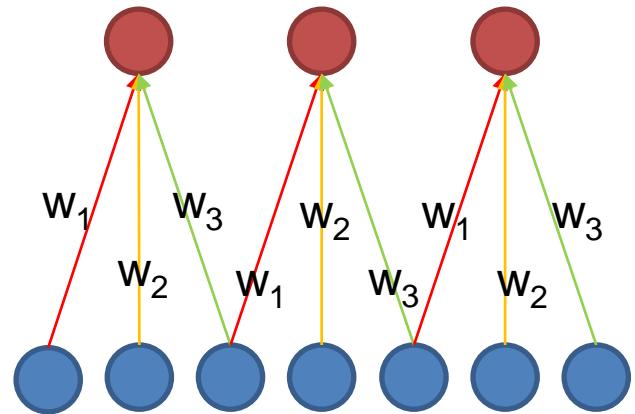
CNN: Weight Sharing



Without weight sharing

Hidden layer

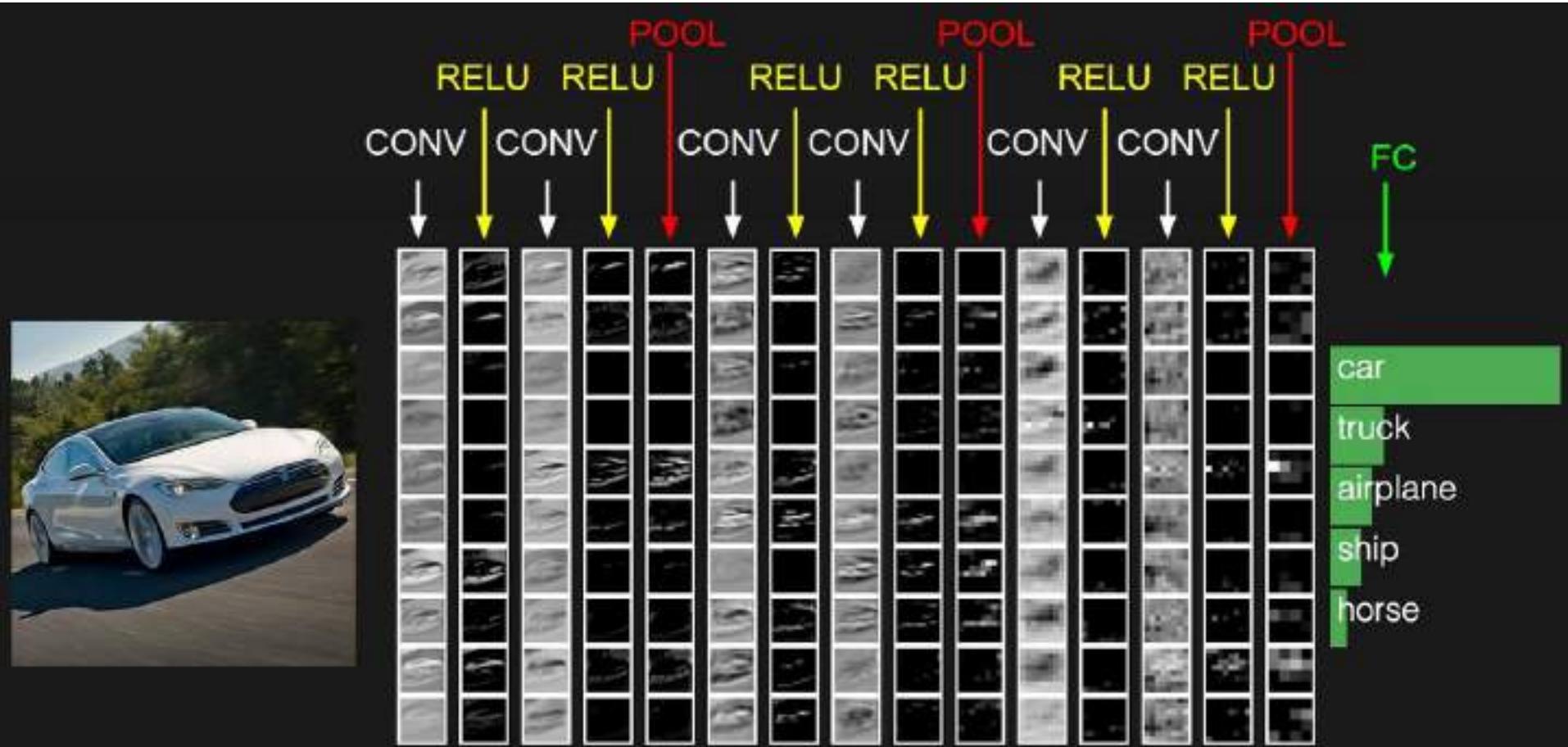
Input layer



With weight sharing

- # input units (neurons): 7
- # hidden units: 3
- **Number of parameters**
 - Without weight sharing: $3 \times 3 = 9$
 - With weight sharing : $3 \times 1 = 3$

Convolutional Neural Networks



Layers used to build ConvNets

Input Layer (Input image)

Convolutional Layer

Non-linearity Layer (such as Sigmoid, Tanh, ReLU, PReLU, ELU, Swish, etc.)

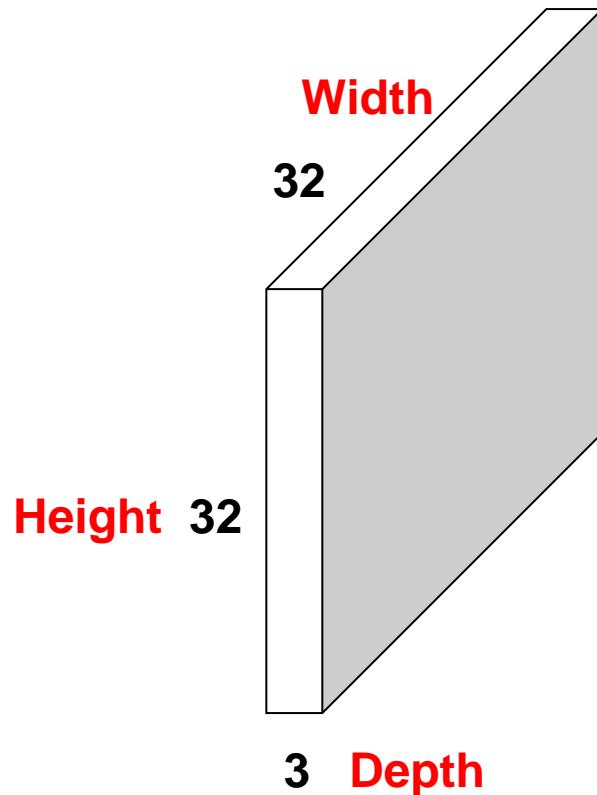
Pooling Layer (such as Max Pooling, Average Pooling, etc.)

Fully-Connected Layer

Classification Layer (Softmax, etc.)

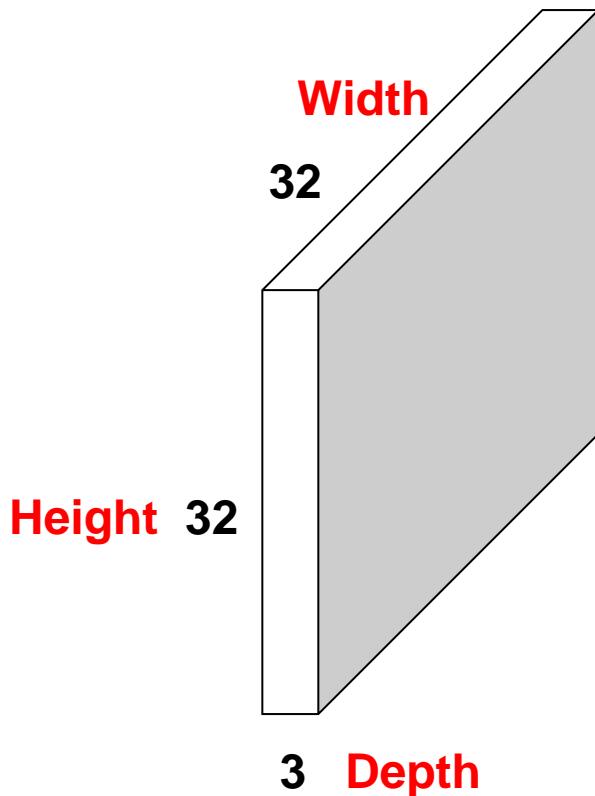
Convolutional Layer

32 × 32 × 3 Image → preserve spatial structure

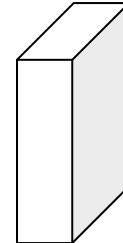


Convolutional Layer

$32 \times 32 \times 3$ Image



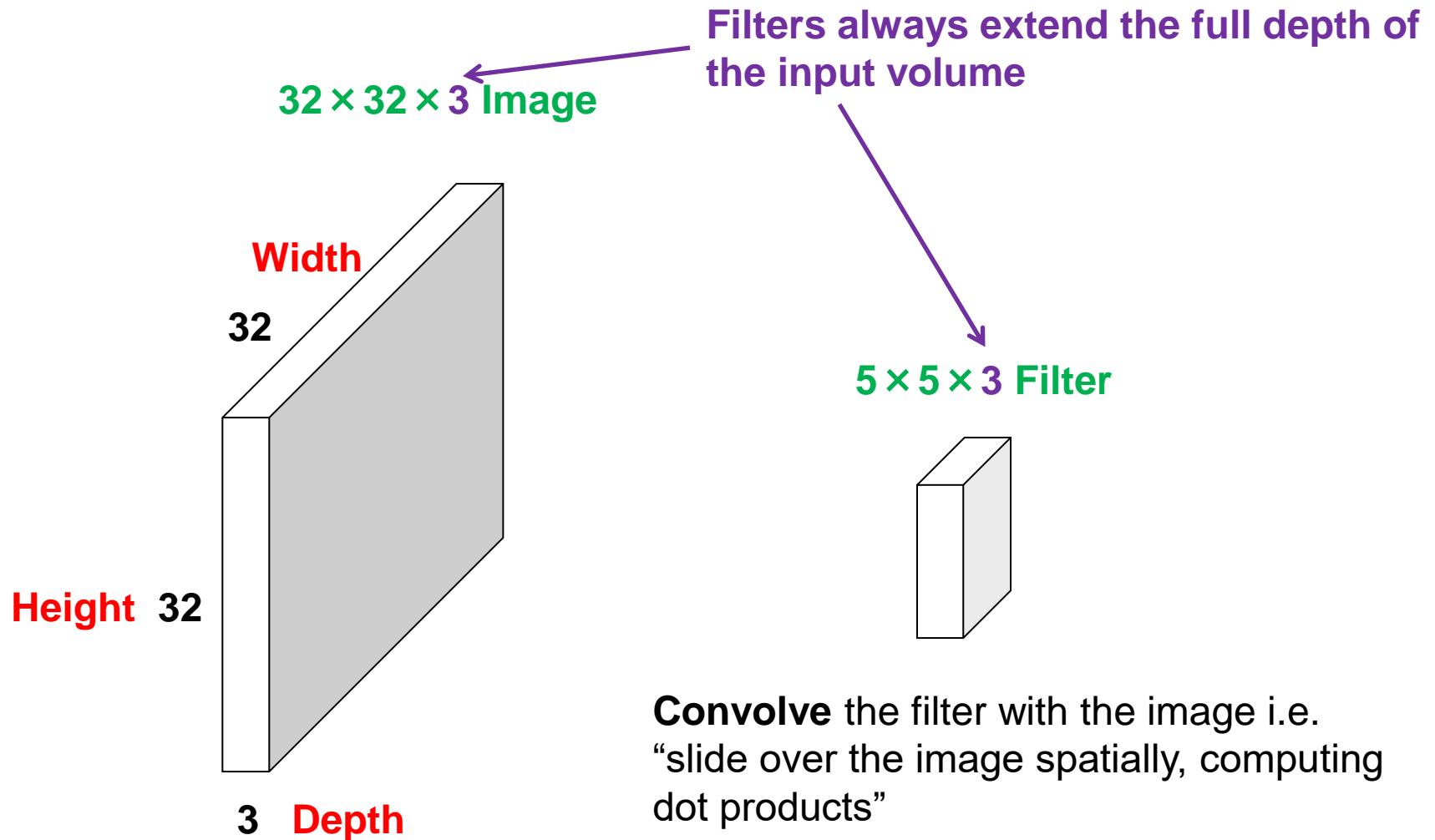
$5 \times 5 \times 3$ Filter



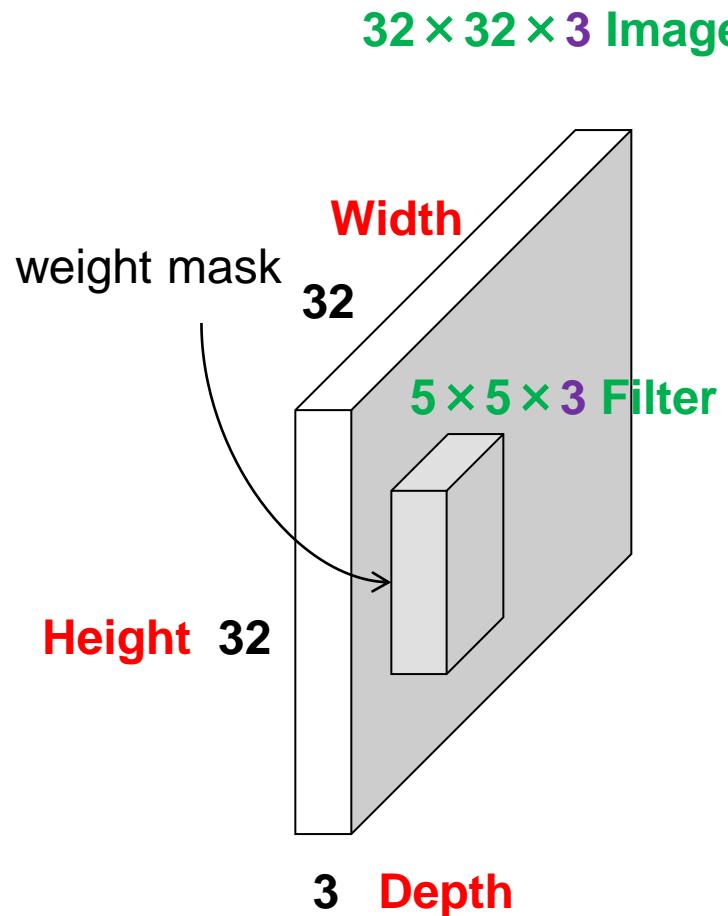
Convolve the filter with the image i.e.
“slide over the image spatially, computing
dot products”

Convolutional Layer

Handling multiple input channels

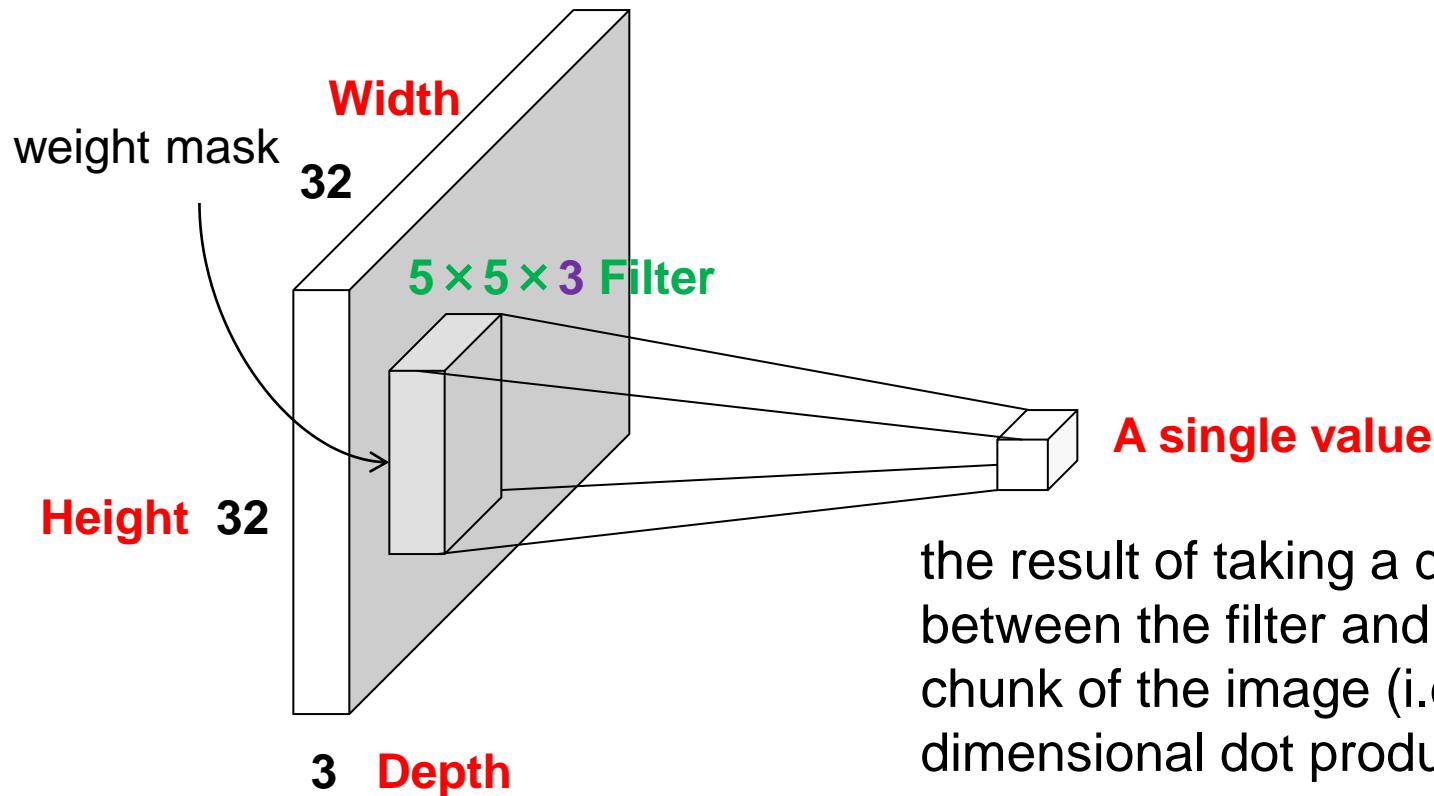


Convolutional Layer



Convolutional Layer

$32 \times 32 \times 3$ Image

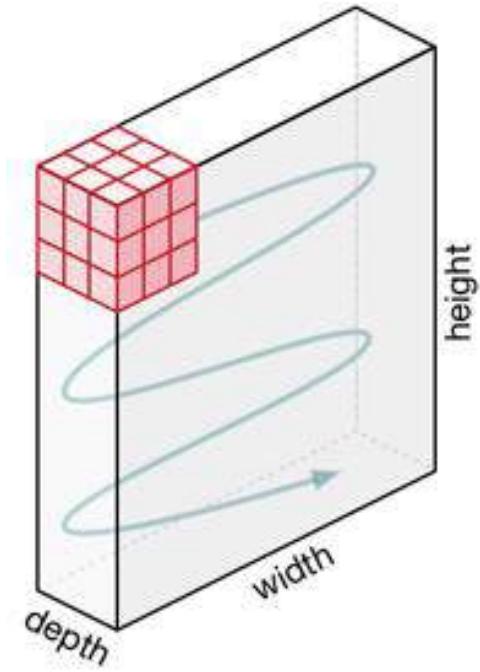
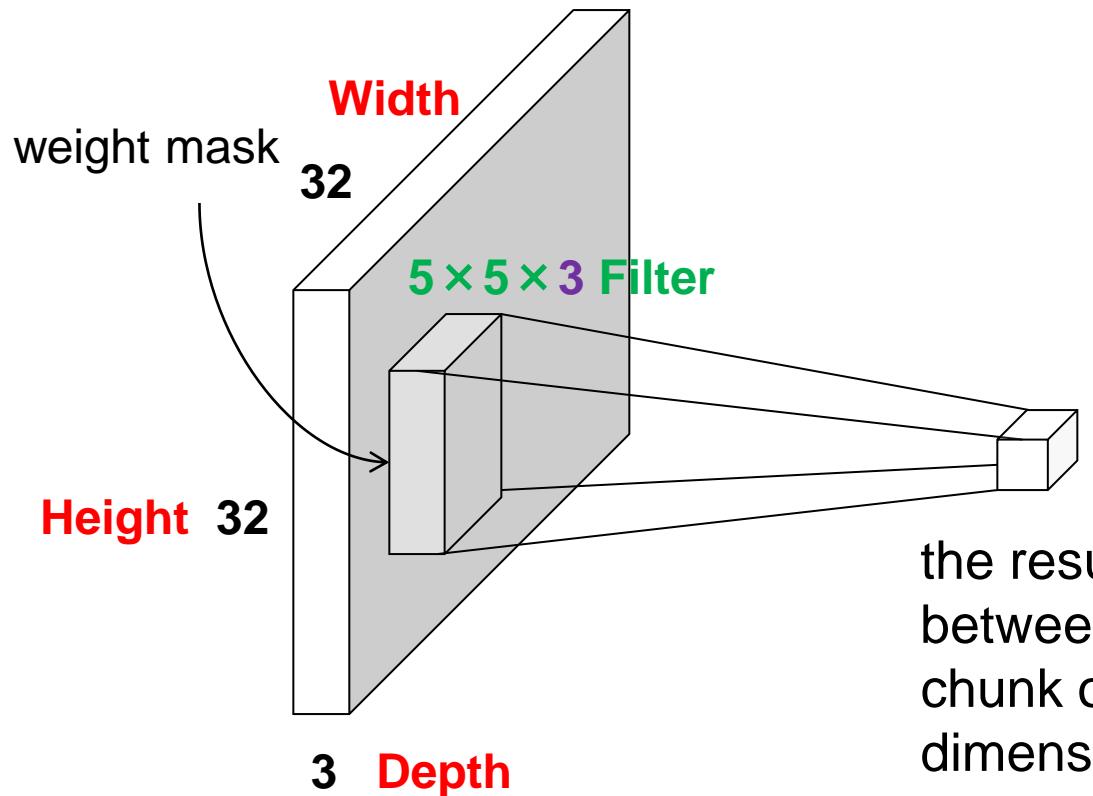


the result of taking a dot product between the filter and a small $5 \times 5 \times 3$ chunk of the image (i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

$$\mathbf{w}^T \cdot \mathbf{x} + \mathbf{b}$$

Convolutional Layer

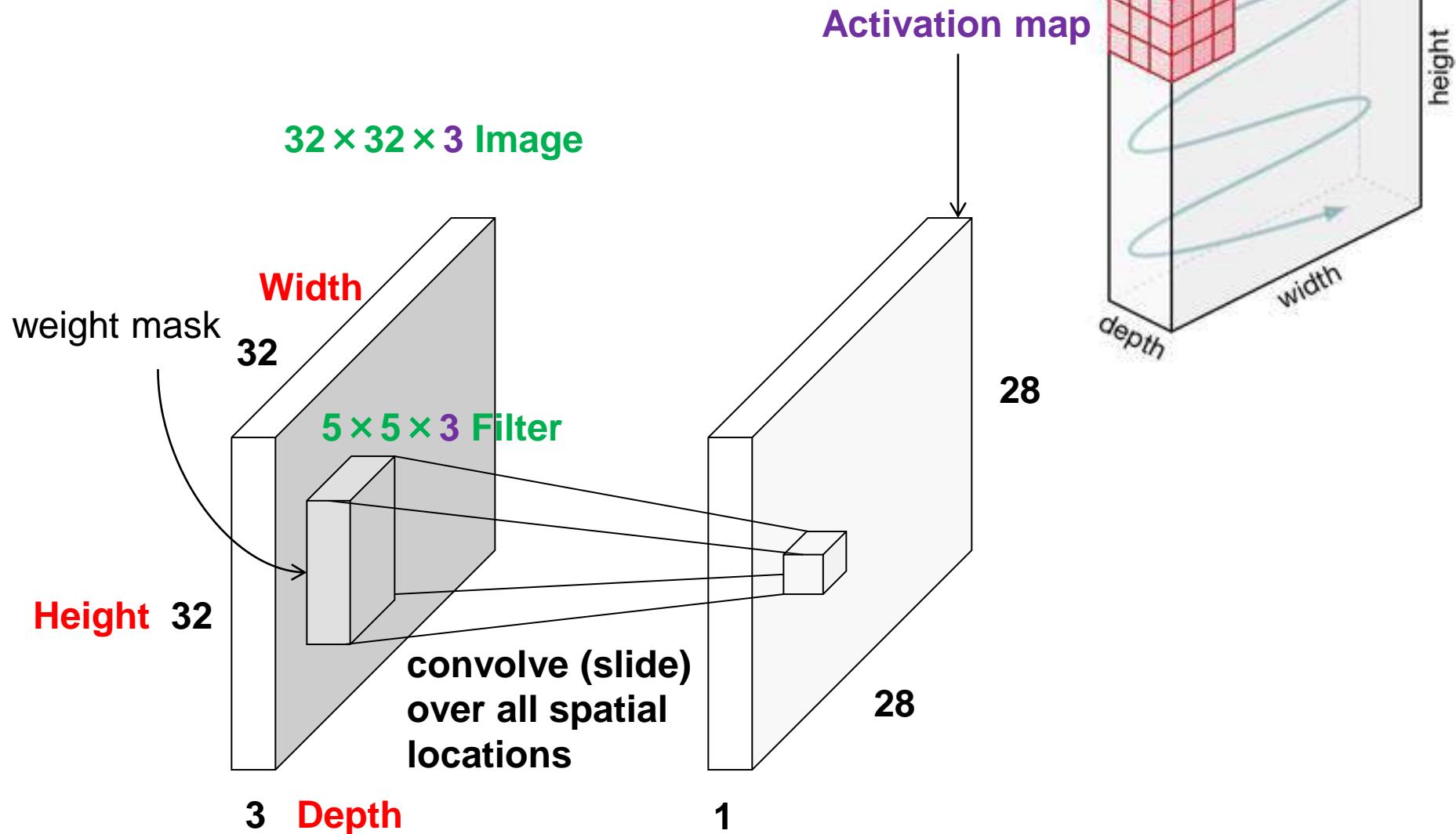
32 × 32 × 3 Image



the result of taking a dot product between the filter and a small $5 \times 5 \times 3$ chunk of the image (i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

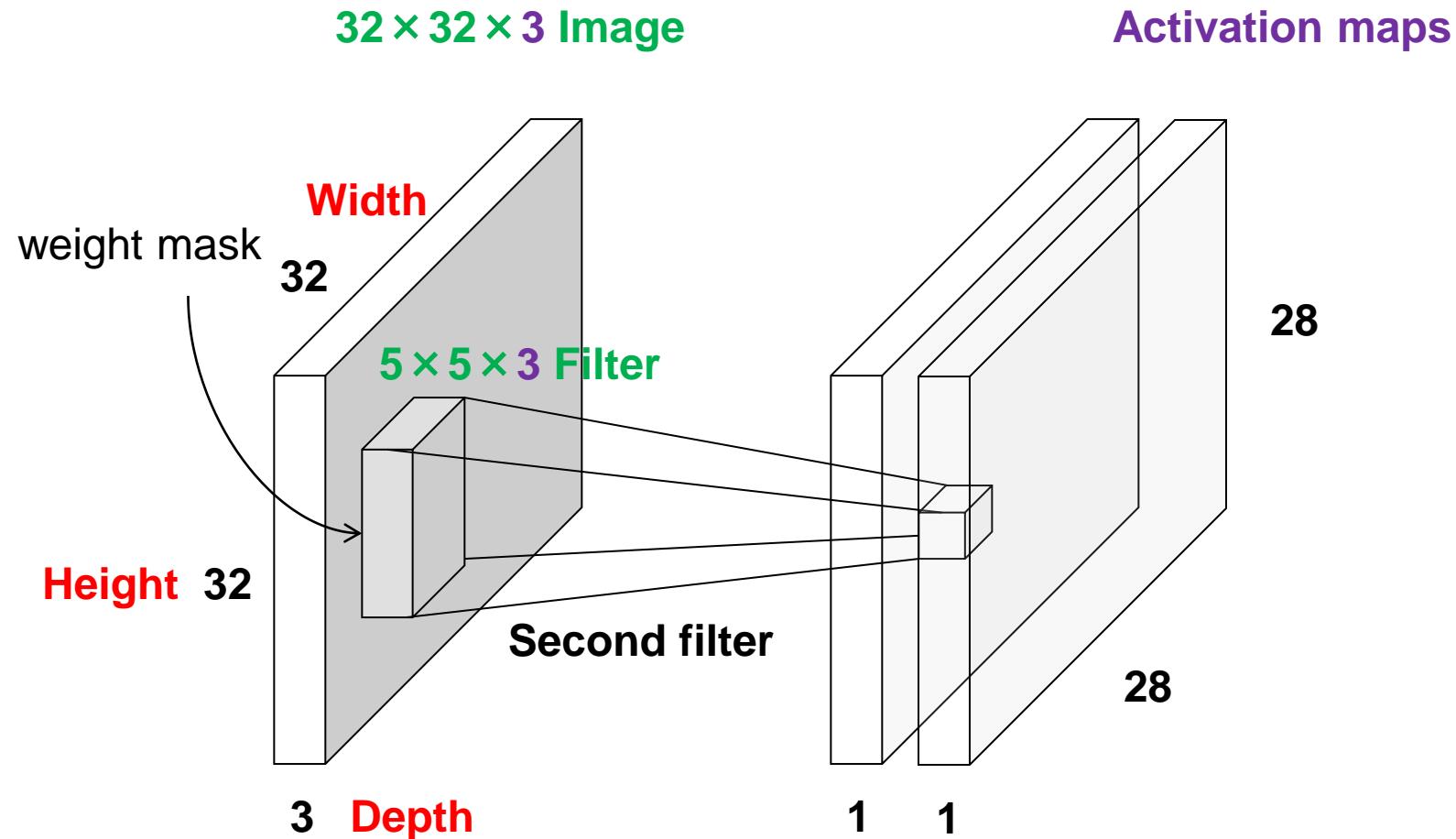
$$\mathbf{w}^T \cdot \mathbf{x} + \mathbf{b}$$

Convolutional Layer



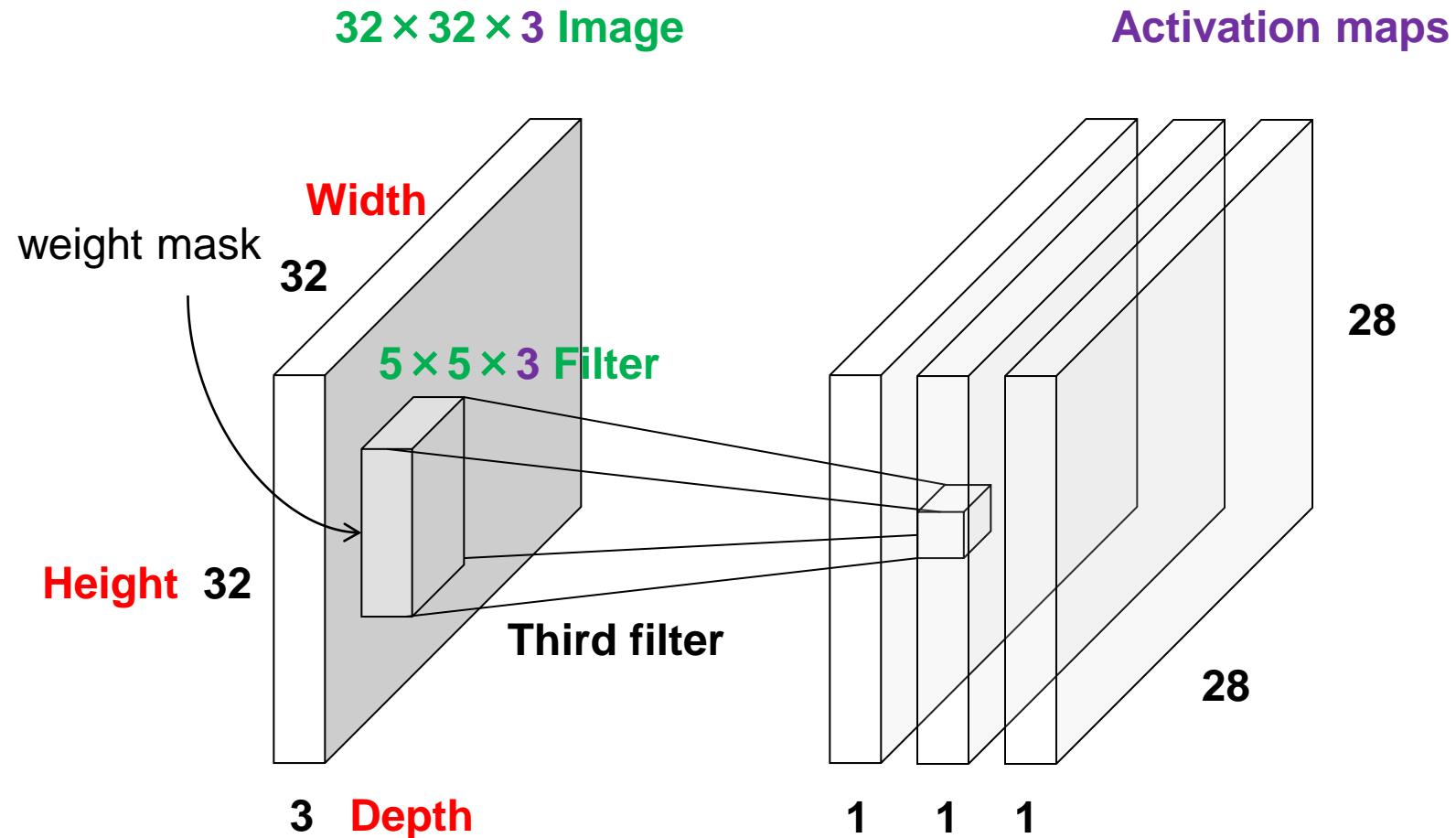
Convolutional Layer

Handling multiple output maps



Convolutional Layer

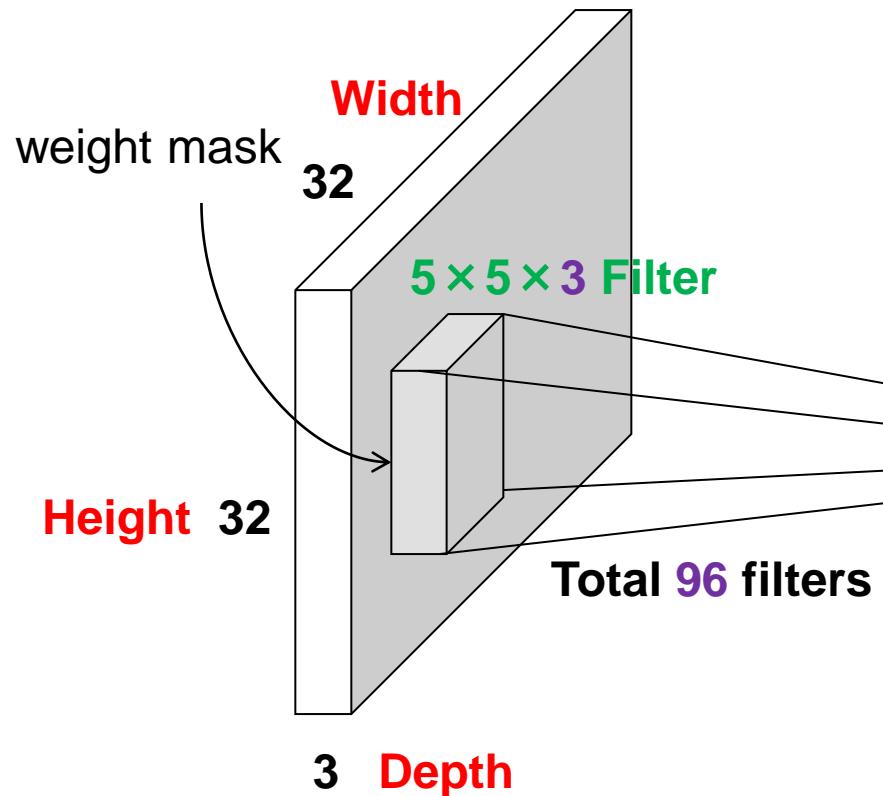
Handling multiple output maps



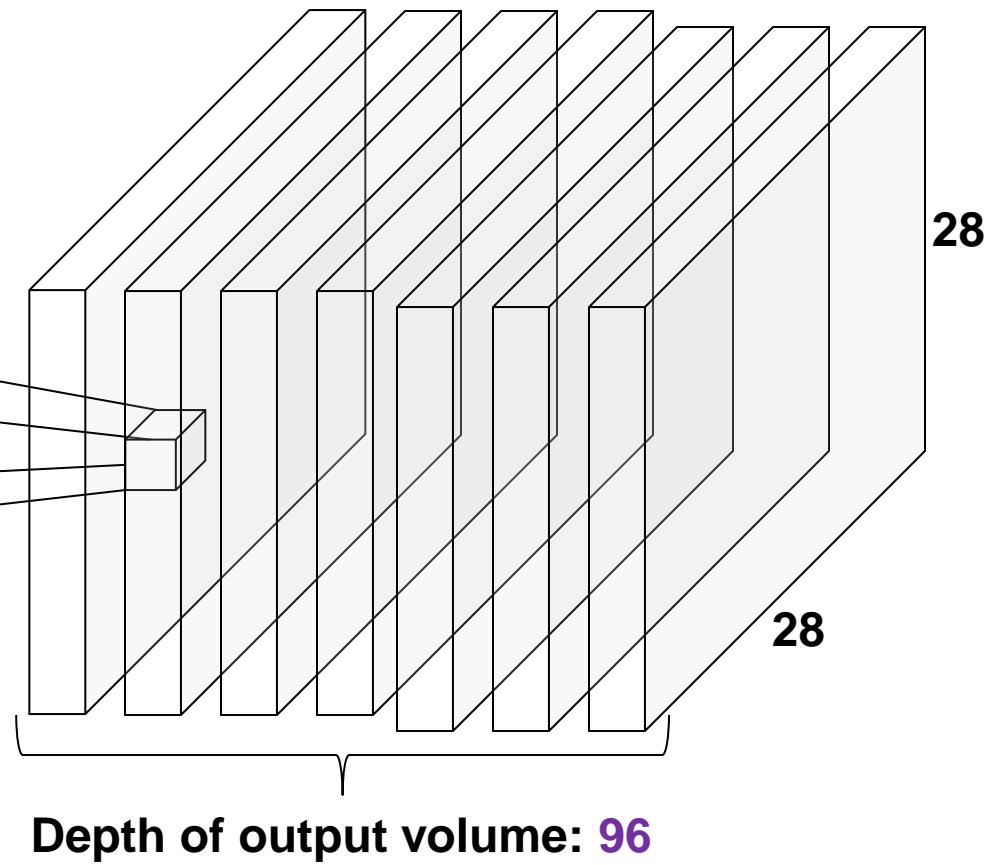
Convolutional Layer

Handling multiple output maps

$32 \times 32 \times 3$ Image



Activation maps



Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

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

Filter W0 (3x3x3)

 $w0[:, :, 0]$

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

 $w0[:, :, 1]$

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

 $w0[:, :, 2]$

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

Bias b0 (1x1x1)

 $b0[:, :, 0]$

1

 $x[:, :, 1]$

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

 $x[:, :, 2]$

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

Filter W1 (3x3x3)

 $w1[:, :, 0]$

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

 $w1[:, :, 1]$

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

 $w1[:, :, 2]$

0	1	0
0	0	-1
1	-1	0

Output Volume (3x3x2)

 $o[:, :, 0]$

-3	-1	0
-1	-8	2
3	0	3

 $o[:, :, 1]$

1	4	2
-1	4	5
3	1	3

Bias b1 (1x1x1)

 $b1[:, :, 0]$

0

toggle movement

Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

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

Filter W0 (3x3x3)

 $w0[:, :, 0]$

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

 $w0[:, :, 1]$

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

 $w0[:, :, 2]$

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

Bias b0 (1x1x1)

 $b0[:, :, 0]$

1

 $x[:, :, 1]$

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

 $x[:, :, 2]$

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

Filter W1 (3x3x3)

 $w1[:, :, 0]$

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

 $w1[:, :, 1]$

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

 $w1[:, :, 2]$

0	1	0
0	0	-1
1	-1	0

Bias b1 (1x1x1)

 $b1[:, :, 0]$

0

Output Volume (3x3x2)

 $o[:, :, 0]$

-3	-1	0
-1	-8	2
3	0	3

 $o[:, :, 1]$

1	4	2
-1	4	5
3	1	3

toggle movement

Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

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

Filter W0 (3x3x3)

 $w0[:, :, 0]$

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

 $w0[:, :, 1]$

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

 $w0[:, :, 2]$

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

 $b0[:, :, 0]$

1

 $x[:, :, 1]$

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

 $x[:, :, 2]$

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

Filter W1 (3x3x3)

 $w1[:, :, 0]$

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

 $w1[:, :, 1]$

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

 $w1[:, :, 2]$

0	1	0
0	0	-1
1	-1	0

Bias b1 (1x1x1)

 $b1[:, :, 0]$

0

Output Volume (3x3x2)

 $o[:, :, 0]$

-3	-1	0
-1	-8	2
3	0	3

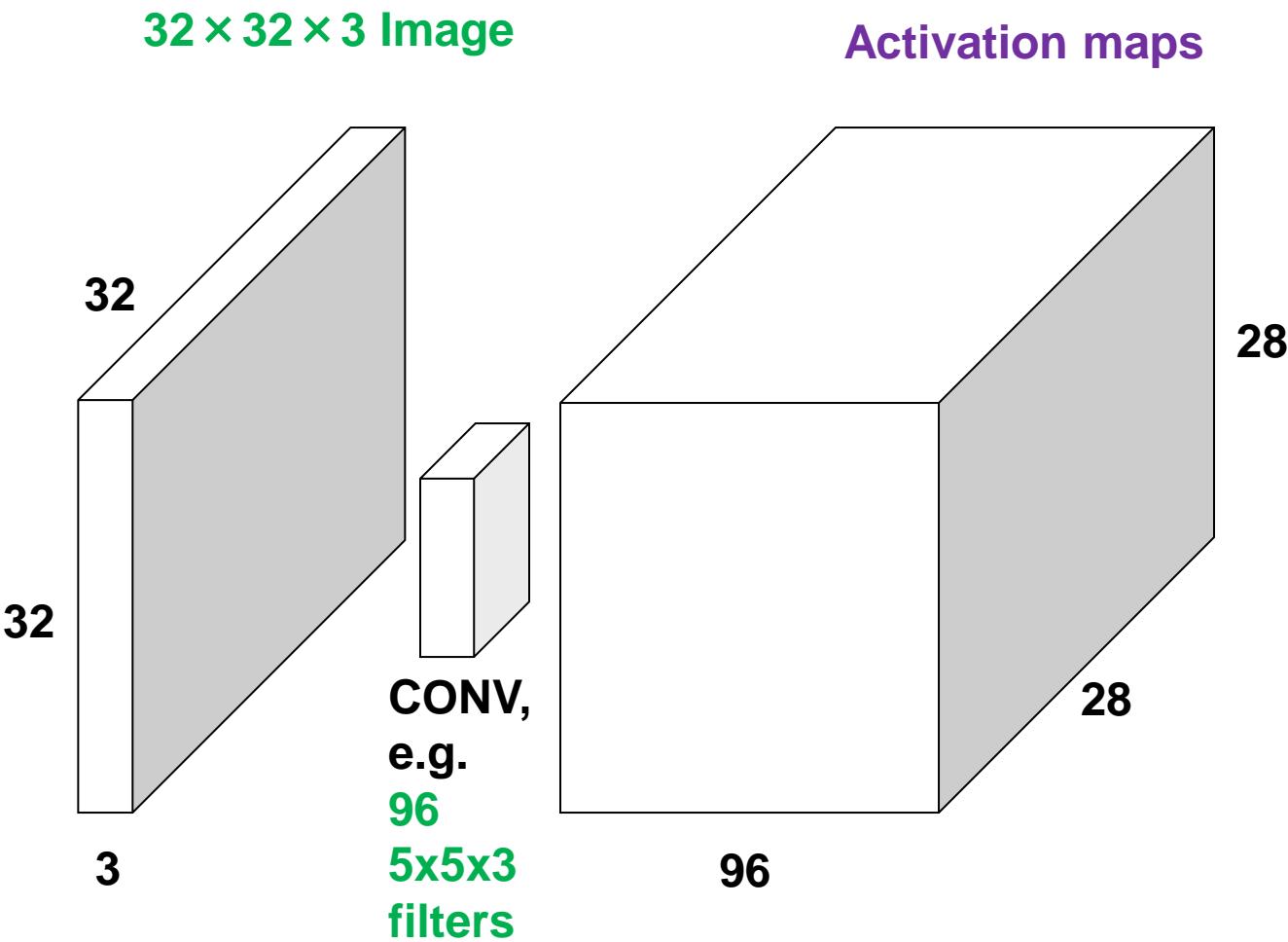
 $o[:, :, 1]$

1	4	2
-1	4	5
3	1	3

toggle movement

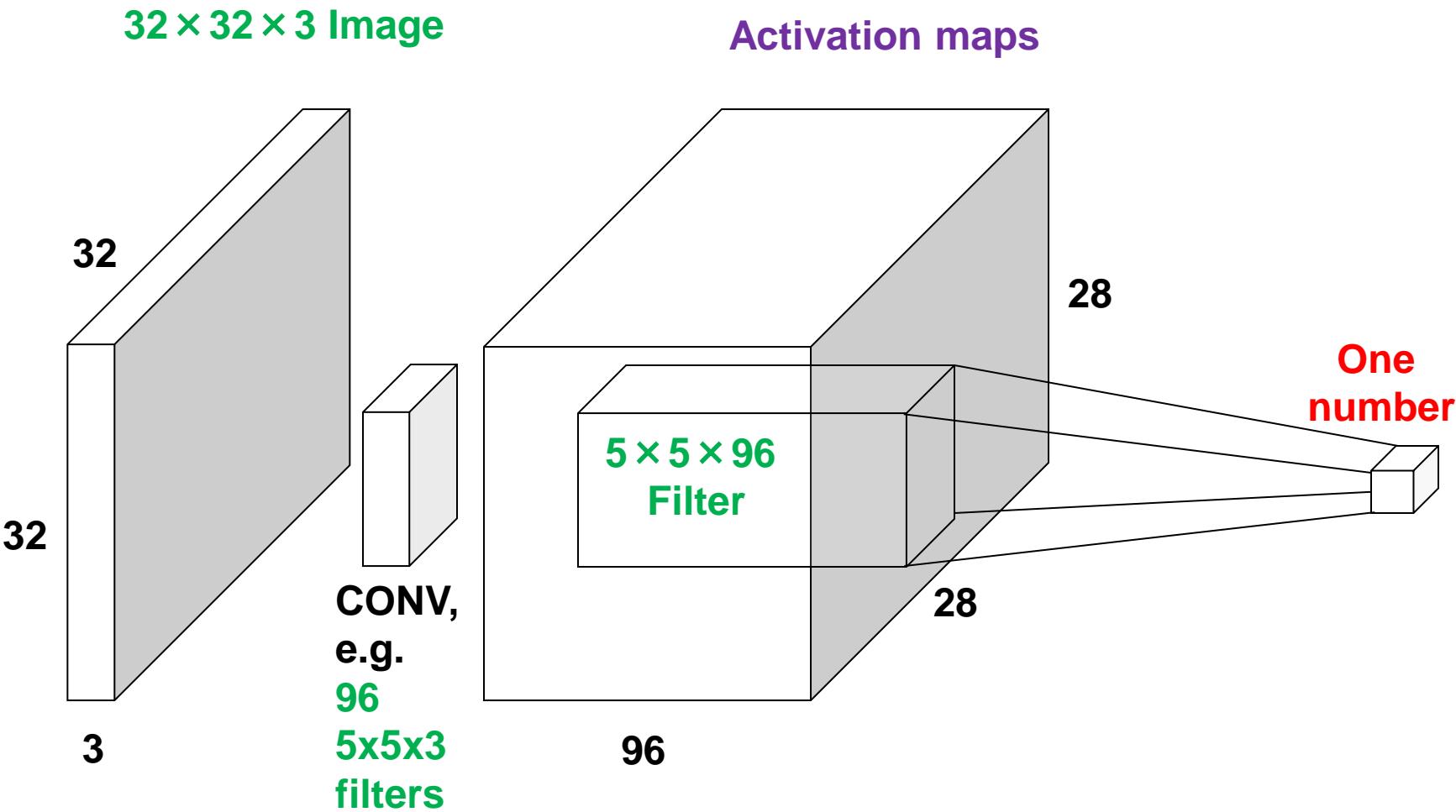
Convolutional Layer

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



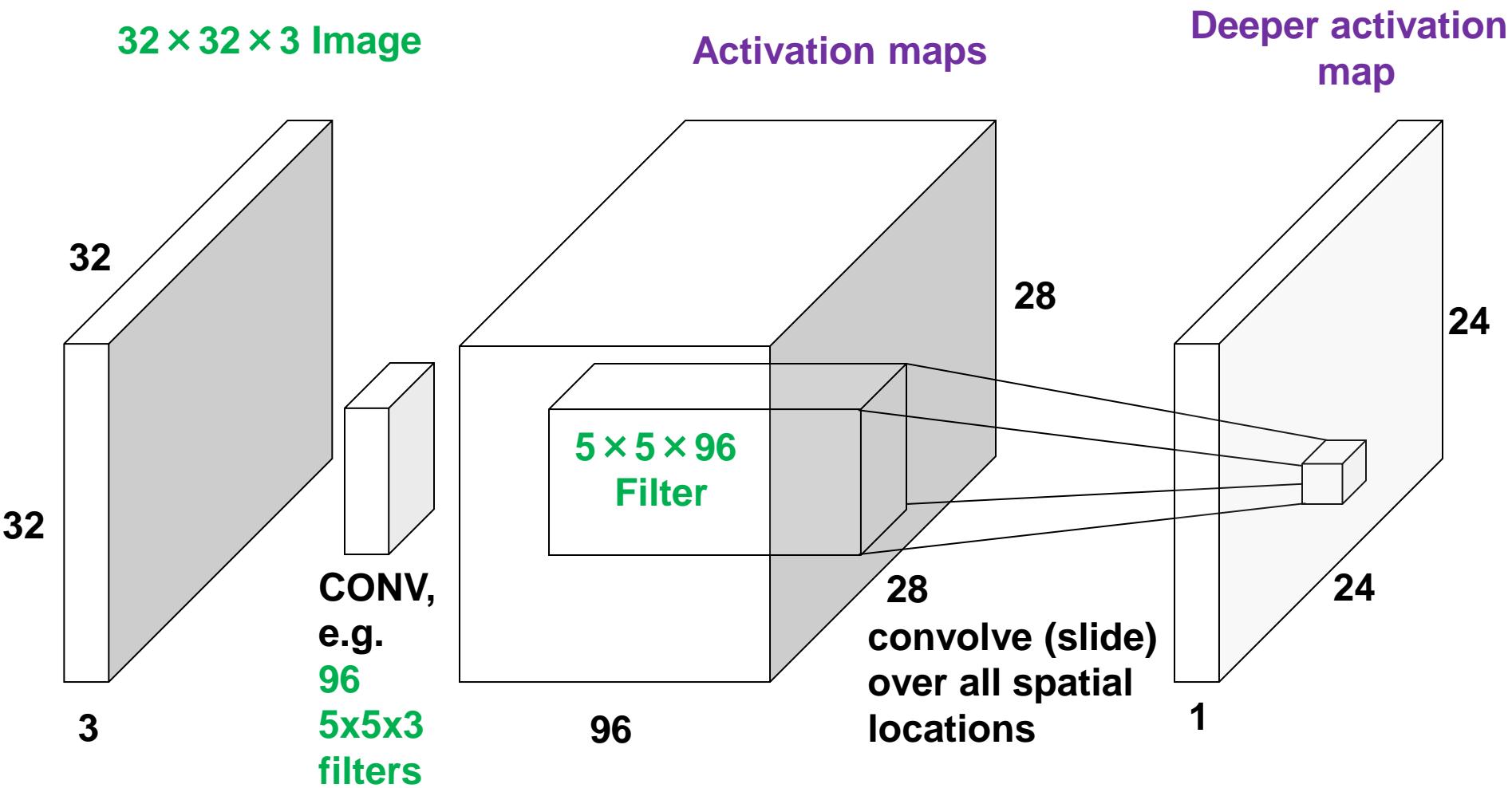
Convolutional Layer

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



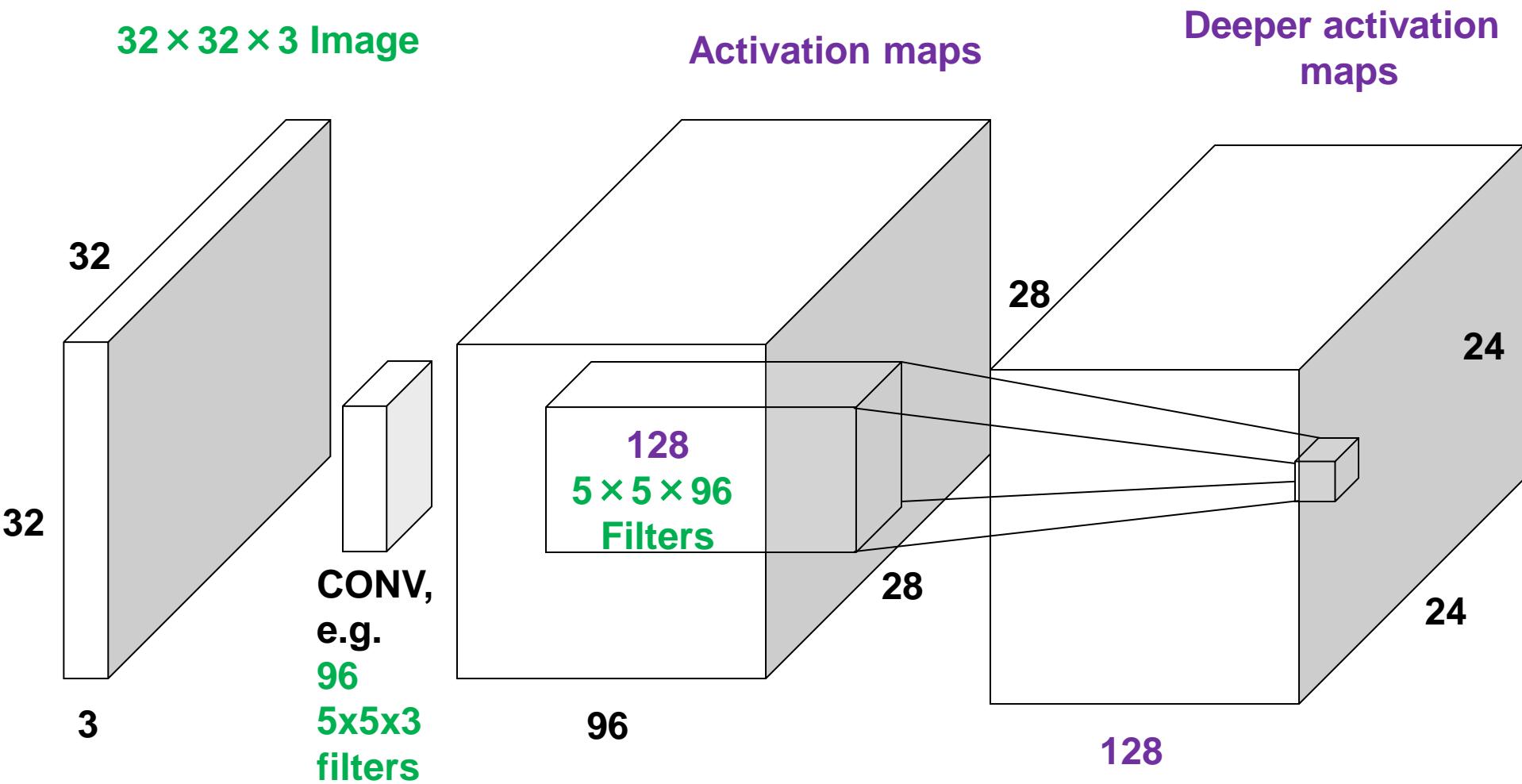
Convolutional Layer

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



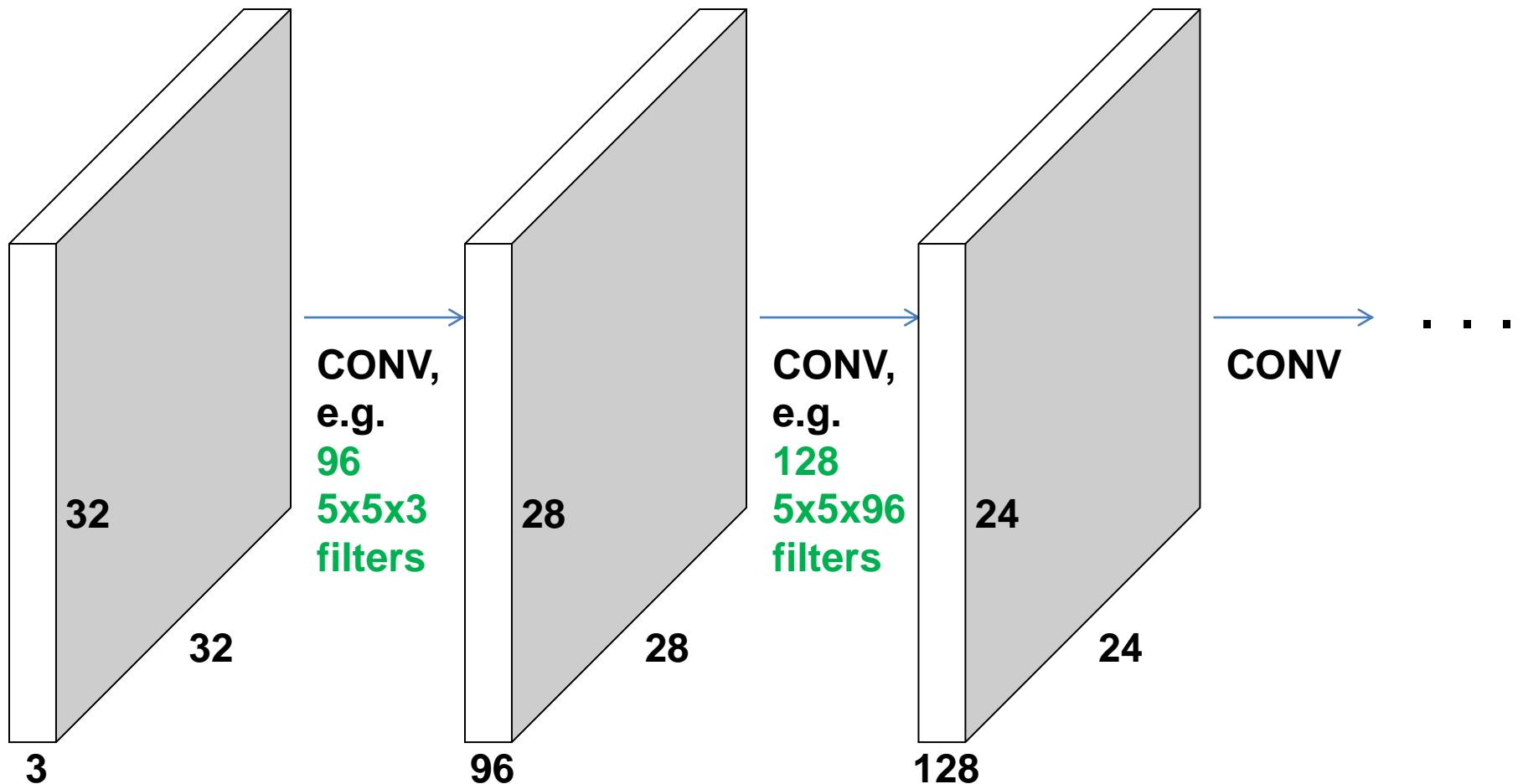
Convolutional Layer

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Multilayer Convolution

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Any Convolution Layer

- Local connectivity
- Weight sharing
- Handling multiple input channels
- Handling multiple output maps

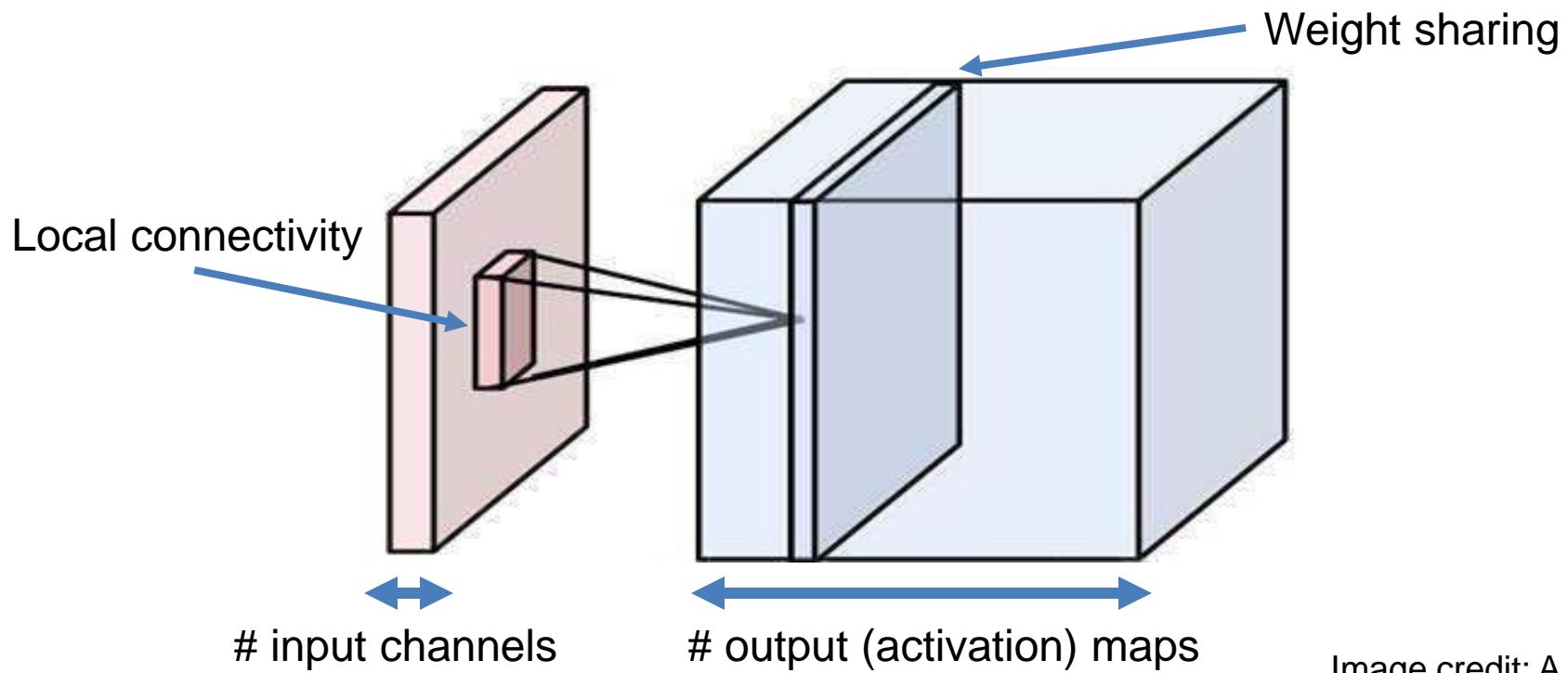
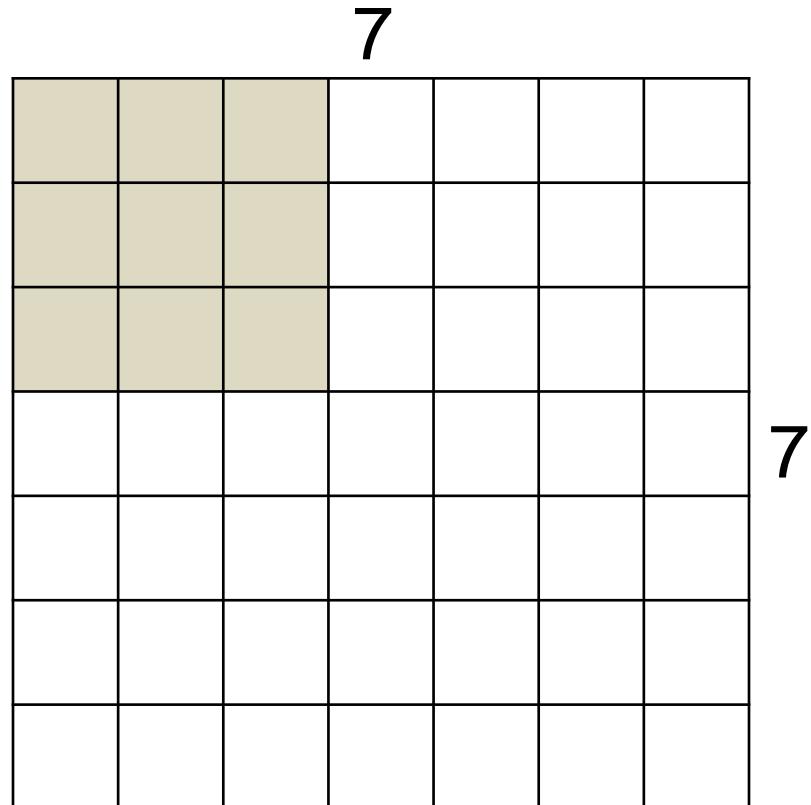


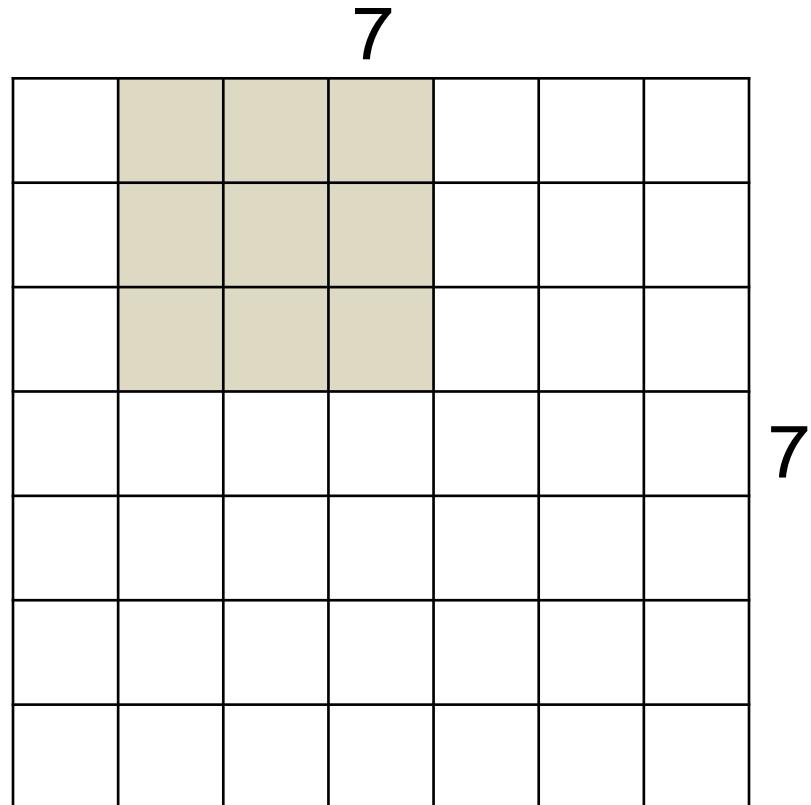
Image credit: A. Karpathy

A closer look at spatial dimensions



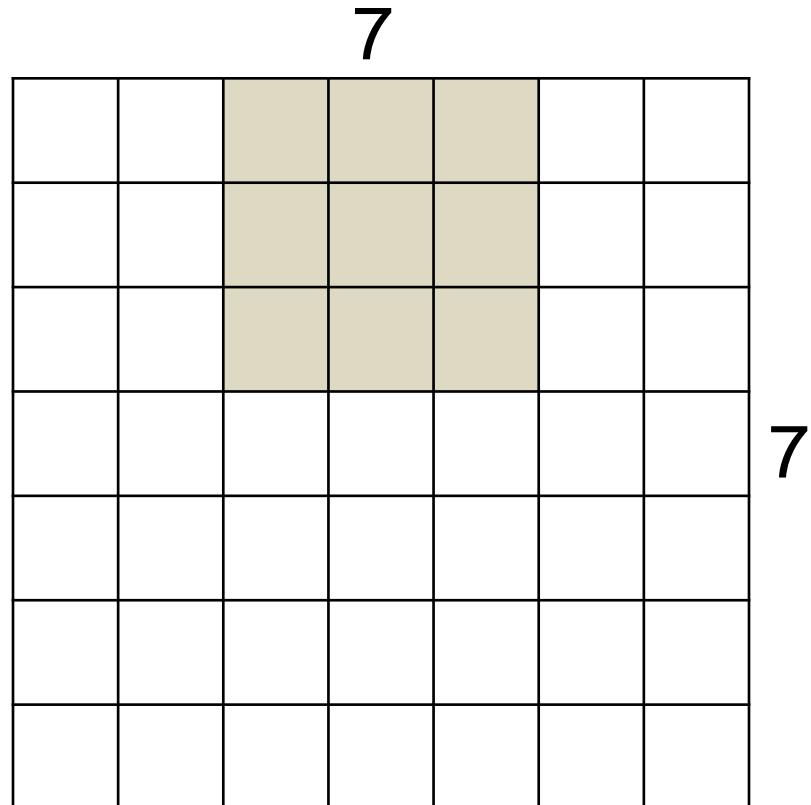
7×7 input (spatially)
assume 3×3 filter

A closer look at spatial dimensions



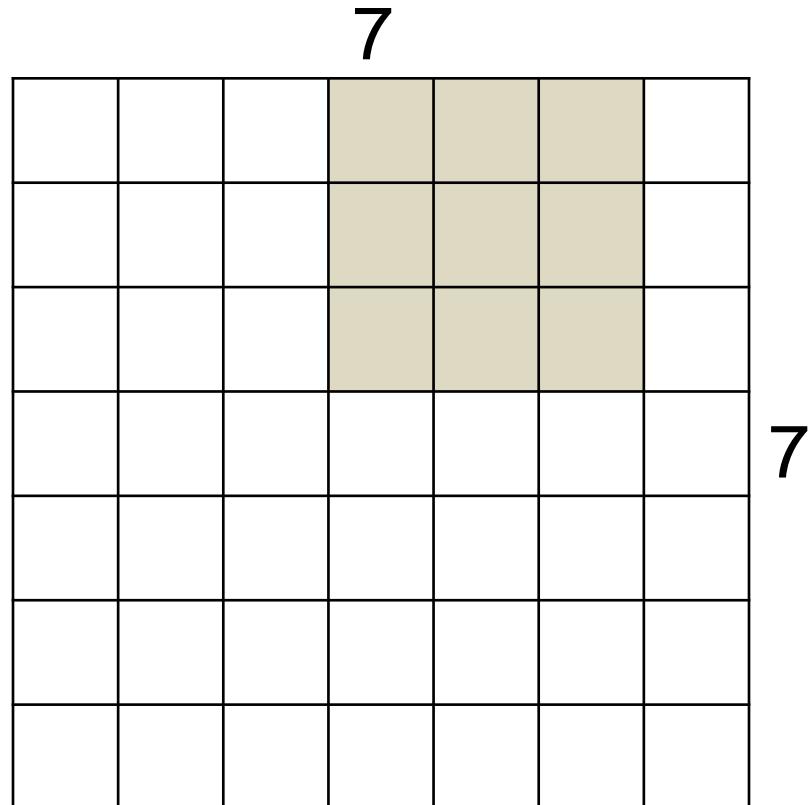
7×7 input (spatially)
assume 3×3 filter

A closer look at spatial dimensions



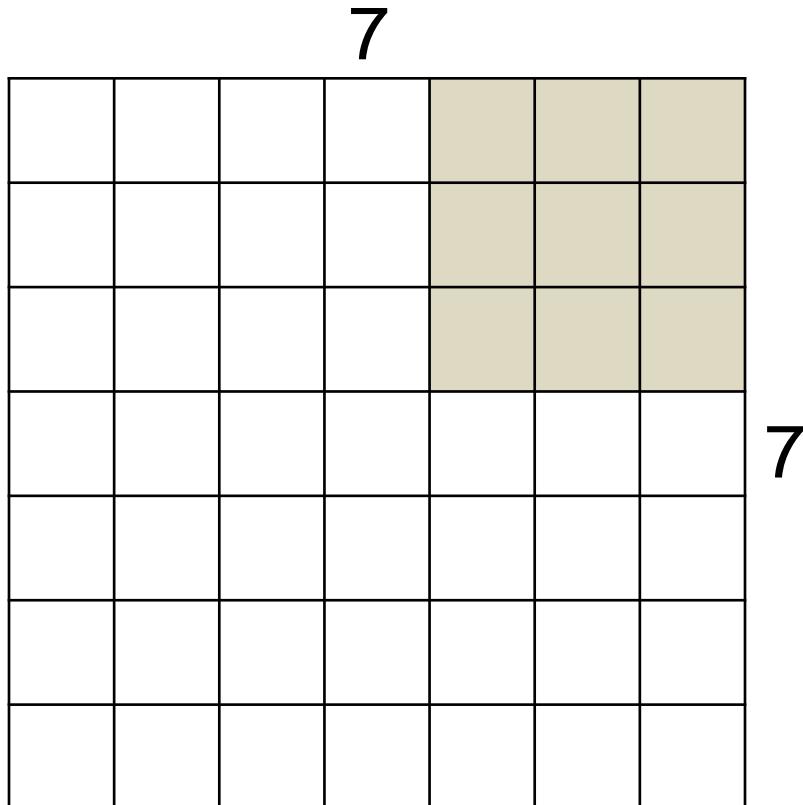
7×7 input (spatially)
assume 3×3 filter

A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter

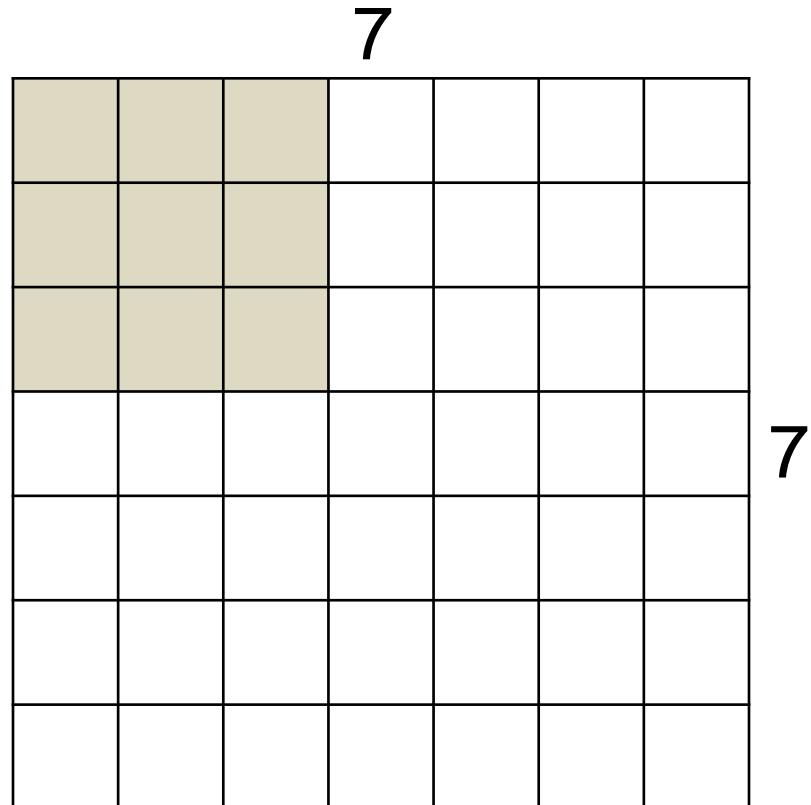
A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter

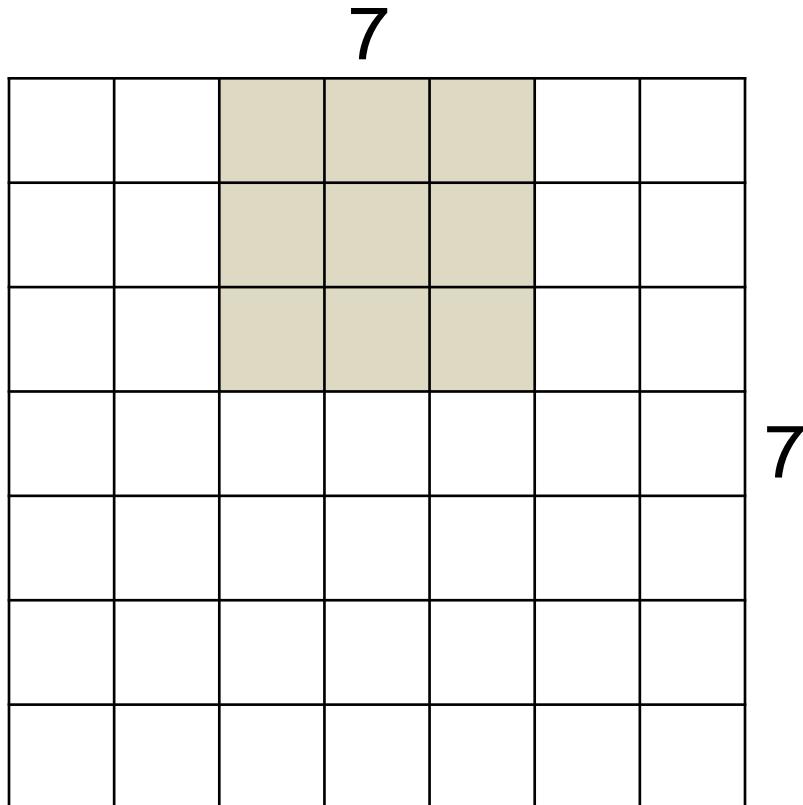
→ **5×5 output**

A closer look at spatial dimensions



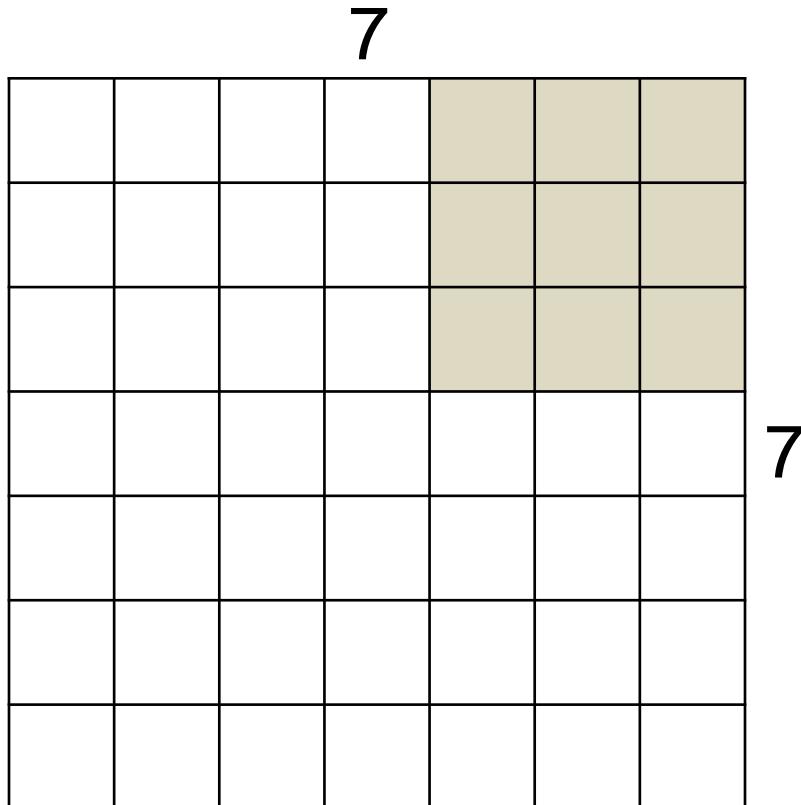
7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

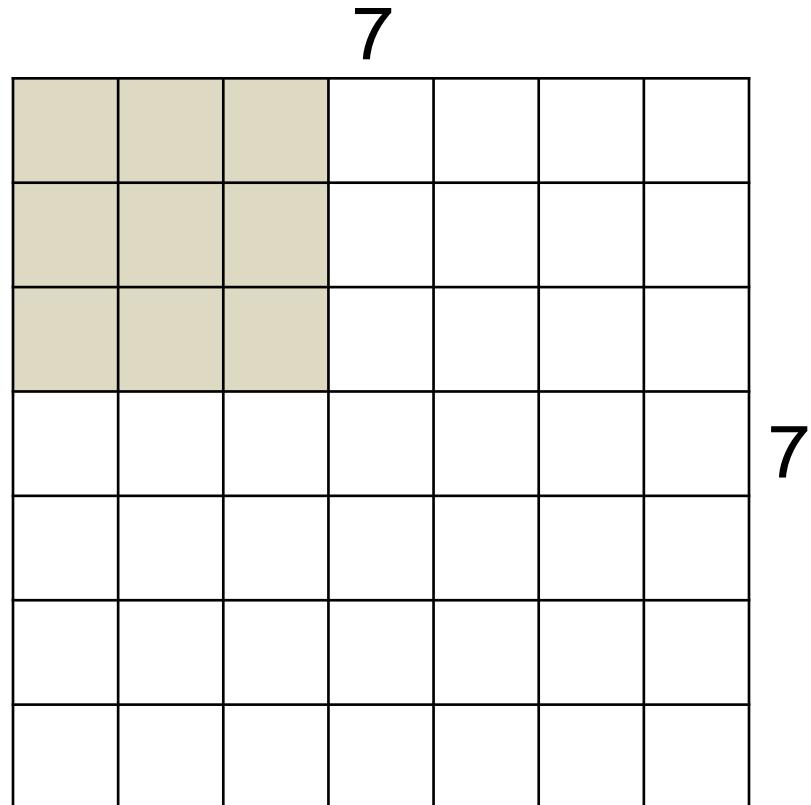
A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

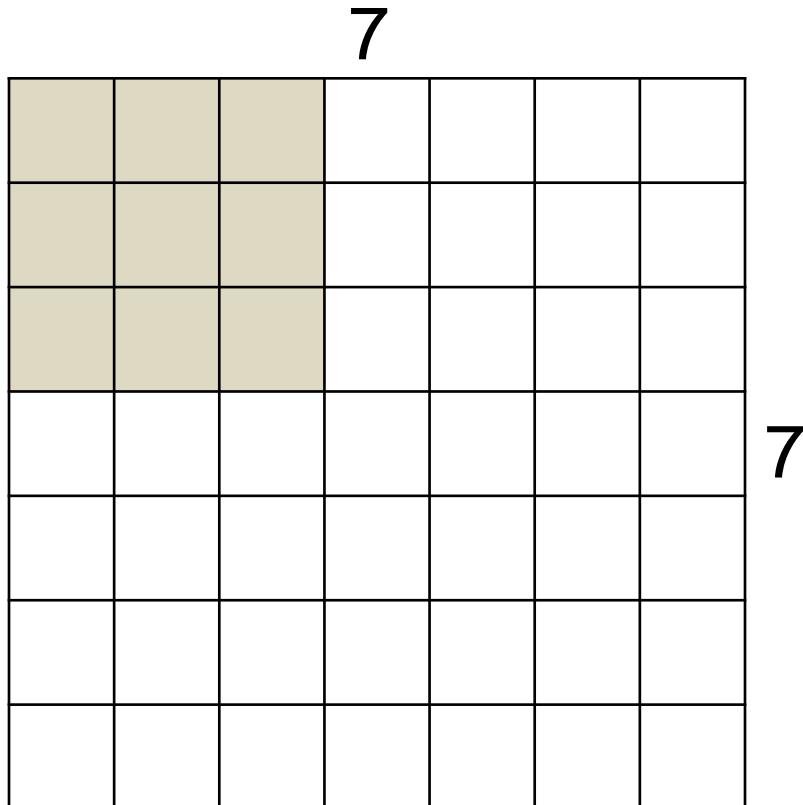
→ **3×3 output**

A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter
applied with **stride 3**

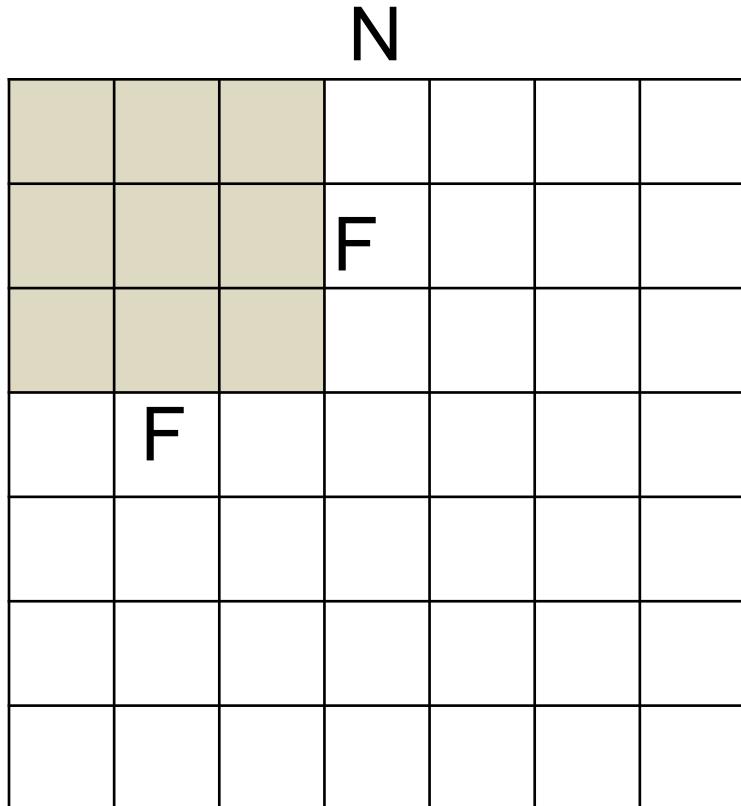
A closer look at spatial dimensions



7×7 input (spatially)
assume 3×3 filter
applied with **stride 3**

doesn't fit!
cannot apply 3×3 filter on
 7×7 input with stride 3.

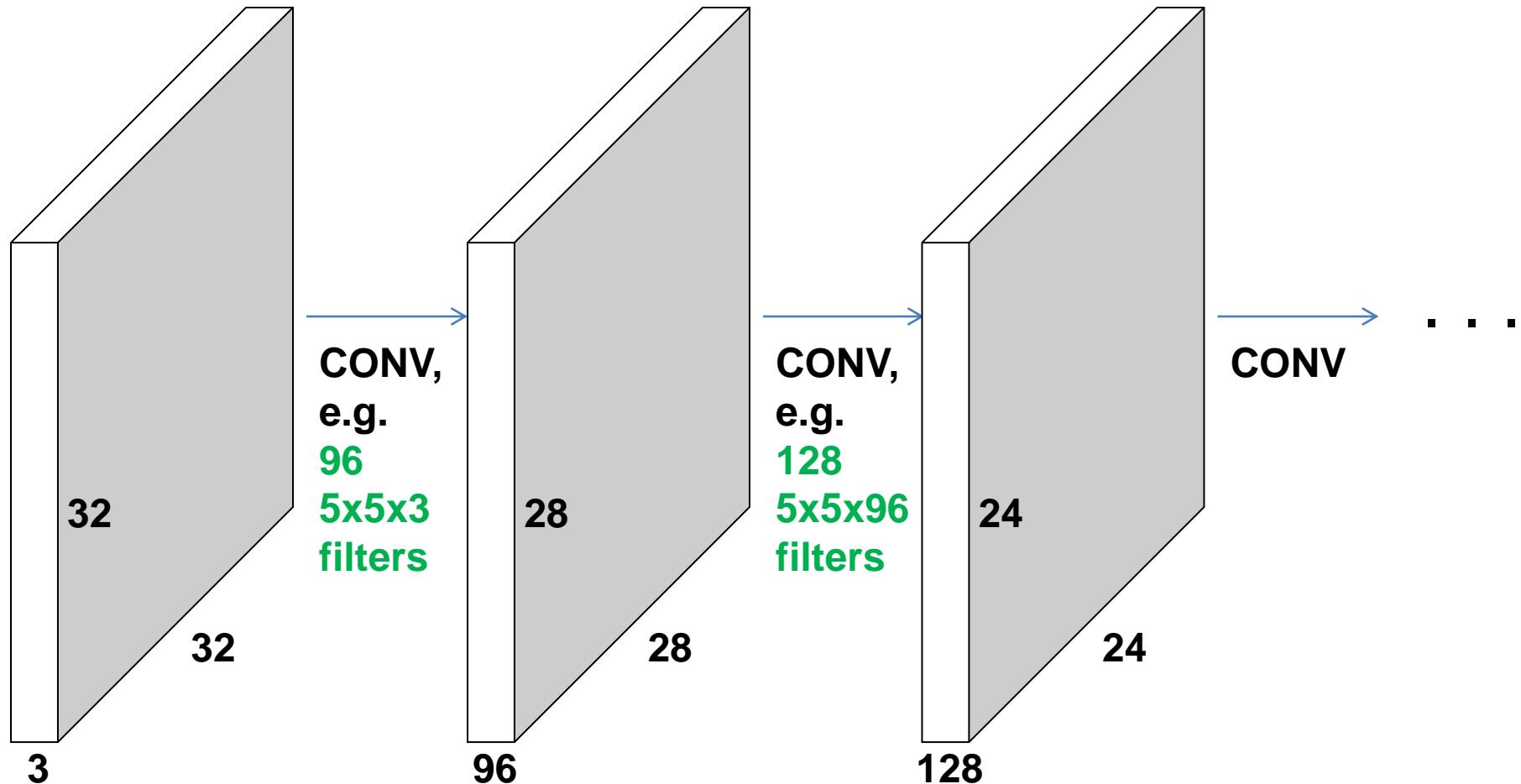
A closer look at spatial dimensions



Output size
 $(N - F) / \text{stride} + 1$

N e.g. $N = 7$, $F = 3$
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$

A closer look at spatial dimensions



E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

In practice: common to zero pad

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

e.g. **input 7×7 (spatially)**
 3×3 filter, applied with **stride 1**
pad with 1 pixel border

What is the output dimension?

In practice: common to zero pad

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

e.g. **input 7×7 (spatially)**
 3×3 filter, applied with **stride 1**
pad with 1 pixel border

7 \times 7 Output

In practice: common to zero pad

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

e.g. **input 7×7 (spatially)
 3×3 filter**, applied with **stride 1
pad with 1 pixel border**

7 × 7 Output

in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)
e.g.

$F = 3 \Rightarrow$ zero pad with 1

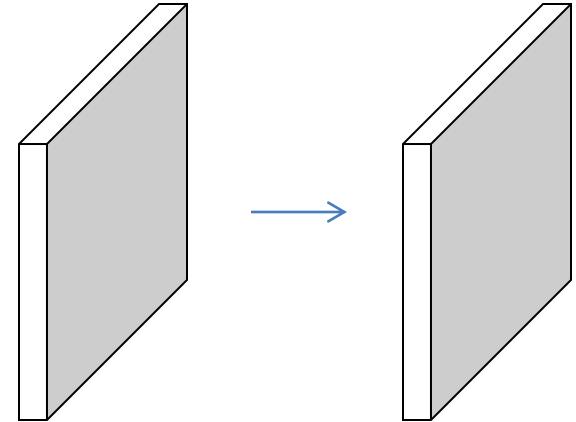
$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Example

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

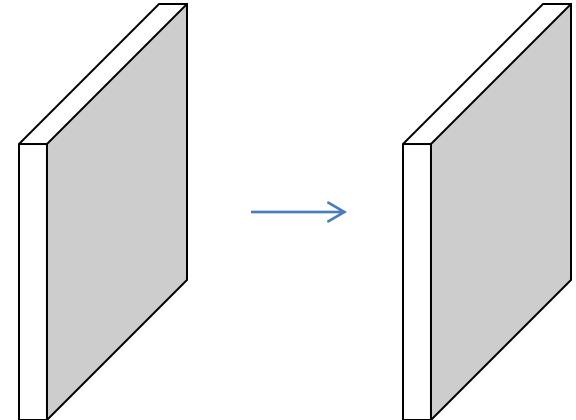


Output volume size: ?

Example

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Output volume size:

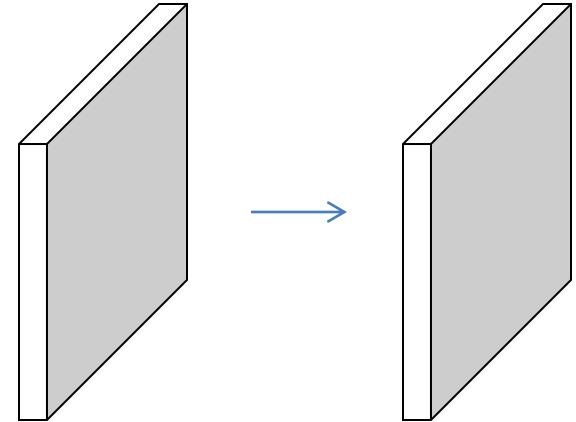
$$(32+2*2-5)/1+1 = 32 \text{ spatially, so}$$

32x32x10

Example

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

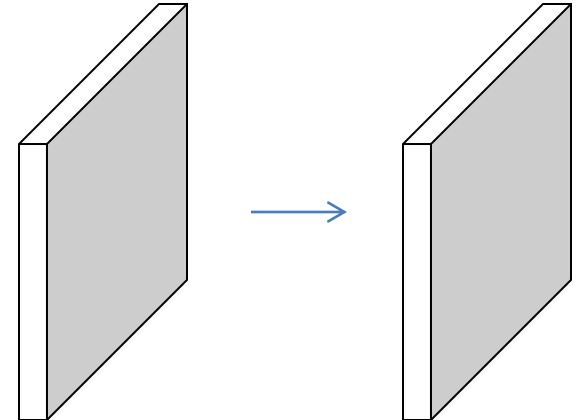


Number of parameters in this layer?

Example

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

each filter has

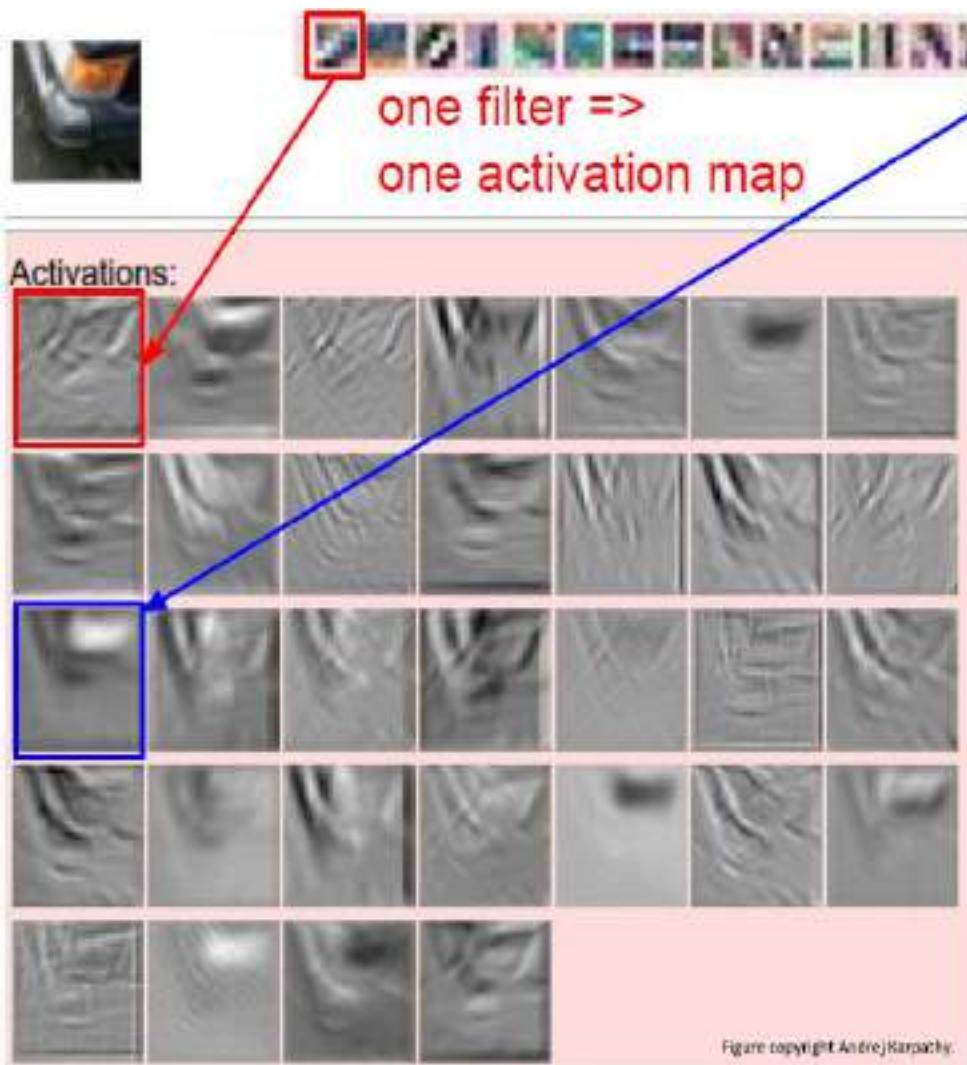
$5*5*3 + 1 = 76$ params (+1 for bias)

$\Rightarrow 76*10 = 760$

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Convolution as feature extraction



example 5x5 filters
(32 total)

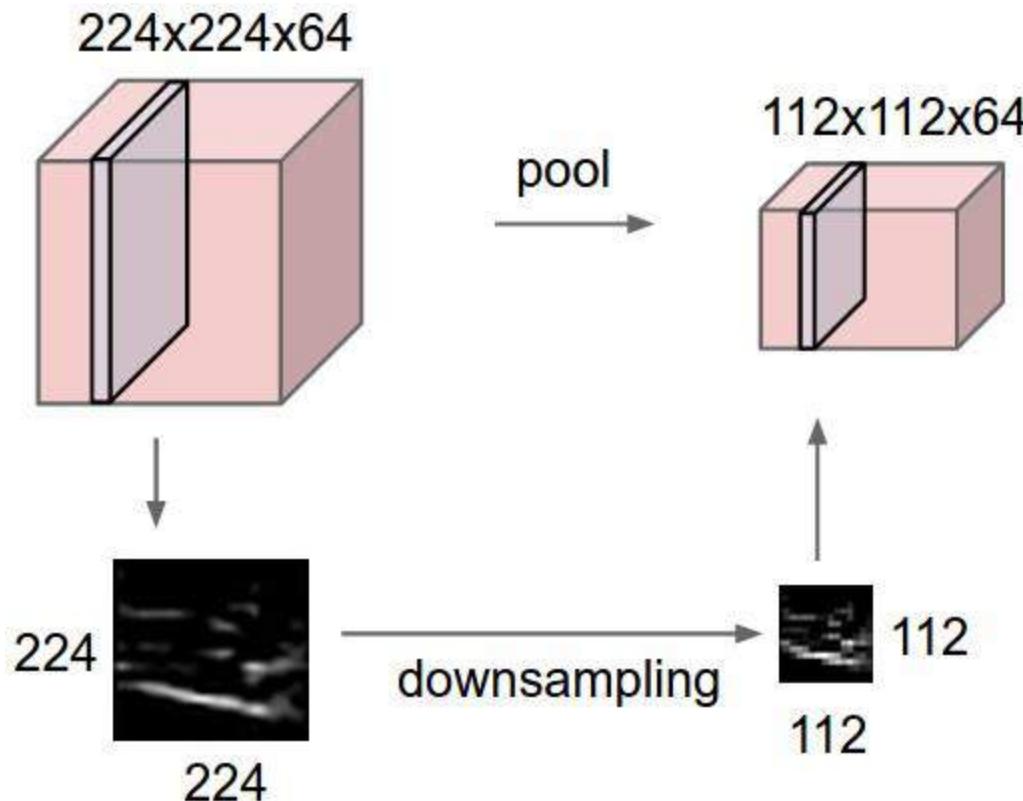
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

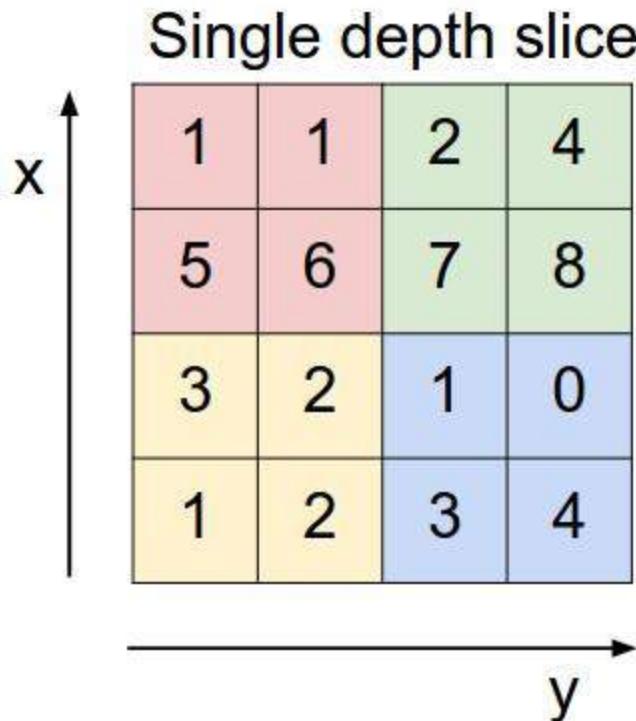
↑
elementwise multiplication and sum of a filter and the signal (image)

Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



max pool with 2x2 filters
and stride 2

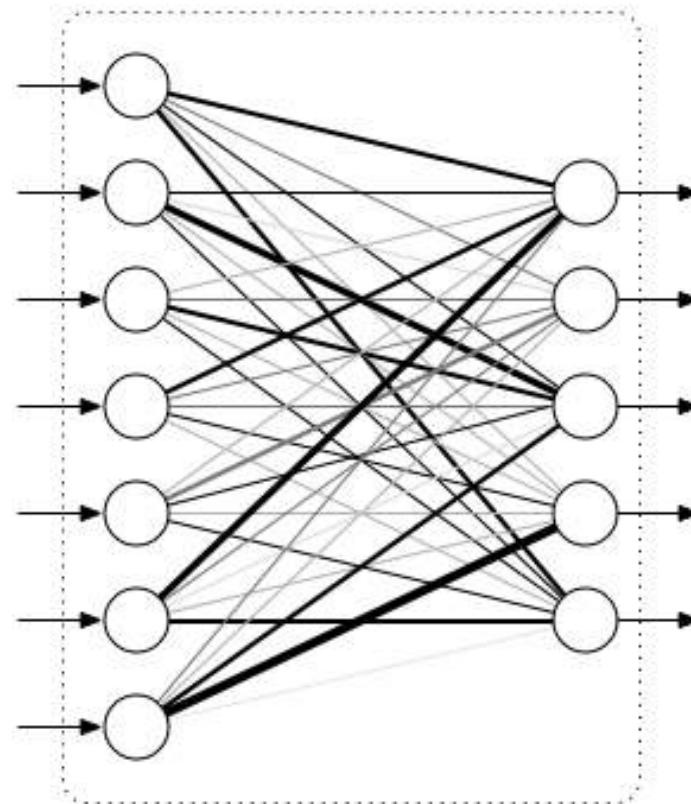
6	8
3	4

Pooling Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Fully Connected Layer

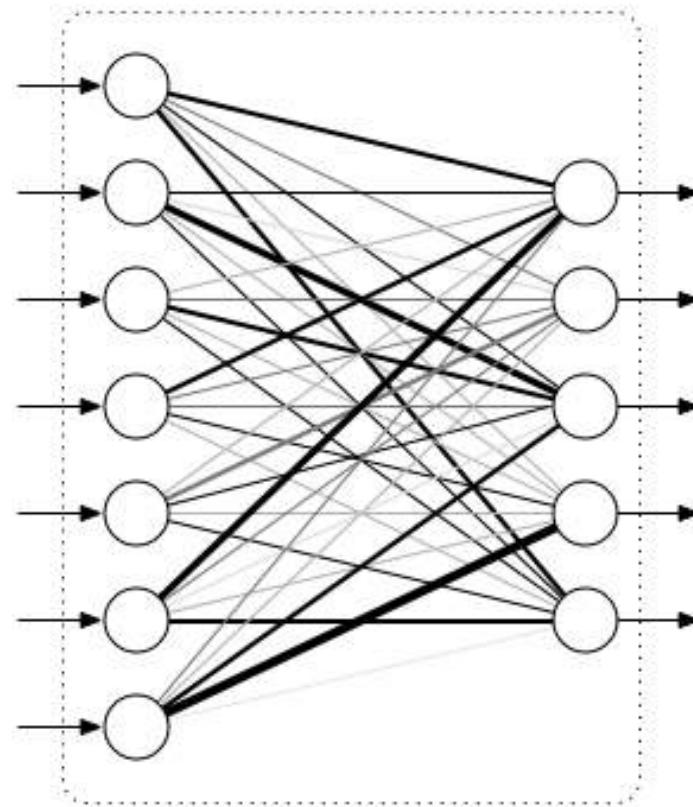
- Connect every neuron in one layer to every neuron in another layer
- Same as the traditional multi-layer perceptron neural network



Fully Connected Layer

- Connect every neuron in one layer to every neuron in another layer
- Same as the traditional multi-layer perceptron neural network

**No. of Neurons (Last FC)
= No. of classes**



Loss/Classification Layer

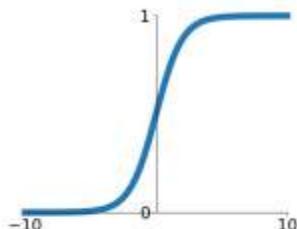
- SVM Classifier (SVM Loss/Hinge Loss/Max-margin Loss)
- Softmax Classifier (Softmax Loss/Cross-entropy Loss)

Non-linearity Layer

Activation Functions

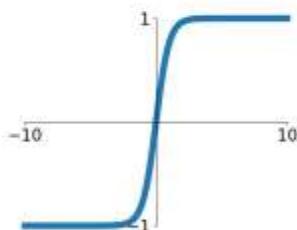
Sigmoid

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



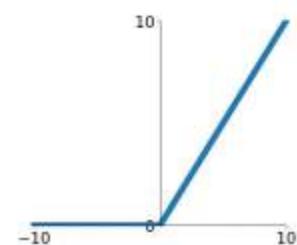
tanh

$$\tanh(x)$$



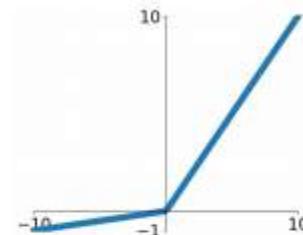
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

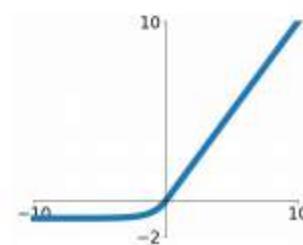


Maxout

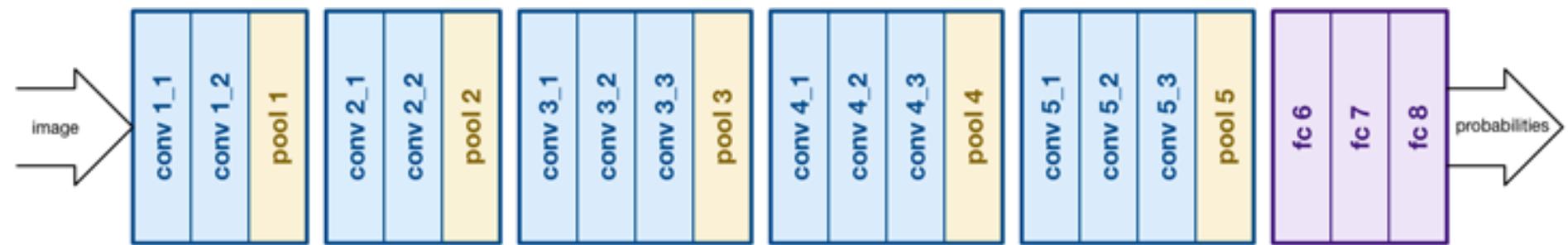
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

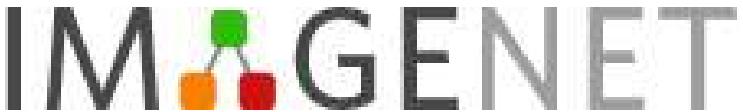
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



A typical CNN structure



ImageNet Challenge



- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- Challenge: 1.2 million training images, 1000 classes

www.image-net.org/challenges/LSVRC/

ImageNet Challenge



GPUS

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk

+

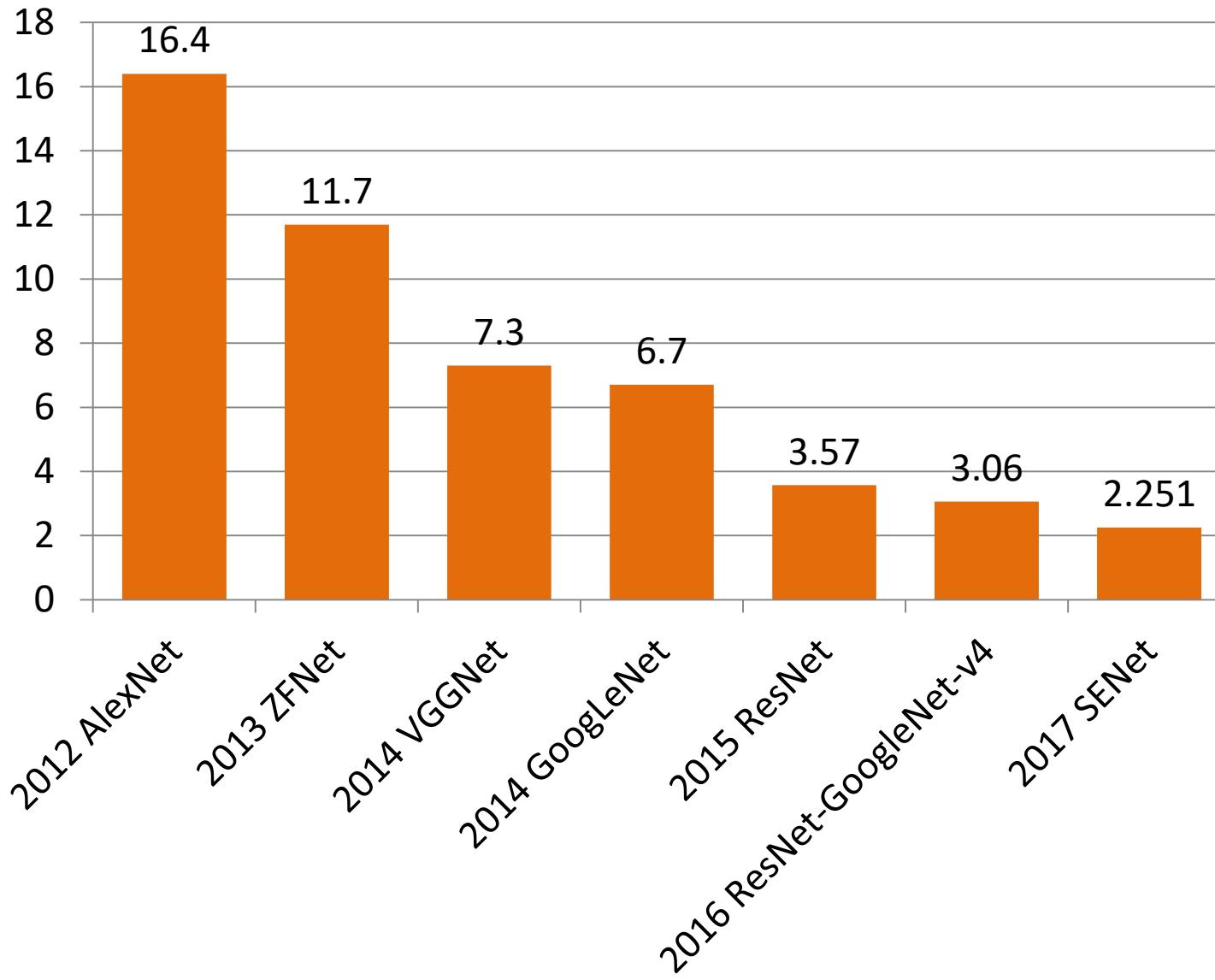
Data

- Challenge: 1.2 million training images, 1000 classes

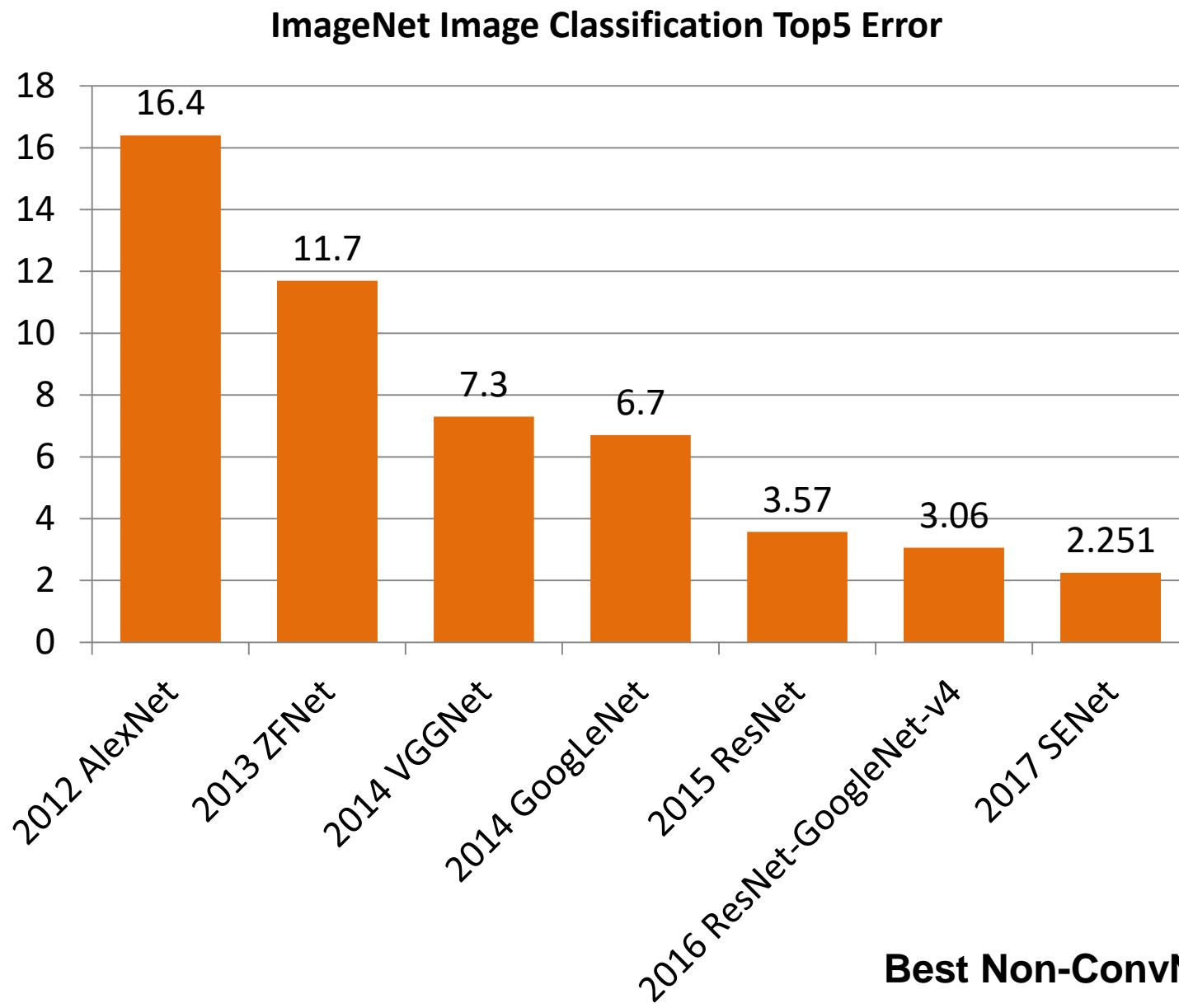
www.image-net.org/challenges/LSVRC/

Progress on ImageNet Challenge

ImageNet Image Classification Top5 Error



Progress on ImageNet Challenge



Things to remember

- Neural network and Image
 - Neuroscience, Perceptron, Problems due to High Dimensionality and Local Relationship
- Convolutional neural network (CNN)
 - Convolution Layer,
 - Nonlinearity Layer,
 - Pooling Layer,
 - Fully Connected Layer,
 - Loss/Classification Layer
- Progress on ImageNet challenge
 - Latest SENet, Winner 2017

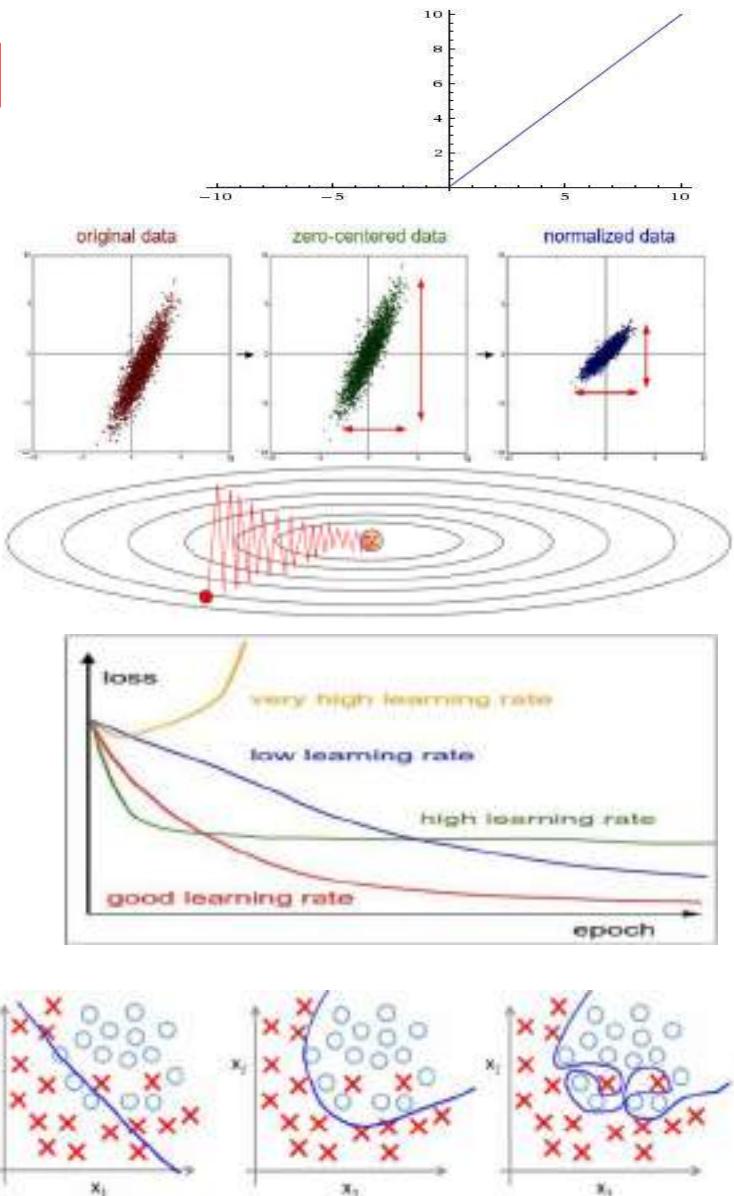
Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Antonio Torralba
 - Rob Fergus
 - Leibe
 - And many more

Next Class

Training Aspects of CNN

- Activation Functions
- Dataset Preparation
- Data Preprocessing
- Weight Initialization
- Optimization Methods
- Learning Rate
- Transfer Learning
- Generalization



Computer Vision

CNN Training 1

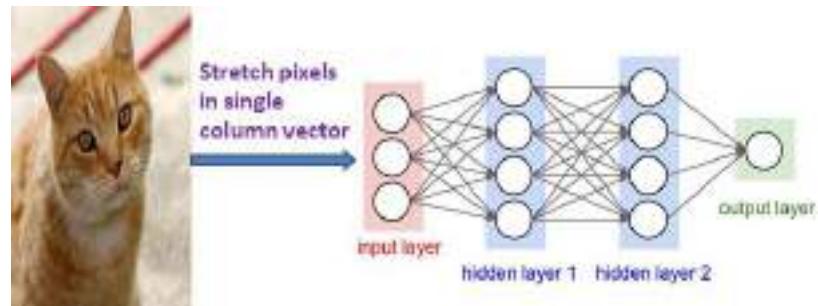
Dr. Mrinmoy Ghorai

**Indian Institute of Information Technology
Sri City, Chittoor**

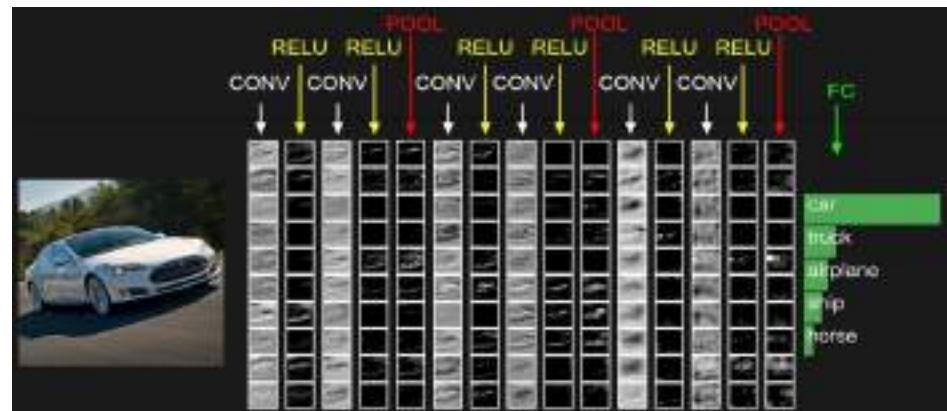


Previous Class

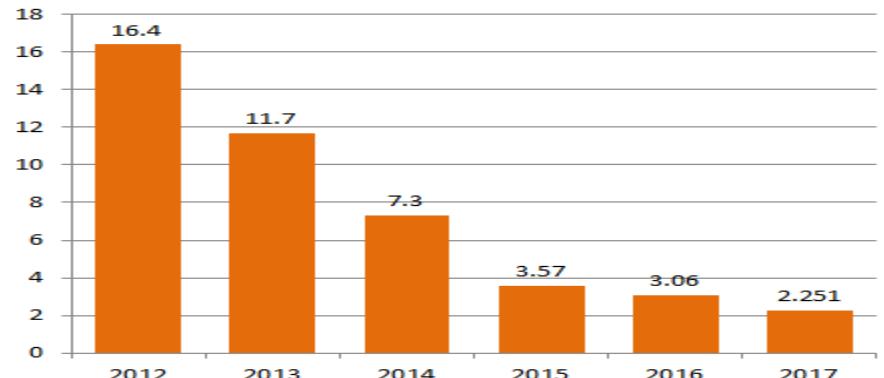
- Neural Network and Image
 - Dimensionality
 - Local relationship



- Convolutional Neural Network (CNN)
 - Convolution Layer
 - Non-linearity Layer
 - Pooling Layer
 - Fully Connected Layer
 - Classification Layer



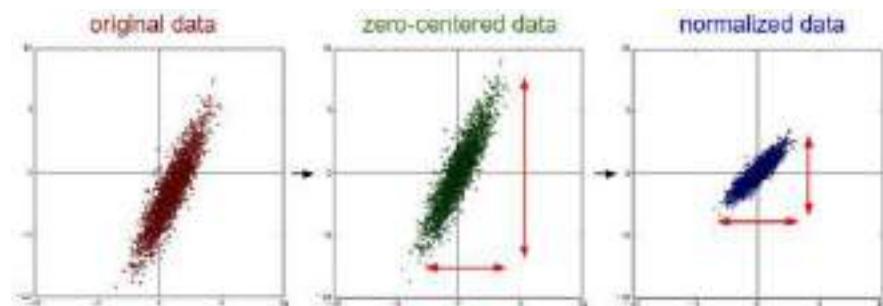
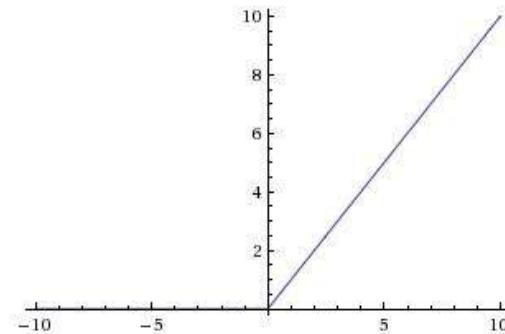
- ImageNet Challenge
 - Progress
 - Human Level Performance



This Class

Training Aspects of CNN

- Activation Functions
- Dataset Preparation
- Data Preprocessing
- Weight Initialization



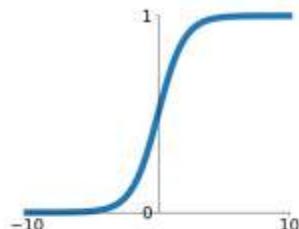
Activation Functions

Non-linearity Layer

Activation Functions

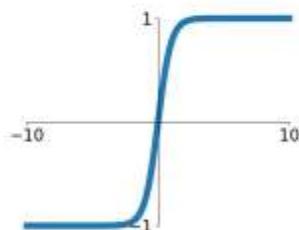
Sigmoid

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



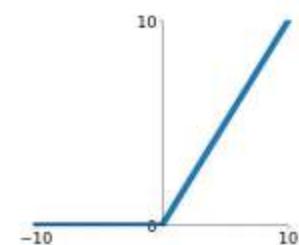
tanh

$$\tanh(x)$$



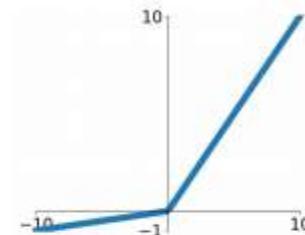
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

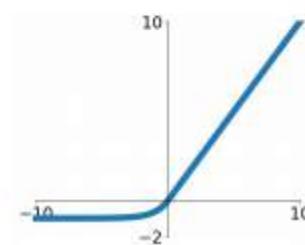


Maxout

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

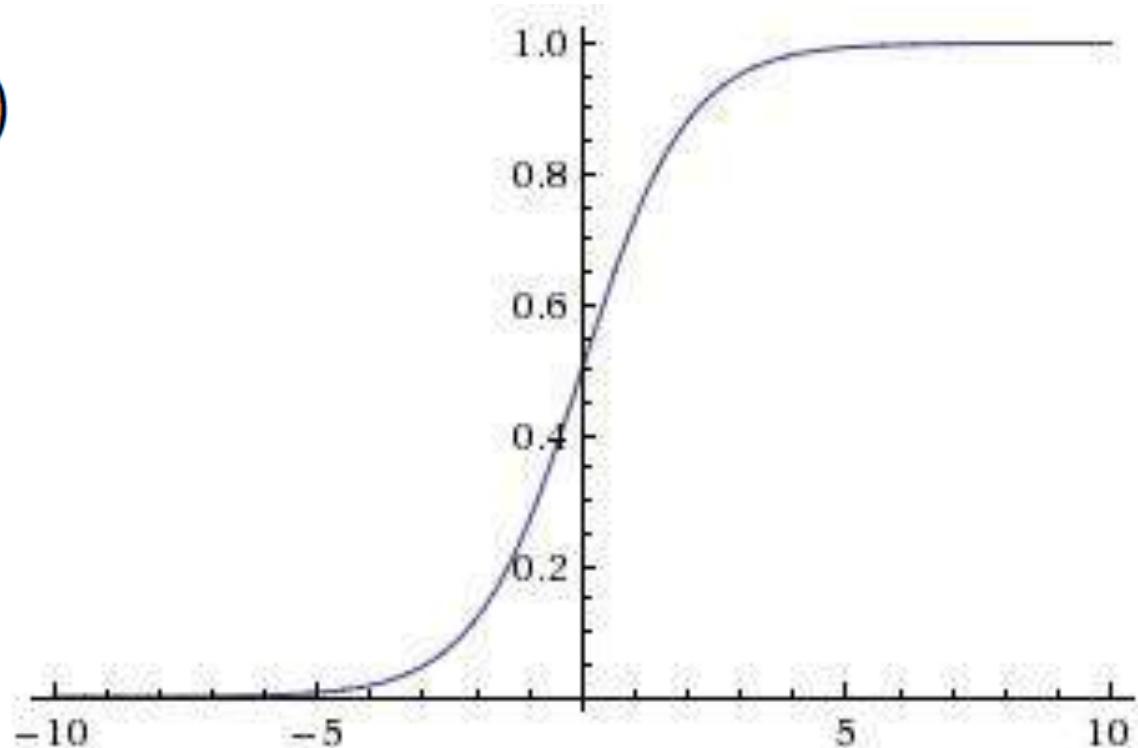
ELU

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



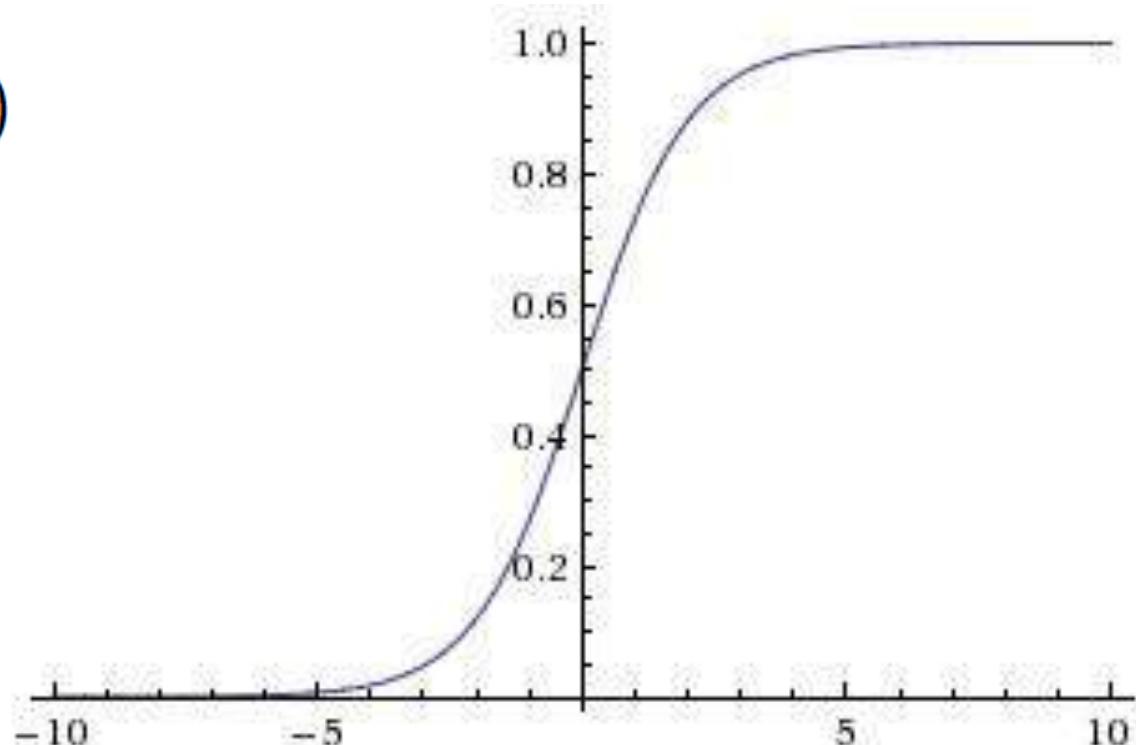
Activation Functions: Sigmoid

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



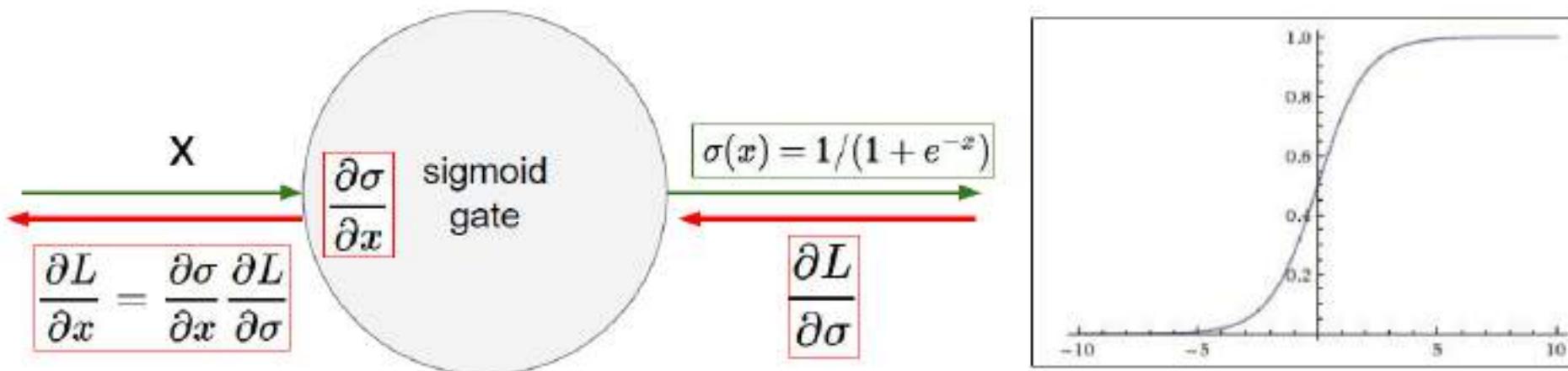
Activation Functions: Sigmoid

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



- Sigmoids saturate and kill gradients.

Activation Functions: Sigmoid



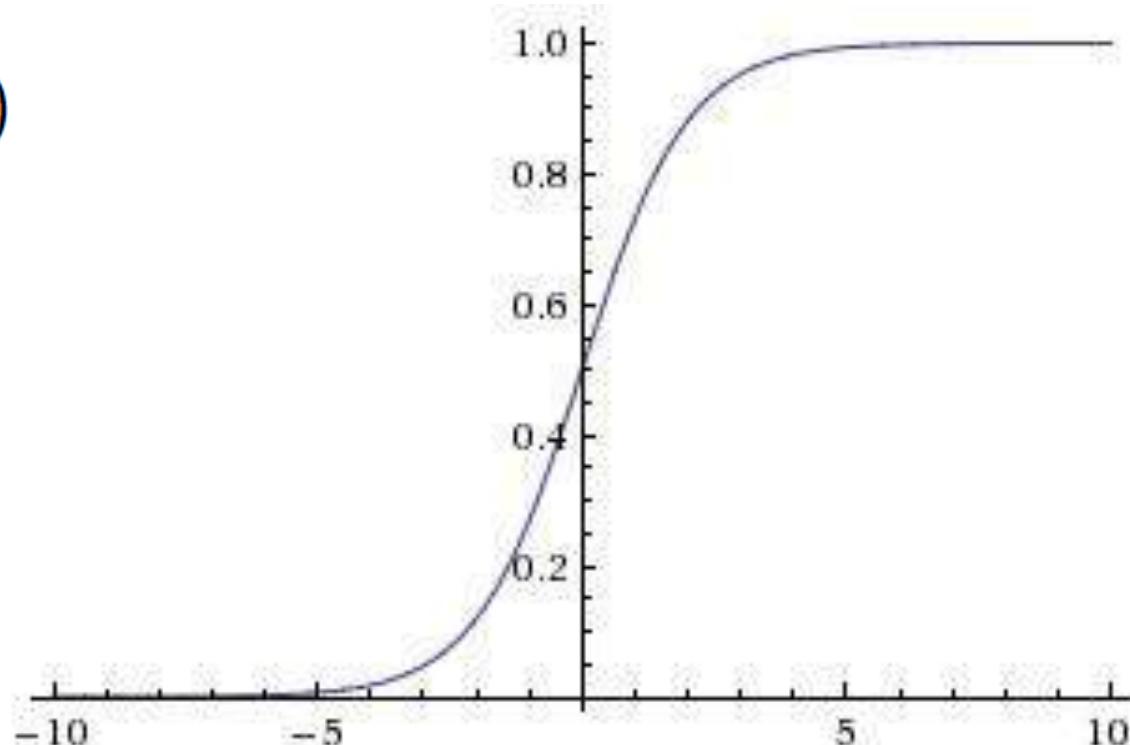
What happens when $x = -10$?

What happens when $x = 0$?

What happens when $x = 10$?

Activation Functions: Sigmoid

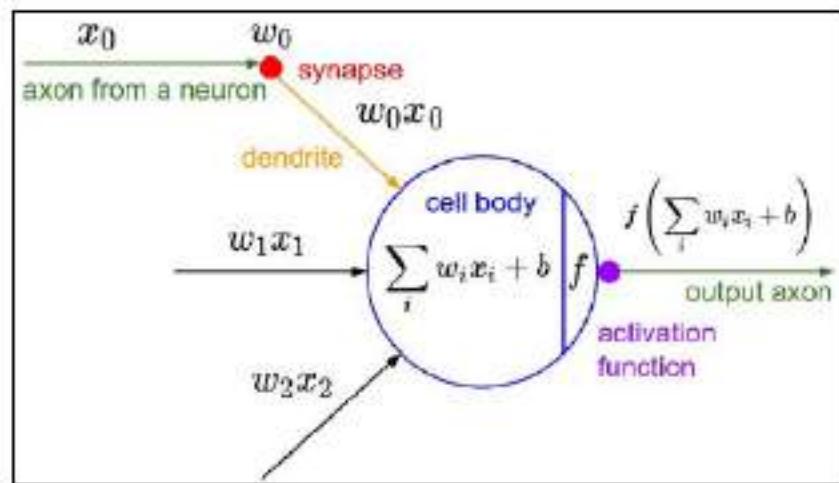
$$\sigma(x) = 1/(1 + e^{-x})$$



- Sigmoids saturate and kill gradients.
- Sigmoid outputs are not zero-centered.

Activation Functions: Sigmoid

Consider what happens when the input to a neuron (x) is always positive:

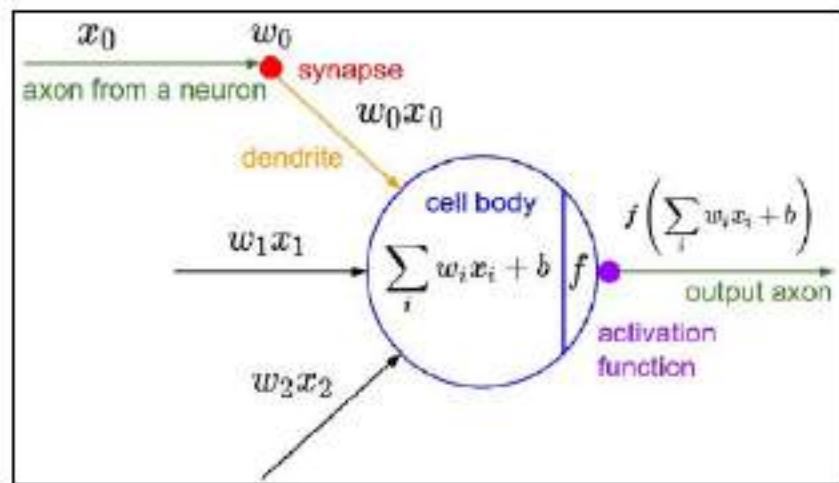


$$f \left(\sum_i w_i x_i + b \right)$$

What can we say about the gradients on w ?

Activation Functions: Sigmoid

Consider what happens when the input to a neuron (x) is always positive:



$$f \left(\sum_i w_i x_i + b \right)$$

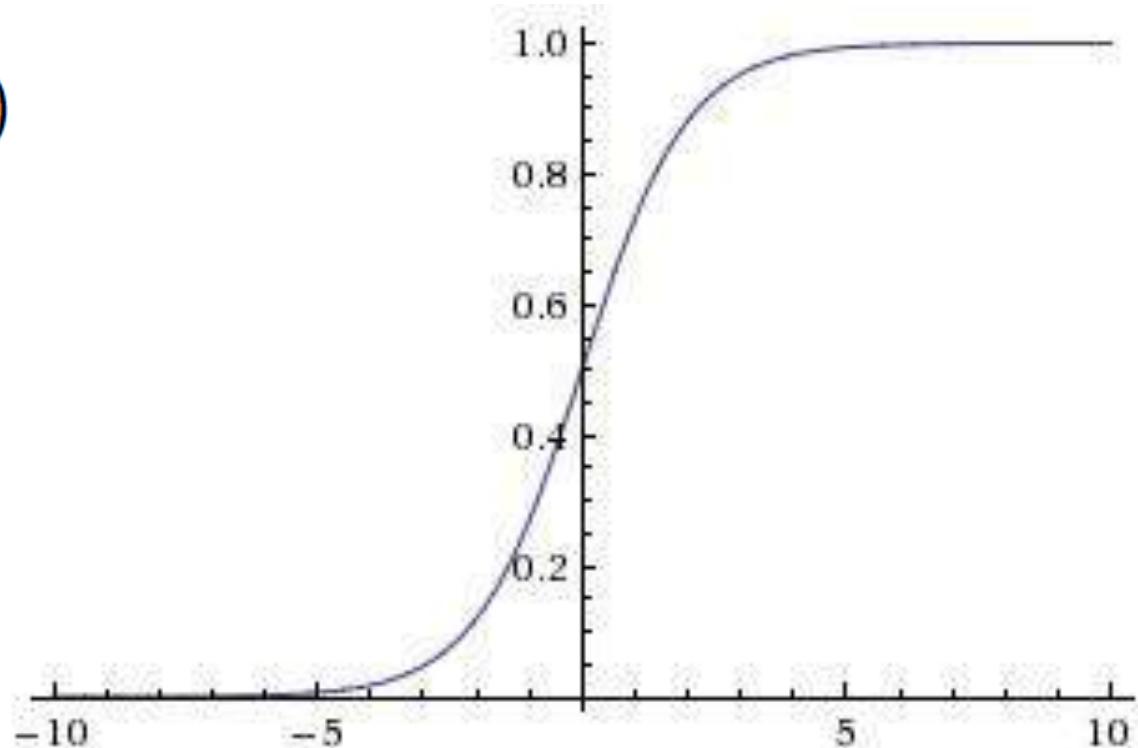
What can we say about the gradients on w ?

Always all positive or all negative

(this is also why you want zero-mean data!)

Activation Functions: Sigmoid

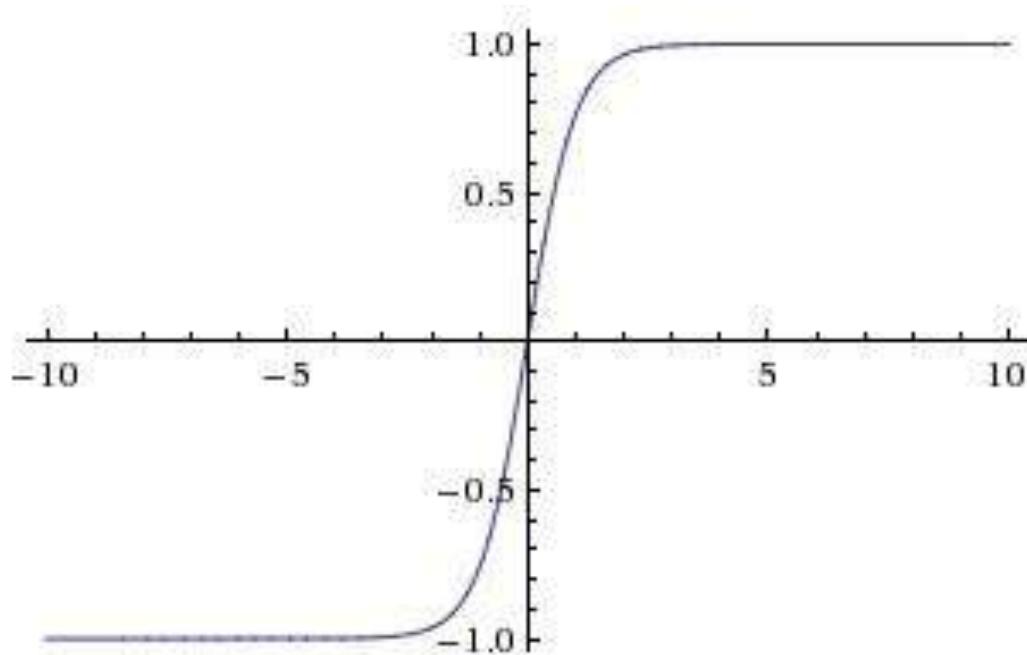
$$\sigma(x) = 1/(1 + e^{-x})$$



- Sigmoids saturate and kill gradients.
- Sigmoid outputs are not zero-centered.
- $\text{Exp}()$ is a bit compute expensive.

Activation Functions: tanh

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



[LeCun et al., 1991]

Source: <http://cs231n.github.io>

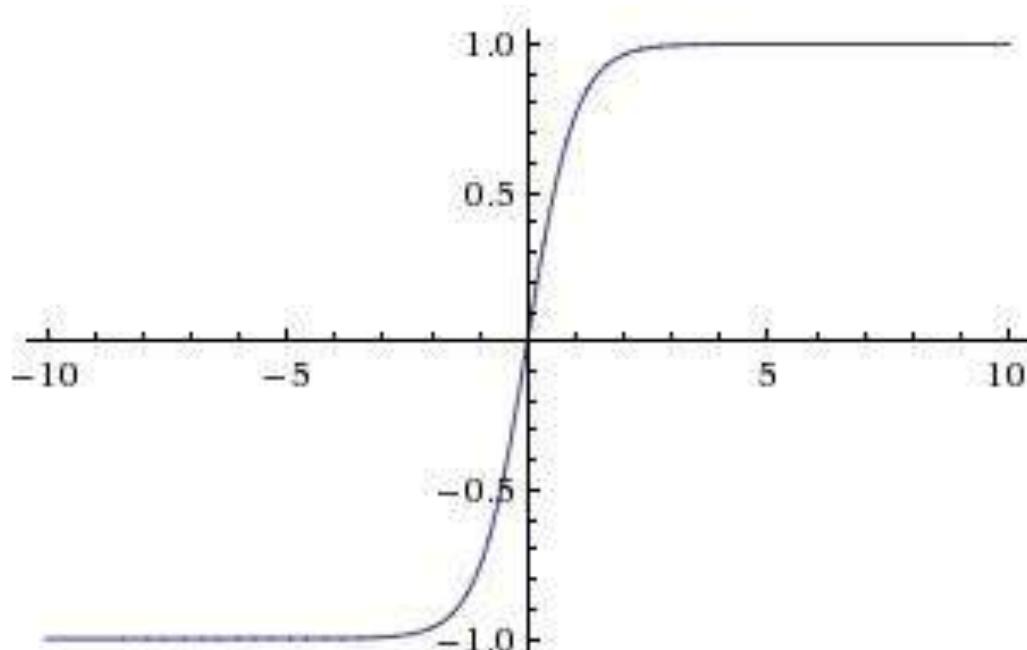
Activation Functions: tanh

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

tanh neuron is simply a scaled sigmoid neuron

$$\tanh(x) = 2\sigma(2x) - 1.$$

↑
Sigmoid



[LeCun et al., 1991]

Source: <http://cs231n.github.io>

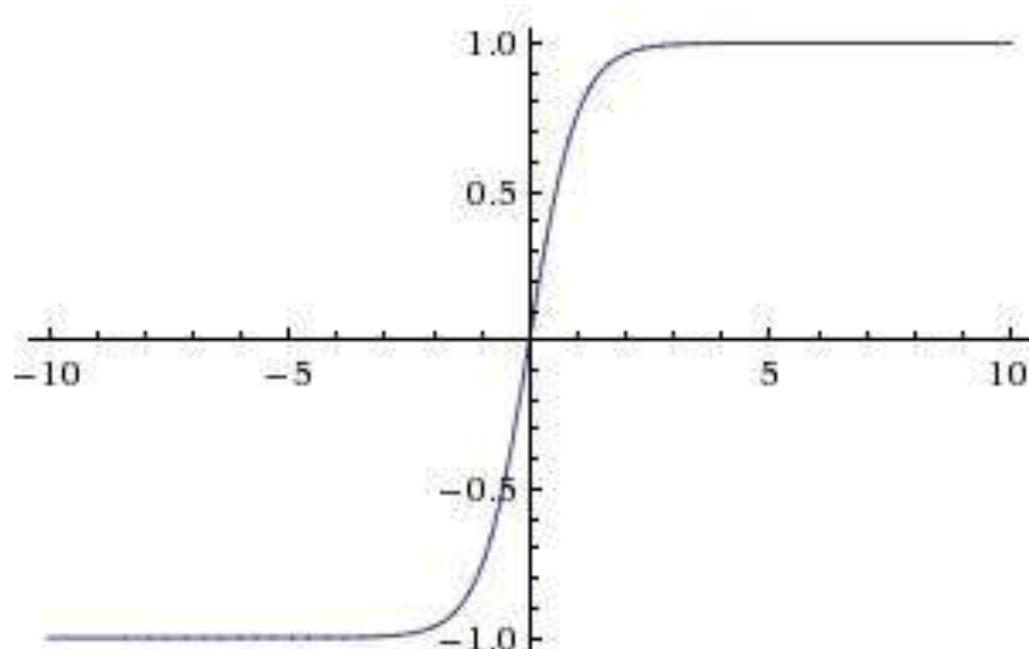
Activation Functions: tanh

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

tanh neuron is simply a scaled sigmoid neuron

$$\tanh(x) = 2\sigma(2x) - 1.$$

↑
Sigmoid



Like the sigmoid neuron, its activations saturate.

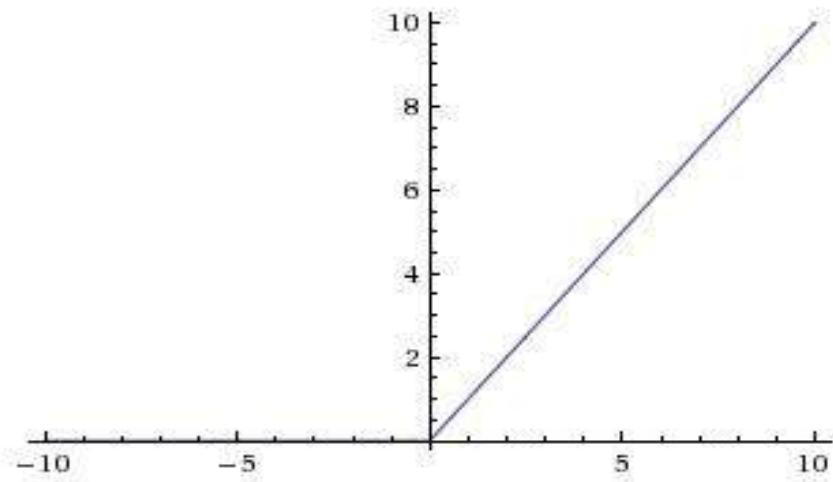
Unlike the sigmoid neuron its output is zero-centered.

In practice the *tanh non-linearity is always preferred to the sigmoid nonlinearity.*

[LeCun et al., 1991]

Activation Functions: ReLU

$$f(x) = \max(0, x)$$

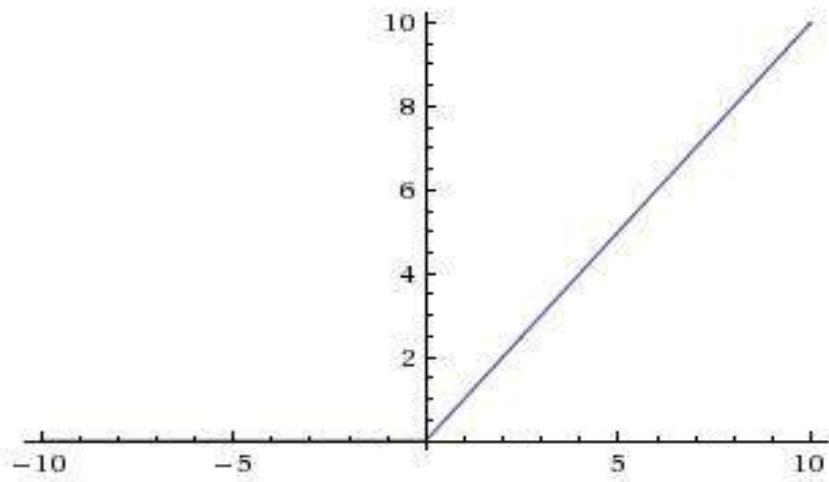


[Krizhevsky et al., 2012]

Source: <http://cs231n.github.io>

Activation Functions: ReLU

$$f(x) = \max(0, x)$$



ReLU is 6 times faster in the convergence of stochastic gradient descent compared to the sigmoid/tanh ([Krizhevsky et al.](#)).

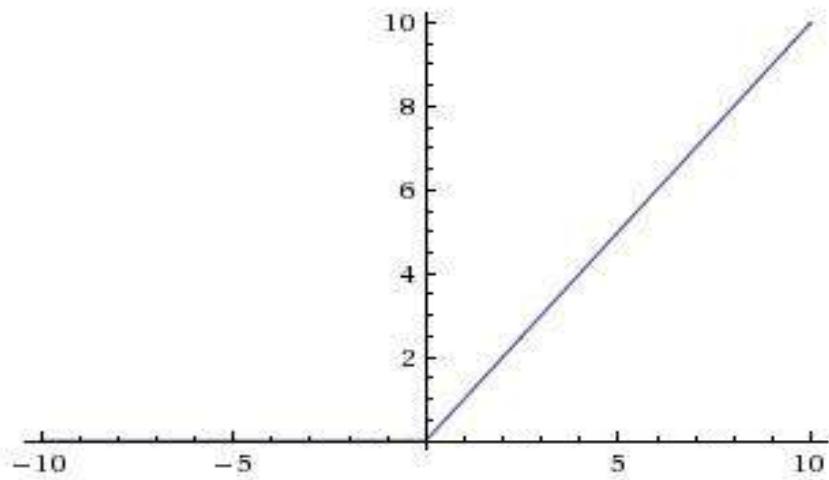
ReLU is simple as compared to tanh/sigmoid that involve expensive operations (exponentials, etc.)

[Krizhevsky et al., 2012]

Source: <http://cs231n.github.io>

Activation Functions: ReLU

$$f(x) = \max(0, x)$$



ReLU is 6 times faster in the convergence of stochastic gradient descent compared to the sigmoid/tanh ([Krizhevsky et al.](#)).

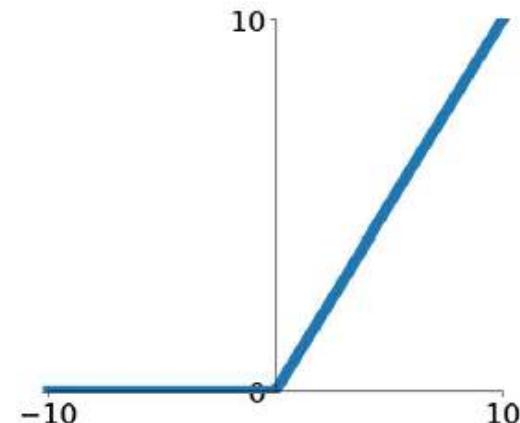
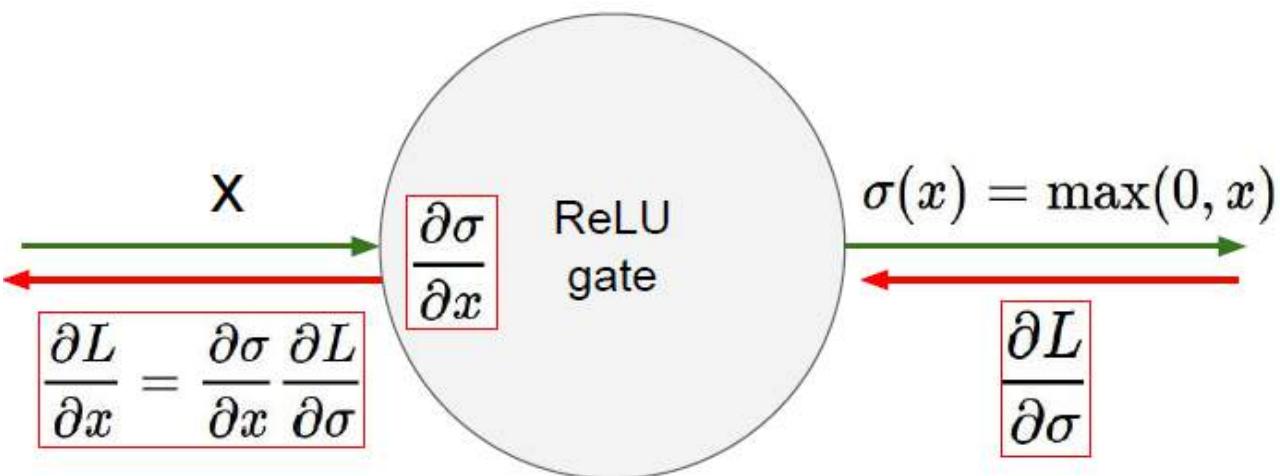
ReLU is simple as compared to tanh/sigmoid that involve expensive operations (exponentials, etc.)

Dying ReLU problem: a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again.

[Krizhevsky et al., 2012]

Source: <http://cs231n.github.io>

Activation Functions: ReLU



What happens when $x = -10$?

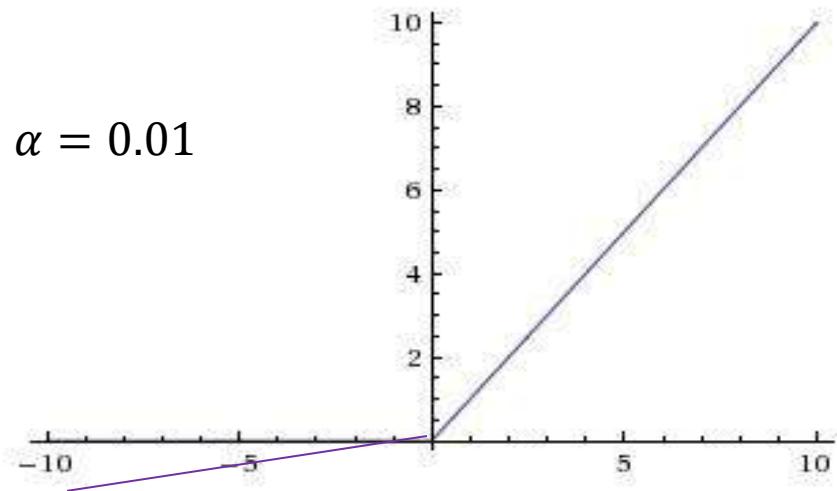
What happens when $x = 0$?

What happens when $x = 10$?

Activation Functions: Leaky ReLU

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

$$\alpha = 0.01$$



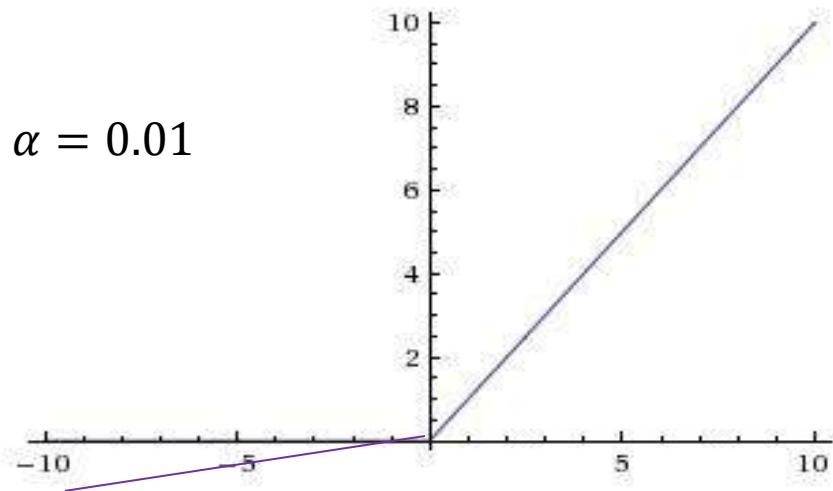
[Mass et al., 2013]

Source: <http://cs231n.github.io>

Activation Functions: Leaky ReLU

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

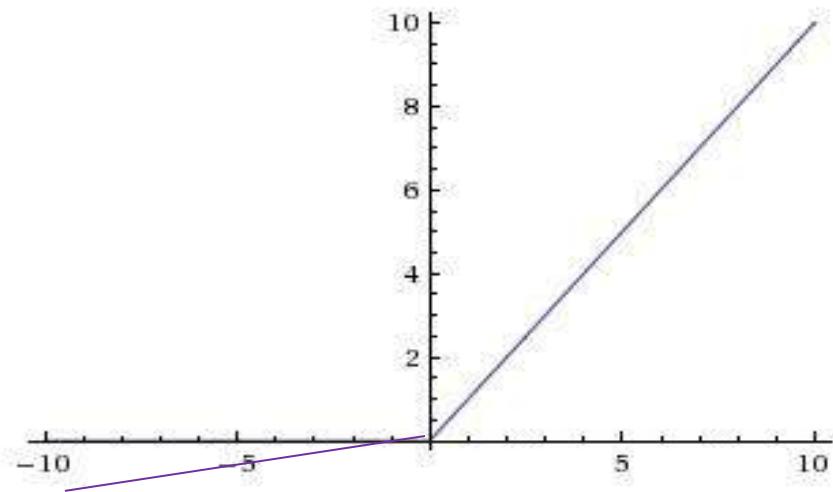
$$\alpha = 0.01$$



Succeeded in some cases, but the results are not always consistent.

Activation Functions: Parametric ReLU

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$$



In PReLU, the slope in the negative region is considered as a parameter of each neuron and learnt from data.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE international conference on computer vision (CVPR)*.

Activation Functions: Maxout

Maxout neuron (introduced by [Goodfellow et al.](#)) generalizes the ReLU and its leaky version.

The Maxout neuron computes the function:

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

[Goodfellow et al., 2013]

Source: <http://cs231n.github.io>

Activation Functions: Maxout

Maxout neuron (introduced by [Goodfellow et al.](#)) generalizes the ReLU and its leaky version.

The Maxout neuron computes the function:

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

Both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU, we have $w1=0, b1=0$, $w2=identity$, and $b2=0$).

[Goodfellow et al., 2013]

Source: <http://cs231n.github.io>

Activation Functions: Maxout

Maxout neuron (introduced by Goodfellow et al.) generalizes the ReLU and its leaky version.

The Maxout neuron computes the function:

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

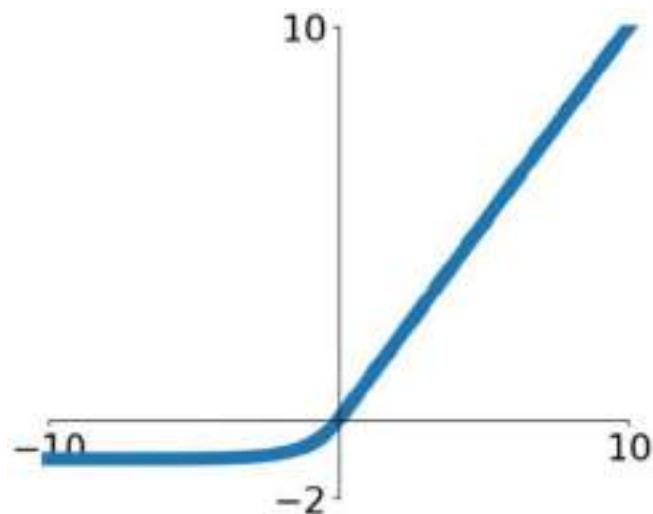
Both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU, we have $w1=0, b1=0$, $w2=identity$, and $b2=0$).

Unlike the ReLU neurons it doubles the number of parameters.

[Goodfellow et al., 2013]

Source: <http://cs231n.github.io>

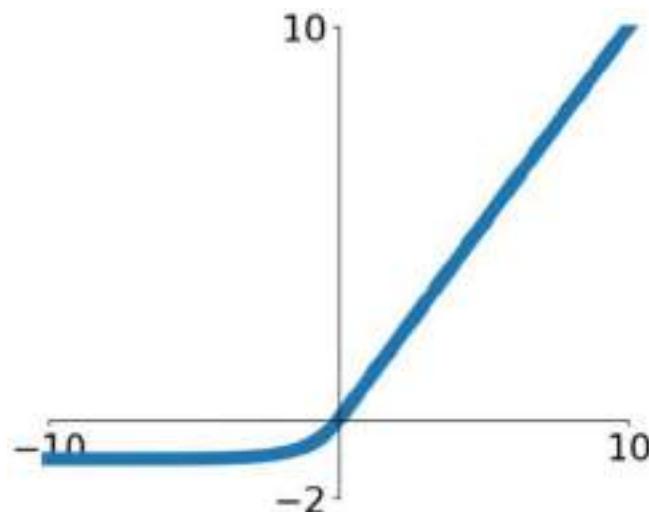
Activation Functions: ELU



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Exponential Linear Unit

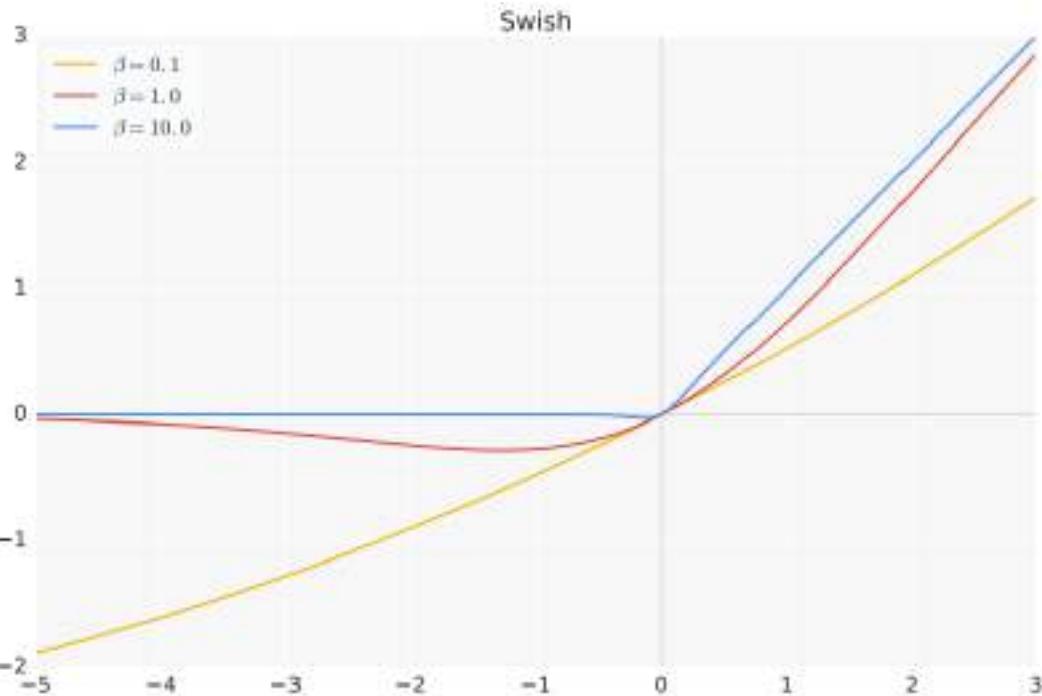
Activation Functions: ELU



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Exponential Linear Unit
- All benefits of ReLU
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise
- Computation requires $\exp()$

Activation Functions: Swish



$$f(x) = x \cdot \text{sigmoid}(\beta x)$$

- ReLU is special case of Swish

Activation Functions: In Practice

- Use ReLU. Be careful with your learning rates
- Try out Swish/Leaky ReLU / Maxout / ELU
- Try out tanh but don't expect much
- Don't use sigmoid

Dataset Preparation

Train/Val/Test sets

In General People Do: Train/Test

- Split data into train and test,
- Choose hyperparameters that work best on test data



In General People Do: Train/Test

- Split data into train and test,
- Choose hyperparameters that work best on test data



BAD: No idea how algorithm will perform on new data

K-Fold Validation

- Split data into folds,
- Try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

K-Fold Validation

- Split data into folds,
- Try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but not used too frequently in deep learning

Better Approach: Train/Val/Test sets

- Split data into **train**, **val**, and **test**;
- Choose hyperparameters on **val** and evaluate on **test**



Better Approach: Train/Val/Test sets

- Split data into **train**, **val**, and **test**;
- Choose hyperparameters on val and evaluate on test



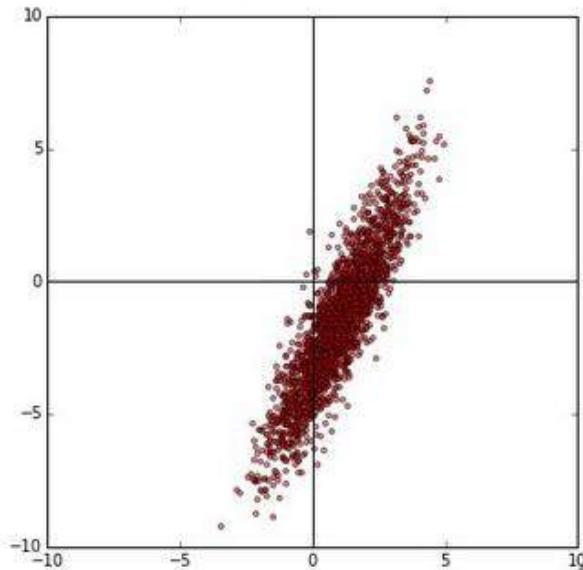
Division can be done based on the size of dataset:

- Roughly 10k or 10% whichever is less for val and test sets.
- Rest in train set.

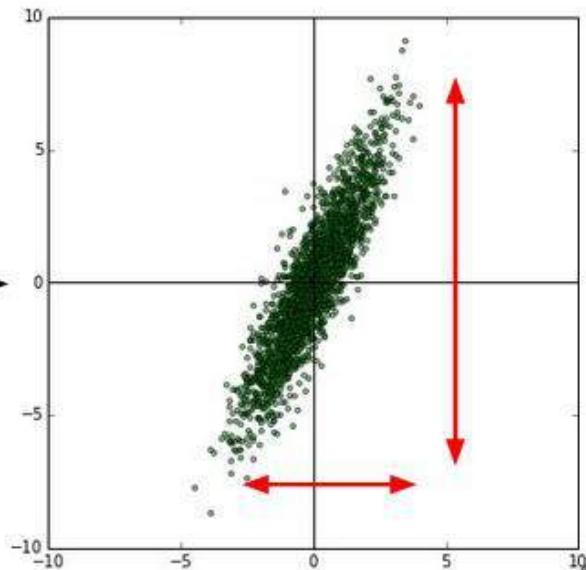
Data Preprocessing

Data Preprocessing

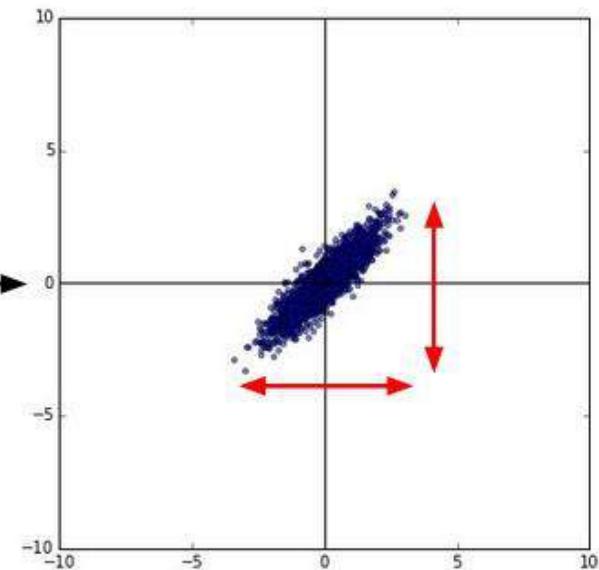
original data



zero-centered data

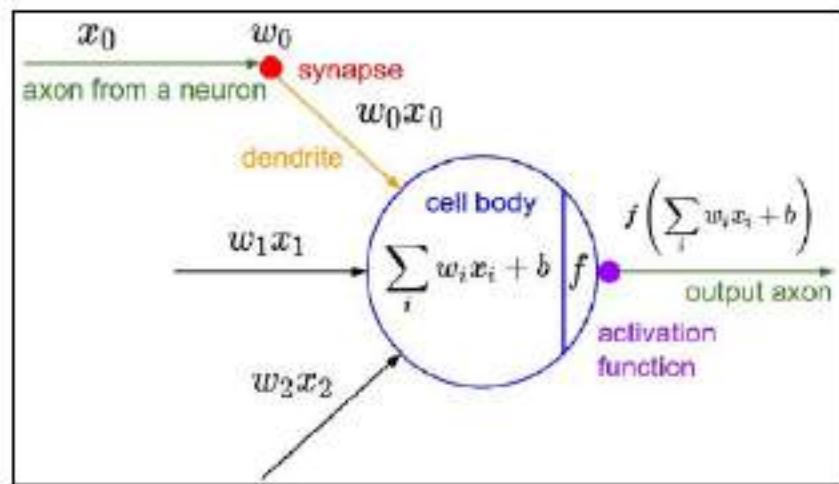


normalized data



Data Preprocessing

Consider what happens when the input to a neuron (x) is always positive:

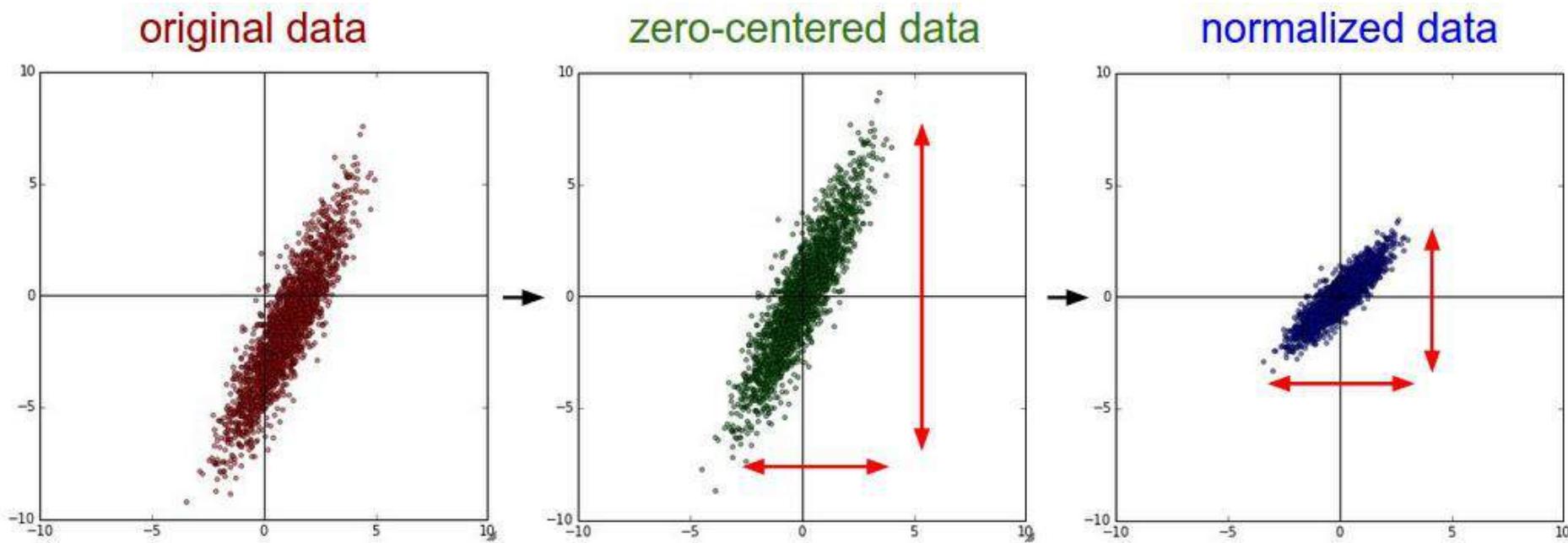


$$f \left(\sum_i w_i x_i + b \right)$$

What can we say about the gradients on w ?

Always all positive or all negative
(this is also why you want zero-mean data!)

Data Preprocessing



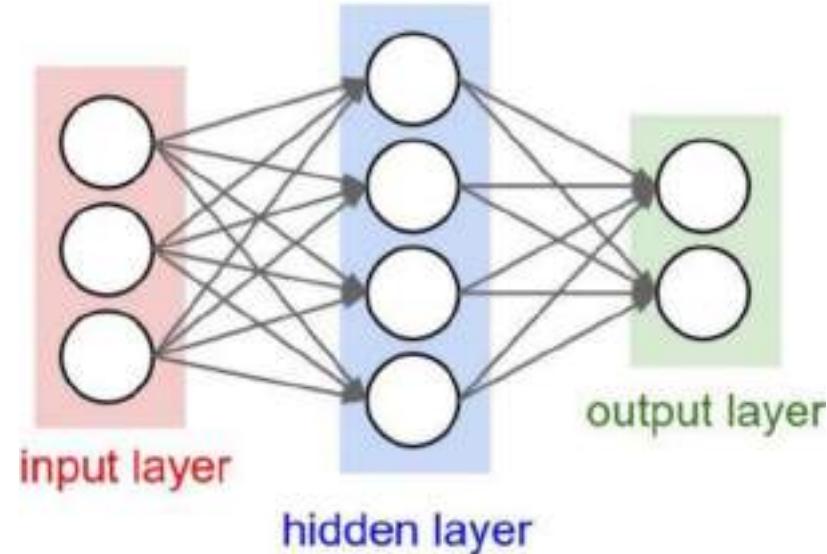
In practice for Images: only centering is preferred
e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
(mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet, ResNet, etc.)
(mean along each channel = 3 numbers)

Weight Initialization

Weight Initialization: Constant

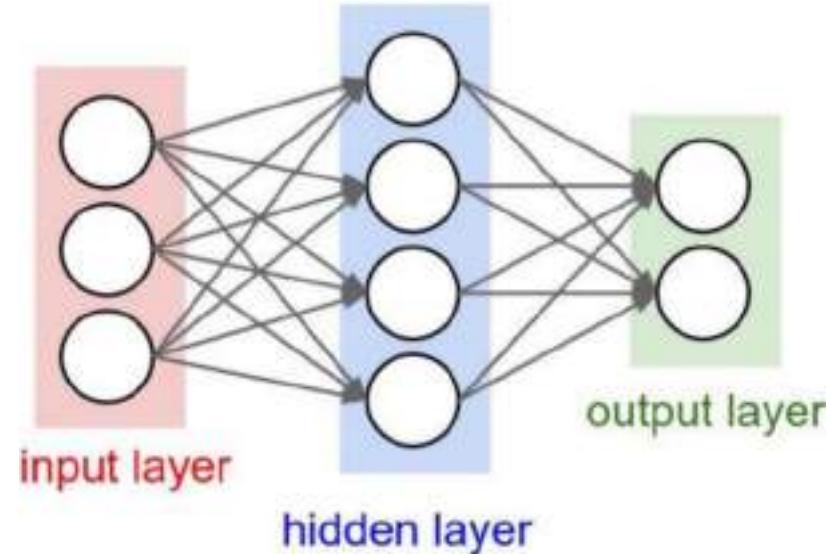
Q: what happens when $W=\text{Constant}$ init is used?



Weight Initialization: Constant

Q: what happens when $W=\text{Constant}$ init is used?

- Every neuron will compute the same output and undergo the exact same parameter updates.
- There is no source of asymmetry between neurons if their weights are initialized to be the same.



Weight Initialization: Gaussian

First idea: **Small random numbers**

(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for
different neurons

Weight Initialization: Gaussian

First idea: **Small random numbers**

(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks,

i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Weight Initialization: Gaussian

First idea: **Small random numbers**

(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks,

i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Increase the standard deviation to 1

Weight Initialization: Gaussian

First idea: **Small random numbers**

(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks,

i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Increase the standard deviation to 1

Almost all neurons completely saturated, either -1 or 1. Gradients will be all zero.

-> gradient diminishing problem.

Weight Initialization: Gaussian

Lets look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)

act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer

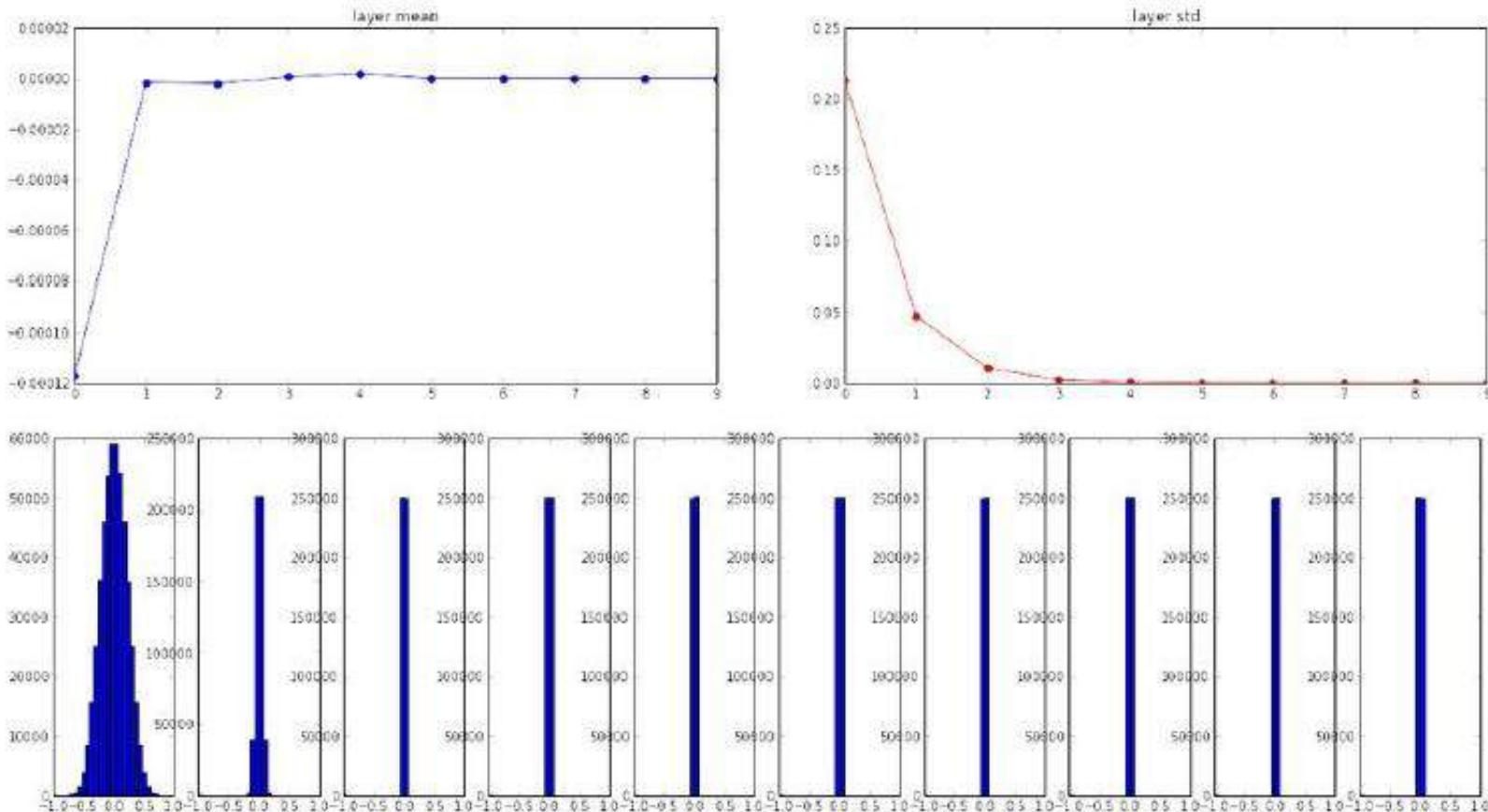
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

Weight Initialization: Gaussian

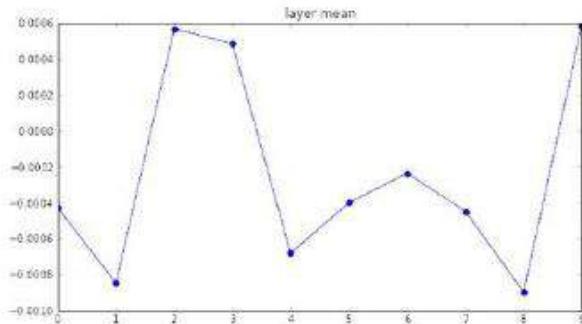
```
input layer had mean 0.000927 and std 0.998388  
hidden layer 1 had mean -0.000117 and std 0.213081  
hidden layer 2 had mean -0.000001 and std 0.047551  
hidden layer 3 had mean -0.000002 and std 0.010630  
hidden layer 4 had mean 0.000001 and std 0.002378  
hidden layer 5 had mean 0.000002 and std 0.000532  
hidden layer 6 had mean -0.000000 and std 0.000119  
hidden layer 7 had mean 0.000000 and std 0.000026  
hidden layer 8 had mean -0.000000 and std 0.000006  
hidden layer 9 had mean 0.000000 and std 0.000001  
hidden layer 10 had mean -0.000000 and std 0.000000
```



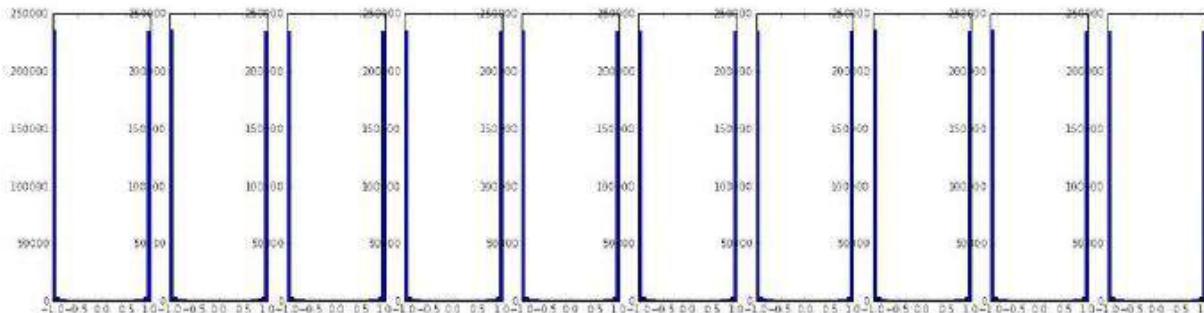
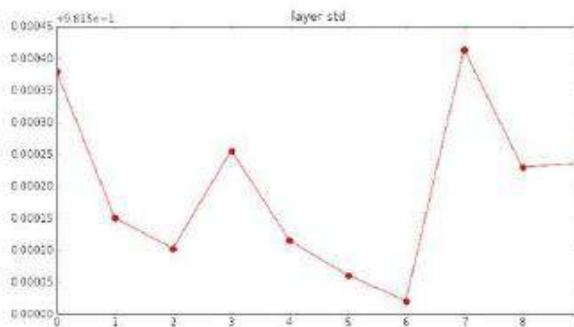
Weight Initialization: Gaussian

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311  
hidden layer 1 had mean -0.000430 and std 0.981879  
hidden layer 2 had mean -0.000849 and std 0.981649  
hidden layer 3 had mean 0.000566 and std 0.981601  
hidden layer 4 had mean 0.000483 and std 0.981755  
hidden layer 5 had mean -0.000682 and std 0.981614  
hidden layer 6 had mean -0.000461 and std 0.981560  
hidden layer 7 had mean -0.000237 and std 0.981520  
hidden layer 8 had mean -0.000448 and std 0.981913  
hidden layer 9 had mean -0.000899 and std 0.981728  
hidden layer 10 had mean 0.000584 and std 0.981736
```



*1.0 instead of *0.01



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

Weight Initialization: Xavier

Calibrating the variances with $1/\sqrt(\text{fan_in})$

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

Reasonable initialization.

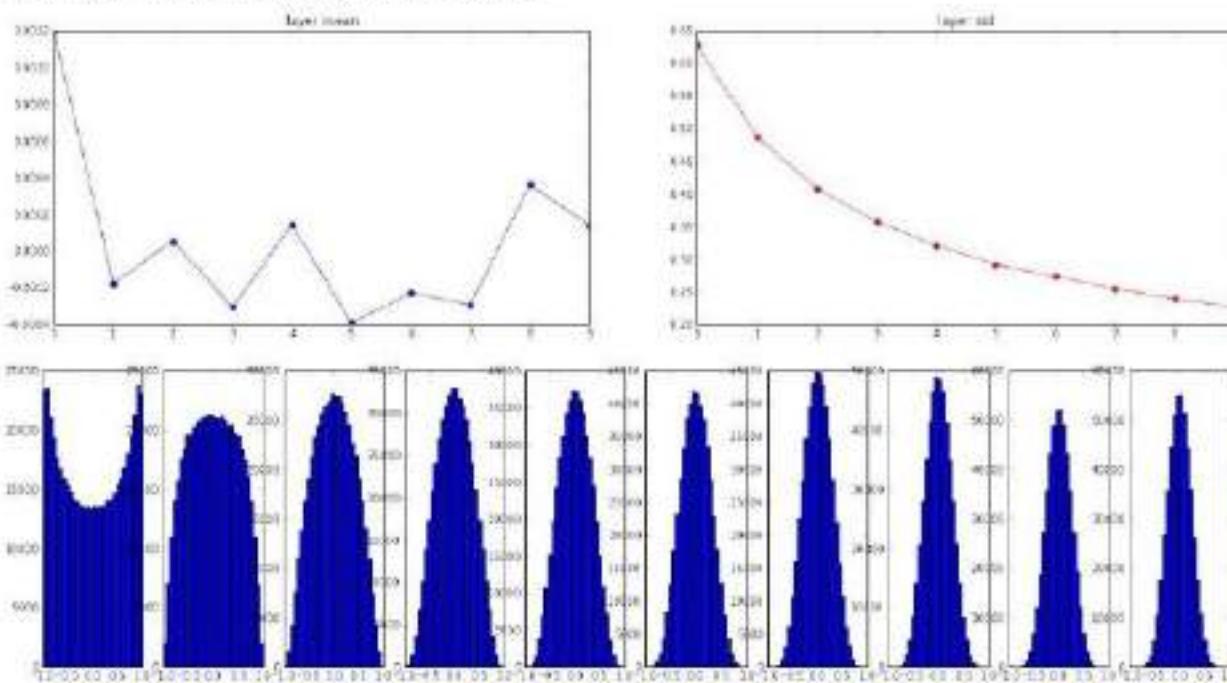
(Mathematical derivation assumes linear activations)

Weight Initialization: Xavier

```
Input layer had mean 0.001890 and std 1.001311  
hidden layer 1 had mean 0.001198 and std 0.627953  
hidden layer 2 had mean -0.000175 and std 0.406051  
hidden layer 3 had mean 0.000055 and std 0.407723  
hidden layer 4 had mean -0.000300 and std 0.357100  
hidden layer 5 had mean 0.000142 and std 0.326917  
hidden layer 6 had mean -0.000389 and std 0.292116  
hidden layer 7 had mean -0.000228 and std 0.273387  
hidden layer 8 had mean -0.000291 and std 0.254935  
hidden layer 9 had mean 0.000361 and std 0.239266  
hidden layer 10 had mean 0.000139 and std 0.228988
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

“Xavier initialization”
[Glorot et al., 2010]



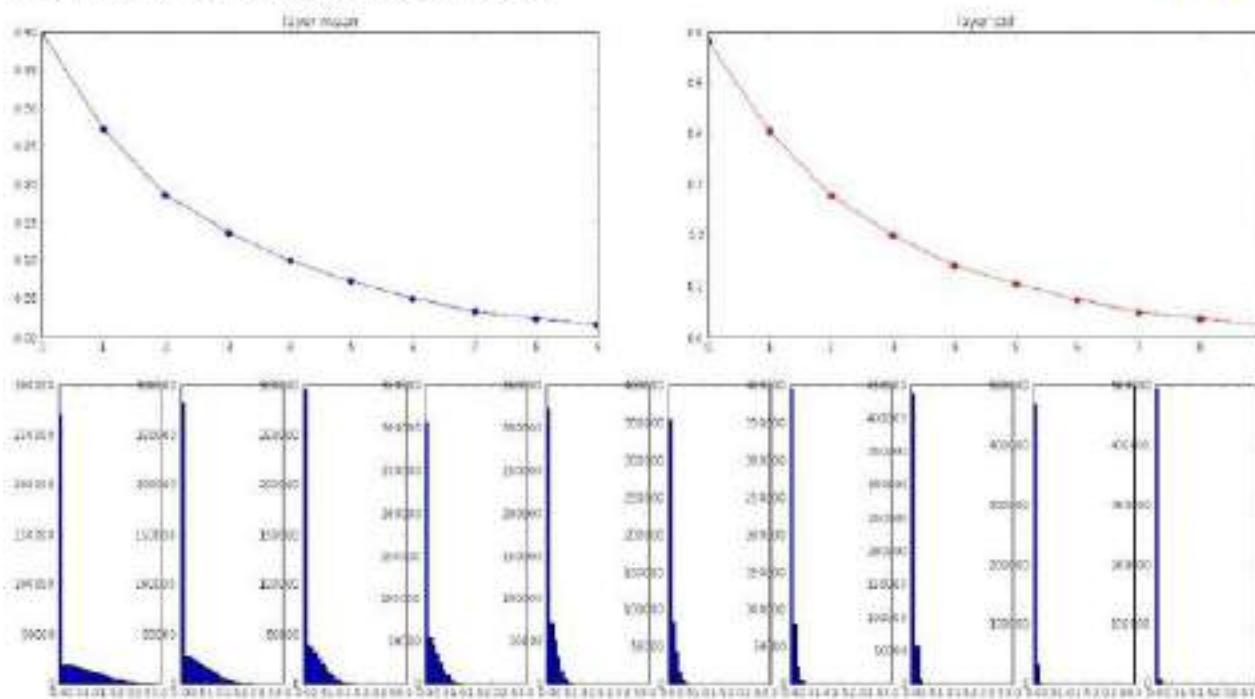
Reasonable initialization.
(Mathematical derivation
assumes linear activations)

Weight Initialization: Xavier

```
input layer had mean 0.000501 and std 0.999444  
hidden layer 1 had mean 0.398623 and std 0.582273  
hidden layer 2 had mean 0.272352 and std 0.403705  
hidden layer 3 had mean 0.186076 and std 0.276912  
hidden layer 4 had mean 0.136442 and std 0.198685  
hidden layer 5 had mean 0.095968 and std 0.148299  
hidden layer 6 had mean 0.072234 and std 0.103200  
hidden layer 7 had mean 0.049775 and std 0.072748  
hidden layer 8 had mean 0.035138 and std 0.051572  
hidden layer 9 had mean 0.025404 and std 0.038583  
hidden layer 10 had mean 0.018408 and std 0.026076
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

but when using the ReLU nonlinearity it breaks.

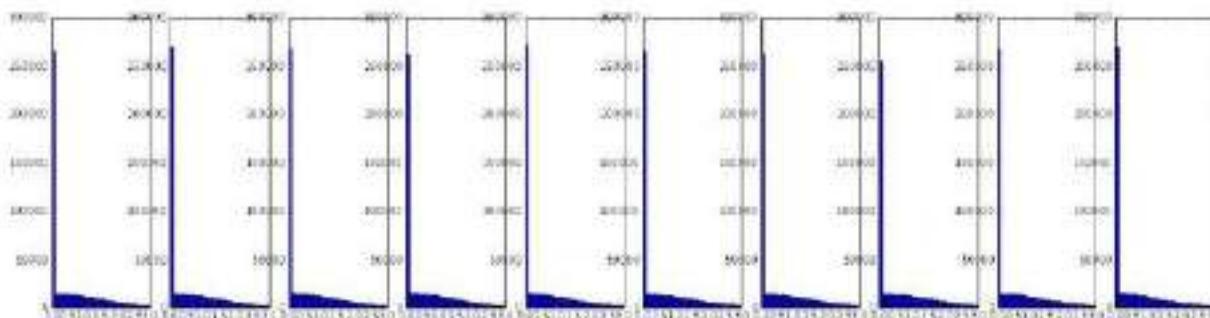
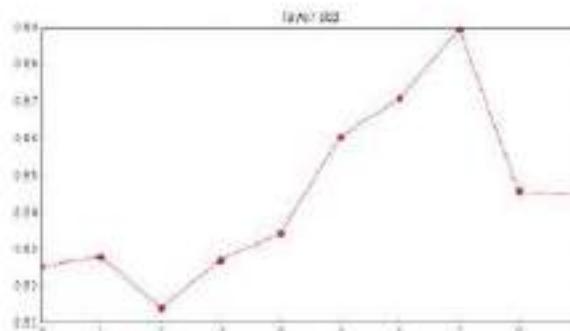
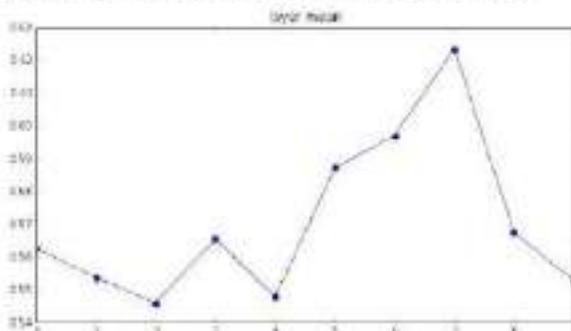


Weight Initialization: XavierImproved

```
input layer had mean 0.000501 and std 0.999444  
hidden layer 1 had mean 0.562488 and std 0.825232  
hidden layer 2 had mean 0.553614 and std 0.827835  
hidden layer 3 had mean 0.545887 and std 0.813055  
hidden layer 4 had mean 0.535395 and std 0.826982  
hidden layer 5 had mean 0.527678 and std 0.834892  
hidden layer 6 had mean 0.520103 and std 0.866835  
hidden layer 7 had mean 0.518687 and std 0.870830  
hidden layer 8 had mean 0.523214 and std 0.889348  
hidden layer 9 had mean 0.527498 and std 0.845357  
hidden layer 10 had mean 0.525231 and std 0.844523
```

```
W = np.random.randn(fan_in, fan_out) X np.sqrt(2/fan_in) # layer initialization
```

He et al., 2015
(note additional 2/)



Proper initialization is an active area of research...

Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init by Mishkin and Matas, 2015

...

Things to remember

- Training CNN
 - Activation Functions: ReLU is common, Swish can be tried
 - Data Preparation: Train/Val/Test
 - Data preprocessing: Centering is common
 - Weight initialization: XavierImproved works well

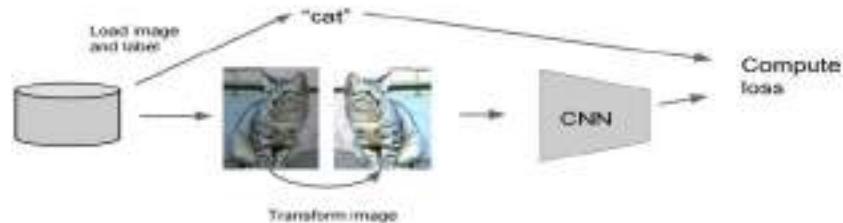
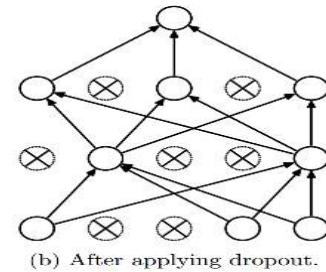
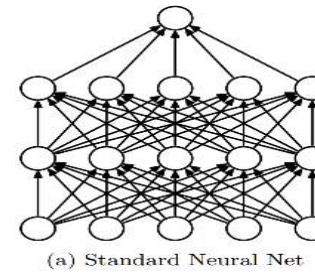
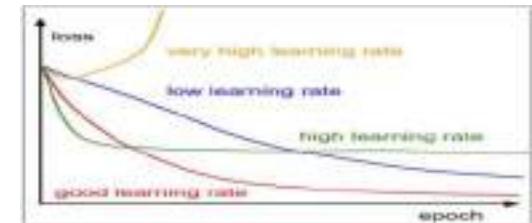
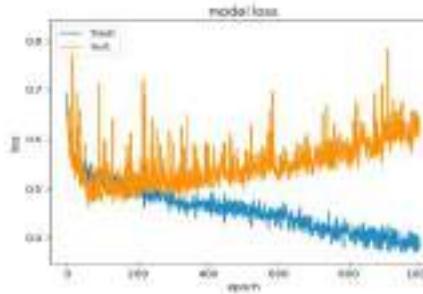
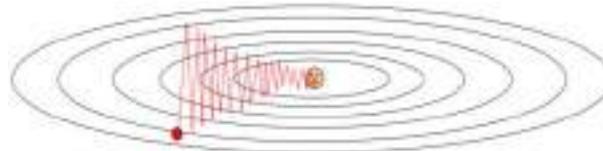
Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Antonio Torralba
 - Rob Fergus
 - Leibe
 - And many more

Next Class

Training Aspects of CNN

- Optimization
- Learning Rate
- Regularization
- Dropout
- Batch Normalization
- Data Augmentation
- Transfer Learning
- Interpreting Loss Curve



Computer Vision

CNN Training 2

Dr. Mrinmoy Ghorai

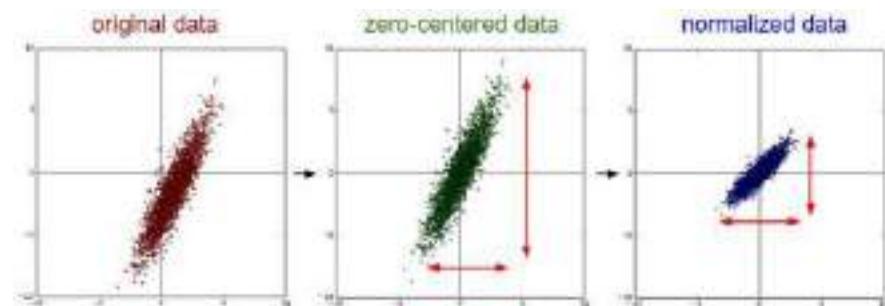
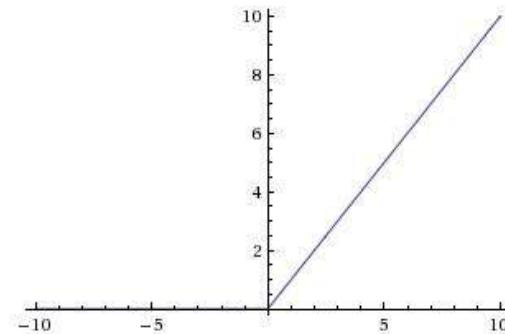
**Indian Institute of Information Technology
Sri City, Chittoor**



Previous Class

Training Aspects of CNN

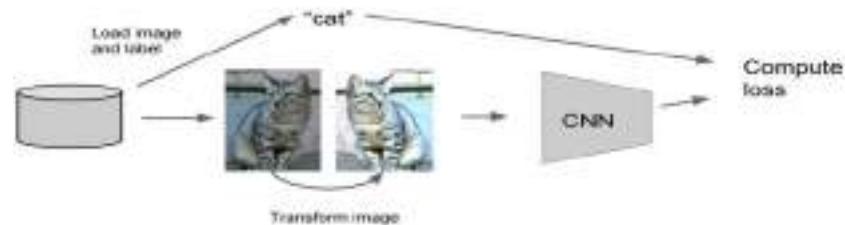
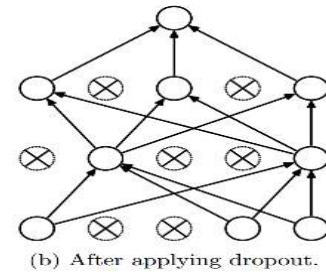
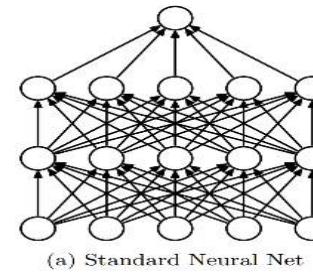
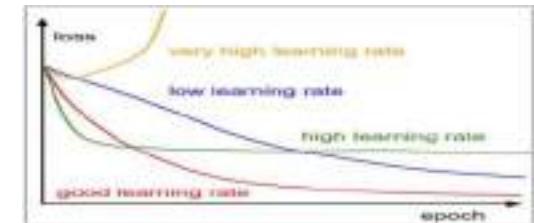
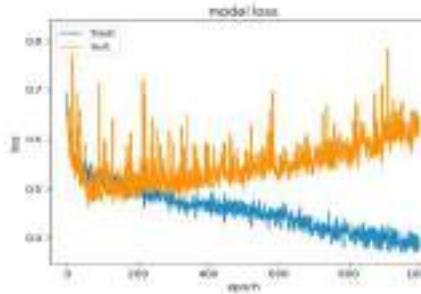
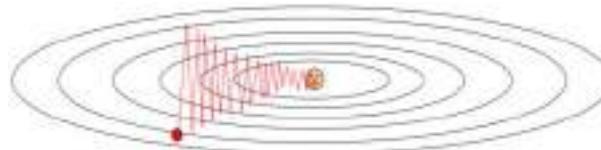
- Activation Functions
- Dataset Preparation
- Data Preprocessing
- Weight Initialization



This Class

Training Aspects of CNN

- Optimization
- Learning Rate
- Regularization
- Dropout
- Batch Normalization
- Data Augmentation
- Transfer Learning
- Interpreting Loss Curve



Optimization



Source: cs231n

Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

Stochastic Gradient Descent (SGD)

The procedure of repeatedly evaluating the **gradient of loss function** and then performing a parameter update.

Vanilla (Original) Gradient Descent:

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

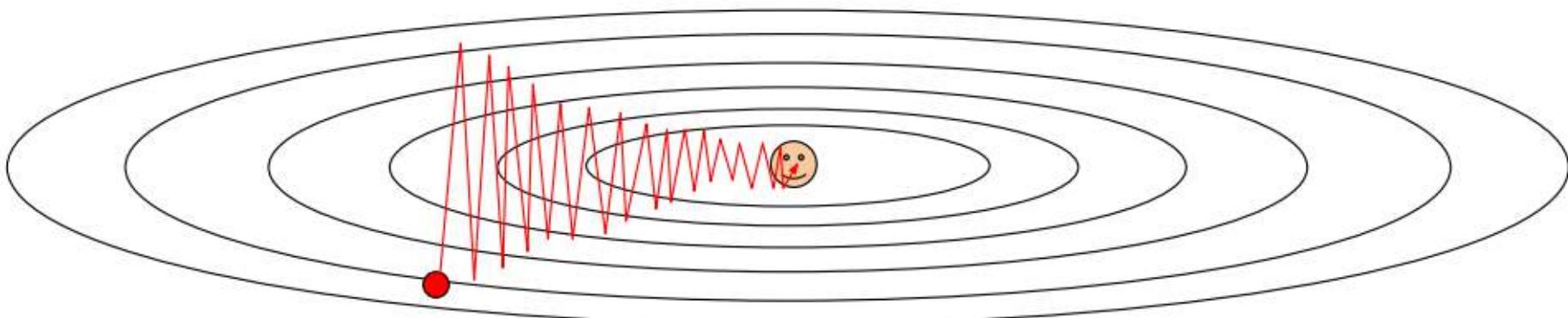
SGD: Problems

What if loss changes quickly in one direction and slowly in another?

SGD: Problems

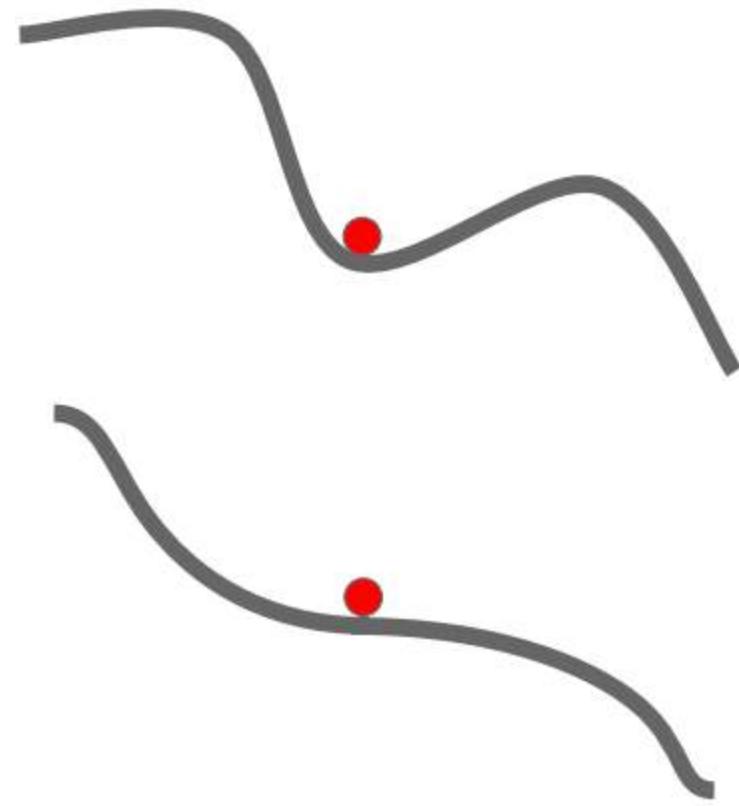
What if loss changes quickly in one direction and slowly in another?

Very slow progress along shallow dimension, jitter along steep dimension



SGD: Problems

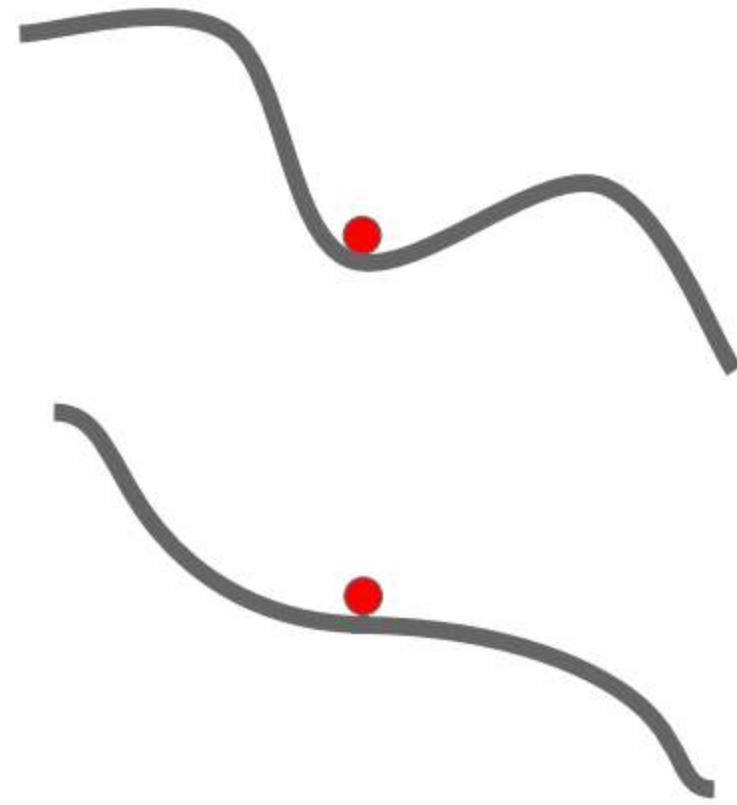
What if the loss function has
a **local minima** or **saddle**
point?



SGD: Problems

What if the loss function has
a **local minima** or **saddle
point**?

Zero gradient,
gradient descent
gets stuck



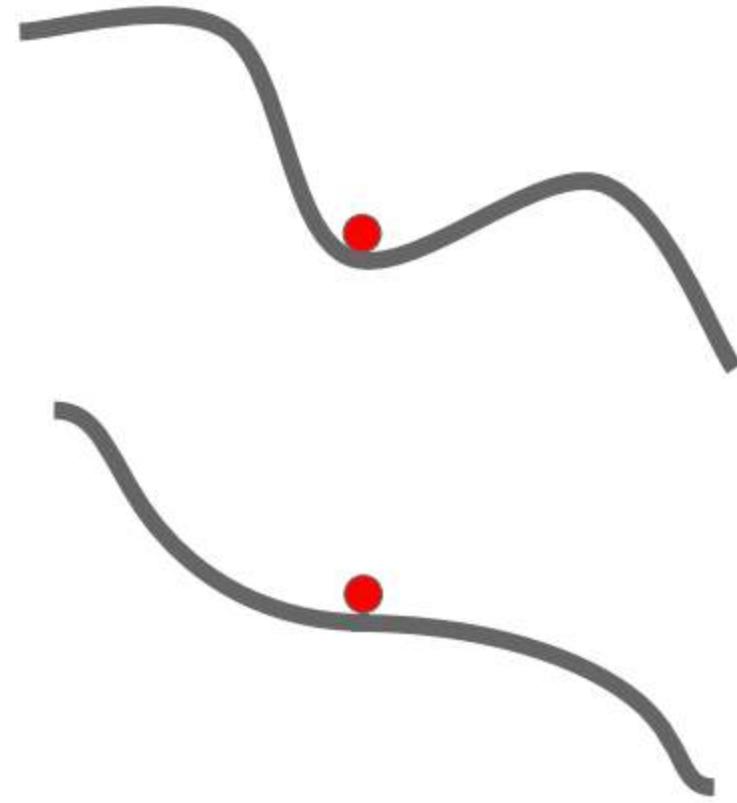
SGD: Problems

What if the loss function has
a **local minima** or **saddle
point**?

Zero gradient,
gradient descent
gets stuck

Saddle points much more
common in high
dimension

Dauphin et al, “Identifying and attacking the saddle
point problem in high-dimensional non-convex
optimization”, NIPS 2014



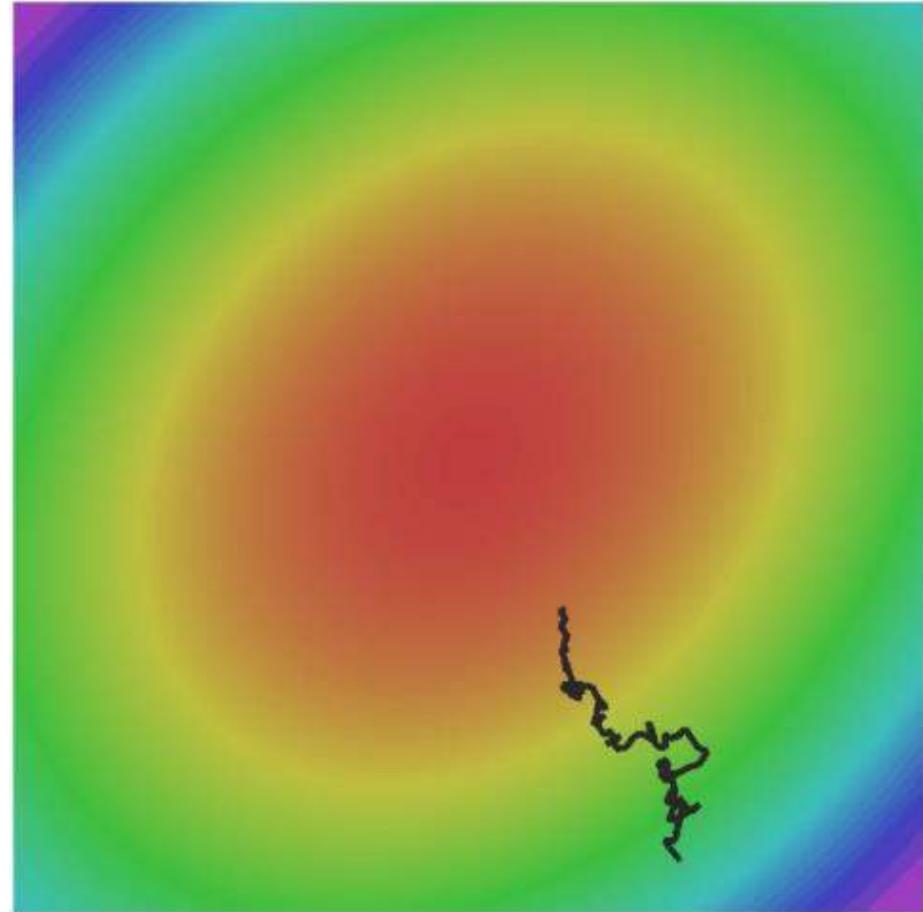
Source: cs231n

SGD: Problems

Our gradients come from
minibatches so they can
be **noisy!**

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

- Build up “velocity” in any direction that has consistent gradient
- Rho gives “friction”; typically rho=0.9 or 0.99

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
```

```
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

SGD+Momentum

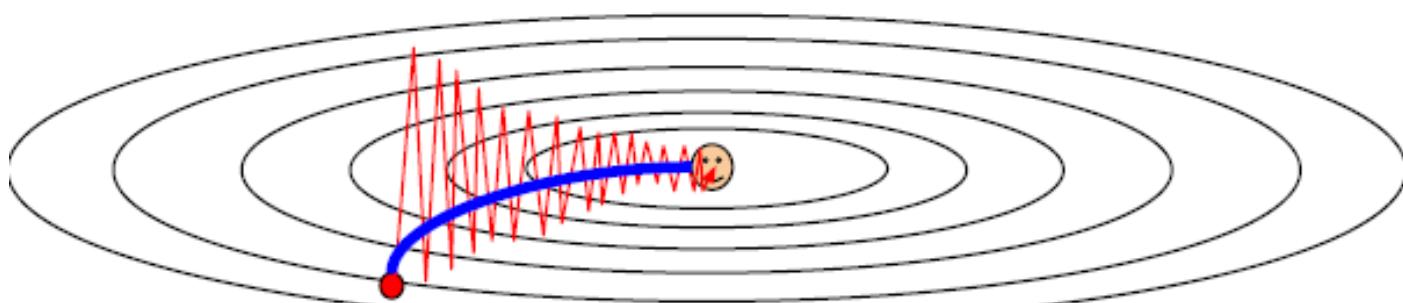
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
```

```
while True:
```

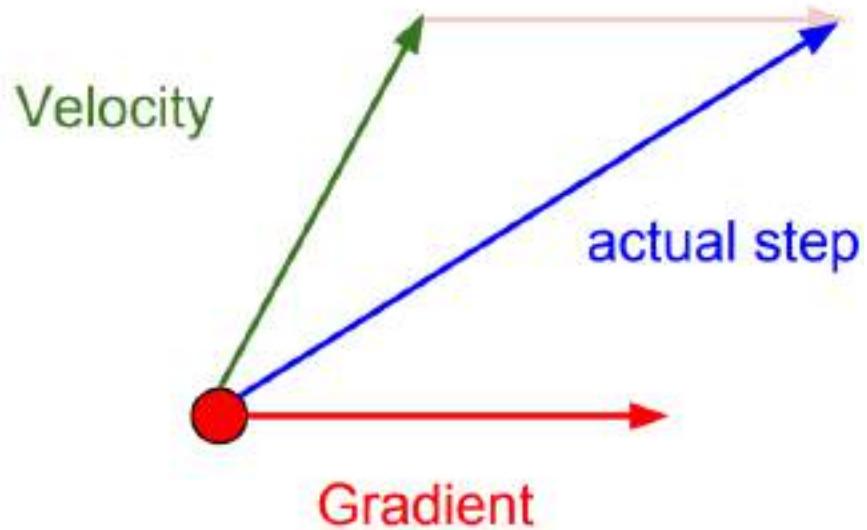
```
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```



Source: cs231n

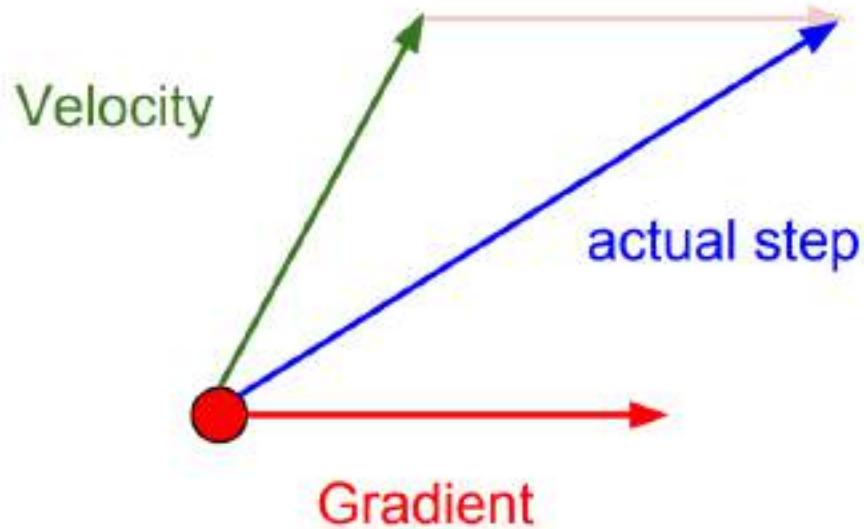
SGD + Momentum

Momentum update:

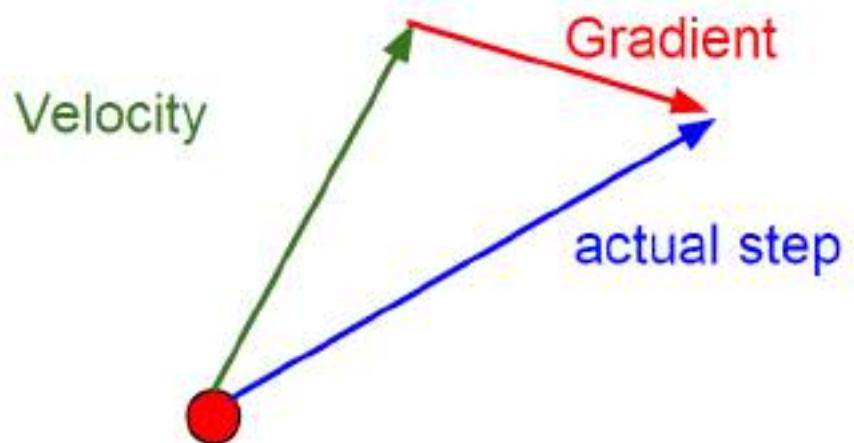


Nesterov Momentum

Momentum update:



Nesterov Momentum



Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Source: cs231n

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Source: cs231n

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1}$$

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1}$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Source: cs231n

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

What happens to the step size over long time?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

What happens to the step size over long time?

Effective learning rate diminishing problem

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



AdaGrad

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Adam

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Adam

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Sort of like RMSProp with Momentum

Adam

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Sort of like RMSProp with Momentum

Problem:

Initially, second_moment=0 and beta2=0.999

After 1st iteration, second_moment \rightarrow close to zero
So, very large step for update of x

Adam (with Bias correction)

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

AdaGrad/
RMSProp

Bias Correction

Bias correction for the fact that first and second moment estimates start at zero

Momentum

Adam (with Bias correction)

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

AdaGrad/
RMSProp

Bias Correction

Momentum

Bias correction for the fact that first and second moment estimates start at zero

Adam with **beta1 = 0.9**,
beta2 = 0.999, and **learning_rate = 1e-3 or 5e-4**
is a **great starting point** for many models!

AMSGrad

Some minibatches (rarely occur) provide large and informative gradients, exponential averaging diminishes their influence, which leads to poor convergence.

AMSGrad

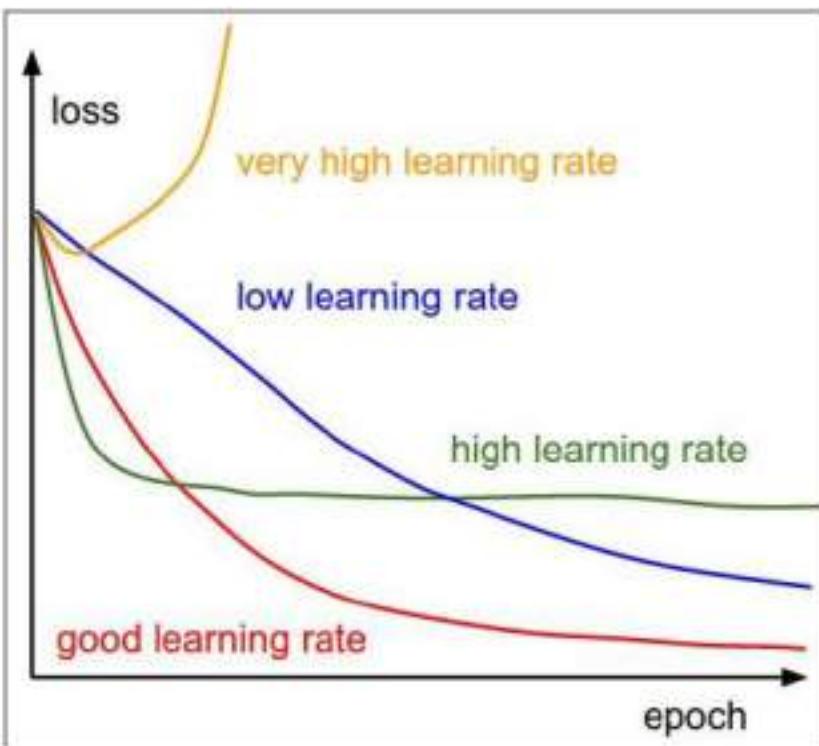
Some minibatches (rarely occur) provide large and informative gradients, exponential averaging diminishes their influence, which leads to poor convergence.

AMSGrad that uses the maximum of past squared gradients v_t

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\\hat{v}_t &= \max(\hat{v}_{t-1}, v_t) \\\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t\end{aligned}$$

Learning Rate

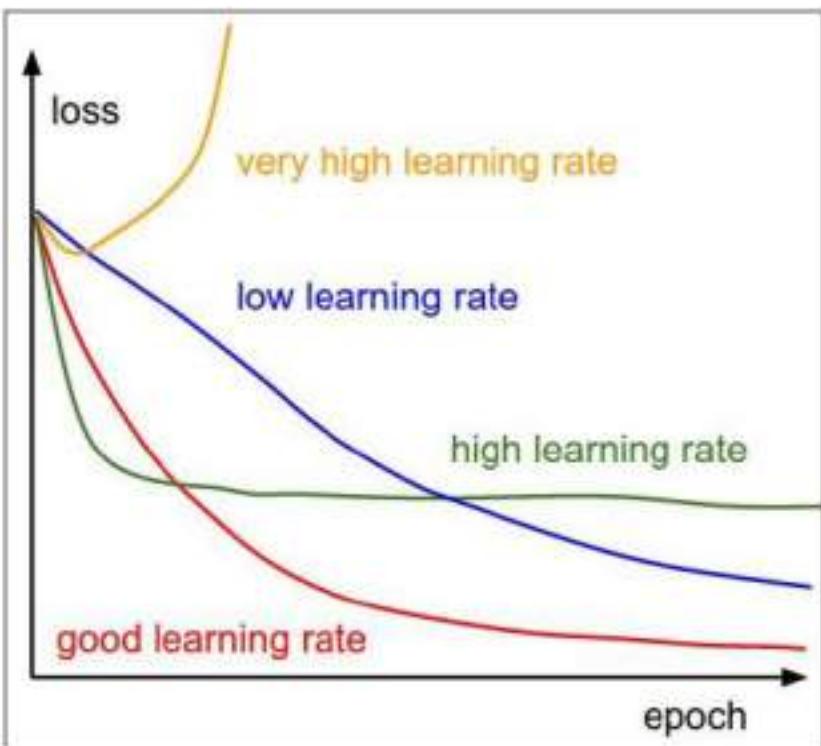
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

Learning Rate

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:

e.g. decay learning rate by half every few epochs.

exponential decay:

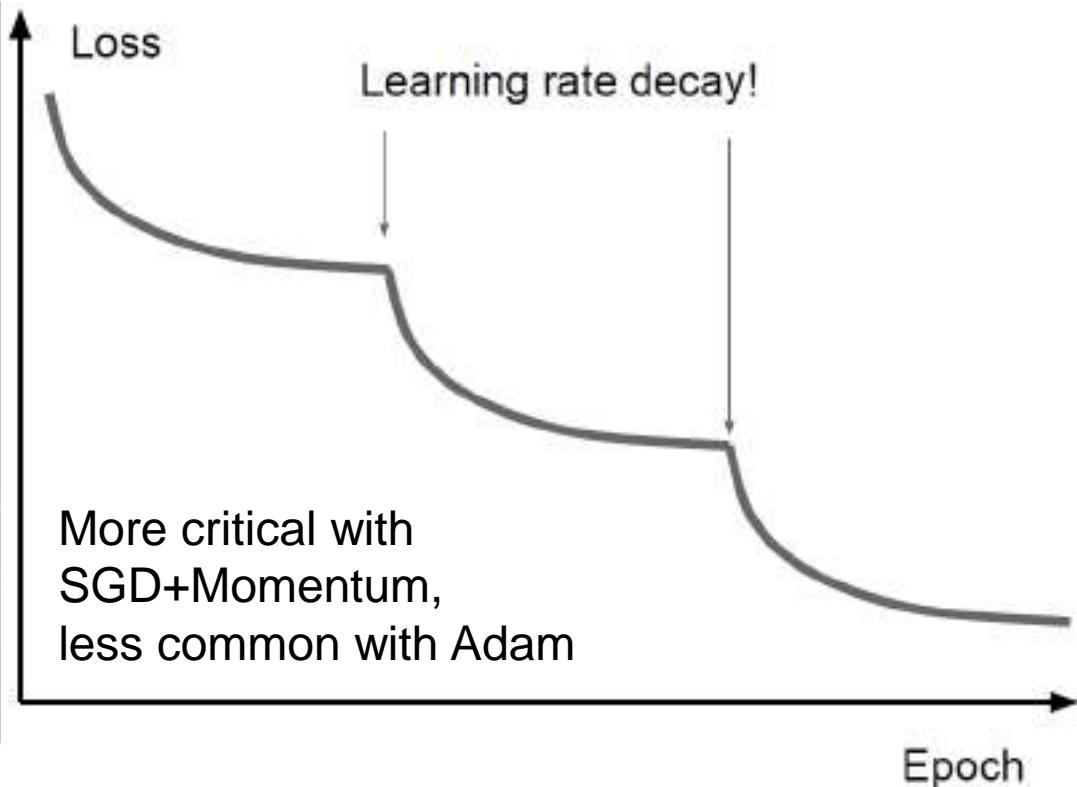
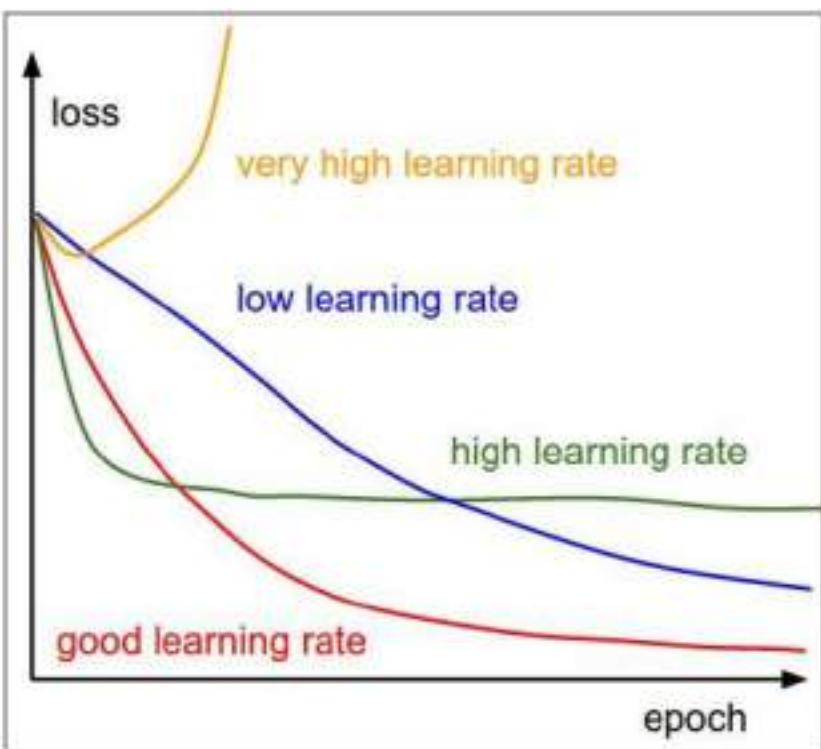
$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$

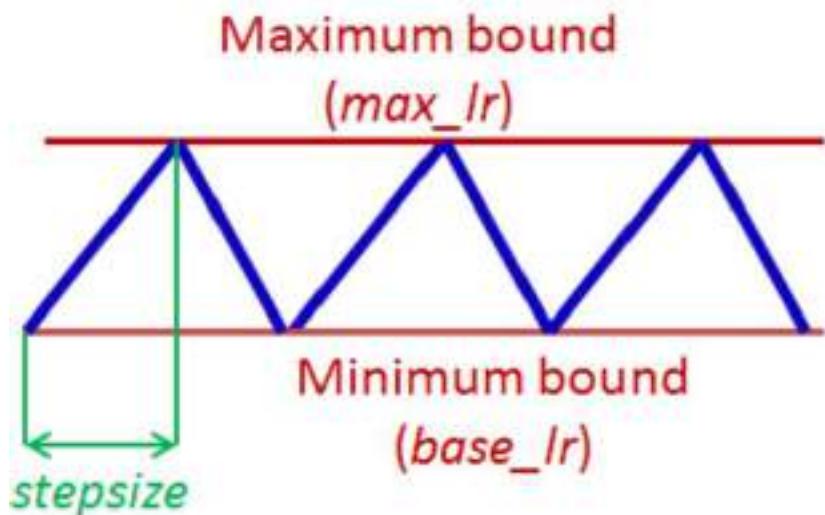
Learning Rate

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



Learning Rate

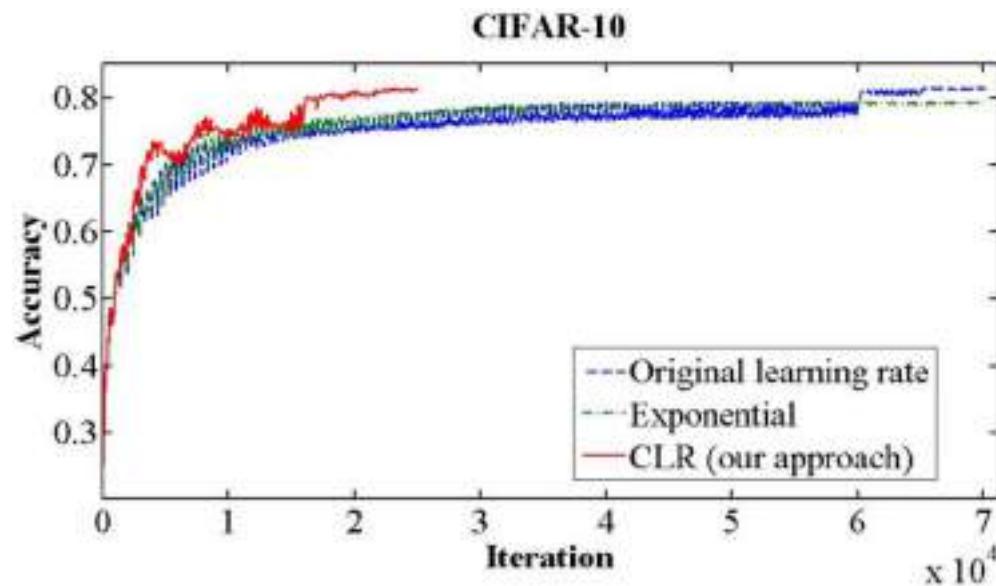
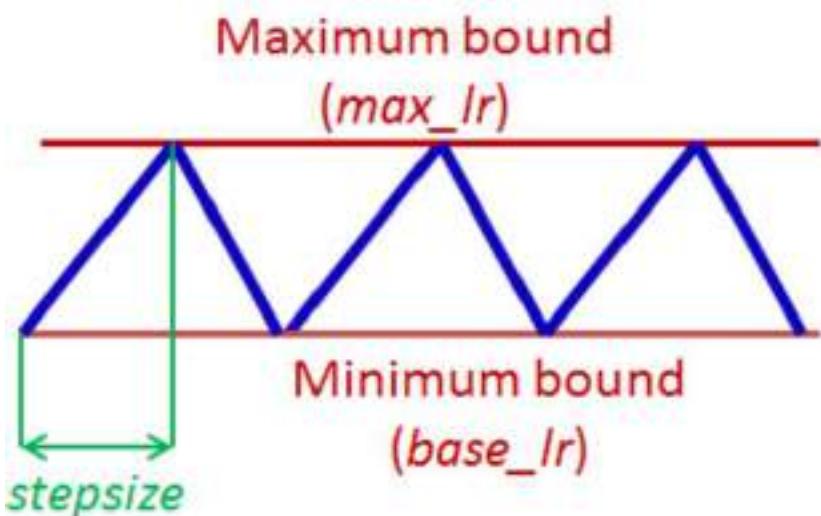
Cyclic Learning Rate



Smith, Leslie N. "Cyclical learning rates for training neural networks." WACV 2017.

Learning Rate

Cyclic Learning Rate



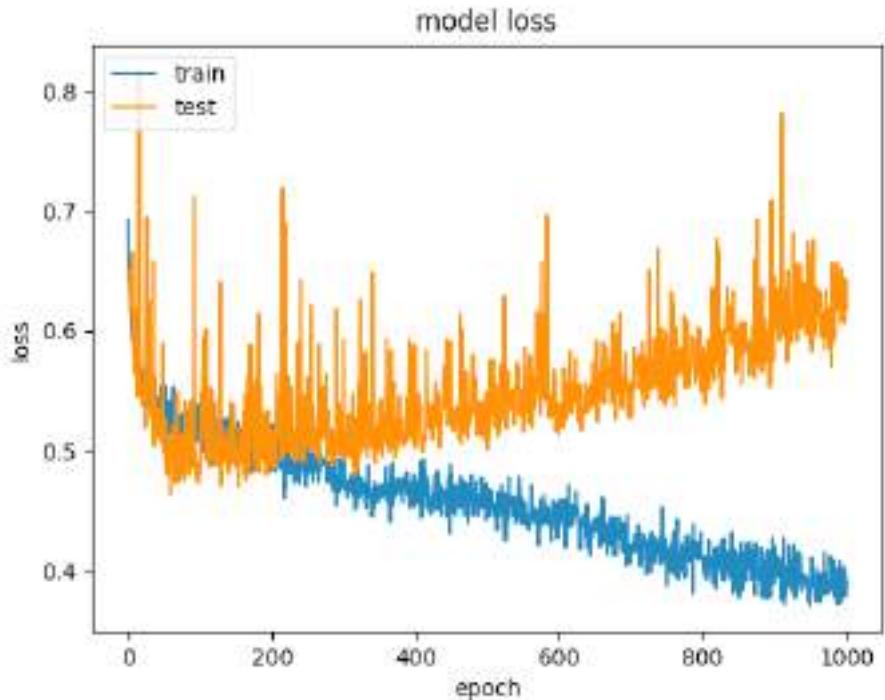
Smith, Leslie N. "Cyclical learning rates for training neural networks." WACV 2017.

Optimizer and Learning Rate

In Practice:

- **Adam** is a good default choice in most cases
- **Learning rate** with step decay is commonly used

More Optimizer: <http://ruder.io/optimizing-gradient-descent/>



Regularization

Image Source: <https://stackoverflow.com/questions/44909134/how-to-avoid-overfitting-on-a-simple-feed-forward-network/44985765>

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}}$$

Data loss: Model predictions should match training data

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \lambda R(W)$$


Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \lambda R(W)$$


Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too well* on training data

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1 \cdot x = w_2 \cdot x = 1$$

Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1 \cdot x = w_2 \cdot x = 1$$

Which W to consider?

Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1 \cdot x = w_2 \cdot x = 1$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Regularization

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1 \cdot x = w_2 \cdot x = 1$$

L2 regularization likes to
“spread out” the weights

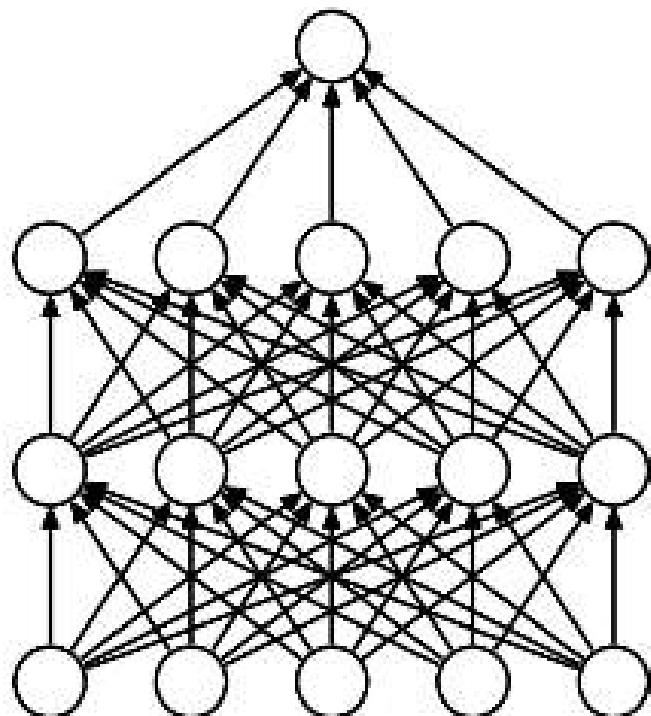
L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

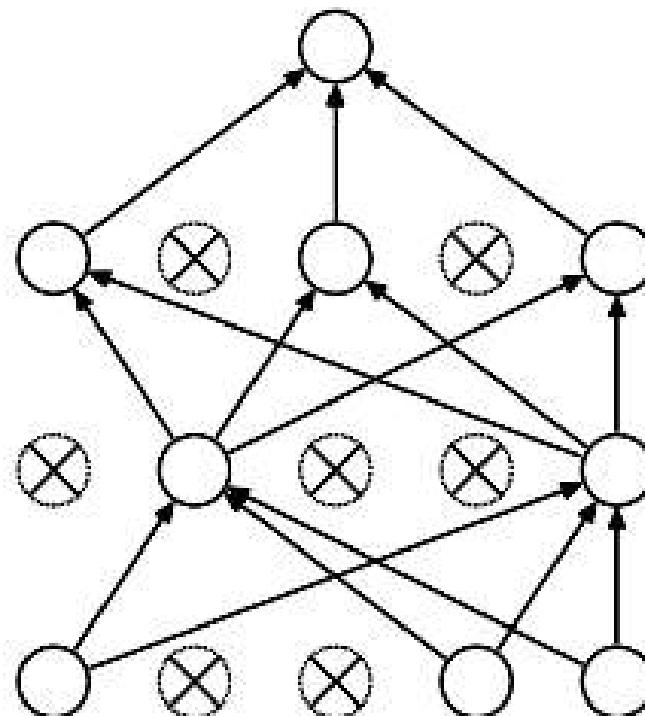
Dropout

Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



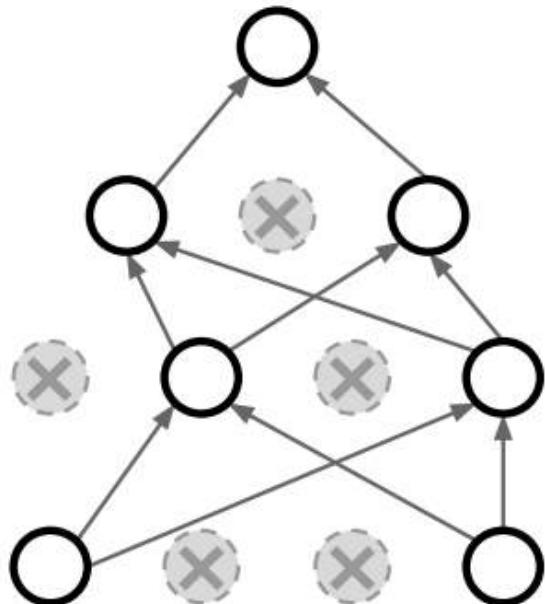
(a) Standard Neural Net



(b) After applying dropout.

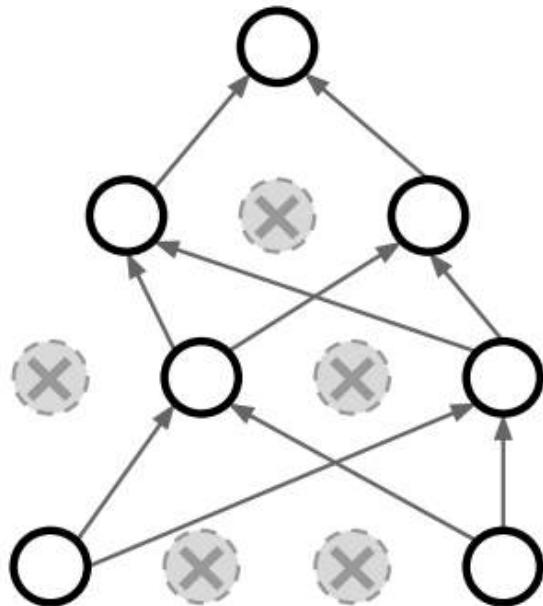
Dropout

How can this possibly be a good idea?



Dropout

How can this possibly be a good idea?



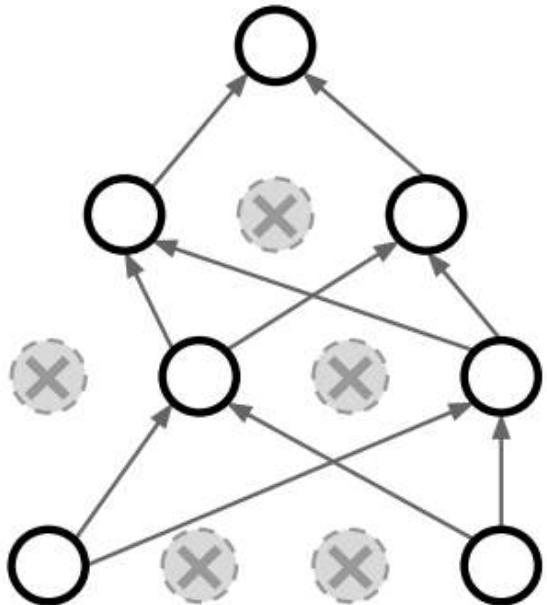
Forces the network to have a redundant representation;
Prevents co-adaptation of features



Dropout

How can this possibly be a good idea?

Dropout is training a large ensemble of models (that share parameters).



Intuition: successful conspiracies

- 50 people planning a conspiracy
- Strategy A: plan a big conspiracy involving 50 people
 - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
 - Likely to succeed!

Dropout: Test Time

```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

At test time all neurons are active always

=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

More common: “Inverted dropout”

Dropout: More common: “Inverted dropout”

We drop and scale at train time and don't do anything at test time.

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

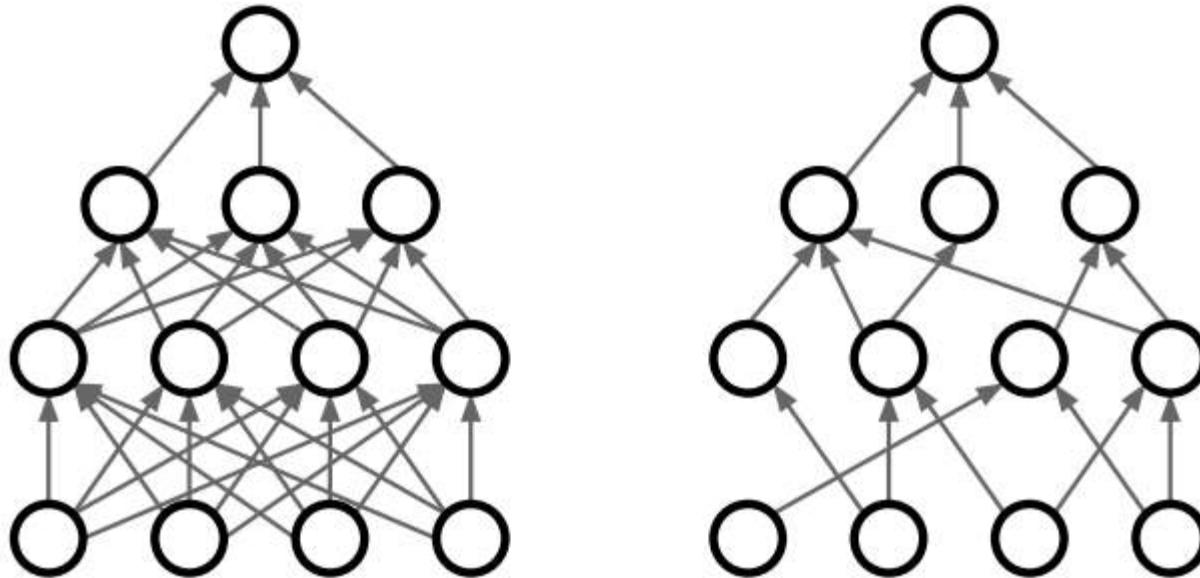
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```

test time is unchanged!



DropConnect

Dropping some connections



Wan et al, “Regularization of Neural Networks using DropConnect”, ICML 2013

Source: cs231n

Batch Normalization

Batch Normalization

“We want zero-mean unit-variance activations? lets make them so.”

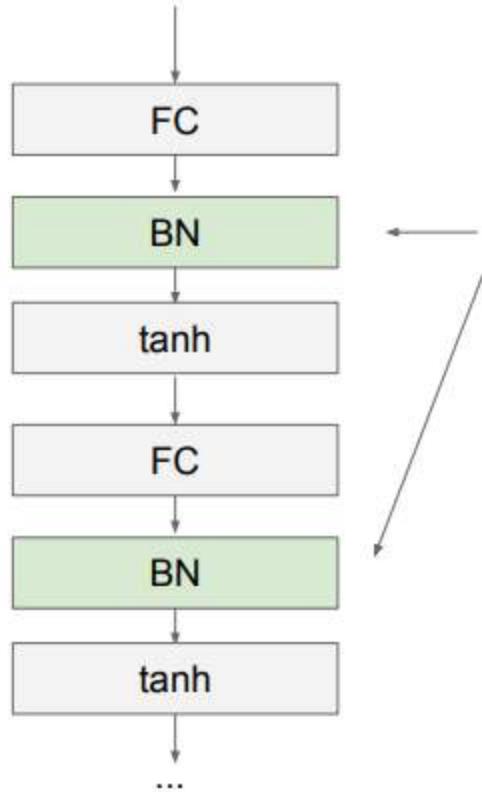
Batch Normalization

“We want zero-mean unit-variance activations? lets make them so.”

consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

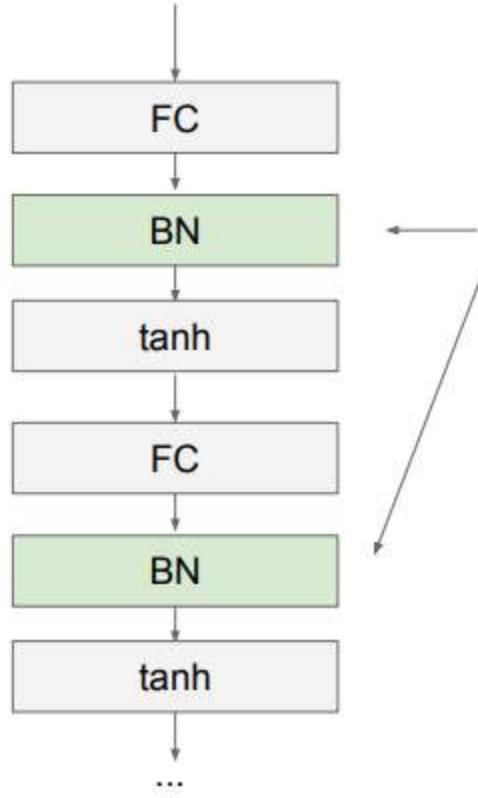
$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Batch Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Batch Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Problem:
do we necessarily want a zero-mean unit-variance input?

Batch Normalization

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Batch Normalization

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Batch Normalization

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization

Batch Normalization

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch.

Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

Batch Normalization: Recent Trends

Layer Normalization:

Ba, Kiros, and Hinton, “Layer Normalization”, arXiv 2016

Instance Normalization:

Ulyanov et al, Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, CVPR 2017

Group Normalization:

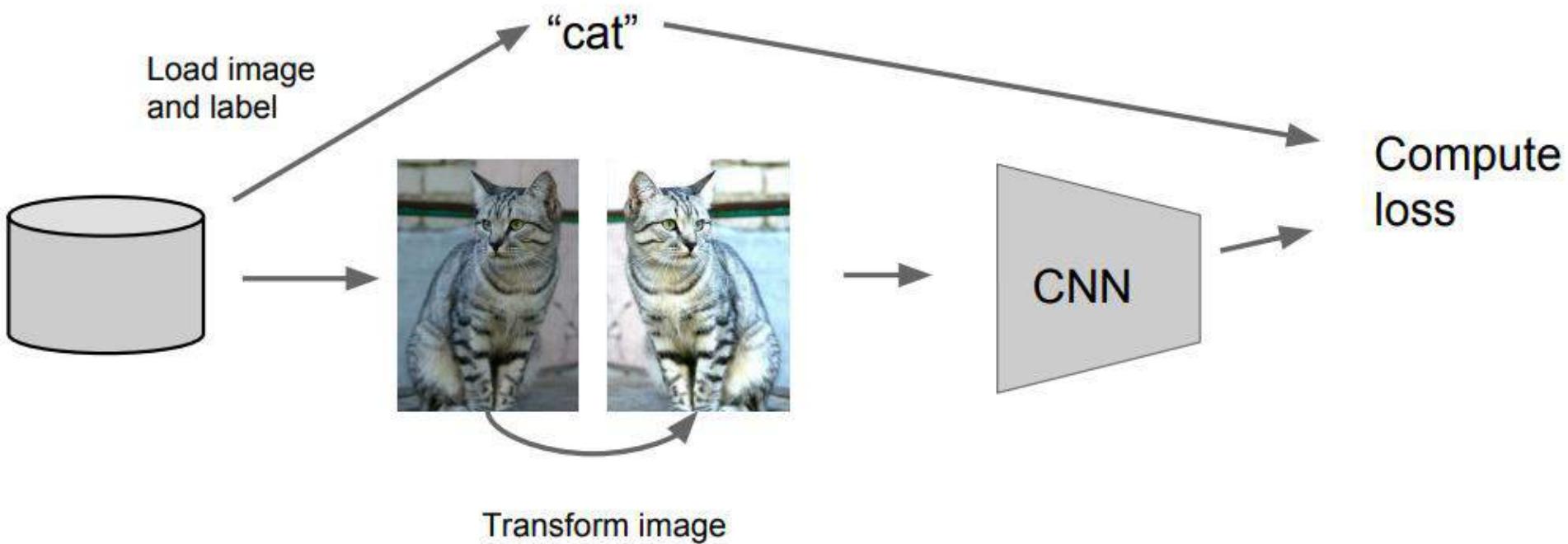
Wu and He, “Group Normalization”, arXiv 2018
(Appeared 3/22/2018)

Decorrelated Normalization:

Huang et al, “Decorrelated Batch Normalization”, arXiv 2018
(Appeared 4/23/2018)

Data Augmentation

Data Augmentation (Jittering)



Data Augmentation (Jittering)

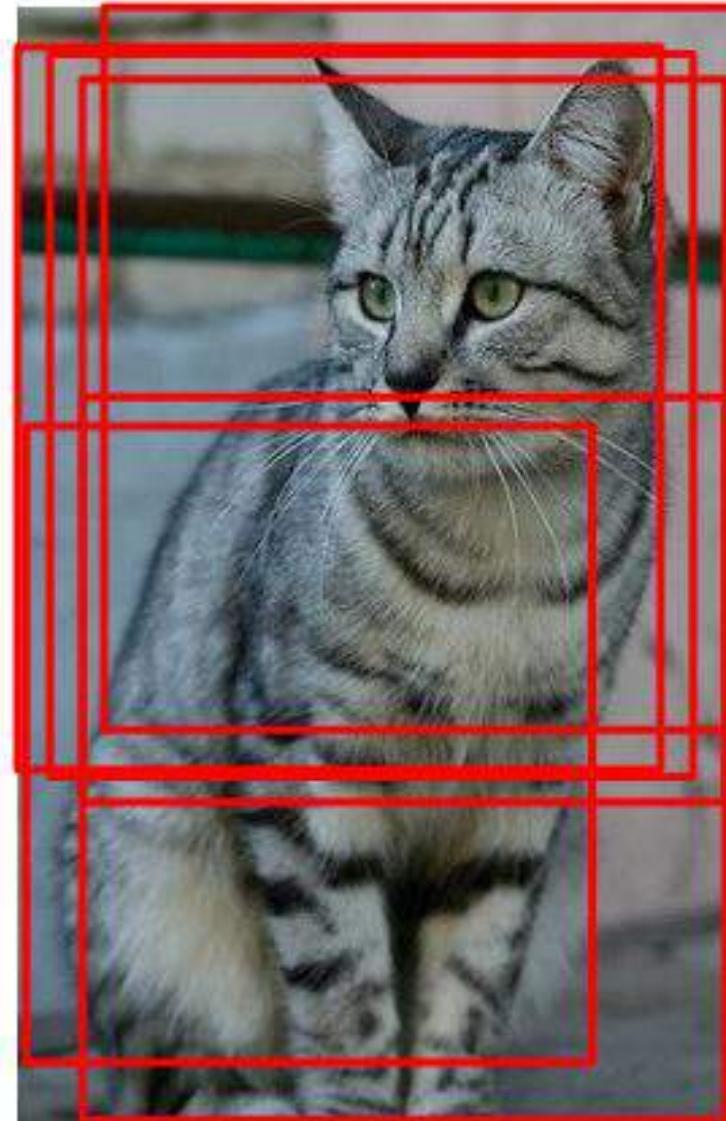
Horizontal Flips



Source: cs231n

Data Augmentation (Jittering)

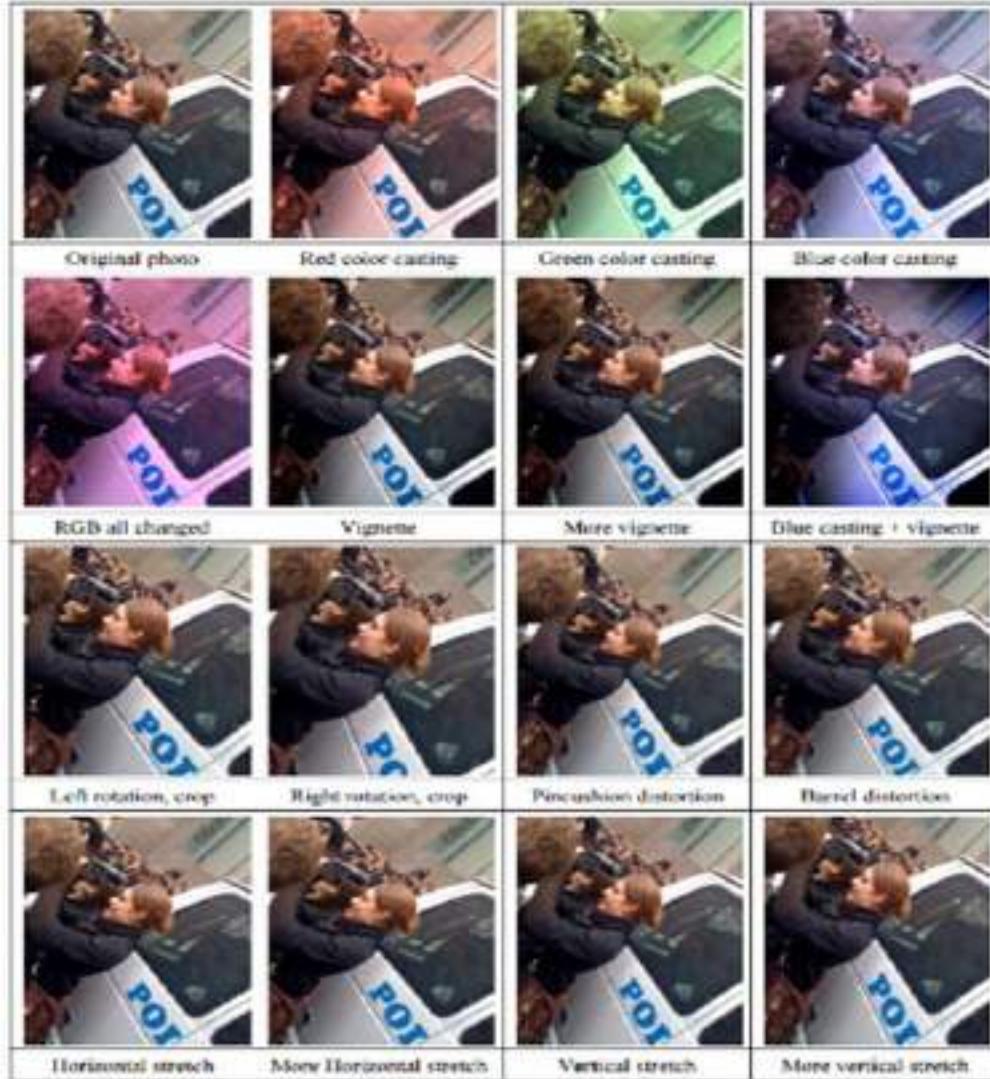
Random crops and scales



Source: cs231n

Data Augmentation (Jittering)

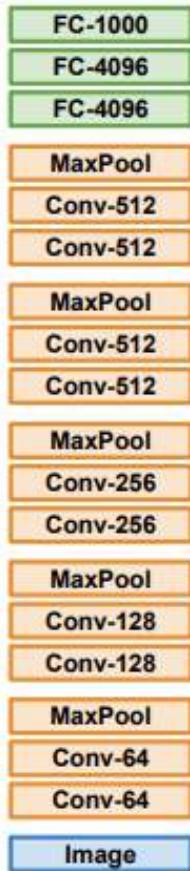
- Create *virtual* training samples
- Get creative for your problem!
 - Horizontal flip
 - Random crop
 - Color casting
 - Randomize contrast
 - Randomize brightness
 - Geometric distortion
 - Rotation
 - Photometric changes



Transfer Learning

Transfer Learning with CNNs

1. Train on Imagenet



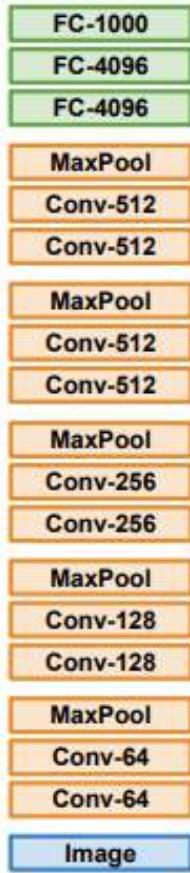
Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014

Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Source: cs231n

Transfer Learning with CNNs

1. Train on Imagenet



2. Small Dataset (C classes)



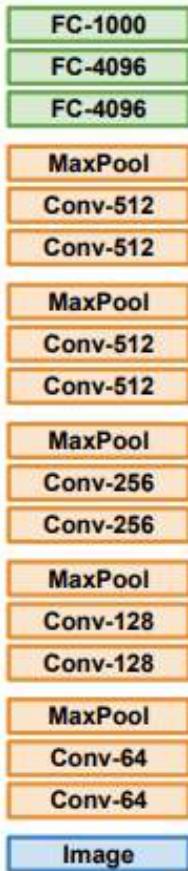
Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014

Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Source: cs231n

Transfer Learning with CNNs

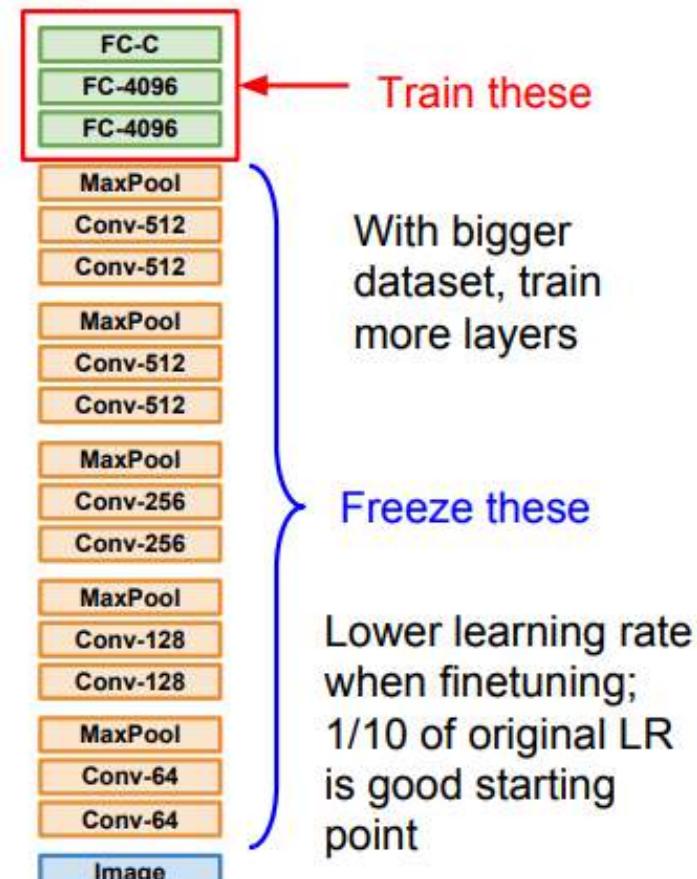
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset

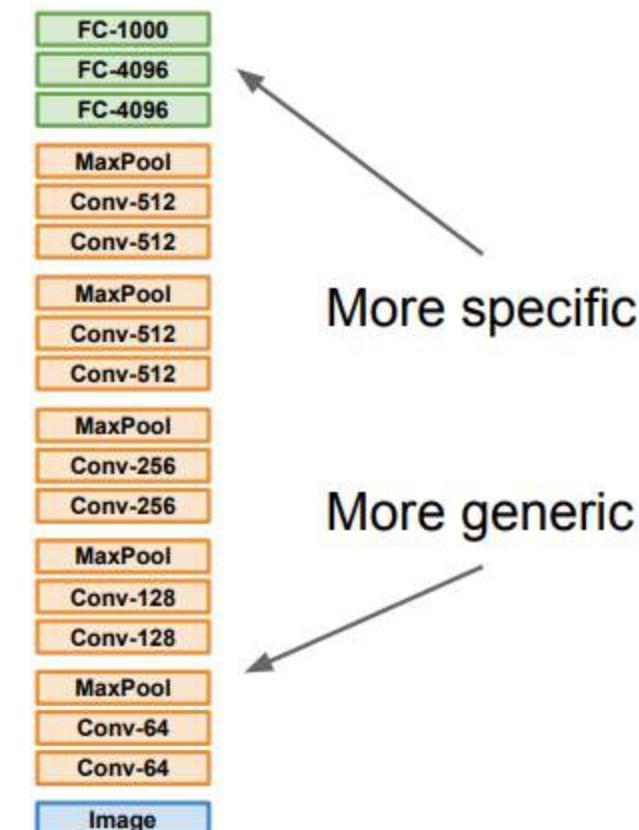


Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014

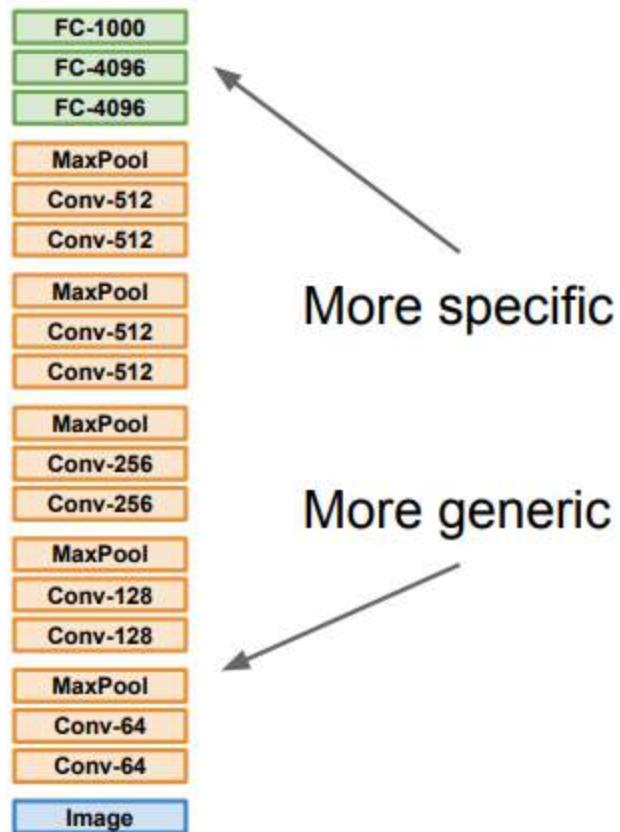
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Source: cs231n

Transfer Learning with CNNs

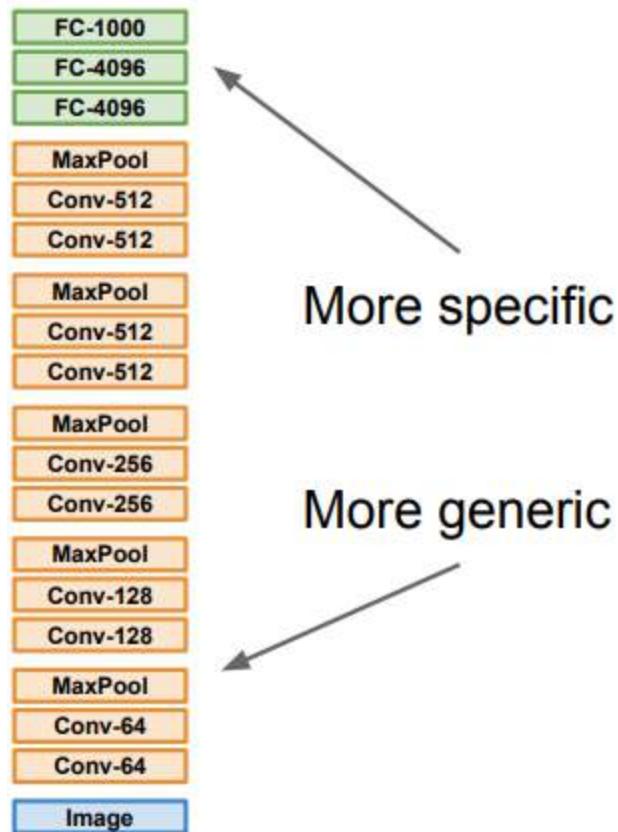


Transfer Learning with CNNs



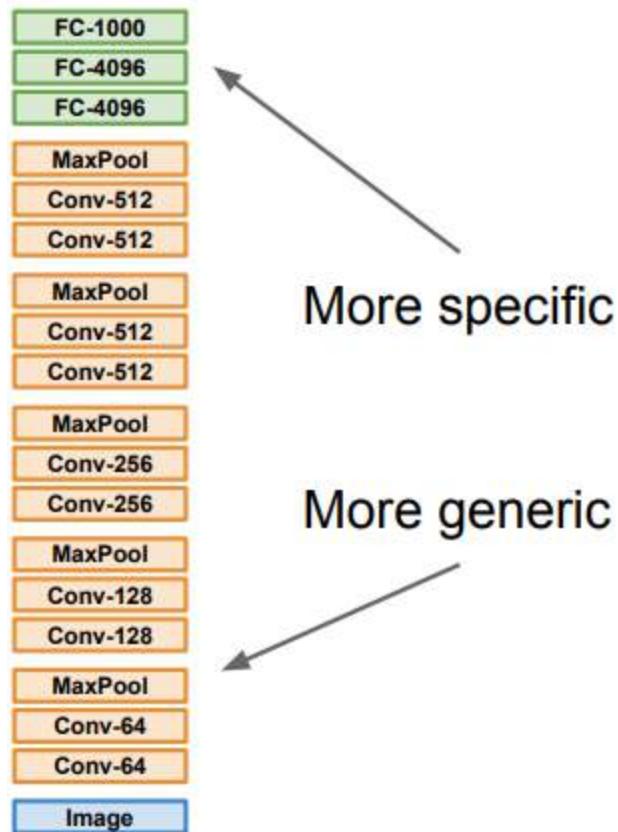
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	
quite a lot of data		

Transfer Learning with CNNs



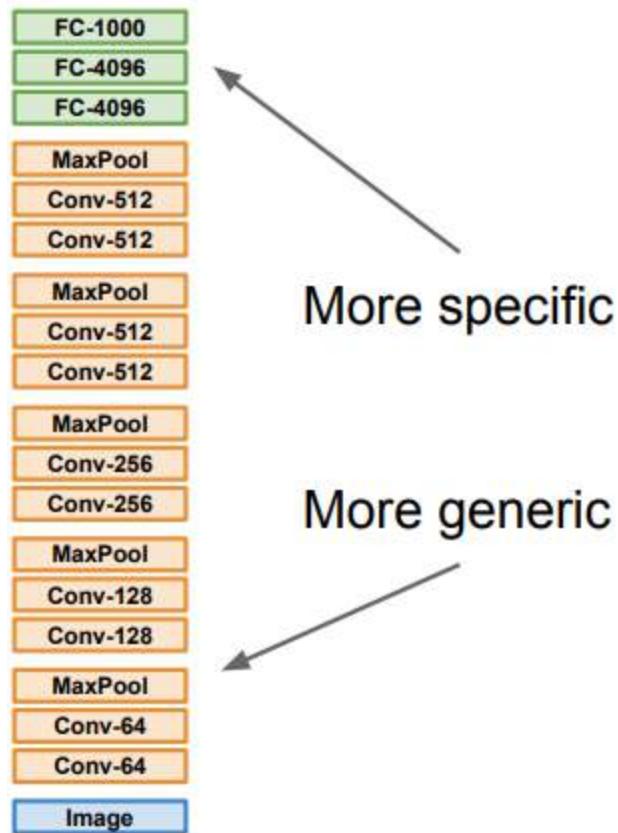
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	
quite a lot of data	Finetune a few layers	

Transfer Learning with CNNs



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

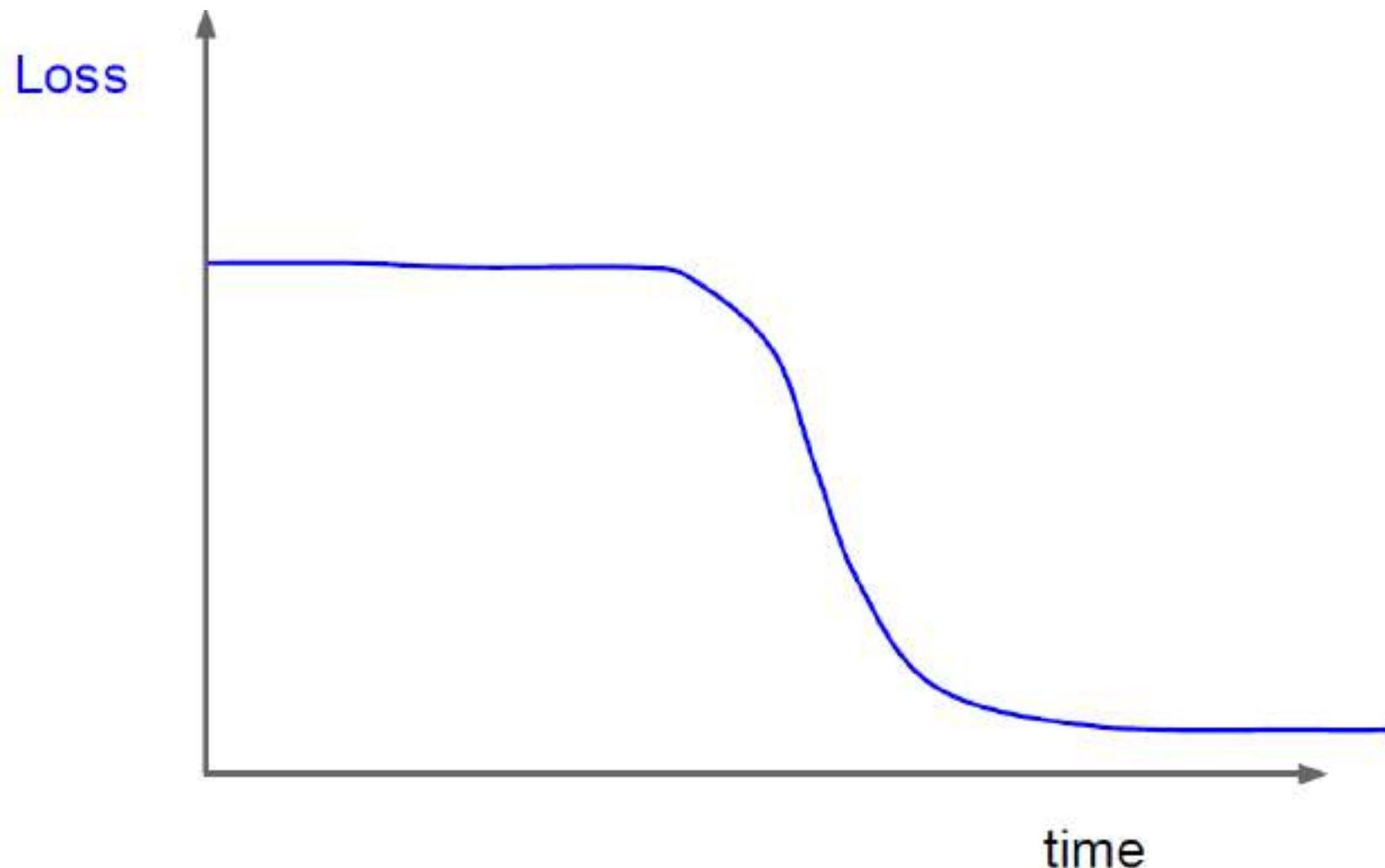
Transfer Learning with CNNs



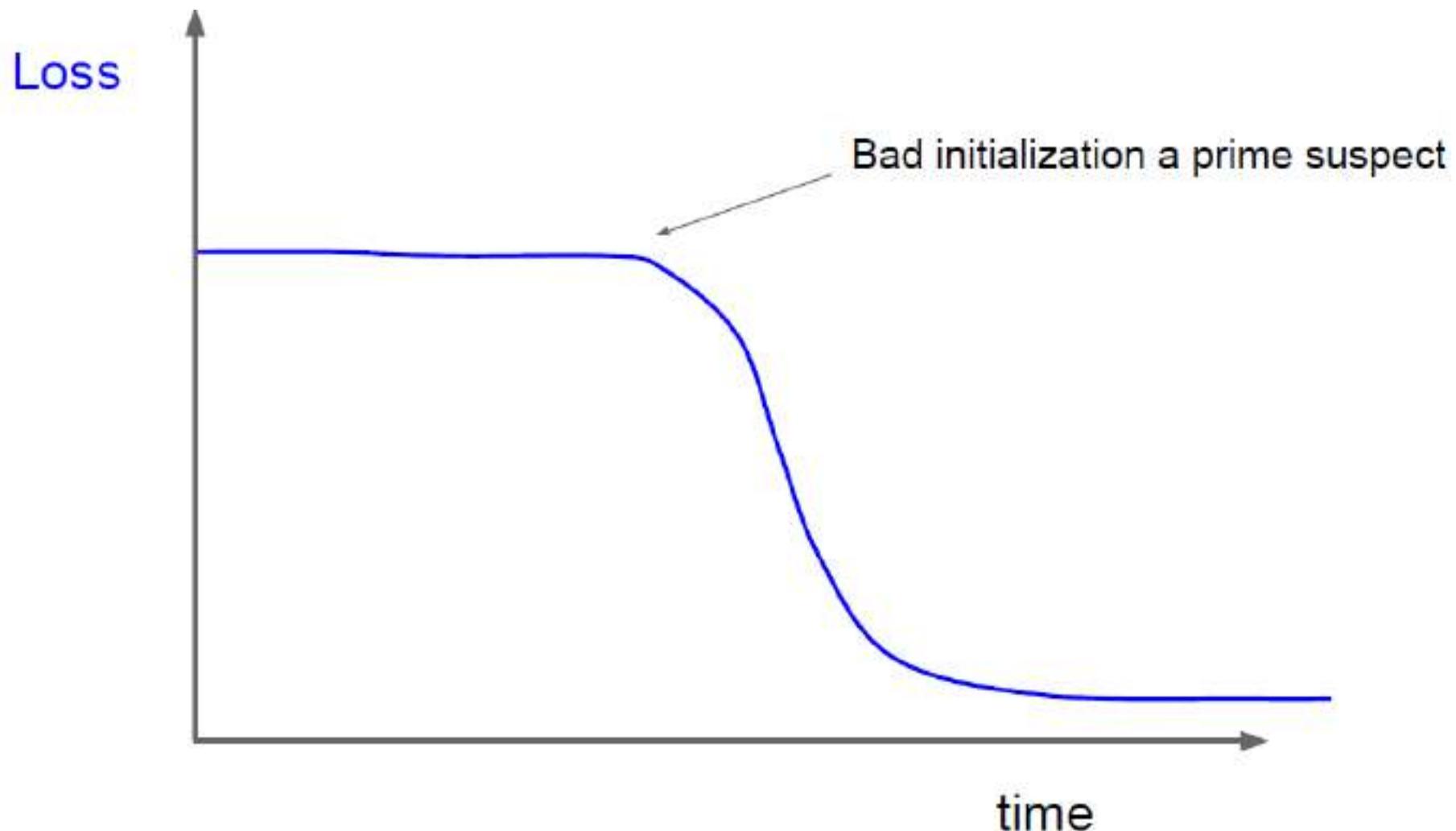
	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Monitor and Visualize the Loss Curve

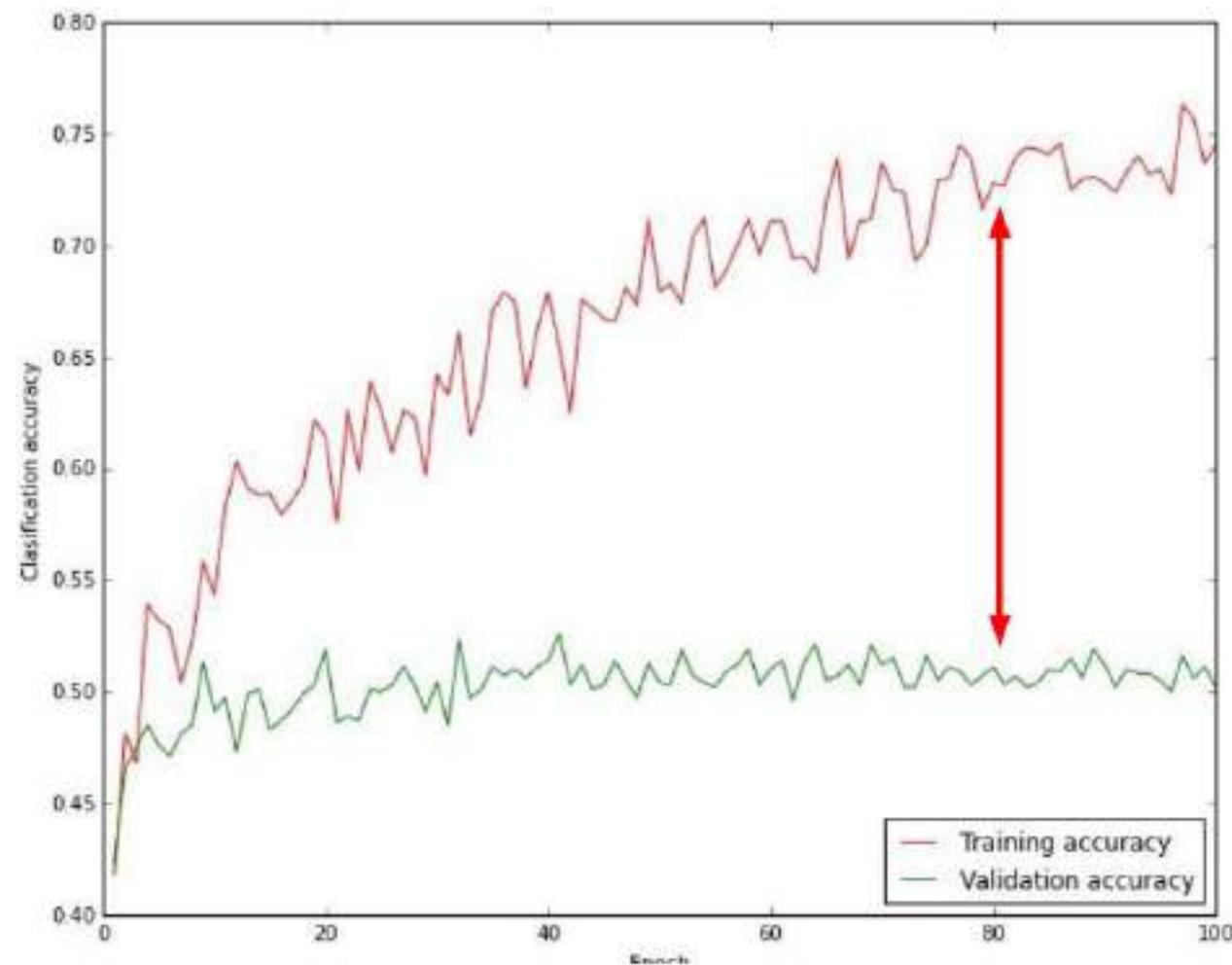
Monitor and visualize the loss curve



Monitor and visualize the loss curve



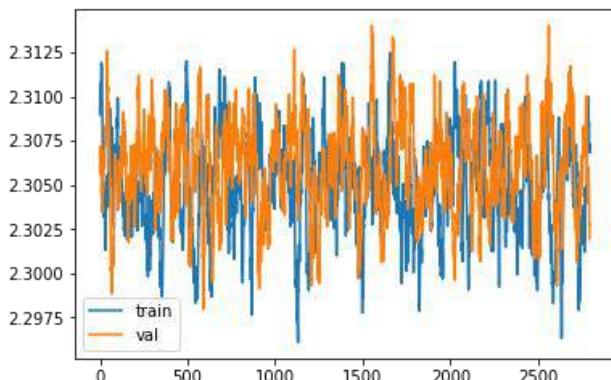
Monitor and visualize the loss curve



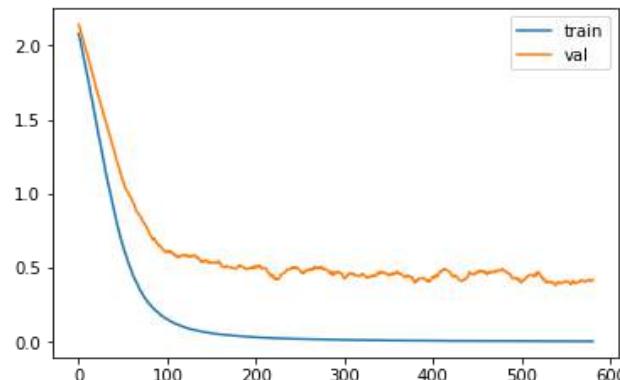
big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

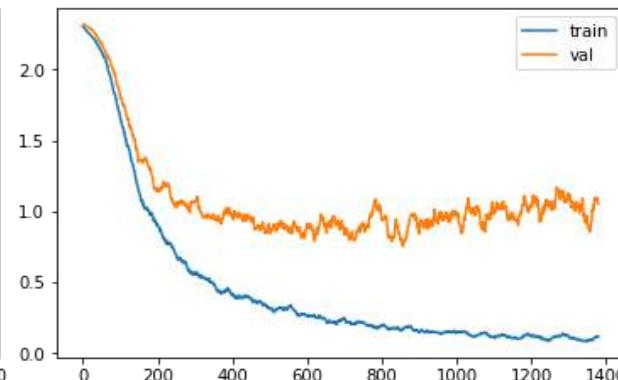
Monitor and visualize the loss curve



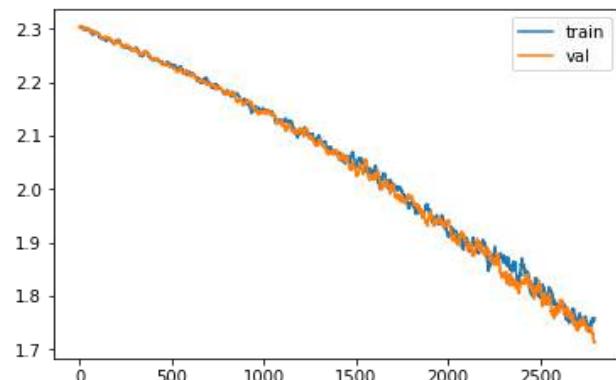
Not learning: gradients not applied to weights



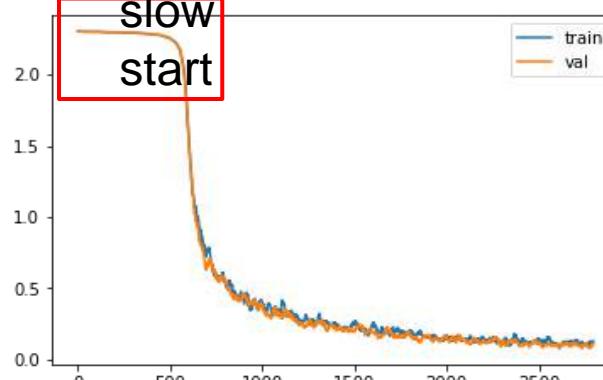
Overfit: model too large/dataset too small



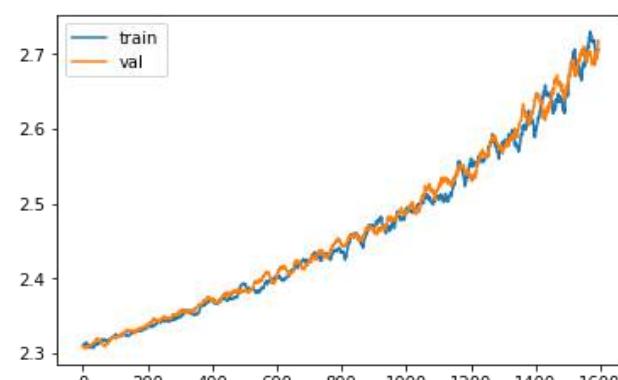
More extreme case of overfitting



Not converged yet: need longer training

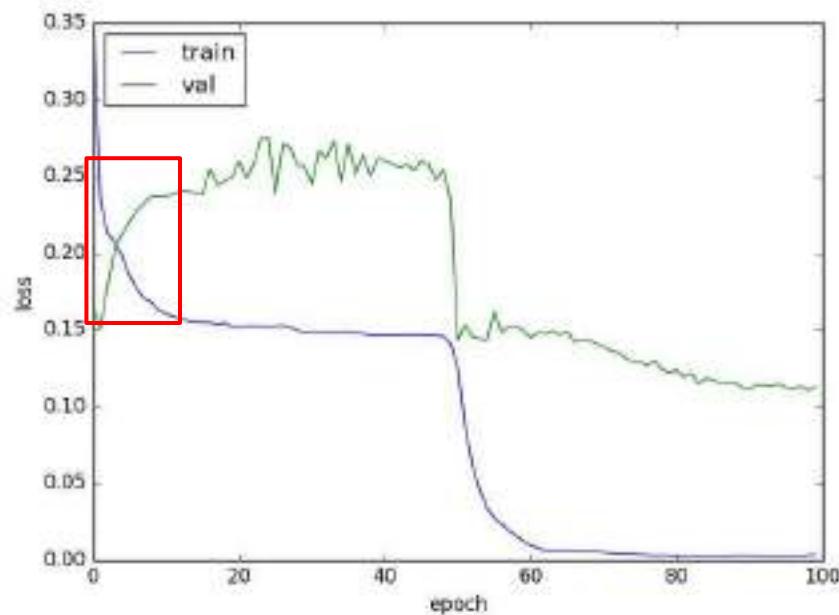


Slow start: initialization weights too small

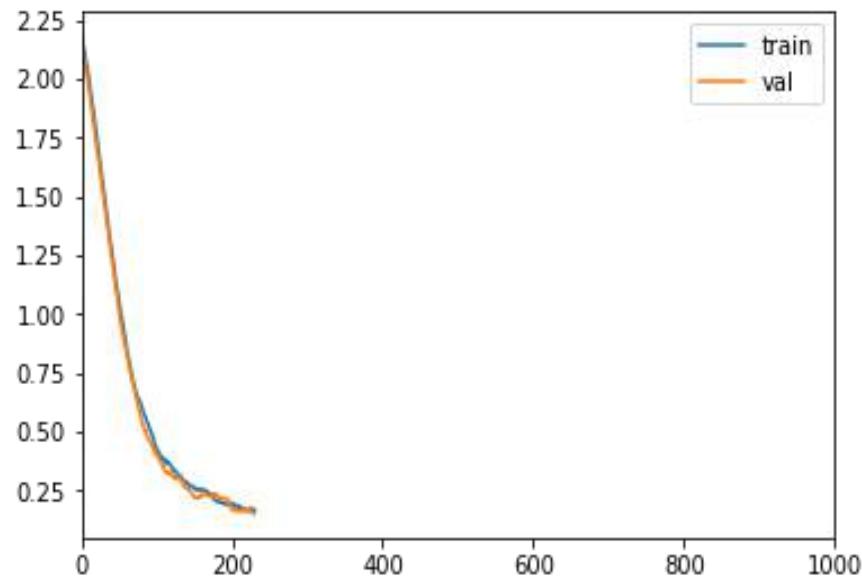


Applied the negative of gradients

Monitor and visualize the loss curve



Problem: val set too small,
statistics not meaningful



Get nans in the loss after a number of
iterations: caused by high learning rate
and numerical instability in models

Things to remember

- Training CNN
 - Adam is common (AMSGrad can be tried)
 - Learning rate: Step decay, Cyclic learning rate
 - Transfer learning, Fine tuning
- Regularization
 - L2/L1/Elastic regularization
 - Dropout and Dropconnect
 - Batch Norm
 - Data Augmentation: Flip, Crop, Contrast, etc.
- Interpreting Loss
 - Bad initialization
 - Overfitting
 - Slow/High learning rates
 - Update in wrong direction
 - Etc.

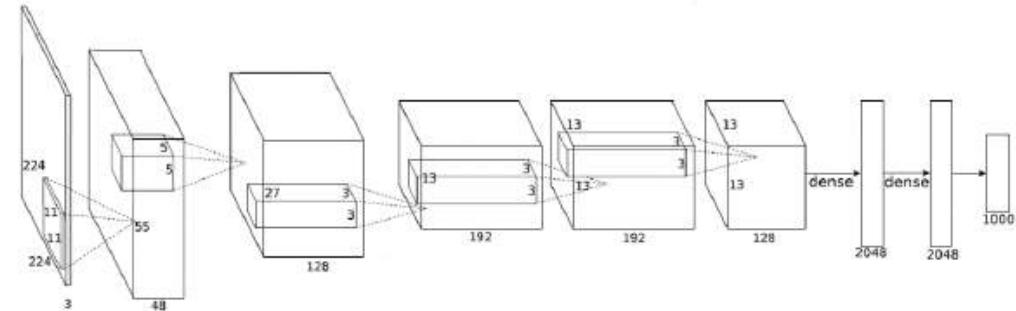
Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Antonio Torralba
 - Rob Fergus
 - Leibe
 - And many more

Next Class

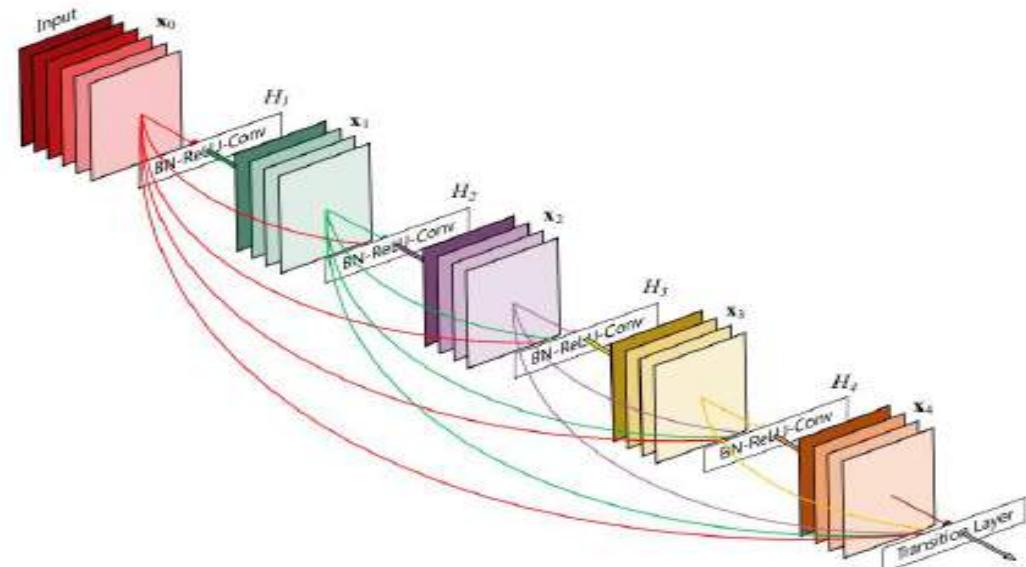
CNN Architectures: Plain Models

- LeNet
- AlexNet
- ZFNet
- VggNet
- Network in Network



CNN Architectures: DAG Models

- GoogLeNet
- ResNet
- Pre-act ResNet
- SENet
- DenseNet
- ResNetXt
- Etc.



Computer Vision

CNN Architectures

Dr. Mrinmoy Ghorai

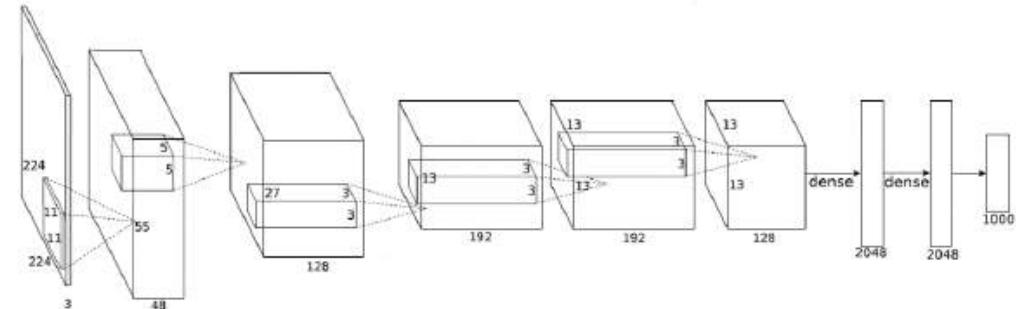
**Indian Institute of Information Technology
Sri City, Chittoor**



This Class

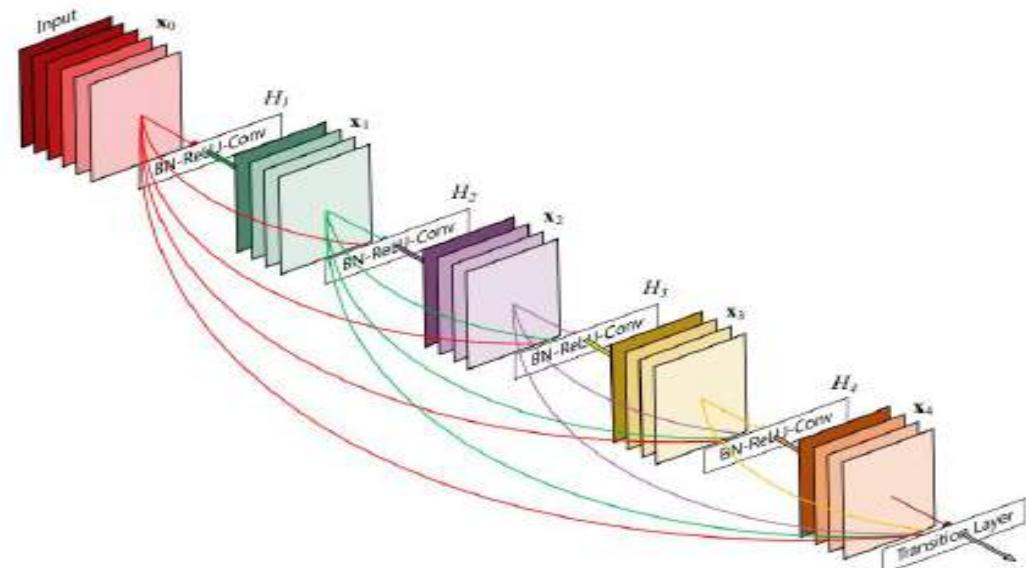
CNN Architectures: Plain Models

- LeNet
- AlexNet
- ZFNet
- VggNet
- Network in Network



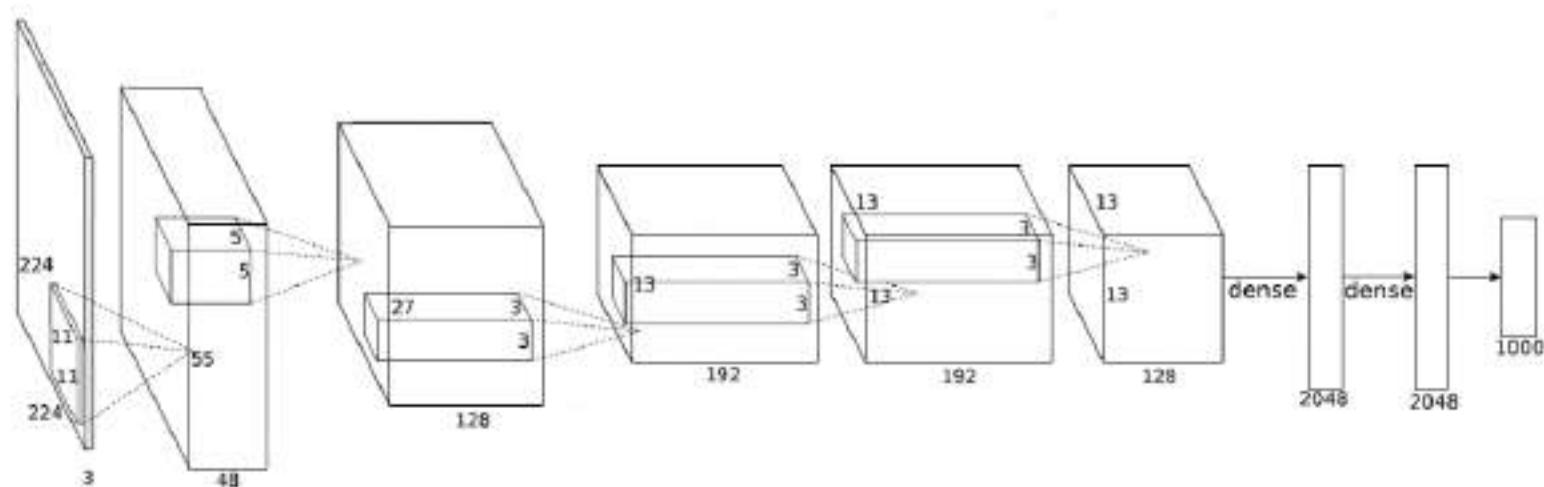
CNN Architectures: DAG Models

- GoogLeNet
- ResNet
- Pre-act ResNet
- SENet
- DenseNet
- ResNetXt
- Etc.

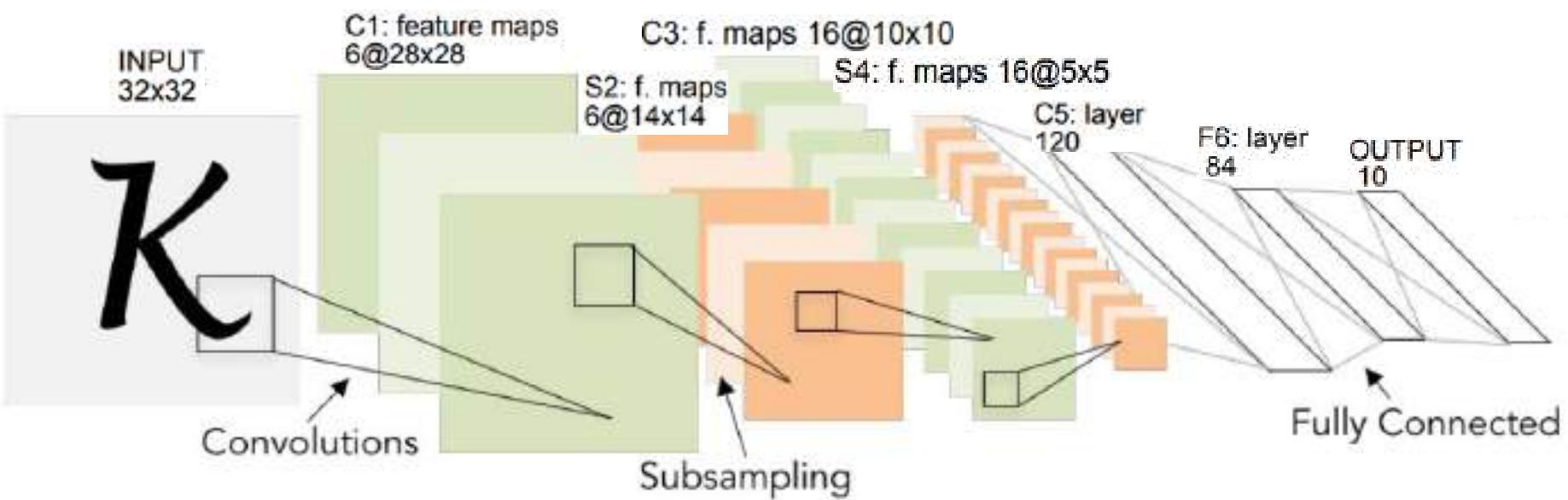


CNN Architectures: Plain Models

- LeNet
- AlexNet
- ZFNet
- VggNet
- Network in Network



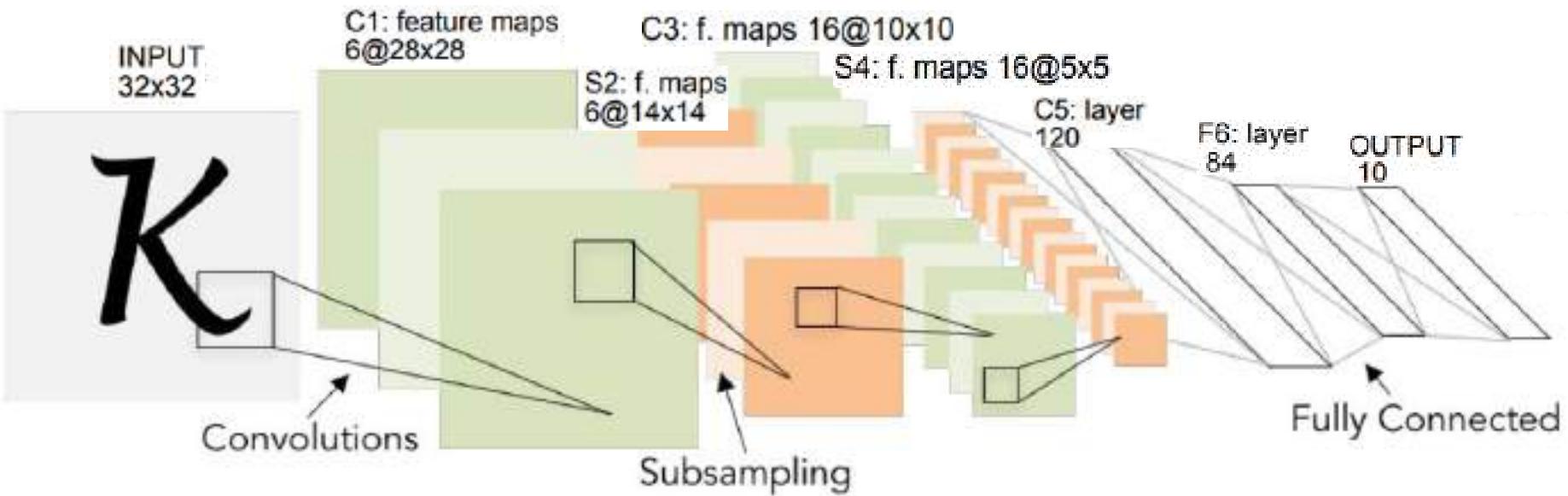
Review: LeNet-5



LeCun et al. Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 1998.

Source: cs231n

Review: LeNet-5

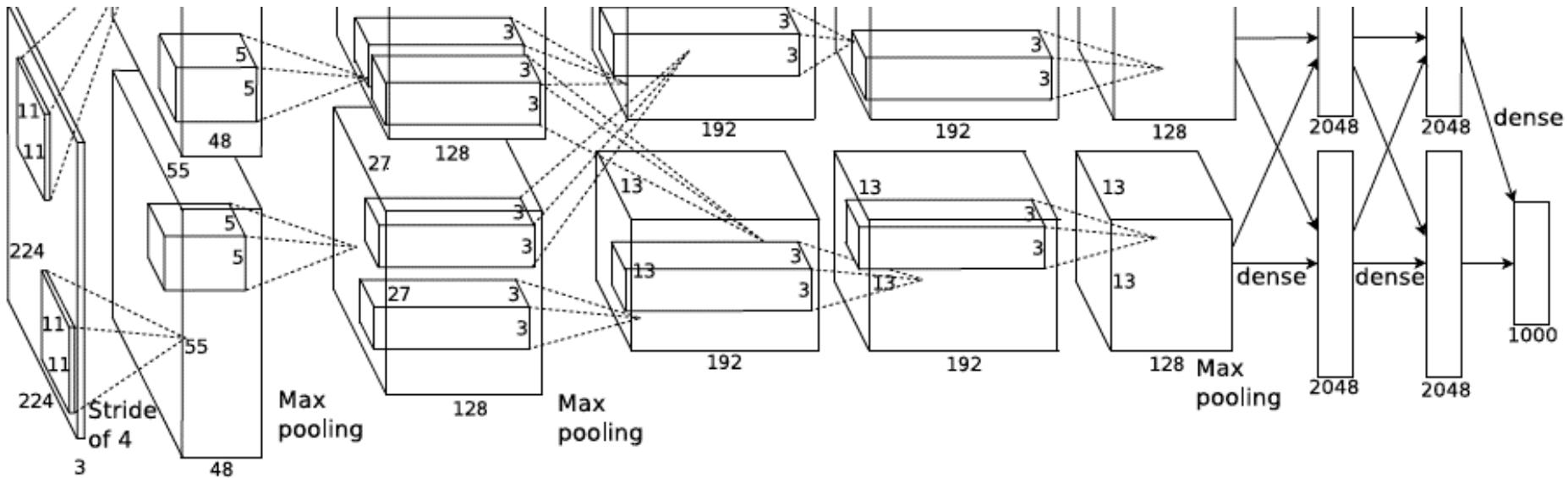


Conv filters are 5×5 , applied at stride 1

Subsampling (Pooling) layers are 2×2 applied at stride 2

i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC-FC]

AlexNet



Architecture:

CONV1

CONV2

CONV3

CONV4

CONV5

FC6

FC7

FC8

MAX POOL1

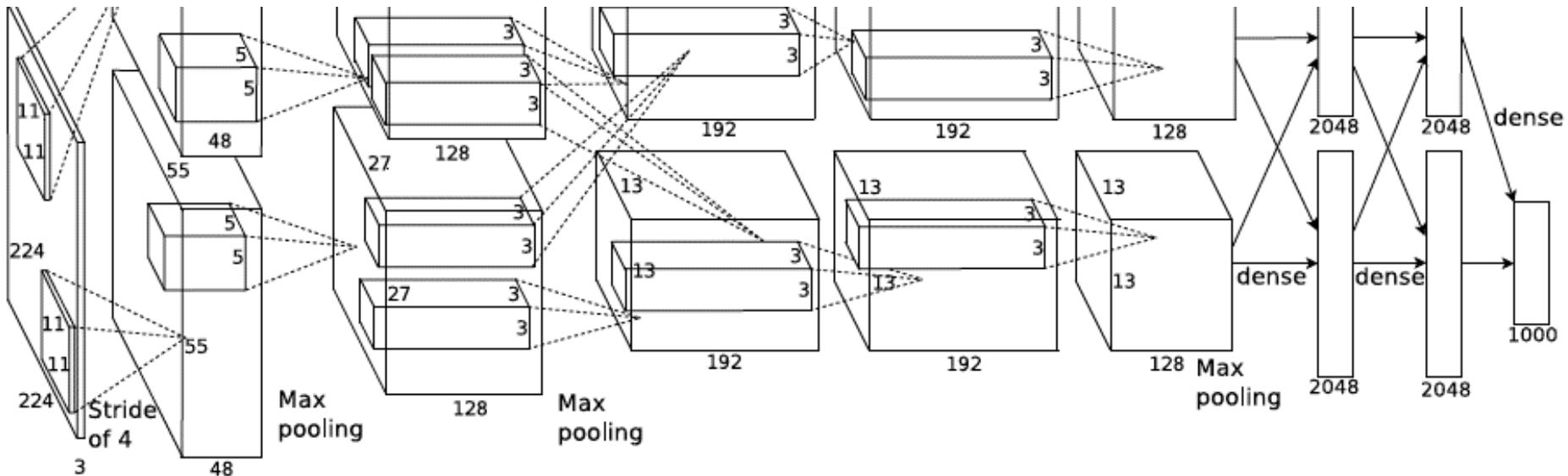
MAX POOL2

Max POOL3

NORM1(Local Response Normalization)

NORM2(Local Response Normalization)

AlexNet



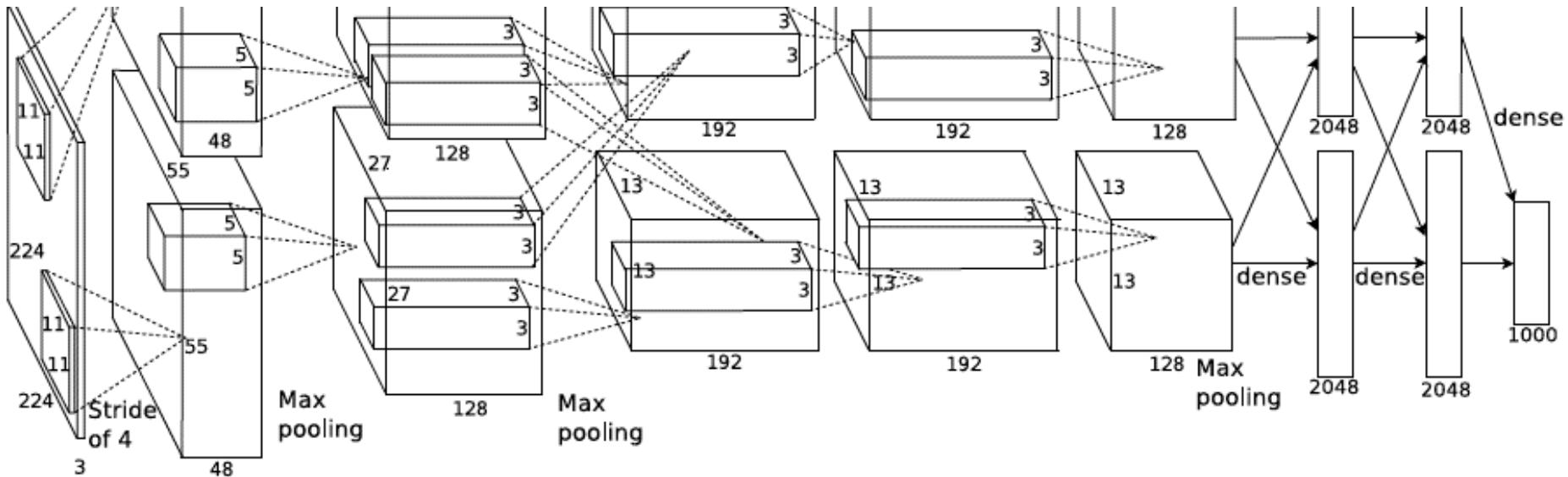
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: $(227-11)/4+1 = 55$

AlexNet



Input: 227x227x3 images

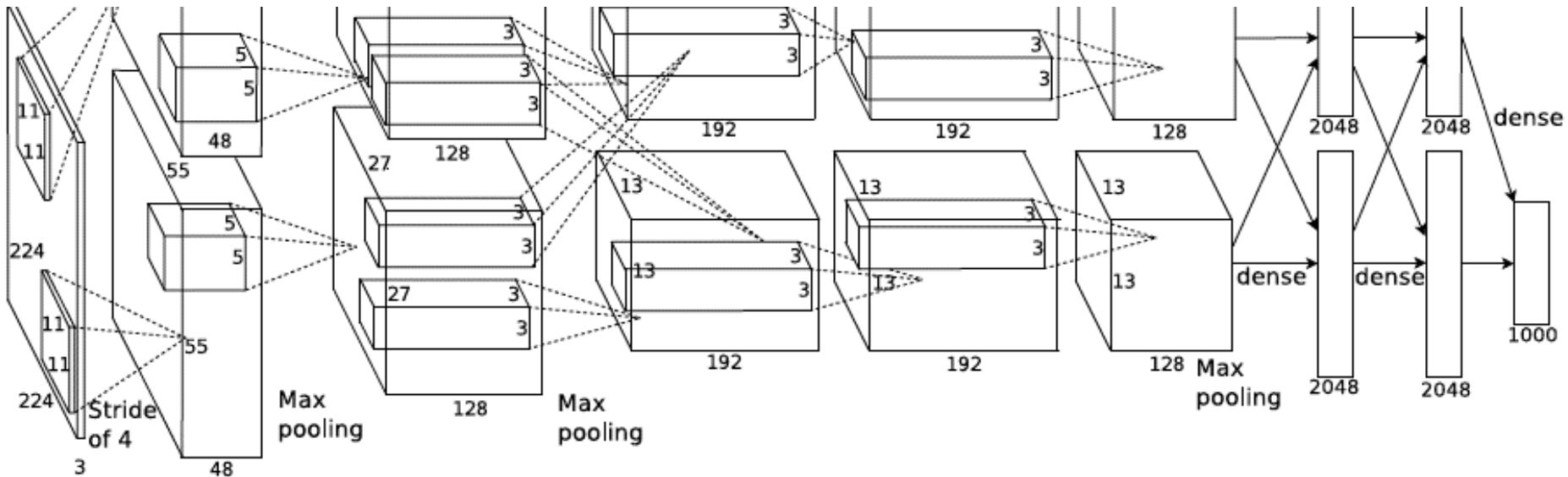
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

AlexNet



Input: 227x227x3 images

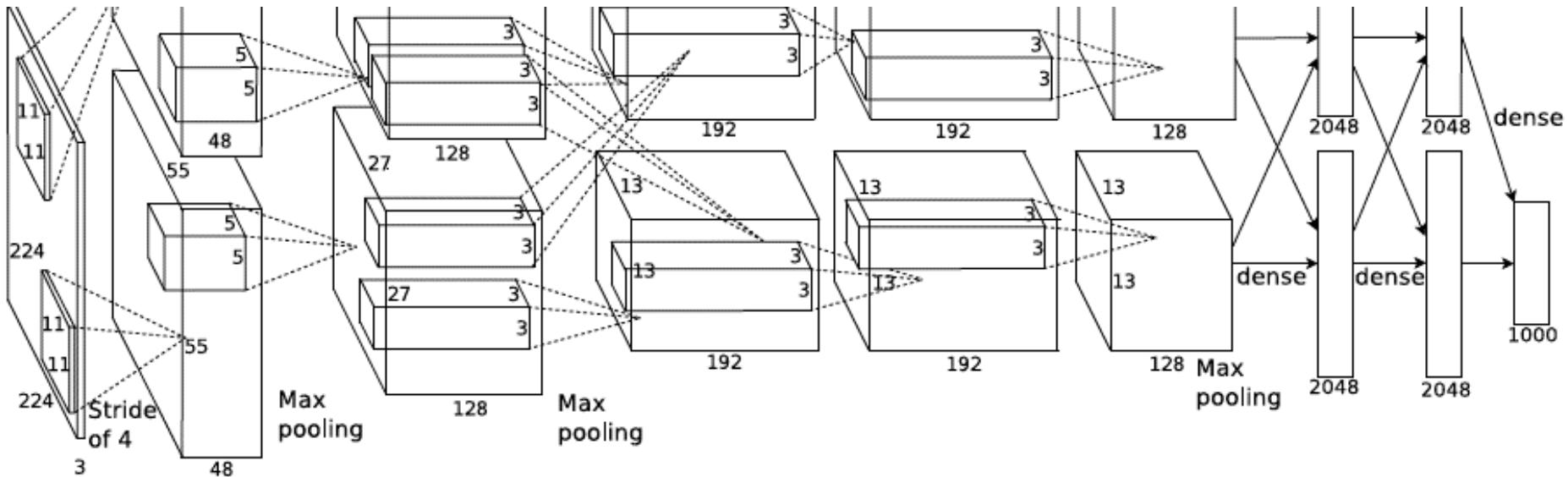
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Parameters: $(11 \times 11 \times 3) \times 96 = 35K$

AlexNet



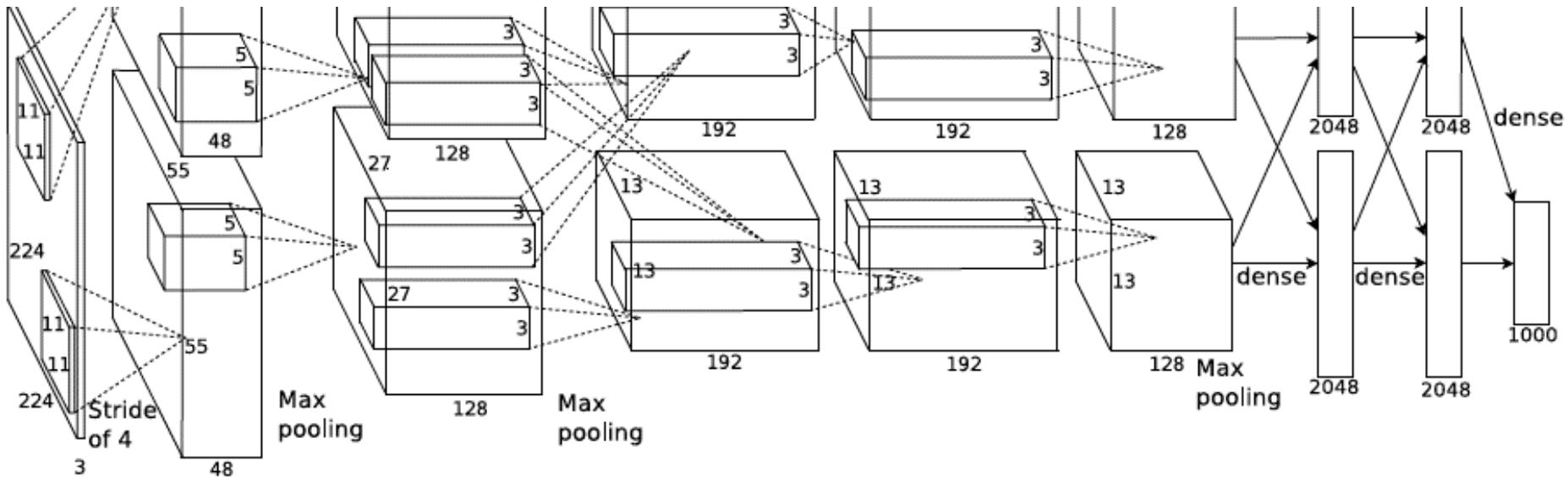
Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: $(55-3)/2+1 = 27$

AlexNet



Input: 227x227x3 images

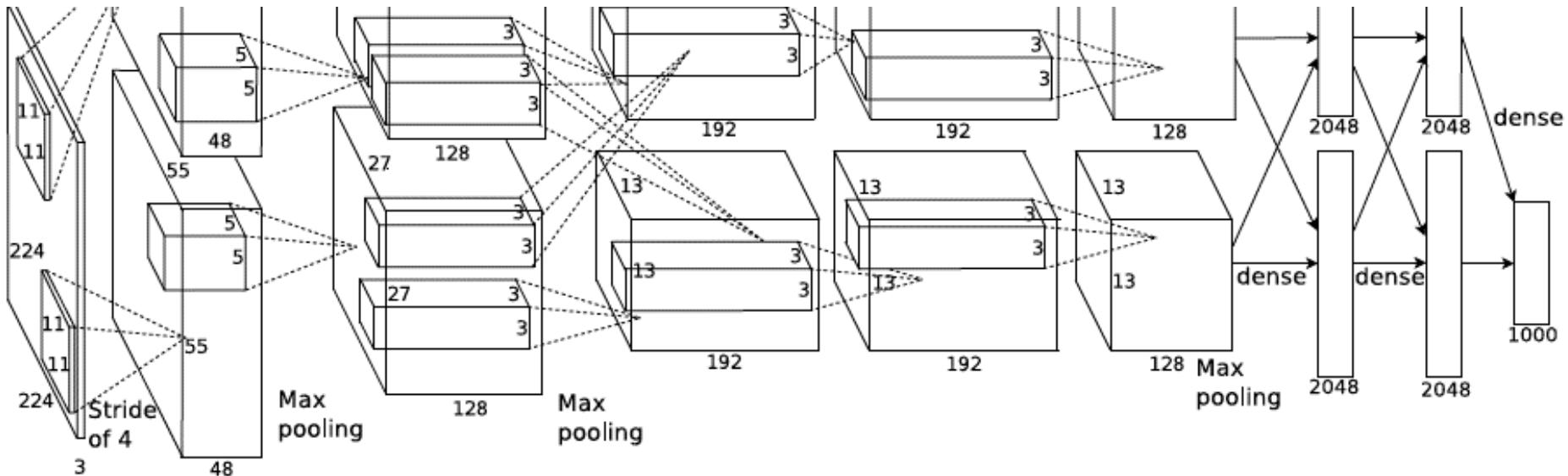
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume [27x27x96]

Q: what is the number of parameters in this layer?

AlexNet



Input: 227x227x3 images

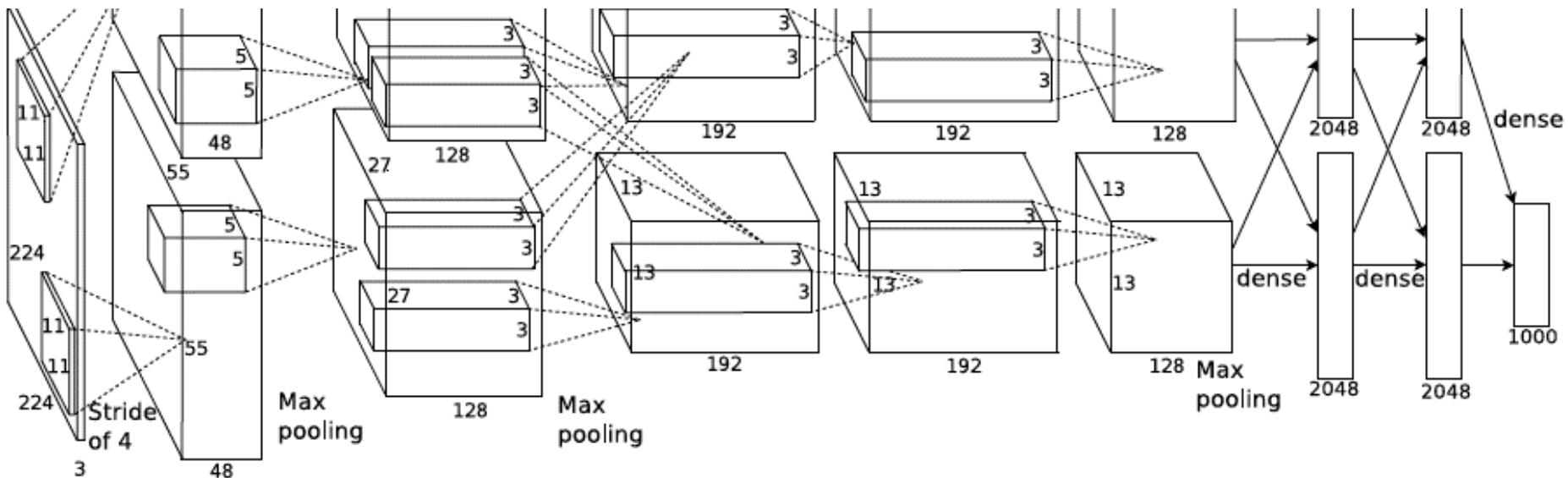
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume [27x27x96]

Parameters: 0!

AlexNet



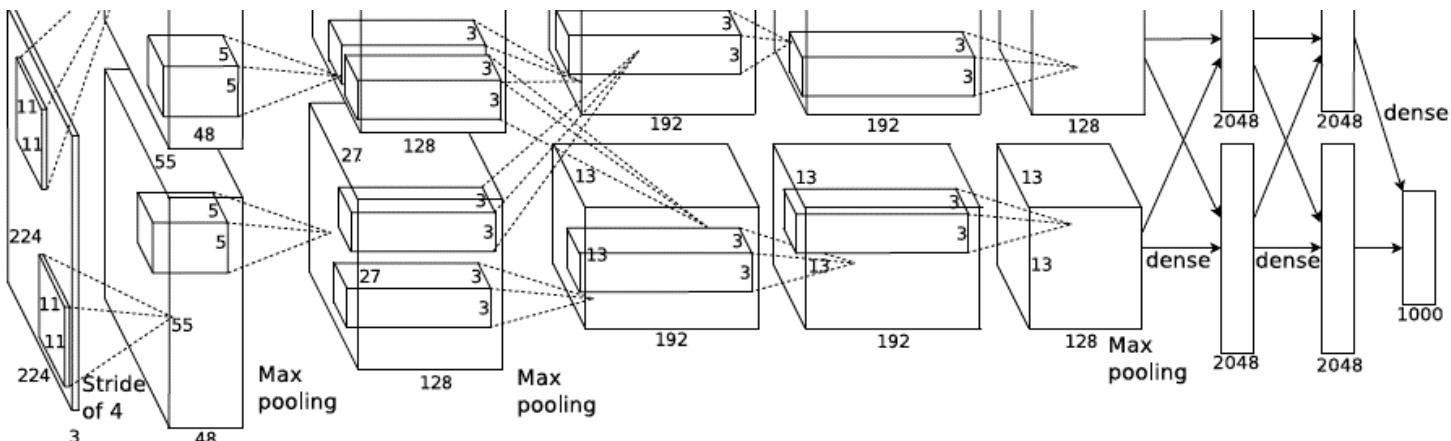
Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

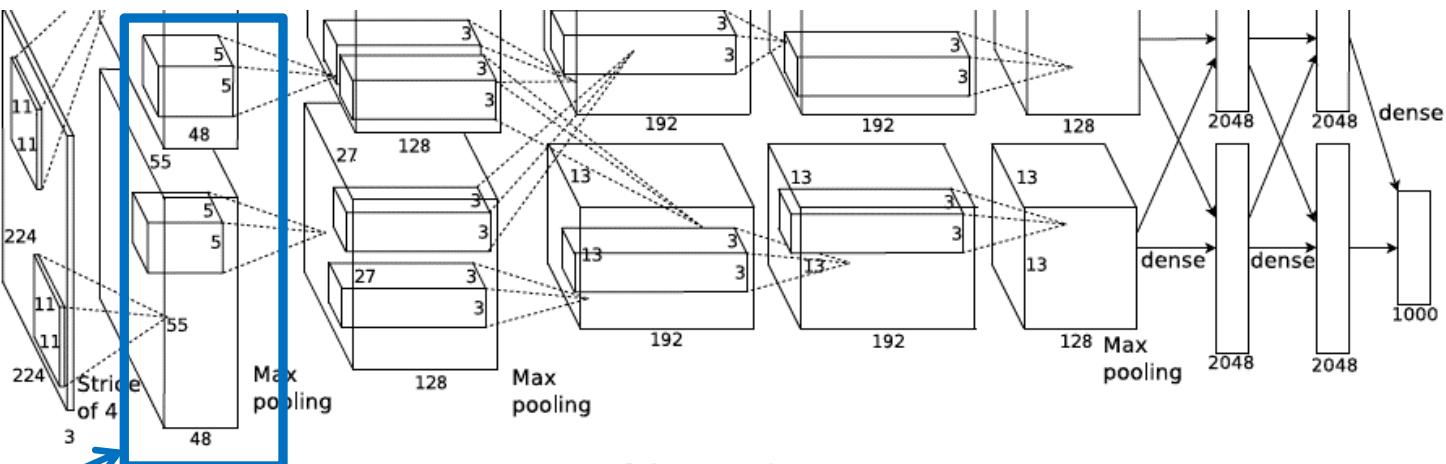
[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

AlexNet



Full (simplified) AlexNet architecture: **[55x55x48] x 2**

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

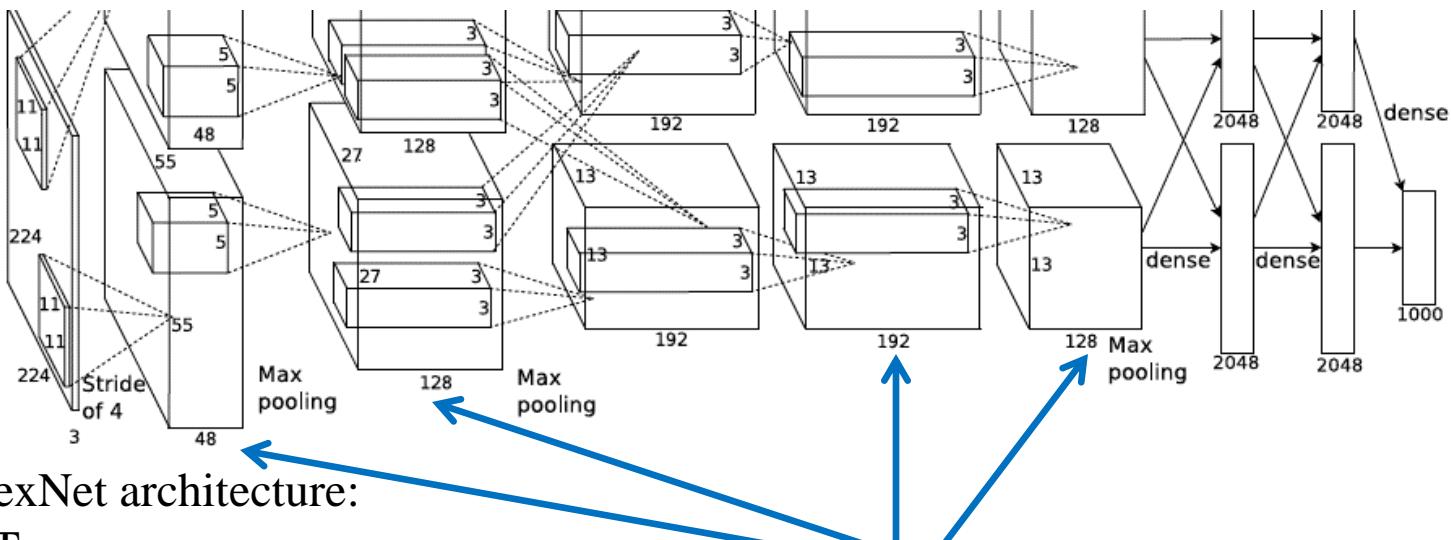
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

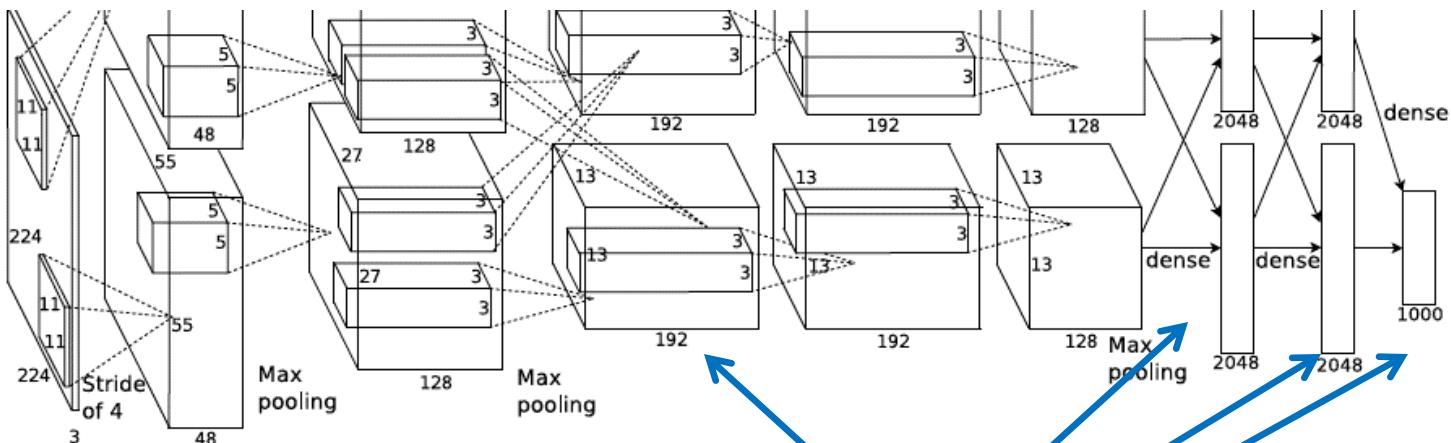
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

**CONV1, CONV2, CONV4,
CONV5: Connections only with
feature maps on same GPU**

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

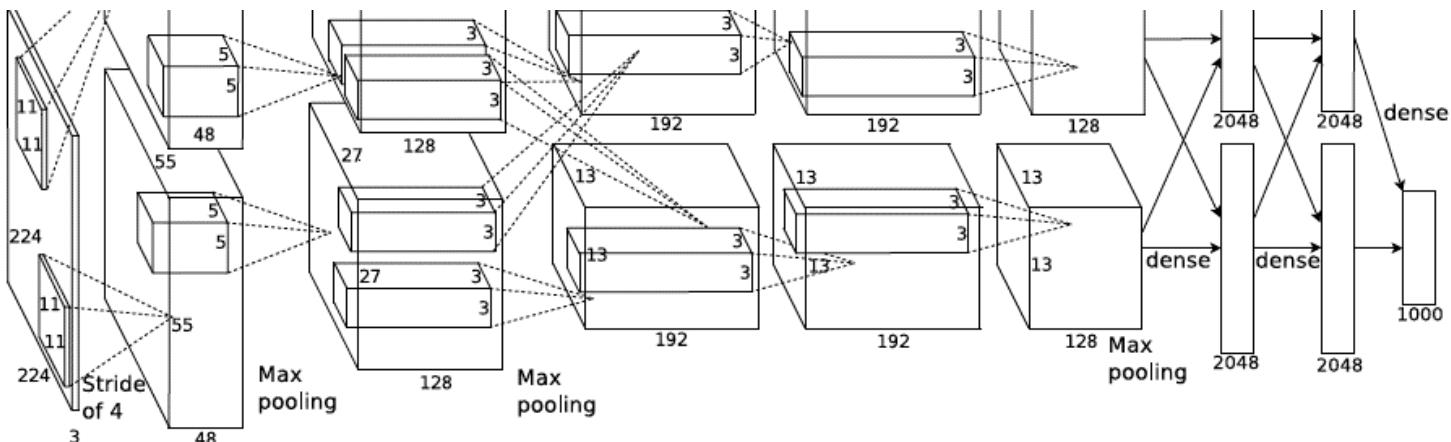
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

CONV3, FC6, FC7, FC8:
Connections with all feature
maps in preceding layer,
communication across GPUs

AlexNet



Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

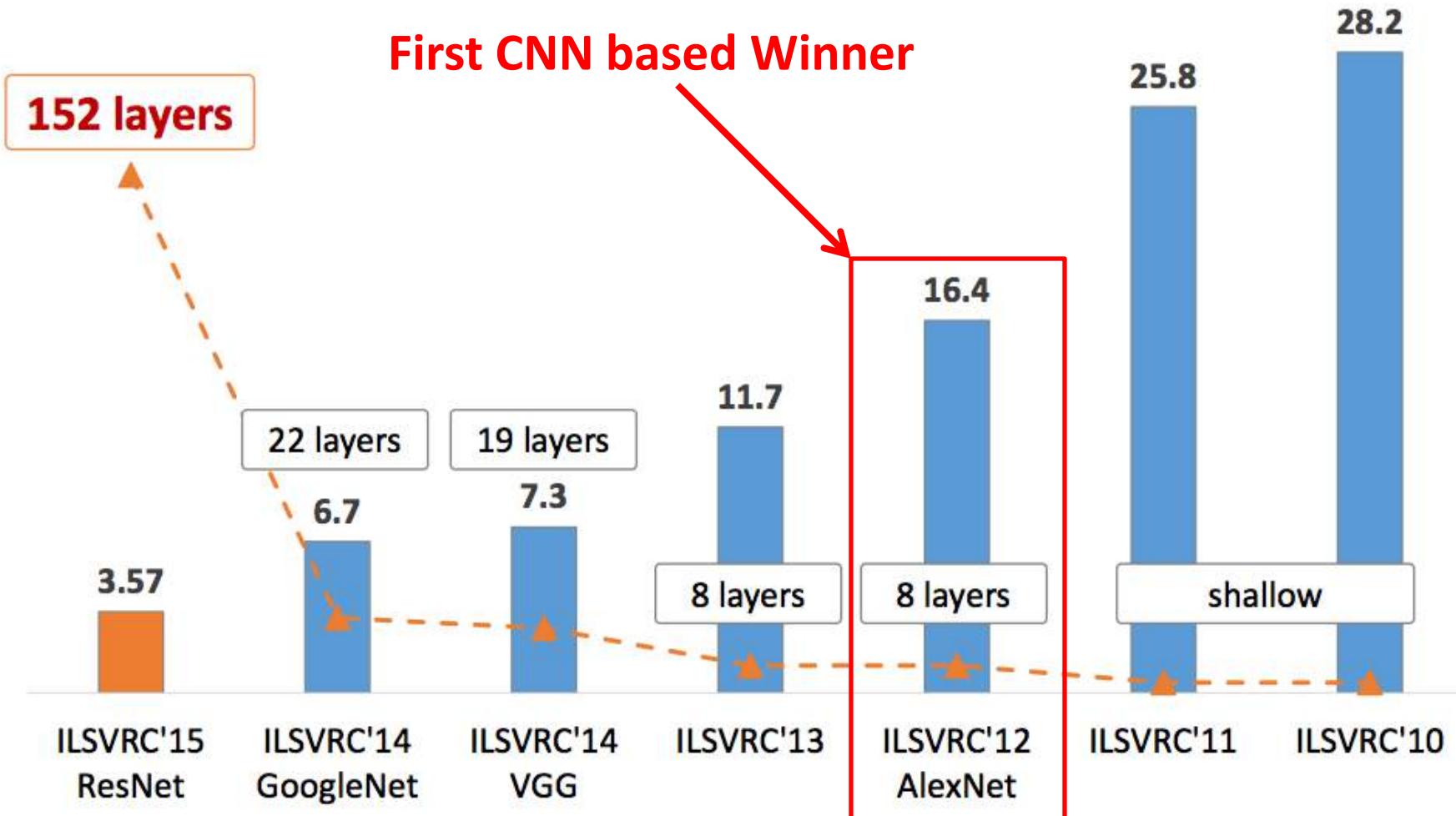
[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

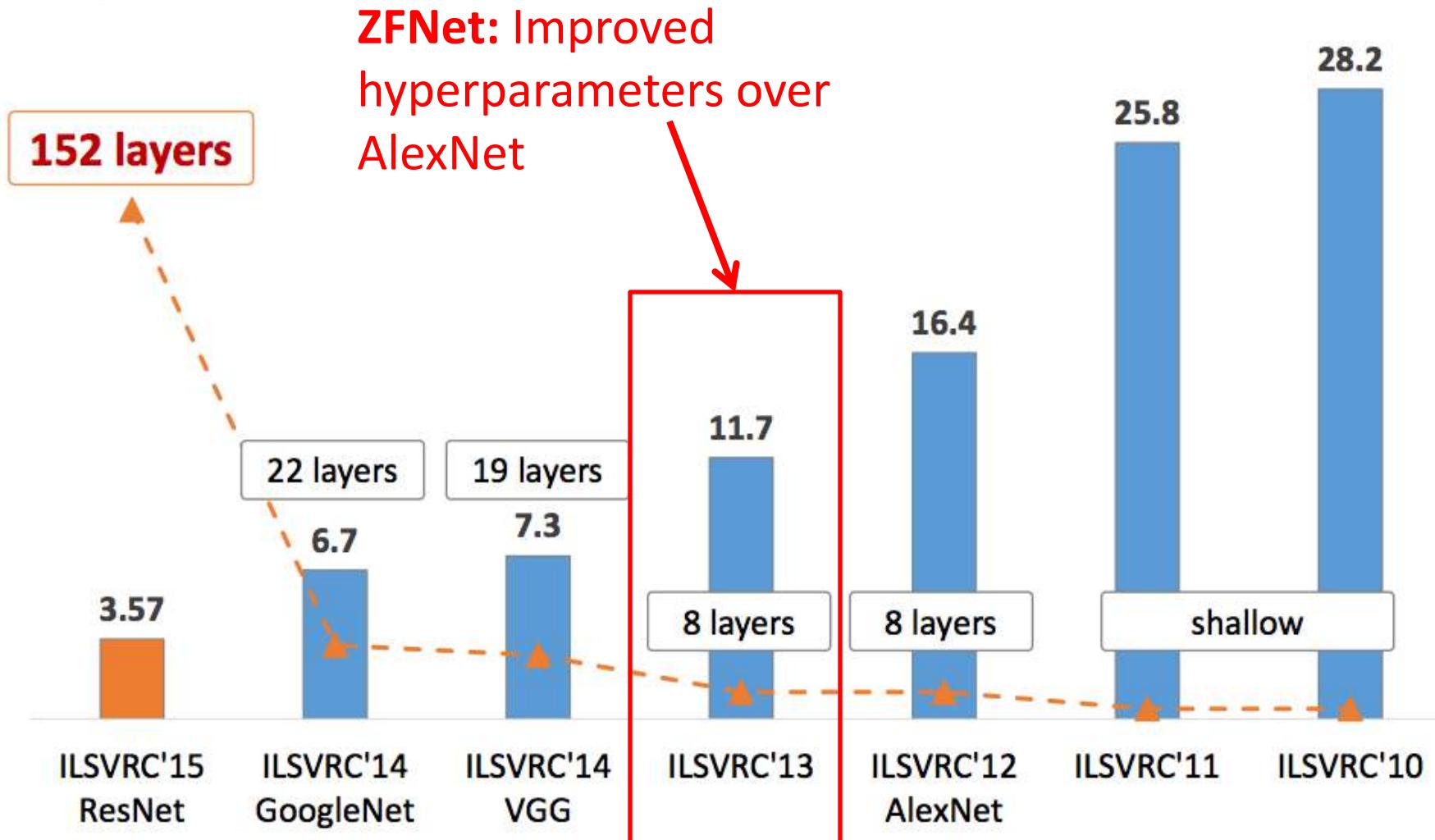
Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- batch size 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced manually when val accuracy saturates

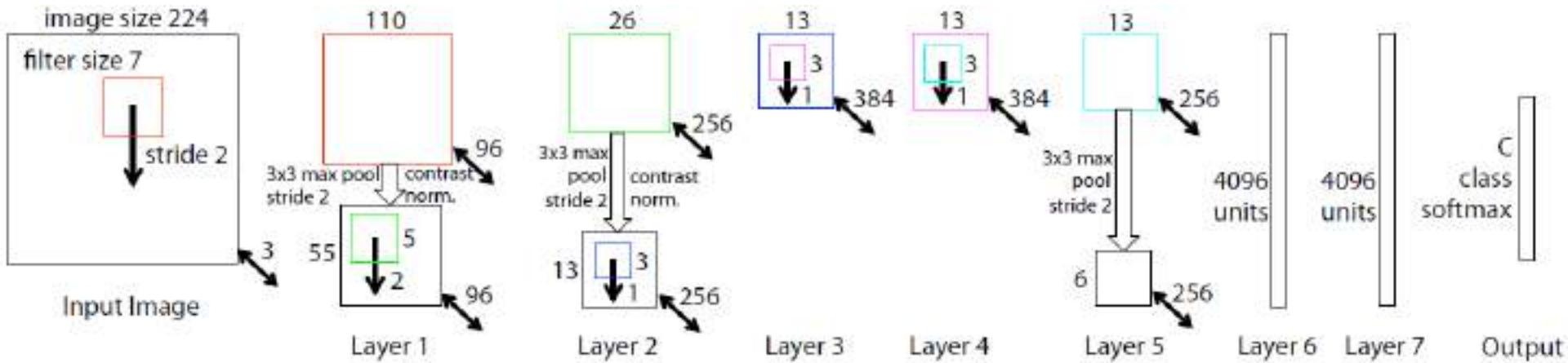
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ZFNet



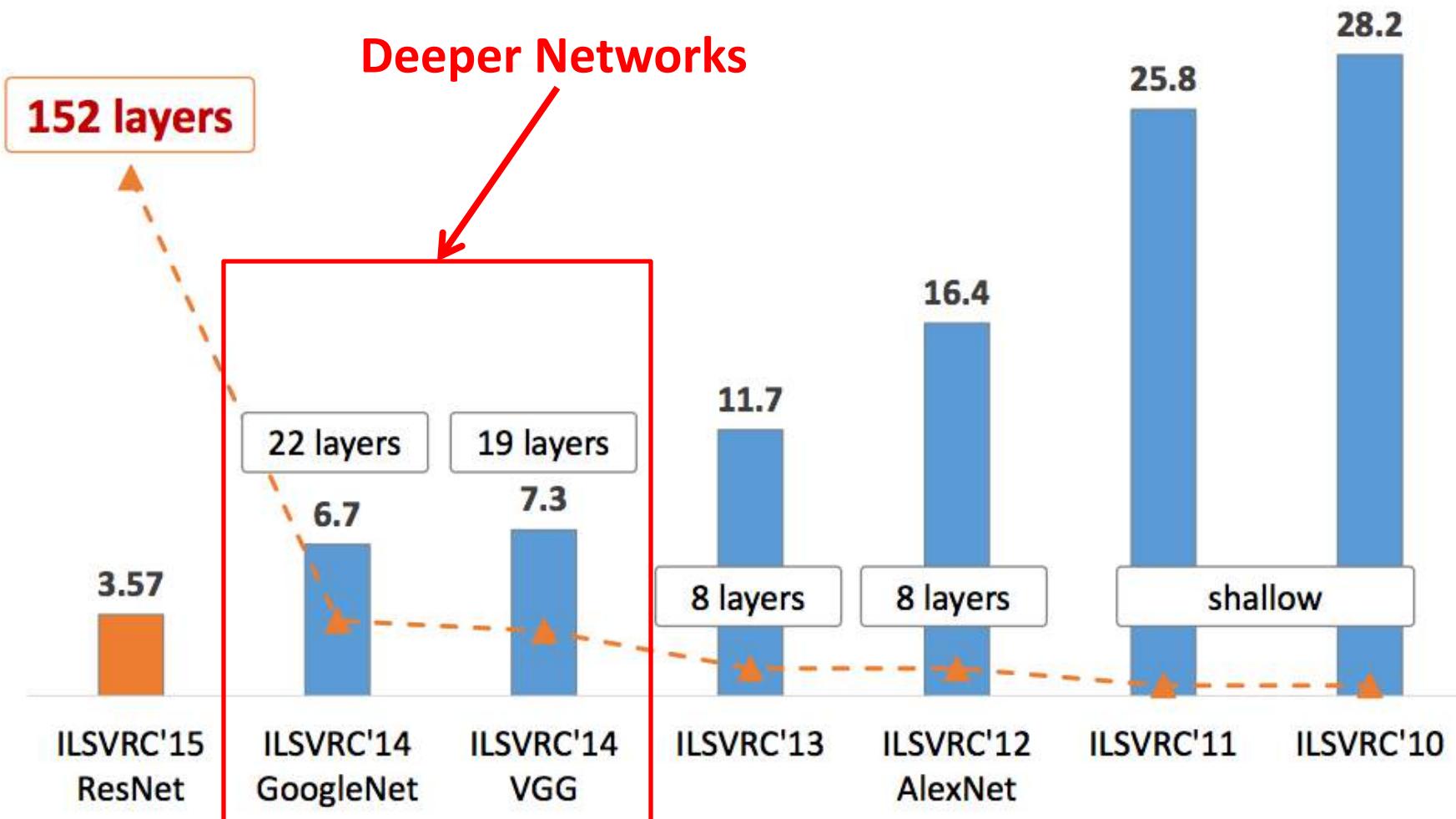
AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

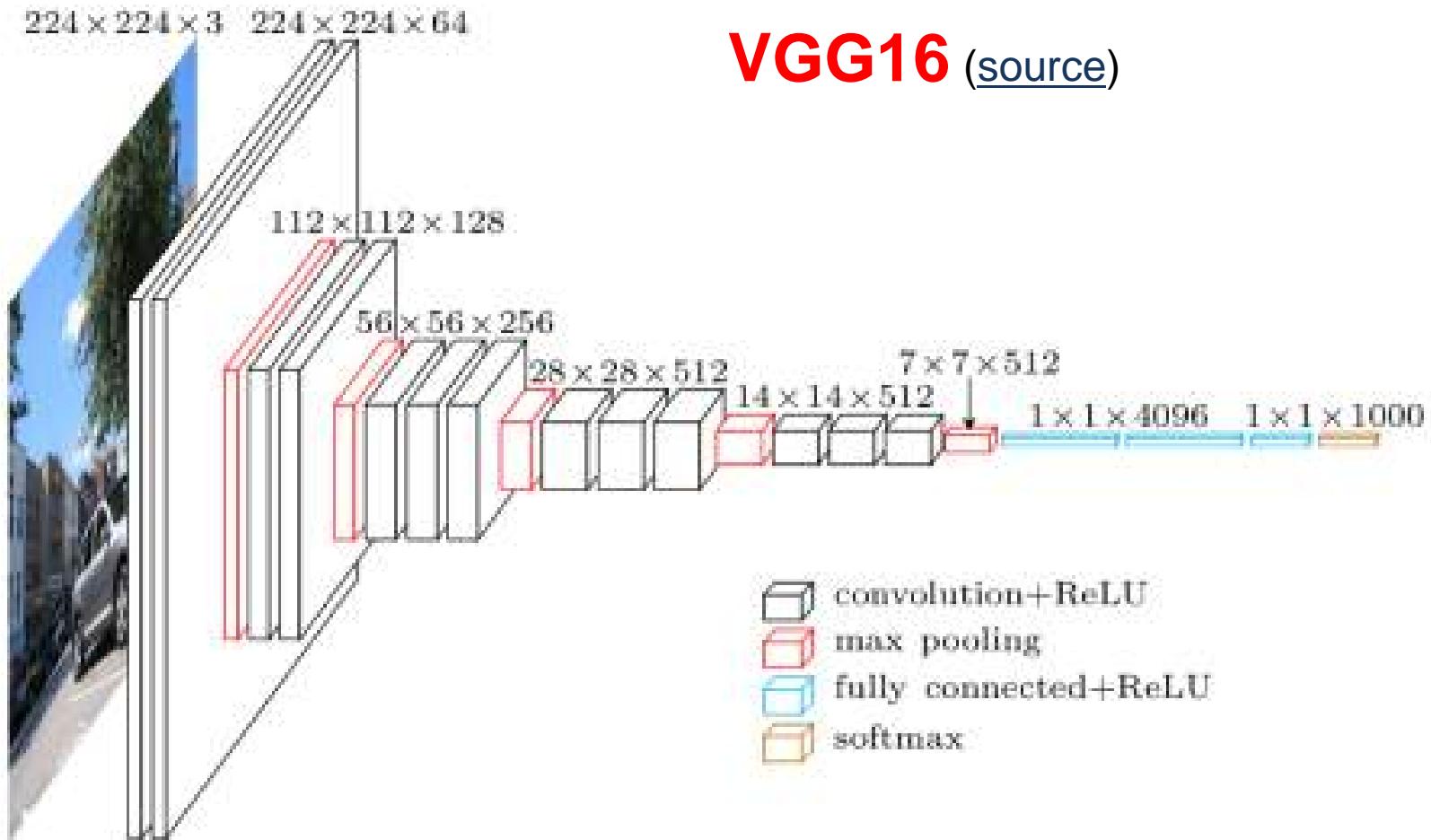
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% \rightarrow 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGGNet



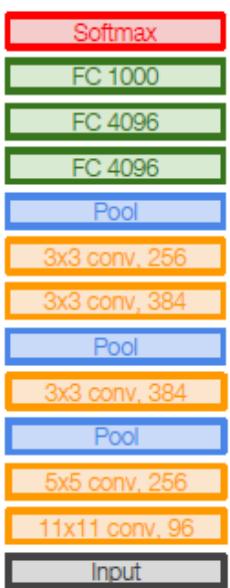
VGGNet

Small filters, Deeper networks

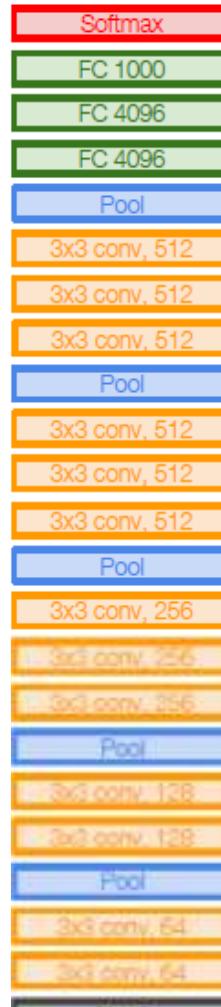
8 layers (AlexNet)

-> 16 - 19 layers (VGGNet)

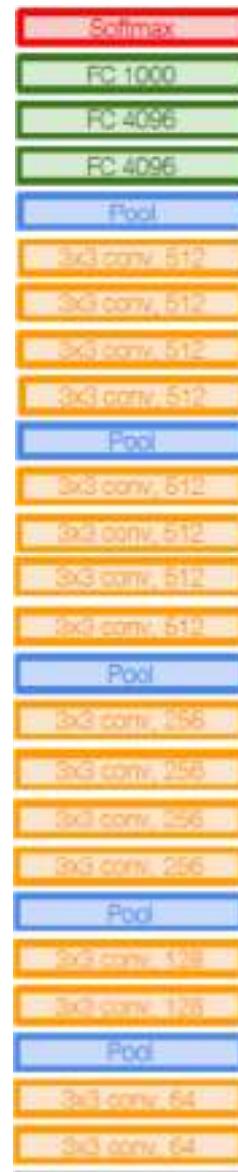
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2



AlexNet



VGG16



VGG19

VGGNet

Small filters, Deeper networks

8 layers (AlexNet)

-> 16 - 19 layers (VGGNet)

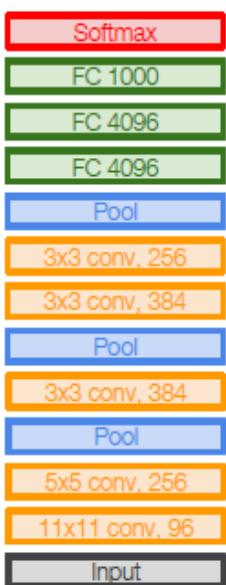
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

ImageNet top 5 error:

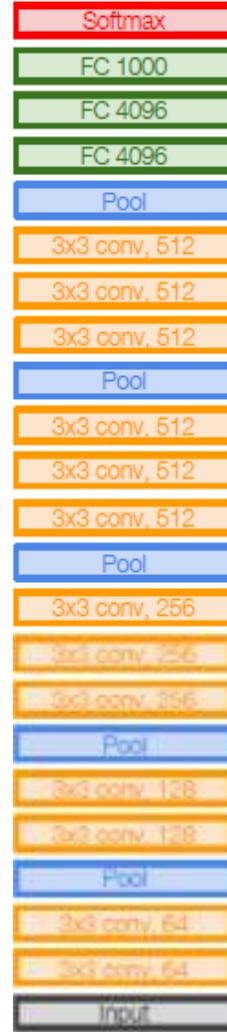
11.4% (ZFNet, 2013)

->

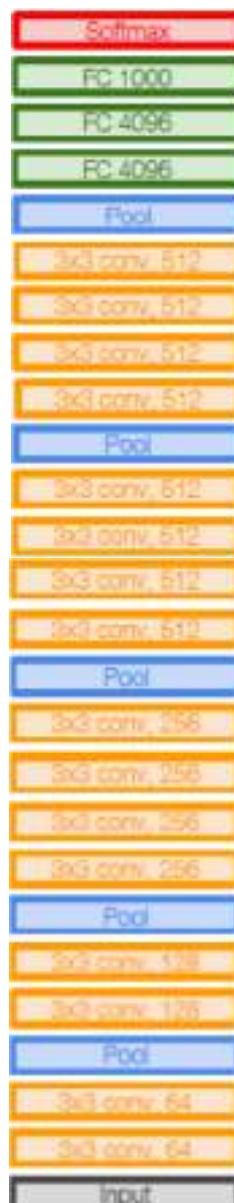
7.3% (VGGNet, 2014)



AlexNet



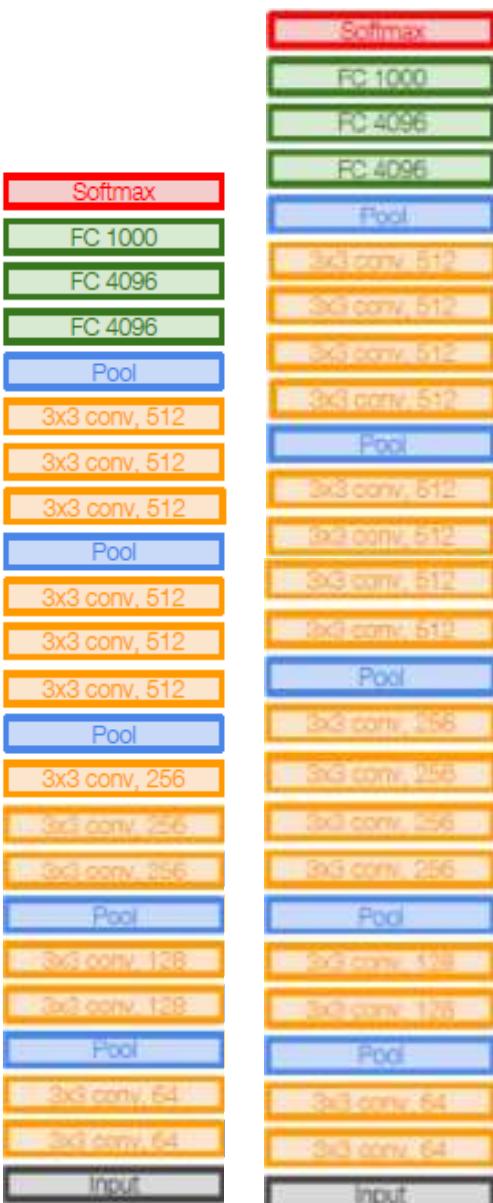
VGG16



VGG19

VGGNet

Q: Why use smaller filters? (3x3 conv)



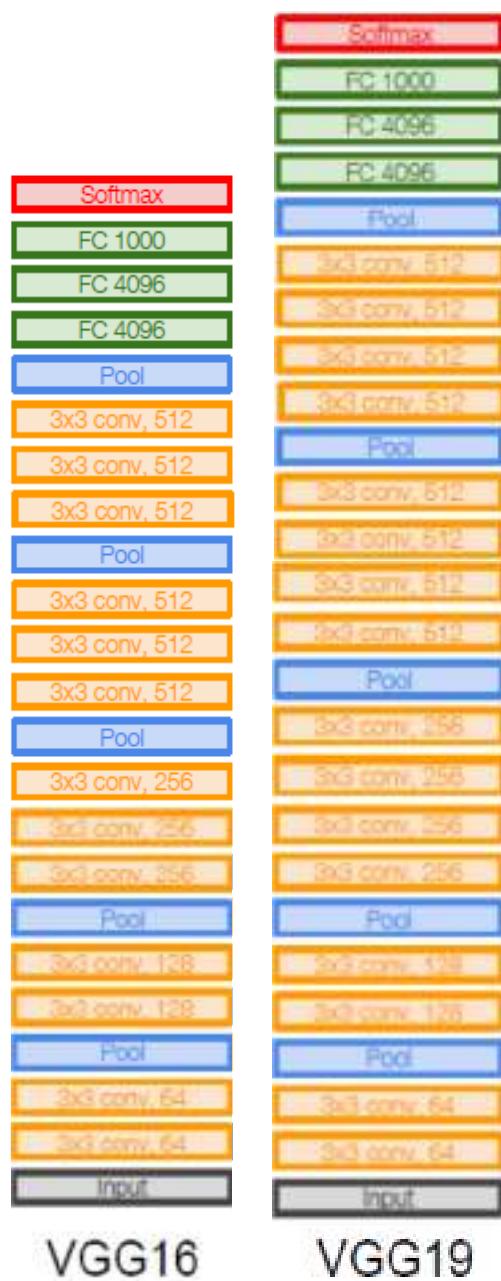
VGG16

VGG19

VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer



VGG16

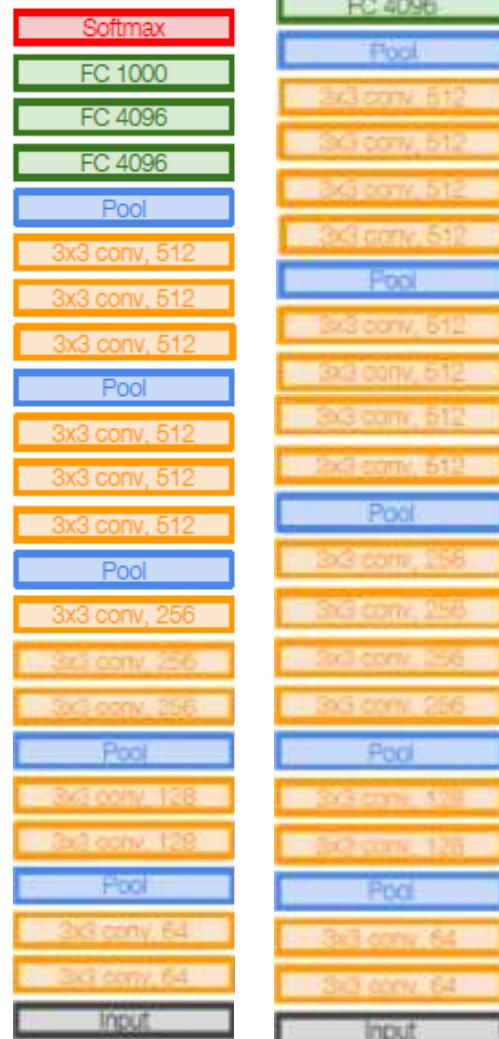
VGG19

VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

Q: What is the effective receptive field of
three 3x3 conv (stride 1) layers?



VGG16

VGG19

VGGNet

Q: Why use smaller filters? (3x3 conv)

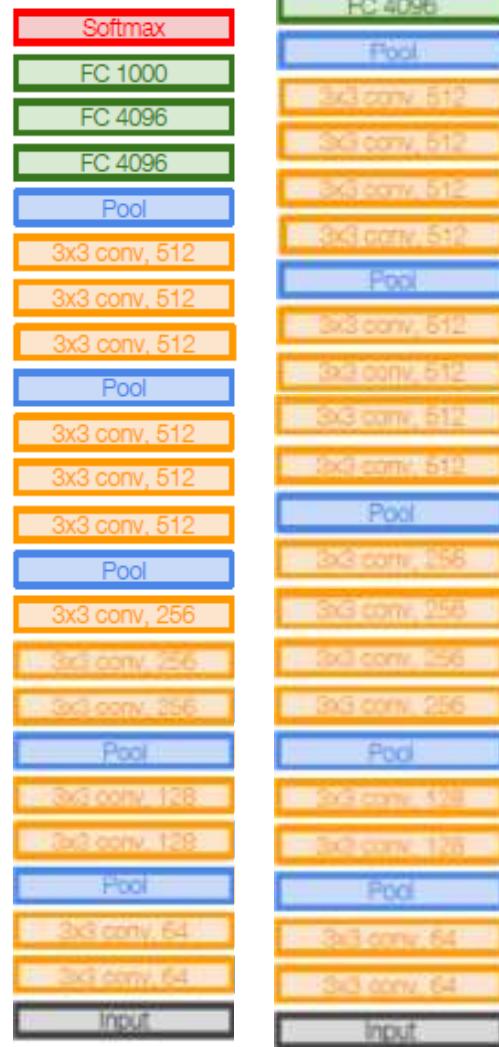
Stack of three 3x3 conv (stride 1) layers
has same **effective receptive field** as
one 7x7 conv layer

Q: What is the effective receptive field of
three 3x3 conv (stride 1) layers?

[7x7]

But deeper, more non-linearities

And fewer parameters: $3 * (3^2 C^2)$ vs.
 $7^2 C^2$ for C channels per layer



VGG16

VGG19

VGGNet

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$
POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0
CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$
CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$
POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0
FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$
FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$
FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

VGGNet

INPUT: [224x224x3] memory: $224 \times 224 \times 3 = 150K$ params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 3) \times 64 = 1,728$
CONV3-64: [224x224x64] memory: $224 \times 224 \times 64 = 3.2M$ params: $(3 \times 3 \times 64) \times 64 = 36,864$
POOL2: [112x112x64] memory: $112 \times 112 \times 64 = 800K$ params: 0
CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 64) \times 128 = 73,728$
CONV3-128: [112x112x128] memory: $112 \times 112 \times 128 = 1.6M$ params: $(3 \times 3 \times 128) \times 128 = 147,456$
POOL2: [56x56x128] memory: $56 \times 56 \times 128 = 400K$ params: 0
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 128) \times 256 = 294,912$
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256: [56x56x256] memory: $56 \times 56 \times 256 = 800K$ params: $(3 \times 3 \times 256) \times 256 = 589,824$
POOL2: [28x28x256] memory: $28 \times 28 \times 256 = 200K$ params: 0
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 256) \times 512 = 1,179,648$
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [28x28x512] memory: $28 \times 28 \times 512 = 400K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: 0
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14 \times 14 \times 512 = 100K$ params: $(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2: [7x7x512] memory: $7 \times 7 \times 512 = 25K$ params: 0
FC: [1x1x4096] memory: 4096 params: $7 \times 7 \times 512 \times 4096 = 102,760,448$
FC: [1x1x4096] memory: 4096 params: $4096 \times 4096 = 16,777,216$
FC: [1x1x1000] memory: 1000 params: $4096 \times 1000 = 4,096,000$

TOTAL memory: 24M * 4 bytes ~ = 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

VGGNet

INPUT: [224x224x3]	memory: 224*224*3=150K	params: 0	(not counting biases)
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728	
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864	
POOL2: [112x112x64]	memory: 112*112*64=800K	params: 0	
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728	
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456	
POOL2: [56x56x128]	memory: 56*56*128=400K	params: 0	
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912	
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824	
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824	
POOL2: [28x28x256]	memory: 28*28*256=200K	params: 0	
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648	
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296	
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296	
POOL2: [14x14x512]	memory: 14*14*512=100K	params: 0	
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296	
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296	
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296	
POOL2: [7x7x512]	memory: 7*7*512=25K	params: 0	
FC: [1x1x4096]	memory: 4096	params: 7*7*512*4096 = 102,760,448	
FC: [1x1x4096]	memory: 4096	params: 4096*4096 = 16,777,216	
FC: [1x1x1000]	memory: 1000	params: 4096*1000 = 4,096,000	

Note:

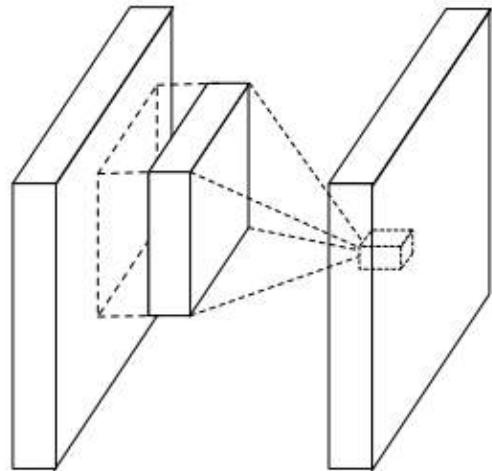
Most memory is in early CONV

Most params are in late FC

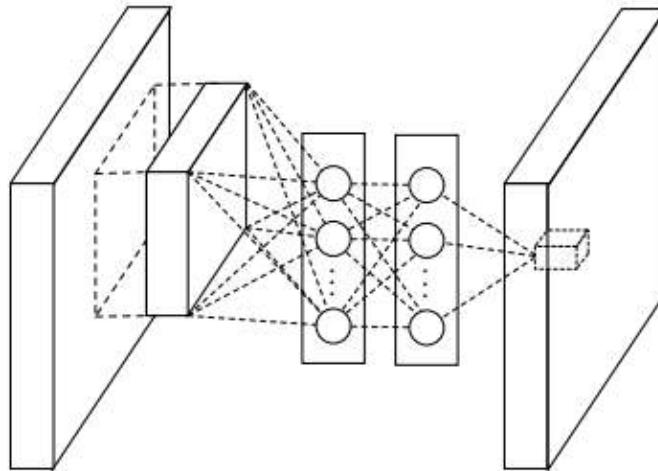
TOTAL memory: 24M * 4 bytes ~ = 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Network in Network (NiN)

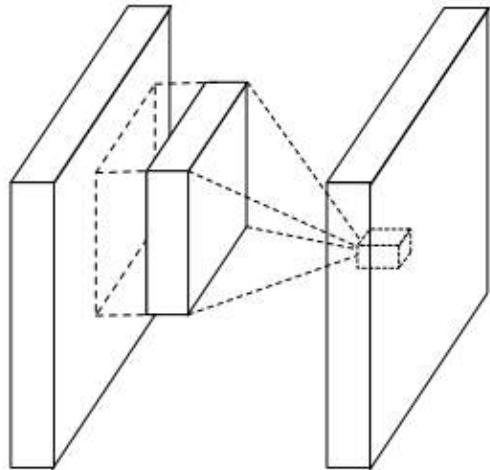


(a) Linear convolution layer

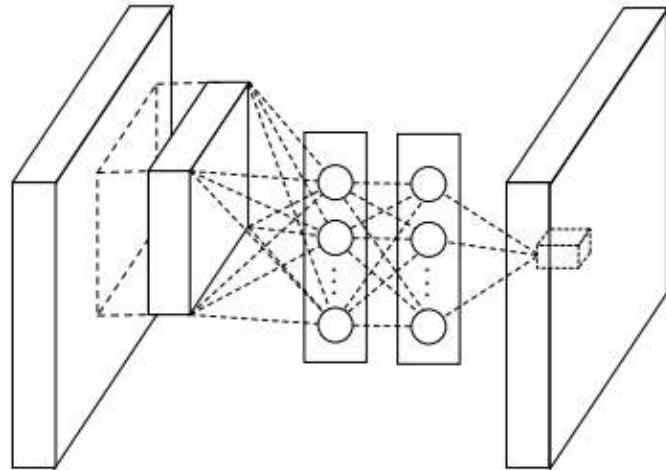


(b) Mlpconv layer

Network in Network (NiN)



(a) Linear convolution layer

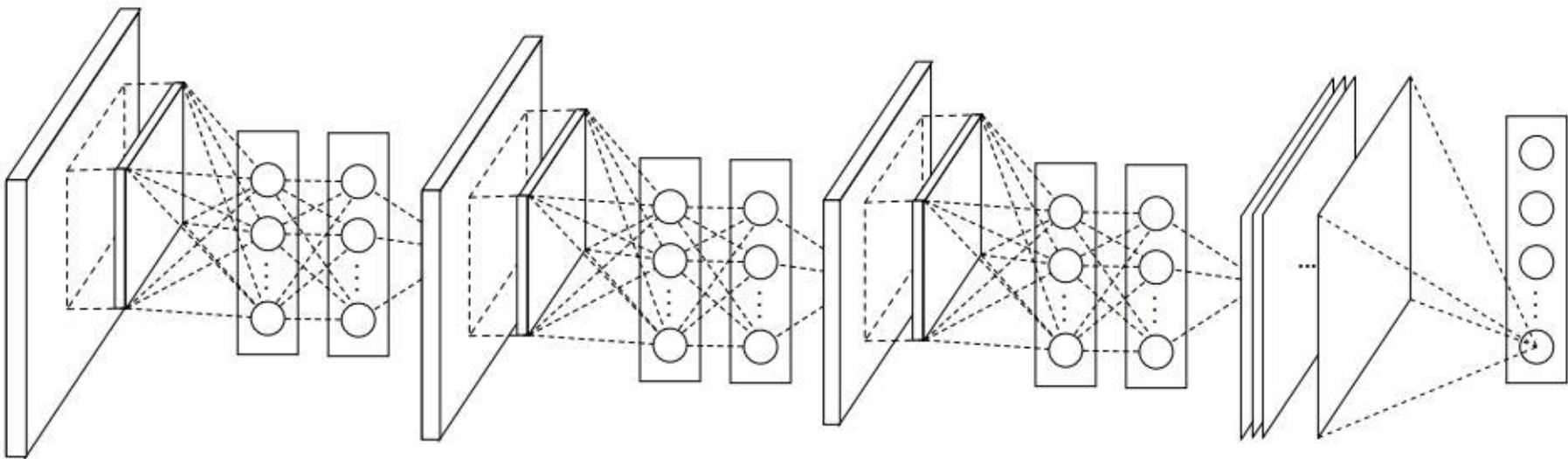


(b) Mlpconv layer

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron (FC, i.e. 1x1 conv layers)

Network in Network (NiN)

The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer



Network in Network (NiN)

The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer

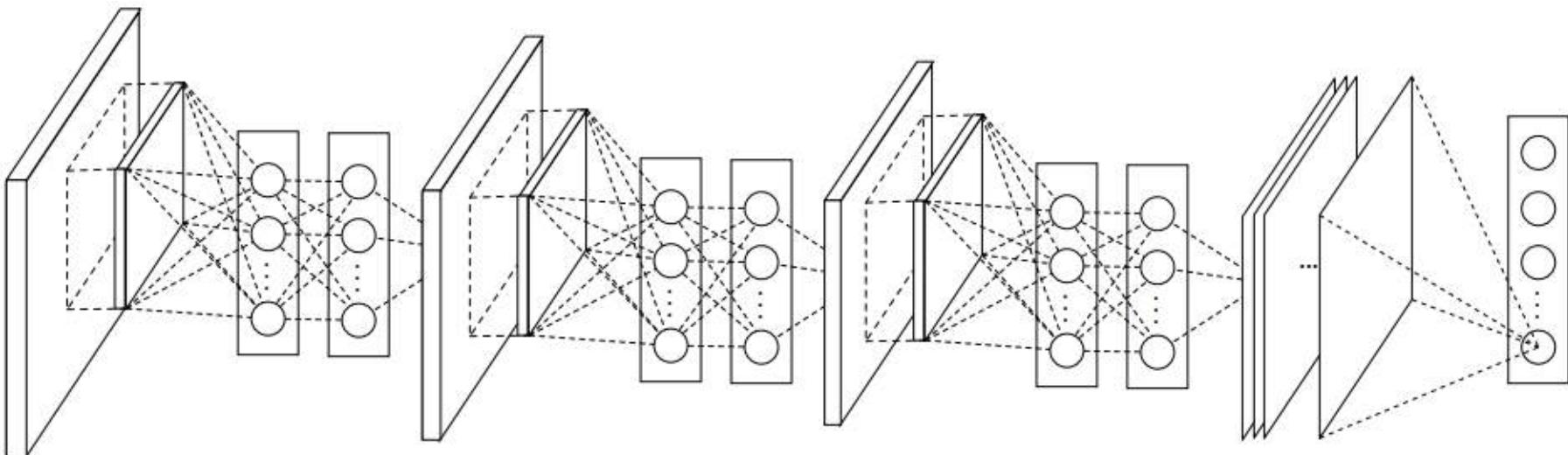
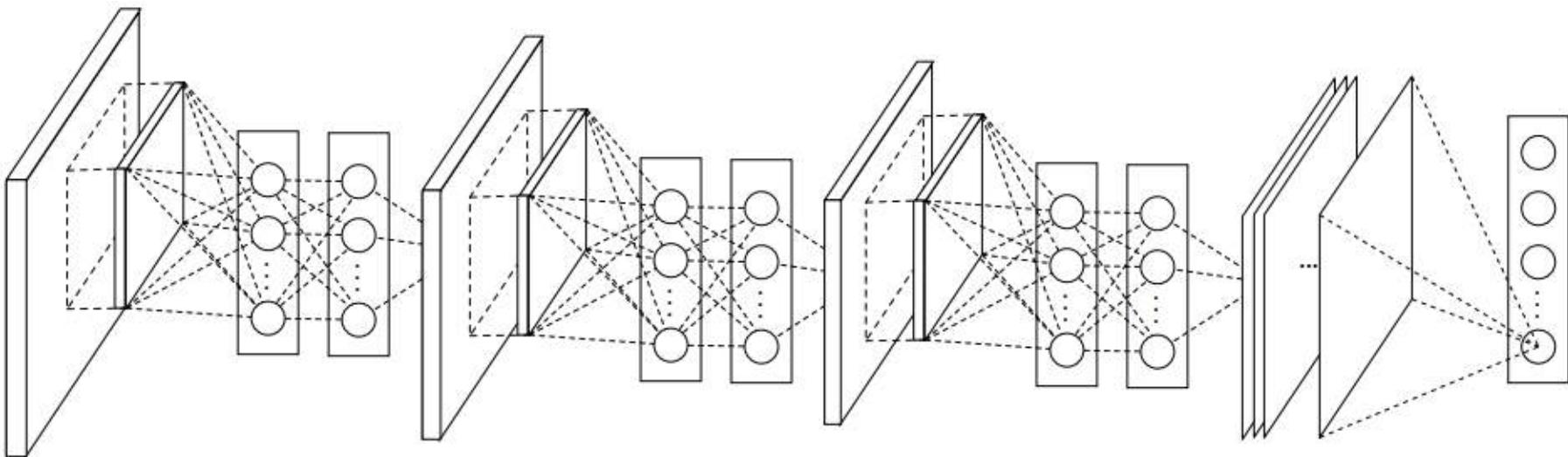


Table 1: Test set error rates for CIFAR-10 of various methods.

Method	Test Error
Stochastic Pooling [11]	15.13%
CNN + Spearmint [14]	14.98%
Conv. maxout + Dropout [8]	11.68%
NIN + Dropout	10.41%
CNN + Spearmint + Data Augmentation [14]	9.50%
Conv. maxout + Dropout + Data Augmentation [8]	9.38%
DropConnect + 12 networks + Data Augmentation [15]	9.32%
NIN + Dropout + Data Augmentation	8.81%

Network in Network (NiN)

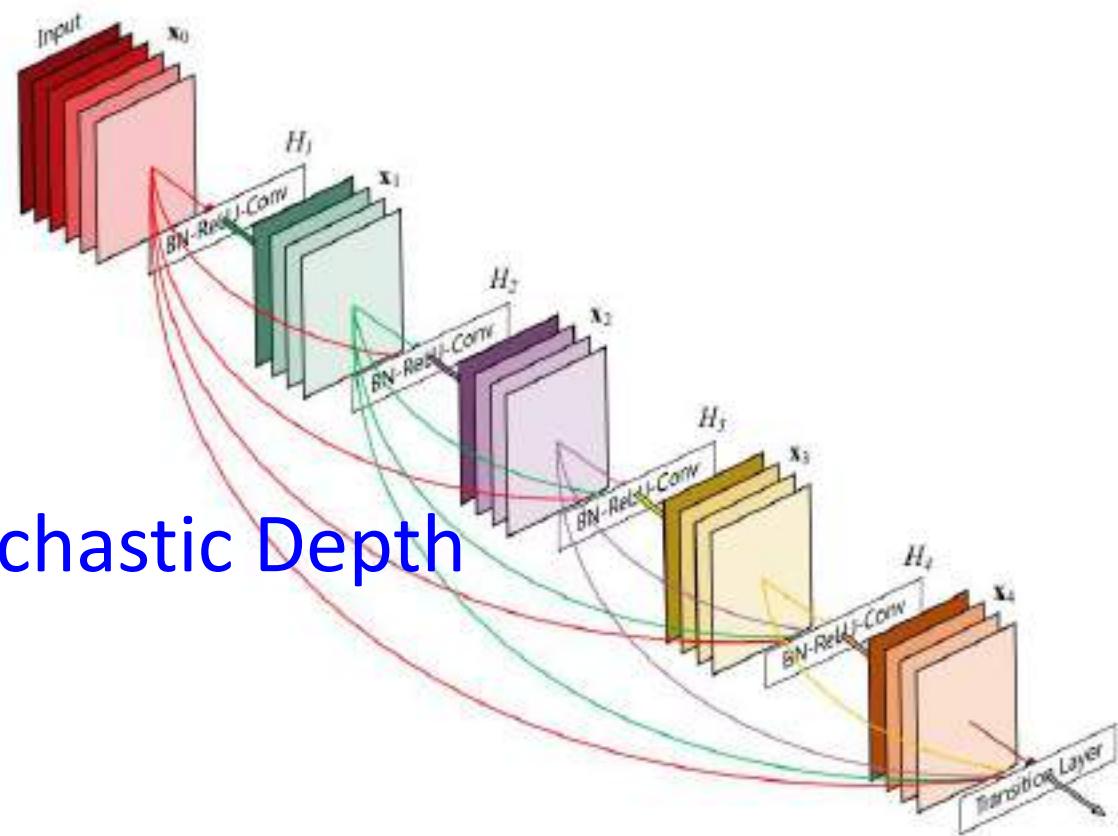
The overall structure of NiN: stacking of three mlpconv layers and one global average pooling layer



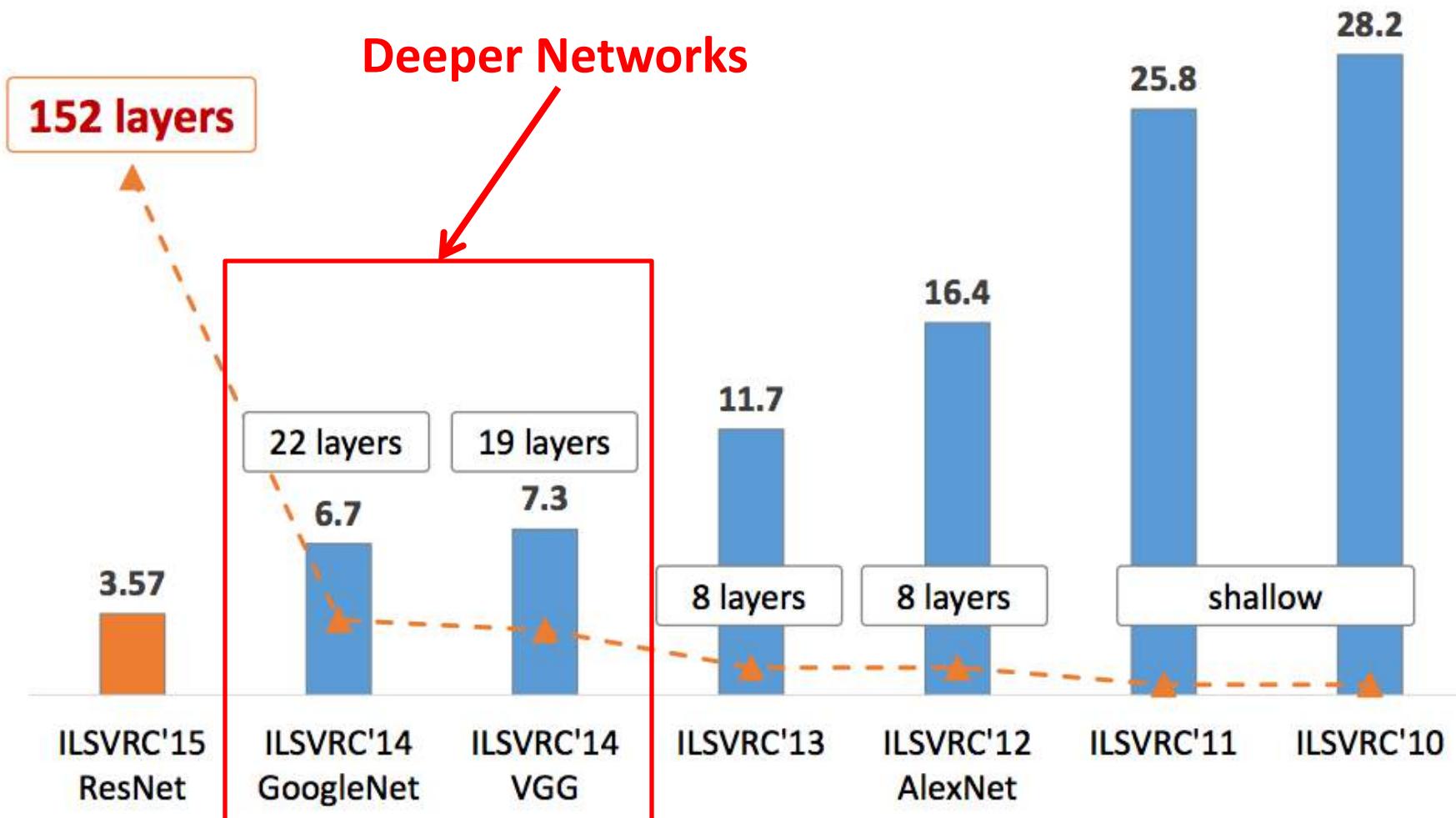
- Precursor to GoogLeNet and ResNet
“bottleneck” layers
- Philosophical inspiration for GoogLeNet

CNN Architectures: DAG Models

- GoogLeNet
- ResNet
- Pre-act ResNet
- SENet
- Network with Stochastic Depth
- DenseNet
- ResNetXt



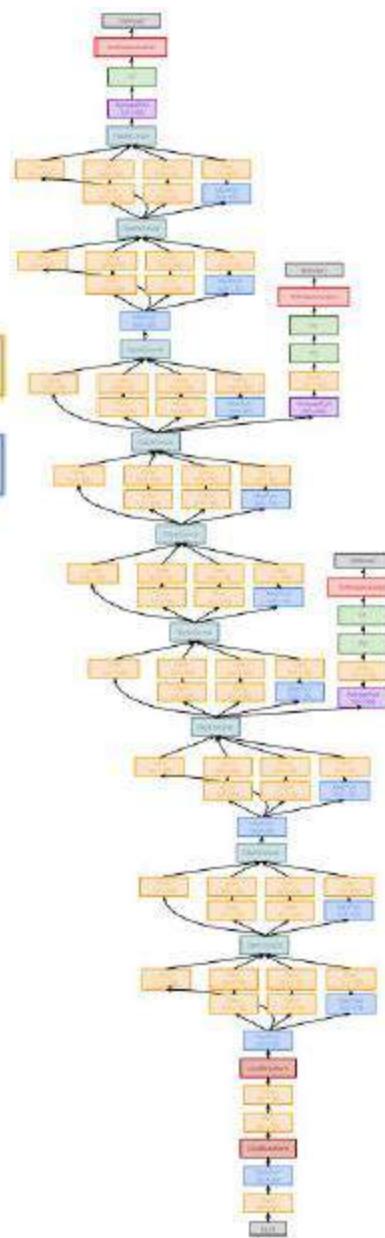
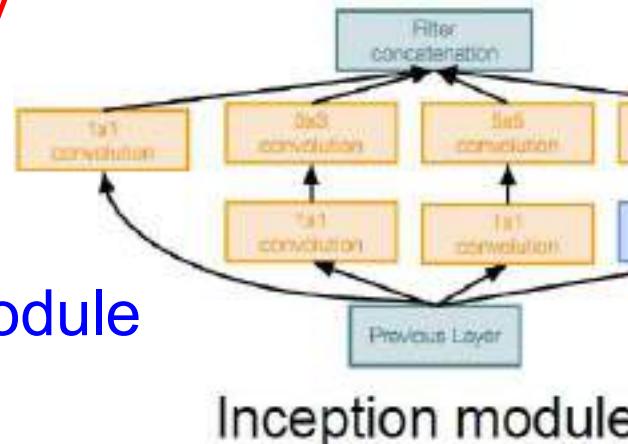
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



GoogLeNet

Deeper networks, with computational efficiency

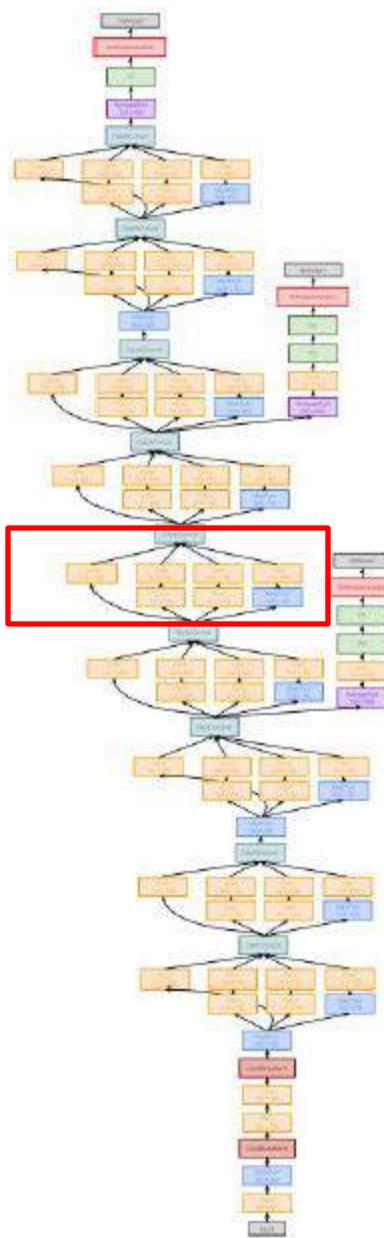
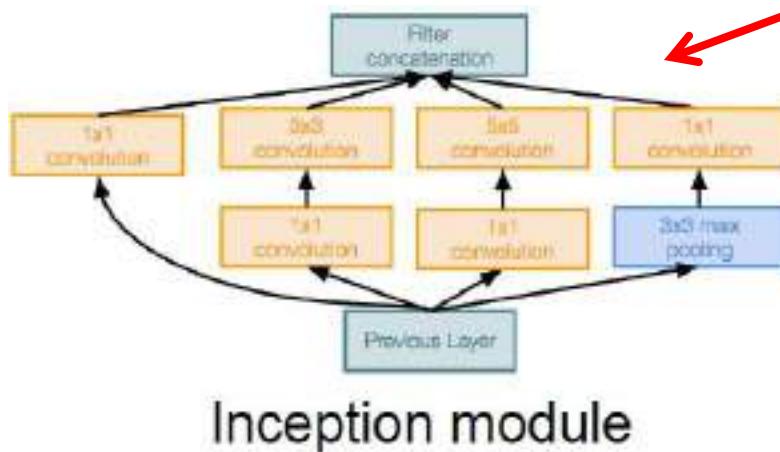
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
12x less than AlexNet
- Imagenet classification winner
(6.7% top 5 error)



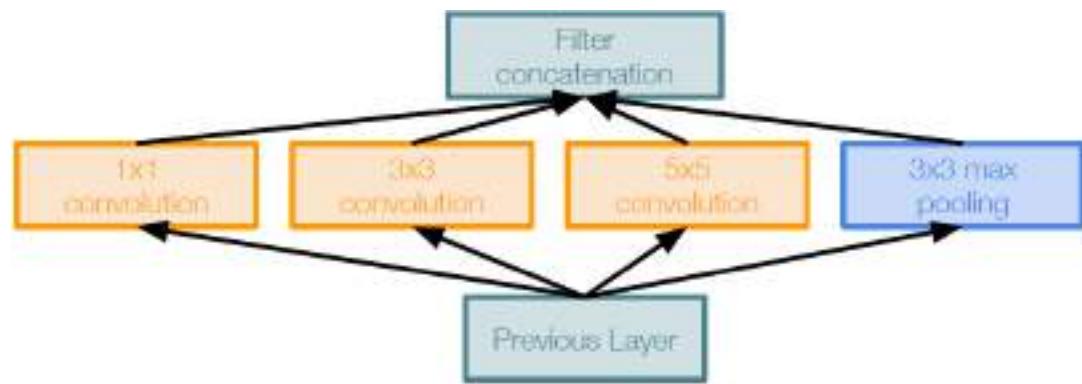
GoogLeNet

“Inception module”:

design a good local network topology and then stack these modules on top of each other



GoogLeNet



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

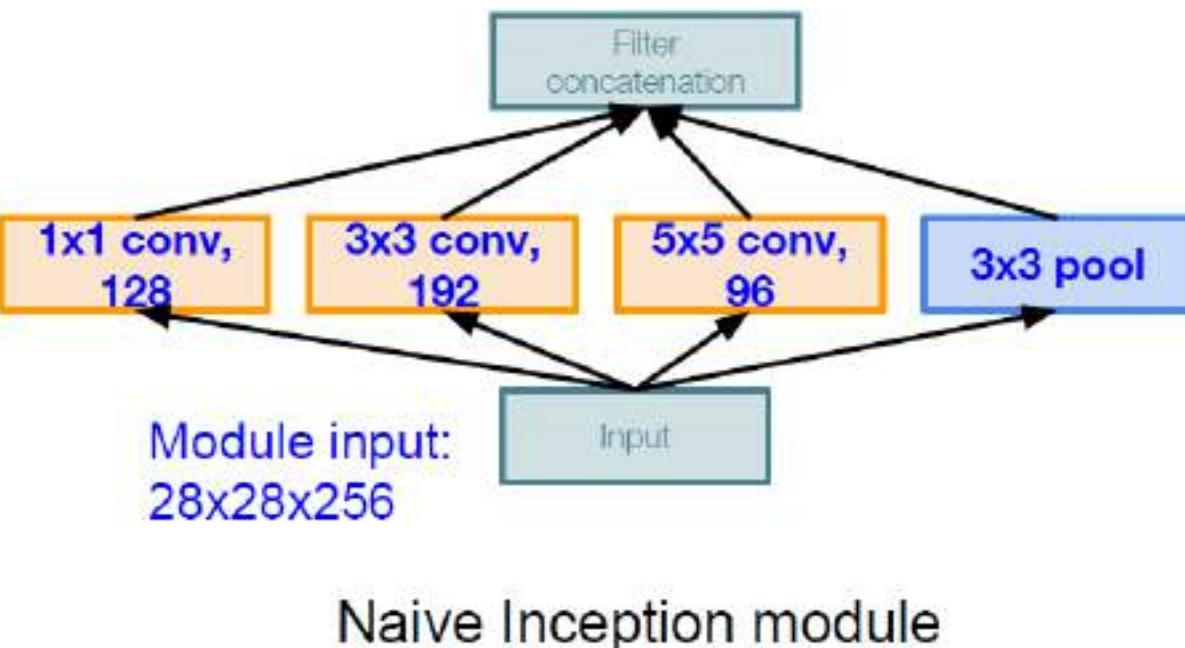
Problem:
Computational Complexity

GoogLeNet

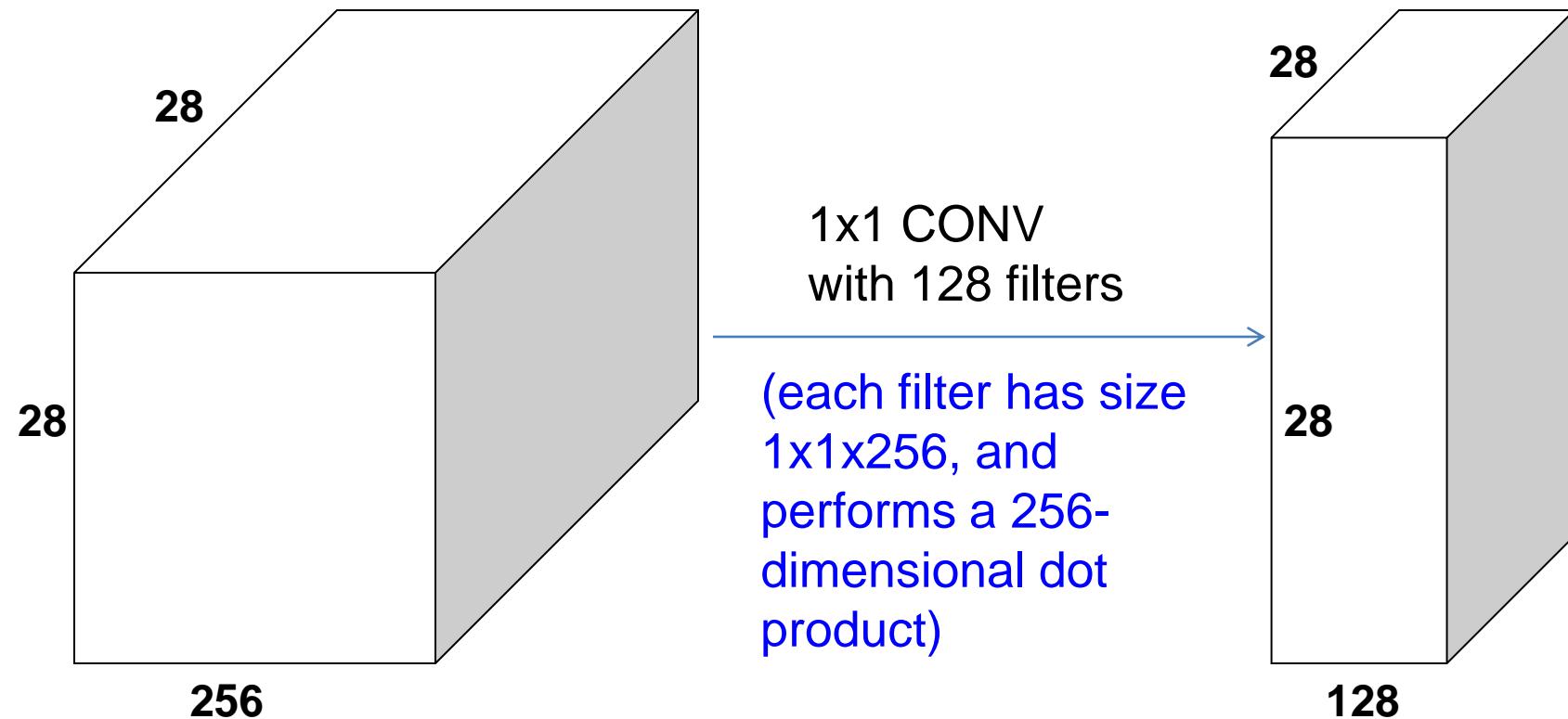
Problem:
Computational Complexity

Q1: What is the output size of the
1x1 conv, with 128 filters?

Example:



1×1 Convolutions

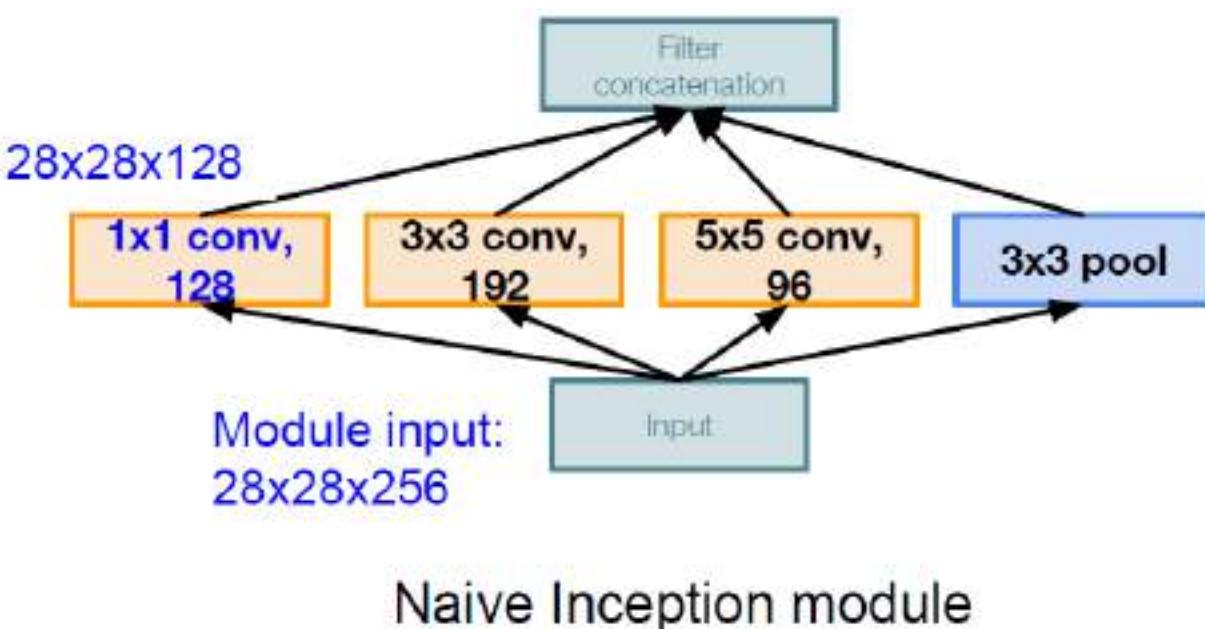


GoogLeNet

Problem:
Computational Complexity

Q1: What is the output size of the
1x1 conv, with 128 filters?

Example:

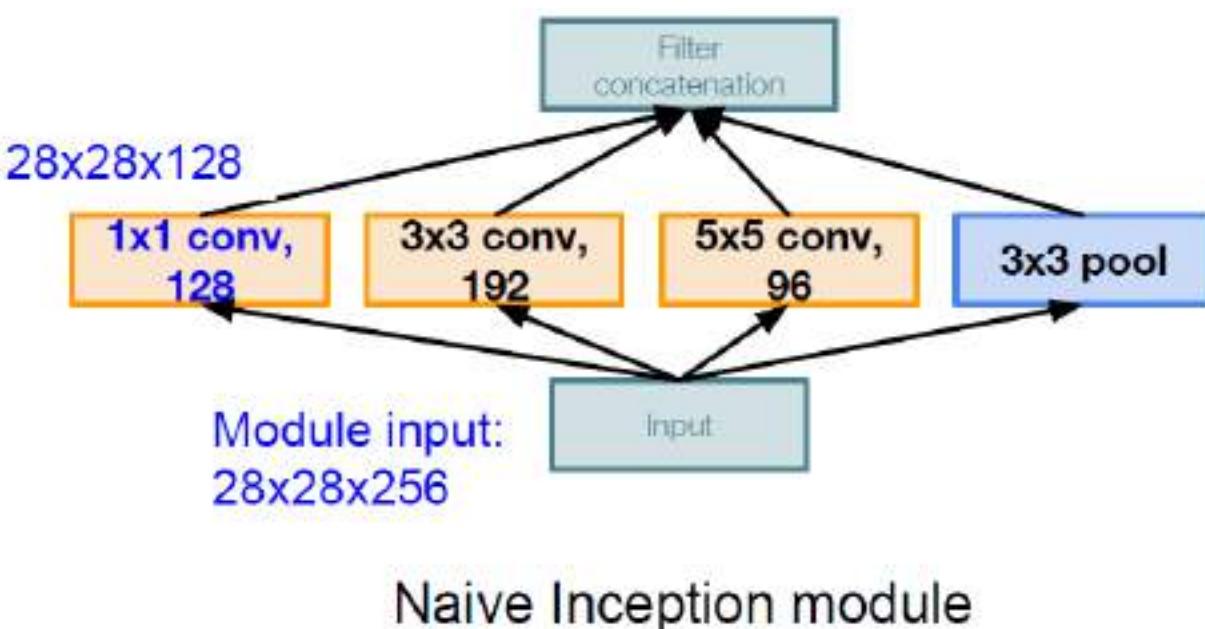


GoogLeNet

Problem:
Computational Complexity

Q2: What are the output sizes of all different filter operations?

Example:

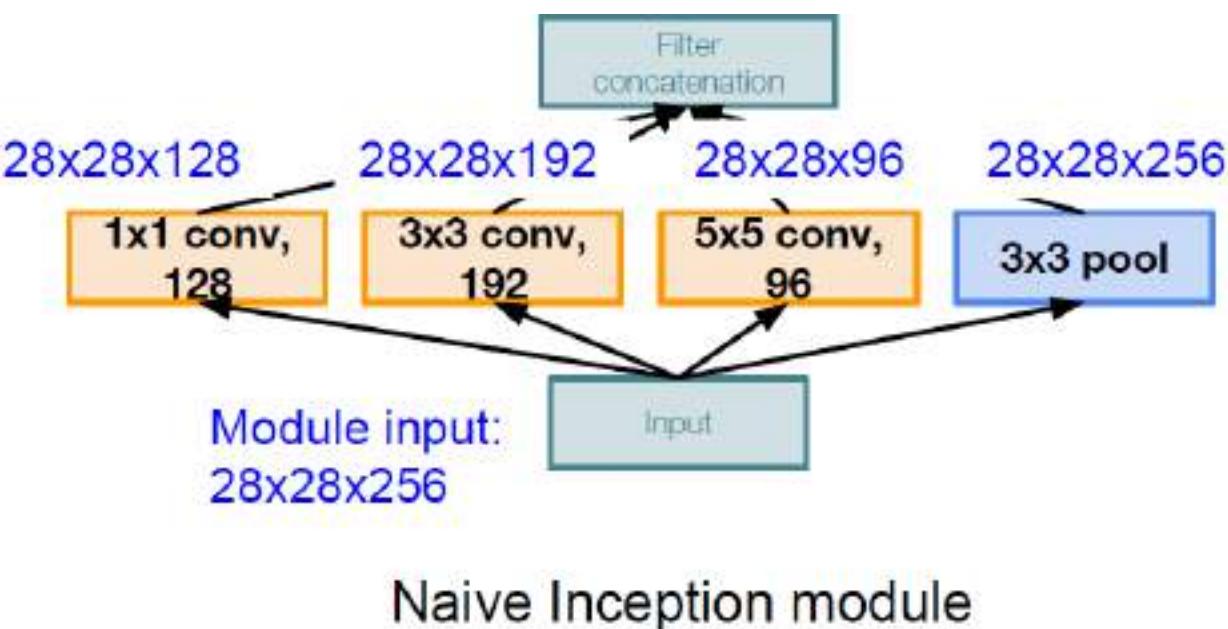


GoogLeNet

Problem:
Computational Complexity

Q2: What are the output sizes of all different filter operations?

Example:

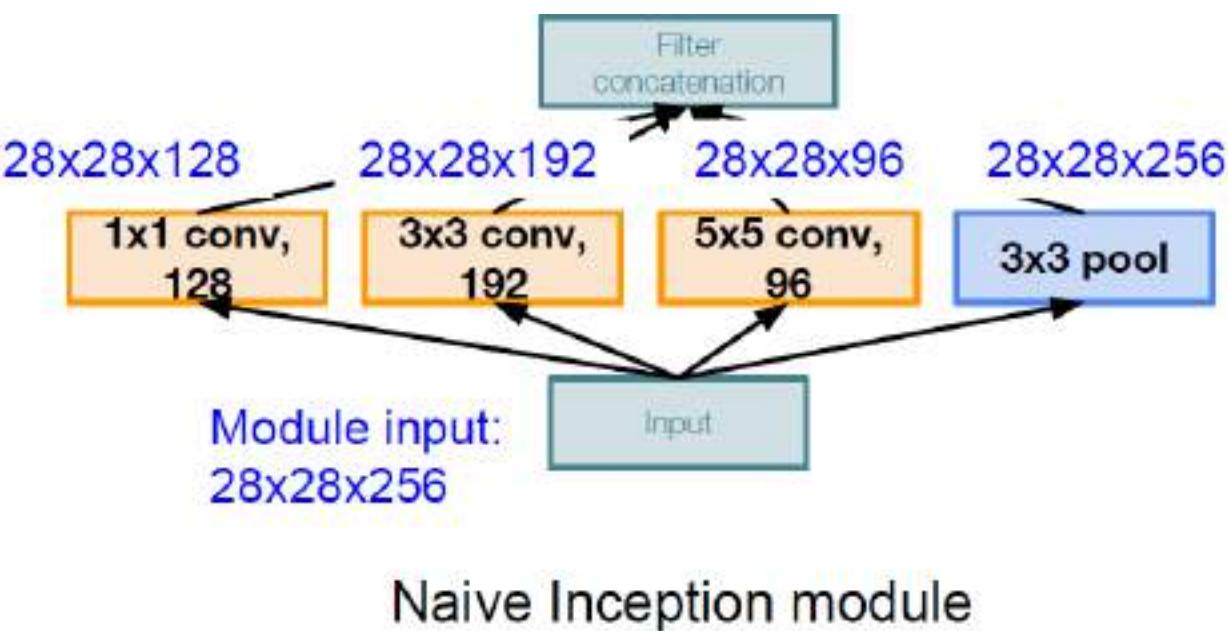


GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:



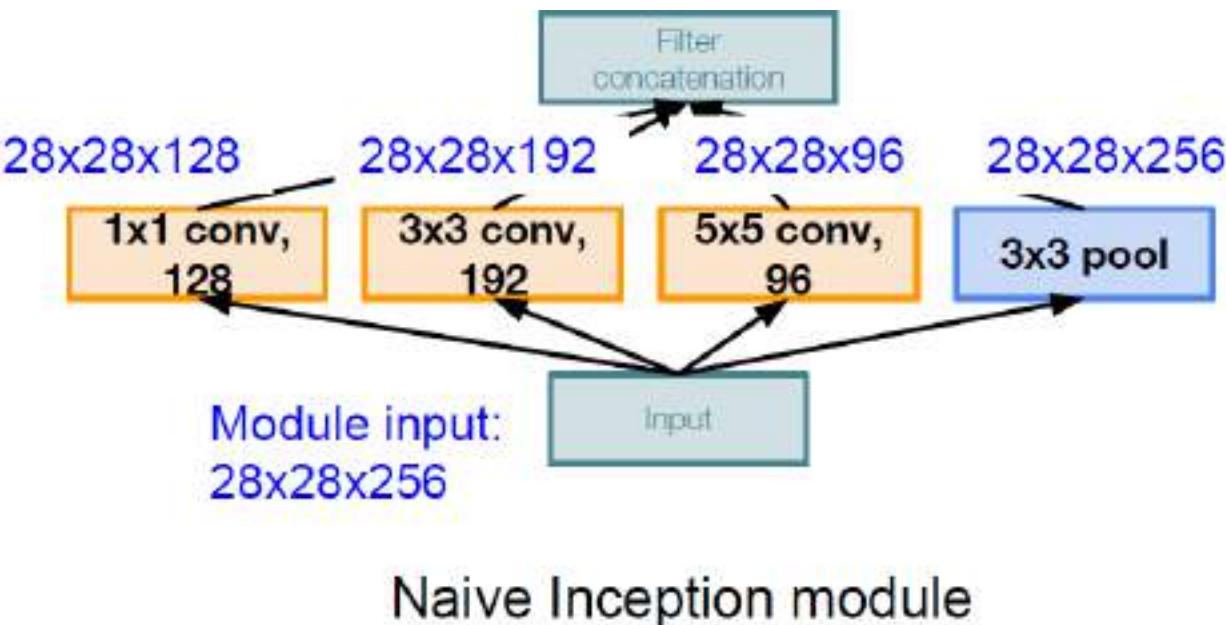
GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



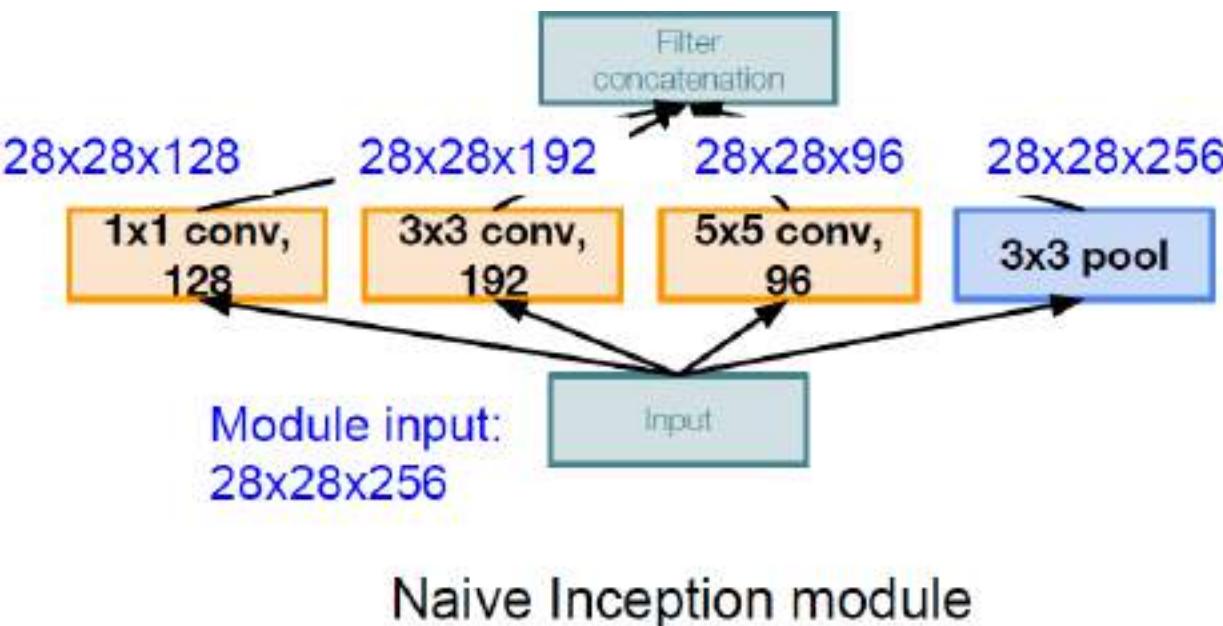
GoogLeNet

Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

Example:

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$



Conv Ops:

[1×1 conv, 128]

$28 \times 28 \times 128 \times 1 \times 1 \times 256$

[3×3 conv, 192]

$28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5×5 conv, 96]

$28 \times 28 \times 96 \times 5 \times 5 \times 256$

Total: 854M ops

**Very expensive
compute**

GoogLeNet

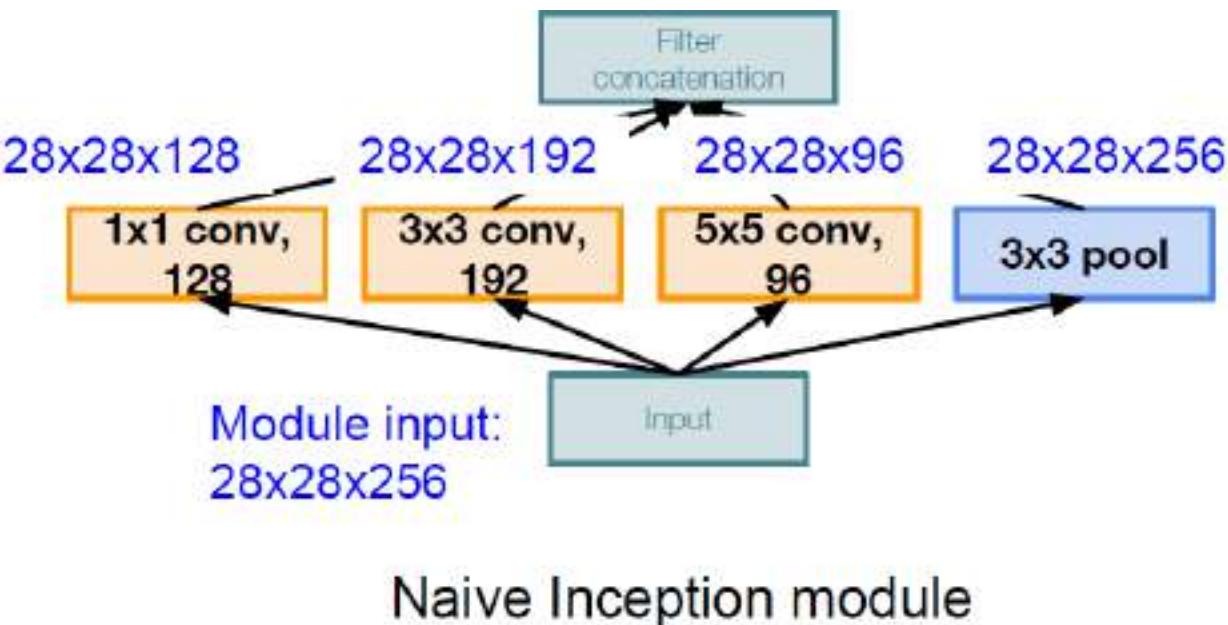
Problem:
Computational Complexity

Q3: What is output size after
filter concatenation?

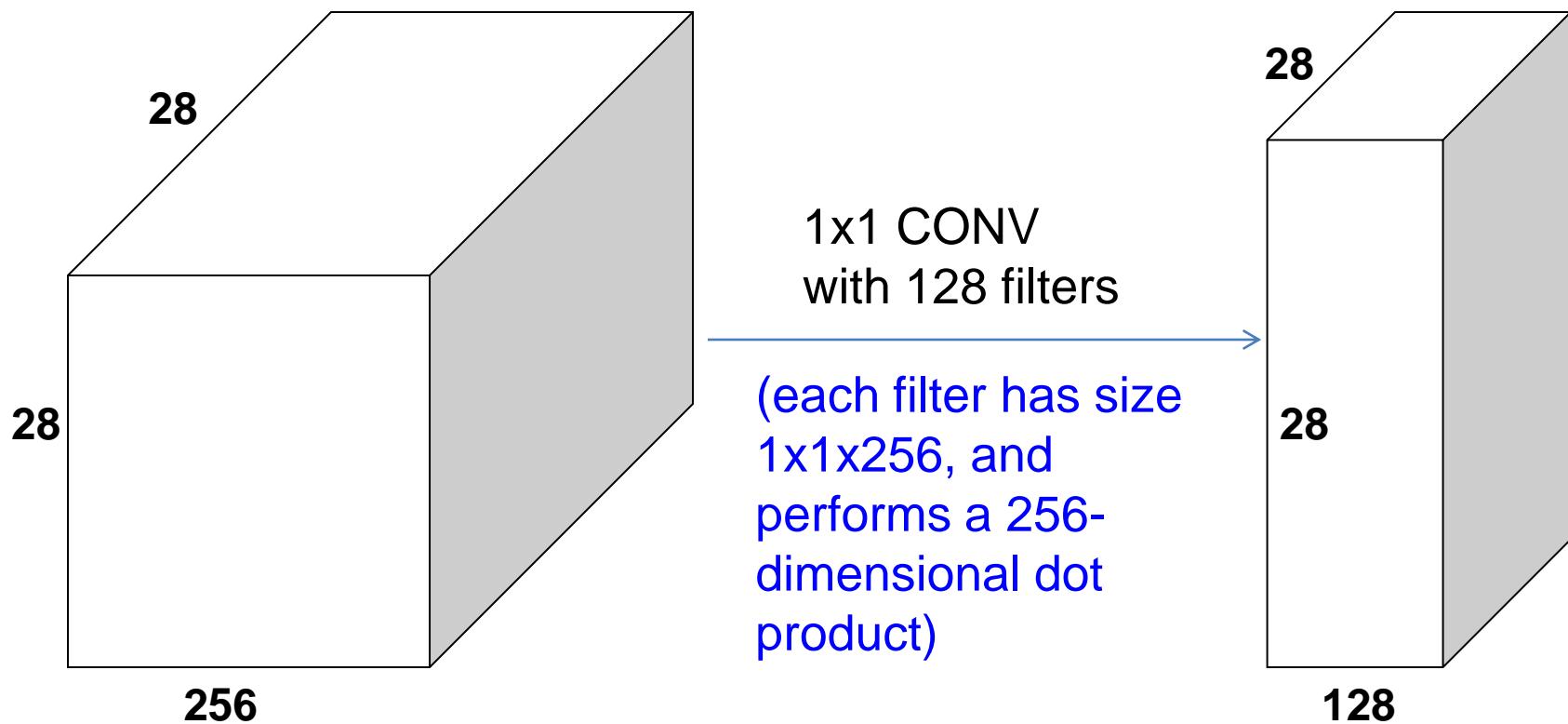
Example:

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672$$

Solution: “bottleneck”
layers that use 1×1
convolutions to reduce
feature depth

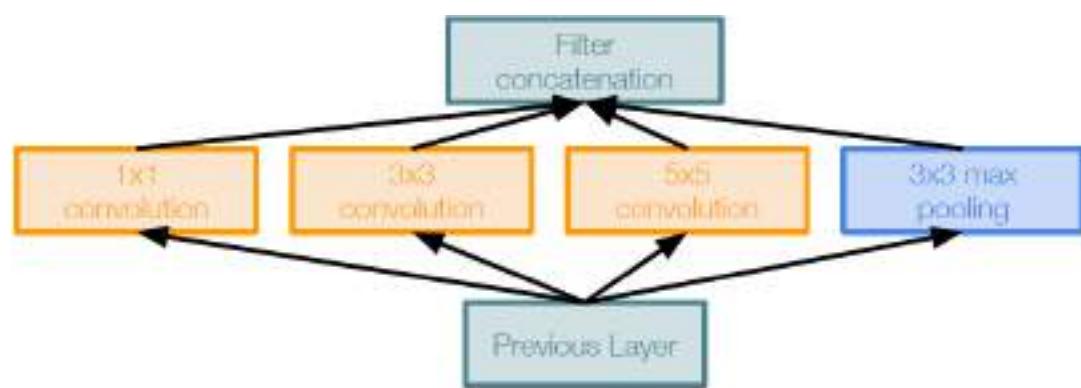


1×1 Convolutions



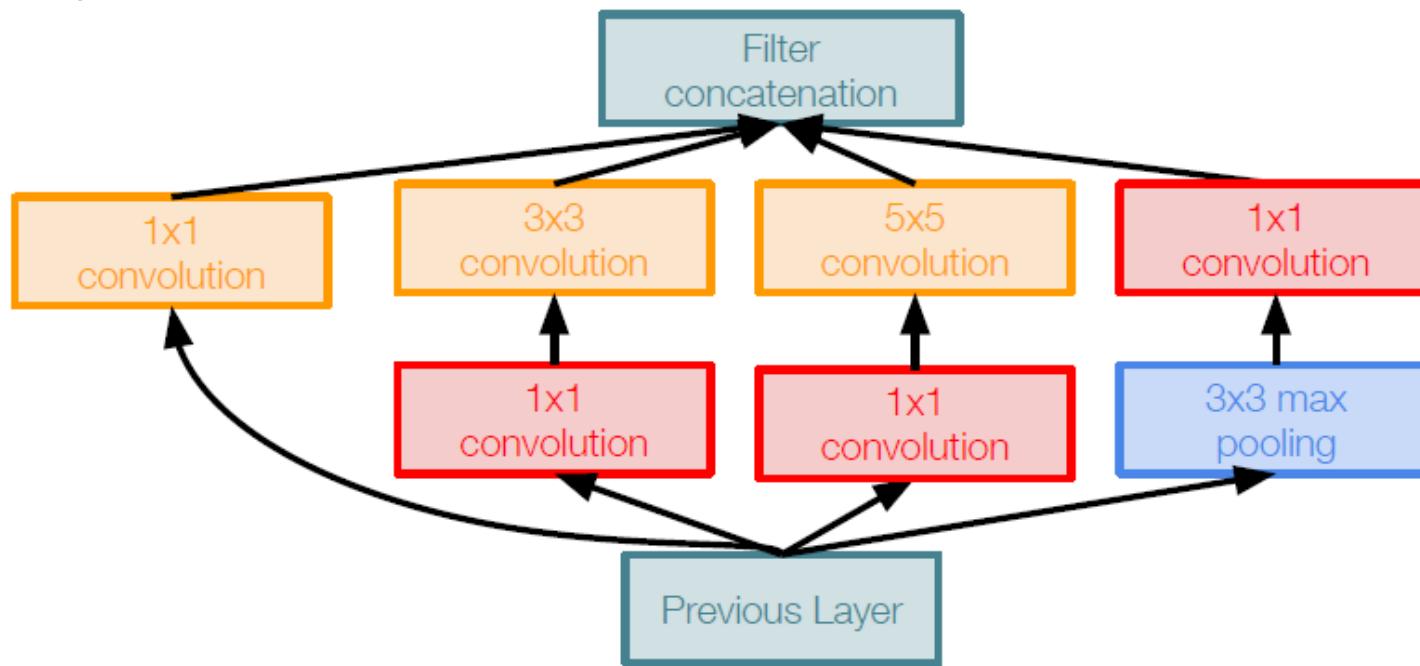
preserves spatial dimensions, reduces depth!
Projects depth to lower dimension (combination
of feature maps)

GoogLeNet



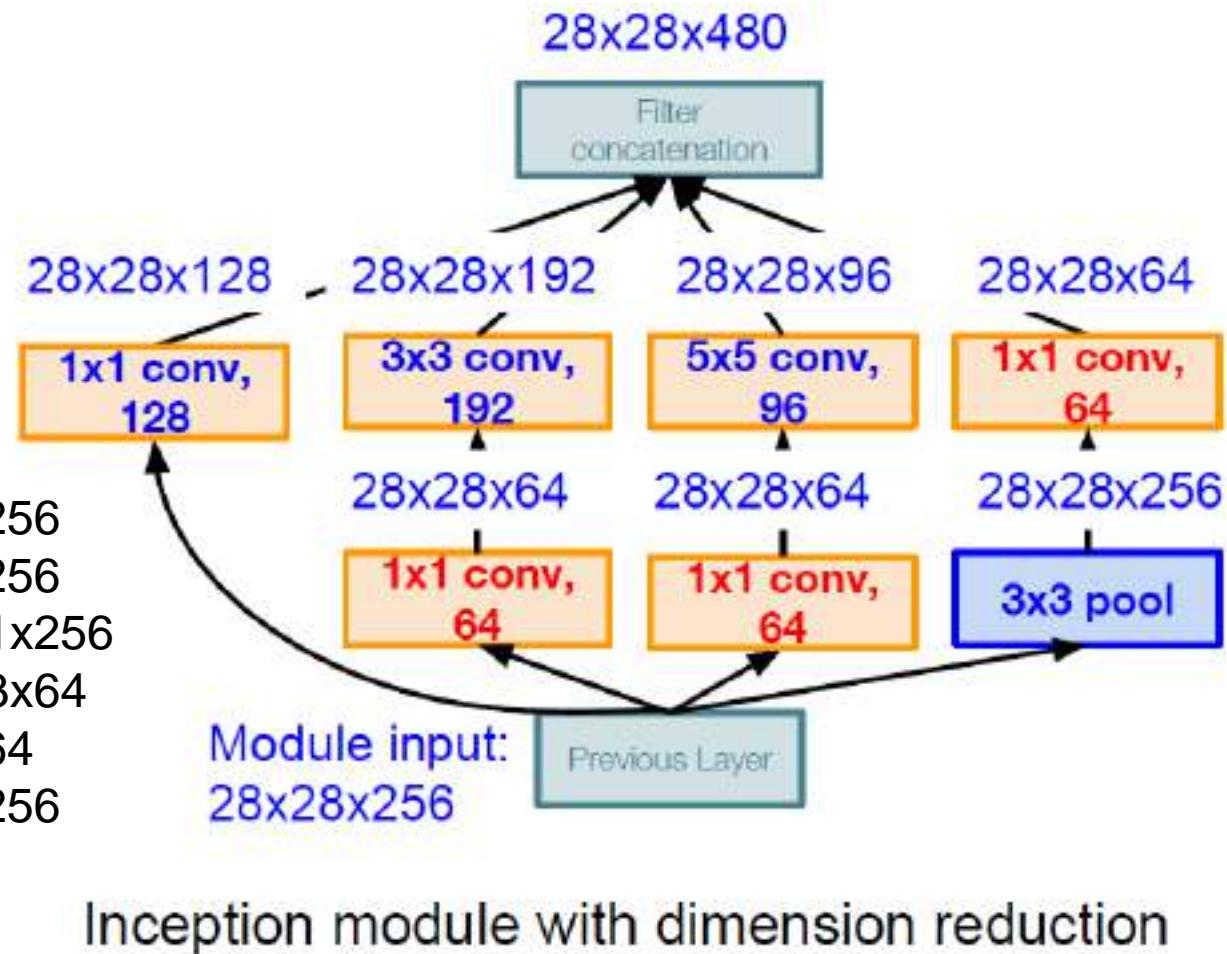
1x1 conv “bottleneck”
layers

Naive Inception module



Inception module with dimension reduction

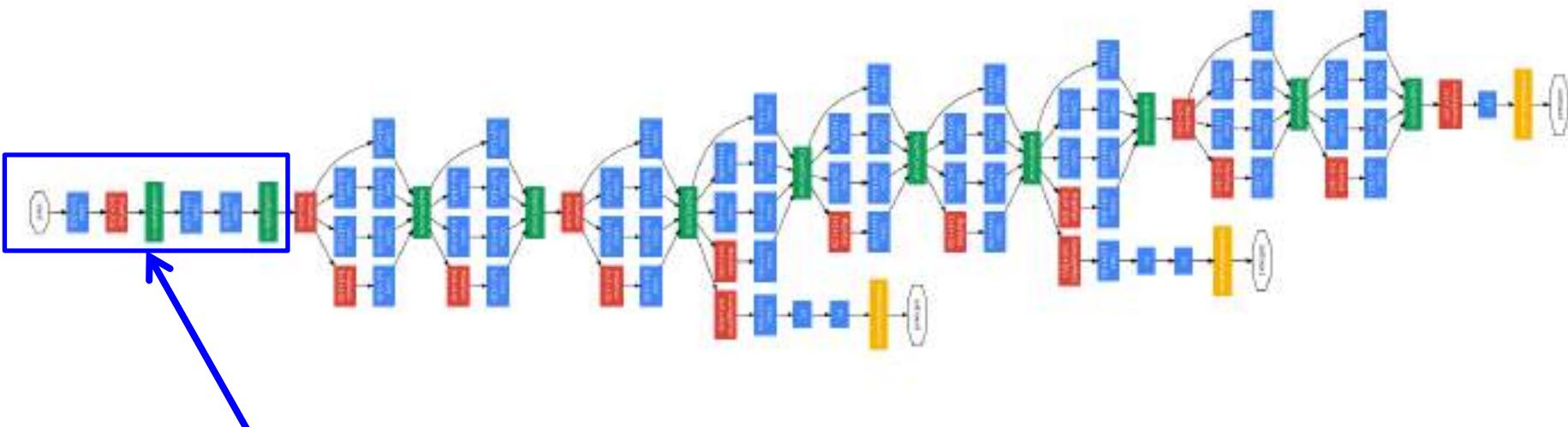
GoogLeNet



Compared to 854M ops for naive version, Bottleneck can also reduce depth after pooling layer

GoogLeNet

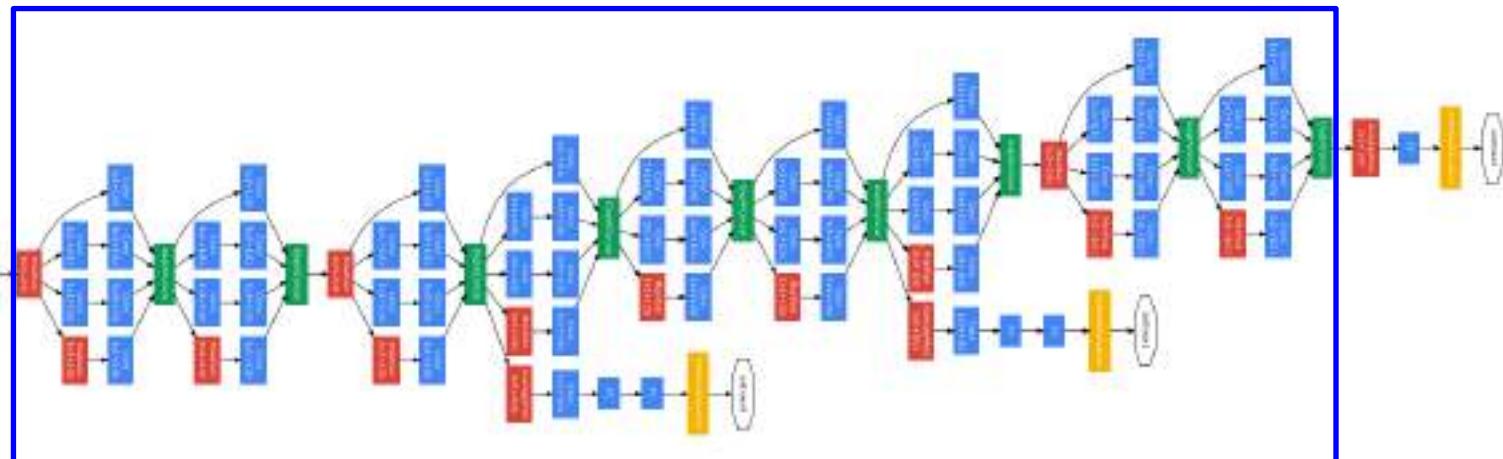
Full GoogLeNet Architecture



Stem Network:
Conv-Pool-
2x Conv-Pool

GoogLeNet

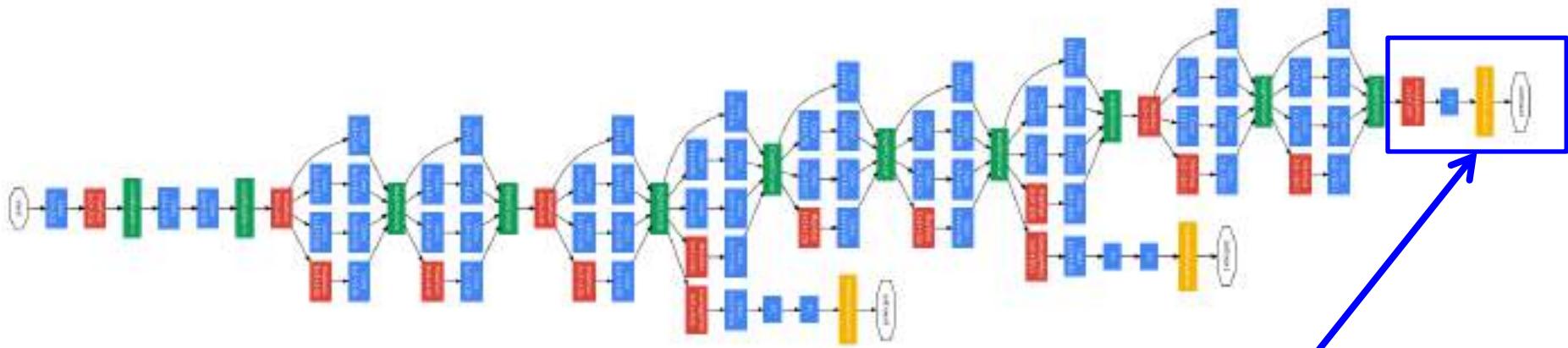
Full GoogLeNet Architecture



Stacked Inception
Modules

GoogLeNet

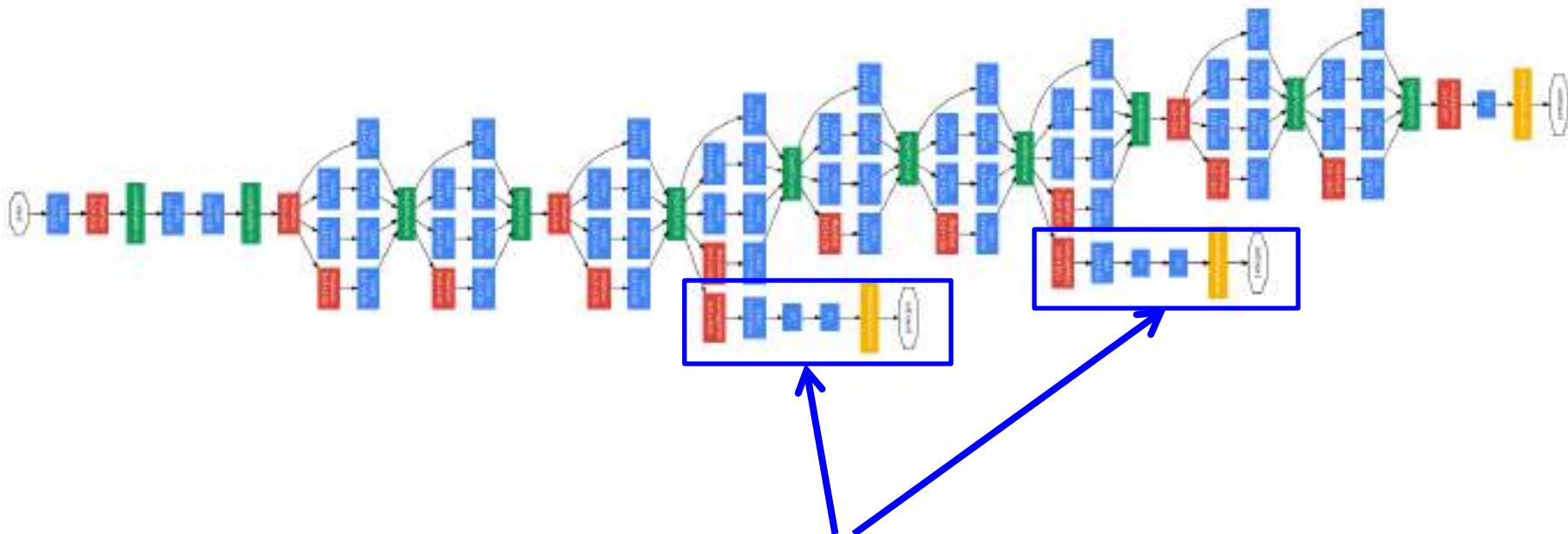
Full GoogLeNet Architecture



Classifier output
(removed expensive FC
layers!)

GoogLeNet

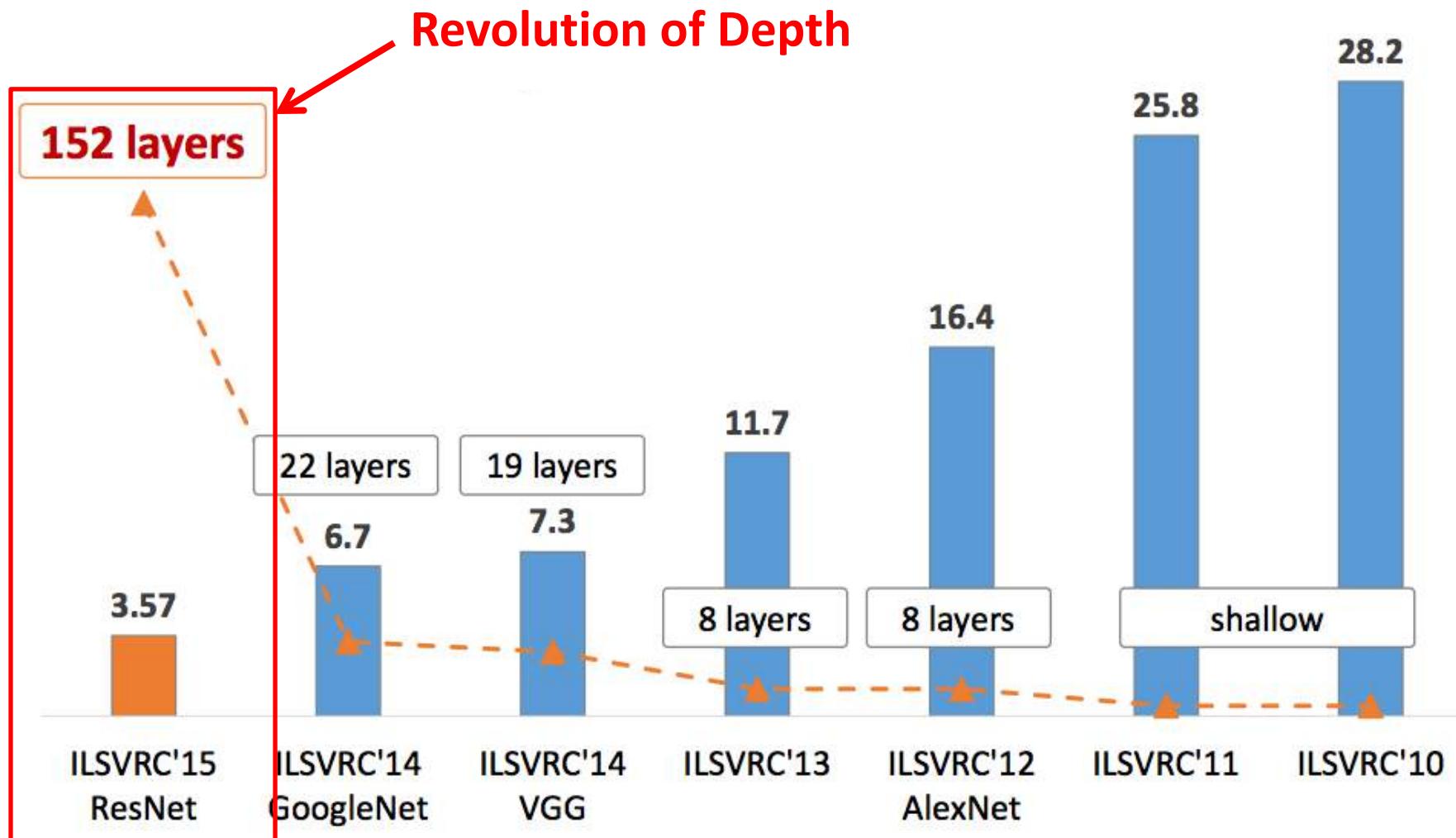
Full GoogLeNet Architecture



Auxiliary classification outputs to inject additional gradient at
lower layers

(AvgPool-1x1Conv-FC-FC-Softmax)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



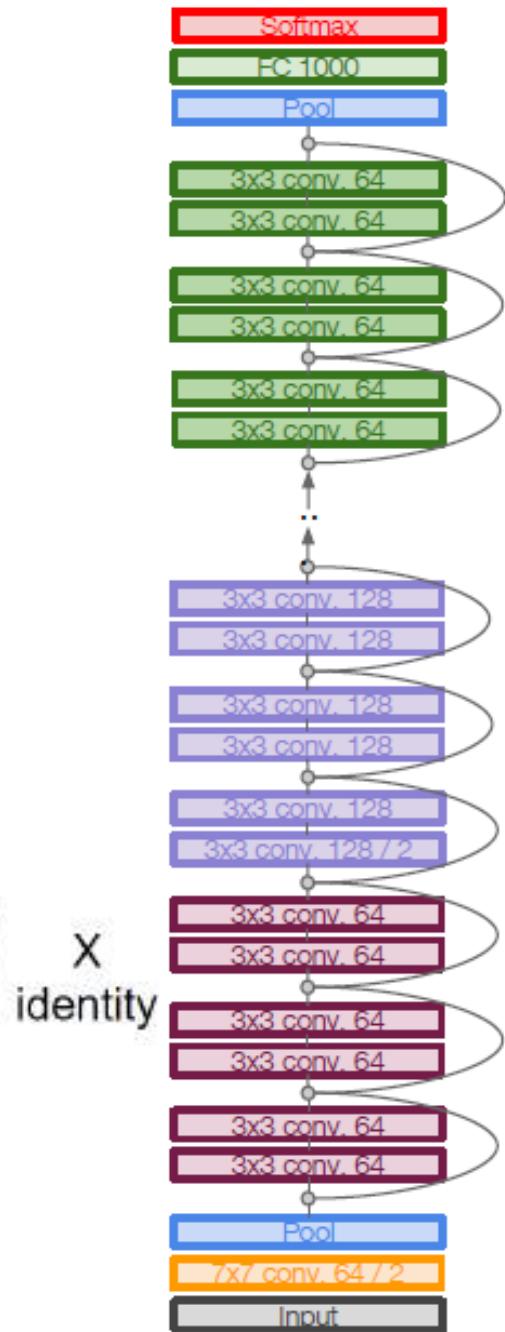
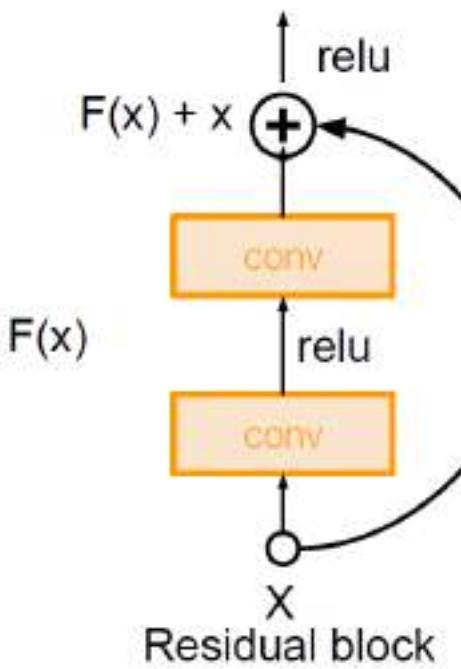
ResNet

Very deep networks using residual connections

- 152-layer model for ImageNet

- ILSVRC'15 classification winner
(3.57% top 5 error)

- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

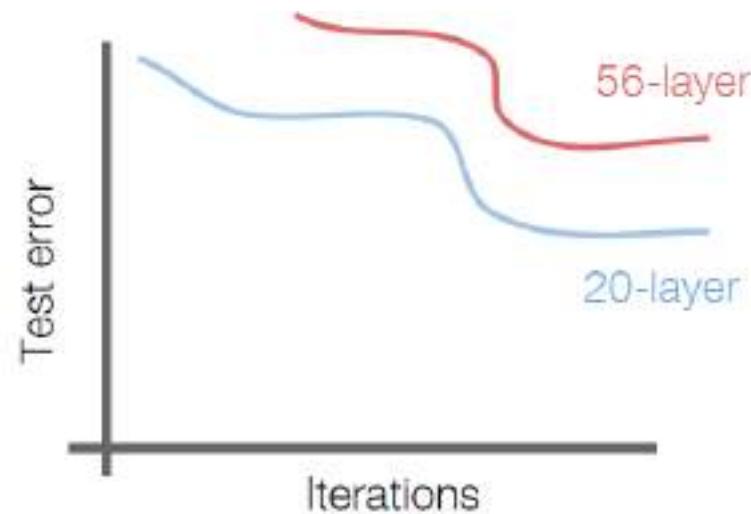
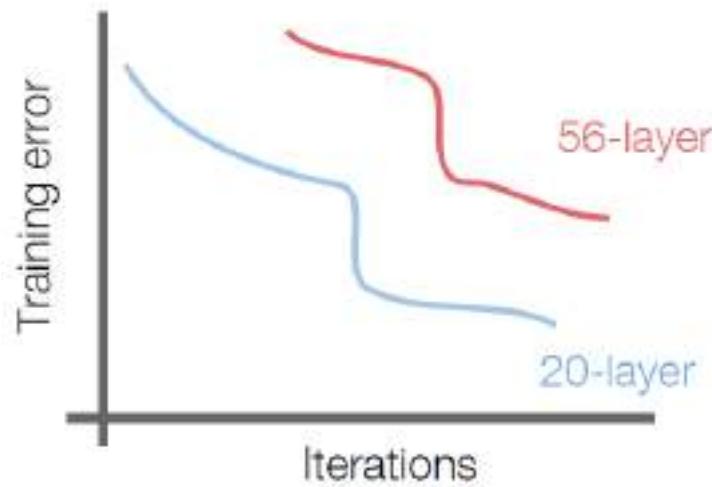


ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it's not caused by overfitting!

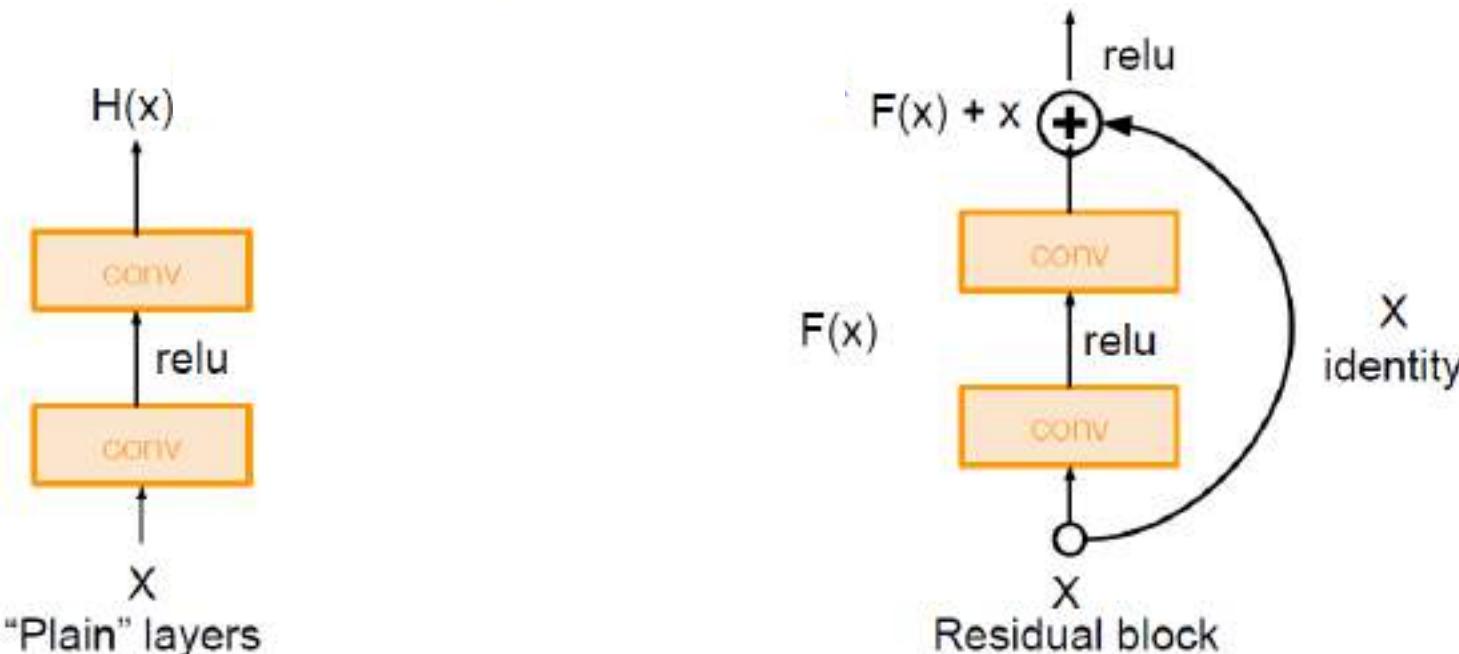
ResNet

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

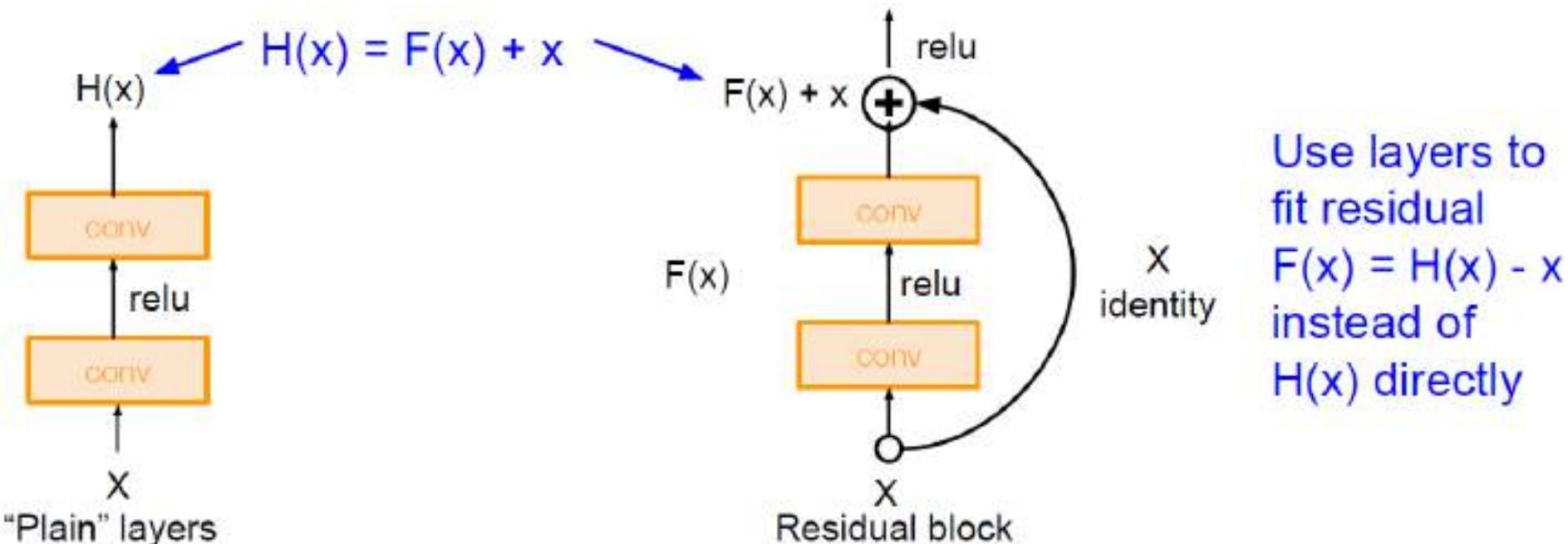
ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet

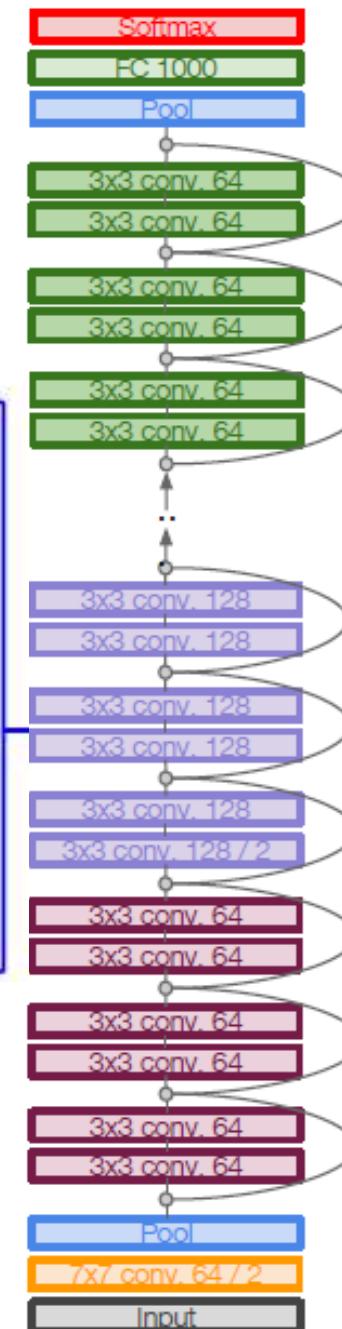
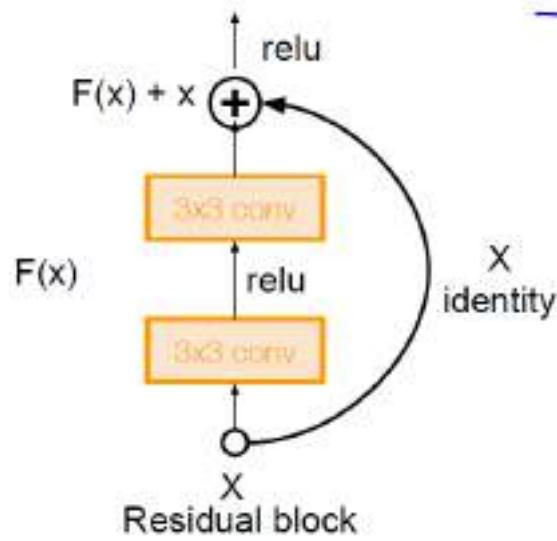
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ResNet

Full ResNet architecture:

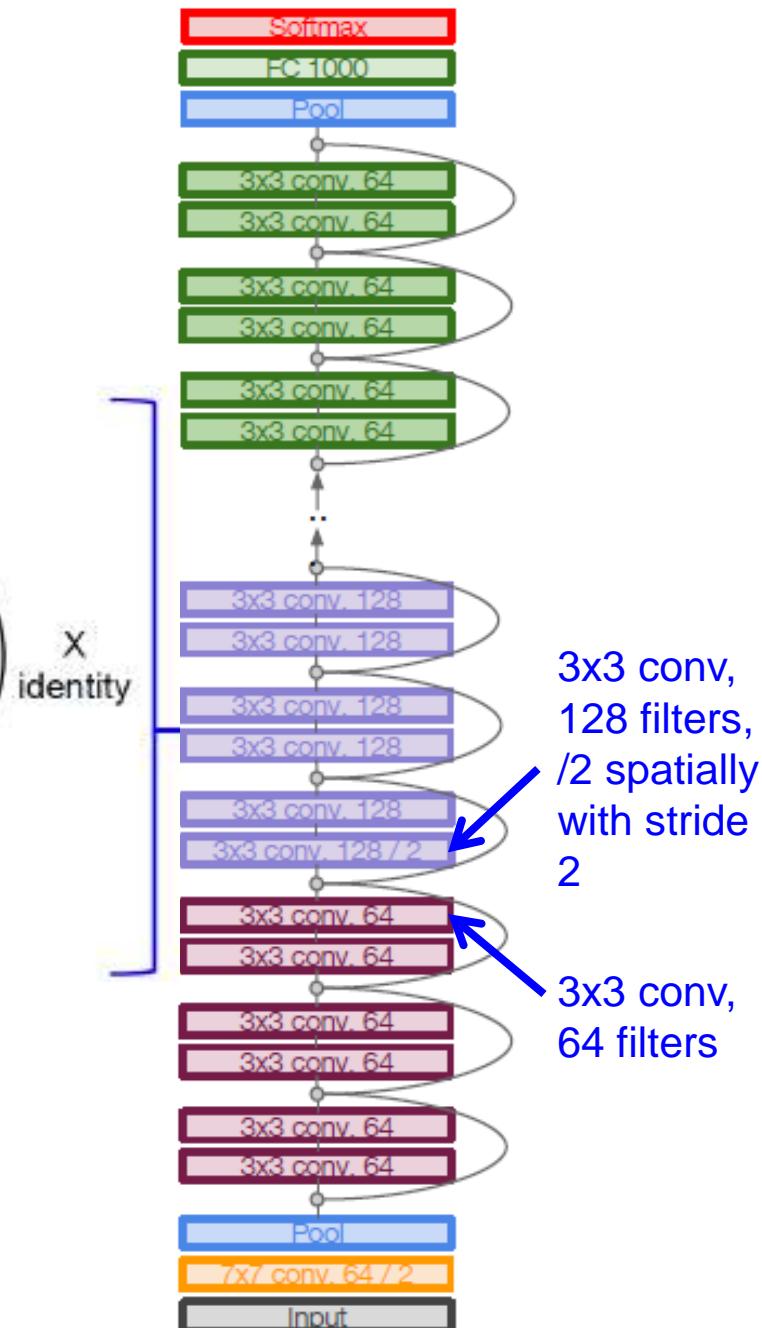
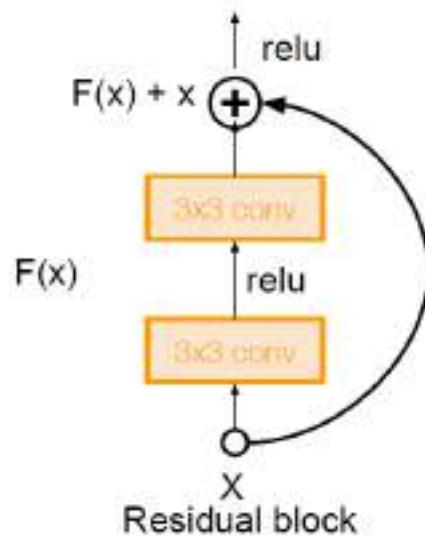
- Stack residual blocks
- Residual block has two 3x3 conv layers



ResNet

Full ResNet architecture:

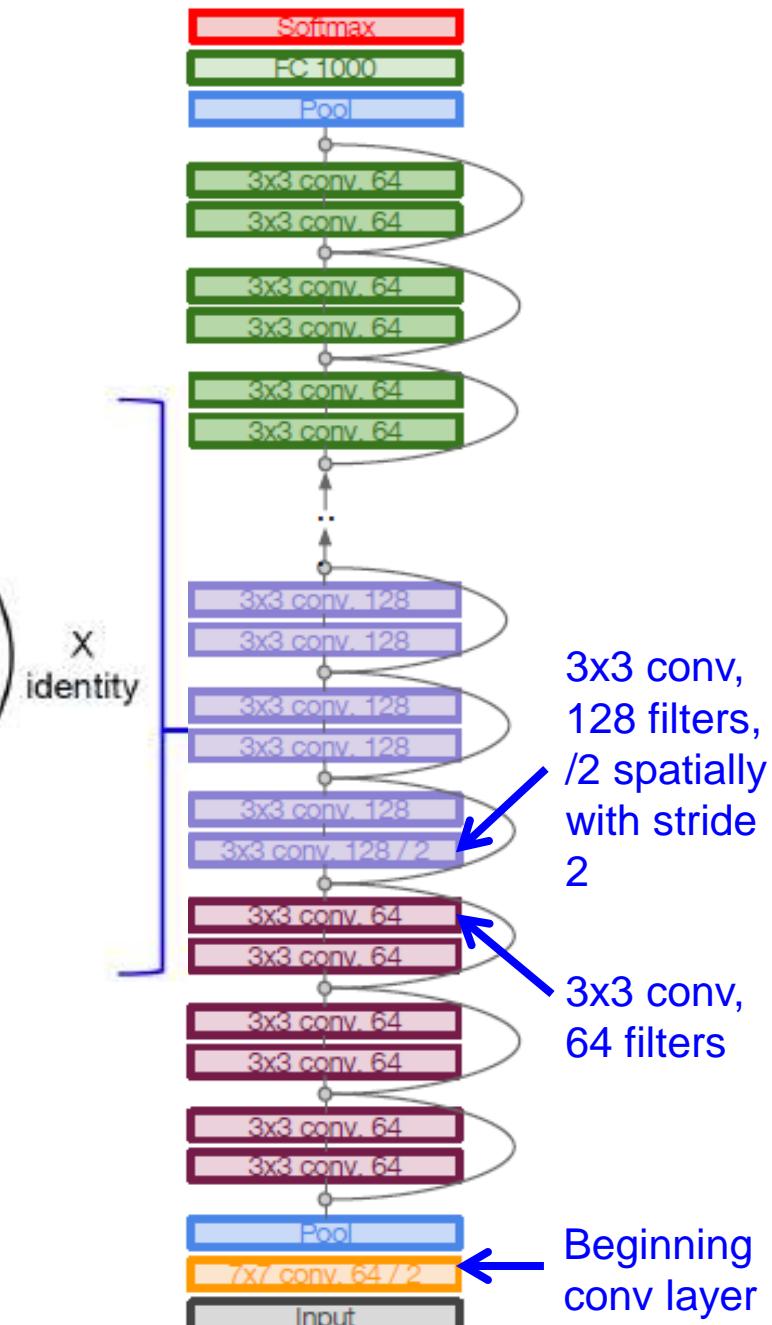
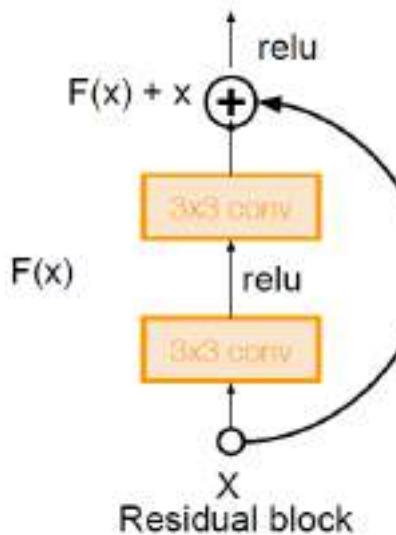
- Stack residual blocks
- Residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



ResNet

Full ResNet architecture:

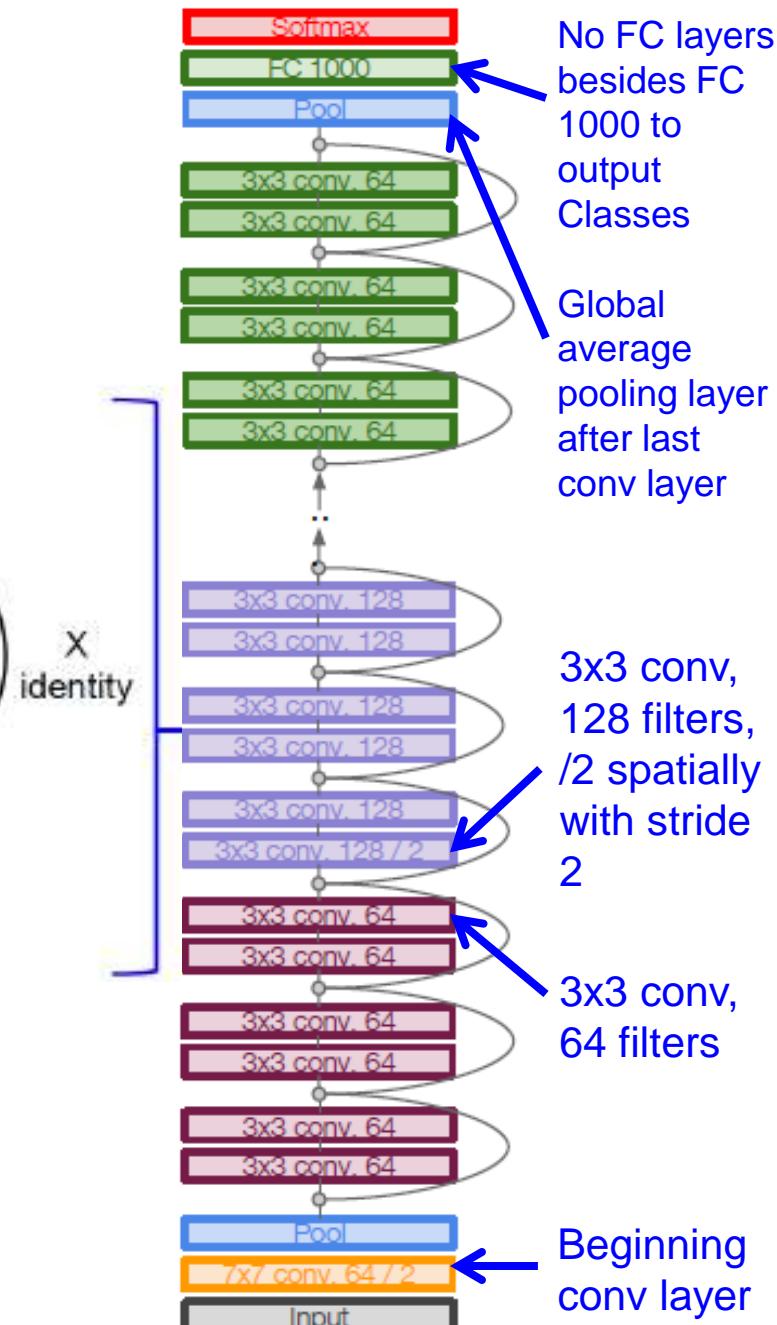
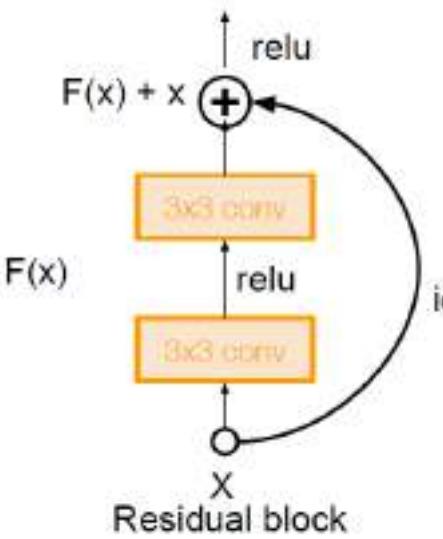
- Stack residual blocks
- Residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



ResNet

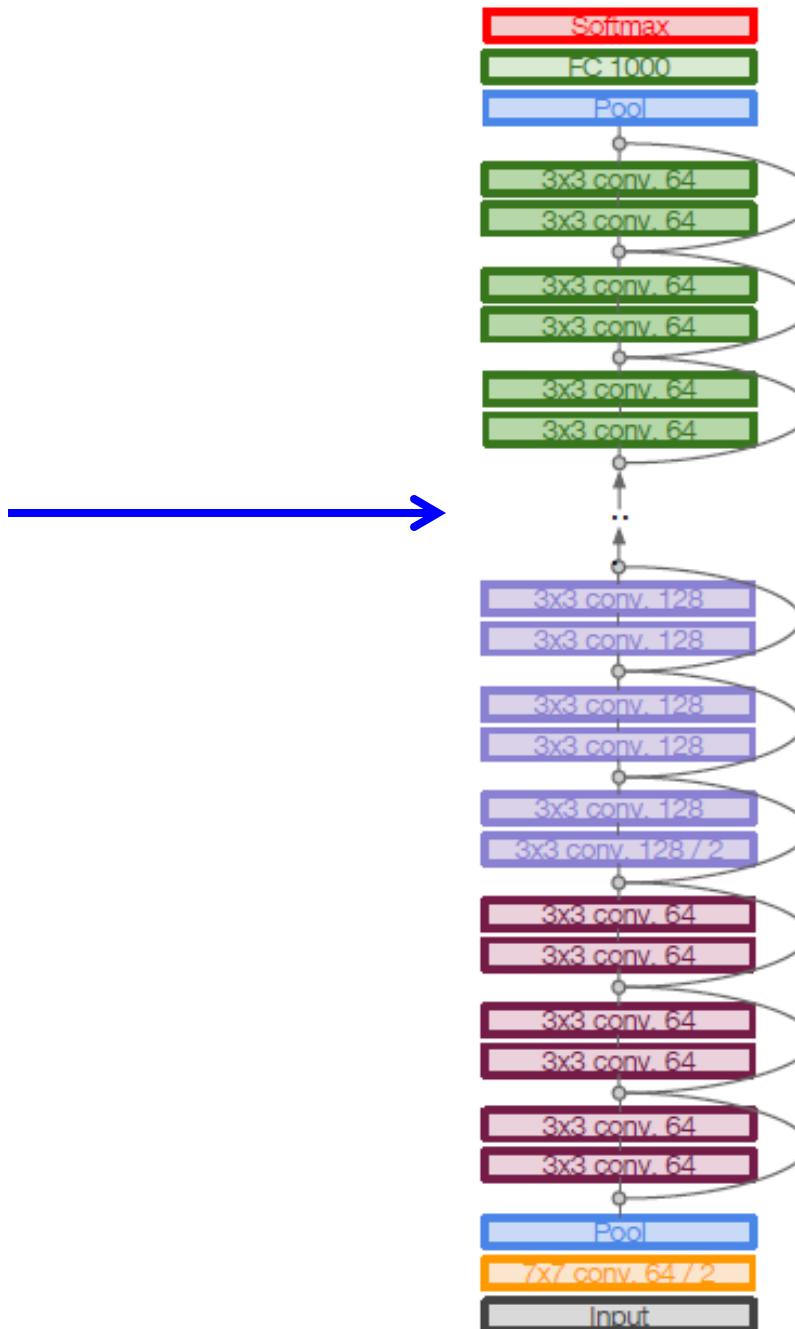
Full ResNet architecture:

- Stack residual blocks
- Residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



ResNet

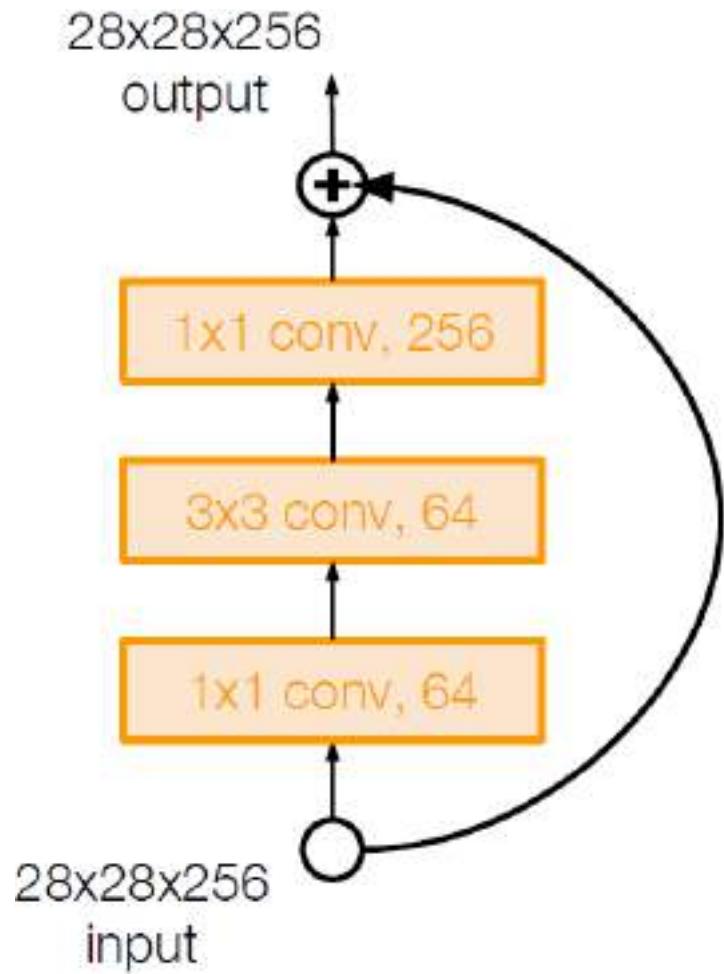
Total depths of 34, 50, 101, or 152 layers for ImageNet



ResNet

For deeper networks (ResNet-50+):

use “bottleneck” layer to improve efficiency
(similar to GoogLeNet)



ResNet

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error saturates
- Mini-batch size 256
- Weight decay of 1e-5 for penalizing regularization term
- No dropout used

ResNet

Experimental Results:

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

ResNet

Experimental Results:

- **1st places in all five main tracks**

- ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

ResNet

Experimental Results:

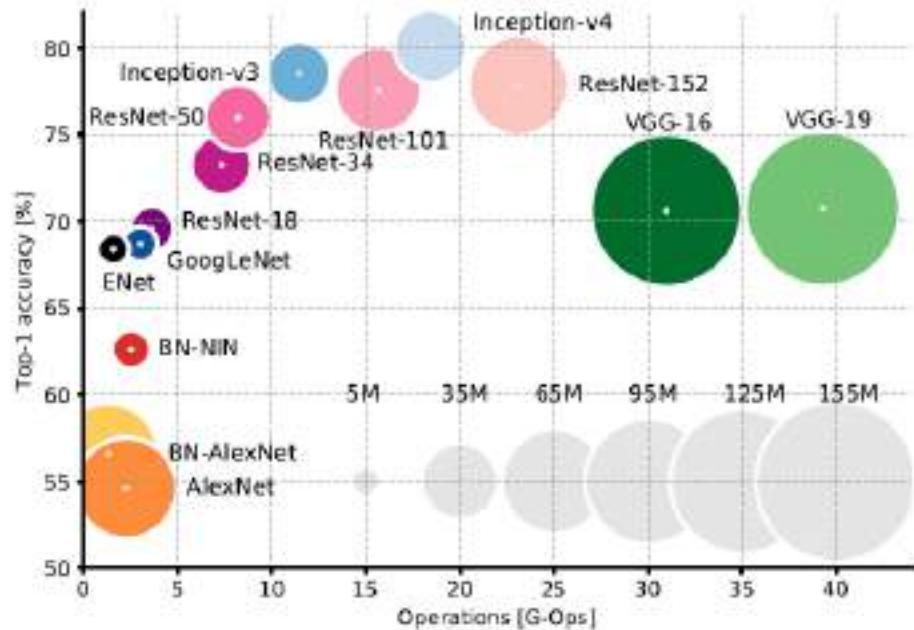
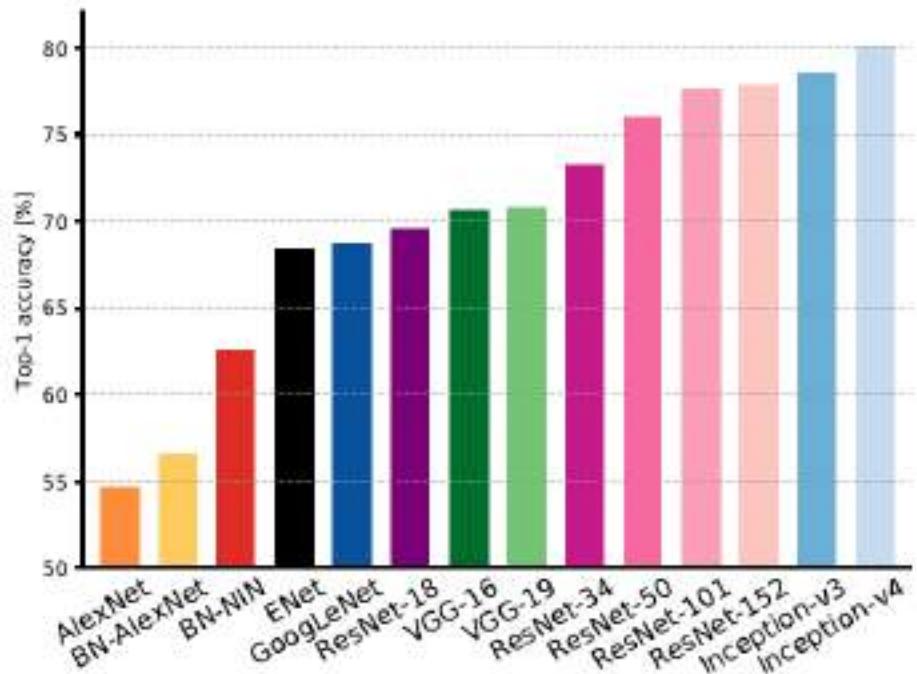
- **1st places in all five main tracks**

- ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

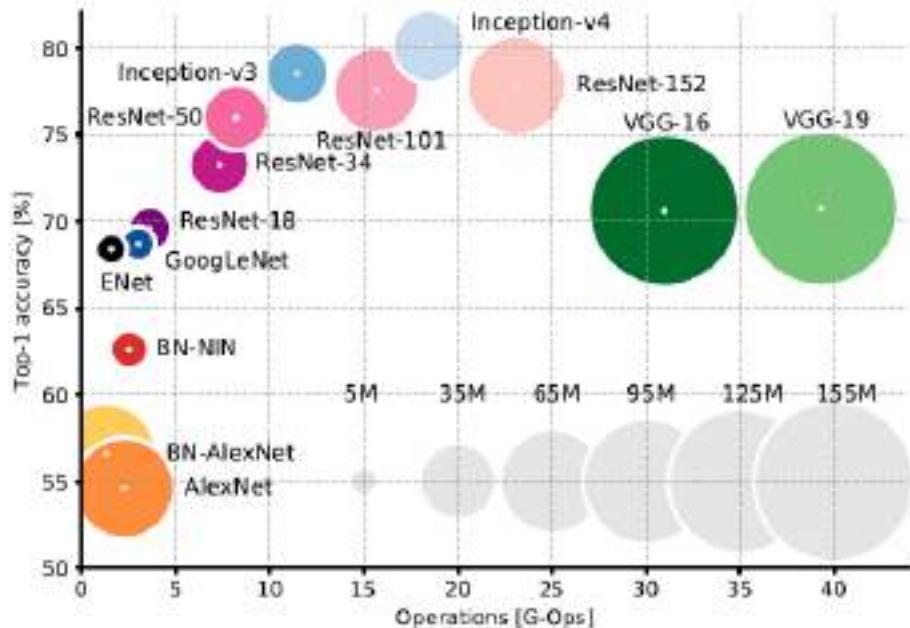
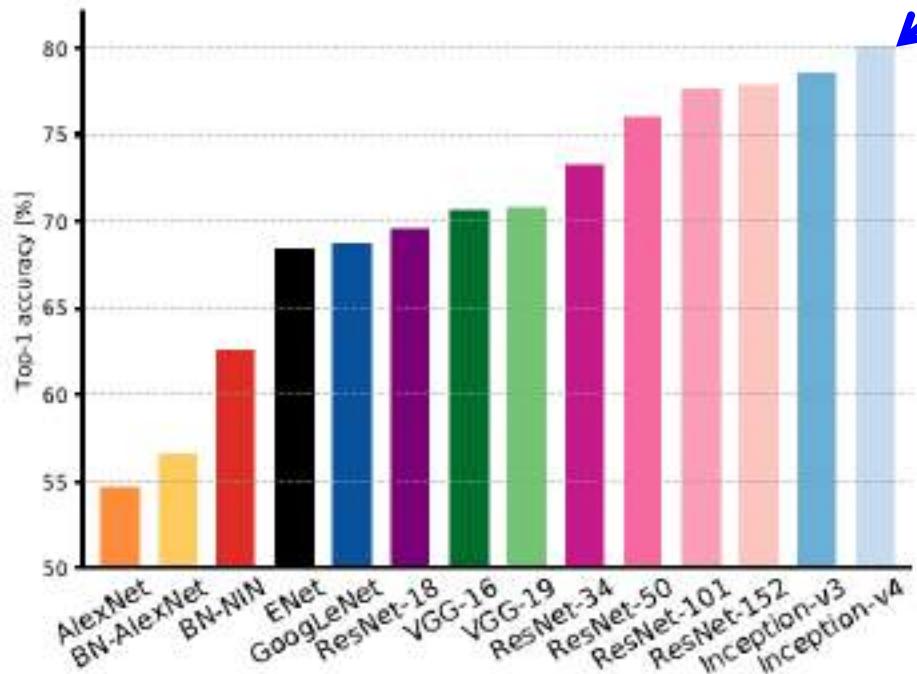
ILSVRC 2015 classification winner
(3.6% top 5 error) -- better than
“human performance”!
(Russakovsky 2014)

Comparing Complexity ...



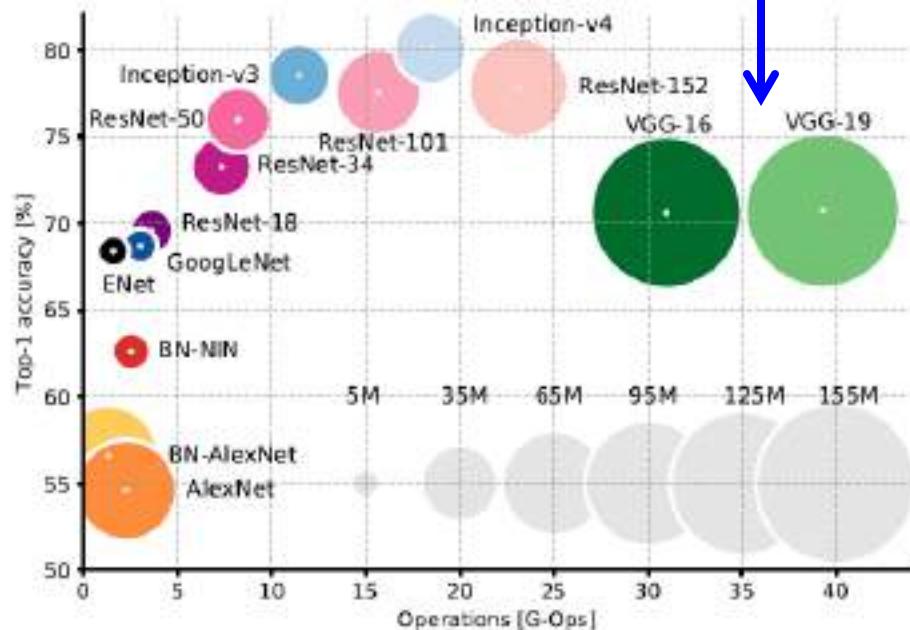
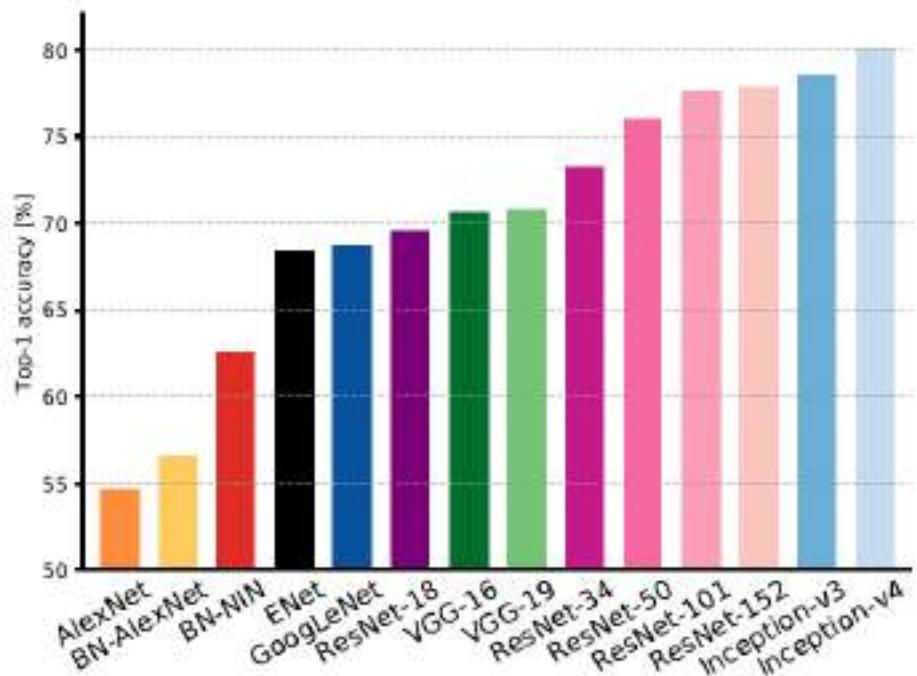
Comparing Complexity ...

Inception-v4: Resnet + Inception!



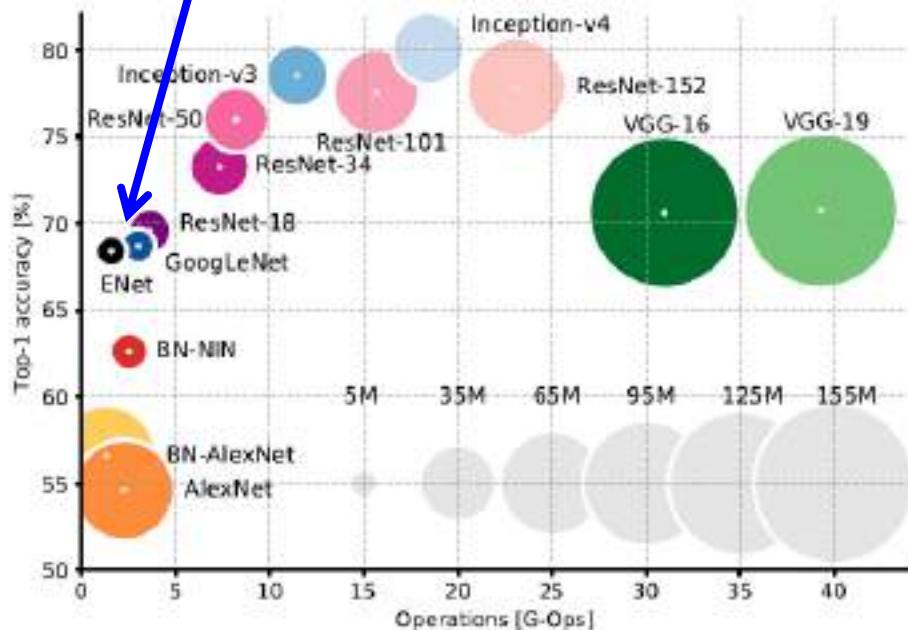
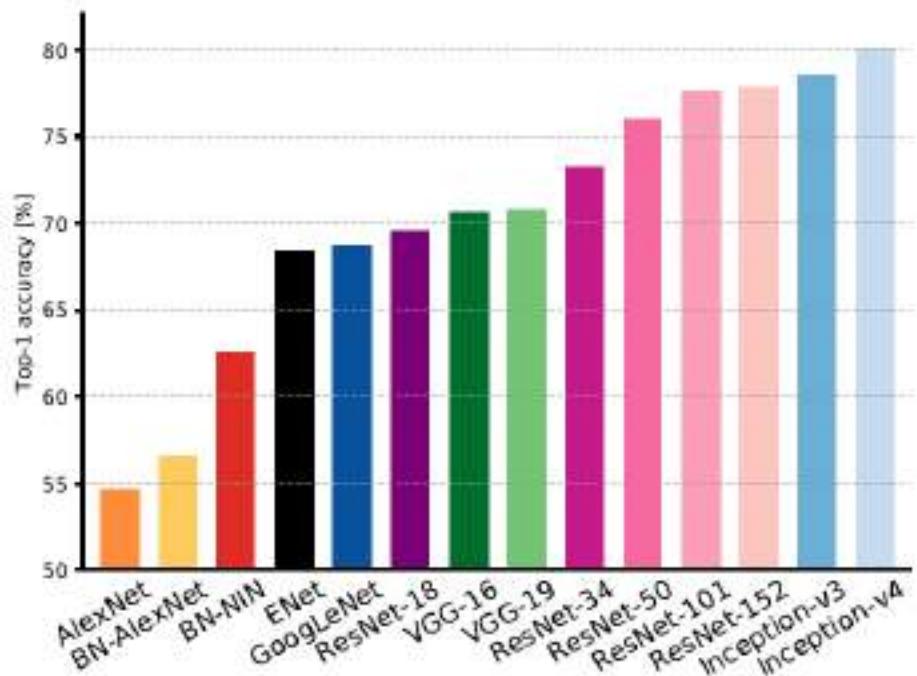
Comparing Complexity ...

VGG: Highest memory, most operations



Comparing Complexity ...

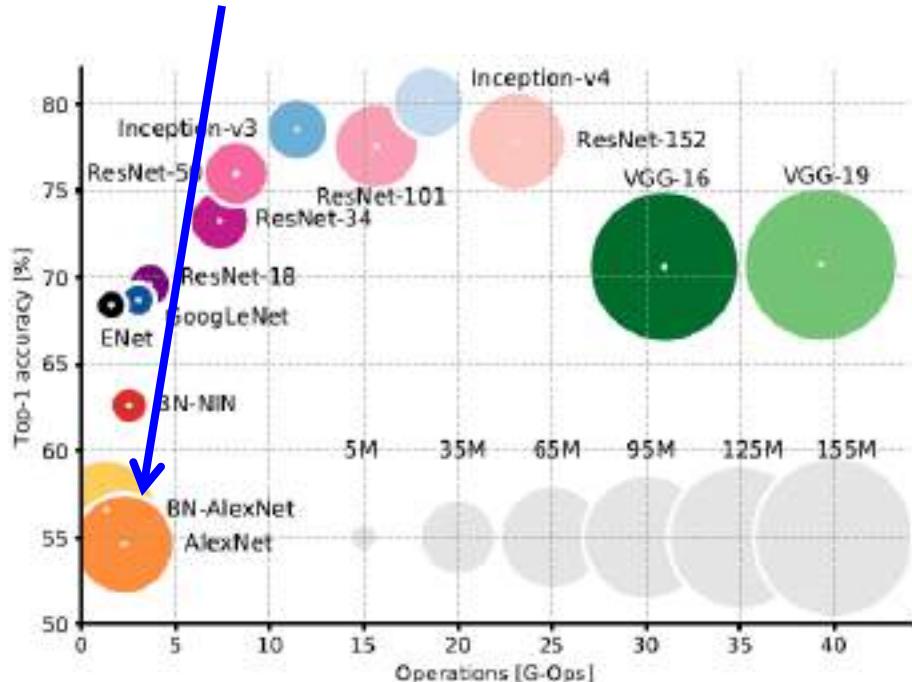
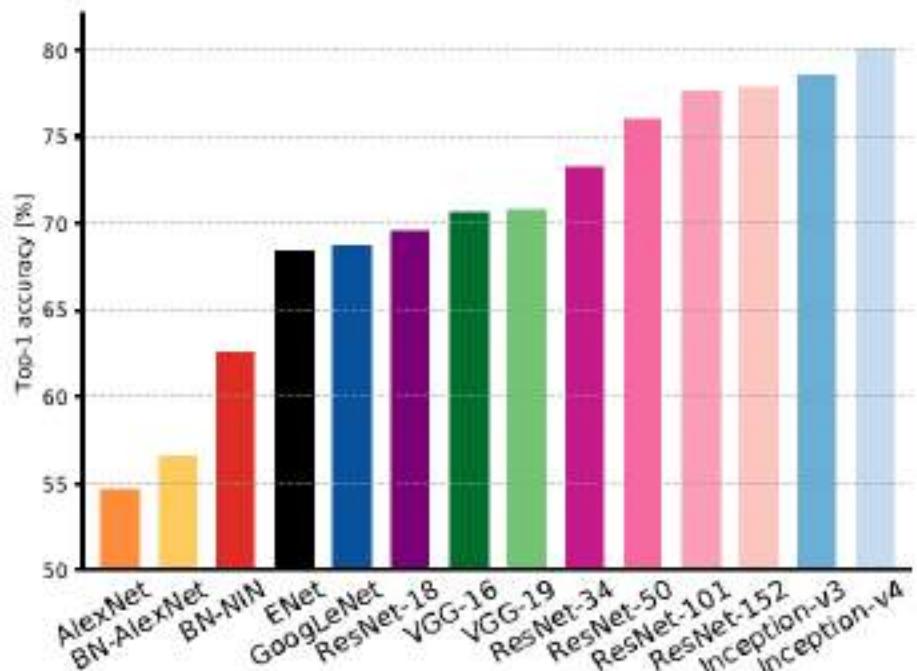
GoogLeNet:
most efficient



Comparing Complexity ...

AlexNet:

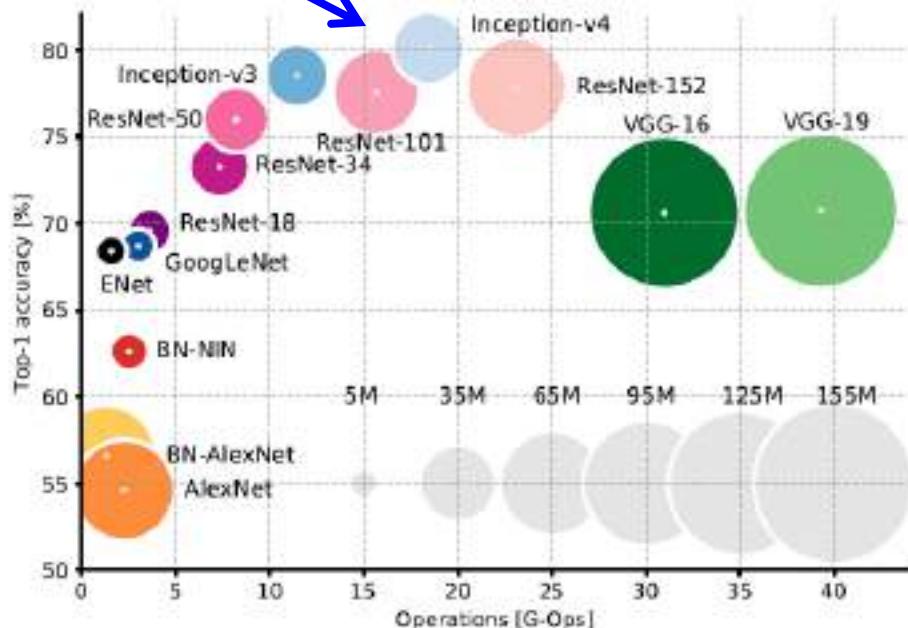
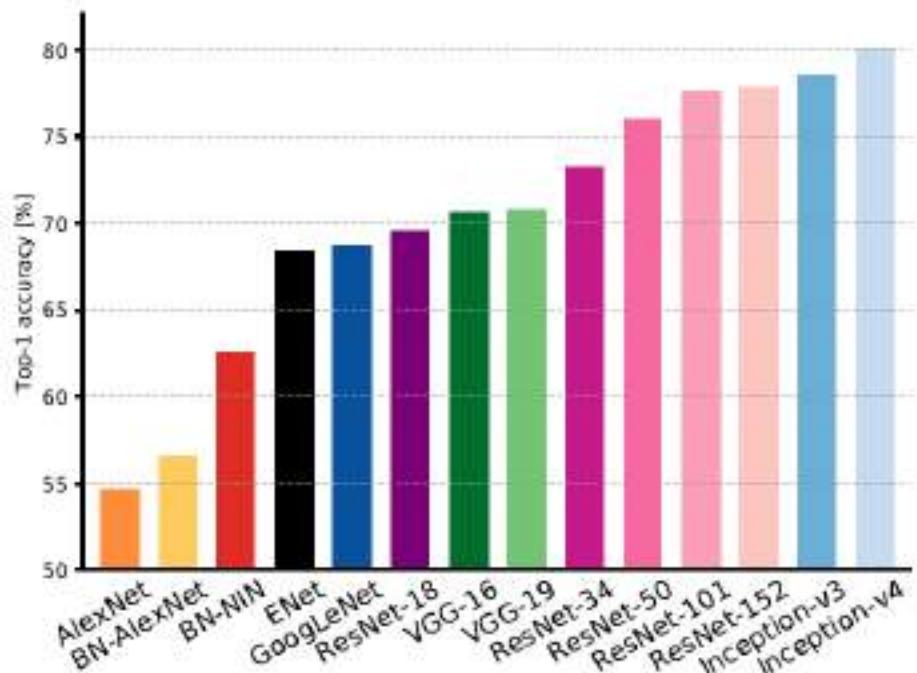
Smaller compute, still memory heavy, lower accuracy



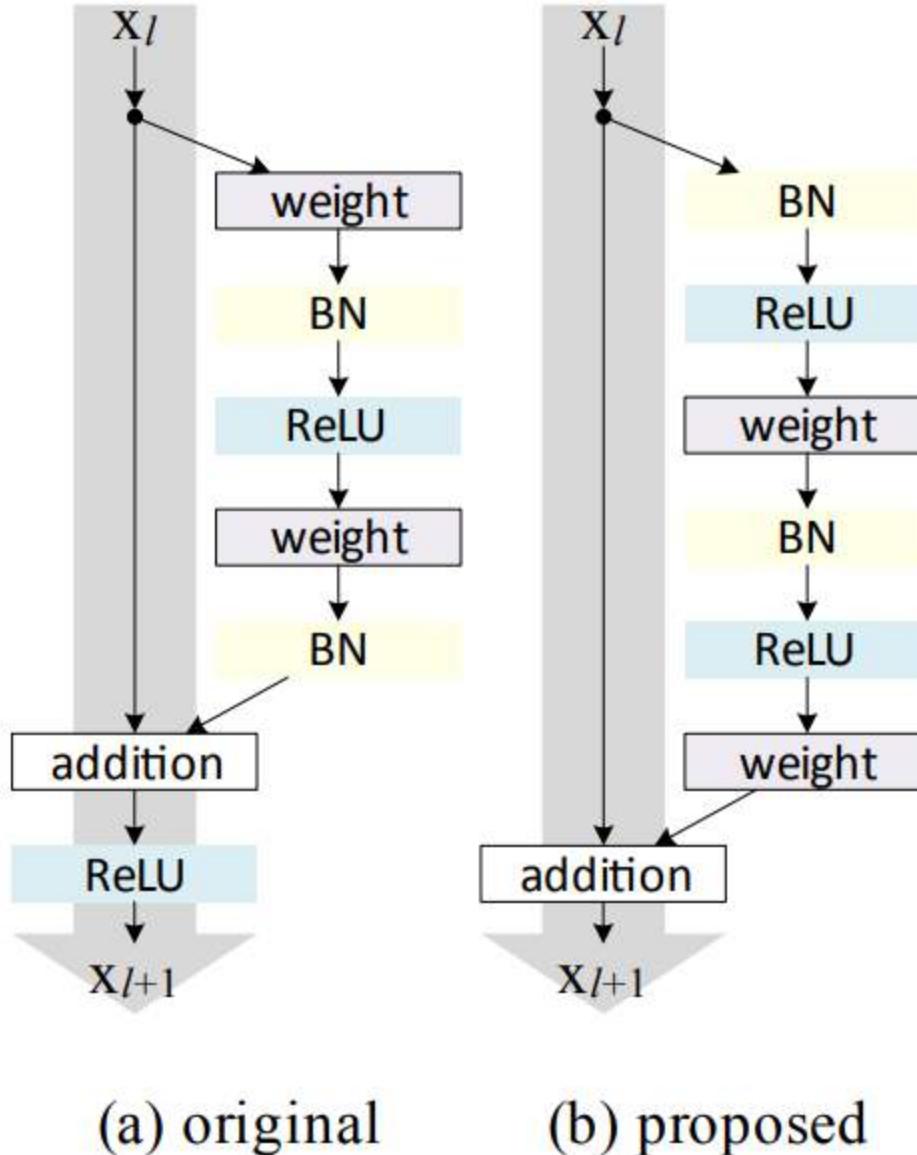
Comparing Complexity ...

ResNet:

Moderate efficiency depending on model, highest accuracy



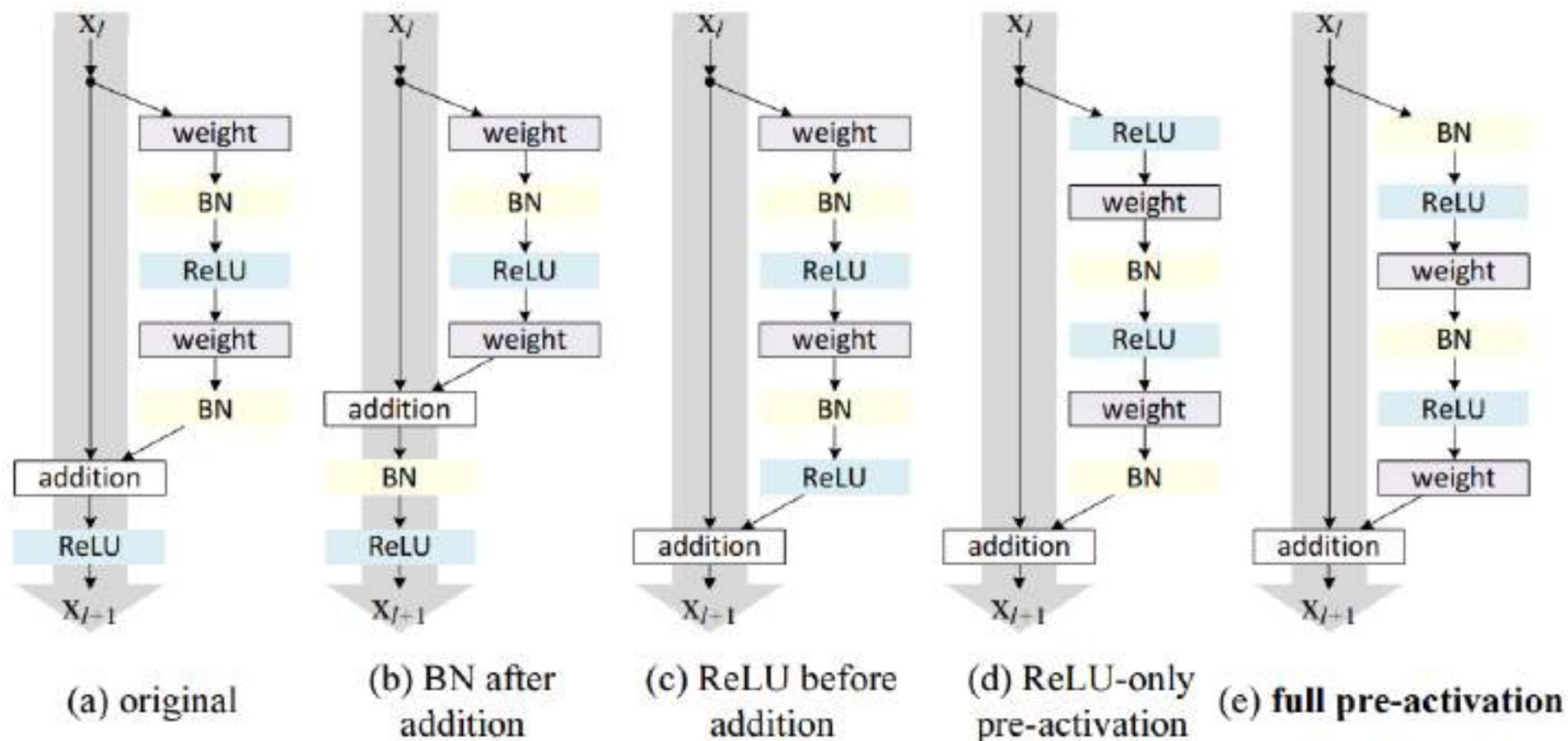
Pre-activated ResNet



(a) original

(b) proposed

Pre-activated ResNet



(a) original

(b) BN after
addition

(c) ReLU before
addition

(d) ReLU-only
pre-activation (e) full pre-activation

Pre-activated ResNet

Classification error (%) on the CIFAR-10 test set using different activation functions.

case	ResNet-110	ResNet-164
original Residual Unit [1]	6.61	5.93
BN after addition	8.17	6.50
ReLU before addition	7.84	6.14
ReLU-only pre-activation	6.71	5.91
full pre-activation	6.37	5.46

SENet (Squeeze and Excitation Network)

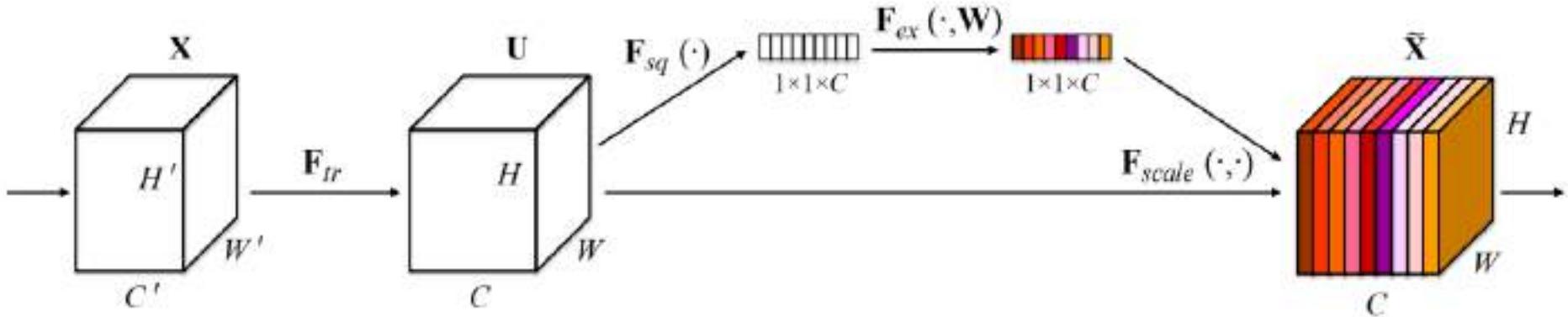
2017 ImageNet Challenge Winner

Top-5 Error: 2.251%

SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%

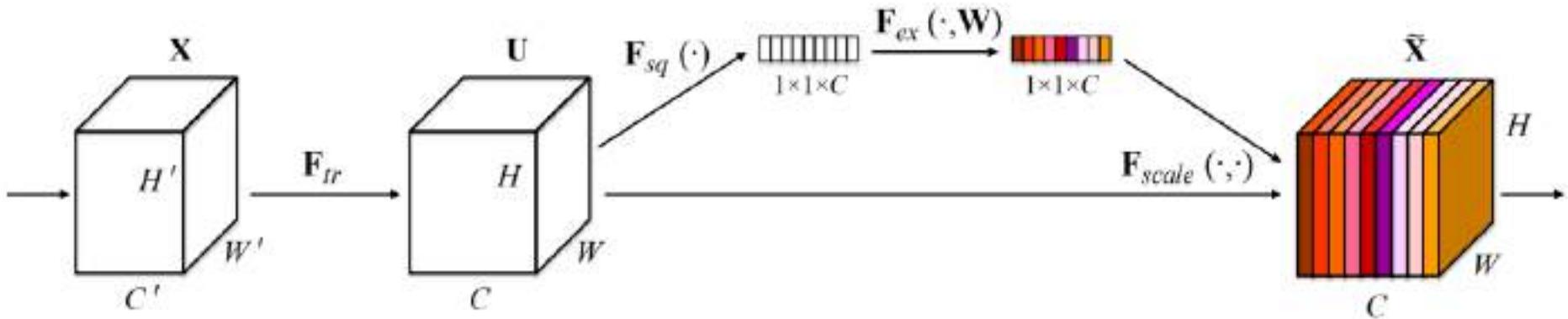


“Squeeze-and-Excitation”(SE) block adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%



Squeeze: Average Global Pooling

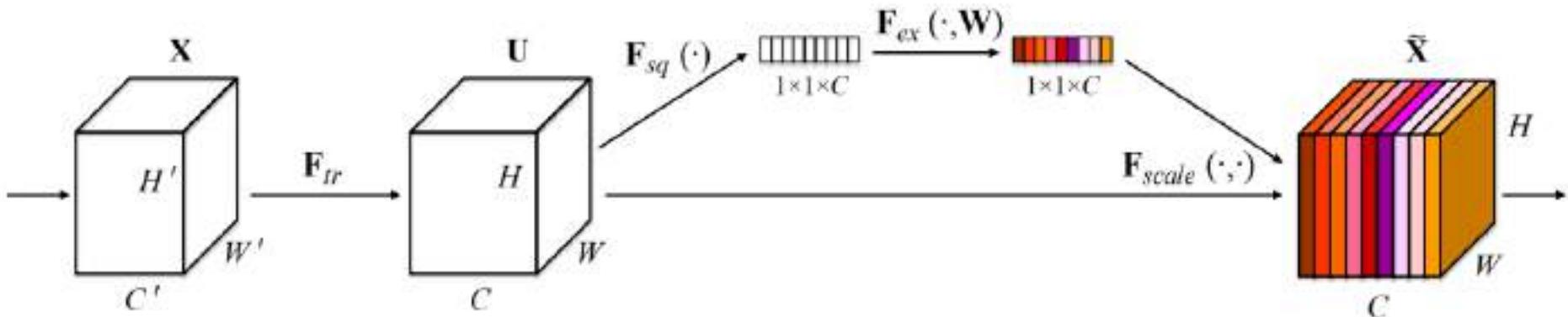
$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{W \times H} \sum_{i=1}^W \sum_{j=1}^H u_c(i, j)$$

cth channel of C

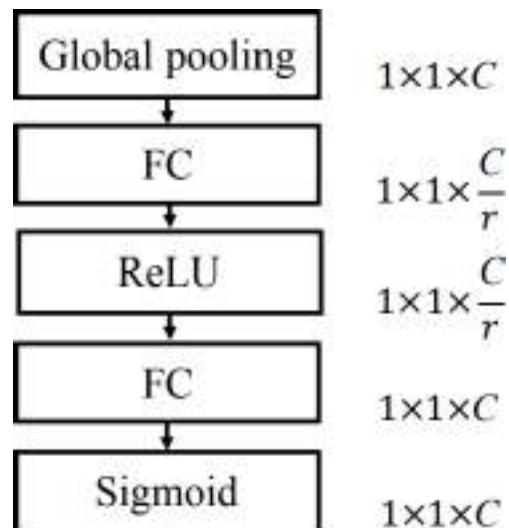
SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

Top-5 Error: 2.251%



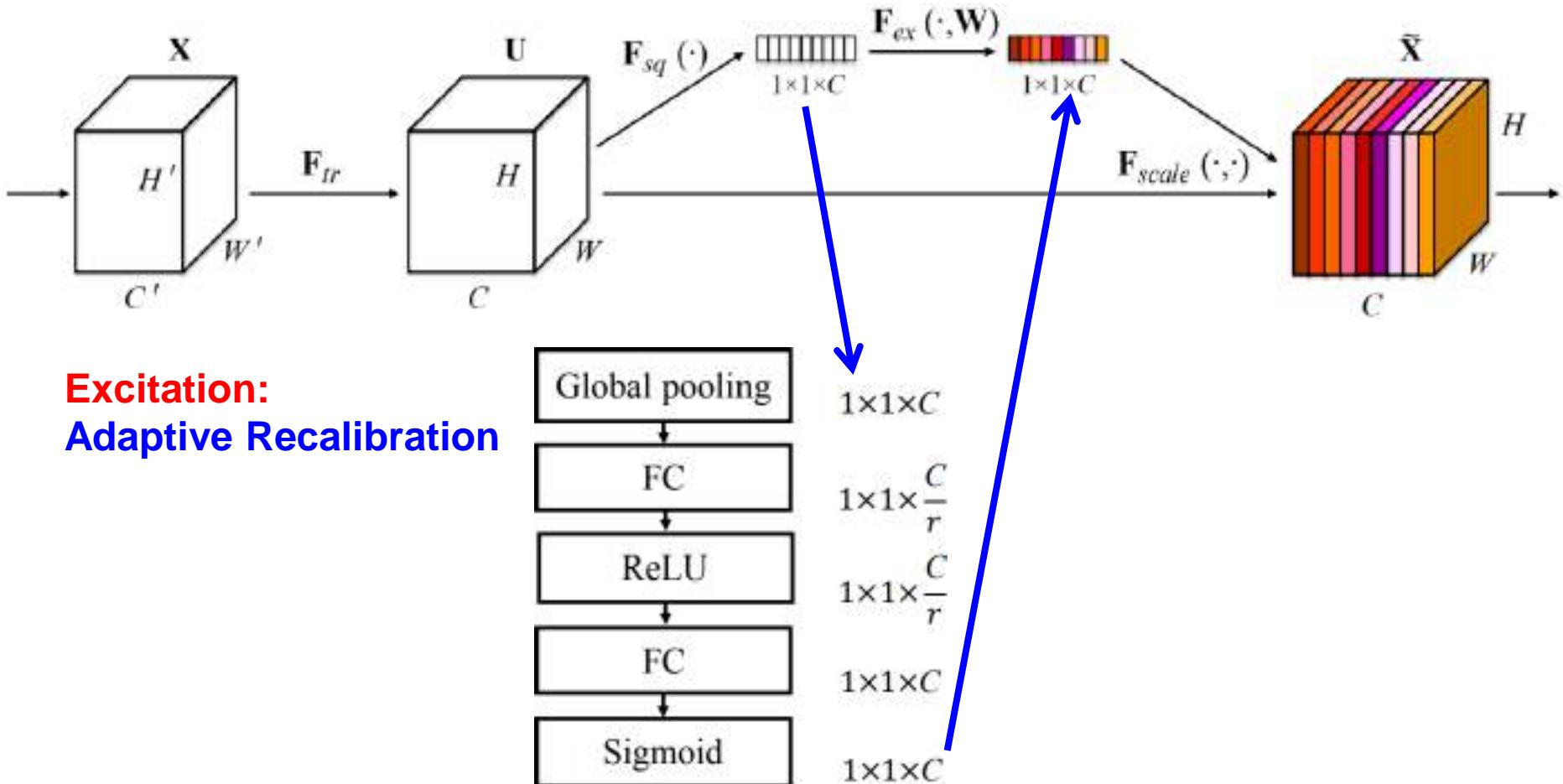
Excitation:
Adaptive Recalibration



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

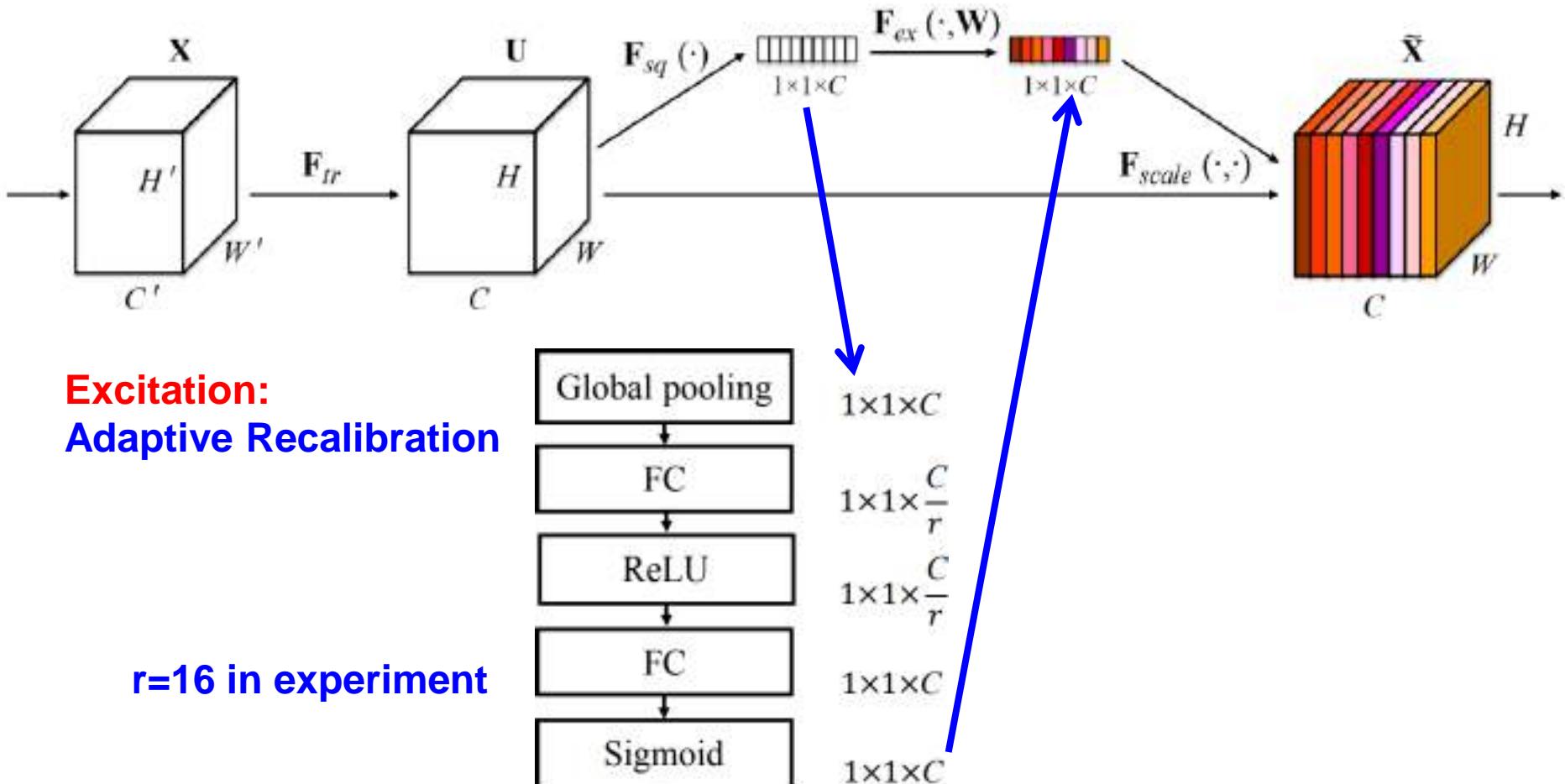
Top-5 Error: 2.251%



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

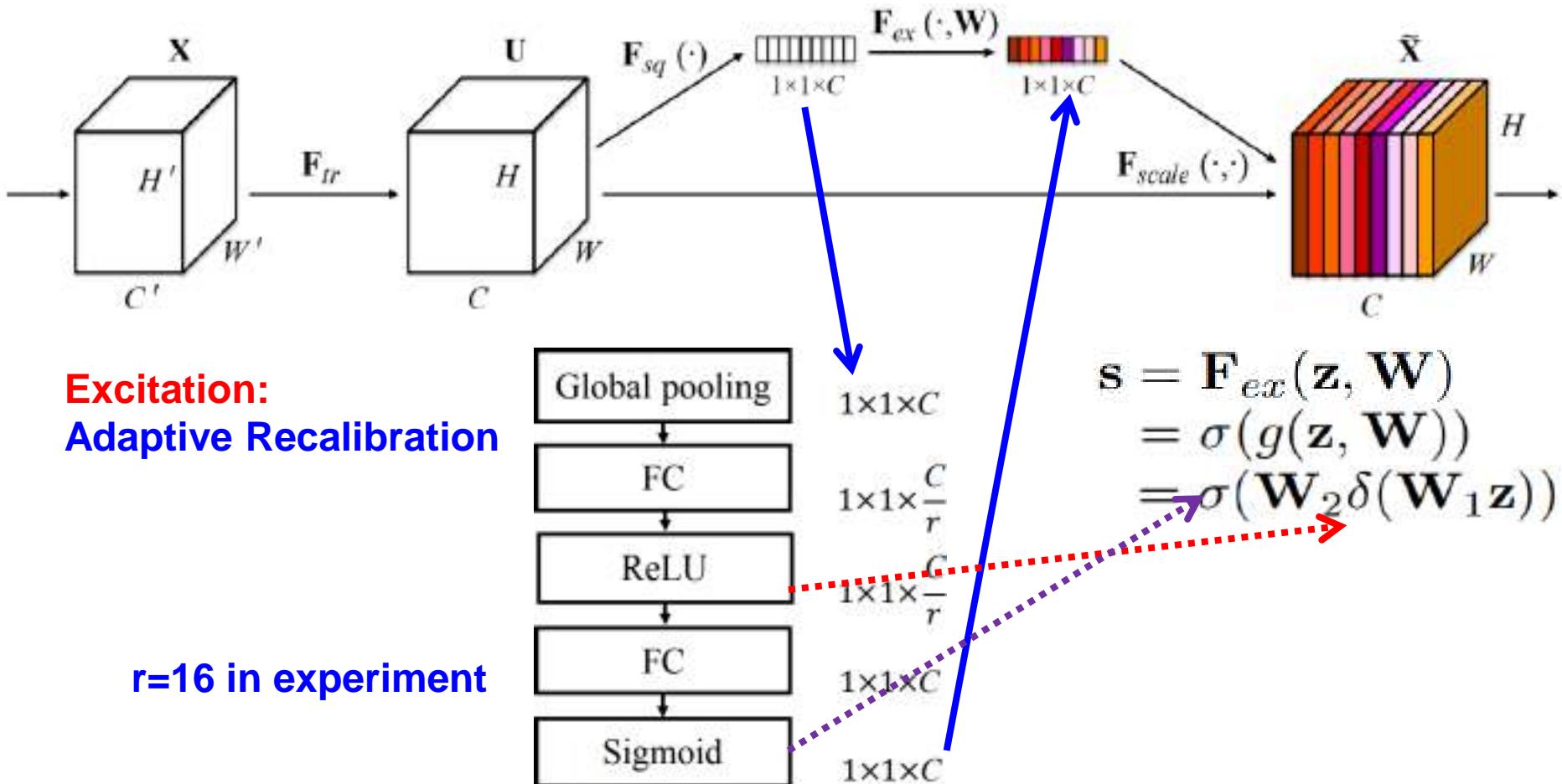
Top-5 Error: 2.251%



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

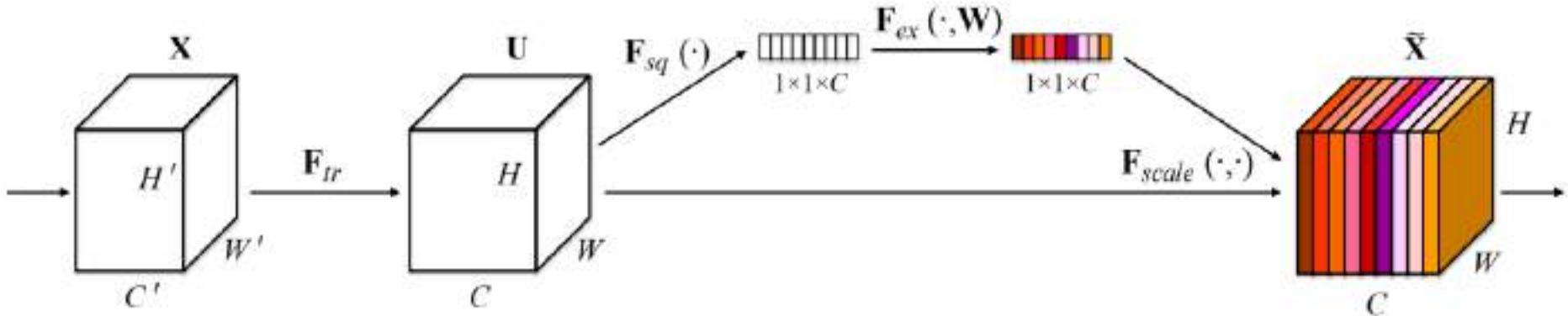
Top-5 Error: 2.251%



SENet (Squeeze and Excitation Network)

2017 ImageNet Challenge Winner

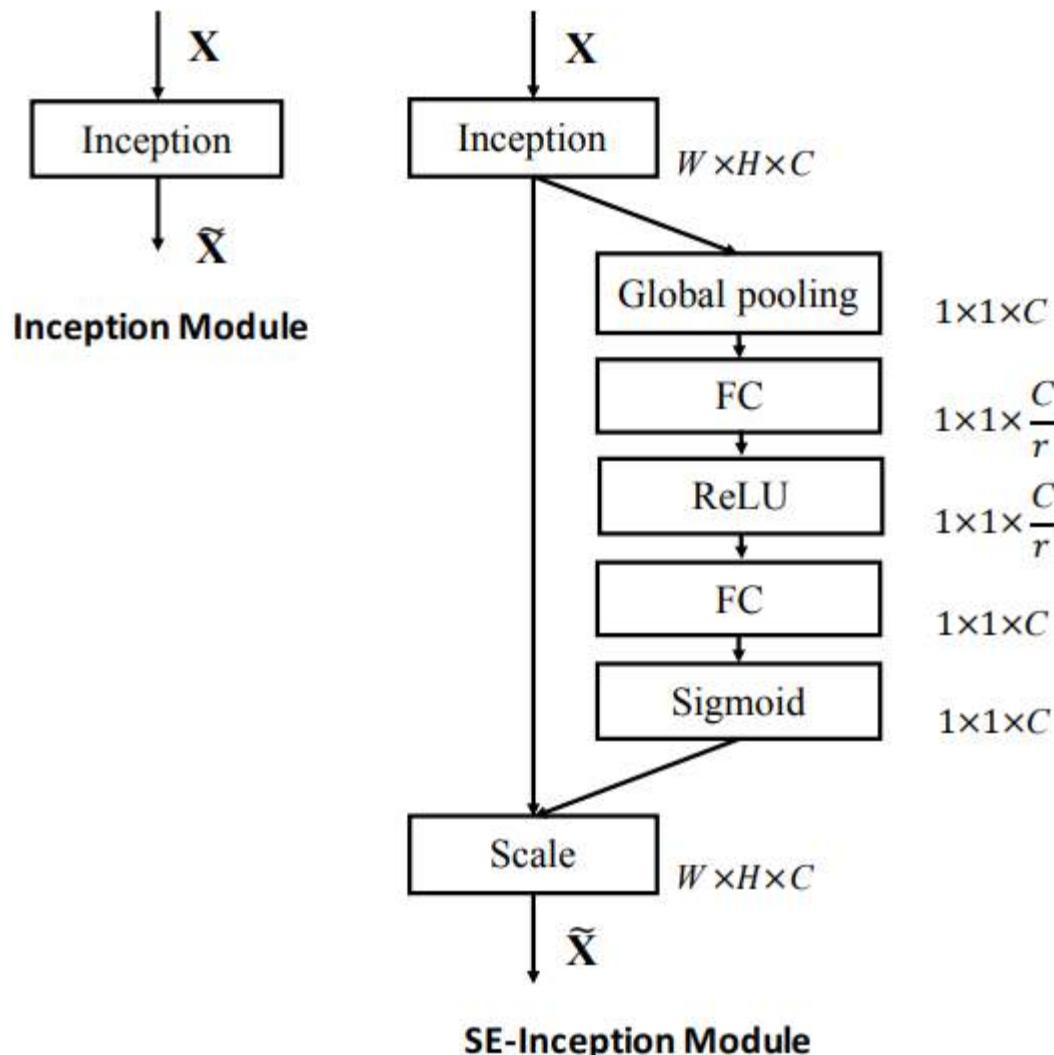
Top-5 Error: 2.251%



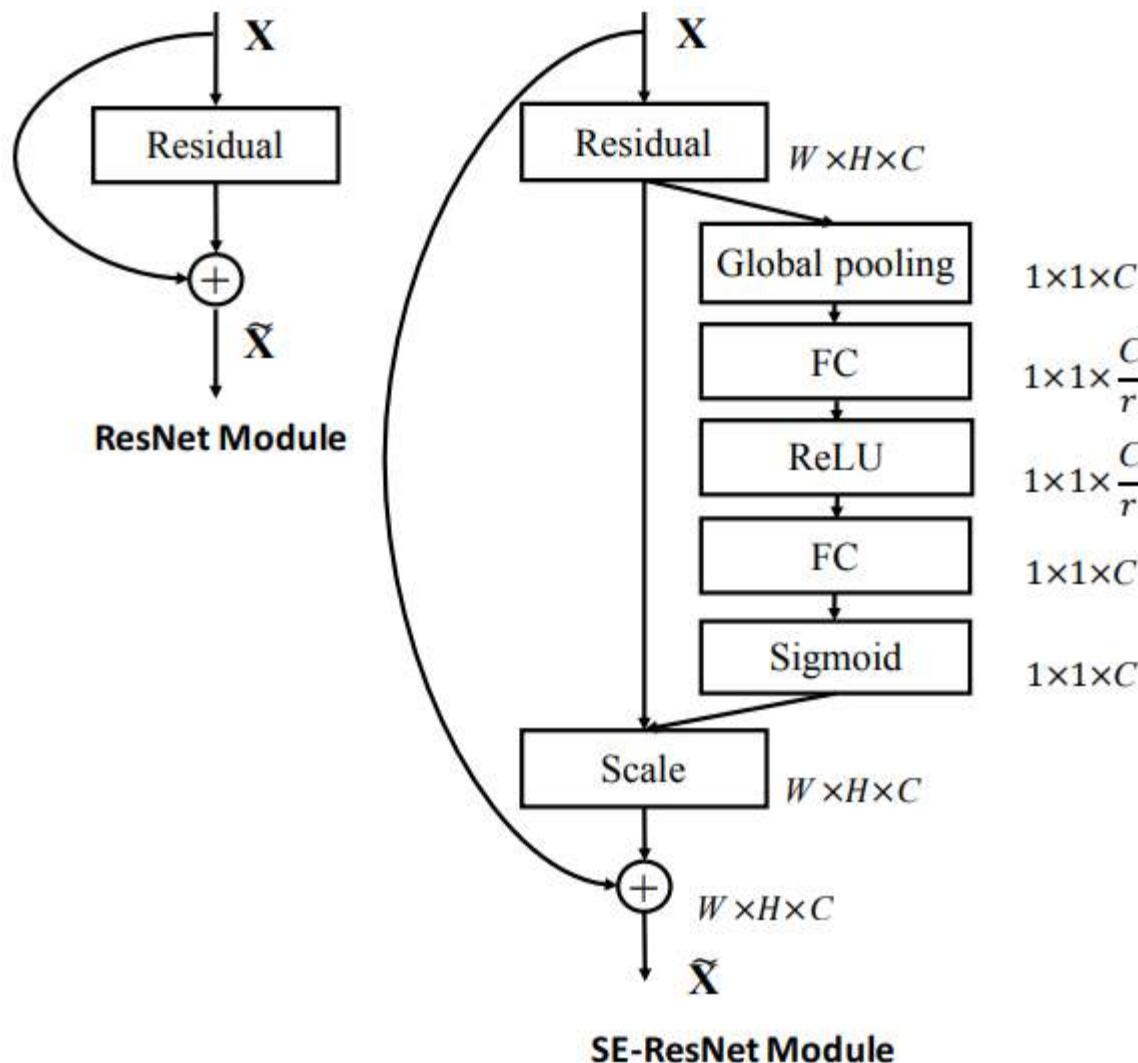
Scaling:

$$\tilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c$$

SE-Inception Module



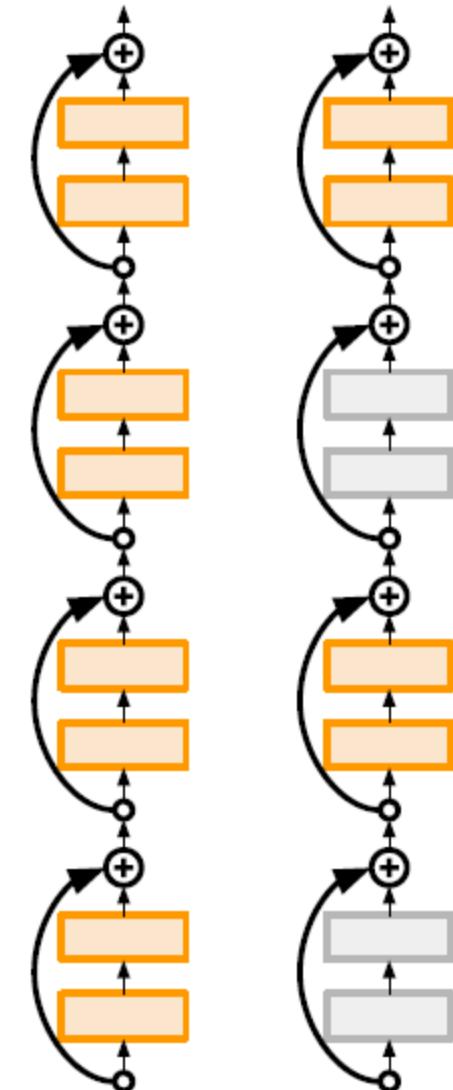
SE-ResNet Module



Other ResNet Improvements to Know ...

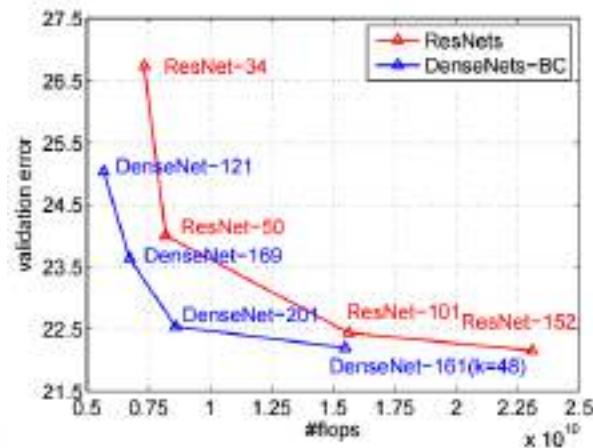
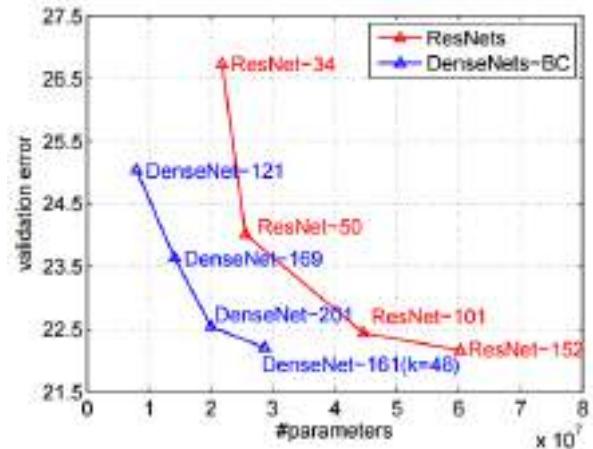
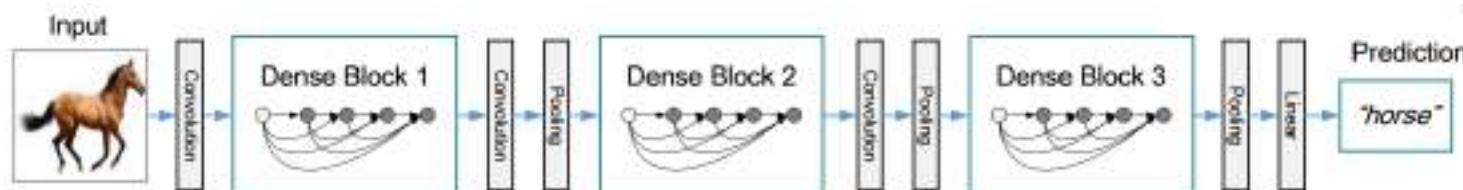
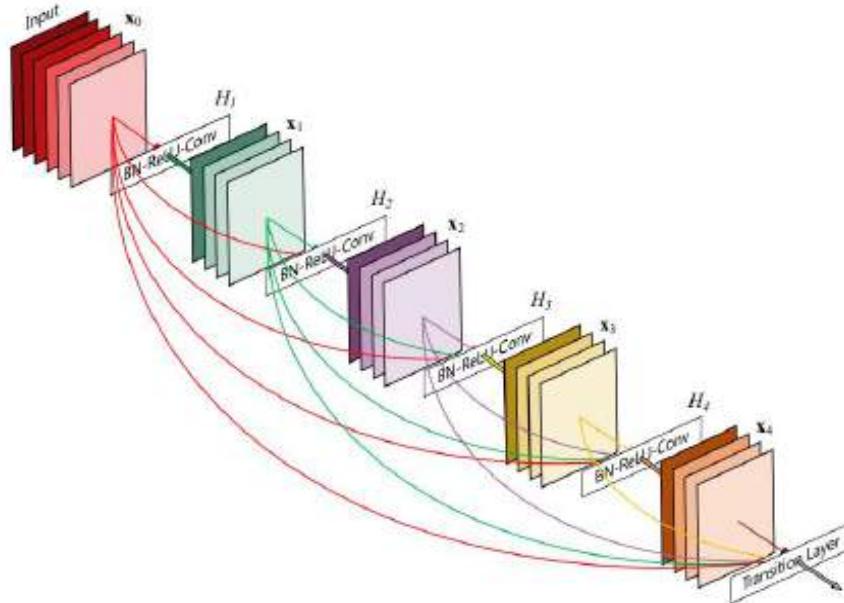
Deep Networks with Stochastic Depth

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time



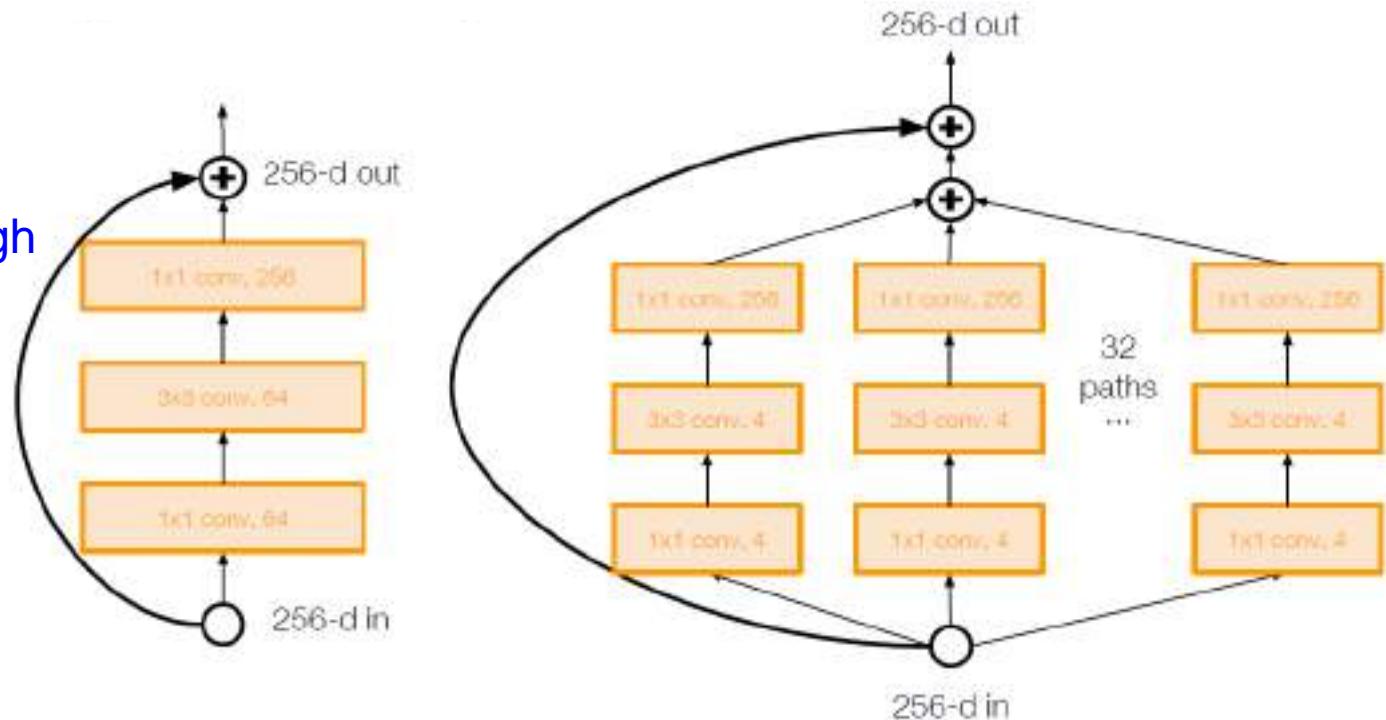
DenseNet

- Shorter connections (like ResNet) help
- Why not just connect them all?



Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

- Improved ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



Things to remember

- Architectures: Plain Models
 - LeNet (1998)
 - 5 Layers
 - No progress till 2012 due to lack of large scale data and computational resources
 - AlexNet (2012)
 - 8 Layers
 - Game changer in Computer Vision Area
 - ZFNet (2013)
 - 8 Layers with improved hypermeter setting
 - VGGNet (2014)
 - Deeper model: 16 or 19 Layers
 - Uniform filters
 - NiN (Network in Network) (2014)
 - Inspiration to DAG Model

Things to remember

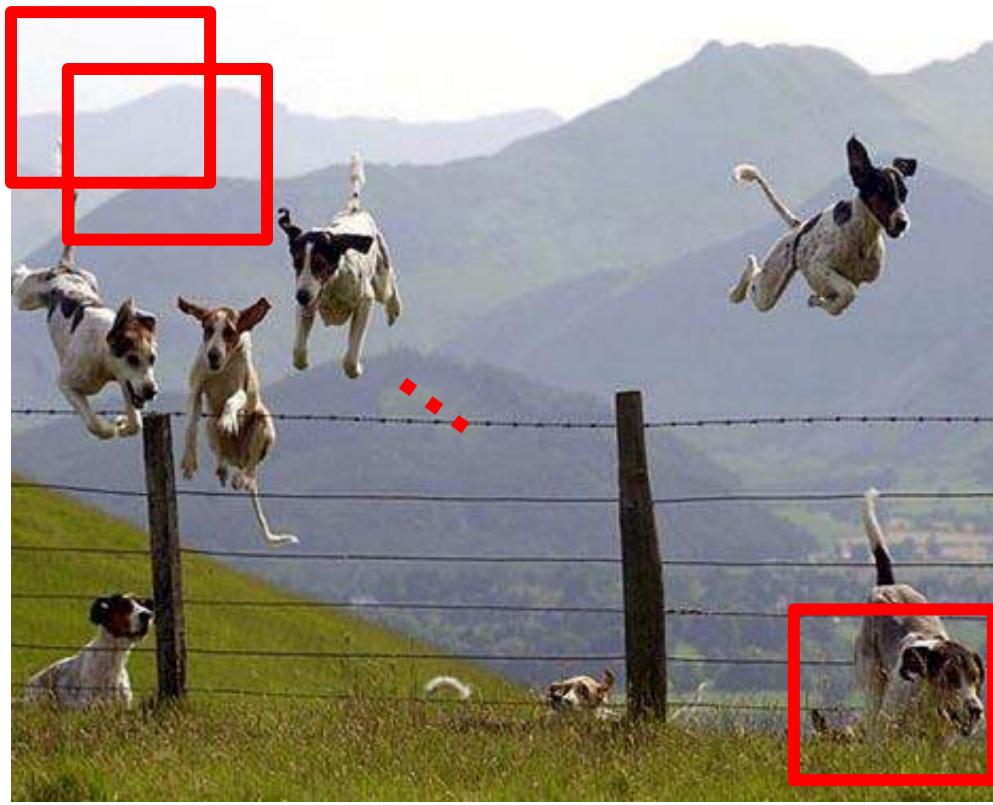
- DAG Architectures
 - GoogLeNet
 - ResNet
 - Pre-activated ResNet
 - SENet
 - Stochastic Depth
 - DenseNet
 - ResNetXt
- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections

Acknowledgements

- Thanks to the following researchers for making their teaching/research material online
 - Forsyth
 - Steve Seitz
 - Noah Snavely
 - J.B. Huang
 - Derek Hoiem
 - D. Lowe
 - A. Bobick
 - S. Lazebnik
 - K. Grauman
 - R. Zaleski
 - Antonio Torralba
 - Rob Fergus
 - Leibe
 - And many more

Next Class

Object Detection



**Object or
Non-Object?**