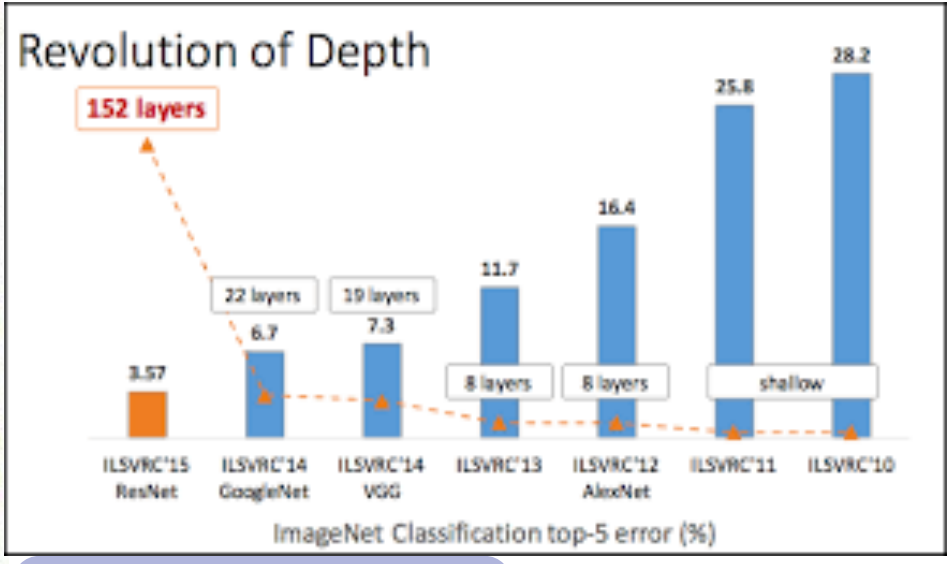
Applications of Deep Learning

Slides from

1) Stanford CS231n Lecture Notes

2) U Michgan EECS 498 Lecture Notes

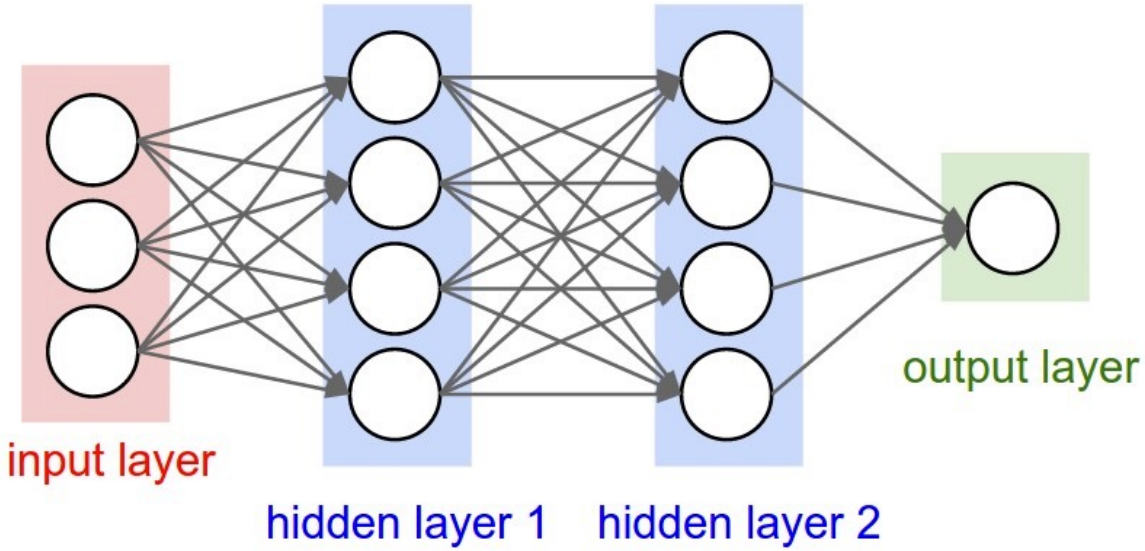
ImageNet Challenge



Convolution Neural Network

Natural Language Processing

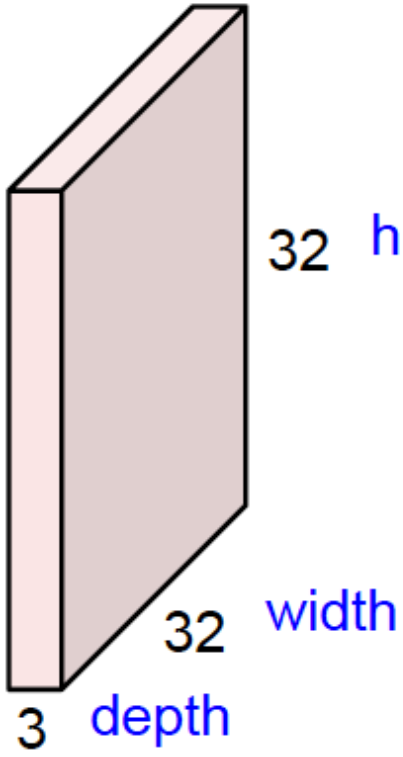
Fully Connected Layers



- ImageNet: 224x224x3 pixels
- One hidden Layer with 5,000 nodes
  - Input -> Hidden Layer: 750 Million weights 224x224x3x5000=750M
  - Hidden Layer -> Output: 5 Million weights

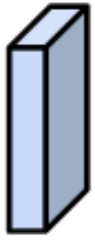
Signal Processing

Filter Parameters



Input Image

5x5x3



Filter

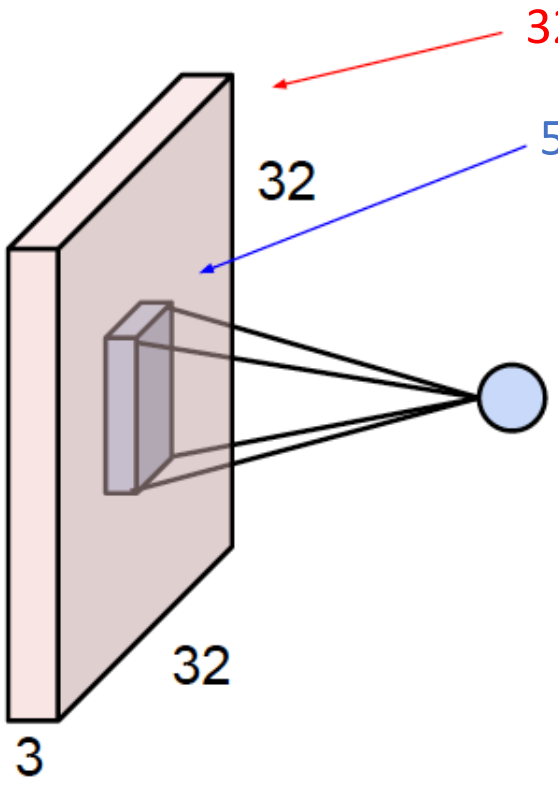
Learn parameters for filter

Input Image
- Height
- Width
- Depth

=

Filter
- Height
- Width
- Depth

Convolution Operation



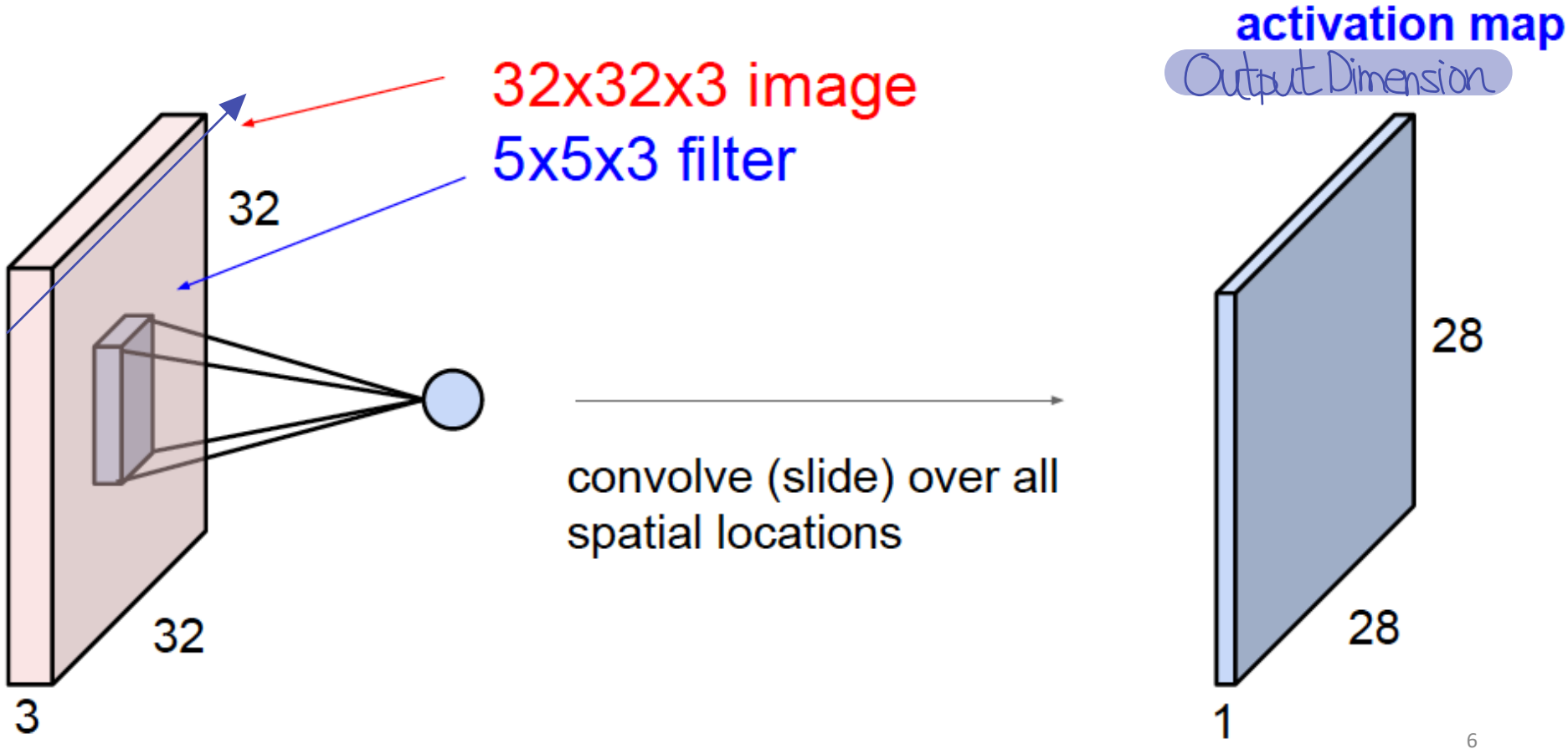
One Value obtained from this operation:

- Filter: **w**
- Image Patch: **x**

$$y = \mathbf{w}^T \mathbf{x} + b$$

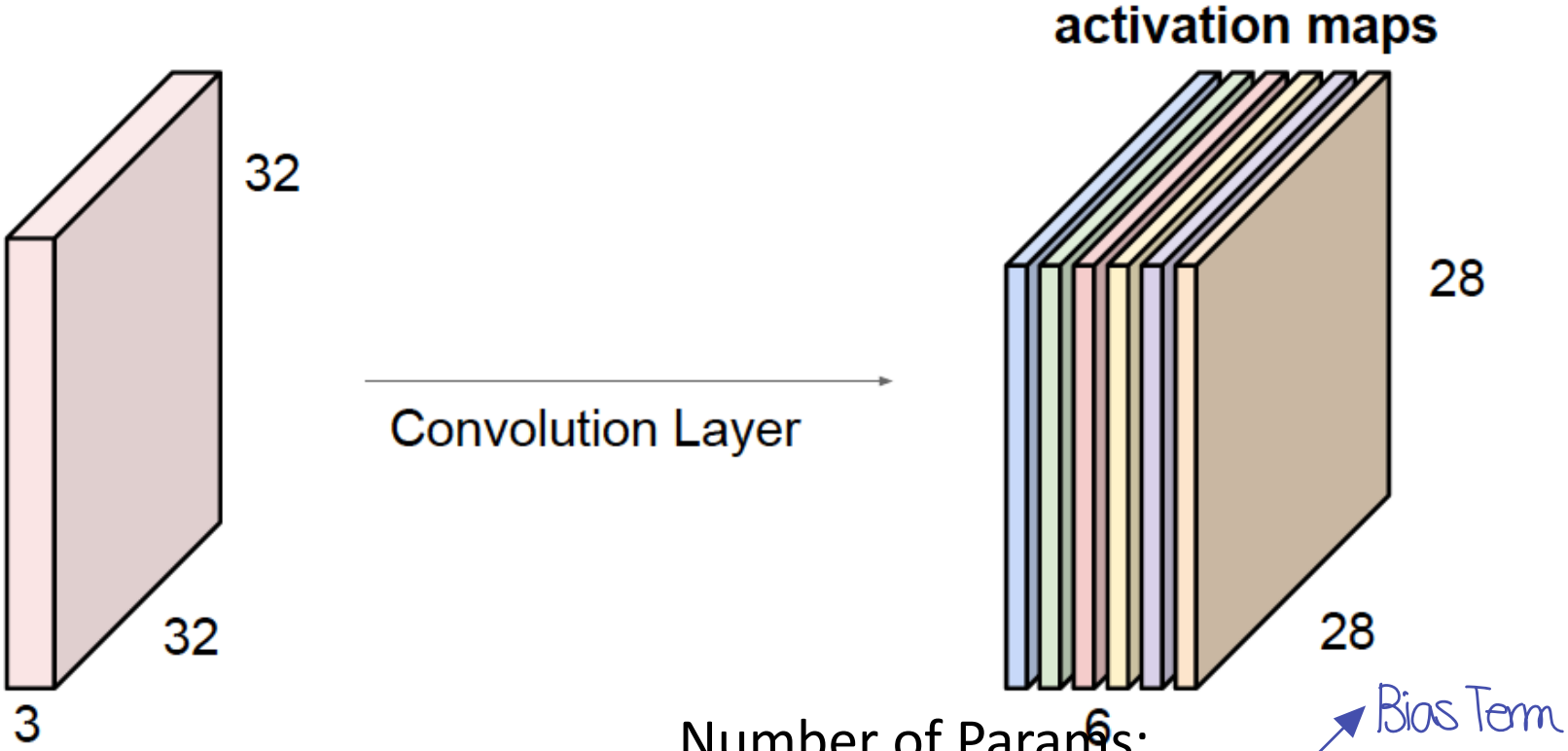
Elementwise Multiplication

Convolution Operation - II



Convolution Operation - III

Convolution Operation - IV



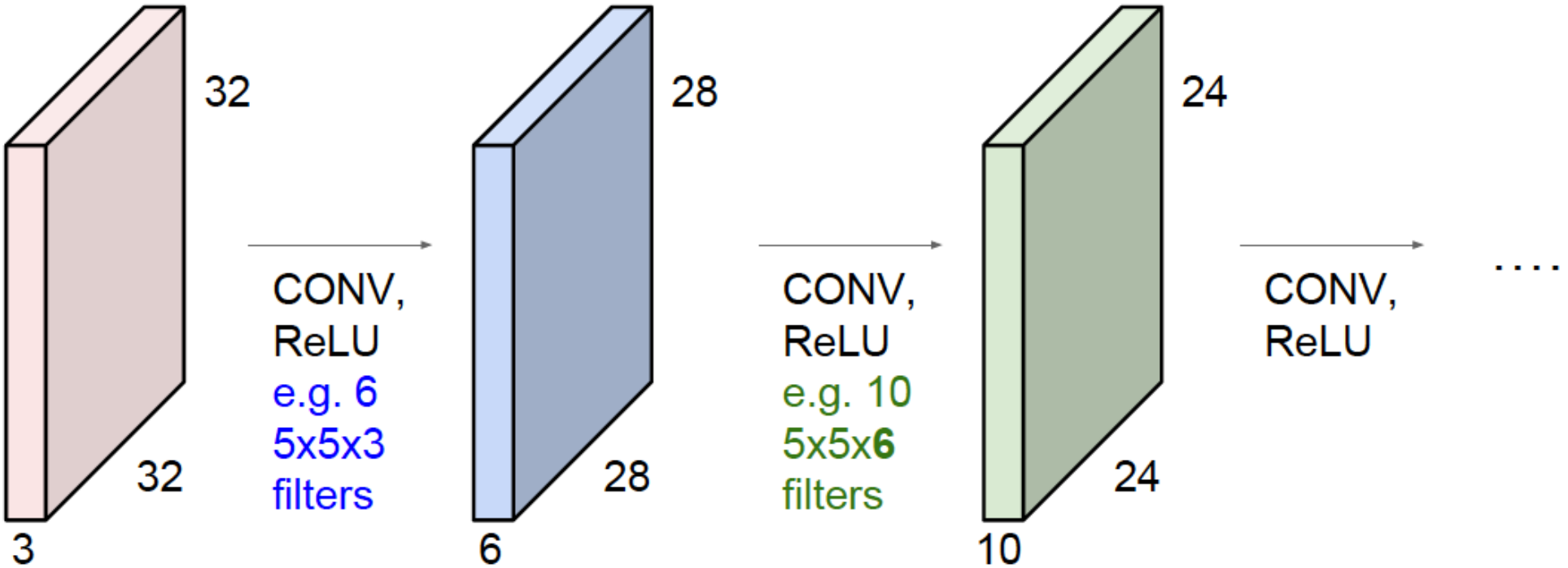
Input Image: 32 x 32 x 3

Output: 28 x 28 x 6

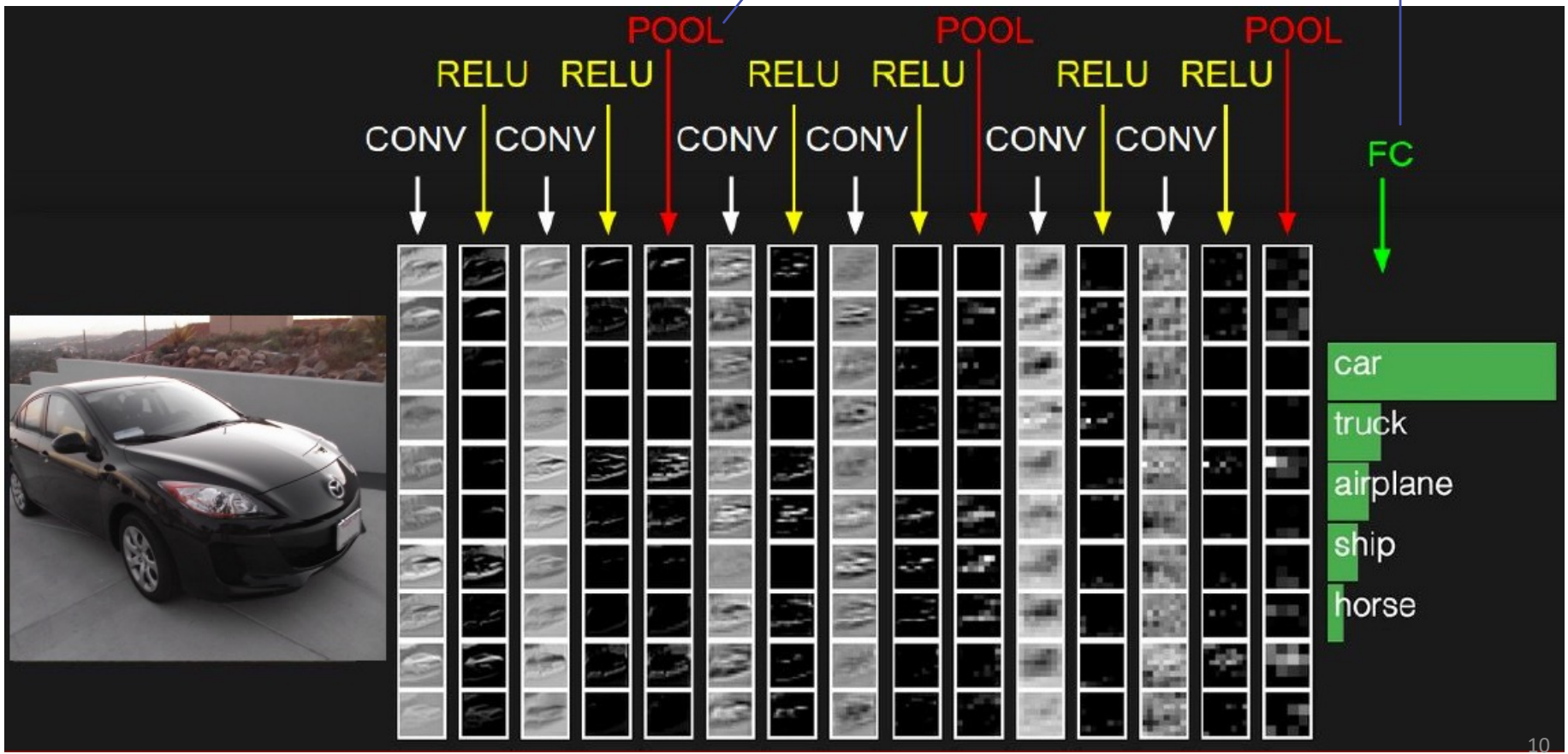
Number of Params:

- Per Filter: 5 x 5 x 3 + 1 = 76 (Bias Term)
- Total #: 6 x 76 = 456

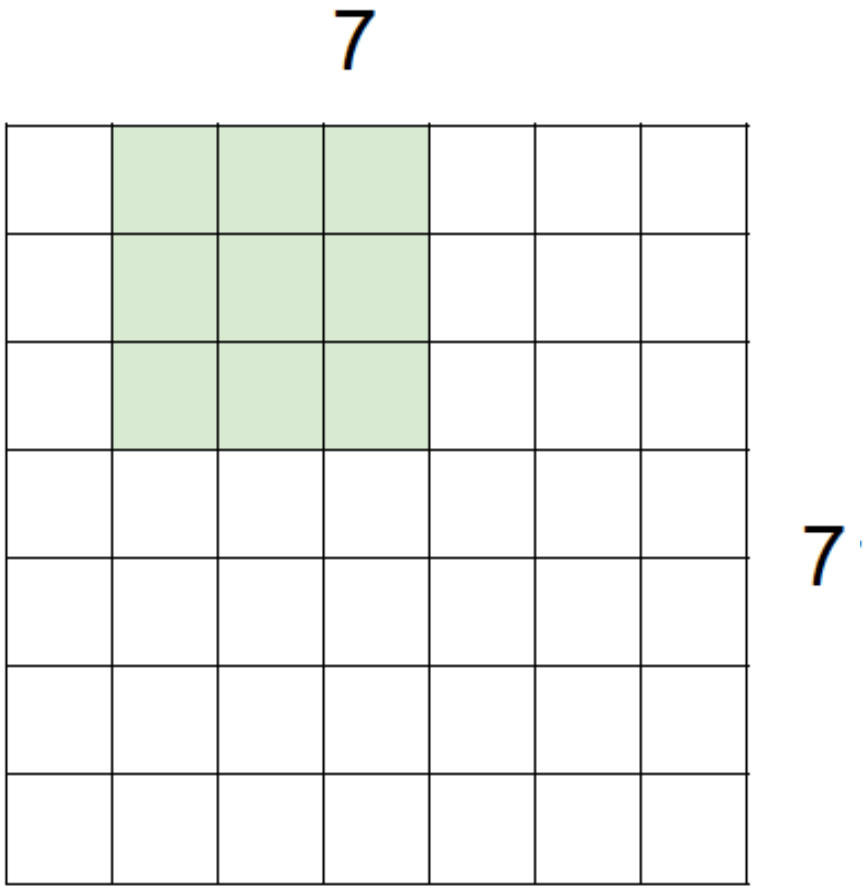
Convolutional Neural Network (CNN)



Architecture of a CNN

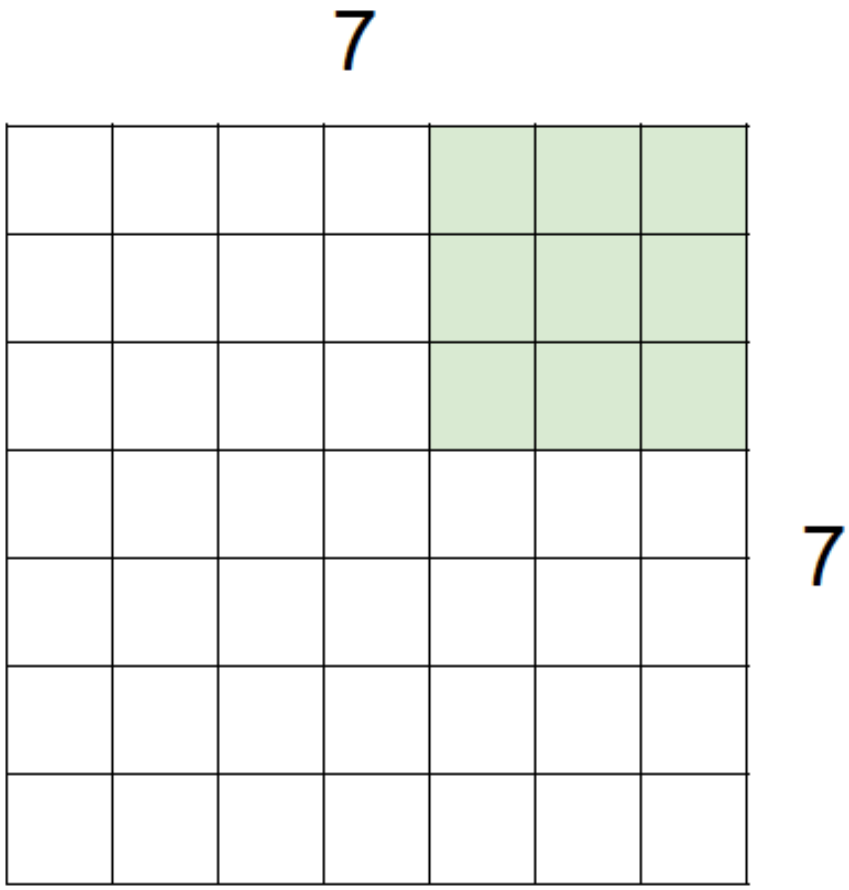


Closer Look at Convolution Operation



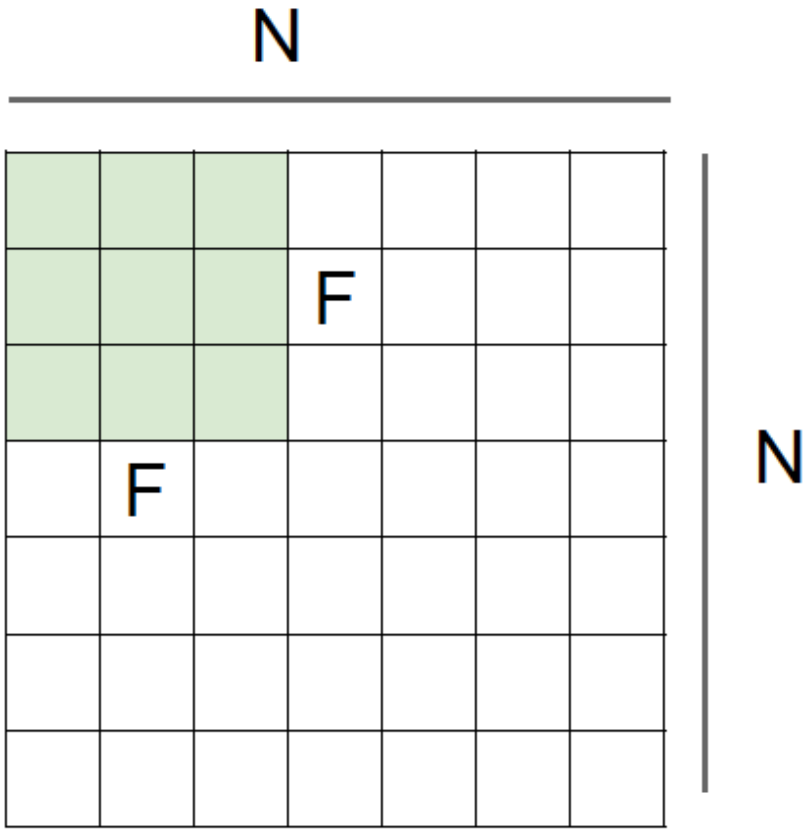
- Input: 7 x 7
- Filter: 3 x 3
- Stride: 1
- Output: 5 x 5

Closer Look at Convolution Operation



- Input: 7 x 7
- Filter: 3 x 3
- Stride: 2 (Jump By 2)
- Output: 3 x 3

Output Dimensions



Output size:

(N - F) / stride + 1

N (Input Size), F (Filter Size)

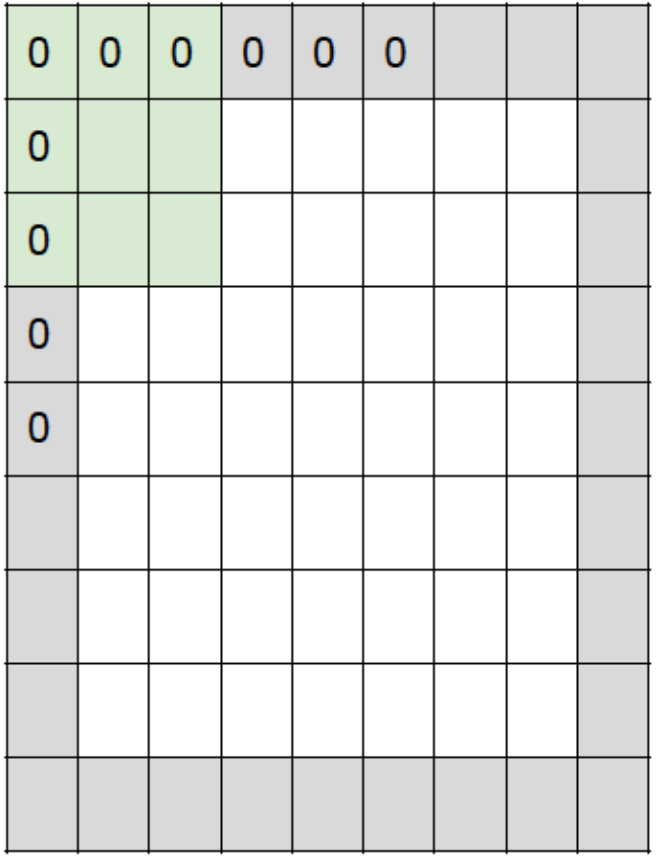
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

~~stride 3 => (7 - 3)/3 + 1 = 2.33~~

Zero Padding Input Layers



- Input 7 x 7
- Filter: 3 x 3
- Stride: 1
- Zero Pad: 1 pixel border
- Output: 7 x 7

Preserving Image Size

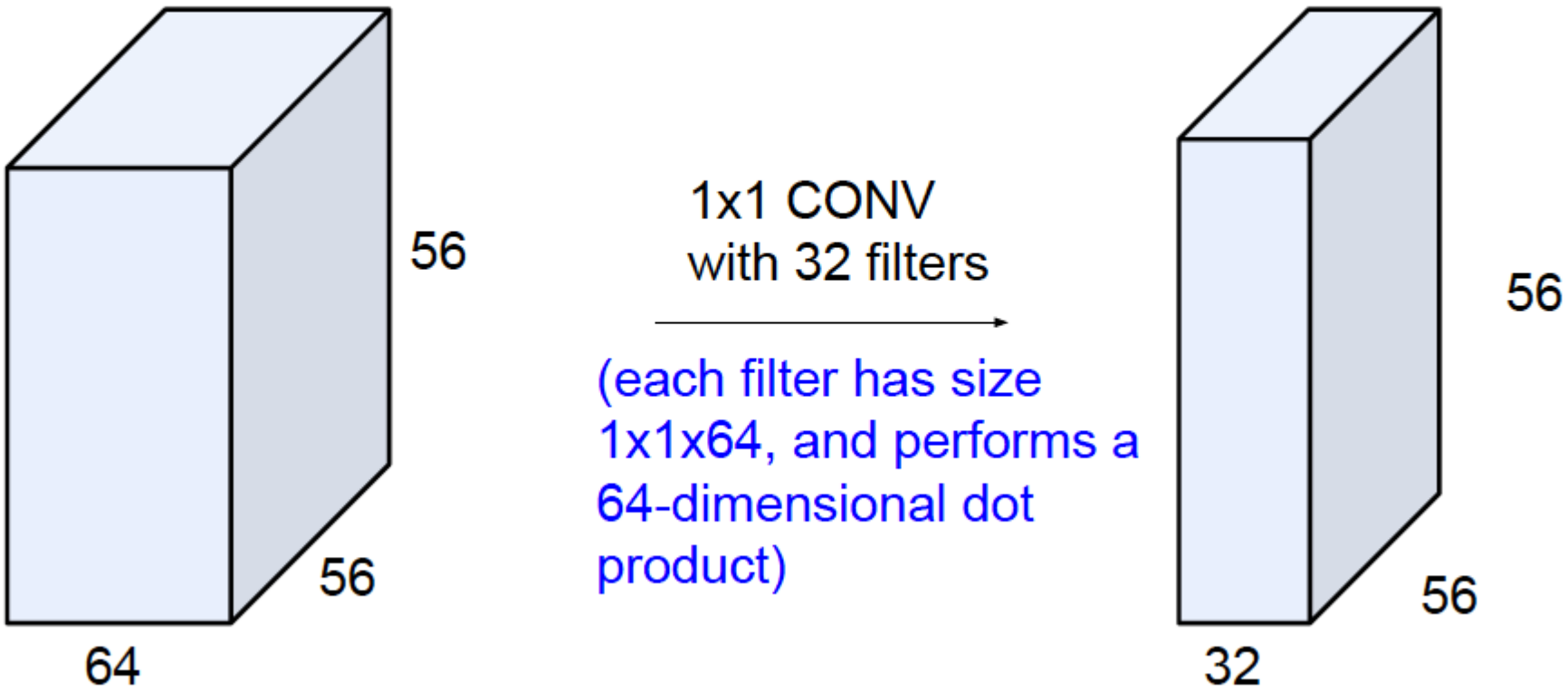
- Filter Size F x F, Stride = 1
- Zero Pad with (F-1)/2 pixel borders
  - 3 x 3 => 1 pixel border
  - 5 x 5 => 2 pixel border
  - 7 x 7 => 3 pixel border

Example

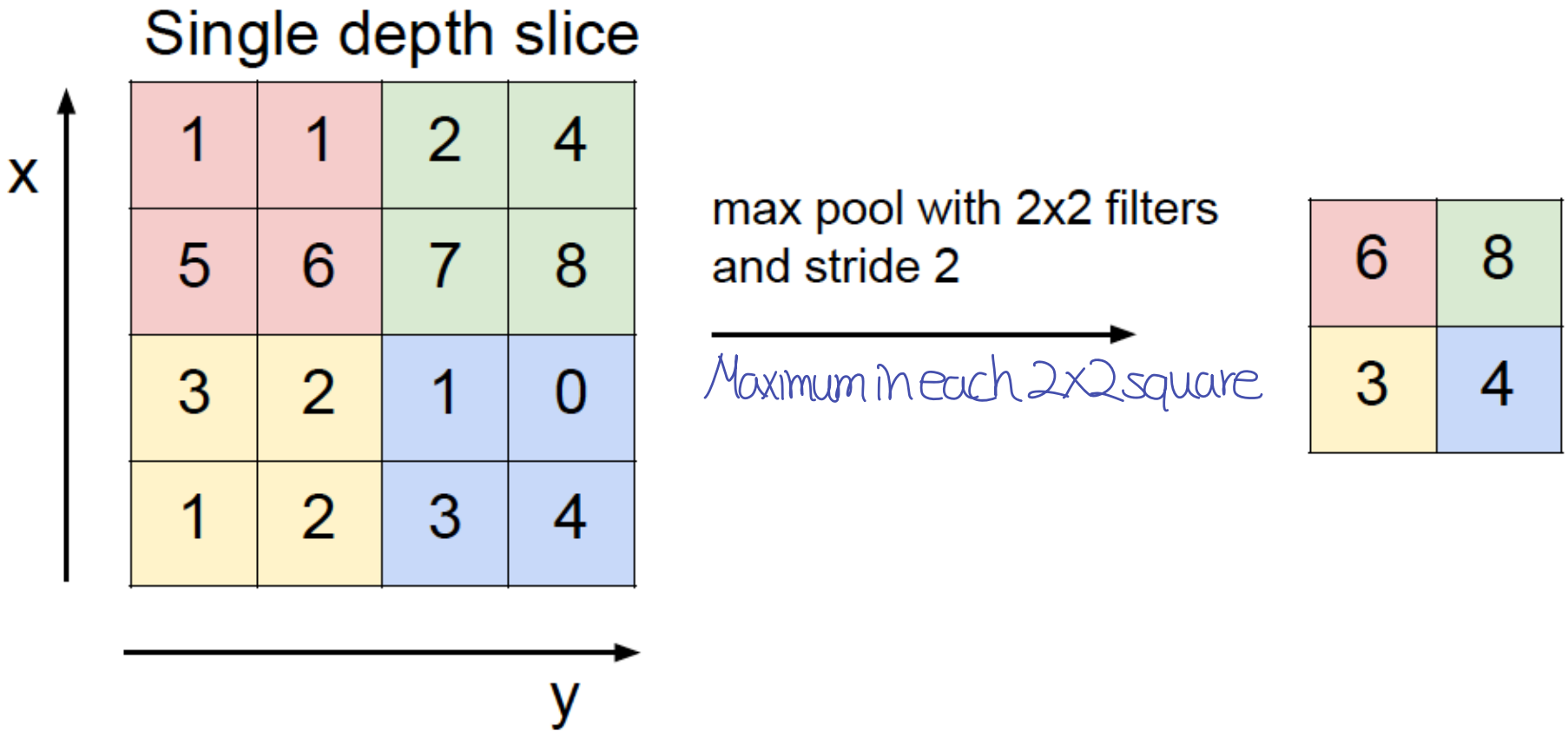
- Input: 32 x 32 x 3
- 10 Filters, each 5 x 5 x 3
- Stride: 1
- Zero Pad: 2 pixels on each border

- Output Dimensions: 32 x 32 x 10 → Ten Filters
- Number of Parameters: 10*(5x5x3+1)=760

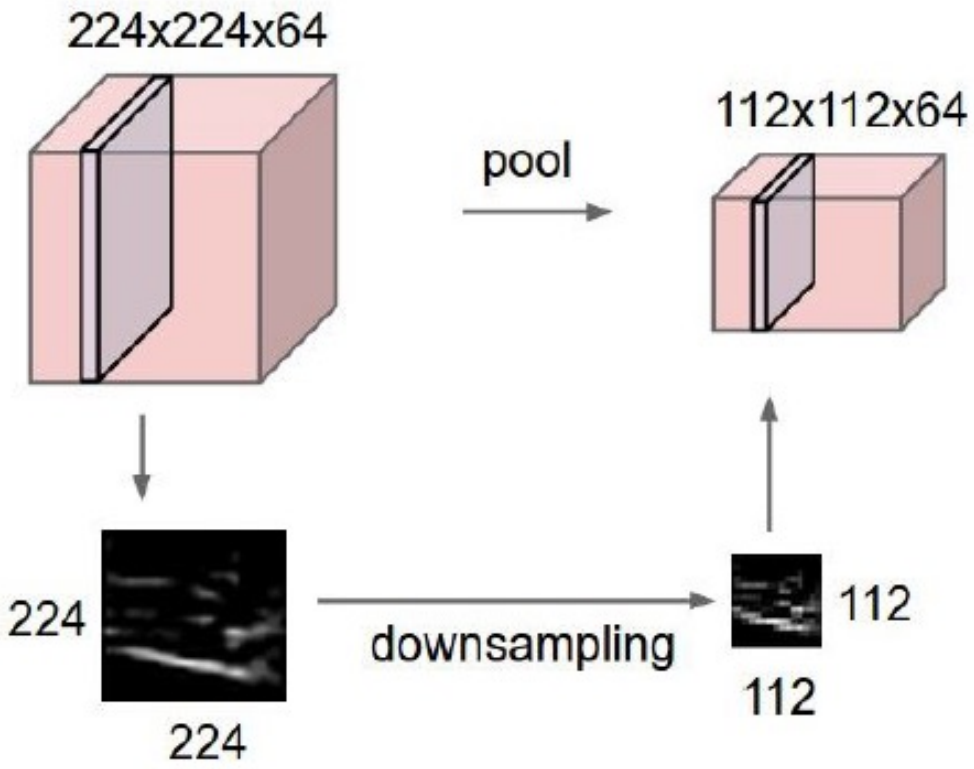
1 x 1 Filters (Changing Depth)



Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
Maximum in each 2x2 square
6 8
3 4

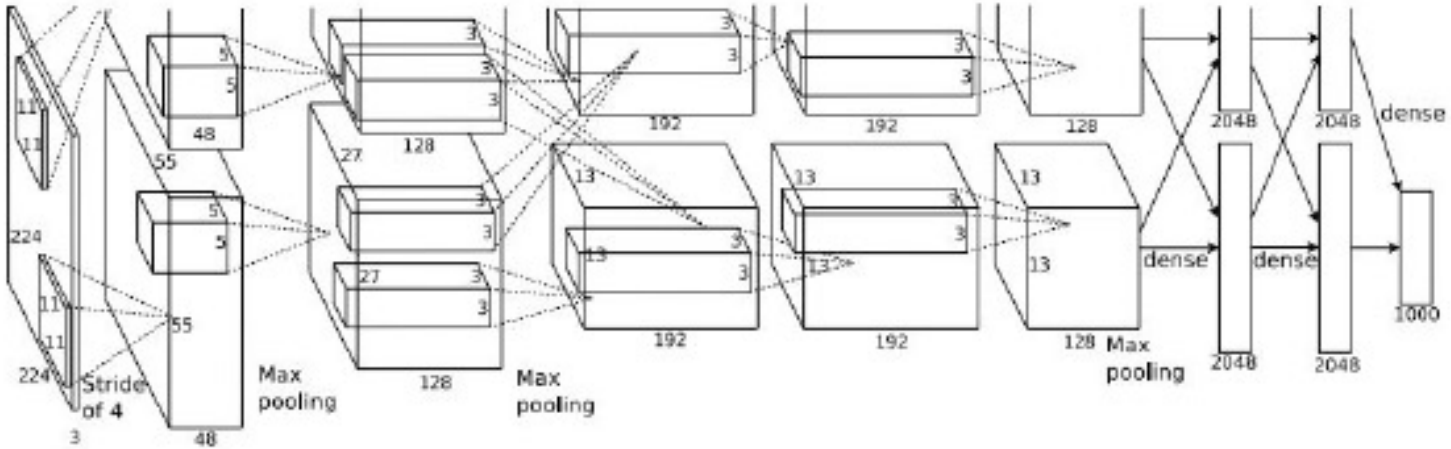
Pooling Layer: Down-Sampling Operation



Alexnet

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton (2012)

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



Layer 1 Parameters:

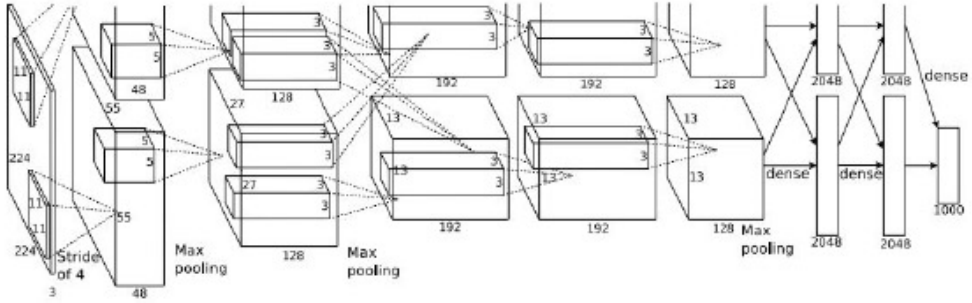
- Input: 227x227x3 images
- Layer 1: 96 11x11 filters applied at stride 4
- Output: 55 x 55 x 96
- Number of Parameters (11*11*3)*96 = 35K

Alexnet

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton (2012)

MAXPOOL Slide Window

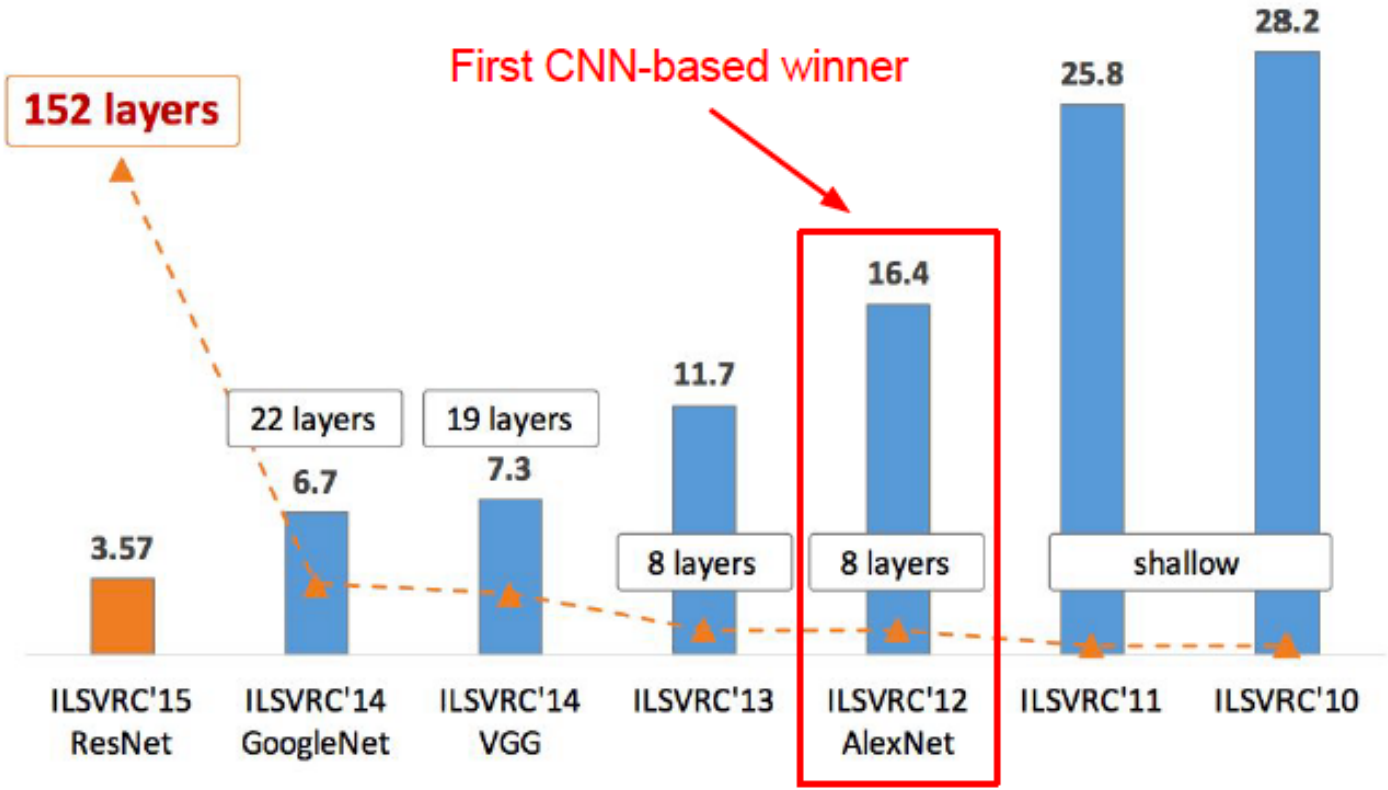
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4,
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1,
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1,
[13x13x384] CONV4: 384 3x3 filters at stride 1,
[13x13x256] CONV5: 256 3x3 filters at stride 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
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Ensemble Learning

ImageNet Large Scale Visual Recognition Challenge



VGG Net

Karen Simonyan & Andrew Zisserman (2014)

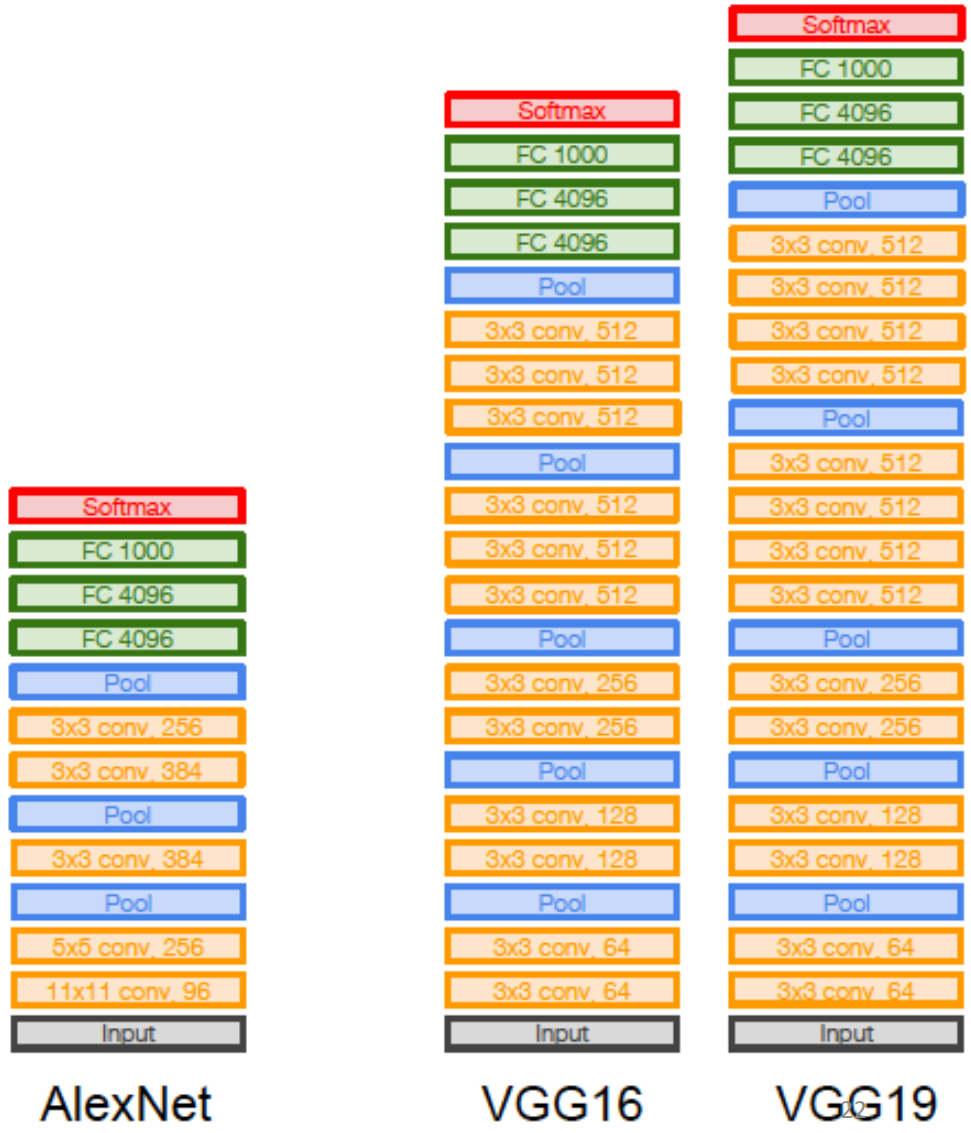
- 8 layers (AlexNet)
- 16 - 19 layers (VGGNet)

- 3x3 CONV stride 1, pad 1
- 2x2 MAX POOL stride 2

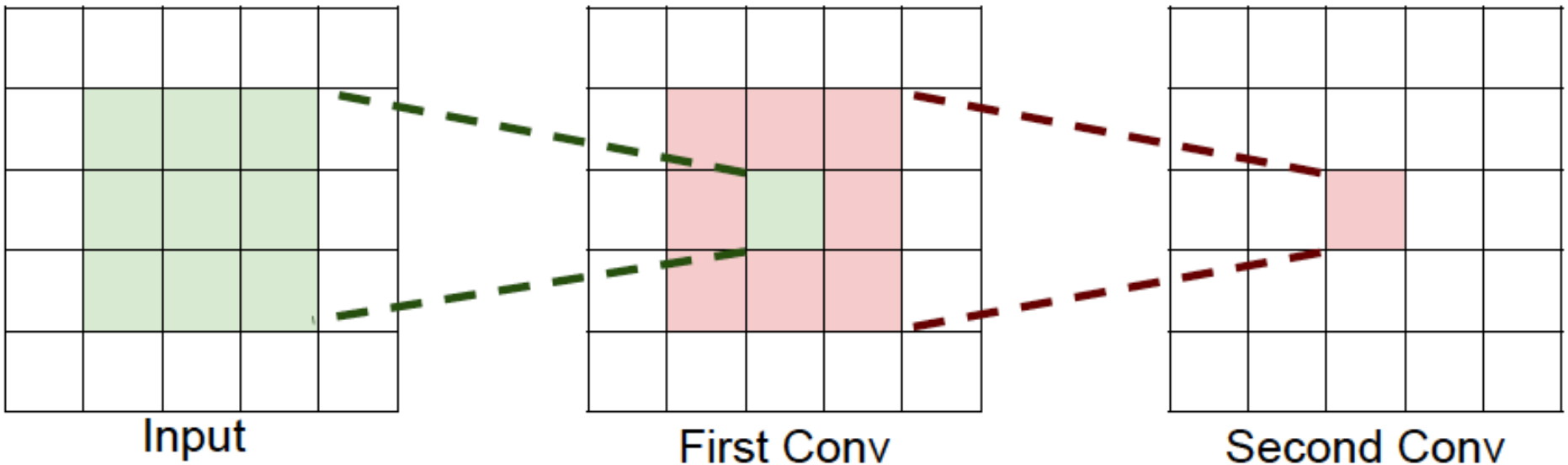
- 7.3% top 5 error in ILSVRC'14 vs
- 16.4% error in AlexNet

- Total Number of Parameters: ~138Million in VGGNet
- ~62 Million in Alexnet

Deeper the better



Why multiple 3x3 layers are preferred?



- Receptor Field of First Layer: 3 x 3
- Effective Receptor Field of 2nd Layer: 5 x 5
- Effective Receptor Field of 3rd Layer: 7 x 7

Why multiple 3x3 layers are preferred?

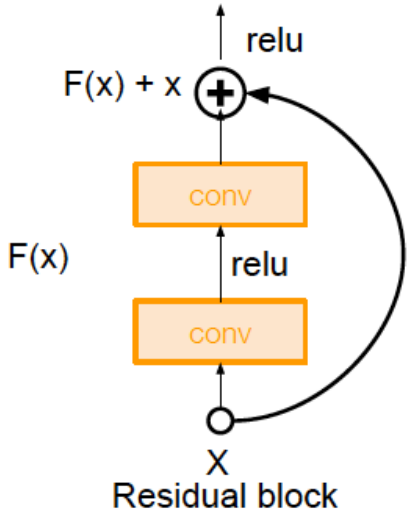
- A stack of three layers with 3 x 3 filters has the same receptive field as a single layer with 7 x 7 filters
- Number of Parameters:
  - Three 3 x 3 filters: 27 (smaller)
  - One 7 x 7 filter: 49

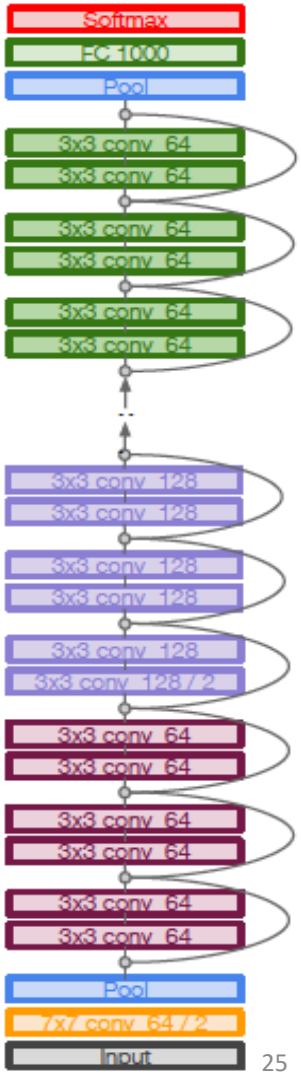
Same receptive field, less parameter

ResNet

He et. al. (2015)

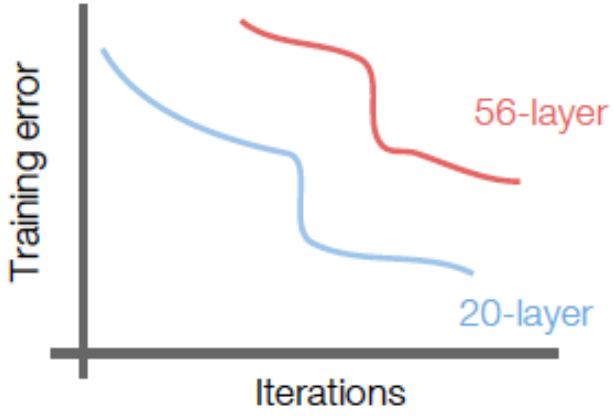
- Very deep networks using residual connections
- 152-layer model for ImageNet
- ILSVRC'15 classification winner
- (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

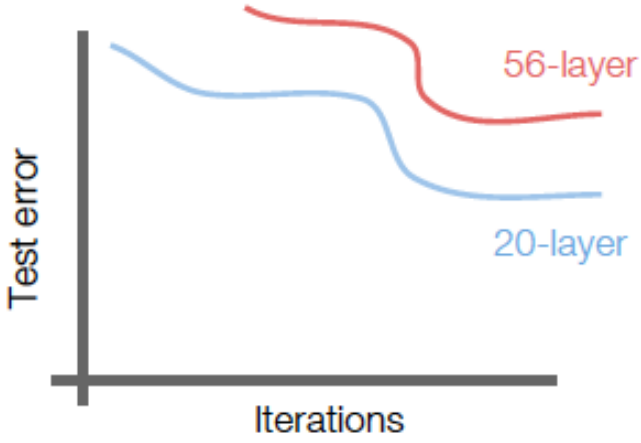




Vanishing Gradient Problem

Effect of having more layers

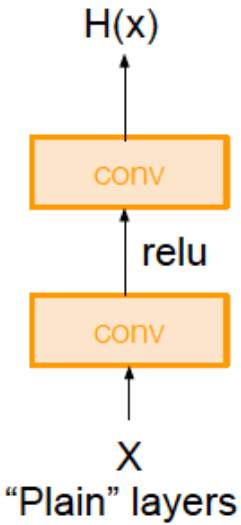


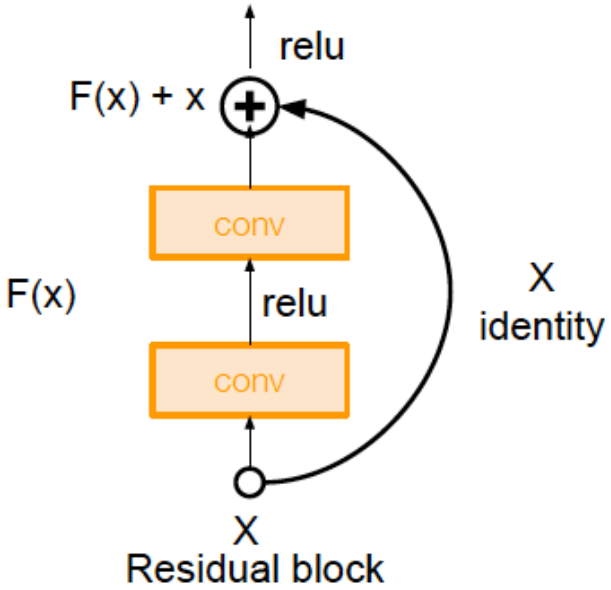


Key Idea

Skip Connection

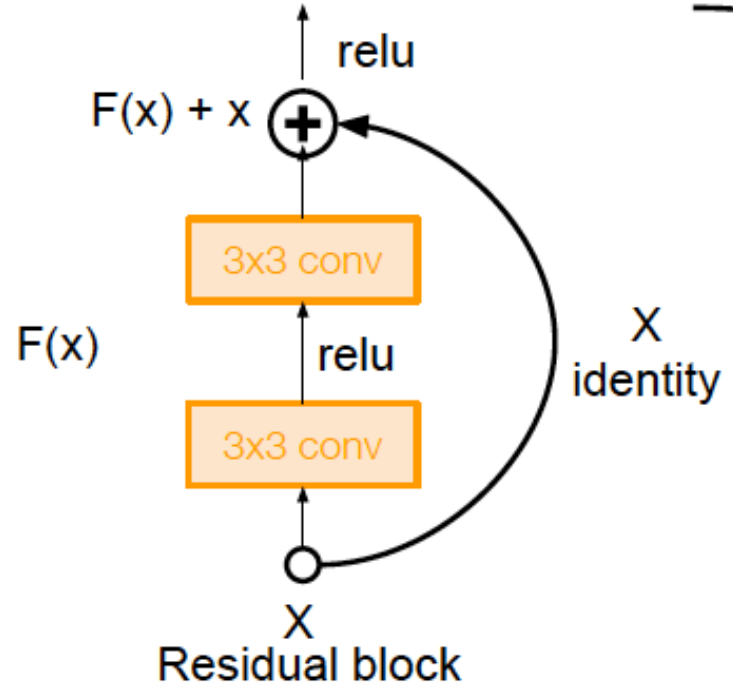
- Deeper Model should have lower training error than Shallower model if the optimization is done properly.

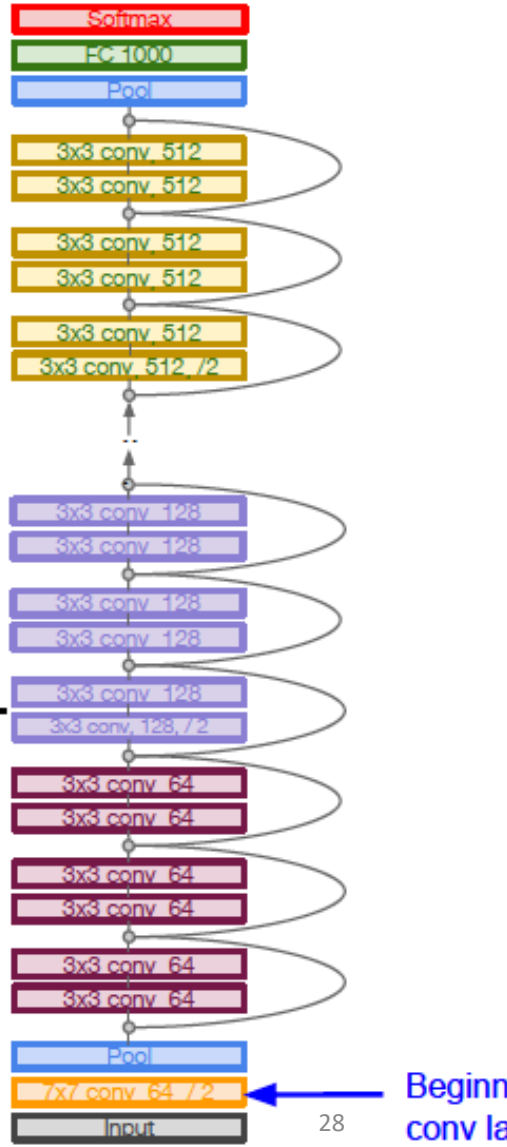




ResNet Architecture

- Full ResNet architecture:
- - Stack residual blocks
- - Every 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)

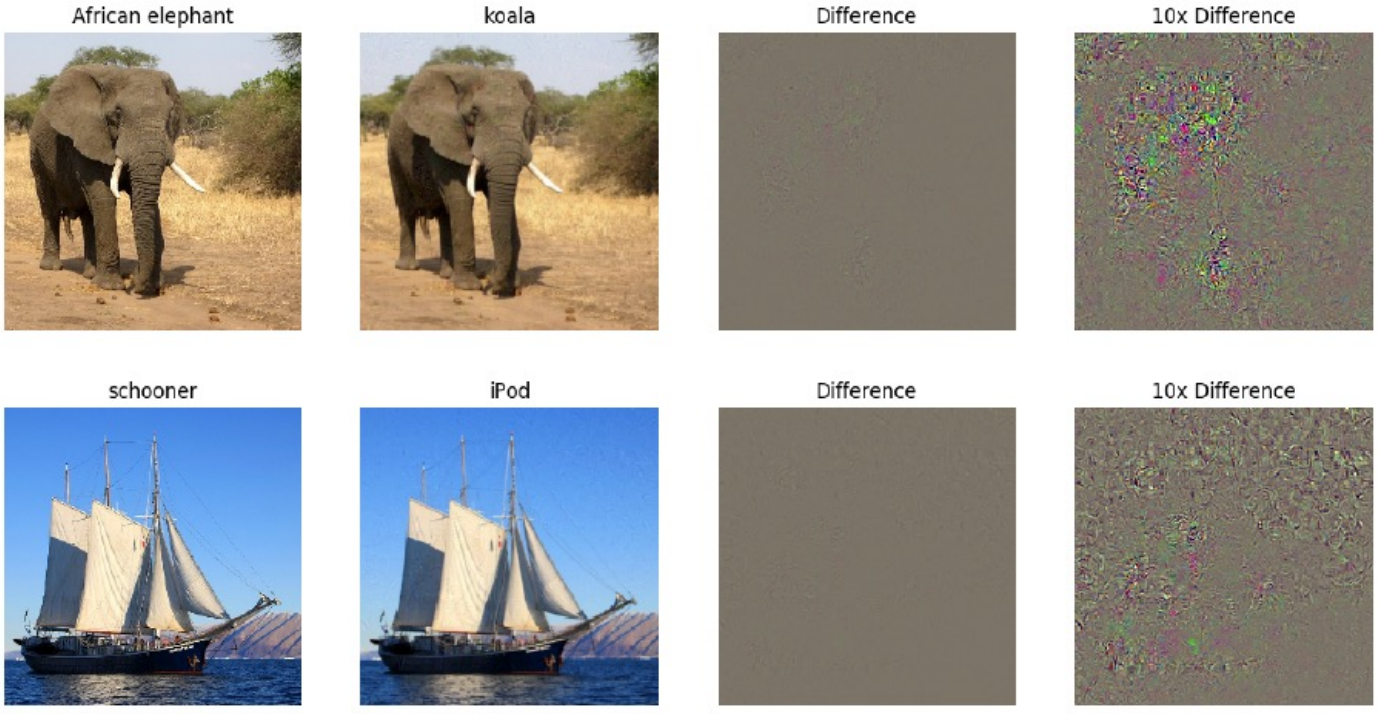




ResNet Architecture II

- 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 plateaus
- - Mini-batch size 256
- - Weight decay of 1e-5
- - No dropout used

Adversarial Examples



oat image is CC0 public domain
lephant image is CC0 public domain

Adversarial Examples

1. Start from an arbitrary image
2. Pick an arbitrary category
3. Modify the image (via gradient ascent) to maximize the class score
4. Stop when the network is fooled

Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- "looks natural" (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

Given feature vector

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

Features of new image

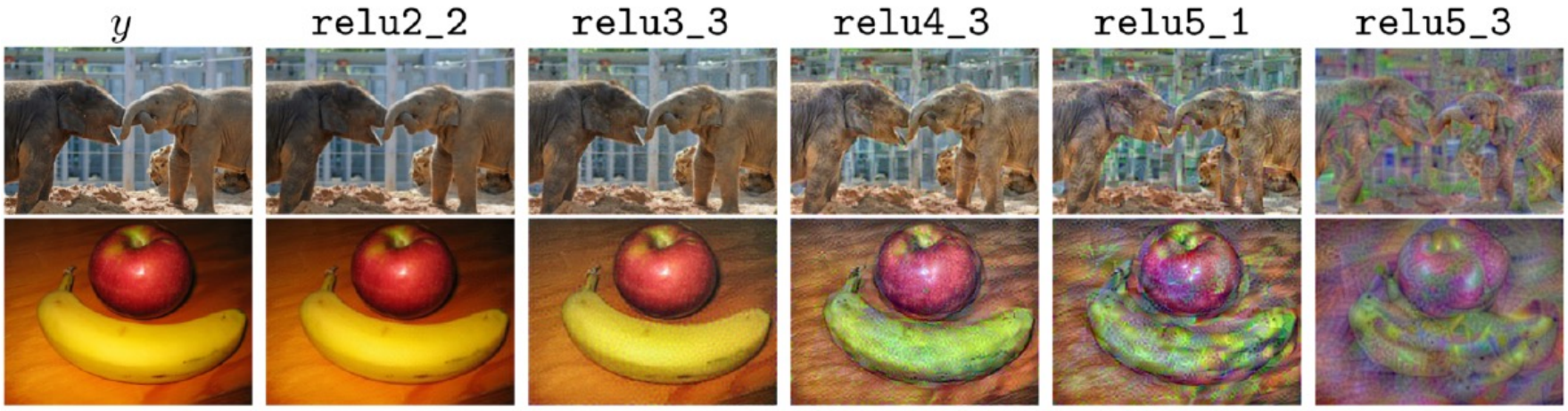
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

Feature Inversion

Reconstructing from different layers of VGG-16



Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

Neural Style Transfer

Content Image

This image is licensed under CC-BY 3.0

+

Style Image

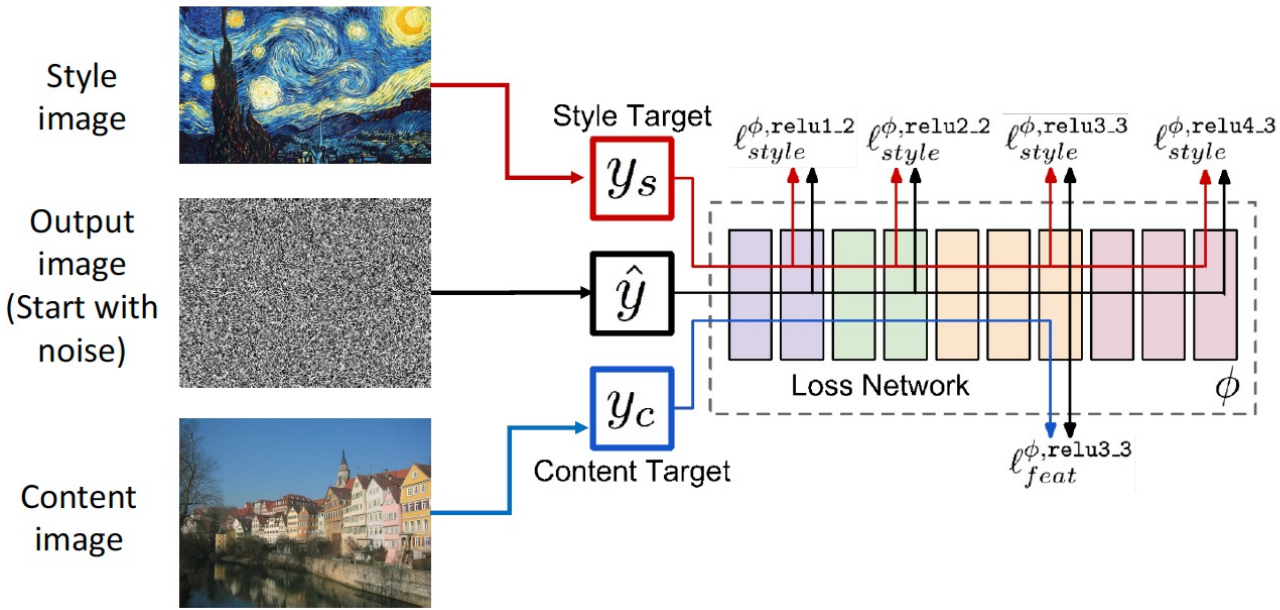
Starry Night by Van Gogh is in the public domain

=

Output Image

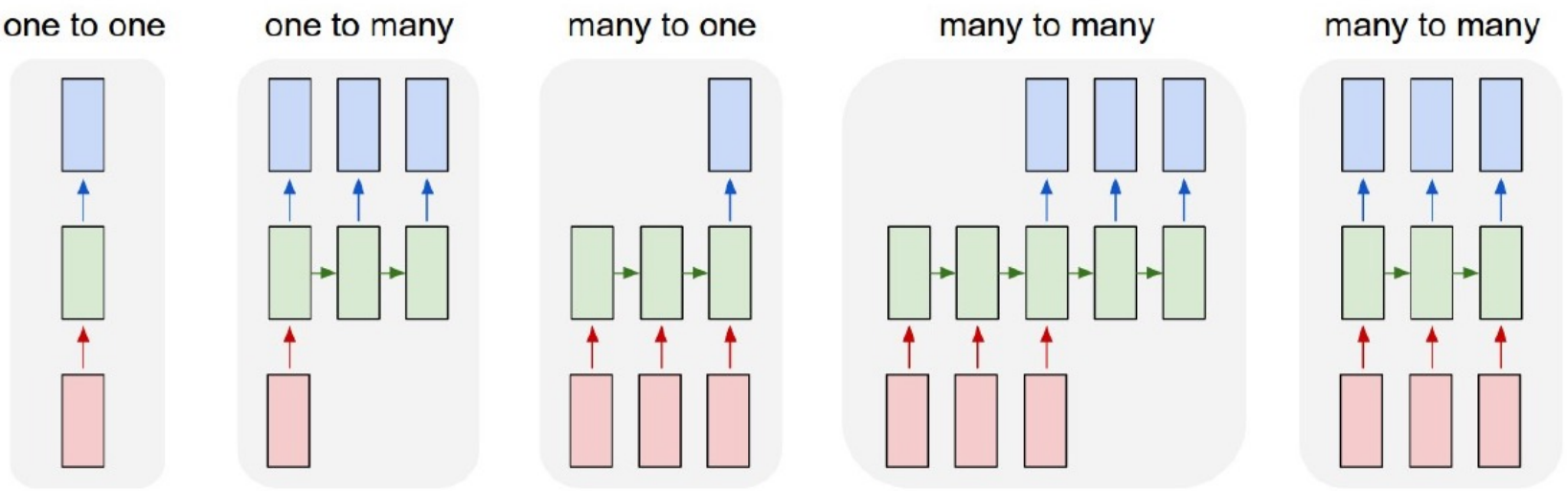
This image copyright Justin Johnson, 2015. Reproduced with permission.

Neural Style Transfer

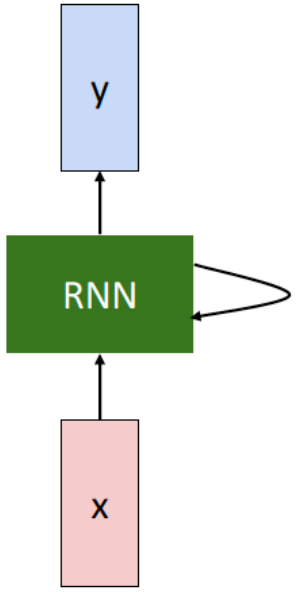


$$\arg\min_y \{\beta\ell(\Phi(y), \Phi_1) + (1-\beta)\ell(\Phi(y), \Phi_2) + \lambda\mathcal{R}(y)\}$$

Recurrent Neural Networks



RNN Architecture



The state consists of a single *"hidden"* vector **h**:

$$h_t = f_W(h_{t-1}, x_t)$$

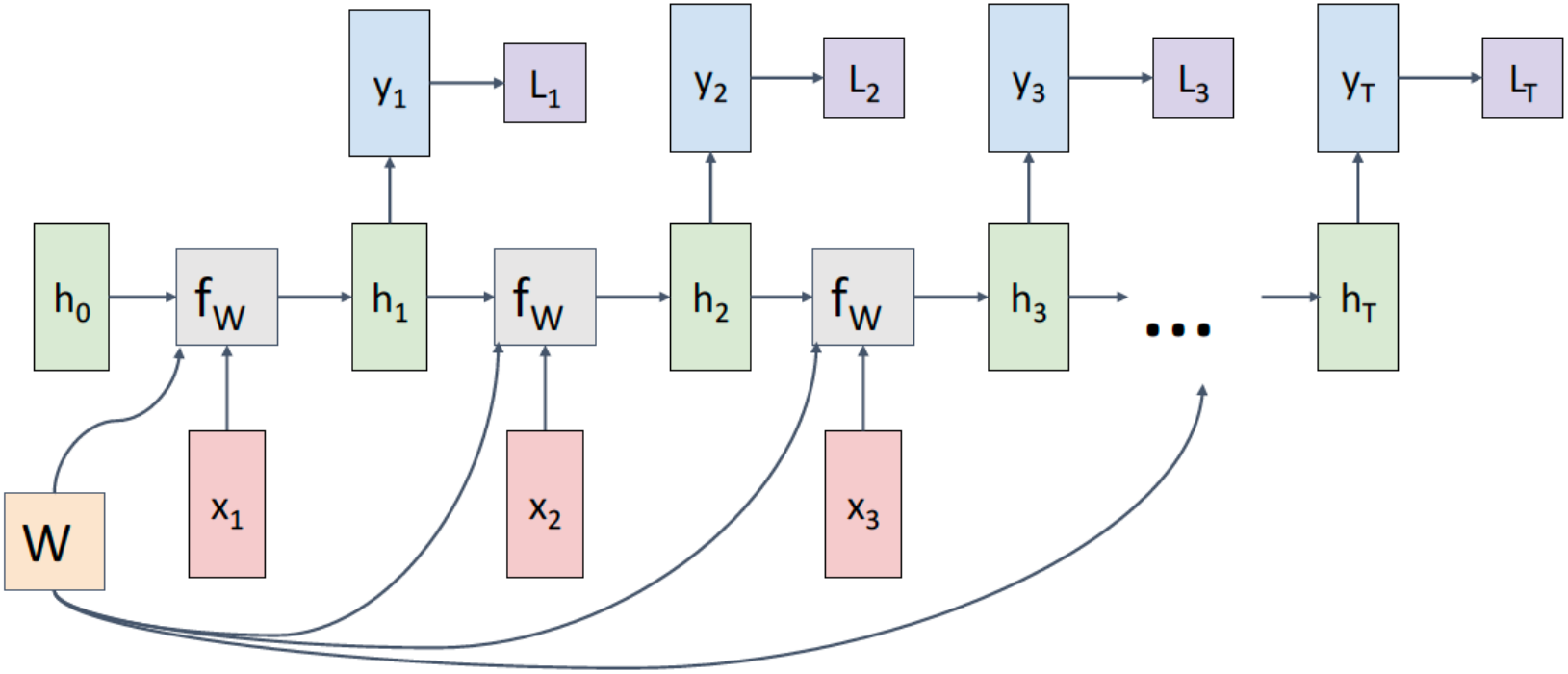
(also bias term)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN Computational Graph



Application

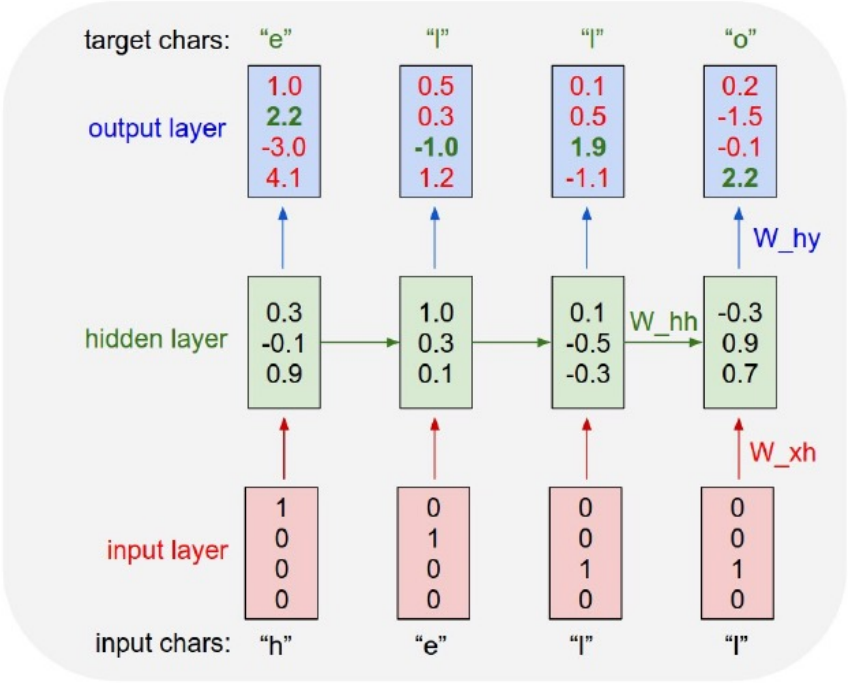
Example: Language Modeling

Given characters 1, 2, …, t,
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



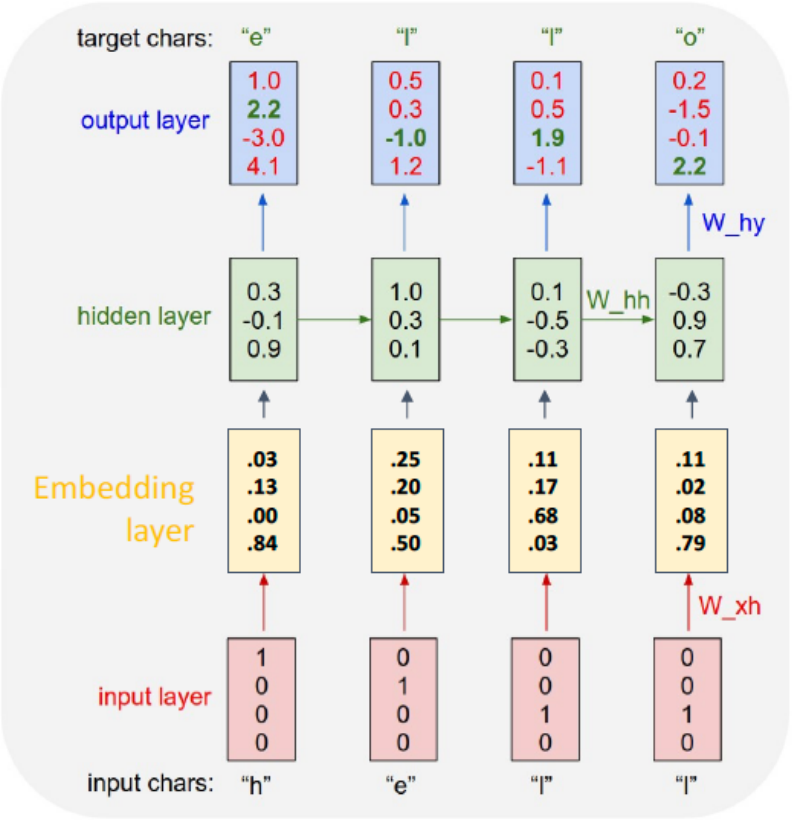
Application

Example: Language Modeling

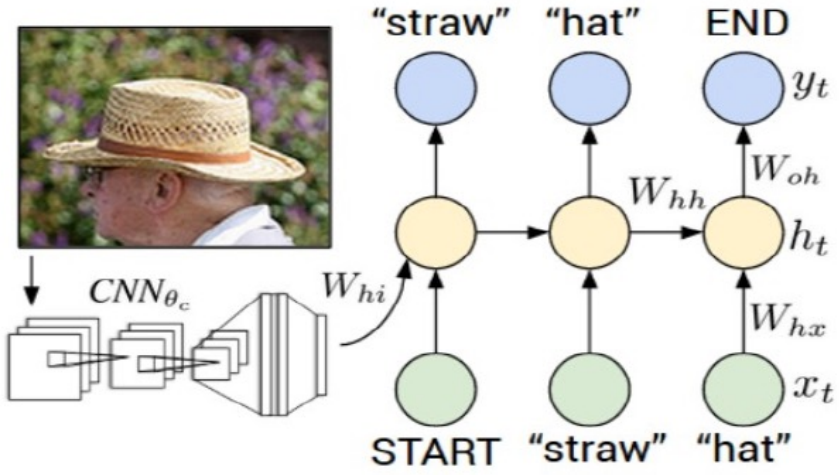
So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



Example: Image Captioning



Application: Image Captioning

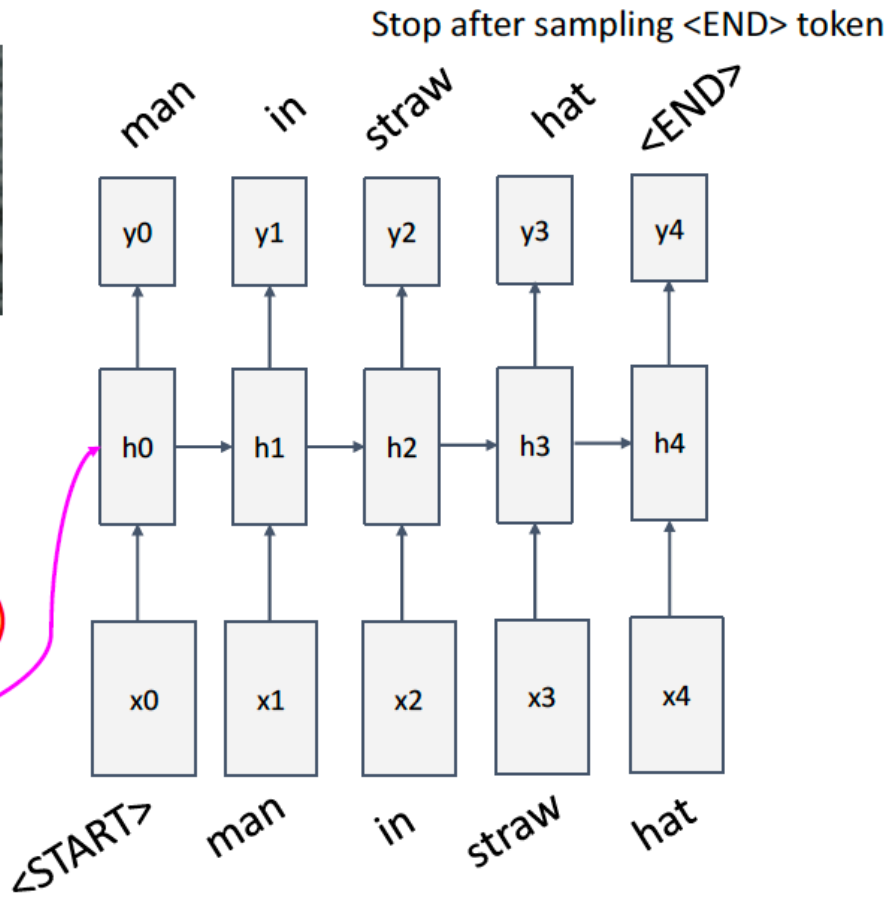
before:

$h = \tanh(W_{xh} * x + W_{hh} * h)$

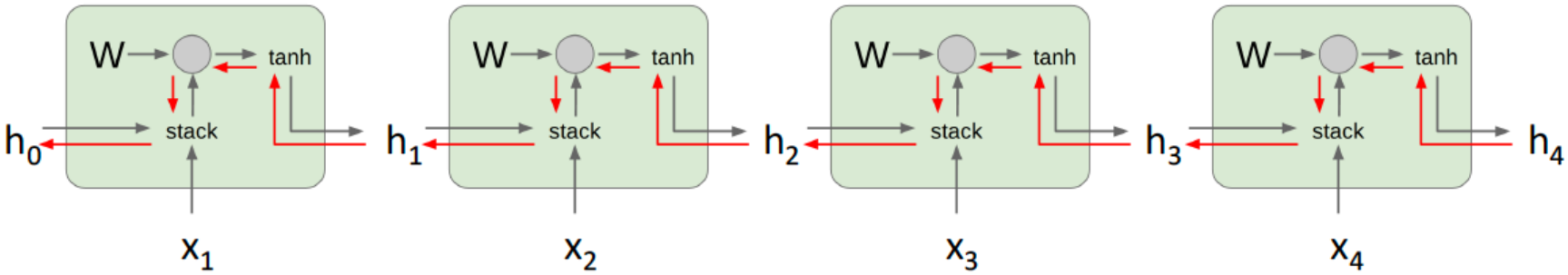
now:

$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$

W_{ih}



Vanishing Gradient Problem



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Long Short Term Memory

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Compute four **gates** at each timestep

Unsupervised Learning (Not On Midterm)

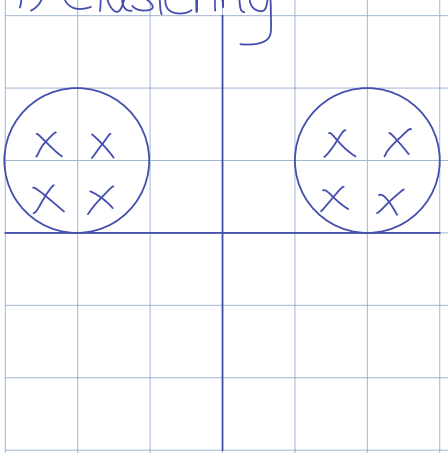
Training Set

$D = \{x_1, x_2, \ldots, x_N\}$

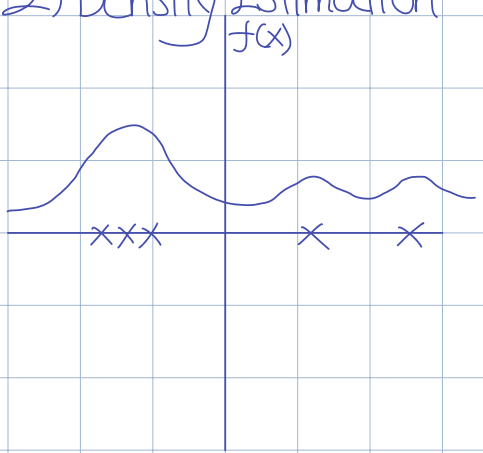
No Labels: $y_1, y_2, \ldots, y_n$

1) Clustering
2) Density Estimation
3) Dimensionality Reduction

1) Clustering

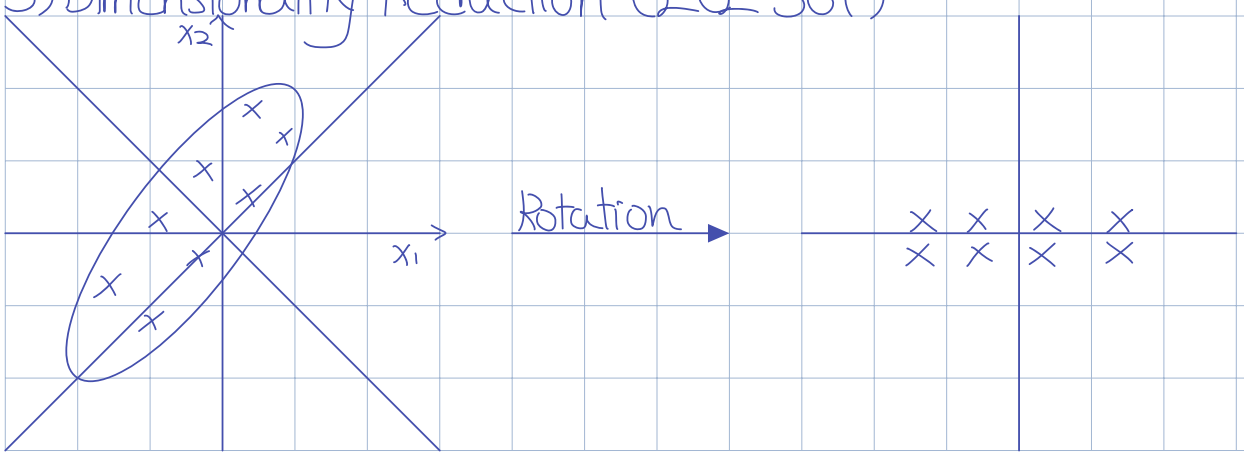


2) Density Estimation



(Matrix Analysis & Optimization)

3) Dimensionality Reduction (ECE 367)



Clustering

Partition training dataset D into k disjoint clusters such that elements within each cluster are close to one another.

Input, $D = \{x_1, x_2, \ldots, x_n\}$

k = # of clusters ($k << N$)

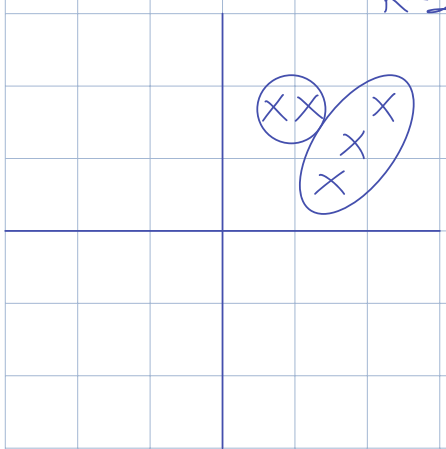
Output, Clusters $S_1, S_2, \ldots, S_k$

$S_i \subseteq D$

$S_i \cap S_j = \emptyset \ (i \neq j)$

$S_1 \cup S_2 \cup \ldots \cup S_k = D$

$K=2$



Cluster Center

Let μ_i be the cluster center for S_i

Approximation error for cluster S_j with cluster center μ_j

$$E_j(S_j, \mu_j) = \sum_{x_n \in S_j} \|x_n - \mu_j\|^2$$

Total Approximation Error

$$E_{in}(S_1, S_2, \dots, S_K, \mu_1, \mu_2, \dots, \mu_K) = \sum_{j=1}^{K} E_j(S_j, \mu_j)$$

$$= \sum_{j=1}^{K} \sum_{x_n \in S_j} \|x_n - \mu_j\|^2$$

$$= \sum_{n=1}^{N} \|x_n - \mu(x_n)\|^2$$

$\mu(x_n)$: Center of cluster associated with x_n

Optimal Clustering

$$\underset{\substack{S_1, \dots, S_K \\ \mu_1, \dots, \mu_K}}{\text{minimize}} E_{in}(S_1, S_2, \dots, S_K, \mu_1, \mu_2, \dots, \mu_K)$$

NP Hard (Bad News)

Excelent Heuristics (Good News)

Find Local Minimum

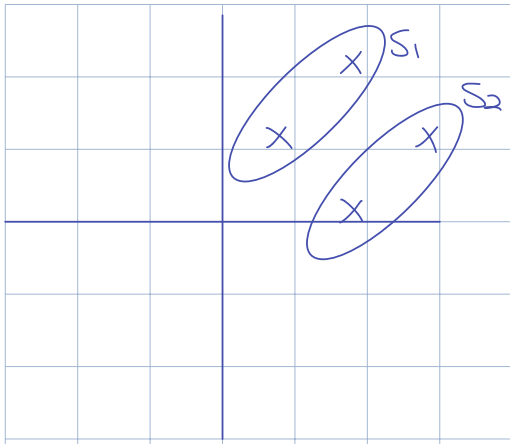
K Means Algorithm

Problem 1, Given $S_1, S_2, \dots, S_K$,

Find $\mu_1, \mu_2, \dots, \mu_K$ to minimize approximation error

Select μ_j to minimize

$$E_j(S_j, \mu_j) = \sum_{x_n \in S_j} \|x_n - \mu_j\|^2$$



$E_1(S_1, \mu_1) + E_2(S_2, \mu_2)$

$\mu_1 = \frac{1}{2}(x_1 + x_2)$

$\nabla_{\mu_j} E_j(S_j, \mu_j) = 0$

$\nabla_{\mu_j} \sum_{x_n \in S_j} \|x_n - \mu_j\|^2 = 0$

$\mu_j = \frac{1}{S_j} \sum_{x_n \in S_j} x_n$

number of points in S_j

Given $\mu_1, \mu_2, \ldots, \mu_K$ (Problem 2)
Find $S_1, S_2, \ldots, S_K$ to minimize

$$E_{in}(S_1, S_2, \ldots, S_K, \mu_1, \mu_2, \ldots, \mu_K) = \sum_{j=1}^{K} E_j(S_j, \mu_j)$$

$$= \sum_{n=1}^{N} \|x_n - \mu(x_n)\|^2$$

Select closest point

$x_n \in S_{j^*}$

if $j^* = \underset{j \in \{1,2,\ldots,k\}}{\operatorname{argmin}} \|x_n - \mu_j\|^2$