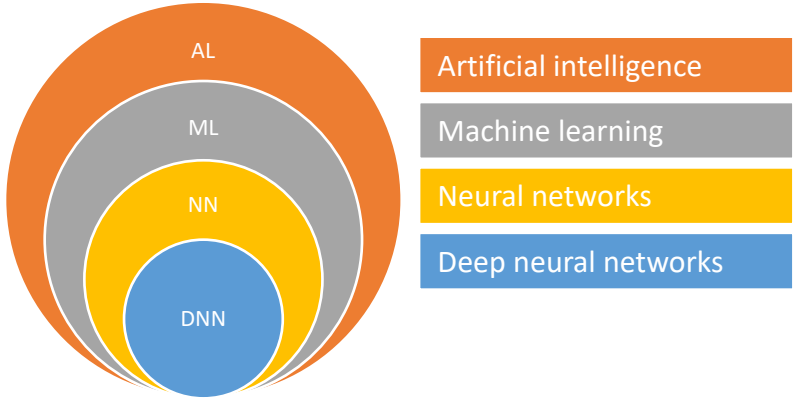
Deep Learning for Vision I

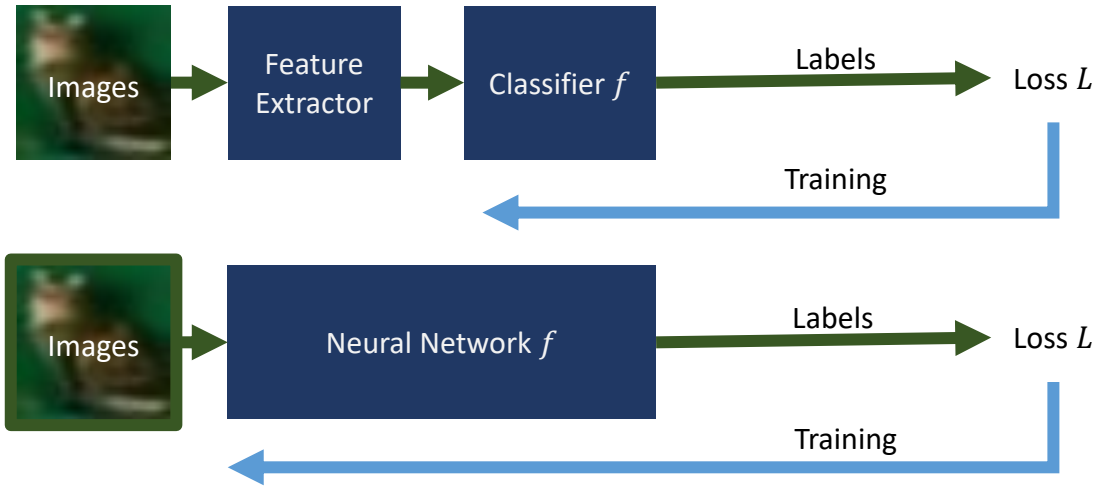
Presented by Ranga Rodrigo

Slides are from the sources particularly from Justin Johnson, Derek Hoiem and Svetlana Lazebnik.

WHAT IS DEEP LEARNING

Deep learning is a machine learning advancement that allows the system to automatically discover (learn) representations (representation learning) using a stack of multiple (deep) layers (of neurons, a neural network).





WHY NOW

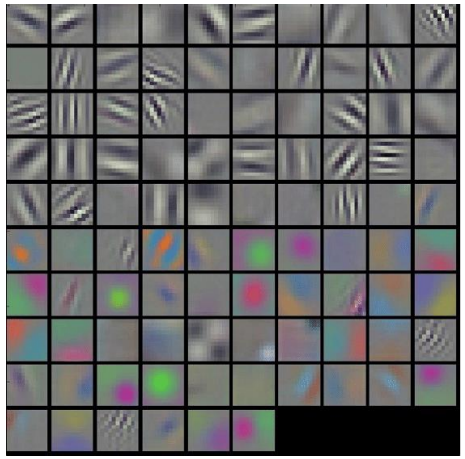
Research → industry
Computation power

Intel i9

Nvidia RTX 3090

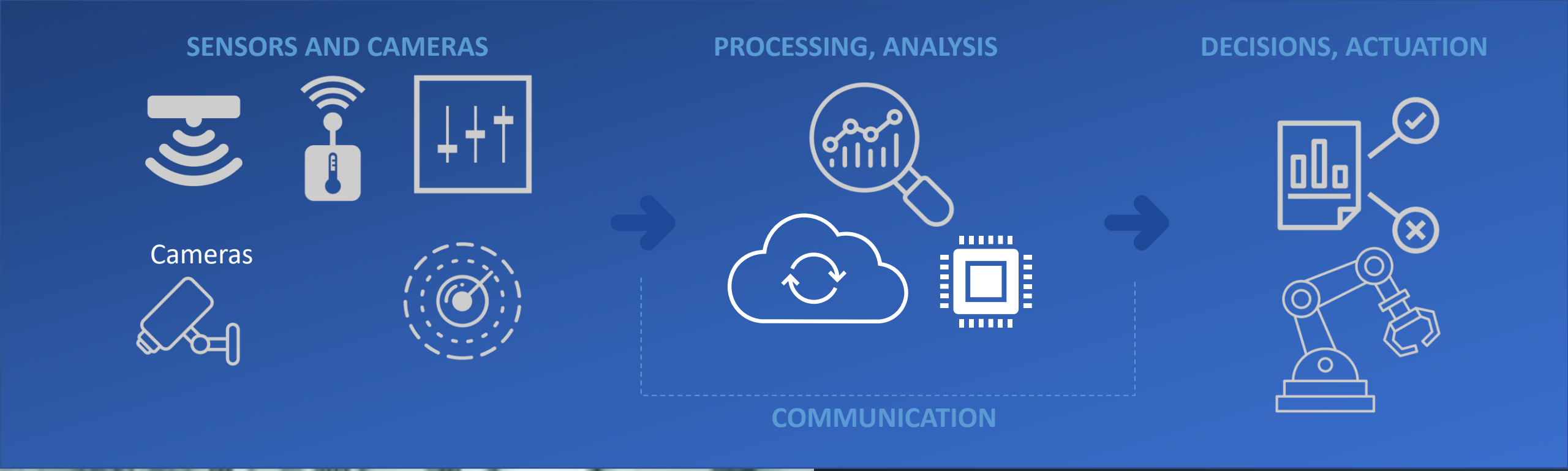
Nvidia A100

Linear classifier: Learnt filters



AlexNet layer 1, 11×11 weights

https://cs231n.github.io/linear-classify/
https://www.nvidia.com/en-us/data-center/a100/
https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090/
https://www.intel.com/content/www/us/en/products/details/processors/core/x.html



https://www.cognex.com/what-is/industry-4-0-machine-vision

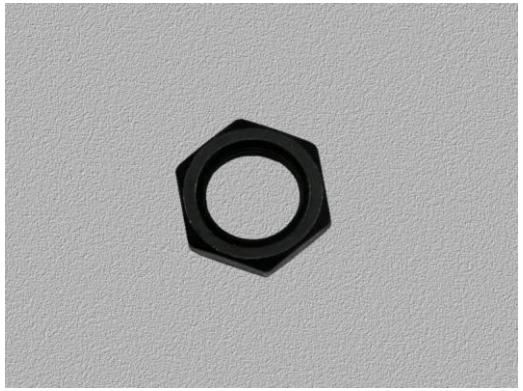
DEEP LEARNING APPLICATIONS IN MANUFACTURING

Vision based image classification
Vision based object detection
Vision based defect or anomaly detection
Vision based metrology
Time-series analysis for monitoring equipment

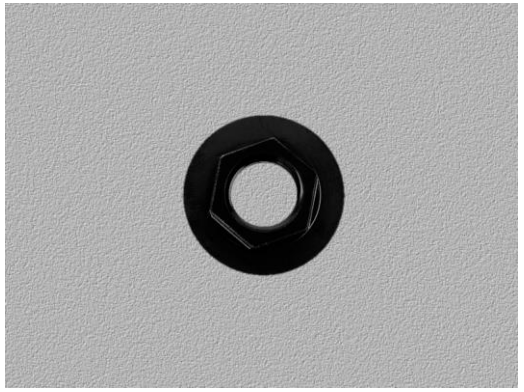
DEEP LEARNING APPLICATIONS IN OTHER DOMAINS

Self driving
Surveillance
Computer graphics
Natural language processing

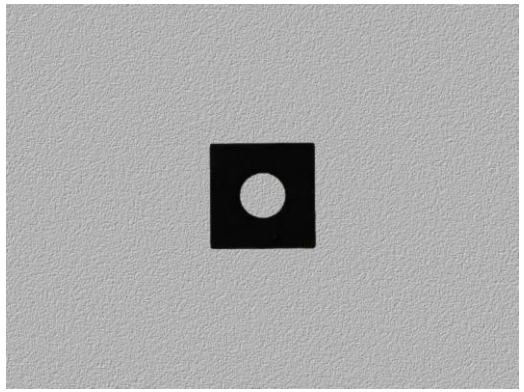
CLASSIFICATION



hex-nut



flanged hex-nut



square nut

OTHER TASKS

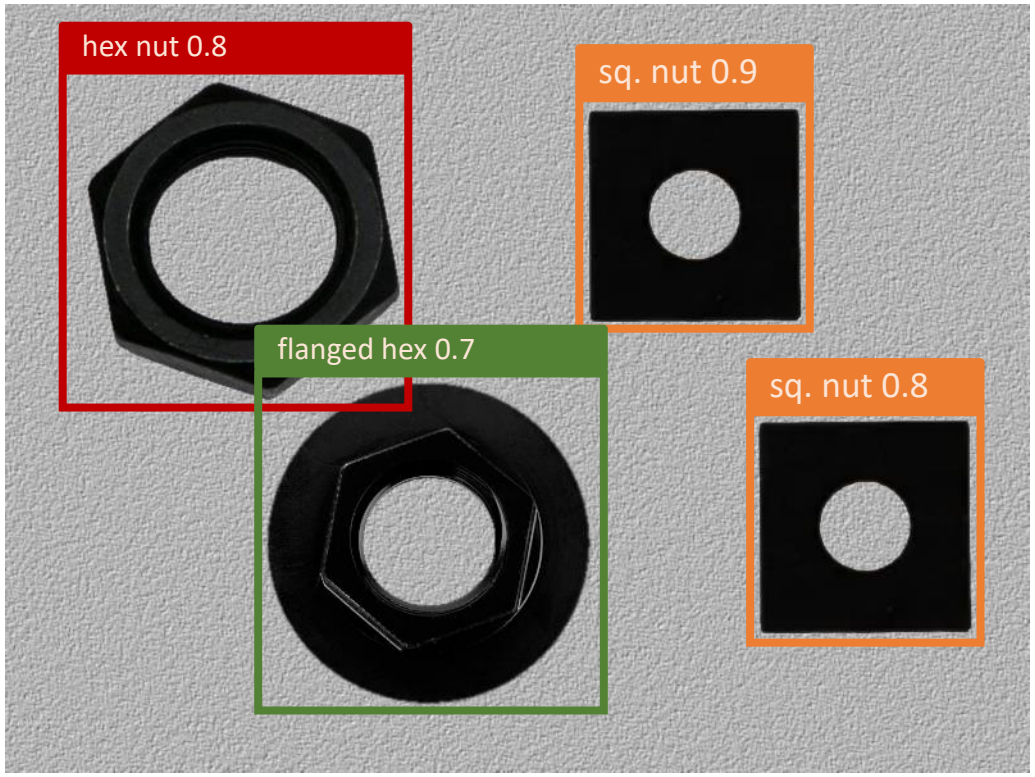
Sematic segmentation

Panoptic segmentation

Examples in MVTec

https://www.mvtec.com/products/deep-learning-tool

OBJECT DETECTION



INSTANCE SEGMENTATION

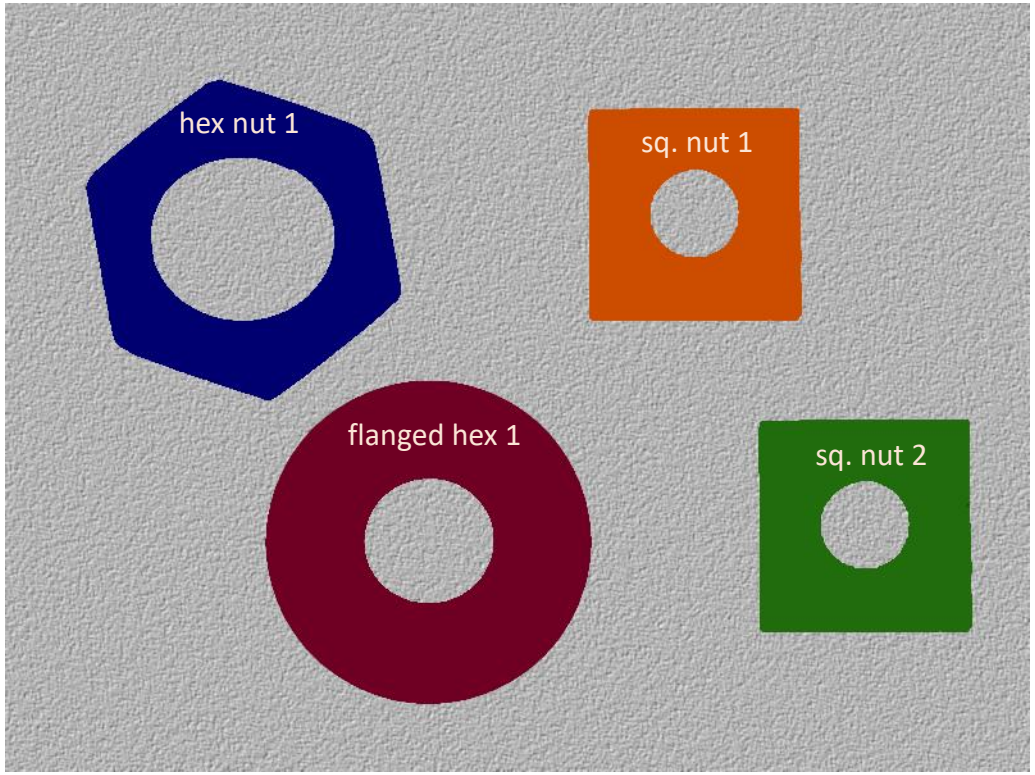


Image Classification on ImageNet

https://paperswithcode.com/sota/image-classification-on-imagenet

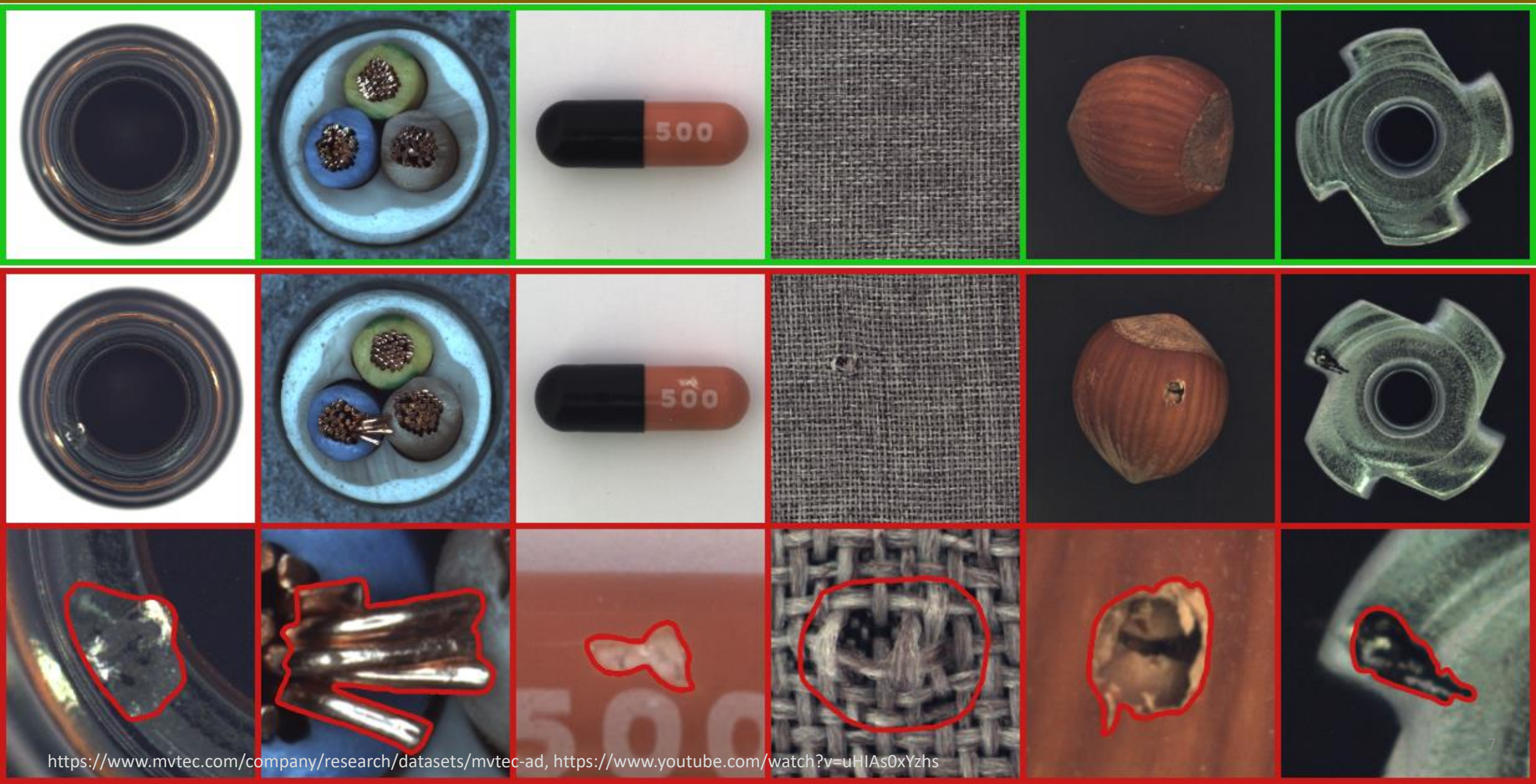
Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**Model soups** (ViT-G/14)	90.94%		1843M	✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	Transformer, JFT-3B
2	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer, JFT-3B
3	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer, JFT-3B
4	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer, JFT-3B
5	**V-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer
6	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels			2021	EfficientNet, JFT-300M

Object Detection on COCO test-dev

https://paperswithcode.com/sota/object-detection-on-coco

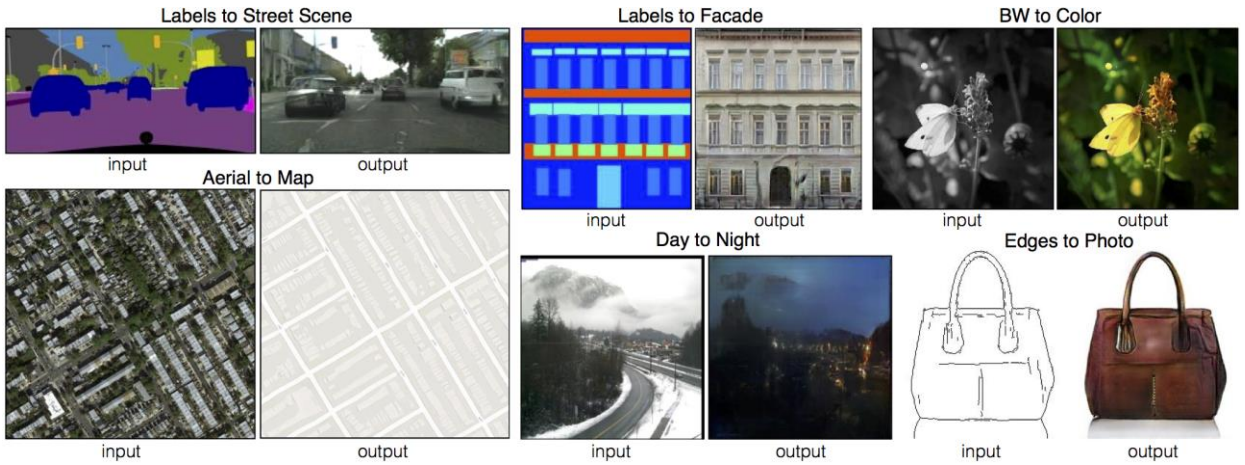
Rank	Model	box AP	AP50	AP75	APS	APM	APL	AP	Params (M)	Extra Training Data	Paper	Code	Result	Year	Tags
1	**DINO** (Swin-L, multi-scale)	63.3	80.8	69.9	46.7	66.0	76.5			×	DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection			2022	multiscale, Swin-Transformer, End-to-End, Query Denoising
2	**DINO** (Swin-L, single-scale)	63.2								×	DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection			2022	Swin-Transformer, End-to-End, Query Denoising
3	**SwinV2-G** (HTC++)	63.1								×	Swin Transformer V2: Scaling Up Capacity and Resolution			2021	multiscale, Swin-Transformer
4	**Florence-CoSwin-H**	62.4								×	Florence: A New Foundation Model for Computer Vision			2021	Swin-Transformer
5	**GLIP** (Swin-L, multi-scale)	61.5	79.5	67.7	45.3	64.9	75.0			×	Grounded Language-Image			2021	multiscale, Vision Language, Dynamic Head

ANOMALY DETECTION



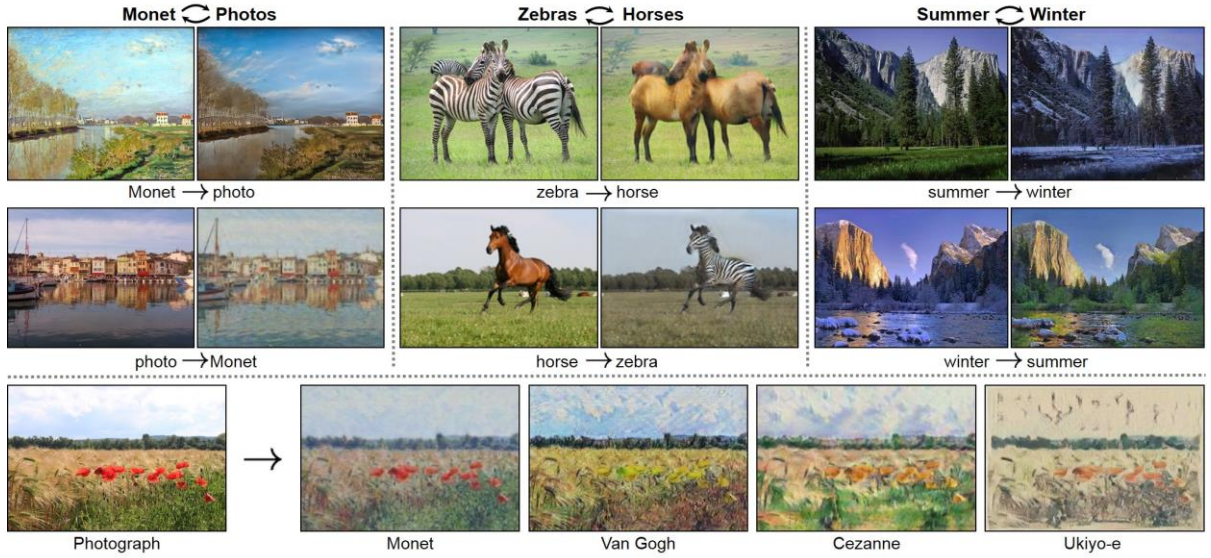
https://www.mvtec.com/company/research/datasets/mvtec-ad, https://www.youtube.com/watch?v=uHIAs0xYzhs

SYNTHESIS



pix2Pix: Isola et al., Image-to-Image Translation with Conditional Adversarial Nets, CAPR2017

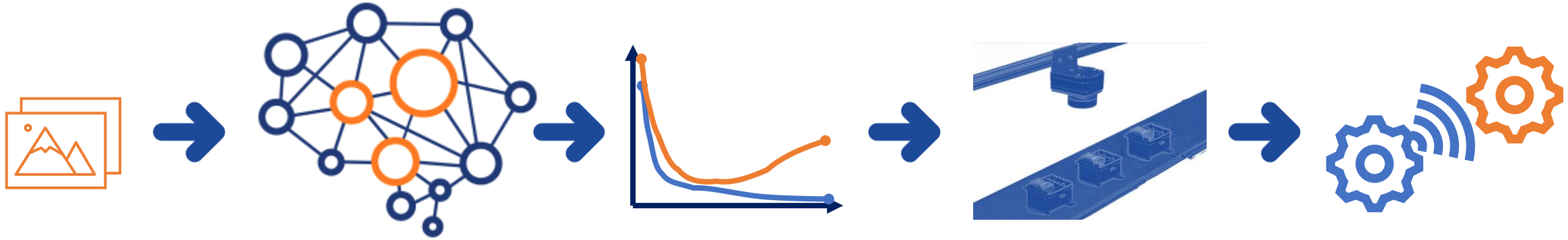
Brock et al, Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR2019.



CycleGAN: Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, IUCCV2017

Saham et al., SinGAN: Learning a Generative Model from a Single Natural Image, ICCV2019

PROCESS



PREPARATION | TRAINING | VALIDATION | INFERNCE | COMMUNICATION M2M, actuation or actionable insights

HARDWARE

Good cameras and lenses: E.g., Basler, Cognex, FLIR

Lighting and imaging setup

Processing hardware (training, inference): GPUs (see slide 2), specialized boards (e.g., Jetson), or cloud

JETSON TX2

Jetson AGX Xavier

Example Basler Camera

Example Basler Lighting

https://www.youtube.com/watch?v=uHIAs0xYzhs, https://icon-library.com/
https://www.cognex.com/what-is/machine-vision/components/lighting, https://www.baslerweb.com/en/products/vision-components/basler-lighting/

History of Deep Convolution Networks

- 1950s: neural nets (perceptron) invented by Rosenblatt
- 1980s - 1990s: Neural nets are popularized and then abandoned as being interesting idea but impossible to optimize or “unprincipled”
- 1990s: LeCun achieves state-of-art performance on character recognition with convolutional network (main ideas of today’s networks)
- 2000s: Hinton, Bottou, Bengio, LeCun, Ng, and others keep trying stuff with deep networks but without much traction/acclaim in vision
- 2010-2011: Substantial progress in some areas, but vision community still unconvinced
- Some neural net researchers get ANGRY at being ignored/rejected
- 2012: shock at ECCV 2012 with ImageNet challenge

Slide: Derek Hoiem

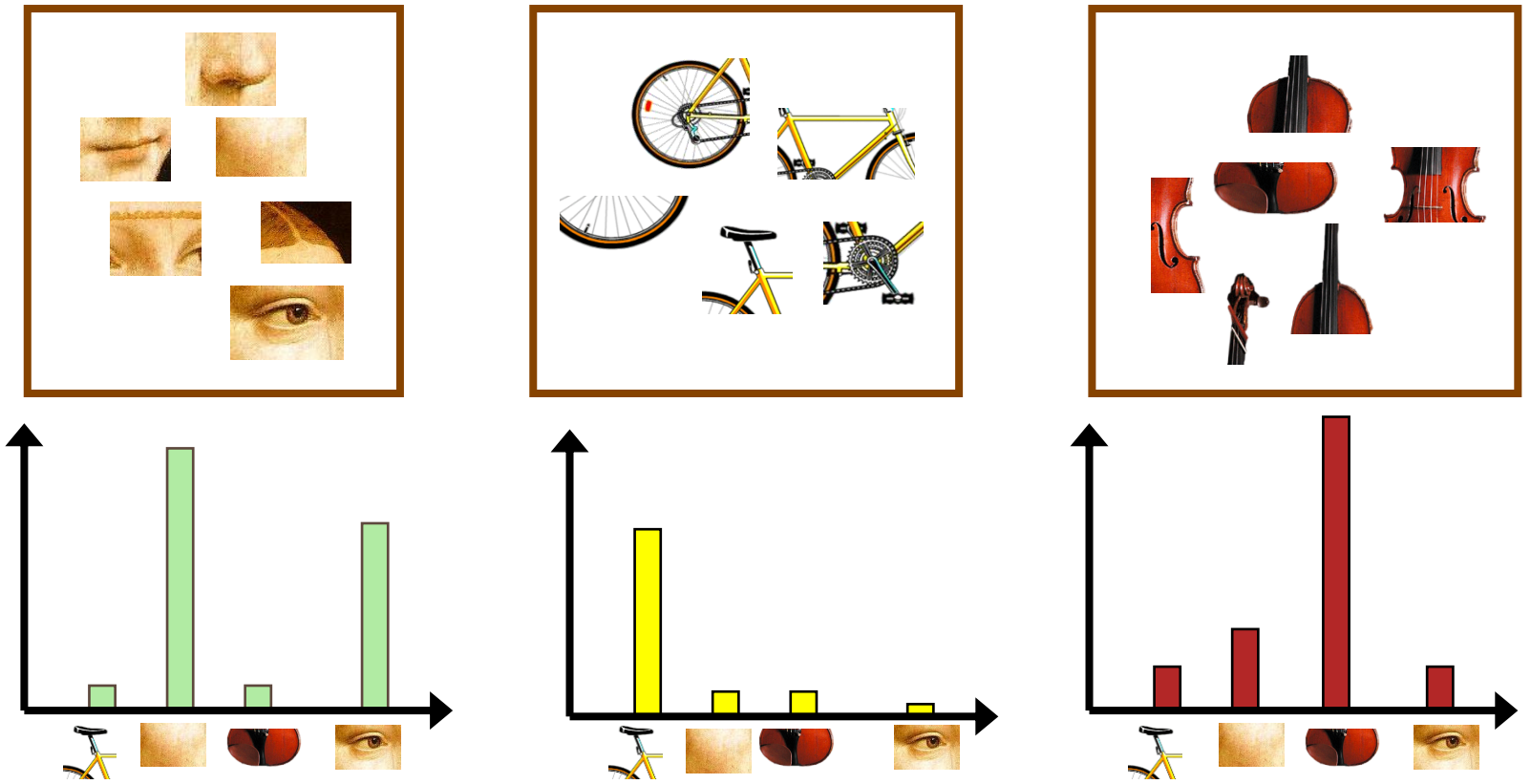
Prior to CNNs

Image Classification

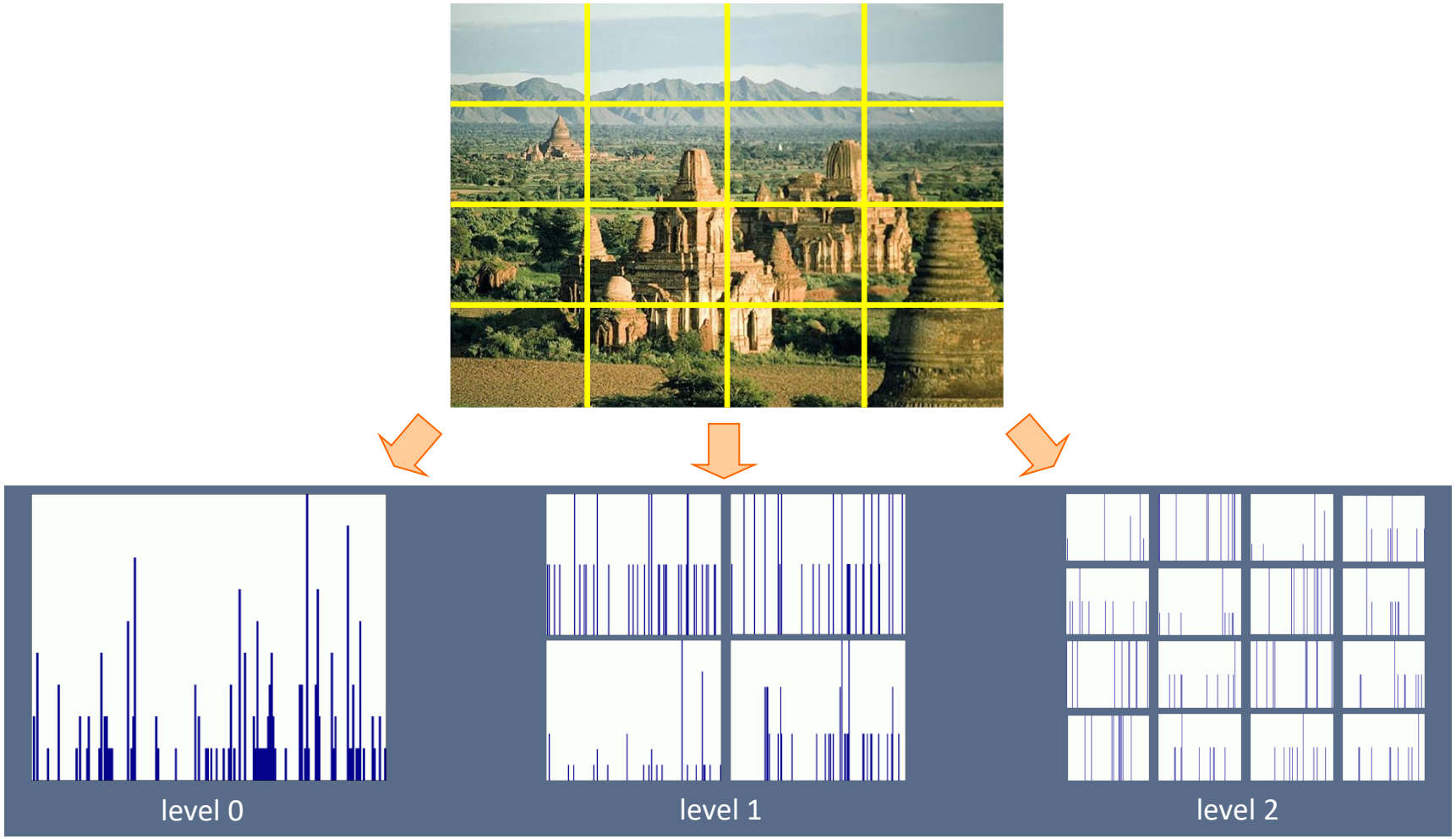
- Color histograms: throw out spatial information.
- Histograms of Oriented Gradients (HOG): uses spatial information, however, they are templates of gradient.
- Bags of Visual Words (BoVW): data-driven.
- We can concatenate features representations to get a long high-dimensional vector.
- Feature extractor and learnable model are distinct: only the last learnable part tunes itself to classify
- Deep learning: single end-to-end pipeline.
- A two-layer NN seems to compute features in the first layer and used them to classify inputs in the second layer.

Classification: Bag of Visual Words Features

1. Extract local features
2. Learn “visual vocabulary”
3. Quantize local features using visual vocabulary
4. Represent images by frequencies of “visual words”
5. Train a classifier (kNN, linear, SVM)



Spatial Pyramids



Lazebnik, Schmid & Ponce (CVPR 2006)

Linear Classification

We must define a score function, and a loss function.

Score function

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

$$f: R^D \mapsto R^K$$

$$x_i \in R^D, \quad i = 1, \ldots, N, \; y_i \in \{1, \ldots, K\}$$

Optimization

In order to find W and b, we can optimize the loss function.

- QP in SVM
- Gradient descent, with many layers, backpropagation to compute gradients.

Loss function:

Multi-Class SVM loss

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

Softmax loss

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

Squared Loss

$$L_i = (\hat{y}_i - y_i)^2$$

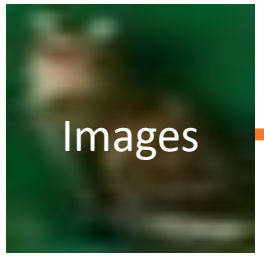
Regularization term needed.

$$R(W) = \sum_{k,l} w_{kl}^2$$

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

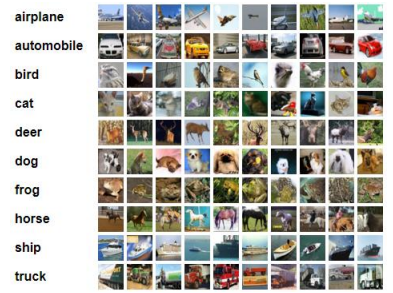
https://cs231n.github.io/linear-classify/

Classical Approach vs. Neural Networks



Feature Extractor

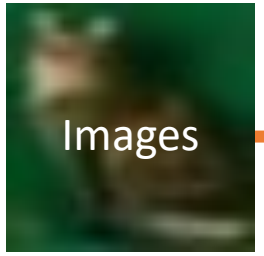
Classifier f



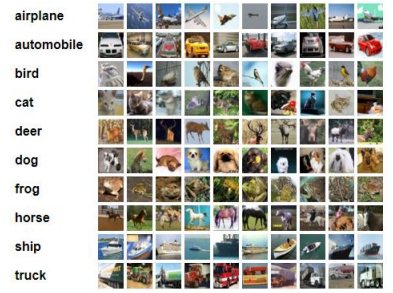
Labels

Loss L

Training



Neural network f

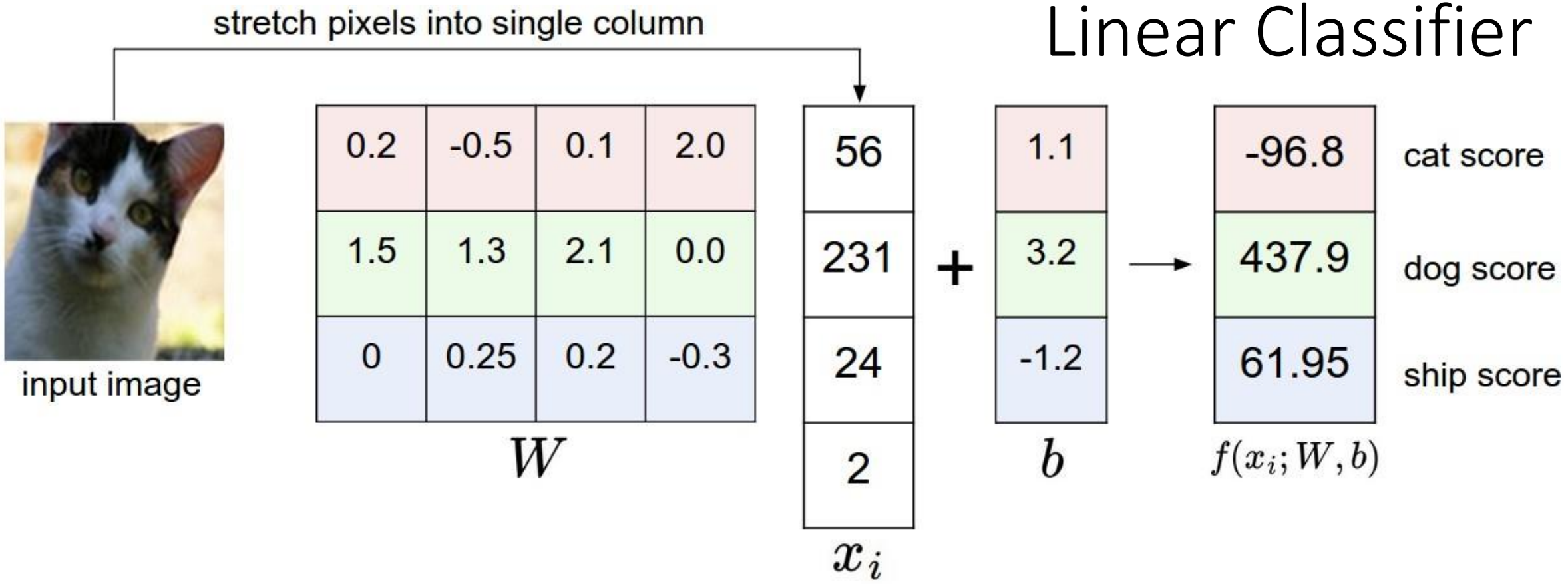


Labels

Loss L

Training

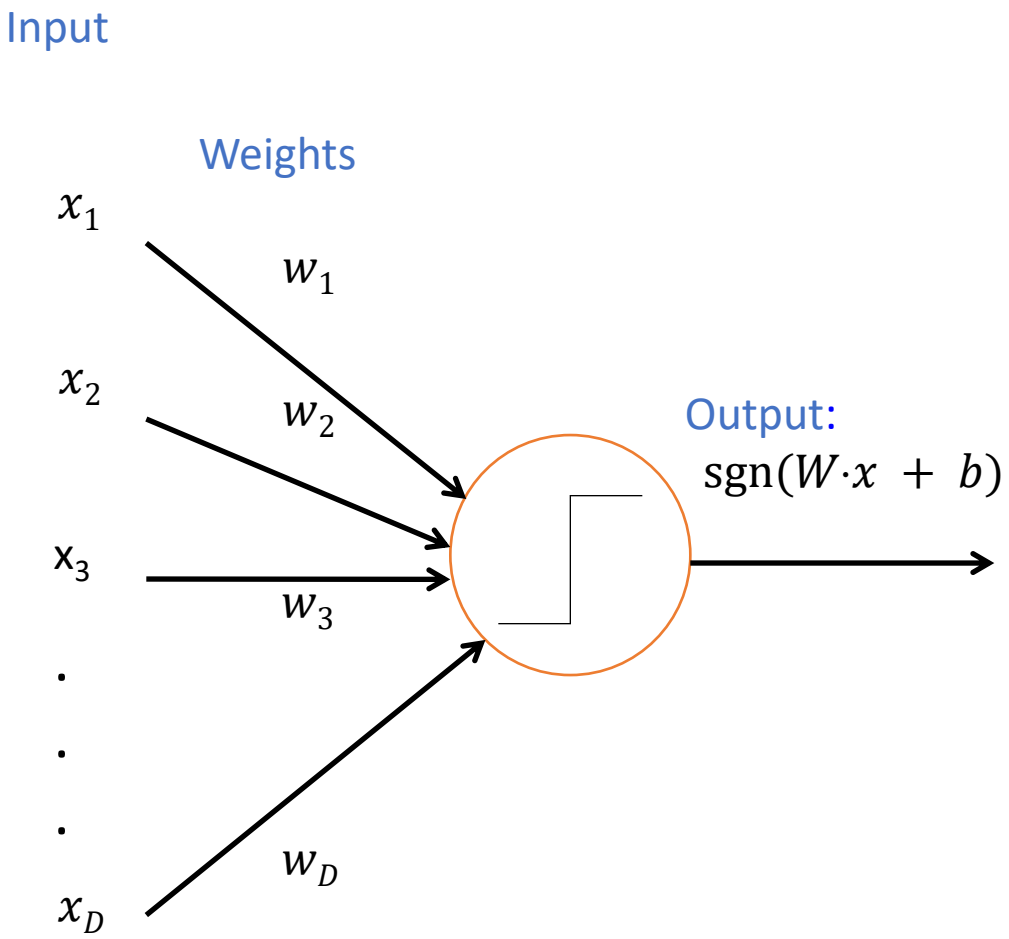
Linear Classifier



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.

https://cs231n.github.io/linear-classify/

The Perceptron



Perceptron is a linear classifier.

Capturing b also into W

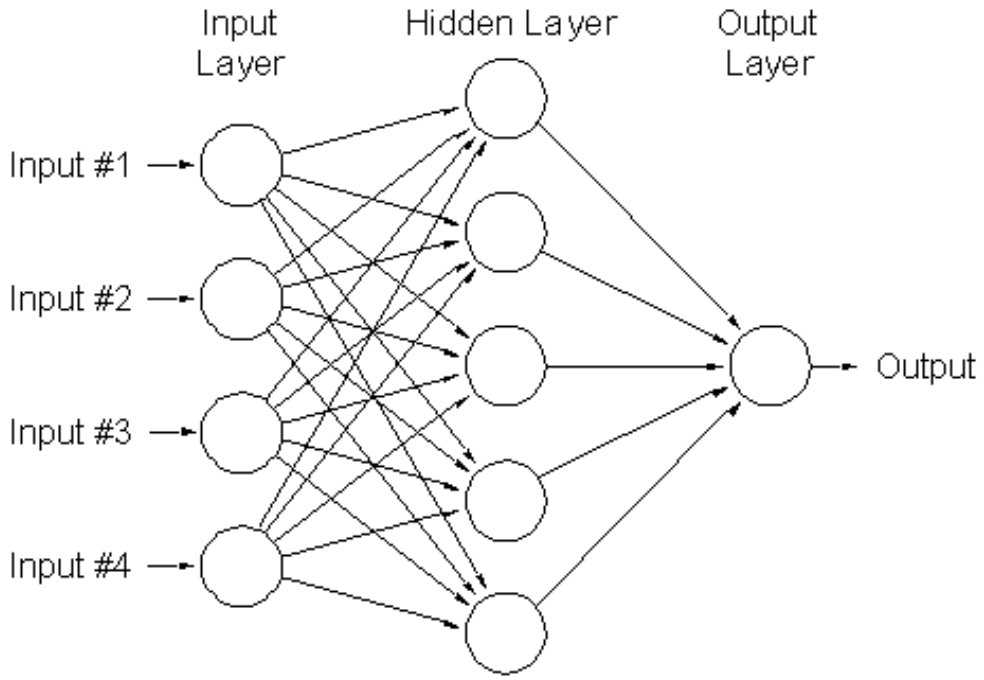
$f = \mathrm{sgn}(Wx), x \in \mathbb{R}^D, W \in \mathbb{R}^{K \times D}$

assuming K classes.

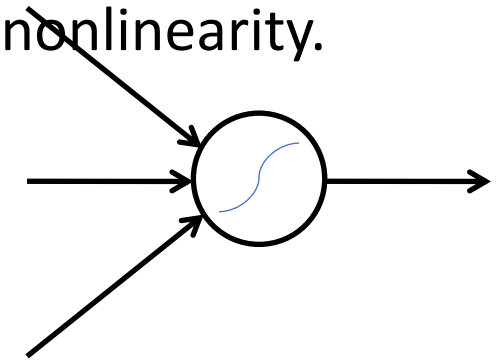
Not powerful.

Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408.

Two-Layer Neural Network



Can learn nonlinear functions provided each perceptron has a differentiable nonlinearity.



Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$

If we add one more layer

$f(x) = \text{sgn}(W_2 W_1 x), W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{K \times H}$

Still

$$f(x) = \text{sgn}(W_3 x), \text{where } W_3 = W_2 W_1$$

is a linear classifier.

To have a powerful classifiers, non-linearity at each layer is important.

If we add non linearities, we can learn a continuous boundary.

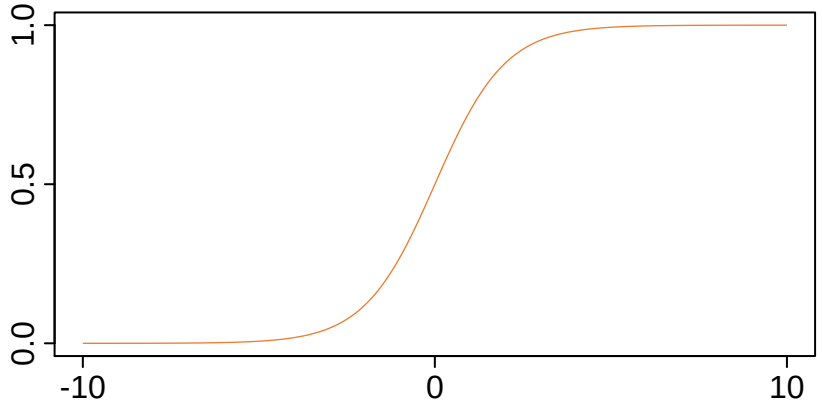
$$f(x) = \max(0, W_2 \max(0, W_1 x))$$

Now, neural net's first layer is bank of templates; Second layer recombines templates

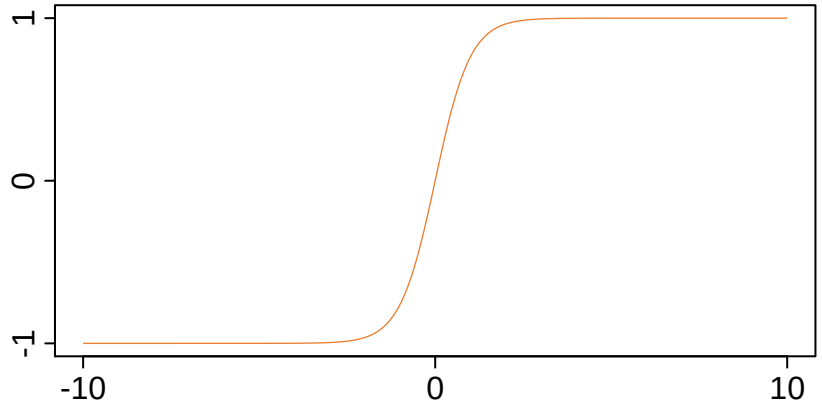
A part of the slide: Lazebnik

Activation Functions

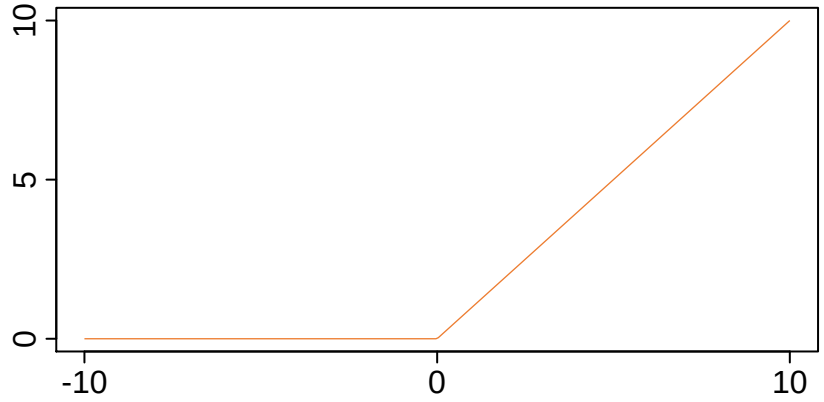
Sigmoid $\sigma(t) = \frac{1}{1 + e^{-t}}$



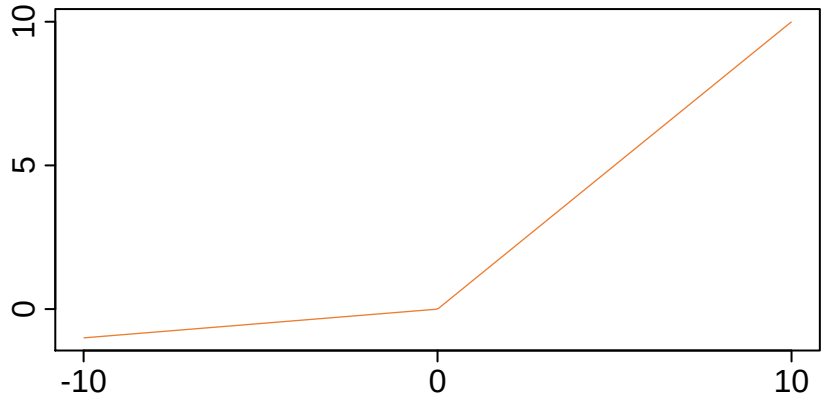
Tanh $\tanh(t)$



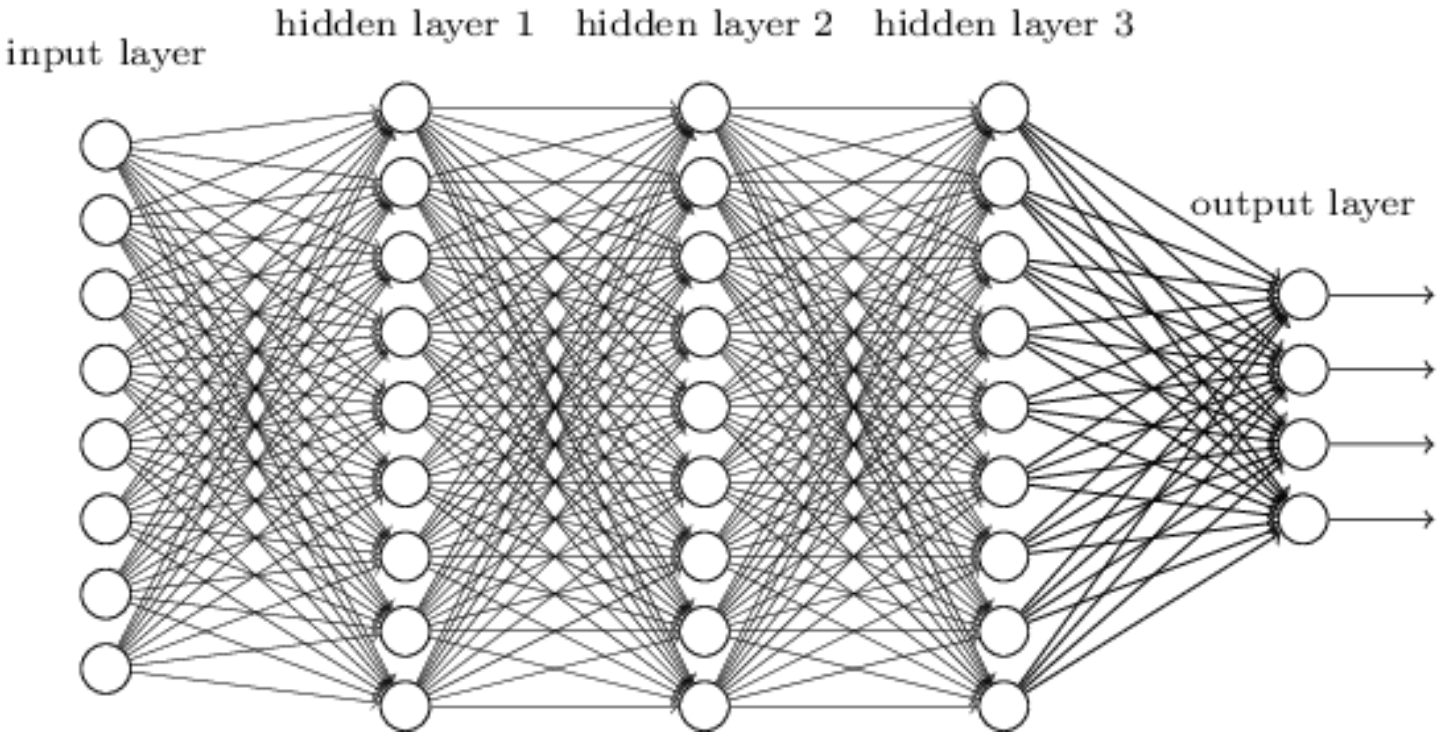
ReLU $\max(0, t)$



Leaky ReLU $\max(0.1t, t)$



Multi-Layer Neural Network



Slide: Lazebnik

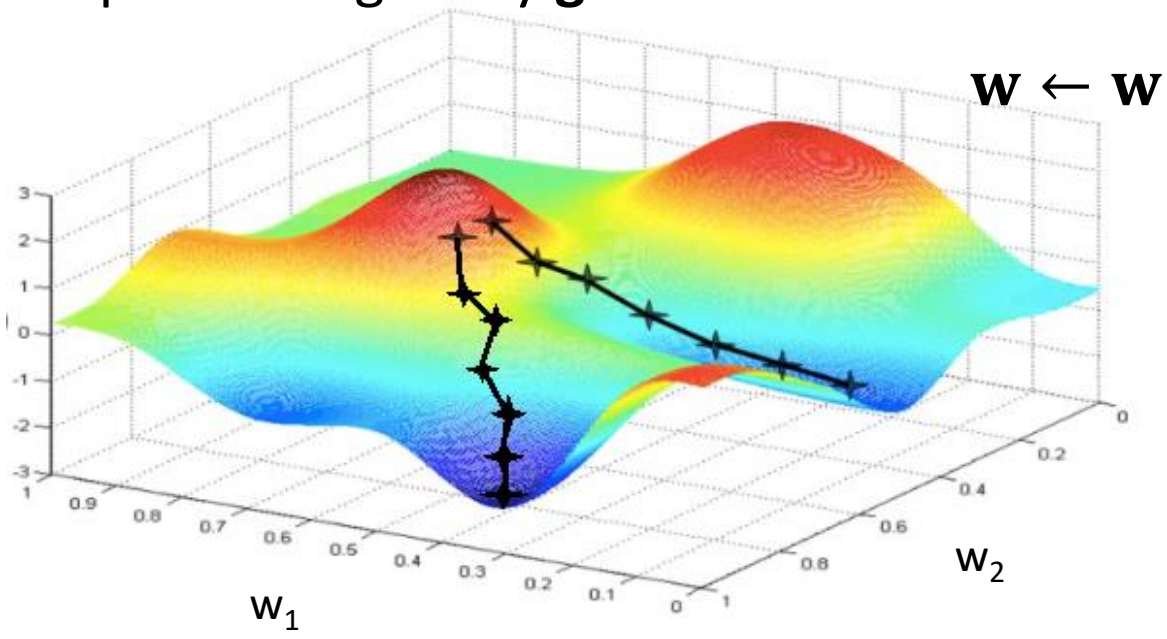
Training of Multi-Layer Networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f_{\mathbf{w}}(\mathbf{x}_i)\right)^2 + R(W)$$

- Update weights by **gradient descent:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$$



Slide: Lazebnik

Training of Multi-Layer Networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f_{\mathbf{w}}(\mathbf{x}_i)\right)^2 + R(W)$$

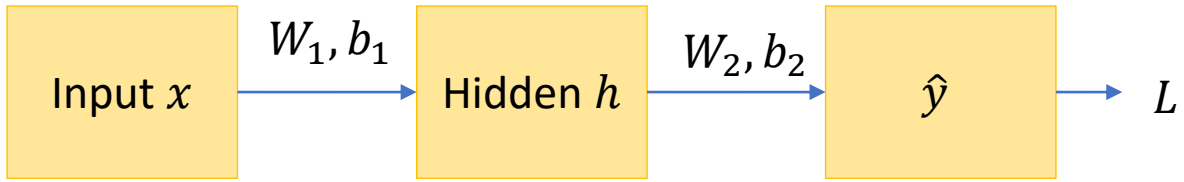
- Update weights by **gradient descent**:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\frac{\partial L}{\partial \mathbf{w}}$$

- **Back-propagation**: gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent**: compute the weight update w.r.t. a small batch of examples at a time, cycle through training examples in random order in multiple epochs

Slide: Lazebnik

Two-Layer Dense Network



- Input layer to hidden layer

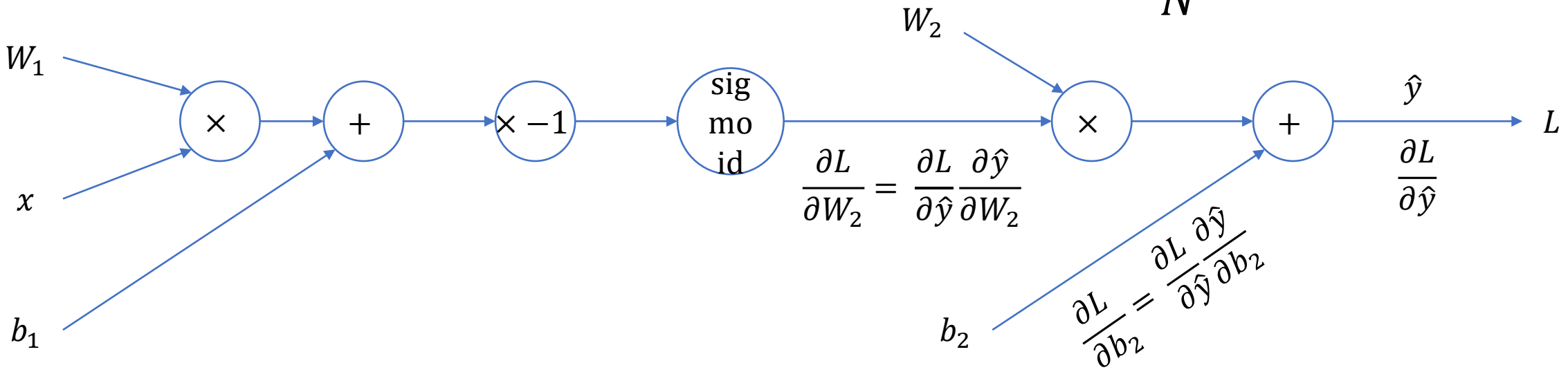
$$h(W_1, b_1) = \frac{1}{1 + \exp(-W_1 x - b_1)}$$

- Hidden layer to the output layer

$$\hat{y}(W_2, b_2) = W_2 h + b_2$$

- Loss

$$L = \frac{1}{N}\sum(\hat{y} - y)^2$$



Need for Backpropagation

- The requirement is to find the parameters W_1 and W_2 such that the loss L gets minimized.
- We use an optimization scheme based on gradient descent.
- For this, we need the gradients of L w.r.t. the parameters

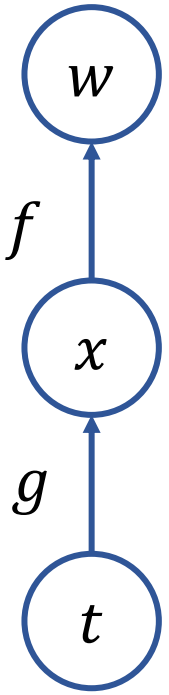
$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}.$$

- Then, we can learn W_1 and W_2.
- However, L is a composite function of W_1 and W_2.
- Backpropagation can efficiently compute these gradients.

Chain Rule

Intuitively, the chain rule states that knowing the instantaneous rate of change of w relative to x and that of x relative to t allows one to calculate the instantaneous rate of change of w relative to t.

$$w = f(x), x = g(t)$$
$$\frac{dw}{dt} = \frac{dz}{dx}\frac{dx}{dt}$$



As put by George F. Simmons: "if a car travels twice as fast as a bicycle and the bicycle is four times as fast as a walking man, then the car travels 2 × 4 = 8 times as fast as the man

Chain Rule

The simplest multivariable chain rule involves a function

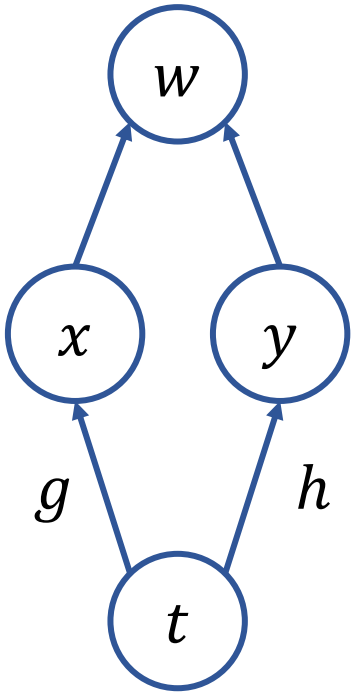
$$w = f(x, y)$$

of two variables x and y, w here x and y are each functions of another variable t, $x = g(t)$ and $y = h(t)$. Then w is a function of t,

$$w = f[g(t), h(t)]$$

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

In the situation it is convenient to call w the dependent variable, x and y the intermediate variables, and t the independent variable.

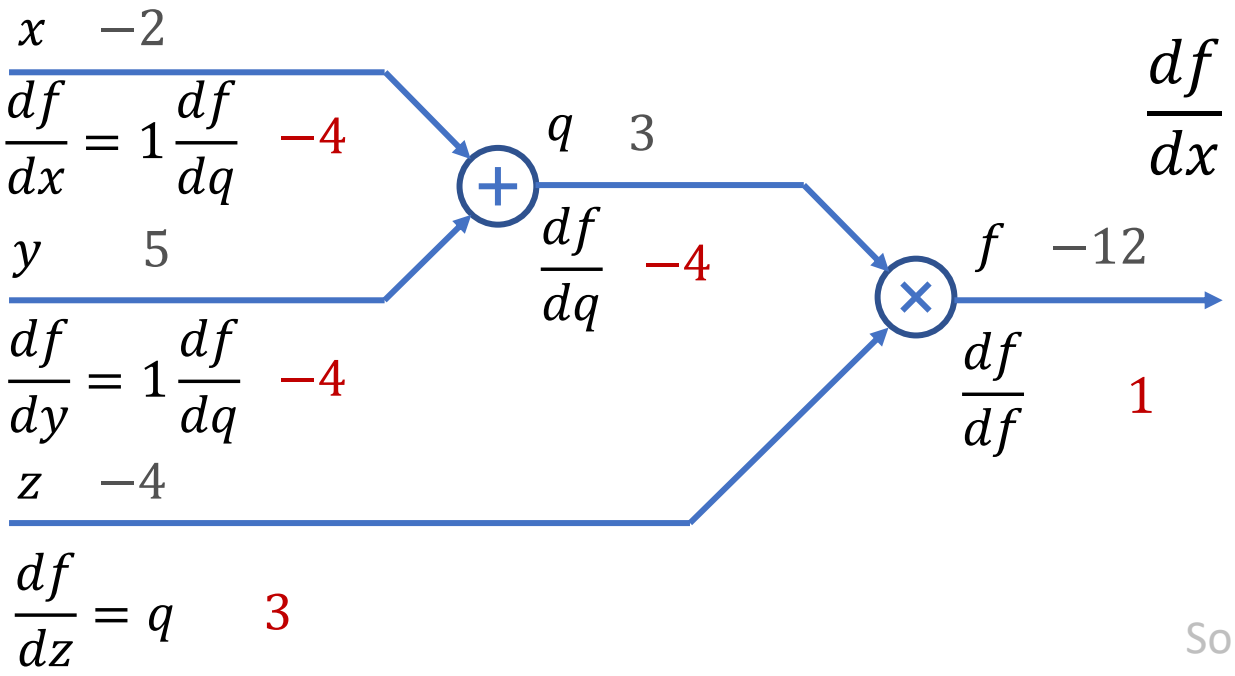


Chain Rule and Backpropagation

$$f(x,y,z) = (x + y)z$$
$$f(x,y,z) = qz$$
$$q = x + y$$
$$x = -2, y = 5, z = -4$$

Forward Pass

Backward Pass



$$\frac{df}{dz} = q$$

$$\frac{df}{dq} = z$$

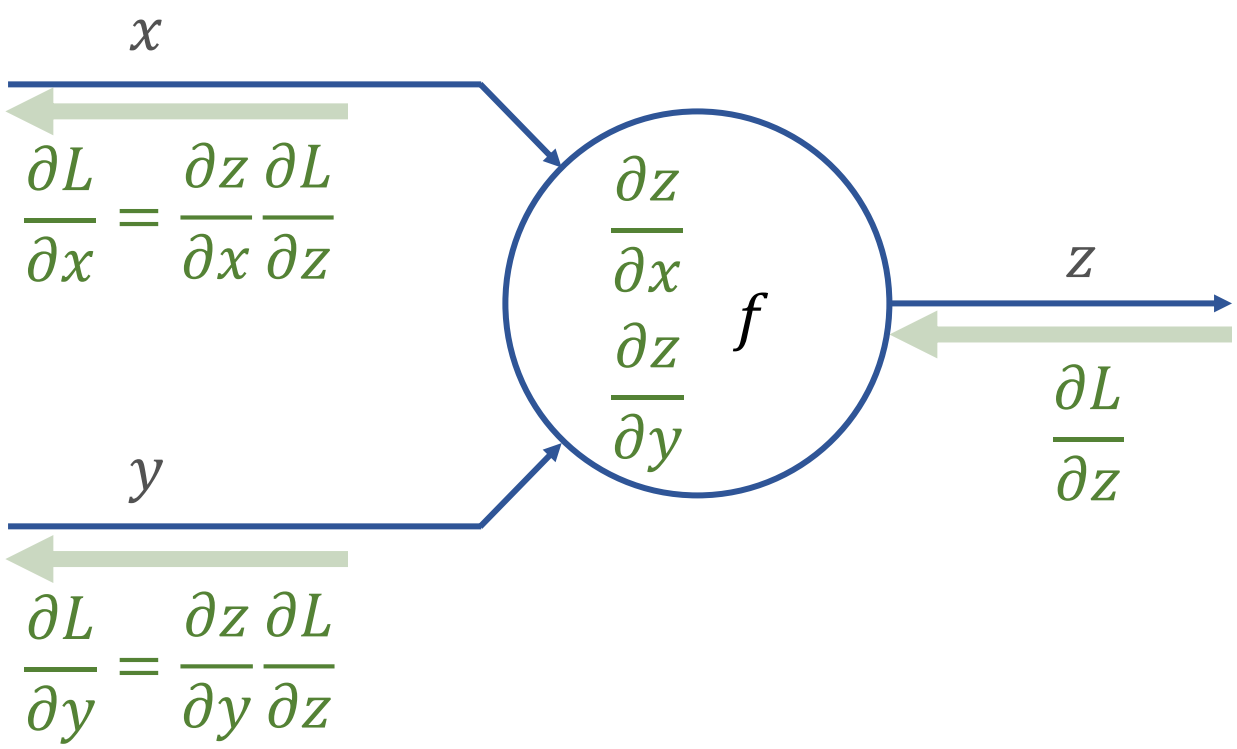
$$\frac{df}{dy} = \frac{df}{dq}\frac{dq}{dy} = 1\frac{df}{dq}$$

$$\frac{df}{dx} = \frac{df}{dq}\frac{dq}{dx} = 1\frac{df}{dq}$$

$$z = f(u,v)$$
$$u = u(t), v = v(t)$$
$$z = f\big(u(t), v(t)\big)$$

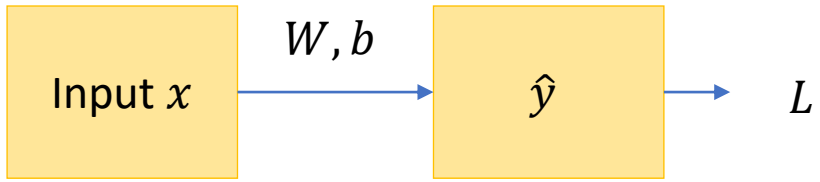
$$\frac{dz}{dt} = \frac{\partial z}{\partial u}\frac{du}{dt} + \frac{\partial z}{\partial v}\frac{dv}{dt}$$

Source: Johnson and https://cs231n.github.io/optimization-2/



Source: Johnson

Implementation Exercise: Linear Network

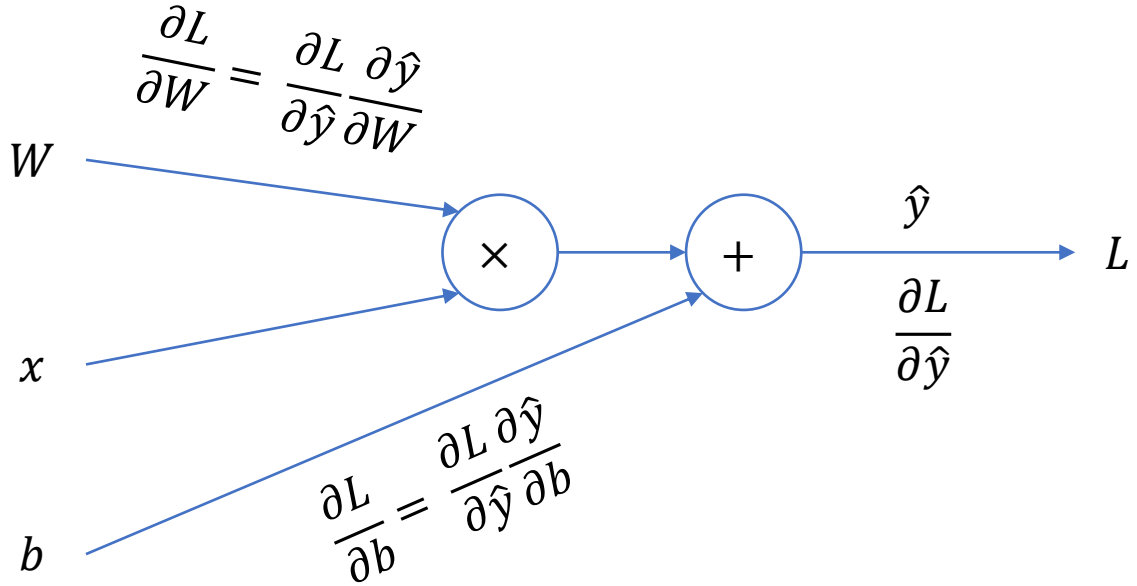


- Input layer to output layer

$$\hat{y}(W, b) = Wx + b$$

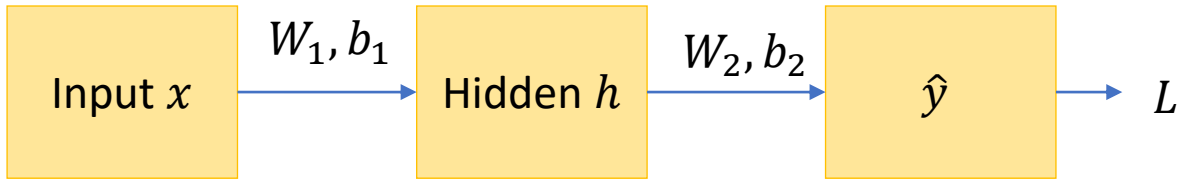
- Loss

$$L = \frac{1}{N}\sum(\hat{y} - y)^2$$

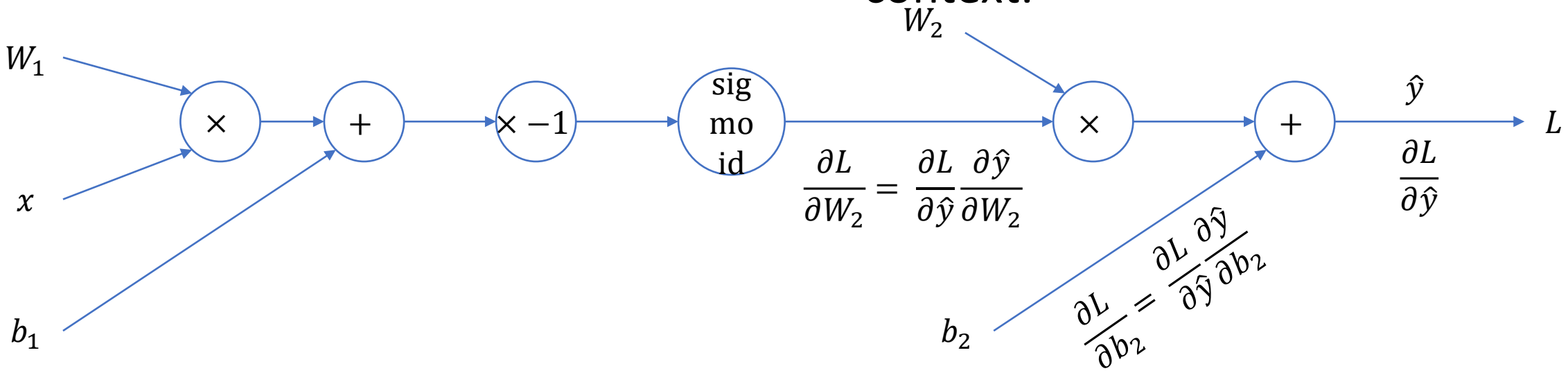


Convolutional Neural Networks

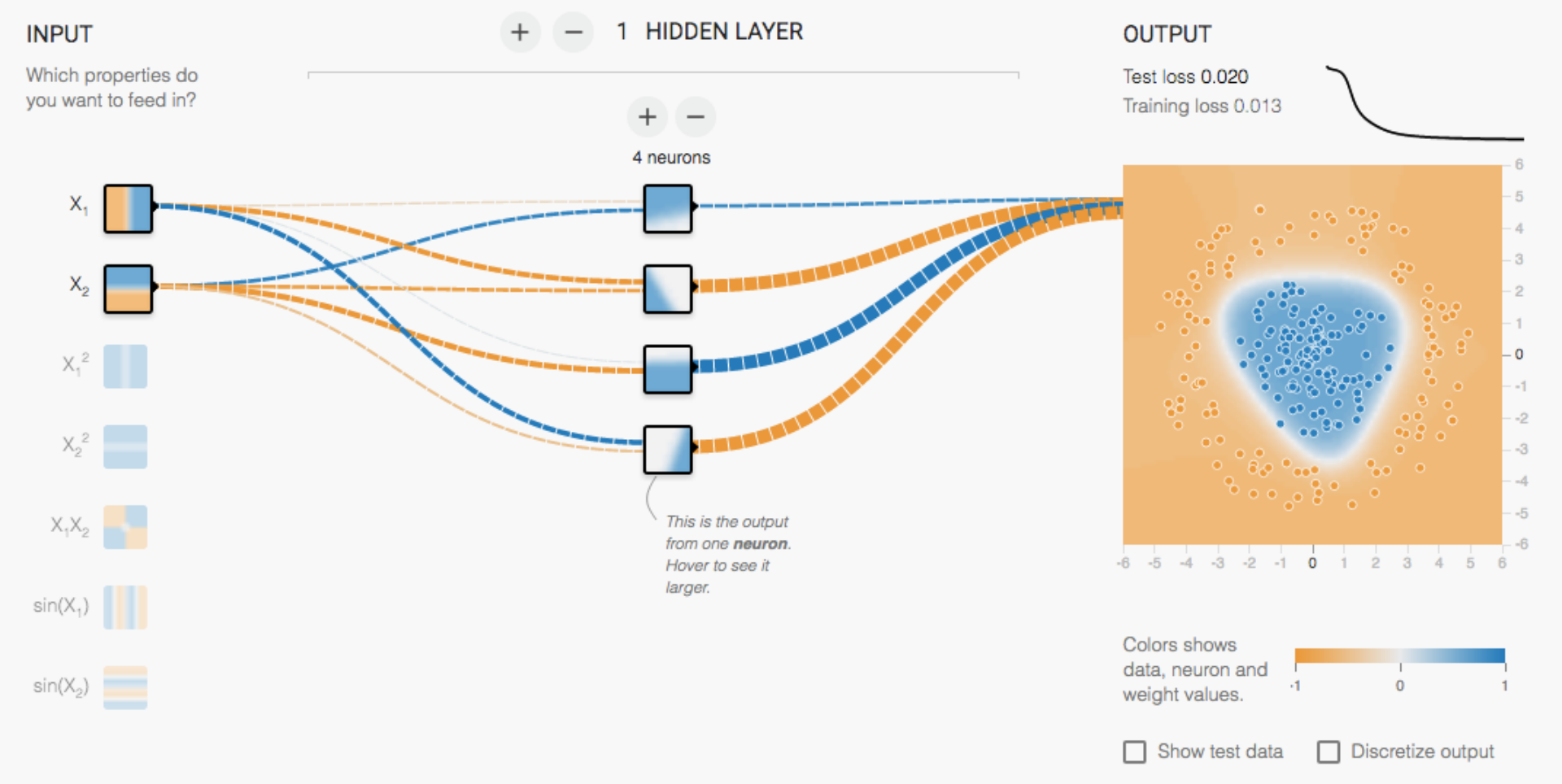
Recap.



- Fully-connected networks (MLPs) with non-linear activation functions can classify by approximating non-linear separating surfaces.
- We can train these using backpropagation.
- However, MLPs disregard special context.

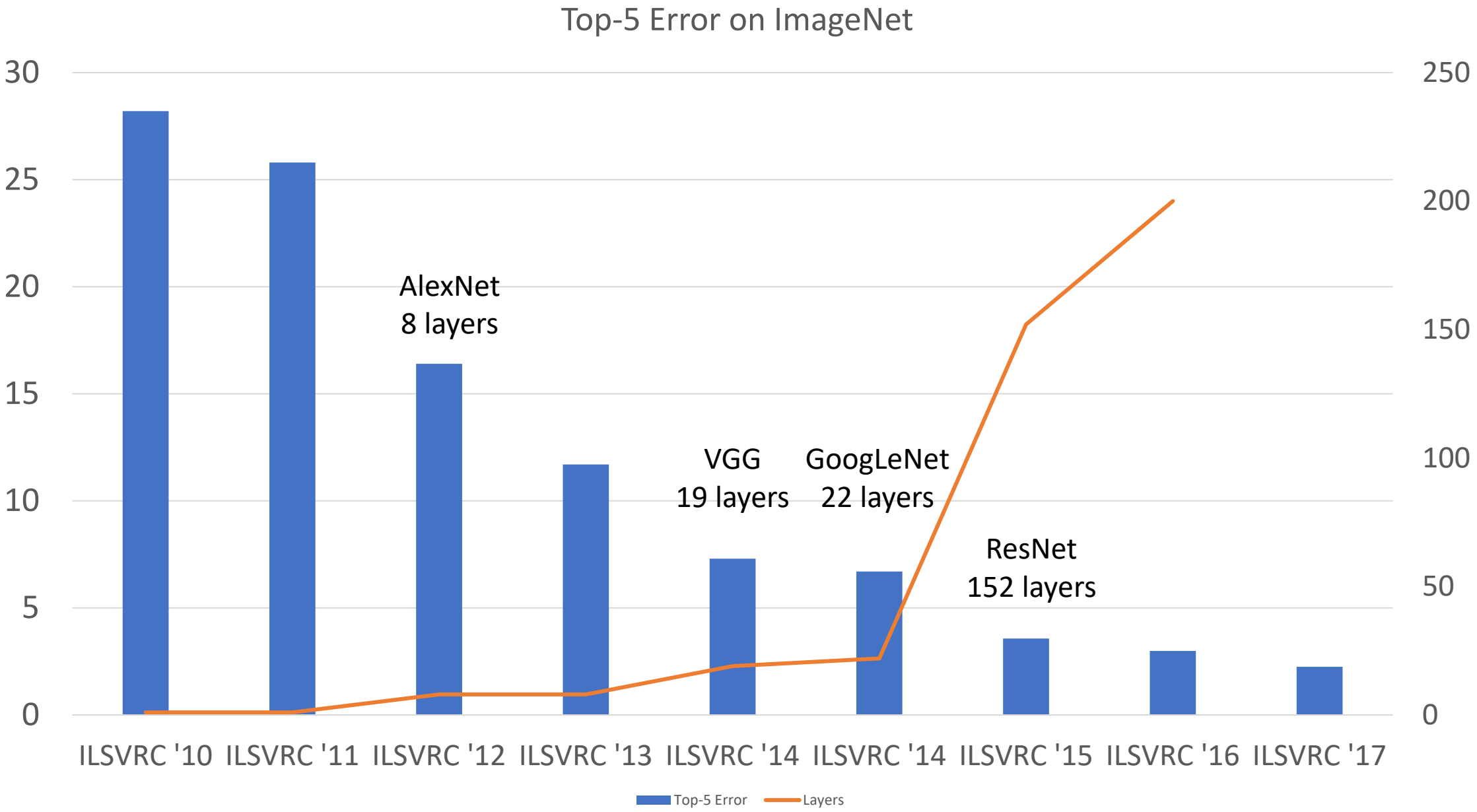


Multi-Layer Network Demo



http://playground.tensorflow.org/

Slide: Lazebnik


Top-5 Error on ImageNet
30
25
20
15
10
5
0
250
200
150
100
50
0
AlexNet
8 layers
VGG
19 layers
GoogLeNet
22 layers
ResNet
152 layers
ILSVRC '10
ILSVRC '11
ILSVRC '12
ILSVRC '13
ILSVRC '14
ILSVRC '14
ILSVRC '15
ILSVRC '16
ILSVRC '17
Top-5 Error
Layers

Convolution Example: Box Filter

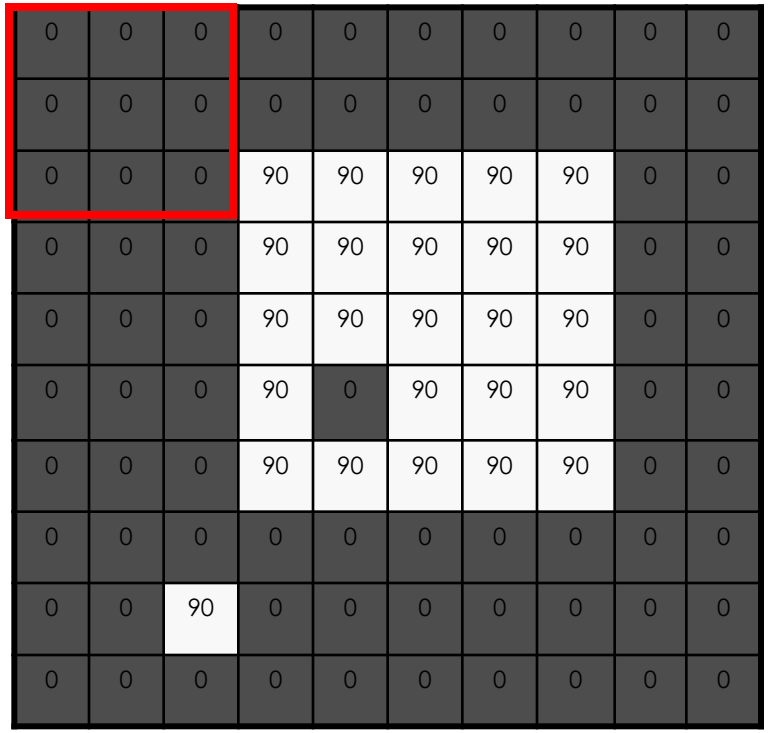
$g[\cdot\,,\cdot\,]$

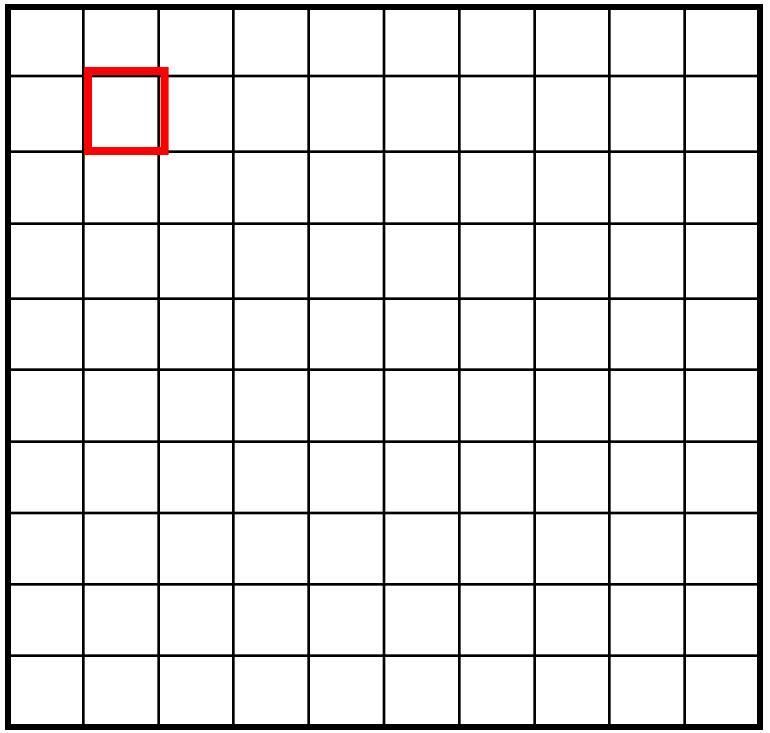
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$h[.,.]$





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

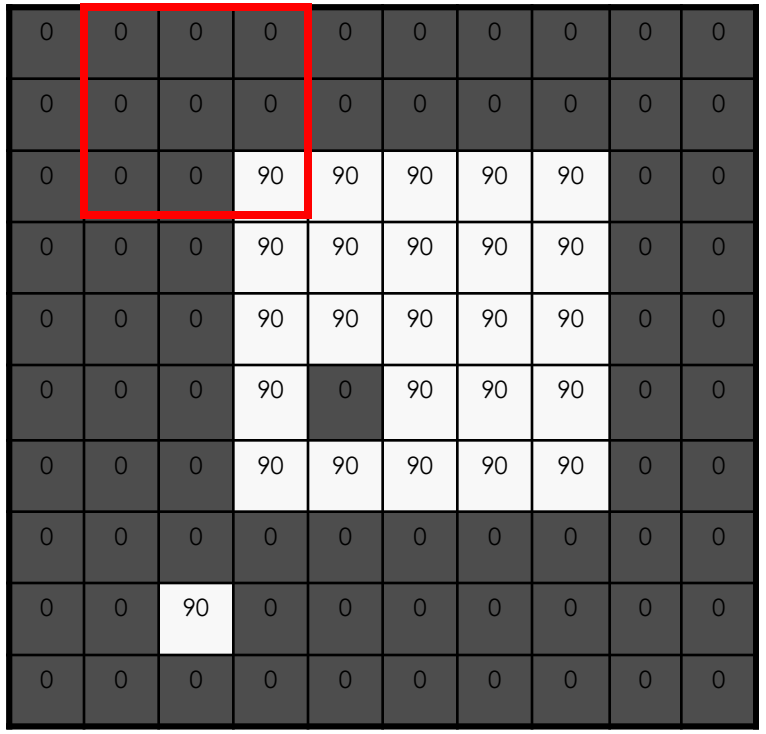
Credit: S. Seitz

Image Filtering

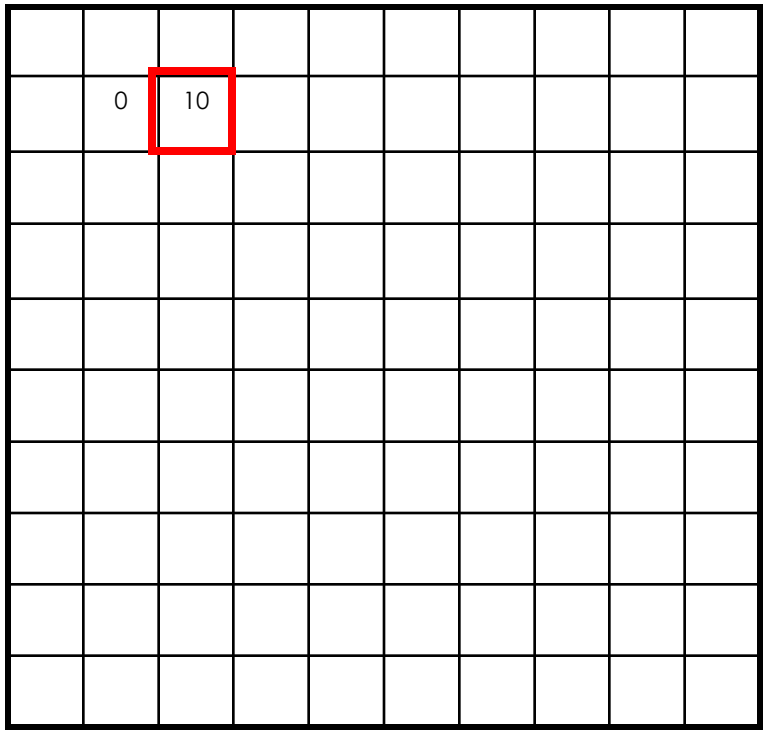
$g[\cdot\,,\cdot\,]$

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

$f[.,.]$



$h[.,.]$



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

Credit: S. Seitz

Image Filtering

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

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20						

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

Credit: S. Seitz

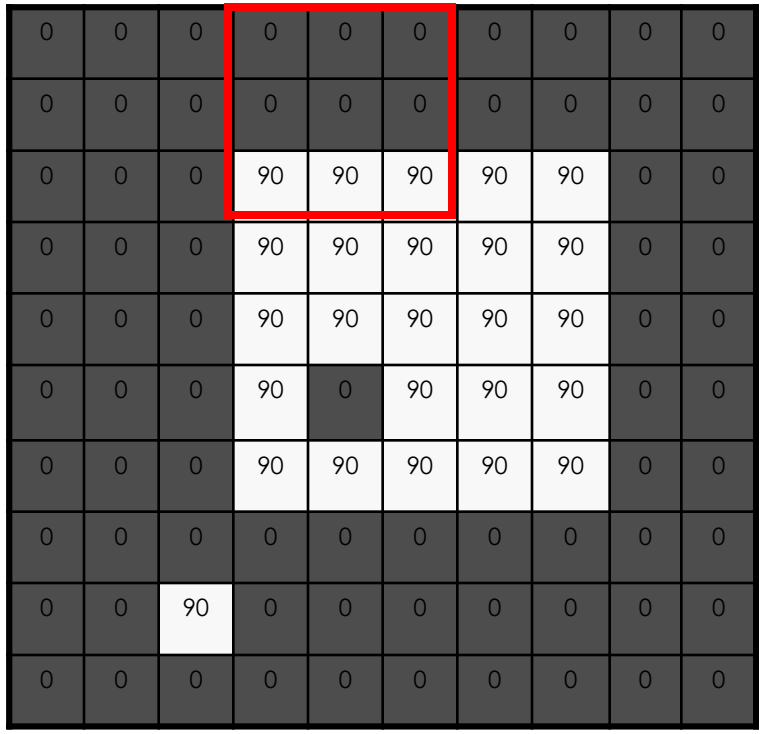
Image Filtering

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

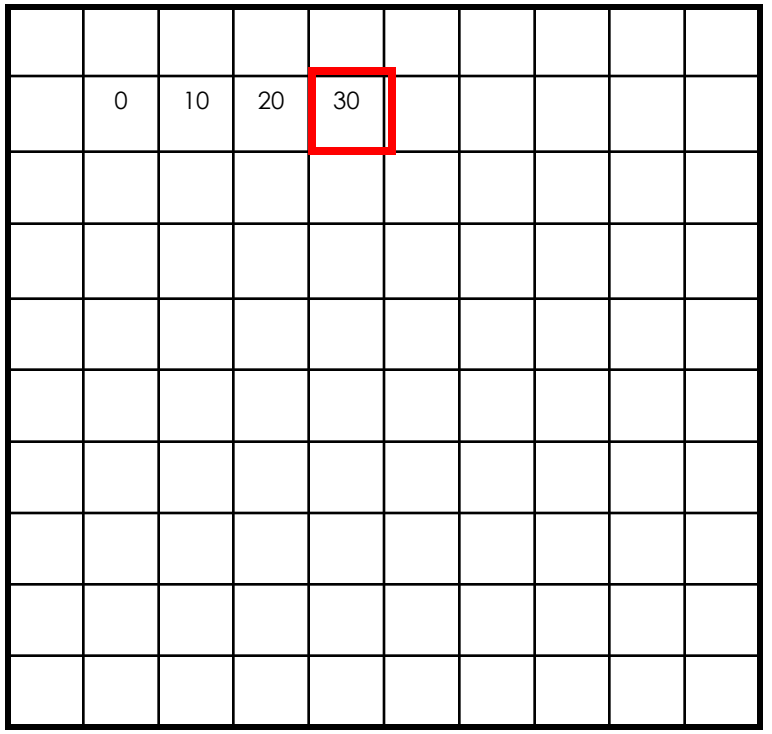
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$



$h[.,.]$



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

Credit: S. Seitz

Image Filtering

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

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

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

Credit: S. Seitz

Image Filtering

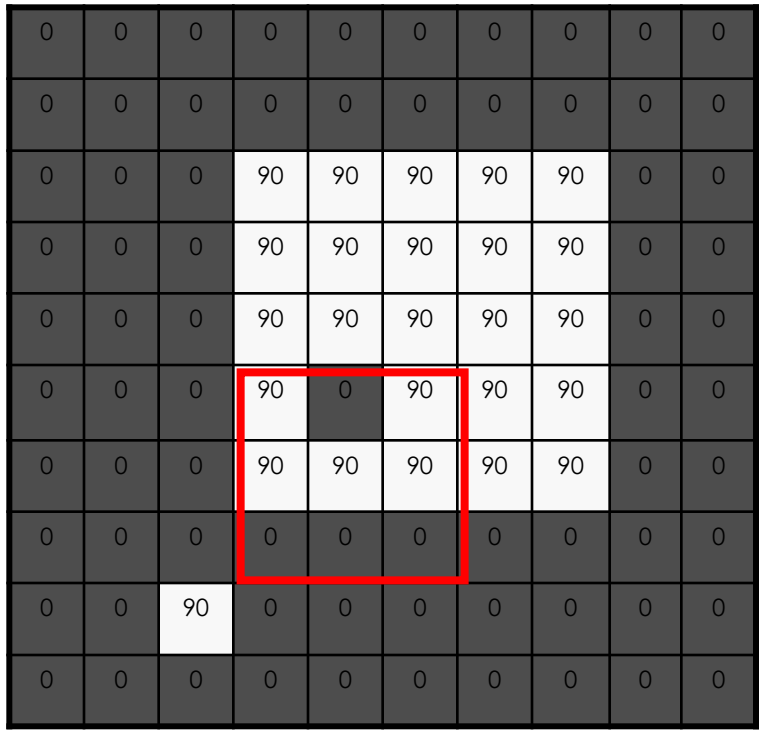
$g[\cdot\,,\cdot\,]$

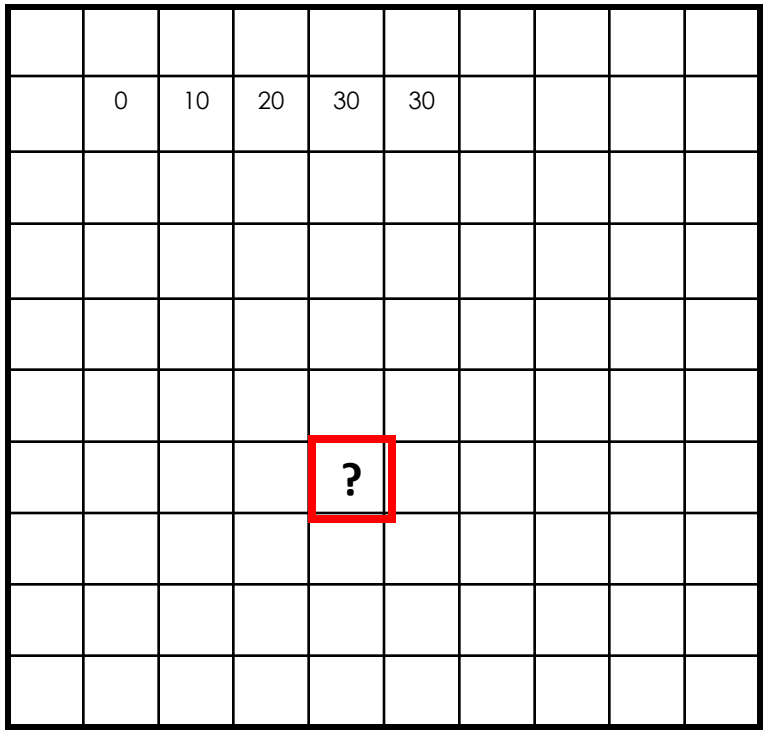
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$h[.,.]$





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

Credit: S. Seitz

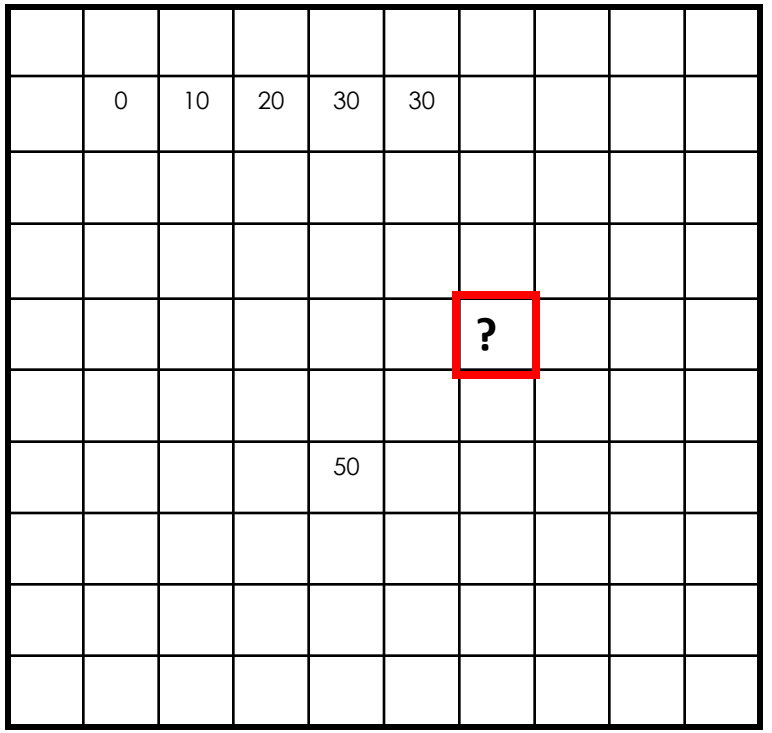
Image Filtering

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

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

$f[.,.]$

$h[.,.]$



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

Credit: S. Seitz

Image Filtering

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

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

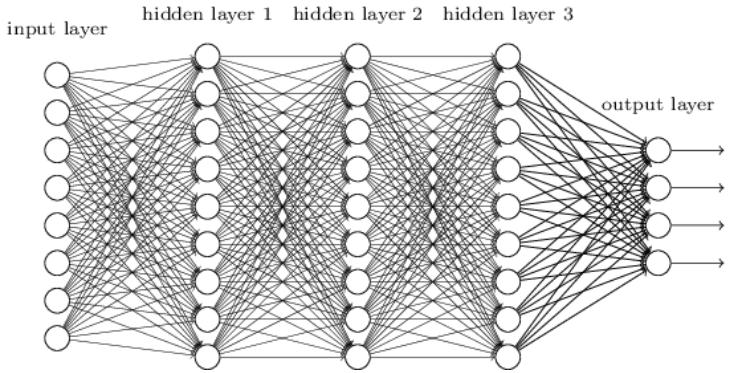
$h[.,.]$

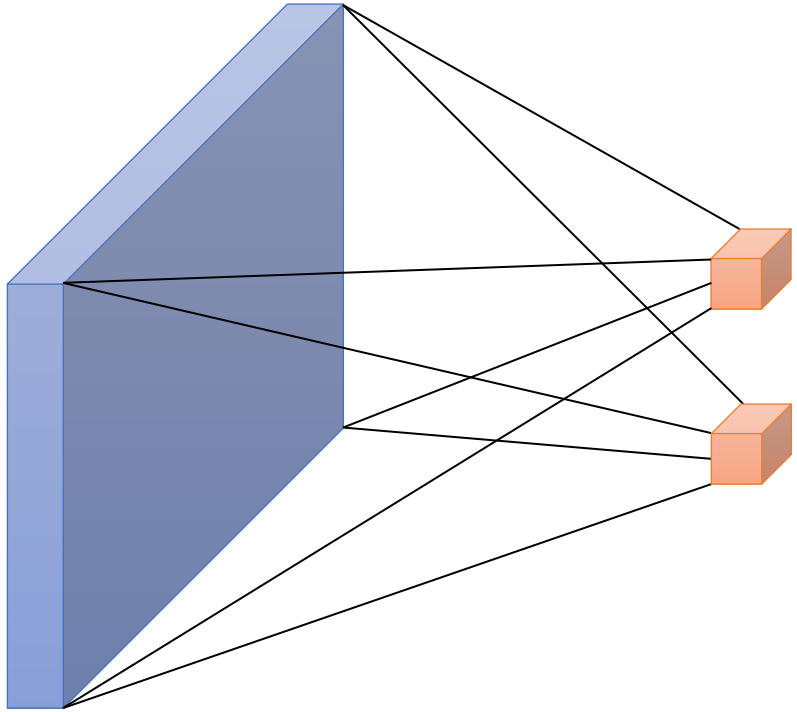
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

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

Credit: S. Seitz

From Fully-Connected to Convolutional Networks



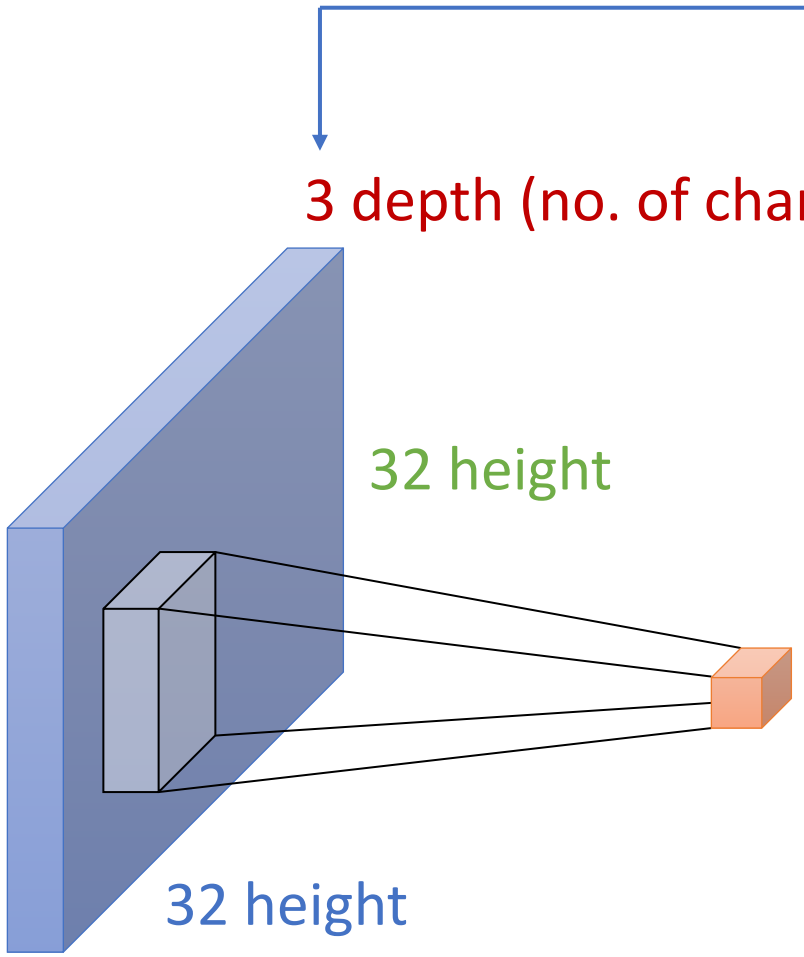


image

Fully connected layer

Slide: Lazebnik

From Fully-Connected to Convolutional Networks



Extends the full depth of the image

3 × 5 × 5 kernel (filter)

$w^{\mathrm{T}}x + b$

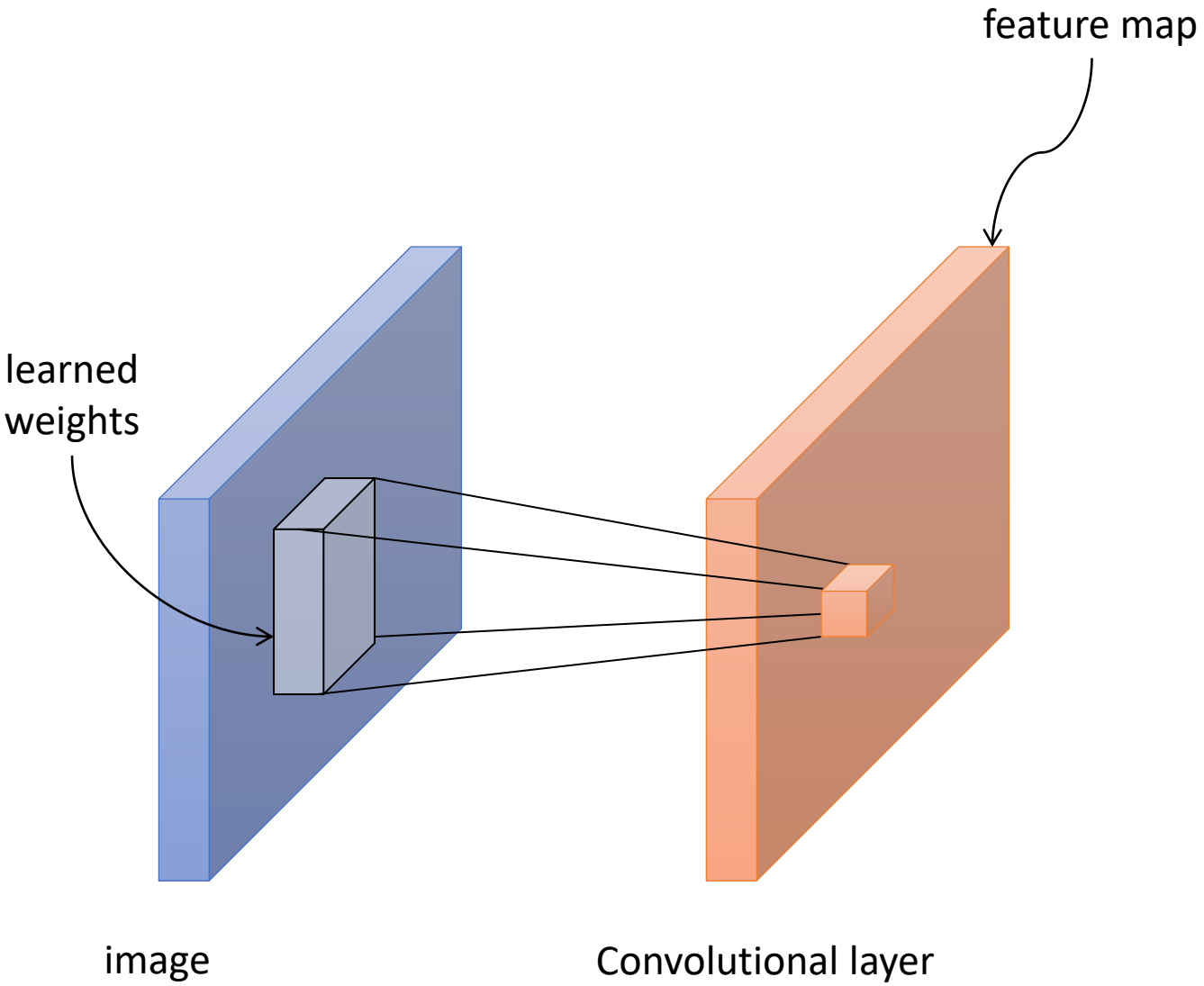
$3 \times 5 \times 5 = 75$ dimensional dot product and addition of bias

Image

Convolutional layer

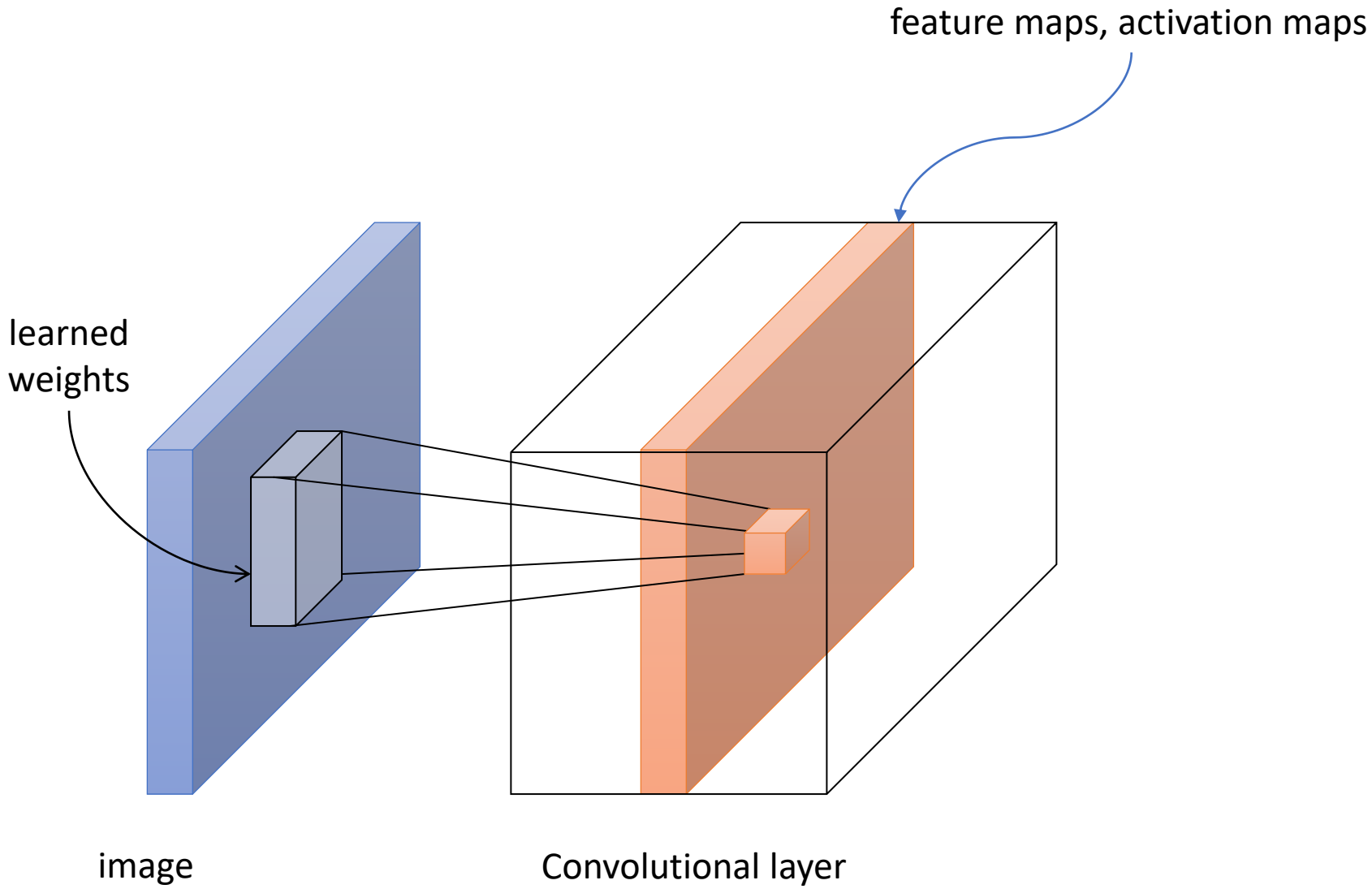
Source: Lazebnik and Johnson

From Fully-Connected to Convolutional Networks



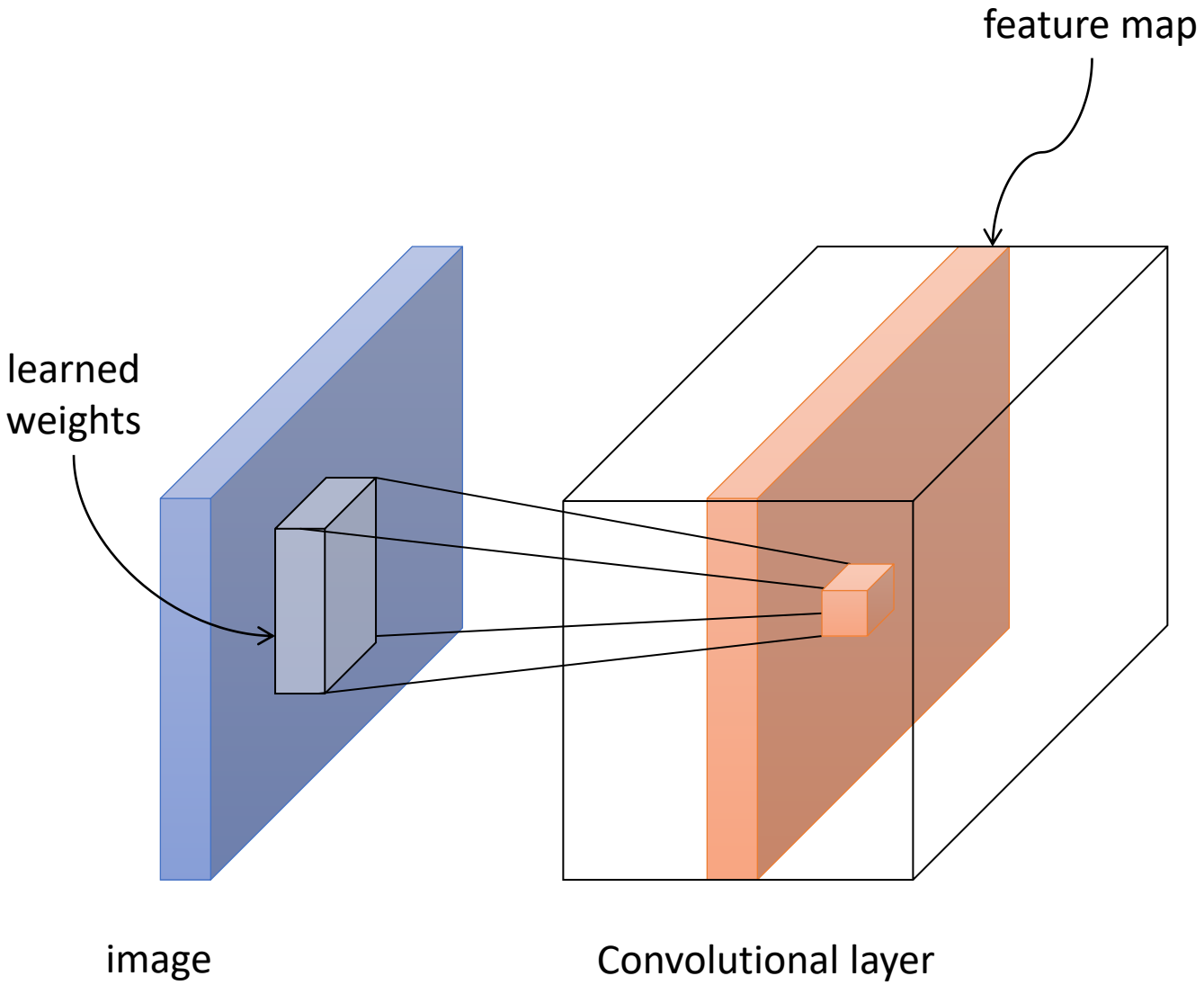
Slide: Lazebnik

From Fully-Connected to Convolutional Networks



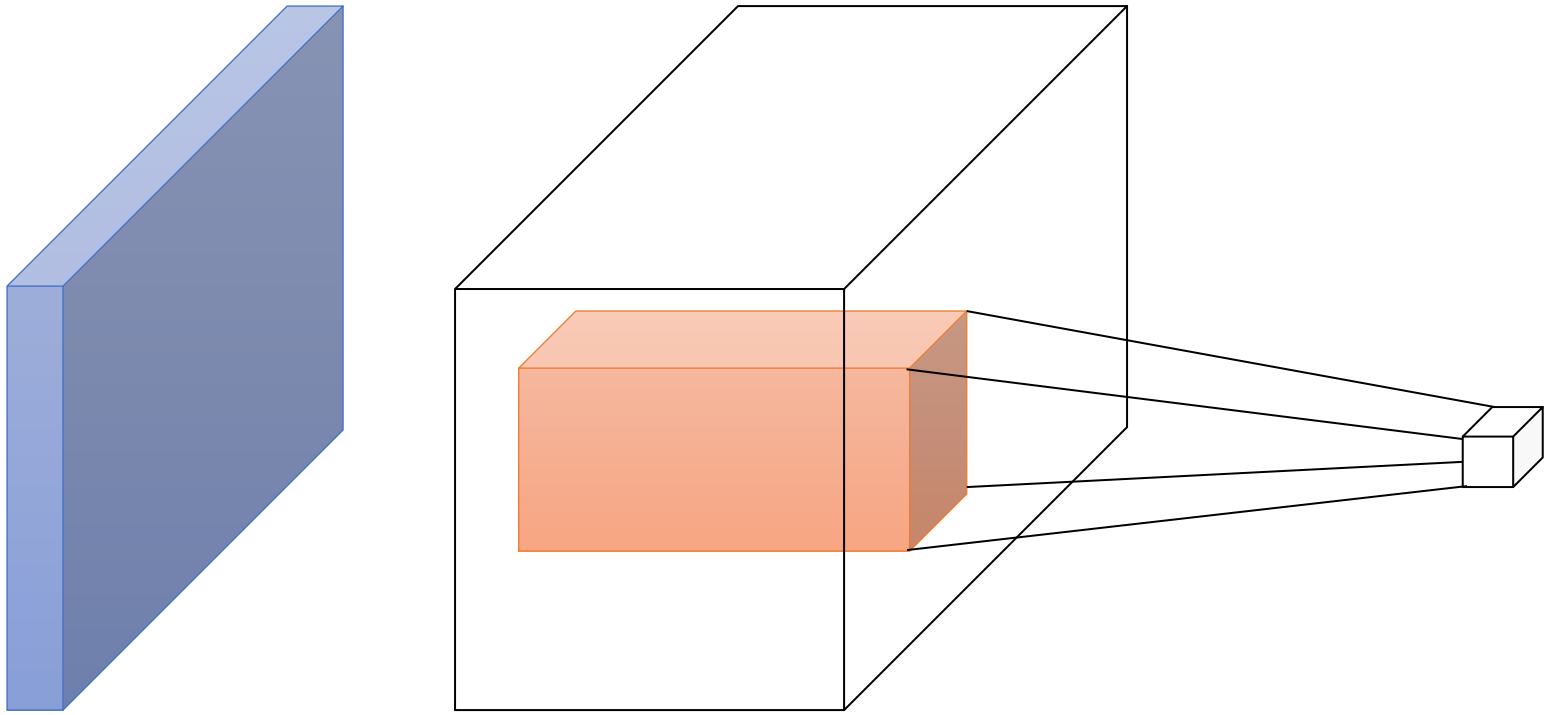
Slide: Lazebnik

From Fully-Connected to Convolutional Networks



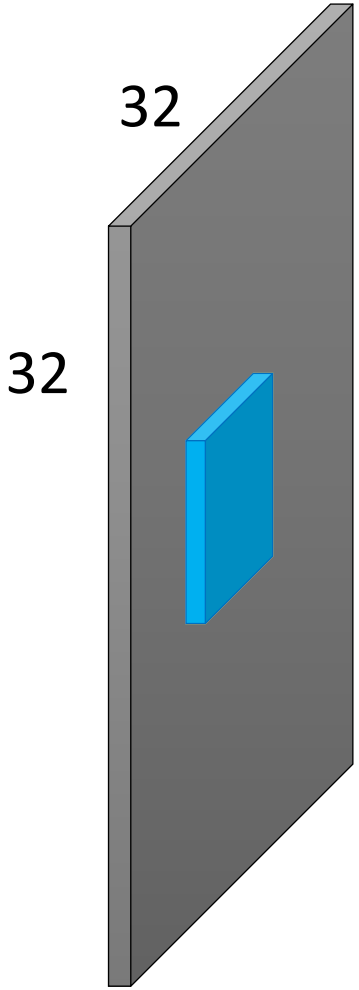
Slide: Lazebnik

From Fully-Connected to Convolutional Networks



Slide: Lazebnik

Convolutional Layer

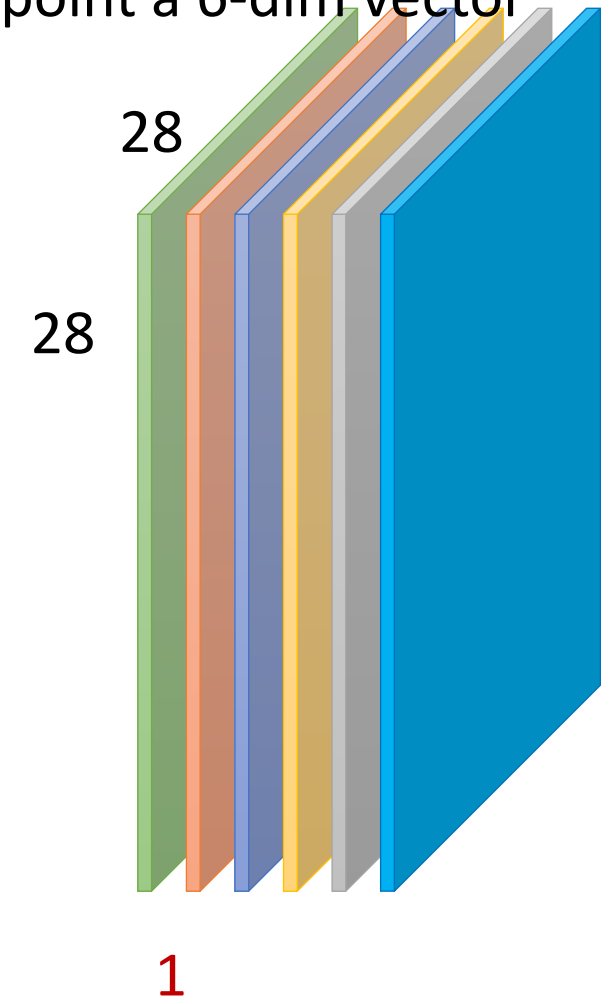


6-dim bias vector

Convoluti
onal layer

$6 \times 3 \times 5 \times 5$
filters

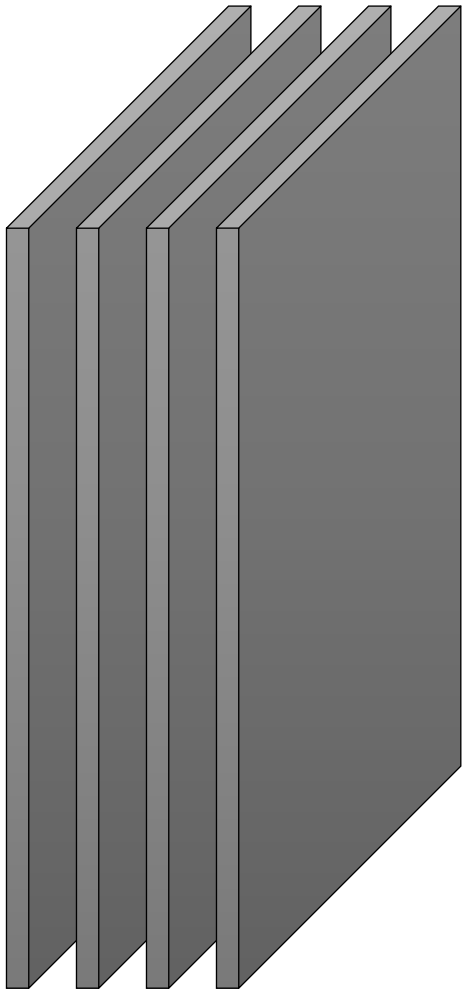
28×28 grid, at each
point a 6-dim vector



Stack activations to get a
$6 \times 28 \times 28$ output “image”

Slide: Johnson

Convolutional Layer

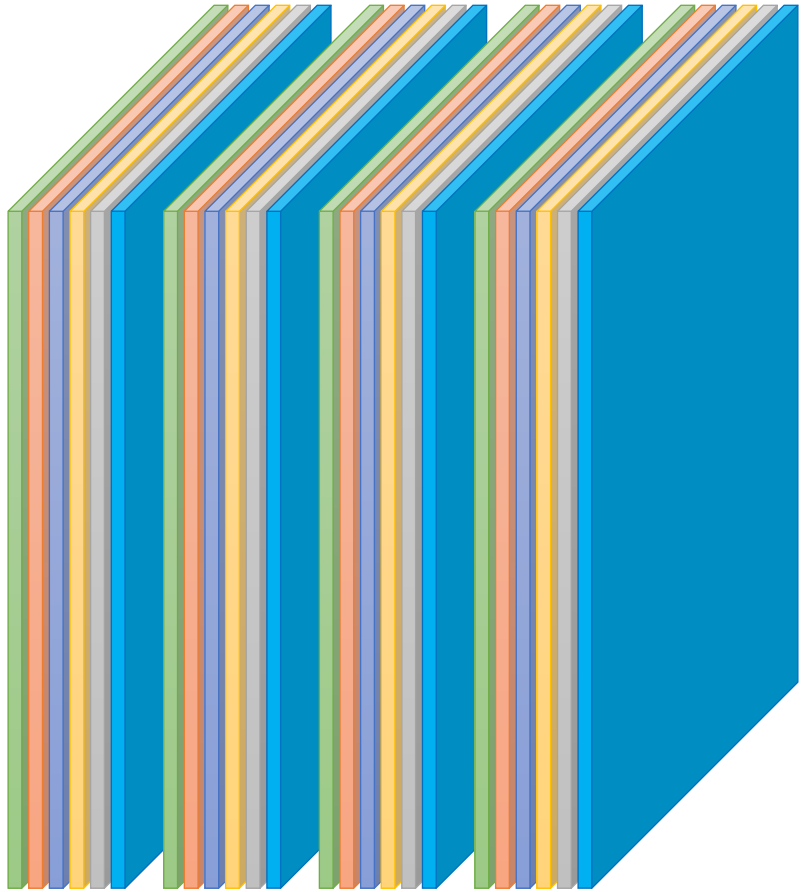


A batch of 4 images
$4 \times 3 \times 32 \times 32$

Convoluti
onal layer

$6 \times 3 \times 5 \times 5$
filters

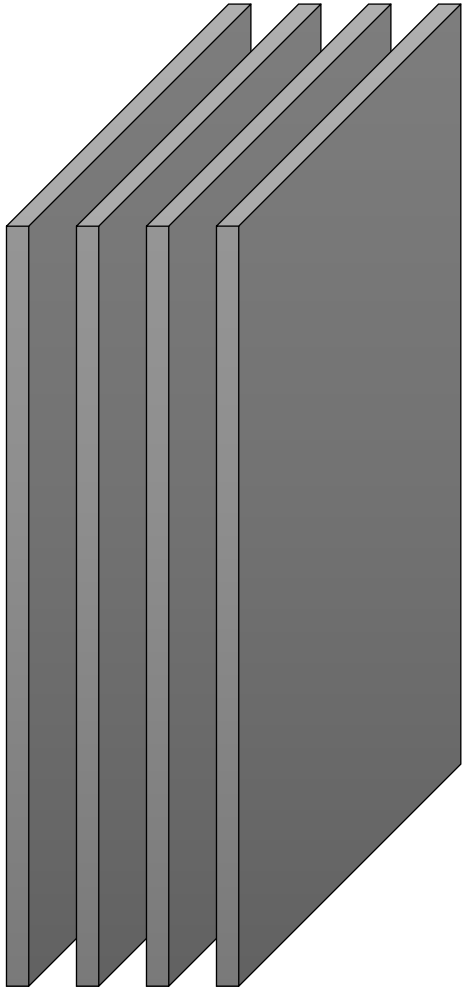
A batch of outputs
$4 \times 6 \times 28 \times 28$



Slide: Johnson

A batch N of images

$N \times C_{\text{in}} \times H \times W$



C_{in}

Bias vector

C_{out}-dim

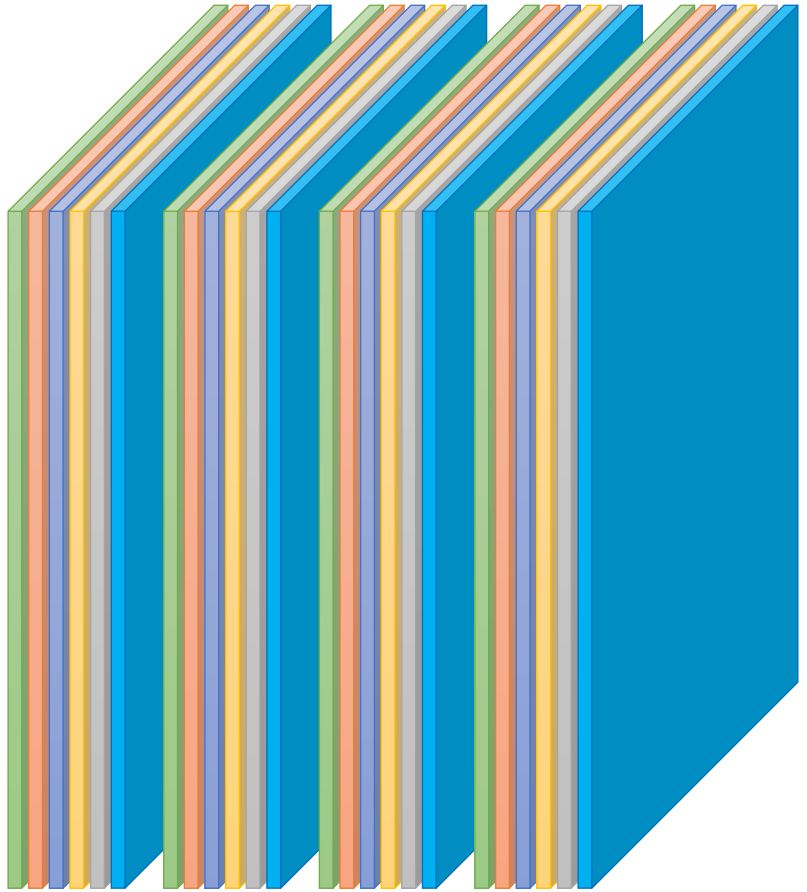
Convoluti
onal layer

Filters

$C_{\text{out}} \times C_{\text{in}} \times K_h \times K_w$

A batch of outputs

$N \times C_{\text{out}} \times H' \times W'$



C_{out}

Slide: Johnson

Learnt Filters

Linear classifier

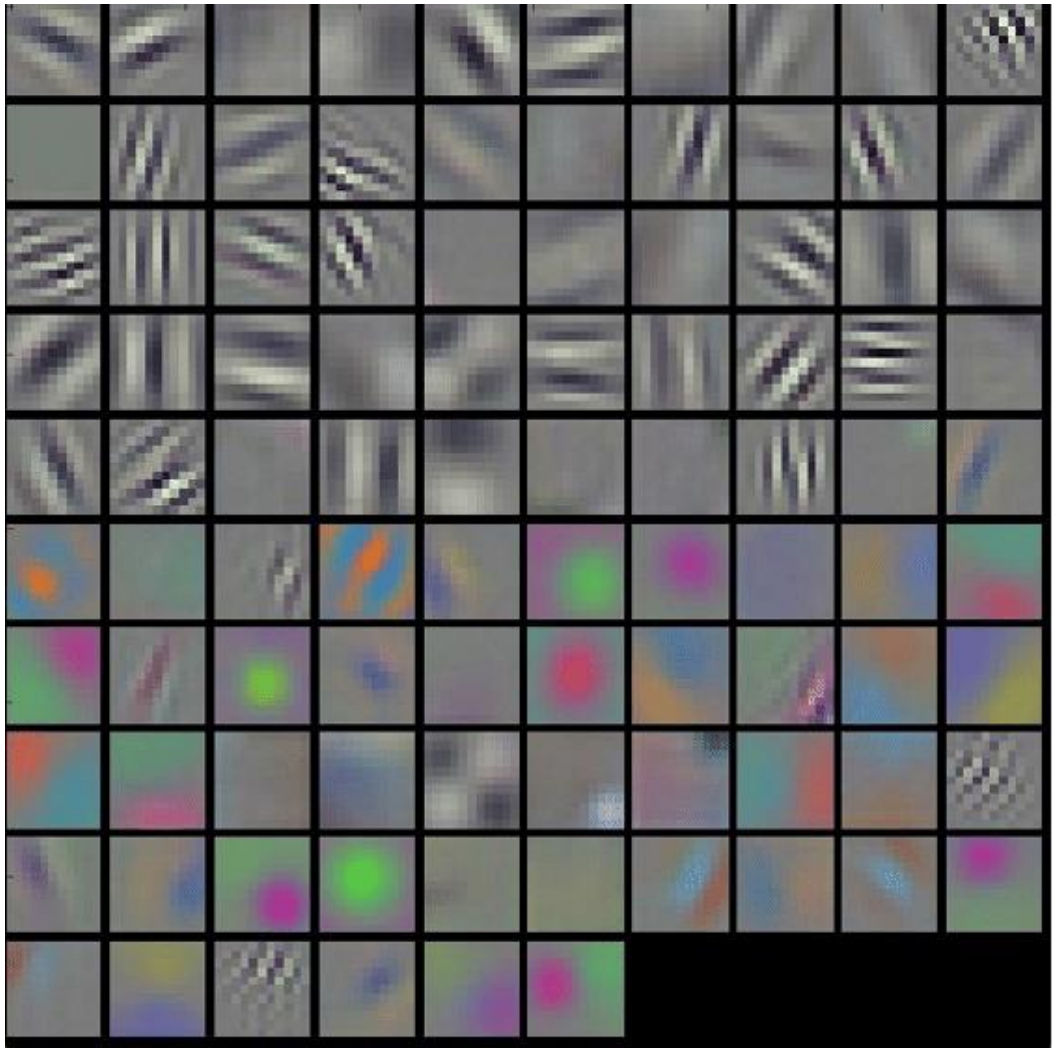
https://cs231n.github.io/linear-classify/

Fully – connected

From Justin Johnson slides

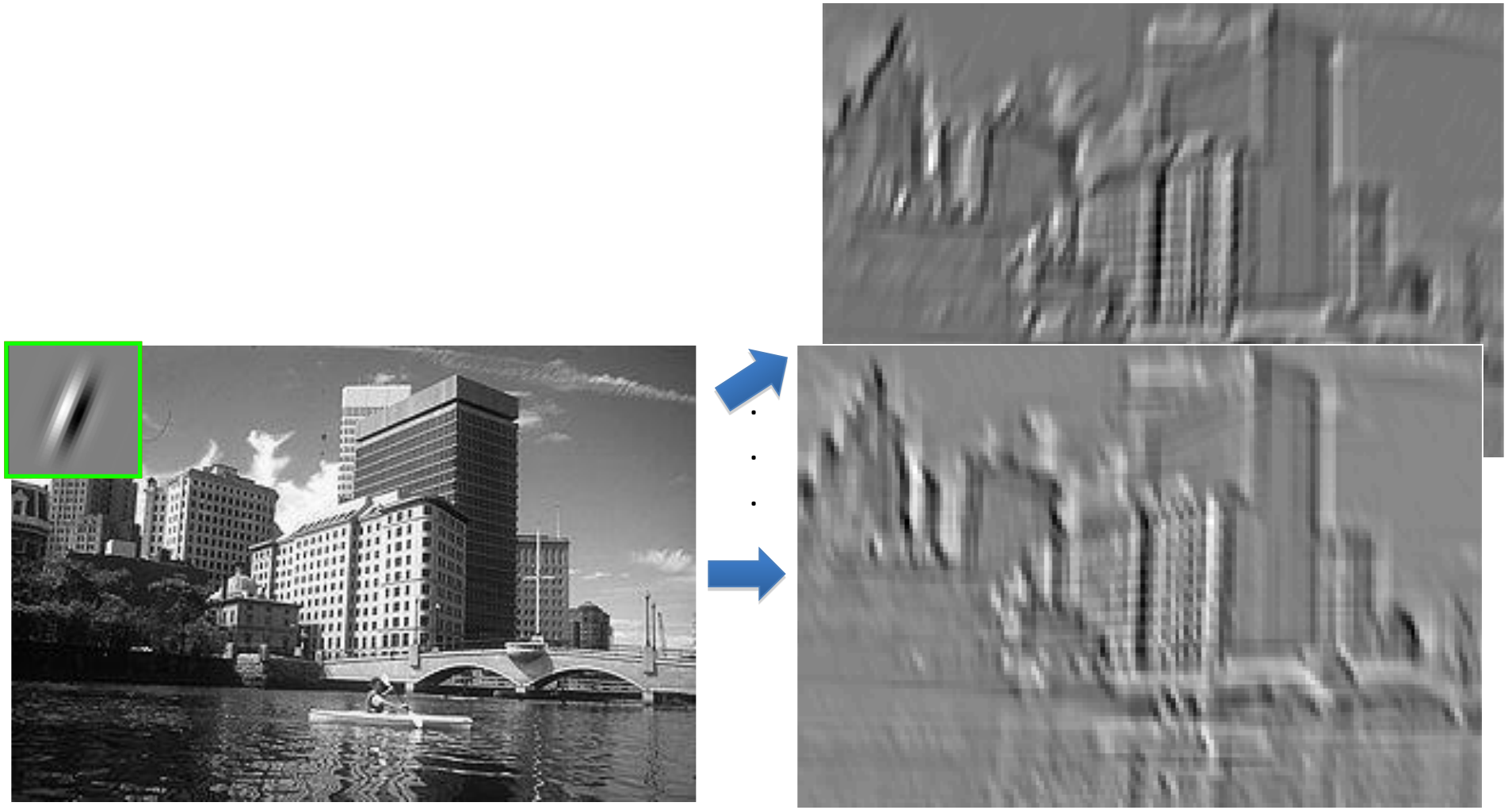
AlexNet layer 1

11 × 11 weights



https://www.researchgate.net/figure/First-layer-filters-of-AlexNet-trained-on-ImageNet-A-considerable-amount-of-redundancy_fig4_338162455

Convolution as Feature Extraction

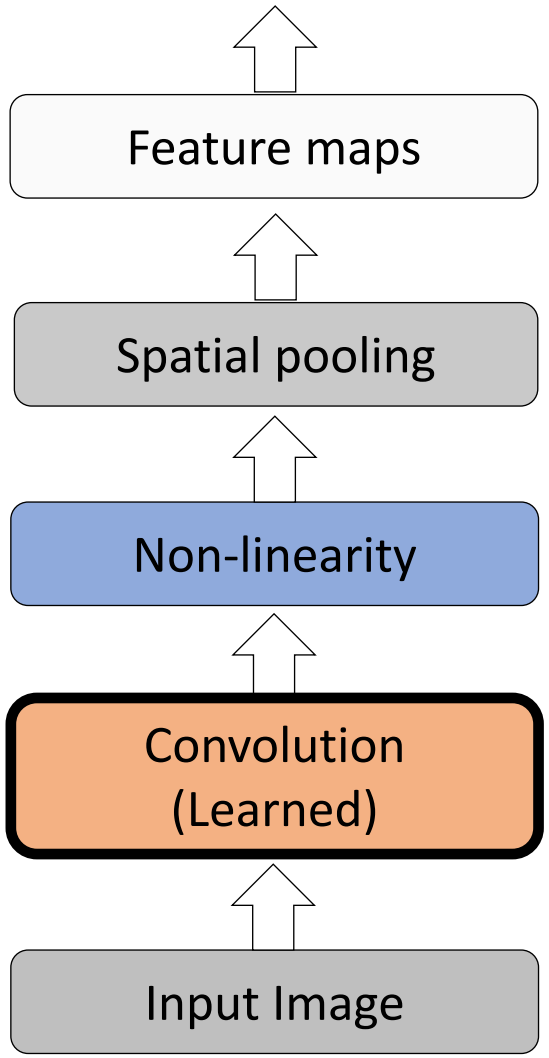


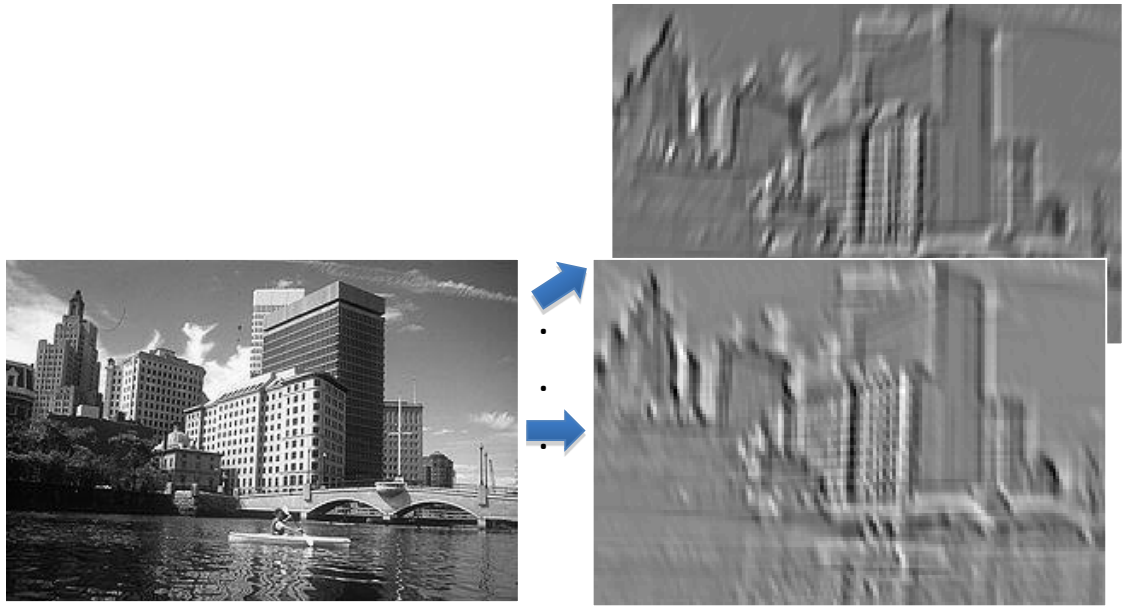
Input

Feature Map

Slide: Lazebnik

Key Operations in a CNN





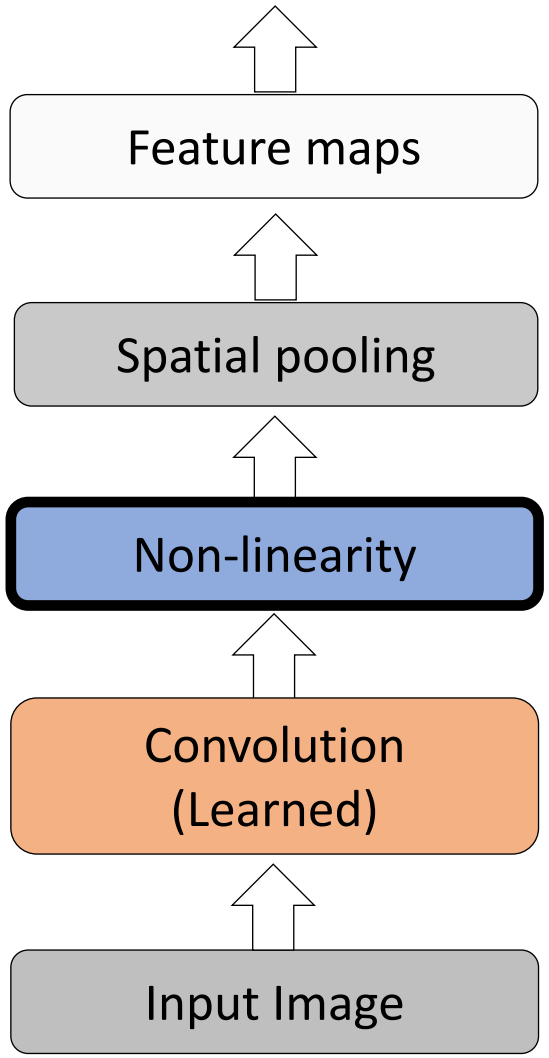
Input

Feature Map

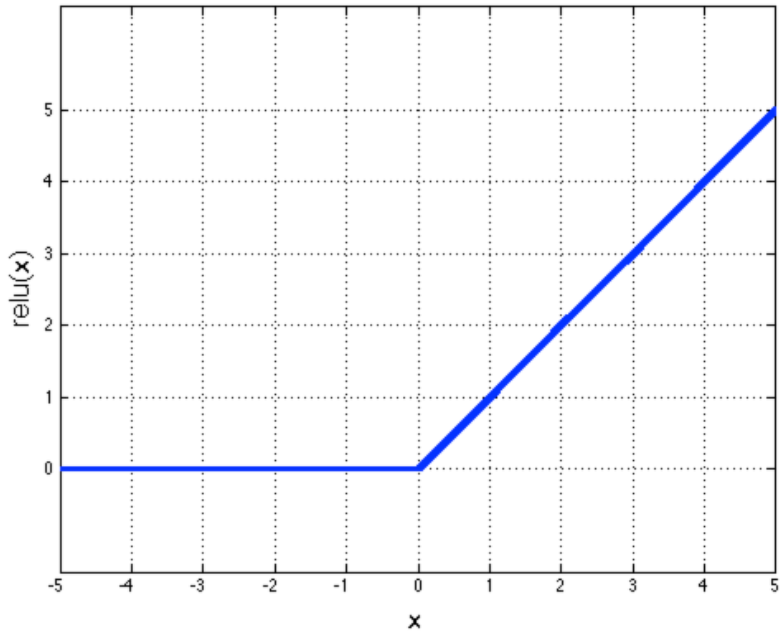
Source: R. Fergus, Y. LeCun

Slide: Lazebnik

Key Operations



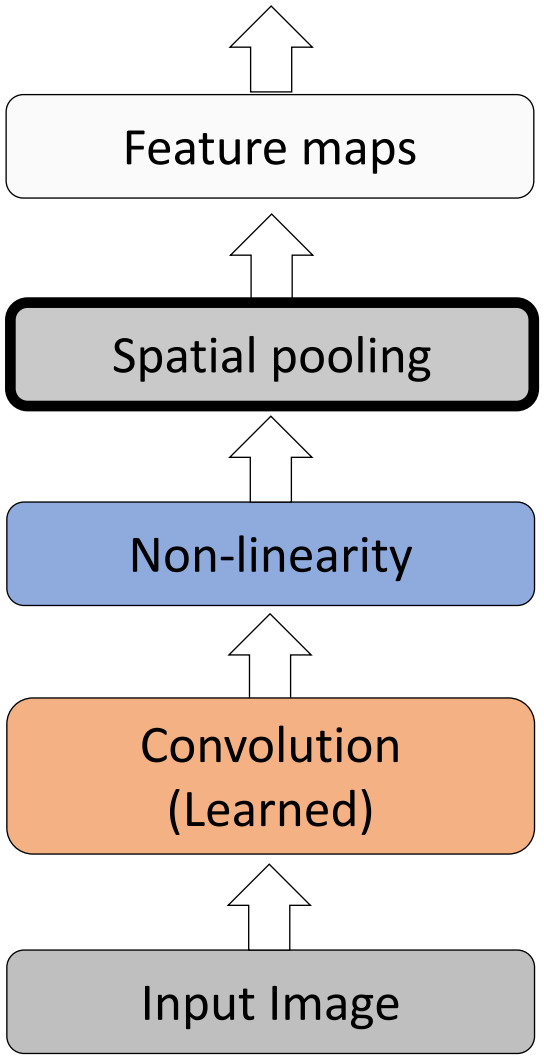
Rectified Linear Unit (ReLU)

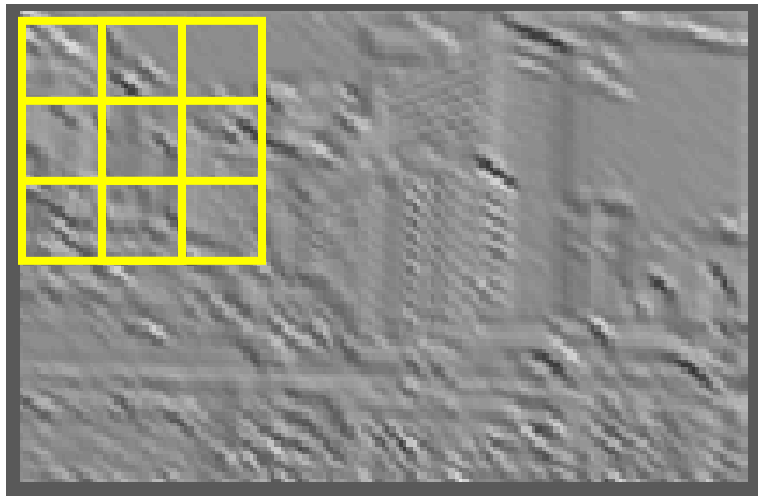


Source: R. Fergus, Y. LeCun

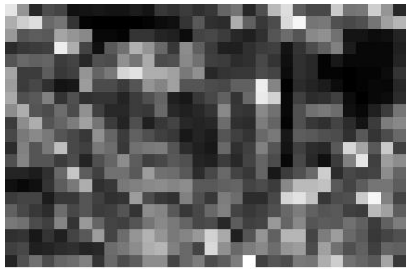
Slide: Lazebnik

Key Operations





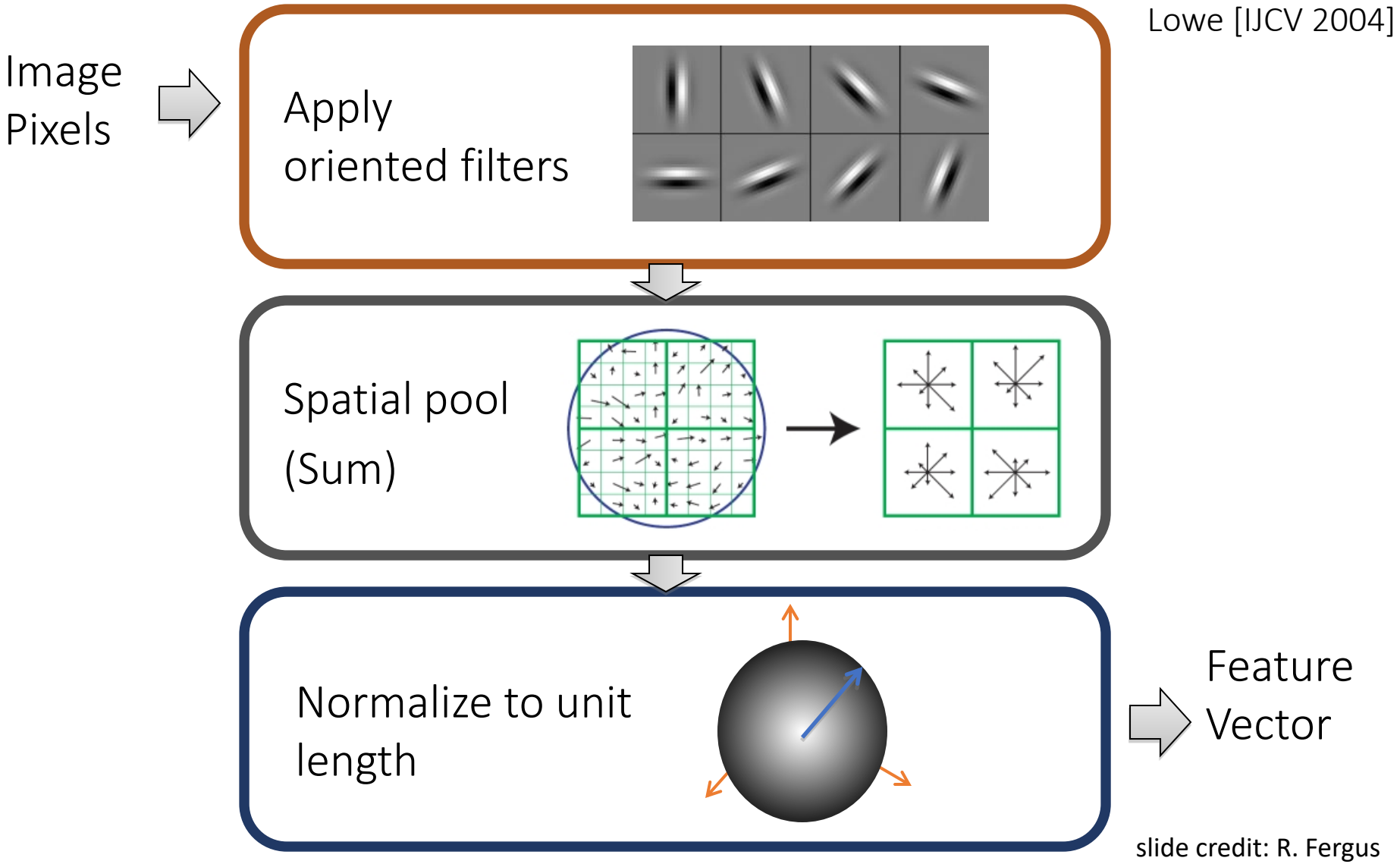
Max



Source: R. Fergus, Y. LeCun

Slide: Lazebnik

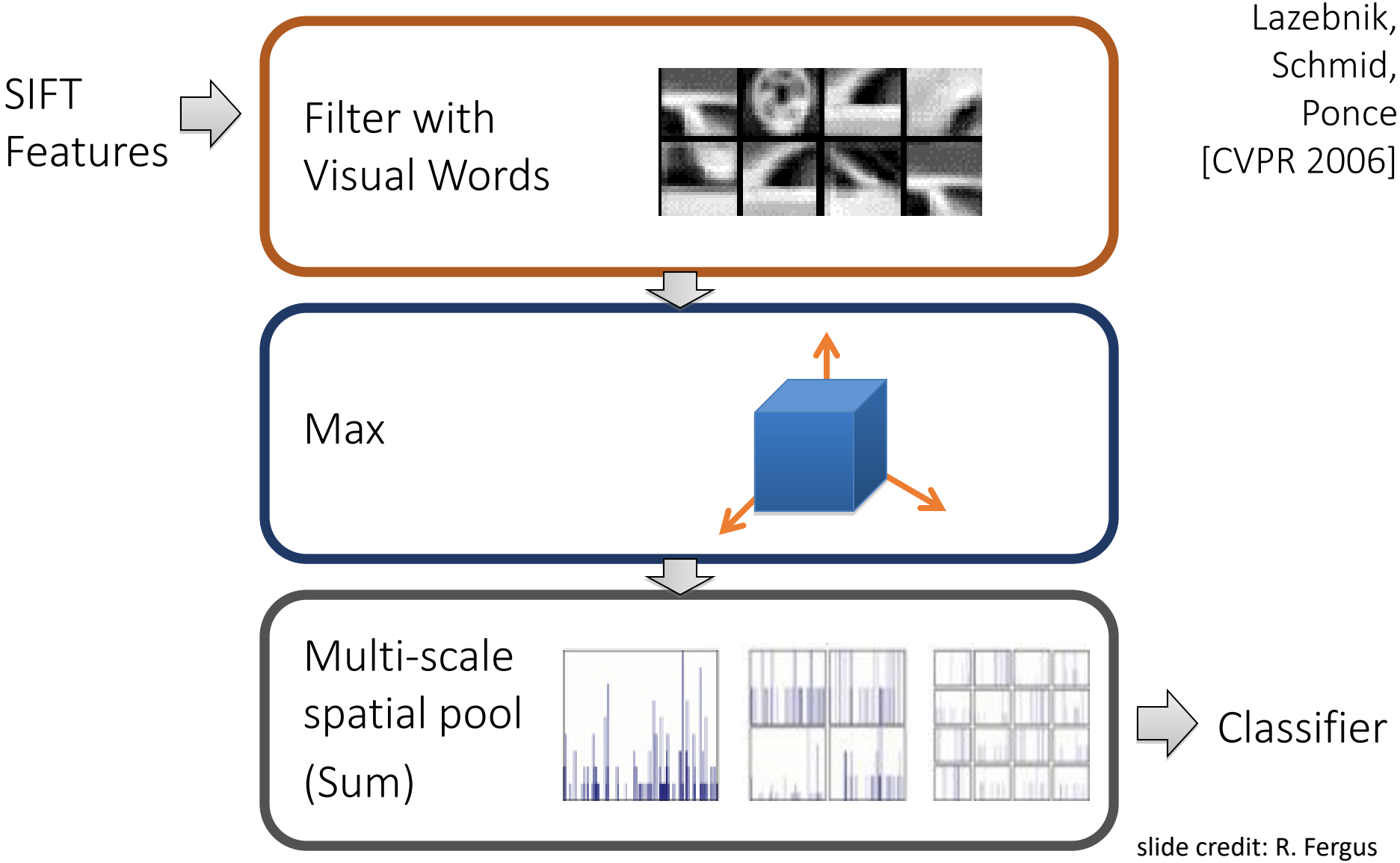
Comparison to Pyramids with SIFT



Lowe [IJCV 2004]

slide credit: R. Fergus

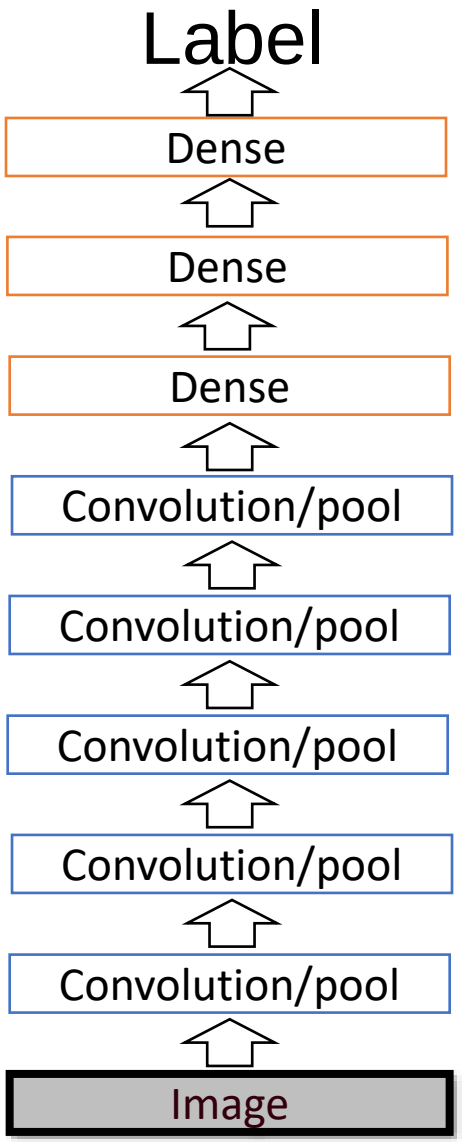
Comparison to Pyramids with SIFT



Lazebnik, Schmid, Ponce [CVPR 2006]

slide credit: R. Fergus

Key Idea: Learn Features and Classifier That Work Well Together ("End-to-End Training")



Shrinking Due to Convolutions and Padding

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

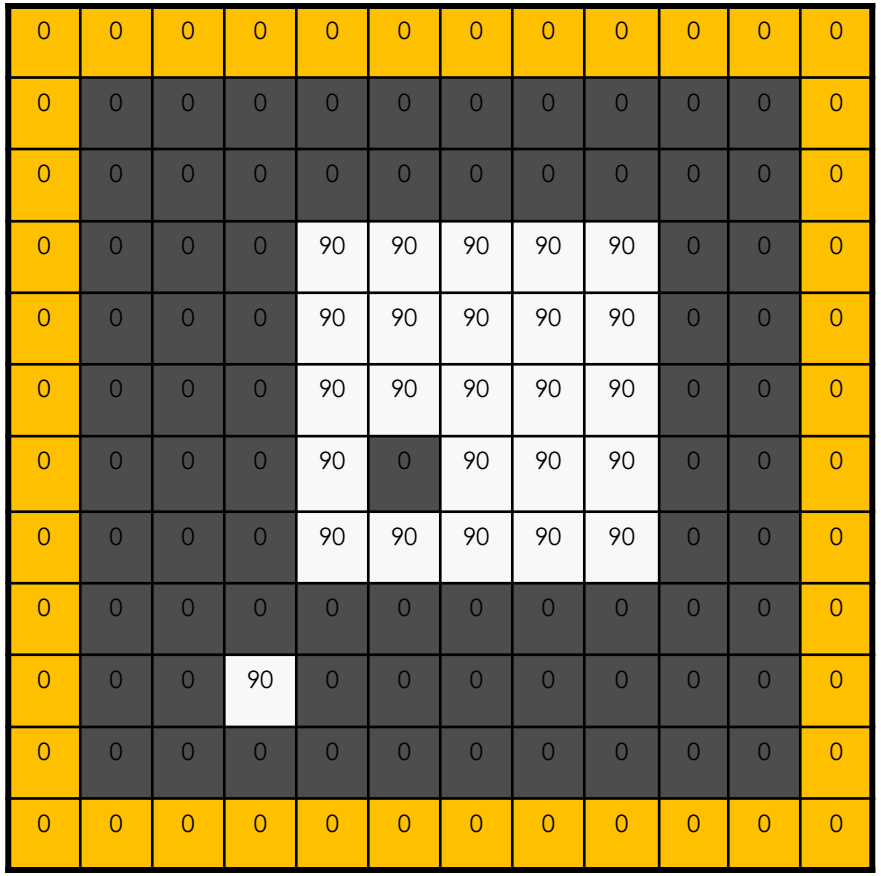
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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

Input: 10x10
Filter: 3x3
Output: 8x8
In general:
Input: W $(W \times W)$
Filter: K $(K \times K)$
Output: W – K + 1

Slide: Modified from Sietz

Shrinking Due to Convolutions and Padding

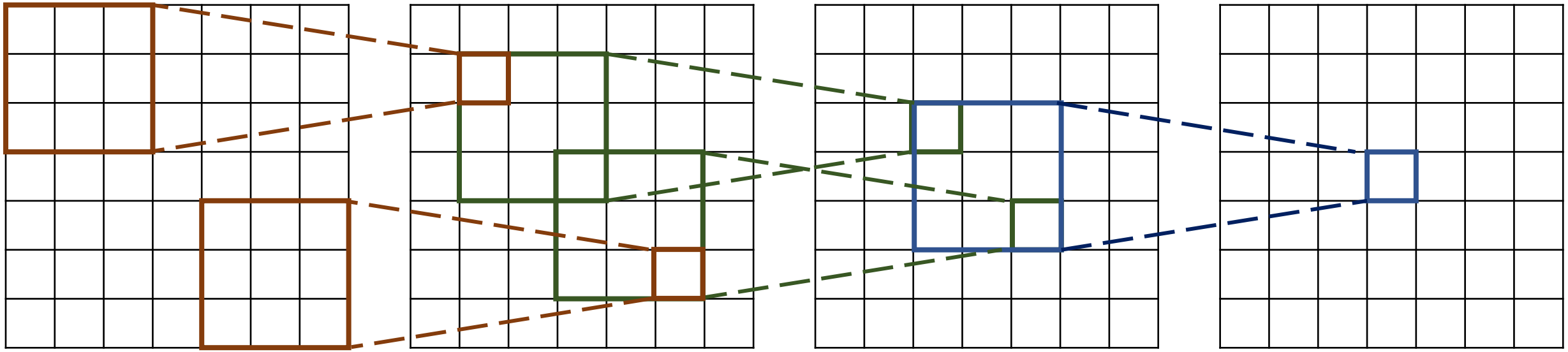


Input: 10x10
Filter: 3x3
Output: 8x8
In general:
Input: W $(W \times W)$
Filter: K $(K \times K)$
Output: W – K + 1
Padding: P
Output: $W - K + 1 + 2P$
Set $P = (K - 1) / 2$ to
make output have same size as input.

Slide: Modified from Sietz

Receptive Field

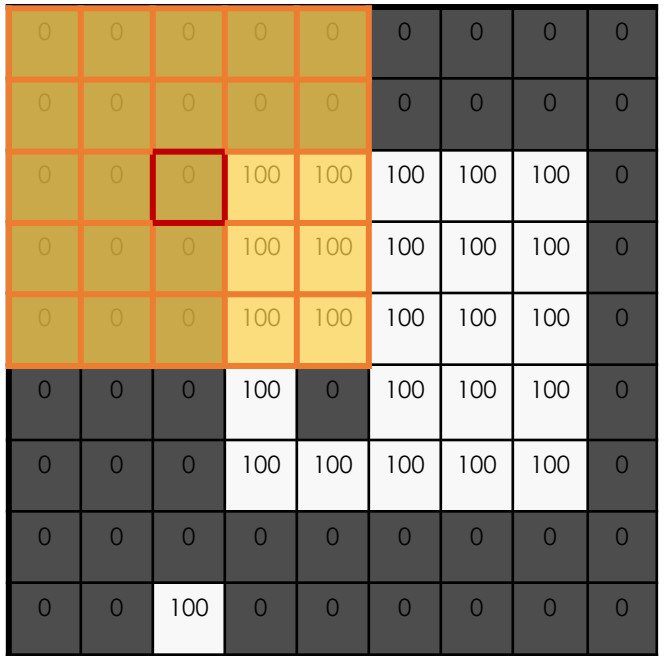
$K = 3$



- Each successive convolution adds $K - 1$ to the receptive field size With L layers the receptive field size is $1 + L \times (K - 1)$
- Receptive field grows linearly with layers.
- Down-sample.

Slide: Johnson

Stride

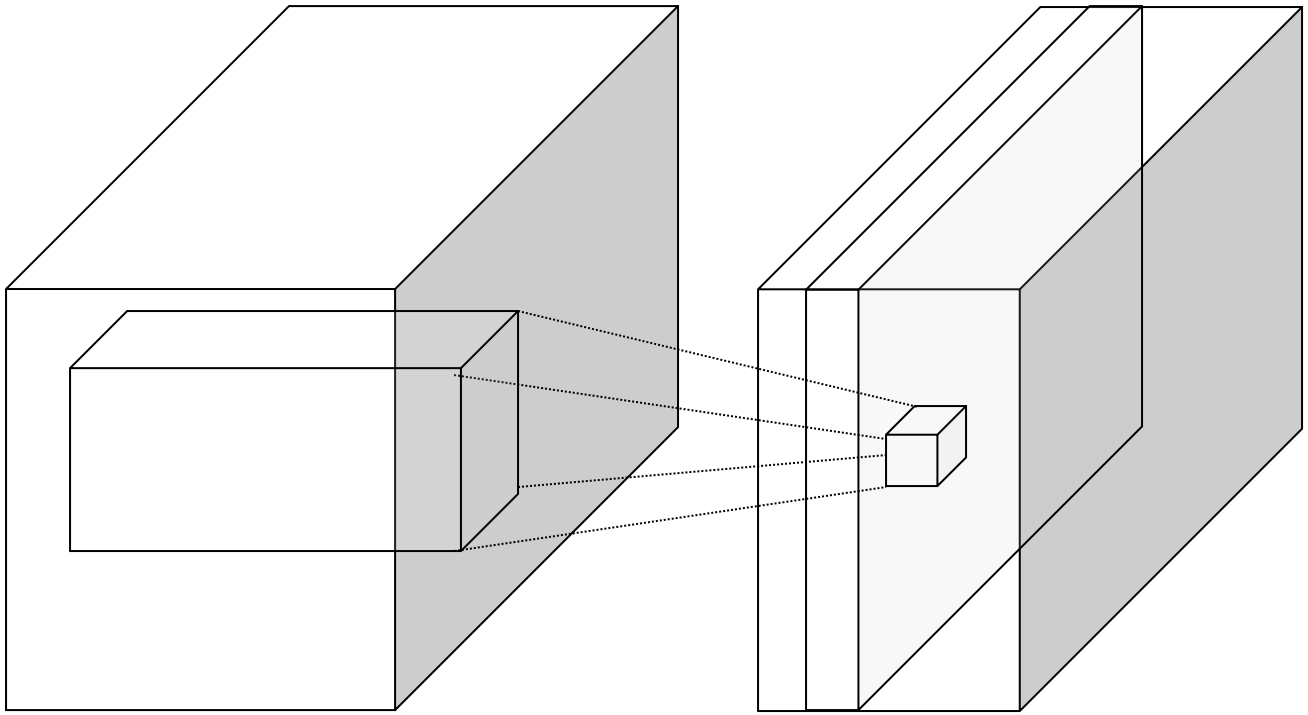


24	64	64
36	60	

Input: 9×9
Filter: 5×5 (all 1/25)
Stride: 2
Output: 3×3
In general:
Input: W $(W \times W)$
Filter: K $(K \times K)$
Stride: S
Output: $(W - K + 2P)/S + 1$

Slide: Modified from Sietz

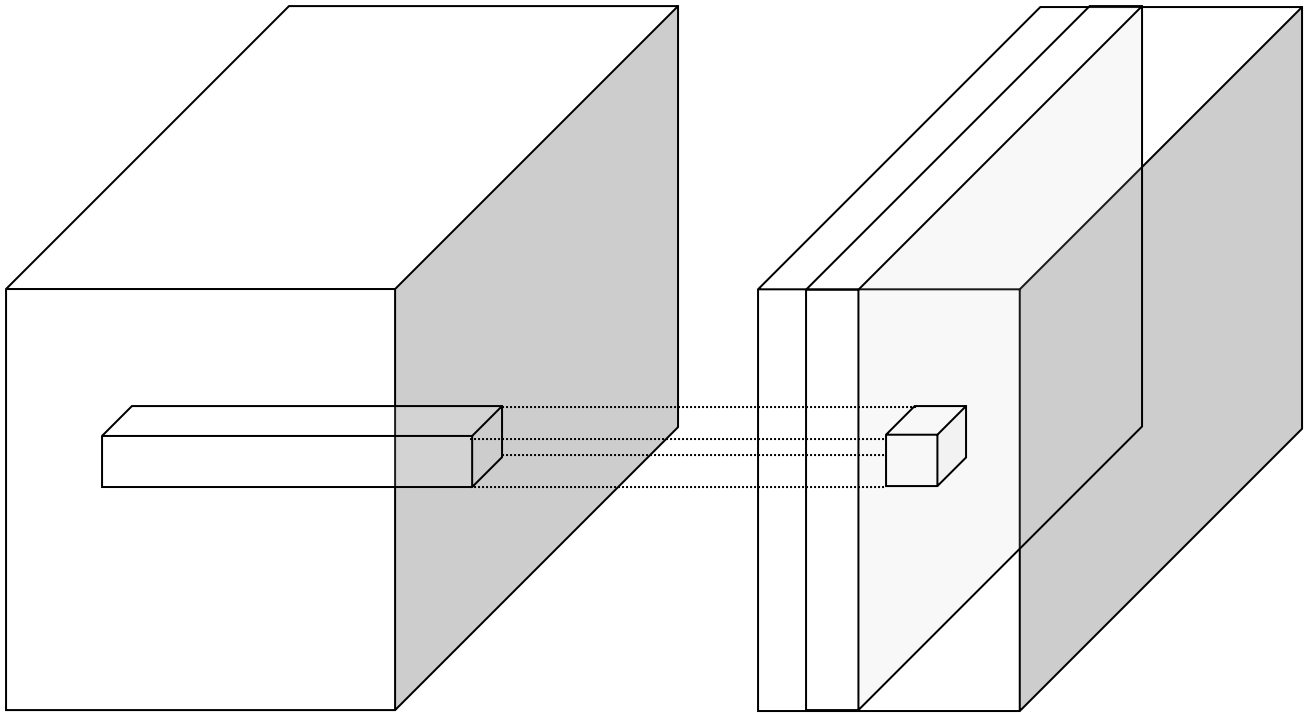
1x1 Convolutions



conv layer

Slide: Lazebnik

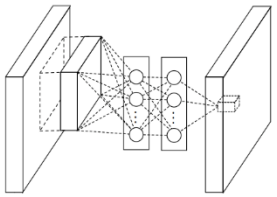
1x1 Convolutions



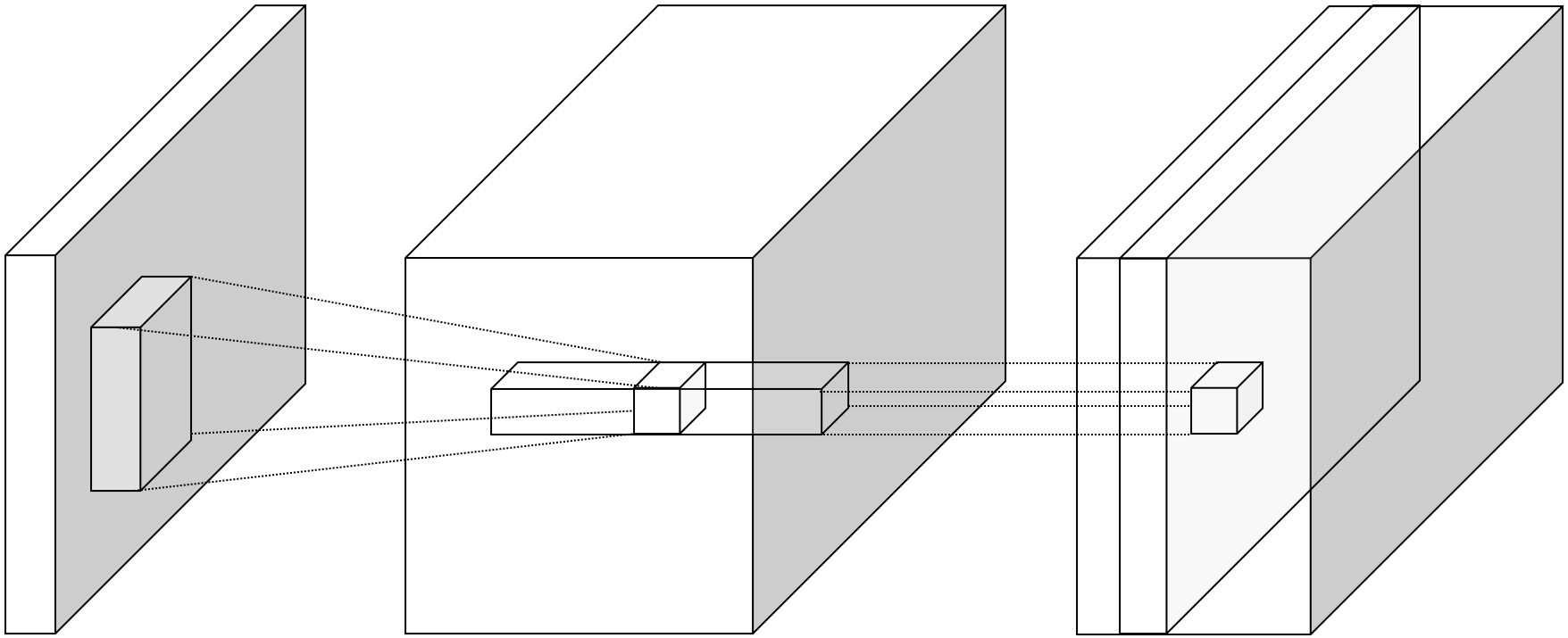
1x1 conv layer

Slide: Lazebnik

1x1 Convolutions



Stacking 1x1 conv layers gives MLP operating on each input position



1x1 conv layer

Slide: Lazebnik

Computing the No. of Learnable Parameters

If an input image is of size 28×28 grayscale, and there are 6 filters of dimension 5×5 filters, and if stride is 1, padding is 2, compute
1. Output feature map size, 2. No. of learnable parameters, 3. No. of learnable parameters if the image were a color image 4. Number of multiply-add operations for color image

Output size

$$(W - K + 2P)/S + 1 = (28 - 5 + 2 \times 1)/1 + 1 = 32 \rightarrow 6 \times 32 \times 32 = 6144$$

No. of learnable parameters

Parameters per filter = $1 \times 5 \times 5 + 1$ (+1 for the bias) = 26
Total no of learnable parameters for the 6 filters = $26 \times 6 = 156$

If color

Parameters per filter = $3 \times 5 \times 5 + 1 = 76$, Total for 6 filters = $76 \times 6 = 456$

Multiply-add operations for color

Each element in output needs 75 operations. Total multiply-add = $75 \times 6144 = 460{,}800$

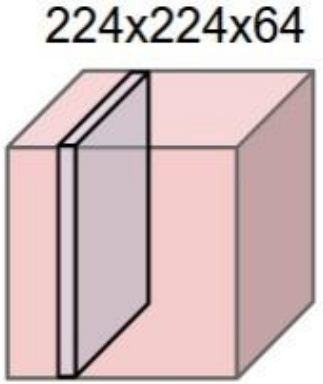
Convolution Summary

Input: $C_{\text{in}} \times H \times W$
Hyperparameters:
- **Kernel size**: $K_H \times K_W$
- **Number filters**: C_{out}
- **Padding**: P
- **Stride**: S
Weight matrix: $C_{\text{out}} \times C_{\text{in}} \times K_H \times K_W$
giving C_{out} filters of size $C_{\text{in}} \times K_H \times K_W$
Bias vector: C_{out}
Output size: $C_{\text{out}} \times H' \times W'$
where:
- $H' = (H - K + 2P)/S + 1$
- $W' = (W - K + 2P)/S + 1$

Common settings:
$K_H = K_W$ (Small square filters)
$P = (K - 1) / 2$ ("Same" padding)
$C_{\text{in}}, C_{\text{out}} = 32, 64, 128, 256$ (powers of 2)
$K = 3, P = 1, S = 1$ (3x3 conv)
$K = 5, P = 2, S = 1$ (5x5 conv)
$K = 1, P = 0, S = 1$ (1x1 conv)
$K = 3, P = 1, S = 2$ (Downsample by 2)

Slide: Johnson

Max Pooling



Single depth slice

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

Max pooling with 2×2 kernel size and stride 2

6	**8**
3	**4**

Introduces invariance to small spatial shifts No learnable parameters!

Slide: Johnson

Max Pooling Summary

Input: $C \times H \times W$
Hyperparameters:
- Kernel size: K
- Stride: S
- Pooling function (max, avg)

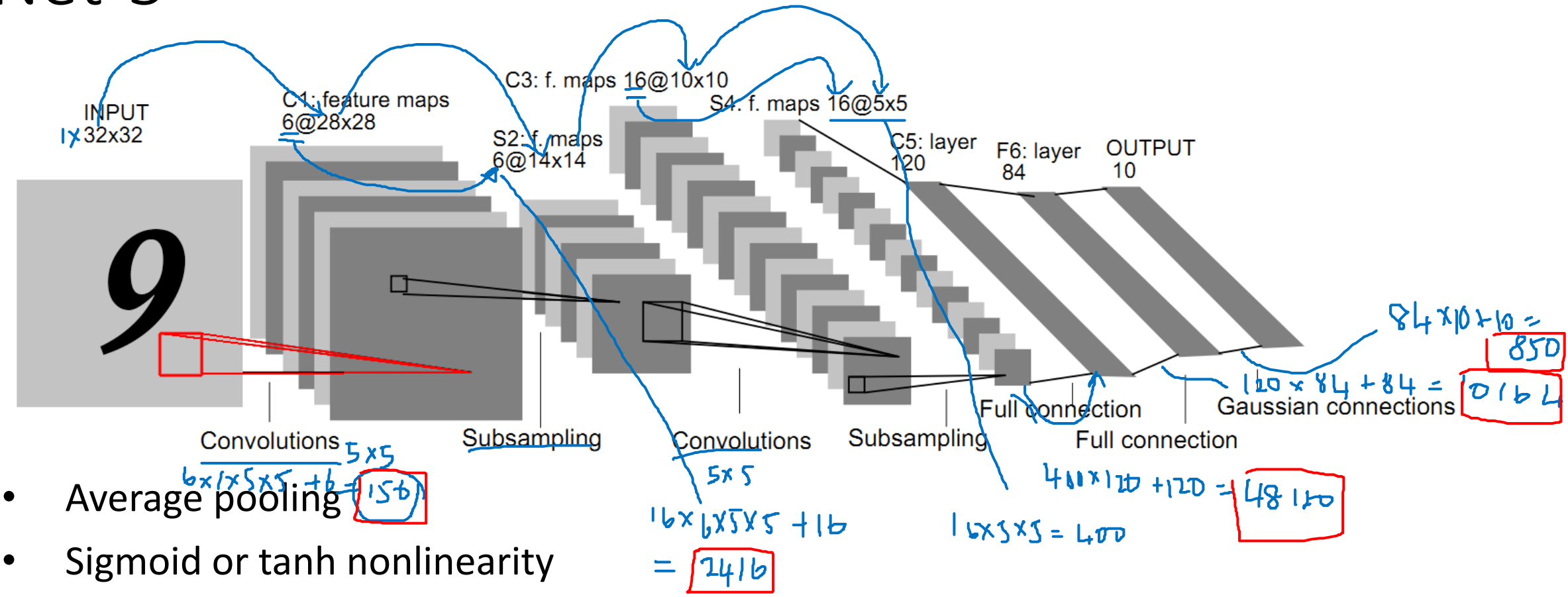
Output: $C \times H' \times W'$ where
- $H' = (H - K) / S + 1$
- $W' = (W - K) / S + 1$

Learnable parameters: None!

Common settings:
max, $K = 2, S = 2$
max, $K = 3, S = 2$ (AlexNet)

Slide: Johnson

LeNet-5



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K traiSning examples

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

Backpropagation

Batch Normalization (from Johnson's CNN slides)

https://www.tensorflow.org/tutorials/keras/classification

Existing Architectures

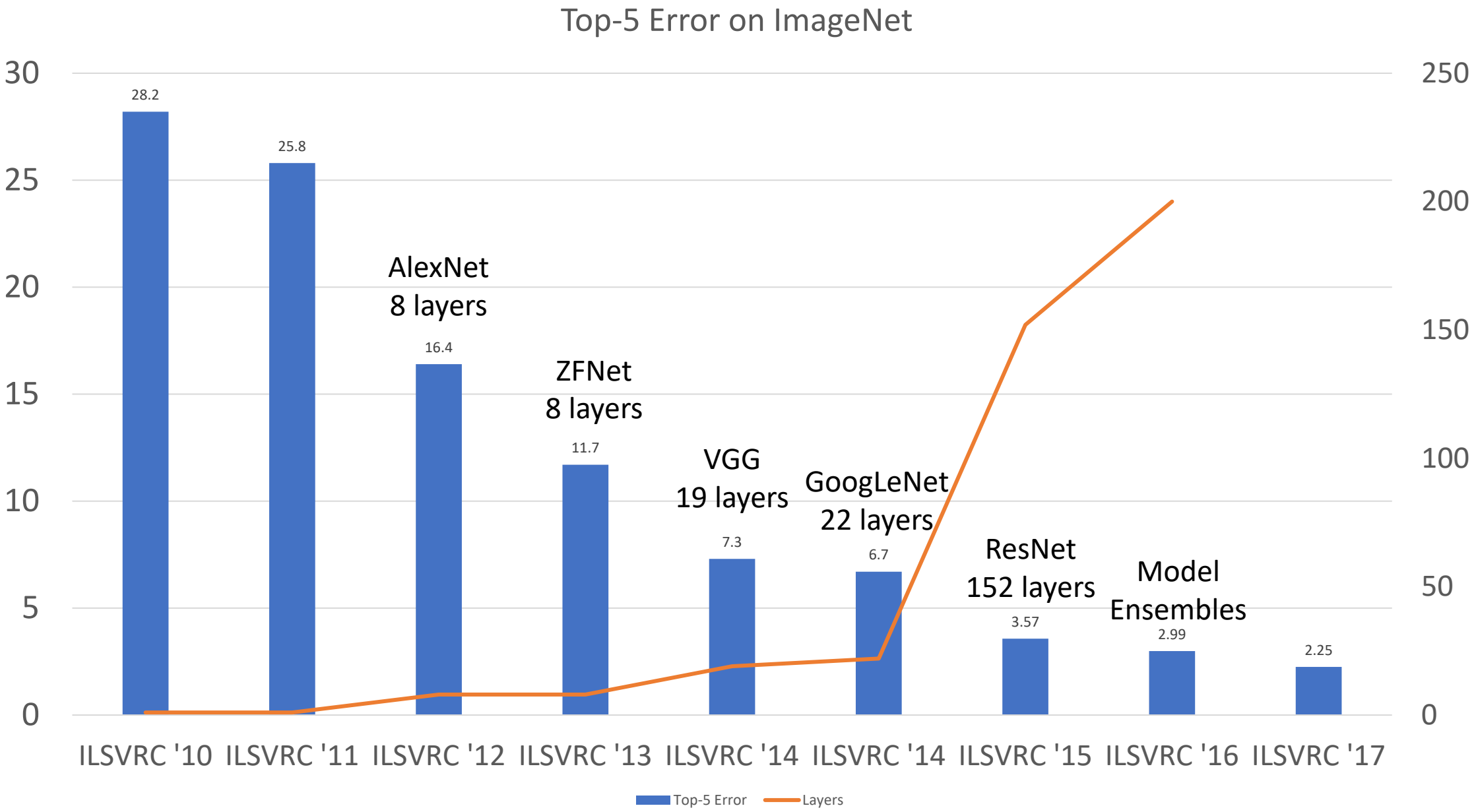
Slides are mainly from Johnson. Some slides are from Lazebnik. Some slides have been modified.

The Arrival of Big Visual Data…

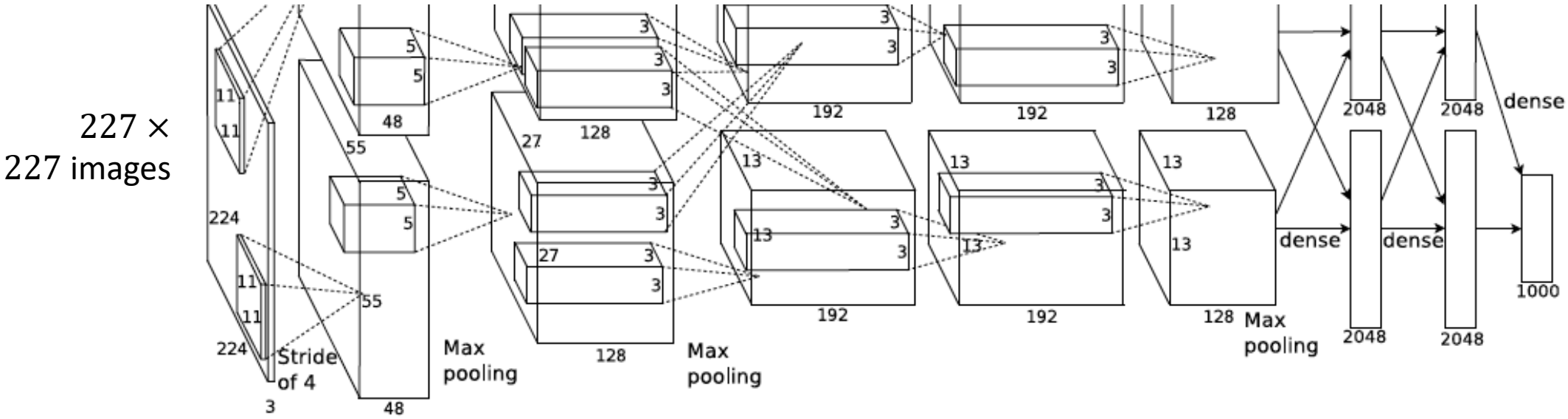
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes

www.image-net.org/challenges/LSVRC/

Slide: Lazebnik


Top-5 Error on ImageNet
30
25
20
15
10
5
0
250
200
150
100
50
0
28.2
25.8
16.4
11.7
7.3
6.7
3.57
2.99
2.25
AlexNet
8 layers
ZFNet
8 layers
VGG
19 layers
GoogLeNet
22 layers
ResNet
152 layers
Model
Ensembles
ILSVRC '10
ILSVRC '11
ILSVRC '12
ILSVRC '13
ILSVRC '14
ILSVRC '14
ILSVRC '15
ILSVRC '16
ILSVRC '17
Top-5 Error
Layers

AlexNet: ILSVRC 2012 Winner



Similar framework to LeNet but:

Max pooling, ReLU nonlinearity

More data and bigger model (7 hidden layers, 5 convolutional layers, 3 fully-connected layers 650K units, 60M params.)

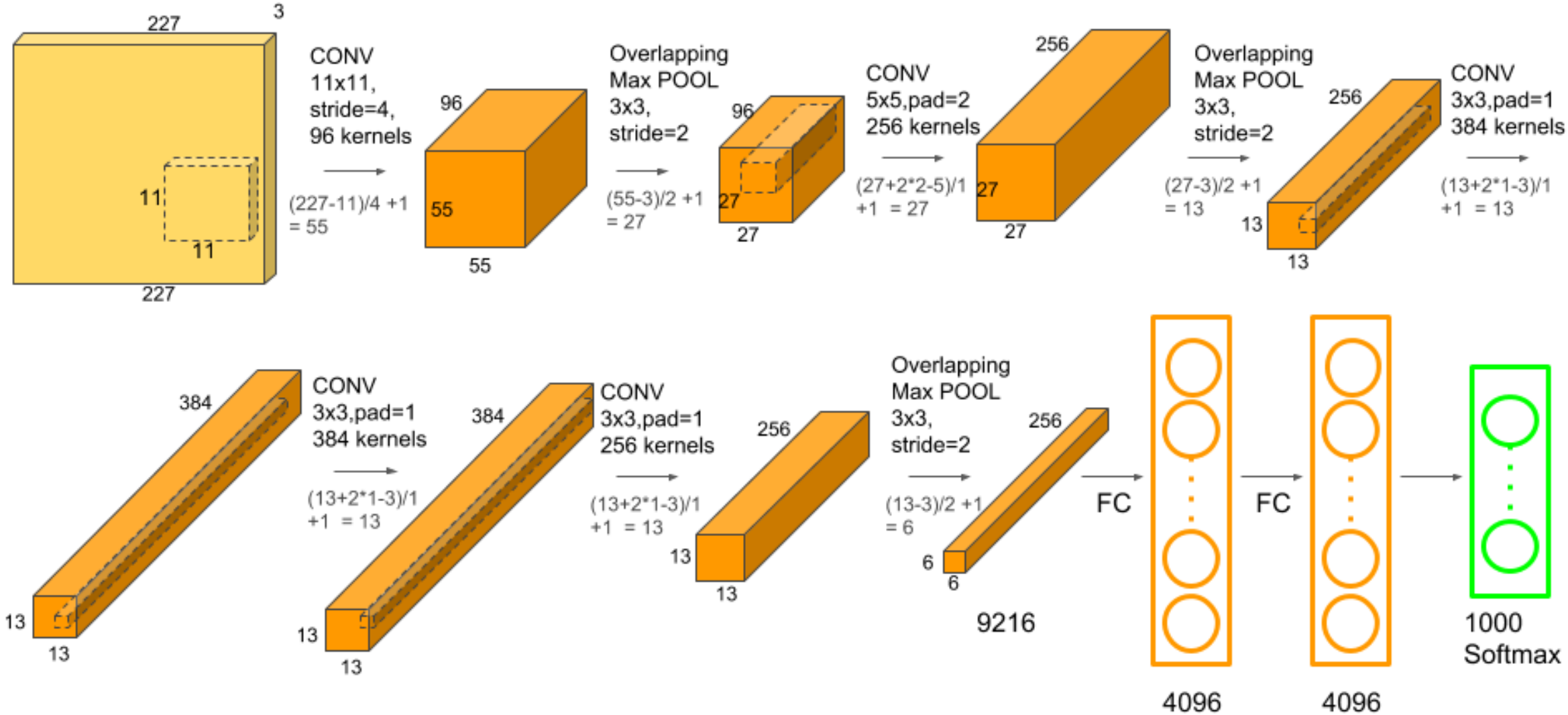
GPU implementation (two 3GB GTX580 cards, 50x speedup over CPU)

Trained on two GPUs for a week

Dropout regularization

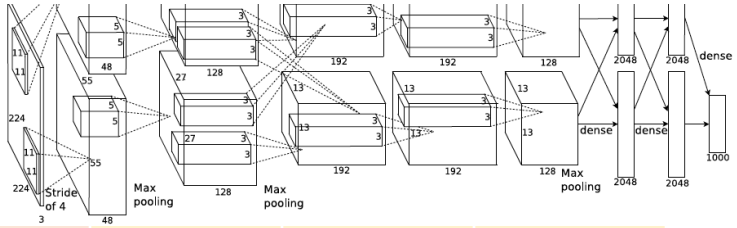
A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

AlexNet (Simplified)



https://www.learnopencv.com/understanding-alexnet/

AlexNet



	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv1	3	227	96	11	4	0	?	?	?	?	?

$W' = (W - K + 2P)/S + 1$
$= (227 - 11 + 2 \times 0)/4 + 1$
$= 55$

$\text{KB} = \text{No. of Elements} \times \text{Bytes per element}/1024$
$= C_{\text{out}} \times H' \times \text{W}' \times 4/1024$ (32-bit floating-point)
$= 96 \times 55 \times 55 \times 4/1024 = 1134$

Params. $=$ Weights $+$ Bias
$= C_{\text{out}} \times C_{\text{in}} \times K \times K + C_{\text{out}}$
$= 96 \times 3 \times 11 \times 11 + 96$
$= 34{,}944$

No. of floating-point operations (multiply-add)
$=$ No. of output elements $\times$ Operations per element
$= (96 \times 55 \times 55) \times (3 \times 11 \times 11) = 105{,}415{,}200$

Slide: Slightly modified from Johnson

AlexNet

	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv1	3	227	96	11	4	2	96	55	1134	35	105
pool1	96	55		3	2	0	?	?	?	?	?

$W' = (W - K + 2P)/S + 1$
$= (56 - 3 + 2 \times 0)/2 + 1$
$= 27.5 \rightarrow \text{floor} \rightarrow 27$

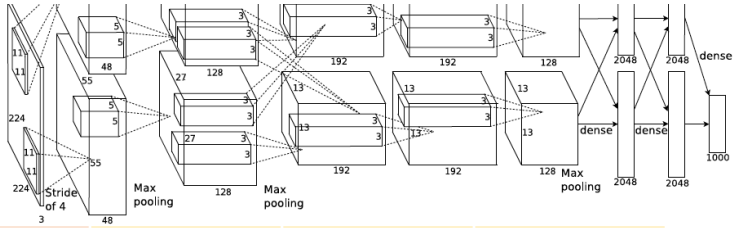
Pooling layers have no learnable parameters.

$\text{KB} = \text{No. of Elements} \times \text{Bytes per element}/1024$
$= C_{\text{out}} \times H' \times \text{W}' \times 4/1024$ (32-bit floating-point)
$= 96 \times 27 \times 27 \times 4/1024 = 273.375$

No. of floating-point operations (multiply-add)
$= \text{No. of output elements} \times \text{Operations per element}$
$= 96 \times 27 \times 27 \times 3 \times 3 = 629{,}856 = 0.6$ Mflops

Slide: Slightly modified from Johnson

AlexNet

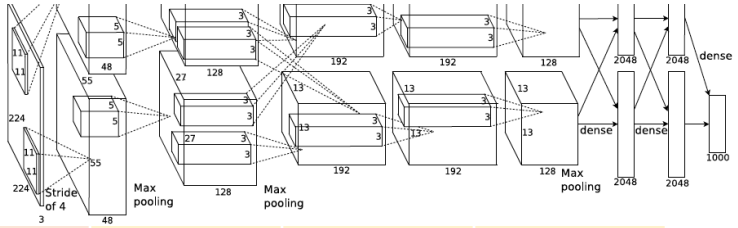


	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv1	3	227	96	11	4	2	96	55	1134	35	105
pool1	96	55		3	2	0	96	27	273	0	0
conv2	96	27	256	5	1	2	256	27	729	615	448
pool2	256	27		3	2	0	256	13	169	0	0
conv3	256	13	384	3	1	1	384	13	254	885	150
conv4	384	13	384	3	1	1	384	13	254	1,327	224
conv5	384	13	256	3	1	1	256	13	169	885	150
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0

Flatten output size
$= C_{\text{in}} \times H \times W$
$= 256 \times 6 \times 6$
$= 9{,}216$

Slide: Slightly modified from Johnson

AlexNet



	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv1	3	227	96	11	4	2	96	55	1134	35	105
pool1	96	55		3	2	0	96	27	273	0	0
conv2	96	27	256	5	1	2	256	27	729	615	448
pool2	256	27		3	2	0	256	13	169	0	0
conv3	256	13	384	3	1	1	384	13	254	885	150
conv4	384	13	384	3	1	1	384	13	254	1,327	224
conv5	384	13	256	3	1	1	256	13	169	885	150
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,753	38

Fc6 params

$= C_{\text{in}} \times C_{\text{out}} + C_{\text{out}}$

$= 9216 \times 4096 + 4096 = 37{,}752{,}832$

Fc6 flops

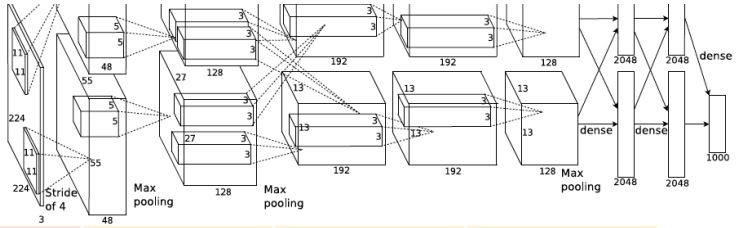
$= C_{\text{in}} \times C_{\text{out}}$

$= 9216 \times 4096 = 37{,}748{,}736$

Slide: Slightly modified from Johnson

AlexNet

Extensive trial and error!

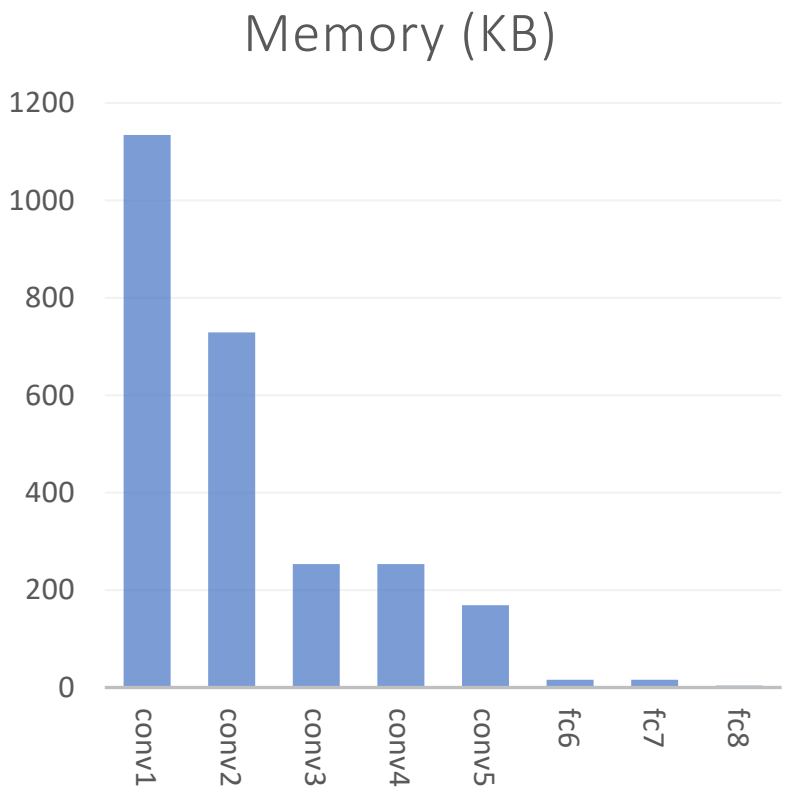


	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv1	3	227	96	11	4	2	96	55	1134	35	105
pool1	96	55		3	2	0	96	27	273	0	0
conv2	96	27	256	5	1	2	256	27	729	615	448
pool2	256	27		3	2	0	256	13	169	0	0
conv3	256	13	384	3	1	1	384	13	254	885	150
conv4	384	13	384	3	1	1	384	13	254	1,327	224
conv5	384	13	256	3	1	1	256	13	169	885	150
pool5	256	13		3	2	0	256	6	36	0	0
flatten	256	6					9216		36	0	0
fc6	9216		4096				4096		16	37,753	38
fc7	4096		4096				4096		16	16,781	17
fc8	4096		1000				1000		4	4,096	4

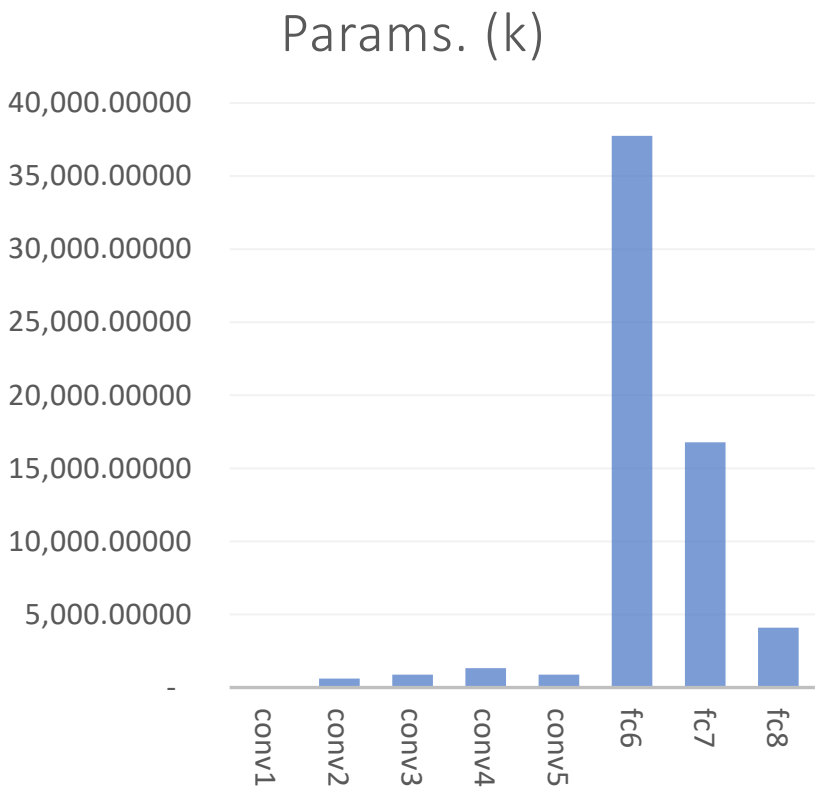
Slide: Slightly modified from Johnson

AlexNet Memory, Parameters, and MFLOPs

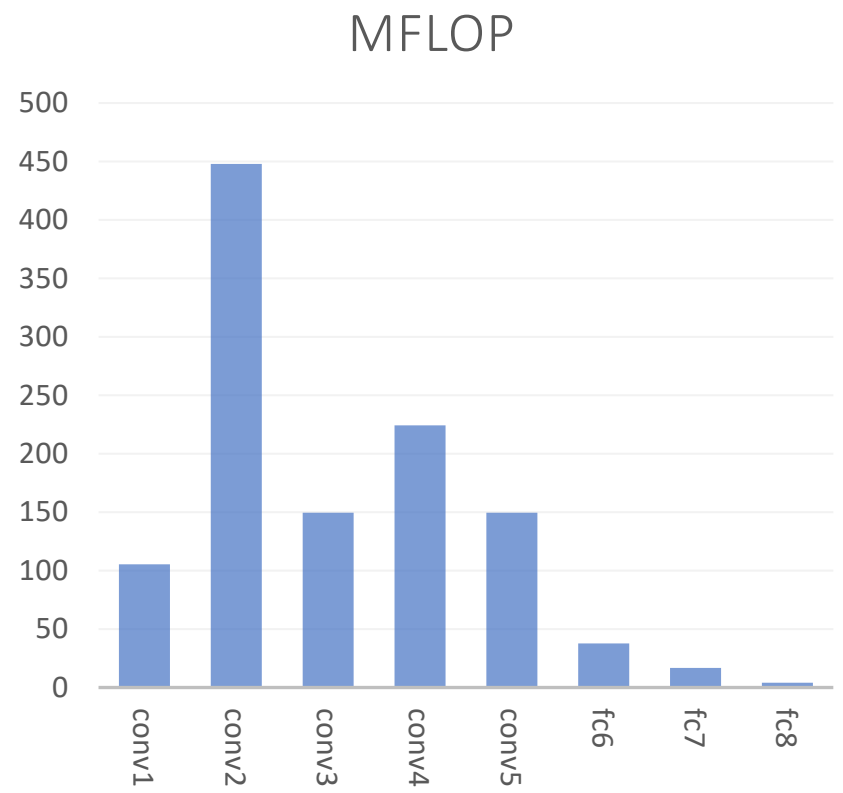
Most of the memory usage is in the early conv. layers



Nearly all parameters are in the fully-connected layers



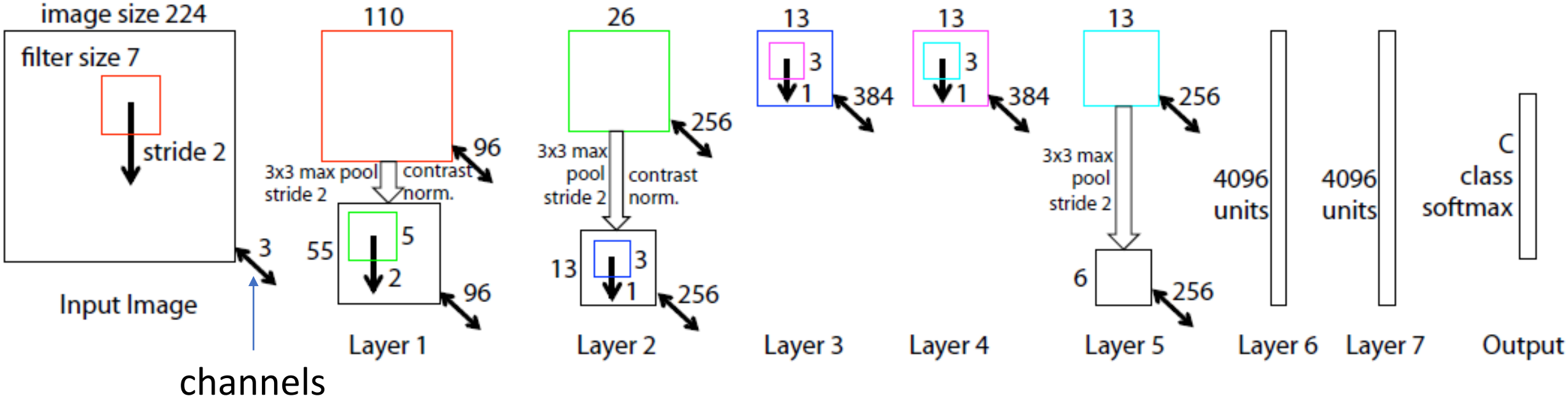
Most floating-point ops. occur in the Conv. layers



Slide: Slightly modified from Johnson

ZFNet (2013 Winner, 2014)

ImageNet top 5 error: 16.4% -> 11.7%



Similar to AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2: down sampling only by a factor of 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

More trial and error, bigger networks work better, not principle to make the networks better

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

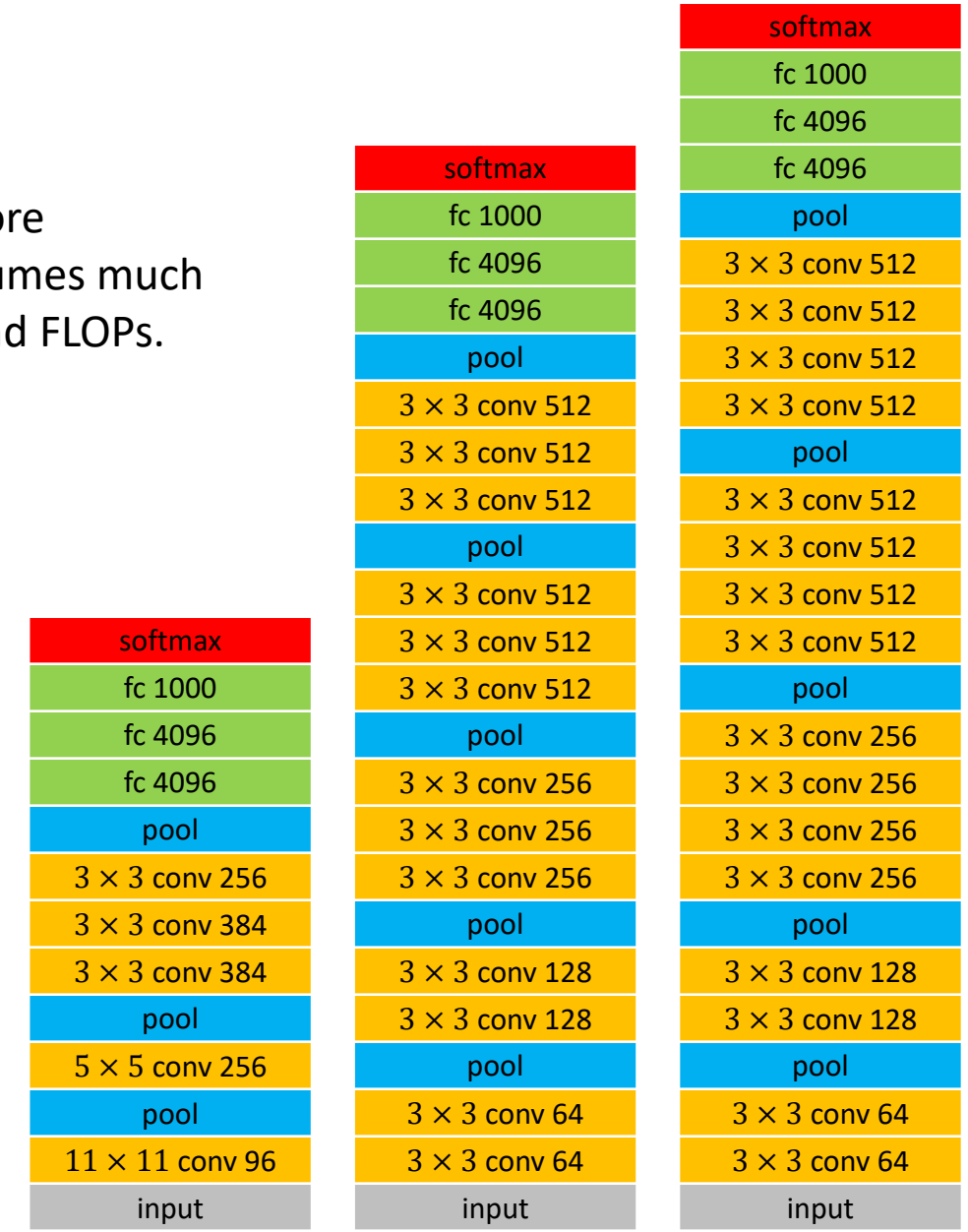
Slide: Slightly modified from Johnson

VGGNet

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

VGG has much more parameters, consumes much more memory, and FLOPs.

AlexNet: 5 conv. layers
VGG has 5 conv. stages:
Stage 1: conv-conv-pool
Stage 2: conv-conv-pool
Stage 3: conv-conv-conv-[conv]-pool
Stage 4: conv-conv-conv-[conv]-pool
Stage 5: conv-conv-conv-[conv]-pool



Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

Slide: Slightly modified from Johnson

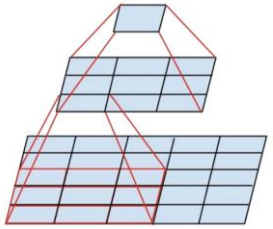
VGGNet

VGG Design rules:

All conv are 3x3 stride 1 pad 1

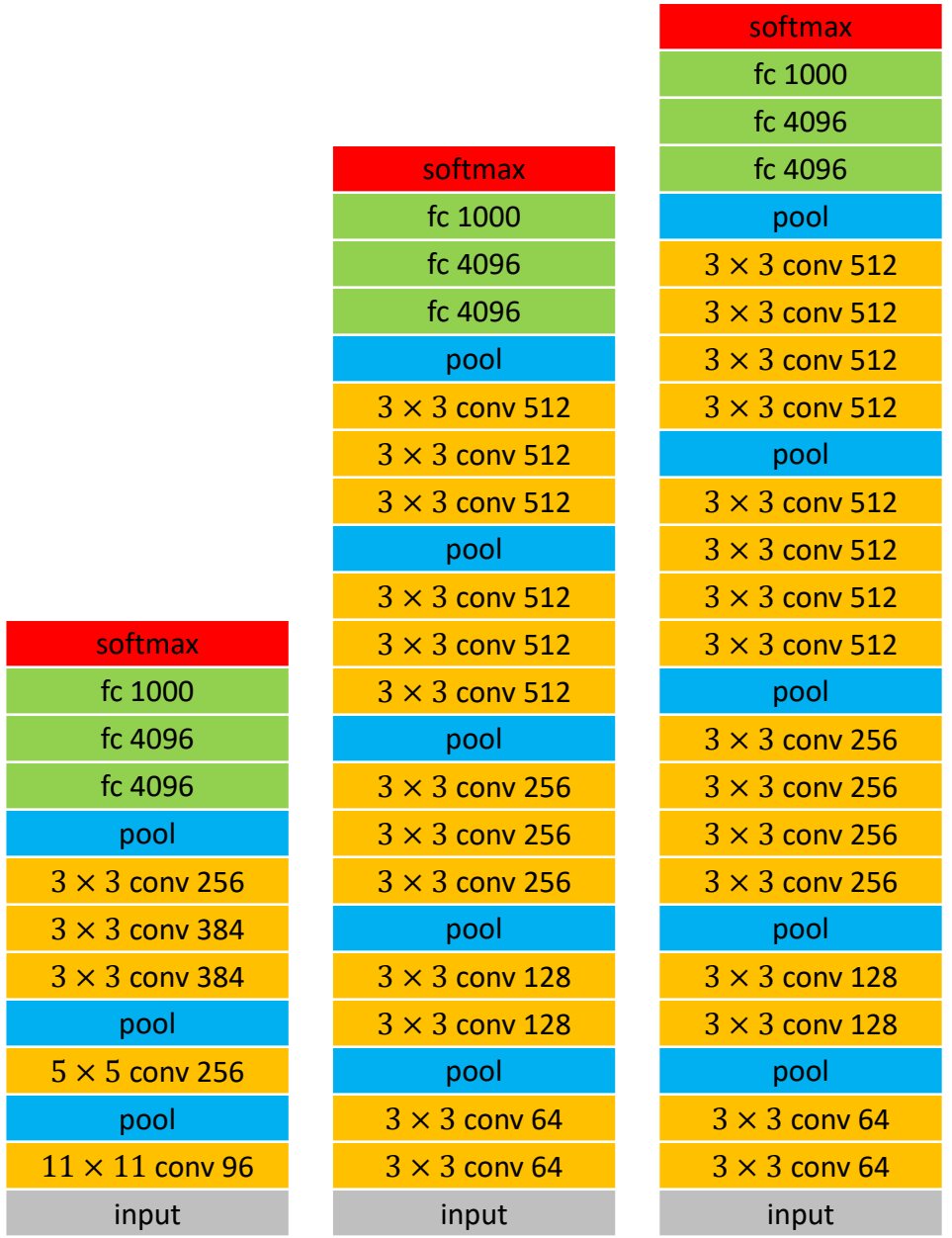
All max pool are 2x2 stride 2

After pool, double #channels



Two 3×3 conv has same receptive field as a single 5×5 conv., but has fewer parameters and takes less computation!

	Option 1: One 5×5 conv.	**Option 2:** Two 3×3 conv.
Layer	Conv($5 \times 5, C \rightarrow C$)	Conv($3 \times 3, C \rightarrow C$) Conv($3 \times 3, C \rightarrow C$)
Params	$25C^2$	$18C^2$
FLOPs	$25C^2HW$	$18C^2HW$



Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

Slide: Slightly modified from Johnson

VGGNet

VGG Design rules:

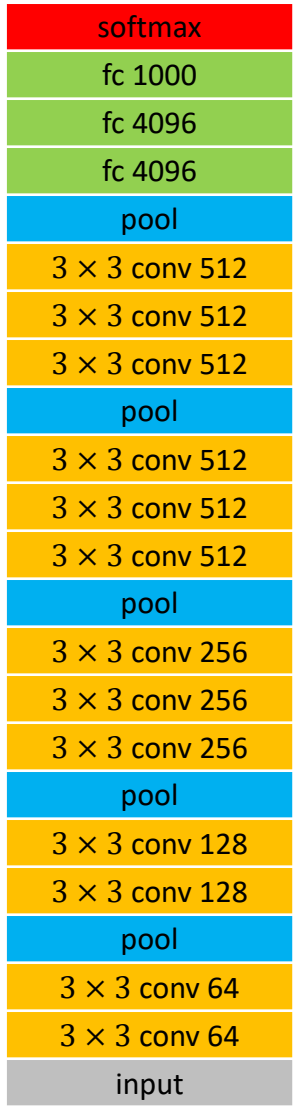
All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

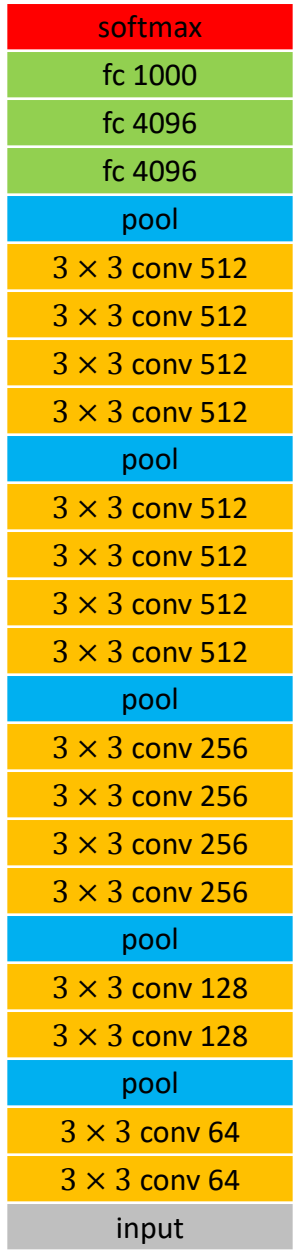
After pool, double #channels

Input	$C \times 2H \times 2W$	$2C \times H \times W$
Layer	Conv$(3 \times 3, C \to C)$	Conv$(3 \times 3, 2C \to 2C)$
Mem.	$4HWC$	$2HWC$
Params	$9C^2$	$36C^2$
FLOPs	$36HWC^2$	$36HWC^2$

AlexNet (8)



VGG16 (16)

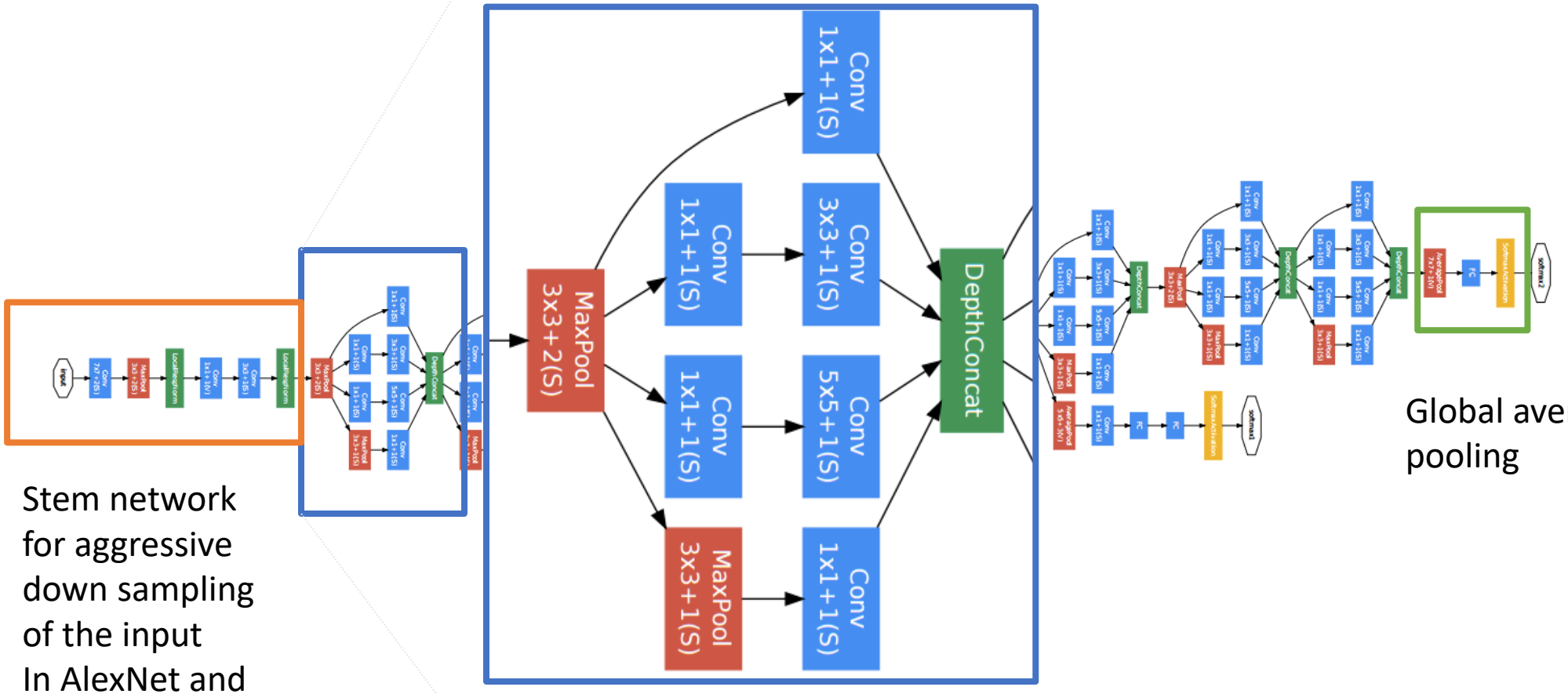


VGG19 (19)

Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

Slide: Slightly modified from Johnson

GoogLeNet (2014):



Stem network for aggressive down sampling of the input
In AlexNet and VGG, early layers were expensive.

Inception module: four parallel branches, all kernel sizes.

Global average pooling

C. Szegedy et al., Going deeper with convolutions, CVPR 2015

Stem Network in GoogLeNet

	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
conv	3	224	64	7	2	3	64	112	3136	9	118
pool	64	112	64	3	2		64	56	784	0	2
conv	64	56	64	1	1	0	64	56	784	4	13
conv	64	56	192	3	1	1	192	56	2352	111	347
pool	192	56	192	3	2	0	192	28	588	0	1

Stem network at the start aggressively down samples input
(Recall in VGG-16: Most of the compute was at the start)

Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

Slide: Slightly modified from Johnson

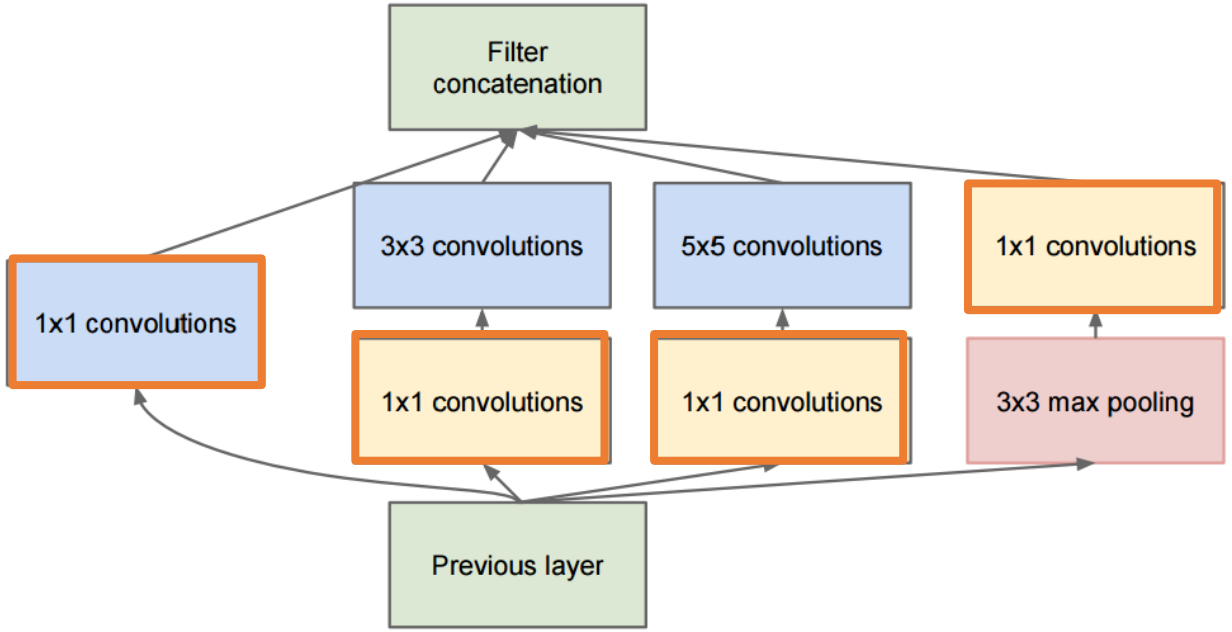
GoogLeNet Inception Module

Inception module

Local unit with parallel branches

Local structure repeated many times throughout the Network

Uses 1x1 "Bottleneck" layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)



Szegedy et al, "Going deeper with convolutions", CVPR 2015

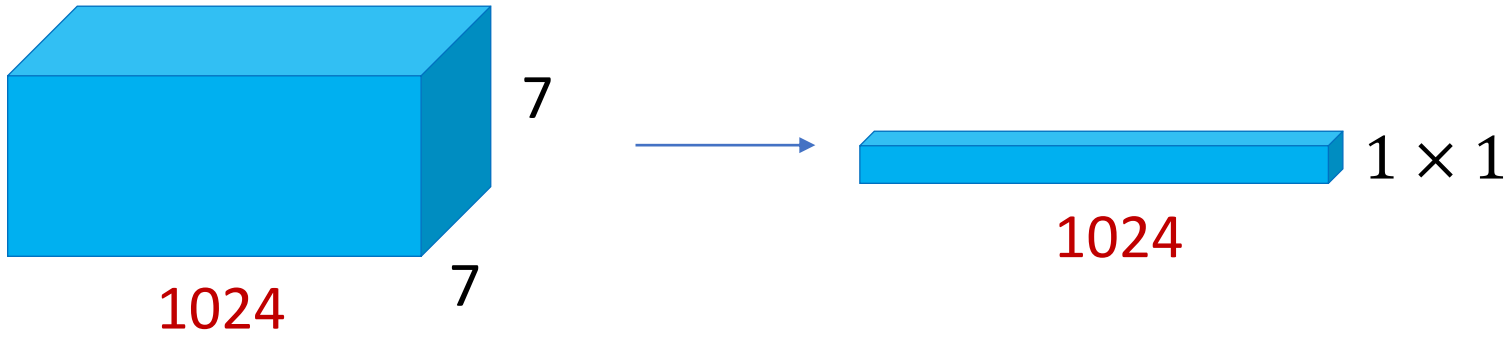
Slide: Slightly modified from Lazebnik

GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses global average pooling to collapse spatial dimensions, and one linear layer to produce class scores
(Recall VGG-16: Most parameters were in the FC layers!)

	Input Size		Layer				Output size				
Layer	C	H / W	Filters	Kernel	Stride	Pad		H' / W'	Mem. (KB)	Param. (k)	Flops (M)
	C_{in}		C_{out}	K	S	P	C_{out}				
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000		0	1025	1

Special information destroyed not by flattening, but by global average pooling. Then only one fc layer.

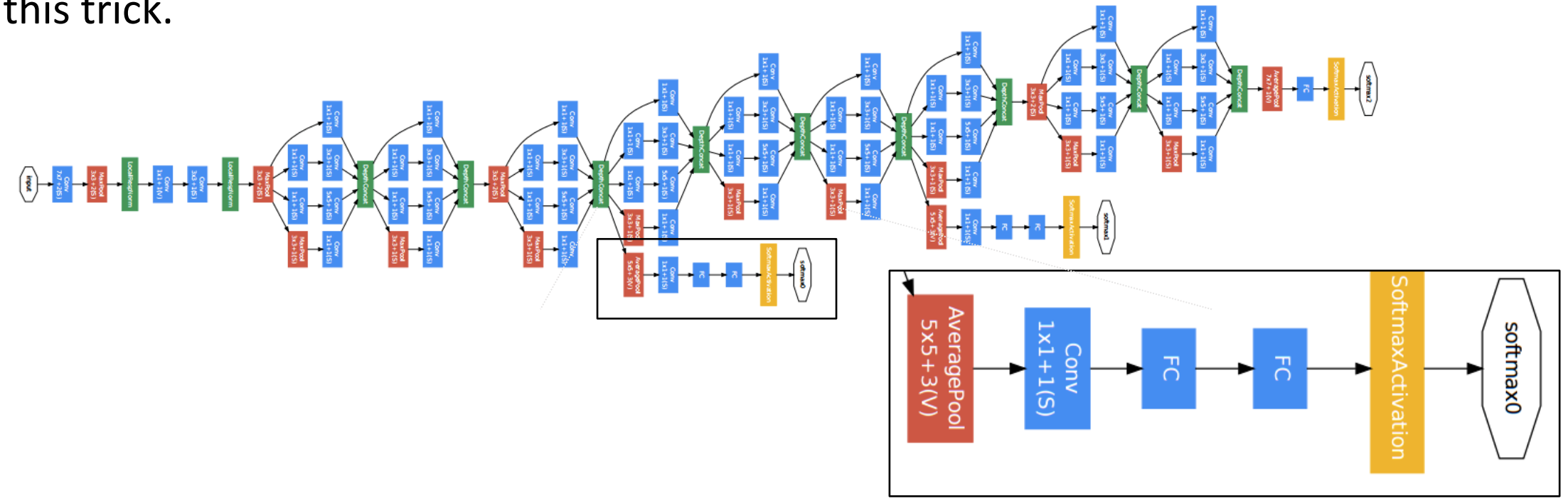


GoogLeNet: Auxiliary Classifiers

Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly.

As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss

GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick.



Auxiliary classifier

Slide: Slide modified from Lazebnik with text from Johnson.

Summary

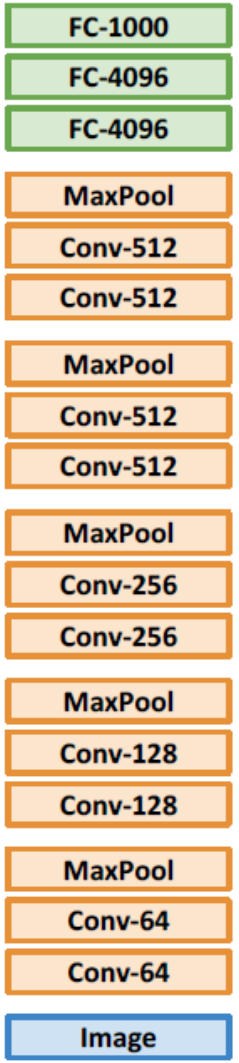
- ImageNet, ILSVRC
- AlexNet: max pooling, ReLU nonlinearity, large no. of parameters
- VGG: all conv are 3x3 stride 1 pad 1, all max pool are 2x2 stride 2, after pool, double #channels
- GoogLeNet: stem network, inception module, global average pooling, auxiliary classifiers

Transfer Learning

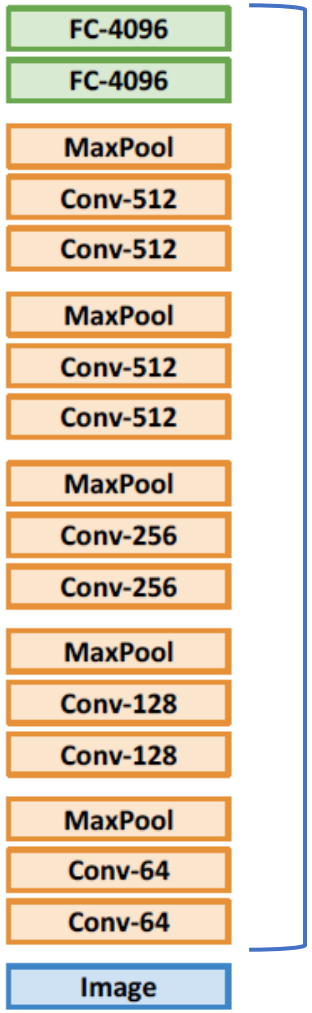
To be moved to the End of DL04

Transfer Learning with CNN

Train on ImageNet



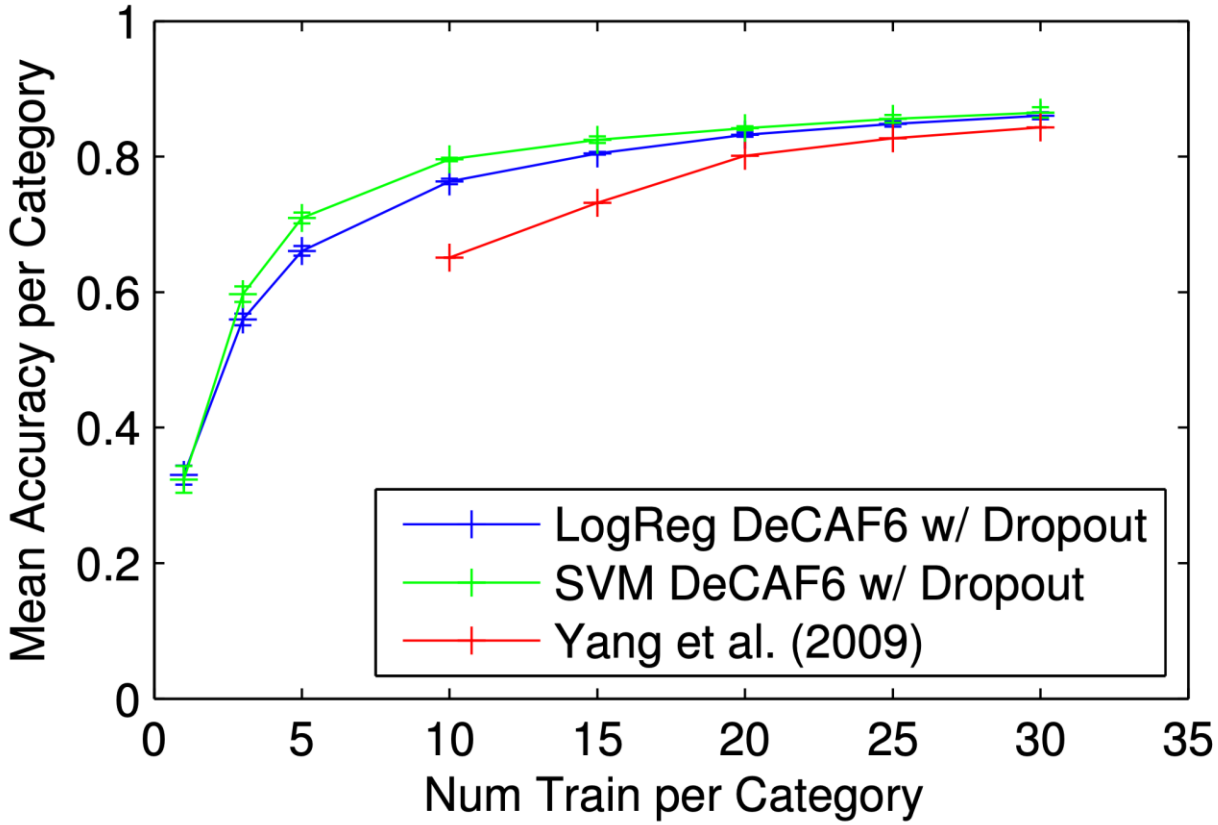
Use CNN as a feature extractor



Remove last layer

Freeze these

Classification on Caltech-101

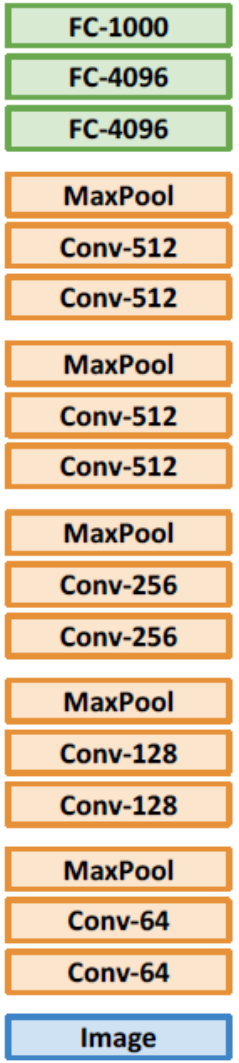


Slide: Slightly modified from Johnson

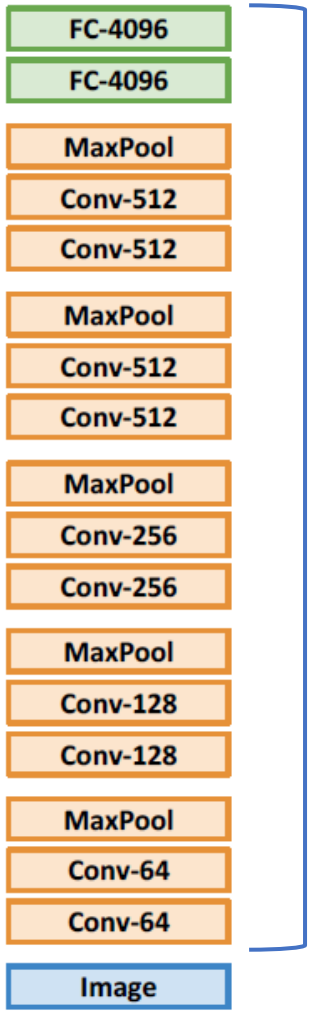
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNN

Train on ImageNet



Use CNN as a feature extractor

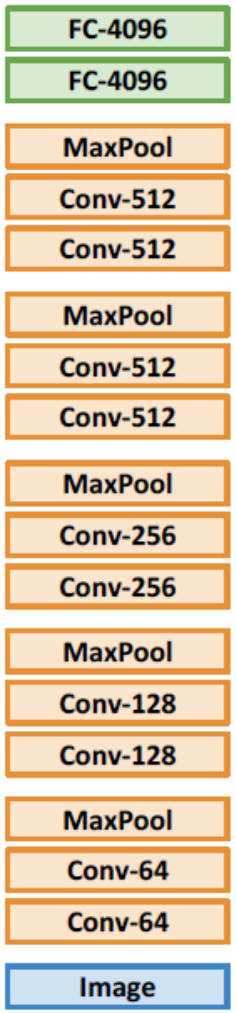


Remove last layer

Freeze these

Bigger Dataset

Fine Tuning



Continue training CNN for new task!

Some tricks:
- Train with feature extraction first before fine-tuning
- Lower the learning rate: use ~1/10 of LR used in original training
- Sometimes freeze lower layers to save computation

Slide: Slightly modified from Johnson

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

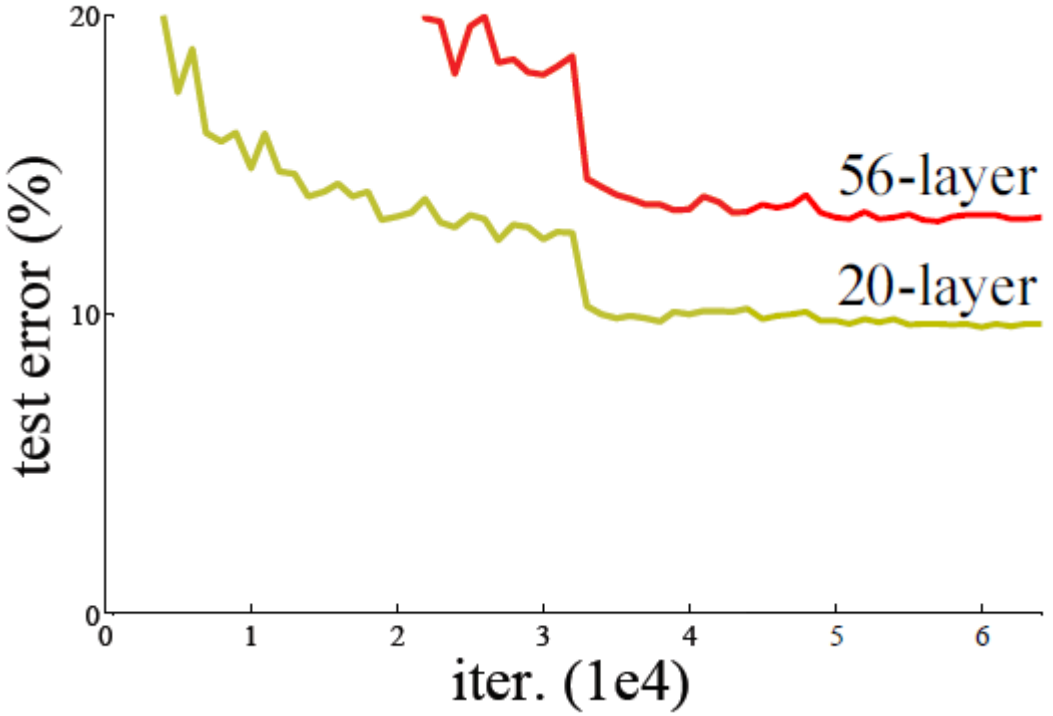
Residual Networks (ResNet): 2015

With batch normalization deeper models could be trained.

Deeper model does worse than shallow model!

Biggert not better, in contract to previous experience.

Initial guess: Deep model is **overfitting** since it is much bigger than the other model



In fact the deep model seems to be underfitting since it also performs worse than the shallow model on the training set! It is actually underfitting. 56-layer network has not been able to emulate he 20-layer network.

Slide: Slightly modified from Johnson

He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

ResNets

A deeper model can **emulate** a shallower model: copy layers from shallower model, set extra layers to identity.

Thus deeper models should do at least as good as shallow models.

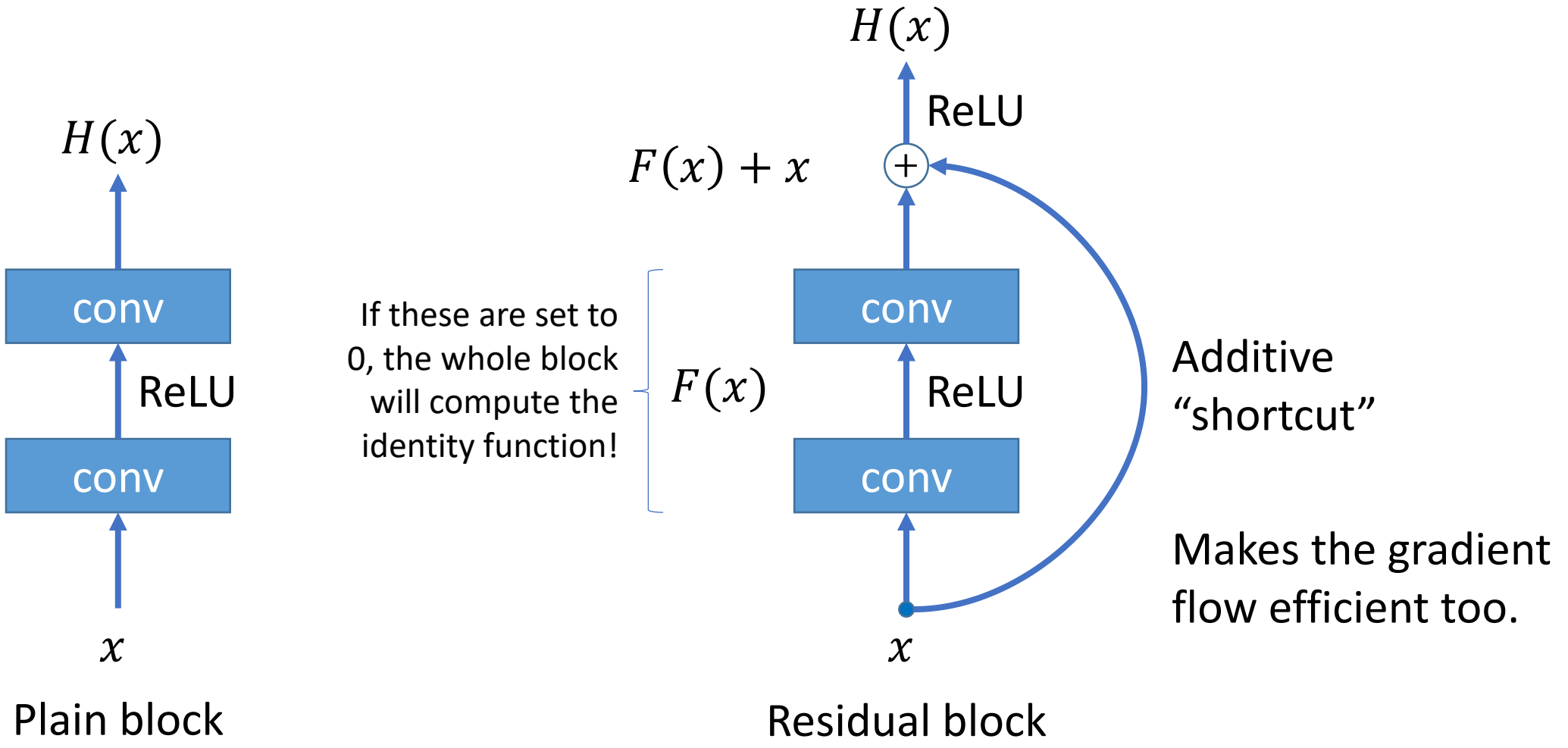
Hypothesis: This is an **optimization** problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models.

Solution: Change the network so learning identity functions with extra layers is easy!

Slide: Slightly modified from Johnson

Residual Networks

Solution: Change the network so learning identity functions with extra layers is easy!



Slide: Slightly modified from Johnson

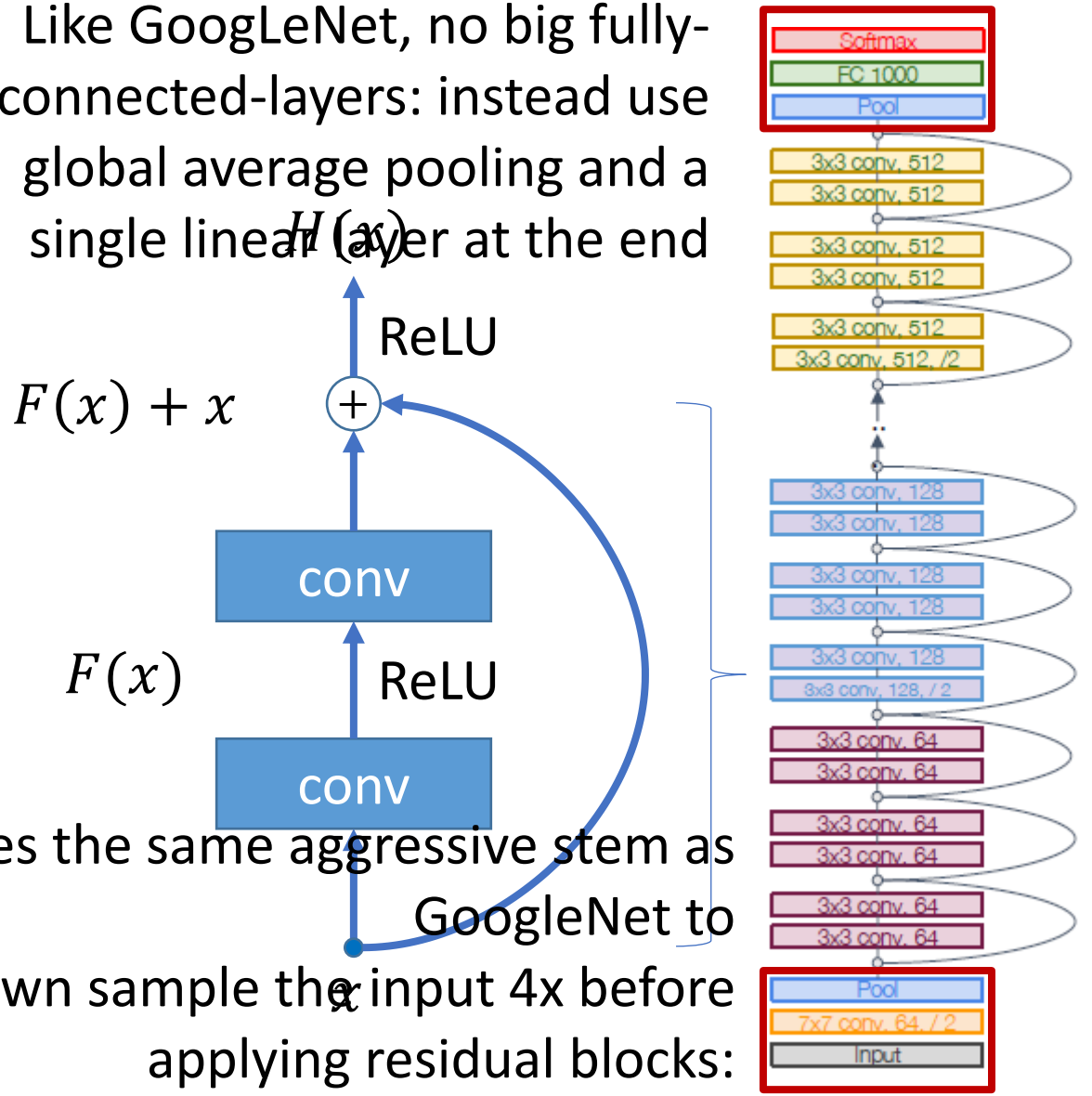
Residual Networks

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

Network is divided into stages: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels

Like GoogLeNet, no big fully-connected-layers: instead use global average pooling and a single linear layer at the end



Uses the same aggressive stem as GoogleNet to down sample the input 4x before applying residual blocks:

Slide: Slightly modified from Johnson

Residual Networks

ResNet-18:

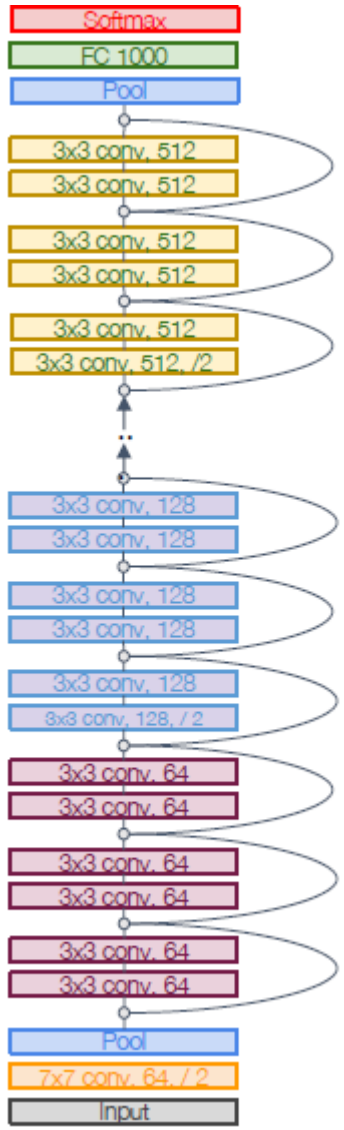
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear
ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:

Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear
ImageNet top-5 error: 8.58
GFLOP: 3.6

VGG-16:

ImageNet top-5 error: 9.62
GFLOP: 13.6



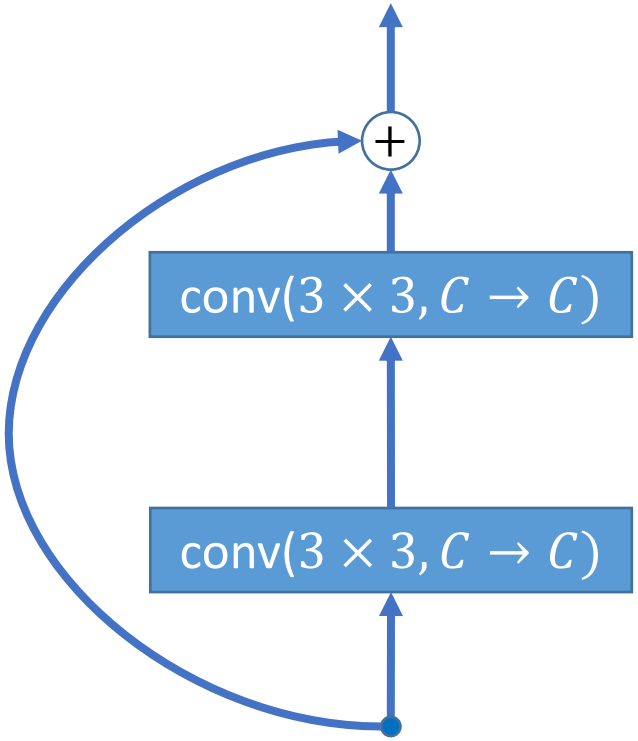
He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016
Error rates are 224x224 single-crop testing, reported by torchvision

Slide: Slightly modified from Johnson

Residual Networks: Bottleneck Block

Reduces the number of channels before computing a 3×3 convolution. Deeper networks without increasing the computational cost.

More layers, less computational cost!

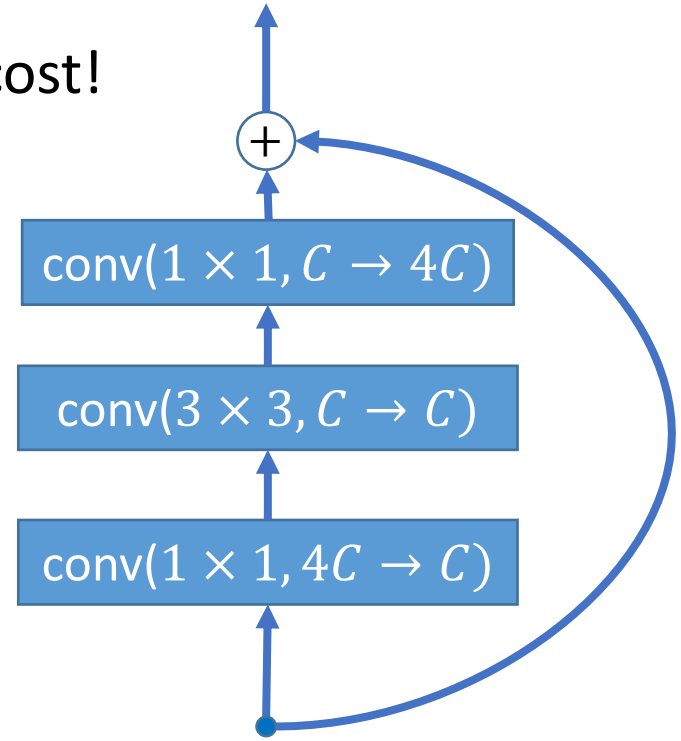


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

Basic residual block

Total FLOPs: $18HWC^2$



FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

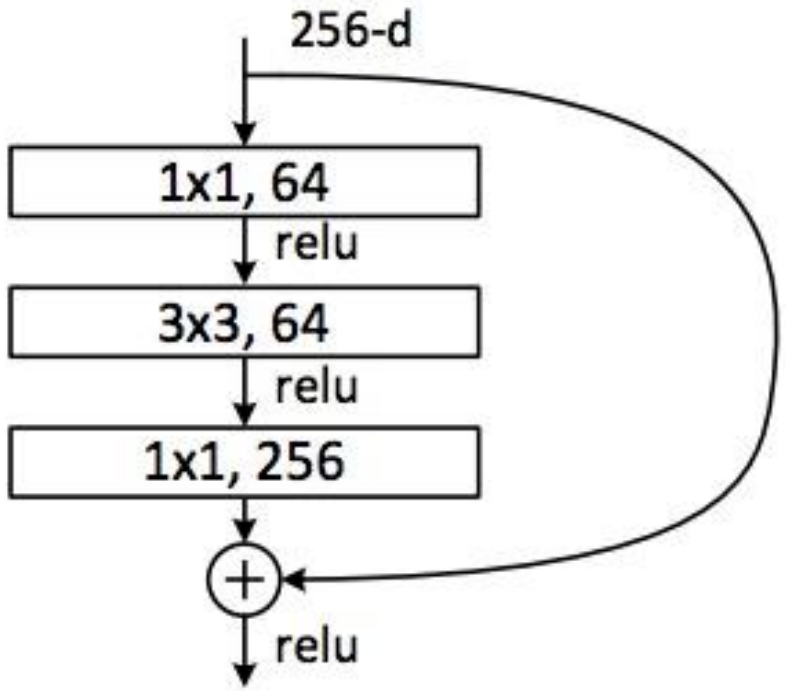
Total FLOPs: $17HWC^2$

Bottleneck residual block

Slide: Slightly modified from Johnson

Residual Networks

Deeper residual module (bottleneck)

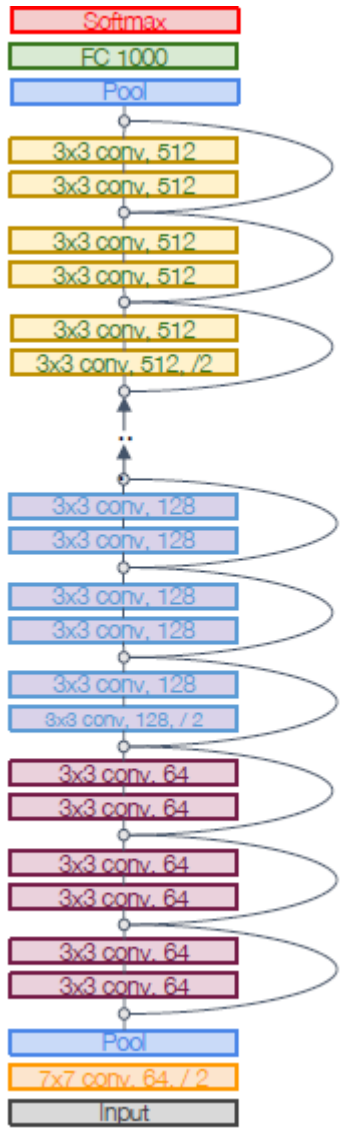


- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Slide: Lazebnik

Residual Networks

			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC layers	GFLOP	ImageNet top-5 error
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58



Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally heavy

Able to train very deep networks

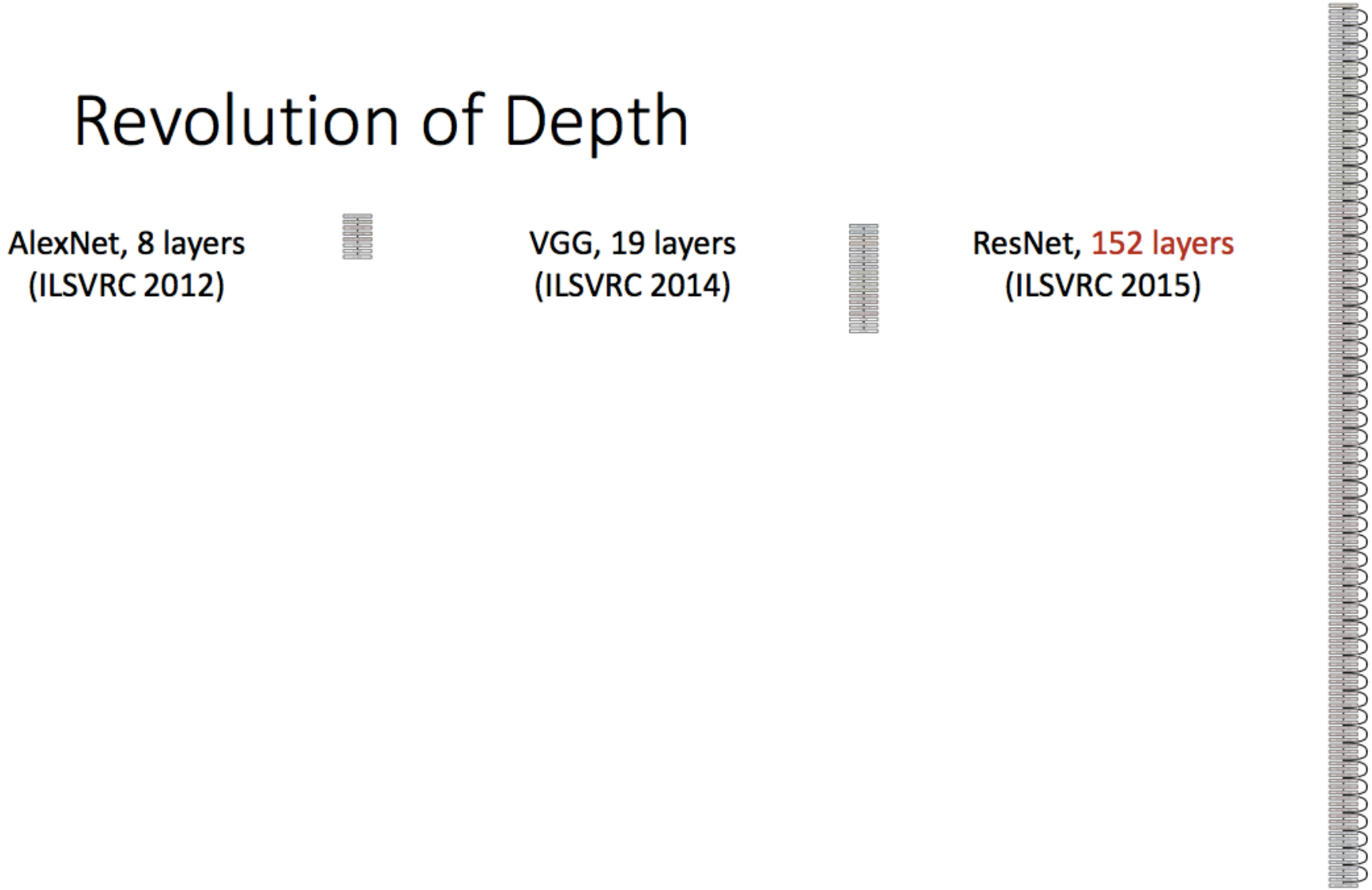
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Still widely used today!

He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Error rates are 224x224 single-crop testing, reported by torchvision

Slide: Slightly modified from Johnson

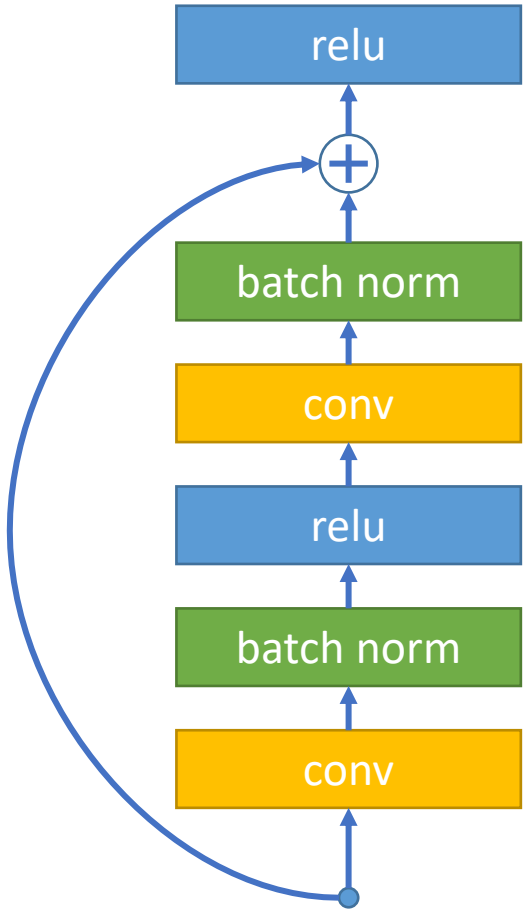
ResNet: Going Real Deep



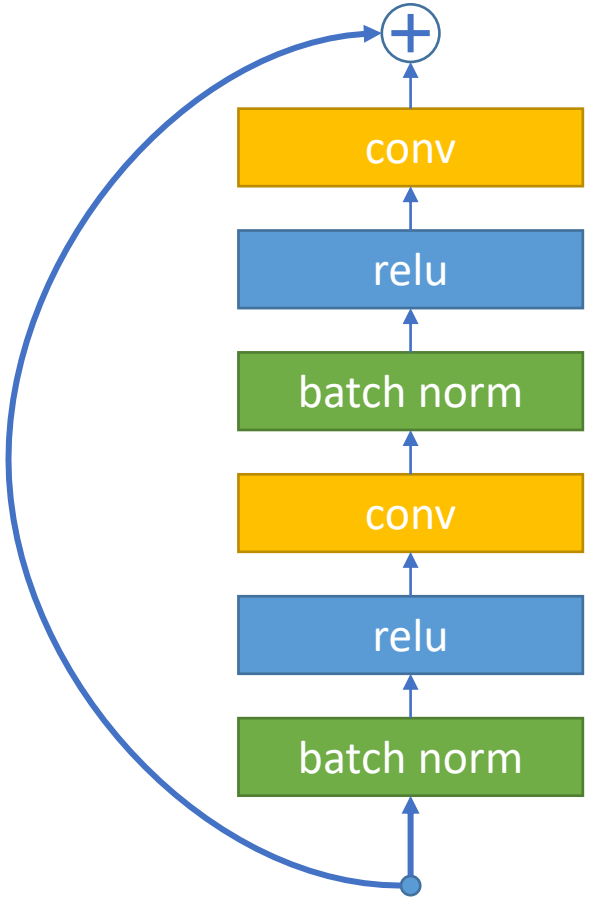
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

Improving Residual Networks: Block Design

Original ResNet block



"Pre-Activation" ResNet Block



Slight improvement in accuracy (ImageNet top-1 error)

ResNet-152: 21.3 vs 21.1
ResNet-200: 21.8 vs 20.7

Not actually used that much in practice

Slide: Slightly modified from Johnson

Bigger Not Better

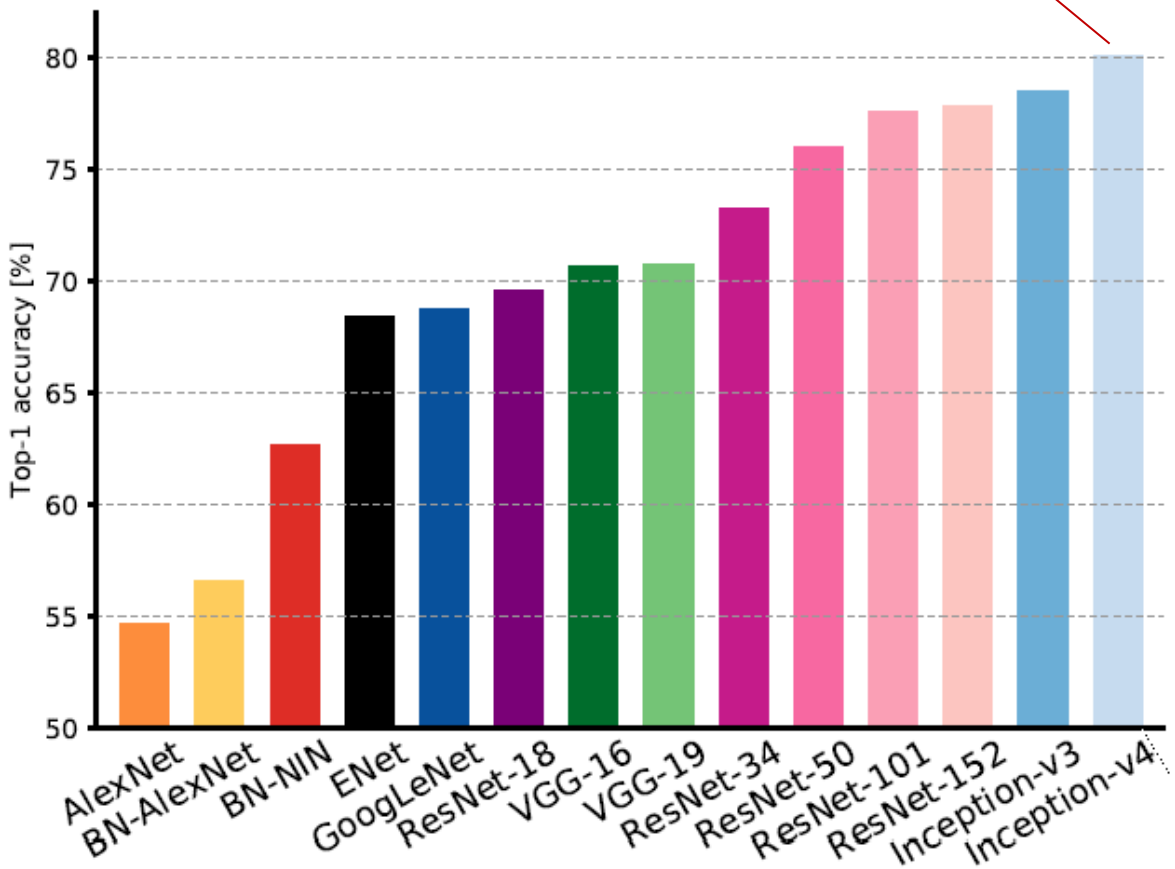
Innovations typically reduce parameters, despite deeper nets.

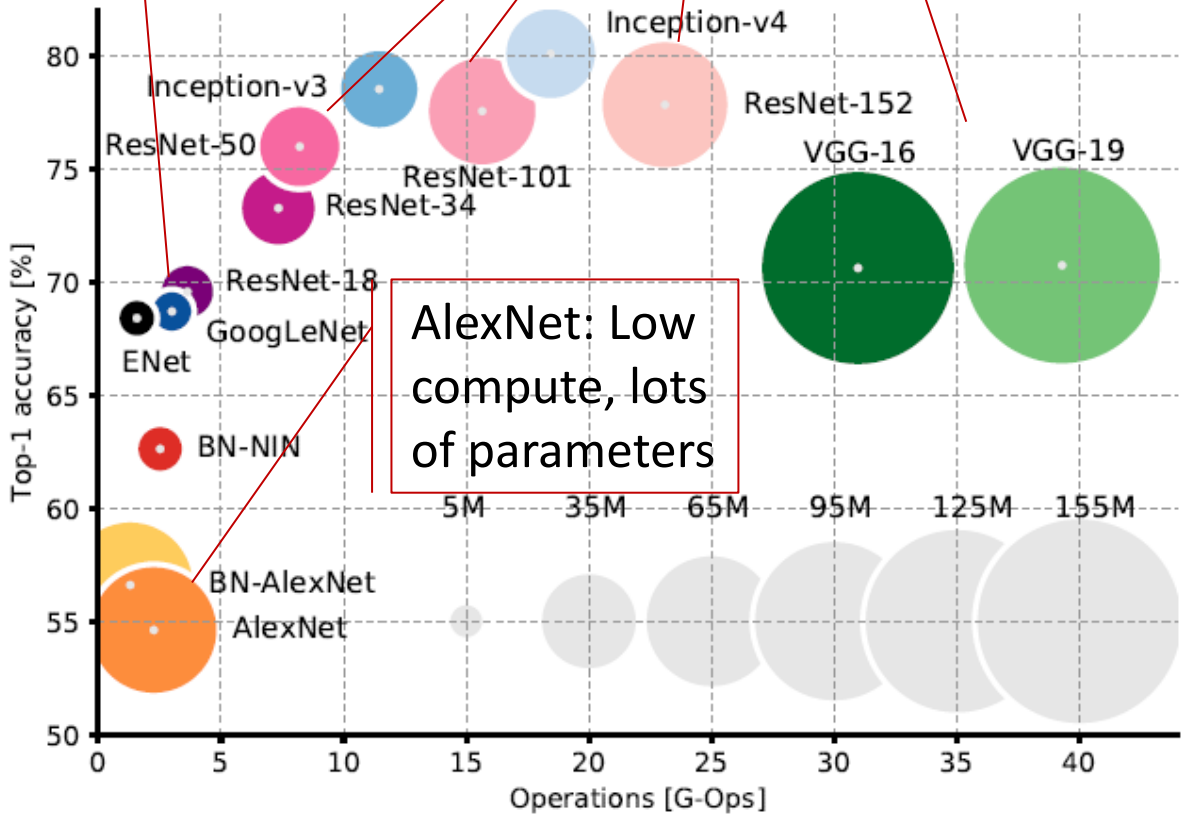
Inception-v4: Resnet + Inception!

GoogLeNet: Very efficient!

VGG: Highest memory, most operations

ResNet: Simple design, moderate efficiency, high accuracy





AlexNet: Low compute, lots of parameters

Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Slide: Slightly modified from Johnson

ImageNet 2016 winner: Model Ensembles

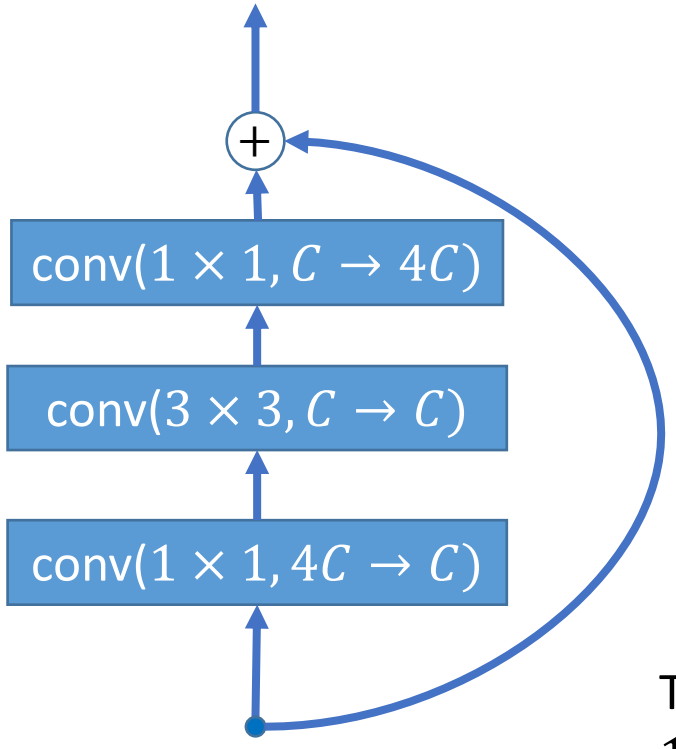
Multi-scale ensemble of Inception, Inception-Resnet, Resnet, Wide Resnet models

	Inception -v3	Inception -v3	Inception-ResNet-v2	ResNet-200	Wrn-68-3	Fusion (val.)	Fusion (test)
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

Slide: Slightly modified from Johnson

Shao et al, 2016

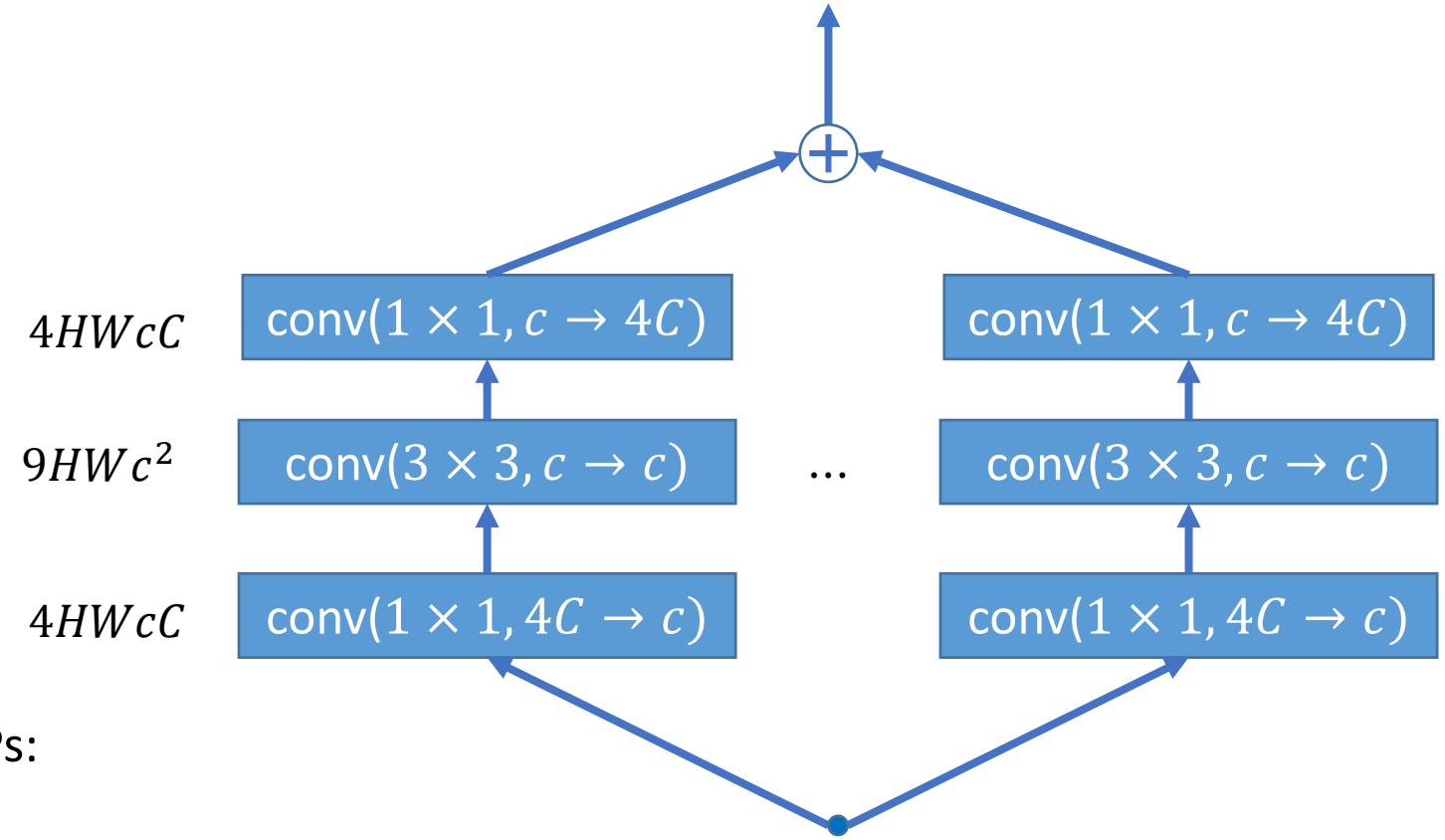
Improving ResNet: ResNeXt



Bottleneck residual block

Total FLOPs:
$17HWC^2$

G parallel pathways



Total FLOPs:
$G(4HWcC + 9HWc^2 + 4HWcC$

$G(4HWcC + 9HWc^2 + 4HWcC) = 17HWC^2$

Example: $C = 64, G = 4, c = 24;\ C = 64, G = 32, c = 4$

Xie et al, "Aggregated residual transformations for deep neural networks", CVPR 2017

Slide: Slightly modified from Johnson

Group Convolutions

Convolution with groups=1:

Normal convolution

Input: $C_{\text{in}} \times H \times W$

Weight: $C_{\text{out}} \times C_{\text{in}} \times K \times K$

Output: $C_{\text{out}} \times H' \times W'$

FLOPs: $C_{\text{out}} \times C_{\text{in}} \times K^2 \times H \times W$

All convolutional kernels touch all C_{in} channels of the input

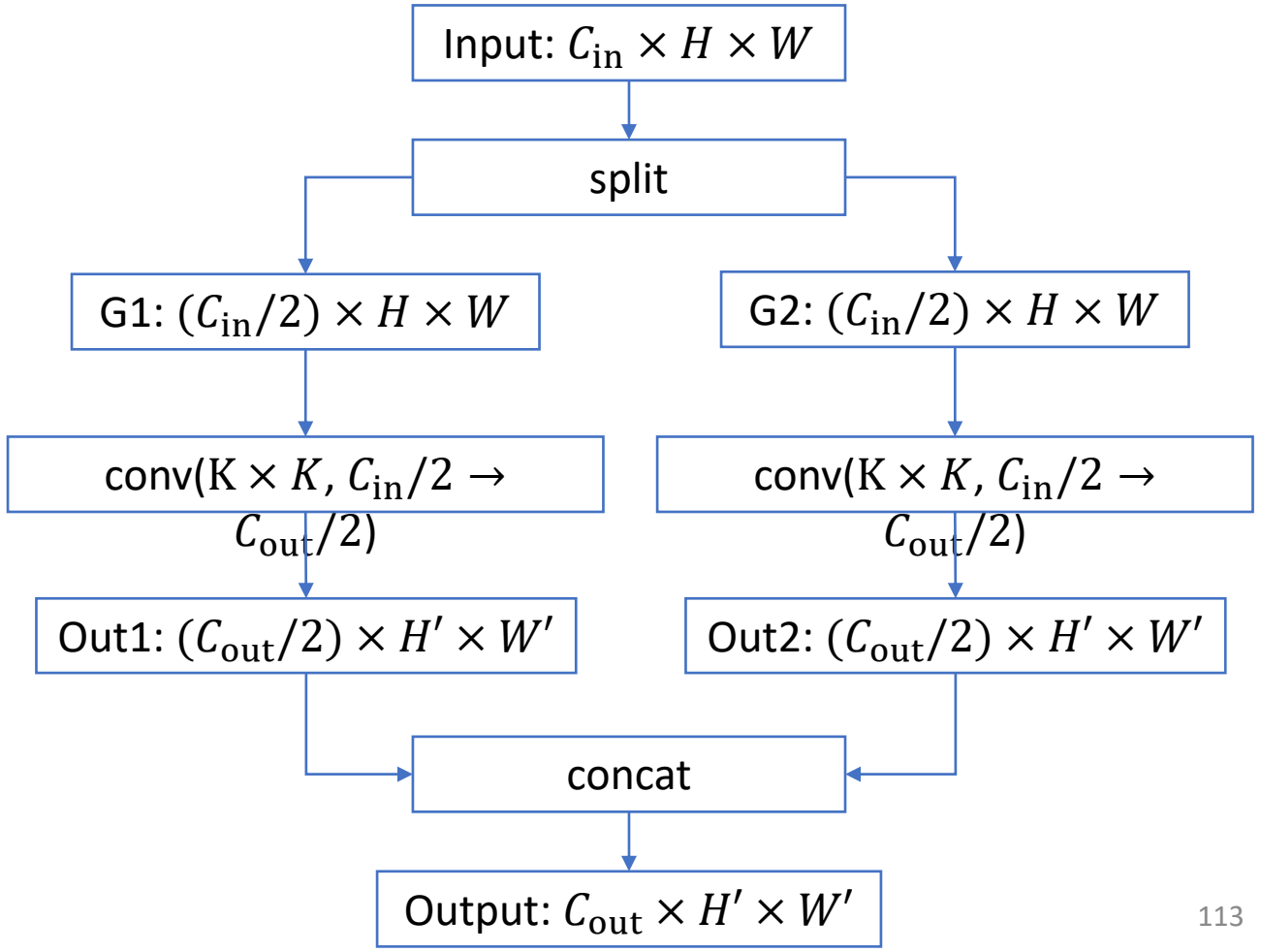
Depthwise Convolutions

Special case: $G = C_{\text{in}}, C_{\text{out}} = nC_{\text{in}}$

Each input channel is convolved with n different $K \times K$ filters to produce n output channels

Convolution with groups=2:

Two parallel convolution layers that work on half the channels



Slide: Slightly modified from Johnson

Group Convolutions

Convolution with groups=1:

Normal convolution

Input: $C_{\text{in}} \times H \times W$

Weight: $C_{\text{out}} \times C_{\text{in}} \times K \times K$

Output: $C_{\text{out}} \times H' \times W'$

FLOPs: $C_{\text{out}} \times C_{\text{in}} \times K^2 \times H \times W$

All convolutional kernels touch all C_{in} channels of the input

Depthwise Convolutions

Special case: $G = C_{\text{in}}, C_{\text{out}} = nC_{\text{in}}$

Each input channel is convolved with n different $K \times K$ filters to produce n output channels

Convolution with groups=G:

G parallel conv layers; each "sees" C_{in}/G input channels and produces C_{out}/G output channels

Input: $C_{\text{in}} \times H \times W$

Split to $G \times [(C_{\text{in}}/G) \times H \times W]$

Weight: $G \times (C_{\text{out}}/G) \times (C_{\text{in}} \times G) \times K \times K$

G parallel convolutions

Output: $G \times [(C_{\text{out}}/G) \times H' \times W']$

Concat to $C_{\text{out}} \times H' \times W'$

FLOPS: $C_{\text{out}} \times C_{\text{in}} \times K^2 \times H \times W$

Slide: Slightly modified from Johnson

ResNeXt: Maintain Computation by Adding Groups

Model	Groups	Group width	Top-1 Error
ResNet-50	1	64	23.9
ResNeXt-50	2	40	23
ResNeXt-50	4	24	22.6
ResNeXt-50	8	14	22.3
ResNeXt-50	32	4	22.2

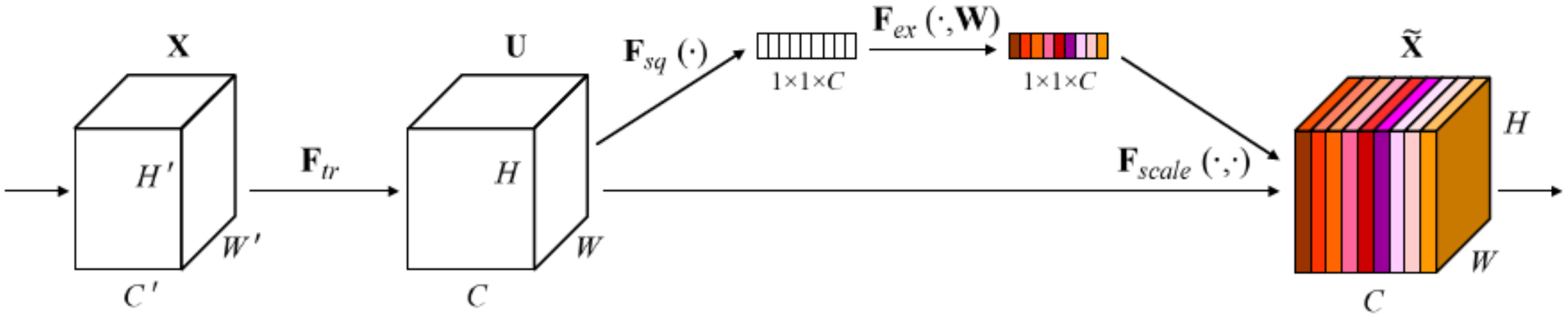
Model	Groups	Group width	Top-1 Error
ResNet-101	1	64	22.0
ResNeXt-101	2	40	21.7
ResNeXt-101	4	24	21.4
ResNeXt-101	8	14	21.3
ResNeXt-101	32	4	21.2

Adding groups improves performance with same computational complexity!

Slide: Slightly modified from Johnson

Xie et al, “Aggregated residual transformations for deep neural networks”, CVPR 2017

Squeeze-and-Excitation Networks (SENet, 2017)

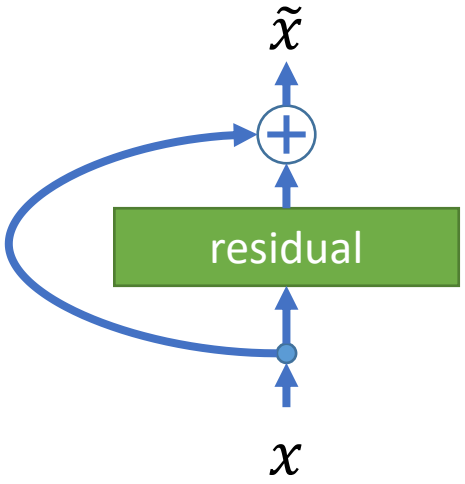


Adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

Slide: Slightly modified from Johnson

Hu et al, “Squeeze-and-Excitation networks”, CVPR 2018

Squeeze-and-Excitation Networks for ResNet

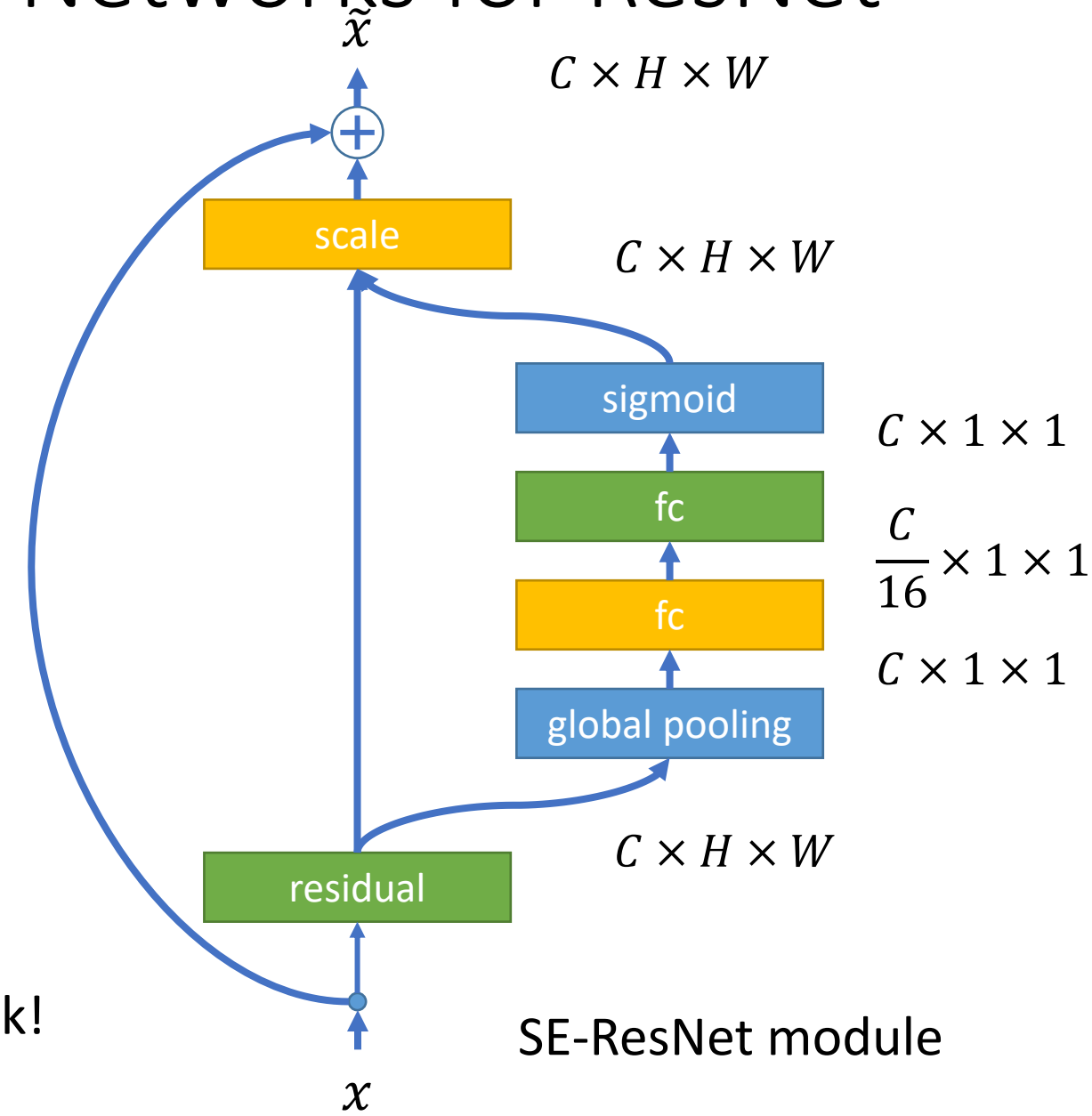


ResNet module

Adds a “Squeeze-and-excite” branch to each residual block that performs global pooling, full-connected layers, and multiplies back onto feature map

Adds global context to each residual block!

Won ILSVRC 2017 with ResNeXt-152-SE

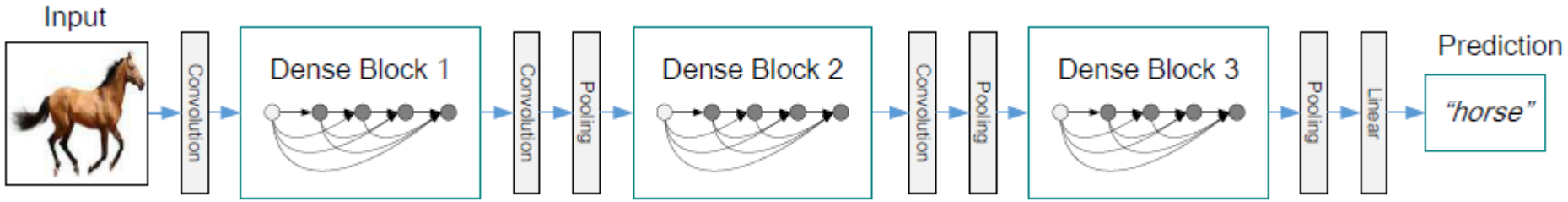


SE-ResNet module

Slide: Slightly modified from Johnson

Hu et al, “Squeeze-and-Excitation networks”, CVPR 2018

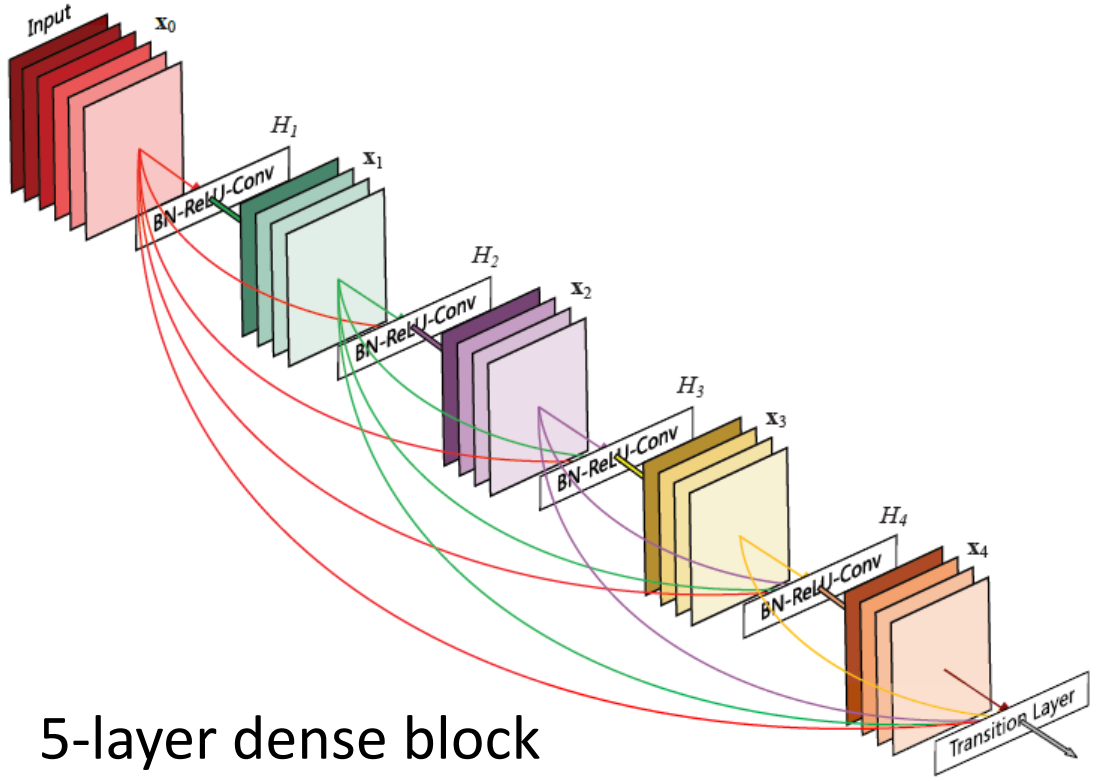
Densely Connected Neural Networks



Connects each layer to every other layer in a feed-forward fashion. Concatenation shortcuts, instead of additive.

traditional convolutional networks with L layers have L cnnections: densely connected network has $L(L+1)/2$ direct connections.

Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



5-layer dense block

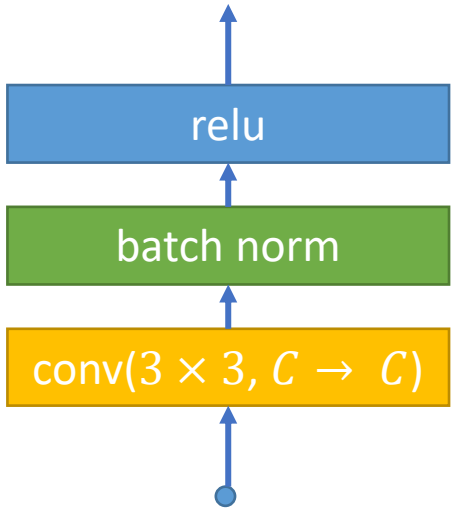
Huang et al, "Densely connected neural networks", CVPR 2017

Slide: Slightly modified from Johnson

MobileNets: Tiny Networks (for Mobile Devices)

Standard Convolution Block

Total cost: $9C^2HW$



$9C^2HW$

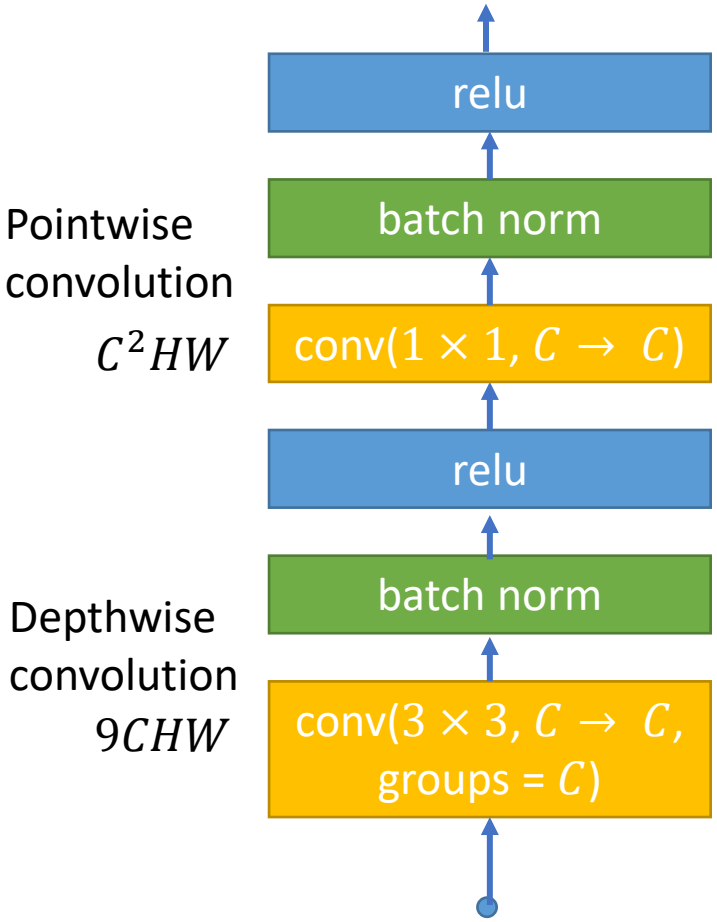
Speed up:

$= 9C^2 HW / (9C + C^2)HW$

$= 9 \text{ (as } C \to \infty)$

Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$



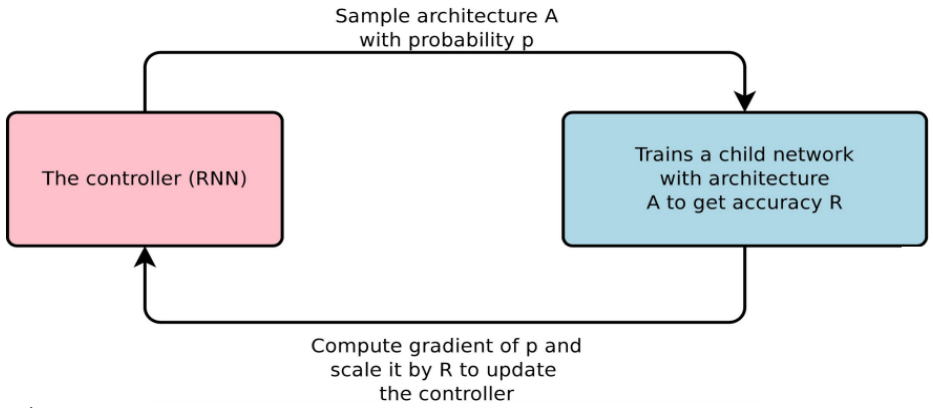
ShuffleNet:
Zhang et al, CVPR 2018
MobileNetV2:
Sandler et al, CVPR 2018
ShuffleNetV2:
Ma et al, ECCV 2018

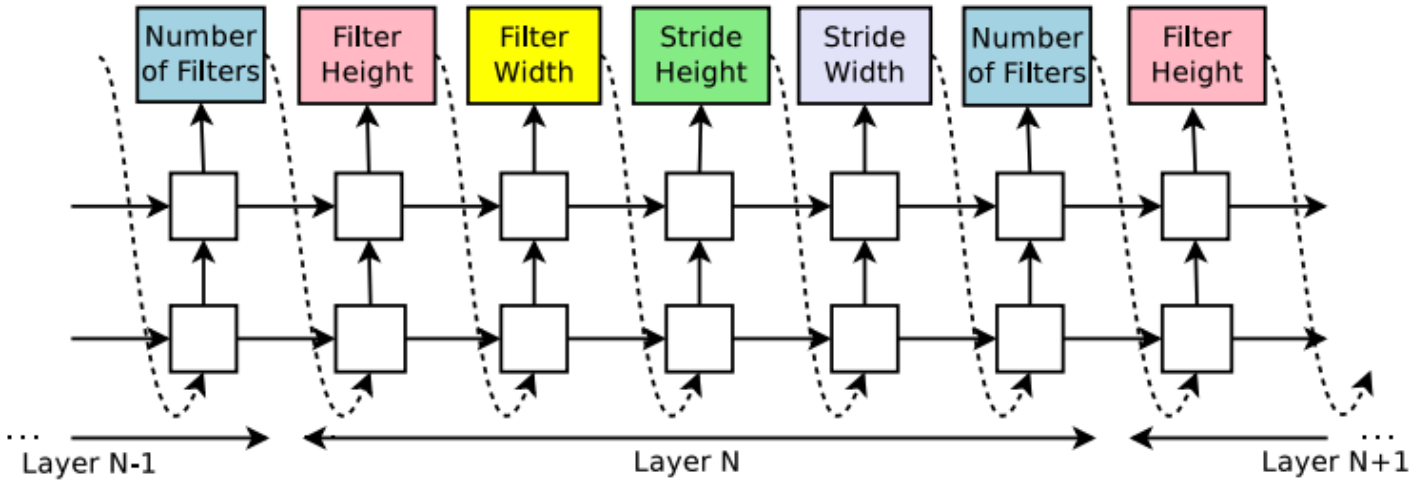
Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

Slide: Slightly modified from Johnson

Neural Architecture Search

Configuration string:
["Filter width: 5", "Filter height: 3", "Num filters: 24'





Designing neural network architectures is hard – let's automate it!
- One network (controller) outputs network architectures
- Sample child networks from controller and train them
- After training a batch of child networks, make a gradient step on controller network (Using policy gradient)
- Over time, controller learns to output good architectures!

800 GPUs concurrently for 28 days
12,800 child models

Zoph and Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Slide: Modified from Johnson

CNN Architectures Summary

- Early work (AlexNet -> ZFNet -> VGG) shows that **bigger networks work better**
- GoogLeNet one of the first to focus on **efficiency** (aggressive stem, 1x1 bottleneck
- convolutions, global avg pool instead of FC layers)
- ResNet showed us how to train extremely deep networks – limited only by GPU
- memory! Started to show diminishing returns as networks got bigger
- After ResNet: Efficient networks became central: how can we improve the accuracy
- without increasing the complexity?
- Lots of **tiny networks** aimed at mobile devices: MobileNet, ShuffleNet, etc.
- Neural Architecture Search promises to automate architecture design

Slide: Slightly modified from Johnson

What Architecture to Use

- Don’t be a hero. For most problems you should use an off-the-shelf architecture; don’t try to design your own!
- If you just care about accuracy, ResNet-50 or ResNet-101 are great choices
- If you want an efficient network (real-time, run on mobile, etc.) try MobileNets and ShuffleNets.

Slide: Johnson