



CRIPAC

智能感知与计算研究中心
Center for Research on Intelligent
Perception and Computing



中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2019

“Deep Learning Lecture”

Lecture 4: Convolutional Neural Network

Liang Wang

Center for Research on Intelligent Perception and Computing (CRIPAC)

National Laboratory of Pattern Recognition (NLPR)

Institute of Automation, Chinese Academy of Science (CASIA)

Outline

1 Course Review

2 Basic Operations

3 Major Architectures

4 CNN for Image Classification

Review: Feedforward Neural Network

- Goal: Approximate some unknown ideal function

$$f^* : \mathcal{X} \rightarrow \mathcal{Y}$$

- Ideal classifier: $y = f^*(x)$ with x and category y
- Feedforward Network: Define parametric mapping

$$y = f(\mathbf{x}, \theta)$$

- Learn parameters θ to get a good approximation to f^* from available sample
- Naming: Information flow in function evaluation begins at input, flows through intermediate computations (that define the function), to produce the category
- No feedback connections (Recurrent Networks!)

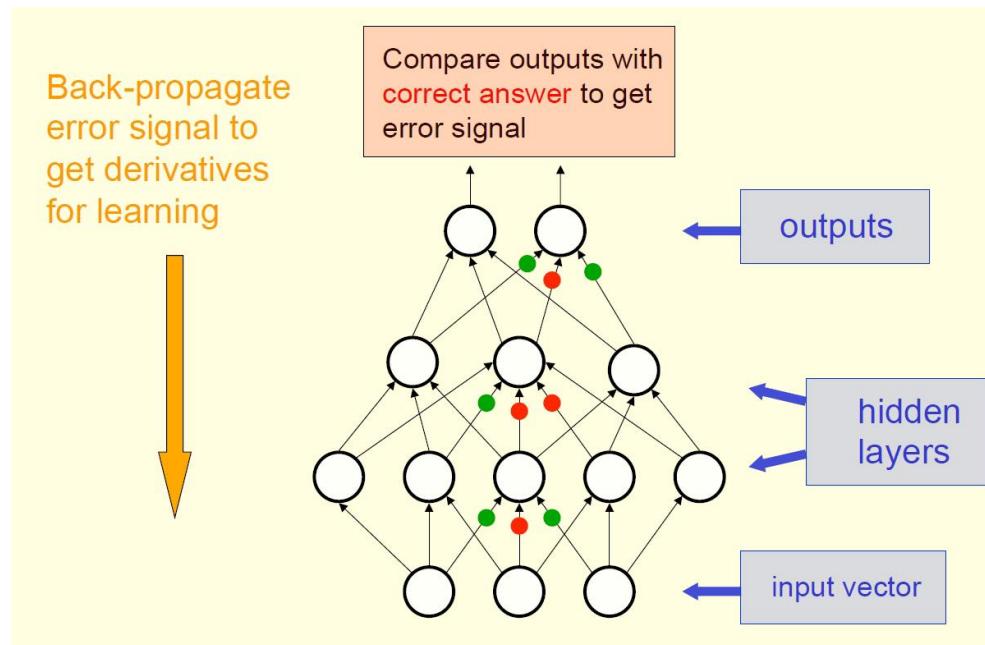
Review: Basic Components

- Cost Functions
- Output Units
 - Linear Units
 - Sigmoid Units
 - Softmax Units
 - ...
- Hidden Units
 - Rectified Linear Units
 - Sigmoid Units
 - Tanh Units
 - ...

Review: Basic Components

- In Feedforward Networks don't use Sigmoid
- When a sigmoidal function must be used, use tanh
- Use ReLU by default, but be careful with learning rates
- Try other generalized ReLUs and Maxout for possible improvement

Review: Backpropagation



- **Feedforward Propagation:** Accept input x , pass through intermediate stages and obtain output \hat{y}
- **During Training:** Use \hat{y} to compute a scalar cost $J(\theta)$
- Backpropagation allows information to flow backwards from cost to compute the gradient

Figure: G. E. Hinton

Review: Backpropagation

- Stochastic learning
 - Estimate of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
 - Faster than batch learning, especially when training data has redundancy
 - Noise often results in better solutions
 - The weights fluctuate and it may not fully converge to a local minimum
- Batch learning
 - Conditions of convergence are well understood
 - Some acceleration techniques only operate in batch learning
 - Theoretical analysis of the weight dynamics and convergence rates are simpler

Outline

1 Course Review

2 Basic Operations

3 Major Architectures

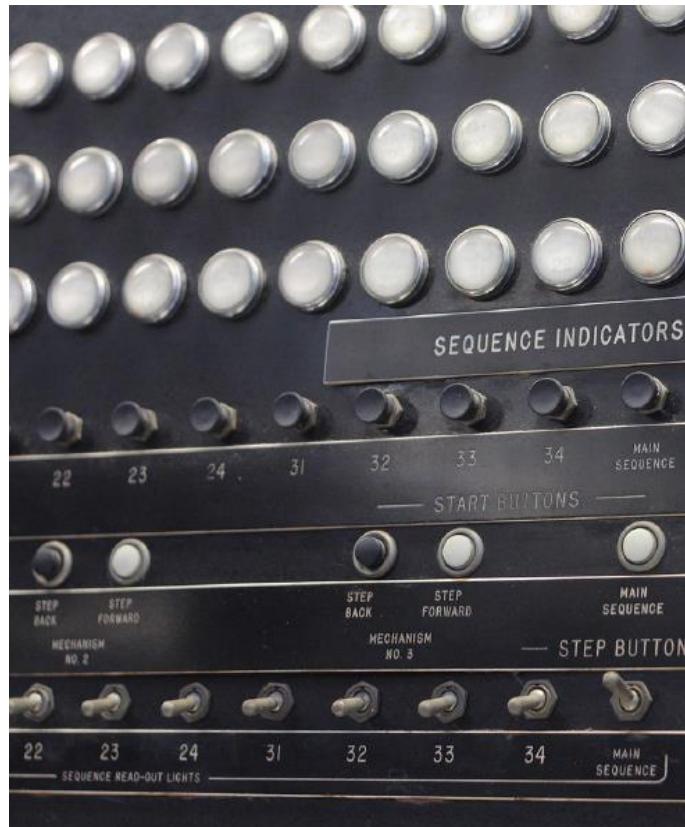
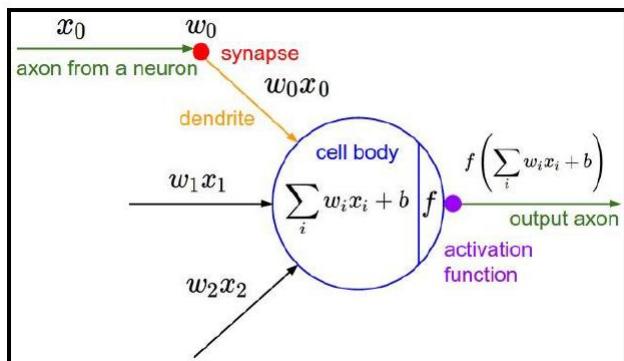
4 CNN for Image Classification

Historical Background

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

Recognized letters of the alphabet



This image by Rocky Acosta is licensed under CC-BY 3.0

Frank Rosenblatt, ~1957: Perceptron

Historical Background

Hubel & Wiesel

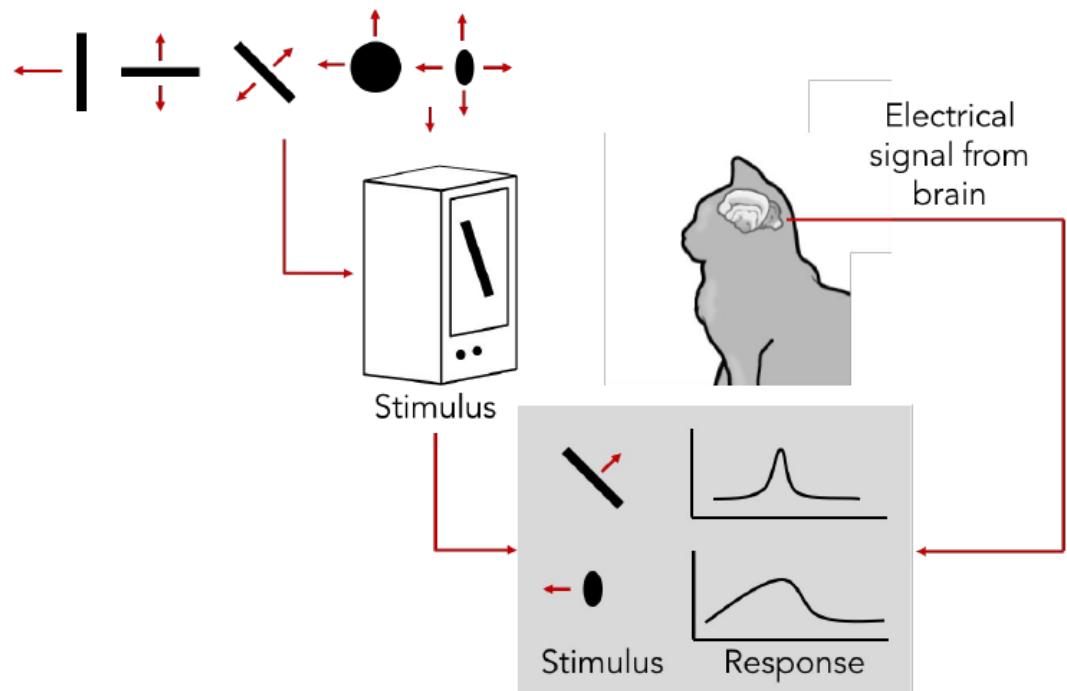
1959

Receptive fields of single neurons in the cat's striate cortex

1962

Receptive fields, binocular interaction and functional architecture in the cat's visual cortex

1968...



What is Convolutional Neural Network?

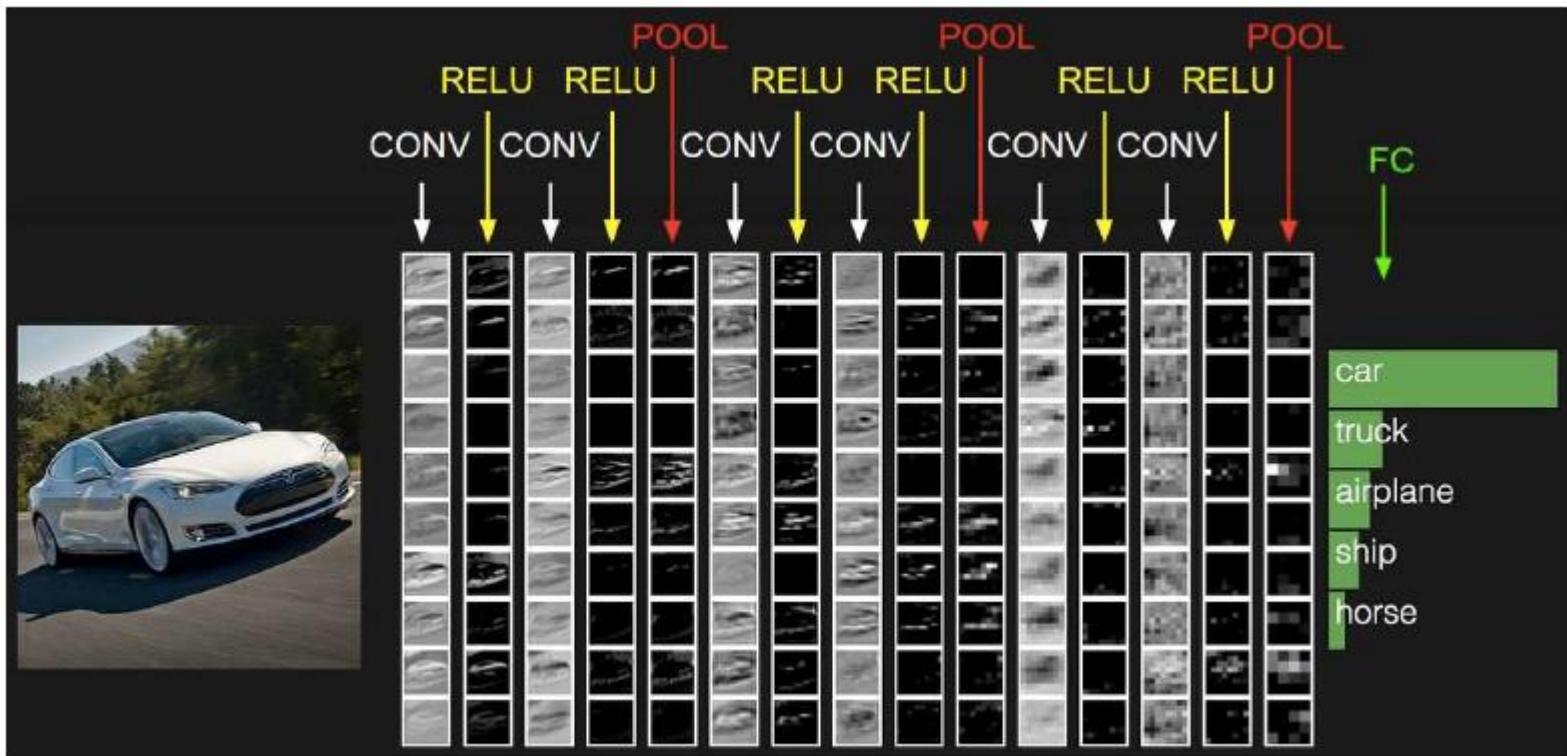


Figure: Andrej Karpathy

- **Convolution Layers**
- **Pooling Layers**
- **Fully-Connected Layers**
- **Non-linear Layers, e.g., ReLU**
- **Other Layers, e.g., RNN Layer, Your Customized Layers ...**

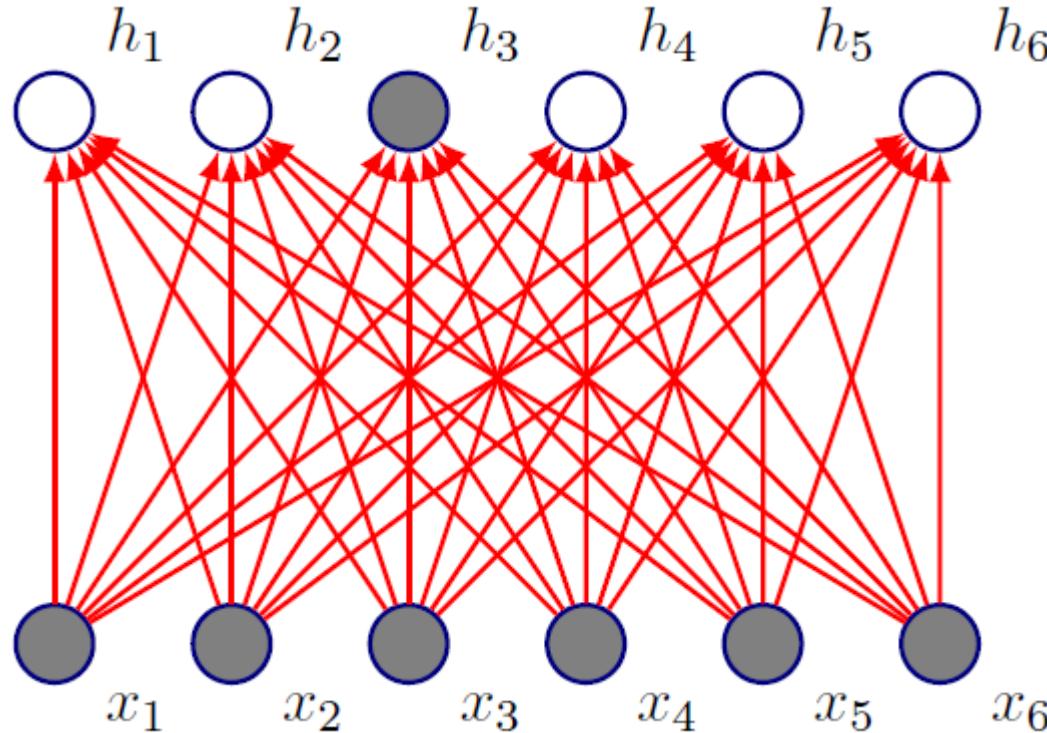


Why do Convolutional Neural
Network (CNNs) make sense ?

Convolutions: Motivation

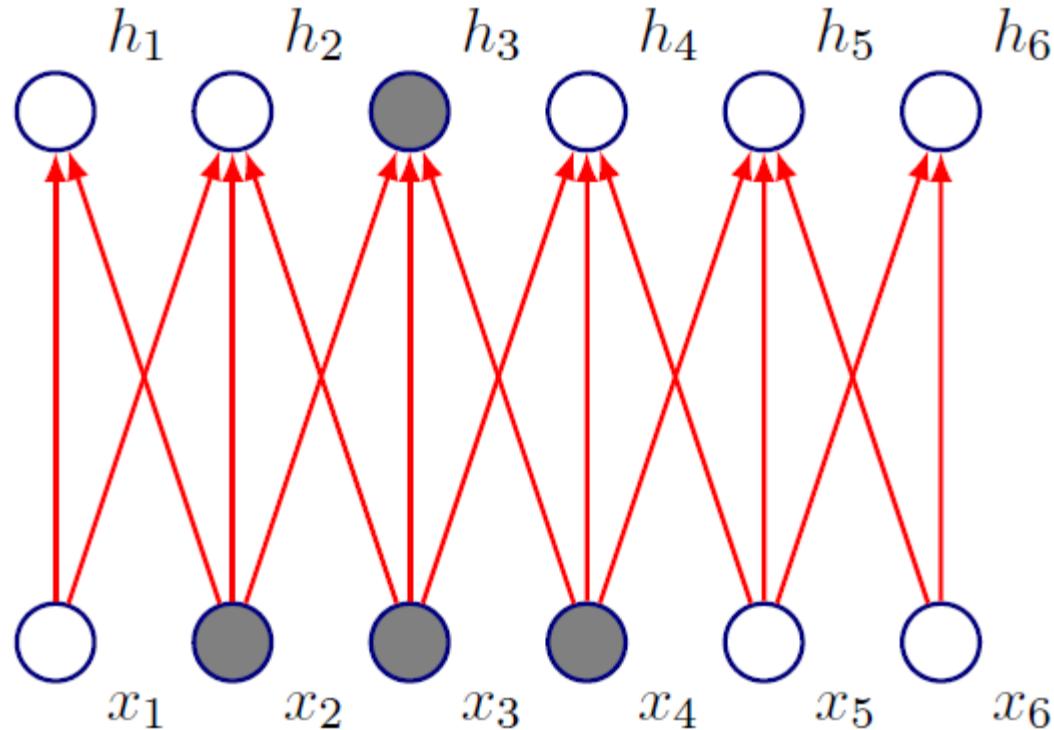
- **Convolution leverages four ideas that can help ML systems:**
 - Sparse interactions
 - Parameter sharing
 - Equivariant representations
 - Ability to work with inputs of variable size
- **Sparse Interactions**
 - Plain Vanilla NN ($y \in \mathbb{R}^n, x \in \mathbb{R}^m$): Need matrix multiplication $y = Wx$ to compute activations for each layer (every output interacts with every input)
 - Convolutional networks have sparse interactions by making kernel smaller than input
 - => need to store fewer parameters, computing output needs fewer operations ($O(m \times n)$ versus $O(k \times n)$)

Motivation: Sparse Connectivity



Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity

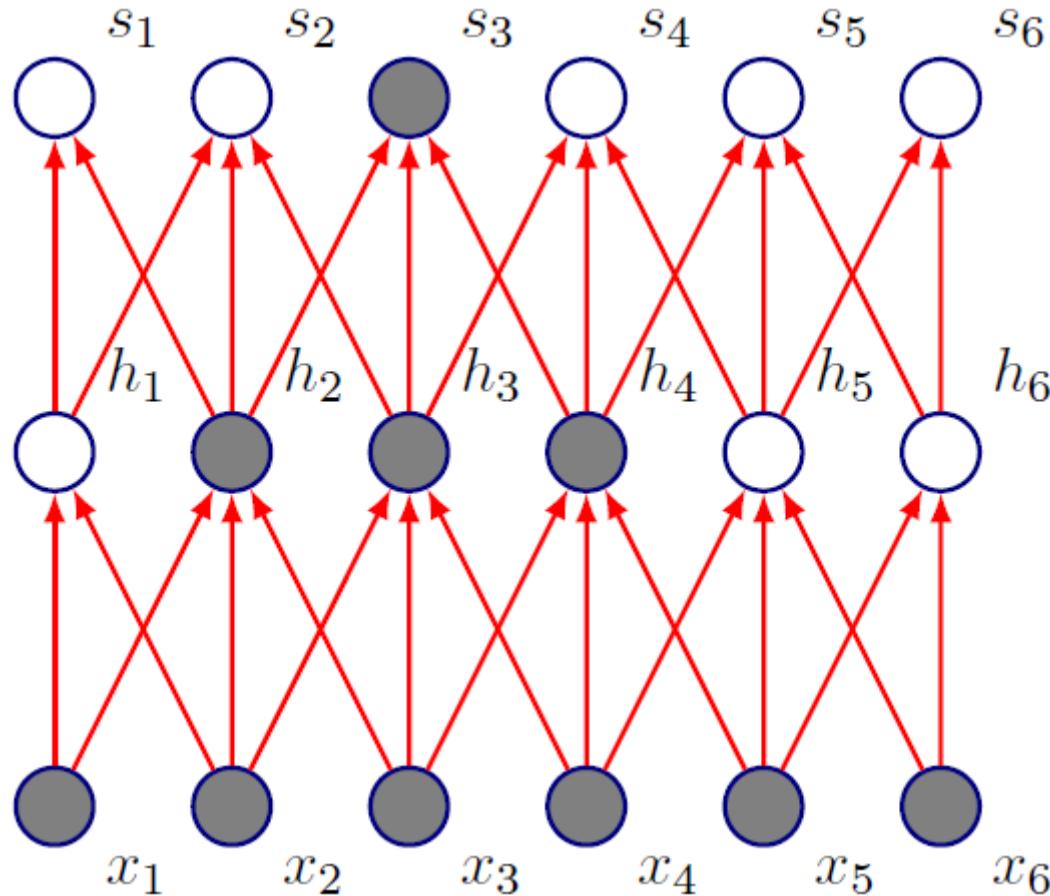
Motivation: Sparse Connectivity



Kernel of size 3, moved with stride of 1

h_3 only depends on x_2, x_3, x_4

Motivation: Sparse Connectivity



Connections in CNNs are sparse, but units in deeper layers are connected to all of the input (larger receptive field sizes)

Motivation: Parameter Sharing

- Plain vanilla NN: Each element of \mathbf{W} is used exactly once to compute output of a layer
- In convolutional neural networks, parameters are tied: weight applied to one input is tied to value of a weight applied elsewhere
- Same kernel is used throughout the image, so instead learning a parameter for each location, only a set of parameters is learnt
- Forward propagation remains unchanged $O(k \times n)$
- Storage improves dramatically as $k \ll m, n$

Motivation: Equivariance

Let's first formally define convolution:

$$s(t) = (x * w)(t) = \int x(a)w(t - a)da$$

In Convolutional Network terminology x is referred to as **input**, w as the **kernel** and s as the **feature map**

Discrete Convolution:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

Convolution is commutative, thus:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

Aside

- The latter is usually more straightforward to implement in ML libraries (less variation in range of valid values of m and n)
- Neither are usually used in practice in Neural Networks
- Libraries implement ***Cross Correlation***, same as convolution, but without flipping the kernel

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

Motivation: Equivariance

- Equivariance: f is equivariant to g if $f(g(x)) = g(f(x))$
- The form of parameter sharing used by CNNs causes each layer to be equivariant to translation
- That is, if g is any function that translates the input, the convolution function is equivariant to g

Motivation: Equivariance

- Implication: While processing time series data, convolution produces a timeline that shows when different features appeared (if an event is shifted in time in the input, the same representation will appear in the output)
- Images: If we move an object in the image, its representation will move the same amount in the output
- This property is useful when we know some local function is useful everywhere (e.g. edge detectors)
- Convolution is not equivariant to other operations such as change in scale or rotation

Pooling: Motivation

- Pooling helps the representation become slightly invariant to small translations of the input
- Reminder: Invariance: $f(g(x)) = f(x)$
- If input is translated by small amount: values of most pooled outputs don't change

Pooling: Invariance

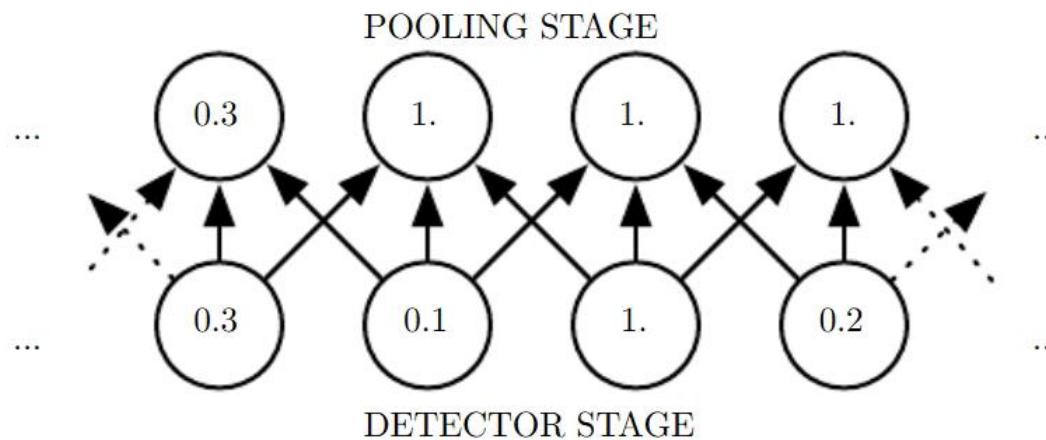
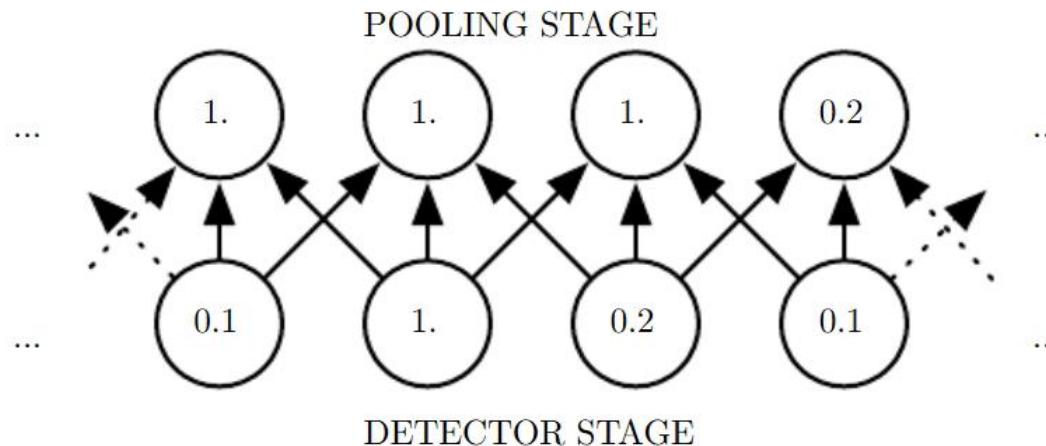
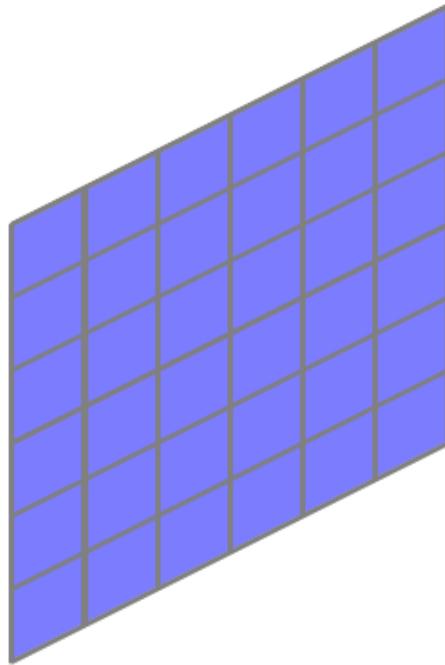


Figure: Goodfellow et al.

Pooling

- Invariance to local translation can be useful if we care more about whether a certain **feature is present rather than exactly where it is**
- Pooling over spatial regions produces invariance to translation, what if we pool over separately parameterized convolutions?
 - Features can learn which transformations to become invariant to (Example: Maxout Networks, Goodfellow et al 2013)
- **One more advantage:** Since pooling is used for downsampling, it can be used to handle inputs of varying sizes

Convolution

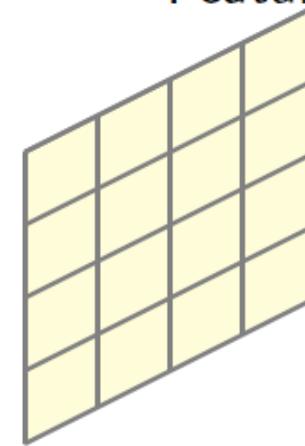


Grayscale Image

Kernel

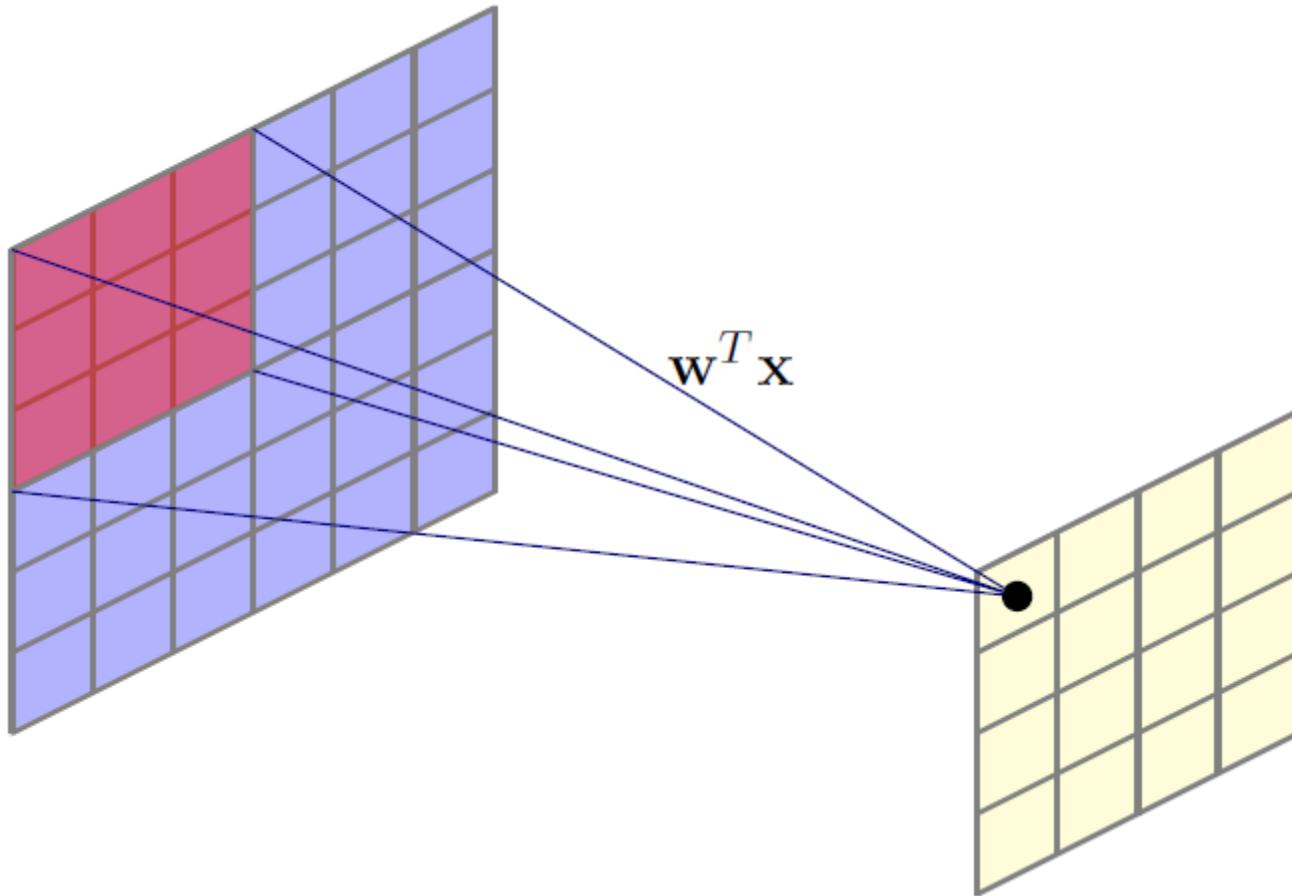
w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

Feature Map

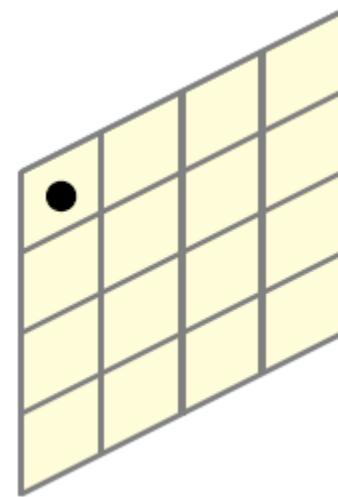
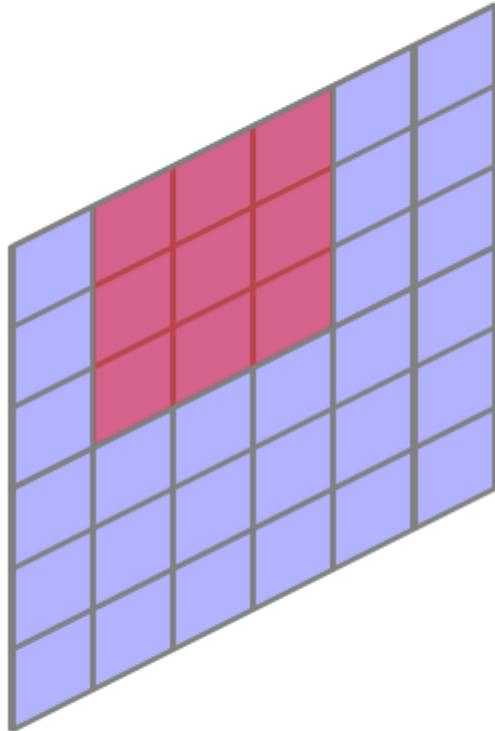


Convolve image with kernel having weights w (learned by backpropagation)

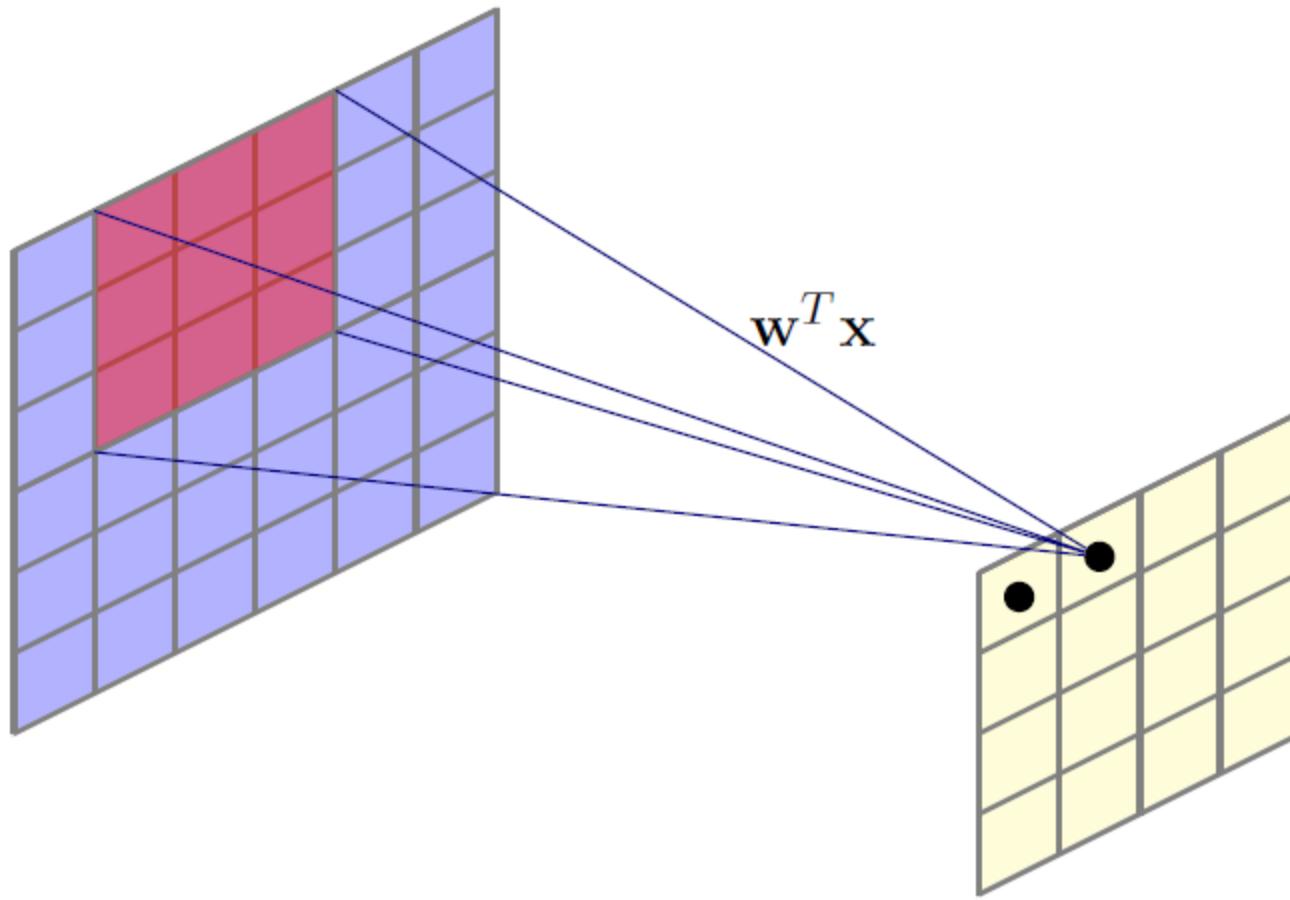
Convolution



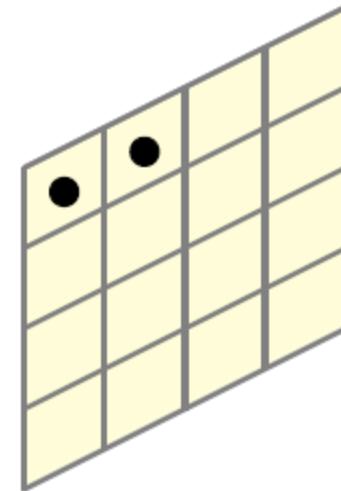
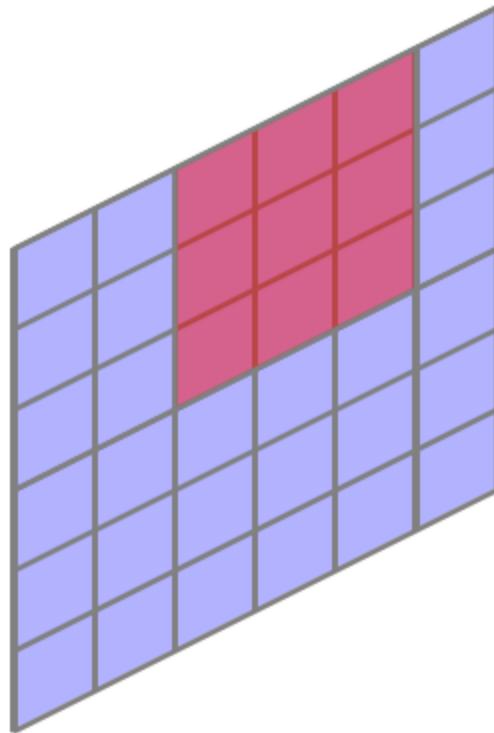
Convolution



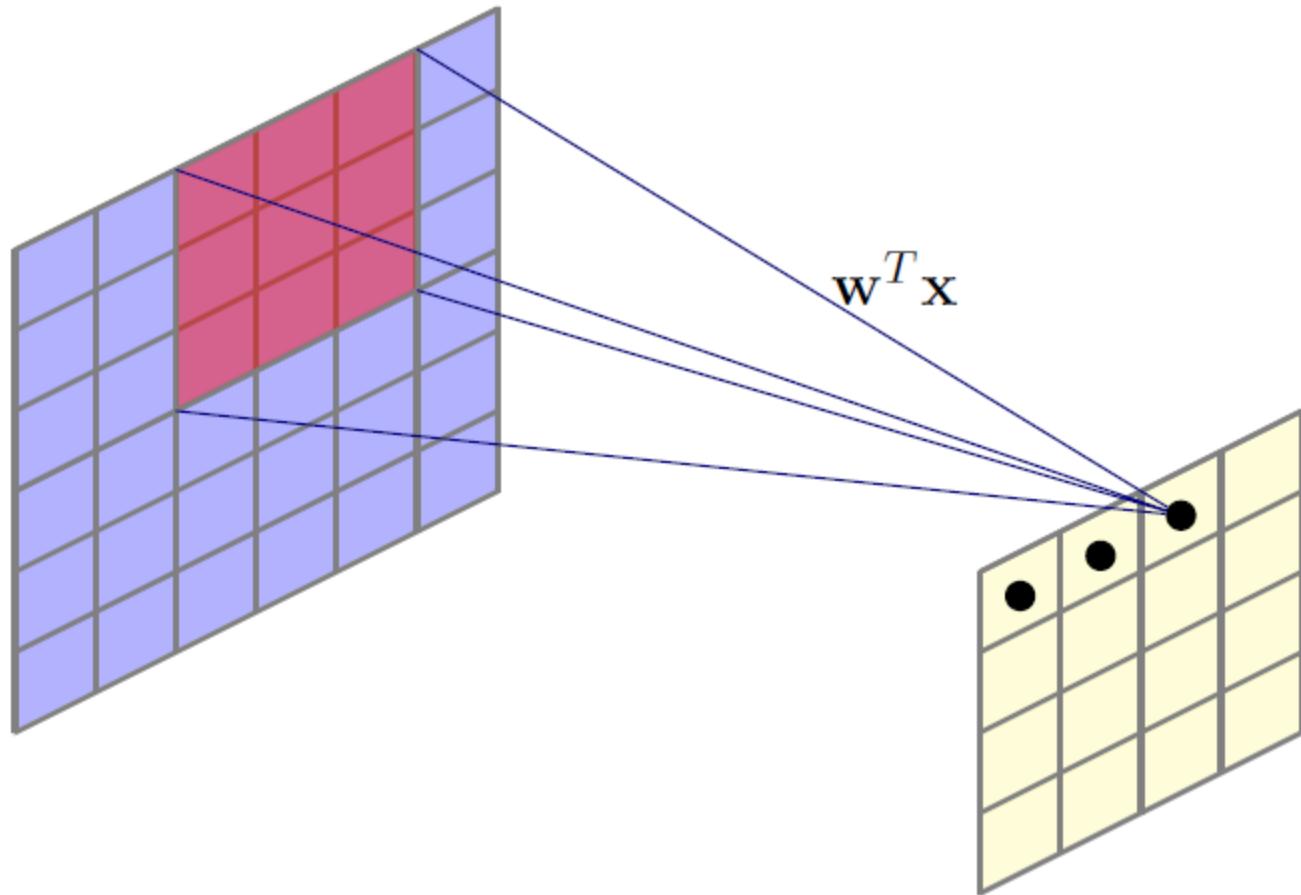
Convolution



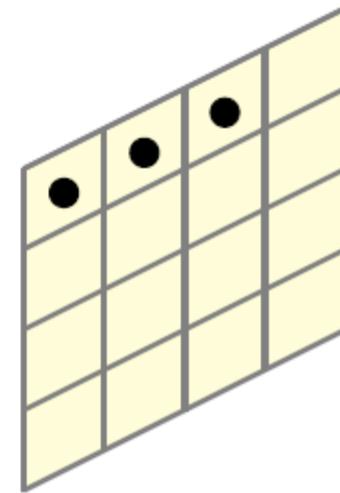
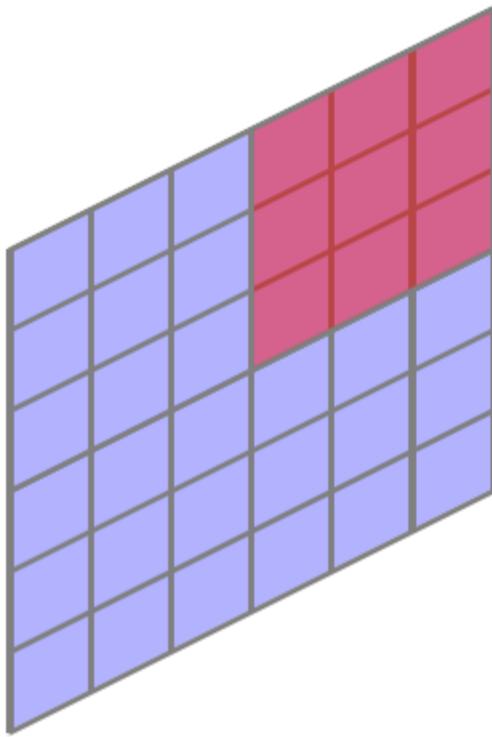
Convolution



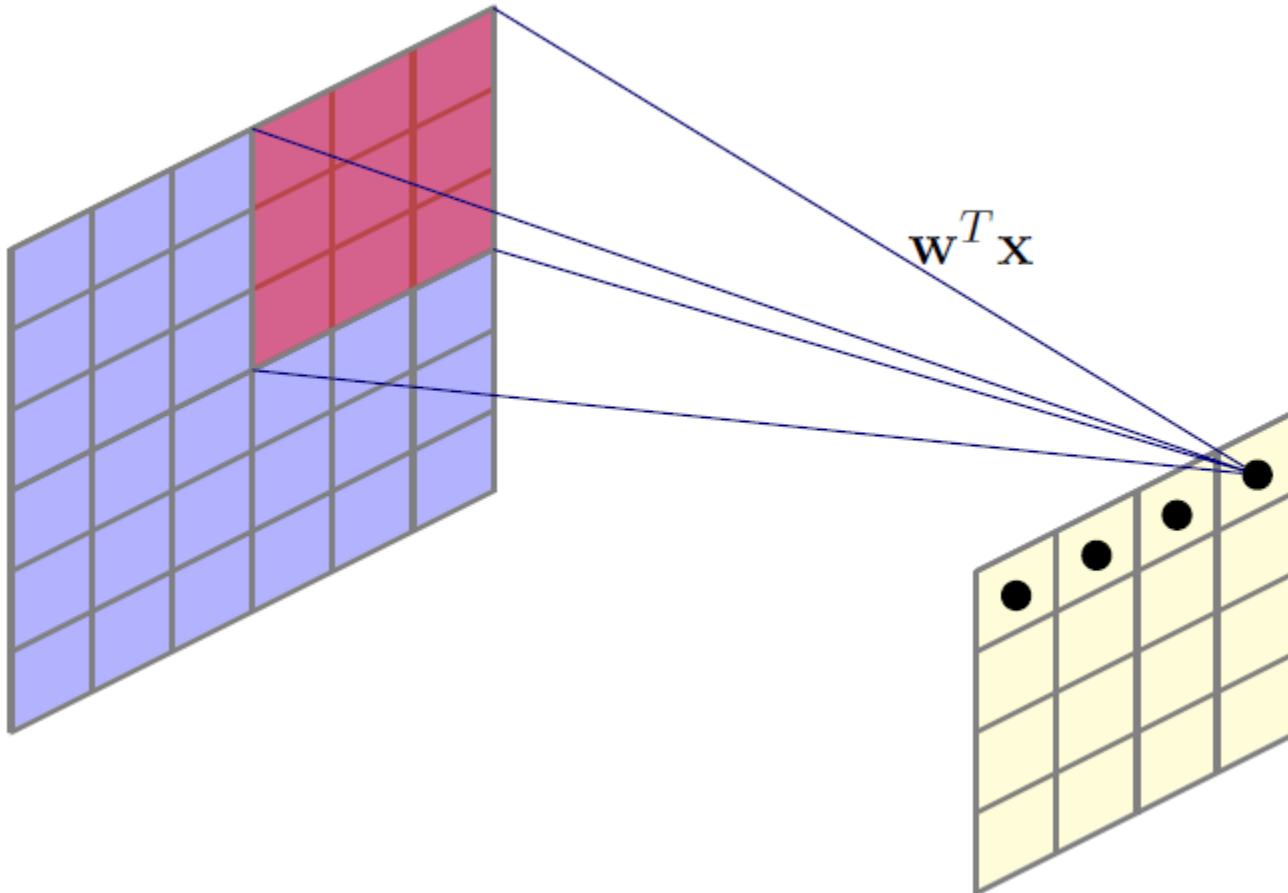
Convolution



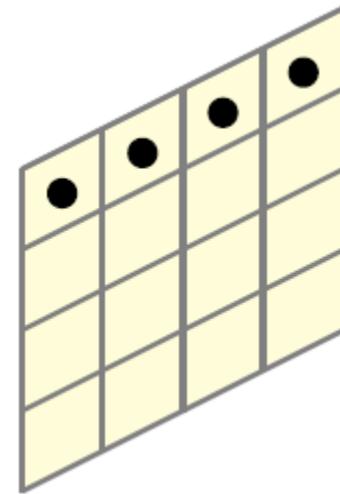
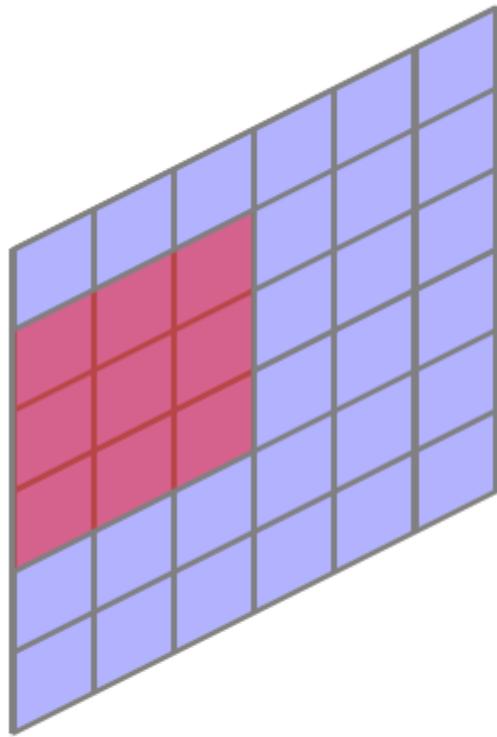
Convolution



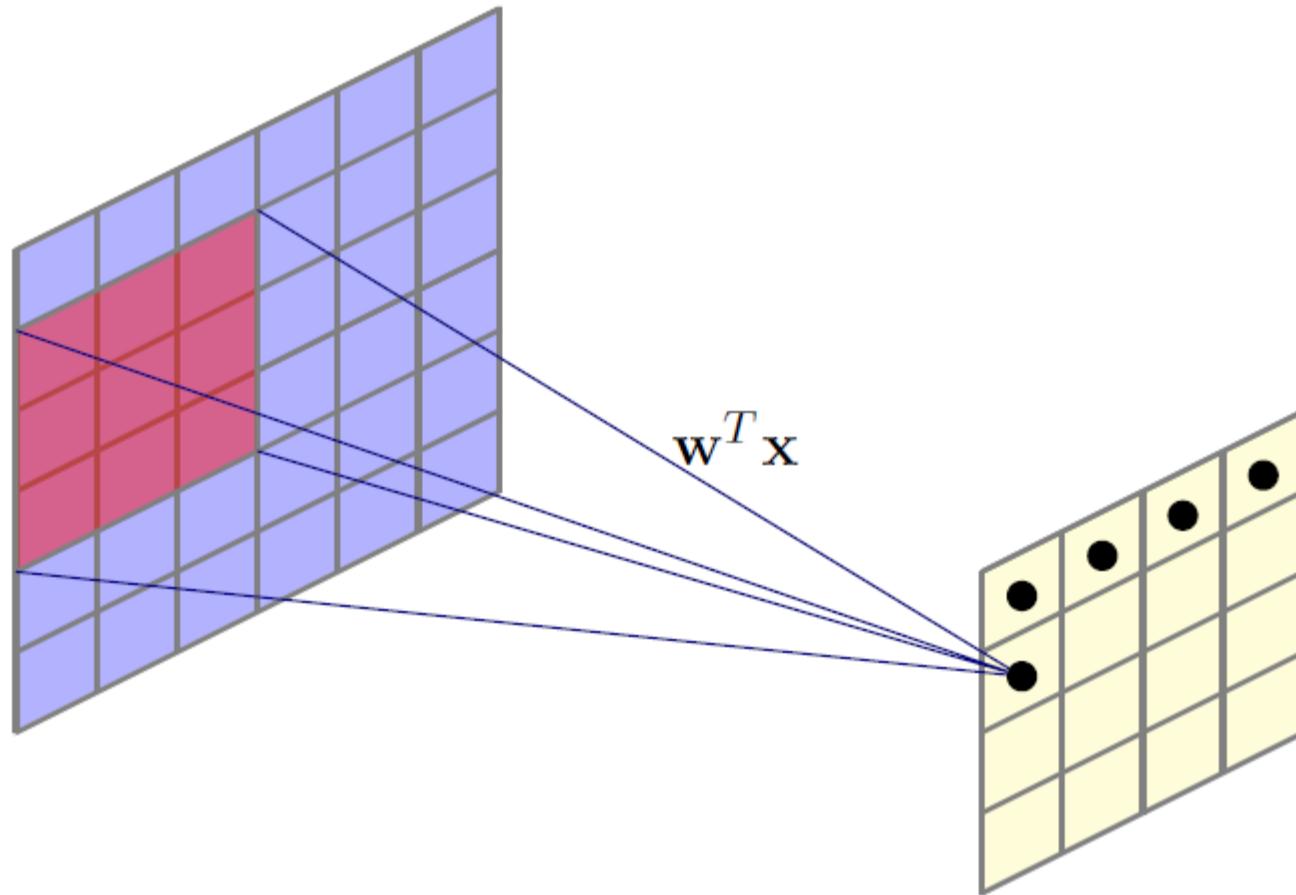
Convolution



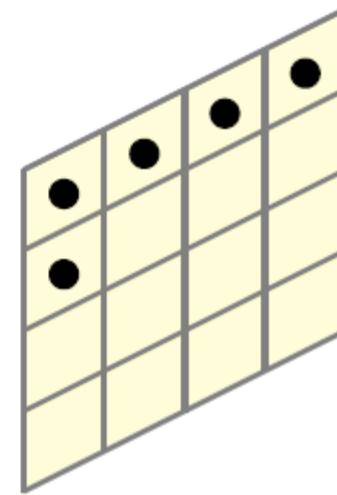
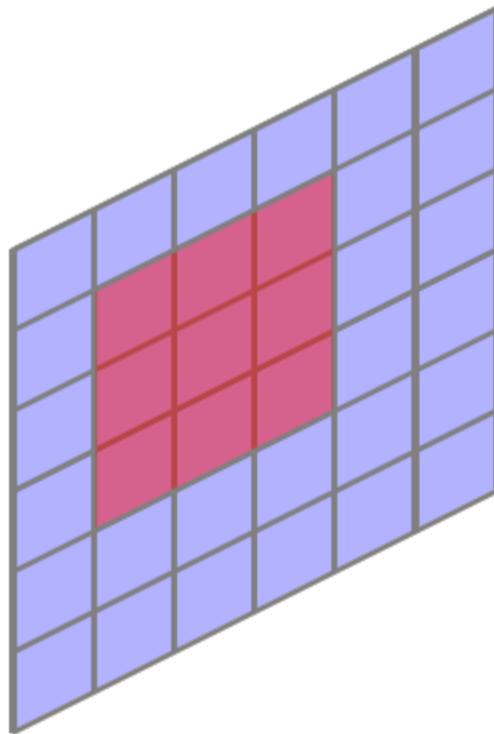
Convolution



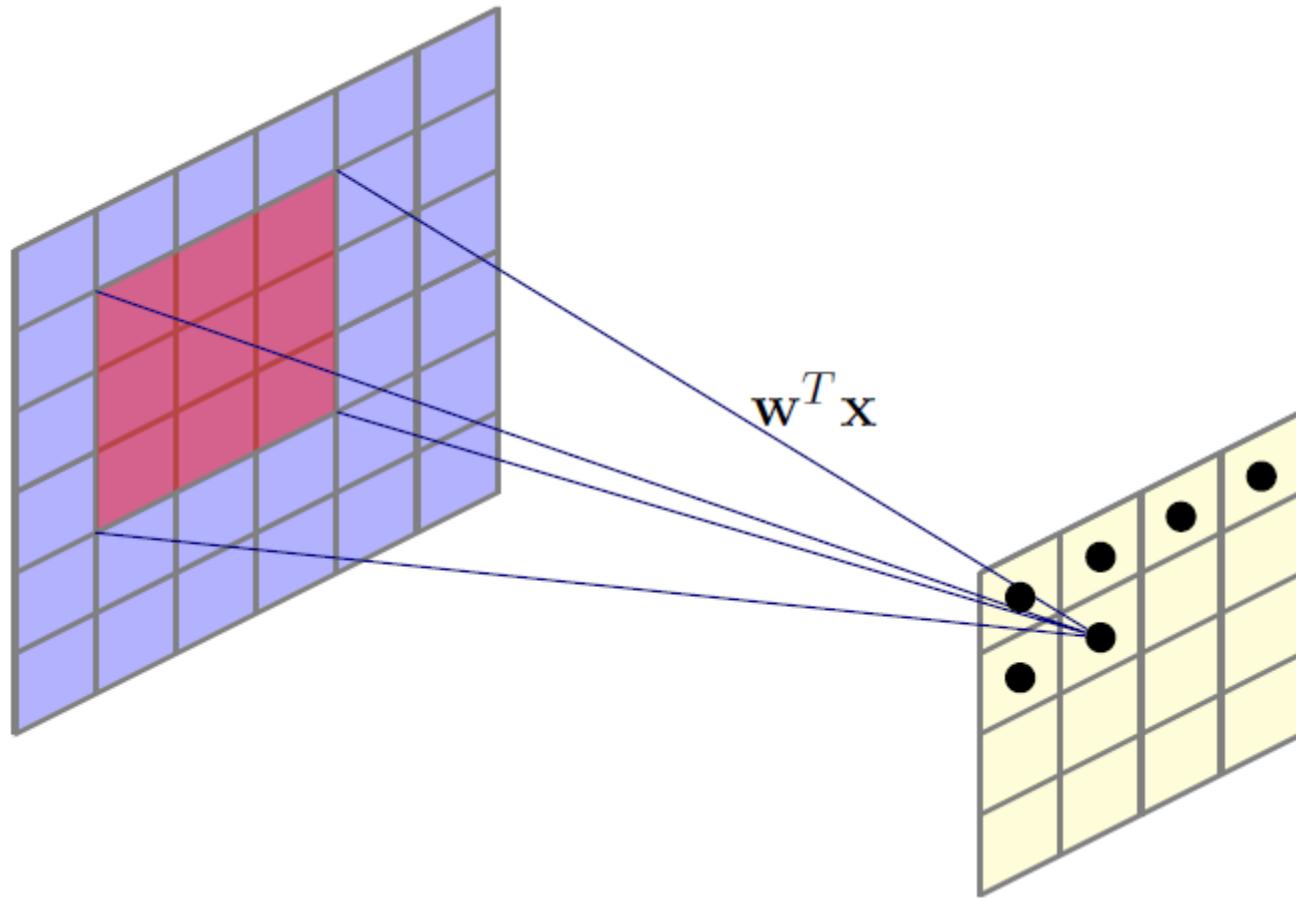
Convolution



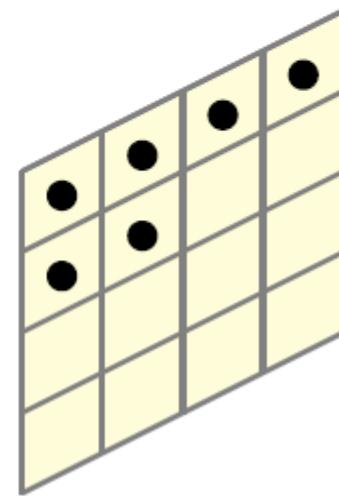
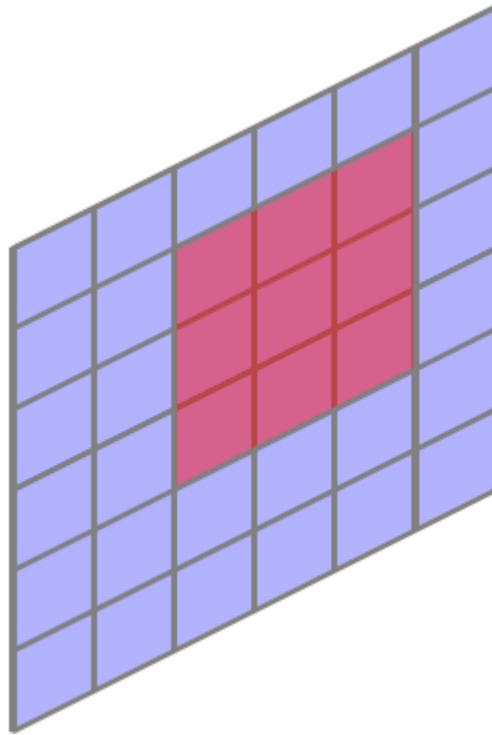
Convolution



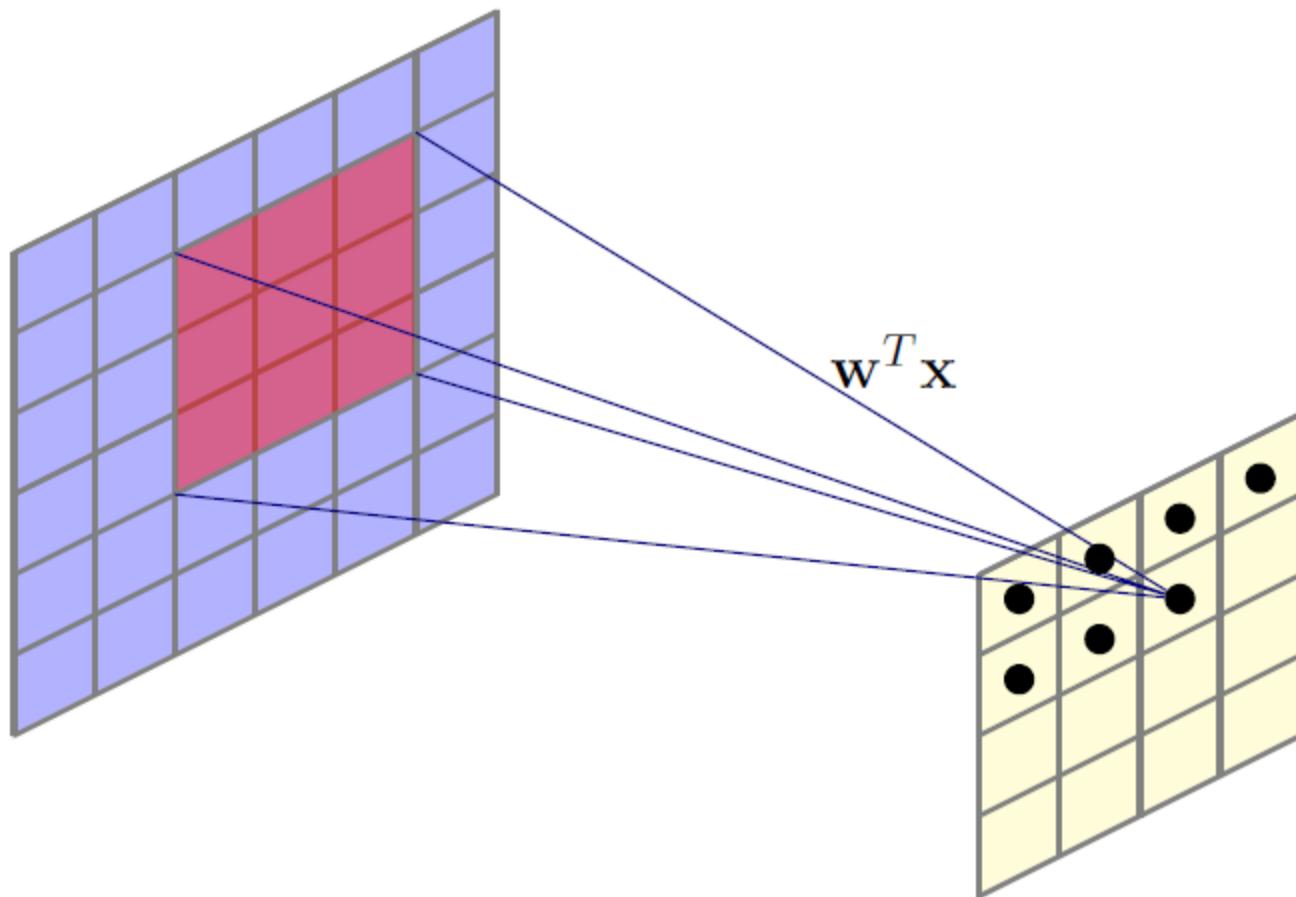
Convolution



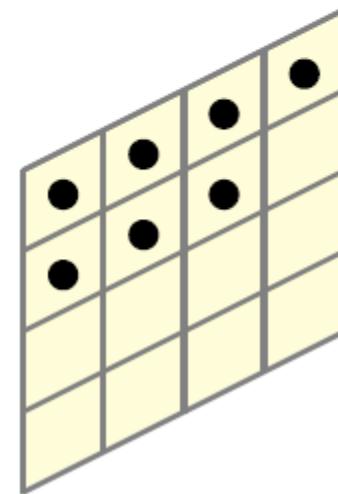
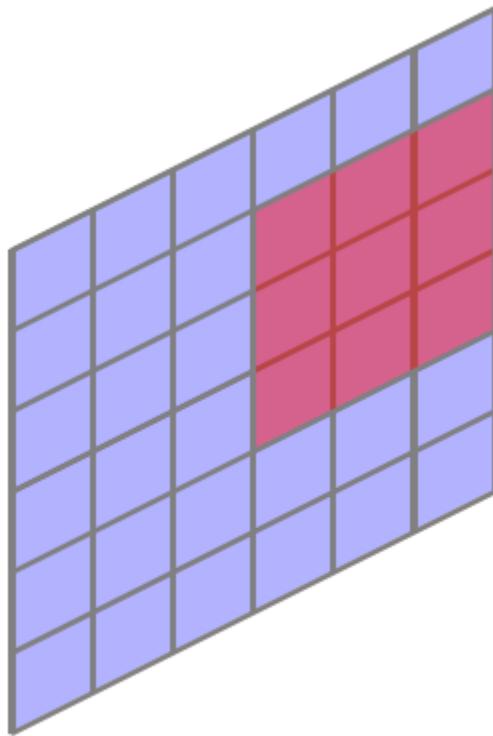
Convolution



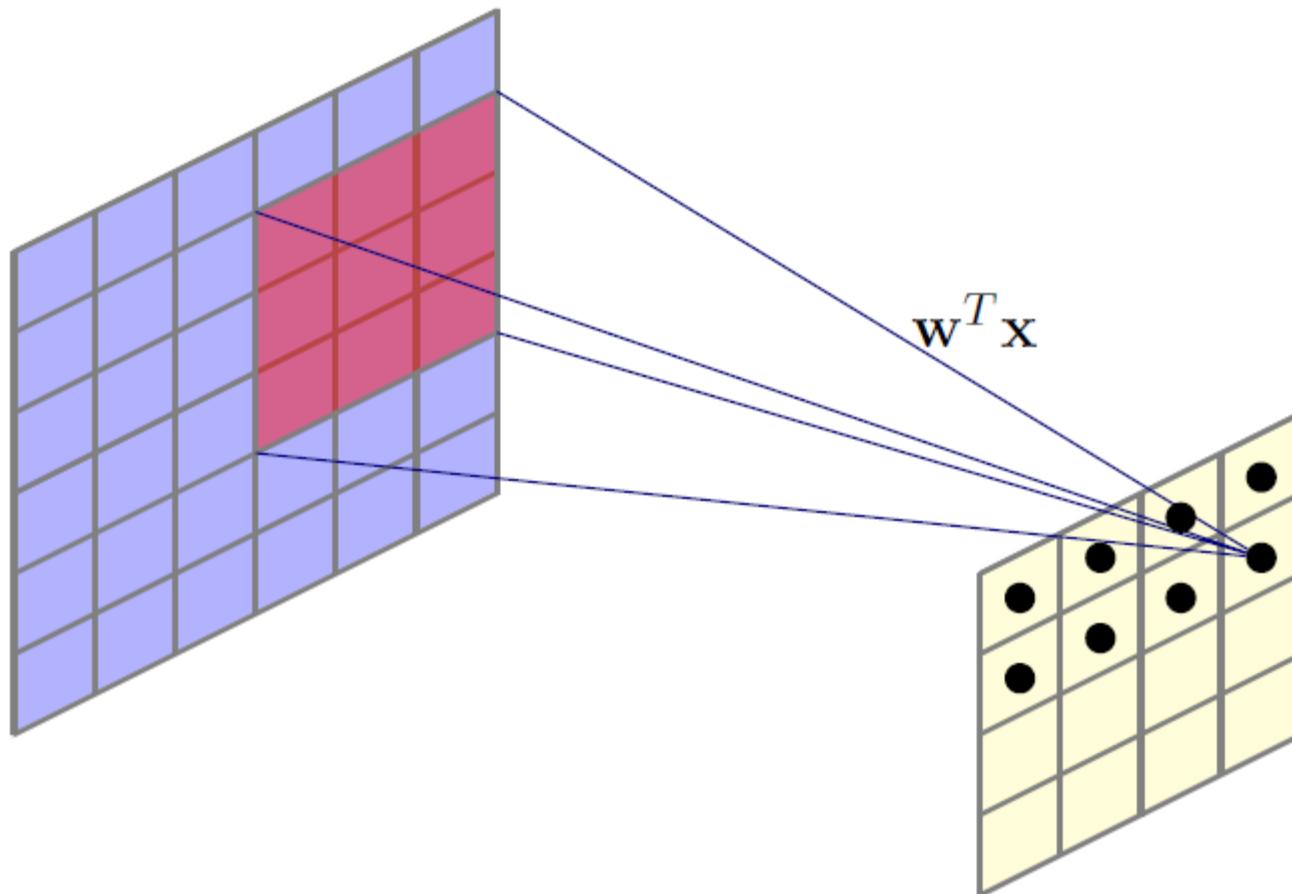
Convolution



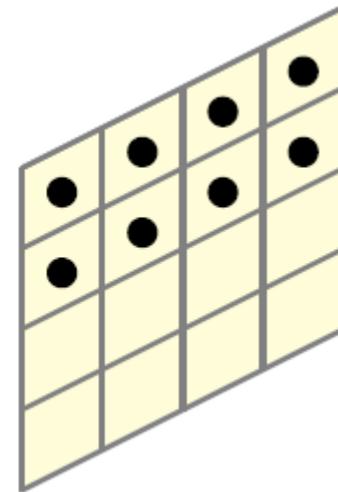
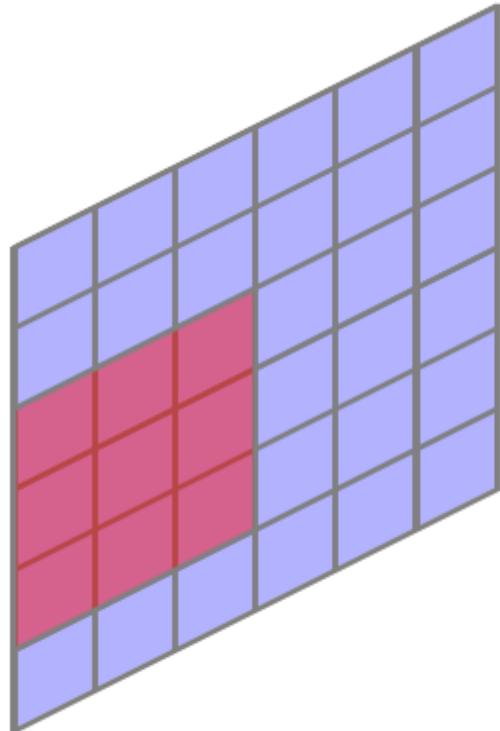
Convolution



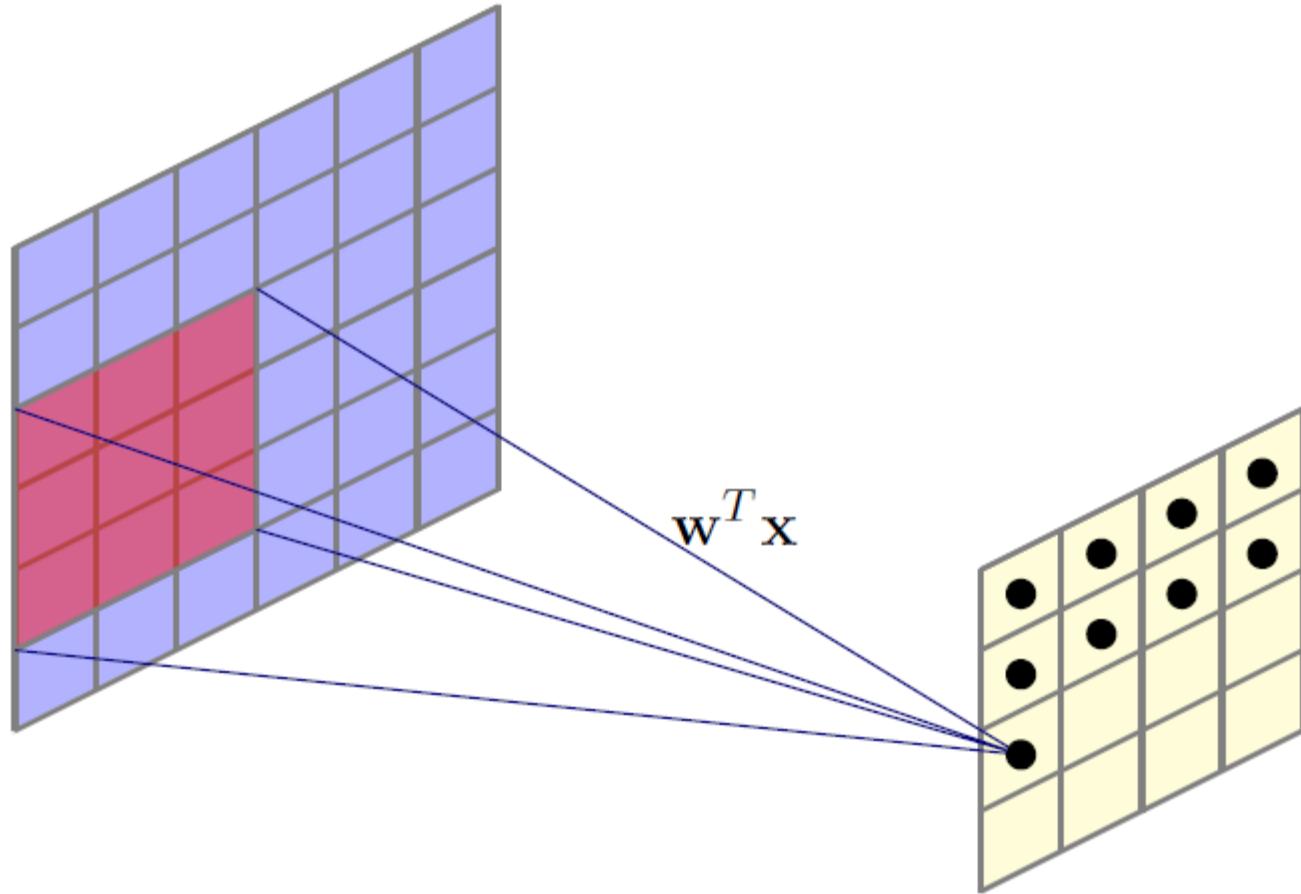
Convolution



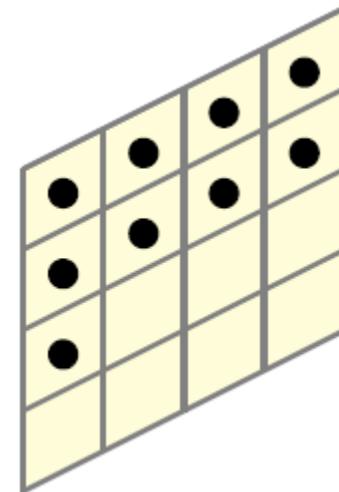
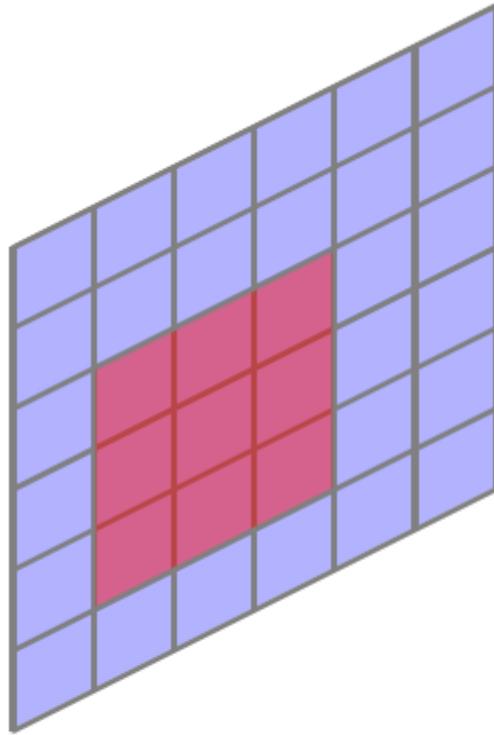
Convolution



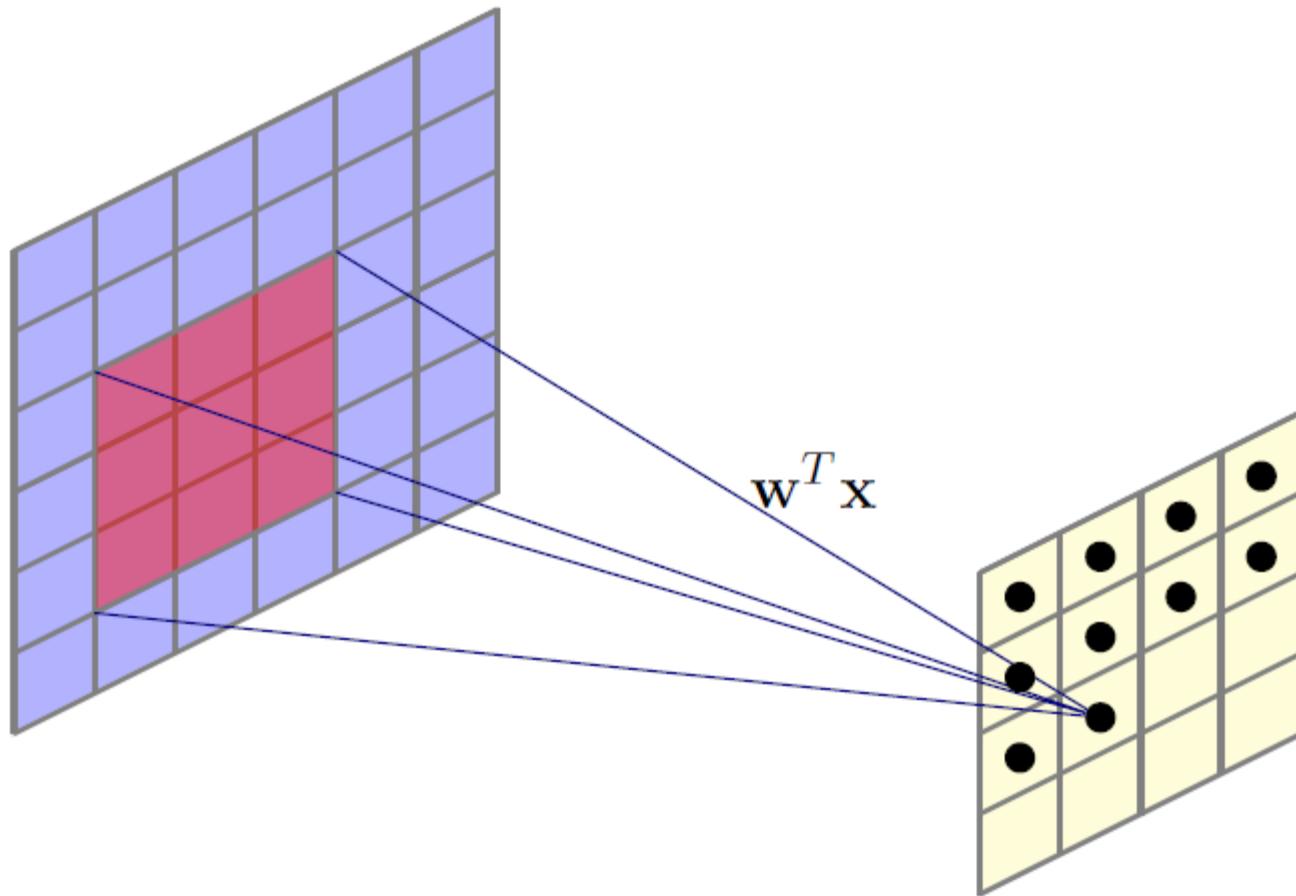
Convolution



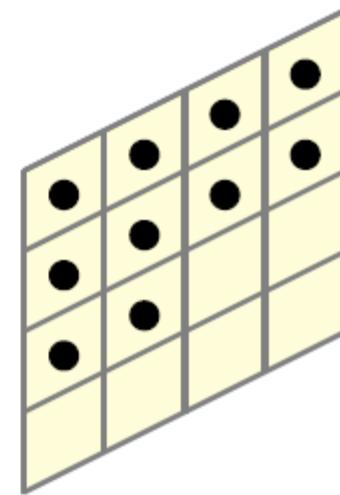
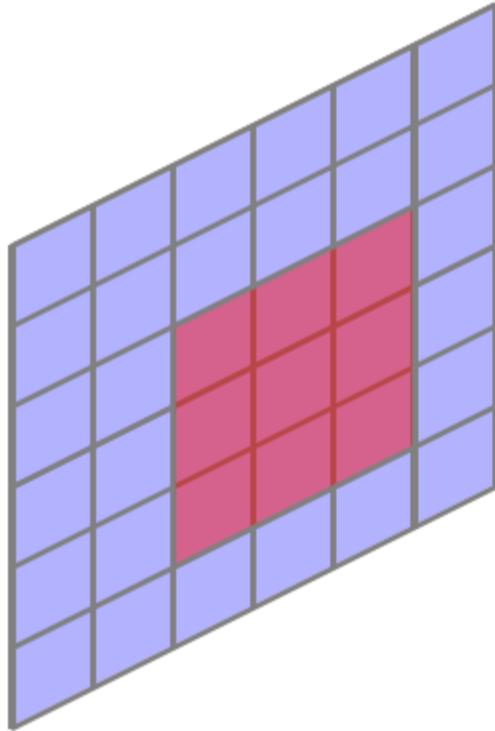
Convolution



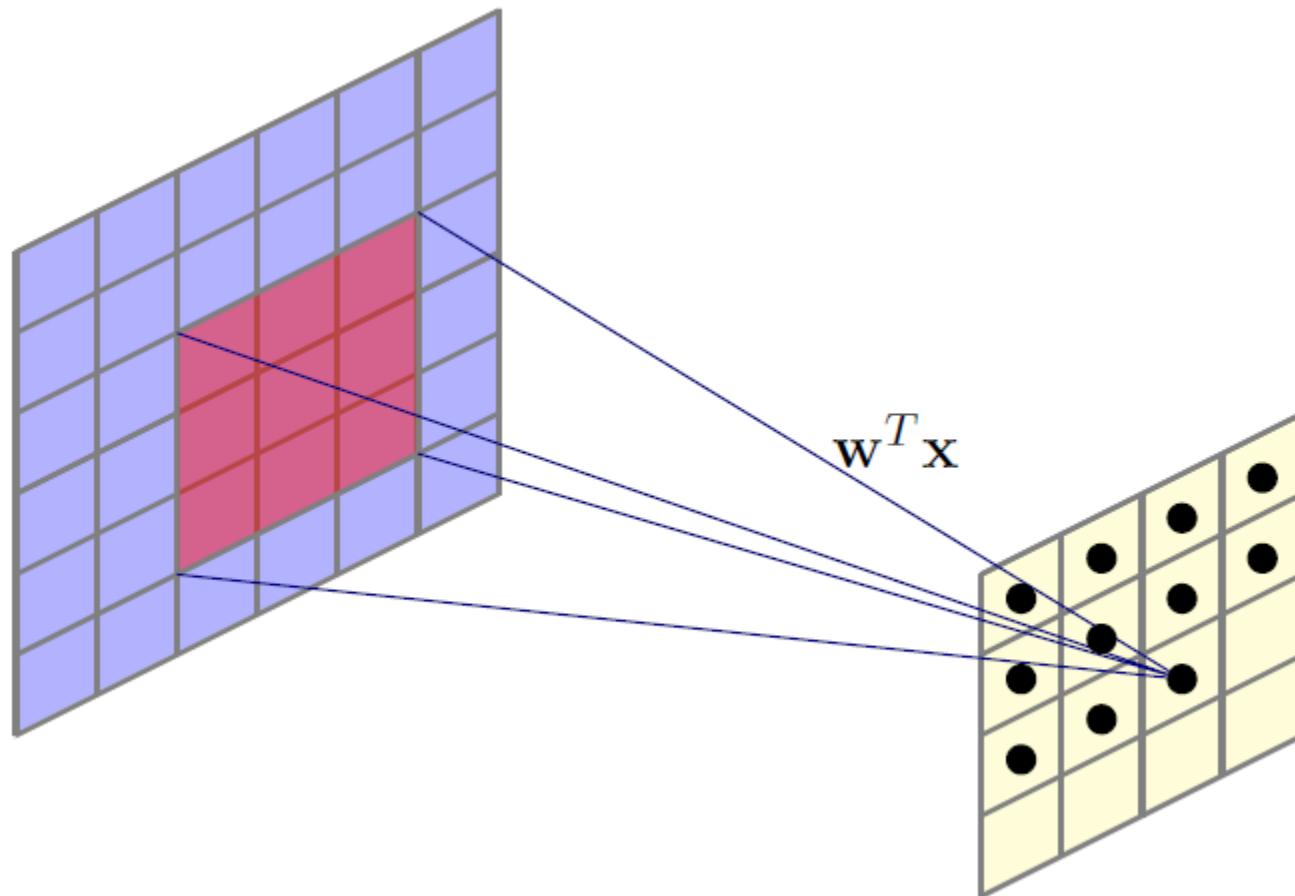
Convolution



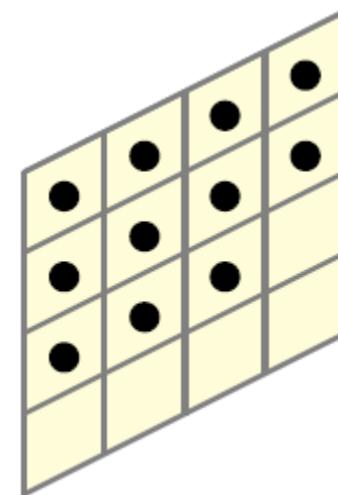
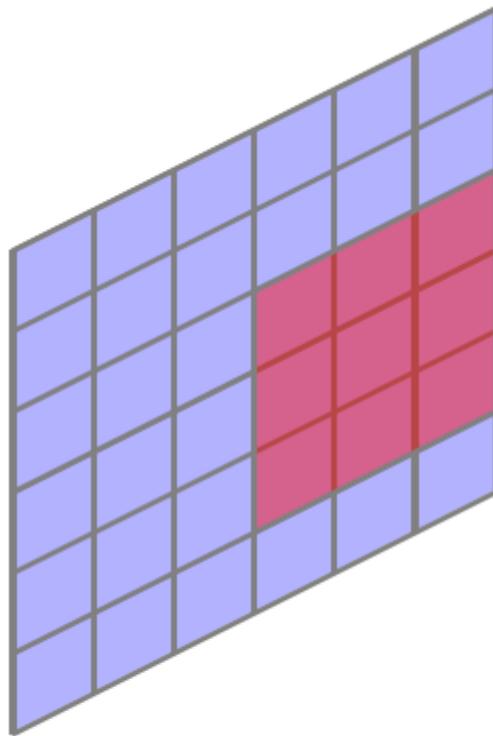
Convolution



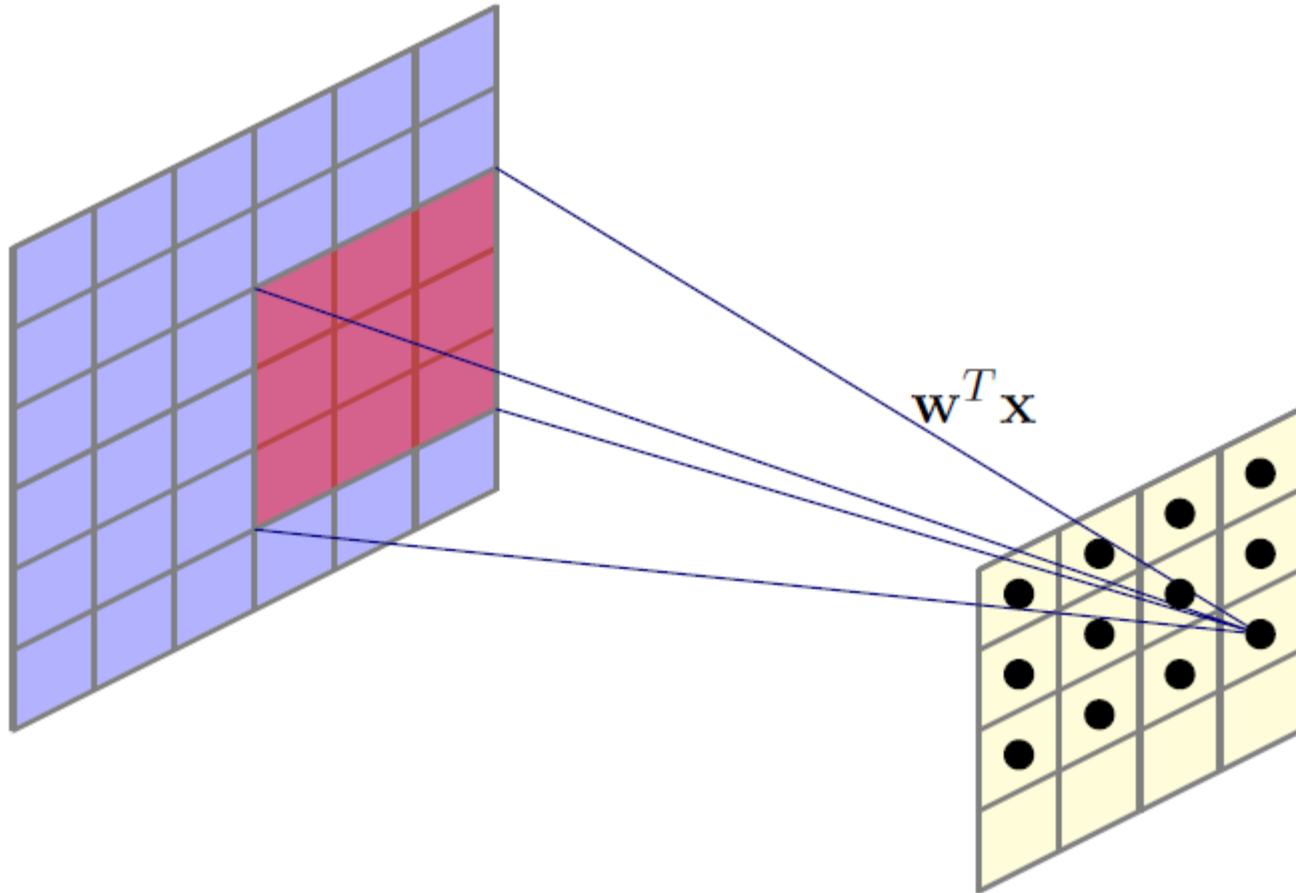
Convolution



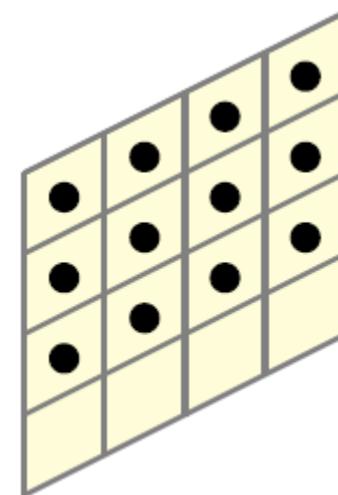
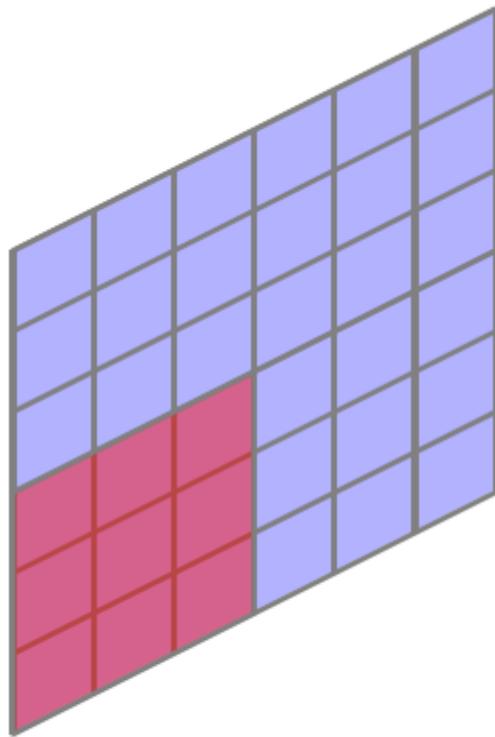
Convolution



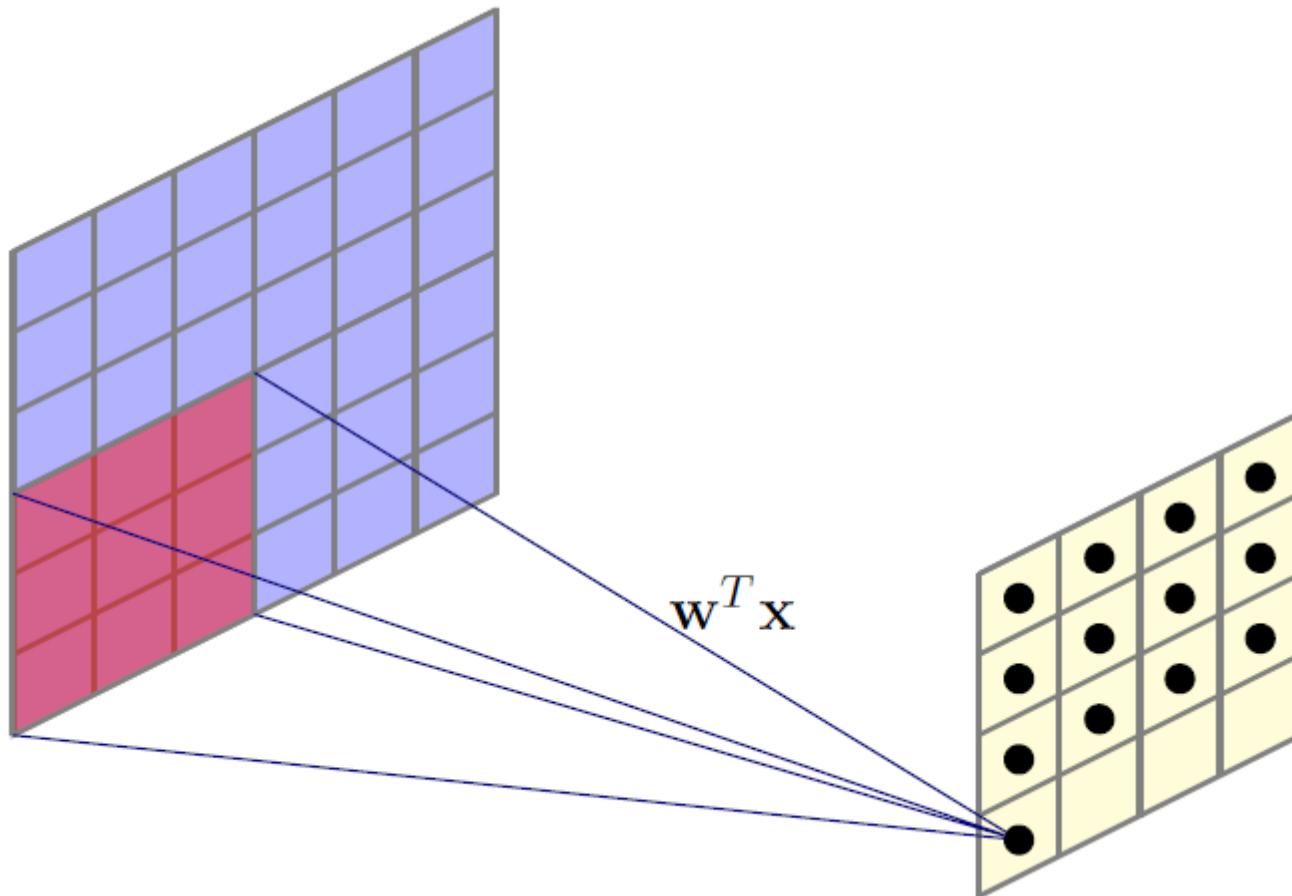
Convolution



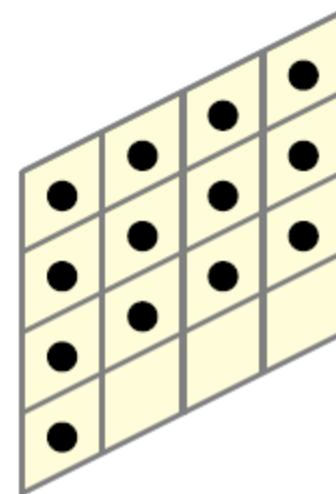
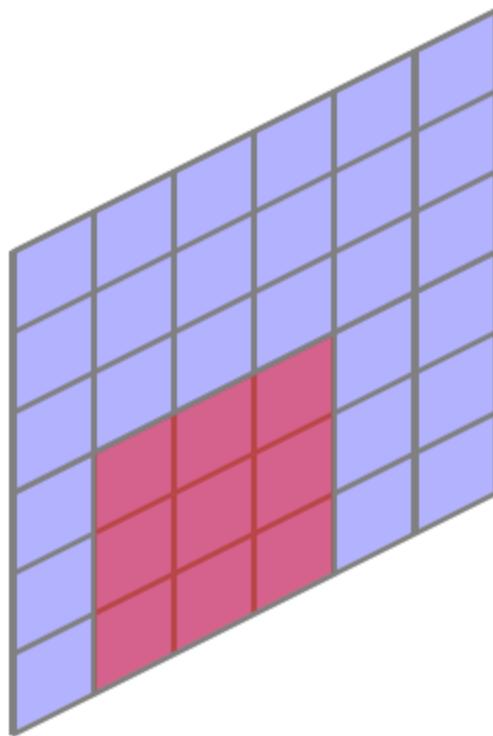
Convolution



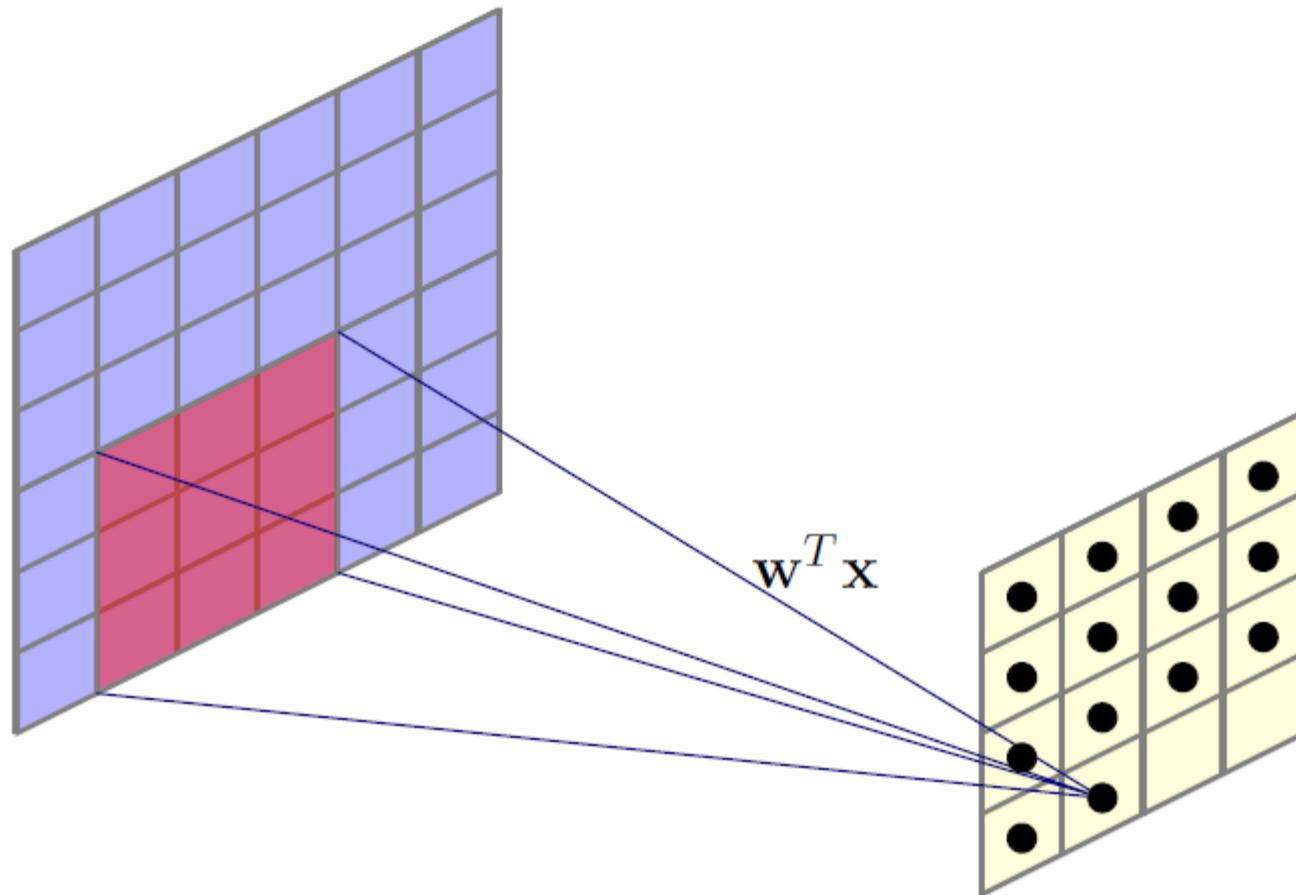
Convolution



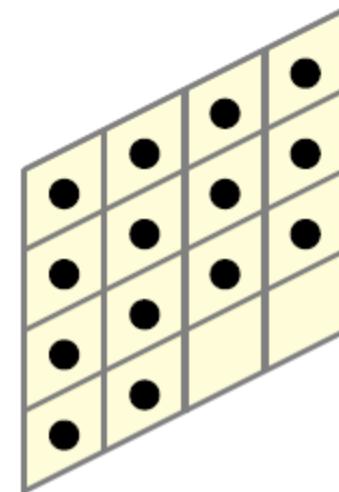
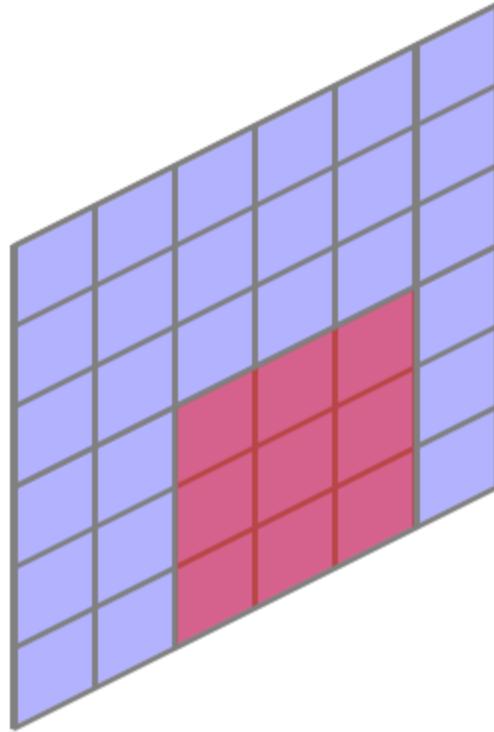
Convolution



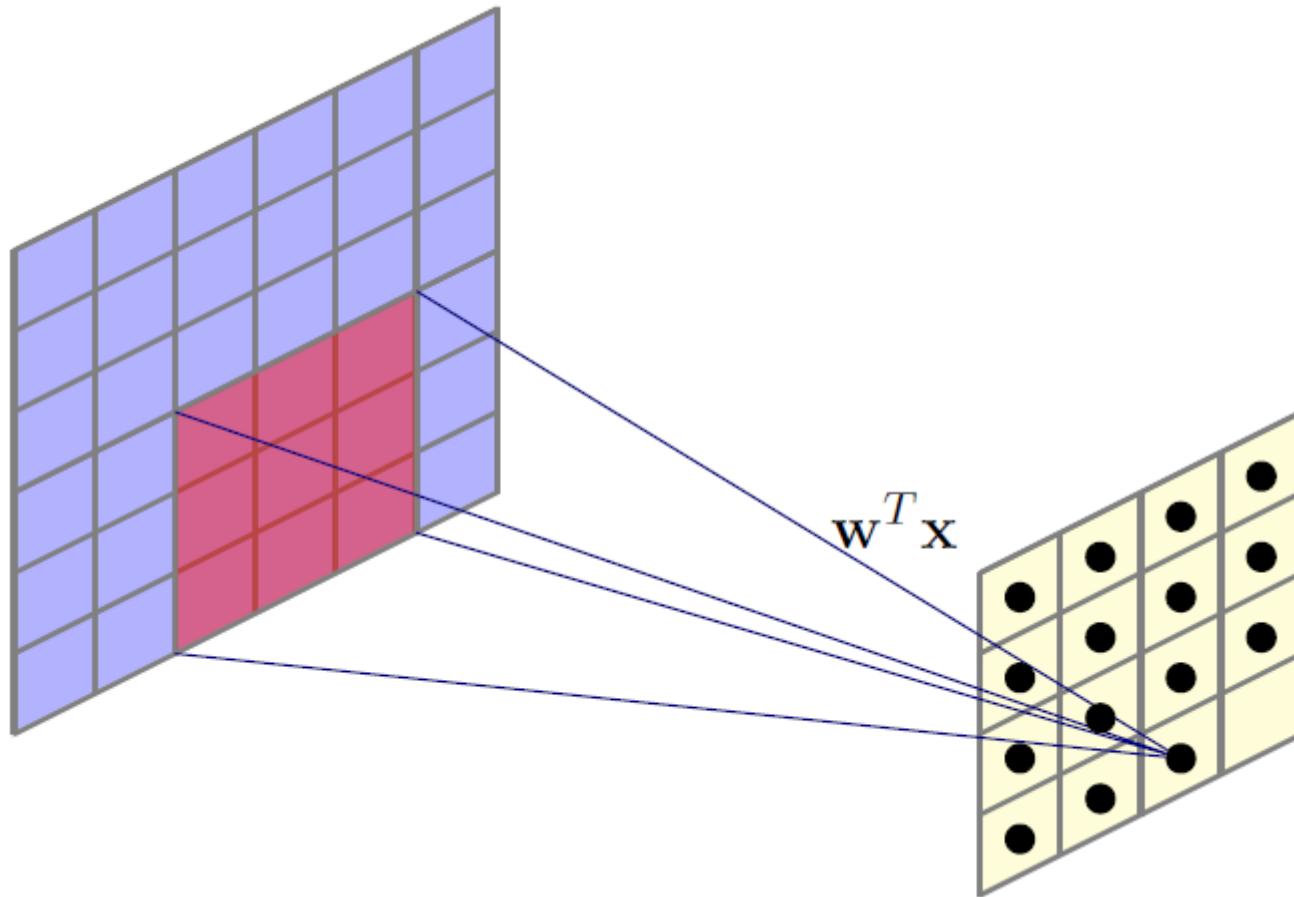
Convolution



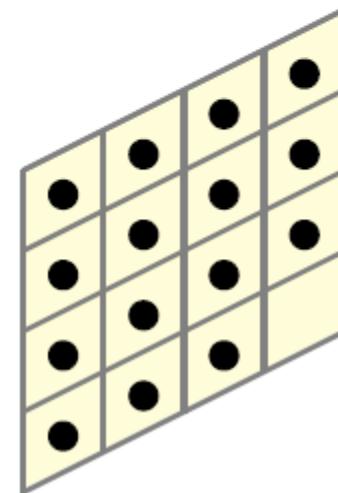
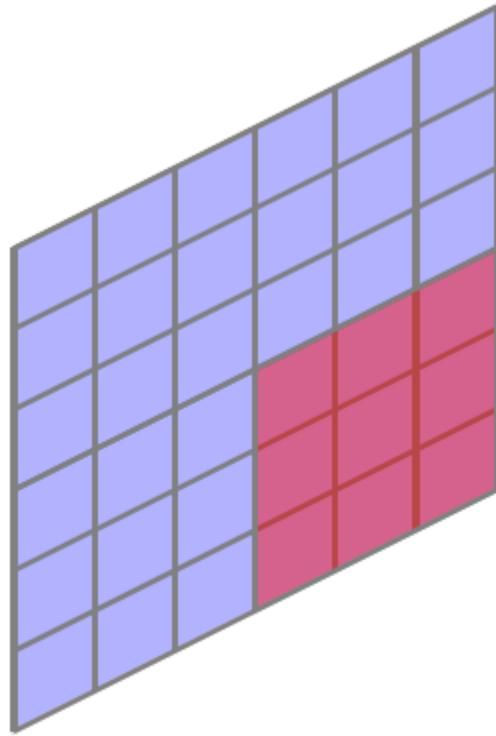
Convolution



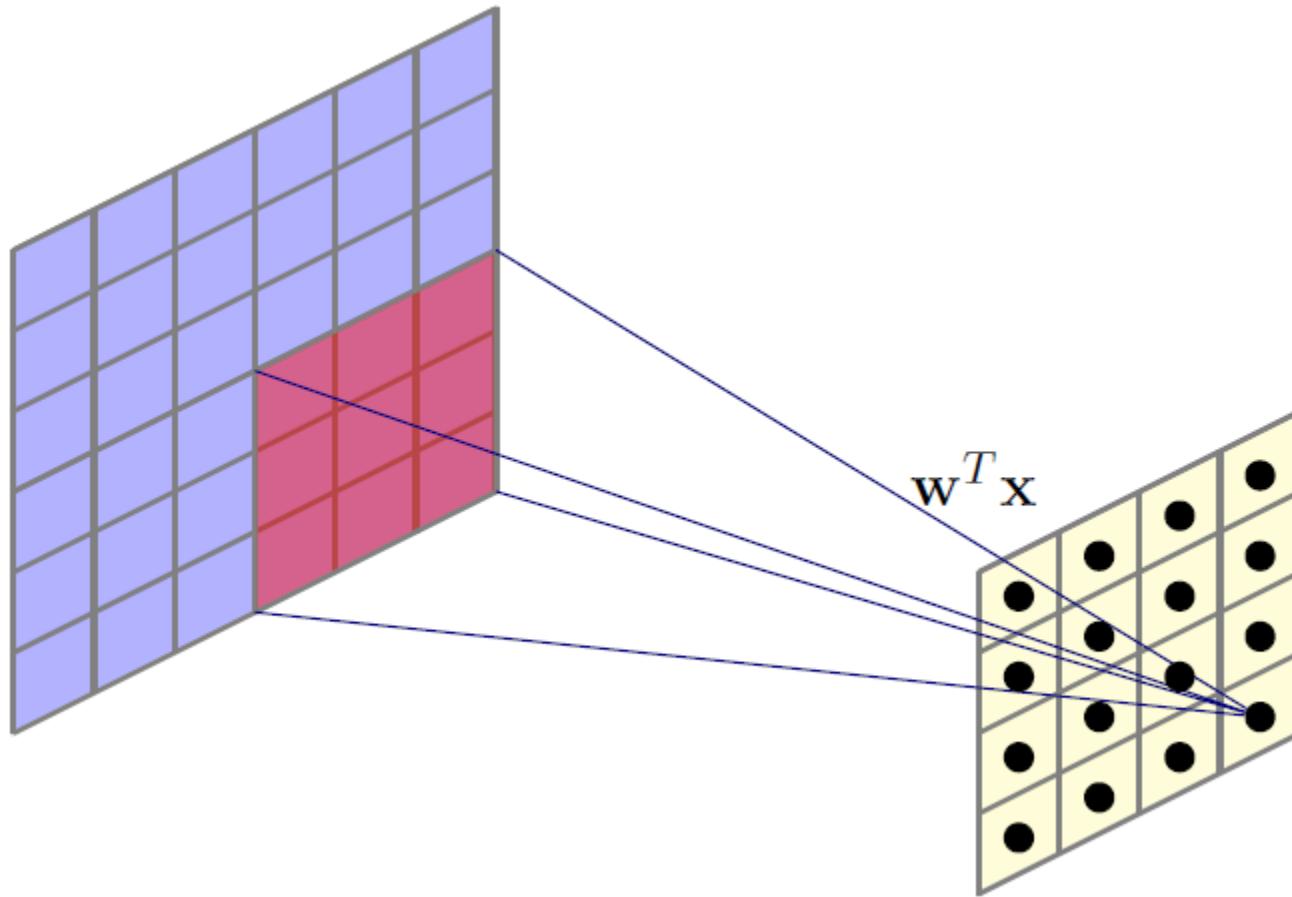
Convolution



Convolution



Convolution



What is the number of parameters?

Output Size

- We used **stride** of 1, kernel with **receptive field** of size 3 by 3
- Output size:

$$\frac{N - K}{S} + 1$$

- In previous example: $N = 6$, $K = 3$, $S = 1$, Output size = 4
- For $N = 8$, $K = 3$, $S = 1$, output size is 6

Zero Padding

- Often, we want the output of a convolution to have the same size as the input. Solution: Zero padding.
- In our previous example:

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

- Common to see convolution layers with stride of 1, filters of size K , and zero padding with $\frac{K-1}{2}$ to preserve size

Learn Multiple Filters

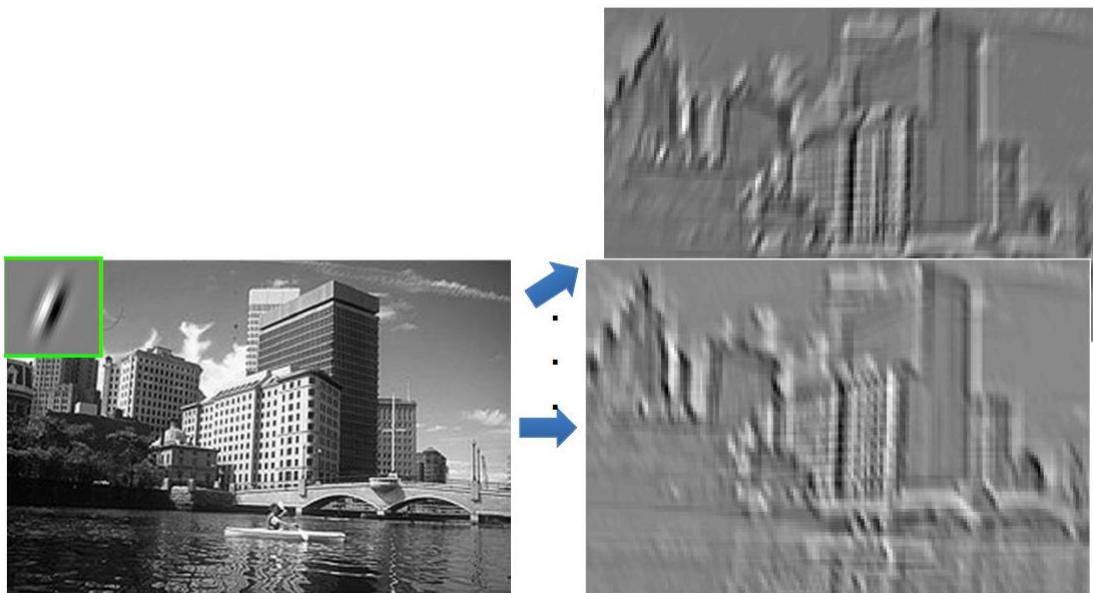


Figure: I. Kokkinos

- If we use 100 filters, we get 100 feature maps

Generalize to Higher Dimensions

- We have only considered a 2-D image as a running example
- But we could operate on volumes (e.g. RGB Images would be depth 3 input, filter would have same depth)

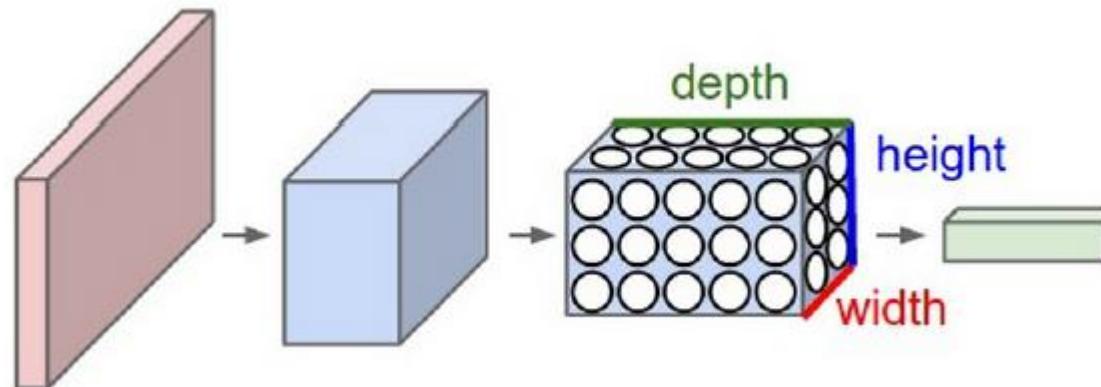


Image from Wikipedia

Output Size

- For convolutional layer:
 - Suppose input is of size $W_1 \times H_1 \times D_1$
 - Filter size is K and stride S
 - We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
 - As before:

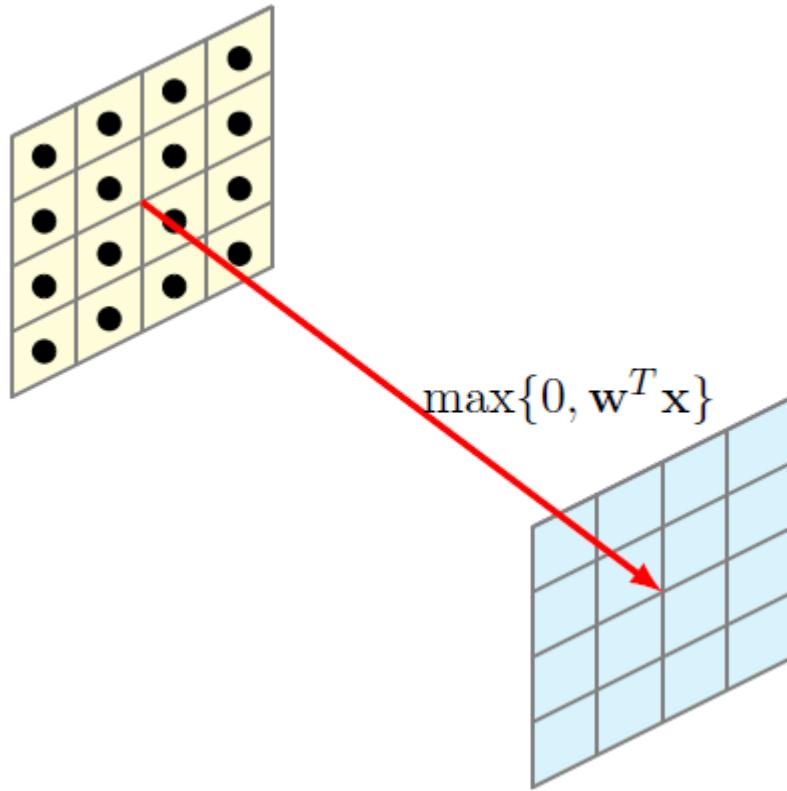
$$W_2 = \frac{W_1 - K}{S} + 1 \text{ and } H_2 = \frac{H_1 - K}{S} + 1$$

- Depths will be equal

Convolutional Layer Parameters

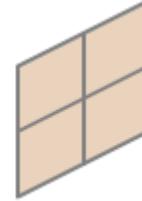
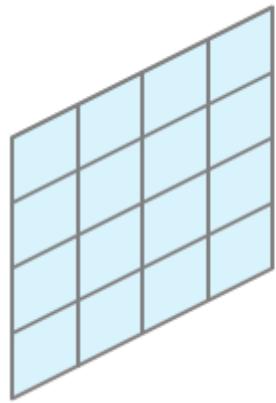
- Example volume: $28 \times 28 \times 3$ (RGB Image)
- 100 3×3 filters, stride 1
- What is the zero padding needed to preserve size?
- Number of parameters in this layer?
- For each filter: $3 \times 3 \times 3 + 1 = 28$ parameters
- Total parameters: $100 \times 28 = 2800$

Non-Linearity

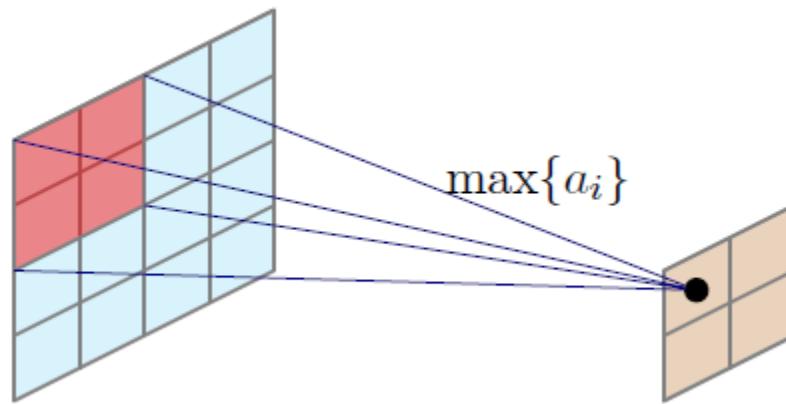


After obtaining feature map, apply an elementwise non-linearity to obtain a transformed feature map (same size)

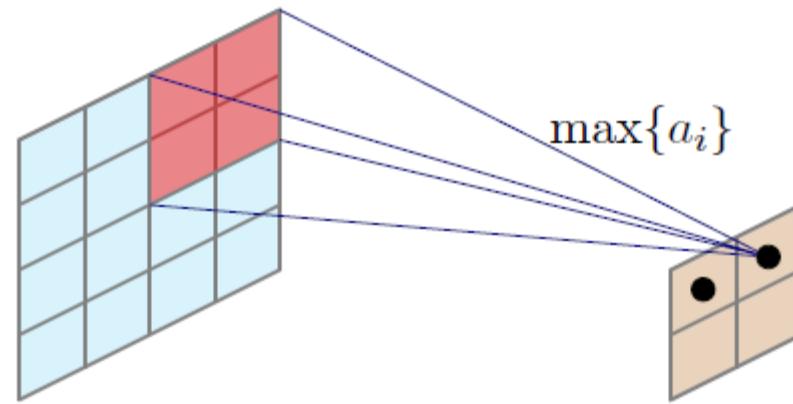
Pooling



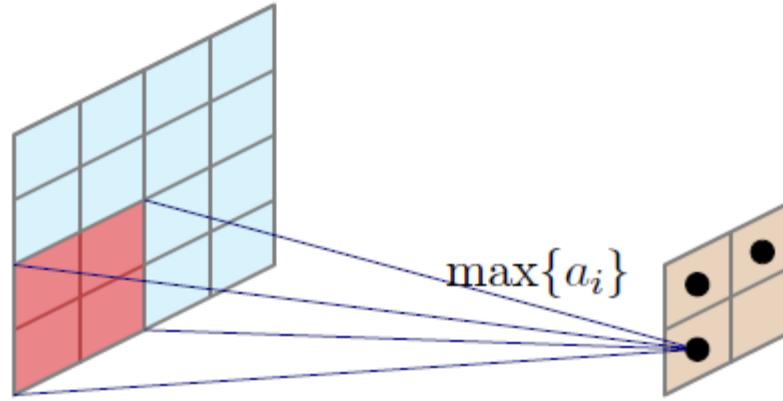
Pooling



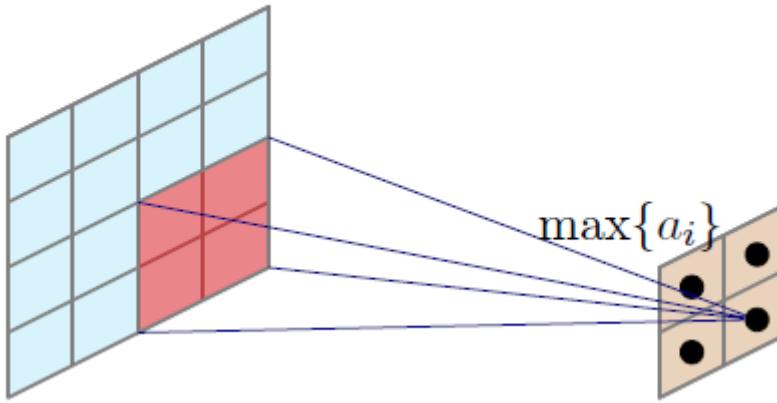
Pooling



Pooling

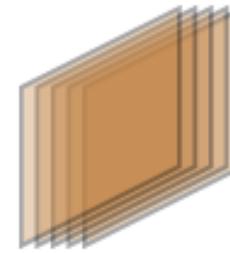
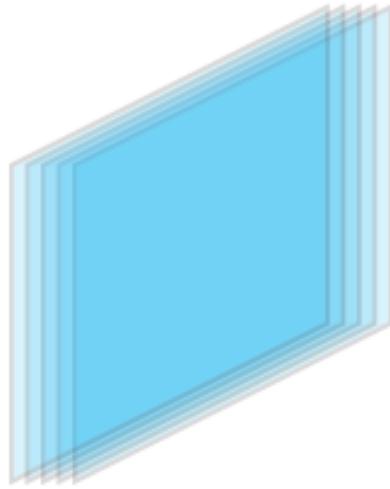


Pooling



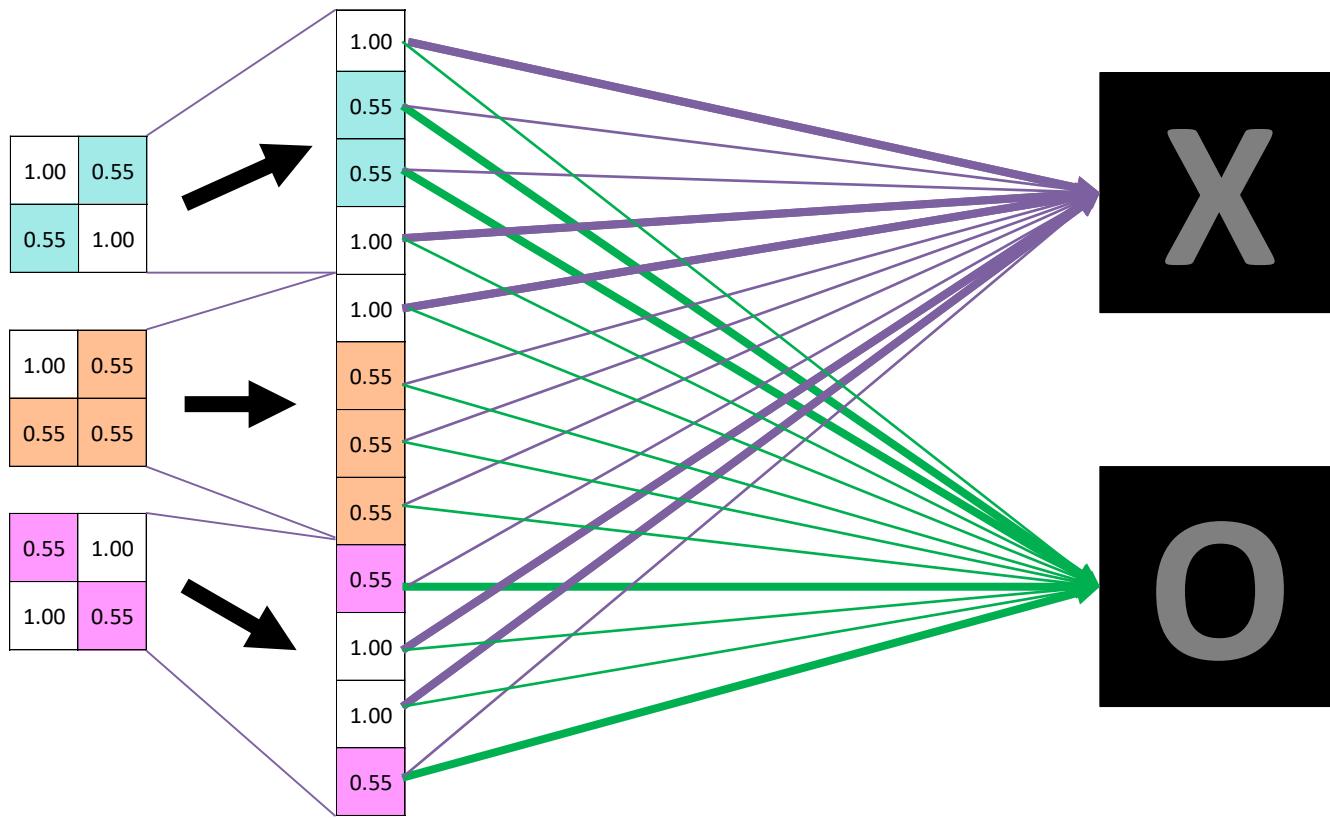
Other options: Average pooling, L2-norm pooling, random pooling

Pooling



- We have multiple feature maps, and get an equal number of subsampled maps
- This changes if cross channel pooling is done

Fully Connected Layer



- Take the high-level filtered images and translate them into vector
- Same as the other layers, they can be stacked to make better decisions

Outline

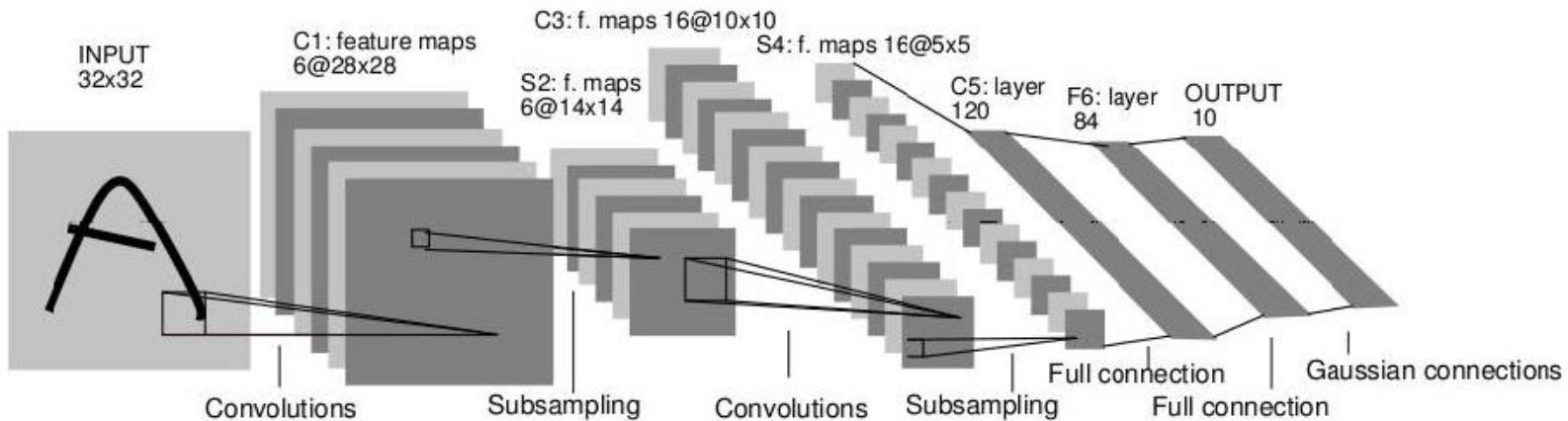
1 Course Review

2 Basic Operations

3 Major Architectures

4 CNN for Image Classification

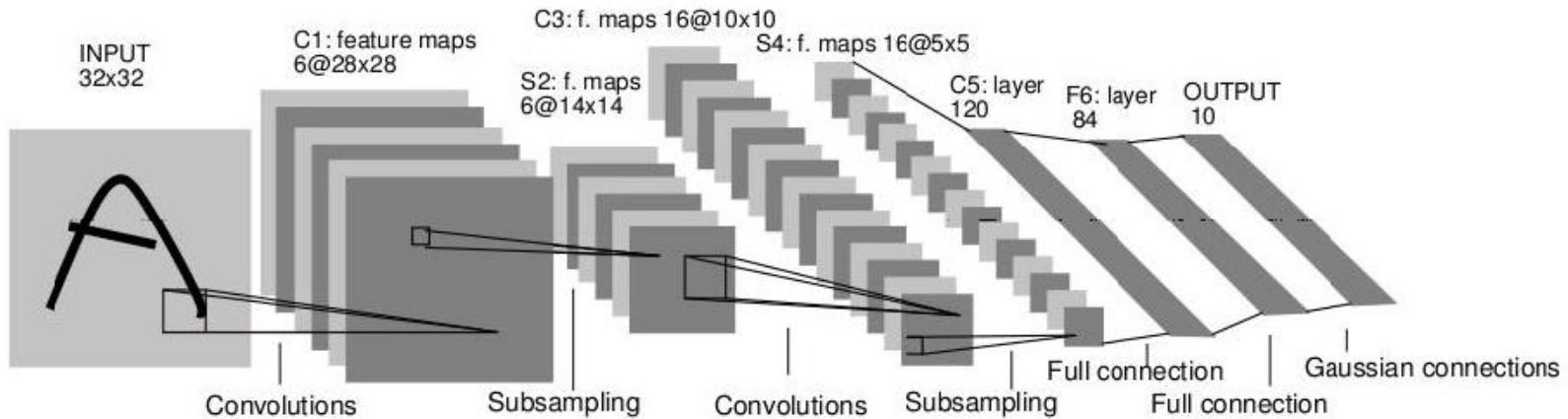
LeNet-5



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

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

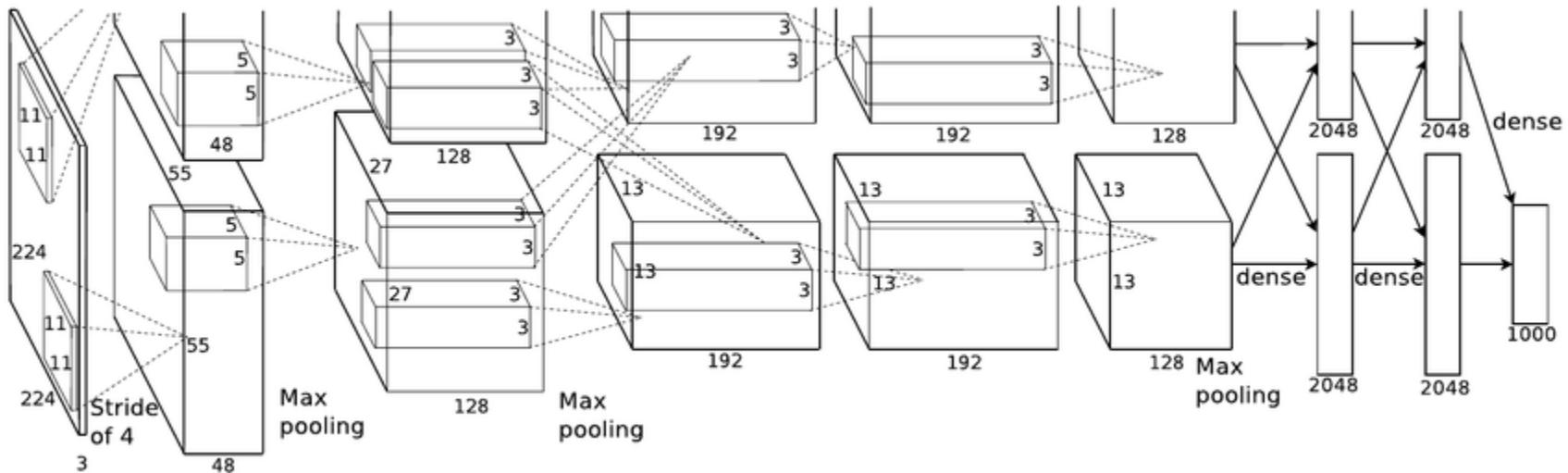
LeNet-5



- Filters are of size 5×5 , stride 1
- Pooling is 2×2 , with stride 2
- How many parameters?

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

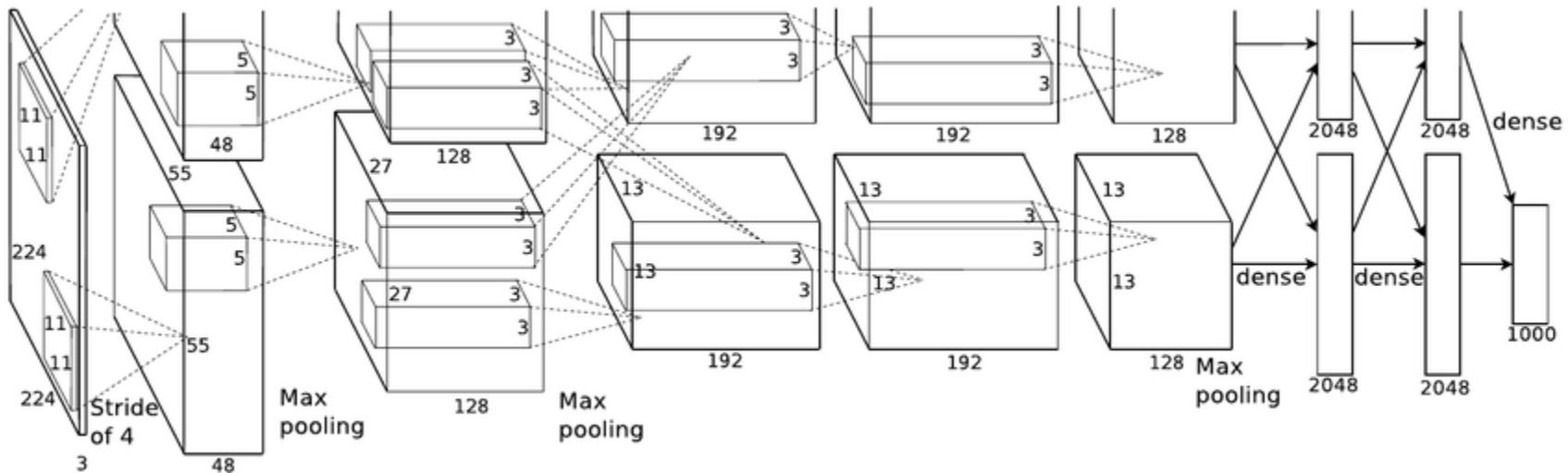
AlexNet: ILSVRC 2012 winner



- Similar framework to LeNet but:
 - Max pooling, ReLU nonlinearity
 - More data and bigger model (7 hidden layers, 650K units, 60M params)
 - GPU implementation (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

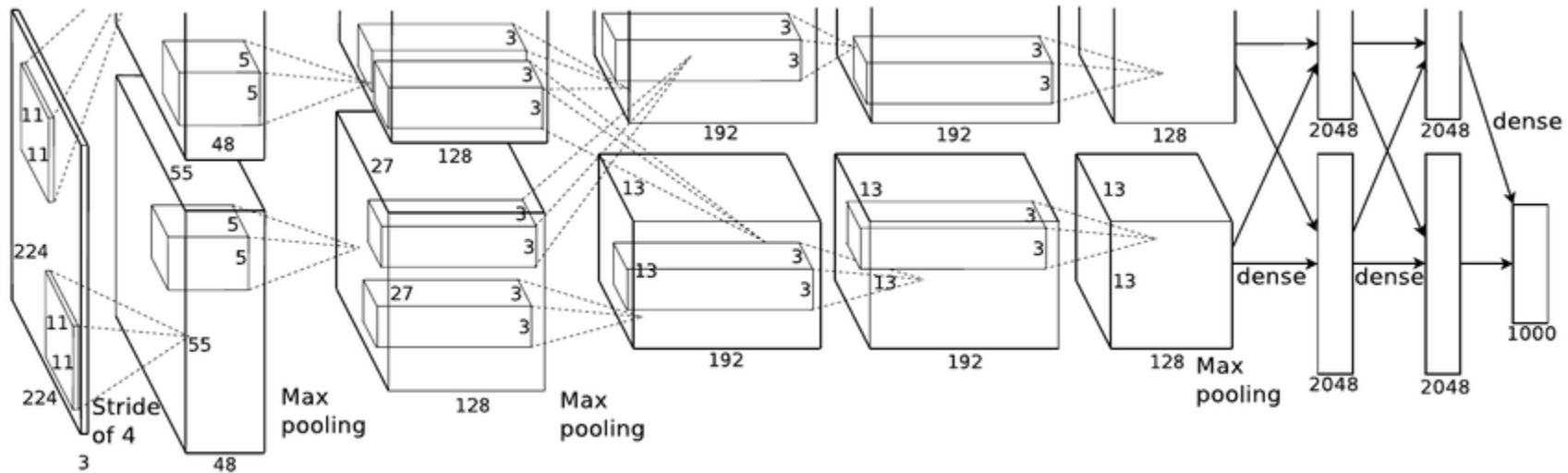
AlexNet: ILSVRC 2012 winner



- Input image: 227 X 227 X 3
- First convolutional layer: 96 filters with K = 11 applied with stride = 4
- Width and height of output: $\frac{227 - 11}{4} + 1 = 55$

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

AlexNet: ILSVRC 2012 winner

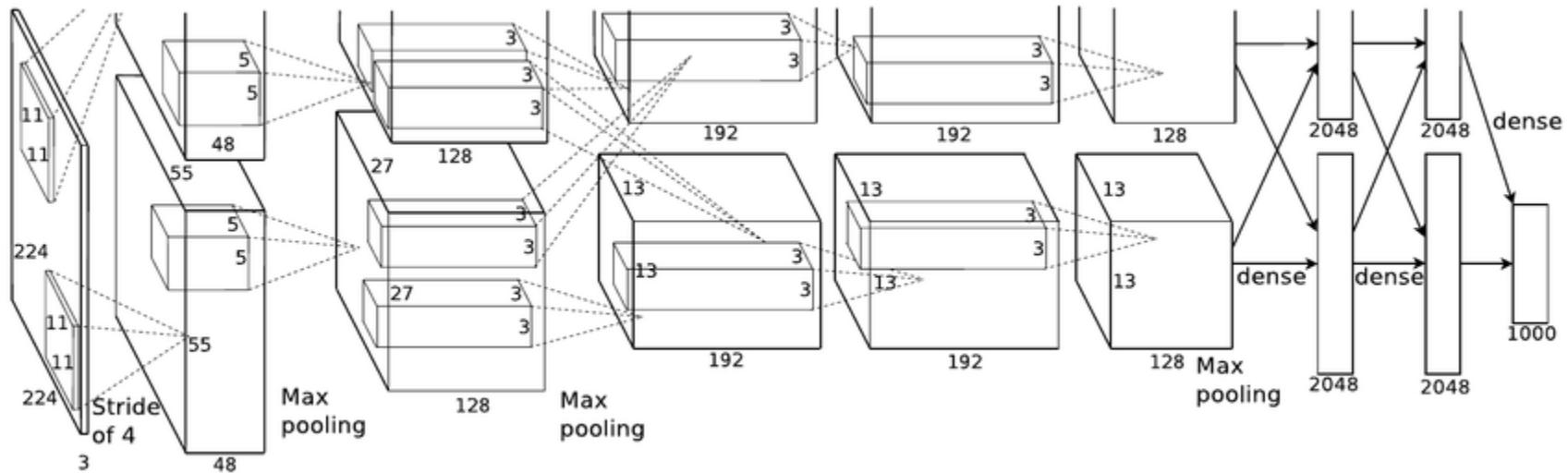


Number of parameters in first layer?

$$11 \times 11 \times 3 \times 96 = 34848$$

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

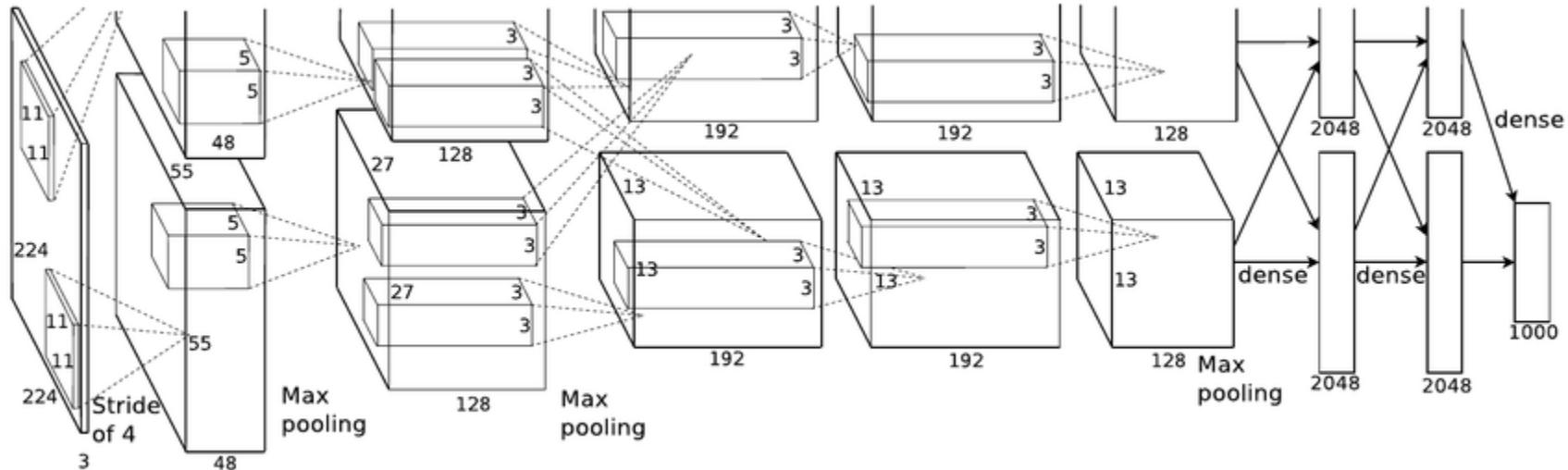
AlexNet: ILSVRC 2012 winner



- Next layer: Pooling with 3 X 3 filters, stride of 2
- Size of output volume: 27
- Number of parameters?

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

AlexNet: ILSVRC 2012 winner



- Popularized the use of ReLUs
- Used heavy data augmentation (flipped images, random crops of size 227 by 227)
- Parameters: Dropout rate 0.5, Batch size = 128, Weight decay term: 0.0005 ,Momentum term = 0.9, learning rate = 0.01, manually reduced by factor of ten on monitoring validation loss

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

Clarifai: ILSVRC 2013 winner

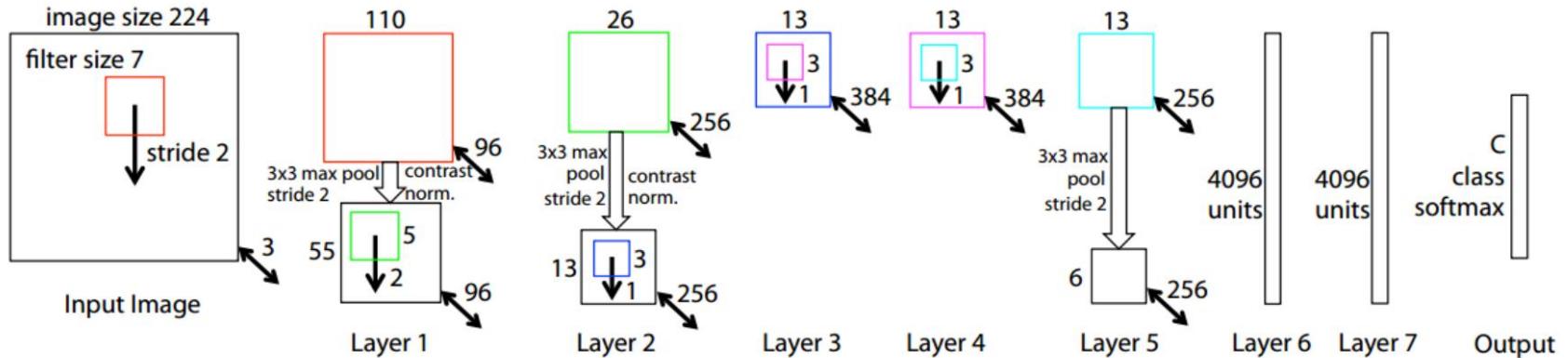


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C -way softmax function, C being the number of classes. All filters and feature maps are square in shape.

- Refinement of AlexNet

M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014 (Best Paper Award winner)

VGGNet: ILSVRC 2014 2nd place

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3×3 convolutions (with ReLU in between)
- One 7×7 conv layer with C feature maps needs $49C^2$ weights, three 3×3 conv layers need only $27C^2$ weights

Best model: Column D.

Error: 7.3 % (top five error)

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

VGGNet: ILSVRC 2014 2nd place

- Total number of parameters: **138 Million** (calculate!)
- Memory (Karpathy): **24 Million X 4** bytes 93 MB per image
- For backward pass the memory usage is doubled per image
- Observations:
 - Early convolutional layers take most memory
 - Most parameters are in the fully connected layers

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

GoogLeNet: ILSVRC 2014 winner

- The Inception Module

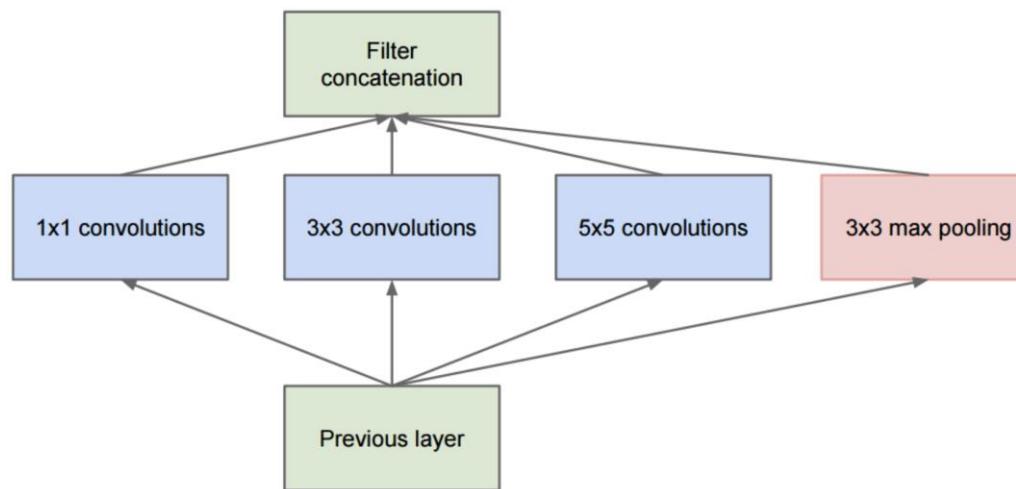


<http://knowyourmeme.com/memes/we-need-to-go-deeper>

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

GoogLeNet: ILSVRC 2014 winner

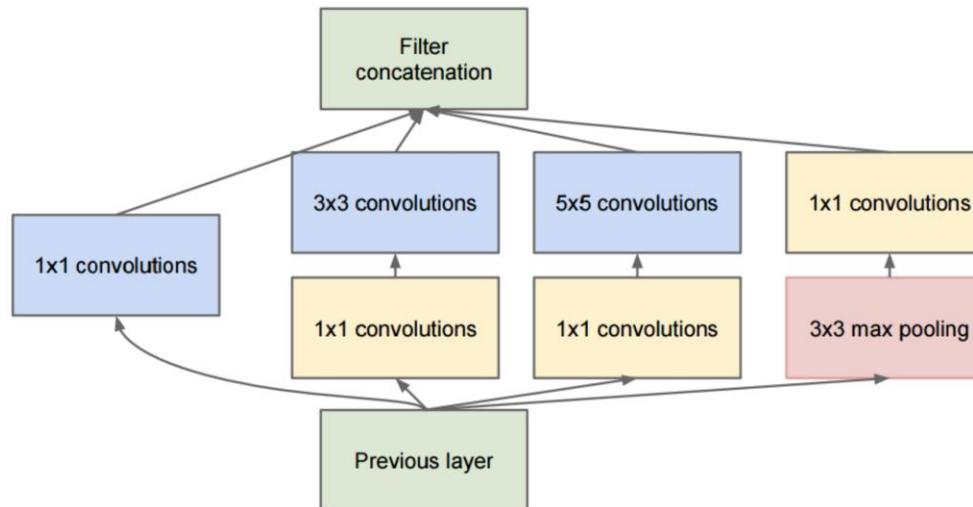
- The Inception Module
 - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps



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

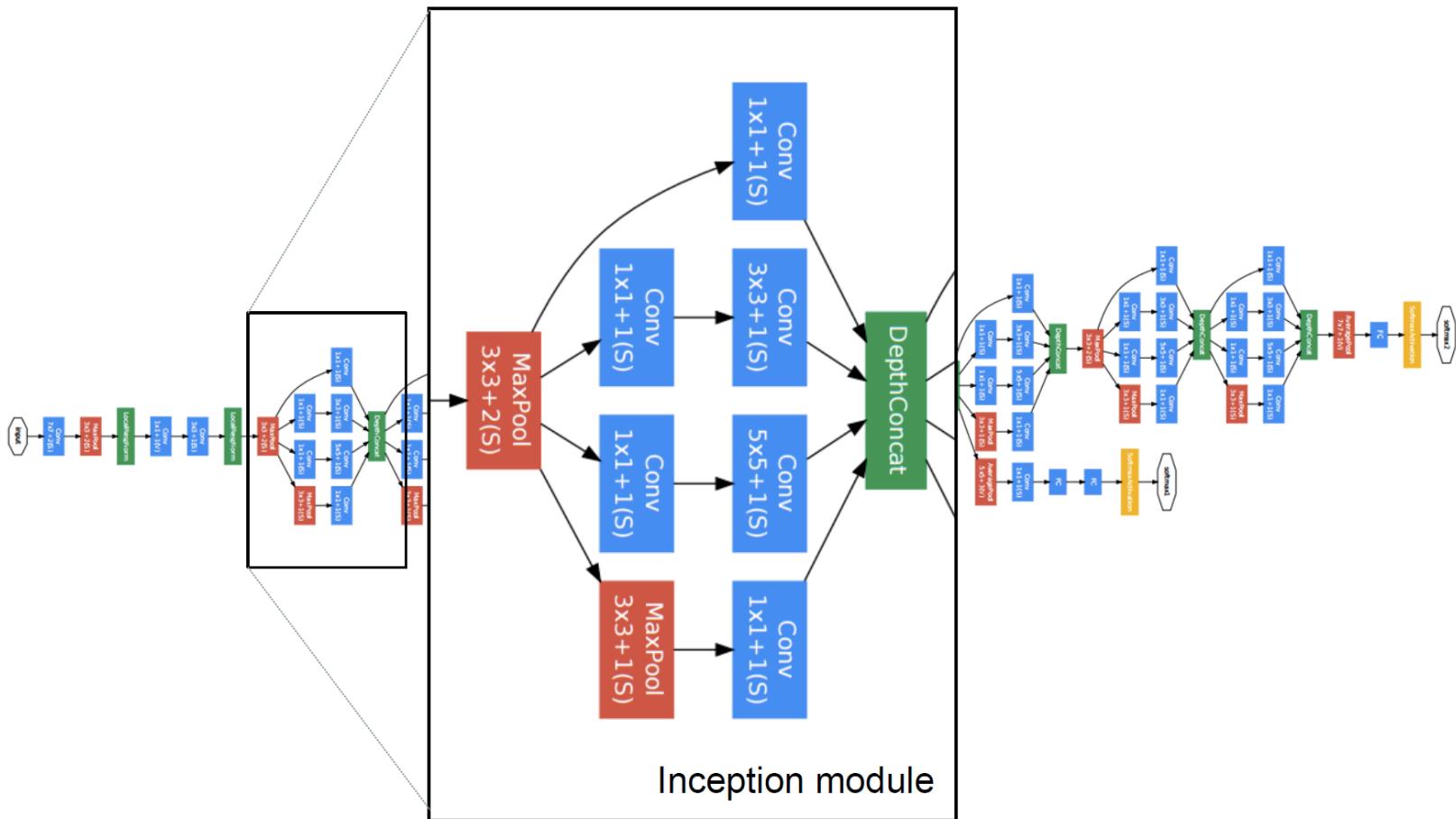
GoogLeNet: ILSVRC 2014 winner

- The Inception Module
 - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
 - Use 1×1 convolutions for dimensionality reduction before expensive convolutions



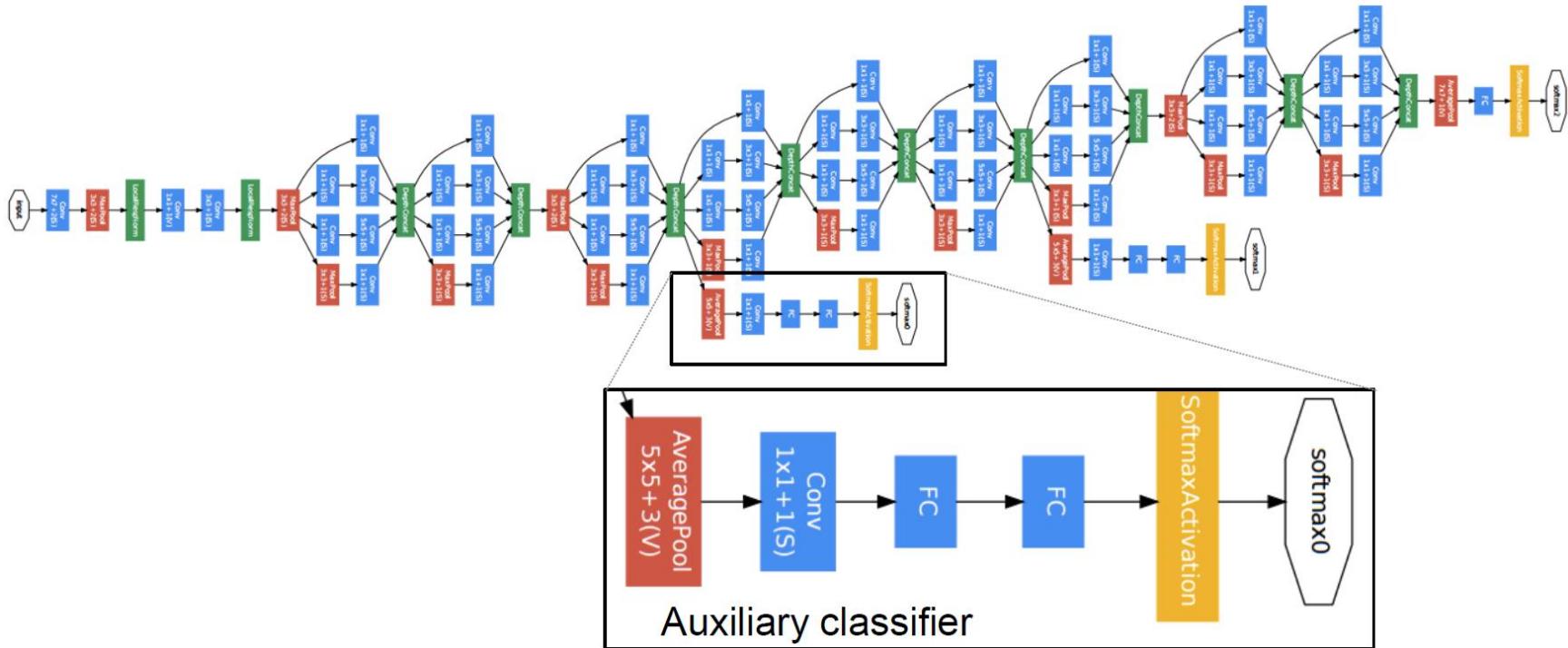
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

GoogLeNet: ILSVRC 2014 winner



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

GoogLeNet: ILSVRC 2014 winner



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

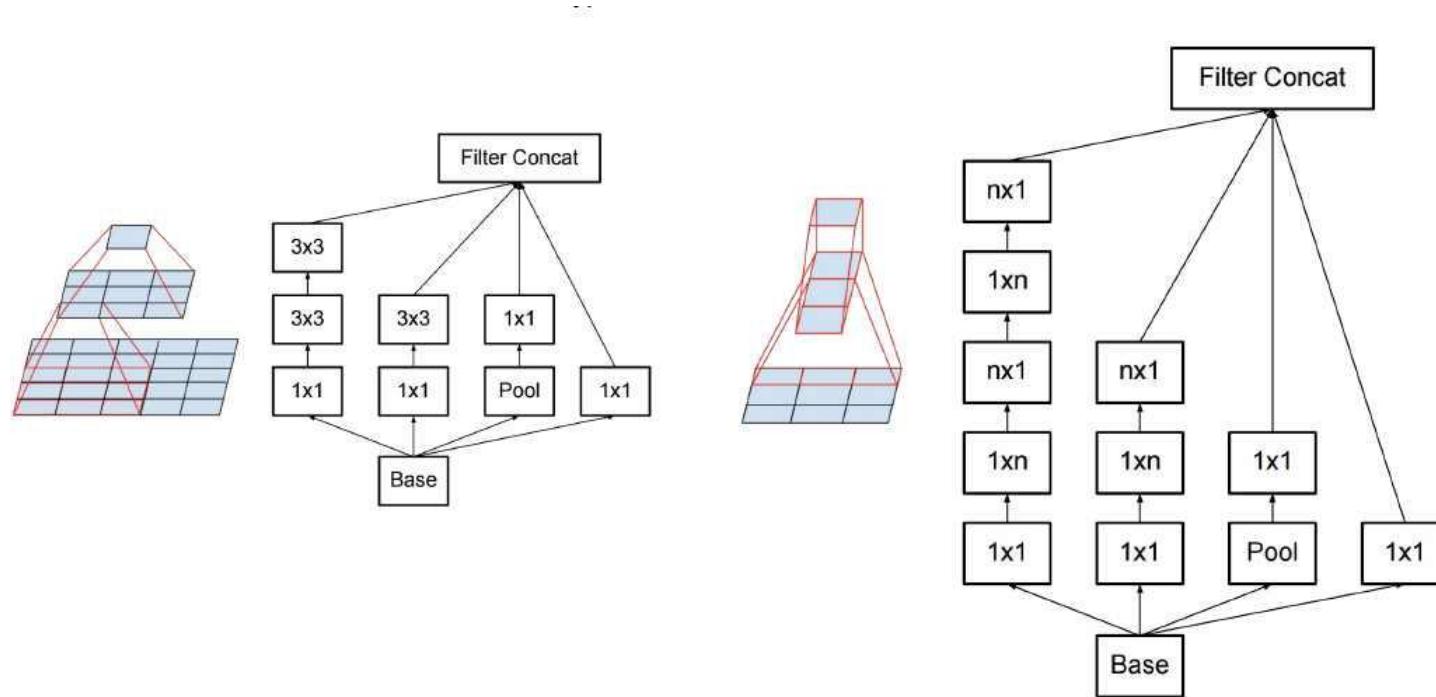
GoogLeNet: ILSVRC 2014 winner

An alternative view:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Has 5 Million or 12X fewer parameters than AlexNet Gets rid of fully connected layers

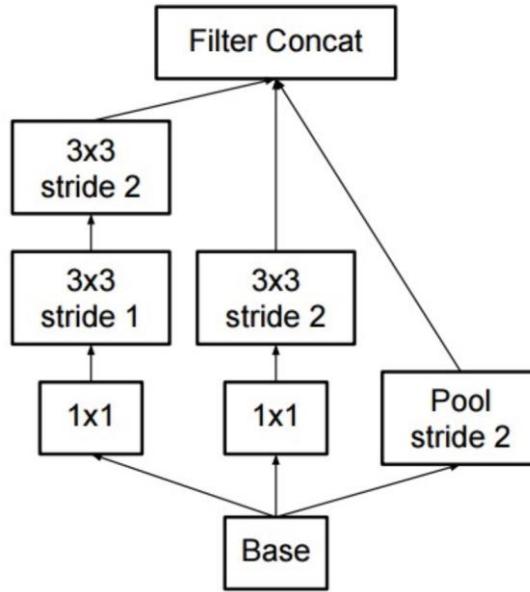
Inception v2, v3



- Regularize training with batch normalization, reducing importance of auxiliary classifiers
- More variants of inception modules with aggressive factorization of filters

C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

Inception v2, v3



- Regularize training with batch normalization, reducing importance of auxiliary classifiers
- More variants of inception modules with aggressive factorization of filters
- Increase the number of feature maps while decreasing spatial resolution (pooling)

C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

ResNet: ILSVRC 2015 winner

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

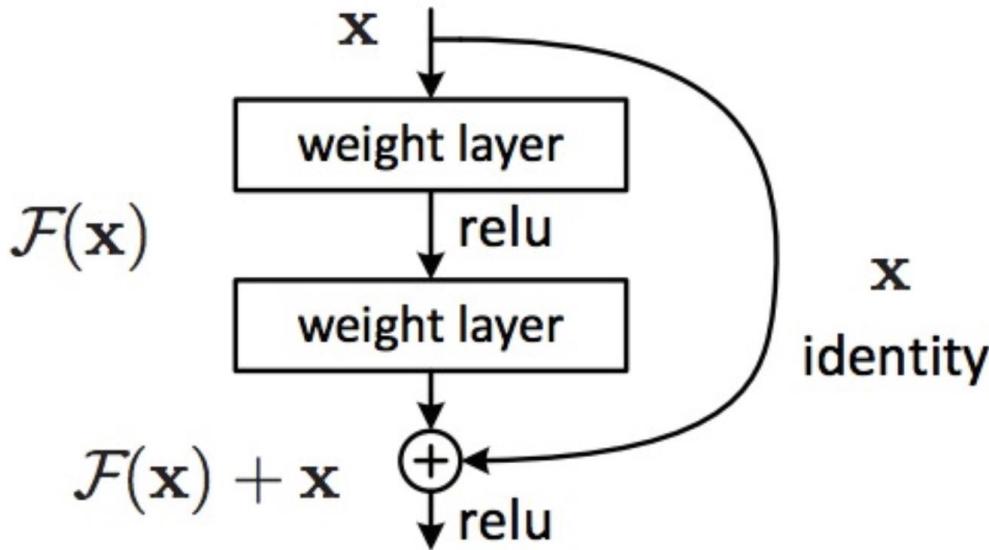


ResNet, **152 layers**
(ILSVRC 2015)



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

ResNet: ILSVRC 2015 winner



- The residual module
 - Introduce skip or shortcut connections (existing before in various forms in literature)
 - Make it easy for network layers to represent the identity mapping

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

ResNet: ILSVRC 2015 winner

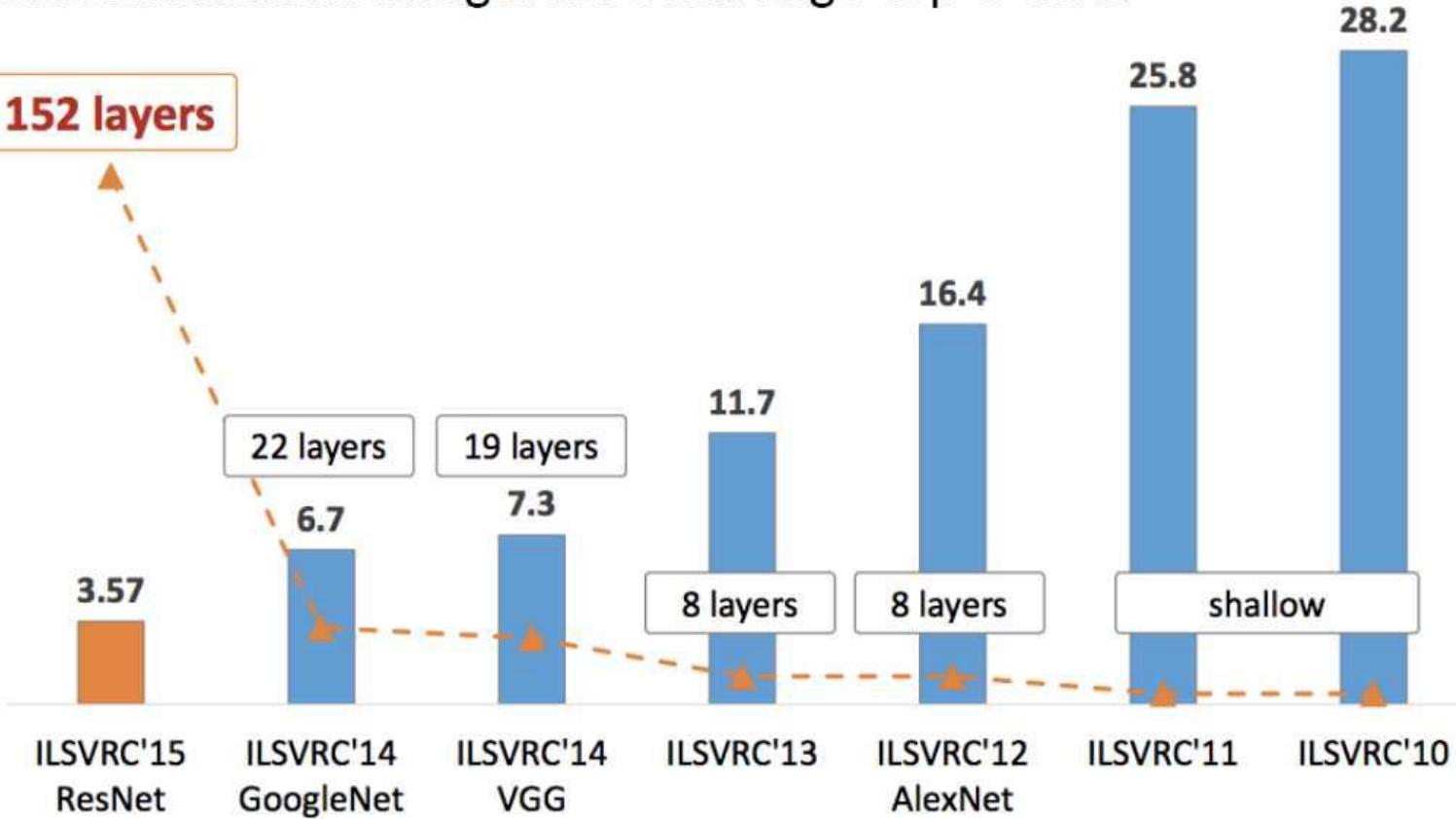
Architectures for ImageNet:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56			3×3 max pool, stride 2		
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

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

Going Deeper

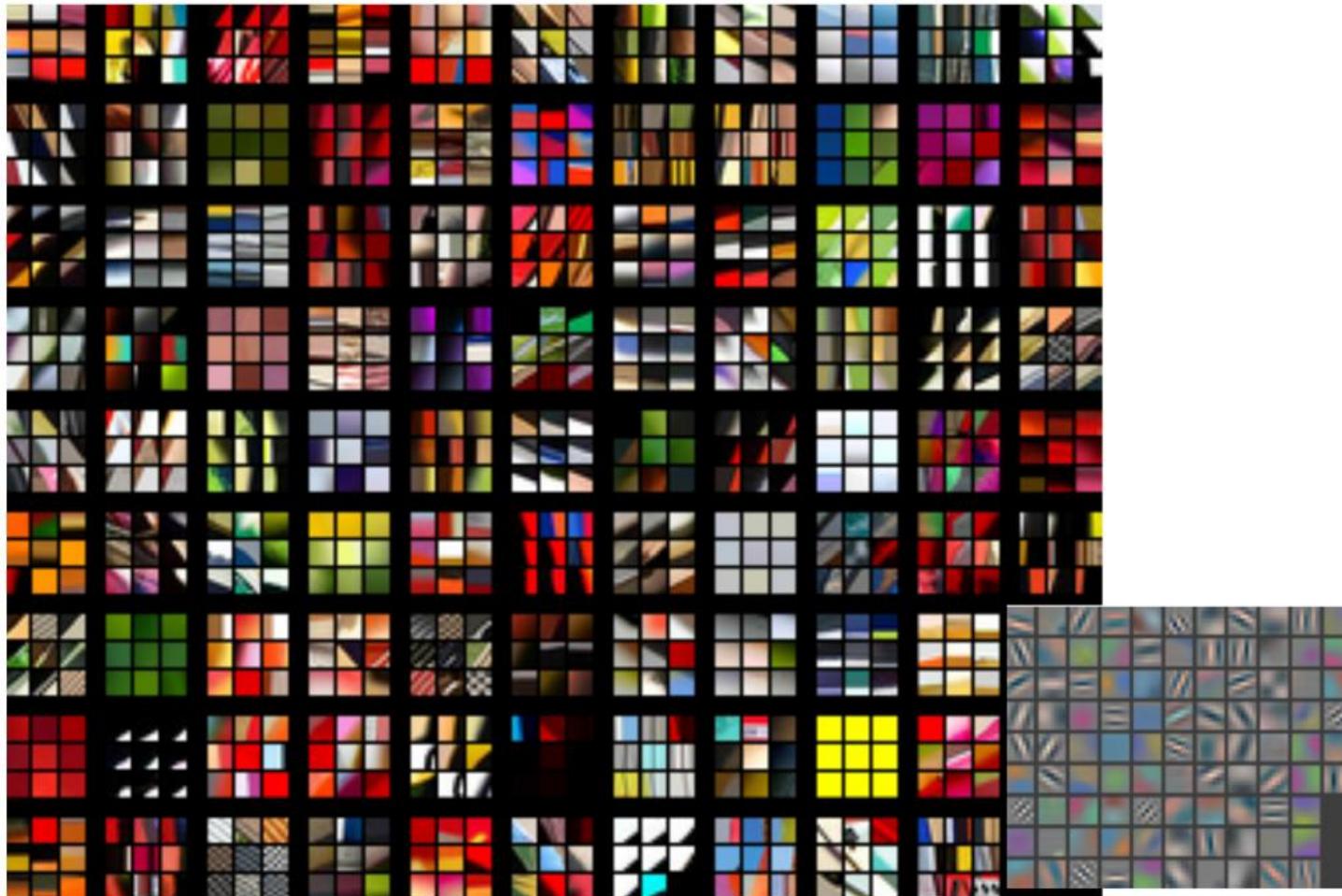
Classification: ImageNet Challenge top-5 error





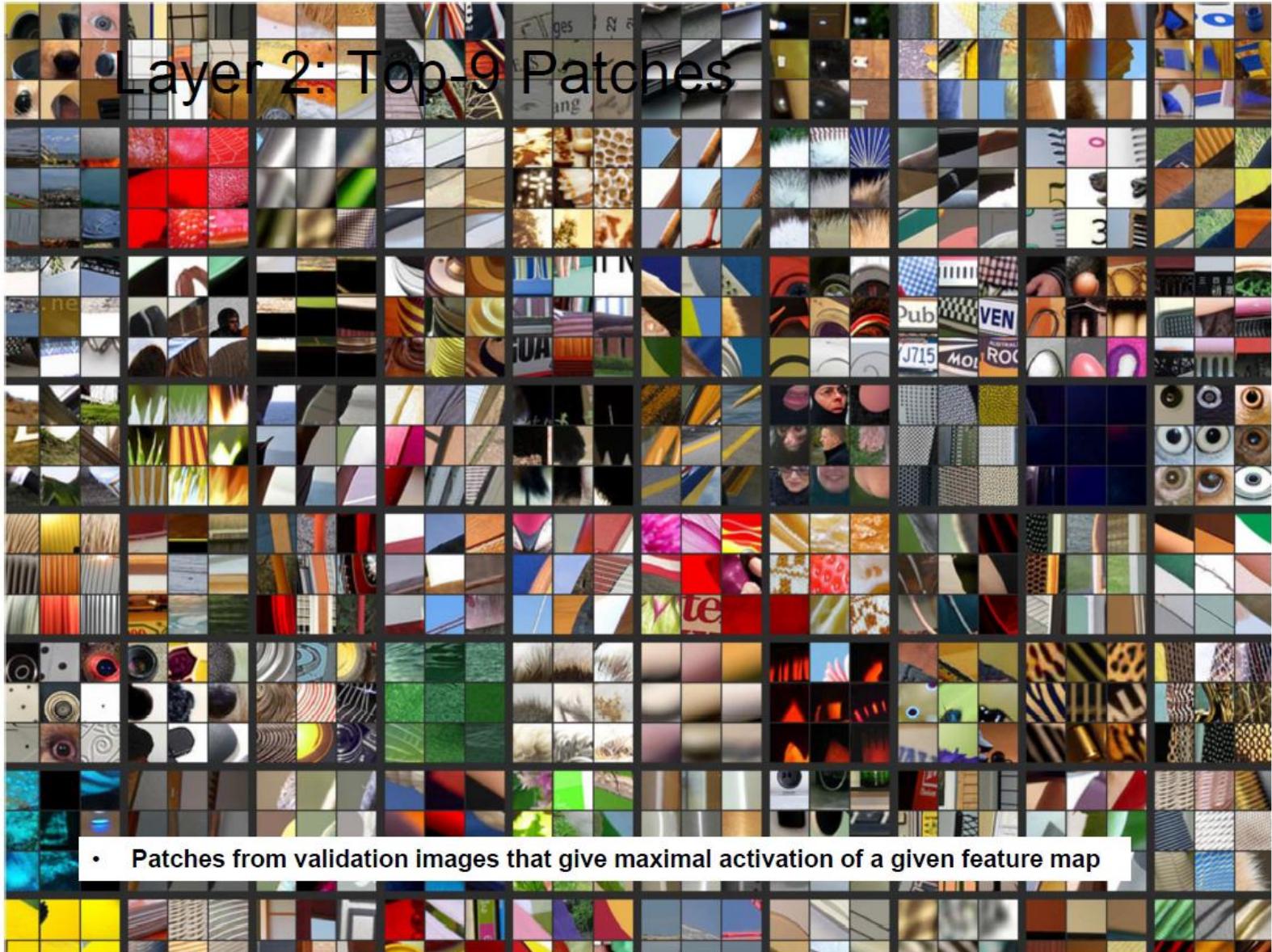
How do the features look like ?

Layer 1 filters



This and the next few illustrations are from Rob Fergus

Layer 2 Patches



Layer 2 Patches

Layer 2: Top-9 Patches



Layer 3 Patches

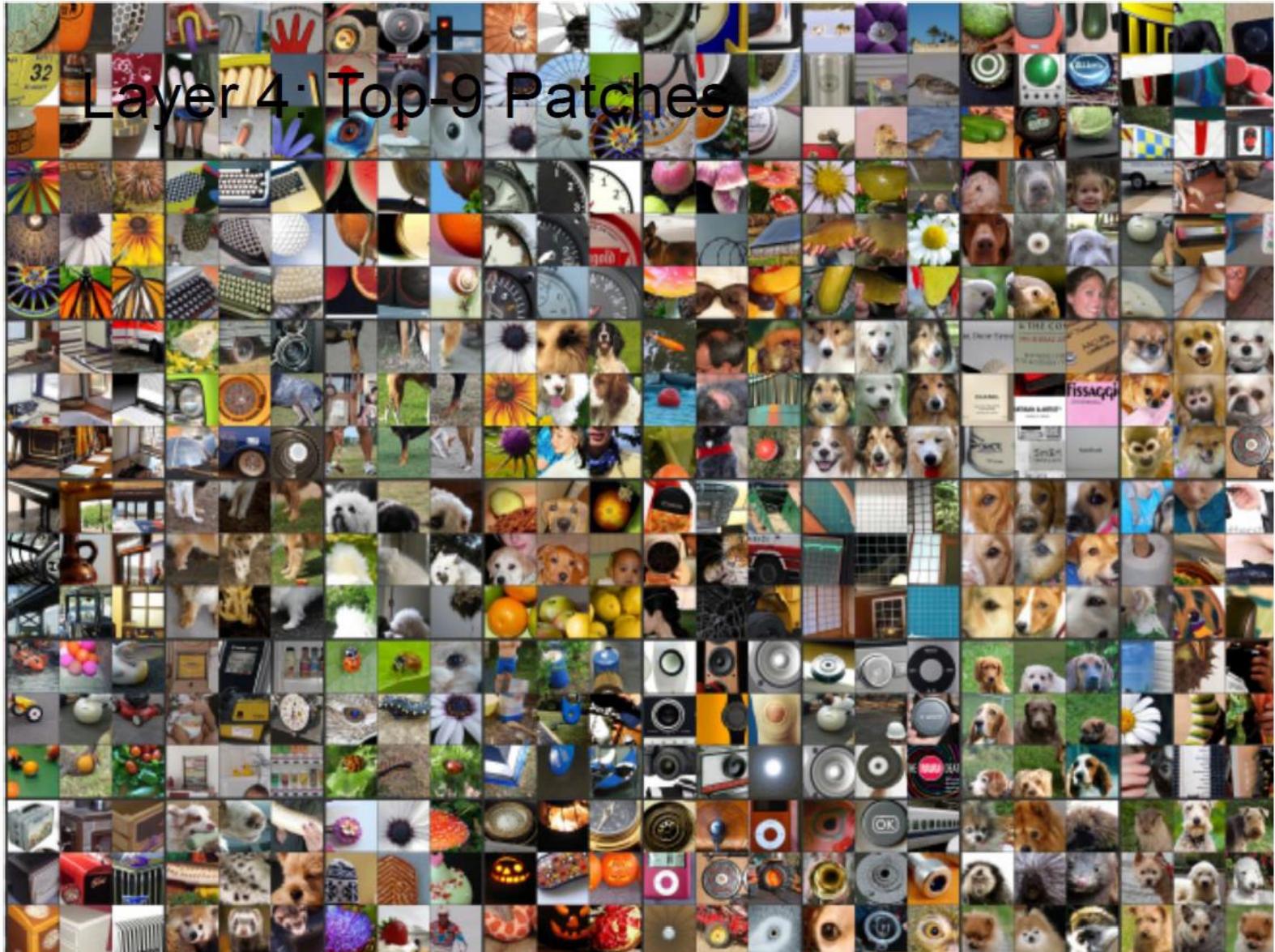


Layer 3: Top-9 Patches

Layer 3 Patches



Layer 4 Patches

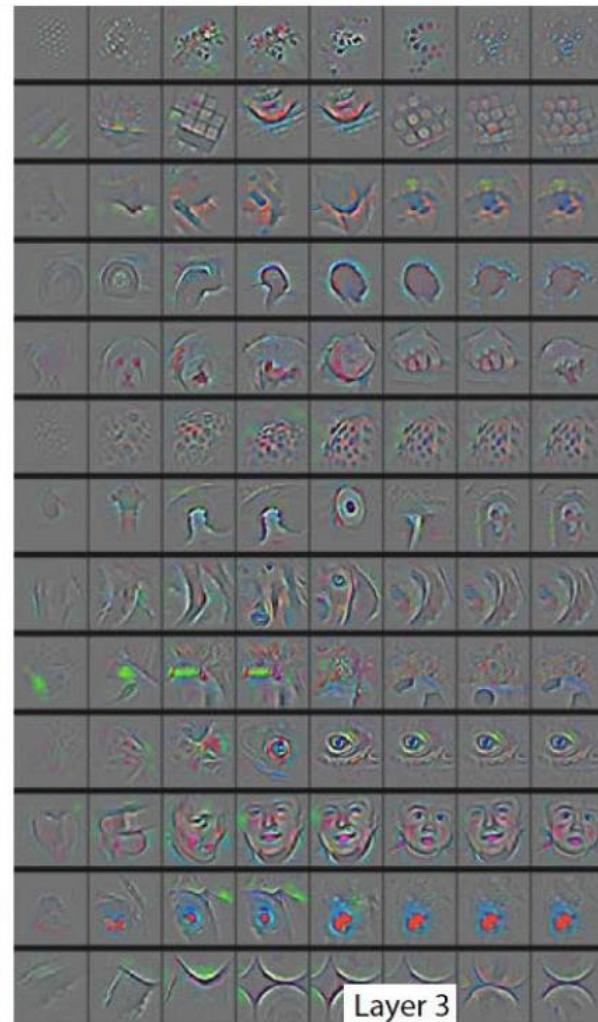
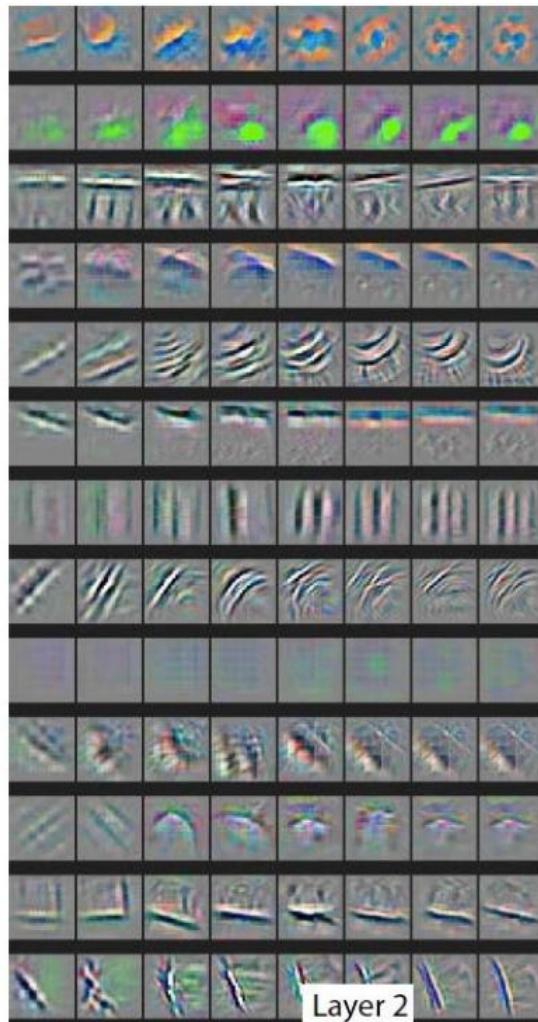
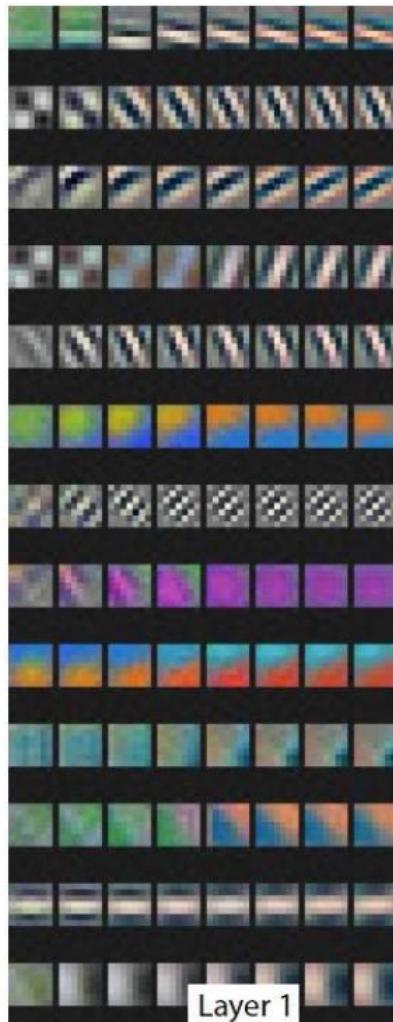


Layer 4 Patches

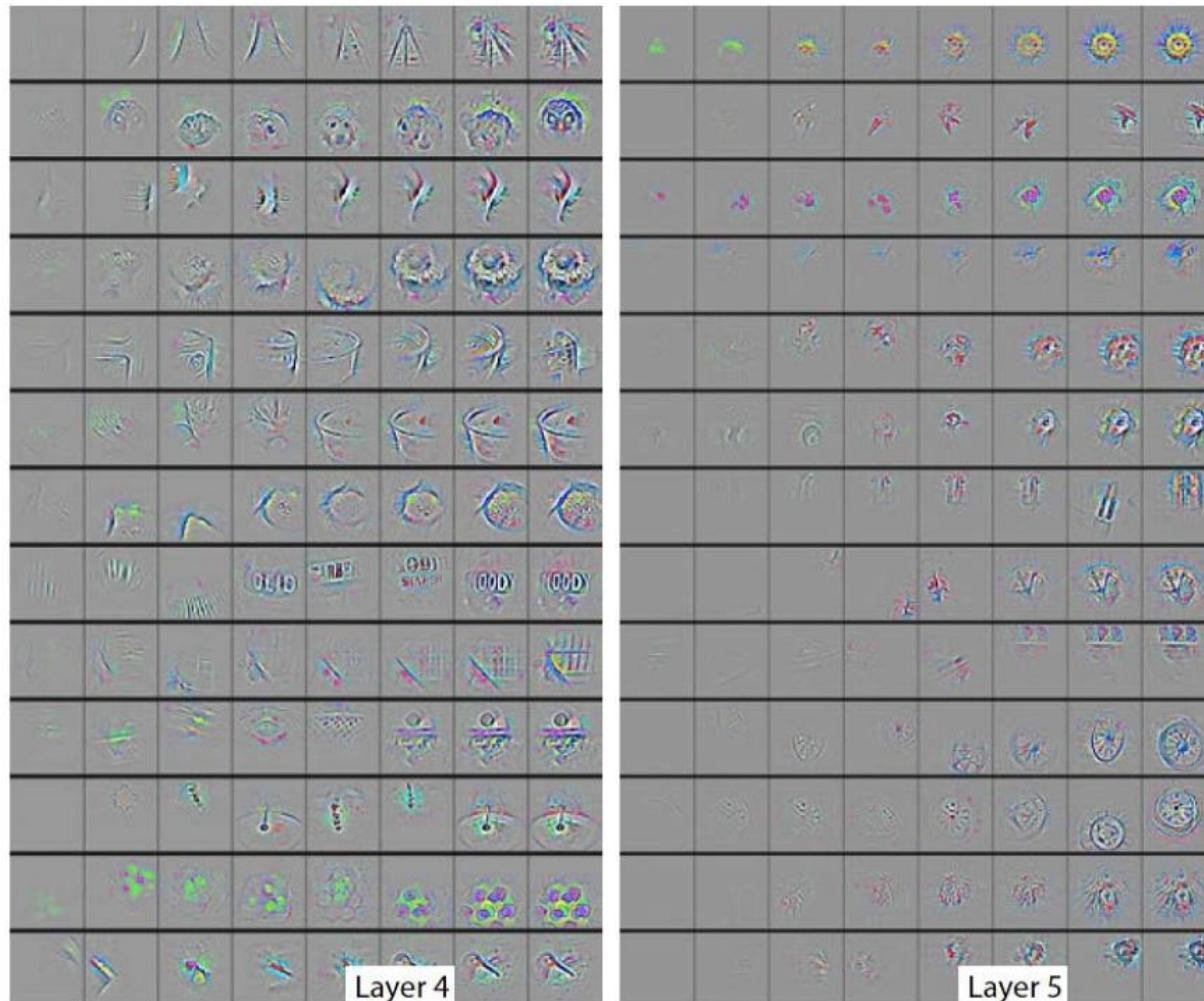
Layer 4: Top-9 Patches



Evolution of Filters



Evolution of Filters



Caveat?

Outline

1 Course Review

2 Basic Operations

3 Major Architectures

4 CNN for Image Classification

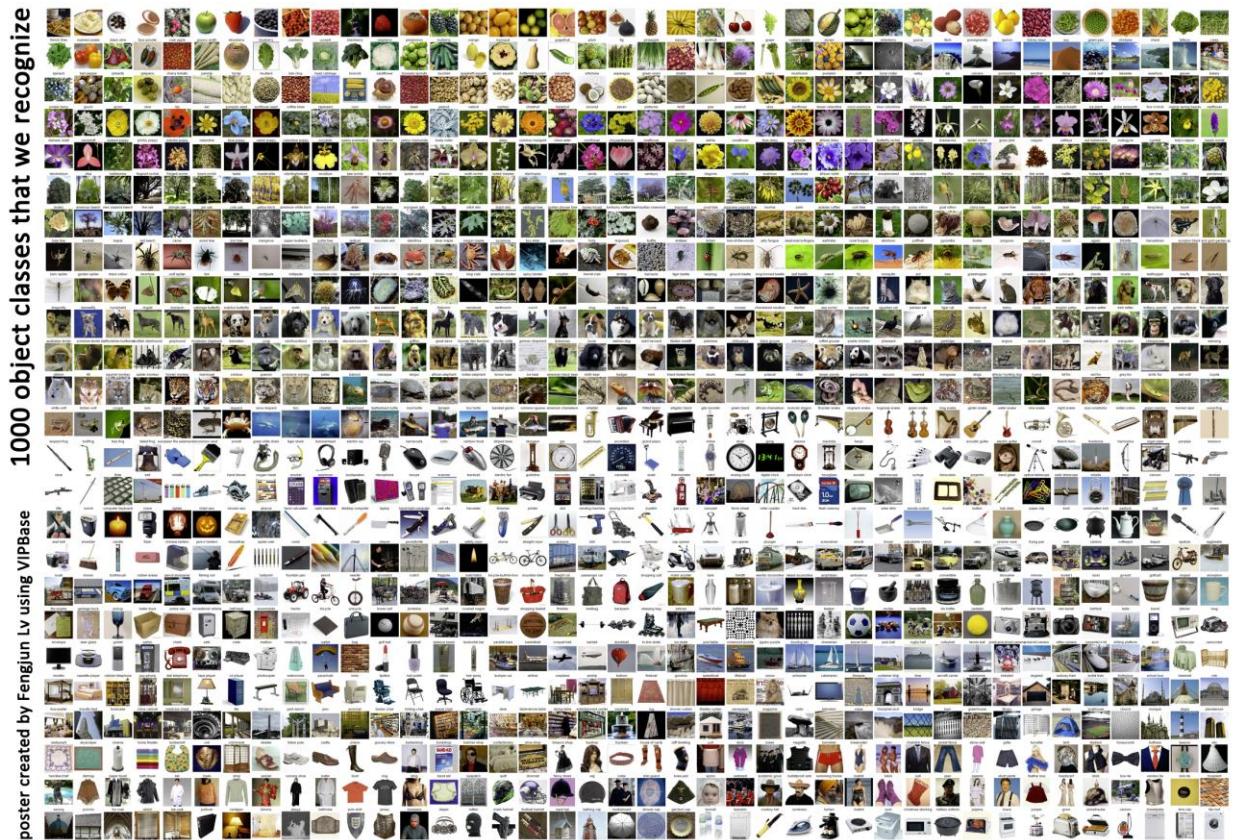
CNN for Image Classification on ImageNet

- Krizhevsky, Sutskever, and Hinton, NIPS 2012
- Trained on one million images of 1000 categories collected from the web with two GPU. 2GB RAM on each GPU. 5GB of system memory
- Training lasts for one week
- Google and Baidu announced their new visual search engines with the same technology six months after that
- Google observed that the accuracy of their visual search engine was doubled

Rank	Name	Error rate	Description
1	U. Toronto	0.15315	Deep learning
2	U. Tokyo	0.26172	Hand-crafted features and learning models.
3	U. Oxford	0.26979	Bottleneck.
4	Xerox/INRIA	0.27058	

ImageNet

- 1000 classes
- 1.28 million training images
- 50k validation images
- 100k test images



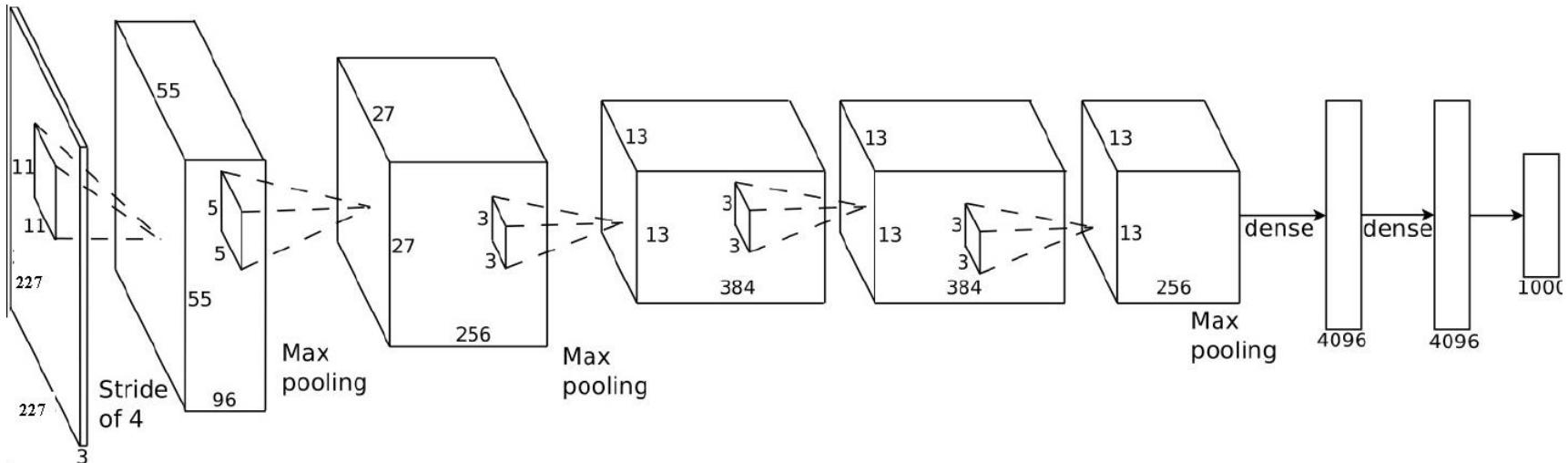
ImageNet

Model	Layers	Top-5 error	Year
AlexNet	8	16.4%	2012
Clarifai	8	11.7%	2013
VGG	19	7.3%	2014
GoogleNet	22	6.7%	2014
ResNet	152	3.57%	2015

Example-AlexNet

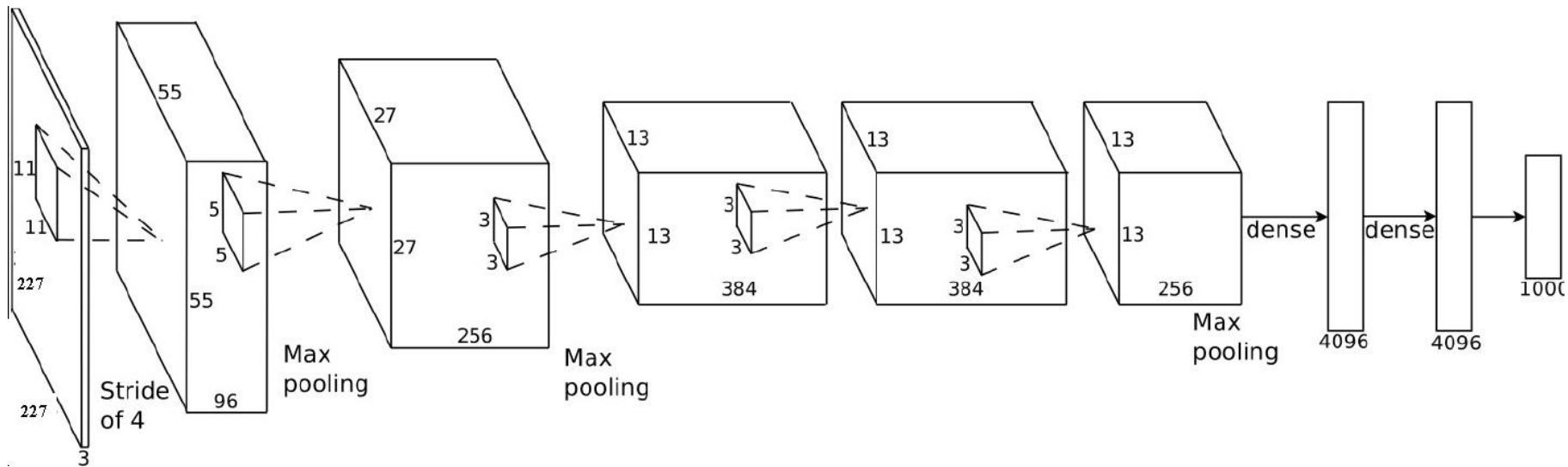
Review the architecture of AlexNet

- 5 convolutional layers and 2 fully connected layers for learning features.
- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000
- 650000 neurons, 60000000 parameters, and 630000000 connections



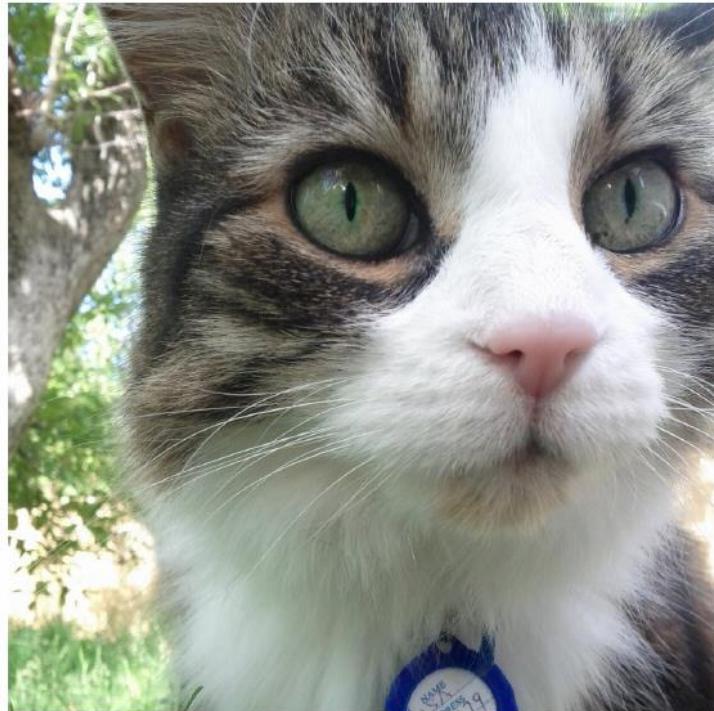
Example-AlexNet

- The first time deep model is shown to be effective on large scale computer vision task.
- The first time a very large scale deep model is adopted.
- GPU is shown to be very effective on this large deep model.

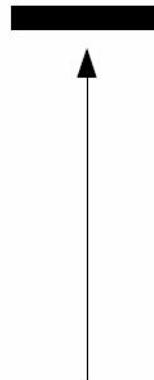


Technical details

- Normalize the input by subtracting the mean image on the training set



An input image (256x256)



Minus sign

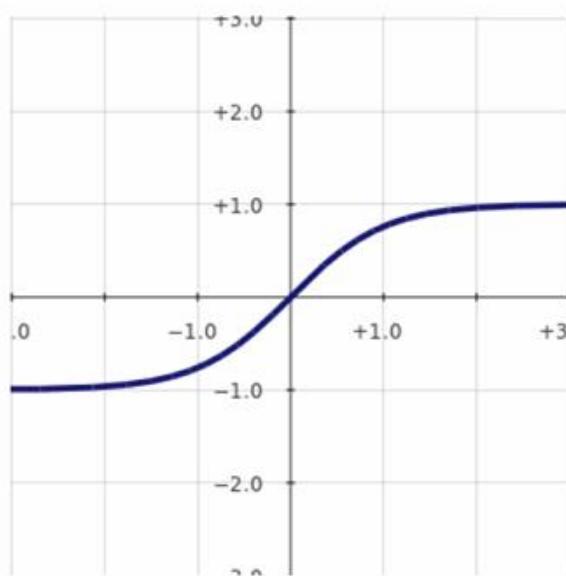


The mean input image

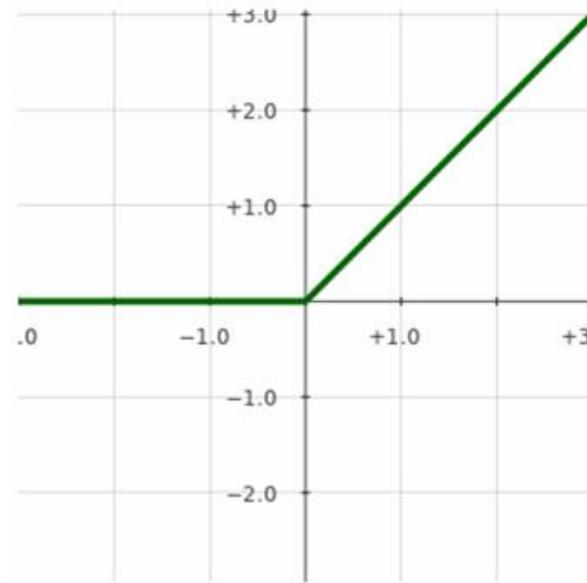
Technical details

- Choice of activation function

$$f(x) = \tanh(x)$$



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

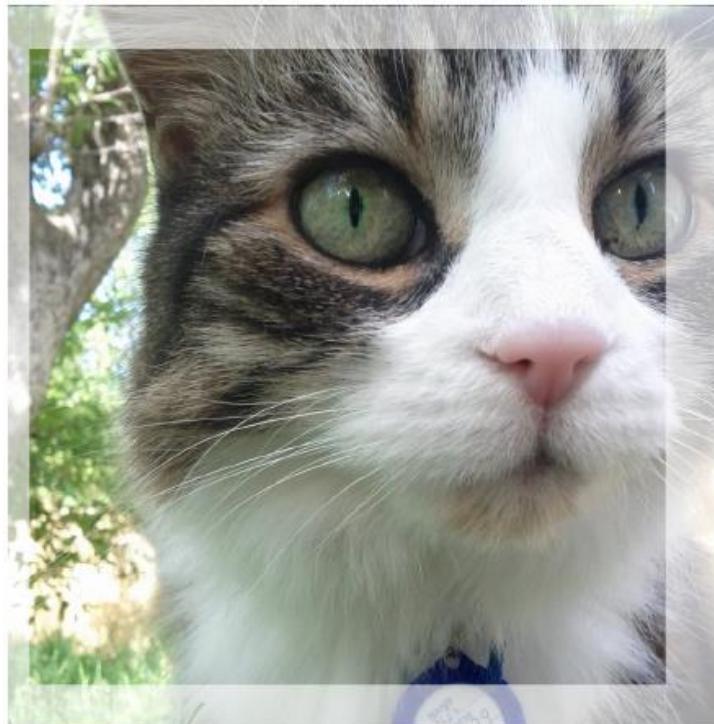


Very bad (slow to train)

Very good (quick to train)

Technical details

- Data augmentation
 - The neural net has 60M real-valued parameters and 650,000 neurons
 - It overfits a lot. 224 x 224 image regions are randomly extracted from 256 images, and also their horizontal reflections



Technical details

- Dropout
 - Independently set each hidden unit activity to zero with 0.5 probability
 - Do this in the two globally-connected hidden layers at the net's output

A hidden layer's activity on a given training image



A hidden unit
turned off by
dropout

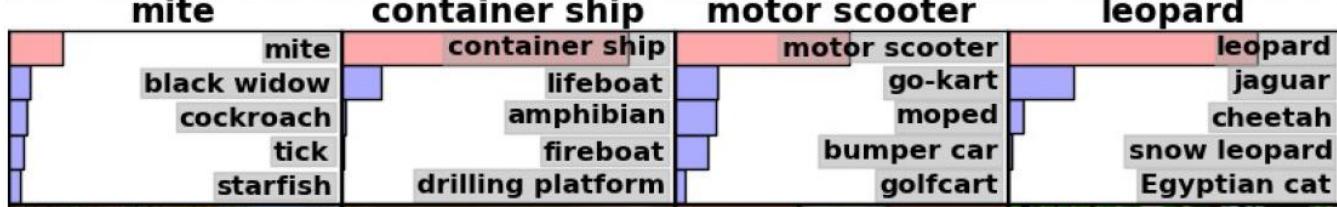


A hidden unit
unchanged

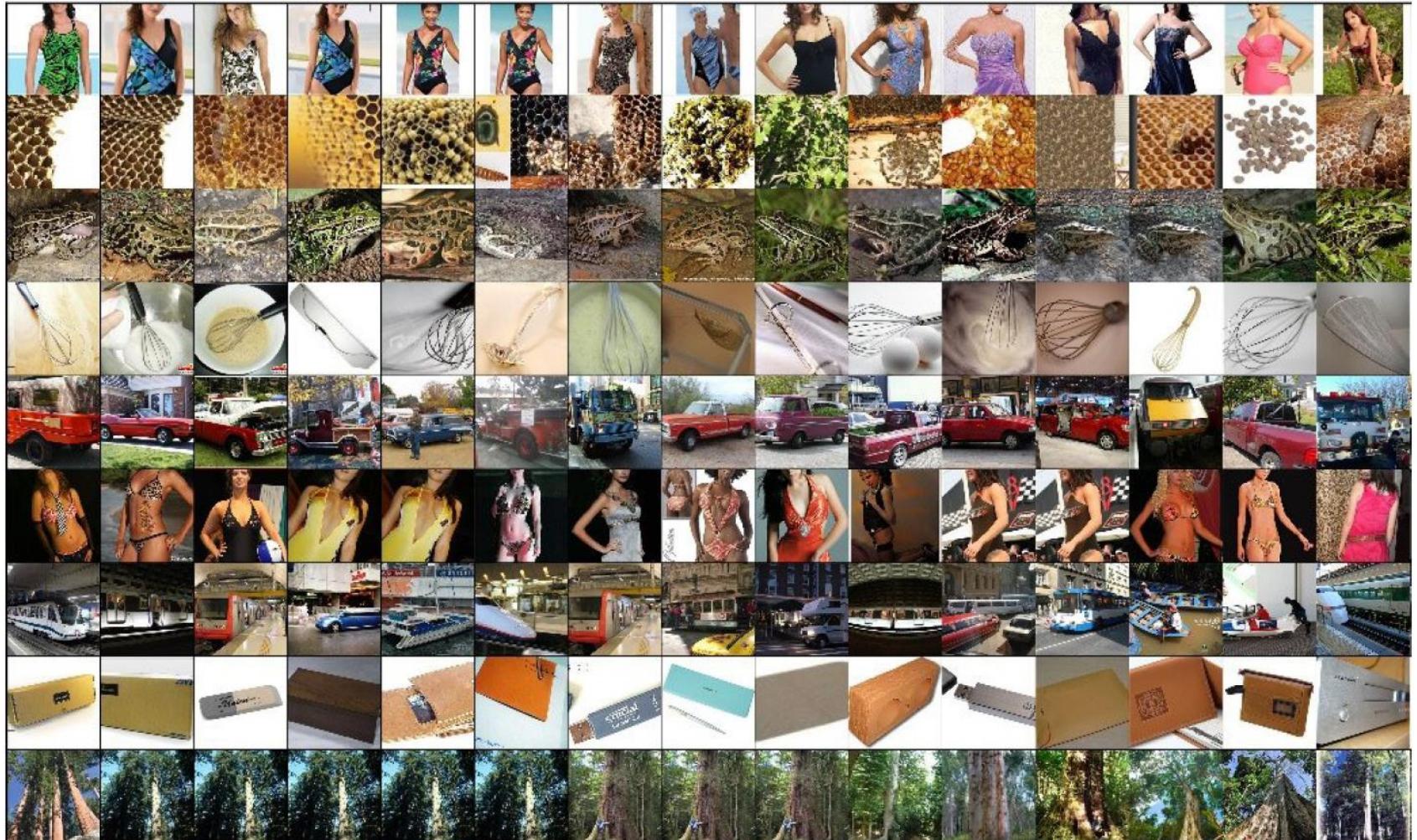
96 learned low-level filters



Classification result



Top hidden layer can be used as feature for retrieval

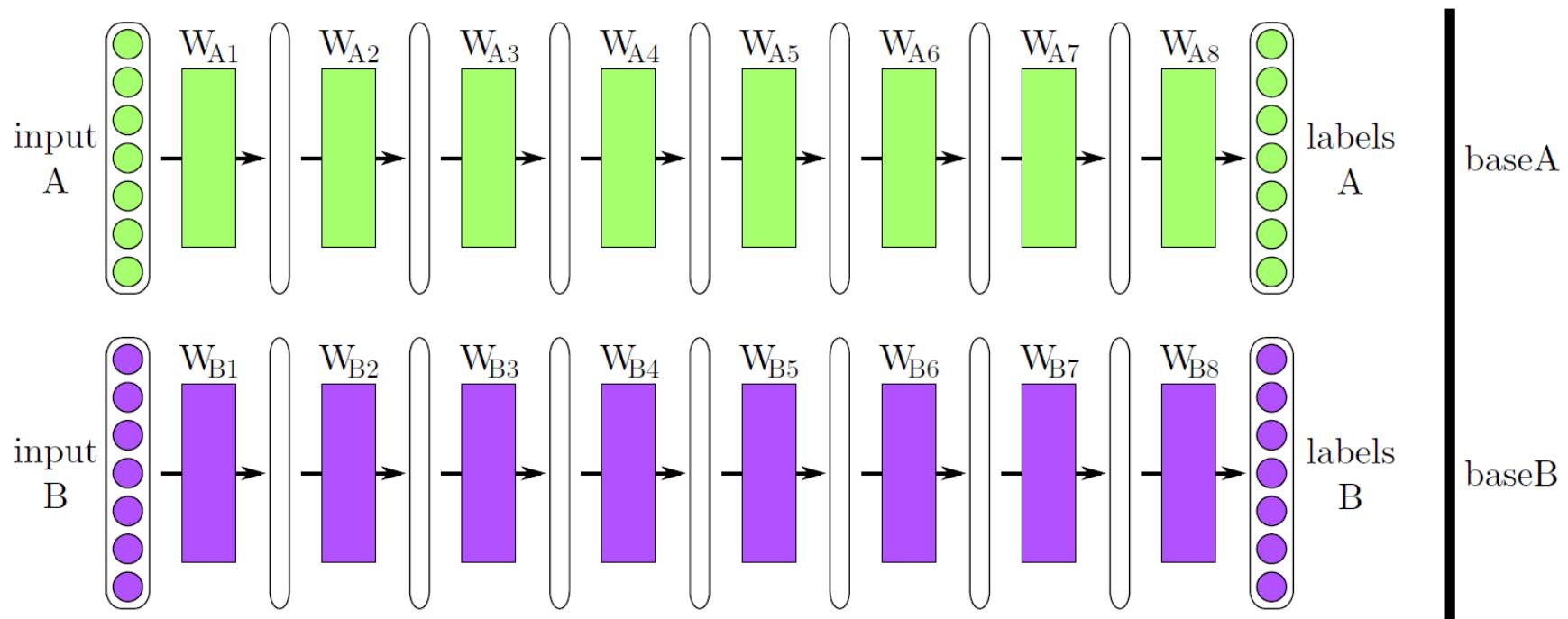


How transferable are features in CNN networks?

- (Yosinski et al. NIPS'14) investigate transferability of features by CNNs
- The transferability of features by CNN is affected by
 - Higher layer neurons are more specific to original tasks
 - Layers within a CNN network might be fragilely co-adapted
- Initializing with transferred features can improve generalization after substantial fine-tuning on a new task

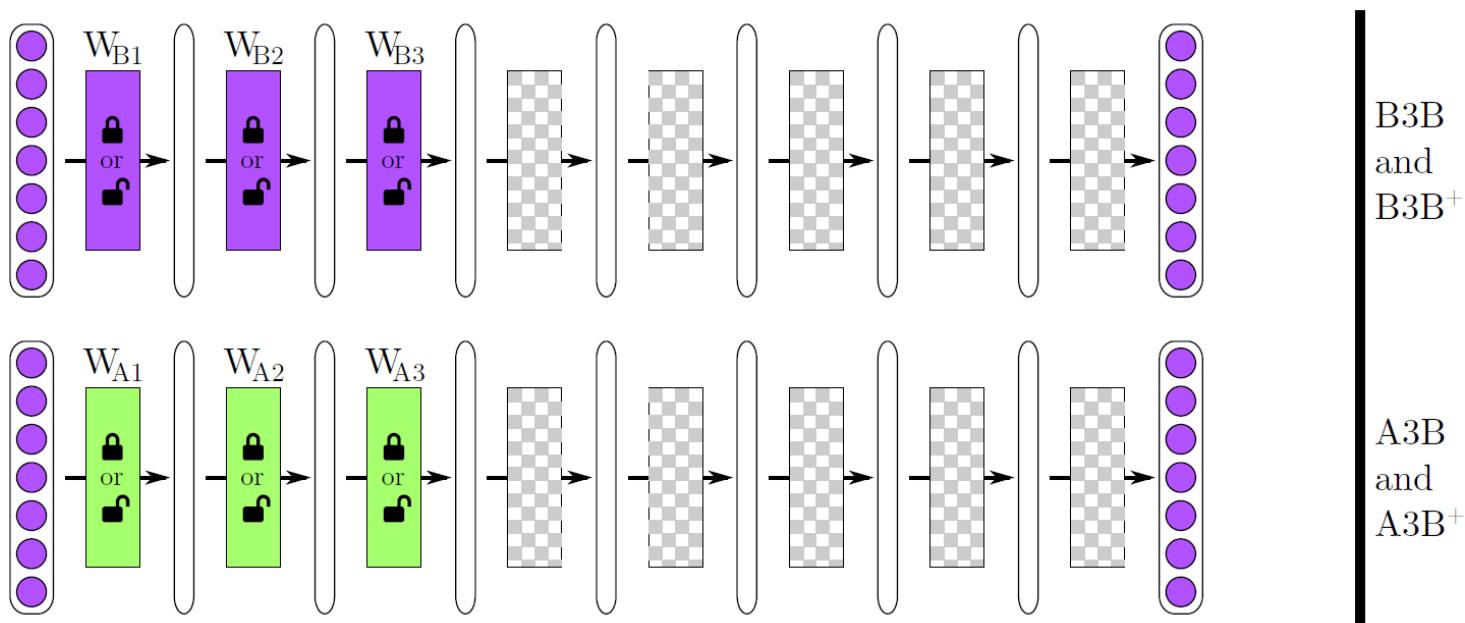
Base tasks

- ImageNet are divided into two groups of 500 classes, A and B
- Two 8-layer AlexNets, baseA and baseB, are trained on the two groups, respectively



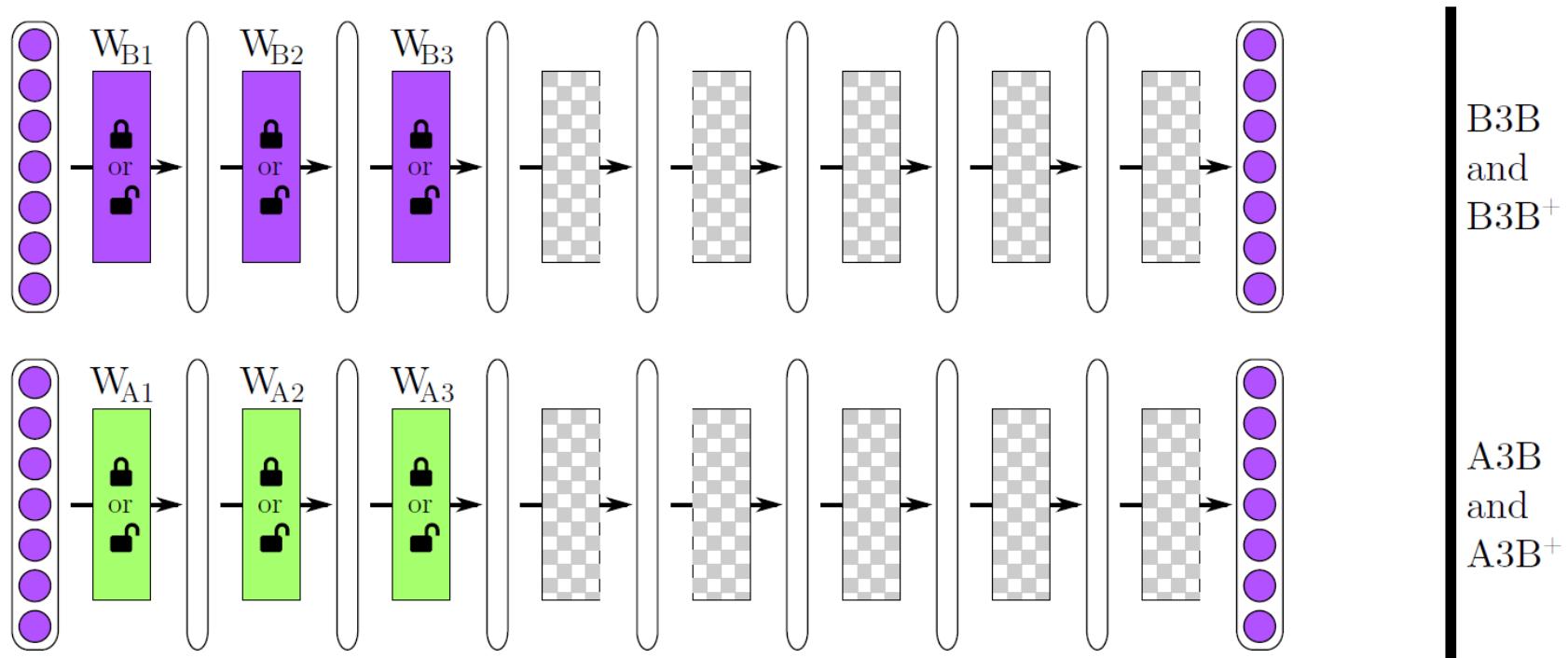
Transfer and selffer networks

- A *selffer* network BnB: the first n layers are copied from baseB and frozen. The other higher layers are initialized randomly and trained on dataset B. This is the control for transfer network
- A *transfer* network AnB: the first n layers are copied from baseA and frozen. The other higher layers are initialized randomly and trained toward dataset B

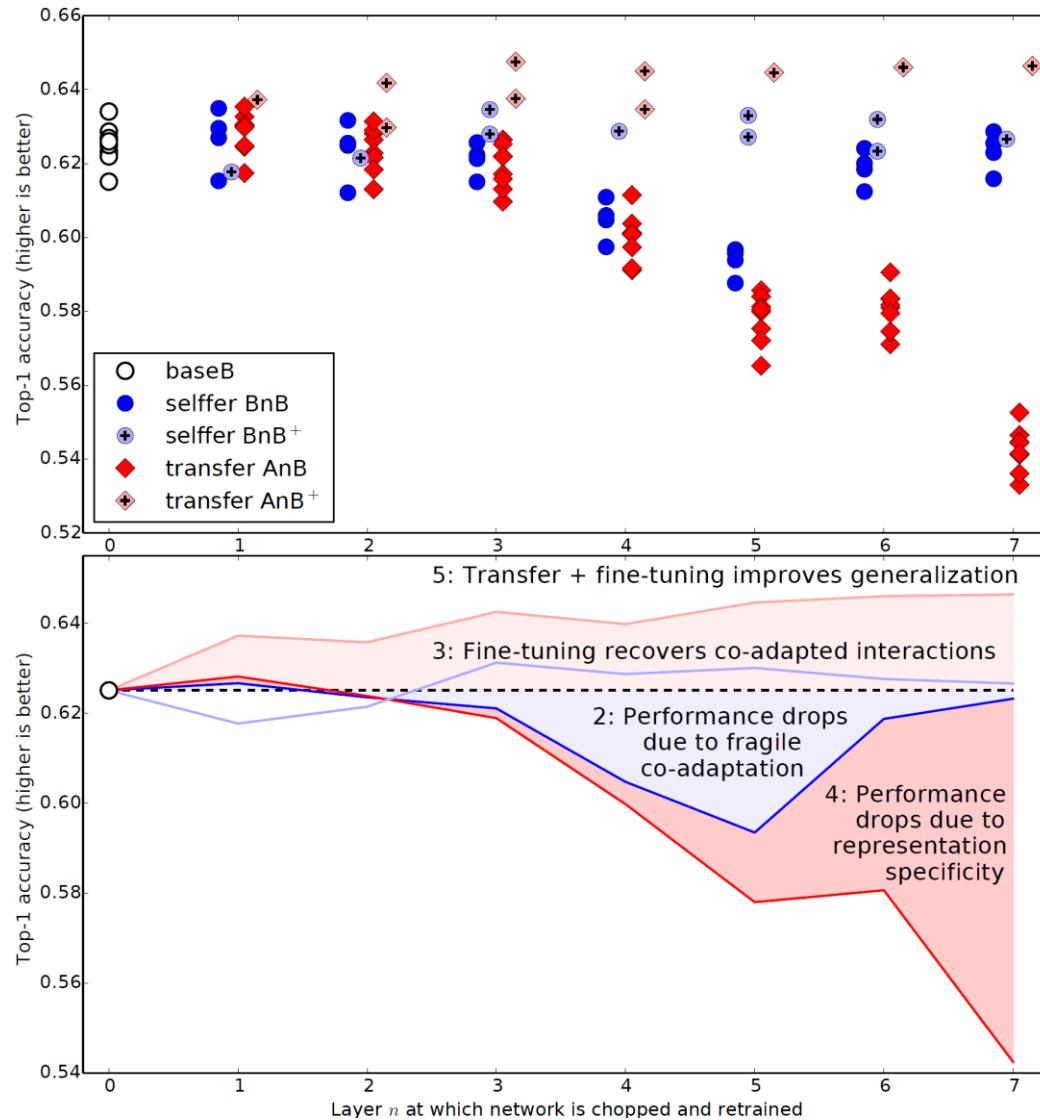


Transfer and selffer networks (cont'd)

- A selffer network BnB+: just like BnB, but where all layers learn
- A transfer network AnB+: just like AnB, but where all layers learn

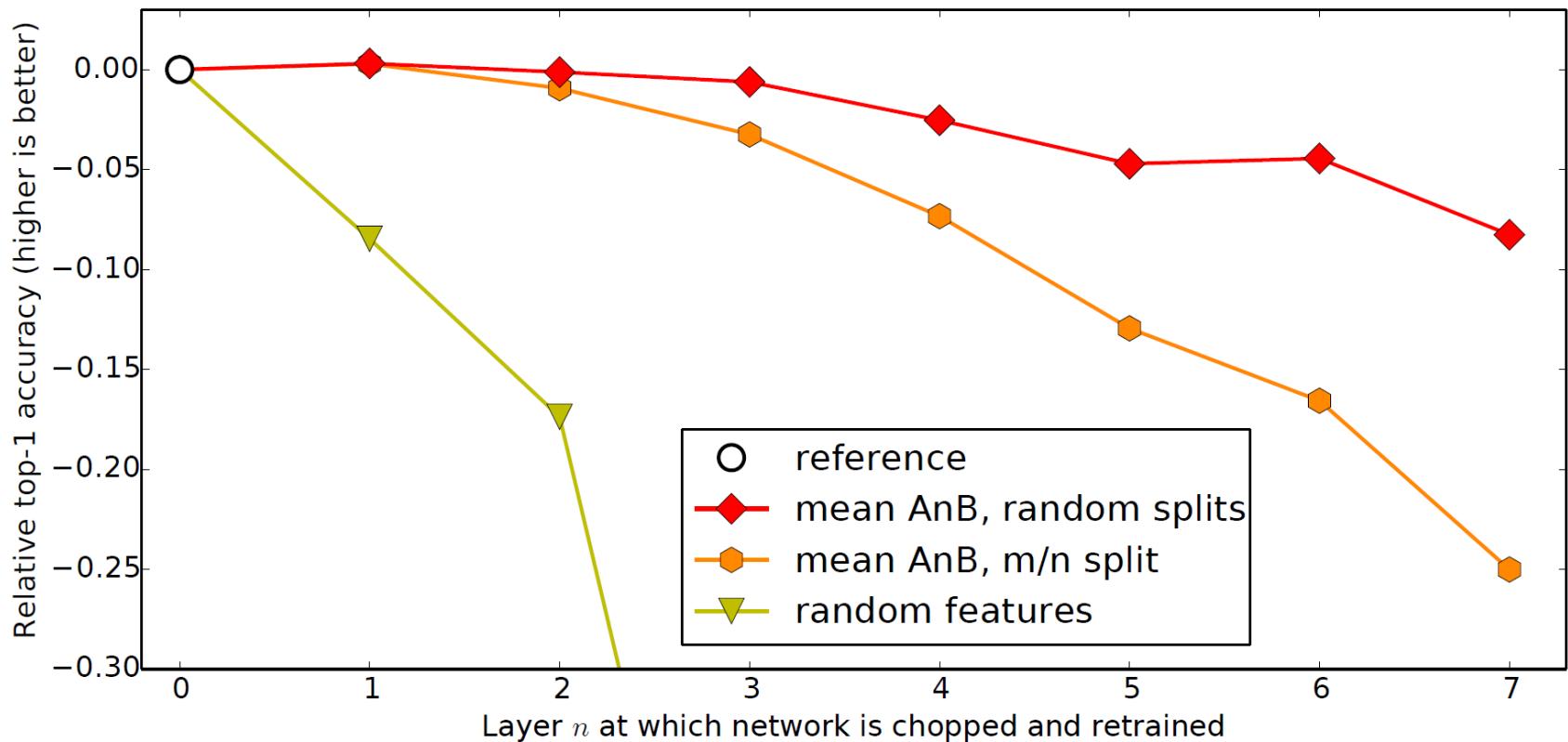


Results



Dissimilar datasets

- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases



Investigate components of CNNs

- Kernel size
- Kernel (channel) number
- Stride
- Dimensionality of fully connected layers
- Data augmentation

Investigate components of CNNs (cont'd)

- (Chatfield et al. BMVC'14) pre-train on ImageNet and fine-tune on PASCAL VOC 2007
- Different architectures
 - mAP: CNN-S > (marginally) CNN-M > (~%2.5) CNN-F
- Different data augmentation
 - No augmentation
 - Flipping (almost no improvement)
 - Smaller dimension downsized to 256, cropping 224 x 224 patches from the center and 4 corners, flipping (~ 3% improvement)

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8	
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1	256x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max	Fast similar to AlexNet
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x2 pool	4096 drop-out	4096 drop-out	1000 soft-max	Medium similar to Clarifai model
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1	512x3x3 st. 1, pad 1 x3 pool	4096 drop-out	4096 drop-out	1000 soft-max	Slow similar to OverFeat Accurate model

Investigate components of CNNs (cont'd)

- Gray-scale vs. color ($\sim 3\%$ drop)
- Decrease the number of nodes in FC7
 - to 2048 (surprisingly, marginally better)
 - to 1024 (marginally better)
 - to 128 ($\sim 2\%$ drop but 32x smaller feature)
- Change the softmax regression loss to ranking hinge loss
 - $w_c \phi(I_{pos}) > w_c \phi(I_{neg}) + 1 - \xi$ (ξ is a slack variable)
 - $\sim 2.7\%$ improvement
 - Note, L2 normalising features account for $\sim 5\%$ of accuracy for VOC 2007
- On ILSVRC-2012, the CNN-S achieved a top-5 error rate of 13.1%
 - CNN-F: 16.7%
 - CNN-M: 13.7%
 - AlexNet: 17%

Next Lecture

- Recurrent Neural Network

Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course

Questions?

Thank You !

