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

2023

"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

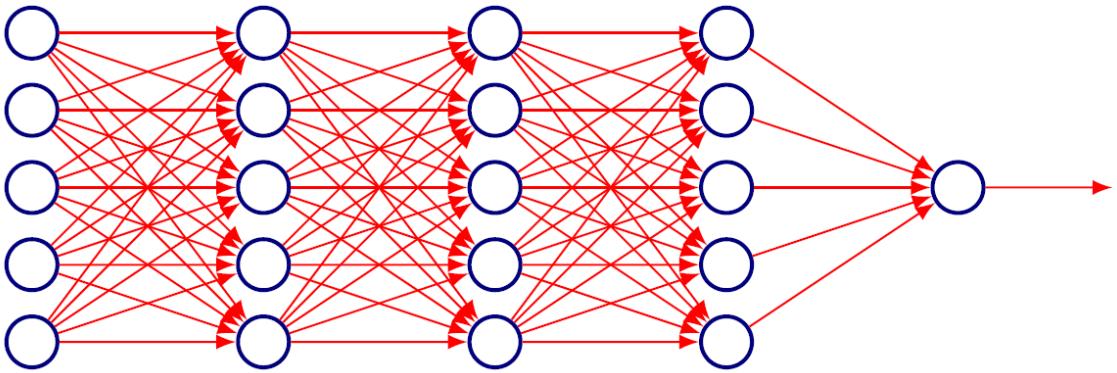
1. Course Review
2. Basic Operations
3. Major Architectures
4. CNN for Image Classification
5. Know More about CNN

Review: Feedforward Neural Network

- Goal: Approximate some unknown ideal function $f^* : \mathcal{X} \rightarrow \mathcal{Y}$
- Ideal classier: $y = f^*(\mathbf{x})$ 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!)



$$f(\mathbf{x}) = f^{(3)}(f^{(2)}|(f^{(1)}(\mathbf{x})))$$

Review: Basic Components and Examples

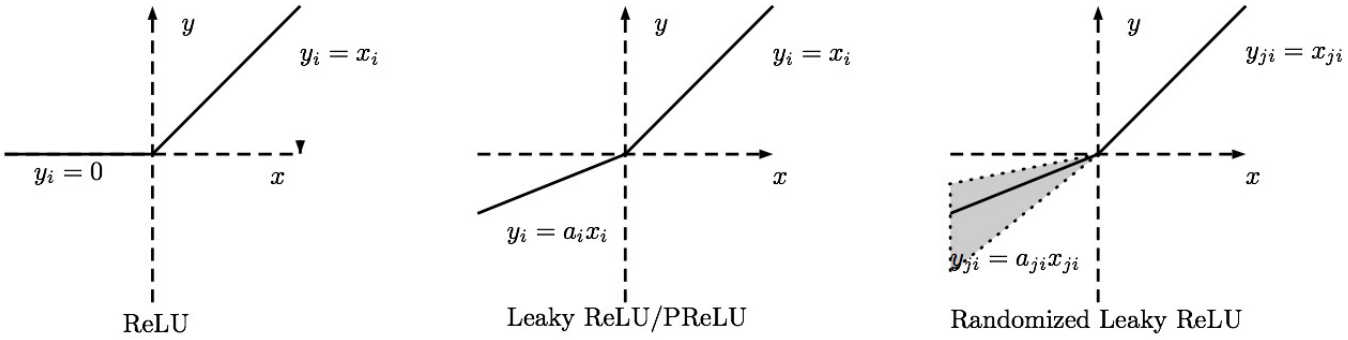
- Cost Functions
  - Cross entropy

$$J(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{data}} \log p_{model}(\mathbf{y}|\mathbf{x})$$

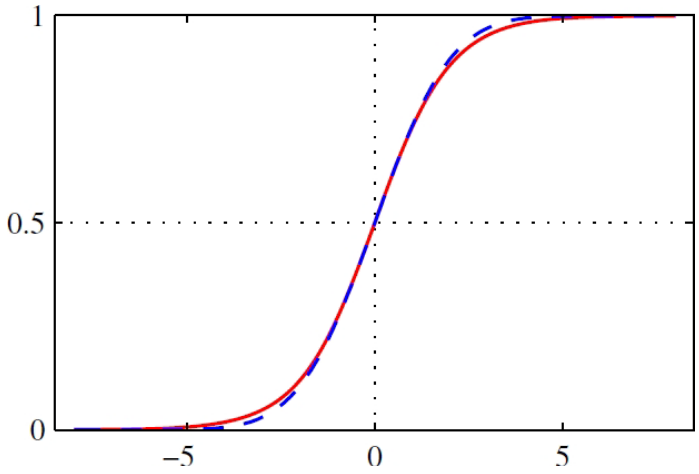
- Output Units
  - Linear Units: $\hat{y} = W^T h + b$, often used to produce the mean of a conditional Gaussian distribution
  - Sigmoid Units: $\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b)$, predict a binary variable y
  - Softmax Units: $\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, often applied to the classification task

Review: Basic Components and Examples

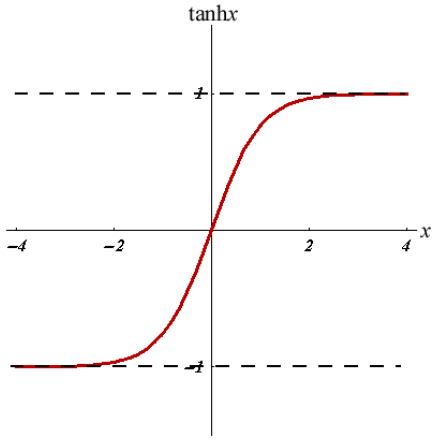
- Hidden Units



ReLU Series



Sigmoid



Tanh

Review: Architecture Design

- **MLPs approximate functions**
  - MLPs can compose universal boolean functions
  - MLPs can compose universal classifiers
  - MLPs can compose universal approximators

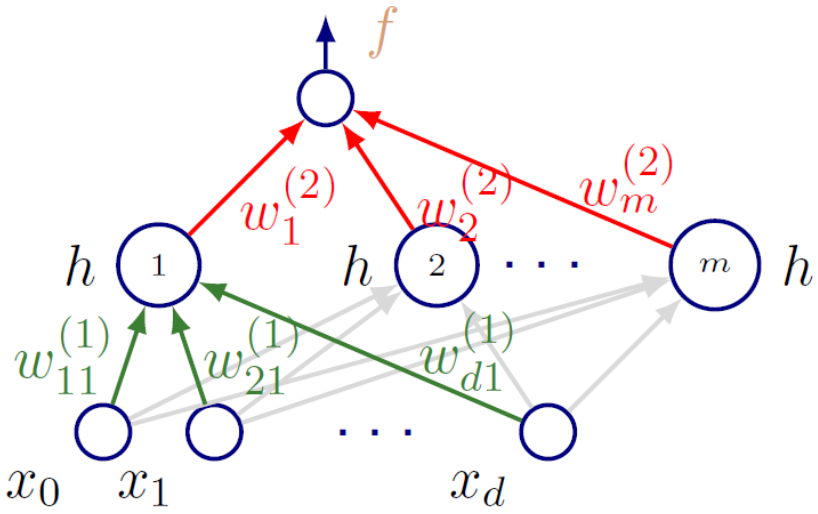
- Regardless of function we are trying to learn, we know a large MLP can represent this function
- But not guaranteed that our training algorithm will be able to learn that function
- A one-hidden-layer neural network will required infinite hidden neurons
- There is a trade-off between width and depth

Review: Backpropagation

- Output: $\mathrm{f}(a) = a$
- Hidden

$$h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}},$$

$$h'(a) = 1 - h(a)^2.$$



- Given example x, feed-forward inputs:

input to hidden: $a_j = \sum_{i=0}^{d} w_{ij}^{(1)} x_i,$

hidden output: $z_j = \tanh(a_j),$

net output: $\hat{y} = a = \sum_{j=0}^{m} w_j^{(2)} z_j.$

Backpropagation: example

$$a_j = \sum_{i=0}^{d} w_{ij}^{(1)} x_i, \quad z_j = \tanh(a_j), \quad \hat{y} = a = \sum_{j=0}^{m} w_j^{(2)} z_j. \quad h'(a) = 1 - h(a)^2.$$

- Error on example x: $L = \frac{1}{2}(y - \hat{y})^2$.
- Output unit: $\delta = \frac{\partial L}{\partial a} = \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$
- Derivate a_j , $\delta_j = \frac{\partial L}{\partial a_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_j} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial a_j}$:

$$\delta_j = (1 - z_j)^2 w_j^{(2)} \delta$$

- Derivatives w.r.t. weights:

$$\frac{\partial L}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}^{(1)}} = \delta_j \cdot x_i \qquad \frac{\partial L}{\partial w_j^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j^{(2)}} = (y - \hat{y}) \cdot z_j = \delta \cdot z_j$$

- Update weights: $\omega_j^{(2)} \leftarrow \omega_j^{(2)} - \eta \delta z_j$ and $\omega_{ij}^{(1)} \leftarrow \omega_{ij}^{(1)} - \eta \delta_j x_j$.
- η is called the weight decay

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 redundance
  - 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

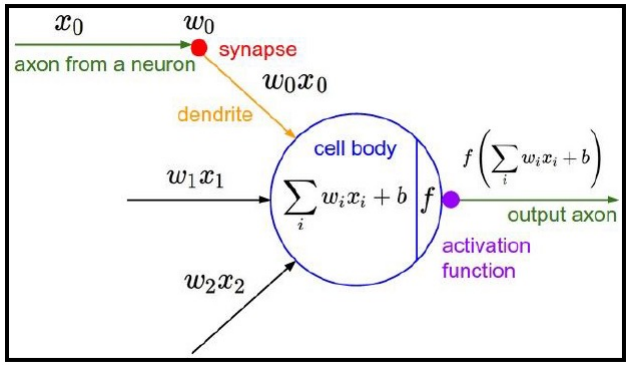
1. Course Review
2. Basic Operations
3. Major Architectures
4. CNN for Image Classification
5. Know More about CNN

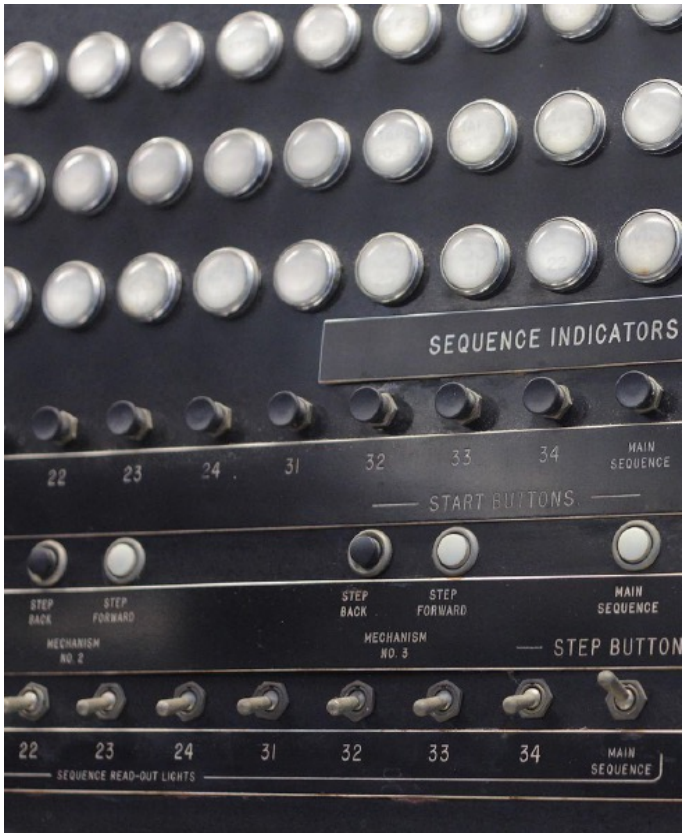
Historical Background

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





Frank Rosenblatt, ~1957: Perceptron

Historical Background

Hubel & Wiesel

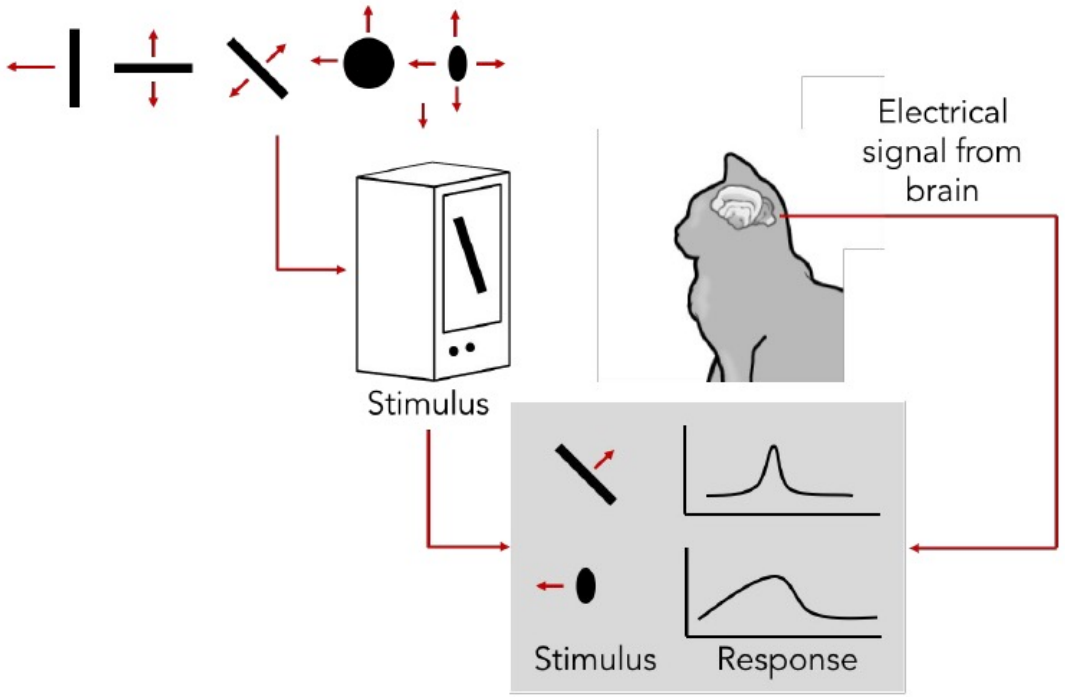
1959

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?

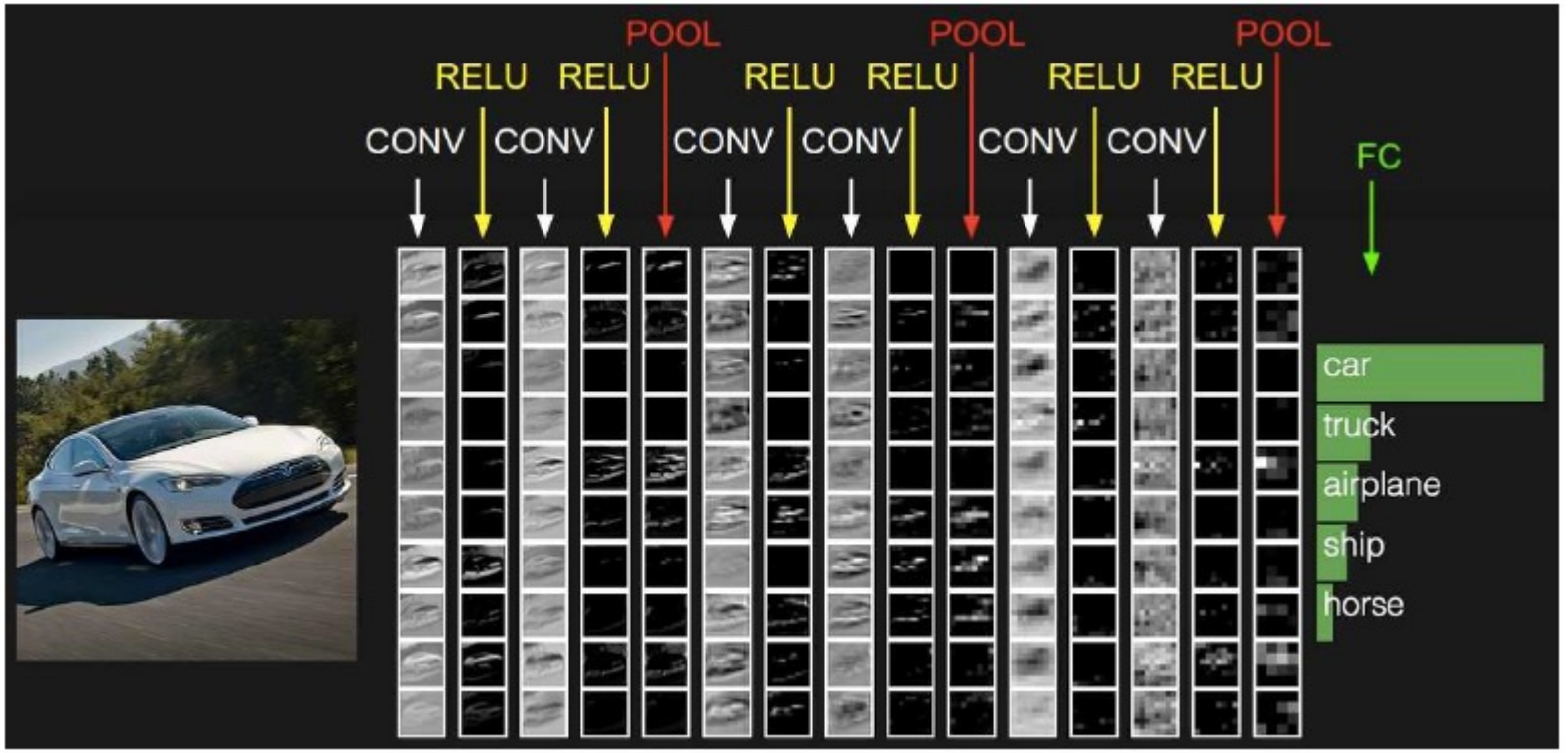


Figure: Andrej Karpathy

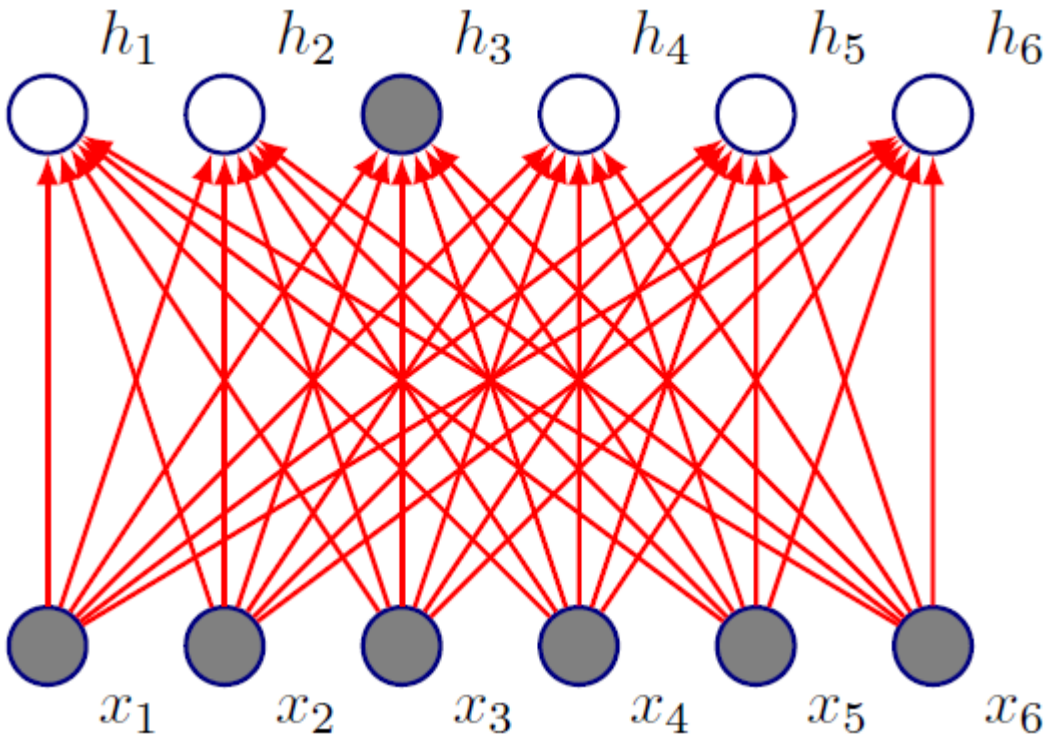
- **Convolution Layers**
- **Pooling Layers**
- **Fully-Connected Layers**
- **Non-linear Layers, e.g., ReLU**
- **Other Layers, e.g., RNN Layer, Your Customized Layers …**

Why do Convolutional Neural Network (CNNs) make sense ?

Convolutions: Motivation

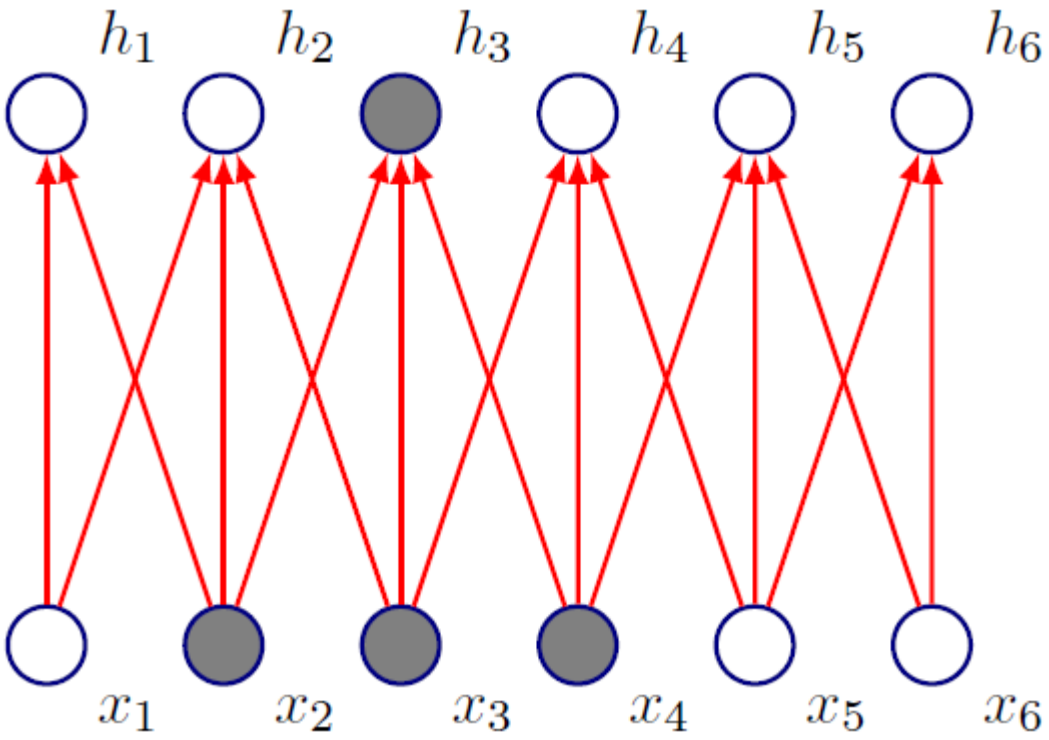
- **Convolution leverages four ideas that can help ML systems:**
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
  - Ability to work with inputs of variable size
- **Sparse Interactions**
  - Plain Vanilla NN ($y \in \mathbf{R}^n, x \in \mathbf{R}^m$): Need matrix multiplication $y = \mathbf{W}x$ 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

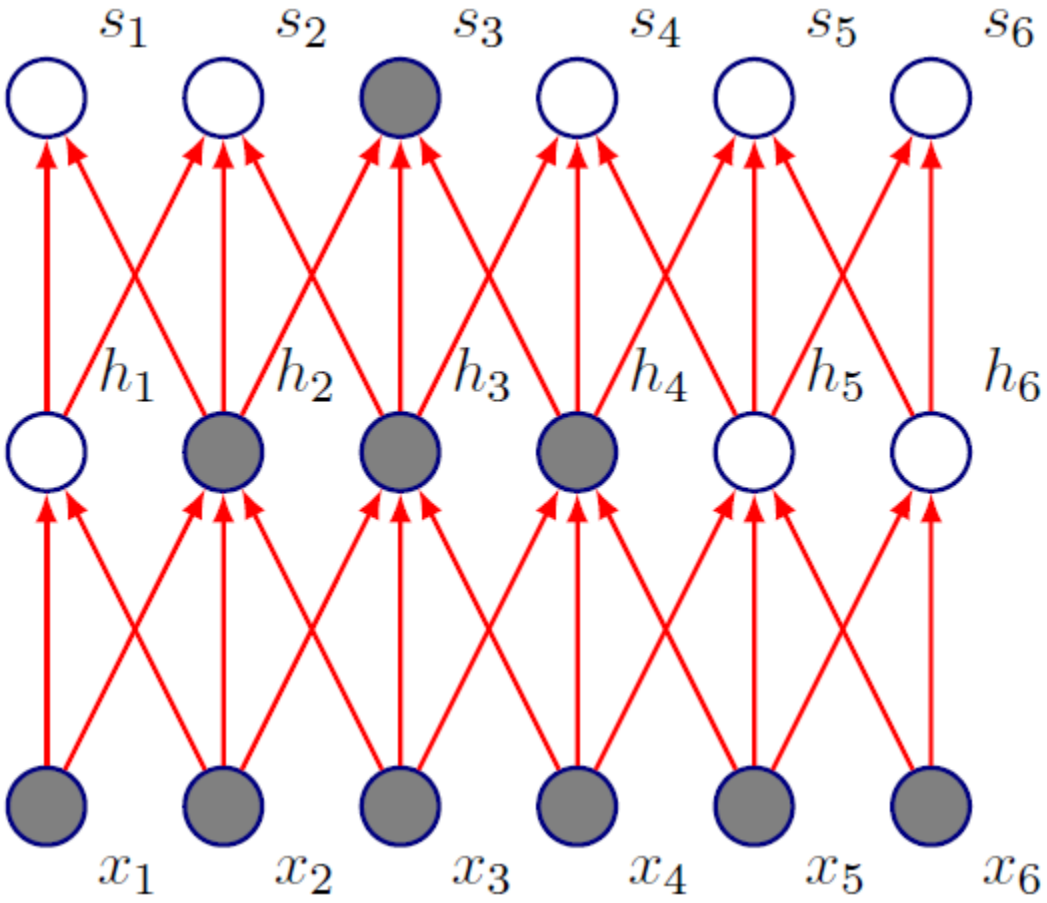
Motivation: Sparse Connectivity



Kernel of size 3, moved with stride of 1

h_3 only depends on x_2 , x_3 , x_4

Motivation: Sparse Connectivity



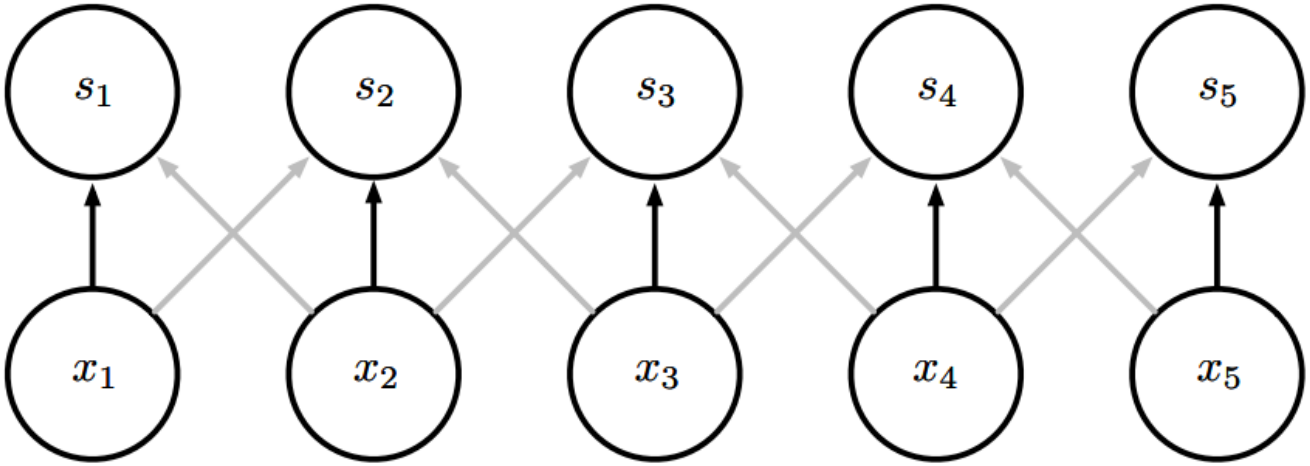
Connections in CNNs are sparse, but units in deeper layers are connected to all of the input (larger receptive field sizes)

Motivation: Parameter Sharing

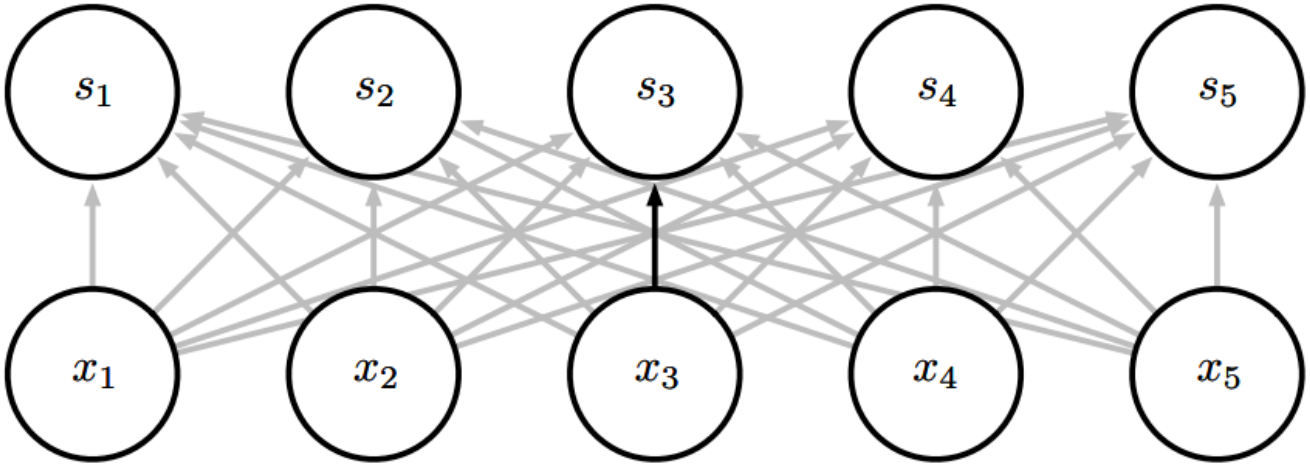
- Plain vanilla NN: Each element of **W** is used exactly once to compute output of a layer
- In convolutional neural networks, parameters are tied: weight applied to one input is tied to value of a weight applied elsewhere
- Same kernel is used throughout the image, so instead learning a parameter for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

Motivation: Parameter Sharing

Convolution shares the same parameters across all spatial locations



Traditional matrix multiplication does not share any parameters



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

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

Vs

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

- 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

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

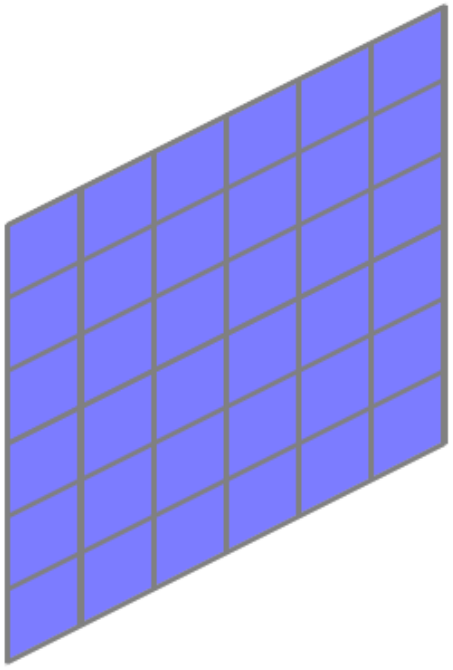
Motivation: Equivariance

- 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

Convolution

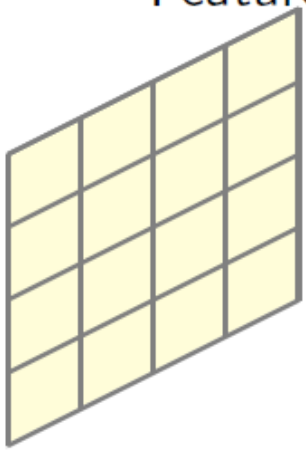


Grayscale Image

Kernel

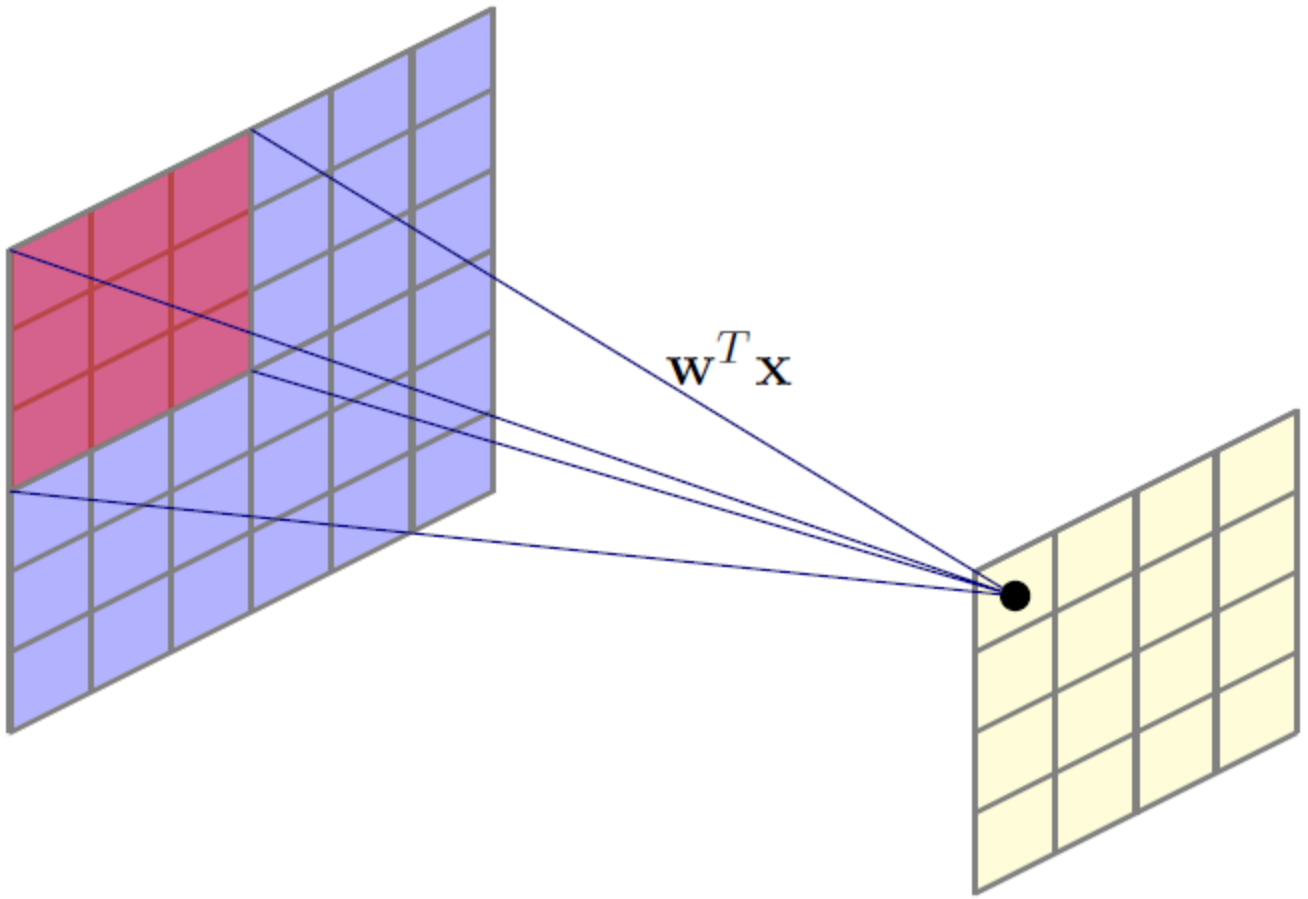
w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

Feature Map

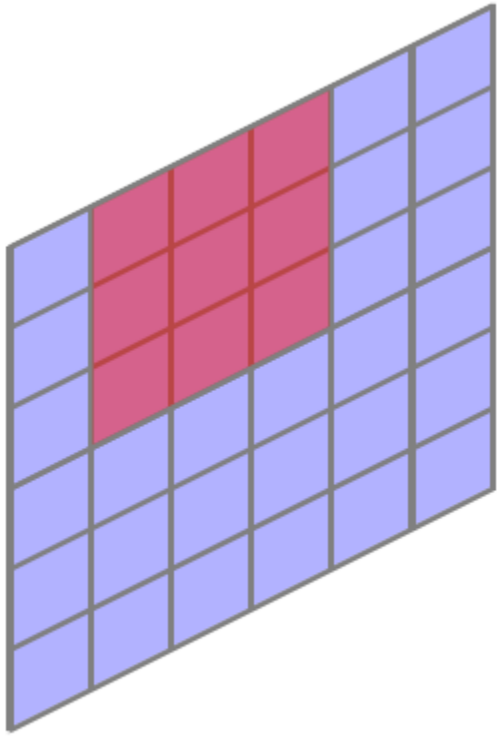


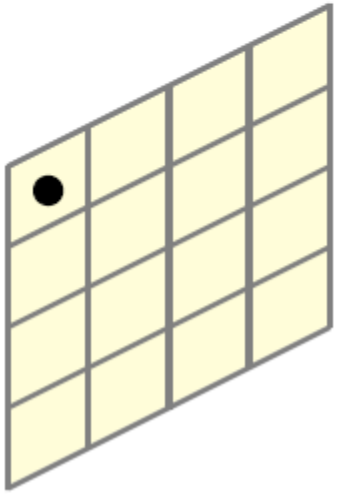
Convolve image with kernel having weights w (learned by backpropagation)

Convolution

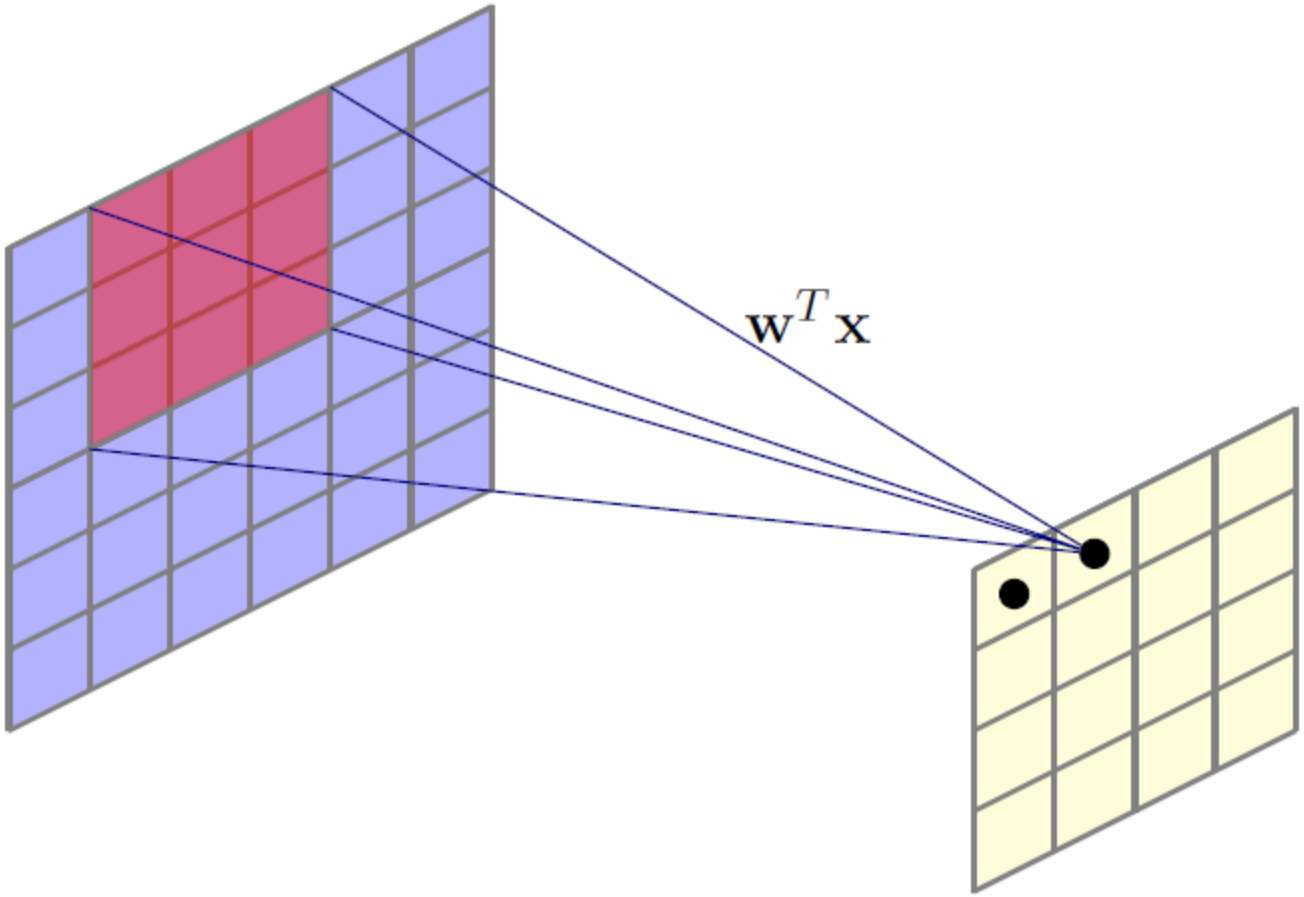


Convolution

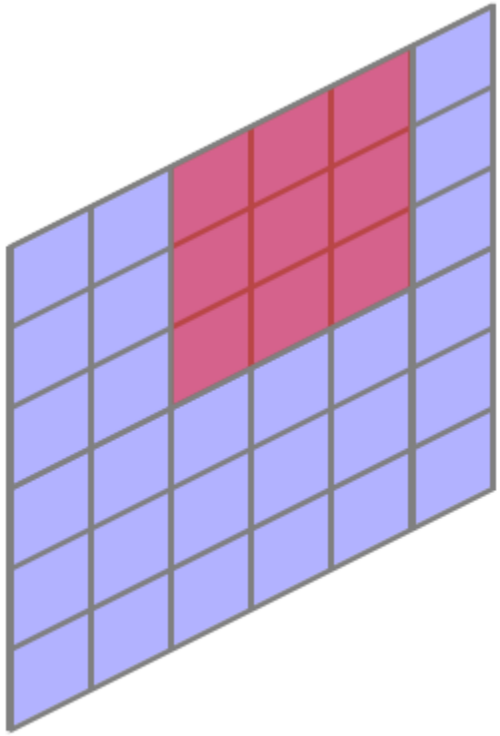


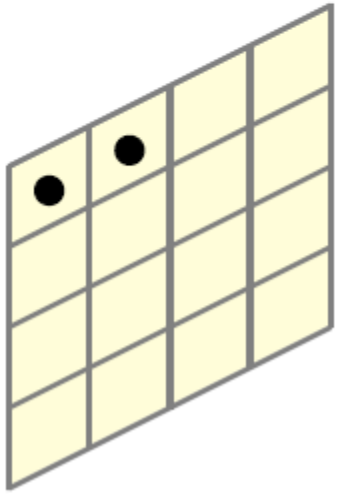


Convolution

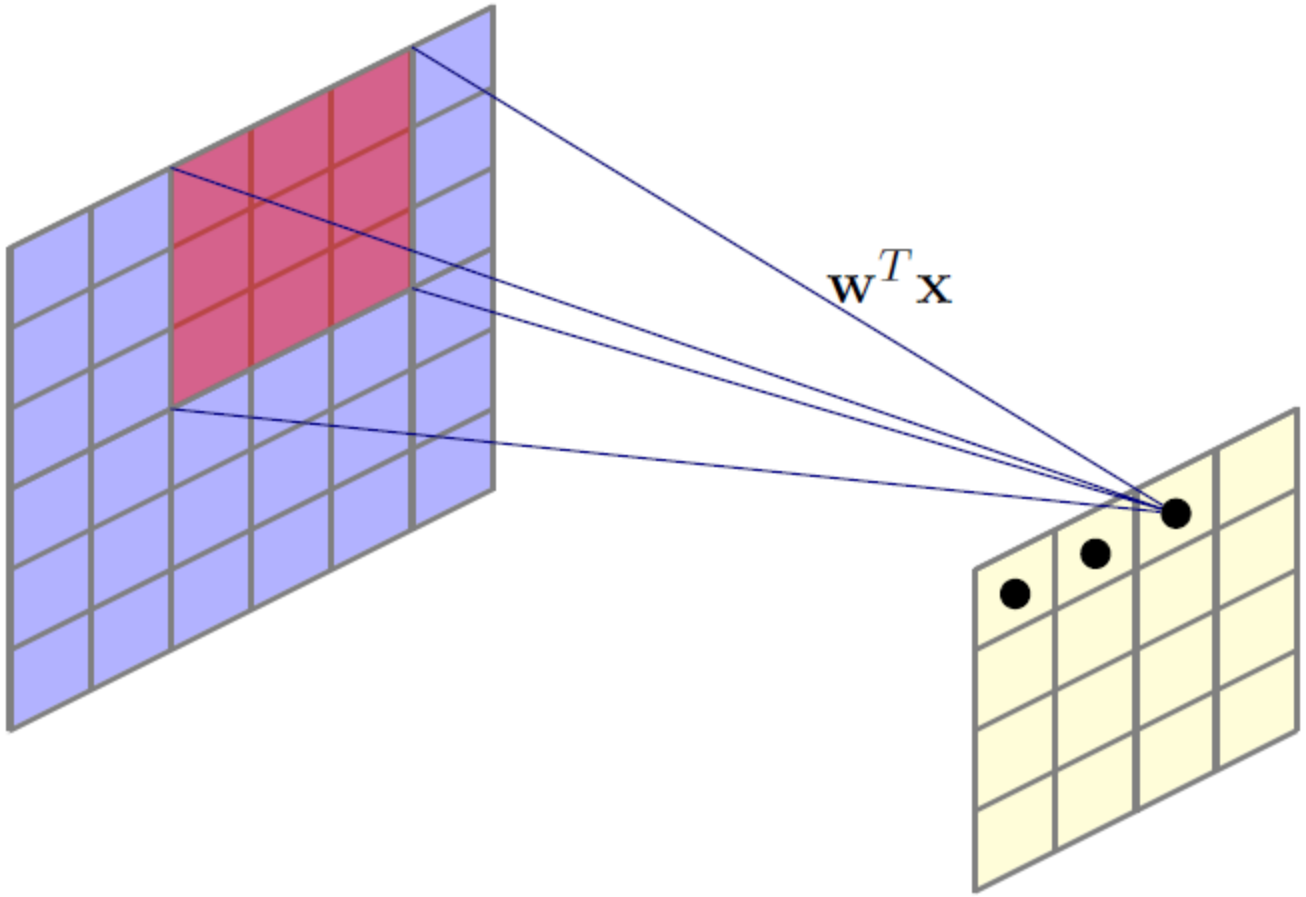


Convolution

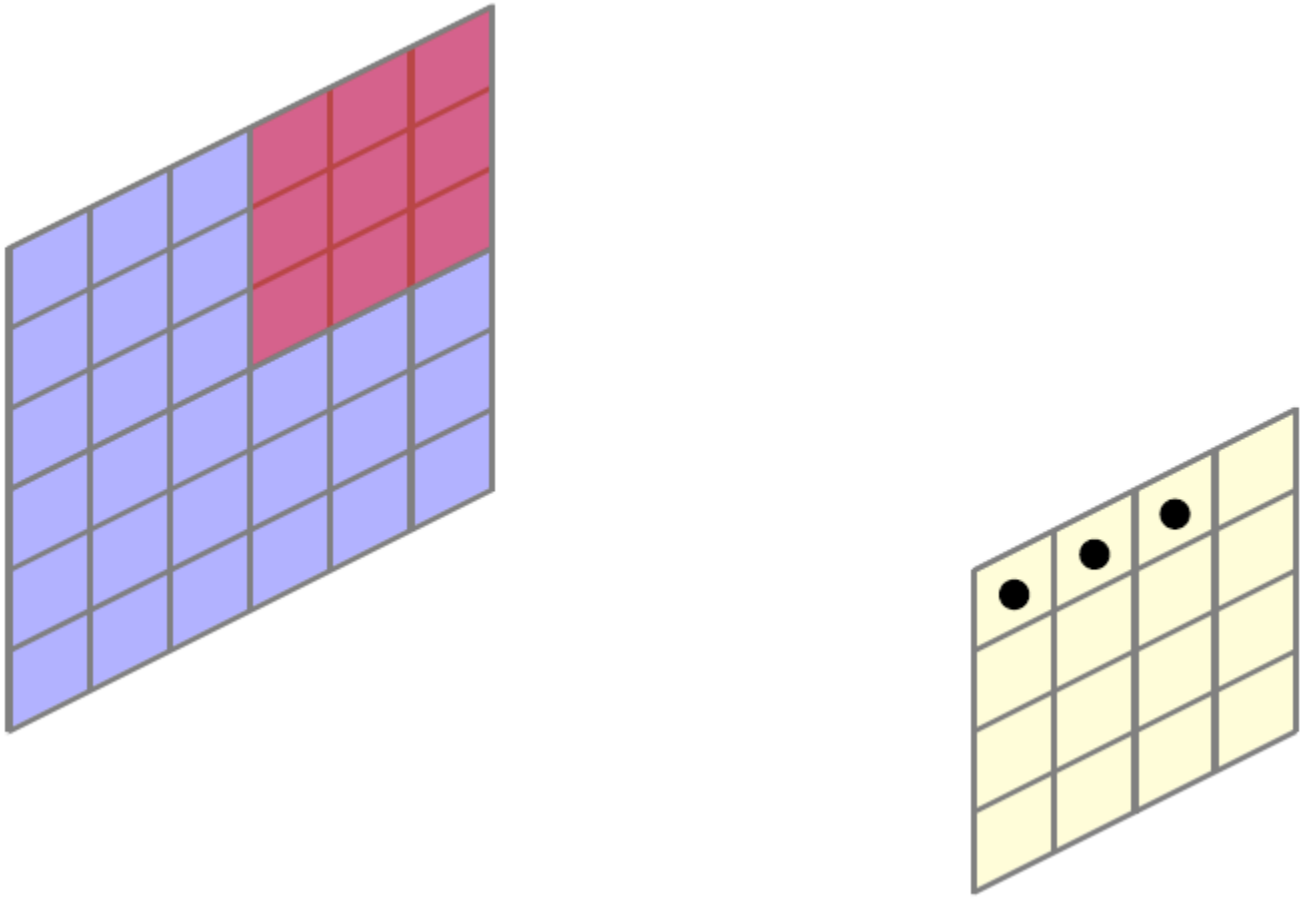




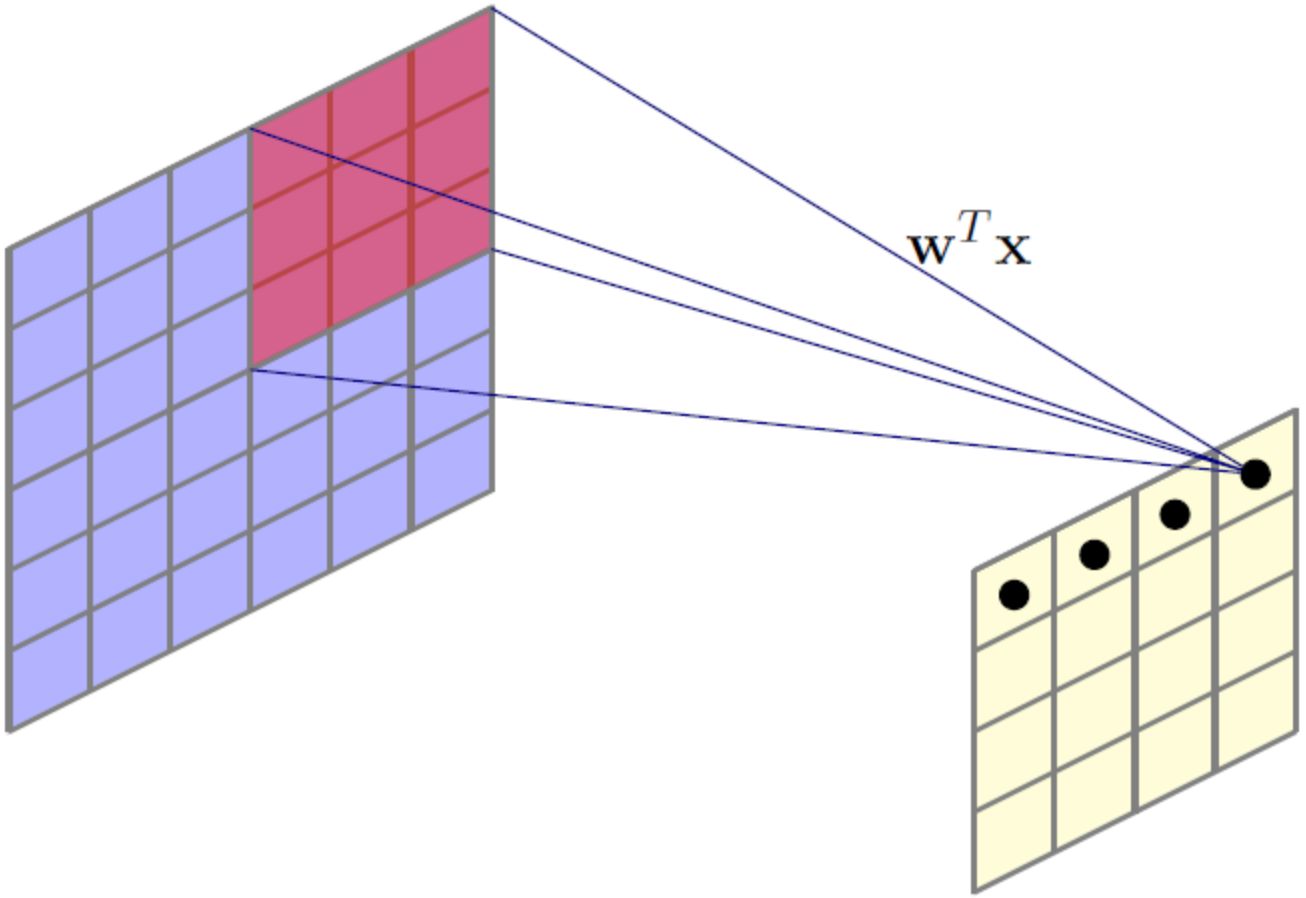
Convolution



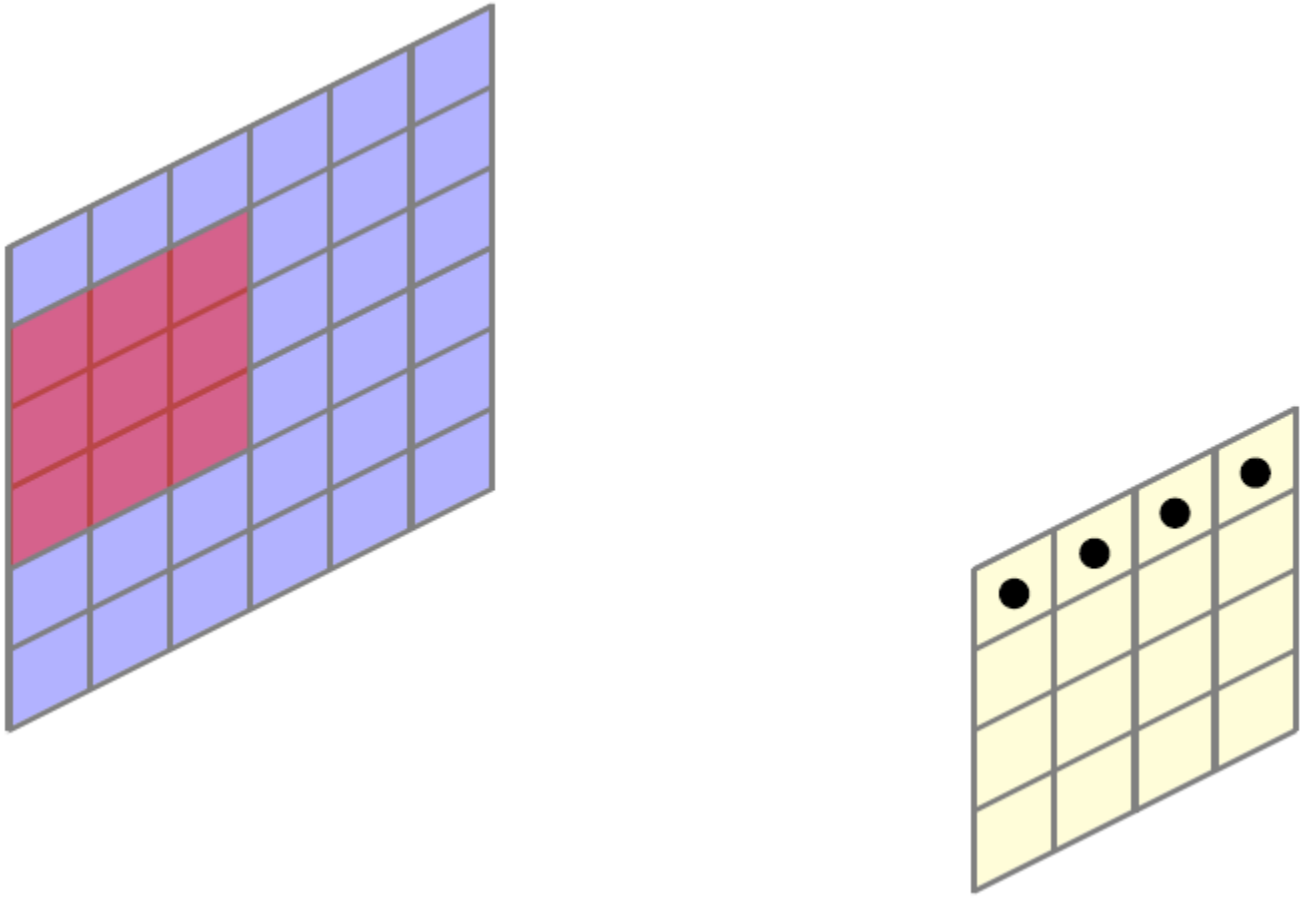
Convolution



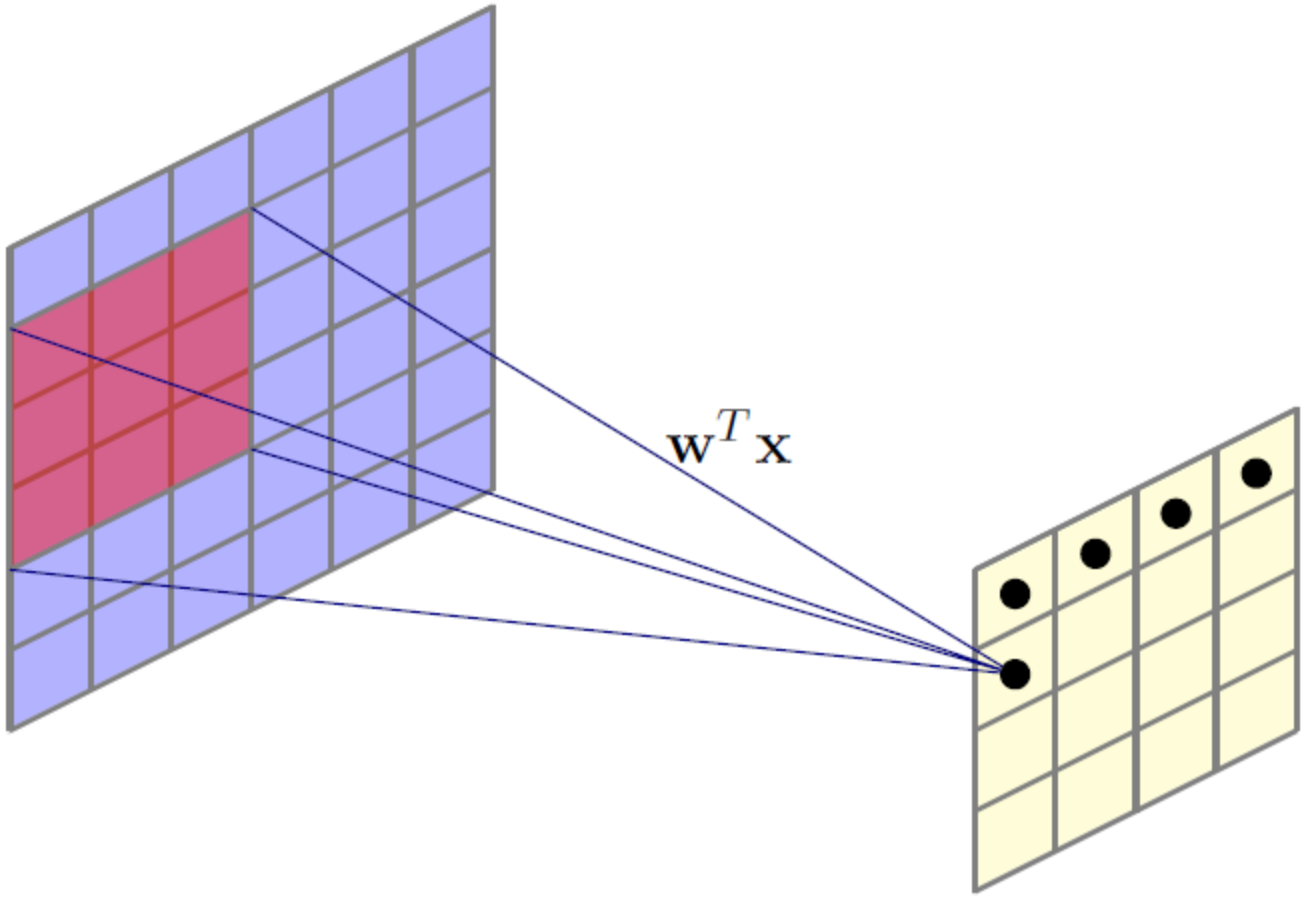
Convolution



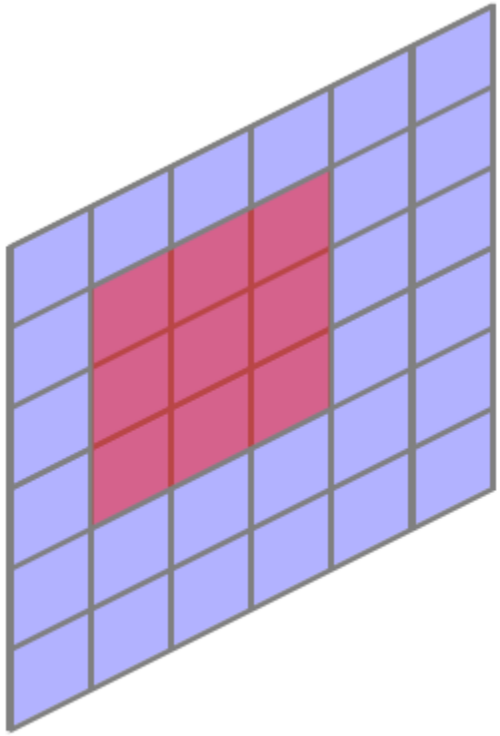
Convolution

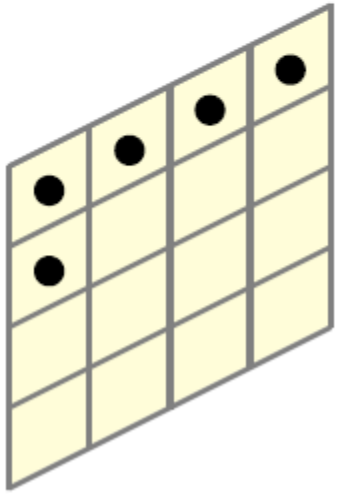


Convolution

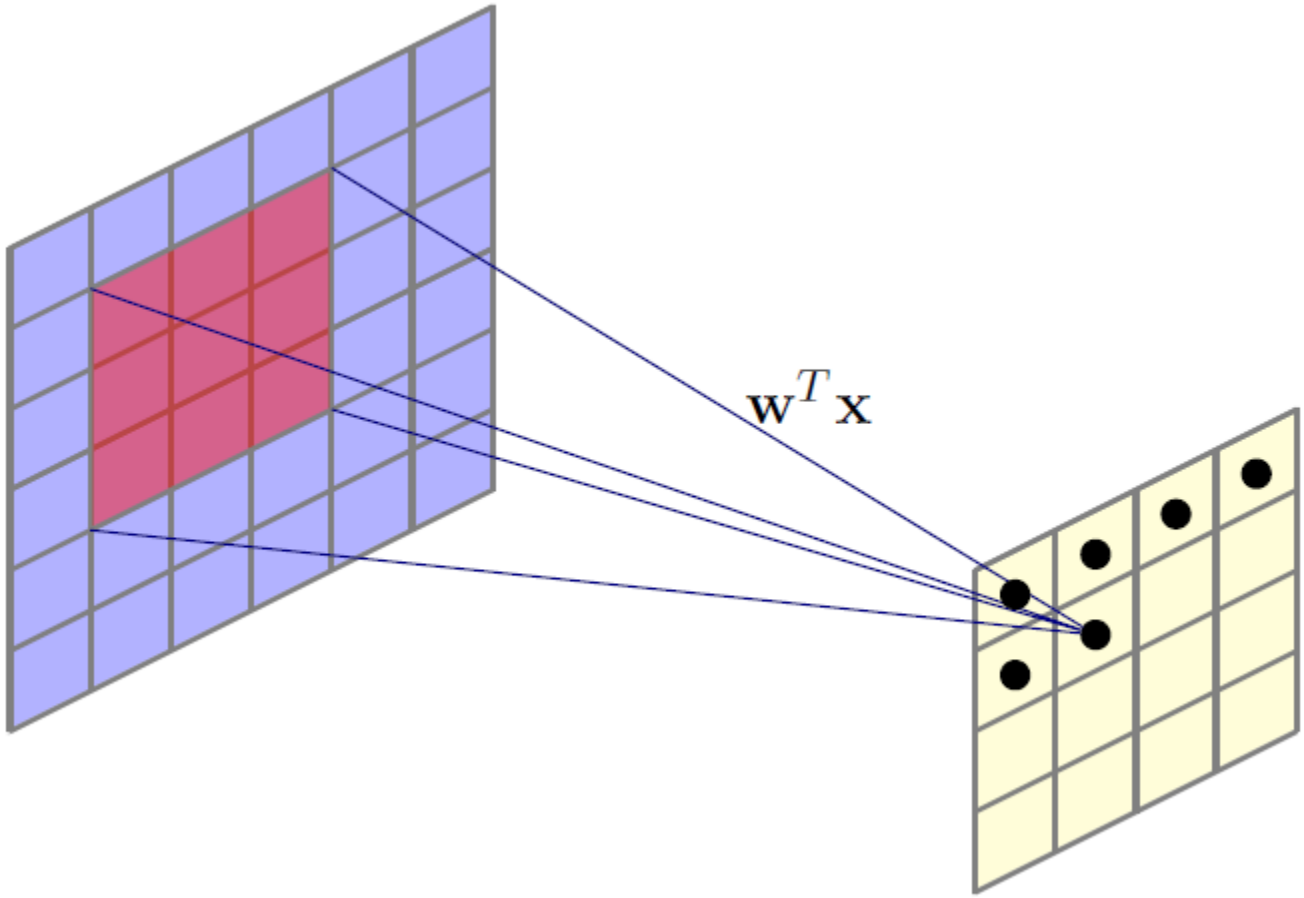


Convolution

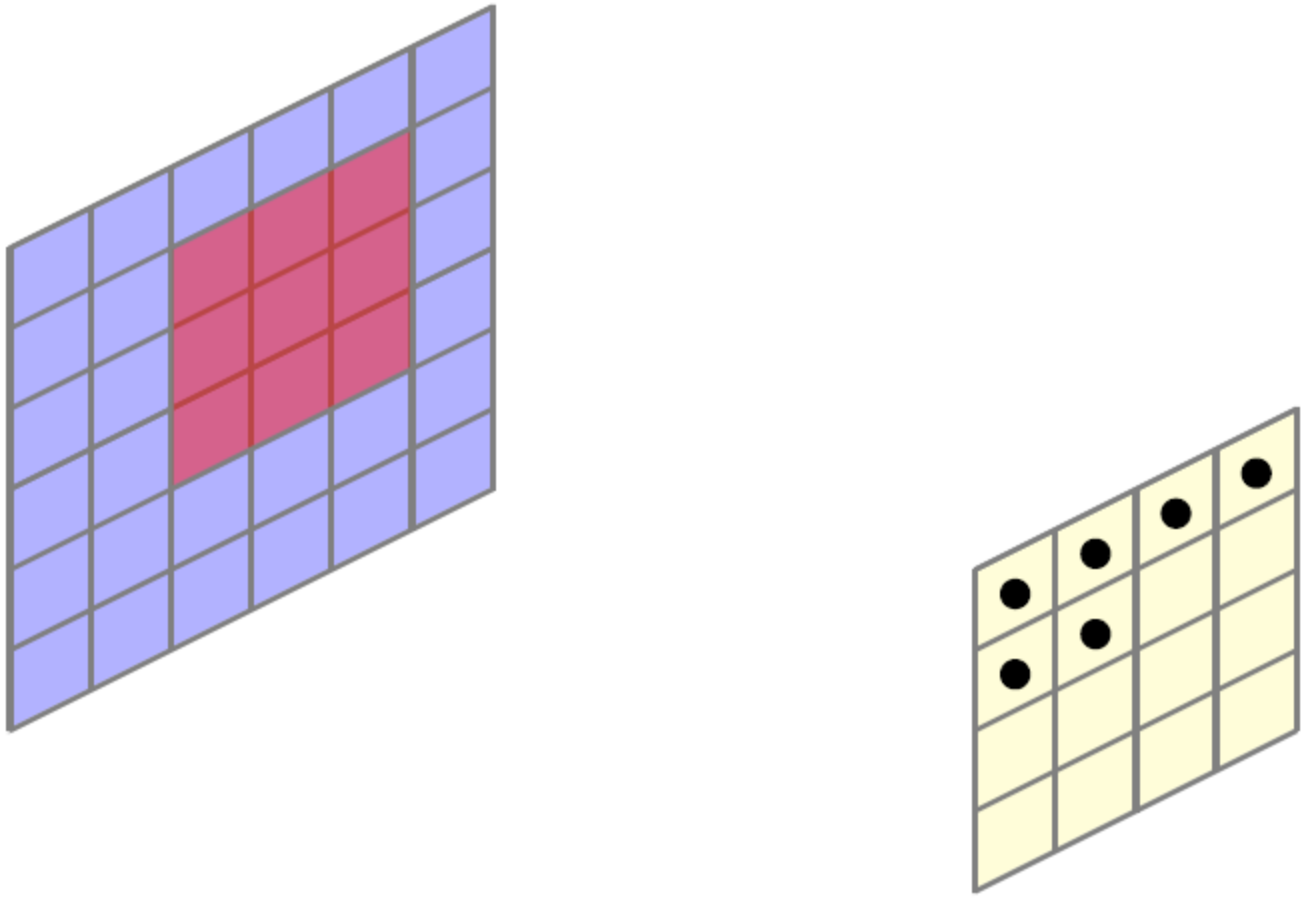




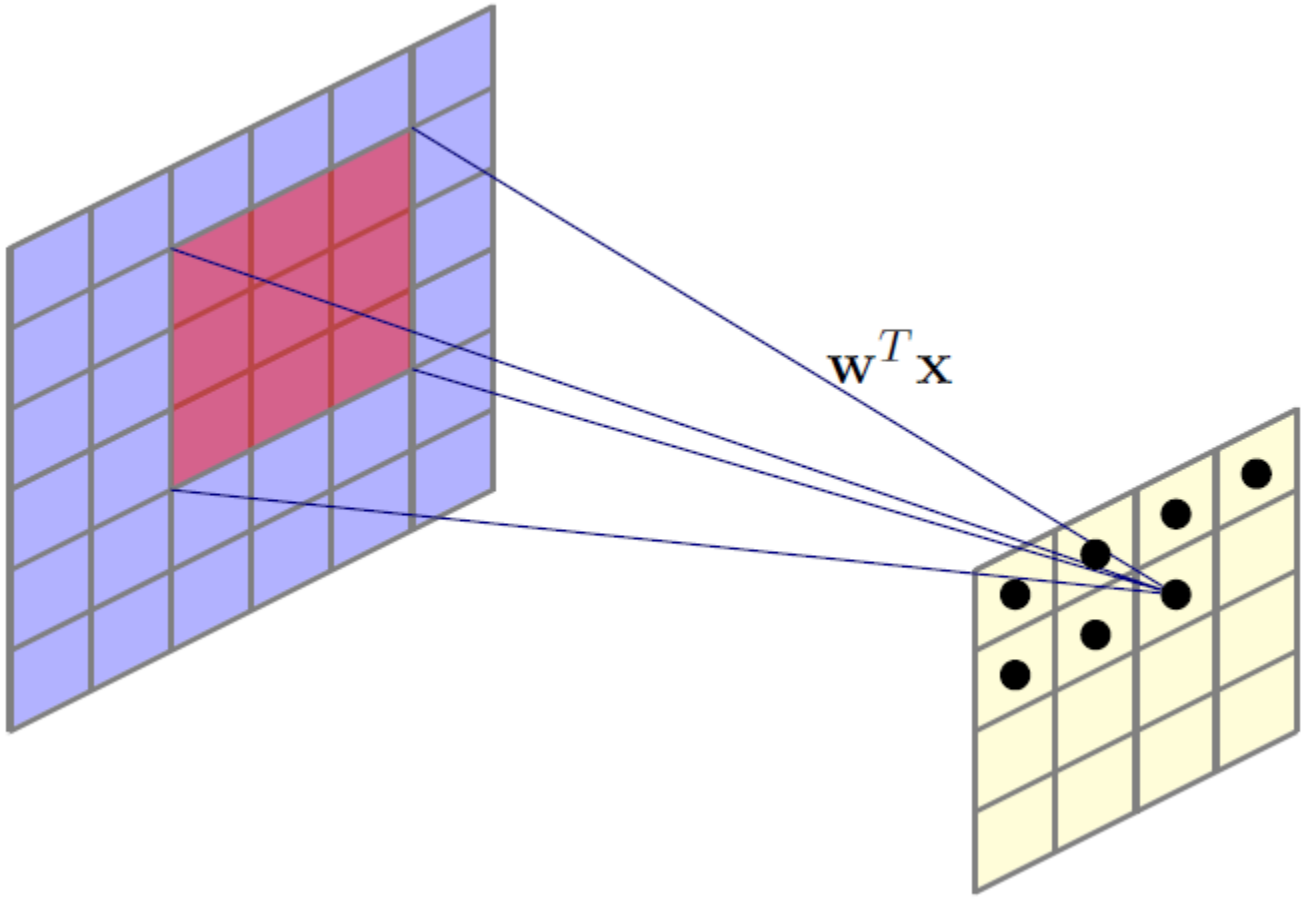
Convolution



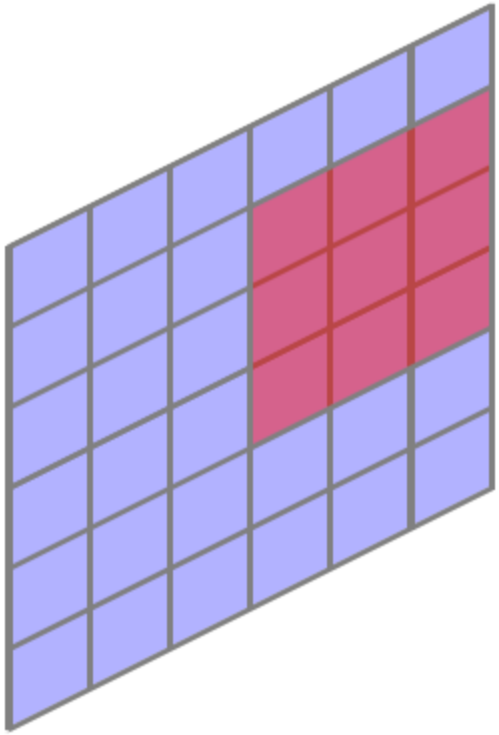
Convolution

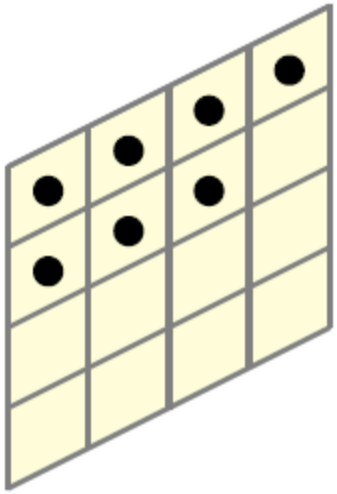


Convolution

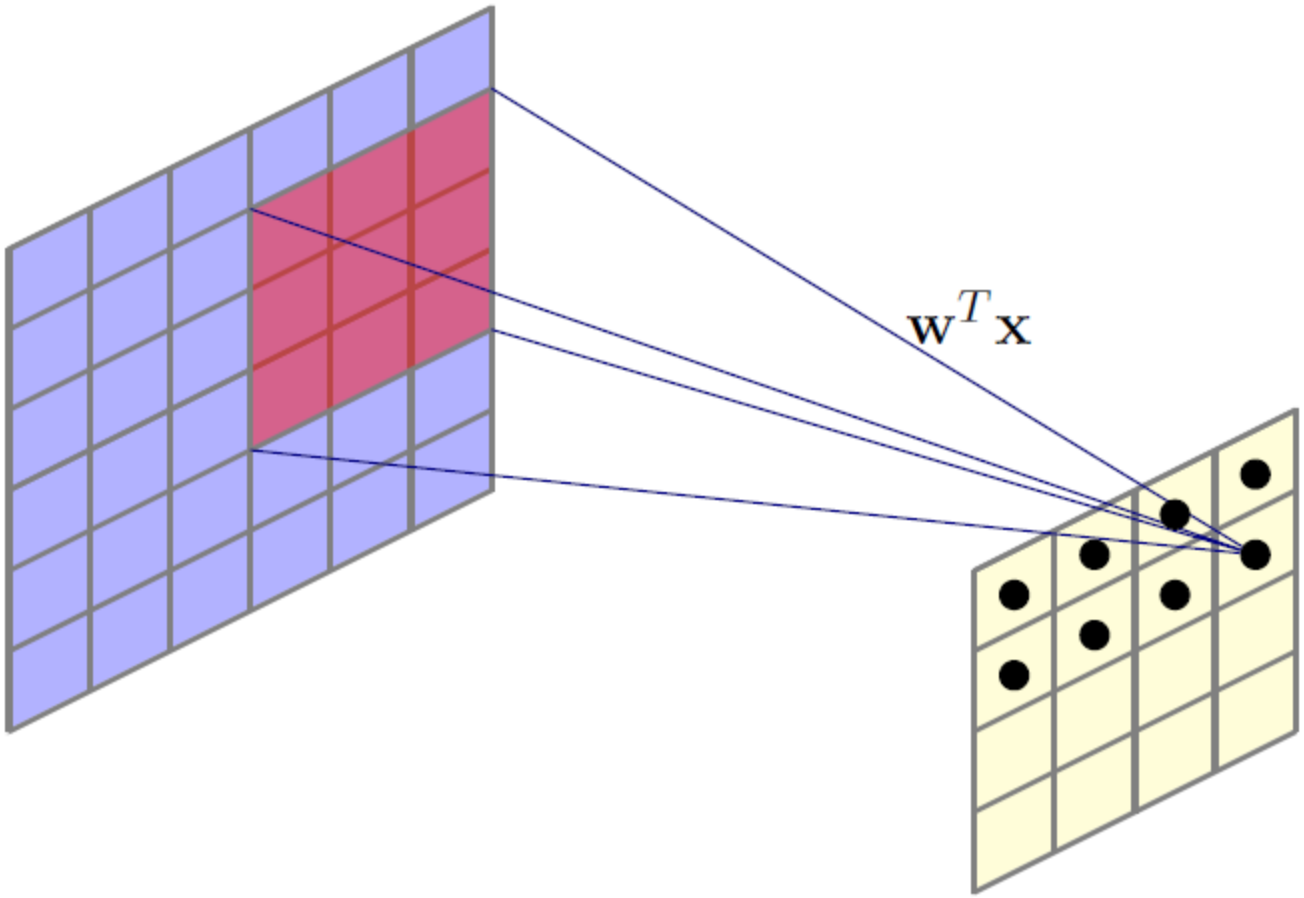


Convolution

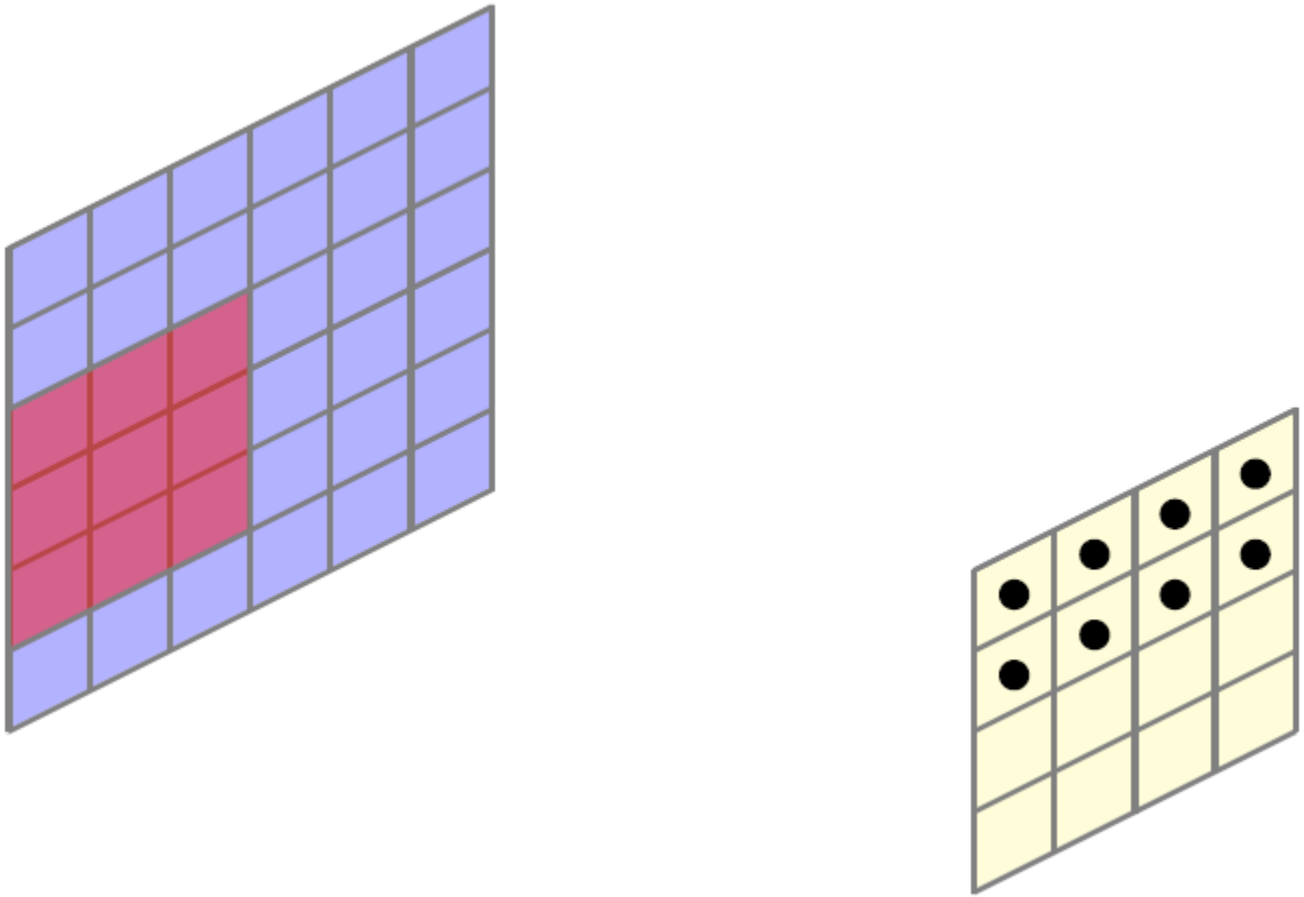




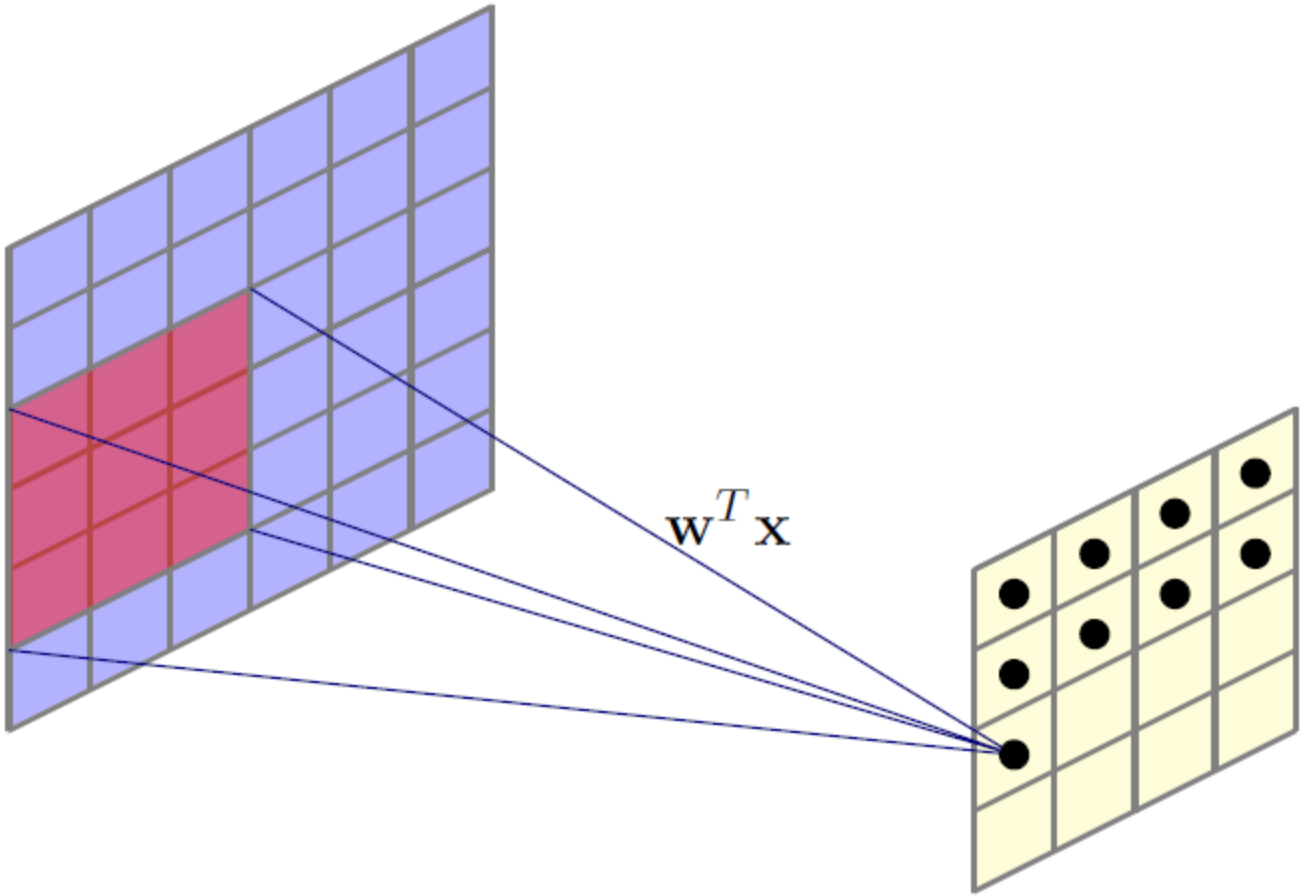
Convolution



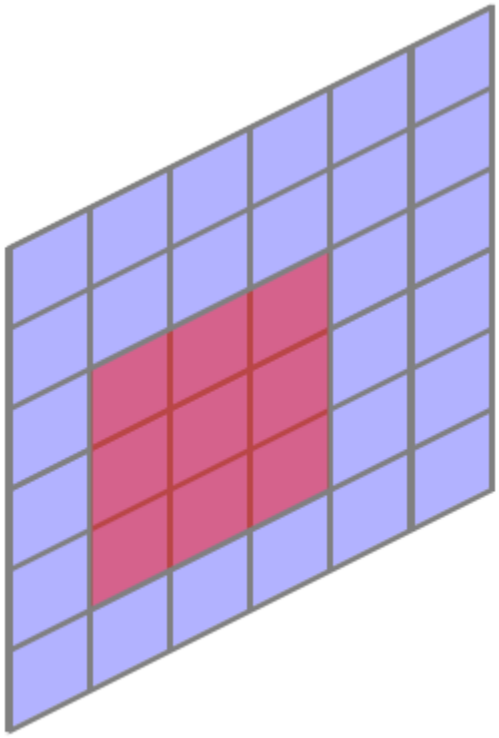
Convolution

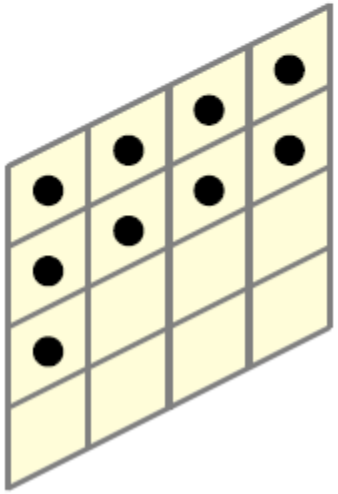


Convolution

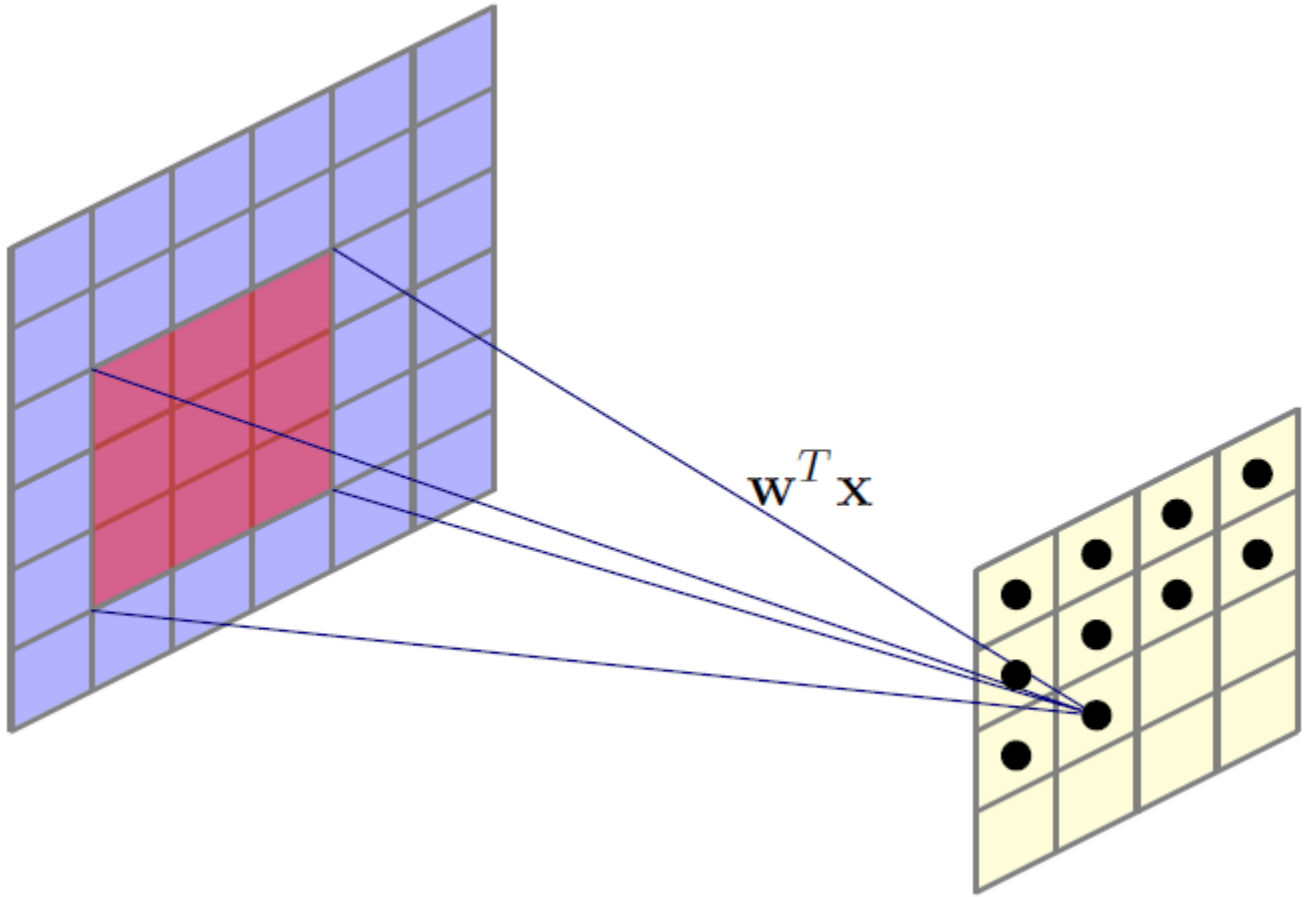


Convolution

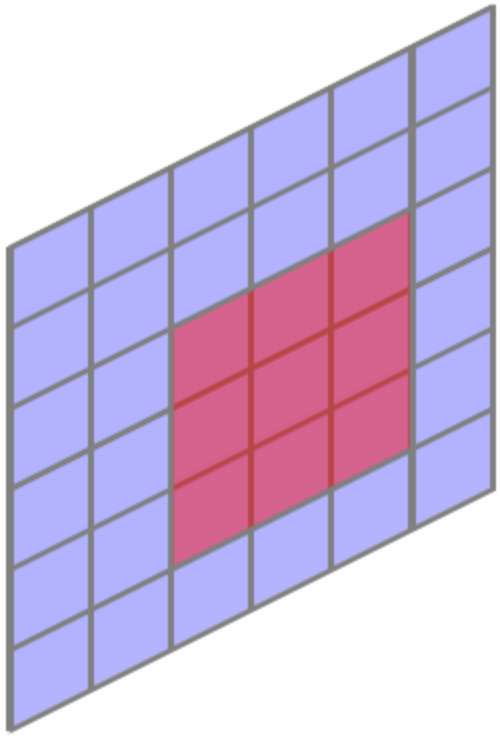


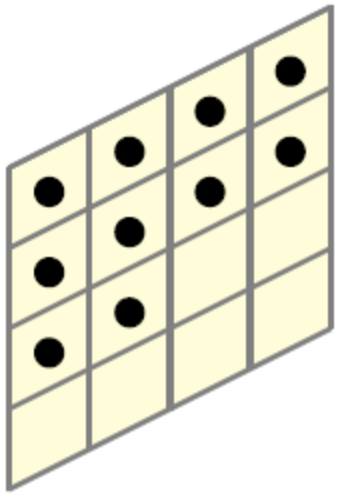


Convolution

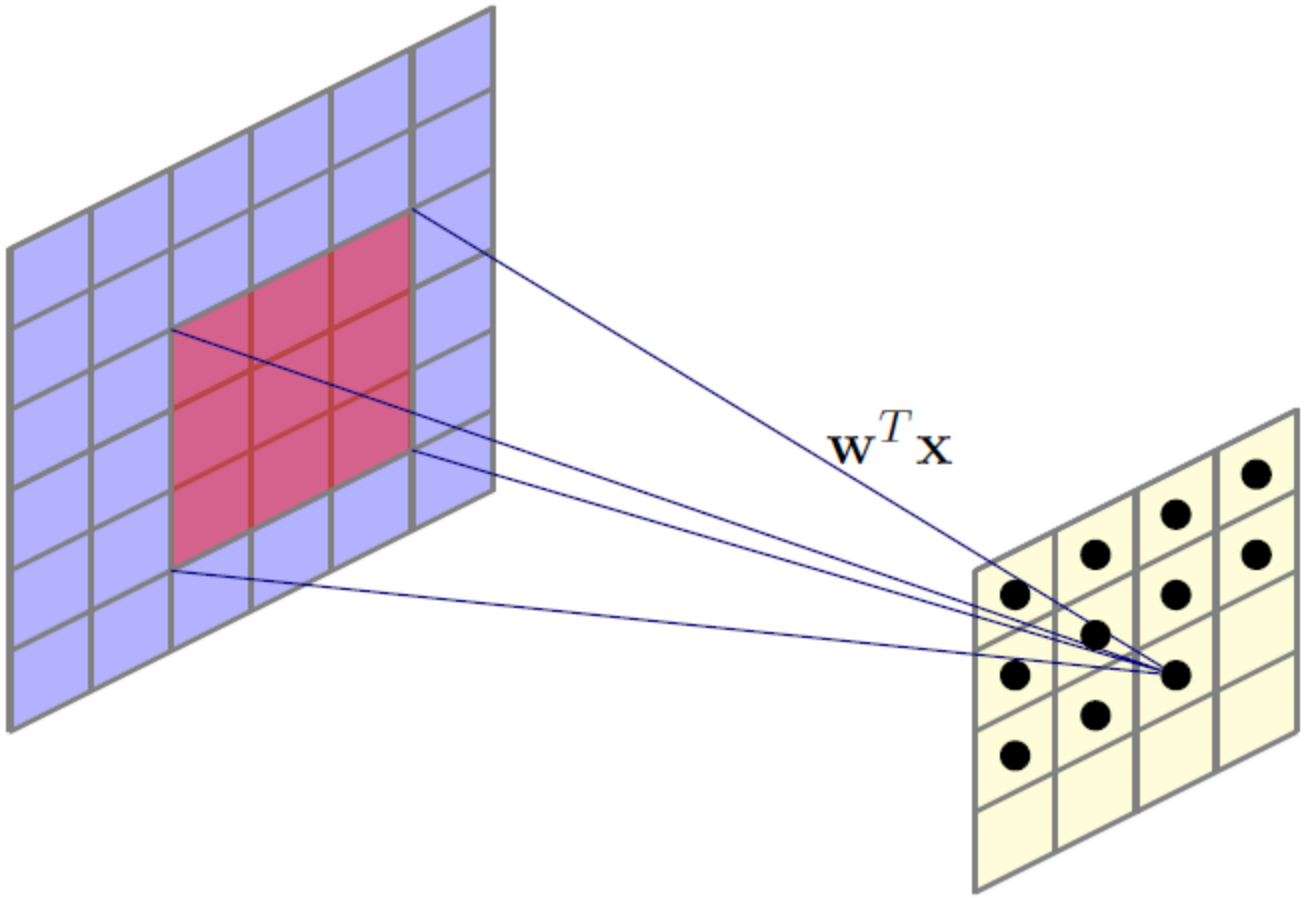


Convolution

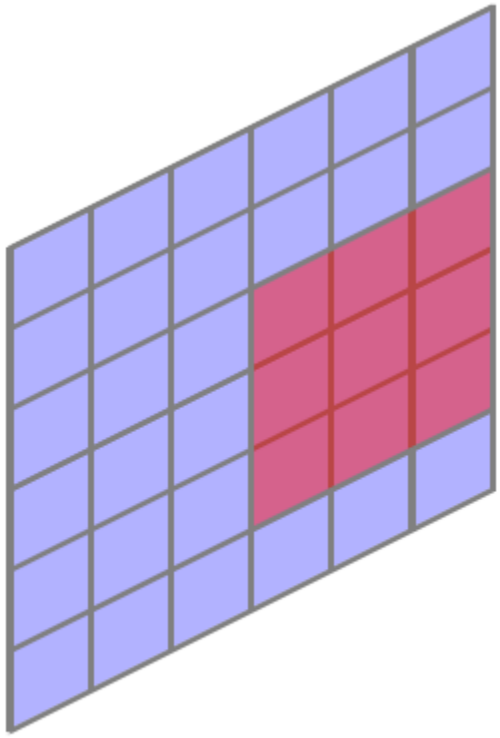


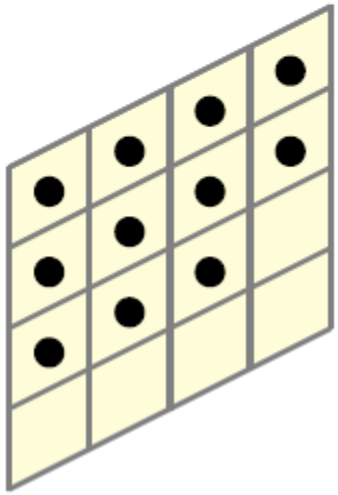


Convolution

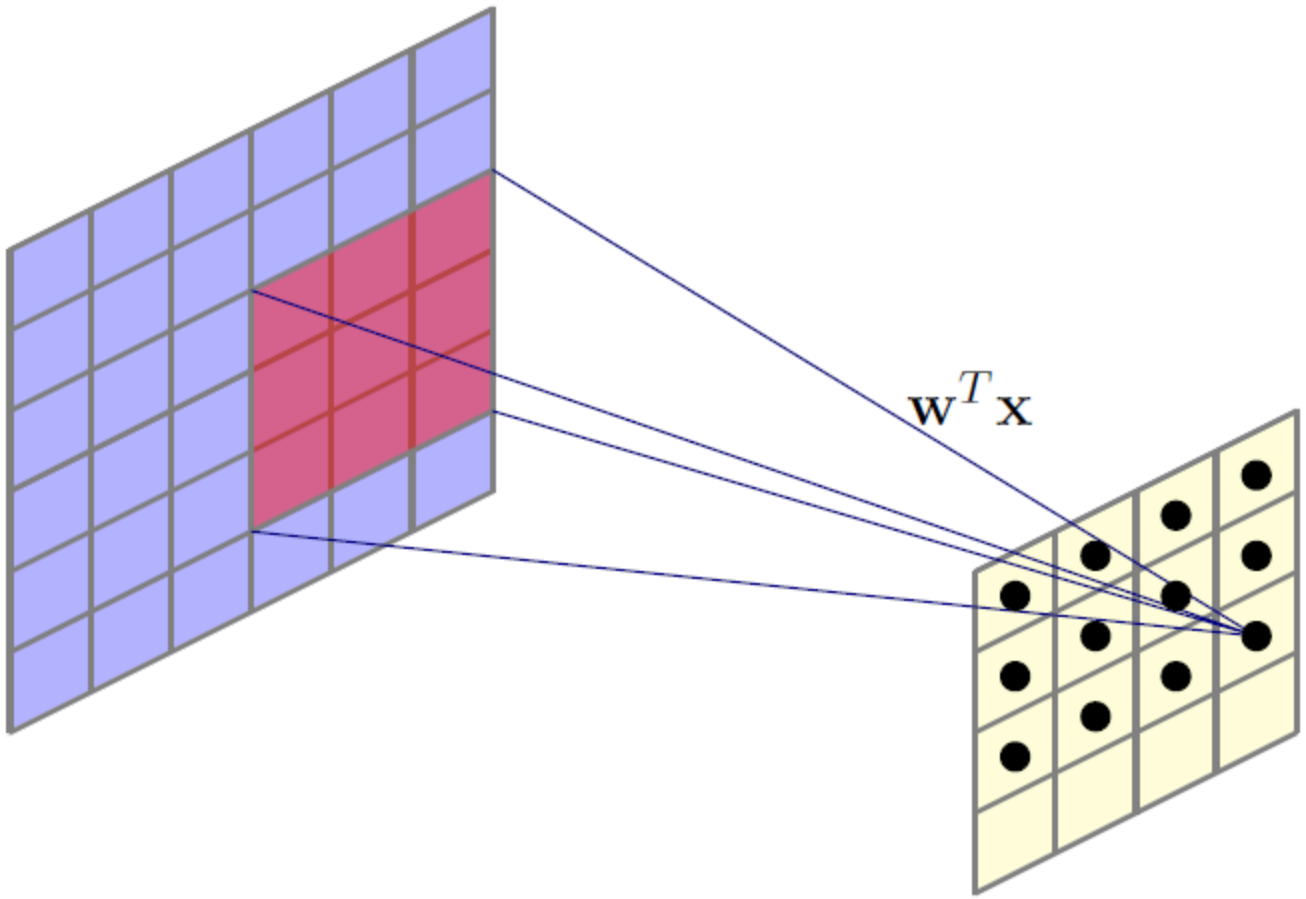


Convolution

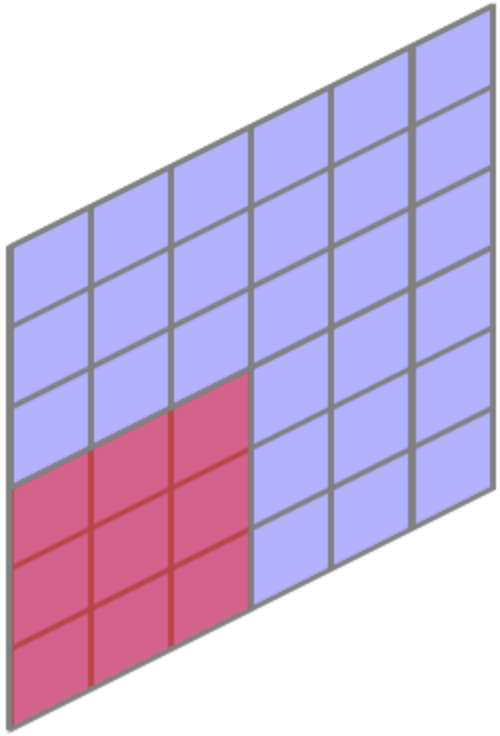


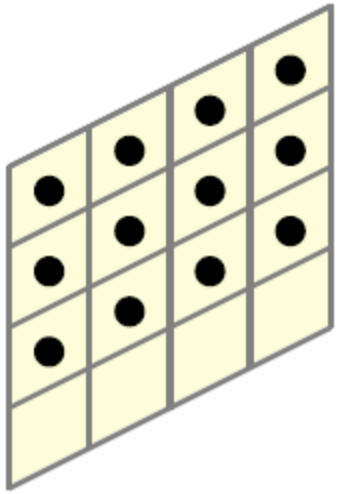


Convolution

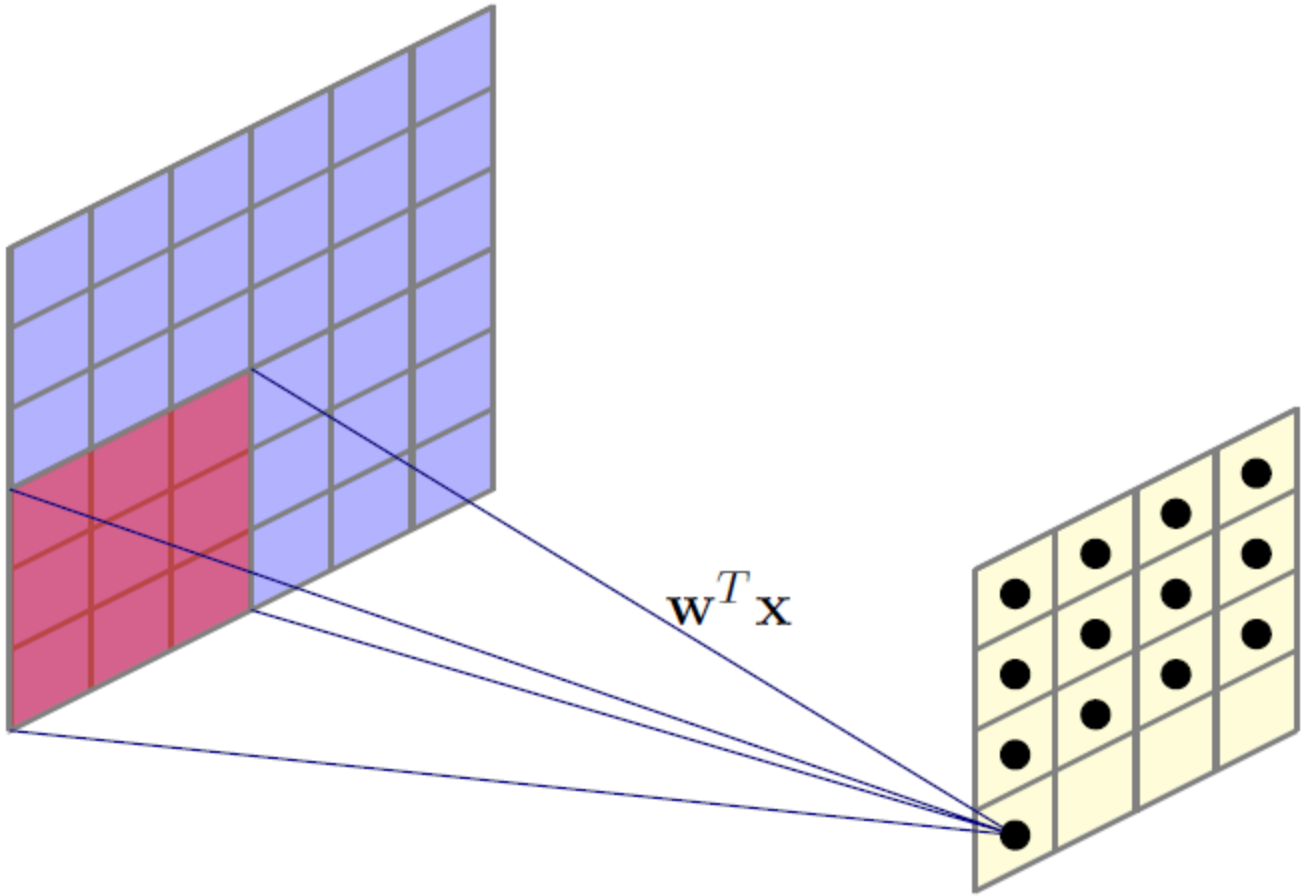


Convolution

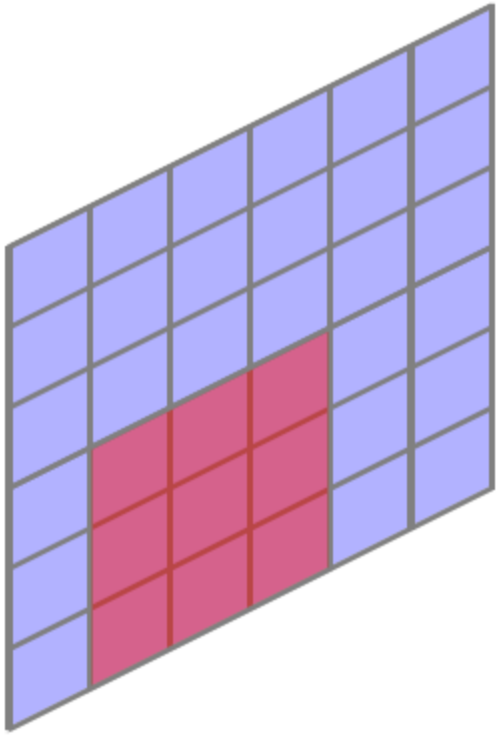


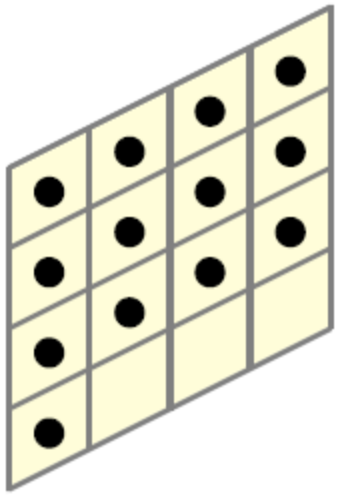


Convolution

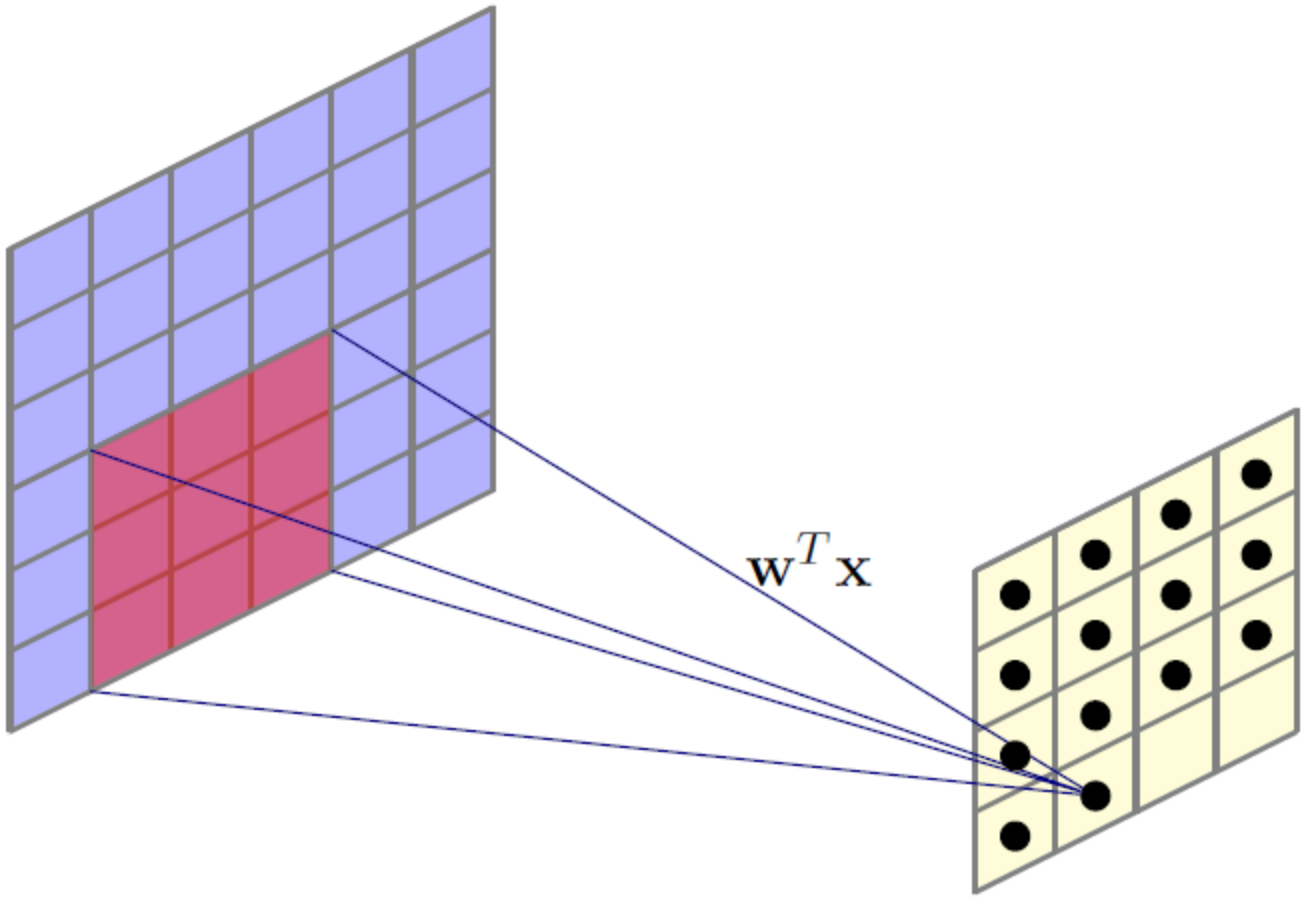


Convolution

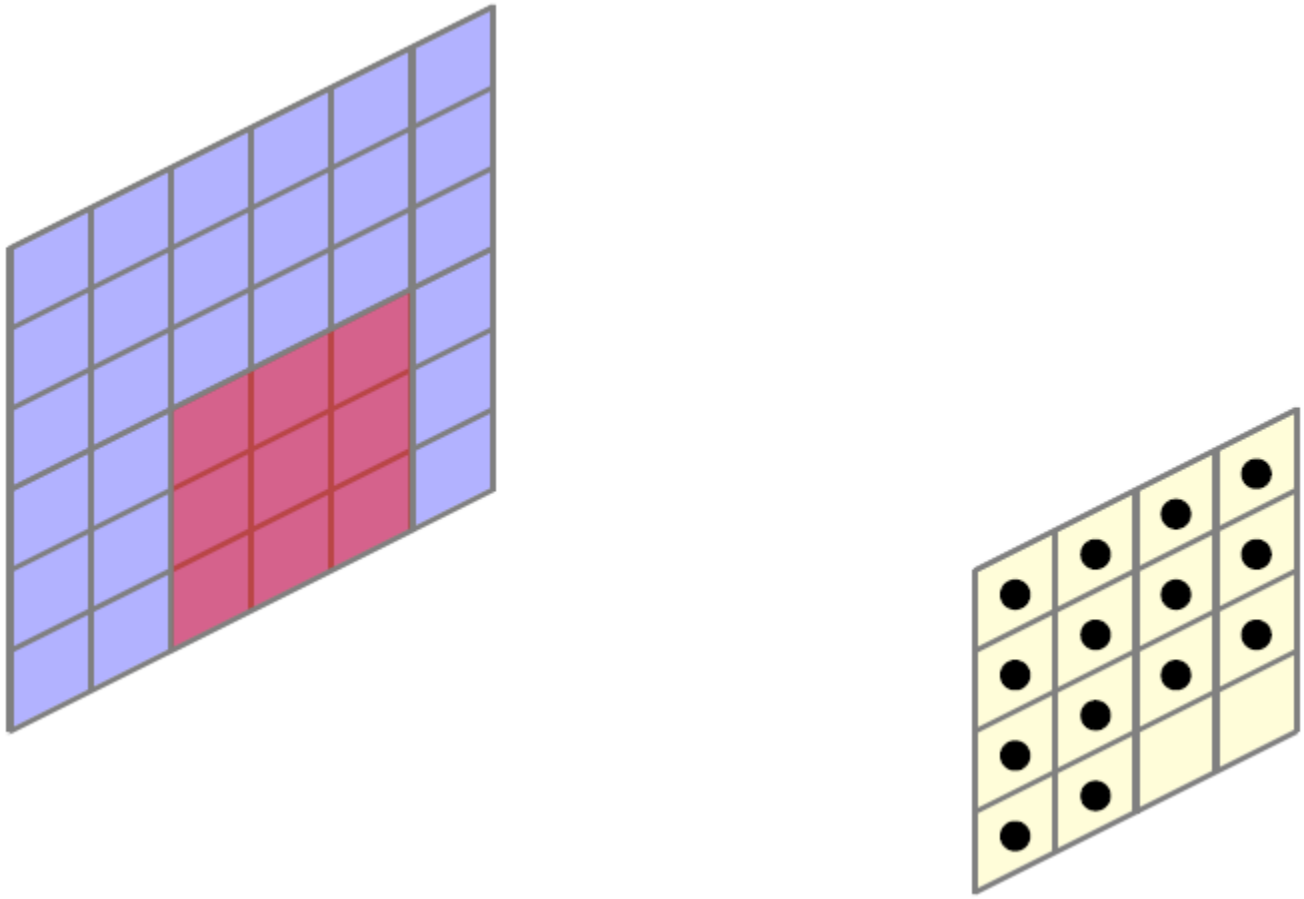




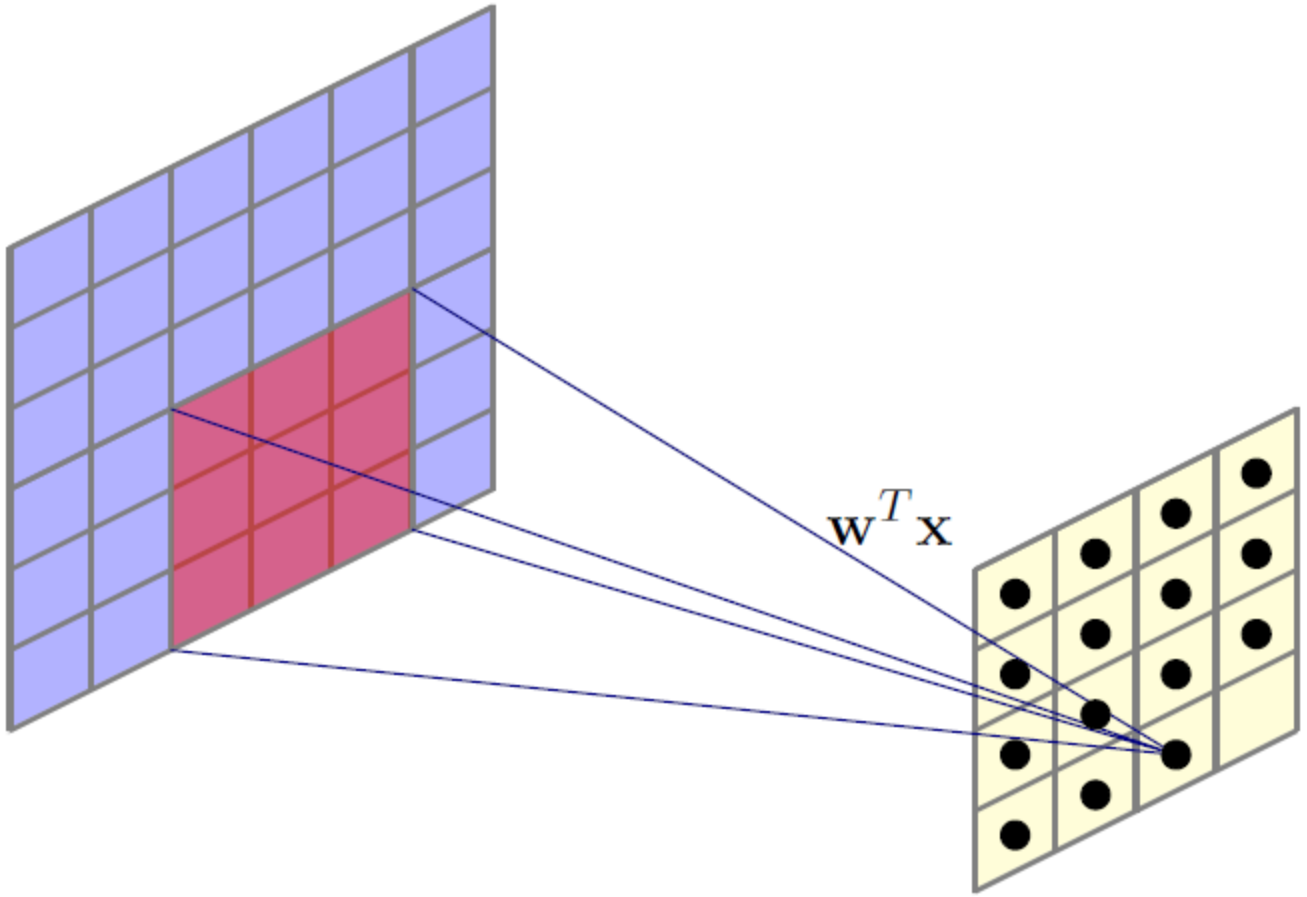
Convolution



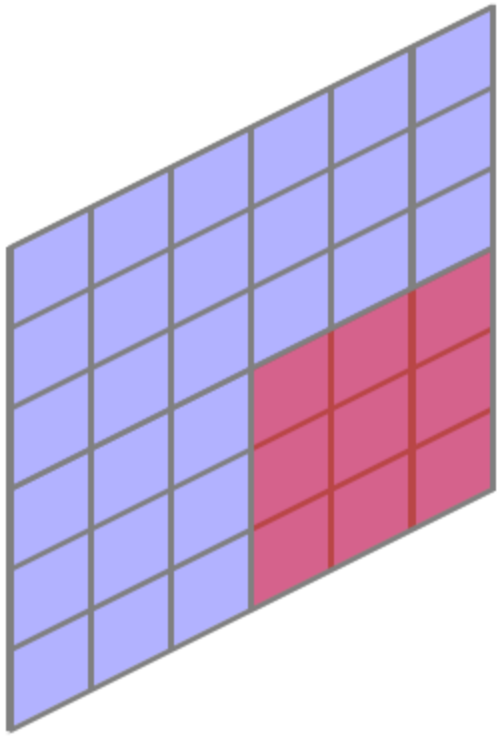
Convolution

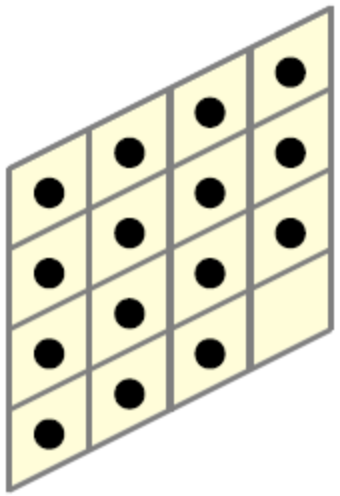


Convolution

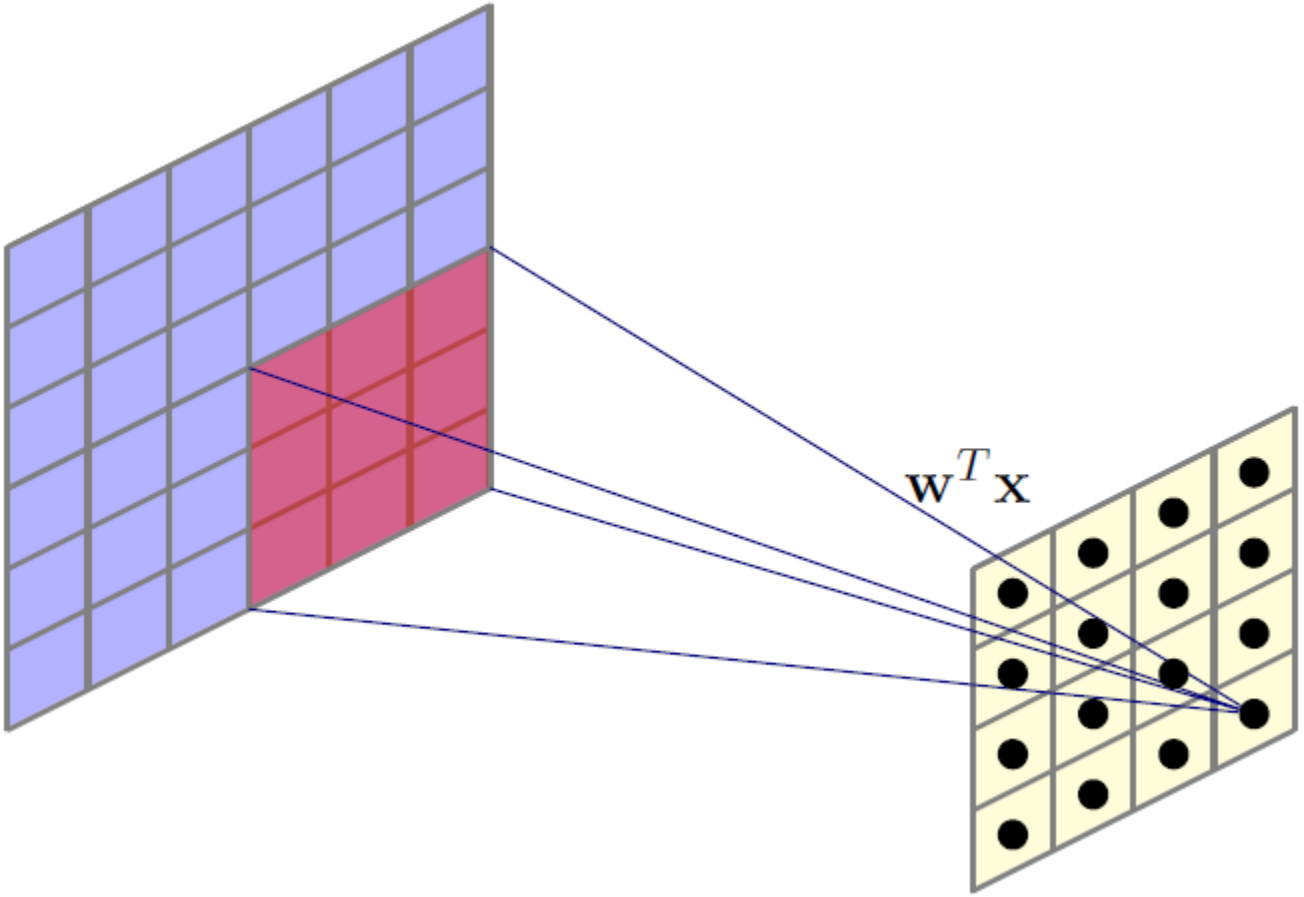


Convolution





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

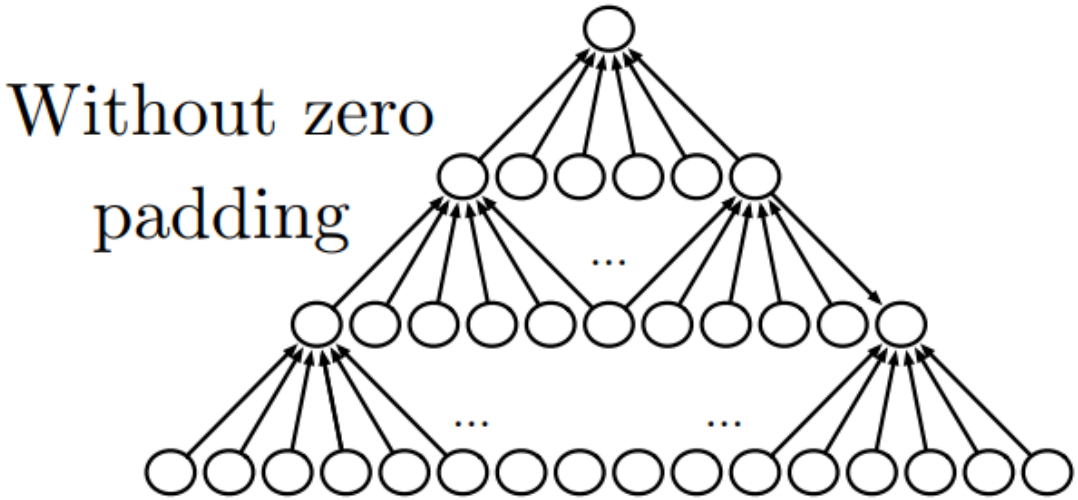
Zero Padding

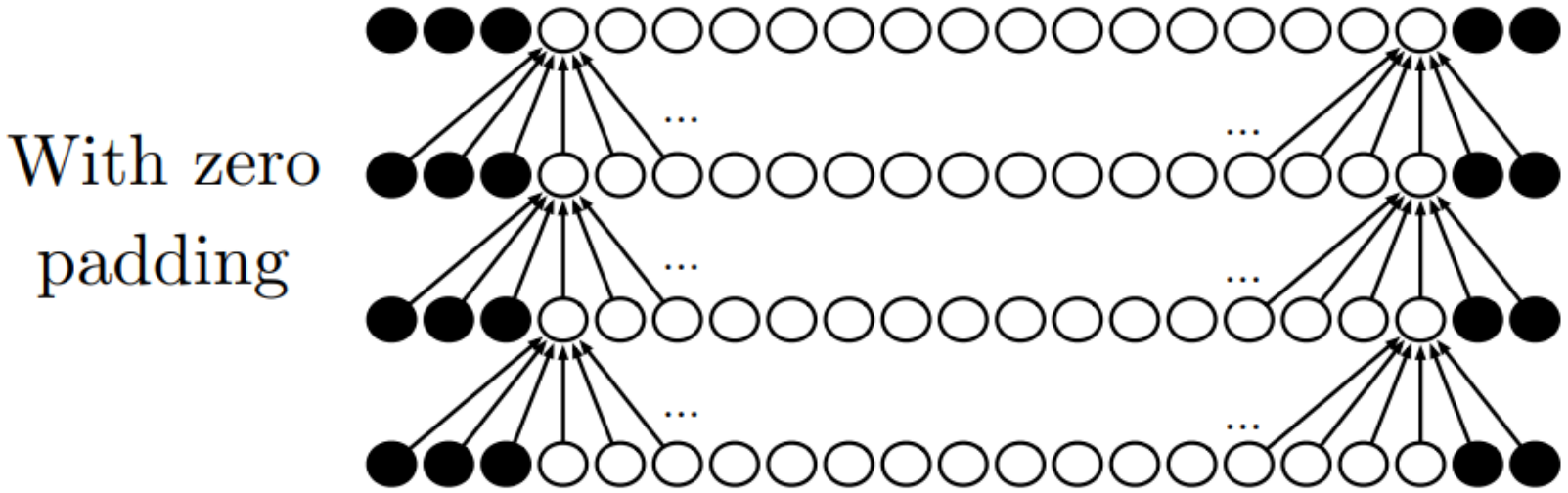
- Often, we want the output of a convolution to have the same size as the input. Solution: Zero padding.
- In our previous example:

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

Zero Padding





Learn Multiple Filters

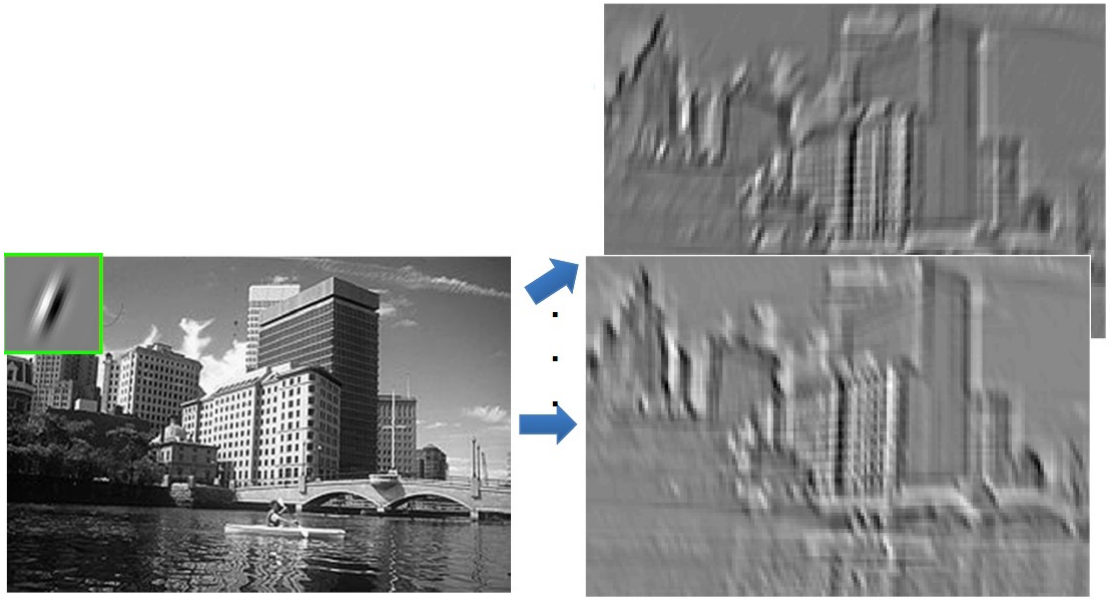


Figure: I. Kokkinos

- 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)

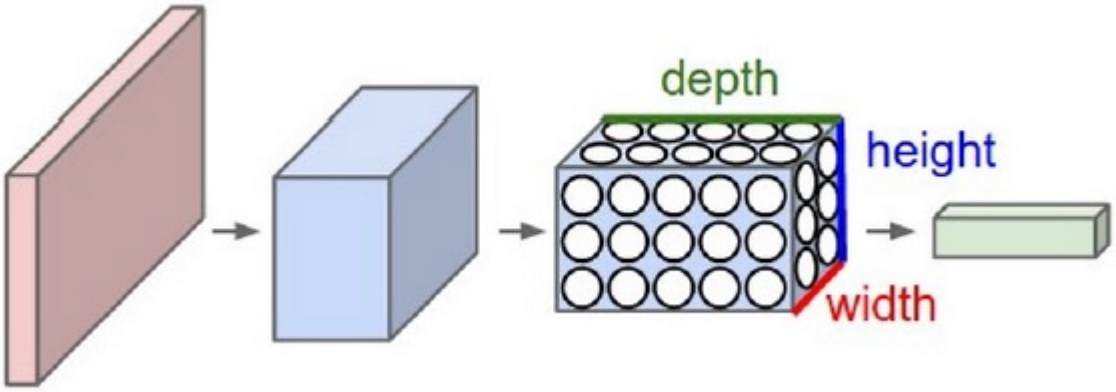


Image from Wikipedia

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:

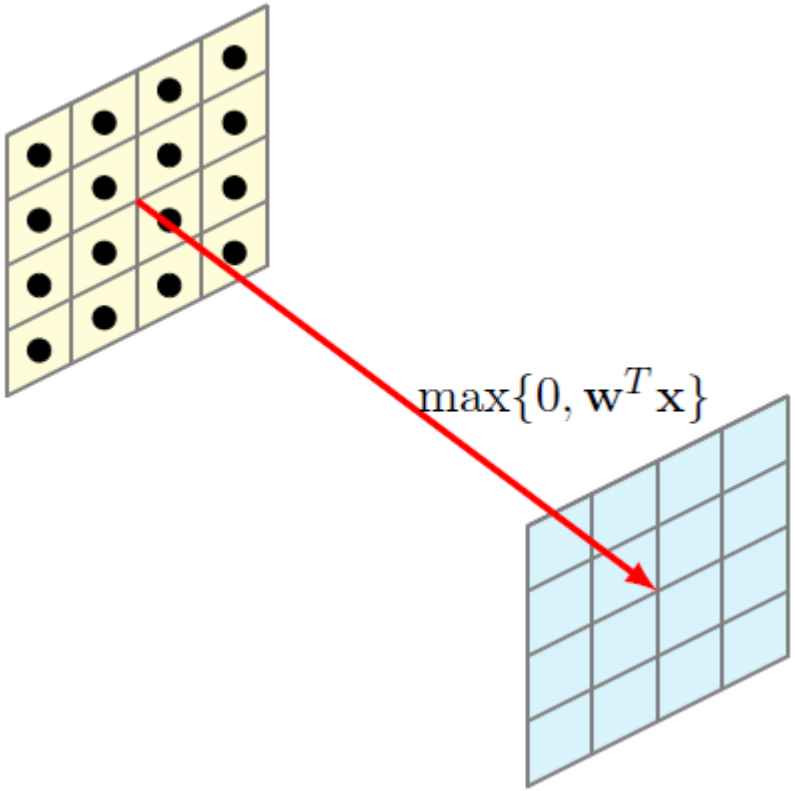
$$W_2 = \frac{W_1 - K}{S} + 1 \text{ and } H_2 = \frac{H_1 - K}{S} + 1$$

  - Depth of filter will be equal to the depth of input

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: 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

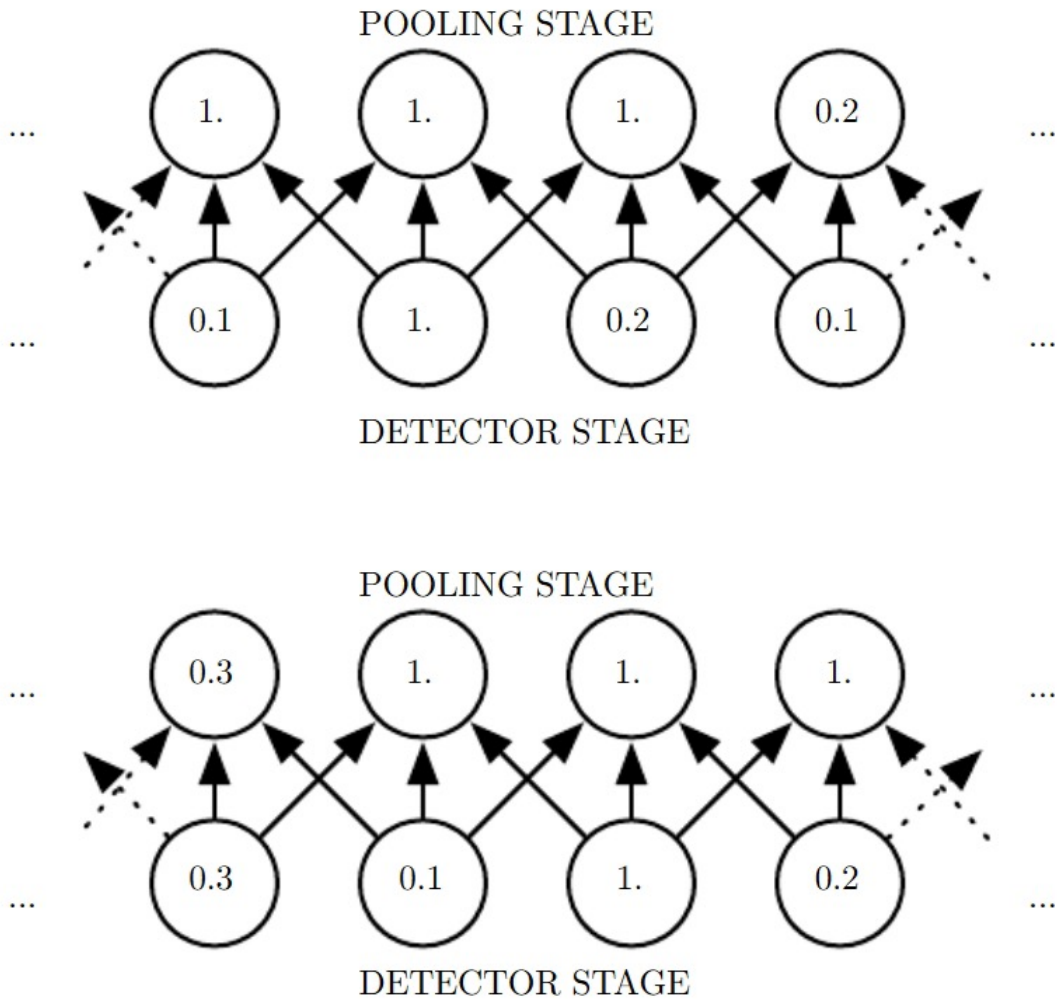
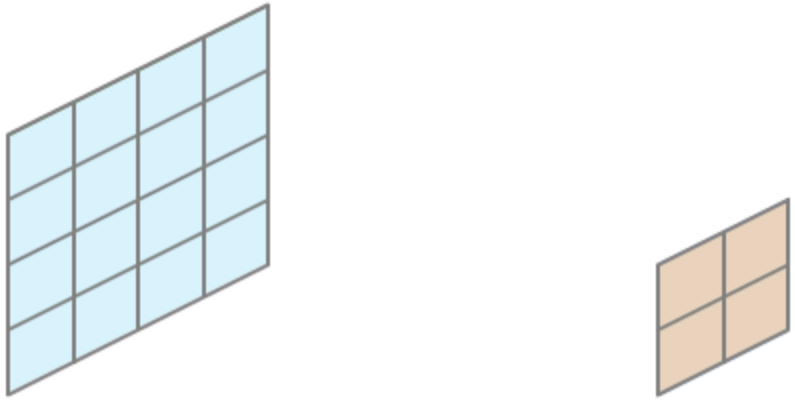
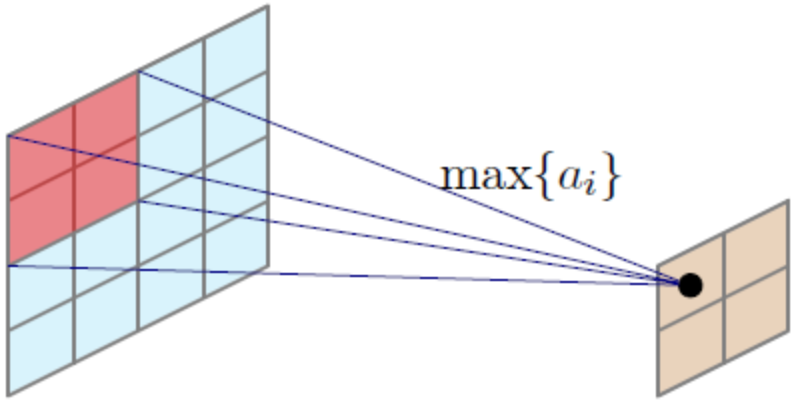


Figure: Goodfellow et al.

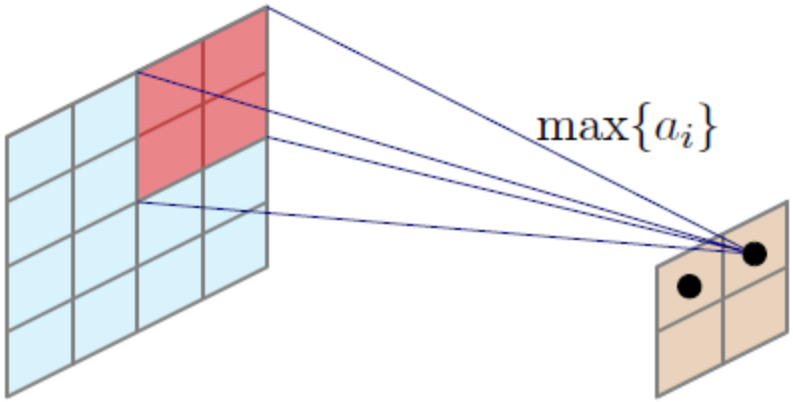
Pooling



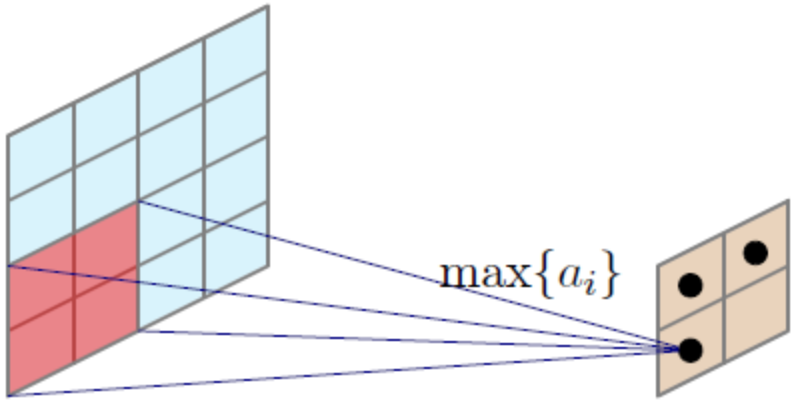
Pooling



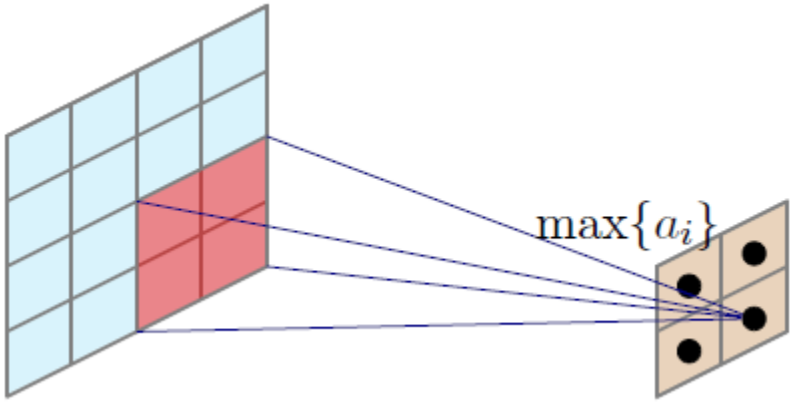
Pooling



Pooling

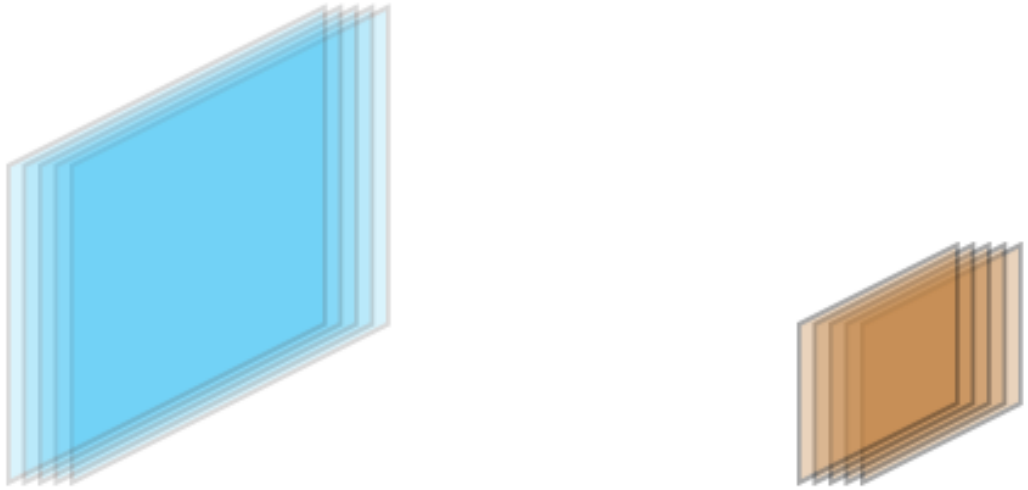


Pooling



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

Pooling

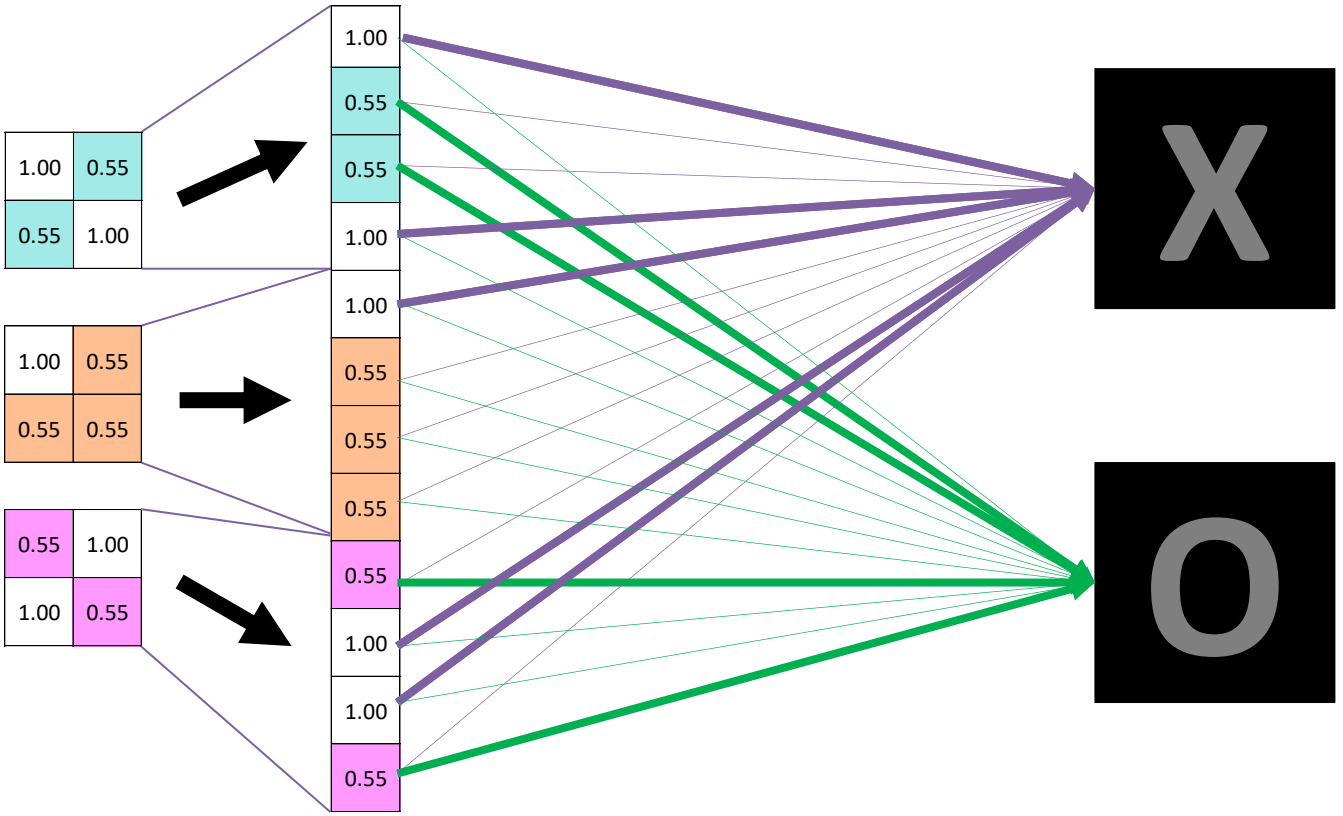


- We have multiple feature maps, and get an equal number of subsampled maps
- This changes if cross channel pooling is done

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

Fully Connected Layer

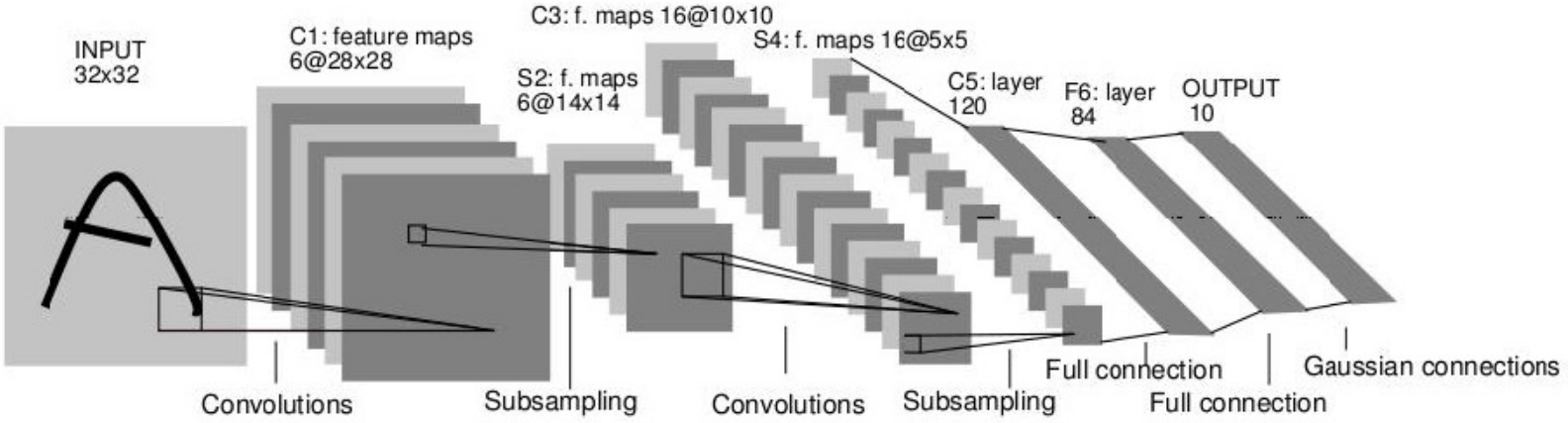


- Take the high-level filtered images and translate them into vector
- Same as the other layers, they can be stacked to make better decisions

Outline

1. Course Review
2. Basic Operations
3. Major Architectures
4. CNN for Image Classification
5. Know More about CNN

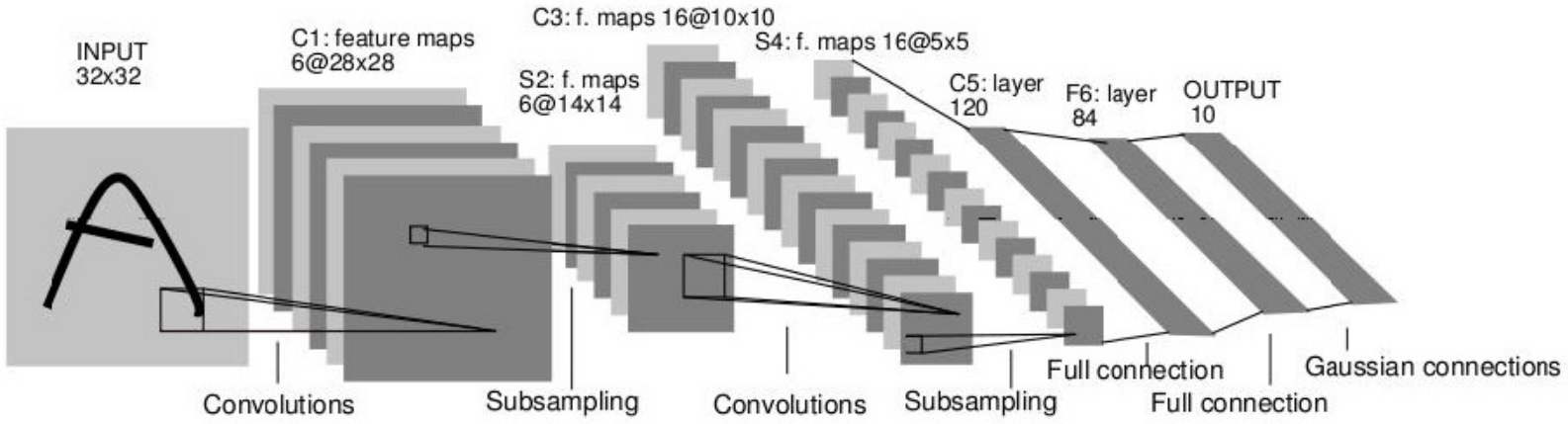
LeNet-5



- 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.

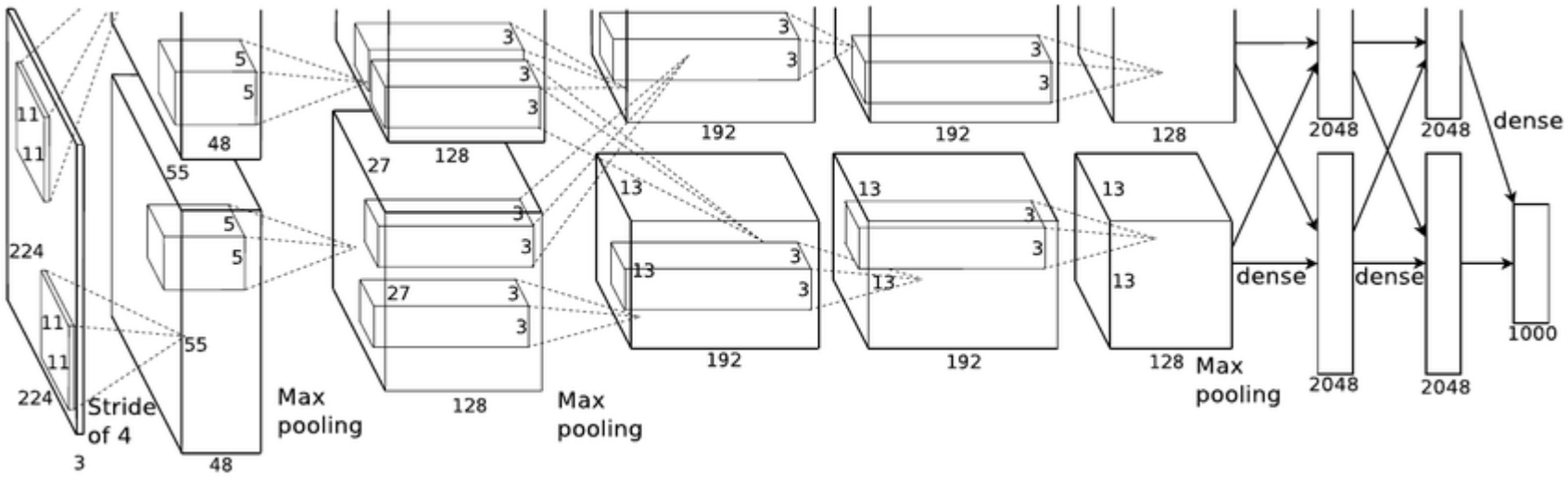
LeNet-5



- 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.

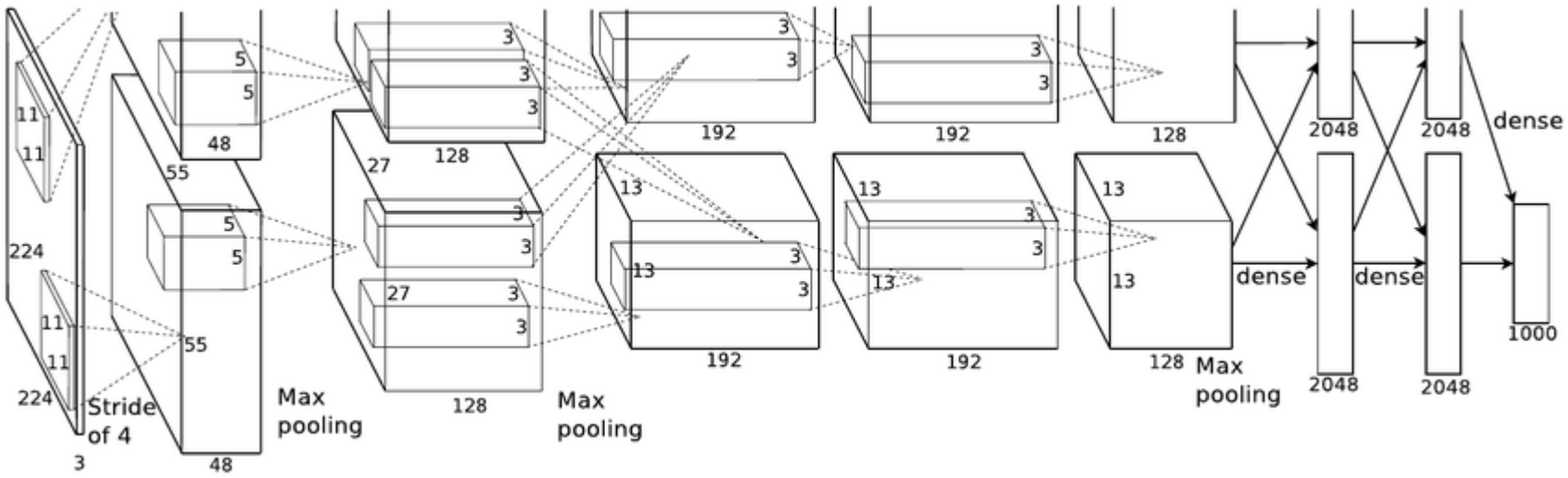
AlexNet: ILSVRC 2012 winner



- 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

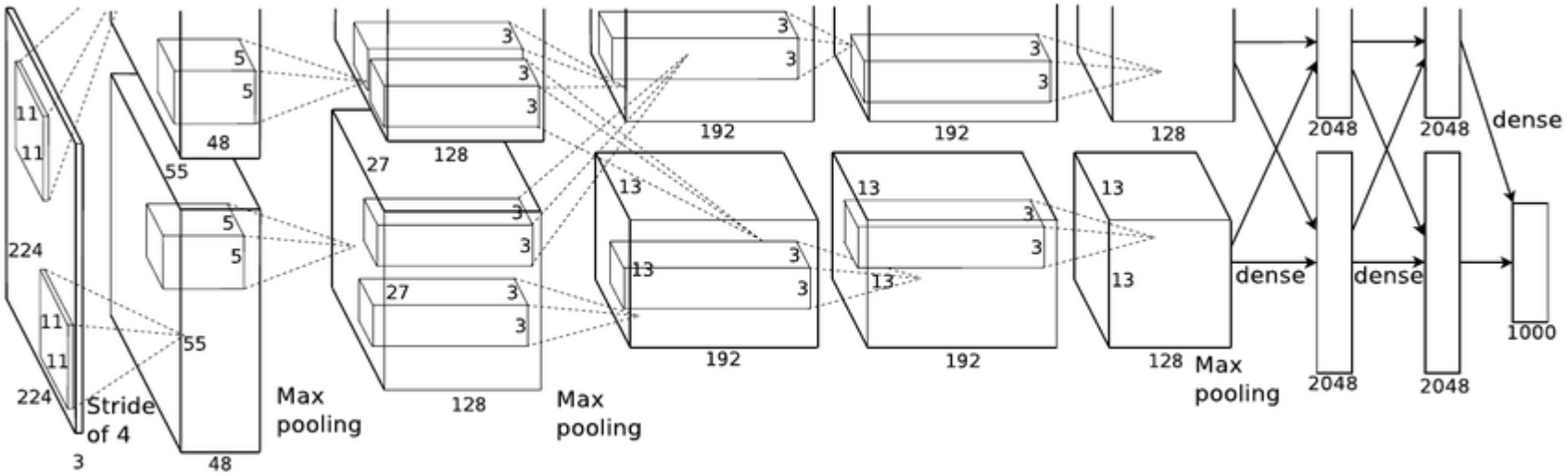
AlexNet: ILSVRC 2012 winner



- 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

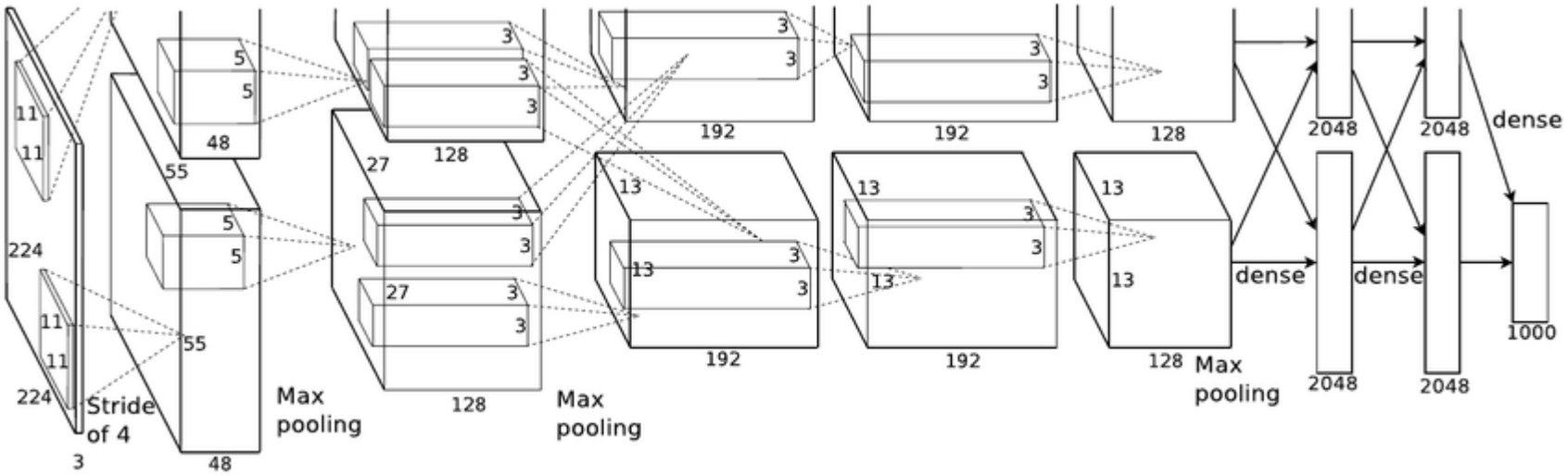


Number of parameters in first layer?

11 X 11 X 3 X 96 +96 × 1 = 34944

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

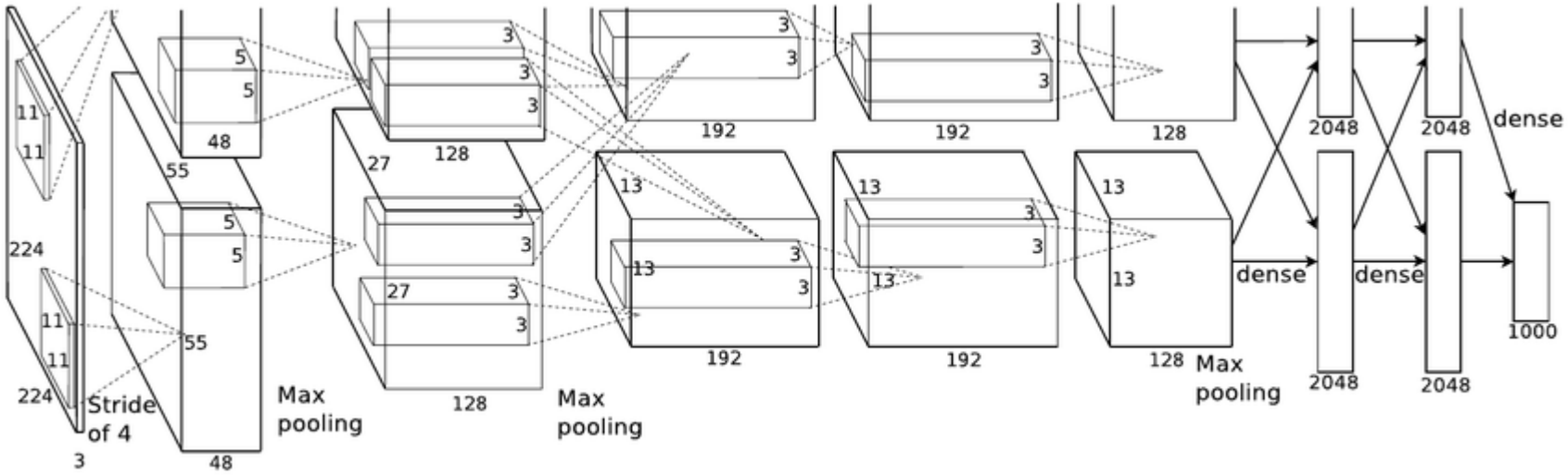
AlexNet: ILSVRC 2012 winner



- 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)
- Hyperparameters: 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

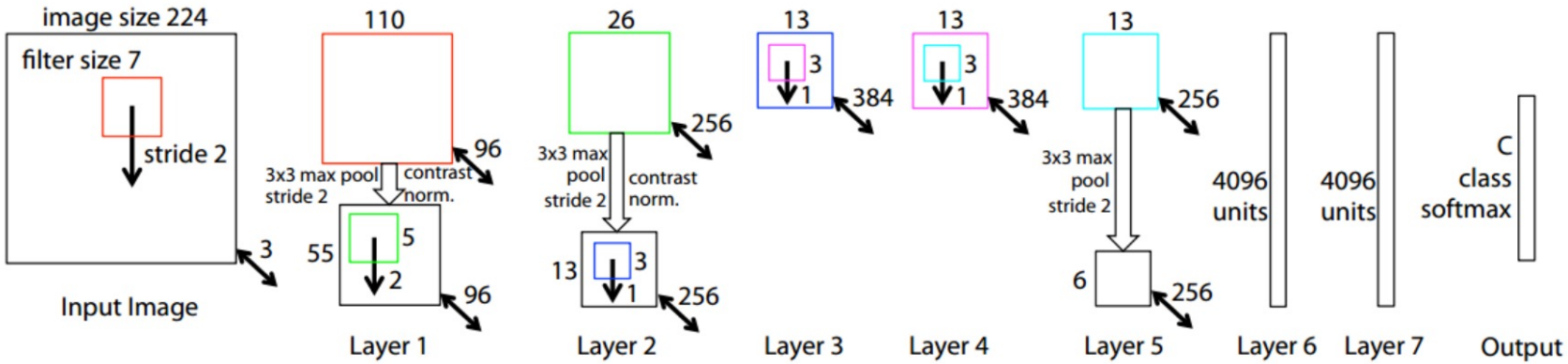


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through 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 3x3 convolutions (with ReLU in between)
- One 7x7 conv layer with C feature maps needs $49C^2$ weights, three 3x3 conv layers need only $27C^2$ weights（not include bias here）

Best model: Column D.

Error: 7.3 % (top five error)

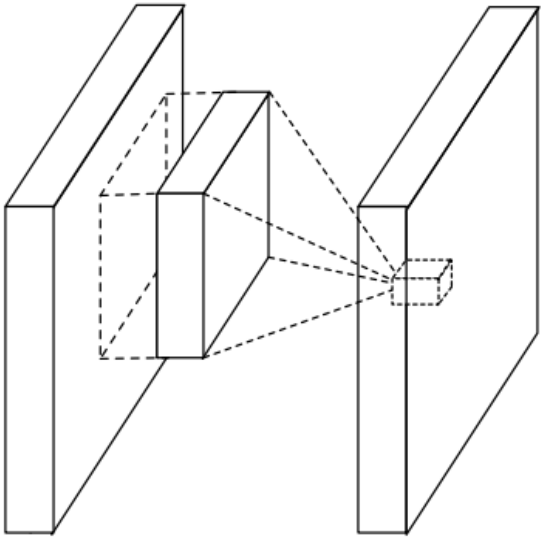
K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

VGGNet: ILSVRC 2014 2nd place

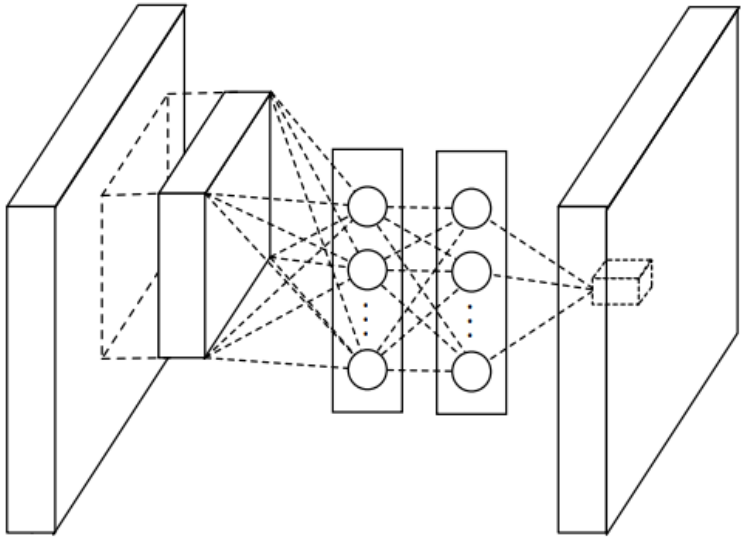
- Total number of parameters: 138 Million (calculate!)
- Memory (Karpathy): 24 Million X 4 bytes 93 MB per image
- For backward pass the memory usage is doubled per image
- Observations:
  - Early convolutional layers take most memory
  - Most parameters are in the fully connected layers

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

Network in Network



(a) Linear convolution layer

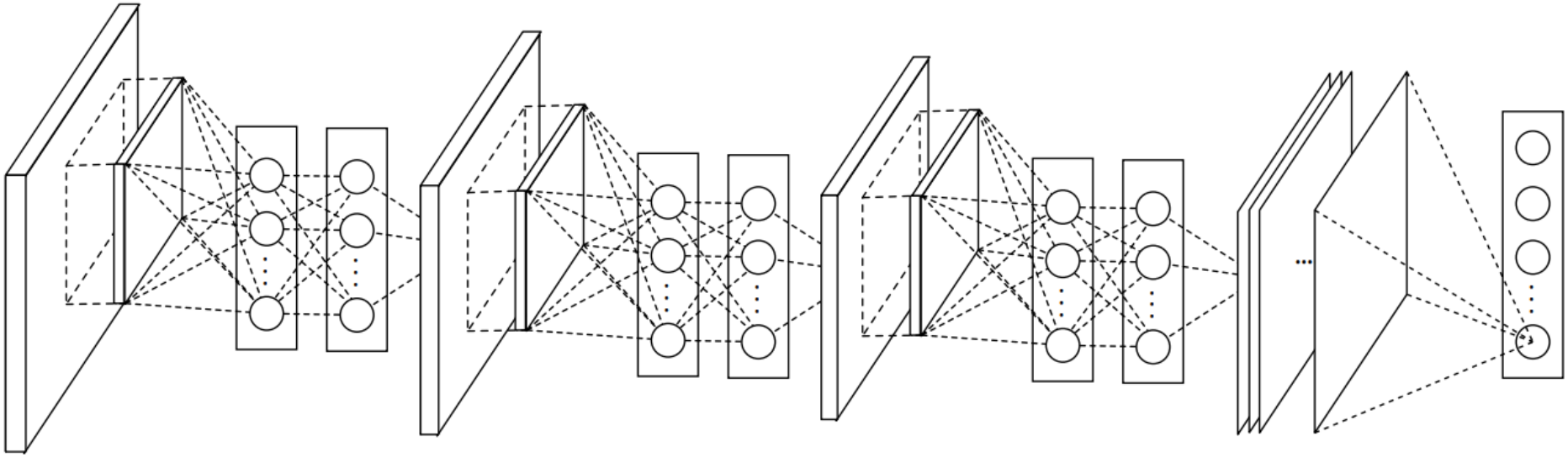


(b) Mlpconv layer

Lin M, Chen Q, Yan S. Network in network, ICLR 2014.

Network in Network

- Remove the two fully connected layers (fc6, fc7) of the AlexNet but add NIN into the AlexNet.



	Parameter Number	Performance	Time to train (GTX Titan)
AlexNet	60 Million (230 Megabytes)	40.7% (Top 1)	8 days
NIN	7.5 Million (29 Megabytes)	39.2% (Top 1)	4 days

Lin M, Chen Q, Yan S. Network in network, ICLR 2014.

GoogLeNet: ILSVRC 2014 winner

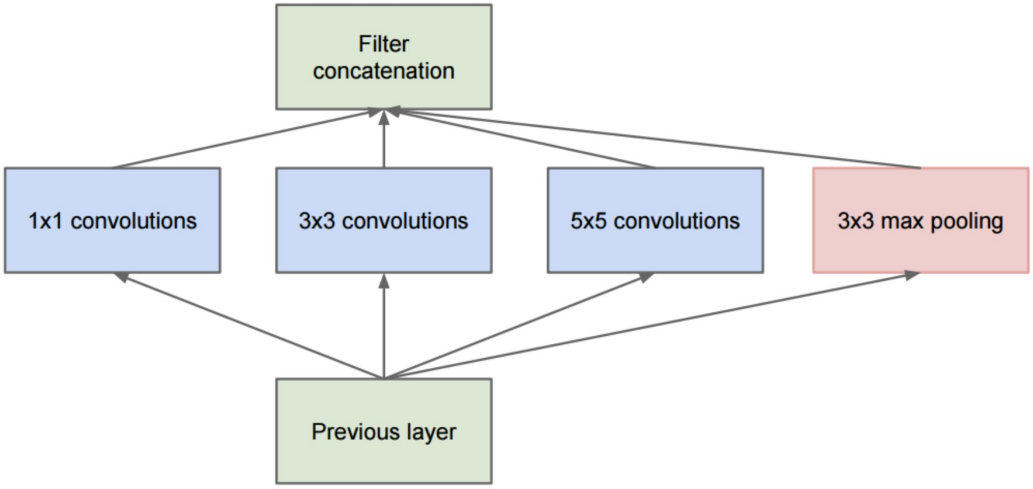
- The Inception Module

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

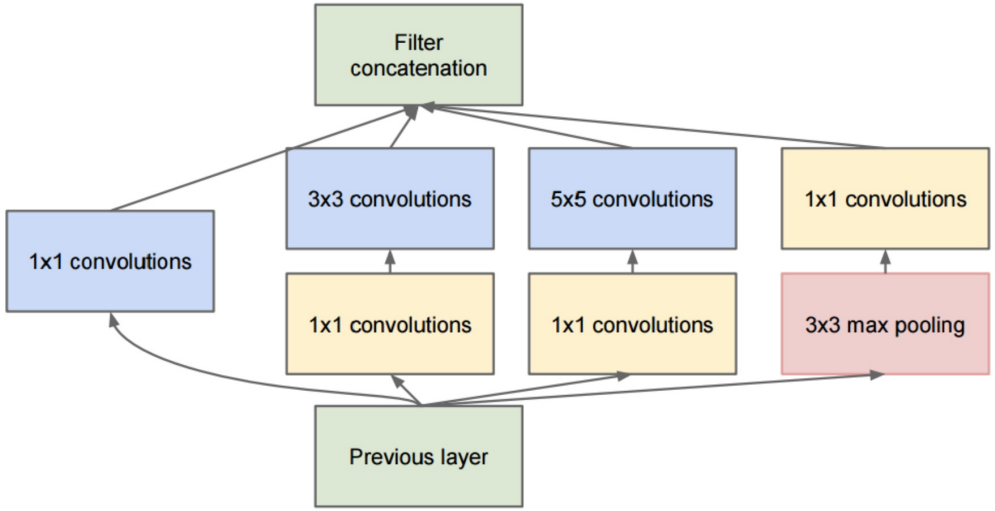


Inception module, naïve version

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

GoogLeNet: ILSVRC 2014 winner

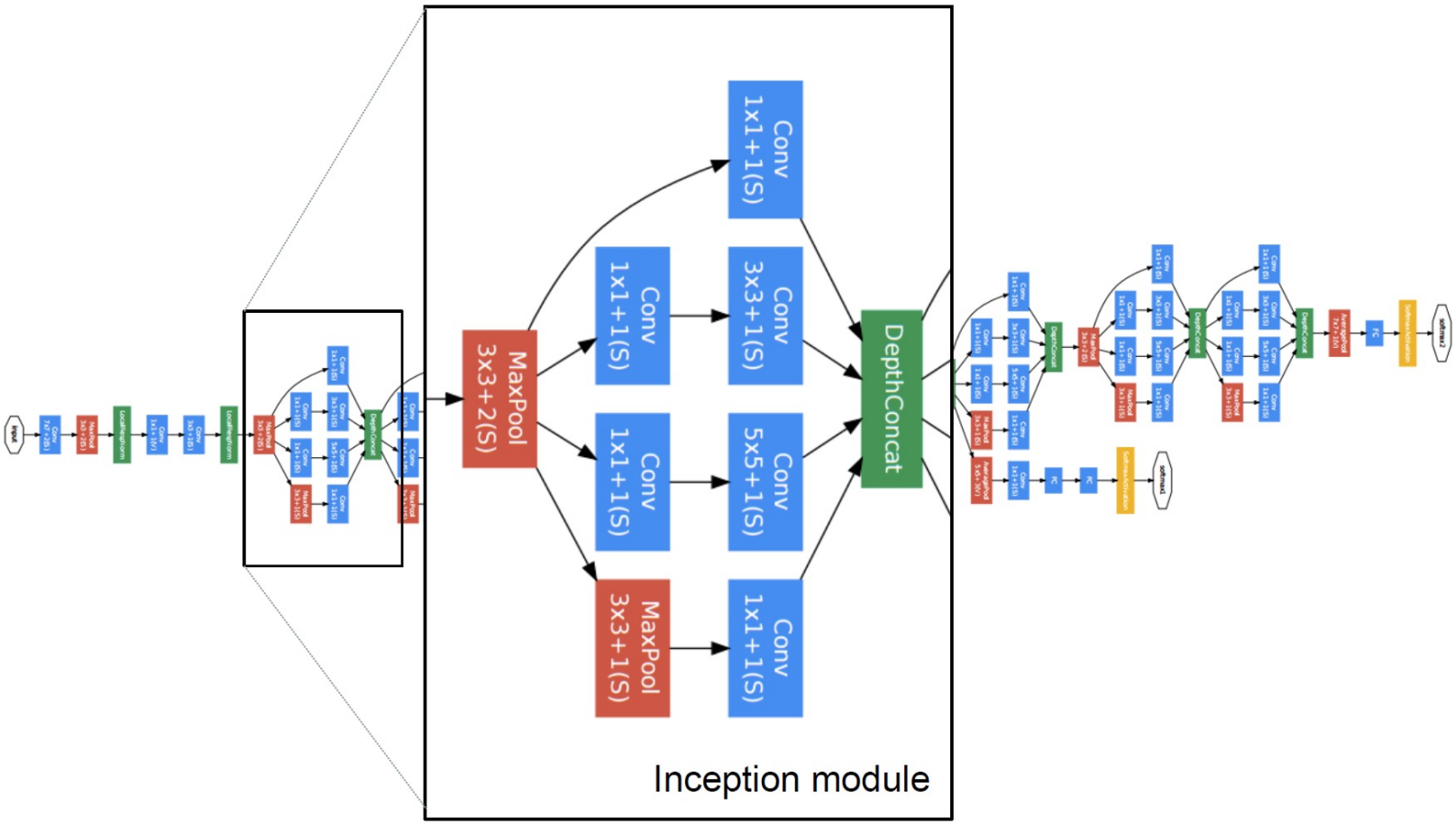
- The Inception Module
  - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
  - Use 1x1 convolutions for dimensionality reduction before expensive convolutions



Inception module with dimension reductions

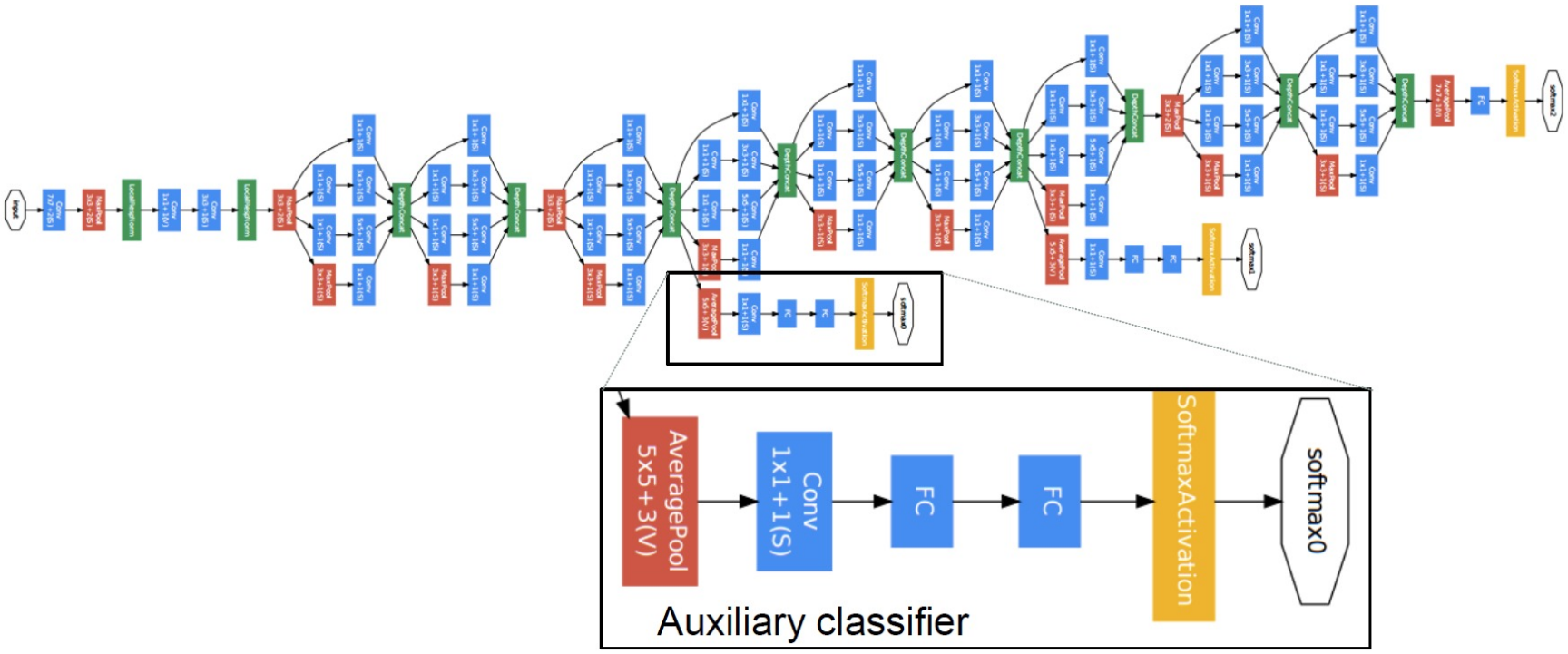
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

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

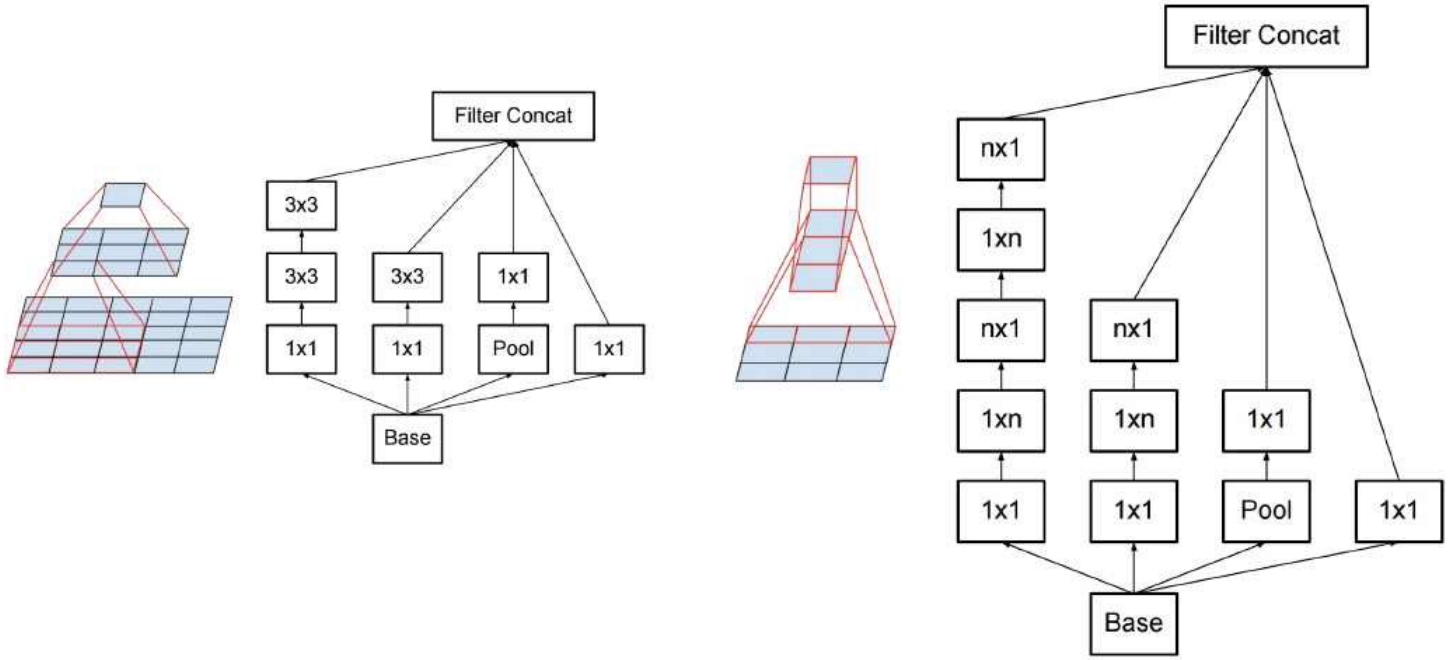
An alternative view:

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

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

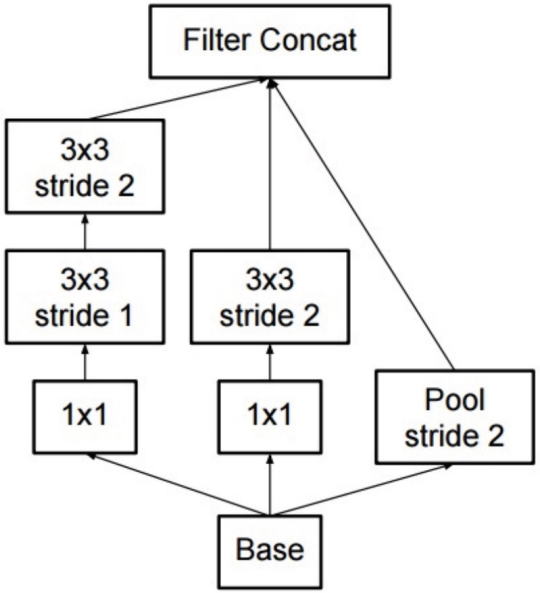
Inception v2, v3



- 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

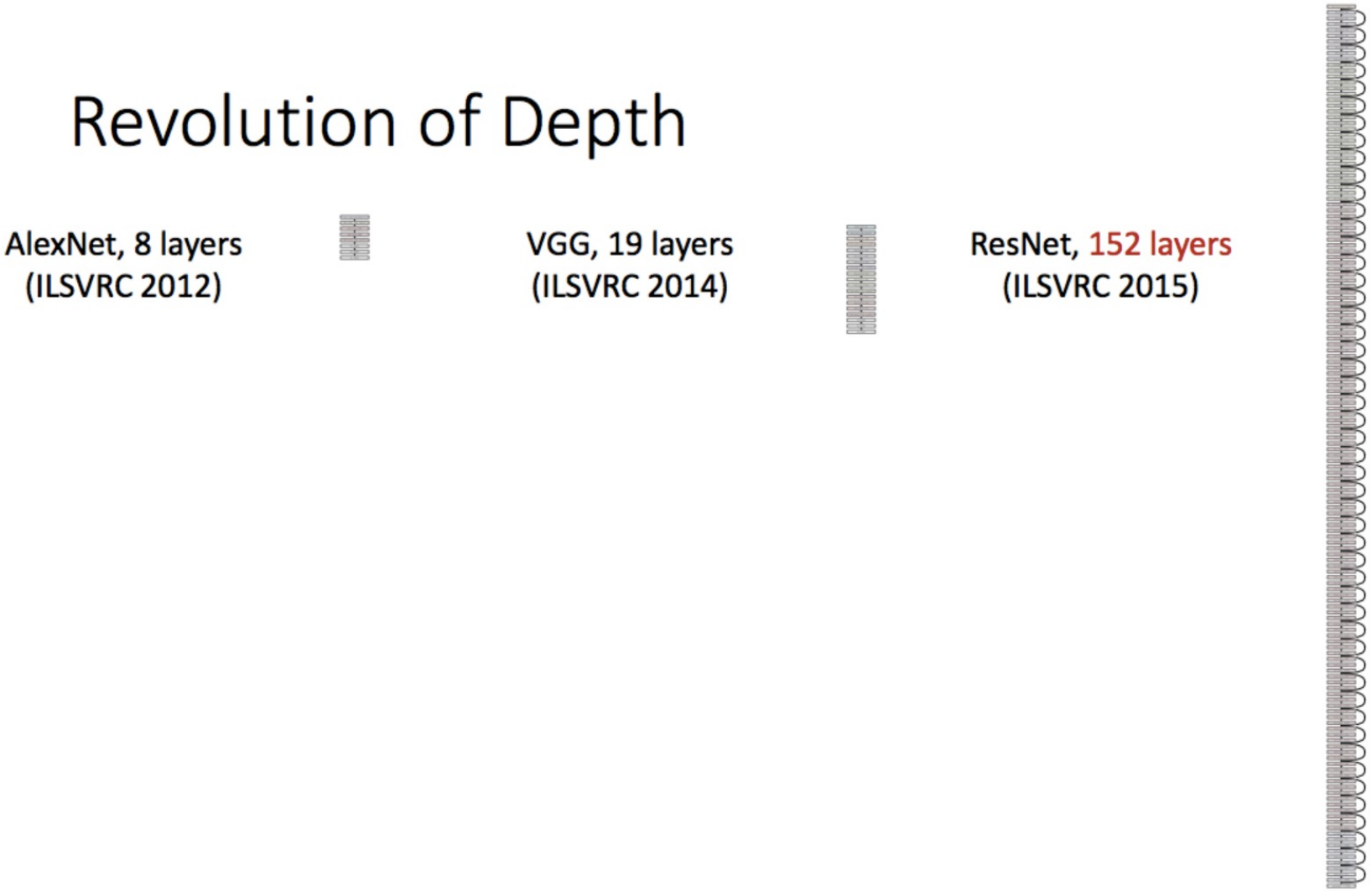
Inception v2, v3



- 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)

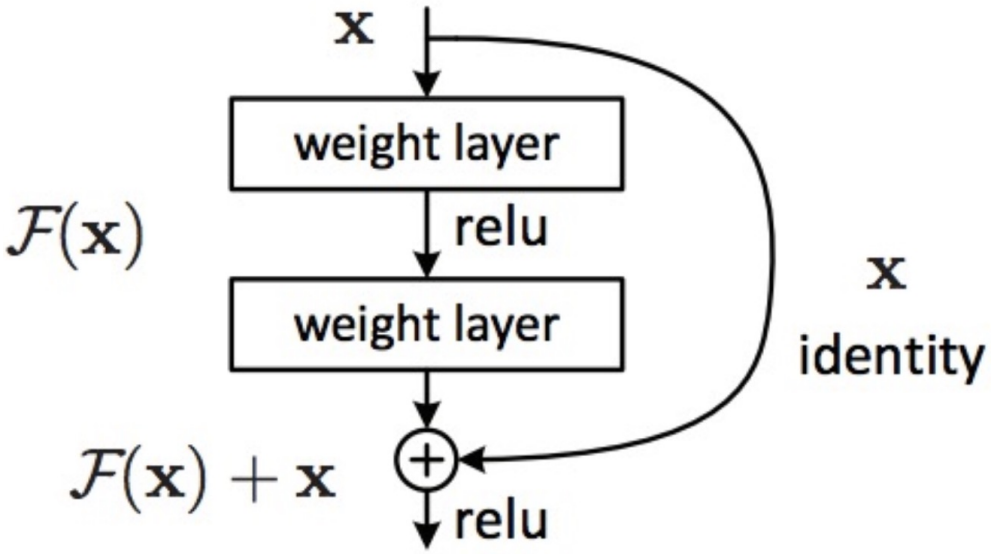
C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016

ResNet: ILSVRC 2015 winner



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				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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)

DenseNet

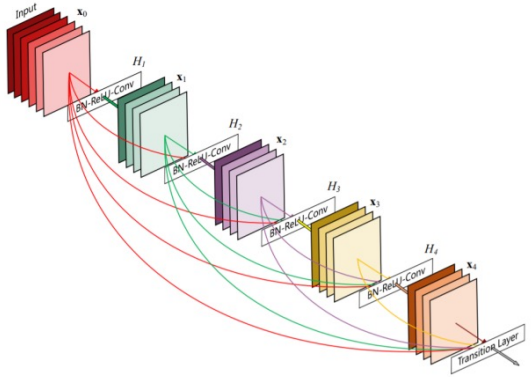
- ResNet solve the gradient vanishing problem by converting the feature mapping equation with **identity addition**

$$x_l = H_l(x_{l-1}) \quad \rightarrow \quad x_l = H_l(x_{l-1}) + x_{l-1}$$

- DenseNets do not sum the output feature maps of the layer with the incoming feature maps but **concatenate** them

$$x_l = H_l([x_0, x_1, \cdots, x_{l-1}])$$

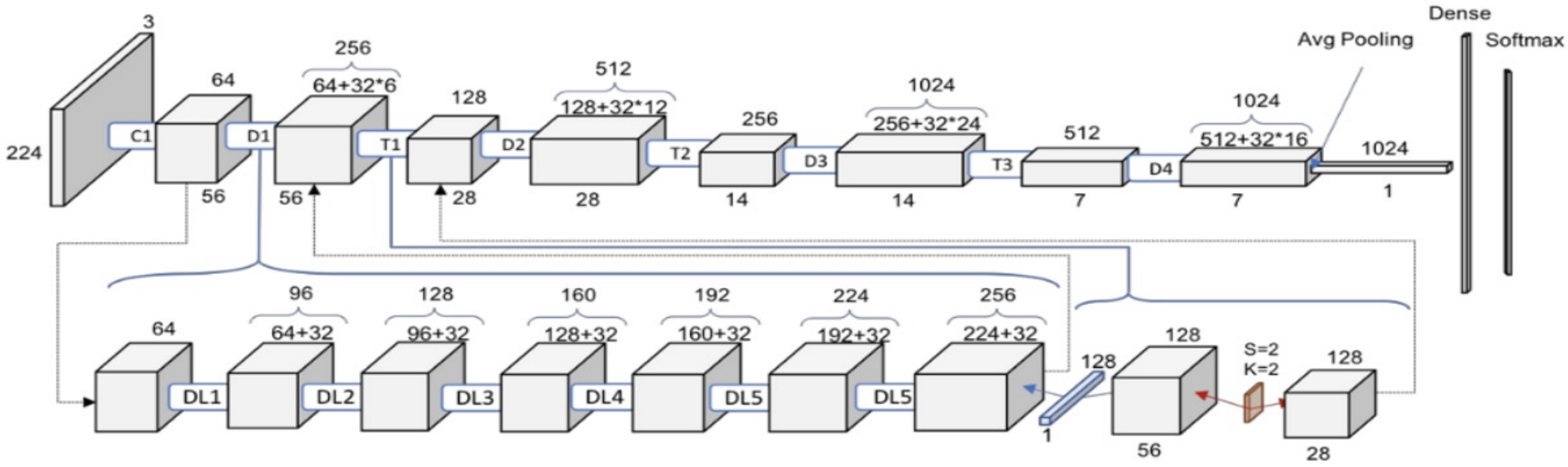
- Every layer has access to its preceding feature maps, and therefore, to the collective knowledge



Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks, CVPR 2017. (Best paper)

DenseNet

- DenseNets are divided into Dense Blocks, where the spatial dimensions of the feature maps remains constant within a block, but the number of filters changes between them
- The feature volume within a dense block remains constant



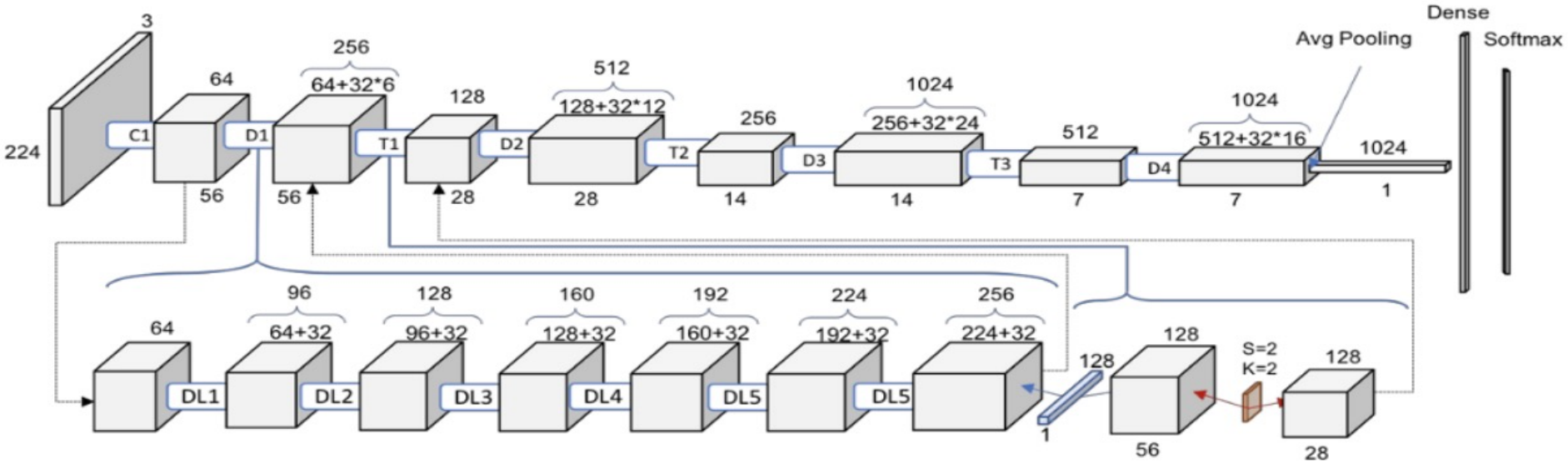
Dense-121. Dx: Dense Block x. Tx: Transition Block x.

Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks, CVPR 2017. (Best paper)

DenseNet

- There is a transition block follows every dense block, which has 1 × 1 convolution that halves the number of feature maps followed by a 2 × 2 pooling with a stride of 2

- The volume and the feature maps are halved after every transition block



Dense-121. Dx: Dense Block x. Tx: Transition Block x.

Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks, CVPR 2017. (Best paper)

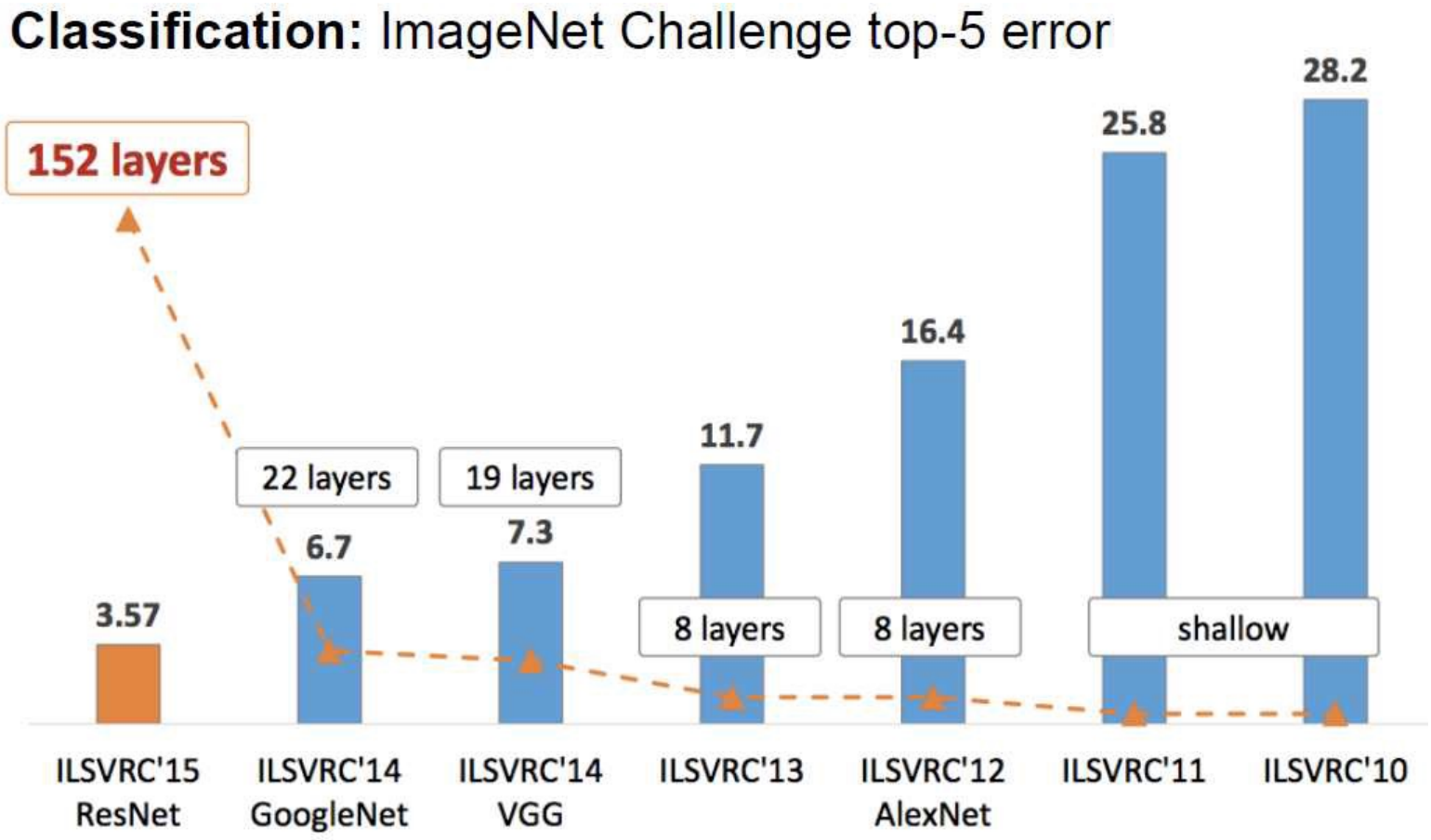
DenseNet

- **Different DenseNet structures**

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks, CVPR 2017. (Best paper)

Going Deeper



Outline

1. Course Review
2. Basic Operations
3. Major Architectures
4. CNN for Image Classification
5. Know More about CNN

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. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

ImageNet

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

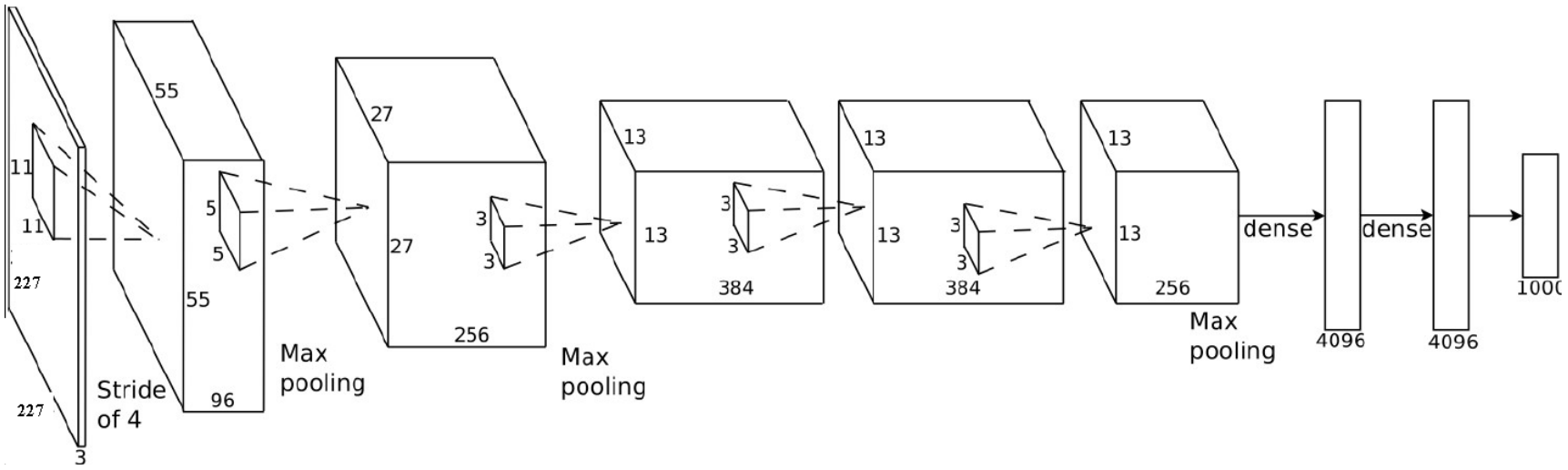
ImageNet

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

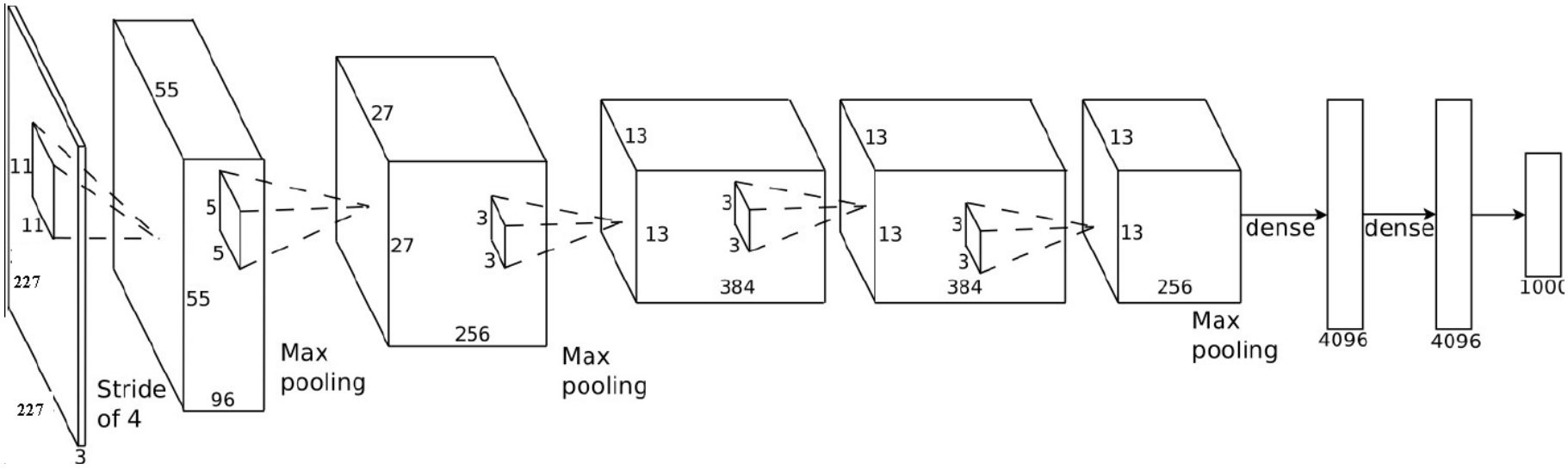
Review the architecture of AlexNet

- 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 154587 (input layer), 290400, 186624, 64896, 64896, 43264, 4096, 4096, 1000
- 813859 neurons(include input layer), 60965224 parameters



Example-AlexNet

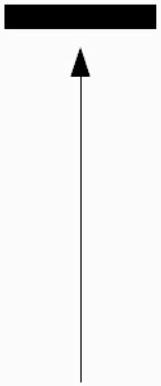
- The first time deep model is shown to be effective on large scale computer vision task.
- The first time a very large scale deep model is adopted.
- GPU is shown to be very effective on this large deep model.



Technical details

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

An input image (256x256)

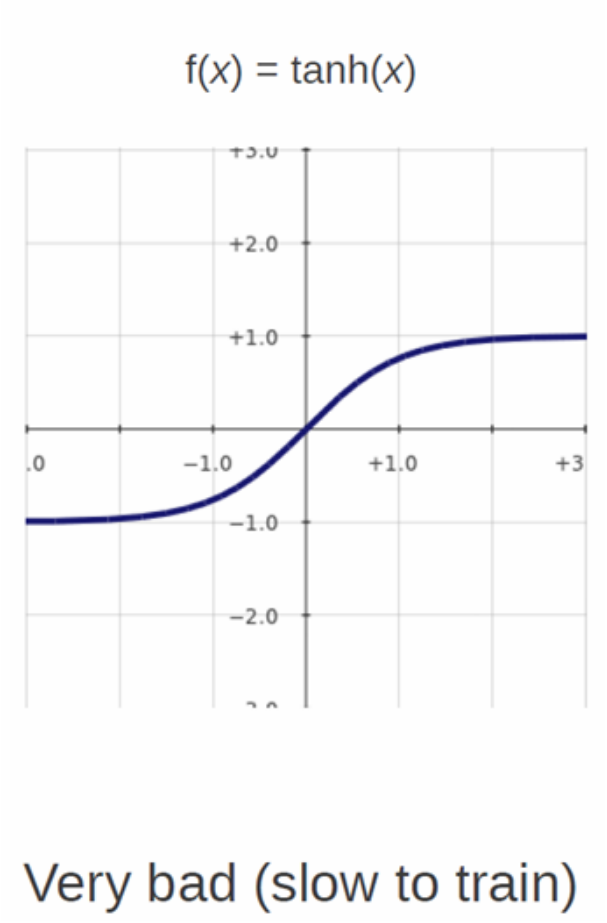


Minus sign

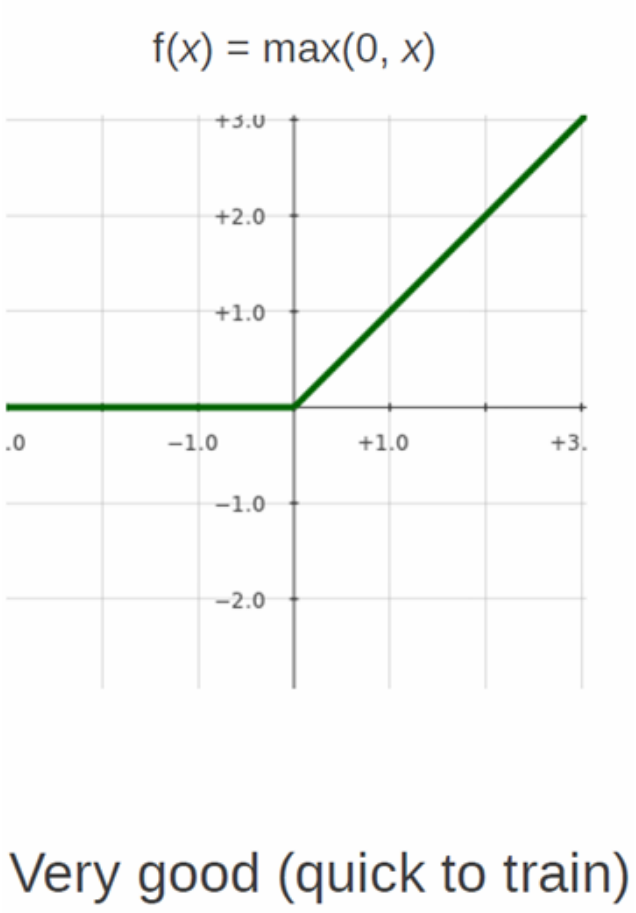
The mean input image

Technical details

- Choice of activation function



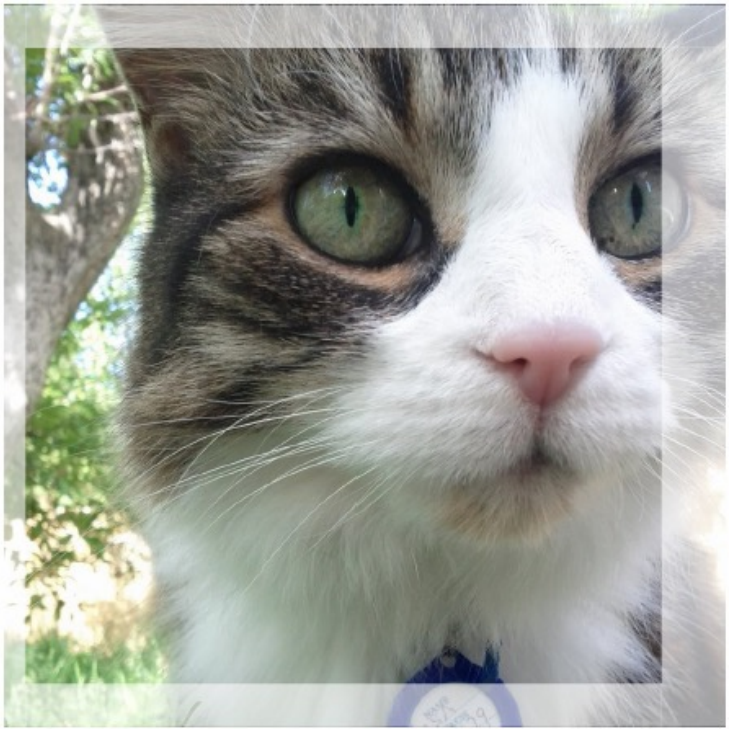
Very bad (slow to train)



Very good (quick to train)

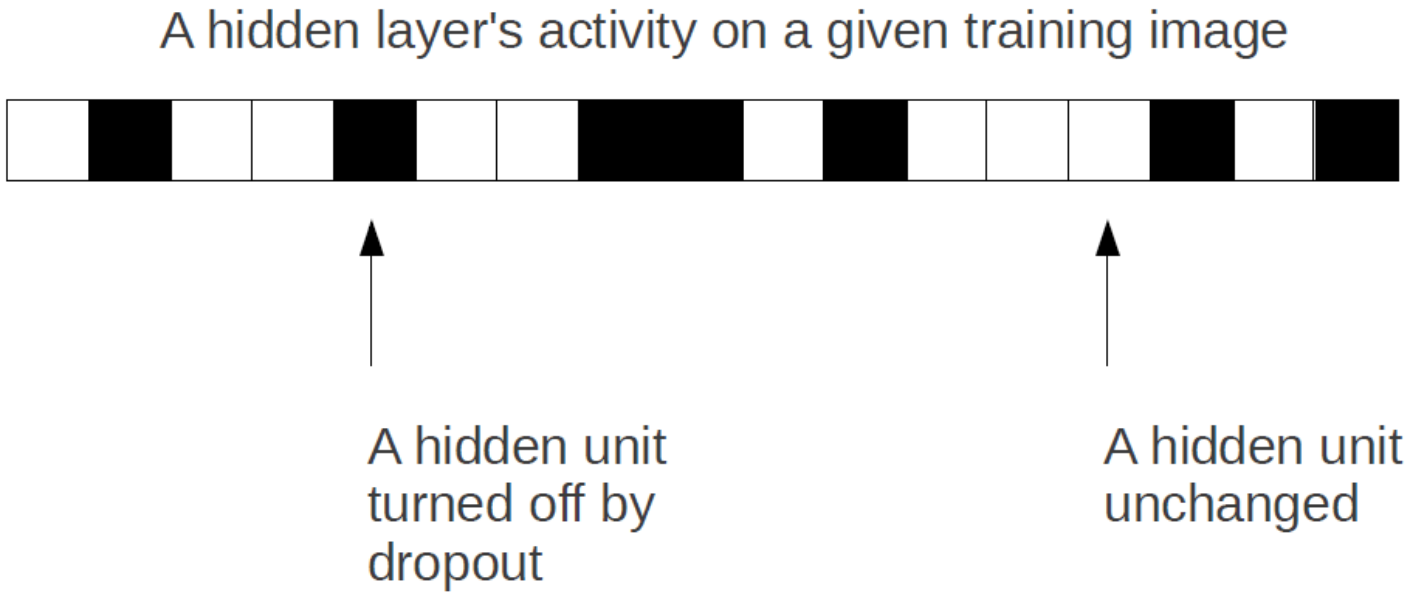
Technical details

- Data augmentation
  - The neural net has 60,965,224 real-valued parameters and 813859 neurons(include input layer)
  - It overfits a lot. 227 x 227 image regions are randomly extracted from 256 images, and also their horizontal reflections

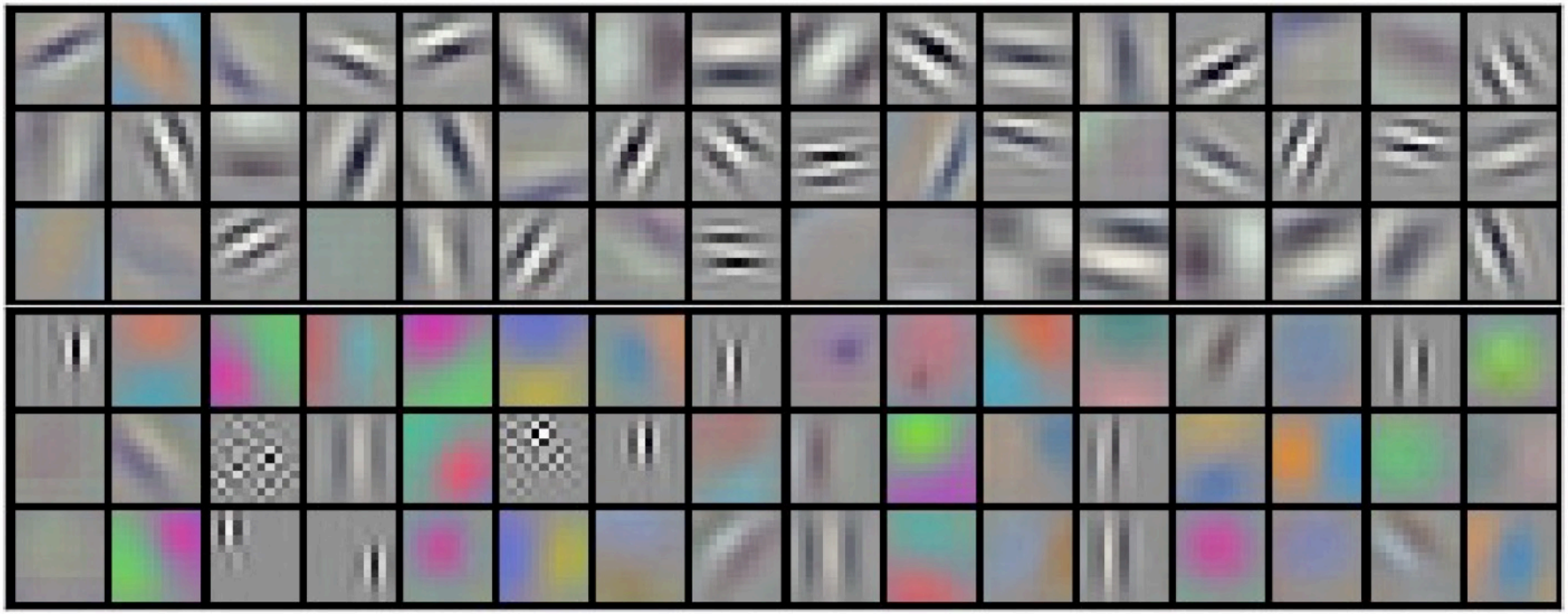


Technical details

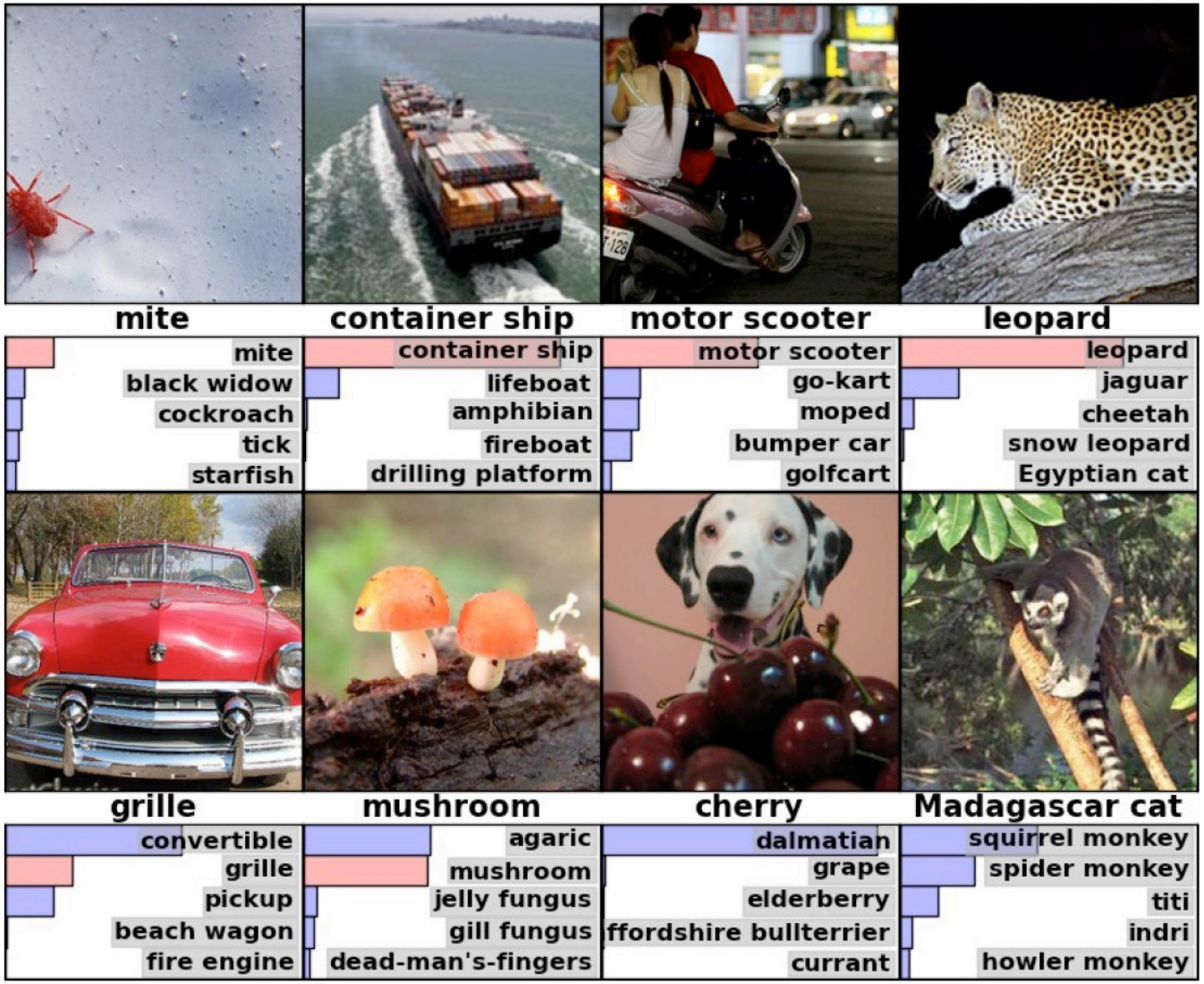
- Dropout
  - Independently set each hidden unit activity to zero with 0.5 probability
  - Do this in the two globally-connected hidden layers at the net's output



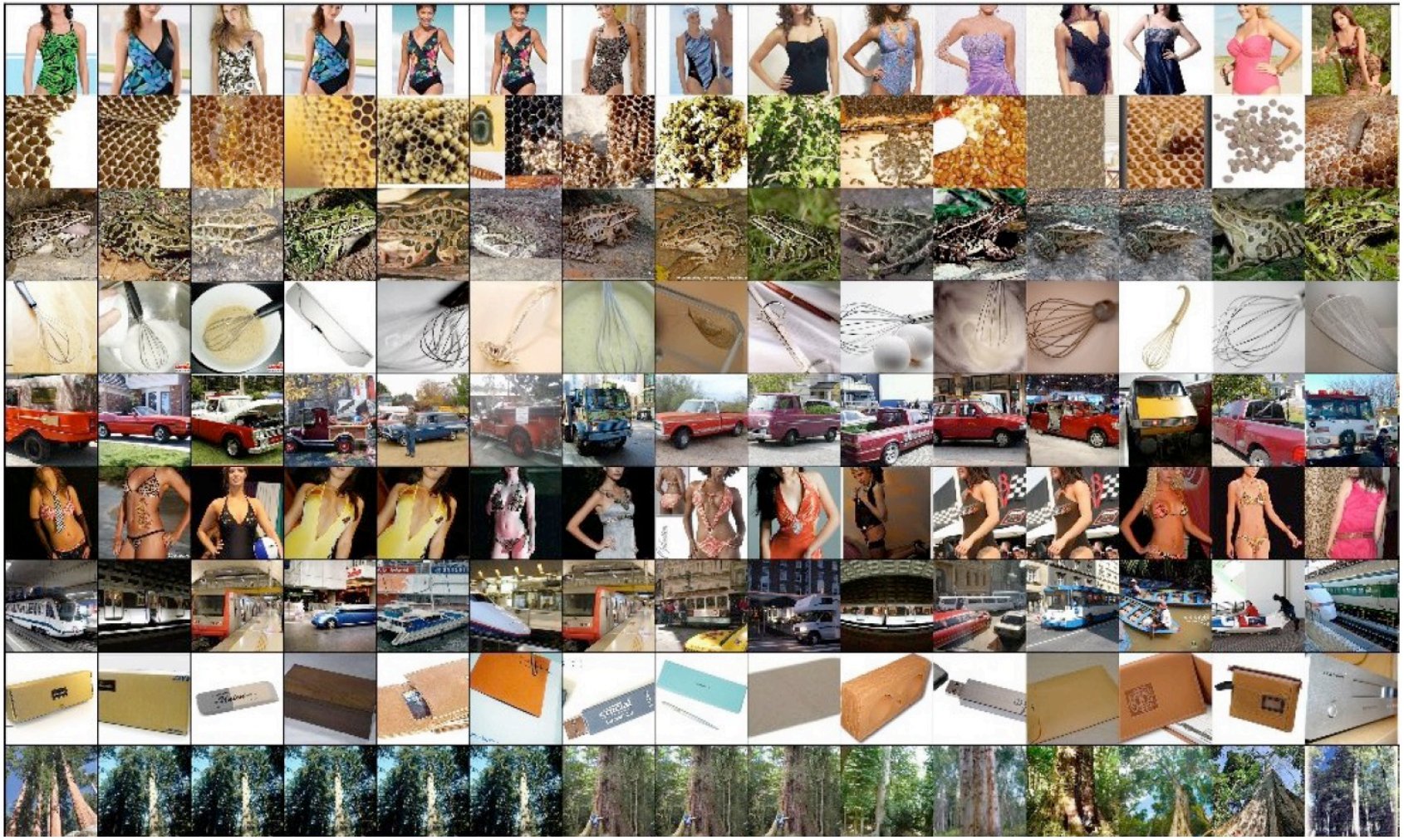
96 learned low-level filters



Classification result



Top hidden layer can be used as feature for retrieval



Outline

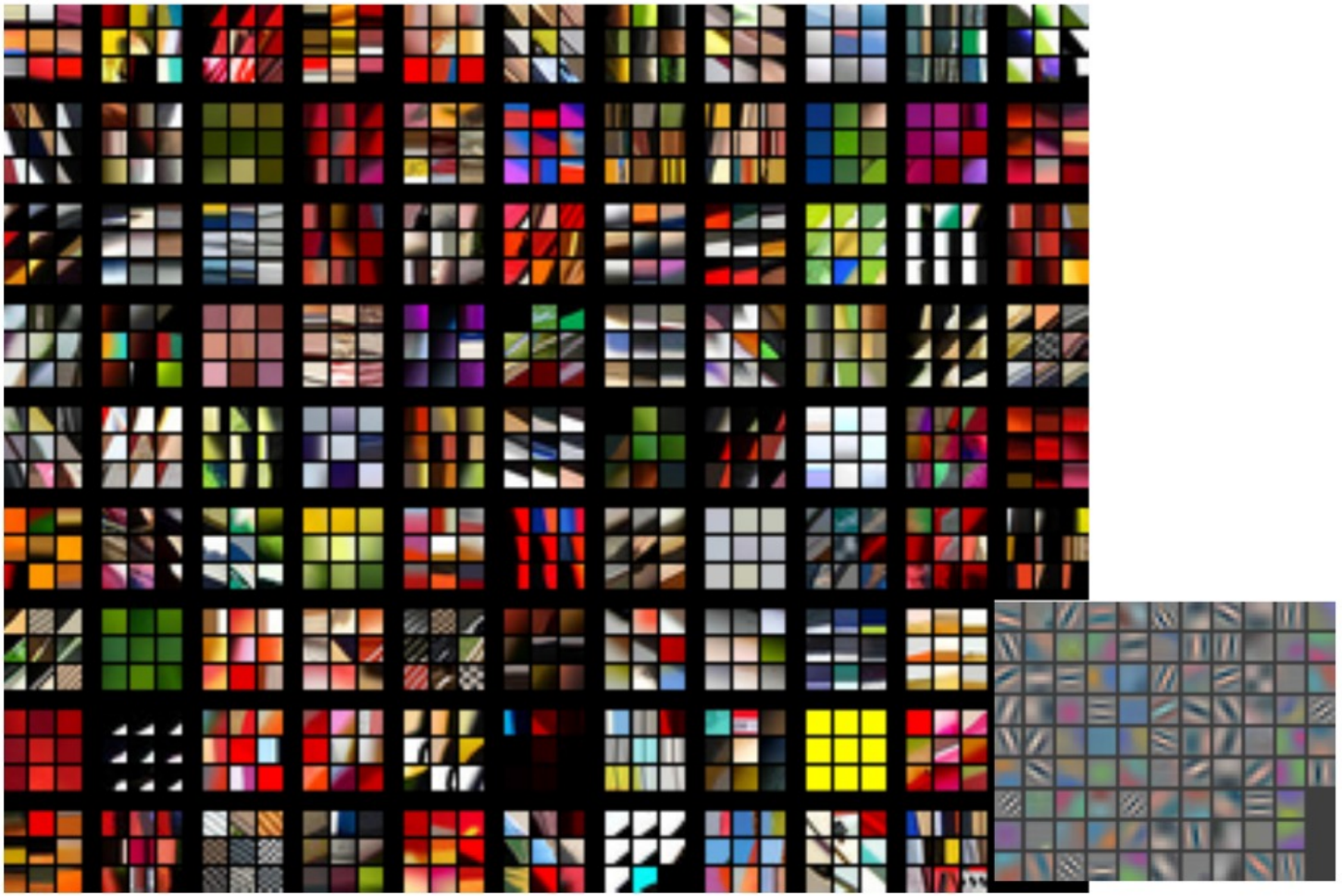
1. Course Review
2. Basic Operations
3. Major Architectures
4. CNN for Image Classification
5. Know More about CNN

Know More about CNN

- How do the features look like ?
- How transferable are features in CNN networks?
- Investigate components of CNNs

Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

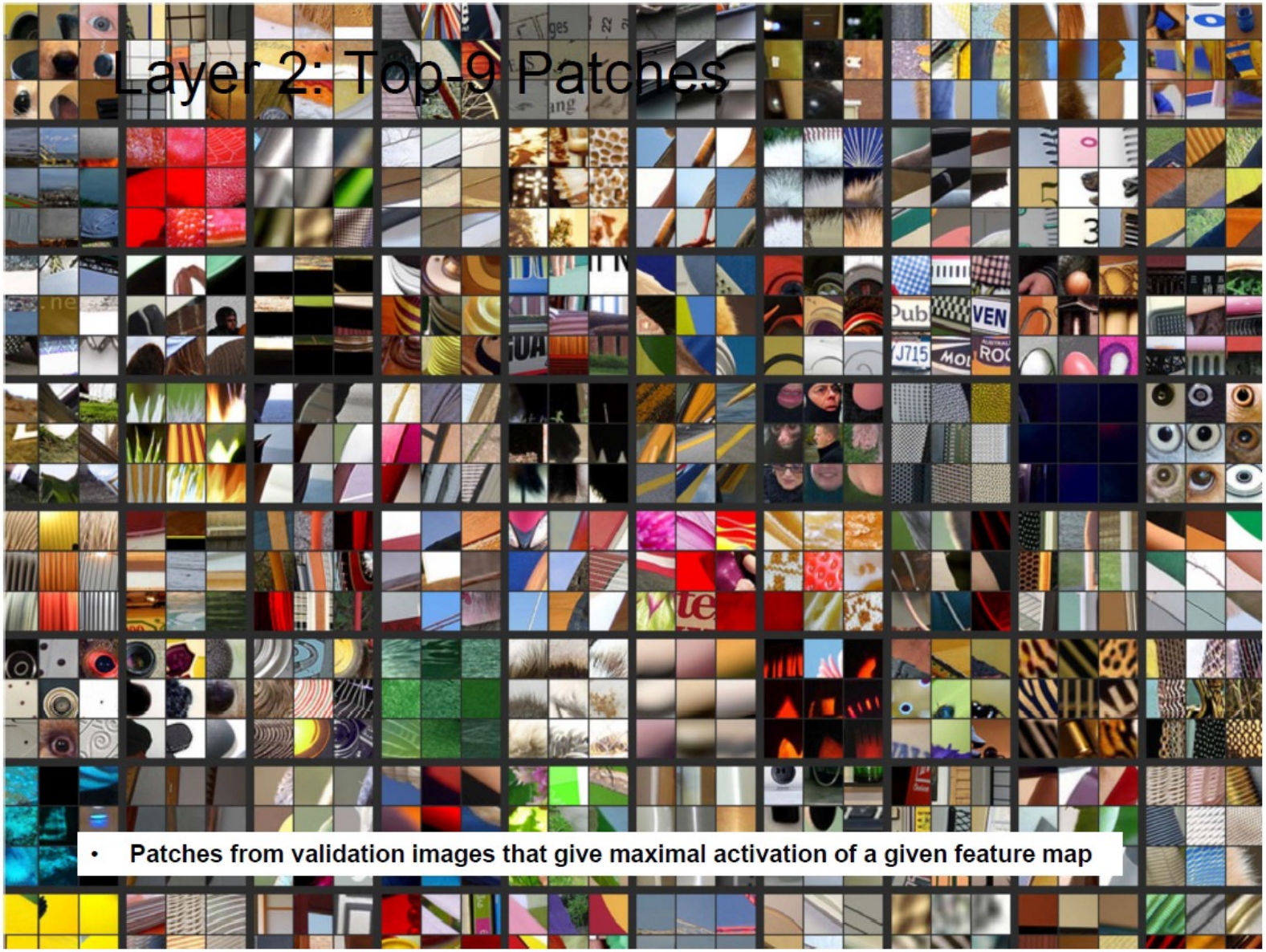
Layer 1 filters



This and the next few illustrations are from Rob Fergus

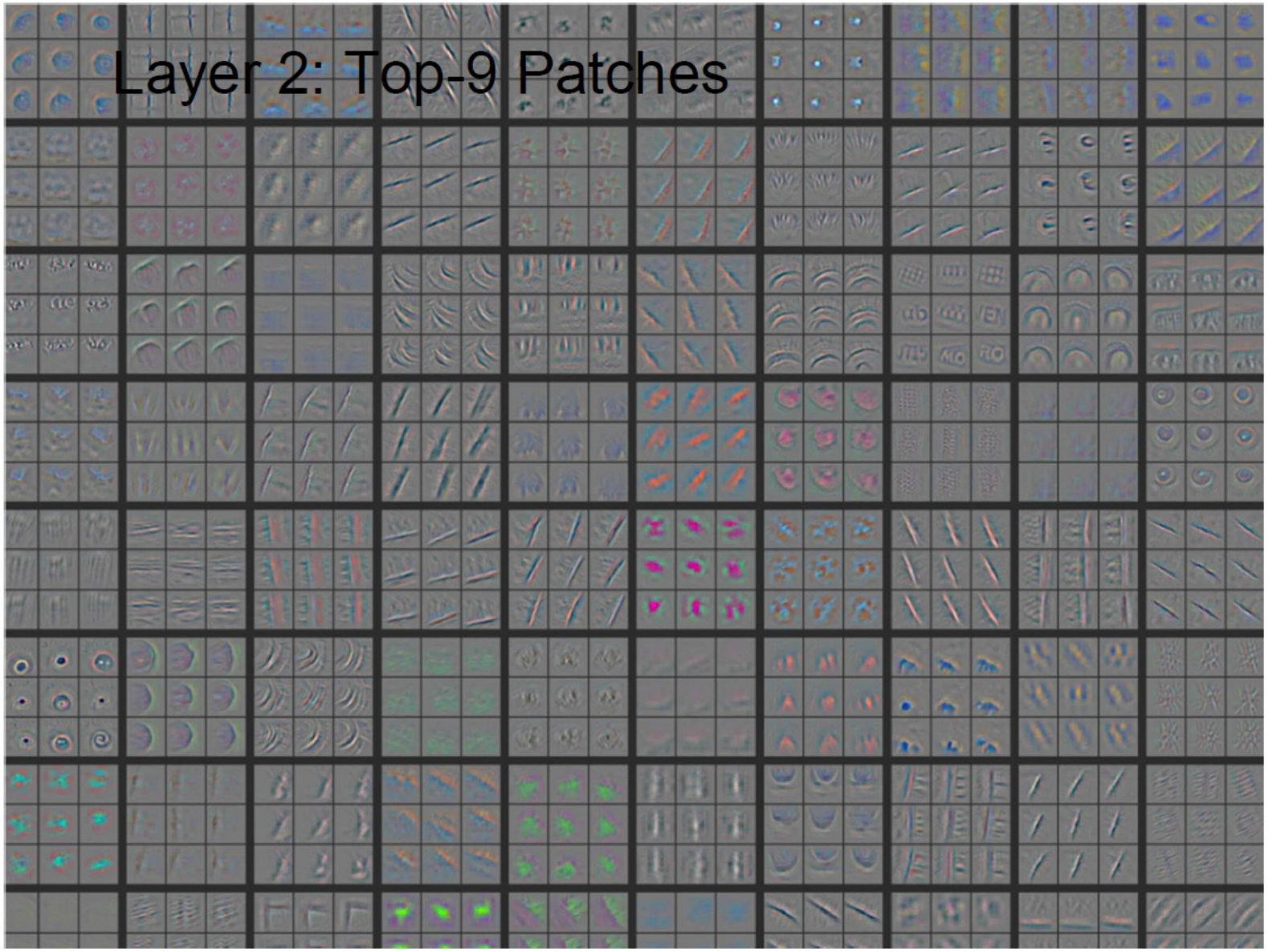
Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 2 Patches



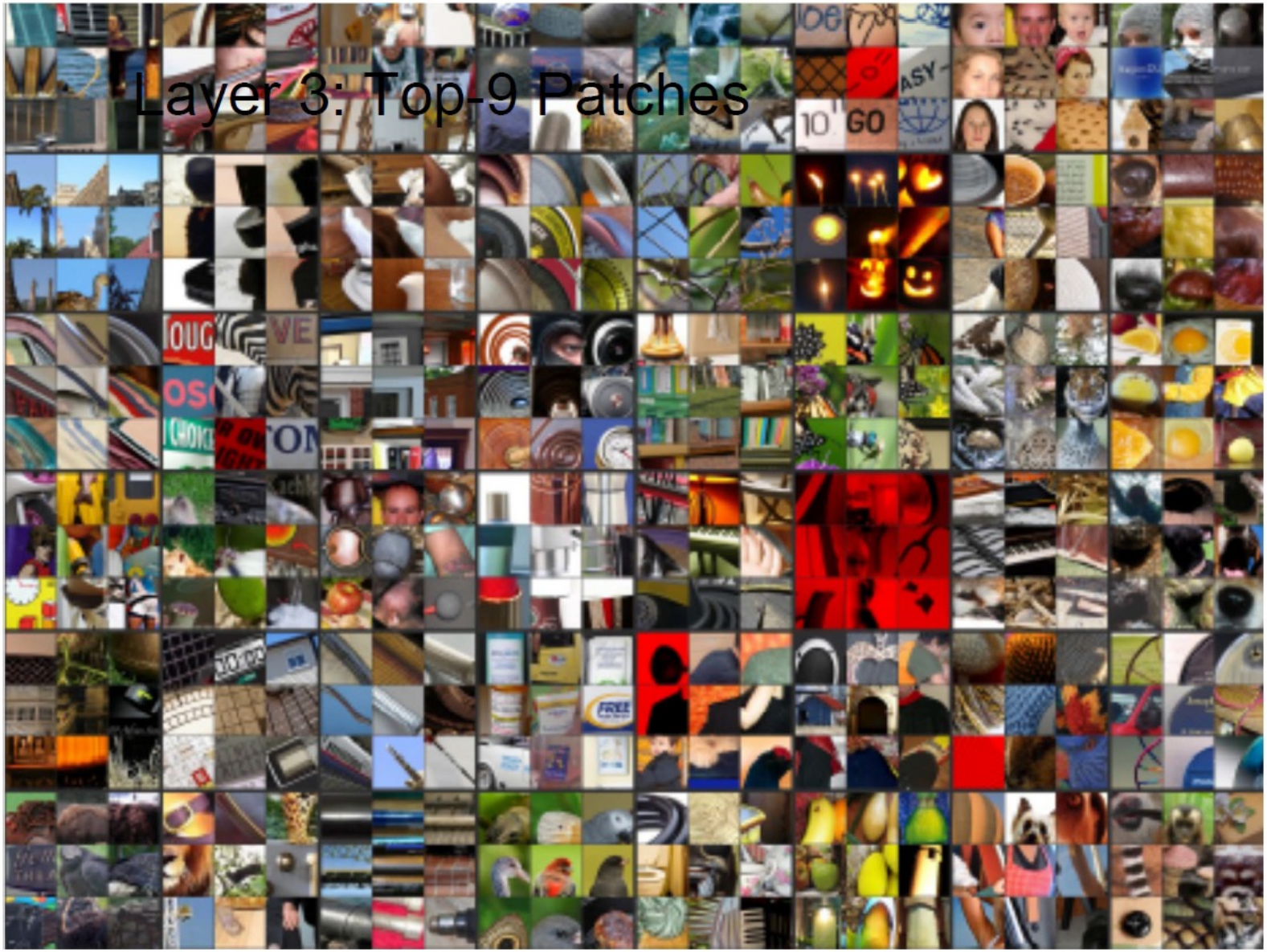
Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 2 Patches



Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 3 Patches

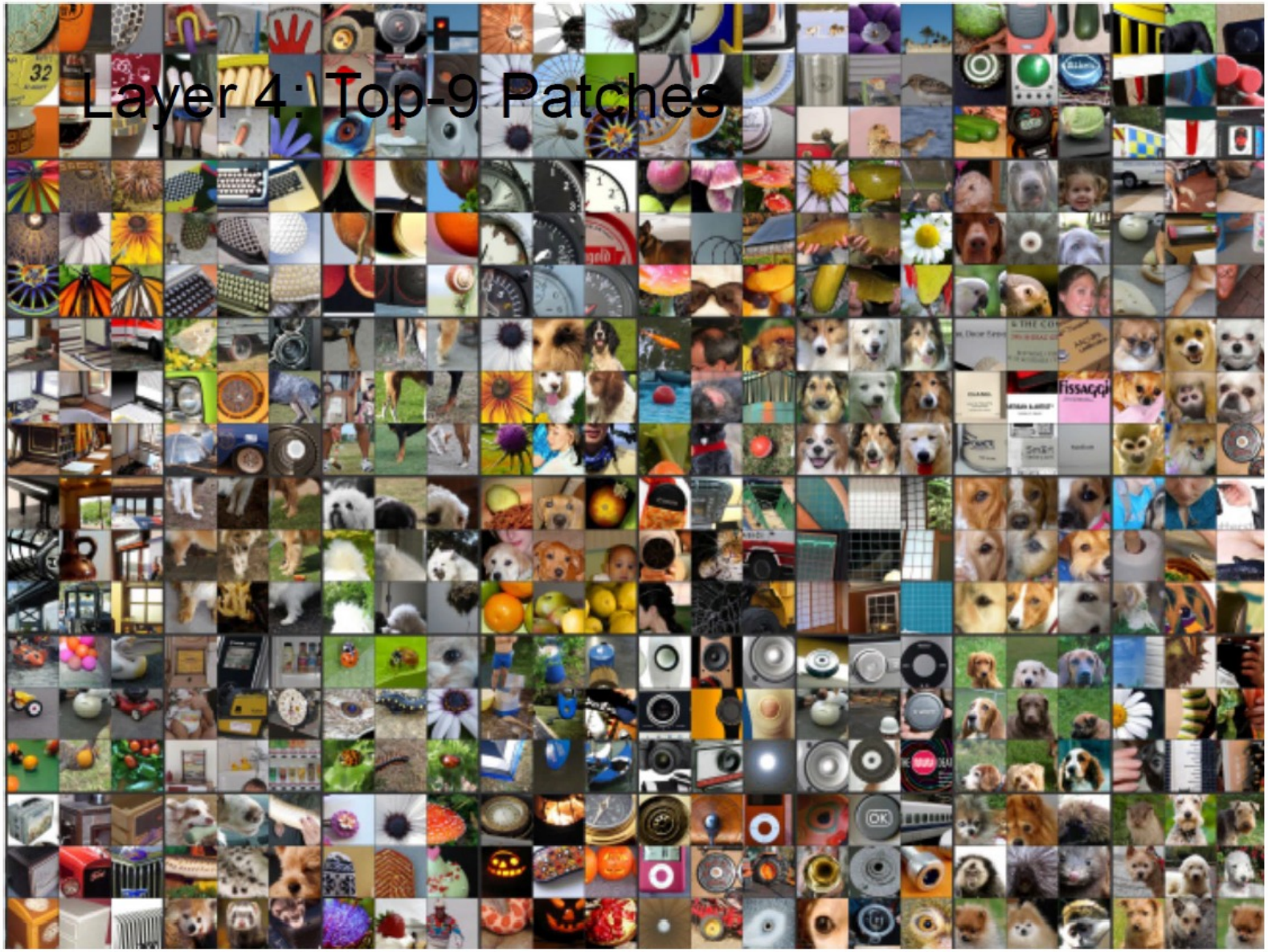


Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 3 Patches

Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 4 Patches

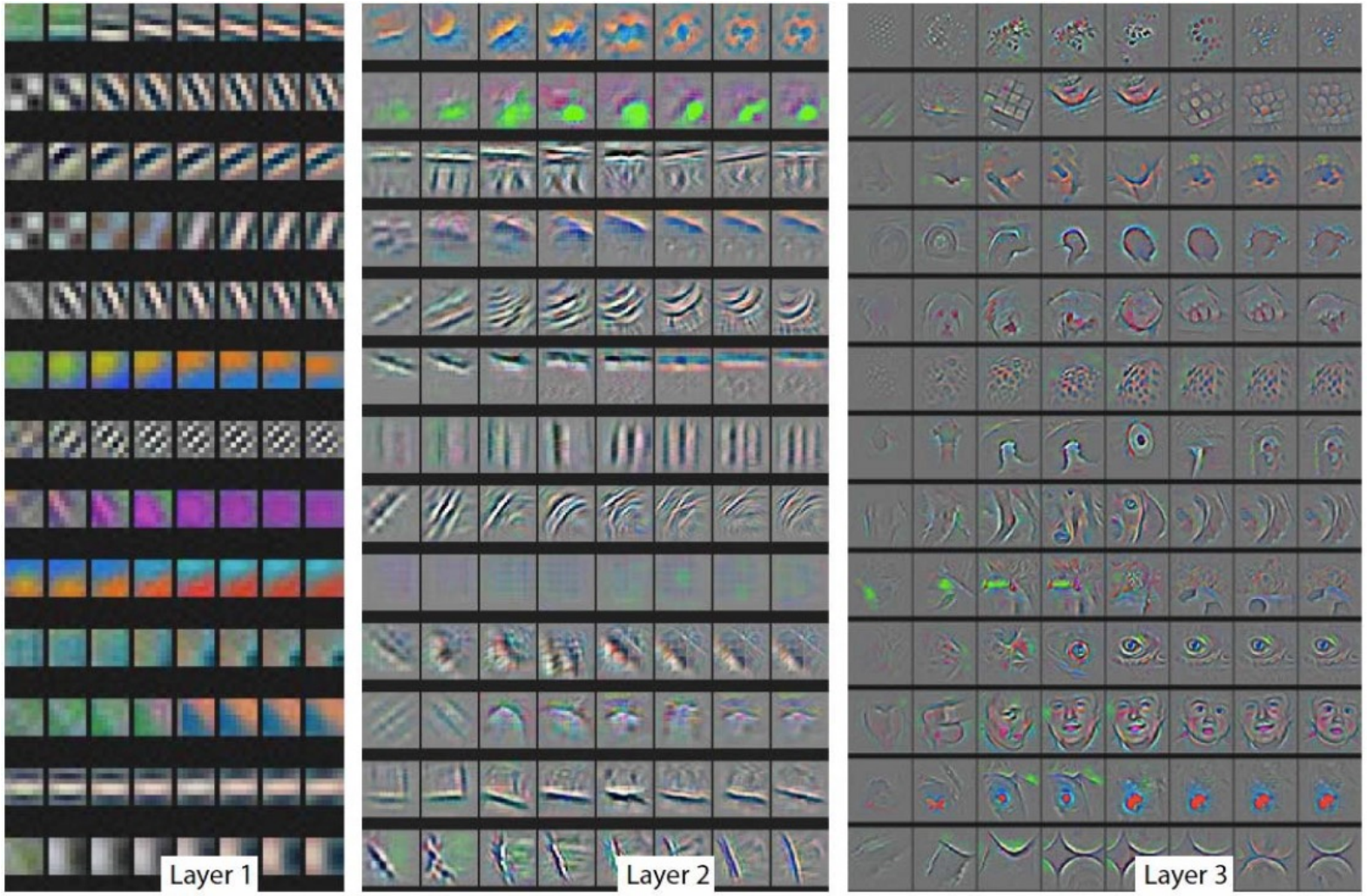


Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Layer 4 Patches

Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

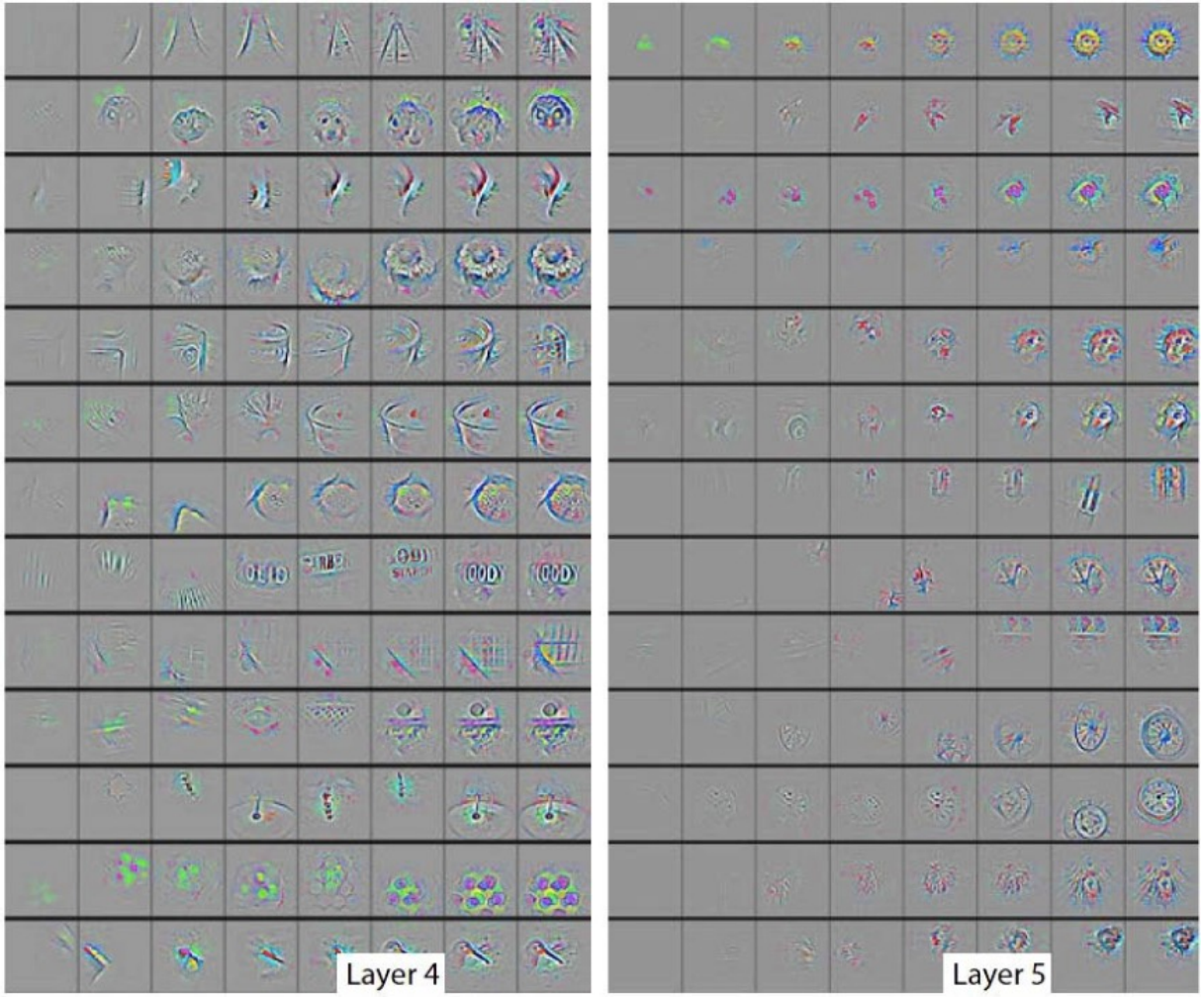
Evolution of Filters



Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Evolution of Filters

Continued



Zeiler M D, Fergus R. Visualizing and understanding convolutional networks, ECCV 2014.

Know More about CNN

- How do the features look like ?
- How transferable are features in CNN networks?
- Investigate components of CNNs

Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

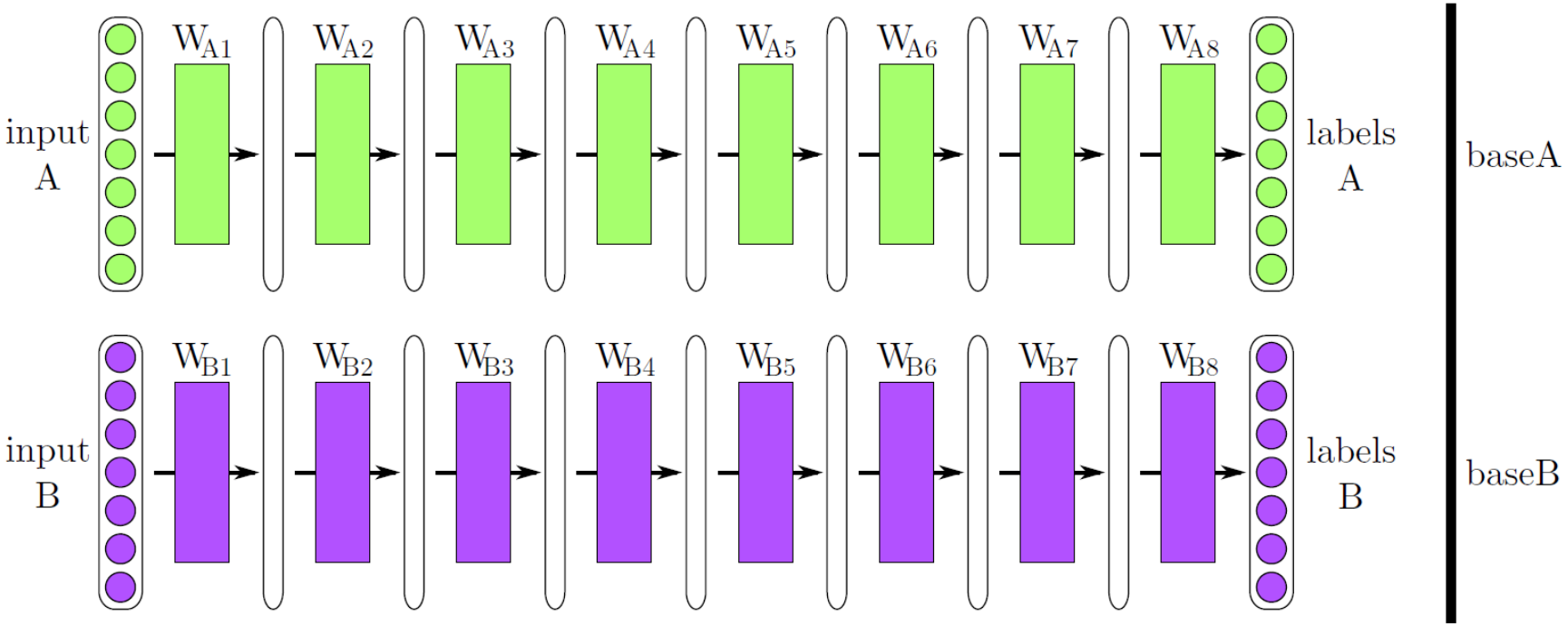
How transferable are features in CNN networks?

- (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

Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

Base tasks

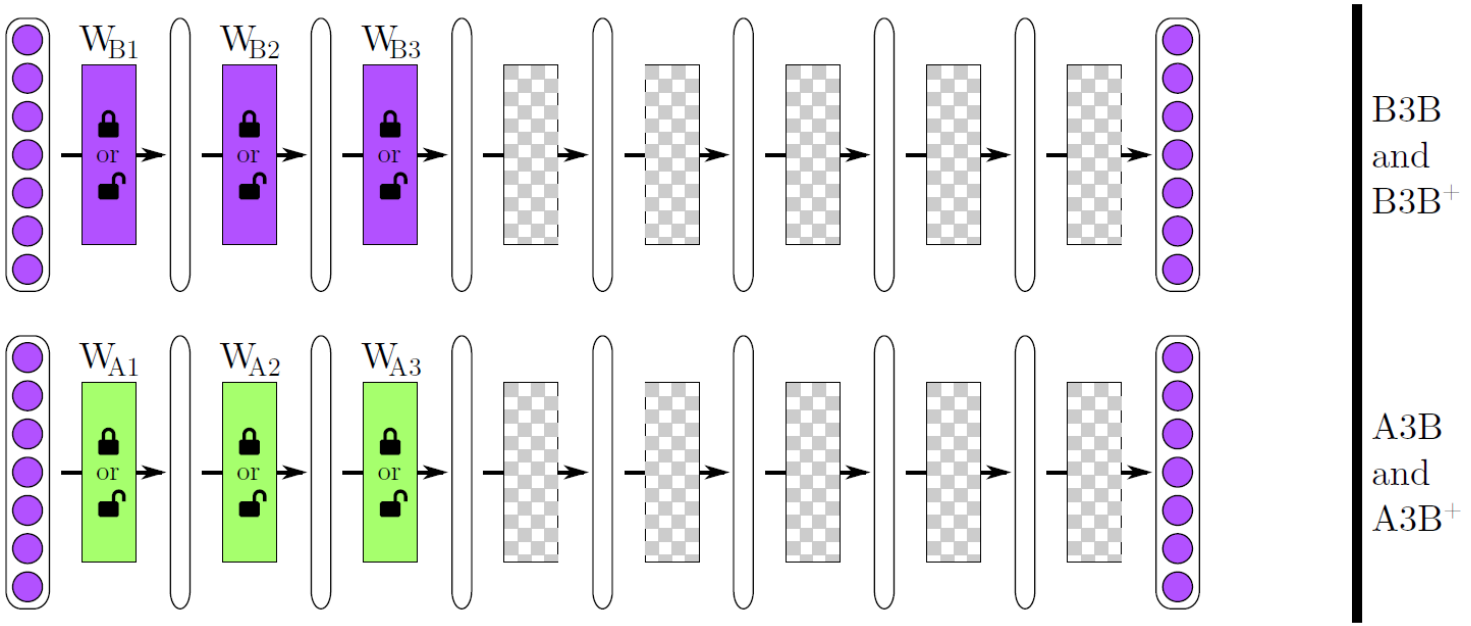
- ImageNet are divied into two groups of 500 classes, A and B
- Two 8-layer AlexNets, baseA and baseB, are trained on the two groups, respectively



Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

Transfer and selffer networks

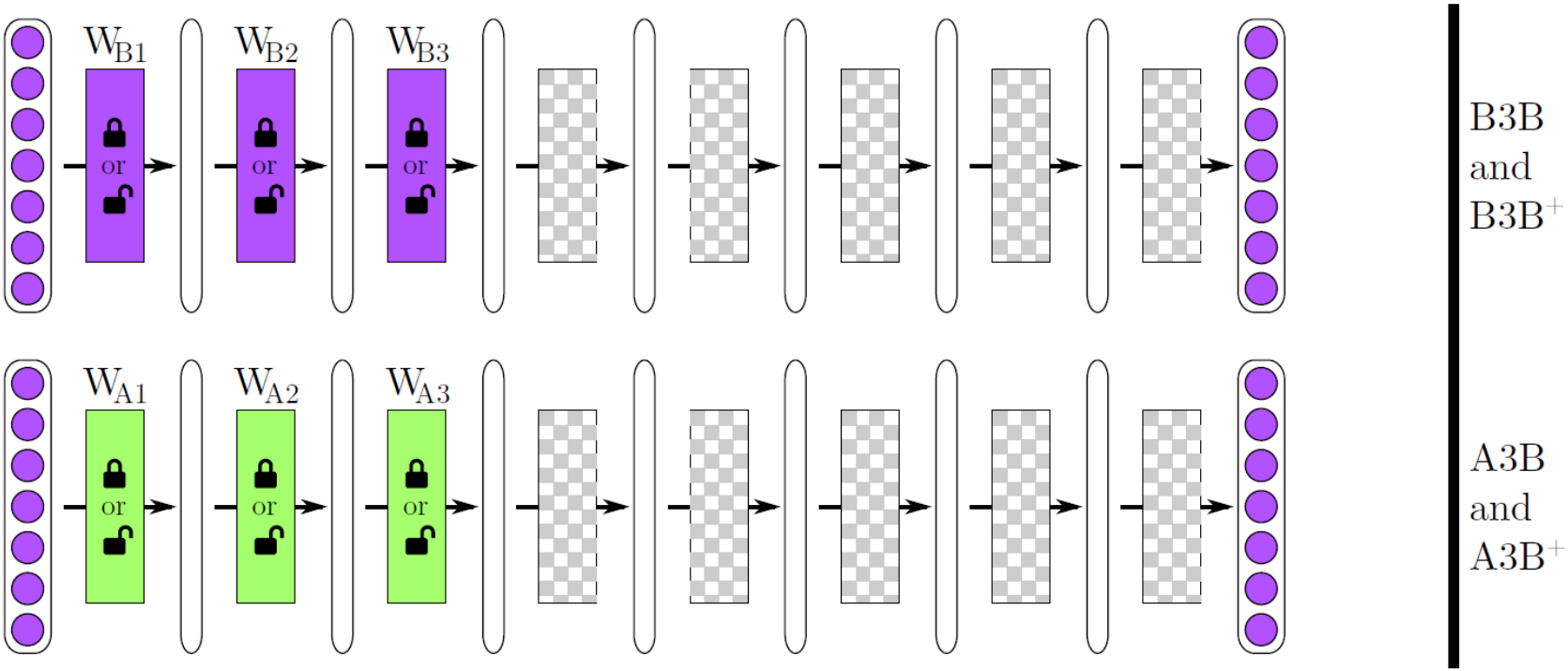
- A *selffer* network BnB: the first n layers are copied from baseB and frozen. The other higher layers are initialized randomly and trained on dataset B. This is the control for transfer network
- A *transfer* network AnB: the first n layers are copied from baseA and frozen. The other higher layers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



Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

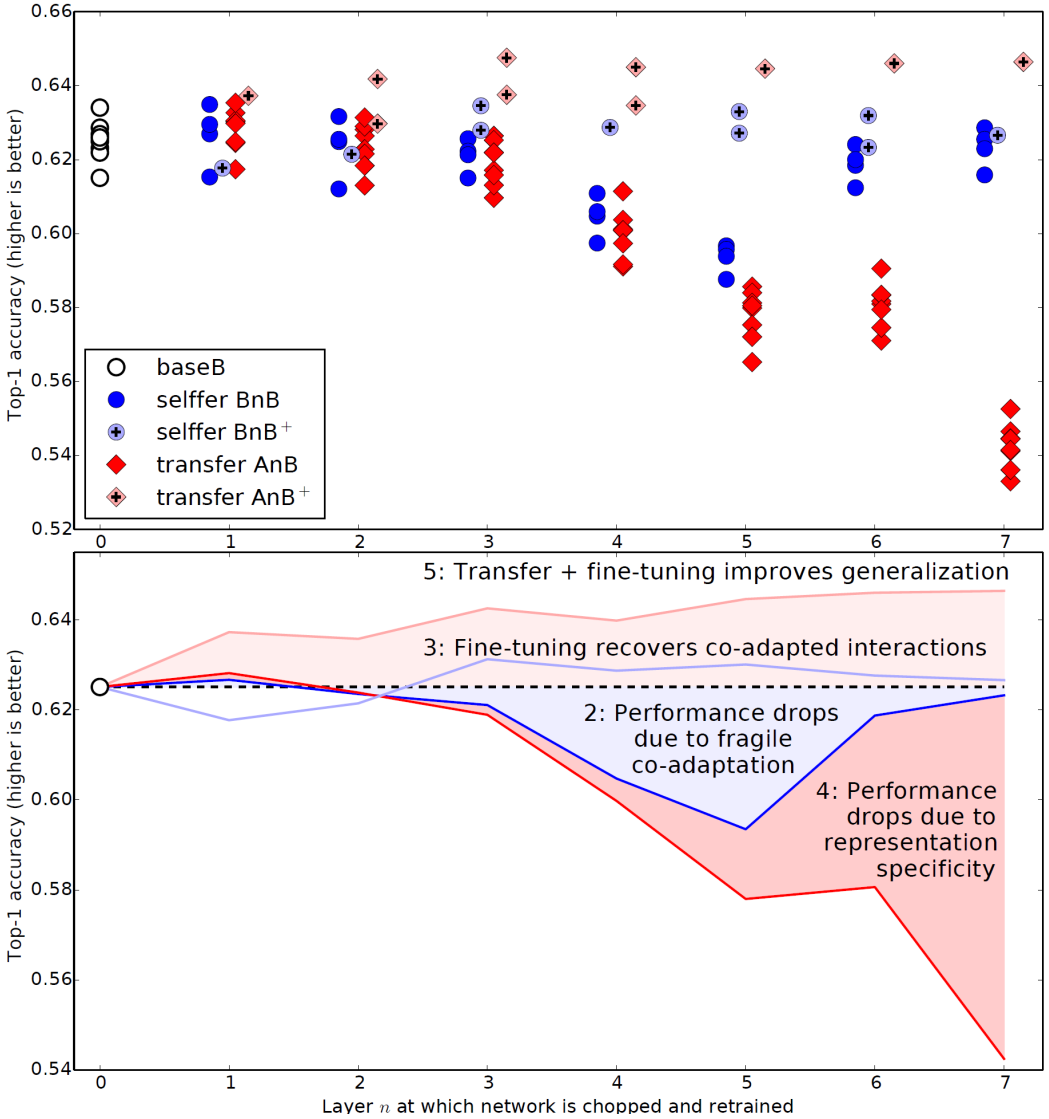
Transfer and selffer networks (cont'd)

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



Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

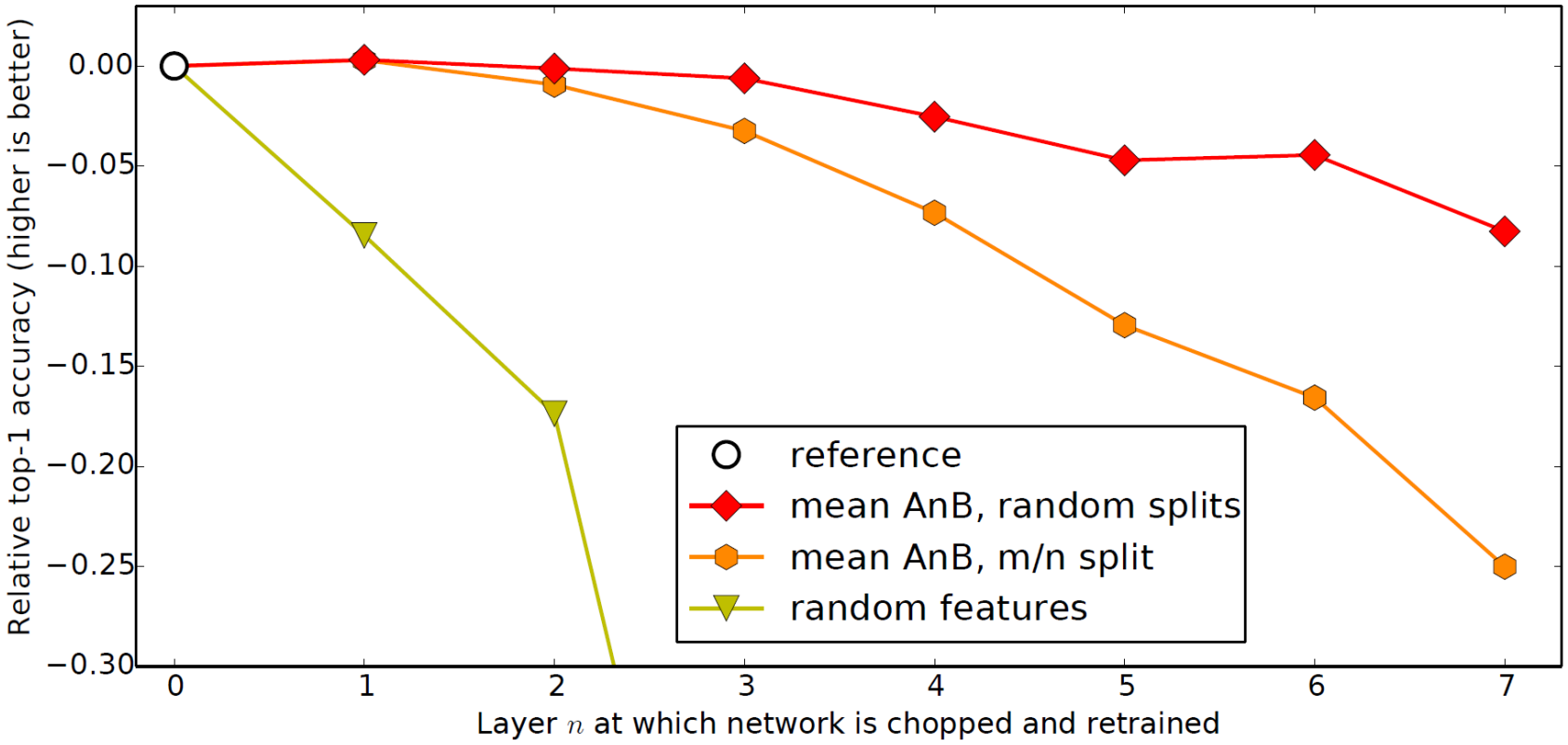
Results



Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

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



Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

Know More about CNN

- How do the features look like ?
- How transferable are features in CNN networks?
- Investigate components of CNNs

Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks?, NeurIPS 2014.

Investigate components of CNNs

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

Chatfield K, Simonyan K, Vedaldi A, et al. Return of the devil in the details: Delving deep into convolutional nets, BMVC 2014. (Best paper)

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

Chatfield K, Simonyan K, Vedaldi A, et al. Return of the devil in the details: Delving deep into convolutional nets, BMVC 2014. (Best paper)

Investigate components of CNNs (cont'd)

- Gray-scale vs. color (~ 3% drop)
- Decrease the number of nodes in FC7
  - to 2048 (surprisingly, marginally better)
  - to 1024 (marginally better)
  - to 128 (~ 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)
  - ~ 2:7% improvement
  - Note, L2 normalising features account for ~ 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%

Chatfield K, Simonyan K, Vedaldi A, et al. Return of the devil in the details: Delving deep into convolutional nets, BMVC 2014. (Best paper)

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 !