

lecture 7: convolution and architectures

deep learning for vision

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, Nov. 2017 – Jan. 2018



outline

introduction

definition and properties

variants and their derivatives

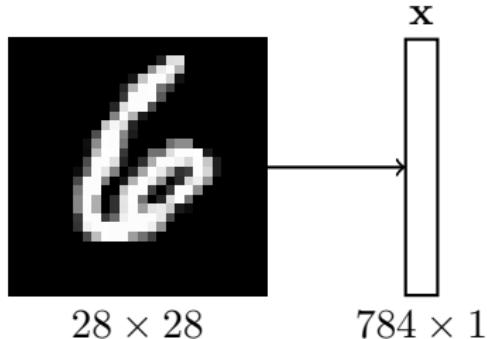
pooling

fun

network architectures

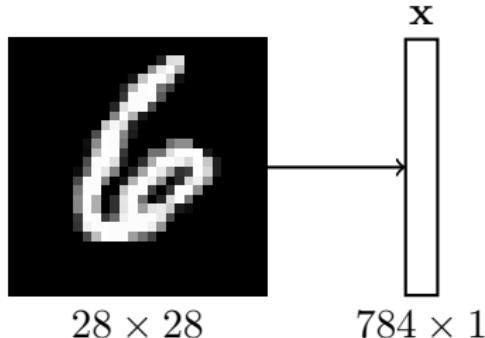
introduction

input image representation



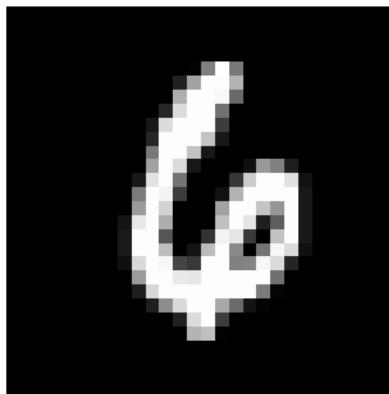
- the two-layer network we have learned on MNIST can easily classify digits with less than 3% error, but **learns more** than actually required
- remember that for both MNIST and CIFAR10, we flattened images (1-channel or 3-channel) into vectors, and the **order** of the elements (pixels) plays no role in learning
- so what if we **permute** the elements in all images, both training and test set?

input image representation

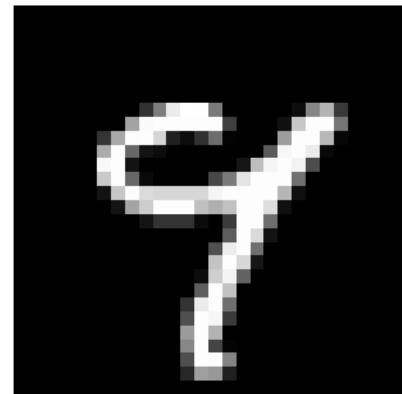
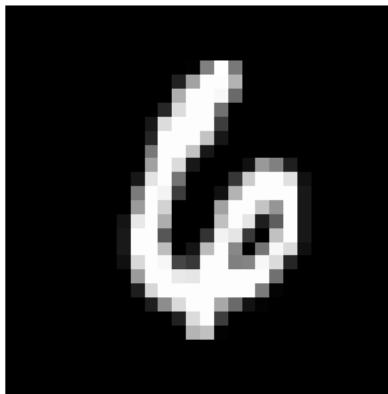
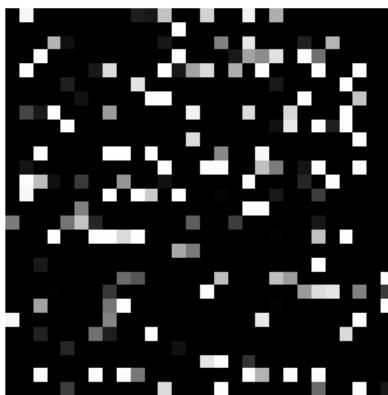
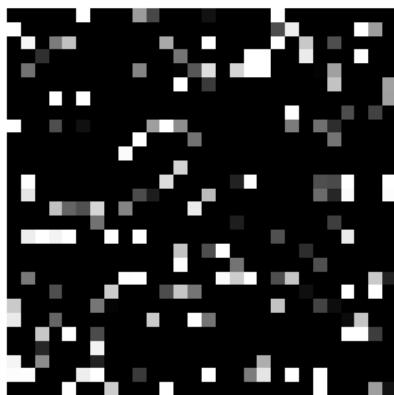


- the two-layer network we have learned on MNIST can easily classify digits with less than 3% error, but **learns more** than actually required
- remember that for both MNIST and CIFAR10, we flattened images (1-channel or 3-channel) into vectors, and the **order** of the elements (pixels) plays no role in learning
- so what if we **permute** the elements in all images, both training and test set?

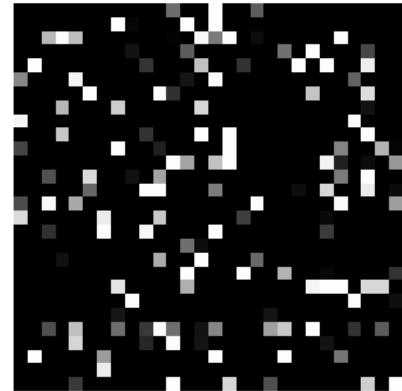
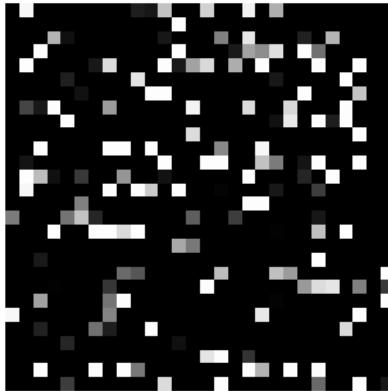
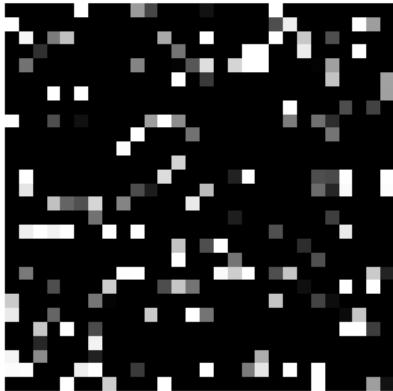
shuffling the dimensions



shuffling the dimensions

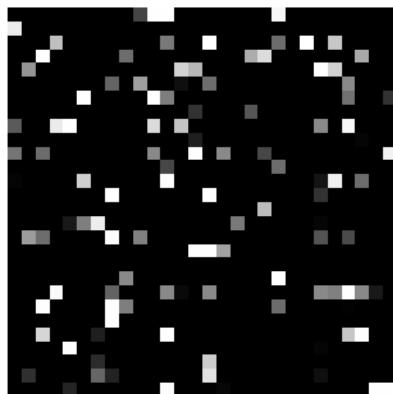
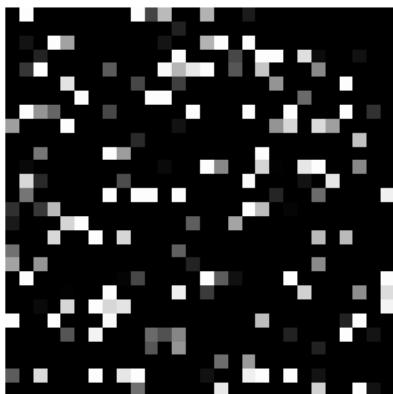
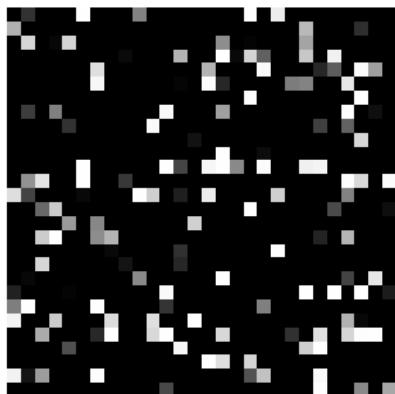
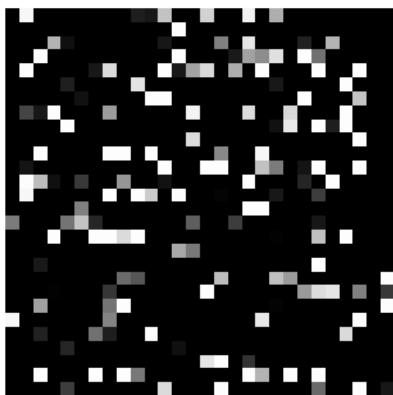
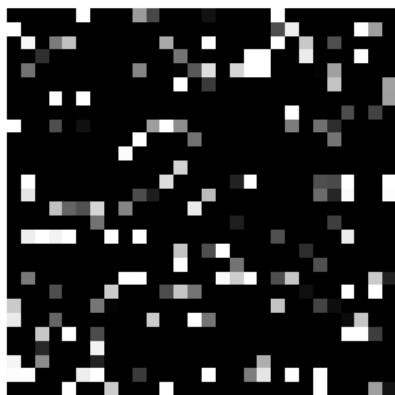


shuffling the dimensions

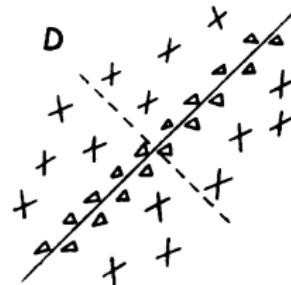
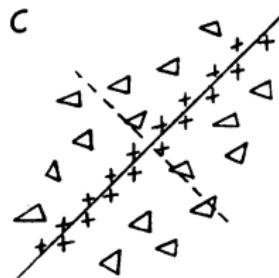


- this is what the computer sees
- it must make more sense when you start looking at more than one samples per class

shuffling the dimensions

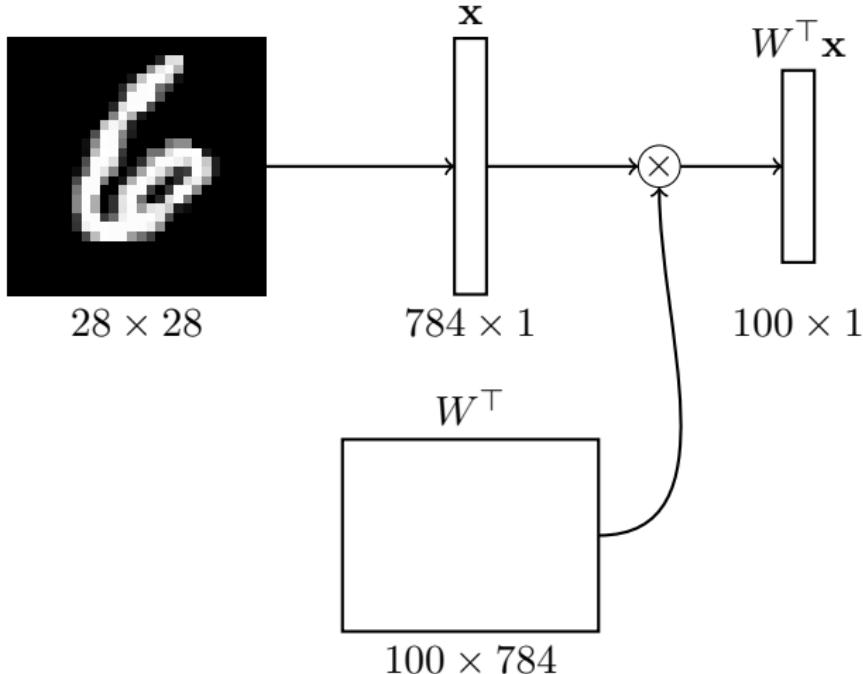


remember receptive fields?



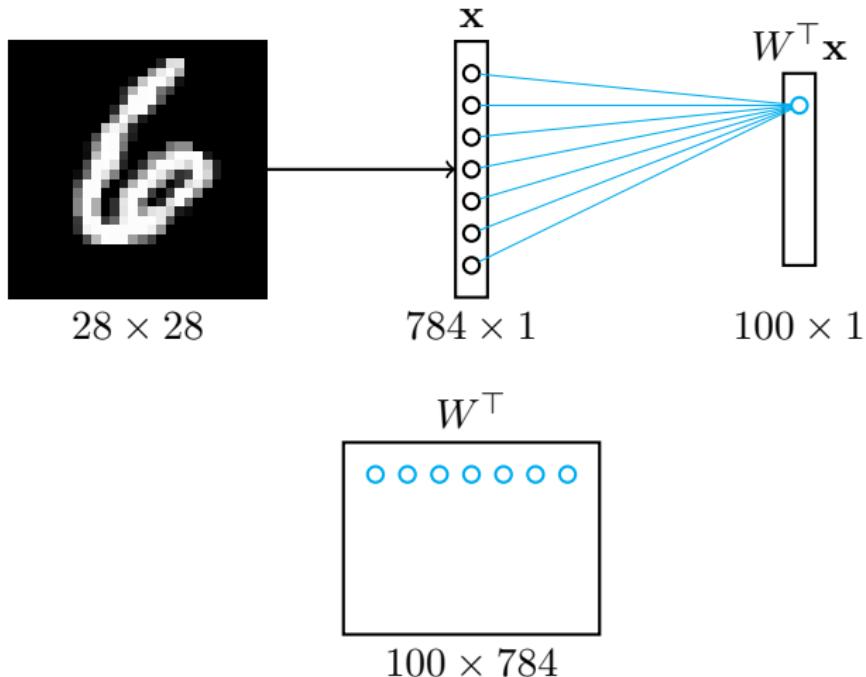
- A: 'on'-center LGN; B: 'off'-center LGN; C, D: simple cortical
- each cell only has a localized response over a **receptive field**
- **x**: excitatory ('on'), \triangle : inhibitory ('off') responses
- **topographic mapping**: there is one cell with the same response pattern centered at each position

matrix multiplication



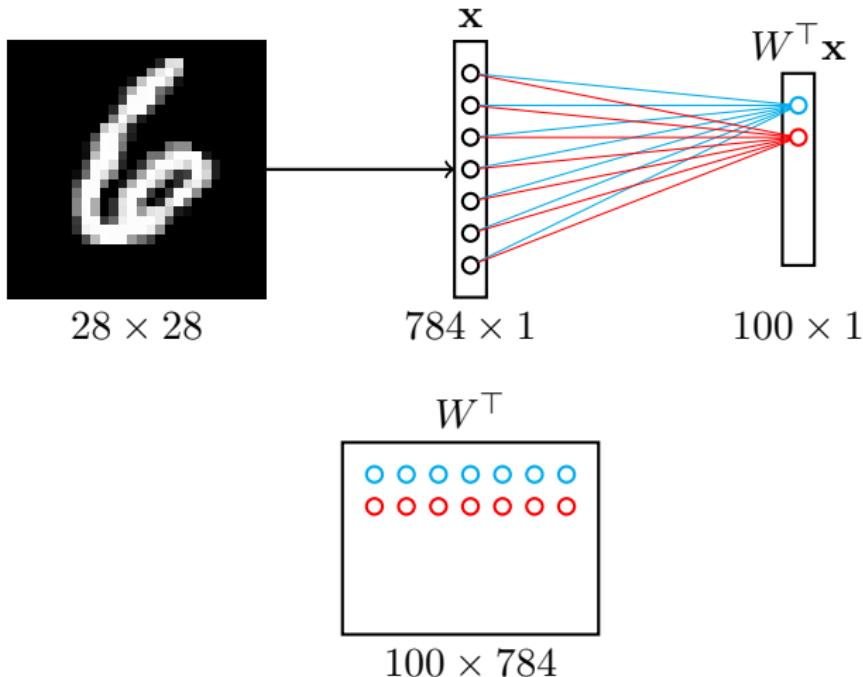
- inputs \mathbf{x} are mapped to activations $W^\top \mathbf{x}$
- columns/rows of W^\top correspond to input/activation elements

matrix multiplication → fully connected



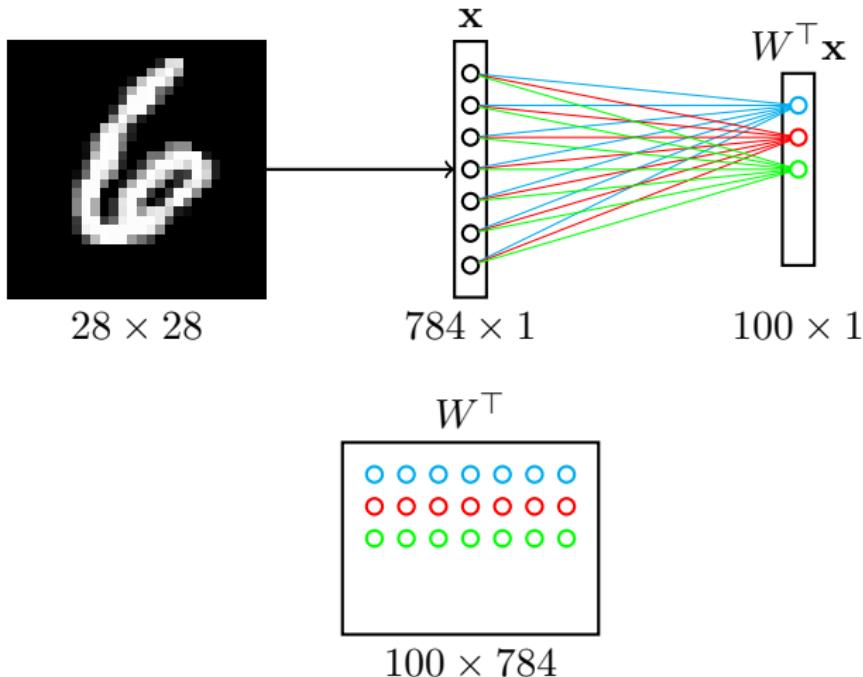
- each row of W^T yields one activation element (**cell**)
- each cell is **fully connected** to all input elements

matrix multiplication → fully connected



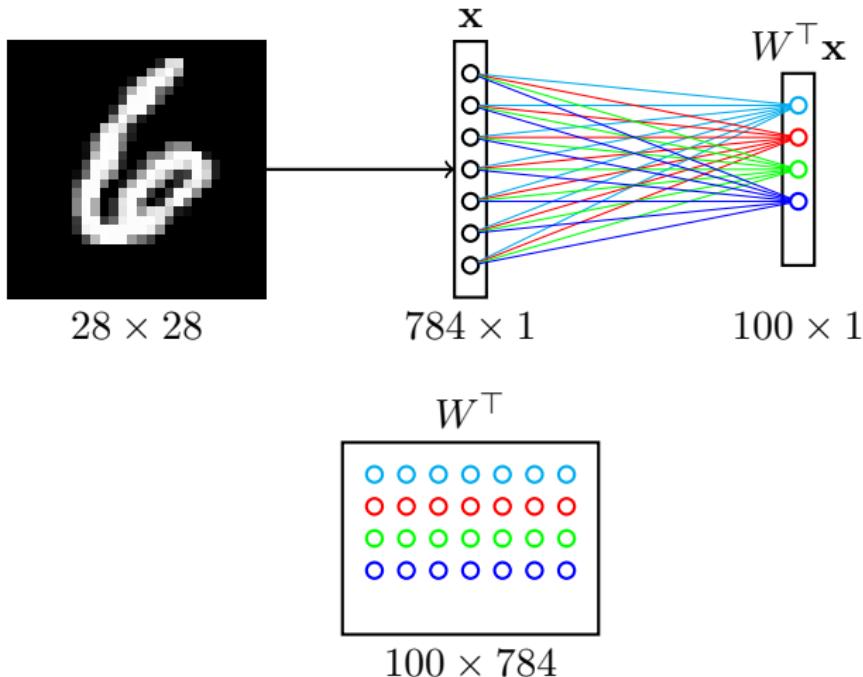
- each row of W^T yields one activation element (**cell**)
- each cell is **fully connected** to all input elements

matrix multiplication → fully connected



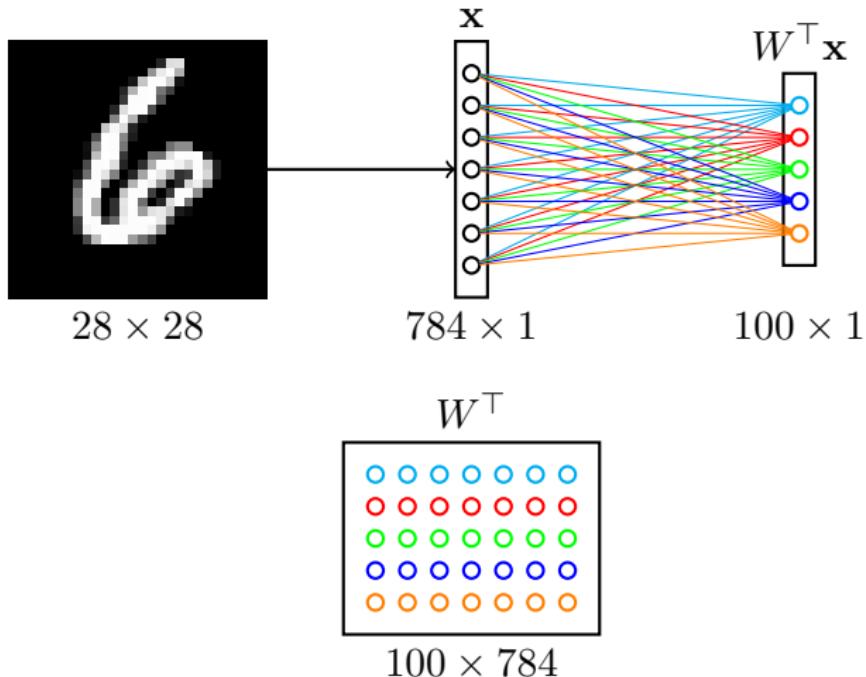
- each row of W^T yields one activation element (**cell**)
- each cell is **fully connected** to all input elements

matrix multiplication → fully connected



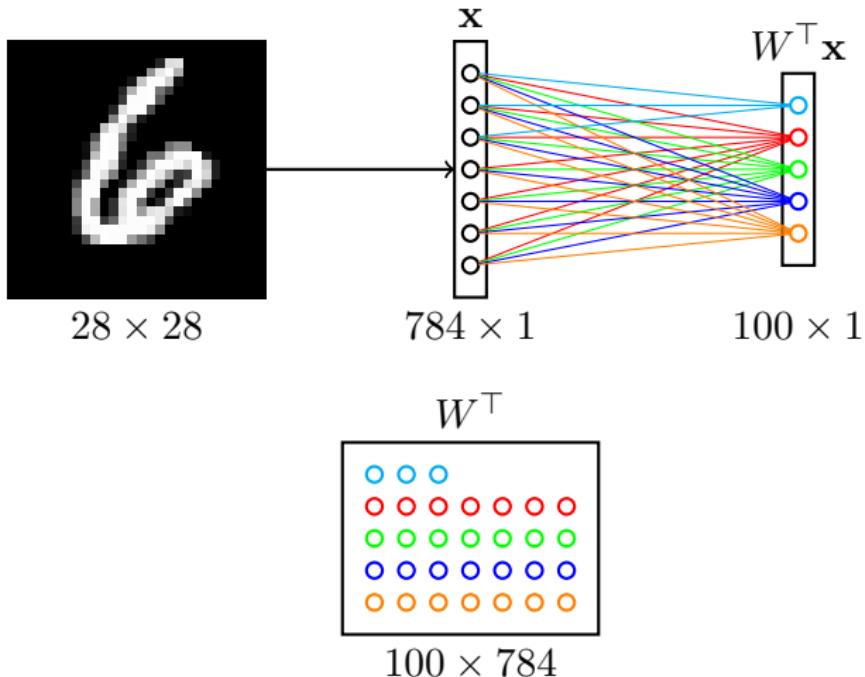
- each row of W^T yields one activation element (**cell**)
- each cell is **fully connected** to all input elements

matrix multiplication → fully connected



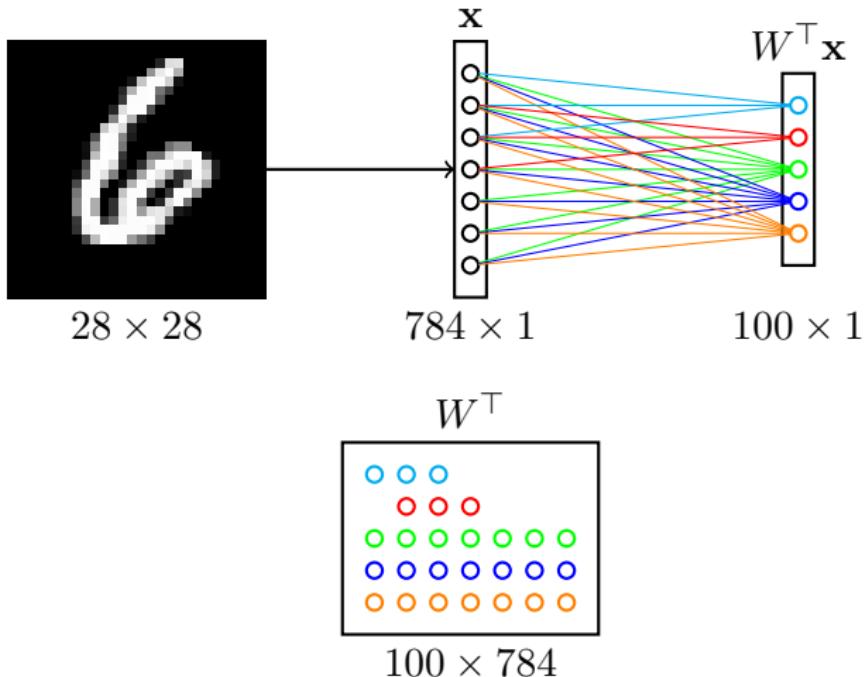
- each row of W^T yields one activation element (**cell**)
- each cell is **fully connected** to all input elements

sparse connections



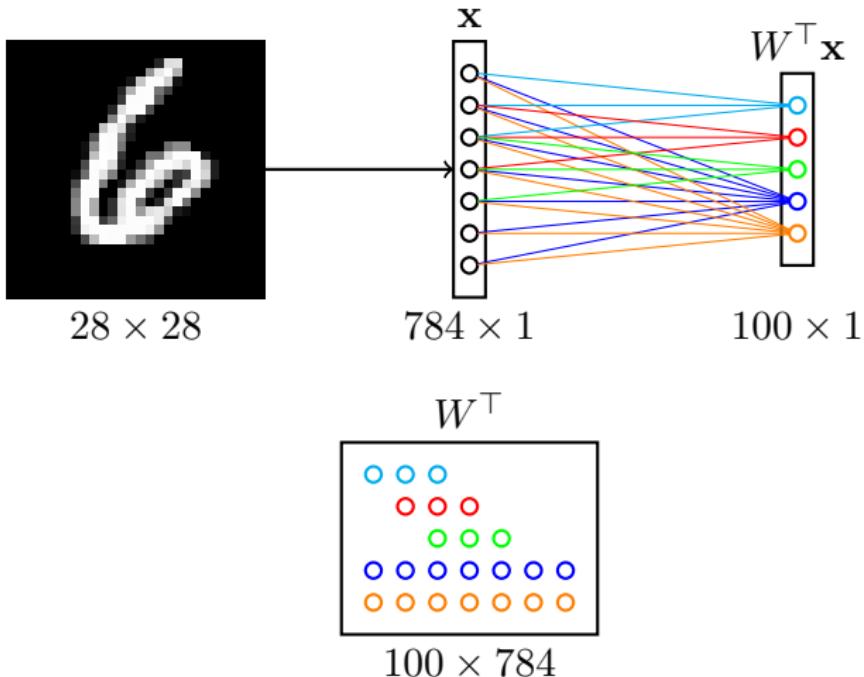
- now, we only keep a **sparse** set of connections
- and matrix W becomes sparse as well

sparse connections



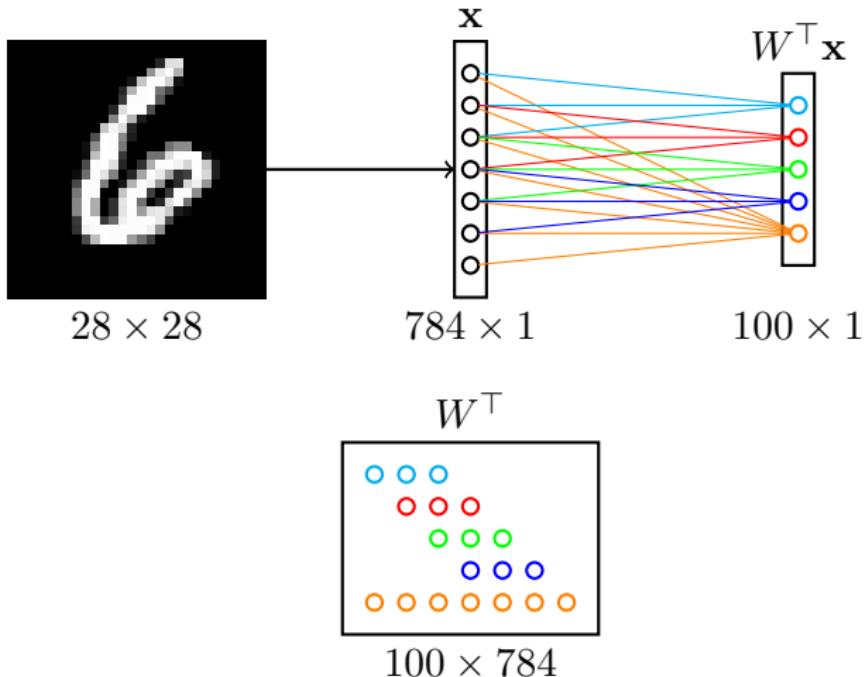
- now, we only keep a **sparse** set of connections
- and matrix W becomes sparse as well

sparse connections



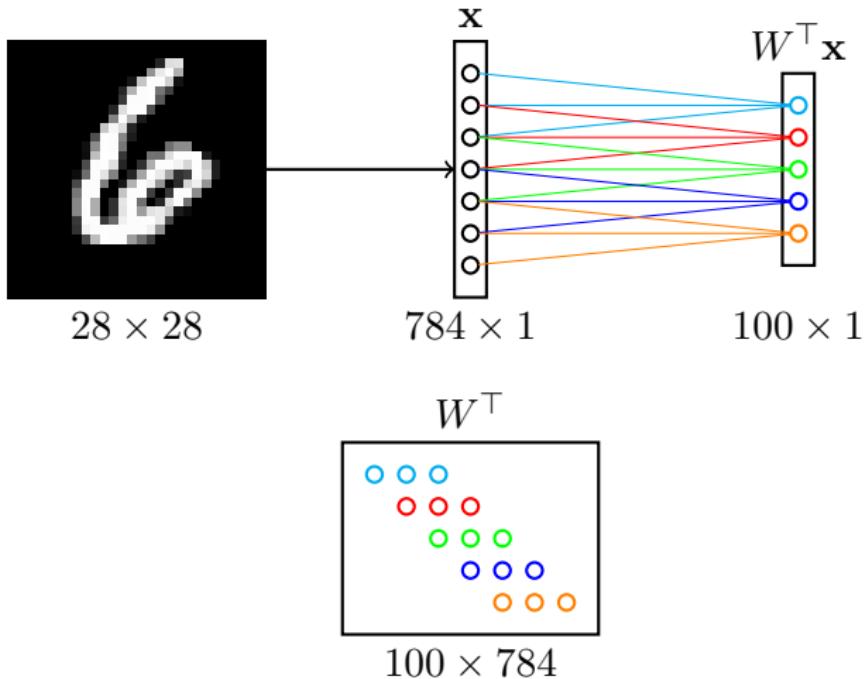
- now, we only keep a **sparse** set of connections
- and matrix W becomes sparse as well

sparse connections



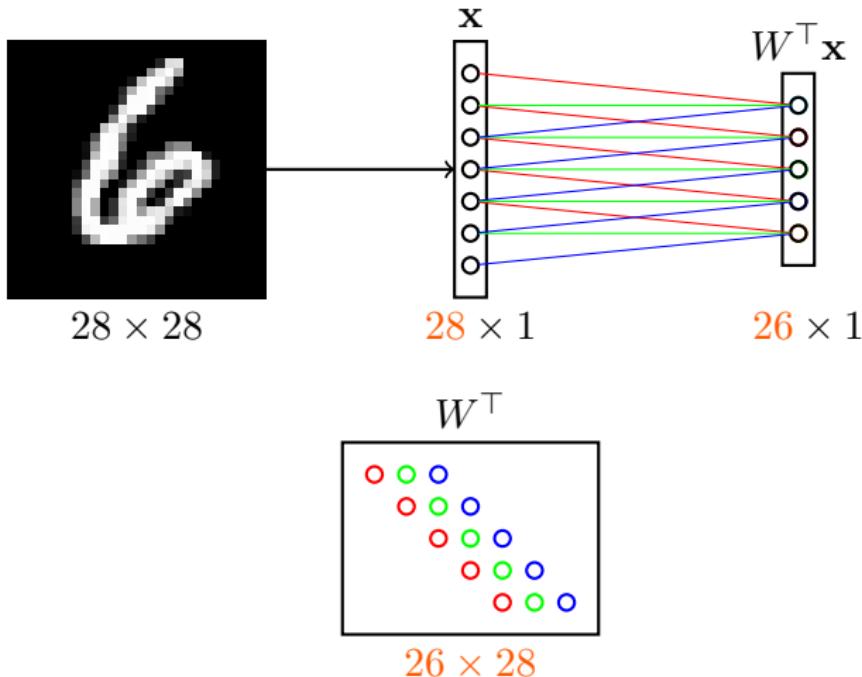
- now, we only keep a **sparse** set of connections
- and matrix W becomes sparse as well

sparse connections



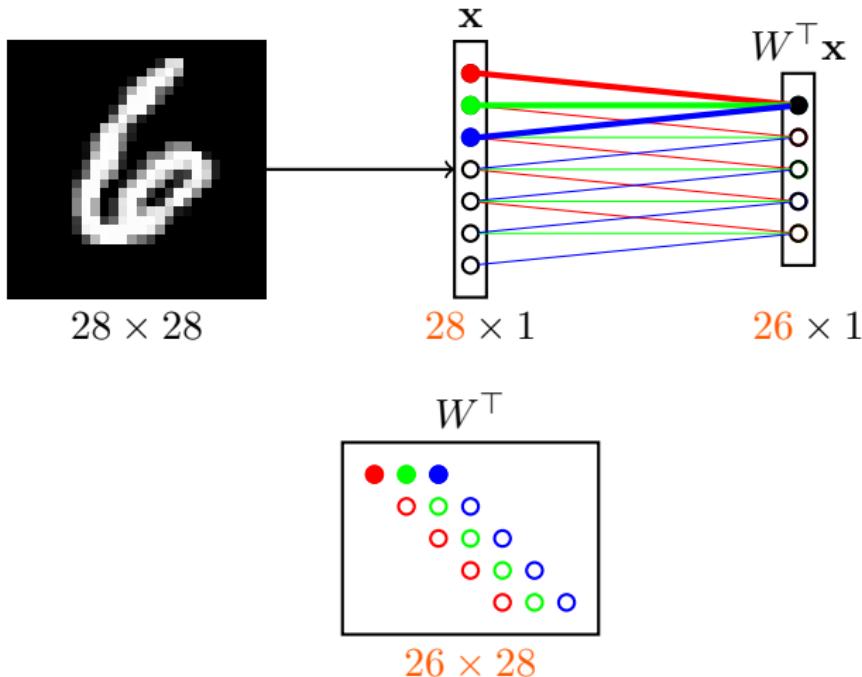
- now, we only keep a **sparse** set of connections
- and matrix W becomes sparse as well

Toeplitz matrix



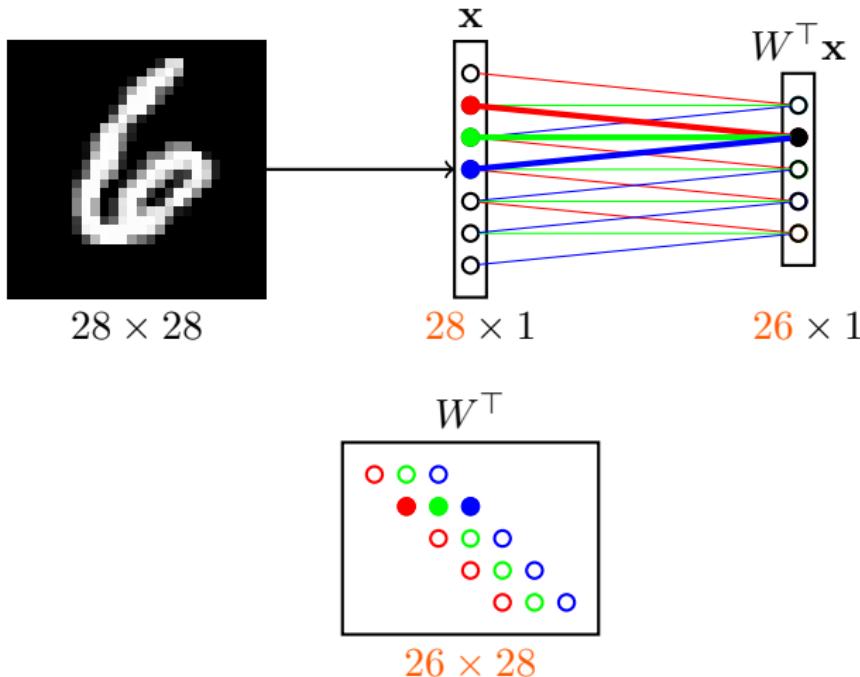
- now, we only refer to one input column; we will repeat
- and all weights having the same color are made equal (**shared**)

Toeplitz matrix \rightarrow convolution



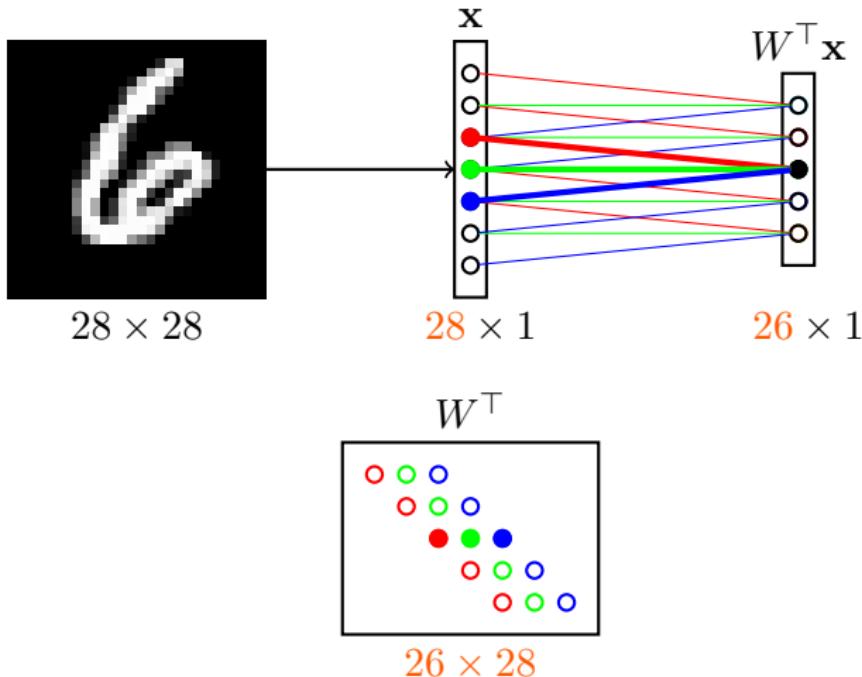
- this can be seen as **shifting** the same weight triplet (**kernel**)
- the set of inputs seen by each cell is its **receptive field**

Toeplitz matrix \rightarrow convolution



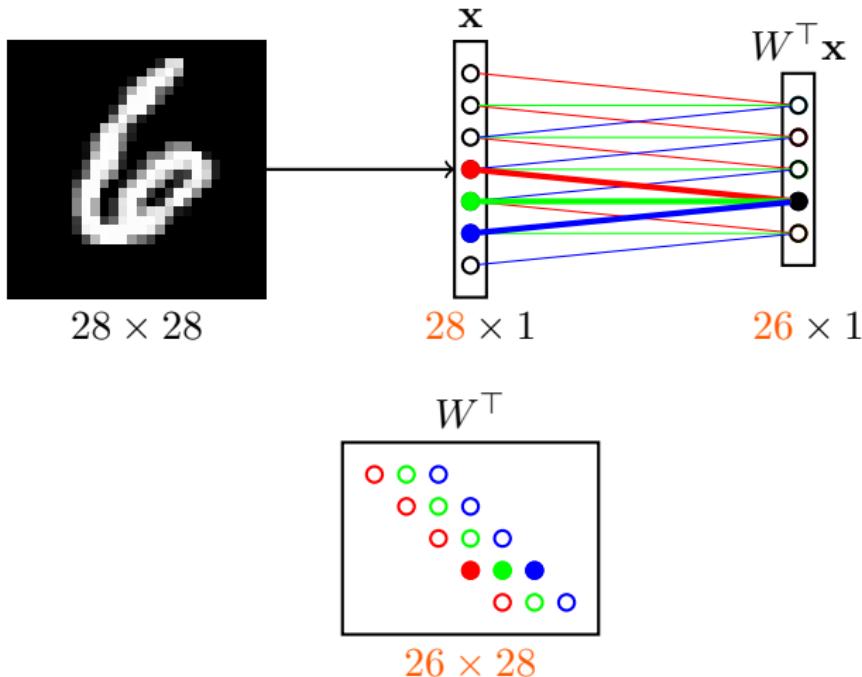
- this can be seen as **shifting** the same weight triplet (**kernel**)
- the set of inputs seen by each cell is its **receptive field**

Toeplitz matrix \rightarrow convolution



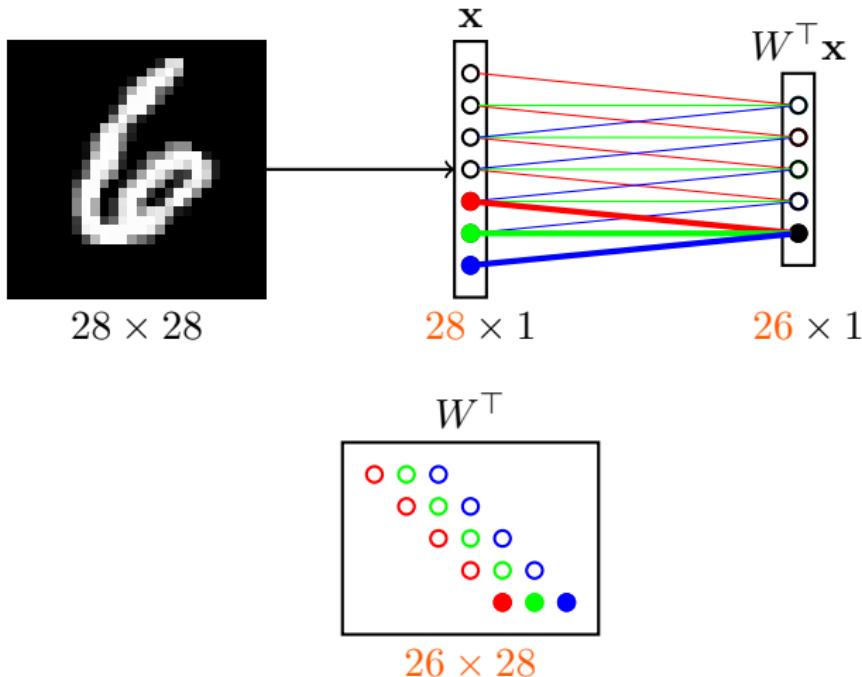
- this can be seen as **shifting** the same weight triplet (**kernel**)
- the set of inputs seen by each cell is its **receptive field**

Toeplitz matrix \rightarrow convolution



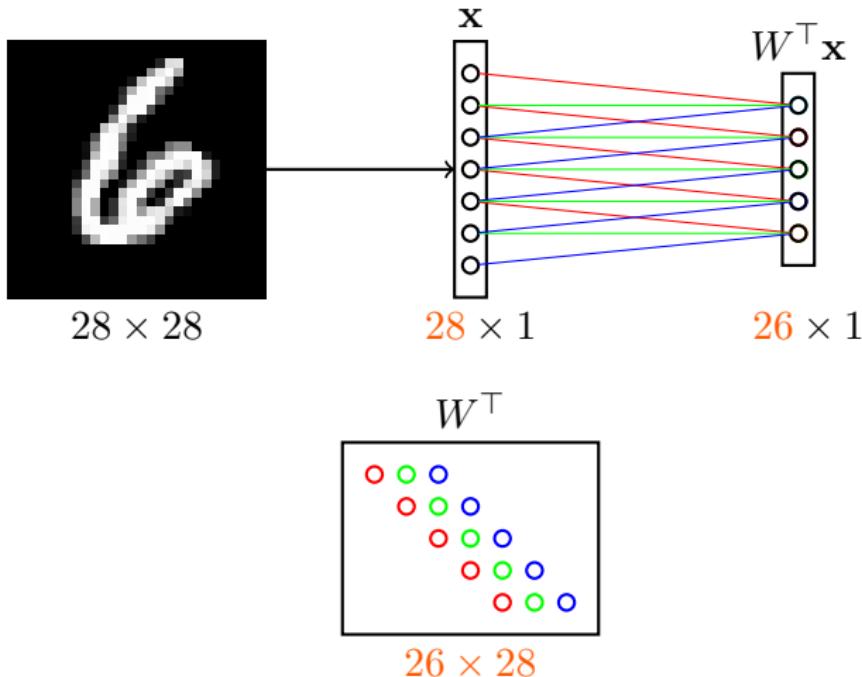
- this can be seen as **shifting** the same weight triplet (**kernel**)
- the set of inputs seen by each cell is its **receptive field**

Toeplitz matrix \rightarrow convolution



- this can be seen as **shifting** the same weight triplet (**kernel**)
- the set of inputs seen by each cell is its **receptive field**

Toeplitz matrix \rightarrow convolution



- this is an 1d **convolution** and generalizes to 2d
- this new mapping is a **convolutional layer**

convolutional networks

convolutional layer

- 1 still linear, still matrix multiplication, just constrained
- 2 local receptive fields → sparse connections between units
- 3 translation equivariant → shared weights
- 4 sparse + shared → regularized: less parameters to learn

convolutional network

- a network of convolutional layers, optionally followed by fully-connected layers
- performs better (less than 1% error on MNIST), but not on shuffled input

convolutional networks

convolutional layer

- 1 still linear, still matrix multiplication, just constrained
- 2 local receptive fields → sparse connections between units
- 3 translation equivariant → shared weights
- 4 sparse + shared → regularized: less parameters to learn

convolutional network

- a network of convolutional layers, optionally followed by fully-connected layers
- performs better (less than 1% error on MNIST), but not on shuffled input

convolutional networks

convolutional layer

- 1 still linear, still matrix multiplication, just constrained
- 2 local receptive fields → sparse connections between units
- 3 translation equivariant → shared weights
- 4 sparse + shared → regularized: less parameters to learn

convolutional network

- a network of convolutional layers, optionally followed by fully-connected layers
- performs better (less than 1% error on MNIST), but not on shuffled input

definition and properties

linear time-invariant (LTI) system

- discrete-time **signal**: $x[n]$, $n \in \mathbb{Z}$
- **system** (filter): $f(x)[n]$, $n \in \mathbb{Z}$
- **translation** (or shift, or delay): $s_k(x)[n] = x[n - k]$, $k \in \mathbb{Z}$
- **linear system**: commutes with linear combination

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- **time-invariant** system: commutes with translation

$$f(s_k(x)) = s_k(f(x))$$

linear time-invariant (LTI) system

- discrete-time **signal**: $x[n]$, $n \in \mathbb{Z}$
- **system** (filter): $f(x)[n]$, $n \in \mathbb{Z}$
- **translation** (or shift, or delay): $s_k(x)[n] = x[n - k]$, $k \in \mathbb{Z}$
- **linear system**: commutes with linear combination

$$f \left(\sum_i a_i x_i \right) = \sum_i a_i f(x_i)$$

- **time-invariant** system: commutes with translation

$$f(s_k(x)) = s_k(f(x))$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned}f(x)[n] &= f\left(\sum_k x[k]s_k(\delta)\right)[n] = \sum_k x[k]s_k(f(\delta))[n] \\&= \sum_k x[k]h[n - k] = x * h[n]\end{aligned}$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned}f(x)[n] &= f\left(\sum_k x[k]s_k(\delta)\right)[n] = \sum_k x[k]s_k(f(\delta))[n] \\&= \sum_k x[k]h[n - k] := (x * h)[n]\end{aligned}$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \boxed{\sum_k x[k]s_k(\delta)[n]}$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f \left(\sum_k x[k]s_k(\delta) \right) [n] = \sum_k x[k]s_k(f(\delta))[n] \\ &= \sum_k x[k]h[n - k] := (x * h)[n] \end{aligned}$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= \boxed{f} \left(\sum_k x[k]s_k(\delta) \right) [n] = \sum_k x[k]s_k(\boxed{f(\delta)})[n] \\ &= \sum_k x[k]h[n - k] := (x * h)[n] \end{aligned}$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$f(x)[n] = f \left(\sum_k x[k]s_k(\delta) \right) [n] = \sum_k x[k]s_k(f(\delta))[n]$$

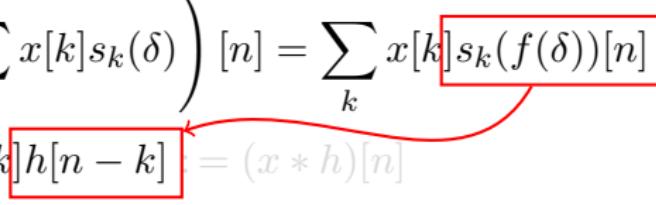
$$= \sum_k x[k]h[n - k] := (x * h)[n]$$

LTI system \equiv convolution

- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f\left(\sum_k x[k]s_k(\delta)\right)[n] = \sum_k x[k]s_k(f(\delta))[n] \\ &= \sum_k x[k]h[n - k] = (x * h)[n] \end{aligned}$$


LTI system \equiv convolution

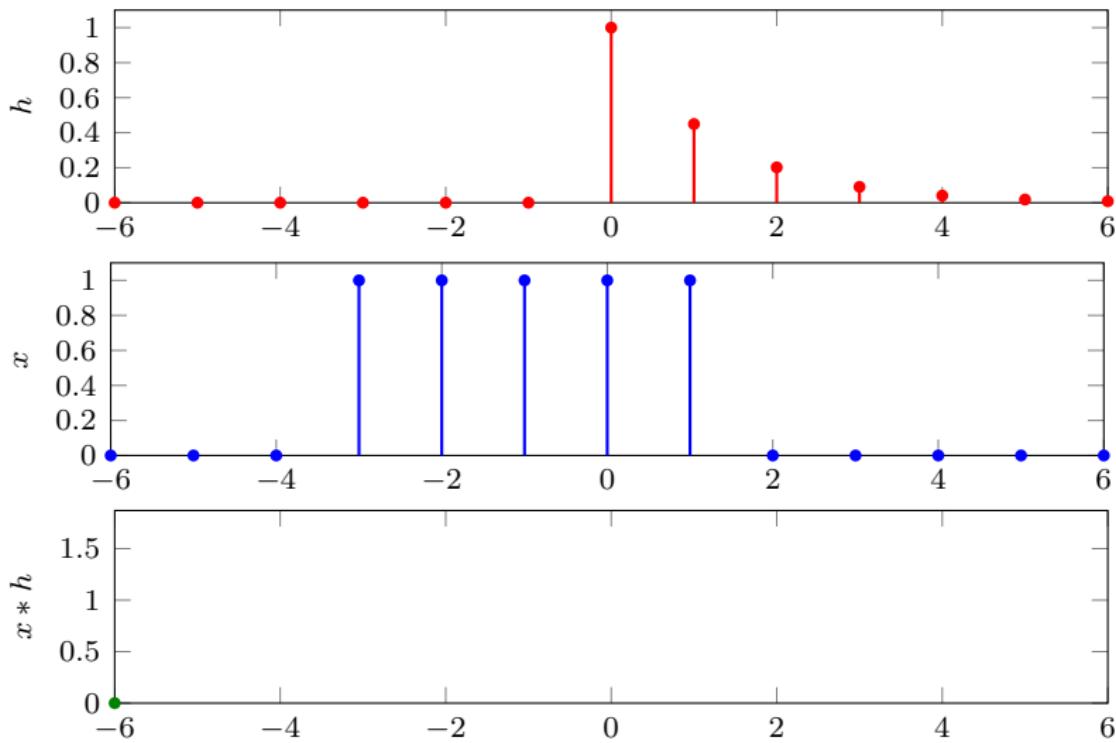
- unit impulse $\delta[n] = \mathbb{1}[n = 0]$
- every signal x expressed as

$$x[n] = \sum_k x[k]\delta[n - k] = \sum_k x[k]s_k(\delta)[n]$$

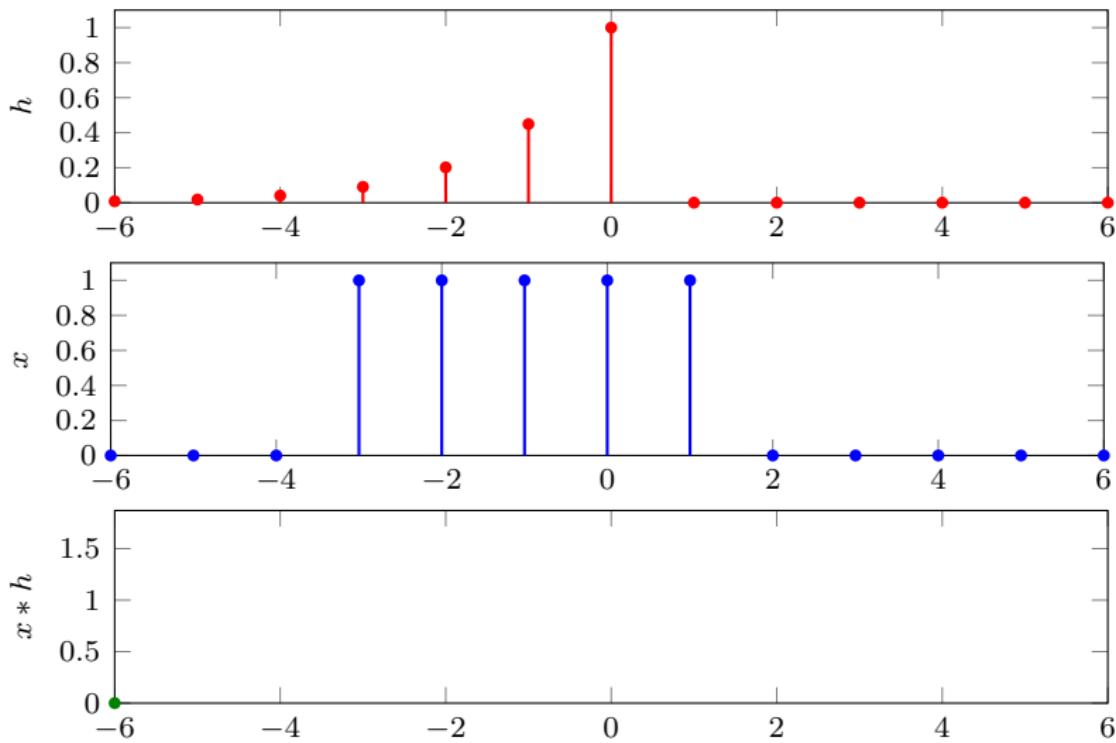
- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f\left(\sum_k x[k]s_k(\delta)\right)[n] = \sum_k x[k]s_k(f(\delta))[n] \\ &= \boxed{\sum_k x[k]h[n - k] := (x * h)[n]} \end{aligned}$$

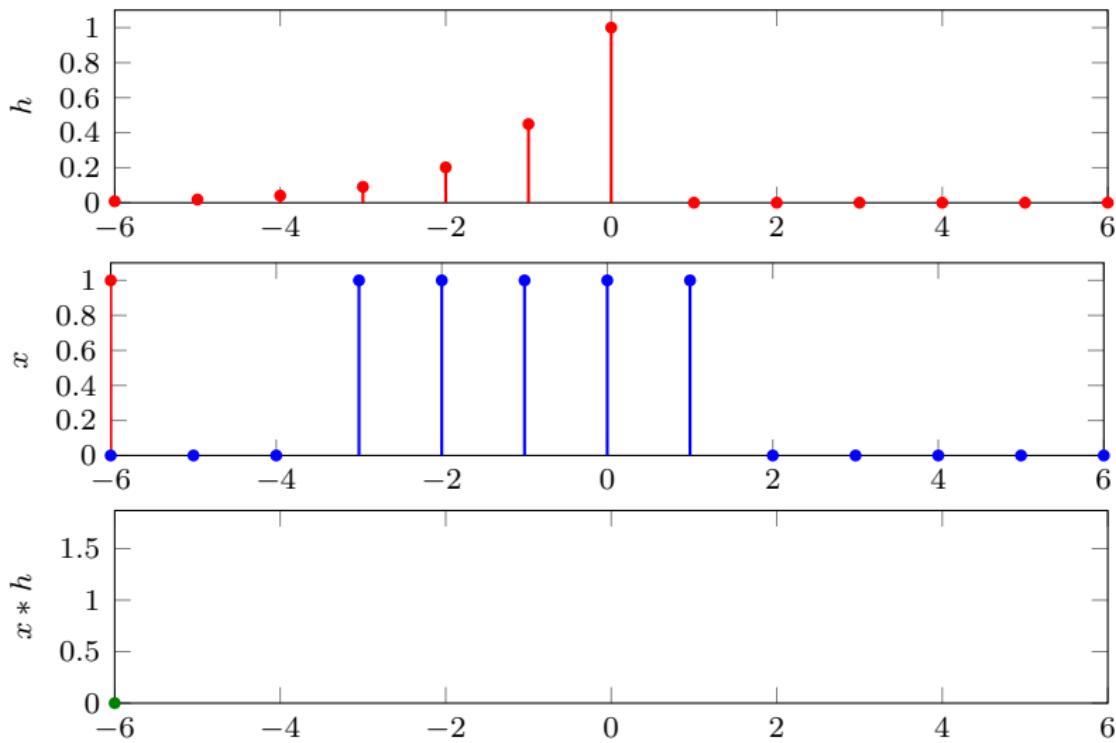
1d convolution



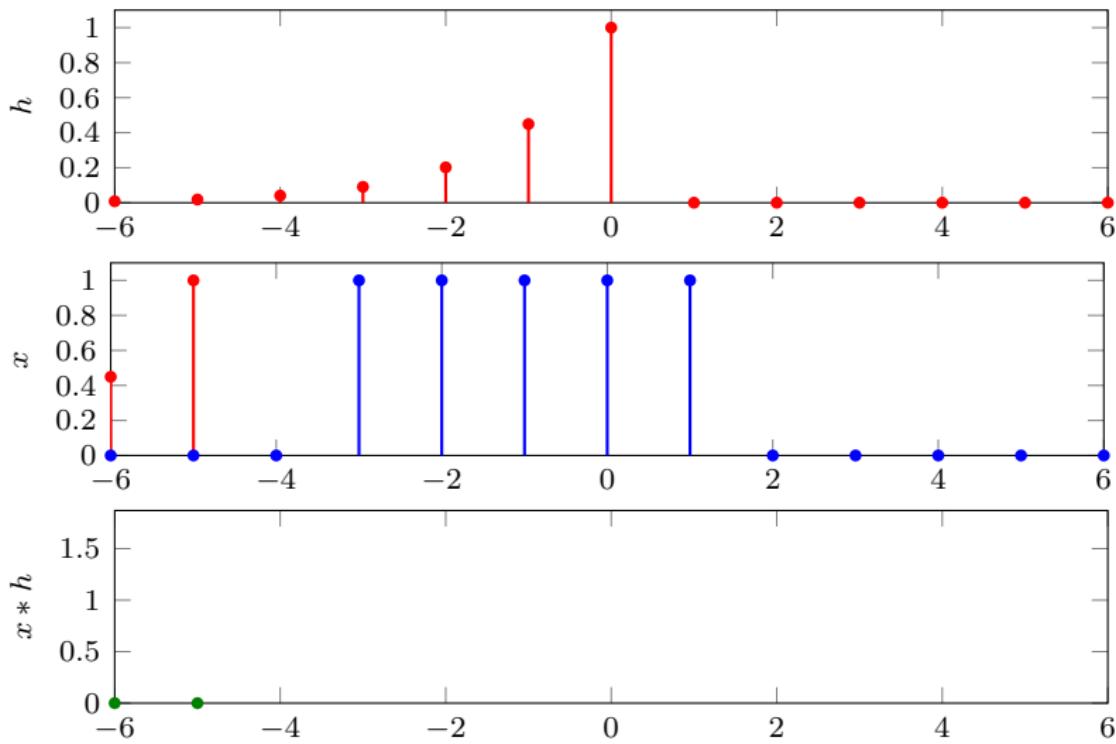
1d convolution



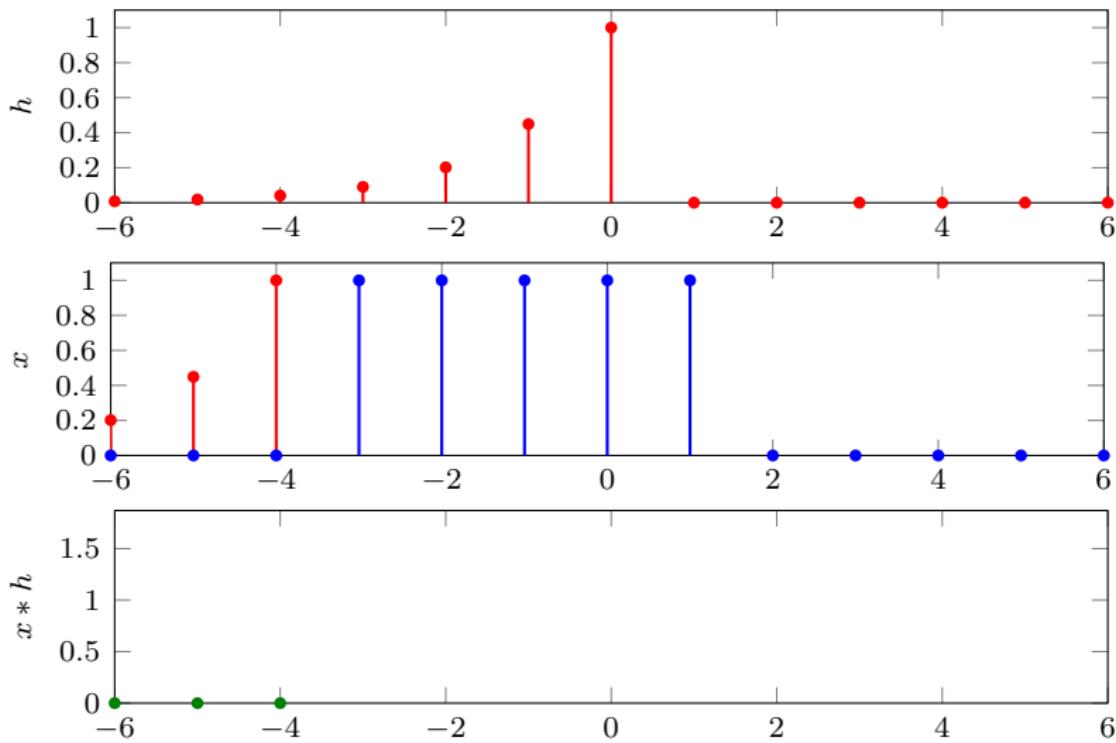
1d convolution



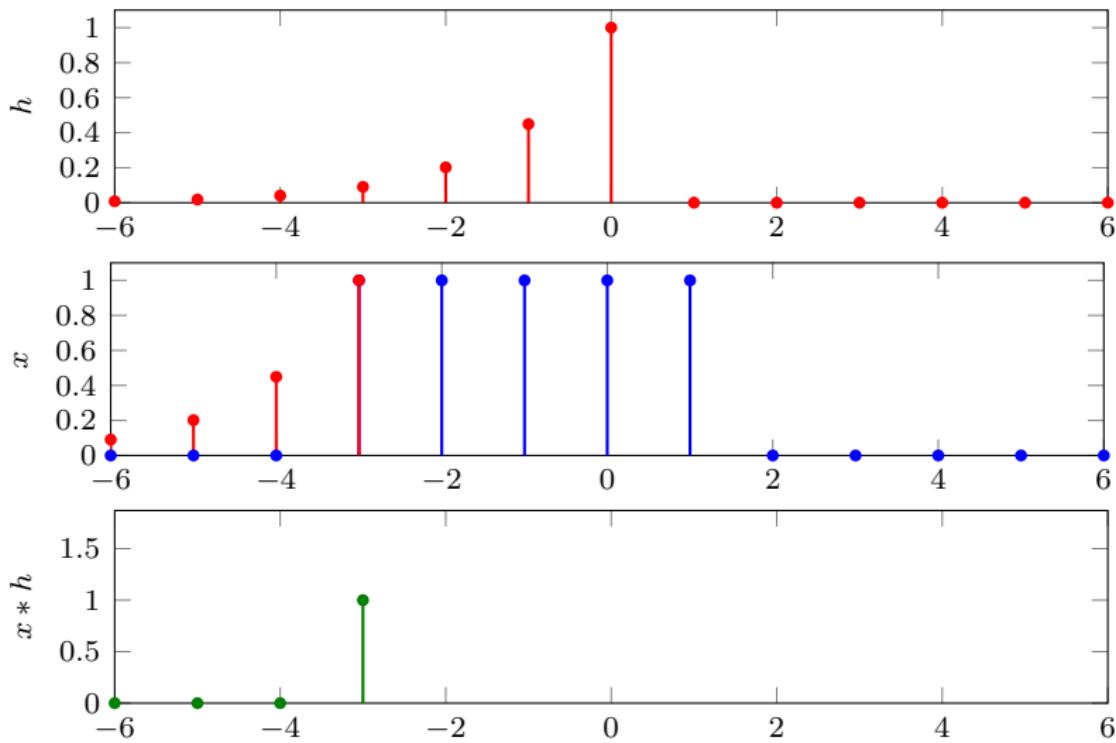
1d convolution



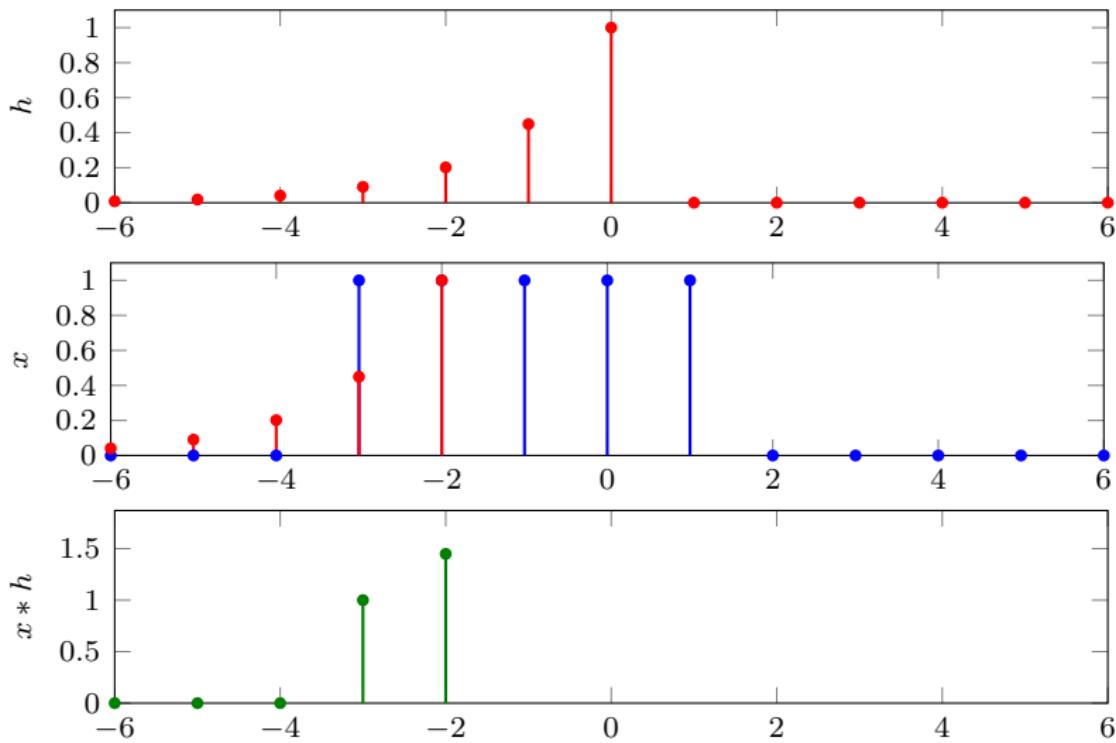
1d convolution



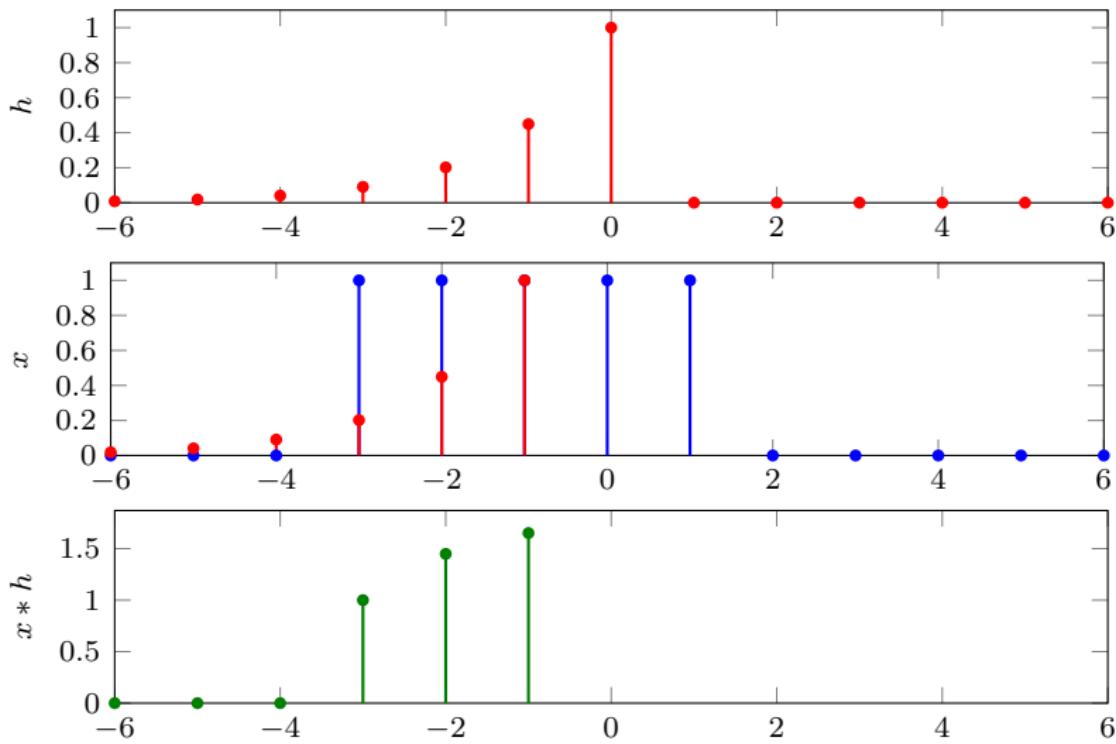
1d convolution



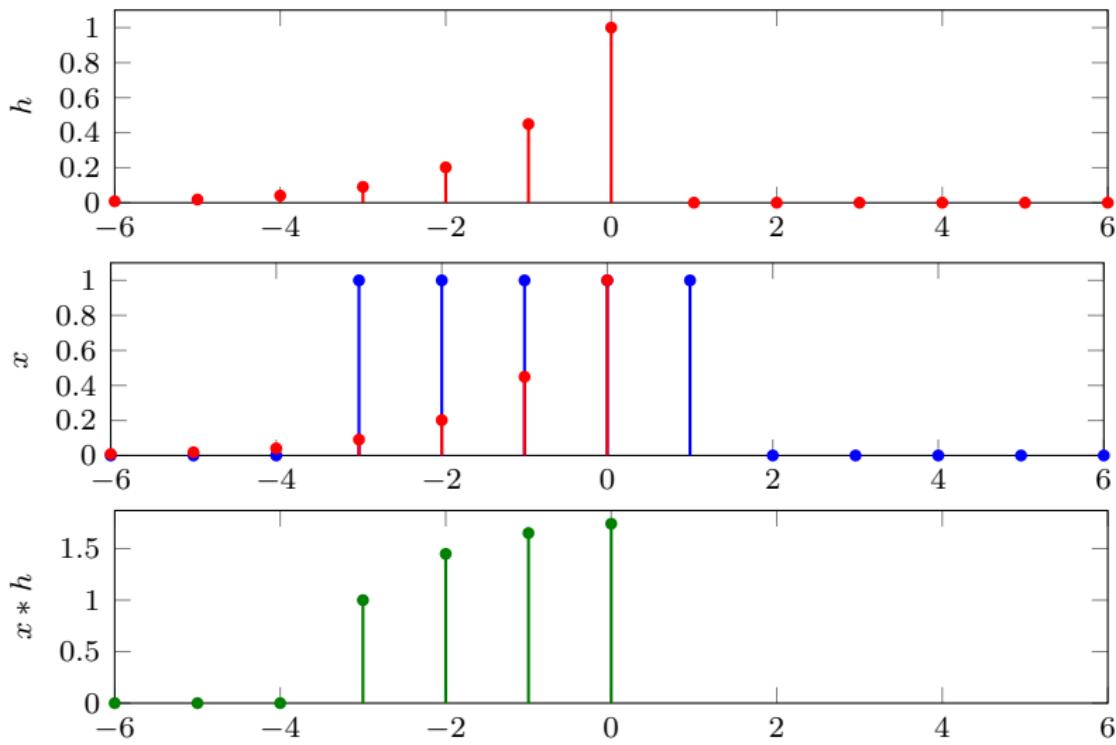
1d convolution



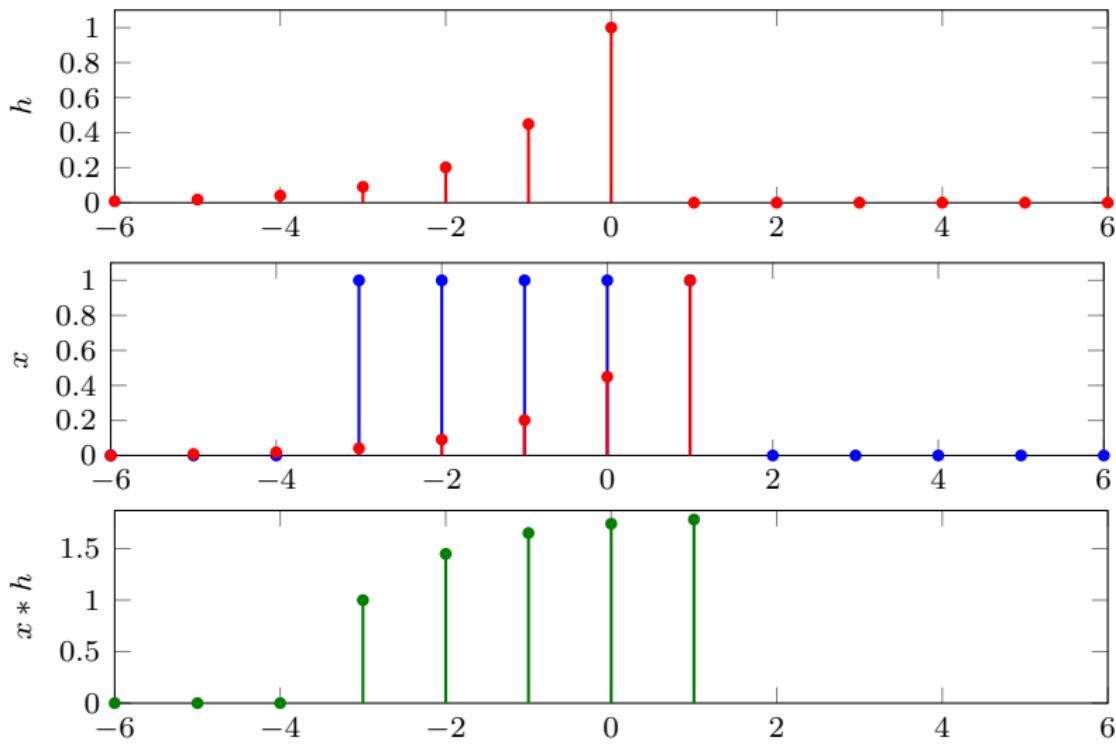
1d convolution



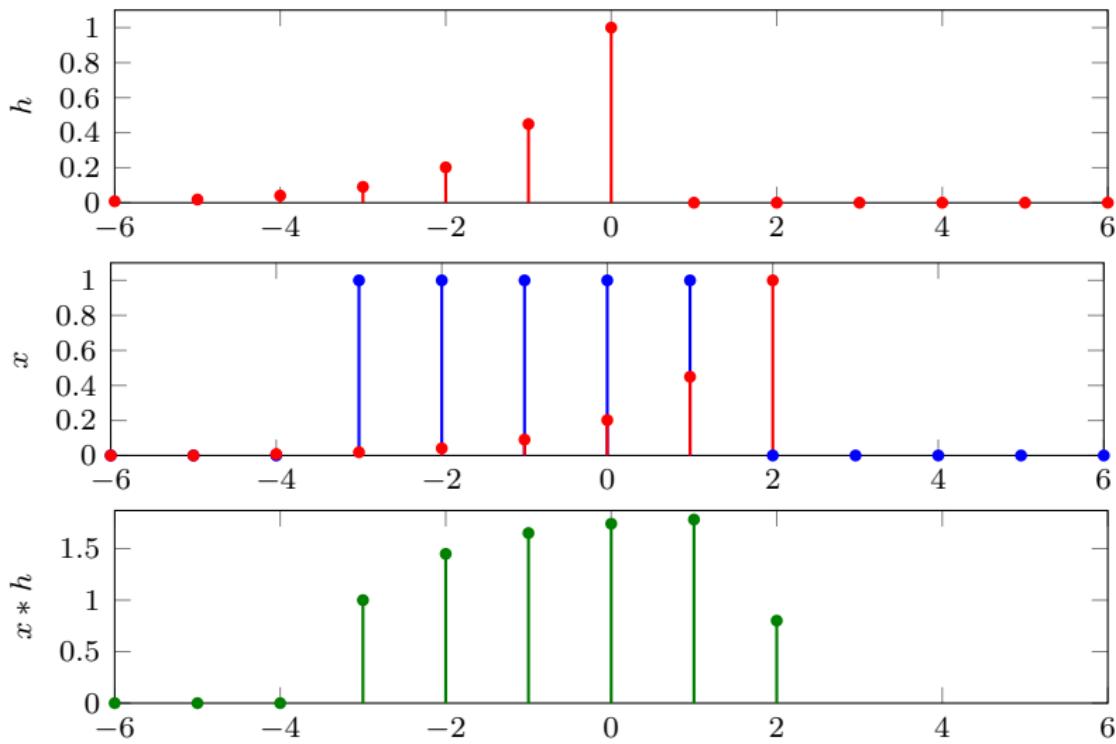
1d convolution



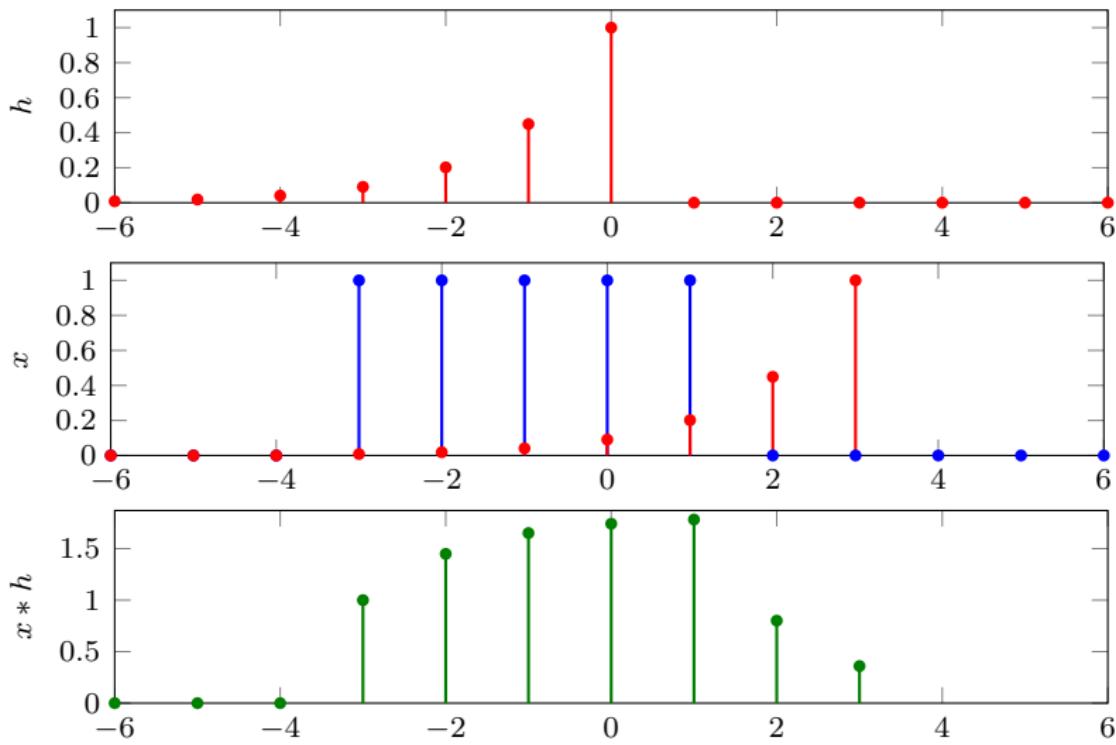
1d convolution



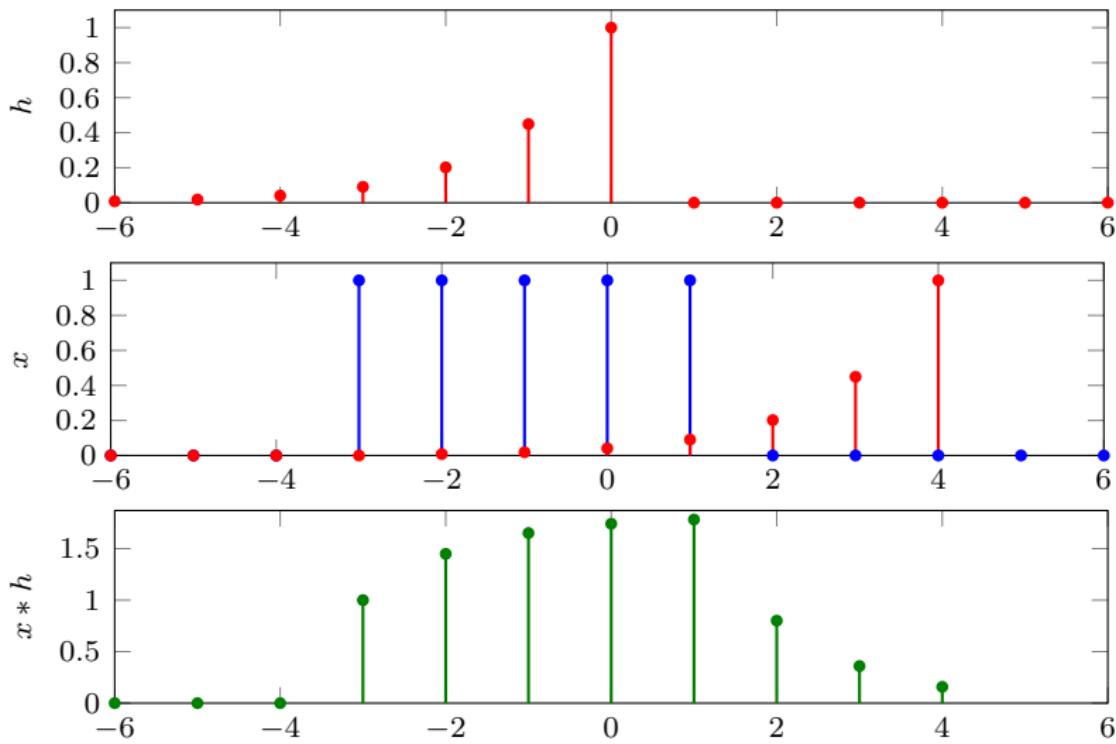
1d convolution



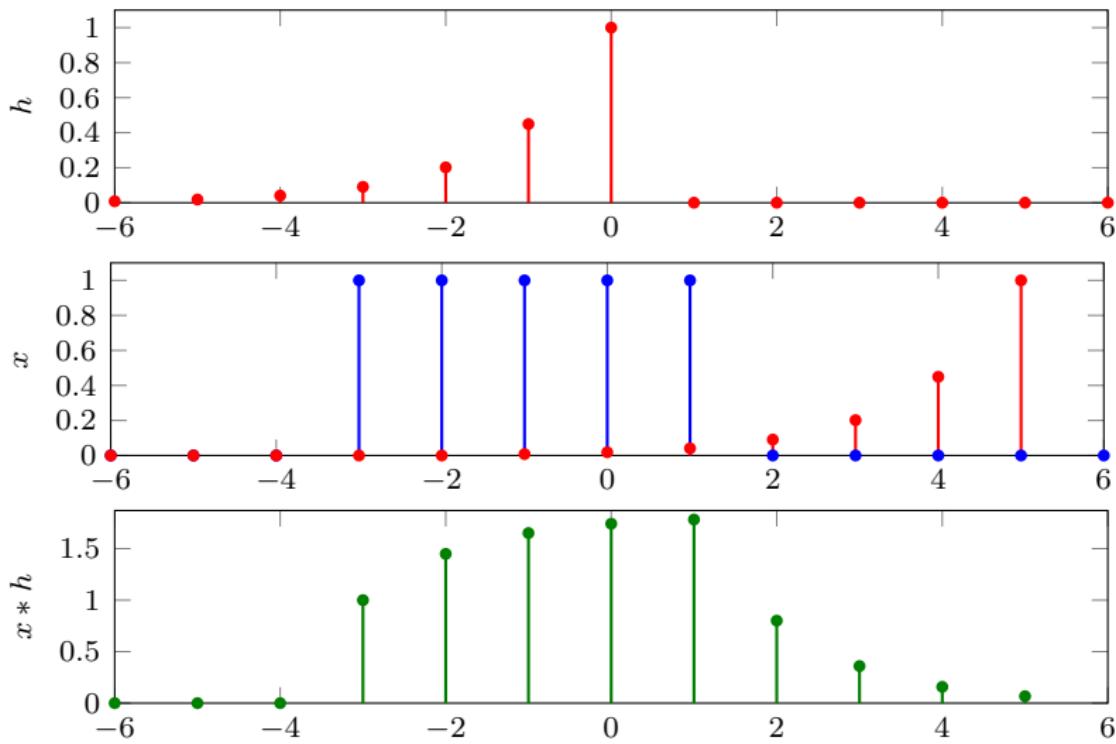
1d convolution



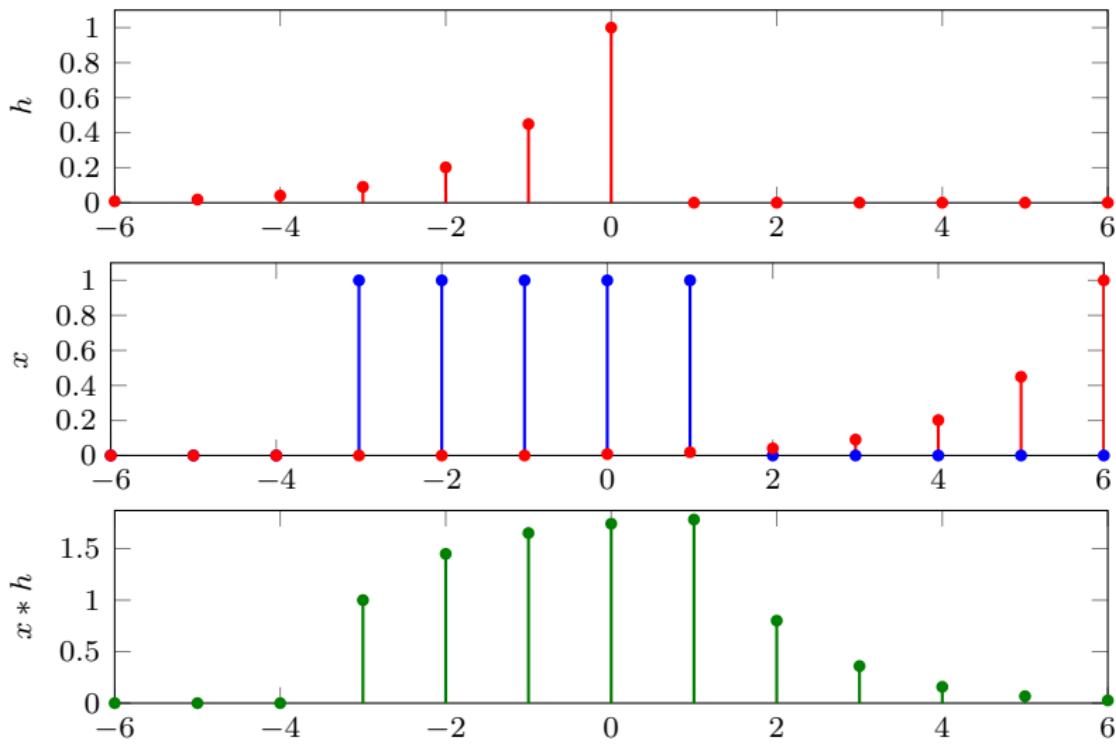
1d convolution



1d convolution



1d convolution



invariance vs. equivariance

- time invariance: invariance to **absolute** time (or position)
- translation (or shift) equivariance: equivariance to **relative** time (or position)
- despite confusion, both mean the same thing: **system commutes with translation**

$$f(s_k(x)) = s_k(f(x))$$

however

- translation (or shift) invariance, means that for all k ,

$$f(s_k(x)) = f(x)$$

- each convolutional layer is translation equivariant; but **pooling** makes a network translation invariant, e.g.

$$\sum_n s_k(x)[n] = \sum_n x[n - k] = \sum_n x[n]$$

invariance vs. equivariance

- time invariance: invariance to **absolute** time (or position)
- translation (or shift) equivariance: equivariance to **relative** time (or position)
- despite confusion, both mean the same thing: **system commutes with translation**

$$f(s_k(x)) = s_k(f(x))$$

however

- translation (or shift) invariance, means that for all k ,

$$f(s_k(x)) = f(x)$$

- each convolutional layer is translation equivariant; but **pooling** makes a network translation invariant, e.g.

$$\sum_n s_k(x)[n] = \sum_n x[n - k] = \sum_n x[n]$$

finite impulse response (FIR)

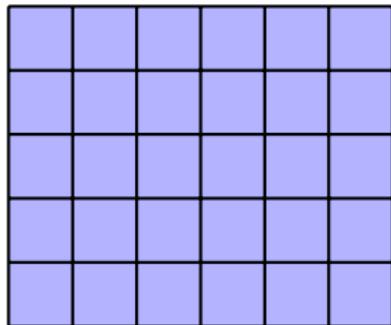
- an FIR system has impulse response h of finite duration (or spatial extent), because it settles to zero in finite time (extent) from the input impulse
- “sparse connections and local receptive fields” mean exactly that h is of finite duration (extent)
- we assume this in the following, starting with a 2d extension, where we write $x[\mathbf{n}]$, $\mathbf{n} \in \mathbb{Z}^2$

2d convolution

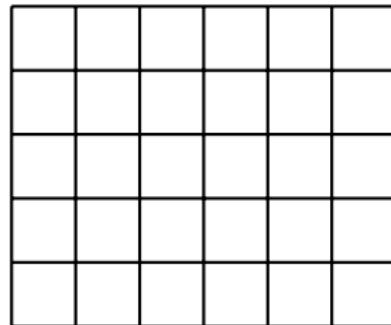
1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



x



$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

9	8	7				
6	5	4				
3	2	1				

x

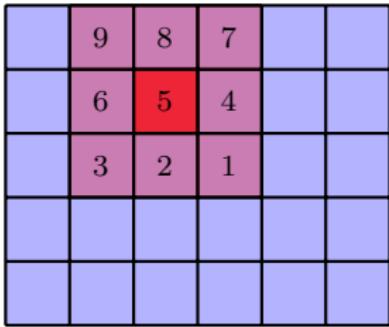
$x * h$

2d convolution

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

h

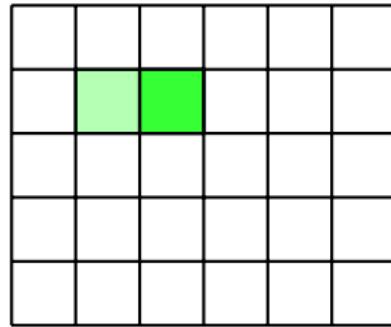
$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



A 5x5 grid of colored squares representing the input image x . The colors are blue, red, and white. The values are arranged as follows:

6	5	4		
3	2	1		

x



A 5x5 grid of colored squares representing the convolved image $x * h$. The colors are white, green, and white. The value 9 is highlighted in green, indicating it is the result of the convolution step where the kernel h was applied to the input x .

$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

		9	8	7	
		6	5	4	
		3	2	1	

x

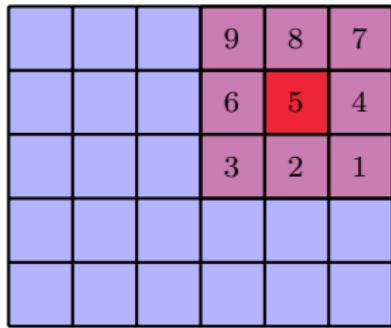
$x * h$

2d convolution

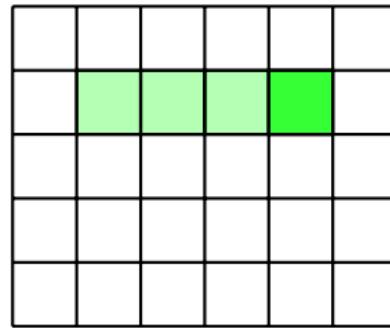
$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



x



$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

9	8	7				
6	5	4				
3	2	1				

x

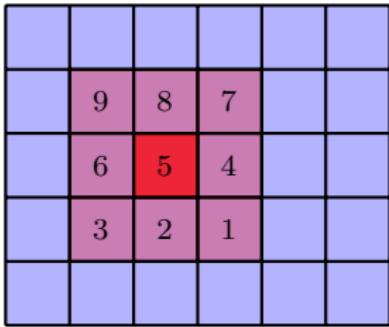
$x * h$

2d convolution

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

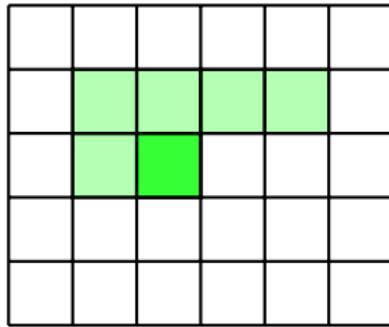
h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



A 5x5 grid of squares. The bottom-left 3x3 subgrid contains values 9, 8, 7; 6, 5, 4; and 3, 2, 1 respectively, all in red/pink. The rest of the grid is filled with light blue squares.

x



A 5x5 grid of squares. The bottom-left 3x3 subgrid contains values 9, 8, 7; 6, 5, 4; and 3, 2, 1 respectively, all in green. The rest of the grid is filled with white squares.

$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$

$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

		9	8	7		
		6	5	4		
		3	2	1		

x

$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

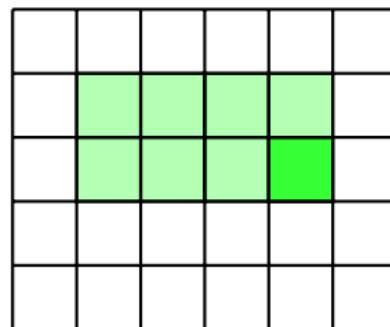
h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$

$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



x



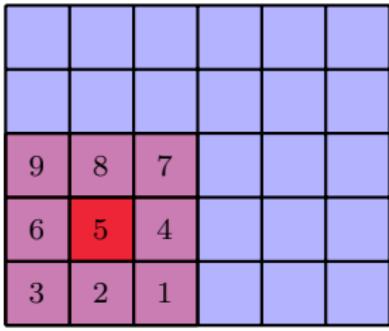
$x * h$

2d convolution

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

h

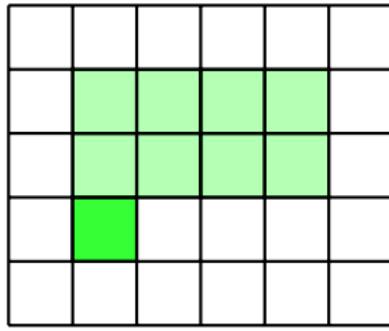
$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



A 3x3 input matrix x with values:

9	8	7
6	5	4
3	2	1

x



A 5x5 output matrix $x * h$ resulting from applying the kernel h to the input x . The result is zero everywhere except for one green cell at position (2, 2) which contains the value 5.

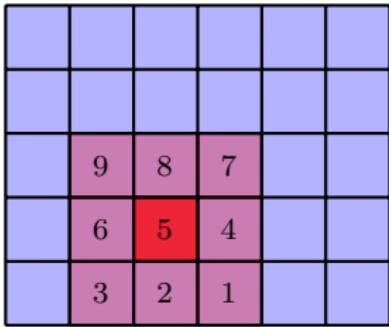
$x * h$

2d convolution

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

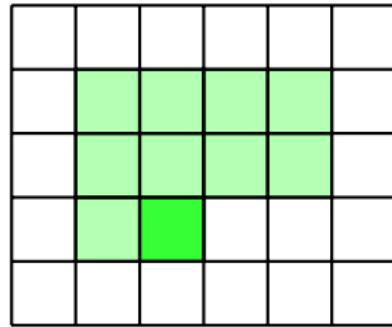
h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$



The input image x is a 5x5 grid of blue squares. It contains a 3x3 subgrid of red squares in the center, labeled with values 9, 8, 7, 6, 5, 4, 3, 2, and 1 respectively.

x



The convolved image $x * h$ is a 5x5 grid. It has a 3x3 green subgrid in the center, with the bottom-right square colored bright green. All other squares are white.

$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$

$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

		9	8	7					
		6	5	4					
		3	2	1					

x

$x * h$

2d convolution

1	2	3
4	5	6
7	8	9

h

$$(x * h)[\mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}]$$

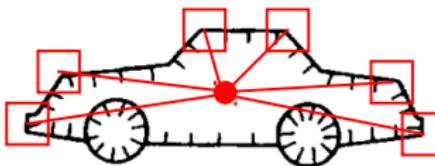
$$= \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}]$$

			9	8	7	
			6	5	4	
			3	2	1	

x

$x * h$

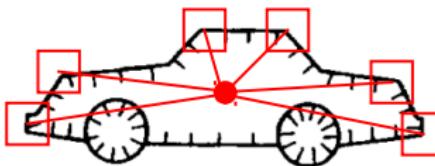
remember Hough in translation space?



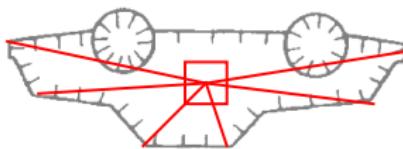
model image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

remember Hough in translation space?



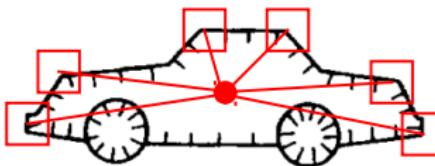
model image



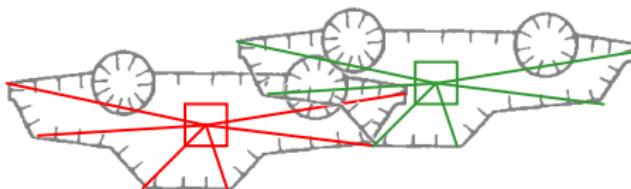
test image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

remember Hough in translation space?



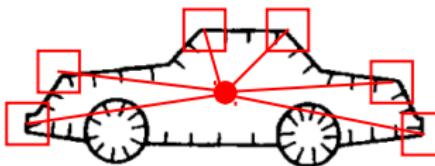
model image



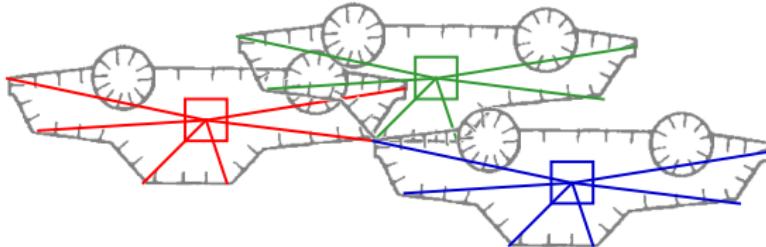
test image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

remember Hough in translation space?



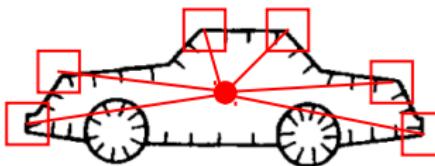
model image



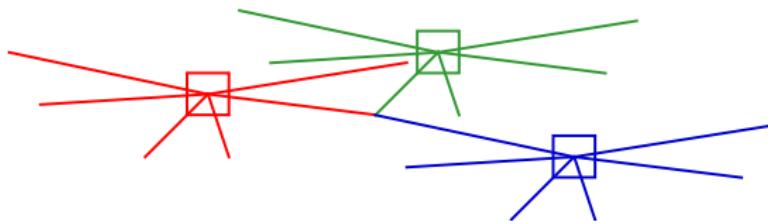
test image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

remember Hough in translation space?



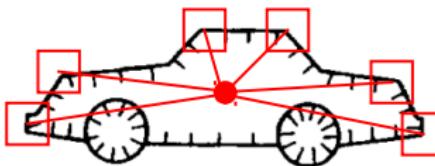
model image



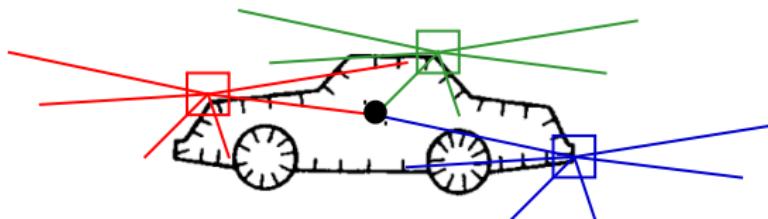
test image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

remember Hough in translation space?



model image



test image

- **model:** record coordinates relative to reference point
- **test:** votes for all possible coordinates of reference point
- impulse response is the **flipped** pattern to be detected

cross-correlation

- convolution is commutative

$$(x * h)[\mathbf{n}] := \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{n} - \mathbf{k}] = \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{n} - \mathbf{k}] = (h * x)[\mathbf{n}]$$

- cross-correlation is not

$$(h \star x)[\mathbf{n}] := \sum_{\mathbf{k}} h[\mathbf{k}]x[\mathbf{k} + \mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}]h[\mathbf{k} - \mathbf{n}] = (x \star h)[-\mathbf{n}]$$

- both are LTI; the only difference is that in cross-correlation, h refers to the **flipped** impulse response
- but if h is even ($h[n] = h[-n]$), then $h \star x = x * h = h * x$
- in the following, we use **cross-correlation** $w \star x$ or **convolution** $x * h$, where $h[n] = w[-n]$ is the impulse response
- we call w the **kernel** of the operation

cross-correlation

- convolution is commutative

$$(x * h)[\mathbf{n}] := \sum_{\mathbf{k}} x[\mathbf{k}] h[\mathbf{n} - \mathbf{k}] = \sum_{\mathbf{k}} h[\mathbf{k}] x[\mathbf{n} - \mathbf{k}] = (h * x)[\mathbf{n}]$$

- cross-correlation is not

$$(h \star x)[\mathbf{n}] := \sum_{\mathbf{k}} h[\mathbf{k}] x[\mathbf{k} + \mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}] h[\mathbf{k} - \mathbf{n}] = (x \star h)[-\mathbf{n}]$$

- both are LTI; the only difference is that in cross-correlation, h refers to the **flipped** impulse response
- but if h is even ($h[n] = h[-n]$), then $h \star x = x * h = h * x$
- in the following, we use **cross-correlation** $w \star x$ or **convolution** $x * h$, where $h[n] = w[-n]$ is the impulse response
- we call w the **kernel** of the operation

cross-correlation

- convolution is commutative

$$(x * h)[\mathbf{n}] := \sum_{\mathbf{k}} x[\mathbf{k}] h[\mathbf{n} - \mathbf{k}] = \sum_{\mathbf{k}} h[\mathbf{k}] x[\mathbf{n} - \mathbf{k}] = (h * x)[\mathbf{n}]$$

- cross-correlation is not

$$(h \star x)[\mathbf{n}] := \sum_{\mathbf{k}} h[\mathbf{k}] x[\mathbf{k} + \mathbf{n}] = \sum_{\mathbf{k}} x[\mathbf{k}] h[\mathbf{k} - \mathbf{n}] = (x \star h)[-\mathbf{n}]$$

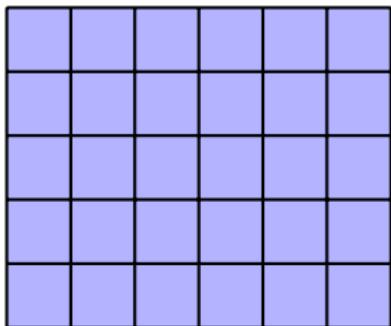
- both are LTI; the only difference is that in cross-correlation, h refers to the **flipped** impulse response
- but if h is even ($h[n] = h[-n]$), then $h \star x = x * h = h * x$
- in the following, we use **cross-correlation** $w \star x$ or **convolution** $x * h$, where $h[n] = w[-n]$ is the impulse response
- we call w the **kernel** of the operation

2d convolution, again

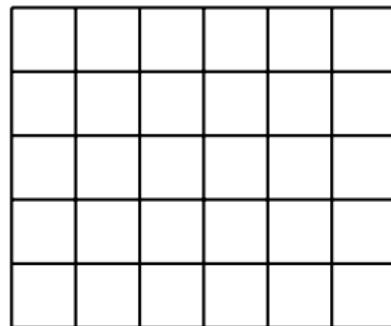
1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



$w \star x$

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

1	2	3			
4	5	6			
7	8	9			

x

w \star x

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

	1	2	3		
	4	5	6		
	7	8	9		

x

w \star x

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

		1	2	3	
		4	5	6	
		7	8	9	

x

w \star x

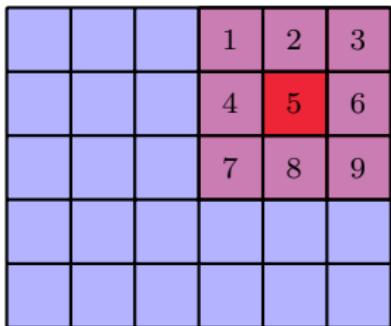
2d convolution, again

1	2	3
4	5	6
7	8	9

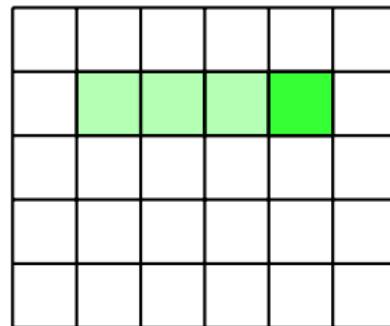
w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$

$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



w \star *x*

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

1	2	3				
4	5	6				
7	8	9				

x

w \star x

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

1	2	3			
4	5	6			
7	8	9			

x

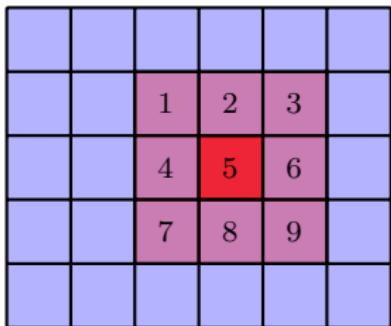
w \star x

2d convolution, again

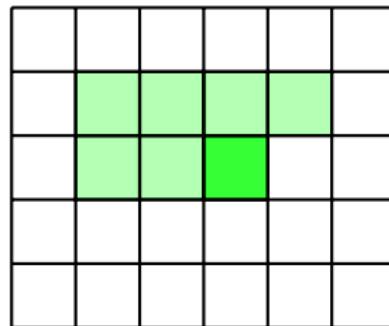
1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



w \star *x*

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

			1	2	3
			4	5	6
			7	8	9

x

w \star x

2d convolution, again

1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$

1	2	3			
4	5	6			
7	8	9			

x

w \star x

2d convolution, again

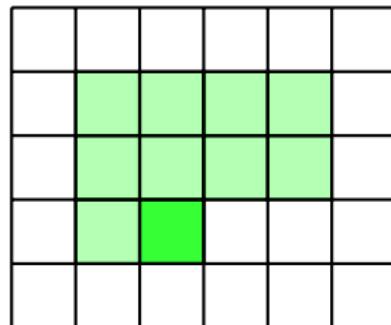
1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



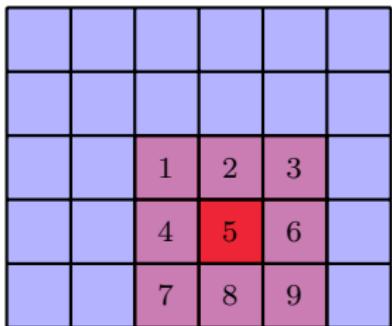
w \star *x*

2d convolution, again

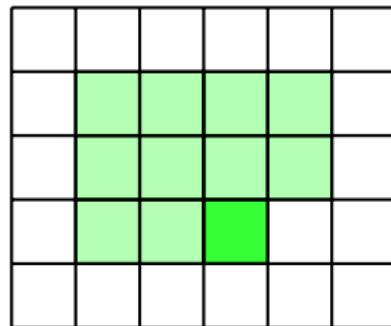
1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



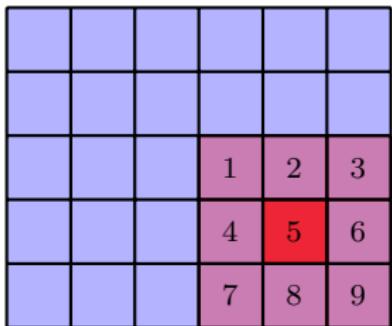
w ∗ x

2d convolution, again

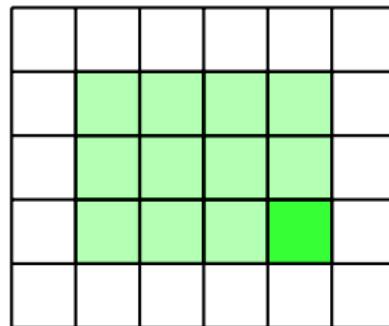
1	2	3
4	5	6
7	8	9

w

$$(w \star x)[\mathbf{n}] = \sum_{\mathbf{k}} w[\mathbf{k}]x[\mathbf{k} + \mathbf{n}]$$
$$= \sum_{\mathbf{k}} x[\mathbf{k}]w[\mathbf{k} - \mathbf{n}]$$



x



w \star x

features

- something is still missing: so far we had activations \mathbf{a} and outputs \mathbf{y} of the form

$$\mathbf{a} = W^\top \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \mathbf{x} + \mathbf{b})$$

where \mathbf{x} is the input, $W = (\mathbf{w}_1, \dots, \mathbf{w}_k)$ a weight matrix and \mathbf{b} a bias

- the elements of \mathbf{x} , \mathbf{a} , \mathbf{b} and \mathbf{y} were representing **features** (or cells);
the elements of W were representing **connections**
- now we have x as a 2d array, w as a 2d kernel, but no features yet

feature maps

- now \mathbf{b} remains a vector but \mathbf{x} , \mathbf{a} , \mathbf{y} become **3d tensors** with input feature i and output feature j at spatial position \mathbf{n} represented by

$$x_i[\mathbf{n}], \quad a_j[\mathbf{n}], \quad b_j, \quad y_j[\mathbf{n}]$$

- x_i and y_j are 2d arrays we call **feature maps**, each corresponding to one feature; and a_j a 2d array we call **activation map**
- if x_i refers to the input image, there is just one feature that is the image intensity of a monochrome image, or three features corresponding to the three **channels** of a color image
- W becomes a **4d tensor** with a connection from input feature i to output feature j at spatial position \mathbf{k} represented by

$$w_{ij}[\mathbf{k}]$$

feature maps

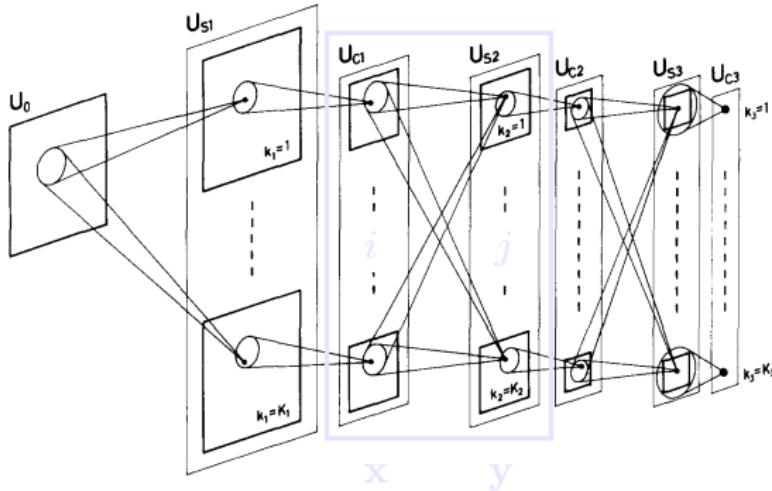
- now \mathbf{b} remains a vector but \mathbf{x} , \mathbf{a} , \mathbf{y} become **3d tensors** with input feature i and output feature j at spatial position \mathbf{n} represented by

$$x_i[\mathbf{n}], \quad a_j[\mathbf{n}], \quad b_j, \quad y_j[\mathbf{n}]$$

- x_i and y_j are 2d arrays we call **feature maps**, each corresponding to one feature; and a_j a 2d array we call **activation map**
- if x_i refers to the input image, there is just one feature that is the image intensity of a monochrome image, or three features corresponding to the three **channels** of a color image
- W becomes a **4d tensor** with a connection from input feature i to output feature j at spatial position \mathbf{k} represented by

$$w_{ij}[\mathbf{k}]$$

convolution on feature maps

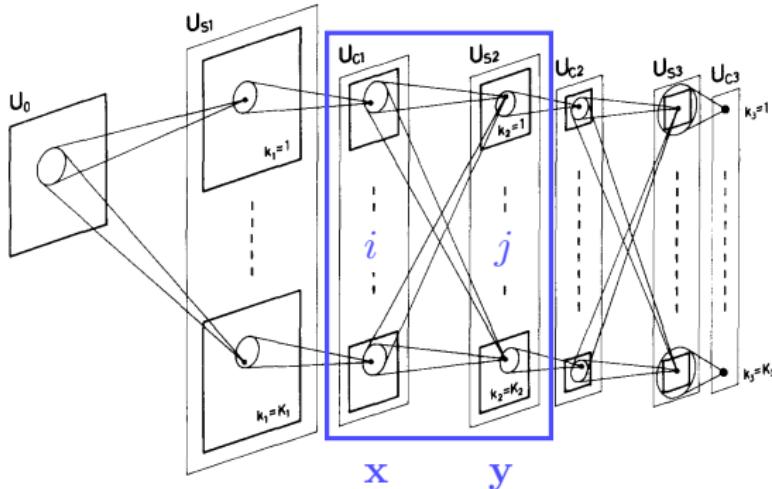


- matrix multiplication and convolution combined

$$\mathbf{a} = W^T \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^T \star \mathbf{x} + \mathbf{b})$$

$$(W^T \star \mathbf{x})_j[n] = (\mathbf{w}_j^T \star \mathbf{x})[n] := \sum_i (w_{ij} \star x_i)[n] = \sum_k w_{ij}[k] x_i[k+n]$$

convolution on feature maps



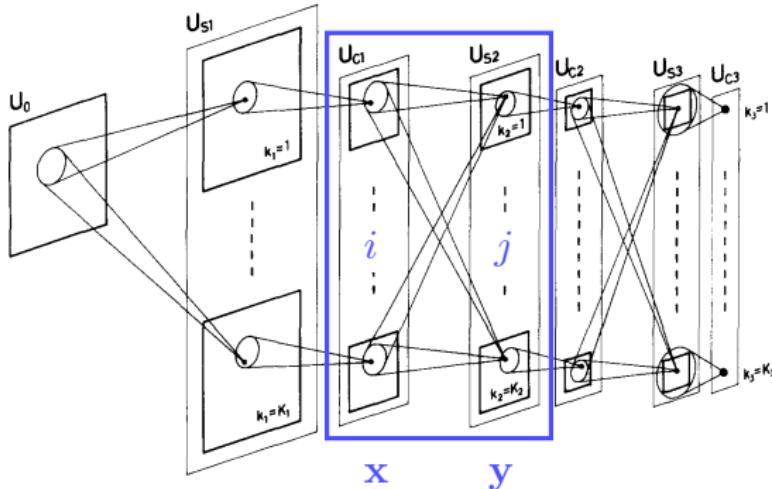
- matrix multiplication and convolution combined

$$\mathbf{a} = W^T \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^T \star \mathbf{x} + \mathbf{b})$$

$$(W^T \star \mathbf{x})_j[n] = (\mathbf{w}_j^T \star \mathbf{x})[n] := \sum_i (w_{ij} \star x_i)[n] = \sum_k w_{ij}[k] x_i[k+n]$$

Fukushima. BC 1980. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected By Shift in Position.

convolution on feature maps



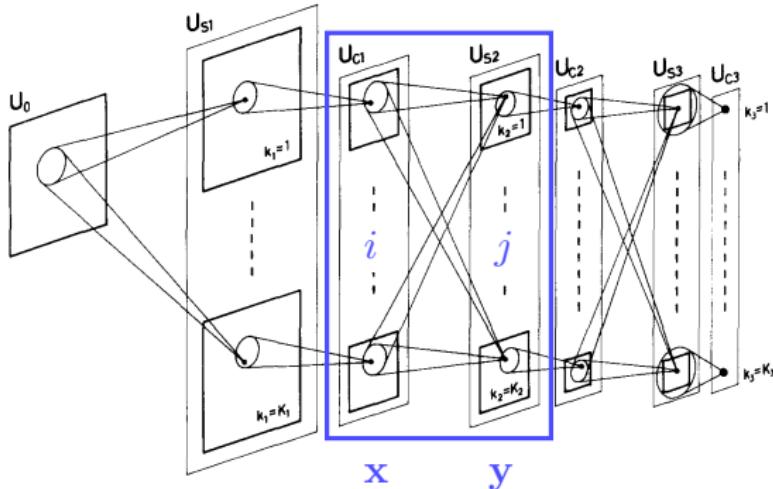
- matrix multiplication and convolution combined

$$\mathbf{a} = W^\top \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \star \mathbf{x} + \mathbf{b})$$

$$(W^\top \star \mathbf{x})_j[\mathbf{n}] = (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] := \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_k w_{ij}[k] x_i[k + n]$$

Fukushima. BC 1980. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected By Shift in Position.

convolution on feature maps



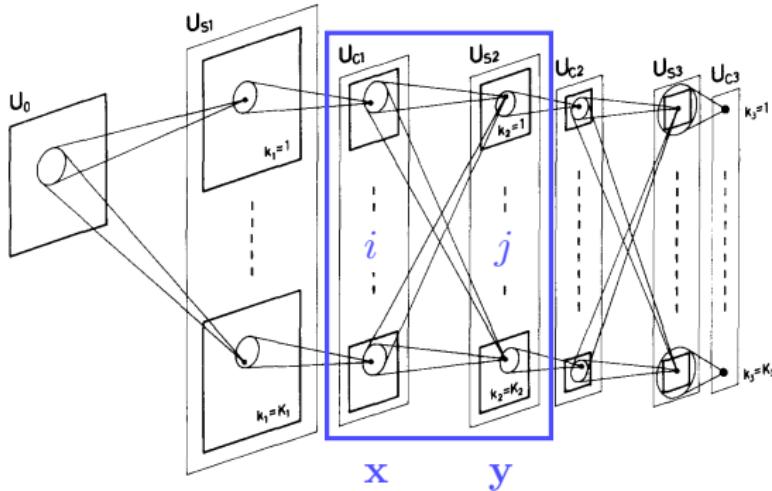
- matrix multiplication and convolution combined

$$\mathbf{a} = W^\top \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \star \mathbf{x} + \mathbf{b})$$

$$(W^\top \star \mathbf{x})_j[\mathbf{n}] = (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] := \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_k w_{ij}[k] x_i[k+n]$$

Fukushima. BC 1980. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected By Shift in Position.

convolution on feature maps

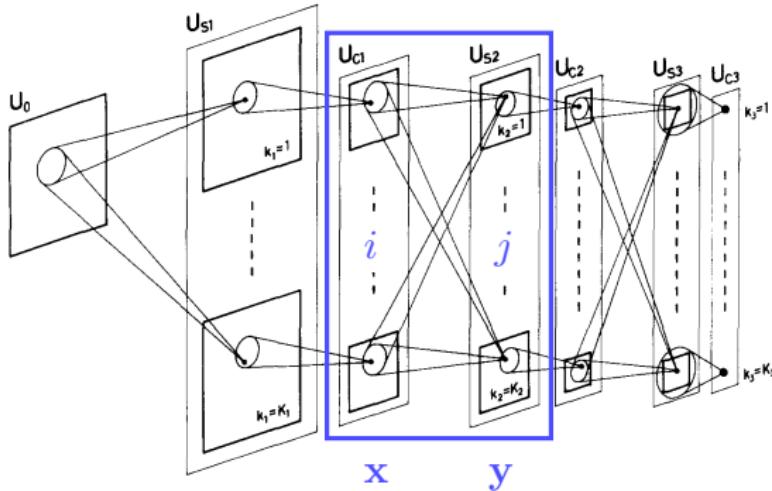


- matrix multiplication and convolution combined

$$\mathbf{a} = W^\top \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \star \mathbf{x} + \mathbf{b})$$

$$(W^\top \star \mathbf{x})_j[\mathbf{n}] = (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] := \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{\mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}]$$

convolution on feature maps

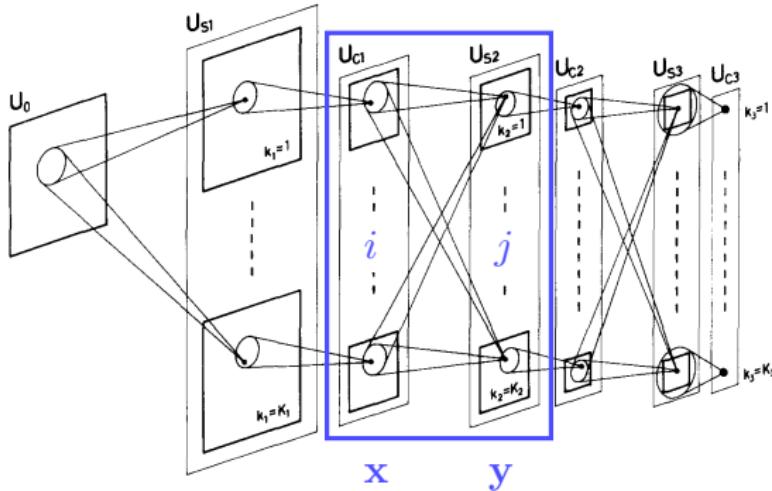


- matrix multiplication and convolution combined

$$\mathbf{a} = W^\top \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \star \mathbf{x} + \mathbf{b})$$

$$(W^\top \star \mathbf{x})_j[\mathbf{n}] = (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] := \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{i, \mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}]$$

convolution on feature maps



- matrix multiplication and convolution combined

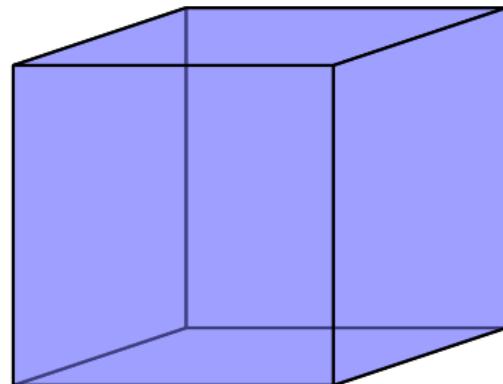
$$\mathbf{a} = W^\top \star \mathbf{x} + \mathbf{b}, \quad \mathbf{y} = h(\mathbf{a}) = h(W^\top \star \mathbf{x} + \mathbf{b})$$

$$(W^\top \star \mathbf{x})_j[\mathbf{n}] = (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] := \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{i,\mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}]$$

convolution on feature maps

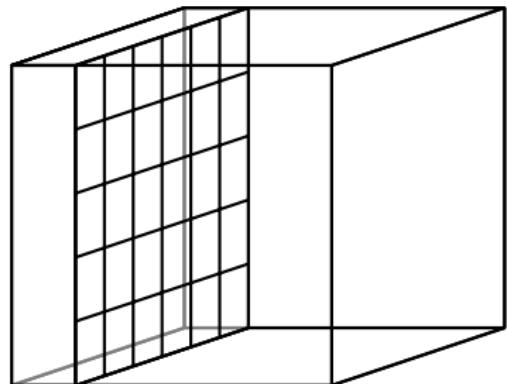


kernel w_1



input x

kernel weights shared
among all spatial positions

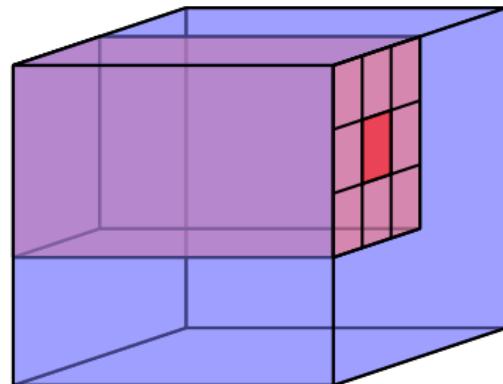


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

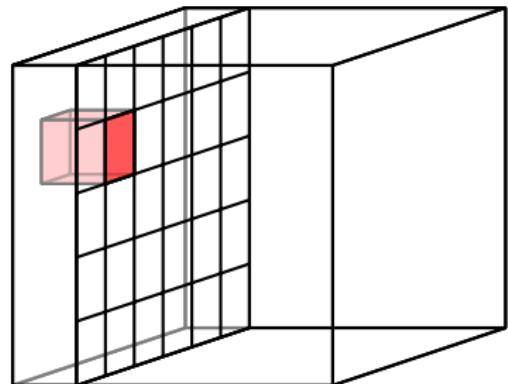


kernel w_1



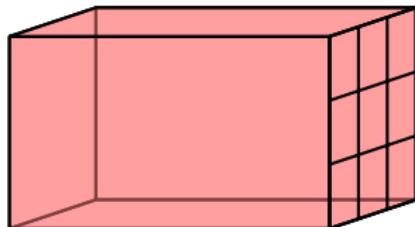
input x

kernel weights shared
among all spatial positions

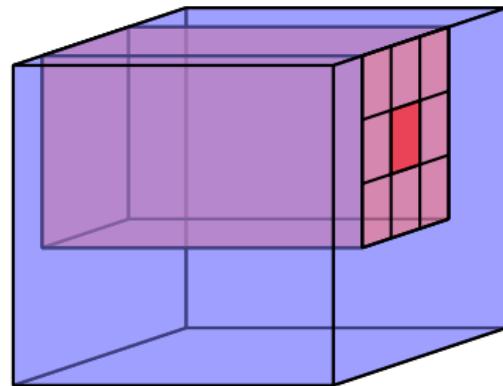


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

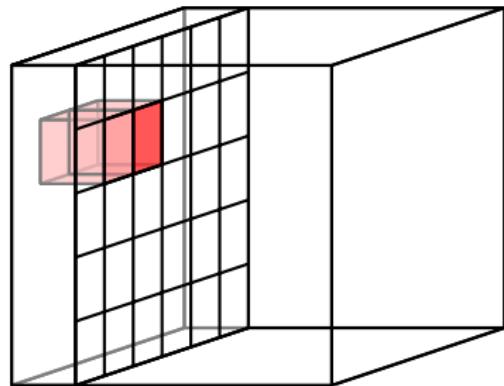


kernel w_1



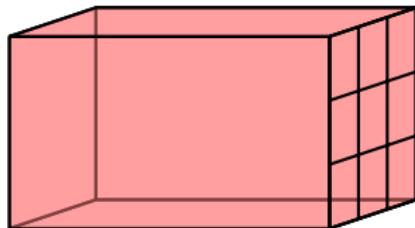
input x

kernel weights shared
among all spatial positions

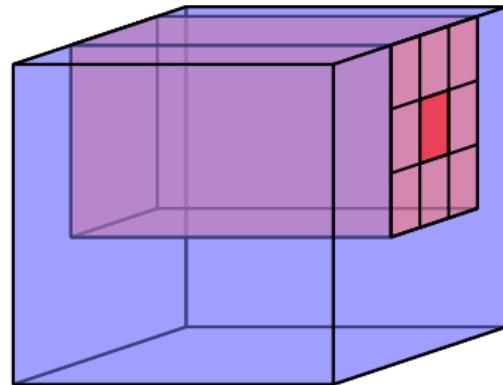


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

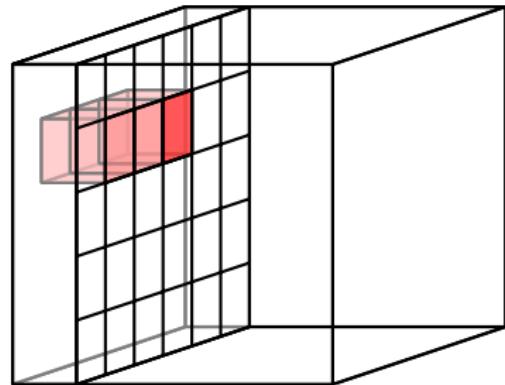


kernel w_1



input x

kernel weights shared
among all spatial positions

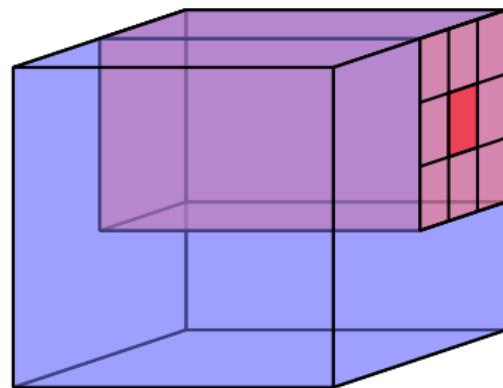


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

convolution on feature maps

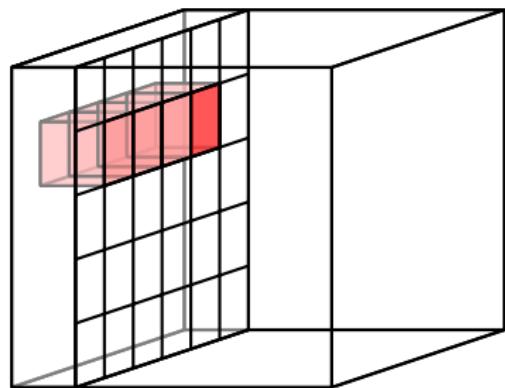


kernel w_1



input x

kernel weights shared
among all spatial positions

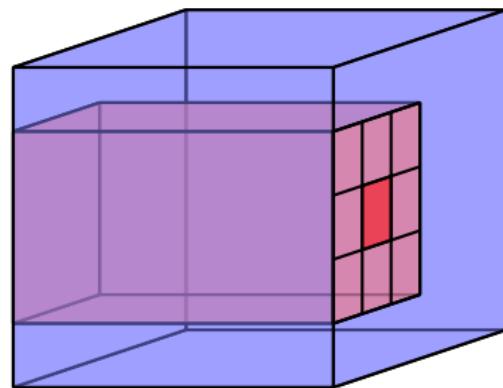


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

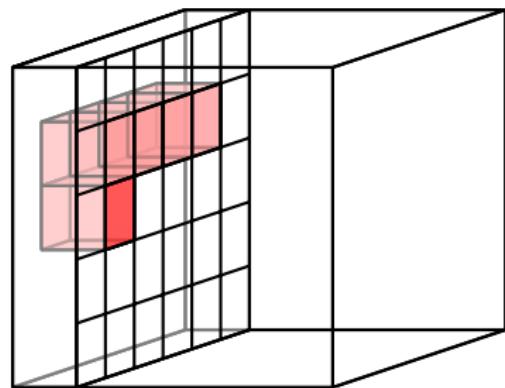


kernel w_1



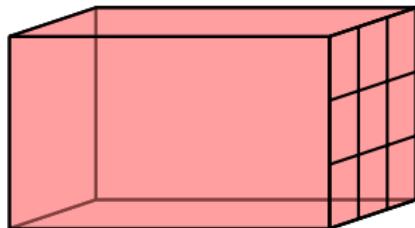
input x

kernel weights shared
among all spatial positions

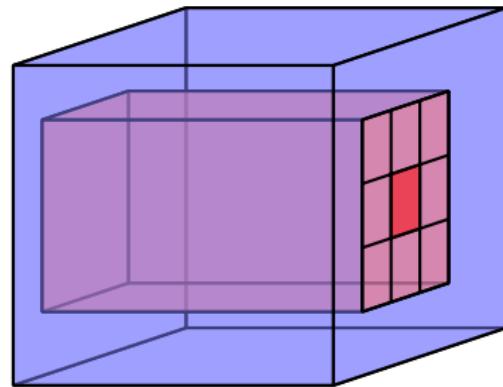


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

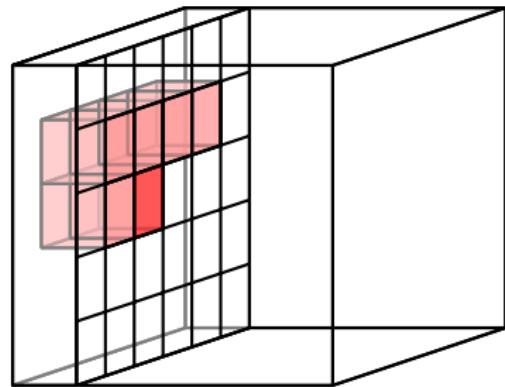


kernel w_1



input x

kernel weights shared
among all spatial positions

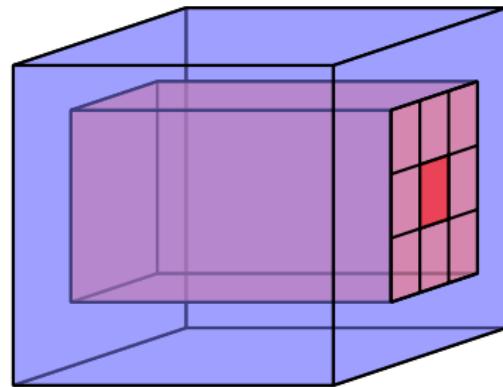


$$\text{output } y_1 = h(w_1^T \star x + b_1)$$

convolution on feature maps

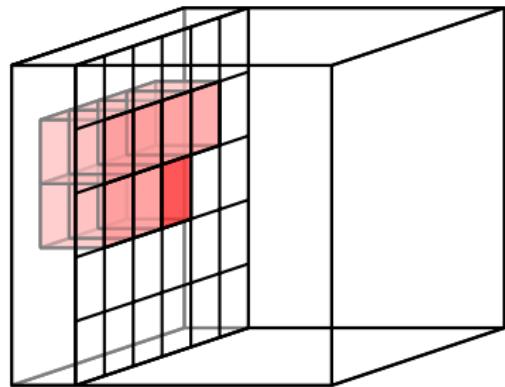


kernel w_1



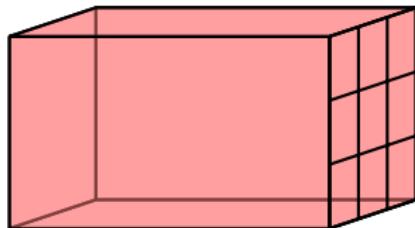
input x

kernel weights shared
among all spatial positions

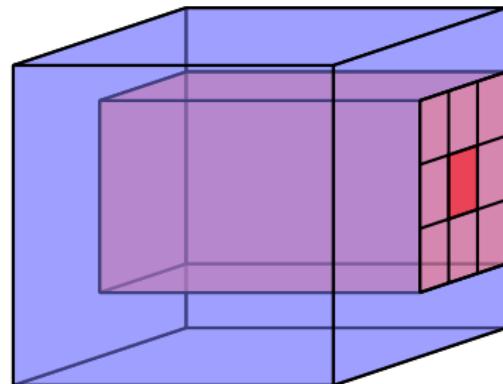


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

convolution on feature maps

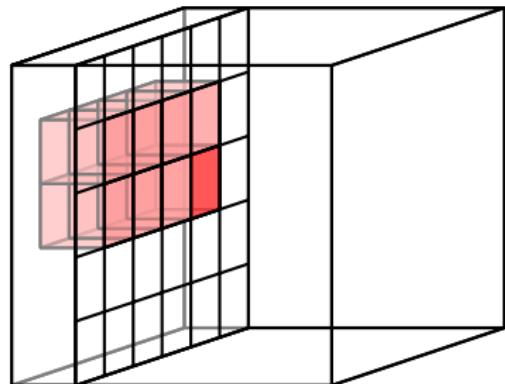


kernel w_1



input x

kernel weights shared
among all spatial positions

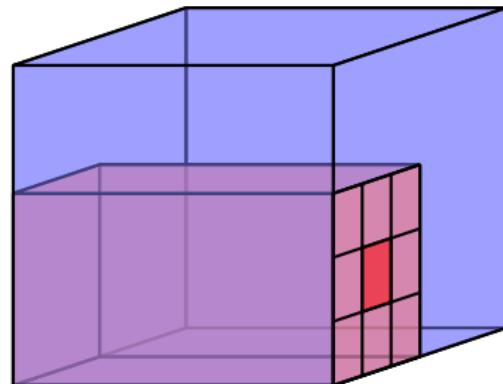


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

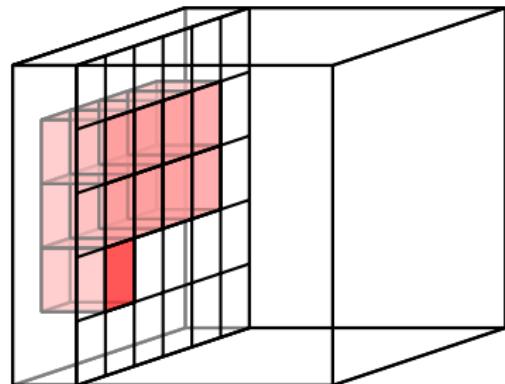


kernel w_1



input x

kernel weights shared
among all spatial positions

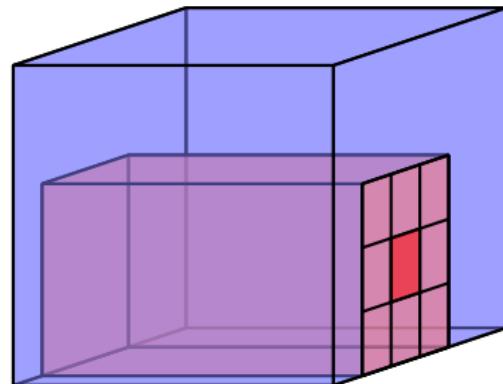


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

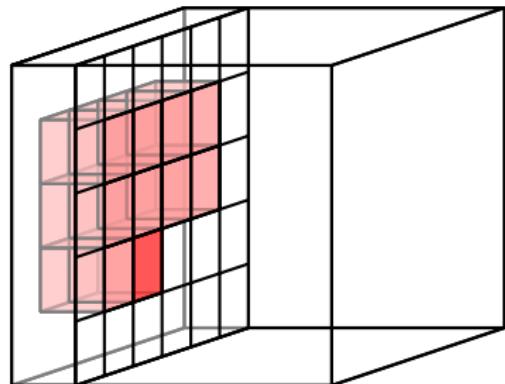


kernel w_1



input x

kernel weights shared
among all spatial positions



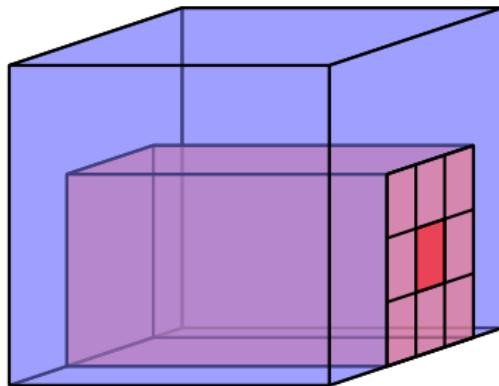
$$\text{output } y_1 = h(w_1^T \star x + b_1)$$

convolution on feature maps

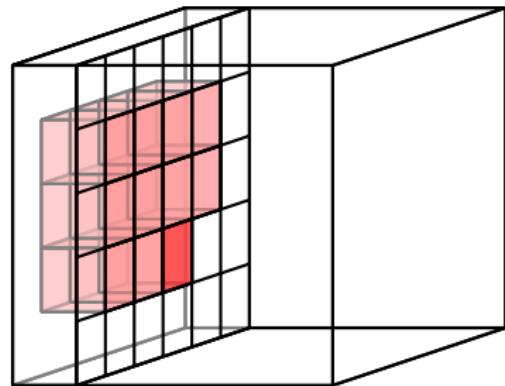


kernel w_1

kernel weights shared
among all spatial positions

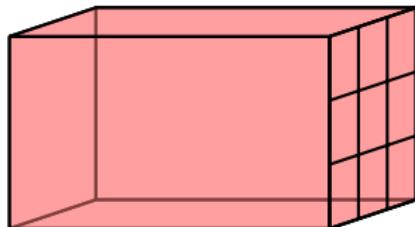


input x

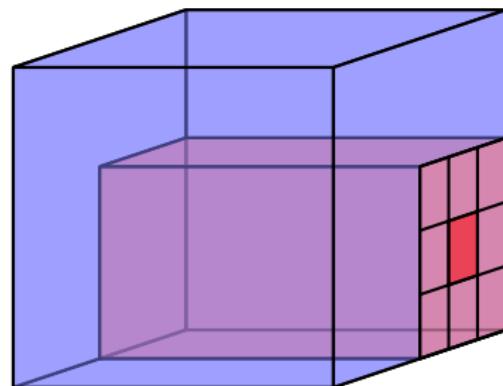


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

convolution on feature maps

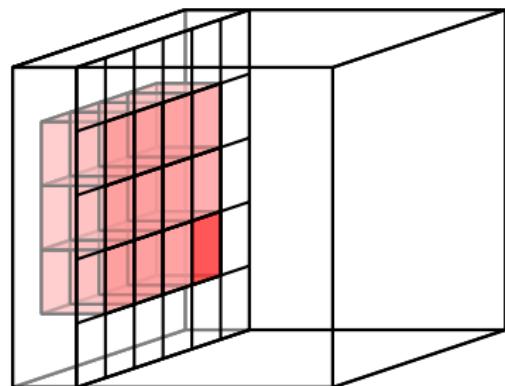


kernel w_1



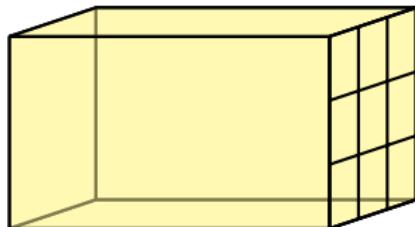
input x

kernel weights shared
among all spatial positions

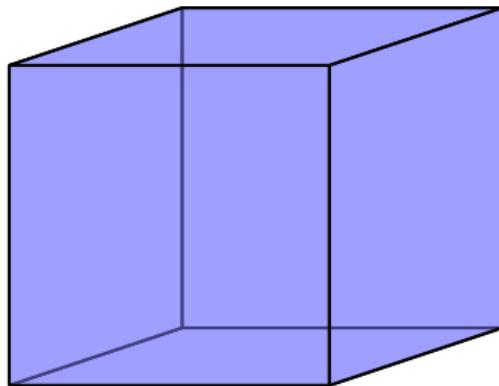


$$\text{output } y_1 = h(w_1^\top \star x + b_1)$$

convolution on feature maps

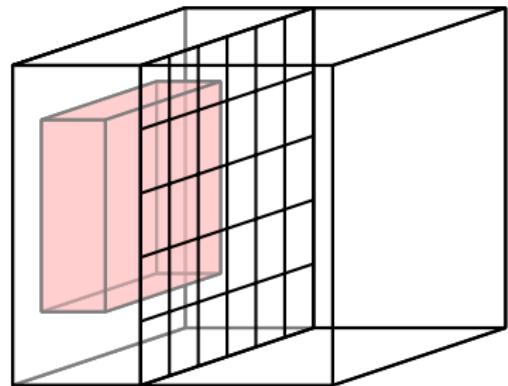


kernel w_2



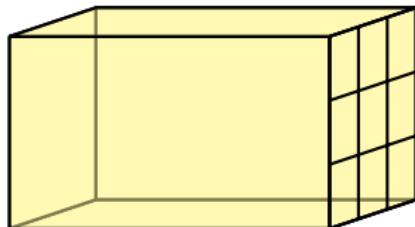
input x

new kernel, but still shared
among all spatial positions

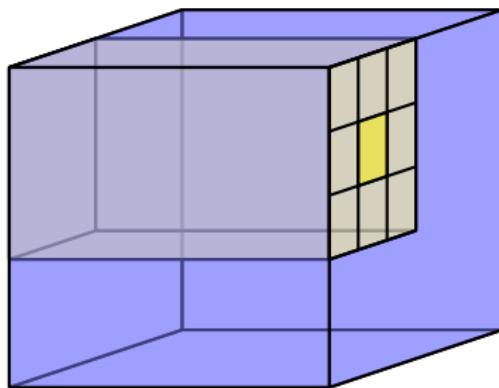


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

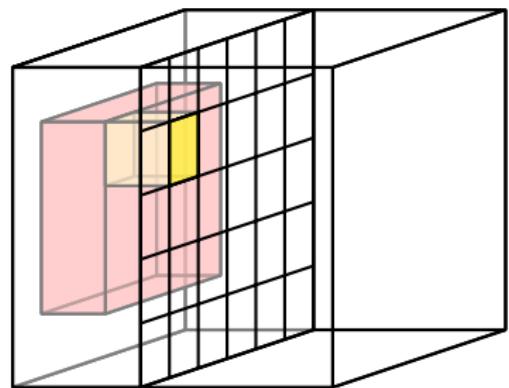


kernel w_2



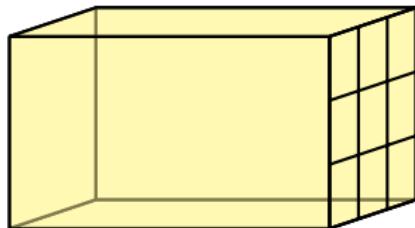
input x

new kernel, but still shared
among all spatial positions

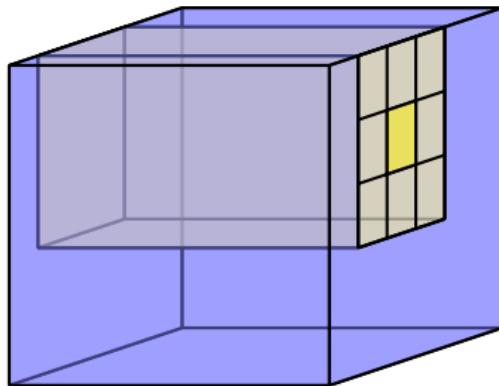


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

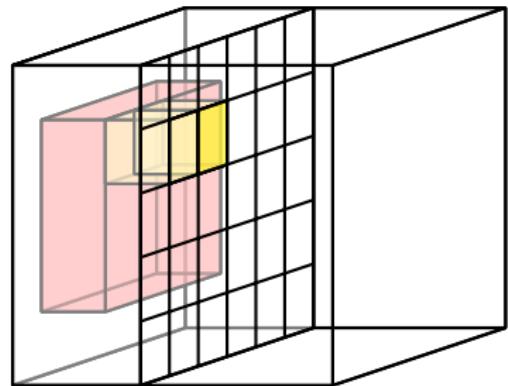


kernel w_2



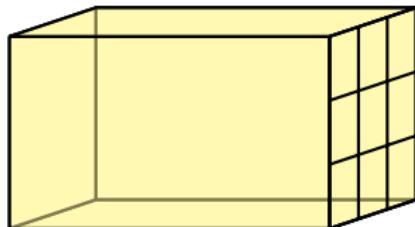
input x

new kernel, but still shared
among all spatial positions

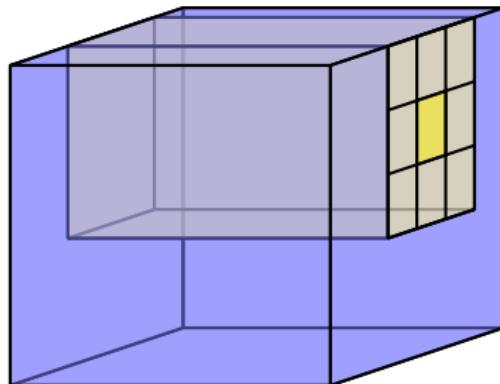


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

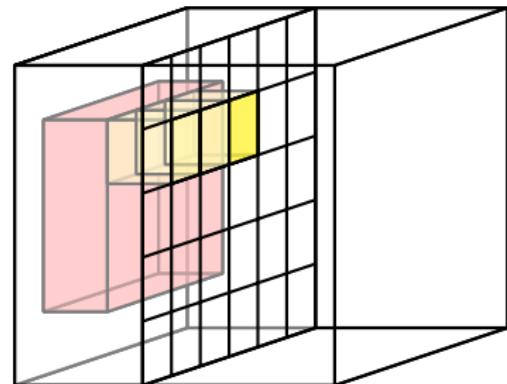


kernel w_2



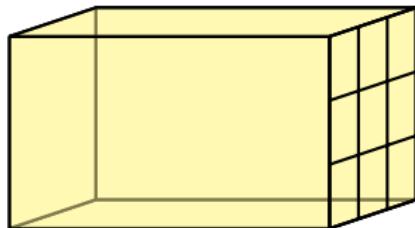
input x

new kernel, but still shared
among all spatial positions

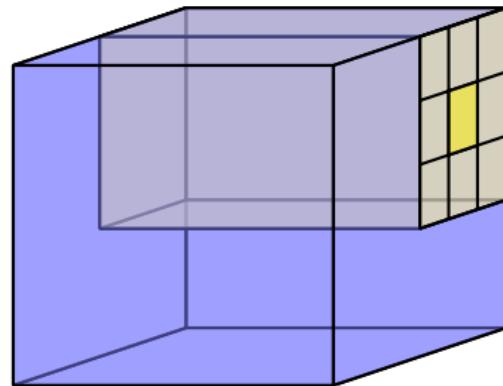


$$\text{output } y_2 = h(w_2^\top \star x + b_2)$$

convolution on feature maps

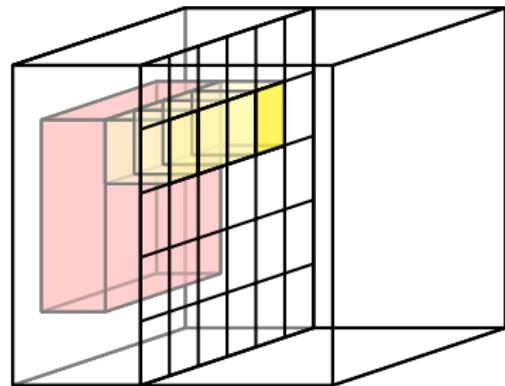


kernel w_2



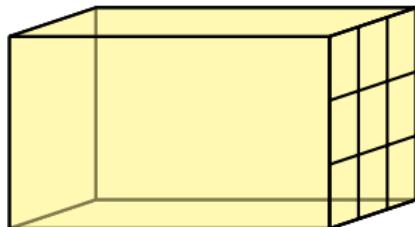
input x

new kernel, but still shared
among all spatial positions

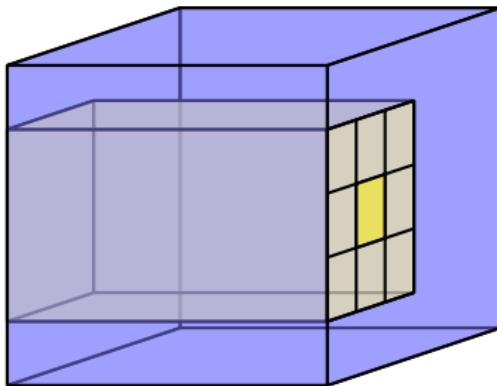


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

convolution on feature maps

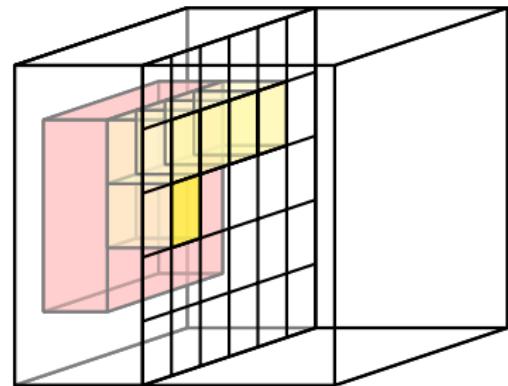


kernel w_2



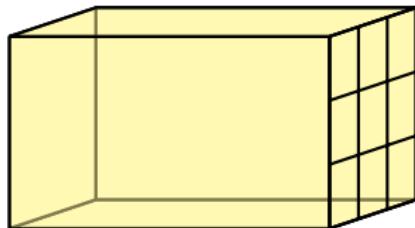
input x

new kernel, but still shared
among all spatial positions

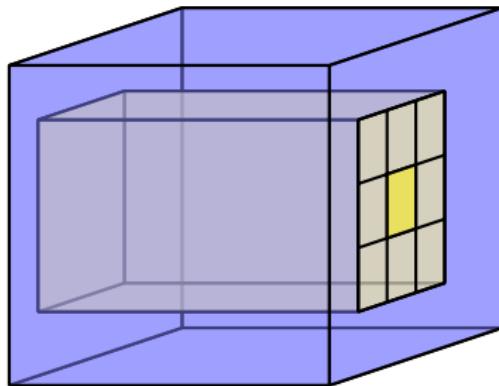


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

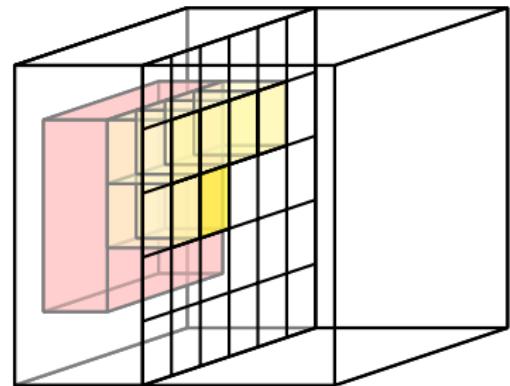


kernel w_2



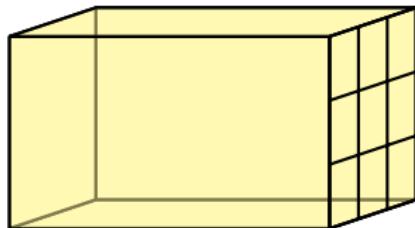
input x

new kernel, but still shared
among all spatial positions

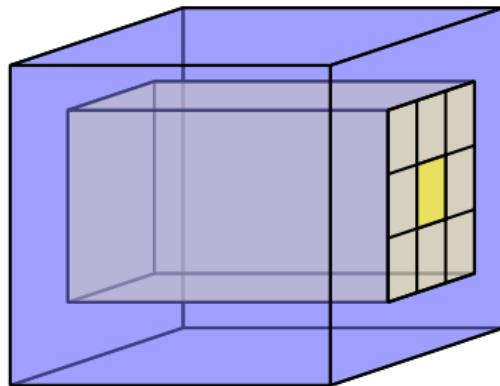


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

convolution on feature maps

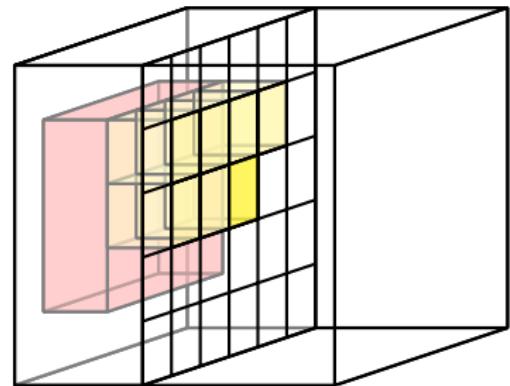


kernel w_2



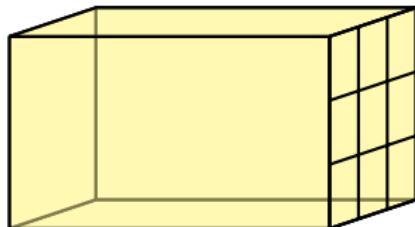
input x

new kernel, but still shared
among all spatial positions

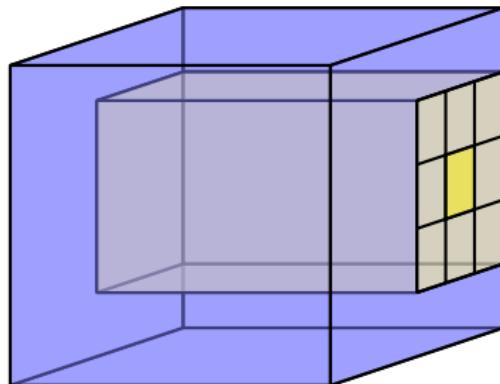


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

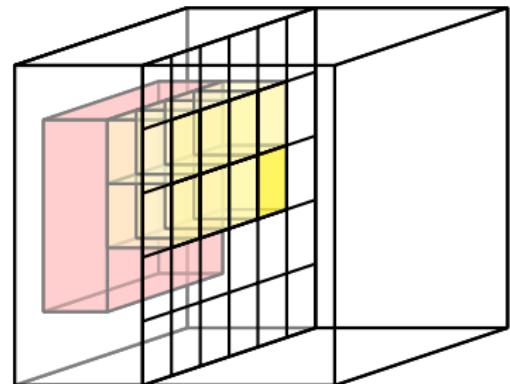


kernel w_2



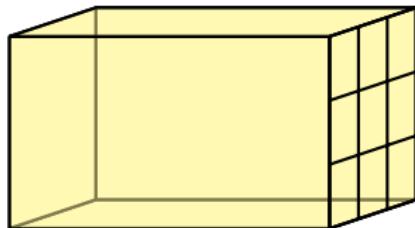
input x

new kernel, but still shared
among all spatial positions

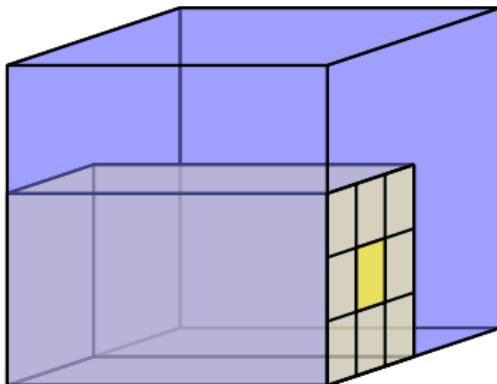


$$\text{output } y_2 = h(w_2^\top \star x + b_2)$$

convolution on feature maps

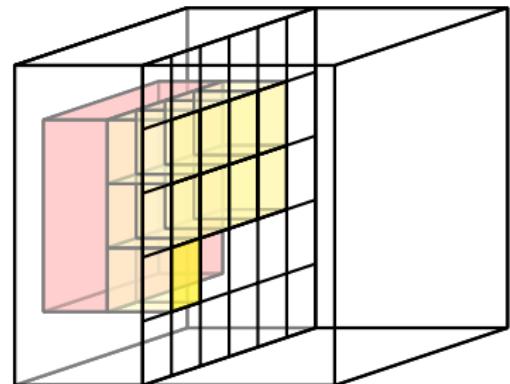


kernel w_2



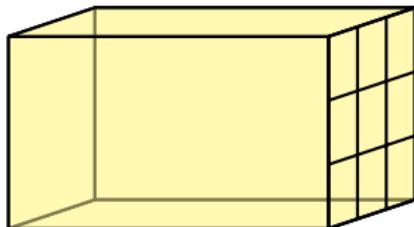
input x

new kernel, but still shared
among all spatial positions

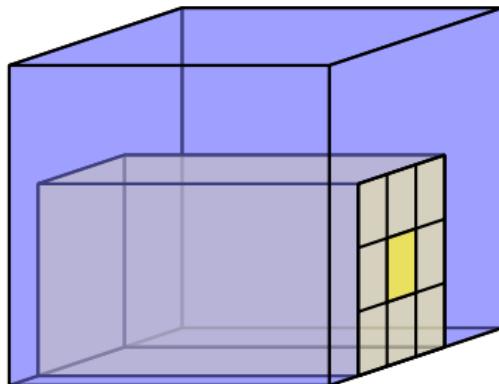


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

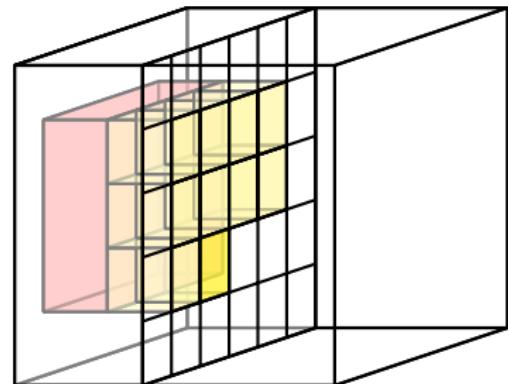


kernel w_2



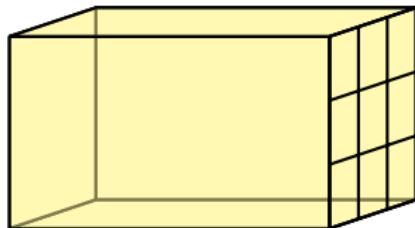
input x

new kernel, but still shared
among all spatial positions

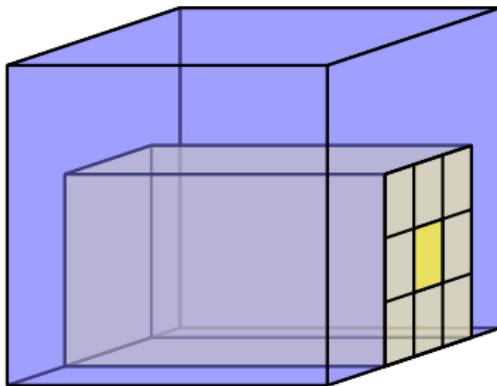


$$\text{output } y_2 = h(w_2^\top \star x + b_2)$$

convolution on feature maps

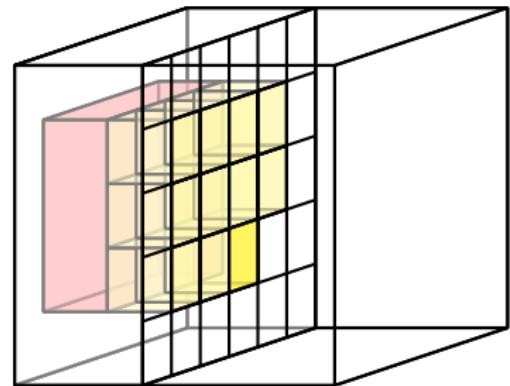


kernel w_2



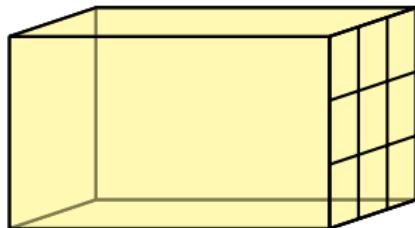
input x

new kernel, but still shared
among all spatial positions

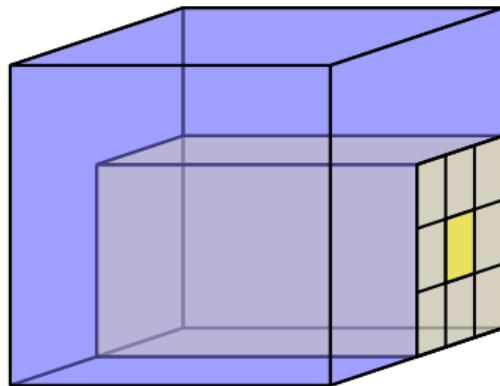


$$\text{output } y_2 = h(w_2^\top \star x + b_2)$$

convolution on feature maps

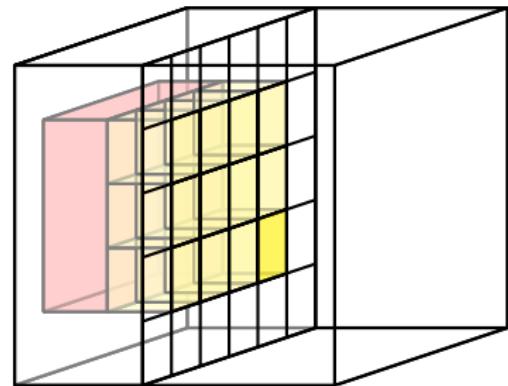


kernel w_2



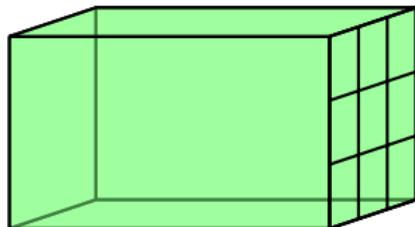
input x

new kernel, but still shared
among all spatial positions

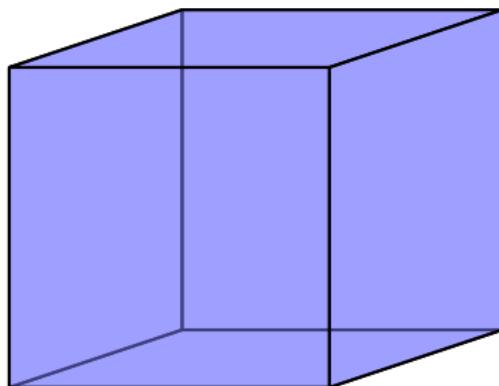


$$\text{output } y_2 = h(w_2^T \star x + b_2)$$

convolution on feature maps

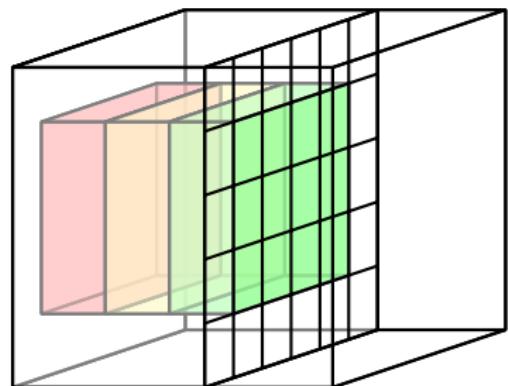


kernel w_3



input x

different kernel for
each output dimension

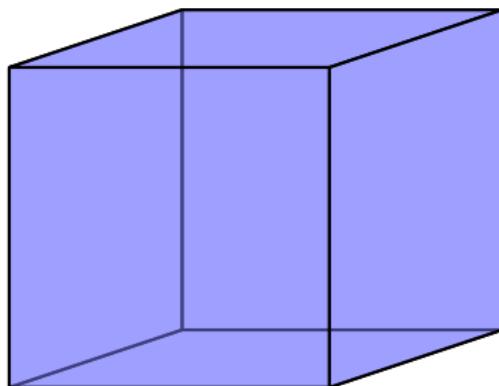


$$\text{output } y_3 = h(w_3^T \star x + b_3)$$

convolution on feature maps

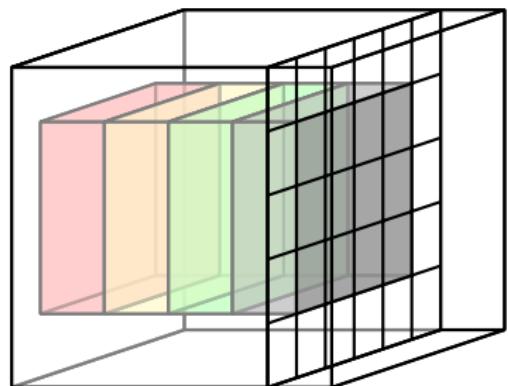


kernel w_4



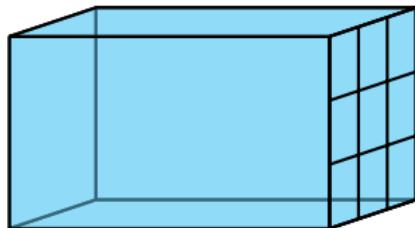
input x

different kernel for
each output dimension

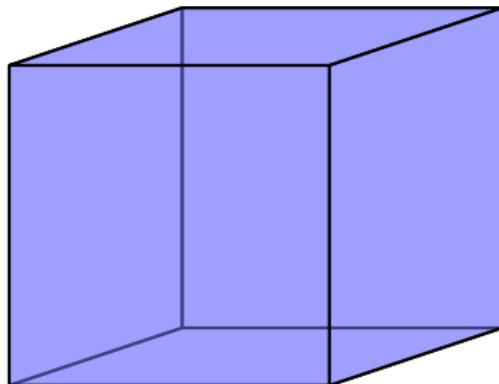


$$\text{output } y_4 = h(\mathbf{w}_4^\top \star \mathbf{x} + b_4)$$

convolution on feature maps

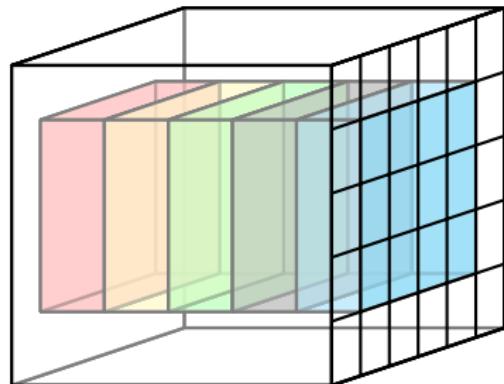


kernel w_5



input x

different kernel for
each output dimension



$$\text{output } y_5 = h(w_5^T \star x + b_5)$$

1×1 convolution

- if W has no spatial extent, it becomes a **2d matrix** again

$$\begin{aligned} (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] &:= \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{i,\mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}] \\ &= \sum_i w_{ij} x_i[\mathbf{n}] = \mathbf{w}_j^\top \mathbf{x}[\mathbf{n}] \end{aligned}$$

- the operation becomes a **matrix multiplication** just as in fully-connected layers, but now it is performed independently at each spatial location

$$(W^\top \star \mathbf{x})[\mathbf{n}] = W^\top \mathbf{x}[\mathbf{n}]$$

$$W^\top \star \mathbf{x} = W^\top \mathbf{x}$$

1×1 convolution

- if W has no spatial extent, it becomes a **2d matrix** again

$$\begin{aligned} (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] &:= \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{i,\mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}] \\ &= \sum_i w_{ij} x_i[\mathbf{n}] = \mathbf{w}_j^\top \mathbf{x}[\mathbf{n}] \end{aligned}$$

- the operation becomes a **matrix multiplication** just as in fully-connected layers, but now it is performed independently at each spatial location

$$(W^\top \star \mathbf{x})[\mathbf{n}] = W^\top \mathbf{x}[\mathbf{n}]$$

$$W^\top \star \mathbf{x} = W^\top \mathbf{x}$$

1×1 convolution

- if W has no spatial extent, it becomes a **2d matrix** again

$$\begin{aligned} (\mathbf{w}_j^\top \star \mathbf{x})[\mathbf{n}] &:= \sum_i (w_{ij} \star x_i)[\mathbf{n}] = \sum_{i,\mathbf{k}} w_{ij}[\mathbf{k}] x_i[\mathbf{k} + \mathbf{n}] \\ &= \sum_i w_{ij} x_i[\mathbf{n}] = \mathbf{w}_j^\top \mathbf{x}[\mathbf{n}] \end{aligned}$$

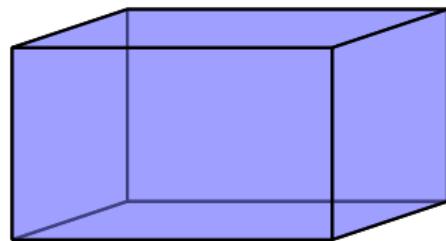
- the operation becomes a **matrix multiplication** just as in fully-connected layers, but now it is performed independently at each spatial location

$$\begin{aligned} (W^\top \star \mathbf{x})[\mathbf{n}] &= W^\top \mathbf{x}[\mathbf{n}] \\ W^\top \star \mathbf{x} &= W^\top \mathbf{x} \end{aligned}$$

1×1 convolution

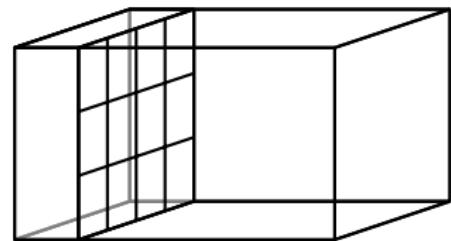


kernel w_1



input x

kernel weights shared
among all spatial positions

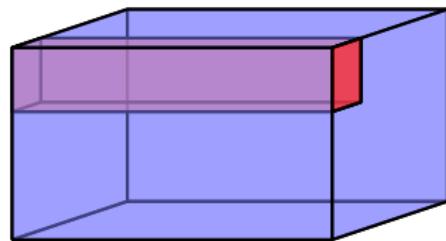


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

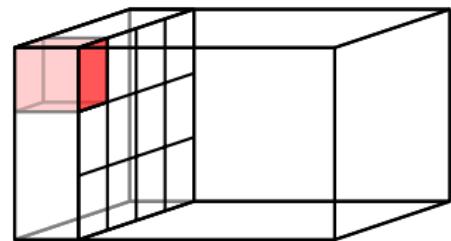


kernel w_1



input x

kernel weights shared
among all spatial positions

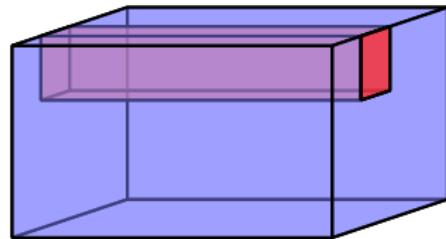


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

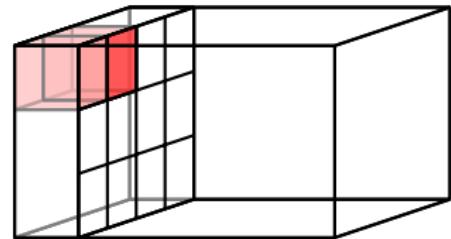


kernel w_1



input x

kernel weights shared
among all spatial positions

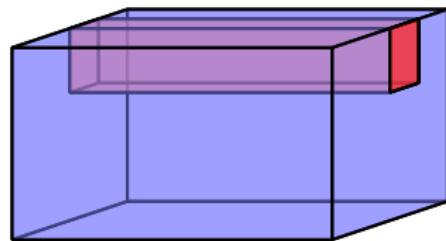


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

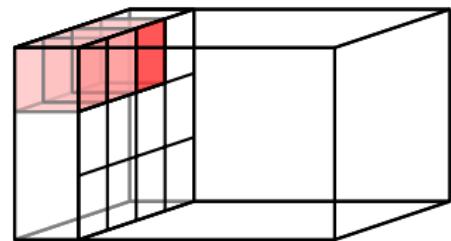


kernel w_1



input x

kernel weights shared
among all spatial positions

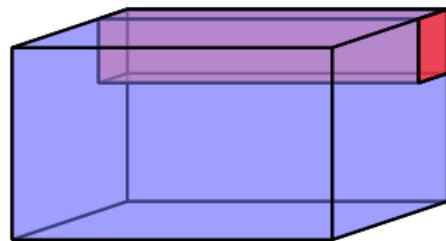


$$\text{output } y_1 = h(w_1^\top * x + b_1)$$

1×1 convolution

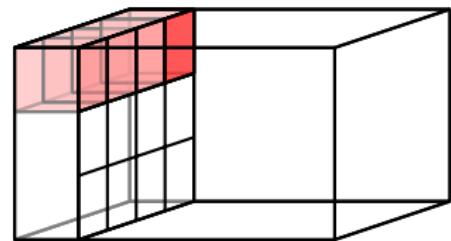


kernel w_1



input x

kernel weights shared
among all spatial positions

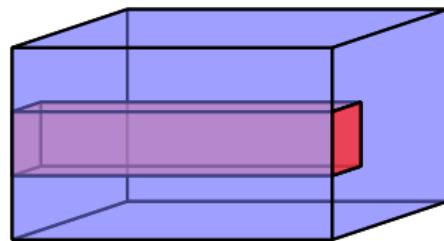


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

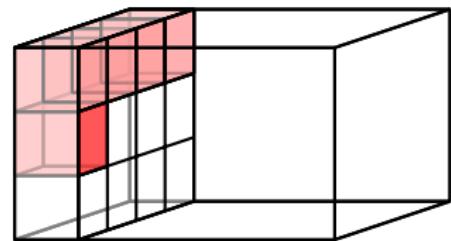


kernel w_1



input x

kernel weights shared
among all spatial positions

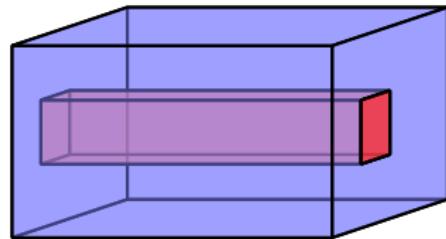


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

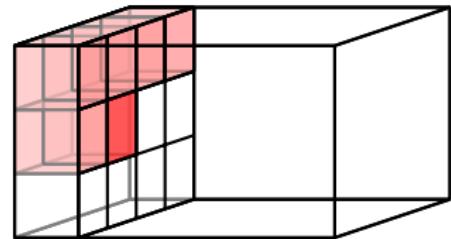


kernel w_1



input x

kernel weights shared
among all spatial positions

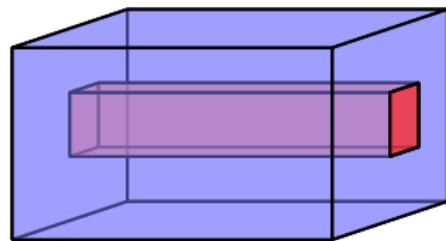


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

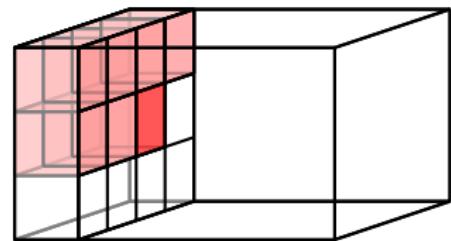


kernel w_1



input x

kernel weights shared
among all spatial positions

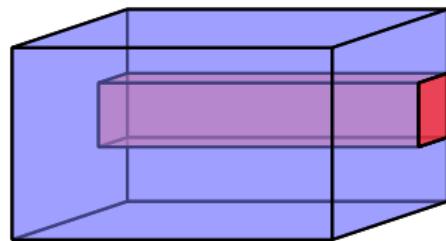


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

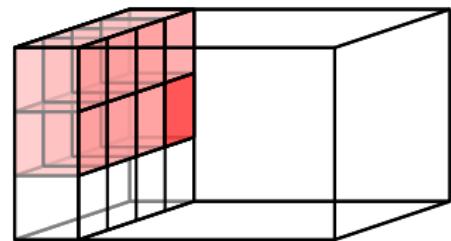


kernel w_1



input x

kernel weights shared
among all spatial positions

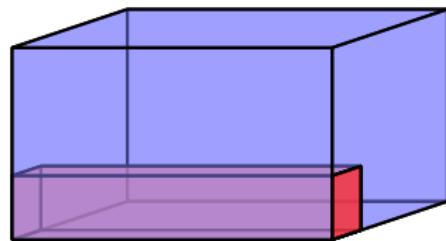


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

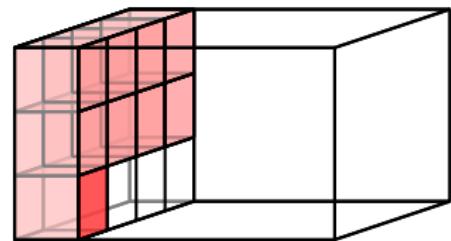


kernel w_1



input x

kernel weights shared
among all spatial positions

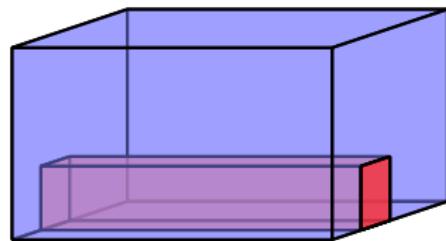


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

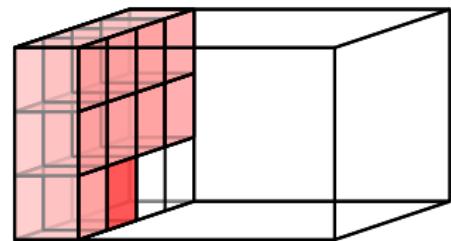


kernel w_1



input x

kernel weights shared
among all spatial positions

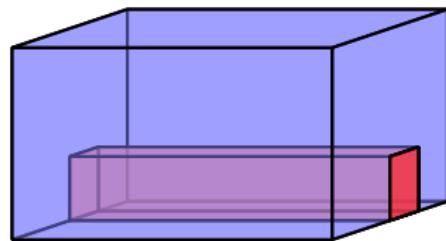


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

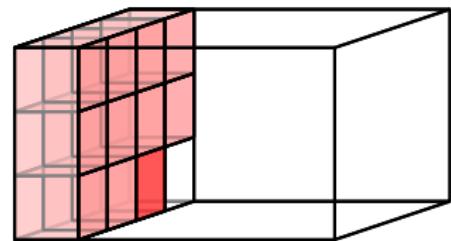


kernel w_1



input x

kernel weights shared
among all spatial positions

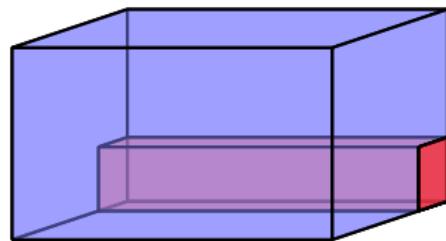


$$\text{output } y_1 = h(\mathbf{w}_1^\top \star \mathbf{x} + b_1)$$

1×1 convolution

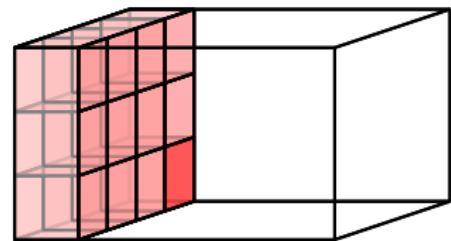


kernel w_1



input x

kernel weights shared
among all spatial positions

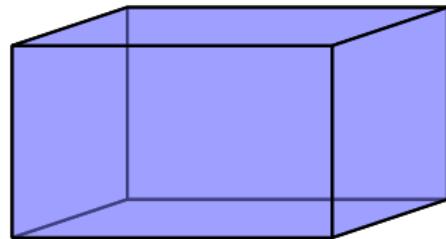


$$\text{output } y_1 = h(w_1^\top * x + b_1)$$

1×1 convolution

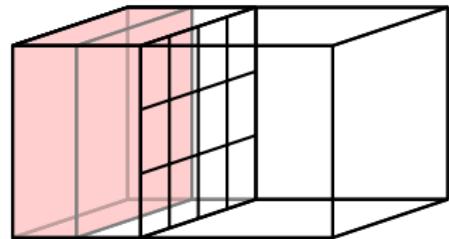


kernel w_2



input x

new kernel, but still shared
among all spatial positions

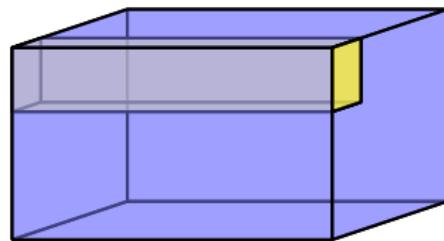


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

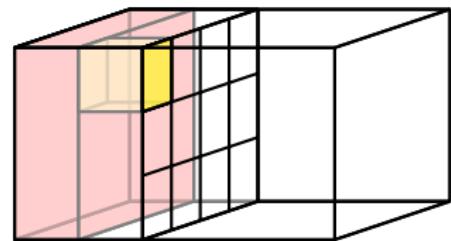


kernel w_2



input \mathbf{x}

new kernel, but still shared
among all spatial positions

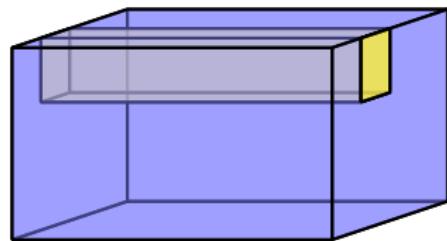


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

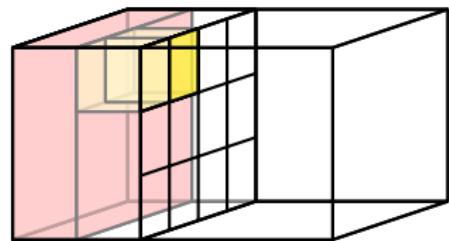


kernel w_2



input x

new kernel, but still shared
among all spatial positions

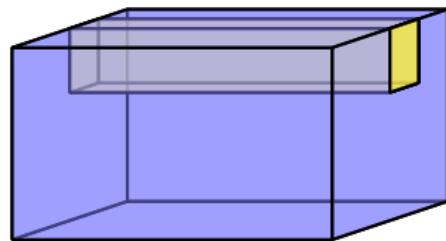


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

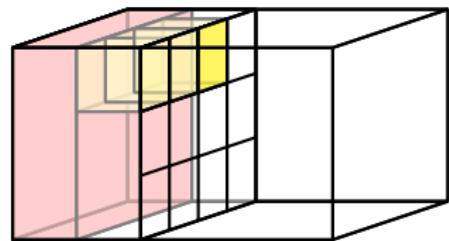


kernel w_2



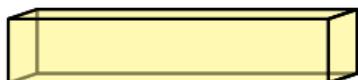
input x

new kernel, but still shared
among all spatial positions

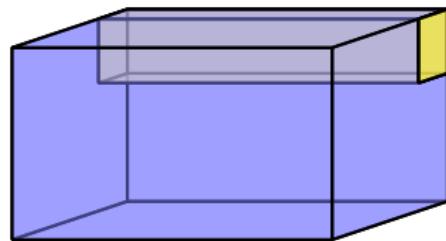


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

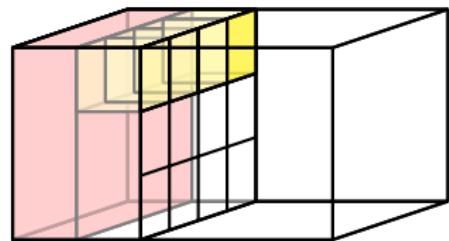


kernel w_2



input x

new kernel, but still shared
among all spatial positions

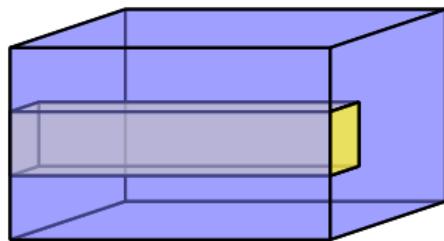


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

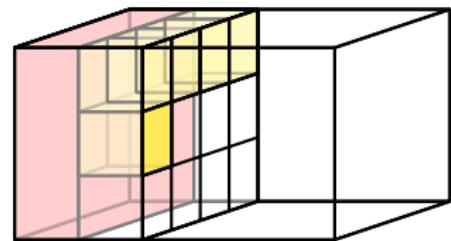


kernel w_2



input x

new kernel, but still shared
among all spatial positions

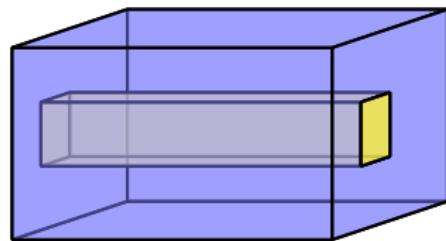


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

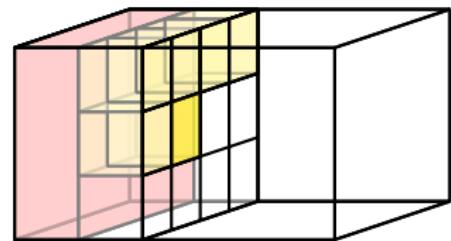


kernel w_2



input x

new kernel, but still shared
among all spatial positions

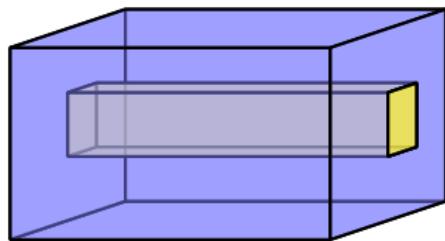


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

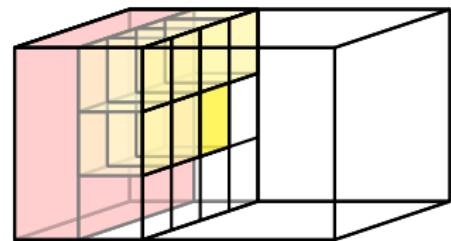


kernel w_2



input x

new kernel, but still shared
among all spatial positions

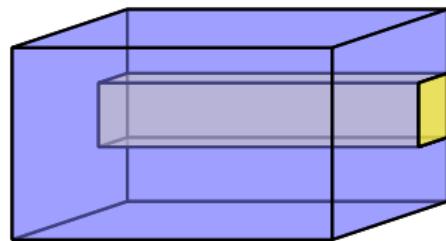


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

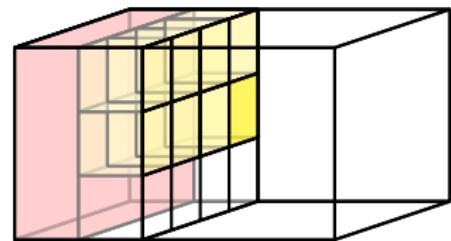


kernel w_2



input x

new kernel, but still shared
among all spatial positions

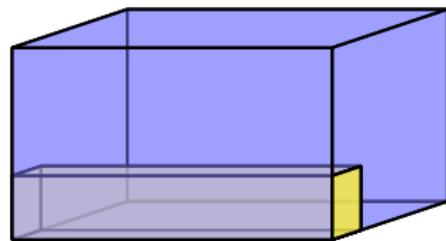


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

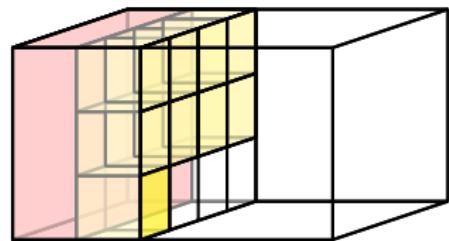


kernel w_2



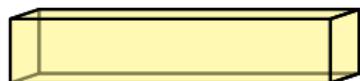
input x

new kernel, but still shared
among all spatial positions

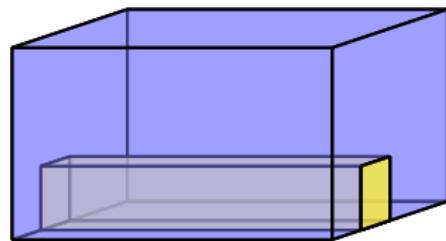


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

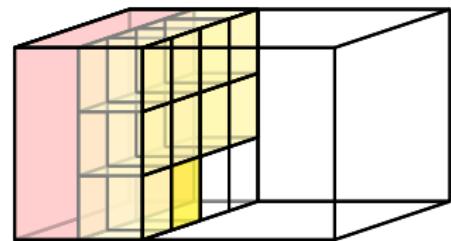


kernel w_2



input x

new kernel, but still shared
among all spatial positions

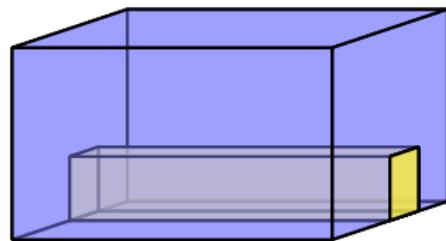


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

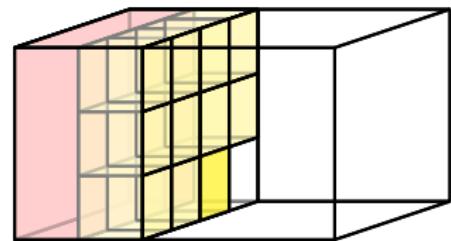


kernel w_2



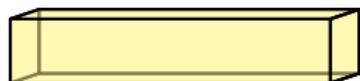
input x

new kernel, but still shared
among all spatial positions

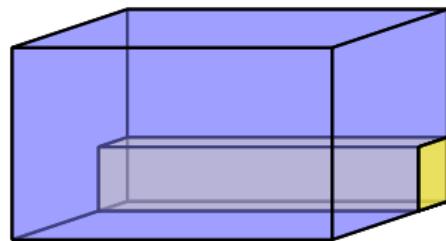


$$\text{output } y_2 = h(w_2^\top * x + b_2)$$

1×1 convolution

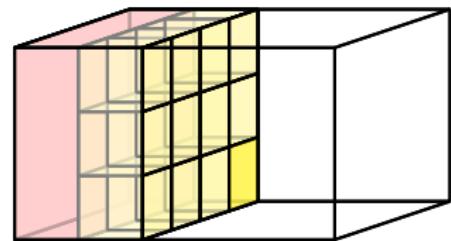


kernel w_2



input \mathbf{x}

new kernel, but still shared
among all spatial positions

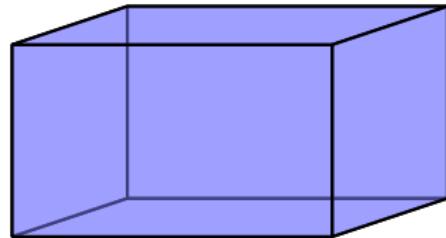


$$\text{output } y_2 = h(\mathbf{w}_2^\top \star \mathbf{x} + b_2)$$

1×1 convolution

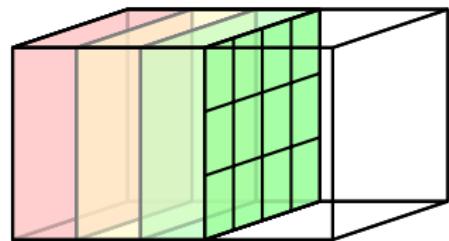


kernel w_3



input x

different kernel for
each output dimension

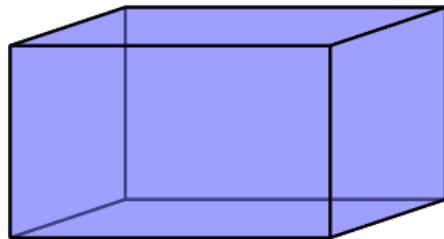


$$\text{output } y_3 = h(\mathbf{w}_3^\top \star \mathbf{x} + b_3)$$

1×1 convolution

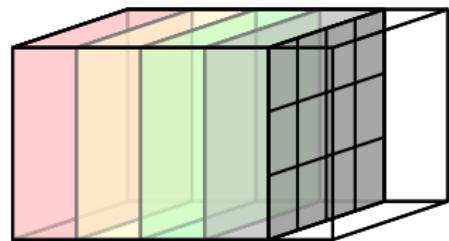


kernel w_4



input x

different kernel for
each output dimension

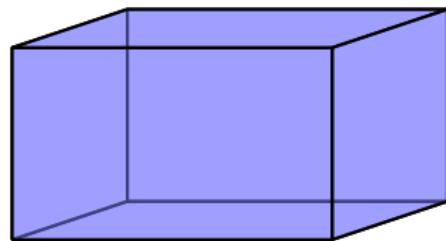


$$\text{output } y_4 = h(\mathbf{w}_4^\top \star \mathbf{x} + b_4)$$

1×1 convolution

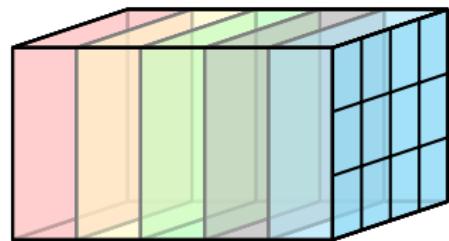


kernel w_5



input x

different kernel for
each output dimension



$$\text{output } y_5 = h(w_5^\top * x + b_5)$$

convolution as regularization

- suppose a fully connected layer is given by

$$\mathbf{a} = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{pmatrix} \mathbf{x}$$

- now if we add the following term to our error function

$$\frac{\lambda}{2} ((w_6 - w_2)^2 + (w_5 - w_1)^2 + w_3^2 + w_4^2)$$

then, as $\lambda \rightarrow \infty$, the weight matrix tends to the constrained **Toeplitz** form

$$\begin{pmatrix} w_1 & w_2 & 0 \\ 0 & w_1 & w_2 \end{pmatrix}$$

and the layer becomes **convolutional**

convolution as regularization

- suppose a fully connected layer is given by

$$\mathbf{a} = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{pmatrix} \mathbf{x}$$

- now if we add the following term to our error function

$$\frac{\lambda}{2} ((w_6 - w_2)^2 + (w_5 - w_1)^2 + w_3^2 + w_4^2)$$

then, as $\lambda \rightarrow \infty$, the weight matrix tends to the constrained **Toeplitz** form

$$\begin{pmatrix} w_1 & w_2 & 0 \\ 0 & w_1 & w_2 \end{pmatrix}$$

and the layer becomes **convolutional**

convolution as Gaussian mixture prior

- remember, weight decay is equivalent to a zero-centered Gaussian prior if the weight vector/matrix is considered a random variable
- in this analogy, error term

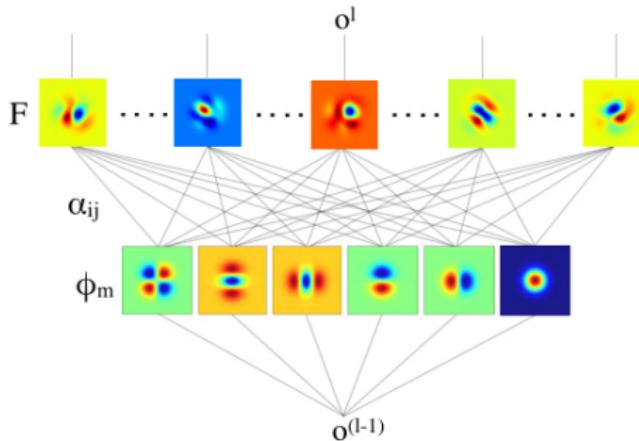
$$\frac{\lambda}{2} ((w_6 - w_2)^2 + (w_5 - w_1)^2 + w_3^2 + w_4^2)$$

corresponds to two Gaussian priors centered at w_1 , w_2 for w_5 , w_6 and one zero-centered Gaussian for w_3 , w_4

- that is, a Gaussian mixture prior

structured convolution

[Jacobsen et al. 2016]



- we can constrain parameters even more by considering a fixed basis of **steerable filters** consisting of **separable Gaussian derivatives**
- the network then only learns the parameters needed to construct a filter as a linear combination of the basis filters
- this applies to all layers

variants and their derivatives

convolution variants

- we will examine a number of variants of convolution, each only in one dimension
- this leaves an extension to one more spatial dimension (convolution), and one more feature dimension (matrix multiplication)
- in each case, we will write convolution as matrix multiplication, where the matrix has some special structure: derivatives are then straightforward

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{|c|c|c|c|c|c|c|} \hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{cccccc} 1 & 2 & 3 & \text{blue} & \text{blue} & \text{blue} \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{ccccc} \text{green} & \text{white} & \text{white} & \text{white} & \text{white} \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{ccccccc} \text{blue} & 1 & \text{red} & 3 & \text{blue} & \text{blue} & \text{blue} \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{ccccc} \text{green} & \text{green} & \text{white} & \text{white} & \text{white} \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{c} \text{blue} \\ \text{blue} \\ \text{purple} \\ | \\ 1 \quad 2 \quad 3 \\ \text{purple} \\ \text{blue} \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{c} \text{green} \\ \text{green} \\ \text{green} \\ \text{white} \\ \text{white} \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{cccccc} \text{blue} & \text{blue} & \text{blue} & 1 & 2 & 3 & \text{blue} \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{cccc} \text{green} & \text{green} & \text{green} & \text{green} & \text{white} \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution

- input size n , kernel size r , output size n'

$$x \quad \begin{array}{ccccccccc} \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{purple} & 1 & 2 & 3 & \text{purple} \end{array} \quad n = 7, r = 3$$

$$a = w \star x \quad \begin{array}{ccccccccc} \text{green} & \text{green} \end{array} \quad n' = n - r + 1 = 5$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

standard convolution: input derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to input \mathbf{x}

$$d\mathbf{x} = W \cdot d\mathbf{a}$$

$$d \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} w_1 & & & & & & \\ w_2 & w_1 & & & & & \\ w_3 & w_2 & w_1 & & & & \\ & w_3 & w_2 & w_1 & & & \\ & & w_3 & w_2 & w_1 & & \\ & & & w_3 & w_2 & w_1 & \\ & & & & w_3 & w_2 & w_1 \end{pmatrix} \cdot d \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

standard convolution: weight derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to weights W

$$dW = \mathbf{x} \cdot d\mathbf{a}^\top$$

$$dW = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \cdot d(a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5)$$

- this is not convenient: we really want $d\mathbf{w} = (dw_1, dw_2, dw_3)$
- if $da_i = \mathbb{1}[i = 4]$, then $d\mathbf{w} = (x_4, x_5, x_6)$: we learn the pattern that generated the activation

standard convolution: weight derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to weights W

$$dW = \mathbf{x} \cdot d\mathbf{a}^\top$$

$$d \begin{pmatrix} w_1 \\ w_2 & w_1 \\ w_3 & w_2 & w_1 \\ & w_3 & w_2 & w_1 \\ & & w_3 & w_2 & w_1 \\ & & & w_3 & w_2 \\ & & & & w_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \cdot d(a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5)$$

- this is not convenient: we really want $d\mathbf{w} = (dw_1, dw_2, dw_3)$
- if $da_i = \mathbb{1}[i=4]$, then $d\mathbf{w} = (x_4, x_5, x_6)$: we learn the pattern that generated the activation

standard convolution: weight derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to weights W

$$dw = da \star x$$

$$d \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = d \begin{pmatrix} a_1 & a_2 & a_3 & a_4 & a_5 \\ & a_1 & a_2 & a_3 & a_4 & a_5 \\ & & a_1 & a_2 & a_3 & a_4 & a_5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

- sharing in forward \equiv adding in backward
- if $da_i = \mathbb{1}[i = 4]$, then $dw = (x_4, x_5, x_6)$: we learn the pattern that generated the activation

standard convolution: weight derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to weights W

$$dw = da \star x$$

$$d \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = d \begin{pmatrix} a_1 & a_2 & a_3 & \textcolor{red}{a}_4 & a_5 \\ & a_1 & a_2 & a_3 & \textcolor{red}{a}_4 & a_5 \\ & & a_1 & a_2 & a_3 & \textcolor{red}{a}_4 & a_5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \textcolor{red}{x}_4 \\ \textcolor{red}{x}_5 \\ x_6 \\ x_7 \end{pmatrix}$$

- sharing in forward \equiv adding in backward
- if $da_i = \mathbb{1}[i = 4]$, then $d\mathbf{w} = (x_4, x_5, x_6)$: we learn the pattern that generated the activation

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{white} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{white} \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{white} \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{ccccccc} 1 & 2 & 3 & & & & \\ \text{red} & \text{red} & \text{purple} & \text{blue} & \text{blue} & \text{blue} & \text{white} \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{cccccc} \text{green} & \text{white} & \text{white} & \text{white} & \text{white} & \text{white} \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \\ & & & & & w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & 1 & 2 & 3 & \text{purple} & \text{purple} & \text{purple} & \text{white} \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \text{green} & \text{green} & \text{white} & \text{white} & \text{white} & \text{white} & \text{white} \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & & & 1 & 2 & 3 & & & \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline & & & \text{green} & & & & \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 & w_3 \\ & & & & & w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{white} & \text{blue} & \text{blue} & \text{1} & \text{2} & \text{3} & \text{blue} & \text{blue} & \text{white} \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|} \hline & \text{green} & \text{green} & \text{green} & \text{green} & \text{white} & \text{white} \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{ } & \text{ } & \text{ } & \text{ } & \text{1} & \text{2} & \text{3} & \text{ } & \text{ } \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline & \text{ } \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{red} & \text{red} & \text{red} & \text{white} \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline & \text{green} & \text{green} & \text{green} & \text{green} & \text{green} & \text{green} & \text{white} \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} w_2 & w_3 \\ w_1 & w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padded convolution

- input size n , kernel size r , padding p , padded input $\mathbf{x}_{(p)} = (\mathbf{0}_p; \mathbf{x}; \mathbf{0}_p)$, output size n'

$$\mathbf{x}_{(p)} \quad \begin{array}{ccccccccc} \text{white} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{purple} & \text{red} & \text{red} \\ \hline \end{array} \quad n = 7, r = 3, p = 1$$

$$a = w \star x_{(p)} \quad \begin{array}{ccccccccc} \text{light green} & \text{bright green} \\ \hline \end{array} \quad n' = (n + 2p) - r + 1 = 7$$

- written as matrix multiplication

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} \textcolor{orange}{w_2} & \textcolor{orange}{w_3} \\ w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \\ & & & w_1 & w_2 & w_3 \\ & & & & w_1 & w_2 \\ & & & & & \textcolor{orange}{w_1} & \textcolor{orange}{w_2} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

padding preserves size

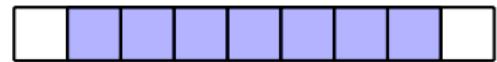
- if kernel size $r = 2\ell + 1$ and $p = \ell$, then $n' = n + 2p - r + 1 = n$ and the size is preserved
- over several layers:

$$p = 0$$

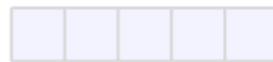
L_1



$$p = 1$$



L_2



L_3

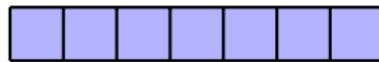


padding preserves size

- if kernel size $r = 2\ell + 1$ and $p = \ell$, then $n' = n + 2p - r + 1 = n$ and the size is preserved
- over several layers:

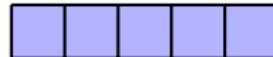
$$p = 0$$

L_1

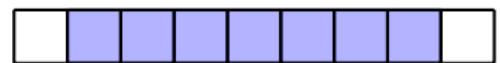
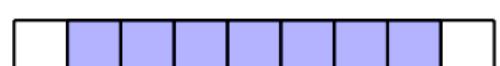


$$p = 1$$

L_2

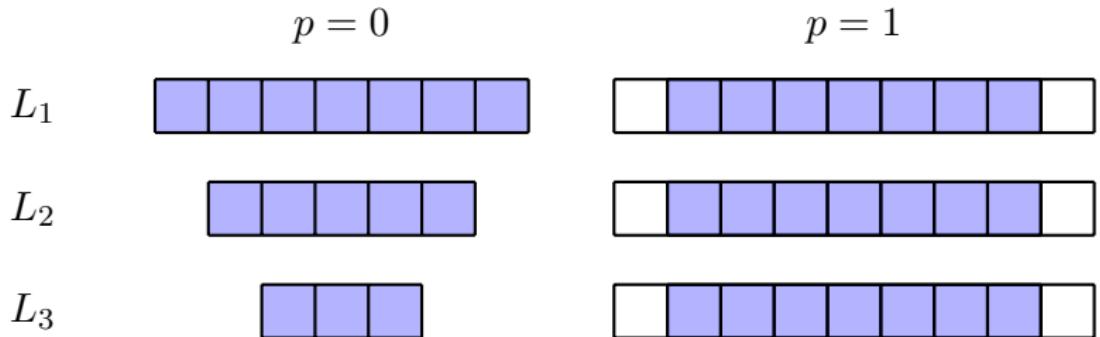


L_3



padding preserves size

- if kernel size $r = 2\ell + 1$ and $p = \ell$, then $n' = n + 2p - r + 1 = n$ and the size is preserved
- over several layers:



strided convolution (down-sampling)

- input size n , kernel size r , stride s , output size n'

$$x \quad \begin{array}{|c|c|c|c|c|c|c|} \hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \end{array} \quad n = 7, r = 3, s = 2$$

$$a = (w \star x) \downarrow_s \quad \begin{array}{|c|c|c|} \hline \textcolor{white}{\square} & \textcolor{white}{\square} & \textcolor{white}{\square} \\ \hline \end{array} \quad n' = \lfloor (n - r)/s \rfloor + 1 = 3$$

- like standard convolution followed by **down-sampling**, but efficient
- written as matrix multiplication (rows sub-sampled)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & & & \\ & w_1 & w_2 & w_3 & & \\ & & w_1 & w_2 & w_3 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

strided convolution (down-sampling)

- input size n , kernel size r , stride s , output size n'

$$x \quad \begin{array}{cccccc} 1 & 2 & 3 & \text{blue} & \text{blue} & \text{blue} \end{array} \quad n = 7, r = 3, s = 2$$

$$a = (w \star x) \downarrow_s \quad \begin{array}{c|c|c} \text{green} & \text{white} & \text{white} \end{array} \quad n' = \lfloor (n - r)/s \rfloor + 1 = 3$$

- like standard convolution followed by **down-sampling**, but efficient
- written as matrix multiplication (rows sub-sampled)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & & & \\ & w_1 & w_2 & w_3 & & \\ & & w_1 & w_2 & w_3 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

strided convolution (down-sampling)

- input size n , kernel size r , stride s , output size n'

$$x \quad \begin{array}{ccccccccc} \text{blue} & \text{blue} & | & 1 & 2 & 3 & | & \text{blue} & \text{blue} \end{array} \quad n = 7, r = 3, s = 2$$

$$a = (w \star x) \downarrow_s \quad \begin{array}{ccc} \text{light green} & \text{green} & | & \text{white} \end{array} \quad n' = \lfloor (n - r)/s \rfloor + 1 = 3$$

- like standard convolution followed by **down-sampling**, but efficient
- written as matrix multiplication (rows sub-sampled)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & & & \\ & w_1 & w_2 & w_3 & & \\ & & w_1 & w_2 & w_3 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

strided convolution (down-sampling)

- input size n , kernel size r , stride s , output size n'

$$x \quad \begin{array}{ccccccccc} & \text{blue} & & \text{blue} & & \text{blue} & & \text{red} & \text{red} \\ & 1 & & 2 & & 3 & & & \end{array} \quad n = 7, r = 3, s = 2$$

$$a = (w \star x) \downarrow_s \quad \begin{array}{ccc} \text{green} & \text{green} & \text{green} \end{array} \quad n' = \lfloor (n - r)/s \rfloor + 1 = 3$$

- like standard convolution followed by **down-sampling**, but efficient
- written as matrix multiplication (rows sub-sampled)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 & & & \\ & w_1 & w_2 & w_3 & & \\ & & w_1 & w_2 & w_3 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

strided convolution: input derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to input \mathbf{x}

$$d\mathbf{x} = W \cdot d\mathbf{a}$$

$$d \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} w_1 & & & & \\ w_2 & w_1 & & & \\ w_3 & w_2 & w_1 & & \\ & w_3 & w_2 & w_1 & \\ & & & & w_3 \end{pmatrix} \cdot d \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

strided convolution: weight derivative

- in general, $C = AB \rightarrow dA = (dC)B^\top, dB = A^\top dC$
- here, $\mathbf{a} = W^\top \mathbf{x}$: derivative with respect to weights W

$$dW = \mathbf{x} \cdot d\mathbf{a}^\top$$

$$d \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = d \begin{pmatrix} a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

- again e.g. by writing W as a function of $\mathbf{w} = (w_1, w_2, w_3)$ and applying the chain rule, or **by just observing the moving pattern**

dilated convolution (up-sampling)

- input size n , kernel size r , dilation factor t , effective kernel size $\hat{r} = r + (r - 1)(t - 1)$, output size n'

$$x \quad \begin{array}{|c|c|c|c|c|c|c|}\hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \end{array} \quad n = 7, r = 3, t = 2$$

$$a = w \uparrow^t \star x \quad \begin{array}{|c|c|c|}\hline \textcolor{blue}{\square} & \textcolor{blue}{\square} & \textcolor{blue}{\square} \\ \hline \end{array} \quad n' = n - \hat{r} + 1 = 3$$

- written as matrix multiplication (like strided backward!)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

dilated convolution (up-sampling)

- input size n , kernel size r , dilation factor t , effective kernel size $\hat{r} = r + (r - 1)(t - 1)$, output size n'

$$x \quad \begin{array}{c} 1 \\ \text{purple} \end{array} \quad \begin{array}{c} 2 \\ \text{red} \end{array} \quad \begin{array}{c} 3 \\ \text{purple} \end{array} \quad n = 7, r = 3, t = 2$$

$$a = w \uparrow^t \star x \quad \begin{array}{c} \text{green} \\ \text{white} \\ \text{white} \end{array} \quad n' = n - \hat{r} + 1 = 3$$

- written as matrix multiplication (like strided backward!)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ w_1 & w_2 & w_3 \\ w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

dilated convolution (up-sampling)

- input size n , kernel size r , dilation factor t , effective kernel size $\hat{r} = r + (r - 1)(t - 1)$, output size n'

$$x \quad \begin{array}{ccccccc} \text{blue} & | & \text{purple} & | & \text{blue} & | & \text{red} & | & \text{blue} & | & \text{purple} & | & \text{blue} \\ 1 & & & & & & 2 & & & & & 3 & & \end{array} \quad n = 7, r = 3, t = 2$$

$$a = w \uparrow^t \star x \quad \begin{array}{ccc} \text{light green} & | & \text{green} & | & \text{white} \end{array} \quad n' = n - \hat{r} + 1 = 3$$

- written as matrix multiplication (like strided backward!)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & & w_2 & & w_3 & & \\ & w_1 & & w_2 & & w_3 & \\ & & w_1 & & w_2 & & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

dilated convolution (up-sampling)

- input size n , kernel size r , dilation factor t , effective kernel size $\hat{r} = r + (r - 1)(t - 1)$, output size n'

$$x \quad \begin{array}{c} \text{blue} \\ \text{blue} \\ 1 \\ \text{blue} \\ \text{red} \\ \text{blue} \\ 3 \\ \text{purple} \end{array} \quad n = 7, r = 3, t = 2$$

$$a = w \uparrow^t \star x \quad \begin{array}{c} \text{light green} \\ \text{light green} \\ \text{green} \end{array} \quad n' = n - \hat{r} + 1 = 3$$

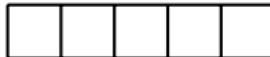
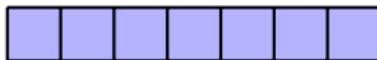
- written as matrix multiplication (like strided backward!)

$$\mathbf{a} = W^\top \cdot \mathbf{x}$$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} w_1 & w_2 & w_3 \\ & w_1 & w_2 & w_3 \\ & & w_1 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

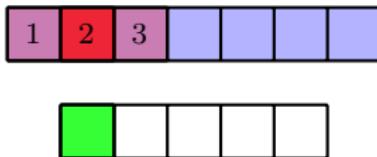


- à trous algorithm: given an input at twice the resolution, apply the same filter dilated by a factor of 2

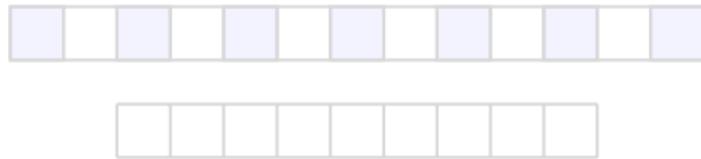


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution



- à trous algorithm: given an input at twice the resolution, apply the same filter dilated by a factor of 2



dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

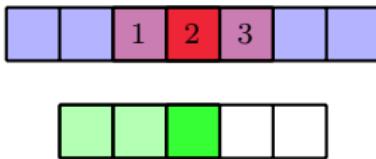


- à trous algorithm: given an input at twice the resolution, apply the same filter dilated by a factor of 2

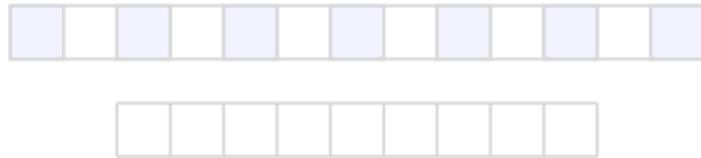


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

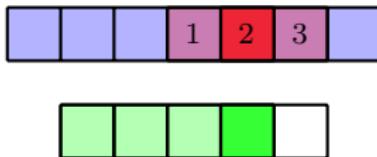


- à trous algorithm: given an input at twice the resolution, apply the same filter dilated by a factor of 2

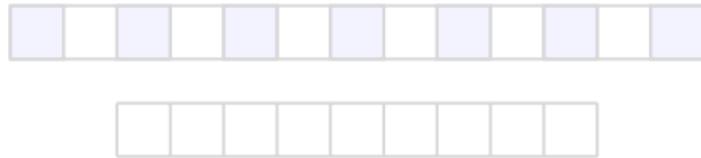


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

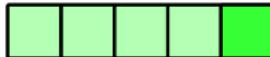
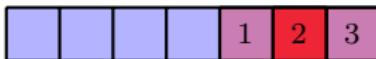


- à trous algorithm: given an input at twice the resolution, apply the same filter dilated by a factor of 2



dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

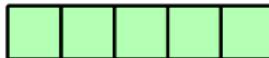
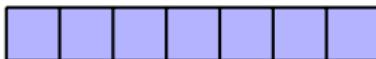


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

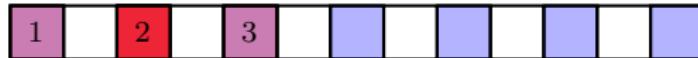


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

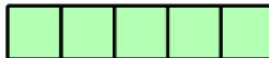
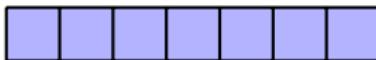


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

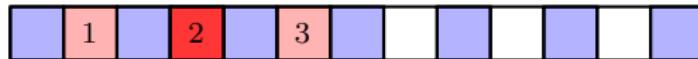


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

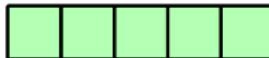
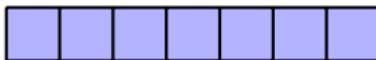


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2



dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

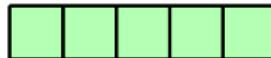
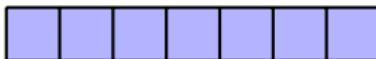


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2



dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

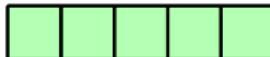
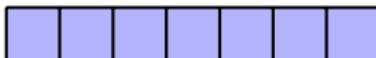


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2



dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

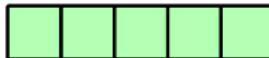
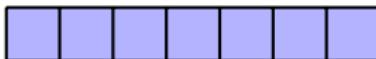


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

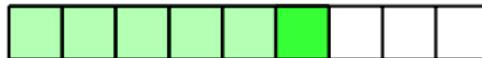
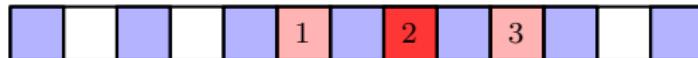


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

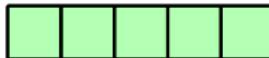
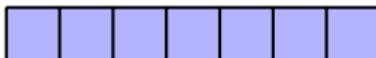


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

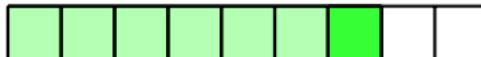
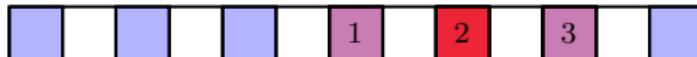


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

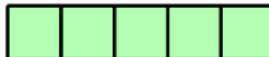
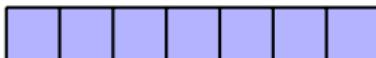


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

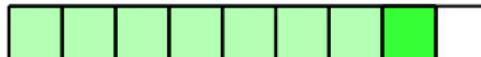


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution

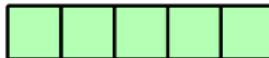
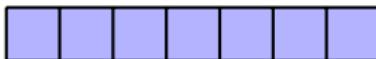


- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2

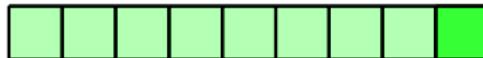


dilated convolution (up-sampling)

- suppose a filter has been trained at a given resolution



- **à trous algorithm:** given an input at twice the resolution, apply the same filter dilated by a factor of 2



convolutional layer arithmetic

- **input volume** $v = w \times h \times k$
- **hyperparameters** k' filters, kernel size r , padding p , stride s , dilation factor t
- effective kernel size $\hat{r} = r + (r - 1)(t - 1)$
- **output volume** $v' = w' \times h' \times k'$ with

$$w' = \lfloor (w + 2p - \hat{r})/s \rfloor + 1$$
$$h' = \lfloor (h + 2p - \hat{r})/s \rfloor + 1$$

- r^2kk' weights, k' biases, $(r^2k + 1)k'$ **parameters** in total
- $(r^2k + 1)v' = (r^2k + 1)k' \times w' \times h'$ **operations** in total

convolutional layer arithmetic

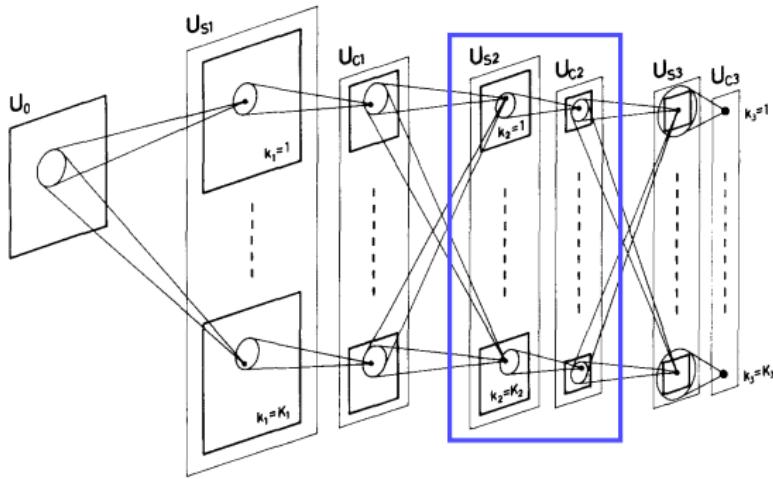
- **input volume** $v = w \times h \times k$
- **hyperparameters** k' filters, kernel size r , padding p , stride s , dilation factor t
- effective kernel size $\hat{r} = r + (r - 1)(t - 1)$
- **output volume** $v' = w' \times h' \times k'$ with

$$w' = \lfloor (w + 2p - \hat{r})/s \rfloor + 1$$
$$h' = \lfloor (h + 2p - \hat{r})/s \rfloor + 1$$

- r^2kk' weights, k' biases, $(r^2k + 1)k'$ **parameters** in total
- $(r^2k + 1)v' = (r^2k + 1)k' \times w' \times h'$ **operations** in total

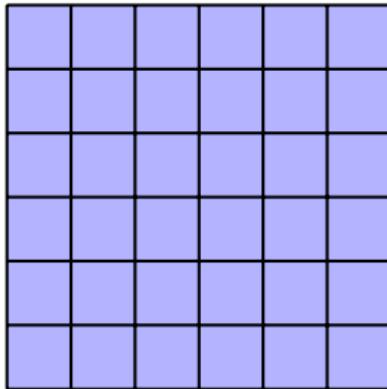
pooling

spatial pooling

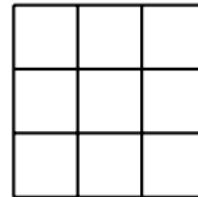


- the deeper a layer is, the larger becomes the **receptive field** of each cell and the **density** of cells decreases accordingly
- gradually introduces translation and deformation **invariance**
- pooling is **independent** per feature map and connections are **fixed**

spatial pooling



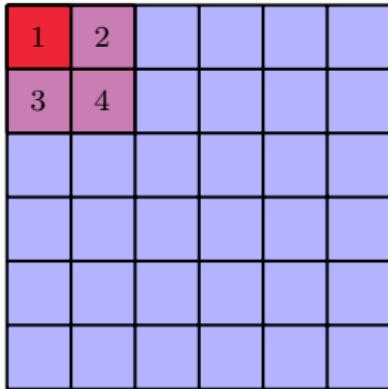
$$n = 6, r = 2, s = 2$$



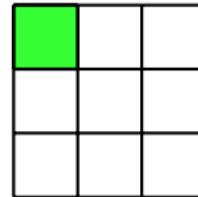
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



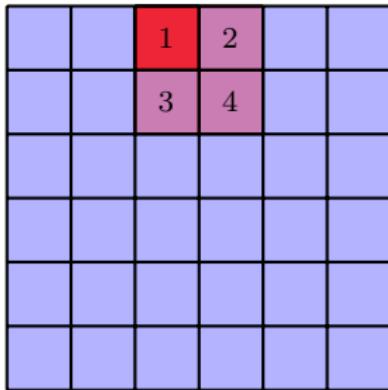
$$n = 6, r = 2, s = 2$$



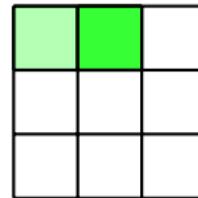
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



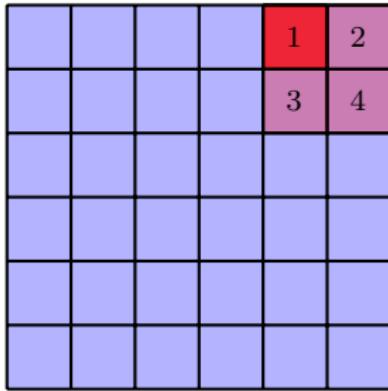
$$n = 6, r = 2, s = 2$$



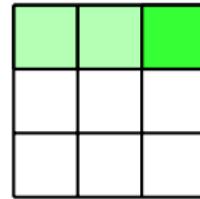
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



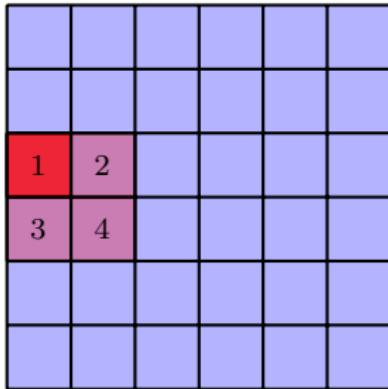
$$n = 6, r = 2, s = 2$$



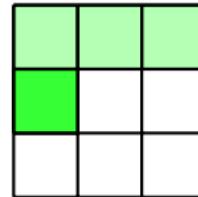
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



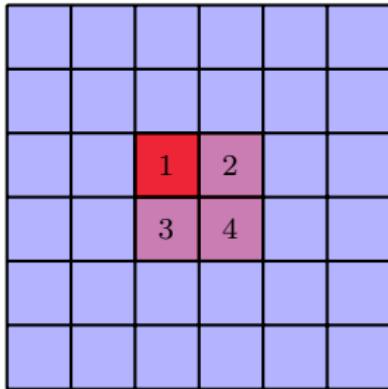
$$n = 6, r = 2, s = 2$$



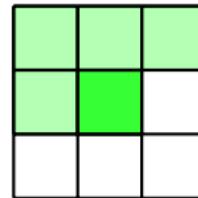
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



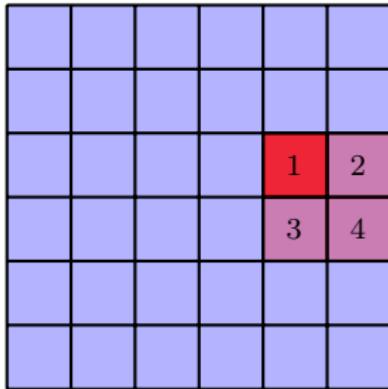
$$n = 6, r = 2, s = 2$$



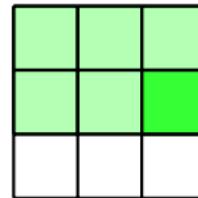
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



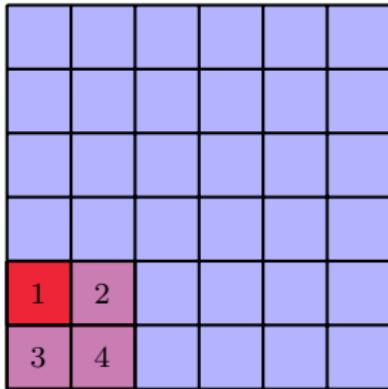
$$n = 6, r = 2, s = 2$$



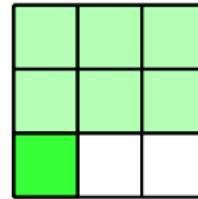
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



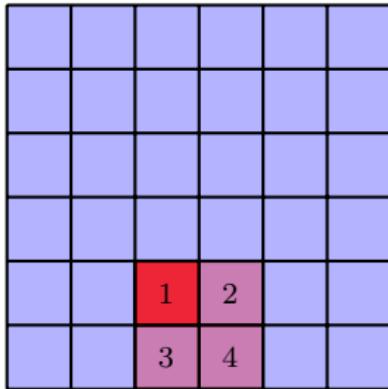
$$n = 6, r = 2, s = 2$$



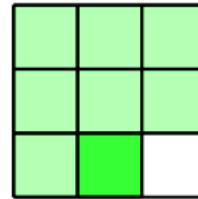
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



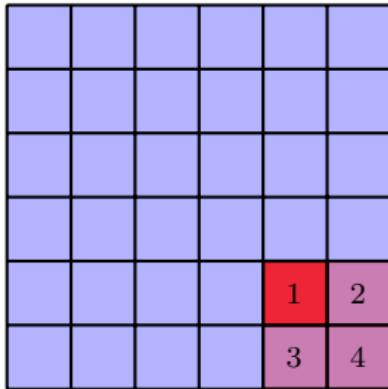
$$n = 6, r = 2, s = 2$$



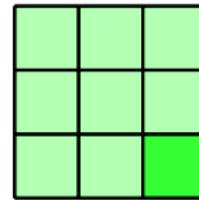
$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

spatial pooling



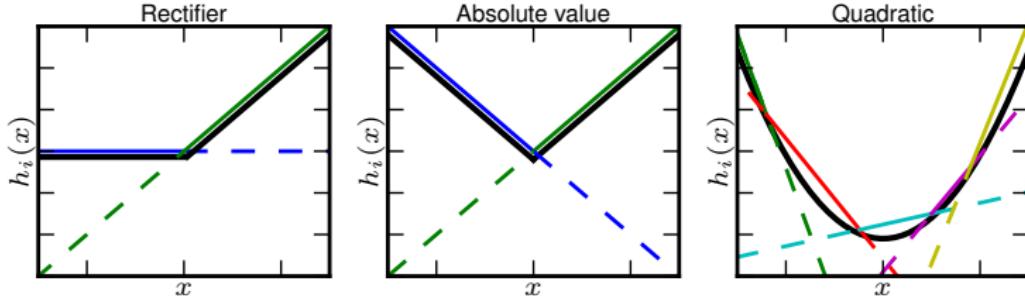
$$n = 6, r = 2, s = 2$$



$$n' = \lfloor n/s \rfloor = 3$$

- same “sliding window” as in convolution, only has **no parameters** and performs orderless pooling rather than dot product per neighborhood, e.g. average or max
- no padding but usually stride $s > 1$
- typically, $r = s$ such that $n' = \lfloor (n - r)/s \rfloor + 1 = \lfloor n/s \rfloor$

feature pooling e.g. maxout

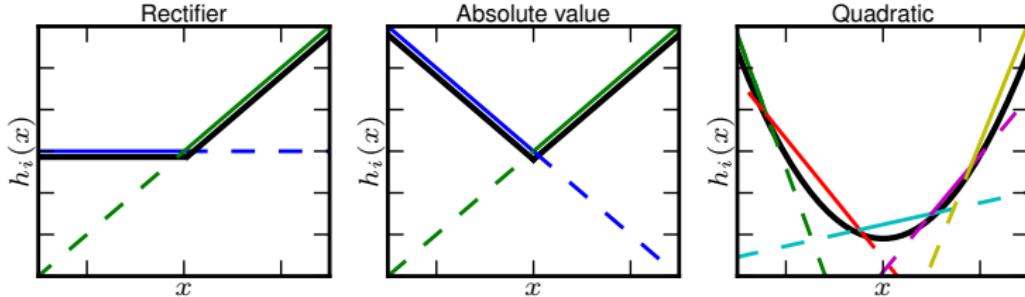


- unlike most activation functions that are element-wise, maxout groups several (e.g. k) activations together and takes their maximum

$$a = \max_j \mathbf{w}_j^\top \mathbf{x} + b_j$$

- does not saturate or “die”, but increases the cost by k
- can approximate any convex function
- two such units can approximate any smooth function!

feature pooling e.g. maxout

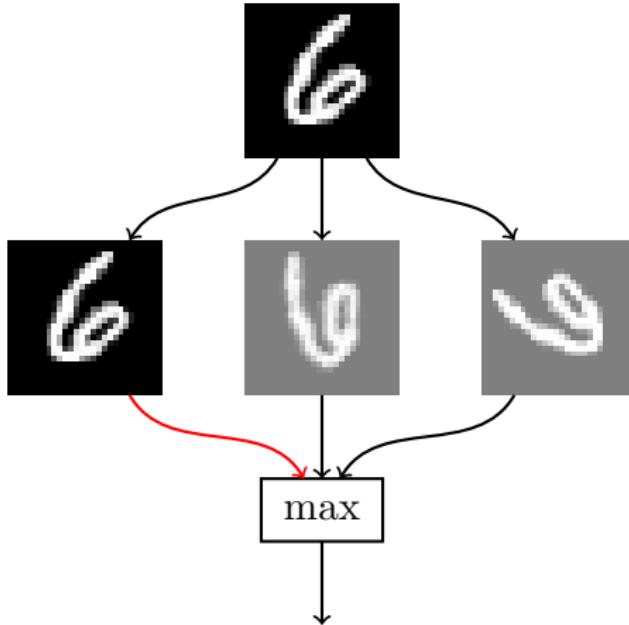


- unlike most activation functions that are element-wise, maxout groups several (e.g. k) activations together and takes their maximum

$$a = \max_j \mathbf{w}_j^\top \mathbf{x} + b_j$$

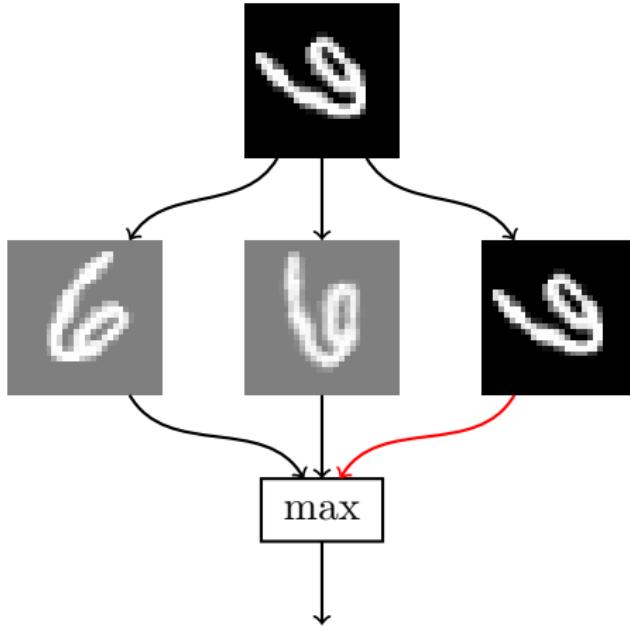
- does not saturate or “die”, but increases the cost by k
- can approximate any convex function
- two such units can approximate any smooth function!

feature pooling: pose invariance



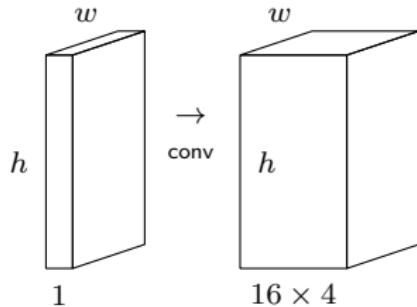
- if each activation responds to a different pose or view, maxout will respond to any

feature pooling: pose invariance



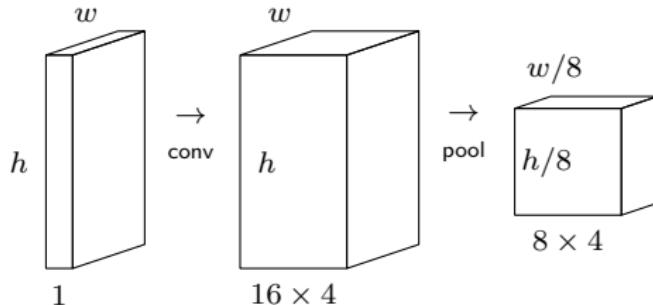
- if each activation responds to a different pose or view, maxout will respond to any

remember HMAX?



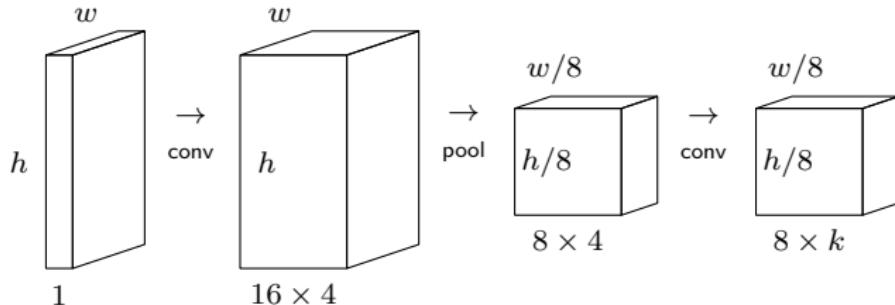
- apply filters at 16 scales \times 4 orientations
- max-pooling over 8×8 spatial cells and over 2 scales
- convolutional RBF matching of input patches X to $k = 4072$ prototypes P (4×4 patches at 4 orientations) extracted at random during learning: activation $Y = \exp(-\gamma \|X - P\|^2)$
- global max pooling over positions and scales
- orientations are not pooled

remember HMAX?



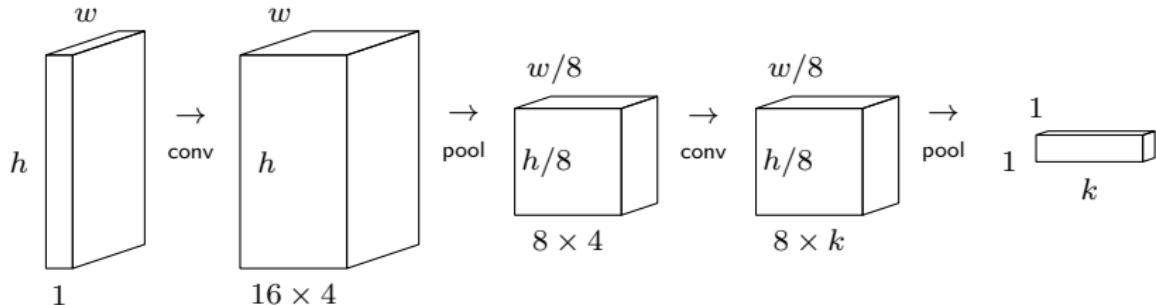
- apply filters at 16 scales \times 4 orientations
- max-pooling over 8×8 spatial cells and over 2 scales
- convolutional RBF matching of input patches X to $k = 4072$ prototypes P (4×4 patches at 4 orientations) extracted at random during learning: activation $Y = \exp(-\gamma \|X - P\|^2)$
- global max pooling over positions and scales
- orientations are not pooled

remember HMAX?



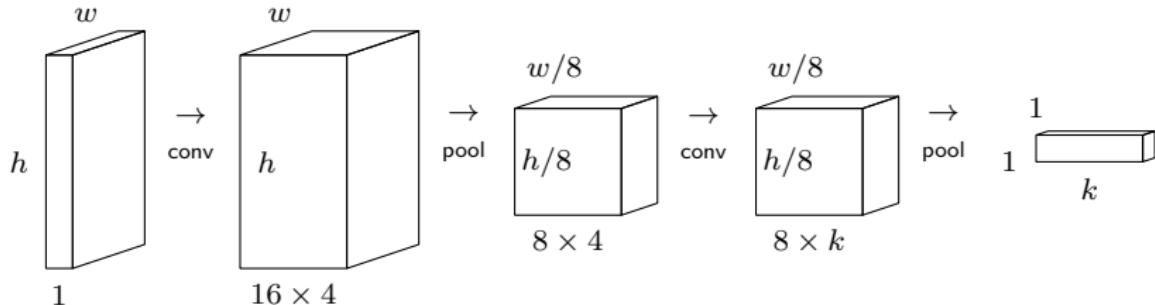
- apply filters at 16 scales \times 4 orientations
- max-pooling over 8×8 spatial cells and over 2 scales
- convolutional RBF matching of input patches X to $k = 4072$ prototypes P (4×4 patches at 4 orientations) extracted at random during learning: activation $Y = \exp(-\gamma \|X - P\|^2)$
- global max pooling over positions and scales
- orientations are not pooled

remember HMAX?



- apply filters at 16 scales \times 4 orientations
- max-pooling over 8×8 spatial cells and over 2 scales
- convolutional RBF matching of input patches X to $k = 4072$ prototypes P (4×4 patches at 4 orientations) extracted at random during learning: activation $Y = \exp(-\gamma \|X - P\|^2)$
- global max pooling over positions and scales
- orientations are not pooled

remember HMAX?



- apply filters at 16 scales \times 4 orientations
- max-pooling over 8×8 spatial cells and over 2 scales
- convolutional RBF matching of input patches X to $k = 4072$ prototypes P (4×4 patches at 4 orientations) extracted at random during learning: activation $Y = \exp(-\gamma \|X - P\|^2)$
- global max pooling over positions and scales
- orientations are not pooled

fun

convolutional network

			MNIST		CIFAR10		
		param	ops	volume	param	ops	volume
x =	input	0	0	$28 \times 28 \times 1$	0	0	$32 \times 32 \times 3$
z ₁ =	conv(5, 32) (x)	832	479232	$24 \times 24 \times 32$	2432	1906688	$28 \times 28 \times 32$
p ₁ =	pool(2) (z ₁)	0	18432	$12 \times 12 \times 32$	0	25088	$14 \times 14 \times 32$
z ₂ =	conv(5, 64) (p ₁)	51264	3280896	$8 \times 8 \times 64$	51264	5126400	$10 \times 10 \times 64$
p ₂ =	pool(2) (z ₂)	0	4096	$4 \times 4 \times 64$	0	6400	$5 \times 5 \times 64$
z ₃ =	fc(100) (p ₂)	102500	102500	100	160100	160100	100
a ₄ =	fc(10) (z ₃)	1010	1010	10	1010	1010	10
y =	softmax (a ₄)	0	0	10	0	0	10

- most **parameters** in first fully connected layer
- most **operations** in second convolutional layer
- most **memory** in first convolutional layer

`conv(r, k'[, p = 0][, s = 1]); (max)- pool(r[, s = r][, p = 0]);`

convolutional network

	MNIST		CIFAR10			
	param	ops	param	ops		
input	0	$28 \times 28 \times 1$	0	$32 \times 32 \times 3$		
conv(5, 32)	832	479232	$24 \times 24 \times 32$	2432	1906688	$28 \times 28 \times 32$
pool(2)	0	18432	$12 \times 12 \times 32$	0	25088	$14 \times 14 \times 32$
conv(5, 64)	51264	3280896	$8 \times 8 \times 64$	51264	5126400	$10 \times 10 \times 64$
pool(2)	0	4096	$4 \times 4 \times 64$	0	6400	$5 \times 5 \times 64$
fc(100)	102500	102500	100	160100	160100	100
fc(10)	1010	1010	10	1010	1010	10
softmax	0	0	10	0	0	10

- most **parameters** in first fully connected layer
- most **operations** in second convolutional layer
- most **memory** in first convolutional layer

$\text{conv}(r, k'[, p = 0][, s = 1]); (\text{max})\text{- pool}(r[, s = r][, p = 0]);$

convolutional network

	MNIST		CIFAR10	
	param	ops	param	ops
		volume		volume
input	0	0	$28 \times 28 \times 1$	$32 \times 32 \times 3$
conv(5, 32)	832	479232	$24 \times 24 \times 32$	$28 \times 28 \times 32$
pool(2)	0	18432	$12 \times 12 \times 32$	$14 \times 14 \times 32$
conv(5, 64)	51264	3280896	$8 \times 8 \times 64$	$10 \times 10 \times 64$
pool(2)	0	4096	$4 \times 4 \times 64$	$5 \times 5 \times 64$
fc(100)	102500	102500	100	160100
fc(10)	1010	1010	10	1010
softmax	0	0	10	10

- most **parameters** in first fully connected layer
- most **operations** in second convolutional layer
- most **memory** in first convolutional layer

$\text{conv}(r, k'[, p = 0][, s = 1]); (\text{max})\text{- pool}(r[, s = r][, p = 0]);$

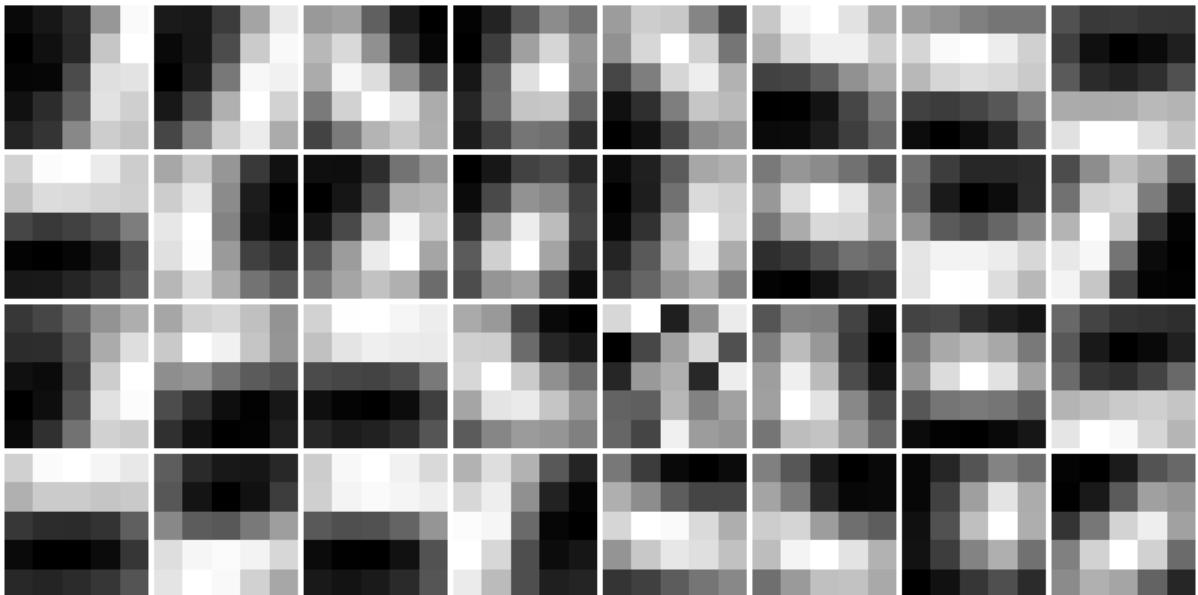
convolutional network

	MNIST		CIFAR10	
	param	ops	param	ops
		volume		volume
input	0	0	$28 \times 28 \times 1$	$32 \times 32 \times 3$
conv(5, 32)	832	479232	$24 \times 24 \times 32$	$28 \times 28 \times 32$
pool(2)	0	18432	$12 \times 12 \times 32$	$14 \times 14 \times 32$
conv(5, 64)	51264	3280896	$8 \times 8 \times 64$	$10 \times 10 \times 64$
pool(2)	0	4096	$4 \times 4 \times 64$	$5 \times 5 \times 64$
fc(100)	102500	102500	100	160100
fc(10)	1010	1010	10	1010
softmax	0	0	10	10

- most **parameters** in first fully connected layer
- most **operations** in second convolutional layer
- most **memory** in first convolutional layer

$\text{conv}(r, k'[, p = 0][, s = 1]); (\text{max})\text{- pool}(r[, s = r][, p = 0]);$

MNIST layer 1 filters



- mini-batch $m = 128$, learning rate $\epsilon = 10^{-2}$, regularization strength $\lambda = 10^{-2}$, Gaussian initialization $\sigma = 0.1$
- test error: 1.2%

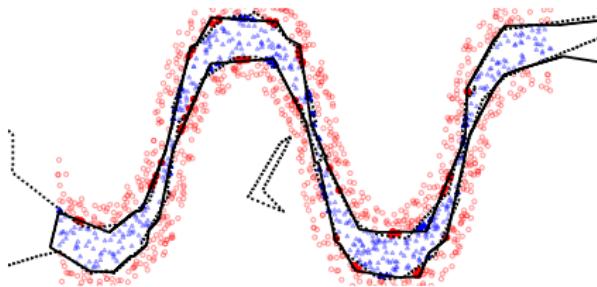
CIFAR10 layer 1 filters



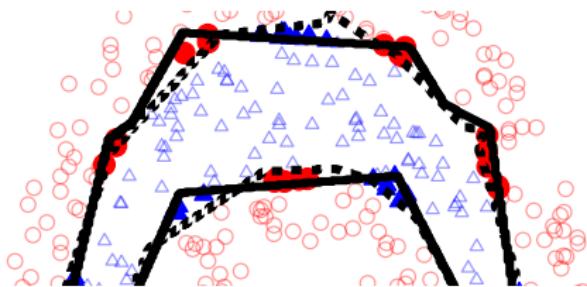
- mini-batch $m = 128$, learning rate $\epsilon = 10^{-2}$, regularization strength $\lambda = 10^{-2}$, Gaussian initialization $\sigma = 0.1$
- test error: 28%

towards deeper networks

[Montufar et al. 2014]



2-layer: solid; 3-layer: dashed
(20 hidden units each)



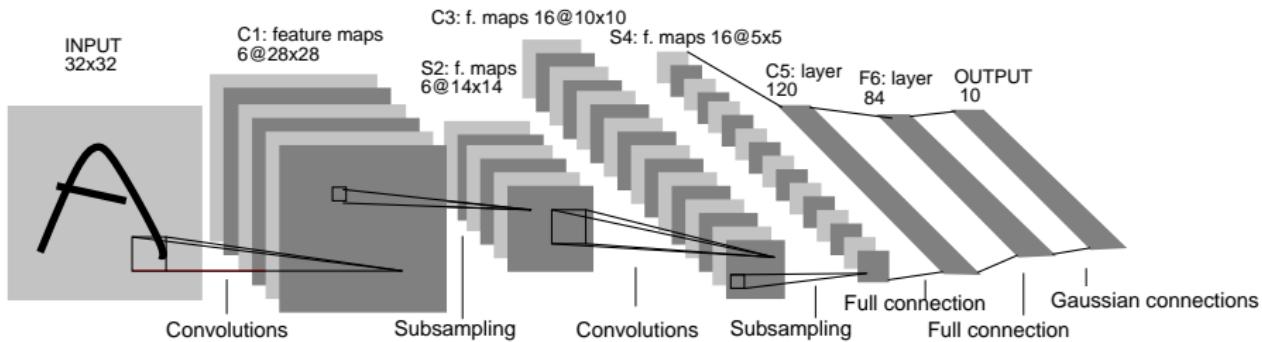
close-up

- “deep networks are able to separate their input space into exponentially more linear response regions than their shallow counterparts, despite using the same number of computational units”

network architectures

LeNet-5

[LeCun et al. 1998]



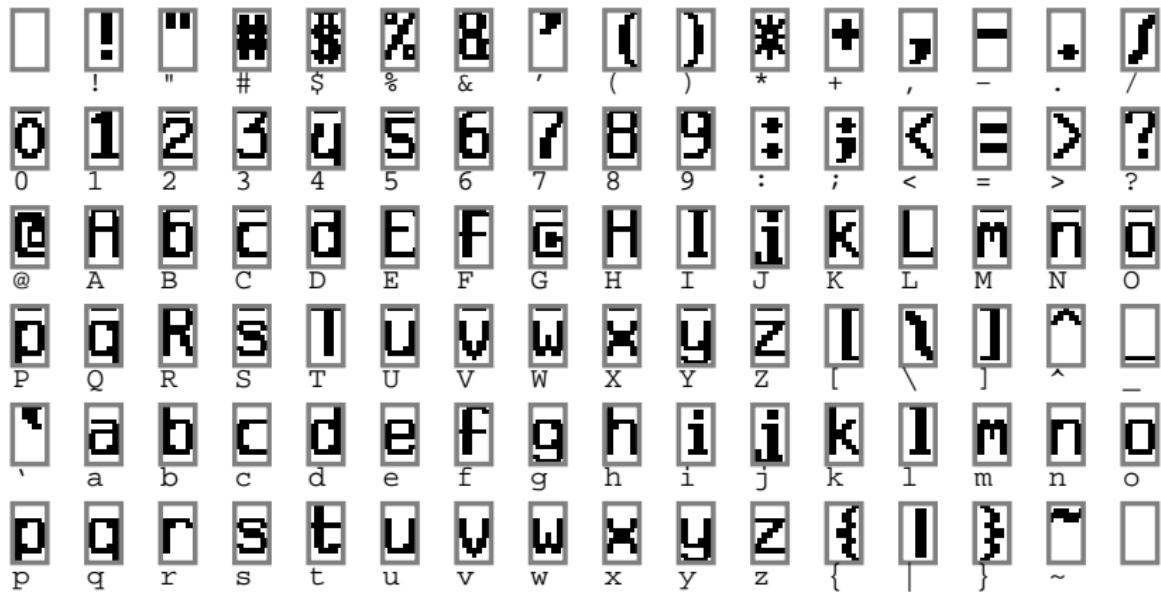
- first convolutional neural network to use back-propagation
- applied to character recognition

LeNet-5

	parameters	operations	volume
input(32, 1)	0	0	$32 \times 32 \times 1$
conv(5, 6)	156	122,304	$28 \times 28 \times 6$
avg(2)	0	4,704	$14 \times 14 \times 6$
conv(5, 16)	2,416	241,600	$10 \times 10 \times 16$
avg(2)	0	1,600	$5 \times 5 \times 16$
conv(5, 120)	48,120	48,120	$1 \times 1 \times 120$
fc(84)	10,164	10,164	84
rbf(10)	850	850	10
softmax	0	10	10

- subsampling by average pooling with learnable global weight and bias
- scaled tanh function after first pooling layer and FC layer
- last convolutional layer allows variable-sized input
- output RBF units: Euclidean distance to 7×12 distributed codes
- softmax-like loss function

LeNet-5 distributed codes



- 7 × 12 character bitmaps
- chosen by hand to initialize the FC-RBF connections
- structured output

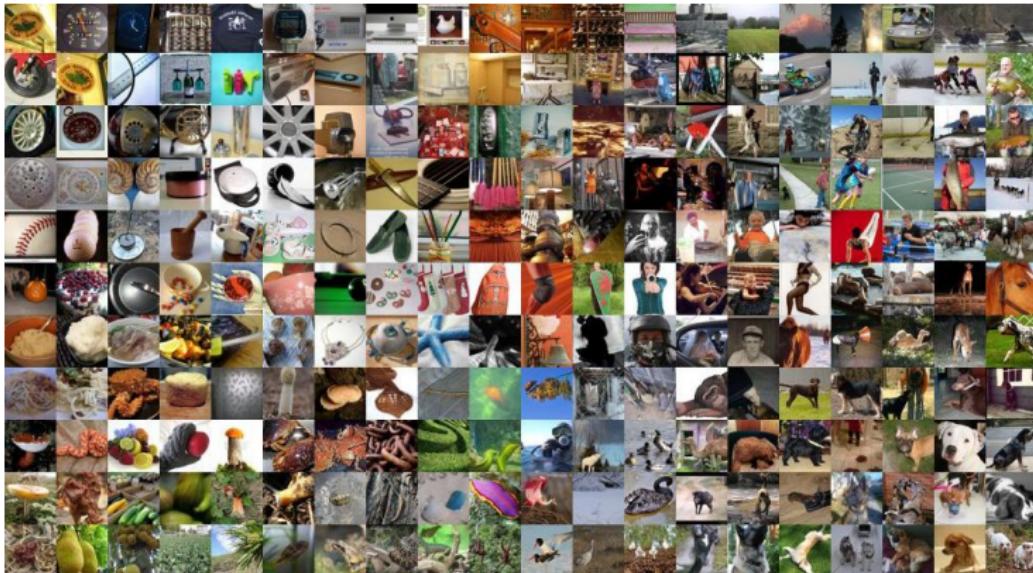
LeNet-5 connections between convolutional layers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X			X	X	X			X	X	X	X		X	X	
1	X	X			X	X	X			X	X	X	X		X	
2	X	X	X			X	X	X			X		X	X	X	
3		X	X	X		X	X	X	X			X		X	X	
4			X	X	X		X	X	X	X		X	X		X	
5				X	X	X			X	X	X	X		X	X	

- number of connections limited
- forces break of symmetry

ImageNet

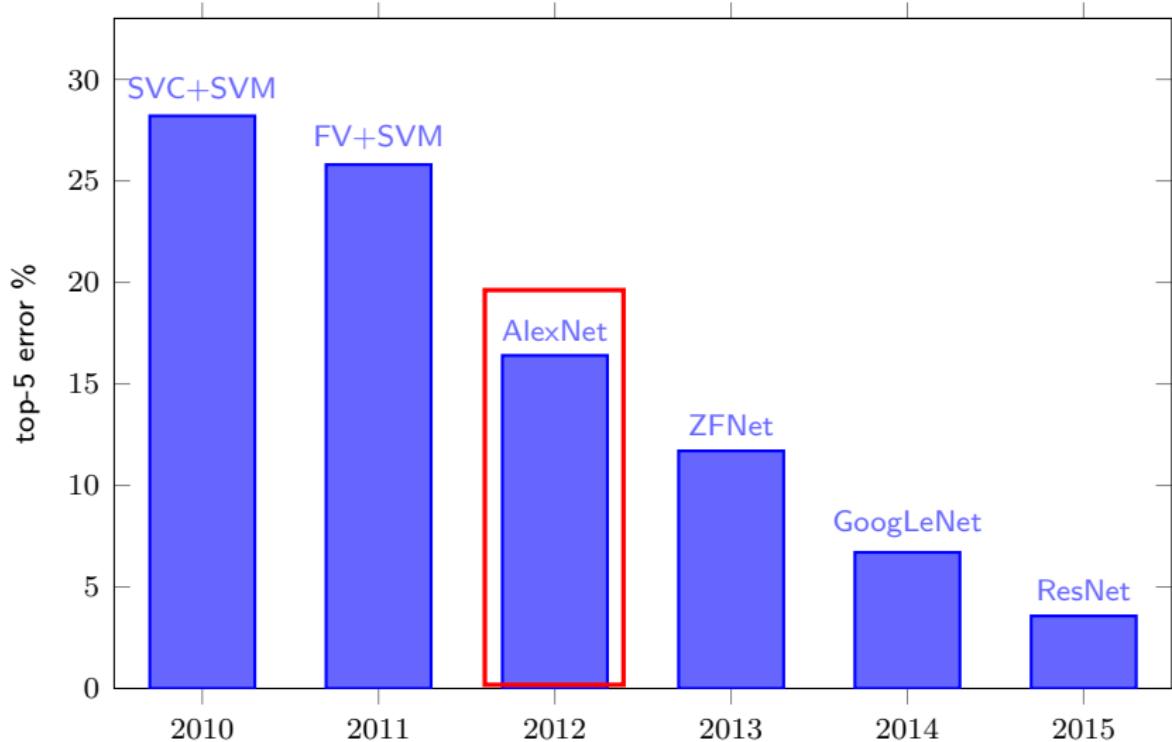
[Russakovsky et al. 2014]



- 22k classes, 15M samples
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1000 classes, 1.2M training images, 50k validation images, 150k test images

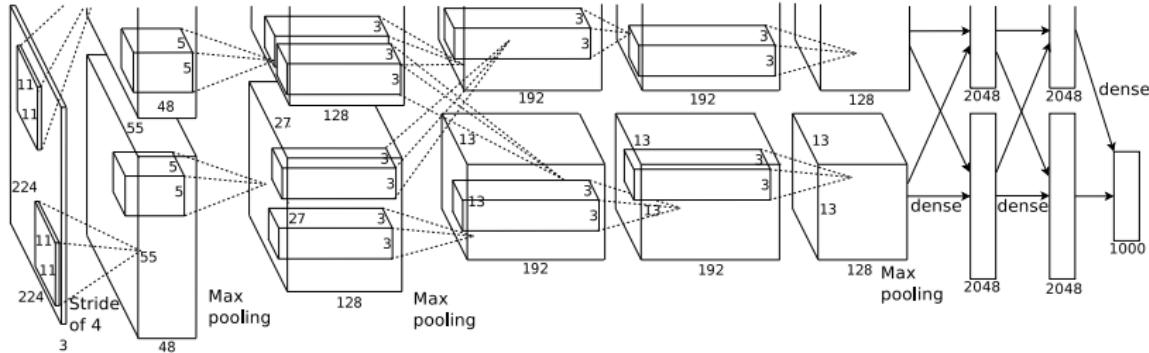
Russakovsky, Deng, Su, Krause, et al. 2014. Imagenet Large Scale Visual Recognition Challenge.

ImageNet classification performance



AlexNet

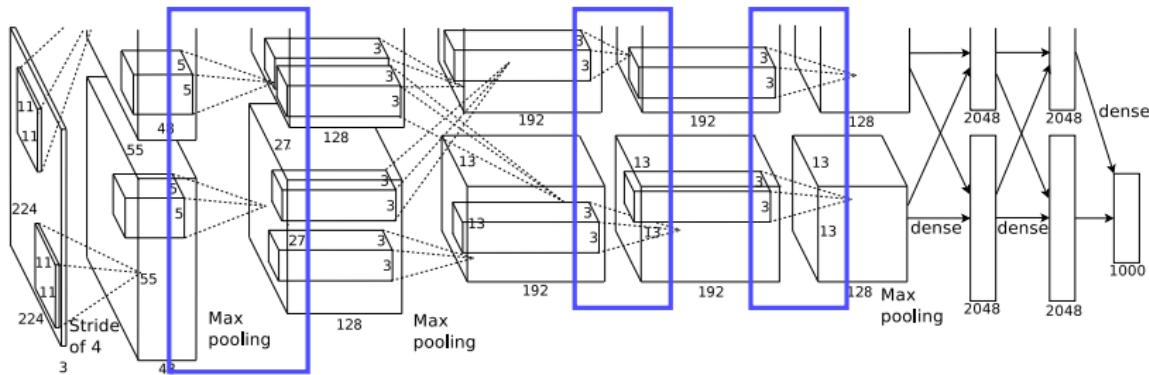
[Krizhevsky et al. 2012]



- 16.4% top-5 error on ILSVRC'12, outperformed all by 10%
- 8 layers
- ReLU, local response normalization, **data augmentation**, **dropout**
- stochastic gradient descent with **momentum**
- implementation on two GPUs; connectivity between the two subnetworks is limited

AlexNet

[Krizhevsky et al. 2012]



- 16.4% top-5 error on ILSVRC'12, outperformed all by 10%
- 8 layers
- ReLU, local response normalization, **data augmentation**, **dropout**
- stochastic gradient descent with **momentum**
- implementation on two GPUs; connectivity between the two subnetworks is limited

learned layer 1 kernels



- 96 kernels of size $11 \times 11 \times 3$
- top: 48 GPU 1 kernels; bottom: 48 GPU 2 kernels

AlexNet (CaffeNet)

	parameters	operations	volume
input(227, 3)	0	0	$227 \times 227 \times 3$
conv(11, 96, s4)	34,944	105,705,600	$55 \times 55 \times 96$
pool(3, 2)	0	290,400	$27 \times 27 \times 96$
norm	0	69,984	$27 \times 27 \times 96$
conv(5, 256, p2)	614,656	448,084,224	$27 \times 27 \times 256$
pool(3, 2)	0	186,624	$13 \times 13 \times 256$
norm	0	43,264	$13 \times 13 \times 256$
conv(3, 384, p1)	885,120	149,585,280	$13 \times 13 \times 384$
conv(3, 384, p1)	1,327,488	224,345,472	$13 \times 13 \times 384$
conv(3, 256, p1)	884,992	149,563,648	$13 \times 13 \times 256$
pool(3, 2)	0	43,264	$6 \times 6 \times 256$
fc(4096)	37,752,832	37,752,832	4,096
fc(4096)	16,781,312	16,781,312	4,096
fc(1000)	4,097,000	4,097,000	1,000
softmax	0	1,000	1,000

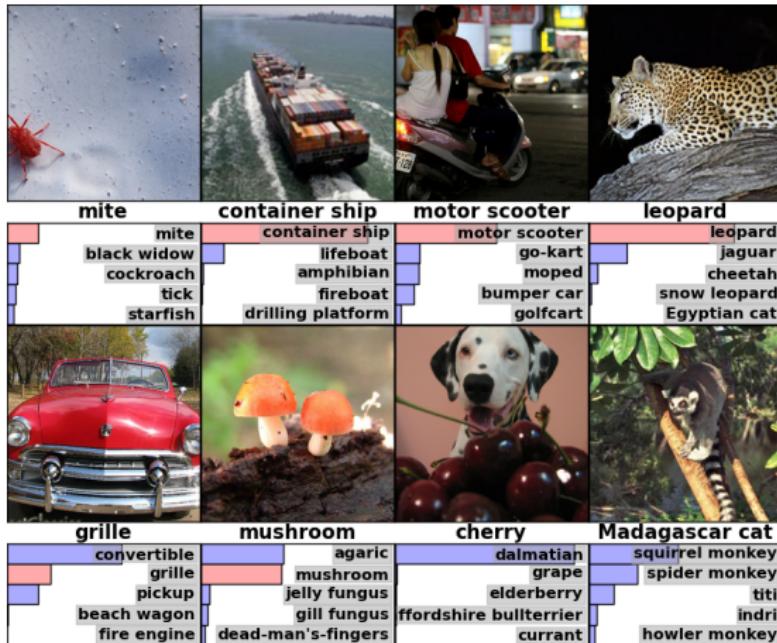
- ReLU follows each convolutional and fully connected layer
- CaffeNet: input size modified from 224×224 , pool/norm switched

AlexNet (CaffeNet)

	parameters	operations	volume
input(227, 3)	0	0	$227 \times 227 \times 3$
conv(11, 96, s4)	34,944	105,705,600	$55 \times 55 \times 96$
pool(3, 2)	0	290,400	$27 \times 27 \times 96$
norm	0	69,984	$27 \times 27 \times 96$
conv(5, 256, p2)	614,656	448,084,224	$27 \times 27 \times 256$
pool(3, 2)	0	186,624	$13 \times 13 \times 256$
norm	0	43,264	$13 \times 13 \times 256$
conv(3, 384, p1)	885,120	149,585,280	$13 \times 13 \times 384$
conv(3, 384, p1)	1,327,488	224,345,472	$13 \times 13 \times 384$
conv(3, 256, p1)	884,992	149,563,648	$13 \times 13 \times 256$
pool(3, 2)	0	43,264	$6 \times 6 \times 256$
fc(4096)	37,752,832	37,752,832	4,096
fc(4096)	16,781,312	16,781,312	4,096
fc(1000)	4,097,000	4,097,000	1,000
softmax	0	1,000	1,000

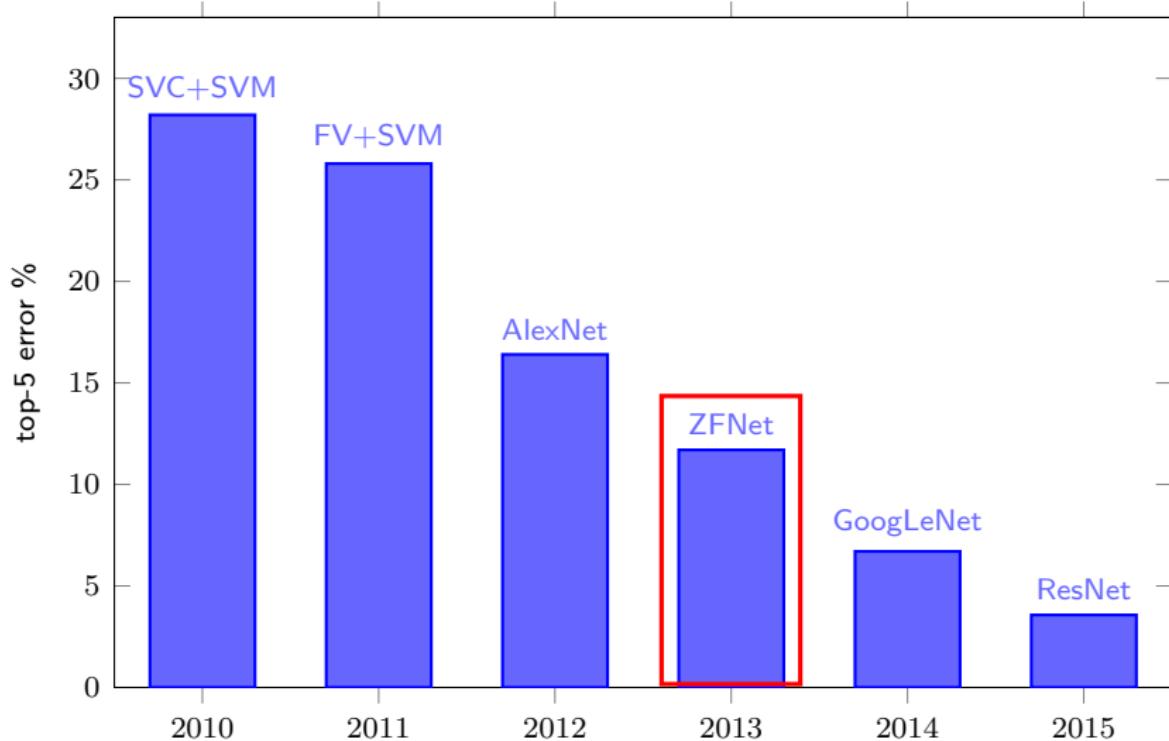
- ReLU follows each convolutional and fully connected layer
- CaffeNet: input size modified from 224×224 , pool/norm switched

AlexNet: classification examples

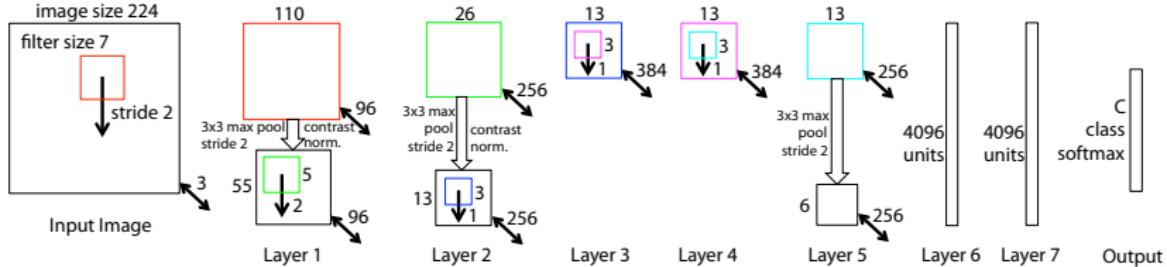


- correct label on top; its predicted probability with red if visible

ImageNet classification performance



ZFNet



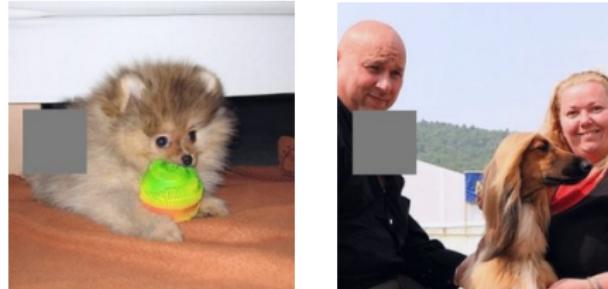
- 11.7% top-5 error on ILSVRC'13
- 8 layers, refinement of AlexNet
- layer 1 kernel size (stride) reduced from 11(4) to 7(2) to reduce aliasing artifacts
- conv3,4,5 width increased to 512, 1024, 512

ZFNet

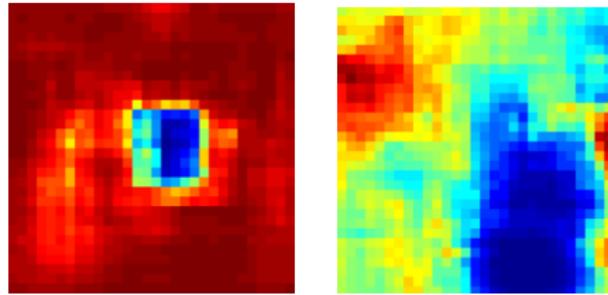
	parameters	operations	volume
input(224, 3)	0	0	$224 \times 224 \times 3$
conv(7, 96, s2, p1)	14,208	171,916,800	$110 \times 110 \times 96$
pool(3, 2, p1)	0	1,161,600	$55 \times 55 \times 96$
norm	0	290,400	$55 \times 55 \times 96$
conv(5, 256, s2)	614,656	415,507,456	$26 \times 26 \times 256$
pool(3, 2, p1)	0	173,056	$13 \times 13 \times 256$
norm	0	43,264	$13 \times 13 \times 256$
conv(3, 512, p1)	1,180,160	199,447,040	$13 \times 13 \times 512$
conv(3, 1024, p1)	4,719,616	797,615,104	$13 \times 13 \times 1024$
conv(3, 512, p1)	4,719,104	797,528,576	$13 \times 13 \times 512$
pool(3, 2)	0	86,528	$6 \times 6 \times 512$
fc(4096)	75,501,568	75,501,568	4,096
fc(4096)	16,781,312	16,781,312	4,096
fc(1000)	4,097,000	4,097,000	1,000
softmax	0	1,000	1,000

- layer widths adjusted by cross-validation; **depth matters**

ZFNet: occlusion sensitivity



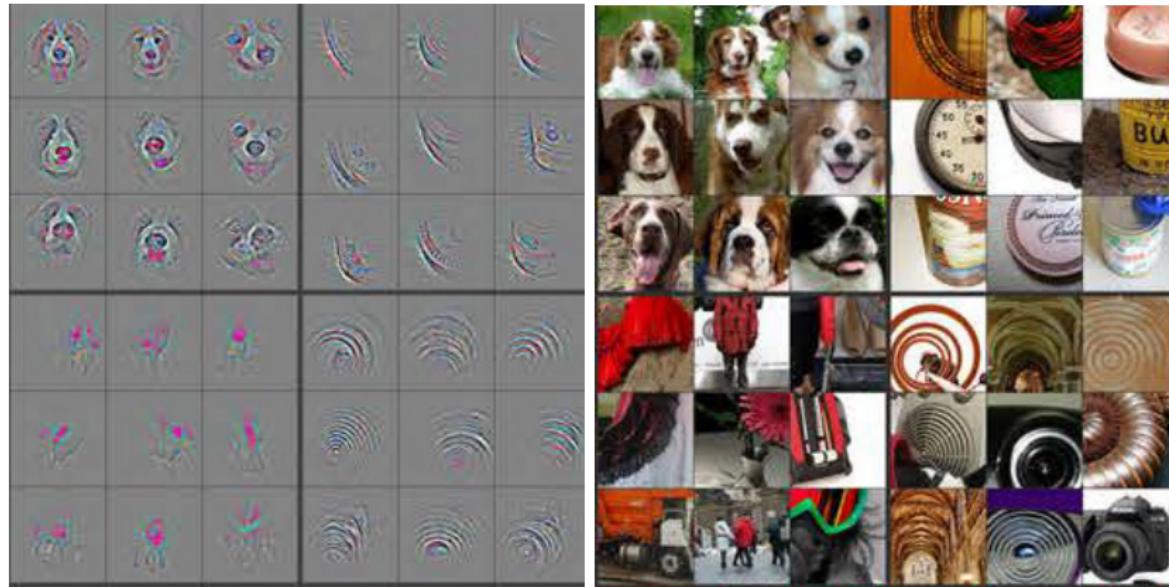
image



correct class probability

- image occluded by gray square
- correct class probability as a function of the position of the square

ZFNet: visualizing intermediate layers



- reconstructed patterns from top 9 activations of selected features of layer 4 and corresponding image patches

VGG

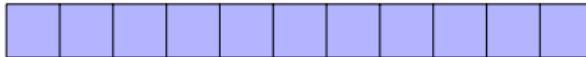
[Simonyan and Zisserman 2014]

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
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					

- 7.3% top-5 error on ILSVRC'14
- depth increased up to 19 layers, kernel sizes (strides) reduced to 3(1)
- local response normalization doesn't do anything
- top/bottom layers of deep models pre-initialized by trained model A

effective receptive field

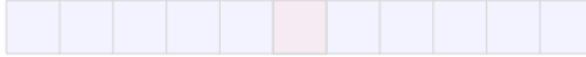
L_0



L_1



L_2

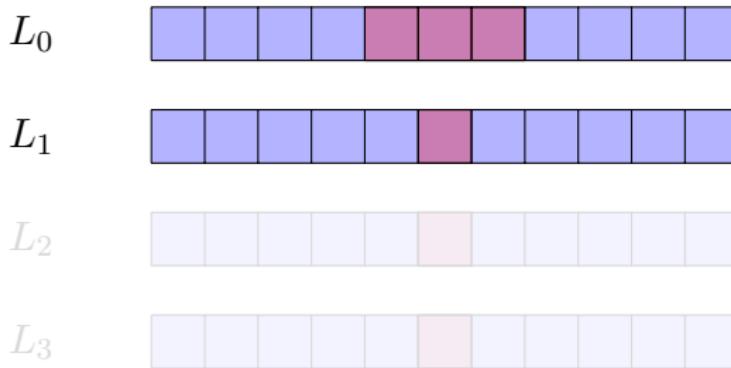


L_3



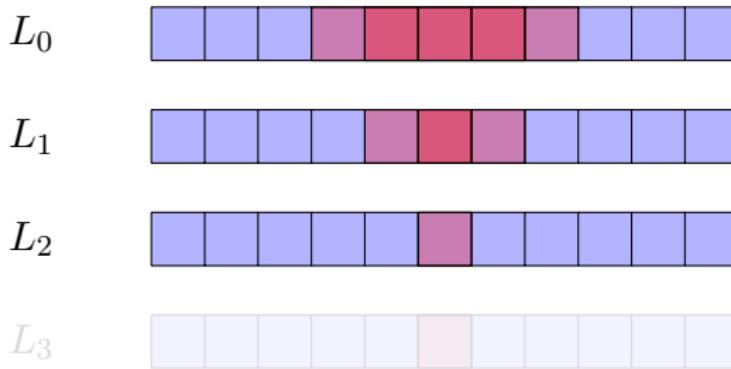
- is the part of the visual input that affects a given cell indirectly through previous layers
- grows linearly with depth
- stack of three 3×3 kernels of stride 1 has the same effective receptive field as a single 7×7 kernel, but fewer parameters

effective receptive field



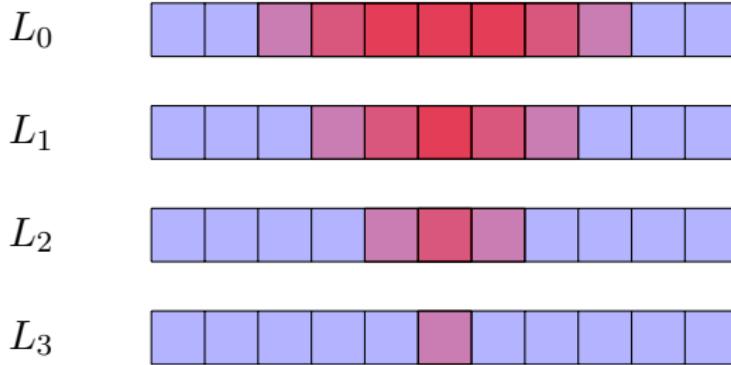
- is the part of the visual input that affects a given cell indirectly through previous layers
- grows linearly with depth
- stack of three 3×3 kernels of stride 1 has the same effective receptive field as a single 7×7 kernel, but fewer parameters

effective receptive field



- is the part of the visual input that affects a given cell indirectly through previous layers
- grows linearly with depth
- stack of three 3×3 kernels of stride 1 has the same effective receptive field as a single 7×7 kernel, but fewer parameters

effective receptive field



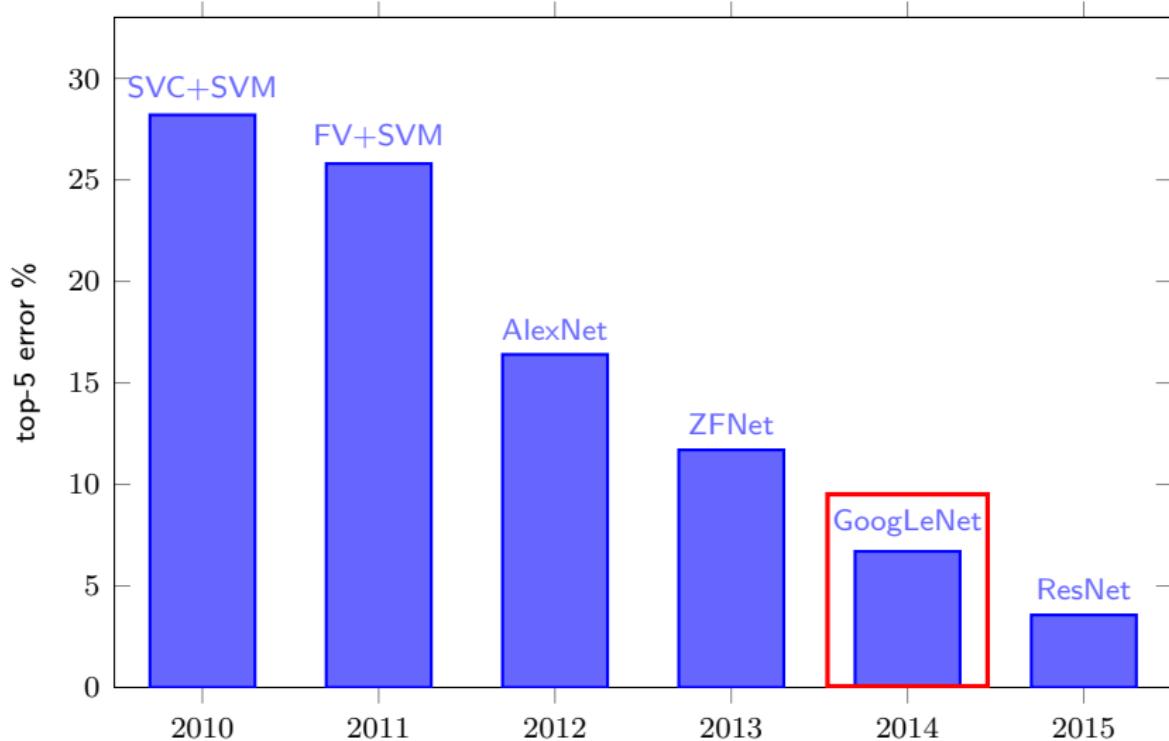
- is the part of the visual input that affects a given cell indirectly through previous layers
- grows linearly with depth
- stack of three 3×3 kernels of stride 1 has the same effective receptive field as a single 7×7 kernel, but fewer parameters

VGG-16

	parameters	operations	volume
input(224, 3)	0	0	$224 \times 224 \times 3$
conv(3, 64, p1)	1,792	89,915,390	$224 \times 224 \times 64$
conv(3, 64, p1)	36,928	1,852,899,328	$224 \times 224 \times 64$
pool(2)	0	3,211,264	$112 \times 112 \times 64$
conv(3, 128, p1)	73,856	926,449,664	$112 \times 112 \times 128$
conv(3, 128, p1)	147,584	1,851,293,696	$112 \times 112 \times 128$
pool(2)	0	1,605,632	$56 \times 56 \times 128$
conv(3, 256, p1)	295,168	925,646,848	$56 \times 56 \times 256$
conv(3, 256, p1)	590,080	1,850,490,880	$56 \times 56 \times 256$
conv(3, 256, p1)	590,080	1,850,490,880	$56 \times 56 \times 256$
pool(2)	0	802,816	$28 \times 28 \times 256$
conv(3, 512, p1)	1,180,160	925,245,440	$28 \times 28 \times 512$
conv(3, 512, p1)	2,359,808	1,850,089,472	$28 \times 28 \times 512$
conv(3, 512, p1)	2,359,808	1,850,089,472	$28 \times 28 \times 512$
pool(2)	0	401,408	$14 \times 14 \times 512$
conv(3, 512, p1)	2,359,808	462,522,368	$14 \times 14 \times 512$
conv(3, 512, p1)	2,359,808	462,522,368	$14 \times 14 \times 512$
conv(3, 512, p1)	2,359,808	462,522,368	$14 \times 14 \times 512$
pool(2)	0	100,352	$7 \times 7 \times 512$
fc(4096)	102,764,544	102,764,544	4,096
fc(4096)	16,781,312	16,781,312	4,096
fc(1000)	4,097,000	4,097,000	1,000
softmax	0	1,000	1,000

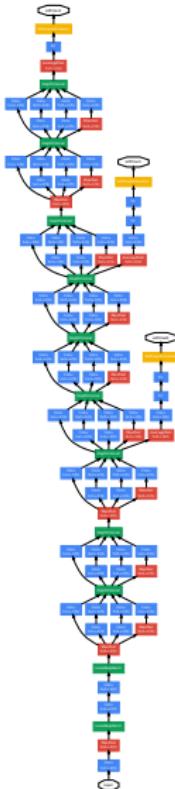


ImageNet classification performance



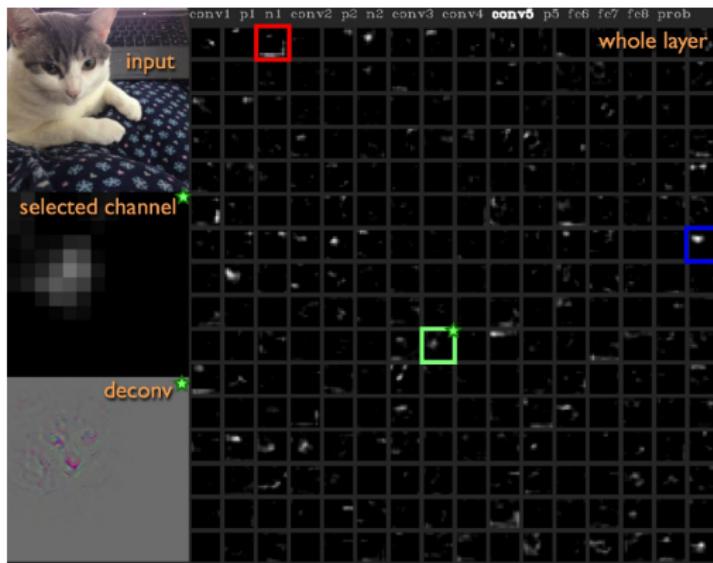
GoogLeNet

[Szegedy et al. 2015]



- 6.7% top-5 error on ILSVRC'14
- depth increased to 22 layers, kernel sizes 1×1 to 5×5
- inception module repeated 9 times
- 1×1 kernels used as “bottleneck” layers (dimensionality reduction)
- 25 times less parameters and faster than AlexNet
- auxiliary classifiers

convolutional features are sparse



- remember, features play the role of codebooks, and bag-of-words representations can be sparse even **after pooling**
- with relu, each feature represents a “**detector**” that fires when the activation is positive

convolutional features are sparse

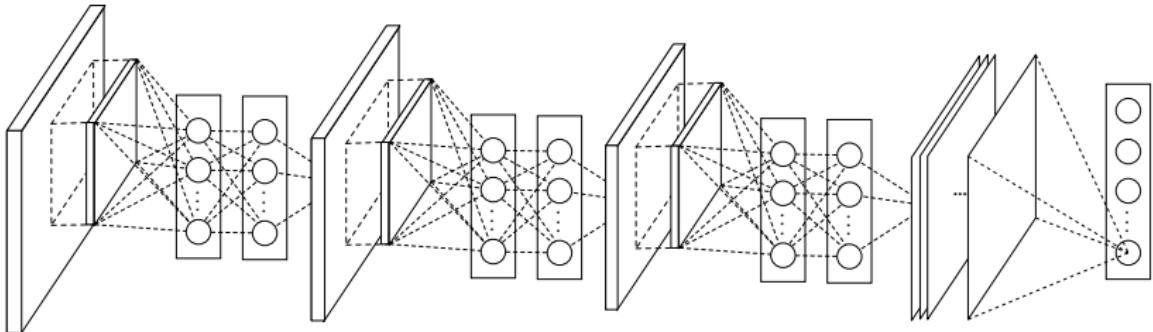
- deep layers have more features (e.g. 1024) and lower resolutions (e.g. 7×7)
 - detected patterns in many cases are as small as 5×5 , 3×3 or even 1×1
 - the convolution operation resembles more (sparse) matrix multiplication than convolution
- sparse matrix multiplication
 - is not as efficient as dense on parallel hardware
 - modern methods use coarse **partitioning** of nonzero elements
 - this is aligned with the Hebbian rule “**cells that fire together wire together**”

convolutional features are sparse

- deep layers have more features (e.g. 1024) and lower resolutions (e.g. 7×7)
 - detected patterns in many cases are as small as 5×5 , 3×3 or even 1×1
 - the convolution operation resembles more (sparse) matrix multiplication than convolution
- sparse matrix multiplication
 - is not as efficient as dense on parallel hardware
 - modern methods use coarse **partitioning** of nonzero elements
 - this is aligned with the Hebbian rule “**cells that fire together wire together**”

network in network (NiN)

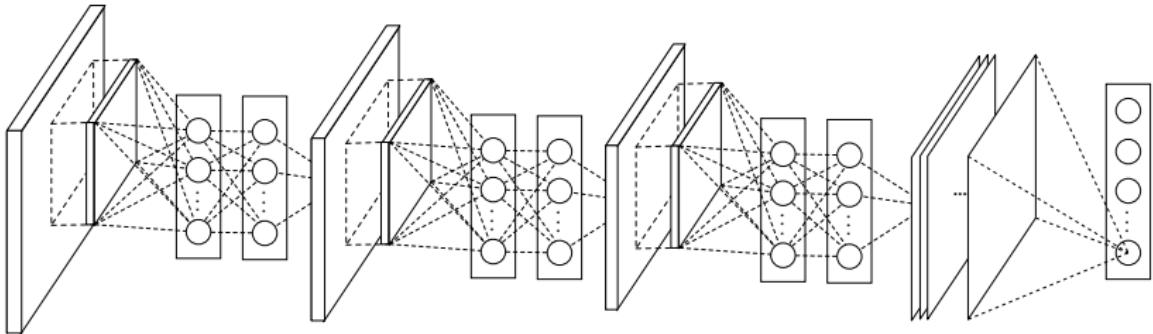
[Lin et al. 2013]



- activation functions are usually element-wise for simplicity; but here an entire **2-layer network** is used as activation function
- fully connected layers are simply replaced by **global average pooling**
- but this is nothing but convolution followed by two 1×1 convolutions
- 1×1 convolutions are just like matrix multiplications and can be used for **dimension reduction**

network in network (NiN)

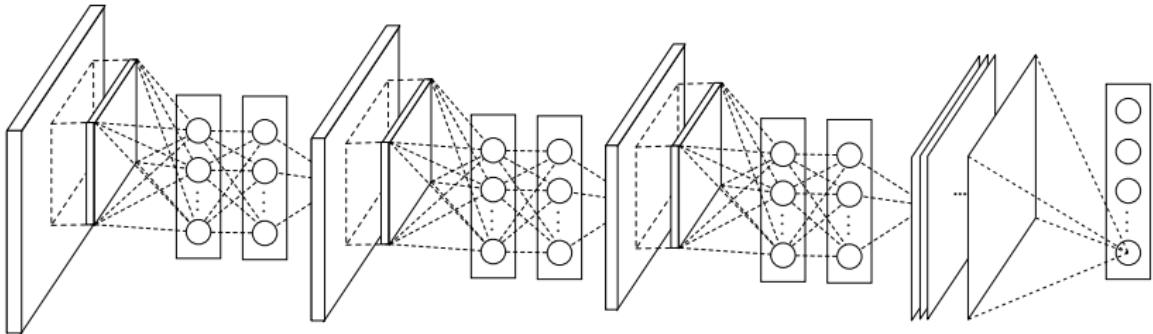
[Lin et al. 2013]



- activation functions are usually element-wise for simplicity; but here an entire **2-layer network** is used as activation function
- fully connected layers are simply replaced by **global average pooling**
- but this is nothing but convolution followed by two **1×1 convolutions**
- 1×1 convolutions are just like matrix multiplications and can be used for **dimension reduction**

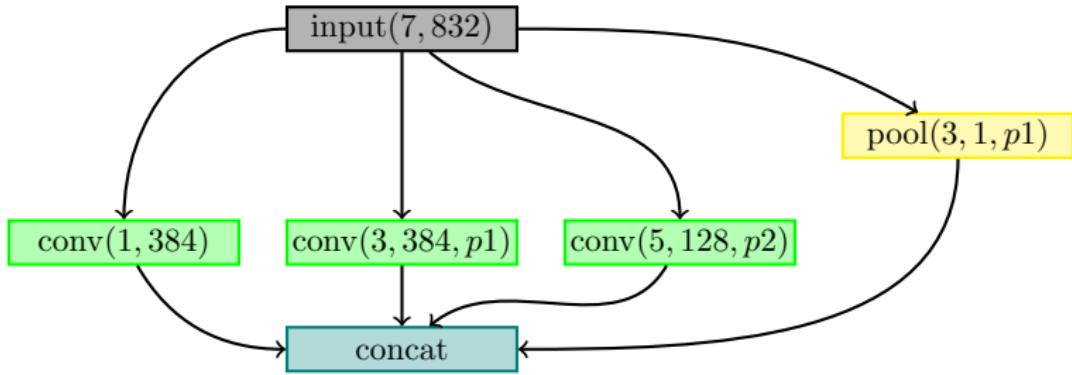
network in network (NiN)

[Lin et al. 2013]



- activation functions are usually element-wise for simplicity; but here an entire **2-layer network** is used as activation function
- fully connected layers are simply replaced by **global average pooling**
- but this is nothing but convolution followed by two **1×1 convolutions**
- 1×1 convolutions are just like matrix multiplications and can be used for **dimension reduction**

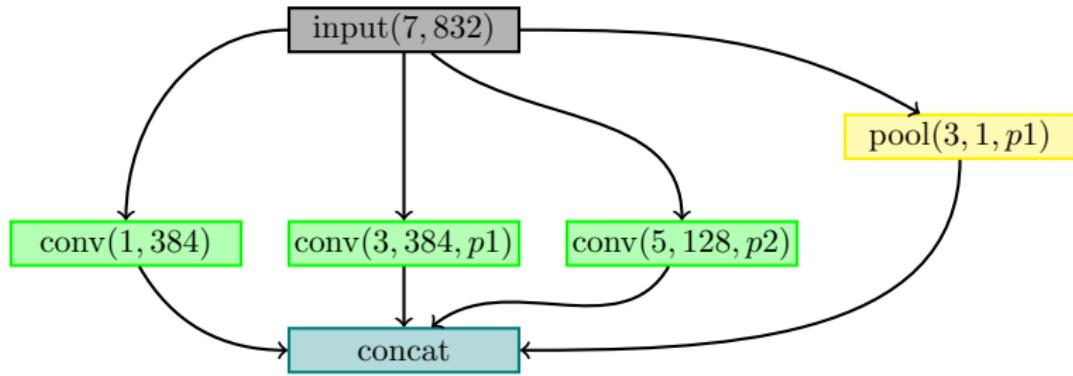
inception module



- **naive** inception module simply concatenates (feature-wise) three convolutions and one max-pooling
 - but this expensive and dimension keeps increasing
 - add **dimension reduction** to control cost, dimensions, and sparsity
 - this is referred to as **inception module**

inception module

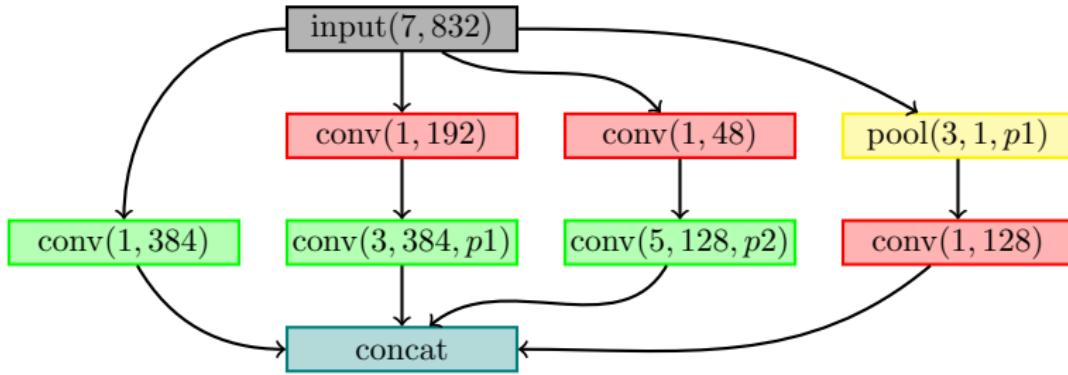
271,418,048 operations



- **naive** inception module simply concatenates (feature-wise) three convolutions and one max-pooling
- but this expensive and dimension keeps increasing
 - add **dimension reduction** to control cost, dimensions, and sparsity
 - this is referred to as **inception module**

inception module

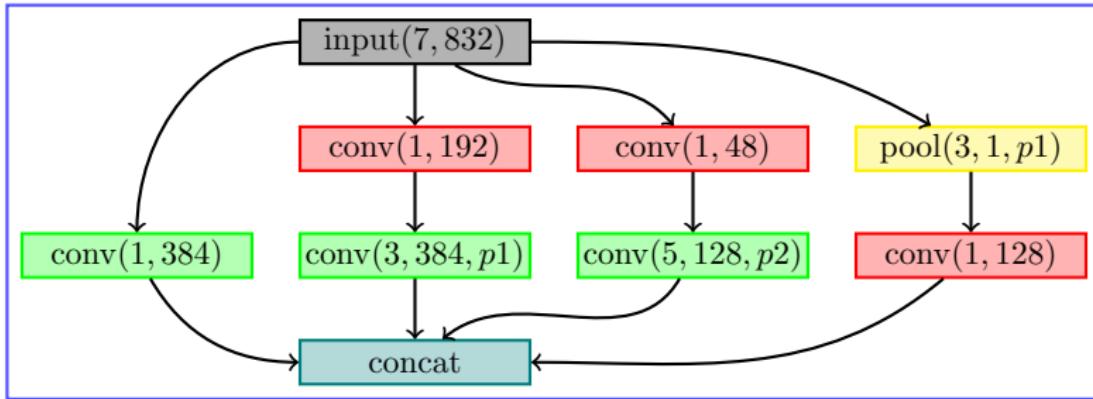
70,800,688 operations



- **naive** inception module simply concatenates (feature-wise) three convolutions and one max-pooling
- but this expensive and dimension keeps increasing
- add **dimension reduction** to control cost, dimensions, and sparsity
- this is referred to as **inception module**

inception module

70,800,688 operations



$\text{inc}(384, (192, 384), (48, 128), 128)$

- **naive** inception module simply concatenates (feature-wise) three convolutions and one max-pooling
- but this expensive and dimension keeps increasing
- add **dimension reduction** to control cost, dimensions, and sparsity
- this is referred to as **inception module**

alternatively: low-rank decomposition

$$Y = h \begin{pmatrix} W & X \end{pmatrix}$$

- X (Y): input (output) features (columns = spatial positions)
- W : weights; h : activation function
- low-rank approximation $W \approx UV^\top$; V is 1×1 spatially
- X was sparse; $V^\top X$ is not
- (in fact, V also includes a non-linearity)

alternatively: low-rank decomposition

$$Y \approx h \begin{pmatrix} U & V^\top & X \end{pmatrix}$$

- X (Y): input (output) features (columns = spatial positions)
- W : weights; h : activation function
- low-rank approximation $W \approx UV^\top$; V is 1×1 spatially
 - X was sparse; $V^\top X$ is not
 - (in fact, V also includes a non-linearity)

alternatively: low-rank decomposition

$$Y \approx h \begin{pmatrix} U & V^\top X \end{pmatrix}$$

- X (Y): input (output) features (columns = spatial positions)
- W : weights; h : activation function
- low-rank approximation $W \approx UV^\top$; V is 1×1 spatially
- X was sparse; $V^\top X$ is not
- (in fact, V also includes a non-linearity)

alternatively: low-rank decomposition

$$Y \approx h \begin{pmatrix} UV^\top X \end{pmatrix}$$

- X (Y): input (output) features (columns = spatial positions)
- W : weights; h : activation function
- low-rank approximation $W \approx UV^\top$; V is 1×1 spatially
- X was sparse; $V^\top X$ is not
- (in fact, V also includes a non-linearity)

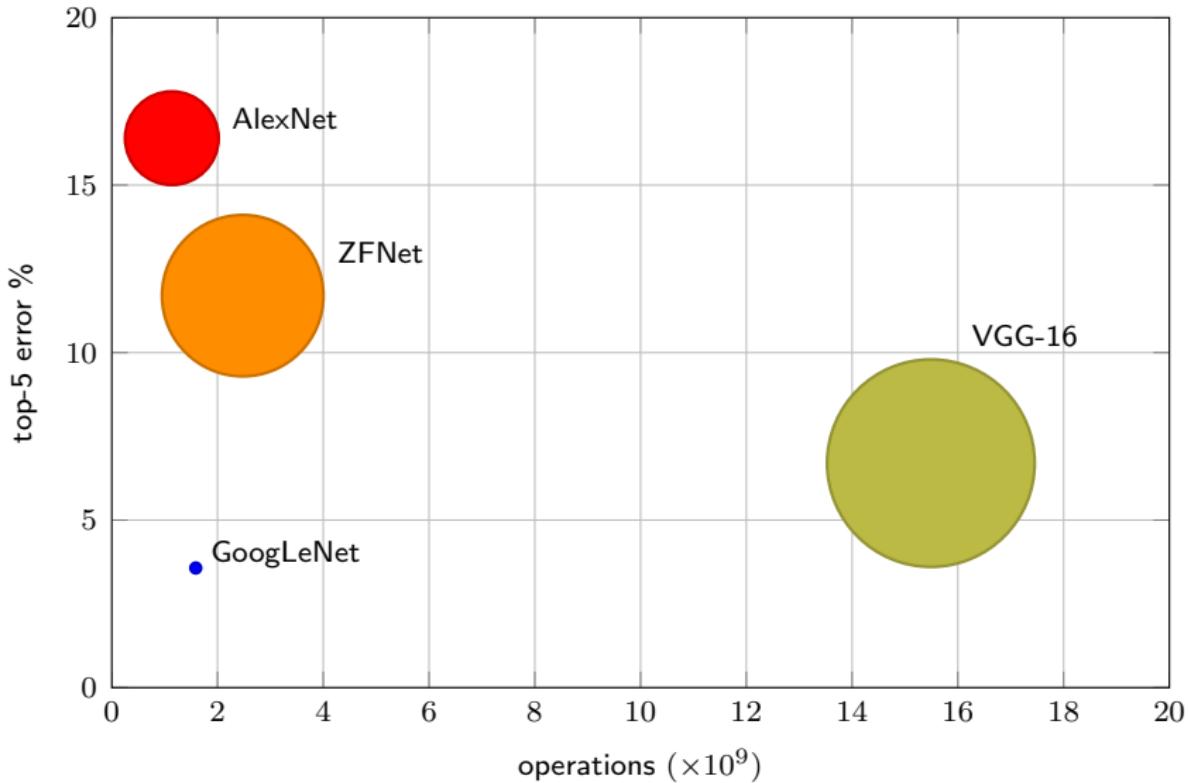
GoogLeNet

		parameters	operations	volume
	input(224, 3)	0	0	224 × 224 × 3
	conv(7, 64, p3, s2)	9,472	118,816,768	112 × 112 × 64
	pool(3, 2, p1)	0	802,816	56 × 56 × 64
	conv(1, 64)	4,160	13,045,760	56 × 56 × 64
	conv(3, 192, p1)	110,784	347,418,624	56 × 56 × 192
	pool(3, 2, p1)	0	602,112	28 × 28 × 192
	inc(64, (96, 128), (16, 32), 32)	163,696	128,488,192	28 × 28 × 256
	inc(128, (128, 192), (32, 96), 64)	388,736	304,969,728	28 × 28 × 480
	pool(3, 2, p1)	0	376,320	14 × 14 × 480
	inc(192, (96, 208), (16, 48), 64)	376,176	73,824,576	14 × 14 × 512
	inc(160, (112, 224), (24, 64), 64)	449,160	88,135,712	14 × 14 × 512
	inc(128, (128, 256), (24, 64), 64)	510,104	100,080,736	14 × 14 × 512
avg(5, 3, p2)	inc(112, (144, 288), (32, 64), 64)	605,376	118,754,048	14 × 14 × 528
conv(1, 128)	inc(256, (160, 320), (32, 128), 128)	868,352	170,300,480	14 × 14 × 832
fc(1024)	pool(3, 2, p1)	0	163,072	7 × 7 × 832
fc(1000)	inc(256, (160, 320), (32, 128), 128)	1,043,456	51,170,112	7 × 7 × 832
softmax	inc(384, (192, 384), (48, 128), 128)	1,444,080	70,800,688	7 × 7 × 1024
	avg(7)	0	50,176	1 × 1 × 1024
	fc(1000)	1,025,000	1,025,000	1,000
	softmax	0	1,000	1,000

GoogLeNet

		parameters	operations	volume
	input(224, 3)	0	0	224 × 224 × 3
	conv(7, 64, p3, s2)	9,472	118,816,768	112 × 112 × 64
	pool(3, 2, p1)	0	802,816	56 × 56 × 64
	conv(1, 64)	4,160	13,045,760	56 × 56 × 64
	conv(3, 192, p1)	110,784	347,418,624	56 × 56 × 192
	pool(3, 2, p1)	0	602,112	28 × 28 × 192
auxiliary classifier	inc(64, (96, 128), (16, 32), 32)	163,696	128,488,192	28 × 28 × 256
	inc(128, (128, 192), (32, 96), 64)	388,736	304,969,728	28 × 28 × 480
	pool(3, 2, p1)	0	376,320	14 × 14 × 480
	inc(192, (96, 208), (16, 48), 64)	376,176	73,824,576	14 × 14 × 512
	inc(160, (112, 224), (24, 64), 64)	449,160	88,135,712	14 × 14 × 512
	inc(128, (128, 256), (24, 64), 64)	510,104	100,080,736	14 × 14 × 512
	inc(112, (144, 288), (32, 64), 64)	605,376	118,754,048	14 × 14 × 528
	inc(256, (160, 320), (32, 128), 128)	868,352	170,300,480	14 × 14 × 832
	pool(3, 2, p1)	0	163,072	7 × 7 × 832
auxiliary classifier	inc(256, (160, 320), (32, 128), 128)	1,043,456	51,170,112	7 × 7 × 832
	inc(384, (192, 384), (48, 128), 128)	1,444,080	70,800,688	7 × 7 × 1024
	avg(7)	0	50,176	1 × 1 × 1024
	fc(1000)	1,025,000	1,025,000	1,000
auxiliary classifier	softmax	0	1,000	1,000
	softmax			

network performance



summary

- convolution \equiv linearity, translation equivariance
- sparse connections, weight sharing: fully connected \rightarrow convolution
- cross-correlation
- feature maps: matrix multiplication & convolution combined
- 1×1 convolution
- convolution as regularization, structured convolution
- standard, padded, strided, dilated; and their derivatives
- pooling and invariance
- deeper networks
- LeNet-5, AlexNet, ZFNet, VGG-16, NiN, GoogLeNet