

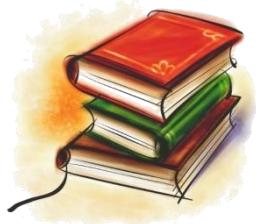
# شبکه‌های عصبی مصنوعی

## شبکه‌های عصبی پیچشی (CNN)

هادی ویسی

[h.veisi@ut.ac.ir](mailto:h.veisi@ut.ac.ir)

دانشگاه تهران - دانشکده علوم و فنون نوین



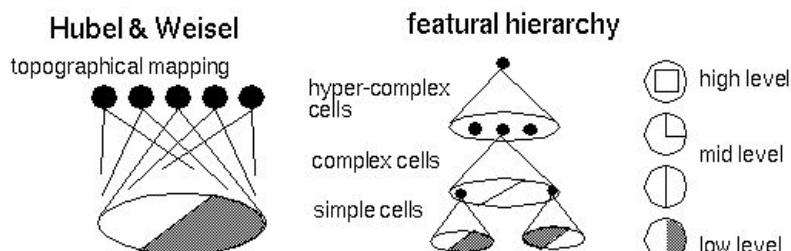
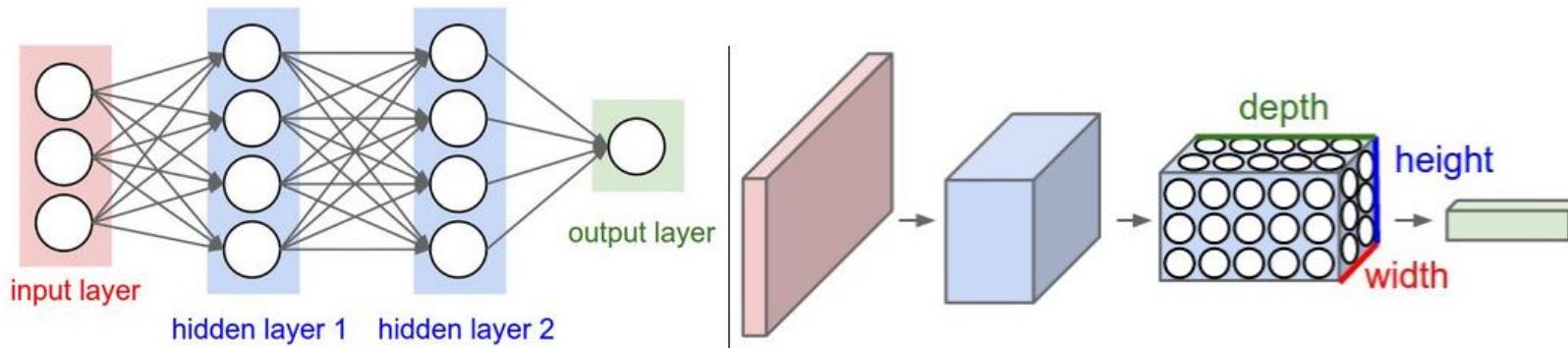
## فهرست

- معرفی شبکه‌های عصبی پیچشی (CNN)
  - ساختار شبکه و لایه‌های آن
    - لایه پیچش
    - لایه نمونه برداری
    - لایه تمام متصل
  - معماری شبکه پیچشی
- آموزش شبکه پیچشی
  - پس انتشار خطا در لایه‌های مختلف
- معماری‌های شناخته شده
  - AlexNet
  - GoogleNet
  - ResNet
- بسترهاي يادگيري عميق

## معرفی ...

### ○ شبکه عصبی پیچشی (CNN: Convolutional Neural Network)

- مشابه شبکه‌های عصبی رو به جلو



- تغییر یافته برای ورودی تصویر
- ارتباط دو بعدی مقادیر پیکسل‌ها با هم
- الهام گرفتن از سیستم بینایی چشم انسان

- روش رایج برای استخراج ویژگی
- استخراج سلسله مراتبی اطلاعات

## ساختار شبکه...

- لایه‌های رایج در این شبکه (به ترتیب)

- ورودی (Input)

- یک تصویر ۳ بعدی: طول، عرض و عمق (مثلاً معادل ۳ برابر با سه مولفه RGB)

- پیچش (Convolution)

- هر نرون بلوکی از پیکسل‌های نزدیک به هم را می‌بیند

- تابع فعال‌سازی ReLU

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

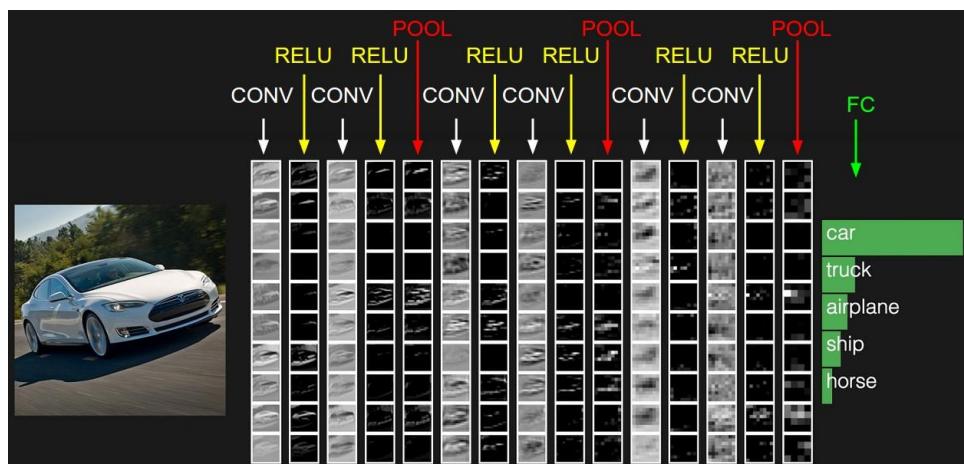
- غیرخطی کردن رفتار شبکه

- نمونه‌برداری (Pooling)

- کاهش اندازه فضای ویژگی‌ها

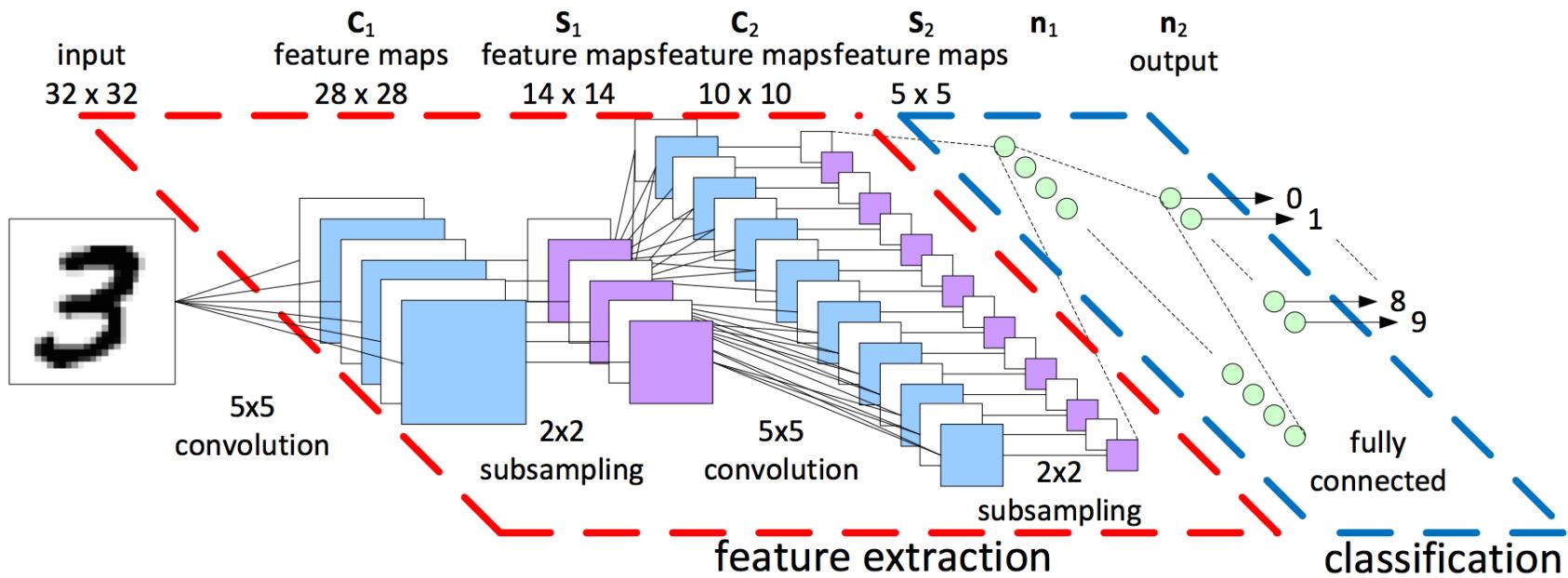
- تمام متصل (Fully Connected)

- برای دسته‌بندی، مانند شبکه‌های عصبی MLP



## ساختار شبکه ...

- یک لایه ورودی
- یک یا چند لایه پیچش (Convolution)
- یک یا چند لایه نمونه‌برداری (Pooling)
- یک لایه خروجی



## ساختار شبکه

- نگاشت یک تصویر سه بعدی به یک بردار
- ورودی هر لایه (ورودی، پیچش و نمونه‌برداری) یک تصویر سه بعدی و خروجی آن نیز یک تصویر سه بعدی است
- لایه‌های پیچش و تمام متصل دارای وزن قابل تنظیم هستند
- لایه نمونه‌برداری ثابت است (عدم نیاز به آموزش)

## نهایه پیچش ...

- به کمک فیلترهایی، وظیفه استخراج ویژگی‌ها را بر عهده دارد
- فیلترها = وزن‌ها
- هر فیلتر برای یادگیری یک ویژگی است

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4



1	0
0	1

=

7		

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4



1	0
0	1

=

7	5	



## لایه پیچش: پارامترها ...

### ○ اندازه فیلتر = receptive field

- اندازه ناحیه‌ای از ورودی که با هم بعد از ضرب در فیلتر به یک ویژگی تبدیل می‌شوند

• مثال: تصویر ورودی [32x32x3] (RGB ۳۲\*۳۲ پیکسل و

○ با فیلترهای ۵\*۵ آنگاه هر نرون در لایه پیچش دارای وزن‌هایی با ابعاد [5x5x3] در لایه ورودی است

$$\text{تعداد وزن ها} = 75 = 5*5*3 + \text{یک بایاس}$$

### ○ تعداد فیلتر = depth

• تعداد فیلترهایی که می‌خواهیم استفاده کنیم

• تعیین توسط طراح شبکه

• مثال: تصویر ورودی [32x32x3] (RGB ۳۲\*۳۲ پیکسل و

○ استفاده از ۱۲ فیلتر، اندازه [32x32x12] را در لایه پیچش نتیجه می‌دهد

○ تبدیل تصویر سه بعدی [32x32x3] به تصویر سه بعدی [32x32x12]

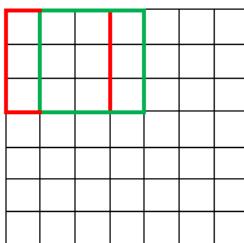
○ اعمال تابع ReLU روی این تصویر جدید

## نیه پیچش: پارامترها ...

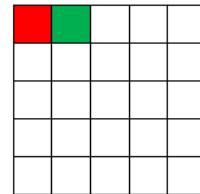
○ گام = stride

- تعداد پیکسل‌های حرکت روی تصویر ورودی
- برای حالتی که اندازه تصویر از اندازه فیلتر بزرگ‌تر است (که در عمل معمولاً اینگونه است)
- تاثیر روی اندازه تصویر خروجی
- هرچه بزرگ‌تر باشد، تصویر خروجی کوچک‌تر می‌شود
- مقدار معمول ۱ و گاهی ۲

7 x 7 Input Volume



5 x 5 Output Volume

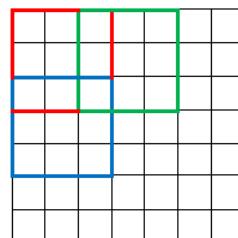


◦ مثال: مقدار ۱

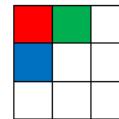
◦ خروجی ۵\*۵

- کوچک‌تر از ورودی

7 x 7 Input Volume



3 x 3 Output Volume



◦ مثال: مقدار ۲

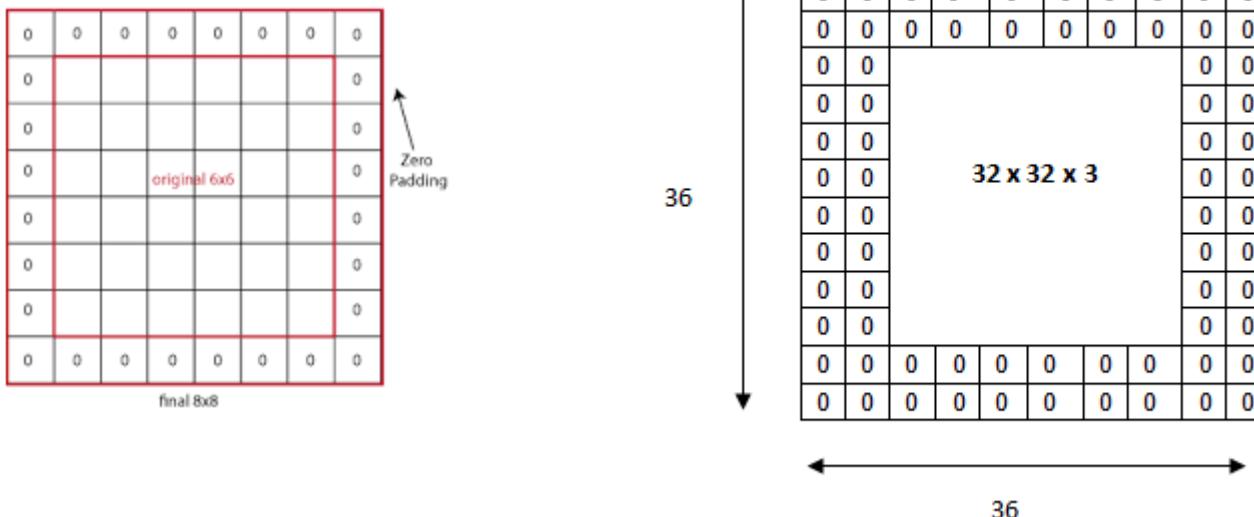
◦ افقی و عمودی

◦ خروجی ۳\*۳

## نیه پیچش: پارامترها ...

### ○ افزودن صفر = zero-padding

- افزودن سطرها و ستون‌های صفر در اطراف تصویر
- کمک به کنترل اندازه تصویر خروجی
- برای اینکه برابر با تصویر ورودی باشد
- می‌تواند مقادیر مختلف (۱ یا ۲ یا بیشتر!) باشد



## نیه پیچش: پارامترها . . .

- تنظیم مقادیر پارامترها . . .

- اندازه تصویر ورودی =  $W$

- اندازه فیلتر =  $F$

- اندازه گام =  $S$

- اندازه افزودن صفر =  $P$

- اندازه تصویر خروجی:  $(W-F+2P)/S+1$

- مثال: اگر تصویر ورودی  $7 \times 7$  باشد و فیلترها  $3 \times 3$  باشد

- با گام ۱ و صفر افزوده نشده باشد، اندازه خروجی  $5 \times 5$  است:  $(7-3+2 \times 0)/1+1 = 5$

- با گام ۲ و صفر افزوده نشده باشد، اندازه خروجی  $3 \times 3$  است:  $(7-3+2 \times 0)/2+1 = 3$

- برای گام ۱، تعداد صفرهای لازم جهت برابری اندازه تصویر ورودی و خروجی:  $(F-1)/2$

- مثال: اگر تصویر ورودی  $7 \times 7$  باشد، فیلترها  $3 \times 3$  باشد و گام ۱ باشد

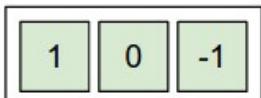
- تعداد صفر لازم =  $(3-2)/2 = 1$

## نیه پیچش: پارامترها ...

◦ تنظیم مقادیر پارامترها: مثال (یک بعدی) ...



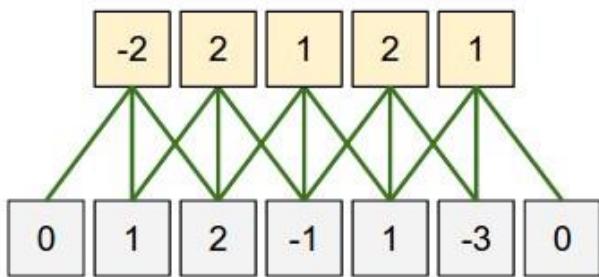
◦ اندازه تصویر ورودی ( $W = 5$ )



◦ اندازه فیلتر ( $F = 3$ )

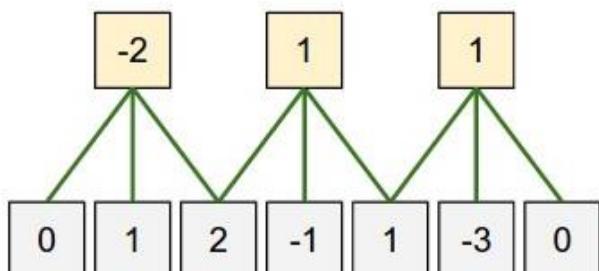


◦ اندازه افزوده صفر ( $P = 1$ )



◦ گام ۱  $\Leftrightarrow$  اندازه تصویر خروجی ( $S = 5$ )

$$(W-F+2P)/S+1=(5-3+2)/1+1=5$$



◦ گام ۲  $\Leftrightarrow$  اندازه تصویر خروجی ( $S = 3$ )

$$(5-3+2)/2+1=3$$

◦ گام ۳  $\Leftrightarrow$  اندازه تصویر خروجی ( $S = ?$ )



## نیه پیچش: پارامترها ...

### ○ تنظیم مقادیر پارامترها: محدودیت در مقدار گام

- مقدار گام باید به گونه‌ای باشد که اندازه تصویر خروجی مقدار معتبری (عدد صحیح) باشد
- در مثال اسلاید قبل برای گام ۳ مقدار خروجی غیرمعتبر می‌شود  
 $(5-3+2)/3+1=2.3$  ○
- راه حل: انتخاب مقدار مناسب یا افزودن صفر به تعداد کافی



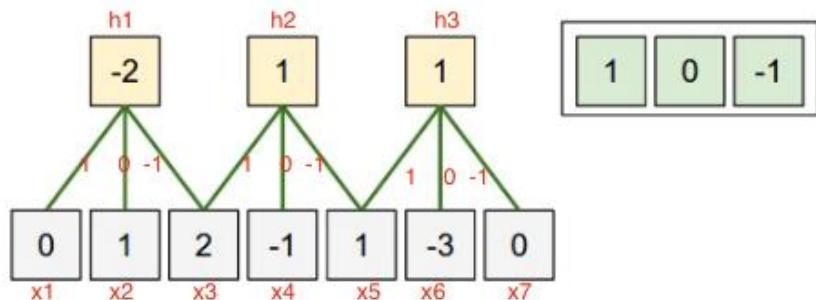
## لایه پیچش: پارامترها ...

### ◦ تنظیم مقادیر پارامترها: نمونه عملی

- شبکه AlexNet در مسابقه ImageNet در ۲۰۱۲
- تصاویر ورودی: [227x227x3]
- در لایه پیچش اول
- اندازه فیلتر = ۱۱
- اندازه گام = ۴
- تعداد صفرهای افزوده شده =
- تعداد فیلتر = ۹۶
- تصویر لایه پیچش اول = [55x55x96]
- $(227 - 11)/4 + 1 = 55$
- هر کدام معادل ناحیه [11x11x3] از تصویر ورودی

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

## لایه پیچش ...



### اشتراک وزن‌ها ...

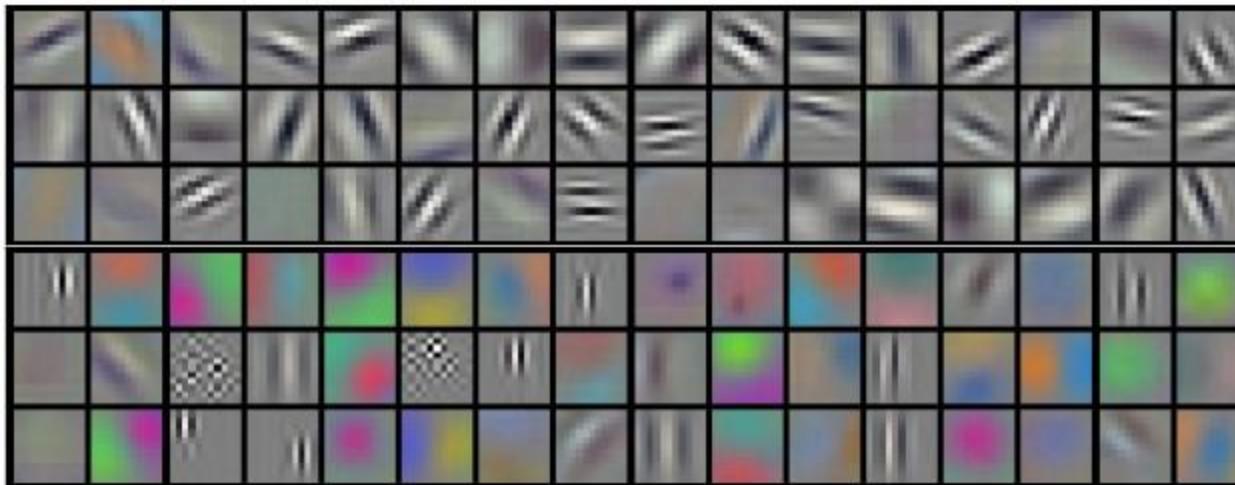
- تعداد خیلی زیاد پارامترهای شبکه
- در AlexNet تعداد وزن‌های بین لایه ورودی به لایه پیچش اول برابر است با 364
- تعداد نرون‌های ورودی =  $11 * 11 * 3 = 363$  به همراه یک بایاس = 55\*55\*96 = 290,400
- تعداد وزن‌ها = 105,705,600 (خیلی زیاد!)
- یک فرض منطقی!: اگر یک ویژگی (مانند لبه) در یک موقعیت مکانی از تصویر برای محاسبه مفید باشد، همان ویژگی در موقعیت دیگری نیز مفید است.
- وزن‌های همه برش‌ها (Slice) از تصویر یکسان در نظر گرفته می‌شود
- در 96\*55\*55 تعداد 96 برش 55\*55 داریم
- در مثال بالا، کلا 96 مجموعه وزن، که هر کدام 11\*11\*3 هستند، در نظر گرفته می‌شود
- تعداد وزن‌ها =  $96 * 11 * 11 * 3 = 34,848$  به همراه 96 بایاس = 34,944
- تعداد 55\*55 نرون در هر برش از یک وزن استفاده می‌کنند
- در فاز آموزش، گرادیان برای همه نرون‌ها محاسبه می‌شود اما گرادیان همه وزن‌های آن برش باهم جمع شده و یک مجموعه وزن را می‌سازند

## لایه پیچش ...

### ◦ اشتراک وزن‌ها ...

◦ در AlexNet

- تعداد نرون‌های ورودی =  $11*11*3 = 363$  به همراه یک بایاس
- تعداد نرون‌های لایه پیچش =  $55*55*96 = 290,400$
- تعداد فیلترها = ۹۶
- تعداد ۹۶ مجموعه وزن (فیلتر)، که هر کدام  $11*11*3$  هستند، در نظر گرفته می‌شود



## لایه پیچش ...

### ◦ اشتراک وزن‌ها

- اگر همه وزن‌های یک برش از تصویر ثابت باشد، آنگاه گام جلو رو در لایه پیچش از شبکه معادل پیچش (convolution) تصویر ورودی و وزن‌ها است
- اسم گذاری شبکه
- برای همین به وزن‌ها فیلتر (یا هسته) هم گفته می‌شود

### ◦ گاهی اشتراک وزن استفاده نمی‌شود

- لایه را Locally-Connected Layer می‌نامند
- در مواردی استفاده می‌شود که تصویر ورودی یک ساختار مرکز مشخص دارد و نیاز است از هر بخش از تصویر یک ویژگی خاص به شبکه آموخته شود
- مثال: چهره-ناحیه چشم با ناحیه موها ویژگی‌های متفاوتی دارند و باید جداگانه آموخته بینند



## نهاده پیچش: دمو ...

دمو!

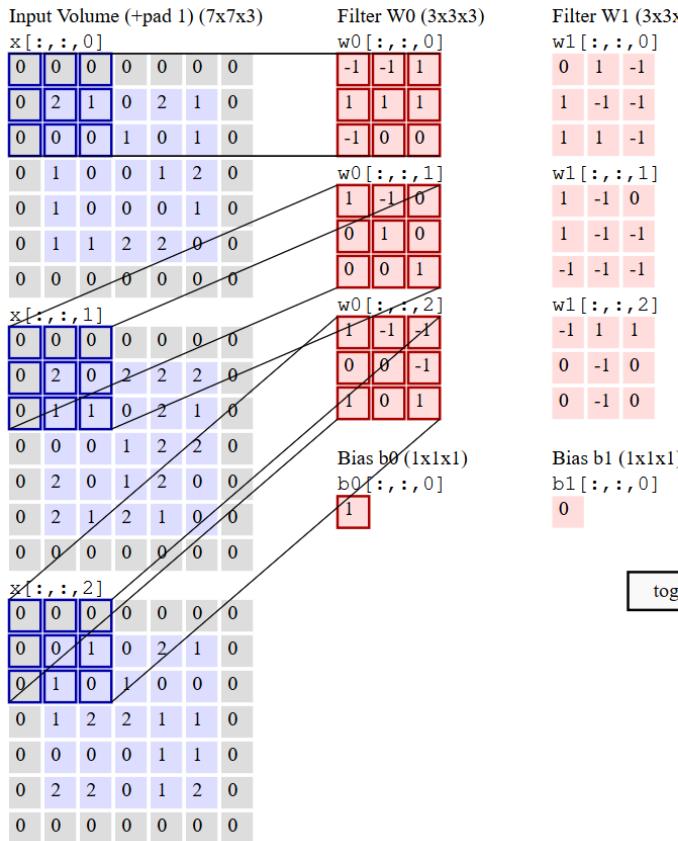
• ورودی: تصویر سه بعدی [5x5x3]

• اندازه فیلتر =  $F=3$

• تعداد فیلتر =  $K=2$

• اندازه گام =  $S=2$

• افزودن صفر =  $P=1$



<http://cs231n.github.io/convolutional-networks>

Hadi Veisi (h.veisi@ut.ac.ir)

## نیه پیچش: دمو ...

Input Volume (+pad 1) (7x7x3)									Filter W0 (3x3x3)		
									w0[:, :, 0]		
0	0	0	0	0	0	0	0	0	-1	-1	1
0	2	1	0	2	1	0	0	0	1	1	1
0	0	0	1	0	1	0	0	0	-1	0	0
0	1	0	0	1	2	0	0	0	1	-1	0
0	1	0	0	0	0	1	0	0	0	-1	0
0	1	1	2	2	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1
x[:, :, 1]									w0[:, :, 1]		
0	0	0	0	0	0	0	0	0	1	-1	0
0	2	0	2	2	2	0	0	0	0	0	-1
0	1	1	0	2	1	0	0	0	1	0	1
0	0	0	1	2	2	0	0	0	1	-1	-1
0	2	0	1	2	0	0	0	0	0	0	-1
0	2	1	2	1	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0
x[:, :, 2]									w0[:, :, 2]		
0	0	0	0	0	0	0	0	0	1	-1	-1
0	0	1	0	2	1	0	0	0	0	0	-1
0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	0	2	1	1	0	0	1	0	1
0	0	0	0	1	1	1	0	0	0	0	0
0	2	2	0	1	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Filter W1 (3x3x3)									Output Volume (3x3x2)		
									o[:, :, 0]		
0	1	-1	6	6	6						
1	-1	-1	-2	4	7						
1	1	-1	0	5	4						
									o[:, :, 1]		
1	-1	0	-8	-8	-2						
1	-1	-1	-3	-6	-2						
-1	-1	-1	-8	-5	4						
									w1[:, :, 2]		
-1	1	1	-1	-1	0						
0	-1	0	0	-1	0						
0	-1	0	0	-1	0						
									Bias b1 (1x1x1)		
									b1[:, :, 0]	0	

toggle movement

فیلتر اول  
اولین نرون برش اول

## نیمه پیچش: دمو ...

Input Volume (+pad 1) (7x7x3)									Filter W0 (3x3x3)		
									w0[:, :, 0]		
0	0	0	0	0	0	0	0	0	-1	-1	1
0	2	1	0	2	1	0	0	0	1	1	1
0	0	0	1	0	1	0	0	0	-1	0	0
0	1	0	0	1	2	0	0	0	1	-1	0
0	1	0	0	0	0	1	0	0	0	1	0
0	1	1	2	2	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1
x[:, :, 1]									w0[:, :, 1]		
0	0	0	0	0	0	0	0	0	1	-1	-1
0	2	0	2	2	2	0	0	0	0	0	-1
0	1	1	0	2	1	0	0	0	1	0	1
0	0	0	1	2	2	0	0	0	1	-1	-1
0	2	0	1	2	0	0	0	0	0	-1	0
0	2	1	2	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	-1	0
x[:, :, 2]									w0[:, :, 2]		
0	0	0	0	0	0	0	0	0	-1	1	1
0	0	1	0	2	1	0	0	0	0	-1	0
0	1	0	1	0	0	0	0	0	1	0	1
0	1	2	2	1	1	0	0	0	1	-1	-1
0	0	0	0	1	1	0	0	0	0	-1	0
0	2	2	0	1	2	0	0	0	0	-1	0
0	0	0	0	0	0	0	0	0	0	-1	0

Filter W1 (3x3x3)  
w1[:, :, 0]

0	1	-1
1	-1	-1
1	1	-1

w1[:, :, 1]

1	-1	0
1	-1	-1
-1	-1	-1

w1[:, :, 2]

-1	1	1
0	-1	0
0	-1	0

Bias b0 (1x1x1)

b0[:, :, 0]

0

Bias b1 (1x1x1)  
b1[:, :, 0]

0

toggle movement

فیلتر اول  
دومین نرون برش اول  
گام = ۲ (افقی)



## این پیچش: دمو ...

Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$

0	0	0	0	0	0	0	0	0
0	2	1	0	2	1	0	1	1
0	0	0	1	0	1	0	-1	0
0	1	0	0	1	2	0	1	-1
0	1	0	0	0	1	0	0	1
0	1	1	2	2	0	0	0	1
0	0	0	0	0	0	0	0	0

$x[:, :, 1]$

0	0	0	0	0	0	0	0	0
0	2	0	2	2	2	0	0	-1
0	1	1	0	2	1	0	1	0
0	0	0	1	2	2	0	1	-1
0	2	0	1	2	0	0	0	0
0	2	1	2	1	0	0	0	0
0	0	0	0	0	0	0	0	0

$x[:, :, 2]$

0	0	0	0	0	0	0	0	0
0	0	1	0	2	1	0	1	-1
0	1	0	1	0	0	0	0	0
0	1	2	2	1	1	0	0	0
0	0	0	0	1	1	0	0	0
0	2	2	0	1	2	0	0	0
0	0	0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$

Filter W1 (3x3x3)

$w1[:, :, 0]$

Output Volume (3x3x2)

$o[:, :, 0]$

0	1	-1
1	-1	-1
1	1	-1

1	-1	0
1	-1	-1
-1	-1	-1

-1	1	1
0	-1	0
0	-1	0

6	6	6
-2	4	7
0	5	4

-8	-8	-2
-3	-6	-2
-8	-5	4

1	-1	0
1	-1	-1
-1	-1	-1

-1	1	1
0	-1	0
0	-1	0

0	1	1
0	-1	0
0	-1	0

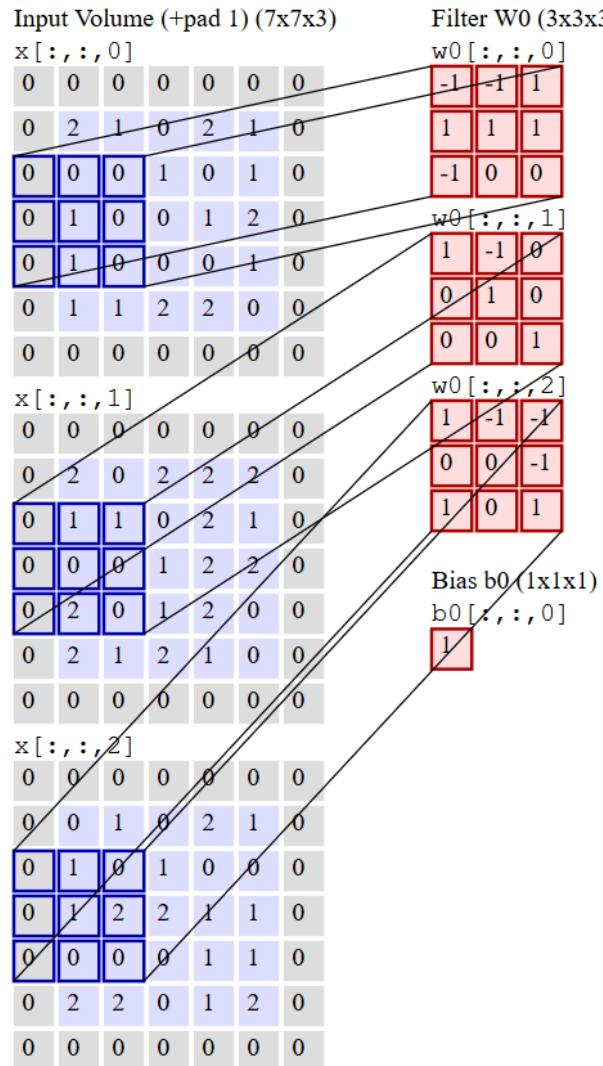
Bias b0 (1x1x1)

$b0[:, :, 0]$

0

toggle movement

## نیمه پیچش: دمو ...



Filter W1 (3x3x3)

w1[:, :, 1]		
1 -1 0	-8 -8 -2	1 -1 0
1 -1 -1	-3 -6 -2	1 -1 -1
-1 -1 -1	-8 -5 4	-1 -1 -1

Output Volume (3x3x2)

o[:, :, 0]		
6 6 6	-2 4 7	0 5 4
0 5 4	-8 -8 -2	-3 -6 -2
-8 -5 4	-1 1 1	0 -1 0
-1 1 1	0 -1 0	0 -1 0

w1[:, :, 2]

-1 1 1	0 -1 0	0 -1 0
0 -1 0	0 -1 0	0 -1 0
0 -1 0	0 -1 0	0 -1 0

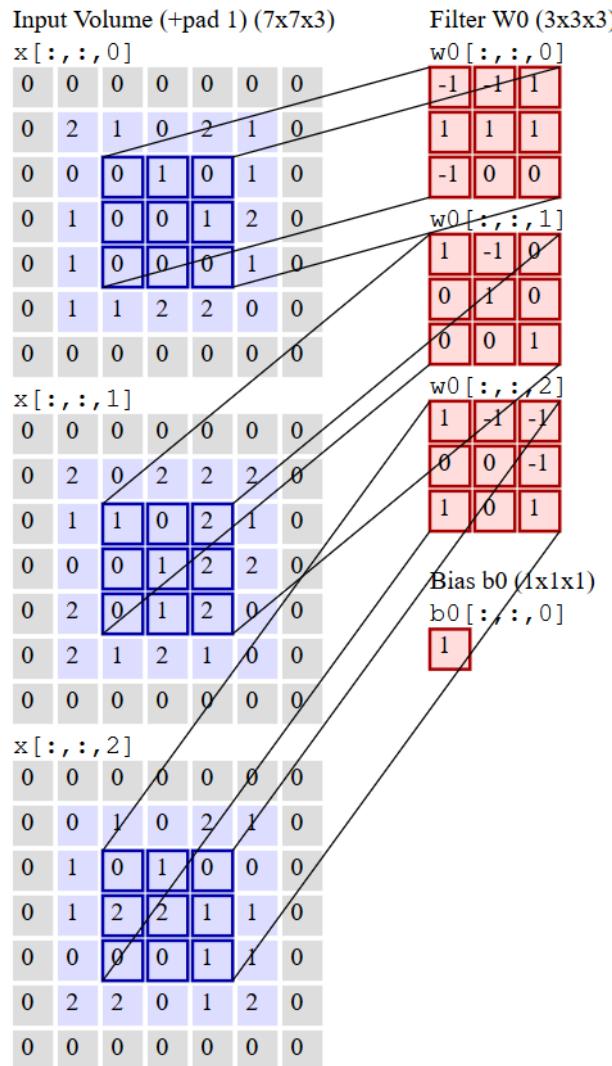
Bias b1 (1x1x1)  
b1[:, :, 0]

0
---

toggle movement

فیلتر اول  
چهارمین نرون برش اول  
گام = ۲ (عمودی)

## این پیچش: دمو ...



Filter W1 (3x3x3)

$w1[:, :, 0]$

0	1	-1	
1	-1	-1	
1	1	-1	

$w1[:, :, 1]$

1	-1	0	
1	-1	-1	
-1	-1	-1	

$w1[:, :, 2]$

-1	1	1	
0	-1	0	
0	-1	0	

Output Volume (3x3x2)

$\circ[:, :, 0]$

6	6	6	
-2	4	7	
0	5	4	

$\circ[:, :, 1]$

-8	-8	-2	
-3	-6	-2	
-8	-5	4	

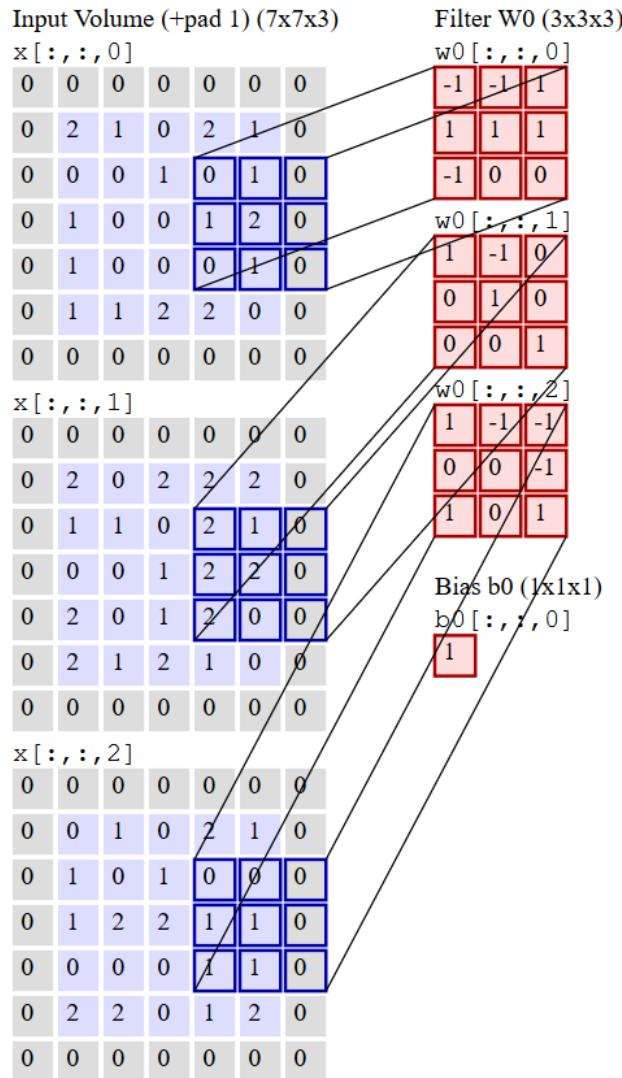
Bias b1 (1x1x1)

$b1[:, :, 0]$

0
---

toggle movement

## این پیچش: دمو ...



Filter W1 (3x3x3)

w1 [ :, :, 0 ]	0 1 -1 1 -1 -1 1 1 -1
w1 [ :, :, 1 ]	1 -1 0 1 -1 -1 -1 -1 -1
w1 [ :, :, 2 ]	-1 1 1 0 -1 0 0 -1 0

Bias b1 (1x1x1)

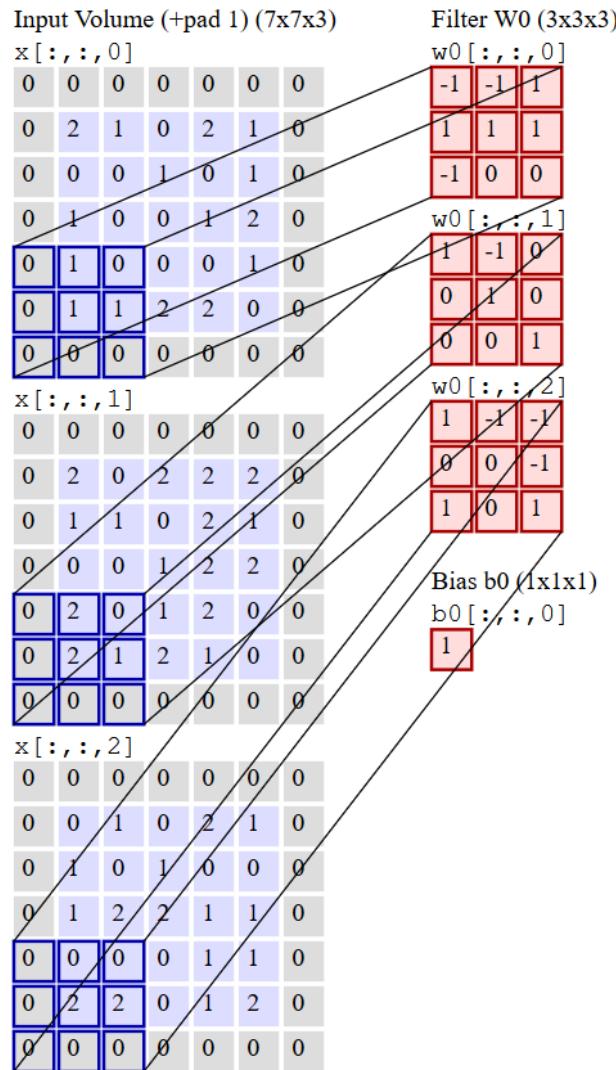
b1 [ :, :, 0 ]	0
----------------	---

Output Volume (3x3x2)

o [ :, :, 0 ]	6 6 6 -2 4 7 0 5 4
o [ :, :, 1 ]	-8 -8 -2 -3 -6 -2 -8 -5 4

toggle movement

## این پیچش: دمو . . .

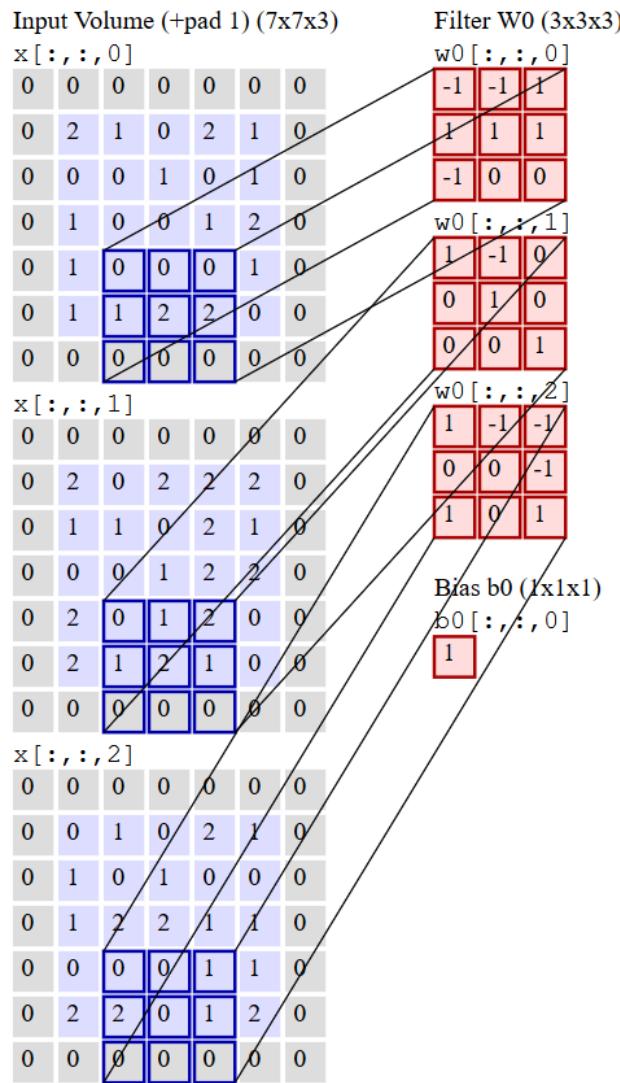


Output Volume (3x3x2)

o[:, :, 0]	
6 6 6	
-2 4 7	
0 5 4	
o[:, :, 1]	
1 -1 0	
1 -1 -1	
-1 -1 -1	
Bias b1 (1x1x1)	
b1[:, :, 0]	
0	

toggle movement

## این پیچش: دمو ...



Filter W1 (3x3x3)

w1[:, :, 0]	0 1 -1 1 -1 -1 1 1 -1
w1[:, :, 1]	1 -1 0 1 -1 -1 -1 -1 -1
w1[:, :, 2]	-1 1 1 0 -1 0 0 -1 0

Output Volume (3x3x2)

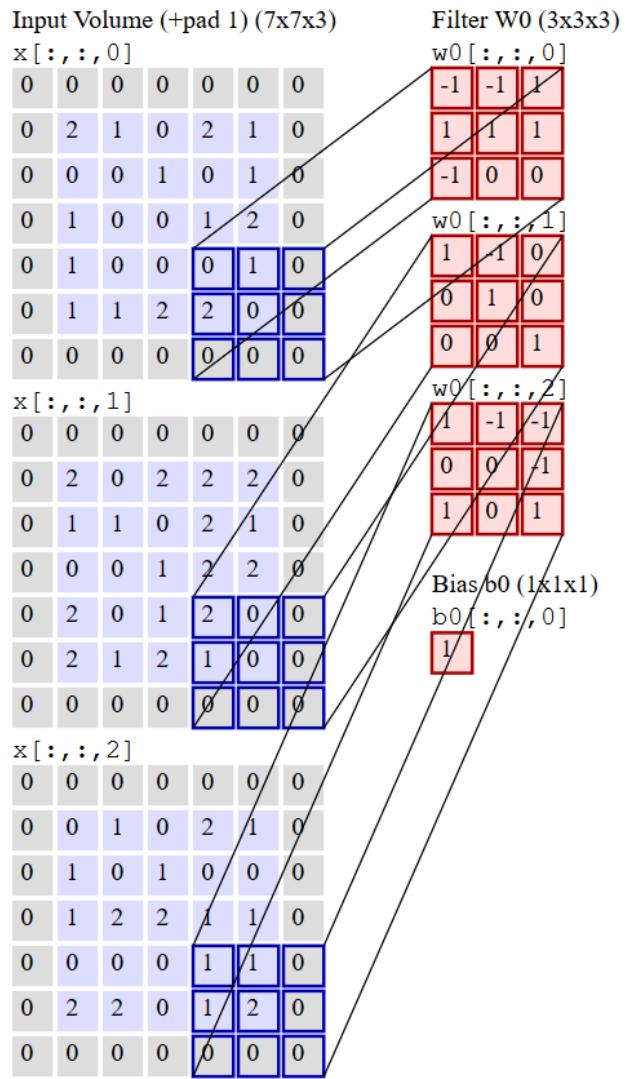
o[:, :, 0]	6 6 6 -2 4 7 0 5 4
o[:, :, 1]	-8 -8 -2 -3 -6 -2 -8 -5 4

Bias b1 (1x1x1)

b1[:, :, 0]	0
-------------	---

toggle movement

## این پیچش: دمو ...



Filter W1 (3x3x3)

w1[:, :, 0]

Output Volume (3x3x2)

o[:, :, 0]

0	1	-1
1	-1	-1
1	1	-1

o[:, :, 1]

1	-1	0
1	-1	-1
-1	-1	-1

w1[:, :, 2]

-1	1	1
0	-1	0
0	-1	0

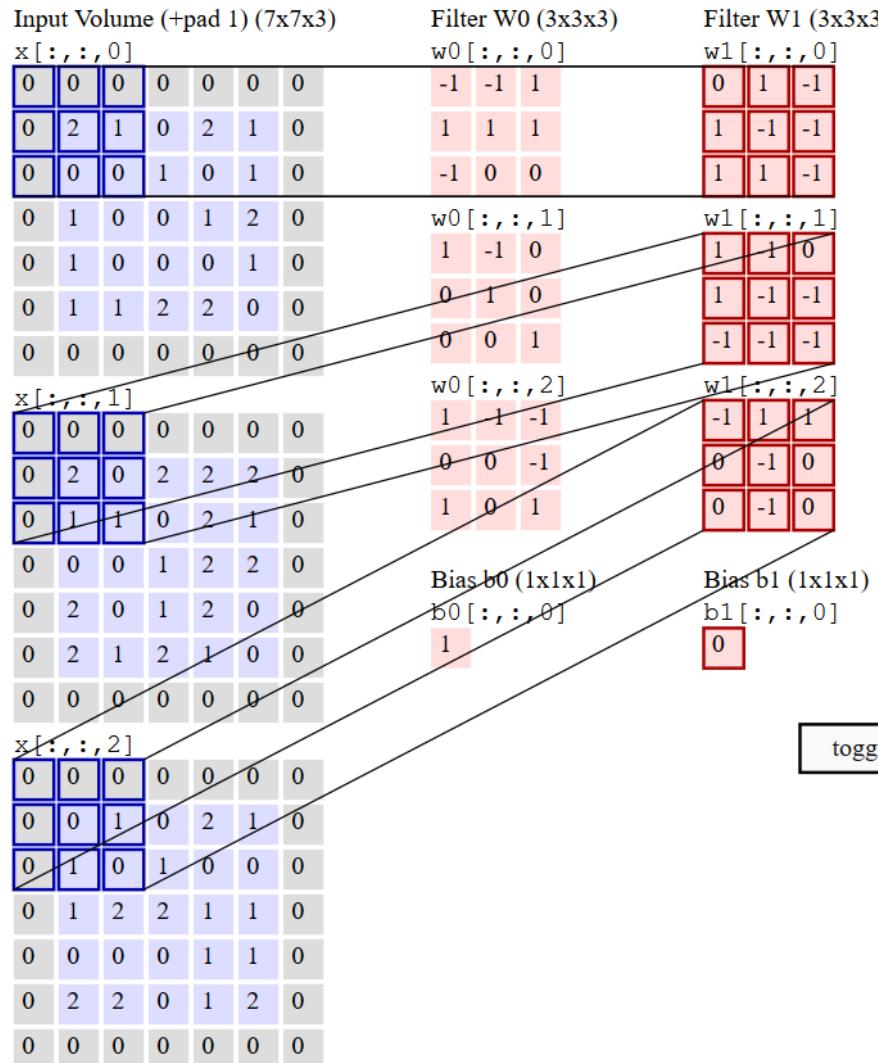
Bias b1 (1x1x1)

b1[:, :, 0]

0
---

toggle movement

## نیه پیچش: دمو ...

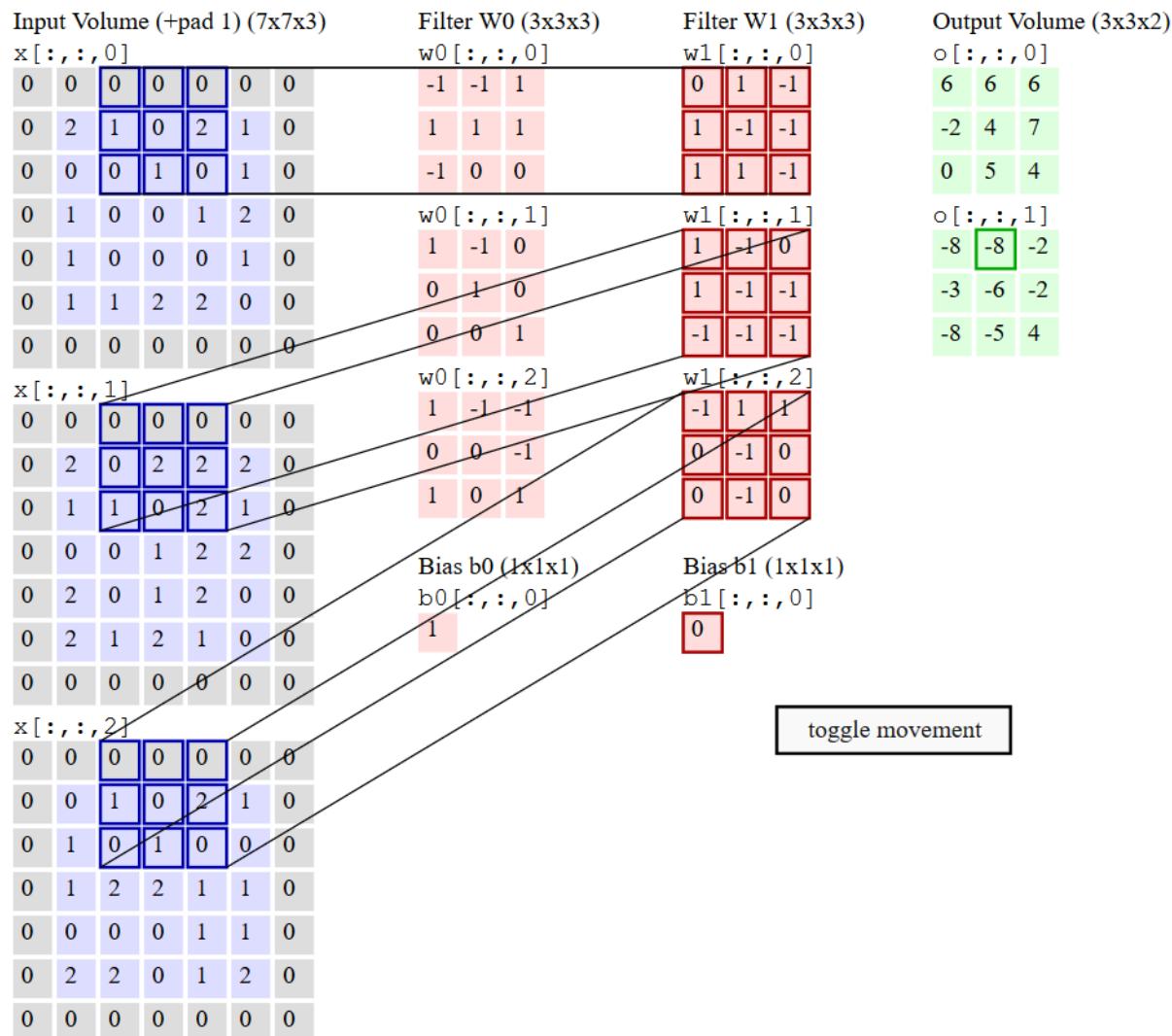


Output Volume (3x3x2)

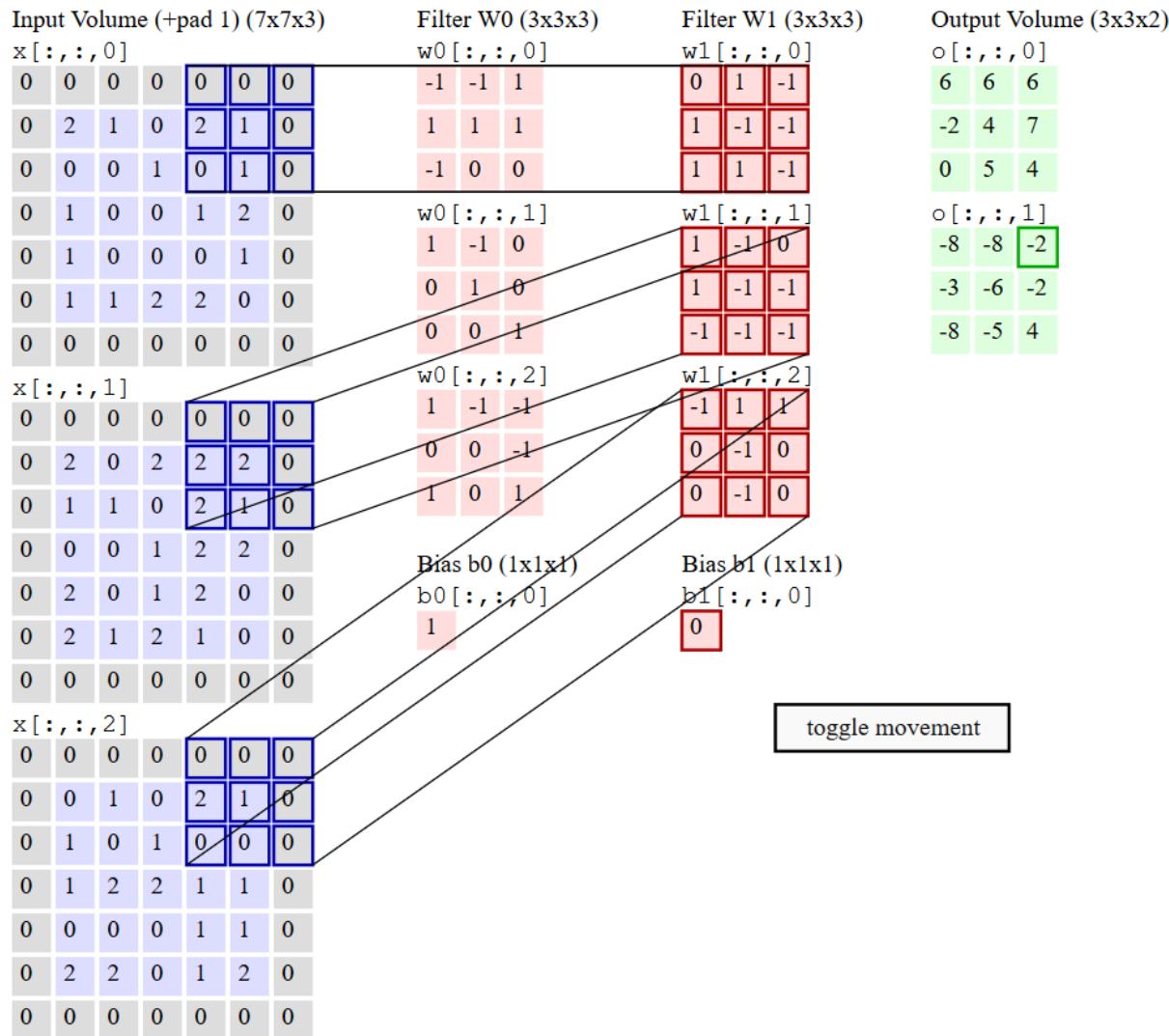
o[:, :, 0]	6 6 6 -2 4 7 0 5 4
o[:, :, 1]	-8 -8 -2 -3 -6 -2 -8 -5 4

فیلتر دوم

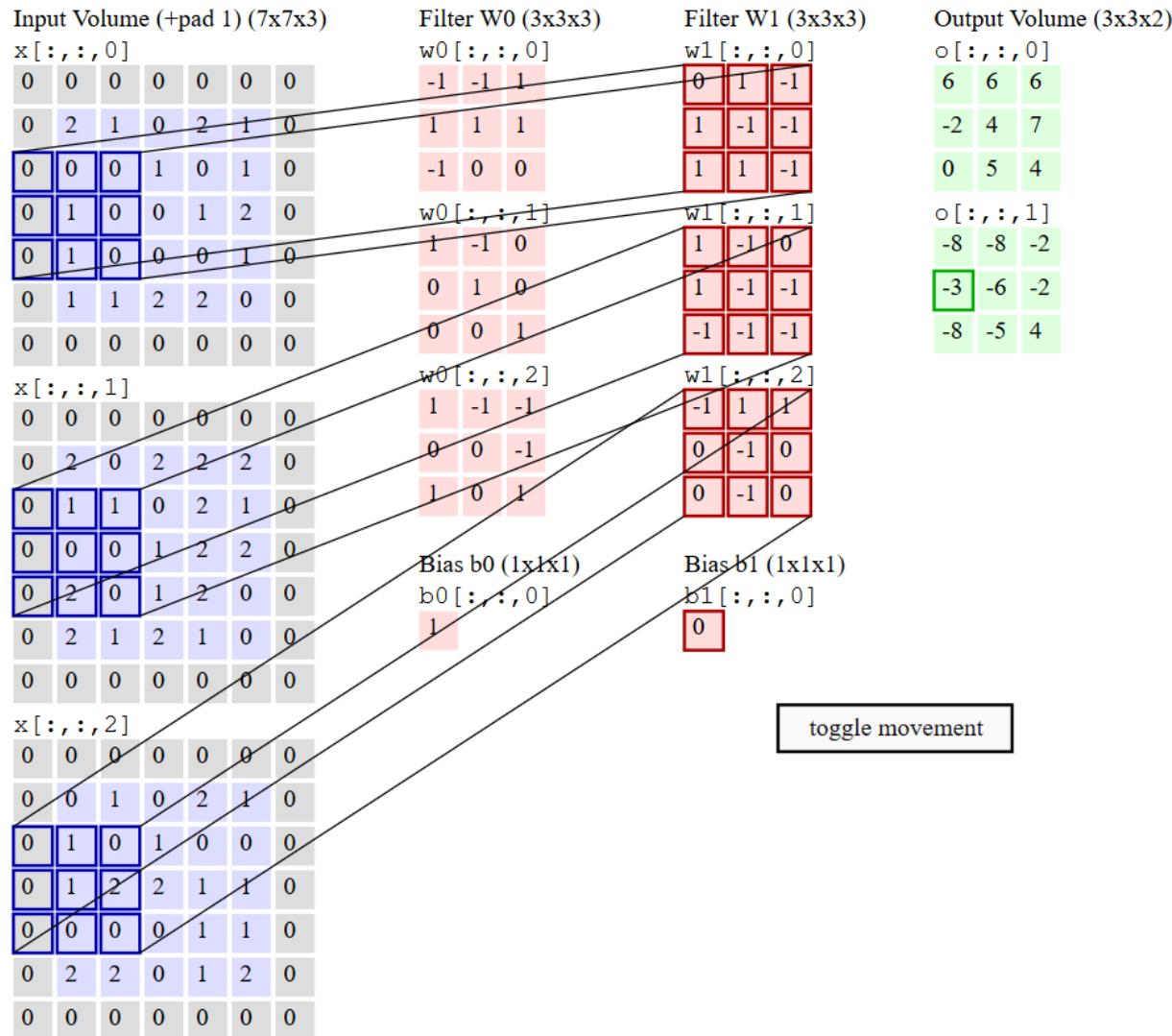
## این پیچش: دمو ...



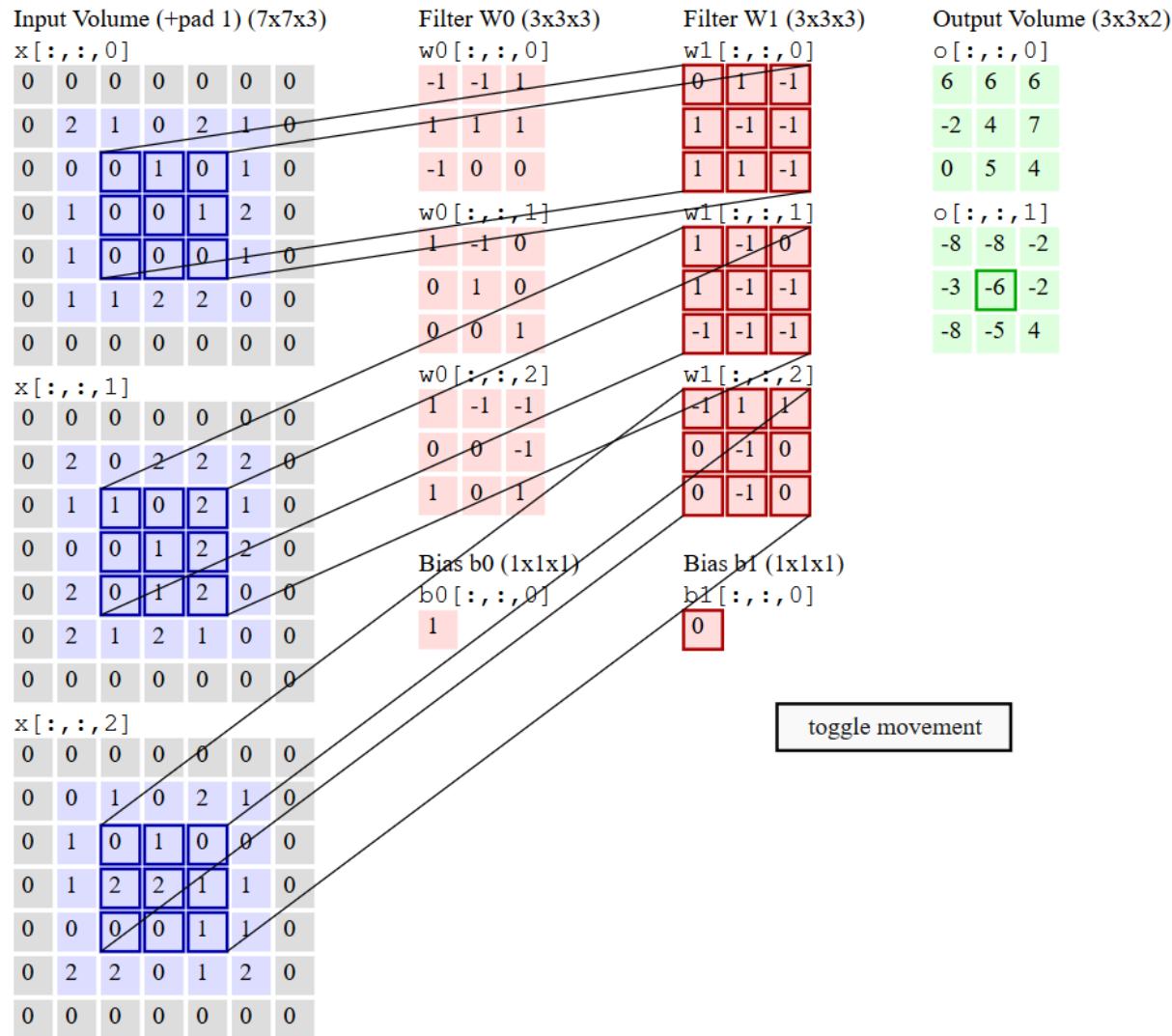
## نهاده پیچش: دمو ...



## این پیچش: دمو ...

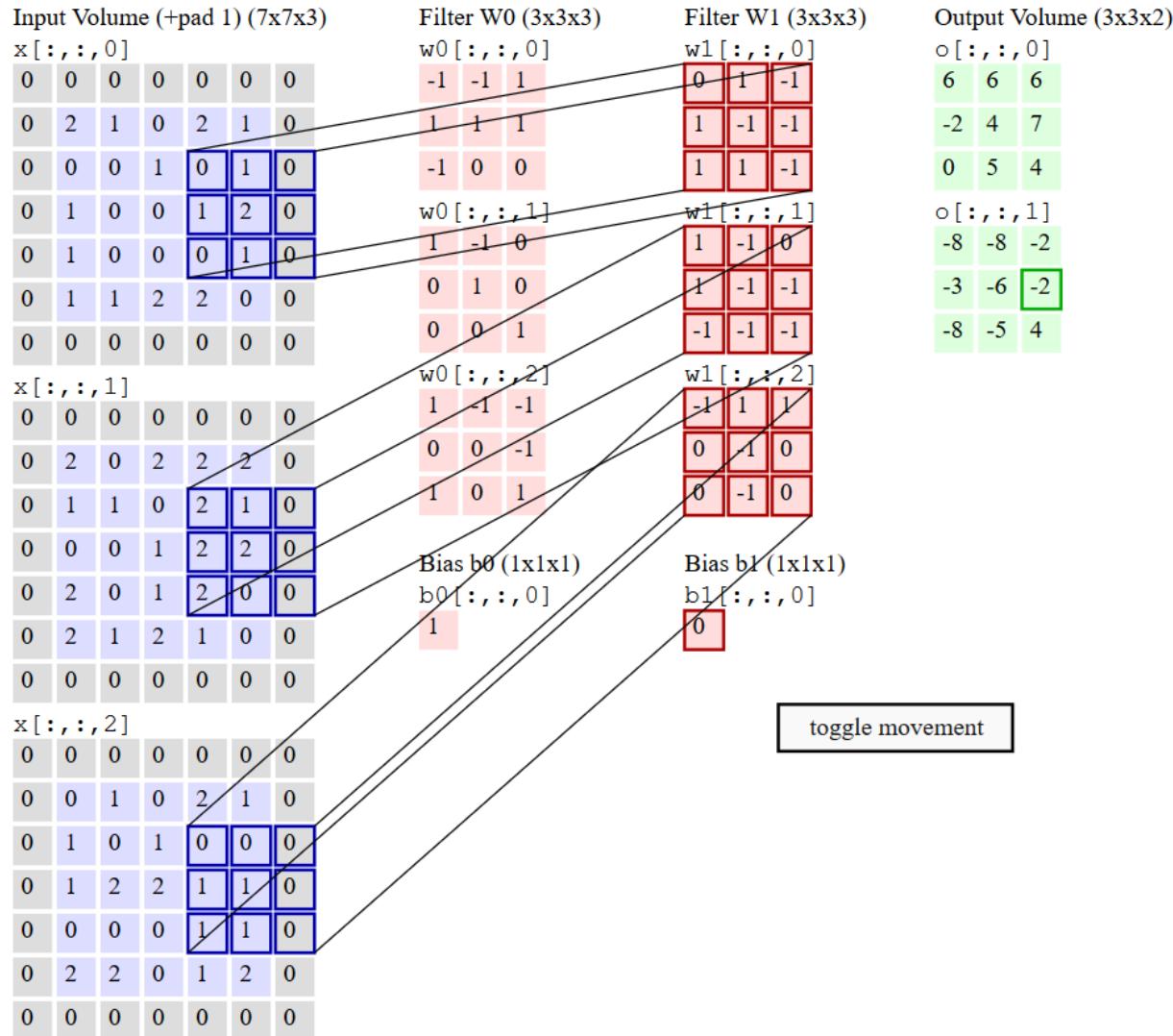


## این پیچش: دمو ...

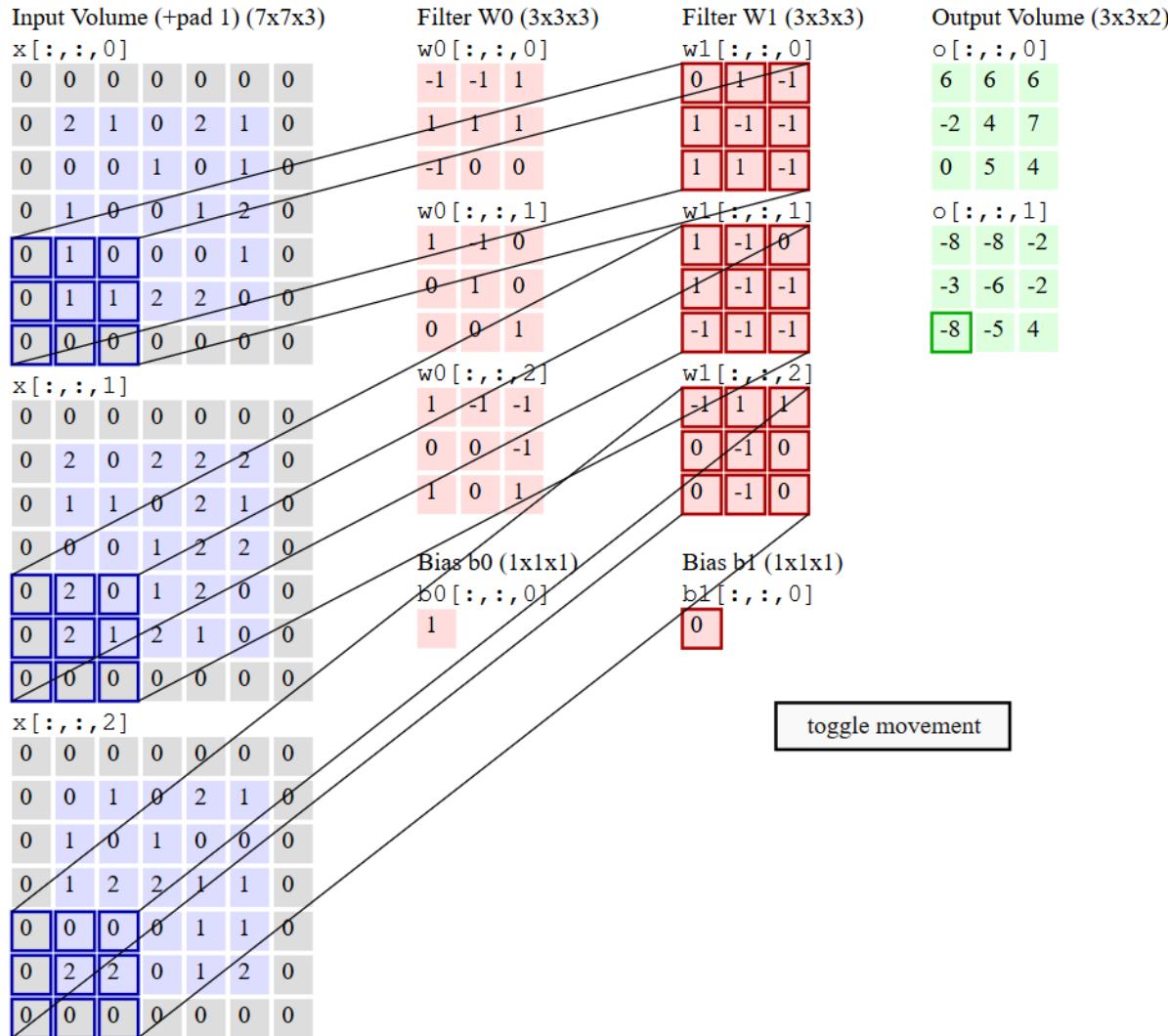




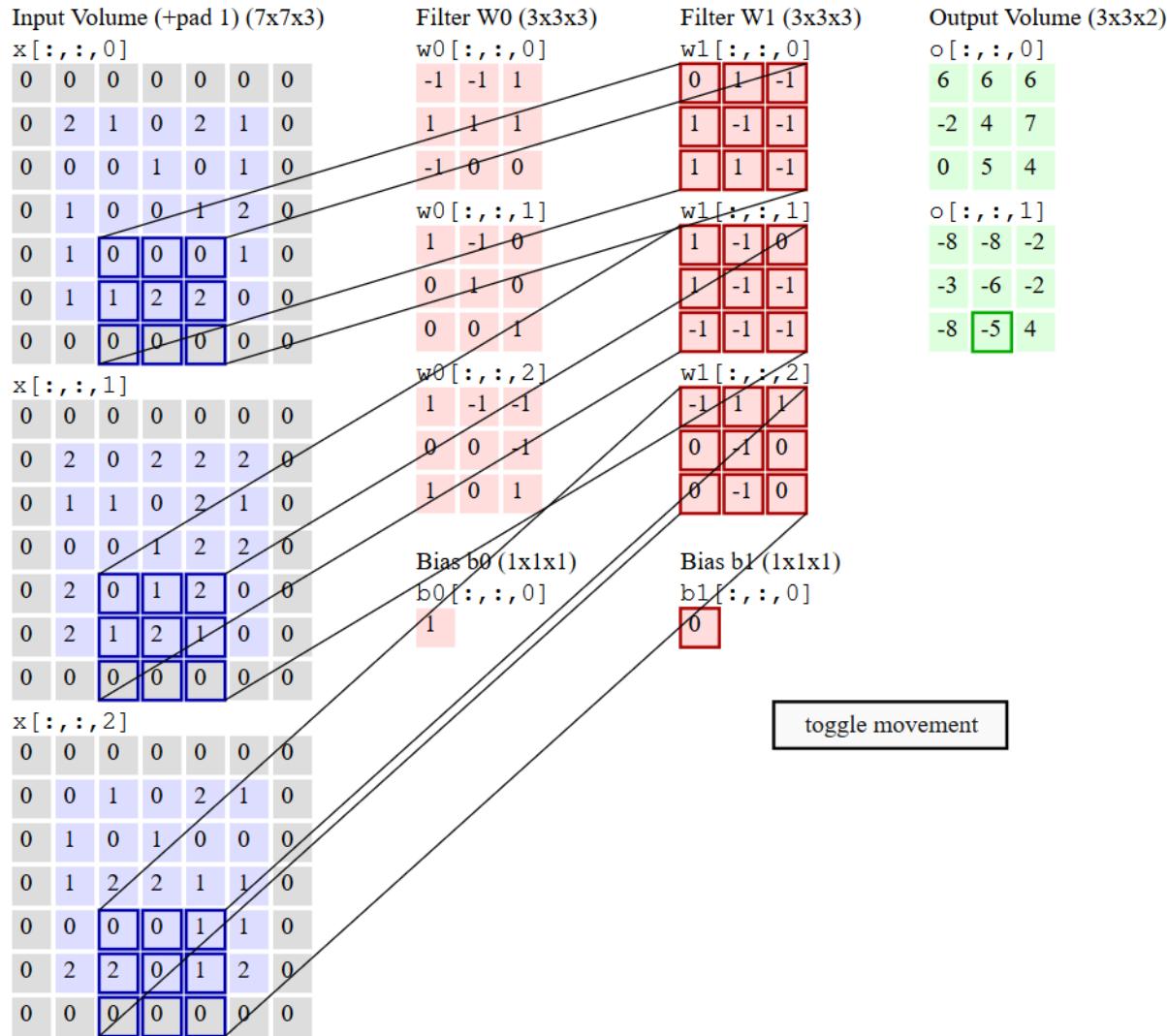
## این پیچش: دمو ...



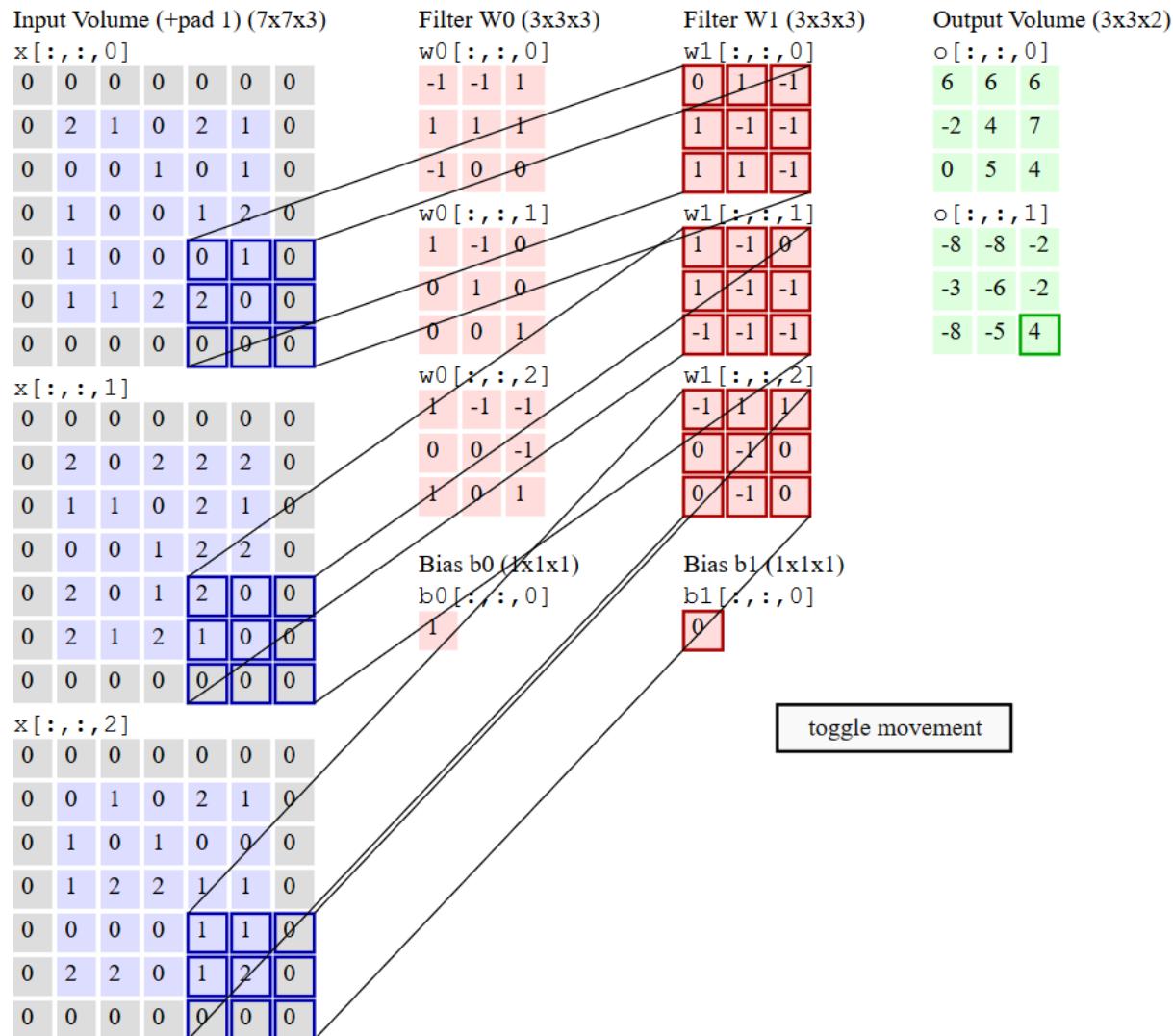
## این پیچش: دمو ...



## این پیچش: دمو ...



## این پیچش: دمو ...



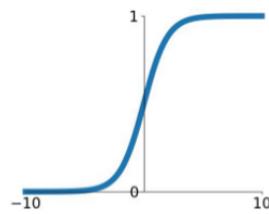
# ReLU پیچش

- یک لایه نیست !

- اعمال تابع فعال‌سازی روی خروجی لایه پیچش
  - افزودن ویژگی غیرخطی به شبکه

## Sigmoid

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



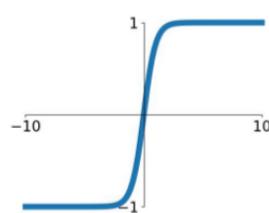
## جایگزین سیگموید

- سرعت محاسبه بالاتر نسبت به  $\exp$

- رفع مشکل اشباع gradient (تا حدودی)

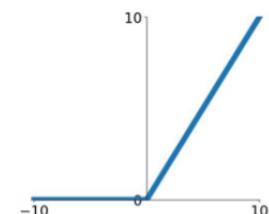
## tanh

$$\tanh(x)$$



## ReLU

$$\max(0, x)$$

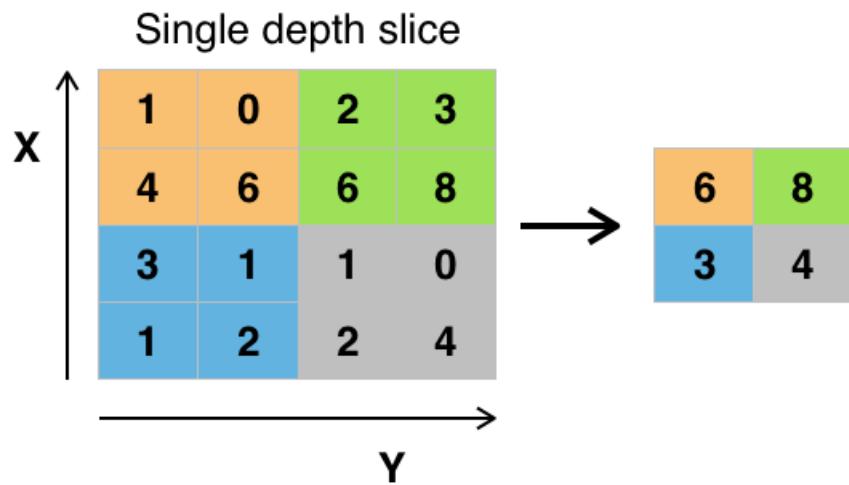


## لایه نمونه‌برداری (Pooling) ...

- استفاده بعد از لایه پیچش

- برای کاهش ابعاد تصویر
- برای کنترل بیش برازش

- اعمال یک فیلتر  $2 \times 2$  با گام ۲ برای بیشینه (Max) گرفتن
- کاهش اندازه تصویر به نصف





# نحوه نمونه‌برداری ... (Pooling)

## روش‌های کاهش ابعاد

- بیشینه (MAX): روش رایج
- میانگین (Average)
- L2 نرم

## پارامترها

$W_1 \times H_1 \times D_1$ : تصویر ورودی

$F$ : اندازه فیلتر

$S$ : اندازه گام

$W_2 \times H_2 \times D_2$ : تصویر خروجی

$$W_2 = (W_1 - F) / S + 1$$

$$H_2 = (H_1 - F) / S + 1$$

$$D_2 = D_1$$



## لایه نمونه‌برداری (Pooling)

- مقادیر رایج پارامترها

- (overlapping pooling F=3, S=2)

- (روش رایج) F=2, S=2

- عدم استفاده از لایه نمونه‌برداری

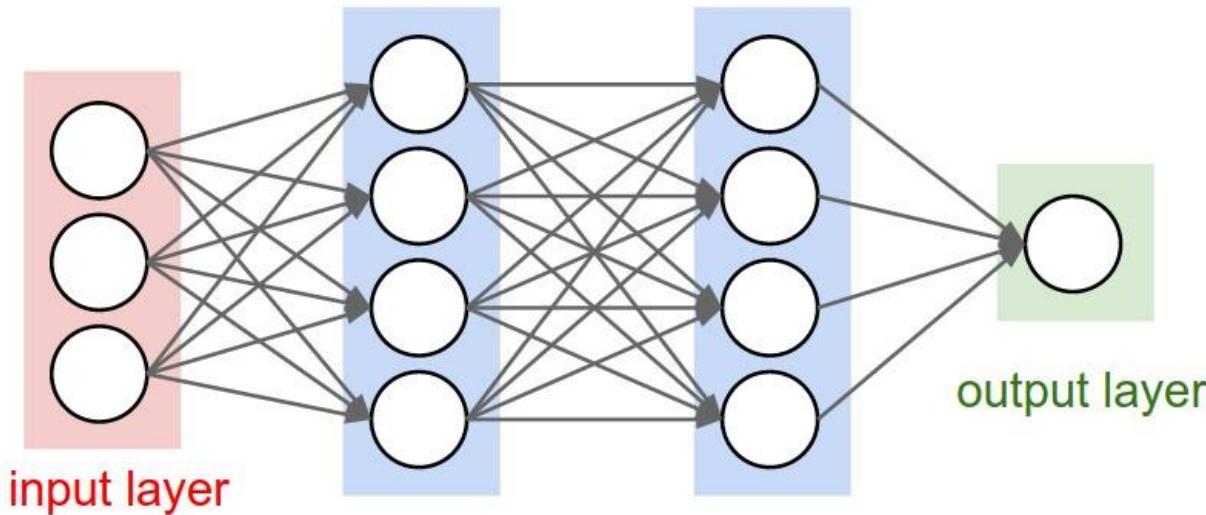
- در برخی موارد به جای استفاده از لایه نمونه‌برداری برای کاهش بعد، از همان لایه‌های پیچش با گام بزرگ استفاده می‌شود

- کارایی بهتر در آموزش مدل‌های تولیدی مانند variational autoencoders (VAEs) و generative adversarial networks (GANs)

# نمایه تمام متصل (Fully Connected)

استفاده به عنوان دسته بند

- ورودی: تصویر آرایه شده آخرین لایه نمونه برداری / پیچش
- خروجی: دسته های مختلف

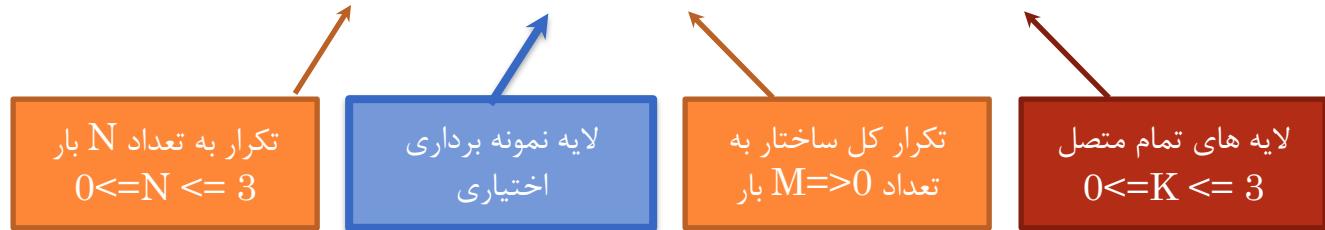


- کار کرد با توابع فعال سازی مختلف مانند سیگموید و SoftMax

# معماری شبکه پیچش

## ○ ساختار کلی شبکه

- INPUT  $\Rightarrow$  [[CONV  $\Rightarrow$  RELU] $^N$   $\Rightarrow$  POOL?]  $^M$   $\Rightarrow$  [FC  $\Rightarrow$  RELU] $^K$   $\Rightarrow$  FC



## ● مثال

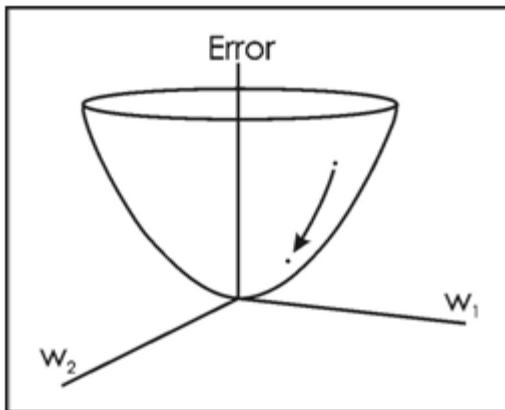
- INPUT  $\Rightarrow$  [CONV  $\Rightarrow$  RELU  $\Rightarrow$  CONV  $\Rightarrow$  RELU  $\Rightarrow$  POOL] $^3$   $\Rightarrow$  [FC  $\Rightarrow$  RELU] $^2$   $\Rightarrow$  FC

# آموزش شبکه پیچشی ...

## ○ مبتنی بر روش کاهش گرادیان

- کمینه کردن مربعات خطای انتشار خطا
- استفاده از روش پس انتشار خطای انتشار خطا

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$



## پس انتشار خطای ...

- لایه پیچش ...

- مرحله پیش رو - انجام پیچش

$$\begin{matrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{matrix} = \text{Convolution} \left( \begin{matrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{matrix}, \begin{matrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{matrix} \right)$$

فیلتر ۱۸۰ درجه چرخانده می شود

$$O_{11} = F_{22}X_{11} + F_{21}X_{12} + F_{12}X_{21} + F_{11}X_{22}$$

$$O_{12} = F_{22}X_{12} + F_{21}X_{13} + F_{12}X_{22} + F_{11}X_{23}$$

$$O_{21} = F_{22}X_{21} + F_{21}X_{22} + F_{12}X_{31} + F_{11}X_{32}$$

$$O_{22} = F_{22}X_{22} + F_{21}X_{23} + F_{12}X_{32} + F_{11}X_{33}$$



## پس انتشار خطای ...

### ○ لایه پیچش ...

- مرحله پیش رو - انجام پیچش
- مرحله پس انتشار - ۱ - محاسبه تغییرات وزن ها

$$\frac{\partial E}{\partial F_{22}} = \frac{\partial E}{\partial O_{11}} X_{11} + \frac{\partial E}{\partial O_{12}} X_{12} + \frac{\partial E}{\partial O_{21}} X_{21} + \frac{\partial E}{\partial O_{22}} X_{22}$$

$$\frac{\partial E}{\partial F_{21}} = \frac{\partial E}{\partial O_{11}} X_{12} + \frac{\partial E}{\partial O_{12}} X_{13} + \frac{\partial E}{\partial O_{21}} X_{22} + \frac{\partial E}{\partial O_{22}} X_{23}$$

$$\frac{\partial E}{\partial F_{12}} = \frac{\partial E}{\partial O_{11}} X_{21} + \frac{\partial E}{\partial O_{12}} X_{22} + \frac{\partial E}{\partial O_{21}} X_{31} + \frac{\partial E}{\partial O_{22}} X_{32}$$

$$\frac{\partial E}{\partial F_{11}} = \frac{\partial E}{\partial O_{11}} X_{22} + \frac{\partial E}{\partial O_{12}} X_{23} + \frac{\partial E}{\partial O_{21}} X_{32} + \frac{\partial E}{\partial O_{22}} X_{33}$$

$$\begin{array}{|c|c|} \hline \partial E / \partial F_{11} & \partial E / \partial F_{12} \\ \hline \partial E / \partial F_{21} & \partial E / \partial F_{22} \\ \hline \end{array} = \text{Convolution} \left( \begin{array}{|c|c|c|} \hline X_{11} & X_{12} & X_{13} \\ \hline X_{21} & X_{22} & X_{23} \\ \hline X_{31} & X_{32} & X_{33} \\ \hline \end{array}, \begin{array}{|c|c|} \hline \partial E / \partial O_{11} & \partial E / \partial O_{12} \\ \hline \partial E / \partial O_{21} & \partial E / \partial O_{22} \\ \hline \end{array} \right)$$



## پس انتشار خطای ...

$$\frac{\partial E}{\partial X_{11}} = \frac{\partial E}{\partial O_{11}} F_{22} + \frac{\partial E}{\partial O_{12}} F_{12} + \frac{\partial E}{\partial O_{21}} F_{21} + \frac{\partial E}{\partial O_{22}} F_{11}$$

$$\frac{\partial E}{\partial X_{12}} = \frac{\partial E}{\partial O_{11}} F_{21} + \frac{\partial E}{\partial O_{12}} F_{22} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{13}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} F_{21} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{21}} = \frac{\partial E}{\partial O_{11}} F_{12} + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} F_{22} + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{22}} = \frac{\partial E}{\partial O_{11}} F_{11} + \frac{\partial E}{\partial O_{12}} F_{12} + \frac{\partial E}{\partial O_{21}} F_{21} + \frac{\partial E}{\partial O_{22}} F_{22}$$

$$\frac{\partial E}{\partial X_{23}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} F_{11} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} F_{21}$$

$$\frac{\partial E}{\partial X_{31}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} F_{12} + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{32}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} F_{11}$$

$$\frac{\partial E}{\partial X_{33}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} F_{11}$$

$\partial E / \partial X_{11}$	$\partial E / \partial X_{12}$	$\partial E / \partial X_{13}$
$\partial E / \partial X_{21}$	$\partial E / \partial X_{22}$	$\partial E / \partial X_{23}$
$\partial E / \partial X_{31}$	$\partial E / \partial X_{32}$	$\partial E / \partial X_{33}$

$$= \text{Full_Convolution} \left( \begin{array}{|c|c|} \hline \partial E / \partial O_{11} & \partial E / \partial O_{12} \\ \hline \partial E / \partial O_{21} & \partial E / \partial O_{22} \\ \hline \end{array}, \begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{21} & F_{22} \\ \hline \end{array} \right)$$

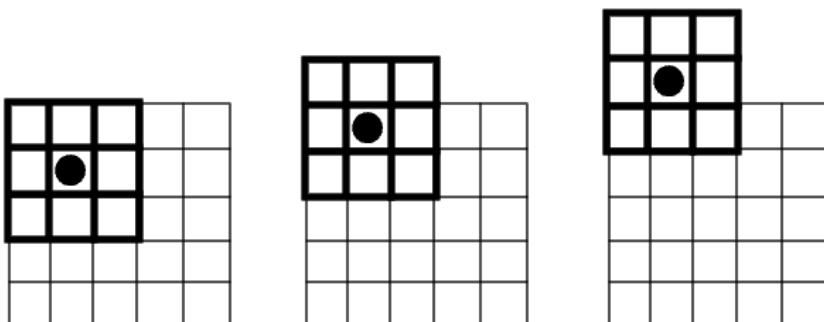
## پس انتشار خطای ...

### ○ لایه پیچش ...

- مرحله پیش رو - انجام پیچش
- مرحله پس انتشار - ۱ - محاسبه تغییرات وزن ها
- مرحله پس انتشار - ۱ - محاسبه تغییرات خروجی های مرحله قبل (ورودی این مرحله)

$$\begin{array}{|c|c|c|} \hline \partial E / \partial X_{11} & \partial E / \partial X_{12} & \partial E / \partial X_{13} \\ \hline \partial E / \partial X_{21} & \partial E / \partial X_{22} & \partial E / \partial X_{23} \\ \hline \partial E / \partial X_{31} & \partial E / \partial X_{32} & \partial E / \partial X_{33} \\ \hline \end{array} = \text{Full Convolution} \left( \begin{array}{|c|c|} \hline \partial E / \partial O_{11} & \partial E / \partial O_{12} \\ \hline \partial E / \partial O_{21} & \partial E / \partial O_{22} \\ \hline \end{array}, \begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{21} & F_{22} \\ \hline \end{array} \right)$$

تصویر زیر عملگر Full Convolution را نمایش می‌دهد





## پس انتشار خطای... .

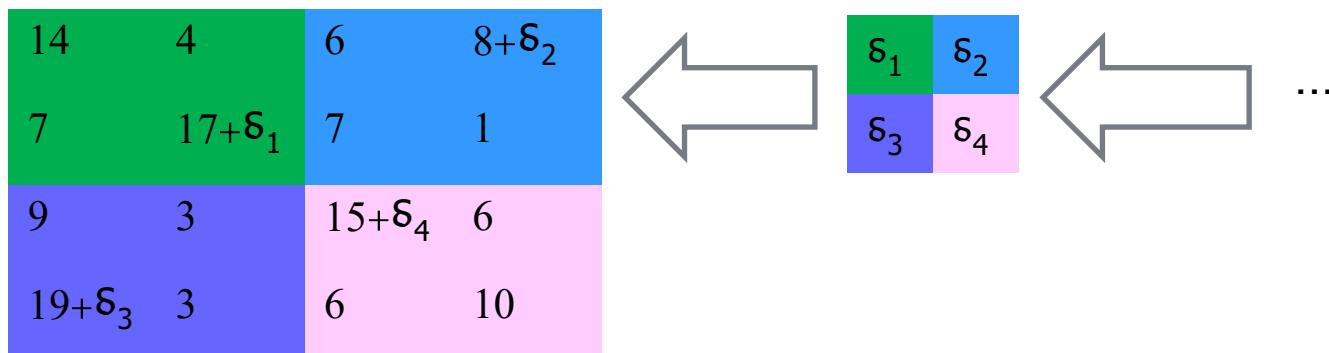
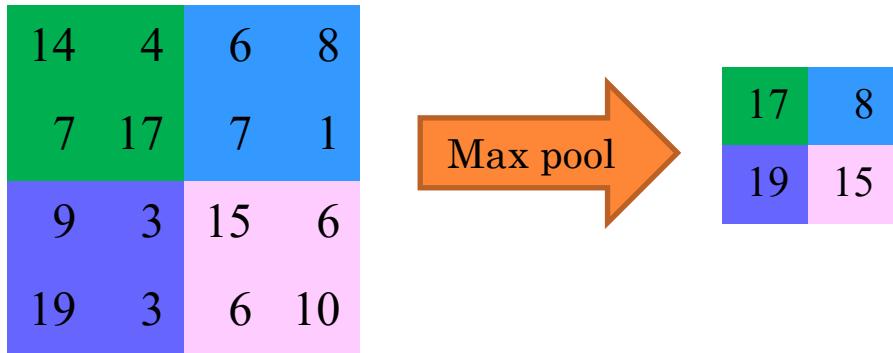
### ○ لایه پیچش

- مرحله پیش رو – انجام پیچش
- مرحله پس انتشار – ۱ - محاسبه تغییرات وزن ها
- مرحله پس انتشار – ۲ - محاسبه تغییرات خروجی های مرحله قبل (ورودی این مرحله)
- به روز رسانی وزن ها با مقادیر بدست آمده در دو مرحله قبل
- این به روز رسانی را می توان از پس انتشار خطای در تمامی لایه های شبکه نیز انجام داد اما برای کاهش میزان حافظه استفاده شده می توانیم در همین مرحله آن را اعمال نماییم

## پس انتشار خطای ...

لایه نمونه برداری ...

Max Pool •

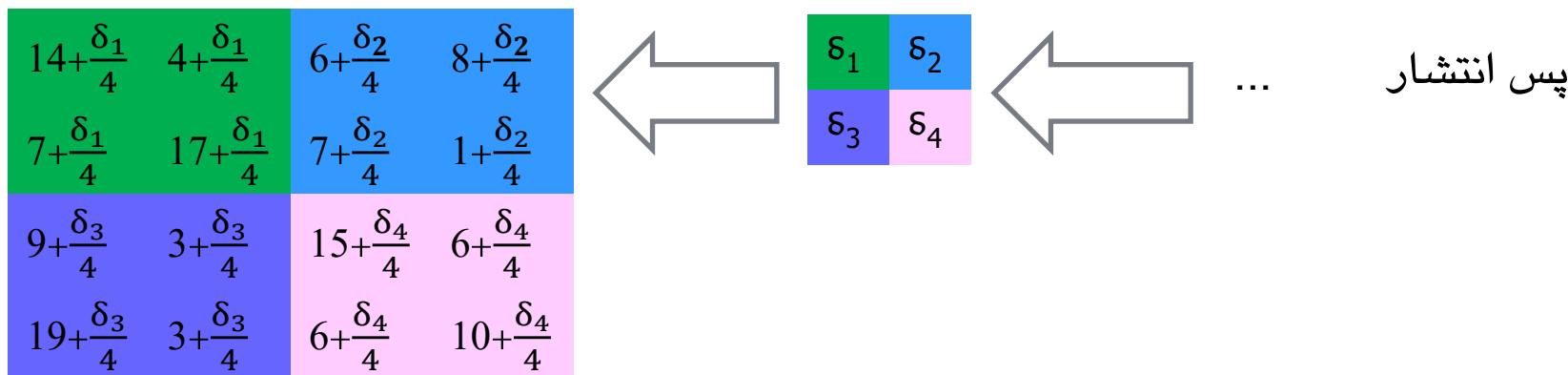
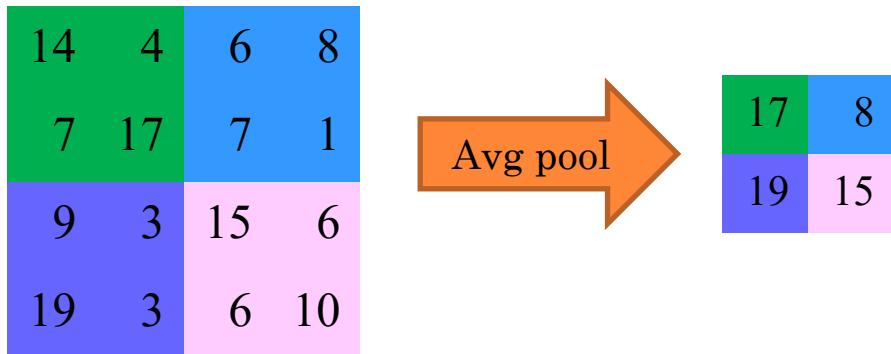


## پس انتشار خطای ...

لایه نمونه برداری

- Max Pool

- Average Pooling





## پس انتشار خطای ...

- لایه تمام متصل - سیگموید
- مشابه اسلاید شبکه عصبی - MLP

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

ReLU

$$\frac{df}{dx} = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

$$-2 \xrightarrow{\text{Relu}} 0$$

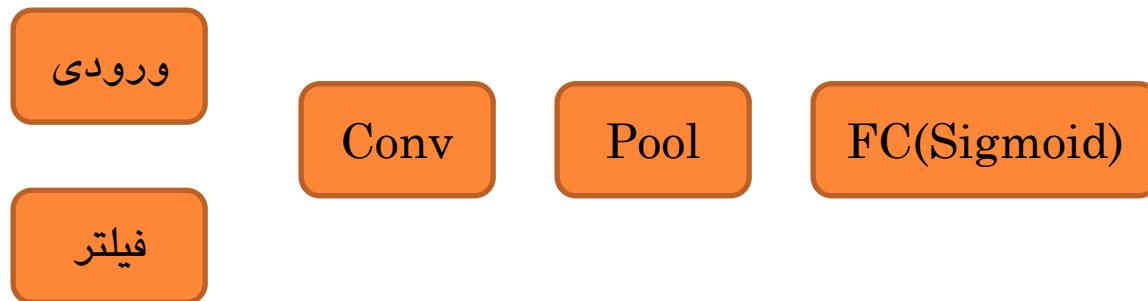
$$-2 + 0 * 8 \leftarrow 8 \quad \text{پس انتشار}$$

$$5 \xrightarrow{\text{Relu}} 5$$

$$5 + 1 * 8 \leftarrow 8 \quad \text{پس انتشار}$$

## مثال . . .

### ○ ساختار شبکه



## مثال . . .

Padding = 0, stride = 1

### پیش خور ورودی‌ها

0.1	0.2	0.3
0.4	0.5	0.6
0.7	0.8	0.9



1.3	1.7
2.5	2.9



2.9

FC - sigmoid

0.947

نتیجه  
اصلی

1	1
2	0



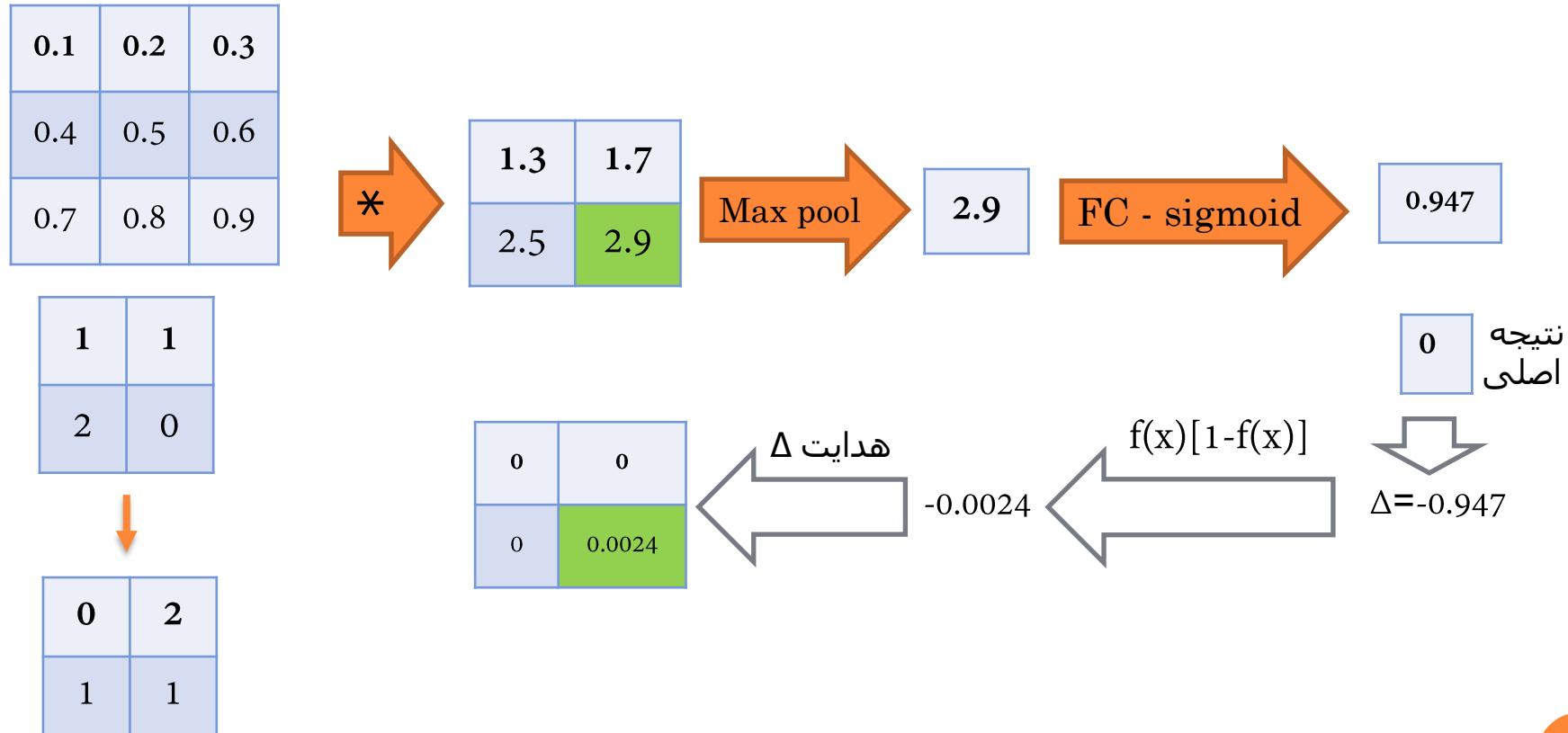
0	2
1	1

# مثال . . .

Padding = 0, stride = 1

## پس انتشار - سیگموید و ReLU

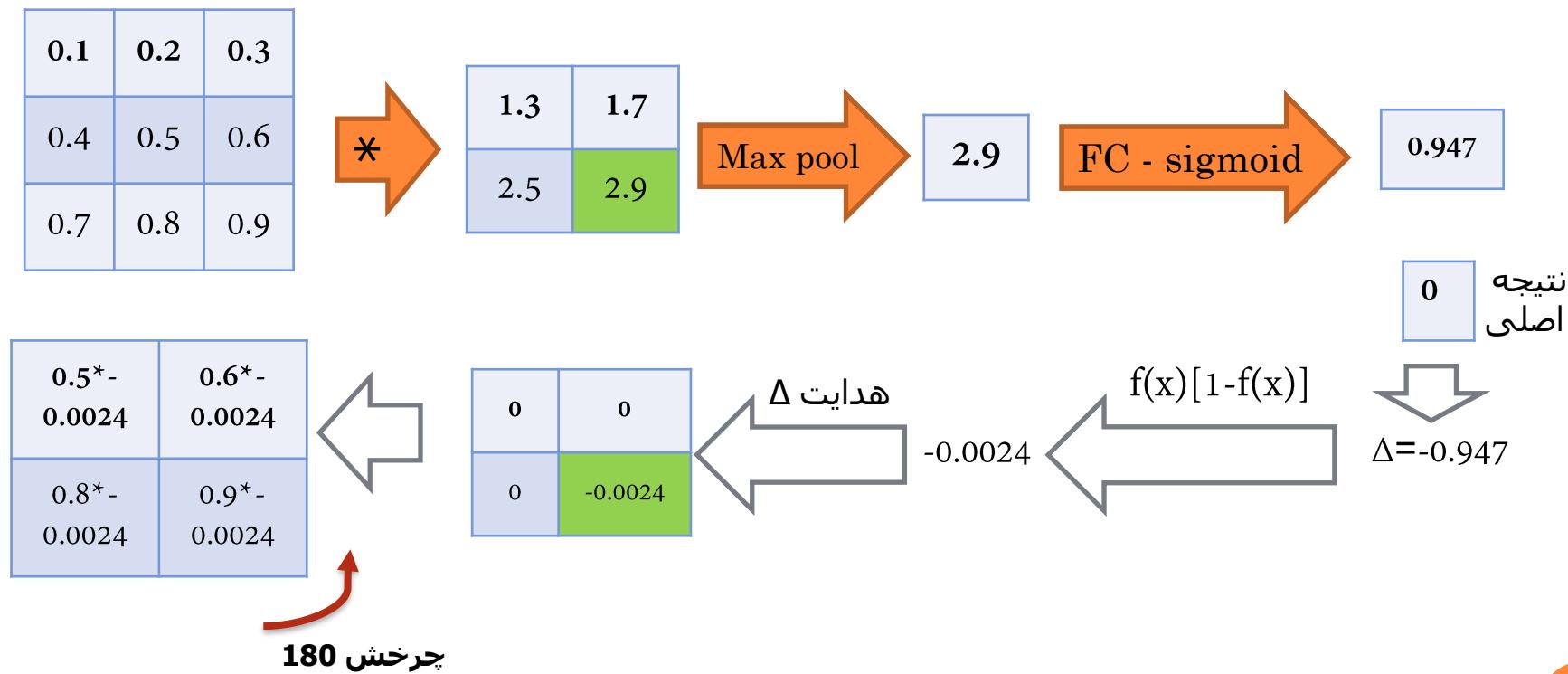
- تابع محاسبه خطا برای سادگی تفاضل در نظر گرفته شده است



مثال . . .

Padding = 0, stride = 1

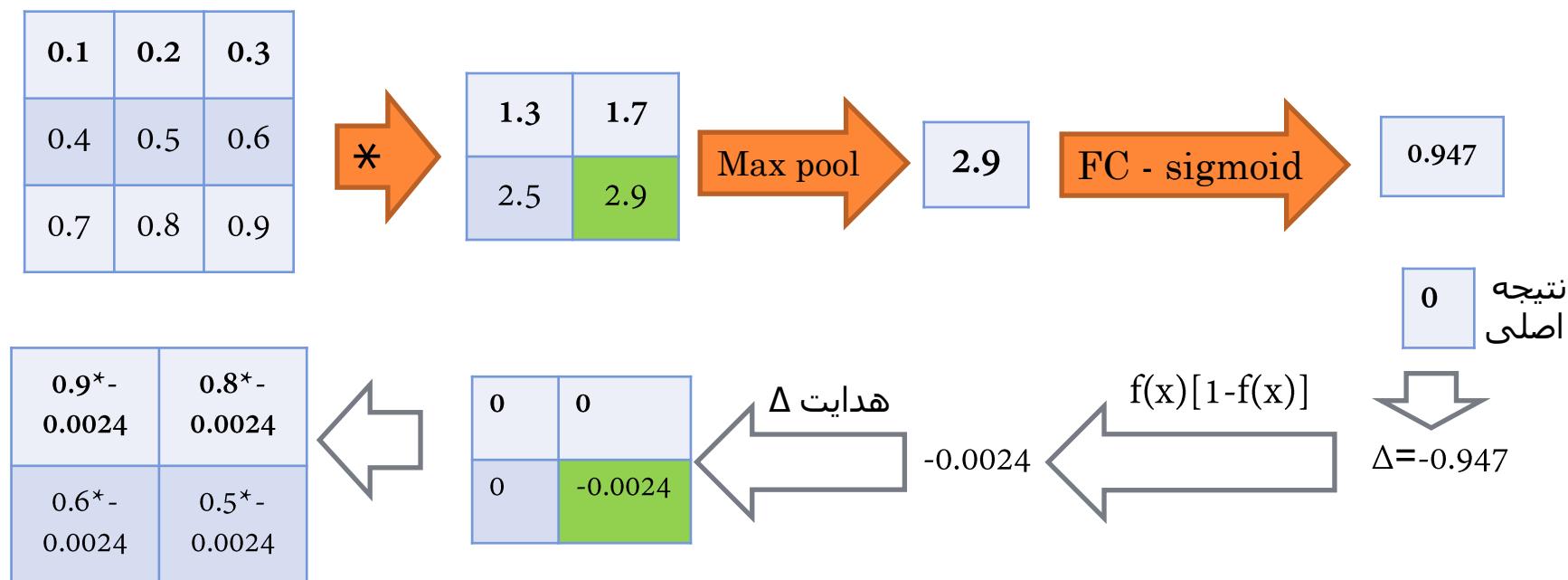
## پس انتشار - پیچش - محاسبه تغییرات وزن ها



مثال . . .

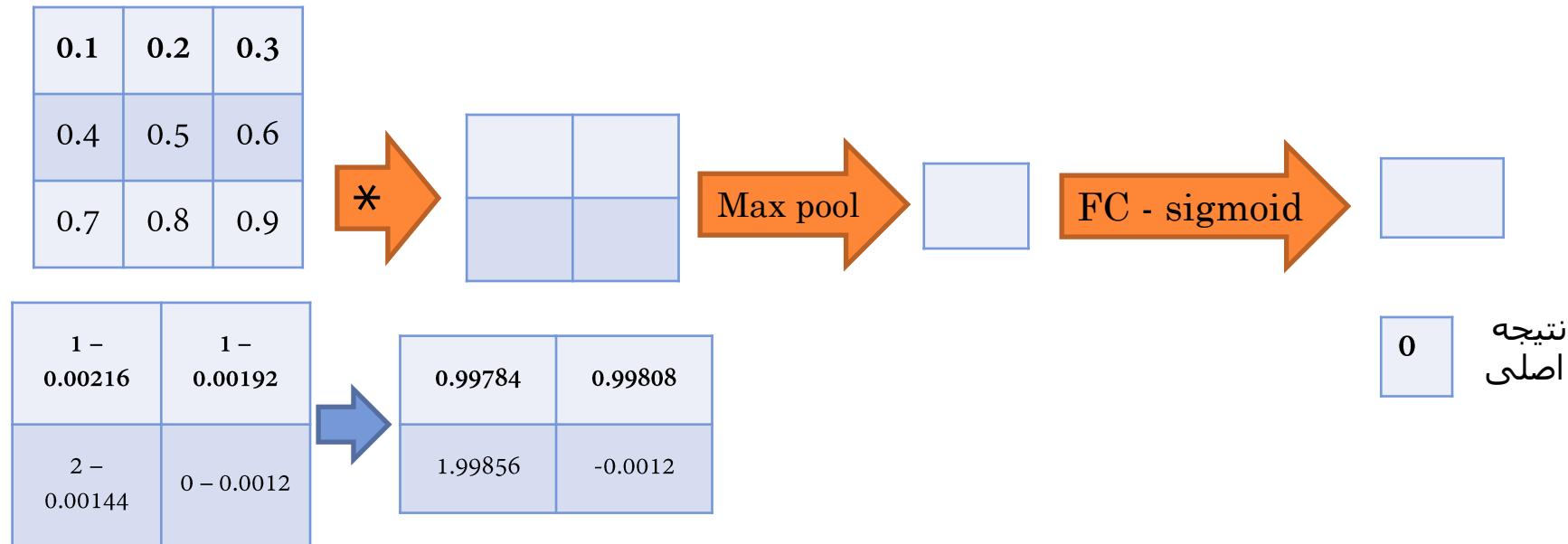
Padding = 0, stride = 1

## پس انتشار - پیچش - محاسبه تغییرات وزن ها



Padding = 0, stride = 1

## ○ پس انتشار - پیچش - اعمال تغییرات وزن‌ها



# معماری‌های شبکه عصبی پیچش

## ○ معماری‌های رایج



### ImageNet Large Scale Visual Recognition Challenges

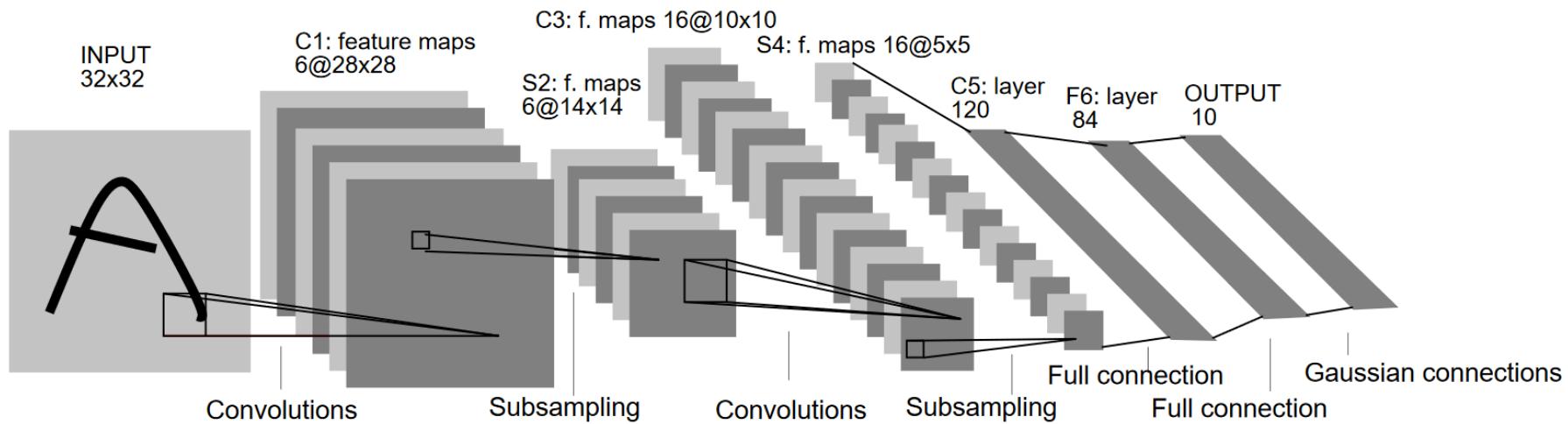


- روی دادگان ImageNet ([image-net.org](http://image-net.org))
- ۱.۲ میلیون تصویر در ۱۰۰۰ دسته

# شبکه LeNet

- ارائه در ۱۹۹۸
- کاربردهای اولیه مانند تشخیص دست نوشته حروف
- ساختار

Conv – Pool – Conv – Pool – Conv – FC •



LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-2324.

## شبکه AlexNet ...

### برنده ILSVRC 2012

- نرخ خطای ۱۵.۴٪ (حدود ۱۰٪ کمتر از برنده دوم)

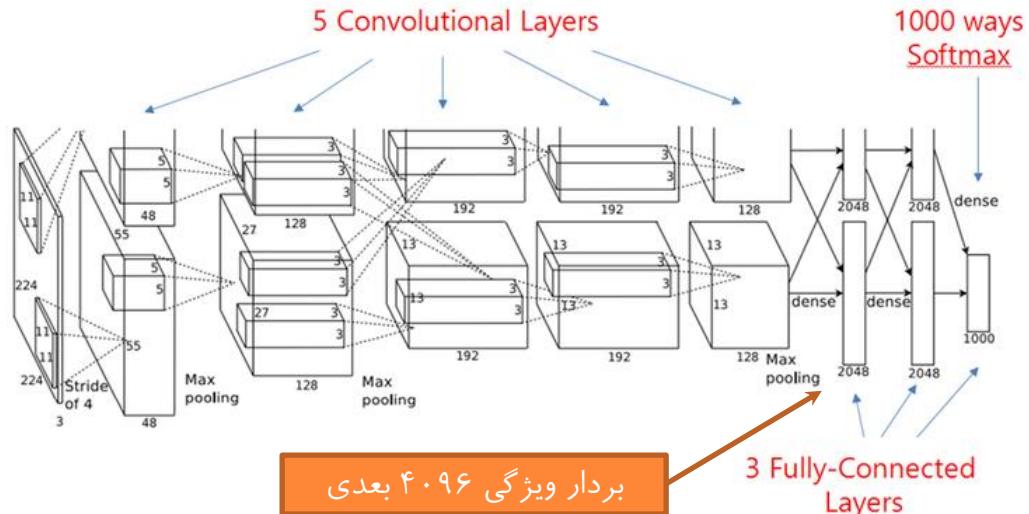
- شروع استفاده از ReLU

### ساختار

- عمیق‌تر از LeNet

- استفاده از چند لایه Conv متواالی برای اولین بار

Conv – Pool – Norm – Conv – Pool – Norm – Conv – Conv – Pool – FC – FC – FC

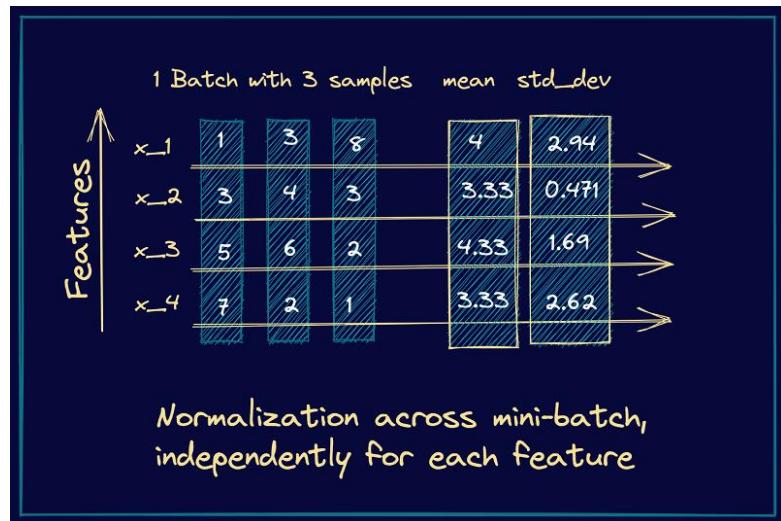


Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

# AlexNet شبکه

## ◦ نرمال‌سازی دسته‌ای (Batch Normalization)

- محاسبه میانگین و واریانس روی مقادیر ورودی نرون‌ها در هر لایه
- جمع بستن روی مقدار داده‌های ورودی یک دسته (هر نرون/بعد جداگانه)



$$\mu_b = \frac{1}{B} \sum_{i=1}^B x_i \quad (1)$$

$$\sigma_b^2 = \frac{1}{B} \sum_{i=1}^B (x_i - \mu_b)^2 \quad (2)$$

$$\hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2}} \quad (3)$$

$$\text{or } \hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \quad (3)$$

Adding  $\epsilon$  helps when  $\sigma_b^2$  is small

$$y_i = \mathcal{BN}(x_i) = \gamma \cdot x_i + \beta \quad (4)$$

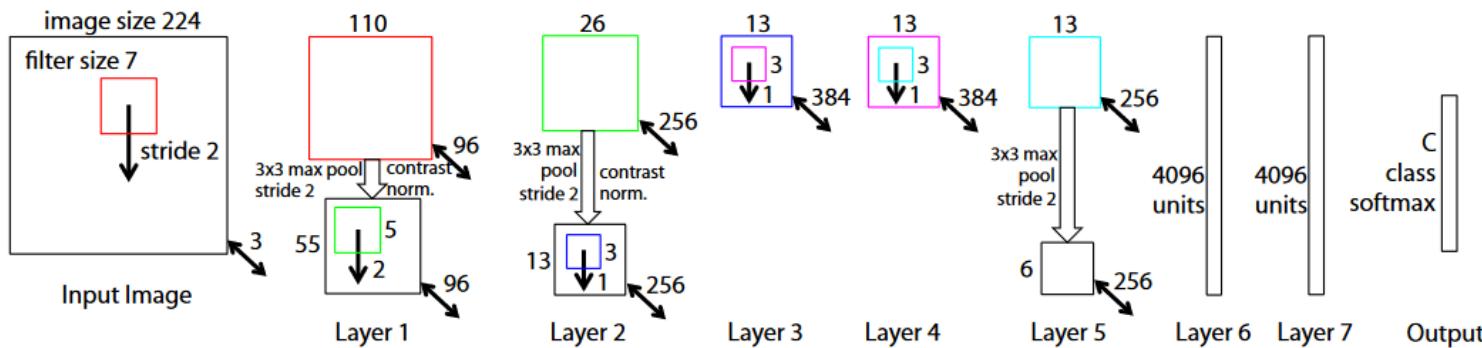
= تعداد نمونه داده‌های هر دسته  
= مقدار ورودی نرون ایام

گاما و بتا = پارامترهای قابل تنظیم در زمان آموزش برای ایجاد نوسان در ورودی (جلوگیری از الزام به صفر شدن همه ناشی از نرمال‌سازی)

## ZF شبکه

### ILSVRC 2013 برنده

- توسعه AlexNet با افزایش تعداد لایه‌های پیچش و کم کردن اندازه فیلتر و گام در لایه اول
- نرخ خطای %۱۴.۸



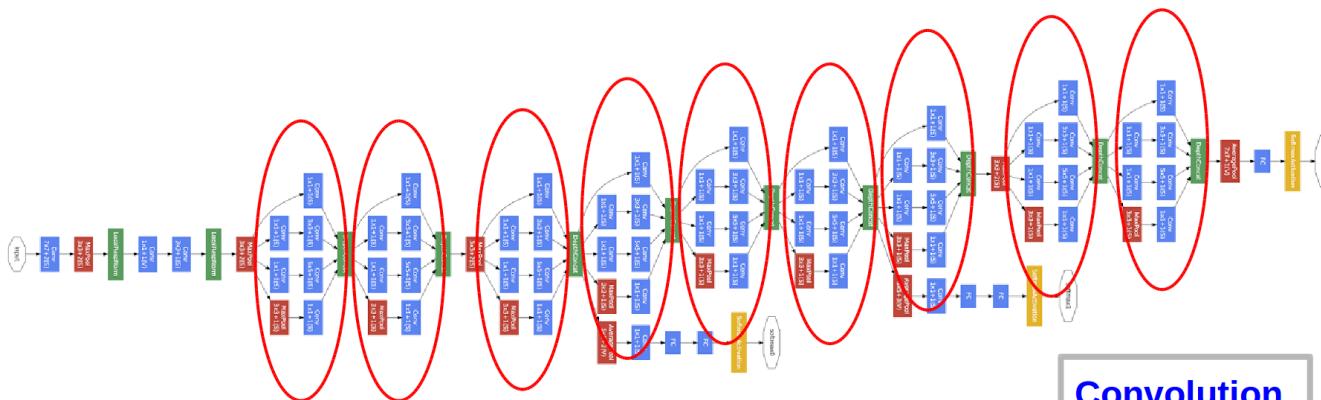
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on computer vision*, pp. 818-833. Springer, Cham, 2014.

# شبکه GoogleNet ...

برنده ILSVRC 2014

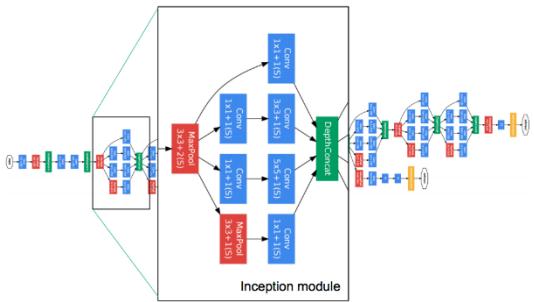
- ۲۲ لايه

- حذف لايه‌های FC متواالی و استفاده از Average Pooling
- کاهش چشمگیر پارامترها و عدم تاثير زیاد در کارایی
- نرخ خطای ۶.۷٪
- معرفی ماژول Inception
- کاهش تعداد پارامترها (حدود ۴ میلیون پارامتر در مقایسه با ۶۰ میلیون پارامتر AlexNet)



Convolution  
Pooling  
Softmax  
Concat/Normalize

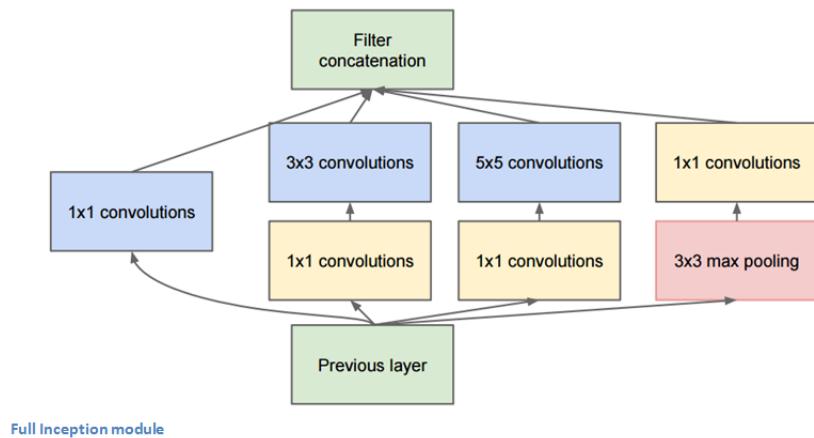
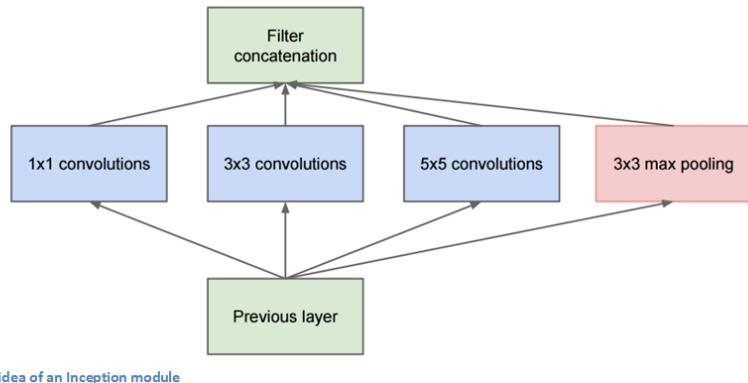
Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." Cvpr, 2015.



# شبکه GoogleNet

## ○ مازول Inception

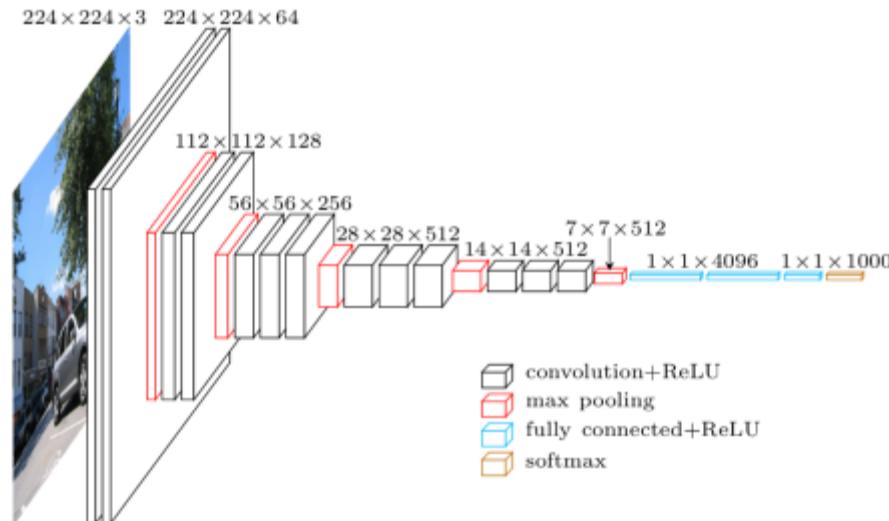
- این مازول اختیار انتخاب لایه‌ها برای رسیدن به بهترین نتیجه را به شبکه می‌دهد
- هر مازول یک Conv  $5 \times 5$ , Conv  $3 \times 3$  و یا صرفا Pooling است
- بدین معنا که تمام این بخش‌ها برای مازول بعدی به عنوان ورودی قابل استفاده است
- اینکه این موارد در کنار هم قرار دارند یعنی هر Inception module توانایی دیدن جزئیات را در سطح ریزدانه تا ویژگی‌های بزرگ در خود دارد.
- استفاده از Conv  $1 \times 1$  برای کاهش حجم در فضای ویژگی‌هاست



# شبکه VGGNet

برنده دوم ILSVRC 2014 (بعد از GoogleNet)

- نرخ خطای ۷.۳٪
- ۱۹ لایه
- لایه‌های پیچش ۳x3 و لایه‌های نمونه برداری 2x2

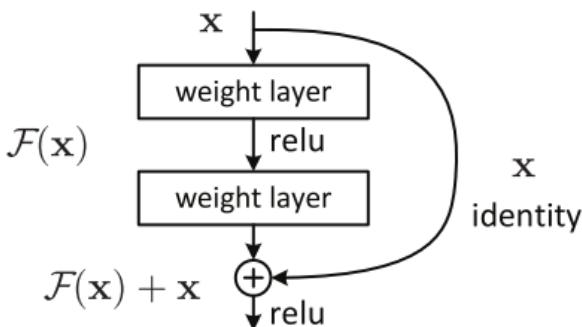
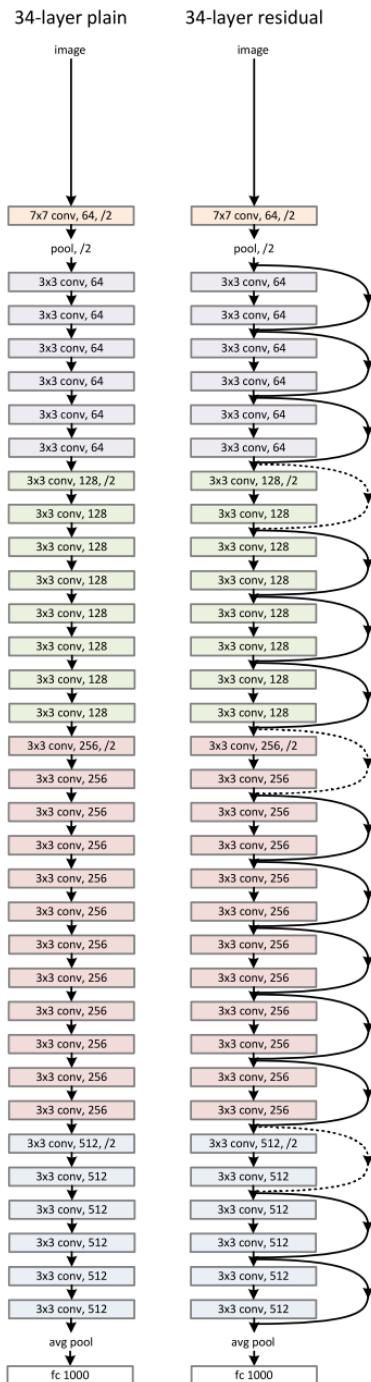


Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

# شبکه ResNet ...

برنده ILSVRC 2015

- ارائه توسط مایکروسافت
- ۱۵۲ لایه – بدون FC‌های متوالی
- نرخ خطای ۳.۶٪
- هر بلاک با قیمانده، شامل یک conv-relu-conv است که در آن به جای اینکه خروجی بدست آمده مستقیماً به مرحله بعد برود، به ورودی اصلی اضافه می‌شود.

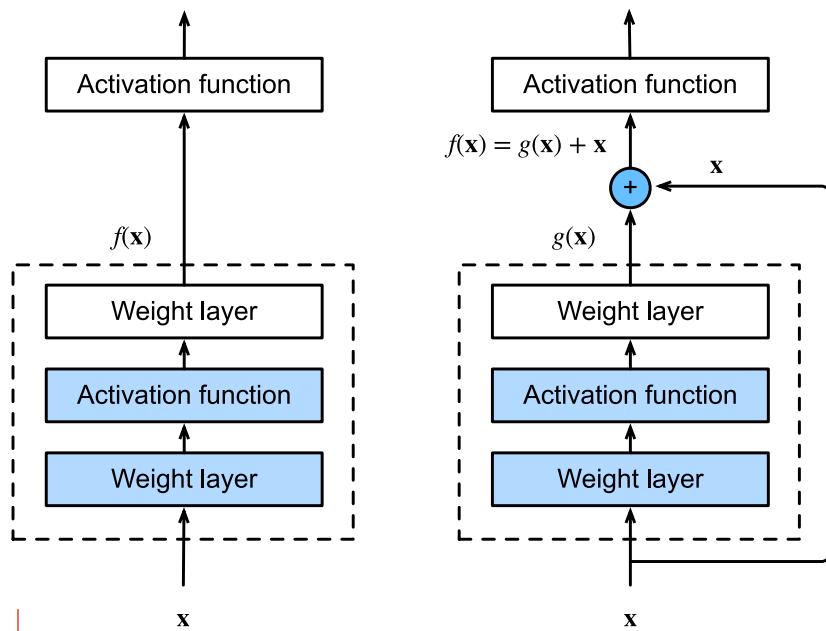


He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

# شبکه ResNet

## Bloc باقیمانده (Residuals)

- باقیمانده = مقدار تفاوت بین مقدار  $x$  و مقدار نگاشت
- اگر  $f(x)$  نگاشت مورد نظر باشد، آنگاه  $x - f(x) = g(x)$  مقدار باقیمانده است
- ایده واحد باقیمانده: یادگیری مقدار باقیمانده به جای خود نگاشت و بدست آوردن مقدار نگاشت از روی آن، یعنی  $f(x) = g(x) + x$



- کمک به یادگیری در تعداد لایه بالا
- اگر لایه اضافه شده مفید نباشد
- این لایه منجر به کاهش کارایی نمی‌شود
- کاهش اثر محو گرادیان
- مشکل degradation در شبکه‌های بسیار عمیق

- کاهش دقیق حتی بعد از تکرار خیلی زیاد

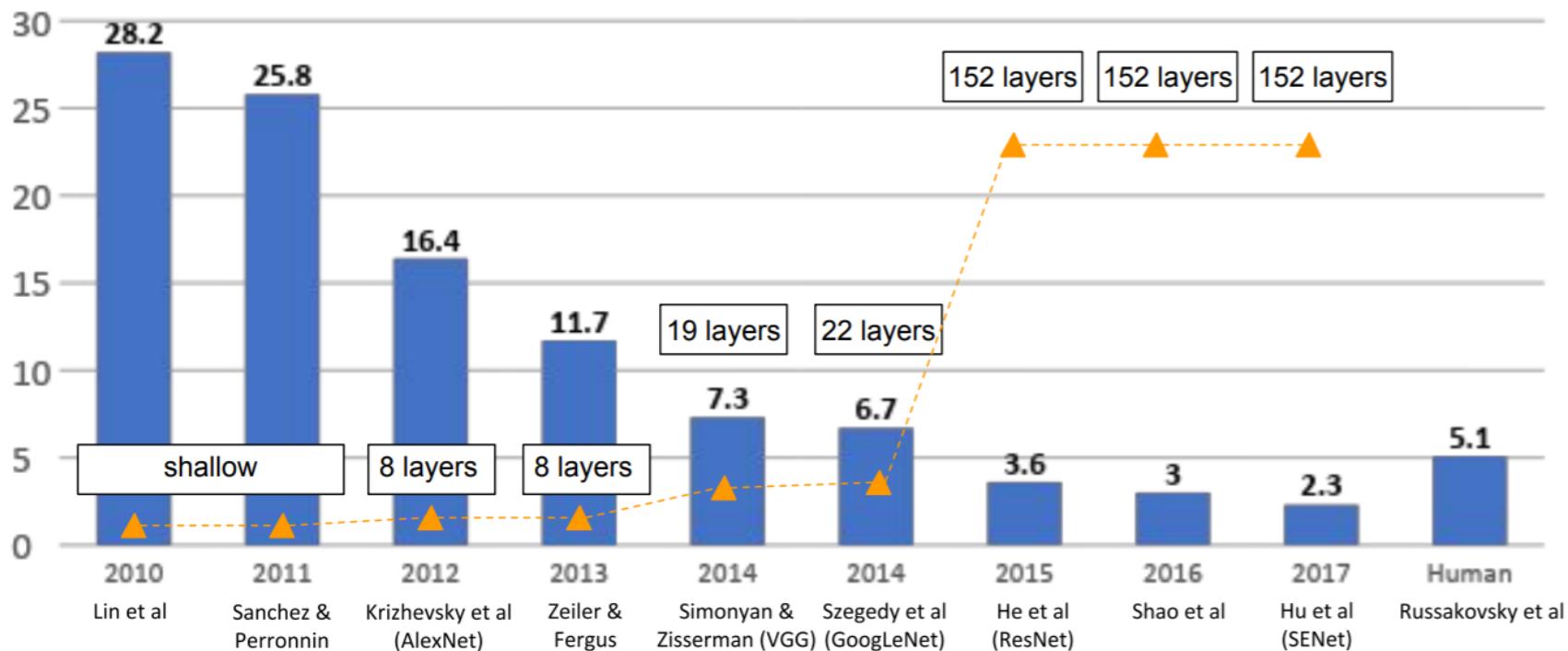
خطا روی ImageNet

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	<b>25.03</b>

# CNN مقایسه کارایی شبکه‌های

- نزدیک شدن کارایی شبکه پیچشی به عملکرد انسان

- نرخ خطای دسته‌بندی روی ImageNet





## دموی برخط

### ConvNetJS CIFAR-10 ◉

- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

# بسترهاي يادگيري عميق . . .

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	



# بسترهاي يادگيري عميق



## Framework Comparison: Basic information\*

Viewpoint	Torch.nn**	Theano***	Caffe	autograd (NumPy, Torch)	Chainer	MXNet	Tensor- Flow
GitHub stars	4,719	3,457	9,590	N: 654 T: 554	1,295	3,316	20,981
Started from	2002	2008	2013	2015	2015	2015	2015
Open issues/PRs	97/26	525/105	407/204	N: 9/0 T: 3/1	95/25	271/18	330/33
Main developers	Facebook, Twitter, Google, etc.	Université de Montréal	BVLC (U.C. Berkeley)	N: HIPS (Harvard Univ.) T: Twitter	Preferred Networks	DMLC	Google
Core languages	C/Lua	C/Python	C++	Python/Lua	Python	C++	C++/Python
Supported languages	Lua	Python	C++/Python MATLAB	Python/Lua	Python	C++/Python R/Julia/Go etc.	C++/Python

\* Data was taken on Apr. 12, 2016

\*\* Includes statistics of Torch7

\*\*\* There are many frameworks on top of Theano, though we omit them due to the space constraints