

Redes Neurais Profundas – Deep Learning

Aula 3 – Redes Neurais Convolucionais

Prof. Anderson Soares

www.inf.ufg.br/~anderson

www.deeplearningbrasil.com.br

Instituto de Informática

Universidade Federal de Goiás

Aula 3: Convolutional Neural Network



Agenda

- Contextualização: problema motivador
- Elementos de uma rede neural convolucional
- Codificação em tensorflow
- Codificação em Keras
- Problemas cuja técnica tem alto impacto
- Considerações finais

Mixed National Institute of Standards and Technology (MNIST)

- Dataset para o problema de reconhecimento de dígitos escritos a mão
- 60,000 imagens de treino, 10,000 de teste, codificados como imagens 28×28 em escala de cinza
- Quantas formas temos para escrever?

MNIST



MNIST

Classifier	Test Error Rate (%)	References
Linear classifier (1-layer neural net)	12.0	LeCun et al. (1998)
K-nearest-neighbors, Euclidean (L2)	5.0	LeCun et al. (1998)
2-Layer neural net, 300 hidden units, mean square error	4.7	LeCun et al. (1998)
Support vector machine, Gaussian kernel	1.4	MNIST Website
Convolutional net, LeNet-5 (no distortions)	0.95	LeCun et al. (1998)

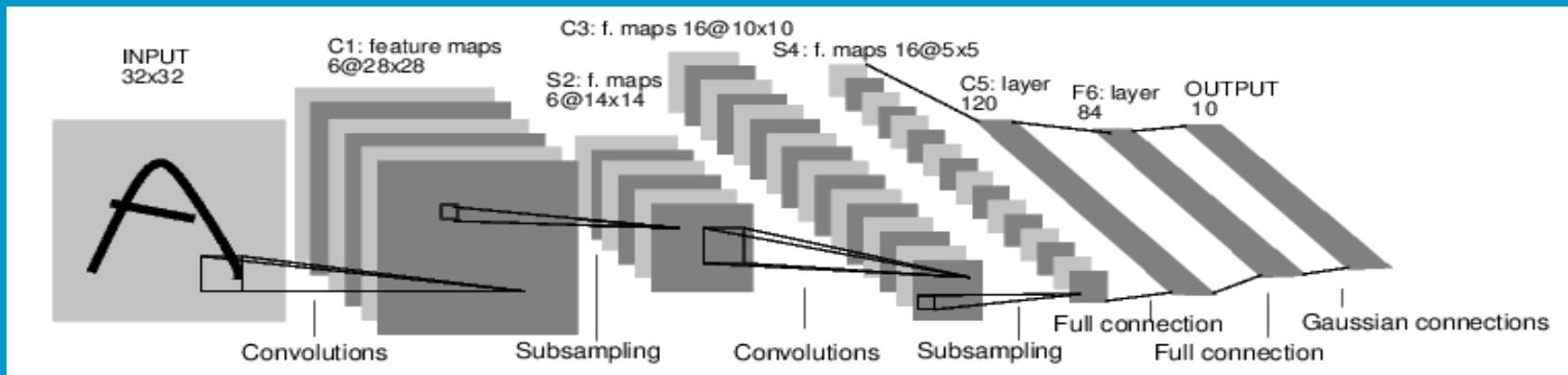
Methods using distortions

Virtual support vector machine, deg-9 polynomial, (2-pixel jittered and deskewing)	0.56	DeCoste and Scholkopf (2002)
Convolutional neural net (elastic distortions)	0.4	Simard, Steinkraus, and Platt (2003)
6-Layer feedforward neural net (on GPU) (elastic distortions)	0.35	Ciresan, Meier, Gambardella, and Schmidhuber (2010)
Large/deep convolutional neural net (elastic distortions)	0.35	Ciresan, Meier, Masci, Maria Gambardella, and Schmidhuber (2011)
Committee of 35 convolutional networks (elastic distortions)	0.23	Ciresan, Meier, and Schmidhuber (2012)



Yann LeCun, Professor of Computer Science
The Courant Institute of Mathematical Sciences
New York University
Room 1220, 715 Broadway, New York, NY 10003, USA.
(212)998-3283 yann@cs.nyu.edu

Yann LeCun e Yoshua Bengio Introduziram o conceito de redes convolucionais (1995).



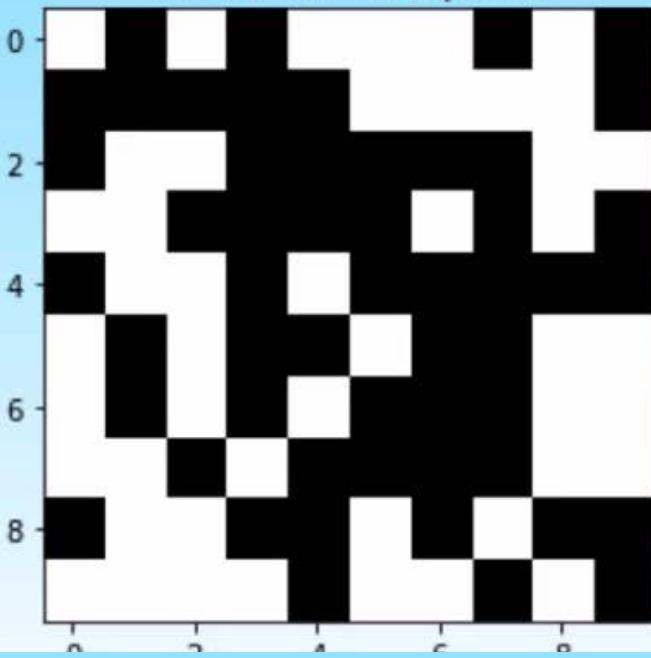
Rede Neural Convolucional

- CNN é um rede feed-forward que extrai propriedades topológicas a partir da imagem.
- Como qualquer outra rede neural a arquitetura é treinada com alguma variação do algoritmo back-propagation.
- Capazes de reconhecer padrões com muita variabilidade (Caracteres escritos a mão).

Rede Neural Convolucional

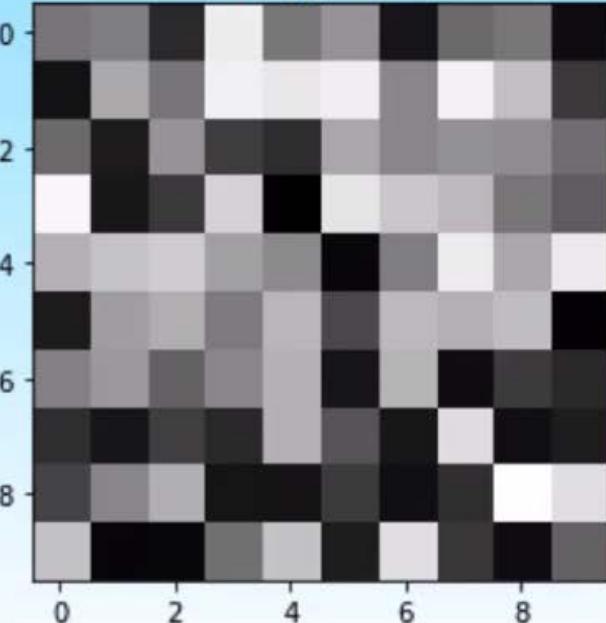
- CNN's foram inspiradas pela sensibilidade local e orientação seletiva do cérebo.
- Projetaram uma rede neural que implicitamente extraíram características relevantes da entrada.
- Redes convolucionais são um tipo especial de **multi-layer neural networks (MLP)**.

Black and White pixels



```
array([[1, 0, 1, 0, 1, 1, 1, 0, 1, 0],  
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 0],  
       [0, 1, 1, 0, 0, 0, 0, 0, 1, 1],  
       [1, 1, 0, 0, 0, 1, 0, 1, 0, 0],  
       [0, 1, 1, 0, 1, 0, 0, 0, 0, 0],  
       [1, 0, 1, 0, 0, 1, 0, 0, 1, 1],  
       [1, 0, 1, 0, 1, 0, 0, 0, 1, 1],  
       [1, 1, 0, 1, 0, 0, 0, 0, 1, 1],  
       [0, 1, 1, 0, 0, 1, 0, 1, 0, 0],  
       [1, 1, 1, 0, 1, 1, 0, 1, 0, 0]])
```

Grey pixels



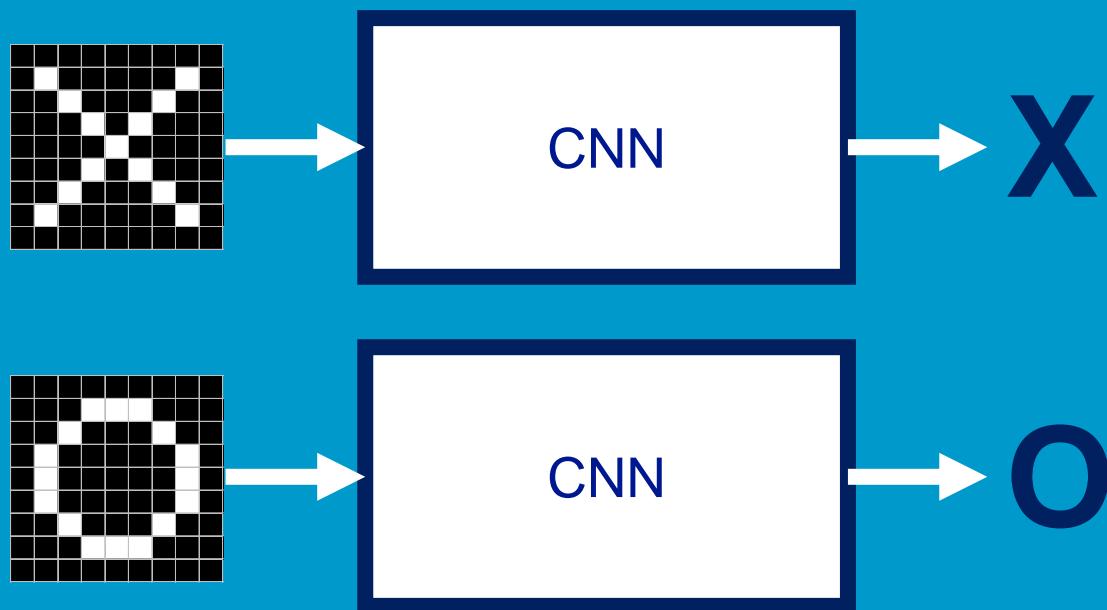
```
array([[127, 136, 48, 234, 128, 155, 24, 114, 128, 12],  
       [18, 177, 127, 239, 230, 236, 145, 240, 195, 65],  
       [116, 29, 155, 70, 51, 174, 144, 153, 149, 120],  
       [243, 26, 66, 213, 1, 227, 203, 191, 128, 105],  
       [183, 199, 207, 167, 146, 7, 136, 234, 175, 232],  
       [31, 166, 180, 133, 187, 82, 192, 182, 194, 2],  
       [139, 161, 108, 144, 183, 23, 185, 12, 67, 47],  
       [53, 24, 72, 46, 182, 94, 25, 221, 16, 34],  
       [76, 144, 181, 22, 21, 67, 16, 53, 250, 222],  
       [197, 6, 7, 122, 198, 34, 222, 64, 12, 108]])
```

Decidir se uma figura contém X ou O

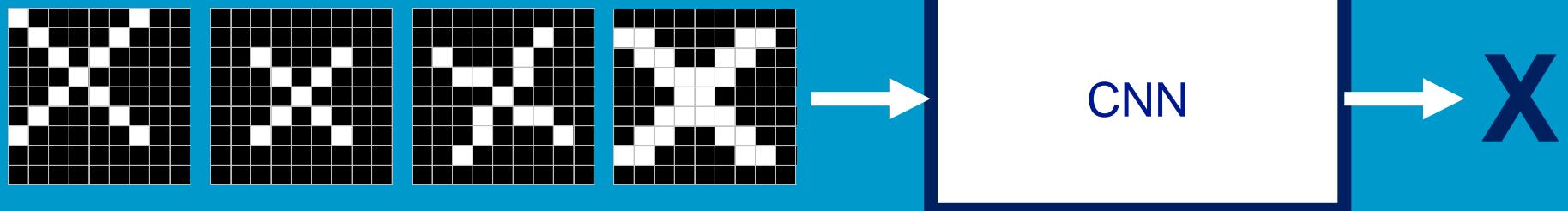
Matriz de pixels



Por exemplo:



Casos mais complicados

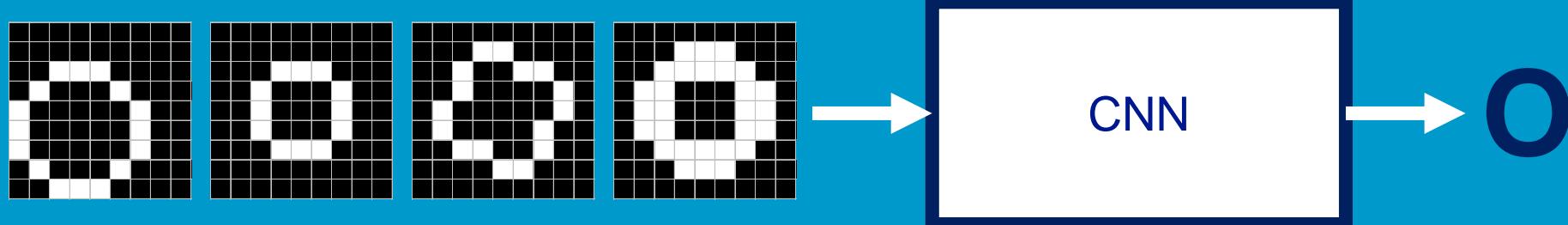


Translação

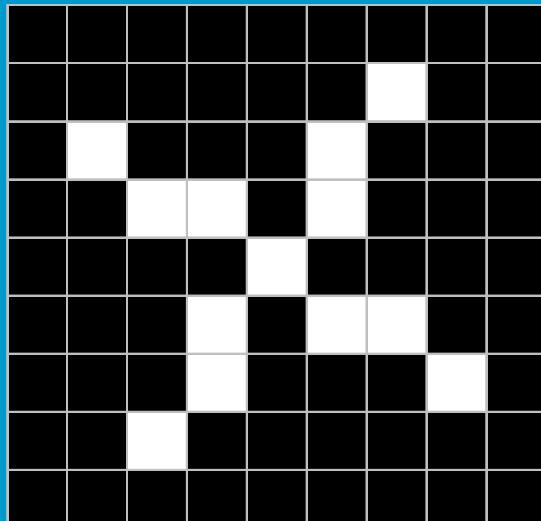
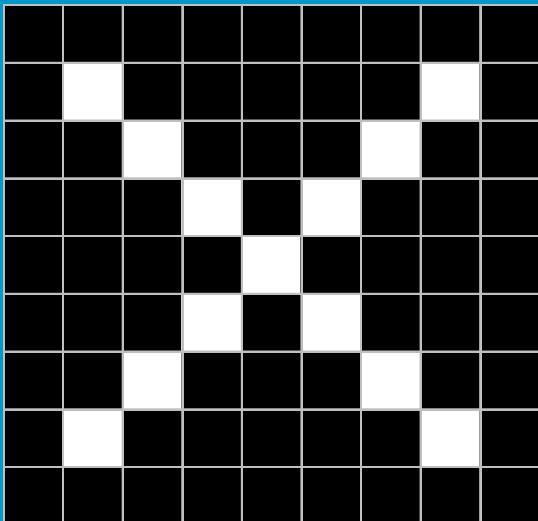
Escala

Rotação

Peso



Decisão difícil



O que o computador “enxerga”?

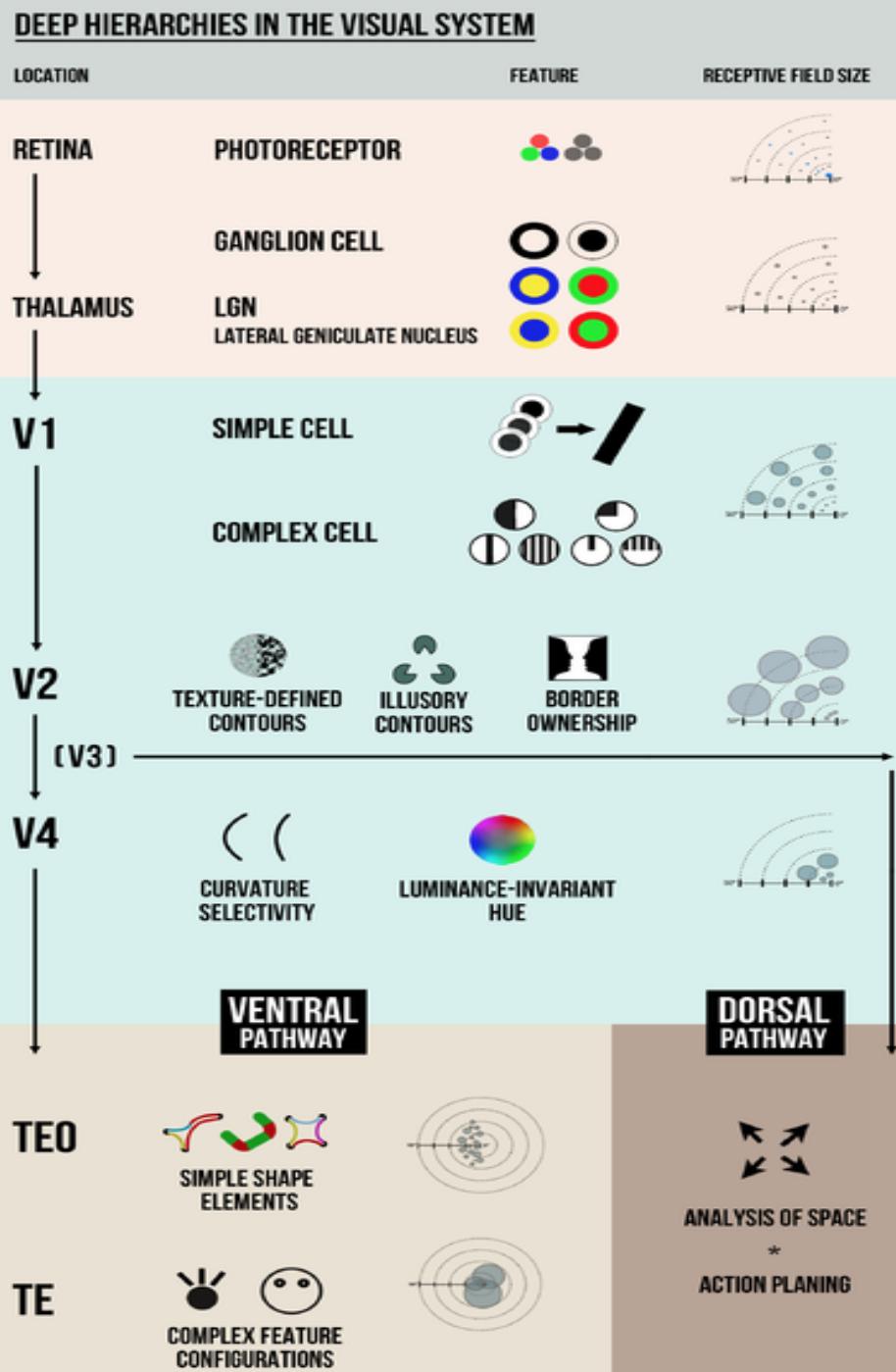


O que o computador “enxerga”?

O que o computador “enxerga”?

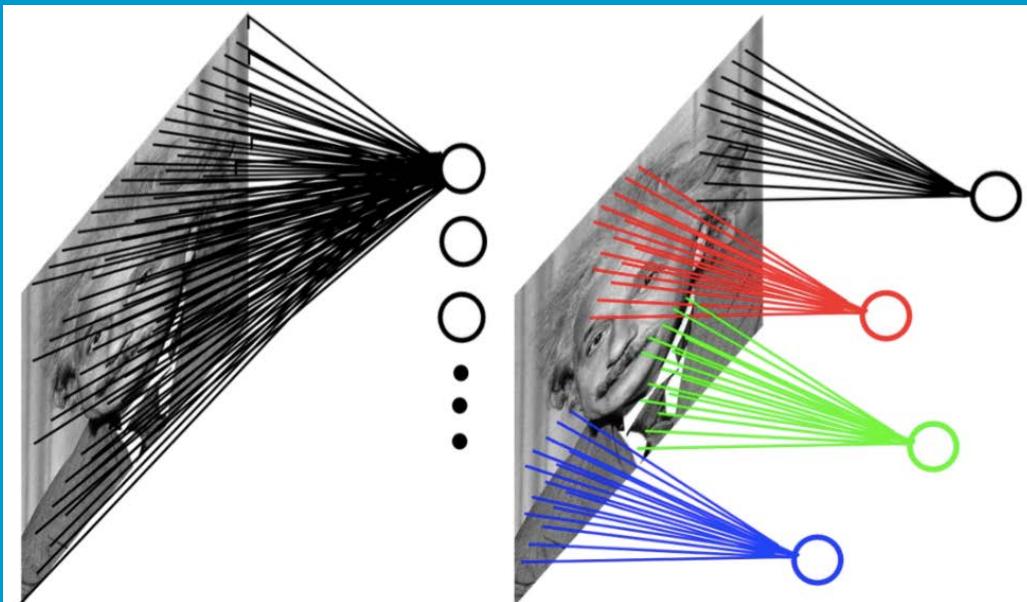


- Vision of the brain.
- Fotorreceptores
- Células ganglionares.
- Visão "central" e "periférica"



Rede Neural Convolucional

- Problema: Entrada pode ter uma dimensão muito alta. Multi-Layer Perceptron.
- Inspirado pelos experimentos de neurofisiologia [Hubel & Wiesel 1962], CNNs são um tipo especial de redes neurais que usam o conceito de campo receptivo local.



Exemplo: 200x200 imagem

- a) MLP: 40,000 hidden units
=> 1.6 bilhões de parâmetros
- b) CNN: 5x5 kernel, 100 campos receptivos => 2,500 parâmetros

Convolução

Convolução é uma operação matemática em duas funções (f e g) que produz uma terceira função, que normalmente é vista como uma versão modificada de uma das funções originais.

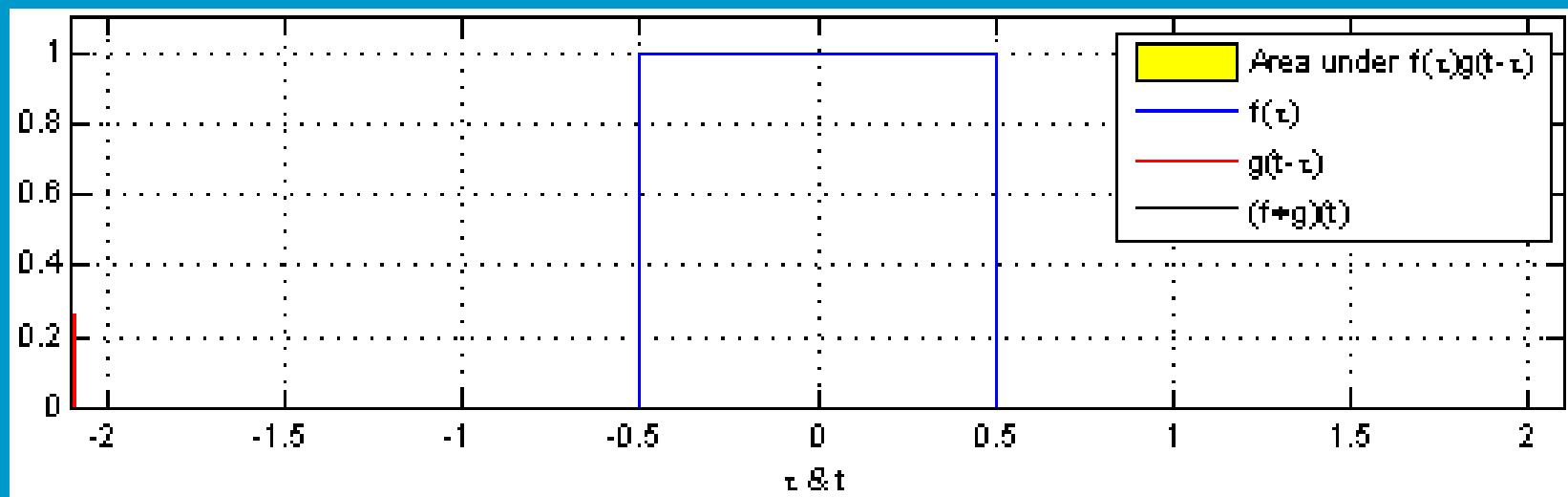
Comum no processamento de imagem:

- Altera as intensidades de um pixel para refletir as intensidades dos pixels circundantes.

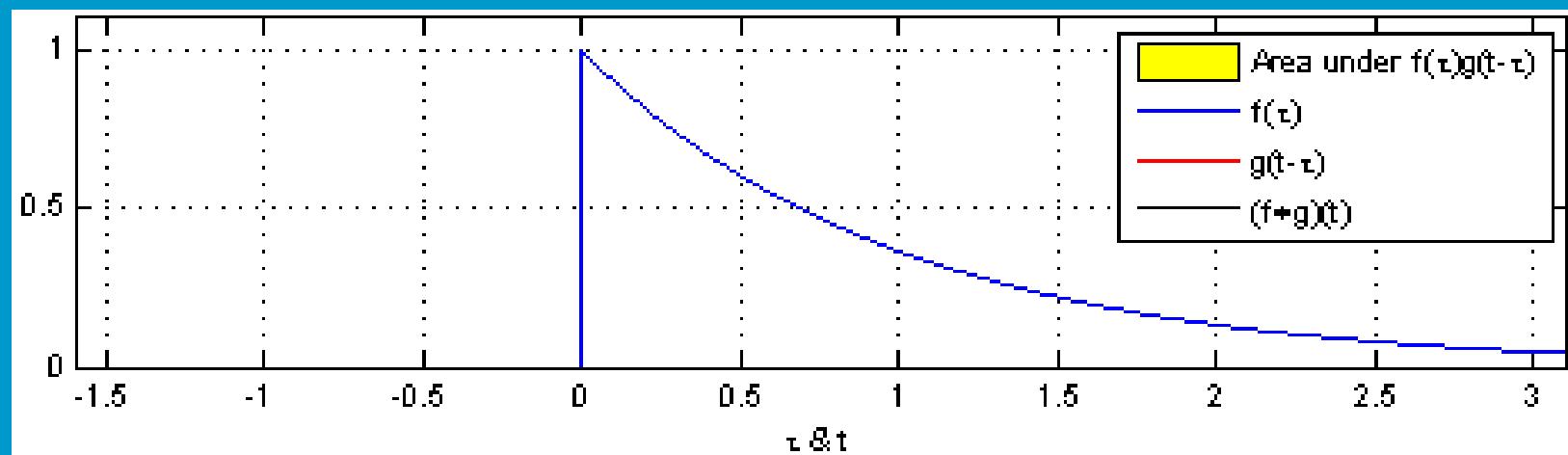
Convolução

- Não confundir com correlação cruzada:
- Ambas envolvem a integração de um produto entre duas funções.
Convolução é usada para expressar a um sinal pela resposta característica de outro sinal.
- Correlação cruzada expressa o grau de correspondência entre os sinais.

Convolução



Convolução

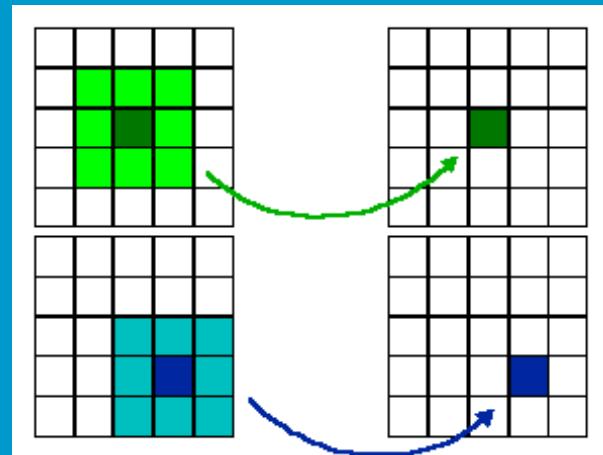


IMPORTANTE

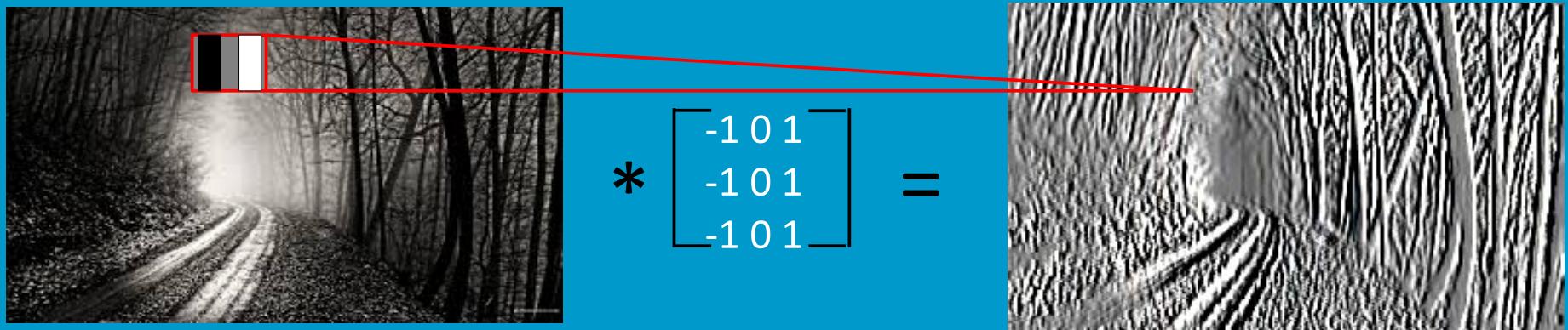
Convolução discreta

$$g(x) = f(x) \otimes h(x) = \sum_{\forall k} h(x - k)f(k)$$

$h(x)$: resposta de impulso (impulse response), *kernel*, máscara, filtro ou *template*.



IMPORTANTE



$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x-n_1, y-n_2]$$

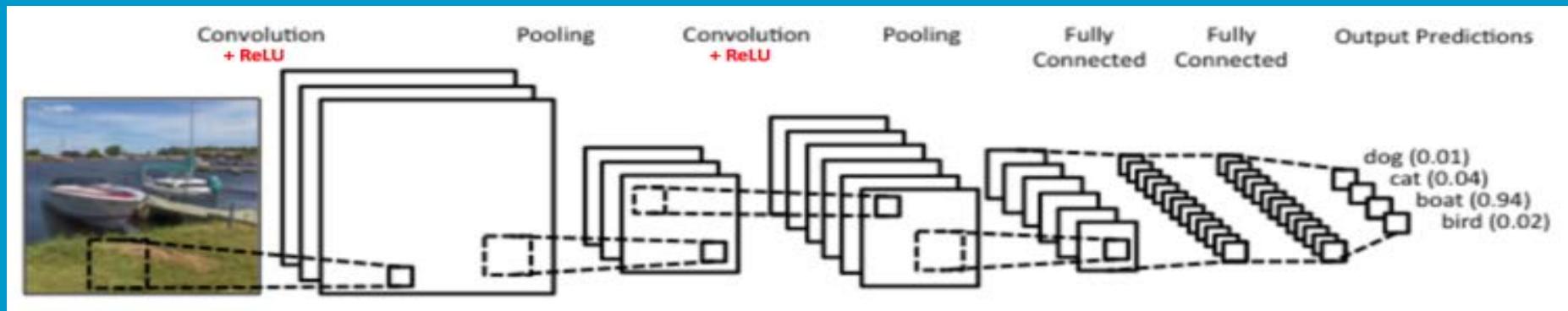
Multiplicação ponto-a-ponto e soma de um filtro com o sinal

CALMA, CALMA,

NÃO CRIEMOS PÂNICO



GERADORMEMES.COM



Entendendo a camada de convolução



Filtros: A matemática atrás da comparação

1. Multiplicar cada pixel da imagem pelo pixel correspondente do filtro.
2. Adicionar ao somatório.
3. Dividir pelo total de elementos.

Filtros: A matemática atrás da comparação

$$H = \begin{array}{|c|c|} \hline -1 & 1 \\ \hline \end{array}$$

$$G = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

$$G \otimes H = ?$$

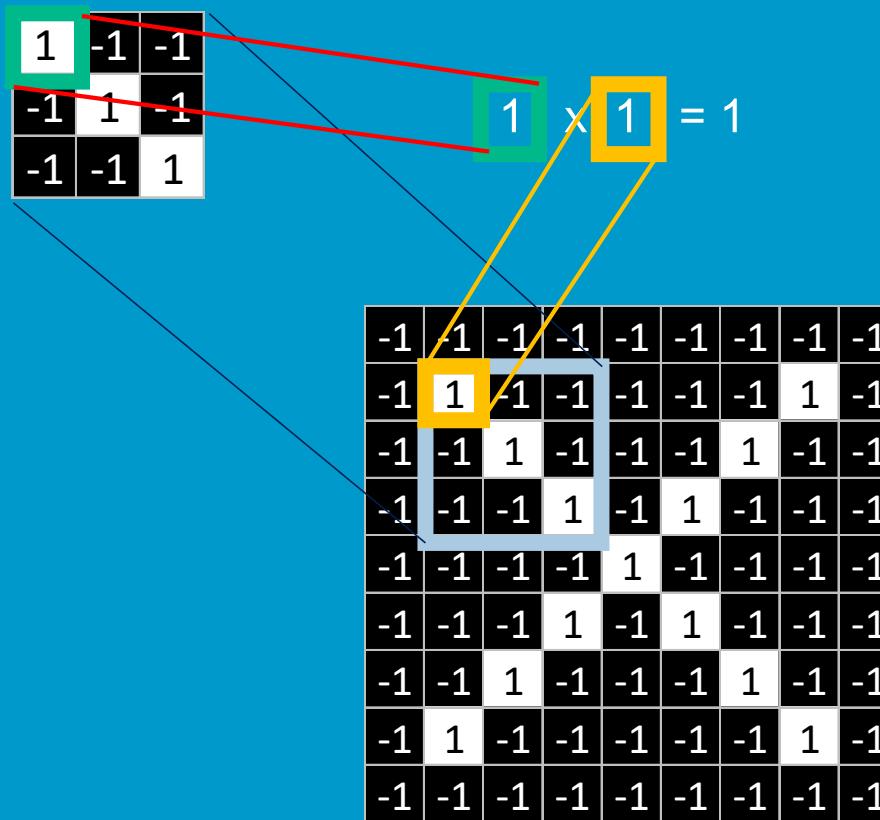
Filtros: A matemática atrás da comparação

$$H = \begin{array}{|c|c|} \hline -1 & 1 \\ \hline \end{array}$$

$$G = \begin{array}{cccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}$$

$$G \otimes H = \begin{array}{cccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Filtros: A matemática atrás da comparação



Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	

Filtros: A matemática atrás da comparação

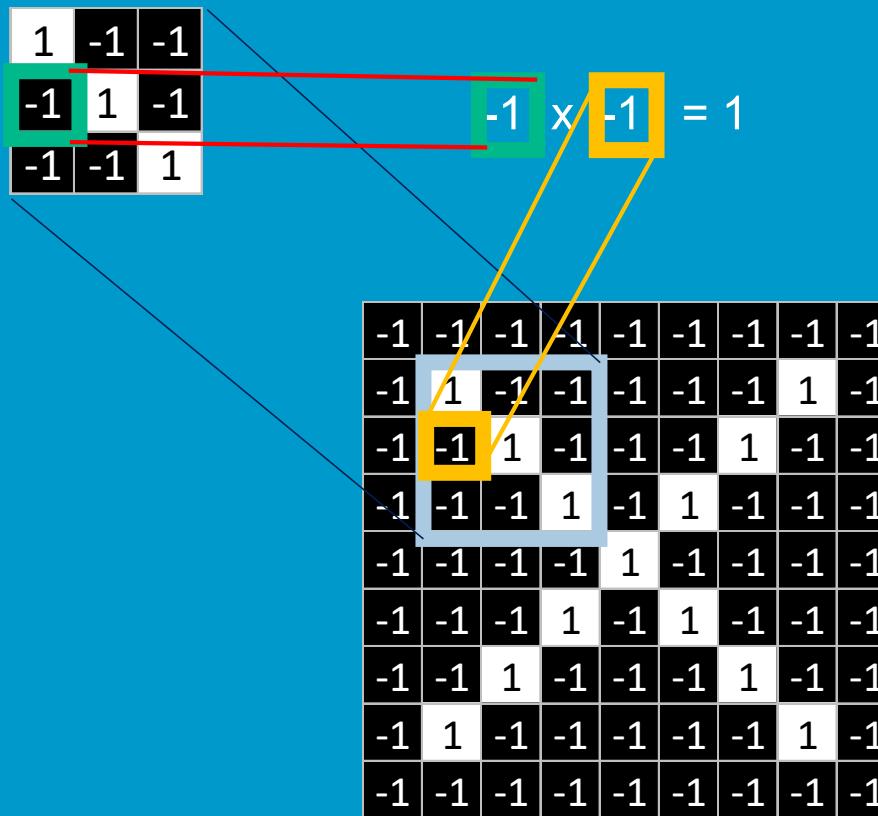
1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1

Filtros: A matemática atrás da comparação



1	1	1
1		

Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	

Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	1

Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	1
1		

Filtros: A matemática atrás da comparação

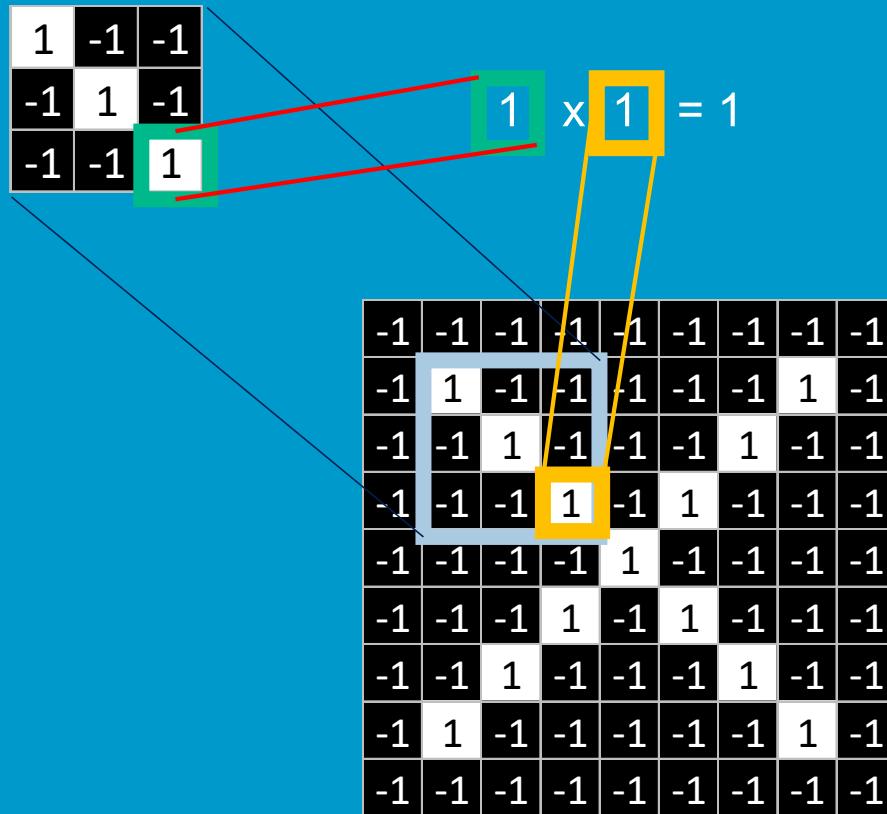
1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	1	-1	1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	1
1	1	

Filtros: A matemática atrás da comparação



1	1	1
1	1	1
1	1	1

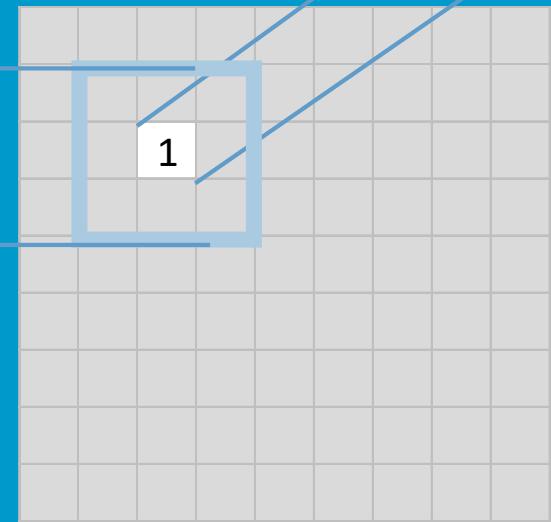
Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

1	1	1
1	1	1
1	1	1

$$\frac{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}{9} = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times 1 = -1$$

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	-1

Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	-1
1	1	1
-1	1	1

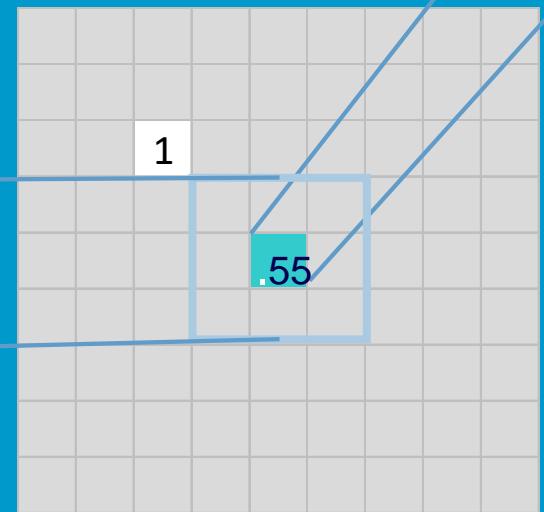
Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

1	1	-1
1	1	1
-1	1	1

$$\frac{1 + 1 - 1 + 1 + 1 + 1 - 1 + 1 + 1}{9} = .55$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Filtros: A matemática atrás da comparação

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Filtros: A matemática atrás da comparação

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Filtros: A matemática atrás da comparação

0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Matriz numérica do filtro convolucional



Table 3.2: Visualização do filtro convolucional



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	0	30	0
0	0	0	0	0	30	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

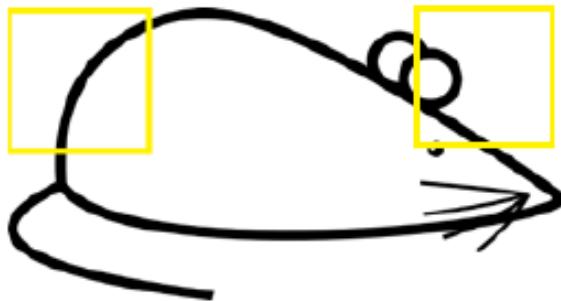


Figure 3.2: Imagem de entrada para aplicação do filtro de convolução

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

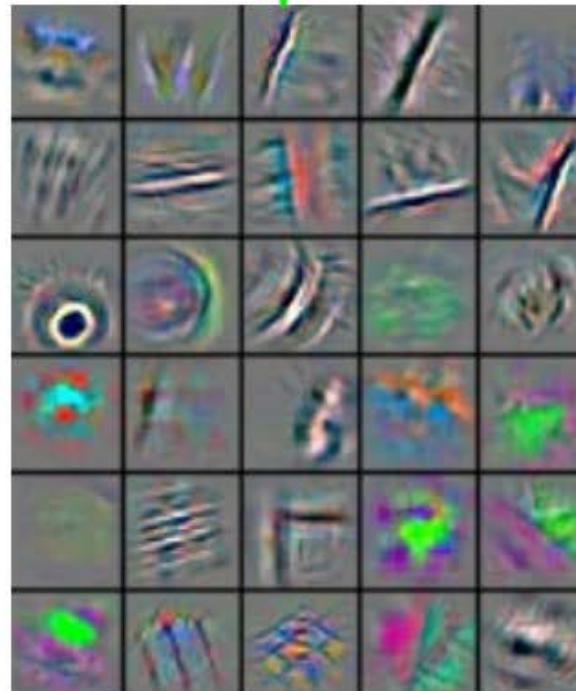
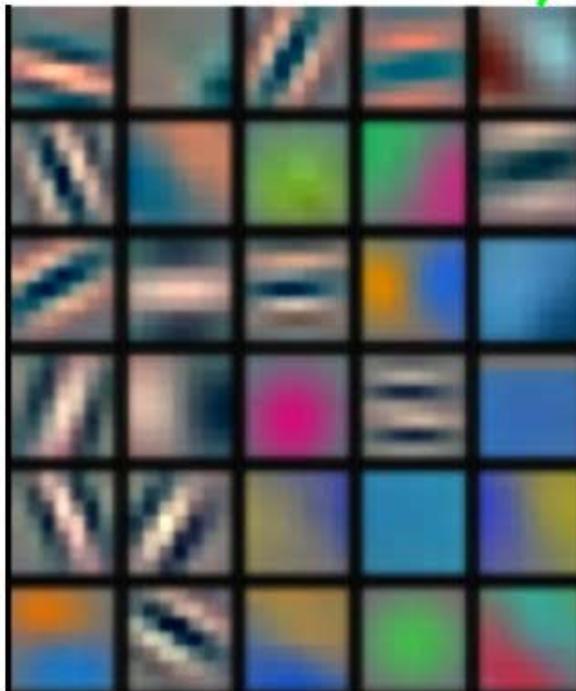
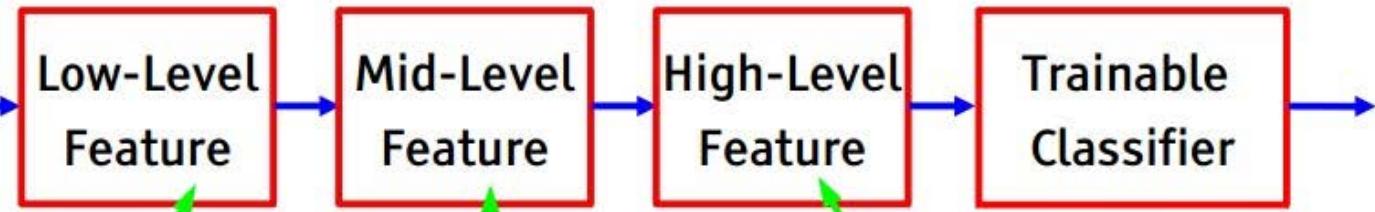
Imagen de entrada considerando a região do amarelo no canto superior esquerdo

0	0	0	0	0	0	30	0
0	0	0	0	0	30	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0

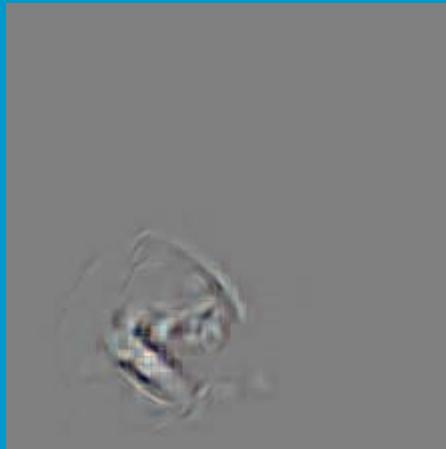
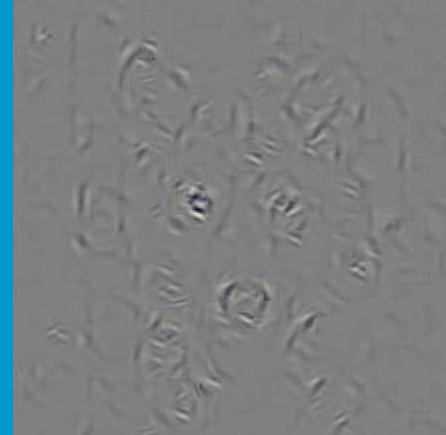
Table 3.4: Filtro convolucional

O resultado, omitindo-se a multiplicação dos valores zerados será dado por $(50*30)+(50*30)+(50*30)+(20*30)+(50*30)=6600$. O valor 6600 quantifica a presença da característica descrita pelo filtro convolucional na imagem.

[From recent Yann LeCun slides]

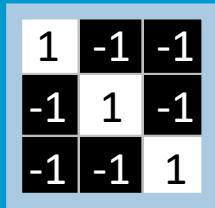


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



Obtido de: https://github.com/InFoCusp/tf_cnnvis
Ver o vídeo de <http://yosinski.com/deepvis>

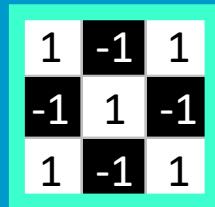
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	1	-1	-1	-1	
-1	-1	-1	1	1	-1	-1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	



=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

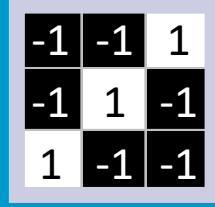
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	1	-1	1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	



=

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	-1	-1	1	1	-1	1	-1	-1	
-1	-1	-1	-1	1	-1	-1	-1	-1	
-1	-1	-1	1	-1	1	-1	-1	-1	
-1	-1	1	-1	-1	-1	1	-1	-1	
-1	1	-1	-1	-1	-1	-1	1	-1	
-1	-1	-1	-1	-1	-1	-1	-1	-1	

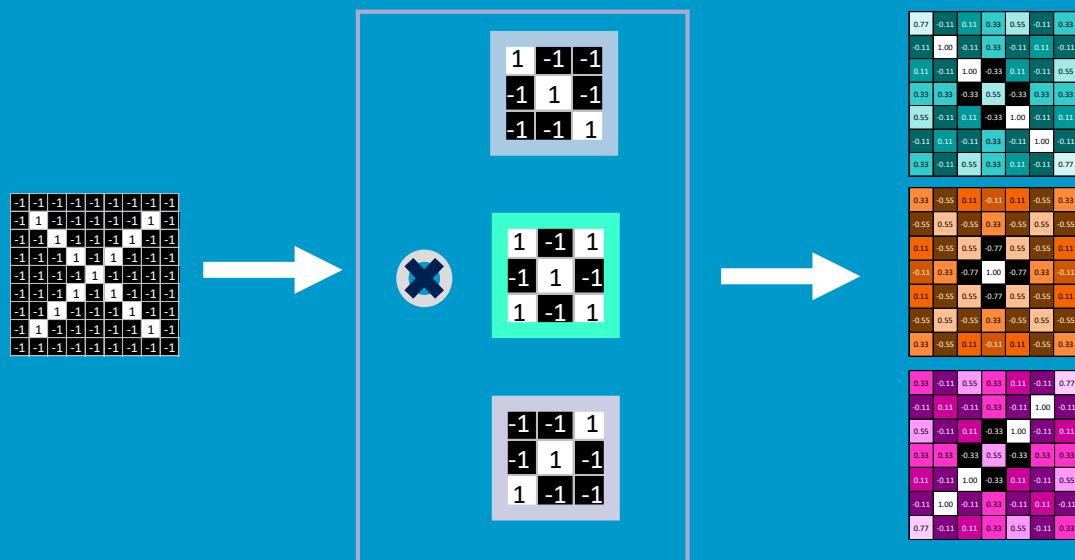


=

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

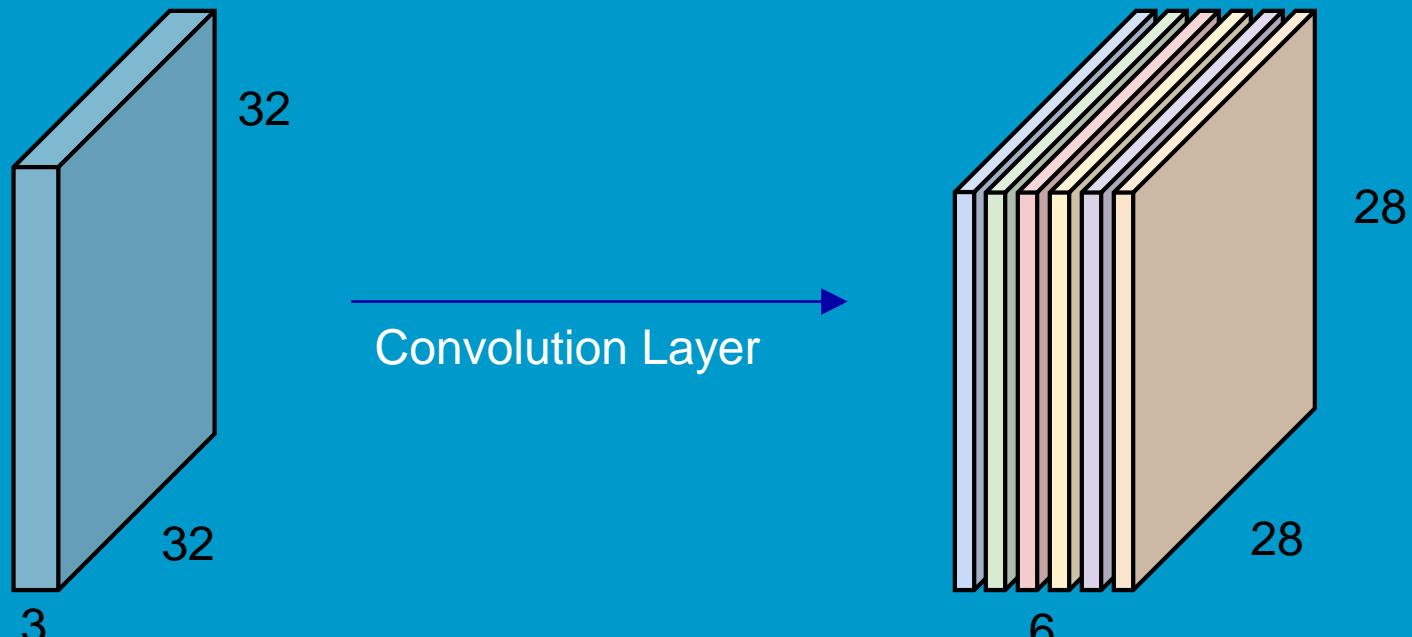
Camada de convolução

Uma imagem gera um conjunto de imagens



Camada de convolução

Por exemplo, se nós temos 6 filtros 5×5 , nós iremos obter 6 mapas de ativação:

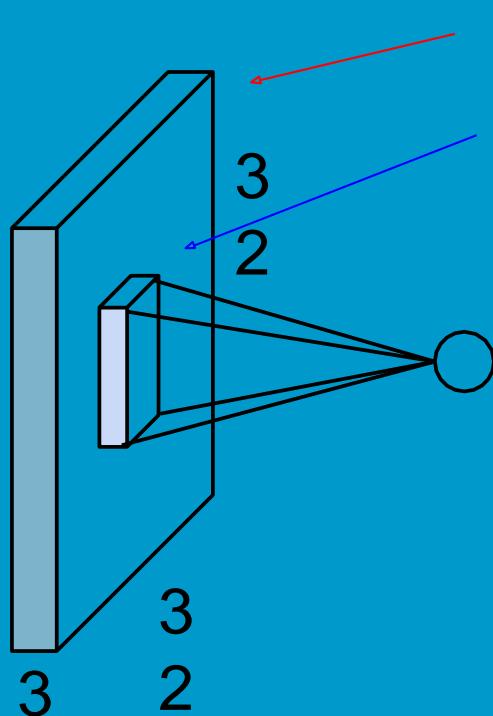


Nós empilhamos estas novamos imagens $28 \times 28 \times 6$!

Camada de convolução: detalhes

Olhando mais de perto as dimensões:

Mapa de ativação



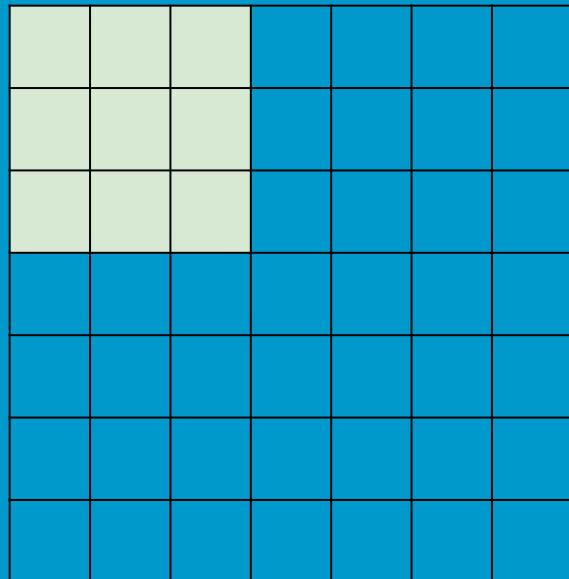
32x32x3 imagem
5x5x3 filtro

Convolua (slide) sobre todas as
posições



Camada de convolução: detalhes

Olhando mais de perto as dimensões:

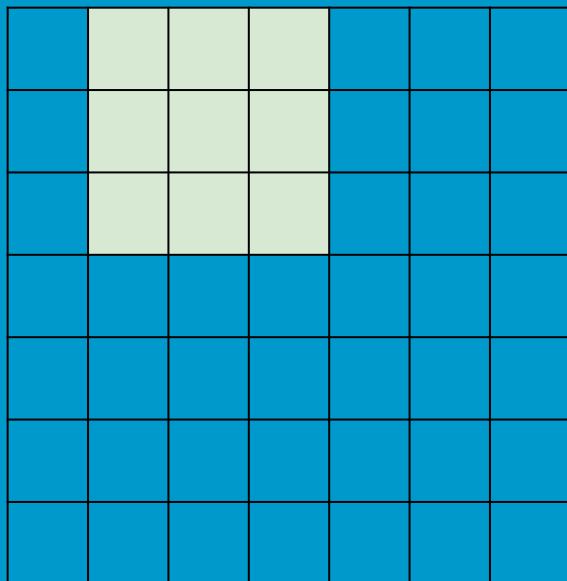


Entrada 7x7

Filtro 3x3

Camada de convolução: detalhes

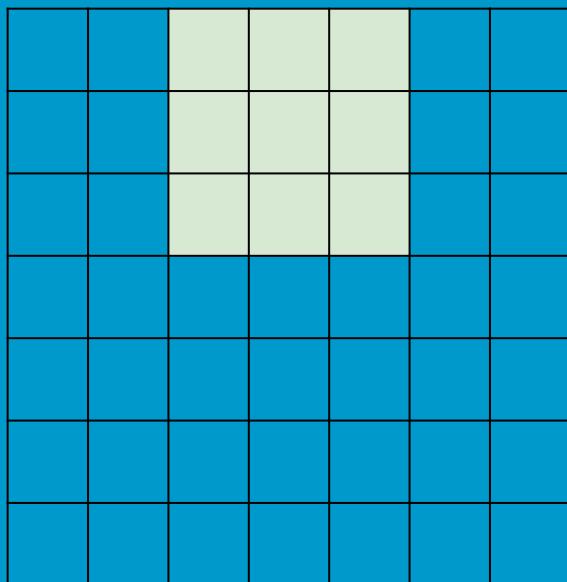
Olhando mais de perto as dimensões:



- Entrada 7×7
- Filtro 3×3

Camada de convolução: detalhes

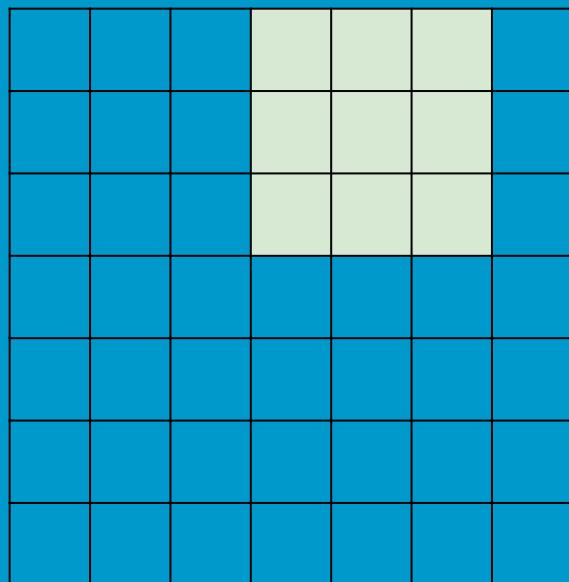
Olhando mais de perto as dimensões:



- Entrada 7×7
- Filtro 3×3

Camada de convolução: detalhes

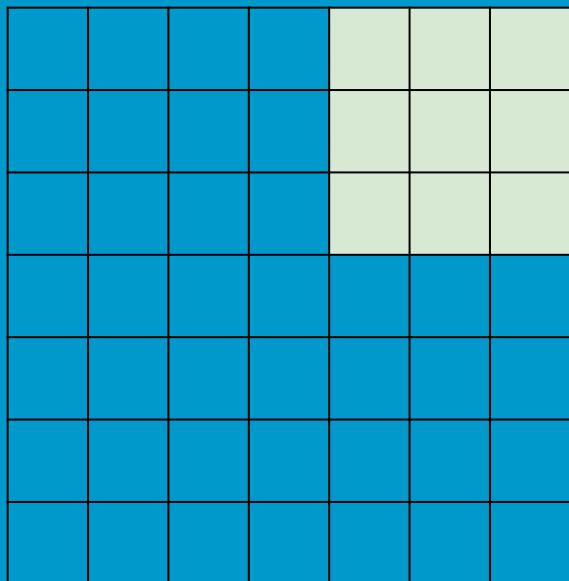
Olhando mais de perto as dimensões:



- Entrada 7×7
- Filtro 3×3

Camada de convolução: detalhes

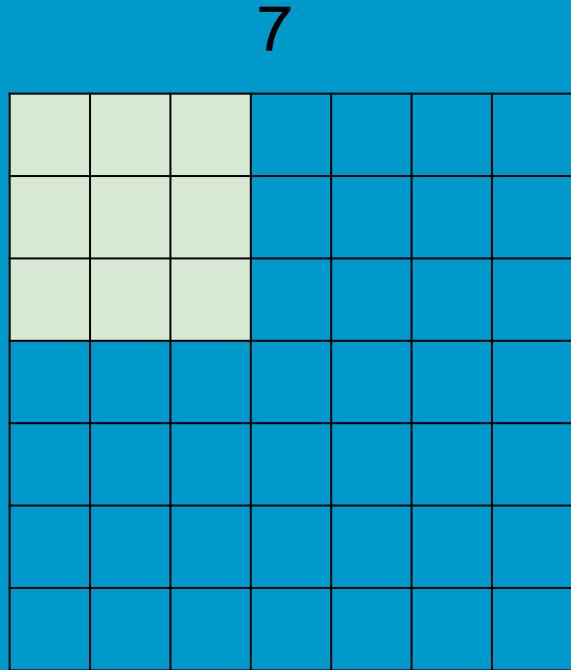
Olhando mais de perto as dimensões:



- Entrada 7×7
- Filtro 3×3
- => **5x5 output**

Camada de convolução: detalhes

Olhando mais de perto as dimensões:



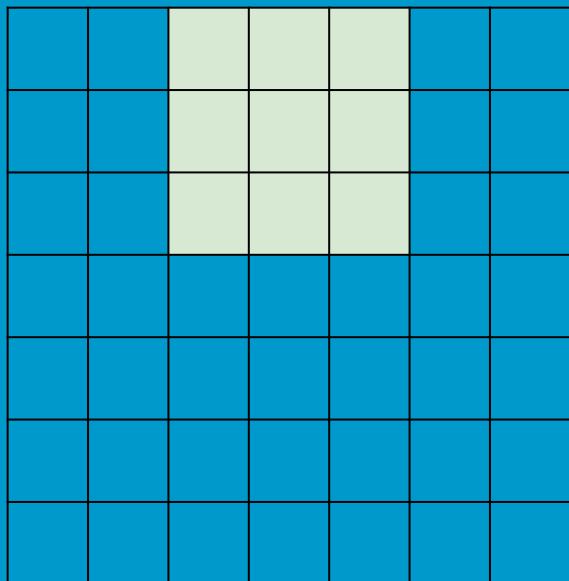
7

Entrada 7x7
Filtro 3x3 filter
com stride 2

Camada de convolução: detalhes

Olhando mais de perto as dimensões:

7

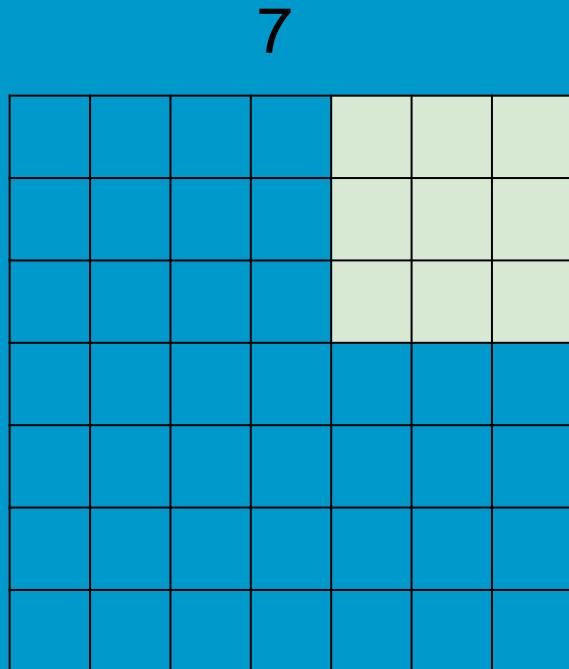


Entrada 7x7
Filtro 3x3 filter
com stride 2

7

Camada de convolução: detalhes

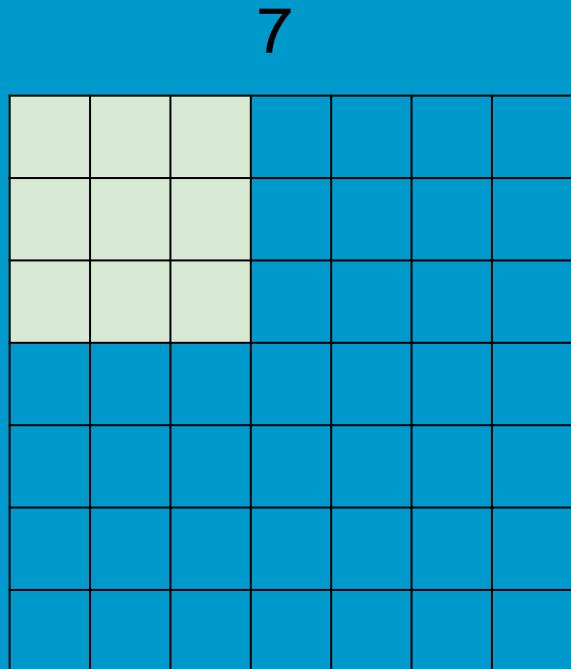
Olhando mais de perto as dimensões:



Entrada 7×7
Filtro 3×3 filter
com stride 2
=> Saída 3×3

Camada de convolução: detalhes

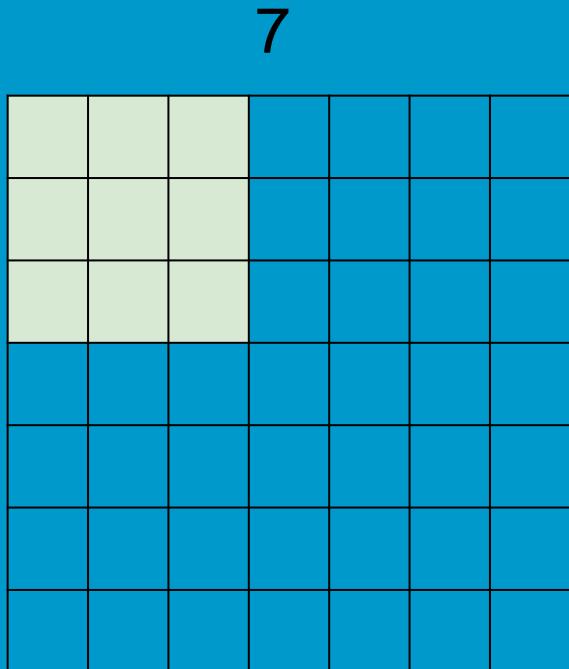
Olhando mais de perto as dimensões:



Entrada 7x7
Filtro 3x3 filter
com stride 3

Camada de convolução: detalhes

Olhando mais de perto as dimensões:

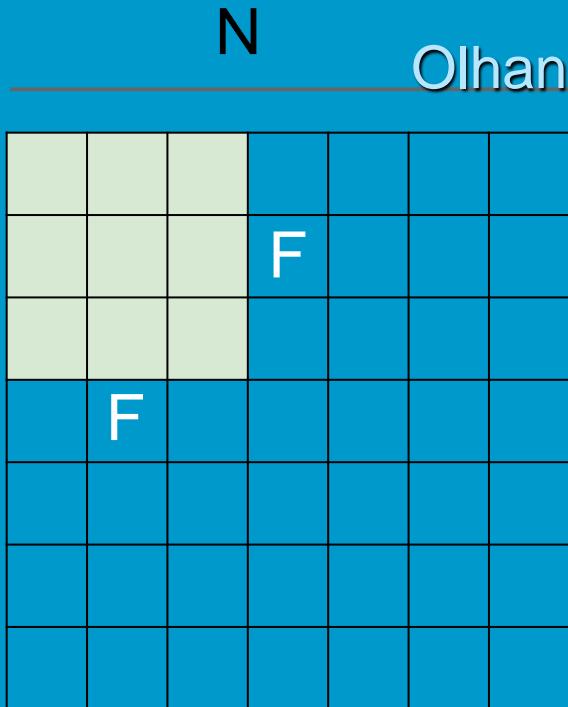


Entrada 7x7
Filtro 3x3 filter
com stride 3

doesn't fit!

Você não consegue usar um filtro 3x3 em uma imagem 7x7 com stride 3.

Camada de convolução: detalhes



Olhando mais de perto as dimensões:

Calculando a saída:
 $(N - F) / \text{stride} + 1$

N

e.g. $N = 7$, $F = 3$:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$:\

Camada de convolução: detalhes

Na prática, costuma-se usar zero na borda

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. Entrada 7x7

Filtro 3x3 com stride 1

pad com 1 pixel de borda

=> Qual é a saída?

(recall:)

$$(N - F) / \text{stride} + 1$$

Camada de convolução: detalhes

Na prática, costuma-se usar zero na borda

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. Entrada 7x7

Filtro 3x3 com stride 1

pad com 1 pixel de borda

=> Qual é a saída?

7x7 output!

Camada de convolução: detalhes

Na prática, costuma-se usar zero na borda

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. Entrada 7x7

Filtro 3x3 com stride 1

pad com 1 pixel de borda

=> Qual é a saída?

7x7 output!

Geralmente o mais comum é camada com stride 1, filtros de tamanho FxF e zero-padding ($F-1)/2$. (preserva a dimensão)

e.g. $F = 3 \Rightarrow$ zero pad with 1 $F = 5 \Rightarrow$ zero pad with 2 $F = 7 \Rightarrow$ zero pad with 3

$$(N + 2 * \text{padding} - F) / \text{stride} + 1$$

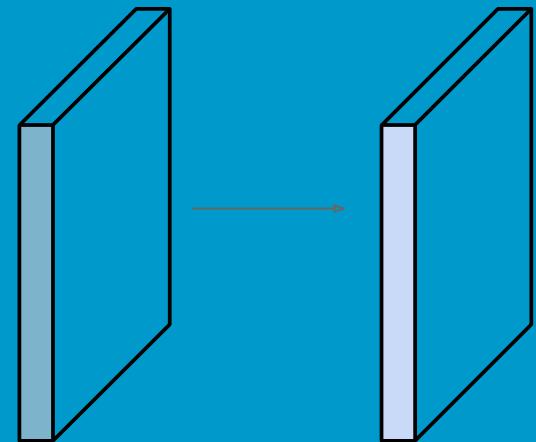
Camada de convolução: detalhes

Exemplos:

Entrada: **32x32x3**

10 filtros 5x5 com stride 1, pad 2

Dimensão da saída: ?

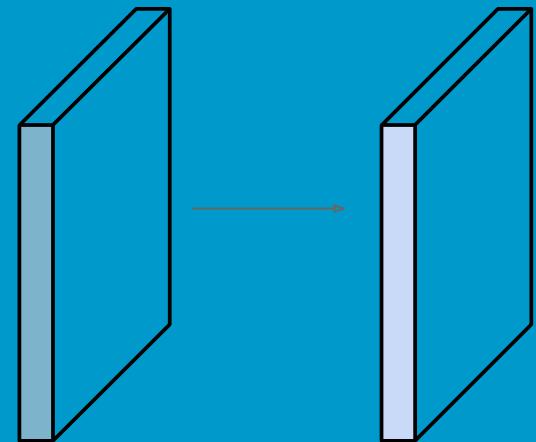


Camada de convolução: detalhes

Exemplos:

Entrada: **32x32x3**

10 filtros 5x5 com stride 1, pad 2



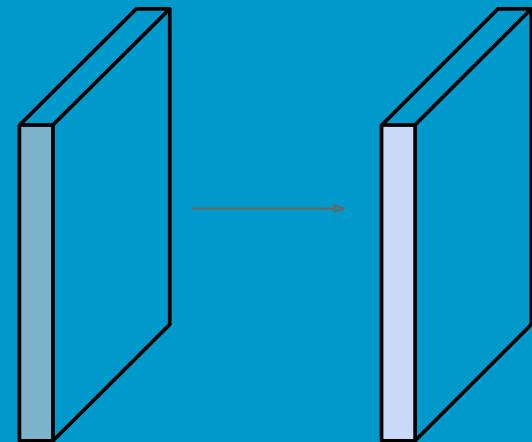
Dimensão da saída:
 $(32+2*2-5)/1+1 = 32$

Logo: **32x32x10**

Camada de convolução: detalhes

Entrada: **32x32x3**

10 filtros **5x5** com stride 1, pad 2

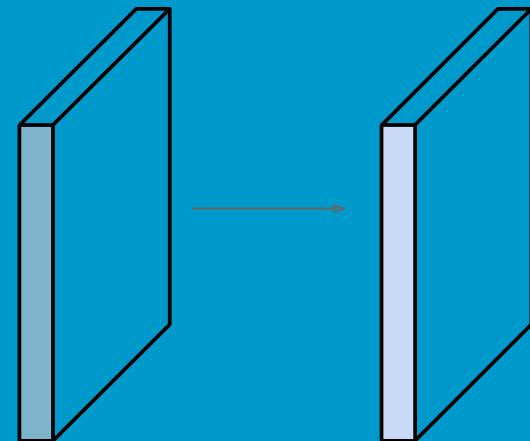


Número de parâmetros na camada?

Camada de convolução: detalhes

Entrada: **32x32x3**

10 filtros **5x5** com stride 1, pad 2



Número de parâmetros na camada?

(+1 for bias)

Cada filtro possui $5 \times 5 \times 3 + 1 = 76$ parâmetros

$$\Rightarrow 76 \times 10 = 760$$

Camada de convolução: detalhes

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

Camada de convolução: implementação

```
# Camada de convolução com 32 filtros e kernel de tamanho 5  
conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)
```

```
# Camada de convolução com 64 filtros e kernel de tamanho 3  
conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)
```

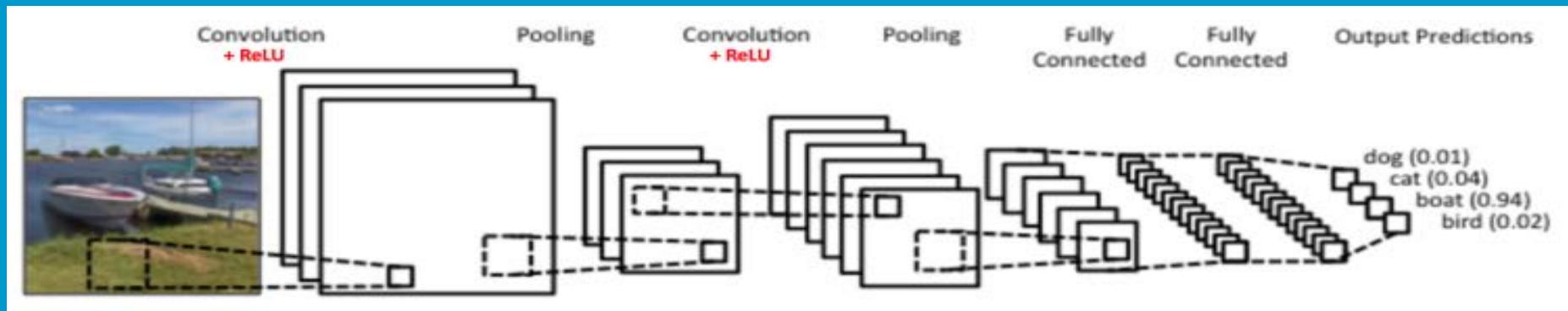
Camada de convolução: demonstrações

Animações dos parâmetros de deslocamento
dos filtros convolucionais

https://github.com/vdumoulin/conv_arithmetic

Animações de convolução:

<http://setosa.io/ev/image-kernels/>



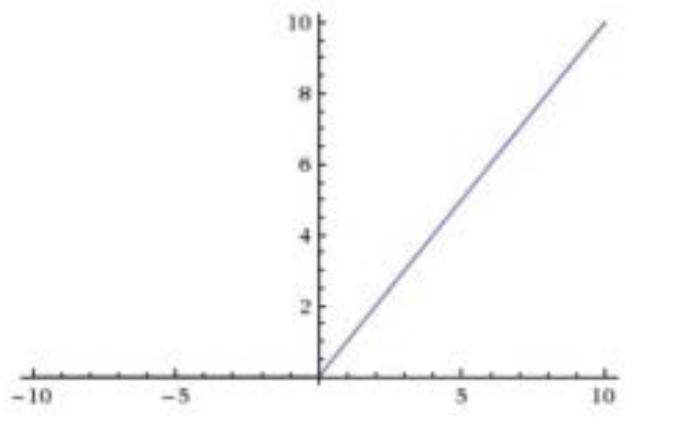
Entendendo a camada ReLU



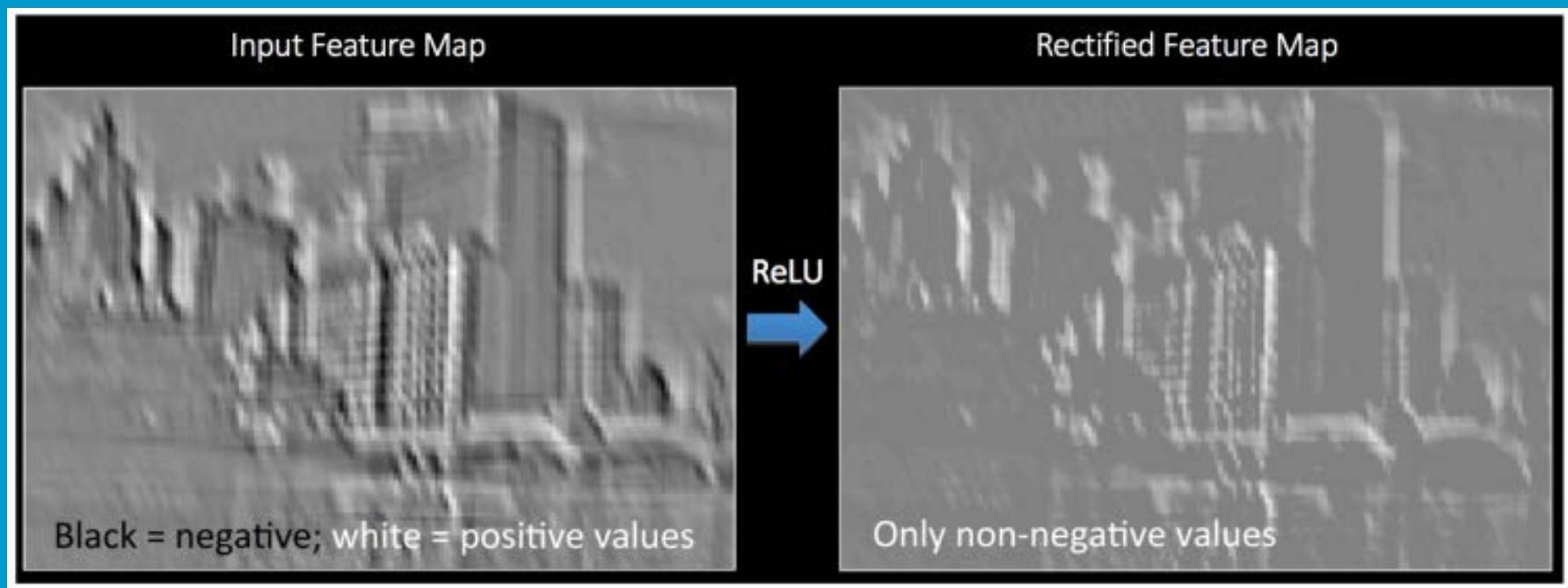
Entendendo a camada ReLU

- Rectified Linear Units
- Função de ativação das deep learnings
 - Veremos outras no curso

Output = Max(zero, Input)

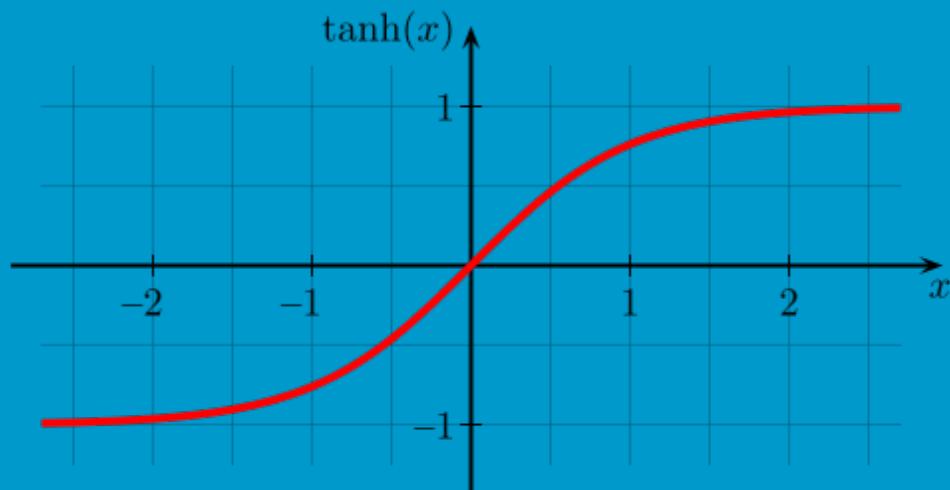


Entendendo a camada ReLU



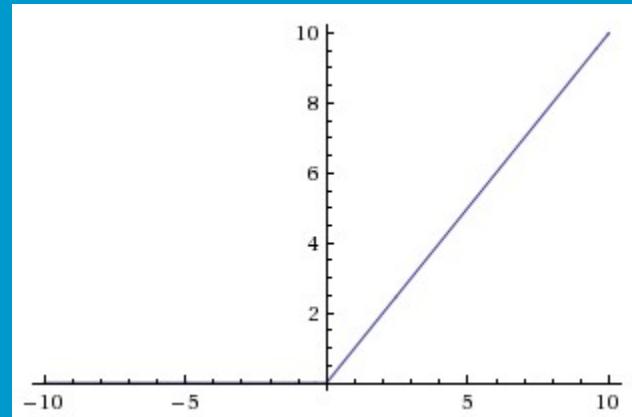
Entendendo a camada ReLU

Tanh(x)



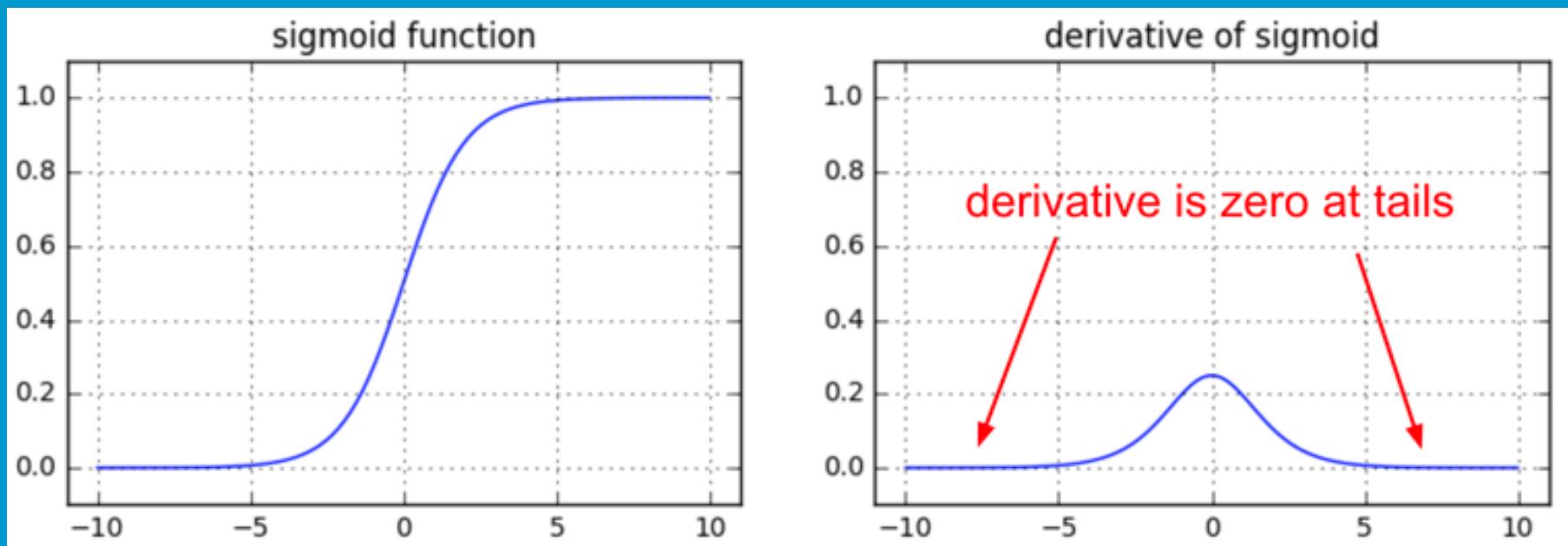
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU

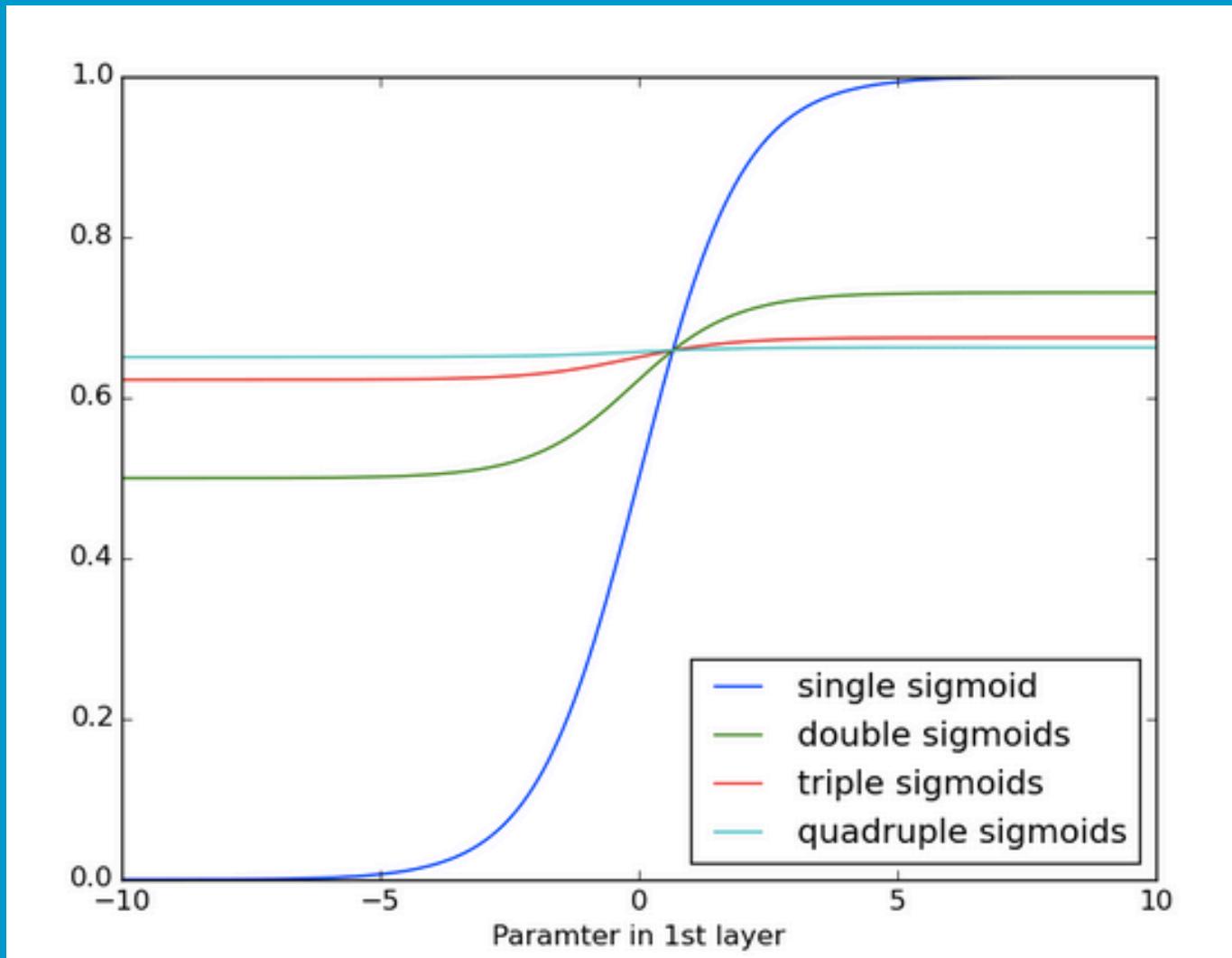


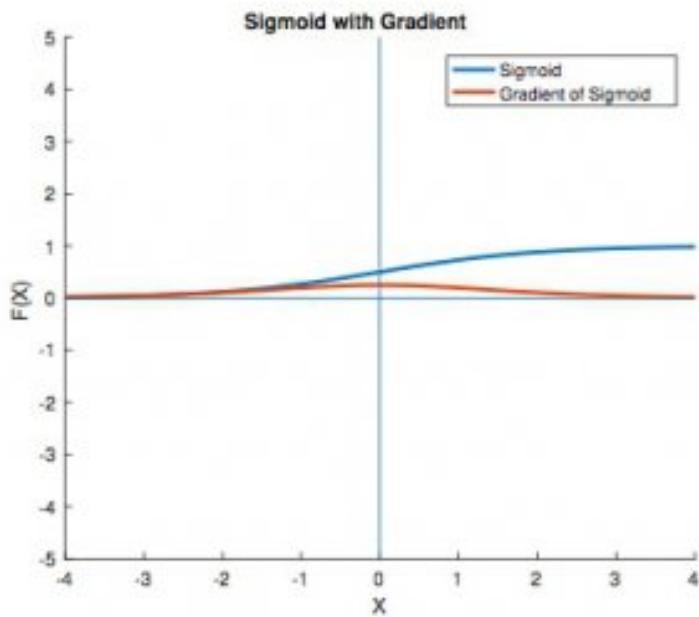
$$f(x) = \max(0, x)$$

Entendendo a camada ReLU

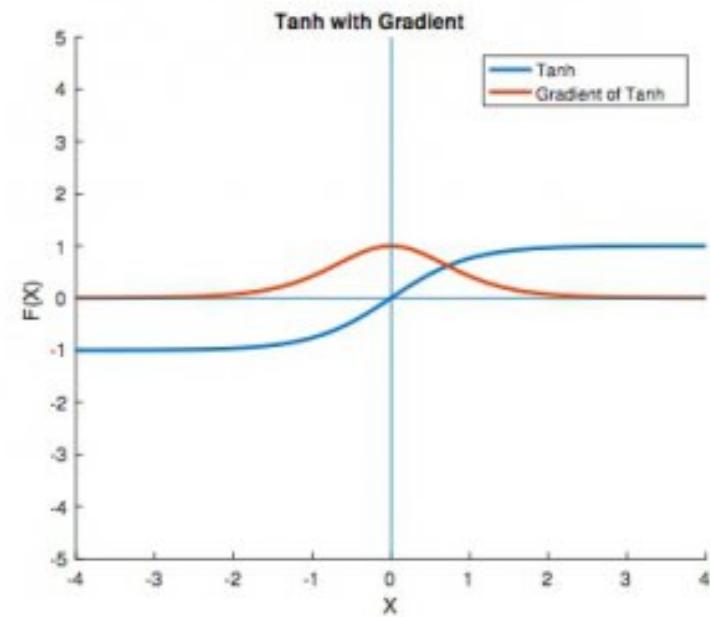


Entendendo a camada ReLU

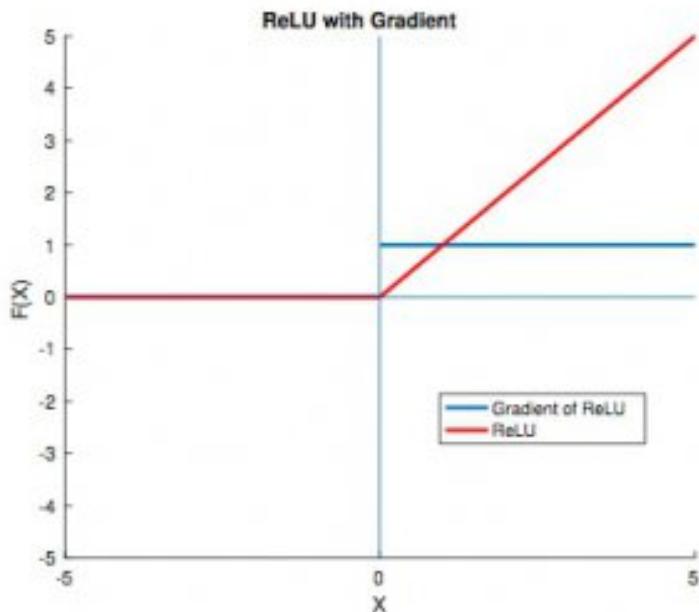




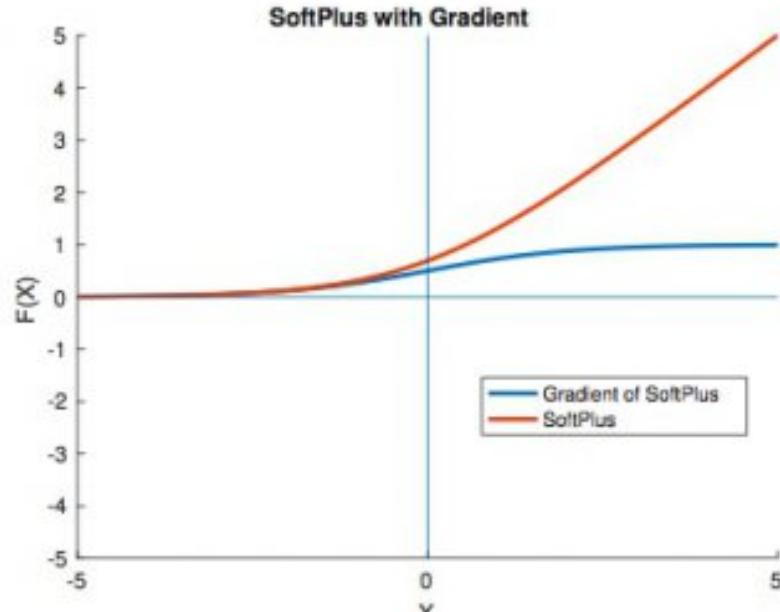
(a) Sigmoid



(b) Tanh

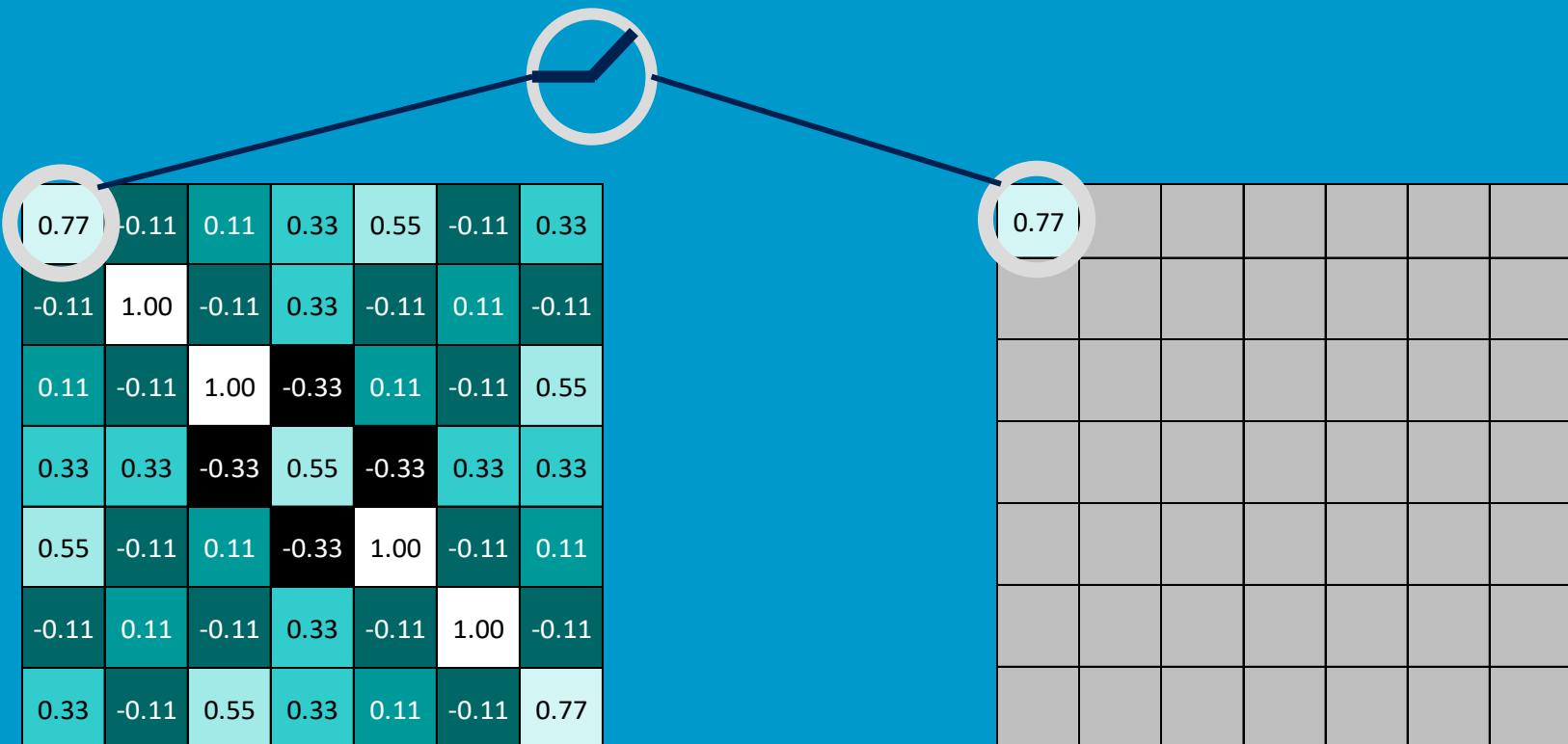


(c) ReLU

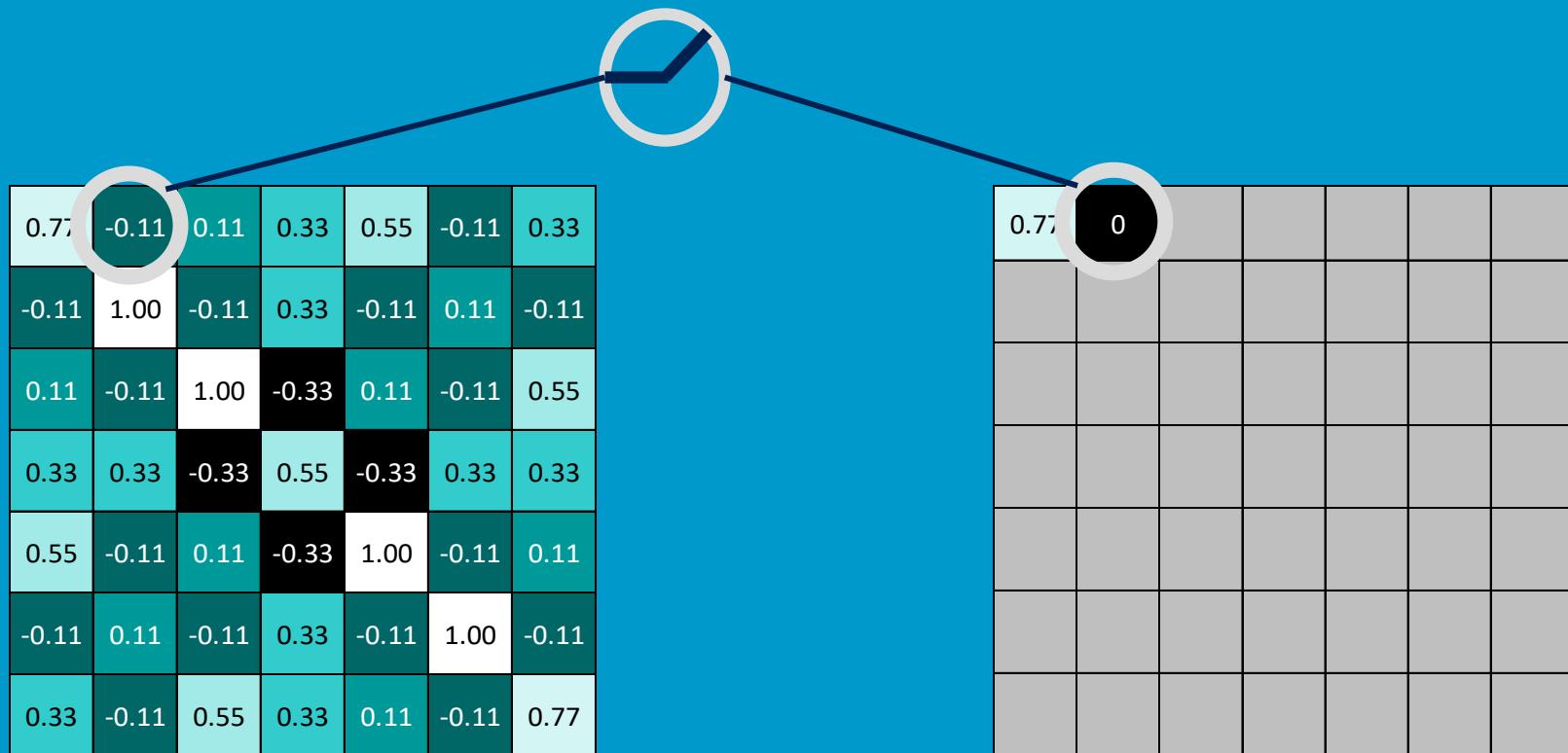


(d) Softplus

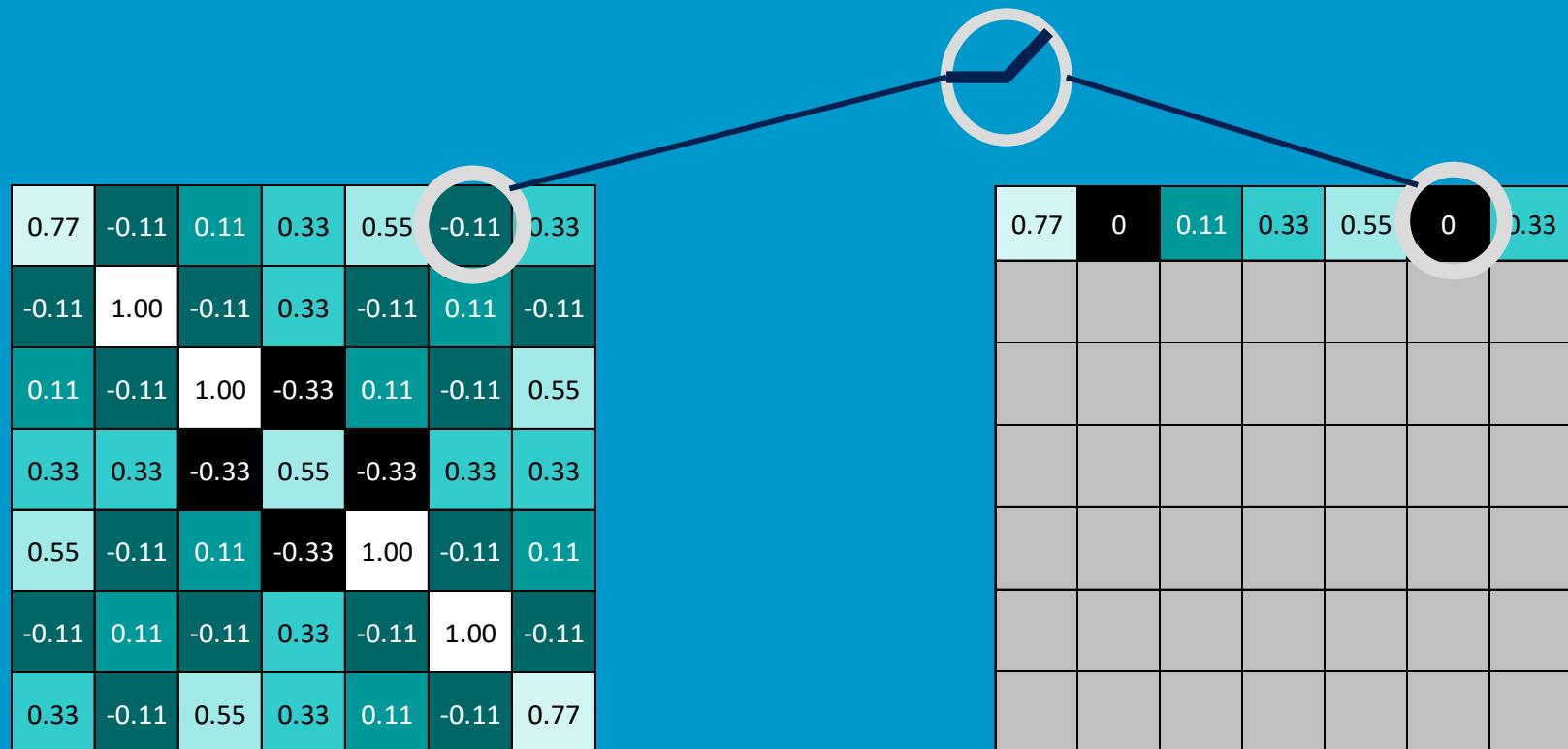
Entendendo a camada ReLU



Entendendo a camada ReLU



Entendendo a camada ReLU

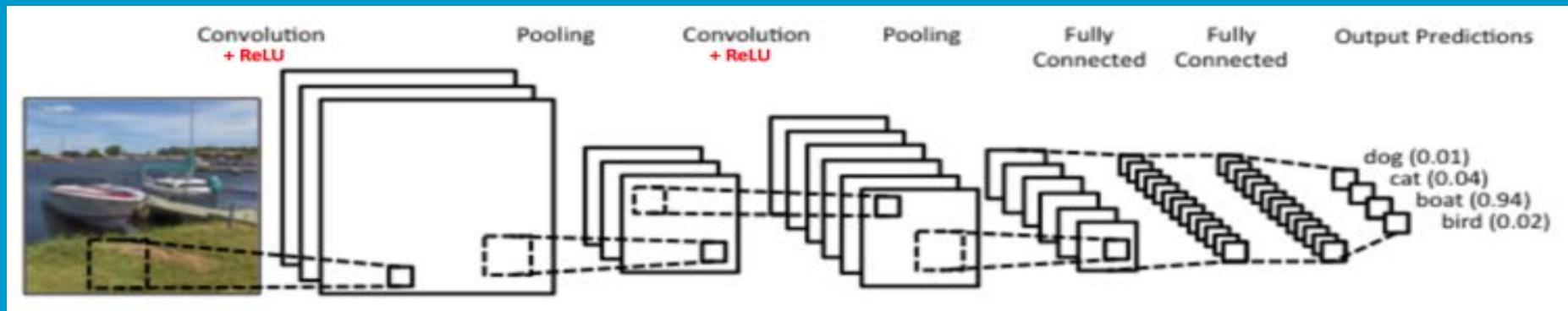


Entendendo a camada ReLU

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77



Entendendo a camada Pooling



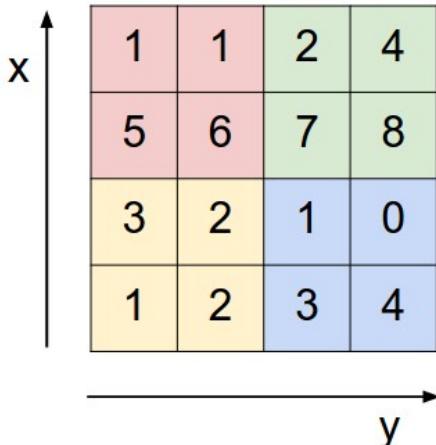
Entendendo a camada de Pooling

- A camada pooling torna a convolução invariante a translação, rotação e shifting (janelamento).
- Também chamada de “downsample”
- Opções: Min, Max, Média.
- Opção mais usada: max-pooling.

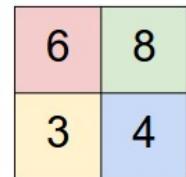
Entendendo a camada de Pooling

- Toma-se a maior ativação para propagar na região de interesse do campo receptivo.
- Mesmo imagens pouco deslocadas, uma vez que estamos à procura de maior ativação, somos capazes de capturar semelhanças entre as imagens.

Single depth slice

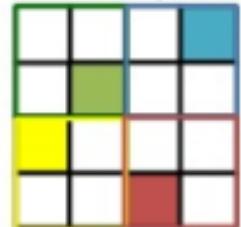


max pool with 2x2 filters
and stride 2

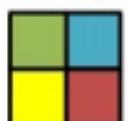


224x224x64

Max pooling



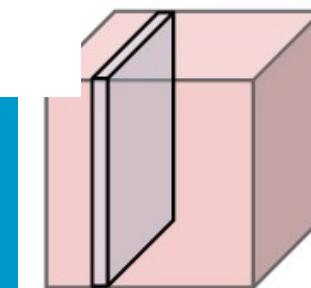
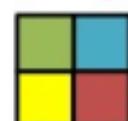
Max in a 2x2 filter



Average pooling



Average in a 2x2 filter

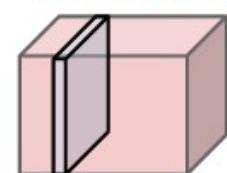


224

224

pool

112x112x64



↑



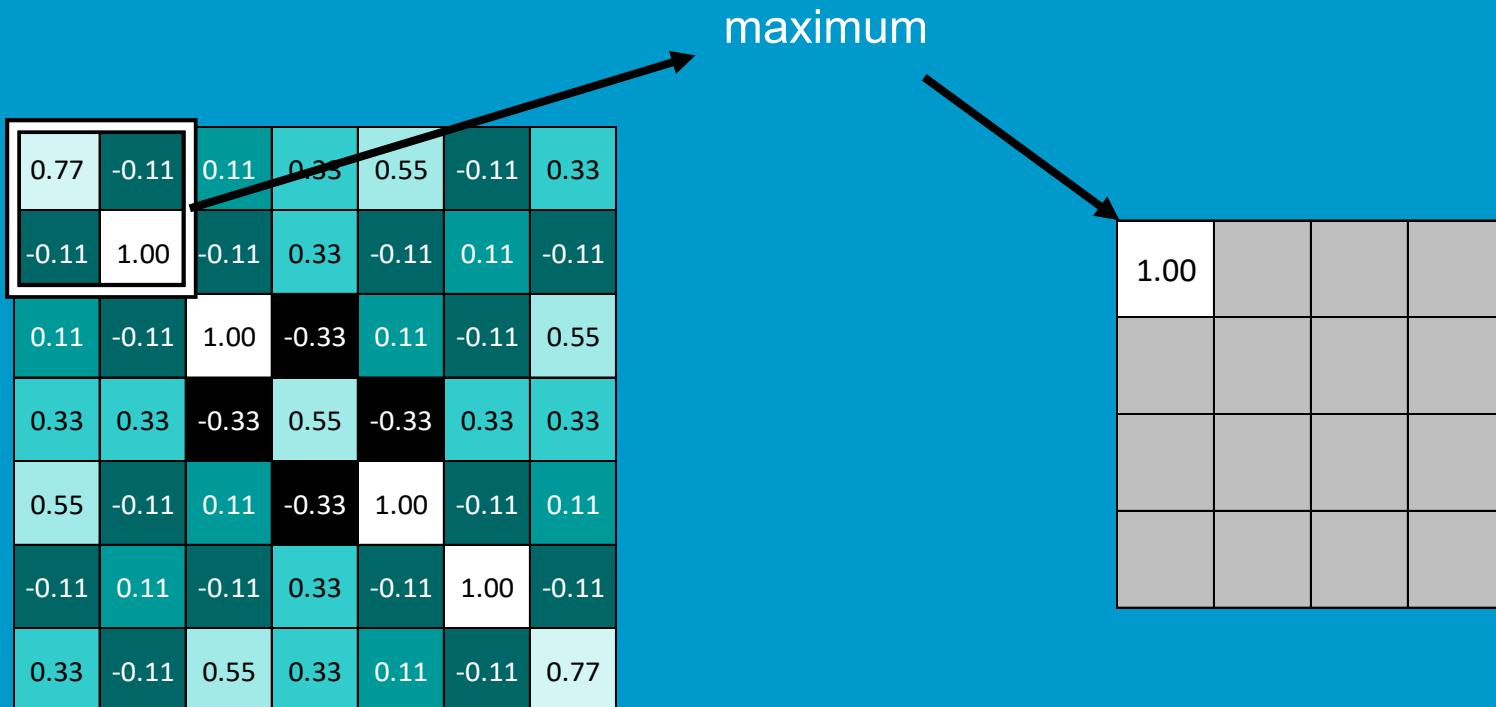
downsampling

112

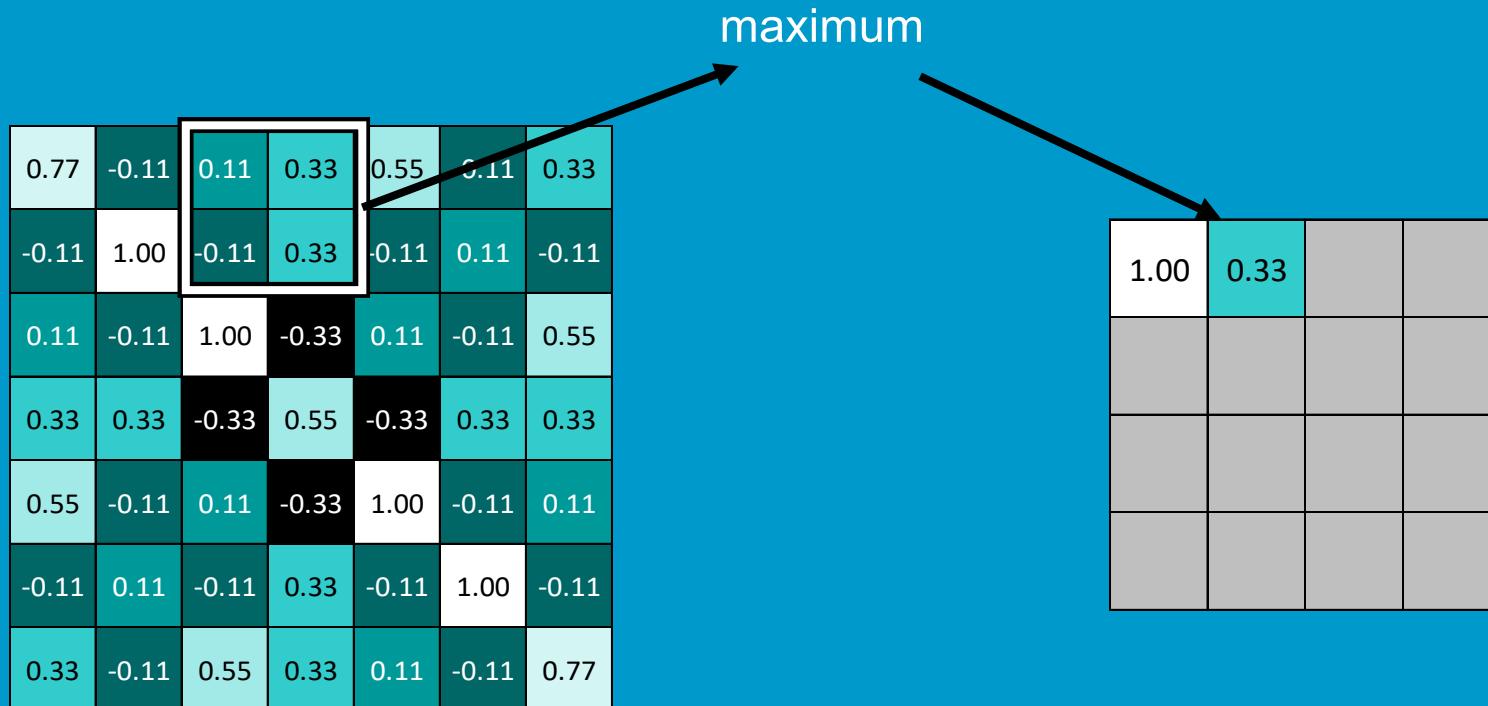
Pooling: Algoritmo

1. Escolha um tamanho de janela (2 ou 3).
2. Caminhe sua janela no sinal.
3. Para cada janela, pegue o valor máximo
(para os caso do Max-pooling).

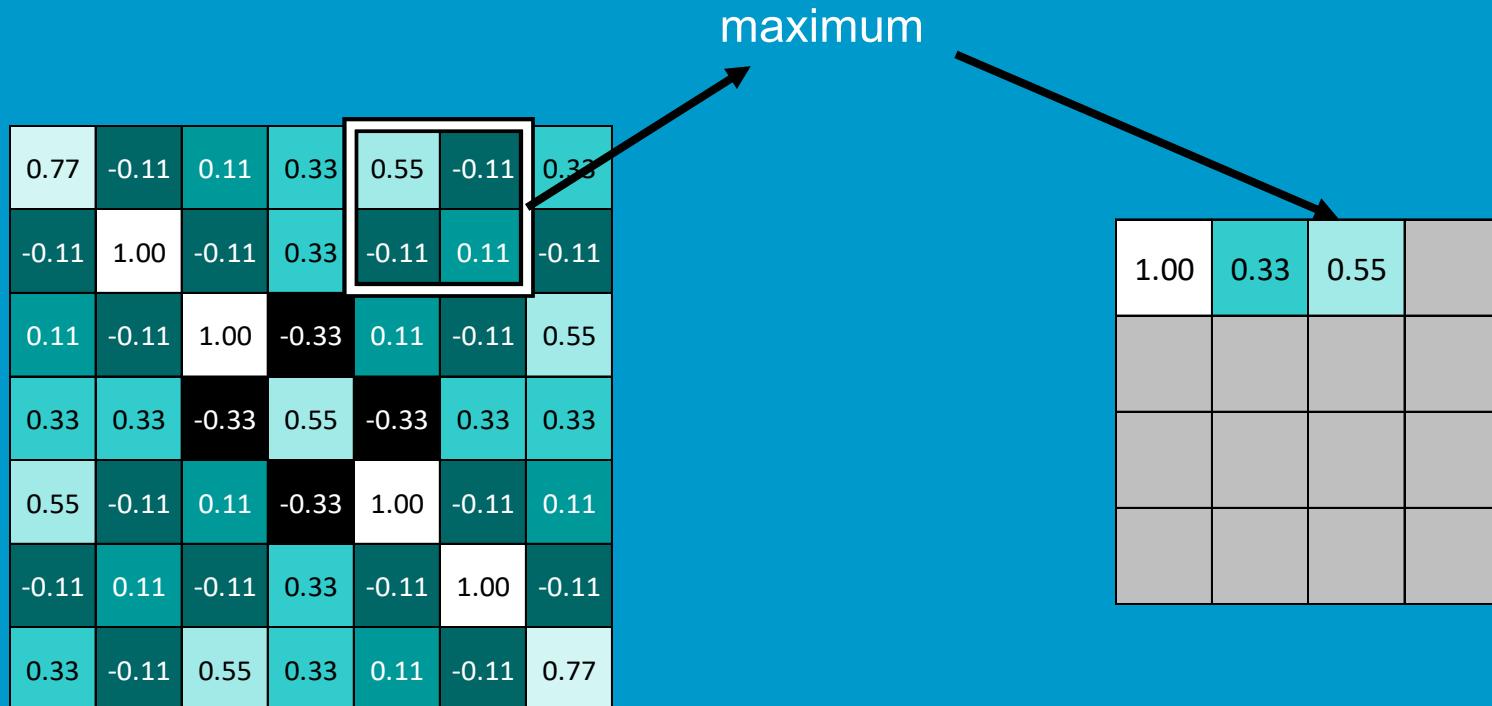
Pooling



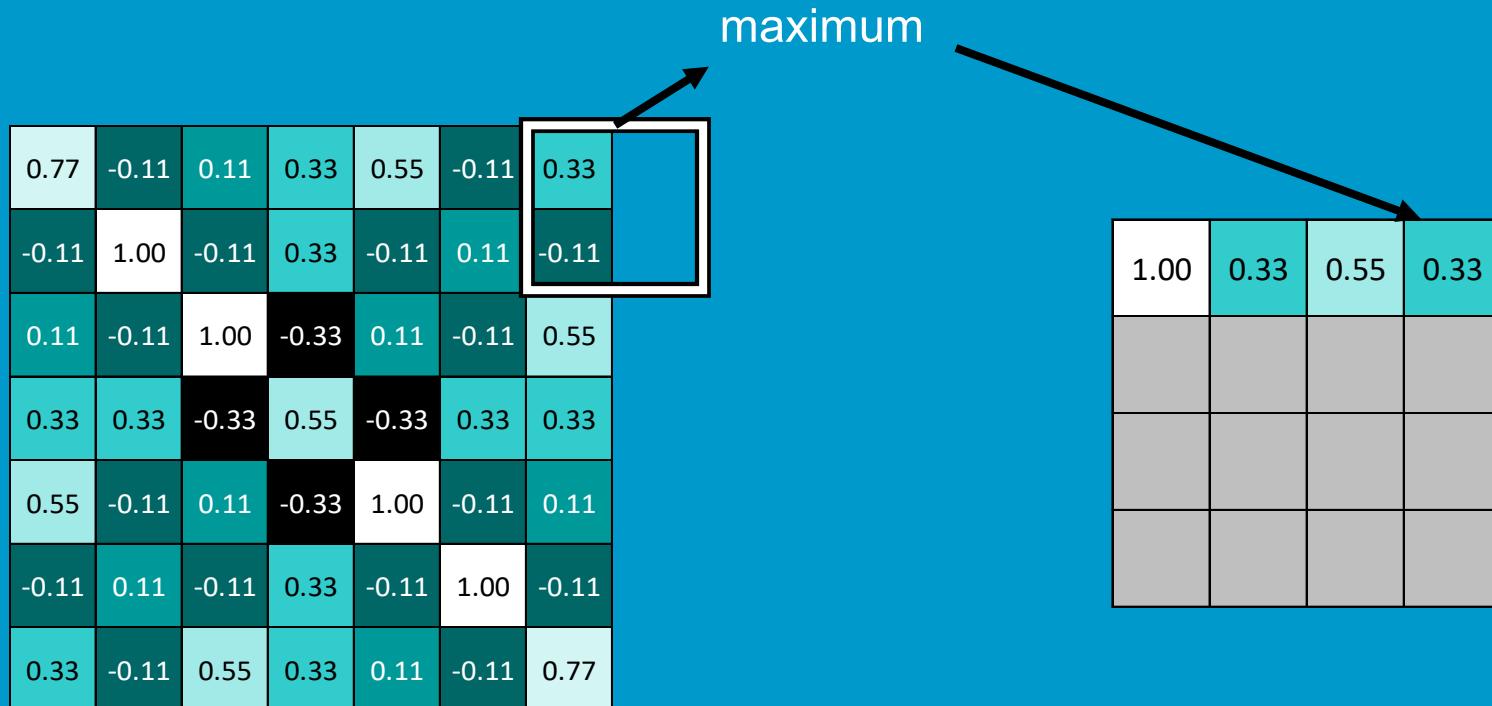
Pooling



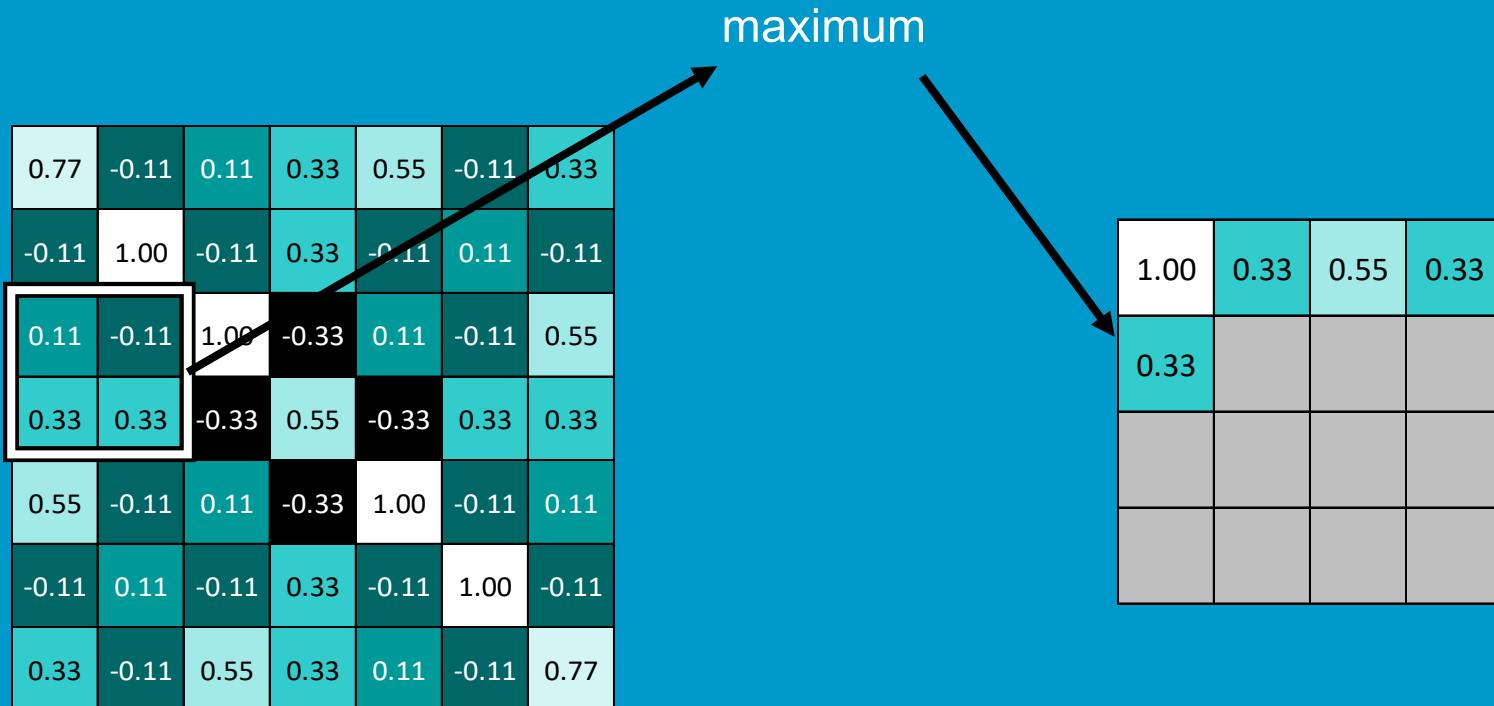
Pooling



Pooling



Pooling



Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

max pooling

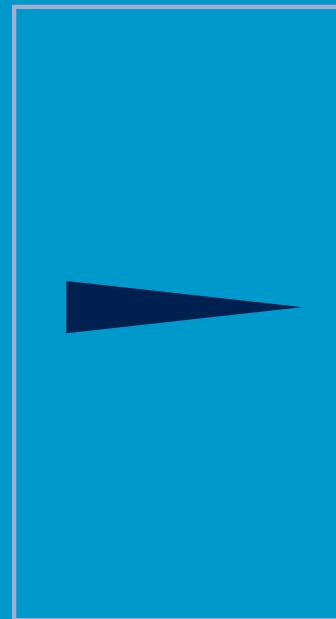


1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Entendendo a camada Pooling

Um conjunto de entradas gera saídas menores

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

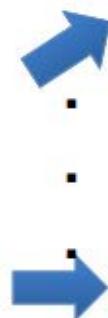


1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

Entendendo a camada Pooling



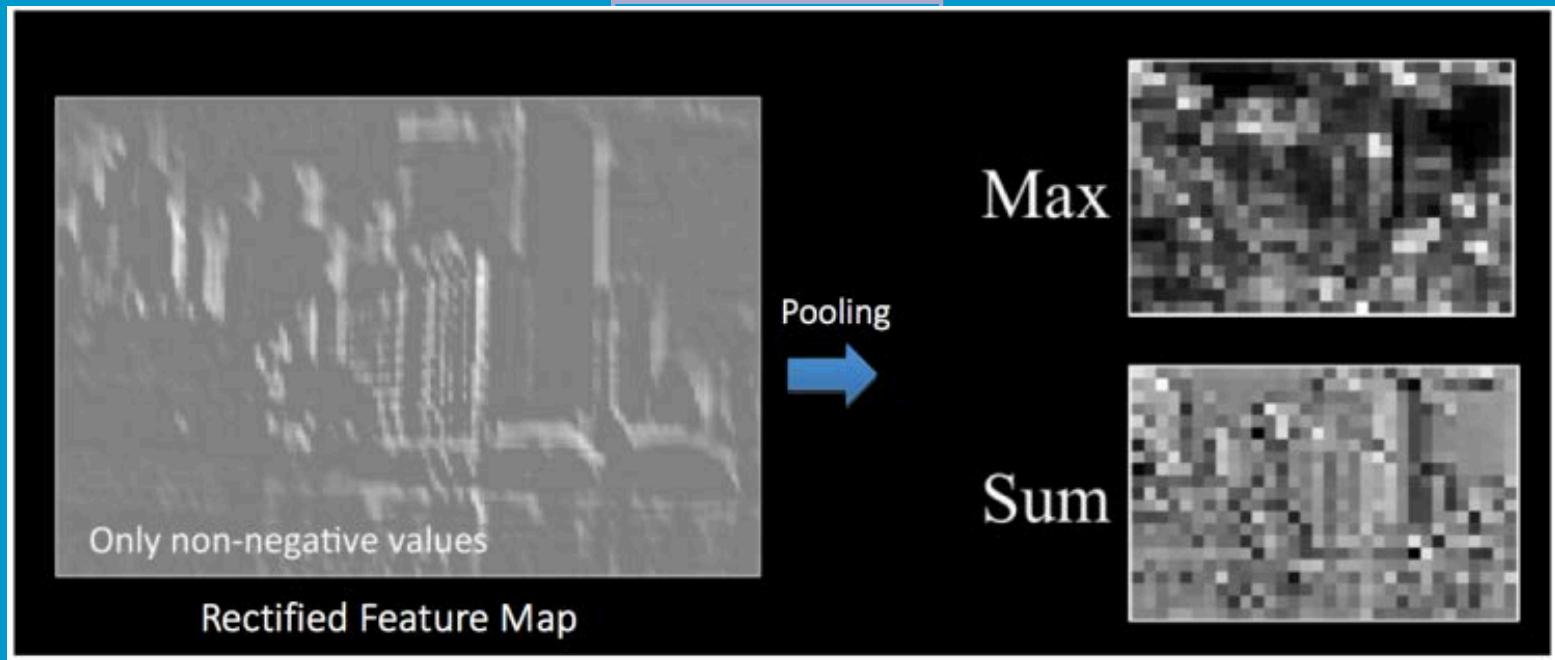
Input



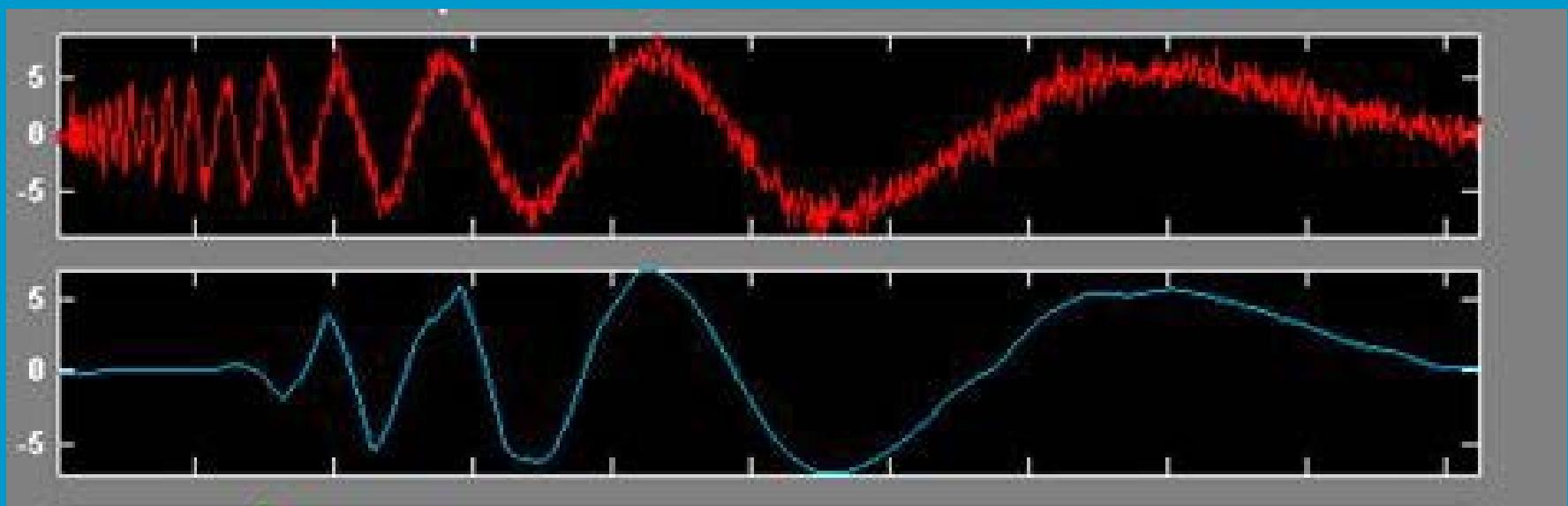
Feature Map

Entendendo a camada Pooling

Um conjunto de entradas gera saídas menores

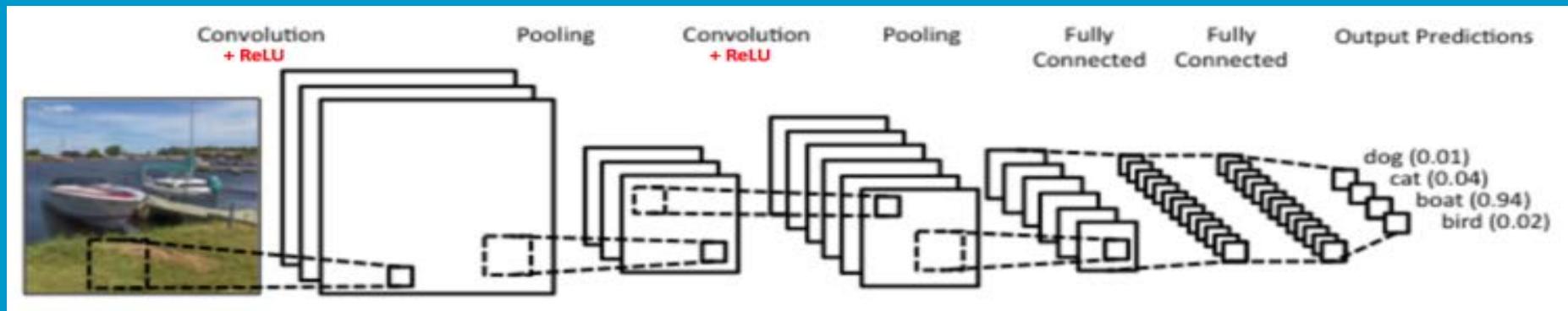


Entendendo a camada Pooling



Entendendo a camada Pooling

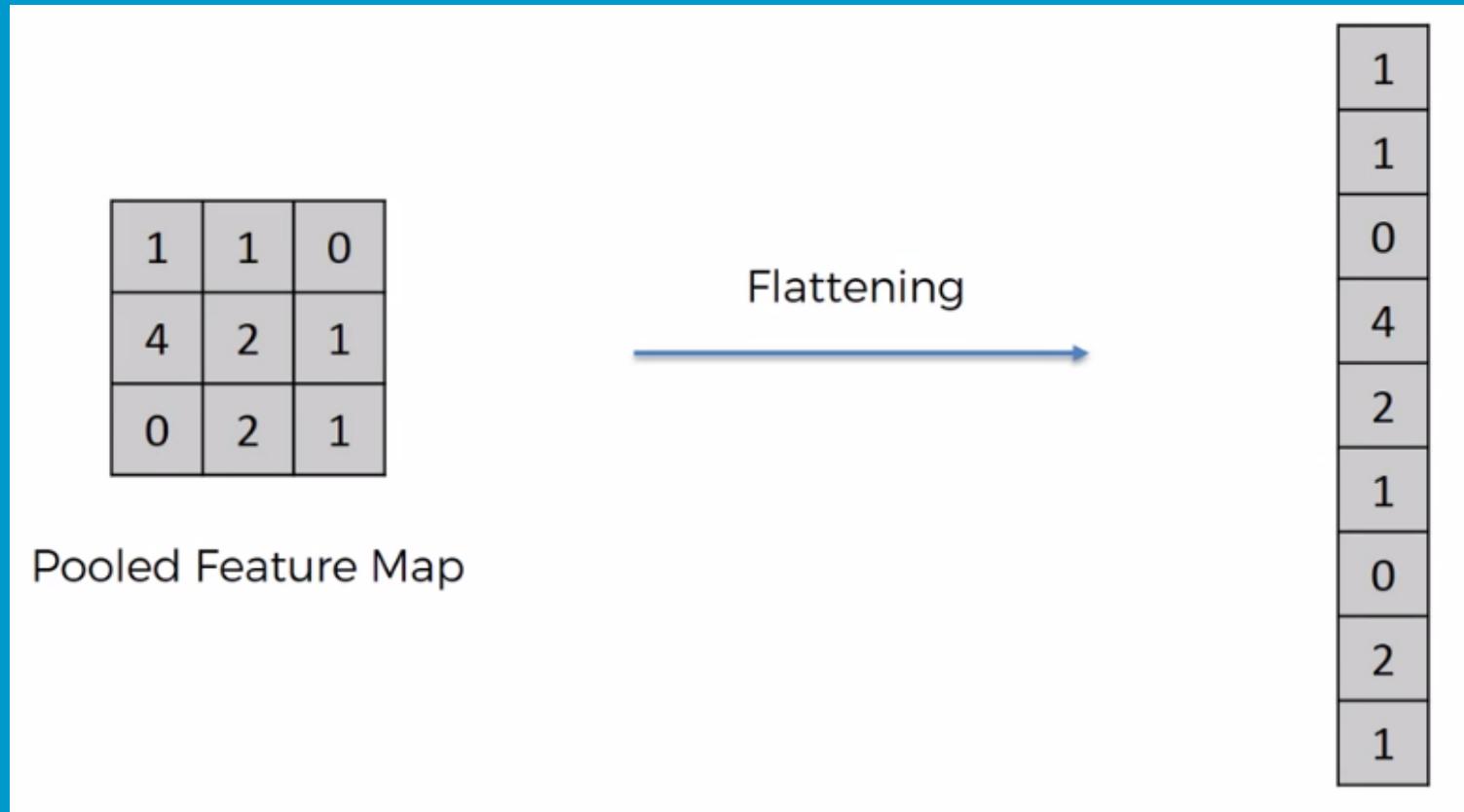




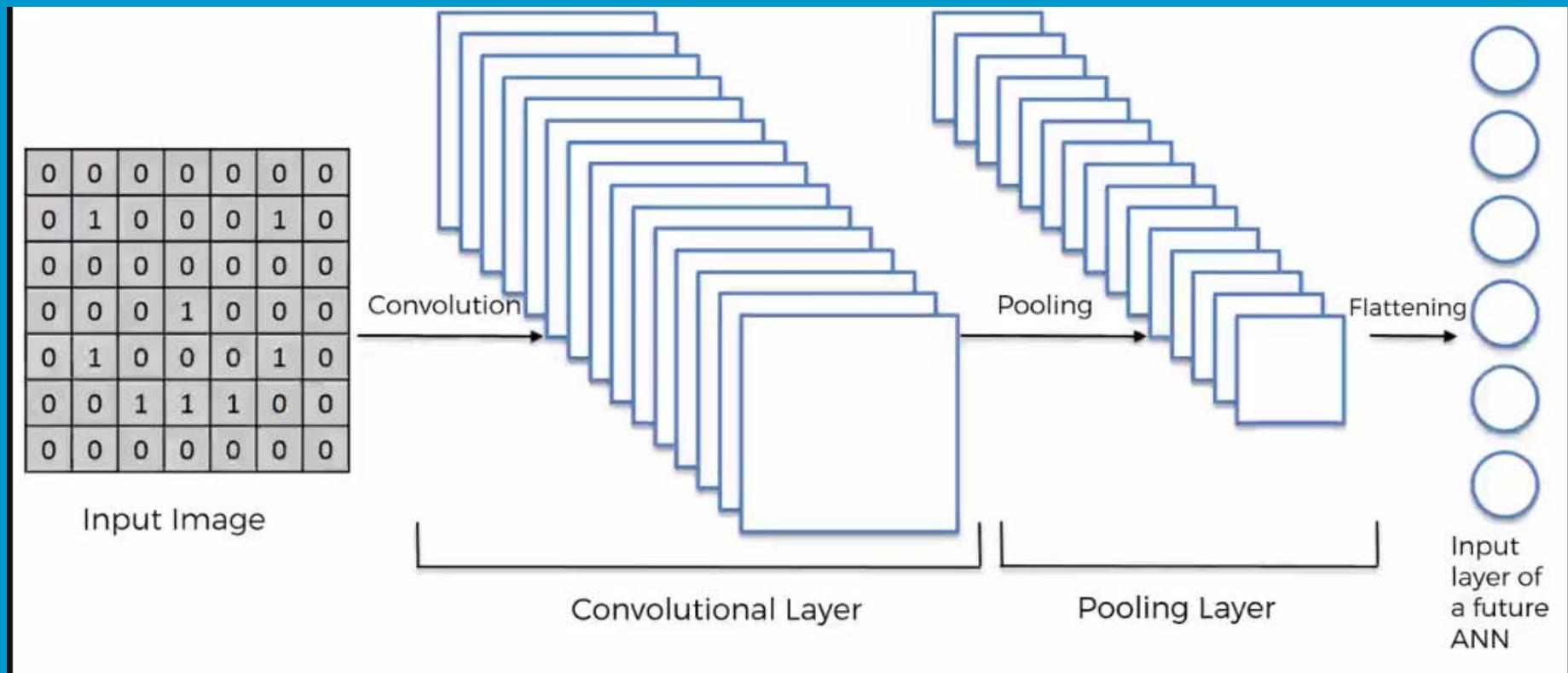
Entendendo a camada Fully Connected



Fully connected layer

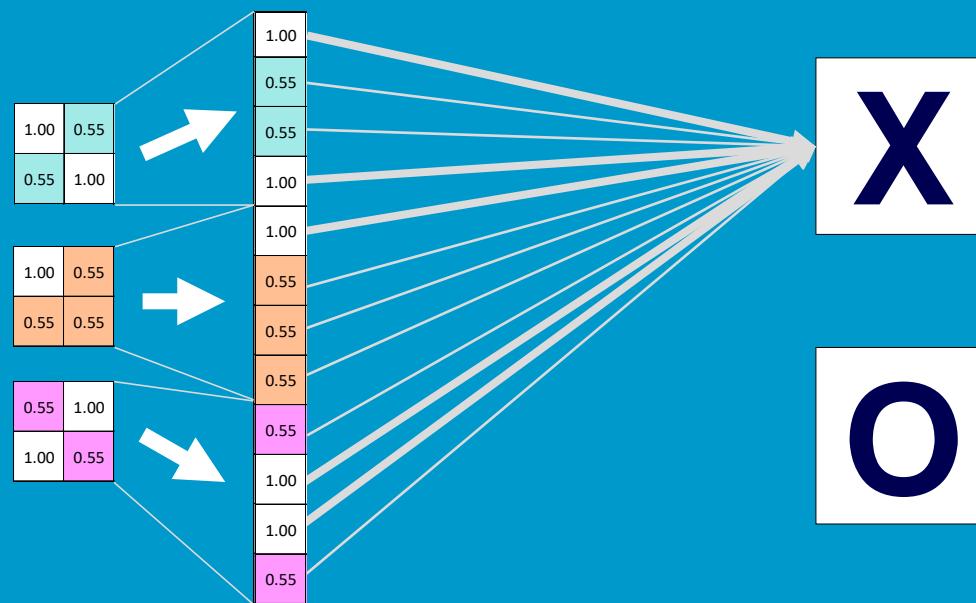


Fully connected layer



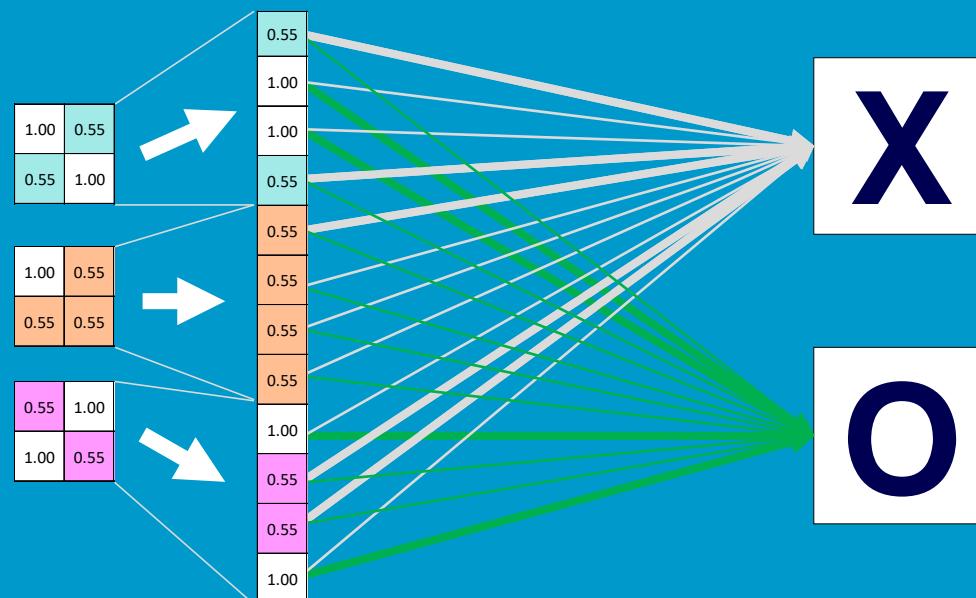
Fully connected layer

Cada valor recebido da camada pooling expressa um valor



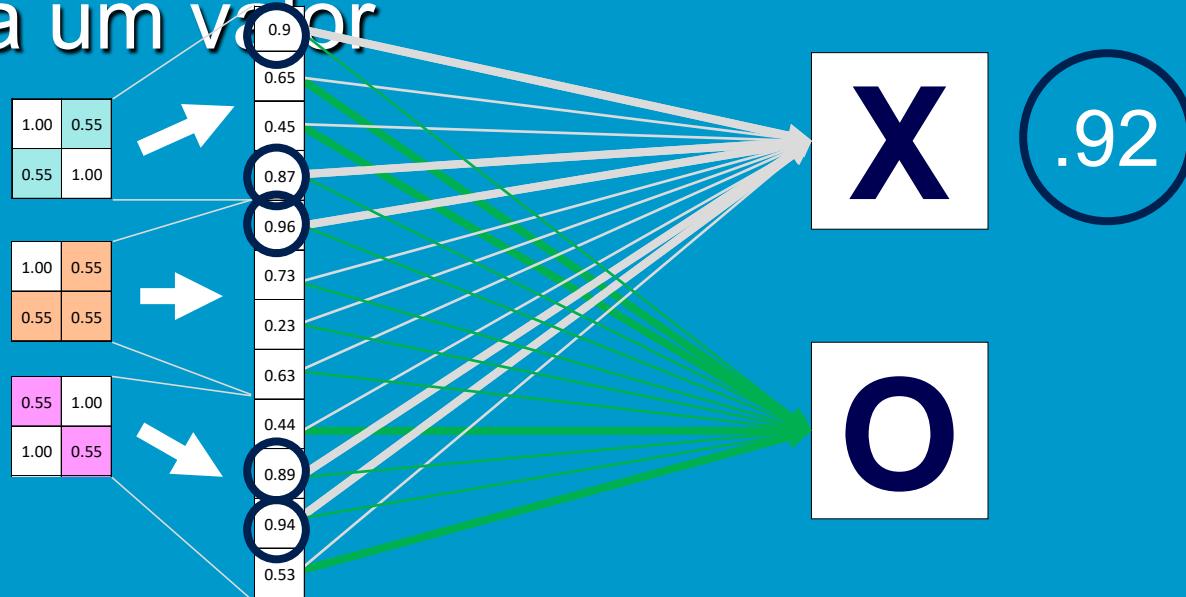
Fully connected layer

Cada valor recebido da camada pooling expressa um valor



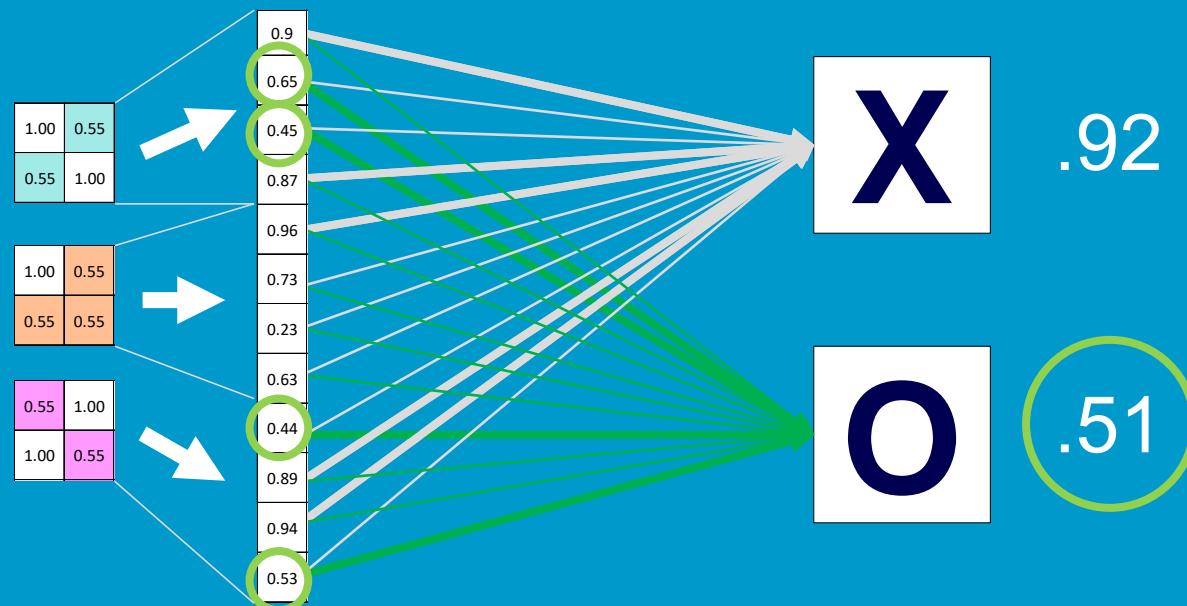
Fully connected layer

Cada valor recebido da camada pooling expressa um valor



Fully connected layer

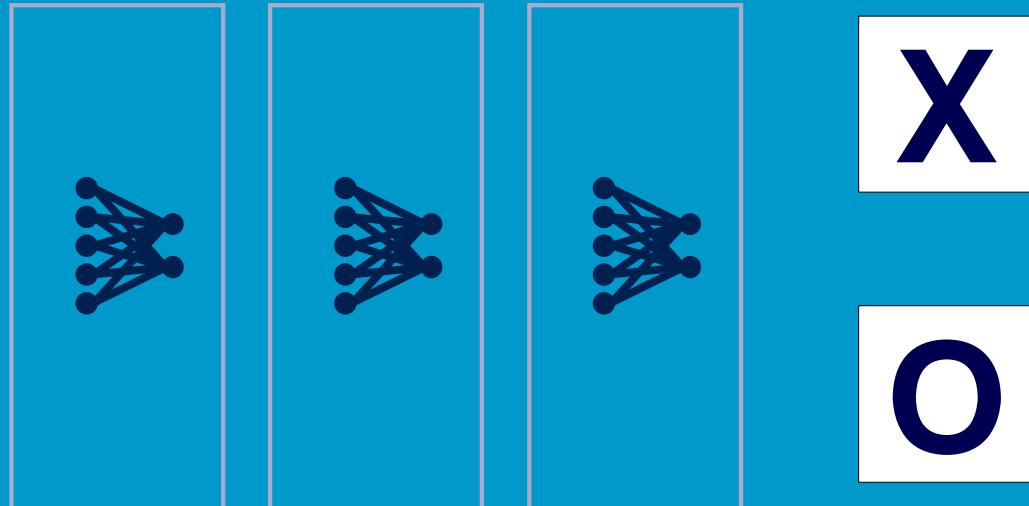
Cada valor recebido da camada pooling expressa um valor



Fully connected layer

0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53

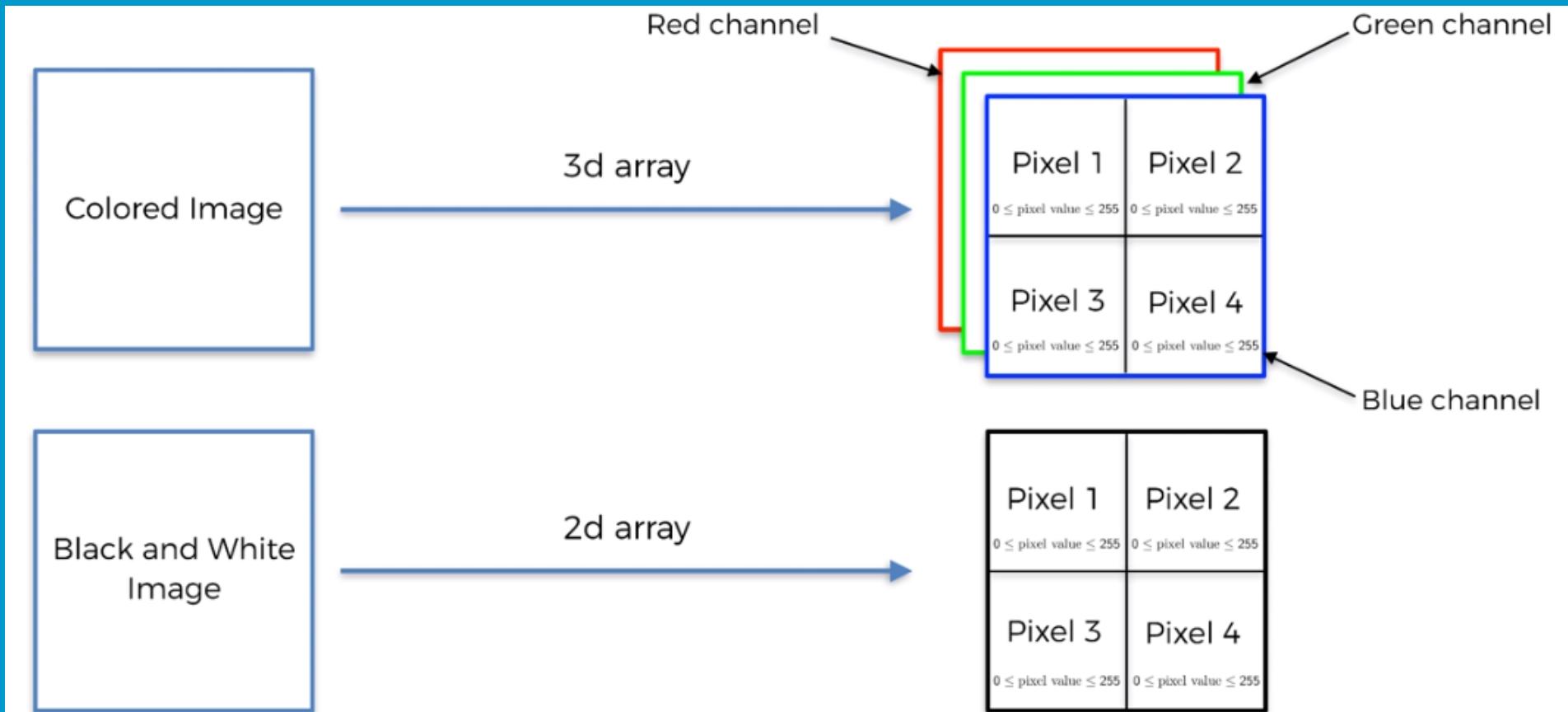
Também pode ser “empilhada”



Codificando...

```
#Considerando o problema de reconhecimento de dígitos  
# MNIST é um vetor 1-D de 784 features (28*28 pixels)  
# Reshape para [Height x Width x Channel]  
# Tensor será 4-D: [Batch Size, Height, Width, Channel]  
x = tf.reshape(x, shape=[-1, 28, 28, 1])
```

Codificando...



Codificando...

```
# Camada de convolução com 32 filtros e kernel de tamanho 5  
conv1 = tf.layers.conv2d(x, 32, 5, activation=tf.nn.relu)  
# Max Pooling (down-sampling) com strides 2 e kernel 2  
conv1 = tf.layers.max_pooling2d(conv1, 2, 2)
```

```
# Camada de convolução com 64 filtros e kernel de tamanho 3  
conv2 = tf.layers.conv2d(conv1, 64, 3, activation=tf.nn.relu)  
# Max Pooling (down-sampling) com stride 2 kernel 2  
conv2 = tf.layers.max_pooling2d(conv2, 2, 2)
```

```
# Fully connected layer  
fc1 = tf.layers.dense(fc1, 1024)  
# Camada de saída, predição de classe  
saida = tf.layers.dense(fc1, n_classes)
```

Softmax e Entropia cruzada

Softmax

Por enquanto: pense como uma outra função de ativação

$$L(\hat{f}(x), y) = -\frac{e^{z_j}}{\sum e^{z_k}}$$

Softmax e Entropia cruzada

Estamos comparando distribuições de probabilidade, logo usaremos função de entropia cruzada como a função de perda:

$$L(\hat{f}(x), y) = -\log\left(\frac{e^{z_j}}{\sum e^{z_k}}\right)$$

$$L(\hat{f}(x), y) = -\sum y_i \log f_i(x)$$

A entropia cruzada procura encontrar as hipóteses de máxima verossimilhança sob o pressuposto de que a saída observada é uma função probabilística da instância de entrada.

Softmax e Entropia cruzada

Detalharemos na próxima aula, para os ansiosos...

Leitura adicional:

A Friendly Introduction to Cross-Entropy Loss

<https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/>

Codificando...

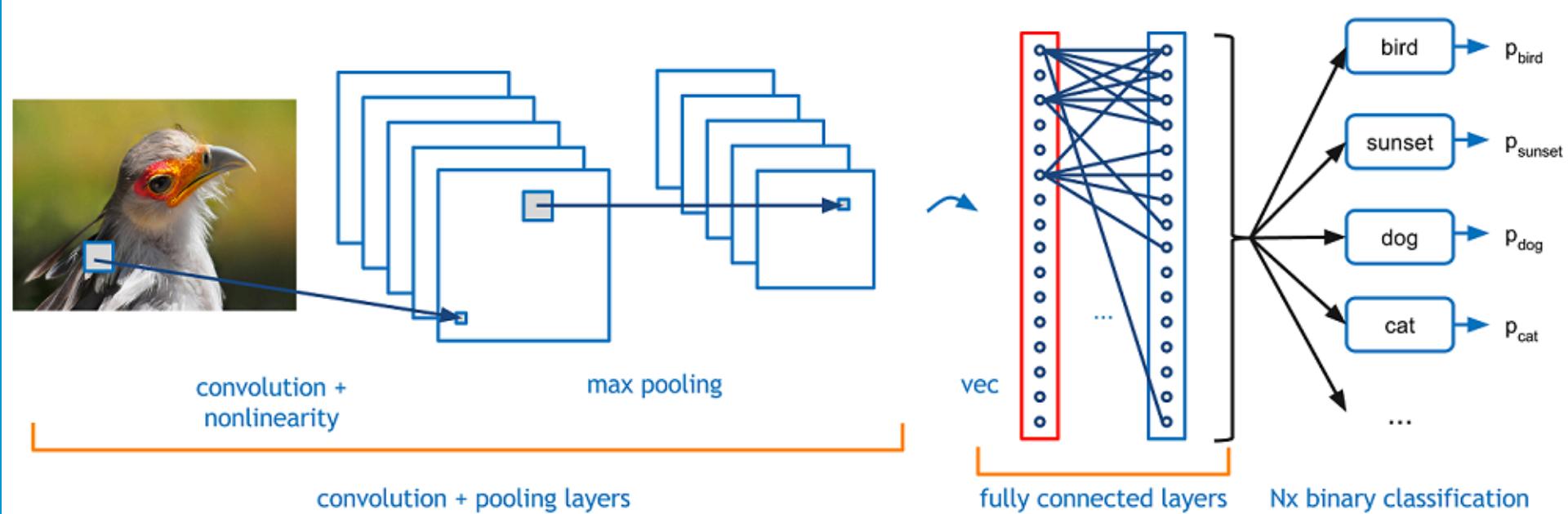
```
logits_train = conv_net(features, num_classes, dropout,  
reuse=False,is_training=True)  
logits_test = conv_net(features, num_classes, dropout,  
reuse=True, is_training=False)  
  
# Predictions  
pred_classes = tf.argmax(logits_test, axis=1)  
pred_probas = tf.nn.softmax(logits_test)
```

Codificando...

```
# Define a função de perda
loss_op =
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits_train, labels=tf.cast(labels, dtype=tf.int32)))

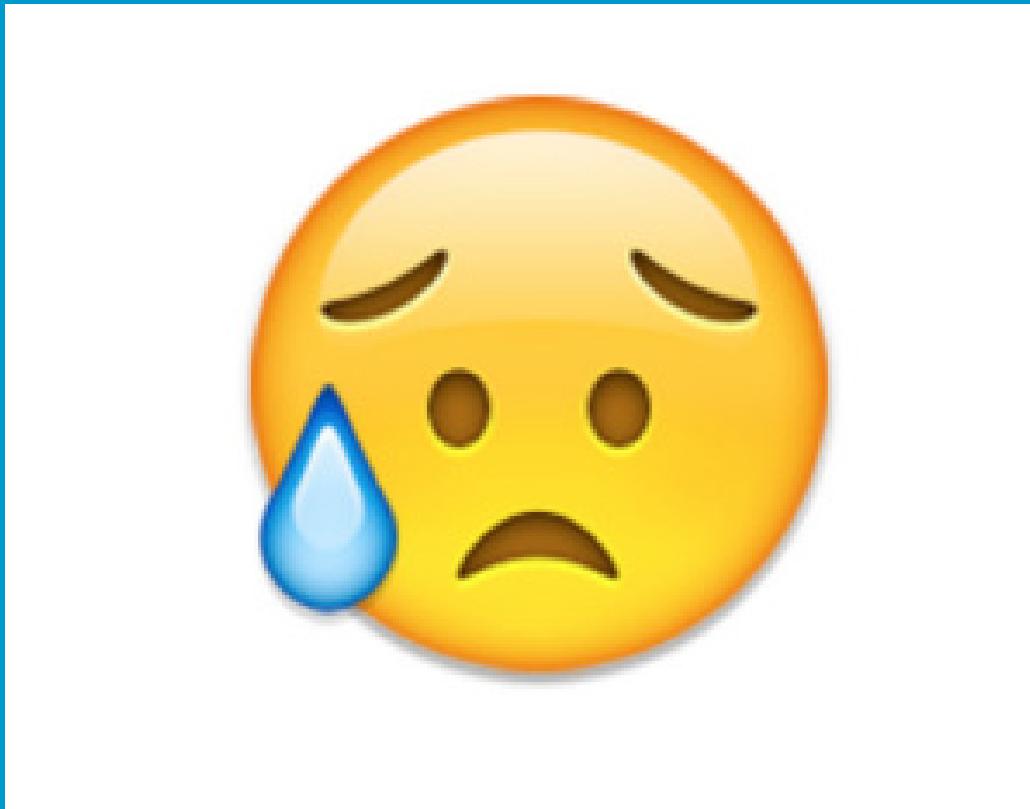
#Define o otimizador
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
train_op = optimizer.minimize(
    loss_op, global_step=tf.train.get_global_step())
```

Fully connected layer



Compreendendo o treinamento...

Tranquem a porta da sala...

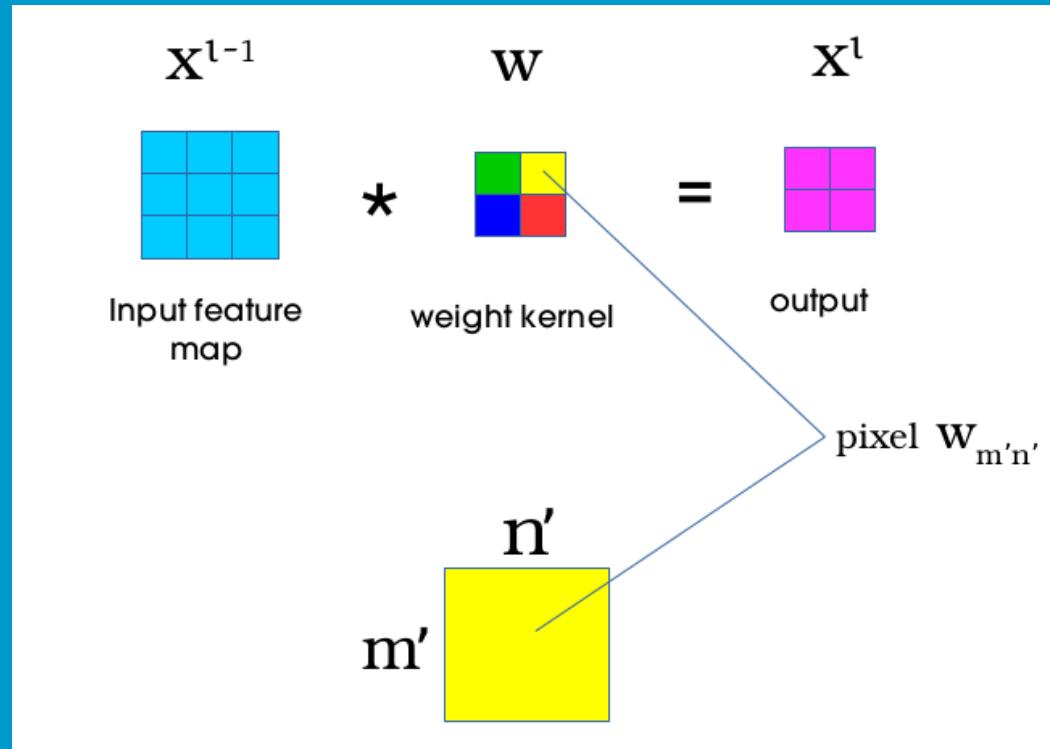


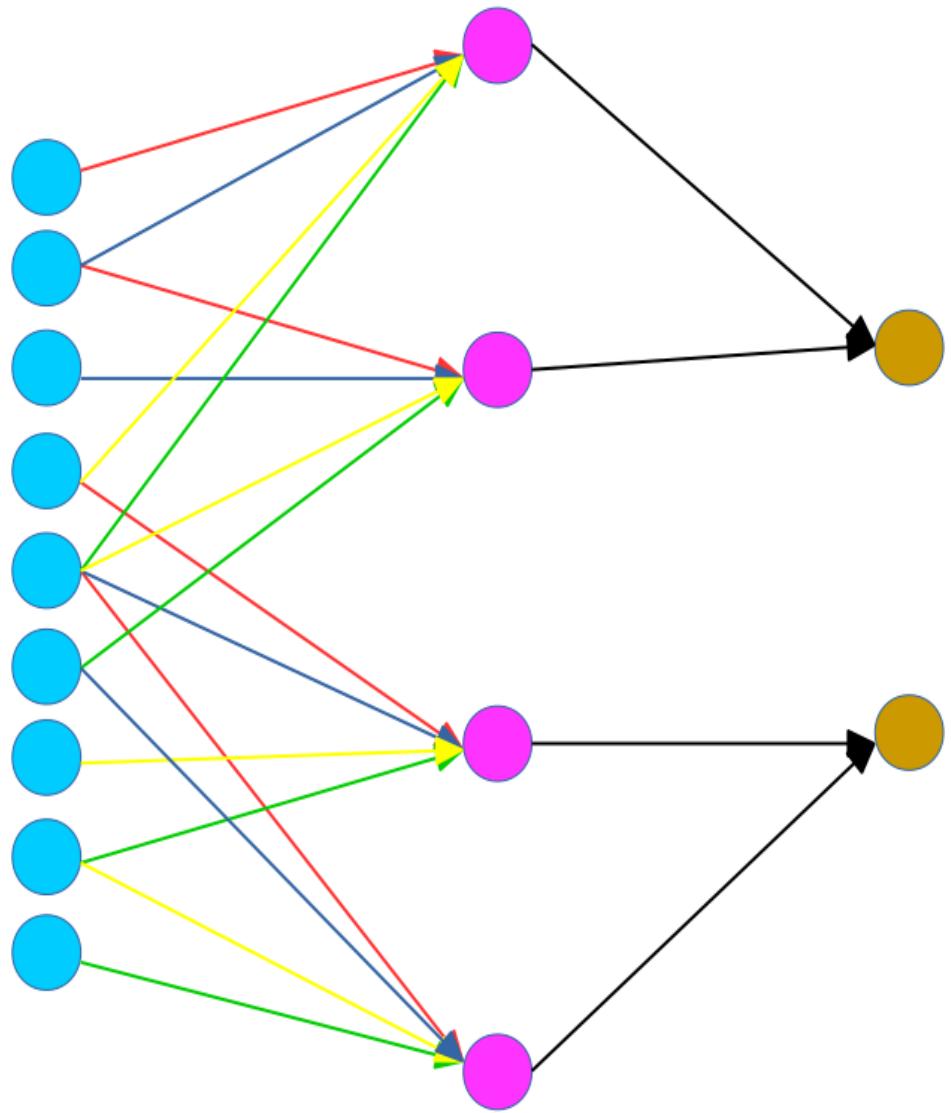
Compreendendo o treinamento...

Importante:

Atualização dos pesos e
deltas.

Como uma mudança de
um pixel no “weight
kernel” afeta a função de
perda?



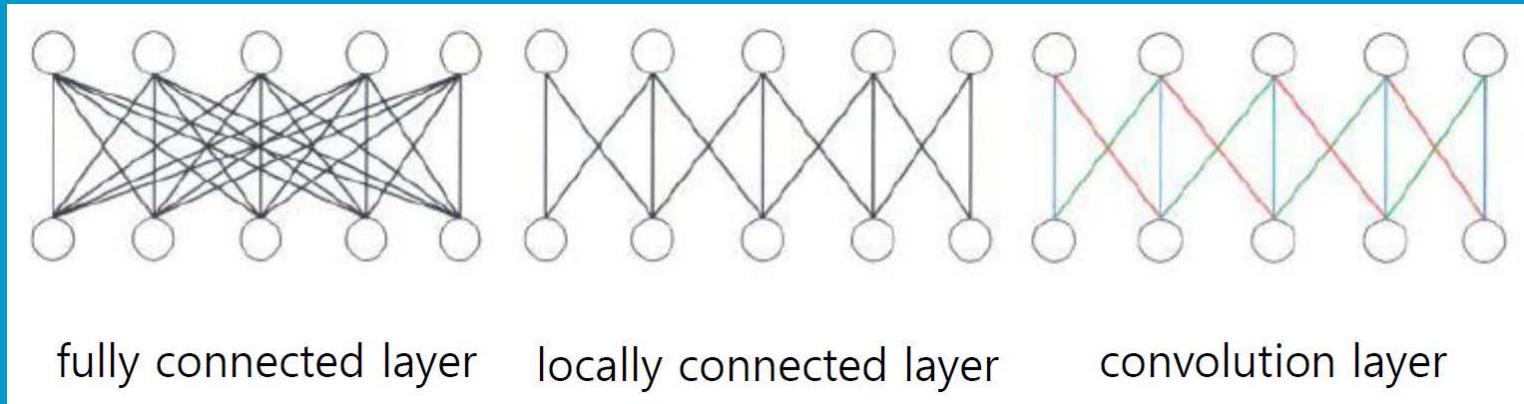


$$\begin{array}{c}
 \text{Input Feature Map} \\
 \times \quad \text{Kernel} \quad = \quad \text{Conv.} \quad | \quad \text{Pool}
 \end{array}$$

As unidades na camada convolucional têm campos receptivos de tamanho 4, portanto, somente estão conectados a 4 neurônios adjacentes na entrada.

Idéia de conectividade esparsa em CNNs onde existe padrão de conectividade local entre neurônios em camadas adjacentes.

Compreendendo o treinamento...



fully connected layer

locally connected layer

convolution layer

Pesos diferentes

Pesos diferentes

Pesos compartilhados

- **Logo, camada convolucional** possui um número muito menor de parâmetros em razão da conexão local e do compartilhamento de pesos

Compreendendo o treinamento...

No final teremos :

$$\begin{aligned}\frac{\partial E}{\partial w_{m',n'}^l} &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^l o_{i+m',j+n'}^{l-1} \\ &= \text{rot}_{180^\circ} \left\{ \delta_{i,j}^l \right\} * o_{m',n'}^{l-1}\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial x_{i',j'}^l} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f' \left(x_{i',j'}^l \right) \\ &= \text{rot}_{180^\circ} \left\{ \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'+m,j'+n}^{l+1} w_{m,n}^{l+1} \right\} f' \left(x_{i',j'}^l \right) \\ &= \delta_{i',j'}^{l+1} * \text{rot}_{180^\circ} \left\{ w_{m,n}^{l+1} \right\} f' \left(x_{i',j'}^l \right)\end{aligned}$$

Compreendendo o treinamento

Considere o caso 1D:

$$X^{l-1} = [a \ b \ c \ d \ e] \text{ (Entrada)}$$

$$K = [k_1 \ k_2] \text{ (Kernel)}$$

$$X^l = [a*k_1 + b*k_2 \ b*k_1 + c*k_2 \ c*k_1 + d*k_2 \ d*k_1 + e*k_2 \ e*k_1] \text{ (Saída)}$$

Agora considere

$$\frac{\partial X^l}{\partial k_1} = [0 \ k_2 \ k_1 \ 0 \ 0] \text{ -- Kernel "flipped".}$$

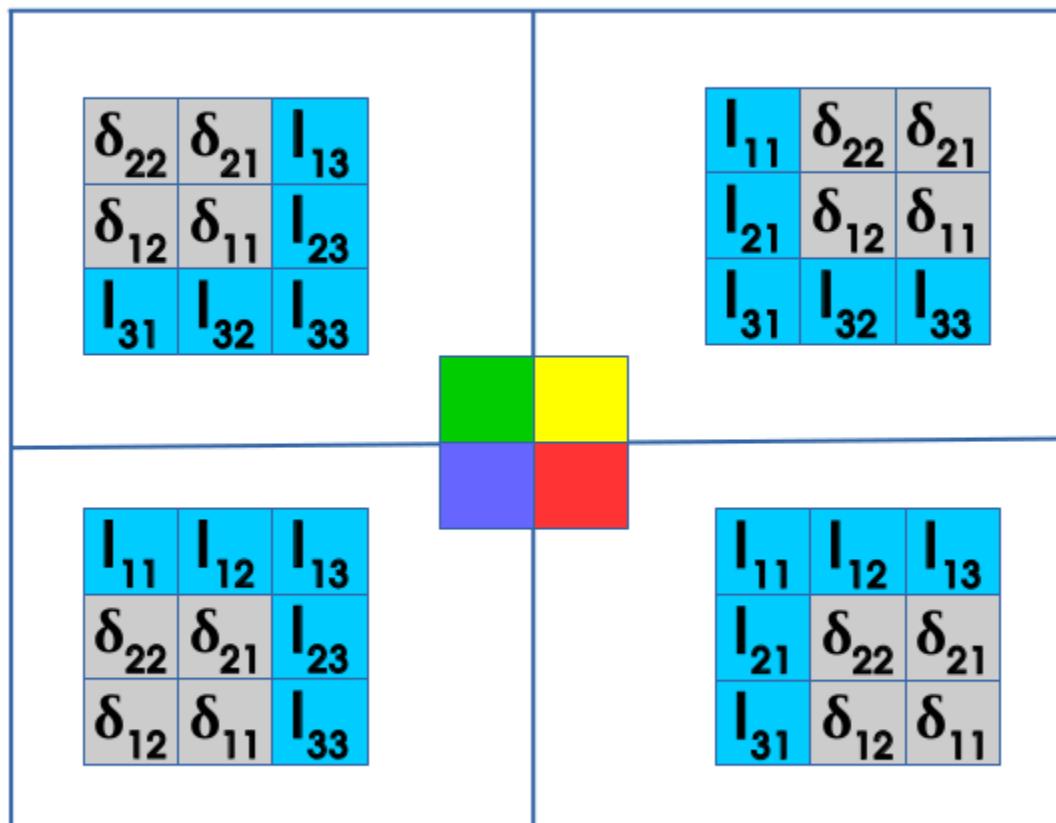
δ_{11}	δ_{12}
δ_{21}	δ_{22}

→

Error matrix Flipped 180°

δ_{22}	δ_{21}
δ_{12}	δ_{11}

$$\delta_{i,j}^l = \frac{\partial E}{\partial x_{i,j}^l}$$



Compreendendo o treinamento...

Dicas de leitura:

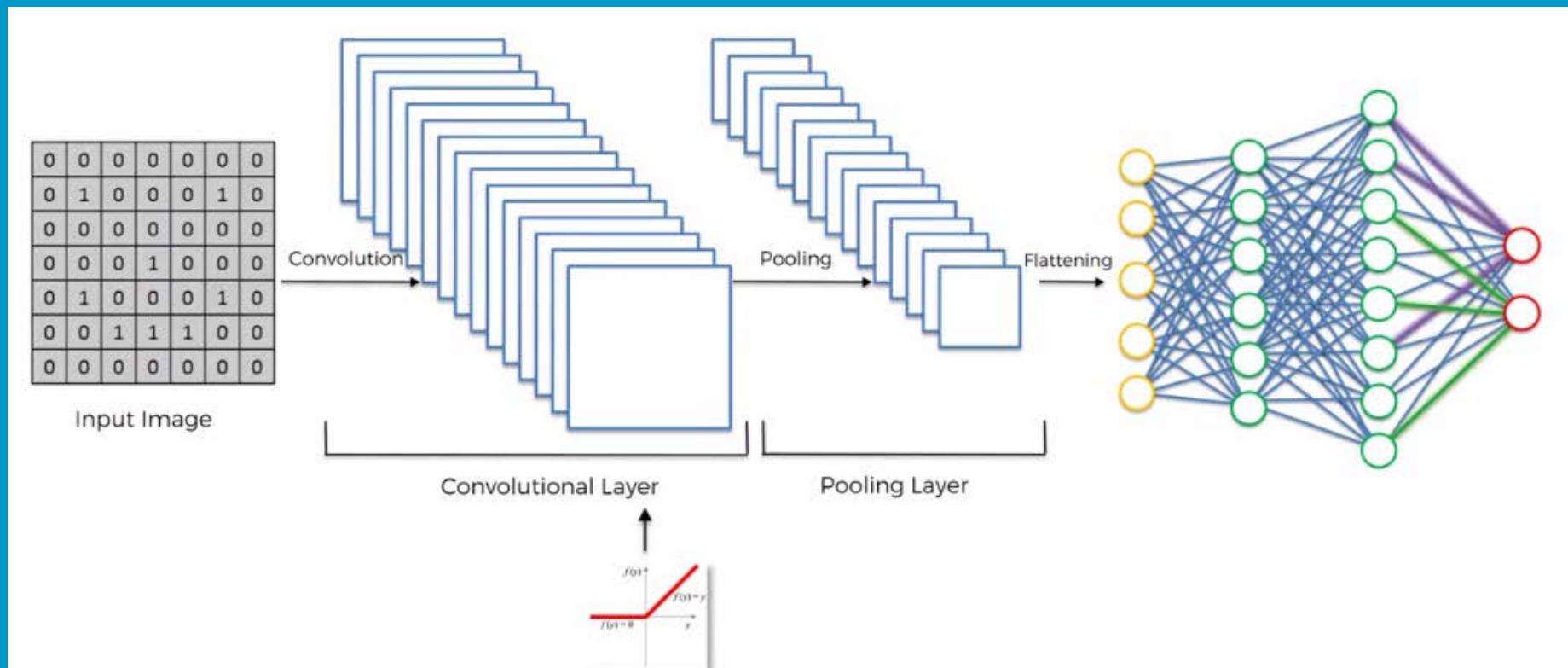
A guide to convolution arithmetic for deep learning. stat 1050 (2016) -

<https://arxiv.org/pdf/1603.07285.pdf>

<http://jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>



Resumindo...



Resumindo...

Camada de convolução com 32 filtros e kernel de tamanho 5
conv1 = tf.layers.conv2d(x, **32**, **5**, activation=tf.nn.relu)

Max Pooling (down-sampling) com strides 2 e kernel 2
conv1 = tf.layers.max_pooling2d(conv1, **2**, **2**)

Camada de convolução com 64 filtros e kernel de tamanho 3
conv2 = tf.layers.conv2d(conv1, **64**, **3**, activation=tf.nn.relu)

Max Pooling (down-sampling) com stride 2 kernel 2
conv2 = tf.layers.max_pooling2d(conv2, **2**, **2**)

Keras



or theano

Interface para
TensorFlow ou Theano

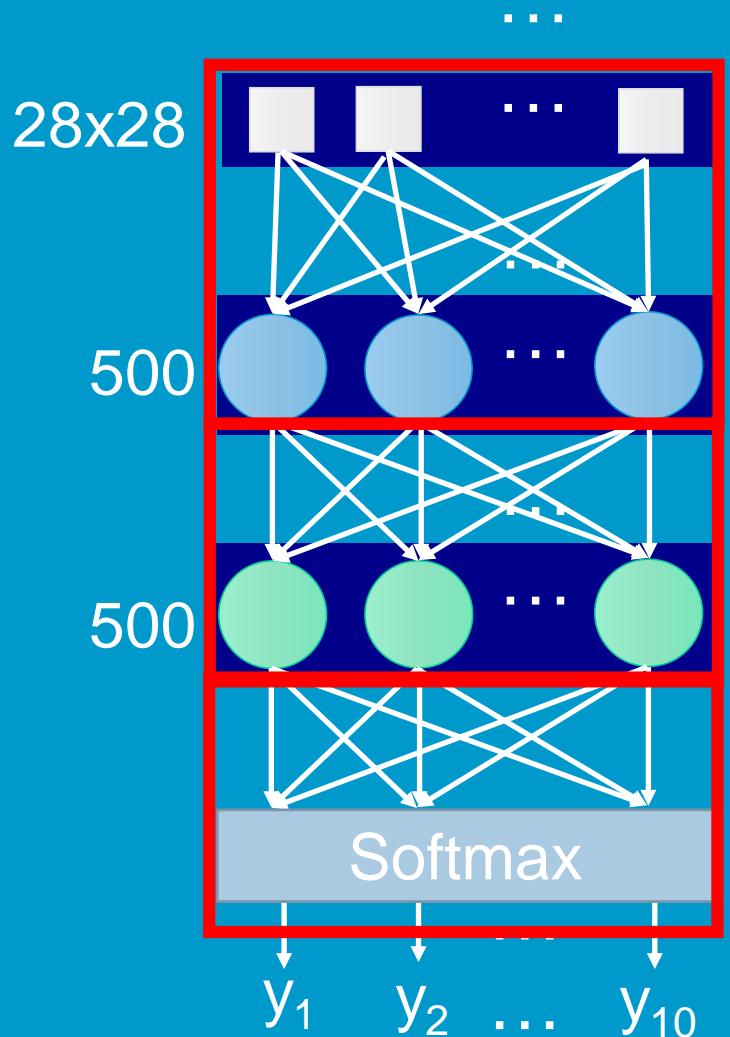


Documentação: <http://keras.io/>

Exemplos:

<https://github.com/fchollet/keras/tree/master/examples>

Keras



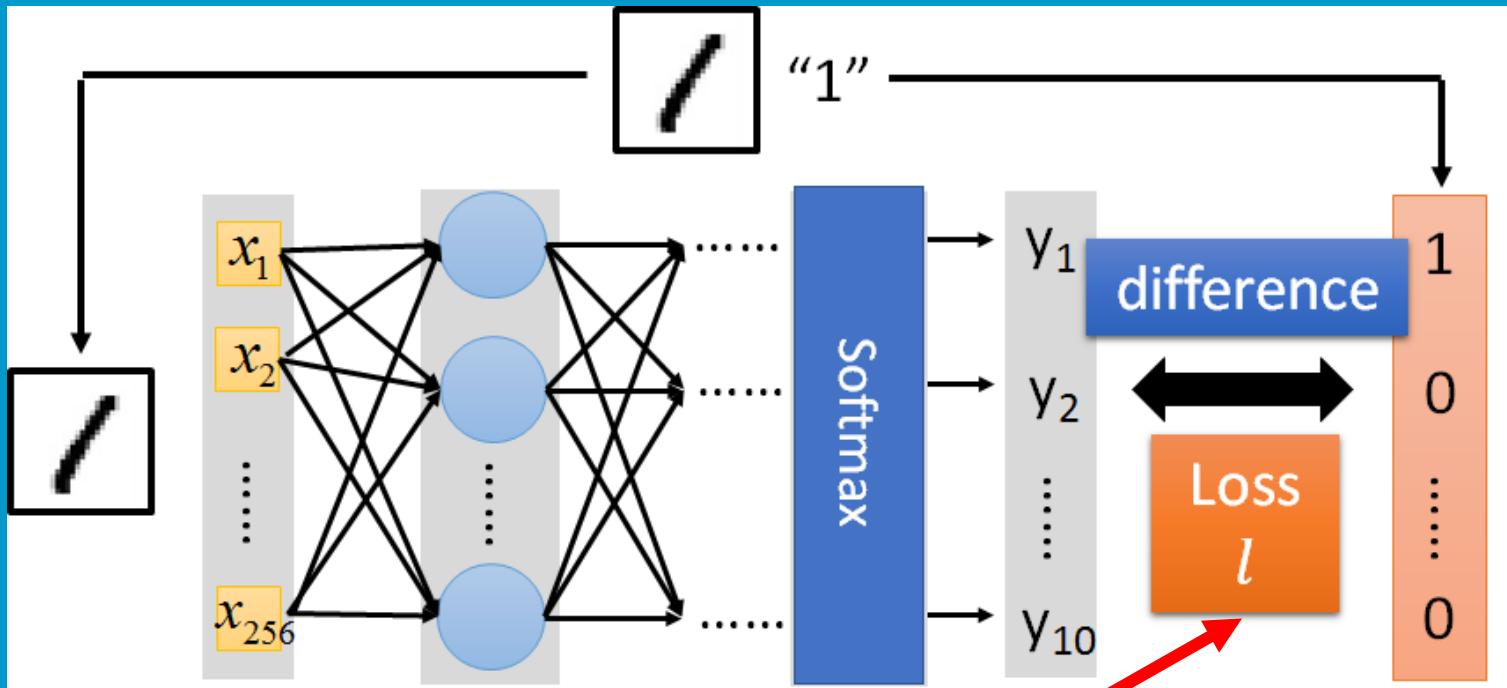
```
model = Sequential()  
model.add( Dense( input_dim=28*28,  
                  output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

softplus, softsign, relu, tanh,
hard_sigmoid, linear

```
model.add( Dense( output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

```
model.add( Dense( output_dim=10 ) )  
model.add( Activation('softmax') )
```

Keras



```
model.compile(loss='categorical_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

Several alternatives: <https://keras.io/objectives/>

Keras

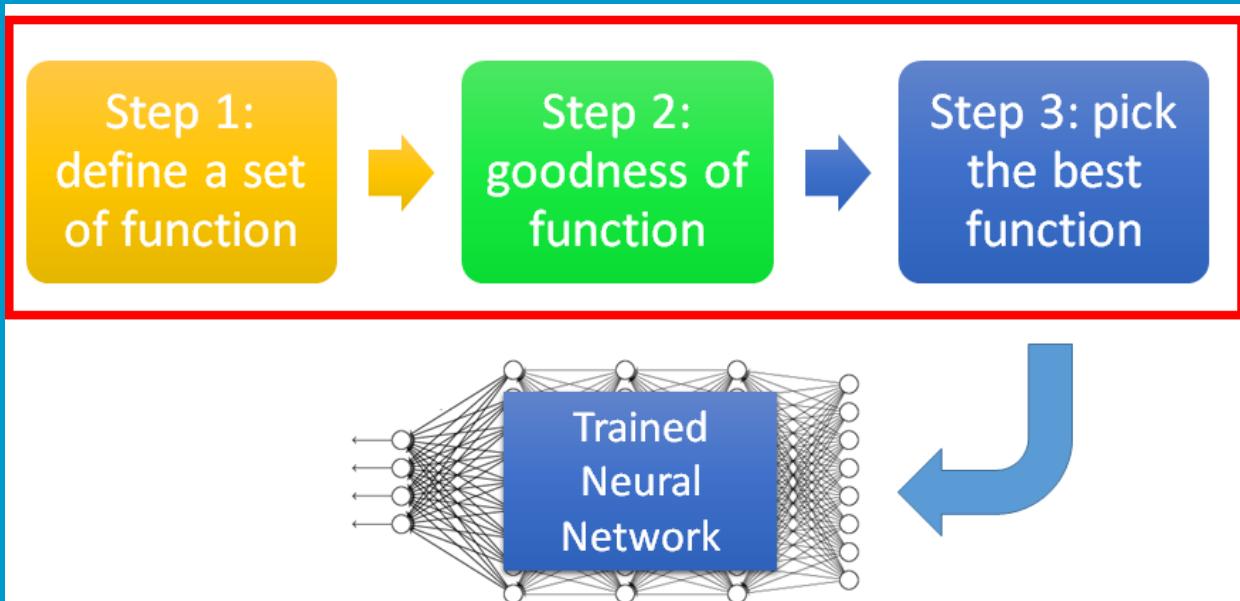
```
model.compile(loss='categorical_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

**SGD, RMSprop, Adagrad, Adadelta, Adam,
Adamax, Nadam**

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Entrada
(Imagens)

Labels
(dgitos)



Salvando a rede neural

<http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model>

```
score = model.evaluate(x_test, y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

```
result = model.predict(x_test)
```

Keras

```
model = Sequential()
```

```
model.add(Conv2D(32, (3, 3), padding='same',  
    input_shape=x_train.shape[1:]))
```

```
model.add(Activation('relu'))
```

```
model.add(Conv2D(32, (3, 3)))
```

```
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(64, (3, 3), padding='same'))
```

```
model.add(Activation('relu'))
```

```
model.add(Conv2D(64, (3, 3)))
```

```
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Keras

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
# Inicializa o otimizador
opt = keras.optimizers.SGD(lr=0.0001)
```

```
# Treina o modelo
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

Juntando tudo...

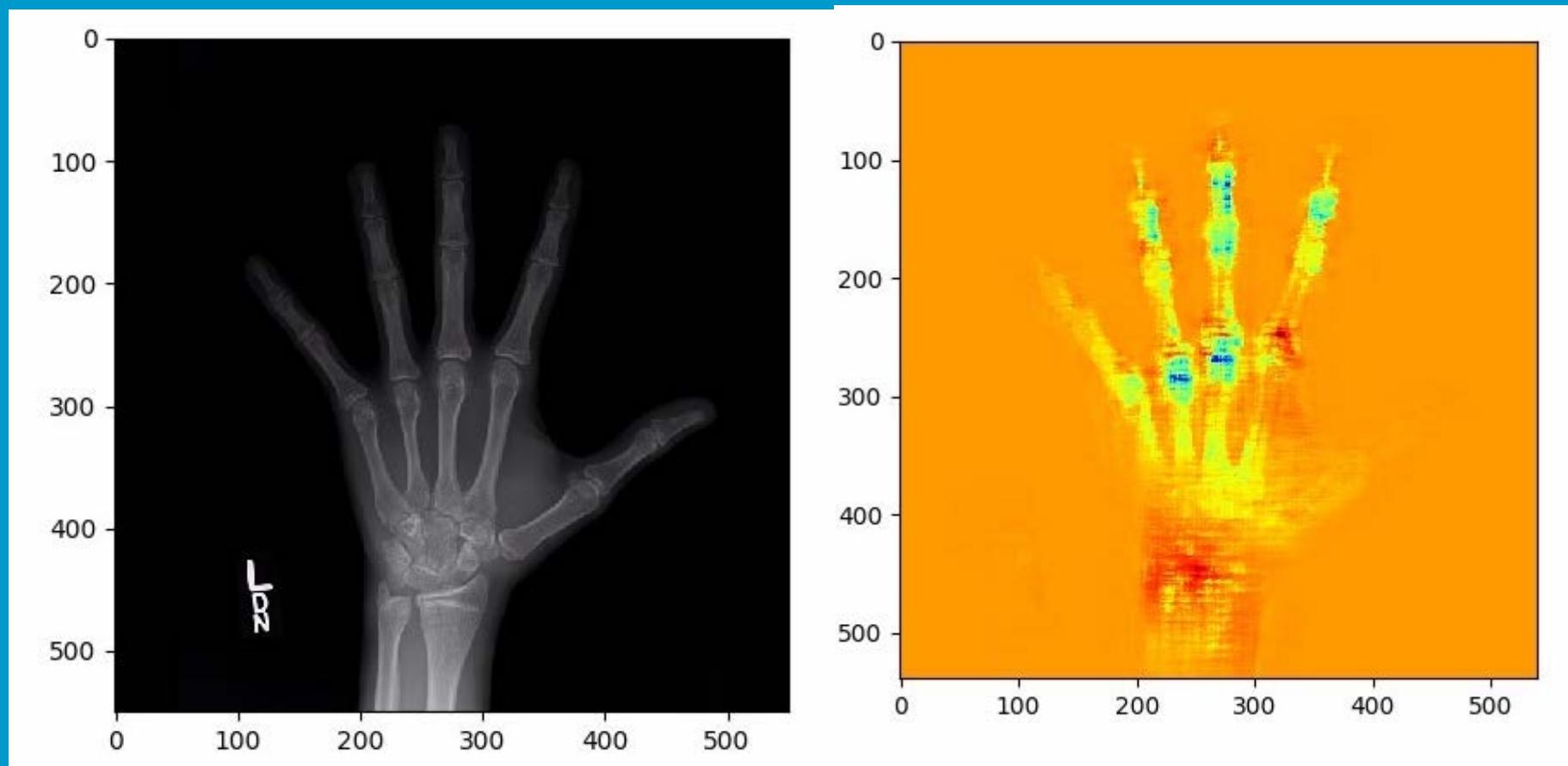


<http://www.cs.cmu.edu/~aharley/vis/>

Demonstração vídeo







Resumindo a primeira parte dos parâmetros

Convolution

Filtro a ser utilizado

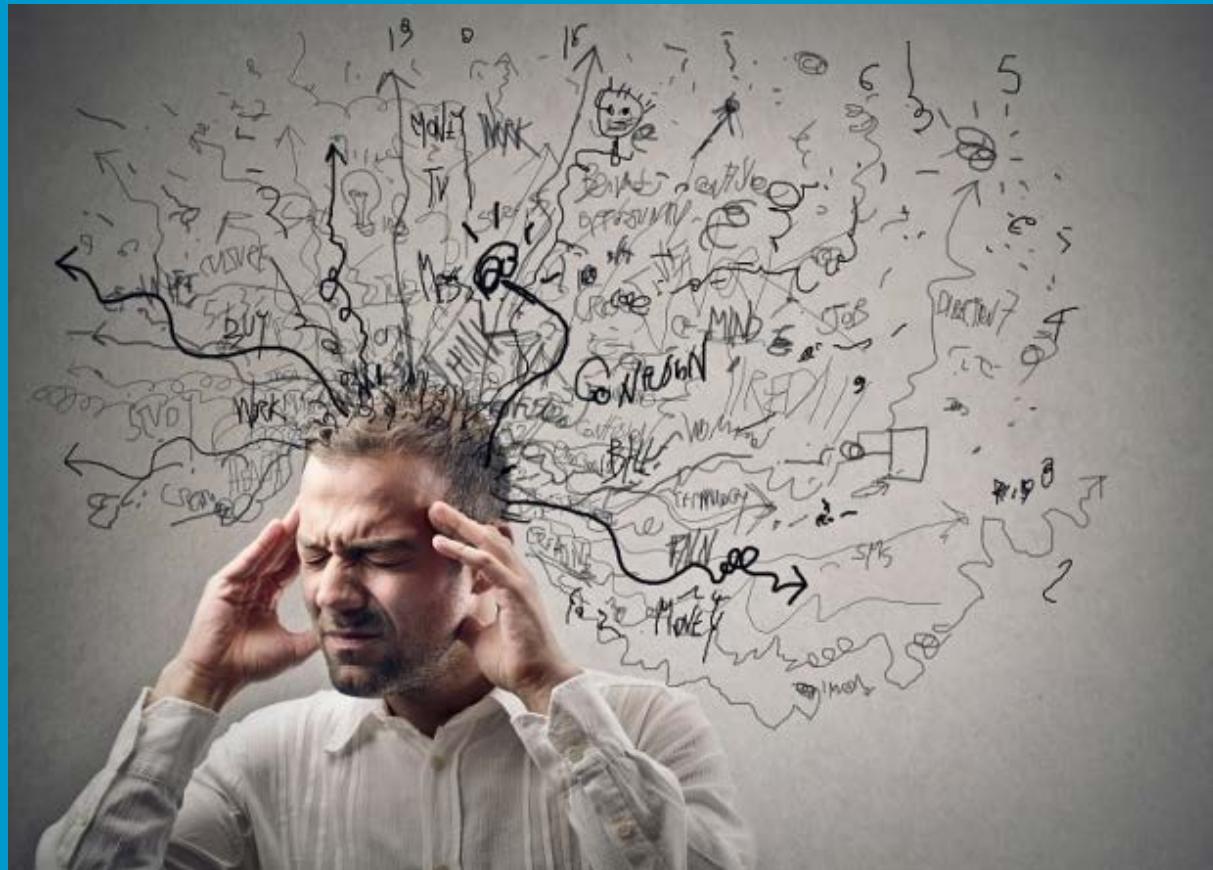
Pooling

Tamanho da janela

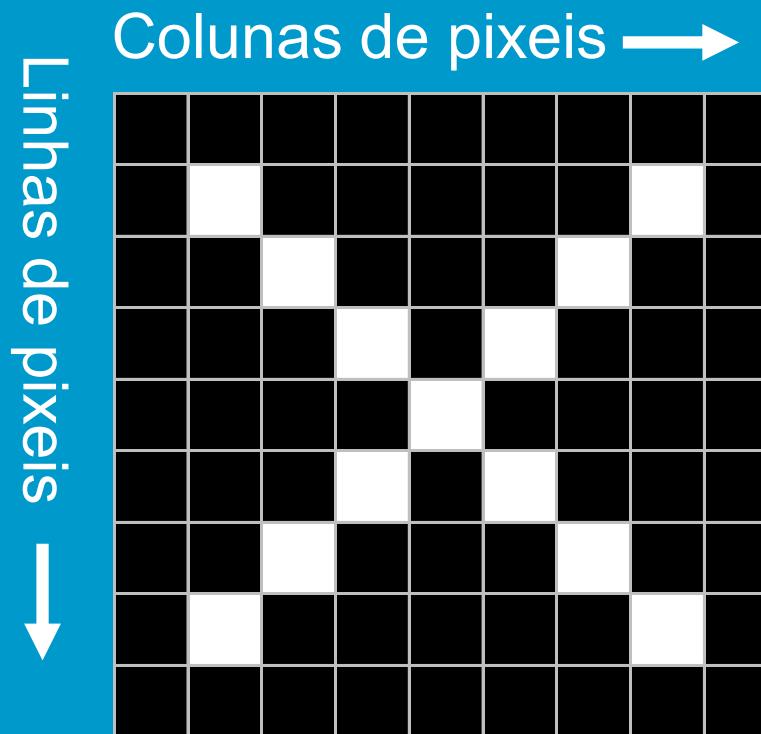
Fully Connected

Número de neurônios

Possibilidades de aplicações

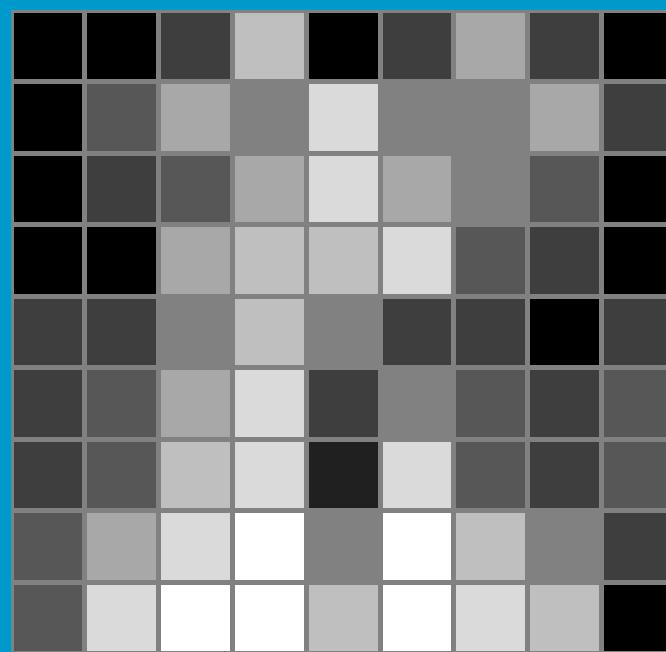


Imagens



Sons

Intervalos de tempo →



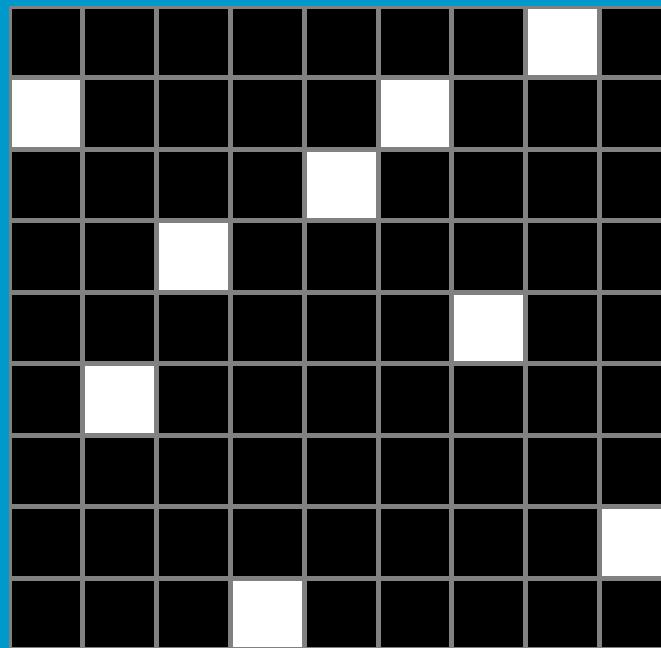
Intensidade em cada frequência de banda

Texto

Posição na sentença



Palavras no dicionário



Dados de cliente

Nome, idade,
endereço, email,
financeiro,
Atividade na internet,...

Clientes



A	22	1A	<u>a@a</u>	1	aa	a1.a	123	aa1
B	33	2B	<u>b@b</u>	2	bb	b2.b	234	bb2
C	44	3C	<u>c@c</u>	3	cc	c3.c	345	cc3
D	55	4D	<u>d@d</u>	4	dd	d4.d	456	dd4
E	66	5E	<u>e@e</u>	5	ee	e5.e	567	ee5
F	77	6F	<u>f@f</u>	6	ff	f6.f	678	ff6
G	88	7G	<u>g@g</u>	7	gg	g7.g	789	gg7
H	99	8H	<u>h@h</u>	8	hh	h8.h	890	hh8
I	111	9I	<u>i@i</u>	9	ii	i9.i	901	ii9

Uma nova era....

Digit Recognizer

Facial Keypoints Detection

Grasp-and-Lift EEG Detection

Diabetic Retinopathy Detection

National Data Science Bowl

Right Whale Recognition

The Merck Molecular Activity Challenge

The Job Salary prediction challenge

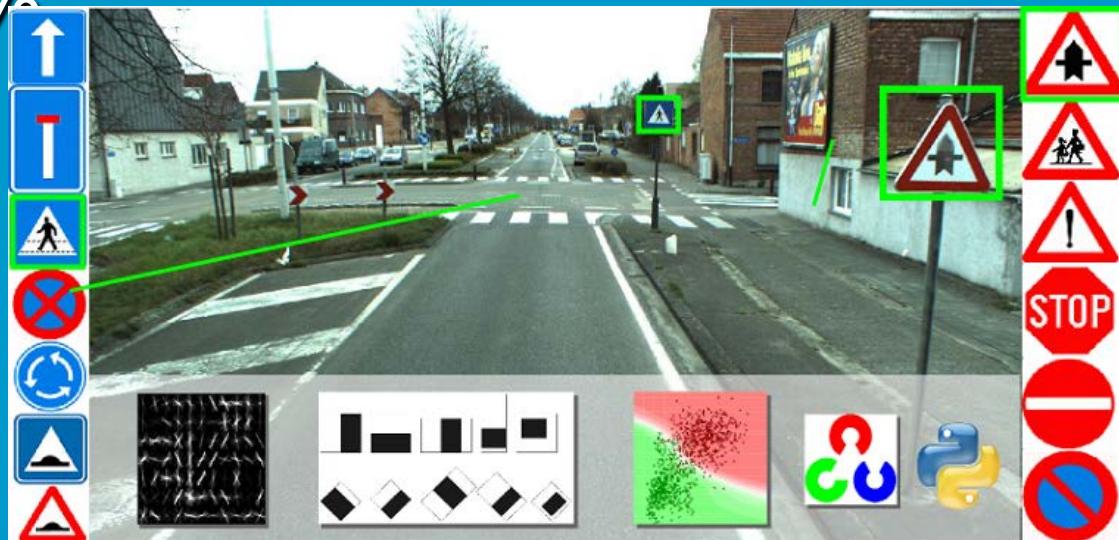
Uma nova era....

Traffic Sign Recognition Competition

Deep learning: 0,56%

Humanos: 1,16%

Outros: 1,69%



CIFAR-10

10 categorias

50,000 Imagens de treinamento, imagens de 32x32x3
10,000 imagens de teste.

airplane



automobile



bird



cat



deer



dog



frog



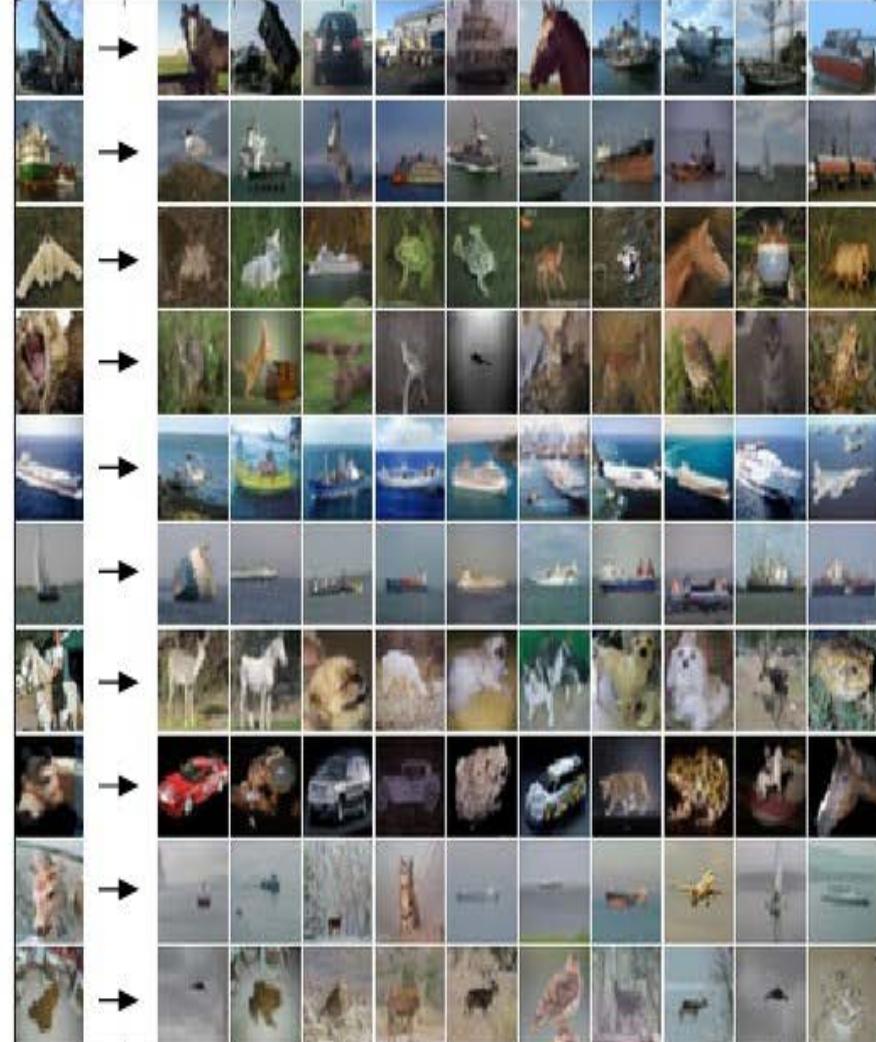
horse



ship



truck



Uma nova era....

ImageNet Challenge 2012

- ~14 milhões de imagens,
- 20 mil classes
- Obtidas na internet
- Desafio: 1.2 milhões de imagens para treinamento,
- 1000 classes



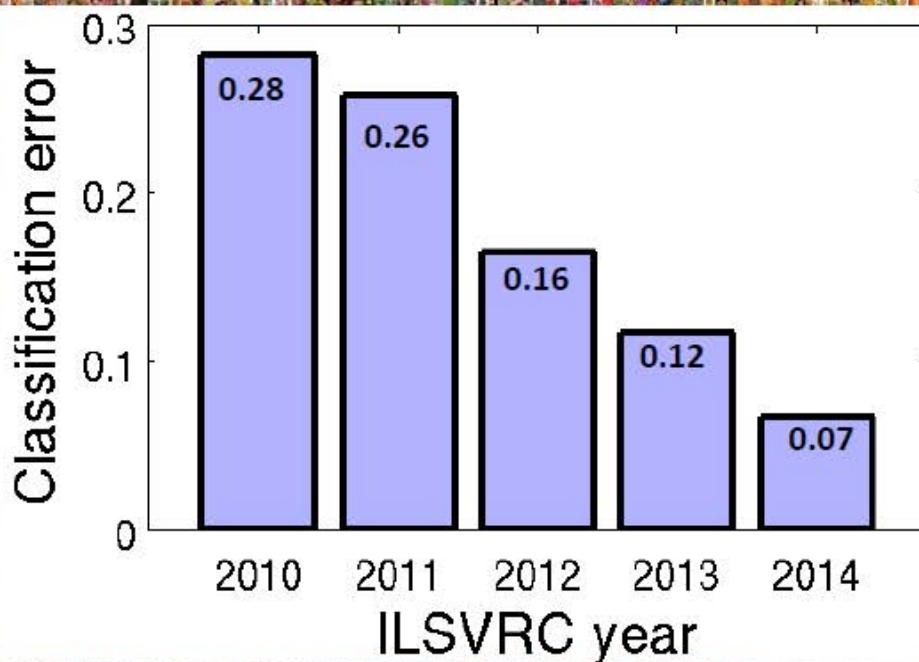
www.image-net.org

22K categories and **14M** images

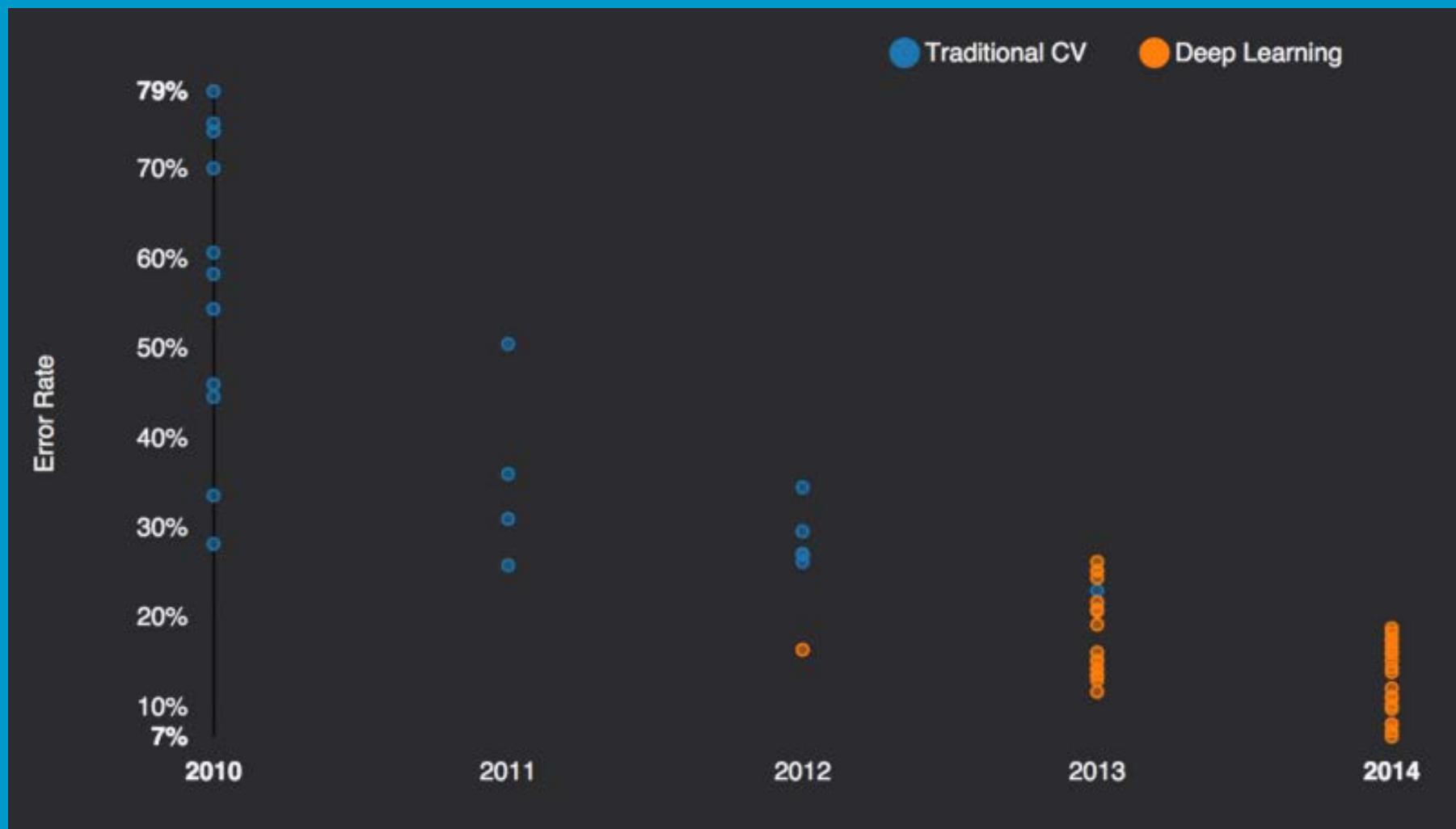
- Animals
 - Bird
 - Fish
 - Mammal
 - Invertebrate
- Plants
 - Tree
 - Flower
 - Food
 - Materials
- Structures
 - Artifact
 - Tools
 - Appliances
 - Structures
- Person
- Scenes
 - Indoor
 - Geological Formations
- Sport Activities

The Image Classification Challenge:

1,000 object classes
1,431,167 images



Uma nova era....



Uma nova era....

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale.

O vencedor de 2014 reconheceu corretamente 93 das 100 imagens de teste, um grande passo comparado a competição de 2010 em que o vencedor acertou 72/100 utilizando técnicas tradicionais de visão computacional.

Uma nova era...

- “Top-5 error” é a porcentagem de vezes que label apareceu entre os 5 com maior probabildiade

Name	Layers	Top-5 Error (%)	References
AlexNet	8	15.3	Krizhevsky et al. (2012)
VGG Net	19	7.3	Simonyan and Zisserman (2014)
ResNet	152	3.6	He et al. (2016)

- Nota: a performance de um humano é de 5.1% top-5

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC



Dense grid descriptor:
HOG, LBP

Coding: local coordinate,
super-vector

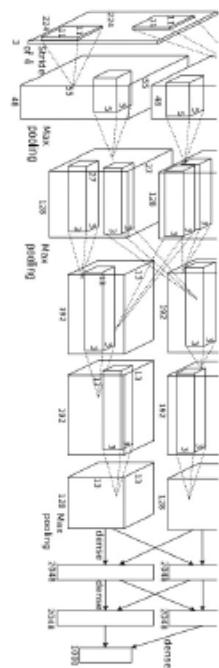
Pooling, SPM

Linear SVM

[Lin CVPR 2011]

Year 2012

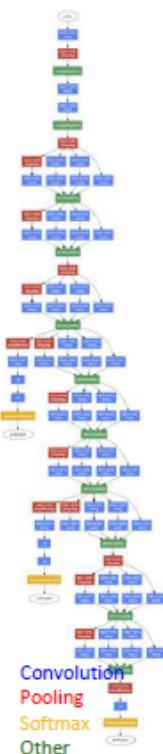
SuperVision



[Krizhevsky NIPS 2012]

Year 2014

GoogLeNet VGG

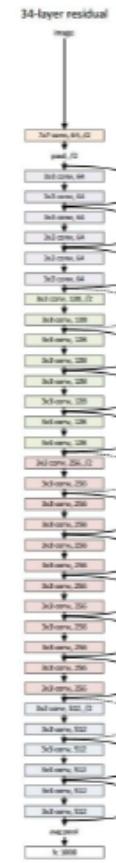


- image
- conv-64
- conv-64
- maxpool
- conv-128
- conv-128
- maxpool
- conv-256
- conv-256
- maxpool
- conv-512
- conv-512
- maxpool
- conv-512
- conv-512
- maxpool
- FC-4096
- FC-4096
- FC-1000
- softmax

[Szegedy arxiv 2014] [Simonyan arxiv 2014]

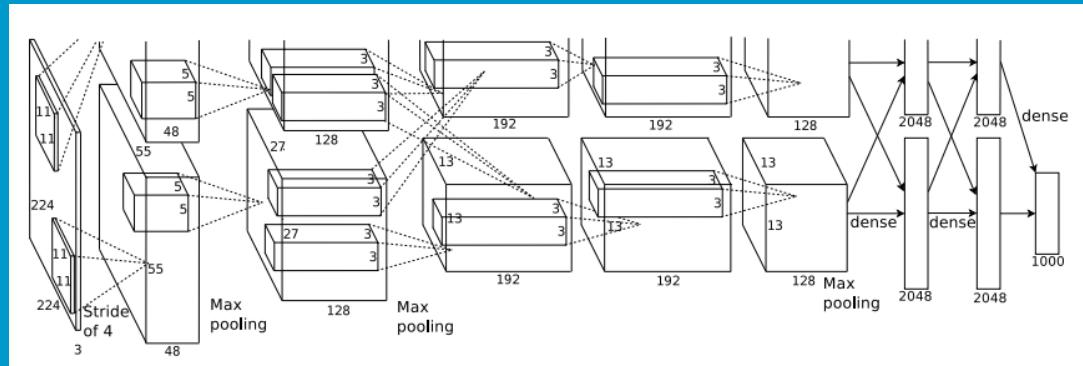
Year 2015

MSRA



Case Study: AlexNet

[Krizhevsky et al. 2012]



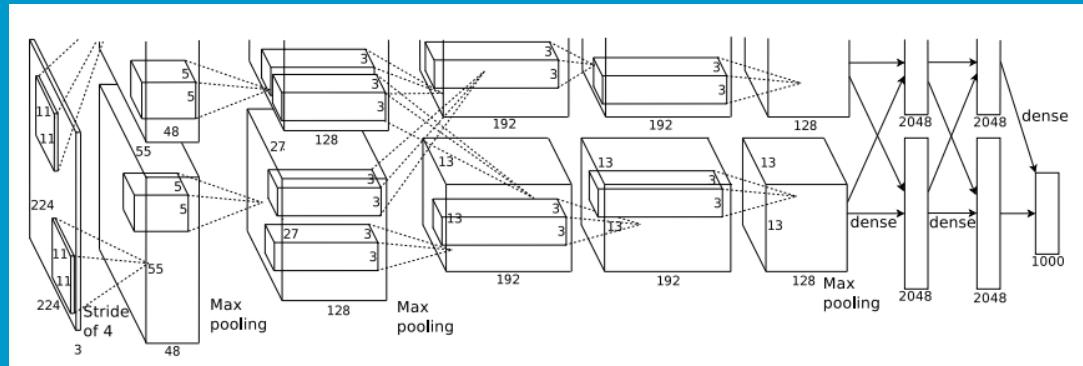
Input: 227x227x3 images

Primeira camada (CONV1): 96 11x11 filtros com stride 4

Saída: $(227-11)/4+1 = 55$

Case Study: AlexNet

[Krizhevsky et al. 2012]

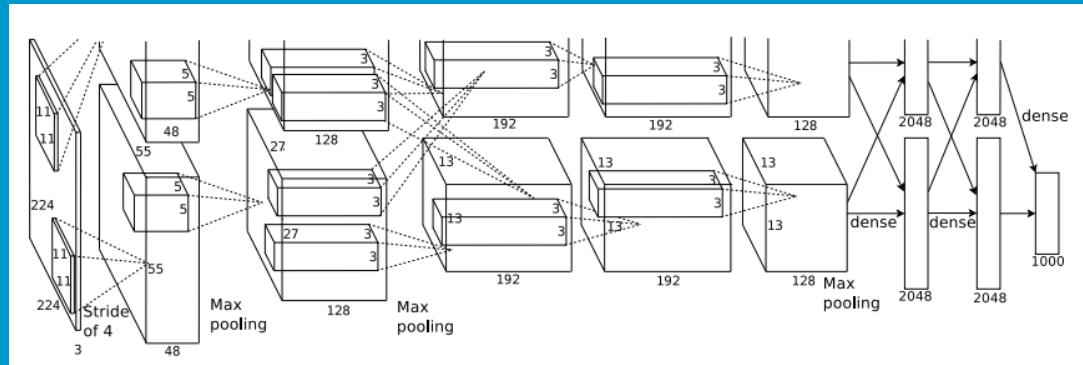


Input: 227x227x3 images

**Primeira camada (CONV1): 96 11x11 filtros com stride 4
Saída [55x55x96]**

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

Primeira camada (CONV1): 96 11x11 filtros com stride 4

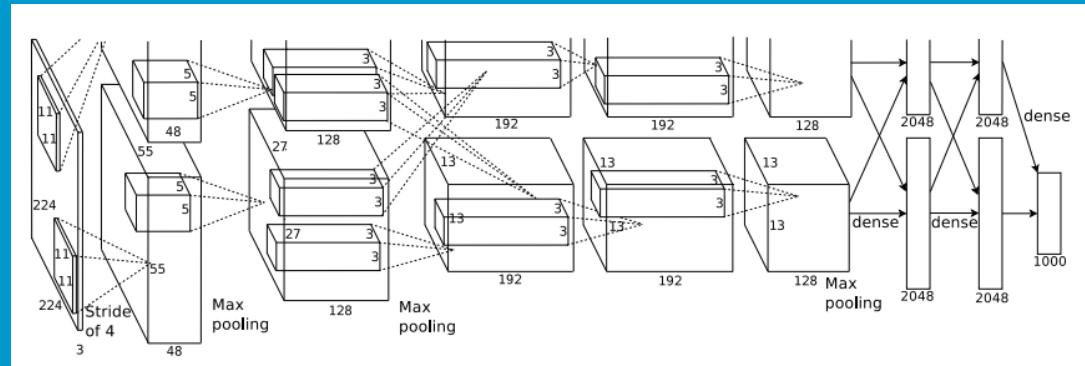
=>

Saída **[55x55x96]**

Parametros: $(11 \times 11 \times 3) \times 96 = 35\text{K}$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

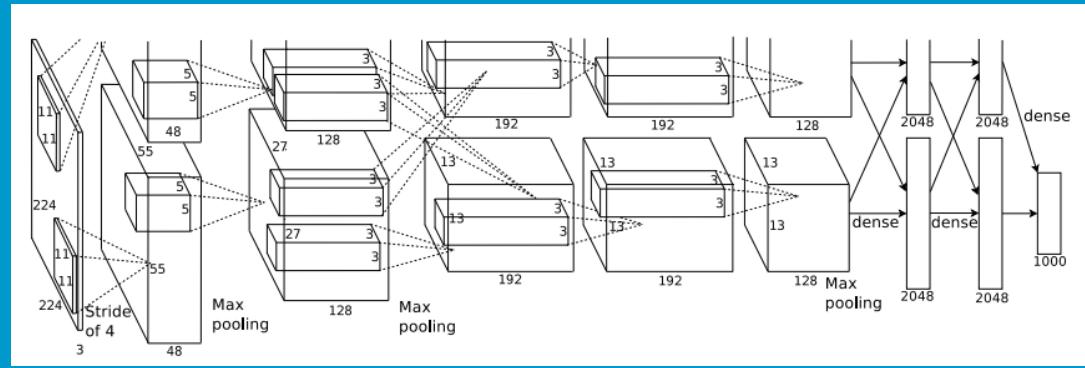
After CONV1: 55x55x96

Segunda camada (POOL1): 3x3 filtros com stride 2

Q: Qual o tamanho da saída? Hint: $(55-3)/2+1 = 27$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

Depois da CONV1: 55x55x96

Segunda camada (POOL1): 3x3 filtros com stride 2

Saída: 27x27x96

Case Study: AlexNet

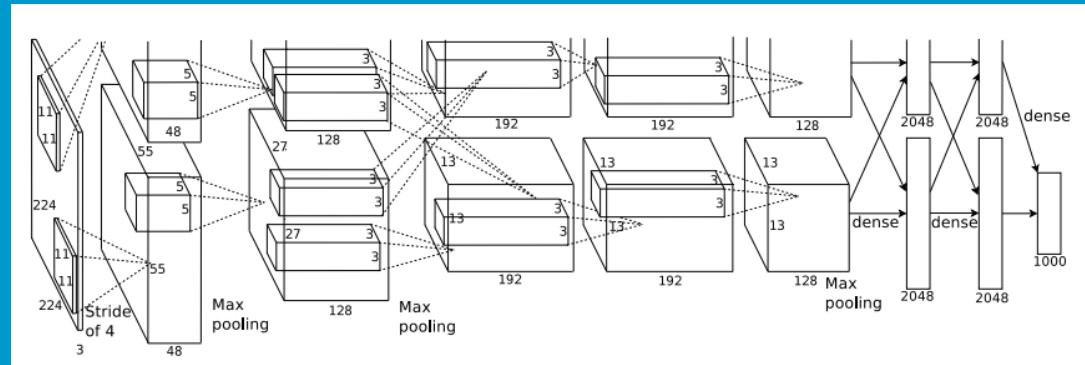
[Krizhevsky et al. 2012]

Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

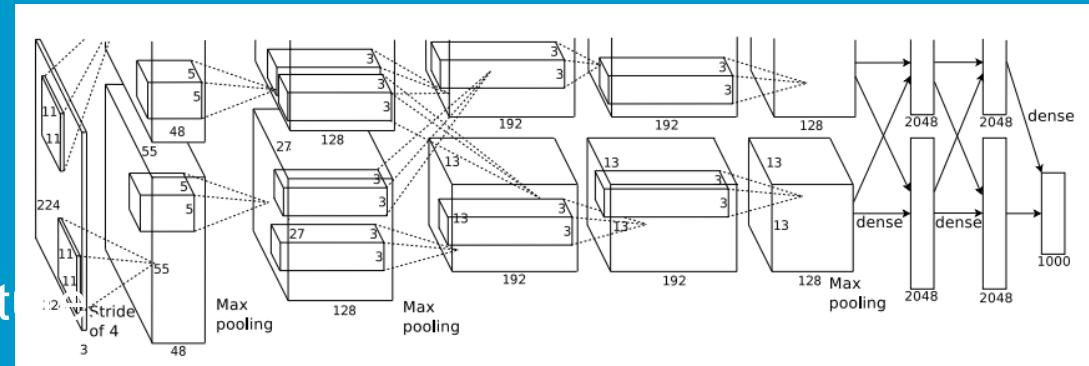
...



Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architect
[227x227x3] INPUT



[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

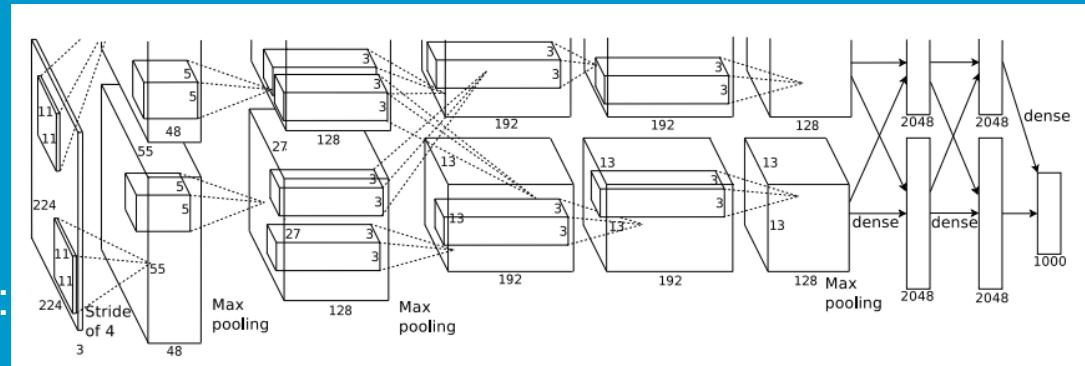
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

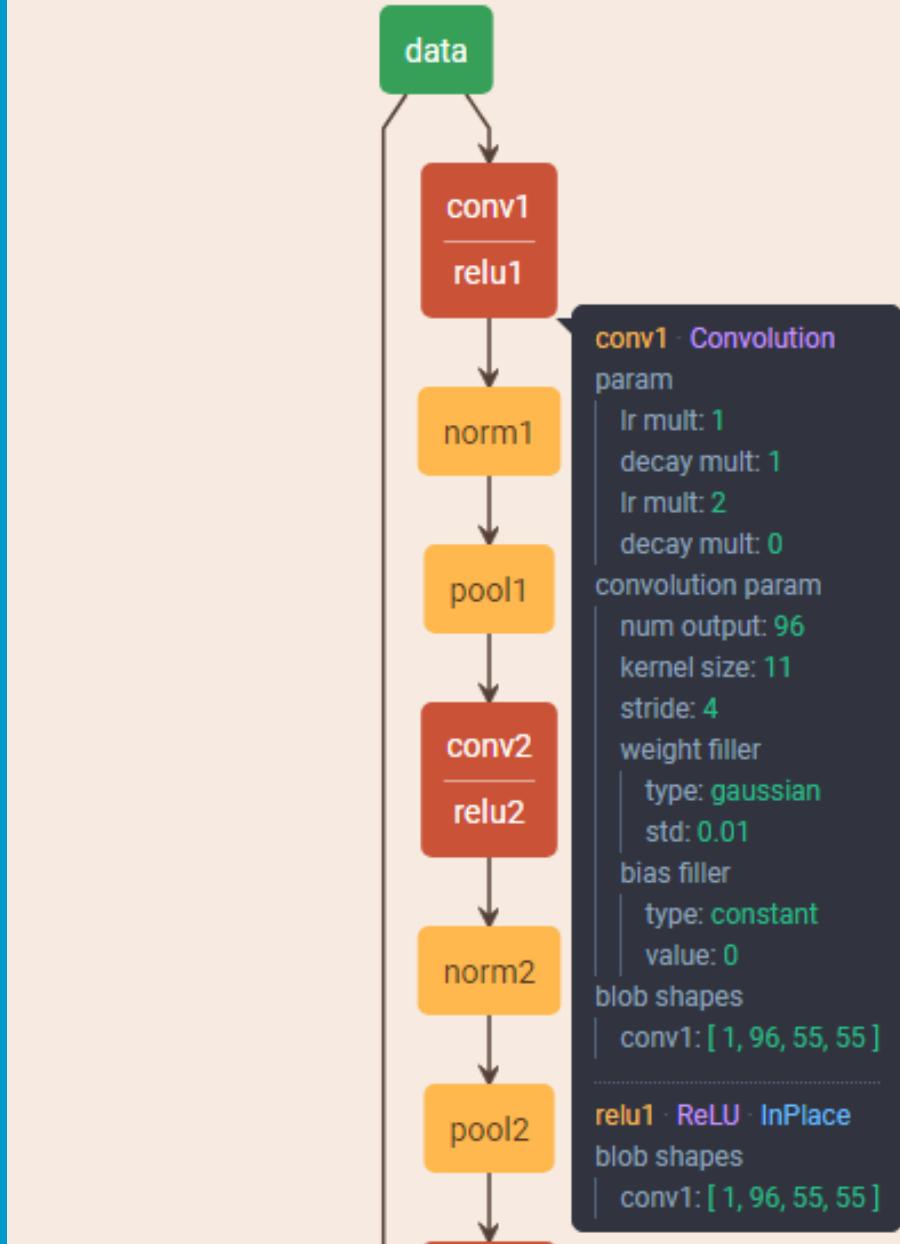
[1000] FC8: 1000 neurons (class scores)



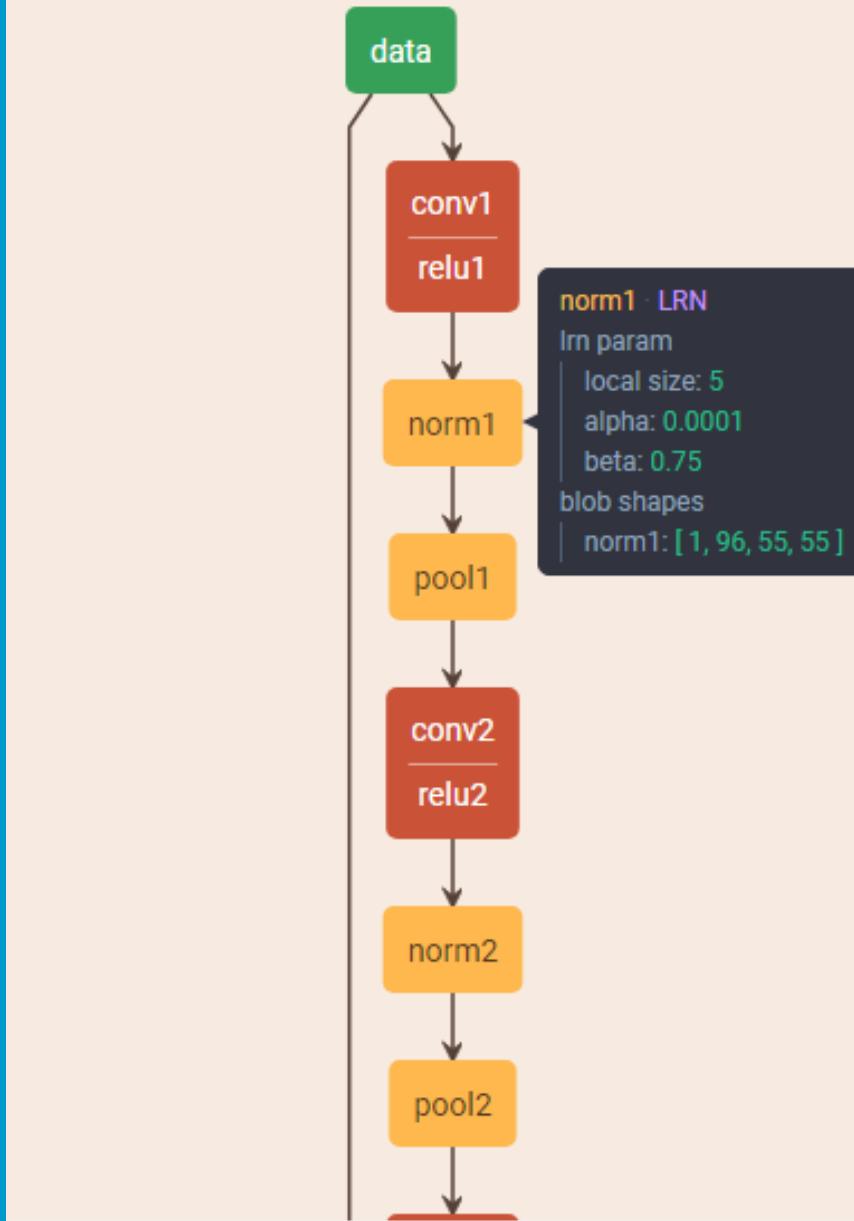
Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% → 15.4%

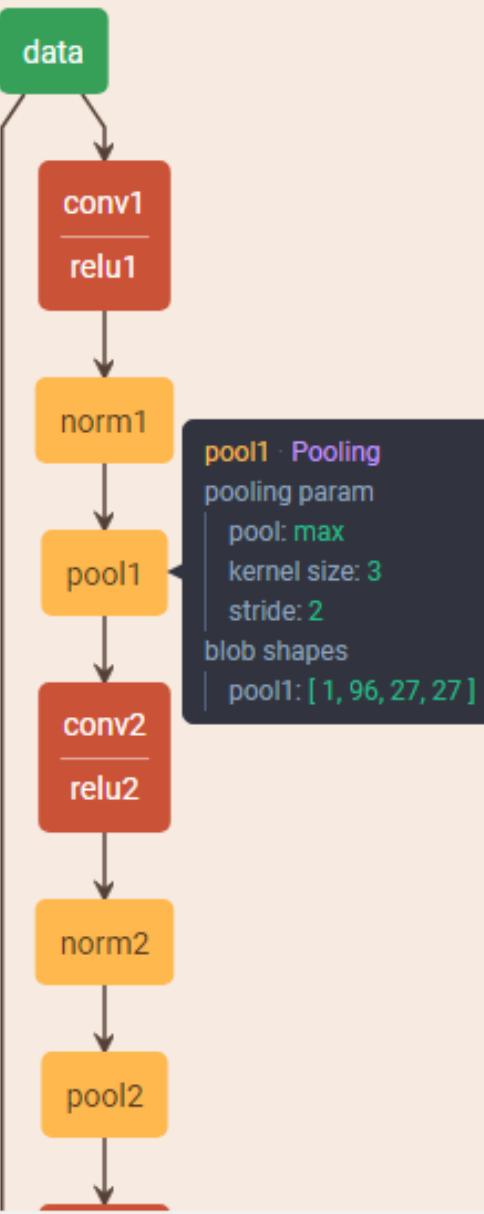
AlexNet



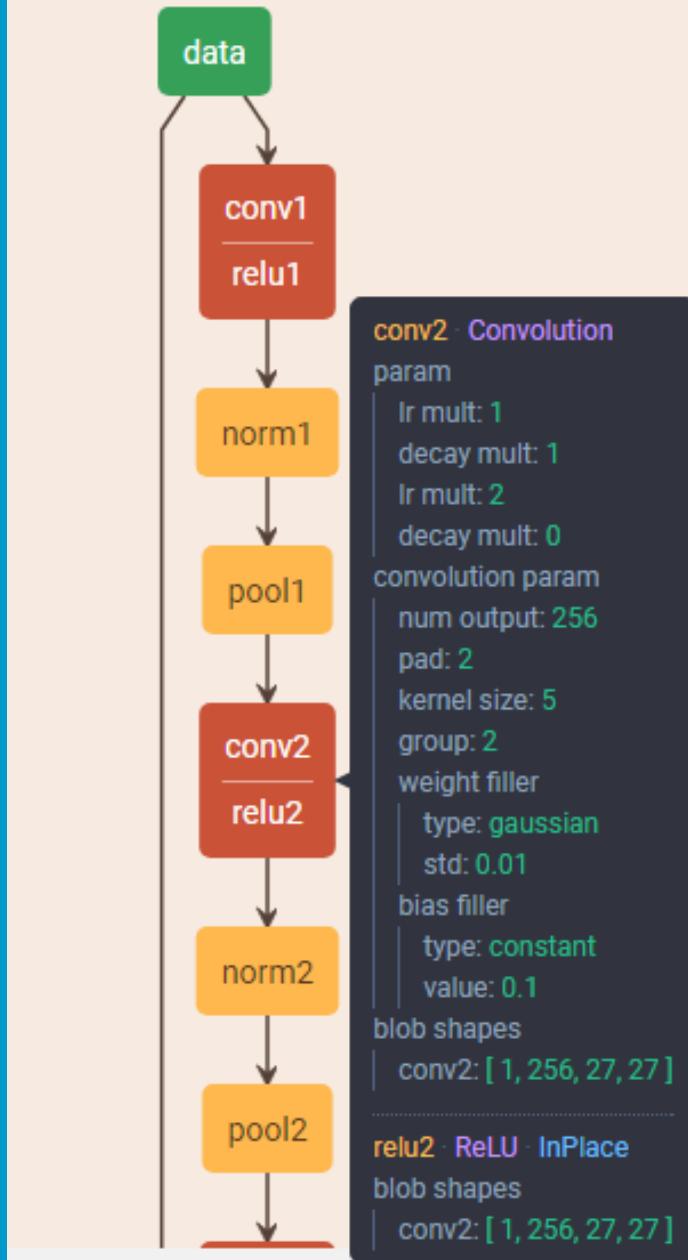
AlexNet



AlexNet



AlexNet

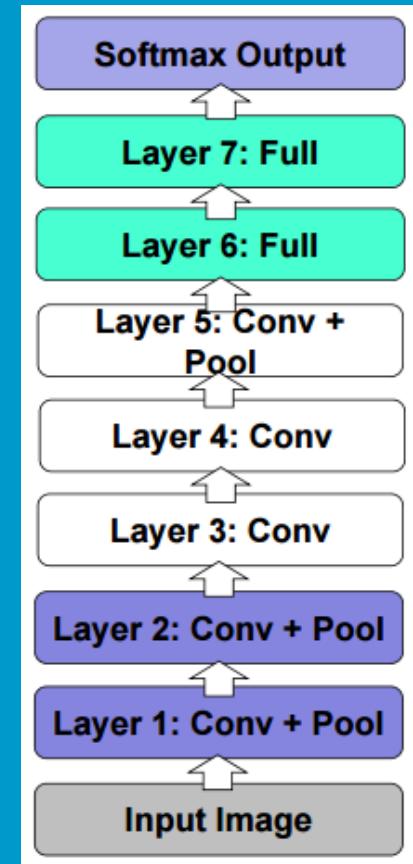


Aspectos de projeto da rede

AlexNet 2012

8 camadas

Erro de 18.1% top-5



Aspectos de projeto da rede

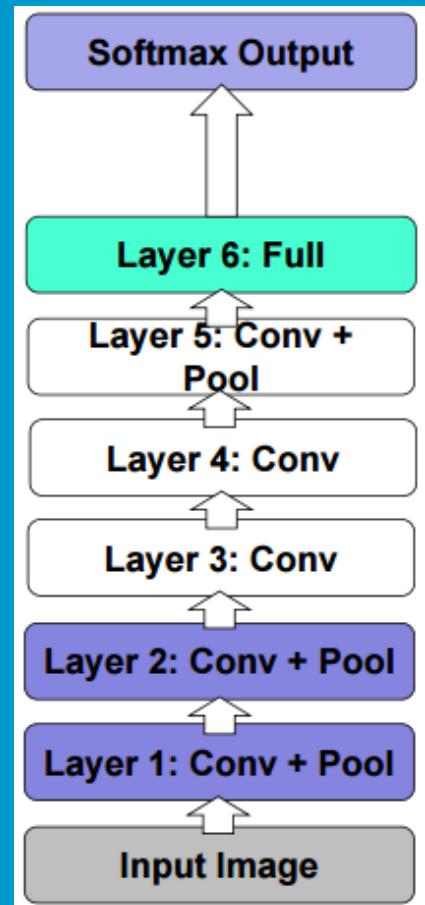
AlexNet 2012

7 camadas

Sem uma das Fully Layer

Menos 17 milhões de parâmetros

Erro de 19.2%



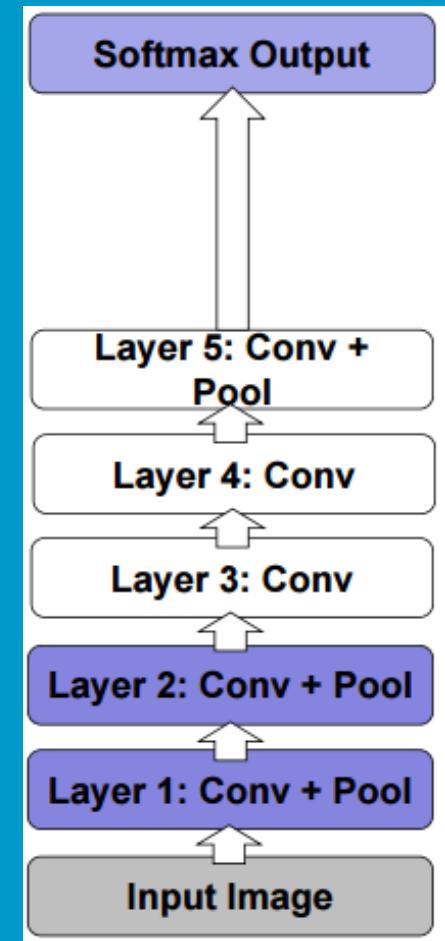
Aspectos de projeto da rede

AlexNet 2012

6 camadas

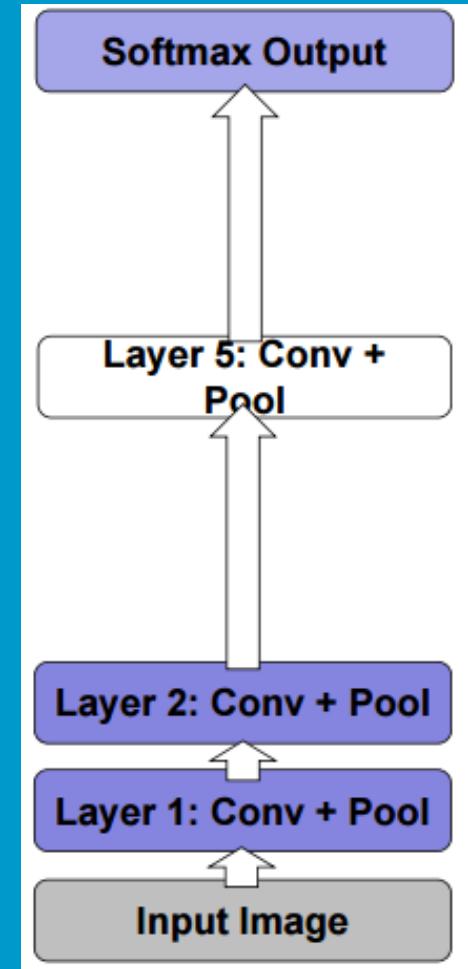
Menos ~50 milhões de parâmetros

Erro ~24%

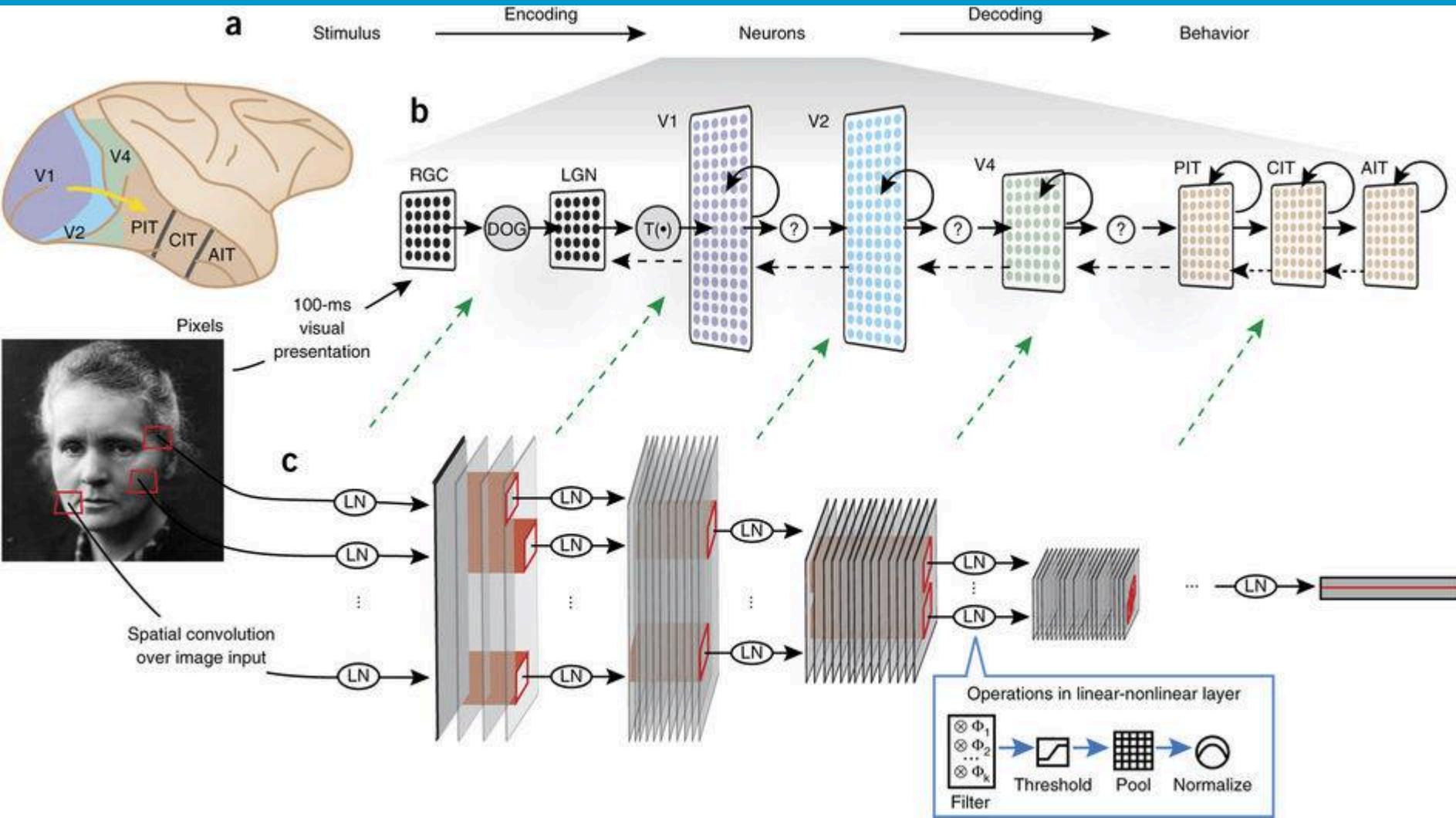


Aspectos de projeto da rede

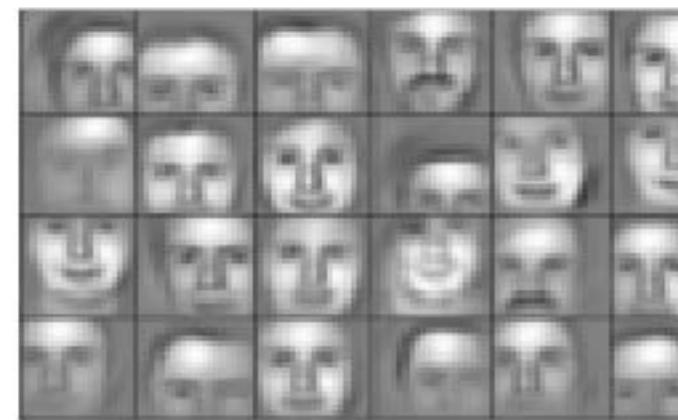
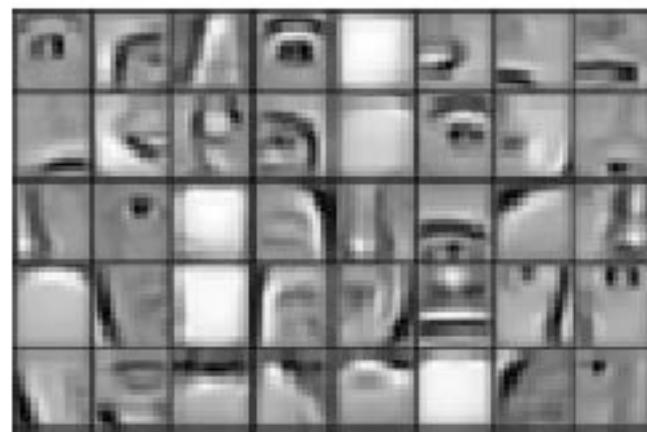
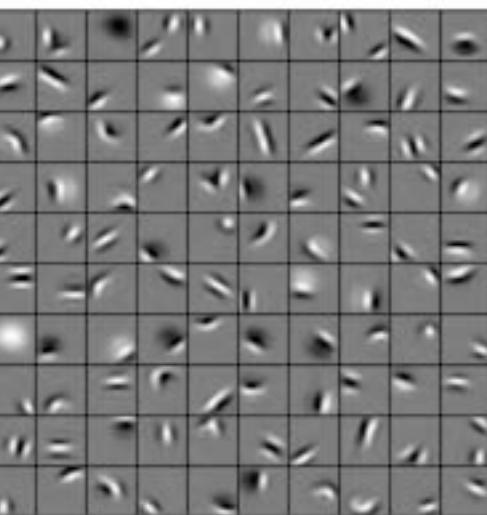
AlexNet 2012
Erro ~33.8



Resumindo...



Resumindo...



Exemplo de uma rede executando no browser

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

Resumindo...

Uma nova era....

Auto-encoders

**Aprendizado não-supervisionado +
supervisionado**

Altos níveis de abstração

Alta carga computacional

Resumindo...



Resumindo...

IA tradicional	Deep learning
Engenharia de variáveis	Modelo hierárquico com camadas de abstrações
Rótulos geralmente definidos por humanos	Dados não rotulados
Computação sequencial	Múltiplas decisões simultâneas

Próxima aula: parametrização de redes neurais profundas



Tarefa para casa: Explique pq o professor montou a imagem abaixo



Por hoje é só pessoal...

