

Convolutional Neural Networks

Uma breve história

Mark I Perceptron

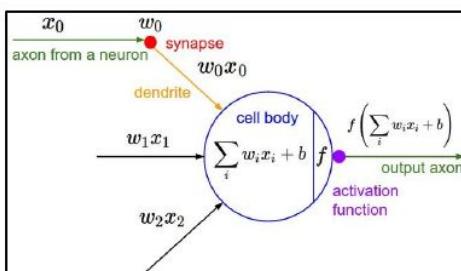
- Essa máquina foi a primeira implementação do algoritmo **perceptron**
- Conectada com uma câmera que usava fotocélulas de sulfeto de cádmio para produzir uma imagem de 400 pixels
- Reconhecia letras do alfabeto

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

regra de atualização dos pesos:

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

1 camada apenas:



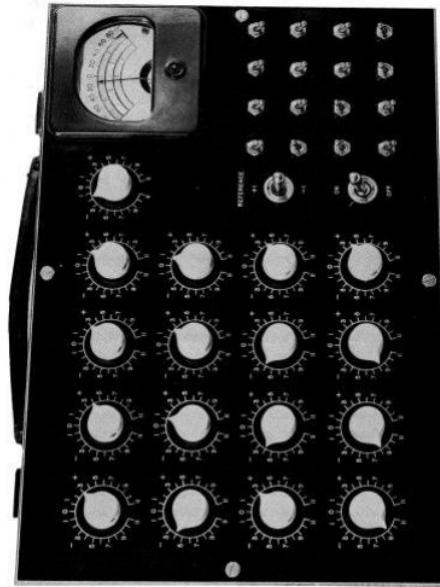
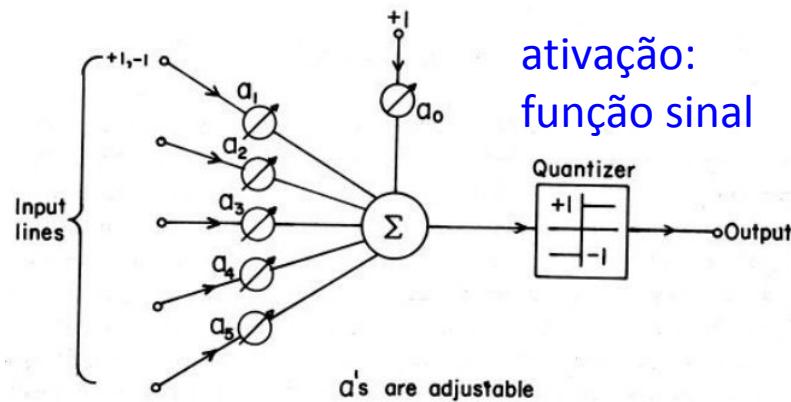
Frank Rosenblatt, ~1957: Perceptron



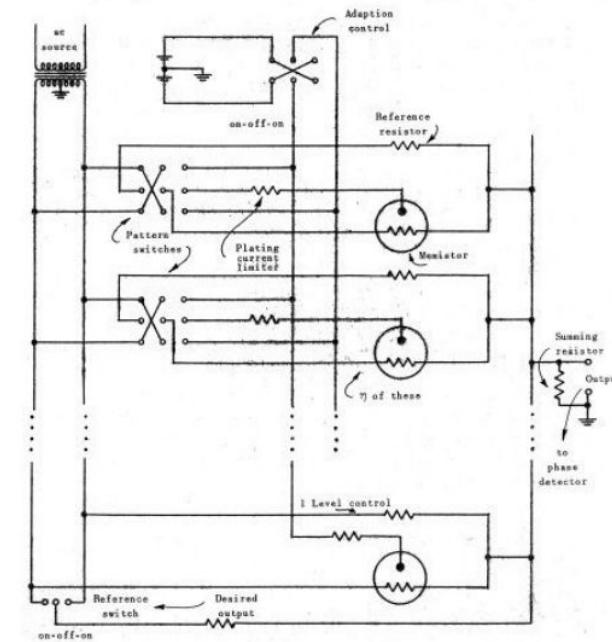
This image by Rocky Acosta is licensed under CC-BY 3.0

Adaline/Madaline

- Primeira rede multicamadas

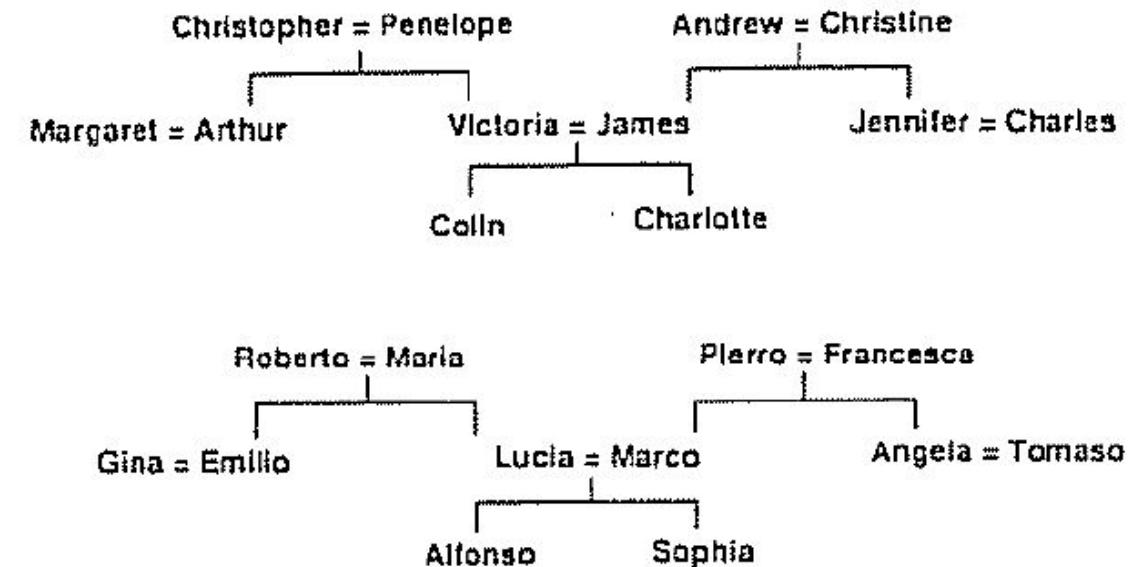
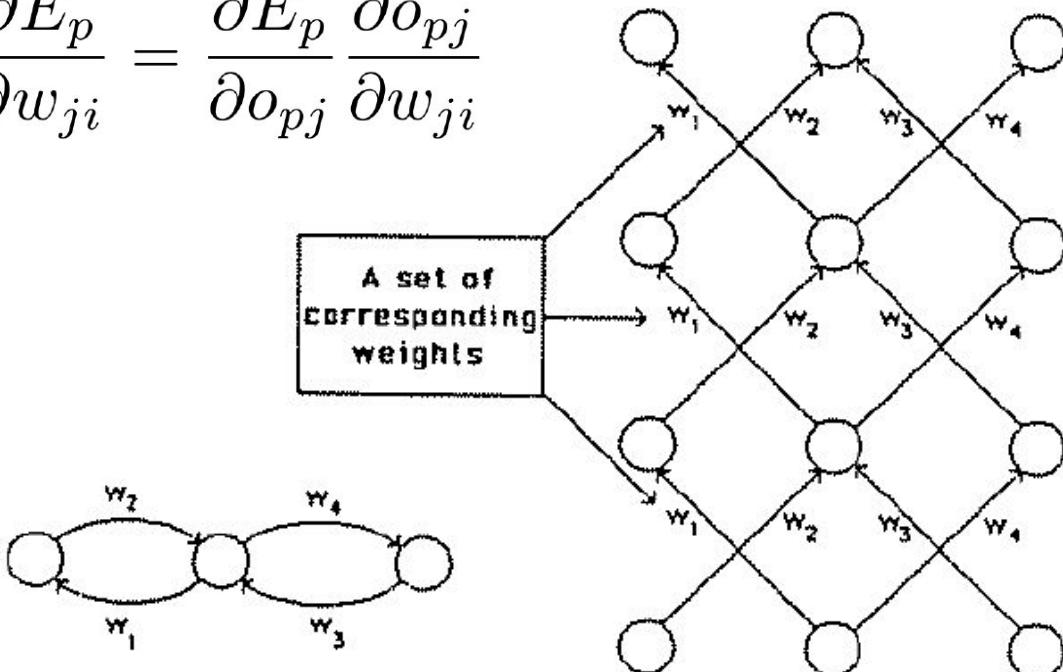


Madaline: três camadas de Adaline



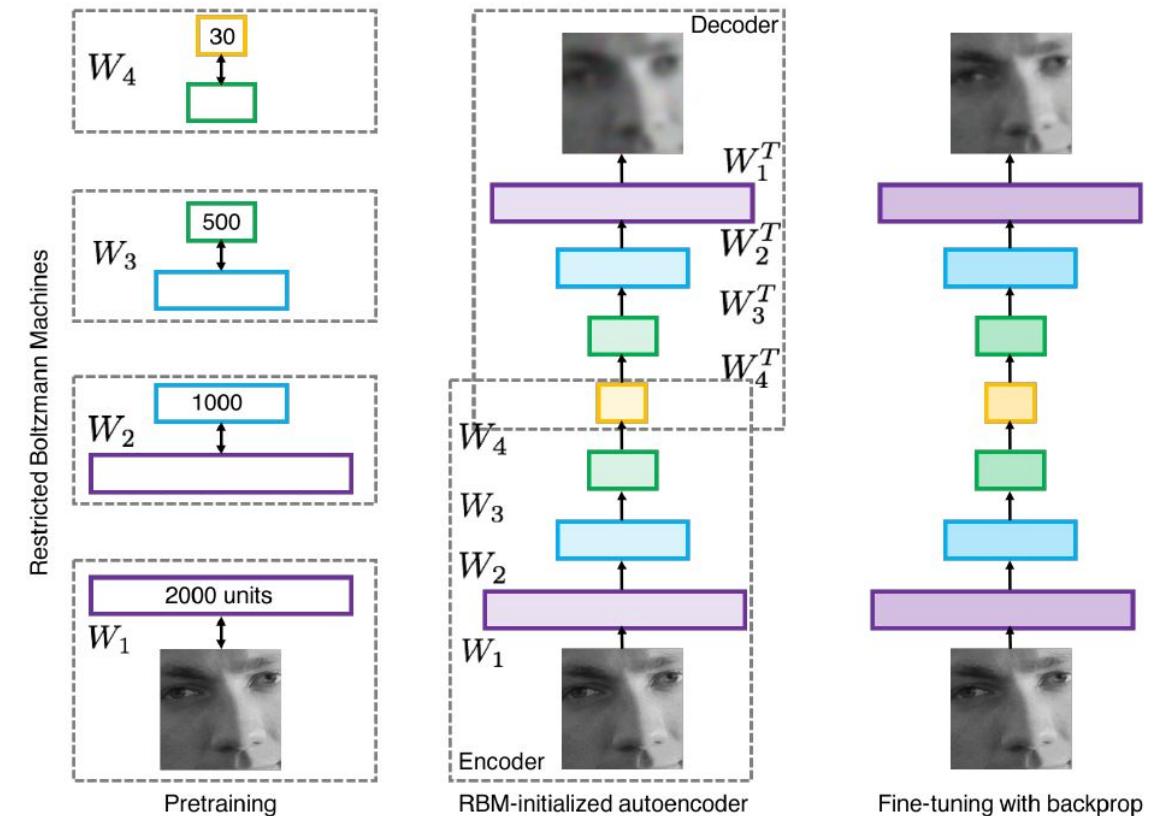
Finalmente, back-propagation!

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$



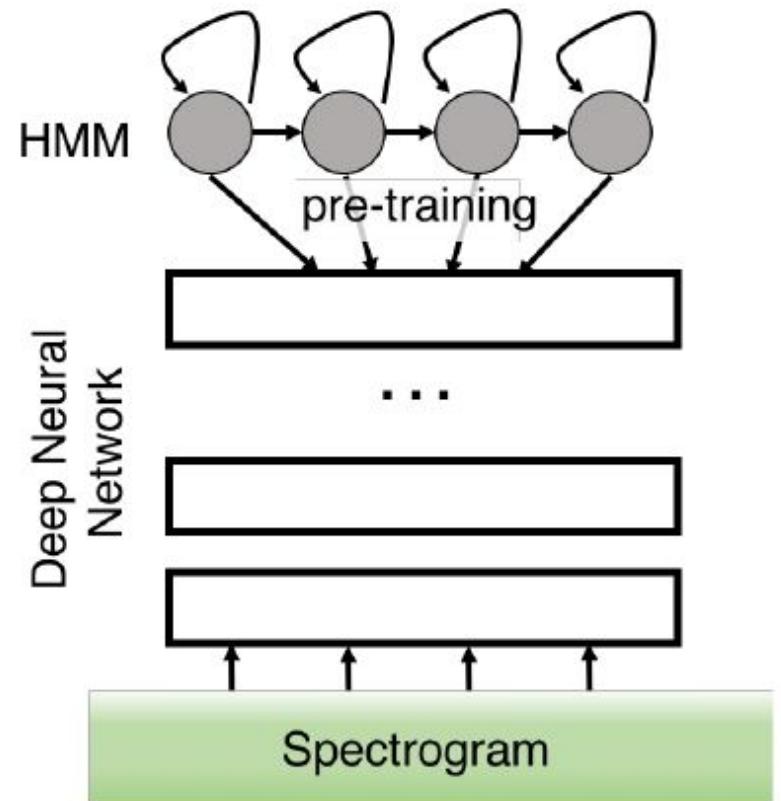
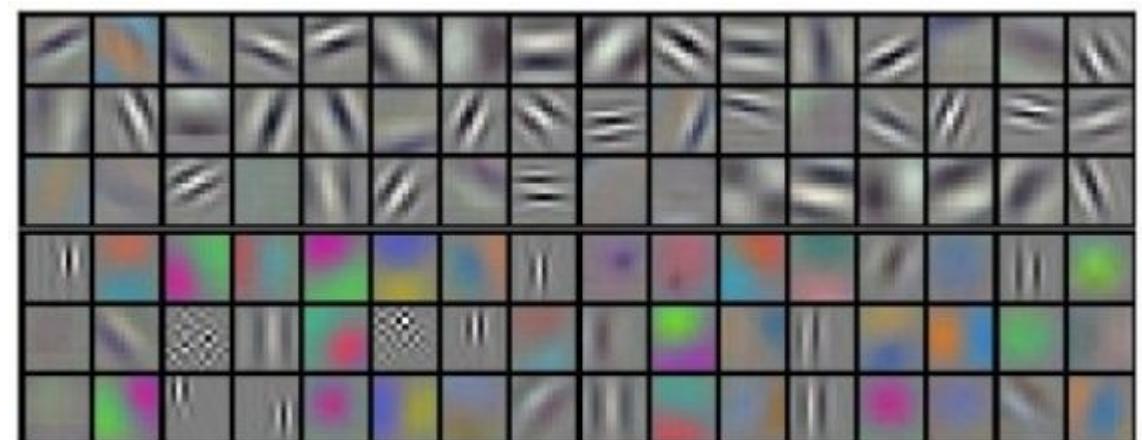
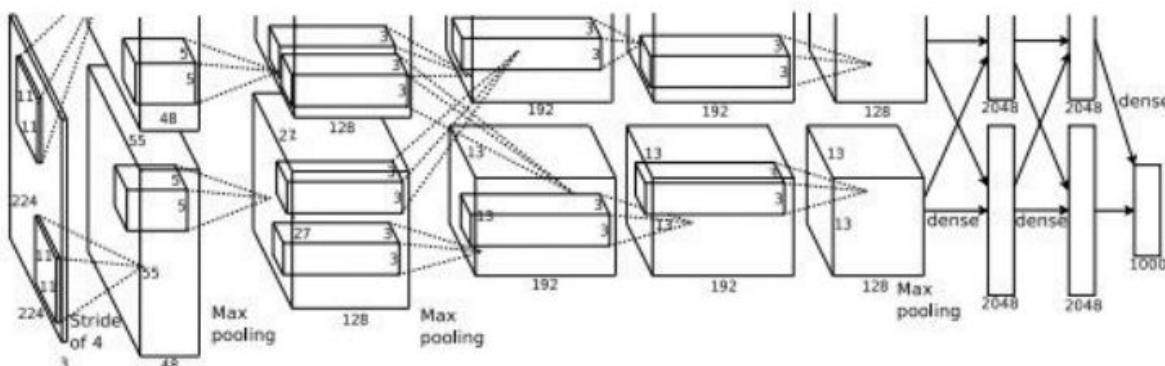
Backprop com inicialização cuidadosa

- Primeiro treina as camadas escondidas com RBMs
- Depois usa Backprop para treinar a rede completa



Primeiros resultados fortes

- Reconhecimento de fala
 - Acoustic Modeling using Deep Belief Networks.
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
 - Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition.
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012
- Classificação de imagens
 - Imagenet classification with deep convolutional neural networks. Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

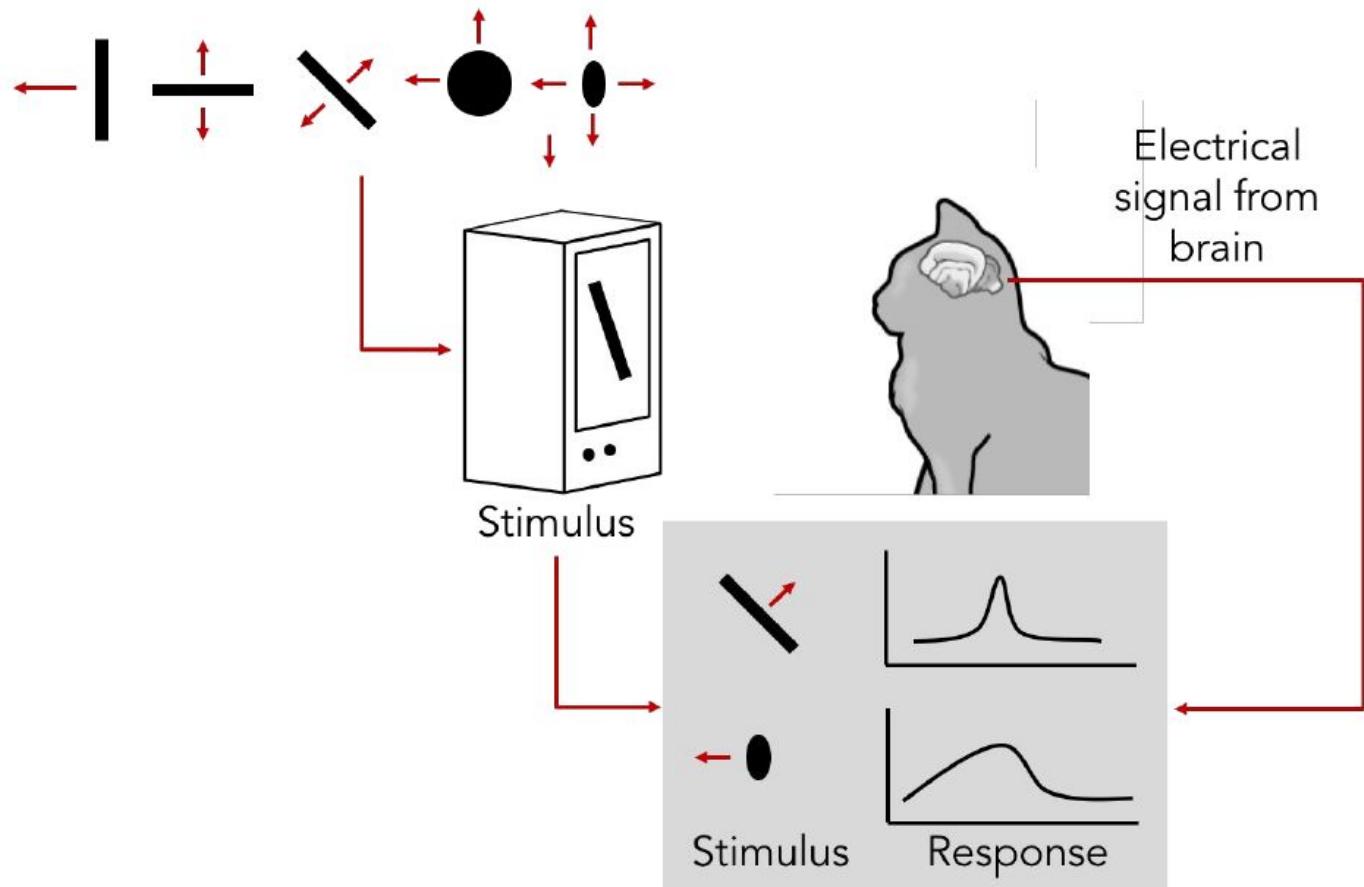


Hubel & Wiesel

1959: Receptive fields of single neurons in the cat's striate cortex

1962: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex

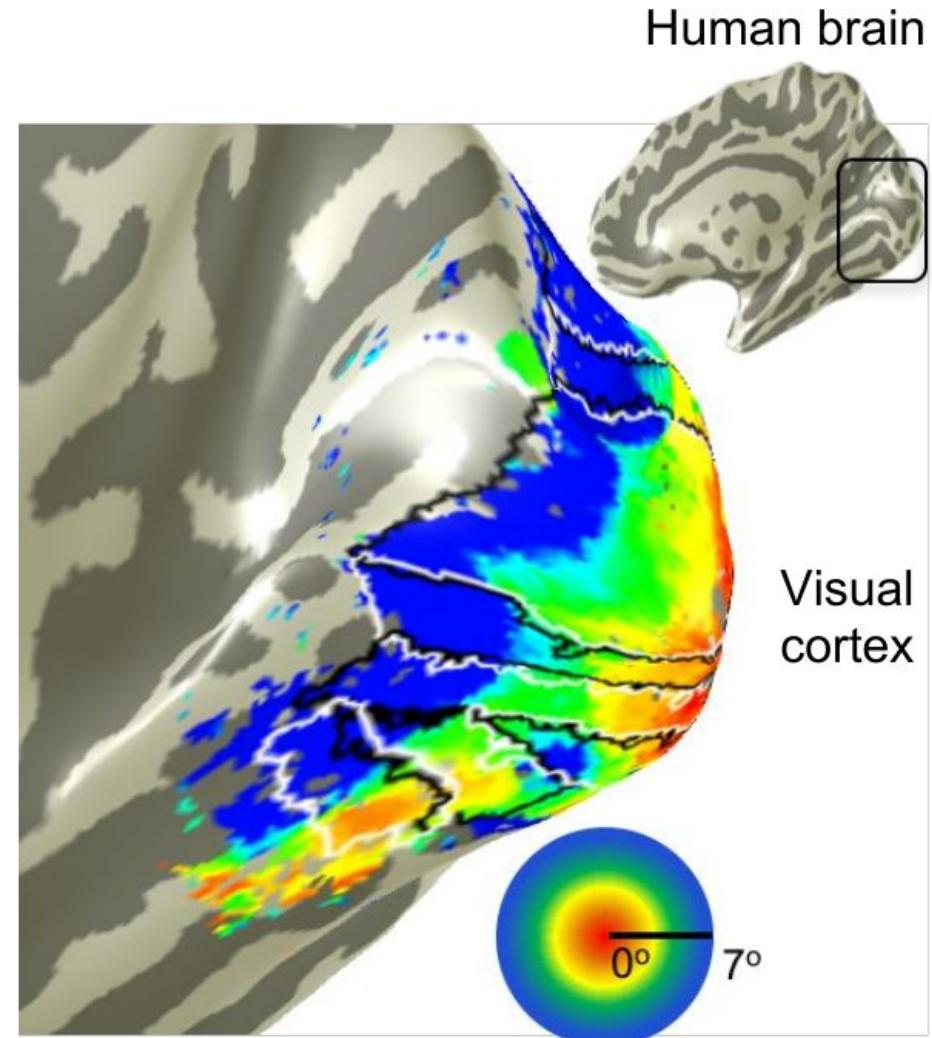
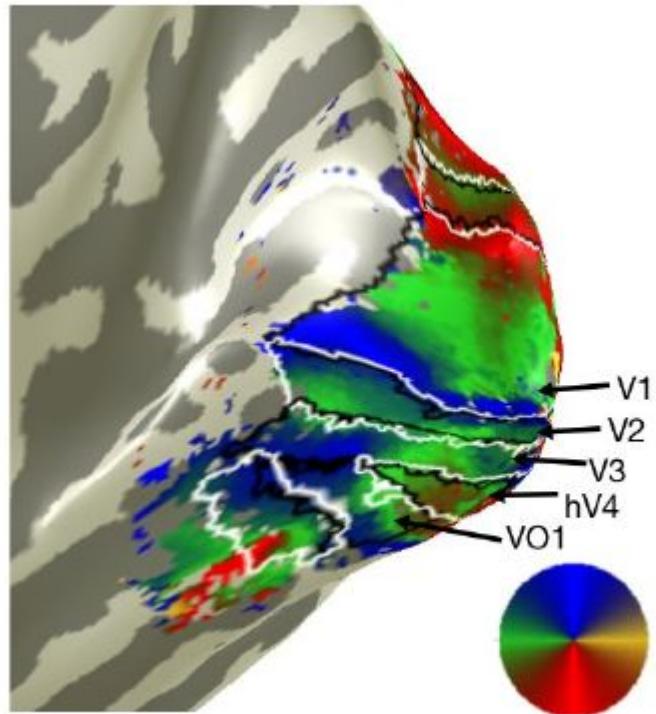
1968...



Hubel & Wiesel

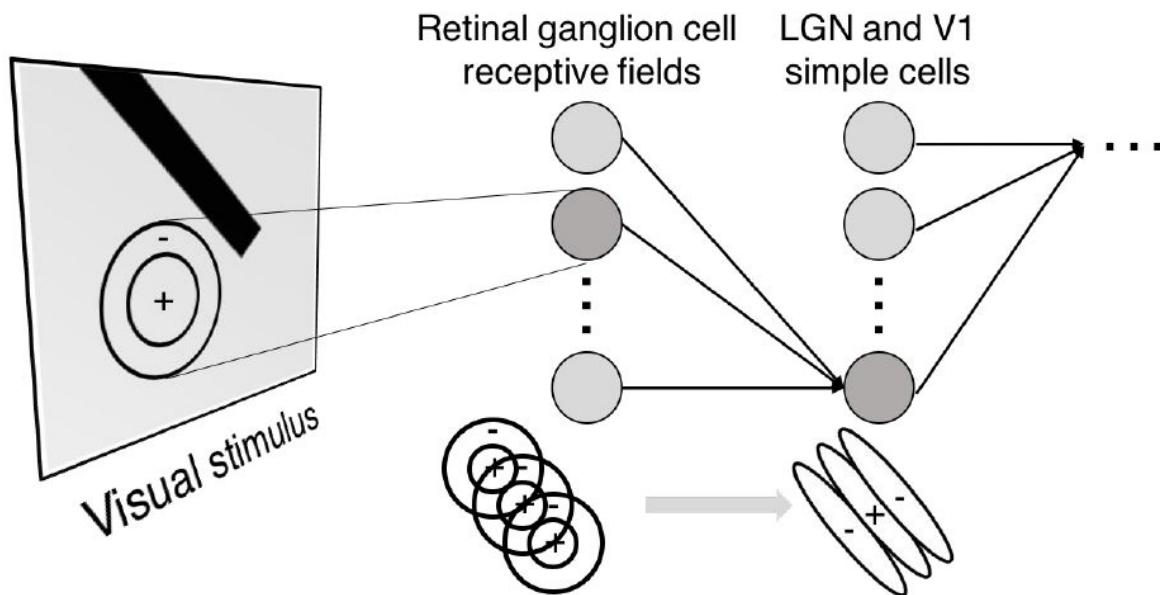
Mapeamento topográfico no córtex:

Células próximas no córtex representam regiões próximas no campo visual



Hubel & Wiesel

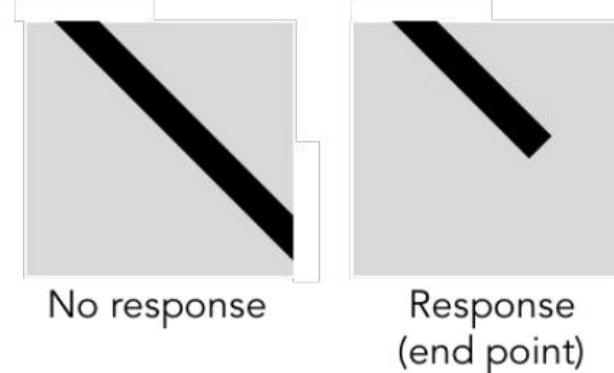
Organização hierárquica



Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point



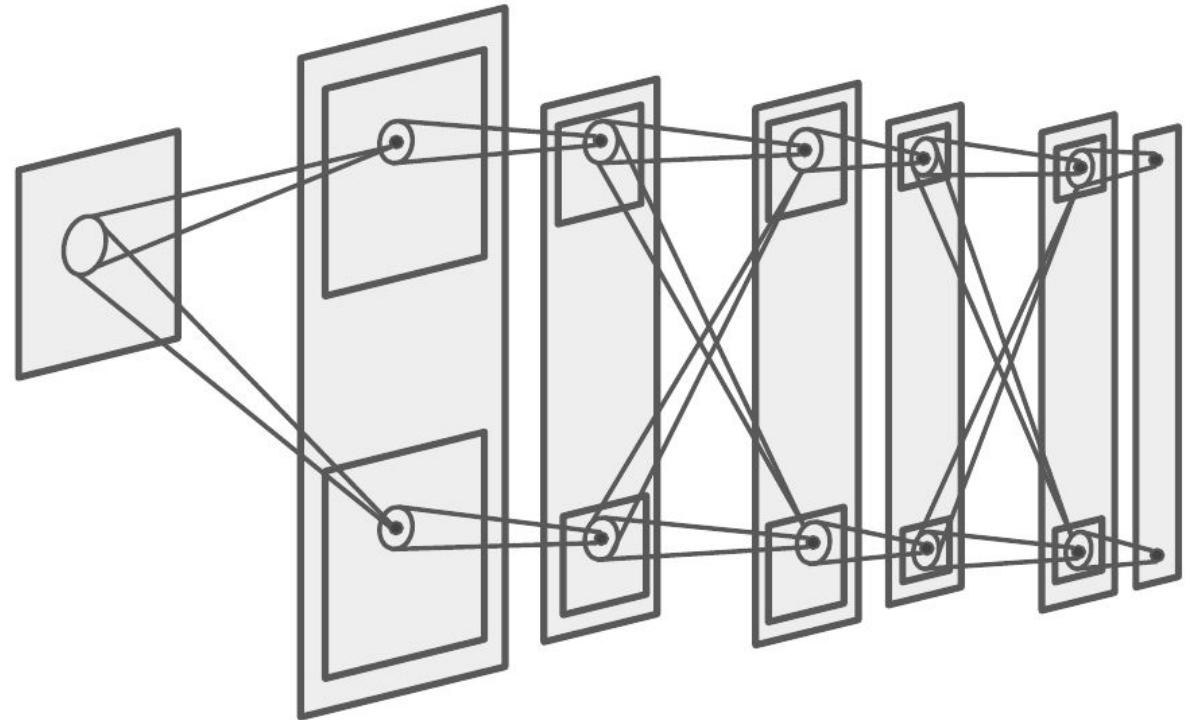
Neocognitron

Reconhecimento de caracteres manuscritos

Arquitetura “sanduíche” (**s**c**s**c**s**c ...)

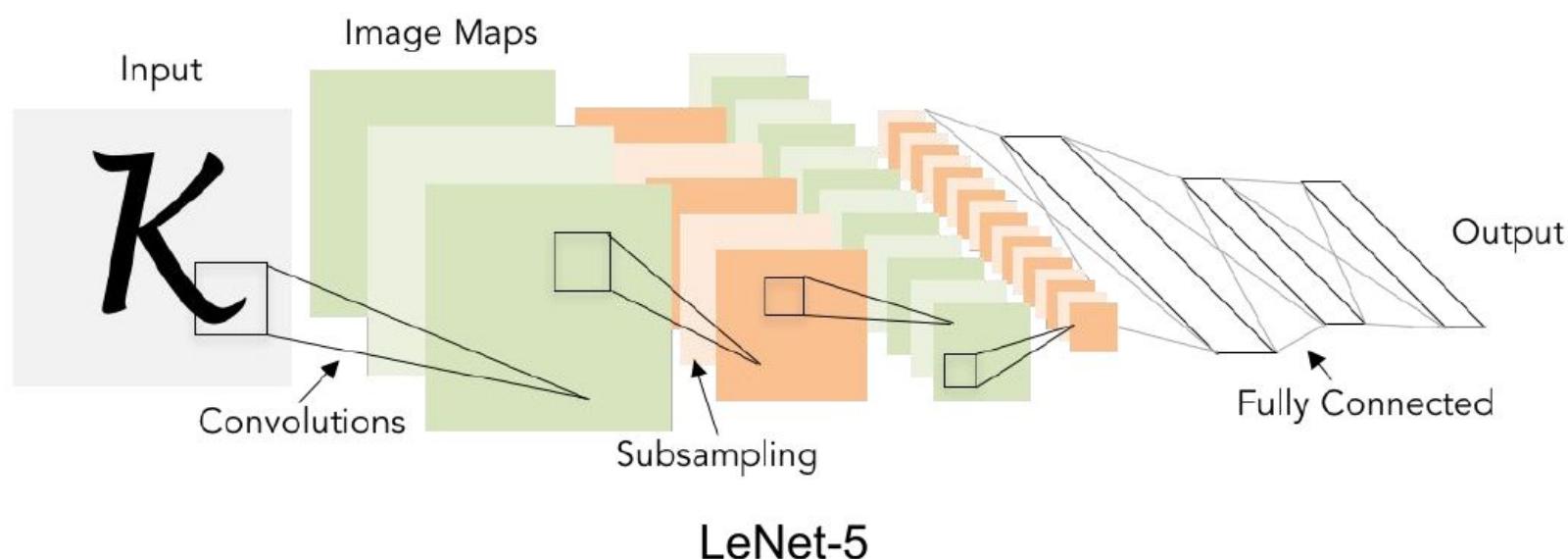
Células simples: parâmetros modificáveis

Células complexas: executar *pooling*



LeNet-5

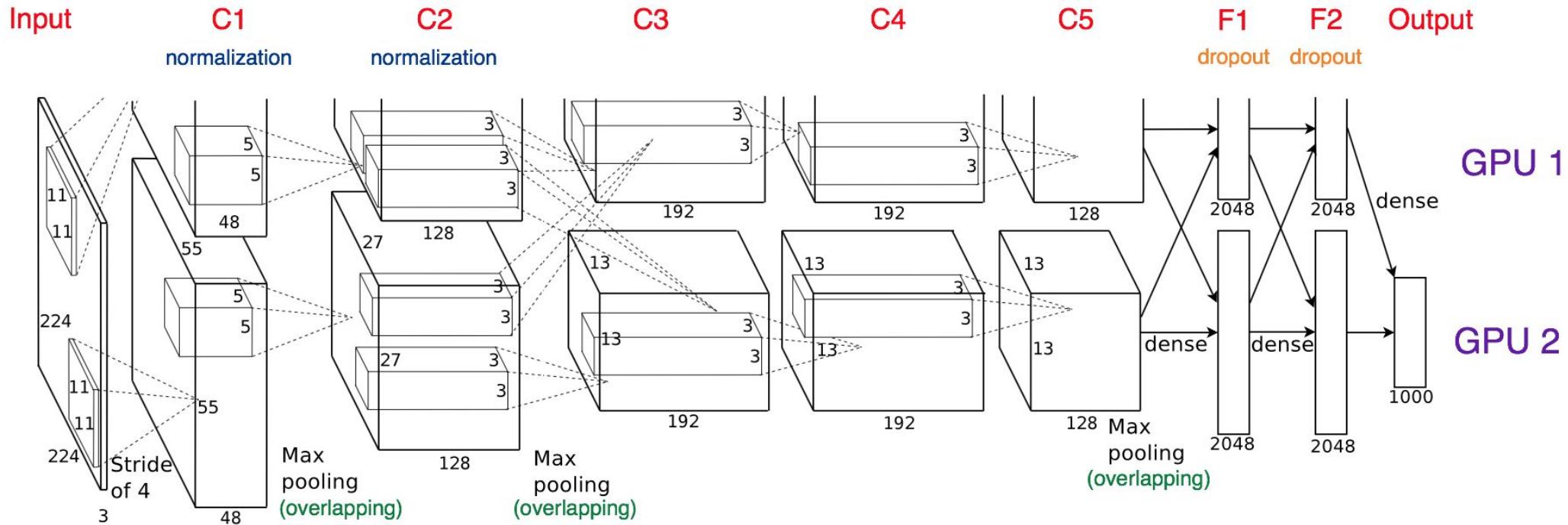
- Primeira CNN com backprop
- Reconhecimento de *Zip codes*
 - Foi usada por correios
- Não escalava muito bem ainda



Gradient-based learning applied to document recognition. LeCun, Bottou, Bengio, Haffner, 1998

AlexNet

- Não muito diferente da LetNet-5, mas
 - Usou **muitos** dados rotulados
 - GPUs



Convolutional Neural Networks

Visão Computacional

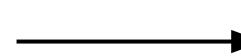
Como identificamos objetos?

Problemas de Visão Computacional

Classificação de Imagens



64x64

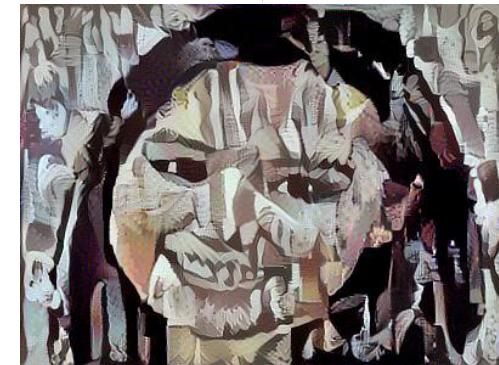
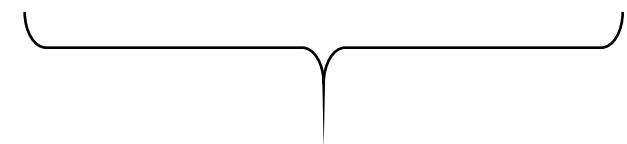


Gato? (0/1)

Detecção de objetos



Transferência de estilo



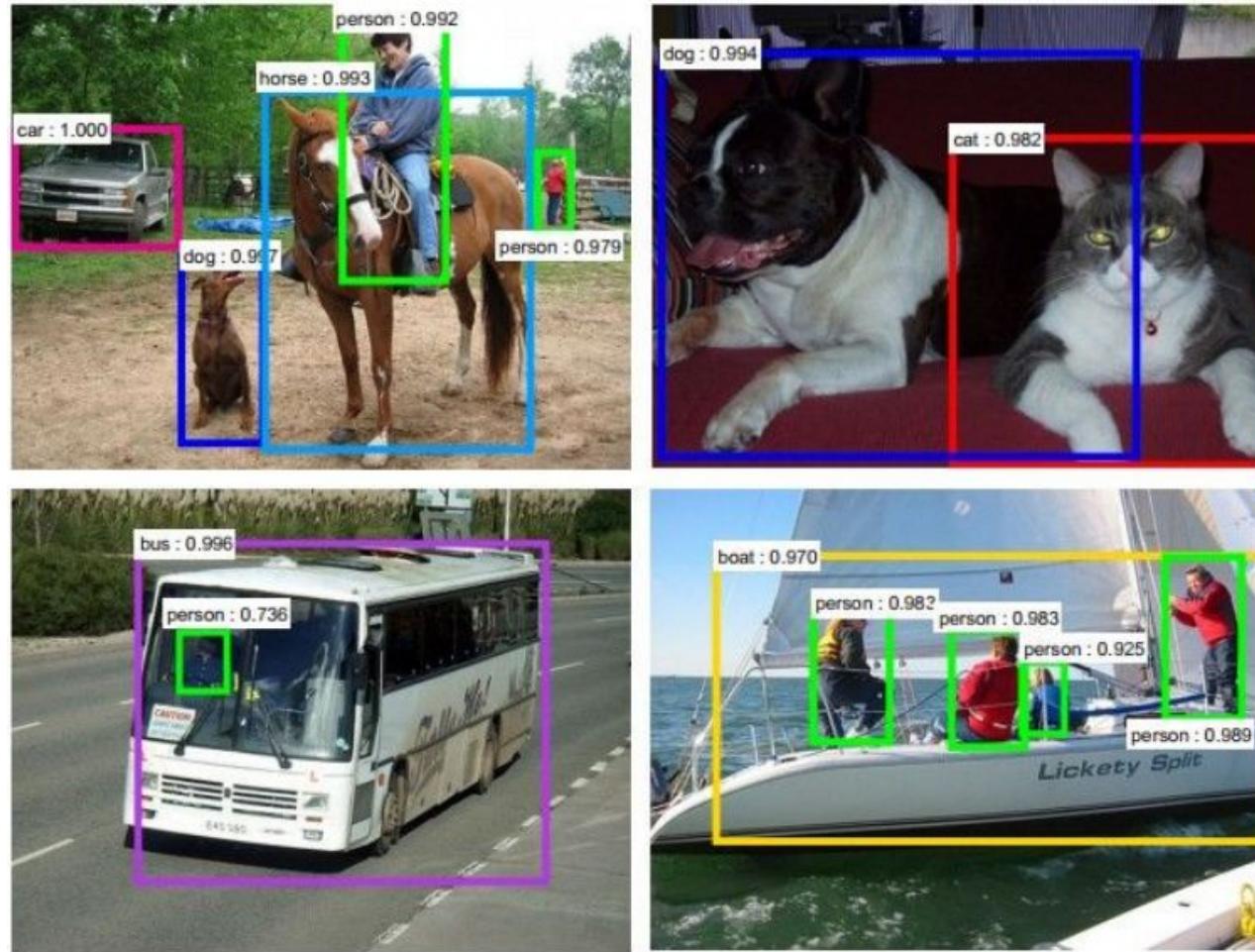
Classificação de imagens



Recuperação de imagens

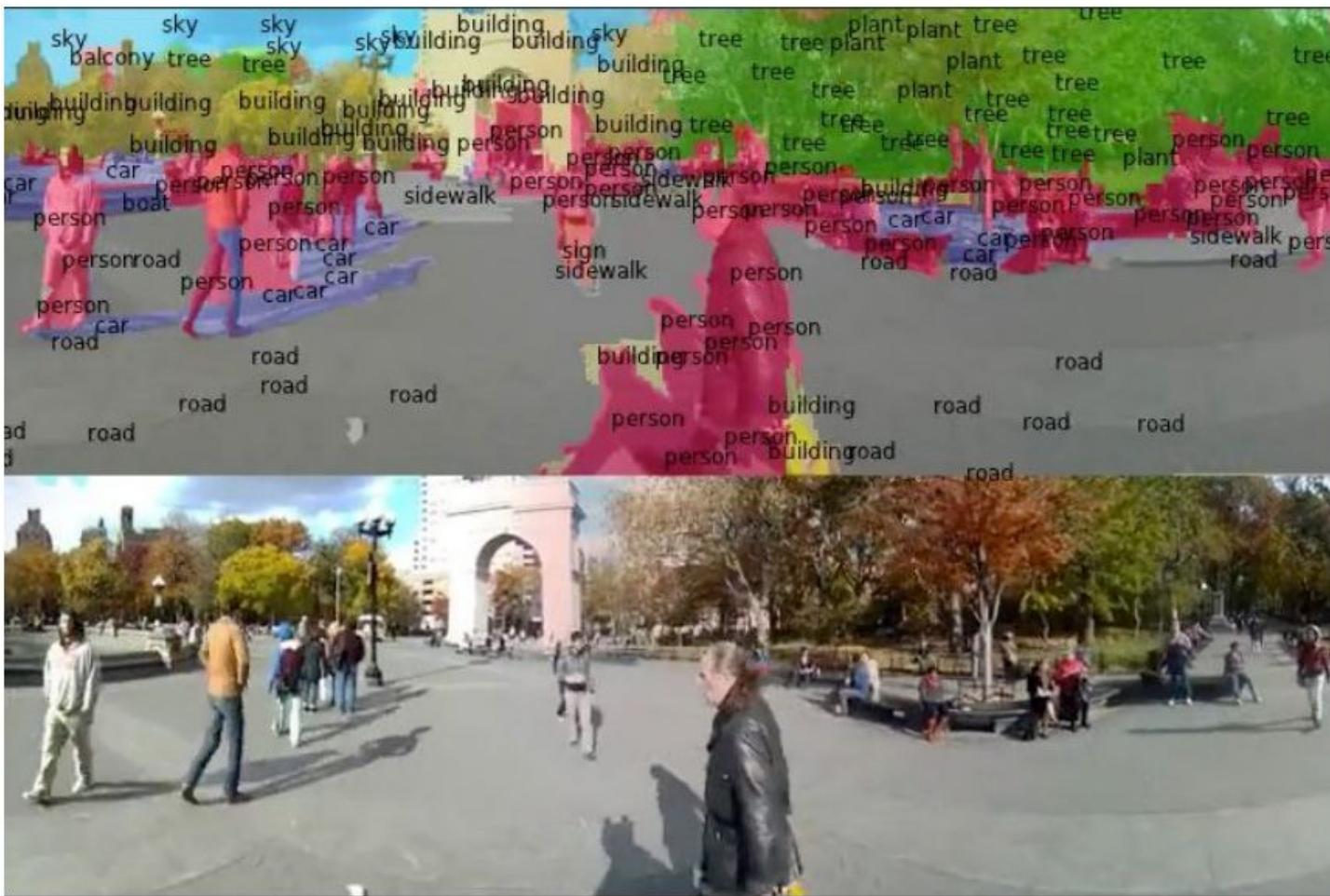


Detecção de objetos



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentação



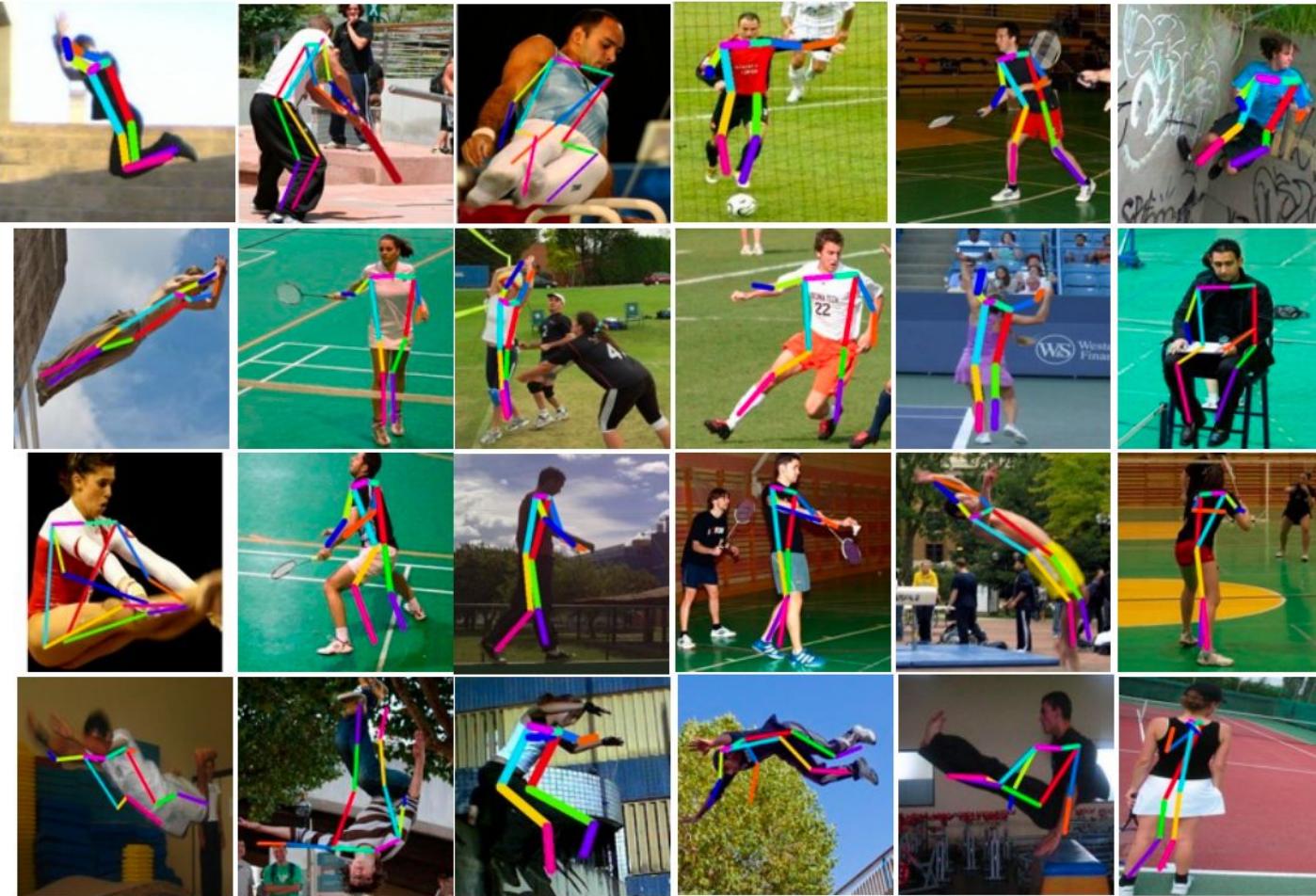
[Farabet et al., 2012]

Self-driving cars

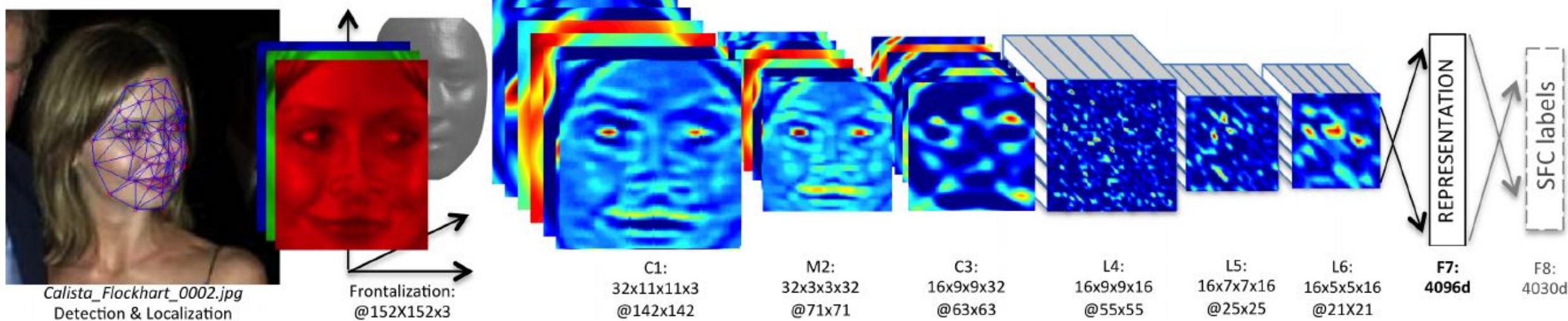
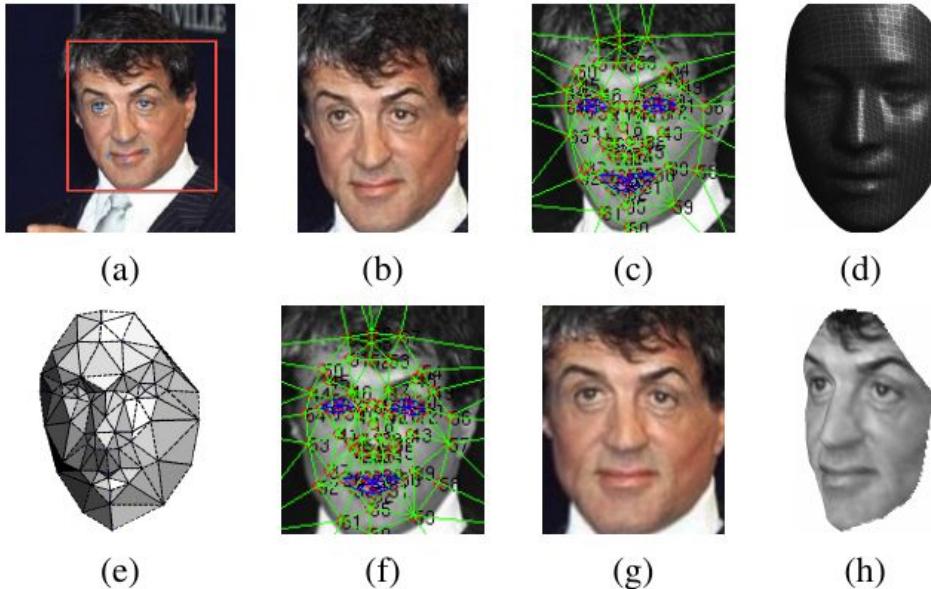


NVIDIA Tesla line

Detecção de pose

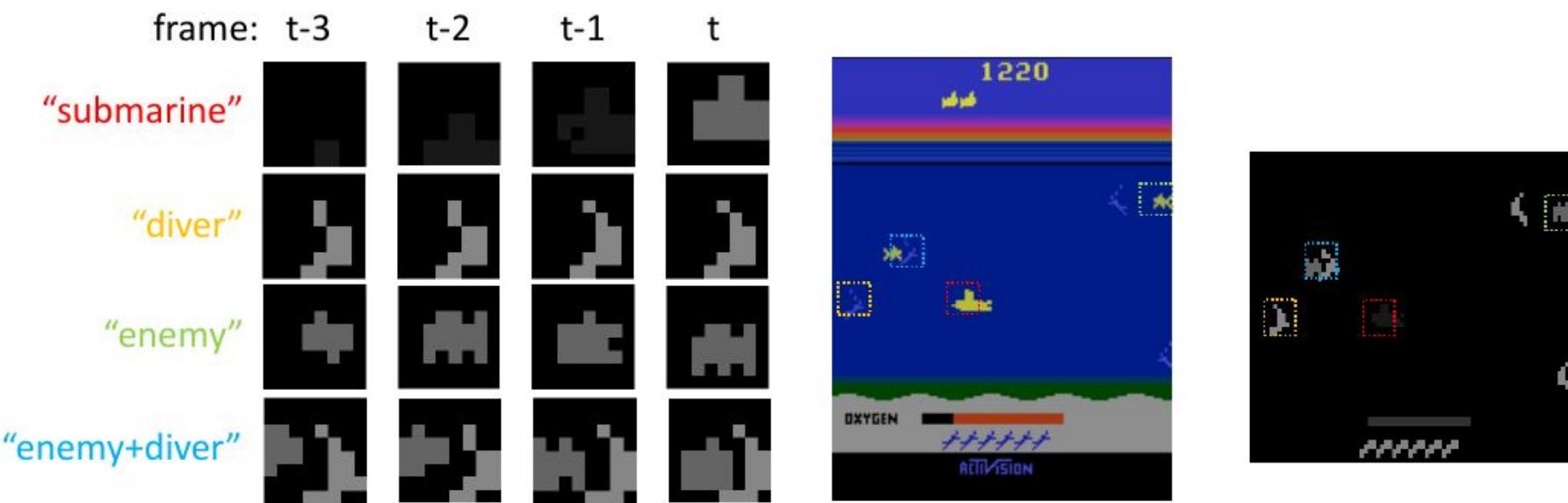
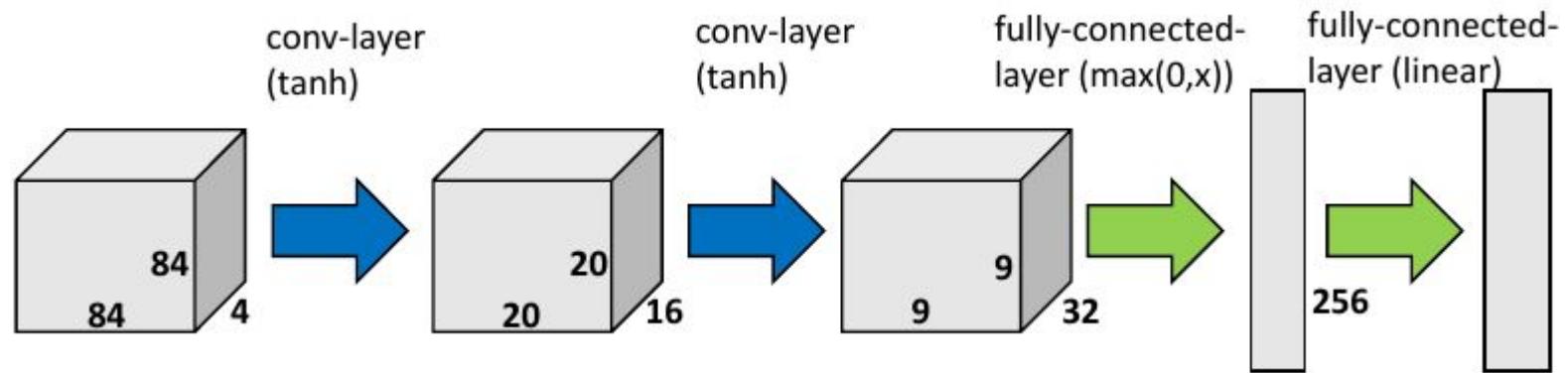


Detecção de face



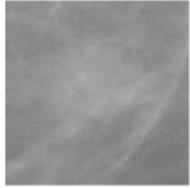
[Taigman, Yaniv, et al. Deepface: Closing the gap to human-level performance in face verification, 2014.](#)

IA para jogos

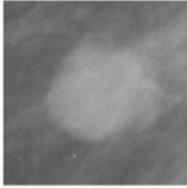


Classificação de imagens médicas

Benign



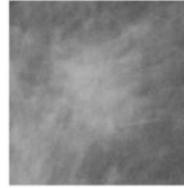
Benign



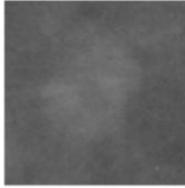
Malignant



Malignant



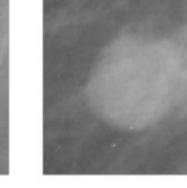
Benign



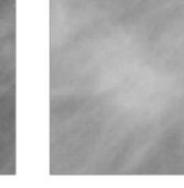
Benign



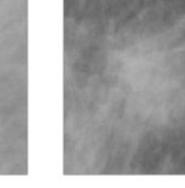
Benign



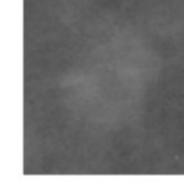
Malignant



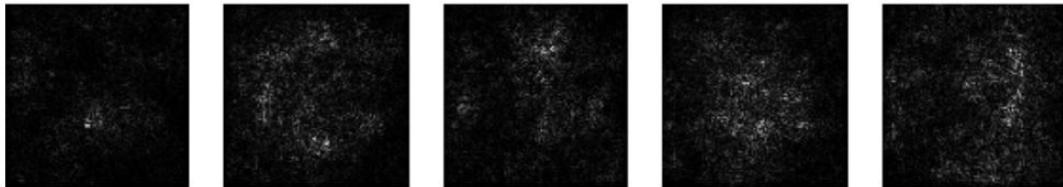
Malignant



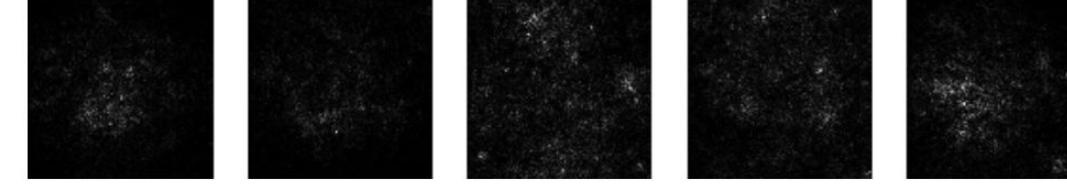
Benign



(a) AlexNet



(b) GoogLeNet



Model	Accuracy	Precision	Recall	# Epochs
Baseline (Aug-Large Context)	0.604	0.587	0.703	35
AlexNet (Aug - Large Context)	0.890	0.908	0.868	30
GoogLeNet (Aug - Large Context)	0.929	0.924	0.934	30

Geração de *captions* de imagens

No errors



*A white teddy bear sitting in
the grass*



*A man riding a wave on
top of a surfboard*

Transferência de estilo



Problemas de Visão Computacional

- A visão computacional é uma das **aplicações** que estão rapidamente ativas graças ao aprendizado profundo
- Mudanças rápidas na visão computacional permitem o desenvolvimento de **novos produtos** que não eram possíveis há alguns anos
- Algumas das **aplicações** da visão computacional que estão usando o aprendizado profundo incluem:
 - Carros autodirigidos
 - Reconhecimento facial
 - Geração de novos tipos de arte

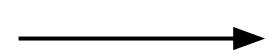
Problemas de Visão Computacional

- Técnicas de aprendizado profundo para a visão computacional estão sempre evoluindo, criando novas arquiteturas que podem nos ajudar em **outras áreas** além da visão computacional
- Por exemplo, algumas idéias de visão computacional foram aplicadas no **reconhecimento de fala**
- Exemplos de **problemas** de visão computacional incluem:
 - Classificação de imagem
 - Detecção de objetos
 - Detectar objeto e localizá-los
 - Transferência de estilo neural
 - Altera o estilo de uma imagem usando outra imagem

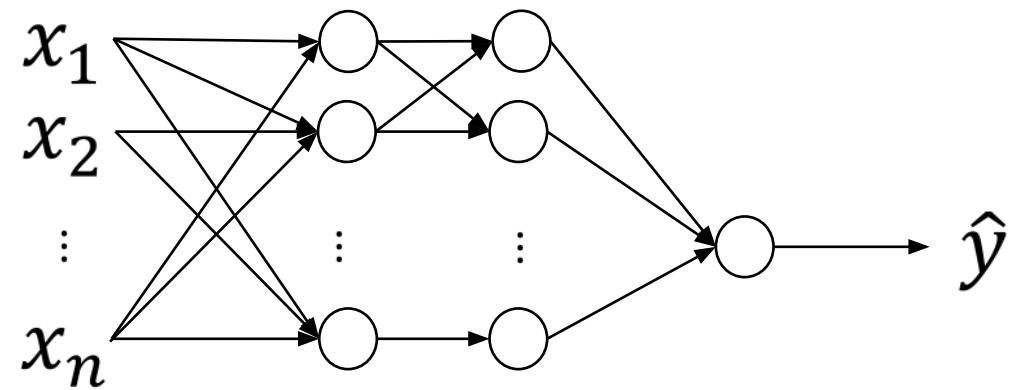
Aprendizado Profundo em Imagens



64x64



Gato? (0/1)



Aprendizado Profundo em Imagens

- Um dos desafios do problema de visão computacional é que as imagens podem ser **tão grandes** e queremos um algoritmo rápido e preciso para trabalhar com isso
- Por exemplo, uma imagem de **1000x1000** será representada por **3 milhões** de *features* para uma rede neural toda conectada
- Se a camada oculta seguinte contém **1000 unidades**, então vamos ter que aprender pesos da forma $(1000 \times 3\text{ milhões})$, ou **3 bilhões** de parâmetros apenas na primeira camada
- Isso é **computacionalmente muito caro!**
- Uma das soluções é criar isso usando **camadas de convolução**, em vez das camadas totalmente conectadas.

Convolutional Neural Networks

Visão geral

Visão geral

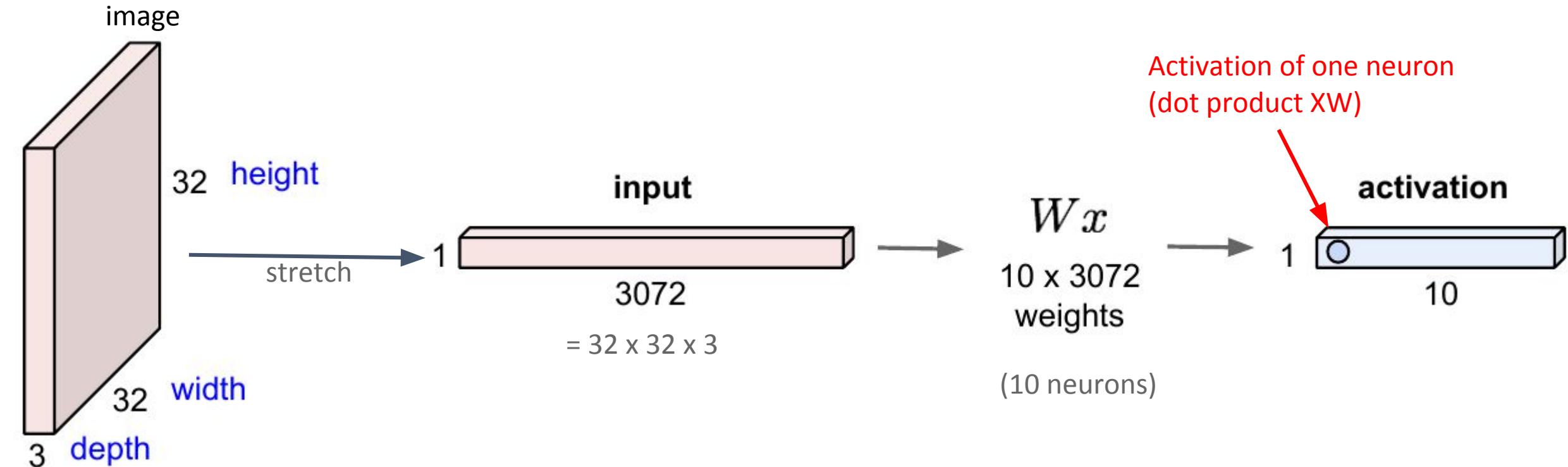


What We See

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 75 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 65 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

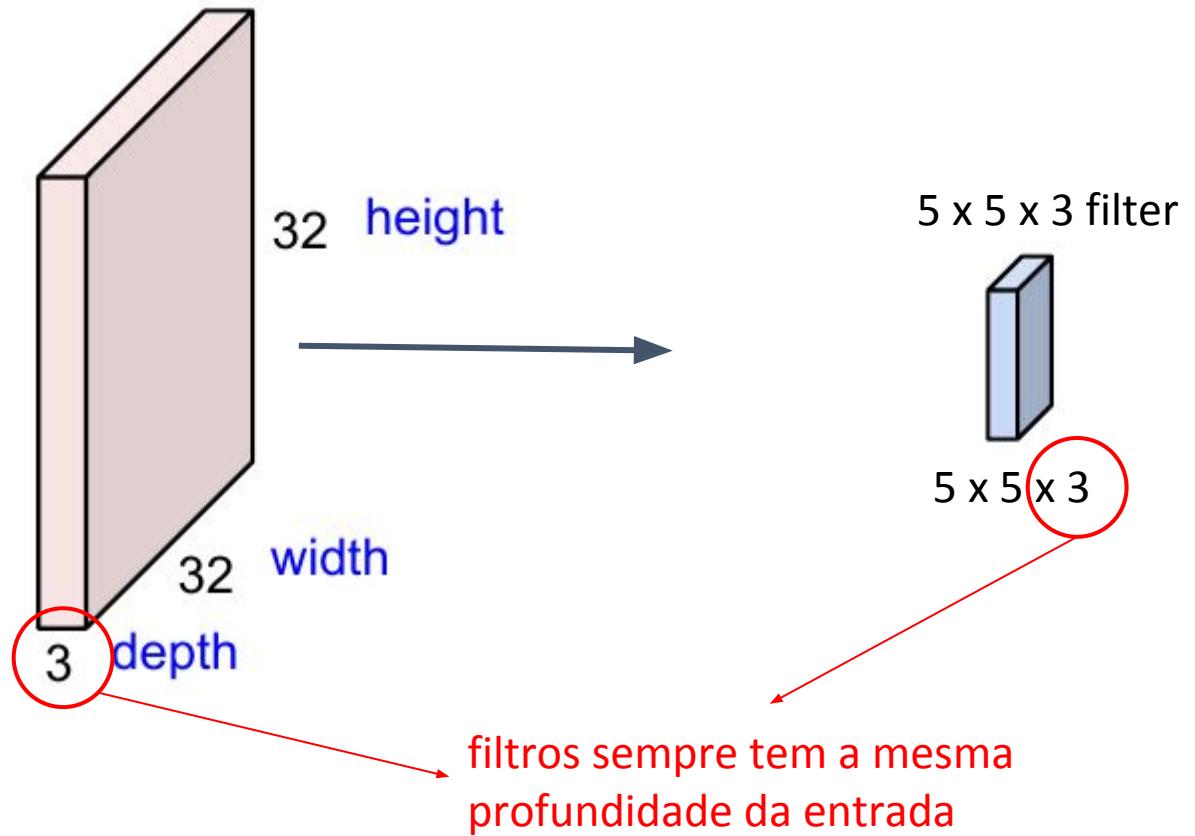
What Computers See

Camada totalmente conectada

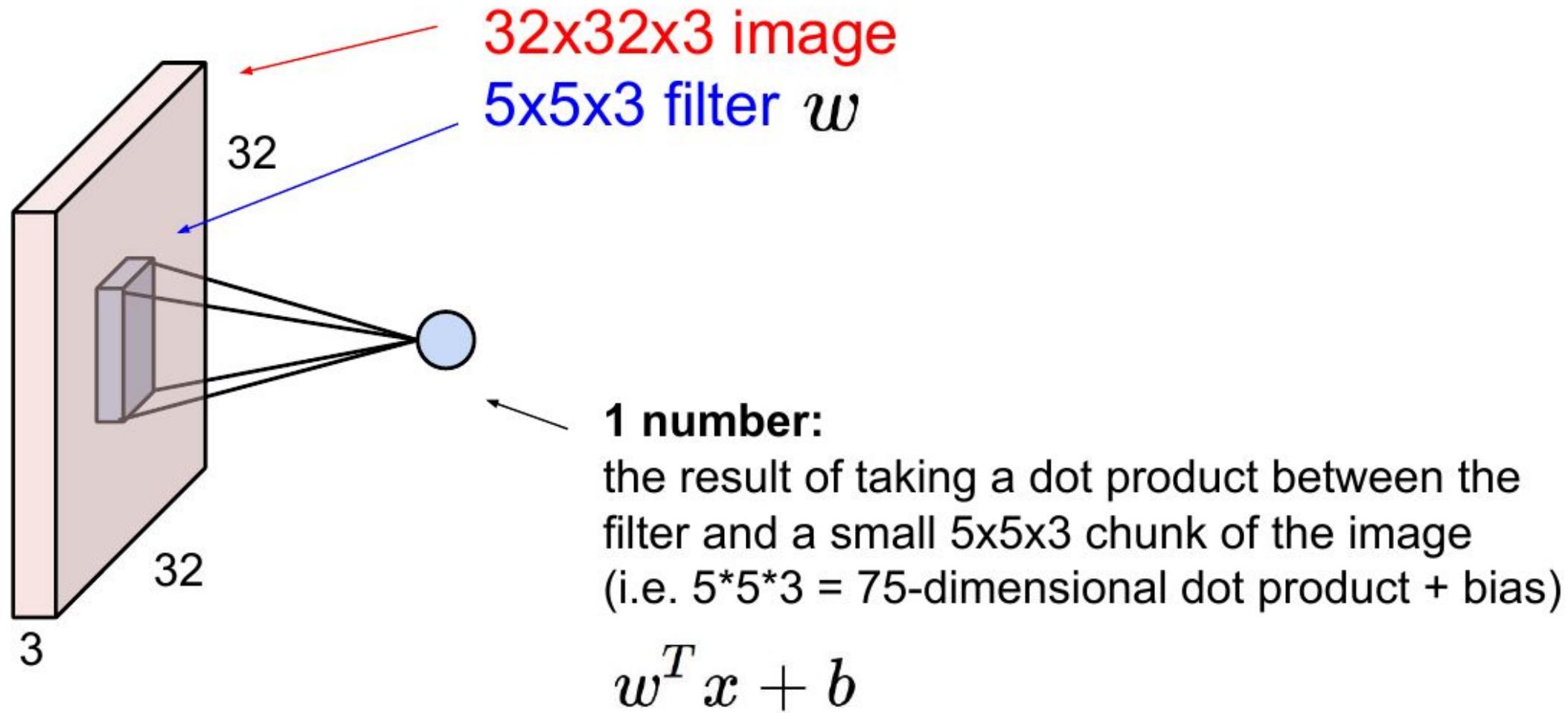


Camada convolucional

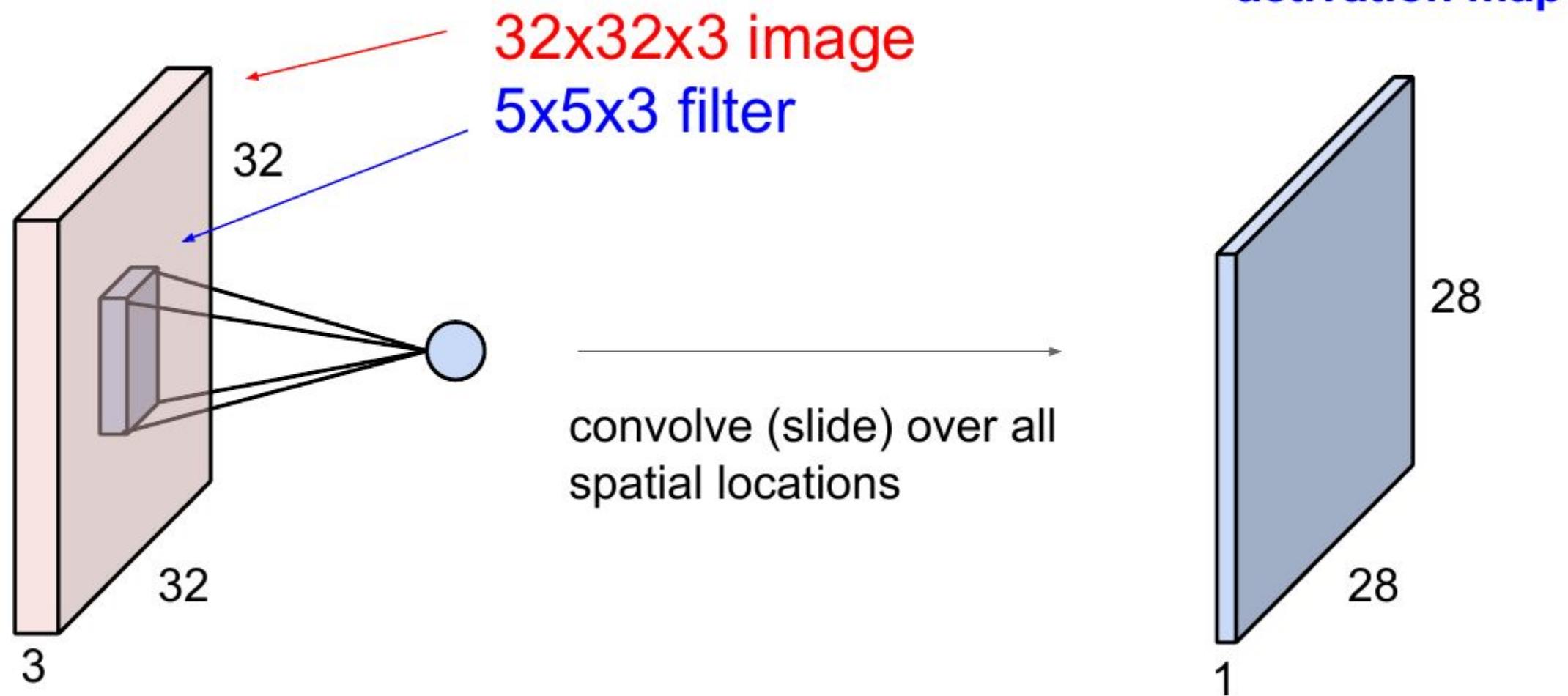
32 x 32 x 3 image



Camada convolucional

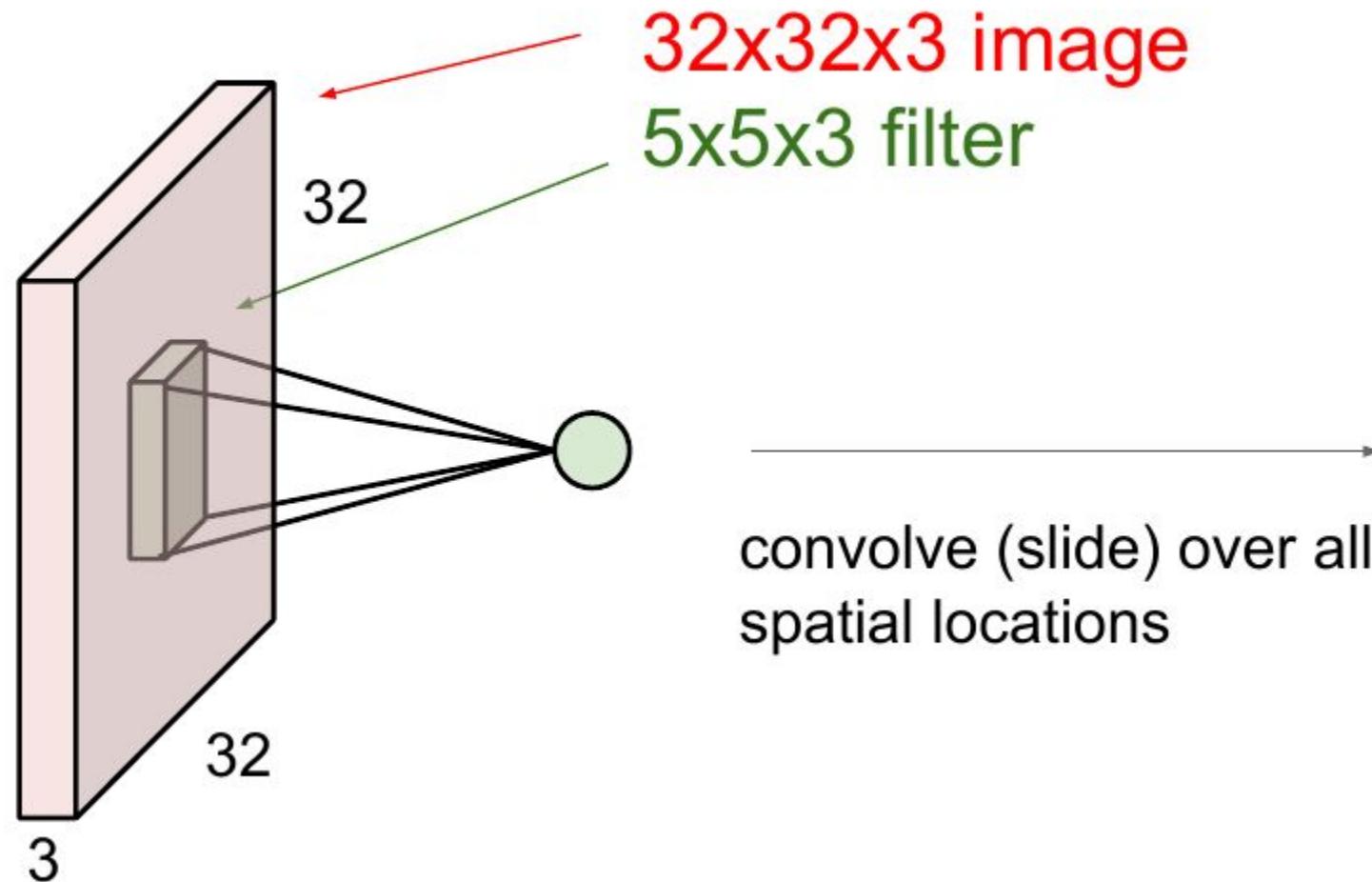


Camada convolucional

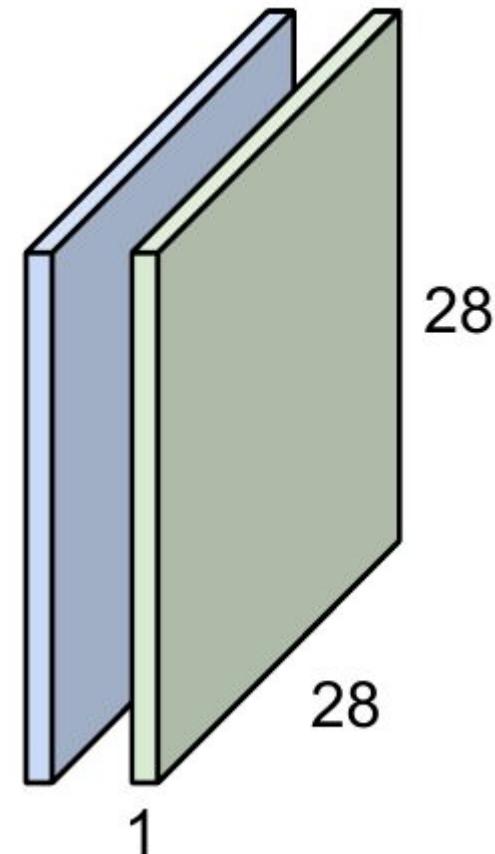


Camada convolucional

Considere um segundo filtro, um “**filtro verde**”

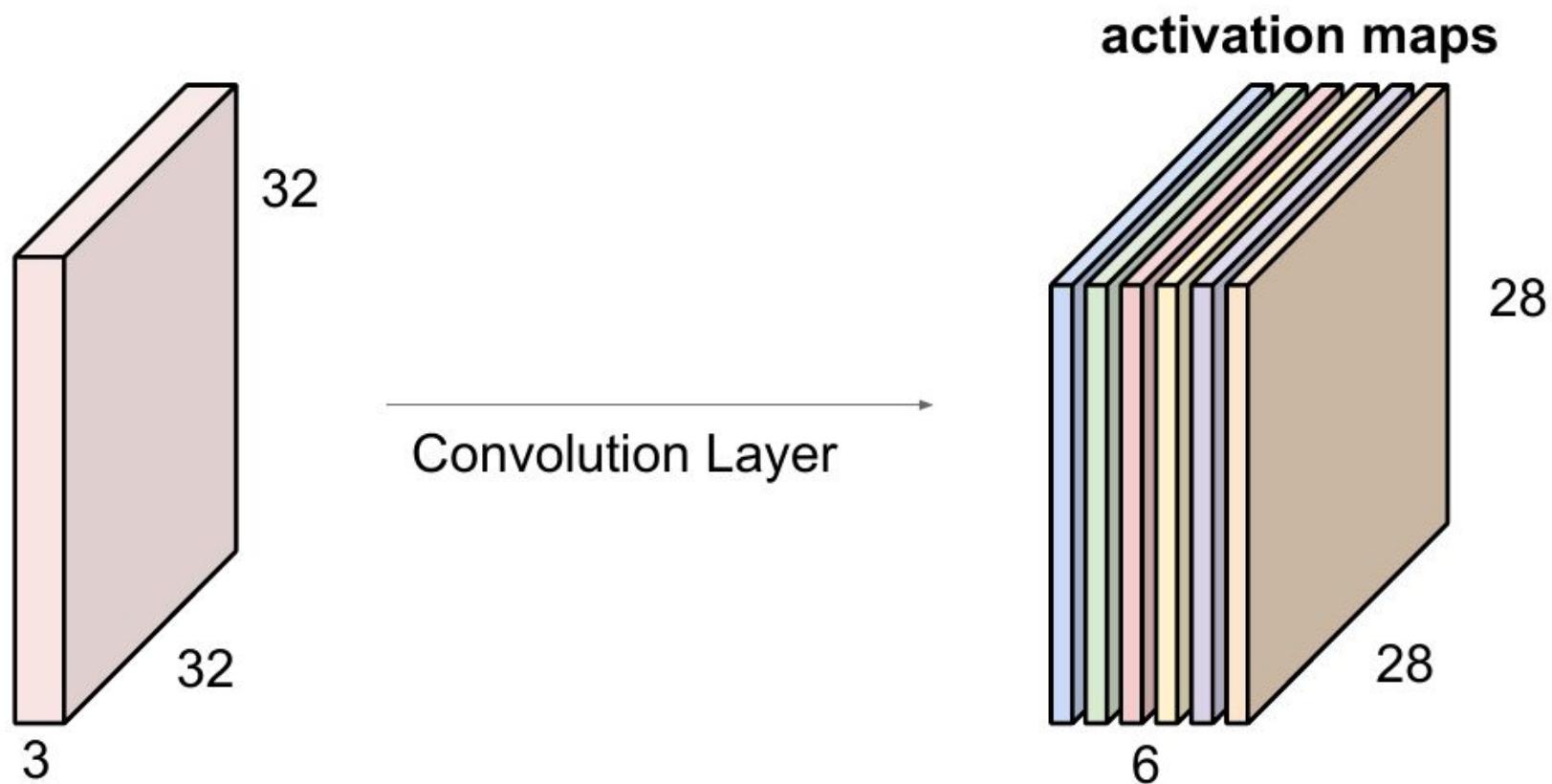


activation maps



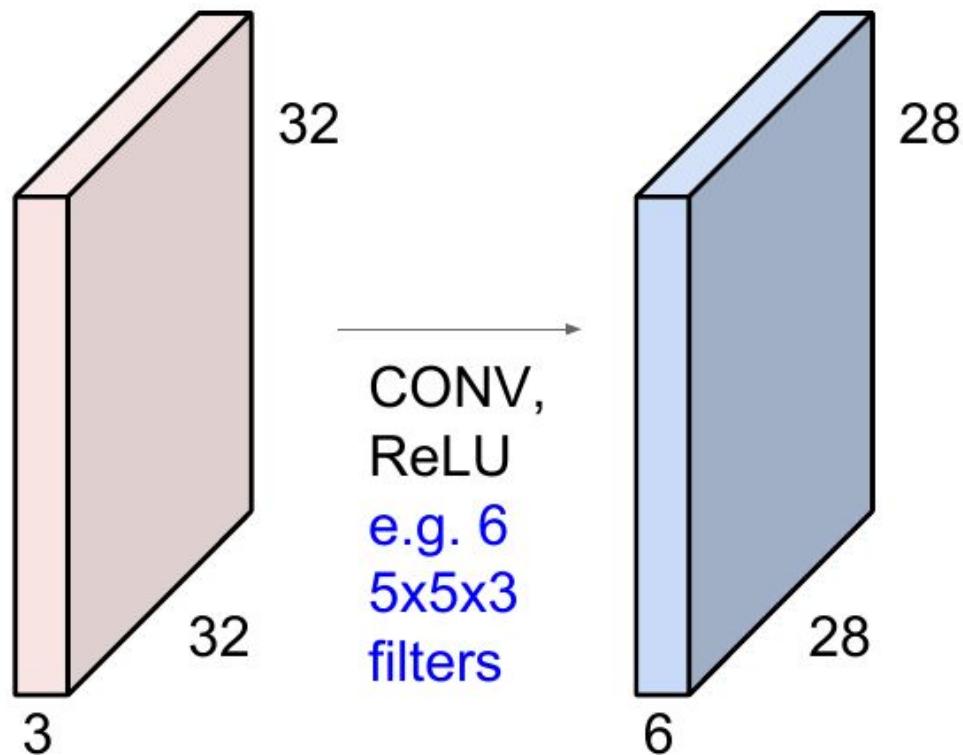
Camada convolucional

Se tivermos, por exemplo, 6 filtros ($5 \times 5 \times 3$), então teremos 6 mapas de ativação:



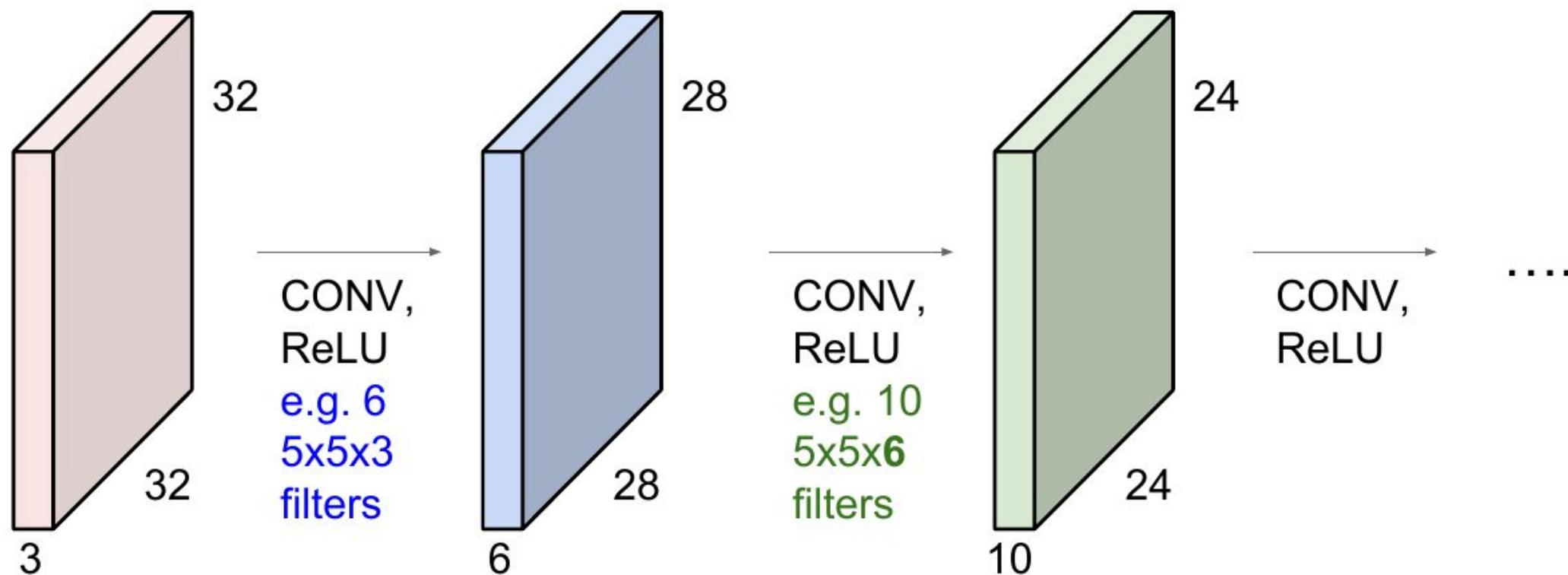
Prévia

Uma ConvNet é uma sequência de camadas de convolução, intercalada com funções de ativação:

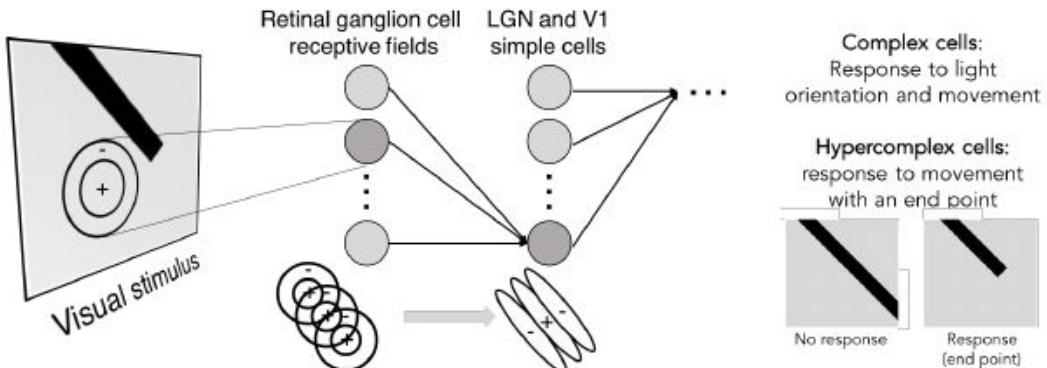
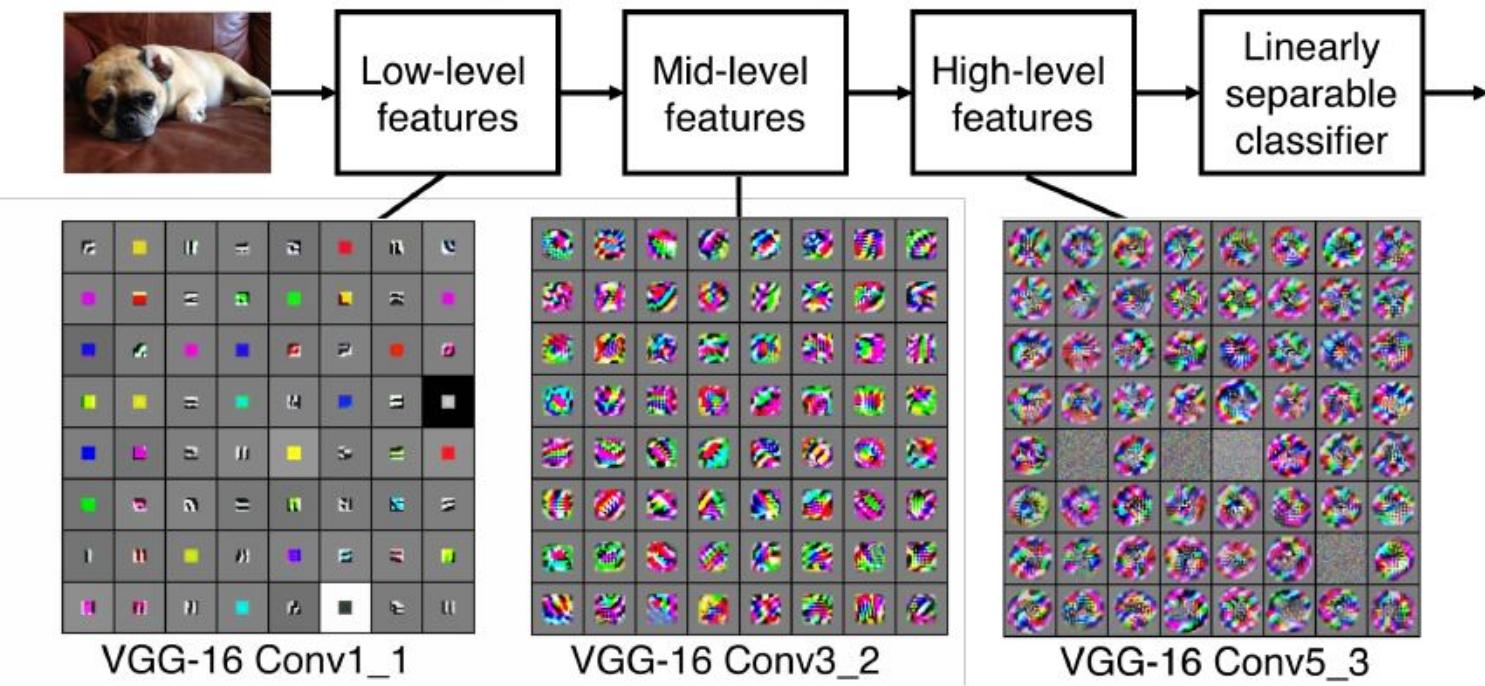


Prévia

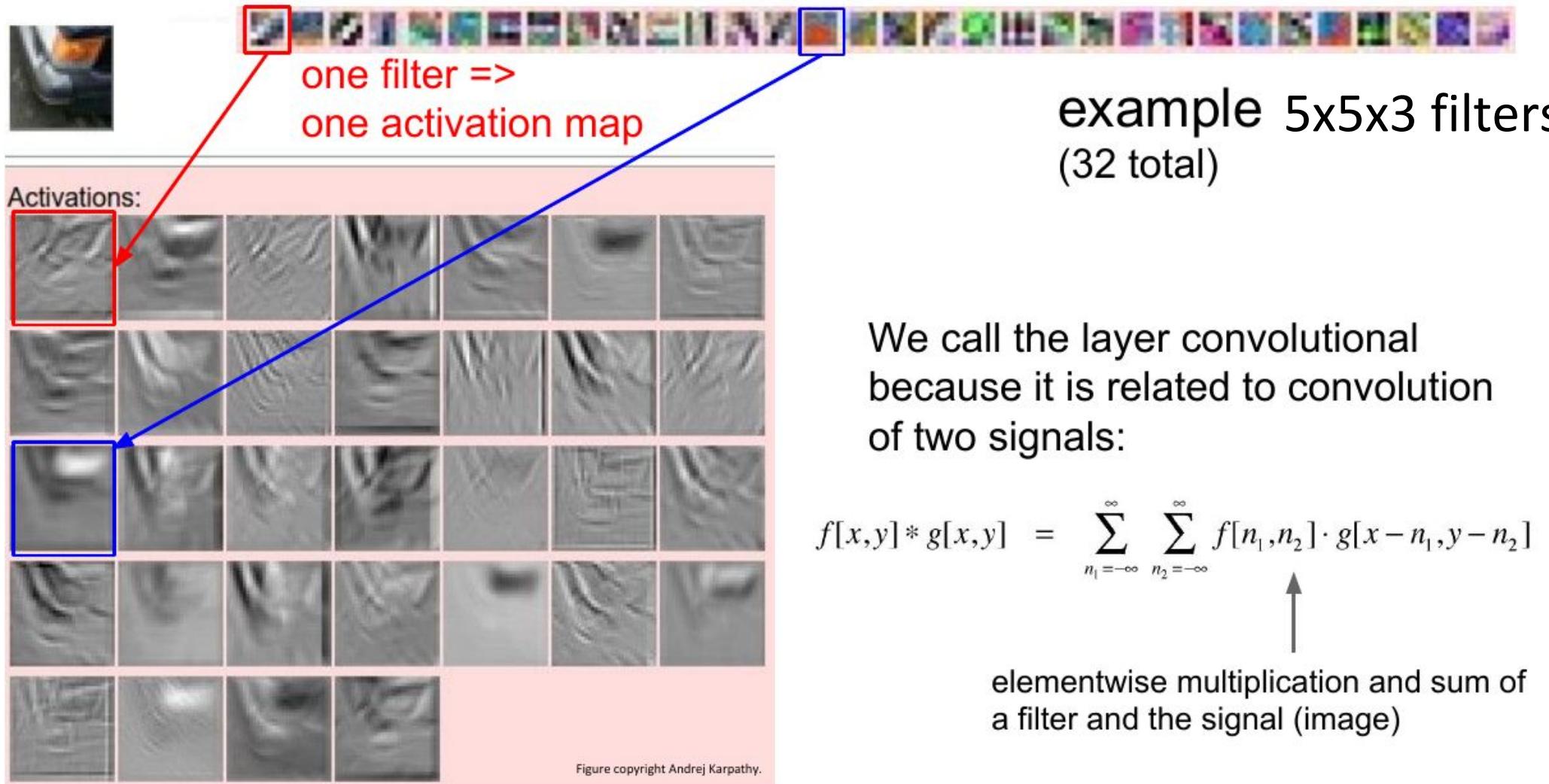
Uma ConvNet é uma sequência de camadas de convolução, intercalada com funções de ativação:



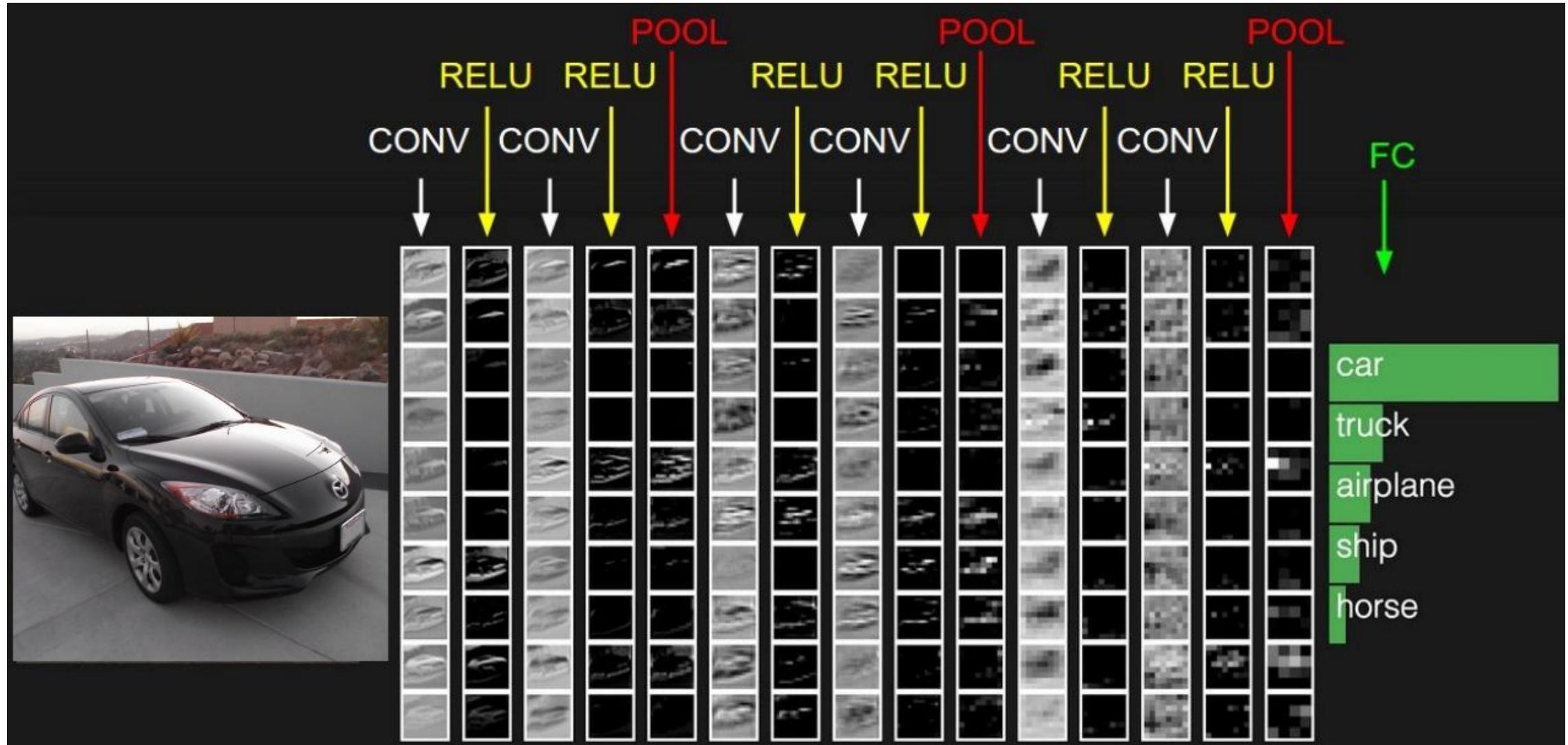
Prévia: hierarquia de filtros



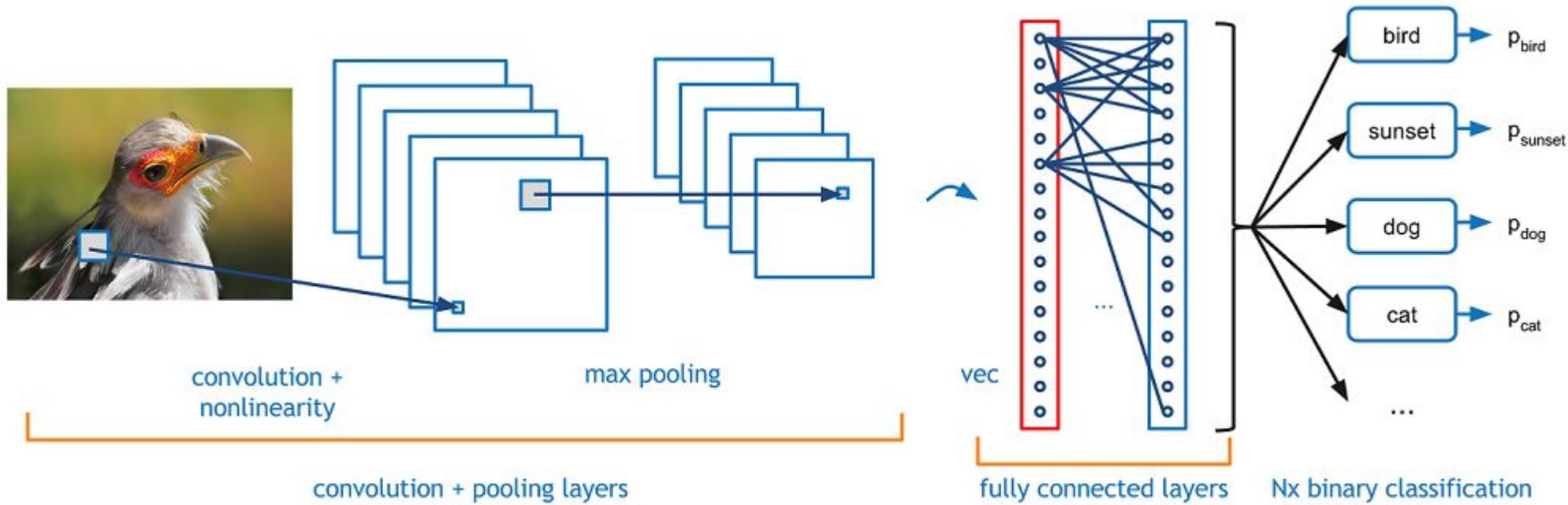
Prévia: mapas de ativação



Prévia



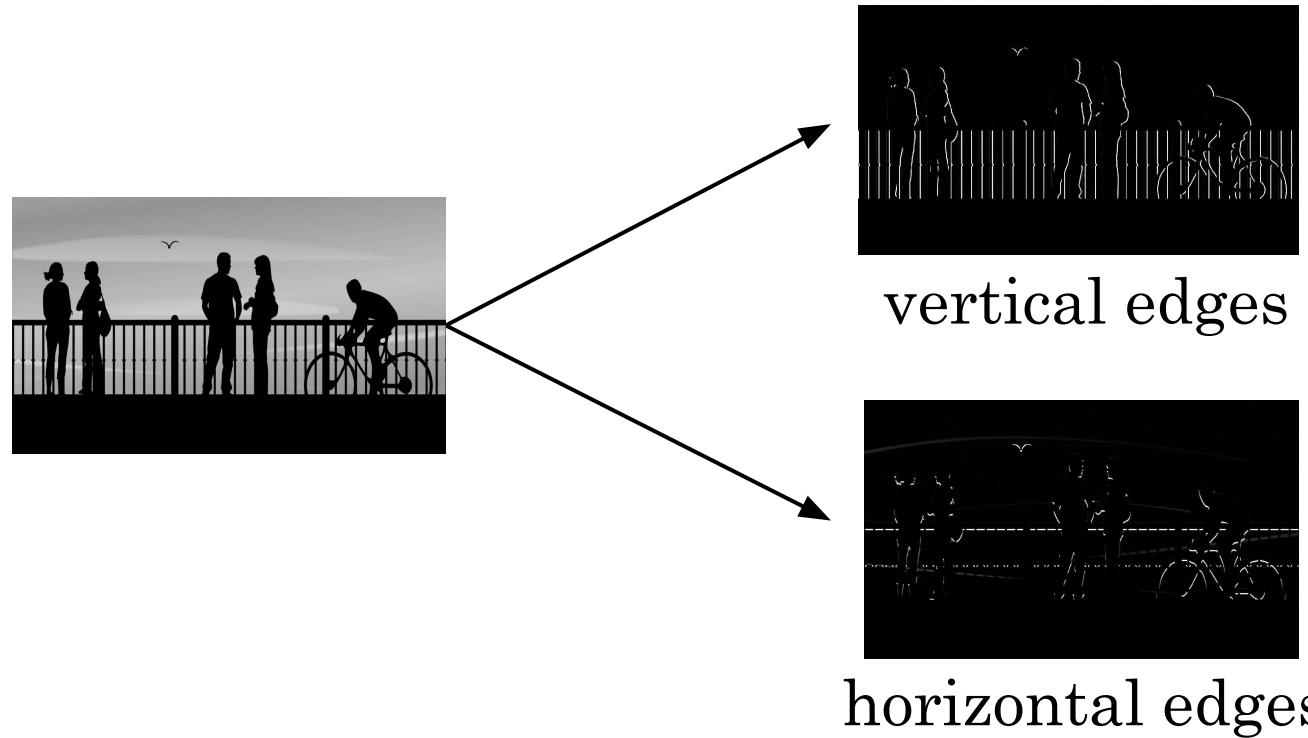
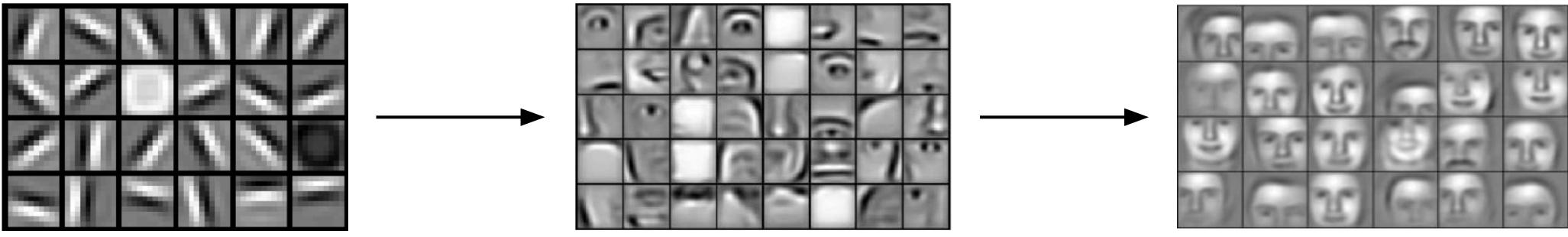
Visão geral



Convolutional Neural Networks

Exemplo de detecção
de bordas

Detecção de bordas



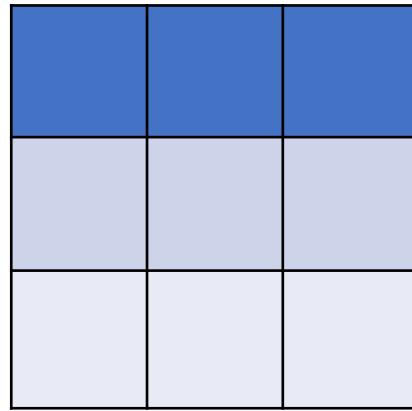
Detecção de bordas

- A **operação de convolução** é um dos blocos fundamentais de uma CNN
- Um dos exemplos sobre a convolução é a operação de **detecção de bordas** da imagem
- As **primeiras camadas** da CNN detectam bordas
- As **camadas intermediárias** detectam partes de objetos
- As **camadas posteriores** juntam essas partes para produzir uma saída
- Em uma imagem, podemos detectar **bordas verticais, bordas horizontais ou borda total**

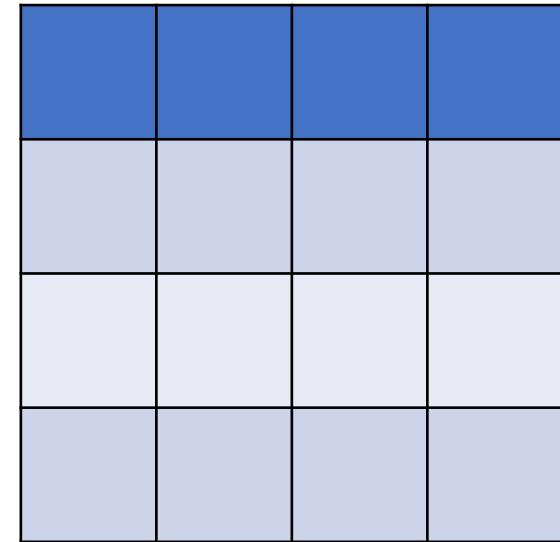
Detecção de bordas verticais

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*



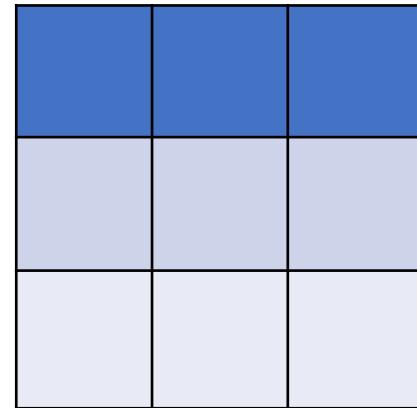
=



Detecção de bordas verticais

3 ¹	0 ⁰	1 ⁻¹	2 ⁻¹	7 ⁰	4 ⁻¹
1 ¹	5 ⁰	8 ⁻¹	9 ⁻¹	3 ⁰	1 ⁻¹
2 ¹	7 ⁰	2 ⁻¹	5 ⁰	1 ⁻¹	3 ⁻¹
0 ¹	1 ⁰	3 ⁻¹	1 ⁻¹	7 ⁰	8 ⁻¹
4	2	1	6	2	8
2	4	5	2	3	9

*



=

0	-2	-4	-7
-3	-2	-3	-16

Detecção de bordas verticais

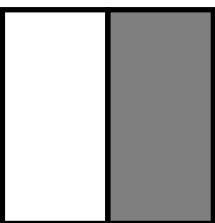
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

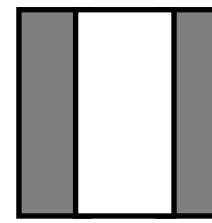
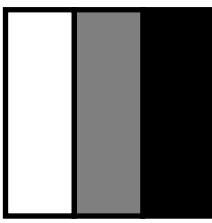
1	0	-1
1	0	-1
1	0	-1

=

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



*



Detecção de bordas verticais

- No último exemplo, uma matriz **6x6** convolvida com filtro/kernel **3x3** nos fornece uma matriz **4x4**
- Se você fizer a operação de convolução no **TensorFlow**, usará a função `tf.nn.conv2d`
- Em **keras** você usará a função `Conv2d`

Detecção de bordas verticais

- O filtro de detecção de bordas verticais encontrará um local 3x3 em uma imagem em que haja uma **região clara** seguida por uma **região escura**
- Se aplicarmos esse filtro a uma **região clara** seguida por uma **região escura**, ele deverá encontrar as arestas entre as duas cores como um **valor positivo**
- Mas se aplicarmos o mesmo filtro a uma **região escura** seguida por uma **região branca**, isso nos dará **valores negativos**
- Para resolver isso, podemos usar a função `abs` (módulo) para torná-lo positivo

Convolutional Neural Networks

Mais sobre
detecção de bordas

Exemplos de detecção de bordas verticais

$$\begin{matrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{matrix}$$


*

$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$


=

$$\begin{matrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{matrix}$$


$$\begin{matrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{matrix}$$


*

$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$


=

$$\begin{matrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{matrix}$$


Detecção de bordas verticais e horizontais

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

Vertical

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

Horizontal

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

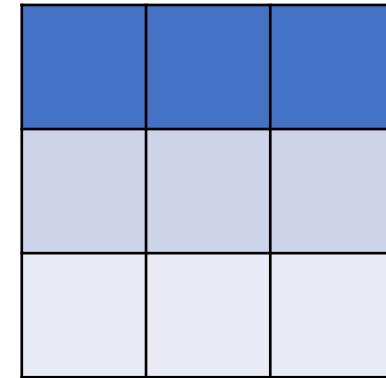
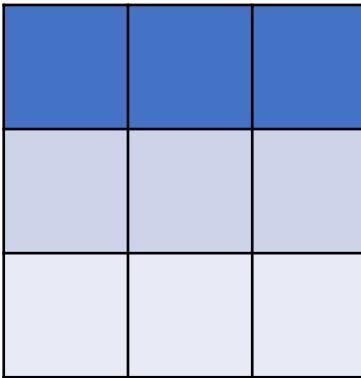
1	1	1
0	0	0
-1	-1	-1

=

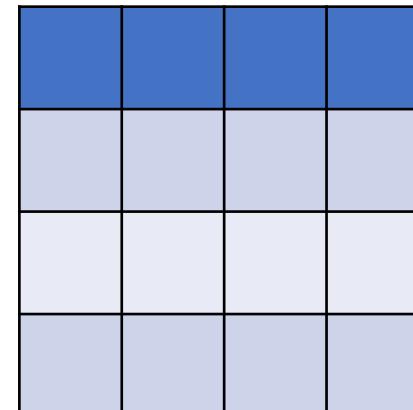
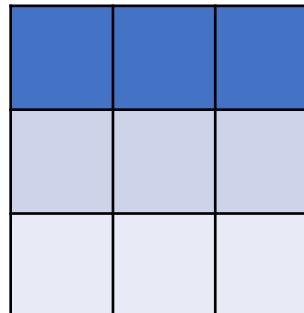
0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Aprendizado de detecção de bordas

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



3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9



Detecção de bordas

- Detecção de borda horizontal

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

Detecção de bordas

- Há muitas maneiras de colocar números dentro das detecções de borda horizontal ou vertical
- Por exemplo, aqui está o **filtro de Sobel**:
 - A ideia é cuidar da linha do meio

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

Detecção de bordas

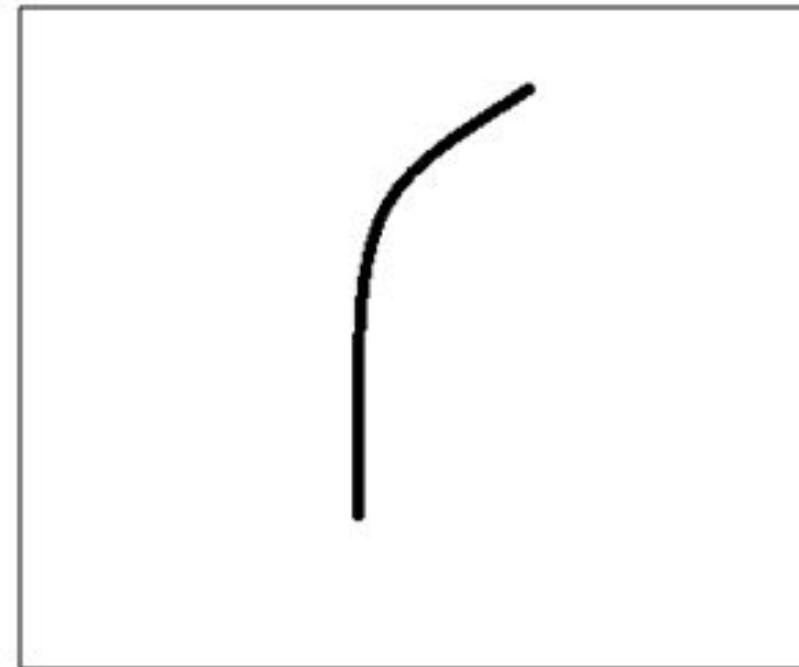
- Também algo chamado **filtro Scharr**:
 - A ideia é tomar muito cuidado com a linha do meio

3	0	-3
10	0	-10
3	0	-3

Detecção de bordas

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

Pixel representation of filter

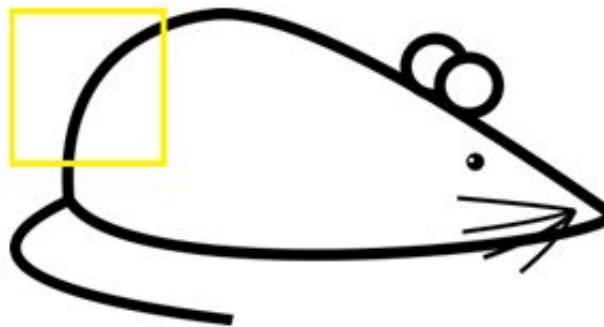


Visualization of a curve detector filter

Detecção de bordas



Original image



Visualization of the filter on the image



Visualization of the receptive field

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

Pixel representation of the receptive field

*

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

Pixel representation of filter

$$\text{Multiplication and Summation} = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 \text{ (A large number!)}$$

Detecção de bordas



Visualization of the filter on the image

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

Detecção de bordas



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	A small image of the same dog's head, identical to the input.
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	A small image where the edges of the dog's features are highlighted in white against a black background.
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	A small image where the edges of the dog's features are highlighted in white against a black background.
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	A small image of the dog's head with increased contrast and some noise.
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	A small image of the dog's head with a smooth, blurred appearance.
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	A small image of the dog's head with a very smooth, blurred appearance.

Detecção de bordas

- O que aprendemos no aprendizado profundo é que **não precisamos fixar** esses números
- Podemos tratá-los como pesos e depois **aprendê-los**
- Ele pode aprender horizontal, vertical, angular ou **qualquer tipo de borda** automaticamente, em vez de obtê-los manualmente

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

Detecção de bordas

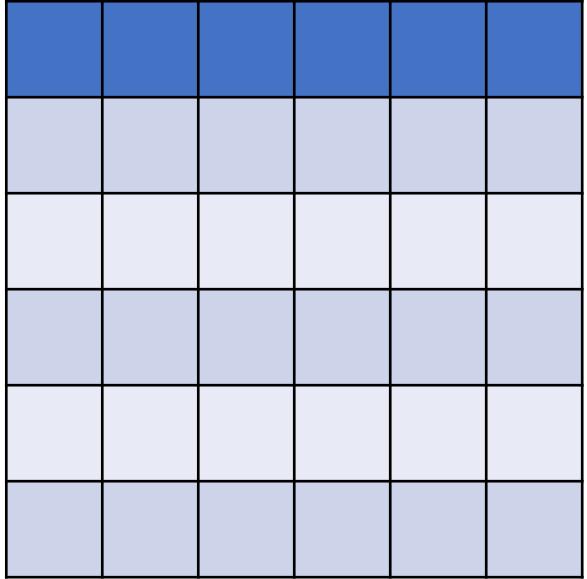


Exemplo de filtros aprendidos por Krizhevsky et al.

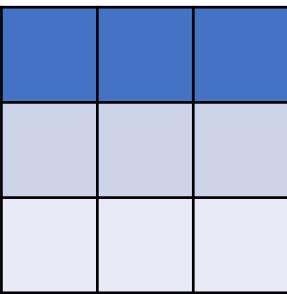
Convolutional Neural Networks

Padding

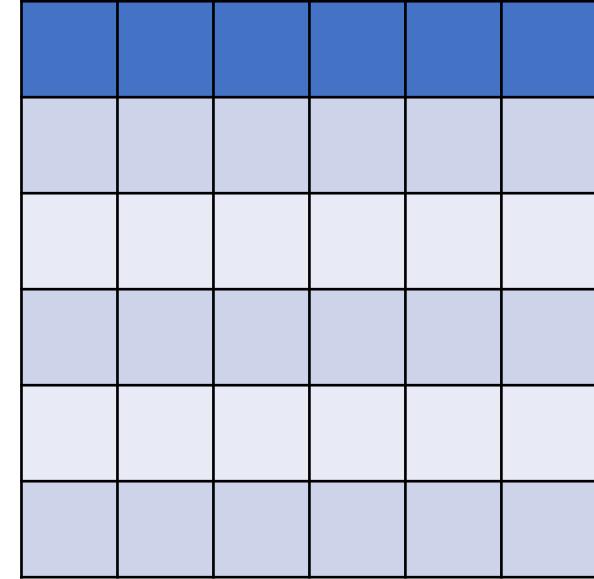
Padding



*



=



Convoluçãoes “Valid” e “Same”

“Valid”:

“Same”: Faça o *pad* de forma que a saída tenha o mesmo tamanho que a entrada

Padding

- Para usar redes neurais profundas, realmente precisamos usar **padding**
- Vimos que uma matriz **6x6** convolvida com filtro **3x3** nos dá uma matriz **4x4**
- Para dar uma regra geral, se uma matriz **nxn** for convolvida com um filtro **fxf**, a saída é uma matriz **(n-f+1) x (n-f+1)**
- A operação de convolução **encolhe** a matriz se **f > 1**

Padding

- Queremos aplicar a operação de convolução várias vezes, mas se a imagem diminuir, **perderemos** muitos dados nesse processo
- Além disso, os pixels das bordas **são usados menos** que outros pixels em uma imagem
- Então os problemas com convoluções são:
 - **Encolhe** a saída
 - Joga fora muita informação que estão nas bordas

Padding

- Para resolver esses problemas, podemos preencher (*pad*) a imagem de entrada antes da convolução, adicionando algumas linhas e colunas a ela
- Vamos chamar a quantidade de preenchimento *p*, que é o número de linhas/colunas que vamos **inserir** na parte **superior**, **inferior**, **esquerda** e **direita** da imagem
- Em quase todos os casos, os **valores** de preenchimento são zeros
- A regra geral agora, se uma matriz **nxn** for convolvida com um filtro **fxf** e *padding p*, a matriz de saída é **(n + 2p-f + 1) x (n + 2p-f + 1)**

Padding

- Exemplo:
 - se $n = 6$, $f = 3$ e $p = 1$
 - a imagem de saída terá $n + 2p - f + 1 = 6 + 2 - 3 + 1 = 6$
 - Ou seja, mantemos o tamanho da imagem original
- As convoluções “*same*” são uma convolução com um bloco para que o tamanho de saída seja o mesmo que o tamanho de entrada
- É dado pela equação:
$$p = (f-1) / 2$$
- Na visão computacional, f geralmente é ímpar, pois precisamos dar foco usualmente para ao valor central

Convolutional Neural Networks

Convoluçãọes
Strided

Convolução *strided*

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

*

3	4	4
1	0	2
-1	0	3

=

Convolução *strided*

2	3	3	4	7	3	4	4	6	3	2	4	9	4	
6	1	6	0	9	1	8	0	7	1	4	0	3	2	
3	-3	4	4	8	-3	3	4	8	-3	9	4	7	4	
1	7	1	8	0	3	1	6	0	6	1	3	0	4	2
4	-3	2	4	1	3	8	4	3	-3	4	4	6	4	
1	3	1	2	0	4	1	1	0	9	1	8	0	3	2
0	-1	1	0	3	-3	9	0	2	-3	1	0	4	3	

*

3	4	4
1	0	2
-1	0	3

=

Convoluçãoes *strided*

- Convolução *strided* é outra peça que é usada em CNNs
- Nós vamos chamar de *stride s*
- Quando estamos fazendo a operação de convolução, usamos **s** para nos informar o número de pixels que saltaremos quando estivermos convolvendo o filtro
- Nos últimos exemplos que descrevemos, **s=1**

Sumário das convoluções

$n \times n$ image $f \times f$ filter

padding p stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

Sumário das convoluções

- Agora a regra geral é:
- Se uma matriz $n \times n$ for convolvida com um filtro $f \times f$, *padding p* e *stride s*, a saída terá tamanho:
$$[(n + 2p - f) / s + 1] \times [(n + 2p - f) / s + 1]$$
- Se $(n + 2p - f) / s + 1$ não for inteiro, podemos pegar o chão deste valor

Nota técnica sobre correlação cruzada vs. convolução

Convolution in math textbook:

2	3	7	4	6	2
6	6	9	8	7	4
3	4	8	3	8	9
7	8	3	6	6	3
4	2	1	8	3	4
3	2	4	1	9	8

$$\begin{matrix} & \begin{matrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{matrix} \\ * & \end{matrix}$$

$$\begin{matrix} & \begin{matrix} 7 & 2 & 5 \\ 9 & 0 & 4 \\ -1 & 1 & 3 \end{matrix} \end{matrix}$$

Nota técnica sobre correlação cruzada vs. convolução

- Nos livros didáticos de matemática, uma operação *conv* (convolução) inverte o filtro
- O que estamos fazendo é uma correlação cruzada, mas a literatura de aprendizado profundo chama de operação *conv* (convolução)

Convolução “*same*”

- Novamente, as convoluções “*same*” fazem com que o tamanho da saída seja o mesmo da entrada

- A regra geral, quando o *stride s* varia, é:

$$p = (n * s - n + f - s) / 2$$

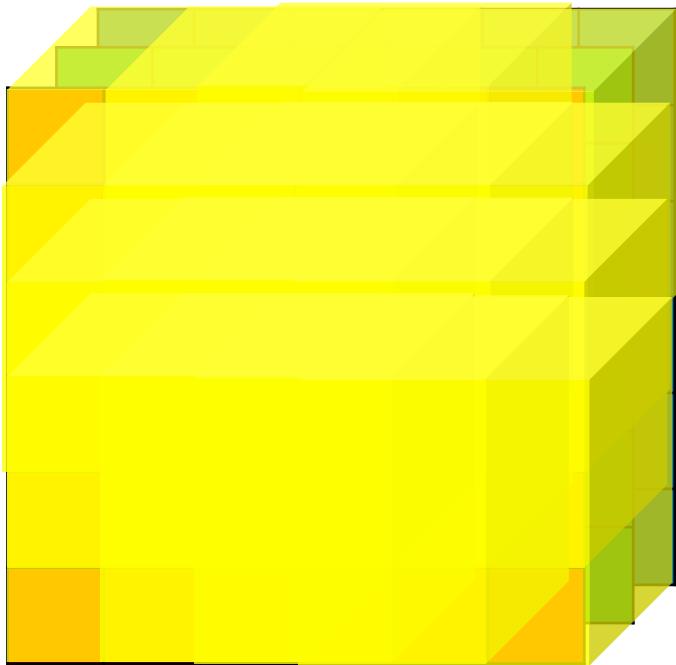
- Quando $s = 1$:

$$p = (f-1) / 2$$

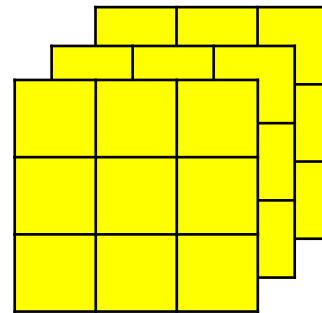
Convolutional Neural Networks

Convoluçãoções sobre
volumes

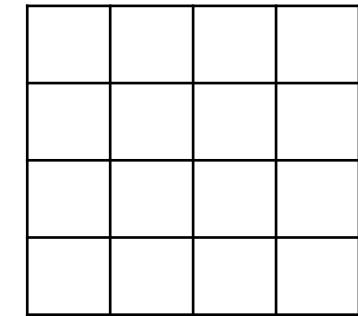
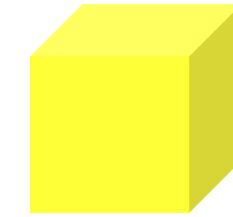
Convoluçãoes em imagens RGB



*

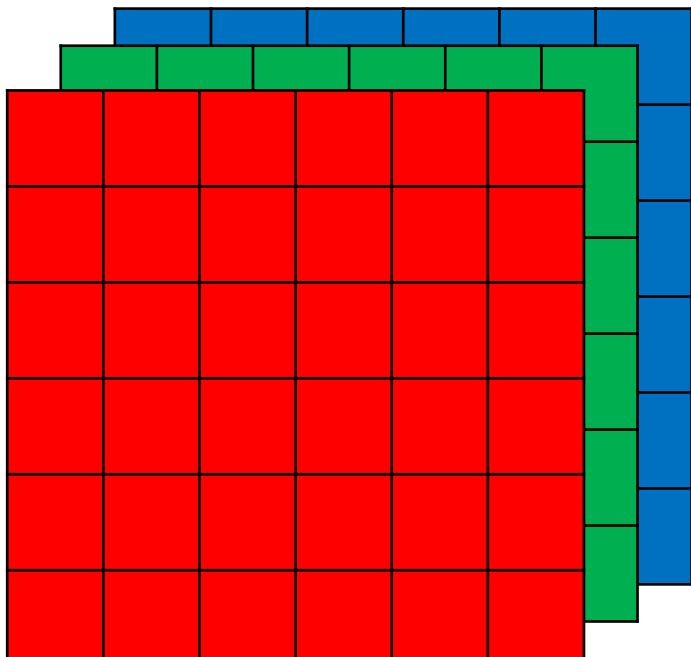


=



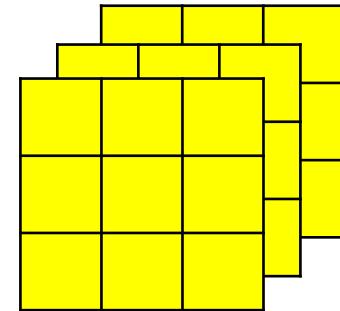
4×4

Múltiplos filtros



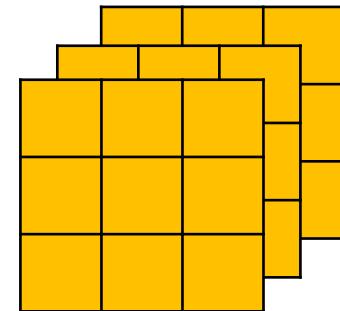
$6 \times 6 \times 3$

*



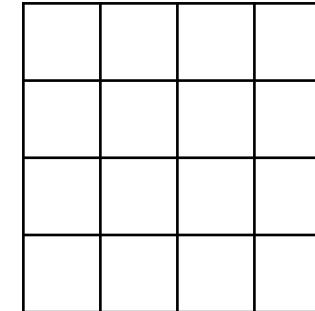
$3 \times 3 \times 3$

*



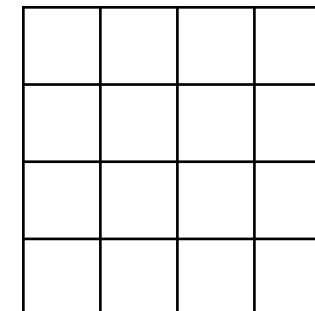
$3 \times 3 \times 3$

=



4×4

=



4×4

Convoluçãoes sobre volumes

- Vemos como a convolução funciona com imagens 2D, agora vamos ver como funciona em imagens 3D (**imagem RGB**)
- A entrada agora é uma imagem de altura, largura, # de canais com um filtro de altura, largura e o **mesmo número de canais**
 - Chamamos isso de filtros empilhados para cada canal

Exemplos

- Imagem de entrada: **6x6x3**
 - Filtro: **3x3x3**
 - Tamanho da imagem de saída: **4x4x1**
 - No último resultado **p = 0, s = 1**
 - A imagem de saída aqui é apenas 2D
- Imagem de entrada: **6x6x3**
 - **10 filtros: 3x3x3**
 - Imagem do resultado: **4x4x10**
 - No último resultado, **p = 0, s = 1**

Sumário de uma *ConvLayer*

- Aceita um volume de dimensões $W_1 \times H_1 \times D_1$
- Requer **quatro** hiperparâmetros:
 - Número de filtros n_c
 - Tamanho do filtro f
 - *Stride* s
 - *Padding* p
- Produz um volume de dimensões $W_2 \times H_2 \times D_2$, em que:
 - $W_2 = (W_1 - f + 2p)/s + 1$
 - $H_2 = (H_1 - f + 2p)/s + 1$
 - $D_2 = n_c$
- Devido ao compartilhamento de parâmetros, introduz $f \times f \times D_1$ pesos (weights) por filtro, para um total de $(f \times f \times D_1) \times n_c$ pesos e n_c biases
- No volume de saída, a d -ésima fatia (de tamanho $W_2 \times H_2$) é o resultado da execução de uma convolução “*valid*” do d -ésimo filtro sobre a entrada com um *stride* de s , e aplicando um deslocamento pelo d -ésimo bias

Configurações comuns:

n_c = potências de 2 (ex: 32, 64, ...)

$f=3, s=1, p=1$

$f=5, s=1, p=2$

$f=5, s=2, p=(o\ que\ couber)$

$f=1, s=1, p=0$