

CURSO PRÁTICO DE DEEP LEARNING COM PYTHON E KERAS

Redes Convolutivas =O

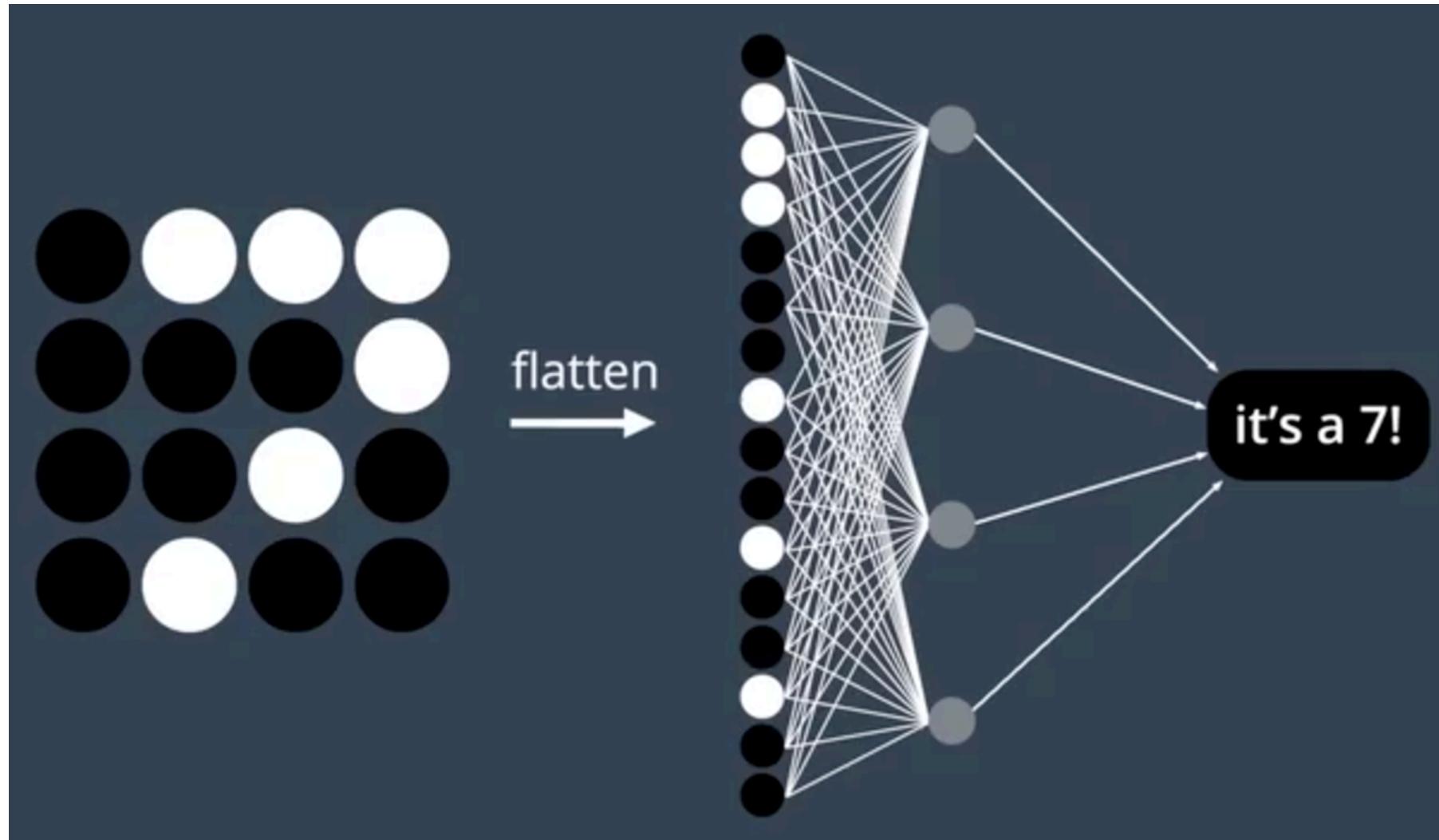
Dog vs Food



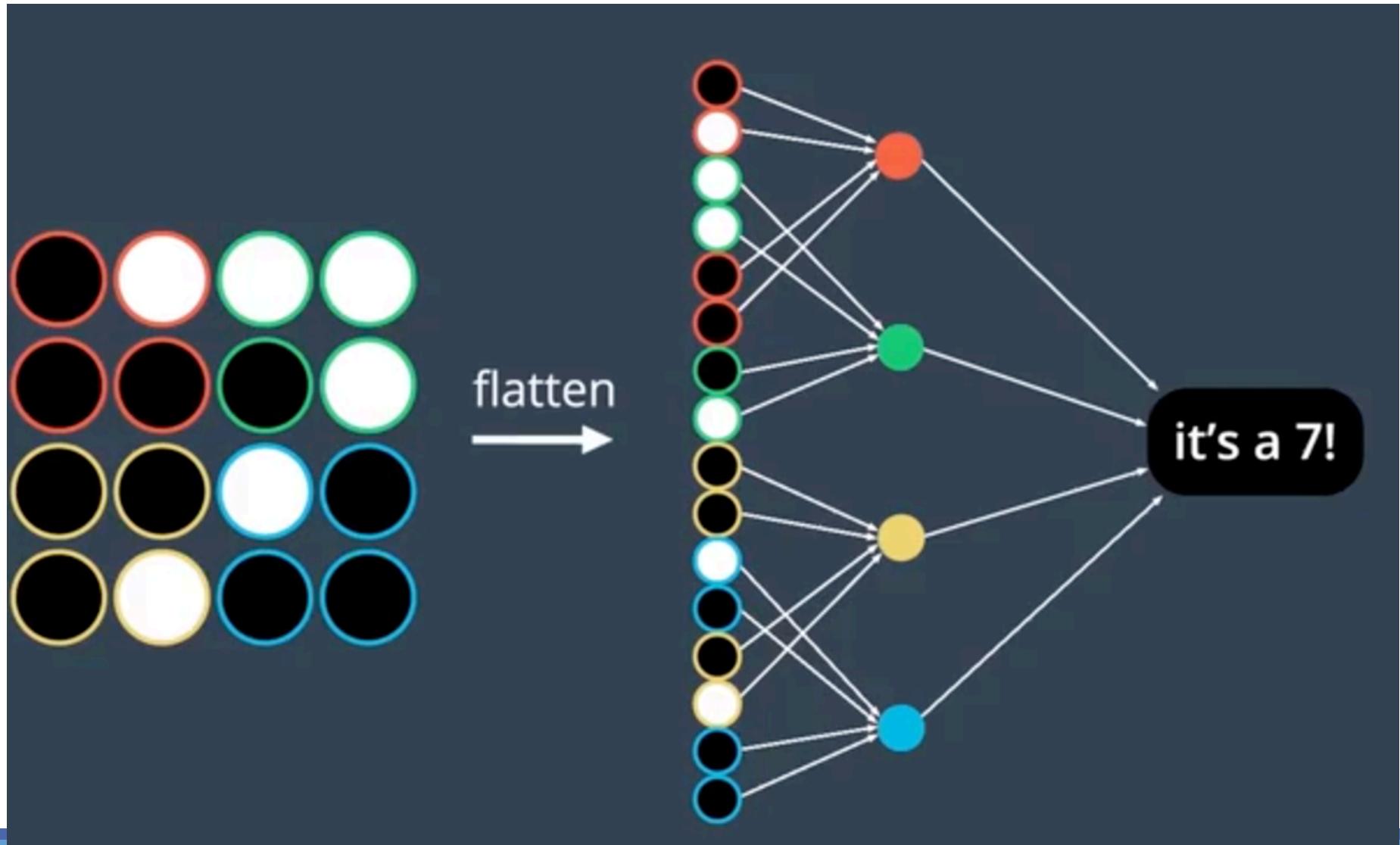
Dog vs Food



Classificação de Imagens com MLP

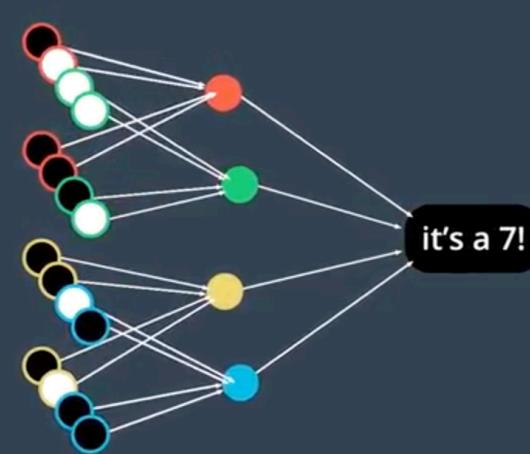
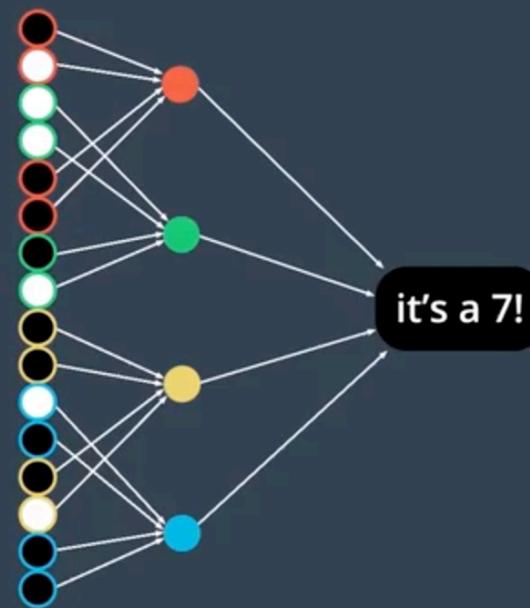


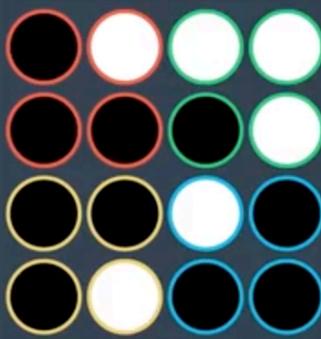
Classificação de Imagens com MLP



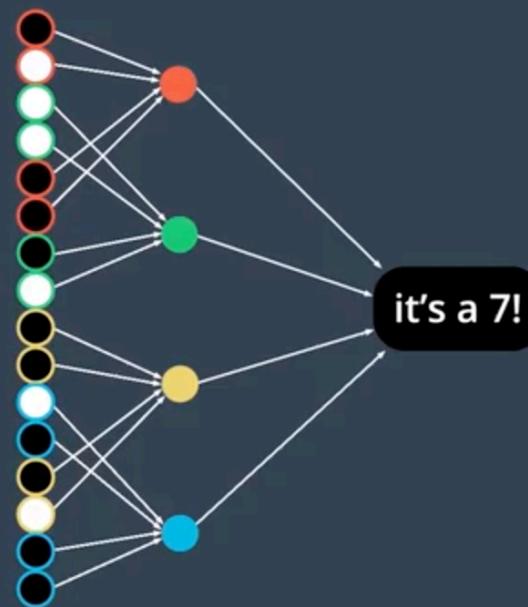


flatten
→

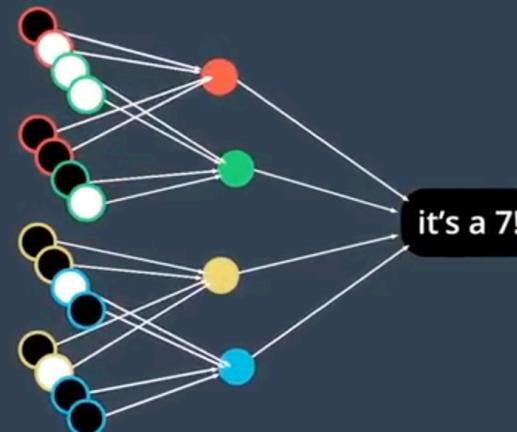




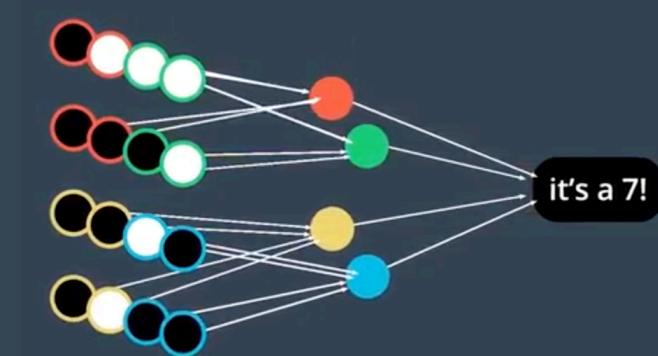
flatten
→



it's a 7!



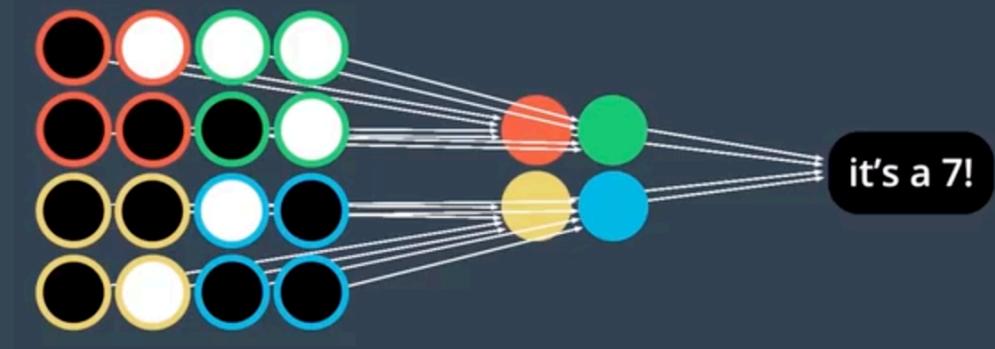
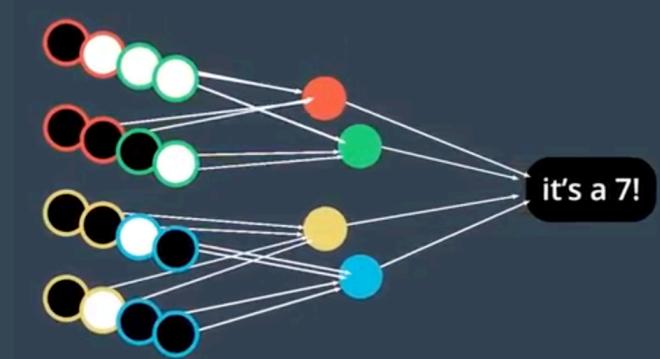
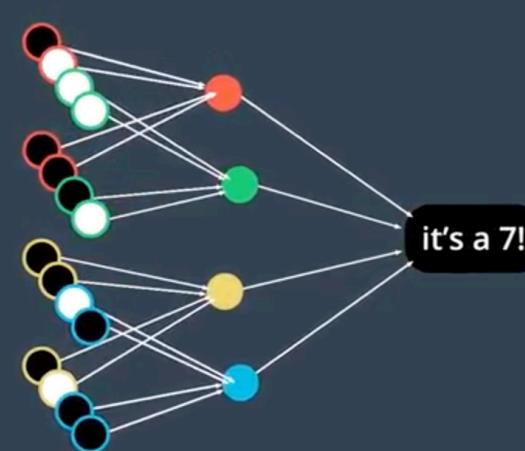
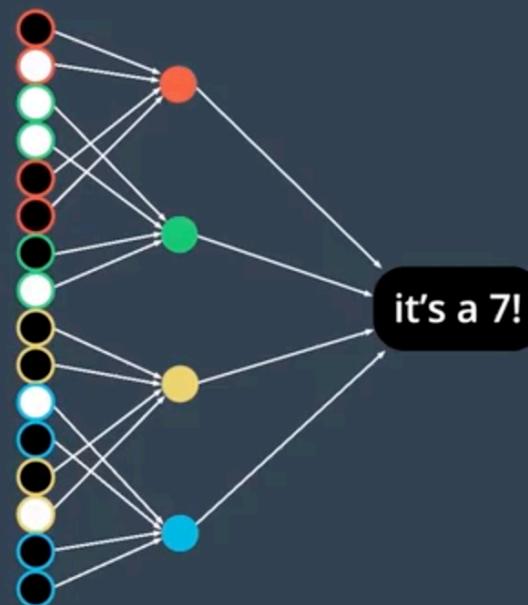
it's a 7!



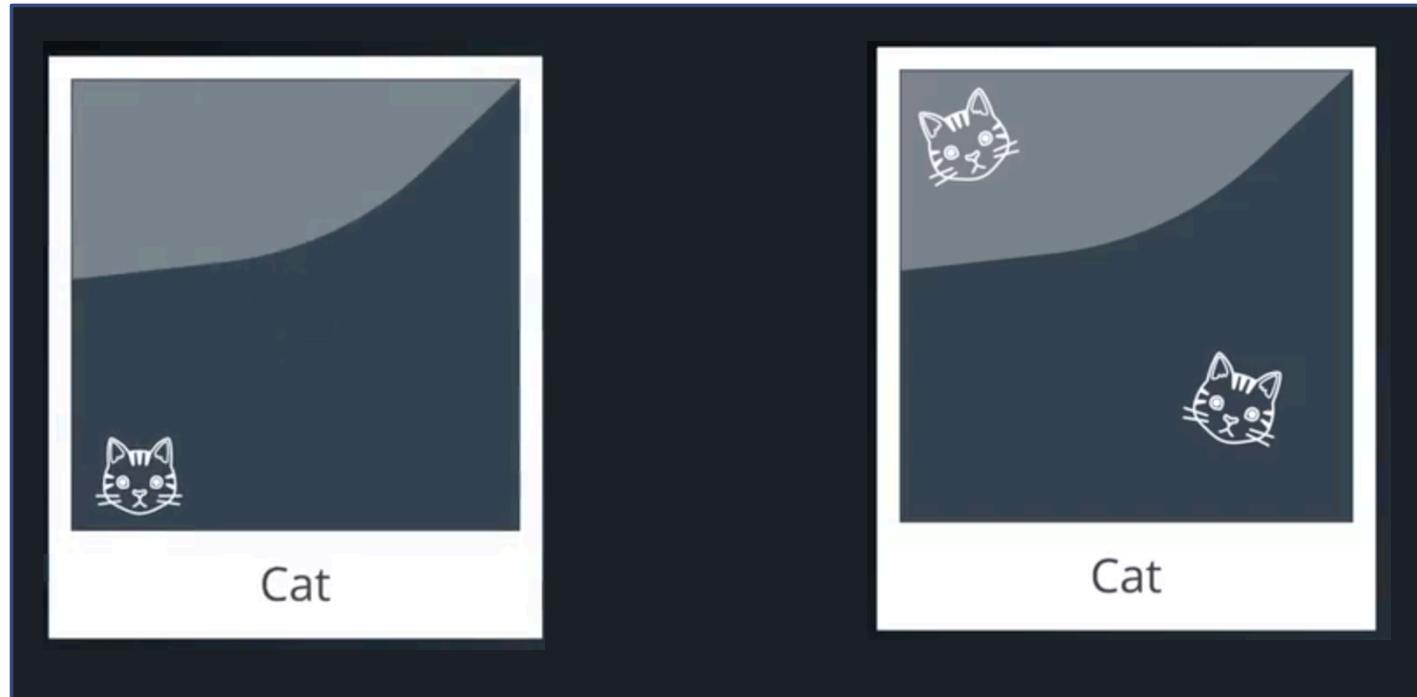
it's a 7!



flatten
→

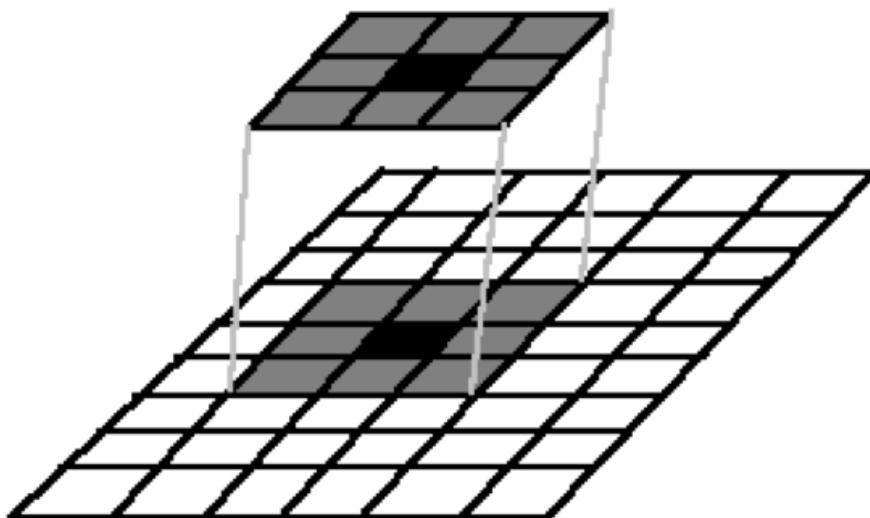


Conectividade Local



Convolução

- ▷ Definição de kernel/ máscara



131	162	232	84	91	207	
104	-1	0	+1	237	109	
243	-2	0	+2	135	126	
185	-1	0	+1	61	225	
157	124	25	14	102	108	
5	155	16	218	232	249	

FILTRAGEM NO DOMÍNIO ESPACIAL

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

			0						

Source: S. Seitz

FILTRAGEM NO DOMÍNIO ESPACIAL

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

0	10								

Source: S. Seitz

FILTRAGEM NO DOMÍNIO ESPACIAL

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20						

Source: S. Seitz

FILTRAGEM NO DOMÍNIO ESPACIAL

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30					

Source: S. Seitz

FILTRAGEM NO DOMÍNIO ESPACIAL

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$

	0	10	20	30	30				

Source: S. Seitz

FILTRAGEM NO DOMÍNIO ESPACIAL

$\frac{1}{9}$	1	1	1
1	1	1	1
1	1	1	1

$F[x, y]$

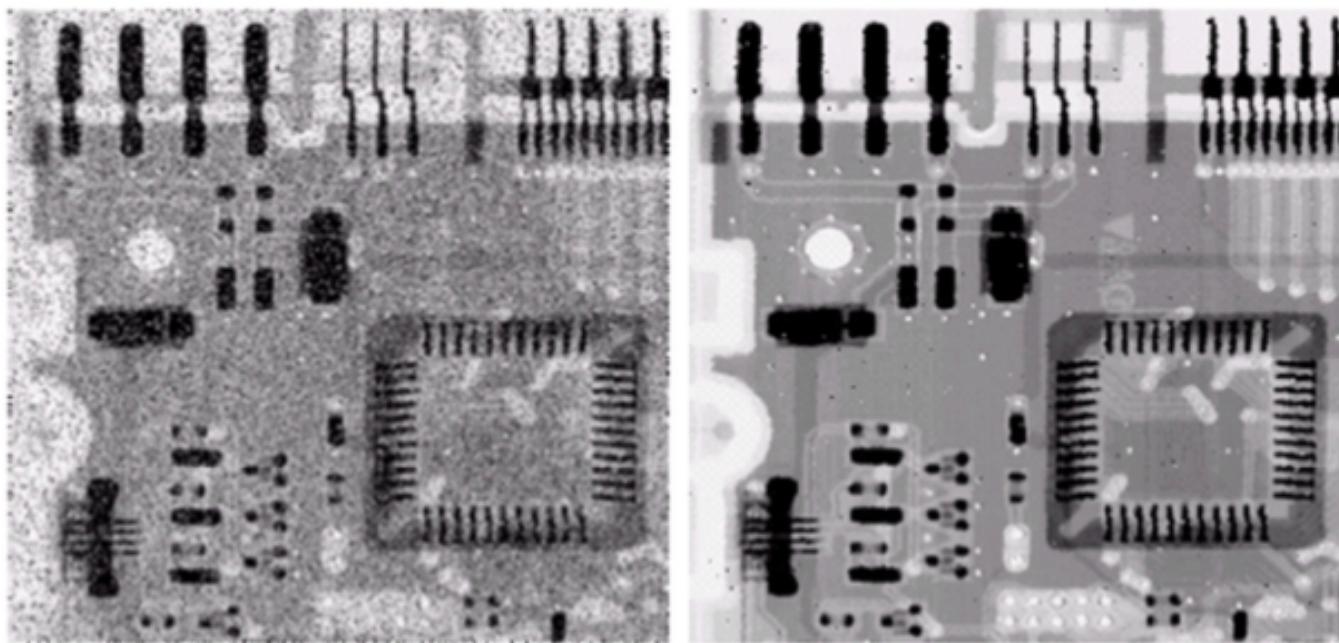
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

0	10	20	30	30	30	20	10		
0	20	40	60	60	60	40	20		
0	30	60	90	90	90	60	30		
0	30	50	80	80	90	60	30		
0	30	50	80	80	90	60	30		
0	20	30	50	50	60	40	20		
10	20	30	30	30	30	20	10		
10	10	10	0	0	0	0	0		

Source: S. Seitz

Filtros Lineares – Suavização



Filtragem Linear

▷ Filtro de Média

- Suavização

$$R = \frac{\sum z_i}{n}, i \in [1, n]$$

Exemplo: máscara 3×3

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Borramento proporcional
ao tamanho da máscara



Filtragem Linear

▷ Filtro Gaussiano

- Suavização

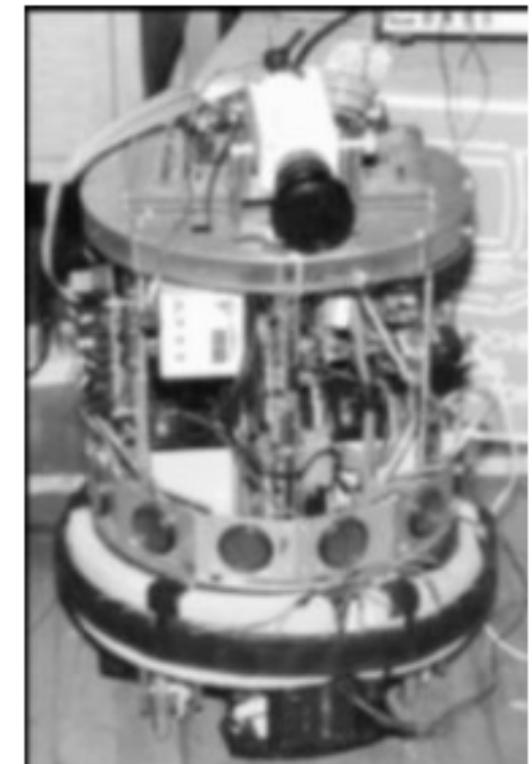
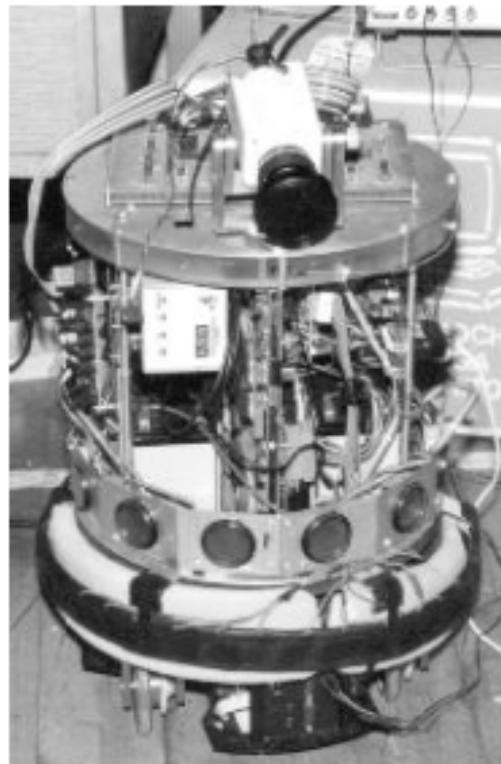
Filtro de média ponderada

Mais importância para pixels
mais próximos do centro da
máscara

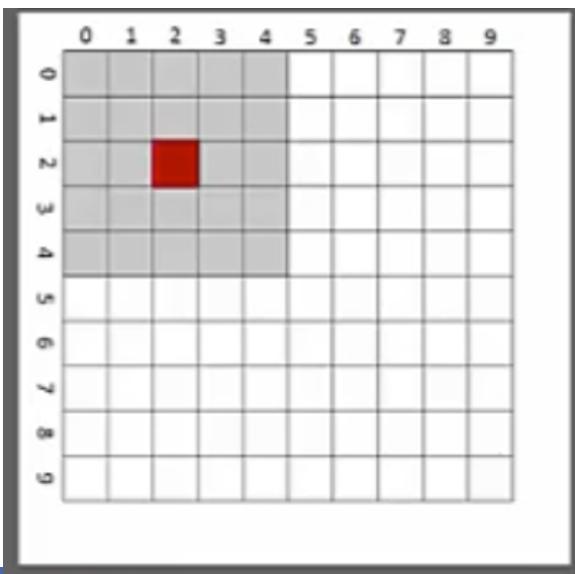
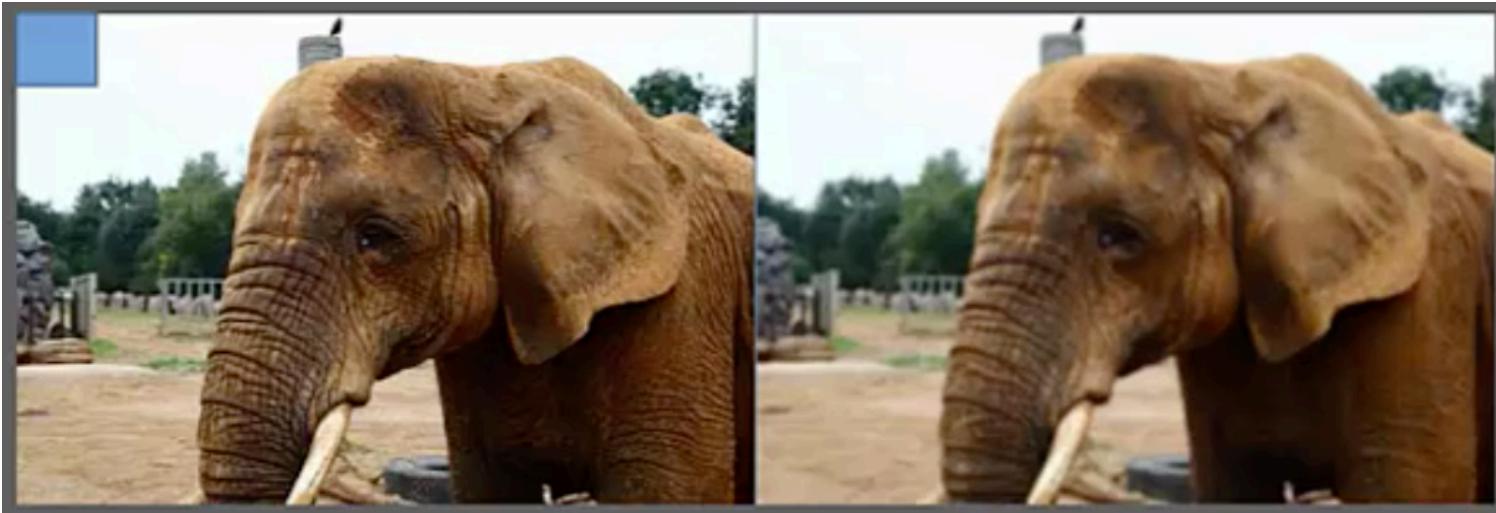
- Menos borramento

Exemplo: máscara 3×3

$$\frac{1}{100} \times \begin{array}{|c|c|c|} \hline 8 & 12 & 8 \\ \hline 12 & 20 & 12 \\ \hline 8 & 12 & 8 \\ \hline \end{array}$$



Suavização



$$Kernel_-= \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Filtro de Aguçamento

▷ O Contrário de suavização



Filtragem no OpenCV

▷ Filtros específicos

- Média

```
#media  
blur = cv2.blur(image,(3,3))  
cv2.imshow('Averaging', blur)
```

- Gaussiano

```
#Kernel Gaussiano  
Gaussian = cv2.GaussianBlur(image,(7,7),0)  
cv2.imshow("Gaussian Blurring", Gaussian)
```

- Mediana

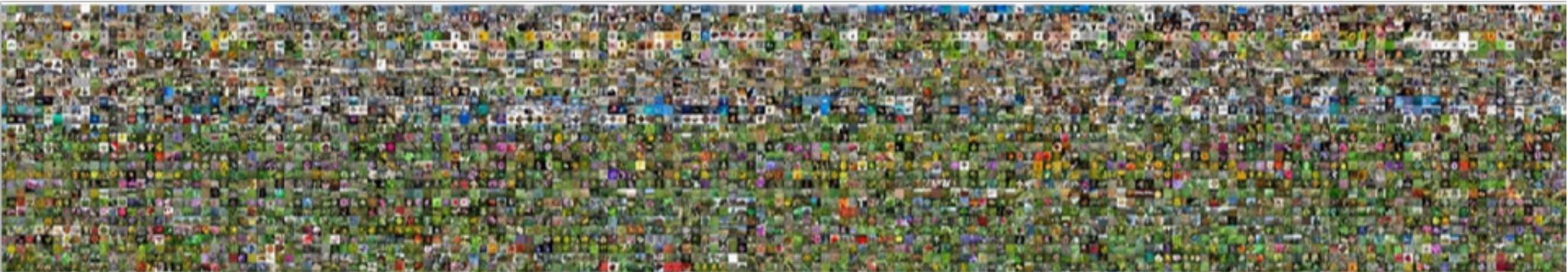
```
#Mediana  
median = cv2.medianBlur(image,5)  
cv2.imshow("Median Blurring", median)
```

- bilateral

```
#Bilateral – remove ruído e mantém contraste  
bilateral = cv2.bilateralFilter(image,9,75,75)  
cv2.imshow('Bilateral', bilateral)
```

Exercício Filtro

- ▷ Implementar um filtro de média (kernel 5x5)
 - filtro_media.ipynb



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



Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images



Output:

Scale
T-shirt
Steel drum
Drumstick
Mud turtle



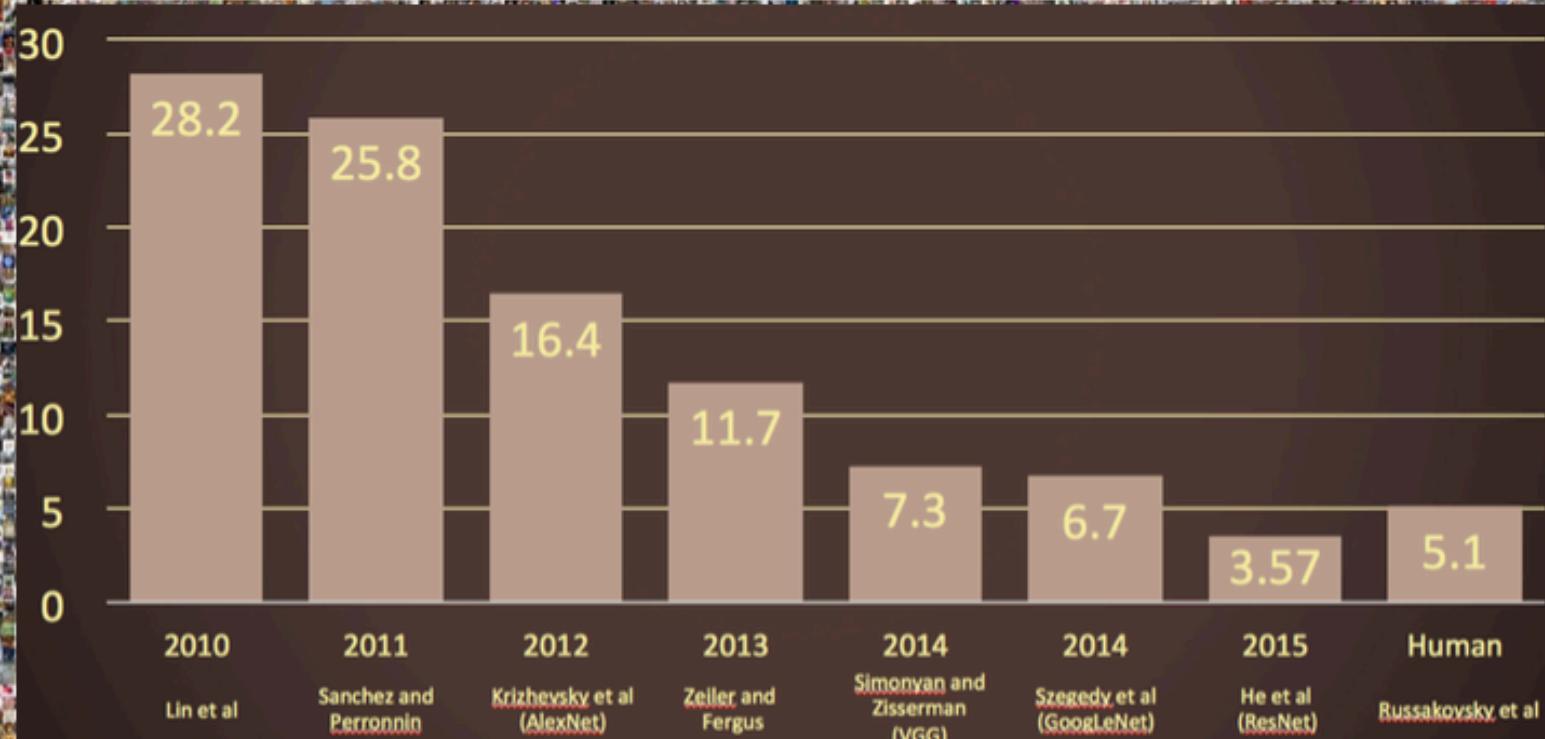
Output:

Scale
T-shirt
Giant panda
Drumstick
Mud turtle



IMAGENET Large Scale Visual Recognition Challenge

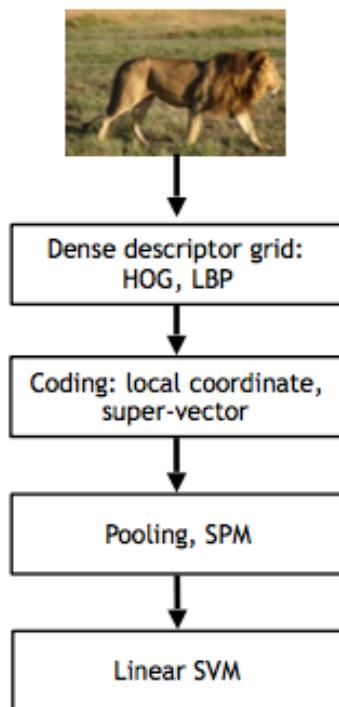
The Image Classification Challenge:
1,000 object classes
1,431,167 images



IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

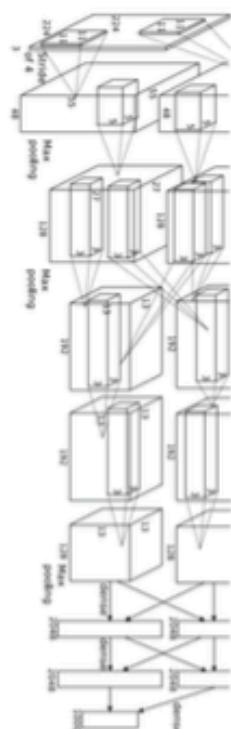


[Lin CVPR 2011]

Lion image by Swissfrog is licensed under CC BY 3.0

Year 2012

SuperVision



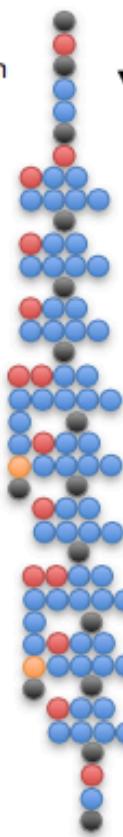
[Krizhevsky NIPS 2012]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Year 2014

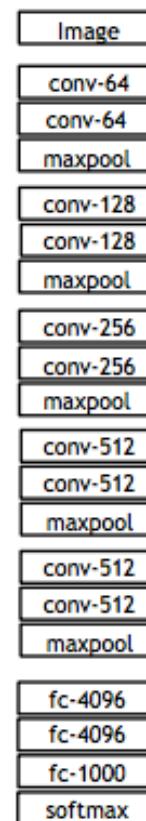
GoogLeNet

- Pooling
- Convolution
- Softmax
- Other



[Szegedy arxiv 2014]

VGG



[Simonyan arxiv 2014]

Year 2015

MSRA

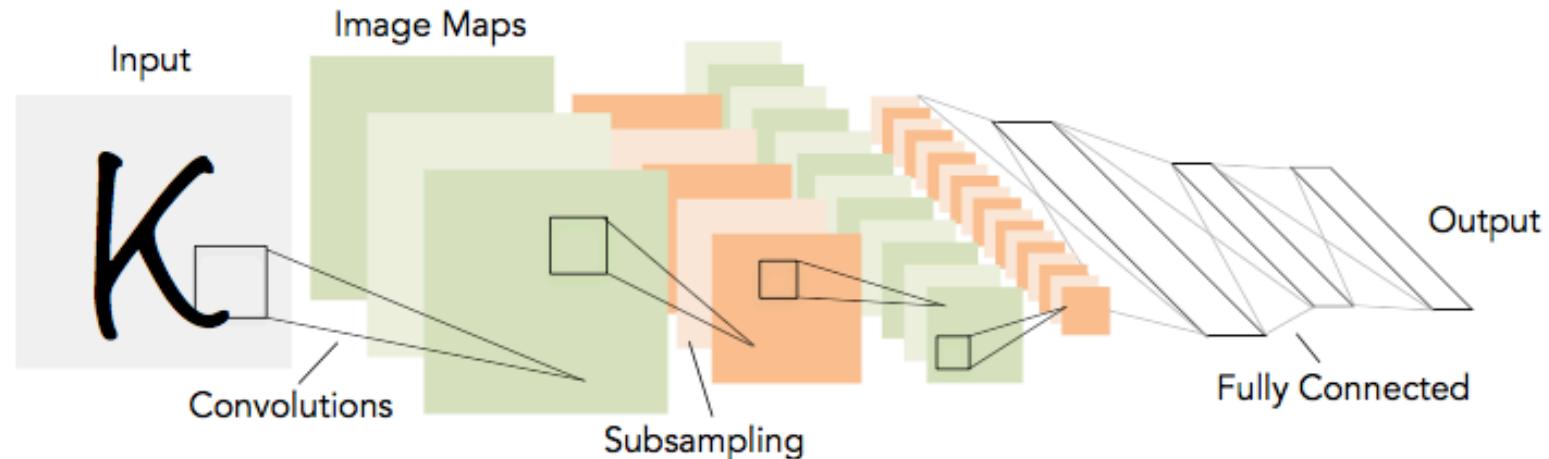


[He ICCV 2015]

Redes Neurais Convolutivas (CNN) não foram inventadas do dia para a noite

1998

LeCun et al.



of transistors



10^6

pentium® II

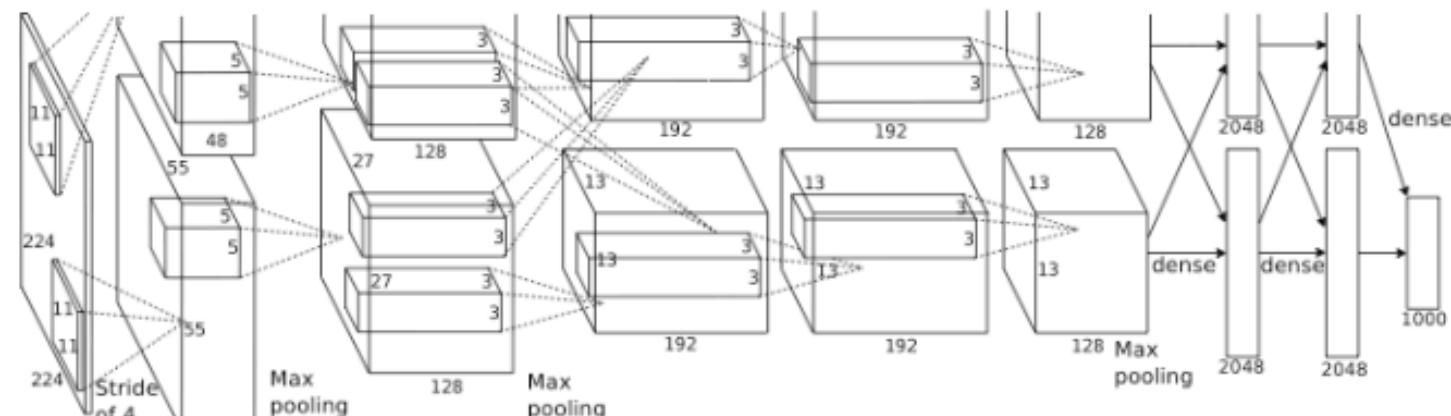
of pixels used in training

10^7



2012

Krizhevsky et al.



of transistors



10^9

GPUs



of pixels used in training

10^{14}

IMAGENET

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012.
Reproduced with permission.

ConvNets estão em todo lugar

Classification



Retrieval



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

ConvNets estão em todo lugar

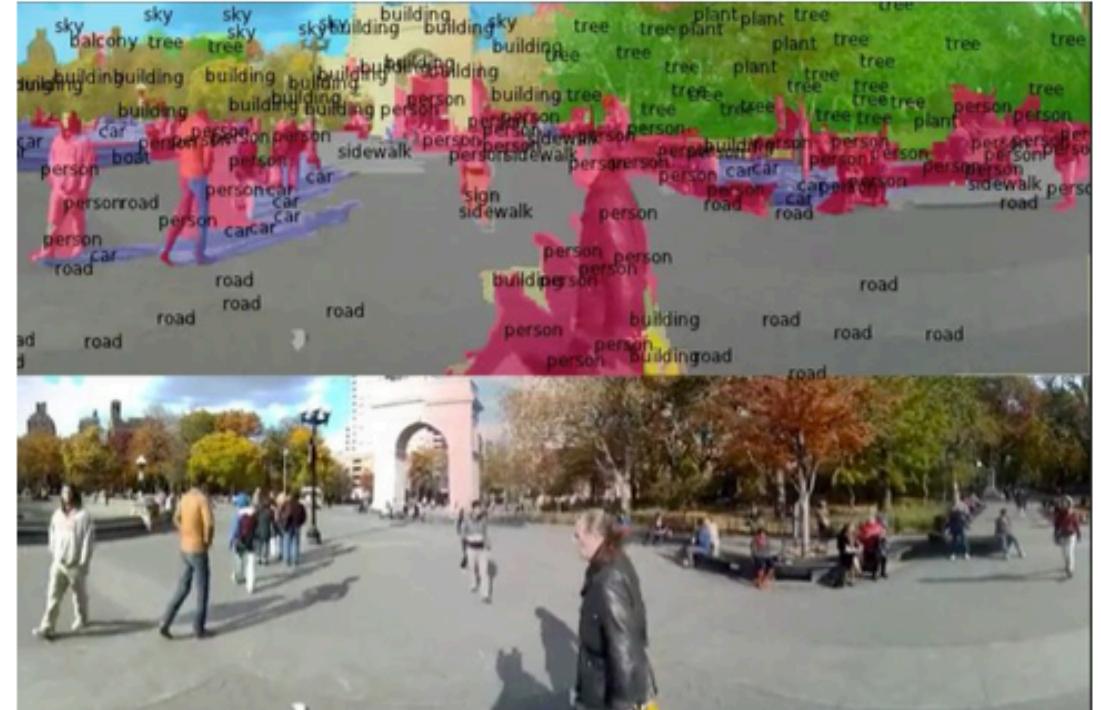
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

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

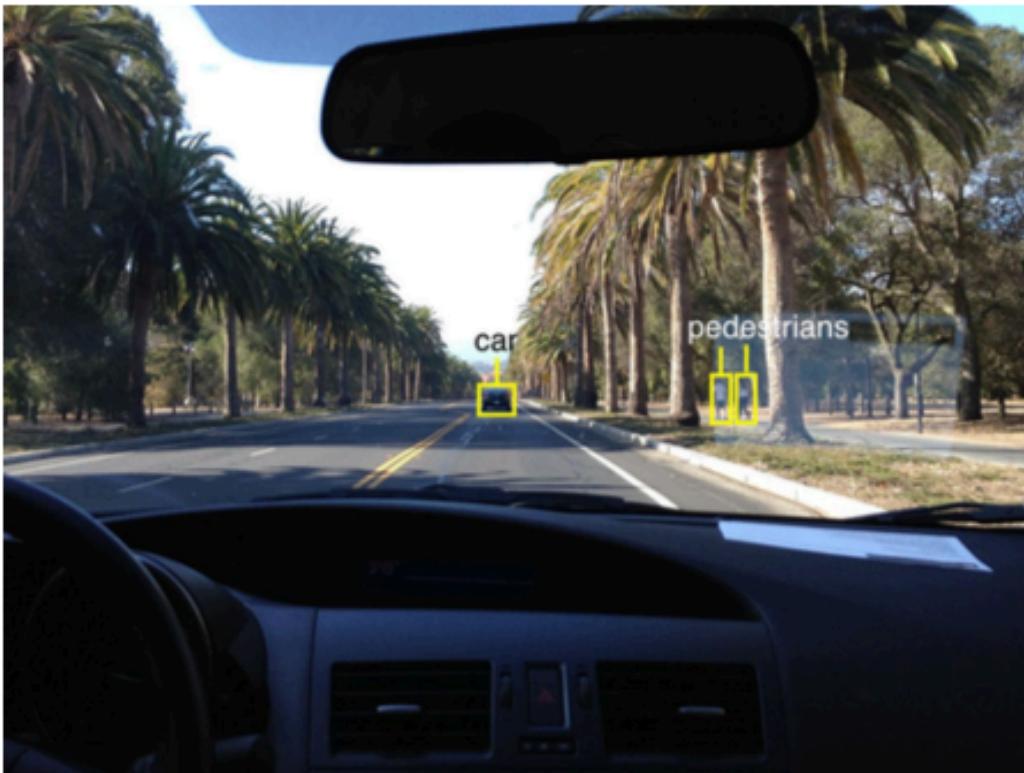
Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

ConvNets estão em todo lugar



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



This image by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

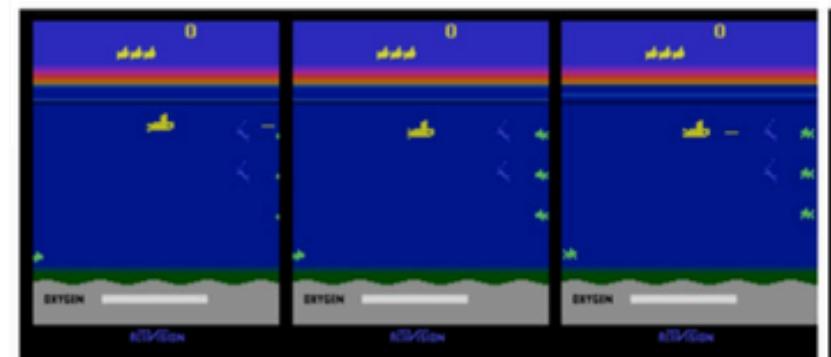
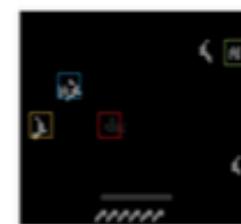
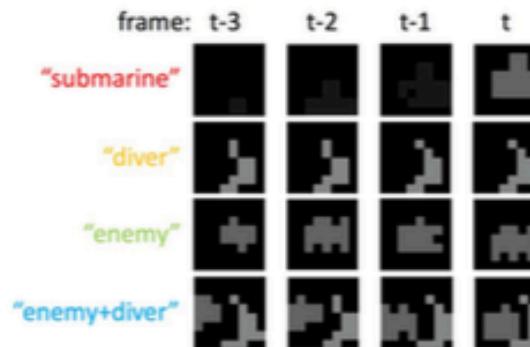
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

ConvNets estão em todo lugar



Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

ConvNets estão em todo lugar

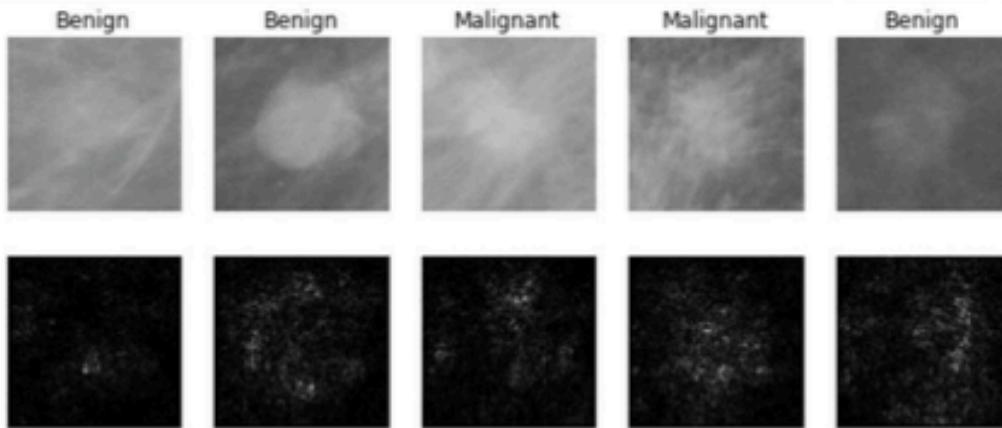


Figure copyright Levy et al. 2016.
Reproduced with permission.



[Dieleman et al. 2014]

From left to right: [public domain by NASA](#), usage permitted by
ESA/Hubble, [public domain by NASA](#), and [public domain](#).



[Sermanet et al. 2011]
[Ciresan et al.]

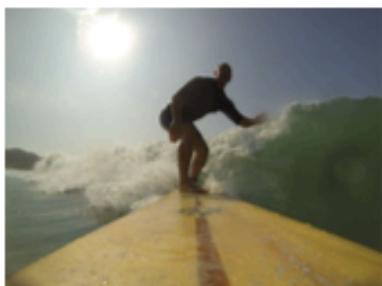
Photos by Lane McIntosh.
Copyright CS231n 2017.

Descrição de Imagens

No errors

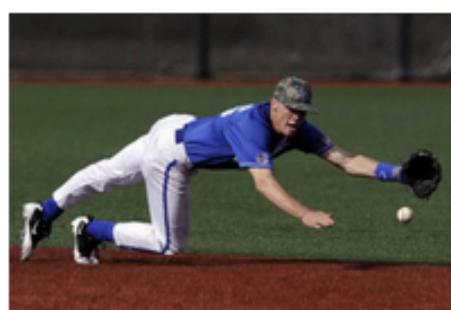


A white teddy bear sitting in the grass



A man riding a wave on top of a surfboard

Minor errors



A man in a baseball uniform throwing a ball



A cat sitting on a suitcase on the floor

Somewhat related



A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

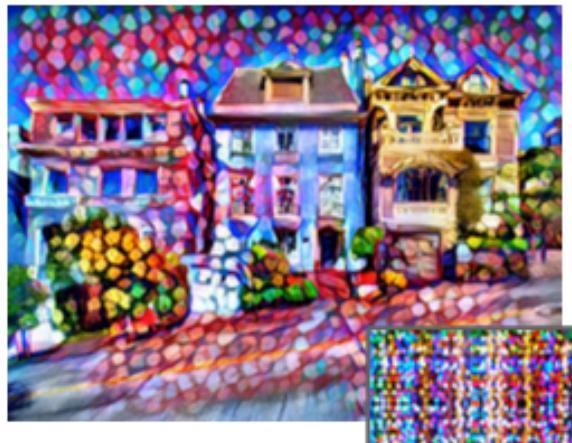
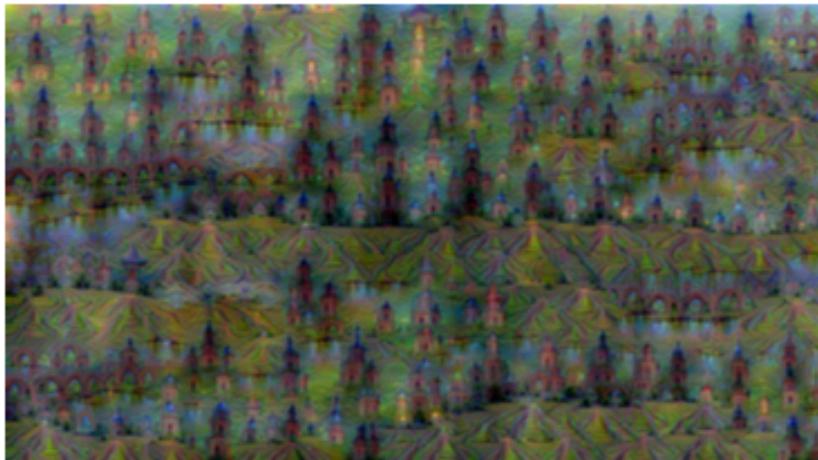
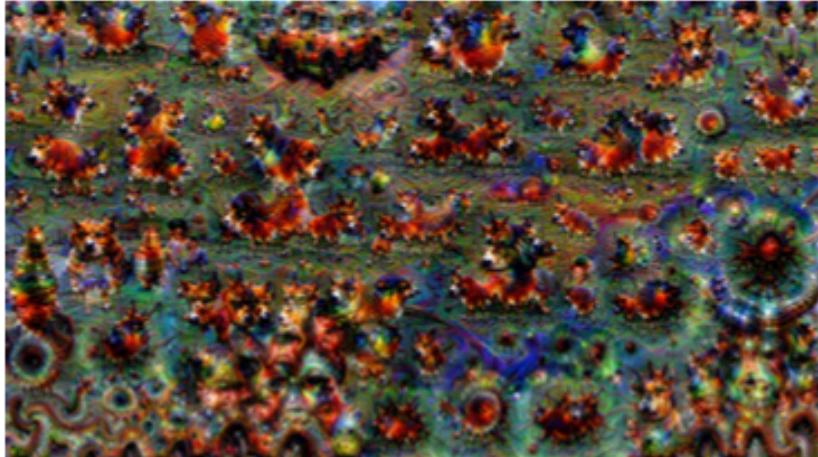
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
<https://pixabay.com/en/luggage-antique-cat-1643010/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-literal-1668716/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983987/>
<https://pixabay.com/en/handstand-lake-meditation-4916008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)

Transferência de Estilo

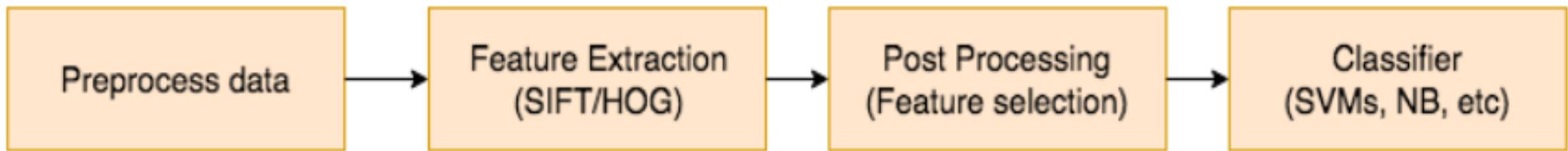


Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.

Original image is CC0 public domain
Starry Night and *Tree Roots* by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized Images copyright Justin Johnson, 2017;
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Era Pré Deep Learning



▷ Cons:

- Características Manuais difíceis de se construir
- Processo que demanda tempo
- Quais conjuntos de características maximizam acurácia?
- Tende a causar overfitting.

Redes Neurais Convolutivas



Fluxograma Deep Learning



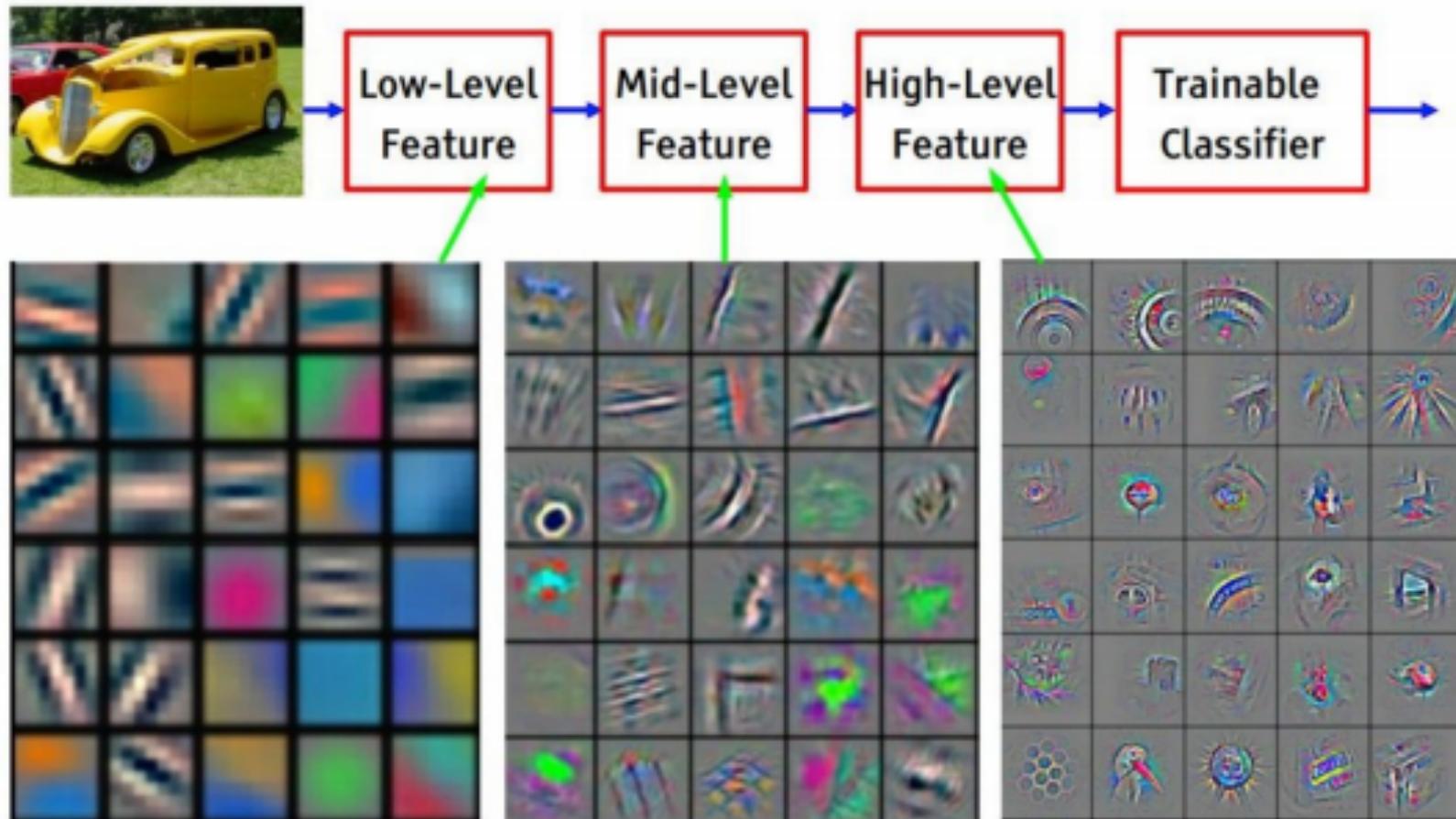
Traditional Machine Learning Flow



Deep Learning Flow

O que é Deep Learning

- ▷ Composição de transformações não lineares dos dados



Um pouco de história..

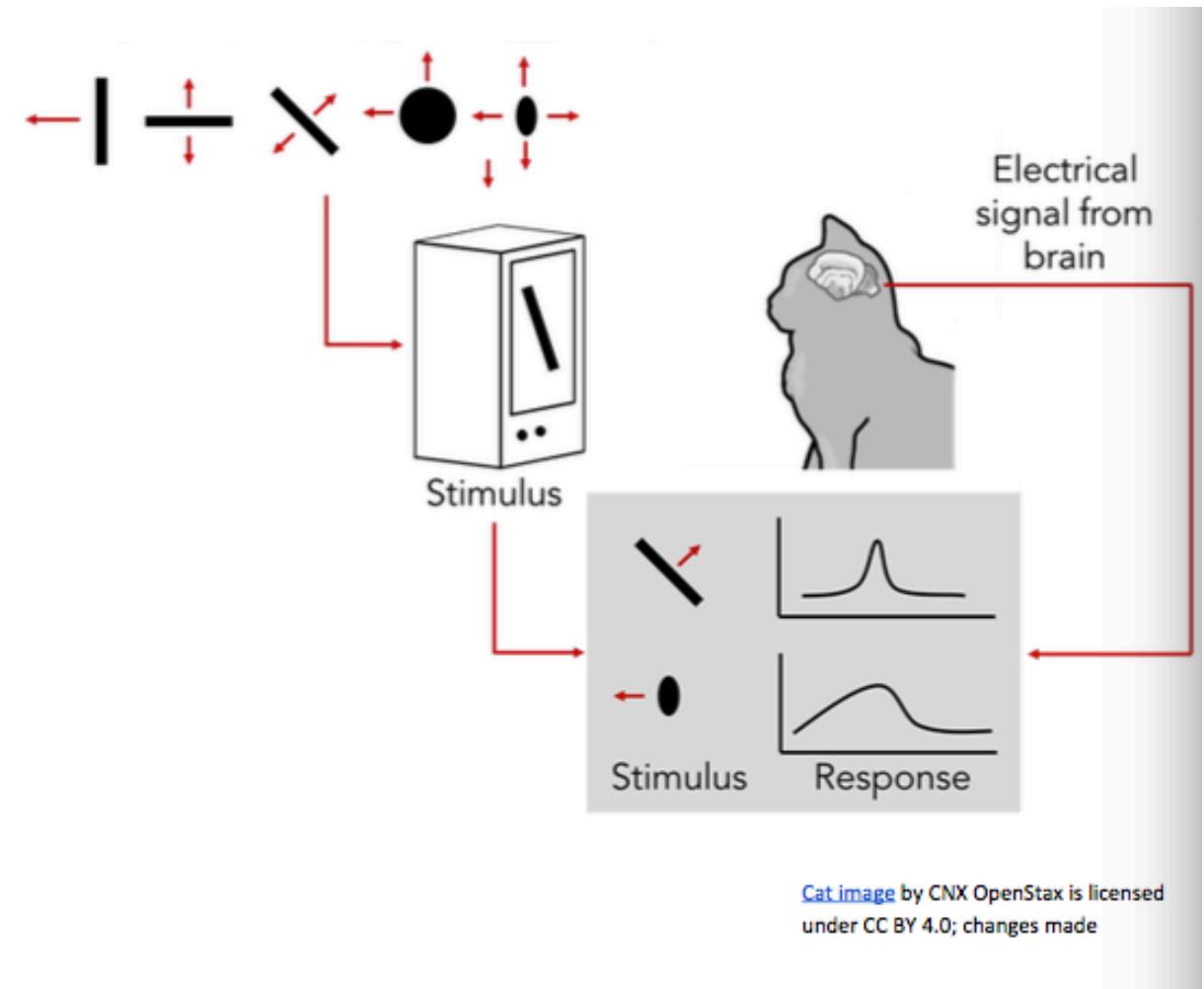
**Hubel & Wiesel,
1959**

RECEPTIVE FIELDS OF SINGLE
NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR
INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...



Cat image by CNX OpenStax is licensed under CC BY 4.0; changes made

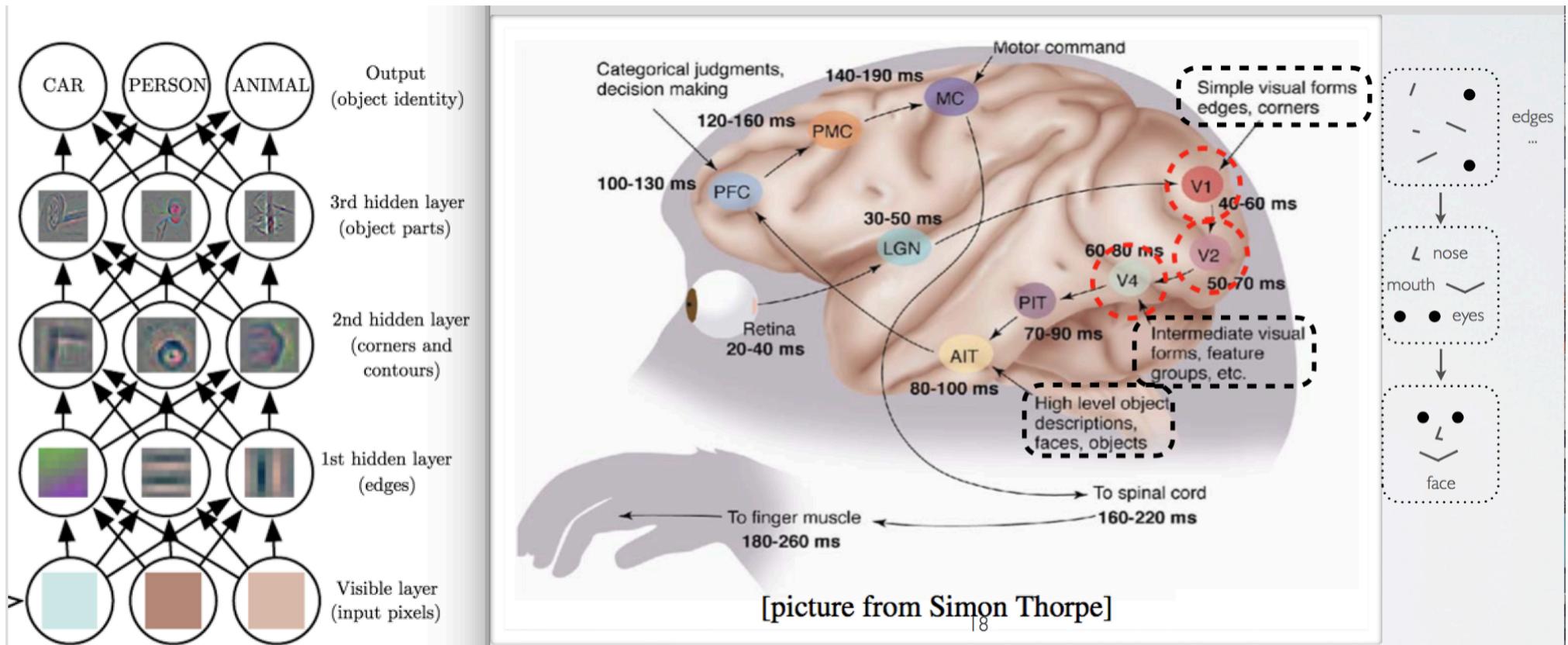
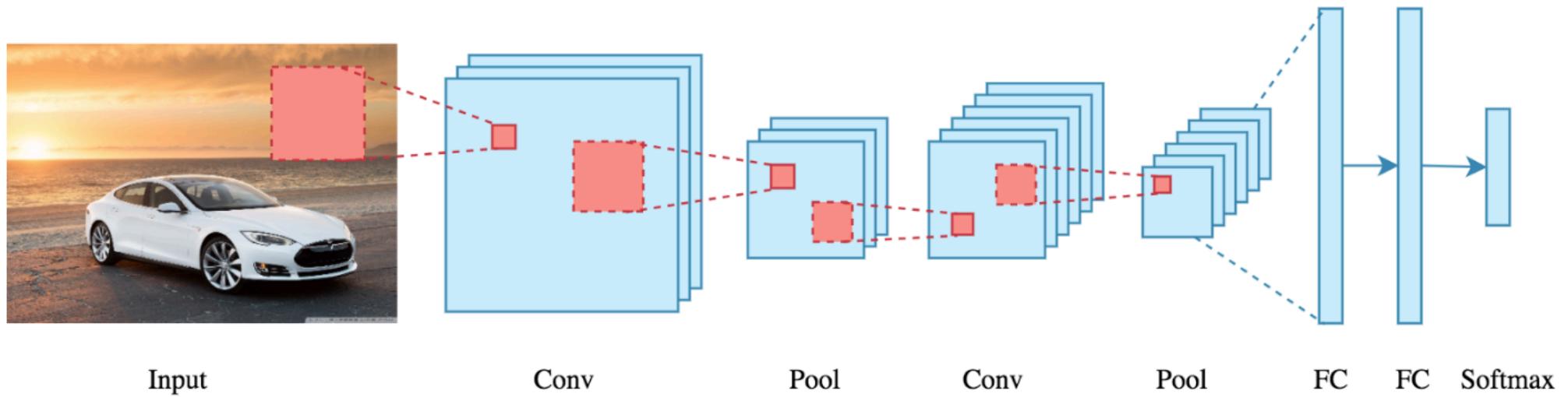


Fig 1. Image Courtesy: Hugo Larochelle Slides.

CNN

► Modelos CNN possuem arquitetura similar

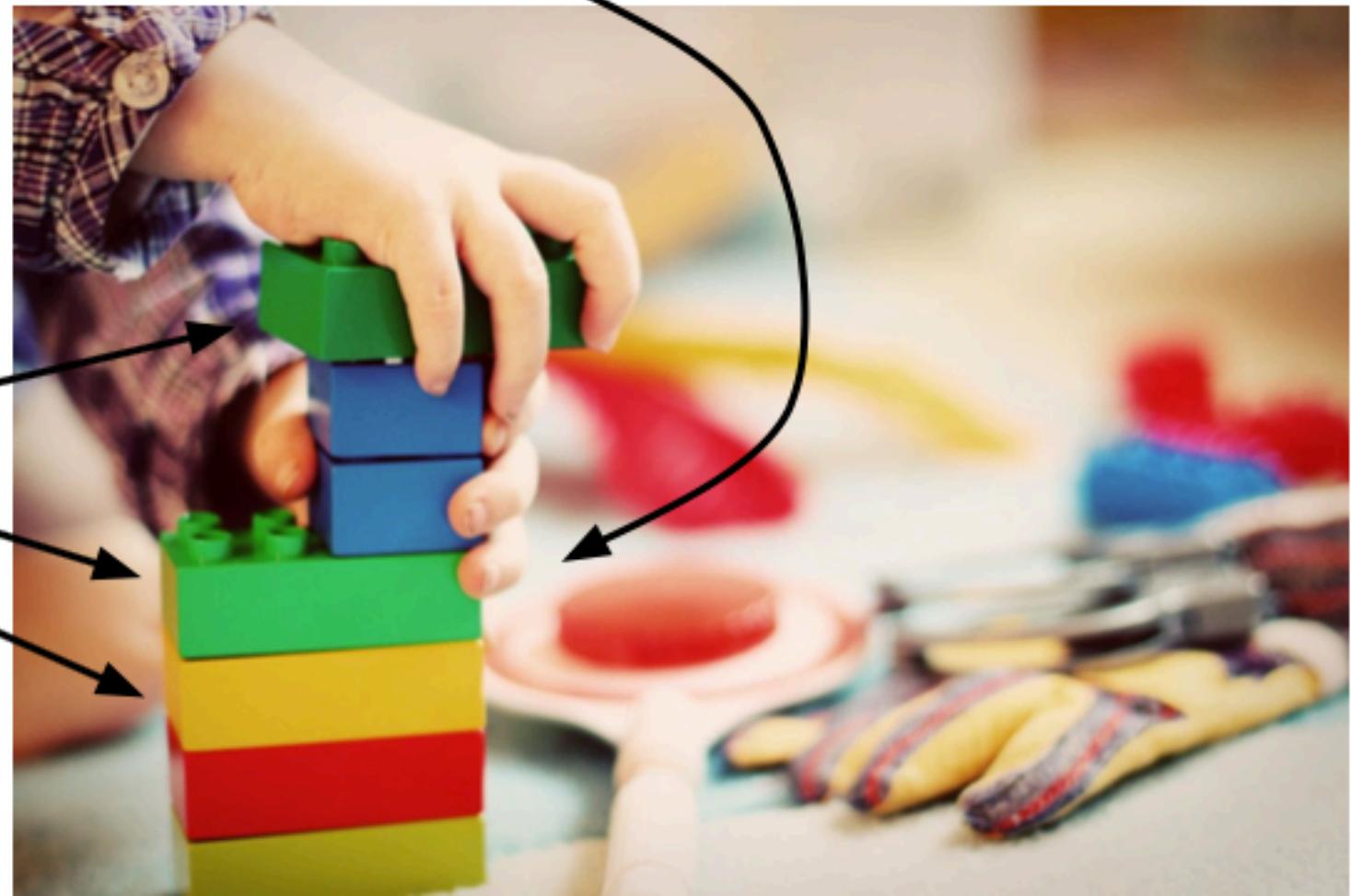


Tipos de Camadas

- ▷ Convolutiva (CONV)
- ▷ Pooling (POOL)
- ▷ Fully-conected (FC)
- ▷ Batch Normalization (BN)
- ▷ Dropout(DO)
- ▷ Softmax

Neural Network

Linear
classifiers

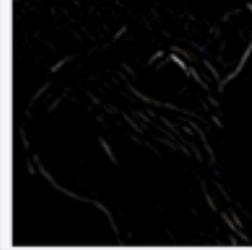
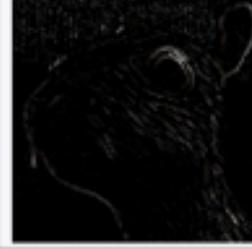
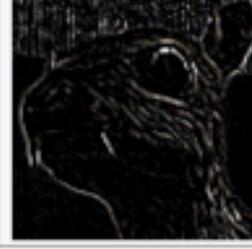


[This image is CC0 1.0 public domain](#)

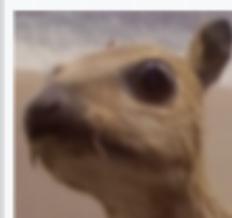
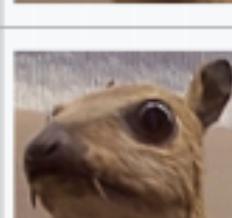
Camada Convolutiva



Convoluçãoções

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Convoluçãoções

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Convolução

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

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

Convolução

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4		

Feature Map

Convolução

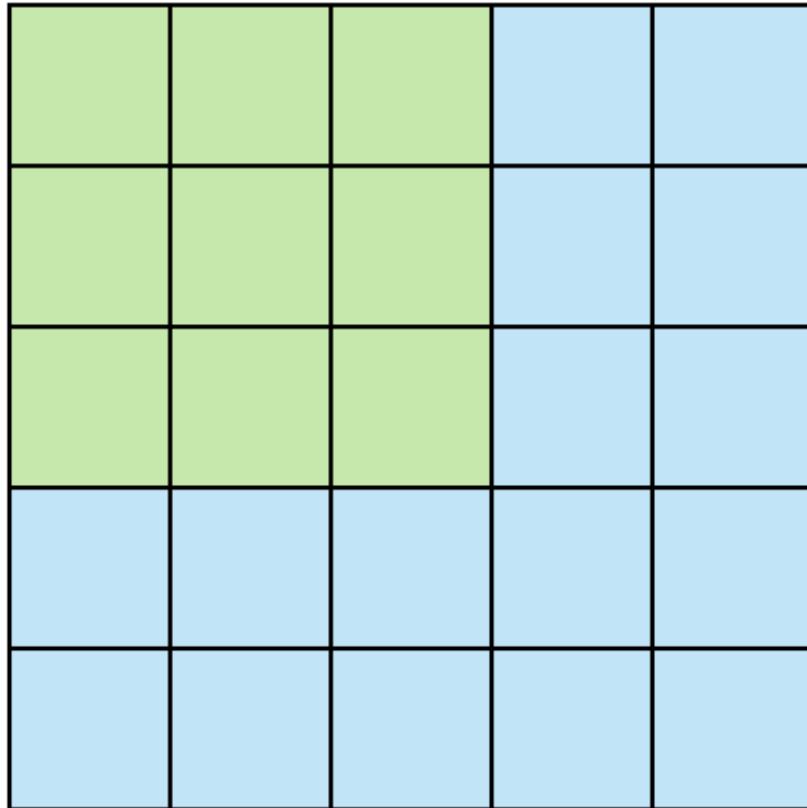
1	1x1	1x0	0x1	0
0	1x0	1x1	1x0	0
0	0x1	1x0	1x1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

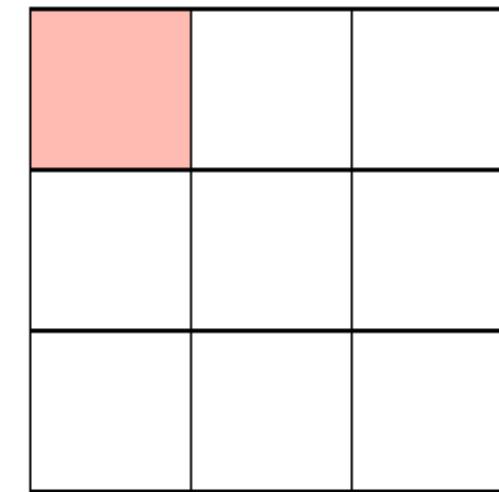
4	3	

Feature Map

Stride

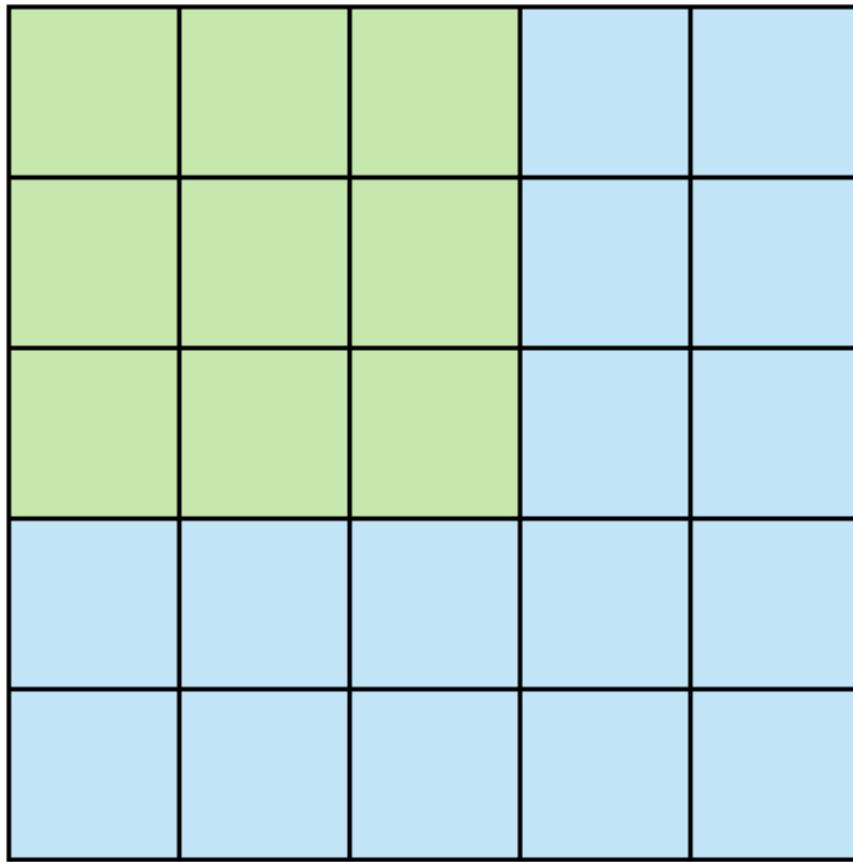


Stride 1

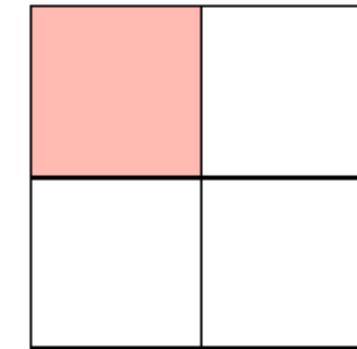


Feature Map

Stride

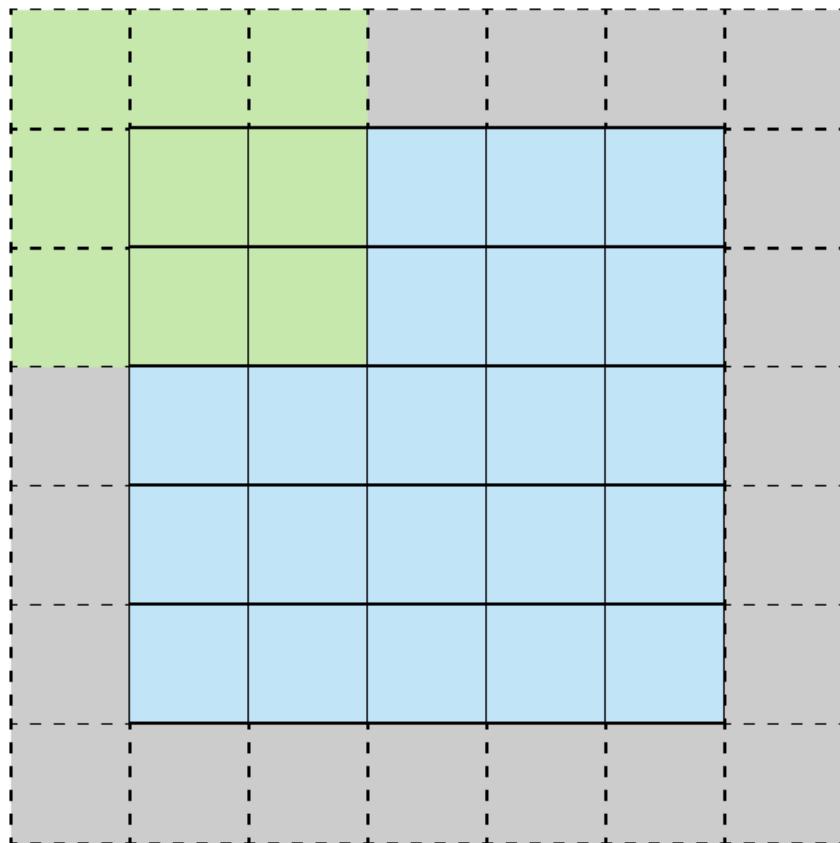


Stride 2

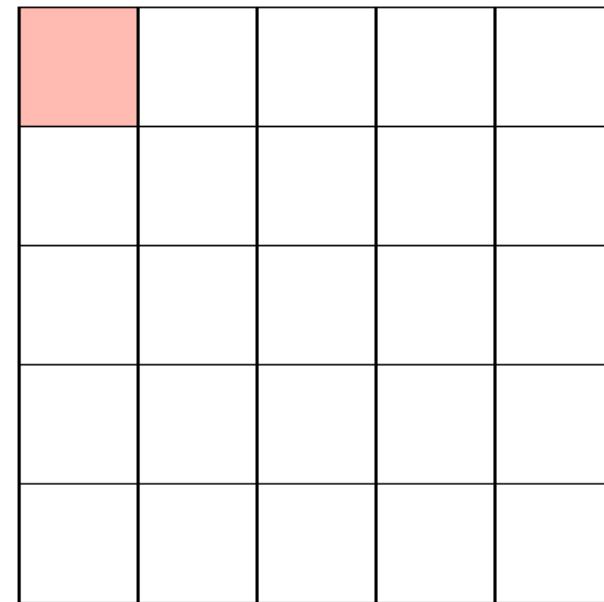


Feature Map

Padding

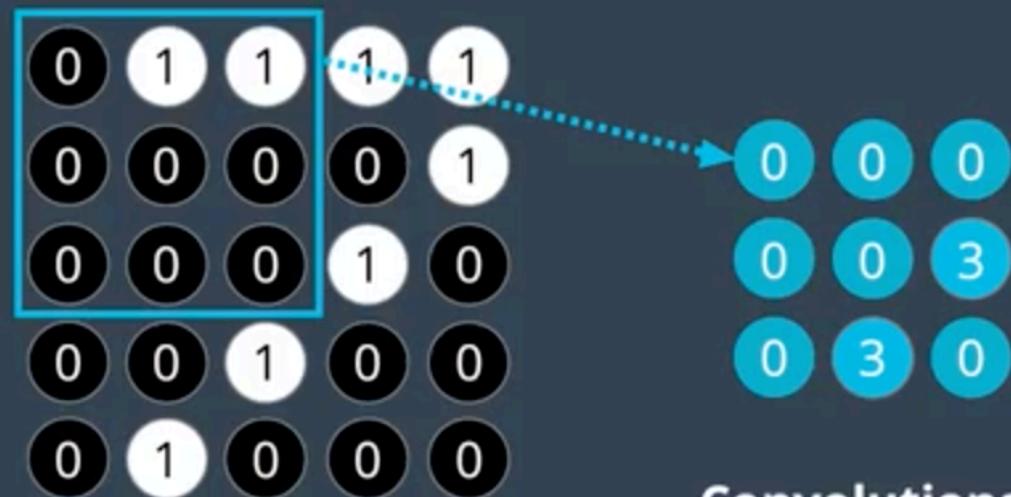


Stride 1 with Padding



Feature Map

Filter		
-1	-1	+1
-1	+1	-1
+1	-1	-1

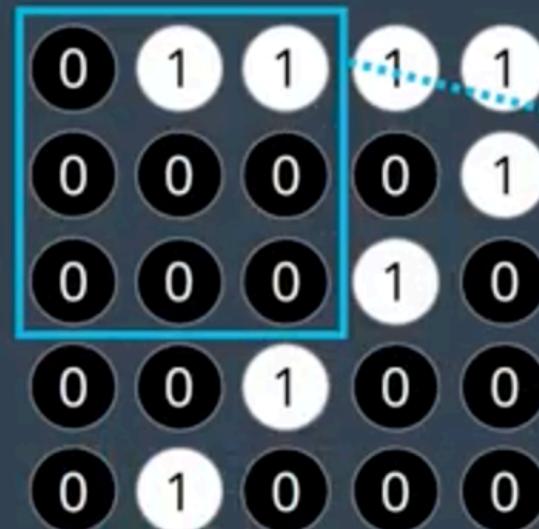


Input Layer

Convolutional Layer

$$\text{ReLU} \left(\text{SUM} \left(\begin{array}{ccc} * & * & * \\ * & * & * \\ * & * & * \end{array} \right) \right) = 0$$

Filter		
-1	-1	+1
-1	+1	-1
+1	-1	-1



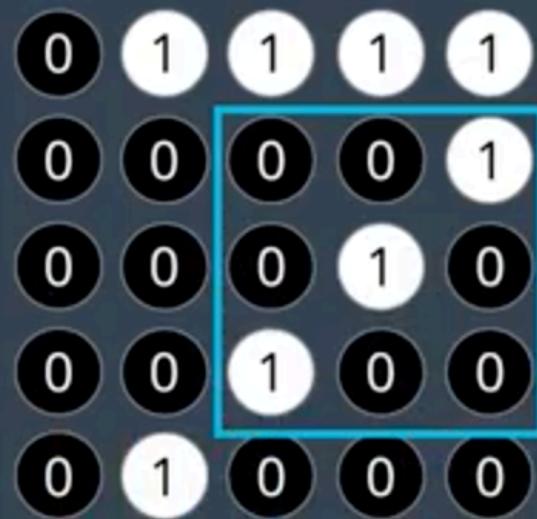
0	0	0
0	0	3
0	3	0

Convolutional Layer

$$\text{ReLU} \left(\text{SUM} \left(\begin{array}{ccc} -1 * 0 & -1 * 1 & +1 * 1 \\ -1 * 0 & +1 * 0 & -1 * 0 \\ +1 * 0 & -1 * 0 & -1 * 0 \end{array} \right) \right) = 0$$

Filter

$$\begin{pmatrix} -1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -1 \end{pmatrix}$$



Convolutional Layer

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 3 & 0 \end{pmatrix}$$

ReLU SUM $\left(\begin{array}{ccc} -1 * 0 & -1 * 0 & +1 * 1 \\ -1 * 0 & +1 * 1 & -1 * 0 \\ +1 * 1 & -1 * 0 & -1 * 0 \end{array} \right) = 3$

Filter		
-1	-1	+1
-1	+1	-1
+1	-1	-1



0	0	0
0	0	3
0	3	0

Convolutional Layer

$$\text{ReLU} \left(\text{SUM} \left(\begin{array}{ccc} -1 * 0 & -1 * 0 & +1 * 1 \\ -1 * 0 & +1 * 1 & -1 * 0 \\ +1 * 1 & -1 * 0 & -1 * 0 \end{array} \right) \right) = 3$$

Filter

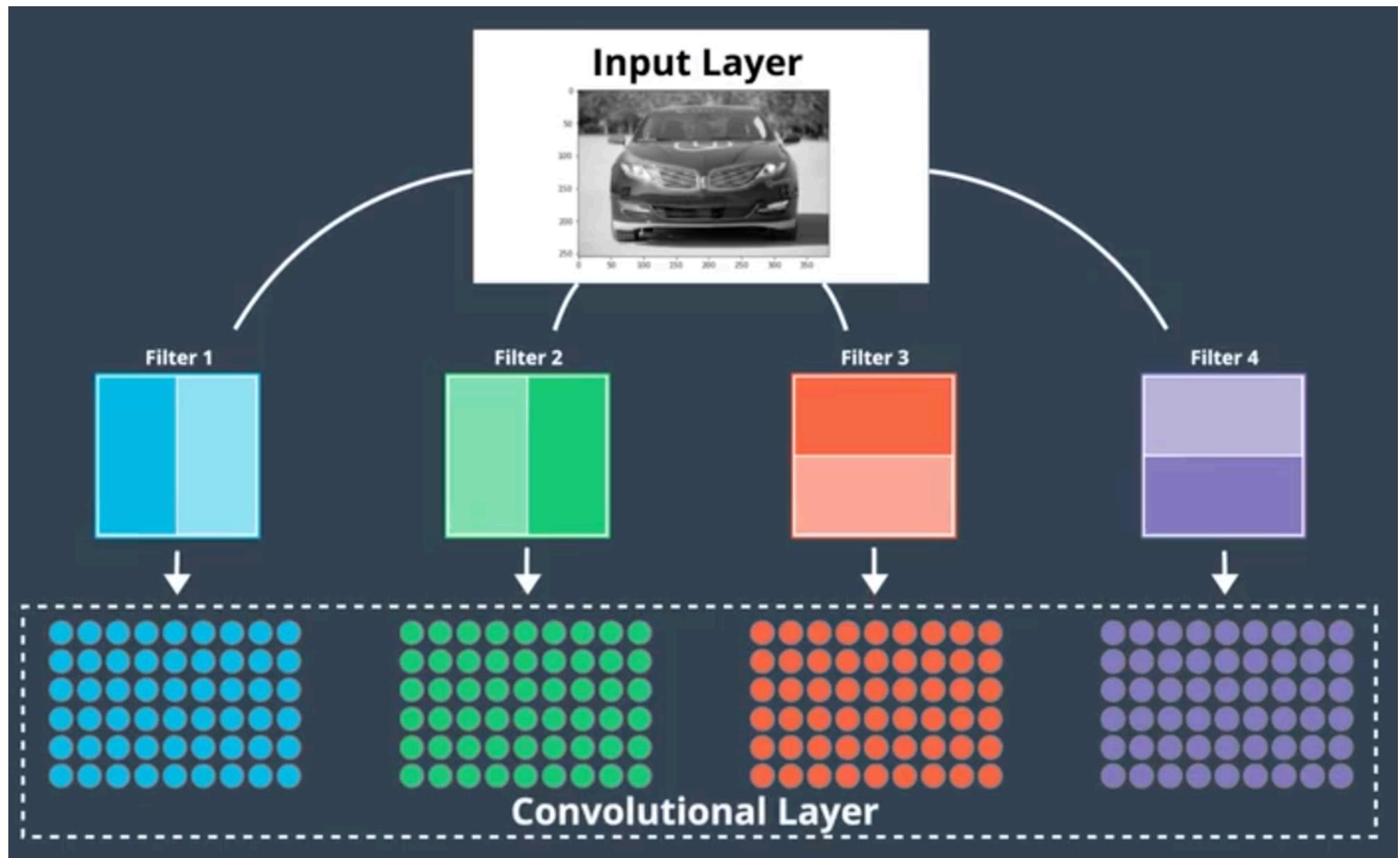
$$\begin{bmatrix} -1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -1 \end{bmatrix}$$

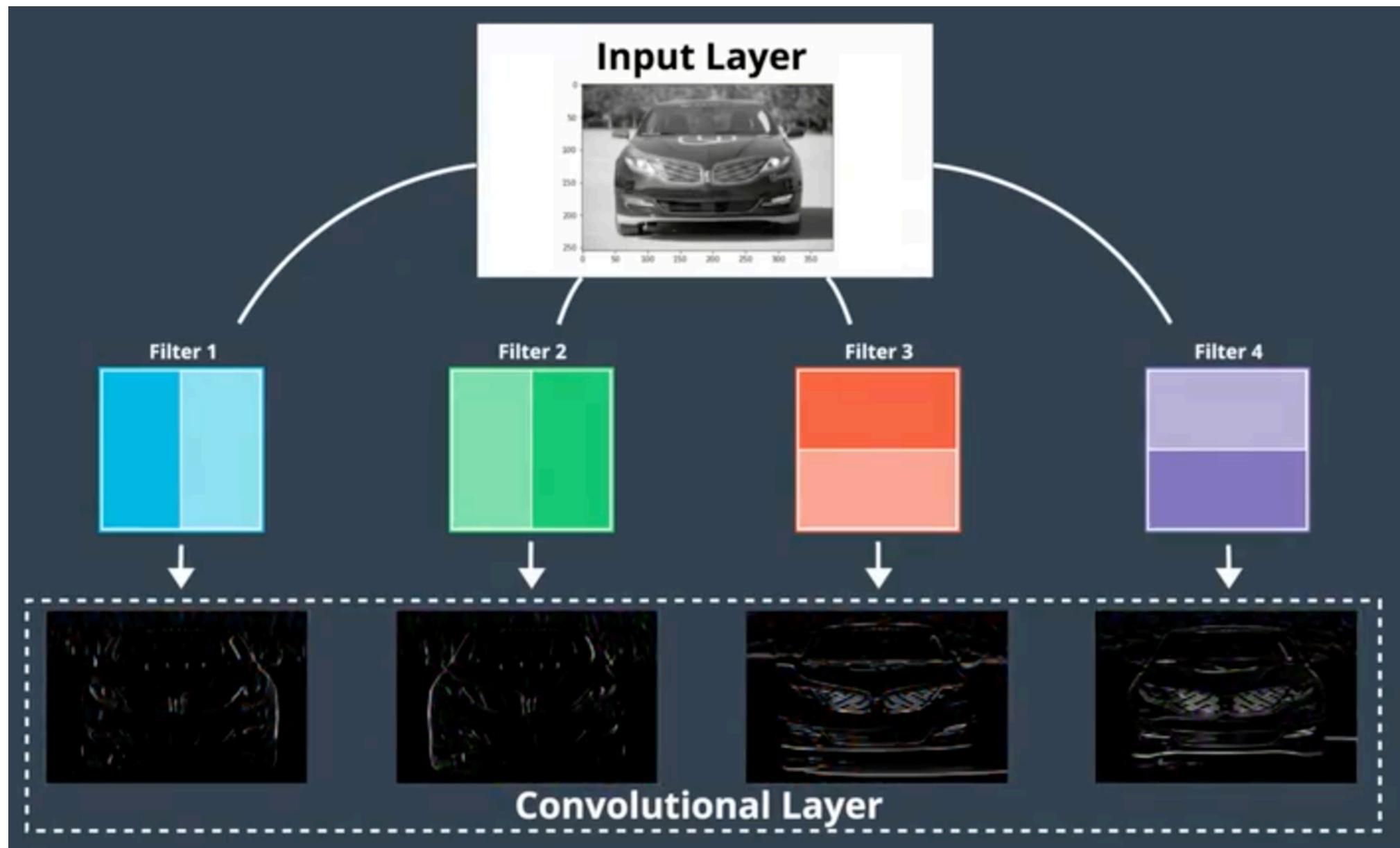


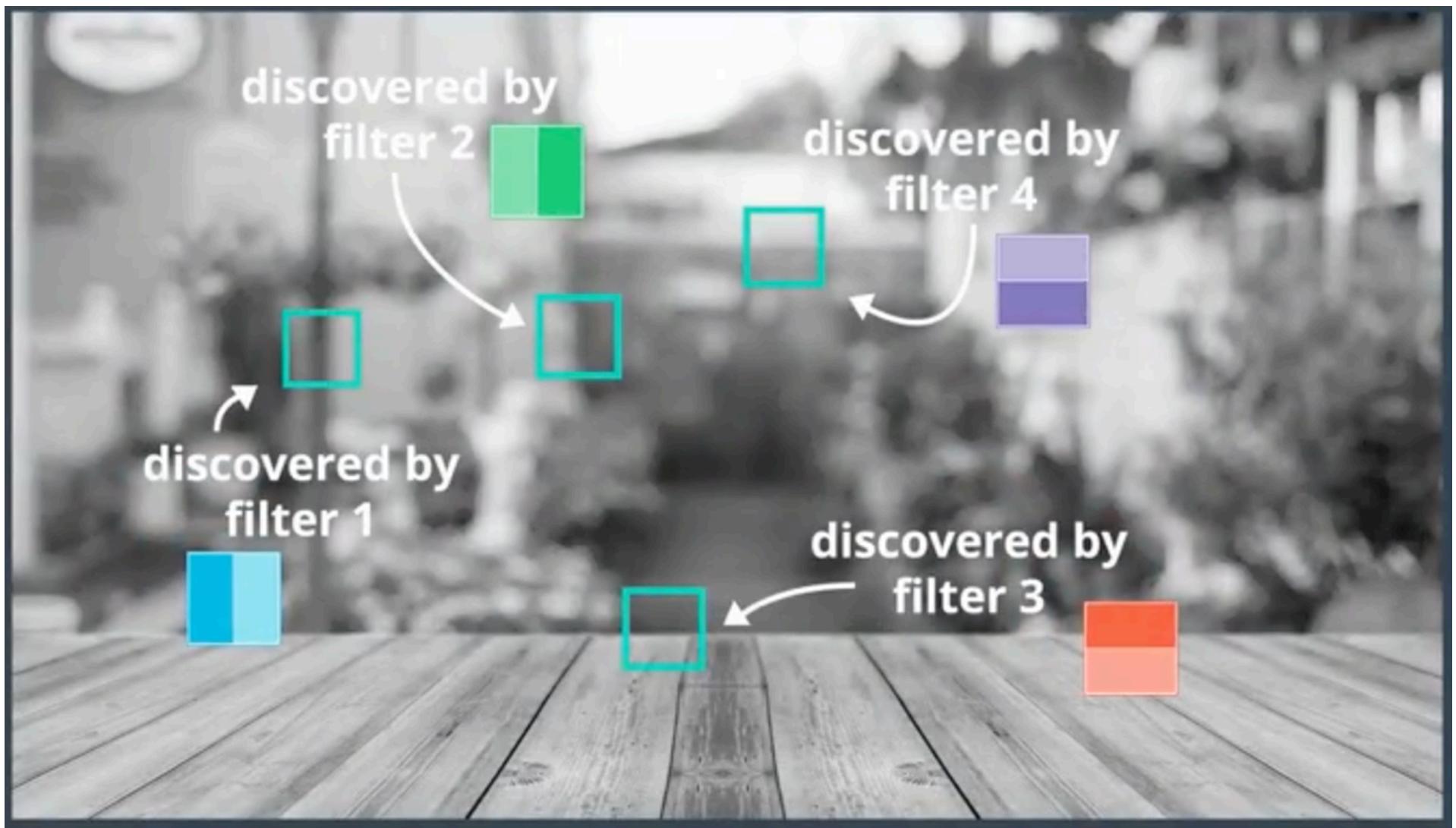
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 3 & 0 \end{bmatrix}$$

Convolutional Layer

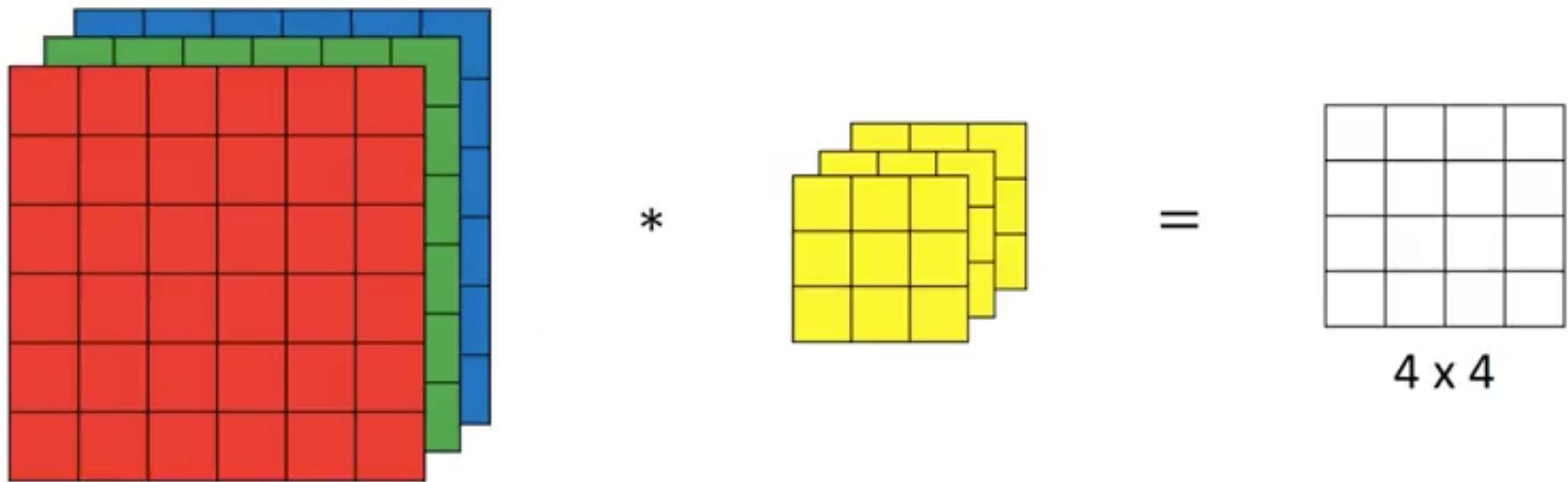
ReLU **SUM** $\left(\begin{array}{ccc} -1 * 0 & -1 * 0 & +1 * 1 \\ -1 * 0 & +1 * 1 & -1 * 0 \\ +1 * 1 & -1 * 0 & -1 * 0 \end{array} \right) = 3$



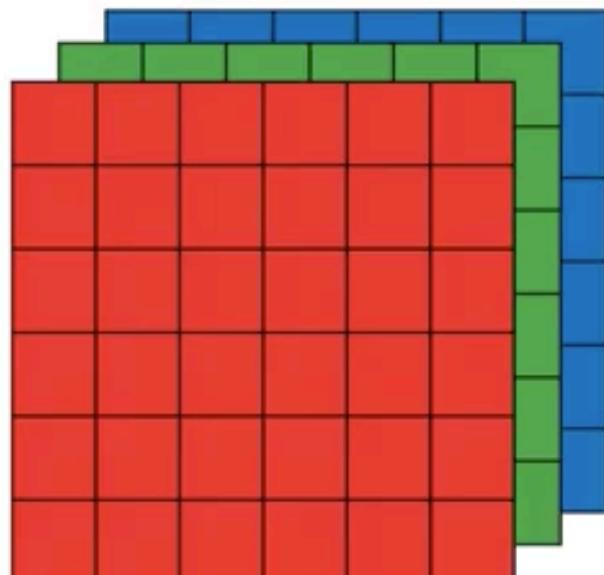




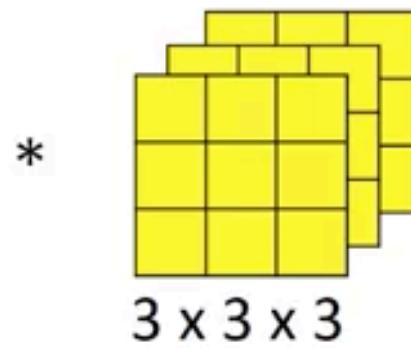
Convolução em Imagem RGB



Múltiplos Filtros



$6 \times 6 \times 3$

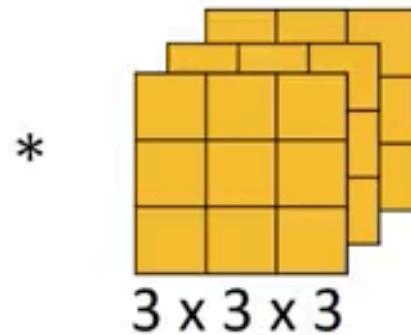


$3 \times 3 \times 3$

$=$

A 2D grid of 16 empty squares, representing a 4x4 output tensor.

4×4



$3 \times 3 \times 3$

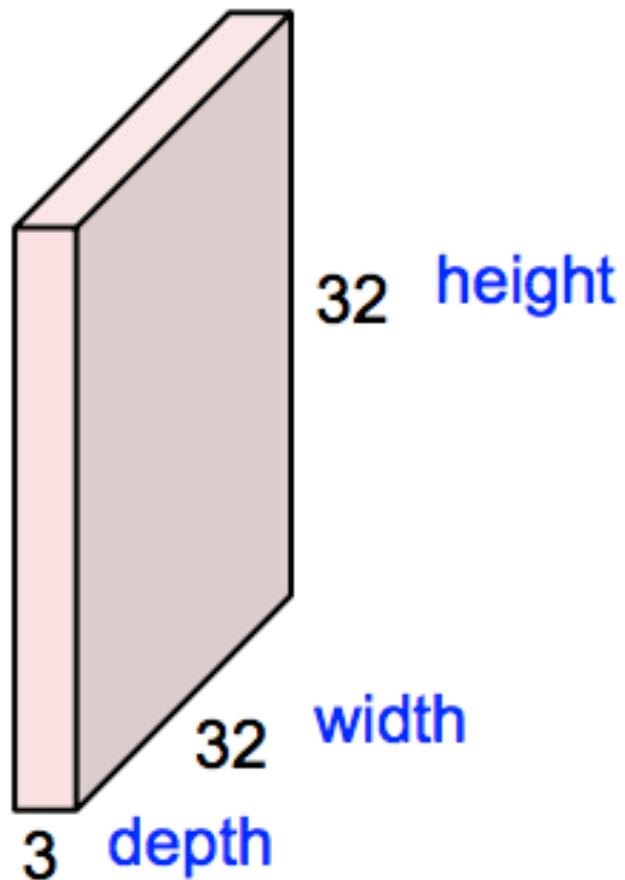
$=$

A 2D grid of 16 empty squares, representing a 4x4 output tensor.

4×4

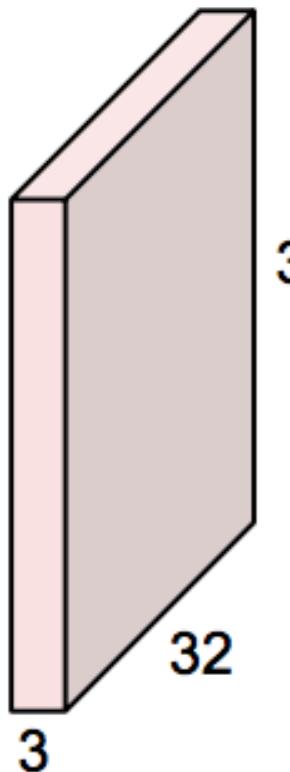
Camada Convolutiva

▷ Imagem 32x32x3

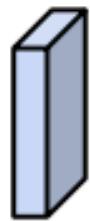


Camada Convolutiva

32x32x3 image

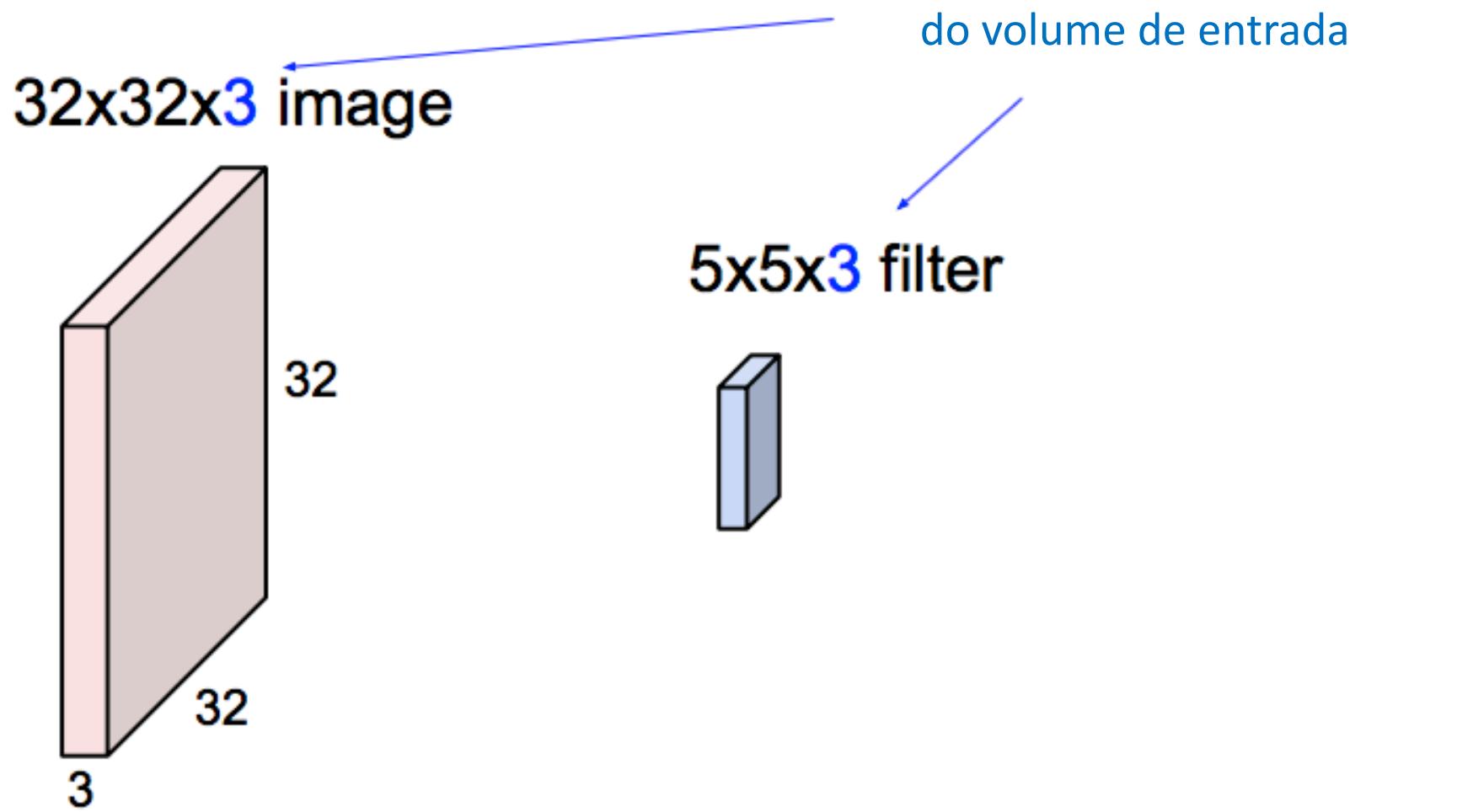


5x5x3 filter

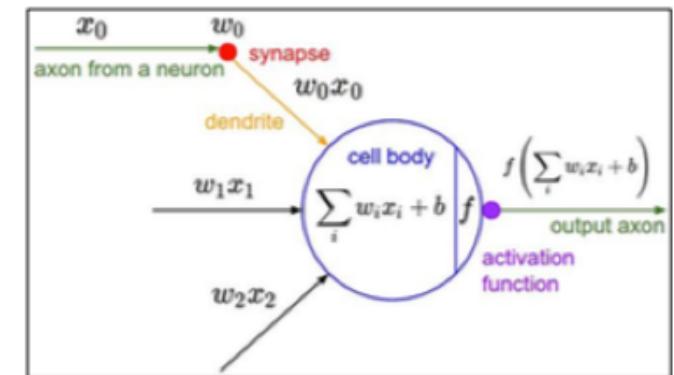
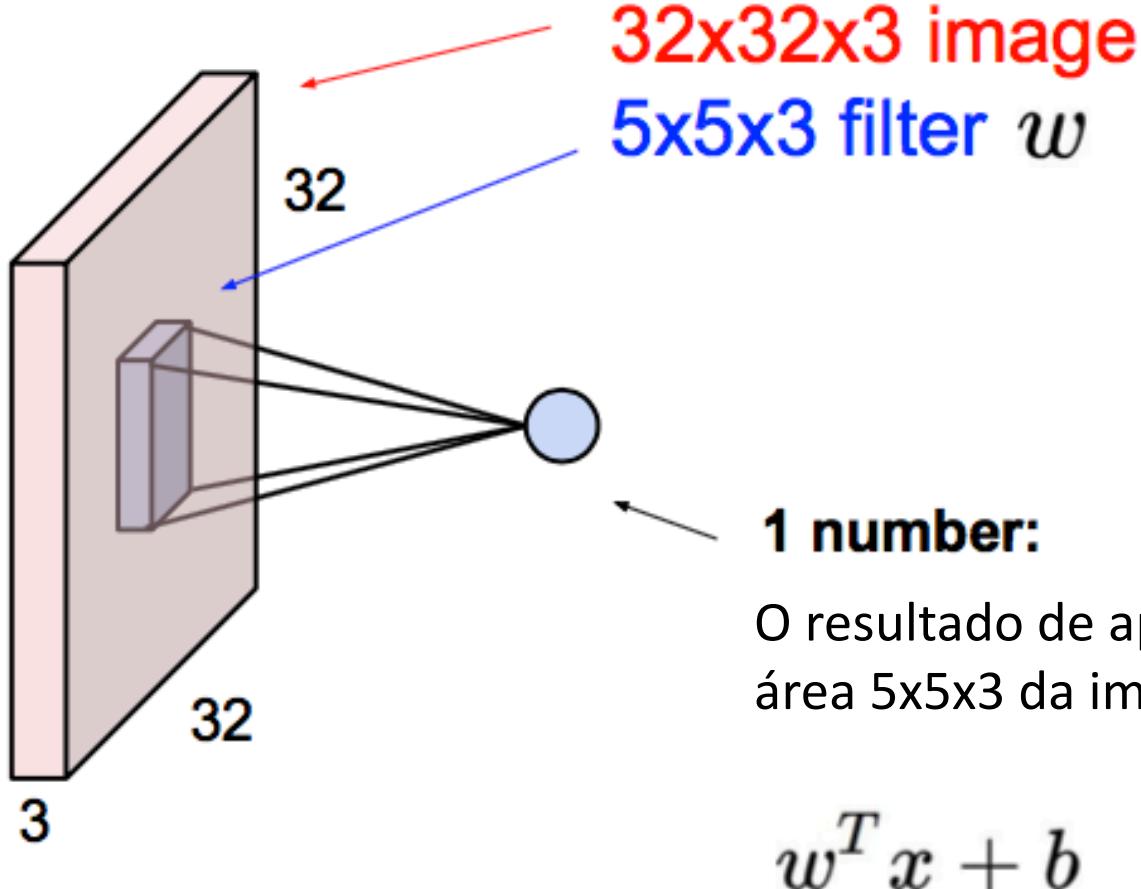


Convolve o filtro com a imagem

Camada Convolutiva

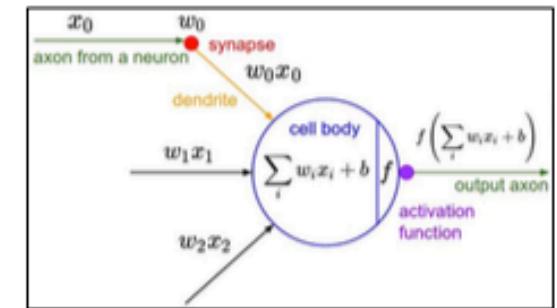
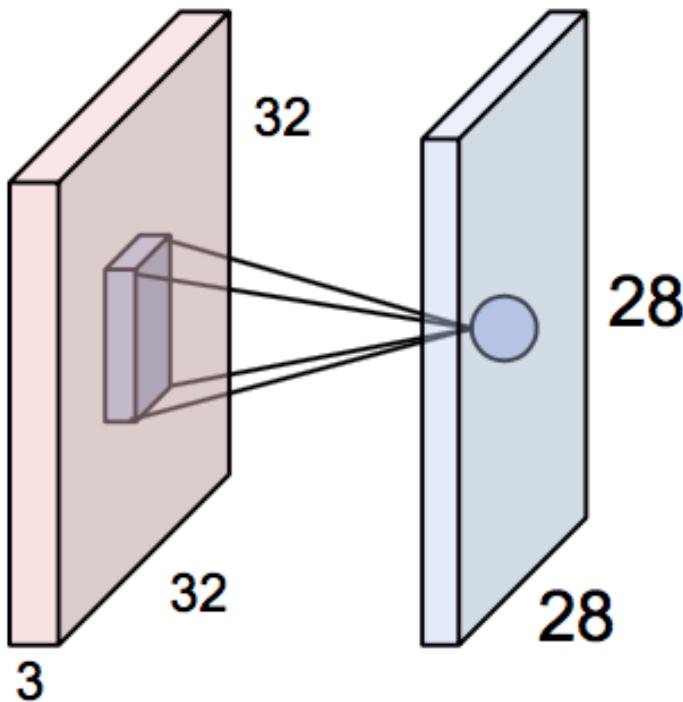


Camada Convolutiva



Neurônio com conectividade local

Camada Convolutiva

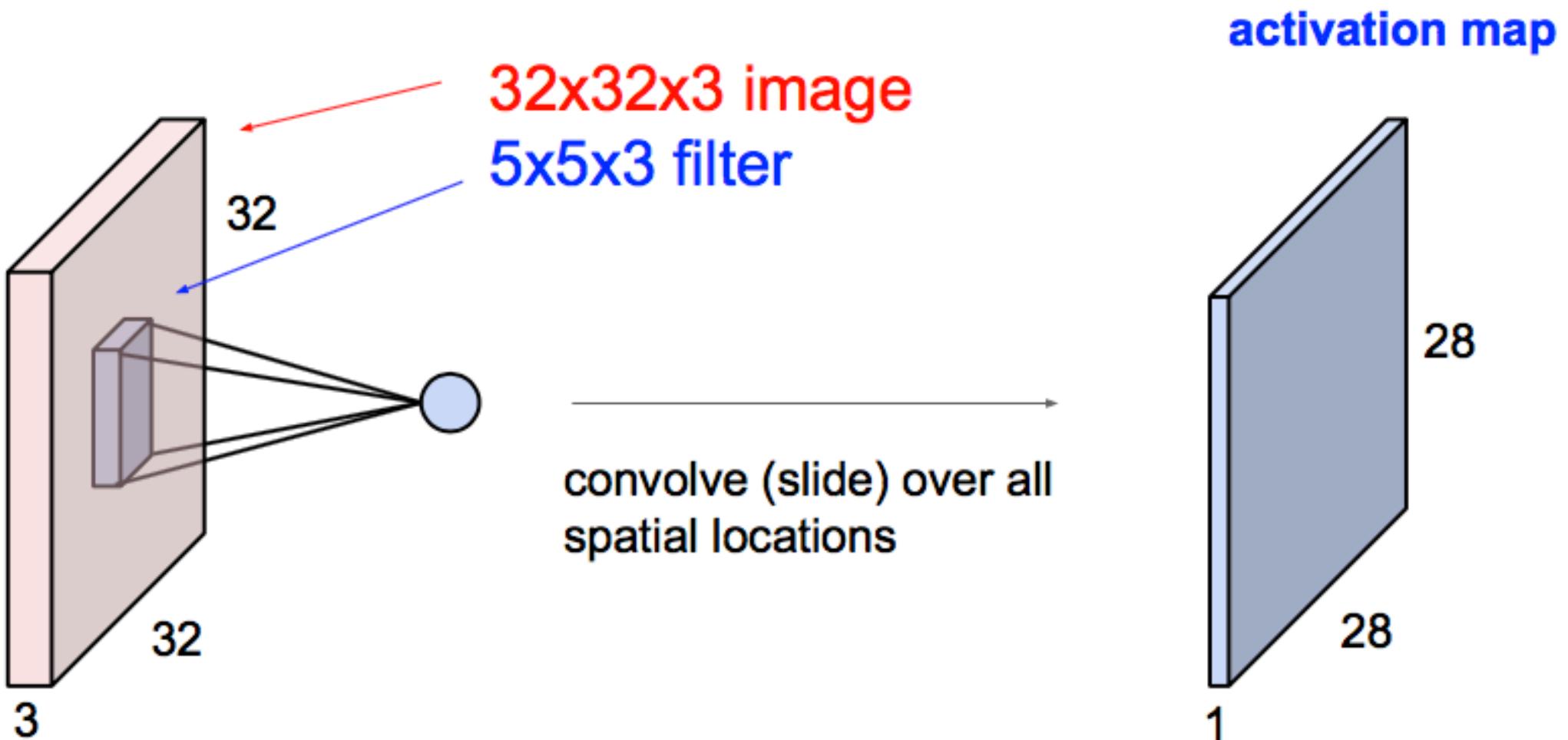


Um mapa de ativação é uma camada 28×28 de saídas de neurônios:

1. Cada uma está conectada a uma pequena região da imagem
2. Todos compartilham os parâmetros

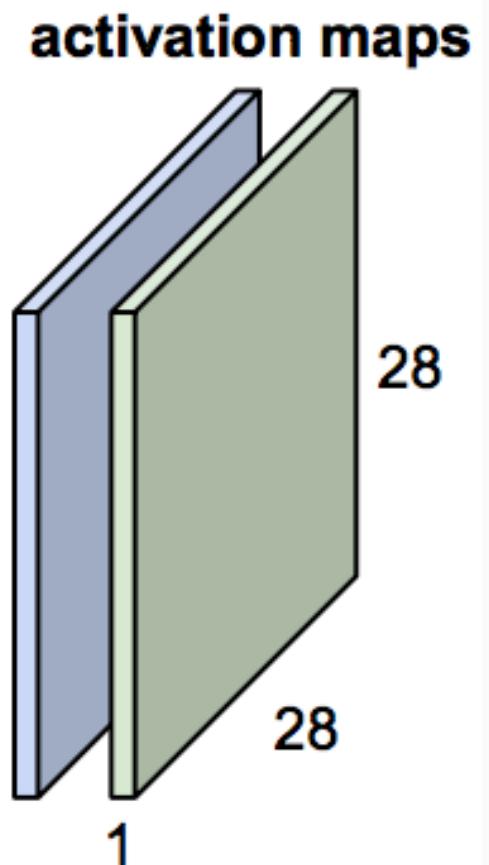
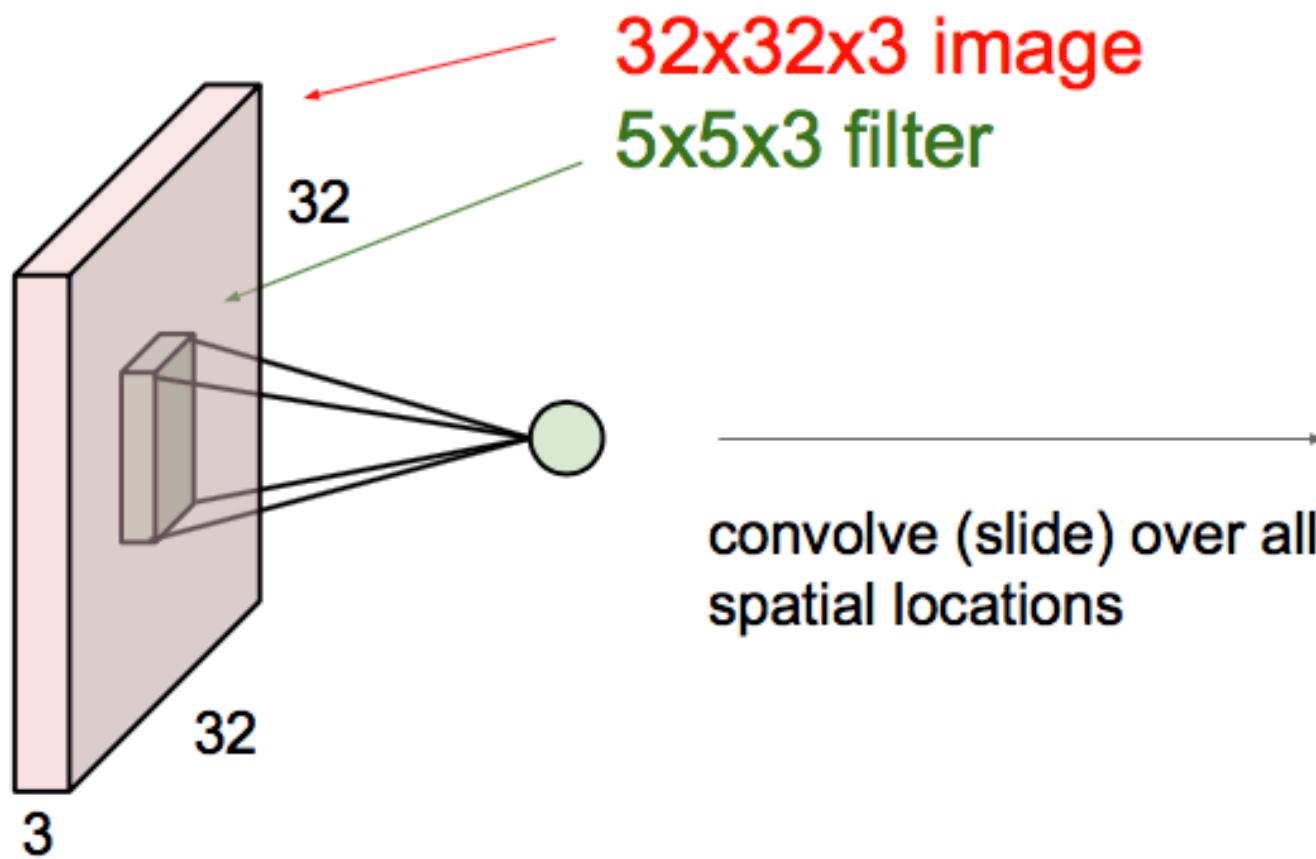
Filtro $5 \times 5 \rightarrow$ campo receptivo 5×5 para cada neurônio

Camada Convolutiva



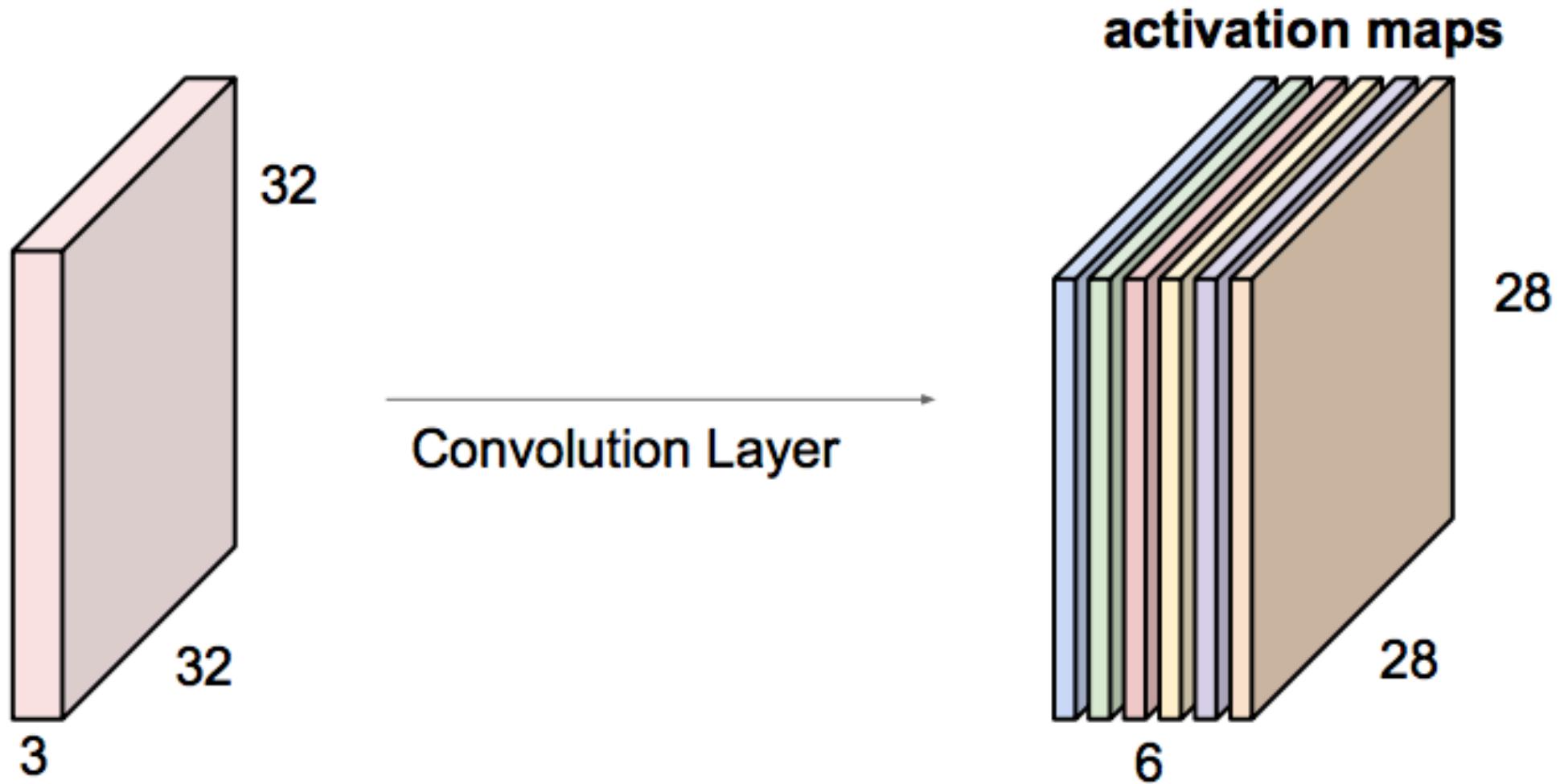
Camada Convolutiva

▷ Considere um segundo filtro **verde**



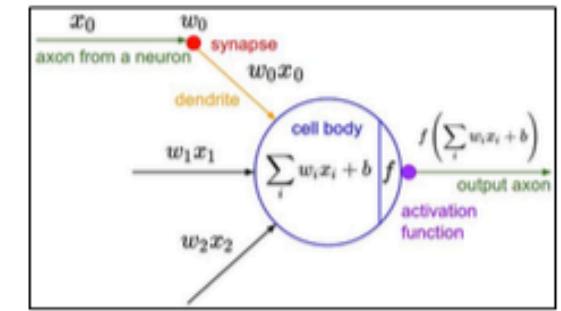
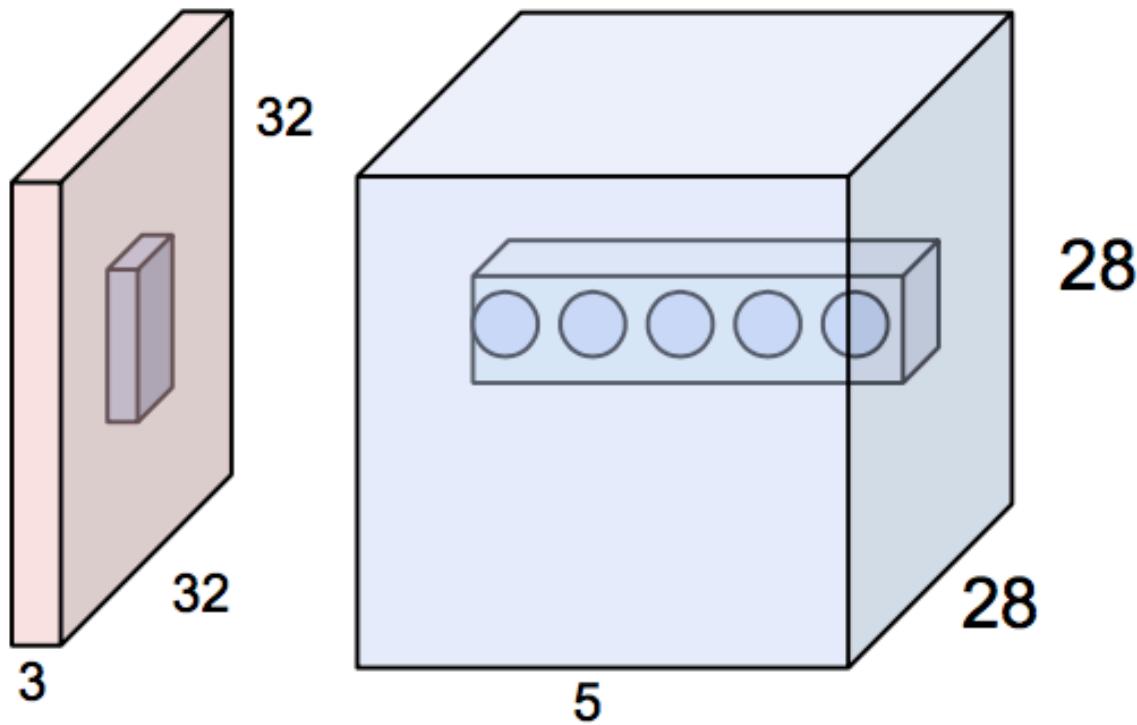
Camada Convolutiva

- Se tivéssemos 6 filtros 5×5 , teríamos 6 mapas de ativação



Juntamos tudo para obter uma "nova imagem" de tamanho $28 \times 28 \times 6$!

Camada Convolutiva

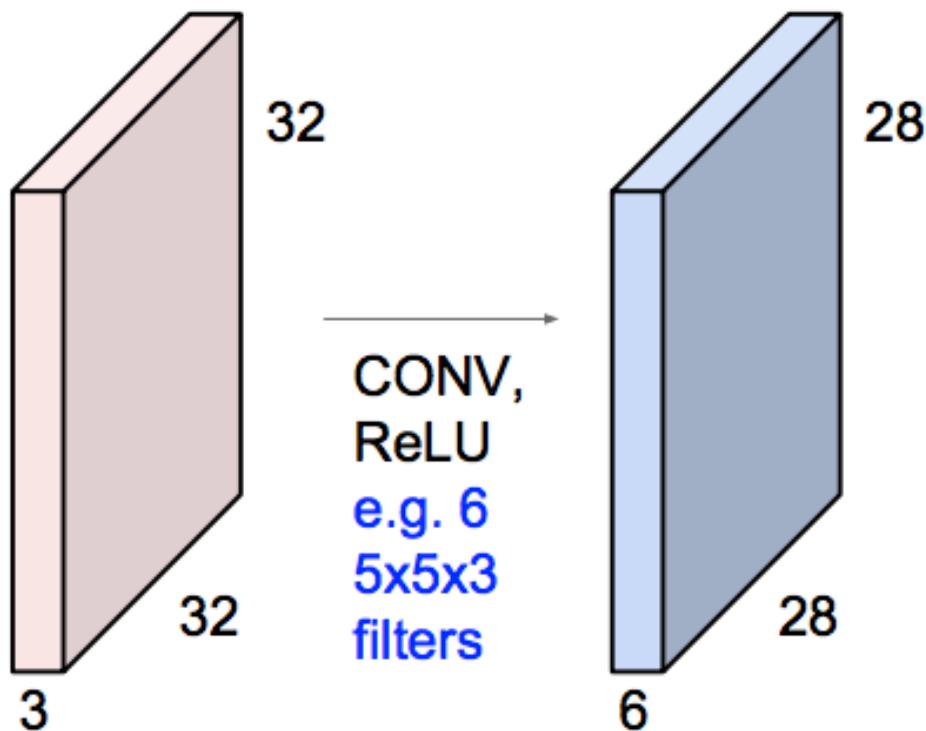


Ex: com 5 filtros, a camada CONV consiste de neurônios agrupados em um grid 3D (28x28x5)

Vão existir 5 diferentes neurônios observando a mesma região do volume de entrada

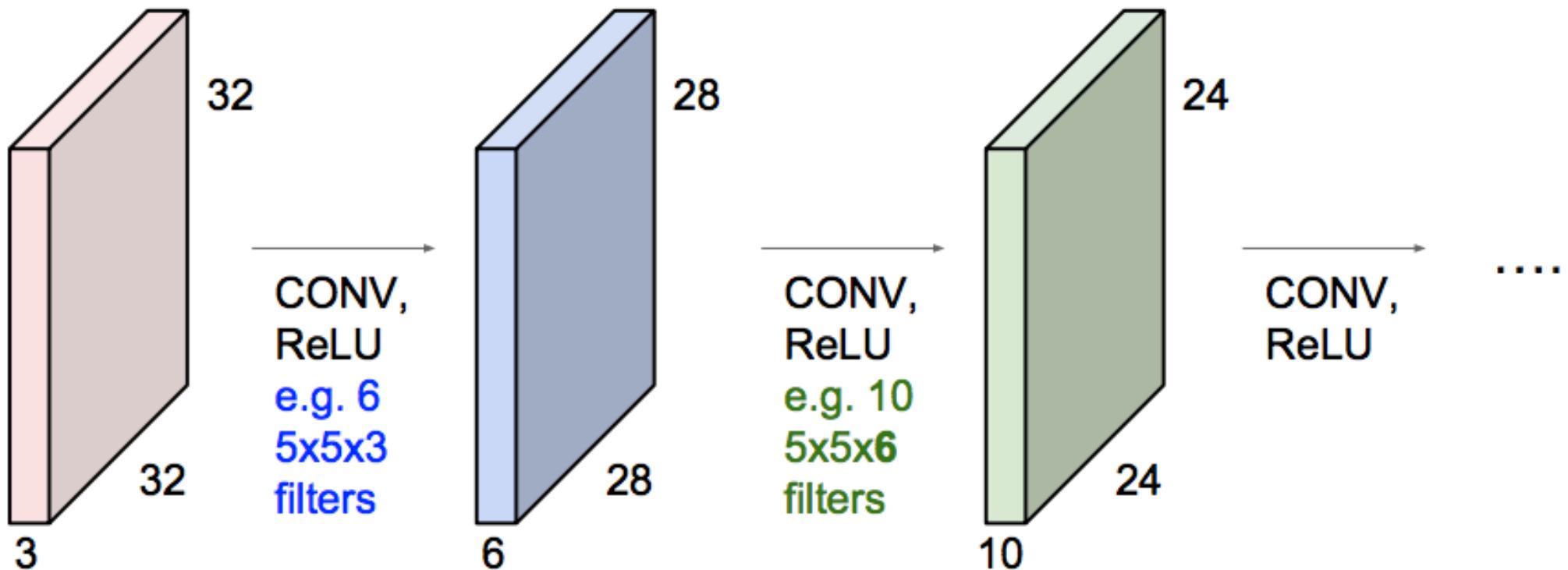
Camadas Convolutivas

- ▷ ConvNet é uma sequencia de Camadas Convolutivas, intercaladas com funções de ativação



Camadas Convolutivas

- ▷ ConvNet é uma sequencia de Camadas Convolutivas, intercaladas com funções de ativação

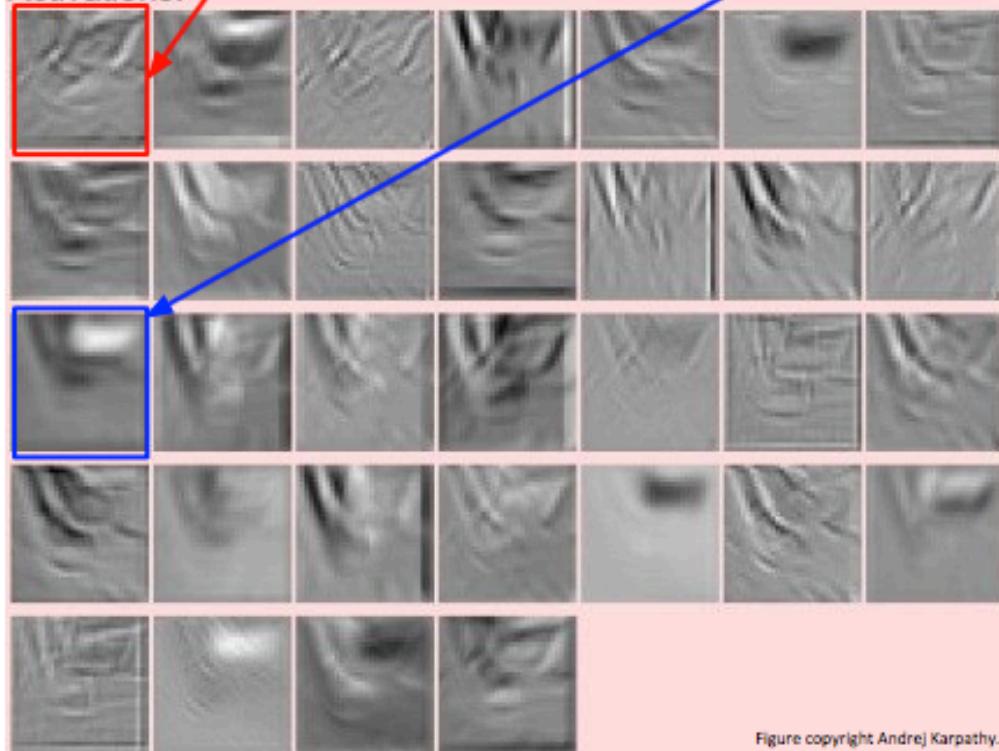


Exemplo ConvNet



one filter =>
one activation map

Activations:



example 5x5 filters
(32 total)

Figure copyright Andrej Karpathy.

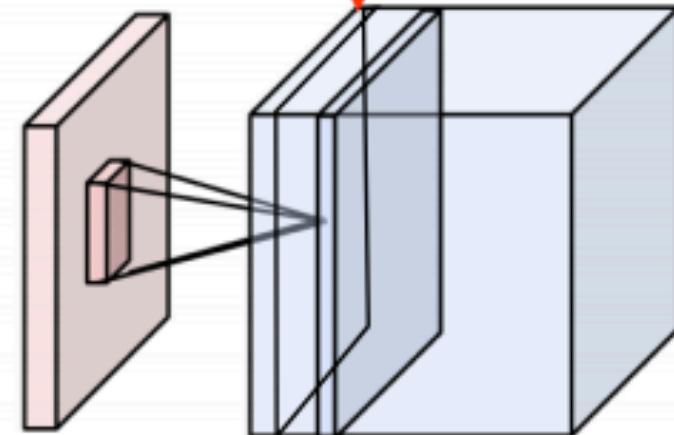
(Input)



one filter = one depth slice (or activation map)

(32 filters, each 3x5x5)

Activations:



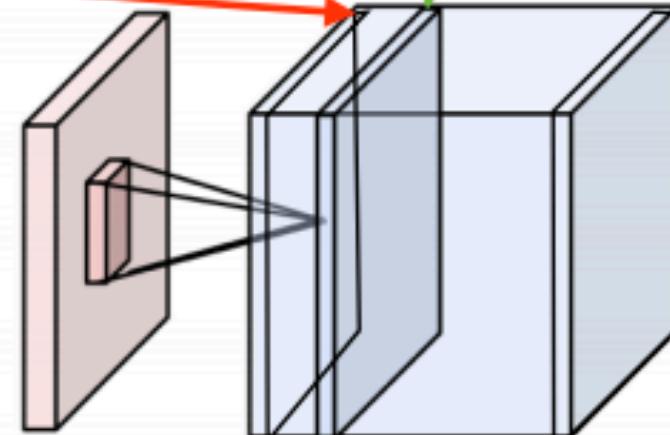
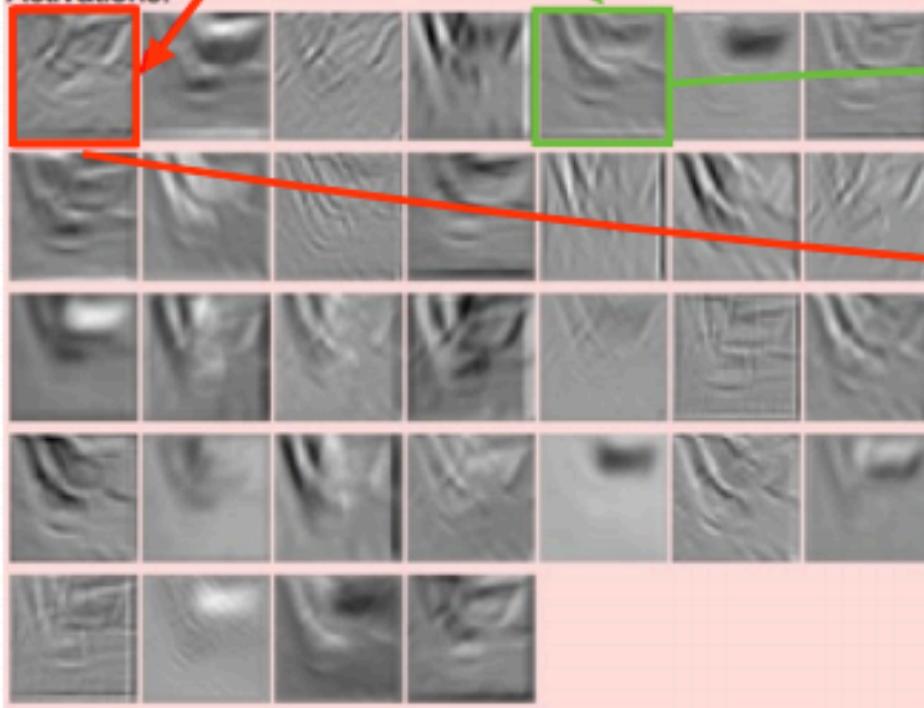
(Input)



one filter = one depth slice (or activation map)

(32 filters, each 3x5x5)

Activations:



(Input)



one filter = one depth slice (or activation map)

(32 filters, each 3x5x5)

Activations:

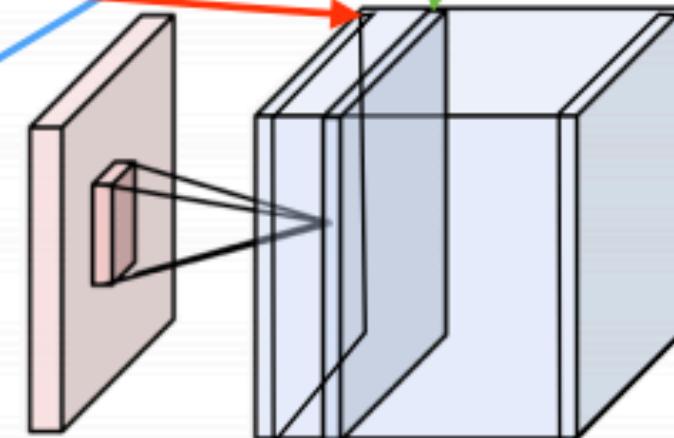
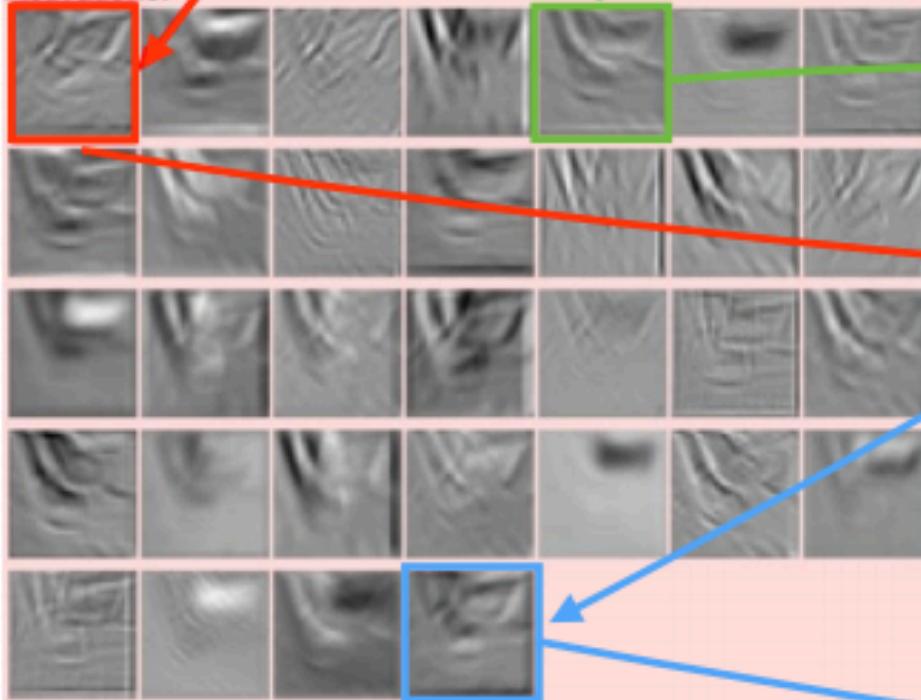


Figure: Andrej Karpathy

Resumo: Camada Convolutiva

▷ Aceita volume do tamanho $W_1 \times H_1 \times D_1$

▷ Requer 4 hiperparâmetros:

- Número de filtros K ,
- Tamanho do Filtro F ,
- Stride S ,
- Quantidade de zero padding P .

▷ Produz um volume $W_2 \times H_2 \times D_2$ onde:

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$
- $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- $D_2 = K$

▷ $F \cdot F \cdot D_1$ pesos por filtro. Total de $(F \cdot F \cdot D_1)$ pesos e K biases;

Resumo: Camada Convolutiva

▷ Aceita volume do tamanho $W_1 \times H_1 \times D_1$

▷ Requer 4 hiperparâmetros:

- Número de filtros **K**,
- Tamanho do Filtro **F**,
- Stride **S**,
- Quantidade de zero padding **P**.

Configurações comuns:

- K**= (potência de 2. Ex: 32,64,128,512)
- $F=3, S=1, P=1$
 - $F=5, S=1, P=2$
 - $F=1, S=1, P=0$

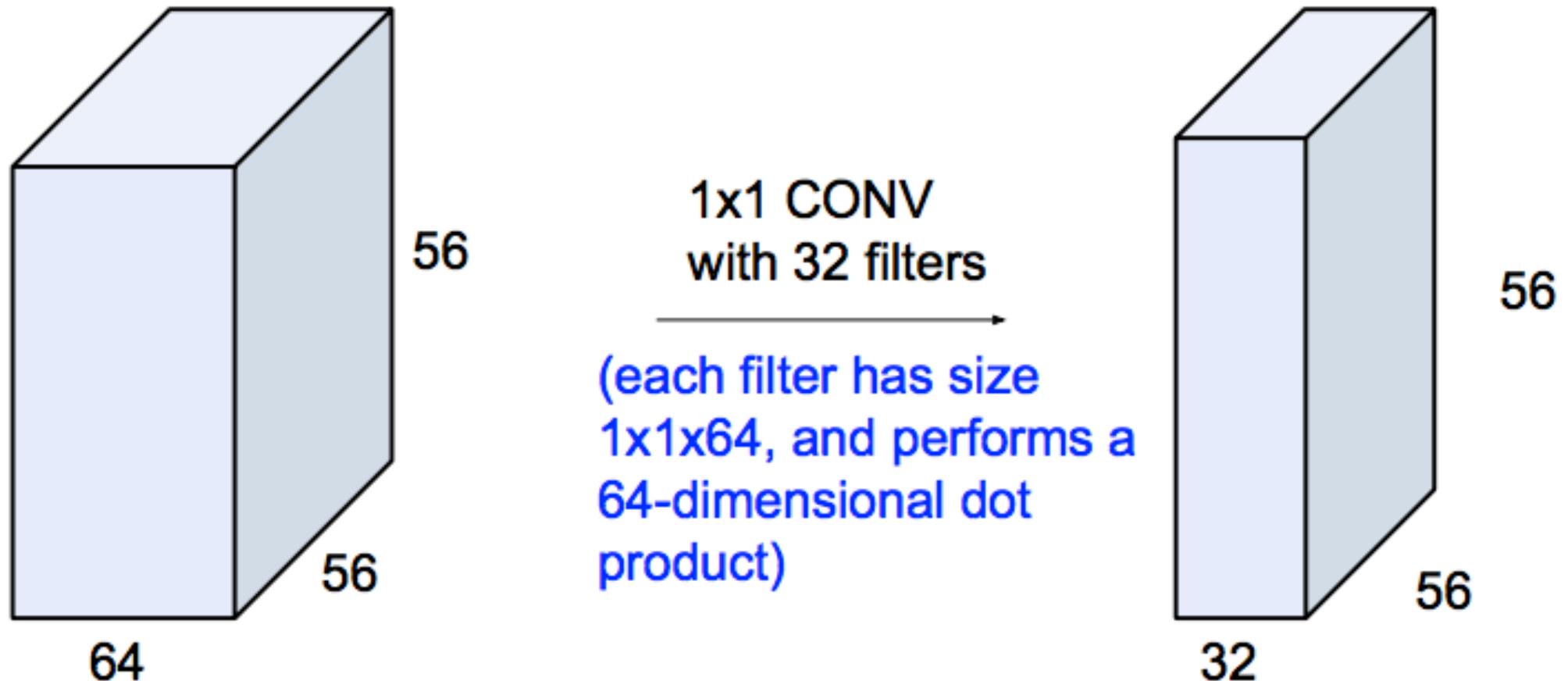
▷ Produz um volume $W_2 \times H_2 \times D_2$ onde:

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$
- $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- $D_2 = K$

▷ $F \cdot F \cdot D_1$ pesos por filtro. Total de $(F \cdot F \cdot D_1) \cdot K$ biases;

Camada convolutiva

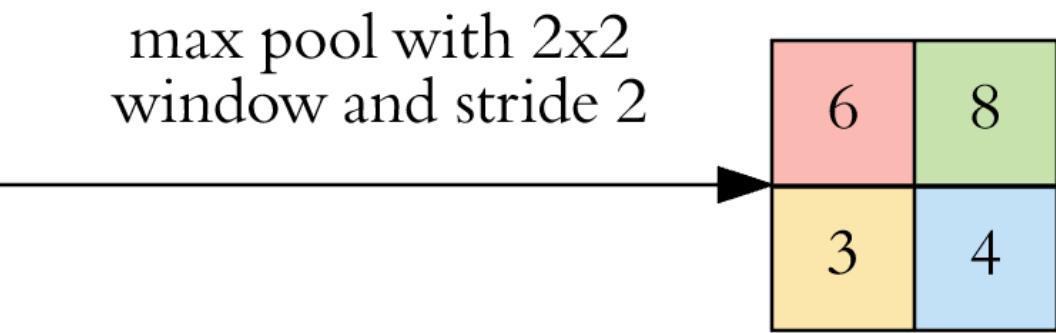
- ▷ Camada 1x1 também faz sentido



Pooling

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

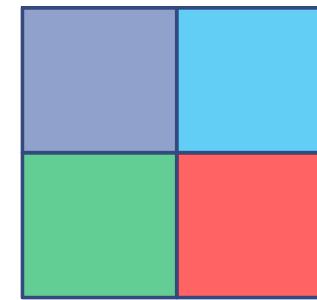
max pool with 2x2
window and stride 2



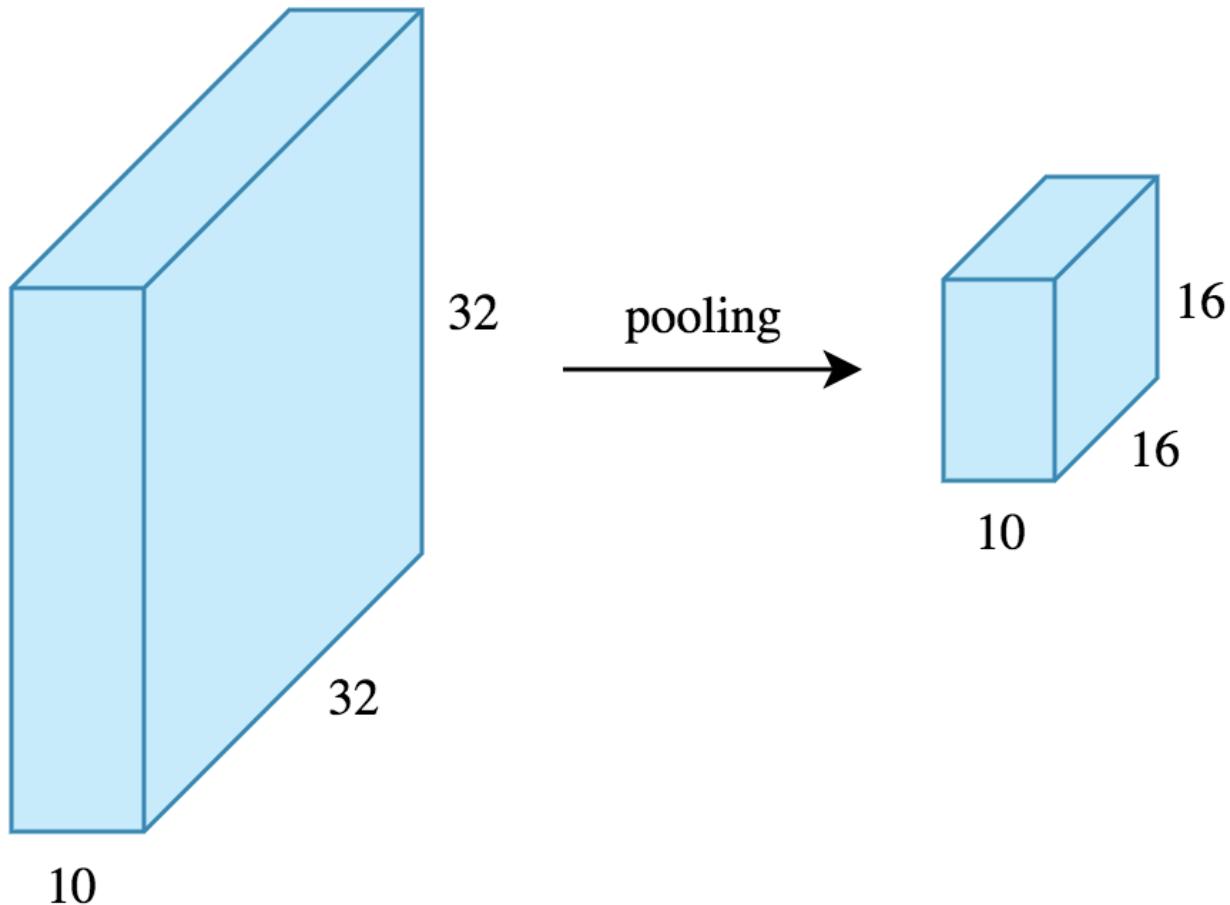
6	8
3	4

Camada de Pooling: Max Pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

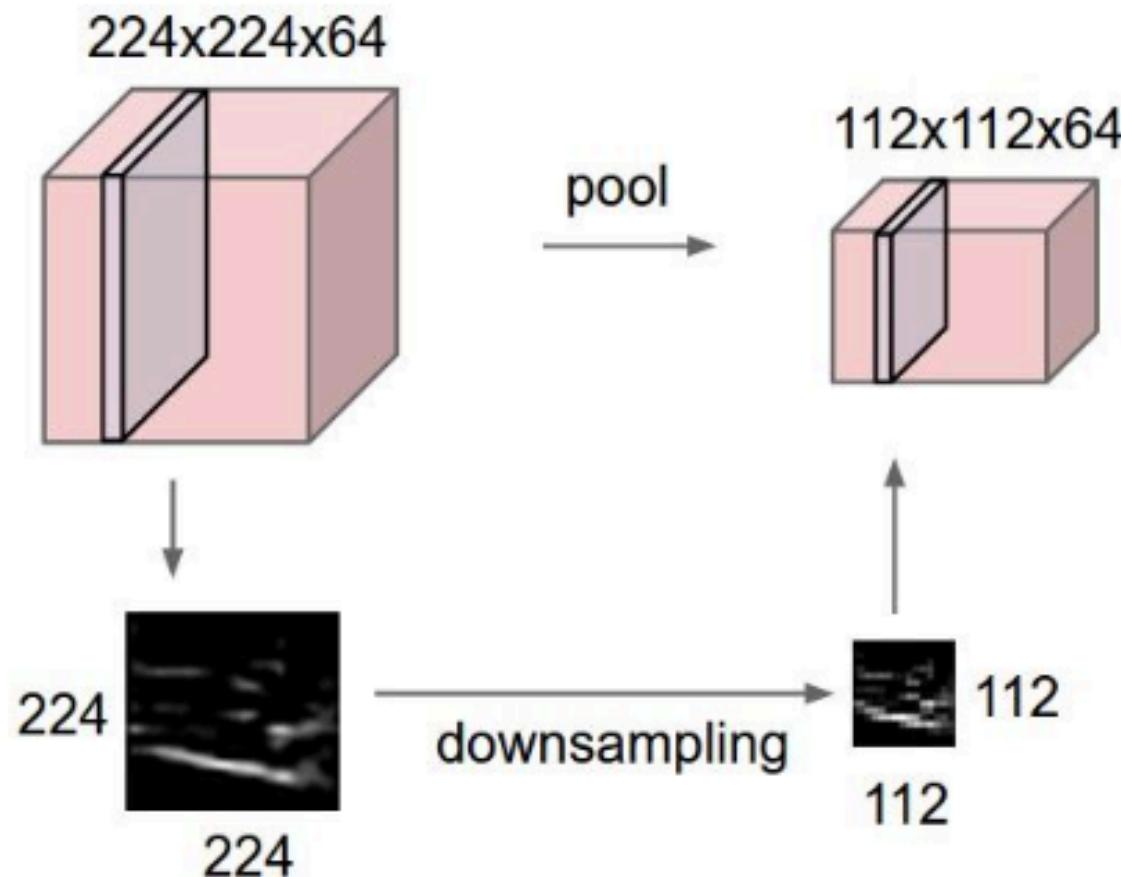


Pooling



Camada de Pooling

- ▷ Torna a representação menor
- ▷ Opera sobre cada mapa de ativação independentemente.



Resumo de Pooling

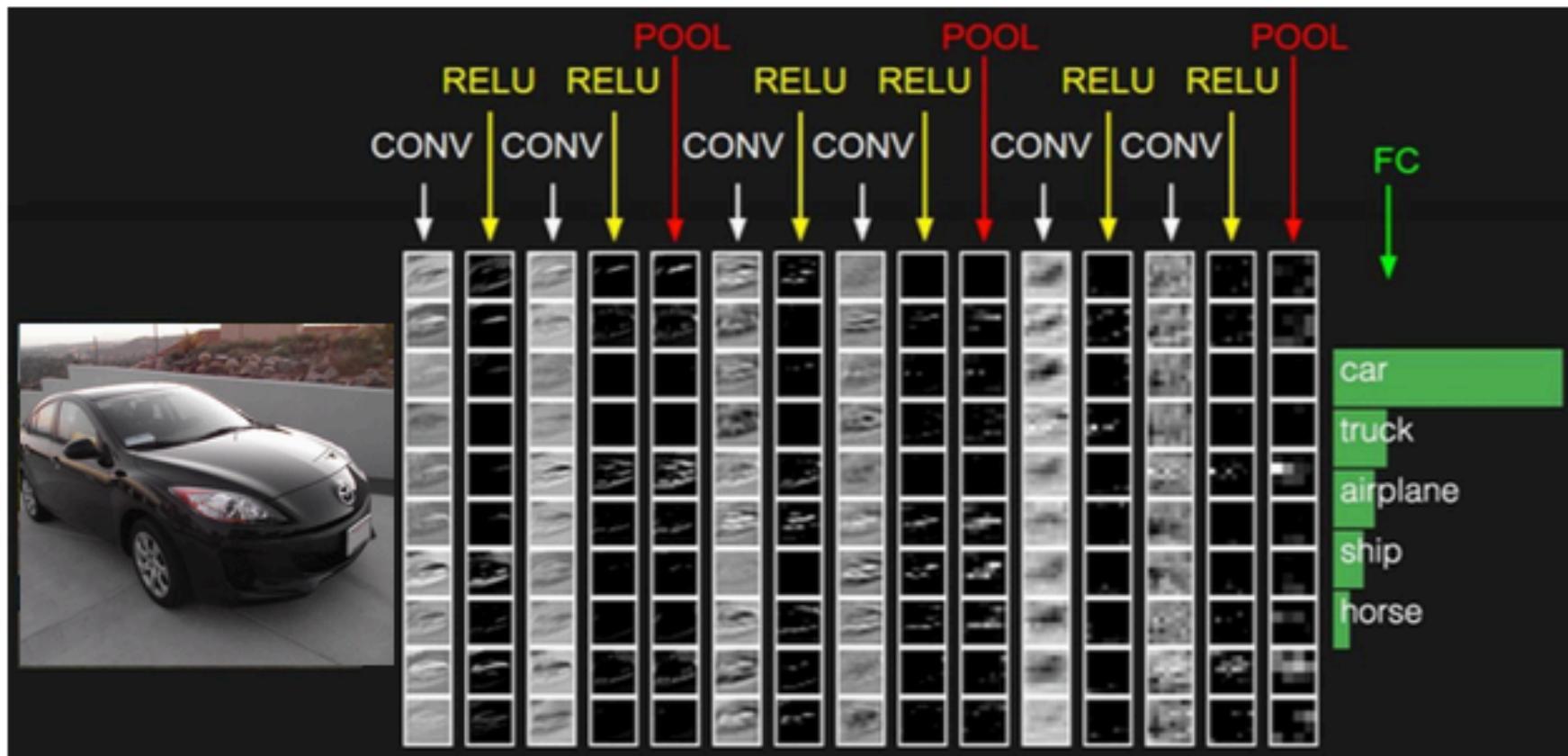
- ▷ Aceita um volume de tamanho $W_1 \times H_1 \times D_1$
- ▷ Requer dois hiperparâmetros:
 - Tamanho da janela F ,
 - Stride S
- ▷ Produz um volume de tamanho $W_2 \times H_2 \times D_2$, onde:
 - $W_2 = \frac{W_1 - F}{S} + 1$
 - $H_2 = \frac{H_1 - F}{S} + 1$
 - $D_2 = D_1$
- ▷ Introduz zero parâmetros
- ▷ Não é comum usar zero-padding para camadas de pooling.

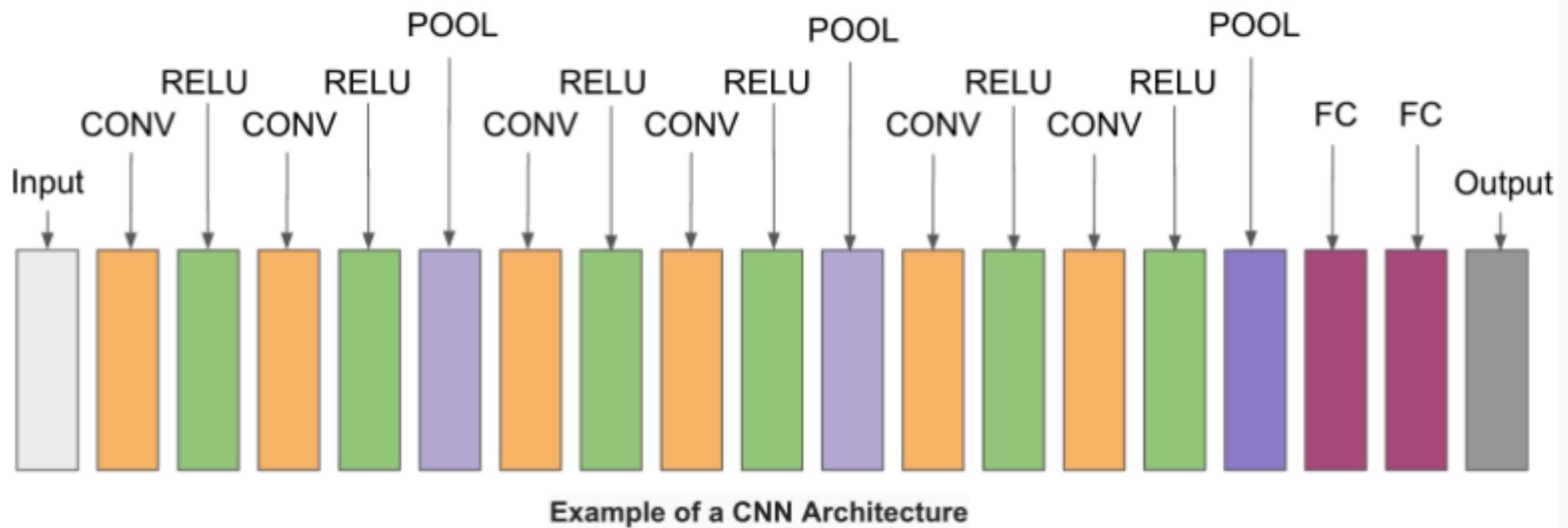
Resumo de Pooling

- ▷ Aceita um volume de tamanho $W_1 \times H_1 \times D_1$ Configurações comuns:
 $F=2, S=2$
 $F=3, S=2$
- ▷ Requer três hiperparâmetros:
 - Tamanho da janela F ,
 - Stride S
- ▷ Produz um volume de tamanho $W_2 \times H_2 \times D_2$, onde:
 - $W_2 = \frac{W_1 - F}{S} + 1$
 - $H_2 = \frac{H_1 - F}{S} + 1$
 - $D_2 = D_1$
- ▷ Introduz zero parâmetros
- ▷ Não é comum usar zero-padding para camadas de pooling.

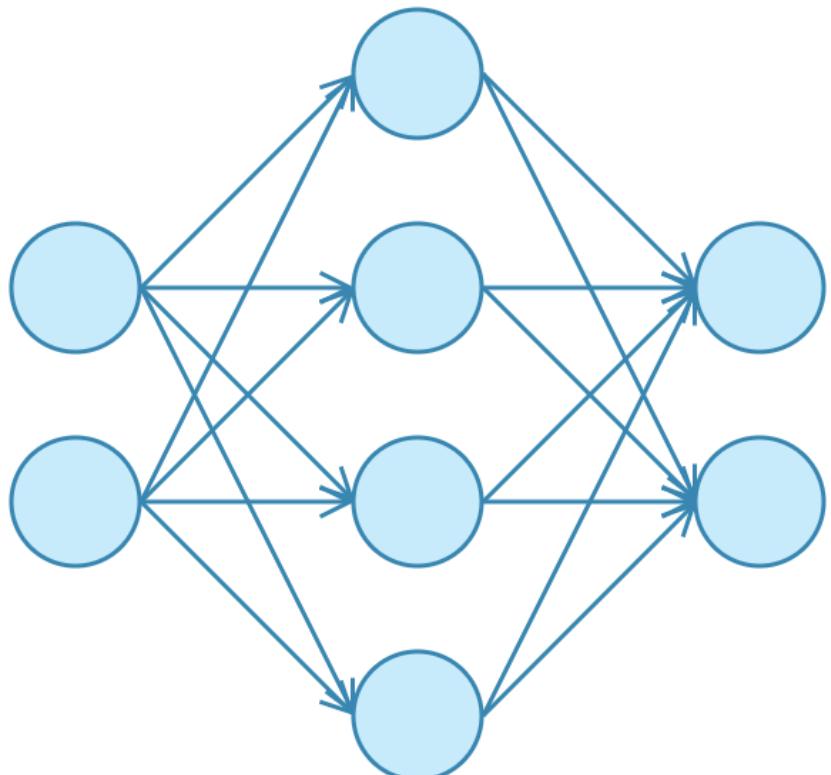
Fully Connected Layer (FC layer)

- ▷ Contem neurônios que conectam todo o volume de entrada, como em redes neurais tradicionais.

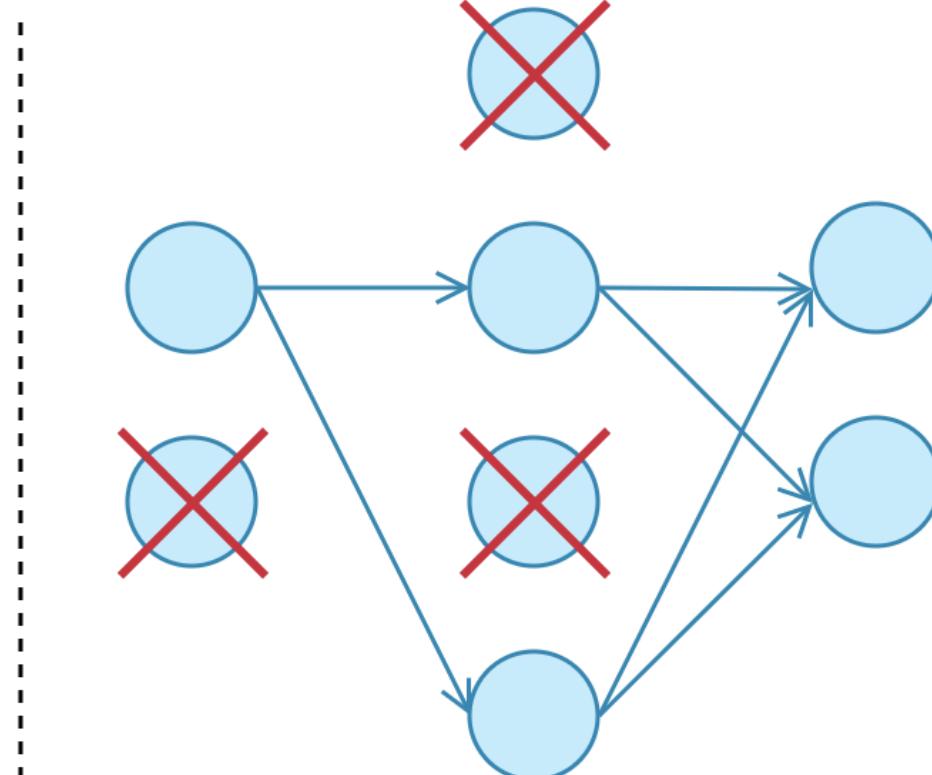




Dropout



Sem Dropout



Com Dropout

Resumo

- ▷ ConvNets enfileiram camadas CONV, POOL, FC;
- ▷ Tendência em direção à filtros menores e arquiteturas mais profundas;
- ▷ Arquiteturas tradicionais se parecem com:
 - **[(CONV-RELU)*N-POOL?] *M-(FC-RELU)*K, SOFTMAX**
 - Onde N é até aprox. 5, M é alto, $0 \leq K \leq 2$
 - Arquiteturas como ResNet/GoogLeNet desafiam esse paradigma.

CNN com Keras



Modelos Sequenciais

```
1 from keras.models import Sequential  
2 from keras.layers import Dense  
3 model = Sequential()  
4 model.add(Dense(2, input_dim=1))  
5 model.add(Dense(1))
```

Modelo Funcional

```
1 # Multilayer Perceptron
2 from keras.utils import plot_model
3 from keras.models import Model
4 from keras.layers import Input
5 from keras.layers import Dense
6 visible = Input(shape=(10,))
7 hidden1 = Dense(10, activation='relu')(visible)
8 hidden2 = Dense(20, activation='relu')(hidden1)
9 hidden3 = Dense(10, activation='relu')(hidden2)
10 output = Dense(1, activation='sigmoid')(hidden3)
11 model = Model(inputs=visible, outputs=output)
12 # ...
```

INPUT

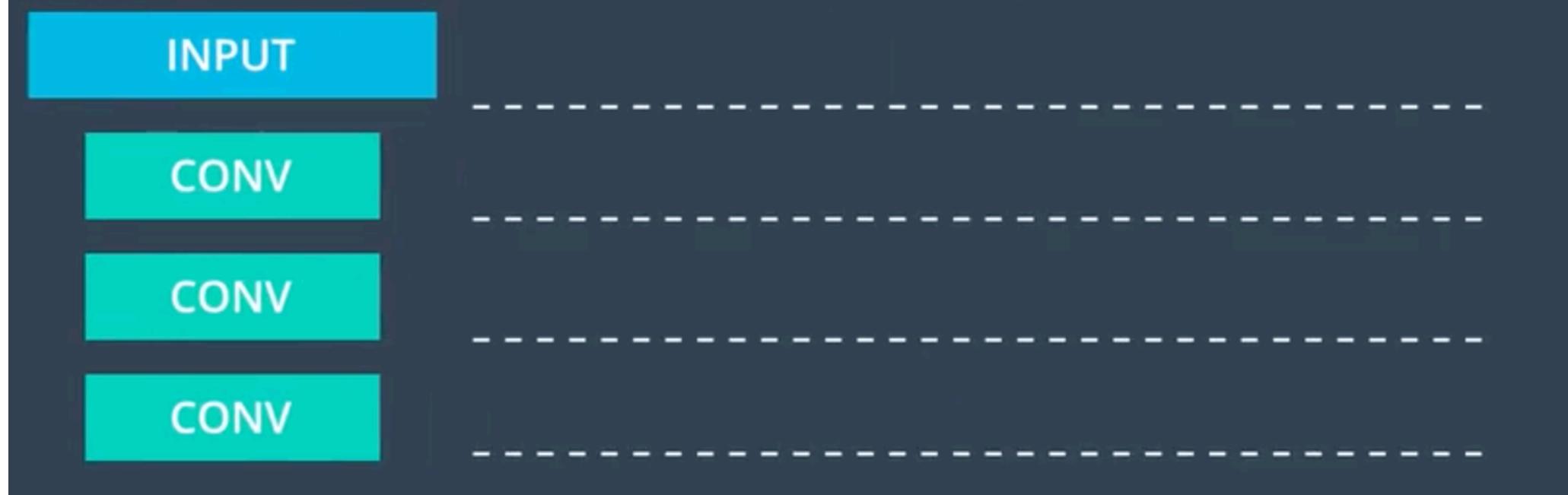
CONV

CONV

CONV

Camada convolutiva no keras

```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```



Camada convolutiva no keras

```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```

INPUT

CONV

CONV

CONV



2

2

2

Camada convolutiva no keras

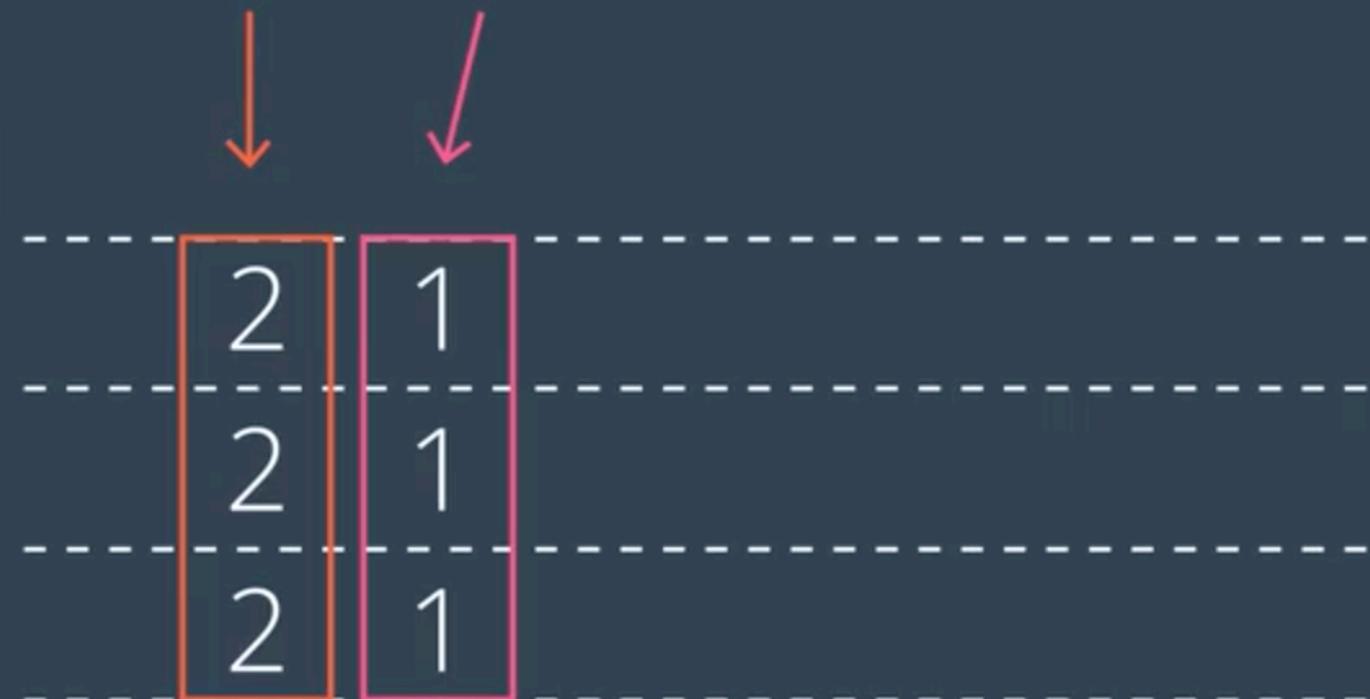
```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```

INPUT

CONV

CONV

CONV



Camada convolutiva no keras

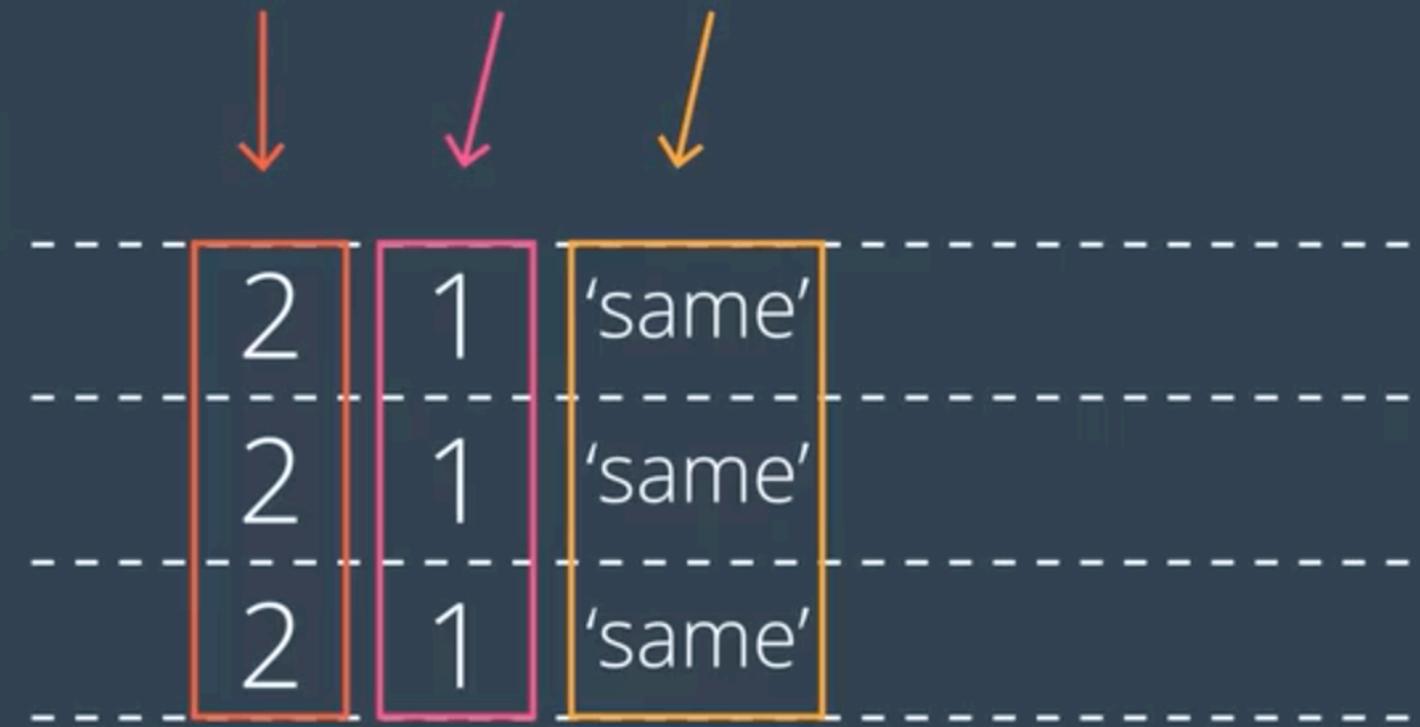
```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```

INPUT

CONV

CONV

CONV



Camada convolutiva no keras

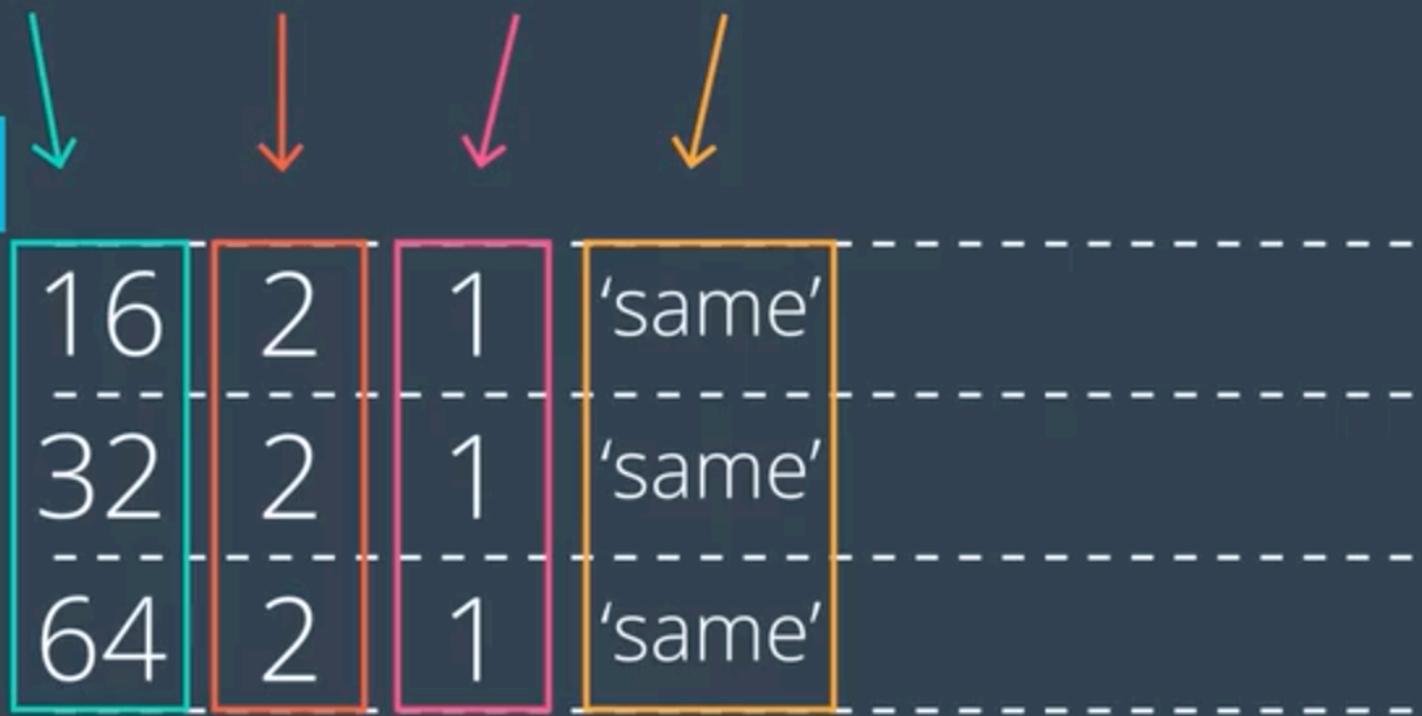
```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```

INPUT

CONV

CONV

CONV



Camada convolutiva no keras

```
Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)
```

INPUT

CONV

CONV

CONV

16

2

1

'same'

(32, 32, 3)

32

2

1

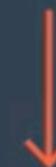
'same'

64

2

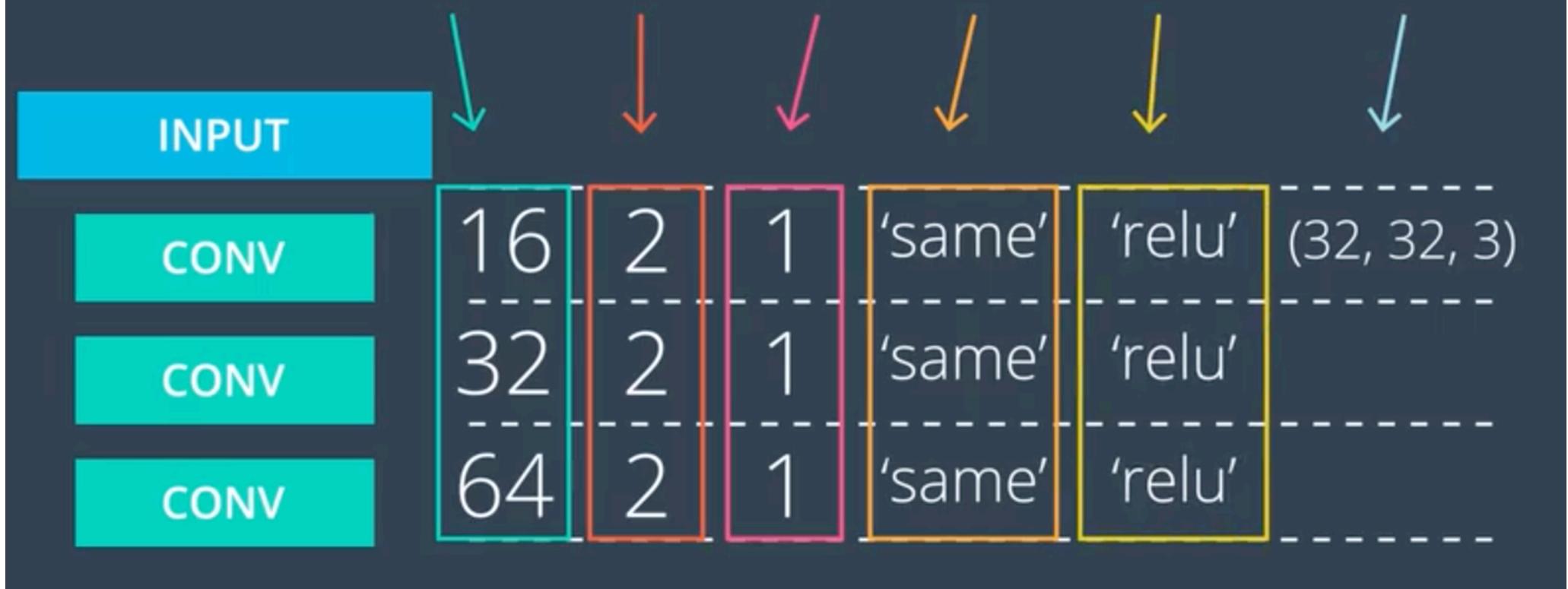
1

'same'



Camada convolutiva no keras

Convolution2D(filters, kernel_size, strides, padding, activation, input_shape)



```
1 from keras.models import Sequential
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
3
4 model = Sequential()
5 model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
6                  input_shape=(32, 32, 3)))
7 model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
8 model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
9
10 model.summary()
11
```

Layer (type)	Output Shape	Param #
=====		
conv2d_4 (Conv2D)	(None, 32, 32, 16)	208
conv2d_5 (Conv2D)	(None, 32, 32, 32)	2080
conv2d_6 (Conv2D)	(None, 32, 32, 64)	8256
=====		
Total params:	10,544	
Trainable params:	10,544	
Non-trainable params:	0	

MaxPooling2D(pool_size, stride, padding)

INPUT

CONV

POOL

CONV

POOL

CONV

POOL

2

2

2

2

2

2

↓

(default ok)

```
1 from keras.models import Sequential
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
3
4 model = Sequential()
5 model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
6                  input_shape=(32, 32, 3)))
7 model.add(MaxPooling2D(pool_size=2))
8 model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
9 model.add(MaxPooling2D(pool_size=2))
10 model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
11 model.add(MaxPooling2D(pool_size=2))
12
13 model.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 32, 32, 16)	208
=====		
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
=====		
conv2d_2 (Conv2D)	(None, 16, 16, 32)	2080
=====		
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 32)	0
=====		
conv2d_3 (Conv2D)	(None, 8, 8, 64)	8256
=====		
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 64)	0
=====		
Total params:	10,544	
Trainable params:	10,544	
Non-trainable params:	0	

Contains a ton
of spatial information



Contains no
spatial information



```

1 from keras.models import Sequential
2 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
3
4 model = Sequential()
5 model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
6                   input_shape=(32, 32, 3)))
7 model.add(MaxPooling2D(pool_size=2))
8 model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
9 model.add(MaxPooling2D(pool_size=2))
10 model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
11 model.add(MaxPooling2D(pool_size=2))
12 model.add(Flatten())
13 model.add(Dense(500, activation='relu'))
14 model.add(Dense(10, activation='softmax'))
15
16 model.summary()

```

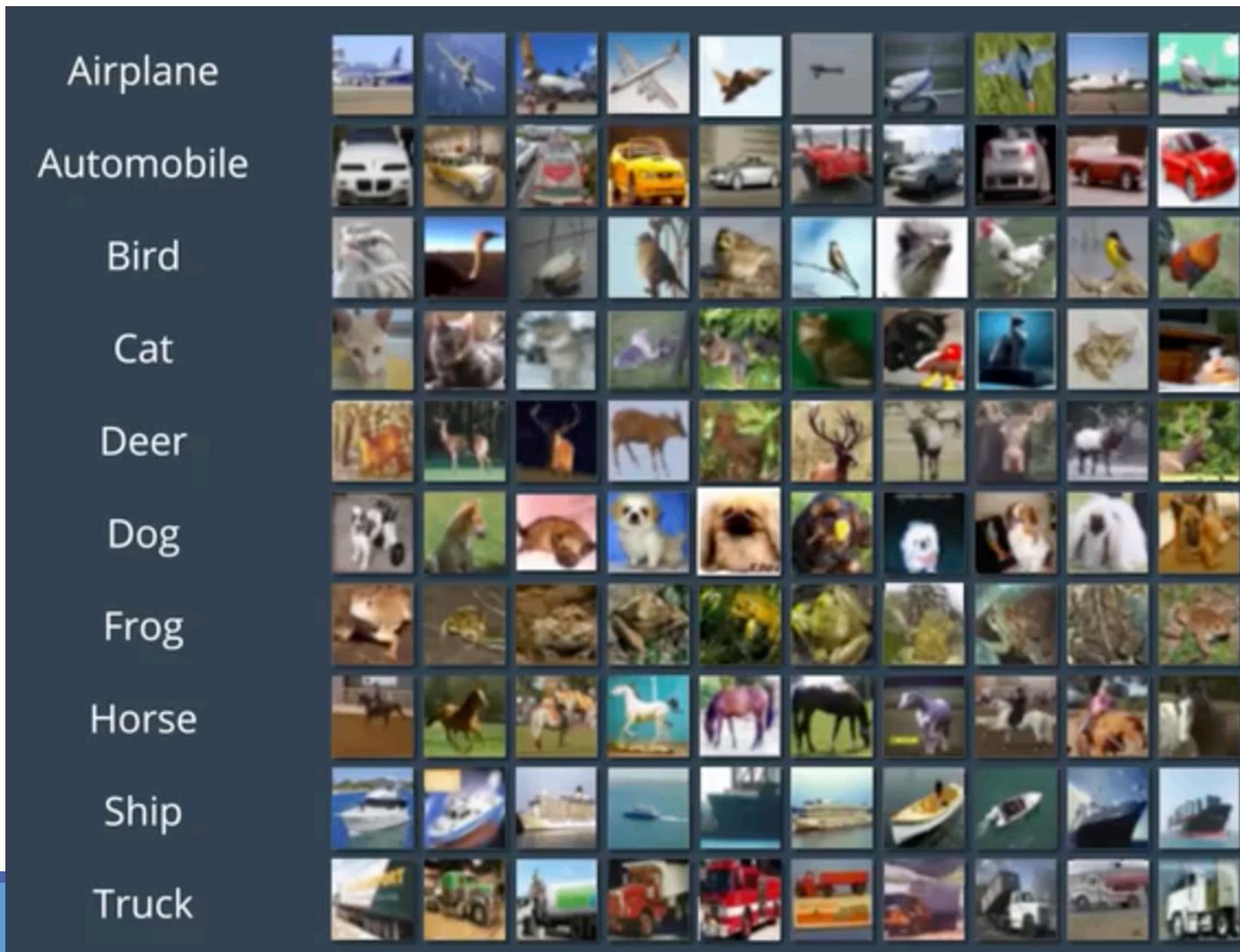
Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 32, 32, 16)	208
<hr/>		
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 16, 16, 32)	2080
<hr/>		
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 32)	0
<hr/>		
conv2d_3 (Conv2D)	(None, 8, 8, 64)	8256
<hr/>		
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 64)	0
<hr/>		
dropout_1 (Dropout)	(None, 4, 4, 64)	0
<hr/>		
flatten_1 (Flatten)	(None, 1024)	0
<hr/>		
dense_1 (Dense)	(None, 500)	512500
<hr/>		
dropout_2 (Dropout)	(None, 500)	0
<hr/>		
dense_2 (Dense)	(None, 10)	5010
<hr/>		

Total params: 528,054
Trainable params: 528,054
Non-trainable params: 0

Exercício

▷ CNN_Happy_house.ipynb

Tarefa



Código

▷ Cifar10_cnn_full.ipynb

▷ Comparar desempenho MLP vs CNN

1. Load CIFAR -10 Database

```
1 import keras
2 from keras.datasets import cifar10
3
4 # load the pre-shuffled train and test data
5 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

2. Visualize the first 24 training images

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 %matplotlib inline
4
5 fig = plt.figure(figsize=(20,5))
6 for i in range(36):
7     ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks[])
8     ax.imshow(np.squeeze(x_train[i]))
```



3. Rescale the images by dividing
every pixel in every image by 255

```
1 # rescale [0,255] --> [0,1]
2 x_train = x_train.astype('float32')/255
3 x_test = x_test.astype('float32')/255
```

4. Break dataset into Training, Testing, and Validation sets

```
from keras.utils import np_utils

# one-hot encode the labels
num_classes = len(np.unique(y_train))
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# break training set into training and validation sets
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]

# print shape of training set
print('x_train shape:', x_train.shape)

# print number of training, validation, and test images
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_valid.shape[0], 'validation samples')
```

x_train shape: (45000, 32, 32, 3)

45000 train samples

10000 test samples

5000 validation samples

MLP

5. Define the model architecture

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten

# define the model
model = Sequential()
model.add(Flatten(input_shape = x_train.shape[1:]))
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 3072)	0
dense_1 (Dense)	(None, 1000)	3073000
dropout_1 (Dropout)	(None, 1000)	0
dense_2 (Dense)	(None, 512)	512512
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130
Total params: 3,590,642.0		
Trainable params: 3,590,642.0		
Non-trainable params: 0.0		

6. Compile the model

```
1 # compile the model
2 model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
3                 metrics=['accuracy'])
```

7. Train the model

```
1 from keras.callbacks import ModelCheckpoint  
2  
3 # train the model  
4 checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=1,  
5                               save_best_only=True)  
6 hist = model.fit(x_train, y_train, batch_size=32, epochs=5,  
7                   validation_data=(x_valid, y_valid), callbacks=[checkpointer],  
8                   verbose=2, shuffle=True)
```

8. Load the model with the best classification accuracy on the validation set

```
1 # load the weights that yielded the best validation accuracy  
2 model.load_weights('model.weights.best.hdf5')
```

9. Calculate classification accuracy on test set

```
1 # evaluate and print test accuracy
2 score = model.evaluate(x_test, y_test, verbose=0)
3 print('\n', 'Test accuracy:', score[1])
```

CNN

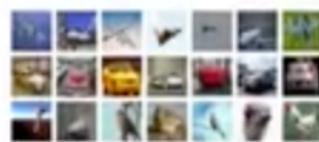
5. Define the model architecture

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
                 input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

	Param #
16)	208
16)	0
12)	2080
)	0
)	8256
)	0
)	0
)	0
dense_1 (Dense)	(None, 500) 512500
dropout_2 (Dropout)	(None, 500) 0
dense_2 (Dense)	(None, 10) 5010
Total params:	528,054.0
Trainable params:	528,054.0
Non-trainable params:	0.0



CIFAR-10 - Object Recognition in Images

Identify the subject of 60,000 labeled images

231 teams · 2 years ago

[Overview](#)[Data](#)[Discussion](#)[Leaderboard](#)[More](#)[Submit Predictions](#)[Overview](#)[Description](#)[Evaluation](#)

CIFAR-10 is an established computer-vision dataset used for object recognition. It is a subset of the [80 million tiny images dataset](#) and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Kaggle is hosting a CIFAR-10 leaderboard for the machine learning community to use for fun and practice. You can see how your approach compares to the latest research methods on Rodrigo Benenson's [classification results page](#).



Exercício

► Construa um modelo com a seguinte arquitetura, e escolha os parâmetros e função de ativação para cada camada

- conv2d
- maxpool
- conv2d
- conv2d
- maxpool
- flatten
- dense
- output

Exercício

- ▷ Verificar melhores acurácia no cifar 10
- ▷ Buscar arquitetura melhor na internet.