REDE NEURAL CONVOLUCIONAL

Prof. Valmir Macário Filho

DC - UFRPE

IMAGENS

VISÃO GERAL

- Imagens
- Redes Neurais Completamente Conectadas
- Redes Neurais Convolucionais
- Exemplos de CNNs

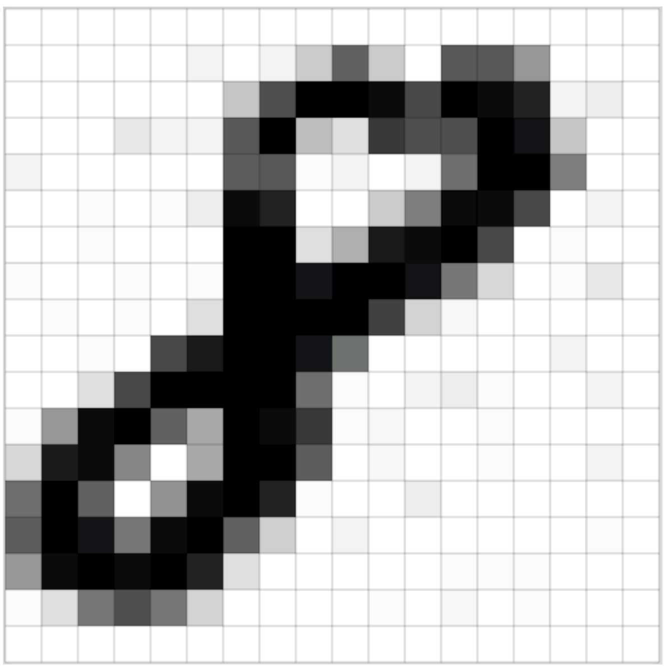
IMAGENS SÃO NÚMEROS

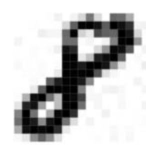
```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
 [  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
 [  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
 [  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
 [  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
 [  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
 [  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
 [  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
 [  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
 [  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
 [  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
 [  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
 [ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
 [ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]]
```

Toda imagem é uma matriz de valores de pixels

https://www.commonlounge.com/discussion/244616b76d3d40f88e8f12103a22743d

IMAGENS SÃO NÚMEROS



 =

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Fonte: https://medium.com/@ageitgey/

RGB

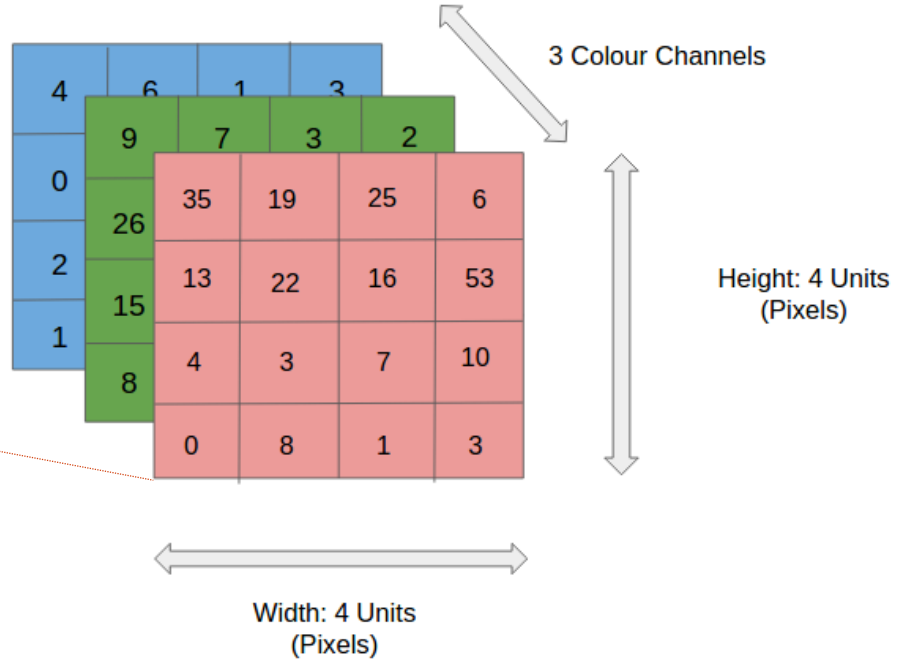
Red

Green

Blue

IMAGENS RGB

- Toda imagem RGB é uma estrutura tridimensional.

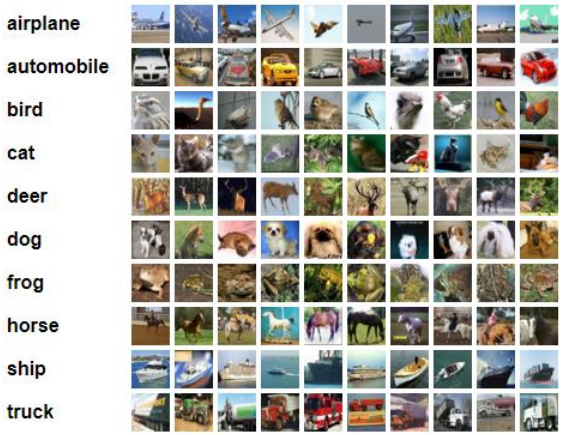


DATASETS DE IMAGENS

- CIFAR-100
- MNIST
- ImageNet

DATASETS: CIFAR

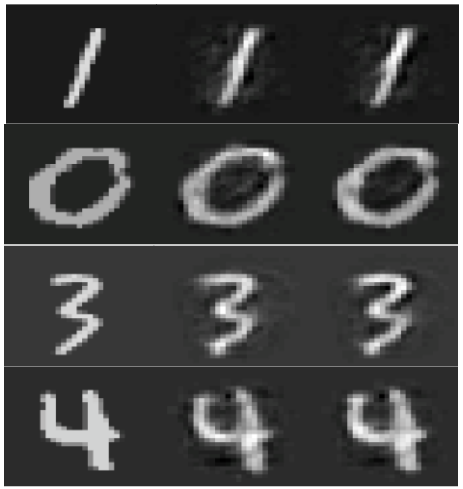
- CIFAR-10
  - 60K imagens RGB 32x32
  - Total de 10 classes
    - 6000 imagens por classe.
  - Treinamento/teste: 50K/10K.

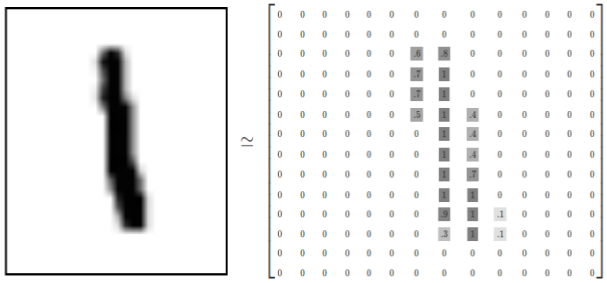


- CIFAR-100
  - similar ao CIFAR-10, mas com100 classes com 600 imagens por classe.

DATASETS: MNIST (*HANDWRITTEN DIGITS DATASET*)

- Conjunto de dados com 70k exemplos
  - Treinamento: 60k
  - Teste: 10k
- Cada exemplo: 28x28 pixels
- Melhores desempenhos:
  - classificador linear: 12% error
  - (Gaussian) SVM 1.4% error
  - ConvNets <1% error

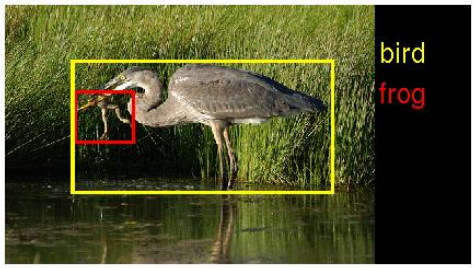




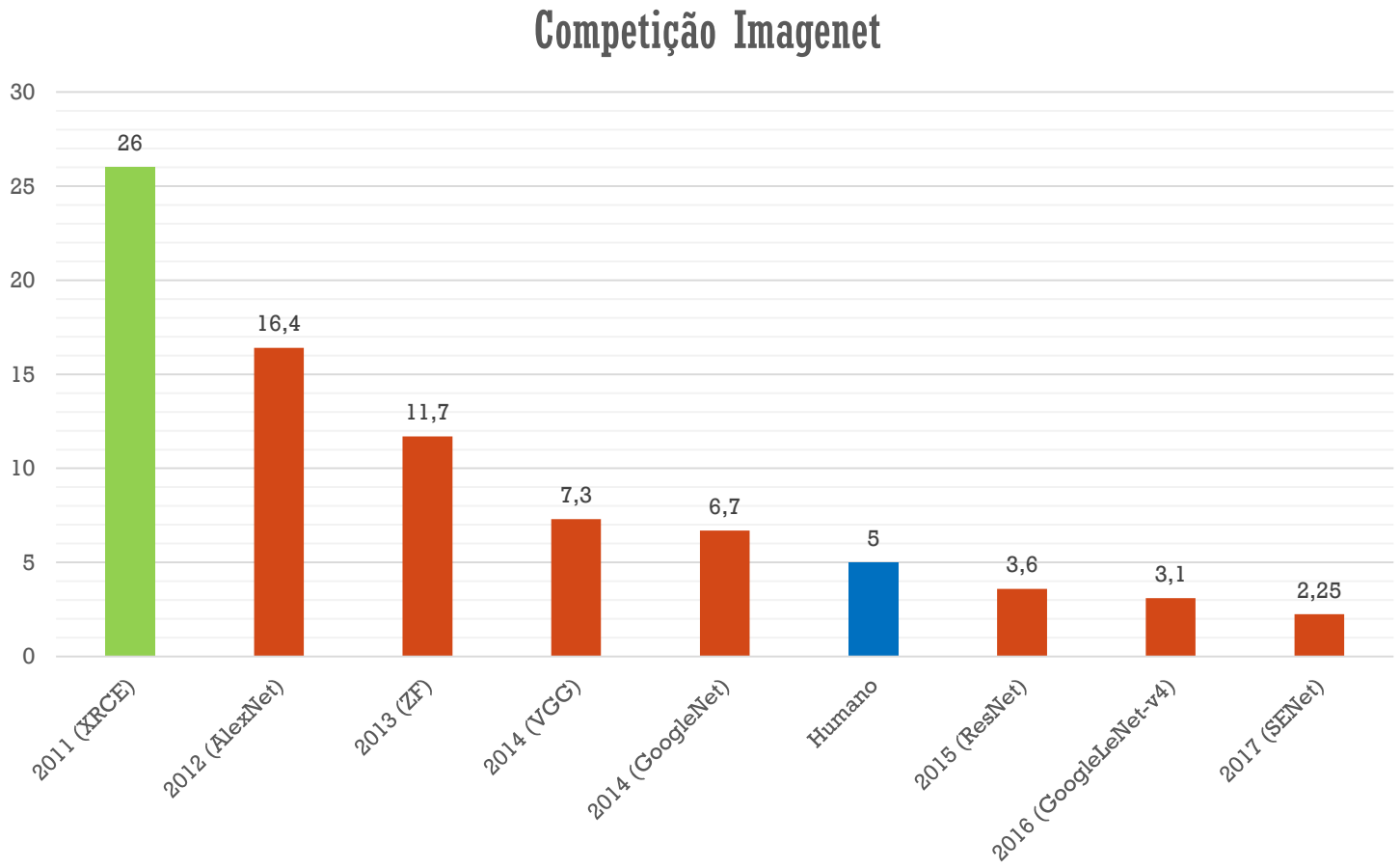
[http://yann.lecun.com/exdb/mnist/]

DATASETS: IMAGENET

- Tamanho médio das imagens: 482x415 pixels.
- 14M+ de imagens no total
- 1M+ de imagens com "bounding boxes" anotados
- Competições:
  - ILSVRC
  - ImageNet Object Detection Challenge (Kaggle)



IMAGENET


Competição Imagenet
30
25
20
15
10
5
0
26
16,4
11,7
7,3
6,7
5
3,6
3,1
2,25
2011 (XRCE)
2012 (AlexNet)
2013 (ZF)
2014 (VGG)
2014 (GoogleNet)
Humano
2015 (ResNet)
2016 (GoogleLeNet-v4)
2017 (SENet)

REDES NEURAIS CONVOLUCIONAIS

(CONVOLUTIONAL NEURAL NETWORKS, CONVNET, CNN)

MLPS VS CNNS

MLPs

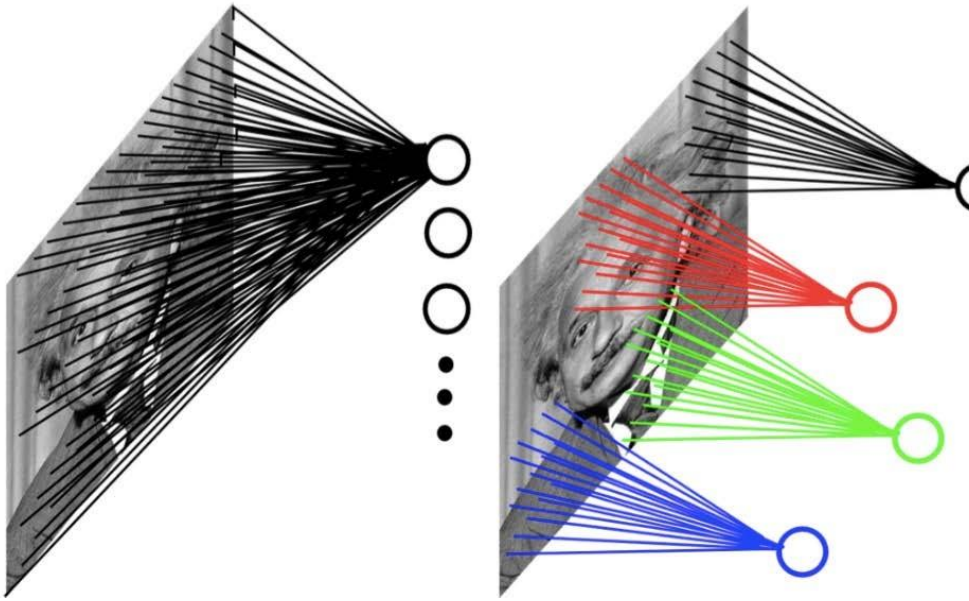
- Utiliza camadas completamente conectadas
- Aceita apenas vetor como entrada

CNNs

- Também pode usar camadas esparsamente conectadas
- Também aceita matriz como entrada

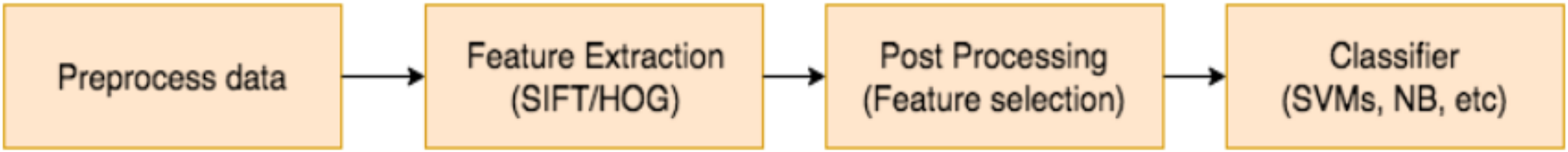
Rede Neural Convolucional

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



Exemplo: 200x200 imagem
a) MLP: 40,000 hidden units => 1.6 bilhões de parâmetros
b) CNN: 5x5 kernel, 100 campos receptivos => 2,500 parâmetros

ERA SEM DEEP LEARNING



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

COM DEEP LEARNING

- CNN´s foram inspiradas pela sensibilidade local e orientação seletiva do cérebro

- Projetaram uma rede neural que implicitamente extrai características relevantes da entrada

- CNN é uma rede de aprendizagem supervisionada que extrai propriedades topológicas a partir da sua entrada

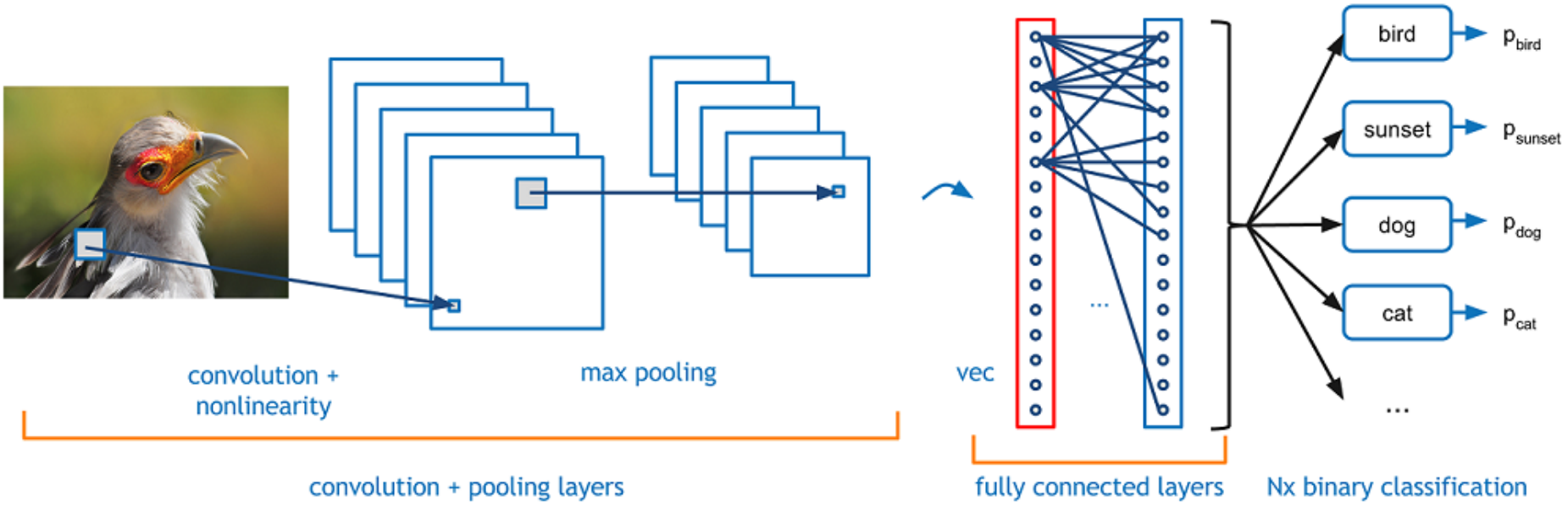
- Redes Convolucionais são um tipo especial de MLP

- Como qualquer outra rede neural a arquitetura é treinada com alguma variação do algoritmo back-propagation

- Capazes de reconhecer padrões com muita variabilidade (caracteres escritos a mão)

- Desenvolvidas para dados de duas dimensões, como imagens e vídeos

Convolutional Neural Network (CNN)



CAMADAS DA REDE CONVOLUCIONAL

Entrada: 32x32x3 -> 32 pixels de altura, 32 de largura e 3 camadas de RBG

Convolução: Camada que calculara a saída dos neurônios conectados a regiões locais da entrada.

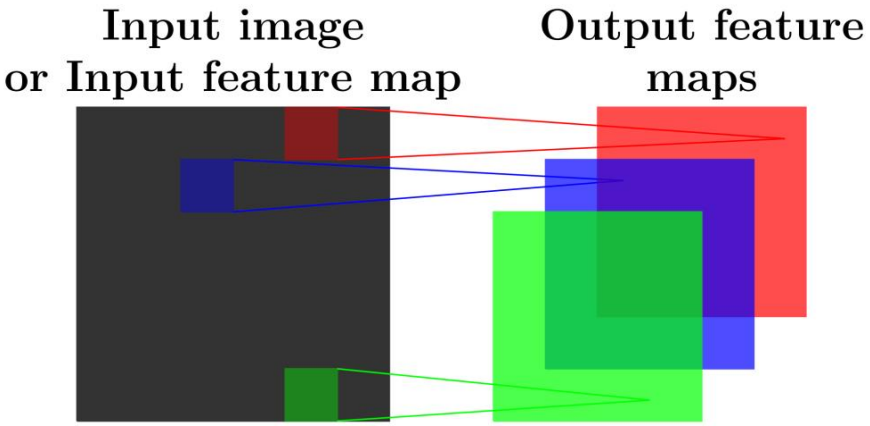
RELU: Função de ativação ponto-a-ponto como por exemplo max, min, avg, etc.

POOL: Cadamada de Downsampling

Classificador: MLP, SVM, Softmax, etc

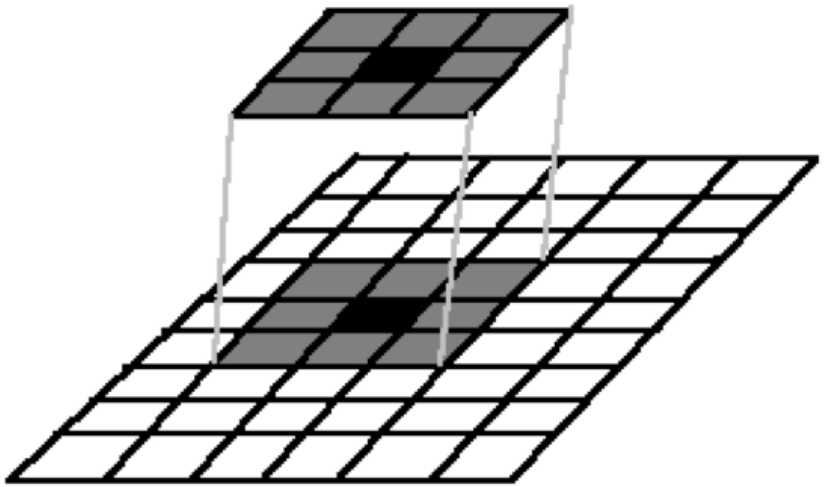
CAMADA DE CONVOLUÇÃO

- É a mais importante operação de uma CNN
- Atuam como extratores de características dos dados de entrada
- Detectam bordas, cantos, pontos de extremidade, etc. usando filtros



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	176	218	232	249

CONVOLUÇÃO

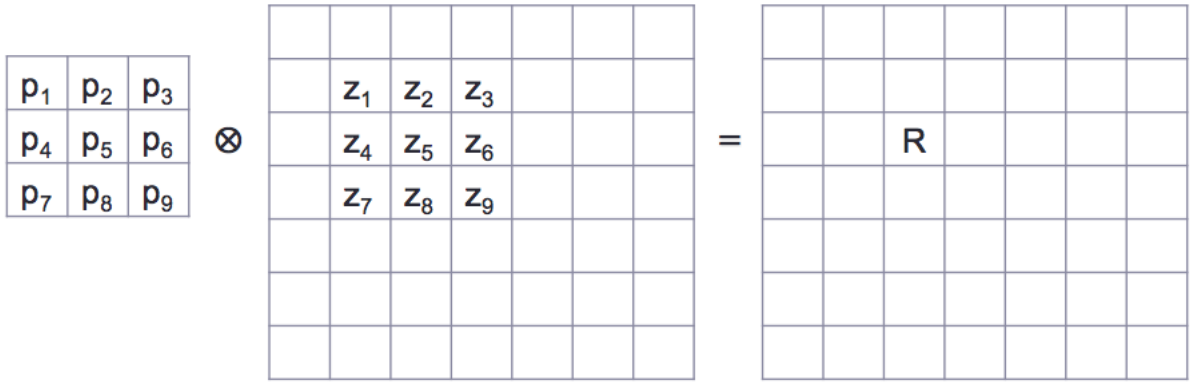
- Seja a máscara 3x3

W1	W2	W3
W4	W5	W6
W7	W8	W9

- e sejam z1, z2, ..., z9 a cor dos pixels sob a máscara
- O novo tom do pixel central será dado por
  - R = w1z1 + w2z2 + ... + w9z9

FILTRAGEM

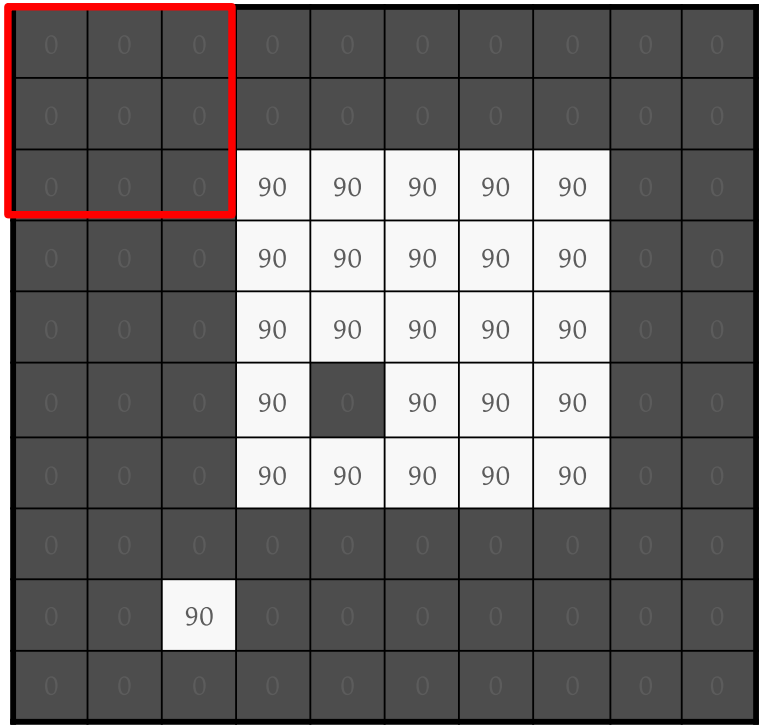
$$R = p_1 z_1 + p_2 z_2 + \cdots + p_n z_n$$



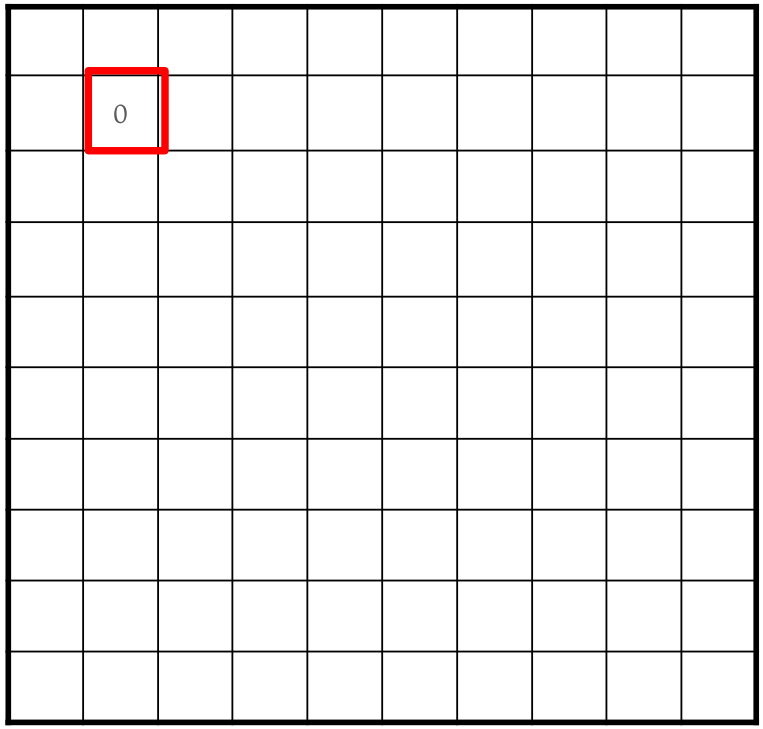
FILTRAGEM NO DOMÍNIO ESPACIAL

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

$F[x, y]$



$G[x, y]$

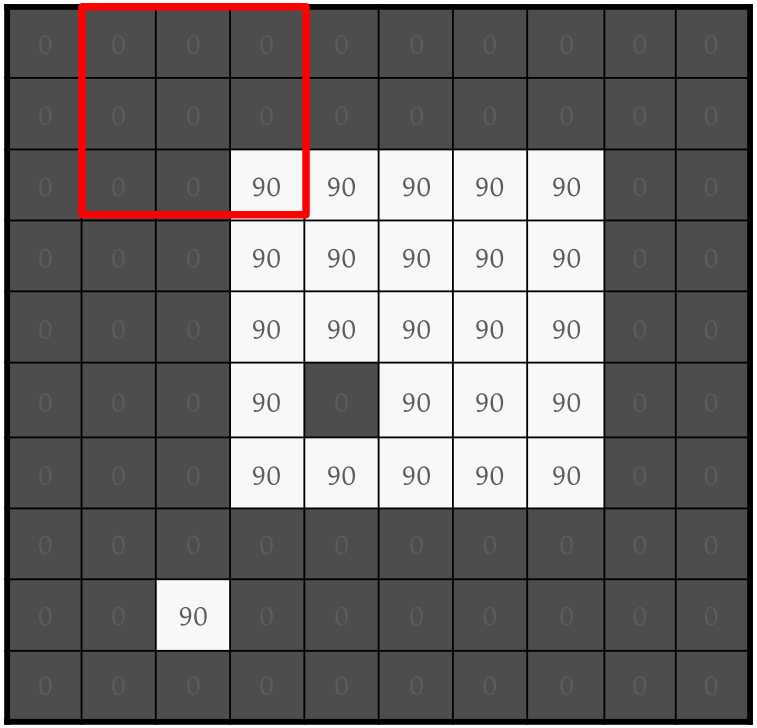


Source: S. Seitz

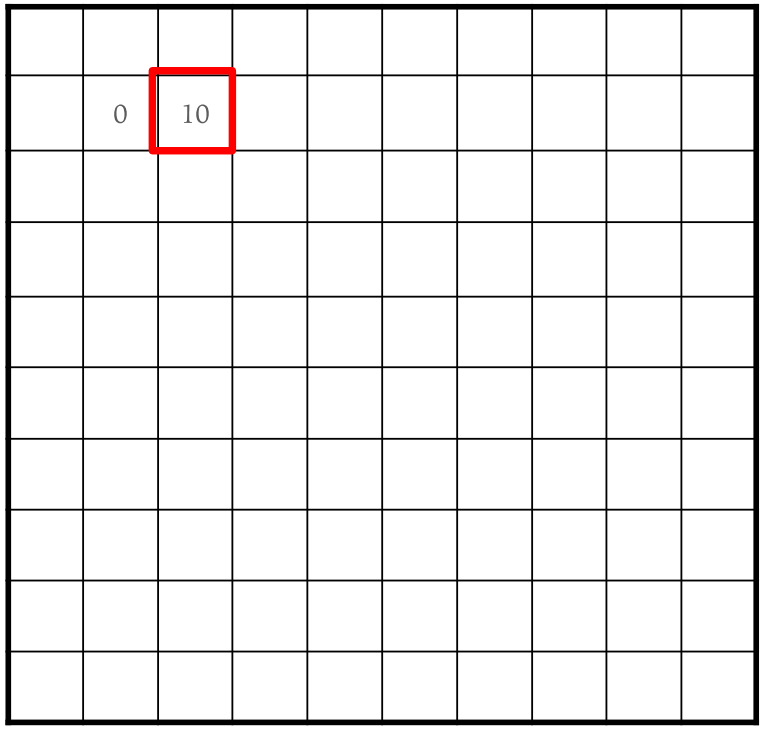
FILTRAGEM NO DOMÍNIO ESPACIAL

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

$F[x, y]$



$G[x, y]$

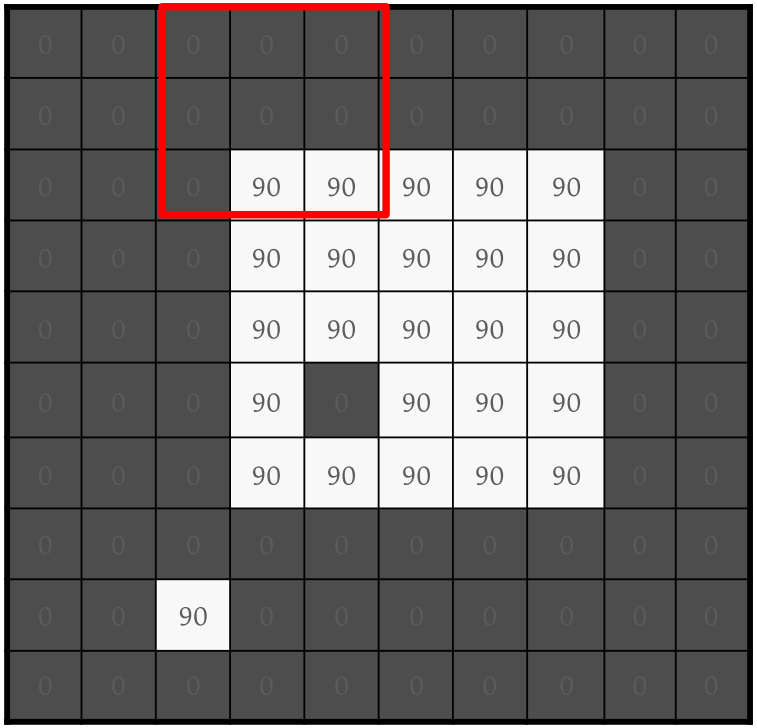


Source: S. Seitz

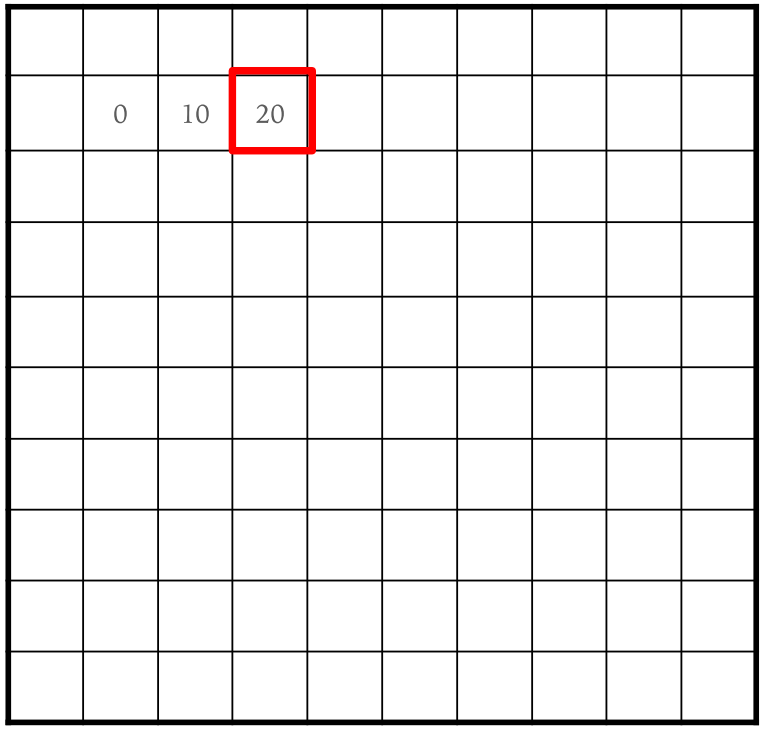
FILTRAGEM NO DOMÍNIO ESPACIAL

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

$F[x, y]$



$G[x, y]$



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					

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

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

$F[x, y]$

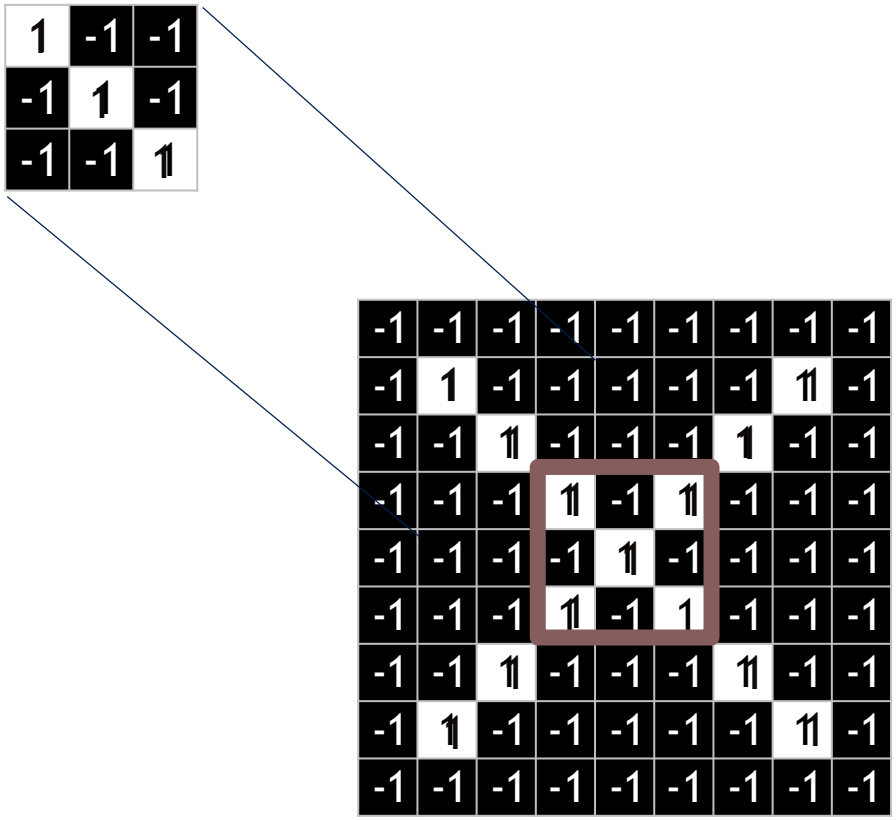
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	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	

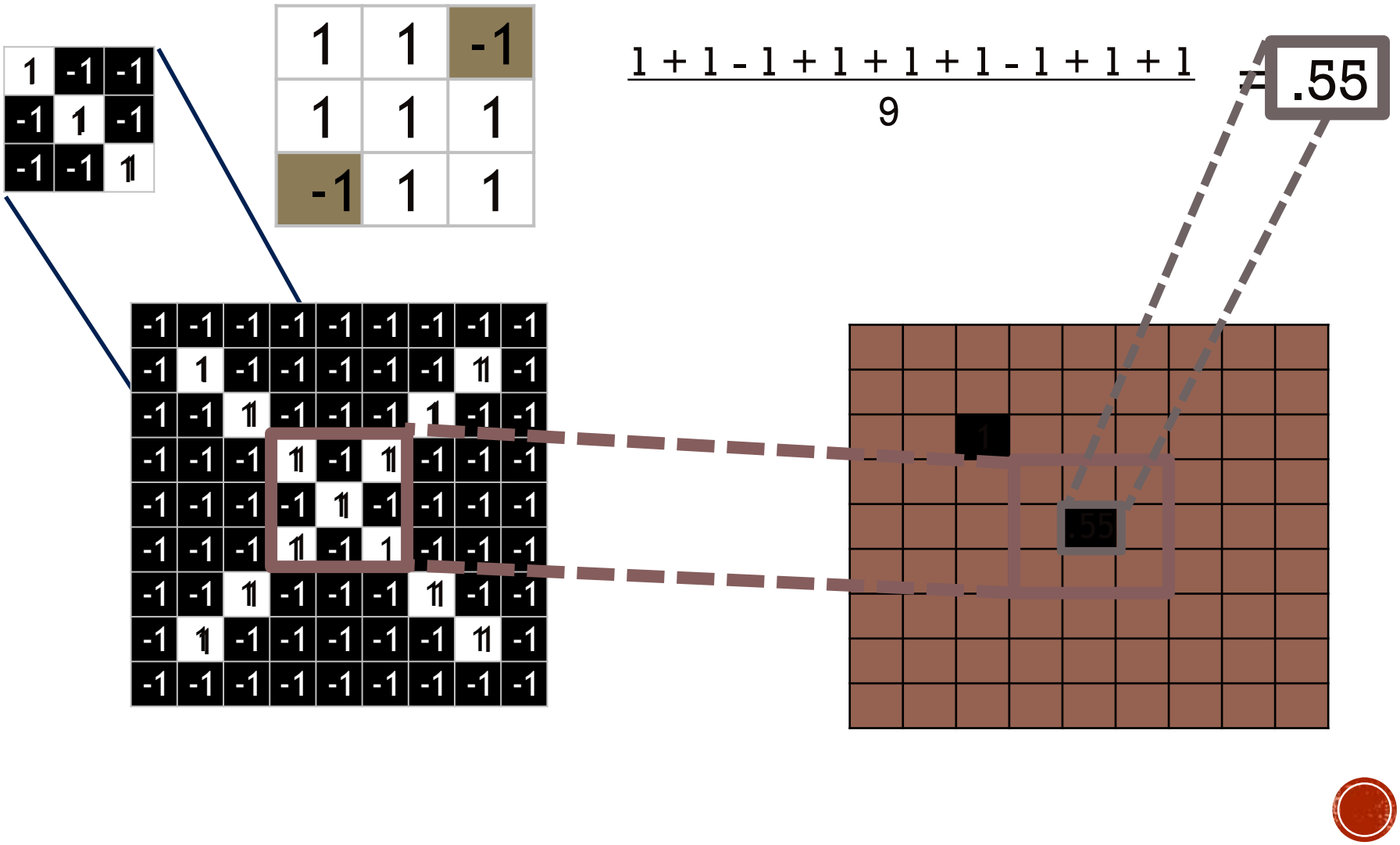
Source: S. Seitz

A MATEMÁTICA ATRÁS DA COMPARAÇÃO



1	1	-1
1	1	1
-1	1	1

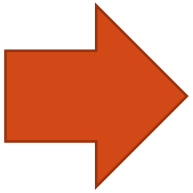
A MATEMÁTICA ATRÁS DA COMPARAÇÃO



A MATEMÁTICA ATRÁS DA COMPARAÇÃO

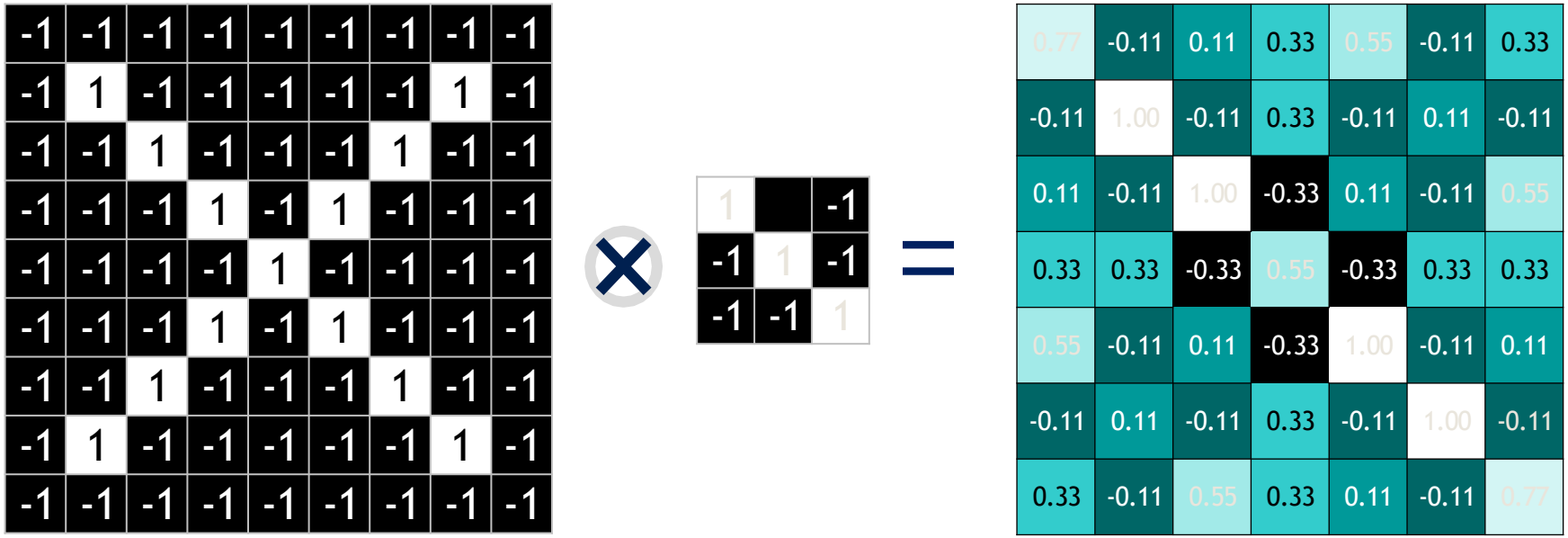
1	-1	-1
-1	1	-1
-1	-1	1

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



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

A MATEMÁTICA ATRÁS DA COMPARAÇÃO



OPERAÇÃO DE CONVOLUÇÃO - EXEMPLO

- Um filtro comumente aprendido em uma CNN (nas CONVs iniciais) é o detector de arestas.

	0	1	0	
	1	-4	1	
	0	1	0	

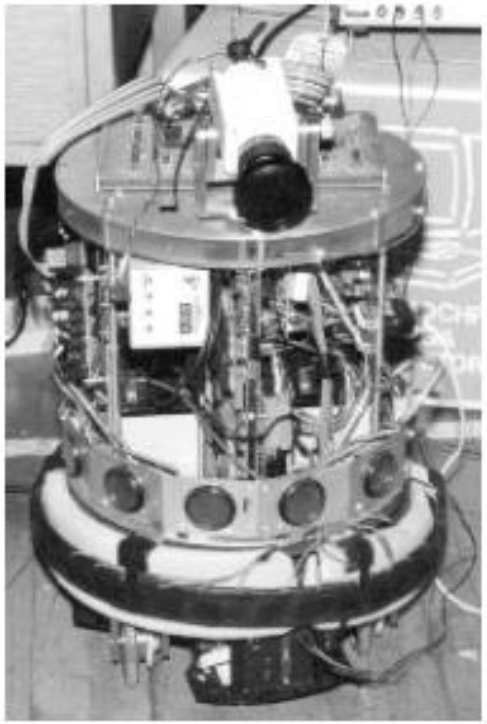
Fonte: https://docs.gimp.org/en/plug-in-convmatrix.html'

FILTRAGEM LINEAR

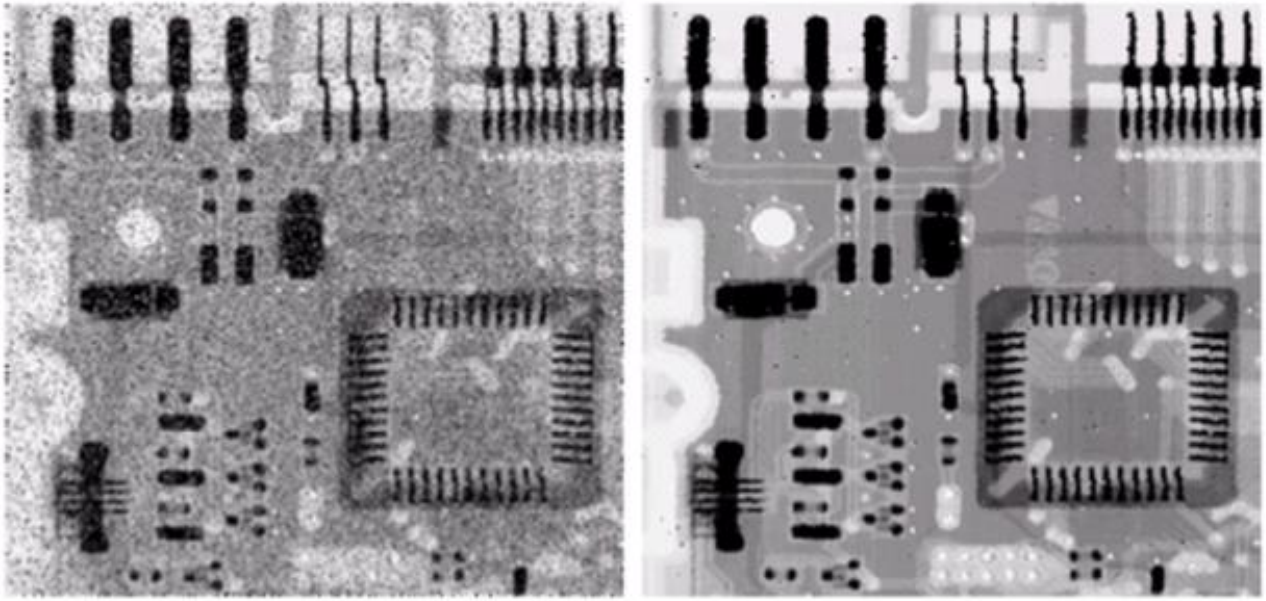
- Filtro de média ponderada
- Mais importância para pixels mais próximos do centro da máscara
  - Menos borramento

Exemplo: máscara 3 x 3

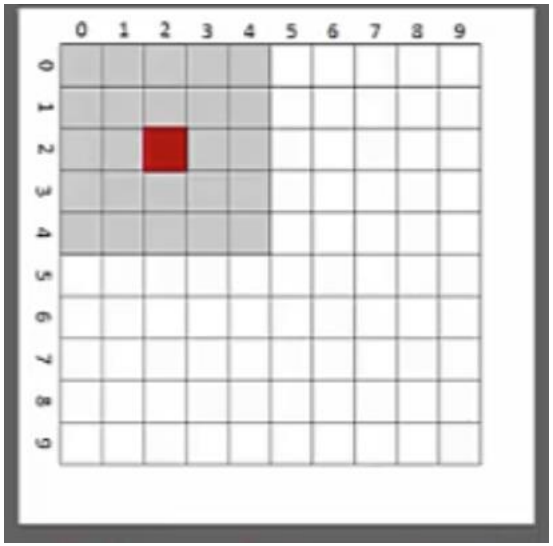
$$\frac{1}{100} \times \begin{array}{|c|c|c|} \hline 8 & 12 & 8 \\ \hline 12 & 20 & 12 \\ \hline 8 & 12 & 8 \\ \hline \end{array}$$



FILTROS LINEARES – SUAVIZAÇÃO

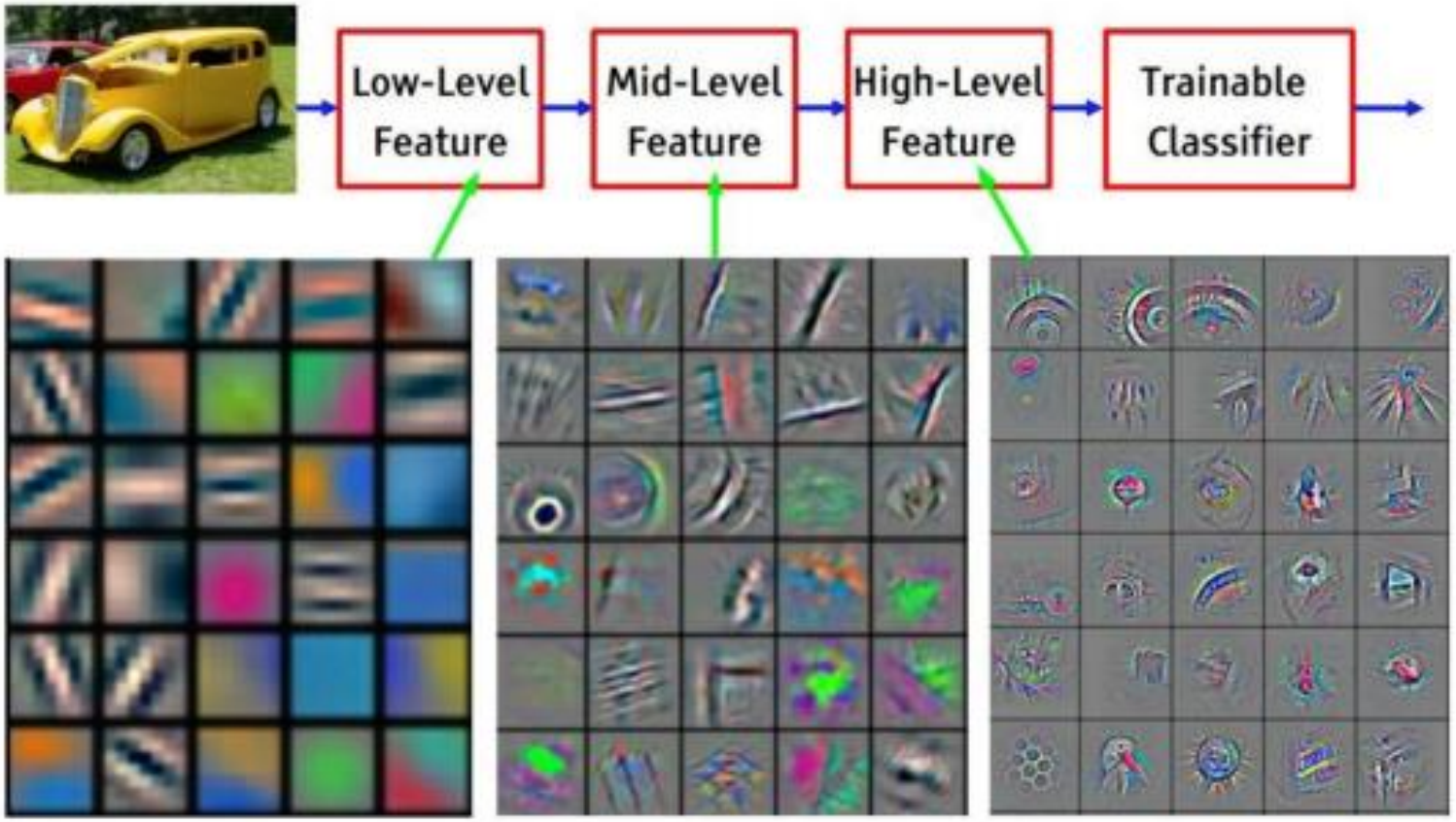


SUAVIZAÇÃO



$$Kernel\_ = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

O QUE É DEEP LEARNING

- 

UM POUCO DE HISTÓRIA..

Hubel & Wiesel,

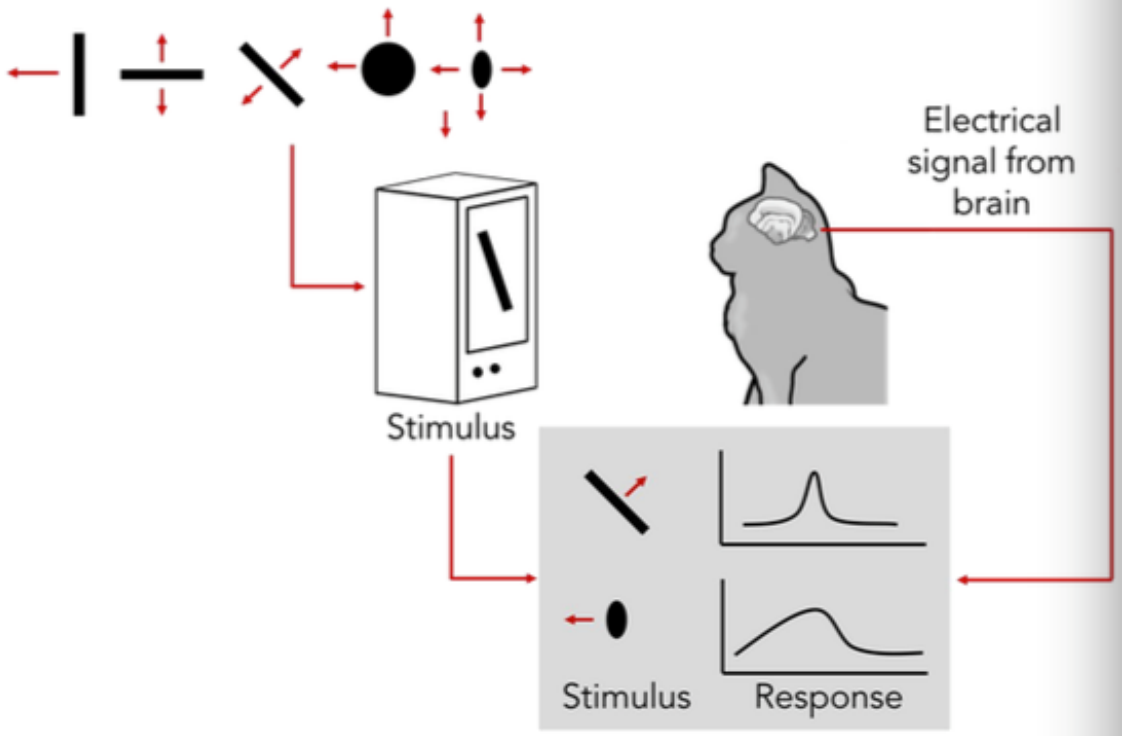
1959

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...



ORGANIZAÇÃO HIERÁRQUICA

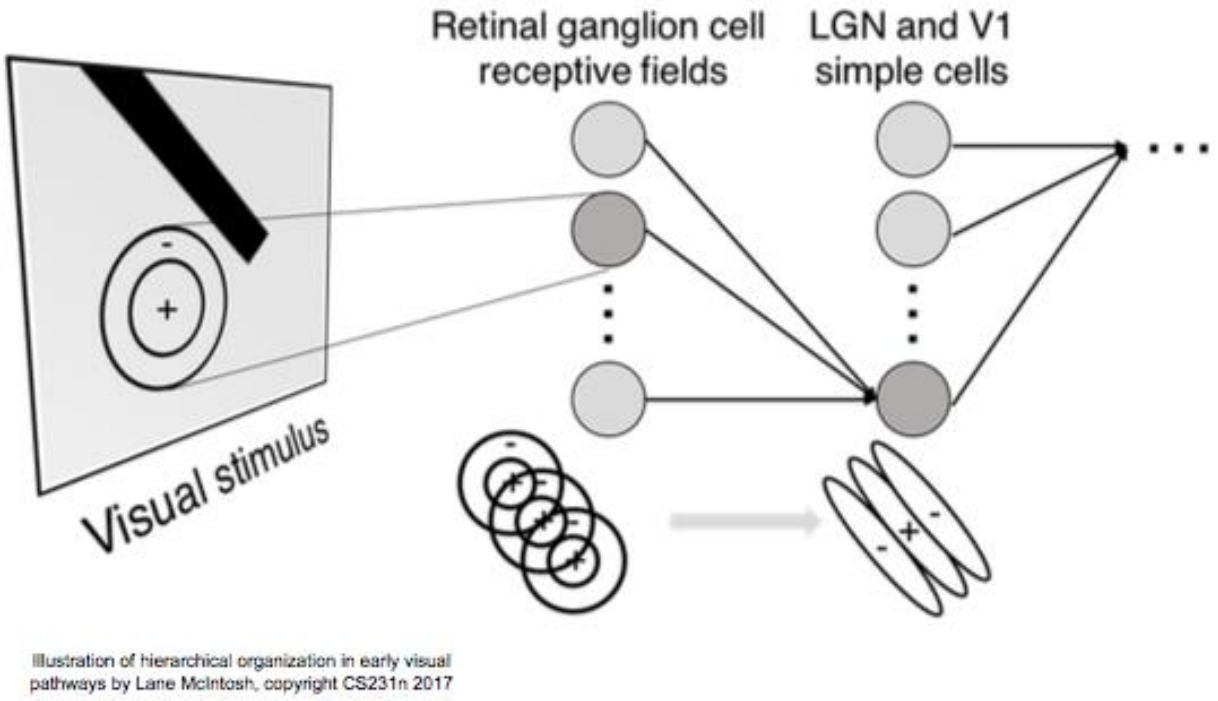


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

No response

Response (end point)

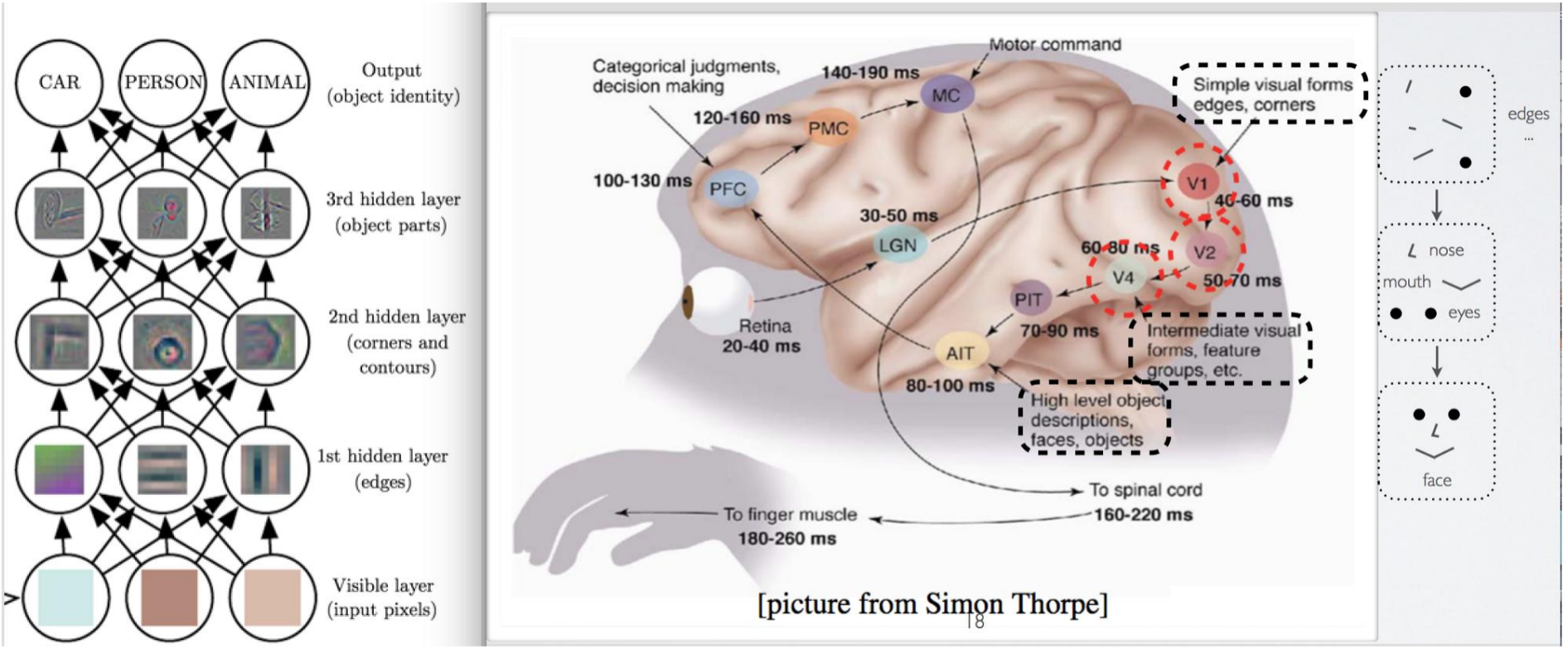
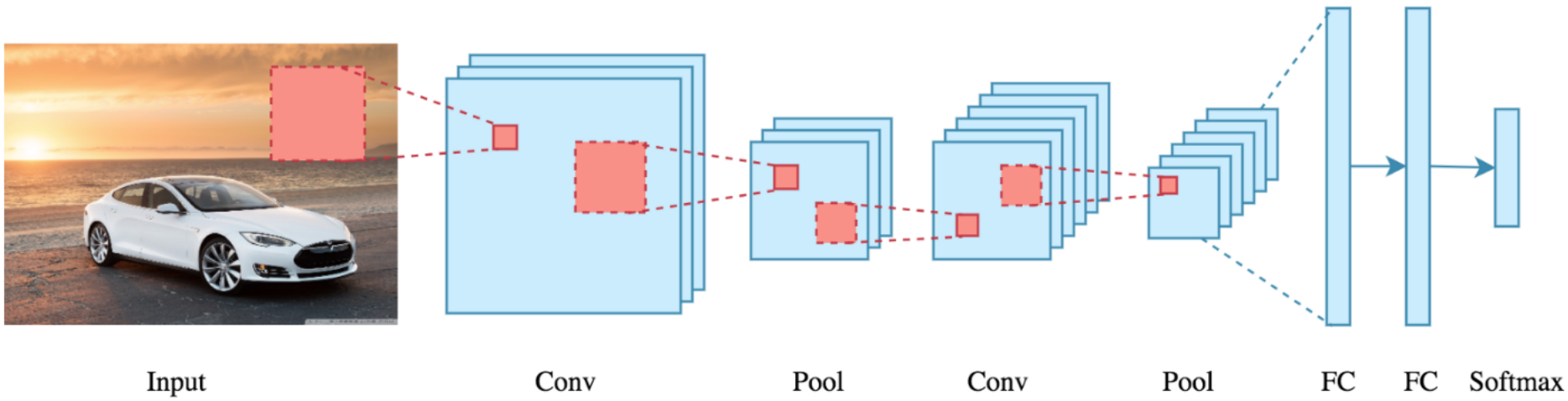


Fig 1. Image Courtesy: Hugo Larochelle Slides.

CNN

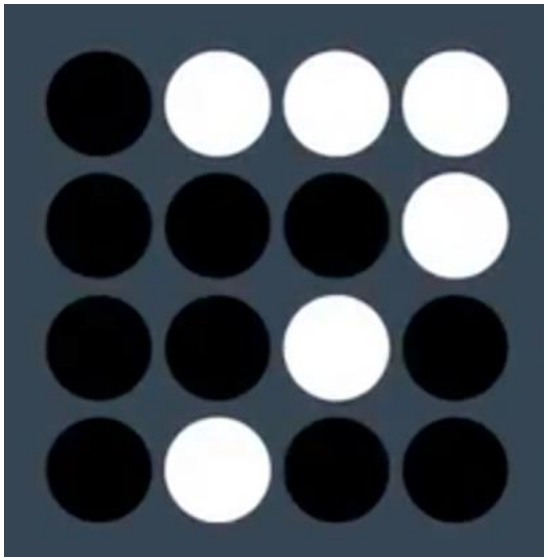


TIPOS DE CAMADAS

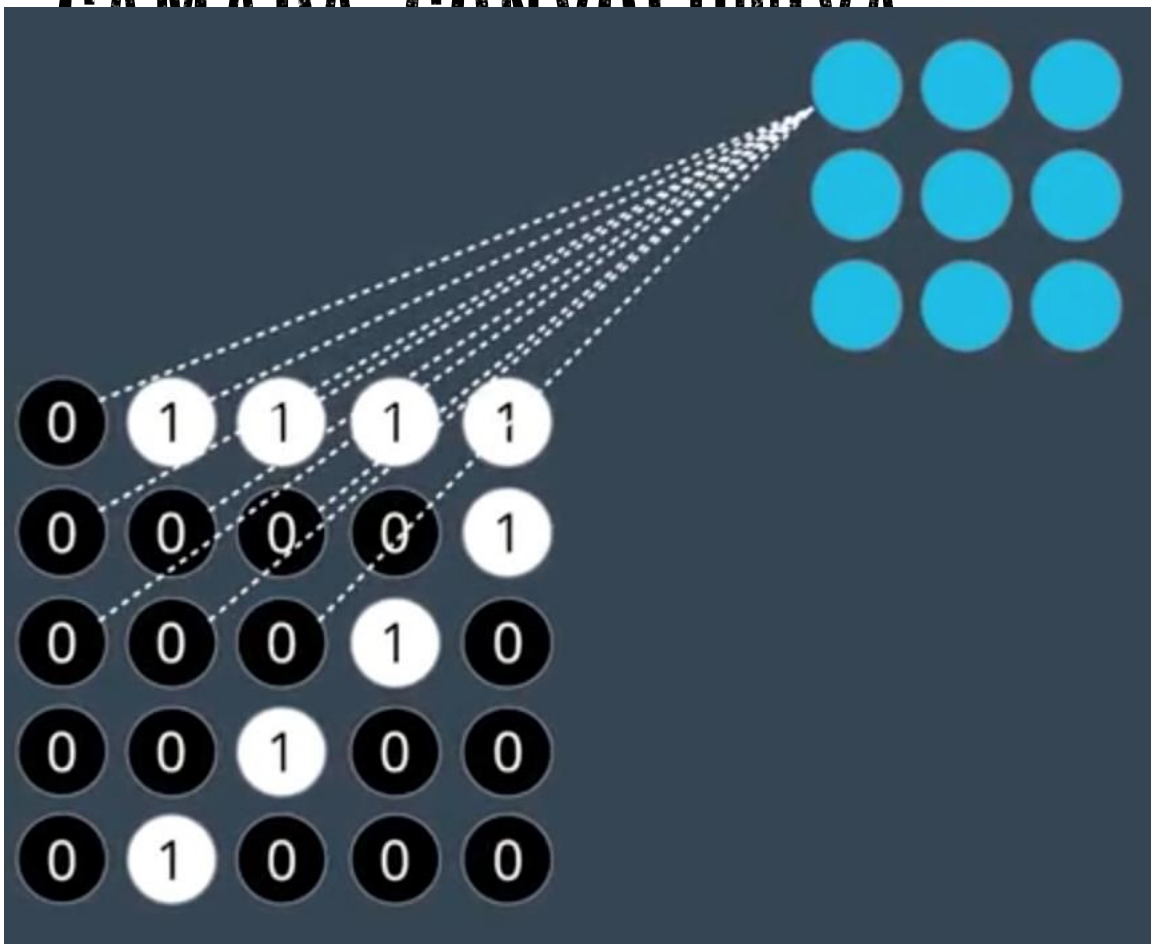
- Convolutiva (CONV)
- Pooling (POOL)
- Fully-conected (FC)
- Dropout(DO)
- Softmax

CAMADA CONVOLUTIVA

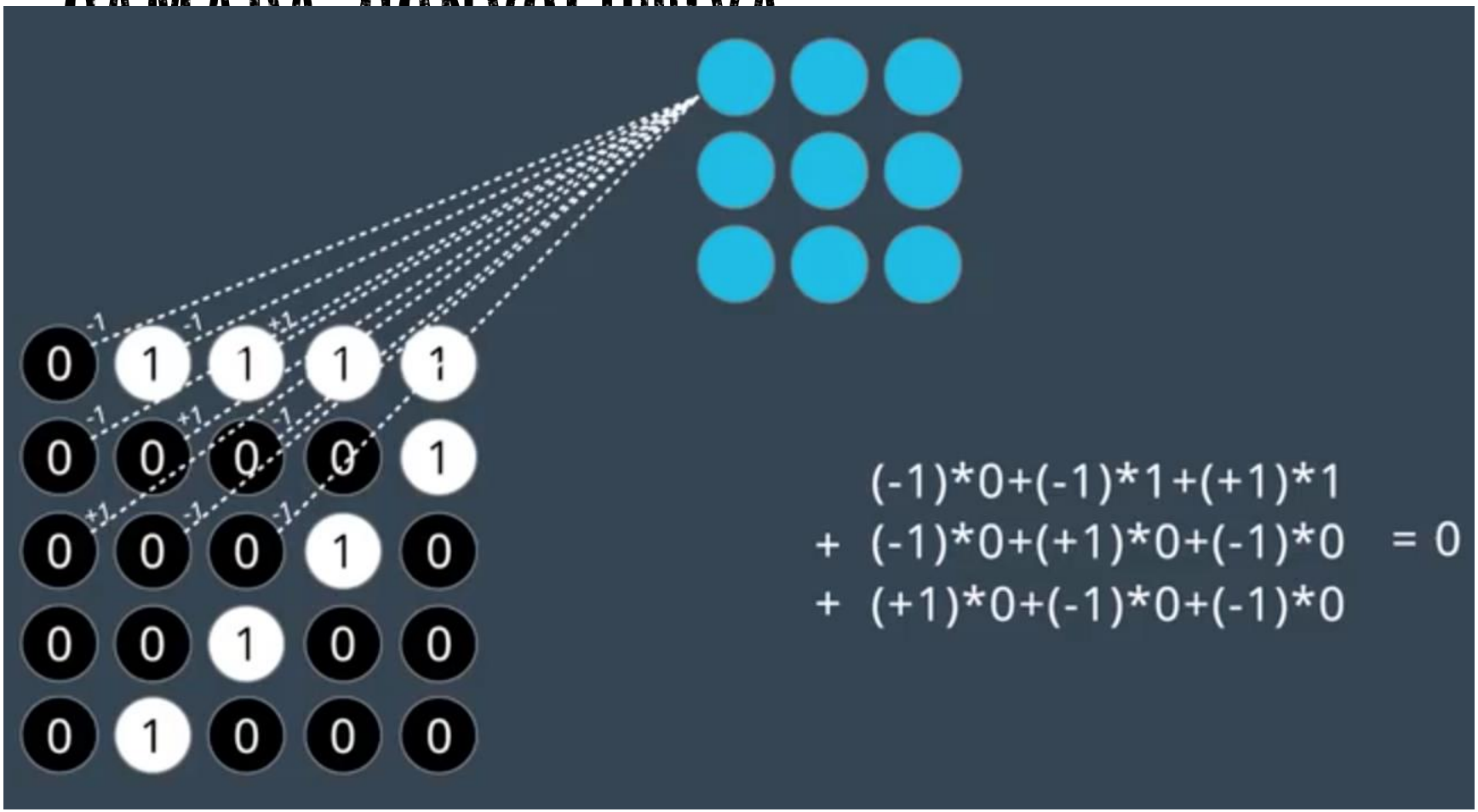
CAMADA CONVOLUTIVA



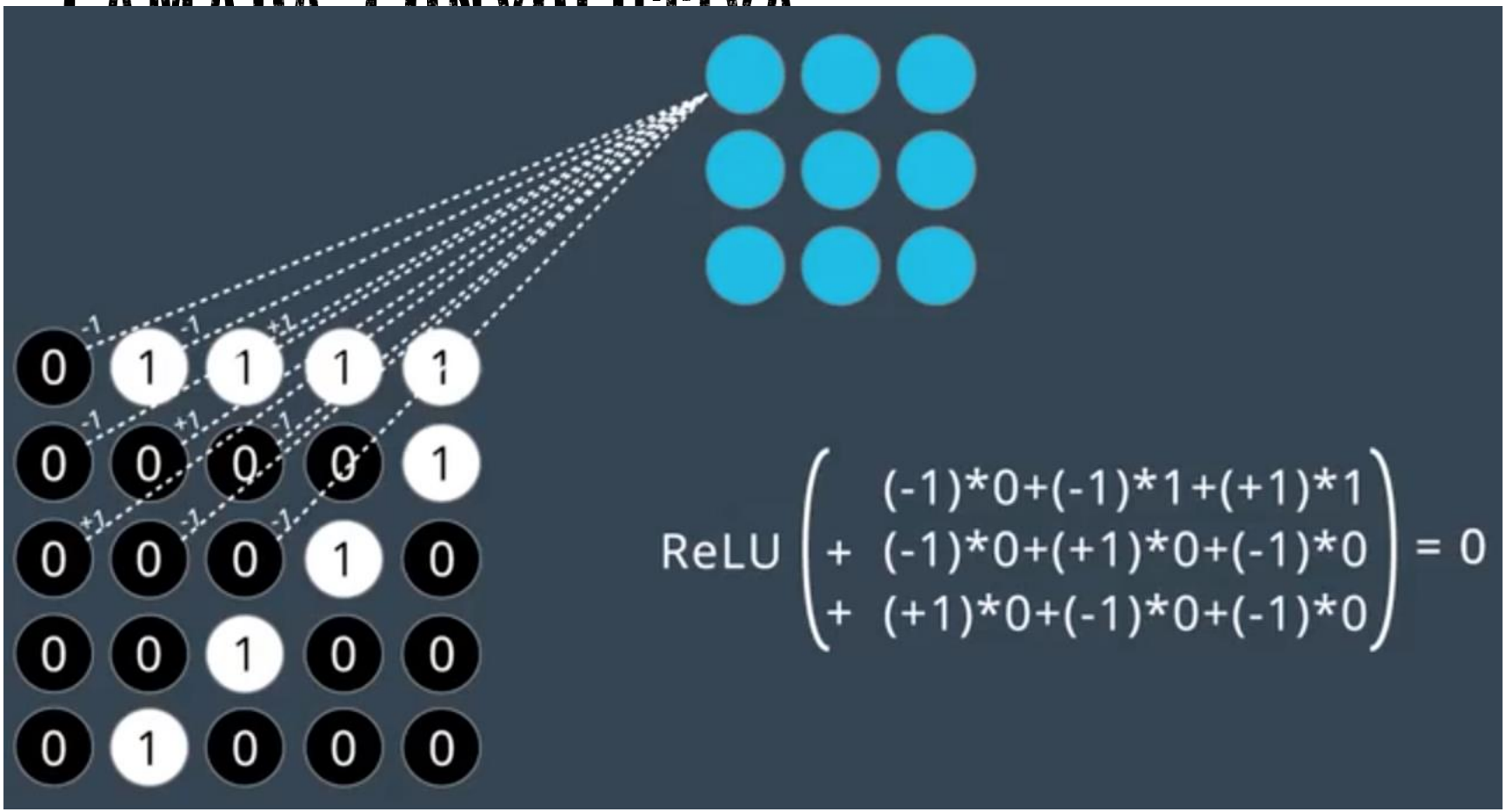
CAMADA CONVOLUCIONAL



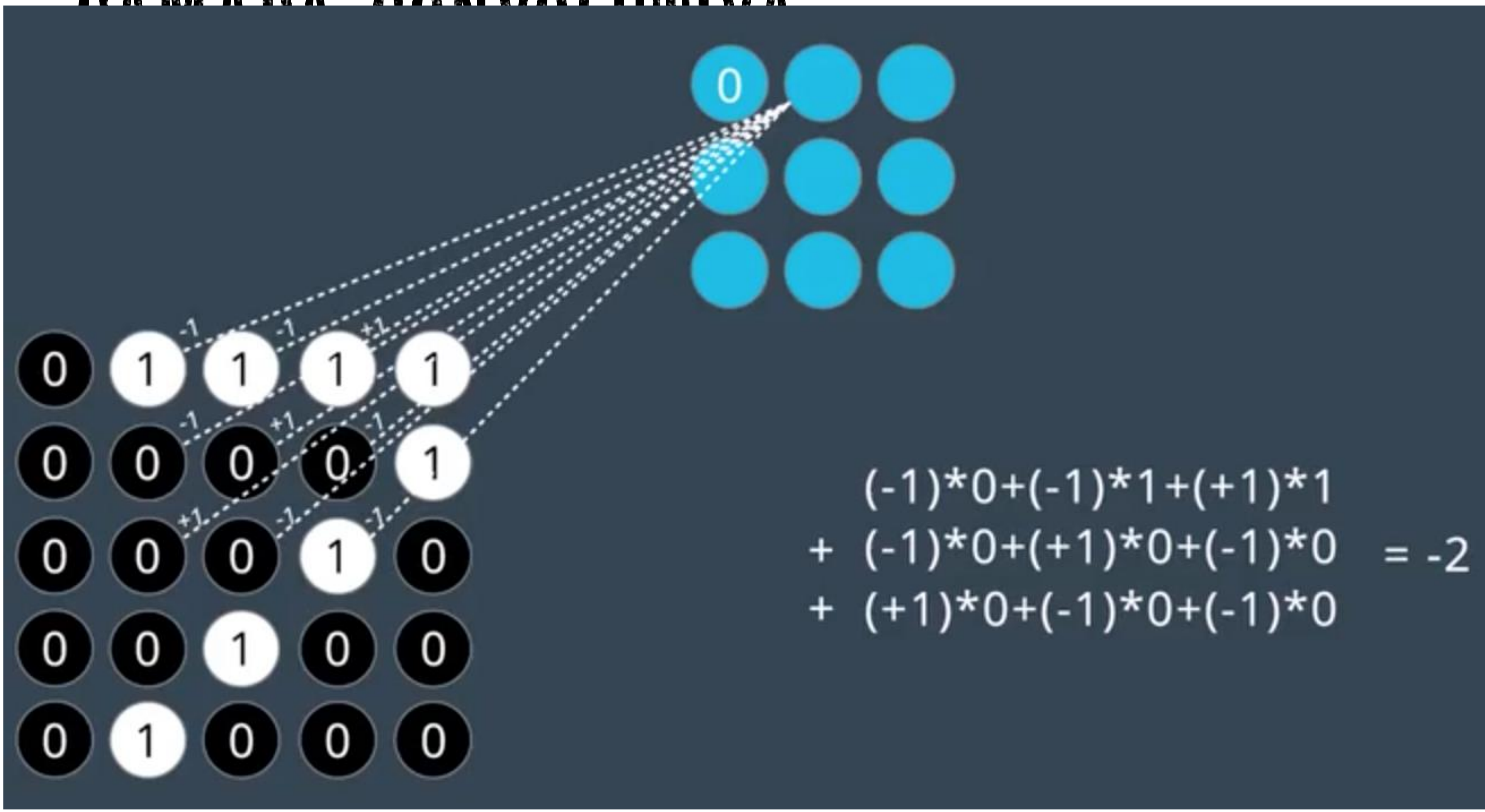
CAMADA CONVOLUCIONAL



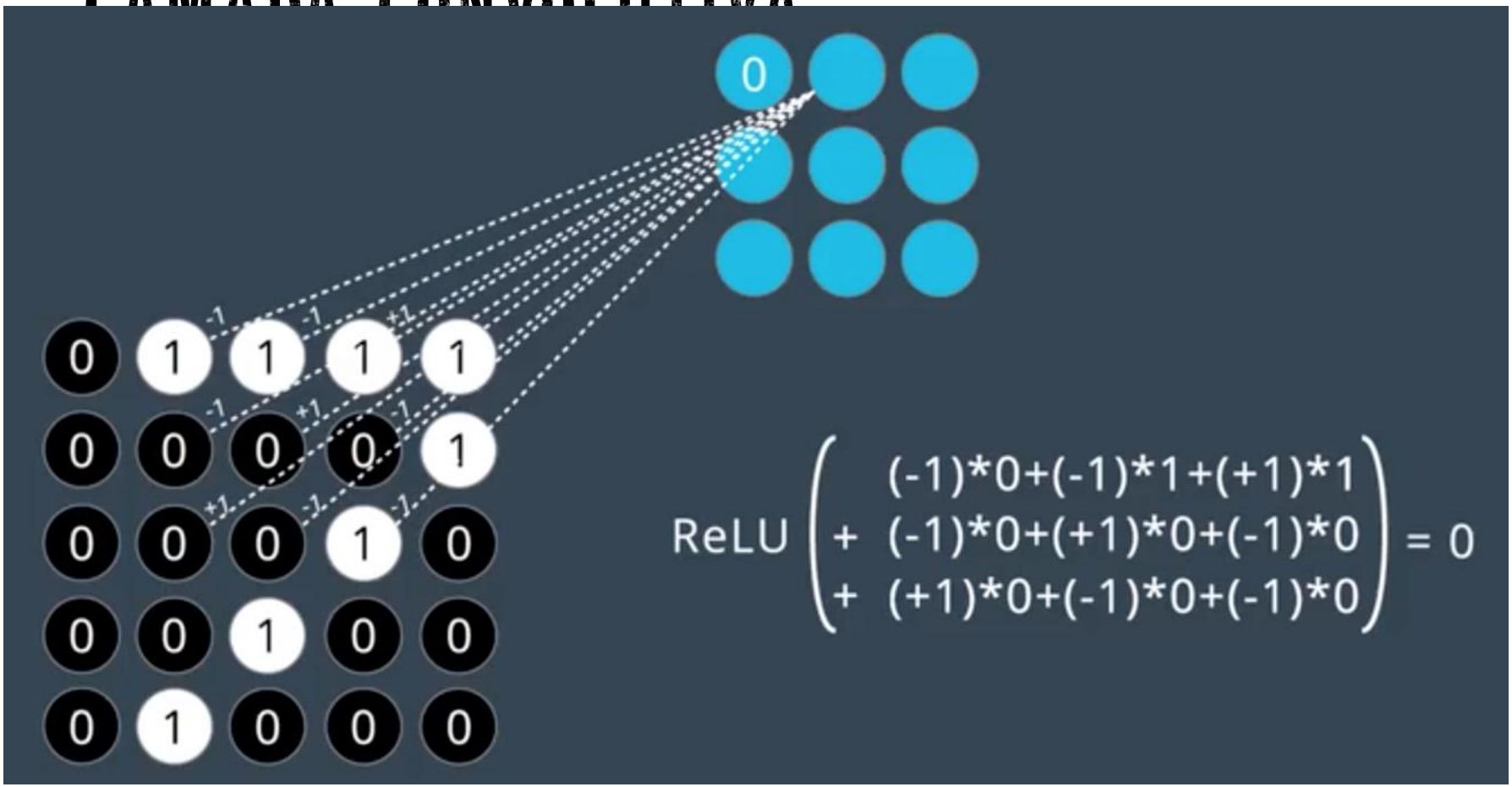
CAMADA CONVOLUTIVA

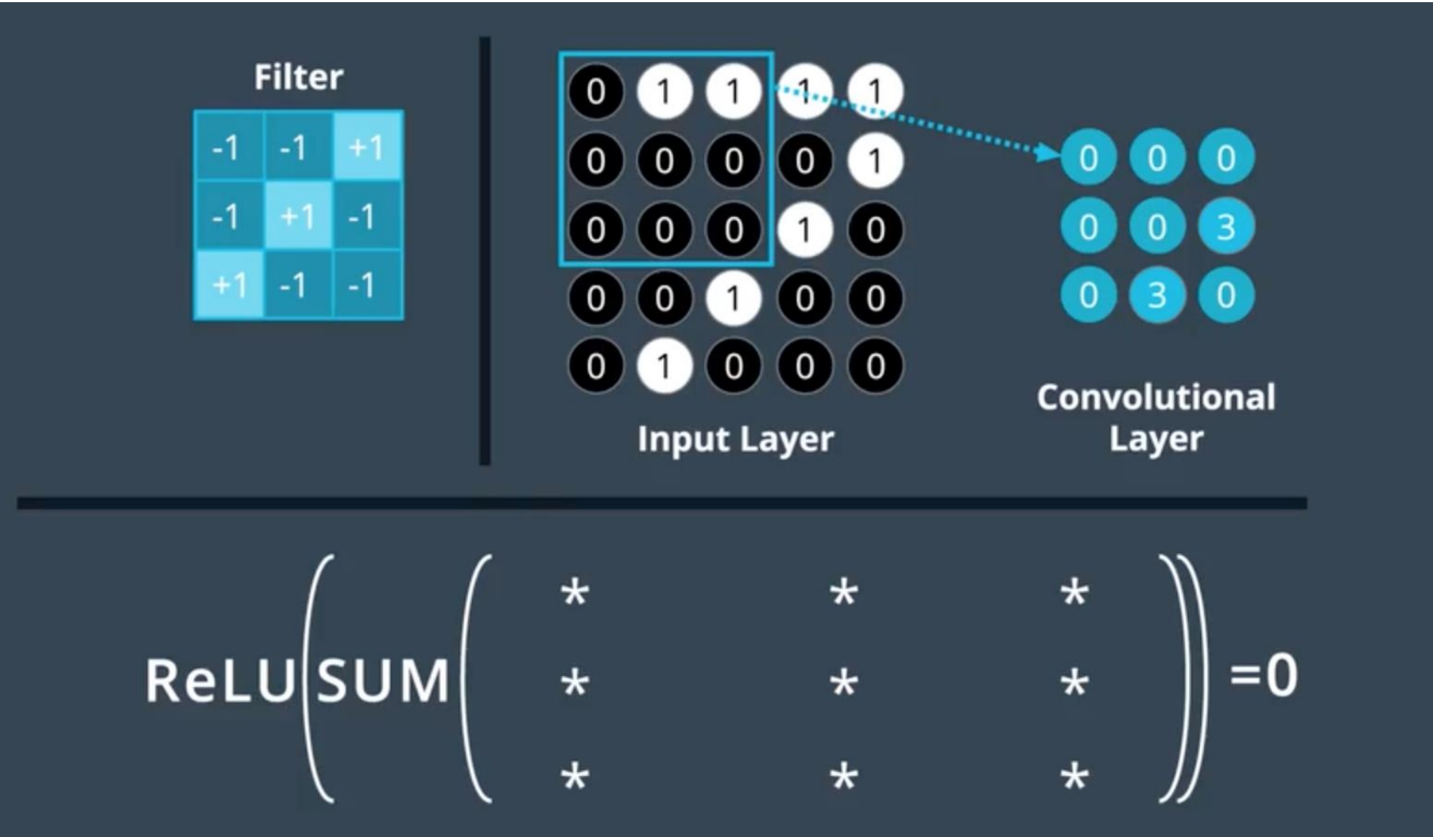


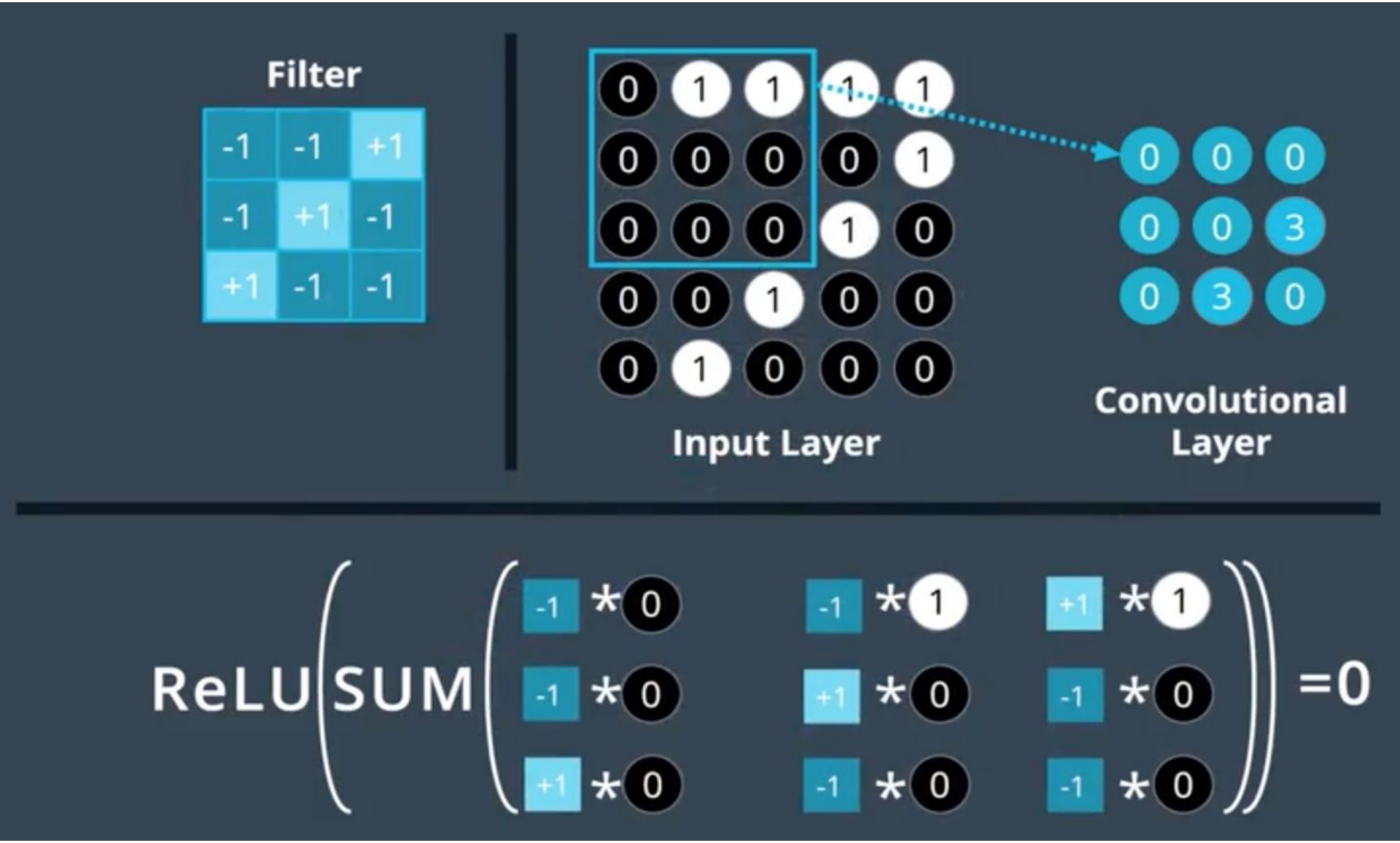
CAMADA CONVOLUTIVA

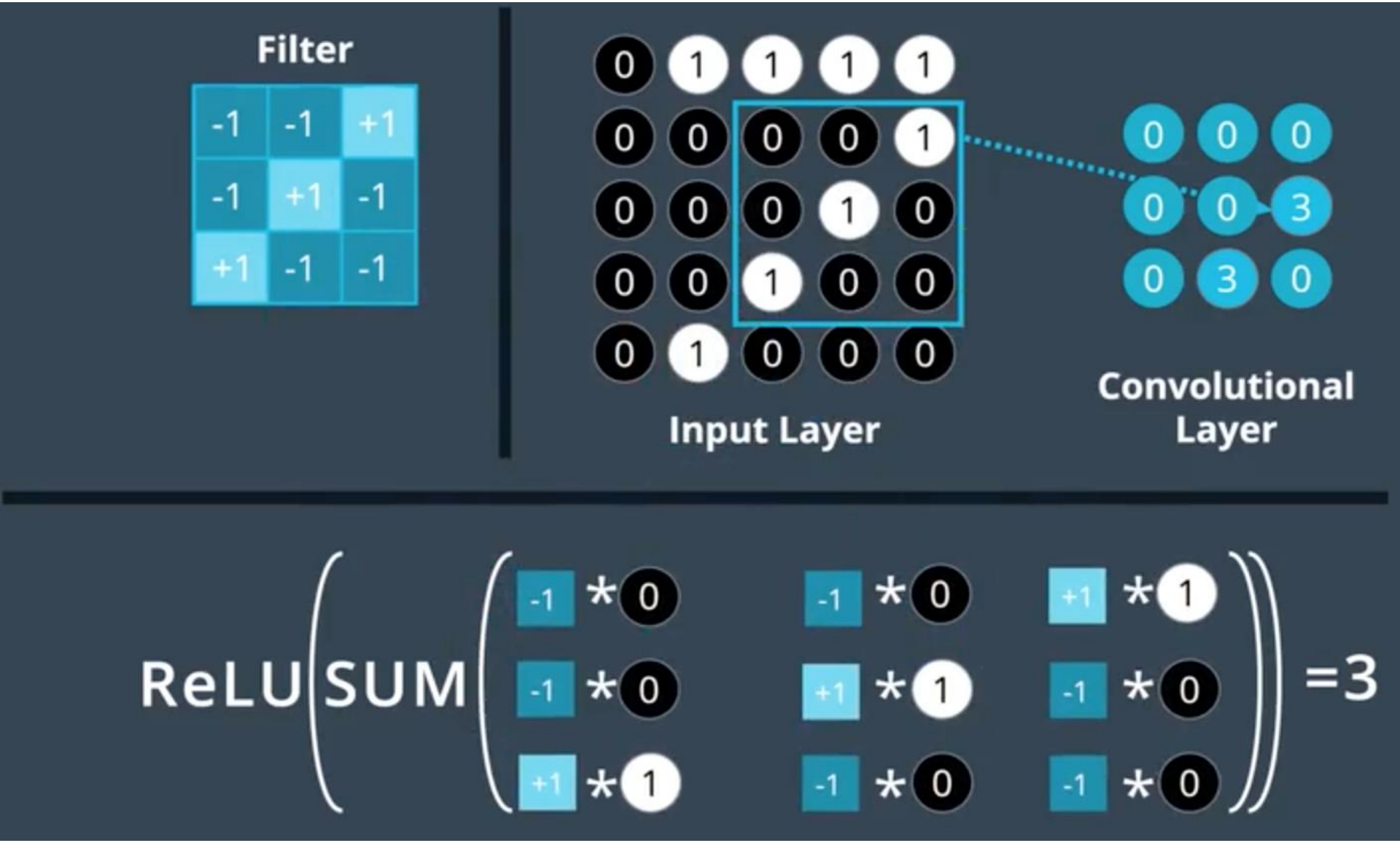


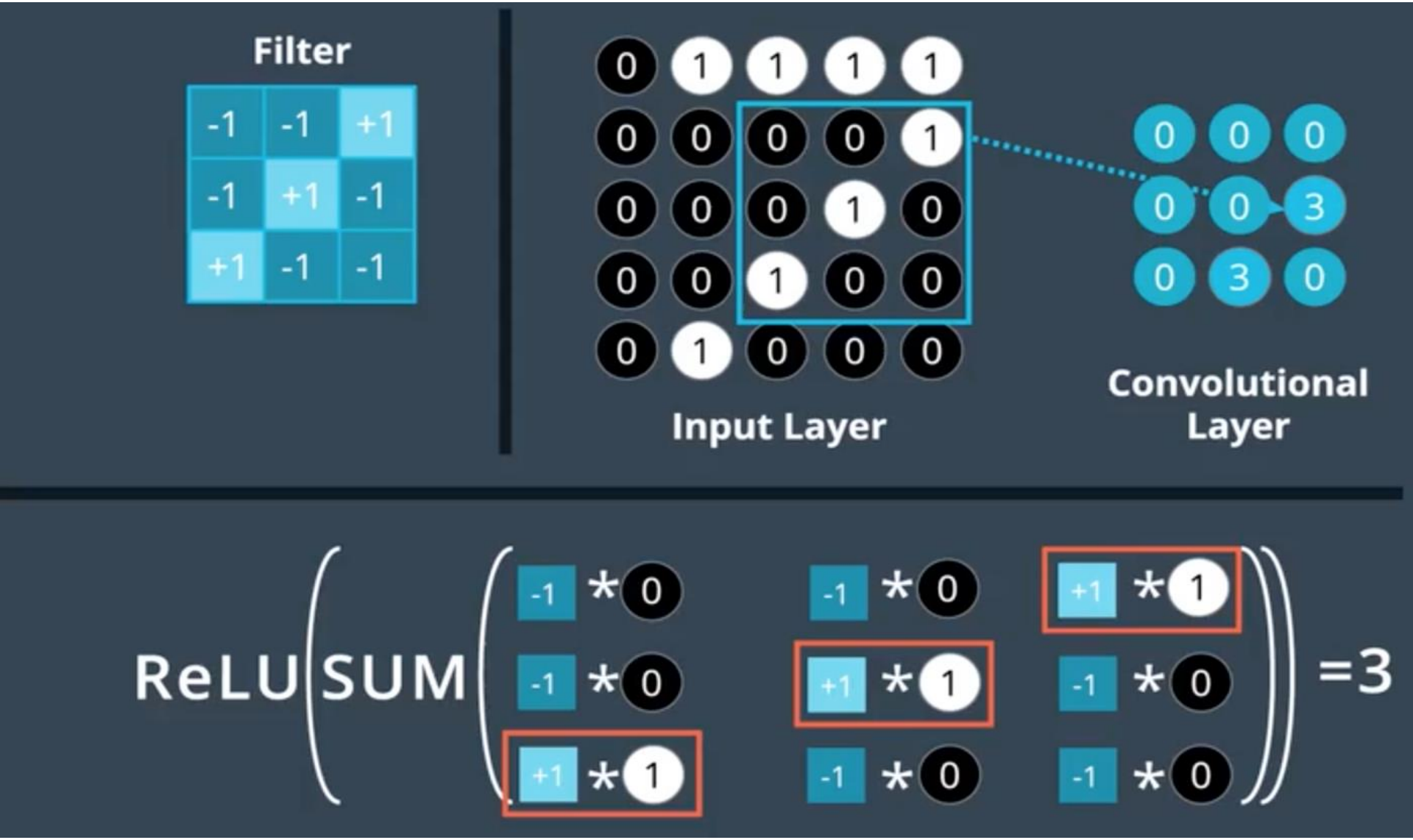
CAMADA CONVOLUTIVA

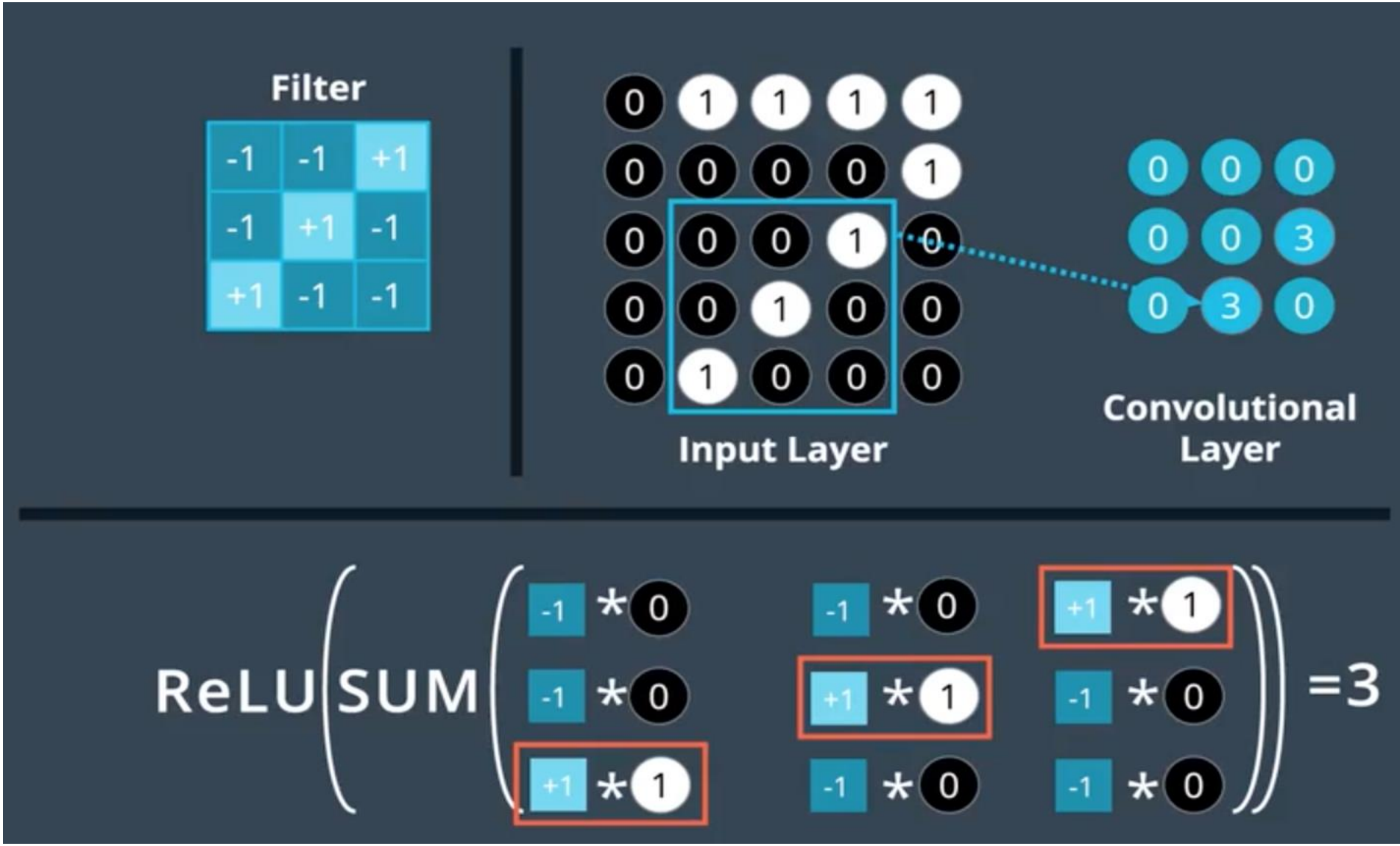



Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM * * * * * * * * * =0

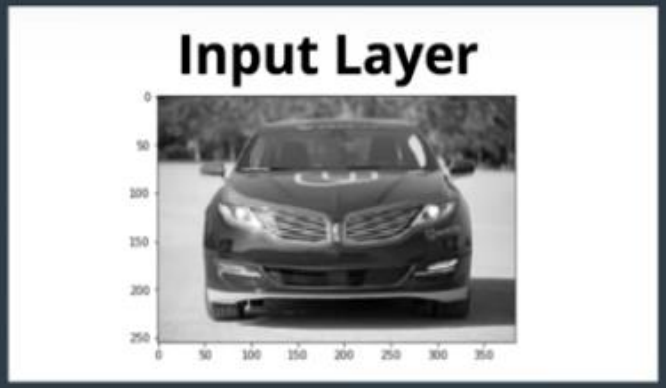

Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM (-1 * 0 -1 * 1 +1 * 1 -1 * 0 +1 * 0 -1 * 0 +1 * 0 -1 * 0 -1 * 0) =0


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM (-1 * 0 -1 * 0 +1 * 1 -1 * 0 +1 * 1 -1 * 0 +1 * 1 -1 * 0 -1 * 0) =3


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3


Filter
-1 -1 +1
-1 +1 -1
+1 -1 -1
0 1 1 1 1
0 0 0 0 1
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
Input Layer
0 0 0
0 0 3
0 3 0
Convolutional Layer
ReLU SUM
-1 * 0
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
+1 * 1
-1 * 0
-1 * 0
=3

COLEÇÃO DE FILTROS



Filter 1

-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1

Filter 2

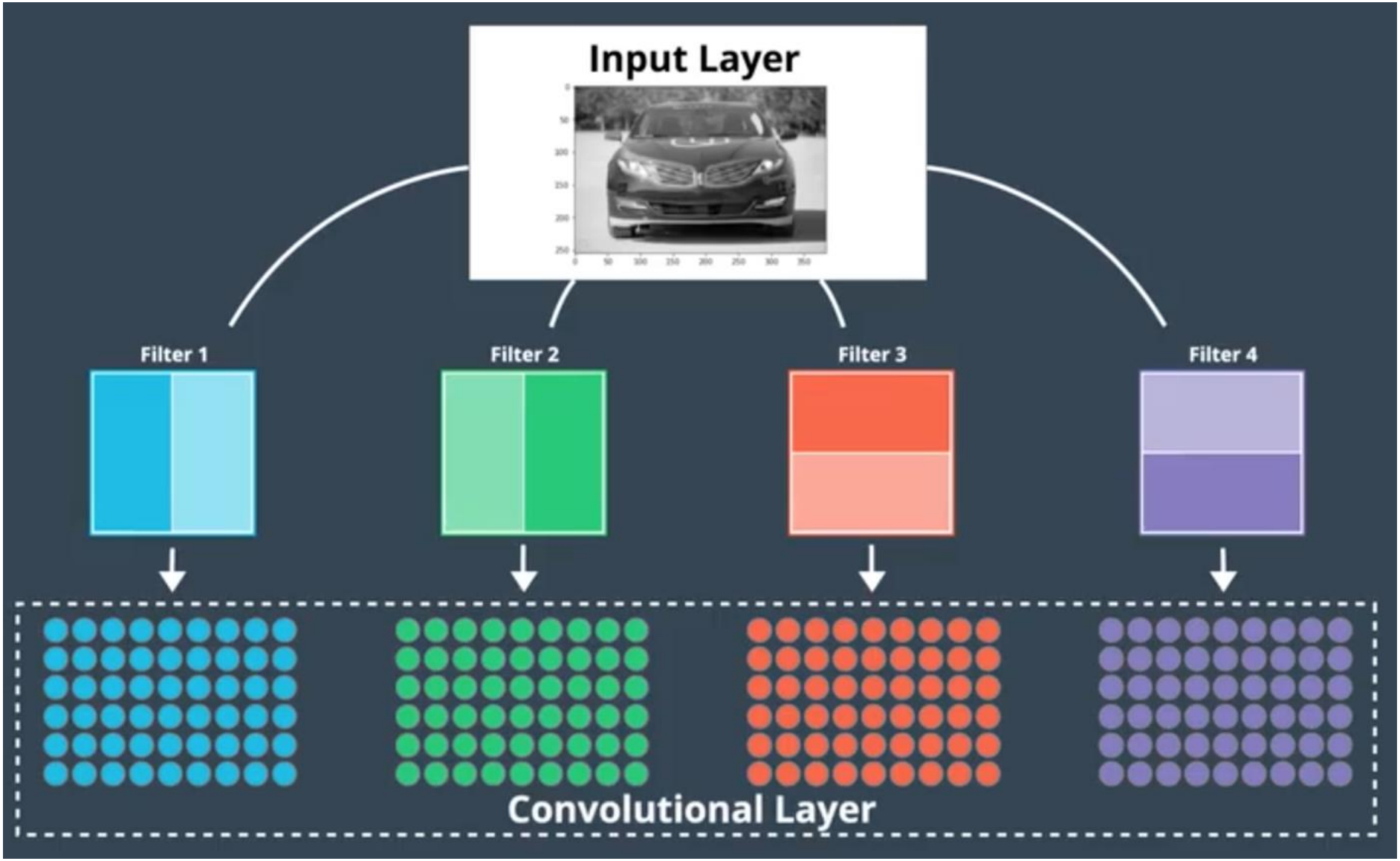
+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1

Filter 3

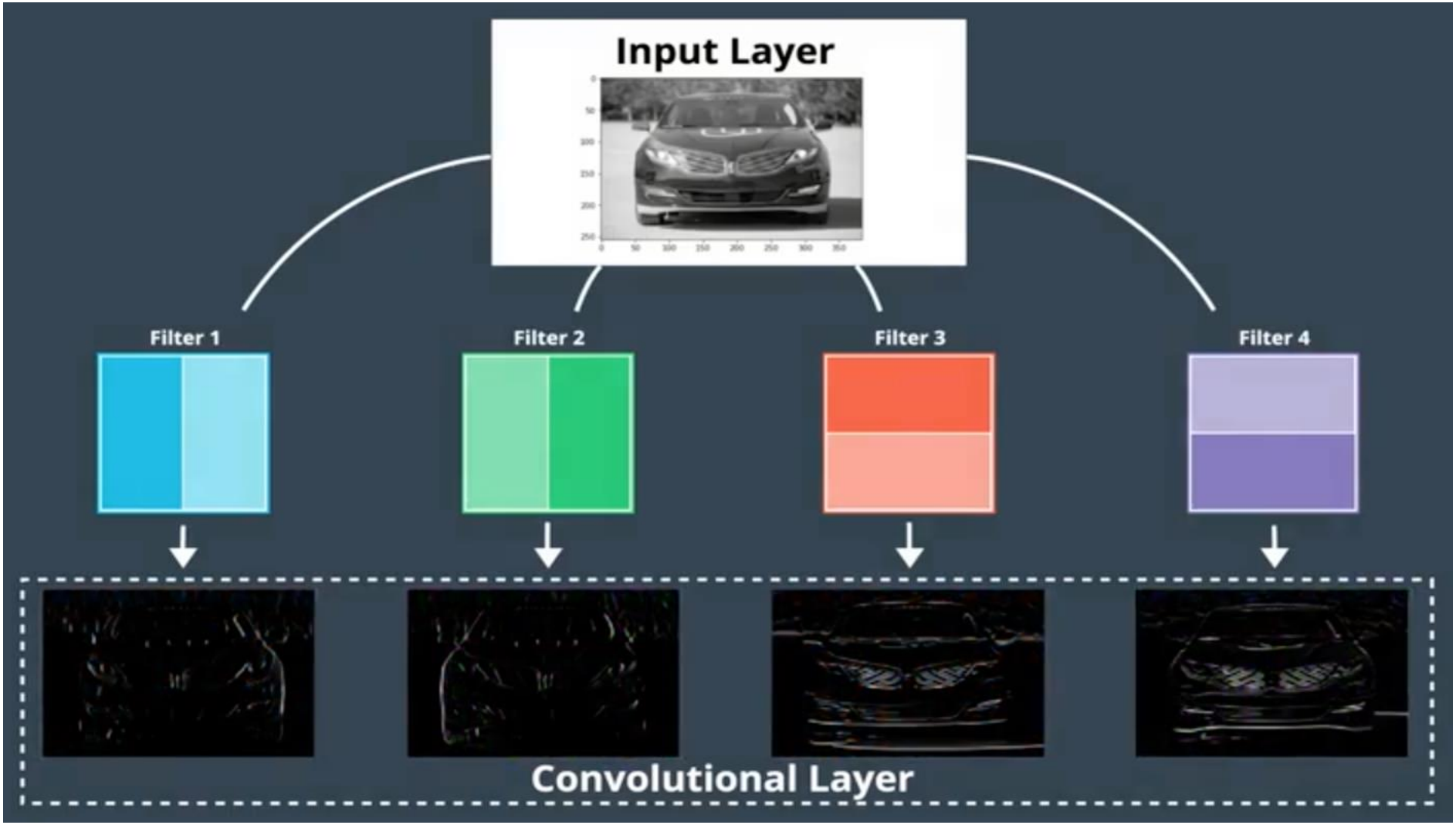
-1	-1	-1	-1
-1	-1	-1	-1
+1	+1	+1	+1
+1	+1	+1	+1

Filter 4

+1	+1	+1	+1
+1	+1	+1	+1
-1	-1	-1	-1
-1	-1	-1	-1


Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer

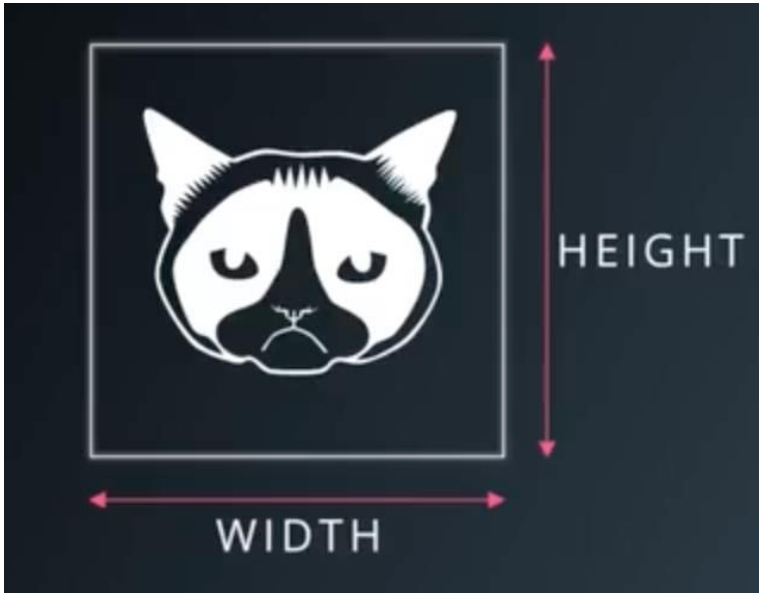
IMAGENS FILTRADAS

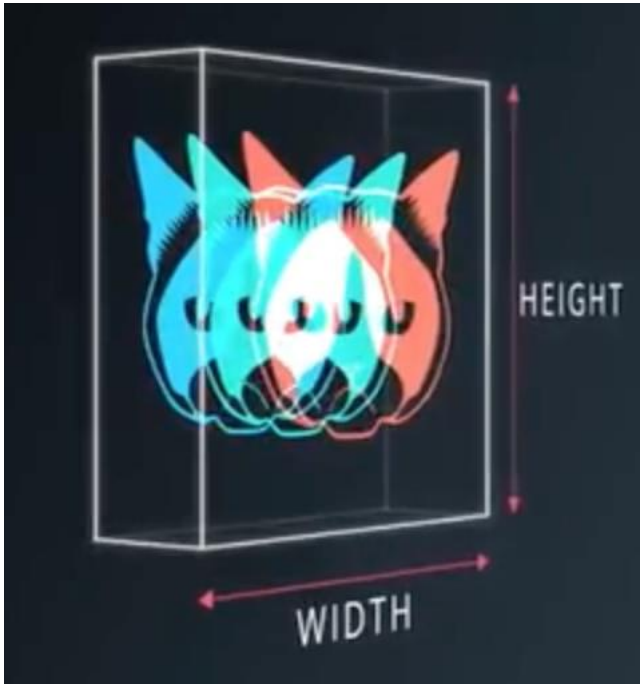


VÁRIAS REGIÕES DETECTADAS

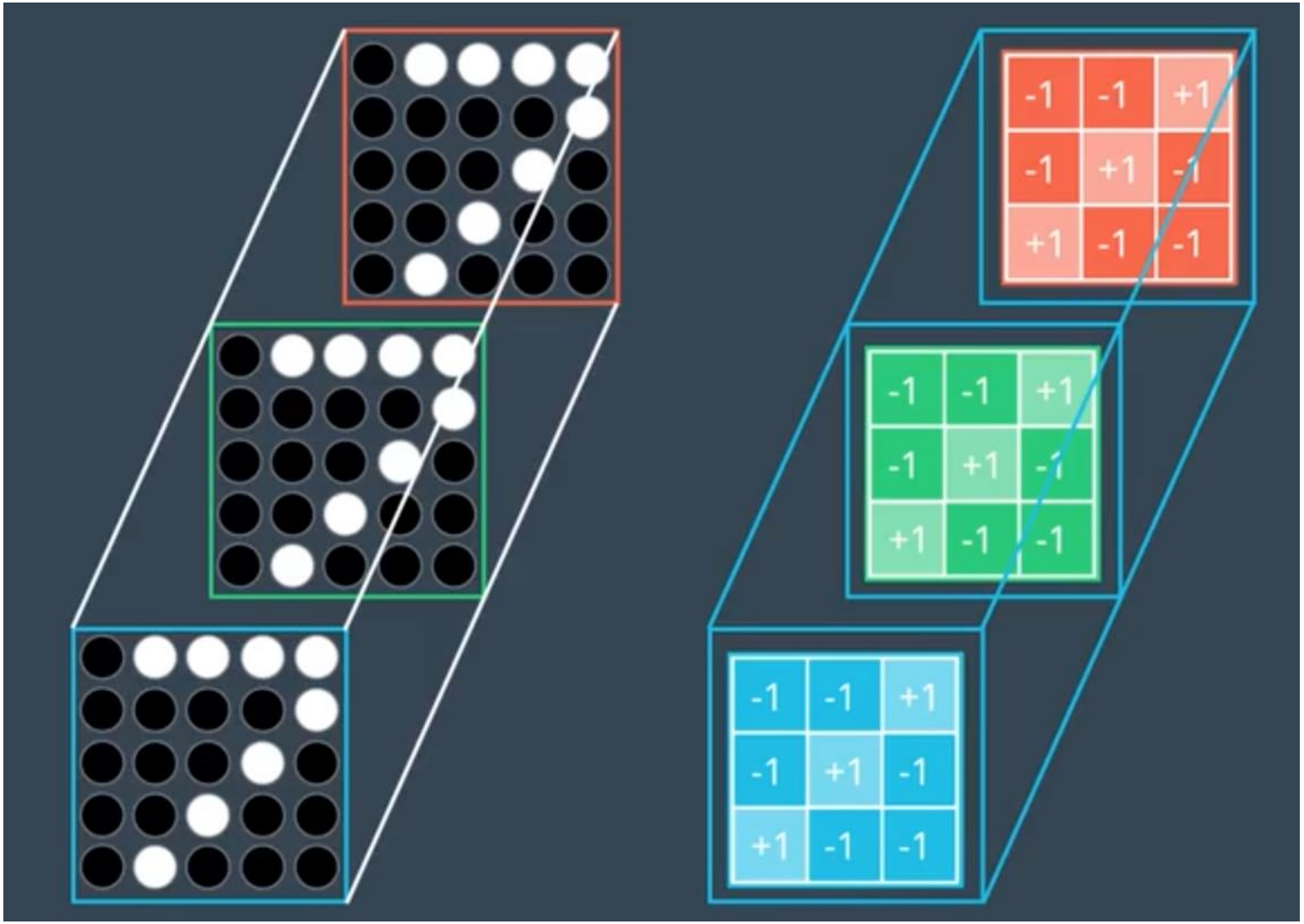
CONVOLUÇÃO EM IMAGENS COLORIDAS

- Imagens RGB (3 dimensões)

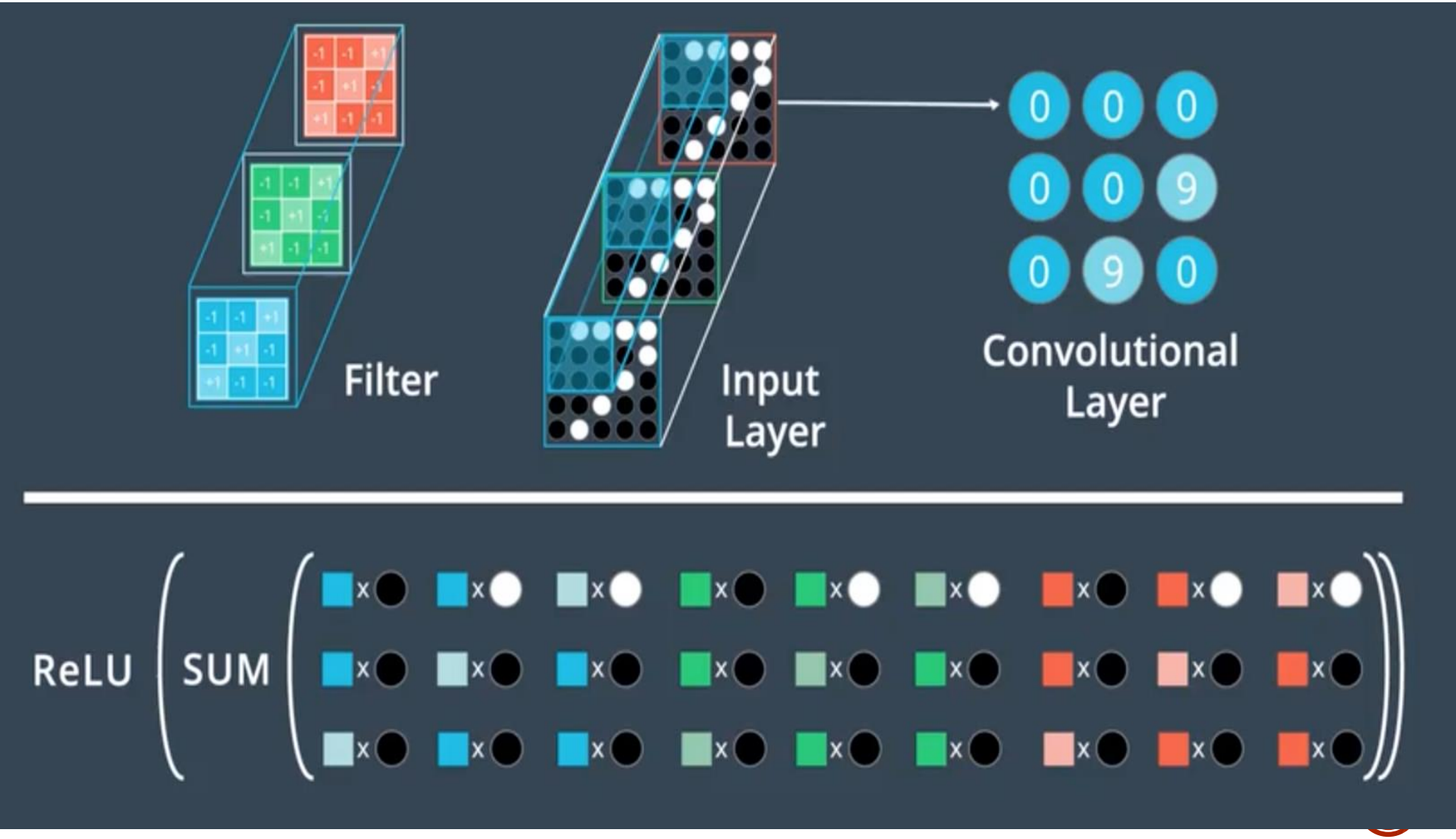




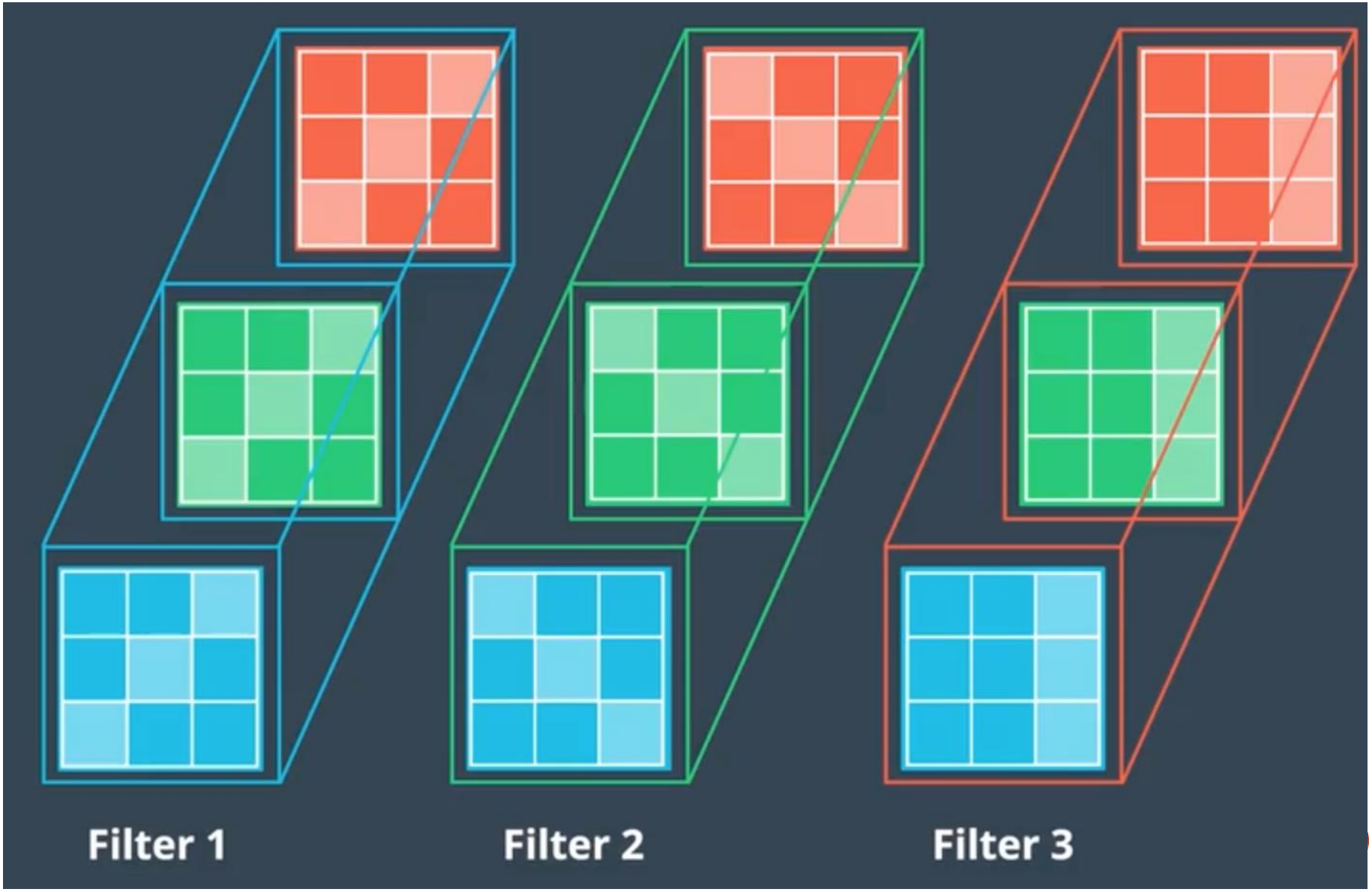
CONVOLUÇÃO EM IMAGENS COLORIDAS



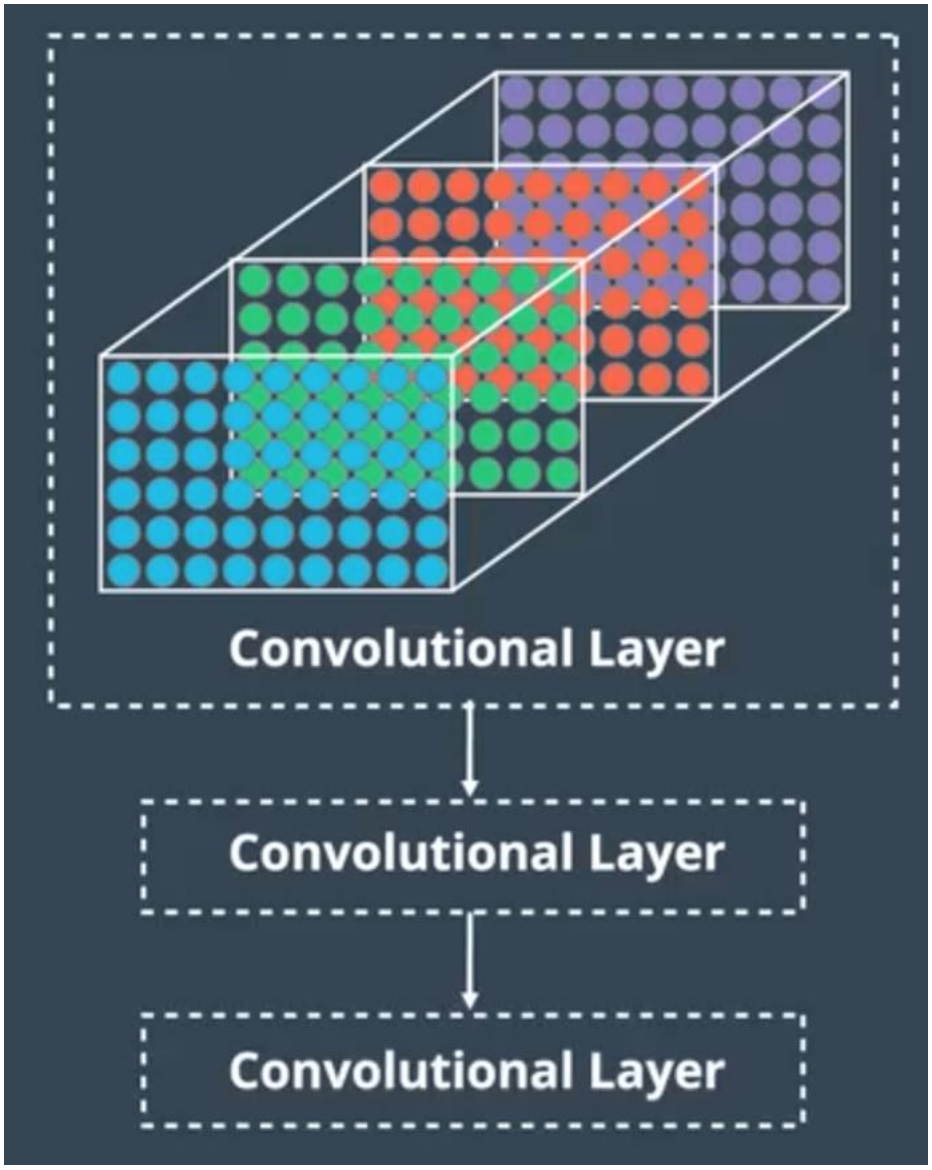
1 FILTRO COLORIDO



3 FILTROS COLORIDOS



VÁRIAS CAMADAS CONVOLUCIONAIS



PROPRIEDADES DA CAMADA CONVOLUCIONAL

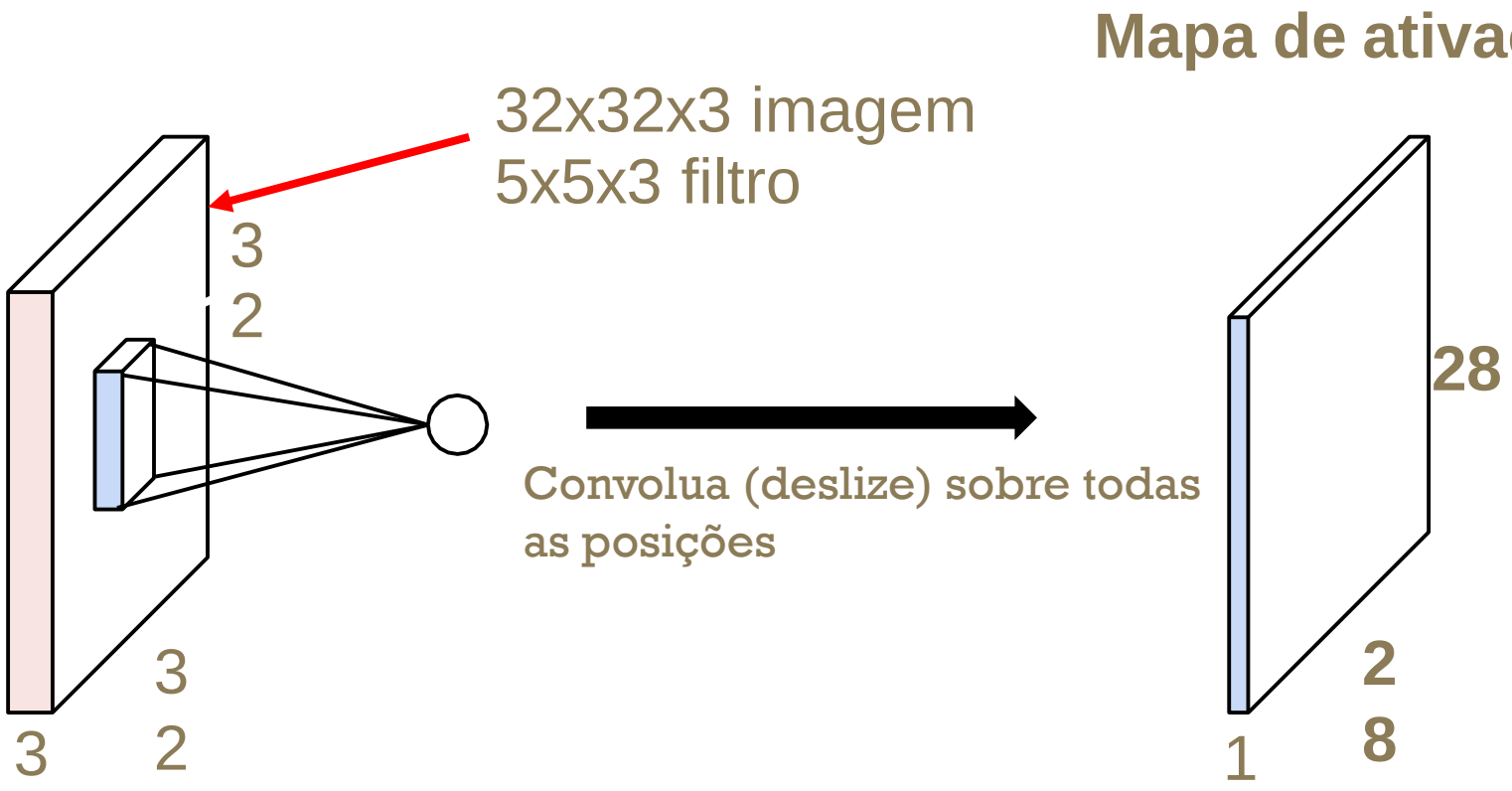
- Funciona da mesma forma que as camadas densas da MLP tradicional
- Os Filtros são iniciados de forma aleatória
- Seus pesos são aprendidos pela rede neural minimizando a função de custo
- Se uma imagem possui um cachorro, a rede neural aprenderá por si só, filtros que se parecem com um cachorro.
- Hiper Parâmetros:
  - Número de filtros
  - Largura x Altura desses filtros

OUTROS PARÂMETROS:

- Stride
- Padding
- Função de Ativação

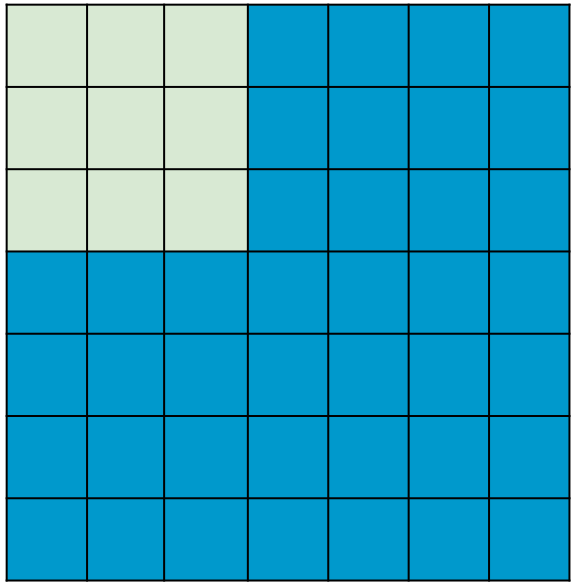
STRIDE

Olhando mais de perto as dimensões



STRIDE

- Olhando mais de perto as dimensões:

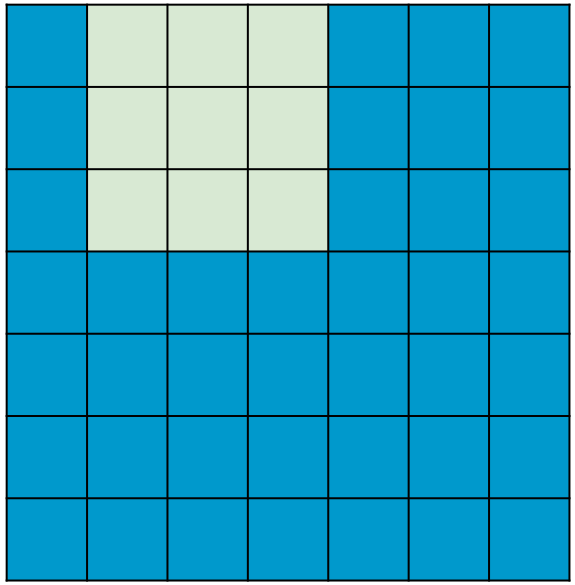


Entrada 7x7

Filtro 3x3

STRIDE

- Olhando mais de perto as dimensões:

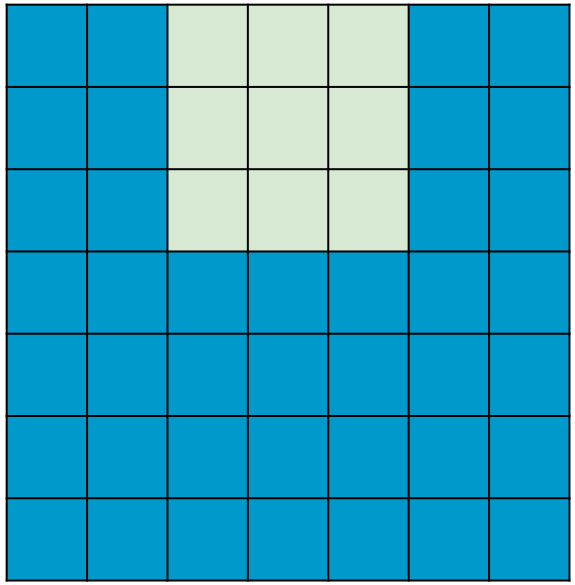


Entrada 7x7

Filtro 3x3

STRIDE

- Olhando mais de perto as dimensões:

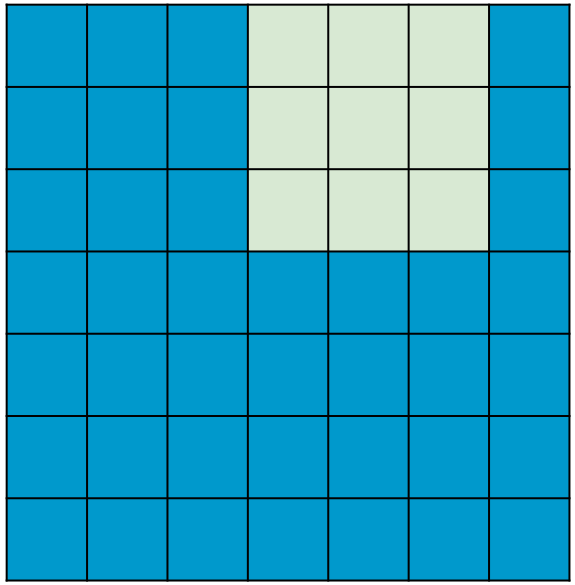


Entrada 7x7

Filtro 3x3

STRIDE

- Olhando mais de perto as dimensões:

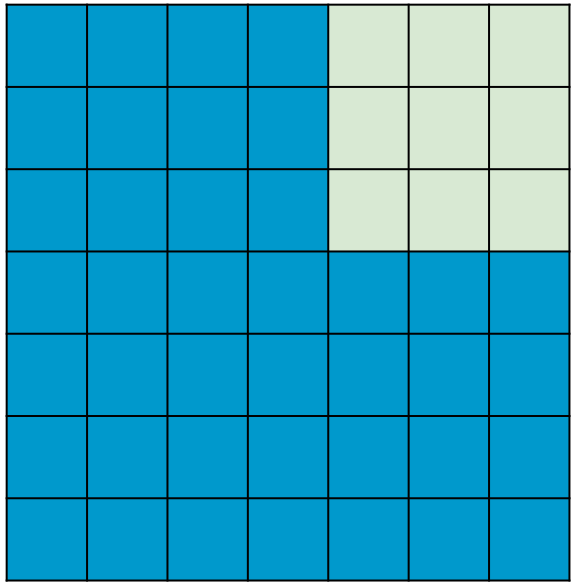


Entrada 7x7

Filtro 3x3

STRIDE

- Olhando mais de perto as dimensões:



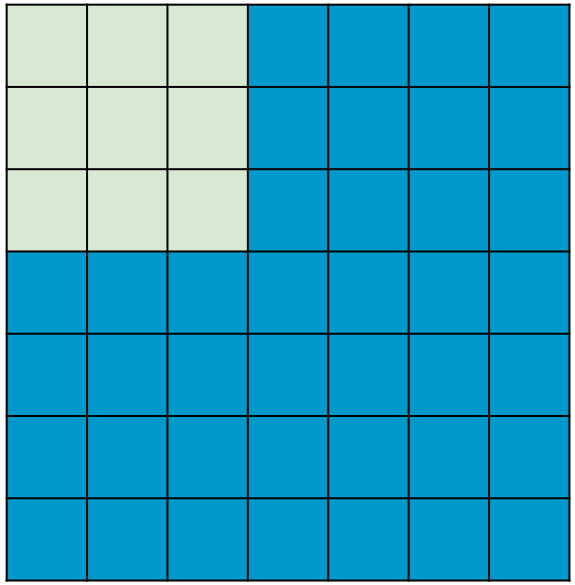
Entrada 7x7

Filtro 3x3

Saída 5 x 5

STRIDE

- Olhando mais de perto as dimensões:

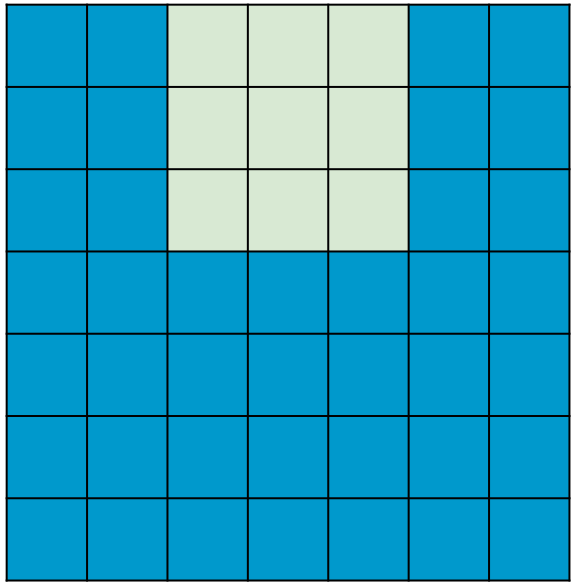


Entrada 7 x 7
Filtro 3 x 3

com stride 2 (stride = passo)

STRIDE

- Olhando mais de perto as dimensões:

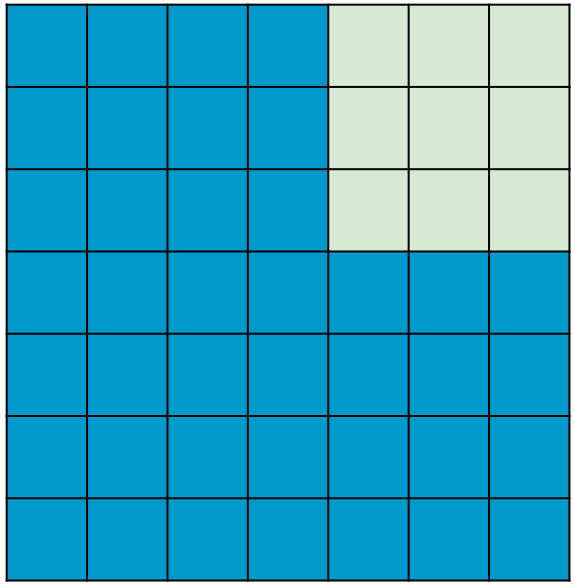


Entrada 7 x 7
Filtro 3 x 3

com stride 2 (stride = passo)

STRIDE

- Olhando mais de perto as dimensões:



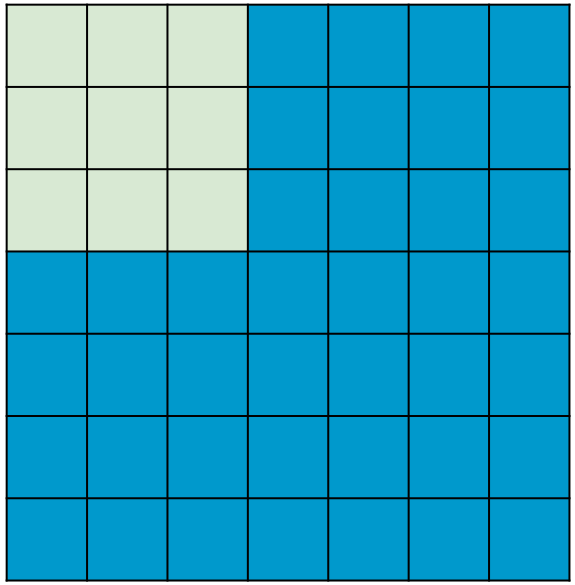
Entrada 7 x 7
Filtro 3 x 3

com stride 2 (stride = passo)

Saída 3 x 3

STRIDE

- Olhando mais de perto as dimensões:

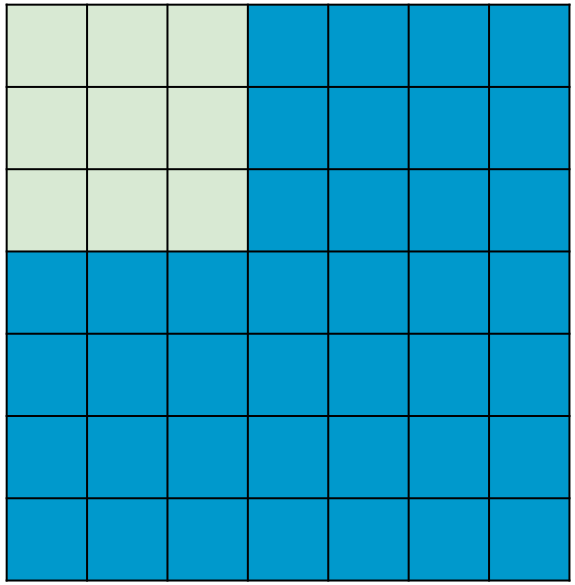


Entrada 7 x 7
Filtro 3 x 3

com stride 3 (stride = passo)

STRIDE

- Olhando mais de perto as dimensões:



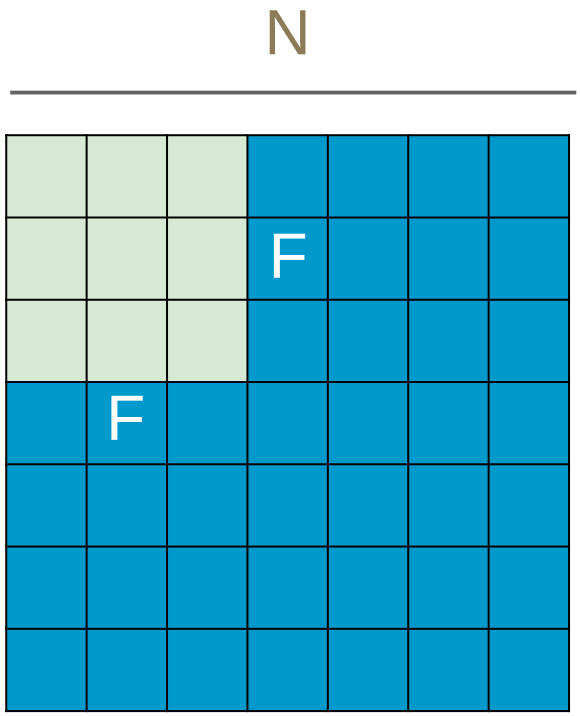
Entrada 7 x 7
Filtro 3 x 3

com stride 3 (stride = passo)

doesn't fit!
Você não consegue usar um filtro 3x3 em uma imagem 7x7 com stride 3.

STRIDE

- Olhando mais de perto as dimensões:

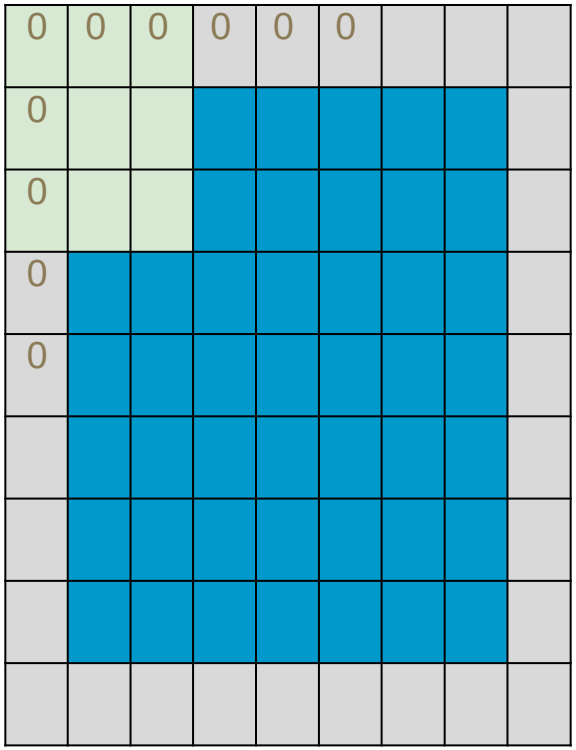


Calculando a saída:
(N - F) / stride + 1

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

PADDING

Na prática, costuma-se usar zero na borda



e.g. Entrada 7x7
Filtro 3x3 com **stride 1** **pad com 1 pixel de** borda

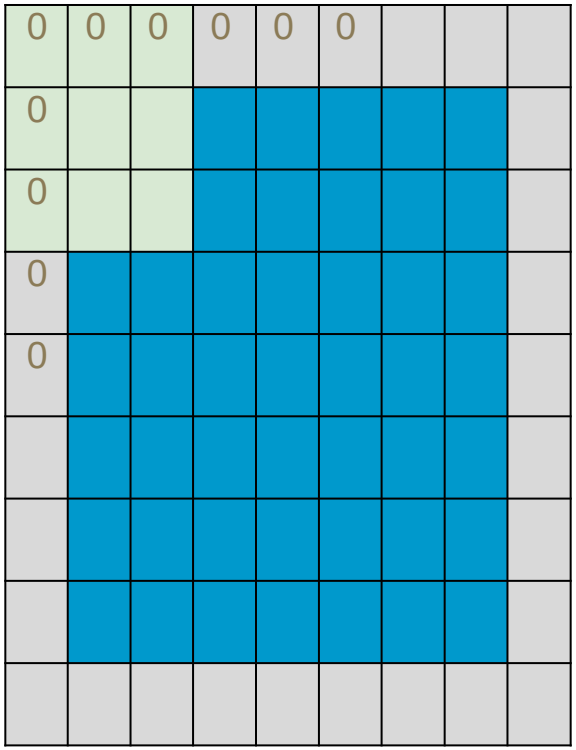
→ Qual é a saída?

(recall:)
(N - F) / stride + 1

Também chamado de padding

PADDING

Na prática, costuma-se usar zero na borda



e.g. Entrada 7x7
Filtro 3x3 com stride 1
pad com 1 pixel de borda

→ **Qual é a saída?**

Saída 7 x 7

(N - F) / stride + 1 → (9 – 3) / 1 + 1 = 7

(N + **2*padding** - F) / stride + 1

CAMADA CONVOLUCIONAL

- Para um volume de entrada *W*x*H*x*D*, um filtro de tamanho *F*, com passo *S e pad P*, a quantidade de neurônios em uma fatia do volume de saída é

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

- Exemplos:
  - Com entrada 7x7 e filtro 3x3 com passo 1 e pad 0, a saída teria 5x5 neurônios por fatia.
  - Com *S*=2, teríamos saída com 3x3 neurônios por fatia.

Fonte: http://cs231n.github.io/convolutional-networks/

RESUMO: CAMADA CONVOLUCIONAL

- 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;

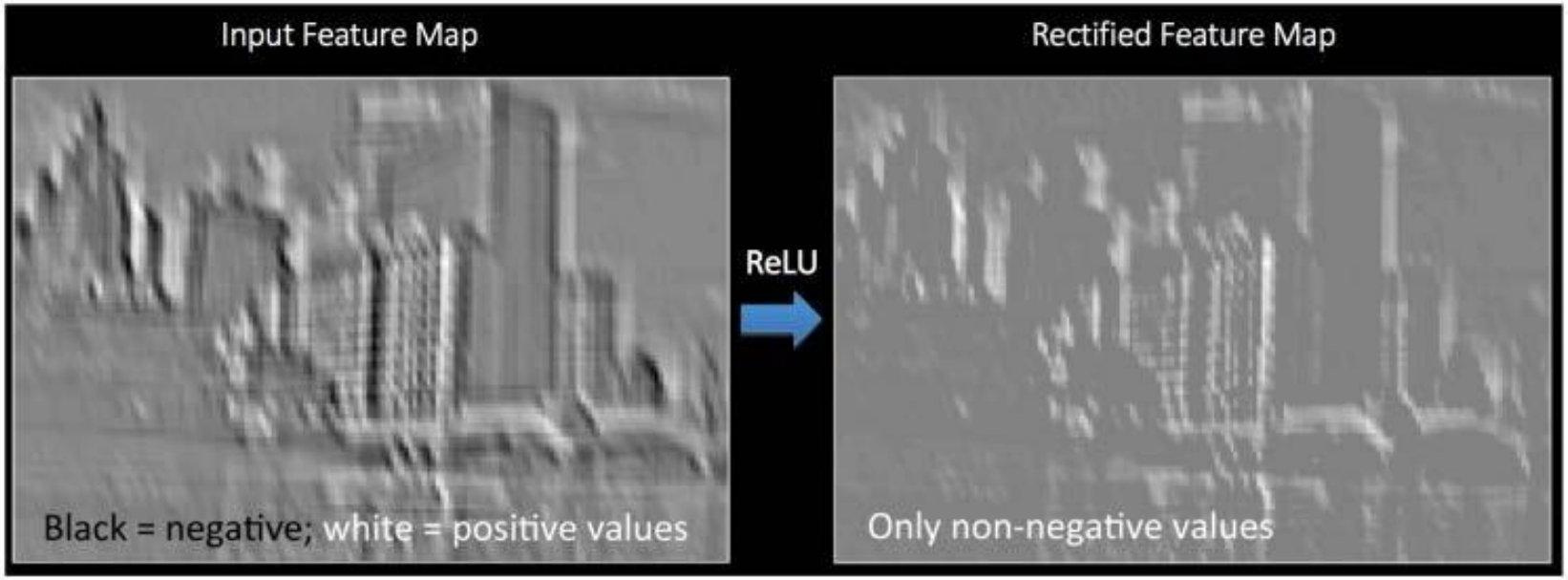
Configurações comuns:

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

RELU (RECTIFIED LINEAR UNITS)

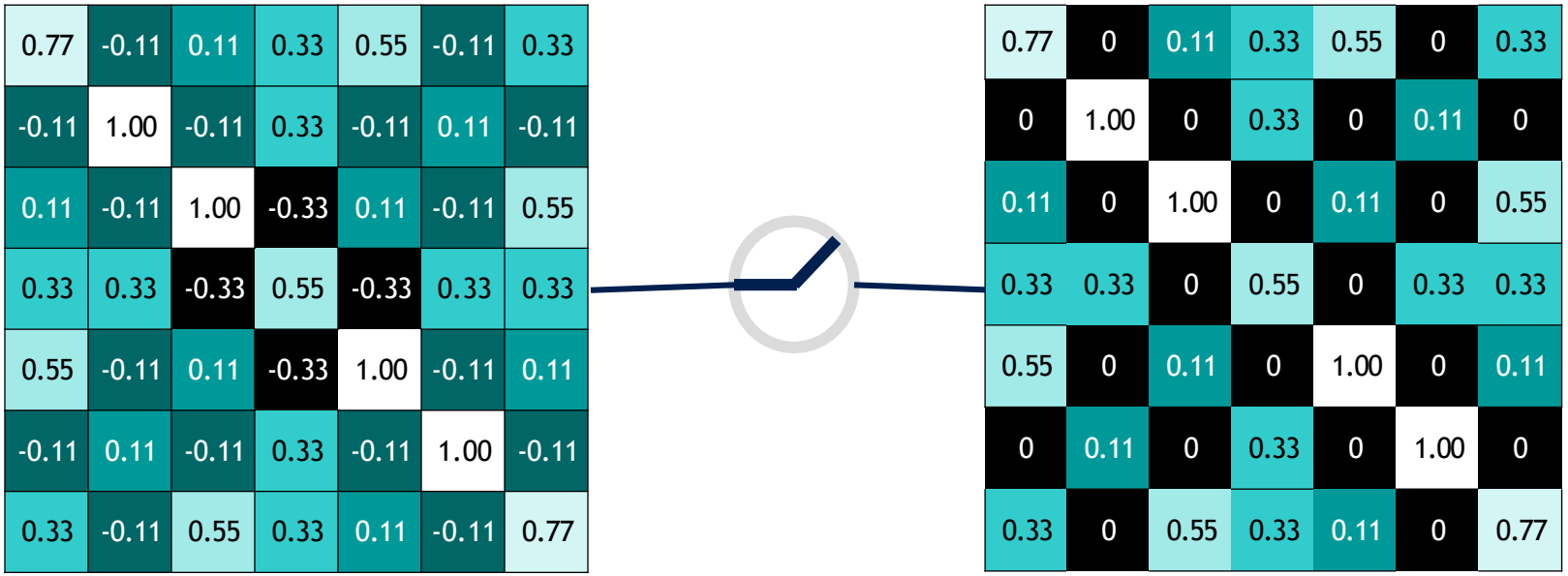
- Função de ativação comumente utilizada após uma camada de convolução
- Objetiva introduzir não linearidade
- Utilizada no lugar de "tanh" e "sigmoid" pois treina mais rápido e não interfere muito na acurácia
- Aplica a função $f(x) = \max(0, x)$
  - Transforma valores negativos em zero

RELU



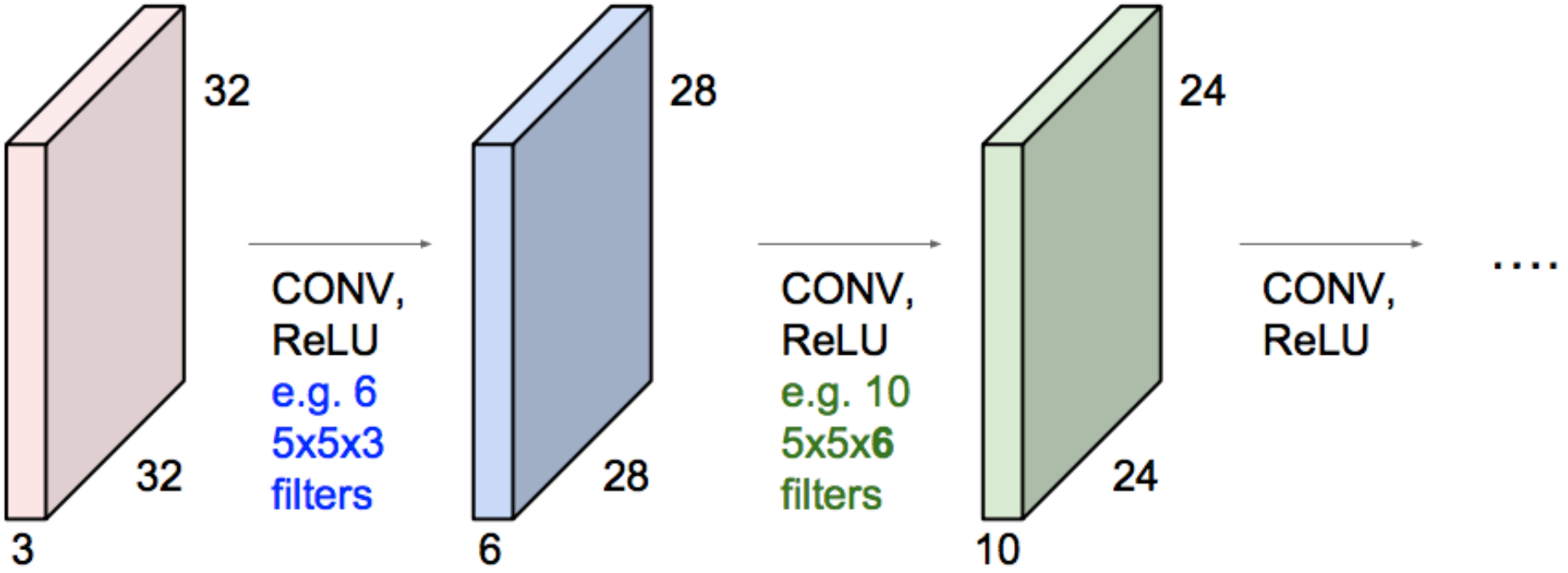
Somente valores positivos

RELU


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

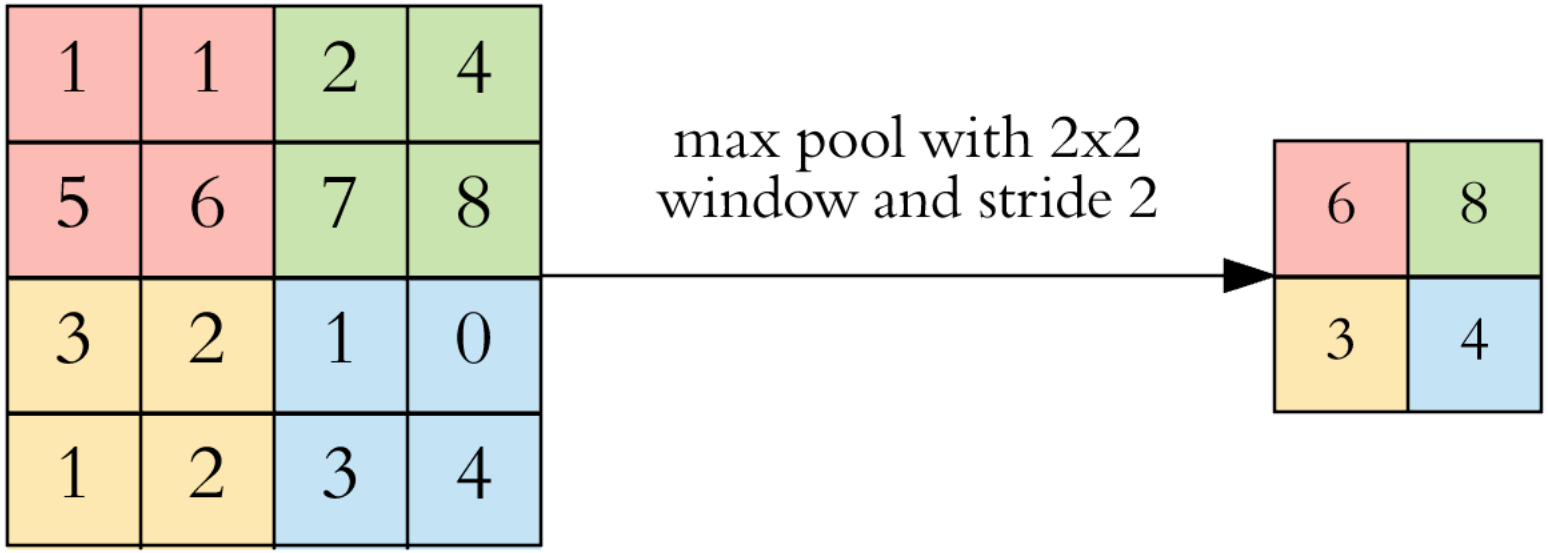
CAMADAS CONVOLUTIVAS

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



POOLING

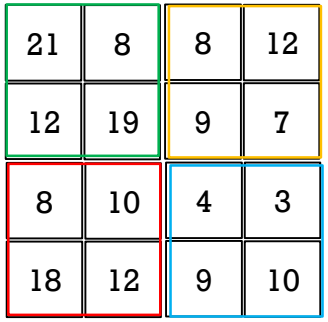
CAMADA DE POOLING



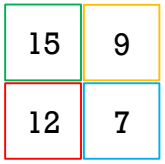
CAMADA DE POOLING – OUTRO EXEMPLO

$(S = 2, F = 2, W_1 = 4, H_1 = 4)$

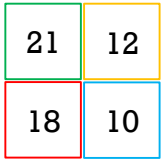
$W_2 = 2, H_2 = 2$



avg pooling



max pooling



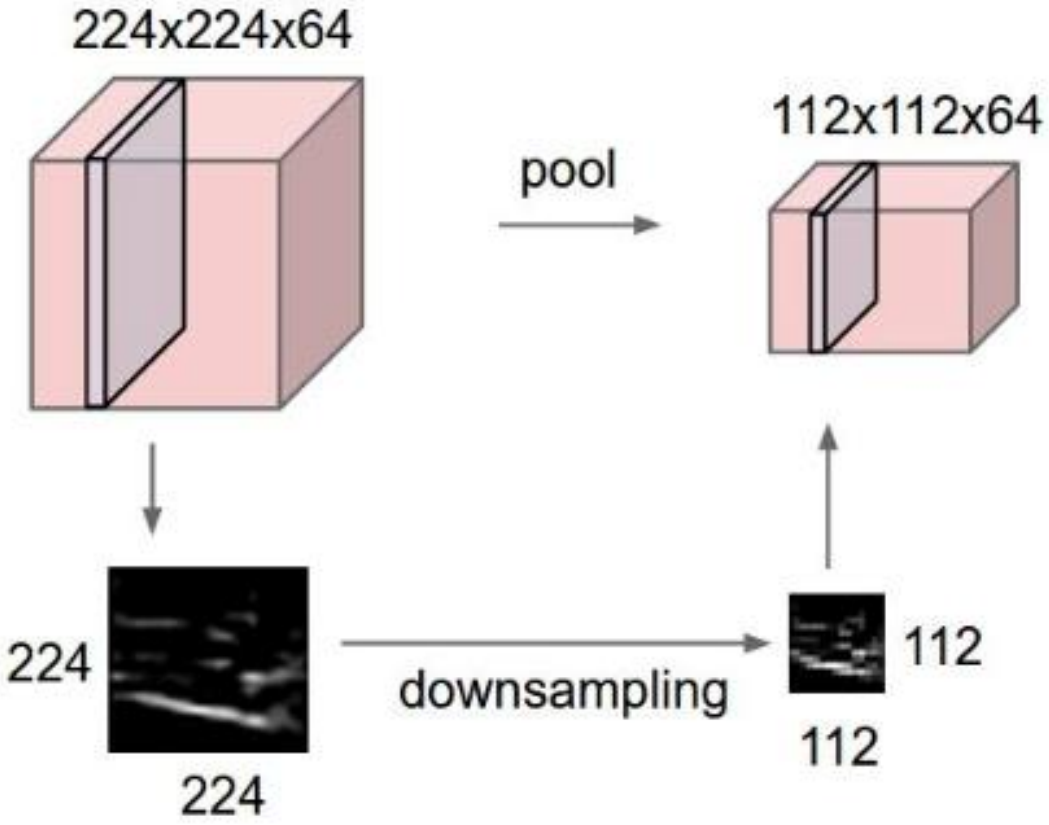
$$W_2 = (W_1 - F)/S + 1$$
$$H_2 = (H_1 - F)/S + 1$$

Repare que 75% das ativações são descartadas!

Imagem reproduzida do material disponibilizado na disciplina cs231n (http://cs231n.stanford.edu/)

CAMADA DE POOLING

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



RESUMO DE POOLING

Configurações comuns:
F=2, S=2
F=3, S=2

- Aceita um volume de tamanho $W_1 \times H_1 \times D_1$
- 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.

POR QUÊ MAX POOLING?

- Reduz tamanho especial da representação;
- Reduz número de parâmetros drasticamente;
- Filtro 2x2 com stride=2 descarta 75% das ativações
- Controla overfitting.
- Provê invariância à translação.

FULLY-CONECTED (FC)

FULLY CONNECTED

- Atua como um classificador
- Recebe uma entrada da camada anterior (Conv ou ReLu ou Pooling) e fornece um vetor N dimensional onde C é o número de classes de prováveis saídas
- Exemplo: reconhecimento de dígitos
  - C = 10, pois são dígitos
  - Saída

    [0 .1 .1 .75 0 0 0 0 0 .05]

    10% ser o digito 1

    10 % ser o digito 2

    75% ser o digito 3

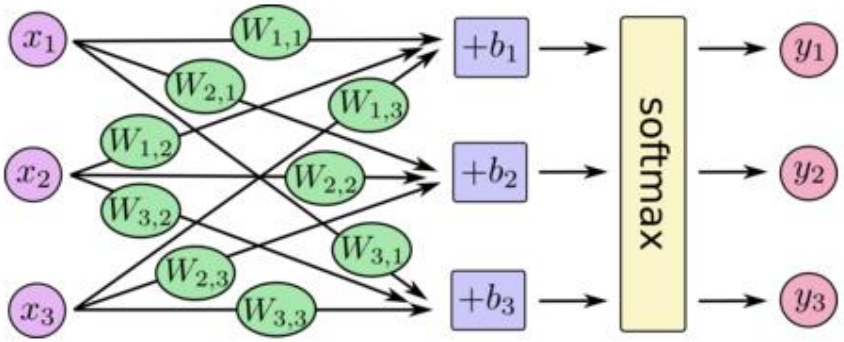
    5% ser o digito 9

softmax approach

SOFTMAX

$$z_i = f(net_i) = \frac{e^{net_i}}{\sum_{c=1}^{C} e^{net_c}}$$

- Quando o propósito de uma RNA é realizar a tarefa de classificação, é comum usar a função softmax.



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \end{pmatrix}$$

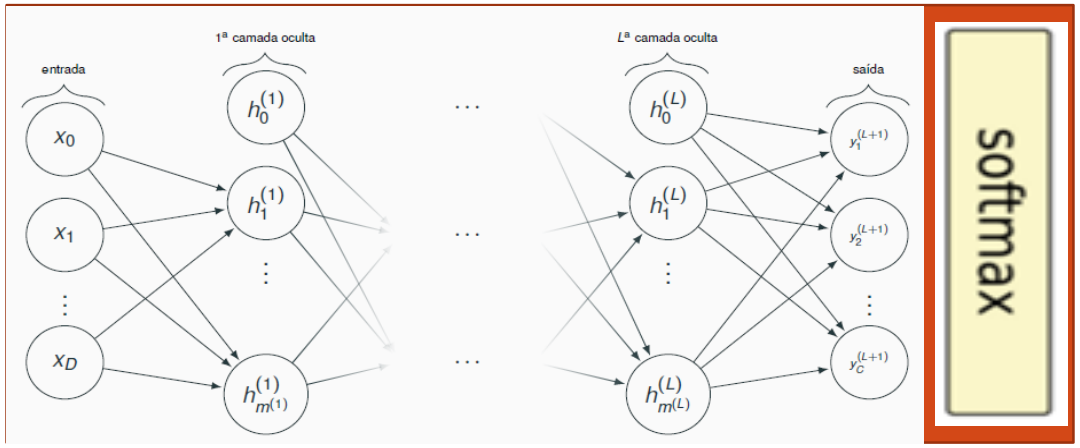
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

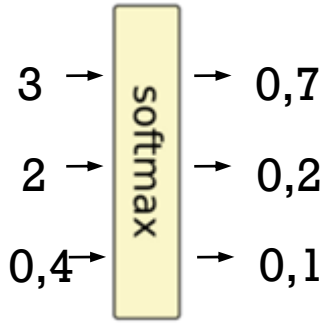
Fonte: https://www.tensorflow.org/versions/r0.9/tutorials/mnist/beginners/index.html

SOFTMAX

$$z_i = f(net_i) = \frac{e^{net_i}}{\sum_{c=1}^{C} e^{net_c}}$$

- Permite interpretar os valores da camada de saída como uma distribuição de probabilidades.





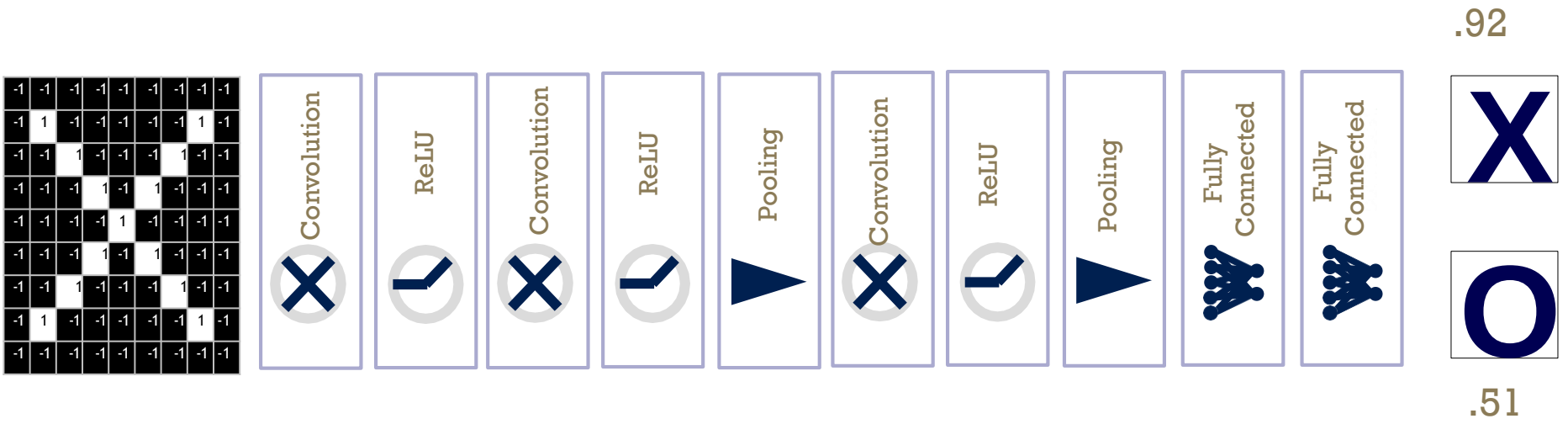
FULLY CONNECTED

- Basicamente a camada utiliza o mapa de ativação da camada anterior com os maiores valores das características

- Exemplo:
  - se o programa visa identificar um cachorro, os mapas de ativação com maiores valores serão aqueles que representam as características de 4 patas, orelhas, etc.
  - se for de um um pássaro os mapas de ativação serão aqueles com características que representam asas, bico, etc

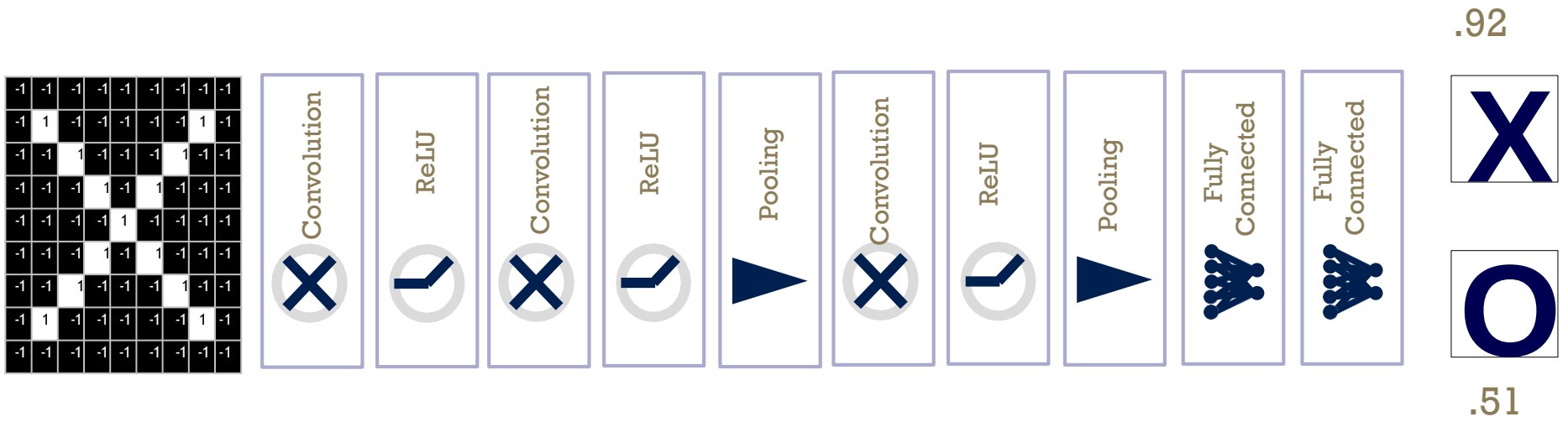
JUNTANDO TUDO...

Um conjunto de pixels que se torna um conjunto de votos



JUNTANDO TUDO...

Algoritmo de backpropagation utilizado para fazer os ajustes



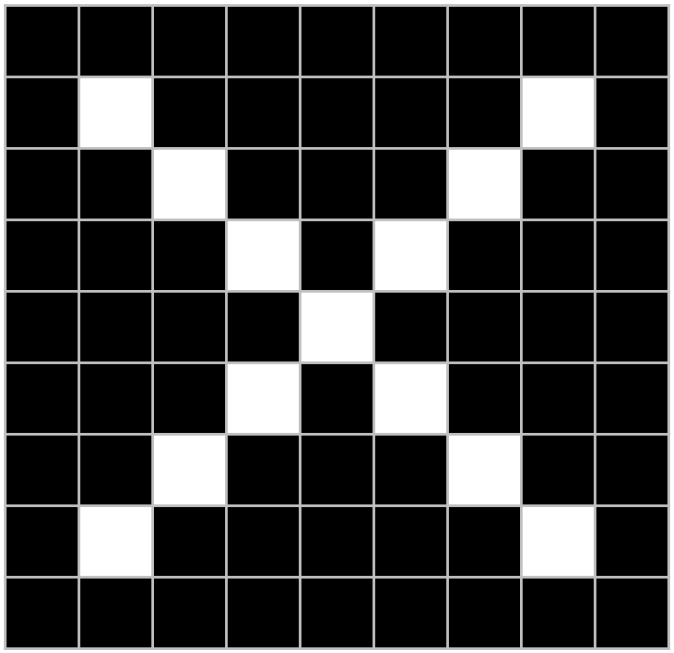
	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
		Total	0.57

POSSIBILIDADES DE APLICAÇÕES

IMAGENS

Coluna de pixels →

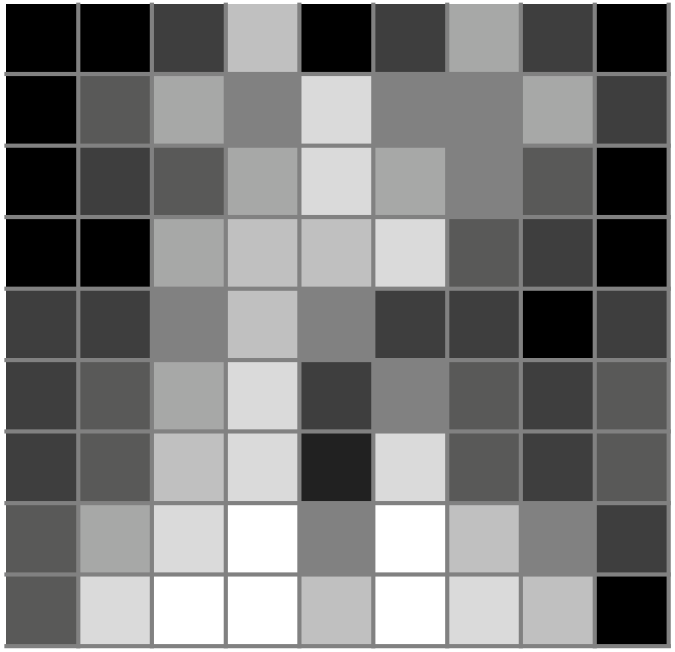
Linha de pixels →



SONS

Intervalos de tempo →

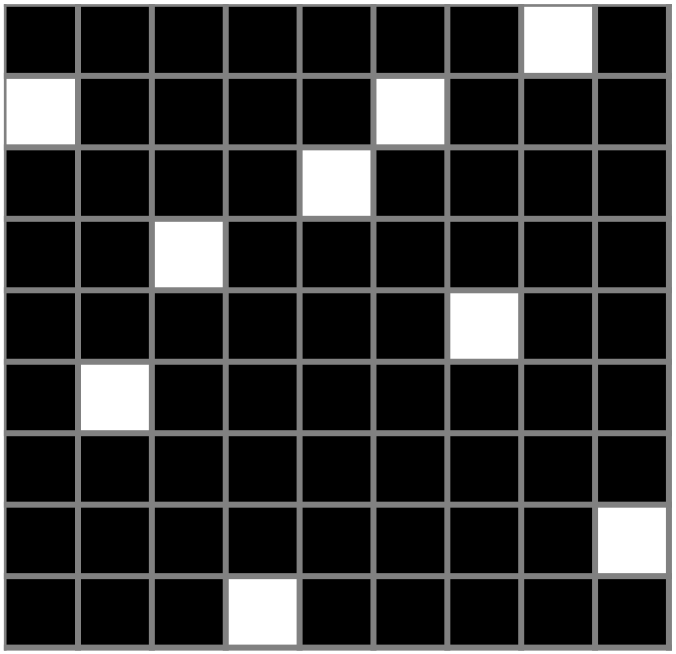
Intensidade em cada freqüência de banda →



TEXTO

Posição na sentença →

Palavras no dicionário →



DADOS DE CLIENTES

Nome, idade, endereço, email,
Financeiro, atividade na internet, …→

Clientes→

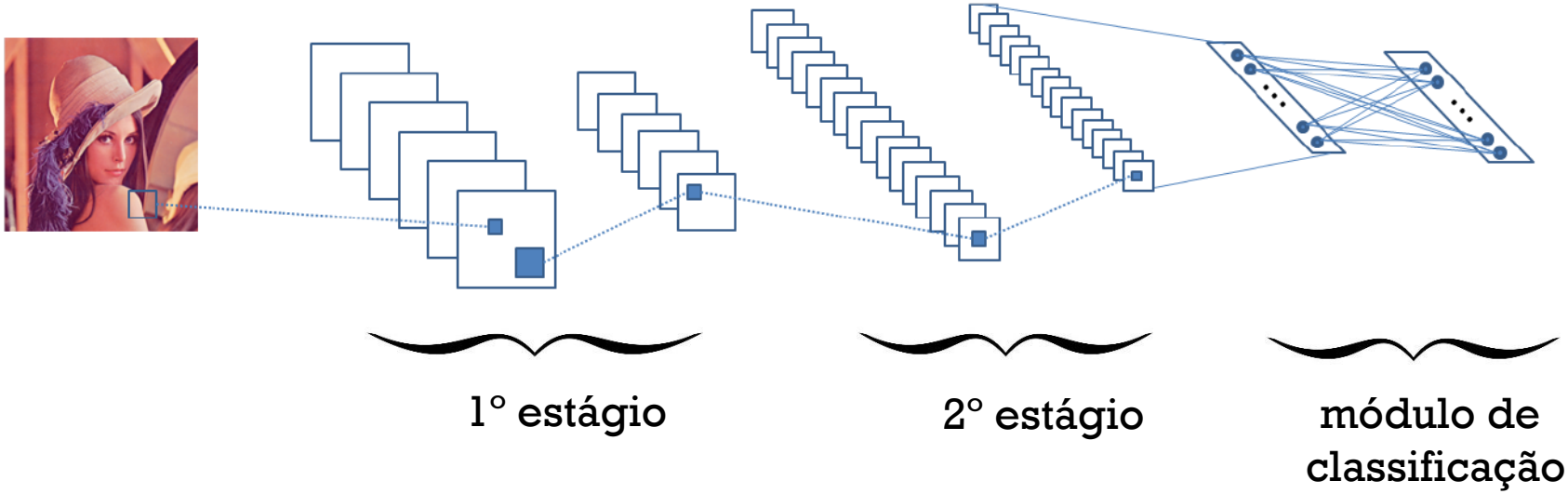
A	22	1A	a@a	1	aa	a1.a	123	aa1
B	33	2B	b@b	2	bb	b2.b	234	bb2
C	44	3C	c@c	3	cc	c3.c	345	cc3
D	55	4D	d@d	4	dd	d4.d	456	dd4
E	66	5E	e@e	5	ee	e5.e	567	ee5
F	77	6F	f@f	6	ff	f6.f	678	ff6
G	88	7G	g@g	7	gg	g7.g	789	gg7
H	99	8H	h@h	8	hh	h8.h	890	hh8
I	111	9I	i@i	9	ii	i9.i	901	ii9

EXEMPLOS DE CNNS

ARQUITETURA TÍPICA

- Em uma CNN, encontramos um ou mais estágios, cada qual composto por camadas:
  - de convolução (CONV),
  - de subamostragem (POOL),
  - de normalização de contraste,
- Ao final, é comum encontrar uma ou mais camadas completamente conectadas seguida de camada softmax (módulo de classificação).

ARQUITETURA TÍPICA

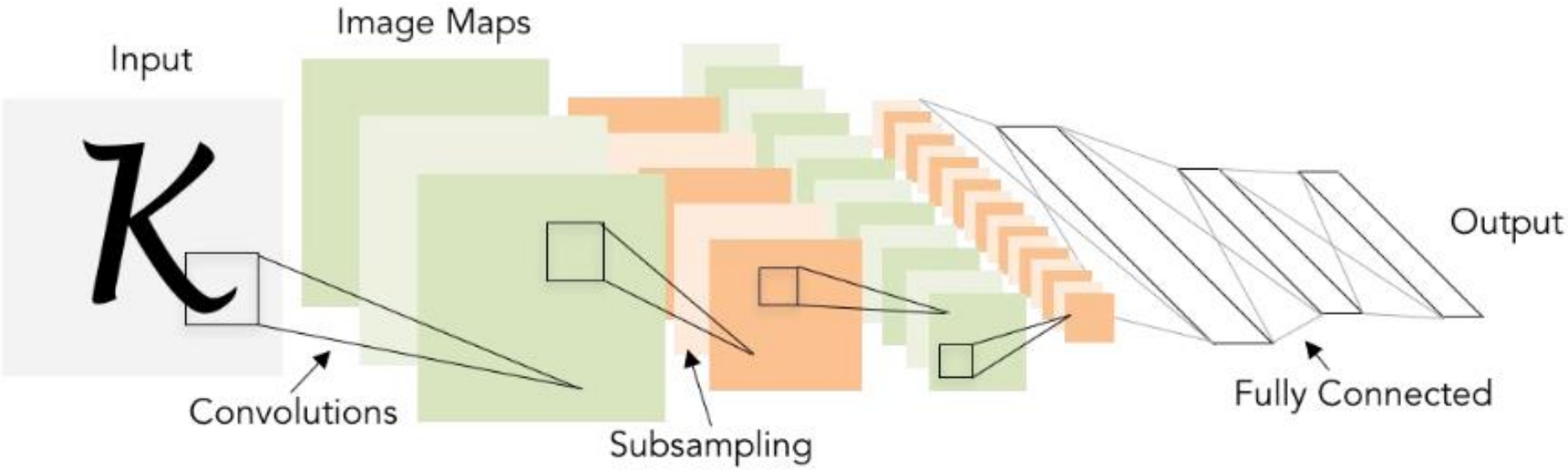


EXEMPLOS DE CNNS

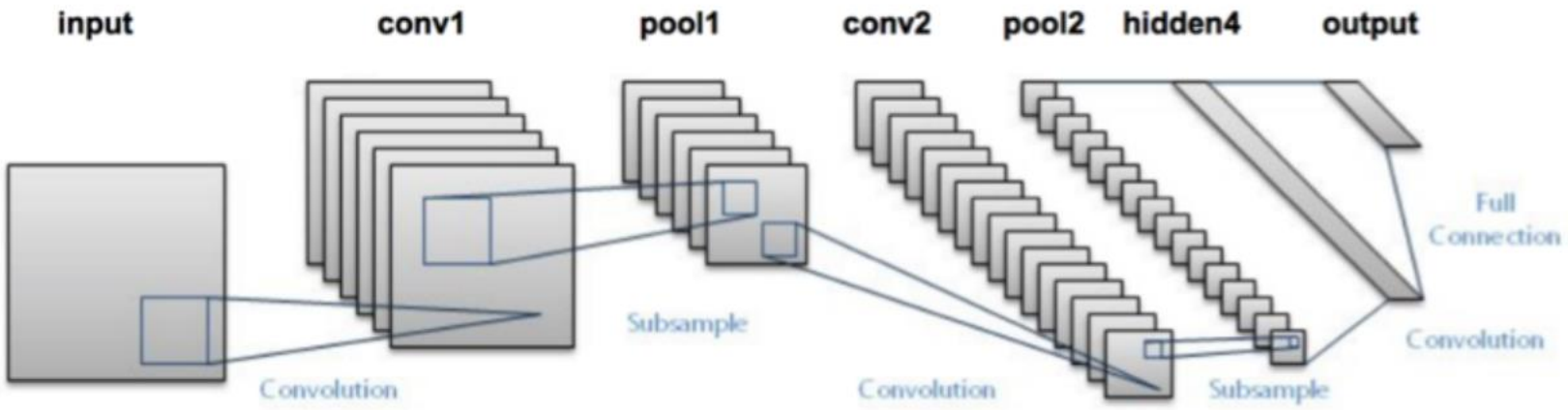
- Estudos de Caso
  - Lenet
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- Existe Também
  - NiN (Network in Network)
  - Wide ResNet
  - ResNetXT
  - Stochastic Depth
  - DenseNet
  - FractalNet
  - SqueezeNet

LENET-5

- Yan Le Cun et al, Convolutional Networks and Applications in Vision (2010)



- Filtros convolutivos 5x5, aplicados com stride 1
- Camadas de pooling com filtro 2x2 aplicado com stride 2
- [CONV-POOL-CONV-POOL-FC-FC]

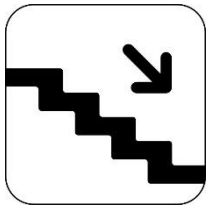


Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$28 \times 28 \times 1$	
CONV	$28 \times 28 \times 20$	$5 \times 5, K = 20$
ACT	$28 \times 28 \times 20$	
POOL	$14 \times 14 \times 20$	2×2
CONV	$14 \times 14 \times 50$	$5 \times 5, K = 50$
ACT	$14 \times 14 \times 50$	
POOL	$7 \times 7 \times 50$	2×2
FC	500	
ACT	500	
FC	10	
SOFTMAX	10	

```
INPUT => CONV => TANH => POOL => CONV => TANH => POOL =>
     FC => TANH => FC
```

ALEXNET, 2012

- Competição ILSVRC (*ImageNet Challenge*)
  - ~1,2M de imagens de alta resolução para treinamento;
  - ~1000 imagens por classe; 1000 classes!
- Ganhou a edição 2012
  - Até então, soluções incluiam características produzidas manualmente, estudadas e melhoradas por décadas.



9%

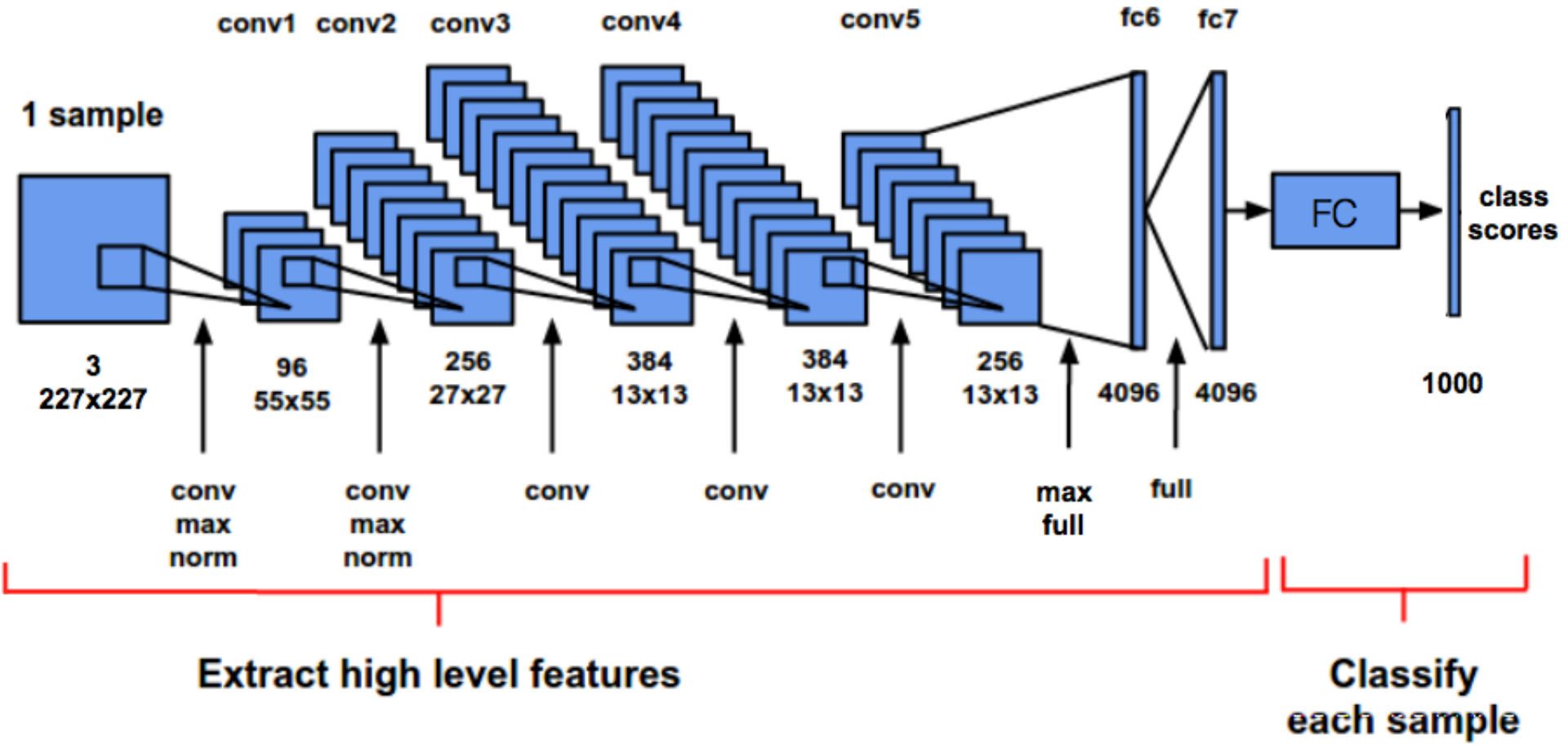
ImageNet Classification with I
https://papers.nips.cc/paper/4824-ima
by A Krizhevsky - 2012 - Cited by 6933 -

Out/2016

ImageNet Classification with Deep C
https://papers.nips.cc/.../4824-imagenet-classi
by A Krizhevsky - 2012 - Cited by 12168 - Relatec

Out/2017

Example: AlexNet [Krizhevsky 2012]

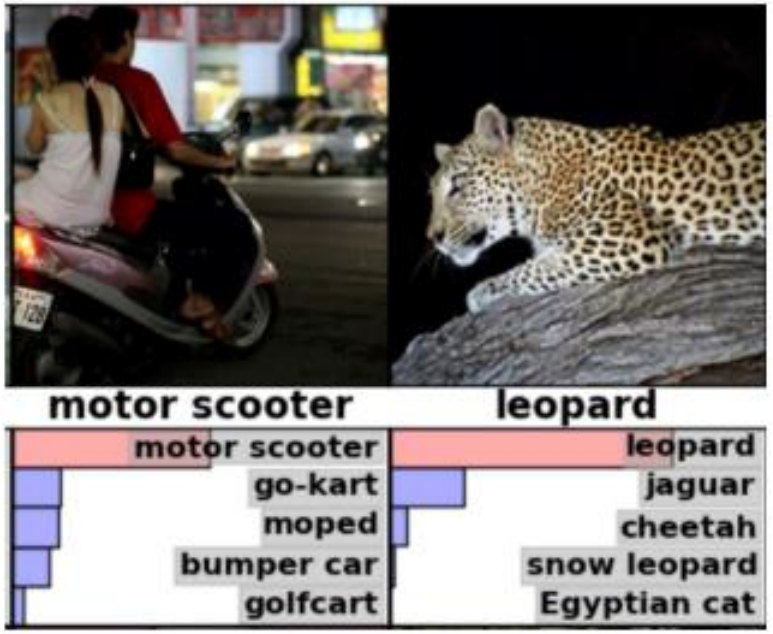


"max": max pooling
"norm": local response normalization
"full": fully connected

Figure: [Karnowski 2015] *(with corrections)*

ALEXNET- ESPECIFICAÇÕES

- 1000 categorias do ImageNet
- 1.2 milhões de imagens de treino
- 50.000 imagens de validação
- 150.000 imagens de teste



- 60M Parâmetros
- Treinado em duas GPUs GTX entre 5 e 6 dias

ALEXNET – PRINCIPAIS IDEIAS

- Usava **ReLu** – tornava convergência mais rápida
- Usava técnicas de **data augmentation**
- Implementou **dropout** para combater overfitting
- Treinou modelo usando **batch stochastic gradient descent**
- Usou **momentum** e **weight decay**

VGG Net - 2014

"Simple and deep"

Top-5 error rate of 7.3% on ImageNet

16 layer CNN - Best result - Conf. D

138 M parameters

Trained on 4 Nvidia Titan Black GPUs

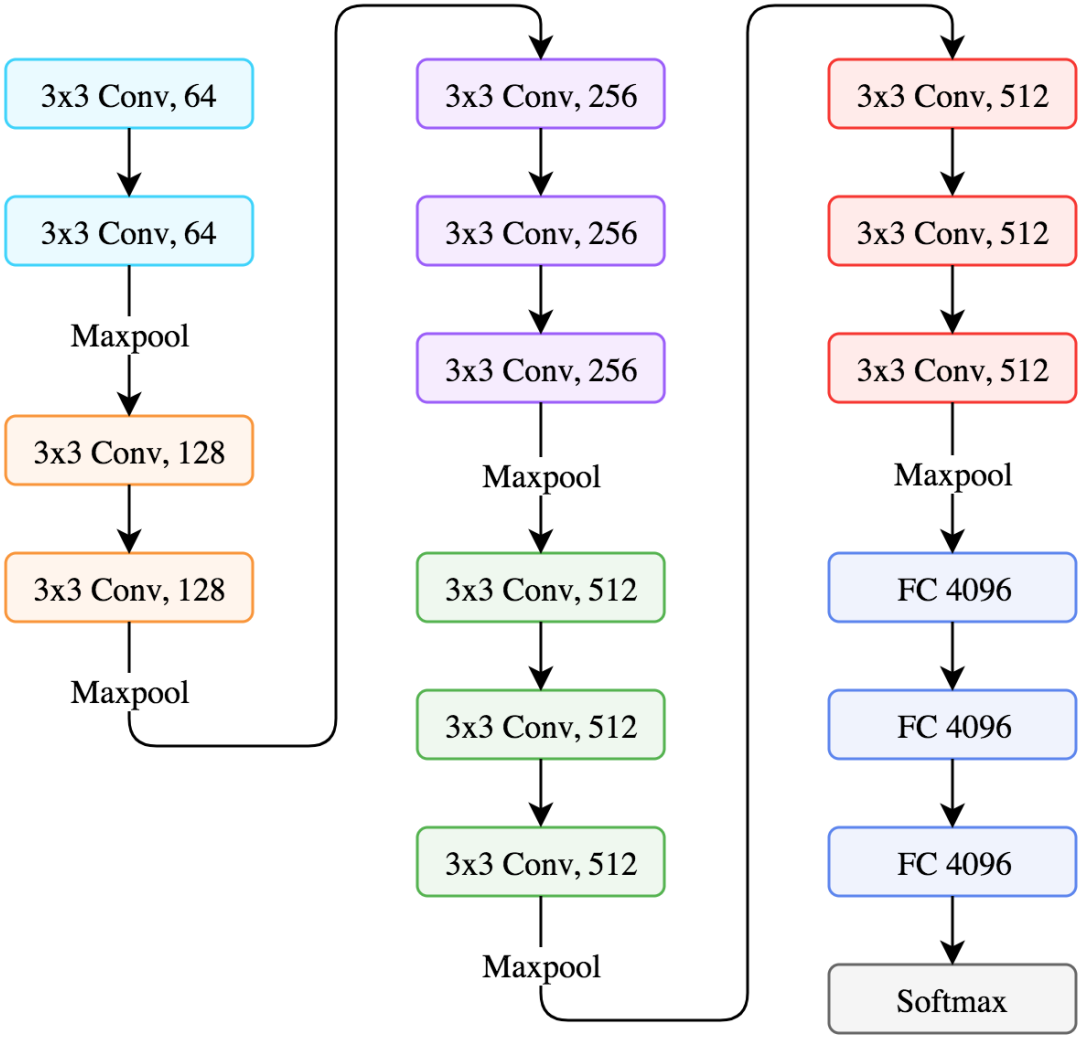
for two to three weeks.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG – IDEIAS PRINCIPAIS

- Usa apenas filtros 3x3
  - Usar várias vezes = maior campo receptivo
- Diminui a dimensão espacial e aumenta profundidade da rede
- Data augmentation com variação de escala
- Uso de Relu depois de cada camada convolutiva
- Número de parâmetros reduzidos
  - 3*3^2, ao invés de 7^2

VGG-16


3x3 Conv, 64
3x3 Conv, 64
Maxpool
3x3 Conv, 128
3x3 Conv, 128
Maxpool
3x3 Conv, 256
3x3 Conv, 256
3x3 Conv, 256
Maxpool
3x3 Conv, 512
3x3 Conv, 512
3x3 Conv, 512
Maxpool
3x3 Conv, 512
3x3 Conv, 512
3x3 Conv, 512
Maxpool
FC 4096
FC 4096
FC 4096
Softmax

VGGNET

[Simonyan and Zisserman, 2014]

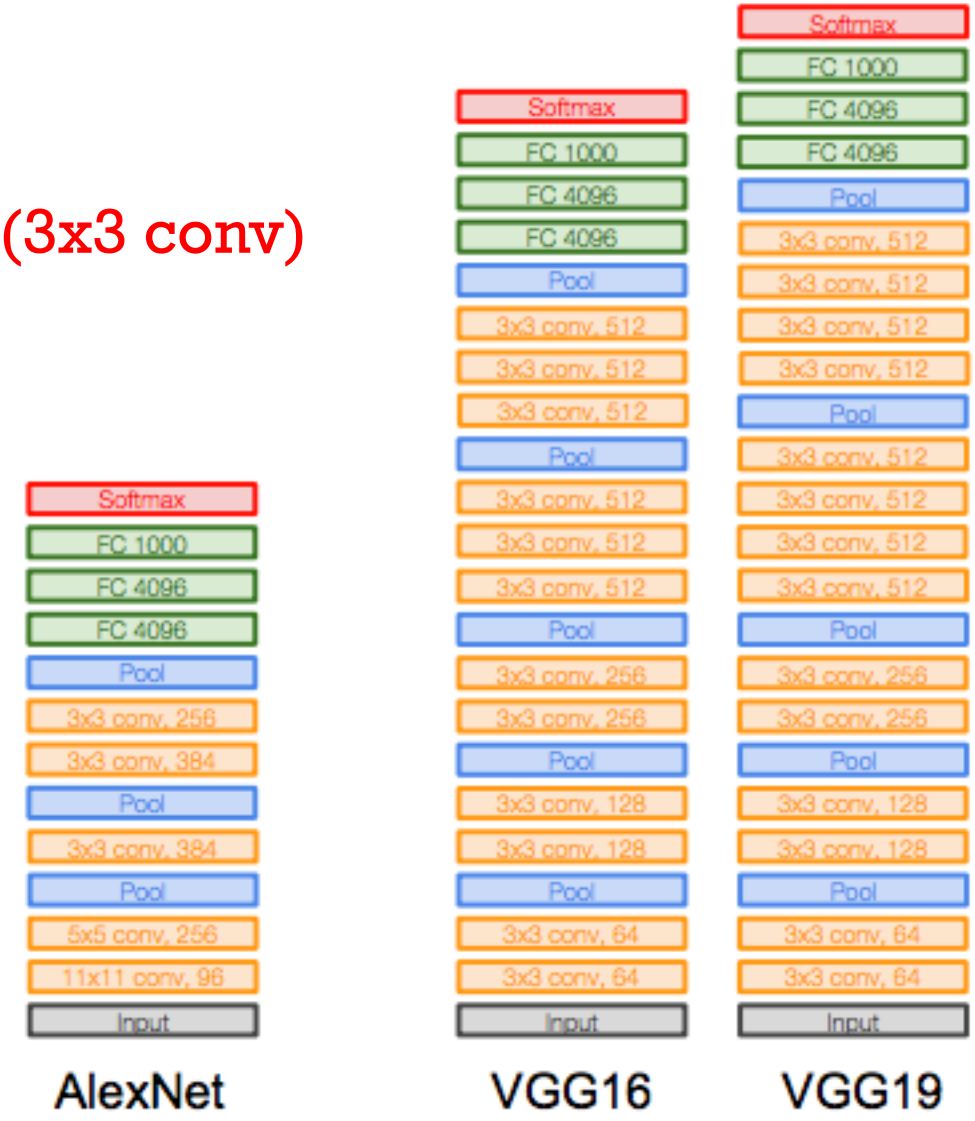
Por que usar filtros menores? (3x3 conv)

VGGNET

[Simonyan and Zisserman, 2014]

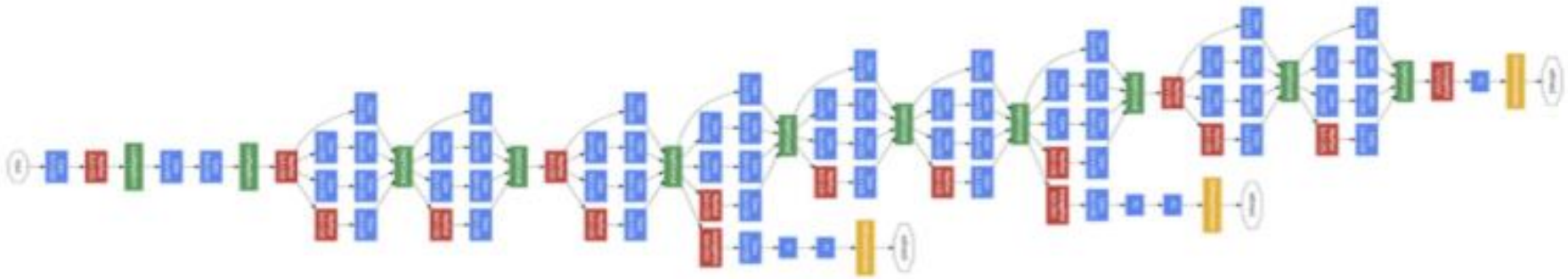
Por que usar filtros menores? (3x3 conv)

Enfileirar três camadas 3x3 conv (stride 1) tem o mesmo **campo receptivo efetivo** de uma camada 7x7 conv



GOOGLENET / INCEPTION (2014)

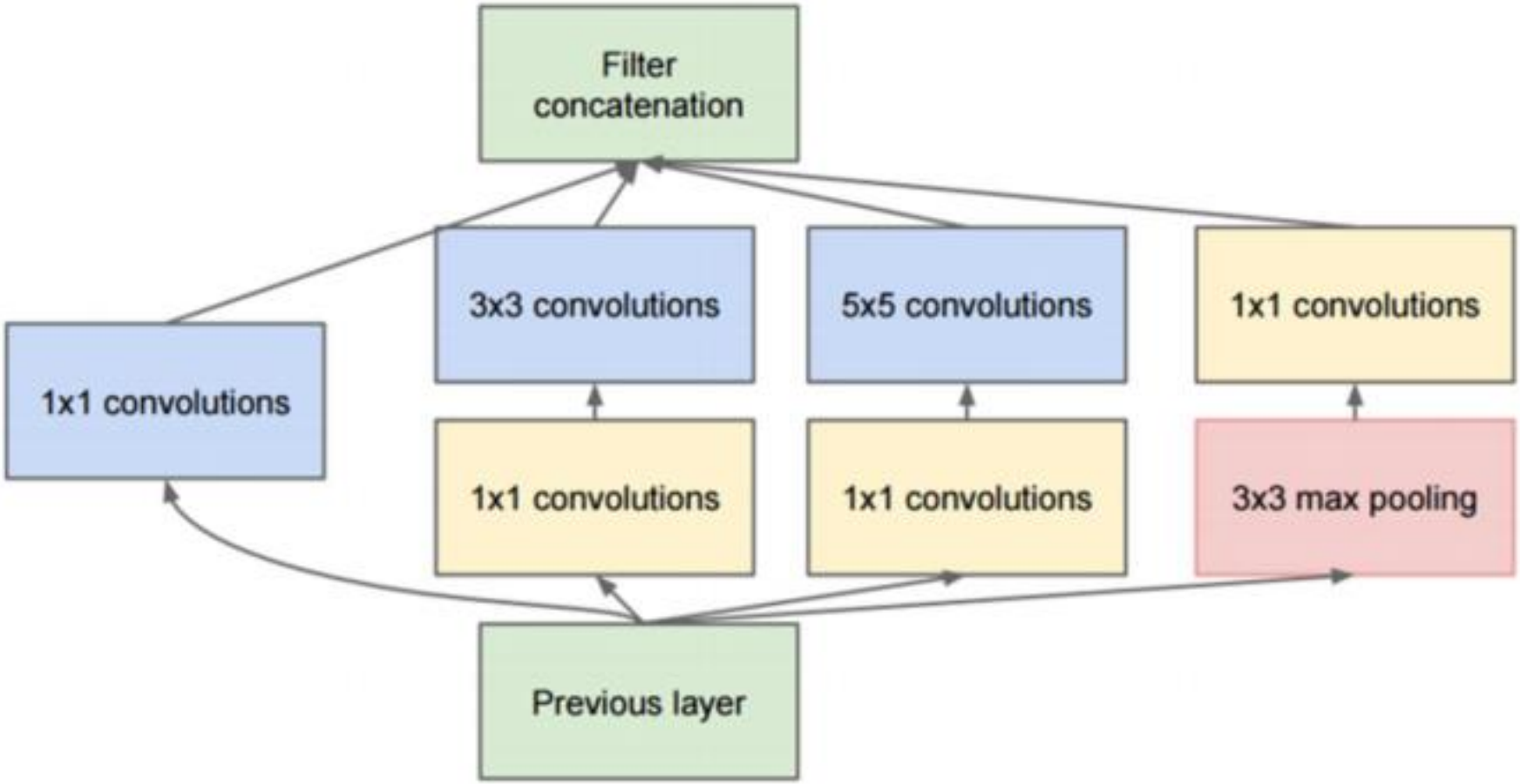
- Vencedora da ILSVRC 2014 com erro de 6.7% (4M parâmetros comparado com 60M do AlexNet)
- Treinado em uma semana (algumas GPUs high-end)



GOOGLENET – PRINCIPAIS IDEIAS

- Usa 9 módulos inceptions
- Não usa camadas FC
  - Usa average pool para transformar 7x7x1024 em 1x1x1024
- Usa 12x menos parâmetros que AlexNet

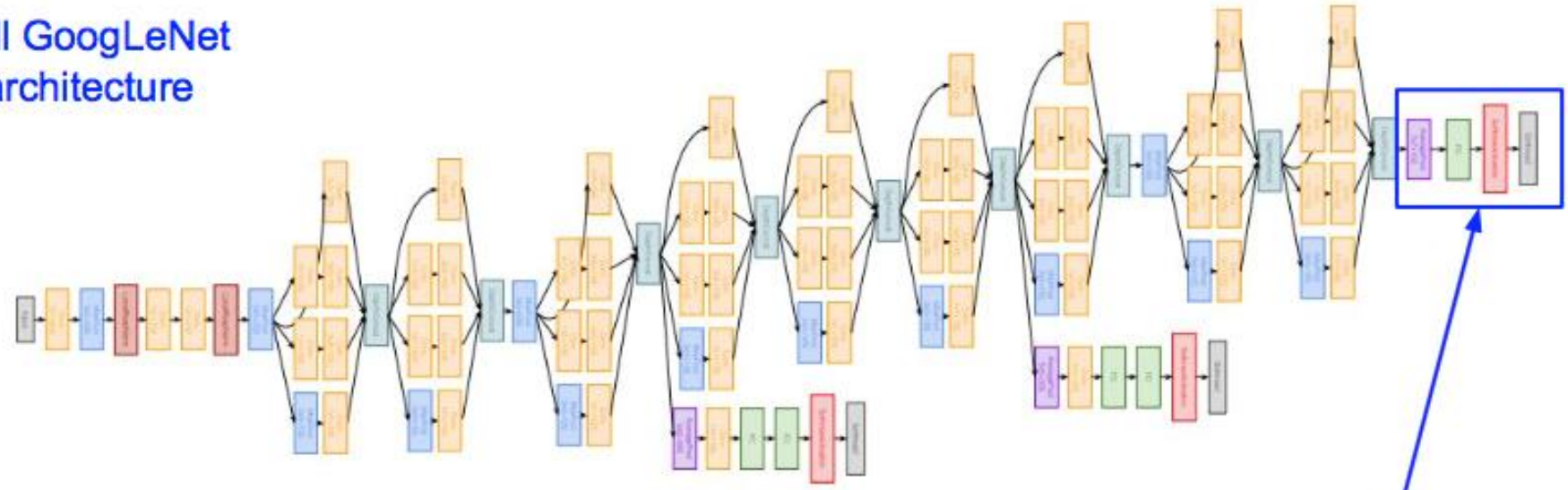
The Inception Module - A closer look



Case Study: GoogLeNet

[Szegedy et al., 2014]

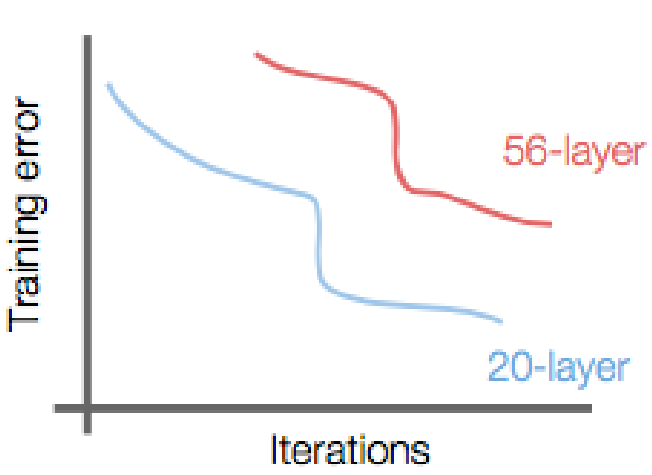
Full GoogLeNet architecture

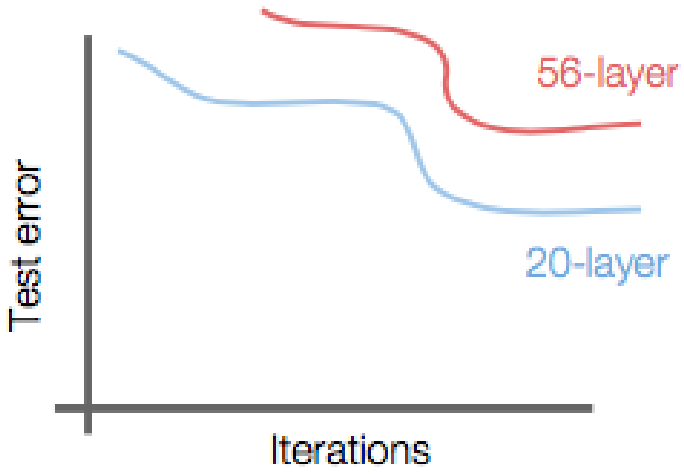


Classifier output
(removed expensive FC layers!)

RESNET (MICROSOFT) [HE ET AL., 2015]

- O que acontece quando continuamos a empilhar mais camadas profundas na CNN?





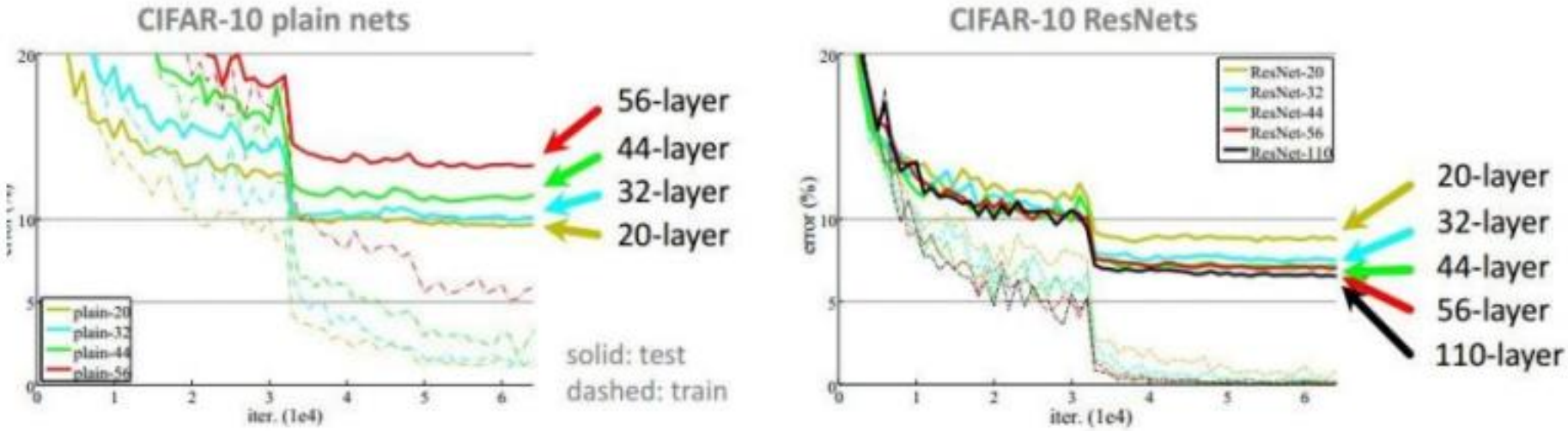
O modelo mais profundo é pior, mas não é por causa de overfitting

RESNET (MICROSOFT) [HE ET AL., 2015]

- Resultados
  - Capaz de treinar uma *very deep network* (152 camadas) sem perder desempenho
  - Primeiro lugar em todas as competições do ILSVRC e COCO
  - 3.6% de erro no ILSVRC (melhor que desempenho humano)

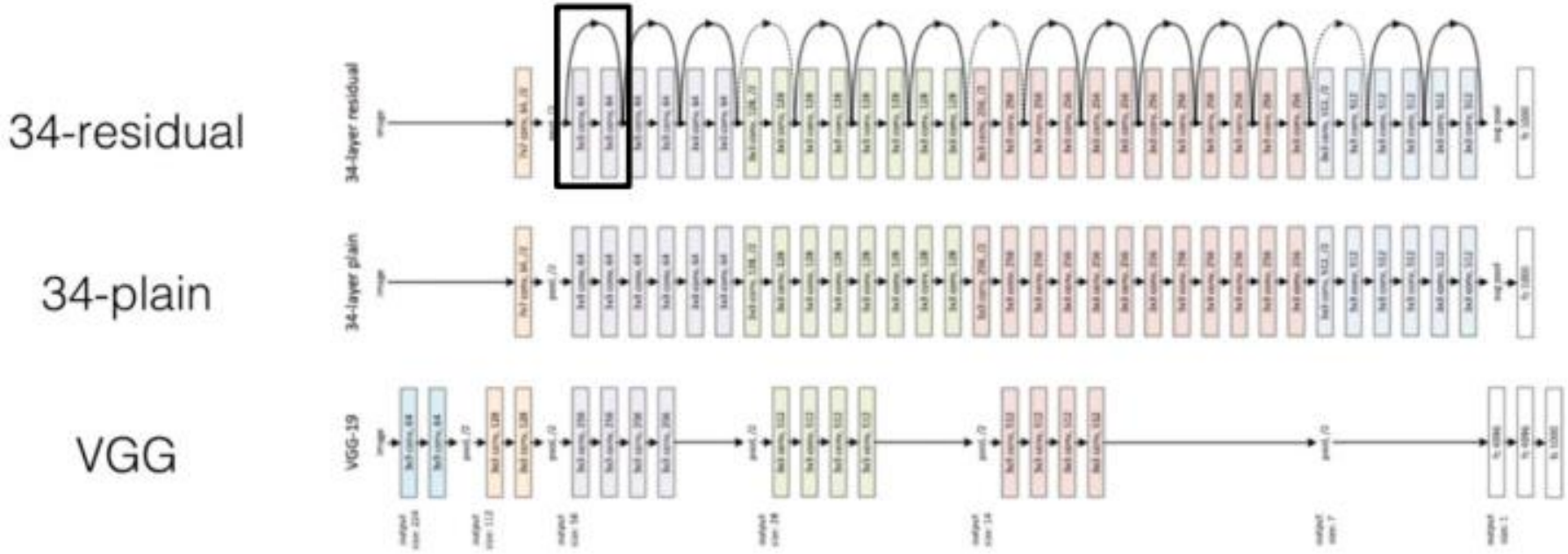
Going deeper

Performance of ResNets versus plain-nets as depth is increased

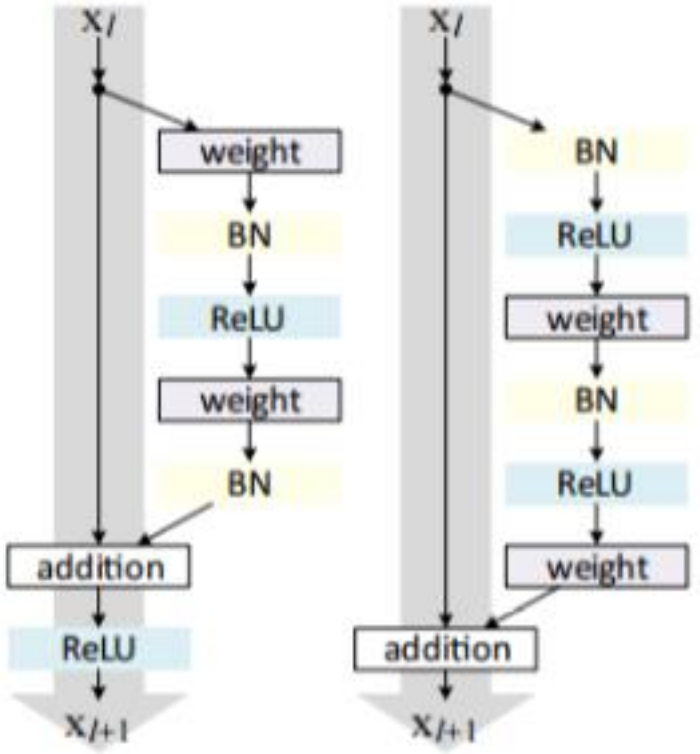


Microsoft ResNet 2015

ResNet won ILSVRC 2015 with an incredible error rate of 3.6%
Humans usually hover around 5-10%
Trained on an 8 GPU machine for two to three weeks.

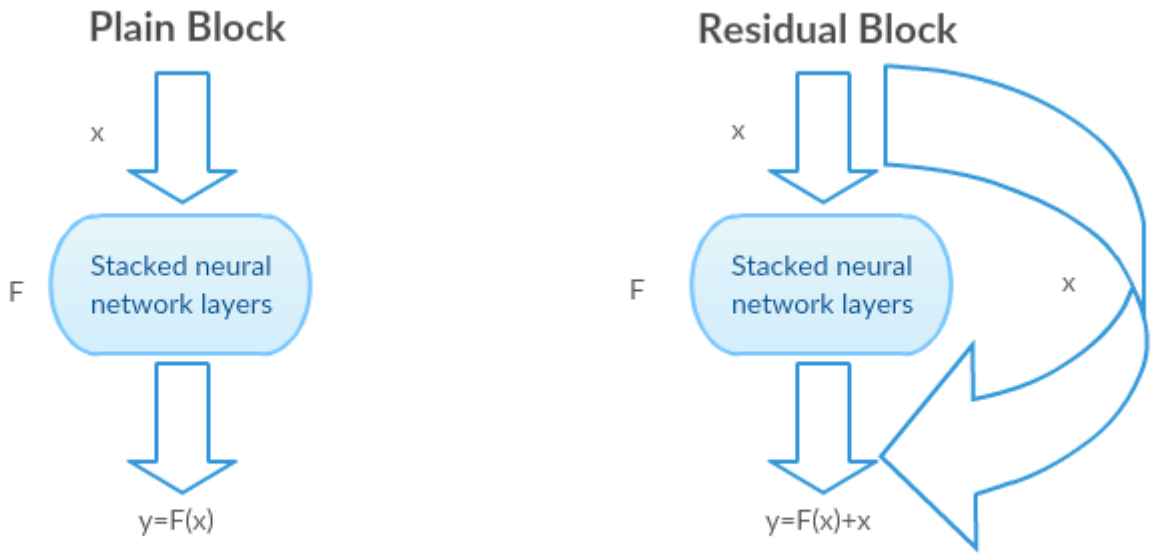


RESNET



(a) original (b) proposed

RESNET



Hard to get F(x)=x and make y=x an identity mapping

Easy to get F(x)=0 and make y=x an identity mapping

RESNET

- Treinamento
  - Batch Normalization depois de toda camada convolutiva
  - Inicialização usando algoritmo de Xavier (He et al.)
  - SGD + Momentum (0.9)
  - Taxa de aprendizado: 0.1
  - Mini-batch size 256
  - Weight decay de 1e-5
  - Sem dropout

REVOLUÇÃO DA PROFUNDIDADE

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

TRANSFER LEARNING E FINE TUNING

É PRECISO TREINAR A REDE TODA VEZ?

- Transfer learning!

MODEL ZOOS (TENSORFLOW)

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	inception_v1_2016_08_28.tar.gz	69.8	89.6
Inception V2	Code	inception_v2_2016_08_28.tar.gz	73.9	91.8
Inception V3	Code	inception_v3_2016_08_28.tar.gz	78.0	93.9
Inception V4	Code	inception_v4_2016_09_09.tar.gz	80.2	95.2
Inception-ResNet-v2	Code	inception_resnet_v2_2016_08_30.tar.gz	80.4	95.3
ResNet V1 50	Code	resnet_v1_50_2016_08_28.tar.gz	75.2	92.2
ResNet V1 101	Code	resnet_v1_101_2016_08_28.tar.gz	76.4	92.9
ResNet V1 152	Code	resnet_v1_152_2016_08_28.tar.gz	76.8	93.2
ResNet V2 50^	Code	resnet_v2_50_2017_04_14.tar.gz	75.6	92.8
ResNet V2 101^	Code	resnet_v2_101_2017_04_14.tar.gz	77.0	93.7
ResNet V2 152^	Code	resnet_v2_152_2017_04_14.tar.gz	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	vgg_16_2016_08_28.tar.gz	71.5	89.8
VGG 19	Code	vgg_19_2016_08_28.tar.gz	71.1	89.8
MobileNet_v1_1.0_224	Code	mobilenet_v1_1.0_224_2017_06_14.tar.gz	70.7	89.5

/sl

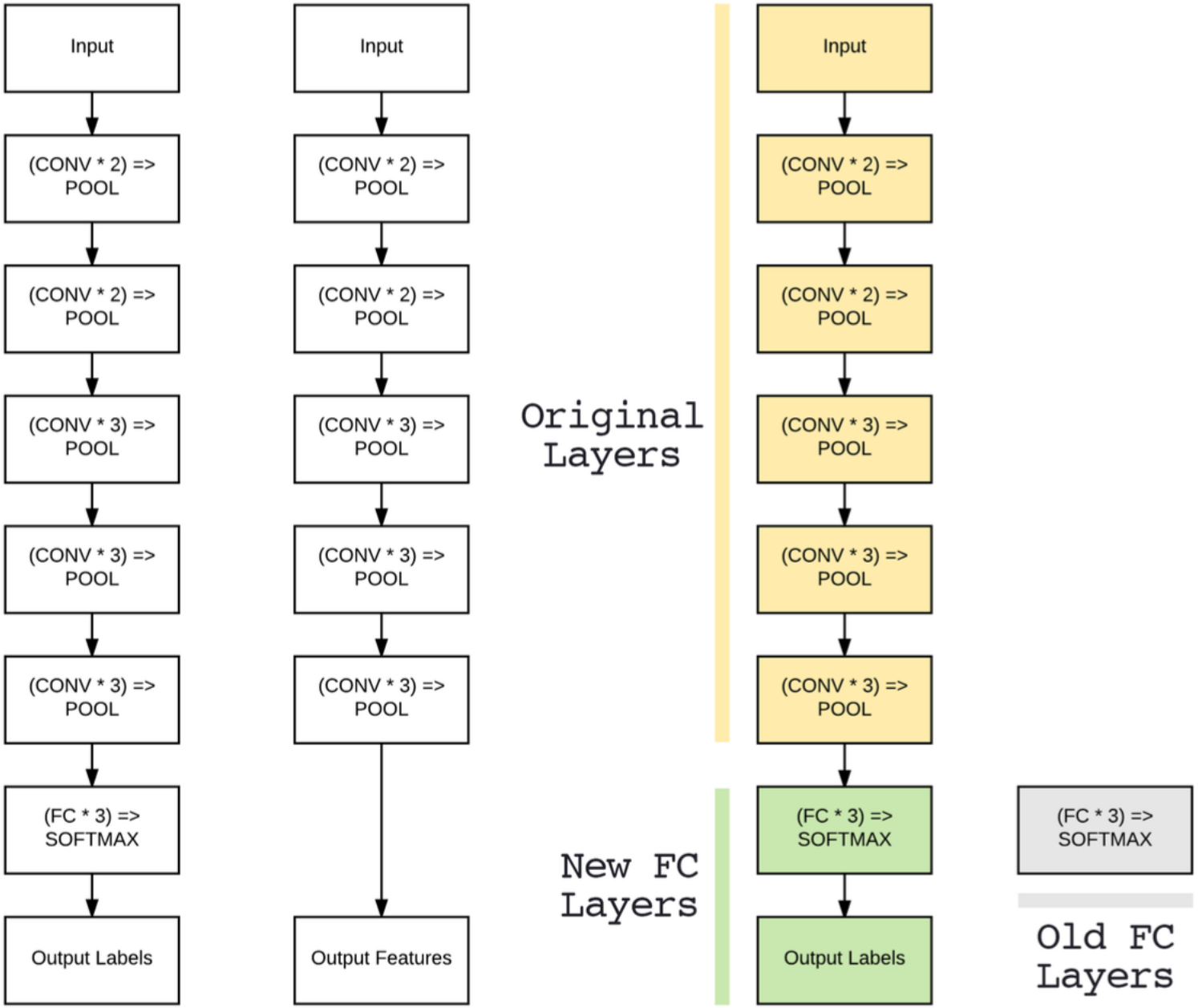
KERAS

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

RESNET 50 (RESNET50.PY)

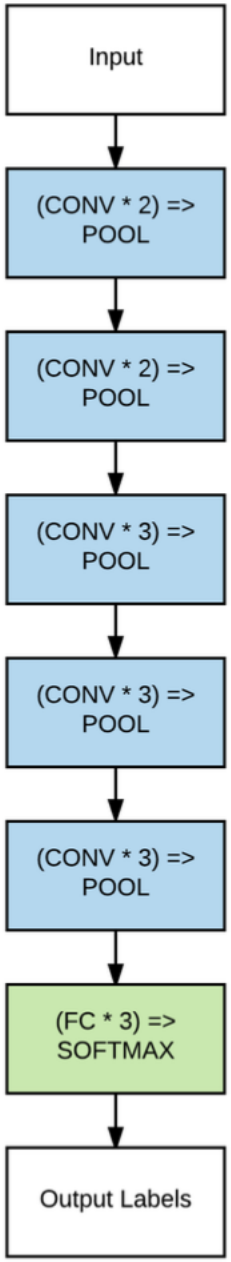
```
1  from keras.applications.resnet50 import ResNet50
2  from keras.preprocessing import image
3  from keras.applications.resnet50 import preprocess_input, decode_predictions
4  import numpy as np
5
6  model = ResNet50(weights='imagenet')
7  img_path = 'elephant.jpg'
8  img = image.load_img(img_path, target_size=(224, 224))
9
10 x = image.img_to_array(img)
11 x = np.expand_dims(x, axis=0)
12 x = preprocess_input(x)
13
14 preds = model.predict(x)
15 P = decode_predictions(preds)
16
17 for (i, (imagenetID, label, prob)) in enumerate(P[0]):
18     print("{}. {}: {:.2f}%".format(i + 1, label, prob * 100))
```

TRANSFER LEARNING

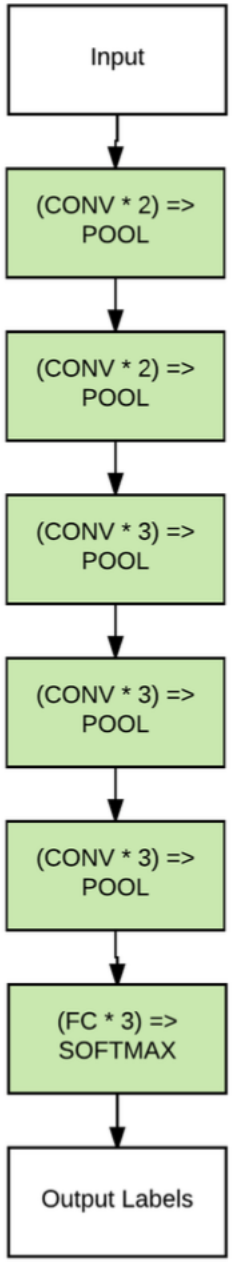

Input
(CONV * 2) => POOL
(CONV * 2) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
(FC * 3) => SOFTMAX
Output Labels
Input
(CONV * 2) => POOL
(CONV * 2) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
Output Features
Original Layers
Input
(CONV * 2) => POOL
(CONV * 2) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
(CONV * 3) => POOL
New FC Layers
(FC * 3) => SOFTMAX
Output Labels
(FC * 3) => SOFTMAX
Old FC Layers

Freeze Early Layers in Network

Only Train FC Layers

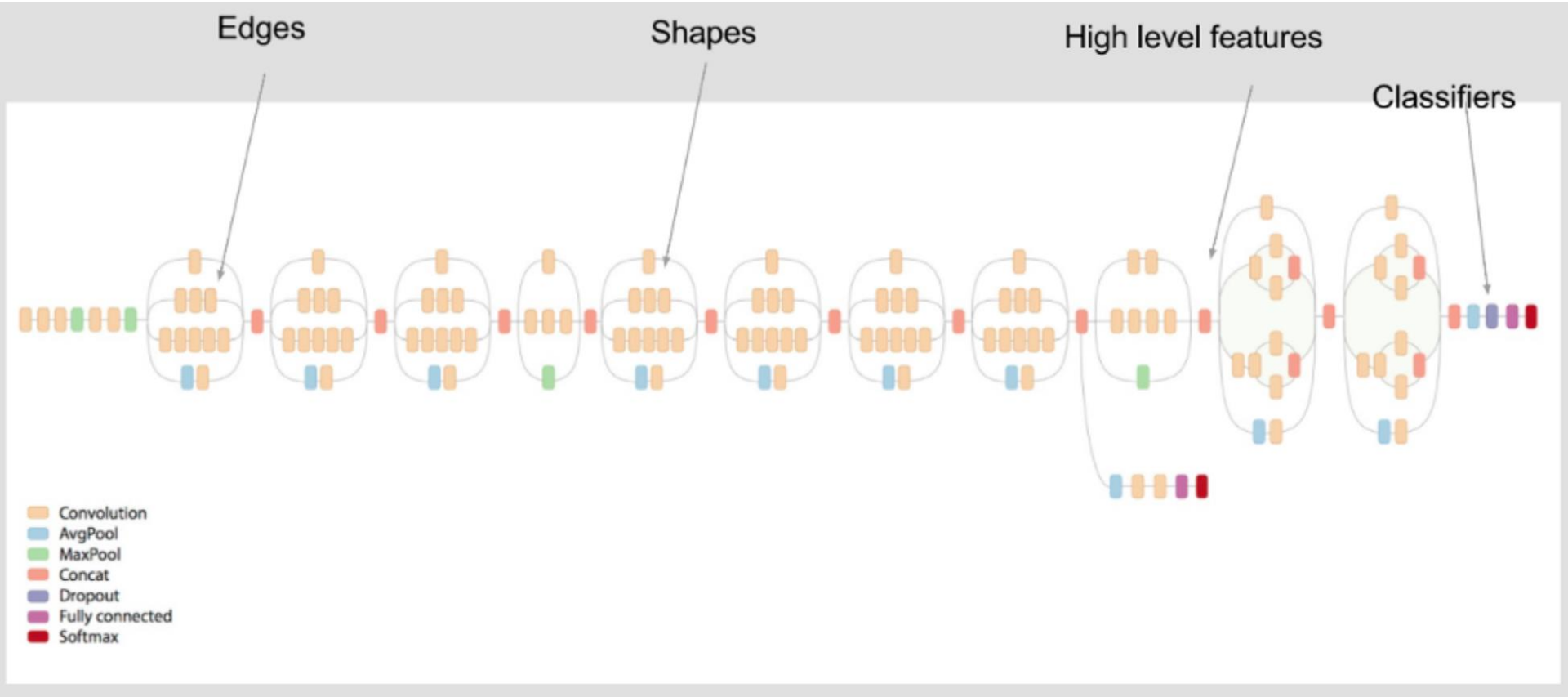


Unfreeze Early Layers & Train All



CÓDIGO

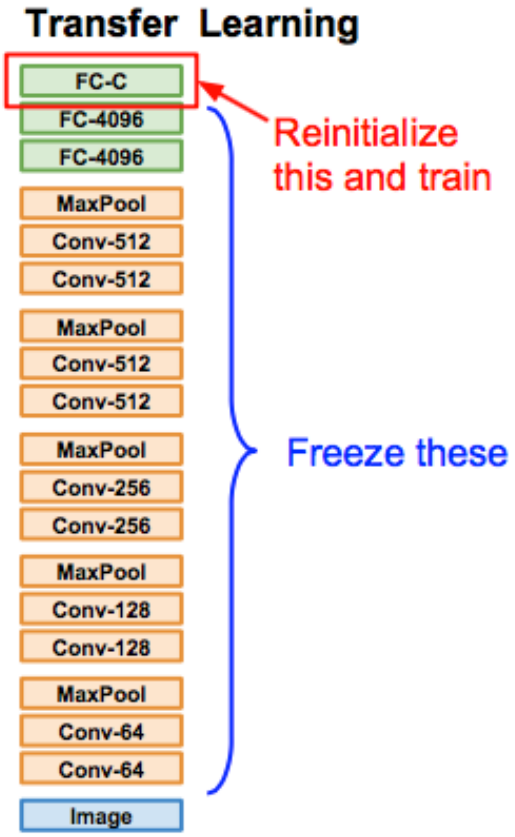
- Fine_tuning.ipynb


Edges
Shapes
High level features
Classifiers
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

TRANSFER LEARNING

TF-Slim: (https://github.com/tensorflow/models/tree/master/slim/nets)

Keras: (https://github.com/fchollet/deep-learning-models)



COMO USAR TRANSFER LEARNING

	Base Similar	Base diferente
Base pequena	Usar modelo treinado	Treinar mais camadas
Base grande	Treinar camadas FC	treinar próprio modelo

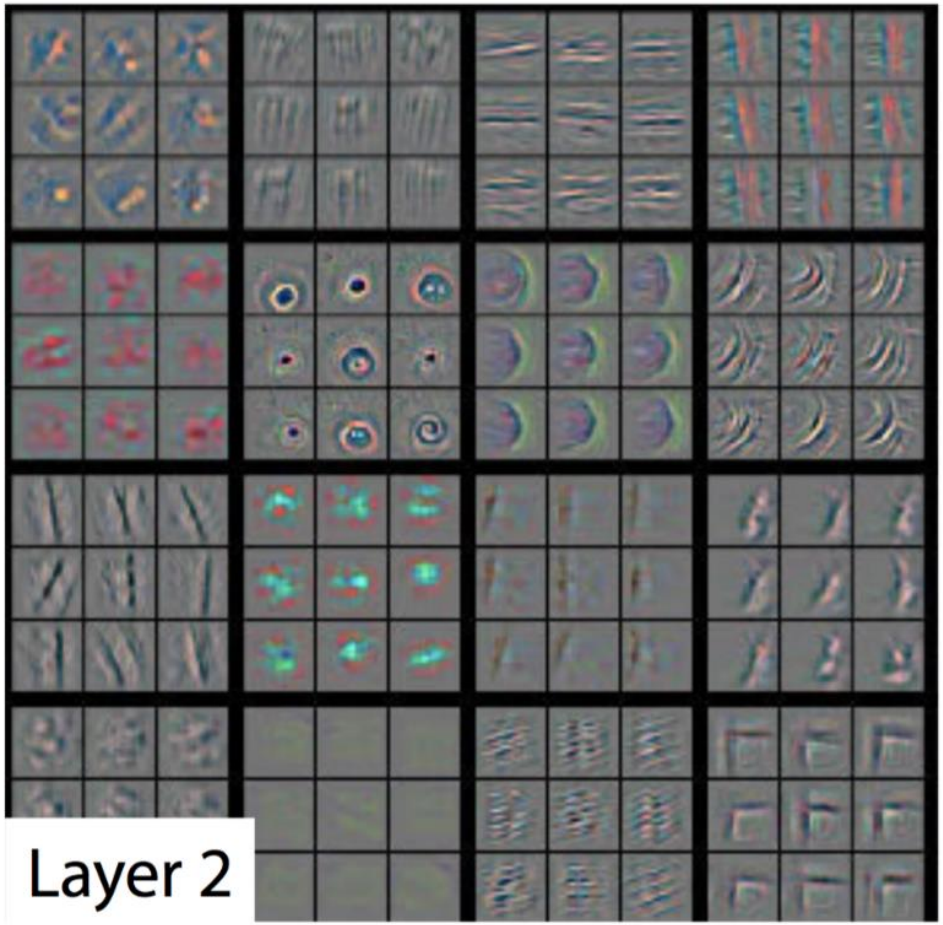
CÓDIGO

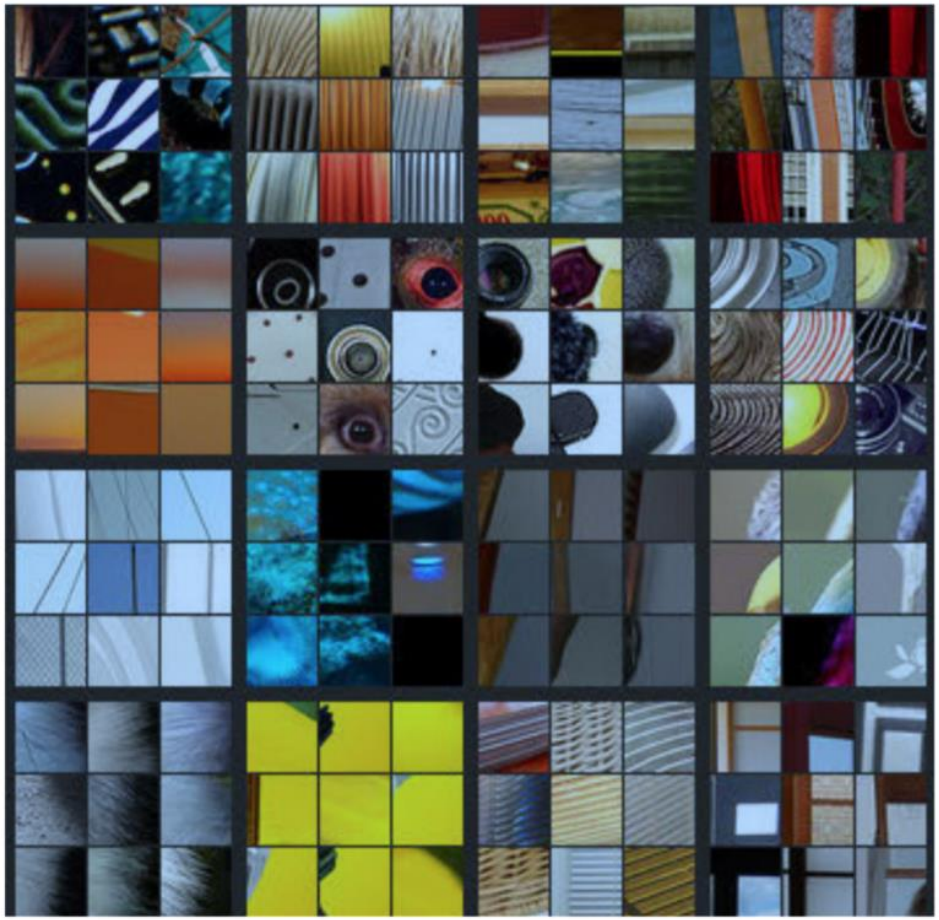
- Transfer_learning_answer.ipynb

```
from keras import applications
from keras.layers import Input
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras.models import Sequential, Model


 img_width, img_height = 48, 48
```

```
#model = ShallowNet.build(width=32, height=32, depth=3, classes=3)

model = applications.VGG16(weights = "imagenet", include_top=False,
    input_shape = (img_width, img_height, 3))

# Freeze the layers which you don't want to train. Here I am
#freezing the first 5 layers.
#for layer in model.layers[:5]:
for layer in model.layers:
    layer.trainable = False

#Adding custom Layers
x = model.output
x = Flatten()(x)
x = Dense(256, activation="relu")(x)
predictions = Dense(3, activation="softmax")(x)

# creating the final model
model_final = Model(input = model.input, output = predictions)
model = model_final

#compila
```

Layer 2





TRANSFER LEARNING

Guide for How to Use Transfer Learning