

Redes Convolucionais para classificação de imagens

Aprendizagem de máquina e aprendizado
profundo - Lux.AI

INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



CNNs

MLP com imagens

- Queremos classificar uma imagem de tamanho 640x480
 - ◆ Temos no total 307.200 pixels
 - ◆ Se a imagem for RGB, cada pixel é composto de 3 intensidades, então
 $m=307.200 * 3 = 921.600$

MLP com imagens

- Queremos classificar uma imagem de tamanho 640x480
 - ◆ Temos no total 307.200 pixels
 - ◆ Se a imagem for RGB, cada pixel é composto de 3 intensidades, então
 $n = 307.200 * 3 = 921.600$
- Para classificar um vetor de features deste tamanho precisamos ter um número considerável de camadas
 - ◆ Podemos reduzir a dimensionalidade, mas perderíamos muita informação da entrada original
 - ◆ Ainda, a MLP recebe como entrada um vetor de dimensão 1xN, então perdemos a relação espacial entre os pixels que a imagem original fornece

MLP com imagens

- Para uma MLP com 3 camadas intermediárias de tamanho $H=[20,10,2]$.
 - ◆ Cada camada deve possuir $[20*n + n, 10*20 + 20, 2*10 + 10]$ pesos, respectivamente
 - ◆ Se n for 640×480 , a primeira camada intermediária possuirá 19.353.600 parâmetros para 20 neurônios apenas
 - ◆ Além disso, no backpropagation ainda precisamos armazenar os gradientes para cada um destes parâmetros

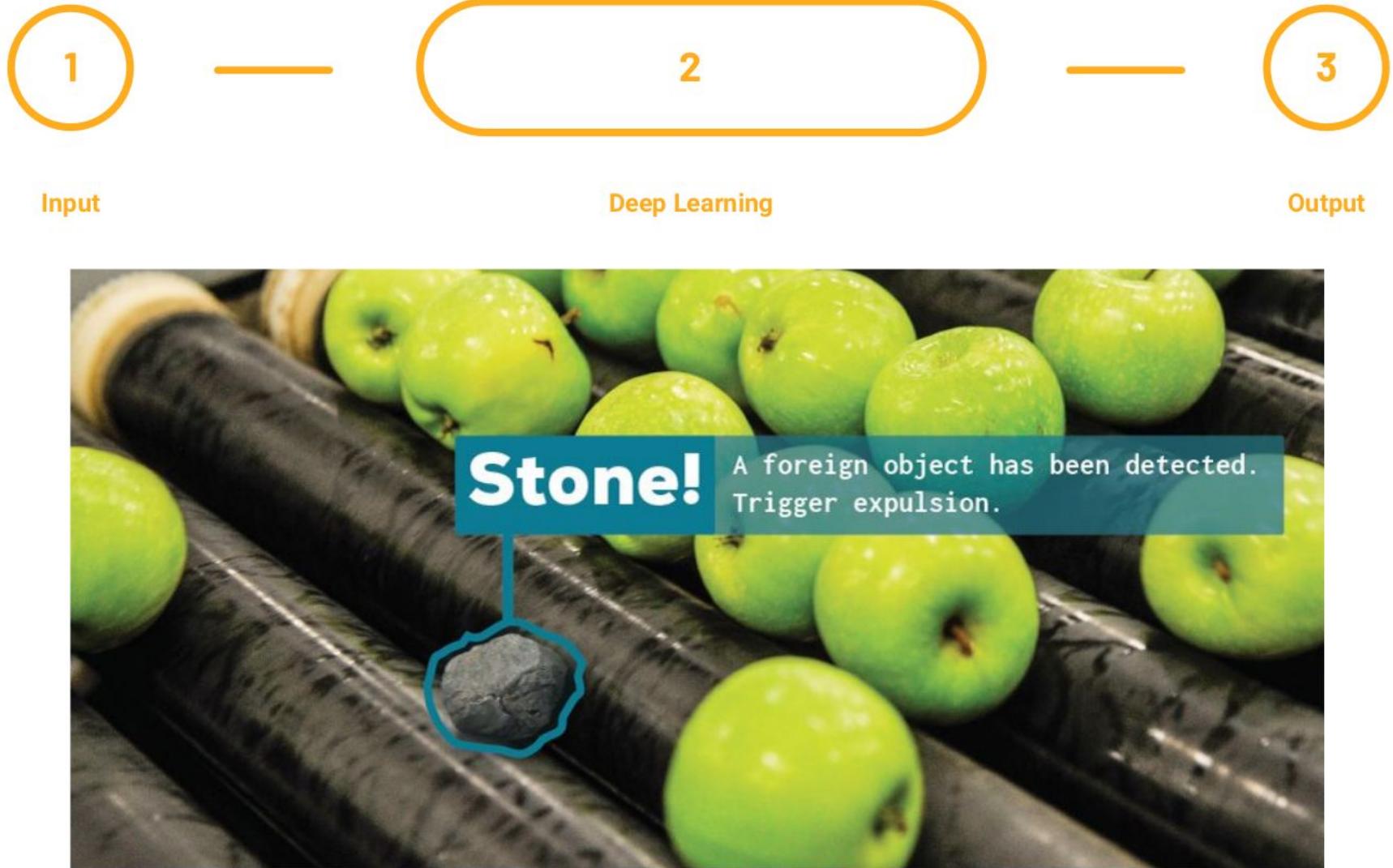
Introdução CNNs

- Com isto percebemos a limitação dos perceptrons multicamadas para imagens.
 - ◆ Uso de redes convolucionais vem com objetivo de suprir esse déficit, uma vez que suas camadas adquirem informações na região de interesse e seus parâmetros são independente da dimensão espacial da entrada

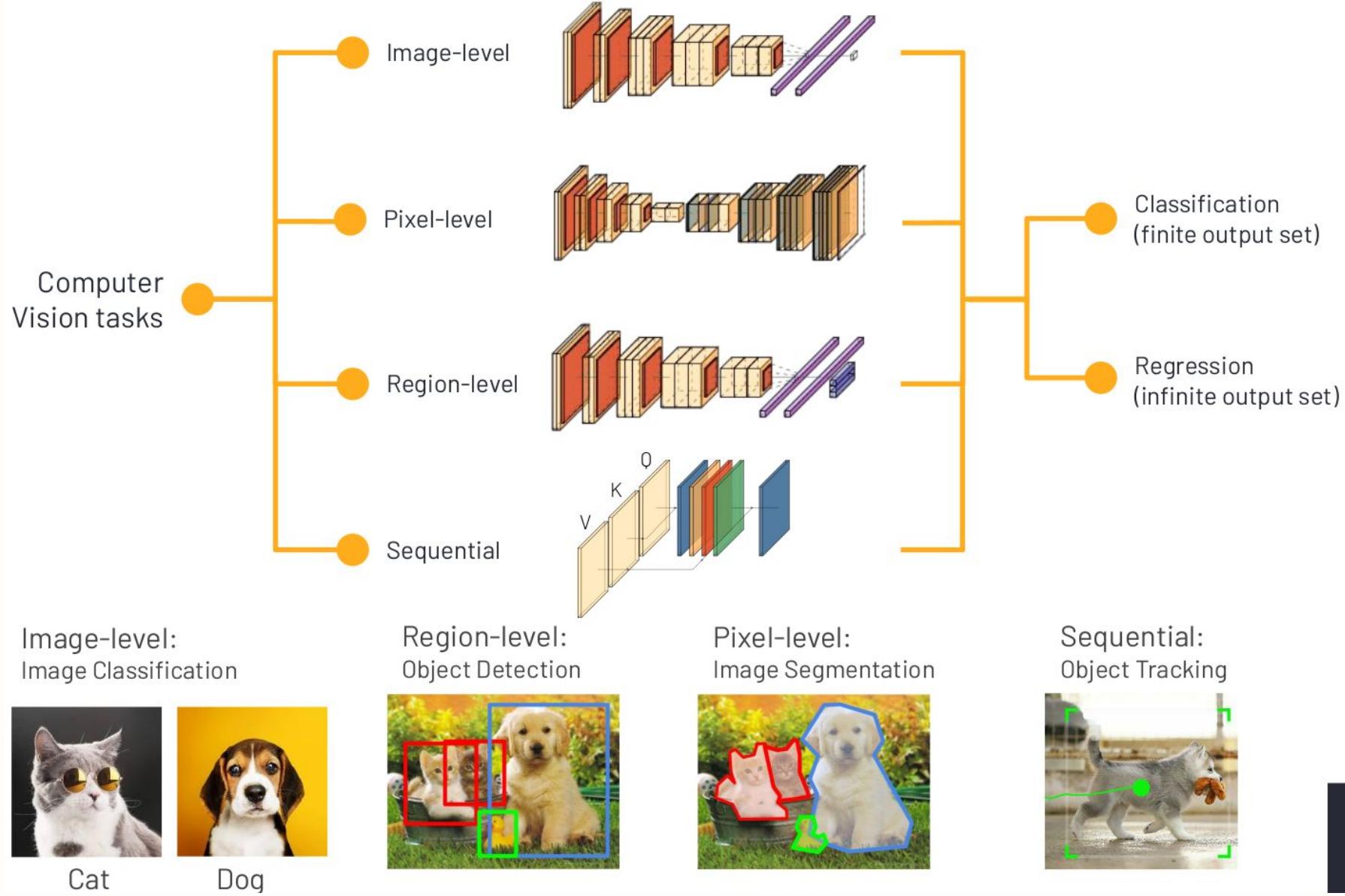
Redes Convolucionais



Redes Convolucionais



Aplicações



Notação - Objetos

Real numbers set \mathbb{R}

Cartesian product $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$

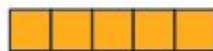
Integer numbers set \mathbb{Z}

Component i-th, j-th $x(i, j)$

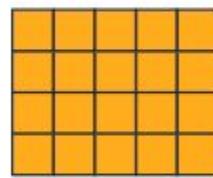
Scalar $x \in \mathbb{R}$



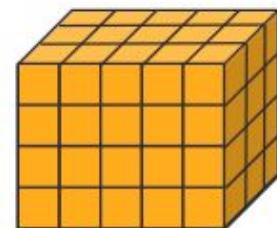
Vector $x: \mathbb{R} \rightarrow \mathbb{R}$



Matrix $x: \mathbb{R}^2 \rightarrow \mathbb{R}$



Tensor $x: \mathbb{R}^n \rightarrow \mathbb{R}$



Rank: 0

Resolution: 1

Rank: 1

Resolution: N

Rank: 2

Resolution: NxM

Rank: n

Resolution: NxMx...

Notação - Imagem n-dimensional

Image $I: \Omega \rightarrow \mathbb{R}^{d_c}$

Pixel/Voxel $p \in \Omega \subset \mathbb{R}^{d_i}$

$$d_c = 3, d_i = 2$$



$$I(p) = (0.98, 0.84, 0.15)$$

Notação - Operações

Sum
$$z(i, j, \dots, k) = x(i, j, \dots, k) + y(i, j, \dots, k)$$

Reduction sum
$$z = \sum_{i=1}^{d_0} \sum_{j=1}^{d_1} \dots \sum_{k=1}^{d_n} x(i, j, \dots, k)$$

Hadamard product
$$z(i, j, \dots, k) = x(i, j, \dots, k) \cdot y(i, j, \dots, k)$$

Matrix product
$$z(i, j) = \sum_{k=1}^{d_0} x(i, k) \cdot y(k, j)$$

Dot product
$$z = \sum_{i=1}^{d_0} x(i) \cdot y(i)$$

Convolution*
$$z(p, q, \dots, r) = \sum_{i=-d_0}^{d_0} \sum_{j=-d_1}^{d_1} \dots \sum_{k=-d_n}^{d_n} x(p+i, q+j, \dots, r+k) \cdot y(i, j, \dots, k)$$

* Misnomer for Cross-Correlation operation

Convolução

- Convolução é uma operação matemática que descreve uma regra de como combinar duas funções ou informações para formar uma terceira função.

$$(f * g)(x) = \int f(\mathbf{z}) g(x - \mathbf{z}) d\mathbf{z}$$

Convolução

→ No caso discreto

$$(f * g)(i) = \sum_a f(a) g(i - a)$$

→ Para 2 dimensões temos:

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b)$$

Convolução

$$z(p, q, \dots, r) = \sum_{i=-d_0}^{d_0} \sum_{j=-d_1}^{d_1} \dots \sum_{k=-d_n}^{d_n} x(p+i, q+j, \dots, r+k) \cdot y(i, j, \dots, k)$$

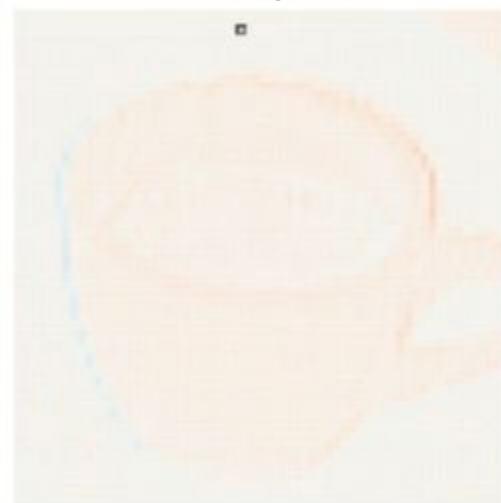
Input



Kernel

0.09	-0.01	0.08
-0.42	0.16	-0.02
-0.32	0.37	-0.08

Output



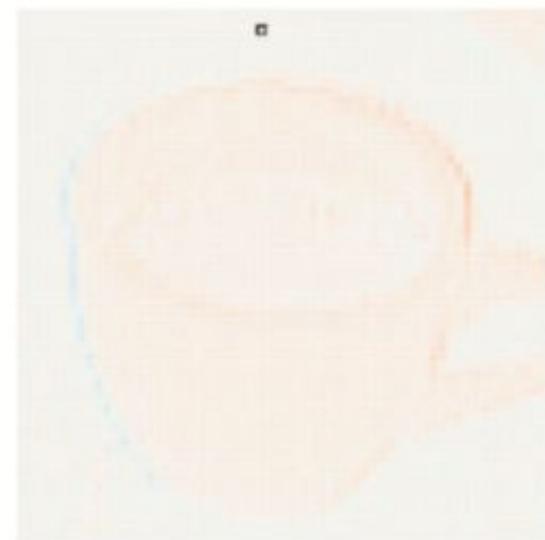
Convolução

$$z(p, q, \dots, r) = \sum_{i=-d_0}^{d_0} \sum_{j=-d_1}^{d_1} \dots \sum_{k=-d_n}^{d_n} x(p+i, q+j, \dots, r+k) \cdot y(i, j, \dots, k)$$

Input (64, 64)



Output (62, 62)



$$\begin{array}{c} 0.16 + 0.15 + 0.16 + \\ \times 0.09 \quad \times -0.01 \quad \times 0.08 \\[10pt] 0.16 + 0.17 + 0.17 + \\ \times -0.42 \quad \times 0.16 \quad \times -0.02 \\[10pt] 0.17 + 0.17 + 0.17 = \\ \times -0.32 \quad \times 0.37 \quad \times -0.08 \\[10pt] -0.03 \end{array}$$

The diagram illustrates the convolution process. It shows three 3x3 kernel weights (0.16, 0.15, 0.16; 0.16, 0.17, 0.17; 0.17, 0.17, 0.17) being applied to a 3x3 receptive field in the input image. The input values are multiplied by the kernel weights, and the results are summed. The final output value is -0.03.

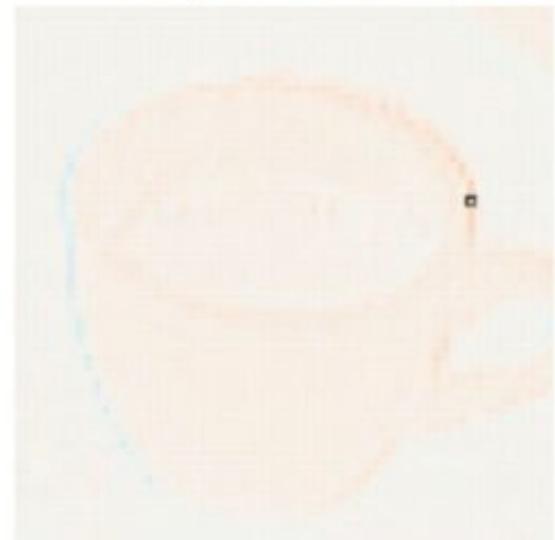
Convolução

$$z(p, q, \dots, r) = \sum_{i=-d_0}^{d_0} \sum_{j=-d_1}^{d_1} \dots \sum_{k=-d_n}^{d_n} x(p+i, q+j, \dots, r+k) \cdot y(i, j, \dots, k)$$

Input (64, 64)



Output (62, 62)



$$\begin{array}{r} 0.77 \quad + \quad 0.21 \quad + \quad 0.11 \quad + \\ \times 0.09 \quad \times -0.01 \quad \times 0.08 \\ \hline 0.81 \quad + \quad 0.2 \quad + \quad 0.18 \quad + \\ \times -0.42 \quad \times 0.16 \quad \times -0.02 \\ \hline 0.76 \quad + \quad 0.32 \quad + \quad 0.13 \quad = \\ \times -0.32 \quad \times 0.37 \quad \times -0.08 \\ \hline -0.37 \end{array}$$

Convolução

Input

0	1	2
3	4	5
6	7	8

Kernel

0	1
2	3

Output

19	25
37	43

Input image



Convolution
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



Convolução

Input	Kernel	Output													
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td><td style="padding: 2px;">2</td></tr><tr><td style="padding: 2px;">3</td><td style="padding: 2px;">4</td><td style="padding: 2px;">5</td></tr><tr><td style="padding: 2px;">6</td><td style="padding: 2px;">7</td><td style="padding: 2px;">8</td></tr></table>	0	1	2	3	4	5	6	7	8	$*$	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">0</td><td style="padding: 2px;">1</td></tr><tr><td style="padding: 2px;">2</td><td style="padding: 2px;">3</td></tr></table>	0	1	2	3
0	1	2													
3	4	5													
6	7	8													
0	1														
2	3														
	$=$	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td style="padding: 2px;">19</td><td style="padding: 2px;">25</td></tr><tr><td style="padding: 2px;">37</td><td style="padding: 2px;">43</td></tr></table>	19	25	37	43									
19	25														
37	43														

→ A convolução pode ser interpretada como um filtro, onde o kernel filtra o mapa de ativação (entrada) para obter determinadas informações



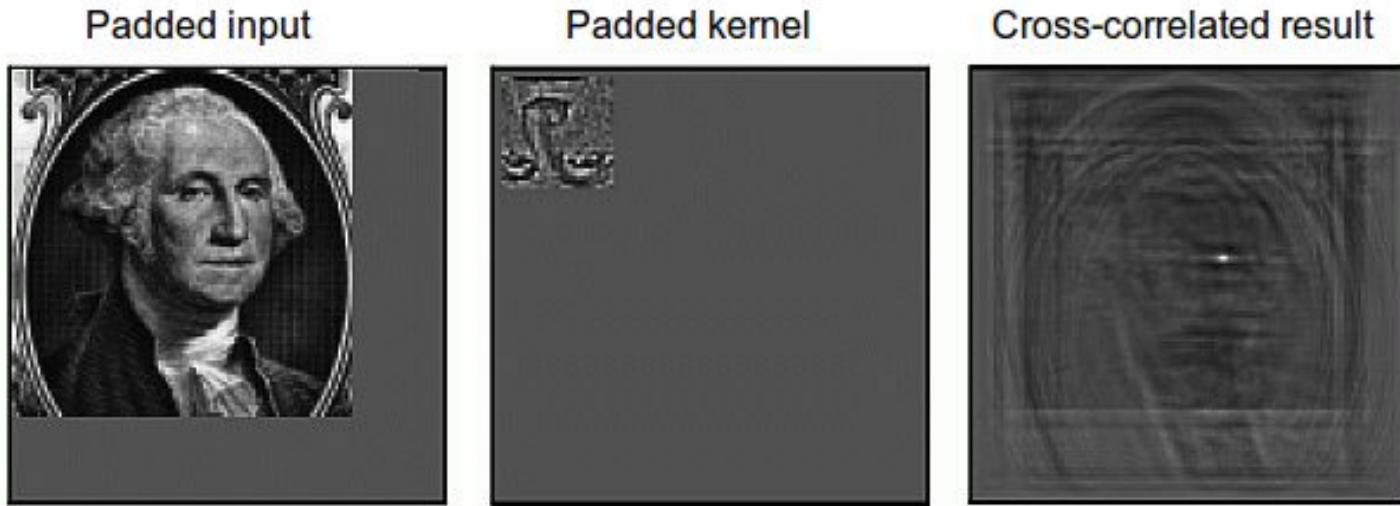
Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



Convolução



Cross-correlation via convolution: The input and kernel are padded with zeros and the kernel is rotated by 180 degrees. The white spot marks the area with the strongest pixel-wise correlation between image and kernel. Note that the output image is in the spatial domain, the inverse Fourier transform was already applied. Images taken from [Steven Smith's excellent free online book](#) about digital signal processing.

Convolução



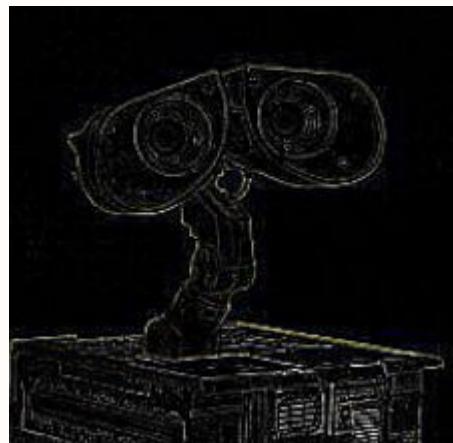
Motion blur



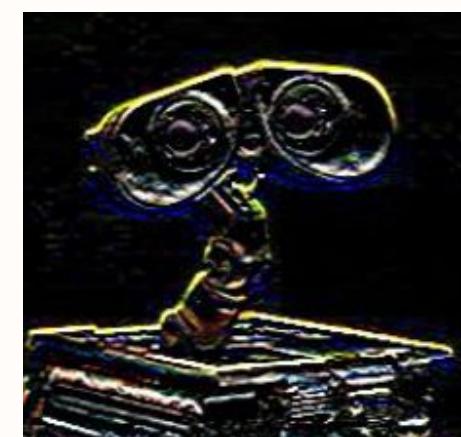
Average



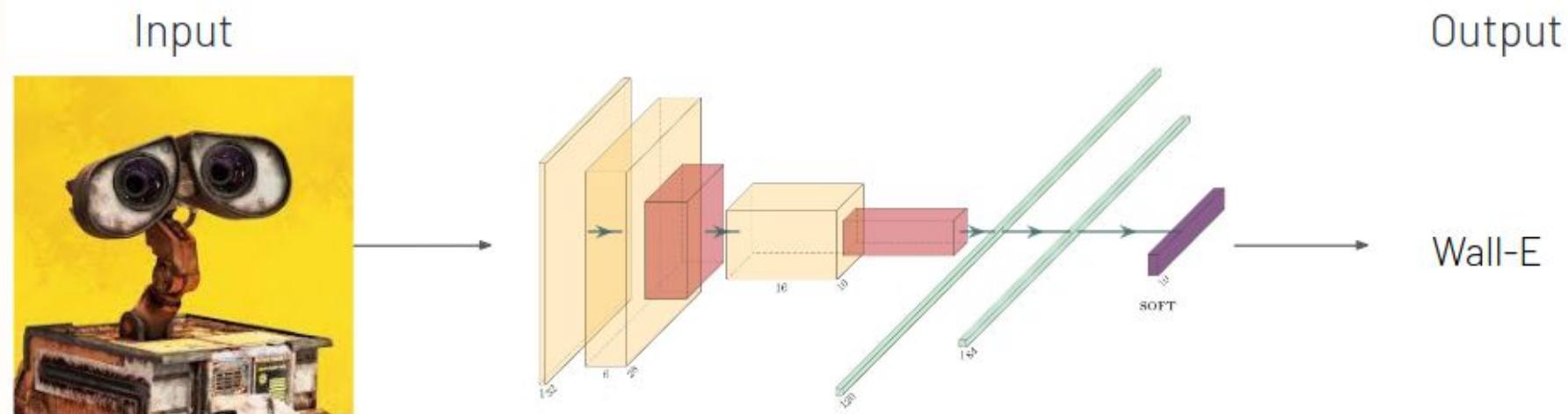
Laplaciano



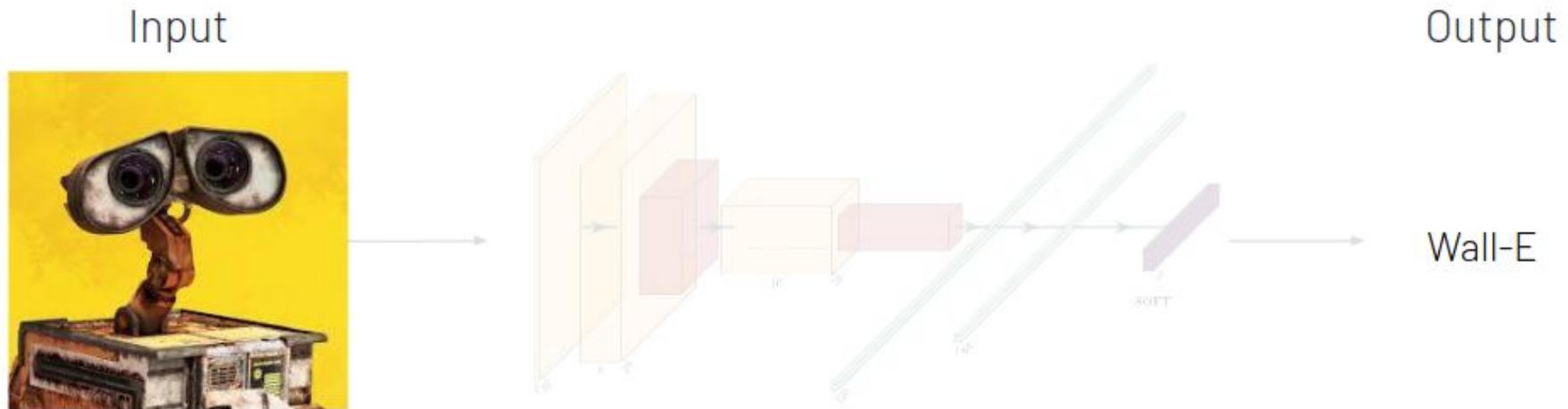
Sobel



Redes Convolucionais (CNNs)



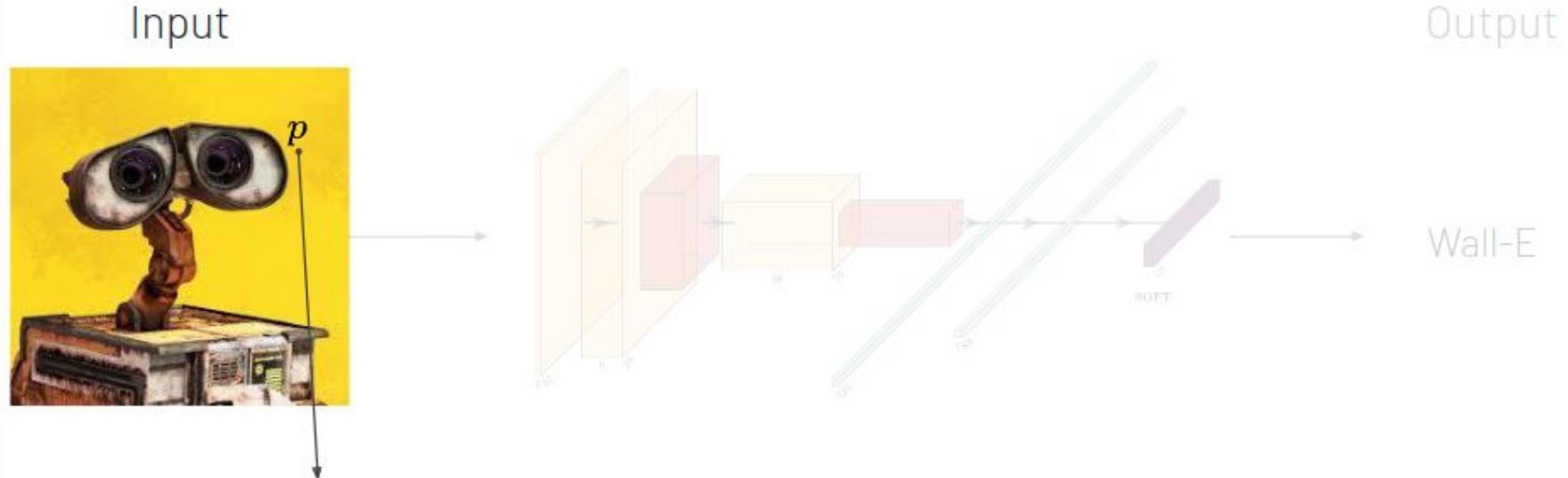
Redes Convolucionais (CNNs) - Representação da entrada



1. Color Model (RGB, HSV, Intensity)
2. Fourier Magnitude and Phase
3. Concatenation of multiples sources
(same or different domains)

1. Constrained (One-hot)
2. Unconstrained (Real value)

Redes Convolucionais (CNNs) - Representação da entrada



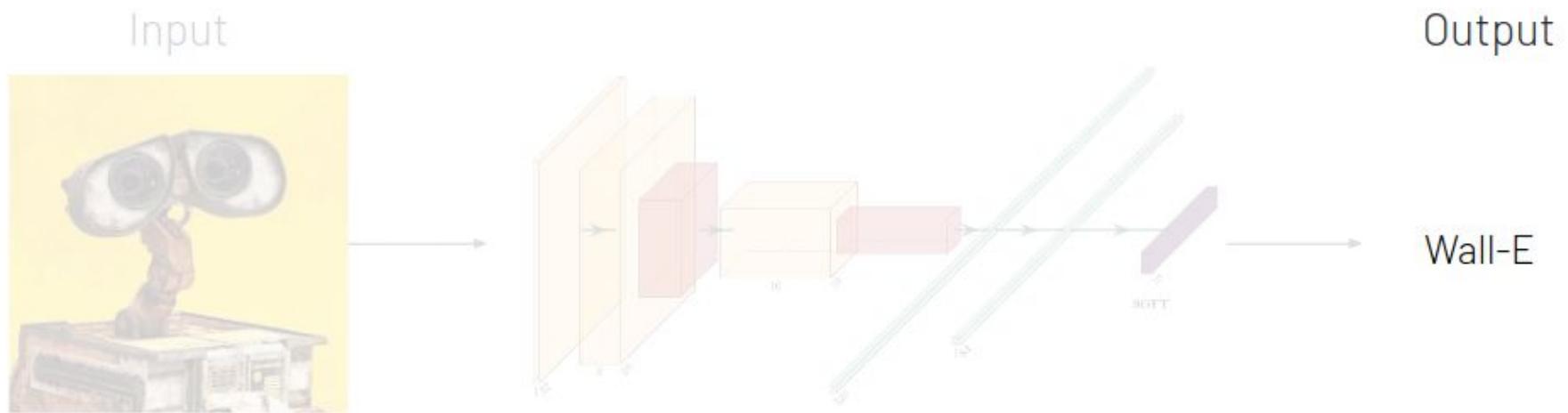
$$\text{RGB } [0, 255] \quad I(p) = (250, 214, 38)$$

$$\text{RGB } [0, 1] \quad I(p) = (0.98, 0.84, 0.15)$$

$$\text{Standardization} \quad I(p) = (2.16, 1.71, -1.14)$$

$$\frac{I(p) - \bar{I}}{\sigma} \quad \bar{I} = (0.485, 0.456, 0.406)$$
$$\sigma = (0.229, 0.224, 0.225)$$

Redes Convolucionais (CNNs) - Representação da saída

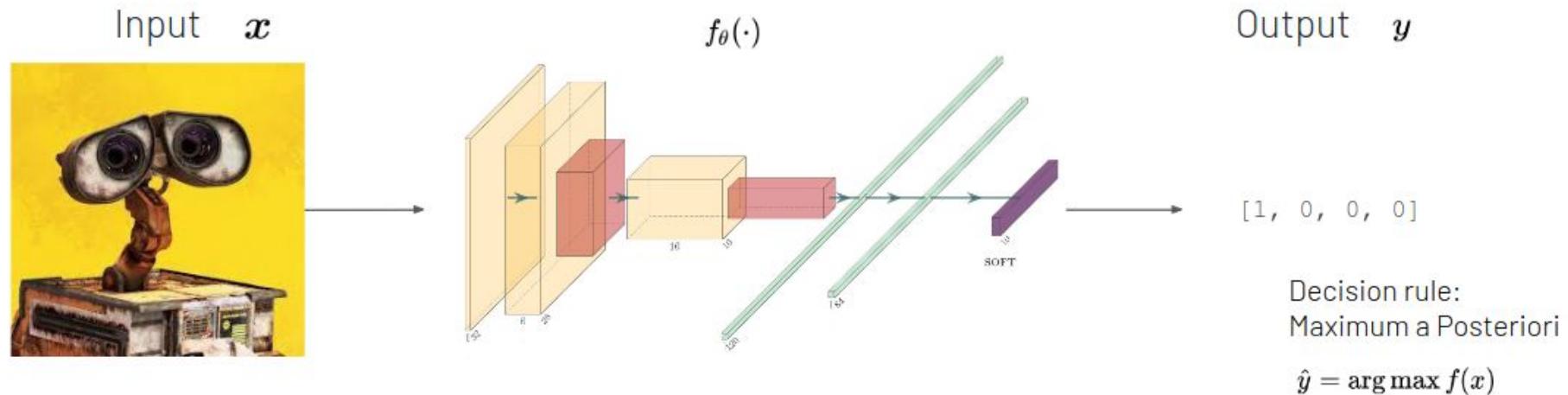


```
out_set={'Wall-E', 'ET', 'RD-D2', 'Bumblebee'}  
num_set={    0 ,    1 ,    2 ,    3 }
```

One-hot representation:

Wall-E	=	[1, 0, 0, 0]
ET	=	[0, 1, 0, 0]
RD-D2	=	[0, 0, 1, 0]
Bumblebee	=	[0, 0, 0, 1]

Redes Convolucionais (CNNs) - Representação da saída



```
out_set={'Wall-E', 'ET', 'RD-D2', 'Bumblebee'}  
num_set={    0 ,   1 ,   2 ,   3 }
```

One-hot representation:

Wall-E	=	[1, 0, 0, 0]
ET	=	[0, 1, 0, 0]
RD-D2	=	[0, 0, 1, 0]
Bumblebee	=	[0, 0, 0, 1]

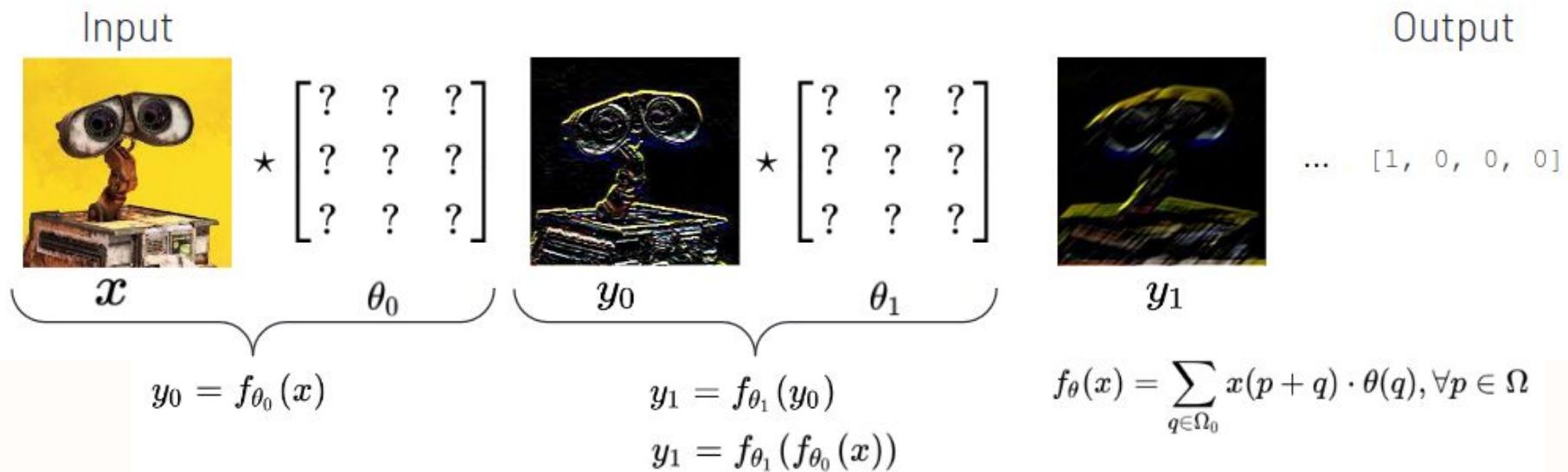
Redes Convolucionais - Funções

Function $f: \mathbb{I} \rightarrow \mathbb{O}$

Composition $(f \circ g)(x) = f(g(x))$

Parameterized function $f_\theta(x) = f(x; \theta)$

Family of functions $\{f_\theta\}_{\theta \in \Theta}$



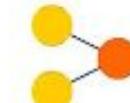
Arquiteturas de redes neurais

- Input Cell
- Backfed Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Gated Memory Cell
- Kernel
- Convolution or Pool

Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

Perceptron (P)



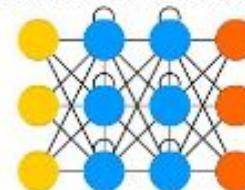
Feed Forward (FF)



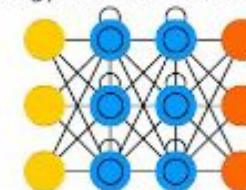
Radial Basis Network (RBF)



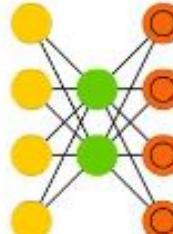
Recurrent Neural Network (RNN)



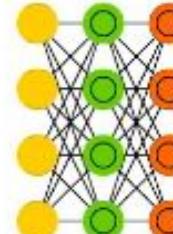
Long / Short Term Memory (LSTM)



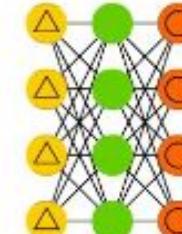
Auto Encoder (AE)



Variational AE (VAE)

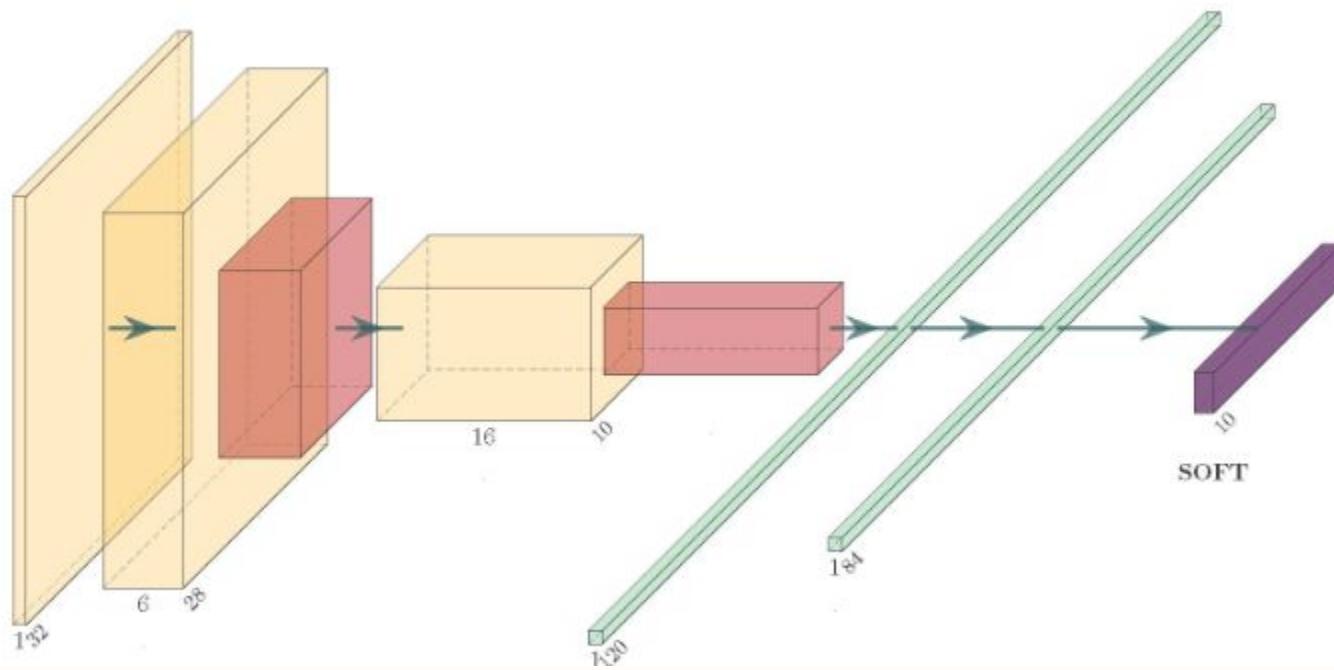


Denoising AE (DAE)



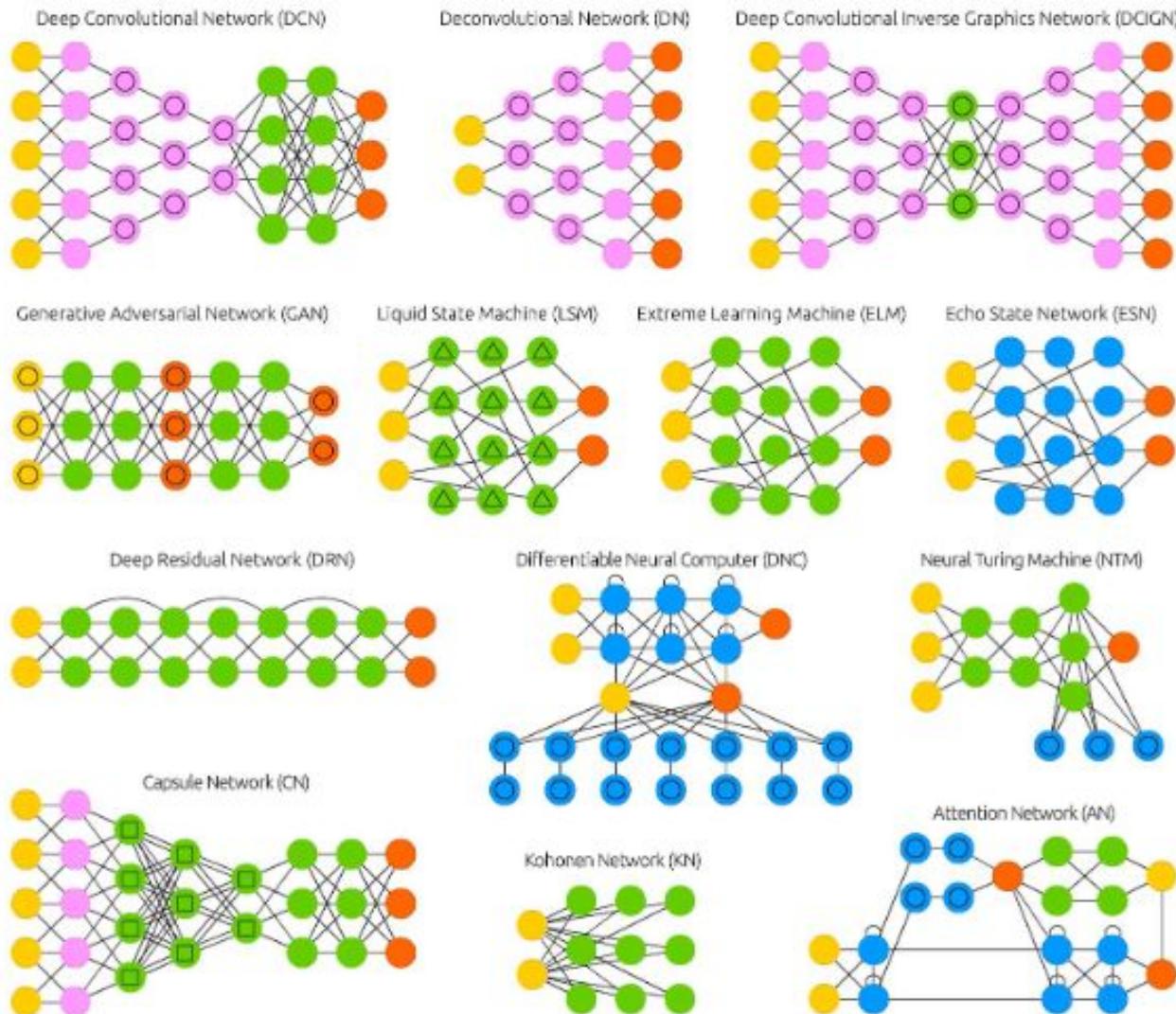
<https://www.asimovinstitute.org/neural-network-zoo/>

Redes Convolucionais - Arquitetura



- █ Convolution+ReLU █ Fully Connected
- █ Max-Pooling █ SoftMax

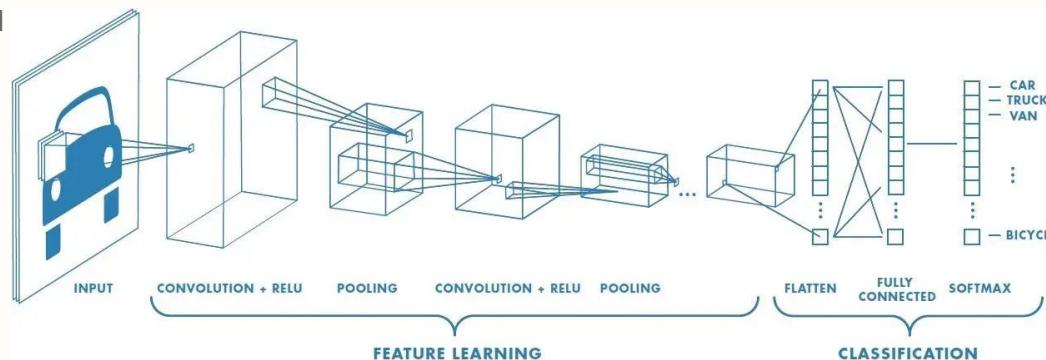
Redes Convolucionais - Arquitetura



<https://www.asimovinstitute.org/neural-network-zoo/>

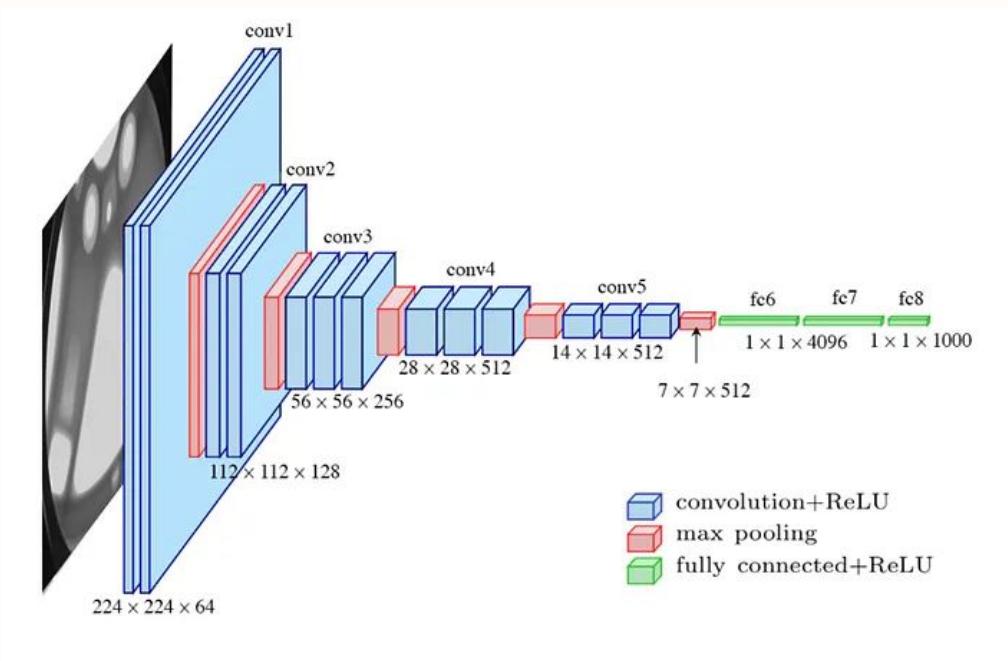
CNNs - Camadas convolucionais

- As CNNs podem ser definidas através dos blocos convolucionais e/ou lineares
 - ◆ De forma intuitiva, podemos visualizar as camadas convolucionais com filtros que são aprendidos pela rede, capturando características relevantes que são extraídas do conjunto de treinamento
 - ◆ As camadas totalmente conectadas interpretam as features, funcionando como o classificador da arquitetura
- A ideia geral é agregar as informações espaciais, diminuindo a dimensionalidade do volume de saída e aumentando o espaço das features extraídas em cada região
 - ◆ Reduzimos a dimensão espacial e aumentamos a profundidade do volume, gerando descritores locais que capturam padrões em cada região do volume de entrada



CNNs - Blocos de Convolução

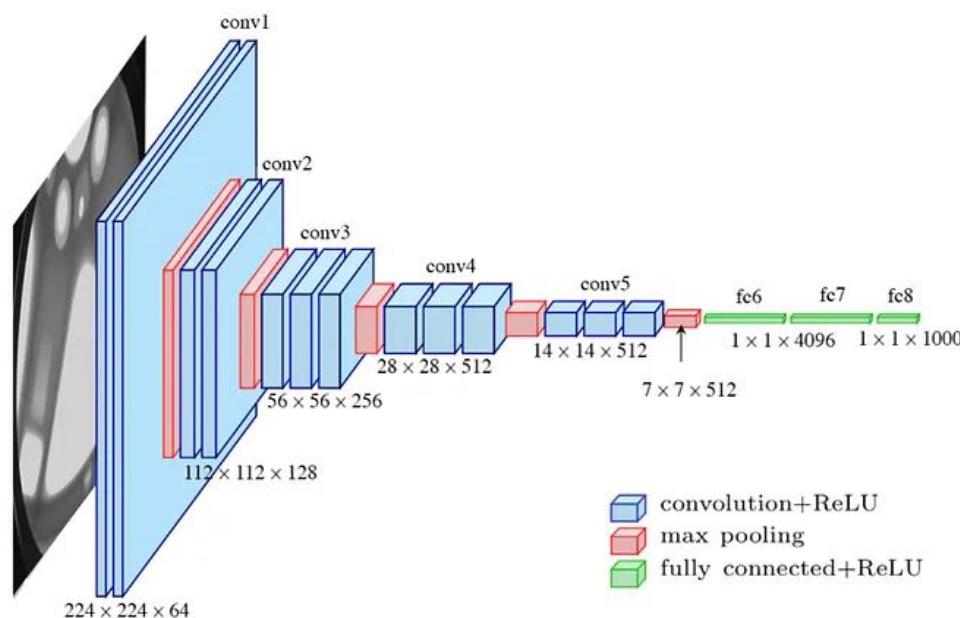
- As CNNs atuam como extratores de características
 - ◆ As camadas iniciais extraem características básicas de baixo nível, enquanto que camadas mais profundas extraem características de alto nível
- Os blocos básicos geralmente são compostos por um conjunto de convoluções de mesmo tamanho, seguidos de uma ativação
- Ao final de cada bloco, uma operação de pooling é realizada, diminuindo a dimensionalidade espacial para o próximo bloco



Componentes Básicos

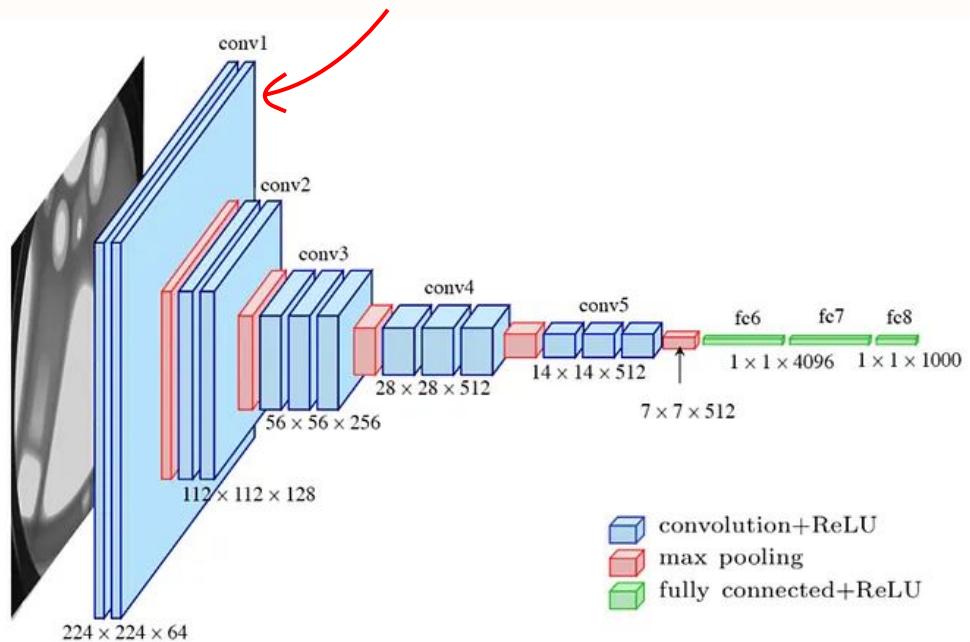
As redes convolucionais tradicionais possuem em sua arquitetura os seguintes componentes básicos:

- **Camada Convolucional:** realiza a extração das features de um volume de entrada (e.g., imagem) a partir dos pesos gerados em cada kernel da camada, gerando um mapa de ativação como saída.
- **Pooling:** reduz a dimensão espacial da entrada, retornando uma subamostra contendo uma representação simplificada
- **Camadas totalmente conectadas (Fully connected Layers):** agrupa as features extraídas pelas camadas anteriores e gera o vetor de probabilidades (predições)



Camada Convolutional (Conv Layer)

- A camada convolucional é o bloco básico que compõe as CNNs. Essa camada consiste de um conjunto de parâmetros que representam filtros (kernels).
 - ◆ Possuem um pequeno campo receptivo, mas atuam através de toda a profundidade do volume de entrada (canais)



Camada Convolutional (Conv Layer)

- Durante o forward, a Conv Layer computa o produto entre o volume de entrada e o kernel, gerando um mapa de ativação 2D
 - ◆ Uma camada convolucional geralmente é composta de N filtros, gerando N mapas de ativações durante o forward
- Os filtros atuam detectando características através das posições espaciais do volume de entrada

Camadas Convolucional - Kernel



Entrada

0.09	-0.01	0.08
-0.42	0.16	-0.02
-0.32	0.37	-0.08

Kernel /
Filtro



Saída

Camada Convolutional (Conv Layer)

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

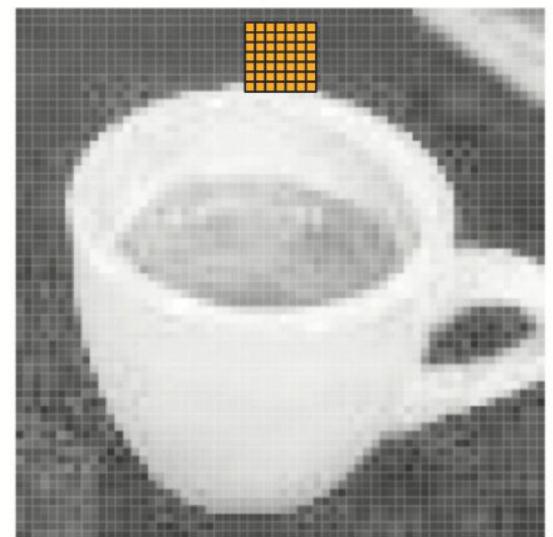
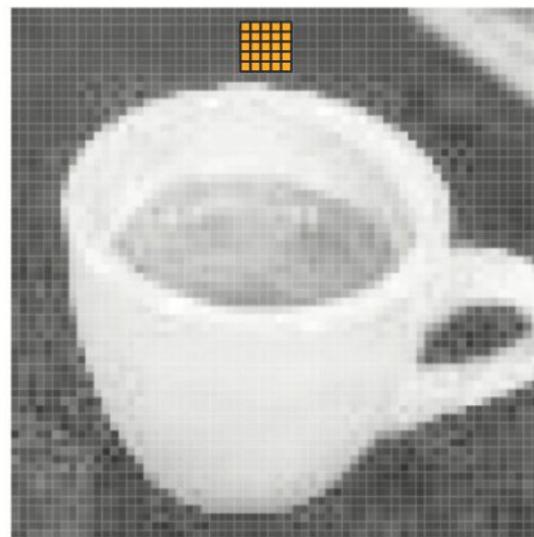
4		

Convolved Feature

Elementos das Camadas Convolucionais

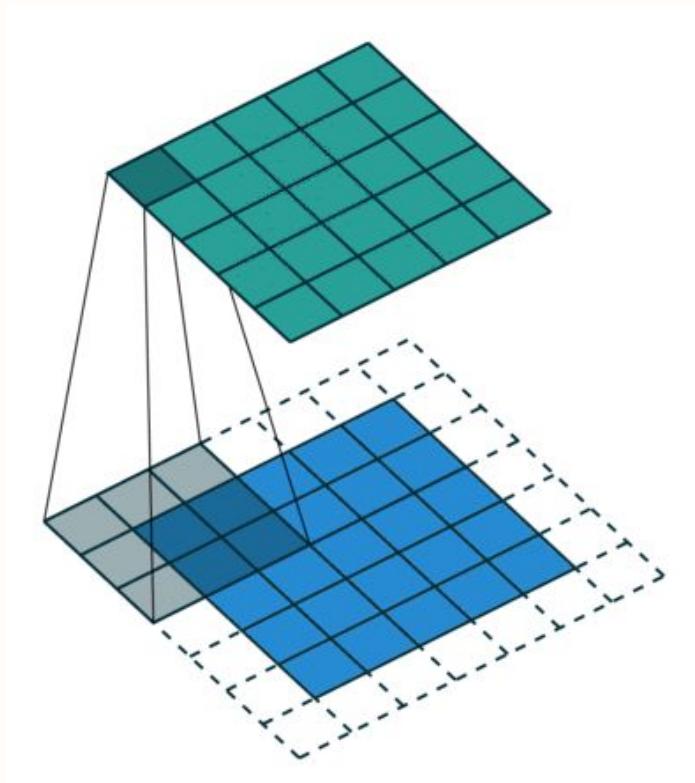
- Os principais elementos que definem uma camada convolucional são:
- ◆ Tamanho do Filtro/Kernel
 - ◆ Padding
 - ◆ Stride
 - ◆ Dilatation

Kernel

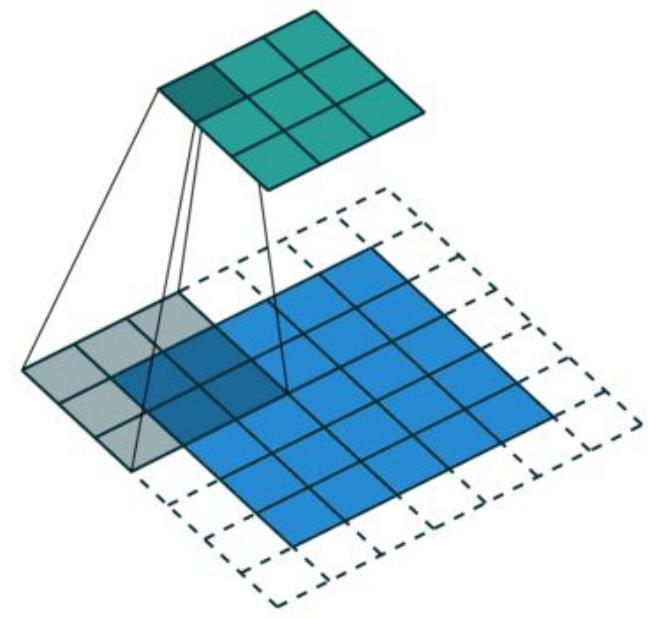


Stride

→ Trata-se do passo que damos no volume para aplicar a convolução



stride=1



stride=2

Padding

→ Expansão das bordas da imagem

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

sem padding

0 ₂	0 ₀	0 ₁	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0
0 ₀	0 ₁	1 ₁	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

padding 0

Padding

- Geralmente, nos frameworks de aprendizagem o padding pode ser denominado como ‘same’ e ‘valid’
 - ◆ **Valid:** A dimensão espacial do volume é reduzida, aplicando a convolução normalmente
 - ◆ **Same:** O padding é aplicado de tal forma que a dimensão espacial do volume de saída permanece igual a dimensão espacial do volume de entrada

Camadas Convolucionais

- Dado um volume de entrada de tamanho W_{in} , e considerando uma camada convolucional com tamanho do kernel K , pad P e stride S . O tamanho espacial do volume de saída é dado por:

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

Camadas Convolucionais

- Dado um volume de entrada de tamanho W_{in} , e considerando uma camada convolucional com tamanho do kernel K , pad P e stride S . O tamanho espacial do volume de saída é dado por:

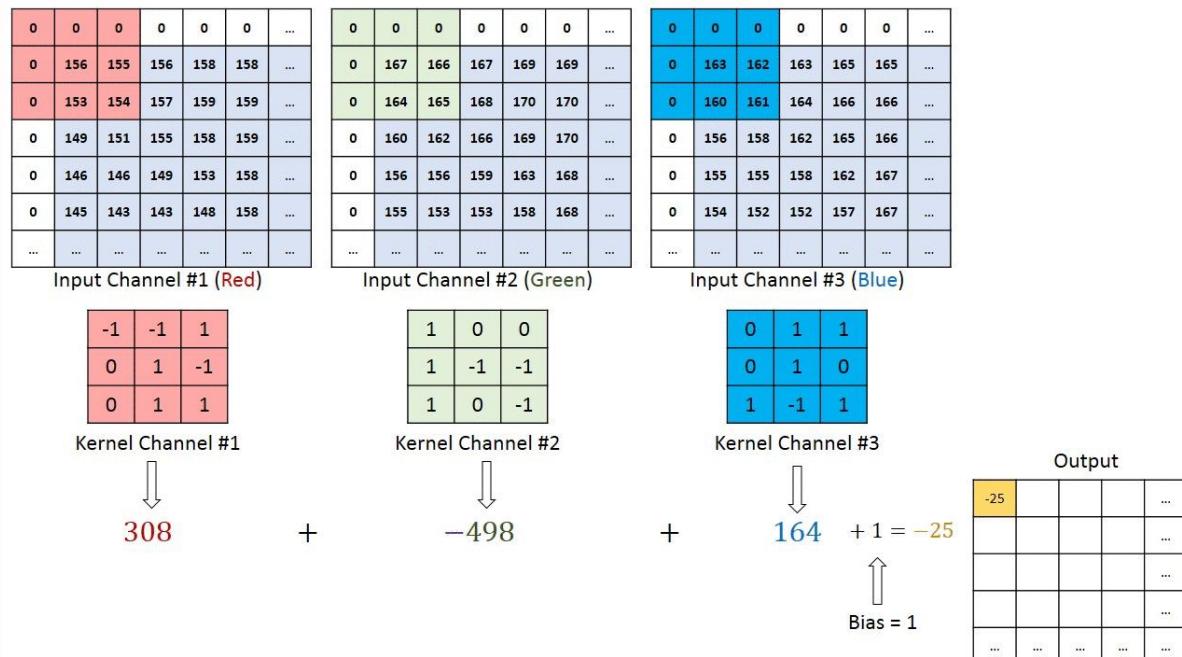
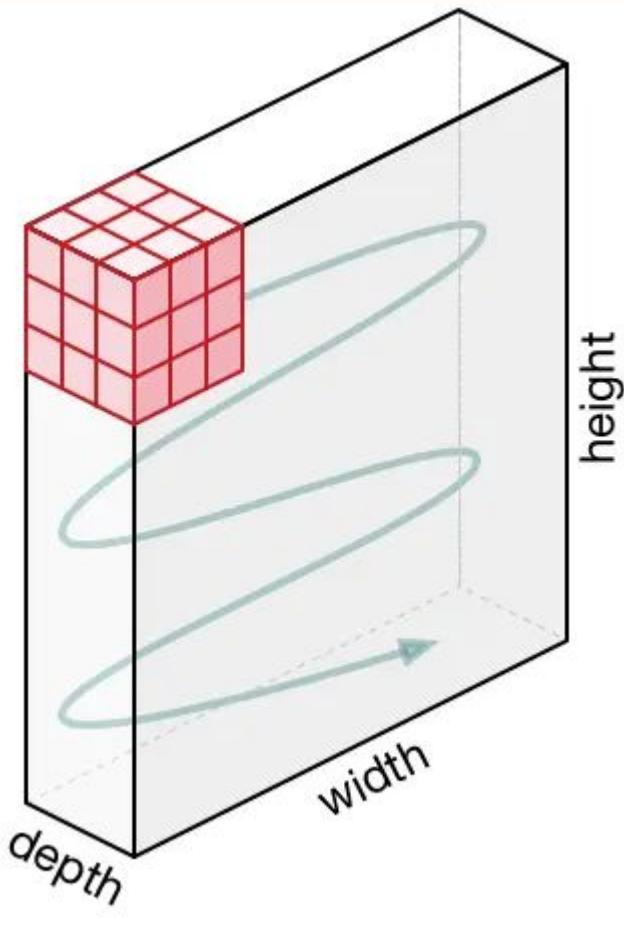
$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1$$

- Se o valor de W_{out} não for um número inteiro, então o valor de stride precisa ser ajustado (neurônios não podem atuar no volume de entrada de forma simétrica)
 - ◆ Geralmente, utilizando zero padding com $P = (K-1)/2$ quando o stride é 1, garante que o volume de entrada e o volume de saída possuirão o mesmo tamanho espacial

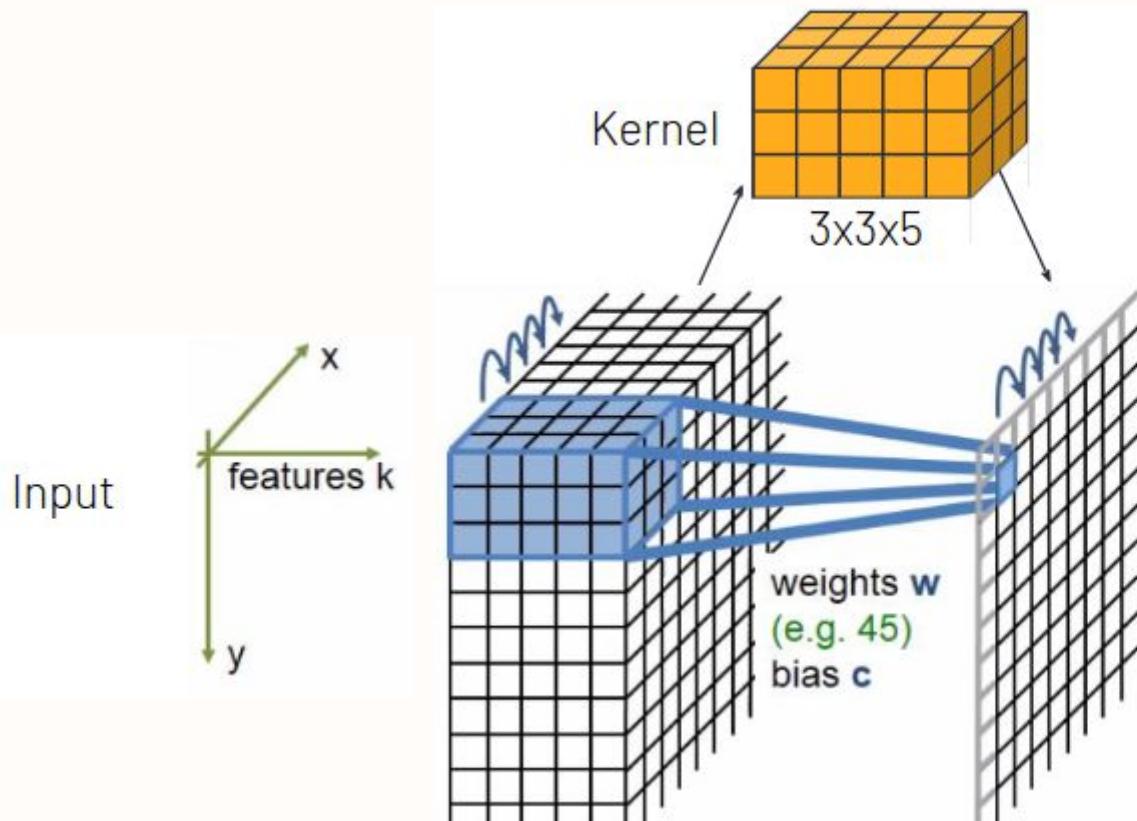
Camadas Convolucionais

- Como mencionado anteriormente, a camada convolucional atua sobre toda a profundidade do volume
- Para uma entrada de dimensão $H_{in} \times W_{in} \times C$, e, considerando uma camada convolucional com N filtros, o volume de saída terá a dimensão $H_{out} \times W_{out} \times N$
 - ◆ Cada filtro da camada convolucional tem o mesmo tamanho de kernel K e, deve ter a mesma profundidade C correspondente ao volume de entrada ($K \times K \times C$)
 - ◆ O volume de saída terá N canais, correspondendo ao número de filtros da camada atual
 - ◆ O tamanho espacial H, W do volume de saída é dado pela equação do slide anterior

Camadas Convolucionais



Camada Convolucional (Conv Layer)



$$y = \sum_{i=1}^5 x(:,:,i) \star k(:,:,i)$$

Differentiable!

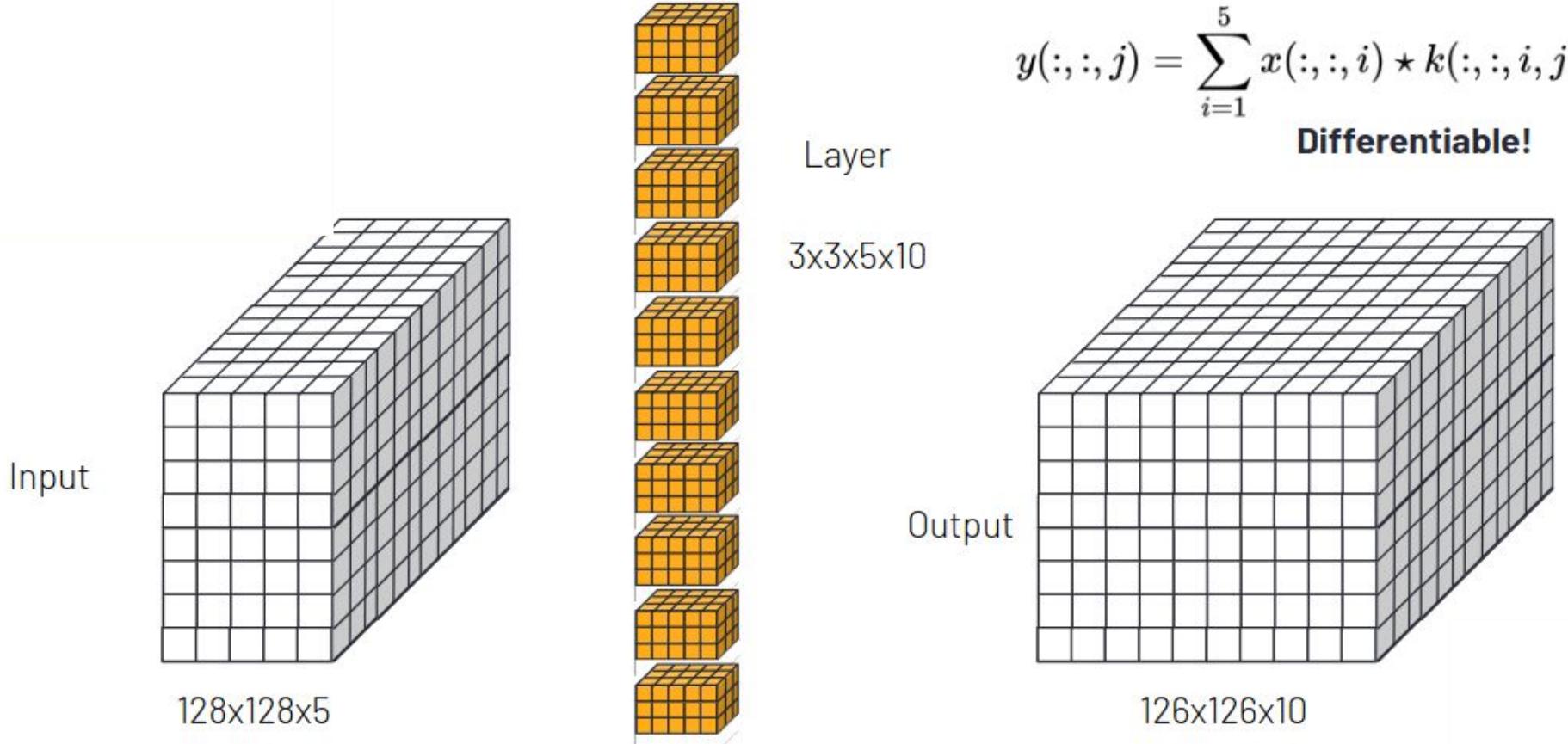
Output

Input: Tensor(w_i, h_i, c)

Parameters: Tensor(w_k, h_k, c, n)

Output: Tensor($w_i - (w_k - 1), h_i - (h_k - 1), n$)

Camada Convolucional (Conv Layer)



Camadas Convolucionais

- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com $\text{kernel}=3$, $\text{pad}=0$ e $\text{stride}=1$
- Qual será o tamanho espacial do volume de saída?

Camadas Convolucionais

- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com kernel=3, pad=0 e stride=1
- Qual será o tamanho espacial do volume de saída?

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1 = \frac{256 - 3 + 2.0}{1} + 1 = 254$$

$$H_{out} = \frac{H_{in} - K + 2P}{S} + 1 = \frac{256 - 3 + 2.0}{1} + 1 = 254$$

Camadas Convolucionais

- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com kernel=3, pad=0 e stride=1
- Qual será a profundidade do volume de saída?

Camadas Convolucionais

- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com kernel=3, pad=0 e stride=1
- Qual será a profundidade do volume de saída?
 - ◆ Temos 10 filtros 3x3, cada um tem profundidade 3 (3 canais, RGB)
 - ◆ Cada filtro atua espacialmente em toda a profundidade, gerando um mapa de ativação de 1 canal
 - ◆ O volume de saída terá o tamanho 254 x 254 x 10

Camadas Convolucionais

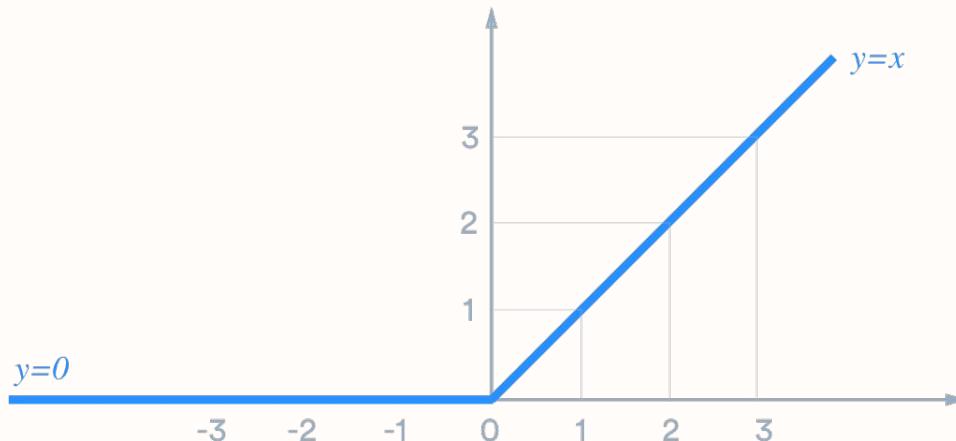
- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com $\text{kernel}=3$, $\text{pad}=0$ e $\text{stride}=1$
- Quantos parâmetros treináveis tem essa camada?

Camadas Convolucionais

- Temos uma imagem de tamanho 256x256 RGB, queremos passar essa imagem por uma camada convolucional que possui 10 filtros com kernel=3, pad=0 e stride=1
- Quantos parâmetros treináveis tem essa camada?
 - ◆ Temos 10 filtros de tamanho $3 \times 3 \times 3 = 10 \times 3 \times 3 \times 3 = 270$ pesos
 - ◆ Para cada filtro, temos um bias que é somado com a saída ($270 + 10$)
 - ◆ No total, temos 280 parâmetros treináveis

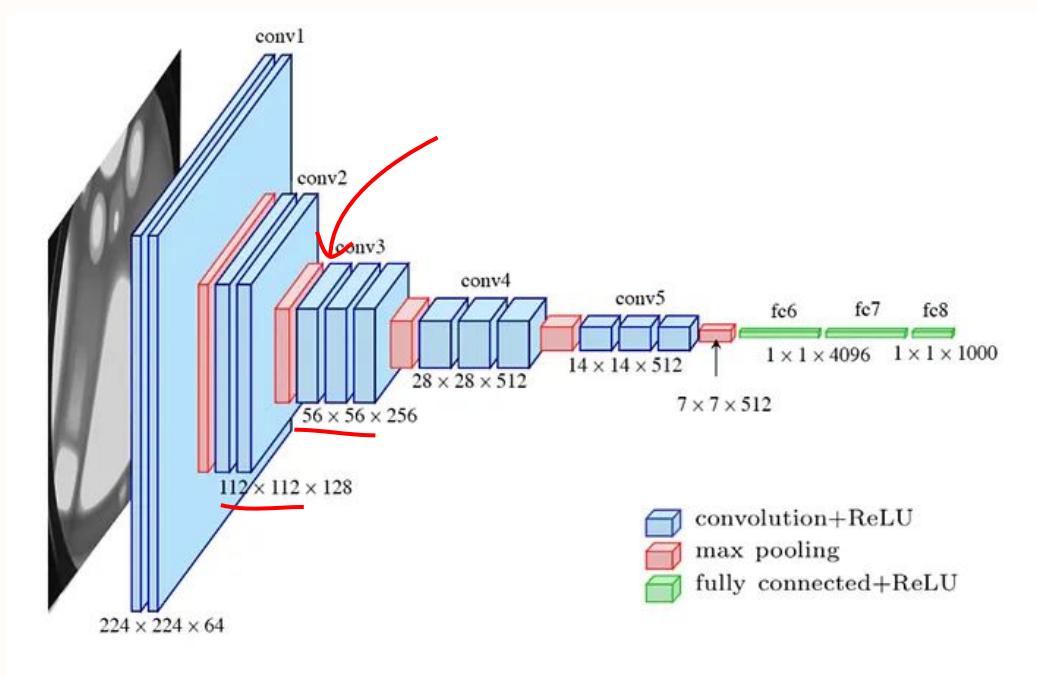
Camadas Convolucionais

- Assim como nas MLPs, as camadas convolucionais também possuem funções de ativação que inserem não-linearidade ao volume de saída logo após a convolução, gerando o mapa de ativação da camada
 - ◆ Como as camadas convolucionais são utilizadas com imagens, geralmente as funções do tipo ReLU são utilizadas (i.e., desconsideram valores negativos e possuem ativação a partir de um threshold)



Pooling

- As camadas de **pooling** aplicam uma operação de corte, para reduzir a dimensão espacial do volume



Pooling

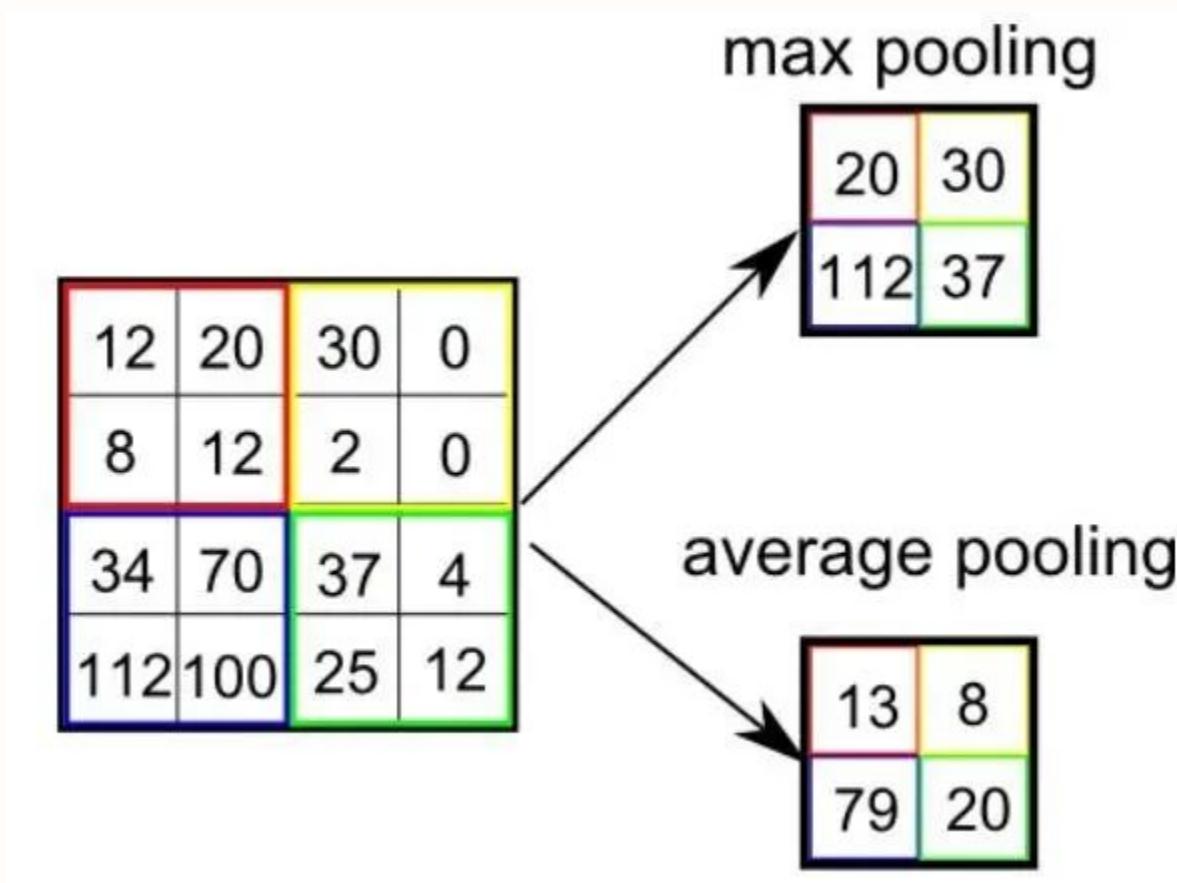
- As camadas de **pooling** aplicam uma operação de corte, para reduzir a dimensão espacial do volume
 - ◆ Agrega regiões do volume, mantendo features dominantes enquanto reduz o custo de processamento da rede (i.e., reduzindo o número de valores que serão processados na próxima camada)
 - ◆ A operação se baseia apenas nos valores do volume, e não requer nenhum parâmetro

0	1	2
3	4	5
6	7	8

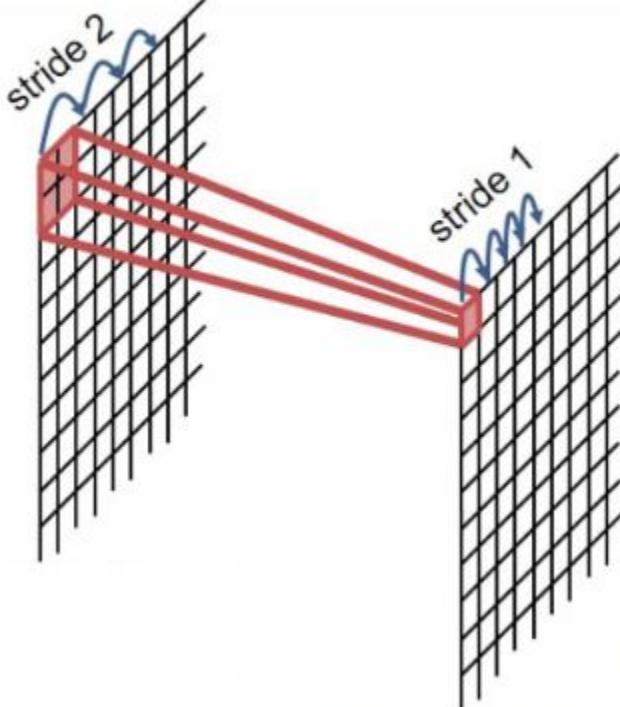
2 x 2 Max
Pooling

4	5
7	8

Pooling



Pooling



$$y(p, q, r) = \max_{i,j \in \{0,1\}} x(2p + i, 2q + j, r)$$

Differentiable!

resulting feature
map has factor 2
lower spatial
resolution

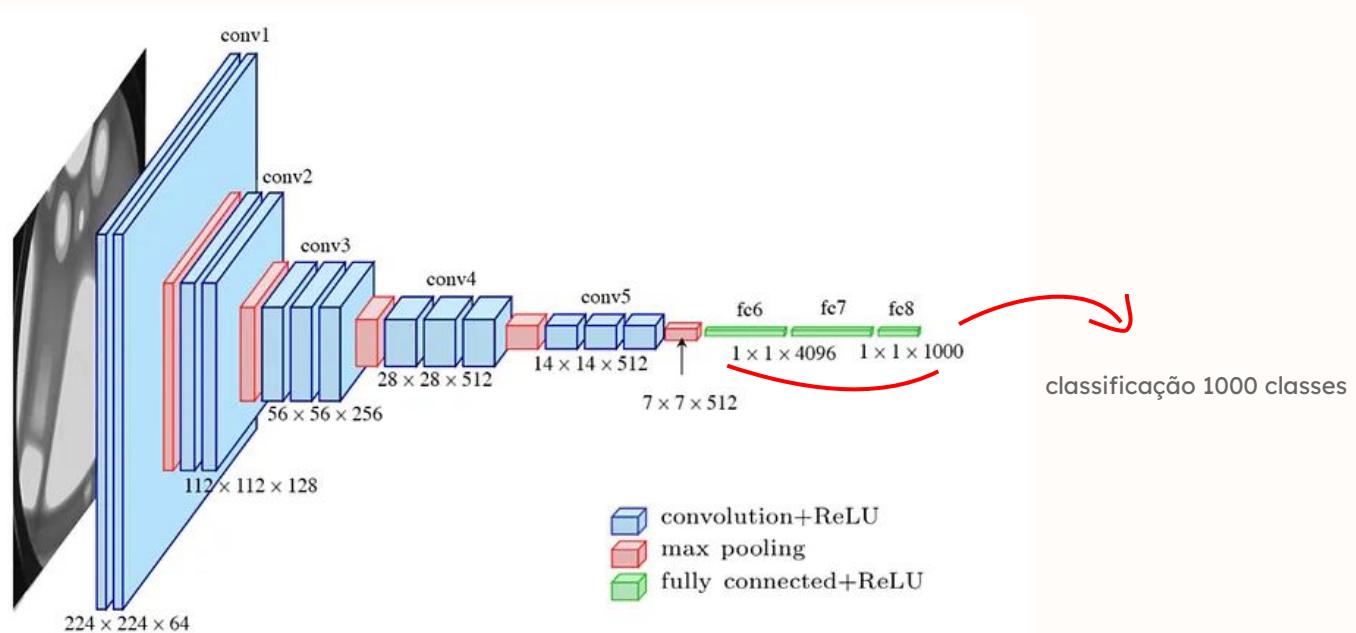
Input: Tensor(w_i, h_i, c)

Parameters: -

Output: Tensor($w_i/w_k, h_i/h_k, c$)

Fully Connected Layers (FC Layer)

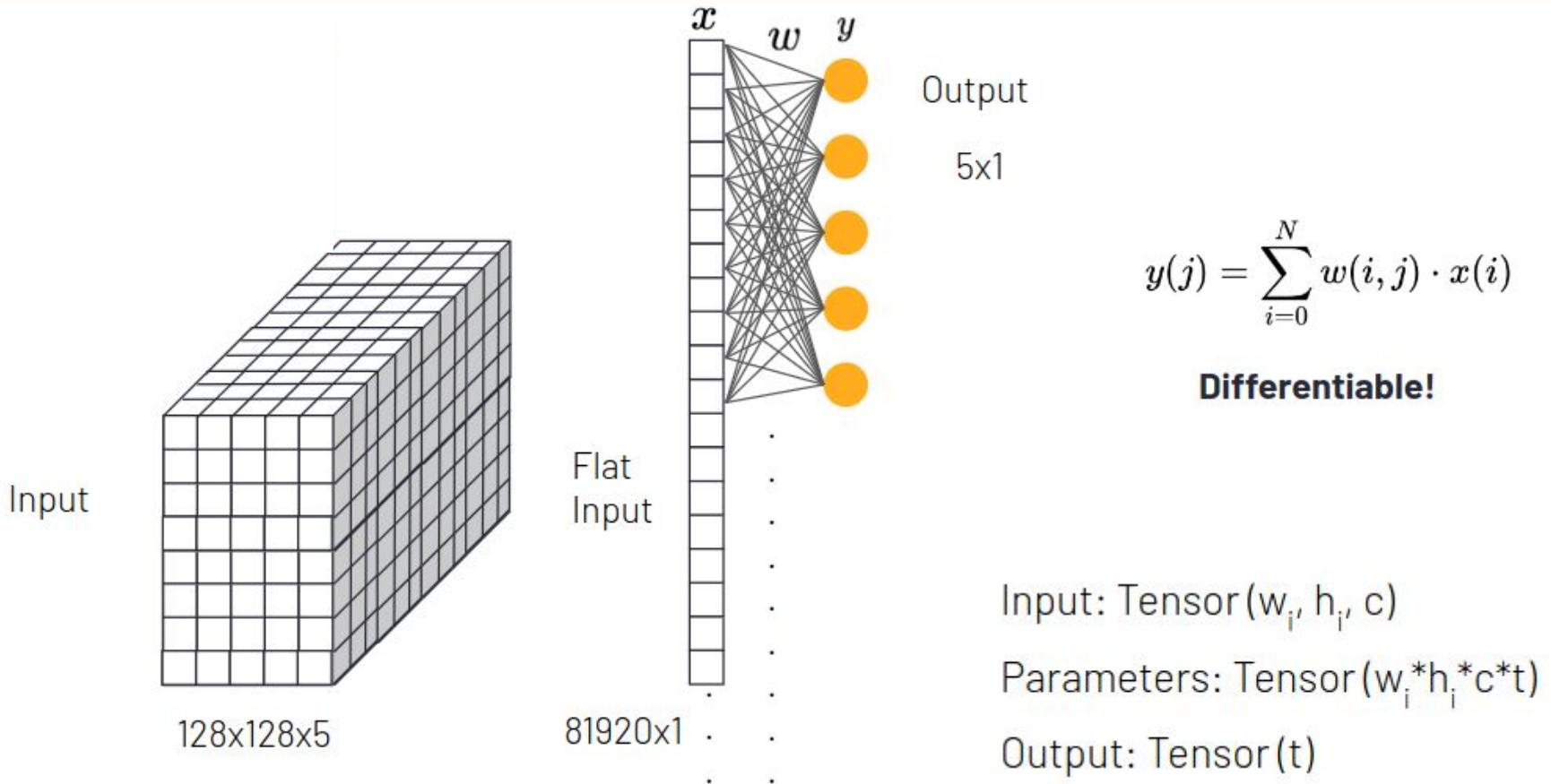
- Fully Connected Layers ou camadas totalmente conectadas são camadas muito utilizadas em algumas arquiteturas para transformar/interpretar as ativações processadas pelas camadas convolucionais, gerando o vetor de saída (predição)



Fully Connected Layers (FC Layer)

- As camadas FC são equivalentes as camadas utilizadas nas MLPs
 - ◆ Geralmente, estas camadas são utilizadas no **final da rede neural**
 - ◆ As FC Layers processam as features capturadas pelas Conv Layers e as **convertem para um vetor de probabilidades** que corresponde a resposta da rede
 - ◆ É uma boa forma de adicionar não-linearidade para combinar as features de alto nível extraídas pelas camadas convolucionais para realizar a classificação
 - ◆ Requer uma entrada de **tamanho fixo**

Fully Connected Layers (FC Layer)

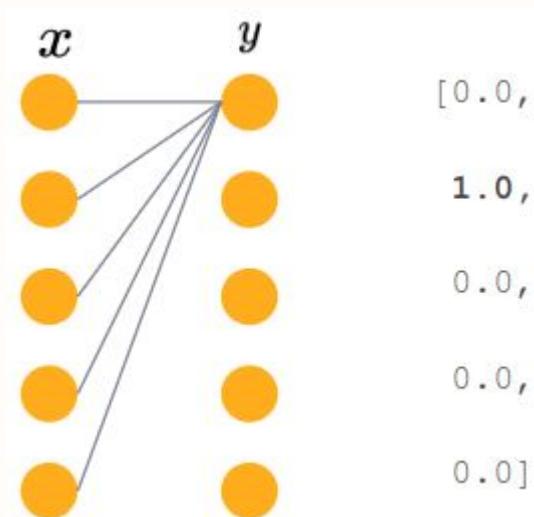


Softmax

- Utilizado para transformar as ativações de saída em um vetor de probabilidades

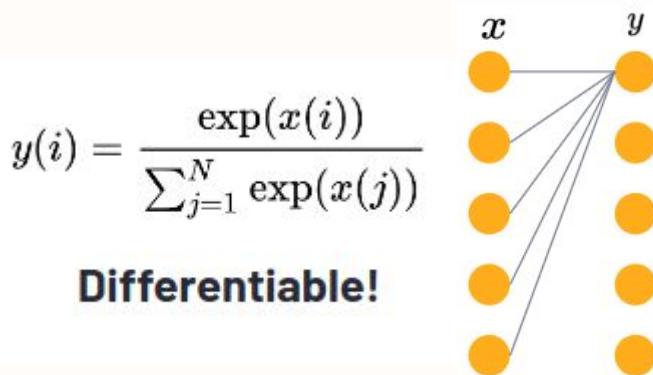
$$y(i) = \frac{\exp(x(i))}{\sum_{j=1}^N \exp(x(j))}$$

Differentiable!



Softmax

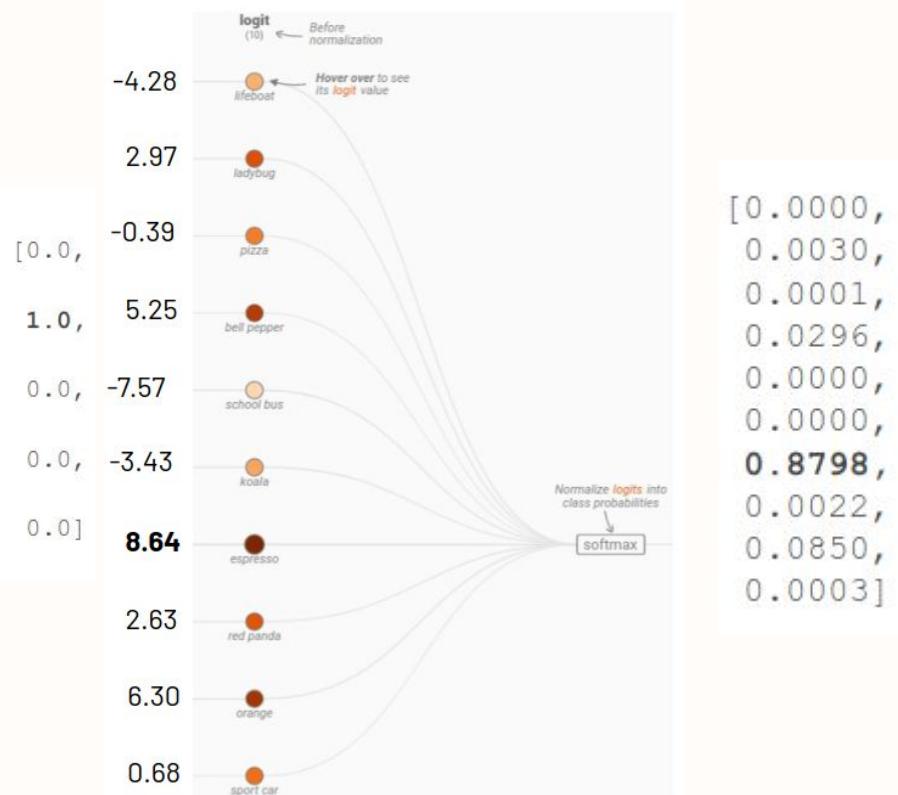
- Utilizado para transformar as ativações de saída em um vetor de probabilidades



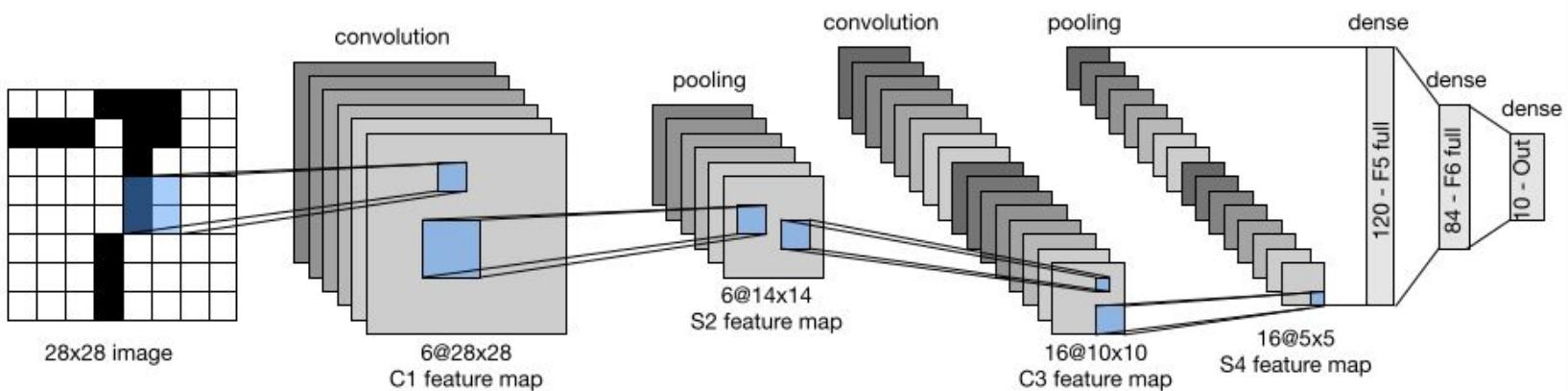
Input: Tensor(w_i, h_i, c)

Parameters: -

Output: Tensor(w_i, h_i, c)



LeNet



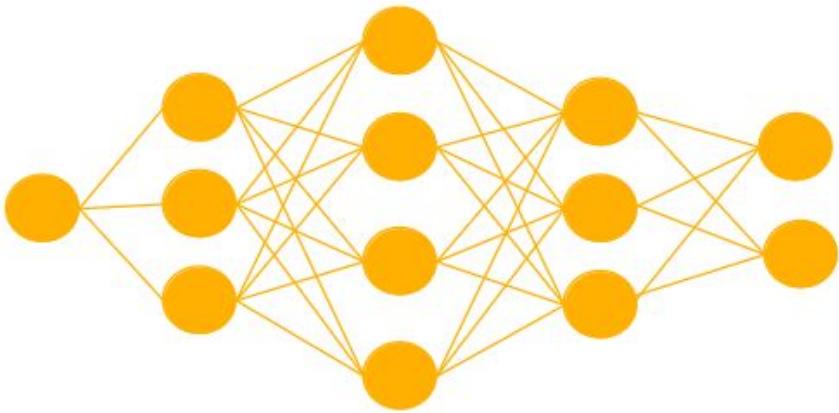
Layer	Input	Params	Output
Conv1+ReLU	$32 \times 32 \times 1$	$5 \times 5 \times 1 \times 6$	$28 \times 28 \times 6$
MaxPool2x2_1	$28 \times 28 \times 6$	-	$14 \times 14 \times 6$
Conv2+ReLU	$14 \times 14 \times 6$	$5 \times 5 \times 6 \times 16$	$10 \times 10 \times 16$
MaxPool2x2_2	$10 \times 10 \times 16$	-	$5 \times 5 \times 16$
Linear1	$5 \times 5 \times 16$	400×120	120×1
Linear2	120×1	120×84	84×1
Linear3+Soft	84×1	84×10	10×1

Parameters: 61,470
 Linear1: 48,000 (78%)

CNNs x MLPs

	CNNs	MLP
Parâmetros	<p>Os parâmetros da camada convolucional não depende da dimensão espacial da entrada e permanece constante uma vez definido o tamanho do kernel e número de canais da entrada;</p> <p>O tamanho espacial do volume de saída é alterado e o número de canais é definido pelo número de filtros utilizado</p>	<p>O número de parâmetros da camada depende do tamanho espacial da entrada (n), e este valor cresce por um fator n para cada neurônio na camada</p>
Dimensionalidade	<p>As camadas convolucionais atuam localmente em cada região da entrada, aproveitando a informação espacial e agregando a informação através dos canais do volume</p>	<p>A MLP recebe uma entrada em forma de vetor ($1 \times n$), perdemos as características espaciais da imagem</p>

Medindo Performance



$$f_{\theta_0} \in \{f_{\theta}\}_{\theta \in \Theta}$$

$$f_{\theta_1} \in \{f_{\theta}\}_{\theta \in \Theta}$$

Performance measurement $\mathcal{P}: \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{R}$

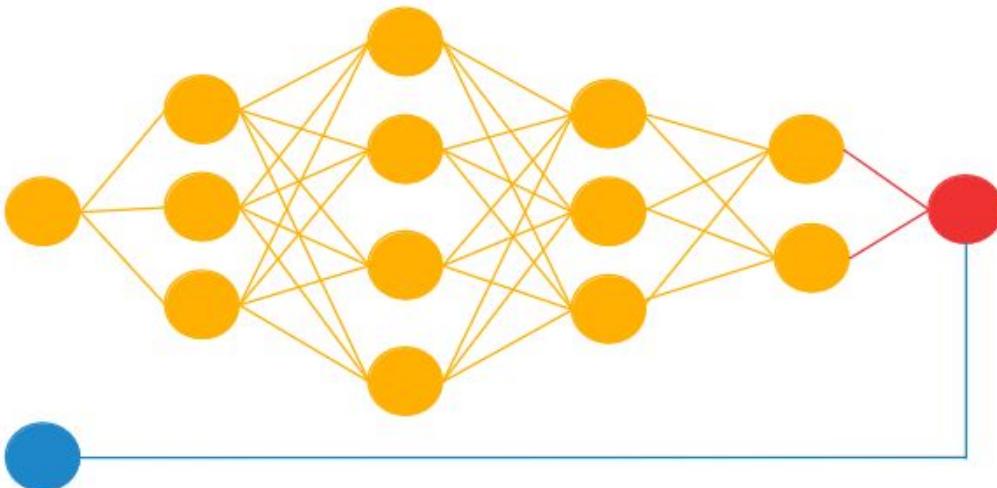
In this particular case $\mathcal{P}(\hat{y}, y) = \mathcal{P}(f_{\theta}(x), y)$

Is **model 0** better than **model 1**?

$$\mathcal{P}(f_{\theta_0}(x), y) < \mathcal{P}(f_{\theta_1}(x), y)$$

$$\mathbb{E} [\mathcal{P}(f_{\theta_0}(X), Y)] < \mathbb{E} [\mathcal{P}(f_{\theta_1}(X), Y)]$$

Medindo Performance



Performance measurement $\mathcal{P}: \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{R}$

Measurement

Accuracy → Cross-Entropy

F-Score → Soft Dice

Tversky → Soft Tversky

Jaccard → Lovasz-Softmax

...

Surrogate (Loss func)

Performance measurement $\mathcal{P}: \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{R}$

Non-differentiable

Accuracy

F-Score

Tversky

Jaccard

Precision

Recall

Specificity

AUC-ROC

...

Differentiable

Cross-Entropy

Soft Dice

Soft Tversky

Lovasz-Softmax

Mean Squared Error

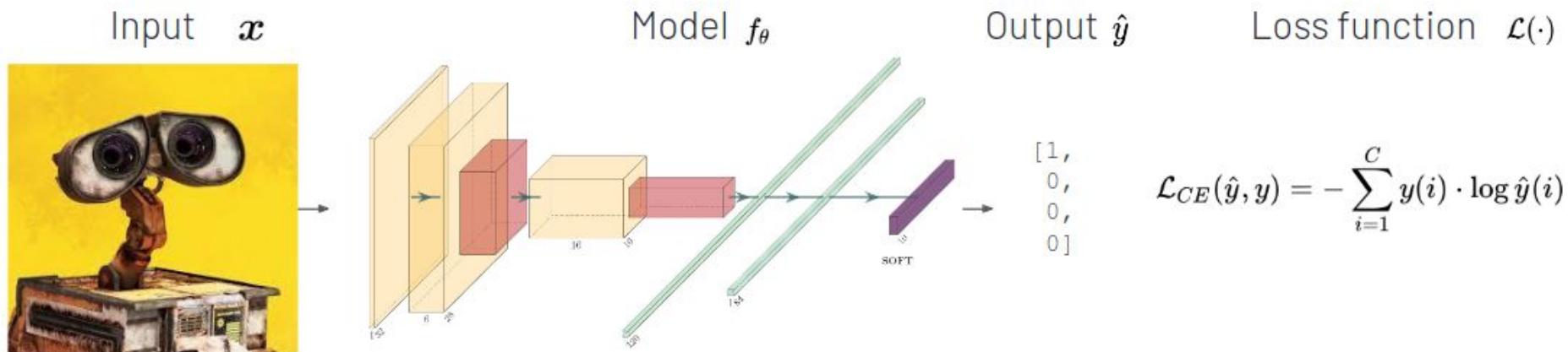
Mean Absolute Error

Huber

Perceptual loss

...

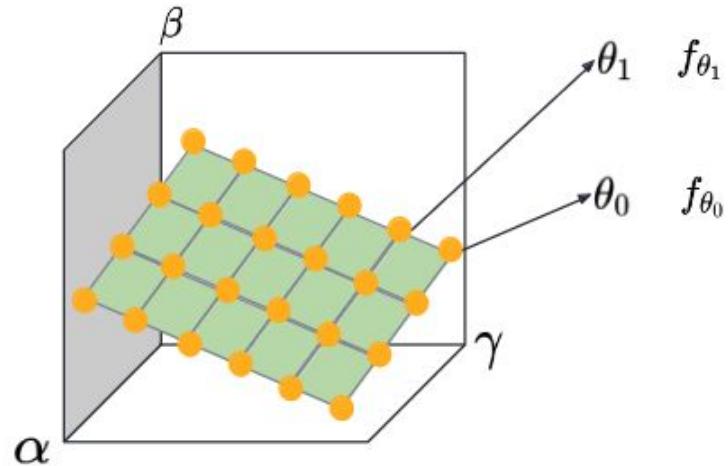
Medindo Performance



Ajuste dos pesos

Architecture $\{f_\theta\}_{\theta \in \Theta}$

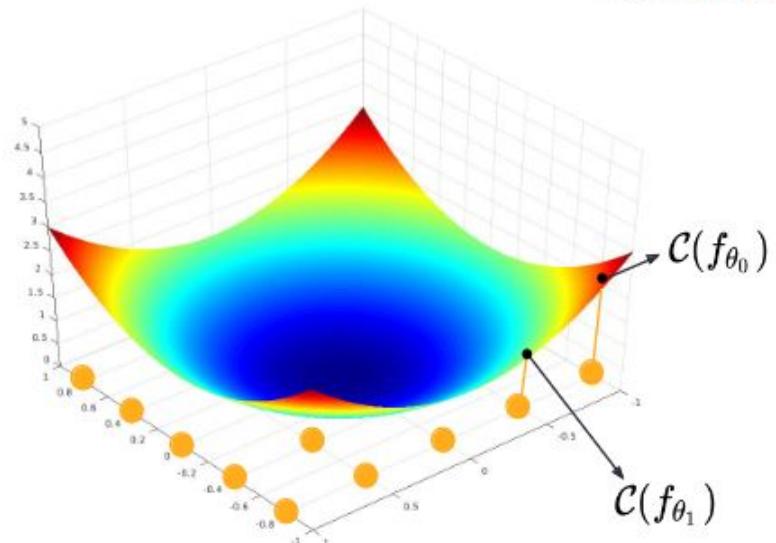
$$\theta = (\alpha, \beta, \gamma)$$



Data set S

$$\text{Cost of a model} \quad c(\theta) = \frac{1}{|S|} \sum_{(x,y) \in S} \mathcal{L}(y, f_\theta(x))$$

Differentiable!



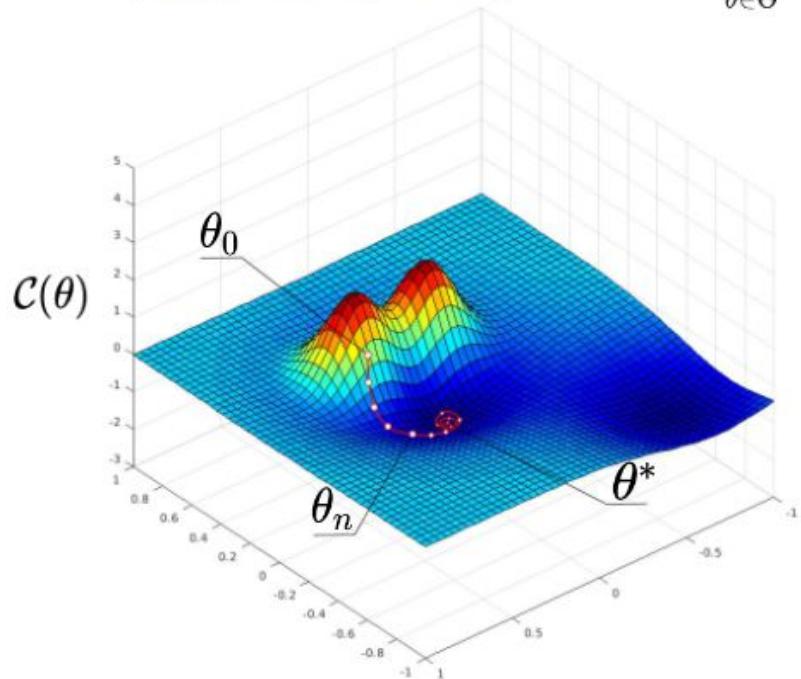
→ Backpropagation

Ajuste dos pesos

Optimization problem $\theta^* = \arg \min_{\theta \in \Theta} \mathcal{C}(\theta)$

Cost of a model $\mathcal{C}(\theta) = \frac{1}{|S|} \sum_{(x,y) \in S} \mathcal{L}(y, f_\theta(x))$

Differentiable!



Gradient Descent $\theta_{n+1} = \theta_n - \eta \cdot \nabla \mathcal{C}(\theta_n)$

CNN Backpropagation

CNN Backpropagation

- Realizamos a operação de convolução para fazer o forward numa camada convolucional
- Mas como podemos fazer o backpropagation?

CNN Backpropagation

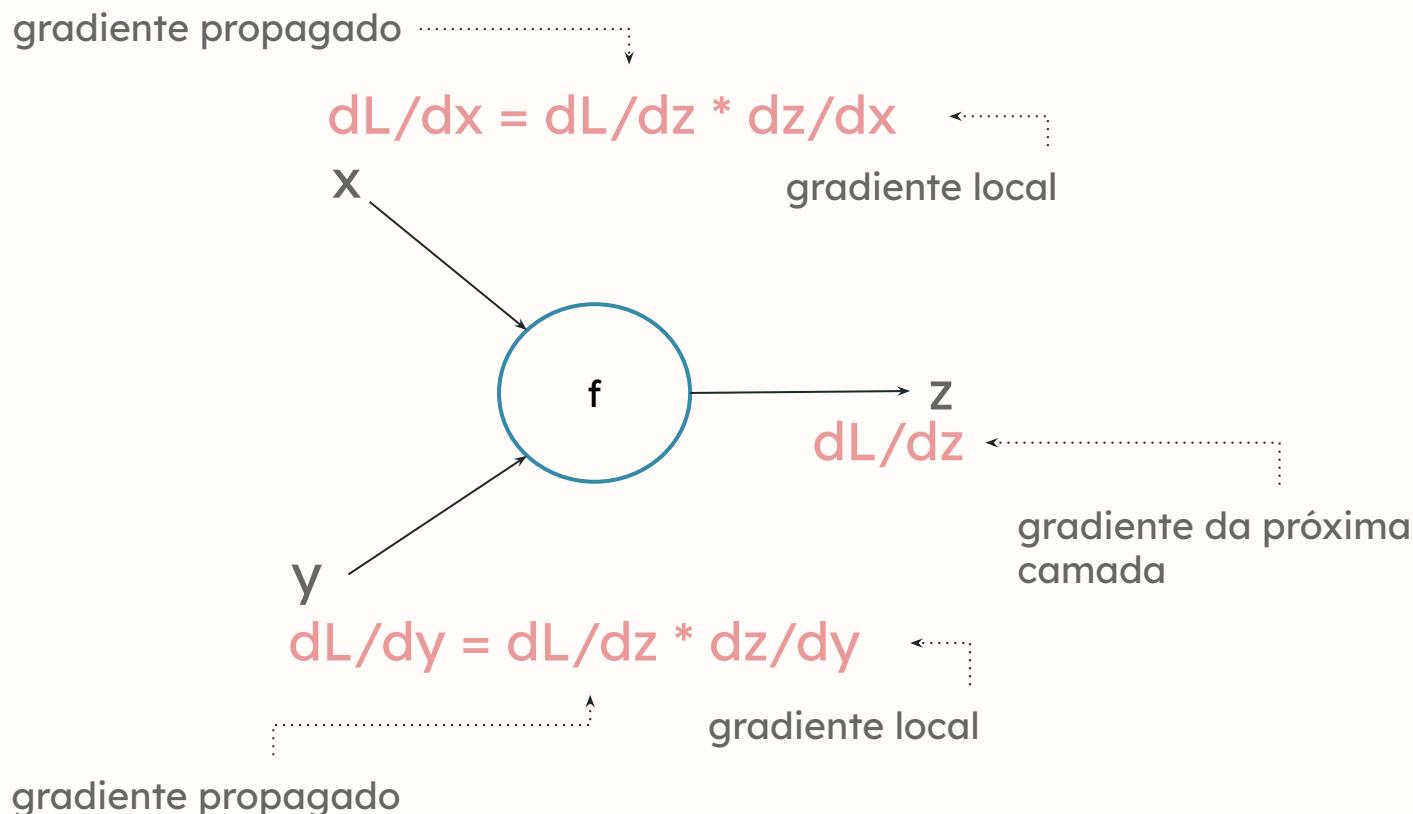
- Realizamos a operação de convolução para fazer o forward numa camada convolucional
- Mas como podemos fazer o backpropagation?
 - ◆ Aplicando outra convolução no sentido inverso

CNN Backpropagation

- Relembrando:
 - ◆ O Backward da rede neural é executado a partir do caminho inverso, usando a regra da cadeia para propagar o erro da saída até a entrada

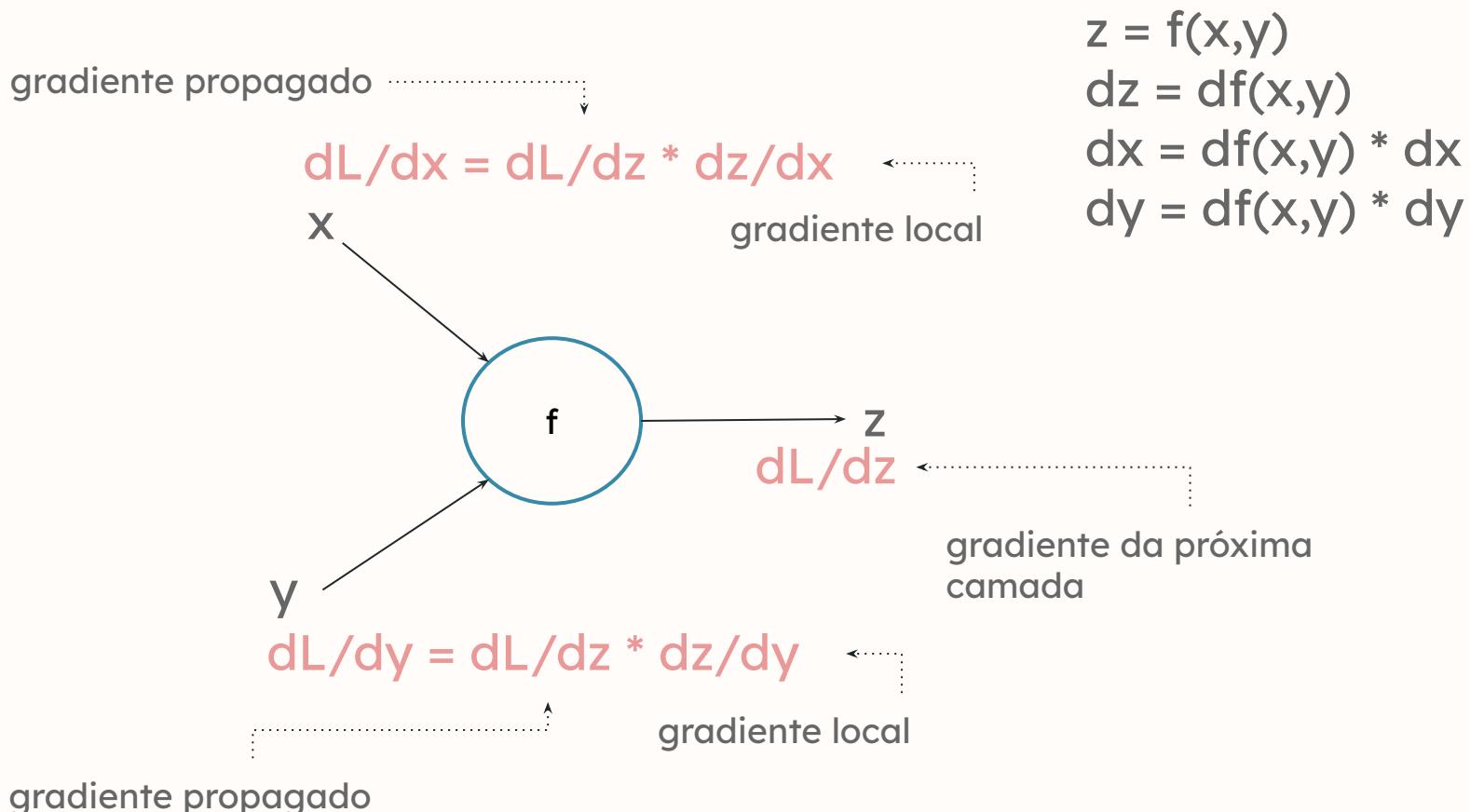
CNN Backpropagation

→ Seguindo essa ideia, para um neurônio temos:



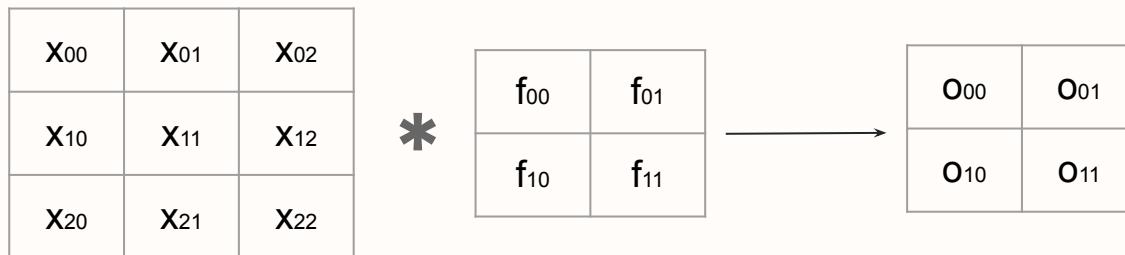
CNN Backpropagation

→ Seguindo essa ideia, para um neurônio temos:



CNN Backpropagation

- Vamos estender essa ideia para uma Conv Layer
 - ◆ Assumindo que temos uma ativação de entrada X ($3 \times 3 \times 1$) e vamos aplicar um filtro F ($2 \times 2 \times 1$), temos o forward como:



$$O_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{10}f_{10} + x_{11}f_{11}$$

$$O_{01} = x_{01}f_{00} + x_{02}f_{01} + x_{11}f_{10} + x_{12}f_{11}$$

$$O_{10} = x_{10}f_{00} + x_{11}f_{01} + x_{20}f_{10} + x_{21}f_{11}$$

$$O_{11} = x_{11}f_{00} + x_{12}f_{01} + x_{21}f_{10} + x_{22}f_{11}$$

CNN Backpropagation

- Quantas derivadas devemos calcular?



- Temos o erro da saída ajustado por dL/dO
- Para atualizar os pesos F , precisamos encontrar dL/dF
 - ◆ Pela regra da cadeia $dL/dF = dL/dO * dO/dF$
 - ◆ $F' = F - lr * dL/dF$
- Também precisamos calcular dL/dX
 - ◆ X será a saída da camada anterior, então calculamos o gradiente para ajustar os pesos da camada que retorna X (com exceção da primeira)
 - ◆ $dL/dX = dL/dO * dO/dX$

CNN Backpropagation

- Pelo exemplo anterior, computamos cada gradiente local com base no valor de saída. Nesse caso, temos que diferenciar a saída O com respeito ao filtro F (pesos)

$$O_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{10}f_{10} + x_{11}f_{11}$$

$$O_{01} = x_{01}f_{00} + x_{02}f_{01} + x_{11}f_{10} + x_{12}f_{11}$$

$$O_{10} = x_{10}f_{00} + x_{11}f_{01} + x_{20}f_{10} + x_{21}f_{11}$$

$$O_{11} = x_{11}f_{00} + x_{12}f_{01} + x_{21}f_{10} + x_{22}f_{11}$$

- Encontrando as derivadas com respeito ao filtro:

$$\frac{\partial O_{00}}{\partial f_{00}} = x_{00}, \quad \frac{\partial O_{00}}{\partial f_{01}} = x_{01}, \quad \frac{\partial O_{00}}{\partial f_{10}} = x_{10}, \quad \frac{\partial O_{00}}{\partial f_{11}} = x_{11}$$

$$\frac{\partial O_{01}}{\partial f_{00}} = x_{01}, \quad \frac{\partial O_{01}}{\partial f_{01}} = x_{02}, \quad \frac{\partial O_{01}}{\partial f_{10}} = x_{11}, \quad \frac{\partial O_{01}}{\partial f_{11}} = x_{12}$$

$$\frac{\partial O_{10}}{\partial f_{00}} = x_{10}, \quad \frac{\partial O_{10}}{\partial f_{01}} = x_{11}, \quad \frac{\partial O_{10}}{\partial f_{10}} = x_{20}, \quad \frac{\partial O_{10}}{\partial f_{11}} = x_{21}$$

$$\frac{\partial O_{11}}{\partial f_{00}} = x_{11}, \quad \frac{\partial O_{11}}{\partial f_{01}} = x_{12}, \quad \frac{\partial O_{11}}{\partial f_{10}} = x_{21}, \quad \frac{\partial O_{11}}{\partial f_{11}} = x_{22}$$

CNN Backpropagation

- Temos dO/dF , agora precisamos encontrar dL/dF
- ◆ O e F são matrizes e dO/dF será a derivada parcial da matrix O com respeito a F
 - ◆ Precisamos calcular a derivada parcial de cada componente de F ; $dL/dF = [dL/df_{00}, dL/df_{01}, dL/df_{10}, dL/df_{11}]$
 - ◆ Usando a regra da cadeia, fazemos para cada elemento de F :

$$O_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{10}f_{10} + x_{11}f_{11}$$

$$O_{01} = x_{01}f_{00} + x_{02}f_{01} + x_{11}f_{10} + x_{12}f_{11}$$

$$O_{10} = x_{10}f_{00} + x_{11}f_{01} + x_{20}f_{10} + x_{21}f_{11}$$

$$O_{11} = x_{11}f_{00} + x_{12}f_{01} + x_{21}f_{10} + x_{22}f_{11}$$

$$\frac{\partial L}{\partial f_{00}} = \frac{\partial L}{\partial O_{00}} \frac{\partial O_{00}}{\partial f_{00}} + \frac{\partial L}{\partial O_{01}} \frac{\partial O_{01}}{\partial f_{00}} + \frac{\partial L}{\partial O_{10}} \frac{\partial O_{10}}{\partial f_{00}} + \frac{\partial L}{\partial O_{11}} \frac{\partial O_{11}}{\partial f_{00}}$$
$$\frac{\partial L}{\partial f_{01}} = \frac{\partial L}{\partial O_{00}} \frac{\partial O_{00}}{\partial f_{01}} + \frac{\partial L}{\partial O_{01}} \frac{\partial O_{01}}{\partial f_{01}} + \frac{\partial L}{\partial O_{10}} \frac{\partial O_{10}}{\partial f_{01}} + \frac{\partial L}{\partial O_{11}} \frac{\partial O_{11}}{\partial f_{01}}$$
$$\frac{\partial L}{\partial f_{10}} = \frac{\partial L}{\partial O_{00}} \frac{\partial O_{00}}{\partial f_{10}} + \frac{\partial L}{\partial O_{01}} \frac{\partial O_{01}}{\partial f_{10}} + \frac{\partial L}{\partial O_{10}} \frac{\partial O_{10}}{\partial f_{10}} + \frac{\partial L}{\partial O_{11}} \frac{\partial O_{11}}{\partial f_{10}}$$
$$\frac{\partial L}{\partial f_{11}} = \frac{\partial L}{\partial O_{00}} \frac{\partial O_{00}}{\partial f_{11}} + \frac{\partial L}{\partial O_{01}} \frac{\partial O_{01}}{\partial f_{11}} + \frac{\partial L}{\partial O_{10}} \frac{\partial O_{10}}{\partial f_{11}} + \frac{\partial L}{\partial O_{11}} \frac{\partial O_{11}}{\partial f_{11}}$$

CNN Backpropagation

- Se substituirmos para cada dO/dF o valor que calculamos anteriormente, então:

$$\begin{aligned}\frac{\partial L}{\partial f_{00}} &= \frac{\partial L}{\partial O_{00}}x_{00} + \frac{\partial L}{\partial O_{01}}x_{01} + \frac{\partial L}{\partial O_{10}}x_{10} + \frac{\partial L}{\partial O_{11}}x_{11} \\ \frac{\partial L}{\partial f_{01}} &= \frac{\partial L}{\partial O_{00}}x_{01} + \frac{\partial L}{\partial O_{01}}x_{02} + \frac{\partial L}{\partial O_{10}}x_{11} + \frac{\partial L}{\partial O_{11}}x_{12} \\ \frac{\partial L}{\partial f_{10}} &= \frac{\partial L}{\partial O_{00}}x_{10} + \frac{\partial L}{\partial O_{01}}x_{11} + \frac{\partial L}{\partial O_{10}}x_{20} + \frac{\partial L}{\partial O_{11}}x_{21} \\ \frac{\partial L}{\partial f_{11}} &= \frac{\partial L}{\partial O_{00}}x_{11} + \frac{\partial L}{\partial O_{01}}x_{12} + \frac{\partial L}{\partial O_{10}}x_{21} + \frac{\partial L}{\partial O_{11}}x_{22}\end{aligned}$$

CNN Backpropagation

- Se compararmos este resultado com a operação realizada no forward, podemos notar a semelhança:

$$O_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{10}f_{10} + x_{11}f_{11}$$

$$O_{01} = x_{01}f_{00} + x_{02}f_{01} + x_{11}f_{10} + x_{12}f_{11}$$

$$O_{10} = x_{10}f_{00} + x_{11}f_{01} + x_{20}f_{10} + x_{21}f_{11}$$

$$O_{11} = x_{11}f_{00} + x_{12}f_{01} + x_{21}f_{10} + x_{22}f_{11}$$

$$\frac{\partial L}{\partial f_{00}} = \frac{\partial L}{\partial O_{00}}x_{00} + \frac{\partial L}{\partial O_{01}}x_{01} + \frac{\partial L}{\partial O_{10}}x_{10} + \frac{\partial L}{\partial O_{11}}x_{11}$$

$$\frac{\partial L}{\partial f_{01}} = \frac{\partial L}{\partial O_{00}}x_{01} + \frac{\partial L}{\partial O_{01}}x_{02} + \frac{\partial L}{\partial O_{10}}x_{11} + \frac{\partial L}{\partial O_{11}}x_{12}$$

$$\frac{\partial L}{\partial f_{10}} = \frac{\partial L}{\partial O_{00}}x_{10} + \frac{\partial L}{\partial O_{01}}x_{11} + \frac{\partial L}{\partial O_{10}}x_{20} + \frac{\partial L}{\partial O_{11}}x_{21}$$

$$\frac{\partial L}{\partial f_{11}} = \frac{\partial L}{\partial O_{00}}x_{11} + \frac{\partial L}{\partial O_{01}}x_{12} + \frac{\partial L}{\partial O_{10}}x_{21} + \frac{\partial L}{\partial O_{11}}x_{22}$$

CNN Backpropagation

- De forma geral:

$$\begin{array}{|c|c|c|} \hline x_{00} & x_{01} & x_{02} \\ \hline x_{10} & x_{11} & x_{12} \\ \hline x_{20} & x_{21} & x_{22} \\ \hline \end{array} * \begin{array}{|c|c|} \hline dL/dO_{00} & dL/dO_{01} \\ \hline dL/dO_{10} & dL/dO_{11} \\ \hline \end{array} = \begin{array}{|c|c|} \hline dL/df_{00} & dL/df_{01} \\ \hline dL/df_{10} & dL/df_{11} \\ \hline \end{array}$$

- dL/dF é apenas a convolução entre a entrada da camada X e o loss (gradiente) da próxima camada dL/dO

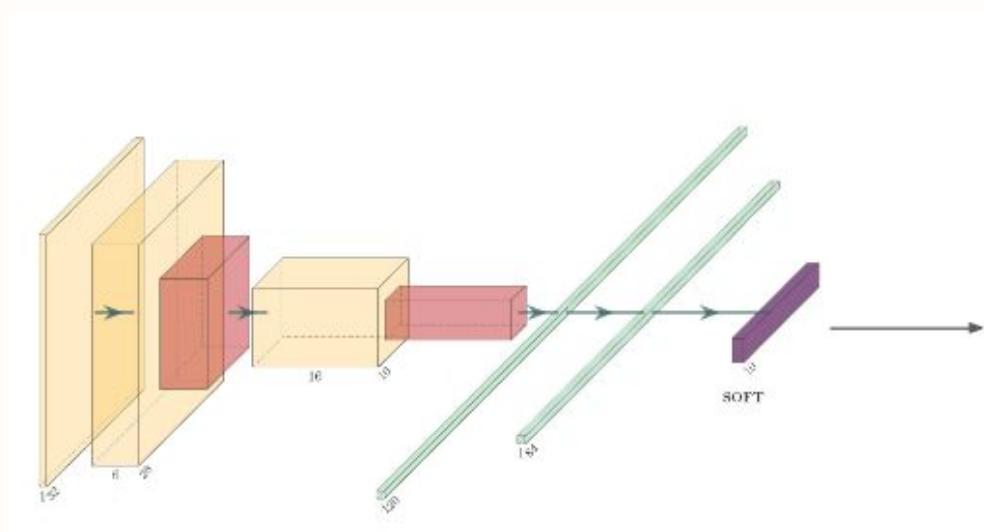
CNN Backpropagation

- Encontramos o nosso gradiente local da nossa camada (e.g., derivada da camada posterior com o filtro da camada atual, dL/dF)
- Para concluir a cadeia da nossa camada, precisamos calcular a derivada da saída com respeito as ativações de entrada (dL/dX), que será propagada para camada anterior

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^M \frac{\partial L}{\partial O_k} \frac{\partial O_k}{\partial X_i}$$

CNN Backpropagation

- Porque calcular a derivada com respeito a entrada?
 - ◆ Com exceção da primeira camada, a entrada de cada camada convolucional (I) é dada pela saída da camada posterior ($I-1$) (que é computada em função dos filtros da camada I)



CNN Backpropagation

$$O_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{10}f_{10} + x_{11}f_{11}$$

$$O_{01} = x_{01}f_{00} + x_{02}f_{01} + x_{11}f_{10} + x_{12}f_{11}$$

$$O_{10} = x_{10}f_{00} + x_{11}f_{01} + x_{20}f_{10} + x_{21}f_{11}$$

$$O_{11} = x_{11}f_{00} + x_{12}f_{01} + x_{21}f_{10} + x_{22}f_{11}$$

$$\frac{\partial L}{\partial X_{00}} = \frac{\partial L}{\partial O_{00}} f_{00}$$

$$\frac{\partial L}{\partial X_{01}} = \frac{\partial L}{\partial O_{00}} f_{01} + \frac{\partial L}{\partial O_{01}} f_{00}$$

$$\frac{\partial L}{\partial X_{02}} = \frac{\partial L}{\partial O_{01}} f_{01}$$

$$\frac{\partial L}{\partial X_{10}} = \frac{\partial L}{\partial O_{00}} f_{10} + \frac{\partial L}{\partial O_{10}} f_{00}$$

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{00}} f_{11} + \frac{\partial L}{\partial O_{01}} f_{10} + \frac{\partial L}{\partial O_{10}} f_{01} + \frac{\partial L}{\partial O_{11}} f_{00}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{01}} f_{11} + \frac{\partial L}{\partial O_{11}} f_{01}$$

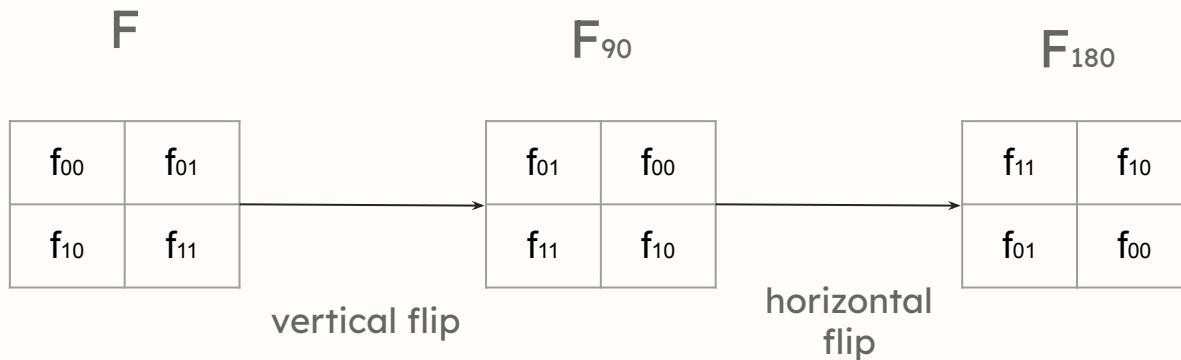
$$\frac{\partial L}{\partial X_{20}} = \frac{\partial L}{\partial O_{10}} f_{10}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{10}} f_{11} + \frac{\partial L}{\partial O_{11}} f_{10}$$

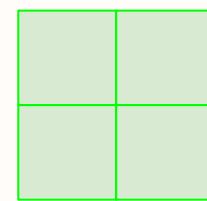
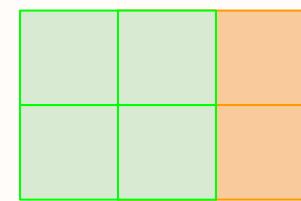
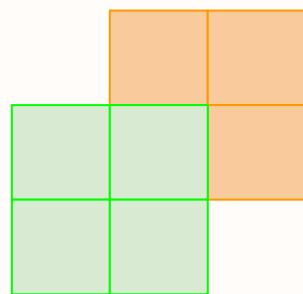
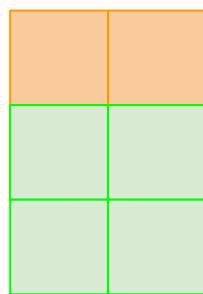
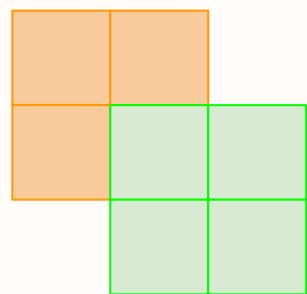
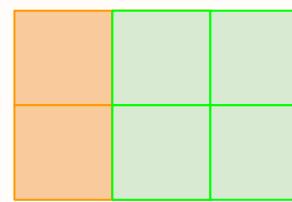
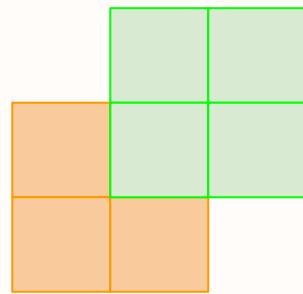
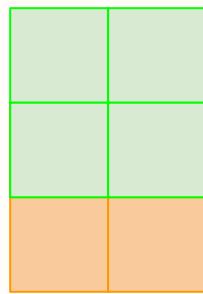
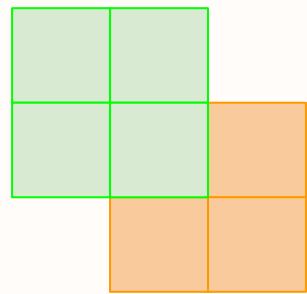
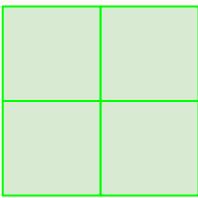
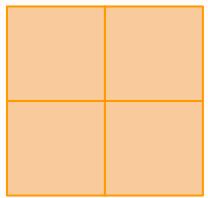
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} f_{11}$$

CNN Backpropagation

- Podemos representar este resultado também como uma convolução
 - ◆ Utilizando a definição padrão de convolução: Rotacionando o kernel F 180 graus e aplicando a operação full-convolution
 - ◆ Para rotacionar uma matriz em 180°, fazemos um flip vertical seguido de um flip horizontal

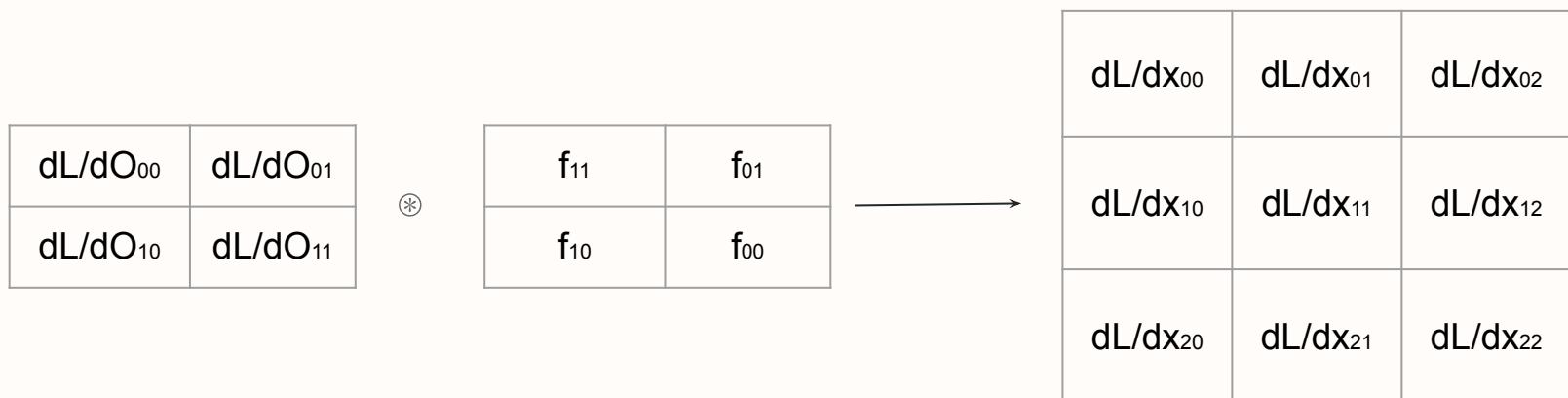


CNN Backpropagation - Full-convolution

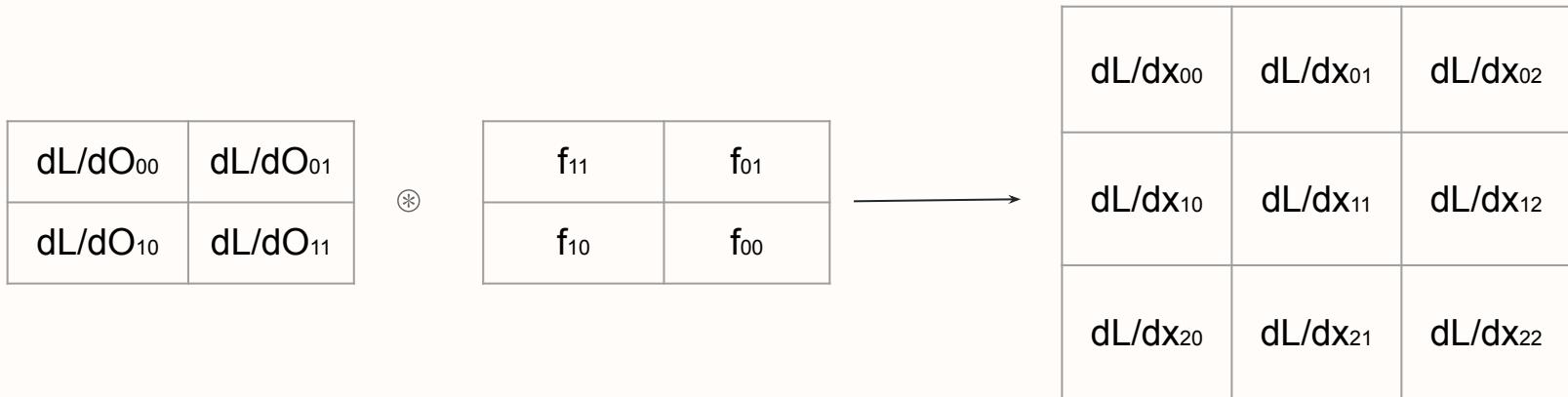


CNN Backpropagation

- Com isso, encontramos dL/dF para ajustar os pesos da camada atual e dL/dX , propagando o erro para as camadas seguintes



CNN Backpropagation



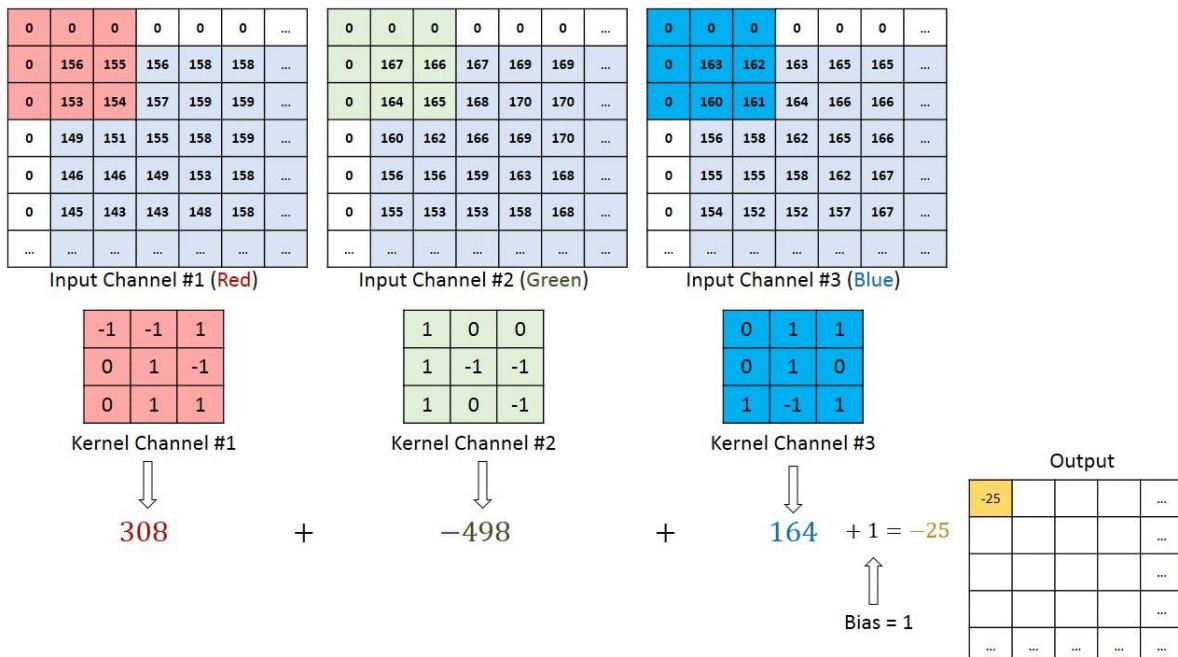
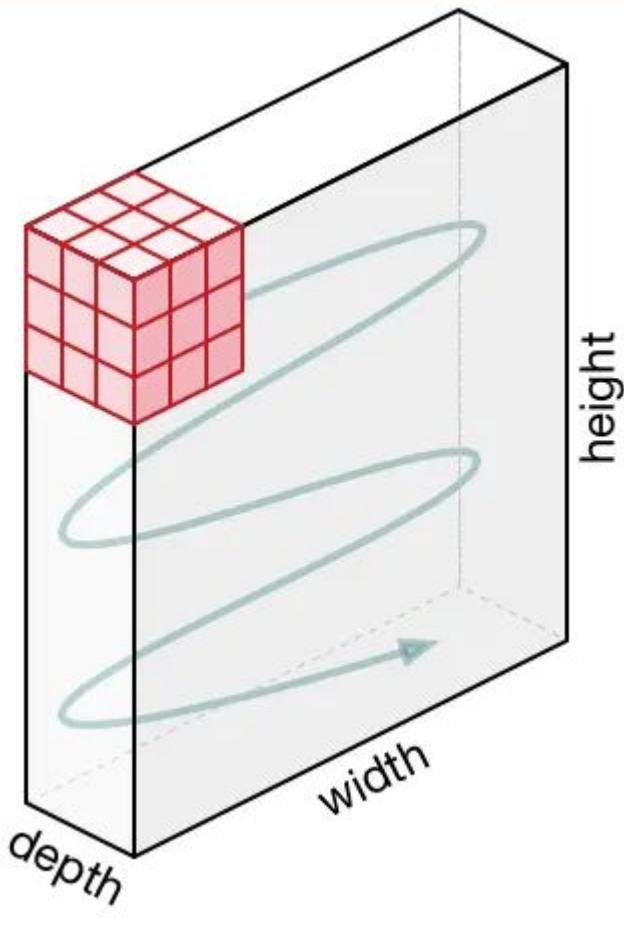
$$\begin{aligned}\frac{\partial L}{\partial X_{00}} &= \frac{\partial L}{\partial O_{00}} f_{00} \\ \frac{\partial L}{\partial X_{01}} &= \frac{\partial L}{\partial O_{00}} f_{01} + \frac{\partial L}{\partial O_{01}} f_{00} \\ \frac{\partial L}{\partial X_{02}} &= \frac{\partial L}{\partial O_{01}} f_{01} \\ \frac{\partial L}{\partial X_{10}} &= \frac{\partial L}{\partial O_{00}} f_{10} + \frac{\partial L}{\partial O_{10}} f_{00} \\ \frac{\partial L}{\partial X_{11}} &= \frac{\partial L}{\partial O_{00}} f_{11} + \frac{\partial L}{\partial O_{01}} f_{10} + \frac{\partial L}{\partial O_{10}} f_{01} + \frac{\partial L}{\partial O_{11}} f_{00}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial X_{12}} &= \frac{\partial L}{\partial O_{01}} f_{11} + \frac{\partial L}{\partial O_{11}} f_{01} \\ \frac{\partial L}{\partial X_{20}} &= \frac{\partial L}{\partial O_{10}} f_{10} \\ \frac{\partial L}{\partial X_{21}} &= \frac{\partial L}{\partial O_{10}} f_{11} + \frac{\partial L}{\partial O_{11}} f_{10} \\ \frac{\partial L}{\partial X_{22}} &= \frac{\partial L}{\partial O_{11}} f_{11}\end{aligned}$$

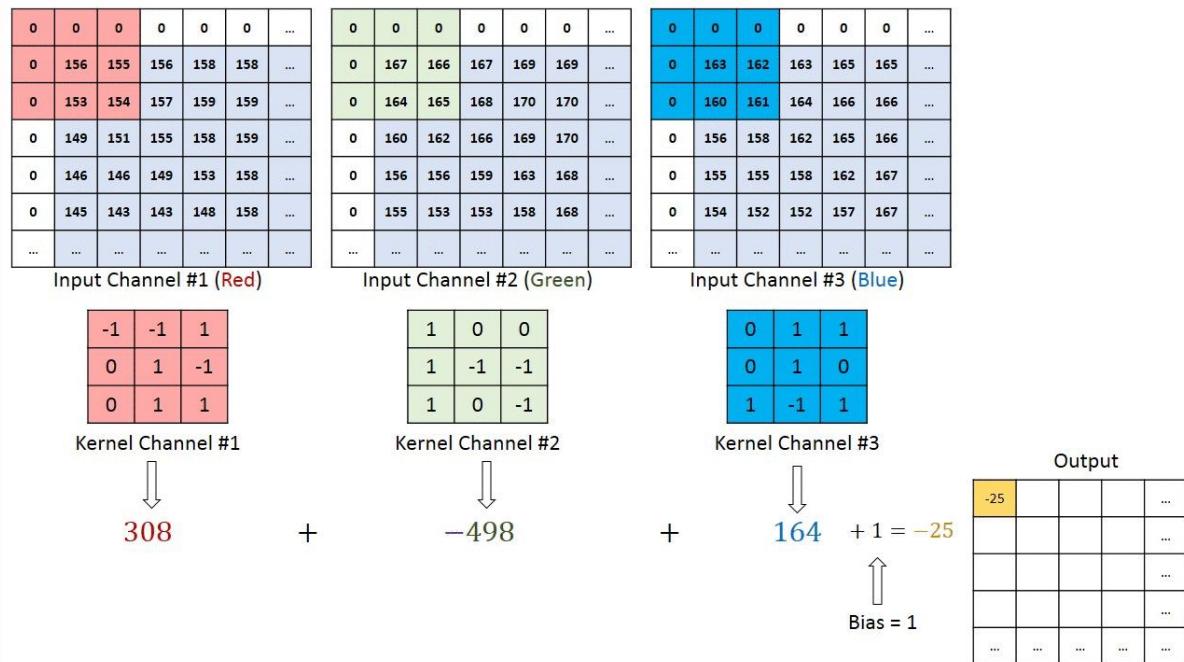
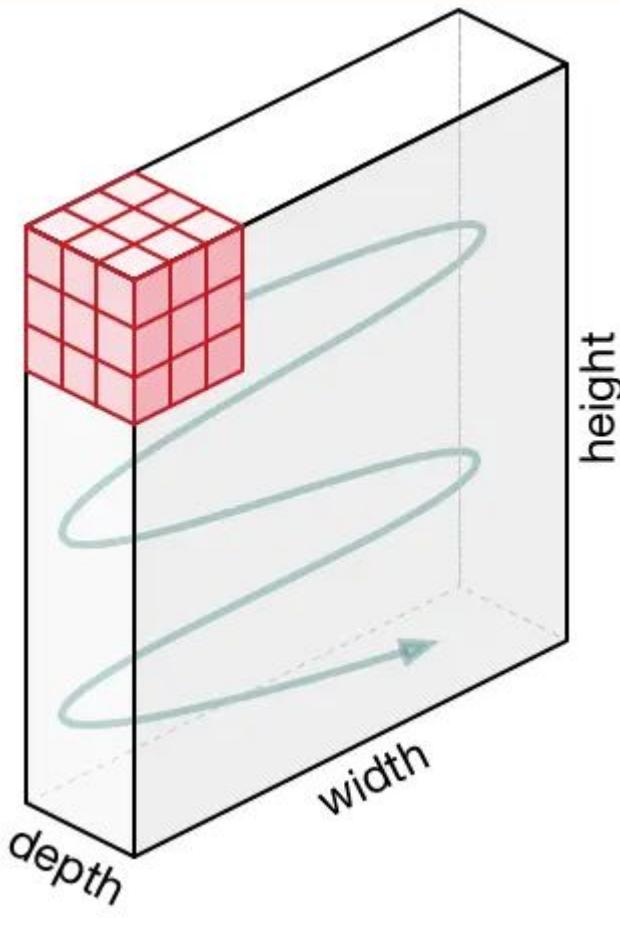
CNN Backpropagation

- Para uma camada convolutional, o forward e o backward é realizado através de convoluções
 - ◆ $dL/dF = \text{Conv}(X, dL/dO)$
 - ◆ $dL/dX = \text{Full-Conv}(F(180), dL/dO)$
 - ◆ Para a camada anterior $X \rightarrow O$
- Na prática, aplicamos essas operações para os N filtros ($K \times K$) da camada, cada um atuando na região correspondente do volume de entrada X , com dimensão ($H \times W \times C$)

Camadas Convolucionais



Camadas Convolucionais



→ Quantos gradientes são armazenados para uma camada convolucional?

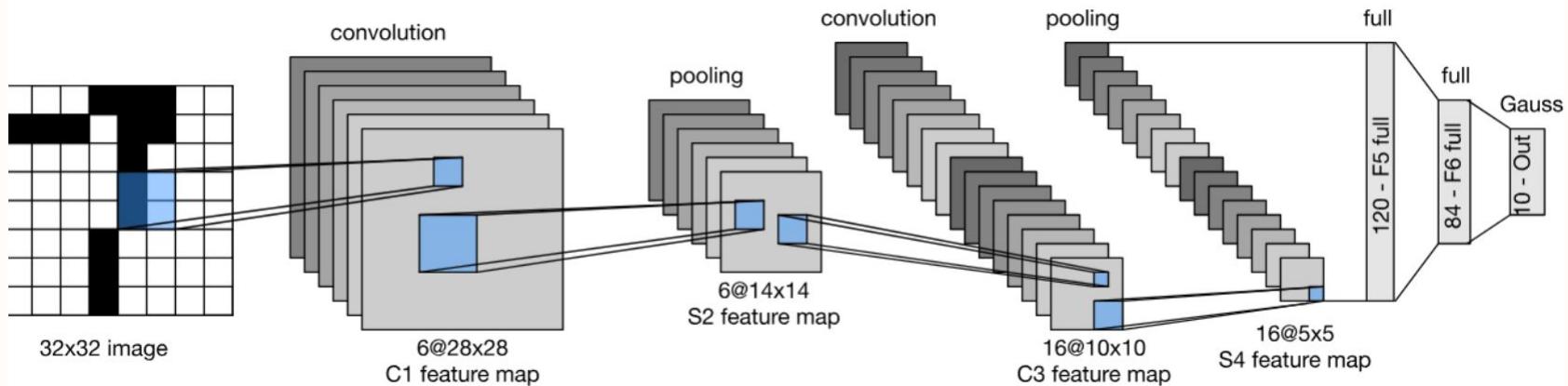
Pooling Backpropagation

- E para o pooling como propagamos o erro?
 - ◆ O pooling não possui parâmetros, sendo uma camada que não possui gradientes
 - ◆ No caso do backward, recebemos a matriz de gradientes da camada posterior e copiamos os valores para a mesma dimensão da camada anterior ao pooling, continuando o processo de propagação

Evolução das arquiteturas de classificação e redes convolucionais modernas

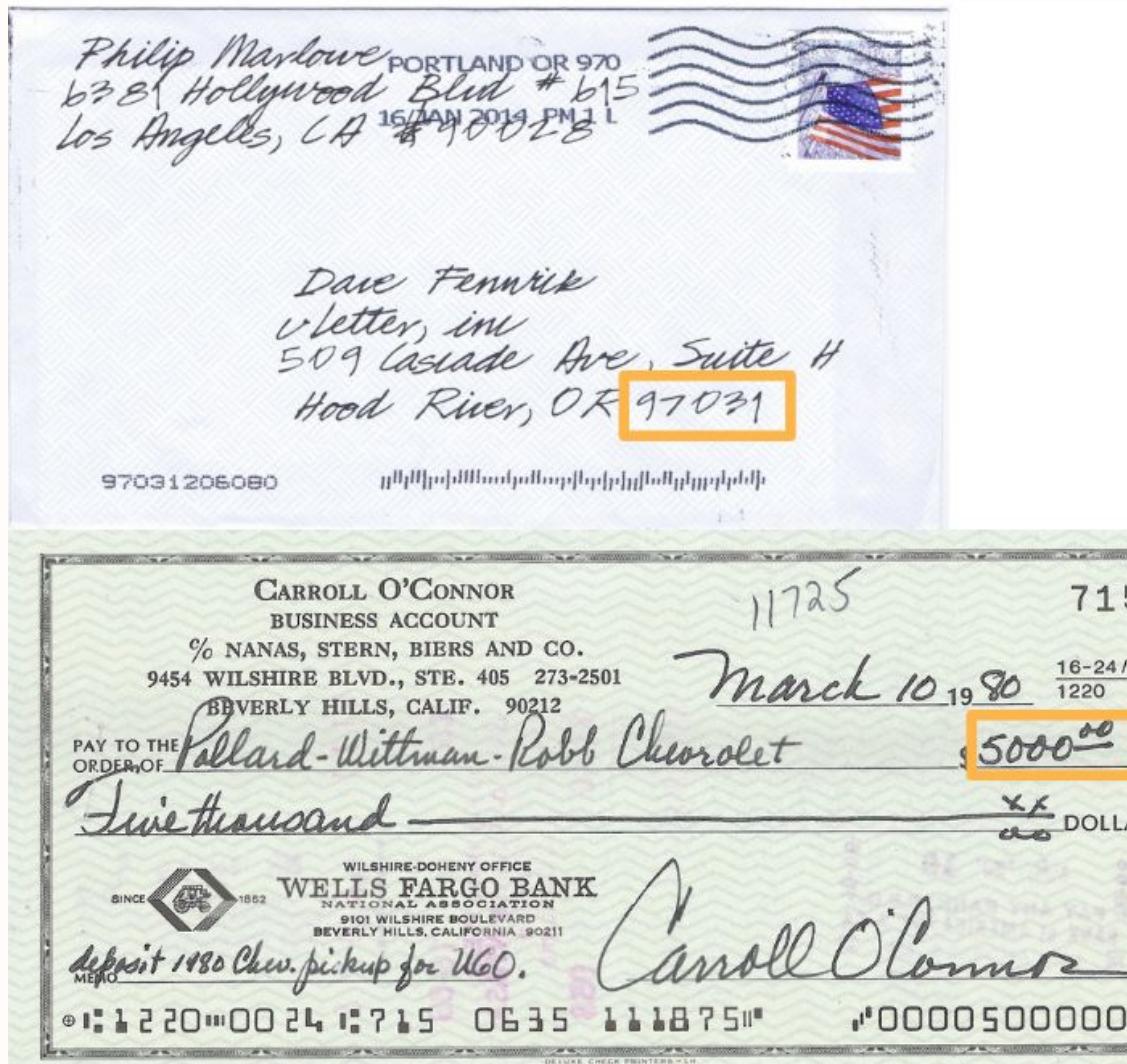
LeNet (1989)

→ LeNet: Reconhecimento de dígitos pela caligrafia.



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

LeNet (1989)



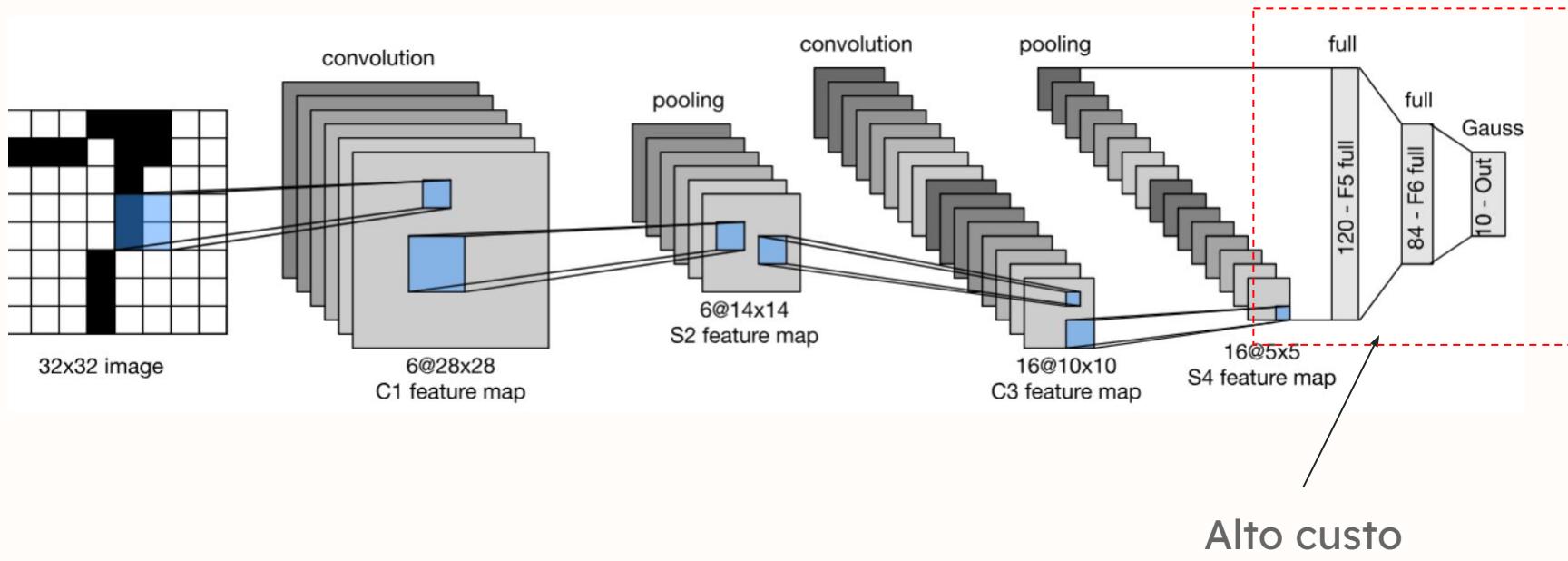
LeNet (1998)

- Dados: MNIST
 - 60.000 dados de treino;
 - 10.000 dados de teste;
 - 28x28 de dimensão;
 - 10 classes
- Dificuldades da época em relação aos hardwares



LeNet (1989)

→ LeNet: Reconhecimento de dígitos pela caligrafia.



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	10^3	10^4 OCR	10^{7-8} web	10^{10} advertising	10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	>1PF (8xP3 Volta)

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	$10x$ 10^3	$10x$ OCR	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ $(8xP3 Volta)$

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10 ² (e.g. iris)	10x 10 ³	10x 10 ⁴ OCR	100x 10 ⁷⁻⁸ web	100x 10 ¹⁰ advertising	1,000x 10 ¹² social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	>1PF (8xP3 Volta) 10,000x

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	$10x$ 10^3	$10x$ 10^4 OCR deep nets	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ (8xP3 Volta) $10,000x$

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	$10x$ 10^3	$10x$ 10^4 OCR deep nets	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ (8xP3 Volta)

Hardwares

	1970	1980	1990	2000	2010	2020
Data (samples)	10^2 (e.g. iris)	$10x$ 10^3	$10x$ 10^4 OCR deep nets	$100x$ 10^{7-8} web	$100x$ 10^{10} advertising	$1,000x$ 10^{12} social nets deep nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	$>1PF$ (8xP3 Volta)

ImageNet Large Scale Visual Recognition challenge (ILSVRC)

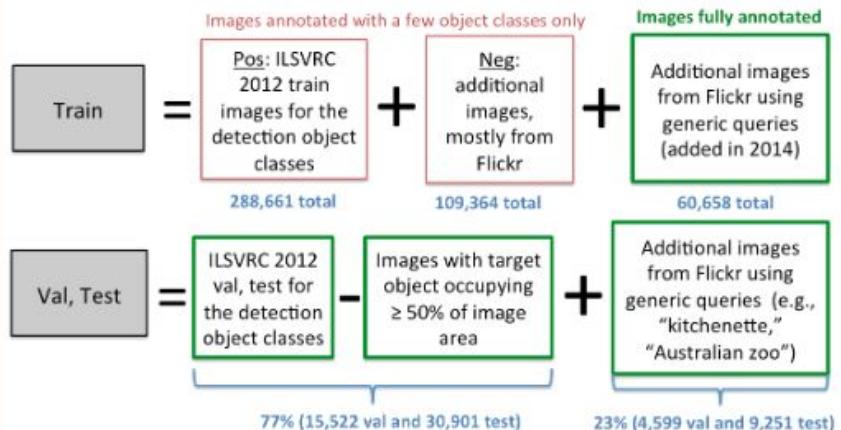


2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6

Images	Color images with nature objects	Gray image for handwritten digits
Size	469 x 387	28 x 28
# examples	1.2 M	60 K
# classes	1,000	10

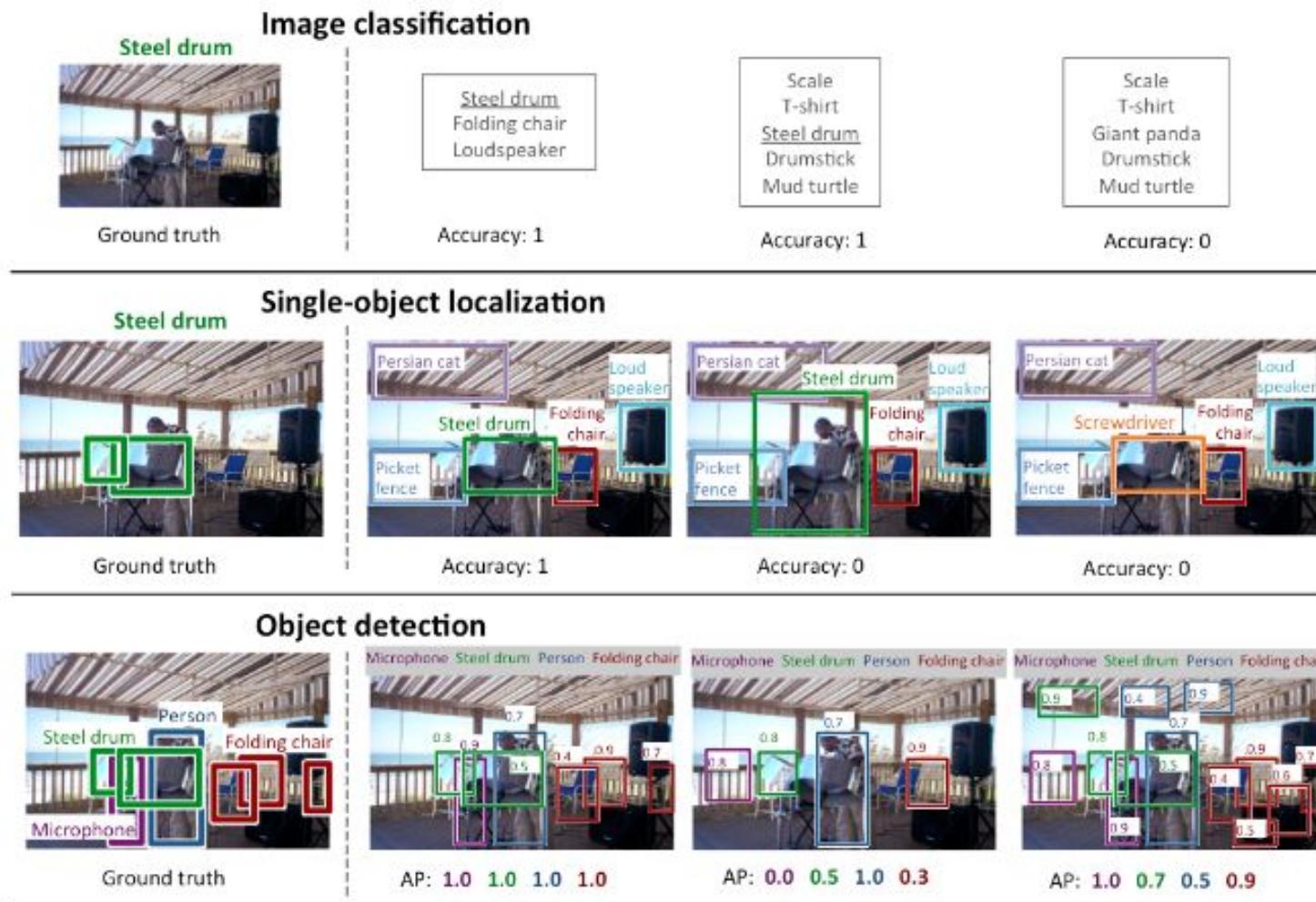
Fonte da Imagem: courses.d2l.ai/berkeley-stat-157

ImageNet Large Scale Visual Recognition challenge (ILSVRC)



Class name in PASCAL VOC (20 classes)	Closest class in ILSVRC-DET (200 classes)	Avg object scale (%)	
		PASCAL VOC	ILSVRC-DET
aeroplane	airplane	29.7	22.4
bicycle	bicycle	29.3	14.3
bird	bird	15.9	20.1
boat	watercraft	15.2	16.5
bottle	wine bottle	7.3	10.4
bus	bus	29.9	22.1
car	car	14.0	13.4
cat	domestic cat	46.8	29.8
chair	chair	12.8	10.1
cow	cattle	19.3	13.5
dining table	table	29.1	30.3
dog	dog	37.0	28.9
horse	horse	29.5	18.5
motorbike	motorcycle	32.0	20.7
person	person	17.5	19.3
potted plant	flower pot	12.3	8.1
sheep	sheep	12.2	17.3
sofa	sofa	41.7	44.4
train	train	35.4	35.1
tv/monitor	tv or monitor	14.6	11.2

ImageNet Large Scale Visual Recognition challenge (ILSVRC)



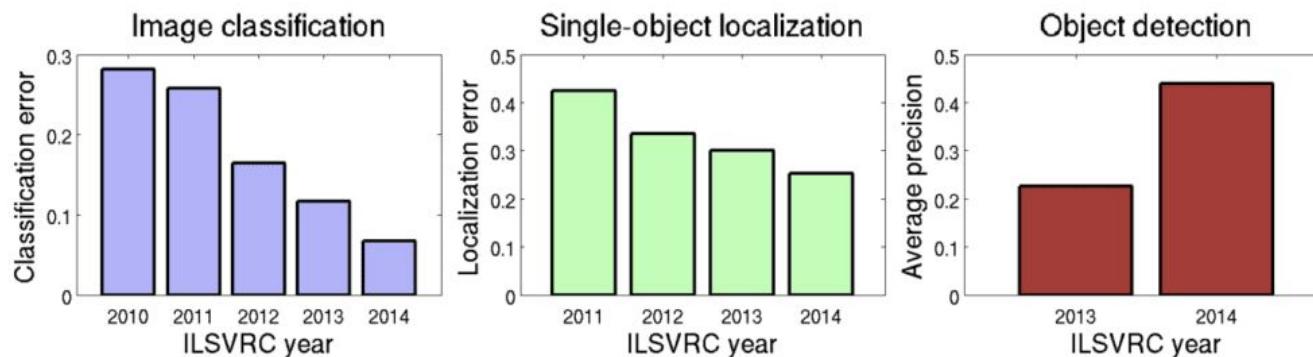
ImageNet Large Scale Visual Recognition challenge (ILSVRC)

Image classification annotations (1000 object classes)

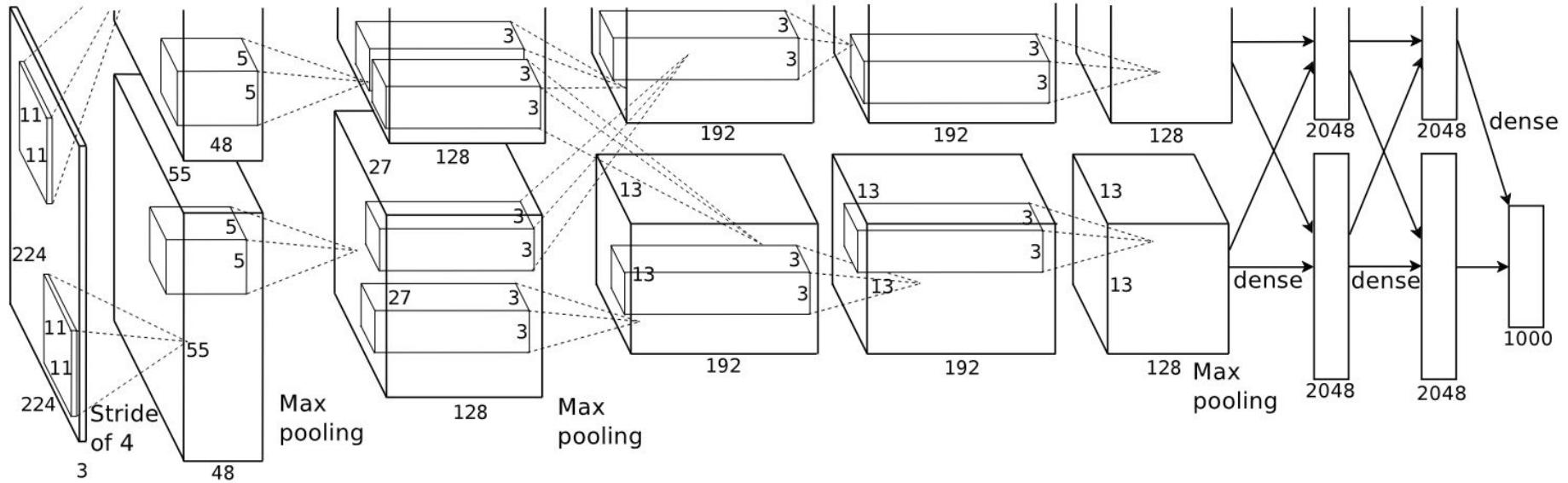
Year	Train images (per class)	Val images (per class)	Test images (per class)
ILSVRC2010	1,261,406 (668-3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384-1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732-1300)	50,000 (50)	100,000 (100)

Additional annotations for single-object localization (1000 object classes)

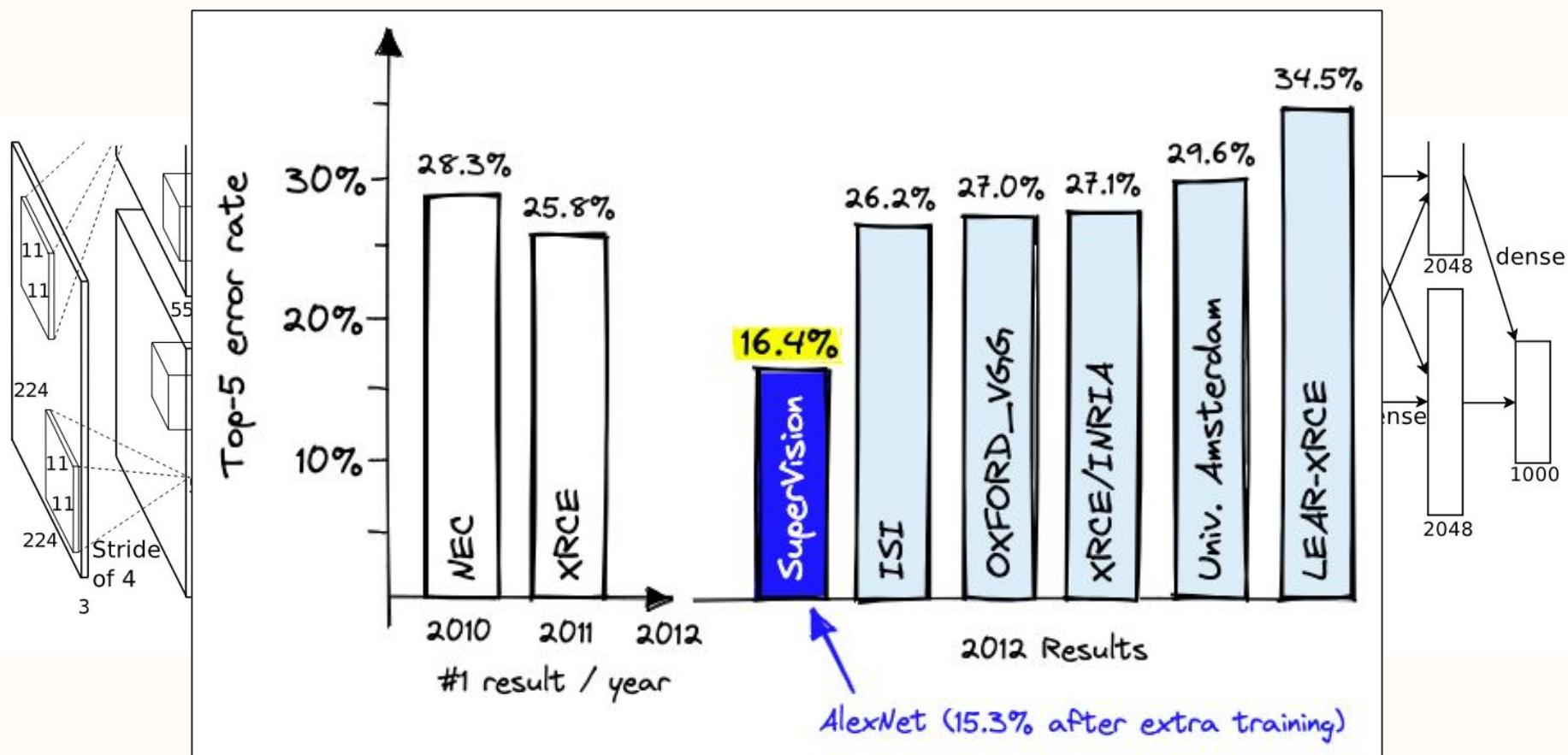
Year	Train images with bbox annotations (per class)	Train bboxes annotated (per class)	Val images with bbox annotations (per class)	Val bboxes annotated (per class)	Test images with bbox annotations
ILSVRC2011	315,525 (104-1256)	344,233 (114-1502)	50,000 (50)	55,388 (50-118)	100,000
ILSVRC2012-14	523,966 (91-1268)	593,173 (92-1418)	50,000 (50)	64,058 (50-189)	100,000



AlexNet



AlexNet



AlexNet

→ Principais melhorias:

- ◆ ReLU
- ◆ Dropout
- ◆ 8 Camadas (5 Camadas convolucionais, 3 camadas conectadas)
- ◆ Múltiplas GPUs
- ◆ Data augmentation

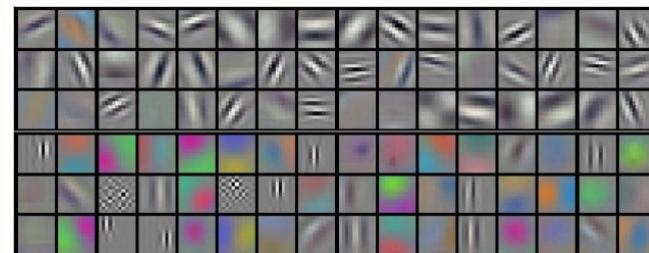
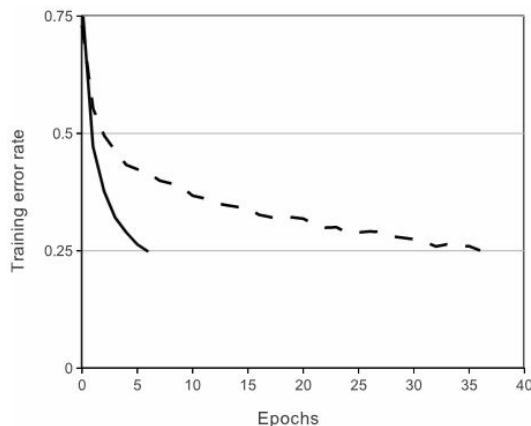
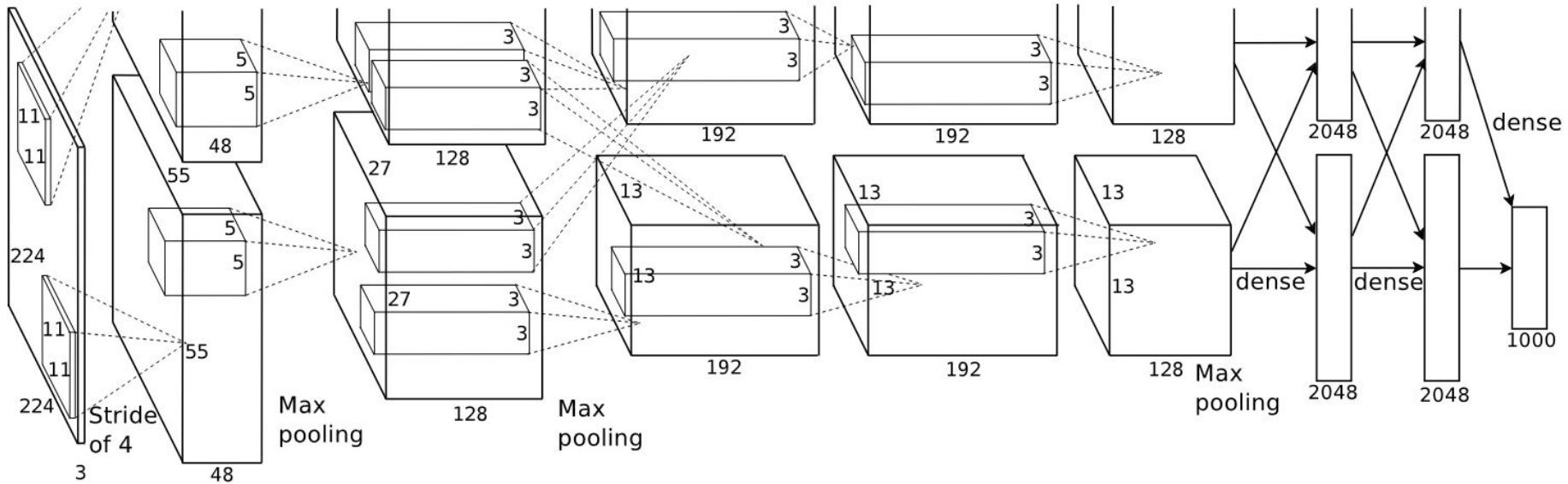


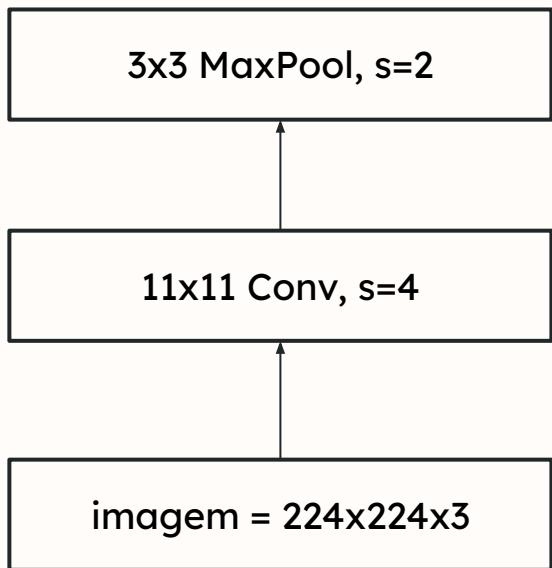
Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

AlexNet

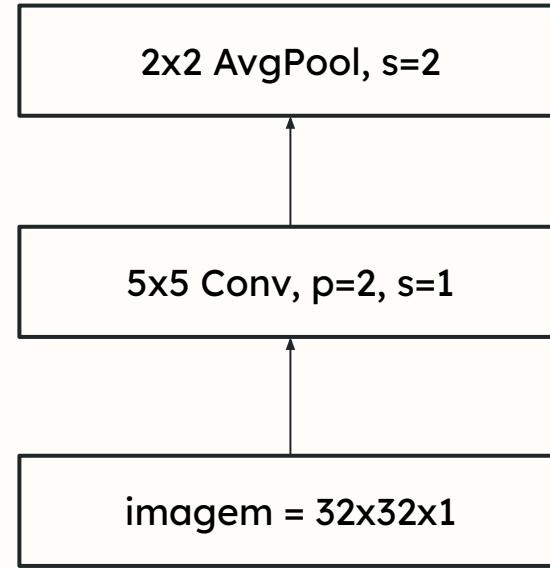


AlexNet

AlexNet



LeNet

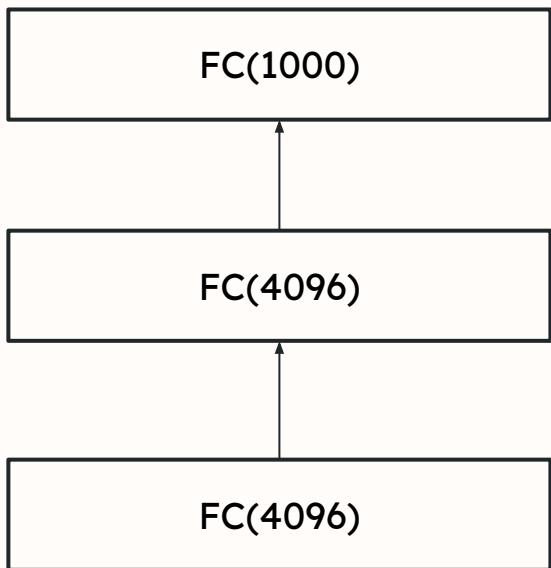


- Tamanho da entrada maior
- Kernels maiores para reduzir entrada, mais canais de saída
- Pooling maior, usando o valor de máximo

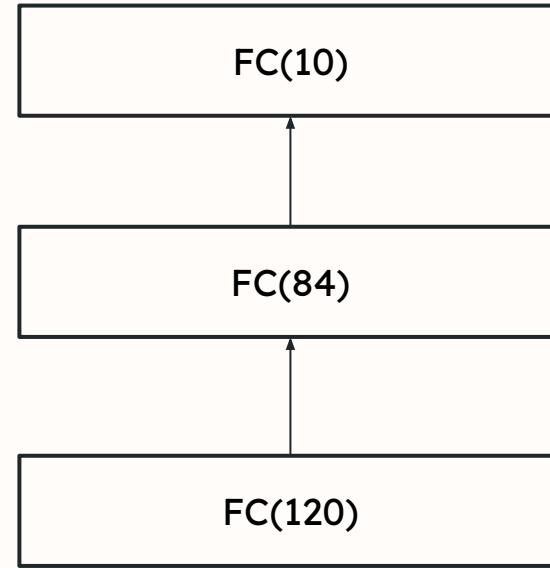
- Imagem pequena, 1 canal
- Kernels menores
- Pooling com valor médio

AlexNet

AlexNet



LeNet

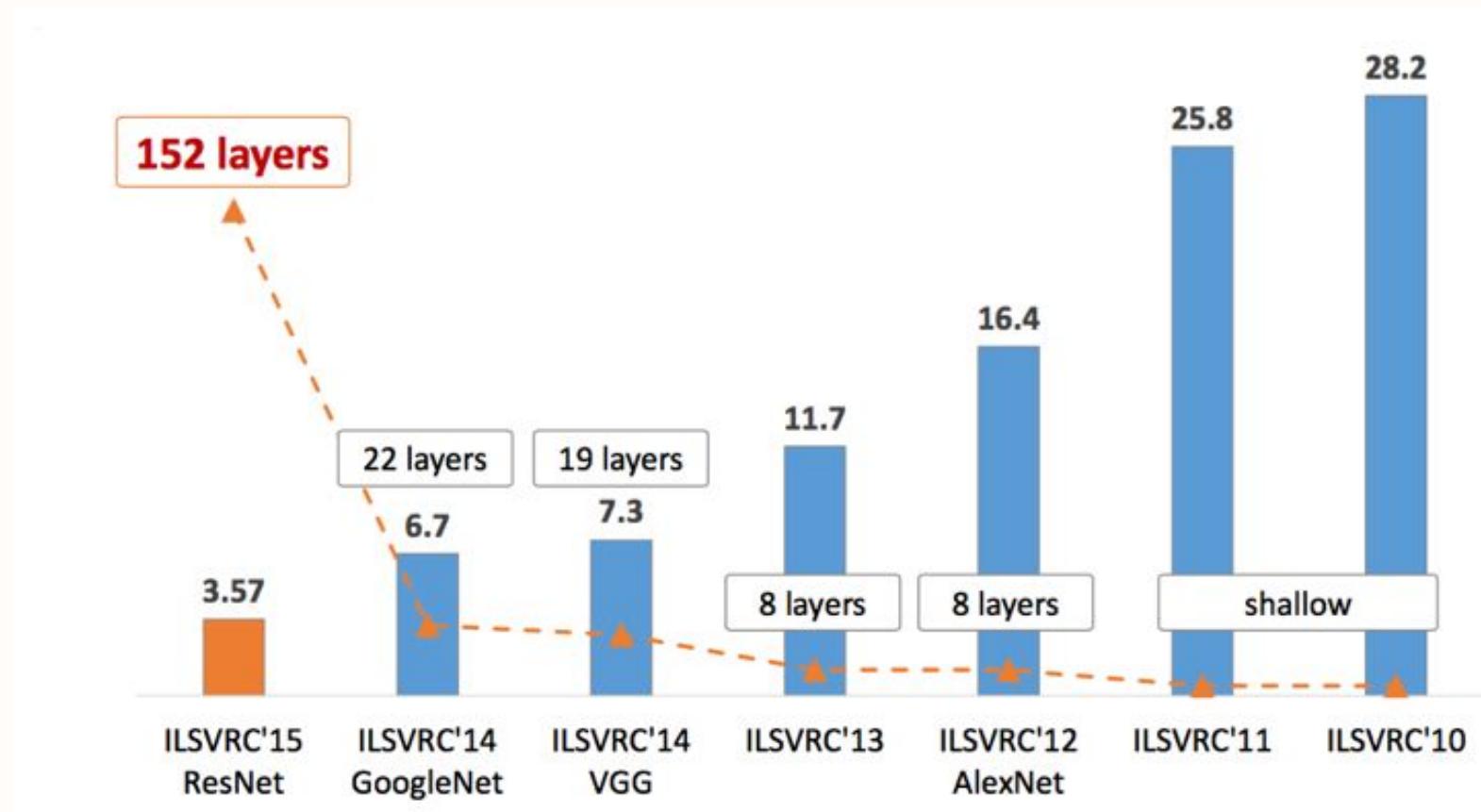


→ 1000 classes de saída

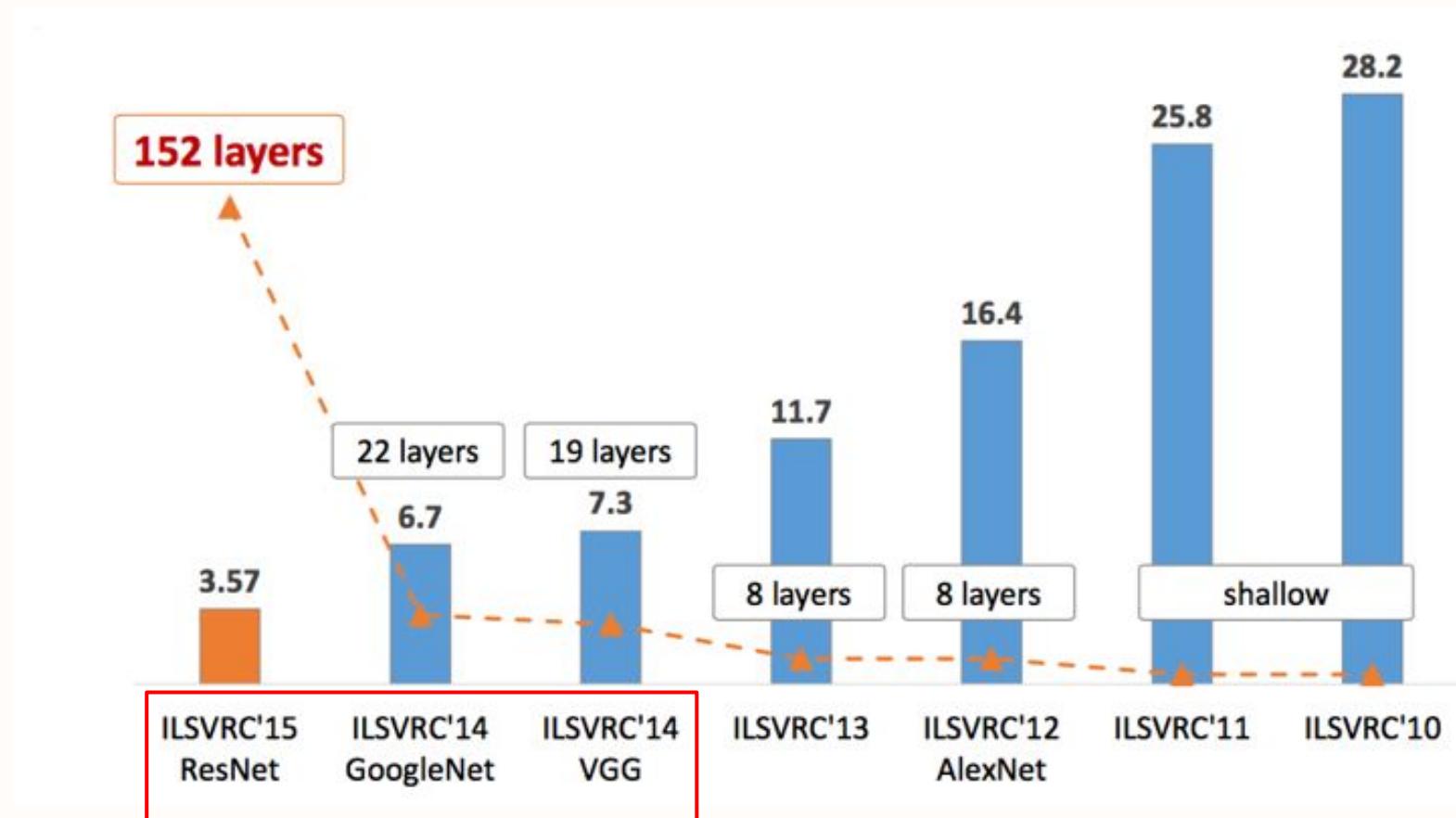
AlexNet

	#parameters		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x

ImageNet - Pós-AlexNet (2012-2015)

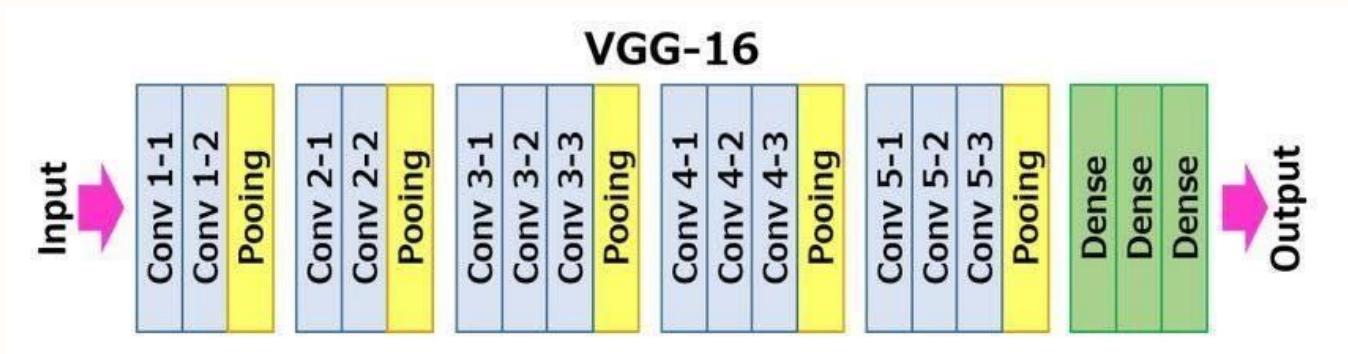


ImageNet - Pós-AlexNet (2012-2015)



VGG

- A VGG é uma arquitetura ‘simples’ e aplica kernels de tamanho 3×3 ao invés de 11×11 como na AlexNet
 - ◆ Ter 2 filtros 3×3 consecutivos é equivalente a um campo receptivo de 5×5
 - ◆ Da mesma forma, 3 filtros 3×3 consecutivos entregam um campo de 7×7
- Com isto, pode-se treinar redes mais profundas, otimizando menos parâmetros
 - ◆ Se uma convolução tem C canais, 3 filtros 3×3 tem $3(3^2C^2) = 27C^2$ parâmetros, enquanto que 1 filtro 7×7 tem $7^2C^2 = 49C^2$



VGG

- Ao invés de utilizar muitos hiperparâmetros a VGG foca em ter camadas convolucionais 3×3 com stride 1, pad constante e max pooling 2×2 após cada bloco
- Para a VGG-16, cada bloco possui camadas convolucionais com 64, 128, 256, 512 e 512 filtros (canais), respectivamente.

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 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 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 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

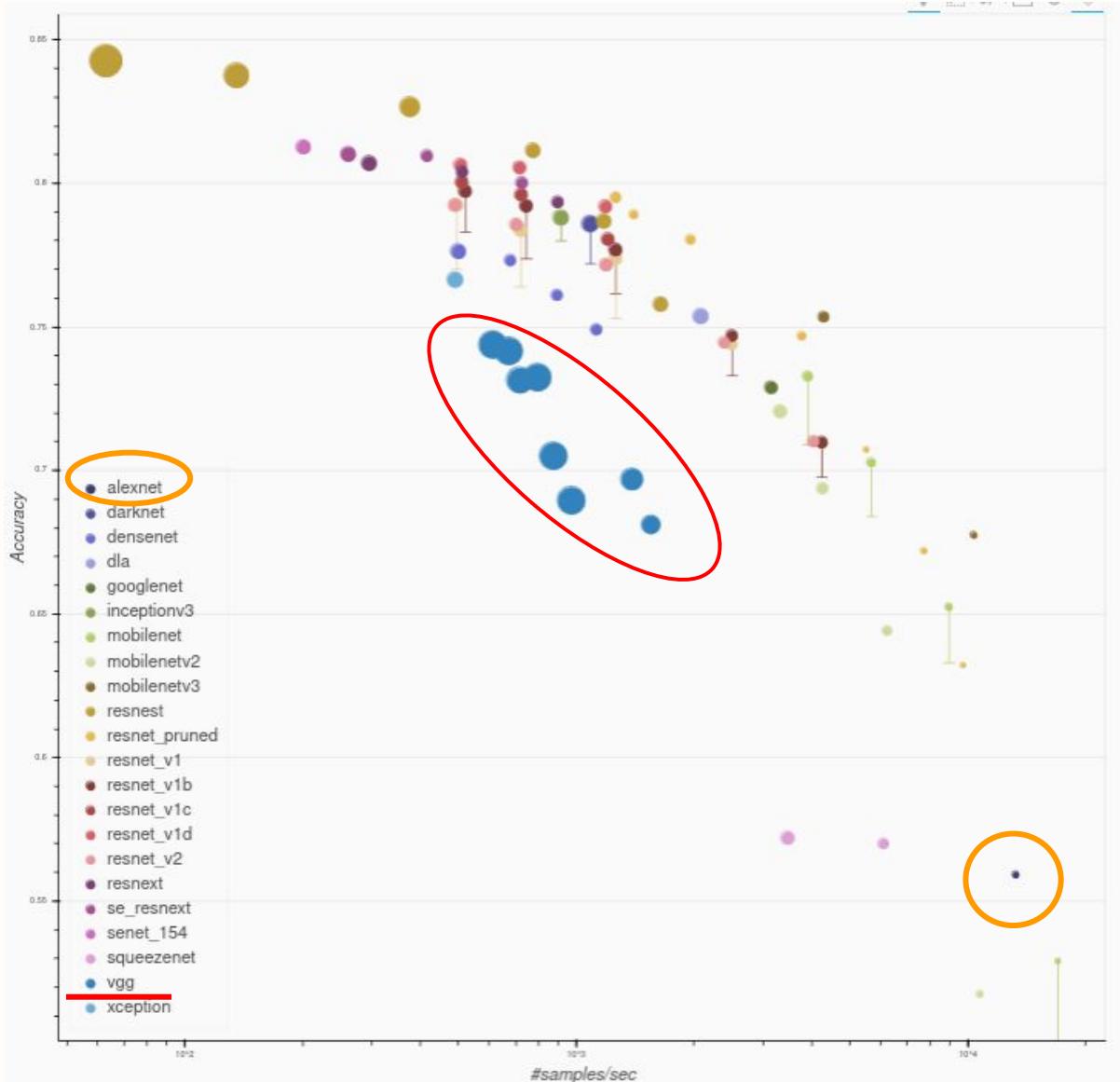
Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG

- LeNet: 2 Conv + pooling, 2 FC Dense
- AlexNet: Maior arquitetura, imagens RGB, uso de ReLU, dropout e data augmentation
- VGG: Maior e mais profundo que AlexNet, kernels menores aplicados em sequência

VGG

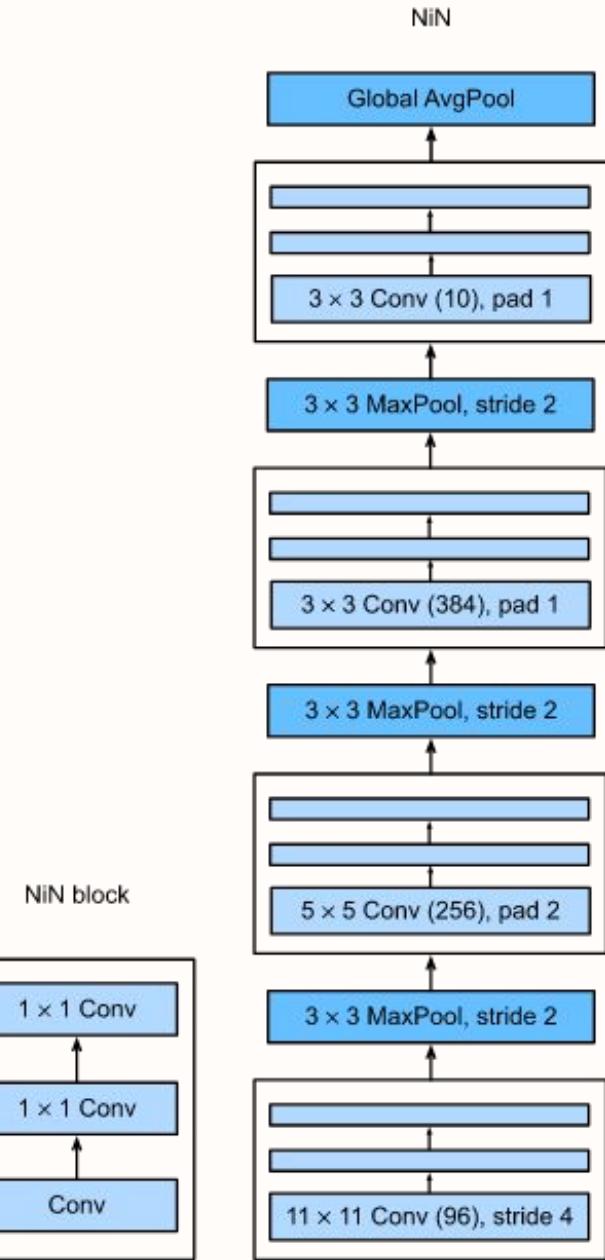


NiN (Network in Network)

- LeNet, AlexNet e VGG compartilham um padrão de design comum: extrair recursos que exploram a estrutura espacial por meio de uma **sequência de convoluções e camadas de pooling**, processando no final as features geradas através de **camadas FC (i.e. Dense Layers)**.
 - ◆ As camadas totalmente conectadas no final da arquitetura possuem um grande número de parâmetros e utilizam grande parte do processamento
 - ◆ Nesse esquema não é possível utilizar camadas totalmente conectadas em partes intermediárias da rede, limitando o grau de não linearidade que podemos adicionar: isso destruiria a estrutura espacial e exigiria potencialmente ainda mais memória.

NiN (Network in Network)

→ Network in Network (NiN)
tenta trazer uma alternativa
com blocos compostos de
convoluçãoções 1x1 para adicionar
não linearidades através dos
canais de ativação e uso de
global average pooling para
integrar todas as posições,
substituindo as camadas
totalmente conectadas



NiN (Network in Network) - 1x1 Convolution

- Uma convolução 1x1 mapeia um pixel da entrada (levando em consideração todos os canais) para uma ativação de saída, não levando em consideração nenhuma informação dos vizinhos
 - ◆ Uma camada de pooling reduz a dimensão espacial mantendo a profundidade do bloco (número de canais)
 - ◆ A convolução 1x1 pode alterar o número de canais, mantendo a dimensão original da entrada

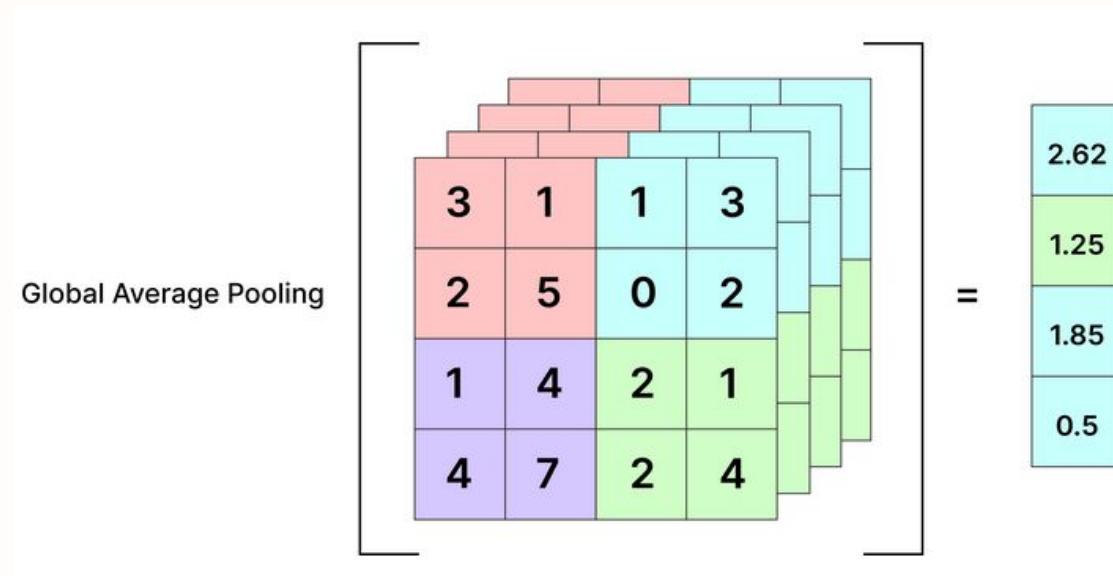
$$(N, C_{in}, H, W) * (C_{out}, C_{in}, K, K) \rightarrow (N, C_{out}, H, W)$$

NiN (Network in Network) - 1x1 Convolution

- Porque utilizar convoluções 1x1?
 - ◆ Podemos adicionar não-linearidade através da profundidade enquanto mantemos a característica espacial do sinal
 - ◆ Reduzir a profundidade do bloco de entrada pode melhorar a performance do modelo (e.g., menos operações vão ser realizadas)

NiN (Network in Network) - Global Average Pooling

- No modelo NiN é proposto o uso da camada Global Average Pooling, para substituir o uso de FC layers
 - ◆ Assim como outras camadas de pooling, Global Average Pooling atua reduzindo a dimensionalidade espacial do bloco de entrada
 - ◆ A redução é realizar até que cada dimensão tenha tamanho 1, preservando a profundidade do bloco (e.g., um bloco de entrada $10 \times 10 \times 64$ seria transformado para $1 \times 1 \times 64$)

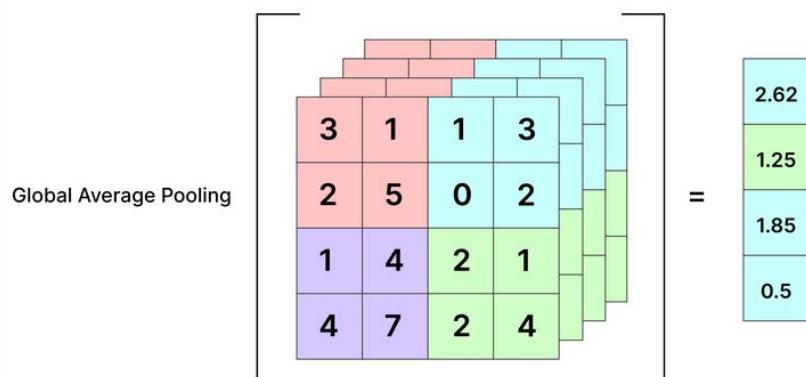


NiN (Network in Network) - Global Average Pooling

- No modelo NiN é proposto o uso da camada Global Average Pooling, para substituir o uso de FC layers
 - ◆ Assim como outras camadas de pooling, Global Average Pooling atua reduzindo a dimensionalidade espacial do bloco de entrada
 - ◆ A redução é realizar até que cada dimensão tenha tamanho 1, preservando a profundidade do bloco (e.g., um bloco de entrada $10 \times 10 \times 64$ seria transformado para $1 \times 1 \times 64$)

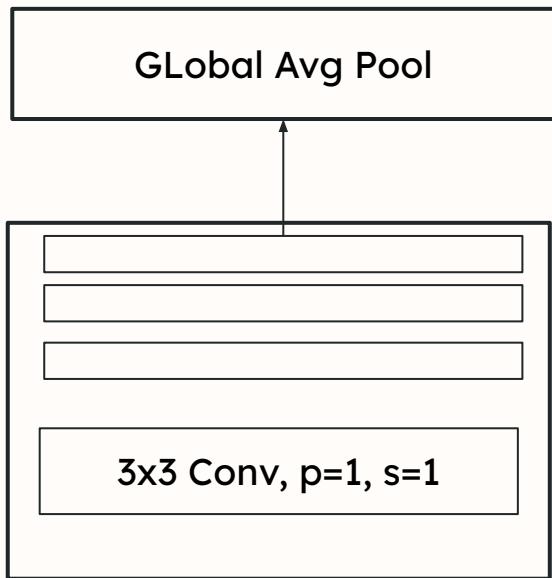
Quais as vantagens em utilizar Global Average Pooling?

1. Não há parâmetros para otimizar, evitando overfitting
2. A Global Average Pooling agrupa a informação espacial através da soma, sendo robusto a transformações de translação do sinal

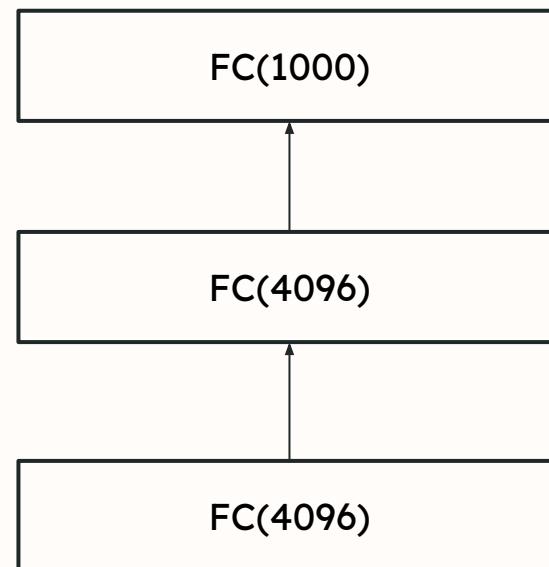


NiN (Network in Network)

NiN



AlexNet



NiN (Network in Network)

```
sequential1 output shape: (1, 64, 112, 112)
sequential2 output shape: (1, 128, 56, 56)
sequential3 output shape: (1, 256, 28, 28)
sequential4 output shape: (1, 512, 14, 14)
sequential5 output shape: (1, 512, 7, 7)
dense0 output shape: (1, 4096)
dropout0 output shape: (1, 4096)
dense1 output shape: (1, 4096)
dropout1 output shape: (1, 4096)
dense2 output shape: (1, 10)
```

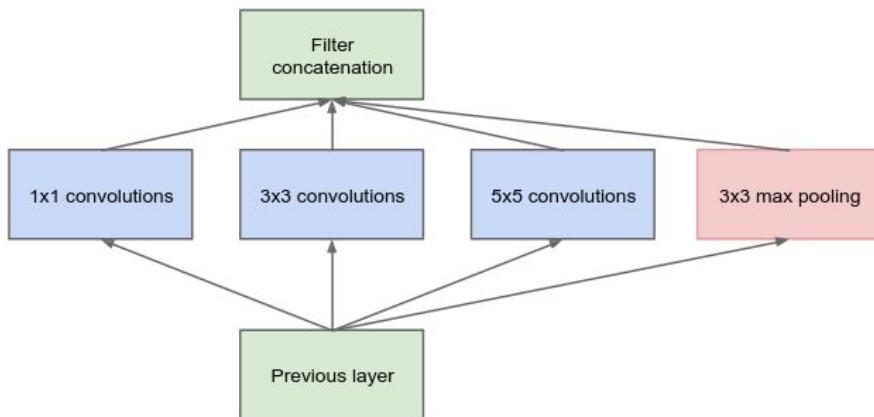
VGG

```
sequential1 output shape: (96, 54, 54)
pool0 output shape: (96, 26, 26)
sequential2 output shape: (256, 26, 26)
pool1 output shape: (256, 12, 12)
sequential3 output shape: (384, 12, 12)
pool2 output shape: (384, 5, 5)
dropout0 output shape: (384, 5, 5)
sequential4 output shape: (10, 5, 5)
pool3 output shape: (10, 1, 1)
flatten0 output shape: (10)
```

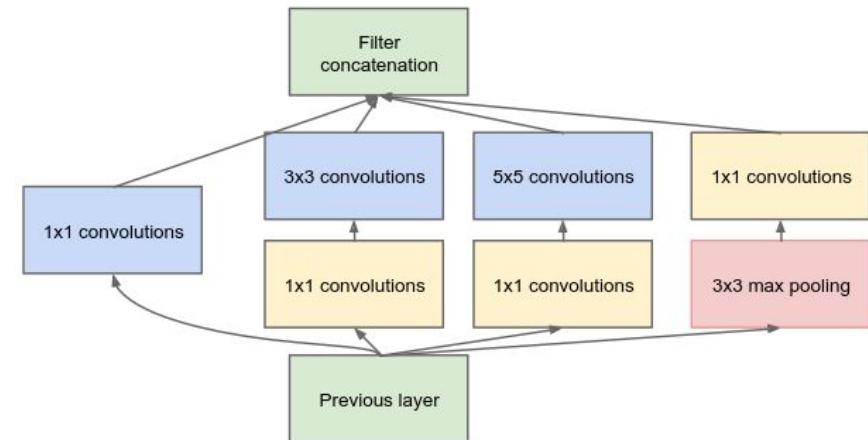
NiN

Inception (GoogleNet)

- Em 2015 a arquitetura Inception diversificou o design das redes profundas, combinando as principais características do NiN e VGG, misturando de convoluções com diferentes tamanhos de kernel



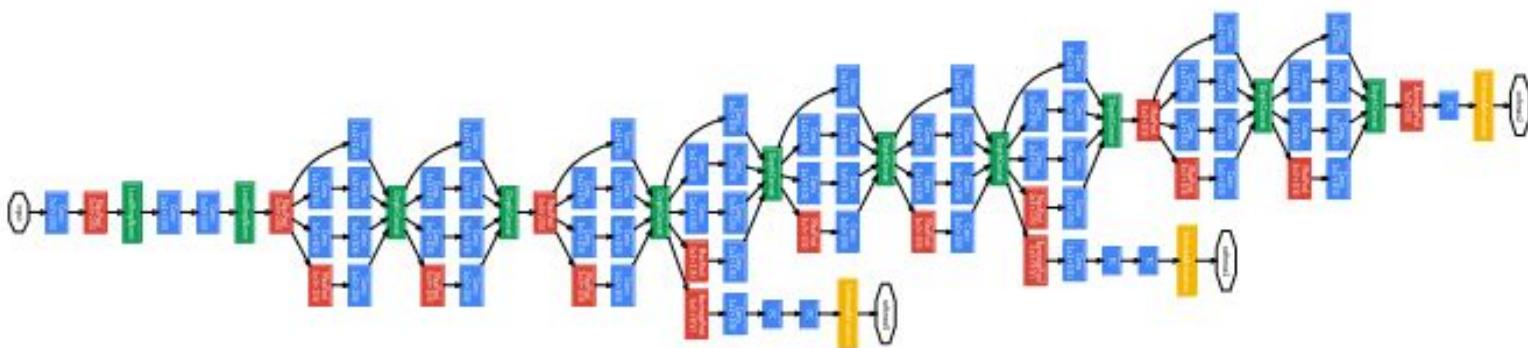
(a) Inception module, naïve version



(b) Inception module with dimension reductions

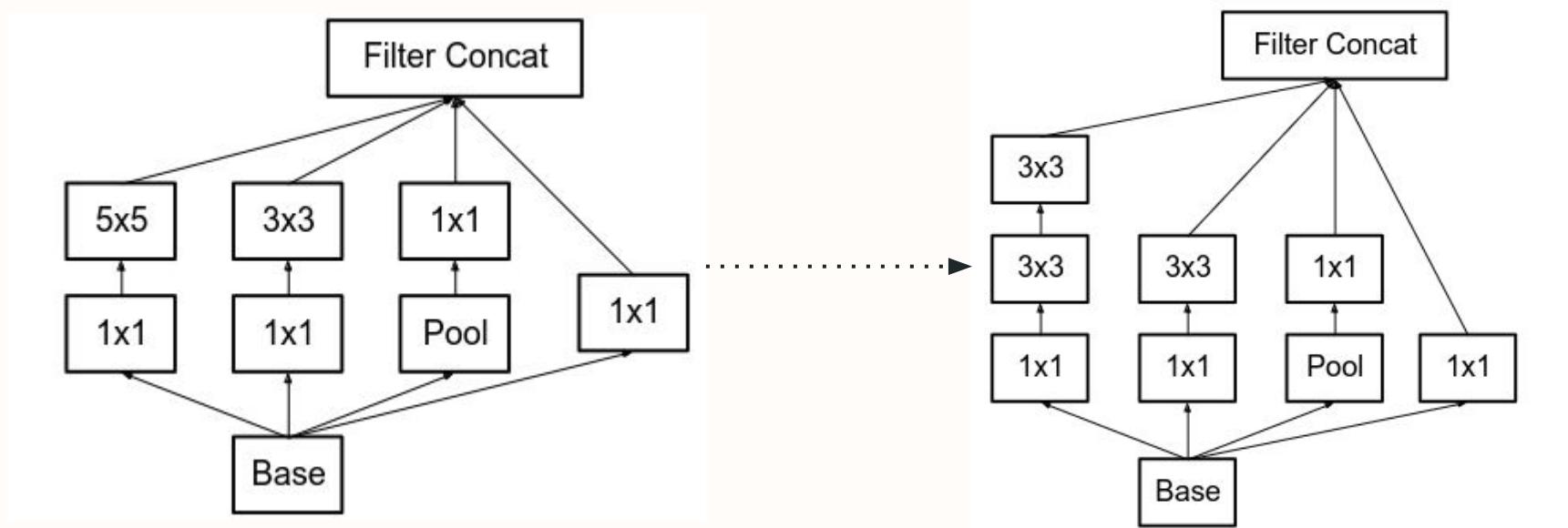
Inception (GoogleNet)

- A GoogleNet resolveu o problema na escolha do tamanho do kernel das convoluções utilizadas pelas arquiteturas
 - ◆ Combinando diferentes convoluções em múltiplas branches
 - ◆ Na arquitetura original, alguns ‘artifícios’ precisaram ser utilizados para estabilizar o treinamento com funções de perda intermediárias aplicadas em múltiplas camadas da rede (Com o avanço dos algoritmos de treinamento, isto não é mais necessário)



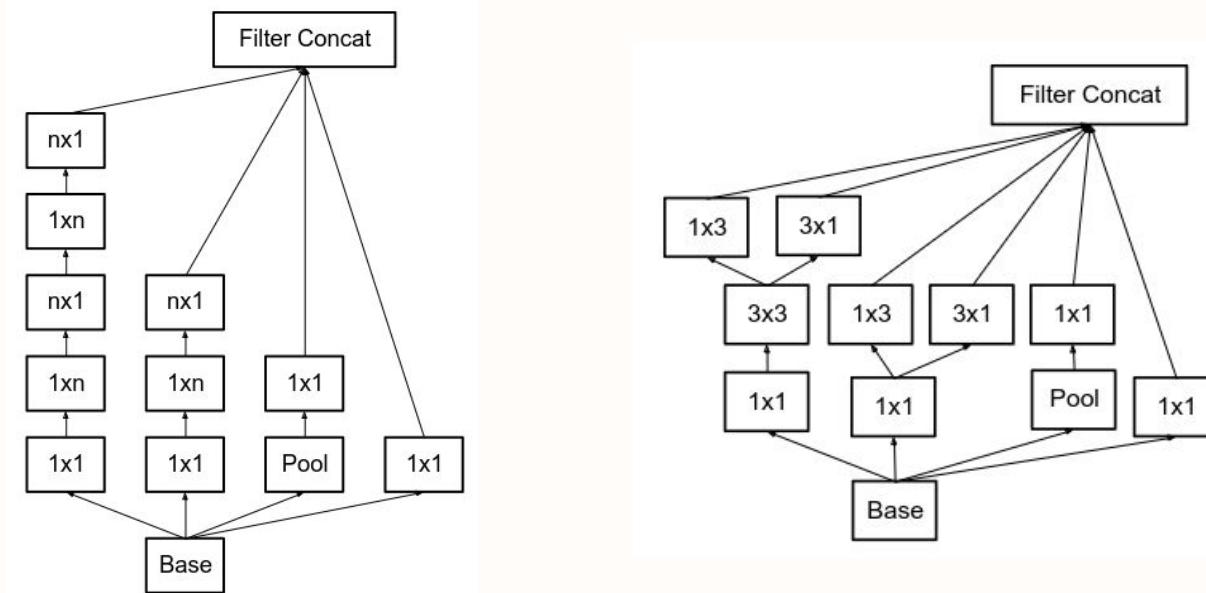
Inception V2

- Na Inception-V2 as convoluções 5x5 foram substituídas por convolução 3x3
 - ◆ Redução do custo computacional (i.e., ~redução de 2.78 na complexidade)



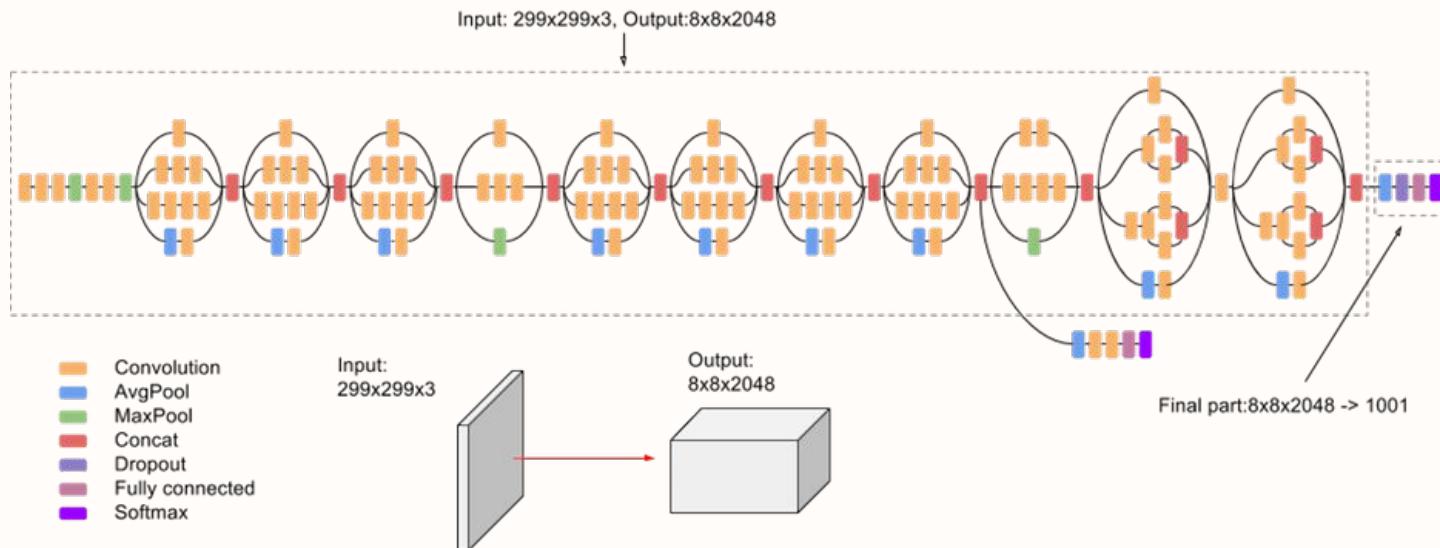
Inception-V2

- Algumas camadas $n \times n$ foram fatorizadas para $1 \times n$ e $n \times 1$
 - ◆ Dividir uma convolução 3×3 é equivalente (em termos do campo receptivo) a uma convolução 1×3 seguida de uma 3×1
 - ◆ As convoluções sequenciais podem trazer um ganho de 33% em relação a uma única convolução 3×3
 - ◆ A fatorização não funciona muito bem para as camadas iniciais (dimensão do bloco muito grande), mas é eficiente para camadas mais profundas



Inception-V3

- Utiliza as mesmas características da V2, adicionando:
 - ◆ Otimização com RMSprop
 - ◆ Batch Normalization
 - ◆ Convoluçãoes 7x7 fatorizadas
 - ◆ Label Smoothing Regularization; Regularização do classificador estimando o efeito do dropout durante o treinamento. Previne a predição de uma classe com muita confiança.



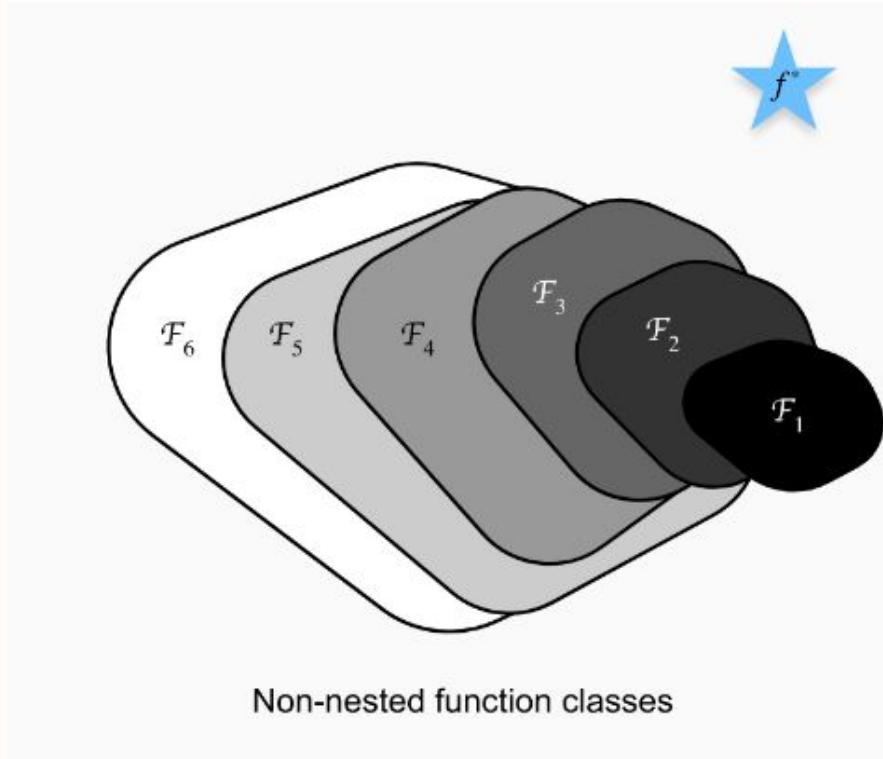
ResNet

- Na medida que mais camadas são adicionadas, aumentamos a complexidade das redes
- Redes profundas são mais difíceis de treinar
 - ◆ Considerando F como o conjunto de funções de uma arquitetura, sabemos que para todo f em F existe um conjunto de parâmetros que obtemos através do treinamento
 - ◆ Queremos encontrar o valor ótimo de parâmetros para as funções na arquitetura (f^*), mas nem sempre o valor final corresponde ao valor ótimo, e através da otimização tentamos encontrar a melhor aproximação:

$$\dot{f}_F = \arg \min_f L(X, y), f \in F$$

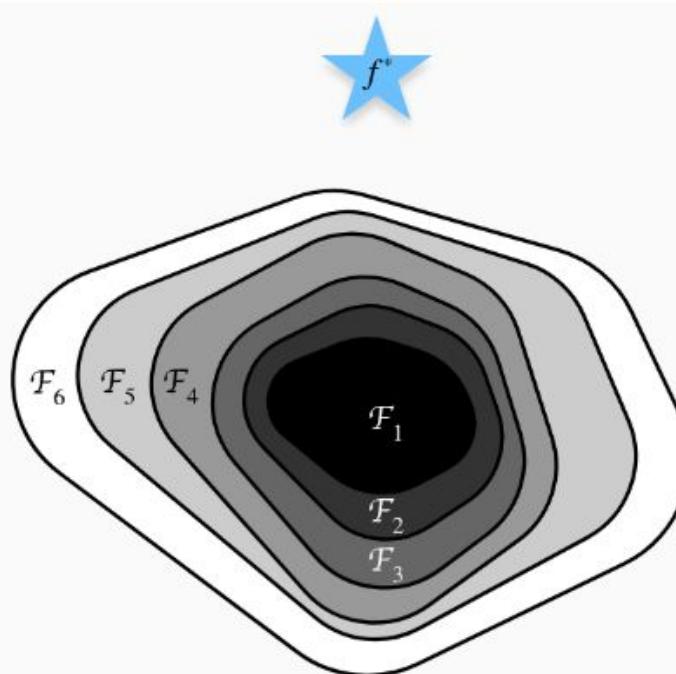
ResNet

- Podemos projetar uma arquitetura mais poderosa (F') para alcançar um resultado melhor. No entanto, se $F \not\subseteq F'$ não há garantias que isto acontecerá (podendo o resultado ser pior)



ResNet

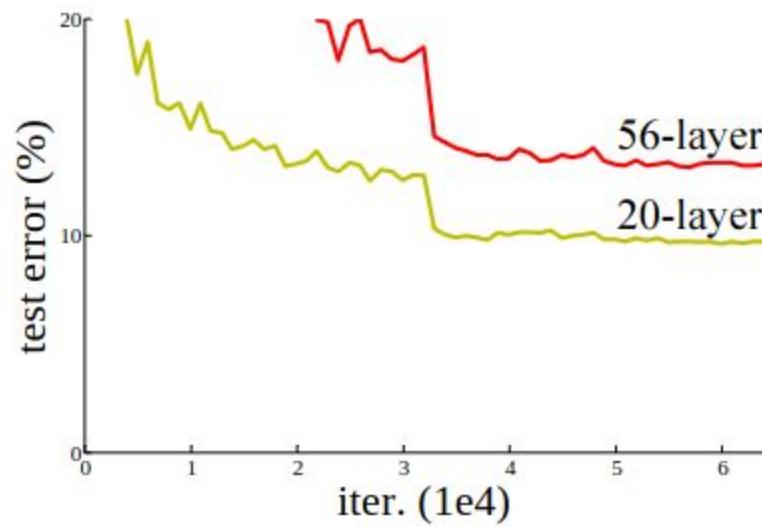
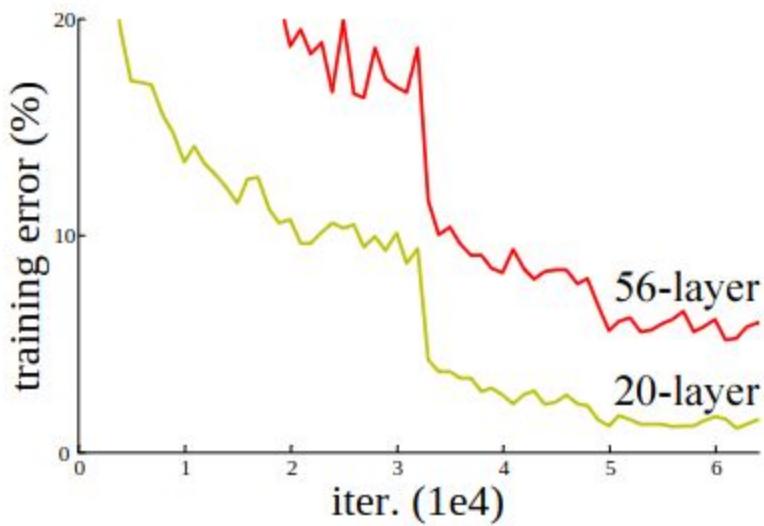
- Se agruparmos um novo espaço de funções, tal que $F_n \subseteq \dots \subseteq F_2 \subseteq F_1$, ainda não temos a garantia que vamos obter o valor ótimo (f^*). No entanto, garantimos que estamos aumentando o poder expressivo da rede
 - ◆ Se pudermos treinar uma nova camada em uma função identidade, o novo modelo será tão eficaz quanto o original. Obtendo uma solução melhor para se ajustar aos dados, a nova camada pode facilitar a reduzir o erro no treinamento



Fonte da Imagem: Dive Into Deep Learning. Cap 7 Modern Convolutional Neural Networks. 7.6 Residual Networks.
(https://pt.d2l.ai/chapter_convolutional-modern/resnet.html)

ResNet

- Com isto vemos que nem sempre empilhar diversas camadas pode trazer um bom resultado (vanishing gradients/explosão de gradientes)
 - ◆ Quando redes muito profundas começam a convergir, o aprendizado pode degradar e a acurácia saturar rapidamente
 - ◆ A degradação indica que nem todos os sistemas são igualmente fáceis de otimizar



ResNet

- Dada uma entrada x , queremos aprender o mapeamento $F(x)$
- Se passarmos x a frente na arquitetura, então para a camada atual precisamos encontrar o mapeamento residual $F(x) - x$
- Se $F(x) = x$, aprender o valor residual é mais fácil
 - ◆ Se o mapeamento de identidade for ideal, é mais fácil levar o resíduo para zero do que ajustar um mapeamento de identidade por através de uma sequência de camadas não lineares.

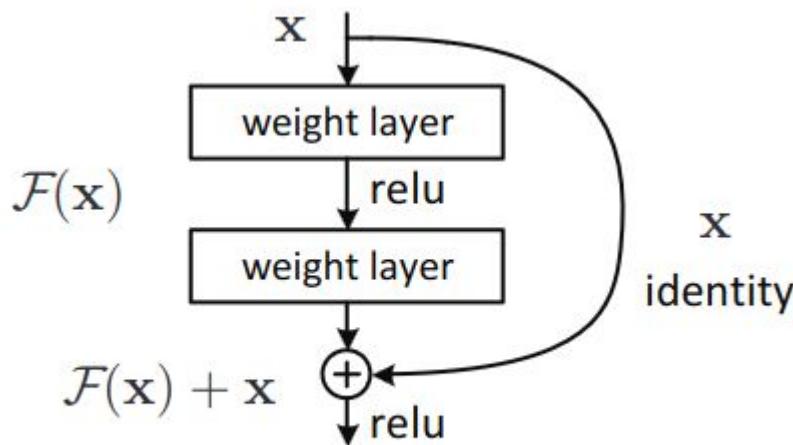


Figure 2. Residual learning: a building block.

ResNet

- Dada uma entrada x , queremos aprender o mapeamento $F(x)$
- Se passarmos x a frente na arquitetura, então para a camada atual precisamos encontrar o mapeamento residual $F(x) - x$
- Se $F(x) = x$, aprender o valor residual é mais fácil
 - ◆ Se o mapeamento de identidade for ideal, é mais fácil levar o resíduo para zero do que ajustar um mapeamento de identidade por através de uma sequência de camadas não lineares.

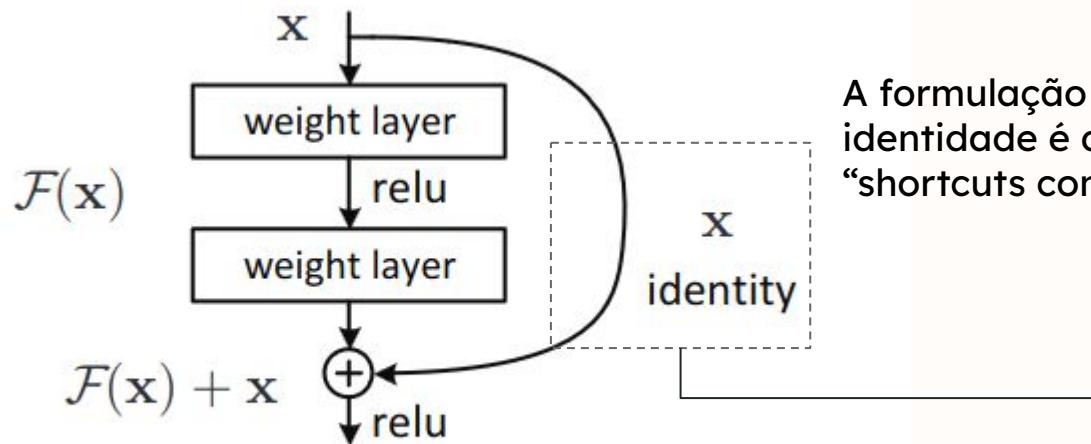
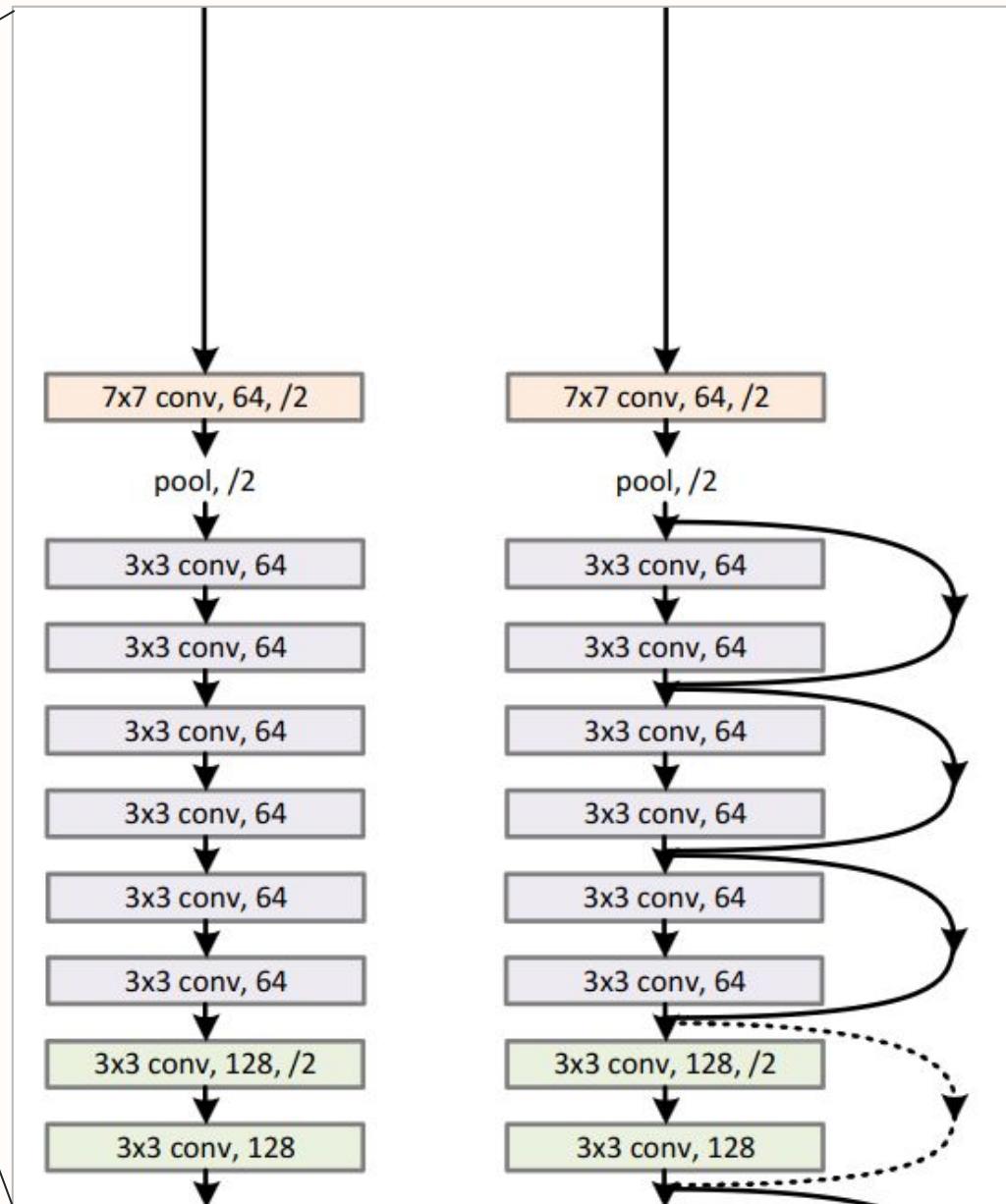
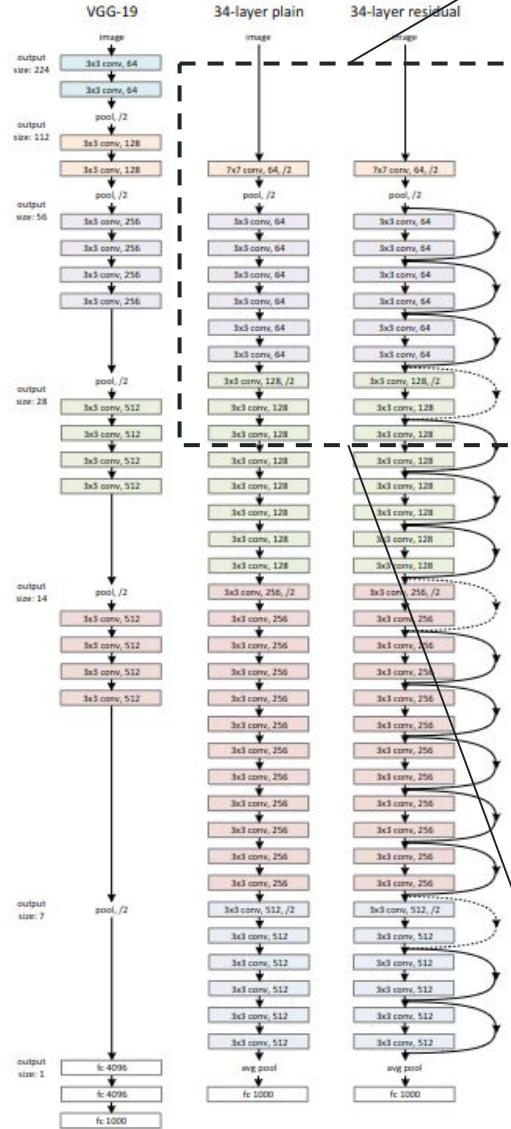


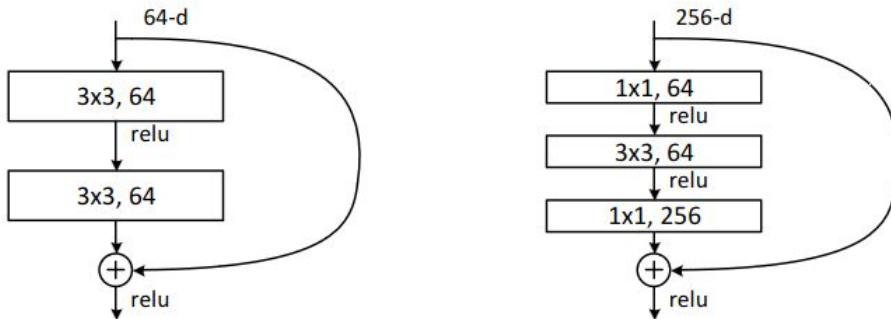
Figure 2. Residual learning: a building block.

ResNet



ResNet

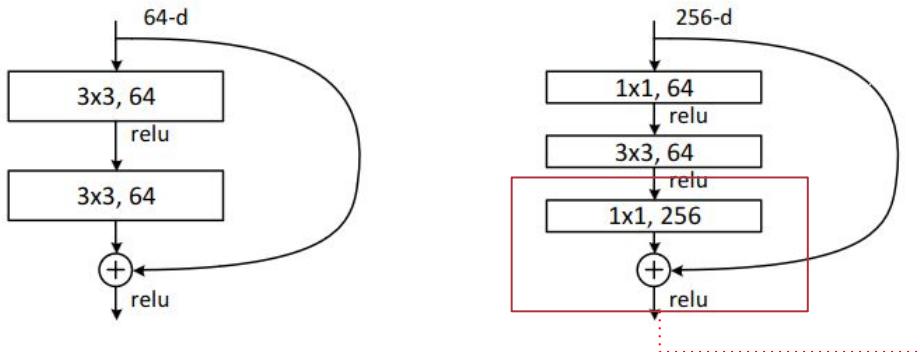
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



Na esquerda é utilizado um bloco com 56x56 mapas de ativações. Na direita temos o bloco de bottleneck para ResNet-50/101/152

ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

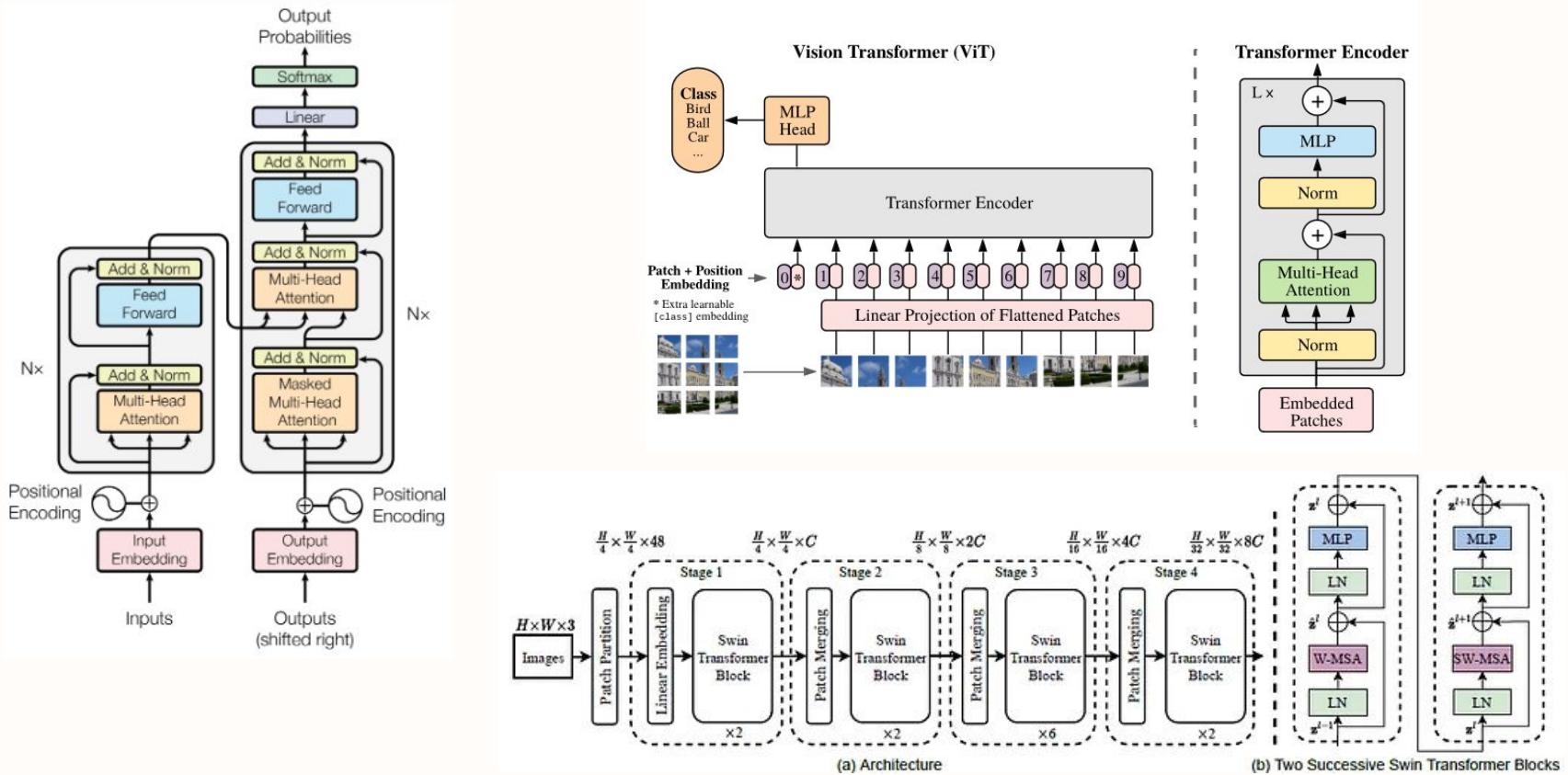


Na esquerda é utilizado um bloco com 56x56 mapas de ativações. Na direita temos o bloco de bottleneck para ResNet-50/101/152

Uma convolução 1x1 é aplicada como uma função de projeção para casar as dimensões de entrada e saída quando há mudança no número de canais

Vision Transformers

→ Attention is all you need



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

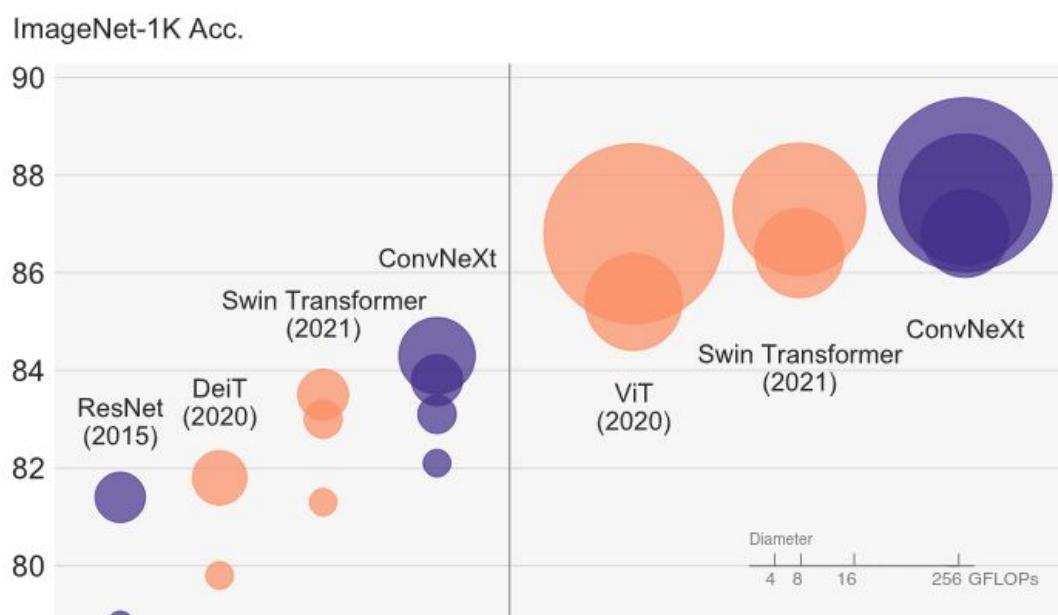
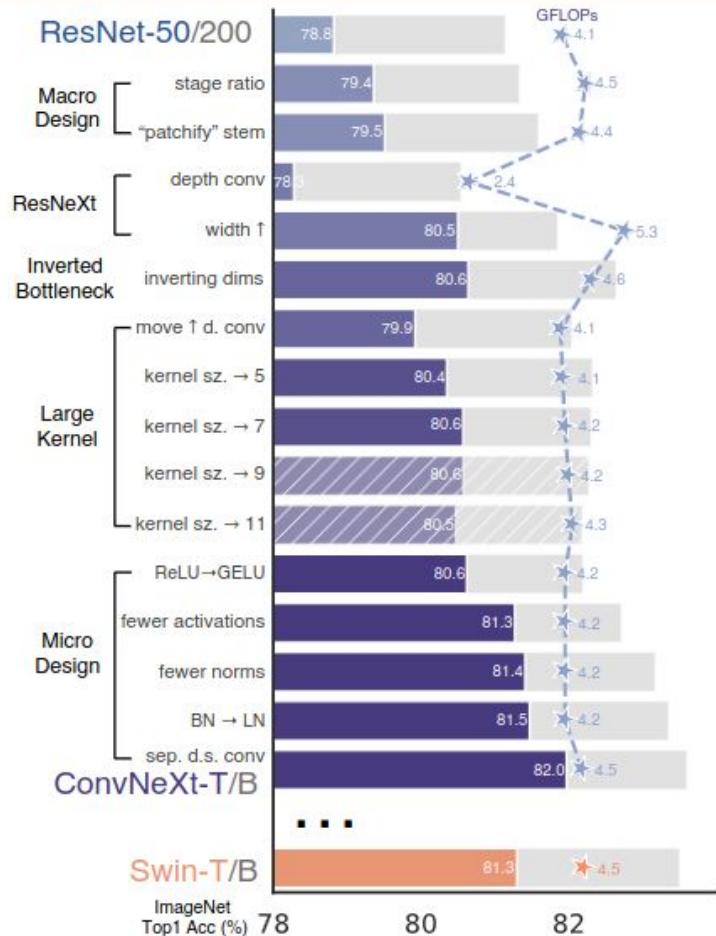
Liu, Ze, et al. "Swin transformer: Hierarchical vision transformer using shifted windows." *Proceedings of the IEEE/CVF international conference on computer vision*. 2021.

Vision Transformers

Method	#Param (M)	GFLOPs	Top-1 Acc (%)	Method	#Param (M)	GFLOPs	Top-1 Acc (%)
ResNet18 [67]*	11.7	1.8	69.8	ResNet101 [67] *	44.7	7.9	77.4
EfficientNet-B3 [87]*	12.0	1.8	81.6	ResNeXt101-32x4d [244]*	44.2	8.0	78.8
DeiT-T [12]	5.7	1.3	72.2	RegNetY-8G [86]*	39.0	8.0	81.7
T2T-ViT-T-7 [35]	5.0	1.3	71.7	EfficientNet-B5 [87] *	30.0	9.9	83.6
LocalViT-T [107]	5.9	1.3	74.8	CvT-21 [96]	32.0	7.1	82.5
CrossViT-T [104]	6.9	1.6	73.4	CaiT-S-24 [243]	32.2	9.4	82.7
PVTv1-T [93]	13.2	1.9	75.1	T2T-ViT-T-19 [35]	39.0	9.8	81.4
ResT-Lite [110]	10.5	1.4	77.2	PVTv1-M [93]	44.2	6.7	81.2
CaiT-XXX-24 [243]	12.0	2.5	77.6	PVTv2-B3 [97]	45.2	6.9	83.2
PVTv2-B1 [97]	13.1	2.1	78.7	NesT-S [111]	38.0	10.4	83.3
Lv-ViT-T [89]	8.5	—	79.1	ResNet152 [67] *	60.2	11.6	78.3
RegionViT-T [100]	13.8	2.4	80.4	CaiT-S-36 [243]	48.0	13.9	83.3
ResNet50 [67]*	25.6	4.1	76.1	T2T-ViT-T-24 [35]	64.0	15.0	82.2
ResNeXt50-32x4d [244]*	25.0	4.3	77.6	PVTv1-L [93]	61.4	9.8	81.7
RegNetY-4G [86]*	21.0	4.0	80.0	TNT-B [88]	66.0	14.1	82.8
EfficientNet-B4 [87]*	19.0	4.2	82.9	Swin-S [36]	50.0	8.7	83.0
DeiT-S [12]	22.1	4.6	79.9	Twins-SVT-B [37]	56.0	8.3	83.2
PVTv1-S [93]	24.5	3.8	79.8	RegionViT-B [100]	72.7	13.0	83.3
LocalViT-S [107]	22.4	4.6	80.8	PVTv2-B4 [97]	62.6	10.1	83.6
CrossViT-S [104]	26.7	5.6	81.0	ResNeXt101-64x4d [244] *	83.5	15.6	79.6
TNT-S [88]	23.8	5.2	81.3	RegNetY-16G [86] *	84.0	16.0	82.9
Swin-T [36]	29.0	4.5	81.3	EfficientNet-B6 [87] *	43.0	19.0	84.0
NesT-T [111]	17.0	5.8	81.5	NesT-B [111]	68.0	17.9	83.8
T2T-ViT-T-14 [35]	21.5	5.2	81.5	ViT-B/16 [11]	86.6	17.6	79.8
CvT-13 [96]	20.0	4.5	81.6	DeiT-B/16 [12]	86.6	17.6	81.8
ResT-B [110]	30.3	4.3	81.6	Swin-B [36]	88.0	15.4	83.3
Twins-SVT-S [37]	24.0	2.8	81.7	Twins-SVT-L [37]	99.2	14.8	83.7
PVTv2-B2-Li [97]	22.6	3.9	82.1	PVTv2-B5 [97]	82.0	11.8	83.8
RegionViT-S [100]	30.6	5.6	82.5	Lv-ViT-M [89]	56.0	16.0	84.1
Lv-ViT-S [89]	26.0	6.6	83.3				

ConvNeXt

→ A ConvNet for the 2020s



ConvNeXt - Inverted Bottleneck

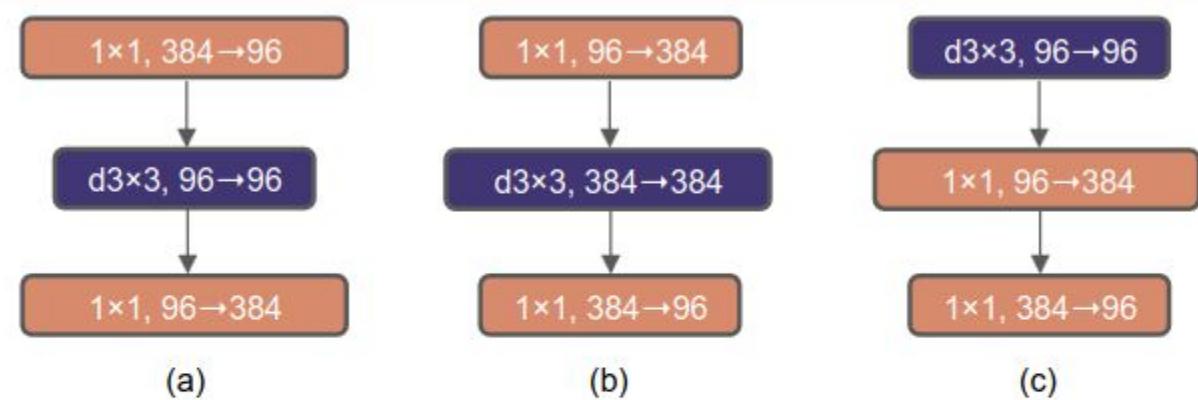
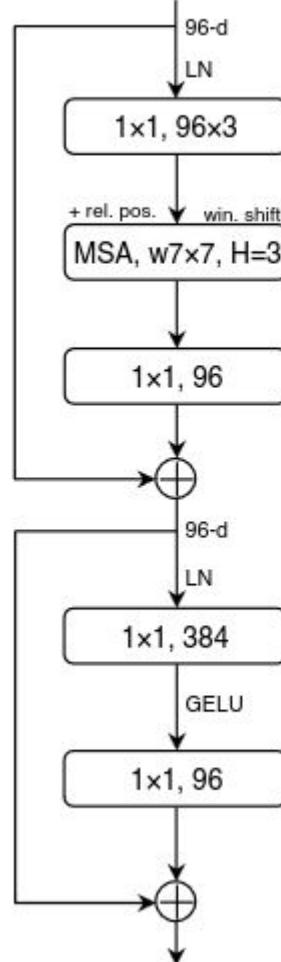


Figure 3. **Block modifications and resulted specifications.** (a) is a ResNeXt block; in (b) we create an inverted bottleneck block and in (c) the position of the spatial depthwise conv layer is moved up.

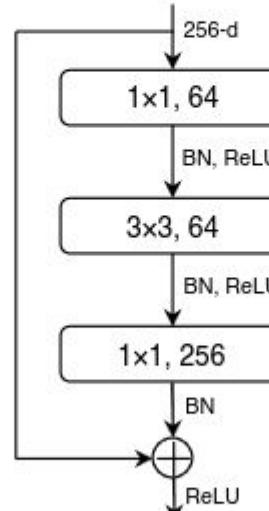
ConvNeXt

- Depth-wise convolutions
- Aumento do tamanho do kernel
- Substituindo ReLU pela GELU
- Redução no uso de camadas de normalização
- Substituição do batch normalization por layer normalization

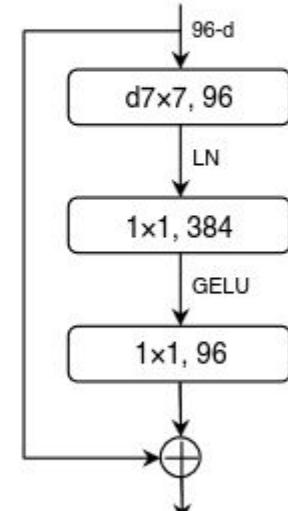
Swin Transformer Block



ResNet Block



ConvNeXt Block



ConvNeXt

ImageNet-1K trained models ↗

name	resolution	acc@1	#params	FLOPs	model
ConvNeXt-T	224x224	82.1	28M	4.5G	model
ConvNeXt-S	224x224	83.1	50M	8.7G	model
ConvNeXt-B	224x224	83.8	89M	15.4G	model
ConvNeXt-B	384x384	85.1	89M	45.0G	model
ConvNeXt-L	224x224	84.3	198M	34.4G	model
ConvNeXt-L	384x384	85.5	198M	101.0G	model

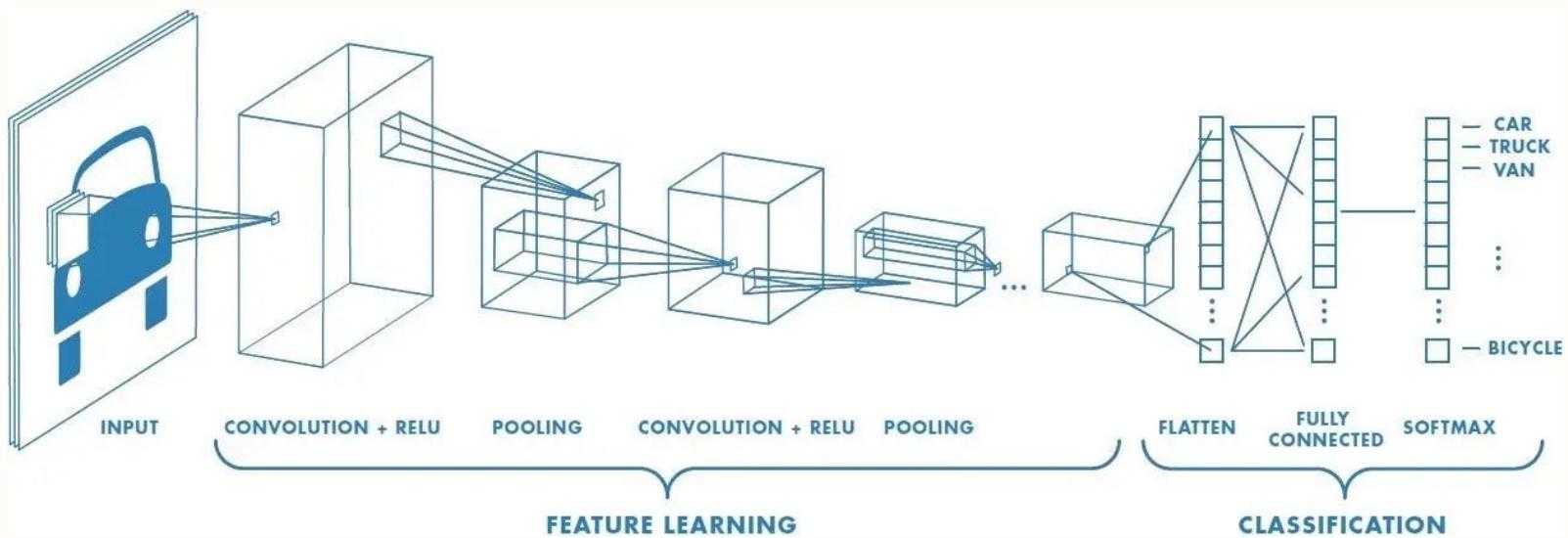
ImageNet-22K trained models ↗

name	resolution	acc@1	#params	FLOPs	22k model	1k model
ConvNeXt-T	224x224	82.9	29M	4.5G	model	model
ConvNeXt-T	384x384	84.1	29M	13.1G	-	model
ConvNeXt-S	224x224	84.6	50M	8.7G	model	model
ConvNeXt-S	384x384	85.8	50M	25.5G	-	model
ConvNeXt-B	224x224	85.8	89M	15.4G	model	model
ConvNeXt-B	384x384	86.8	89M	47.0G	-	model
ConvNeXt-L	224x224	86.6	198M	34.4G	model	model
ConvNeXt-L	384x384	87.5	198M	101.0G	-	model
ConvNeXt-XL	224x224	87.0	350M	60.9G	model	model
ConvNeXt-XL	384x384	87.8	350M	179.0G	-	model

Transfer Learning

Transfer Learning - Introdução

- As camadas convolucionais das CNNs podem ser visualizadas como extratores de características, aprendendo features relevantes em cada conjunto para discriminar os exemplos
 - ◆ As camadas finais aprendem a interpretar estas características gerando a resposta desejada



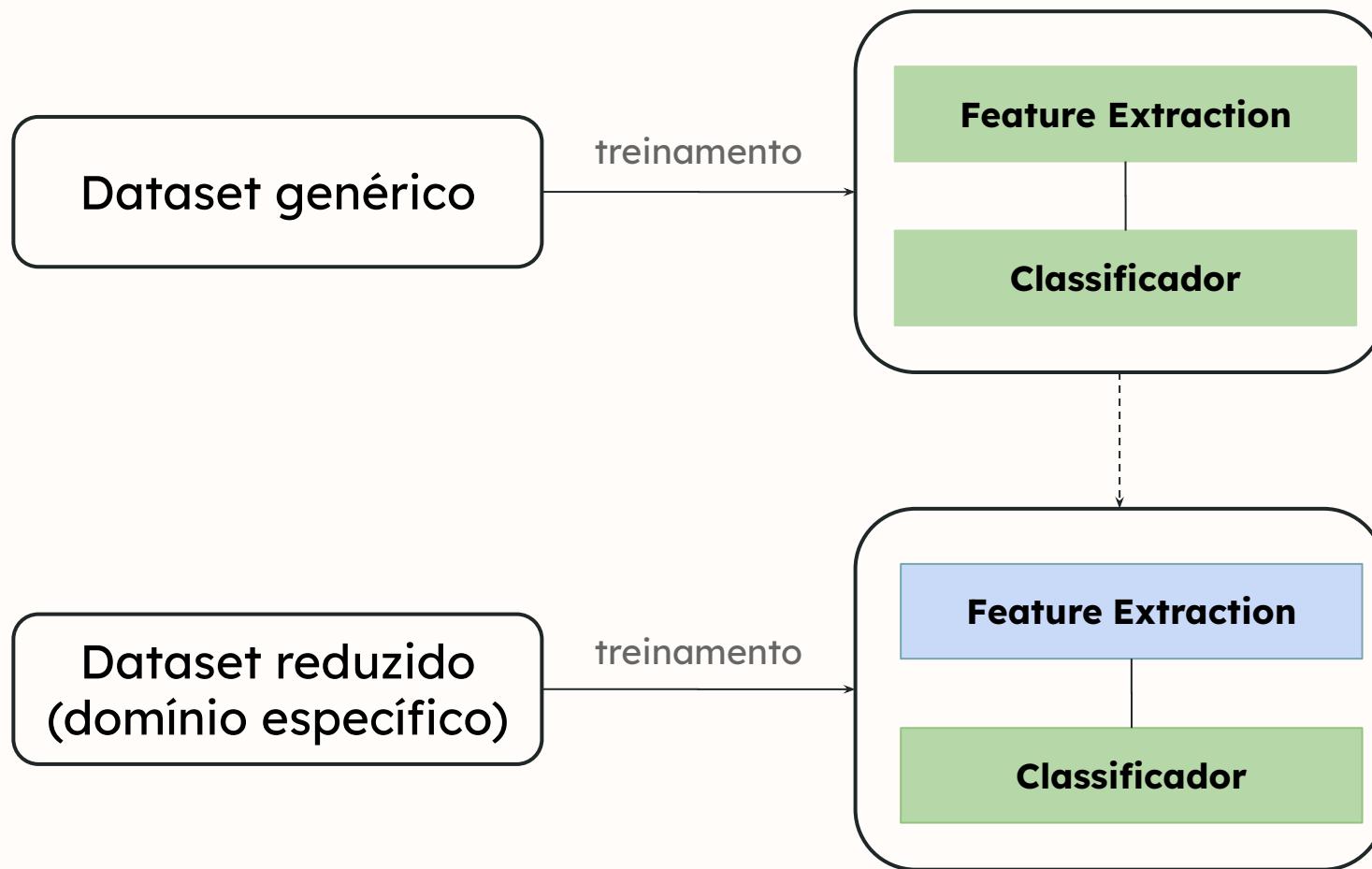
Transfer Learning - Introdução

- Para treinar estes modelos muitas imagens são necessárias para gerar uma representação discriminativa o suficiente para se obter um bom resultado
- Geralmente, para aplicações em domínio específico, não temos um conjunto de dados grande o suficiente, ou que represente com fidelidade a distribuição de teste que desejamos

Transfer Learning - Introdução

- Para treinar estes modelos muitas imagens são necessárias para gerar uma representação discriminativa o suficiente para se obter um bom resultado
- Geralmente, para aplicações em domínio específico, não temos um conjunto de dados grande o suficiente, ou que represente com fidelidade a distribuição de teste que desejamos
 - ◆ E se aproveitarmos as características já aprendidas pelo modelo para resolver uma tarefa anterior?

Transfer Learning



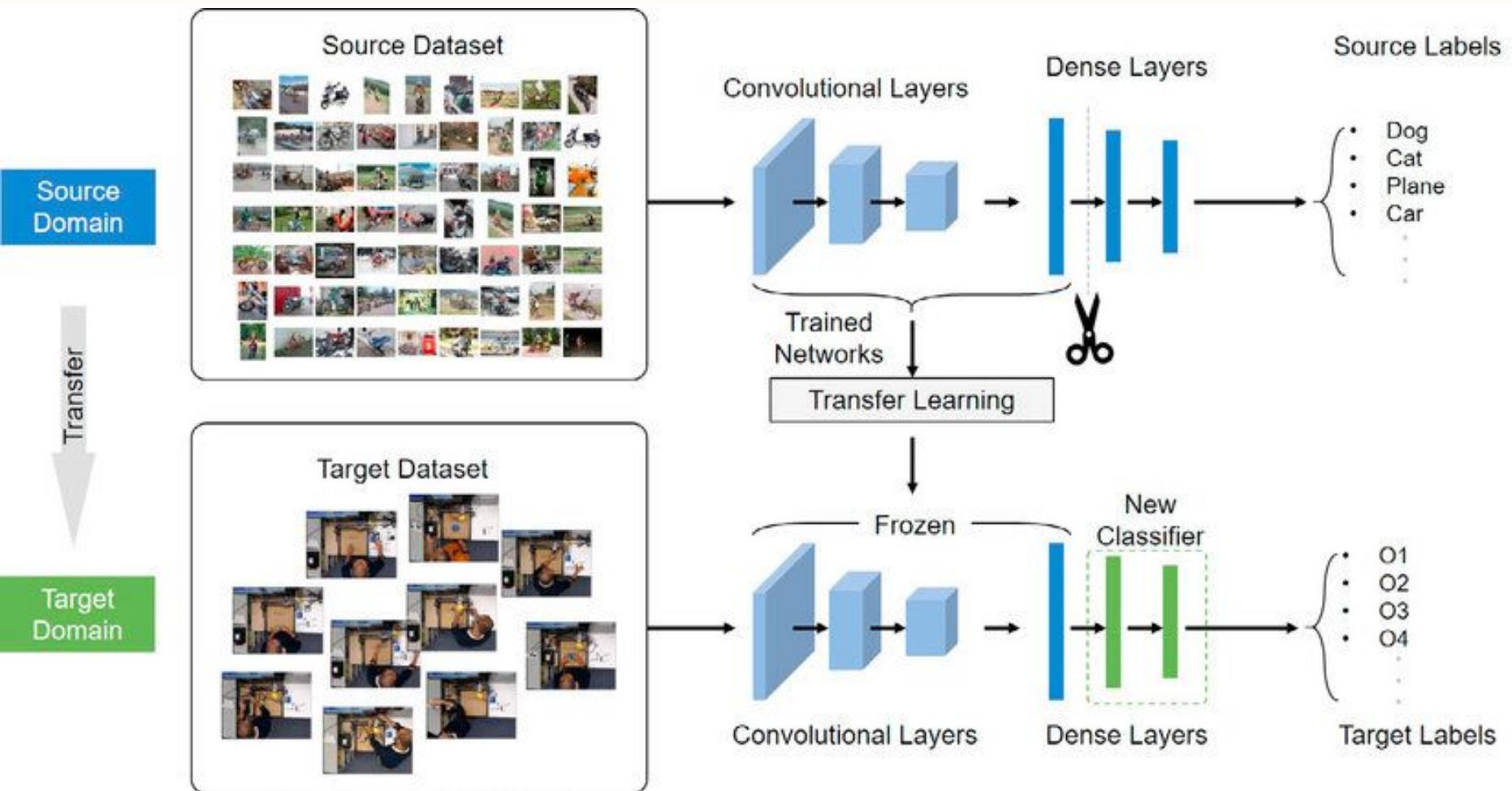
Pesos congelados

Parâmetros Treináveis

Transfer Learning

- Aproveitamos as features extraídas a partir de um treinamento com dataset maior e aplicamos este modelo em um novo treinamento, limitando o conjunto de parâmetros que são atualizados
 - ◆ Aproveitamos as camadas iniciais que extraem features de baixo nível para classificar os novos exemplos
 - ◆ Precisamos treinar as camadas de classificação para ajustar as previsões a partir do novo conjunto de ativação gerado

Transfer Learning

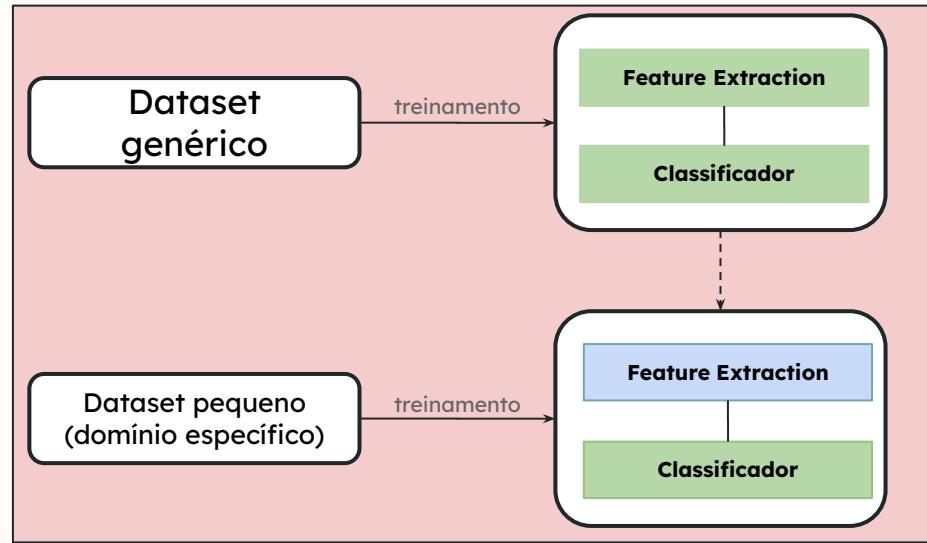


Tao, Wenjin, et al. "Real-time assembly operation recognition with fog computing and transfer learning for human-centered intelligent manufacturing." *Procedia Manufacturing* 48 (2020): 926-931.

Transfer Learning

- Transfer Learning é uma estratégia que nos permite aproveitar o conhecimento aprendido por um modelo que será utilizado em uma nova tarefa
 - ◆ Geralmente, as features iniciais são compartilhadas entre diferentes problemas por descrevem características de alto nível (bordas verticais, bordas horizontais, cores, etc)
 - ◆ Nesse contexto, conjunto de dados que são relacionados tendem a compartilhar mais características, melhorando o resultado do transfer-learning
 - ◆ O transfer learning ajuda a diminuir o custo e tempo de treinamento
 - ◆ Podemos realizar o transfer-learning de diferentes formas, dependendo do tipo do modelo, tamanho do dataset, tipo do dataset, etc.

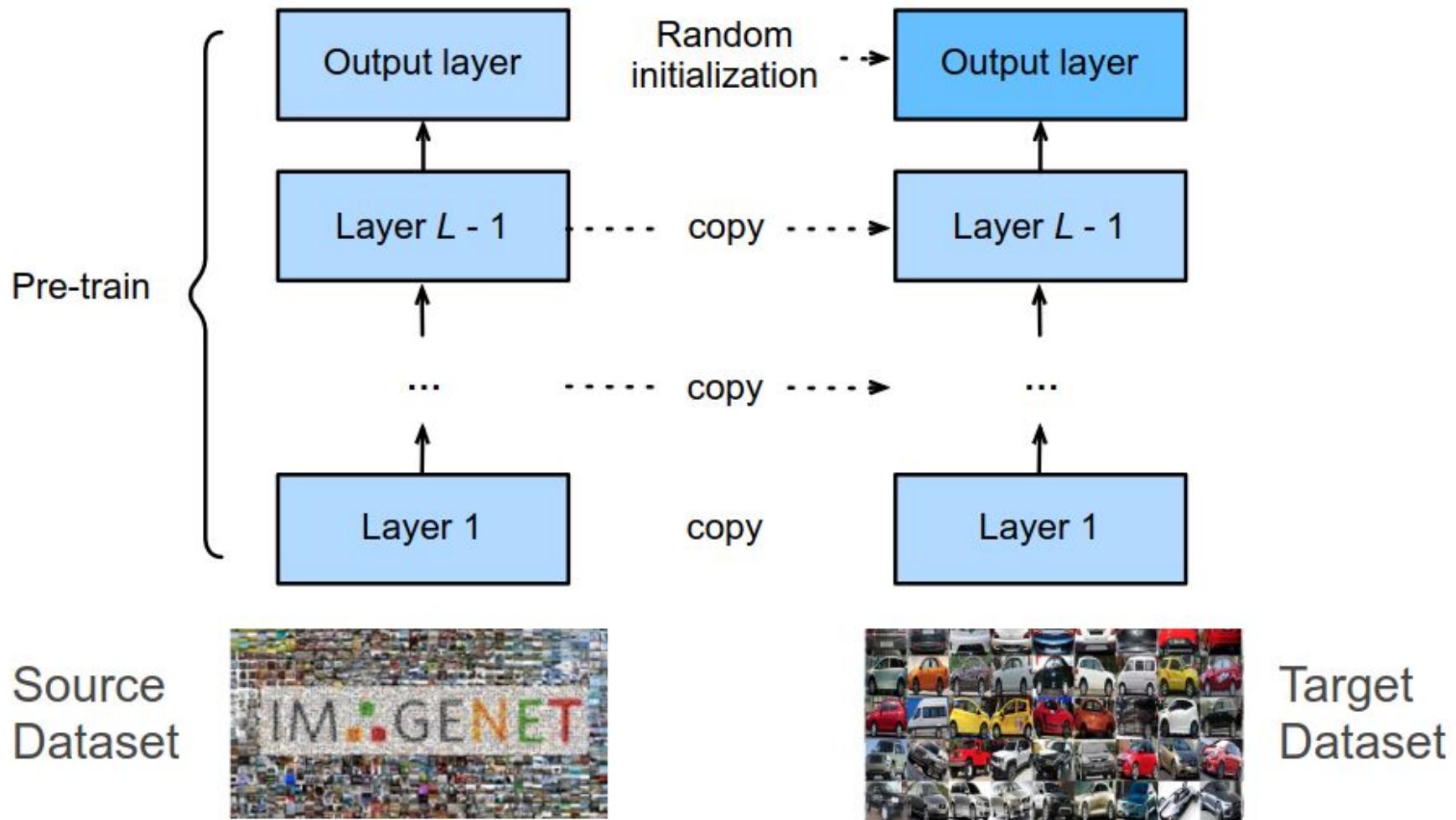
Transfer Learning



Pesos congelados

Parâmetros Treináveis

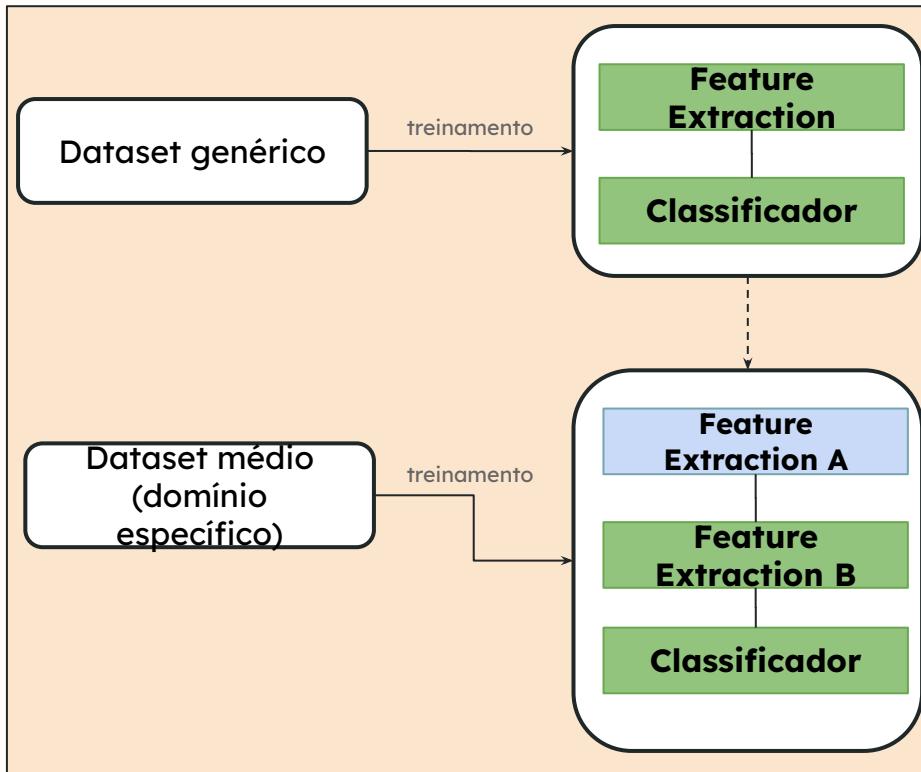
Transfer Learning



Transfer Learning

- Podemos re-usar os parâmetros do classificador na inicialização
 - ◆ Idealmente quando as categorias dos datasets (source e target) são as mesmas
- Podemos também fixar algumas camadas convolucionais e atualizar um sub-conjunto para ajustar as features extraídas de acordo com o novo dataset

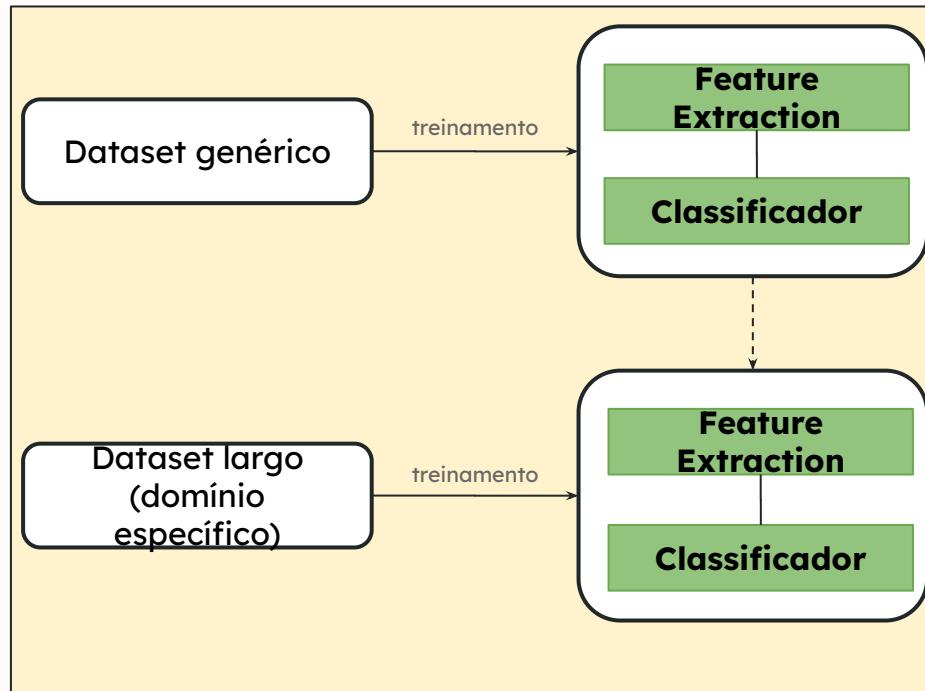
Transfer Learning



Pesos congelados

Parâmetros Treináveis

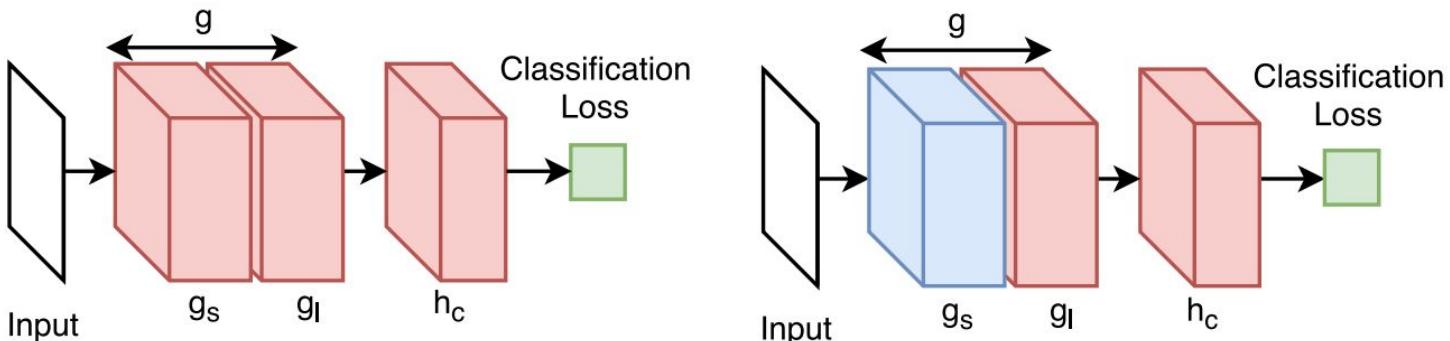
Transfer Learning



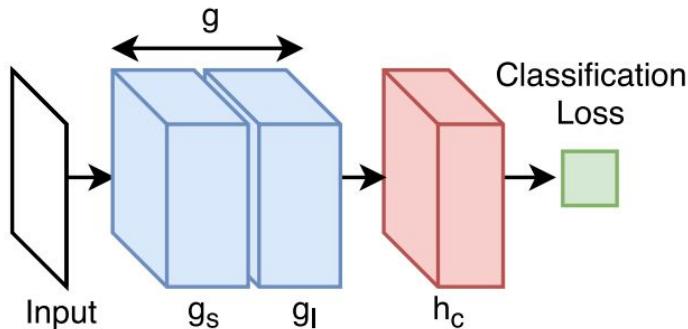
Pesos congelados

Parâmetros Treináveis

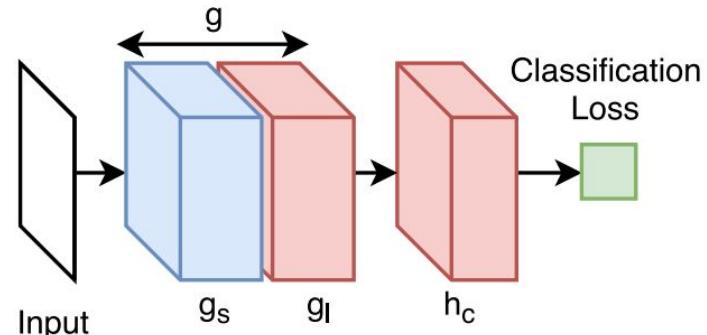
Transfer Learning



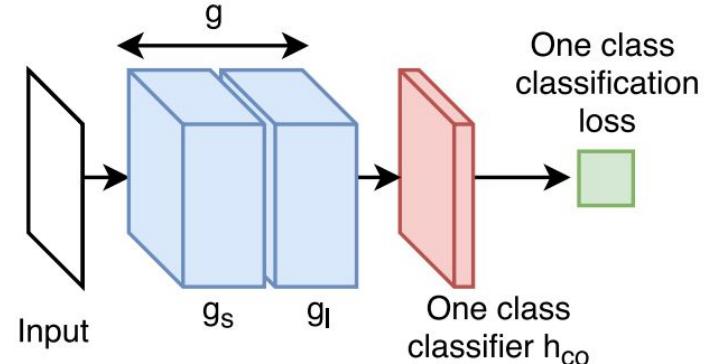
(a) **Deep Learning** : multiple Classes, Many images



(b) **Fine Tuning** : multiple classes, few images



(c) **Fine Tuning** : multiple classes, medium number of images

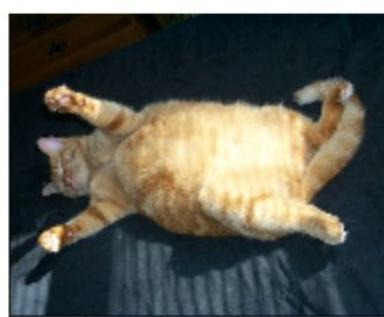


(d) Single class classification with deep features

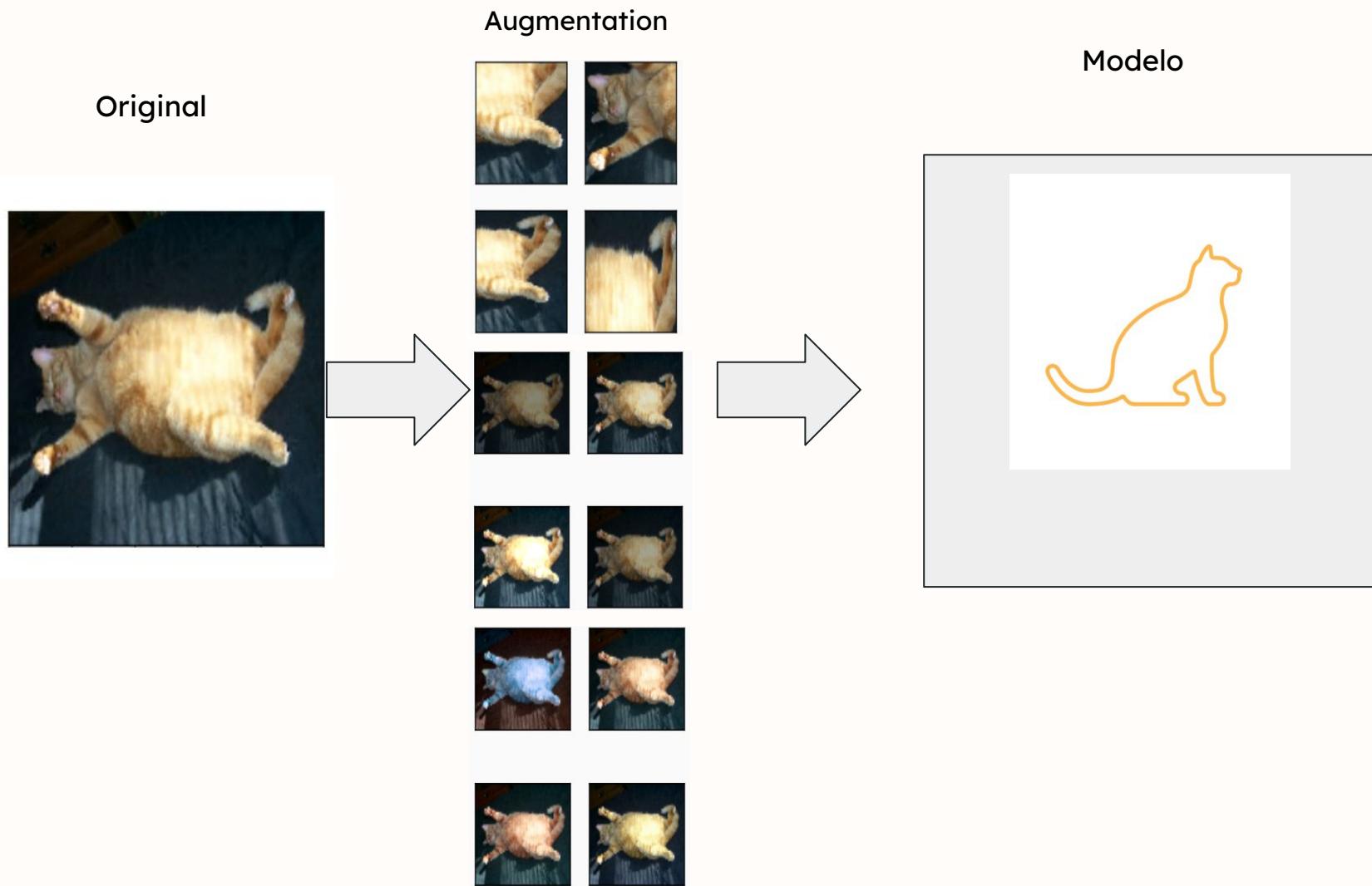
Augmentation

Introdução

- Como visto anteriormente, podemos ter limitações quanto às imagens obtidas para treinamento dos modelos, por exemplo, podemos ter um dataset faltando exemplos com variações para iluminação, rotação, colorização etc.
- Nesse sentido, a utilização de *Data Augmentation* tem como objetivo aumentar a quantidade de imagens para treinamento além de disponibilizar formatos diferentes e ruídos aos dados.

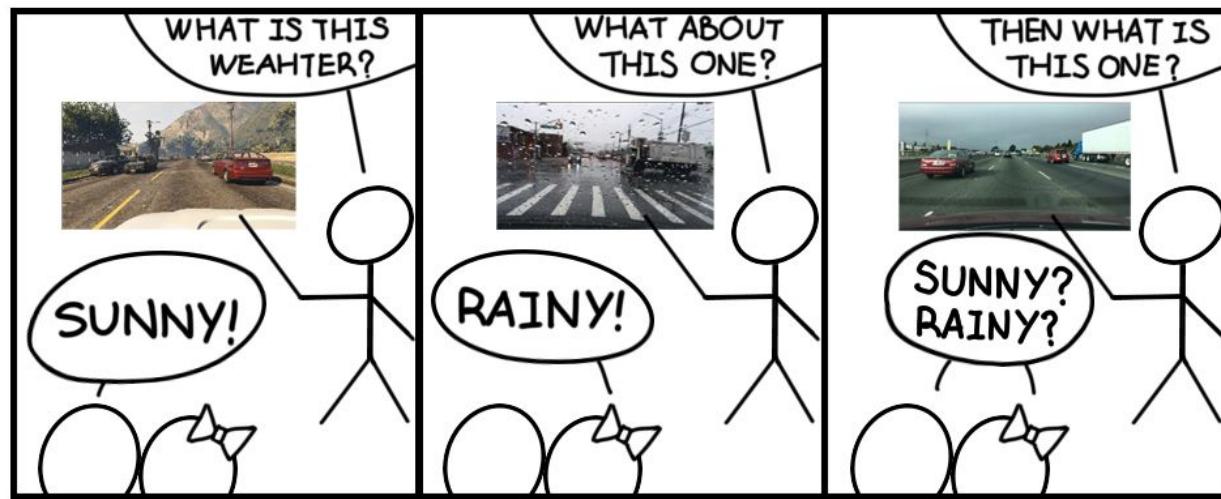


Treinamento com *Data Augmentation*



Treinamento com Data augmentation

- Quando podemos utilizar o data augmentation?
 - ◆ Quando o conjunto de treinamento é pequeno, o que pode causar overfitting no modelo
 - ◆ Quando o conjunto de treinamento não mapeia adequadamente ao conjunto alvo (teste)
 - ◆ Aumentar a robustez para variações nos dados causados por ruídos ou ‘forçar’ o aprendizado de características relevantes



Treinamento com Data augmentation

- Para aplicar data augmentation efetivamente, devemos entender quais são as transformações que podem ocorrer e fazem sentido serem aplicadas no domínio que estamos avaliando
 - ◆ Exemplo: Vertical flip pode não ser interessante para treinar um modelo que identifica carros a partir de uma câmera acoplada em outro automóvel

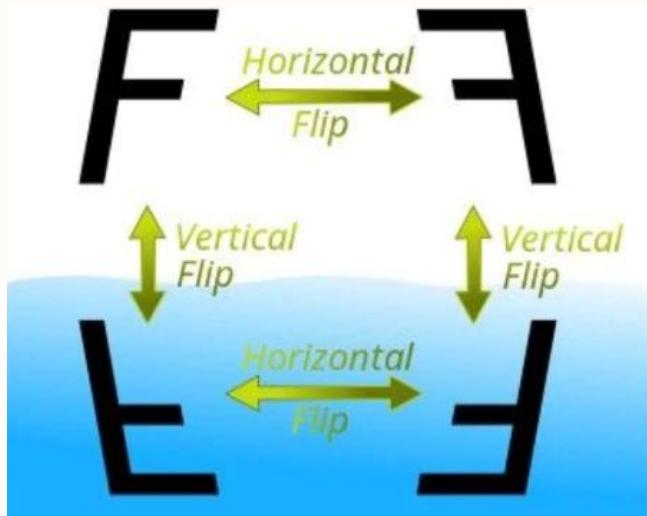
Blur

- Multiplique o valor de cada pixel vizinho pelo valor do Kernel correspondente;
- Some os resultados;
- O número resultante é o valor do pixel transformado;
- Repita o processo até que toda a imagem seja coberta.

$$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \otimes \begin{bmatrix} 0.0 & 0.5 & 0.0 \\ 0.5 & 0.5 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{bmatrix} = [0.5]$$



Flip



→ Faz o espelhamento da imagem em relação ao eixo horizontal ou vertical



Color Jitter



Brightness



Hue



Crop

→ Aplica um recorte da imagem, no qual aproxima determinadas partes da imagem para interesse.



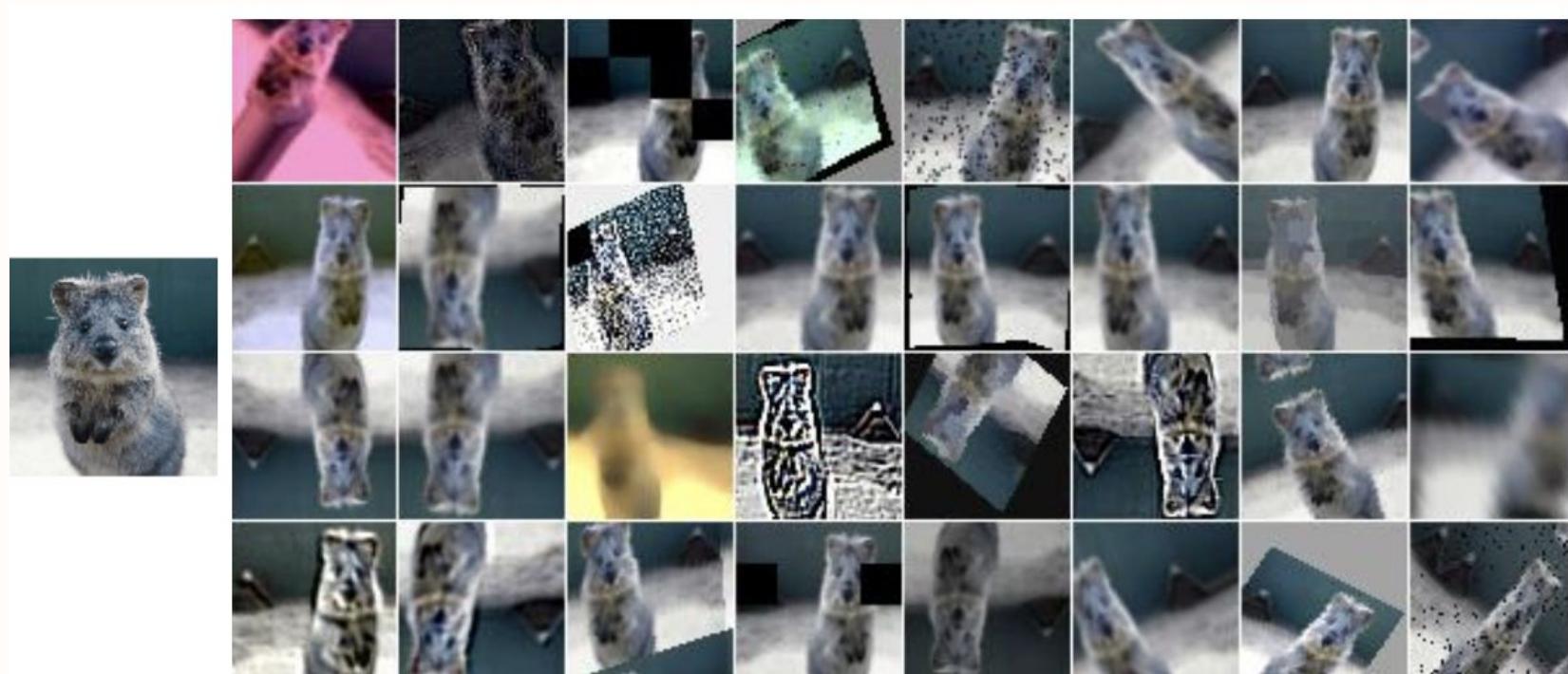
Rotation

→ Aplica rotações na imagem em diferentes graus.



Augmentations - Pipeline

- Augmentations são aplicados aleatoriamente em cada batch
- Cada nova época altera os exemplos de uma forma diferente
- Escolhemos as operações realizadas de acordo com o problema que estamos avaliando



Data Augmentation - PyTorch

- Operações de augmentation já são implementadas pelos frameworks de deep learning / machine learning
 - ◆ Ex. PyTorch: Transforms

Geometry	Cropping	Others	Color
<p>Resizing</p> <pre>v2.Resize(size[, interpolation, max_size, ...])</pre> <pre>v2.ScaleJitter(target_size[, scale_range, ...])</pre> <pre>v2.RandomShortestSize(min_size[, max_size, ...])</pre> <pre>v2.RandomResize(min_size, max_size[, ...])</pre> <p>Functionals</p> <pre>v2.functional.resize(inpt, size[, ...])</pre>	<p>Cropping</p> <pre>v2.RandomCrop(size[, padding, ...])</pre> <pre>v2.RandomResizedCrop(size[, scale, ratio, ...])</pre> <pre>v2.RandomIoUCrop([min_scale, max_scale, ...])</pre> <pre>v2.CenterCrop(size)</pre> <pre>v2.FiveCrop(size)</pre> <pre>v2.TenCrop(size[, vertical_flip])</pre>	<pre>v2.RandomHorizontalFlip([p])</pre> <pre>v2.RandomVerticalFlip([p])</pre> <pre>v2.Pad(padding[, fill, padding_mode])</pre> <pre>v2.RandomZoomOut([fill, side_range, p])</pre> <pre>v2.RandomRotation(degrees[, interpolation, ...])</pre> <pre>v2.RandomAffine(degrees[, translate, scale, ...])</pre> <pre>v2.RandomPerspective([distortion_scale, p, ...])</pre> <pre>v2.ElasticTransform([alpha, sigma, ...])</pre>	<p>Color</p> <pre>v2.ColorJitter([brightness, contrast, ...])</pre> <pre>v2.RandomChannelPermutation()</pre> <pre>v2.RandomPhotometricDistort([brightness, ...])</pre> <pre>v2.Grayscale([num_output_channels])</pre> <pre>v2.RandomGrayscale([p])</pre> <pre>v2.GaussianBlur(kernel_size[, sigma])</pre> <pre>v2.RandomInvert([p])</pre> <pre>v2.RandomPosterize(bits[, p])</pre> <pre>v2.RandomSolarize(threshold[, p])</pre> <pre>v2.RandomAdjustSharpness(sharpness_factor[, p])</pre> <pre>v2.RandomAutocontrast([p])</pre> <pre>v2.RandomEqualize([p])</pre>

Torchvision - Transforms

Docs > Transforming and augmenting images



Shortcuts

TRANSFORMING AND AUGMENTING IMAGES

Torchvision supports common computer vision transformations in the `torchvision.transforms` and `torchvision.transforms.v2` modules. Transforms can be used to transform or augment data for training or inference of different tasks (image classification, detection, segmentation, video classification).

[Transforming and augmenting images](#)

[Start here](#)

[+ Supported input types and conventions](#)

[V1 or V2? Which one should I use?](#)

[Performance considerations](#)

[Transform classes, functionals, and kernels](#)

<https://pytorch.org/vision/main/transforms.html>

Torchvision - Transforms.Compose

- Aplicamos diversas transformações nos exemplos de treinamento
 - ◆ Devemos montar um pipeline com cada transformação e o intervalo de parâmetros que queremos utilizar
 - ◆ Geramos um pipeline para treino e validação
 - ◆ No geral, as transformações que alteram as propriedades dos dados devem ser aplicadas no conjunto de treino
 - ◆ As transformações de validação englobam apenas a conversão de tipos e normalização dos dados

Torchvision - Transforms.Compose

COMPOSE

CLASS `torchvision.transforms.Compose(transforms)` [\[SOURCE\]](#)

Composes several transforms together. This transform does not support torchscript. Please, see the note below.

Parameters:

transforms (list of `Transform` objects) – list of transforms to compose.

Example

```
>>> transforms.Compose([
>>>     transforms.CenterCrop(10),
>>>     transforms.PILToTensor(),
>>>     transforms.ConvertImageDtype(torch.float),
>>> ])
```

<https://pytorch.org/vision/main/generated/torchvision.transforms.Compose.html?highlight=compose#torchvision.transforms.Compose>

Torchvision - Transforms.Compose

COMPOSE

CLASS `torchvision.transforms.Compose(transforms)` [SOURCE]

Composition

Param

`v2.Compose(transforms)`

[BETA] Composes several transforms together.

Example

`v2.RandomApply(transforms[, p])`

[BETA] Apply randomly a list of transformations with a given probability.

>>
>>
>>
>>

`v2.RandomChoice(transforms[, p])`

[BETA] Apply single transformation randomly picked from a list.

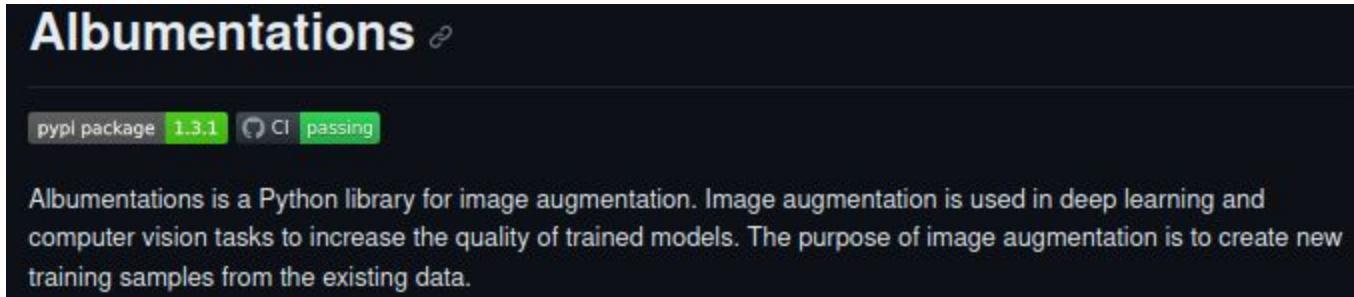
`v2.RandomOrder(transforms)`

[BETA] Apply a list of transformations in a random order.

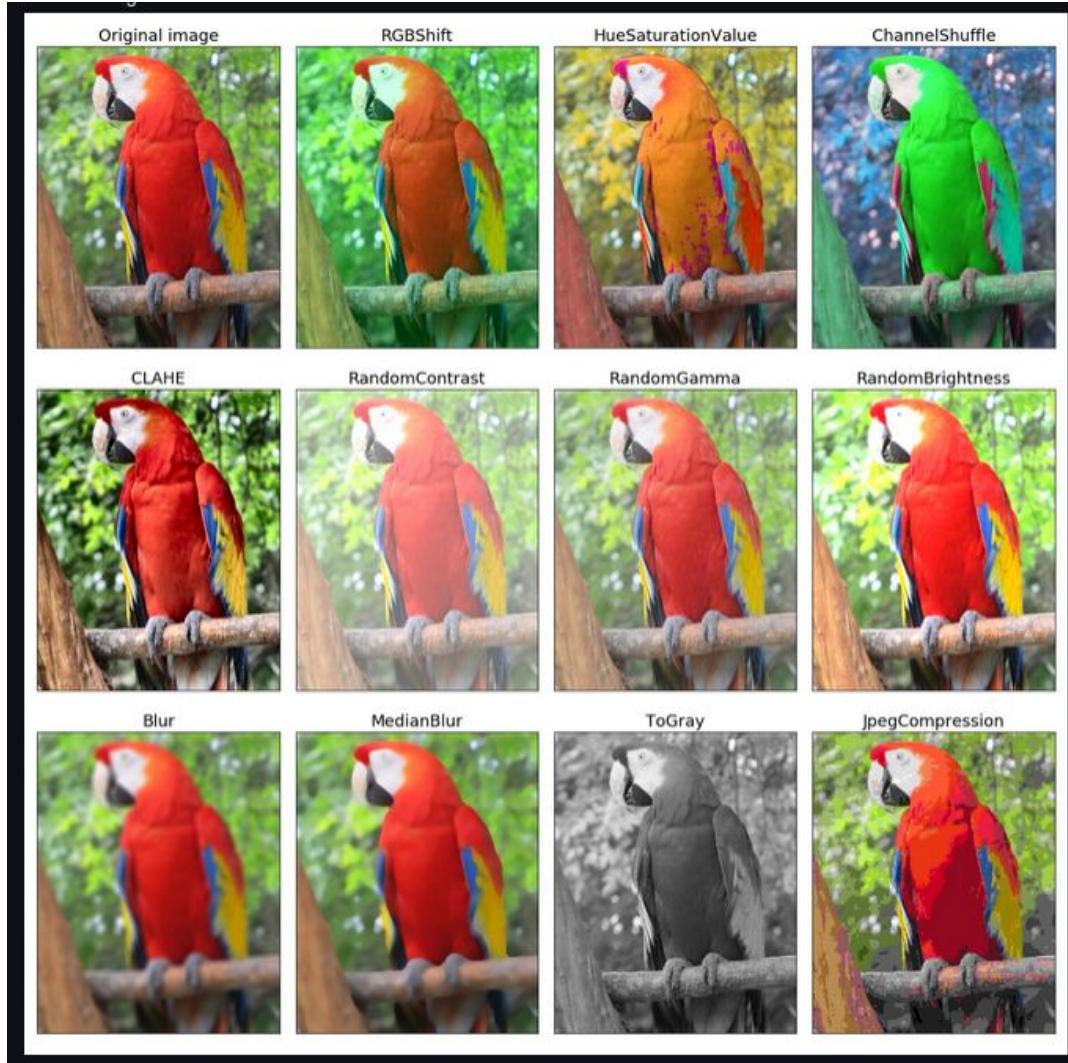
<https://pytorch.org/vision/main/generated/torchvision.transforms.Compose.html?highlight=compose#torchvision.transforms.Compose>

Albumentations

- TorchVision funciona apenas para manipulação de imagens da classe PIL.Image ou tensores
- OpenCV (array)? → Albumentations

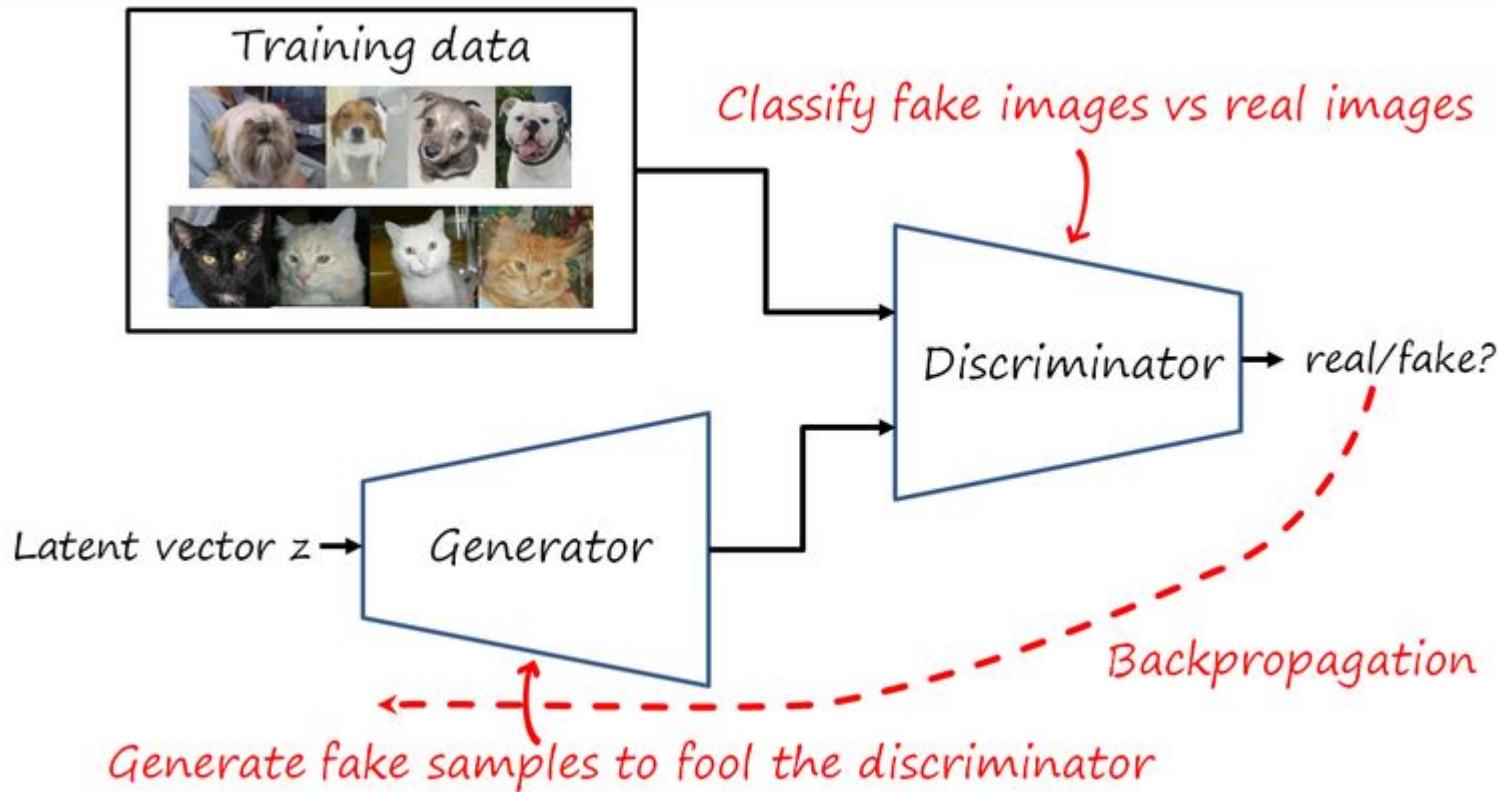


Albumentations



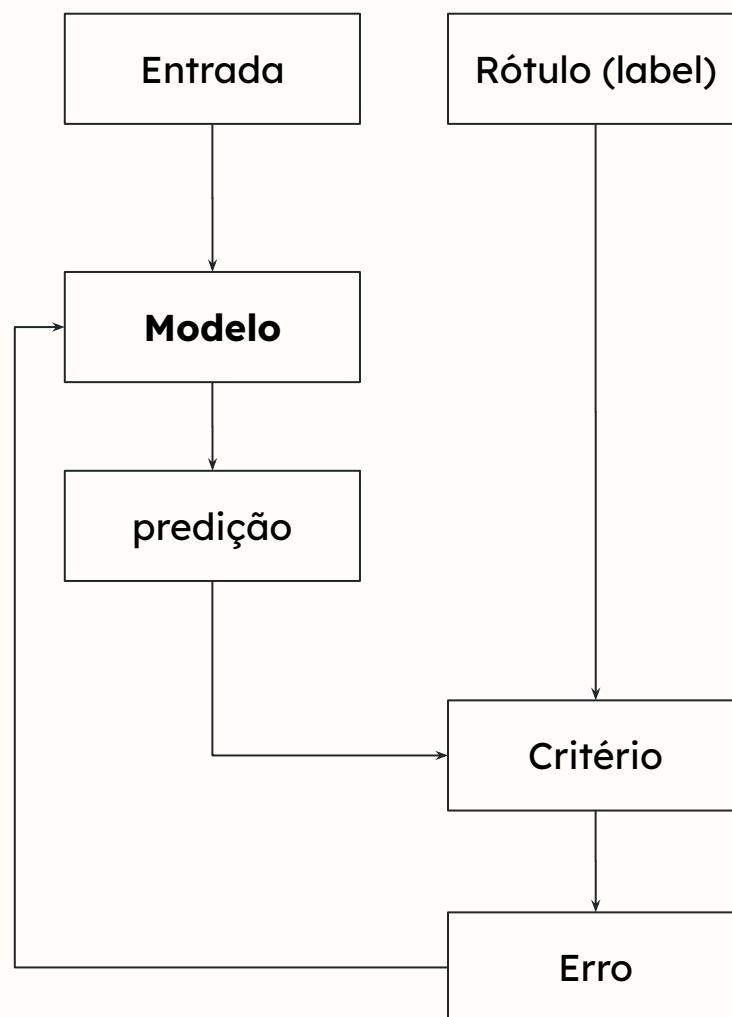
Generative Models

Generative Adversarial Networks (GANs)

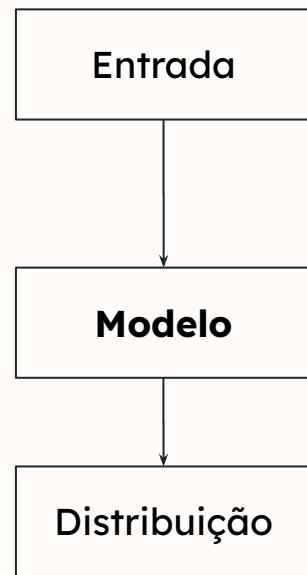


- Combinar o aprendizado não supervisionado e supervisionado

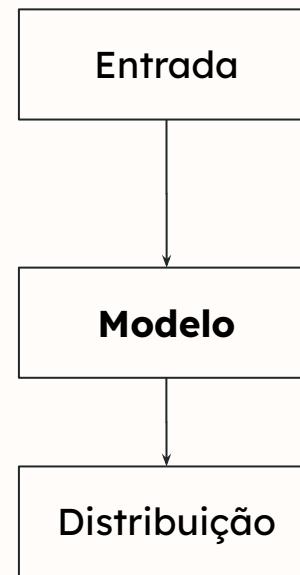
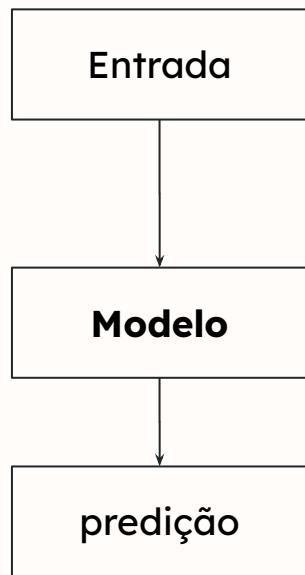
Aprendizado supervisionado



Aprendizado supervisionado

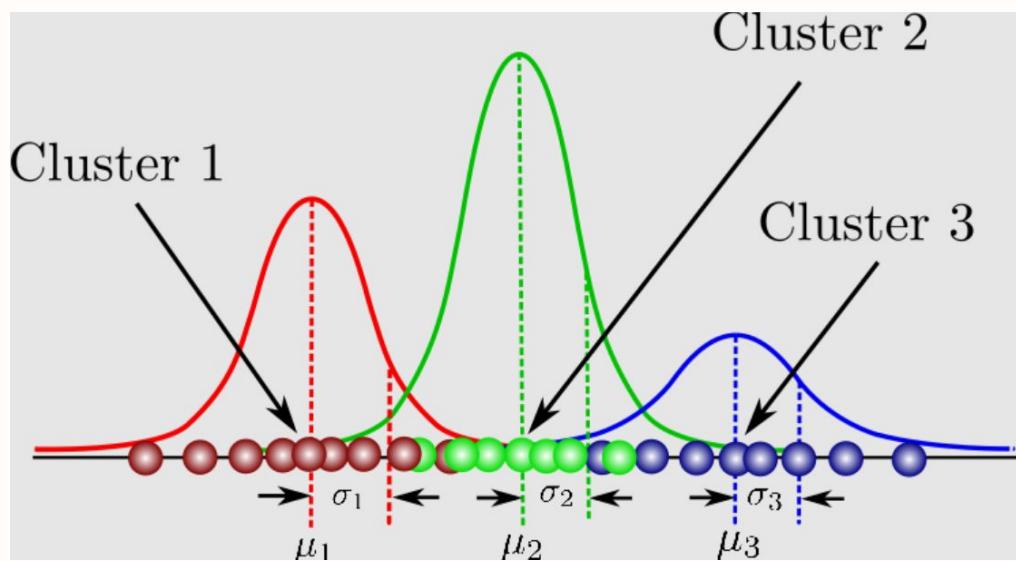


Modelos discriminativos e generativos

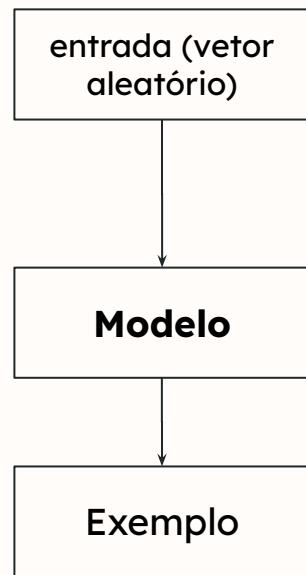


Modelos generativos

- Bayes
- GMM (gaussian mixture models)
- RBM (restricted boltzmann machine)
- DBN (deep belief networks)
- VAE (variational autoencoders)
- GANs (Generatives Adversarial Networks)

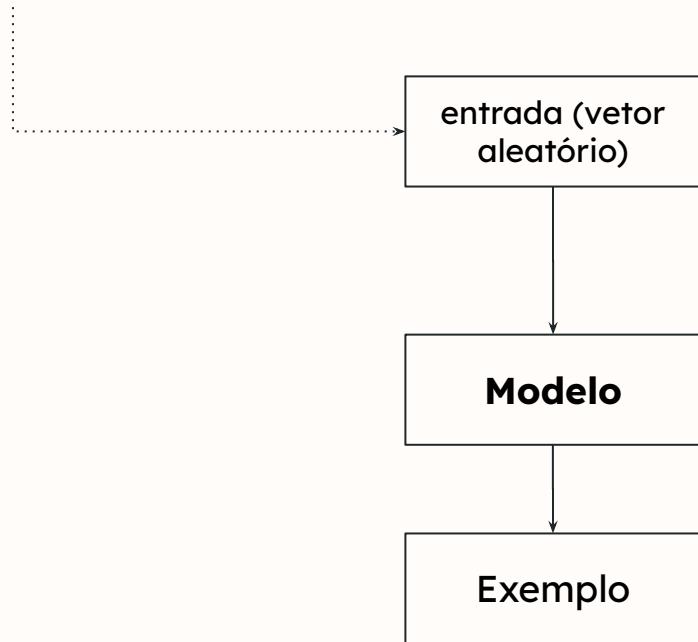


Gerador



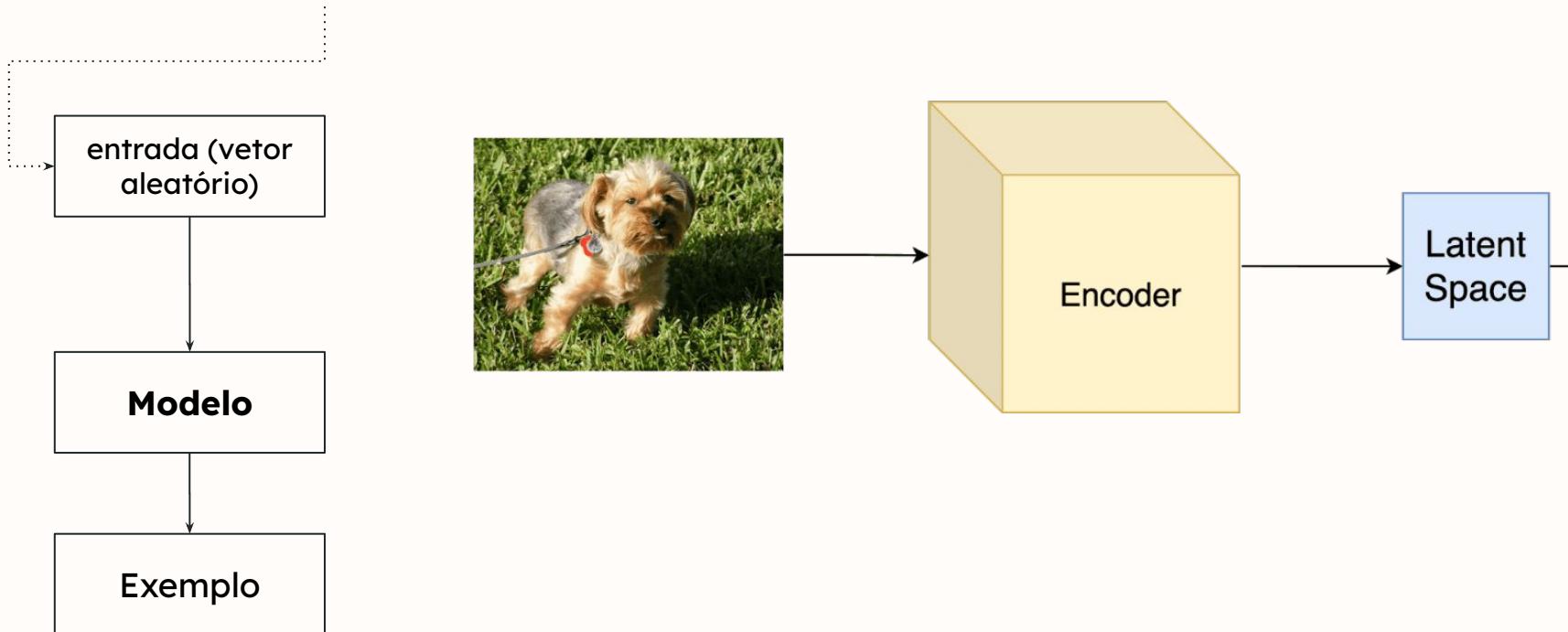
Gerador

espaço latente



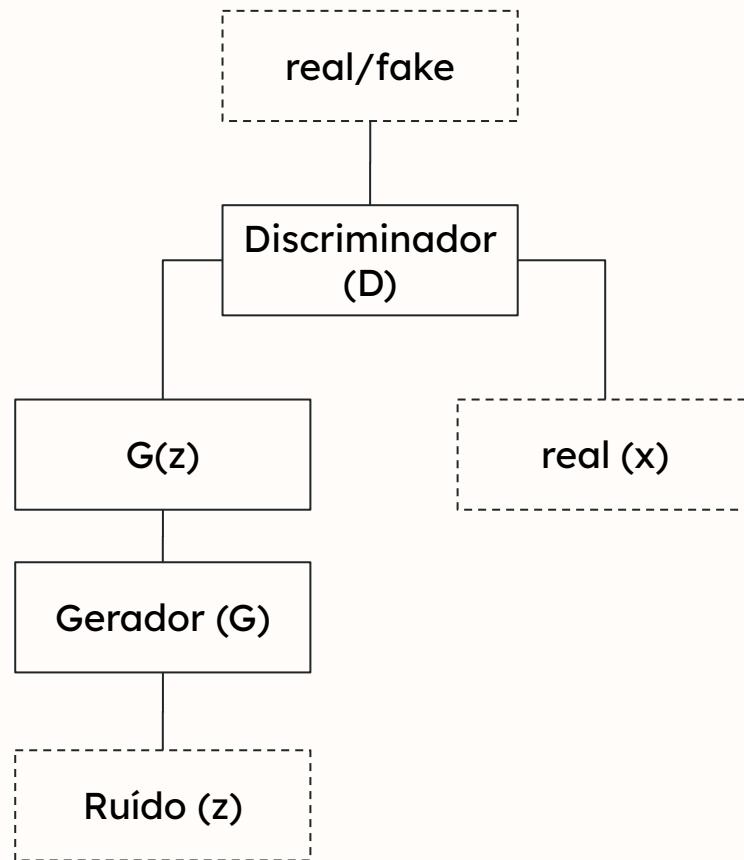
Gerador

espaço latente



Generative Adversarial Networks (GANs)

- Duas redes aprendendo tarefas de forma concorrente
 - ◆ Um gerador tenta criar exemplos para confundir o discriminador
 - ◆ O discriminador aprende a distinguir imagens reais de fakes



Generative Adversarial Networks (GANs)

- D é um classificador binário que distingue se um exemplo é real ou fake. O discriminador é treinado minimizando a função cross-entropy

$$\min (-y \log D(x) - (1 - y) \log (1 - D(x)))_D$$

Generative Adversarial Networks (GANs)

- O Gerador G, processa a variável z (ruído), i.e., $G(z)$. Gerando, $G(z) = x'$.
 - ◆ z é gerado a partir de uma distribuição normal $z \sim N(0,1)$. Chamamos z de variável latente
 - ◆ Queremos maximizar o cross-entropy para quando $y=0$
 - ◆ Se o gerador conseguir confundir D, então $D(x') \sim 1$

$$\max_G(-(1-y) \log (1 - D(G(z)))) = \max_G(-\log (1 - D(G(z))))$$

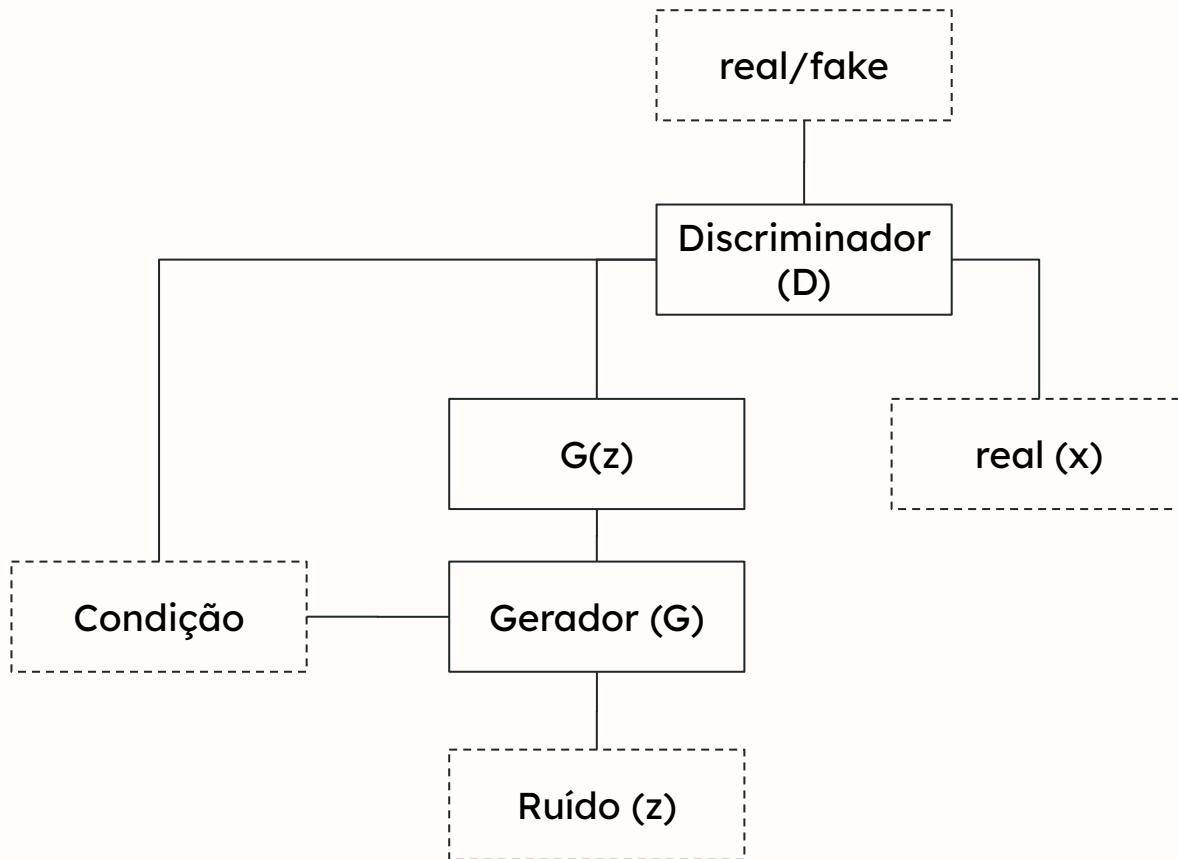
$$\min_G(-y \log (D(G(z)))) = \min_G(-\log (D(G(z))))$$

Generative Adversarial Networks (GANs)

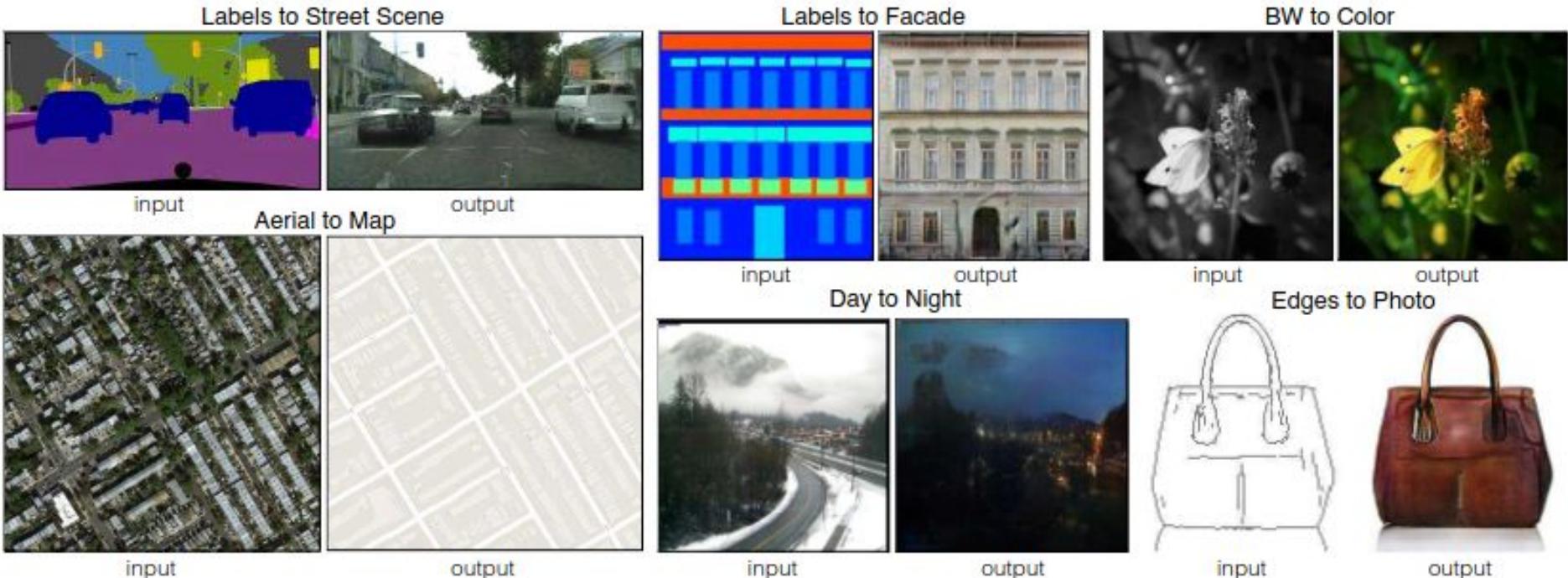
- A loss total é soma onde queremos minimizar o erro em D e maximiza-lo com G

$$\min (\max (-E_x \log D(x) - E_z \log (1 - D(G(z))))_G)_D$$

Conditional Generative Adversarial Networks (GANs)

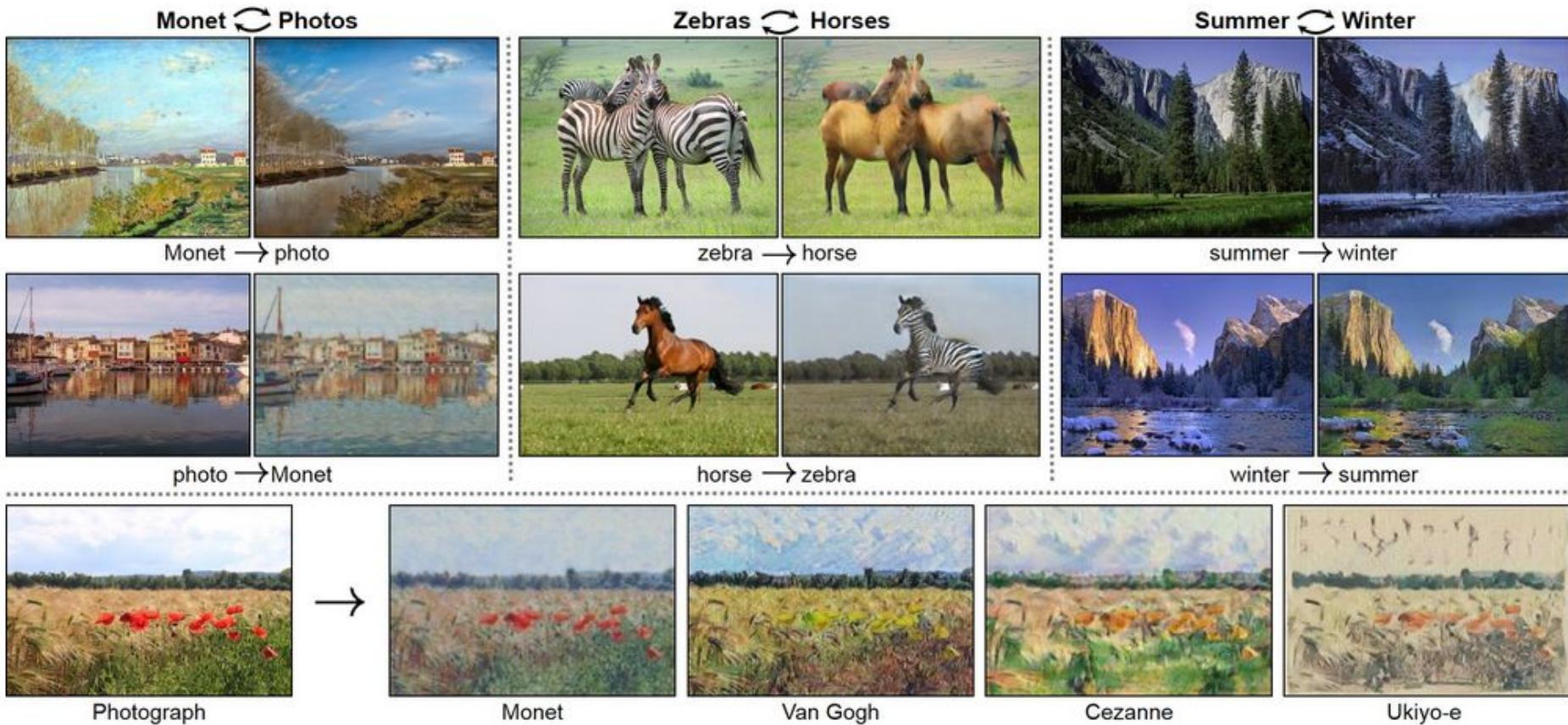


Aplicações GANs



Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

Aplicações GANs



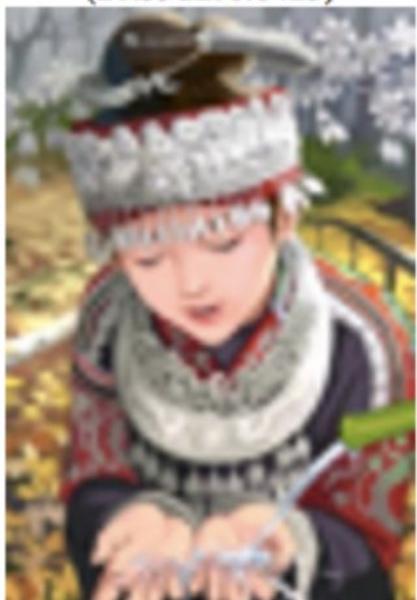
Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." *Proceedings of the IEEE international conference on computer vision*. 2017.

Aplicações GANs

original



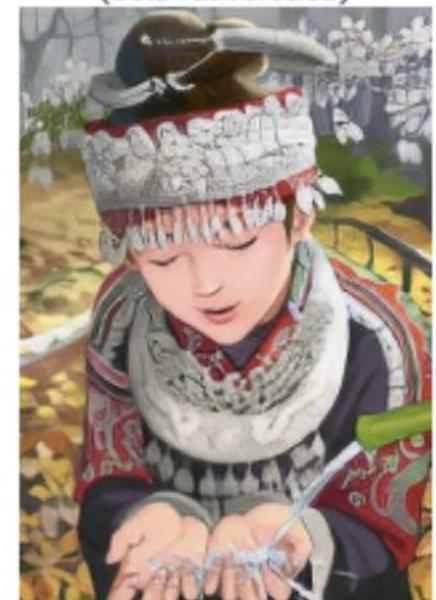
bicubic
(21.59dB/0.6423)



SRResNet
(23.44dB/0.7777)



SRGAN
(20.34dB/0.6562)



Aplicações GANs

Deepfake Videos: GAN Synthesizes a Talking Head From a Single Photo

25 May 2019



<https://neurohive.io/en/news/deepfake-videos-gan-synthesizes-a-video-from-a-single-photo/>