

Convolutional Neural Networks

1001513 – Aprendizado de Máquina 2
Turma A – 2023/2
Prof. Murilo Naldi



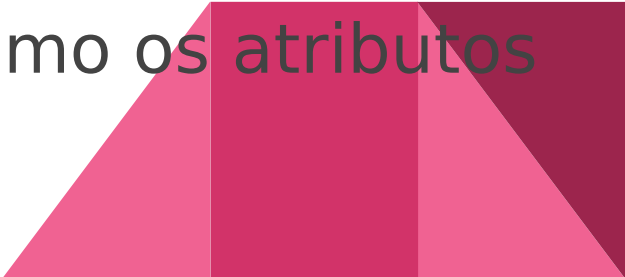
naldi@ufscar.br



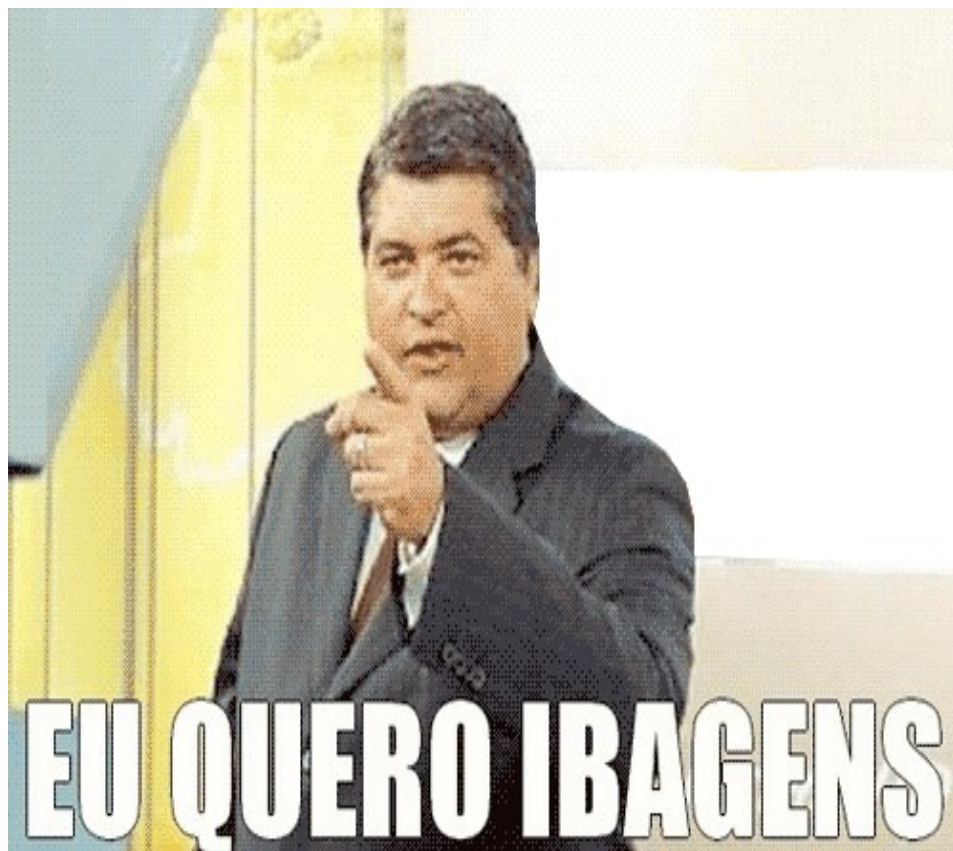
Agradecimentos

- Pessoas que colaboraram com a produção deste material: Diego Silva, Ricardo Cerri e Moacir Ponti

De Camadas Totalmente Conectadas para Convoluções

- Redes totalmente conectadas são mais adequadas para dados tabulados (registros):
 - Linhas representa exemplos de treinamento
 - Colunas representam características (atributos)
 - Pode haver interações entre os atributos
 - Porém, não assumimos a priori nenhuma estrutura correspondente a como os atributos interagem
- 

O que eu quero pra aula de hoje?



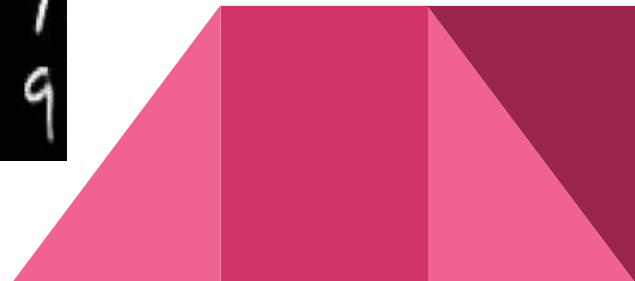
Créditos
Diego Silva

Sabemos trabalhar com dados estruturados

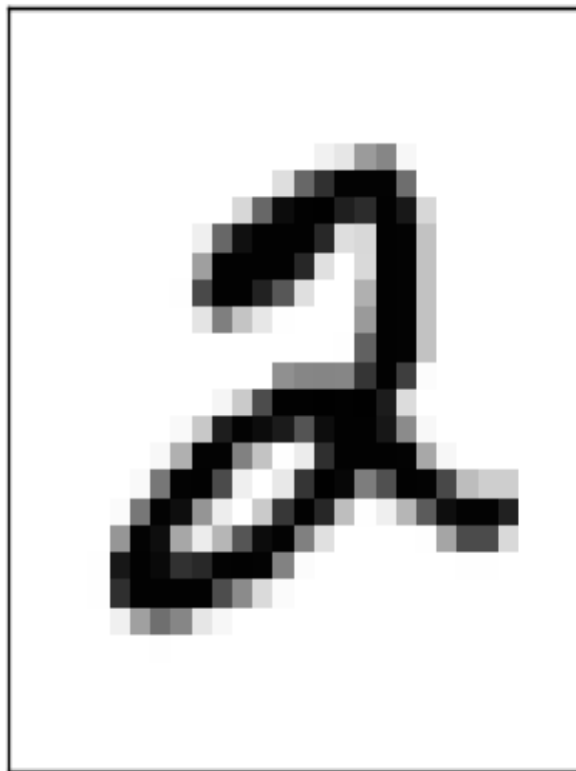
E o que fazemos com imagens?



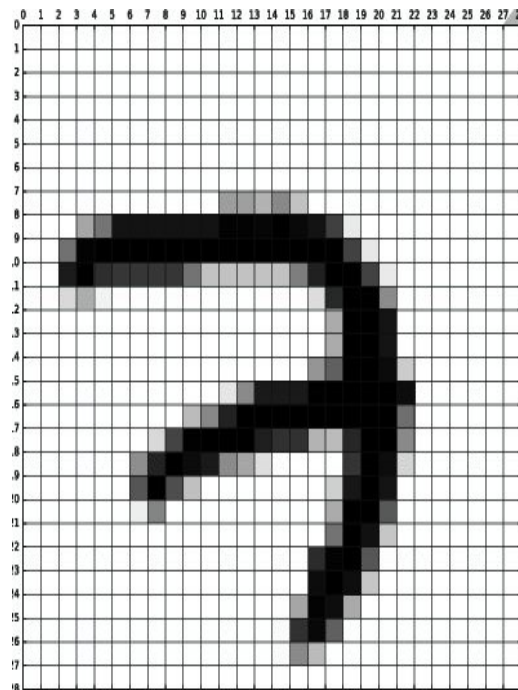
MNIST dataset



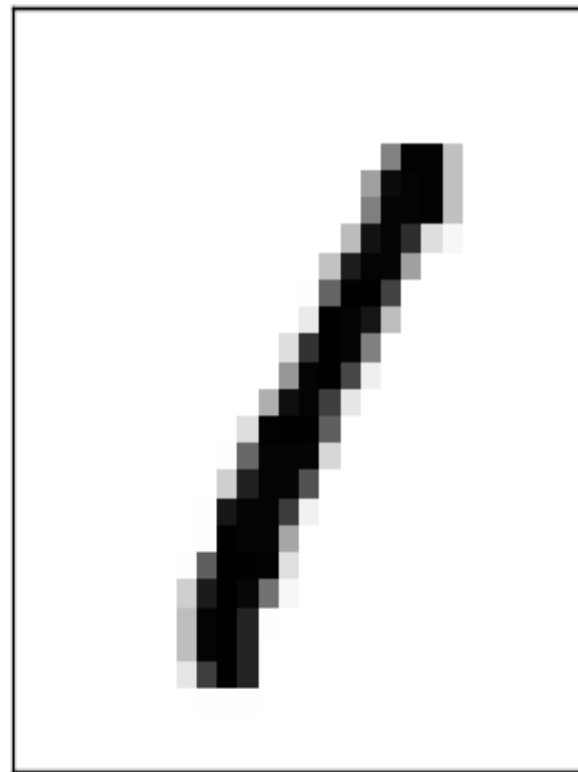
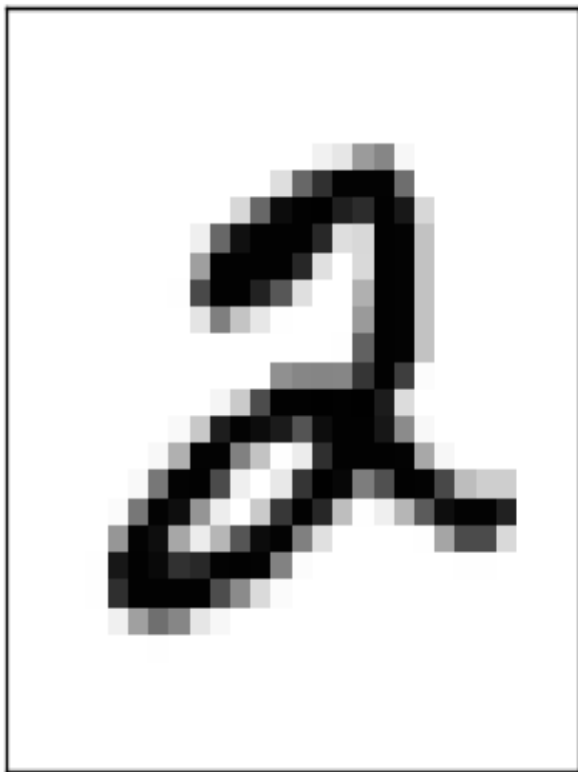
Exemplo de RNA em imagens



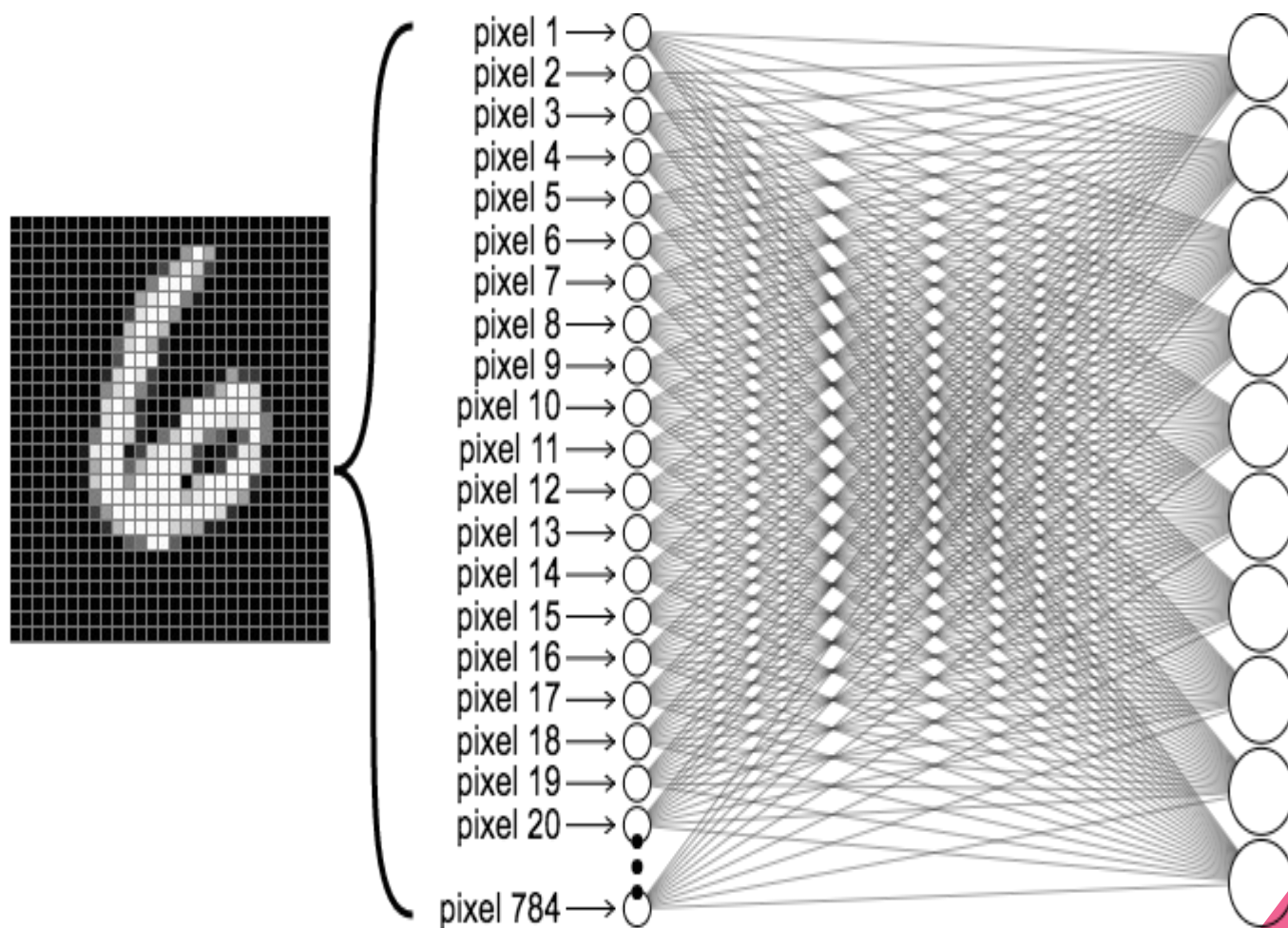
Exemplo de RNA em imagens



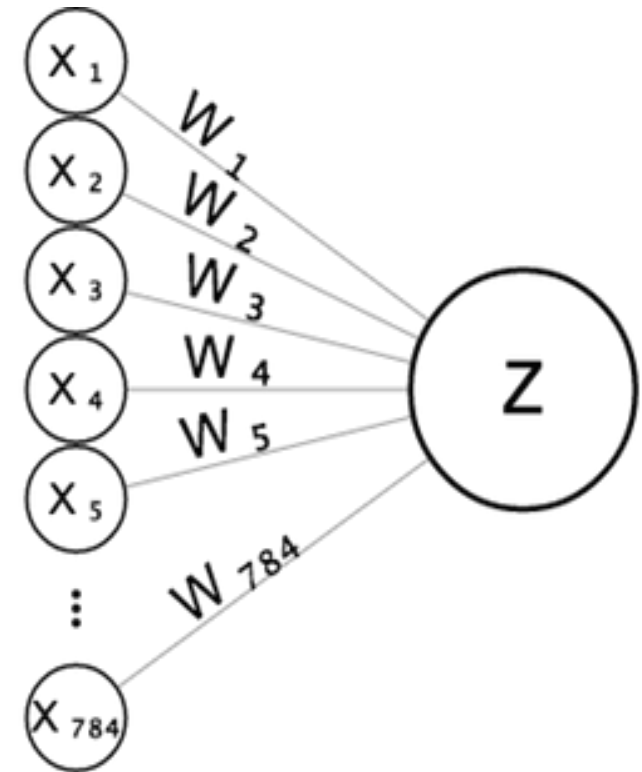
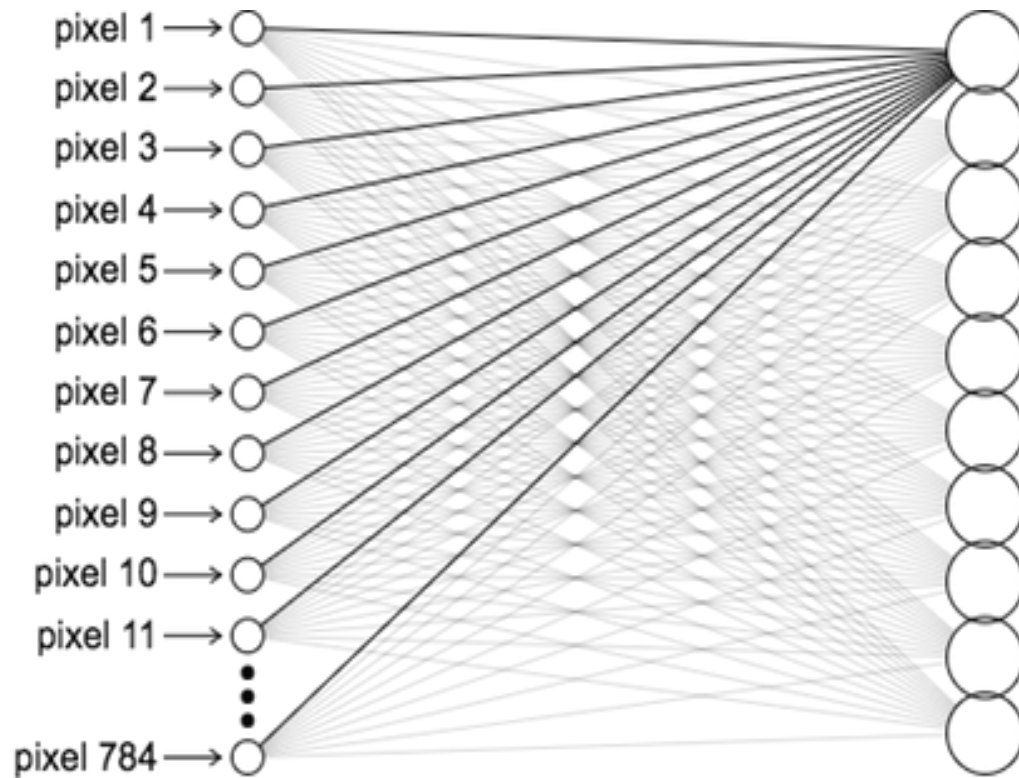
Exemplo de RNA em imagens



Exemplo de RNA em imagens

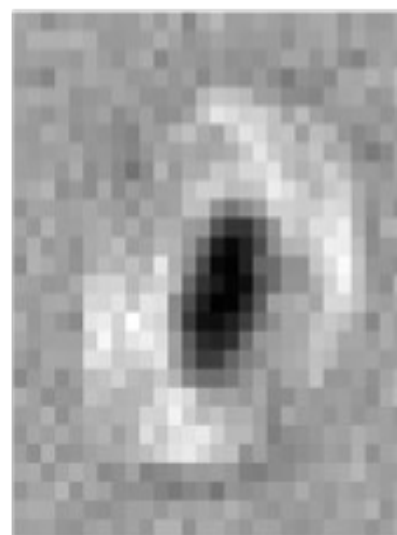


Exemplo de RNA em imagens - um parênteses



Exemplo de configuração de pesos

W_1	W_2	W_3	\dots	W_{28}
W_{29}	W_{30}	W_{31}		W_{56}
W_{57}	W_{58}	W_{59}		W_{84}
\vdots			\ddots	
W_{757}	W_{758}	W_{759}		W_{784}

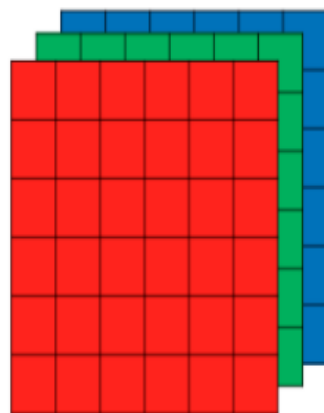


Visualização
dos pesos

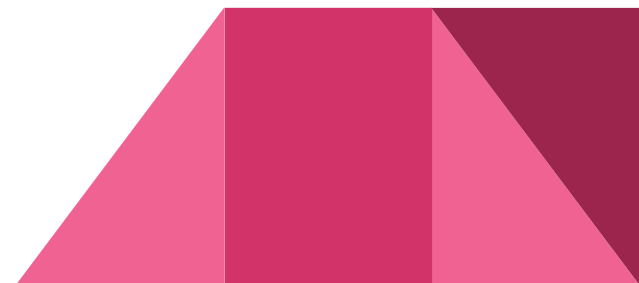
Se complicar só um pouquinho...

... já quebra tudo.

Por exemplo, o que fazer com canais de cores?



6x6x3

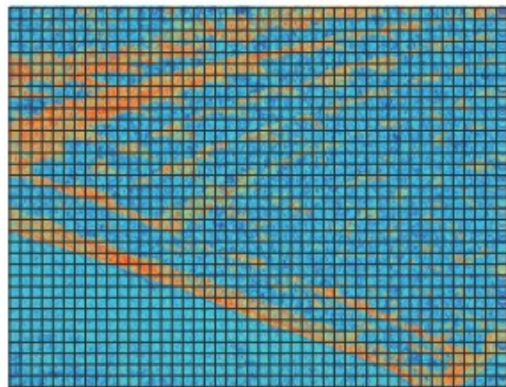


Se complicar só um pouquinho...

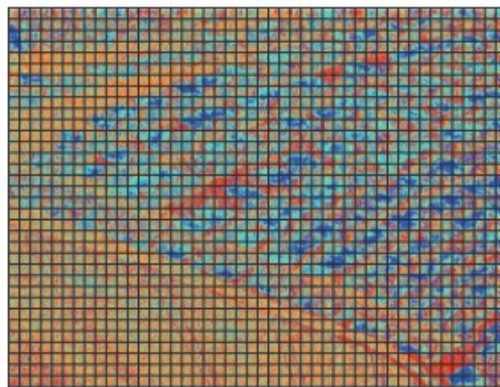
Original



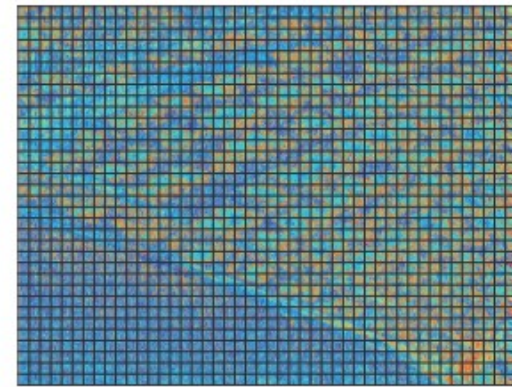
Água



NDVI (índice de vegetação)



Vegetação



Exemplo: CIFAR-10

airplane



automobile



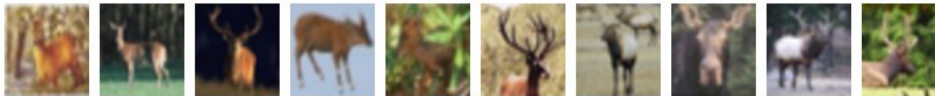
bird



cat



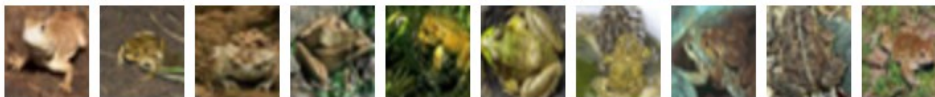
deer



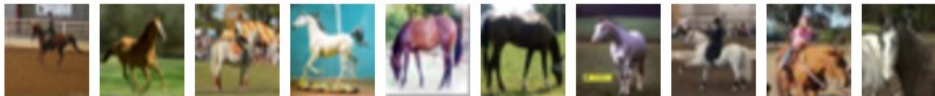
dog



frog



horse



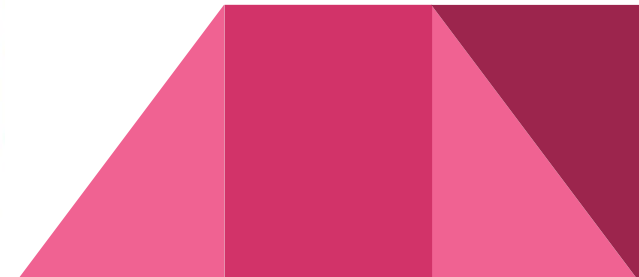
ship



truck



6000 imagens
32x32 pixels
10 classes
Classes
balanceadas



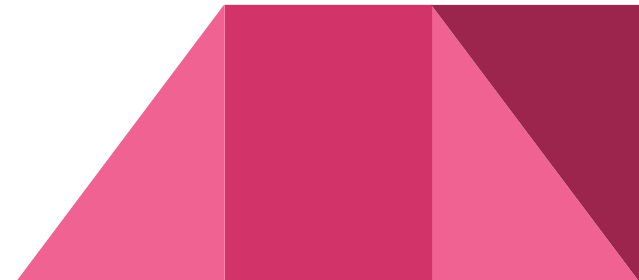
Exemplo: CIFAR-10

Usar a imagem como **dado bruto** (o valor de cada *pixel* ser um valor de atributo) ignora completamente a dependência entre *pixels* vizinhos.


Uma forma (ou contorno, textura, etc) não é dada pelo valor de um *pixel*, mas sim por um **conjunto de *pixels* próximos** uns aos outros.

Alternativa: extração de características

Alternativa 2: aprender como extrair essas características




Exemplo: CIFAR-10

 Playground Prediction Competition

CIFAR-10 - Object Recognition in Images

Identify the subject of 60,000 labeled images

 Kaggle · 231 teams · 6 years ago











[Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#)

Join Competition

[Public Leaderboard](#) [Private Leaderboard](#)

The private leaderboard is calculated with approximately 97% of the test data.
This competition has completed. This leaderboard reflects the final standings.

Refresh

#	Δpub	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	—	DeepCNet			0.95530	18	6y
2	—	jiki			0.94740	42	6y
3	—	Anil Thomas			0.94300	3	6y
4	—	Frank Sharp			0.94190	13	6y
5	—	nagadomi			0.94150	16	6y
6	—	Phil & Triskelion & Kazanova		  	0.94120	107	6y
7	—	Daniel Nouri			0.93540	4	6y
8	—	Terry			0.93330	3	6y

Convolutional Neural Networks (CNN)

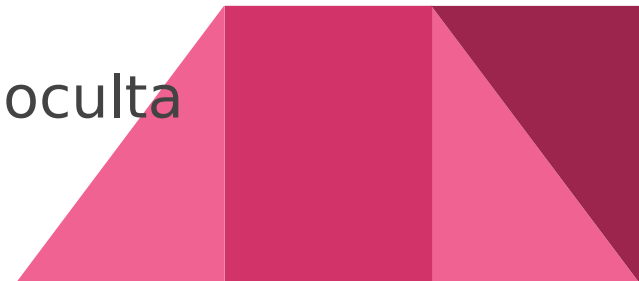
As CNNs foram propostas para o reconhecimento visual, baseando-se em conceitos da visão humana



Convolutional Neural Networks (CNN)

Considera a **relação entre pixels vizinhos**, dada por **filtros**, aplicados por uma operação de **convolução**.

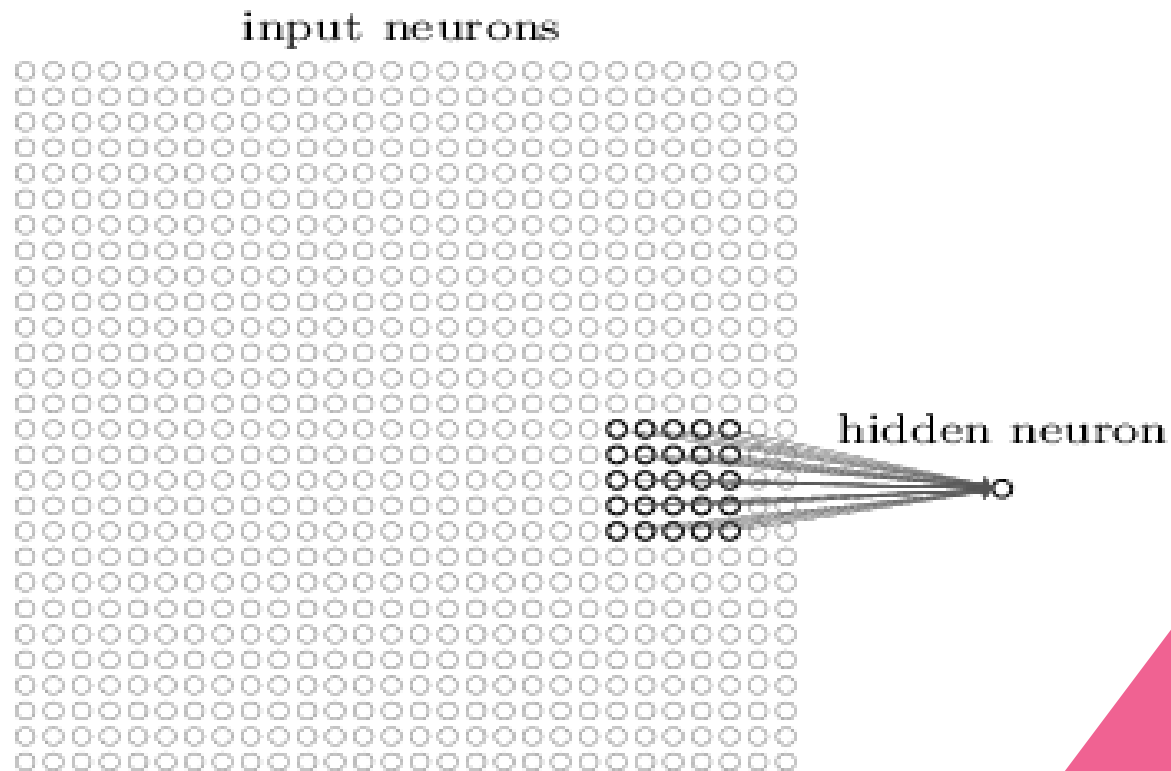
- Basicamente, multiplicação com janela deslizante
- Camadas convolucionais tipicamente requerem bem menos parâmetros do que camadas totalmente conectadas
- Noção de localidade aonde considera uma pequena vizinha
 - Faz o papel similar a uma camada oculta



Convolutional Neural Networks (CNN)

Cada região é chamada de *local receptive field*

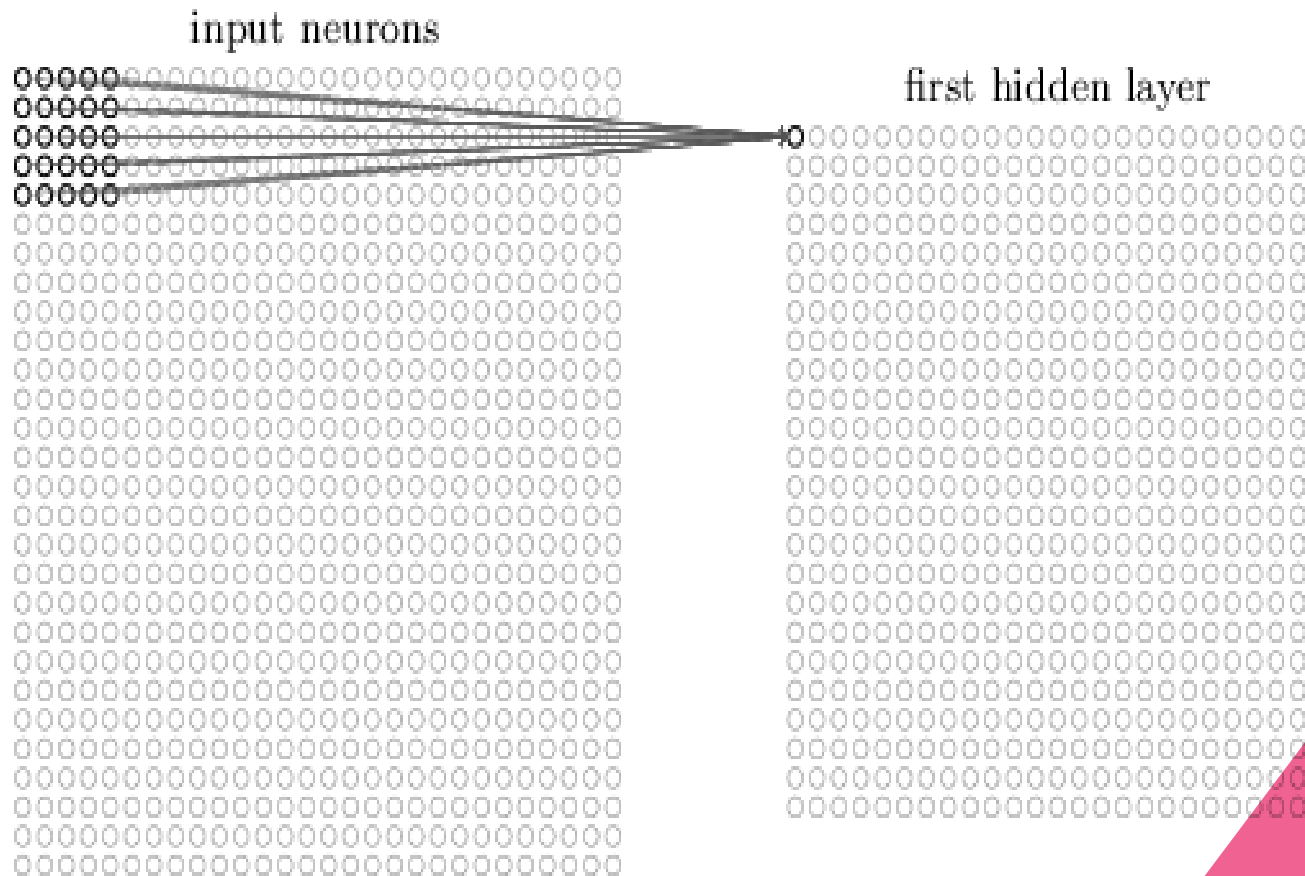
- Para cada região, temos um neurônio da camada escondida
- Onde também é aplicada uma função de ativação



Convolutional Neural Networks (CNN)

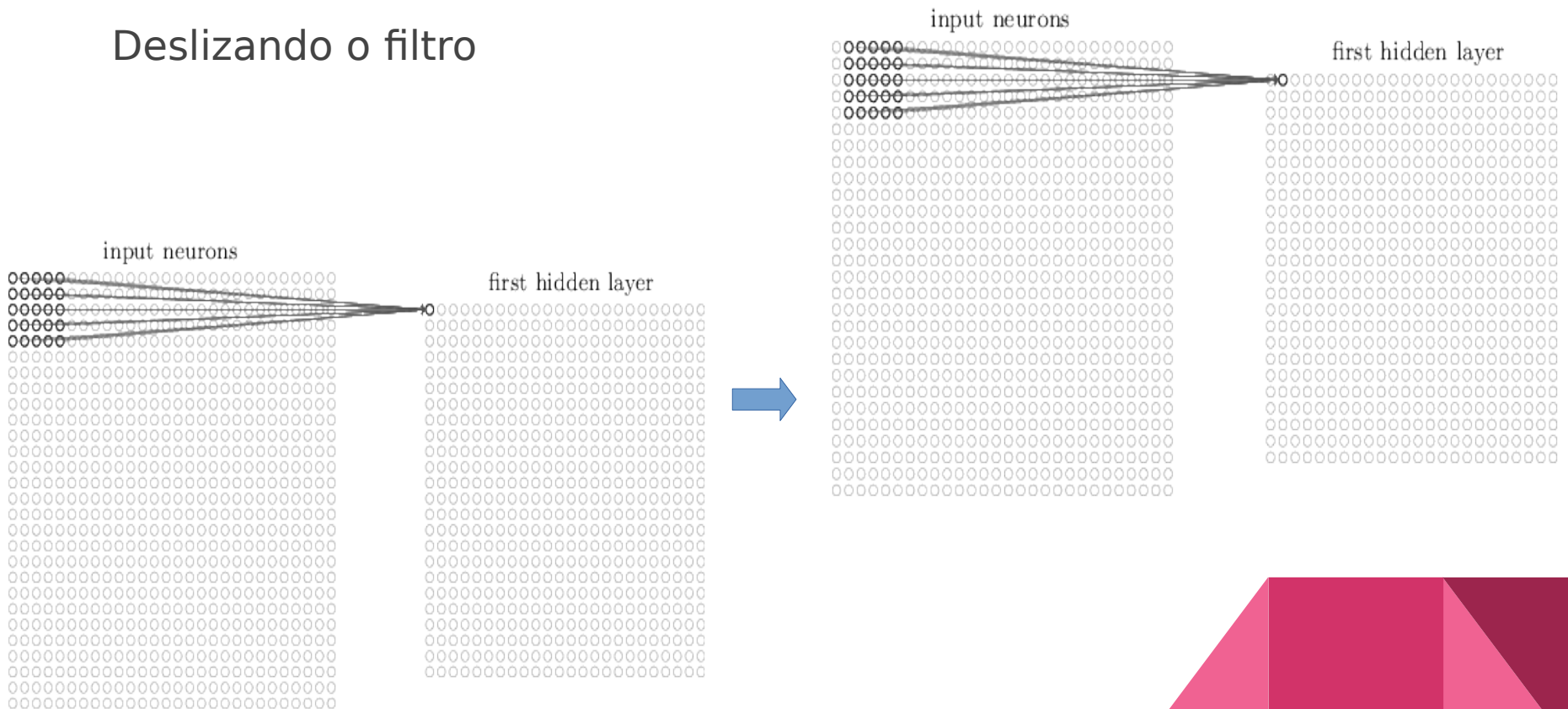
O mapeamento da entrada para a escondida é chamado de *feature map*

- Os pesos do *feature map* são os *shared weights*
- Esses pesos definem um *kernel* ou filtro



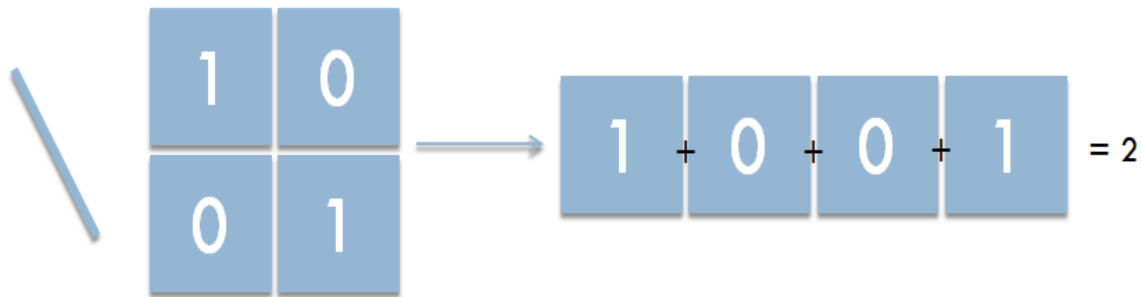
Convolutional Neural Networks (CNN)

Deslizando o filtro

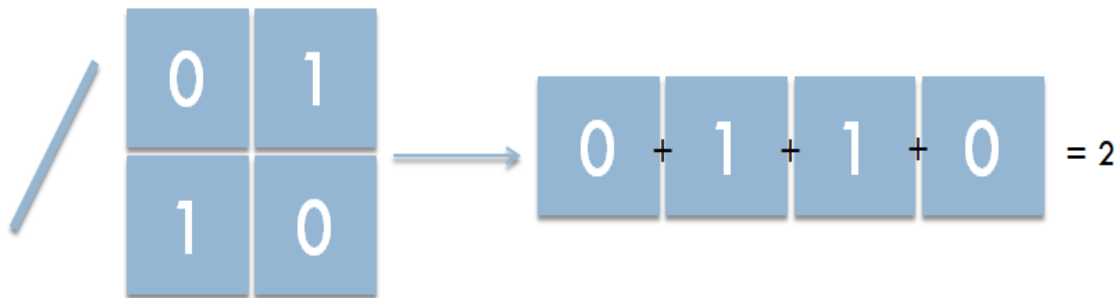


Convolutional Neural Networks (CNN)

Mas o que são esses “filtros”?

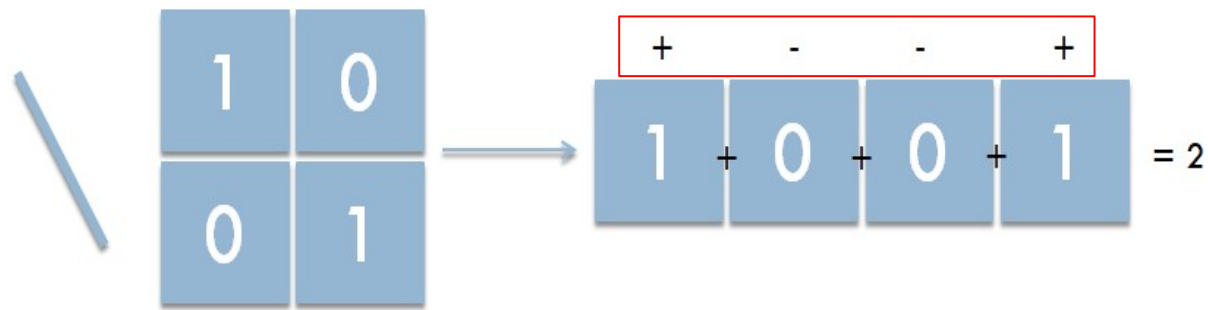


Esse é um bom filtro?

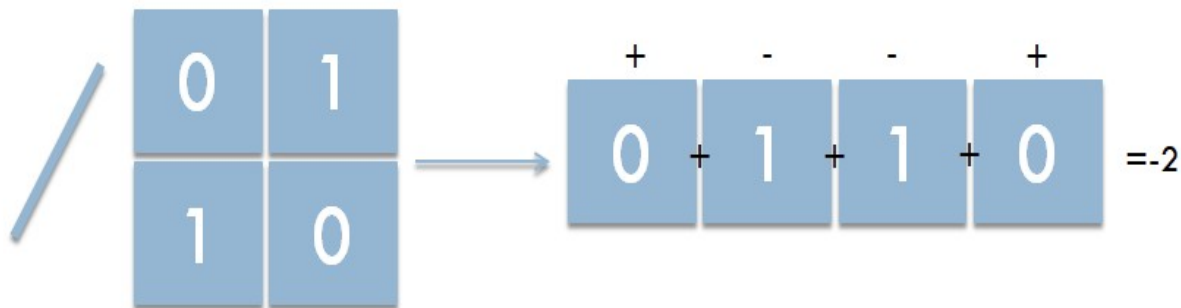


Convolutional Neural Networks (CNN)

Mas o que são esses “filtros”?



Esse é um bom filtro?



Convolutional Neural Networks (CNN)

Para uma imagem 2x2, com apenas + e -, possuímos $2^4=16$ possibilidades

×	×	×	×	×	×	×	✓																																
<table><tr><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td></tr></table>	-	-	-	-	<table><tr><td>+</td><td>-</td></tr><tr><td>-</td><td>-</td></tr></table>	+	-	-	-	<table><tr><td>-</td><td>+</td></tr><tr><td>-</td><td>-</td></tr></table>	-	+	-	-	<table><tr><td>-</td><td>-</td></tr><tr><td>+</td><td>-</td></tr></table>	-	-	+	-	<table><tr><td>-</td><td>-</td></tr><tr><td>-</td><td>+</td></tr></table>	-	-	-	+	<table><tr><td>+</td><td>+</td></tr><tr><td>-</td><td>-</td></tr></table>	+	+	-	-	<table><tr><td>+</td><td>-</td></tr><tr><td>+</td><td>-</td></tr></table>	+	-	+	-	<table><tr><td>+</td><td>-</td></tr><tr><td>-</td><td>+</td></tr></table>	+	-	-	+
-	-																																						
-	-																																						
+	-																																						
-	-																																						
-	+																																						
-	-																																						
-	-																																						
+	-																																						
-	-																																						
-	+																																						
+	+																																						
-	-																																						
+	-																																						
+	-																																						
+	-																																						
-	+																																						
✓	×	×	×	×	×	×	×																																
<table><tr><td>-</td><td>+</td></tr><tr><td>+</td><td>-</td></tr></table>	-	+	+	-	<table><tr><td>-</td><td>+</td></tr><tr><td>-</td><td>+</td></tr></table>	-	+	-	+	<table><tr><td>-</td><td>-</td></tr><tr><td>+</td><td>+</td></tr></table>	-	-	+	+	<table><tr><td>+</td><td>+</td></tr><tr><td>+</td><td>-</td></tr></table>	+	+	+	-	<table><tr><td>+</td><td>+</td></tr><tr><td>-</td><td>+</td></tr></table>	+	+	-	+	<table><tr><td>+</td><td>-</td></tr><tr><td>+</td><td>+</td></tr></table>	+	-	+	+	<table><tr><td>-</td><td>+</td></tr><tr><td>+</td><td>+</td></tr></table>	-	+	+	+	<table><tr><td>+</td><td>+</td></tr><tr><td>+</td><td>+</td></tr></table>	+	+	+	+
-	+																																						
+	-																																						
-	+																																						
-	+																																						
-	-																																						
+	+																																						
+	+																																						
+	-																																						
+	+																																						
-	+																																						
+	-																																						
+	+																																						
-	+																																						
+	+																																						
+	+																																						
+	+																																						

Convolutional Neural Networks (CNN)

Para uma imagem 2x2, com apenas + e -, possuímos $2^4=16$ possibilidades

✗	✗	✗	✗	✗	✗	✗	✓																																
<table><tr><td>-</td><td>-</td></tr><tr><td>-</td><td>-</td></tr></table>	-	-	-	-	<table><tr><td>+</td><td>-</td></tr><tr><td>-</td><td>-</td></tr></table>	+	-	-	-	<table><tr><td>-</td><td>+</td></tr><tr><td>-</td><td>-</td></tr></table>	-	+	-	-	<table><tr><td>-</td><td>-</td></tr><tr><td>+</td><td>-</td></tr></table>	-	-	+	-	<table><tr><td>-</td><td>-</td></tr><tr><td>-</td><td>+</td></tr></table>	-	-	-	+	<table><tr><td>+</td><td>+</td></tr><tr><td>-</td><td>-</td></tr></table>	+	+	-	-	<table><tr><td>+</td><td>-</td></tr><tr><td>+</td><td>-</td></tr></table>	+	-	+	-	<table><tr><td>+</td><td>-</td></tr><tr><td>-</td><td>+</td></tr></table>	+	-	-	+
-	-																																						
-	-																																						
+	-																																						
-	-																																						
-	+																																						
-	-																																						
-	-																																						
+	-																																						
-	-																																						
-	+																																						
+	+																																						
-	-																																						
+	-																																						
+	-																																						
+	-																																						
-	+																																						
✓	✗	✗	✗	✗	✗	✗	✗																																
<table><tr><td>-</td><td>+</td></tr><tr><td>+</td><td>-</td></tr></table>	-	+	+	-	<table><tr><td>-</td><td>+</td></tr><tr><td>-</td><td>+</td></tr></table>	-	+	-	+	<table><tr><td>-</td><td>-</td></tr><tr><td>+</td><td>+</td></tr></table>	-	-	+	+	<table><tr><td>+</td><td>+</td></tr><tr><td>+</td><td>-</td></tr></table>	+	+	+	-	<table><tr><td>+</td><td>+</td></tr><tr><td>-</td><td>+</td></tr></table>	+	+	-	+	<table><tr><td>+</td><td>-</td></tr><tr><td>+</td><td>+</td></tr></table>	+	-	+	+	<table><tr><td>-</td><td>+</td></tr><tr><td>+</td><td>+</td></tr></table>	-	+	+	+	<table><tr><td>+</td><td>+</td></tr><tr><td>+</td><td>+</td></tr></table>	+	+	+	+
-	+																																						
+	-																																						
-	+																																						
-	+																																						
-	-																																						
+	+																																						
+	+																																						
+	-																																						
+	+																																						
-	+																																						
+	-																																						
+	+																																						
-	+																																						
+	+																																						
+	+																																						
+	+																																						

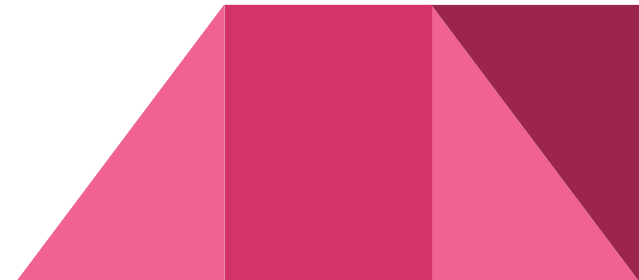
- E se aumentarmos a matriz?
- E se forem modificadas tal que possamos ter valores reais arbitrários, como 1.2, -0.75, 0.98, ...?

Convolutional Neural Networks (CNN)

A vantagem dos pesos compartilhados é a redução de parâmetros

- Supondo um filtro de tamanho 5
- Cada neurônio escondido tem um bias e uma matriz de pesos 5x5 conectada a imagem (*local receptive field*)
- Os mesmos pesos e bias são utilizados para cada um dos neurônios escondidos (*shared weights*)
- Para o j,k-ésimo neurônio escondido, a saída é dada por:

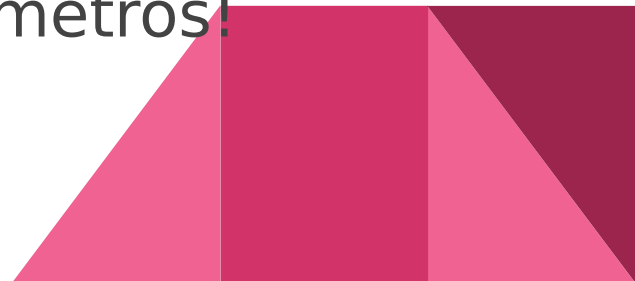
$$\sigma \left(b + \sum_{l=0}^4 \sum_{m=0}^4 w_{l,m} a_{j+l,k+m} \right)$$



Convolutional Neural Networks (CNN)

Para cada filtro de 5×5 precisamos de 25 pesos mais o bias. Se utilizarmos 20 filtros, temos um total de $20 \times 26 = 520$ parâmetros definindo a camada convolucional

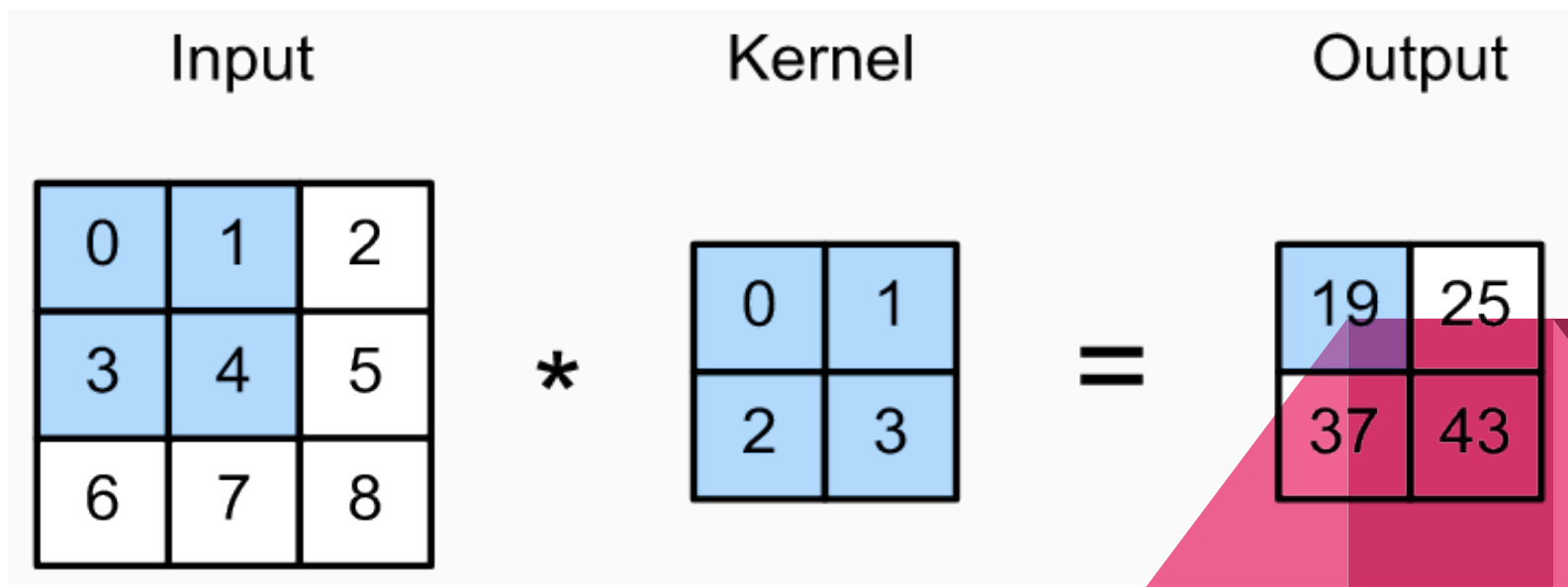
- Em uma rede convencional totalmente conectada, para a base MNIST teríamos $28 \times 28 = 784$ entradas. Com 30 neurônios na camada escondida e 1 bias, teríamos $784 \times 31 + 31 = 24304$ parâmetros!



Convolutional Neural Networks (CNN)

Considere como entrada 3x3 e um filtro (*kernel*) de dimensão 2x2

- Começamos com o filtro no canto superior esquerdo
- Deslizamos da esquerda para a direita e de cima para baixo
- A submatriz de entrada e o filtro são multiplicados elemento a elemento
- A matriz resultante é somada produzindo um único valor escalar.



Convolutional Neural Networks (CNN)

Deslizamos através da matriz de entrada, da esquerda para a direita e de cima para baixo

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

5 x 5 – Image Matrix



1	0	1
0	1	0
1	0	1

3 x 3 – Filter Matrix



1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

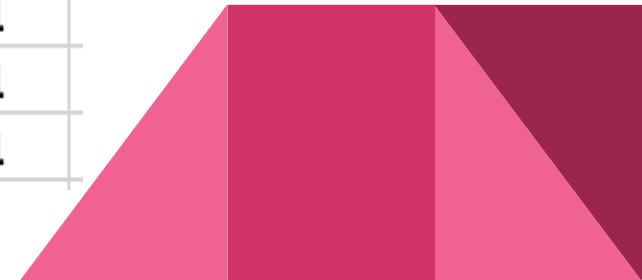
Convolved
Feature

Detecção de padrões

Vamos imaginar um kernel para identificar bordas em imagens:

- Detectar a borda do objeto encontrando o local de mudança de pixel
- Exemplo a imagem abaixo, representada por uma matriz
 - 0 representa a cor preta e 1 a cor branca

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

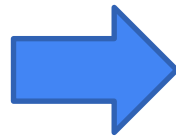


Detecção de padrões

Vamos utilizar um kernel de altura 1 e largura 2: $[1, -1]$

Ao deslizarmos o kernel pela imagem, obtemos a seguinte matriz

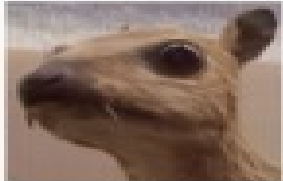



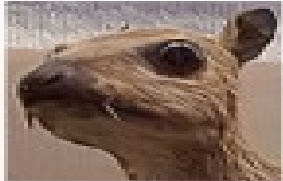
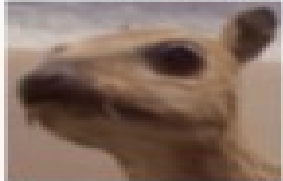
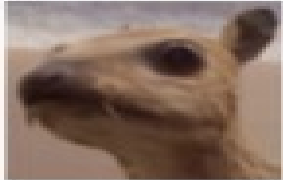
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1



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

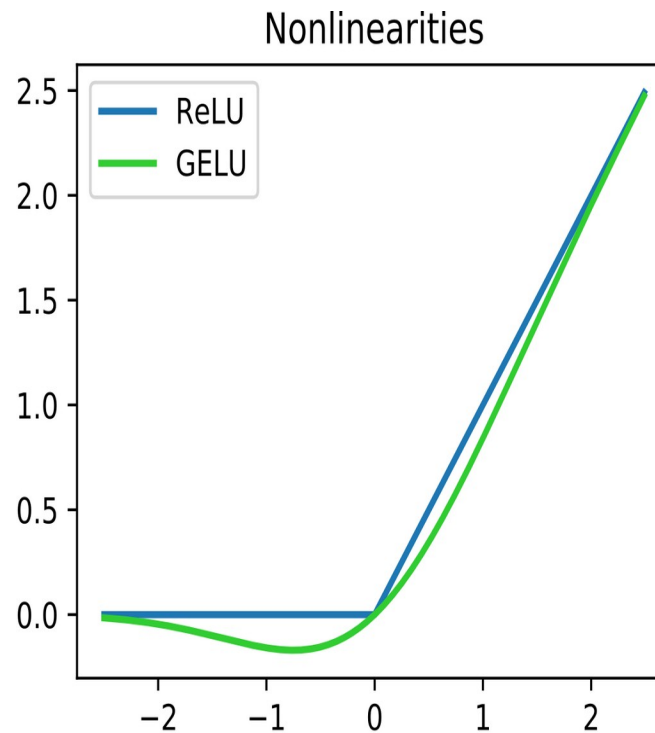
O filtro tem como saída o valor 1 para bordas de branco para preto, e -1 para bordas de preto para branco

Convolução

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolutional Neural Networks (CNN)

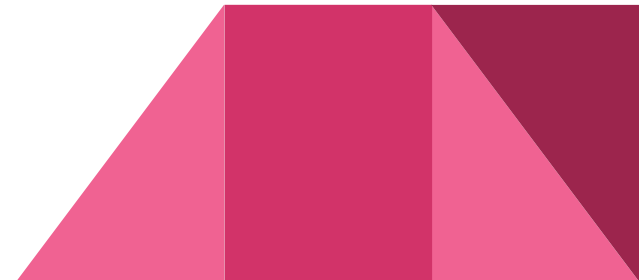
Ativação



Padding

Como vimos, a dimensão da saída da camada convolucional é definida pela dimensão da entrada e pela dimensão do kernel de convolução

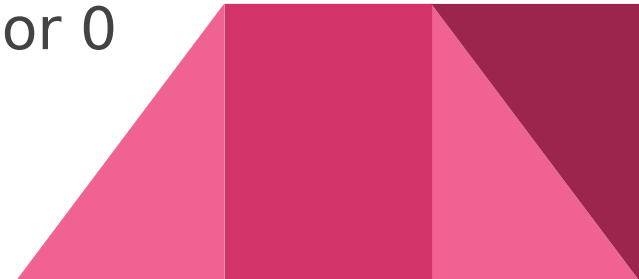
- Se a dimensão da entrada for $(n_h \times n_w)$, e a dimensão do kernel for $(k_h \times k_w)$, então a saída terá dimensão $(n_h - k_h + 1) \times (n_w - k_w + 1)$
- Podemos incorporar técnicas que afetam o tamanho da saída. Padding é uma delas



Padding

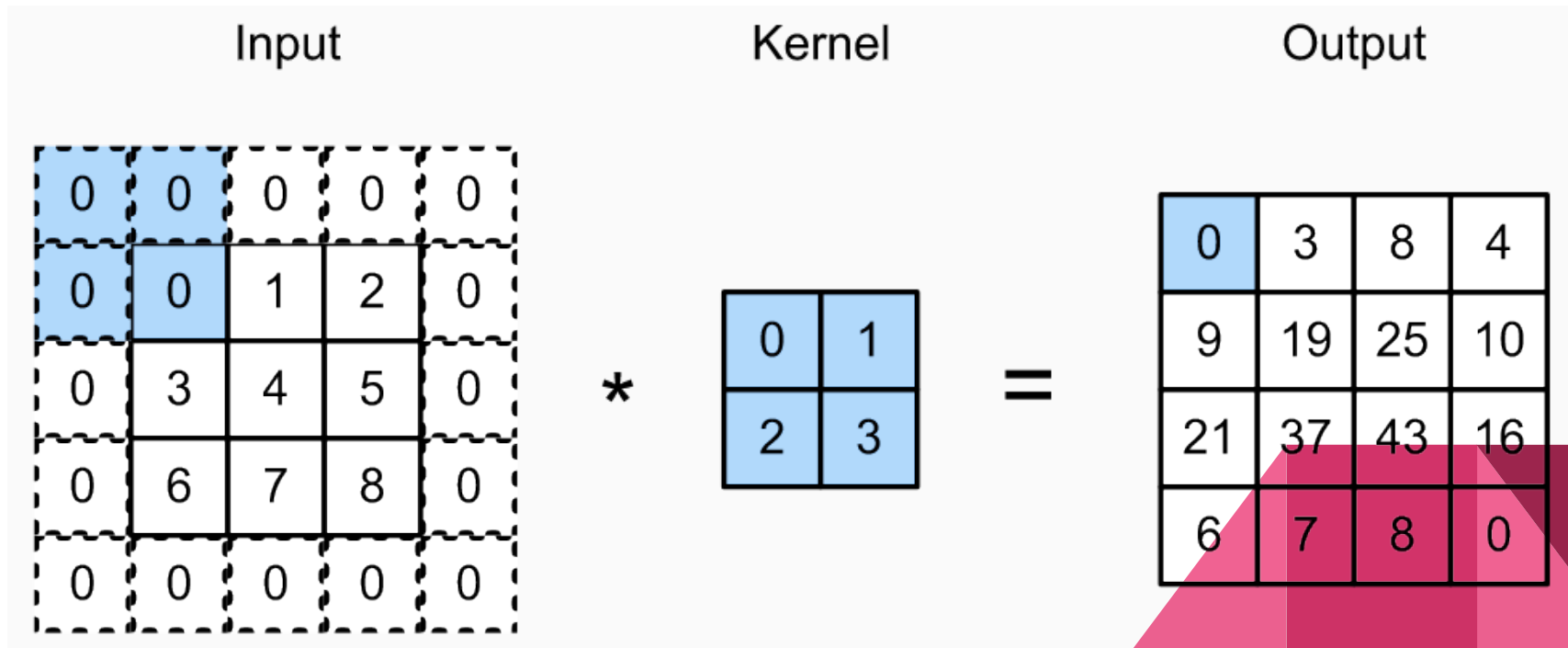
Motivação: quando aplicamos a convolução, há uma tendência a termos uma saída com dimensão consideravelmente menor do que a dimensão de entrada

- Com isso, tendemos a perder os pixels nas bordas das imagens
- Uma solução direta para este problema é adicionar pixels extras de preenchimento ao redor da borda da imagem de entrada, aumentando assim seu tamanho efetivo
- Tipicamente, adicionamos pixels de valor 0



Padding

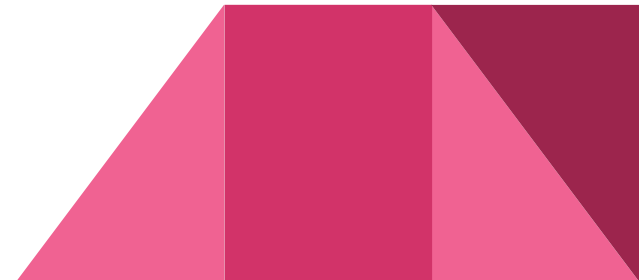
Exemplo: preenchemos uma entrada 3 x 3, aumentando seu tamanho para 5 x 5, com saída aumentada em uma dimensão 4 x 4



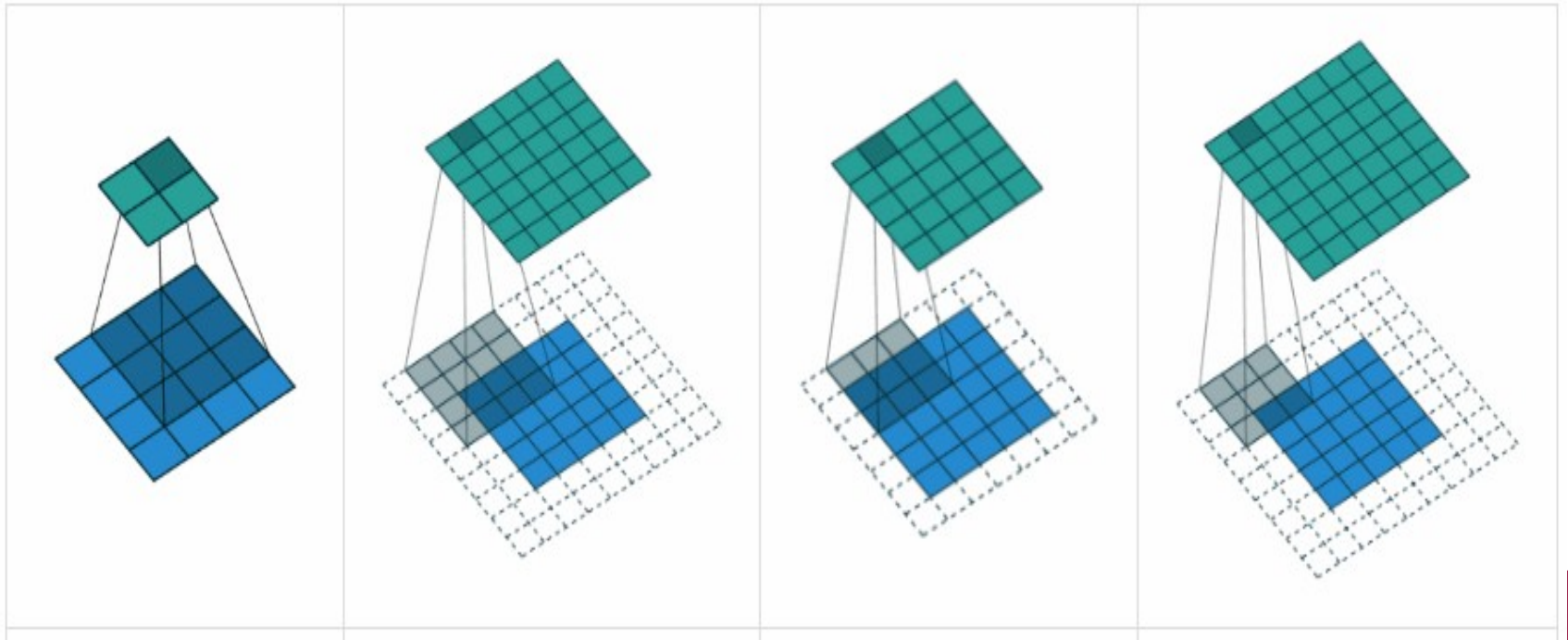
Padding

De maneira geral, são adicionadas um total de p_h linhas de padding (metade em cima e metade em baixo), e um total de p_w colunas de padding (metade na esquerda e metade na direita). A dimensão da saída será então:

- $(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$



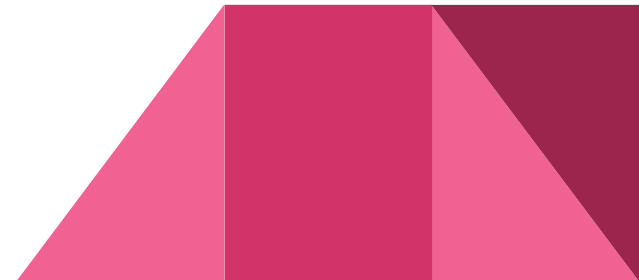
Padding



Stride

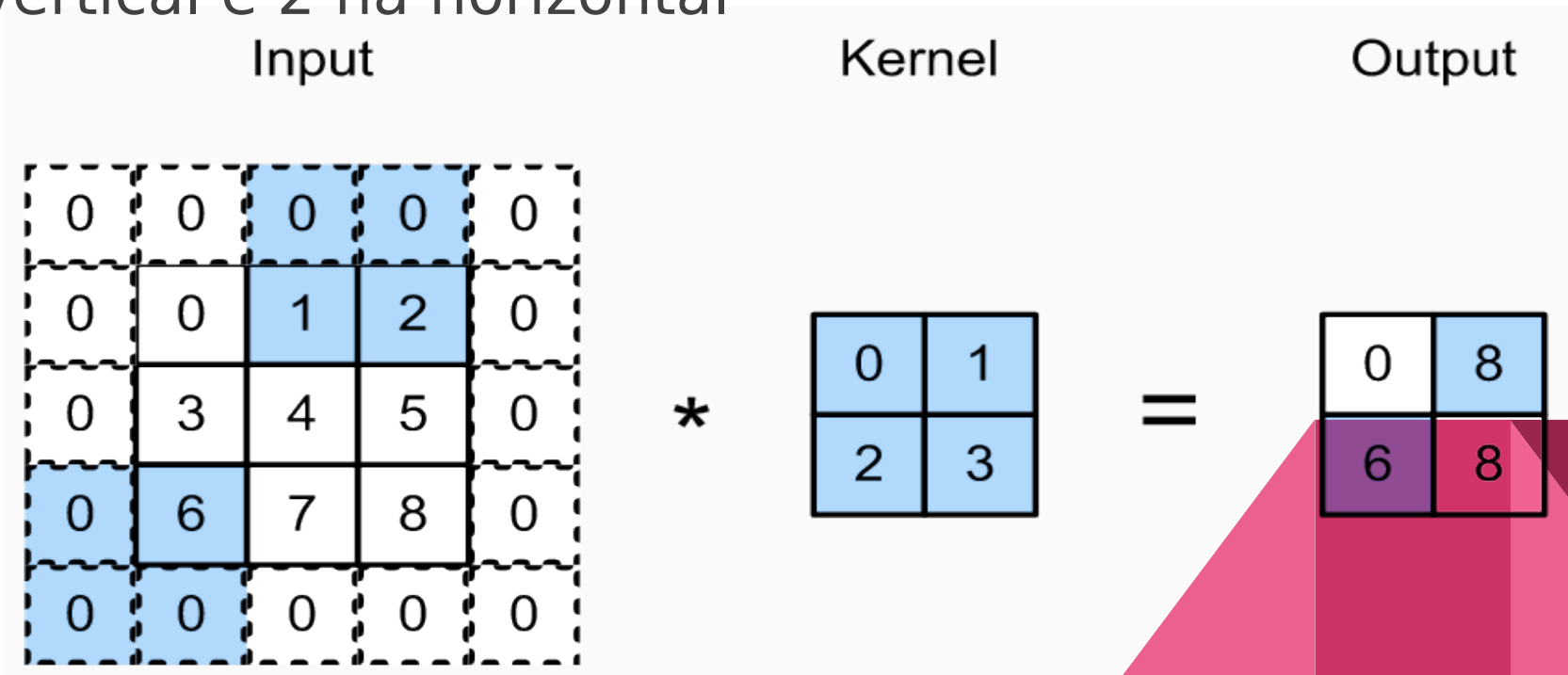
Nos referimos ao número de linhas e colunas percorridas a cada movimentação do filtro como *stride*. Até agora, usamos *stride* de tamanho 1, tanto para altura quanto para largura. Às vezes, podemos querer usar um *stride* maior.

Exemplo: *stride* 3 na vertical e 2 na horizontal



Stride

O número de linhas e colunas percorridas a cada movimentação do filtro é o *stride*. Até aqui usamos *stride* 1, mas podemos usar, por exemplo, *stride* 3 na vertical e 2 na horizontal



Stride

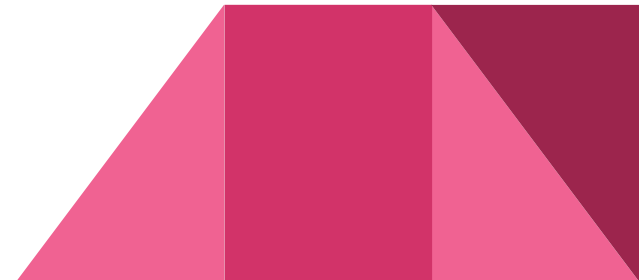
De maneira geral, quando a altura do *stride* é s_h , e a largura do *stride* é s_w , a dimensão da saída é dada por:

- $[(n_h - k_h + p_h + s_h) / s_h] \times [(n_w - k_w + p_w + s_w) / s_w]$
- Por padrão, geralmente o *padding* é igual a 0 e o *stride* é igual a 1. Na prática, geralmente não usa-se *strides* e *padding* heterogêneos. Ou seja, geralmente usa-se $p_h = p_w$ e $s_h = s_w$

Padding e Stride

O *padding* pode aumentar a altura e a largura da saída. Isso pode ser usado para dar à saída a mesma altura e largura da entrada, e ajuda a considerar as bordas das imagens.

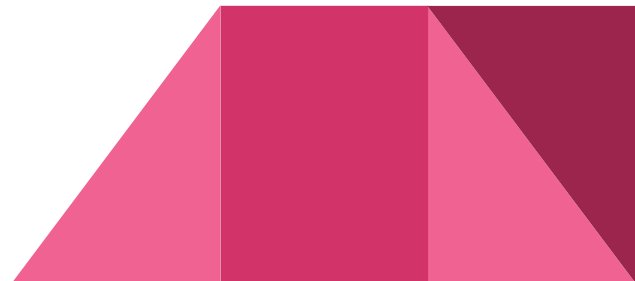
- O *stride* pode reduzir a resolução da saída
- *Padding* e *stride* podem ser usados para ajustar a dimensionalidade dos dados



Canais de Entrada e Saída Múltiplos

Até agora, simplificamos todos os nossos exemplos trabalhando com apenas uma única entrada e um único canal de saída. Isso nos permitiu pensar em nossas entradas, *kernels* de convolução e saídas como sendo bidimensionais.

- No entanto, uma imagem pode ter múltiplos canais, por exemplo 3 canais RGB para indicar as quantidades de vermelho, verde e azul.
- Nesse caso, adicionando os canais, cada imagem de entrada vai ter dimensão. Esse eixo de tamanho 3 é definido como a dimensão do canal.



Canais de Entrada e Saída Múltiplos

Convolução (3 canais)

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

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

Kernel Channel #1



308

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

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

Kernel Channel #2



-498

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25

↑
Bias = 1

Output

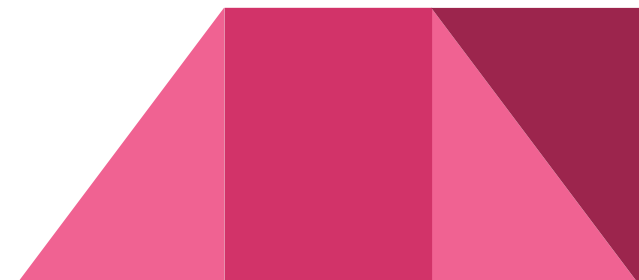
-25				...
				...
				...
				...
...

Canais de Entrada e Saída Múltiplos

Quando os dados de entrada contêm vários canais, precisamos construir um kernel de convolução com o mesmo número de canais de entrada que os dados de entrada.

- Temos um kernel $k_h \times k_w$ para cada canal de entrada. Com c_i canais, a concatenação resulta em um kernel de dimensão $c_i \times k_h \times k_w$
- É realizada a convolução entre cada kernel e matriz de entrada para cada canal

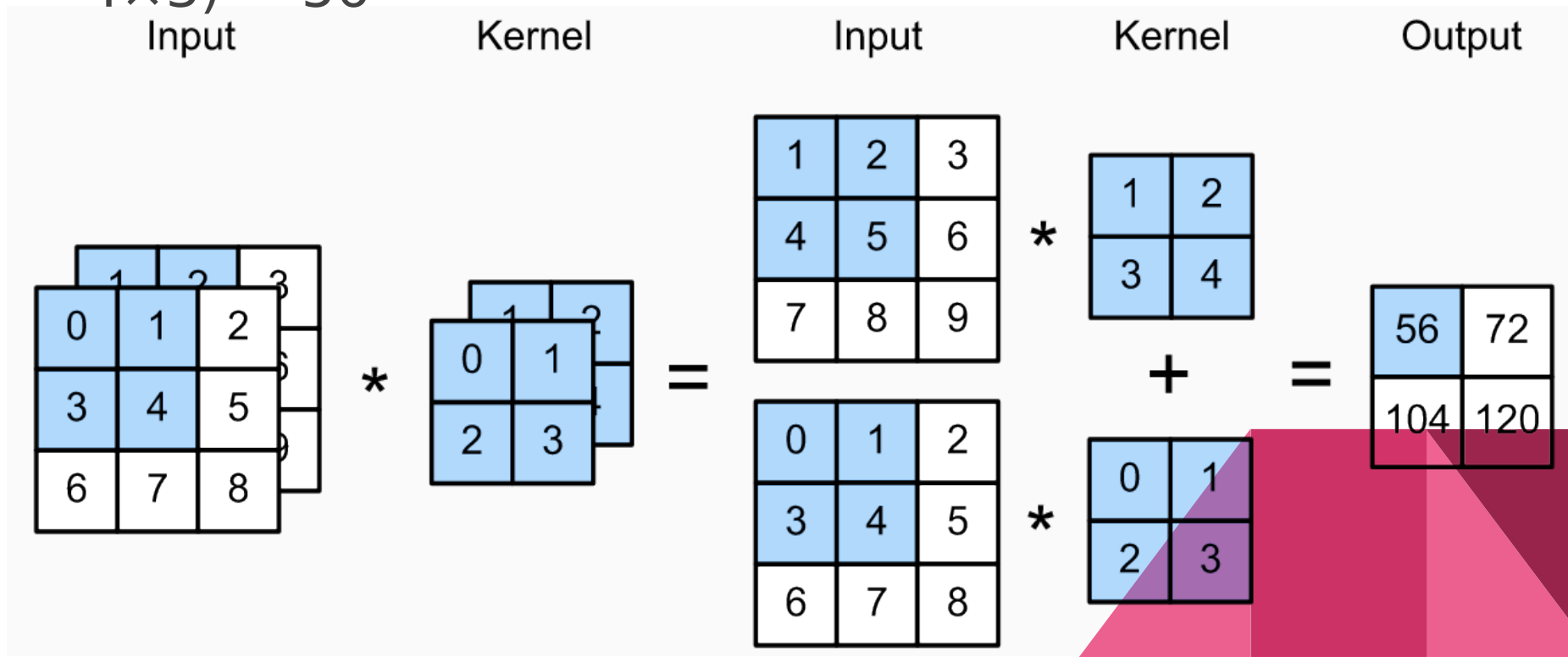
Os c_i resultados são agregados, resultando na saída bidimensional



Canais de Entrada e Saída Múltiplos

Exemplo:

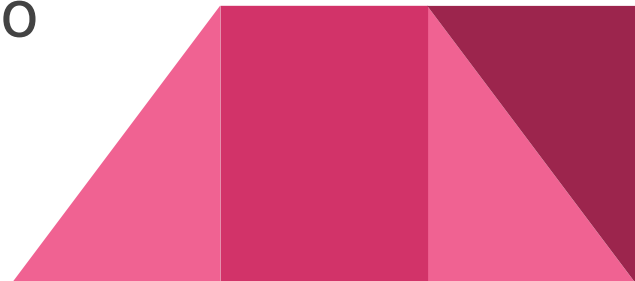
- $(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$



Canais de Entrada e Saída Múltiplos

Independentemente do número de canais de entrada, até agora sempre acabamos com um canal de saída.

- No entanto, aumentamos a dimensão do canal conforme a camada da rede neural fica mais profunda e, ao mesmo tempo, reduzimos a resolução para compensar a resolução espacial por maior profundidade de canal
- Intuitivamente, você pode pensar em cada canal como respondendo a algum conjunto diferente de atributos.

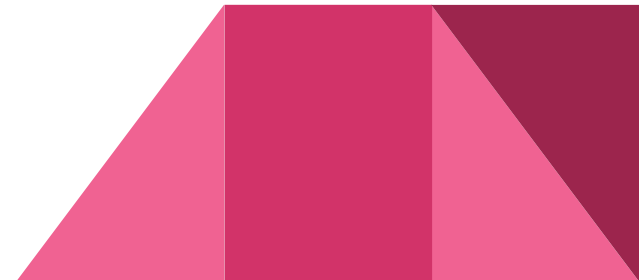


Canais de Entrada e Saída Múltiplos

Seja c_i e c_o o número de canais de entrada e saída, respectivamente, e seja k_h e k_w a altura e largura do kernel.

- Para uma saída com múltiplos canais, é necessário um kernel com dimensão $c_i \times k_h \times k_w$ para cada canal de saída
- Eles são concatenados à dimensão do canal de saída, dando origem a um kernel de convolução de dimensão $c_o \times c_i \times k_h \times k_w$

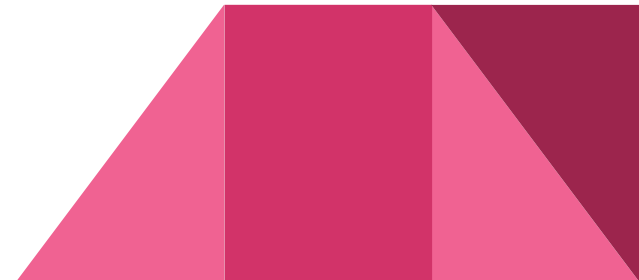
O resultado em cada canal de saída é calculado a partir do kernel de convolução correspondente a esse canal de saída, obtendo sua entrada de todos os canais da entrada



Canais de Entrada e Saída Múltiplos

Exemplo: entrada $32 \times 32 \times 3$

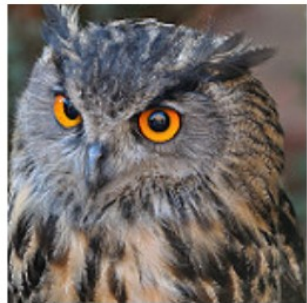
- Filtro (*kernel*) ou neurônio convolucional
 - \mathbf{w} com $k \times k \times c$, e.g. $5 \times 5 \times 3$
 - Cada neurônio realiza a convolução da entrada e gera um volume (matriz/tensor) de saída
- Se analisarmos um pixel específico, temos:
 - $\mathbf{w}^t \mathbf{x} + b$
 - Sim, temos *bias*...



Canais de Entrada e Saída Múltiplos

Mapas de ativação (ou características) são obtidos após convolução e função de ativação (e.g. ReLU);

- Empilhados formam um tensor que será a entrada da próxima camada

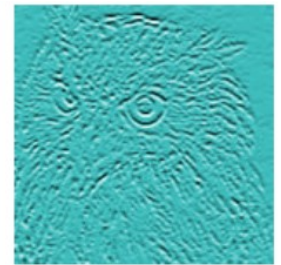


$(32 \times 32 \times 3)$

camada convolucional
10 filtros $5 \times 5 \times 3$



01



02



...

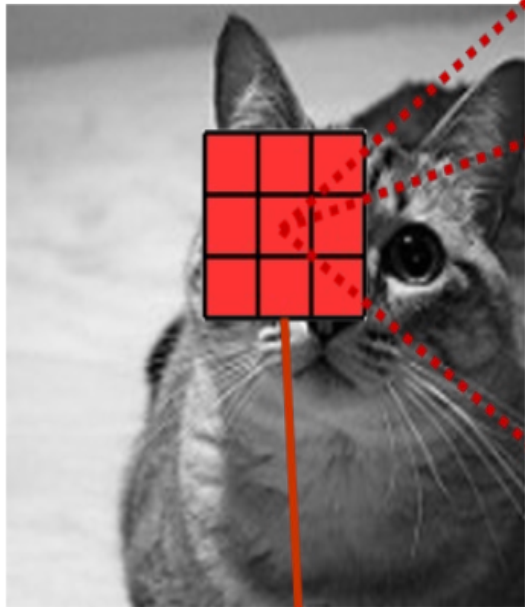
10



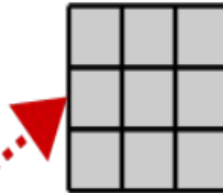
Convolução espacial
com n filtros

n feature maps

(entrada)
feature map

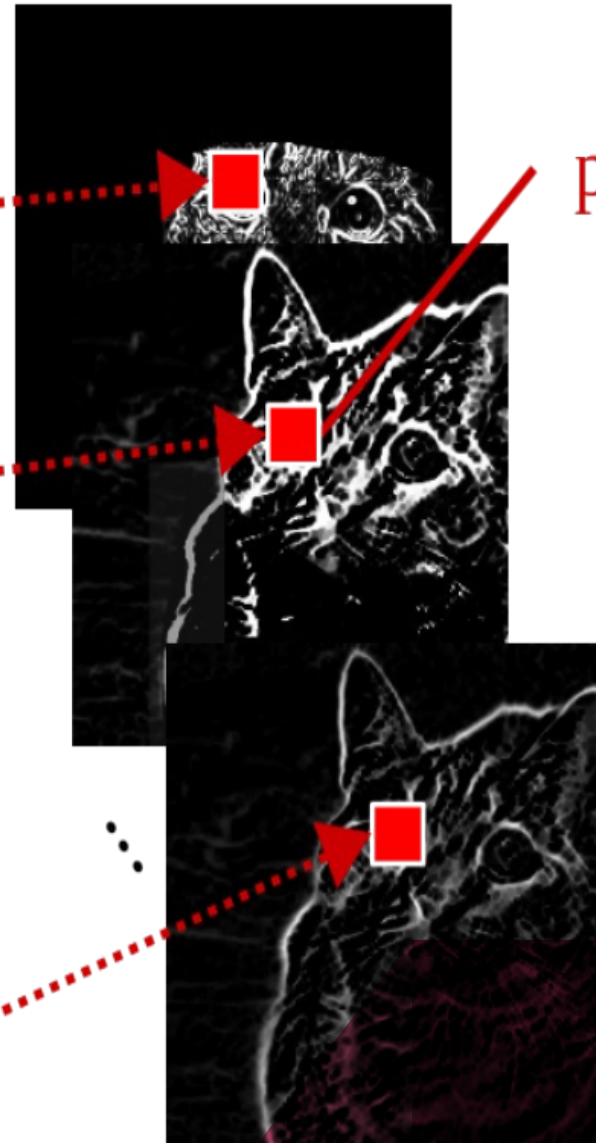
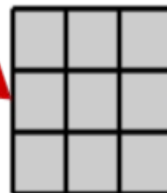


campo receptivo local

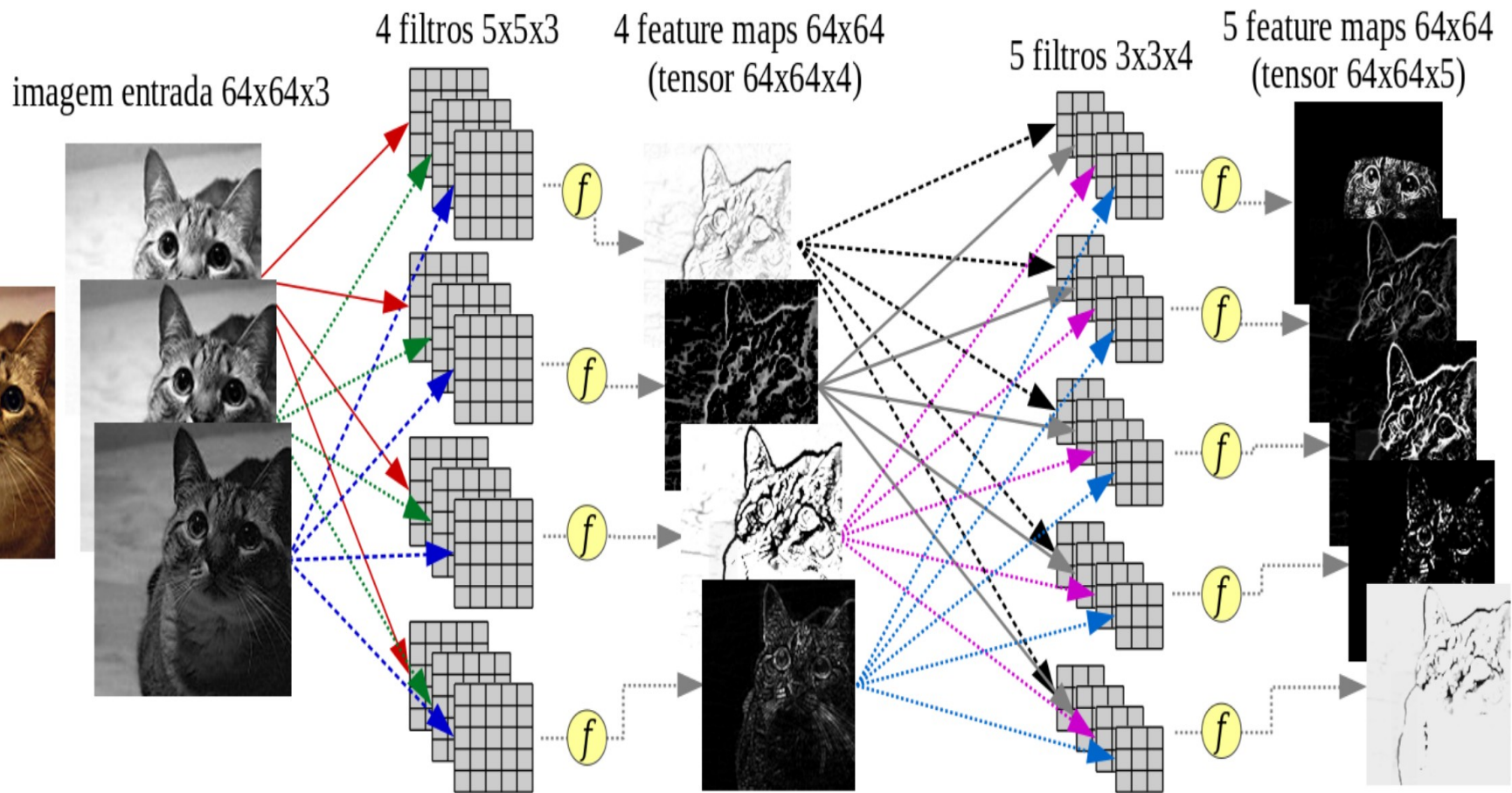


*Filtro i
com pesos w_i*

...



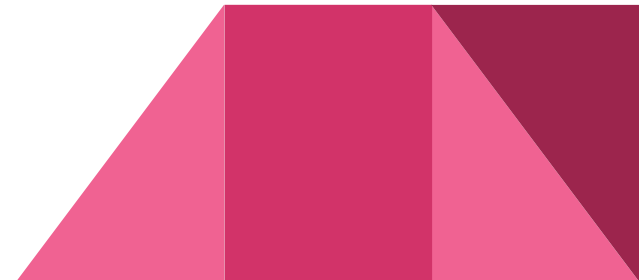
$f(i, x, y)$
pixel de
saída



Pooling

A medida que processamos imagens queremos reduzir gradualmente a resolução espacial das representações ocultas

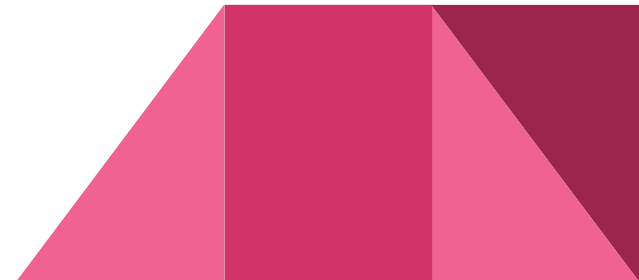
- Geralmente queremos responder a uma pergunta mais global, por exemplo, se a imagem contém um gato
- Ao agregar informações produzindo mapas cada vez menores, alcançamos esse objetivo de aprender uma representação global
- Ajuda as representações a serem mais invariantes à translação



Pooling

Pooling consiste em aplicar uma janela de formato fixo deslizada sobre todas as regiões na entrada de acordo o *stride*, computando uma única saída

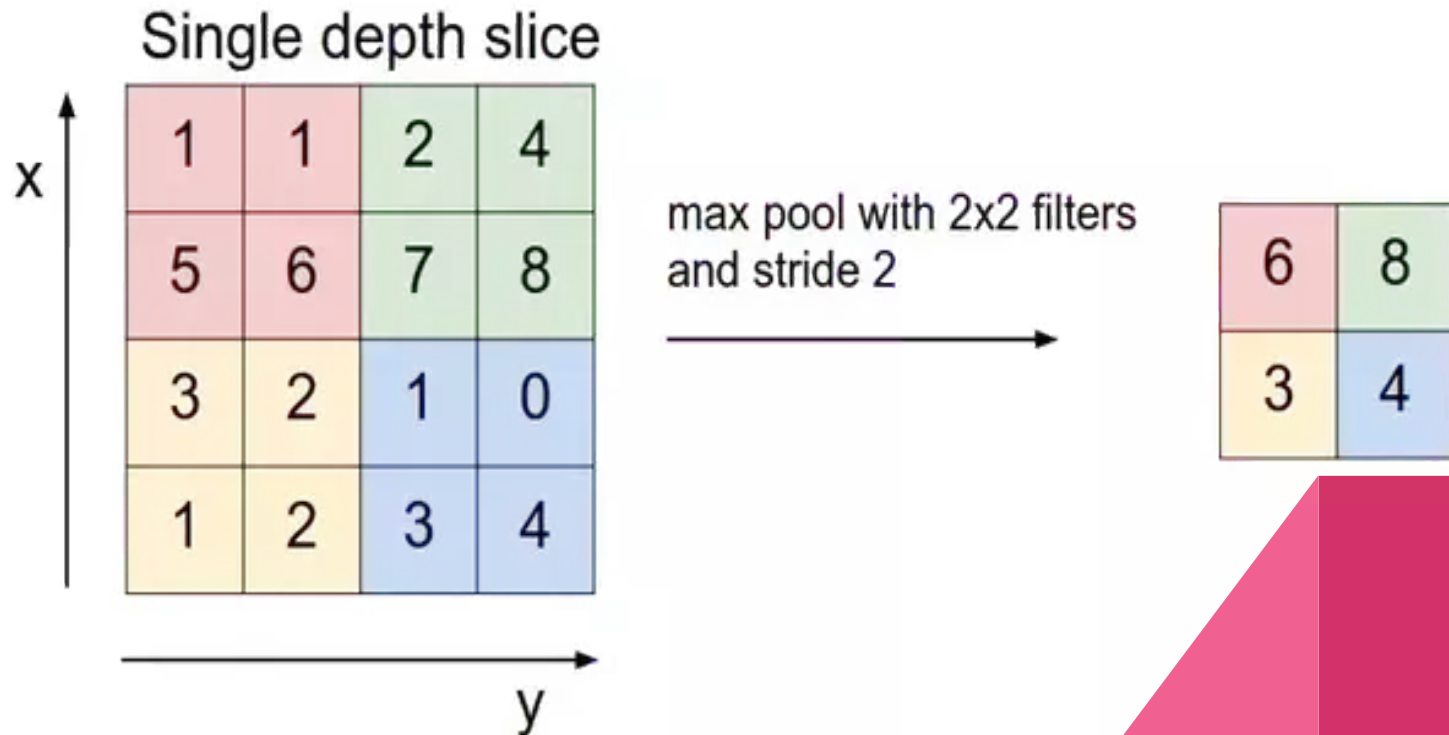
- A camada de *pooling* não contém parâmetros (não há *kernel*)
- Determinísticos, normalmente calculando o valor máximo ou médio dos elementos na janela



Pooling

Pooling reduz a informação (dimensionalidade) agregando valores

Exemplo: *Max pooling*



Convolutional Neural Networks (CNN)

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



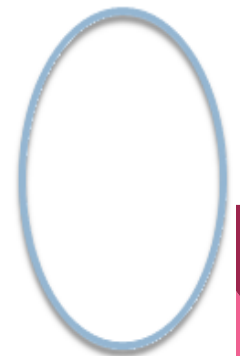
1	-1	1
-1	1	-1
1	-1	1



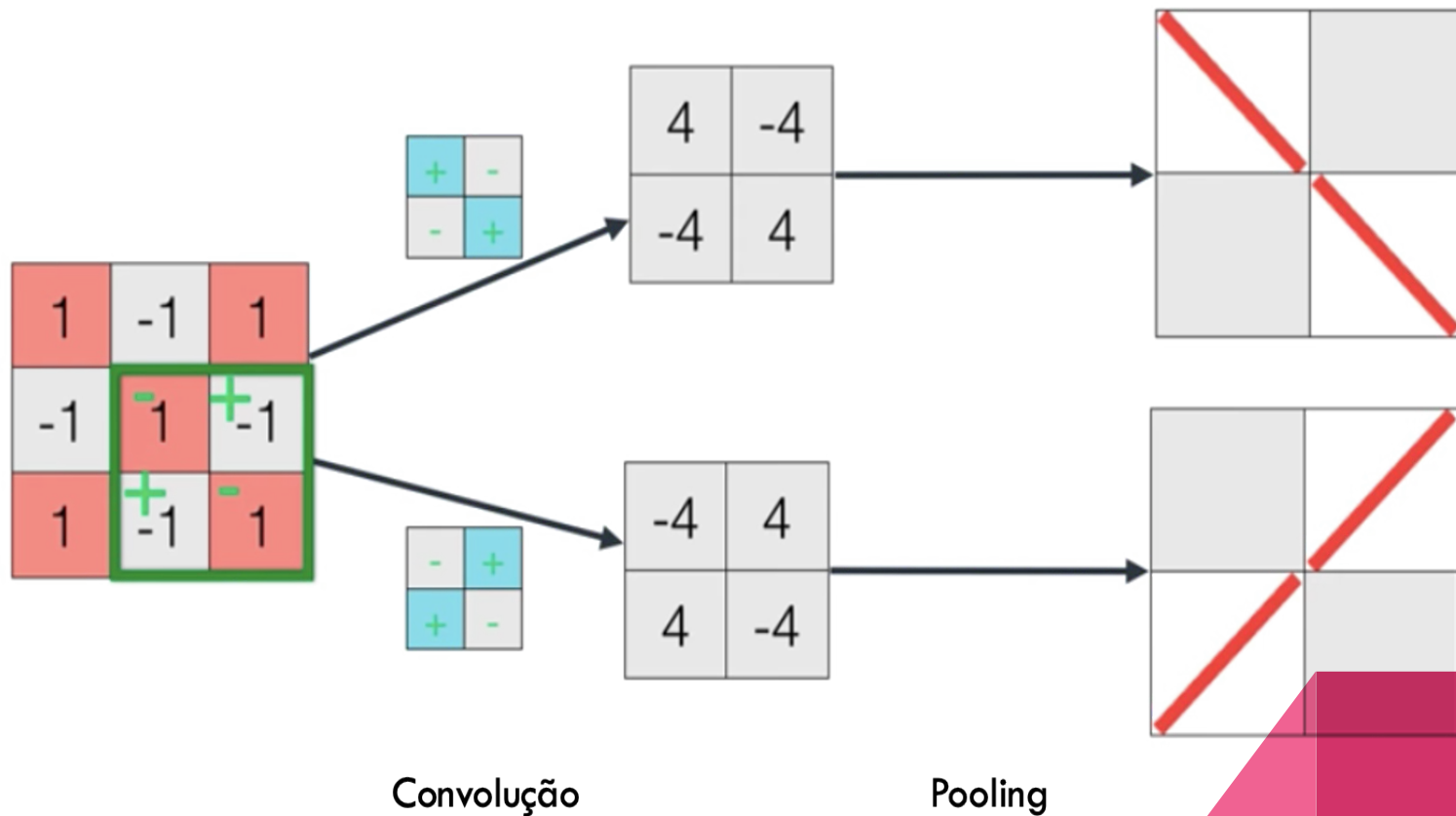
-1	-1	1
-1	1	-1
1	-1	-1



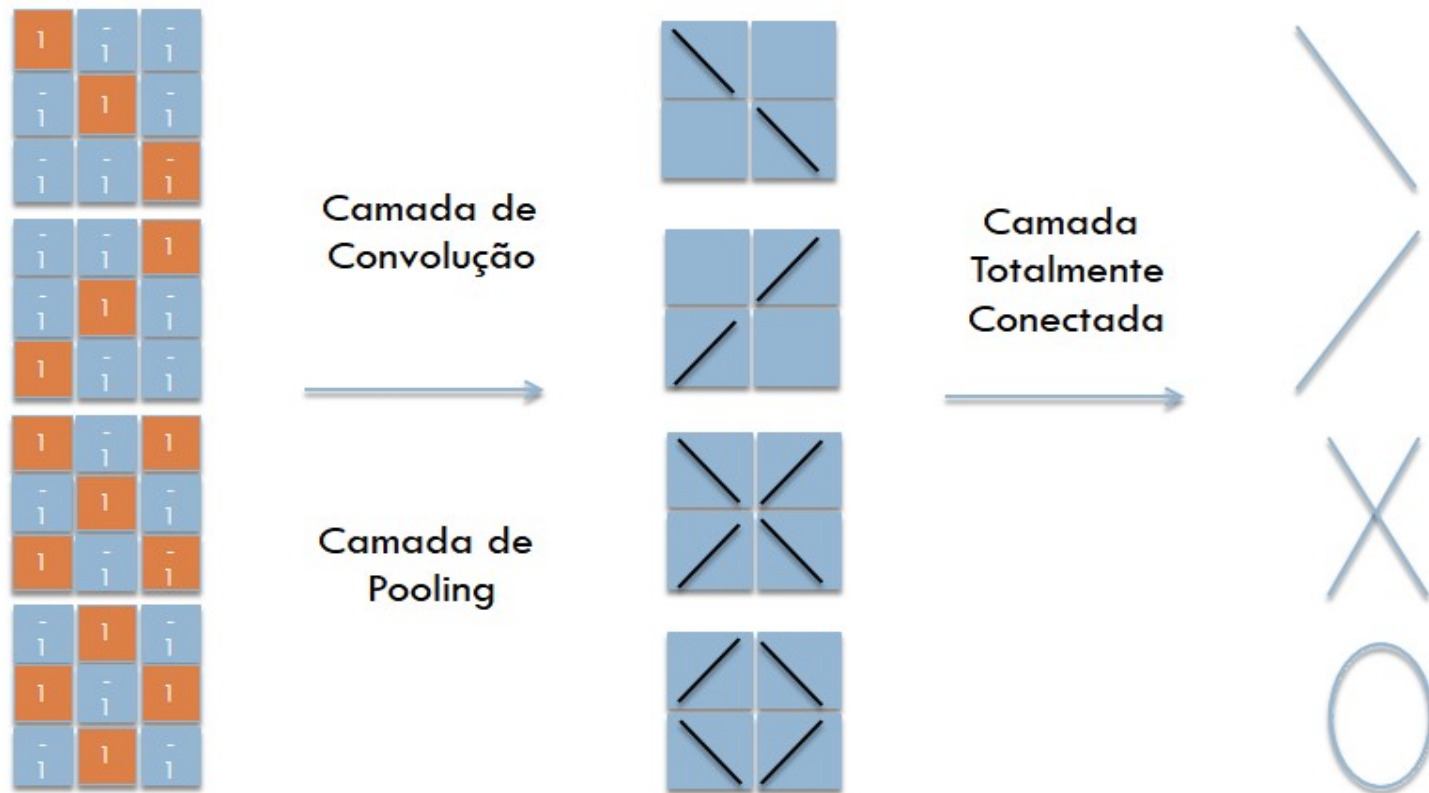
-1	1	-1
1	-1	1
-1	1	-1



Convolutional Neural Networks (CNN)



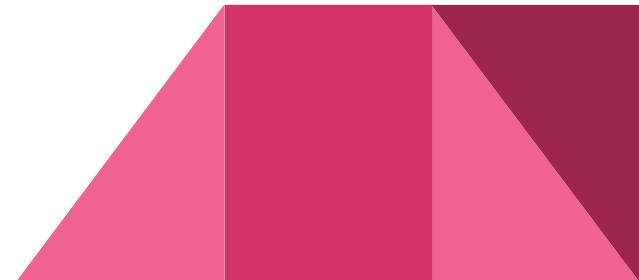
Convolutional Neural Networks (CNN)



Montando uma Rede Neural Convolucional

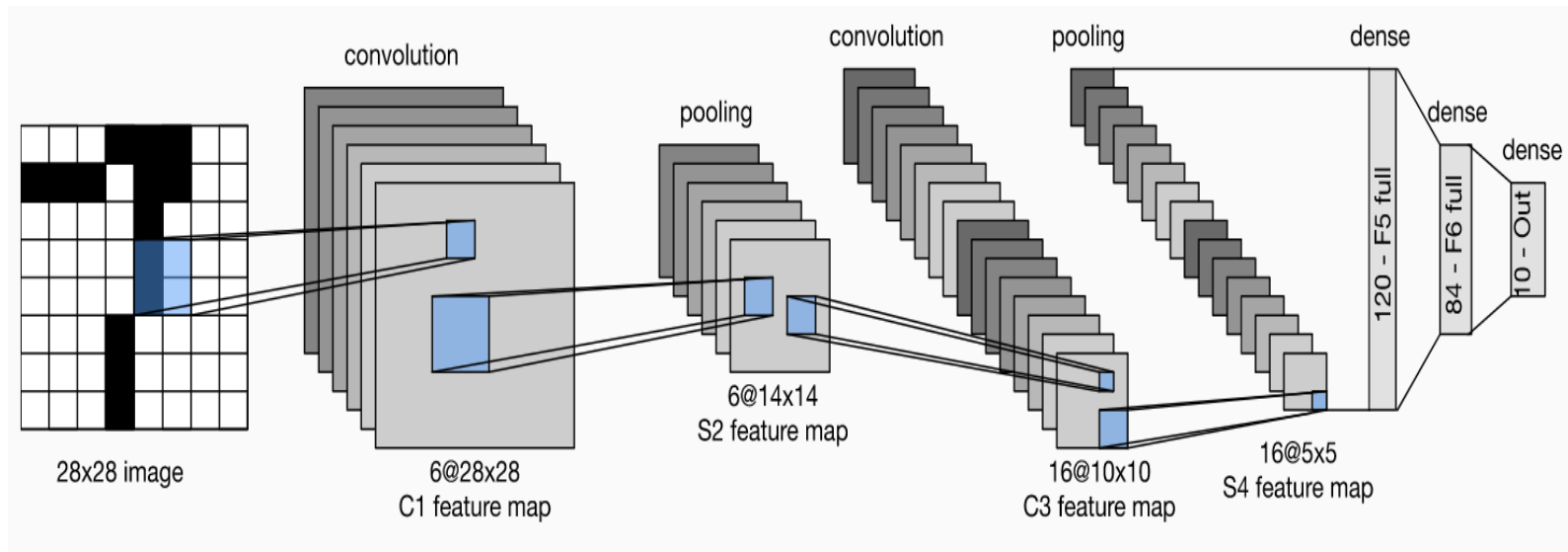
Vamos apresentar aqui a LeNet, uma das primeiras CNNs publicadas para tarefas de visão computacional

- O modelo foi introduzido por Yann LeCun, então pesquisador da AT&T Bell Labs, com o objetivo de reconhecer dígitos manuscritos em imagens (LeCun et al., 1998).
- Na época, a LeNet alcançou resultados excelentes comparados ao desempenho das SVMs, então uma abordagem dominante no aprendizado supervisionado



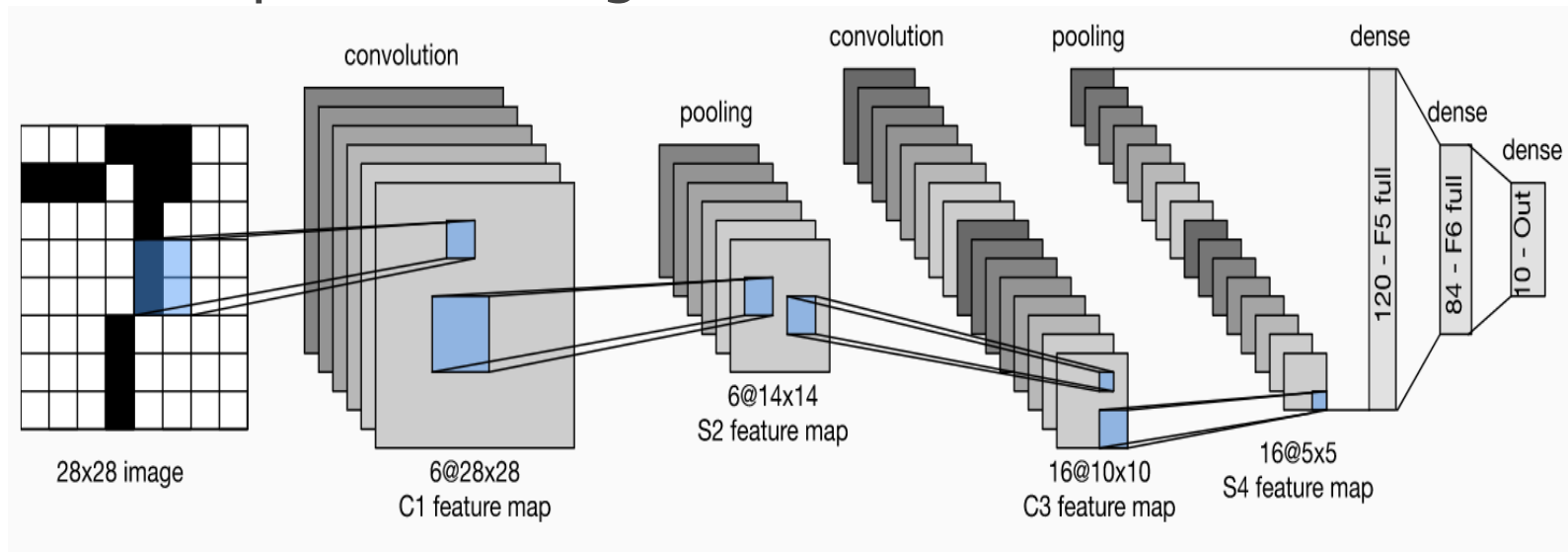
Montando uma Rede Neural Convolucional

De maneira geral, a LeNet (LeNet-5) possui duas partes: i) um codificador convolucional, consistindo de duas camadas de convolução; e ii) um bloco denso, consistindo de três camadas completamente conectadas



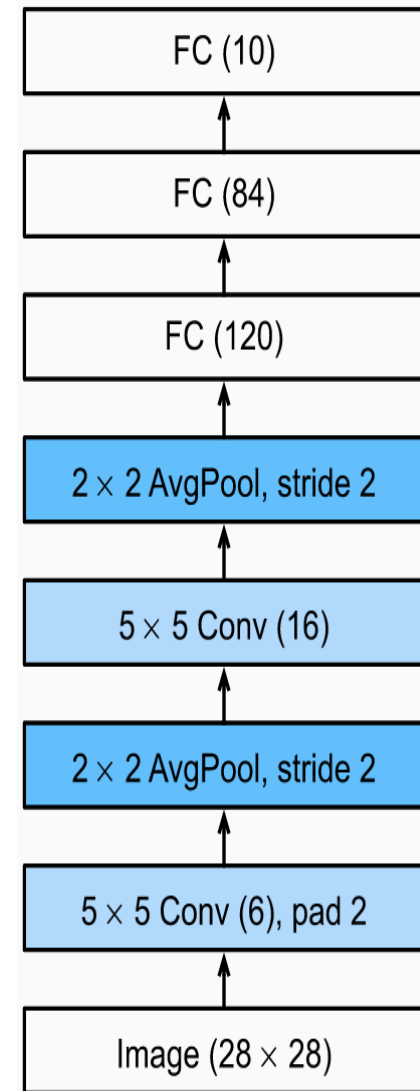
Montando uma Rede Neural Convolucional

Uma camada de convolução com um kernel 5x5, com função de ativação *sigmoidal*, seguida de uma operação de *average pooling* (2x2) com *stride* 2. A primeira camada convolucional tem 6 canais de saída, enquanto a segunda tem 16.



Passando pela rede uma única imagem (28x28 pixel) de 1 canal (branco e preto), podemos verificar a dimensão da saída de cada camada

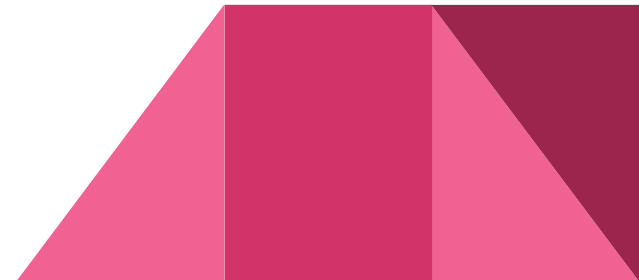
- A primeira camada convolucional usa 2 pixels de *padding* para compensar a redução na altura e largura que, de outra forma, resultaria do uso de um *kernel* 5x5
- A segunda camada convolucional dispensa o *padding* e, portanto, a altura e a largura são reduzidas em 4 pixels
- Veja que conforme a profundidade aumenta, o número de canais aumenta, indo de 1 até 16, porém a altura e largura são diminuídas pelo pooling
- Finalmente as camadas totalmente conectadas reduzem a dimensionalidade até obter um número de saídas que corresponde ao número de classes



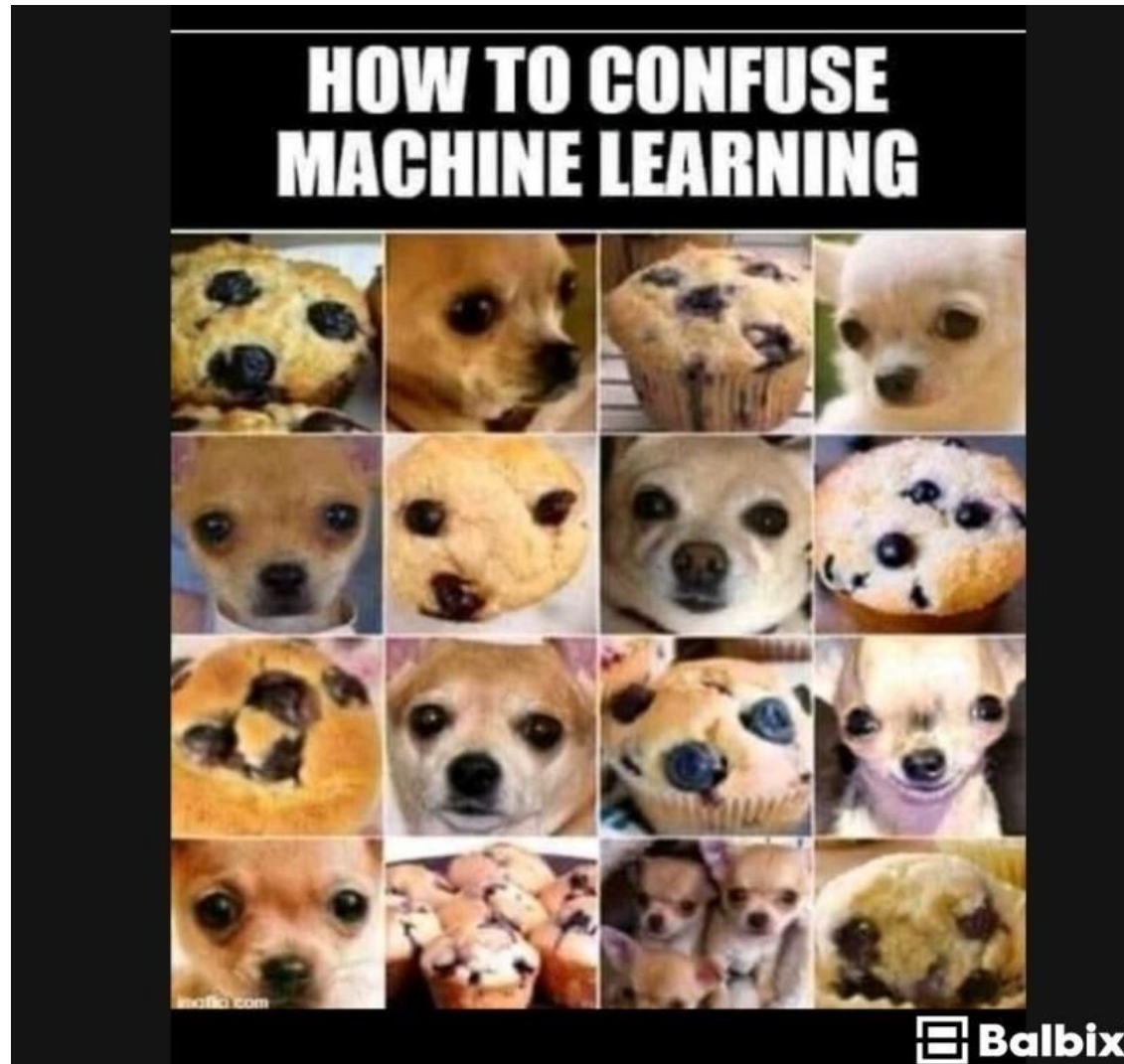
Convolutional Neural Networks (CNN)

Outros modelos:

- AlexNet (2012)
- VGG 19 (2014)
- GoogLeNet (2014)
- ResNet 34 (2015)
- Densenet (2016)
- Se Net (2018)
- CBAM (2018)
- Channel Boosting (2018)
- NASNet (2018)
- ShuffleNet (2018)
- EfficientNet (2019)
- Trabalhos mais recentes que melhoram essas arquiteturas



Agora já sabemos criar modelos para classificar isso



ou não!