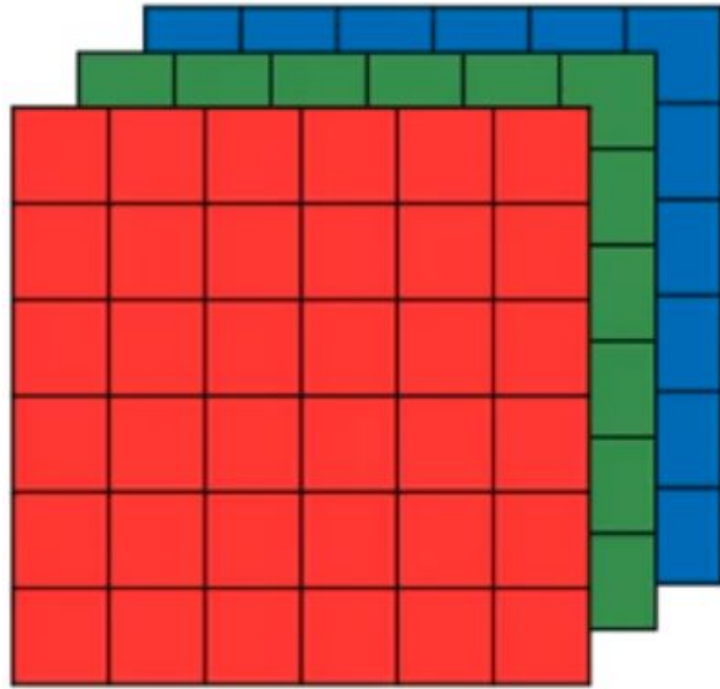


# *Convolutional Neural Networks*

---

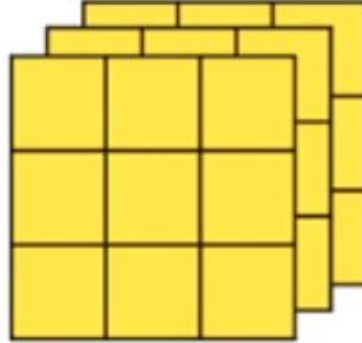
Uma camada de  
uma CNN

# Exemplo de uma camada



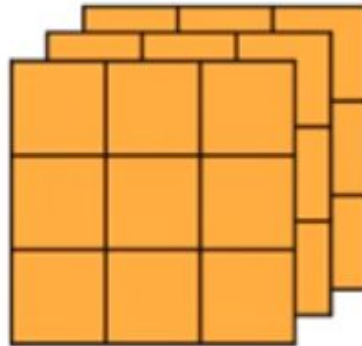
$6 \times 6 \times 3$

\*



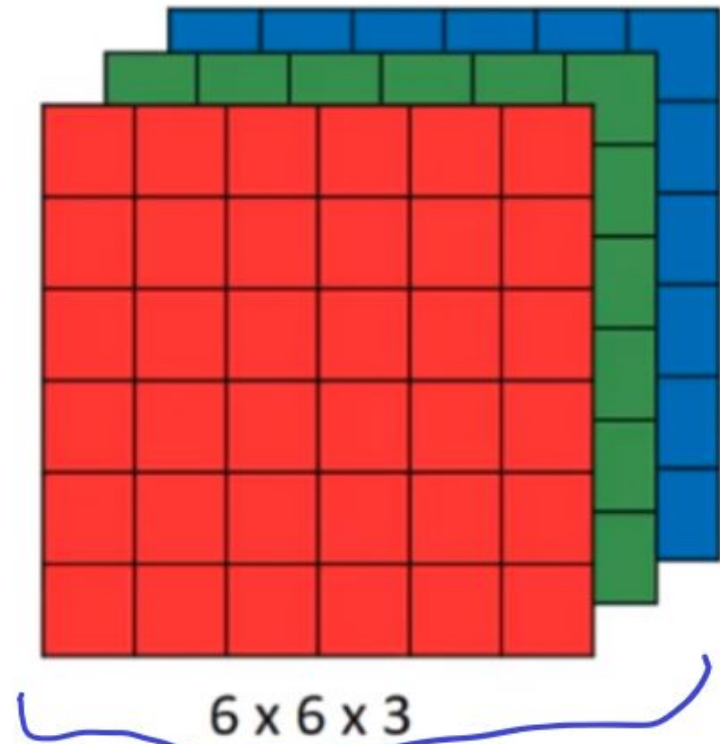
$3 \times 3 \times 3$

\*

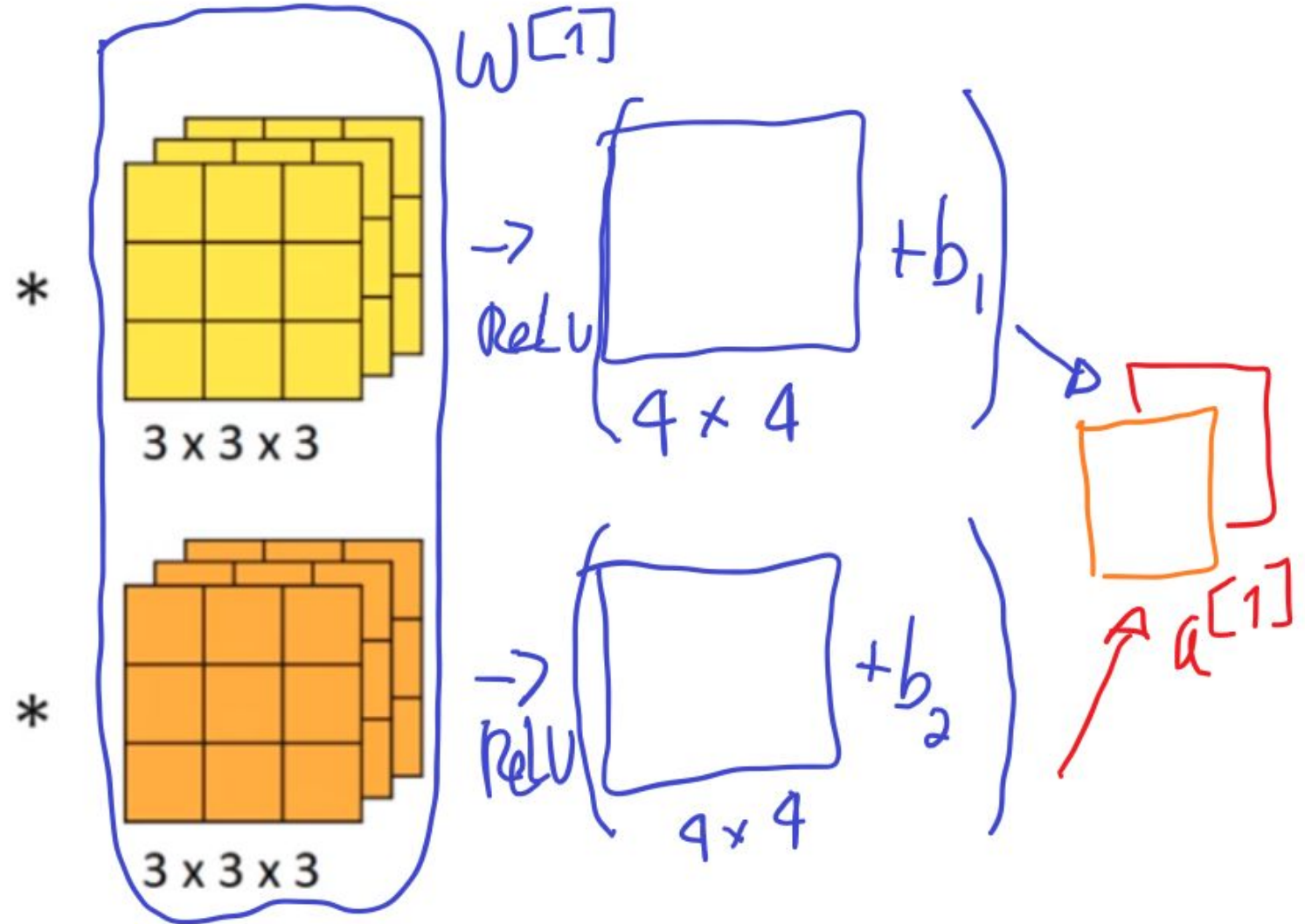


$3 \times 3 \times 3$

# Exemplo de uma camada



$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$



# Número de parâmetros em uma camada

Se você tem 10 filtros  $3 \times 3 \times 3$  em uma camada de uma rede neural, quantos parâmetros essa camada tem?

# Sumário da notação

Se a camada  $l$  é uma camada convolucional:

$f^{[l]}$  = tamanho do filtro

Entrada:

$p^{[l]}$  = padding

Saída:

$s^{[l]}$  = stride

$n_c^{[l]}$  = número de filtros

Cada filtro é:

Ativações:

Pesos:

Bias:

# Sumário da notação

camada  $l$

Se a camada  $l$  é uma camada convolucional:

$f^{[l]}$  = tamanho do filtro

$p^{[l]}$  = padding

$s^{[l]}$  = stride

$n_c^{[l]}$  = número de filtros

Cada filtro é:  $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Ativações:  $a^{[l]} \Rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Pesos:  $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

Bias:  $n_c^{[l]} \rightarrow (1, 1, 1, n_c^{[l]})$

Entrada:  $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Saída:  $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]} + 1}{s^{[l]}}$$

matriz de ativ. p/  $m$   
exemplos de treino

$$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

# Uma camada de uma CNN

- Primeiro, convolvemos alguns filtros para uma determinada entrada
- Em seguida, adicionamos um viés a cada saída do filtro
- Por fim, obtemos o RELU do resultado

# Uma camada de uma CNN

- Exemplo:

Imagem de entrada: **6x6x3** #  $a_0$

10 filtros: **3x3x3** #  $W_1$

Resultado: **4x4x10** #  $W_1 a_0$

Adicionar **b** (viés) com **10x1** nos levará: imagem **4x4x10** #  $W_1 a_0 + b$

Aplicar **RELU** nos levará: imagem **4x4x10** #  $A_1 = \text{RELU}(W_1 a_0 + b)$

No último resultado **p = 0, s = 1**

O número de parâmetros aqui é: **(3x3x3x10) + 10 = 280**

- **Dica:** não importa o tamanho da entrada, o número de parâmetros é o mesmo se o tamanho do filtro for o mesmo
  - Isso torna menos propenso a overfitting



# Se a camada **1** for uma camada conv

$f[l]$  = filter size

$p[l]$  = padding # Default is zero

$s[l]$  = stride

$nc[l]$  = number of filters

Input:  $n[l-1] \times n[l-1] \times nc[l-1]$       **Or**       $nH[l-1] \times nW[l-1] \times nc[l-1]$

Output:  $n[l] \times n[l] \times nc[l]$       **Or**       $nH[l] \times nW[l] \times nc[l]$

Where  $n[l] = (n[l-1] + 2p[l] - f[l] / s[l]) + 1$

Each filter is:  $f[l] \times f[l] \times nc[l-1]$

Activations:  $a[l]$  is  $nH[l] \times nW[l] \times nc[l]$

$A[l]$  is  $m \times nH[l] \times nW[l] \times nc[l]$  # In batch or minibatch training

Weights:  $f[l] * f[l] * nc[l-1] * nc[l]$

bias: (1, 1, 1,  $nc[l]$ )

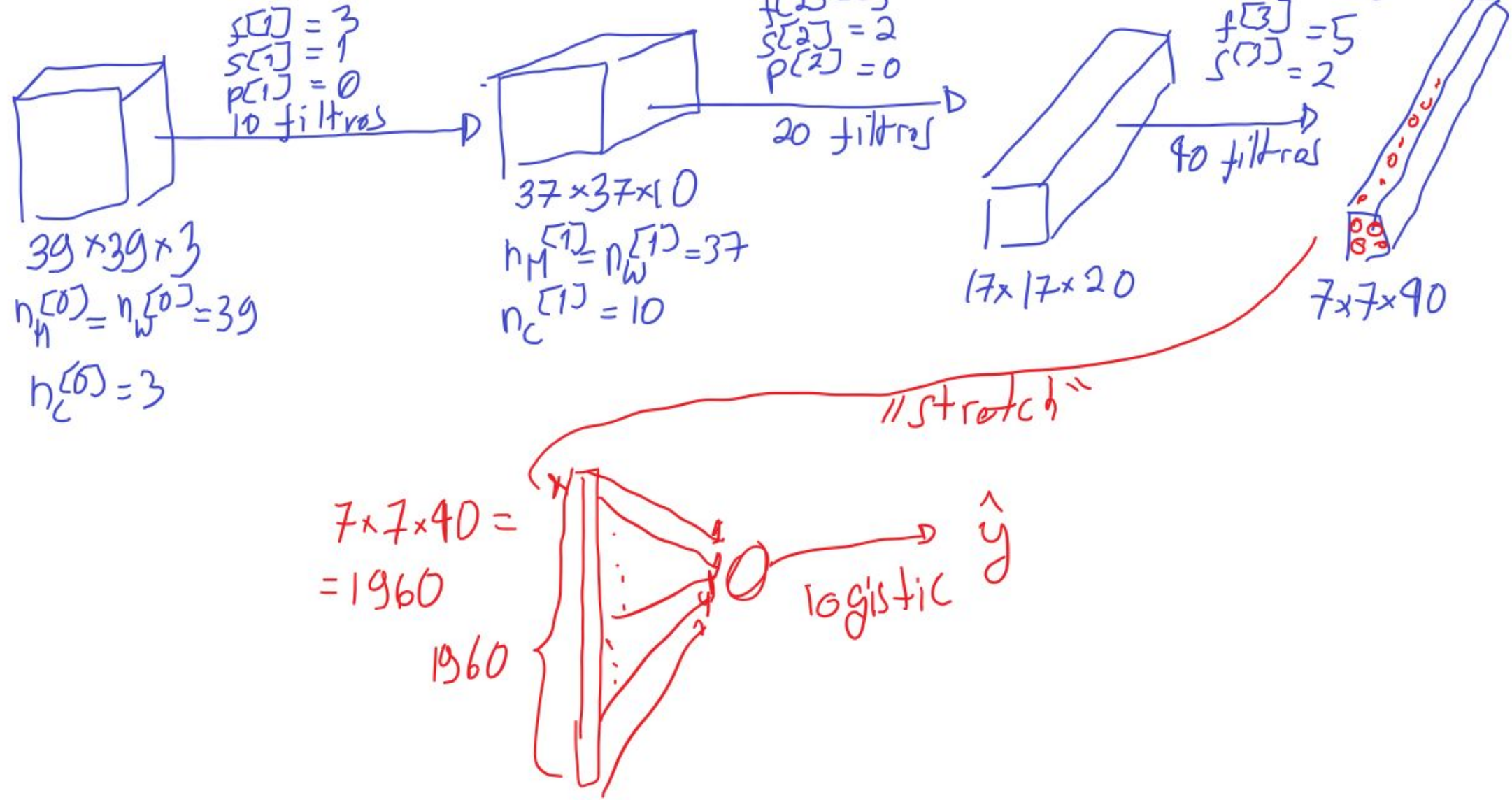
# *Convolutional Neural Networks*

---

Um exemplo simples  
de CNN

# Exemplo de ConvNet

# Exemplo de ConvNet



# Um grande exemplo:

- Entrada: imagens são  **$a_0 = 39 \times 39 \times 3$** 
  - **$n_0 = 39$  e  $nc_0 = 3$**
- Primeira camada (camada **Conv**):
  - **$f_1 = 3$ ,  $s_1 = 1$ , e  $p_1 = 0$**
  - número de filtros = **10**
  - Então, as saídas são:  **$a_1 = 37 \times 37 \times 10$** 
    - **$n_1 = 37$  e  $nc_1 = 10$**
- Segunda camada (camada **Conv**):
  - **$f_2 = 5$ ,  $s_2 = 2$ ,  $p_2 = 0$**
  - número de filtros = **20**
  - As saídas são  **$a_2 = 17 \times 17 \times 20$** 
    - **$n_2 = 17$ ,  $nc_2 = 20$**
  - **Dica:** o encolhimento é muito mais rápido, pois o *stride* é **2**
- Terceira camada (camada **Conv**):
  - **$f_3 = 5$ ,  $s_3 = 2$ ,  $p_3 = 0$**
  - número de filtros = **40**
  - As saídas são  **$a_3 = 7 \times 7 \times 40$** 
    - **$n_3 = 7$ ,  $nc_3 = 40$**
- Quarta camada (Softmax todo conectado)
  - **$a_3 = 7 \times 7 \times 40 = 1960$**  como um vetor
- Imagens estão ficando cada vez menor após cada camada.

# Tipos de camadas em uma rede convolucional:

- Convolucional (Conv)
- Pooling (Pool)
- Toda conectada (*Fully connected*)

# *Convolutional Neural Networks*

---

Camadas de *pooling*

# Camada de *pooling*: *Max pooling*

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




# Camada de *pooling*: *Max pooling*

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


# Camada de *pooling*: *Max pooling*

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


# Camada de *pooling*: *Average pooling*

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2




# Camadas de *pooling*

- Além das camadas **conv**, as CNNs geralmente usam camadas de pooling para
  - reduzir o tamanho das entradas
  - acelerar a computação
  - tornar algumas das *features* que ele detecta **mais robustas**

# Max *pooling*

- O ***max pooling*** está fazendo o seguinte: se *a feature* for detectada **em qualquer lugar** desse filtro, mantenha um número **alto**
- A principal razão pela qual as pessoas o estão usando é porque ele funciona bem na prática e reduz os cálculos
- Não tem parâmetros para aprender

# Max *pooling*

- Exemplo de max *pooling* em entrada 3D:
- Entrada: 4x4x10
- Tamanho do max *pooling* = 2 e *stride* = 2
- Saída: 2x2x10

# *Average pooling*

- O *average pooling* está tirando as médias dos valores em vez de obter os valores máximos
- O *max pooling* é usado com mais frequência do que o *average pooling* na prática
- Se o *stride* do *pooling* for igual ao tamanho, então ele aplicará o efeito de encolhimento

# Sumário de *pooling*

Hiperparâmetros:

f : tamanho do filtro

s : *stride*

*Max* ou *average pooling*



# *Convolutional Neural Networks*

---

Outro exemplo de  
CNN

# Exemplo de CNN

# Exemplo de CNN

[illegible]

# Exemplo de CNN

- Hiperparâmetros são **muitos**
- Para **escolher** o valor de cada um, você deve seguir a diretriz que discutiremos mais tarde ou verificar a literatura e tirar algumas ideias e números dela
- Normalmente, o tamanho da entrada **diminui** em camadas, enquanto o número de filtros **aumenta**
- Uma CNN geralmente consiste em **uma ou mais convoluções** (não apenas uma como os exemplos mostrados) seguidas por um ***pooling***
- Camadas totalmente conectadas têm a **maioria dos parâmetros**
- Para considerar o uso desses blocos juntos, você deve **procurar outros exemplos** de trabalho para obter algumas intuições

# *Convolutional Neural Networks*

---

Por que  
convoluções?

# Por que convoluções?

# Por que convoluções?

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

 $*$ 

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

 $=$ 

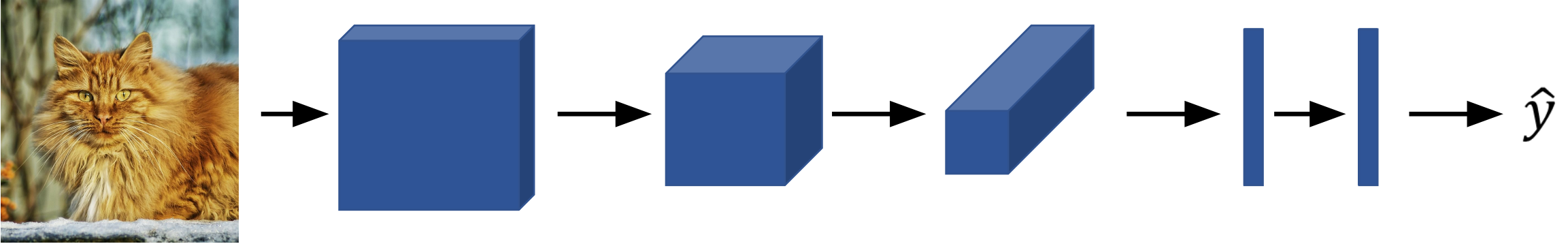
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

**Compartilhamento de parâmetros:** um detector de *features* (como um detector de borda vertical) útil em uma parte da imagem é provavelmente útil em outra parte

**Esparsidade de conexões:** em cada camada, cada valor de saída depende apenas de um pequeno número de entradas

# Colocando tudo junto

Training set  $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ .



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use o gradiente descendente para otimizar os parâmetros que reduzem  $J$



# Para brincar

## [ConvNetJS](#) CIFAR-10 demo

### Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

# Links úteis

<https://towardsdatascience.com/backpropagation-in-a-convolutional-layer-24c8d64d8509>