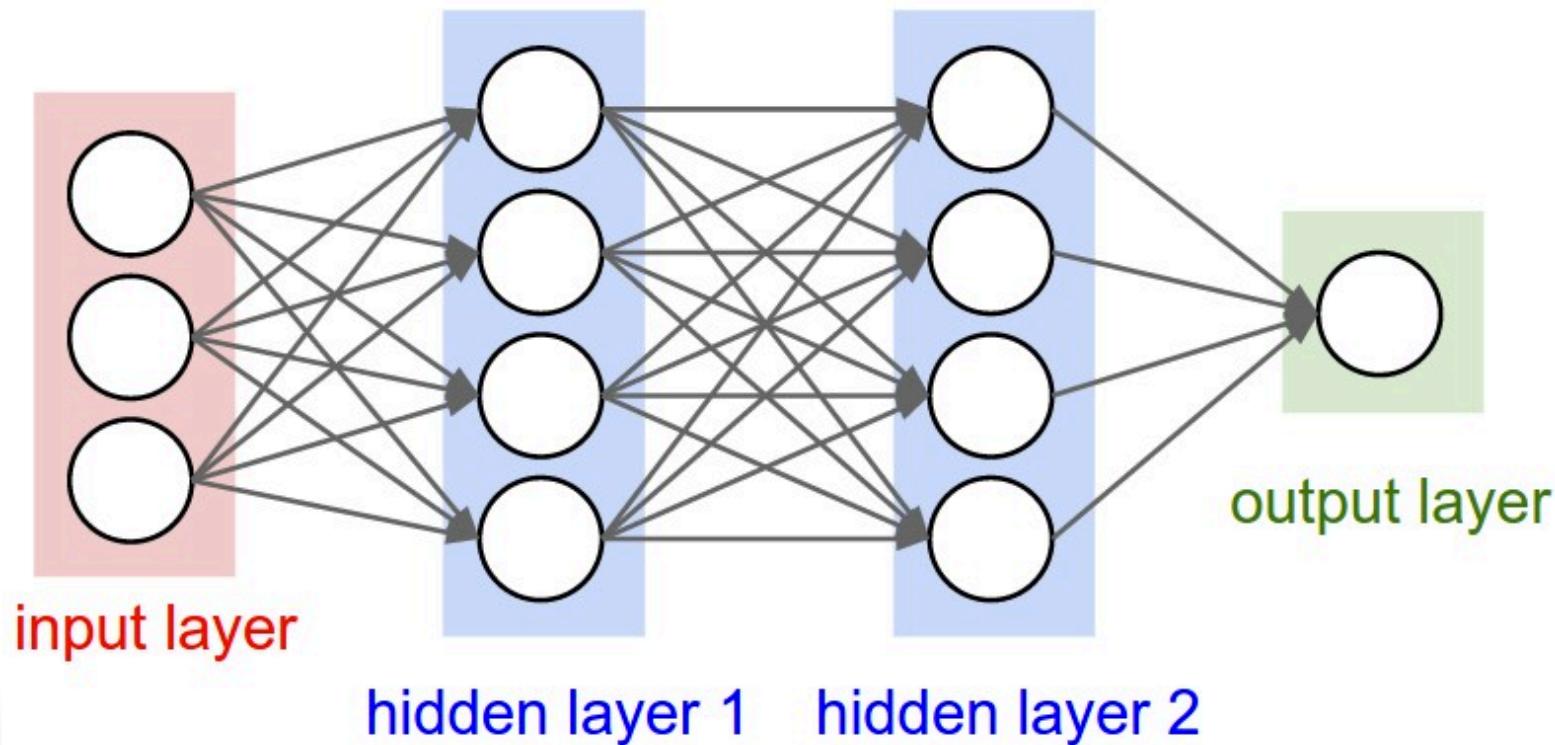


# Convolutional Neural Networks - CNN

Visão Computacional

# Motivação

- Imagine uma MLP
  - Múltiplas camadas



# Motivação

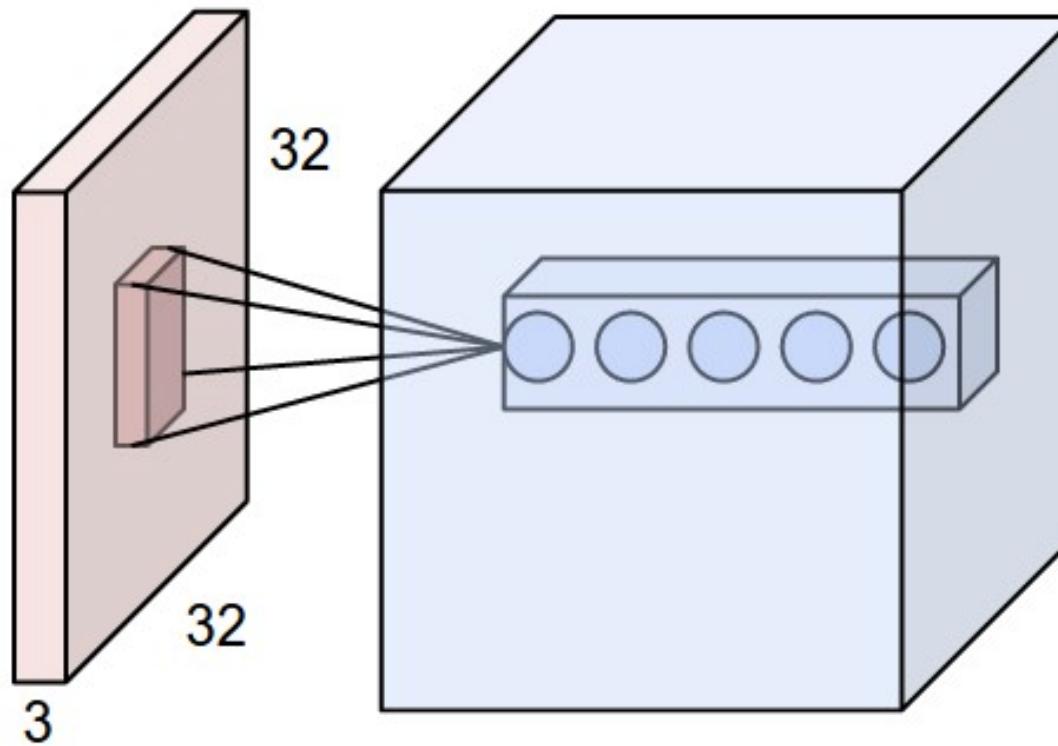
- Conecte como entrada uma **imagem** de 32x32x3 (RGB)
- Quantos pesos necessitam ser ajustados nessa rede?
  - 3072 (só na camada de entrada)
  - A coisa piora quando a imagem cresce
- **Pense sobre isso....**
  - **quais as implicações? (tempo, generalização ...)**

# Ideia

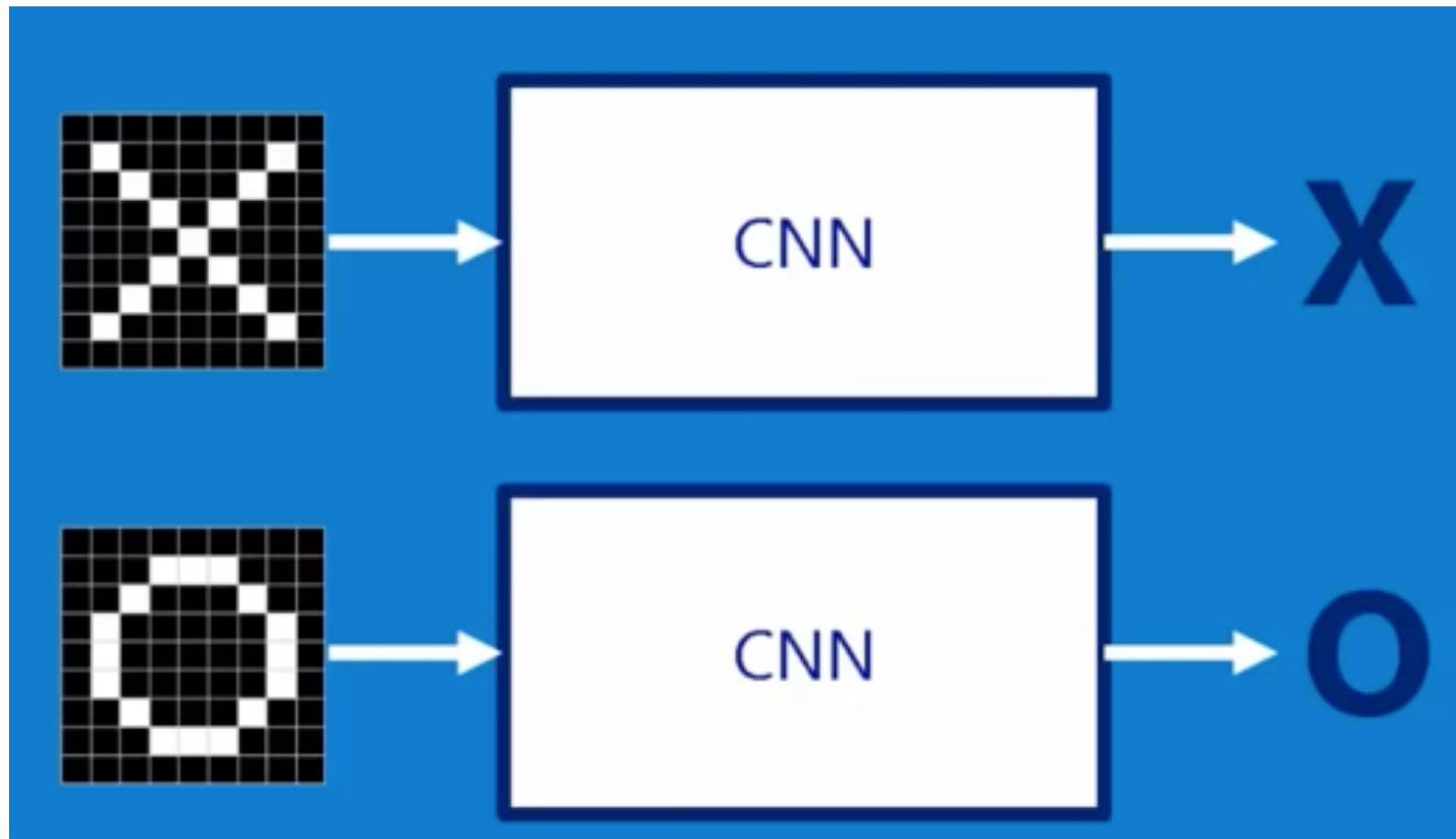
- Sabendo que se trata de uma imagem, podemos assumir que:
  - Existe um **relacionamento espacial** entre pixels próximos
  - Operações realizadas em um canal da imagem, podem ser realizadas em paralelo para todos os canais (o peso poderia ser único)

# Ideia

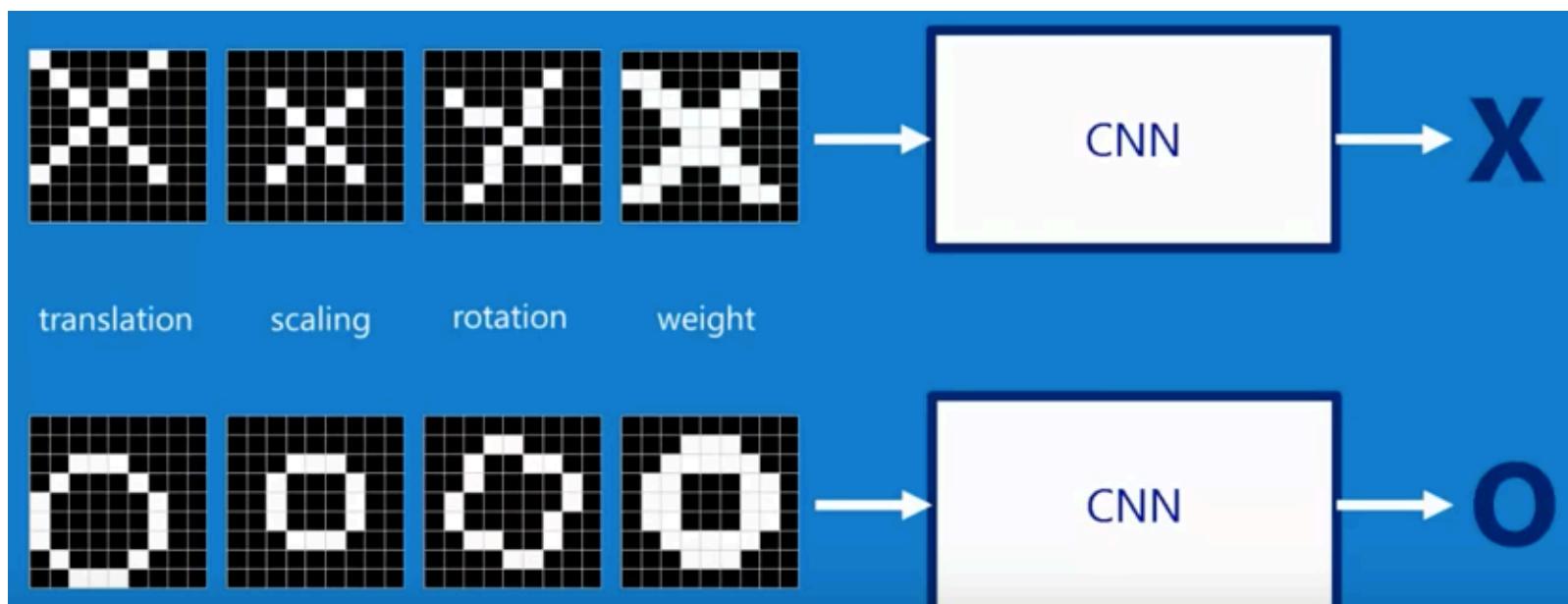
- Assim as **ativações** na imagem podem ser **localizadas** a uma região, em busca de **features** que definam a região



# Ideia



# Ideia



# Ideia

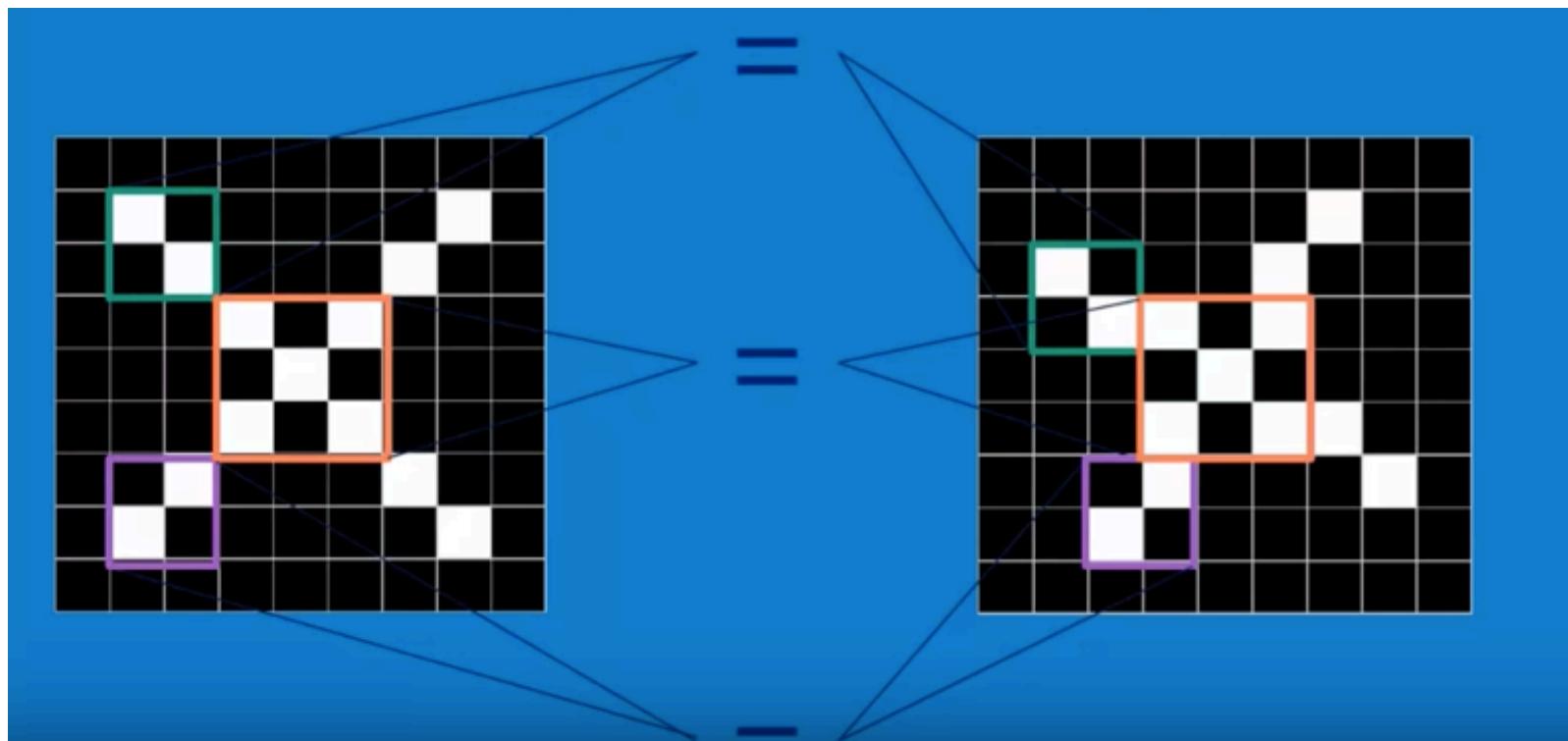
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

?

\_\_\_\_\_

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

# Ideia



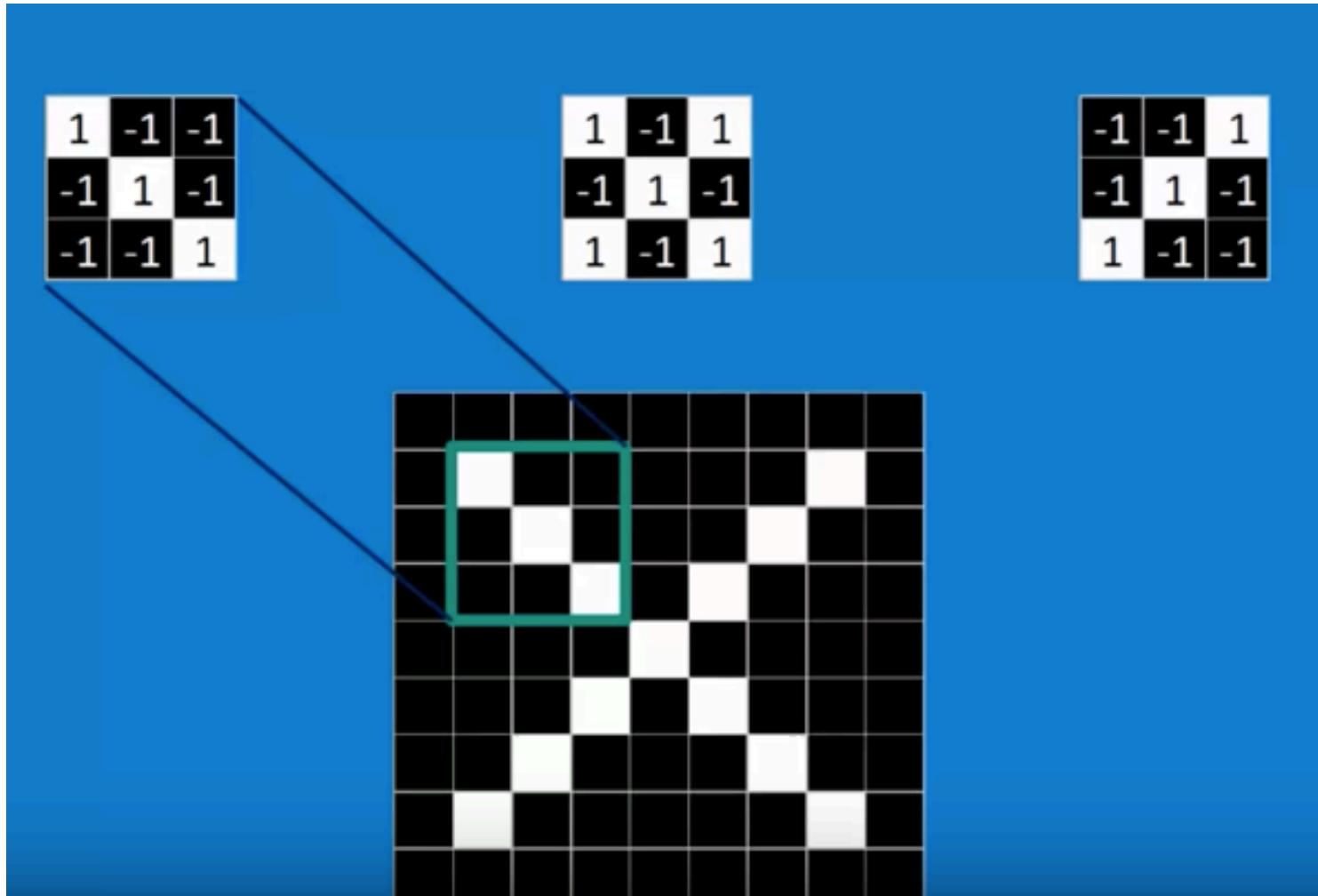
# Ideia

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

# Ideia



# Ideia

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

$$1 \times 1 = 1$$

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

1	1	1
1	1	1
1	1	1

( 12 )

# Ideia

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

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



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

( 13 )

# Ideia

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



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

=

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

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



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

=

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

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



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

=

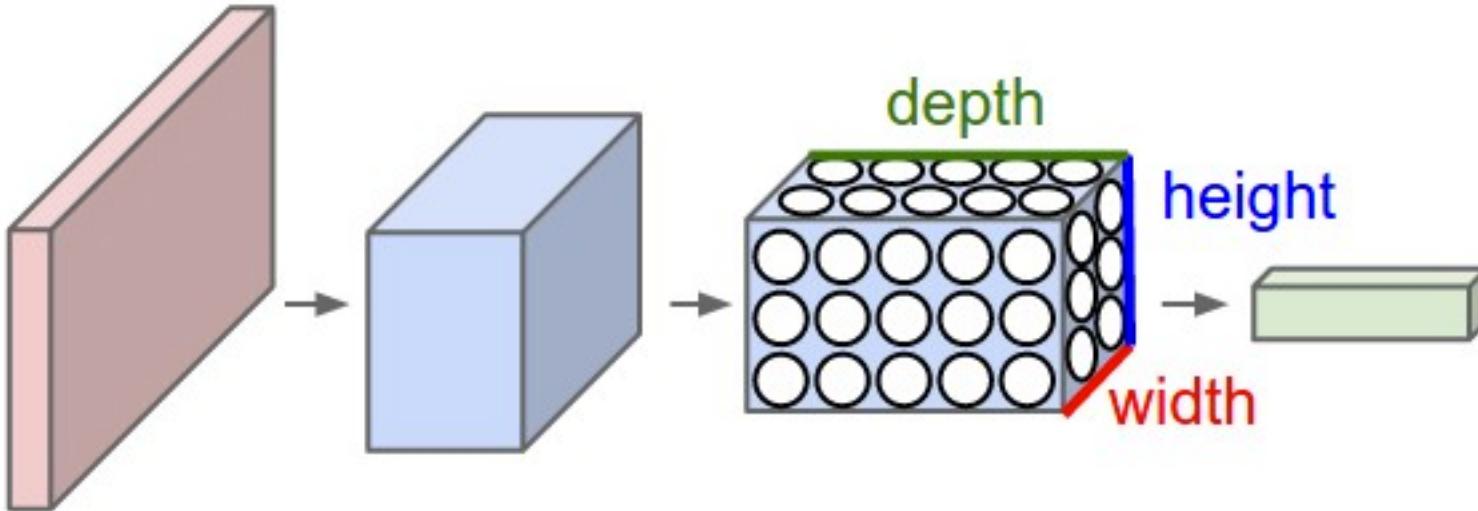
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

( 14 )

# Ideia

- Pode-se aprender várias features.
  - Gerando um volume que representam as ativações dos neurônios (pesos)
- Uma feature também é uma matriz
  - A ativação pode então ser obtida por uma operação de **convolução**
  - Que preserva ou não o tamanho original

# Ideia



O processo consiste em codificar a imagem numa quantidade menor de neurônios que preserve os relacionamentos espaciais anteriores

# Ideia

- Mas não vai gerar a mesma quantidade? Reduza o tamanho com Pooling

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

max pooling

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

( 17 )

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

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

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

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

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

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

# Ideia

- E esses valores negativos? Imagem não tem!  
ReLU

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



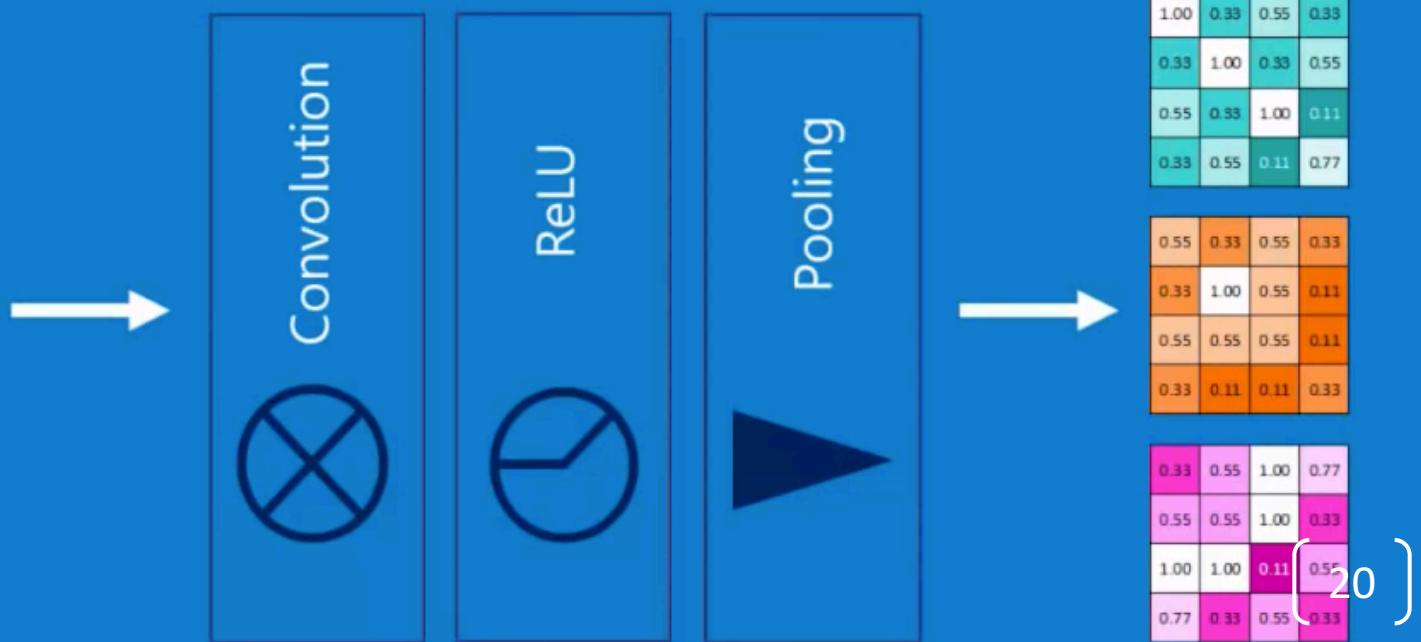
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

19

# Ideia

- E quem reconhece?

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

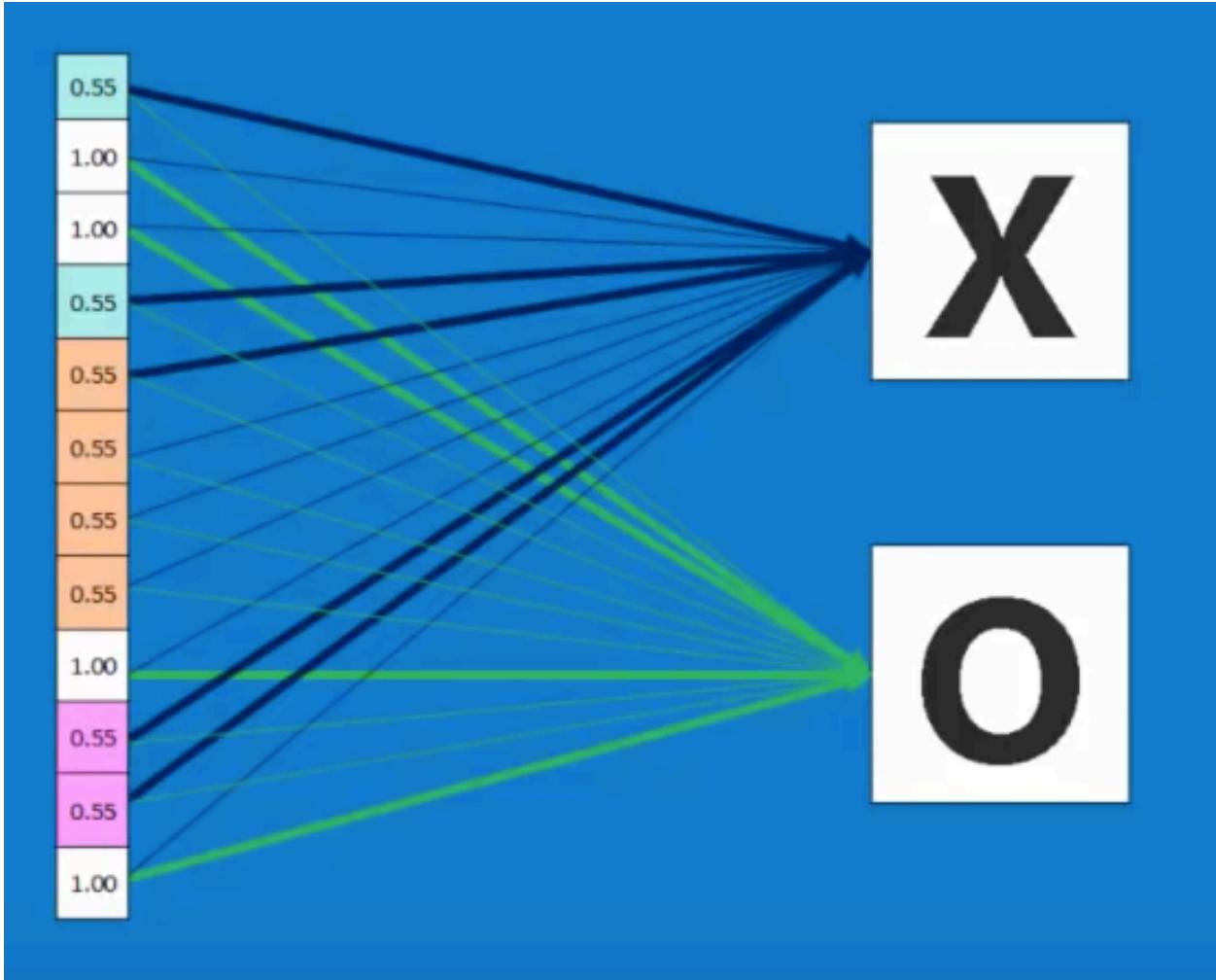


# Ideia

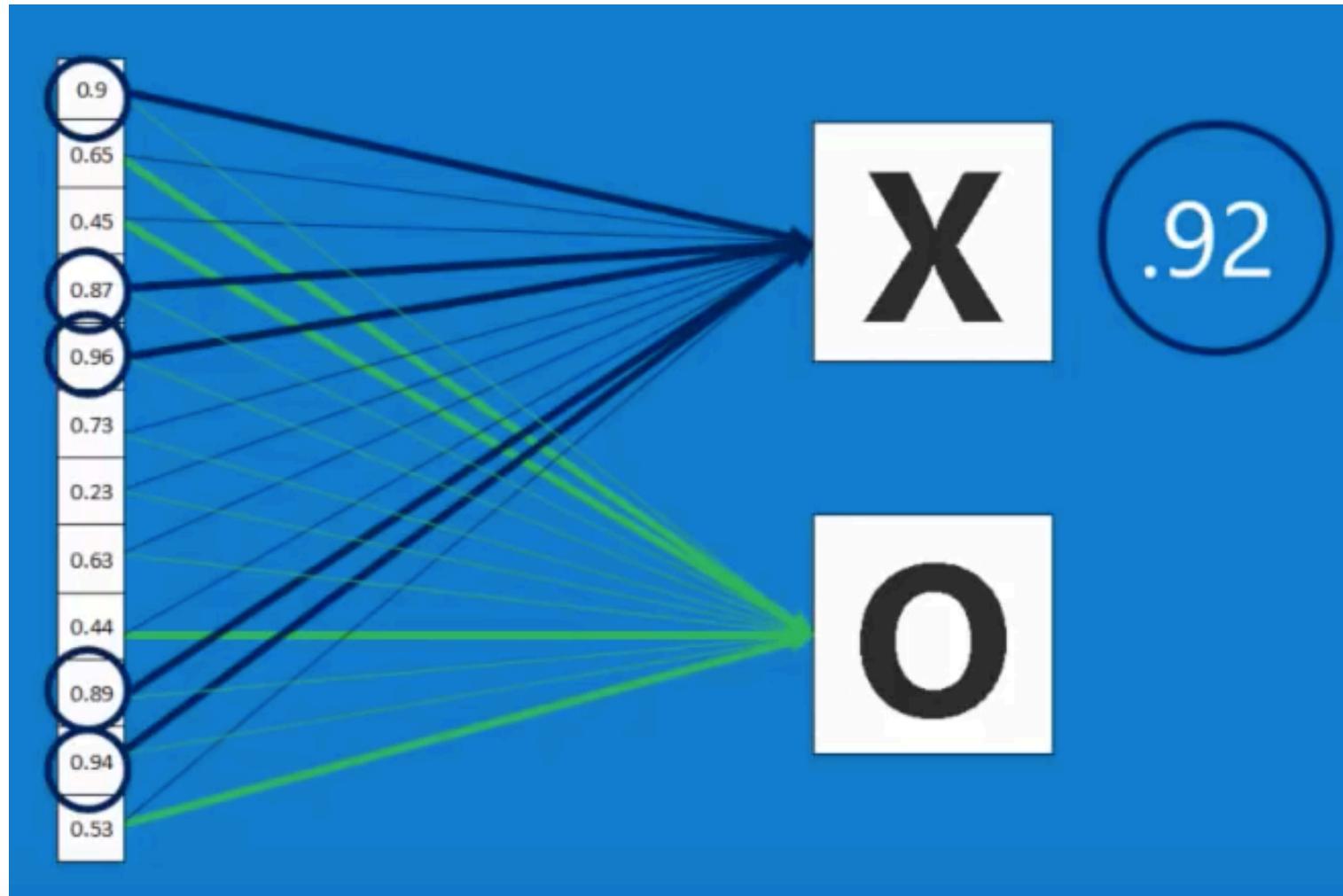
- Quem reconhece?
- MLP
  - Camada Totalmente Conectada



# Ideia

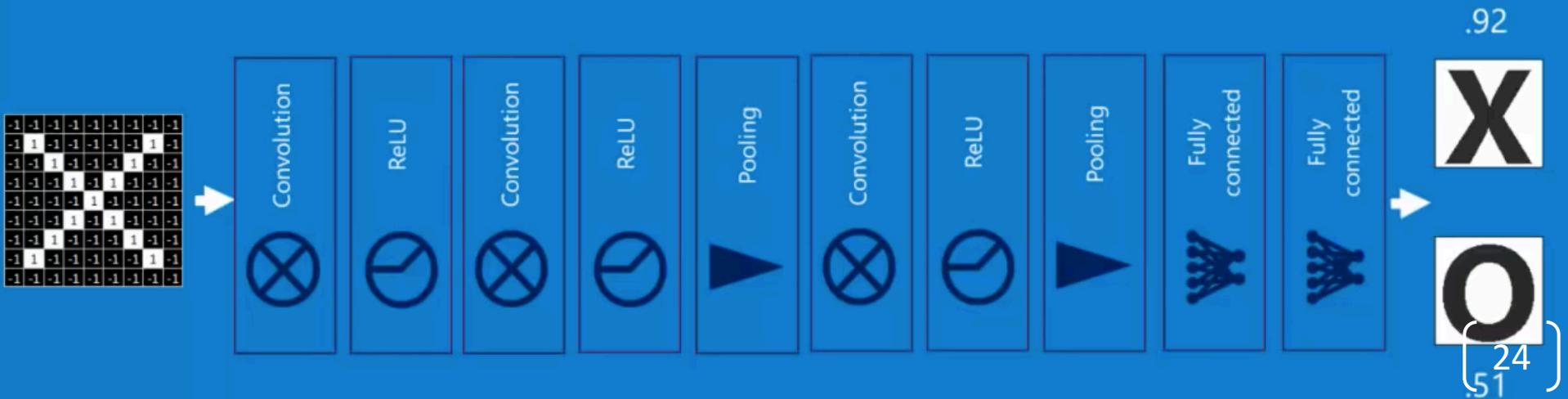


# Ideia



# Ideia

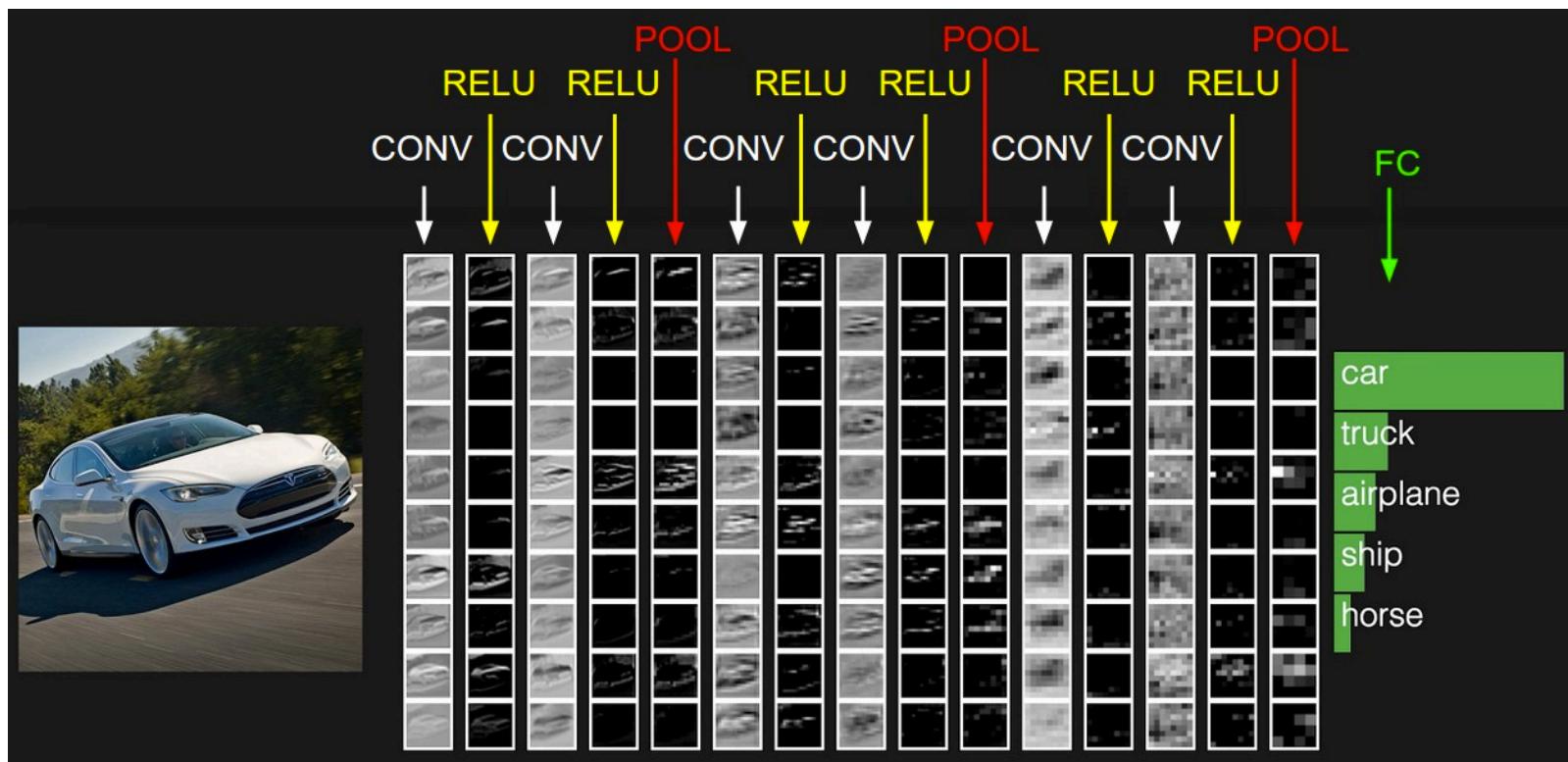
- O que eu faço com o erro?
- **Backpropagation**

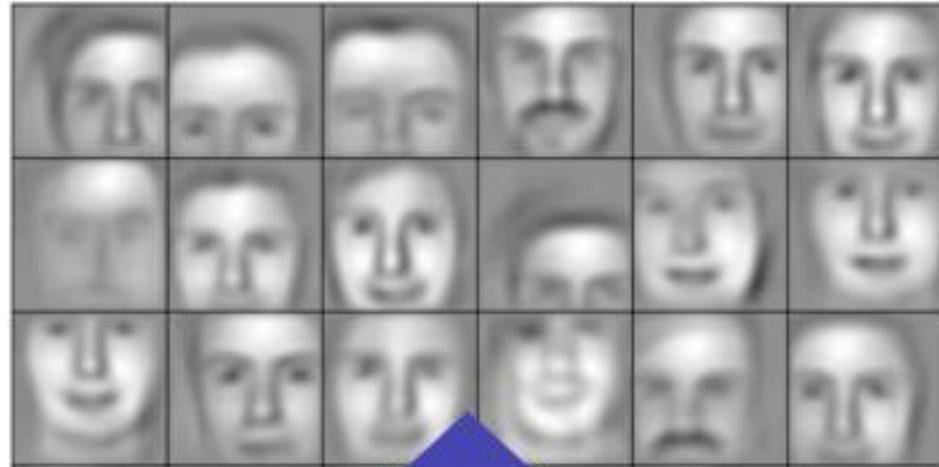


# Porque Deep?

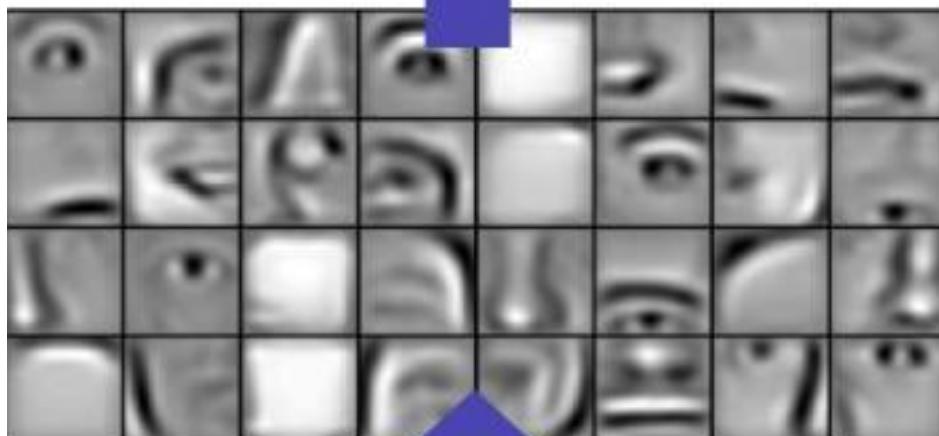
- A ideia é conseguir trabalhar com o espaço grande de características presente
- E capturar as informações que compõe o padrão
- **Do nível mais básico para o mais elevado**
- **De características de baixo nível → para características de alto nível**
  - Hierarquicamente

# Visualizando

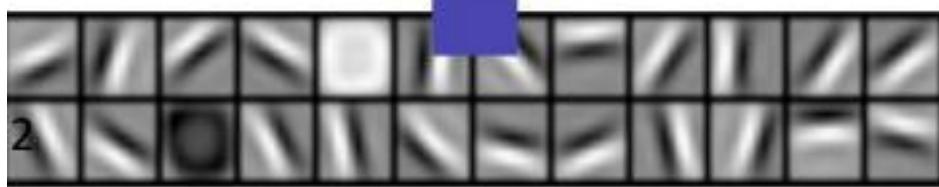




Layer 3



Layer 2



Layer 1

# Camadas Normalmente Usadas numa CNN

- **Convolucional** – gerar neurônios conectados a regiões das imagens
- **RELU** – Rectified Linear Unit
- **POOL** – Redução de dimensionalidade
- **FC** – Realizar a classificação

# Por Camada: Convolucional

- Predicado Local: não funciona conectar cada pixel a um neurônio
- Ideia: conectar um neurônio a uma região espacial da imagem
- Regulada por um hyperparameter: **receptive field** (se considerar como um filtro, então o **tamanho do filtro**)
  - Mesma profundidadade da imagem. → **sempre**

# Convolucional – Tamanho da Saída

- O tamanho do volume de resultado é controlado em termos de 3 variáveis (**de uso mutuamente exclusivas**):
  - **Depth** (hyperparameter): quantidade de filtros
  - **Stride** (hyperparameter): Quantidade de pixels que o filtro pula durante a convolução. Padrão 1 ou 2
    - Quando 1, o tamanho permanece igual em termos de altura x largura
    - Quando maior, diminui

# Convolucional - Tamanho da Saída

- **Zero-Padding:** preenchimento com zeros a borda para prevenir efeito de corte da borda pela convolução
- Relacionamento (**a conta sempre tem que ser inteira**):

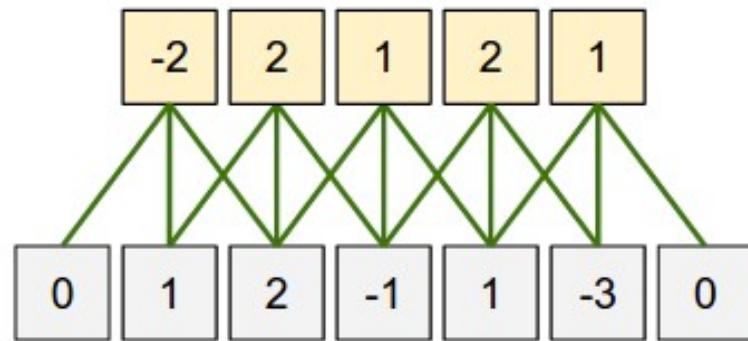
$$\underline{(W-F+2P)/S + 1}$$

onde W = tamanho da entrada

F = tamanho do filtro

P = tamanho do zero-padding

S = tamanho do stride

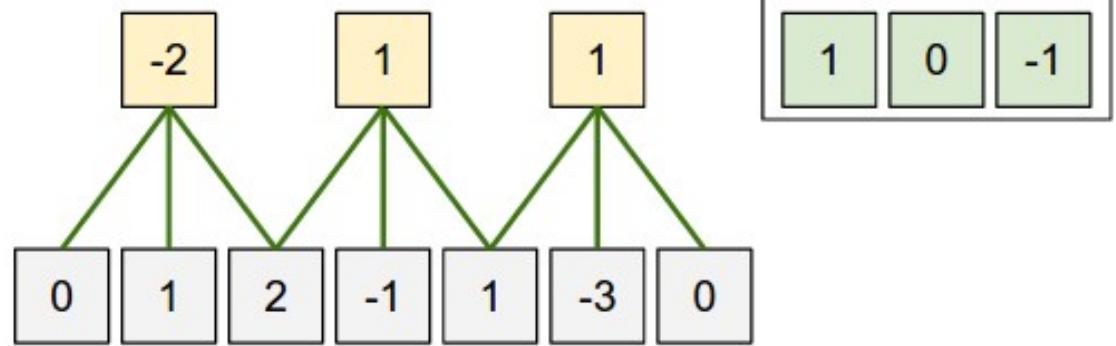


$$F = 3$$

$$W = 5$$

$$S = 1$$

$$P = 1$$



$$F = 3$$

$$W = 5$$

$$S = 2$$

$$P = 1$$

# Um exemplo: AlexNet

- Aceita imagens [227x227x3]
- Primeira Camada de Convolução
  - $F = 11$
  - $S = 4$
  - $P = 0$
  - $K = 96$  (quantidade de filtros)
    - $(227-11)/4 + 1 = 55$
- Final:  $(55 \times 55 \times 96)$  conectado cada um a uma região  $(11 \times 11 \times 3)$  → este último é o tamanho do filtro

# Funcionamento

**Summary.** To summarize, the Conv Layer:

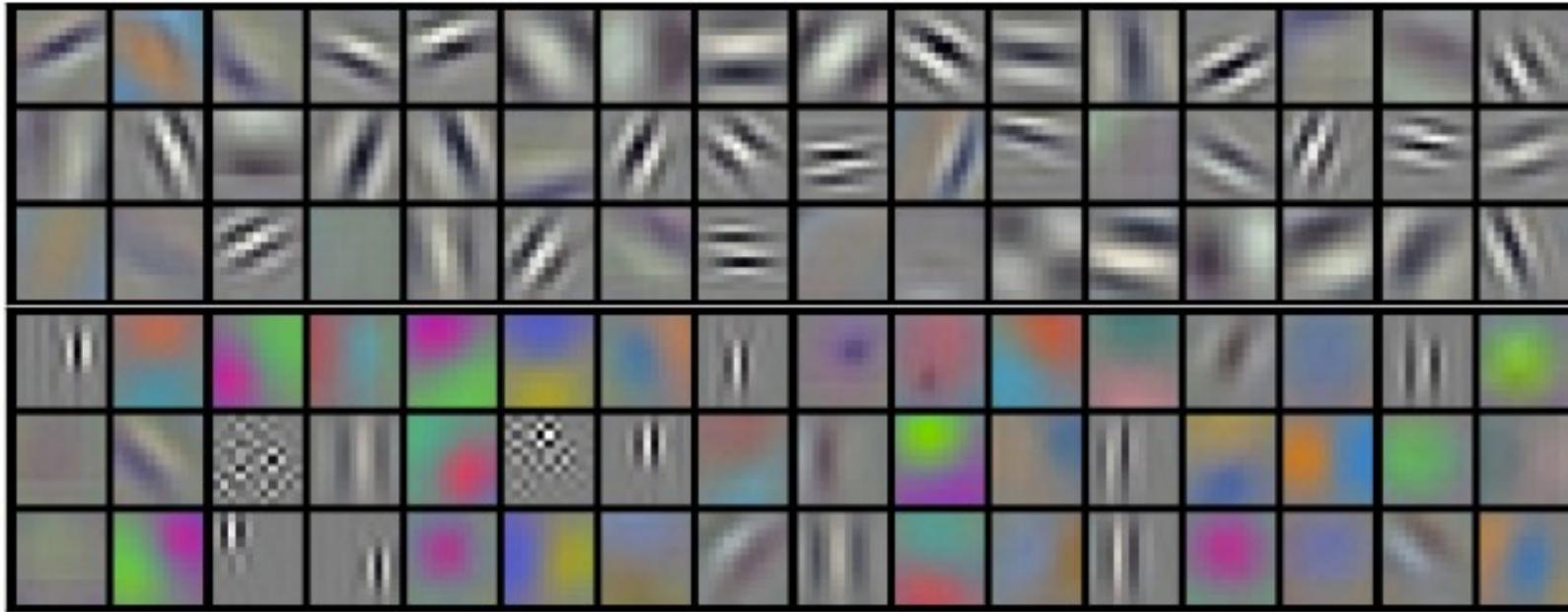
- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) is the result of performing a valid convolution of the  $d$ -th filter over the input volume with a stride of  $S$ , and then offset by  $d$ -th bias.



# Os filtros

- São aleatoriamente obtidos
  - Ou se usa uma base pré-treinada (**Transfer Learning**)
  - Mesmo com Transfer ainda é necessário se adaptar a nova base
- Representam os fatores de peso da rede (**w**) (neurônios na convolução)
  - Com o seu próprio bias (**b**)
- São apreendidos via **backpropagation**
- Cada filtro representa uma característica local na imagem
  - Quantos são necessários? Tamanho? São parâmetros

# Exemplos de Filtros Aprendidos



Uma dica para saber se colocou a quantidade suficiente de filtros:

- visualizar os filtros nas várias camadas e
- checar se estão se aproximando a algo relacionado com a base e se não são “aleatórios”

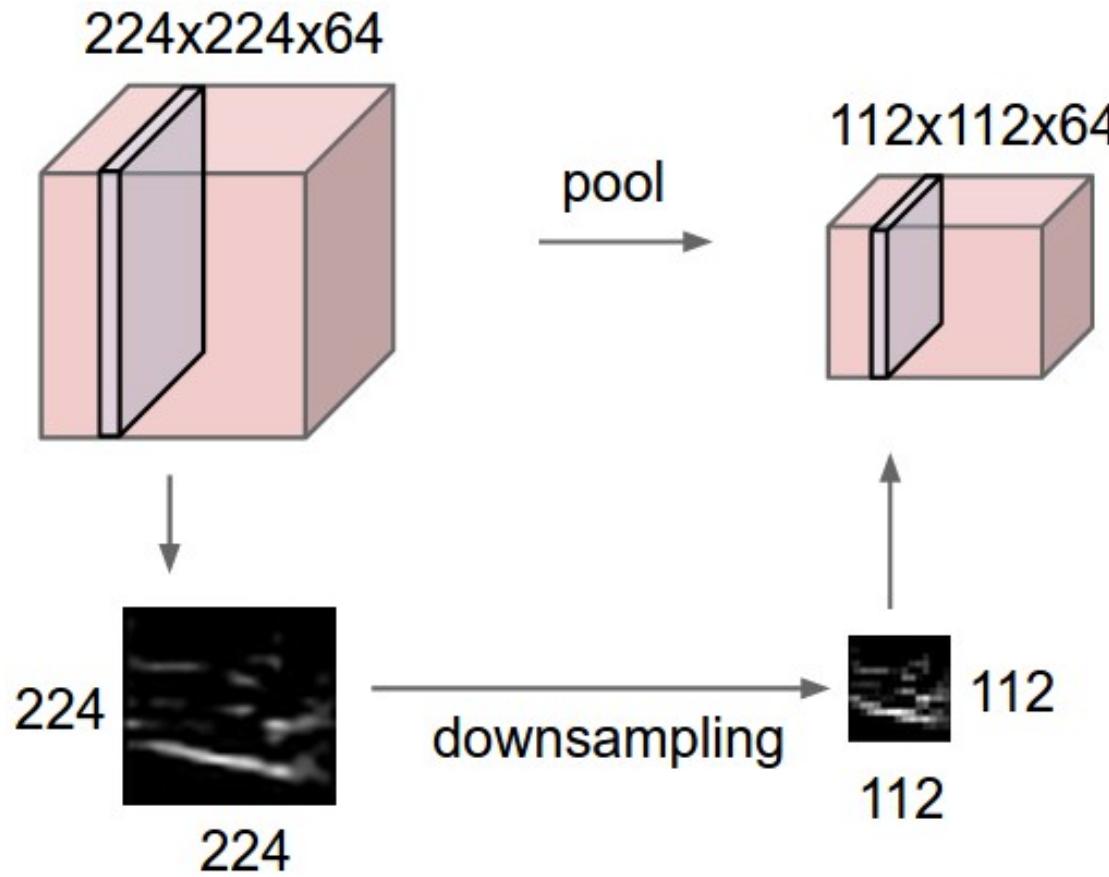
# RELU

- Ao final da convolução, pode-se aplicar RELU para remover valores negativos

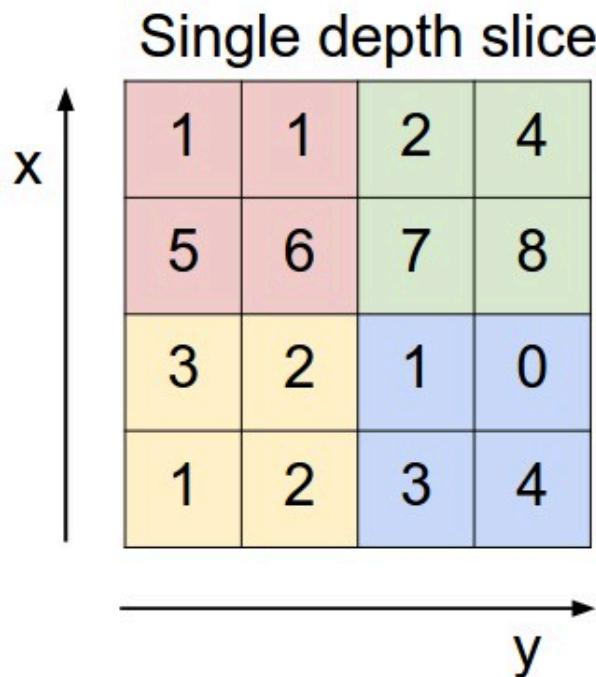
# Pooling

- Basicamente reduz dimensionalidade
  - Controla overfitting
  - Supõe que um valor represente bem uma área pequena
- Max pooling: mais comum, obtém o maior valor
- Mean pooling: obtém a média
- **Normal: F=2, S=2**

# Pooling



# Pooling



max pool with 2x2 filters  
and stride 2



6	8
3	4

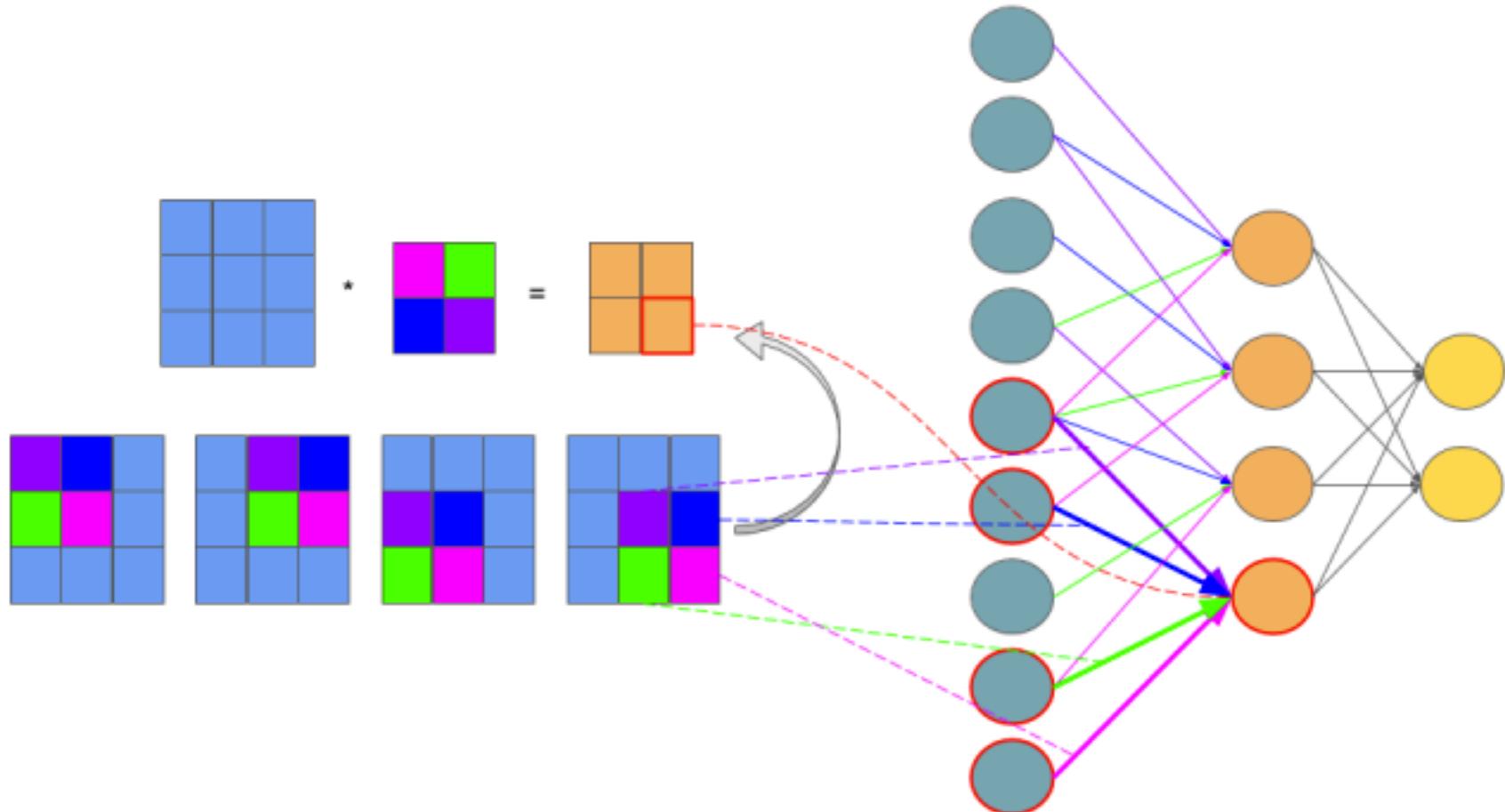
# Fully-Connected

- Basicamente uma **MLP** (qualquer outra técnica que se adapte ao modelo de aprendizado)
- O resultado das camadas anteriores é convertido em apenas um canal e vetORIZADO (sob sua convenção):
  - Na última camada =  $7 \times 7 \times 512$
  - Na entrada da MLP
    - troca entrada por uma convolução por  $F=7$  e  $K=4096$
    - Fica  $[1 \times 1 \times 4096]$

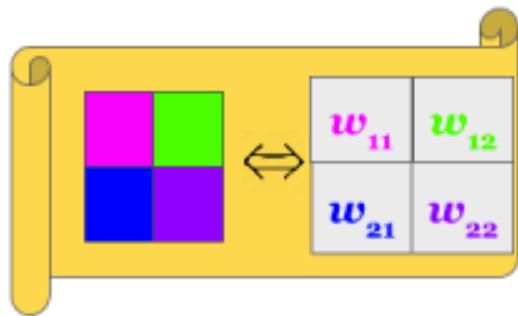
# Fully-Connected

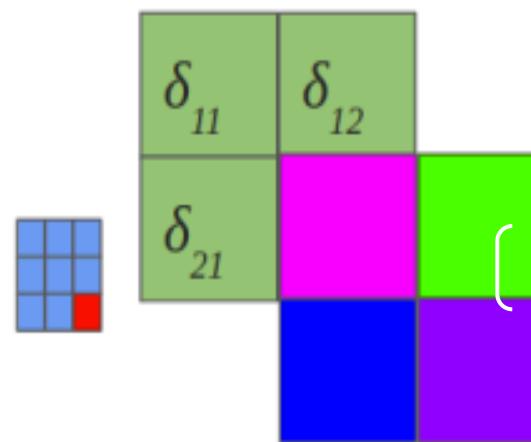
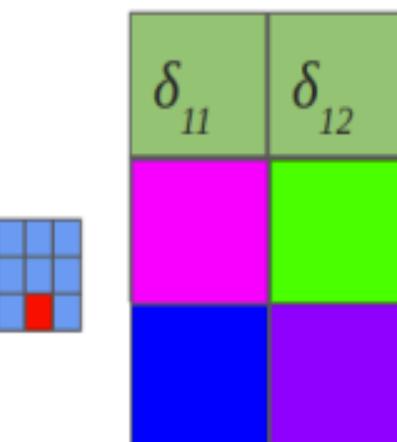
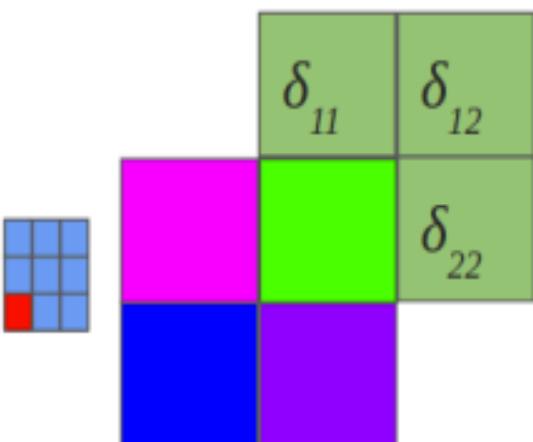
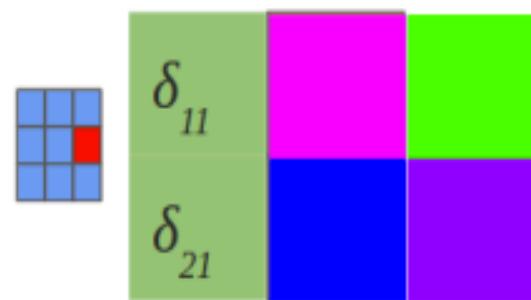
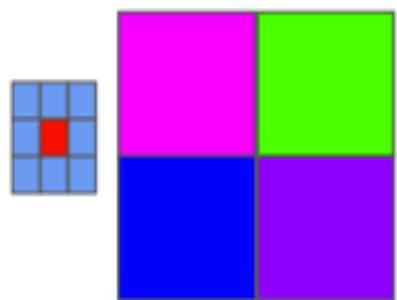
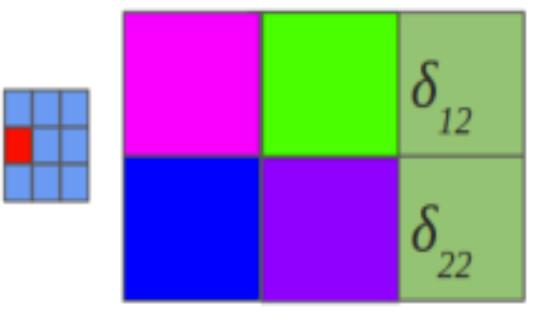
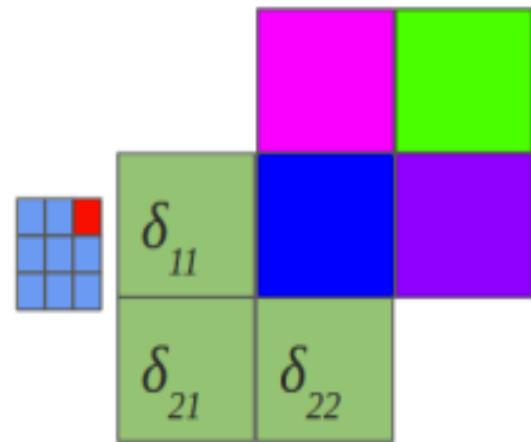
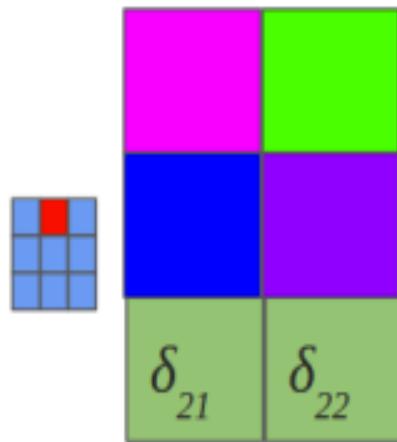
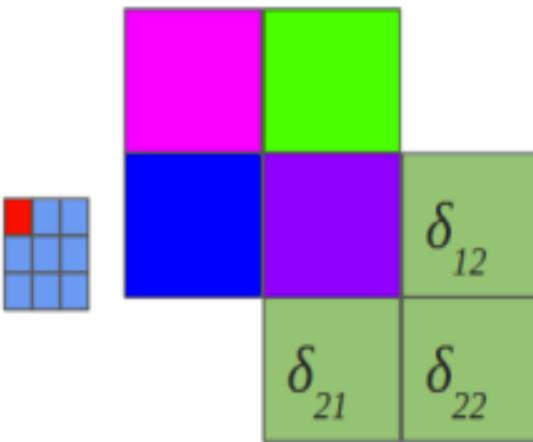
- Ao final se tem o erro de classificação
- Esse erro é devolvido para ajustar todos os pesos (backpropagation):
  - Dos neurônios na MLP
  - Dos neurônios no filtro de Pooling
  - Dos neurônios do filtro de Convolução

# Backpropagation



# Backpropagation

$$\begin{array}{c} \begin{array}{|c|c|} \hline w_{22} & w_{21} \\ \hline w_{12} & w_{11} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|} \hline \delta_{11} & \delta_{12} \\ \hline \delta_{21} & \delta_{22} \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|} \hline \delta_{11}w_{22} & \delta_{11}w_{21} + \delta_{12}w_{22} & \delta_{12}w_{21} \\ \hline \delta_{11}w_{12} + \delta_{21}w_{22} & \delta_{11}w_{11} + \delta_{12}w_{12} + \delta_{21}w_{21} + \delta_{22}w_{22} & \delta_{12}w_{11} + \delta_{22}w_{21} \\ \hline \delta_{21}w_{12} & \delta_{21}w_{11} + \delta_{22}w_{12} & \delta_{22}w_{11} \\ \hline \end{array} \\ \text{rot\_180}(w) \qquad \qquad \text{grads from orange layer} \\ \Leftrightarrow \begin{array}{|c|c|} \hline w_{11} & w_{12} \\ \hline w_{21} & w_{22} \\ \hline \end{array} \end{array}$$




# Backpropagation

- Também é uma convolução
  - O erro volta também como um filtro (convolução)

Gradiente do erro:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

onde (usando convolução):

$$z_{x,y}^{l+1} = w_{x,y}^{l+1} * \sigma(z_{x,y}^l) + b_{x,y}^{l+1} = \sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x-a,y-b}^l) +$$

$$b_{x,y}^{l+1}$$

sendo (ativação do neurônio)

$$a_j^l = \sigma(z_j^l)$$

# Backpropagation

Contribuição do erro de classificação na camada seguinte x a derivada do erro na camada atual levando em consideração a camada seguinte

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial (\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x'-a,y'-b}^l) + b_{x',y'}^{l+1})}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l)$$

Só conta neurônios ativados  
 $x = x' - a$   $y = y' - b$   
O resto zera

# Backpropagation

Troca a e b pelos valores de entrada

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l) = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} \sigma'(z_{x,y}^l)$$

Substitui pela convolução da função de erro pela função de peso multiplicada pela ativação

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{x'-x,y'-y}^{l+1} \sigma'(z_{x,y}^l) = \delta_{x,y}^{l+1} * w_{-x,-y}^{l+1} \sigma'(z_{x,y}^l)$$

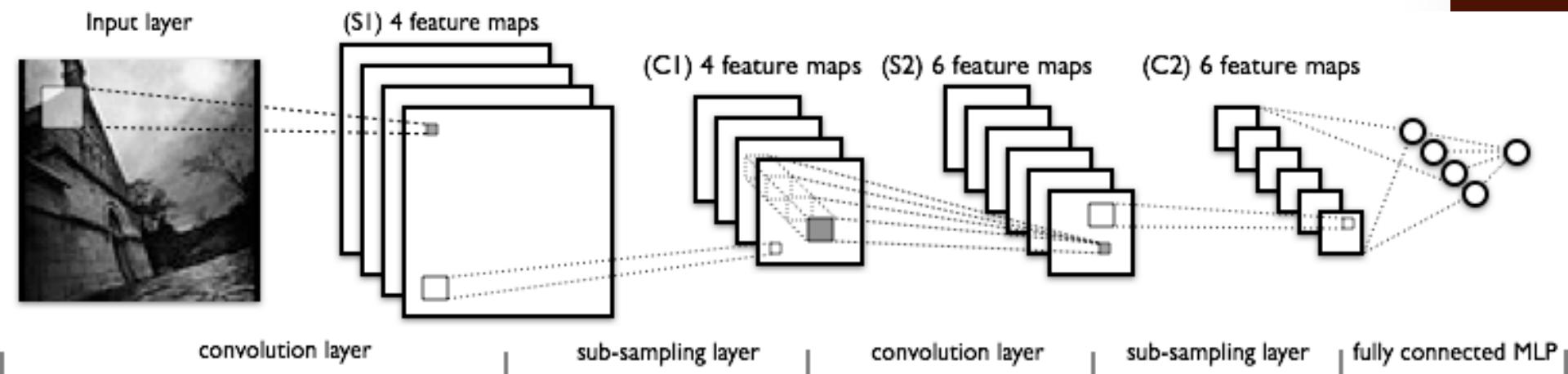
# Backpropagation

Sendo:  $ROT180(w_{x,y}^{l+1}) = w_{-x,-y}^{l+1}$

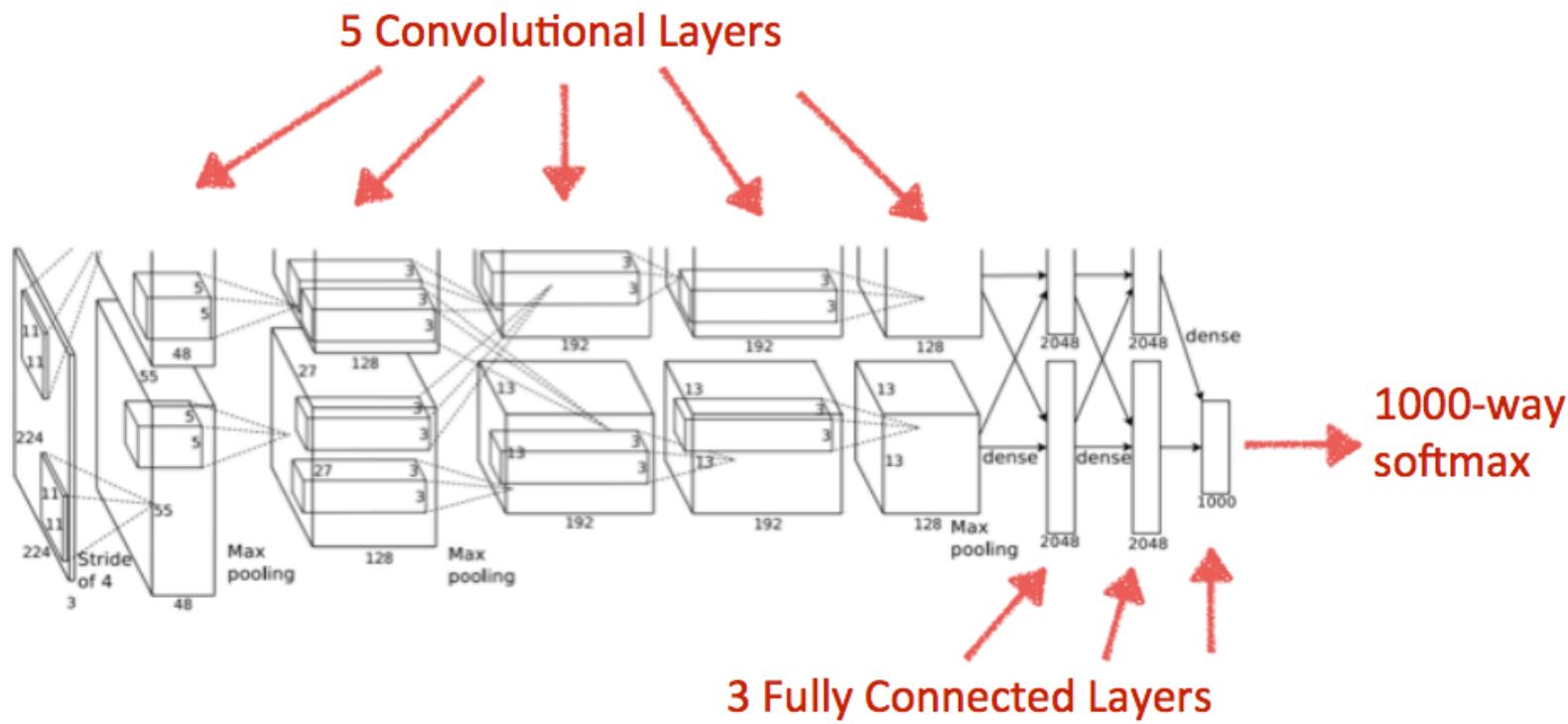
$$\frac{\partial C}{\partial w_{a,b}^l} = \sum_x \sum_y \frac{\partial C}{\partial z_{x,y}^l} \frac{\partial z_{x,y}^l}{\partial w_{a,b}^l} = \sum_x \sum_y \delta_{x,y}^l \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^l \sigma(z_{x-a',y-b'}^l) + b_{x,y}^l)}{\partial w_{a,b}^l} =$$
$$\sum_x \sum_y \delta_{x,y}^l \sigma(z_{x-a,y-b}^{l-1}) = \delta_{a,b}^l * \sigma(z_{-a,-b}^{l-1}) = \delta_{a,b}^l * \sigma(ROT180(z_{a,b}^{l-1}))$$

# Algumas arquiteturas famosas

- LeNet (1990)

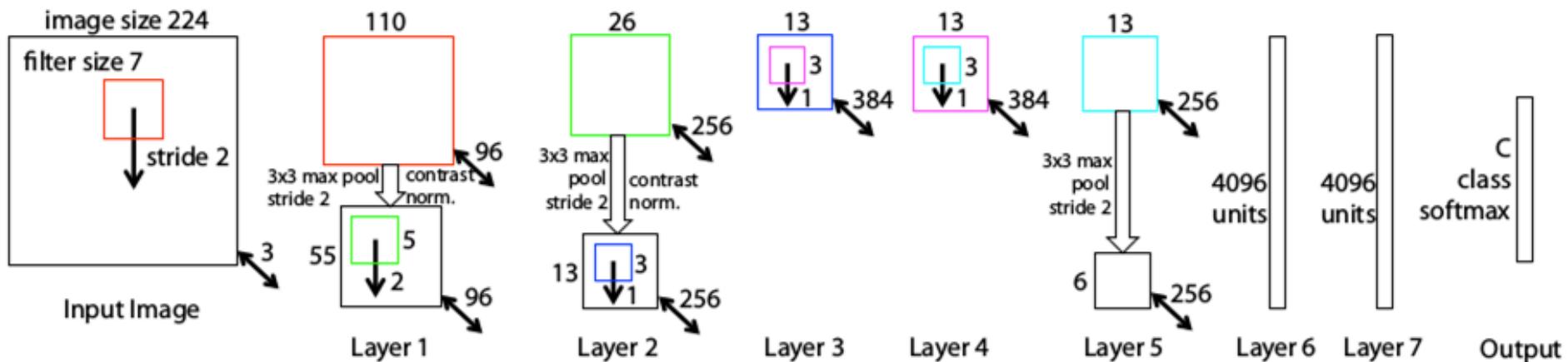


# AlexNet (2012)



# ZFNet (2013)

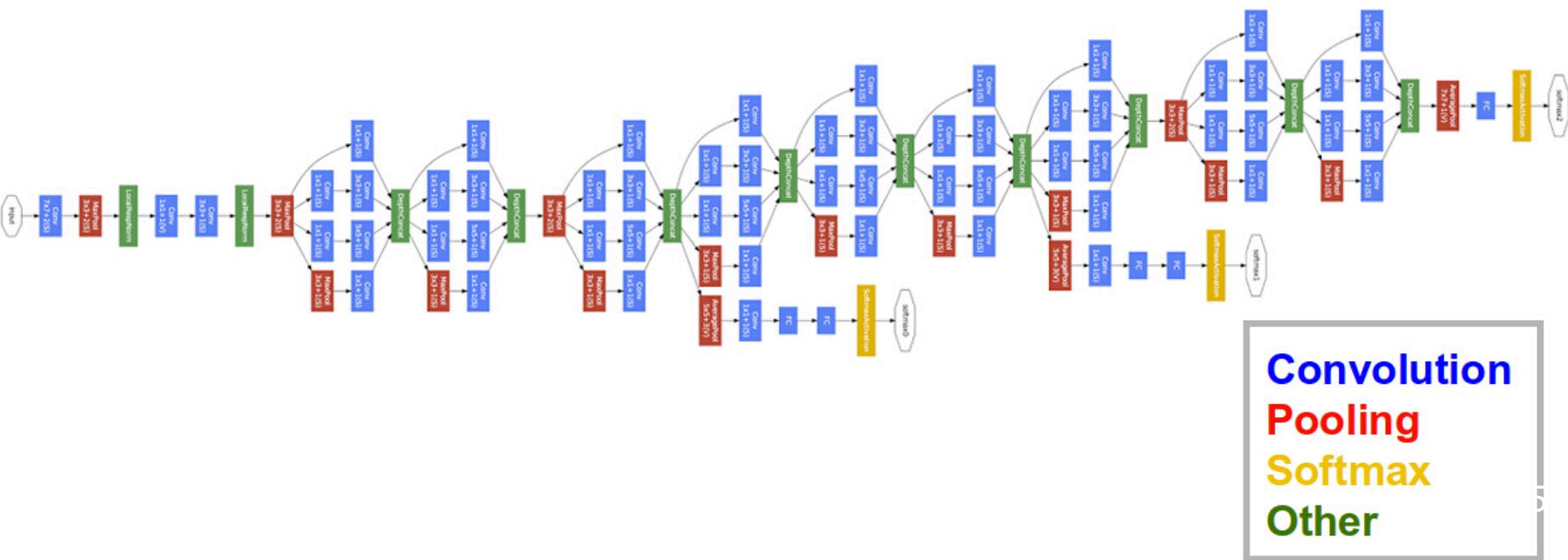
- Aumenta o tamanho da AlexNet



ZF Net Architecture

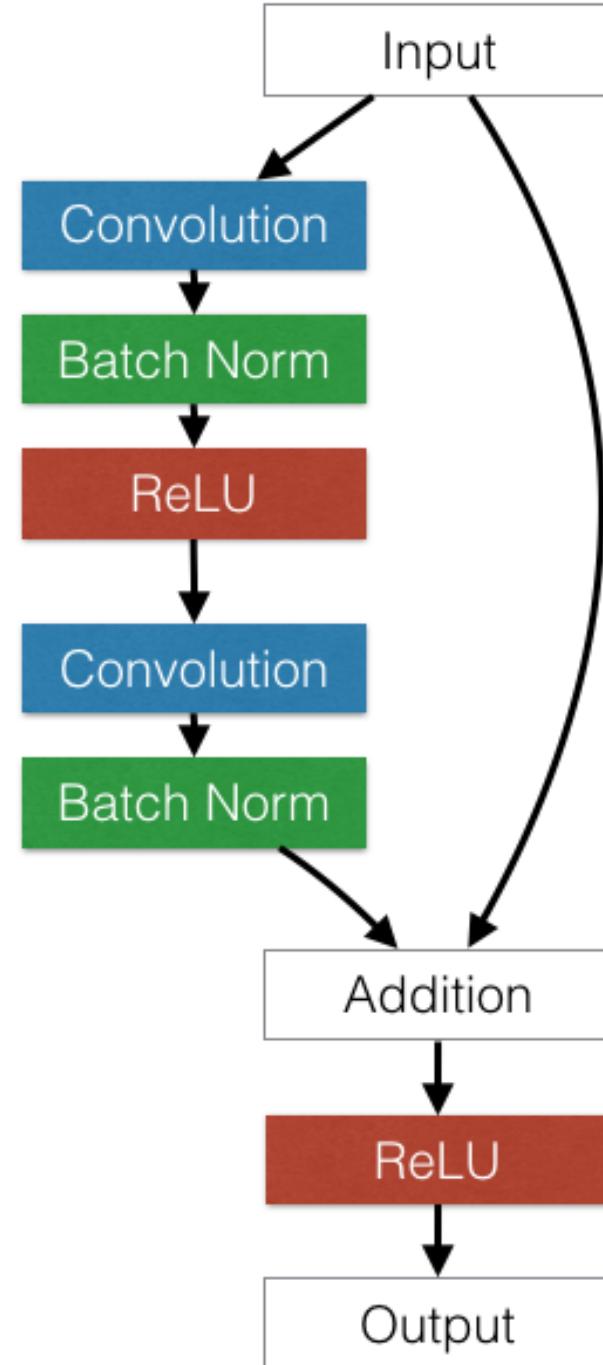
# GoogLeNET (2014)

- Inception Mode e Average Pooling ao invés de FC

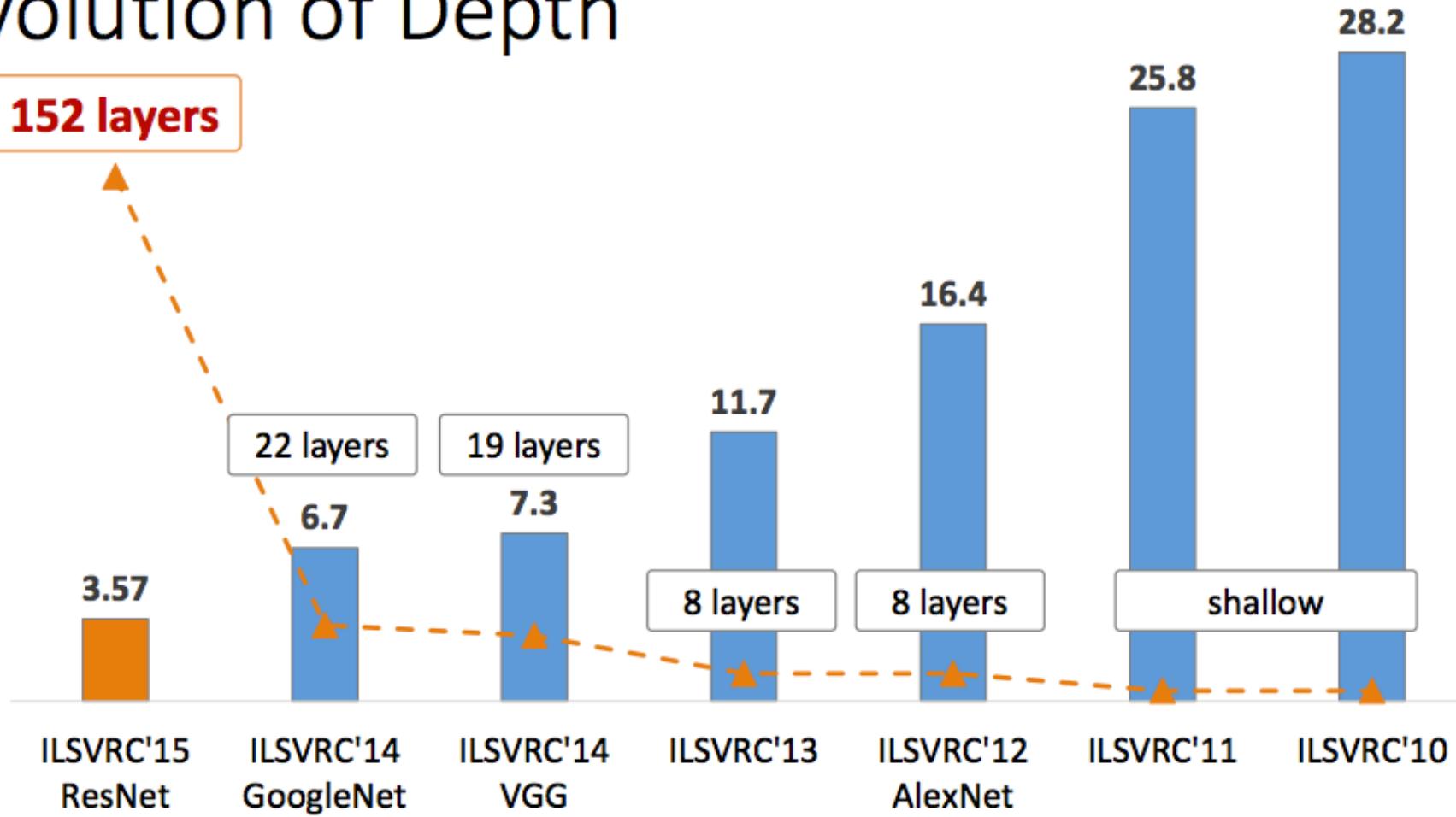


# ResNet (2015)

- Aprende a pular camadas quando necessário
- Originalmente 152 camadas
  - Mas a medida que treina, identifica “saltos”



# Revolution of Depth



# Considerações

- Principal consideração
  - Como definir a arquitetura e os hyperparametros?
- Por enquanto:
  - Partir de um que existe, e
  - Otimizar
  - Visualize os filtros
    - O que está saindo faz sentido?

# Considerações

- Existem vários tipos de aprendizado profundo ou métodos de aprendizado (além da CNN)
  - Stacked Auto-Encoders → aprendizado via reconstrução
  - Deep Belief Networks (DBN) → recurrent
  - Hierarchical Temporal Memory → reconhecimento espacial e inferência temporal
  - Deep Spatial Temporal Inference Network (DESTIN) → hierarquia de aprendizados não supervisionados com relacionamento temporal e local
- Leia mais ;)