"How do you make a computer blink?" (Kasparov vs. Deep Blue, 1997).

Convolutional Neural Networks

Paulo Ricardo Lisboa de Almeida

DSBD

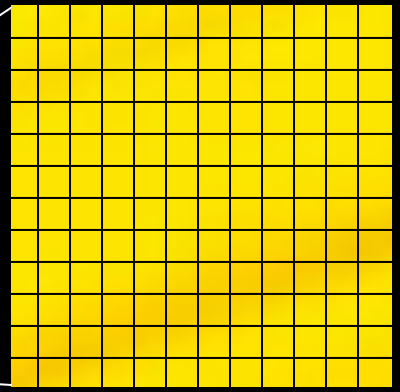
UFPR

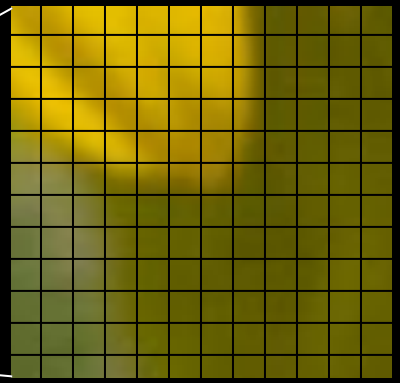
Correlação entre as entradas

Em muitas aplicações existe correlação entre os dados de entrada.

Os dados são estruturados.

Correlação entre as entradas





Existe correlação entre as cores de um pixel e dos seus vizinhos.

Matriz de Convolução

Também chamada de Kernel ou Filtro.

Exemplo supondo uma Matriz de convolução de 3x3, e uma imagem em escala de cinza.

Exemplo

Imagem Original

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

Kernel de Convolução

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

Imagem Resultante

Exemplo

Imagem Original

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

58*0.11 + 40*0.11 + 35*0.11 +
120*0.11 + 62*0.11 + … = 93

Kernel de Convolução

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

Imagem Resultante

93						

Exemplo

Imagem Original

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

40*0.11 + 35*0.11 + 20*0.11 + 62*0.11
+ 55*0.11 + … = 64

Kernel de Convolução

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

Imagem Resultante

93	64					

Exemplo

Imagem Original

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

35*0.11 + 20*0.11 + 18*0.11 + 55*0.11
+ 48*0.11 + … = 41

Kernel de Convolução

0.11	0.11	0.11
0.11	0.11	0.11
0.11	0.11	0.11

Imagem Resultante

93	64	41				

Box Blur

A matriz de convolução do exemplo aplica um Borramento de Caixa (Box Blur).

No exemplo o kernel é 6x6. Fonte da imagem original: https://en.wikipedia.org/wiki/Jet_engine#/media/File:JT9D_on_747.JPG

Matriz de Convolução

Alterar o tamanho e os valores de Matrizes de convolução altera o resultado. Alguns exemplos:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$ identidade

$$\begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix}$$ Box Blur

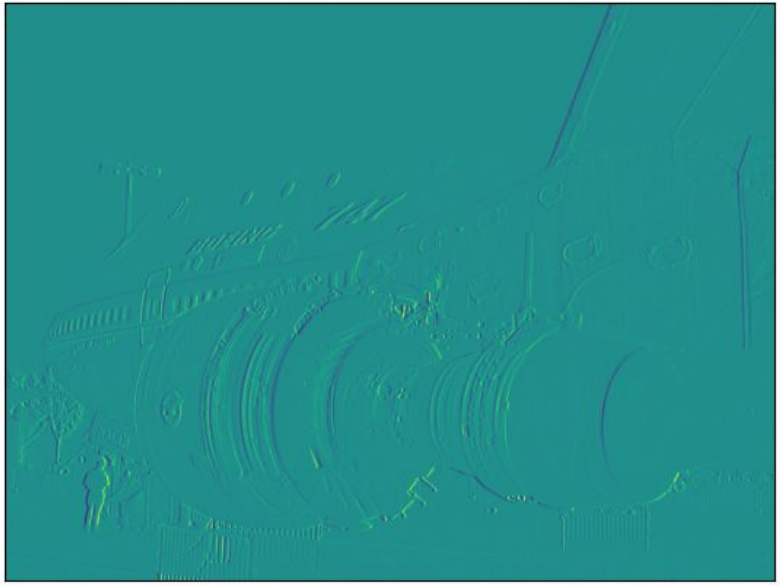
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$ aguçar (*sharpen*)

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$ Sobel no eixo X (detector de bordas)

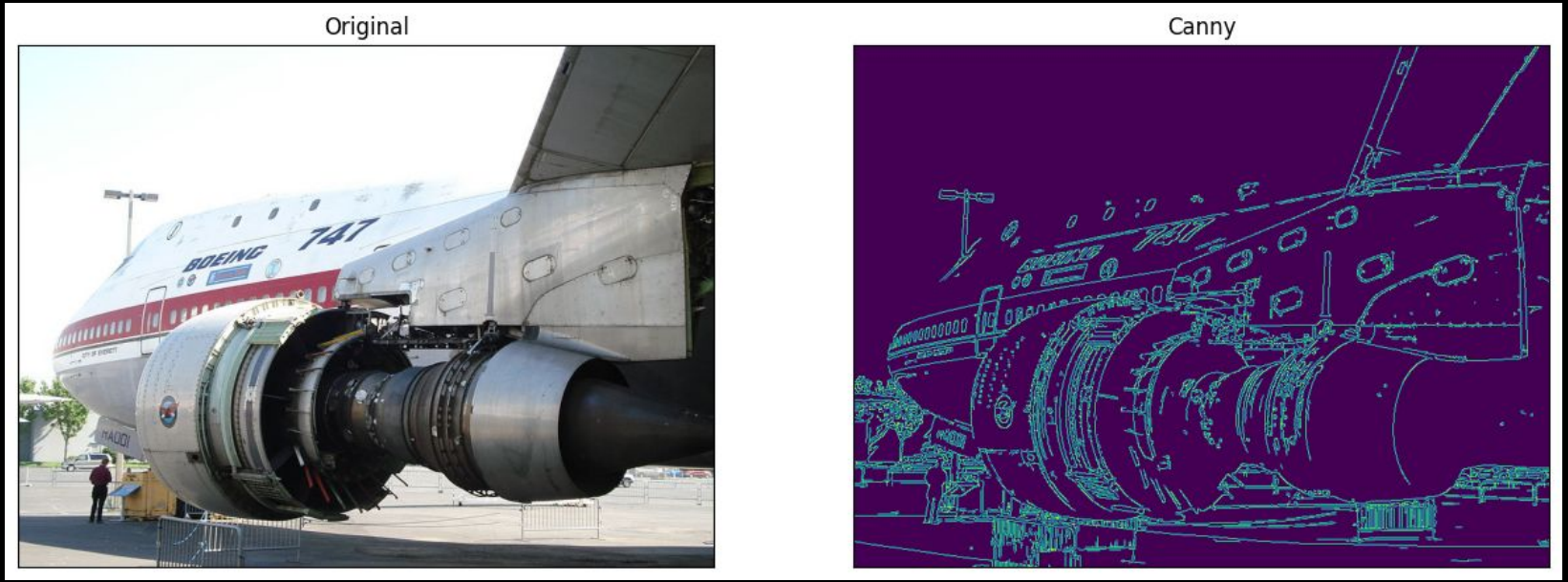
Exemplo - Sobel

Original

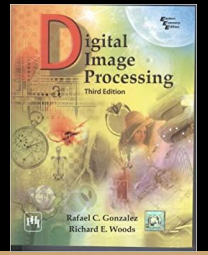
Sobel



Exemplo - Algoritmo de Canny



O Algoritmo de Canny na verdade é o resultado da aplicação de várias operações em sequência (não é uma única matriz de convolução).
Veja em Gonzalez, R. C., Woods, R. E. Digital image processing, 2008.



Faça você mesmo

Execute os filtros de exemplo disponibilizados no Google Colab.

Tente carregar outras imagens e filtros, e modificar seus parâmetros.

Matrizes de Convolução em sequência

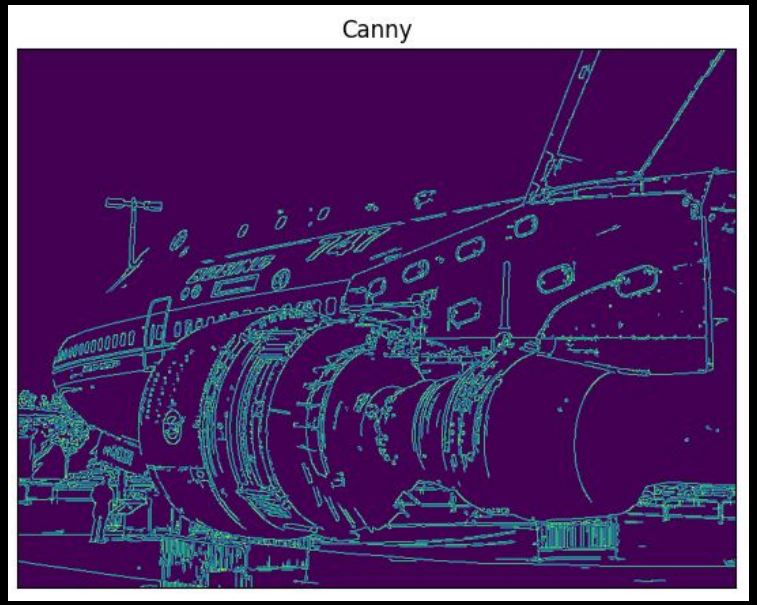
Podemos aplicar as matrizes de convolução em sequência para melhorar o resultado esperado.

Por exemplo, se o objetivo é encontrar contornos.

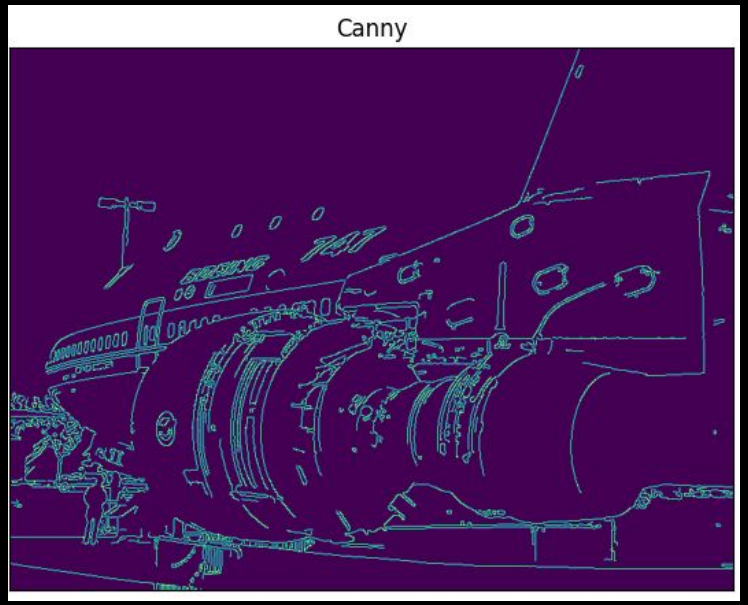
imagem -› Box Blur -› Sobel (ou Canny) -› Resultado.

Matrizes de Convolução em sequência

Sem Box Blur.



Com Box Blur.



Uso tradicional

1. Definir a tarefa a ser realizada -› Exemplo. Verificar se discos de freio estão rachados.

Uso tradicional

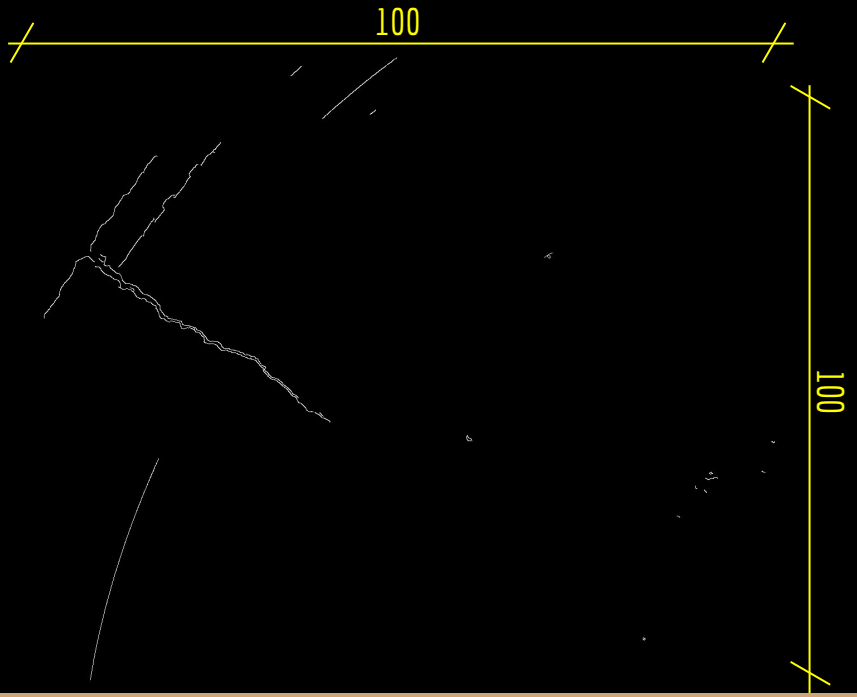
2. Um especialista define os filtros necessários para evidenciar a informação, e transformar em um vetor de características.

No exemplo, usamos Box Blur e Canny. Mas **tudo depende do problema**.

Uso tradicional

3. O especialista transforma a informação em um vetor de características algo que pode ser processado.

Por exemplo, criar um histograma que contra quantos pixels brancos existem no eixo y e no eixo x.



0	0
1	0
…	…
35	20
36	25
…	…
100	0

0	1	…	10	11	…	100
0	0	…	5	7	…	0

Uso tradicional

0	0
1	0
…	…
35	20
36	25
…	…
100	0

4. Usar os histogramas como vetores de características para treinar classificadores (exemplo MLPs).

0	1	…	10	11	…	100
0	0	…	5	7	…	0

Deep Learning Versus Engenharia de Características

A técnica de definir os filtros para extrair características é um tipo de engenharia de características.

- Funciona bem.
- Baixo custo computacional.

Deep Learning Versus Engenharia de Características

A técnica de definir os filtros para extrair características é um tipo de engenharia de características.

+ Funciona bem.
+ Baixo custo computacional.
- Requer conhecimento detalhado do domínio do problema.
- Muitas vezes é complexo definir os extratores de características.

Deep Learning Versus Engenharia de Características

A técnica de definir os filtros para extrair características é um tipo de engenharia de características.

+ Funciona bem.
+ Baixo custo computacional.
- Requer conhecimento detalhado do domínio do problema.
- Muitas vezes é complexo definir os extratores de características.

Poderíamos utilizar um MLP com “Deep Learning simples”, ligando cada pixel a um neurônio de entrada.

+ Não precisamos definir os extratores. A rede vai aprender sozinha.
+ Pode levar a resultados melhores do que com a engenharia de características.

Deep Learning Versus Engenharia de Características

A técnica de definir os filtros para extrair características é um tipo de engenharia de características.

+ Funciona bem.
+ Baixo custo computacional.
- Requer conhecimento detalhado do domínio do problema.
- Muitas vezes é complexo definir os extratores de características.

Poderíamos utilizar um MLP com “Deep Learning simples”, ligando cada pixel a um neurônio de entrada.

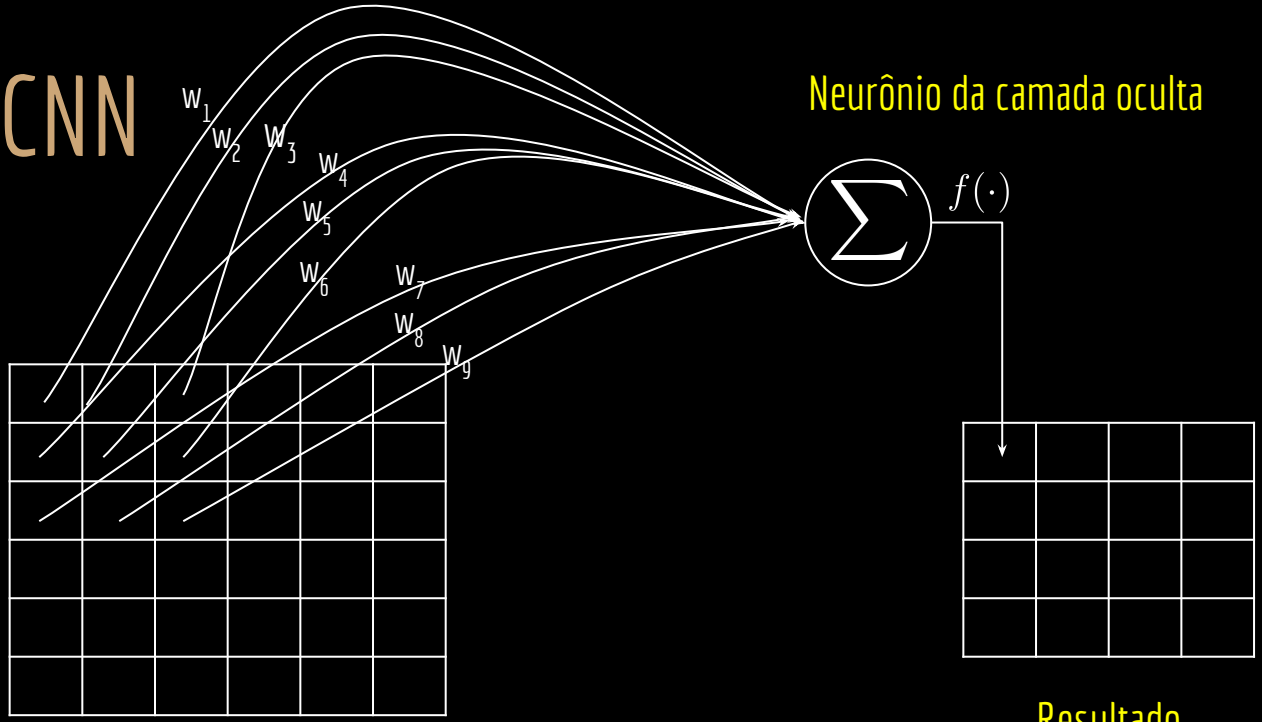
+ Não precisamos definir os extratores. A rede vai aprender sozinha.
+ Pode levar a resultados melhores do que com a engenharia de características.
- A rede pode ficar gigantesca. Ex.: para imagens RGB 100 x 100, com 1.000 neurônios na camada oculta, serão necessários **3 bilhões** de parâmetros só na primeira camada.
- Necessidade de muitos dados de treinamento.

CNN - Convolutional Neural Network

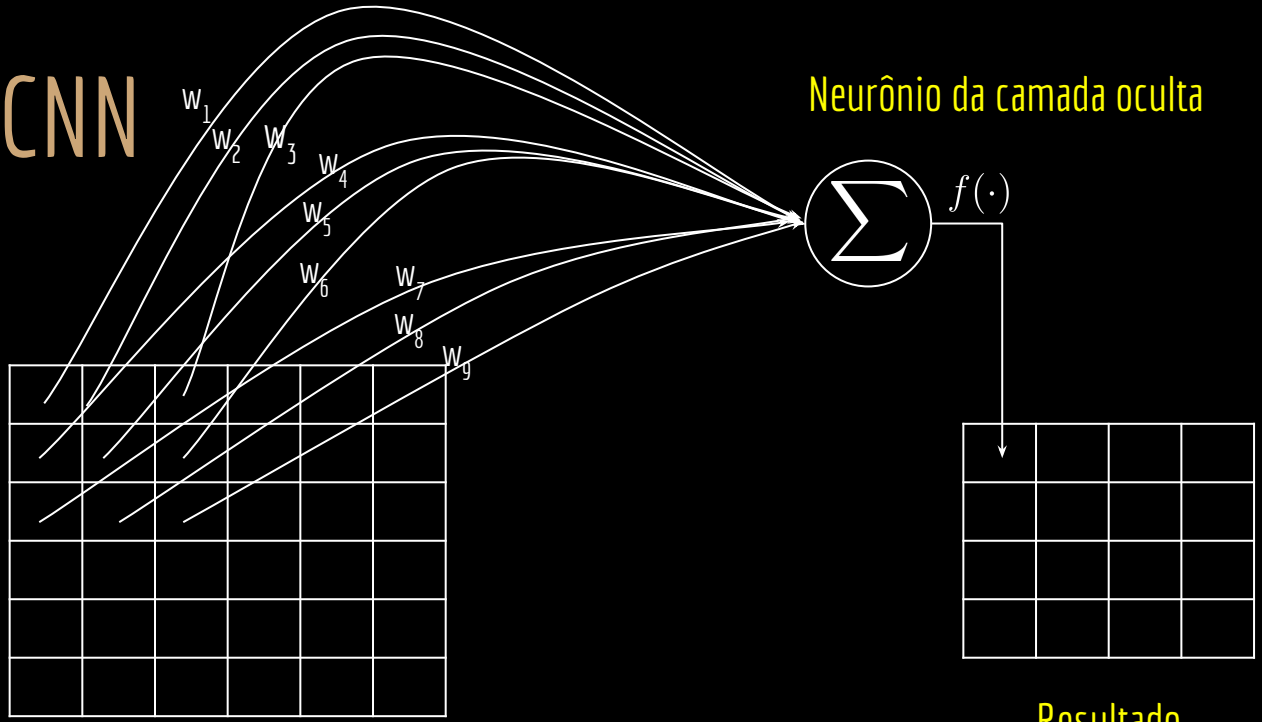
Uma CNN tenta utilizar a capacidade de ajuste do Deep Learning com a ideia de correlações das matrizes de convolução em uma rede.

Portanto, temos uma **Rede Neural Convolucional**.

CNN

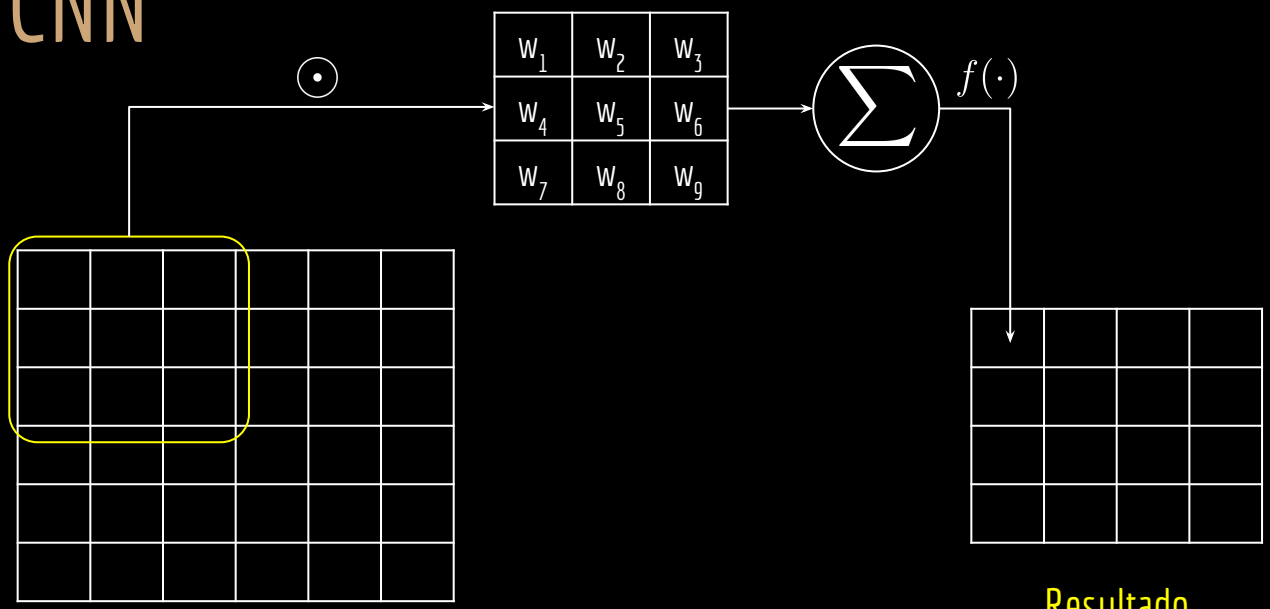

w1
w2
w3
w4
w5
w6
w7
w8
w9
Neurônio da camada oculta
$\sum$
$f(\cdot)$
Resultado
Pixels de imagem original

CNN



Os pesos do neurônio operam em uma pequena região (3 x 3 no exemplo) da imagem. Logo, os pesos do neurônio podem ser interpretados como uma matriz de convolução. Durante o treinamento, ajustar esses pesos é ajustar a matriz.

CNN



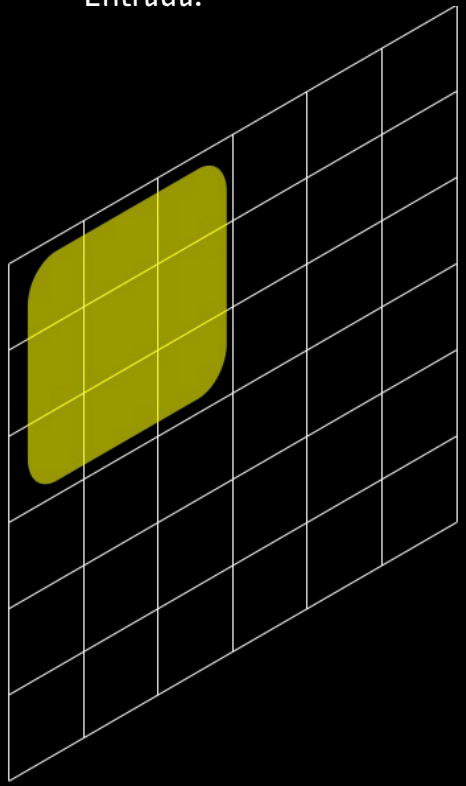
$\odot$ é o produto de Hadamard.

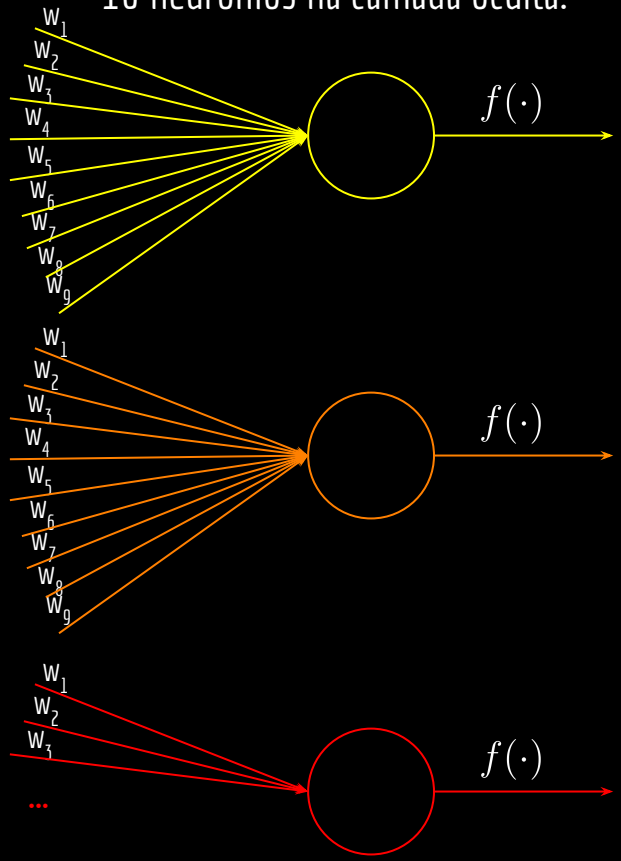
Os pesos do neurônio operam em uma pequena região (3 x 3 no exemplo) da imagem. Logo, os pesos do neurônio podem ser interpretados como uma matriz de convolução. Durante o treinamento, ajustar esses pesos é ajustar a matriz.

Pesos compartilhados

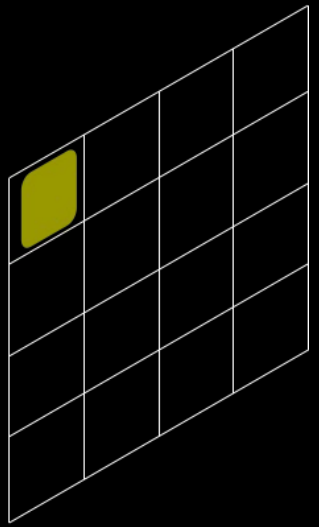
16 neurônios na camada oculta.

Entrada.



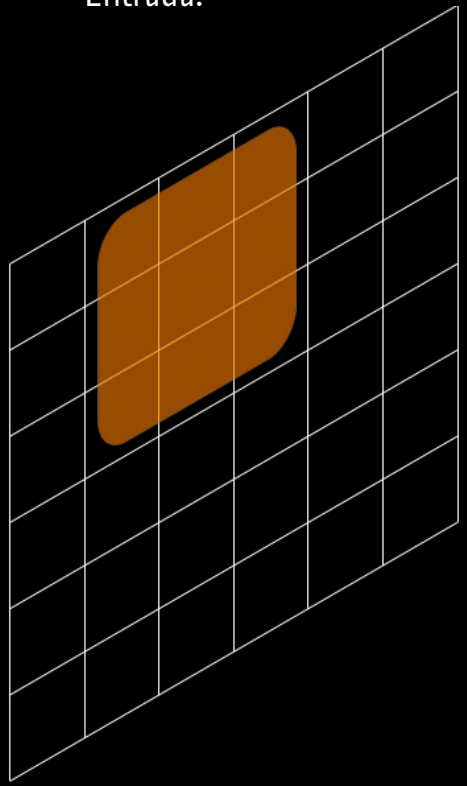


Feature Map de Saída.

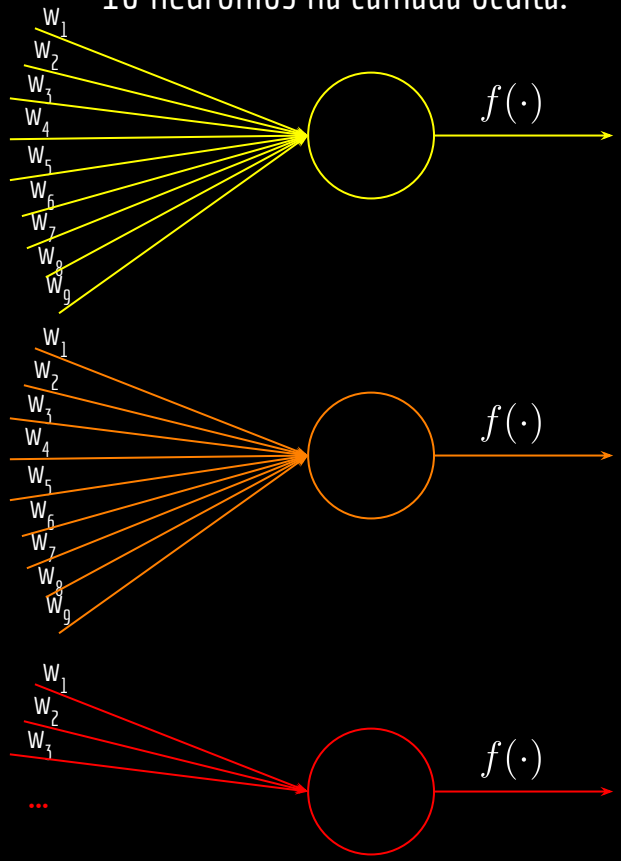


Pesos compartilhados

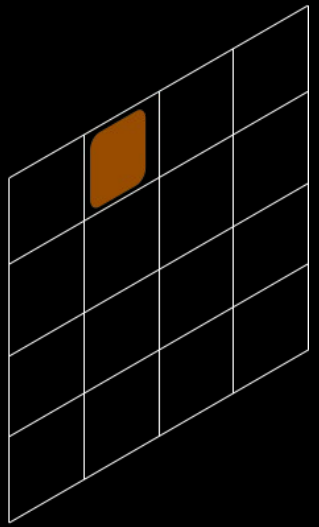
Entrada.



16 neurônios na camada oculta.

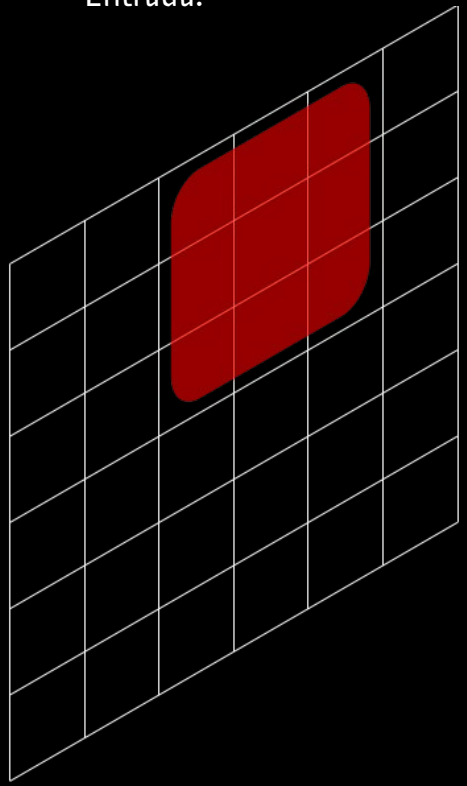


Feature Map de Saída.

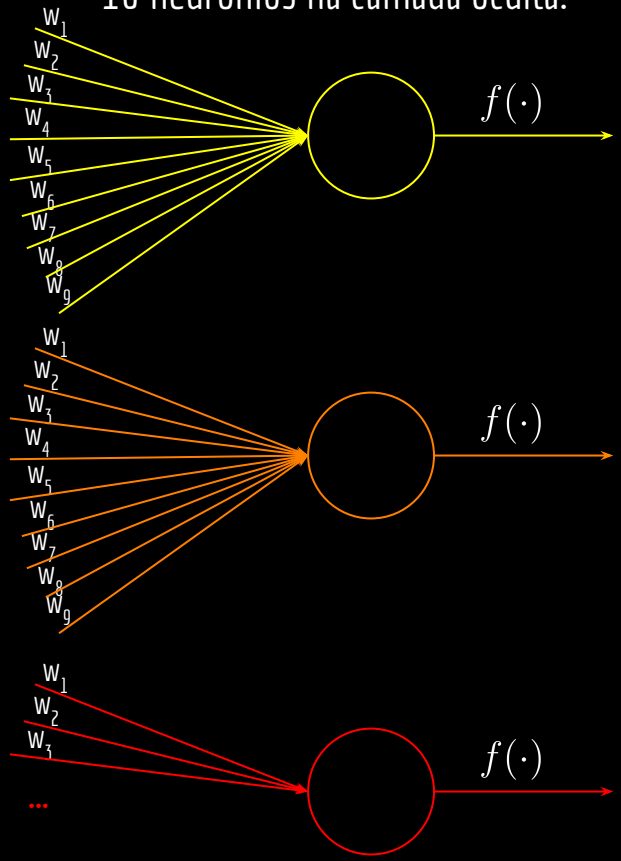


Pesos compartilhados

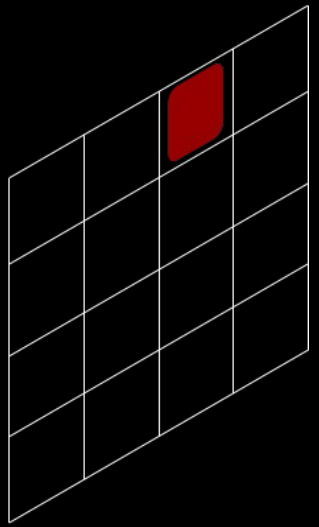
Entrada.



16 neurônios na camada oculta.

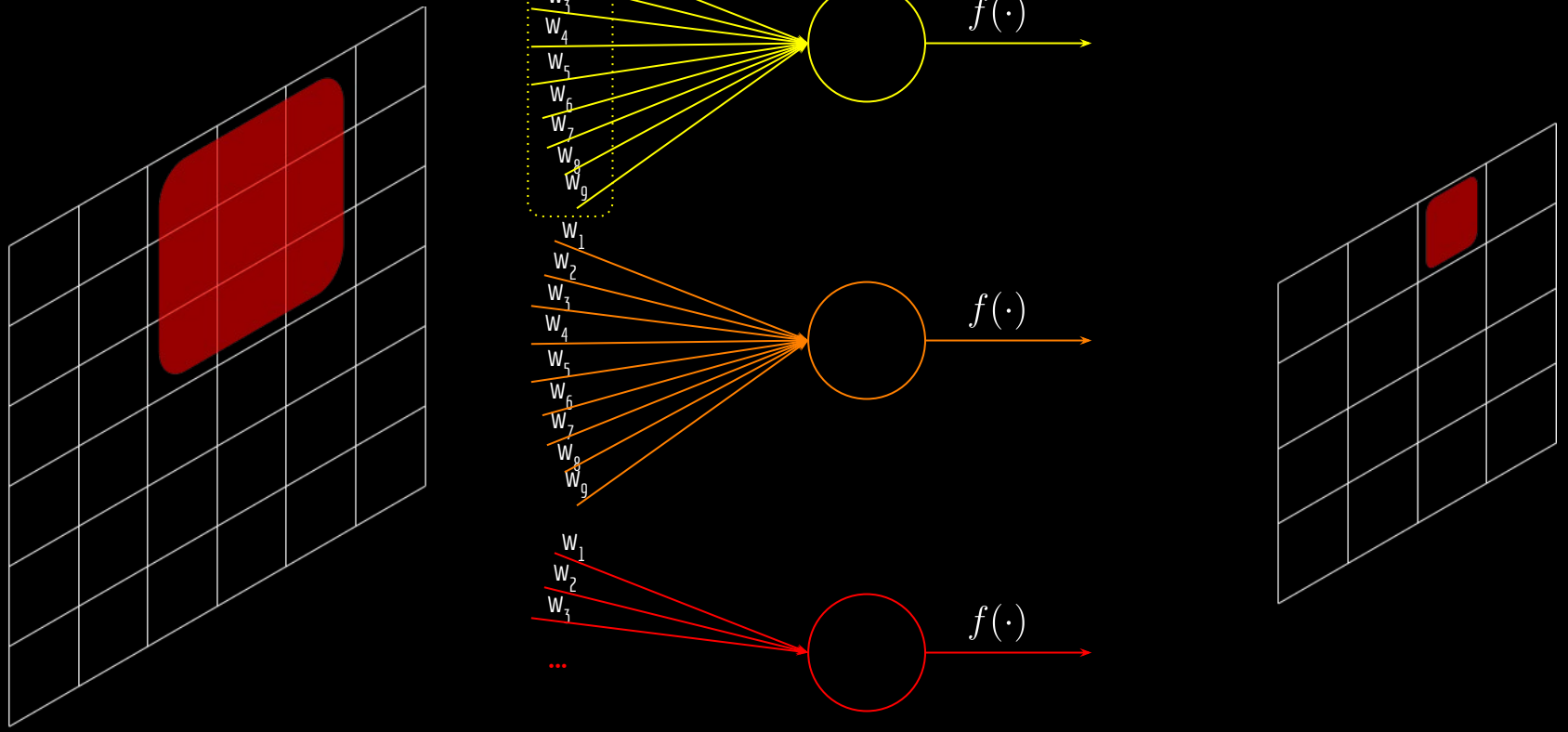


Feature Map de Saída.

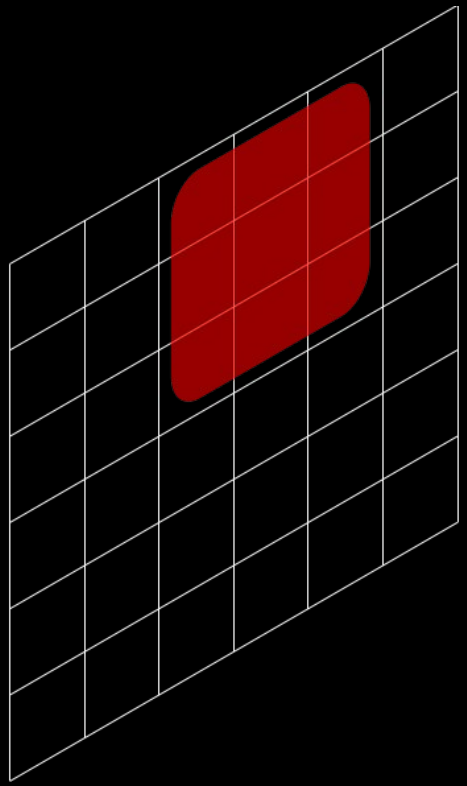


Pesos compartilhados

Considerando convoluções 3x3, todos neurônios compartilham os mesmos 9 pesos, e podem ser computados em paralelo.



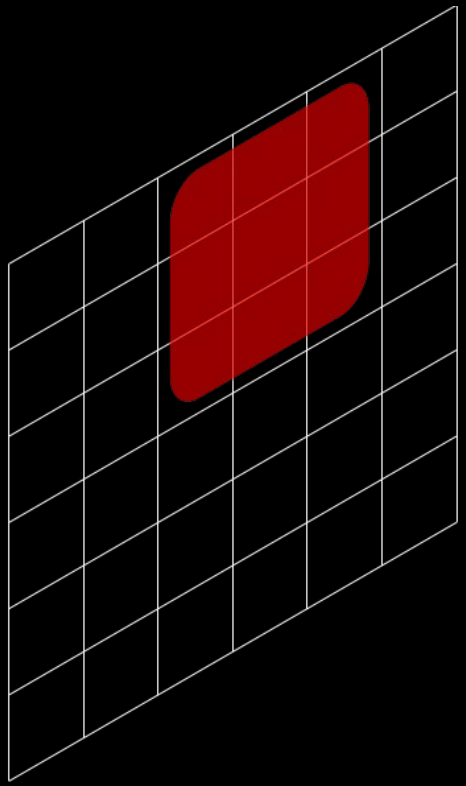
Faça você mesmo



Considerando a imagem de entrada do exemplo (de tamanho 6 x 6 e em escala de cinza). Quantos parâmetros serão necessários na primeira camada oculta considerando:

1. Uma rede convolucional, com um filtro de 3x3 na primeira camada.
   R: ?
2. Um MLP fully connected, com 16 neurônios na camada oculta;
   R: ?

Faça você mesmo



Considerando a imagem de entrada do exemplo (de tamanho 6 x 6 e em escala de cinza). Quantos parâmetros serão necessários na primeira camada oculta considerando:

1. Uma rede convolucional, com um filtro de 3x3 na primeira camada.
   R: 9
2. Um MLP fully connected, com 16 neurônios na camada oculta;
   R: 16 * 6 * 6 = 576

obs.: desconsiderando os biases.

Padding

Considerando:

Imagem de *L x A* pixels.

Filtro de convolução de *M x M*.

A imagem convoluída (*feature map* resultante) terá o tamanho *(L - M + 1) x (A - M + 1).*

Padding

Considerando:

Imagem de *L x A* pixels.

Filtro de convolução de *M x M*.

A imagem convoluída (*feature map* resultante) terá o tamanho *(L - M + 1) x (A - M + 1).*

Podemos usar técnicas de Padding para que a imagem resultante tenha o mesmo tamanho da imagem original.

O principal problema é escolher o valor de padding. Pesquise.

Padding - Exemplo

Imagem Original + Padding

0	0	0	0	0	0	0	0	0	0	0
0	58	40	35	20	18	17	20	23	16	0
0	120	62	55	48	36	33	32	25	19	0
0	197	202	65	50	39	40	48	23	32	0
0	227	226	100	70	68	66	45	36	20	0
0	250	248	250	200	69	50	48	50	30	0
0	189	265	249	221	198	55	38	36	19	0
0	217	221	199	200	202	98	48	29	28	0
0	217	221	250	221	198	98	97	24	15	0
0	217	265	265	221	221	221	199	18	36	0
0	228	197	197	221	224	223	248	189	180	0
0	227	227	197	221	249	248	247	240	190	0
0	0	0	0	0	0	0	0	0	0	0

Stride

Ao computar a convolução em determinada área da matriz, podemos “pular” mais que um Pixel no sentido horizontal ou vertical.

O tamanho do passo vertical ou horizontal que é dado é chamado de stride.

Strides maiores produzem feature maps menores.

Exemplo S = 2

Exemplo considerando um kernel *3 x 3* e um Stride *S = 2*.

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

Exemplo S = 2

Exemplo considerando um kernel *3 x 3* e um Stride *S = 2*.

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

Exemplo S = 2

Exemplo considerando um kernel *3 x 3* e um Stride *S = 2*.

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

Exemplo S = 2

Exemplo considerando um kernel *3 x 3* e um Stride *S = 2*.

58	40	35	20	18	17	20	23	16
120	62	55	48	36	33	32	25	19
197	202	65	50	39	40	48	23	32
227	226	100	70	68	66	45	36	20
250	248	250	200	69	50	48	50	30
189	265	249	221	198	55	38	36	19
217	221	199	200	202	98	48	29	28
217	221	250	221	198	98	97	24	15
217	265	265	221	221	221	199	18	36
228	197	197	221	224	223	248	189	180
227	227	197	221	249	248	247	240	190

Tensores

Escalar: um número qualquer. Por exemplo: $x \in \mathbb{R}$.

Tensores

Escalar: um número qualquer. Por exemplo: $x \in \mathbb{R}$.

Vetor: já estamos acostumados com vetores. Um vetor de números reais com *n* componentes: $\mathbf{x} \in \mathbb{R}^n$.

Tensores

Escalar: um número qualquer. Por exemplo: $x \in \mathbb{R}$.

Vetor: já estamos acostumados com vetores. Um vetor de números reais com *n* componentes: $\mathbf{x} \in \mathbb{R}^n$.

Matriz: uma matriz de duas dimensões, com *m* linhas e *n* colunas. Uma matriz de números reais: $\mathbf{A} \in \mathbb{R}^{m \times n}$.

Tensores

Escalar: um número qualquer. Por exemplo: $x \in \mathbb{R}$.

Vetor: já estamos acostumados com vetores. Um vetor de números reais com *n* componentes: $\mathbf{x} \in \mathbb{R}^n$.

Matriz: uma matriz de duas dimensões, com *m* linhas e *n* colunas. Uma matriz de números reais: $\mathbf{A} \in \mathbb{R}^{m \times n}$.

Tensor: podemos continuar adicionando dimensões. O caso geral é um *tensor*, que possui *k* dimensões. Um tensor de números reais pode ser definido como: $\mathbf{A} \in \mathbb{R}^{m \times n \times o \times p \times \ldots}$.

> Para indexar um tensor, comumente usamos a notação $A_{i,j,k,l,\ldots}$, onde *i,j,k,l, ...* são as coordenadas sendo acessadas em cada dimensão.
>
> Escalares, vetores e matrizes podem ser vistos como casos especiais de tensores.

Imagens RGB

Imagens coloridas geralmente possuem três canais: Red, Green e Blue.

Existem vários outros formatos, como HSI ou CMYK.

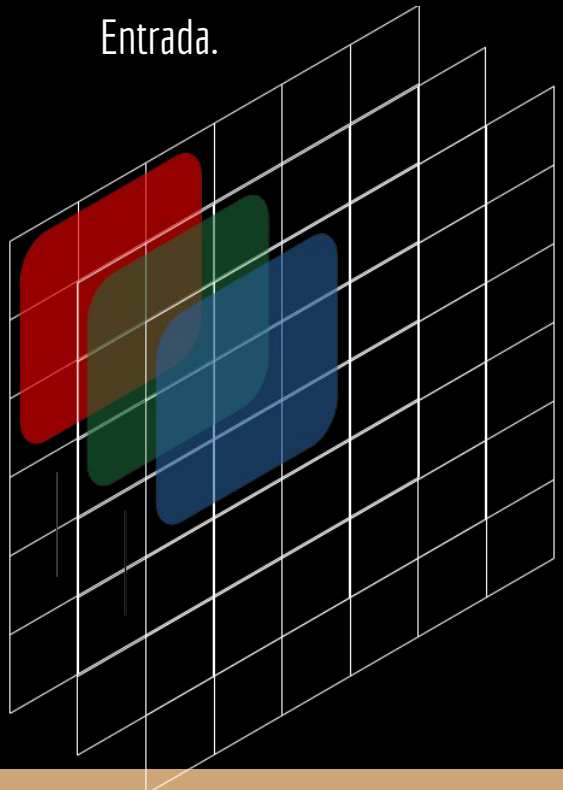
Uma imagem de tamanho *L x A*, com *C* canais de cores, pode ser interpretada como um tensor de dimensões *L x A* x *C*.

Podemos realizar convoluções nessas imagens utilizando filtros de convolução, que também são tensores, de tamanho *M x M x C*.

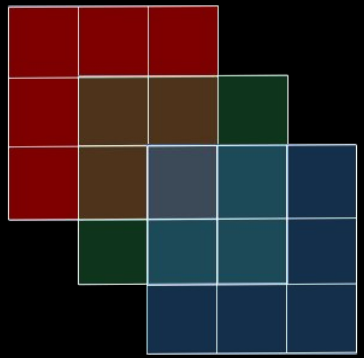
Obs.: Assumindo filtros quadrados, que são os mais comuns.

Convoluções Multidimensionais- Exemplo

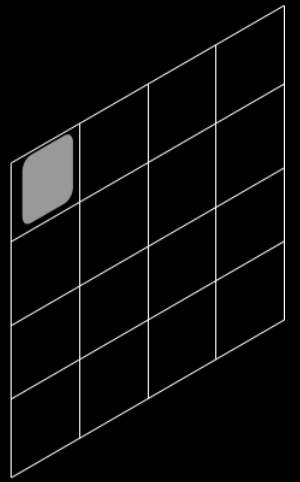
Entrada.



Kernel de convolução *3x3x3*.

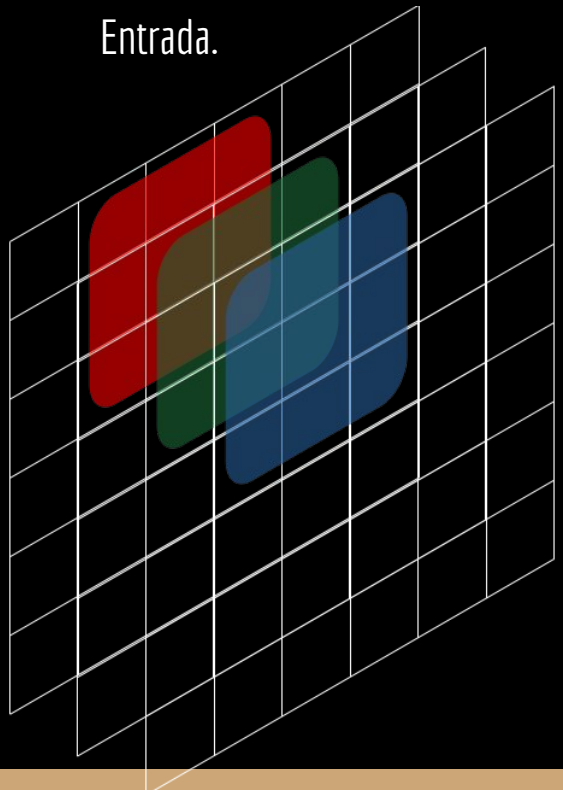


Feature Map de Saída.

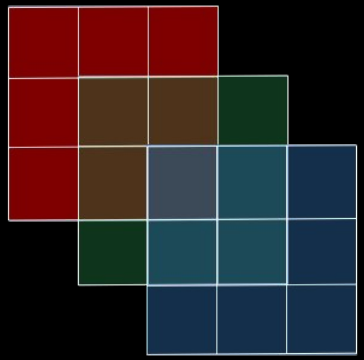


Convoluções Multidimensionais- Exemplo

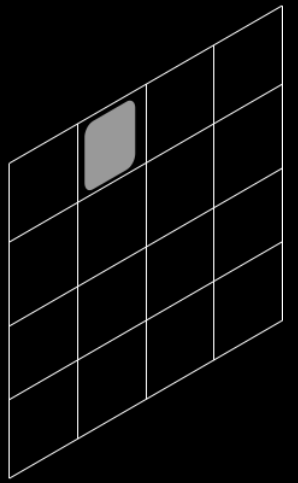
Entrada.



Kernel de convolução *3x3x3*.

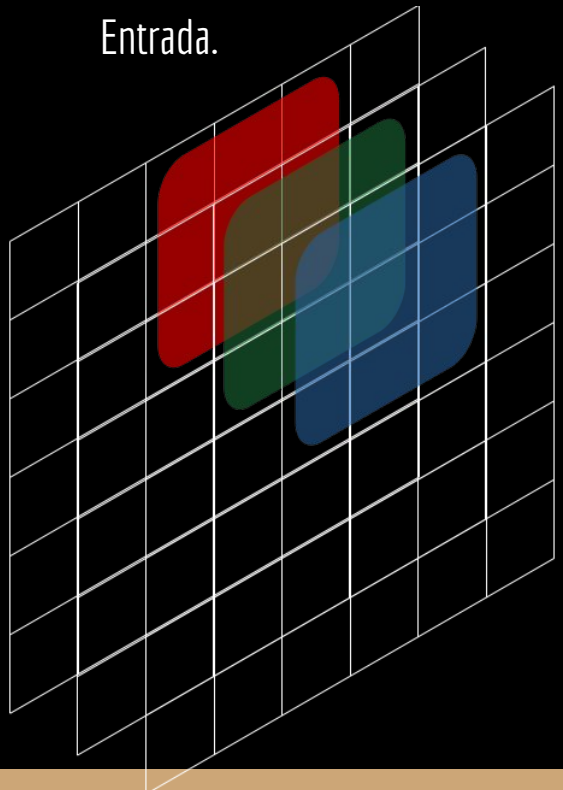


Feature Map de Saída.

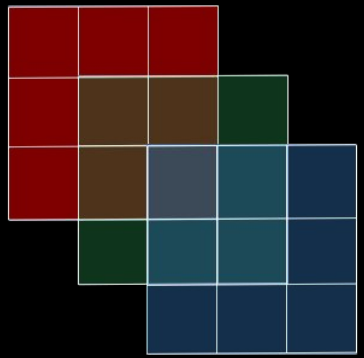


Convoluções Multidimensionais- Exemplo

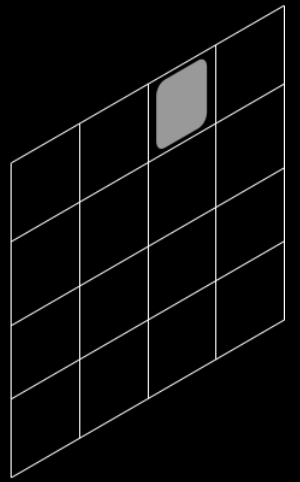
Entrada.



Kernel de convolução *3x3x3*.



Feature Map de Saída.



Múltiplos filtros

Um filtro de $M \times M$ que opera em C canais possui $M^2C + 1$ parâmetros.

Até agora estamos considerando um único filtro.

O equivalente a um único neurônio na camada oculta.

Múltiplos filtros

Um filtro de $M \times M$ que opera em C canais possui $M^2C + 1$ parâmetros.

Até agora estamos considerando um único filtro.

- O equivalente a um único neurônio na camada oculta.

Podemos adicionar múltiplos filtros.

- Cada filtro vai gerar seu próprio feature map, que comumente é chamado de canal de saída.

Múltiplos filtros

Com C_{cout} filtros, cada um de tamanho M x M operando em C canais de entrada, temos que os filtros podem ser representados por um tensor M x M x C x C_{cout}.

Dessa forma, temos $(M^2C + 1)C_{cout}$ parâmetros.

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120			

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120	55		

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120	55	36	

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120	55	36	32

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120	55	36	32
227			

Pooling - Exemplo

Feature map original.

58	40	35	20	18	17	20	23
120	62	55	48	36	33	32	25
197	202	65	50	39	40	48	23
227	226	100	70	68	66	45	36
250	248	250	200	69	50	48	50
189	265	249	221	198	55	38	36
217	221	199	200	202	98	48	29
217	221	250	221	198	98	97	24
217	265	265	221	221	221	199	18
228	197	197	221	224	223	248	189

Max-pooling 2x2.

120	55	36	32
227	100	68	48
250	250	198	50
221	250	202	97
228	265	224	248

Pooling

Não há pesos a serem aprendidos em uma camada de pooling.

Existem outros tipos de pooling, como o average pooling.

O pooling geralmente é aplicado em **cada canal separadamente**.

Ao aplicar pooling:

- Reduzimos a dimensionalidade da camada de saída.
- Adicionamos invariância a pequenas translações de posição dos objetos na imagem.
- Parte da informação posicional dos objetos é perdida.

Múltiplas camadas

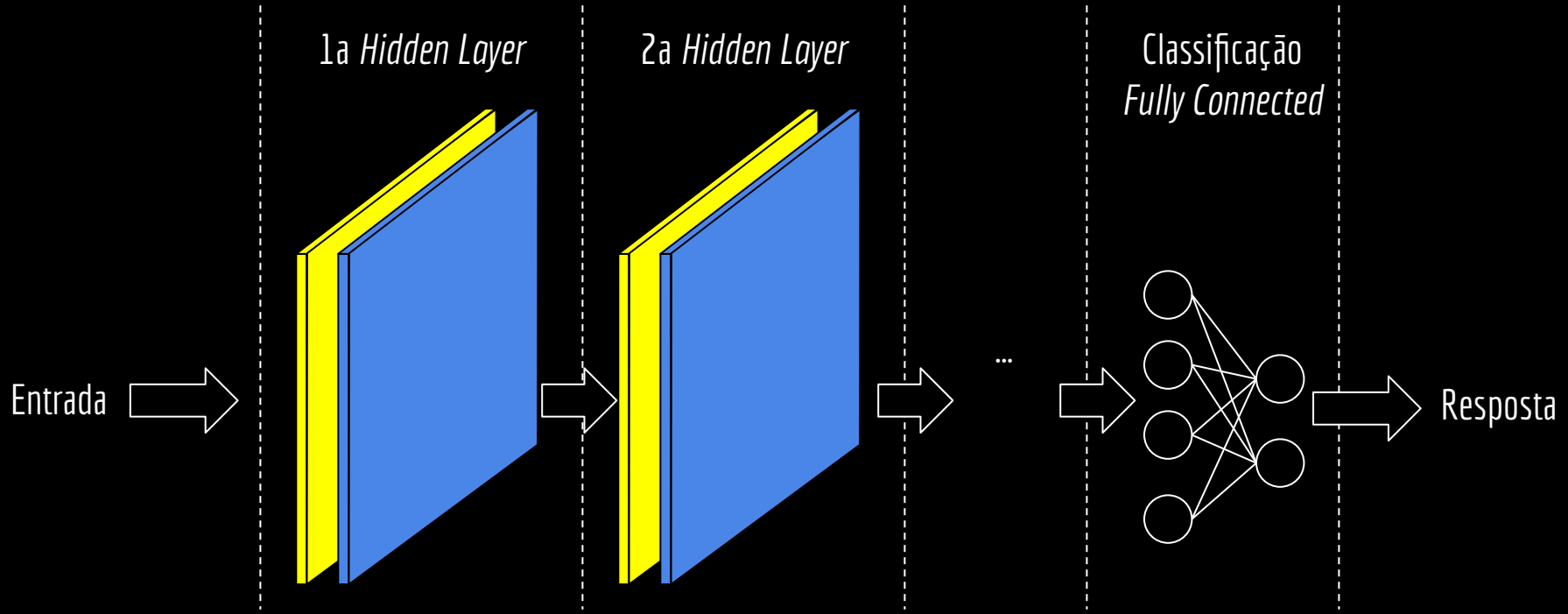
Podemos adicionar uma sequência de camadas de convolução, seguidas ou não de pooling, para criar uma rede profunda de múltiplas camadas.

Se a tarefa era de classificação, por exemplo, podemos usar uma camada fully connected na última camada, que irá usar os últimos *feature maps* da rede para tomar alguma decisão.

Múltiplas camadas

Camada Convolucional

Pooling (Opcional)

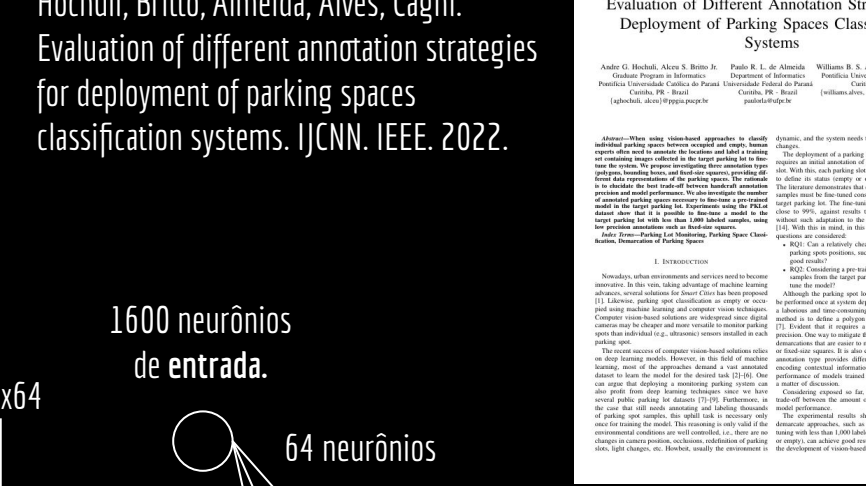


Exemplo

Convolucional filtro 3x3

Max-Pooling

Hochuli, Britto, Almeida, Alves, Cagni. Evaluation of different annotation strategies for deployment of parking spaces classification systems. IJCNN. IEEE. 2022.



Evaluation of Different Annotation Strategies for Deployment of Parking Spaces Classification Systems

André G. Hochuli, Alceu S. Britto Jr.
Graduate Program in Informatics
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{aghochuli, alceu}@ppgia.pucpr.br

Paulo R. L. de Almeida
Department of Informatics
Universidade Federal do Paraná
Curitiba, PR - Brazil
paulorla@ufpr.br

Williams B. S. Alves, Fábio M. C. Cagni
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brazil
{williams.alves, fabio.cagni}@pucpr.edu.br

***Abstract*—When using vision-based approaches to classify individual parking spaces between occupied and empty, human experts often need to annotate the locations and label a training set containing images collected in the target parking lot to fine-tune the system. We propose investigating three annotation types (poly-gons, bounding boxes, and fixed-size squares), providing different data representations of the parking spaces. The rationale is to elucidate the best trade-off between handcraft annotation precision and model performance. We also investigate the number of annotated parking spaces necessary to fine-tune a pre-trained model in the target parking lot. Experiments using the PKLot dataset show that it is possible to fine-tune a model to the target parking lot with less than 1,000 labeled samples, using low precision annotations such as fixed-size squares.**

***Index Terms*—Parking Lot Monitoring, Parking Space Classification, Demarcation of Parking Spaces**

I. Introduction

Nowadays, urban environments and services need to become innovative. In this vein, taking advantage of machine learning advances, several solutions for *Smart Cities* has been proposed [1]. Likewise, parking spot classification as empty or occupied using machine learning and computer vision techniques. Computer vision-based solutions are widespread since digital cameras may be cheaper and more versatile to monitor parking spots than individual (e.g., ultrasonic) sensors installed in each parking spot.

The recent success of computer vision-based solutions relies on deep learning models. However, in this field of machine learning, most of the approaches demand a vast annotated dataset to learn the model for the desired task [2]–[6]. One can argue that deploying a monitoring parking system can also profit from deep learning techniques since we have several public parking lot datasets [7]–[9]. Furthermore, in the case that still needs annotating and labeling thousands of parking spot samples, this uphill task is necessary only once for training the model. This reasoning is only valid if the environmental conditions are well controlled, i.e., there are no changes in camera position, occlusions, redefinition of parking slots, light changes, etc. Howbeit, usually the environment is dynamic, and the system needs to be updated frequently to fit changes.

The deployment of a parking lot monitoring system usually requires an initial annotation of the boundary of each parking slot. With this, each parking slot can be cropped and classified to define its status (empty or occupied) [7], [8], [10]–[13]. The literature demonstrates that even a model trained on many samples must be fine-tuned considering a set of images of the target parking lot. The fine-tuning process reports accuracies close to 99%, against results that are often less than 95% without such adaptation to the target domain [7], [8], [13], [14]. With this in mind, in this work, the following research questions are considered:

- RQ1: Can a relatively cheap approach to demarcate the parking spots positions, such as bounding boxes, provide good results?
- RQ2: Considering a pre-trained model, how many labeled samples from the target parking lot are necessary to fine-tune the model?

Although the parking spot locations demarcation needs to be performed once at system deployment or maintenance, it is a laborious and time-consuming task. A standard annotation method is to define a polygon bounding each parking spot [7]. Evident that it requires a high level of attention and precision. One way to mitigate this cost is to use low-precision demarcations that are easier to make, such as bounding boxes or fixed-size squares. It is also essential to consider that each annotation type provides different representations, such as encoding contextual information of neighborhoods. So, the performance of models trained with these representations is a matter of discussion.

Considering exposed so far, we also investigate the best trade-off between the amount of labeled data necessary and model performance.

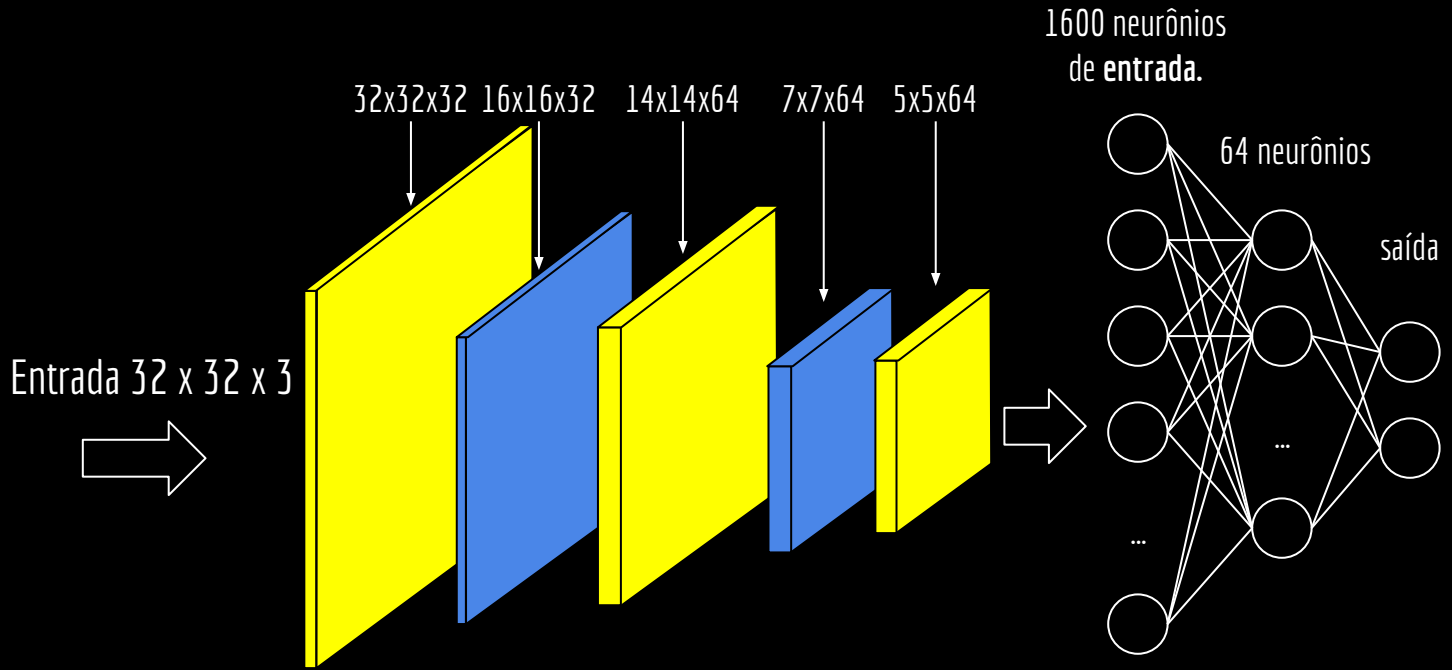
The experimental results show that combining easy to demarcate approaches, such as bounding boxes, and a fine-tuning with less than 1,000 labeled samples (between occupied or empty), can achieve good results. Our findings may lead to the development of vision-based systems that are accurate and

Pergunta

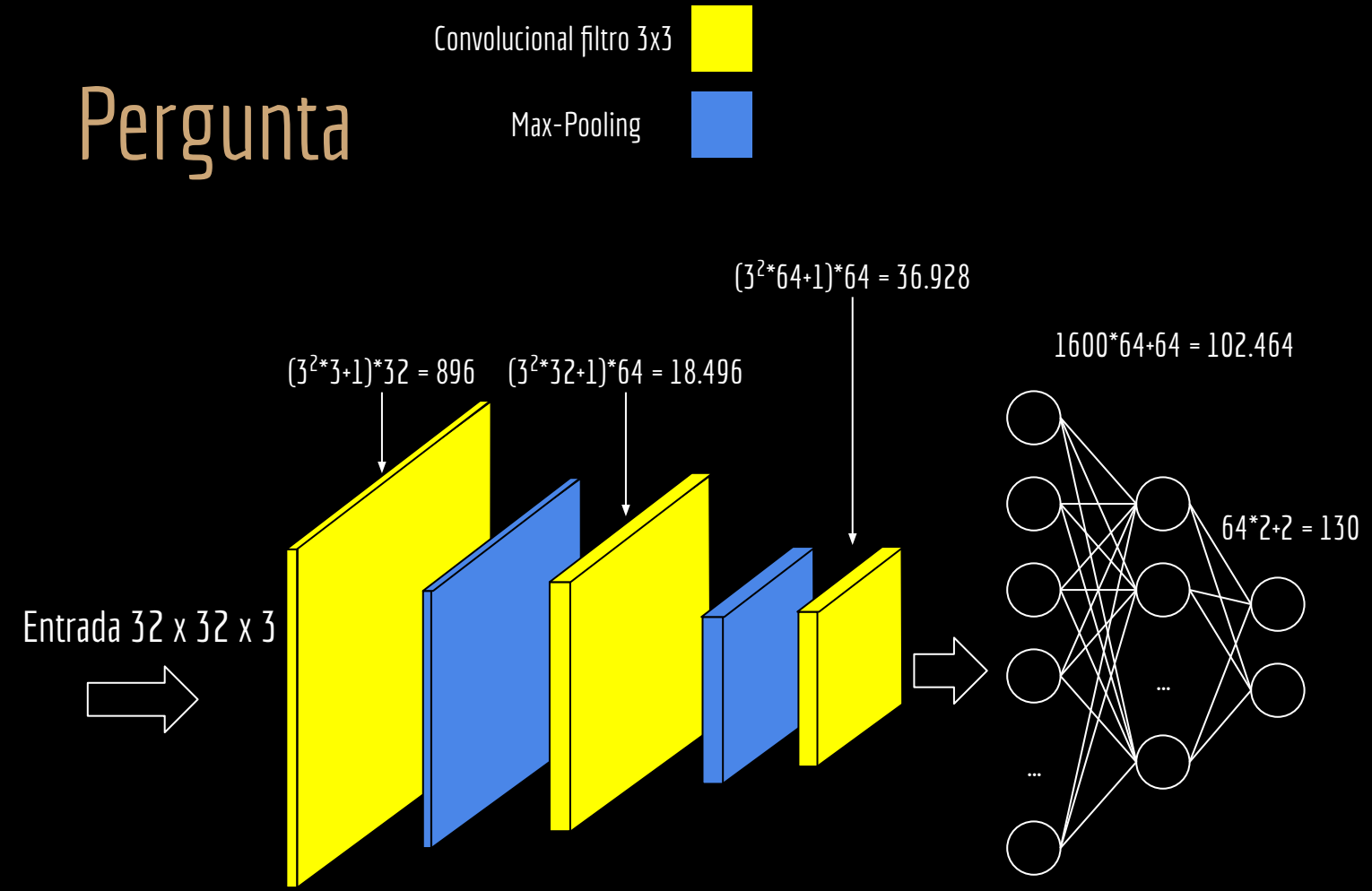
Convolucional filtro 3x3

Max-Pooling

Quantos parâmetros (pesos) essa rede possui?



Pergunta



Total: 158.914.

Exercícios

1. Execute a rede convolucional para classificação de vagas disponibilizada no Google Colab.
    a. Faça testes modificando, por exemplo, o tamanho dos filtros de convolução.

Referências

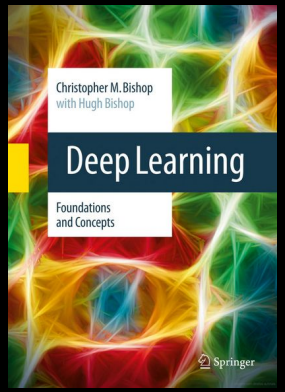
Bishop, C. M., Bishop, H. Deep Learning: Foundations and Concepts. 2023.

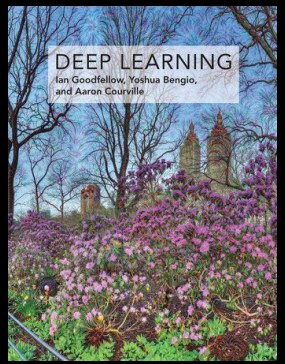
Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. 2016.

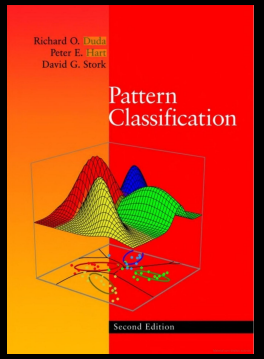
Duda, R. O., Hart, P. E., Stork, D. G. Pattern Classification. 2012.

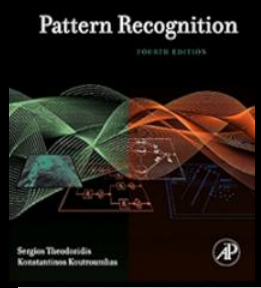
Theodoridis, S., Koutroumbas, K. Pattern Recognition & Matlab Intro. 2010.

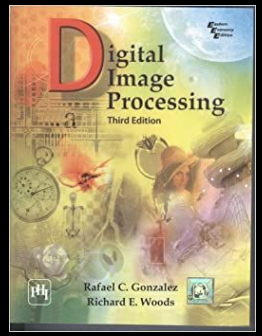
Gonzalez, R. C., Woods, R. E. Digital image processing. 2008.











Licença