

VI Escola Avançada de Big Data Analysis

Redes Convolucionais

Prof. Ricardo Cerri
Departamento de Computação
Universidade Federal de São Carlos

De Camadas Totalmente Conectadas para Convoluções

Redes totalmente conectadas são mais adequadas para dados tabulados

- 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

De Camadas Totalmente Conectadas para Convoluções

Em Redes Neurais totalmente conectadas, o número de parâmetros para serem aprendidos pode ser proibitivo

- Considere um problema de distinguir entre imagens de gatos e cachorros
- Considere que cada imagem tem 1 megapixel
- Assim, cada exemplo de entrada na rede neural vai ter 10^6 dimensões
- Mesmo que diminuamos drasticamente a dimensão da camada escondida, por exemplo 10^3 neurônios, ainda assim teremos que aprender:
- Número de parâmetros = $10^6 \times 10^3 = 10^9$ pesos

De Camadas Totalmente Conectadas para Convoluções

Sabemos então que imagens possuem uma estrutura que pode ser explorada tanto por nós humanos quanto pelos modelos de aprendizado de máquina.

- Redes Neurais Convolucionais são uma maneira criativa de explorar essas estruturas conhecidas em imagens
- Redes Neurais Convolucionais: Redes neurais com camadas de convolução
- Convolução (basicamente): multiplica uma matriz de pixels com uma matriz chamada de filtro ou 'kernel', e soma os valores da multiplicação

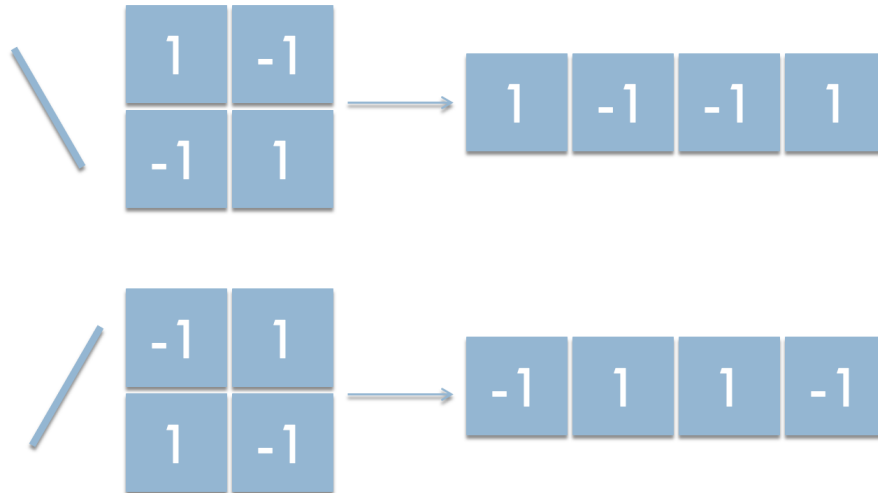
De Camadas Totalmente Conectadas para Convoluções

Redes Neurais Convolucionais são um tipo especial de Redes Neurais Artificiais que contêm camadas de convolução

- Em processamento de imagens, camadas convolucionais tipicamente requerem bem menos parâmetros do que camadas totalmente conectadas
- Há localidade, o que significa que somente uma pequena vizinhança de pixels é utilizada para calcular a correspondente representação escondida

Convolução

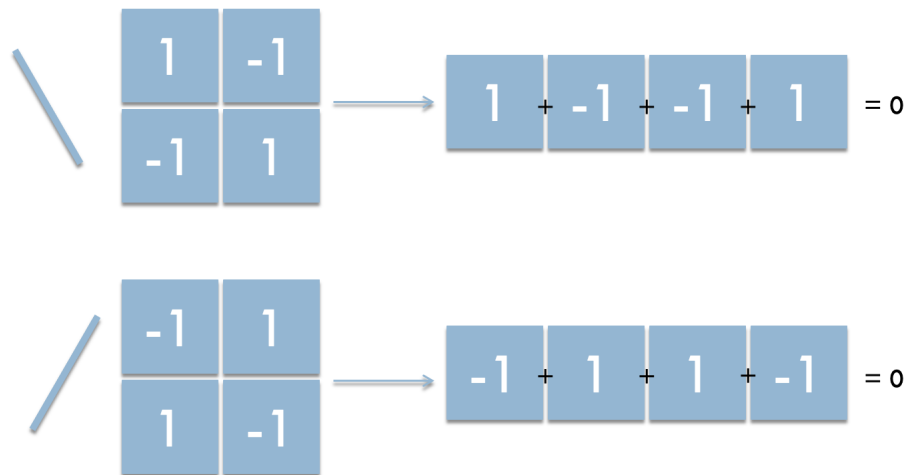
De maneira simplificada, vamos representar como um computador “enxerga” uma imagem



Convolução

Como podemos diferenciar as imagens?

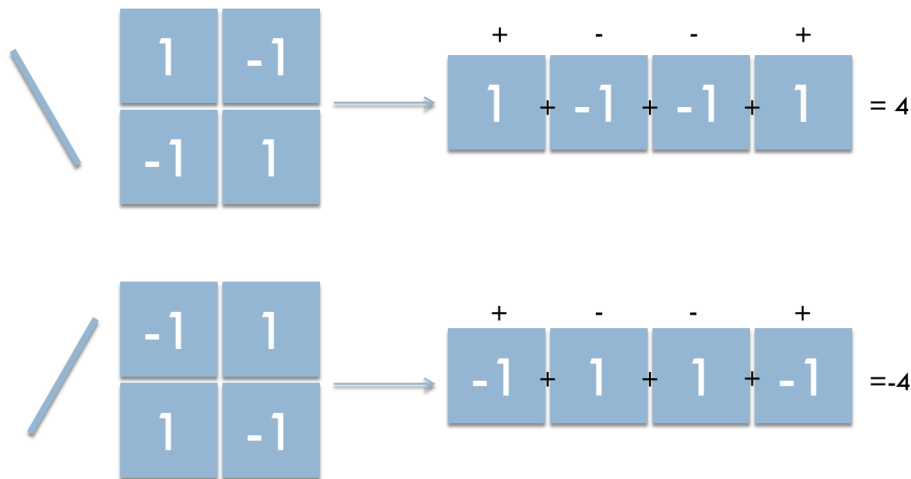
- Existem hipóteses ruins!



Convolução

Como podemos diferenciar as imagens?

- Existem hipóteses boas!



Convolução

Como podemos diferenciar as imagens?

- Com quatro dígitos, temos $2^4 = 16$ possibilidades para encontrar uma boa hipótese

×	×	×	×	×	×	×	✓
-	-	+	-	-	-	+	+
-	-	-	-	+	-	-	+

✓	×	×	×	×	×	×	×
-	+	-	+	+	+	+	+
+	-	-	+	+	-	+	+

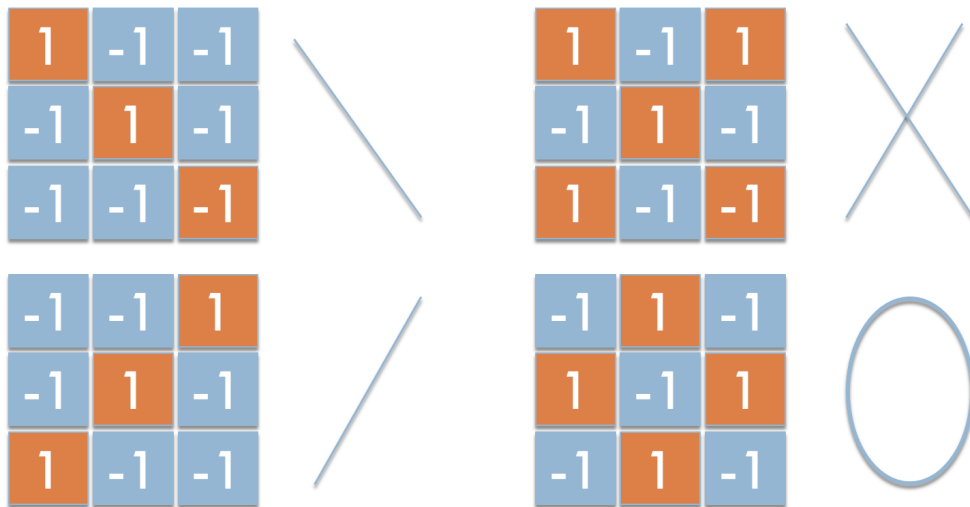
Convolução

Como podemos diferenciar as imagens?

- Com quatro dígitos, temos $2^4 = 16$ possibilidades para encontrar uma boa hipótese
- E se tivermos muitos mais dígitos?
- E se ao invés de valores -1 e +1 para os pesos, usarmos valores reais? 0.5, 1.2, -0.6, etc?

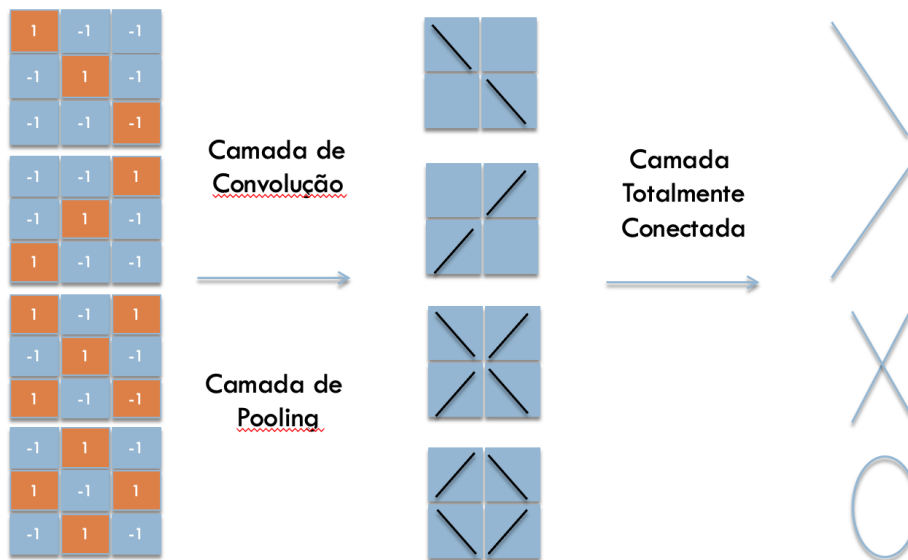
Convolução

Vamos para um cenário um pouco mais complicado



Convolução

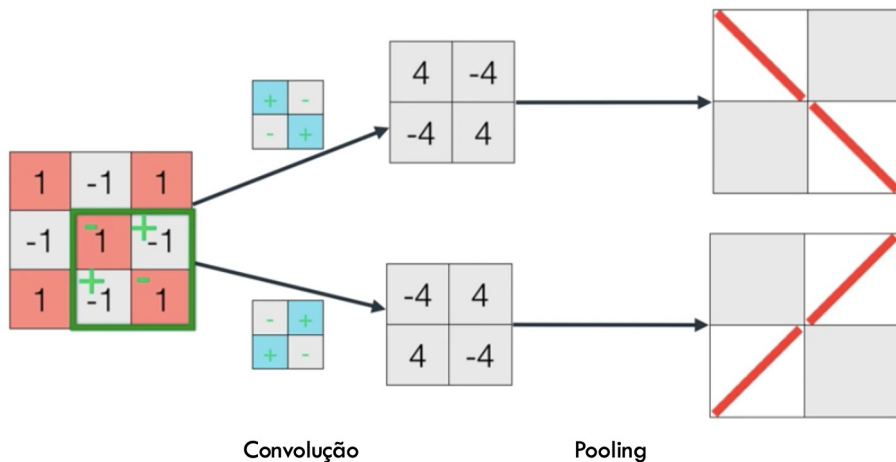
Com a hipótese correta, podemos reconhecer partes das imagens.



Convolução

Como se dá o Reconhecimento?

- Aplicamos a nossa hipótese nas imagens



Convolução

- Cada região lida na imagem de entrada é chamada de *local receptive field*
- Para cada região, temos um neurônio da camada escondida
- Nessa região também é aplicada uma função de ativação

Convolução

- O mapeamento da camada de entrada para a escondida também é chamado de *feature map*
- Os pesos definindo o *feature map* são chamados de *shared weights*
- Esses pesos (*shared weights*) definem um *kernel* ou filtro

Convolução

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:

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

Convolução

A vantagem dos pesos compartilhados é a redução do número de parâmetros

- Para cada filtro de 5x5 precisamos de 25 pesos mais o bias. Se utilizarmos 26 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 $784 \times 30 + 30 = 23550$ parâmetros
- Vamos para mais exemplos!

Convolução

Considere como entrada uma matriz 3x3, um filtro (kernel) de dimensão 2x2

- Começamos com o filtro posicionado no canto superior esquerdo da matriz de entrada
- Deslizamos através da matriz de entrada, da esquerda para a direita e de cima para baixo
- Quando em uma determinada posição, a submatriz de entrada contida nessa posição e o filtro são multiplicados elemento a elemento
- A matriz resultante é somada produzindo um único valor escalar.

Input		Kernel		Output																	
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	*	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	=	<table border="1"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43
0	1	2																			
3	4	5																			
6	7	8																			
0	1																				
2	3																				
19	25																				
37	43																				

Convolução

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

Convolução

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)

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)

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)

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

Kernel Channel #1



308

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

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25



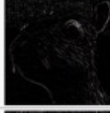

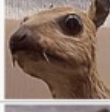


↑
Bias = 1

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

Output

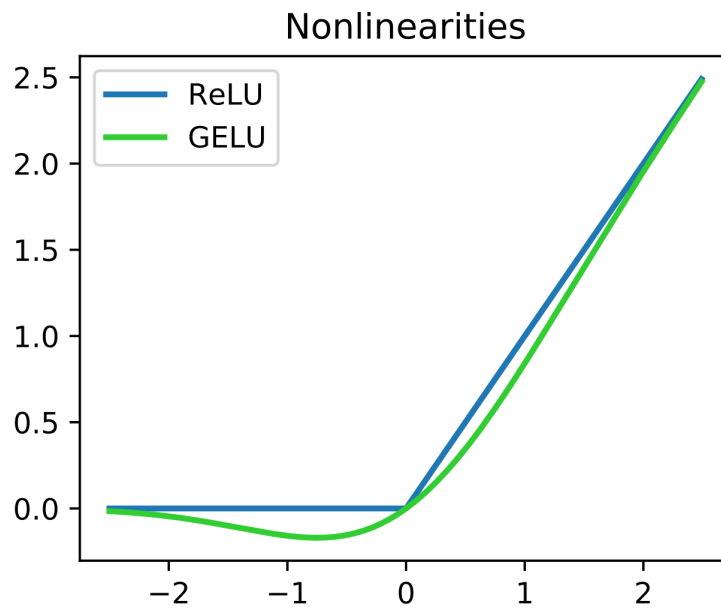
Convolução

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)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

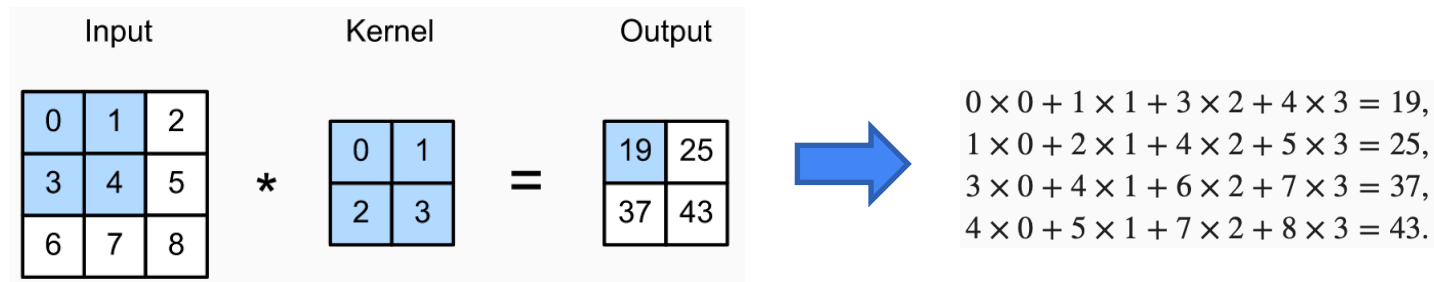
Convolução

Função de ativação



Convolução

Considere como entrada uma matriz 3x3, um filtro (kernel) de dimensão 2x2



- Veja que a dimensão da saída é dada pela dimensão da entrada ($n_h - n_w$) menos a dimensão do filtro ($k_h - k_w$), da seguinte maneira:

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

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

- A saída correspondente é aumentada para uma dimensão 4 x 4

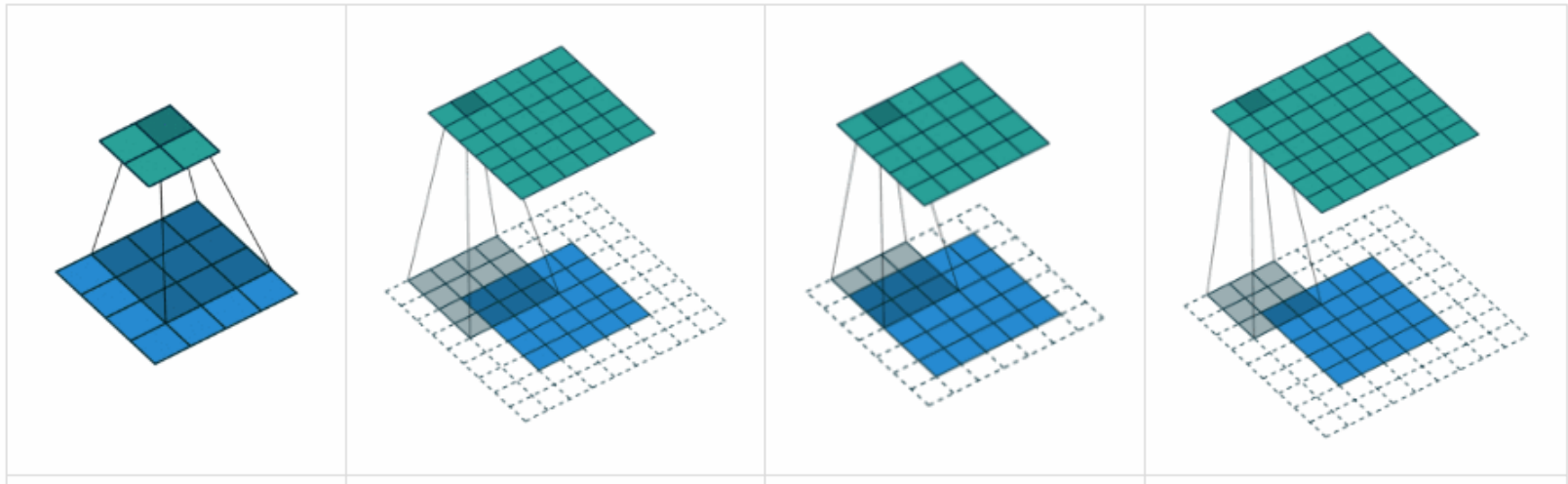
Input		Kernel		Output																																													
<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>0</td></tr><tr><td>0</td><td>3</td><td>4</td><td>5</td><td>0</td></tr><tr><td>0</td><td>6</td><td>7</td><td>8</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	1	2	0	0	3	4	5	0	0	6	7	8	0	0	0	0	0	0	*	<table><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	=	<table><tr><td>0</td><td>3</td><td>8</td><td>4</td></tr><tr><td>9</td><td>19</td><td>25</td><td>10</td></tr><tr><td>21</td><td>37</td><td>43</td><td>16</td></tr><tr><td>6</td><td>7</td><td>8</td><td>0</td></tr></table>	0	3	8	4	9	19	25	10	21	37	43	16	6	7	8	0
0	0	0	0	0																																													
0	0	1	2	0																																													
0	3	4	5	0																																													
0	6	7	8	0																																													
0	0	0	0	0																																													
0	1																																																
2	3																																																
0	3	8	4																																														
9	19	25	10																																														
21	37	43	16																																														
6	7	8	0																																														

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:

- $(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$
- Assim, a altura e a largura da saída são aumentadas em p_h e p_w respectivamente

Padding



Stride

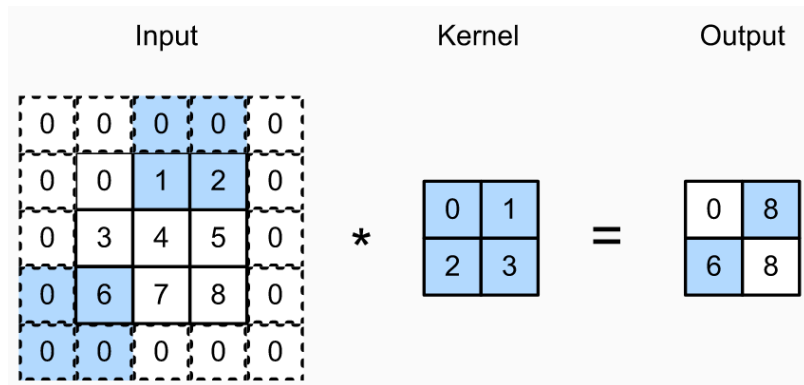
Quando computando a convolução, começamos com a janela de convolução no canto superior esquerdo da matriz de entrada e, em seguida, deslizamos sobre todos os locais da matriz para baixo e para a direita

- Por padrão, deslizamos o filtro um elemento por vez
- No entanto, às vezes, seja por eficiência computacional ou porque desejamos reduzir a resolução, deslizamos o filtro mais de um elemento por vez, pulando os locais intermediários

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

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:

- $\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$
- 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 $3 \times h \times w$. Esse eixo de tamanho 3 é definido como a dimensão canal.

Múltiplos Canais de Entrada

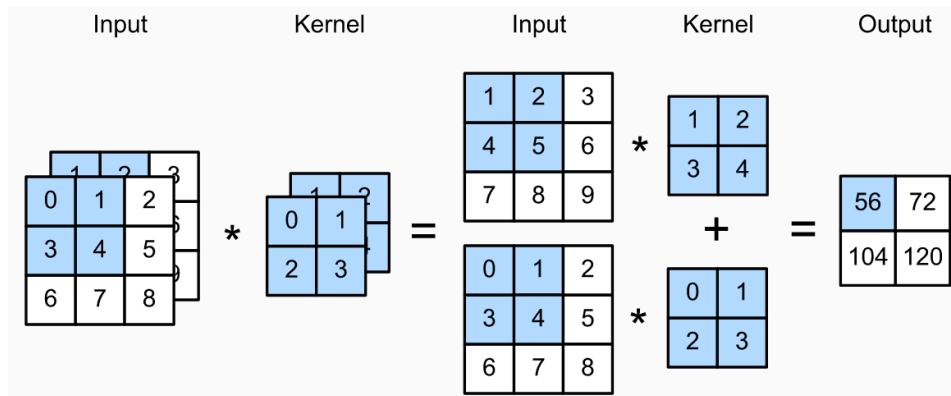
Quando os dados de entrada contêm vários canais, precisamos construir um kernel de convolução com o mesmo número de canais de entrada que os dados de entrada.

- 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 somados, resultando na saída bidimensional

Múltiplos Canais de Entrada

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.

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



Múltiplos Canais de Saída

Independentemente do número de canais de entrada, até agora sempre acabamos com um canal de saída. No entanto, nas arquiteturas mais populares, na verdade aumentamos a dimensão do canal conforme a camada da rede neural fica mais profunda, normalmente reduzindo 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.

Múltiplos Canais de Saída

- 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

Pooling

Muitas vezes, à medida que processamos imagens, queremos reduzir gradualmente a resolução espacial das representações ocultas, agregando informações de forma que quanto mais profunda a rede, maior o *receptive field* ao qual cada nó oculto é sensível

- Geralmente queremos responder a uma pergunta mais global, por exemplo, se a imagem contém um gato. Assim, as unidades da camada de saída devem ser sensíveis a toda a entrada
- Ao agregar informações gradativamente, produzindo mapas cada vez menores, alcançamos esse objetivo de, em última análise, aprender uma representação global, enquanto mantemos todas as vantagens das camadas convolucionais nas camadas intermediárias de processamento
- Ajuda as representações a sejam um tanto invariantes à translação

Pooling

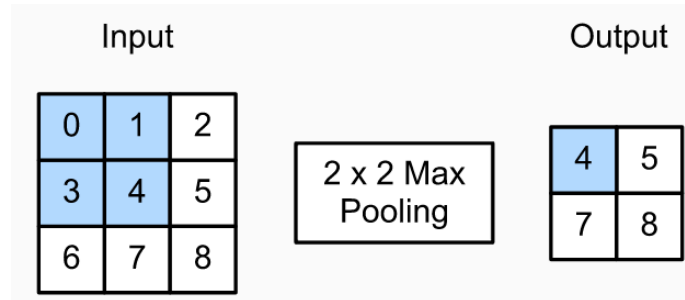
Operadores de pooling consistem em uma janela de formato fixo que é deslizada sobre todas as regiões na entrada de acordo o stride, computando uma única saída para cada região atravessada pela janela de formato fixo.

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

Pooling

Exemplo: max pooling de altura 2 e largura 2. Os quatro elementos da saída são o valor máximo em cada janela de pooling

- $\max(0,1,3,4) = 4$
- $\max(1,2,4,5) = 5$
- $\max(3,4,6,7) = 7$
- $\max(4,5,7,8) = 8$



Pooling

A camada de pooling sumariza os atributos presentes em uma região do feature map gerado por uma camada de convolução. Portanto, outras operações são realizadas em atributos sumarizados em vez de atributos precisamente posicionados gerados pela camada de convolução. Isso torna o modelo mais robusto a variações na posição dos atributos na imagem de entrada.

- Reduz a quantidade de parâmetros a serem aprendidos e quantidade de computação



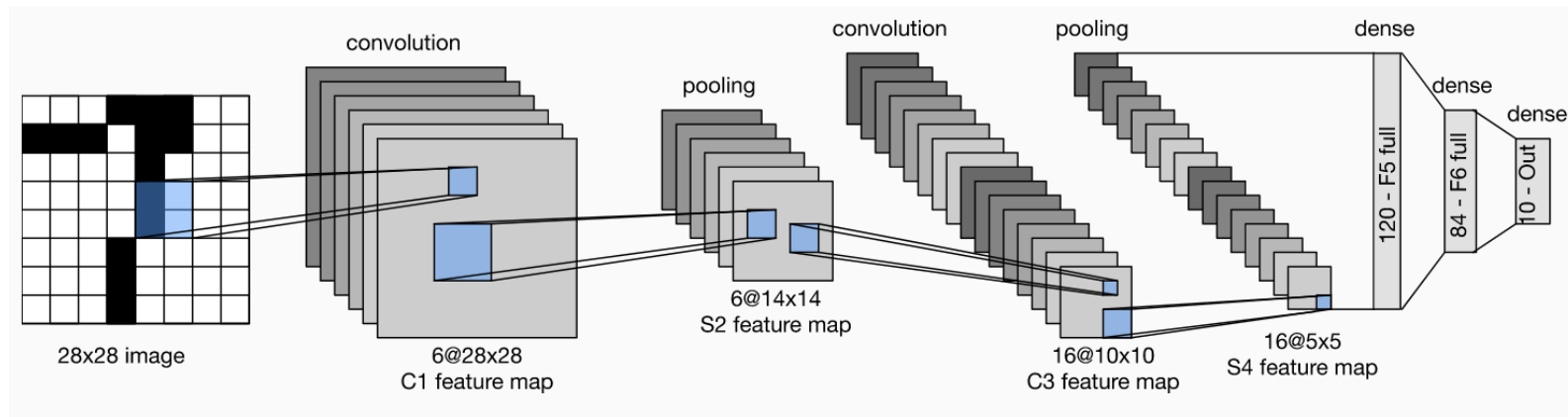
Montando uma Rede Neural Convolucional

Como exemplo de uma Rede Neural Convolucional (CNN) completa, 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

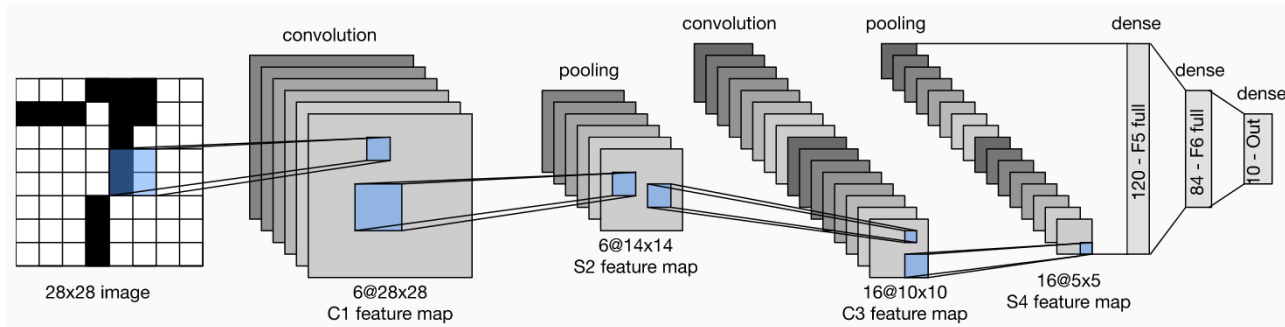
De maneira geral, a LeNet (LeNet-5) possui duas partes: i) um codificador convolucional, consistindo de duas camadas de convolução; e ii) um bloco denso, consistindo de três camadas completamente conectadas



Montando uma CNN - LeNet

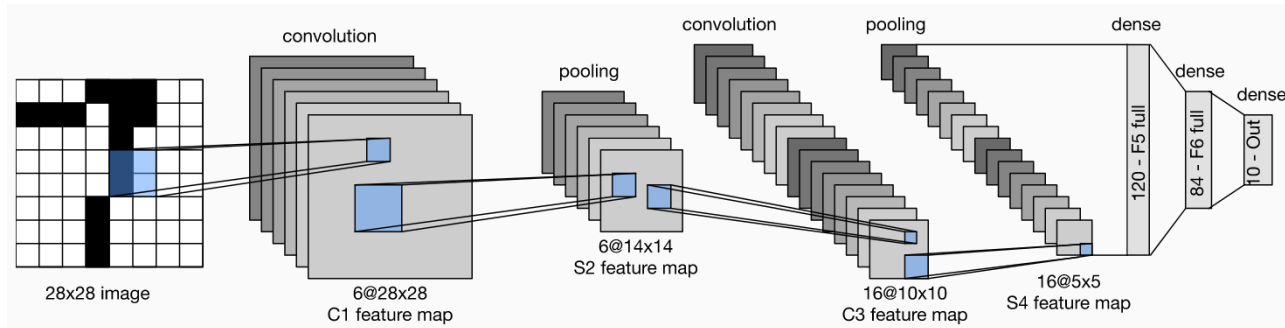
Unidades básicas em cada bloco de convolução

- 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



Montando uma CNN - LeNet

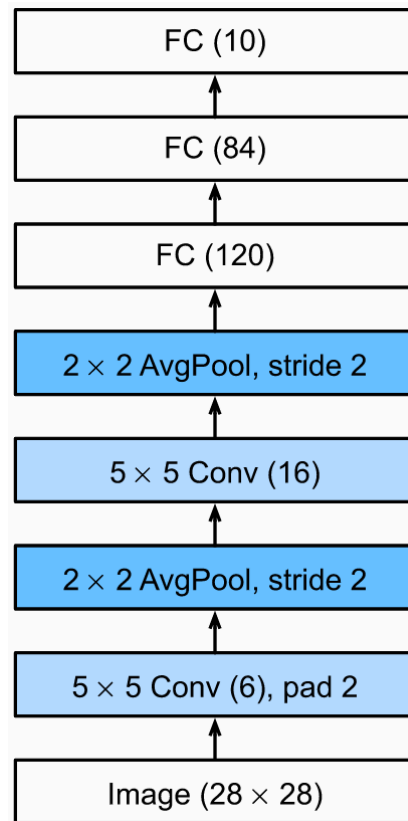
- A saída do bloco de convolução passa por um processo de transformação (flatten), em que cada exemplo é transformado em um vetor
- O bloco denso tem 3 camadas totalmente conectadas com 120, 84 e 10 saídas. Na última camada, 10 corresponde ao número de classes do problema



Montando uma CNN - LeNet

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



Redes Neurais Convolucionais - Sumário

- Uma CNN é uma rede neural que usa camadas convolucionais
- Em uma CNN, intercalamos convoluções, não linearidades e (frequentemente) operações de pooling
- Em uma CNN, as camadas convolucionais são normalmente organizadas de modo que diminuam gradualmente a resolução espacial das representações, enquanto aumentam o número de canais
- Em CNNs tradicionais, as representações codificadas pelos blocos convolucionais são processadas por uma ou mais camadas totalmente conectadas antes de emitir a saída

Perguntas?

