

Projeto Final - ALA (Images Convolution)

Rio de Janeiro - Julho de 2022

Autor

Nome: Riquelme Freitas Gomes

DRE: 120032785

Disciplina

Nome: Álgebra Linear Algorítmica (ALA)

Professor: João Antonio Recio Da Paixão

Convolução

A palavra “convolução” soa como um termo sofisticado e complicado, mas na verdade não é. Se você já trabalhou com visão computacional, processamento de imagens ou OpenCV antes, você já aplicou convoluções, quer você perceba ou não.

Já aplicou desfoque ou suavização? Sim, isso é uma convolução. E a detecção de borda? Sim, convolução. Você abriu o Photoshop ou o GIMP para aprimorar uma imagem? Você adivinhou, convolução.

As convoluções são um dos blocos de construção mais críticos e fundamentais na visão computacional e no processamento de imagens. Porém o termo em si tende a assustar as pessoas – na verdade, no primeiro momento, a palavra até parece ter uma conotação negativa. No entanto, a convolução é um conceito bem simples de ser entendido.

Na realidade, uma convolução (de imagem) é simplesmente uma multiplicação elementar de duas matrizes seguida por uma soma.

Sério. É isso. Você acabou de aprender o que é convolução:

1. Pegue duas matrizes (ambas com as mesmas dimensões).
2. Multiplique-as, elemento por elemento (ou seja, não o produto escalar, apenas uma simples multiplicação).
3. Some os resultados.

Pense desta forma: uma imagem é apenas uma matriz multidimensional. Nossa imagem tem uma largura (número de colunas) e uma altura (número de linhas), assim como uma matriz.

Mas, ao contrário das matrizes tradicionais com as quais trabalhamos na escola, as imagens também têm uma profundidade nelas – o número de canais na imagem. Para uma imagem RGB padrão, temos uma profundidade de 3 – um canal para cada um dos canais Vermelho, Verde e Azul, respectivamente.

A partir disso, podemos pensar em uma **imagem** como uma **grande matriz** e **kernel** (ou matriz convolucional) como **uma pequena matriz** que é usada para desfoque, nitidez, detecção de borda e outras funções de processamento de imagem.

Essencialmente, esse minúsculo kernel fica em cima da imagem grande e desliza da esquerda para a direita e de cima para baixo, aplicando uma operação matemática (isto é, uma convolução) em cada coordenada (x, y) da imagem original.

É normal definir manualmente os kernels para obter várias funções de processamento de imagem. Na verdade, você já deve estar familiarizado com desfoque (suavização média, suavização gaussiana, suavização mediana, etc.), detecção de bordas (Laplacian, Sobel, Scharr, Prewitt, etc.) e sharpening.

Todas essas operações são formas de kernels definidos à mão que são projetados especificamente para executar uma função específica.

Kernels

Novamente, vamos pensar em uma imagem como uma grande matriz e um kernel como uma pequena matriz (pelo menos em relação à imagem original da “grande matriz”):

131	162	232	84	91	207
104	-1	0	+1	237	109
243	-2	0	+2	135	26
185	-1	0	+1	61	225
157	124	25	14	102	108
5	155	16	218	232	249

Como a figura acima demonstra, estamos deslizando o kernel da esquerda para a direita e de cima para baixo ao longo da imagem original.

Em cada coordenada (x, y) da imagem original, paramos e examinamos a vizinhança do pixel localizado no centro do kernel da imagem. Em seguida,

pegamos essa vizinhança de pixels, fazemos a convolução com o kernel e obtemos um único valor de saída. Este valor de saída é então armazenado na imagem de saída nas mesmas coordenadas (x, y) do centro do kernel.

Antes de vermos exemplos, vamos primeiro dar uma olhada na aparência de um kernel:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Acima definimos um kernel quadrado 3 x 3.

Os kernels podem ter um tamanho arbitrário de M x N pixels, desde que M e N sejam inteiros ímpares.

Obs1: A maioria dos kernels que você normalmente verá são, na verdade, matrizes quadradas N x N, o que nos permite tirar proveito de bibliotecas de álgebra linear otimizadas que operam com mais eficiência em matrizes quadradas.

Obs2: Usamos um tamanho de kernel ímpar para garantir que haja uma coordenada inteira válida (x, y) no centro da imagem:

131	162	232
104	93	139
243	26	252

131	162
104	?

À esquerda, temos uma matriz 3×3 . O centro da matriz está obviamente localizado em $x = 1, y = 1$, onde o canto superior esquerdo da matriz é usado como origem e nossas coordenadas são indexadas a zero.

Mas à direita, temos uma matriz 2×2 . O centro desta matriz estaria localizado em $x = 0,5, y = 0,5$. Porém, sem aplicar interpolação, não existe localização de pixel $(0,5, 0,5)$. Nossas coordenadas de pixel devem ser números inteiros!

Esse raciocínio é exatamente o motivo pelo qual usamos tamanhos de kernel ímpares: para sempre garantir que haja uma coordenada (x, y) válida no centro do kernel.

Entendendo convoluções de imagem

Agora que discutimos o básico dos kernels, vamos falar sobre o termo matemático chamado convolução, aplicado à imagens.

Em processamento de imagem, uma convolução requer três componentes:

1. Uma imagem de entrada.
2. Uma matriz de kernel que vamos aplicar à imagem de entrada.
3. Uma imagem de saída que armazena a imagem de entrada alterada pelo kernel.

Dessa forma, a convolução em si é bem simples. Tudo o que precisamos fazer é:

1. Selecionar uma coordenada (x, y) da imagem original.
2. Colocar o centro do kernel nesta coordenada (x, y) .
3. Pegar a multiplicação elemento a elemento da região da imagem de entrada e o kernel, então somar os valores dessas operações de multiplicação em um único valor. A soma dessas multiplicações é chamada de **saída do kernel**.

4. Usar as mesmas coordenadas (x, y) da Etapa 1, mas desta vez, armazenar a saída do kernel no mesmo local (x, y) da imagem de saída.

Abaixo, temos um exemplo de convolução (denotado matematicamente como o operador “*”) de uma região 3 x 3 de uma imagem com um kernel 3 x 3 usado para desfoque:

$$O_{i,j} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 197 & 50 & 213 \\ 3 & 181 & 203 \\ 231 & 2 & 93 \end{bmatrix} = \begin{bmatrix} 1/9 \times 197 & 1/9 \times 50 & 1/9 \times 213 \\ 1/9 \times 3 & 1/9 \times 181 & 1/9 \times 203 \\ 1/9 \times 231 & 1/9 \times 2 & 1/9 \times 93 \end{bmatrix}$$

Portanto,

$$O_{i,j} = \sum \begin{bmatrix} 21 & 5 & 23 \\ 0 & 20 & 22 \\ 25 & 0 & 10 \end{bmatrix} = 126$$

Após aplicar essa convolução, definimos o pixel localizado na coordenada (i, j) da imagem de saída O para $O_{i,j} = 126$.

E é basicamente isso. A convolução é simplesmente a soma da multiplicação elemento a elemento entre o kernel e a vizinhança que o kernel cobre da imagem de entrada.

Obs: É importante entender que o processo de “deslizar” uma matriz convolucional por uma imagem, aplicando a convolução e armazenando na saída, na prática, diminuirá as dimensões espaciais de nossa imagem de saída. Por que isso acontece?

Lembre-se de que “centralizamos” nossa computação em torno da coordenada central (x, y) da imagem de entrada sobre a qual o kernel está posicionado atualmente. Isso implica que não existem pixels “centralizados” para pixels que estão ao longo da borda da imagem. A diminuição da dimensão espacial é simplesmente um efeito colateral da aplicação de convoluções nas imagens. Às vezes esse efeito é desejável e outras vezes não, simplesmente depende da sua aplicação.

No entanto, na maioria dos casos, queremos que nossa imagem de saída tenha as mesmas dimensões que nossa imagem de entrada. Para garantir isso, aplicamos preenchimento (**padding**).

Podemos simplesmente replicar os pixels ao longo da borda da imagem, de modo que a imagem de saída corresponda às dimensões da imagem de entrada. Existem outros métodos de preenchimento, incluindo preenchimento de zero (preenchendo as bordas com zeros) e “wrap around” (onde os pixels da borda são determinados examinando a

extremidade oposta da imagem). Na maioria dos casos, você verá o preenchimento replicado ou de zero.

Alguns filtros de imagem utilizando convolução

- **Desfoque / Blur**

Utilizado para o borramento ou reduzir os detalhes de uma imagem. Quando uma imagem não possui foco, ela perde definição, fica "embaçada" ou "borrada", e os detalhes são difíceis de distinguir.

Exemplos de **kernels**:

$$\frac{1}{5} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{32} \begin{bmatrix} 1 & 3 & 1 \\ 3 & 16 & 3 \\ 1 & 3 & 1 \end{bmatrix} \quad \frac{1}{8} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Obs1: Quanto maiores as dimensões do kernel, mais a imagem ficará borrada.

Obs2: A saída da aplicação do kernel será simplesmente uma média da região de entrada.

Exemplo do filtro “blur” por convolução:



- **Nitidez / Sharpen**

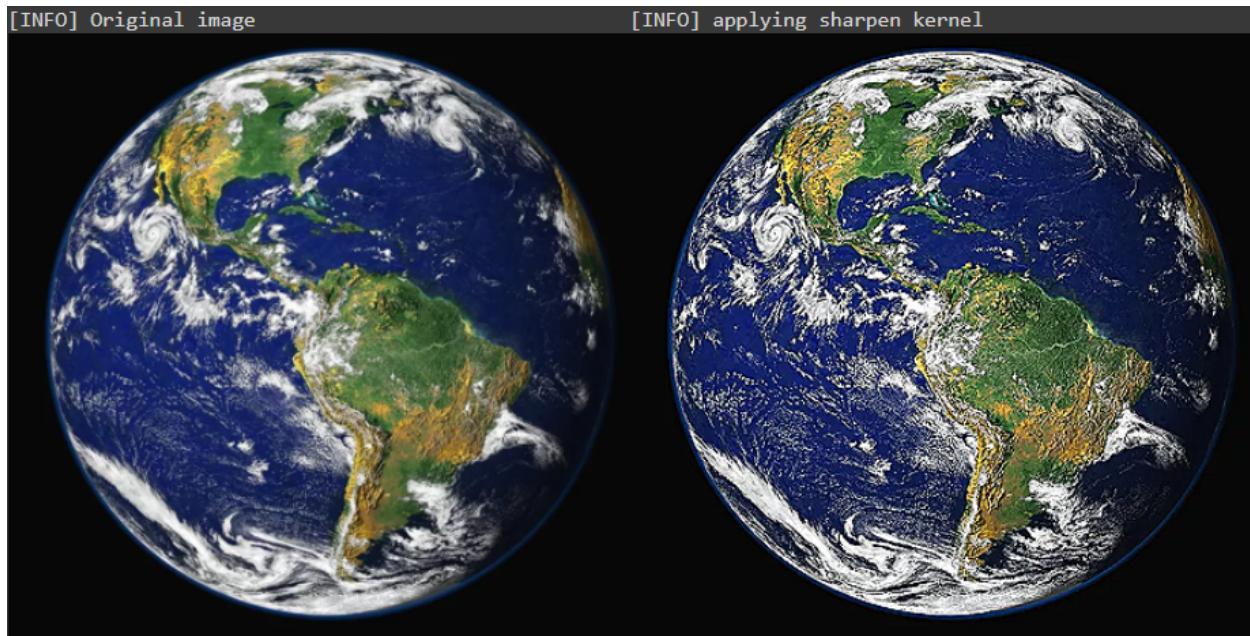
Tem como objetivo destacar as transições de intensidade na imagem, proporcionando um aumento de detalhe e nitidez sempre que há um contorno entre duas áreas distintas. Enfatiza as regiões de bordas e os ruídos e não enfatiza regiões constantes ou com variações de intensidade suaves.

Exemplos de **kernels**:

0	-1	0
-1	5	-1
0	-1	0

-1	-1	-1
-1	9	-1
-1	-1	-1

Exemplo do filtro “**sharpen**” por convolução:



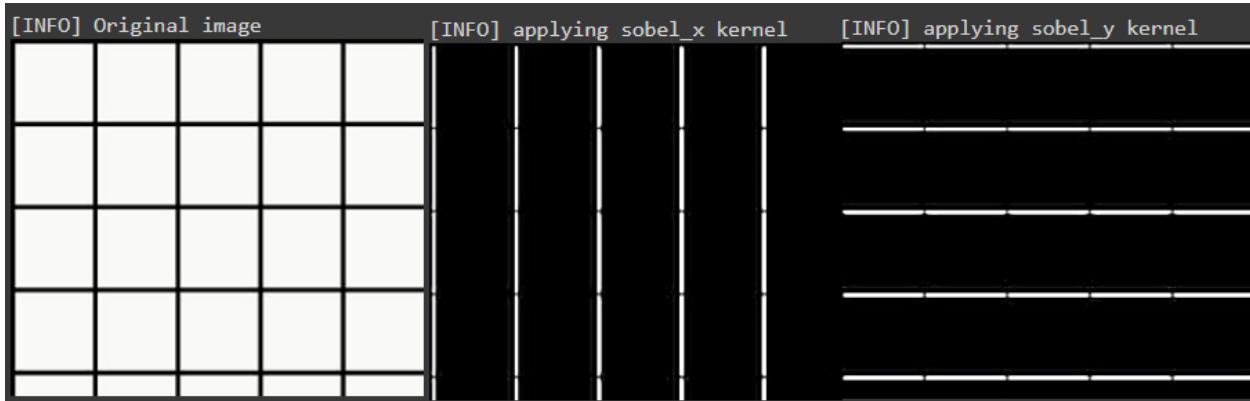
- **Sobel (bordas horizontais e verticais)**

Detecta bordas horizontais e verticais separadamente em uma imagem em escalas-de-cinza. As cores da imagem são transformadas de RGB para escalas-de-cinza. O resultado é uma imagem transparente com linhas pretas e alguns restos de cores.

Exemplos de **kernels**:

$$\left(\begin{array}{ccc} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array} \right) \quad \left(\begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{array} \right)$$

Exemplos do filtro “**sobel**” por convolução:



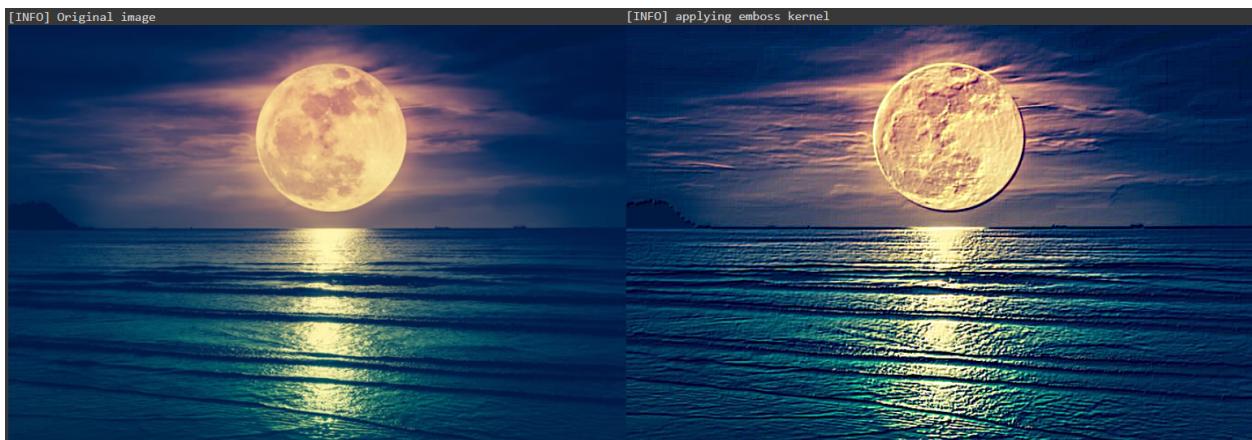
- **Relevo / Emboss**

Gera alguma luminosidade e sombras para criar um efeito que faz com que a imagem pareça gravada em relevo. Áreas mais claras são colocadas em alto relevo, e áreas escuras deixadas como se estivessem afundadas.

Exemplo de **kernel**:

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

Exemplo do filtro “**emboss**” por convolução:



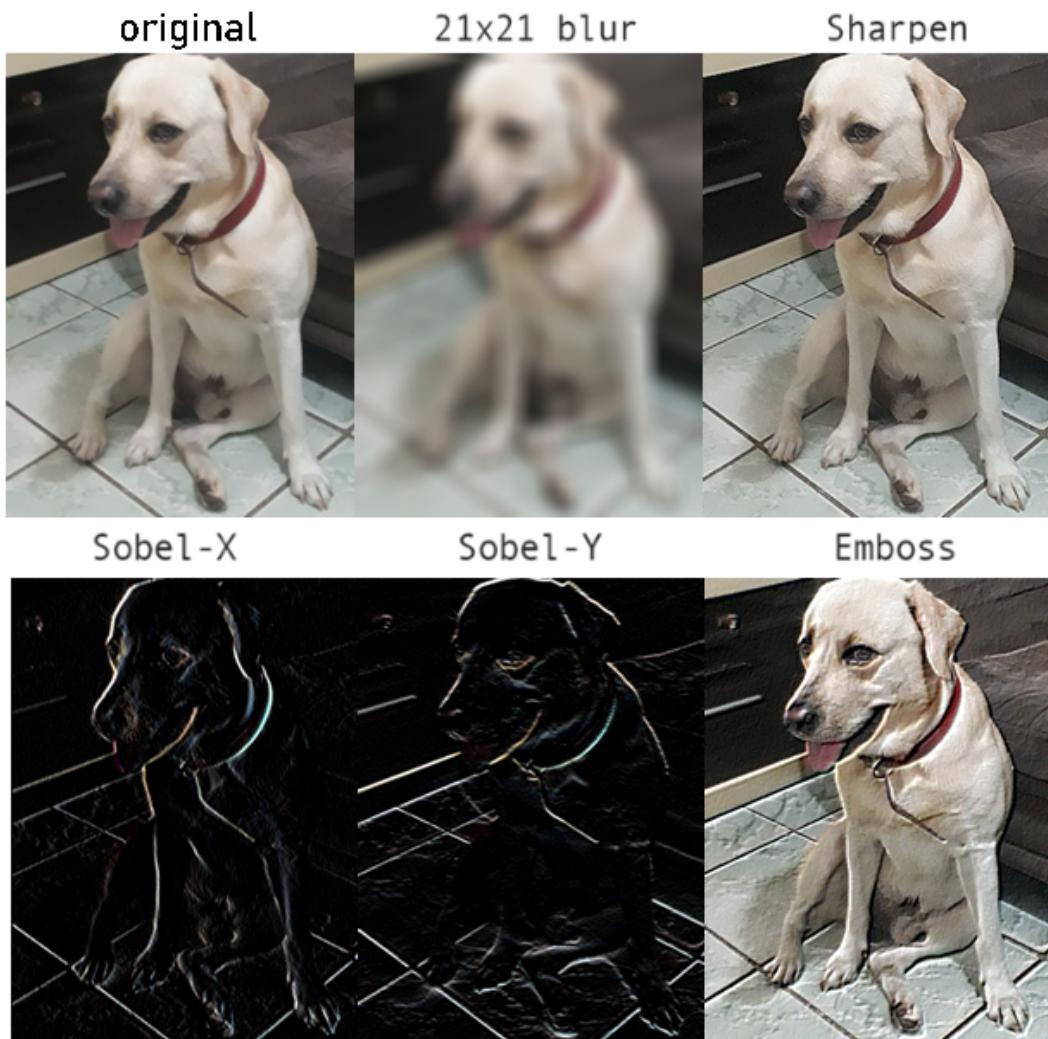
Resumo

Vimos kernels e convoluções de imagens. Se pensarmos em uma imagem como uma grande matriz, então um kernel de imagem é apenas uma pequena matriz que fica em cima da imagem.

Esse kernel então desliza da esquerda para a direita e de cima para baixo, calculando a soma das multiplicações de elementos entre a imagem de entrada e o kernel ao longo do caminho - chamamos esse valor de saída do kernel.

A saída do kernel é então armazenada em uma imagem de saída nas mesmas coordenadas (x, y) da imagem de entrada (depois de contabilizar qualquer preenchimento para garantir que a imagem de saída tenha as mesmas dimensões da entrada).

Utilizei uma função com OpenCV e Python para aplicar uma série de kernels às imagens. Esses operadores nos permitem aplicar diversos filtros, como desfocar uma imagem, torná-la mais nítida e etc.



Implementações de convolução de imagens

- [Google Colab](#) (com OpenCV e Python)
- [Site](#) (Image Kernels - explained visually)

Referências

- [Ludwig_ImageConvolution.ppt](#)
- [Entendendo de vez a convolução: base para processamento de imagens](#)
- [Convolutions with OpenCV and Python](#)
- [Convolução](#)