



UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC  
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GUSTAVO GINO SCOTTON

ALGORITMOS DE CONVOLUÇÃO - KARNEL

Araranguá  
2018

GUSTAVO GINO SCOTTON

## ALGORITMOS DE CONVOLUÇÃO - KARNEL

Trabalho realizado para a disciplina de Tópicos especiais III da Universidade Federal de Santa Catarina, no curso de Engenharia de Computação para obtenção de uma das notas referentes aos trabalhos.

Professor: Dr. Antonio Carlos Sobieranski

Araranguá

2018

## SUMÁRIO

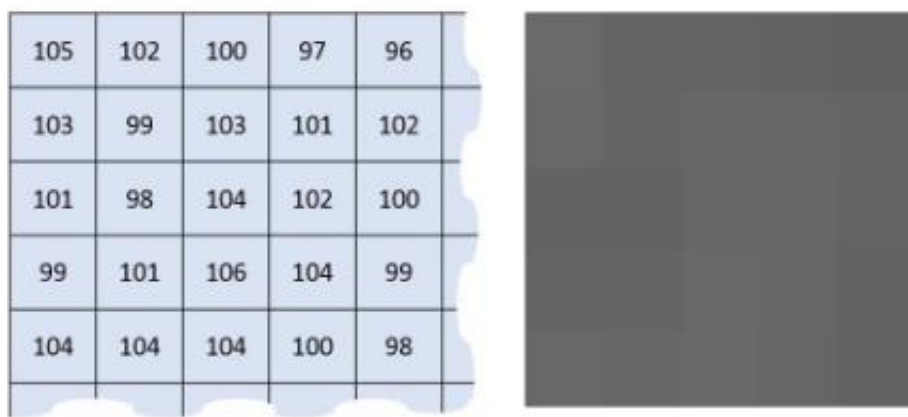
1. INTRODUÇÃO.....	4
2. CONVOLUÇÃO .....	5
3. INICIALIZANDO O PROGRAMA .....	7
4. NITIDEZ.....	11
5. DESFOQUE .....	13
6. ENTALHAMENTO .....	16
7. INSERÇÃO DE KERNEL PRÓPRIO.....	17
8. ALGORITMO DE CONVOLUÇÃO .....	18
9. CONCLUSÃO .....	19

## 1. INTRODUÇÃO

A convolução é uma das operações mais importantes no processamento de sinal e imagem. Ele poderia operar em 1D (por exemplo, processamento de fala), 2D (por exemplo, processamento de imagem) ou 3D (processamento de vídeo). Neste trabalho, utilizaremos a convolução em 2D espacial, que é usada principalmente no processamento de imagens, para extração de características.

Geralmente, podemos considerar uma imagem como uma matriz cujos elementos são números entre 0 e 255 (se for uma imagem de 8 bits). O tamanho dessa matriz é (altura da imagem) x (largura da imagem) x (número de canais de imagem). Uma imagem em escala de cinza possui 1 canal, enquanto que uma imagem colorida possui 3 canais (RGB).

Se utilizarmos uma imagem de apenas 1 canal, ou seja, em escalas de cinza de 8 bits (0 – 255), podemos dizer que cada pixel da imagem tem seu valor, para melhor demonstrar podemos ver a imagem a seguir.

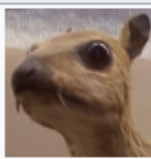



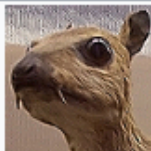
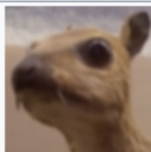
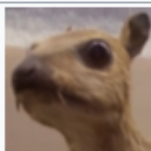
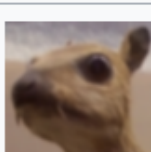
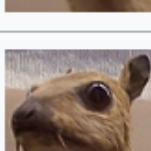


Se a imagem fosse colorida, teríamos basicamente o mesmo estilo, porém em 3 dimensões, ou seja, um cubo, RGB.

Utilizarei da biblioteca OpenCV e da linguagem Python 3.6 para o desenvolvimento deste trabalho.

## 2. CONVOLUÇÃO

Cada operação de convolução tem um kernel que pode ser qualquer matriz menor que a imagem original em altura e largura. Cada kernel é útil para uma tarefa específica, como nitidez, desfoque, detecção de bordas e muito mais. Os kernel que implementei foram retirados do wikipedia, o link pode ser acessado através de: [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

Operação	Núcleo	Resultado da imagem
<b>Identidade</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
<b>Detecção de bordas</b>	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
<b>Afiar</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Desfoque de caixa</b> (normalizado)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Desfoque gaussiano 3 x 3</b> (aproximação)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
<b>Desfoque gaussiano 5 x 5</b> (aproximação)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
<b>Máscara de nitidez 5 x 5</b> Baseado no desfoque Gaussiano com quantidade como 1 e limite como 0 (sem máscara de imagem)	$\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Segundo a própria Wikipedia, convolução é o processo de adicionar cada elemento da imagem aos seus vizinhos locais, ponderados pelo kernel. Isso está relacionado a uma forma de convolução matemática.

Por exemplo, se tivermos duas matrizes de três por três, a primeira um kernel e a segunda uma peça de imagem, convolução é o processo de inverter as linhas e colunas do kernel e, em seguida, multiplicar as entradas e somas semelhantes localmente. O elemento nas coordenadas [2, 2] (isto é, o elemento central) da imagem resultante seria uma combinação ponderada de todas as entradas da matriz da imagem, com pesos dados pelo kernel. Sua formula de implementação é:A formula pode ser notada a seguir:

$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Já no algoritmo, utilizando o OpenCV, temos nossa vida facilitada, já que existe a função **cv2.filter2D**, que basicamente, inserindo uma imagem e a matriz de kernel, realiza a interpolação e resulta na imagem esperada.

A seguir, temos as demonstrações de implementação realizadas, juntamente com seu código fonte. Vale dizer que criei um arquivo único, com um menu de opções, onde a imagem pode ser inserida pelo usuário, e sendo possível visualiza-la e salva-la diretamente no programa. A imagem utilizada para realizar os testes foi a do personagem da série Dr. House (Hugh Laurie), a imagem pode ser encontrada no google.



### 3. INICIALIZANDO O PROGRAMA

Primeiramente, devemos inserir o nome da imagem a ser editada, no nosso caso, a imagem “drhouse.jpg”.



```
main.py
C:\Users\Gustavo\Desktop\Processamento de imagem\Algoritmo de Convulção>main.py

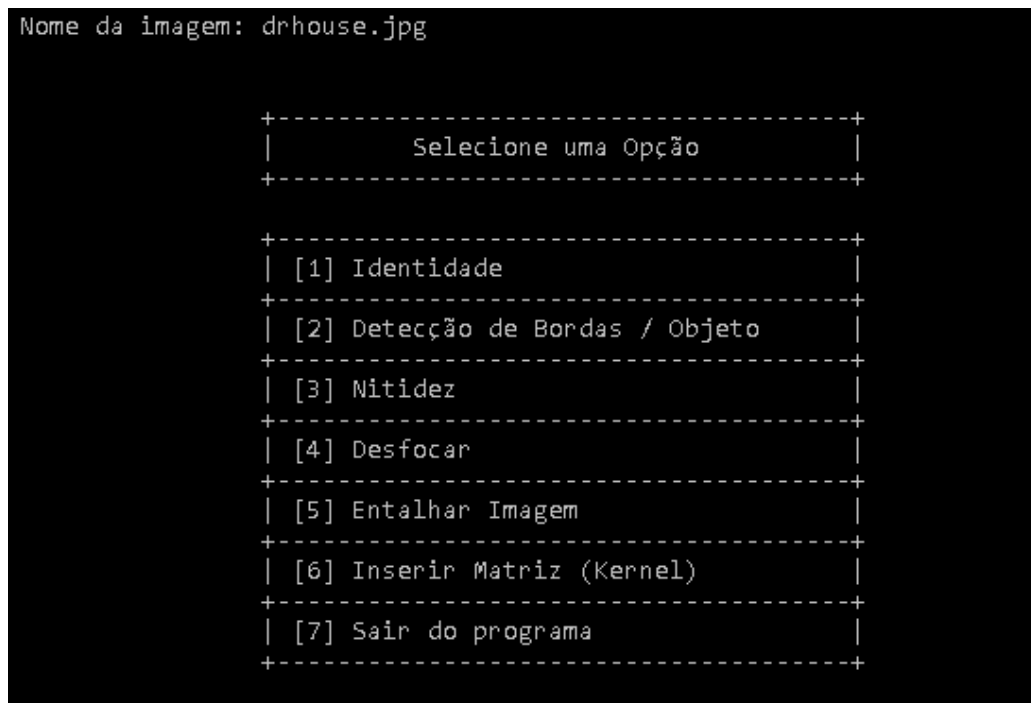
Tópicos III

ALGORITMO DE CONVULÇÃO - TRABALHO II
GUSTAVO GINO SCOTTON

Digite o nome e a extensão da imagem que deseja abrir para editar.
Exemplo: imagem.jpg

Nome da imagem: drhouse.jpg
```

Prosseguindo a diante, temos agora o menu de opções.



```
Nome da imagem: drhouse.jpg

+-----+
| Seleccione uma Opção |
+-----+

+-----+
| [1] Identidade |
+-----+
| [2] Detecção de Bordas / Objeto |
+-----+
| [3] Nitidez |
+-----+
| [4] Desfocar |
+-----+
| [5] Entalhar Imagem |
+-----+
| [6] Inserir Matriz (Kernel) |
+-----+
| [7] Sair do programa |
+-----+
```

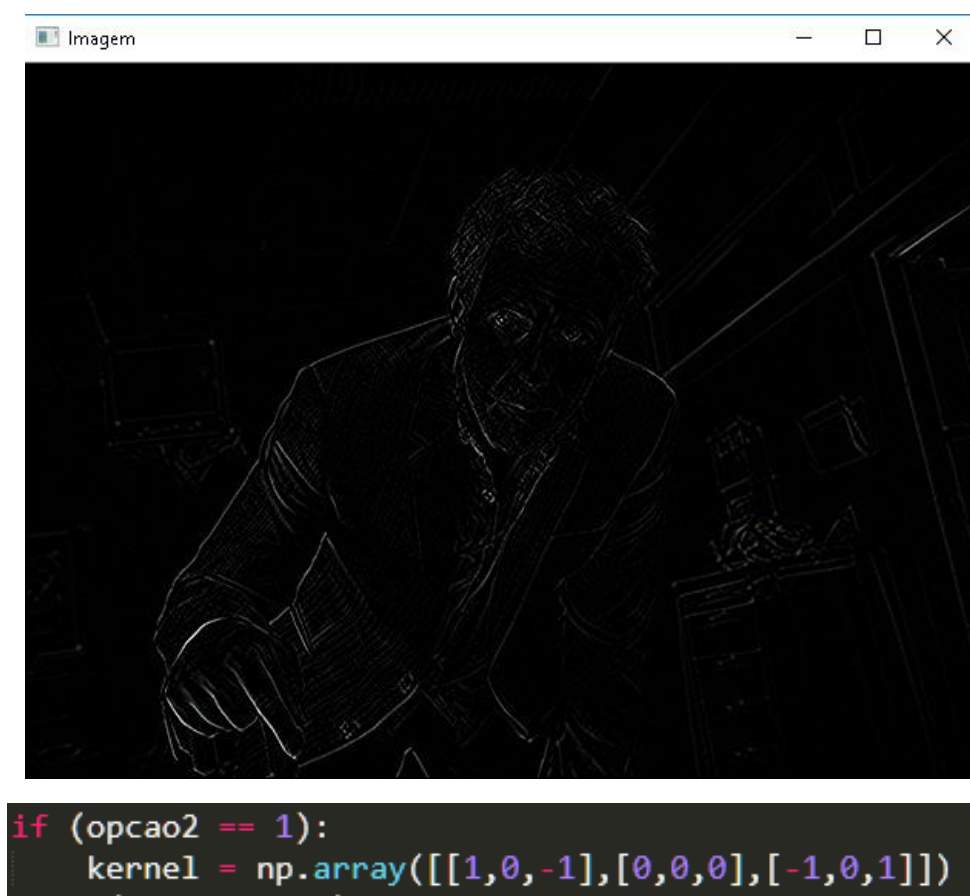
Como nossa opção 1 é a própria matriz identidade, o seu resultado será a imagem original já demonstrada anteriormente, portanto, selecionaremos a opção 2, detecção de bordas.

```
Digite a opção escolhida: 2

+-----+
| Seleccione o tipo de Detecção |
+-----+

+-----+
| [1] Detecção Baixa           |
+-----+
| [2] Detecção Média          |
+-----+
| [3] Detecção Alta           |
+-----+
| [4] Detecção Máxima         |
+-----+
```

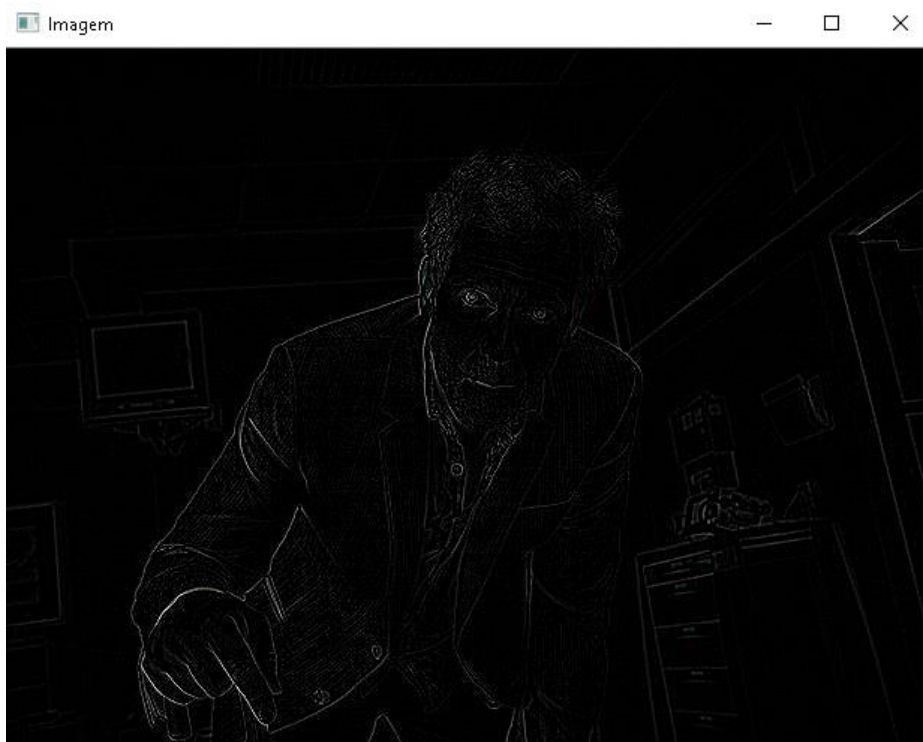
Entre os tipos de detecção, temos 3 modelos implementados, a detecção fraca, média e máxima. A seguir veremos o seu código e resultado de ambos os tipos de detecções. Começaremos por ordem numérica. Portanto, opção número 1.



O código de geração da imagem, é o mesmo para todos, mudando apenas o Kernel, por este motivo, demonstrarei apenas o kernel nesta etapa, e o algoritmo de geração no final.



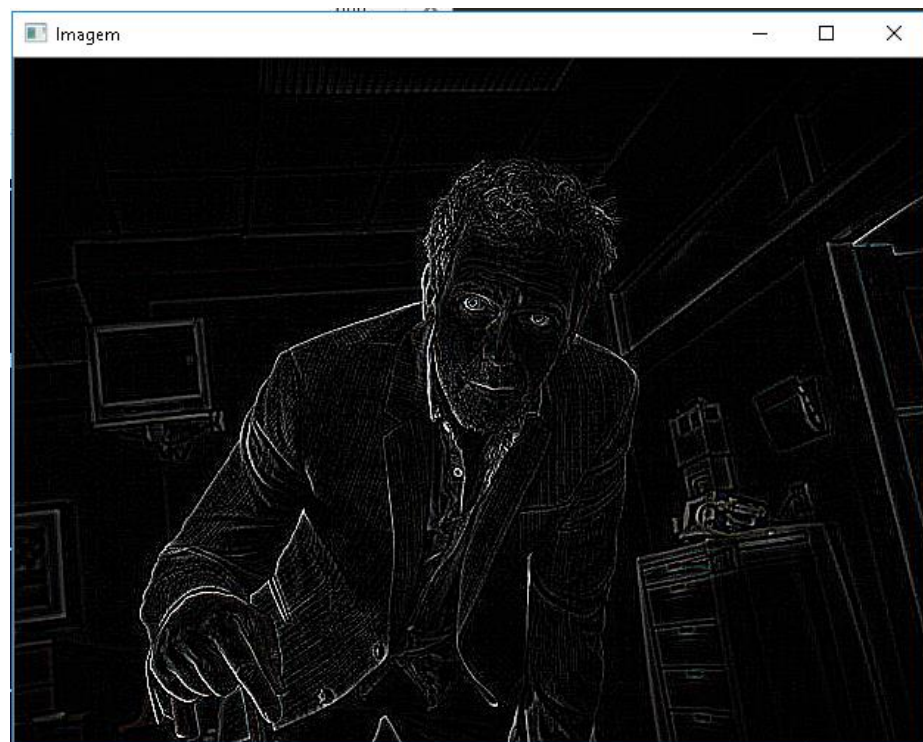
Seguindo a diante, selecionamos a opção 2 e como resultado temos:



Com seu código fonte de kernel sendo:

```
elif (opcao2 == 2):  
    kernel = np.array([[0,1,-0],[1,-4,1],[0,1,0]])
```

Temos a opção 3 a ser selecionada, resultando em:



```
elif (opcao2 == 3):  
    kernel = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])  
    else:
```

Por fim, temos a opção 4, com a maior detecção possível, o resultado obtido pode ser observado a seguir.



Sendo seu código fonte de geração:

```
elif (opcao2 == 4):  
    kernel = np.array([[-1.5, -1.5, -1.5], [-1.5, 12, -1.5], [-1.5, -1.5, -1.5]])
```

Conforme é possível observar no código, ele aumenta em 50% os valores de detecção do kernel da opção 3.

## 4. NITIDEZ

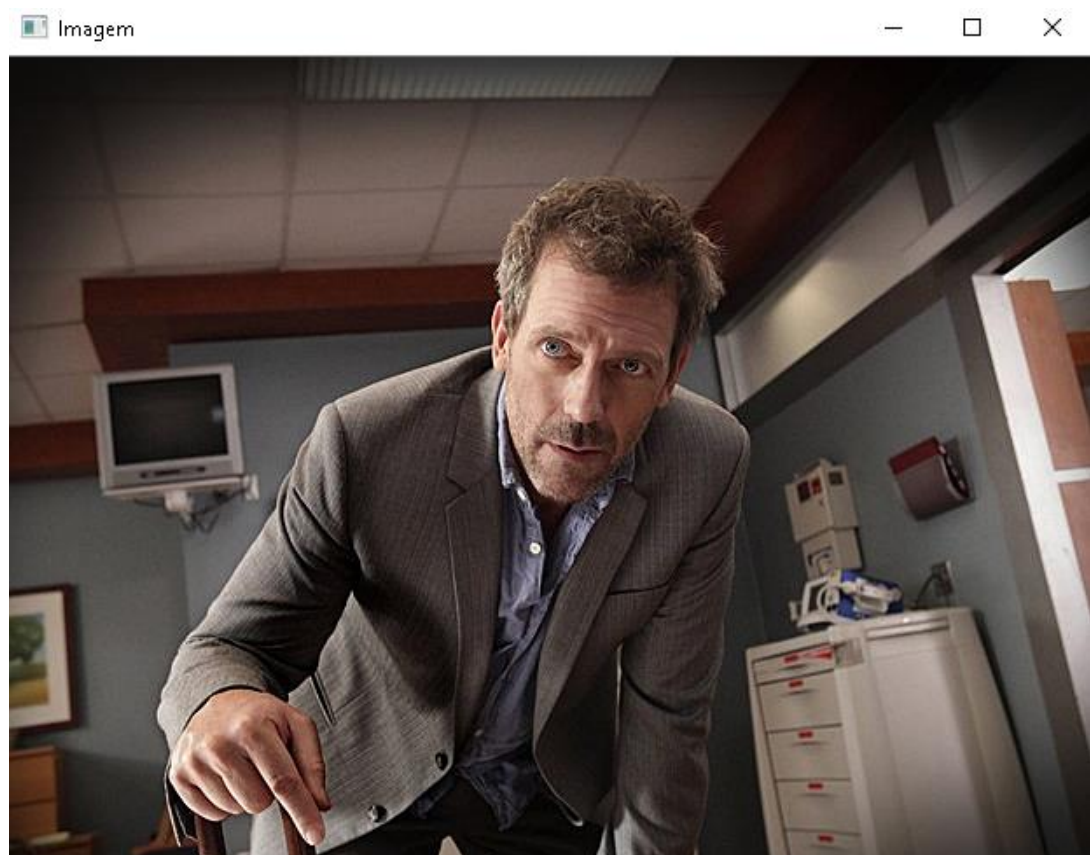
Agora iremos trabalhar com a nitidez da imagem, acessando o menu temos as seguintes opções:

```
Digite a opção escolhida: 3

+-----+
| Seleccione o tipo de Nitidez |
+-----+

+-----+
| [1] Nitidez razoavel        |
+-----+
| [2] Nitidez Máxima          |
+-----+
```

Seguindo novamente em ordem numérica, temos a opção 1 resultando:

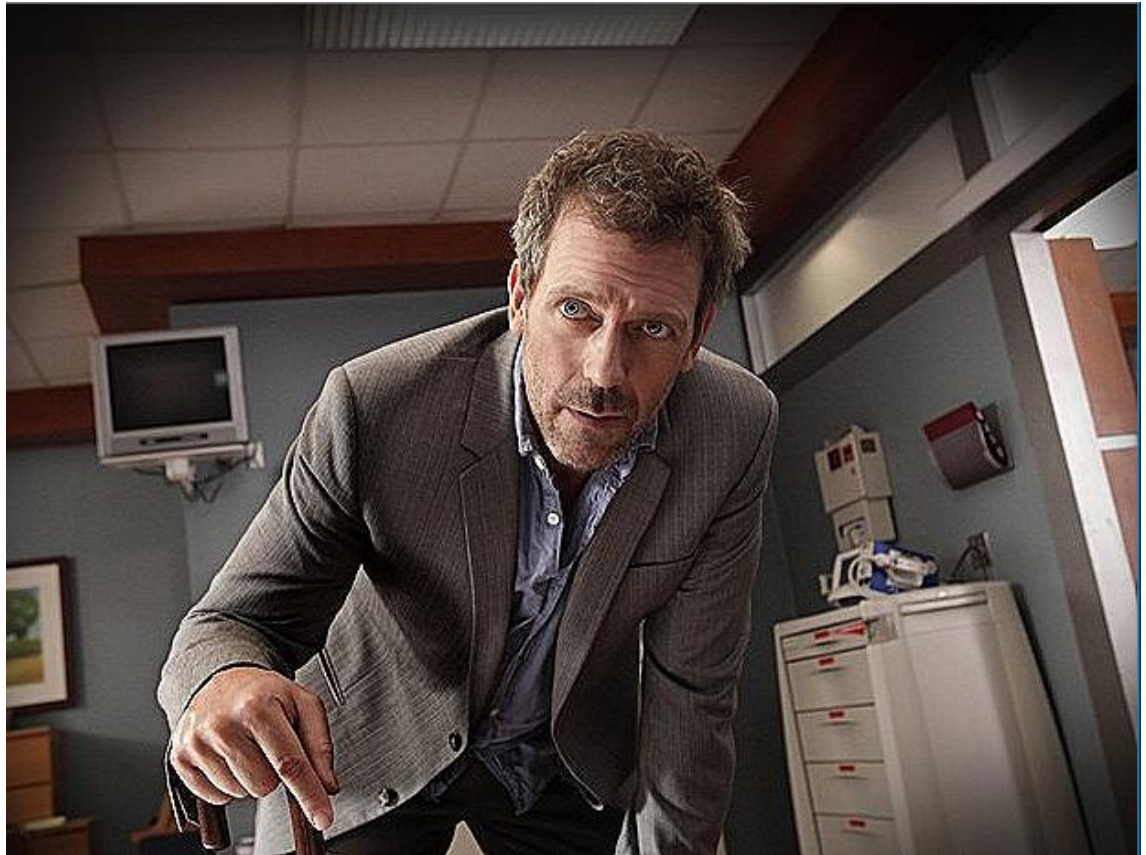


Com seu kernel sendo:

```
if (opcao2 == 1):
    kernel = np.array([[1,4,6,4,1], [4,16,24,16,4], [6,24,-476,24,6], [4,16,24,16,4], [1,4,6,4,1]])*(-1/256)
```



A diferença de nitidez é difícil de se notar, é preciso se abster aos detalhes. Seguindo, temos agora o resultado da opção 2, nitidez máxima.



Sua matriz de kernel sendo:

```
elif (opcao2 == 2):  
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
```

Utilizando a nitidez máxima, é possível ver com mais clareza este efeito na imagem. Com isso terminamos nossa demonstração de nitidez e podemos voltar ao menu principal, e usarmos o kernel de desfoque.

## 5. DESFOQUE

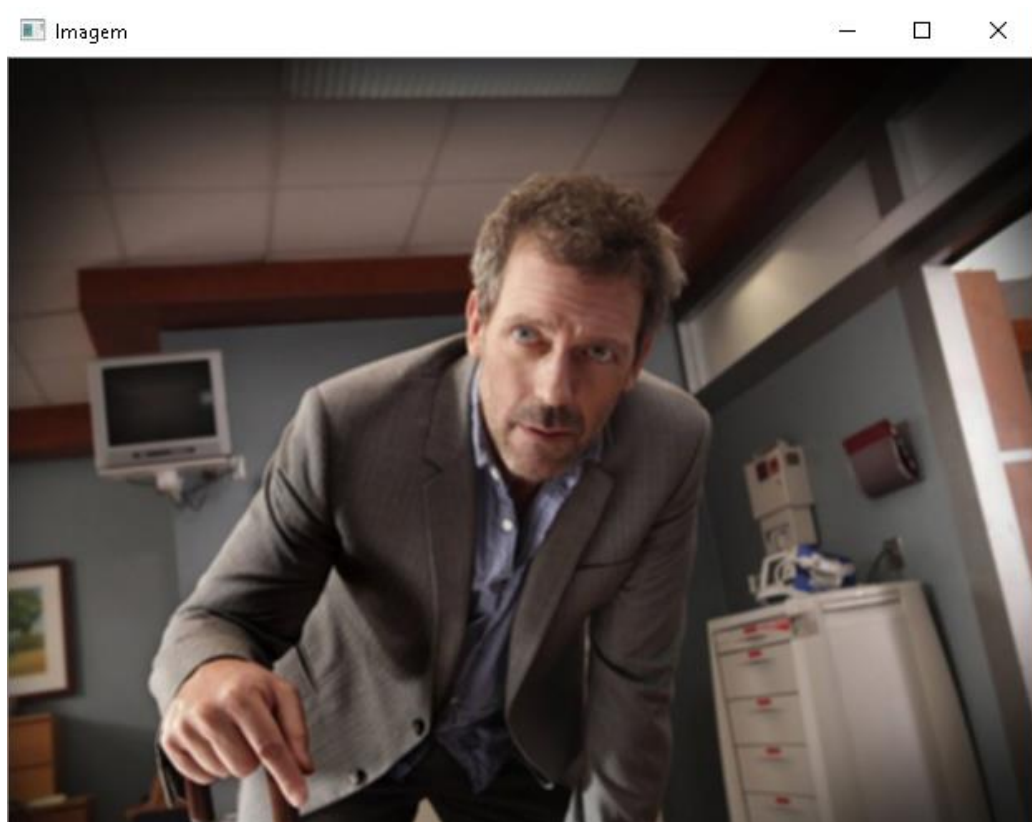
Realizando a seleção da opção 4, temos um novo submenu:

```
Digite a opção escolhida: 4

+-----+
| Selecione o tipo de Desfoque |
+-----+

+-----+
| [1] Desfoque Baixo           |
+-----+
| [2] Desfoque Médio           |
+-----+
| [3] Desfoque Alto            |
+-----+
```

Novamente, seguindo o padrão, utilizaremos da ordem numérica para demonstrar os resultados obtidos nesta etapa.



```
if (opcao2 == 1):
    kernel = (np.array([[1,1,1],[1,1,1],[1,1,1]])/9)
```

Agora, selecionando a opção 2, ou seja, desfoque médio.



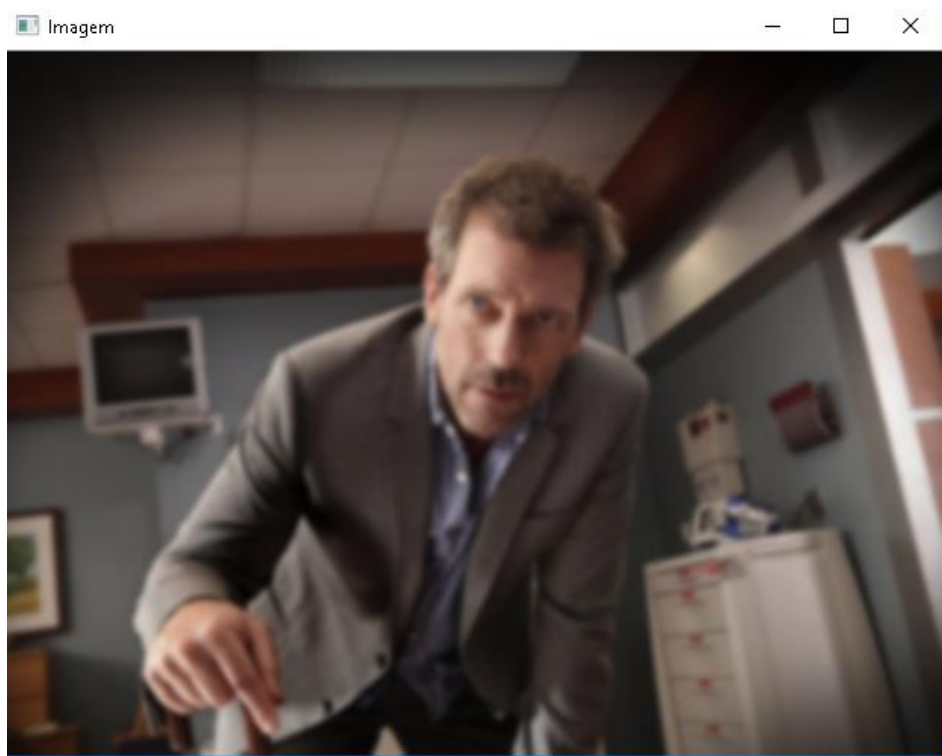
```
elif (opcao2 == 2):  
    kernel = (np.array([[1,2,1],[2,4,2],[1,2,1]])/16)  
    if (opcao3 == 3):
```

Podemos notar pouca diferença entre o pouco e médio, isso se dá pela semelhança na matriz de kernel. Na opção 3 será possível visualizar de forma mais notável este desfoque.

Vale lembrar que o programa permite realizar o desfoque várias vezes (não somente o desfoque como todos os outros efeitos), aumentando ainda mais o seu grau de eficiência, realizaremos este teste a seguir com a imagem de desfoque máximo.



Aqui podemos notar um desfoque maior, embora não seja o extremo, porém se utilizarmos o desfoque seguidamente (especificamente 3x), um sobre o outro, podemos resultar na seguinte imagem:



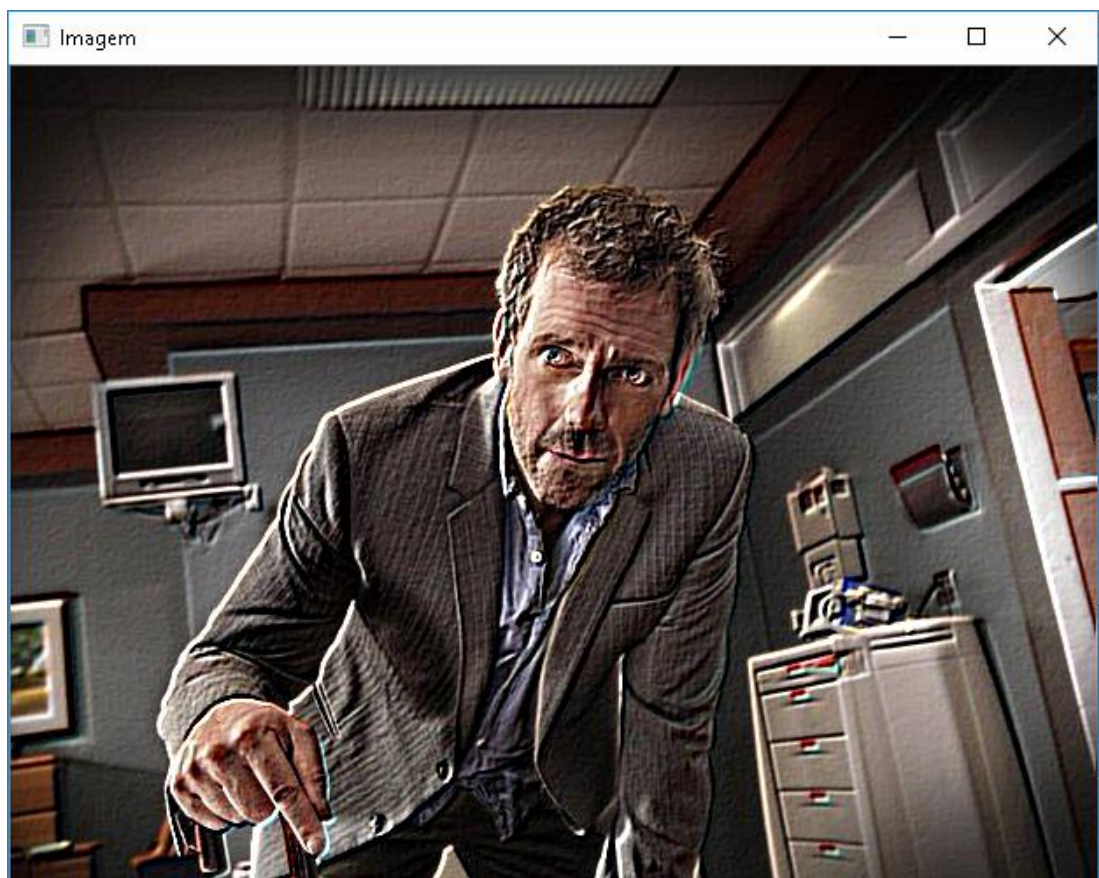
```
elif (opcao2 == 3):  
    kernel = (np.array([[1,4,6,4,1], [4,16,24,16,4], [6,24,36,24,6], [4,16,24,16,4], [1,4,6,4,1]])/256)
```



## 6. ENTALHAMENTO

Esse filtro estampa e talha a camada ou seleção ativas, dando à imagem partes altas e partes baixas. Áreas mais claras são colocadas em alto relevo, e áreas escuras deixadas como se estivessem afundadas.

Acessando a opção 6 do menu principal, temos o seguinte resultado de imagem:



Tendo sua matriz de kernel em forma de algoritmo sendo:

```
elif (opcao == 5):  
    kernel = np.array([[ -2, -1, 0], [-1, 1, 1], [0, 1, 2]])
```



## 7. INSERÇÃO DE KERNEL PRÓPRIO

Também se implementou a opção de inserir sua própria matriz de convolução 3X3, que pode ser observada a seguir o algoritmo e exibição no prompt de comandos.

```
Digite a opção escolhida: 6

Matriz de convolução 3X3 :

Digite o valor da posição [1,1] : 0
Digite o valor da posição [1,2] : 0
Digite o valor da posição [1,3] : 0
Digite o valor da posição [2,1] : 0
Digite o valor da posição [2,2] : 1
Digite o valor da posição [2,3] : 0
Digite o valor da posição [3,1] : 0
Digite o valor da posição [3,2] : 0
Digite o valor da posição [3,3] : 0
Digite o valor da divisão da Matriz : 1

Deseja salvar a imagem? [Y] [N] : n

Deseja visualizar a imagem agora? [Y] [N] : y

*****
*   Digite qualquer tecla para fechar.   *
*****
```

Código fonte:

```
elif (opcao ==6):

    print("")
    print("    Matriz de convolução 3X3 : ")
    print("")
    v_11=float(input(" Digite o valor da posição [1,1] : "))
    v_12=float(input(" Digite o valor da posição [1,2] : "))
    v_13=float(input(" Digite o valor da posição [1,3] : "))
    v_21=float(input(" Digite o valor da posição [2,1] : "))
    v_22=float(input(" Digite o valor da posição [2,2] : "))
    v_23=float(input(" Digite o valor da posição [2,3] : "))
    v_31=float(input(" Digite o valor da posição [3,1] : "))
    v_32=float(input(" Digite o valor da posição [3,2] : "))
    v_33=float(input(" Digite o valor da posição [3,3] : "))
    v_div=float(input(" Digite o valor da divisão da Matriz : "))

    kernel = (np.array([[v_11,v_12,v_13],[v_21,v_22,v_23],[v_31,v_32,v_33]])/v_div)
```

## 8. ALGORITMO DE CONVOLUÇÃO

Agora, explicaremos o algoritmo utilizado. Como mencionado na introdução, utilizou-se do OpenCV, tornando mais fácil o desenvolvimento dos filtros.

Por exemplo:

```
opcao2 = int(input("Digite a opção escolhida: "))
if (opcao2 == 1):
    kernel = (np.array([[1,1,1],[1,1,1],[1,1,1]])/9)
elif (opcao2 == 2):
    kernel = (np.array([[1,2,1],[2,4,2],[1,2,1]])/16)
elif (opcao2 == 3):
    kernel = (np.array([[1,4,6,4,1], [4,16,24,16,4], [6,24,36,24,6], [4,16,24,16,4], [1,4,6,4,1]])/256)
else:
    print("    A opção digitada não existe, por padrão escolhemos a primeira opção. (Detecção fraca)")
    kernel = np.array([[1,0,-1],[0,0,0],[-1,0,1]])

elif (opcao == 5):
    print("")
    print("    Finalizando o programa...")
    sys.exit()
    cv2.destroyAllWindows()
    sys.exit()

else:
    print("")
    print("-----")
    print("    - Valor digitado não corresponde as opções, digite uma opção válida! -")
    print("-----")
    print("")

img_kernel = cv2.filter2D(img,-1,kernel) # Aplica o filtro de kernel na imagem
resposta=input("    Deseja salvar a imagem? [Y] [N] : ")
if (resposta == "Y" or resposta=="y"):
    print("    Como você gostaria de chamar esta imagem?")
    print("    Exemplo: Editada.jpg")
    print("")
    write = input("Nome da imagem: ")
    cv2.imwrite(write,img_kernel)
    print("    Imagem salva com sucesso!")
    print("")
resposta=input("    Deseja visualizar a imagem agora? [Y] [N] : ")
if (resposta == "Y" or resposta=="y"):
    print("    Digite qualquer tecla para fechar.")
    cv2.imshow("Imagem",img_kernel)
    cv2.waitKey()
    cv2.destroyAllWindows()
    img = img_kernel # Atualiza a img a ser editada, para poder realizar edição sobre edição.
```

Esta é a parte final do código, onde temos as matrizes de kernel sendo selecionadas conforme a opção escolhida pelo usuário, e na linha:

```
145     img_kernel = cv2.filter2D(img,-1,kernel) # Aplica o filtro de kernel na imagem
```

O algoritmo de convolução é aplicado, conforme explicado no tópico 2 deste trabalho. Portanto, o OpenCV realiza esta tarefa árdua para nós.

## **9. CONCLUSÃO**

Com isso, podemos concluir que o trabalho foi de grande aprendizado nas áreas envolvendo filtros de imagem. Foi gratificante conseguir construir todo o algoritmo e vê-lo funcionar conforme o esperado. Busquei realizar a tarefa da maneira mais utilizável possível para futuros usuários, já que compartilharei este código no meu GitHub. Por fim, fico ansioso para futuros trabalhos, que consequentemente me trarão muitos aprendizados.