Trabalho 01 de Introdução ao Processamento de Imagem Digital

Fabio Fogliarini Brolesi (brolesi@gmail.com) - RA 023718

13 de abril de 2021

Sumário

1 Imagens coloridas

1.1 Primeira etapa

A proposta para a primeira etapa do trabalho com imagens coloridas é a de executar um cálculo para cada um dos canais, a saber, vermelho, verde e azul.

A partir daí, cada um deles teria uma nova composição, baseada numa função de cada um dos três canais originais.

Usamos como base a figura 1 para comparativo dos resultados.

Uma vez a imagem carregada, então a variável que possui os dados é um `array` do tipo `uint8` de dimensões $(y, x, 3)$, onde y e x são as dimensões de **altura** e **largura** da imagem respectivamente (em pixels), e o valor 3 refere-se aos canais de cor (vermelho, verde e azul, respectivamente).

Criamos uma matriz $M_{3\times3}$ com os valores da transformação, conforme segue:

Figura 1: Imagem base para comparação

```
M = np.array([[0.393, 0.769, 0.189],
              [0.349, 0.686, 0.168],
              [0.272, 0.534, 0.131]])
```

Identificamos a partir da necessidade do problema o uso do produto interno, matematicamente definido por:

$$A \cdot B = \sum_{i=0}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

A partir daí, realizamos o produto interno da variável que continha a imagem original

(alocada na variável `imagem` no trecho abaixo), com a matriz M^T na linguagem python, desta forma:

```
imagem_linha = np.dot(imagem, M.T).
    astype(np.uint8)
```

Tornando a variável `imagem_linha` a imagem transformada a partir dos valores da matriz M.

A utilização do método `np.dot` vem a partir da necessidade de multiplicação e soma de elementos e da análise da documentação da função [1] que explicita que:

> Se a é uma matriz $N - D$ e b é uma matriz $M - D$ (onde $M >= 2$), `np.dot` é um produto da soma sobre o último eixo de a e o penúltimo eixo de b[1]:

```
dot(a, b)[i,j,k,m] = sum(a[i,j
    ,:] * b[k,:,m])
```

O resultado da multiplicação foi definido como o tipo `np.uint8`. Propusemos a limitação para o valor máximo 255 logo após o resultado do produto interno, da seguinte forma:

```
imagem_linha[imagem_linha > 255] = 255
```

O resultado para a figura base 1 é exibido na figura 2:

1.2 Segunda etapa

Para a segunda etapa da atividade com imagens coloridas, a partir de uma função de `R`, `G`, `B` (vermelho, verde e azul respectivamente), geramos uma imagem monocromatica conforme a função que segue:

[1] traduzido pelo autor

Figura 2: Imagem com os canais R, G, B alterados

$$I = 0,2989 \times R + 0,5870 \times G + 0,1140 \times B$$

Para isso, no python criamos a estrutura de cálculo a seguir:

```
M = np.array([0.2989, 0.5870, 0.1140])
imagem_linha = np.dot(imagem, M.T).
    astype(np.uint8)
```

Onde `imagem` é a variável que contém a matriz de imagem com dimensões $(y, x, 3)$, onde y e x são as dimensões de **altura** e **largura** da imagem respectivamente (em pixels), e o valor 3 refere-se aos canais de cor (vermelho, verde e azul, respectivamente).

O cálculo é um produto escalar que considera cada um dos valores da variável `M` multiplicado por cada um dos canais.

O resultado final é uma matriz de dimensões (y, x) onde y e x são as dimensões de **altura** e **largura** respectivamente. Como só há um canal, então não há terceira dimensão na matriz resultante.

2 Imagens monocromáticas

Na manipulação das imagens monocromáticas, aplicamos convoluções baseadas em filtros definidos entre h_1 e h_9, e um filtro $h_{10} = \sqrt{(h_1)^2 + (h_2)^2}$.

A aplicação do filtro nada mais é do que "deslizar" uma matriz (também chamada kernel) sobre cada um dos pixels da imagem original, afim de gerar novos valores e com isso mostrar certas características da imagem, ou mesmo algum tipo de transformação da mesma.

2.1 Filtros

Para entender o que cada um dos 10 filtro faz h_1 a h_9 e $h_{10} = \sqrt{(h_1)^2 + (h_2)^2}$, criamos uma imagem monocromática com apenas 2 cores (preto e branco) e aplicamos os filtros a partir daí. A imagem tem 42 pixels de altura e 82 de largura. A figura original para análise é a 3 e os resultados estão na figura 4.

Conseguimos observar a partir da imagem de referência 3 as seguintes características dos filtros:

- o filtro h_1 (na imagem 4a) tem a características de detecção bordas (em particular é do tipo Sobel) para trazer nitidez à imagem. A ideia por trás de usar um valor de peso de 2 na parte central é alcançar alguma suavização, dando mais importância ao ponto central. Os coeficientes em todos os kernels que somam zero, dariam uma resposta de zero em áreas de intensidade constante, conforme esperado de um operador baseado em derivada. Este filtro marca características expressas na vertical e neste caso, à direita.

- o filtro h_2 (na imagem 4b) é igual a h_1^T. Ele tem a características de detecção bordas (novamente, em particular do tipo Sobel). Por ser o transposto do filtro h_1, olha para a característica das bordas da imagem na horizontal e, em particular, na parte de baixo.

- o filtro h_3 (na imagem 4c) é um kernel Laplaciano, um filtro passa-baixa. Na imagem de exemplo ele detectou as bordas, ele dá destaque ao pixel central, e um peso bem menor aos pixels da borda. Ele serve neste caso para detecção de contornos (outlines). A proposta dele é suavizar os valores de bordas e realçar o valor central. Para regiões homogêneas também colocará o valor do pixel central para 0.

- o filtro h_4 (na imagem 4d) é um filtro que trata das médias dos valores do entorno do pixel e do próprio pixel. Ele fará com que o valor do pixel seja a média dos pixels da sua volta. Considerado um filtro passa-baixa pois atenua eventuais ruídos (para cima ou para baixo). Ele causa o borramento da imagem, como é possível perceber nas bordas.

- o filtro h_5 (na imagem 4e) é um filtro cuja diagonal secundária tem um peso grande, enquanto que os outros elementos são homogêneos com valor -1. Ele vai fazer com que bordas diagonais à 45 graus (considerando a diagonal secundária) sejam mais marcadas. Numa área homogênea, o resultado será uma cor mais escura na medida em que a soma é 0. Isso indica que a luminosidade e o contraste podem ser afetados.

- o filtro h_6 (na imagem 4f) é um equivalente transposto ao filtro h_5. Ele considera a diagonal principal como importante, desta forma a detecção de bordas na diagonal principal será mais efetiva, suavizando bordas em outras direções.

- o filtro h_7 (na imagem 4g) também é de detecção de bordas, os valores que são con-

siderados são os da diagonal secundária, com pesos opostos. a média do filtro é 0, de modo que há aqui uma suavização em regra geral, e como o valor da maior parte dos elementos é 0, então tende a trazer para valores baixos boa parte dos pixels das imagens, enquanto que as bordas são enfatizadas. Na imagem referente ao resultado, observamos que as bordas das regões onde a imagem original é branca se preservaram à direita e acima.

- o filtro h_8 (na imagem 4h) possui a mais alta intensidade no pixel central, sendo que os pixels à volta tem pesos negativos. Observa-se que a aplicação é em torno do raio, excluindo-se os cantos inferiores e superiores à direita e à esquerda. Para a imagem padrão, verificou-se que ela destaca bordas, mas reduz o contraste da imagem. Dentro das áreas brancas da imagem padrão ele foi capaz de manter a borda, mas pelo fato de a área ser homogênea, então restante do interior destes quadrados fica com o valor 0.

- o filtro h_9 (na imagem 4i) também conhecido como filtro gaussiano. Ele suaviza e atenua as diferenças entre os pixels. Nota-se que na imagem ele realizou um desfoque (borramento), e diminuiu o contraste entre os pixels e minimizaação do brilho.

- o filtro h_{10} (na imagem 4j) marca mais as bordas do filtro, não dando relevância para o pixel central. O resultado dele é uma imagem com resultado final de pixels com alta intensidade de brilho, apenas com grupos de pixels mais escuros aglutinados sendo relevantes para mudança de intensidade para a imagem resultante.

$$h_{10} = \begin{array}{|c|c|c|} \hline \sqrt{2} & 2 & \sqrt{2} \\ \hline 2 & 0 & 2 \\ \hline \sqrt{2} & 2 & \sqrt{2} \\ \hline \end{array}$$

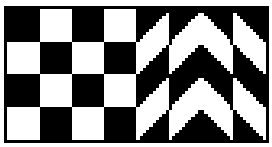
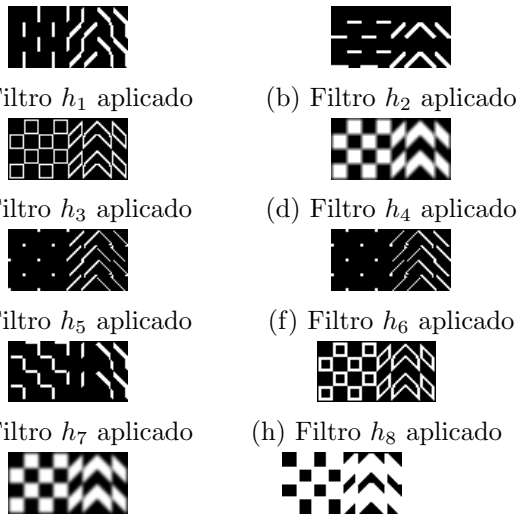


Figura 3: Imagem padrão



(a) Filtro h_1 aplicado (b) Filtro h_2 aplicado

(c) Filtro h_3 aplicado (d) Filtro h_4 aplicado

(e) Filtro h_5 aplicado (f) Filtro h_6 aplicado

(g) Filtro h_7 aplicado (h) Filtro h_8 aplicado

(i) Filtro h_9 aplicado (j) Filtro h_{10} aplicado

Figura 4: Resultados de aplicações de filtros sobre a figura padrão 3

2.2 Aplicação dos filtros

Para o processamento, utilizamos o recurso de criar uma moldura para a imagem antes de executar a convolução, afim de, após o processamento com o kernel, as imagens transformadas tivessem a mesma dimensão da imagem original. Para tanto, utilizamos um método da biblioteca `cv2` para o Python, o `copyMakeBorder`[5]. Esta borda copia os pixels do entorno para ser criada.

```
image = cv2.copyMakeBorder(image,
    padding, padding, padding, padding,
     cv2.BORDER_REPLICATE)
```

Após a criação da borda na imagem original, então a convolução é realizada, utilizando o processamento a seguir:

```
imagem_transformada = np.asarray([(
    image[y:y + kernel_height, x:x +
    kernel_height] * kernel).sum() for
    x, y in np.ndindex(image_width,
    image_height)]).reshape((
    image_width, image_height)).T.
    astype(np.float32())
```

A partir dos resultados, realizamos uma normalização dos dados utilizando o método `rescale_intensity` da biblioteca `scikit-learn` no módulo `exposure` conforme [4].

Uma vez a convolução aplicada, comparamos o resultado realizado manualmente com o método `cv2.filter2D` para avaliar potenciais discrepâncias. A comparação é da forma abaixo:

```
(imagem_transformada == cv2.filter2D(
    img, -1, kernel, borderType=cv2.
    BORDER_REPLICATE)).all()
```

Para a avaliação dos filtros, colocamos o resultado feito manualmente (utilizando vetorização e broadcasting) e comparamos os resultados deles com o método `cv2.filter2D`:

Apenas as imagens com filtros h_4, h_9 e h_{10} tiveram resultado `False` considerando as condições acima. Ainda assim, visualmente os resultados destes filtros não tiveram diferença expressiva.

Para a comparação de porcentagem de diferenças entre os filtros executados manualmente versus os métodos *built-in* da biblioteca `CV2`, utilizamos a imagem https://www.ic.unicamp.br/~helio/imagens_png/baboon.png.

O filtro h_4 tem $44,44\%$ de diferenças entre o executado manualmente e o método `cv2.filter2D`. O filtro, h_9, $45,05\%$ de diferença nos pixels e o filtro h_{10}, apenas $7,63 \times 10^{-3}\%$, considerando-se a imagem original.

Outro ponto a ser considerado é que o valor máximo da diferença entre as matrizes resultantes dos filtros

```
imagem_com_filtro_cv2 -
    imagem_com_filtro_manual
```

foi de 1, novamente mostrando que visualmente a diferença é mínima.

Colocamos nas figuras 5, 6 e 7 as imagens na seguinte ordem: imagem gerada a partir do filtro aplicado manualmente, imagem gerada a partir do filtro built-in da biblioteca `openCV`, diferença conforme o código

```
(imagem_com_filtro_cv2 -
    imagem_com_filtro_manual) * 255
```

para exacerbarmos as diferenças (uma vez que a diferença máxima entre todos os filtros era de 1).

3 Limitações

Não foi feito um benchmark para avaliar as vantagens em termos de recursos computacio-

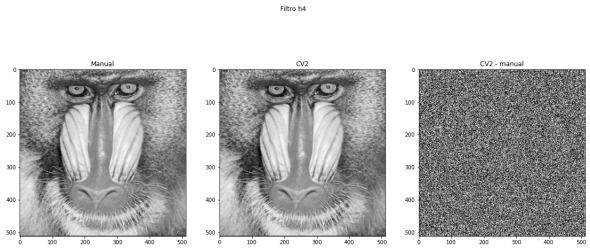


Figura 5: Imagens comparativas do filtro h_4

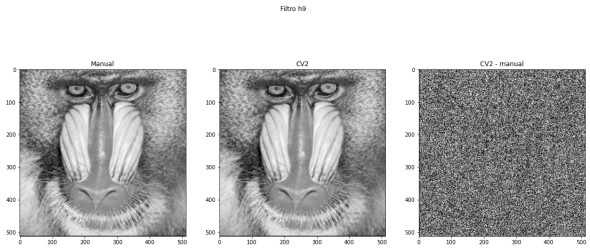


Figura 6: Imagens comparativas do filtro h_9

nais das técnicas de vetorização e *broadcasting* versus uso de laços de repetição e condicionais de uma implementação tradicional.

O programan não trata de imagens com quantidade de canais diferentes de 1 (imagens monocromáticas) ou 3 (para imagens coloridas). Imagens com camada de de transparência (alpha, equivalente à 4a camada numa imagem colorida, por exemplo), não são tratadas neste programa.

Foram realizados testes para imagens quadradas ($largura = altura$), e imagens retangulares ($largura > altura$ e $largura < altura$) para ambos os exercícios (images coloridas e monocromáticas).

Afim de garantir que as dimensões das imagens fossem preservadas no output, utilizamos a bilbioteca `imageio` para realizar a leitura e gravação das imagens, mas mantivemos a estrutura que utilizaríamos com o matplotlib, forçando a máscara de pixels para que estes, ao final das manipulações ficassem entre 0 e 255.

Não foi analisado se a diferença dos filtros

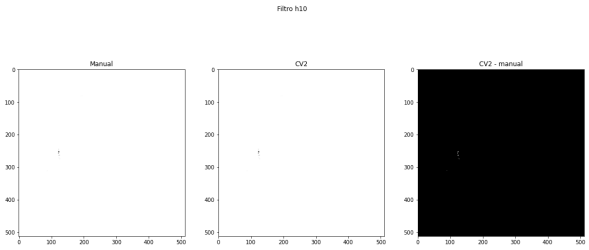


Figura 7: Imagens comparativas do filtro h_{10}

h_4, h_9 e h_{10} tem a ver com a diferença de tipos `np.uint8`, `np.uint16` e outros.

Sobre a decisão de executar a criação da borda nas imagens monocromáticas baseadas na cópia dos pixels ao redor, ela foi uma decisão dentre várias possíveis, incluindo a de não criar uma borda, fazendo com que a imagem resultante fosse menor que a original. Vale salientar que a borda não é considerada confiável para a análise.

Referências

[1] numpy.dot, https://numpy.org/doc/stable/reference/generated/numpy.dot.html Acessado em: 2021-03-31

[2] matplotlib.pyplot.imshow, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html Acessado em: 2021-04-05

[3] Image Filtering, https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#ga27c049795ce870216ddfb366086b5a04 Acessado em: 2021-04-04

[4] Module: exposure, https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.rescale_intensity Acessado em: 2021-04-06

[5] Adding borders to your images, https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.rescale_intensity Acessado em: 2021-04-09

[6] GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. 4th. ed. USA: Pearson Education Ltd., 2018. ISBN 0-13-335672-8