Trabalho 0

Marianna de Pinho Severo
MO443 - Introdução ao Processamento de Imagem Digital
Unicamp

20 de março de 2020

Algumas observações

O primeiro passo realizado para que fosse possível resolver todas as questões do trabalho foi a importação das bibliotecas que seriam usadas. Elas foram duas: a *numpy*, que nos permite realizar diversos tipos de operações numéricas sobre estruturas que podem ter uma ou mais dimensões; e a *cv2*, que nos permite trabalhar com a *opencv*.
Outro passo realizado em todas as questões é a leitura das imagens com as quais trabalhamos. Para lermos uma imagem utilizando *opencv*, podemos usar o método *imread*, em que o primeiro argumento é o caminho para a imagem e o segundo argumento indica o espaço de cores no qual queremos carregá-la. No caso de uma imagem monocromática, podemos passar o valor 0 para o segundo argumento. Assim, a intensidade de um pixel será representada por apenas um valor. Esse método retorna um array numpy, no formato de matriz, em que cada elemento representa um pixel da imagem.
Para salvarmos as imagens geradas, utilizamos, em todas as questões, o método *imwrite*, em que o primeiro argumento é o caminho e o nome com o qual desejamos salvar uma determinada imagem e o segundo argumento é a estrutura que a armazena.
Também utilizamos os métodos *imshow*, que gera uma nova janela com a imagem que passamos como argumento, *waitKey*, que espera que alguma tecla seja apertada para que o programa possa prosseguir, e *destroyAllWindows*, que fecha todas as janelas abertas pelo programa.
Para a resolução das questões, foram utilizadas as imagens disponibilizadas no Site da Disciplina.

Questão 01

Transformação de Intensidade - Transformar o espaço de intensidades (níveis de cinza) de uma imagem monocromática para:

1.a) Obter o negativo da imagem, ou seja, o nível de cinza 0 será convertido para 255, o nível 1 para 254 e assim por diante.

Para conseguirmos obter o negativo de uma imagem, conforme especificado na questão, podemos usar a função *bitwise_not* da biblioteca opencv. Ela realiza a operação *not* em cada bit do cojunto de bits de cada pixel da imagem.

Dessa forma, dada uma imagem em que cada pixel possui 8 bits, temos que os valores possíveis para a intensidade de cada pixel vão de 0 a 255. Assim, dado um pixel com valor 145, por exemplo, temos que sua representação binária é dada por: 10010001_2. Ao aplicarmos a função *bitwise_ not*, obteremos 01101110_2, que equivale a 110_{10}, que é o mesmo que $255 - 145 = 110$.
A Figura 1a apresenta a imagem antes da transformação e a Figura 1b mostra a imagem após a transformação pedida na Questão 1.a.

(a) Imagem original (b) Negativo da imagem

Figura 1: Imagens antes e depois de aplicar a transformação da Questão 1.a.

1.b) Converter o intervalo de intensidades para [100, 200].

Para convertermos o intervalo de intensidades dos pixels, utilizamos uma função linear. Para construirmos essa função, adotamos o seguinte raciocínio:
Quando o valor for 0, queremos que ele se torne 100, e quando ele for 255, queremos que ele se transforme em 200; entre 0 e 255 há 255 unidades, já entre 100 e 200 há 100 unidades. Entretanto, para mapear o primeiro intervalo para o segundo, queremos que todos os valores que cabem no primeiro também tenham valores correspondentes no segundo; dessa forma, dividimos 100 por 255, ou seja, quebramos o comprimento do segundo intervalo em 255 unidades. Assim, o valor mínimo será 0 e o máximo será 100. Todavia, como queremos que eles estejam entre 100 e 200, somamos 100 ao resultado obtido.
A função descrita pode ser observada na Equação 1, em que x é o valor de intensidade de um pixel da imagem:

$$f(x) = \frac{100}{255} * x + 100 \tag{1}$$

Um problema encontrado durante a resolução desse problema foi o tipo de dados da nova imagem gerada. Como estamos realizando uma operação de divisão em ponto flutuante, o resultado é um array de floats. Entretanto, a função *imshow* do *opencv* representa um array de floats como uma imagem em branco. Assim, convertemos os tipos dos elementos do array

de saída para o tipo *uint8* - que é o tipo dos dados da imagem original quando carregada - com o método *astype*. Apesar disso, a função *imwrite* consegue salvar a imagem, mesmo que seus valores sejam do tipo float.
Na Figura 2 é possível observar a imagem antes e depois da transformação.

(a) Imagem original (b) Intensidades no intervalo [100,200]

Figura 2: Imagens antes e depois de aplicar a transformação da Questão 1.b.

1.c) Inverter os valores dos pixels das linhas pares da imagem, ou seja, os valores dos pixels da linha 0 serão posicionados da direita para esquerda, os valores dos pixels da linha 2 serão posicionados da direita para a esquerda e assim por diante.

Nesta questão, primeiro salvamos a imagem em uma estrutura auxiliar (new_image) para não alterarmos a imagem original. Para selecionarmos apenas as linhas pares, percorremos o array acessando seus elementos a cada duas linhas, inclusive a primeira, como pode ser visto em **new_image[::2]**.
Para invertermos os valores dos pixels, utilizamos o método *flip* do *opencv*. Como primeiro argumento, especificamos o array de valores que queremos inverter, e o segundo argumento indica em que direção inverteremos os valores (1 significa que giraremos em torno do eixo y). O resultado dessa transformação pode ser visto na Figura 3

1.d) Espelhar as linhas da metade superior da imagem na parte inferior da imagem.

Para espelharmos as linhas da metade superior para a parte inferior, primeiro calculamos o meio da imagem, ou seja, a linha que se encontrava na metade da matriz. Depois percorremos a matriz a partir do meio para baixo, substituindo seus valores pelos valores da metade superior invertidos. Para invertermos a parte superior, aplicamos novamente o método *flip*, mas dessa vez passando o valor 0 como argumento, indicando que queríamos girar em torno do eixo x.
Na Figura 4 podemos observar o resultado do espelhamento.

(a) Imagem original (b) Linhas pares invertidas

Figura 3: Imagens antes e depois de aplicar a transformação da Questão 1.c.

(a) Imagem original (b) Imagem espelhada verticalmente

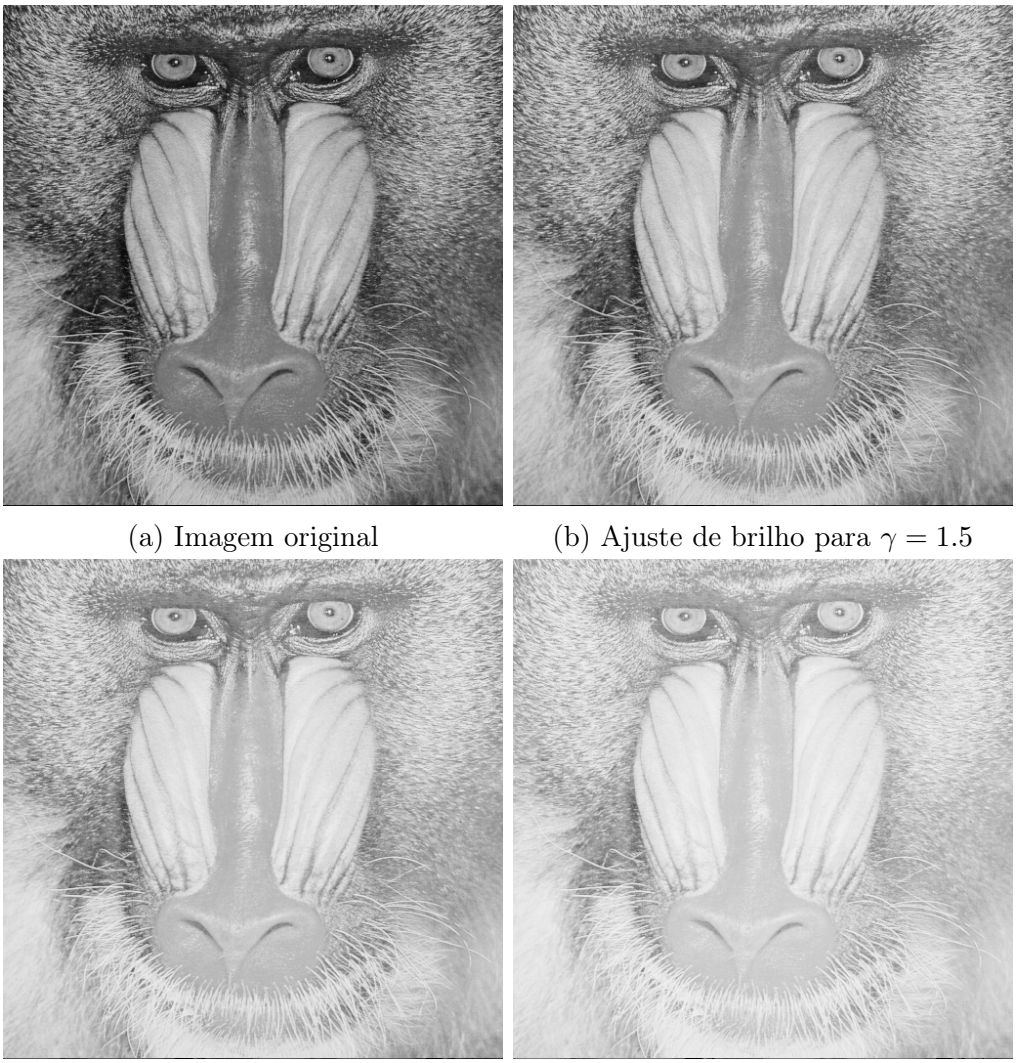
Figura 4: Imagens antes e depois de aplicar a transformação da Questão 1.d.

Questão 02

Ajuste de Brilho - Aplicar a correção gama para ajustar o brilho de uma imagem monocromática A de entrada e gerar uma imagem monocromática B de saída. A transformação pode ser realizada (a) convertendo-se as intensidades dos pixels para o intervalo de [0,255] para [0,1], (b) aplicando-se a equação $B = A^{(1/\gamma)}$ e (c) convertendo-se os valores resultantes de volta para o intervalo [0, 255].

Para ajustarmos o brilho da imagem, utilizamos o passo a passo indicado no enunciado da questão. Primeiro, convertemos as intensidades dos pixels da imagem do intervalo de [0,255] para o de [0,1]. Para isso, dividimos os valores de cada pixel da imagem por 255.

Então, aplicamos a equação $B = A^{\frac{1}{\gamma}}$, em que A é a imagem cuja escala foi transformada e B é a saída ao aplicarmos a transformação com Gama. Por fim, transformamos a escala de volta para o intervalo [0,255], multiplicando os valores por 255, e convertemos o tipo de dados de cada elemento do array que armazena a imagem de saída para o tipo *uint8*.
Nas Figuras 5a, 5b, 5c e 5d, podemos observar a imagem original e os resultados das transformações para $\gamma = 1.5$, $\gamma = 2.5$ e $\gamma = 3.5$, respectivamente.



(a) Imagem original (b) Ajuste de brilho para $\gamma = 1.5$

(c) Ajuste de brilho para $\gamma = 2.5$ (d) Ajuste de brilho para $\gamma = 3.5$

Figura 5: Imagens antes e depois de aplicar a transformação da Questão 2.

Questão 03

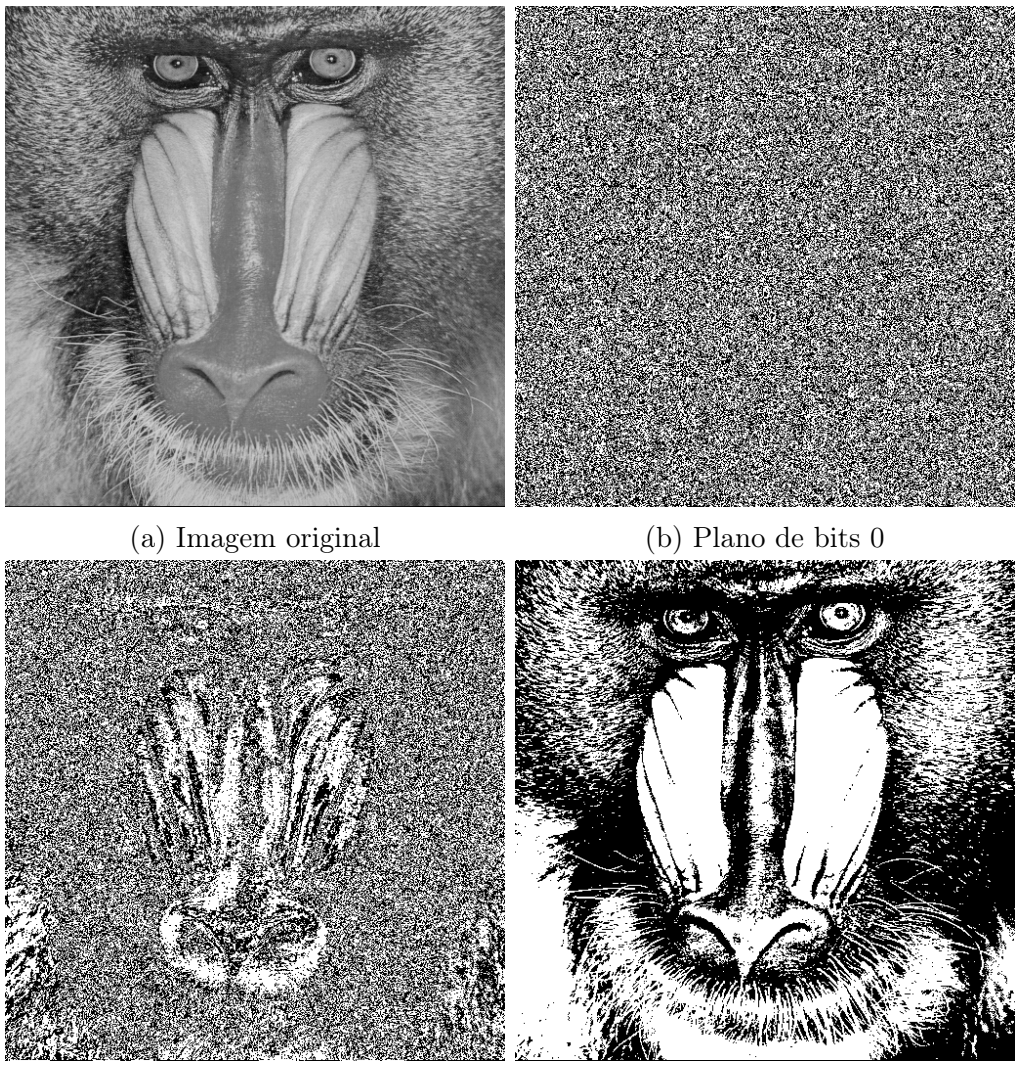
Planos de Bits - Extrair os planos de bits de uma imagem monocromática.

Planos de bits consistem em uma representação dos valores de cada bit dos pixels de uma imagem. Em outras palavras, imagine uma imagem de 8 bits, ou seja, cada um de seus pixels possui 8 bits - valores podem ir de 0 a 255. Essa imagem possui 8 planos de bits, que vão desde o plano 0 até o plano 7. No plano 0 representamos os valores do bit 0 de cada pixel, no plano 1 representamos os valores do bit 1 de cada pixel e assim por diante. Dessa maneira, cada plano corresponde a uma matriz com as mesmas dimensões da imagem original, mas com os valores dos bits correspondentes ao plano observado.
Para calcularmos os planos de bits da imagem monocromática carregada, fazemos uma operação *bitwise and* para o bit que queremos. Para isso, elevamos 2 à posição do bit, o que resultará em um valor cuja representação binária possuirá 1 apenas na posição do bit desejado. Então, deslocamos o valor obtido da operação bitwise para a direita, até que ele chegue à posição 0. Isso resultará em valores 0 ou 1. Por fim, como queremos que 0 seja a cor mais escura e 1 seja a mais clara, multiplicamos o resultado por 255. Um exemplo pode ser visto abaixo:
Imagine que queremos o valor do bit 4 do pixel cujo valor é 10010101_2, então fazemos $2^4 = 16_{10} = 00010000_2$. A seguir, realizamos a operação *bitwise and*: 10010101 & 00010000 = 00010000. Então, deslocamos o valor para a direita 4 posições: (1)0000**1**000 $\rightarrow$ (2)00000**1**00 $\rightarrow$ (3)00000**1**0 $\rightarrow$ (4)00000000**1**. Assim, descobrimos que o valor do bit 4 do pixel observado é 1. Multiplicar por 255 fará com que 0 produza o pixel mais escuro, enquanto 1 produzirá o pixel mais claro.
Nas Figuras 6a, 6b, 6c e 6d, podemos observar a imagem original e os resultados para os planos de bits 0, 4 e 7, respectivamente.

Questão 04

Mosaico - Construir um mosaico de 4 x 4 blocos a partir de uma imagem monocromática.

Para resolver essa questão, primeiro determinamos o número de linhas (n_lines) e colunas (n_cols) da imagem, em quantas partes ela seria divida horizontal e verticalmente (n_slices) e a quantidade de pixels por linha (n_cols_slice) e por coluna (n_lines_slice) de cada parte. Também, criamos uma lista (pattern) com a ordem em que cada parte da imagem deve ser reorganizada para formar o mosaico desejado. Além disso, foram criados um dicionário (img_slices) para armazenar as partes a serem retiradas da imagem, e uma matriz (mosaic) com a mesma dimensão da imagem original, para armazenar o mosaico.
O próximo passo após definir as estruturas foi percorrer a imagem original, retirando matrizes menores (pedaços) de tamanho n_cols_slice * n_lines_slice e salvando no dicionário *img_slices* (primeiro par de loops), em que a chave representa a posição da matriz menor - que vai de 1 a 16 - e o valor é a matriz menor. A imagem foi percorrida da esquerda para a direita e de cima para baixo.
Uma vez possuindo todos os pedaços da imagem original, preencheu-se a matriz *mosaic*, também da esquerda para a direita e de cima para baixo, mas colocando as matrizes menores correspondentes à ordem estabelecida na estrutura *pattern*.
Na Figura 7 é possível observar as imagens antes e após a criação do mosaico.



(a) Imagem original (b) Plano de bits 0

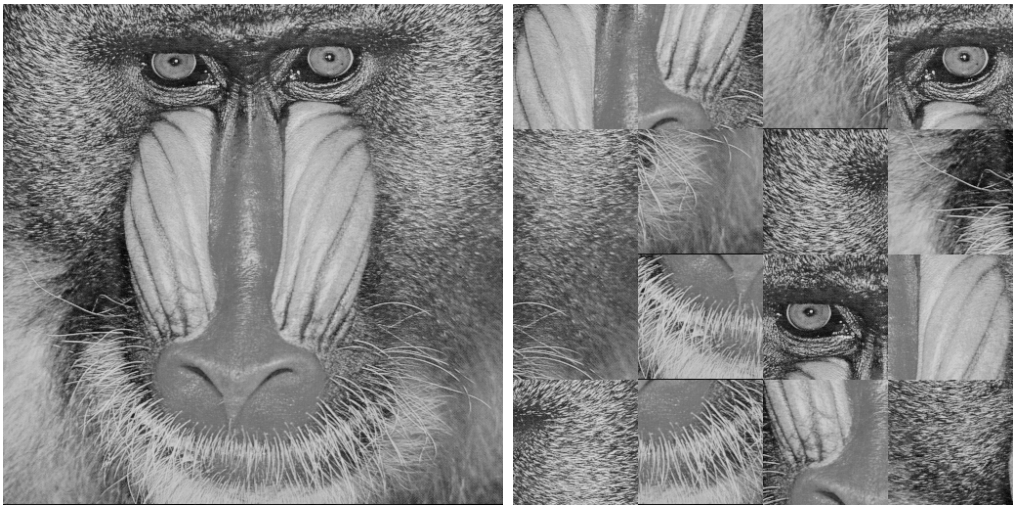
(c) Plano de bits 4 (d) Plano de bits 7

Figura 6: Imagem original e planos de bits 0, 4 e 7.

Questão 05

Combinação de Imagens - Combinar duas imagens monocromáticas de mesmo tamanho por meio da média ponderada de seus níveis de cinza.

Para resolvermos essa questão, usamos o método *addWeighted* do *opencv*. Ele realiza a soma ponderada das imagens, ou seja, multiplica cada imagem pelo respectivo coeficiente antes de somá-las.
O primeiro argumento é a estrutura que representa a primeira imagem, o segundo é o coeficiente que multiplicará seus elementos, o terceiro argumento é a segunda imagem, o quarto é o coeficiente que multiplicará os elementos desta e o quinto argumento, chamado de *bias*, adiciona um valor ao resultado da operação. A Equação 2 apresenta a fórmula calculada por esse método.



(a) Imagem original (b) Mosaico

Figura 7: Imagem antes e depois da transformação da Questão 4.

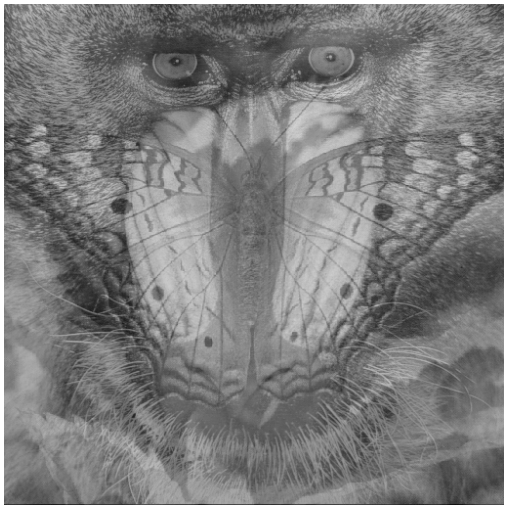
$$output\_image = image1 * \alpha + image2 * \beta + \gamma \tag{2}$$

Nas Figuras 8a e 8b, podemos observas as imagens originais utilizadas. Já nas Figuras 9a, 9b e 9c, podemos ver os resultados para quando α e β variam entre os valores [0.2, 0.5 e 0.8].

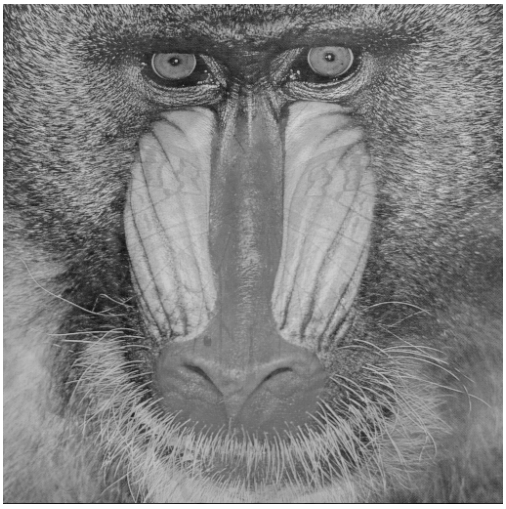
(a) Imagem 1 original (b) Imagem 2 original

Figura 8: Imagens originais da Questão 5.

(a) Combinação com $\alpha = 0.2$ e $\beta = 0.8$



(b) Combinação com $\alpha = 0.5$ e $\beta = 0.5$



(c) Combinação com $\alpha = 0.8$ e $\beta = 0.2$

Figura 9: Resultados das transformações da Questão 5.