Trabalho 1 de Introdução ao Processamento de Imagens

Nome: Miguel Augusto S. Guida - RA: 174847

29 de Março de 2020

1 Introdução

O objetivo deste trabalho é realizar alguns processamentos básicos em imagens digitais. Quando pertinente a vetorização dos comandos deve ser empregada na implementação.
O trabalho foi dividio em 3 problemas.
O primeiro consiste em alterar a resolução de uma imagem, diminuindo a densidade de pixels e reduzindo sucessivamente pela metade a resolução da imagem.
O segundo problema trata de quantização de imagens, e consiste em diminuir a profundidade de uma imagem monocromática (bits necessários para armazenar a imagem), para representar uma imagem com diferentes níveis de quantização.
O terceiro tem o objetivo de alterar a Escala de Cinza de uma imagem. Para isso, aplicaremos transformaçooes lineares e não-lineares na imagem. Foram implementadas 5 transformaçooes diferentes: logaritmo, exponencial, quadrado, raiz quadrada e alargamento de contraste.

2 O programa

O programa foi desenvolvido no Jupyter Notebook, utilizando Python 3.7.6. As bibliotecas utilizadas foram Numpy 1.18.1, OpenCV 4.2.0, Matplotlib 3.1.3.

2.1 Como executar

Cada secção do programa corresponde a uma tarefa proposta no trabalho. Executar o programa através do comando "Run" do Jupyter Notebook.

2.2 Entrada

A entrada do programa foi definida como a imagem "baboon.png", e está em anexo com o programa.

2.3 Saída

O programa irá gerar diversas imagens baseado na imagem de entrada, e serão adicionadas no diretório em que se encontra o programa. Muitas delas estão apresentadas no relatório.

3 Resolução de Imagens

A ideia inicial para resolver o problema foi calcular a média de intensidades de cada secção 2x2 pixels da imagem original "A" e armazenar o valor obtido no pixel correspondente de uma nova matriz "B". A nova matriz possui dimensooes correspondentes à metade da imagem original e o valor médio das amostras é mantido. Dessa forma, obtemos "B" com metade da resolução de "A".
Tomando "B" como a imagem original nas iteraçooes seguintes e operando esse processo sucessivamente, obtemos imagens com resolucoes menores, sempre mantendo o valor médio das amostras correspondentes ao setor de pixels da imagem original e reduzindo a resolução pela metade.
O algoritmo foi implementado utilizando métodos vetoriais para percorrer a imagem original, assim como para calcular a média dos valores dos pixels (utilizando numpy.mean()).

Por mais que tenhamos chegado ao resultado desejado utilizando o algoritmo descrito acima, podemos ainda substituir o cálculo da média de secçooes 2x2 da matriz original pela função resize() da biblioteca OpenCV. Passando as dimensooes da imagem original divido por 2 e o fator de interpolação "interpolation = INTER_AREA" como parâmetro, obtemos o mesmo resutado desejado, mantendo as proporçooes e intensidades da imagem redimensionada, da mesma forma que calculamos no método anterior e de forma mais eficiente.
Para que pudéssemos manter as dimensooes originais ao salvar as imagens redimensionadas, utilizamos a função cv2.resize(), passando as dimenssooes da imagem original e o fator de interpolação "interpolation = INTER_AREA" como parâmetro.
O resultado do algoritmo está apresentado na Figura 1.

4 Quantização de Imagens

Como o objetivo deste problema consiste em representar uma imagem com diferentes níveis de quantização, foi desenvolvido um algoritmo que desloca para a direita os bits dos pixels de forma vetorial.
Dessa forma, alteramos a profundidade da imagem, e consequentemente, diminuímos a quantização da imagem, ou seja, o número de níveis de cinza presente na imagem monocromática utilizada. Para o deslocamento dos bits, utilizamos a função "shift_right" do Numpy, que realiza esta operação vetorialmente. A cada deslocamento à direita dos bits, dividimos a quantidade de níveis de cinza por 2, obtendo através de uma imagem com 256 níveis, imagens com 128, 64, 32,

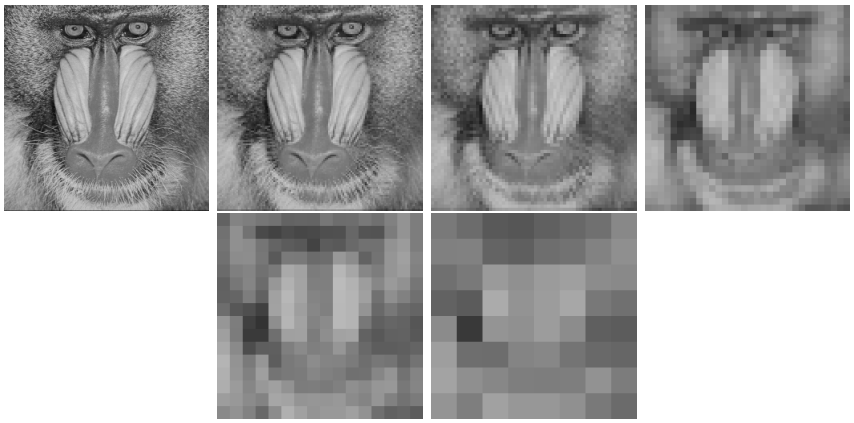

Figure 1: Imagem com resolução reduzida sucessivamente pela metade

16, 8, 4 e 2 níveis de cinza. As imagens obtidas estão apresentadas na Figura 2. Um fato interessante é que, por diminuirmos os bits necessários para representar a imagem ao executar o "shift right", podemos perceber que o tamanho da imagem diminui depois dessa operação.

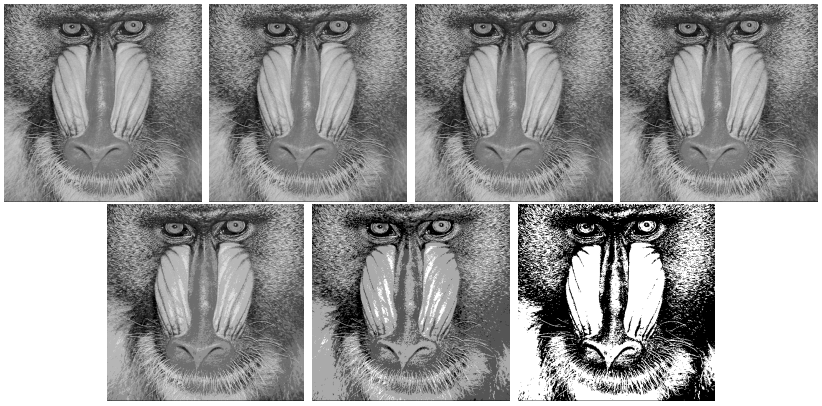

Figure 2: Imagem com diferentes níveis de cinza

5 Escala de Cinza

Neste problema, aplicamos transformaçooes lineares e não lineares na imagem escolhida. As transformaçooes propostas teem como objetivo gerar uma nova imagem com alteraçooes no brilho e contraste da imagem original. Essa mudança é possivel através da manipulação dos níveis de cinza da imagem.

5.1 Função Logaritmo

Nesta transformação, cada valor de pixel da imagem é subtituído pelo seu logaritmo. No caso desta aplicação, optou-se por utilizar o logaritmo de base 10. Pelo fato de que o logaritmo de um número é um valor baixo, esta transformação enaltece os pixels de menor intensidade, presentes nas partes mais escuras da imagem.
O algoritmo desenvolvido recebe uma imagem de entrada e aplica a função logaritmo em todos seus pixels. A aplicação é realizada através da função log10() do Numpy, que calcula o logaritmo de base 10 para os valores passados como parâmetro.

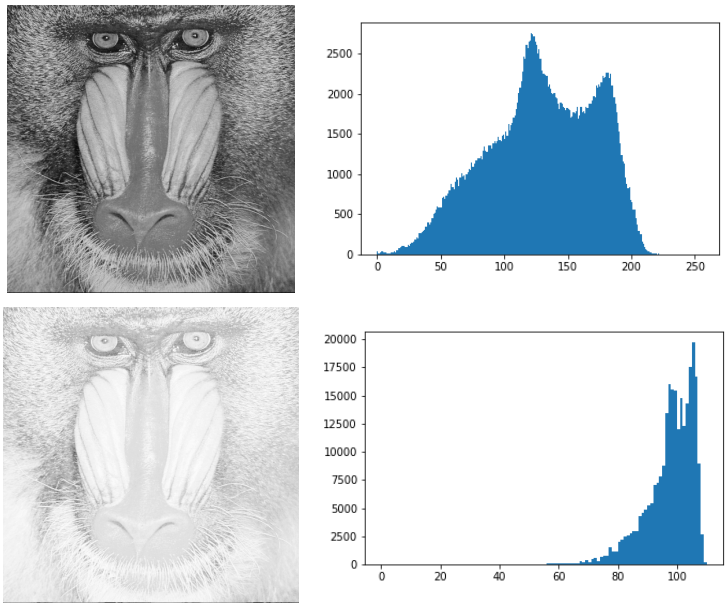

Figure 3: Imagem Original e Histograma x Imagem transformada pela Função Logaritmo e Histograma

5.2 Função Exponencial

Nesta transformação, cada valor de pixel da imagem é subtituído pelo seu exponencial. Esta transformação enaltece os pixels de maior intensidade, presentes nas partes mais claras da imagem.
Por estarmos calculando o exponencial dos valores contidos nos pixels, utilizamos uma técnica de normalização antes de aplicar a função exponencial

na imagem, para não termos casos de overflow, convertendo assim a escala de cinza de [0:L] para [0:1]. Multiplicamos todos pixels por uma constante "ratio", de valor 1/max, onde max é o valor do pixel de maior intensidade da imagem. Para converter os pixels de volta da normalização, multiplicamos cada pixel por uma constante "a", de valor max/np.exp(1) (np.exp(1) representa o exponencial do maior pixel).
O algoritmo desenvolvido recebe uma imagem de entrada e aplica a função exponencial em todos seus pixels. A aplicação é realizada através da função exp() do Numpy, que calcula o exponencial para os valores passados como parâmetro.

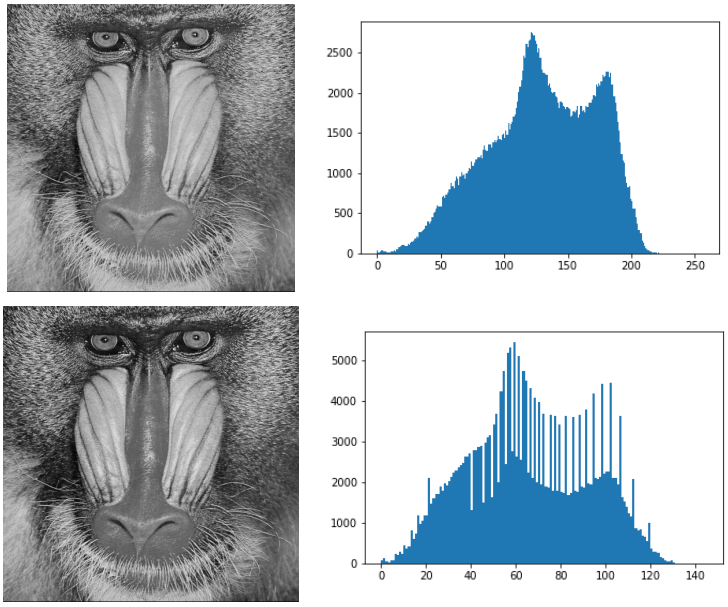

Figure 4: Imagem Original e Histograma x Imagem transformada pela Função Exponencial e Histograma

5.3 Função Quadrado

Nesta transformação, cada valor de pixel da imagem é subtituído pelo seu valor elevao ao quadrado. Esta transformação, assim como a exponencial, enaltece os pixels de maior intensidade da imagem.
Aqui também utilizaremos a normalização do valor dos pixels para evitar que ocorra overflow. A normalização será feita de forma semelhante à do exponencial, com a difereça que a constante "a" será igual ao valor do pixel de maior intensidade.
O algoritmo desenvolvido recebe uma imagem de entrada e aplica a função

quadrática em todos seus pixels. A aplicação é realizada através da função multiply() do Numpy, que calcula o valor da multiplicação dos dois parâmetros. Neste caso passamos a matriz normalizada da imagem duas vezes como parâmetro, para receber a imagem ao quadrado.

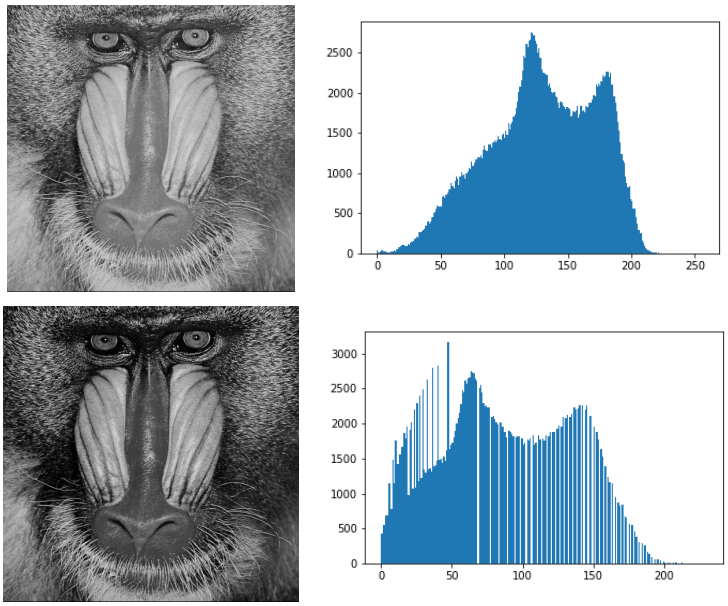

Figure 5: Imagem Original e Histograma x Imagem transformada pela Função Quadrado e Histograma

5.4 Função Raiz Quadrada

Nesta transformação, cada valor de pixel da imagem é subtituído pela raiz quadrada do seu valor. Esta transformação aumenta o contraste das regiooes da imagem com baixa e média intensidades.
O algoritmo desenvolvido recebe uma imagem de entrada e aplica a função raiz quadrada em todos seus pixels. A aplicação é realizada através da função sqrt() do Numpy, que calcula a raiz quadrada para todos pixels da imagem passada como parâmetro.
Podemos observar o resultado da transformação na Figura 6, que apresenta a imagem original e seu histograma em comparação com a imagem transformada e seu histograma.

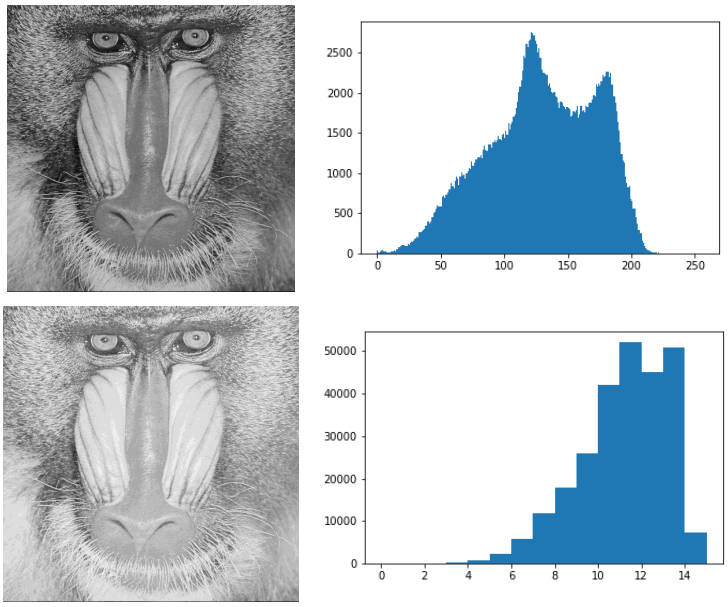

Figure 6: Imagem Original e Histograma x Imagem transformada pela Função Raiz Quadrada e Histograma

5.5 Alargamento de Contraste

Este problema consiste em uma transformação linear por partes.
Para pixels dentro de tres intervalos, deveriam ser atribuídos valores diferentes, de acordo com as regras presentes na função:

$$g = \begin{cases} \alpha f & se\ \ 0 \leq f \leq a \\ \beta(f-a) + \alpha a & se\ \ a < f \leq b \\ \gamma(f-b) + \beta(b-a) + \alpha a & se\ \ b < f \leq L \end{cases}$$

Definimos os intervalos atribuindo valores para a, b, e L da função, sendo a = 64, b = 128 e L = 255. Para os valores de alpha, beta e gamma , utilizamos valores variáveis, para fazer análises diferentes. A Figura 7 exibe 3 casos de alargamento de contraste para valores diferentes de alpha, beta e gamma. Abaixo da imagem original está o caso em que alpha = 0.5, beta = 0.5 e gamma = 1.5.
Em seguida o caso em que alpha = 0.5, beta = 1.5 e gamma = 0.5.
E por fim, o caso em que alpha = 1.5, beta = 0.5 e gamma = 0.5.

Para aplicar transformação na imagem de entrada, utilizei a função where() do Numpy. Utilizei where() aninhados para definir o intervalo de intensidades da imagem, e aplicar as regras respectivas a cada intervalo.

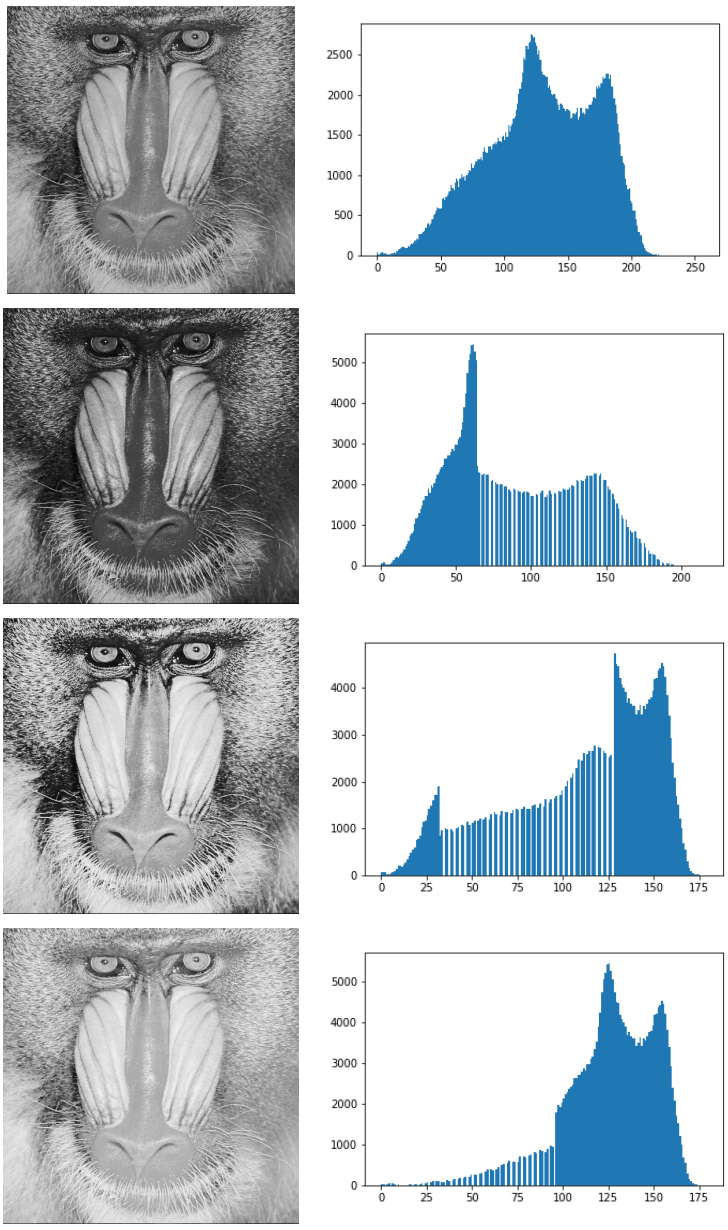

Figure 7: Imagem Original e Histograma x 3 Transformaçooes de Alargamento de Contraste e Histogramas