Trabalho #0:

MO443 – Introdução ao Processamento de Imagem Digital

Mário A. Costa Jr.

RA: 226496

mariokozta@gmail.com

I. Especificação dos Problemas

O processamento de imagem digital, faz parte do nosso cotidiano, temos como exemplo, a identificação biométrica de digital ou de face que utilizam diversas técnicas de processamento de imagem. Entretanto, para que o processo de aprendizado desta técnica seja trilhado, é necessário entender alguns conceitos básicos de processamento apresentados neste trabalho proposto, assim como, o uso de vetorização [1] de comandos que comprovadamente otimizam estas operações se implementadas de forma correta.

Os processamentos básicos apresentados nestas atividades consistem em realizar transformações nas imagens, como as descritas abaixo:

- Negativo da imagem.
- Espelhamento vertical.
- Converter intervalo de intensidade.
- Inversão de pixel de linhas pares.
- Espelhamento de metade da imagem.
- Ajuste de brilho
- Separar planos de bits.
- Construção de mosaico.
- Combinação de Imagens.

II. Entrada de dados

A solução apresentada para executar as atividades propostas pelo trabalho, foi desenvolvida em Python e através de um arquivo chamado **trabalho0.py**. Este aplicativo foi desenvolvido para aceitar imagens em tons de cinza no formato RGB do tipo PNG (*Portable Network Graphics*). Para executar o aplicativo, o usuário deve chamar o arquivo trabalho0.py. Após a execução do aplicativo, o sistema irá requisitar ao usuário informações apresentadas abaixo.

```
Insira o caminho do diretório (ex: /home/teste/):
Insira o nome da imagem 1 (ex: city.png)
Insira o nome da imagem 2 (ex: baboo.png)
Insira o nome da imagem 3 (ex: butterfly.png)
```

Inserida as informações requisitadas, o sistema irá executar todas as atividades básicas propostas por este trabalho.

Obs: O aplicativo foi testado com as imagens disponíveis na web (http://www.ic.unicamp.br/~helio/imagens_png/).

III. Código e Decisões Tomadas

As imagens selecionadas pelo usuário serão carregadas no sistema com o auxílio da biblioteca **scikit-image** [2] modulo **io**, através do comando **imread**,

que irá gerar uma saída em uma matriz I_{mxn} onde *m* representa o número de linhas e *n* o número de coluna, cada posição desta matriz está preenchida com valores entre 0 e 255 (que definem os tons de cinza em um pixel) de acordo com a imagem selecionada. Para realizar a primeira operação e obter o negativo da imagem conforme a figura fig: 1.1 Atividade – B, utiliza da vetorização [1], para executar a operação proposta, foi executada através do código apresentado abaixo.

```
image = 255 – image
```

Esta operação inverte os valores dos tons de cinza contidos em cada posição da matriz.

Fig: 1.1 Atividade – B

A atividade de espelhamento vertical figura: fig: 1.1 Atividade – C; foi realizada em 2 passos: 1) realiza o processo de espelhamento da imagem através do comando *fliplr*; 2) executa o processo de rotação de 180º da imagem através do comando *rotate*; como apresentado no quadro abaixo.

```
Image = fliplr(image)
Image = rotate(image,angle=180)
```

O resultado desta operação apresenta a imagem original com efeito de espelhamento invertido.

Fig: fig: 1.1 Atividade – C

A atividade de conversão do intervalo de intensidade figura: fig: 1.1 Atividade – D; altera todos os valores de tons de cinza presentes na matriz da imagem, todos valores que estão abaixo de 100, serão alterados para o valor 0, todos valores que estão acima de 200, serão alterados para o valor 200. Para isso, foi utilizada da vetorização, os comandos utilizados estão apresentados no quadro abaixo.

```
image[image < 100] = 0
image[image > 200] = 255
```

O resultado desta operação, apresenta uma imagem com uma alteração na intensidade dos tons de cinza, apresentando linhas mais definidas.

fig: 1.1 Atividade – D

Atividades de inversão das linhas figura: fig: 1.1 Atividade – E; tem como objetivo inverter as linhas pares das imagens, em outras palavras, as linhas selecionadas terão os valores da esquerda para direita, reposicionados da direita para esquerda. Para isso, foi utilizada a técnica de vetorização, através do operador *slice,* iniciado pelo 0, com saltos de 2 posições, o que leva a captura das linhas pares. Como apresentado no comando abaixo.

```
image[0::2, :] = image[0::2, ::-1]
```

O resultado desta operação, apresenta uma imagem espelhada e uma imagem original, sem o efeito da sobreposição, porém afeta a definição da imagem.

fig: 1.1 Atividade – E

A atividade de espelhamento de metade da imagem conforme figura: fig: 1.1 Atividade – F; tem como objetivo espelhar a metade superior da imagem na parte inferior. Para realizar esta atividade, foi identificado o número de linhas da imagem através do comando *shape* e dividido por 2 por meio do comando *trunc* da biblioteca *math*[4], após esta etapa, foi utilizado da técnica de vetorização para capturar a metade superior da imagem e espelhar a mesma metade capturada, já a união das duas metades foi realizada por meio do comando *vstack* da biblioteca *numpy* [3], os comandos utilizados estão apresentados no quadro abaixo.

```
dim = image.shape
mid = math.trunc((dim[0]/2))
a = image[0:mid:1]
b = image[mid::-1]
image = np.vstack((a,b))
```

O resultado desta atividade, irá apresentar a reflexão horizontal da imagem de forma duplicada de forma invertida, como um um efeito espelho.

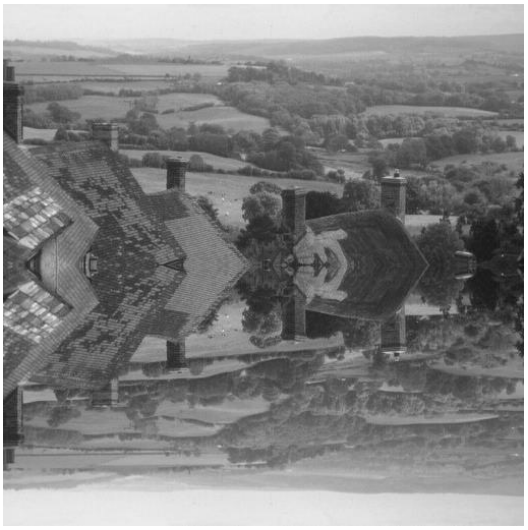

fig: 1.1 Atividade – F

A atividade de ajuste de brilho, tem como proposta transformar uma imagem de acordo com o fator de brilho Gama e foi através da formula $B = A^{(1/Y)}$, Sendo A = entrada, B = saída, Y = gama. Esta atividade, proporciona a variação de gama para os seguintes valores: 1.5, 2.5, 3.5. Para desenvolver esta solução, foi localizado o maior valor da matriz através do comando *max* da biblioteca *numpy*, este número foi transformados em ponto flutuante por meio do comando *float* e dividido por todos elementos da imagem, após esta transformação, estes elementos foram elevados a potência de (1/gama), onde gama é a variável a ser alterada, o comando *power* biblioteca *numpy* foi um dos comandos utilizados para

realizar esta operação. Cada imagem gerada por esta atividade, apresenta um fator de gama para cada Figura abaixo: 1.2 Brilho fator 1.5, Brilho fator 2.5 e Brilho fator 3.5. Os comandos desta atividade estão apresentados no quadro abaixo.

```
fator = 1.5
image =
np.power(image/float(np.max(image)),1/fator)
fator = 2.5
image =
np.power(image/float(np.max(image)),1/fator)
fator = 3.5
image =
np.power(image/float(np.max(image)),1/fator)
```

O resultados obtidos, são imagens com diferentes tons de cinza, quanto maior o gama aplicado, maior é intensidade de brilho presente na imagem.

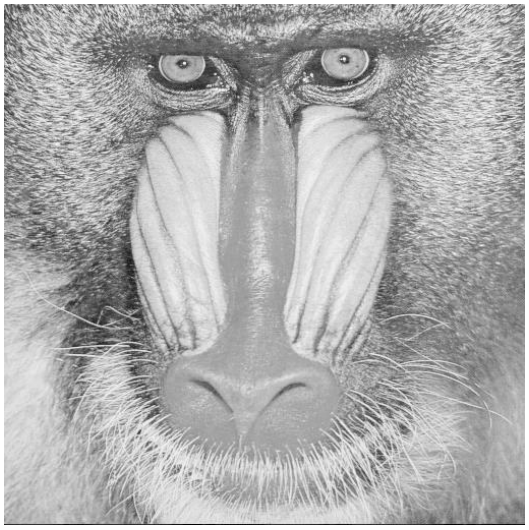

fig: 1.2 Brilho fator 1.5

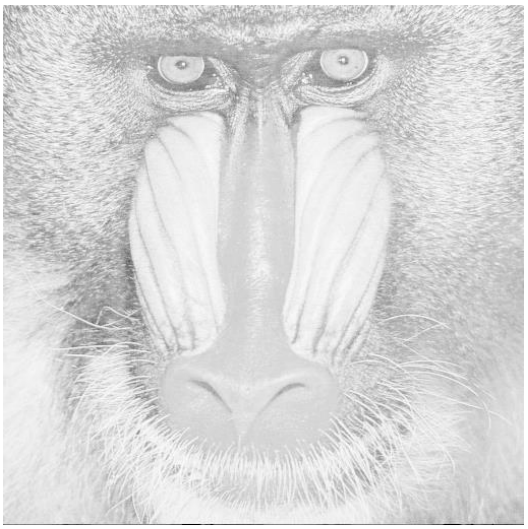

fig: 1.2 Brilho fator 2.5

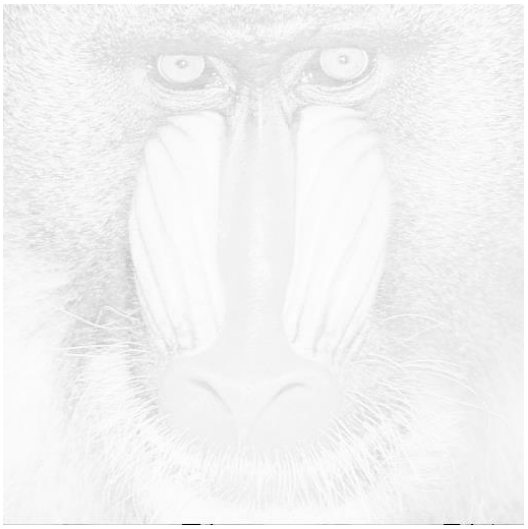

fig: 1.2 Brilho fator 3.5

A atividade plano de bits, consiste em fatiar a imagem monocromática fornecida e extrair o conjunto de bits (fatia) que descreve um pixel. Quando estes conjuntos estão sobrepostos eles formam uma imagem. Para extrair estes planos, a primeira etapa a ser realizada consiste em aplicar a formula 2**x (2 elevado ao quadrado) para toda matriz, onde x, significa o plano de bit a ser identificado, por exemplo: 2**1 (plano 1) ou 2**4 (plano 4). Após implementada operação, é executada uma operação binária entre a matriz que foi elevada ao quadrado (cálculo anterior), com a matriz da imagem sem alteração, para extrair os bits iguais entre as matrizes, o comando

bitwise_and biblioteca *cv2* foram utilizados para realizar este atividade, após esta função, o valor de 255 em todos elementos com salto de 1 (este valor representa o bit selecionado ou plano, quando maior que 0 (este 0 representa o range de 0 a 255 dos tons de cinza). Para ficar mais claro os comandos utilizados nesta atividade serão apresentados no quadro abaixo com exemplo de bit 4.

```
x=4
arr[:,:,x] = 2**x
row[:,:,x] = cv2.bitwise_and(image,arr[:,:,x])
mask=row[:,:,x]>0
row[mask]=255
cv2.imshow('1.3 - Plano de Bit 4',row[:,:,x])
```

O resultado desta operação, é a apresentação das imagens no plano de bit selecionado, neste caso apresentado no quadro acima, o plano 4 será exibido.

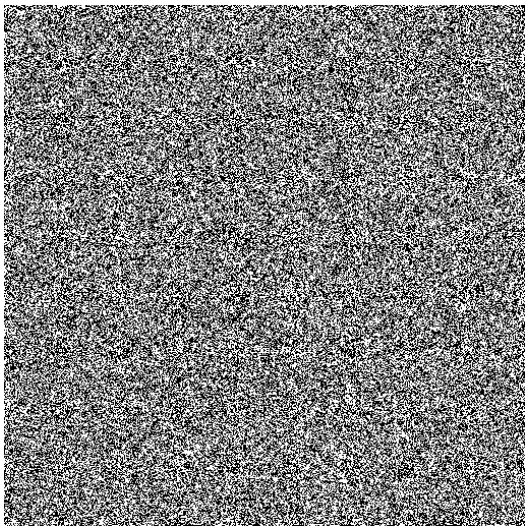

fig: 1.3 Plano de Bit 1

As figuras : 1.3 plano de Bit 1, 1.3 plano de Bit 4 e 1.3 plano de Bit 7, apresentam os resultados dos filtros aplicados.

fig: 1.3 Plano de Bit 4

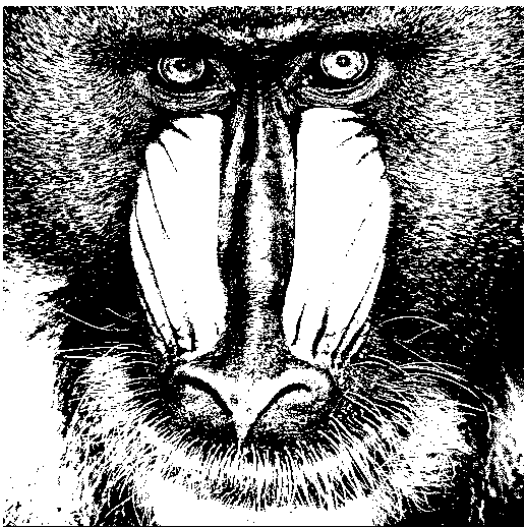

fig: 1.3 Plano de Bit 7

A atividade da construção do mosaico, consiste em extrair as dimensões da imagem através do comando *shape*, para que estes pedaços sejam divididos para e alocados em uma matriz de dimensão 4x4. Esta imagem foi dividida de forma sequencial, iniciado pela linha, da esquerda para direita, por exemplo: linha 1 (1, 2, 3, 4), linha 2 (5, 6, 7, 8), linha 3 (9, 10, 11, 12), linha 4 (13, 14, 15, 16). Desta forma foi possível povoar as posições da divisão da imagem selecionada na matriz criada, através de um comando de enlace, que separa os pedaços da imagem e insere estes pedaços em uma posição do novo array (array criado). Após esta ação, a matriz é concatenada através do comando

concatenate biblioteca *numpy*, a figura: 1.4 – Mosaico – Original; apresenta a imagem original reconstruída conforme o exemplo de divisão da imagem. Já a figura: 1.4 – Mosaico - Nova ordem; apresenta as posições da imagem de forma embaralhada conforme especificado pela tarefa: linha 1 (6, 11, 13, 3), linha 2 (8, 16, 1, 9), linha 3 (12, 14, 2, 7), linha 4 (4, 15, 10, 5), de acordo com a atividade proposta. Para ficar mais claro, vamos apresentar os comandos utilizados no quadro abaixo:

```
for i in range(0,nRows):
   for j in range(0, mCols):
      roi =
image[i*int(sizeY/nRows):i*int(sizeY/nRows) +
int(sizeY/nRows)
,j*int(sizeX/mCols):j*int(sizeX/mCols) +
int(sizeX/mCols)]
      size[i][j] = np.asarray(roi)

#concatena (imagem original)
lin1 =
np.concatenate((size[0][0],size[0][1],size[0][2],siz
e[0][3]),axis=1)
lin2 =
np.concatenate((size[1][0],size[1][1],size[1][2],siz
e[1][3]),axis=1)
lin3 =
np.concatenate((size[2][0],size[2][1],size[2][2],siz
e[2][3]),axis=1)
lin4 =
np.concatenate((size[3][0],size[3][1],size[3][2],siz
e[3][3]),axis=1)
aux = np.concatenate((lin1,lin2,lin3,lin4),axis=0)

#concatena (nova ordem)
lin1 = np.concatenate((size[1][1], size[2][2],
size[3][0], size[0][2]),axis=1)
lin2 = np.concatenate((size[1][3], size[3][3],
size[0][0], size[2][0]),axis=1)
lin3 = np.concatenate((size[2][3], size[3][1],
size[0][1], size[1][2]),axis=1)
lin4 = np.concatenate((size[0][3], size[3][2],
size[2][1], size[1][0]),axis=1)
aux = np.concatenate((lin1,lin2,lin3,lin4),axis=0)
```

Os resultados obtidos nestas operações podem ser observados nas figuras relatadas nesta atividade.

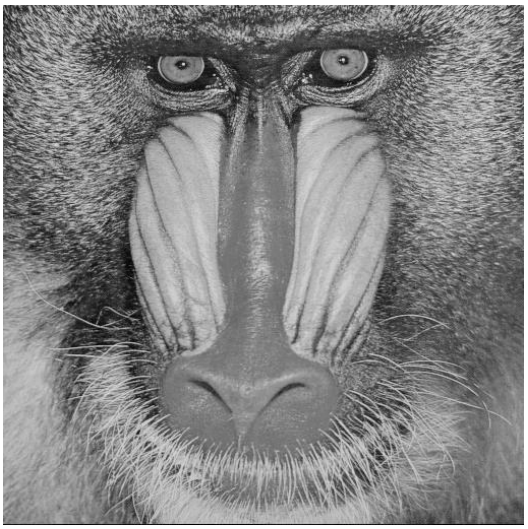

fig: 1.4 Mosaico (Original)

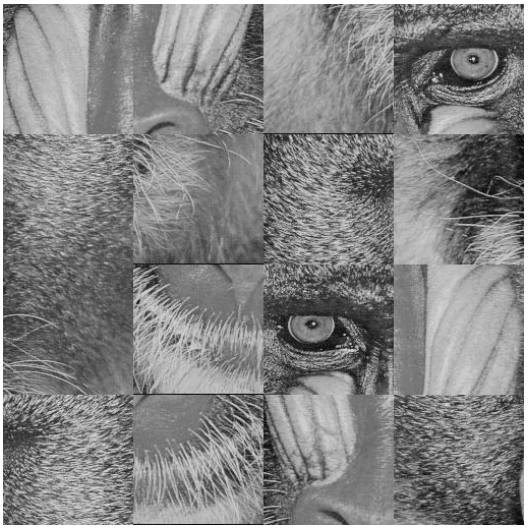

fig: 1.4 Mosaico (Nova ordem)

Por último é apresentada a atividade de combinação de imagens. Esta atividade consiste em misturar as imagens através do comando *addWeighted* biblioteca *cv2*, e apresentar cada uma das figuras , com mais ou menos ênfase de acordo com a configuração proposta pela atividade. Para ficara mais claro, no quadro abaixo está o comando executado.

```
sobre = cv2.addWeighted(im1,0.2,im2,0.8,0)
```

As figuras abaixo apresentam cada uma das operações requisitadas pela atividade proposta. É importante salientar que: quanto maior o valor da inserido na configuração do comando

(e.g. 0.8), mais a imagem tem destaque na apresentação. Como pode ser observado abaixo.

fig: 1.5 - Combinação de Imagem (0.2 + 0.8)

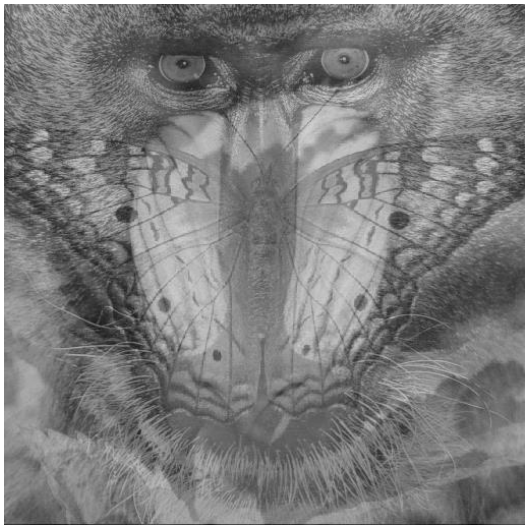

fig: 1.5 - Combinação de Imagem (0.5 + 0.5)

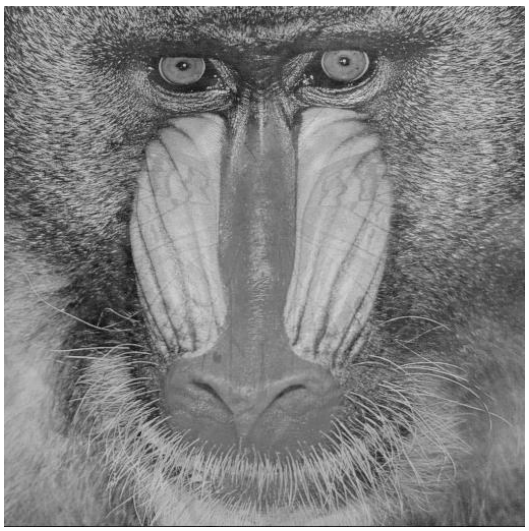

fig: 1.5 - Combinação de Imagem (0.8 + 0.2)

IV. Saída de Dados

O sistema gera 16 imagens de saída, cada qual nominada com o número e o subgrupo das atividades realizadas. As imagens serão armazenadas no mesmo diretório informado no início da execução do sistema. As imagens geradas são: 1.1Atividade-B.png, 1.1Atividade-C.png, 1.1Atividade-D.png, 1.1Atividade-E.png, 1.1Atividade-F.png, 1.2Brilho fator-1.5.png, 1.2Brilho fator-2.5.png, 1.2Brilho fator-3.5.png, 1.3Plano de Bit 1.png, 1.3Plano de Bit 4.png, 1.3Plano de Bit 7.png, 1.4Mosaico (Nova ordem).png, 1.4Mosaico (Original).png, 1.5Combinação de Imagem(0.2 + 0.8).png, 1.5Combinação de Imagem(0.5 + 0.5).png, 1.5Combinação de Imagem(0.8 + 0.2).png.

Cada uma destas imagens é apresentada na tela durante a execução do programa.

V. Resultados

Os resultados das atividades realizadas ao longo deste trabalho foram satisfatórios e similares aos resultados propostos pelo projeto. Porém, um problema encontrado ao longo do desenvolvimento não foi corrigido, este problema se remete atividade “1.4 do mosaico”, cuja a divisão do tamanho da imagem foi realizada, porém o resto oriundo da divisão não foi adicionado a imagem. Outro problema ocorrido durante o desenvolvimento do projeto, se refere ao comando de salvar a imagem (*cv2.imwrite*) em disco, que para algumas imagens, o resultado do da gravação não foi satisfatório, entretanto este problema foi corrigido.

VI. Bibliografia:

[1] Deilmann, M. A guide to vectorization with intel® c++ compilers. Tech. rep., Intel Corporation, 2012.

[2] Scikit-image: Docs for 0.18.0.dev0. https://scikit-image.org/docs/dev/index.html. Acesso em 02/10/2020.

[3] Numpy: NumPy v1.19 Manual https://numpy.org/doc/stable/. Acesso em 02/10/2020

[4] Python : Documentation 3.8.6. https://docs.python.org/3/library/math.html. Acesso em 02/10/2020