

Trabalho 0

Tiago de Paula Alves (187679)
tiagodepalves@gmail.com

31 de março de 2025

1 Introdução

O objetivo deste trabalho é de realizar processamentos básicos de imagem e se familiarizar com Python e suas ferramentas. Pensando nisso, o programa desenvolvido utiliza a biblioteca OpenCV para leitura, codificação e decodificação de imagens PNG e usa Numpy para tratamento vetorizado da imagem.



Figura 1: Imagem colorida usada como comparação dos resultados dos processamentos.

Todos os processamentos serão aplicados nas mesmas imagens da especificação do trabalho e adicionalmente em uma imagem colorida, apresentada na figura 1.

2 O Programa

2.1 Código Fonte

O programa foi desenvolvido em 4 arquivos de Python separados:

main.py É o corpo do programa, responsável por processar os comandos e opções da linha de comando.

inout.py Funções que tratam da entrada e saída do programa, como leitura e escrita de arquivos de imagem e também da apresentação da imagem em uma janela gráfica.

lib.py Funções de transformação da imagem, como ajuste de brilho e acesso do plano de bits.

tipos.py Definição dos tipos para tipagem estática (apenas o protocolo¹ `Image`).

¹ PEP 544 – *Protocols: Structural subtyping (static duck typing)*. URL: <https://www.python.org/dev/peps/pep-0544/>.

A tipagem estática é aplicada para facilitar o desenvolvimento e a leitura do código, mas ela limita a execução em Python para as versões 3.7 ou superior.²

Todas as figuras base utilizadas neste relatório podem ser encontradas na pasta `imagens` do código fonte (a figura 1, por exemplo, está em `imagens/color.png`). Os resultados de processamento apresentados na seção 3 serão rotulados com o caminho da figura original nessa pasta para facilitar a comparação. Também faz parte do código fonte um arquivo textual `padrao.txt` como parte da operação de mosaico, como explicado na seção 3.10.

2.2 Execução

A execução deve ser feita através do interpretador de Python 3.7+. A única entrada obrigatória é o caminho para a imagem PNG que será processada. As entradas seguintes serão tratadas como comandos de processamento da imagem. Ao final da execução, a imagem resultante será exibida na tela. Por exemplo, o comando abaixo apresenta a figura 2 em uma nova janela gráfica.

```
$ python3.8 main.py imagens/color.png monocromatico plano.bit 6 negativo \
> mosaico padrao.txt
```

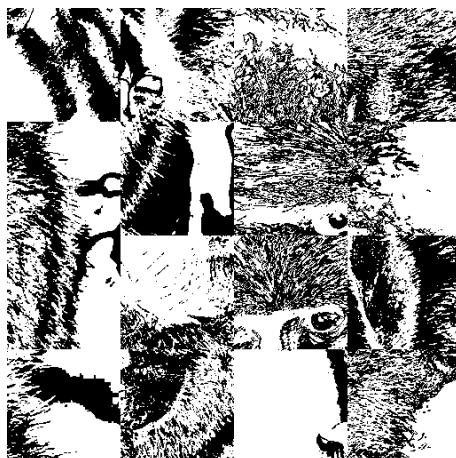


Figura 2: Aplicação de alguns processamentos na figura 1.

Além das entradas posicionais, existem dois argumentos opcionais. Com `--output`, ou `-o`, é possível gravar o resultado em um arquivo PNG em vez de exibir na tela. O comando abaixo resulta na mesma imagem da figura 2, mas guarda em um arquivo `out.png`, sem exibir na tela.

```
$ python3.8 main.py imagens/color.png -o out.png monocromatico plano.bit 6 \
> negativo mosaico padrao.txt
```

Se é desejável tanto a exibição da imagem quanto o salvamento no arquivo, o argumento `--force-show` ou `-f` pode ser usado.

2.3 Comandos Entendidos

O programa entende dez comandos distintos, apresentados abaixo, que podem ser repetidos quantas vezes forem necessárias. Os comandos são executados na ordem em que aparecem na linha de comando.

² PEP 563 – Postponed Evaluation of Annotations. URL: <https://www.python.org/dev/peps/pep-0563/#implementation>.

Todos os comandos devem ser usados da mesma forma como aparecem aqui, em letras minúsculas, sem acentos e com ponto final onde for descrito abaixo. Os nomes em letras maiúsculas são os argumentos daquele comando, que devem aparecerem após o nome do comando (a maioria deles não tem argumentos).

monocromatico	Transforma a imagem em monocromática, com escalas de cinza. Só faz diferença para entradas coloridas.
negativo	Inverte a intensidade de cada píxel da imagem, em todos os canais de cores.
esp.verical	Faz o espelhamento vertical da imagem.
conv.intervalo	Converte o intervalo de intensidades para [100, 200].
inverte.pares	Inverte a direção dos píxeis das linhas pares da imagem.
reflexao	Faz a reflexão das linhas superiores.
aj.brilho GAMA	Aplica um ajuste de brilho com fator GAMA.
plano.bit M	Extrai o plano de bit de ordem M para ordens em [0, 8).
combina IMAGEM ALPHA	Faz a combinação com outra imagem PNG no arquivo IMAGEM por média ponderada com um fator de peso ALPHA.
mosaico ORDENACAO	Contrói um mosaico da imagem a partir de nova ordem dos blocos descrita no arquivo ORDENACAO (detalhes na seção 3.10).

3 Implementação

A parte de processamento de imagem se encontra no código fonte `lib.py`. Cada subseção mostra no trecho de código qual função nesse arquivo se encontra a implementação da operação. Apesar do nome ser o mesmo, o corpo da função é diferente, ignorando detalhes como cópias de *buffer* e tipagem estática, mas mantendo as operações principais.

Quando possível, a função equivalente do OpenCV também será apresentada, já que as acelerações de GPU podem ser facilmente acessadas usando `cv2.cuda` em vez de apenas `cv2`.³ Todas as operações também podem ser feitas por uma *lookup table*, com a função `cv2.LUT`⁴ ou a classe `cv2.cuda.LookUpTable`.⁵ Entretanto, as funções deste trabalho foram implementadas apenas com Numpy, visando a familiarização com técnicas de vetorização.

³ OpenCV: CUDA-accelerated Computer Vision. URL: https://docs.opencv.org/4.4.0/d1/d1e/group__cuda.html.

⁴ OpenCV: Operations on arrays - `cv::LUT`. URL: https://docs.opencv.org/4.4.0/d2/de8/group__core__array.html#gab55b8d062b7f5587720ede032d34156f.

⁵ OpenCV: `cv::cuda::LookUpTable` Class Reference. URL: https://docs.opencv.org/4.4.0/df/d29/classcv_1_1cuda_1_1LookUpTable.html.

Assuma que as bibliotecas são importadas como:

```
import numpy as np
import cv2
```

3.1 Conversão para Monocromático

A transformação para escala de cinza é feita através da média dos três canais de cores, truncada para inteiro. A matriz resultante é então repetida novamente para os três canais, para que a operação possa ser repetida sem erros de execução do programa.



(a) `imagens/color.png`



(b) `imagens/city.png`

Figura 3: Imagem em escala de cinza.

```
def grayscale(imagem):
    gray = np.mean(imagem, axis=2).astype(np.uint8)
    return np.stack([gray, gray, gray], axis=2)
```

Código 3.1: Comando monocromático

Em vez de `np.mean(imagem, ...)`, a conversão poderia ser implementado também com `cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)`.⁶

3.2 Negativo da Imagem

Para essa conversão basta fazer `255 - intensidade` para cada pixel. Como 255 também é o maior valor de um *byte*, a operação é equivalente a inverter todos os bits da imagem.

```
def negativo(imagem):
    return ~imagem
```

Código 3.2: Comando negativo

Em vez de usar o *not* (`~`) do ndarray, existe também a função `cv2.bitwise_not(imagem)`.⁷

⁶ OpenCV: Color Space Conversions - `cv::cvtColor`. URL: https://docs.opencv.org/4.4.0/d8/d01/group_imgproc_color_conversions.html#ga397ae87e1288a81d2363b61574eb8cab.

⁷ OpenCV: Operations on arrays - `cv::bitwise_not`. URL: https://docs.opencv.org/4.4.0/d2/de8/group_core_array.html#ga0002cf8b418479f4cb49a75442baee2f.

(a) `imagens/color.png`(b) `imagens/city.png`

Figura 4: Imagem com intensidade invertida.

3.3 Espelhamento Vertical

(a) `imagens/color.png`(b) `imagens/city.png`

Figura 5: Imagem espelhada verticalmente.

```
def espelhamento_vertical(imagem):
    return imagem[::-1]
```

Código 3.3: Comando `esp.vertical`

Pode ser implementado também com `cv2.flip(imagem, 0)`.⁸

3.4 Conversão de Intervalo

Converte o intervalo de intensidade da imagem de [0, 255] para [100, 200] linearmente.

⁸ OpenCV: Operations on arrays - `cv::flip`. URL: https://docs.opencv.org/4.4.0/d2/de8/group__core__array.html#gaca7be533e3dac7feb70fc60635adf441.

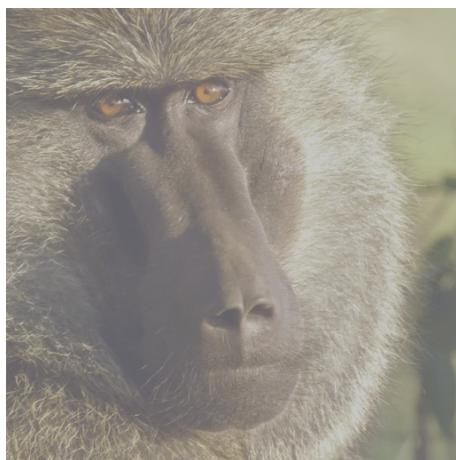
(a) `imagens/color.png`(b) `imagens/city.png`

Figura 6: Intervalo de intensidade convertido.

```
def converter_intervalo(imagem):
    zmin, zmax = np.min(imagem), np.max(imagem)
    img = 100 * (image / 255) + 100
    return img.astype(np.uint8)
```

Código 3.4: Comando `conv.intervalo`

3.5 Inversão das Linhas Pares

Inverte todas as linhas pares horizontalmente.

(a) `imagens/color.png`(b) `imagens/city.png`

Figura 7: Linhas pares invertidas.

```
def inverte_linhas_pares(imagem):
    imagem[::2] = image[::2, ::-1]
    return imagem
```

Código 3.5: Comando `inverte.pares`

3.6 Reflexão das Linhas Superiores

(a) `imagens/color.png`(b) `imagens/city.png`

Figura 8: Linhas superiores refletidas.

```
def reflexao_linhas(imagem):
    mid = len(image) // 2
    imagem[-mid:] = imagem[:mid] [::-1]
    return imagem
```

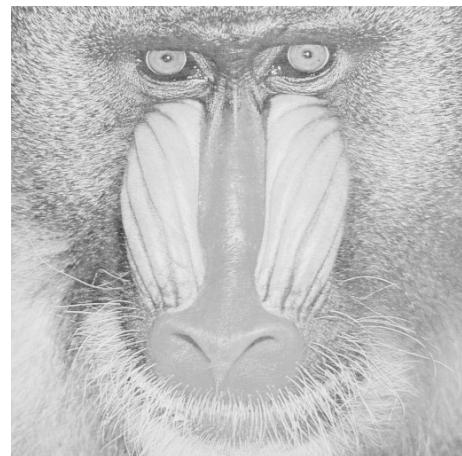
Código 3.6: Comando `reflexao`

3.7 Ajuste de Brilho

Ajuste de brilho por correção gama. A implementação é feita convertendo a imagem para uma matriz A com valores em $[0, 1]$. Com isso, é feita a transformação $B = A^{\frac{1}{\gamma}}$, sendo B convertida para uma imagem com valores em $[0, 255]$ novamente.

O fator γ é passado como argumento para o comando `aj.brilho`. Por exemplo, a figura 9a foi feita com o seguinte comando:

```
$ python main.py imagens/color.png -o out.png aj.brilho 2.5
```

(a) `imagens/color.png`(b) `imagens/baboon.png`Figura 9: Brilho ajustado com $\gamma = 2.5$.

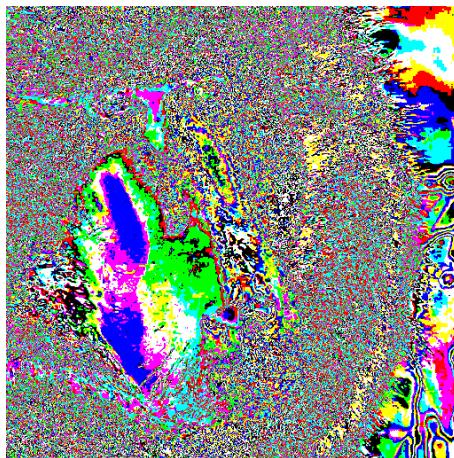
```
def ajuste_brilho(imagem, gama):
    A = imagem / 255
    B = A ** (1 / gama)
    return (B * 255).astype(np.uint8)
```

Código 3.7: Comando aj.brilho GAMA

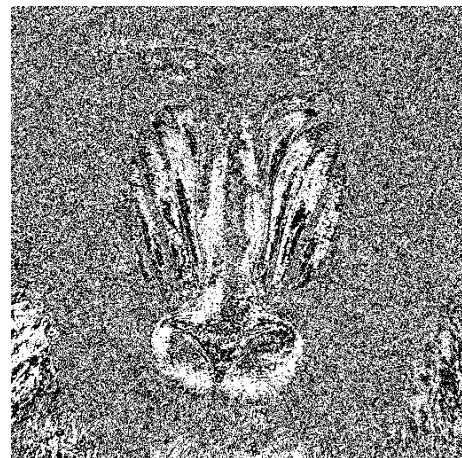
3.8 Planos de Bit

Extração de um plano de bit M da imagem. O argumento M deve aparecer logo após o comando, como em:

```
$ python main.py imagens/baboon.png plano.bit 4
```



(a) imagens/color.png



(b) imagens/babooon.png

Figura 10: Plano de bit 4.

```
def plano_de_bit(imagem, bit):
    plano = (imagem >> bit) & 1
    return plano * 255
```

Código 3.8: Comando plano.bit M

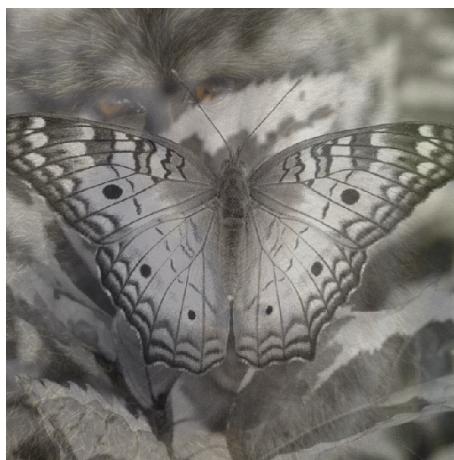
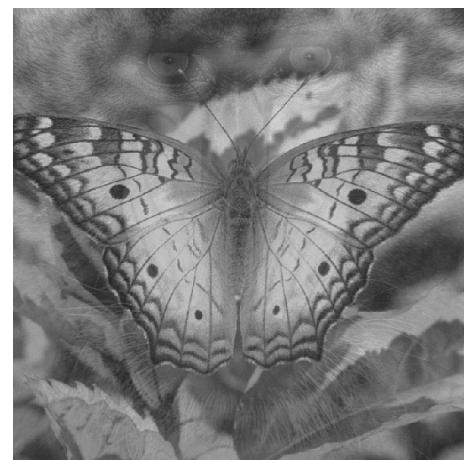
3.9 Combinação de Imagens

Combinação da imagem atual A com uma segunda imagem B , com peso α , resultando em $\alpha A + (1 - \alpha)B$. Tanto a imagem B quanto o peso α devem ser passados como argumentos, nessa ordem.

```
$ python main.py imagens/baboon.png combina imagens/butterfly.png 0.2
```

```
def combinacao(A, B, alpha):
    img = A * alpha + B * (1 - alpha)
    return img.astype(np.uint8)
```

Código 3.9: Comando combina IMAGEM ALPHA

(a) `imagens/color.png`(b) `imagens/babooon.png`Figura 11: Imagem combinada com `imagens/butterfly.png`, com $\alpha = 0.2$.

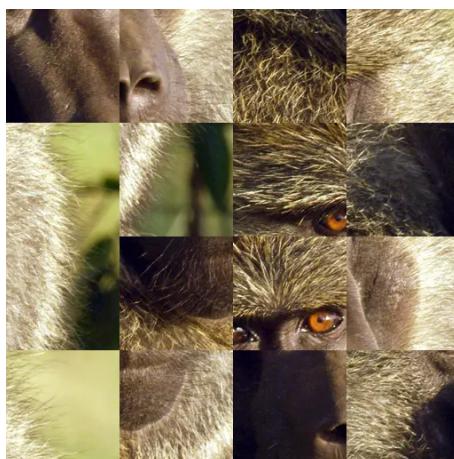
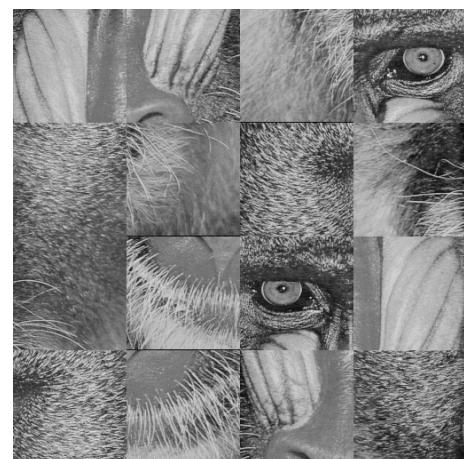
Usando a biblioteca OpenCV, essa função poderia ser implementada com o `addWeighted`,⁹ como mostrada abaixo.

```
def combinacao(A, B, alpha):
    return cv2.addWeighted(A, alpha, B, 1-alpha, 0)
# ou na arquitetura CUDA
return cv2.cuda.addWeighted(A, alpha, B, 1-alpha, 0)
```

3.10 Mosaico

Transformação em mosaico da imagem por uma reordenação dos seus blocos. O argumento `ORDENACAO` deve conter uma caminho para o arquivo com essa nova ordenação e deve aparecer na sequência após o comando `mosaico`, como no invocação a seguir.

```
$ python main.py imagens/baboon.png mosaico padrao.txt
```

(a) `imagens/color.png`(b) `imagens/babooon.png`Figura 12: Mosaico da imagem pelo `padrao.txt`.

⁹ OpenCV: Operations on arrays - `cv::addWeighted`. URL: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#gafafb2513349db3bcff51f54ee5592a19.

```

def mosaico(imagem, ordem):
    N, M = ordem.shape
    # split e flatten
    bloco = [
        bloco
        for lin in np.vsplit(imagem, N)
        for bloco in np.hsplit(lin, M)
    ]
    # acesso (em lista) e concatena
    imagem = np.concatenate([
        np.concatenate([bloco[i-1] for i in lin], axis=1)
        for lin in ordem
    ])
    return imagem

```

Código 3.10: Comando mosaico ORDENACAO

O arquivo de reordenação deve ser composto de inteiros únicos entre 1 e $N \times M$, dispostos em N linhas e M colunas, e separados apenas por espaços (para a leitura com `np.loadtxt`¹⁰). As dimensões de tamanho da imagem também devem ser divisíveis por N e M (altura por N e largura pro M) para que os blocos possam ser reposicionados. No código fonte existe um arquivo `padrao.txt` com a reordenação do enunciado para imagens com dimensões múltiplas de 4.

```

6 11 13 3
8 16 1 9
12 14 2 7
4 15 10 5

```

Código 3.11: Arquivo padrao.txt com a reordenação proposta no enunciado.

Como a ordem do mosaico da imagem depende da entrada, o código não foi vetorizado, como pode ser visto no trecho 3.10.

4 Resultados

Apesar de não ter nenhuma métrica de comparação das imagens, a análise visual permite dizer que o resultado foi satisfatório, isto é, as operações com as imagens obtiveram resultados próximos com o apresentado na especificação do trabalho. Em especial, as figuras 3b a 12b ficaram muito similares às apresentadas no enunciado, com exceção talvez da figura 6b.

Além disso, as funções puderam ser vetorizadas facilmente, no geral. A única exceção foi a operação de mosaico (seção 3.10), que foi implementado com alguns laços explícitos sobre os blocos da imagem.

¹⁰ `numpy.loadtxt` — NumPy v1.19 Manual. URL: <https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html> (acesso em 03/10/2020).