Trabalho 01 de Introdução ao Processamento de Imagem Digital

MARIANNA DE PINHO SEVERO

RA: 264960

I. INTRODUÇÃO

O processamento de imagens é uma importante atividade desempenhada dentro da área de Visão Computacional, sendo responsável pela captura, representação e transformação de imagens. Ele permite a extração de informações e a transformação de certas características de imagens, de maneira a facilitar tanto a percepção humana como a interpretação realizada por computadores [1].

Com respeito à transformação de imagens, diversas técnicas podem ser utilizadas, de acordo tanto com as necessidades do domínio de aplicação em que as imagens serão empregadas, como com os objetivos das transformações.

Neste trabalho realizamos o processamento de imagens sob três diferentes perspectivas: a alteração da resolução de imagens, do número de níveis de cinza utilizados para representá-las (quantização) e de sua escala de cinza. Para esta última, cinco transformações foram empregadas: a logarítmica, a exponencial, a quadrada, a raiz quadrada e o alargamento de contraste (transformação linear por partes).

Na Seção 2 são apresentadas as bibliotecas utilizadas neste trabalho; na Seção 3 são descritas a saída e entrada de dados; na Seção 4 apresenta-se as técnicas empregadas para a resolução de cada questão; na Seção 5 são apresentados os resultados e discussões; por fim, na Seção 6, apresenta-se a conclusão e as principais dificuldades encontradas ao longo do desenvolvimento do trabalho.

II. DEPENDÊNCIAS

Para a implementação dos algoritmos utilizados neste trabalho, as seguintes bibliotecas foram empregadas:

- numpy: usada para a manipulação dos *arrays* que representam as imagens.
- opencv: utilizada para a leitura e escrita de imagens e para algumas das operações de transformação.
- skimage: também usada para algumas das transformações das imagens.
- matplotlib: empregada para a apresentação das imagens.

III. ENTRADA E SAÍDA DE DADOS

As imagens utilizadas para os testes neste trabalho foram retiradas do Site da Disciplina e estão todas no formato PNG (*Portable Network Graphics*). Elas foram armazenadas em um diretório chamado **input_images**.

Para utilizá-las, em cada questão elas foram carregadas utilizando o trecho de código 1, em que o primeiro parâmetro indica o caminho da imagem e o segundo informa o formato para o qual queremos carregá-la, neste caso, 0 indica que carregaremos uma imagem no formato monocromático. A função empregada retorna um *array* numpy em que cada elemento representa a intensidade de um pixel da imagem.

```
image = cv.imread('input_images/name_image.png',0)
```

Código 1: Carregar uma imagem com OpenCV.

Por sua vez, as imagens geradas - que também são representadas por *arrays* numpy no programa - são armazenadas em um diretório chamado **output_images**, todas no formato PNG. Para salvar as imagens, utilizamos o trecho de código 2, em que o primeiro argumento é o caminho onde salvaremos a imagem e o segundo é a estrutura de dados que a representa.

```
cv.imwrite('output_images/name_image.png', image)
```

Código 2: Salvar uma imagem com OpenCV.

Os algoritmos implementados neste trabalho foram escritos utilizando-se a plataforma Jupyter Notebook. Dessa forma, o arquivo onde os algoritmos foram escritos possui a extensão .ipynb.

IV. QUESTÕES E SOLUÇÕES

Neste trabalho foram propostas três questões, cujos métodos empregados para sua resolução são detalhados nos próximos parágrafos.

A. Resolução de Imagens

A resolução de uma imagem está relacionada à densidade de pixels que ela possui, ou seja, à quantidade de pixels por unidade de área. É importante, entretanto, que tomemos cuidado ao associar a ideia de quantidade de pixels com a de resolução. Uma imagem pode ter mais pixels que outra, mas não podemos afirmar que ela possui resolução maior, a não ser que ambas as imagens possuam a mesma área (largura x altura).

Na questão, foi pedido que reduzíssemos pela metade, sucessivamente, a resolução de uma imagem. Dessa maneira, dada uma imagem com 512 x 512 pixels, deveríamos reduzir sua resolução para 256 x 256 pixels, para 128 x 128 pixels, para 64 x 64 pixels, e assim por diante, até 1 x 1 pixels.

Todavia, um problema surgiu durante a solução dessa questão. De acordo com a definição de resolução, era desejado que a quantidade de pixels fosse alterada, mas as dimensões da imagem permanecessem as mesmas. Porém, no programa, uma imagem é representada por um *array* em que cada elemento representa um pixel e, ao diminuirmos as dimensões desse

array, também alteramos as dimensões da imagem gerada, o que não desejamos. Ou seja, desejávamos alterar o tamanho dos pixels, mas não o tamanho da imagem.

Como não encontramos um método que fizesse diretamente o que desejávamos - mudar a resolução, sem alterar o tamanho da imagem - adotamos outra abordagem:

- Foi criado um vetor de elementos indo de 2 até o valor da maior dimensão do *array* que representa a imagem, em que cada elemento é uma potência de 2. Por exemplo, para uma imagem de dimensões 512 x 512 pixels, o vetor gerado possui os elementos [2, 4, 8, 16, 32, 64, 128, 256, 512].
- Então, dividimos as dimensões do *array* que representa a imagem por cada um dos valores do vetor descrito, gerando novos *arrays* com dimensões menores. Para isso, utilizamos o método **resize** da biblioteca opencv. Assim, como cada elemento do *array* representa um pixel da imagem, ao dividir suas dimensões, diminuimos a quantidade de pixels, de acordo com as resoluções desejadas. É importante destacar que o método resize nos permite determinar o método de interpolação a ser usado para o cálculo dos valores dos novos pixels. O método escolhido foi o **INTER_AREA**, uma vez que ele calcula os novos valores por um processo de reamostragem que relaciona *pixels* a áreas.
- Por fim, utilizamos a função **resize**, novamente, para redimensionar os *arrays* gerados, de modo que todos tivessem as mesmas dimensões da imagem original. Para a interpolação foi usada novamente a **INTER_AREA**.

Para tratar imagens retangulares, ou seja, que não possuem a mesma quantidade de *pixels* por altura e largura, as dimensões passadas para o método *resize* durante o processo de geração de imagens com dimensões menores foram calculadas como o valor máximo entre 1 e a nova dimensão.

Assim, como o vetor descrito anteriormente depende da maior dimensão da imagem, se esta for quadrada, ambas as dimensões serão reduzidas uniformemente. Entretanto, se a imagem for retangular, continuaremos a reduzir a maior dimensão, mesmo quando a menor dimensão tiver chegado ao menor tamanho possível, que é 1. No trecho de código 3 é possível observar esta operação.

```
new_image = cv.resize(image,(max(1,image.shape[1]//
    factor), max(1,image.shape[0]//factor)),
    interpolation = cv.INTER_AREA)
```

Código 3: Diminuir resolução de uma imagem.

B. Quantização de imagens

Assim como a resolução está associada à frequência de amostragem de uma imagem, a quantidade de níveis de cinza (ou cores) que ela pode possuir está relacionada à quantização. Esta determina o número de intensidades diferentes que cada *pixel* de uma imagem pode possuir e depende da profundidade de uma imagem. A profundidade, por sua vez, indica o número de *bits* que cada *pixel* possui [1].

Dessa maneira, seja p a profundidade de uma imagem e L a quantidade de níveis de cinza disponível, L pode ser calculada a partir da Equação 1. Por exemplo, se $p = 8$, então $L = 256$, ou seja, a imagem possui 256 níveis de cinza e um *pixel* pode assumir valores entre [0, 255], que é sua escala de cinza.

$$L = 2^p \tag{1}$$

Na questão foi pedido que representássemos uma imagem com diferentes níveis de quantização. Para isso, assumimos que a imagem utilizada para teste possui 256 níveis de cinza e geramos novas imagens a partir dela, diminuindo o número de níveis de cinza pela metade. Assim, foram geradas imagens com 256, 128, 64, 32, 16, 8, 4 e 2 níveis de cinza. Também foram testados alguns níveis de cinza ímpares, como 3, 5 e 7.

Para alterar o número de níveis de cinza, utilizamos as Equações 2 e 3, retiradas de [2] e [3], respectivamente. L_{max} é o número máximo de níveis de cinza da imagem original, por exemplo, 256; L_i é o número de níveis de cinza que desejamos obter; e fator é o valor pelo qual devemos dividir e multiplicar a imagem original, de maneira a obter a nova quantização.

$$fator = \frac{L_{max}}{L_i} \tag{2}$$

$$NovaImagem = \frac{ImagemOriginal}{fator} * fator \tag{3}$$

Para que a Equação 3 funcione conforme esperado, é preciso que a divisão seja inteira. Assim, os novos valores de cinza gerados serão múltiplos do fator, existindo apenas L_i deles.

Por exemplo, se o número máximo de níveis de cinza da imagem original for 256 e quisermos gerar uma nova imagem com 8 níveis de cinza, teremos o $fator = \frac{256}{8} = 32$. Assim, qualquer valor da imagem original entre 0 e 31 será mapeado para 0, os valores entre 32 e 63 serão convertidos para 32, os entre 64 e 95 serão mapeados para 64 e assim por diante.

Entretanto, essa abordagem possui um problema. Se continuarmos seguindo o exemplo anterior, os valores entre 224 e 255 serão mapeados para 224. Assim, nunca utilizaremos toda a escala de cinza [0, 255].

Para resolver esse problema, utilizamos a função **rescale_intensity** da biblioteca skimage. Ela recebe como parâmetros a imagem e o intervalo para o qual seus valores devem ser mapeados - ou deduz esse intervalo a partir do tipo de dados da imagem, por exemplo, para dados do tipo *uint8* o intervalo será [0,255]- e retorna uma nova imagem com os valores ajustados para esse intervalo, respeitando a quantidade de níveis de cinza da imagem dada como entrada.

Dessa maneira, o algoritmo empregado para resolver essa questão consiste em duas etapas:

- A alteração da quantidade de níveis de cinza da imagem original.
- O ajuste dos valores gerados de maneira que eles utilizem toda a largura da escala de cinza disponível.

C. Escala de Cinza

A transformação da escala de cinza pode ser utilizada para destacar determinadas características de uma imagem [1]. Neste trabalho, cinco transformações foram propostas e a implementação de cada uma delas é detalhada nos próximos parágrafos.

1) Transformação logarítmica: Nesta transformação, o valor de cada pixel é substituído pelo seu logaritmo. Ela geralmente é empregada quando se deseja dar maior realce às regiões mais escuras de uma imagem, ou seja, aumentar sua intensidade. Para calcular o logaritmo, utilizamos a função **log** do numpy.

Dois problemas precisaram ser tratados: quando a intensidade do *pixel* é igual a 0, pois o logaritmo de zero é indefinido; e o fato de os valores gerados pelo logarítmo terem uma escala de cinza menor do que a escala possível de [0,255], fazendo com que as imagens resultantes não tenham o efeito visual esperado.

Para tratar o primeiro problema, adicionamos 1 ao valor de cada *pixel* da imagem, assim os *pixels* de valor zero passarão a ter valor 1 e poderemos calcular o logaritmo. Já, para tratar o problema da escala de cinza, a imagem gerada pelo cálculo do logaritmo é normalizada, dividindo-se o valor de cada *pixel* pelo maior valor presente na imagem, de maneira que os valores resultantes fiquem no intervalo [0,1]. Então, a imagem é multiplicada por 255. Assim, a imagem final gerada terá seus *pixels* com valores utilizando toda a largura do intervalo [0,255]. Essas operações podem ser observadas nas Equações 4 e 5.

$$LogImagem = log(Imagem + 1) \tag{4}$$

$$NovaImagem = \frac{LogImagem}{max(LogImagem)} * 255 \tag{5}$$

2) Transformação exponencial: A transformação exponencial, por sua vez, realça as regiões mais claras de uma imagem. Para calculá-la, utilizamos a função **exp** do numpy.

Nesta transformação, três problemas surgiram: o primeiro está relacionado à "explosão"de valores que pode acontecer, ou seja, a exponencial pode gerar valores muito grandes; já o segundo problema é que a exponencial de zero é um, o que faria com que os valores gerados pela exponencial nunca fossem iguais a zero; por fim, o último problema diz respeito à utilização de toda a escala de cinza, conforme aconteceu na transformação anterior.

Para resolver o primeiro problema, também utilizamos um processo de normalização, mas da imagem original. Assim, dividimos todos os valores da imagem original pelo maior valor encontrado nela, fazendo com que a intensidade de cada *pixel* ficasse no intervalo [0,1], de maneira que a exponencial não gerasse valores exageradamente grandes. Já no segundo problema, subtraímos um do resultado da exponencial. Dessa forma, quando realizarmos a exponencial de zero, o resultado final será mapeado para o valor zero, conforme desejado. Por último, o terceiro problema foi resolvido da mesma maneira que na transformação logarítmica: dividimos a imagem resultante da exponencial pelo maior valor encontrado nela - gerando valores no intervalo [0,1] - e multiplicamos o resultado por 255, fazendo com que todos os valores resultantes ficassem no intervalo [0,255]. As Equações 6, 7 e 8 mostram a resolução desses três problemas, respectivamente.

$$NormImagem = \frac{Imagem}{max(Imagem)} \tag{6}$$

$$ExpImagem = exp(NormImagem) - 1 \tag{7}$$

$$NovaImagem = \frac{ExpImagem}{max(ExpImagem)} * 255 \tag{8}$$

3) Transformação quadrado: A transformação quandrado é utilizada para realçar regiões da imagem que possuem média ou alta intensidade. Para calculá-la, utilizamos a função **power** da numpy.

Assim como aconteceu nas outras transformações, os valores de saída precisaram ser ajustados para o intervalo [0,255]. Para fazermos isso, dividimos a imagem resultante da operação quadrado pelo maior valor encontrado nela e multiplicamos por 255. Também foi necessário normalizar as imagens de entrada, assim como fizemos com a transformação exponencial. A operações empregadas para a realização da transformação quadrado podem ser observadas nas Equações 9 e 10 e 11.

$$NormImagem = \frac{Imagem}{max(Imagem)} \tag{9}$$

$$QuadImagem = NormImagem^2 \tag{10}$$

$$NovaImagem = \frac{QuadImagem}{max(QuadImagem)} * 255 \tag{11}$$

4) Transformação raiz quadrada: A transformação raiz quadrada, semelhantemente à logarítmica, realça as regiões de baixa e média intensidades em uma imagem. Para calculá-la, utilizamos a função **sqrt** da numpy.

Assim como nas outras questões, ajustamos a imagem de saída para o intervalo de valores [0,255], dividindo os valores resultantes da raiz quadrada pelo maior valor encontrado e multiplicando por 255. Nas Equações 12 e 13, podemos observar as operações realizadas.

$$SqrtImagem = \sqrt{Imagem} \tag{12}$$

$$NovaImagem = \frac{SqrtImagem}{max(SqrtImagem)} * 255 \tag{13}$$

5) Transformação linear por partes: Neste tipo de transformação, podemos aplicar diferentes operações lineares sobre intervalos específicos da escala de cinza de uma imagem, também de maneira a realçar ou melhorar determinadas características. Para sua implementação, foram seguidas as operações especificadas no trabalho e representadas pela Equação 14, em que α, β e γ são parâmetros que controlam o contraste da imagem, ao passo que a e b determinam as regiões da escala de cinza sobre as quais queremos aplicar as transformações.

$$g = \begin{cases} \alpha f & \text{se } 0 \leq f \leq a \\ \beta(f-a) + \alpha a & \text{se } a < f \leq b \\ \gamma(f-b) + \beta(b-a) + \alpha a & \text{se } b < f \leq L \end{cases} \quad (14)$$

Uma vez que as operações foram aplicadas, mais uma vez ajustamos os valores da imagem gerada, de maneira a colocá-los no intervalo [0,255]. Para isso, dividimos a nova imagem pelo maior valor entre seus *pixels* e multiplicamos por 255.

V. Resultados e discussões

Nos próximos parágrafos, apresentamos os testes realizados e resultados obtidos para cada uma das questões propostas no trabalho.

A. Resolução de Imagens

Conforme explicado na Seção IV-A, tratou-se tanto imagens quadradas como retangulares. Três testes foram realizados: um com uma imagem de 512 x 512 *pixels*, apresentada na Figura 1; outro com uma imagem de 512 x 275 *pixels*, representada pela Figura 2; e o último, com uma imagem de 241 x 387 *pixels*, como mostra a Figura 3.

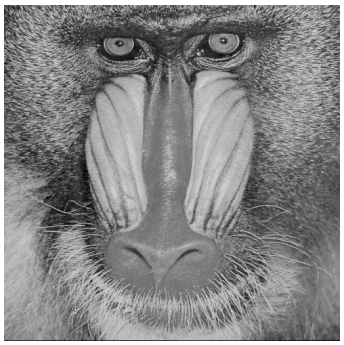


Figura 1: Imagem original de 512 x 512 *pixels*.

Na Figura 4 podemos observar a imagem da Figura 1 com diferentes valores de resolução, menores do que a da imagem original. Para os valores de resolução mais baixos, podemos perceber que a quantidade de *pixels* por largura e altura é a mesma, conforme esperado de uma imagem quadrada. Também podemos observar que conforme a resolução diminui, também se torna menor a quantidade de informações que podemos capturar da imagem, pois o que ela representa vai se tornando menos reconhecível.

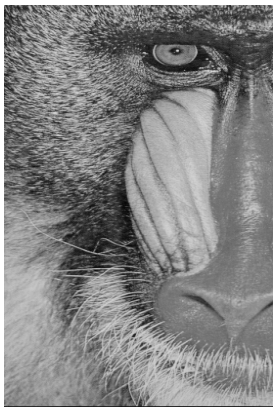


Figura 2: Imagem original com 512 x 275 *pixels*.

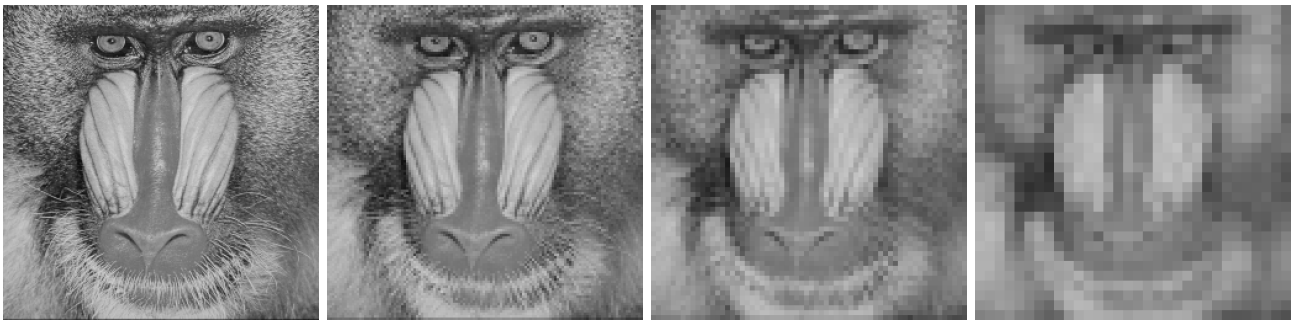
Figura 3: Imagem original com 241 x 387 *pixels*.

Já nas Figuras 5 e 6, podemos observar as imagens das Figuras 2 e 3 também com diferentes resoluções, respectivamente. Como as imagens originais são retangulares, não obtemos a mesma quantidade de *pixels* por largura e altura para nenhuma das novas imagens. Porém, é possível observar que as novas resoluções respeitam as quantidades de *pixels* esperadas pela divisão inteira por potências de 2.

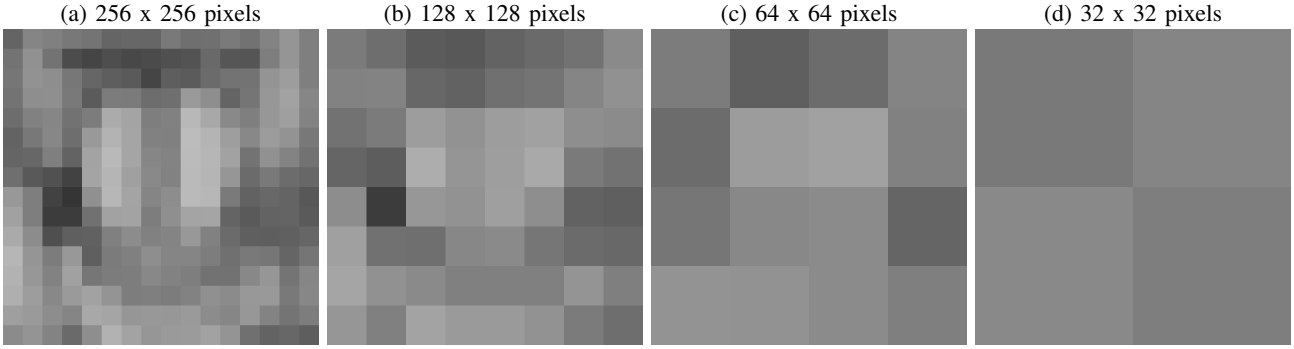
B. Quantização de imagens

Os testes de quantização de imagens foram realizados com uma imagem monocromática com escala de cinza [0,255]. A imagem utilizada como entrada está representada na Figura 1. Já na Figura 7, podemos observar as imagens geradas para diferentes quantizações, conforme pedido pela questão.

É interessante observar como, quando alteramos a quantidade de valores de níveis de cinza de uma imagem monocromática, determinadas características podem se tornar mais ou menos evidentes. Por exemplo, o brilho de determinadas regiões da imagem. Outro detalhe interessante é que, mesmo diminuindo a quantidade de níveis de cinza da imagem original, a informação que ela carrega continuou perceptível - por exemplo, que ela retrata um babuíno. Apesar de que acredito que podem ter outras informações importantes que podem ter sido perdidas. Todavia, uma vez que recursos computacionais são importantes quando tratamos de processamento de imagens digitais, conseguir representar as informações de uma imagem com uma menor quantidade *bits* (menos níveis de cinza), por exemplo, pode fazer grande diferença em todo o processo.

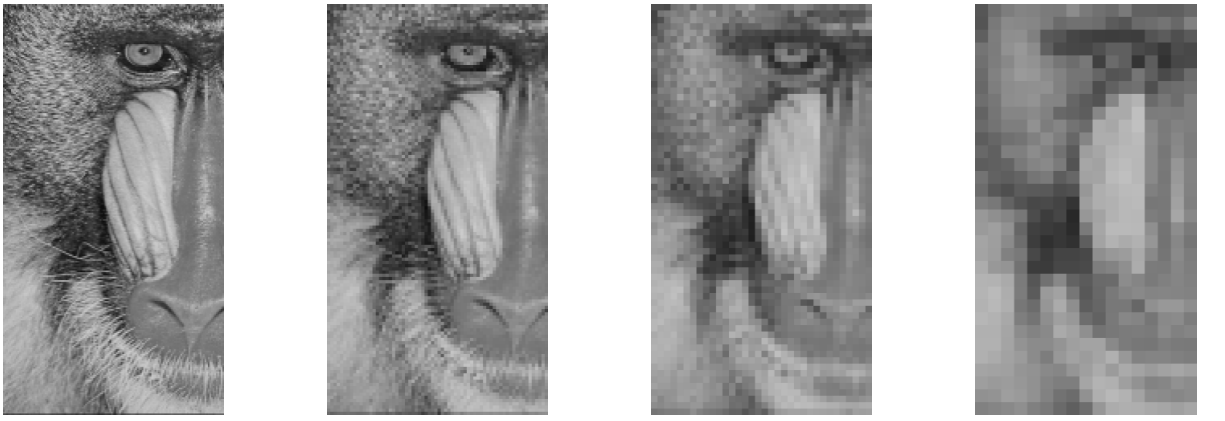


(a) 256 x 256 pixels (b) 128 x 128 pixels (c) 64 x 64 pixels (d) 32 x 32 pixels

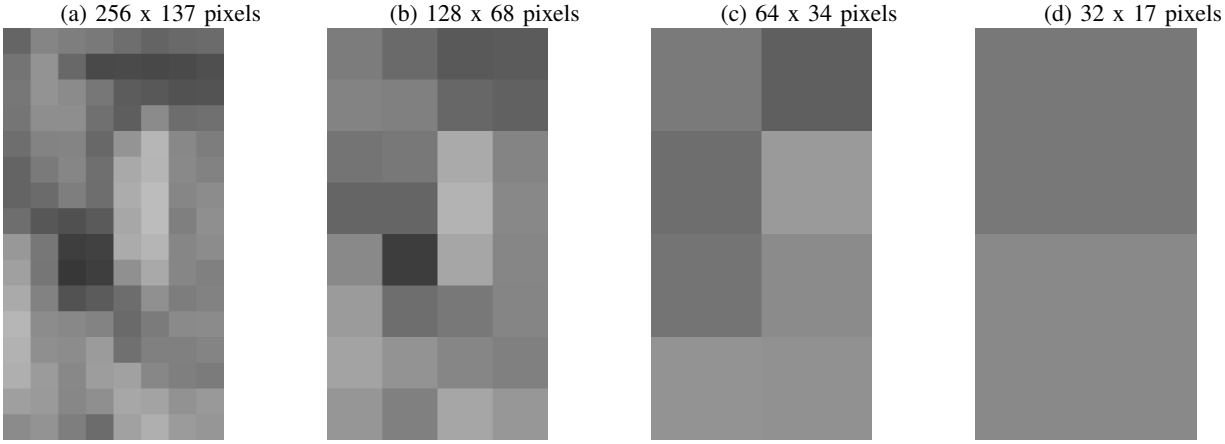


(e) 16 x 16 pixels (f) 8 x 8 pixels (g) 4 x 4 pixels (h) 2 x 2 pixels

Figura 4: Imagens com diferentes resoluções - Teste 1.

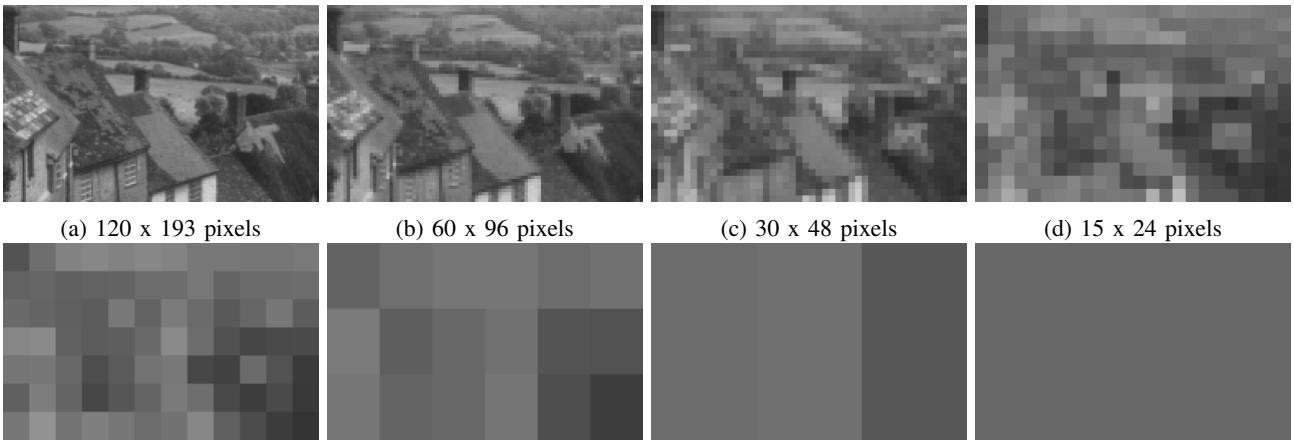


(a) 256 x 137 pixels (b) 128 x 68 pixels (c) 64 x 34 pixels (d) 32 x 17 pixels



(e) 16 x 8 pixels (f) 8 x 4 pixels (g) 4 x 2 pixels (h) 2 x 1 pixels

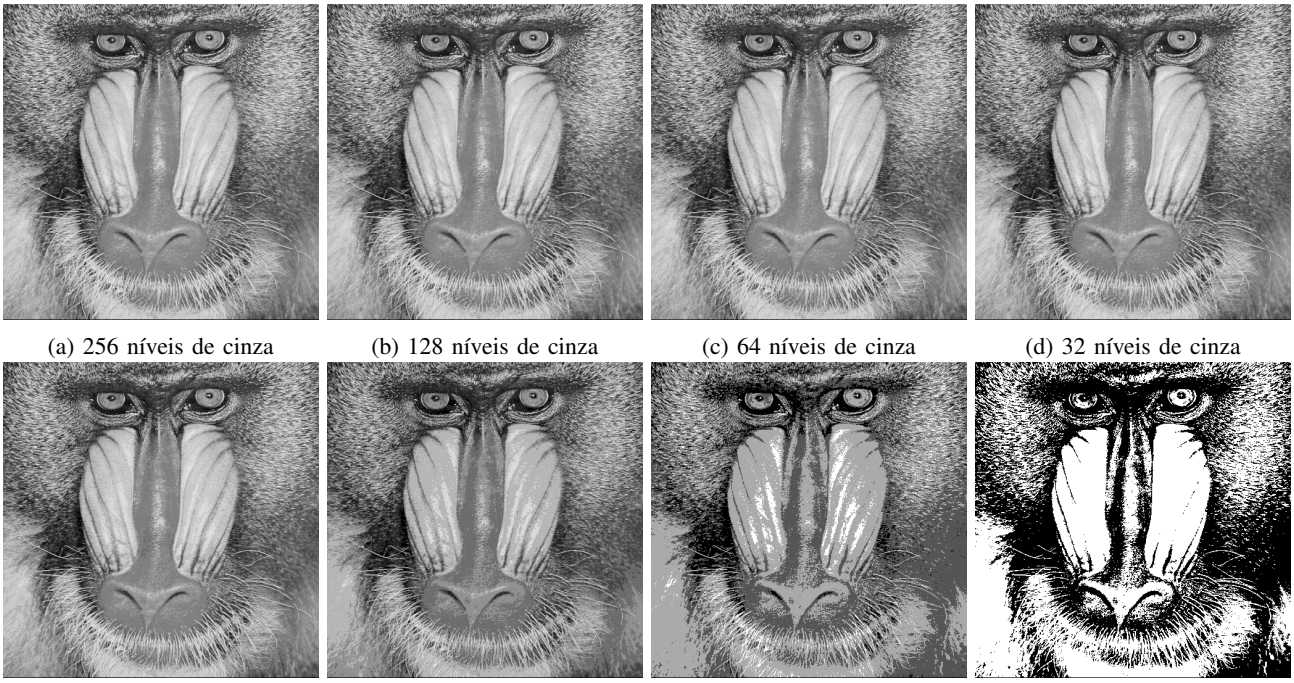
Figura 5: Imagens com diferentes resoluções - Teste 2.



(a) 120 x 193 pixels (b) 60 x 96 pixels (c) 30 x 48 pixels (d) 15 x 24 pixels

(e) 7 x 12 pixels (f) 3 x 6 pixels (g) 1 x 3 pixels (h) 1 x 1 pixels

Figura 6: Imagens com diferentes resoluções - Teste 3.



(a) 256 níveis de cinza (b) 128 níveis de cinza (c) 64 níveis de cinza (d) 32 níveis de cinza

(e) 16 níveis de cinza (f) 8 níveis de cinza (g) 4 níveis de cinza (h) 2 níveis de cinza

Figura 7: Imagens com diferentes níveis de cinza.

C. Escala de Cinza

Por sua vez, as imagens geradas a partir das cinco transformações de escala de cinza pedidas na Questão 1.3 do trabalho podem ser observadas na Figura 9. A imagem utilizada como entrada é mostrada na Figura 8.

É importante destacar os parâmetros utilizados em cada transformação: na logarítmica, na exponencial, na quadrada e na raiz quadrada foram passados como argumentos a imagem a ser transformada e o parâmetro **c** foi calculado de tal forma a ajustar as intensidades das imagens de saída no intervalo [0,255]; já na transformação linear por partes, foi passada a imagem a ser alterada e seus outros parâmetros receberam os valores $\alpha = 0.3$, $\beta = 1$, $\gamma = 2$, $a = 45$, $b = 190$ e $L = 255$. Todas as imagens retornadas pelas transformações foram convertidas para o tipo **uint8**, utilizando-se o método **astype** da numpy.

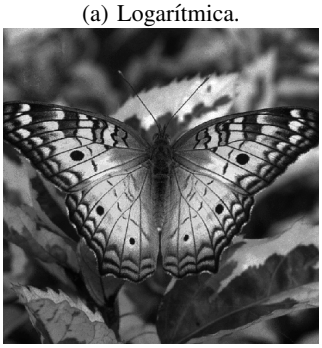
Conforme pode ser observado na Figura 9a, a operação de logaritmo aumenta a intensidade de todos os pixels da imagem, tornando-a mais clara e destancando, principalmente, as regiões que anteriormente eram mais escuras. Isso acontece porque as regiões mais escuras sofrem um aumento bem maior de intensidade do que as regiões mais claras, de acordo com a curva da função logarítmica.

Por sua vez, a transformação de raiz quadrada também torna a imagem mais clara, conforme pode ser visto na Figura 9d.

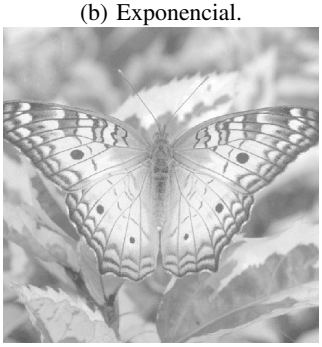
Figura 8: Imagem original a ser transformada.

(a) Logarítmica.

(b) Exponencial.



(c) Quadrado.



(d) Raiz quadrada.

(e) Linear por partes.

Figura 9: Imagens com diferentes transformações.

Entretanto, ela altera menos a intensidade dos *pixels* do que a transformação logarítmica, o que pode ser explicado pelos gráficos de suas funções.

Já as transformações exponencial e quadrada escurecem mais a imagem, sendo a quadrada a que mais escurece, conforme pode ser observado nas Figuras 9b e 9c. Esse escurecimento faz com que as regiões de maior intensidade - mais claras - na imagem original ganhem ainda mais destaque.

Por fim, a transformação linear por partes pode causar diferentes efeitos na imagem original, dependendo dos intervalos e coeficientes usados como parâmetros. Na imagem representada na Figura 9e, o objetivo foi escurecer mais as regiões mais escuras (intervalo de [0,45]), manter os valores de regiões médias (de [46,190]) e clarear mais aéras de maior intensidade (intervalo [191,255]). É possível observar, principalmente, que as regiões mais claras ganharam um pouco mais de destaque, o que pode ser melhorado alterando-se os parâmetros.

VI. Conclusão

Conforme pode ser observado em todas as imagens apresentadas, conseguimos fazer o que cada questão do Trabalho 01 pediu. Uma limitação deste trabalho, que precisa ser destacada, é que todas as questões foram respondidas considerando-se imagens monocromáticas com escala de cinza [0,255].

Reduzimos pela metade, sucessivamente, a resolução de uma imagem, de maneira que pudemos observar a relação entre a resolução e os detalhes que podemos perceber. Foi visto que há diversas técnicas que podem ser utilizadas quando desejamos calcular o valor de um novo *pixel*, seja para diminuir ou aumentar o tamanho de uma imagem. Uma das principais dificuldades encontradas foi entender como alterar a resolução, mas manter as dimensões da imagem. Por fim, entendendo melhor o funcionamento do método **resize** da biblioteca opencv, foi possível realizar essa operação.

Com relação aos níveis de cinza, ou quantização de uma imagem, foi observado que não necessitamos utilizar todos os níveis disponíveis para que possamos reconhecer as informações da imagem. Dependendo da aplicação e das limitações encontradas, imagens podem ser processadas de maneira a se diminuir ou aumentar a quantidade de dados que necessitam sem prejudicar as informações que carregam. Nas resoluções apresentadas, todas as imagens foram carregadas como monocromáticas e com tipo de dados **uint8**, o que as permitia ter até 256 níveis de cinza. Uma dúvida que surgiu é quantos níveis de cinza possui uma imagem com tipo de dados **float**, em que a escala de cinza é [0,1].

Por fim, as transformações das escalas de cinza de imagens foi uma das questões que gerou mais dúvidas, principalmente devido à explosão dos valores para o tipo de dados adotado. Quando utilizando a transformação exponencial, por exemplo, foi necessário normalizar a imagem de entrada antes de realizar a transformação. Isso gerou a dúvida se deveríamos normalizar os valores de acordo com o maior entre eles ou utilizando 255, uma vez que a escala de cinza é de [0,255]. Além disso, quão diferente é a exponencial de uma imagem normalizada daquela em que a imagem não está normalizada.

Referências

[1] H. Pedrini and W. R. Schwartz, *Análise de imagens digitais: princípios, algoritmos e aplicações*. Thomson Learning, 2008. 1, 2, 3

[2] stack overflow. (2017, may) Opencv: Grayscale color reduction. [Online]. Available: https://stackoverflow.com/questions/40161626/opencv-grayscale-color-reduction 2

[3] OpenCV. (2019, dec) How to scan images, lookup tables and time measurement with opencv. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html 2