MC920 - INTRODUÇÃO AO PROCESSAMENTO DIGITAL DE IMAGEM
RELATÓRIO - TRABALHO 1

Andressa Gabrielly Macedo Marçal
RA 262878

1. Especificação do Problema

O propósito do uso das técnicas de pontilhado é reduzir a quantidade de cores (quantização de cores) utilizadas para exibir uma imagem, possibilitando uma boa percepção por parte do usuário. O objetivos deste trabalho consiste em implementar um programa para alterar os níveis de cinza [fmin...fmax] de uma imagem f(x, y) por meio das técnicas de pontilhado (halftoning) ordenado e pontilhado com difusão de erro com algumas abordagens, produzindo uma imagem g(x, y).

1.1. Pontilhado Ordenado

O algoritmo de pontilhado ordenado utiliza um conjunto de padrões formados por pontos pretos e brancos. Os valores das células da matriz podem ser utilizados como limiares. Se o valor do pixel (normalizado entre 0 e 9) for menor do que o número correspondente a célula da matriz, o pixel será substituído pelo valor preto, caso contrario, será substituído pelo valor branco.

1.2. Pontilhado por Difusão de Erro

O algoritmo de pontilhado por difusão de erro também transforma a imagem original em uma imagem contendo apenas os valores preto e branco, entretanto, leva em consideração os valores ao redor de cada pixel. Um exemplo da técnica **aplicada no algoritmo de Floyd-Steinber**, resume-se aos seguintes passos:

- Percorre todos os pixels da imagem, segundo uma ordem pré-definida.
- Para cada pixel, SE seu valor for maior do que 128, troca-se para 255 (pixel branco). SENÃO, troque-o para 0 (pixel preto).
- Armazena-se o erro, ou seja, a diferença entre o valor exato do pixel e o valor aproximado.
- Distribui-se o erro aos pixels adjacentes, da seguinte maneira:
  - Adiciona 7/16 do erro ao pixel localizado à direita.
  - Adiciona 3/16 do erro ao pixel localizado abaixo e à esquerda.
  - Adiciona 5/16 do erro ao pixel localizado abaixo.
  - Adiciona 1/16 do erro ao pixel localizado abaixo e à direita.

A ordem na qual a imagem é percorrida pode produzir resultados diferentes no processo de *dithering*. A varredura da esquerda para a direita pode gerar padrões indesejados ou a impressão de uma certa direcionalidade na imagem resultante. Para evitar esses efeitos, uma alternativa é modificar a direção de varredura a cada linha.

- **Abordagem Zigue Zague:**
  - Na abordagem de *Zigue Zague*, o laço do script percorrerá as linhas no sentido da esquerda para direita nas linhas Pares.
  - O laço percorrerá no sentido da direita para esquerda nas linhas Ímpares.
  - Irá percorrer as colunas da matriz aplicando o *halftone* caso o valor seja superior à intensidade média da profundidade da imagem, será definido como branco, caso contrário, será definido preto.
  - O tamanho da imagem original deverá ser o mesma da final.

Figure 1. Baboon com Floyd e Steinberg em Zigue-Zague

2. Especificações do Script

- A implementação foi realizada no editor de código VSCode, por isso que irão em anexos, separados por nomes de cada função.
- A linguagem utilizada foi Python na versão 3.7.3.
- Utilizei o Jupyter Notebook para processar e gerar um arquivo PDF para auxiliar na analise de uma maneira mais fluida, mas também envio o script .py
- As bibliotecas utilizadas foram:
  - OpenCV >= 3.4.2
  - Numpy >= 1.15.4
  - Pillow >= 6.1.0

2.1. Execução do Script

O programa deve ser executado no terminal (tomando como base o S.O Linux), chamando o interpretador **python3**. Os arquivos estão separados por nome, onde cada função será executada por cada script correspondente.
Ou se preferir, executar as células do script **ipynb**.

2.2. Entrada de Dados

Irá em anexo os diretórios de imagens utilizadas para os testes, onde o diretório **input** será o que irá carregar as imagens para serem utilizadas nos scripts.

Boa parte das amostras seguem com a imagem base do *baboon.png*

2.3. Saída de Dados

As imagens processadas pelos scripts, serão armazenadas no diretório **Output**, que irá em anexo no projeto. Dentro de Output terá mais dois repositórios o **output2**, que segue a abordagem da esquerda para a direita, e o **output_zigzag** que conterá imagens que seguiram a abordagem Zigue Zague.

2.4. Parâmetros Utilizados pelo Script

Todas as imagens utilizadas para execução do programa estão presentes no diretório Input/ e foram disponibilizadas pelo Professor Hélio no seguinte endereço eletrônico (https://www.ic.unicamp.br/ helio/imagens_coloridas/).

Figure 2. Amostra de imagem baboon.png, utilizadas para os testes

3. Resultados

Nesta seção irei apresentar os resultados coletados da execução do script Python para cada abordagem solicitada no trabalho.

3.1. Técnica Floyd e Steinberg

	f(x,y)	7/16
3/16	5/16	1/16

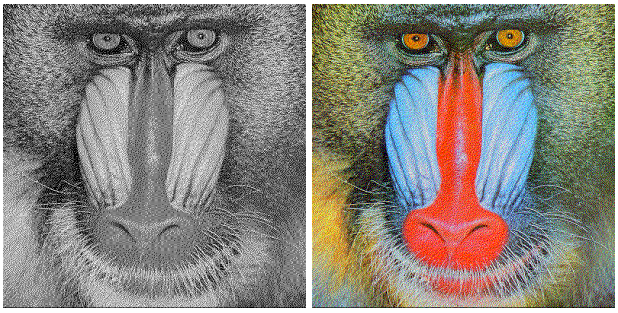

Figure 3. Baboon com Floyd e Steinberg

3.2. Técnica Stevenson e Arce

			f(x,y)		32/200	
12/200		26/200		30/200		16/200
	12/200		26/200		12/200	
5/200		12/200		12/200		5/200

Figure 4. Baboon com Stevenson e Arce

3.3. Técnica Burkes

		f(x,y)	8/32	4/32
2/32	4/32	8/32	4/32	2/32

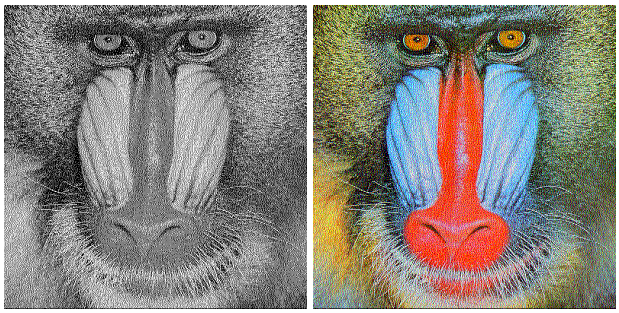

Figure 5. Baboon com Burkes

3.4. Técnica Sierra

		f(x,y)	5/32	3/32
2/32	4/32	5/32	4/32	2/32
	2/32	3/32	2/32	

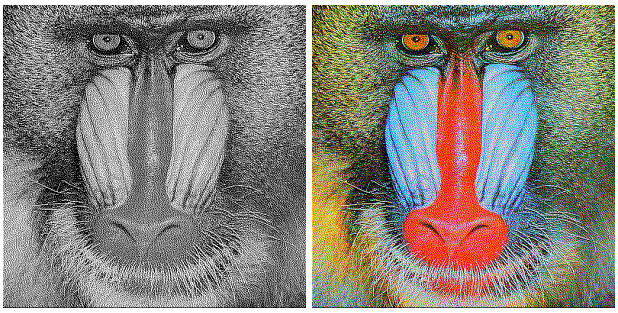

Figure 6. Baboon com Sierra

3.5. Técnica Stucki

		f(x,y)	8/42	4/42
2/42	4/42	8/42	4/42	2/42
1/42	2/42	4/42	2/42	1/42

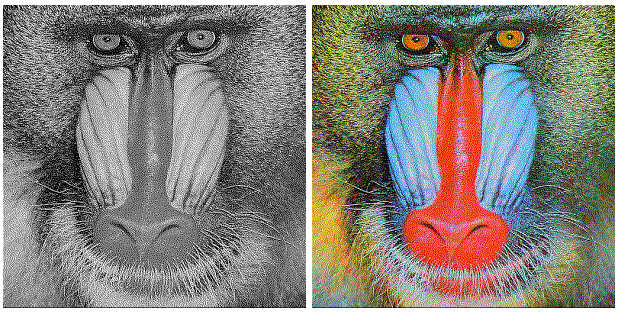

Figure 7. Baboon com Stucki

3.6. Técnica Jarvis, Judice e Ninke

		f(x,y)	7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

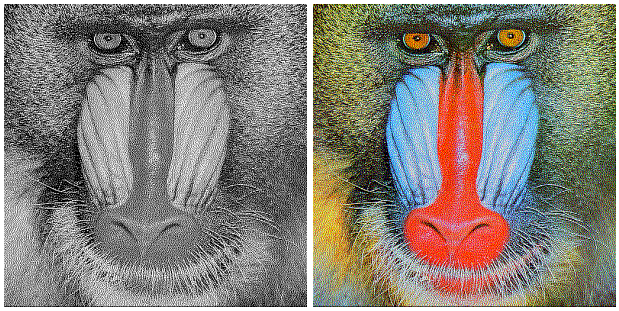

Figure 8. Baboon com Jarvis, Judice e Ninke

3.7. Leitura das Amostras

As imagens de entrada, exemplo delas encontra-se na Figura 1 deste trabalho, será lida pela função **cv2.imread()** que encontra-se na biblioteca OpenCV2, na qual passamos o diretório onde a imagem está armazenada, e então eu declaro duas variáveis para capturar a altura e largura da imagem, o seu shape, onde condiz que shape[na posição 0] que pega a largura, shape[na posição 1] pegará sua altura e shape[na posição 2] pegará os canais de cores.
Para obter o resultado em tons de cinza a imagem é convertida para níveis de cinza através da função **cv2.cvtColor()**, também encontra-se na biblioteca OpenCV2 que recebeu como parâmetros a imagem e **cv2.COLOR_BGR2GRAY** que serve para transformar a imagem RGB em escala de cinza.

3.8. Algoritmo de Pontilhado Ordenado

Foram apresentadas no Trabalho1 algumas abordagens para distribuição de erro em técnicas de pontilhado para serem aplicadas em uma imagem digital no formato PNG(*Portable Network Graphics*).

- As funções utilizadas foram desenvolvidas para aplicar os métodos propostos pelo trabalho1, recebendo como parâmetros de entrada a imagem(*img*) *baboon.png*, e dentro de cada função é implementada constantes com cada valor da matriz correspondente ao filtro.

- Os pixels não são substituídos a partir da comparação com uma matriz-base , mas sim com um determinado limiar, que no caso foi definido 128;

- Ao realizar a chamada da função (cada método terá como função, seu próprio nome, ex: floyd, stevenson, stucki..)

- Para cada pixel substituído na imagem passada como parâmetro, armazena-se o erro (*quant_err*) entre o valor do pixel original (*old_pixel* e o valor do novo pixel/pixel aproximado (*new_pixel*))

- A variável *quant_err* é distribuída aos pixels adjacentes;

- Resume-se que cada pixel é transformado em 255 (branco) se seu valor original for maior que 128, ou em 0 (preto) caso seu valor seja menos que 128. [SE image[x][y] for menor que 128, ele será substituído por 0, SENAO será substituído por 255.]

- O erro é adicionado aos valores originais dos pixels adjacentes conforme os pesos especificados nas matrizes ilustradas em cada método proposto.

3.9. Algoritmo de Pontilhado de Difusão de Erro

As técnicas de pontilhado por difusão de erro resultam em imagens com menos perda de informação que as de pontilhado ordenado. Embora algum grau de perda de informação seja até inevitável.
Ao aplicar a abordagem do laço tradicional(varredura comum) no *peppers.png*, por exemplo, algumas áreas se tornam muito ruidosas, como se houvesse muito ruído estilo ”sal e pimenta”.
Contudo, detalhes menos finos mantêm-se sendo identificáveis, havendo por exemplo, menor distorção de bordas do que na técnica anterior.

Os resultados serão apresentados na **Figura 9**.

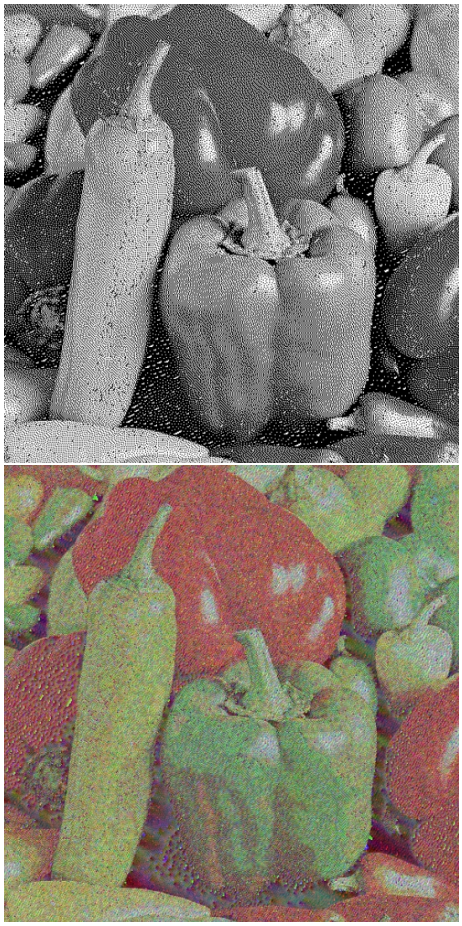

Figure 9. Mais alguns exemplos de imagens processadas com de difusão de erro

4. Dificuldades Encontradas

- No decorrer do desenvolvimento do código, tive alguns problemas por não ter domínio de algumas bibliotecas. Já programava em Python, o que de fato me foi mais ”confortável”, porém senti a necessidade de me aprofundar na biblioteca Numpy, e um pouco mais na scikit, pois penso que talvez minha implementação chegasse a ser mais ”enxuta” utilizando métodos dessas bibliotecas.

- Utilizei o jupyter, além de um script comum que chamei de *full_script* onde fiz por um editor de código com terminal embutido, porém notei diferença na plotagem das imagens, não são iguais em termos de tonalidade

5. Conclusão

Todas as imagens que foram geradas nesse Trabalho1 foram anexas no diretório comprimido com título **trabalho1_ra262878.zip**, e também, foram exibidas na execução das células no Jupyter Notebook, utilizando as funções de exibição da biblioteca *Pillow*. As imagens foram salvas no diretório **Output/**, onde sua atual organização esta:

- **Output:**
  - **output2:** Imagens com abordagem da Esquerda para Direta;
  - **output_zigzag:** Imagens com abordagem Zigue Zague;
  - **pdf:** A execução do Jupyter Notebook e seus comentarios
  - **script:** Todos os algoritmos de todos os filtros do trabalho 1, e suas abordagens da esquerda para direita, e da direita para esquerda na linguagem Python.

Dentro do diretório principal também estará contido este relatório em PDF, a execução do Jupyter Notebook em PDF, o script Python para executar no terminal, caso queira, e as imagens de entrada e saída, como citadas acima.