INSTITUTO FEDERAL
São Paulo

Exercícios #03

Instituto Federal de São Paulo (IFSP)
Campus Caraguatatuba
Tópicos Avançados (TPA A6)

17 de Novembro de 2021

Objetivos A terceira lista de exercícios da disciplina de "Tópicos Avançados" tem como objetivo implementar filtros para remoção de ruídos em imagens digitais. Você irá trabalhar com a geração de ruído sintético (i.e., ruído gerado propositalmente no computador) e a implementação de filtros específicos para remoção e/ou redução de tais ruídos. Todos os algorítmos implementados nessa lista de exercícios utilizam as bibliotecas OpenCV e Numpy.

Exercício 3.01. Neste exercício, você irá gerar um conjunto de histogramas de (i) uma imagem em tons de cinza, (ii) uma imagem com elementos embaralhados, e (iii) imagens de multi-canais (i.e., RGB e HSV). Para responder esse exercício, você deve utilizar o arquivo `Ex301_histogram.py` disponível no material de suporte.

(a) **Histograma em tons de cinza:** Use a função `cv2.calcHist()` para gerar o histograma da imagem em tons de cinza com os seguintes parâmetros:

- `images`: uma lista com as imagens de entrada a serem processadas. Para gerar um histograma de um canal único, você deve utilizar um par de colchetes entre o nome da imagem de entrada (e.g., `[grayscale]`). Para gerar histogramas de n-dimensões, você deve utilizar vírgulas (`,`) para separar cada canal da imagem de entrada em uma lista (e.g. `[image[0], image[1]]`);
- `channels`: uma lista dos canais utilizados para gerar o histograma. Para gerar um histograma de canal único utilize `[0]`, e para gerar histogramas de múltiplo canais utilize `[0, 1]`;
- `mask`: uma máscara *opcional*. A máscara é utilizada para selecionar uma região de interesse na imagem processada. Para esse exercício utilize o valor `None`;
- `histSize`: um *array* do tamanho do histograma em cada canal. Como você irá utilizar imagens em tons de cinza (ou cada canal individualmente), utilize o valor 256 para gerar um histograma entre 0 e 255. Para um histograma de canal único, utilize o código `[256]`, e para um histograma de n-dimensões utilize `[256, 256]`;

- `ranges`: um *array* com a altura máxima da barra do histograma em cada dimensão. Você deve informar o intervalo do histograma. Para um histograma de canal único, utilize o código `[0, 255]`, e para um histograma de n-dimensões utilize `[0, 255, 0, 255, 0, 255]`.

(b) **Histograma da imagem embaralhada:** Utilize a função `numpy.random.shuffle()` para embaralhar os elementos (i.e. *pixels*) das linhas e colunas de uma imagem de entrada. Gere o histograma da imagem embaralhada. Compare os histogramas da imagem em tons de cinza original e da imagem embaralhada.

(c) **Histograma da imagem RGB:** Utilize a imagem de entrada para calcular o histograma de cada canal individualmente. Una os três histogramas de cada canal RGB em uma estrutura Numpy *array* (256×3). Compare os histogramas RGB com o histograma em tons de cinza.

(d) **Histograma da imagem HSV:** Faça o mesmo procedimento do item anterior com uma imagem HSV. Compare o histograma HSV com o histograma RGB.

Nesse exercício, você deve mostar os resultados em duas janelas *Matplotlib* com múltiplas images, a saber: (i) a primeira janela com as imagens processadas, como ilustrado na Figura 1; e (ii) a segunda janela com os respectivos histogramas. Você pode utilizar as funções `showImage()` e `showHistogram()` disponíveis no arquivo `Ex301_histogram.py` para criar ambas as janelas.

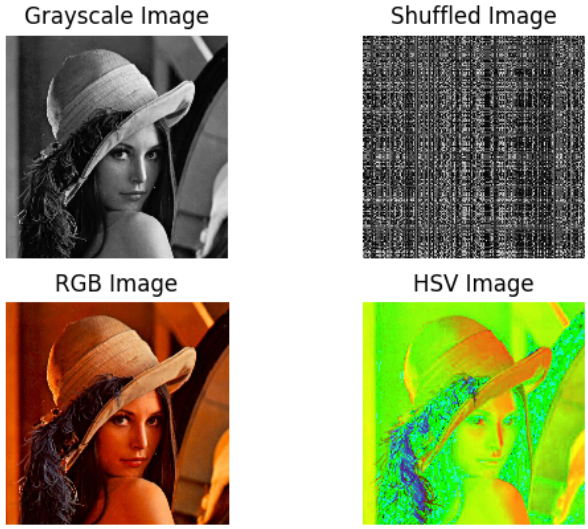

Figure 1: Imagens utilizadas para gerar o histograma apresentado nesse exercício.

Exercício 3.02. Nesse exercício, você irá gerar ruído sintético nas imagens de entrada, bem como, implementar um conjunto de métodos para remover ruído de imagens. Utilize o arquivo `Ex302_filtering_image.py` para implementar o código-fonte desse exercício. A Figura 2 (A) mostra uma imagem em tons de cinza com uma linha branca no qual representa uma linha da imagem selecionada pelo usuário. A Figura 2 (B) ilustra um sinal unidimensional que representa a linha selecionada. Você deve utilizar o sinal unidimensional para avaliar as funções de remoção de ruído.

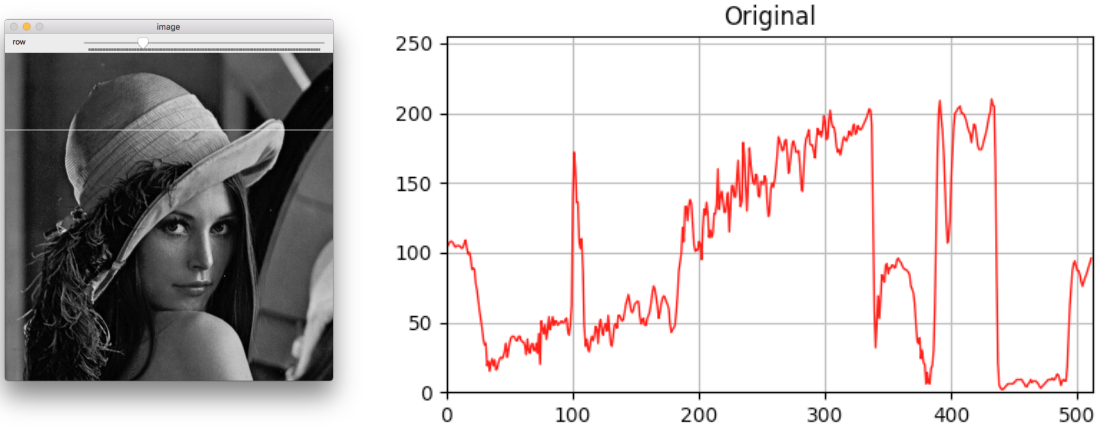

Figure 2: Exemplo de linha selecionada na imagem de entrada e seu respectivo sinal unidimensional.

(a) **Gerando e adicionando ruído:** Nesse exercício, você deve implementar um conjunto de funções para gerar diferente tipos de ruídos e adicionar na imagem de entrada. O ruído baseado em uma distribuição randomica deve ser gerado como uma matrix de tamanho $N \times M$, i.e., com a mesma resolução da imagem de entrada.

- **Ruído de sal-e-pimenta:** Implemente uma funcão `saltAndPepperNoise()` para gerar e adicionar o ruído sal-e-pimenta basedo em um fator de densidade, i.e., um fator entre 0 e 1 que define o quanto de ruído a função irá adicionar na imagem de entrada. Por exemplo:

```
image[((np.random.rand(M, N)) < density)] = [0]
```

  Agora, pense em como gerar o *ruído de sal*!

- **Ruído Gaussiano:** Implemente uma função `gaussianNoise()` para gerar e adicionar um ruído Gaussiano com uma distribuição média μ e desvio padrão σ. Utilize a função `np.random.normal()` para gerar uma distribuição normal. Nesse item, é importante lembrar-se que:
  - A função `np.random.normal()` gera valores em `np.float64`.
  - A função `to_uint8()` converte o *array* para *8-bits* (i.e. `np.uint8`).

  - A função `cv2.add()` adicionar duas matrizes de mesma resolução (e.g., a imagem de entrada e o ruído).
  - A função `scipy.stats.normaltest()` testa se o ruído é baseado em uma distribuição normal.

- **Ruído uniforme:** Faça uma função `uniformNoise()` para gerar e adicionar um ruído uniforme na imagem de entrada. Utilize a função `np.random.uniform()` para gerar amostras de distribuição uniforme.

(b) **Mostra as imagens ruídosas:** O arquivo `Ex302_1D_filter.py` executa as funções que você implementou nos itens anteriores e mostra as imagens com ruídos, como ilustrado na Figura 3.

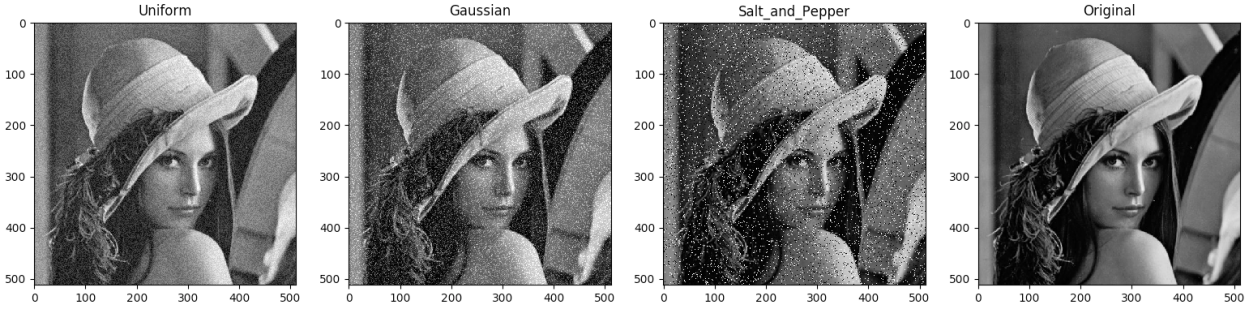

Figure 3: Exemplo de imagens com ruídos sintéticos.

(c) **Filtragem de ruído:** Nesse exercício, você irá implementar um conjunto de funcões para filtar o ruído nas imagens ruídosas.

- **Filtro de sal-e-pimenta:** Implemente a função `saltAndPepperFilter()` para remover o ruído de sal-e-pimenta da imagem de entrada utilizando um filtro de tamanho $N \times N$.
- **Filtro Gaussiano:** Implemente a função `gaussianFilter()` para remover o ruído Gaussiano da imagem de entrada utilizando um filtro de tamanho $N \times N$.
- **Filter uniforme:** Implemente a função `uniformFilter()` para remover o ruído uniforme da imagem de entrada utilizando um filtro de tamanho $N \times N$.

(d) **Mostre as imagens filtradas:** Execute o arquivo `Ex302_1D_filter.py` para ver as imagens filtradas com as funções que você implementou anteriormente, como ilustrado na Figura 4.

(e) **Avaliar os filtros de remoção de ruído:** Você deve avaliar os filtros com imagens ruídosas utilizando filtros de diferentes tamanhos, e.g. $N = 3, 5, 9, 11$. Então, calcule a soma dos quadrados do residual das diferenças entre: (i) a imagem de entrada original e a imagem ruídosa; e (ii) a imagem de entrada original e a imagem filtrada. Você deve avaliar quais os melhores resultados que você alcançou baseado no tipo de ruído na imagem ruídosas, o filtro utilizado e o tamanho do filtro.

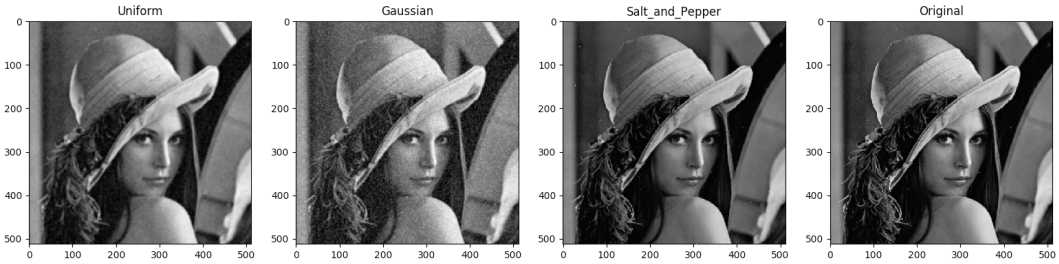

Figure 4: Exemplo de imagens filtradas.

Agora, o *script* irá mostrar uma janela *Matplotlib* que permite você selecionar uma linha na imagem de entrada. O *script* converte a linha selecionada em um sinal unidimensional, como ilustrado na Figura 5.

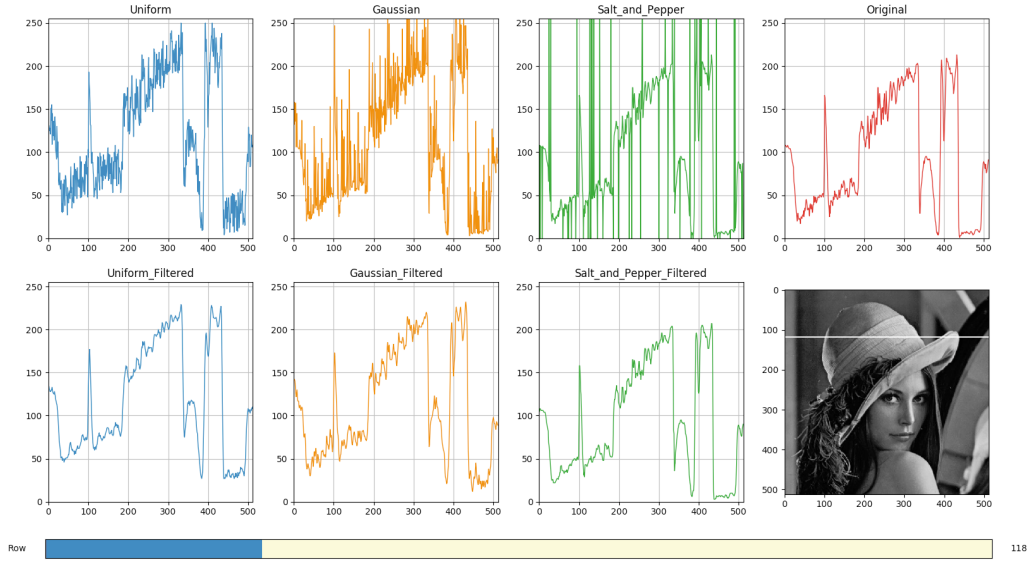

Figure 5: A influência do ruído e da filtragem no sinal unidimensional de uma linha da imagem de entrada.