Image Processing Assignment 1

Smoothing Images using Python and OpenCV

Abhinav Kumar
15BCE1046

Smoothing Images

Goals:

-Blur images with various low pass filters

-Apply custom-made filters to images (2D convolution)

Image Blurring

- Image blurring is achieved by colvolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content from the image resulting ing edges being blurred when the filter is applied.
- OpenCV provides 4 types of blurring techniques:
  - Averaging
  - Gaussian Filtering
  - Median Filtering
  - Bilateral Filtering

Averaging

- This filter works by simply taking the average of all the pixels under kernel area and replaces the central element with the average.
- So, a 3x3 normalized box filter would look like:

$$K = 1/9*[[1,1,1],[1,1,1],[1,1,1]]$$

Averaging : Usage and working

- Code:

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('test.png')
6
7 kernel = np.ones((5,5),np.float32)/25
8 dst = cv2.filter2D(img,-1,kernel)
9
10 plt.subplot(121),plt.imshow(img),plt.title('Original')
11 plt.xticks([]), plt.yticks([])
12 plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
13 plt.xticks([]), plt.yticks([])
14 plt.show()
```

Averaging : Output

Original

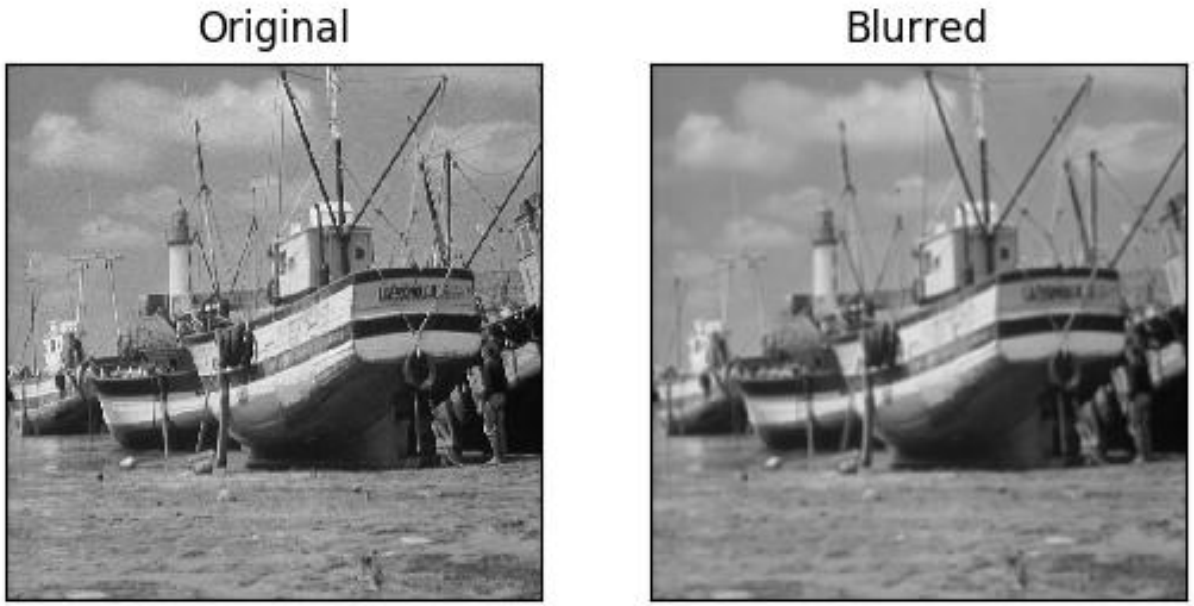
Averaging

Blur using OpenCV

- Code:

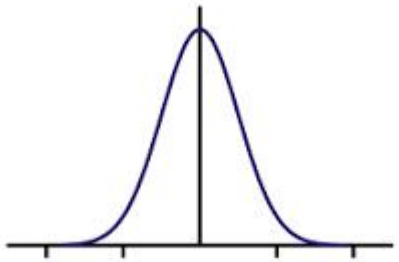
```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('test.png')
6
7 blur = cv2.blur(img,(5,5))
8
9 plt.subplot(121),plt.imshow(img),plt.title('Original')
10 plt.xticks([]), plt.yticks([])
11 plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
12 plt.xticks([]), plt.yticks([])
13 plt.show()
```

Blur Output:



Gaussian Filtering

- A Gaussian filer is the result of blurring an image by a Gaussian function.



- In this approach, instead of a box filter consisting of equal filter coefficients, a gaussian kernel is used.

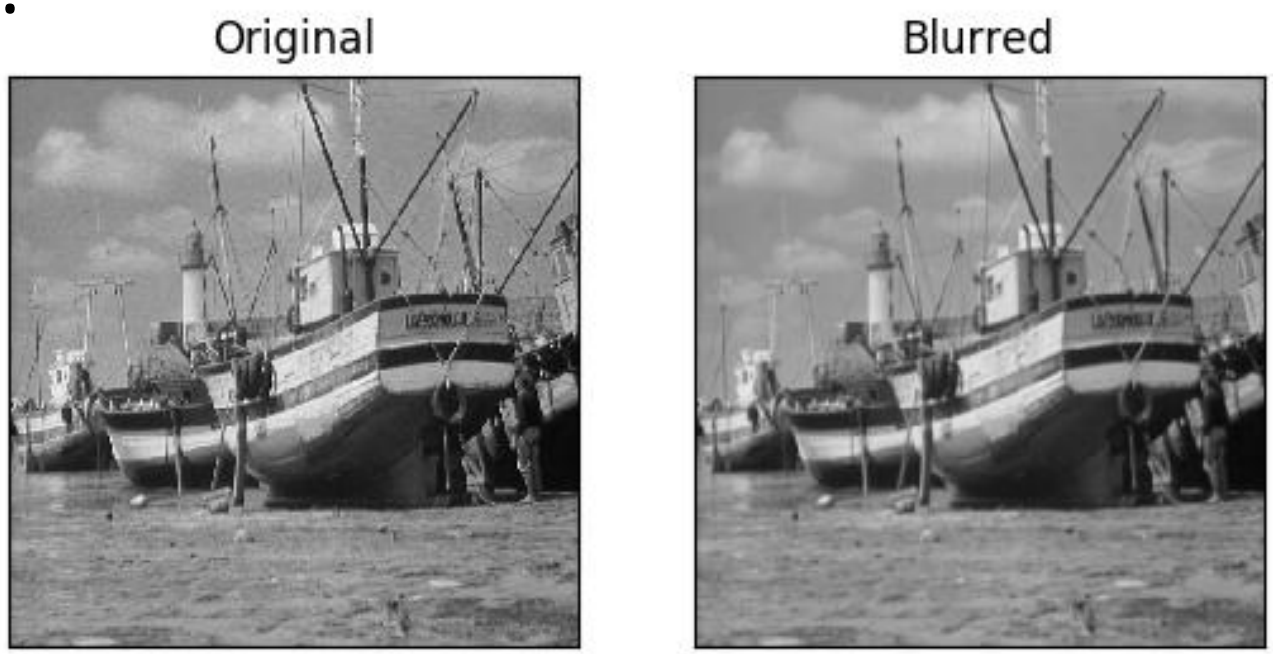
Gaussian Filtering in Python

- Code:

```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('test.png')
6
7 blur = cv2.GaussianBlur(img,(5,5),0)
8
9 plt.subplot(121),plt.imshow(img),plt.title('Original')
10 plt.xticks([]), plt.yticks([])
11 plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
12 plt.xticks([]), plt.yticks([])
13 plt.show()
```

Gaussian filtering

- Output:



Median Filtering

- Here, the filter works by computing the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise.
- One interesting note is that using box or gaussian filter, the central pixel can be any value which might not be present in the image. But in median filter, non new pixel values are computed. The new center pixel is just the median of all the surrounding pixels.

Median Filtering

- Code:

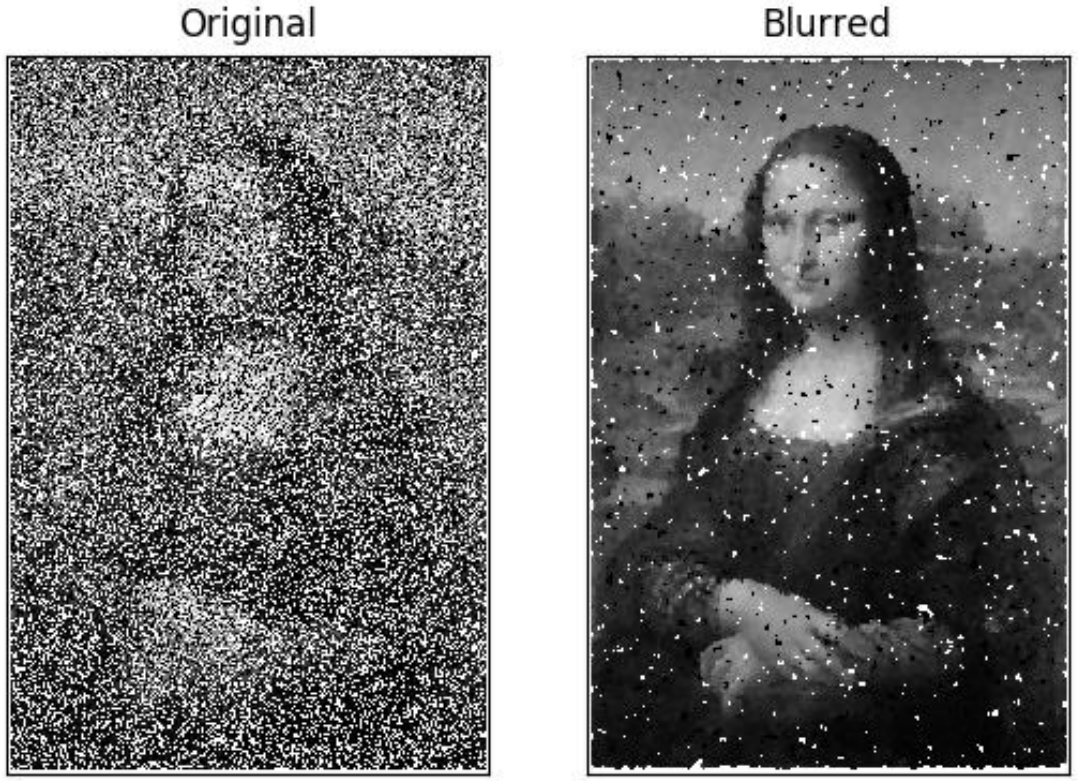
```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 img = cv2.imread('test.png')
6
7 median = cv2.medianBlur(img,5)
8
9 plt.subplot(121),plt.imshow(img),plt.title('Original')
10 plt.xticks([]), plt.yticks([])
11 plt.subplot(122),plt.imshow(median),plt.title('Blurred')
12 plt.xticks([]), plt.yticks([])
13 plt.show()
```

Median Filtering

- Output 1 for normal image:

Median Filtering

- Output for Salt-and-pepper noise in image:



Bilateral Filtering

- Bilateral filter was developed for removing noise while preserving the edges.
- Bilateral filter uses a Gaussian filter in the space domain, but it also uses one more Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are 'spatial neighbors' are considered for filtering.

Bilateral Filtering

- Code:

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('test.png')

blur = cv2.bilateralFilter(img,9,75,75)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Bilateral Filtering

- Output: