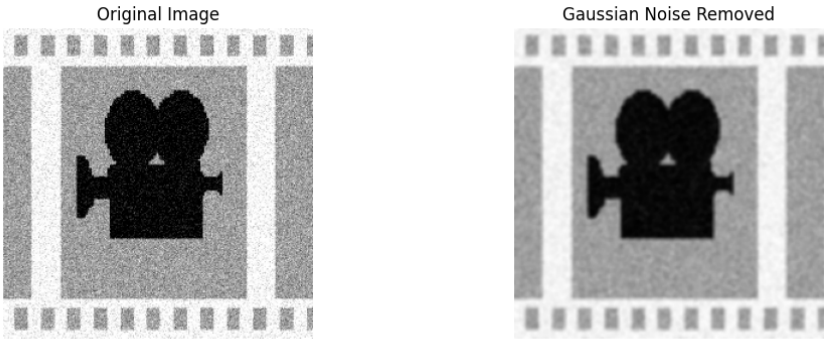
Spatial filtering

When using OpenCV for image processing, I found it very convenient, but I also encountered some unexpected situations due to my unfamiliarity with OpenCV functions.

gaussian_blur

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
image_path = 'image/gaussian_noisy_image.png'
original_image = cv2.imread(image_path, 1)
dst_image = original_image.copy()
def get_gaussian_kernel(size, sigma):
    # Create gaussian_kernel
    kernel = np.zeros((size, size), dtype=np.float32)
    center = size // 2
    for x in range(-center, center + 1):
        for y in range(-center, center + 1):
            kernel[x + center, y + center] = (
                (1 / (2 * np.pi * sigma ** 2)) *
                np.exp(-(x ** 2 + y ** 2) / (2 * sigma **
2))
            )
    kernel /= np.sum(kernel)
    return kernel
def gaussian_blur(src_image, size, sigma):
    img = src_image.copy()
    dst_image = np.zeros_like(img, dtype=np.float32)
    gaus = get_gaussian_kernel(size, sigma)
    center = size // 2
    img_padded = cv2.copyMakeBorder(img, center, center, ce
nter, center,cv2.BORDER_REPLICATE)
```

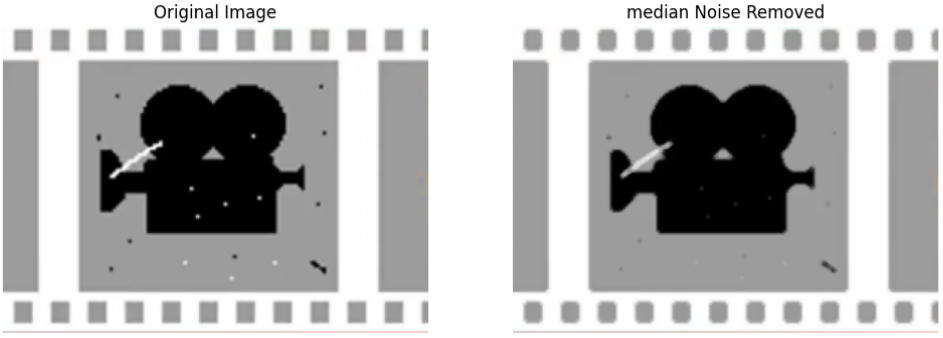
```
    for i in range(center, img_padded.shape[0] - center):
        for j in range(center, img_padded.shape[1] - cente
r):
            for x in range(-center, center + 1):
                for y in range(-center, center + 1):
                    dst_image[i - center, j - center] += im
g_padded[i + x, j + y]* gaus[x + center, y + center]
    dst_image = np.clip(dst_image, 0, 255).astype(np.uint8)
    return dst_image
# denoise = cv2.GaussianBlur(original_image, ksize=3)
denoise = gaussian_blur(original_image, 5, 1.5)
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].imshow(original_image, cmap='gray')
axs[0].set_title("Original Image")
axs[0].axis("off")
axs[1].imshow(denoise, cmap='gray')
axs[1].set_title("Gaussian Noise Removed")
axs[1].axis("off")
plt.show()
```



median

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
image_path = 'image/01.png'
original_image = cv2.imread(image_path, 1)
```

```
median = cv2.medianBlur(original_image, 5)
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].imshow(original_image, cmap='gray')
axs[0].set_title("Original Image")
axs[0].axis("off")
axs[1].imshow(median, cmap='gray')
axs[1].set_title("median Noise Removed")
axs[1].axis("off")
plt.show()
```



mean_filtered and GaussianBlur

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('image/turumai-132.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
def add_gaussian_noise(image, percentage=0.9, mean=0, std=2
5):
    noisy_image = image.copy()
    num_noisy_pixels = int(percentage * image.size / 3)
    for _ in range(num_noisy_pixels):
        x = np.random.randint(0, image.shape[0])
        y = np.random.randint(0, image.shape[1])
        noisy_image[x, y, 0] = np.clip(noisy_image[x, y, 0]
+ np.random.normal(mean, std), 0, 255)
        noisy_image[x, y, 1] = np.clip(noisy_image[x, y, 1]
+ np.random.normal(mean, std), 0, 255)
```

```
        noisy_image[x, y, 2] = np.clip(noisy_image[x, y, 2]
+ np.random.normal(mean, std), 0, 255)
    return noisy_image
noisy_image = add_gaussian_noise(image, percentage=0.9)
mean_filtered = cv2.blur(noisy_image, (5, 5))
gaussian_filtered = cv2.GaussianBlur(noisy_image, (5, 5),
0)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(noisy_image)
axes[0].set_title("noisy image")
axes[0].axis("off")
axes[1].imshow(mean_filtered)
axes[1].set_title("average filter")
axes[1].axis("off")
axes[2].imshow(gaussian_filtered)
axes[2].set_title("gaussian filter")
axes[2].axis("off")
plt.show()
```

random_binary_noise

```
def add_large_binary_noise(image, percentage=0.0005, noise_
size=3):
    noisy_image = image.copy()
    num_noisy_pixels = int(percentage * image.size / 3)
    for _ in range(num_noisy_pixels):
        x = np.random.randint(0, image.shape[0] - noise_siz
e)
        y = np.random.randint(0, image.shape[1] - noise_siz
e)
        color = [0, 0, 0] if np.random.rand() > 0.5 else [2
55, 255, 255]
        noisy_image[x:x + noise_size, y:y + noise_size] = c
olor
    return noisy_image
binary_noisy_image = add_random_binary_noise(image)
```

median filter

```
median_filtered = cv2.medianBlur(noisy_image, 5)
```

bilateralFilter

```
edge_image = cv2.bilateralFilter(noisy_image, 9, 75, 75)
```

PSNR

```
psnr_value = cv2.PSNR(image, gaussian_filter)
```

motion_blur

```
def motion_blur(image, degree=12, angle=45):
    image = np.array(image)
    M = cv2.getRotationMatrix2D((degree / 2, degree / 2), a
ngle, 1)
    motion_blur_kernel = np.diag(np.ones(degree))
    motion_blur_kernel = cv2.warpAffine(motion_blur_kernel,
M, (degree, degree))
    motion_blur_kernel = motion_blur_kernel / degree
    blurred = cv2.filter2D(image, -1, motion_blur_kernel)
    # convert to uint8
    cv2.normalize(blurred, blurred, 0, 255, cv2.NORM_MINMA
X)
    blurred = np.array(blurred, dtype=np.uint8)
    return blurred
```

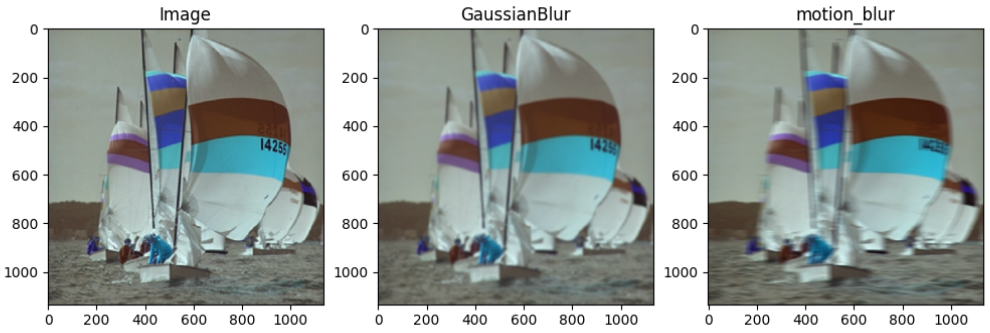


Image
GaussianBlur
motion_blur
0
200
400
600
800
1000