Contents:

Noise:

- Uniform
- Normal
- Gaussian
- Salt and Pepper

Filtering:

- Averaging
- Median
- Gaussian
- Bilateral

Generation of noise

```
import cv2
import numpy as np
import random
a = np.random.uniform(0,1) #  obtain 2 random numbers from uniform distribution.
b = np.random.uniform(0,1)

print(' a: ',a,'\n','b: ',b)
a = np.random.normal(0,1) # get two random numbers from a normal distribution
b = np.random.normal(0,1)

print(' a: ',a,'\n','b: ',b)
```

```
 a:  0.6589342565821048
 b:  0.2647996038326472
 a:  0.9393620489439746
 b:  1.4709091741973537
```

Gaussian noise

cv2. randn(image, mean, standard deviation)

Fills the image with normally distributed random numbers with specified mean and standard

```
from google.colab import files
uploaded=files.upload()
```

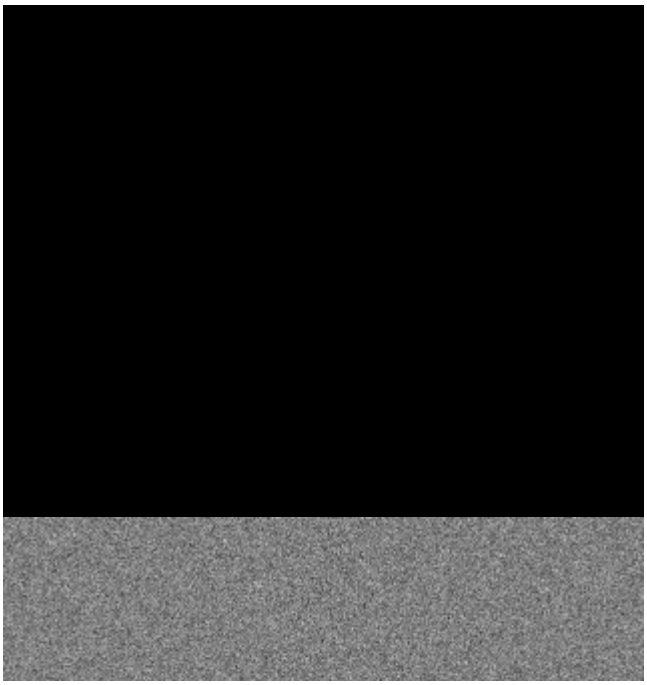
Choose Files flower2.jpg

- **flower2.jpg**(image/jpeg) - 36251 bytes, last modified: 6/22/2021 - 100% done

```
Saving flower2.jpg to flower2.jpg
```

```
img = cv2.imread('flower2.jpg',cv2.IMREAD_GRAYSCALE)
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

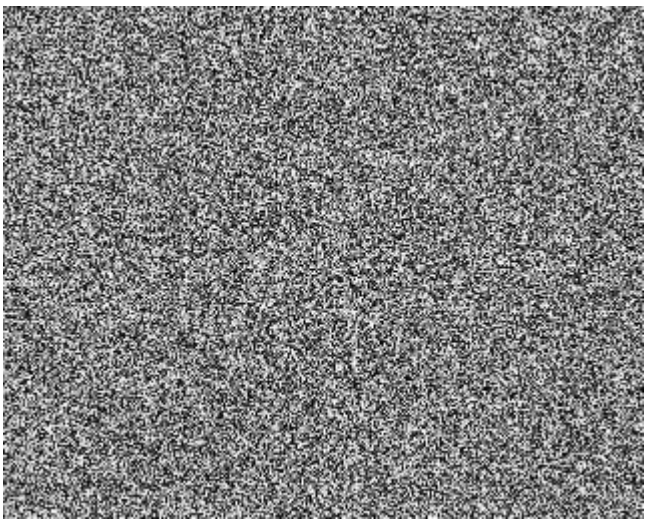
```
# Let's first create a zero image with the same dimensions of the loaded image

gaussian_noise = np.zeros((img.shape[0], img.shape[1]),dtype=np.uint8)

cv2_imshow(gaussian_noise)

# Now, we can set the pixel values as a Gaussian noise. We have set a mean value to 128 an
cv2.randn(gaussian_noise, 128, 20)

cv2_imshow(gaussian_noise)
cv2.imwrite("Gaussian random noise.jpg",gaussian_noise)
```



Uniform noise

cv2.randu(dst, low, high)

```
1 uniform_noise = np.zeros((img.shape[0], img.shape[1]),dtype=np.uint8)
2 cv2.randu(uniform_noise,0,255)
3 cv2_imshow(uniform_noise)
4 cv2.imwrite("Uniform random noise.jpg",uniform_noise)
```

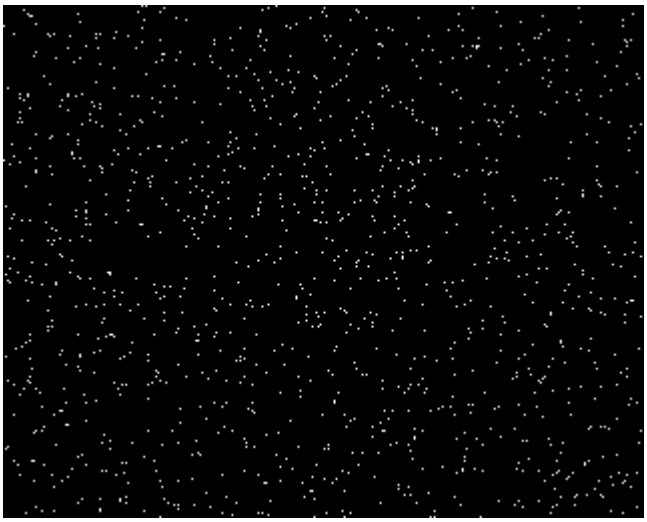


```
True
```

Impulse noise

```
1 impulse_noise = uniform_noise.copy()
2 ret,impulse_noise = cv2.threshold(uniform_noise,250,255,cv2.THRESH_BINARY)
```

```
2 ret,impulse_noise = cv2.threshold(uniform_noise,250,255,cv2.THRESH_BINARY)
3 cv2_imshow(impulse_noise)
4 cv2.imwrite("Impuls noise.jpg",impulse_noise)
```



```
True
```

Adding noise to the image

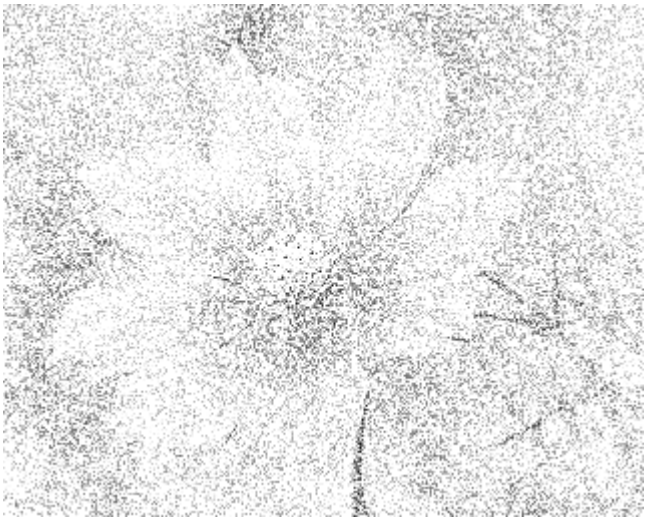
f'(x,y)=f(x,y)+η (x,y)

```
1 # Adding Gaussian noise
2 gaussian_noise = (gaussian_noise*0.5).astype(np.uint8)
3 noisy_image1 = cv2.add(img,gaussian_noise)
4
5 cv2_imshow(noisy_image1)
6
7 cv2.imwrite("Noisy image1.jpg",noisy_image1)
```

```
True
```

```
1 noisy_image2 = cv2.add(img,uniform_noise)
2 cv2_imshow(noisy_image2)
```

```
3
4 cv2.imwrite("Noisy image2.jpg",noisy_image2)
```



```
True
```

```
1 impulse_noise = (impulse_noise).astype(np.uint8)
2 noisy_image3 = cv2.add(img,impulse_noise)
3
4 cv2_imshow(noisy_image3)
5 cv2.imwrite("Noisy image3.jpg",noisy_image3)
```

```
True
```

Median filtering

```
1 blurred1 = cv2.medianBlur(noisy_image1, 3)
2 st1=np.hstack((noisy_image1,blurred1))
3 cv2_imshow(st1)
4 cv2.imwrite("Median filter - Gaussian noise.jpg",blurred1)
5
6 blurred2 = cv2.medianBlur(noisy_image2, 3)
```

```
st2=np.hstack((noisy_image2,blurred2))
cv2_imshow(st2)
cv2.imwrite("Median filter - Uniform noise.jpg",blurred2)

blurred3 = cv2.medianBlur(noisy_image3, 3)
st3=np.hstack((noisy_image3,blurred3))
cv2_imshow(st3)
cv2.imwrite("Median filter - Impuls noise.jpg",blurred3)
```

```
True
```

Averaging filtering

blur() also smoothens the image.

```
processed_image1 = cv2.blur(noisy_image1,(5,5))
st1=np.hstack((noisy_image1,processed_image1))
cv2_imshow(st1)

processed_image2 = cv2.blur(noisy_image2,(5,5))
st2=np.hstack((noisy_image2,processed_image2))
cv2_imshow(st2)

processed_image3 = cv2.blur(noisy_image3,(5,5))
st3=np.hstack((noisy_image3,processed_image3))
cv2_imshow(st3)

```

Gaussian Filtering

Gaussian filter performs better than averaging filter on noisy image.

Syntax: cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])

```
1
2 image_gaussian_processed = cv2.GaussianBlur(noisy_image1,(5,5),1)
3 st1=np.hstack((noisy_image1,processed_image1,image_gaussian_processed))
4 cv2_imshow(st1)
5 #cv2_imshow(image_gaussian_processed)
```

Bilateral Filtering

Syntax: cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])

```
1 # Apply bilateral filter with diameter of each pixel neighbourhood = 15,
2 # sigmaColor = sigmaSpace = 75.
3 bilateral = cv2.bilateralFilter(noisy_image1, 5, 75, 75) # we can change sigmacolor and si
4 #cv2_imshow(bilateral)
5 st1=np.hstack((processed_image1,image_gaussian_processed,bilateral))
6 cv2_imshow(st1)
7 # Save the output.
8 cv2.imwrite('flower_bilateral.jpg', bilateral)
```

```
True
```

2D Convolution (Image Filtering)

Syntax: cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])

when ddepth=-1, the output image will have the same depth as the source.

```
1 kernel = np.ones((5,5),np.float32)/25 # 5 X 5 averaging mask
2 dst = cv2.filter2D(img,-1,kernel)
3 cv2_imshow(img)
4 cv2_imshow(dst)
```

✓ 0s completed at 12:29