Experiment-4

Contrast Enhancement (Histogram Eq.) and Spatial Filtering

Name : N U Praneeth Reddy **Reg.No: 21BAI1500**

Aim: To create programs in Python to complete the given image processing tasks which include image sharpening, smoothening and histogram equalization.

Resources Used: Anaconda Python Environment

Google Colab Jupyter Notebook

Theory :

OpenCV stands as an open-source library designed for computer vision and machine learning applications. Its primary goal is to offer a unified foundation for computer vision projects and to facilitate the integration of machine perception into various commercial products.

On the other hand, NumPy serves as a Python library, enabling support for large, multi-dimensional arrays and matrices, accompanied by an extensive array of high-level mathematical functions for manipulating these arrays.

Additionally, Matplotlib functions as a Python plotting library, directly connected to the numerical mathematics capabilities of NumPy. It delivers an object-oriented API for seamlessly embedding plots within applications.

Tasks:

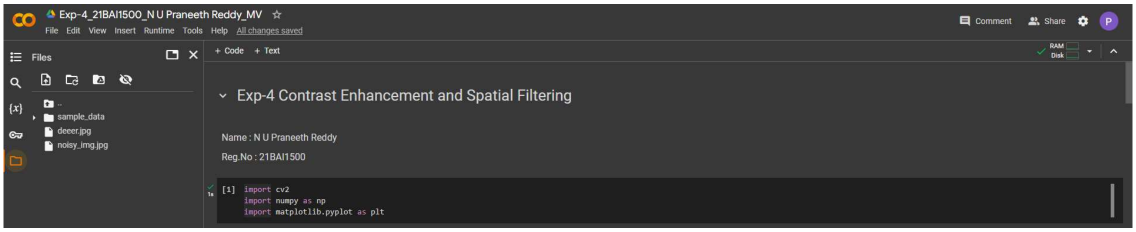
1) Perform Contrast Enhancement using Histogram equalization of a sample image and observe the histogram before and after processing. Provide your inference.

2) Perform image smoothening of an input image based on ideal filters (take kernels of different size) and gaussian filters. Observe the results.

3) Perform image sharpening of an input image based on ideal filters (take different kernels values). Observe the results.

Procedure :

- Open Google Colab and create a new Jupyter Notebook.
- Import important libraries namely OpenCV, Numpy and Matplotlib.


Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
+ Code + Text
RAM
Disk
sample_data
deeer.jpg
noisy_img.jpg

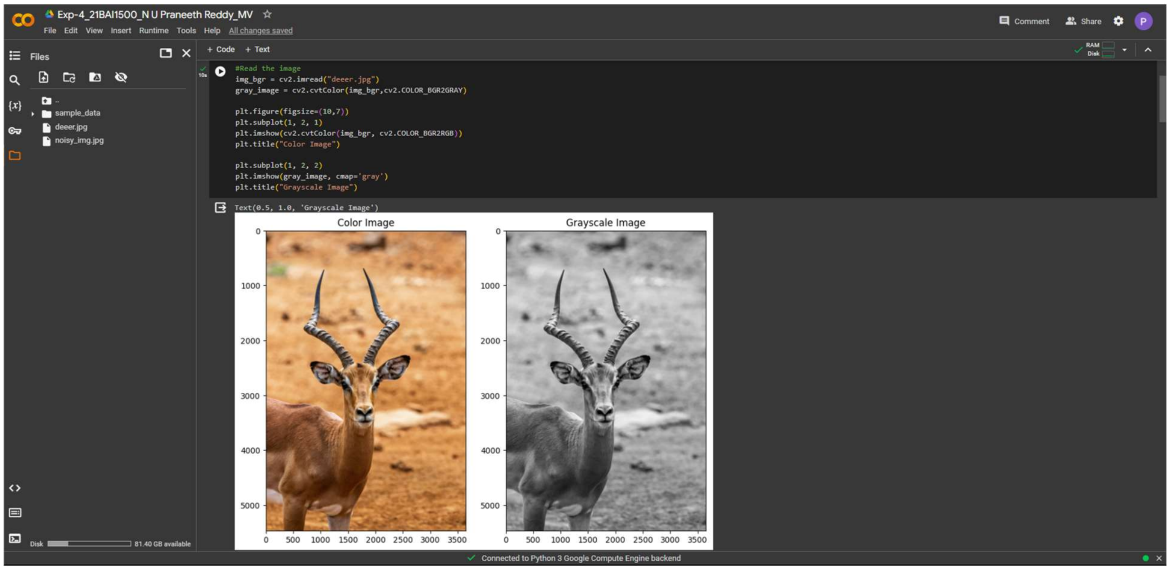
Exp-4 Contrast Enhancement and Spatial Filtering

Name : N U Praneeth Reddy

Reg.No : 21BAI1500

```
[1] import cv2
    import numpy as np
    import matplotlib.pyplot as plt
```

- Read the image using imread in the OpenCV library in BGR (Blue-Green-Red) format and convert it to gray scale and print both the images side by side using subplot.

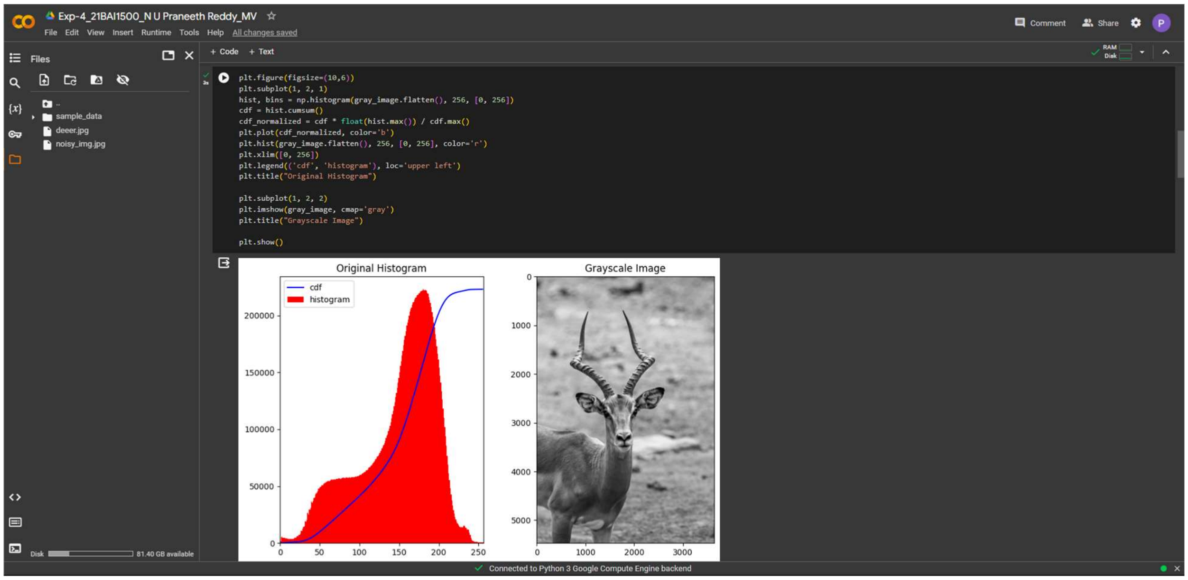


```
#Read the image
img_bgr = cv2.imread("deeer.jpg")
gray_image = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)

plt.figure(figsize=(10,7))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB))
plt.title("Color Image")

plt.subplot(1, 2, 2)
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")
```

```
Text(0.5, 1.0, 'Grayscale Image')
```

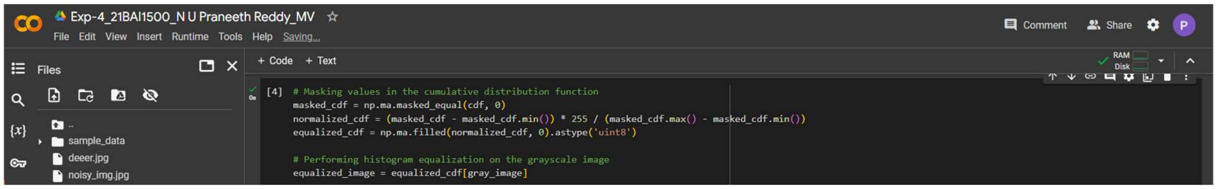
<u>Task – 1 :</u> Perform Contrast Enhancement using Histogram equalization of a sample image and observe the histogram before and after processing. Provide your inference.

- Before Histogram equalization



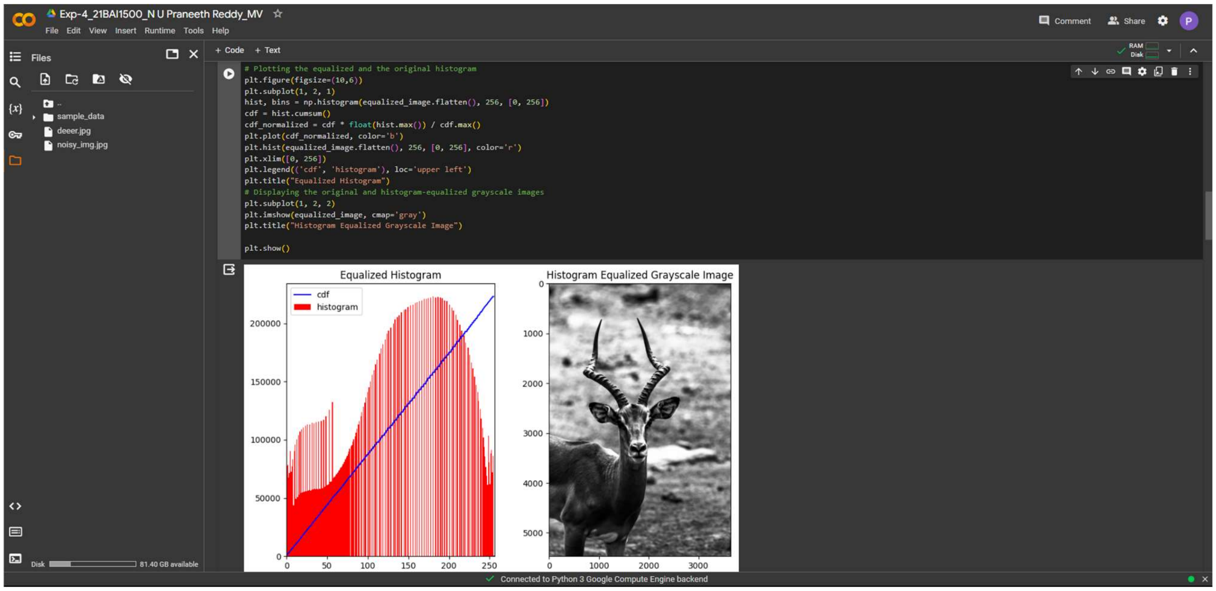
```
plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
hist, bins = np.histogram(gray_image.flatten(), 256, [0, 256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color='b')
plt.hist(gray_image.flatten(), 256, [0, 256], color='r')
plt.xlim([0, 256])
plt.legend(('cdf', 'histogram'), loc='upper left')
plt.title("Original Histogram")

plt.subplot(1, 2, 2)
plt.imshow(gray_image, cmap='gray')
plt.title("Grayscale Image")

plt.show()
```

- After Histogram equalization



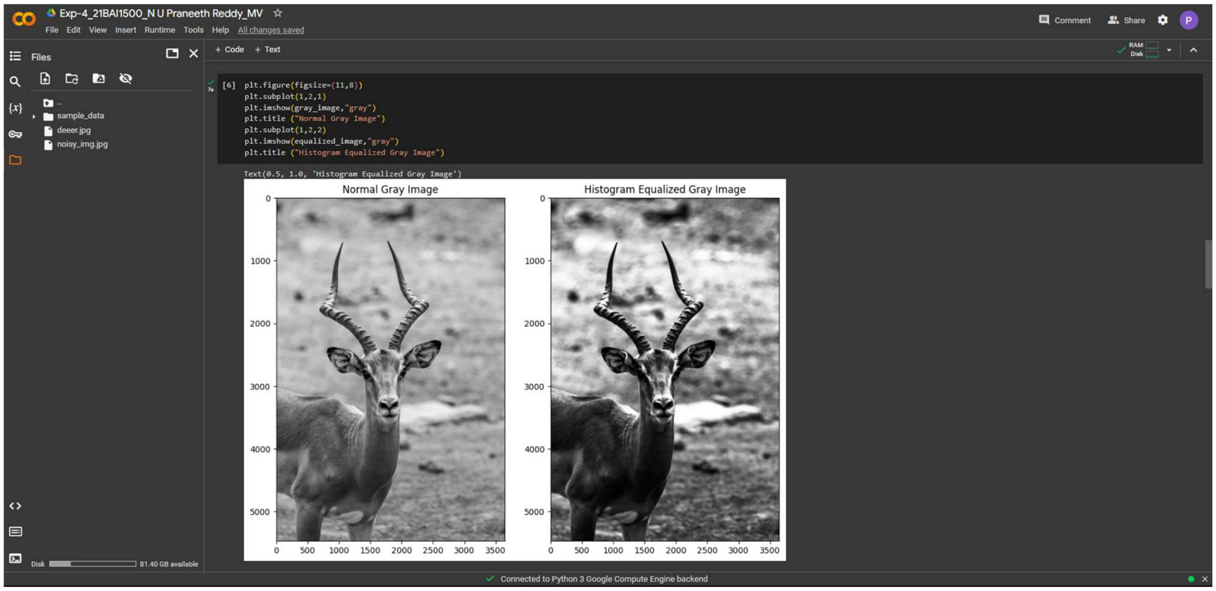
```
[4] # Masking values in the cumulative distribution function
    masked_cdf = np.ma.masked_equal(cdf, 0)
    normalized_cdf = (masked_cdf - masked_cdf.min()) * 255 / (masked_cdf.max() - masked_cdf.min())
    equalized_cdf = np.ma.filled(normalized_cdf, 0).astype('uint8')

    # Performing histogram equalization on the grayscale image
    equalized_image = equalized_cdf[gray_image]
```

Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help Saving...
Comment
Share
Files
sample_data
deeer.jpg
noisy_img.jpg
+ Code + Text
RAM
Disk



```
# Plotting the equalized and the original histogram
plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
hist, bins = np.histogram(equalized_image.flatten(), 256, [0, 256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
plt.plot(cdf_normalized, color='b')
plt.hist(equalized_image.flatten(), 256, [0, 256], color='r')
plt.xlim([0, 256])
plt.legend(('cdf', 'histogram'), loc='upper left')
plt.title("Equalized Histogram")
# Displaying the original and histogram-equalized grayscale images
plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title("Histogram Equalized Grayscale Image")

plt.show()
```

- Viewing the images side by side,


Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
+ Code
+ Text
sample_data
deeer.jpg
noisy_img.jpg
Normal Gray Image
Histogram Equalized Gray Image
0
1000
2000
3000
4000
5000
0
500
1000
1500
2000
2500
3000
3500
Disk
81.40 GB available
Connected to Python 3 Google Compute Engine backend

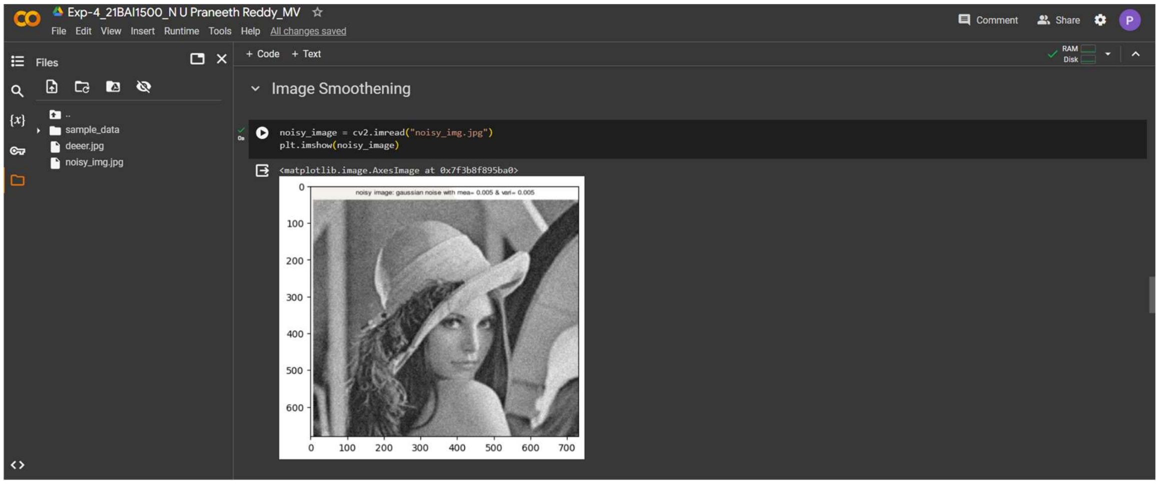
```
[6] plt.figure(figsize=(11,8))
    plt.subplot(1,2,1)
    plt.imshow(gray_image,"gray")
    plt.title ("Normal Gray Image")
    plt.subplot(1,2,2)
    plt.imshow(equalized_image,"gray")
    plt.title ("Histogram Equalized Gray Image")
```

```
Text(0.5, 1.0, 'Histogram Equalized Gray Image')
```

- We can see that the image after the histogram equalization has more distributed intensity compared to before and is much clear due to the contrast enhancement.

Task -2 : Perform image smoothening of an input image based on ideal filters (take kernels of different size) and gaussian filters. Observe the results.

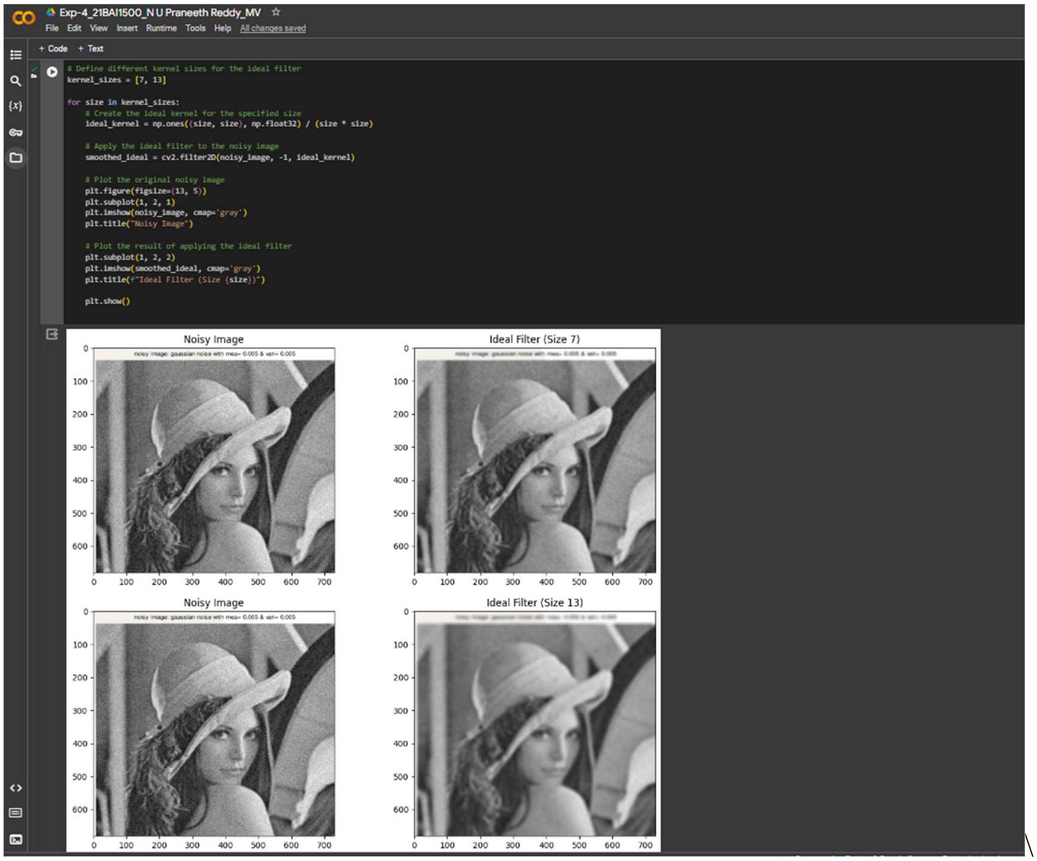
- Take a noisy version of an image,


Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
+ Code
+ Text
sample_data
deeer.jpg
noisy_img.jpg
Image Smoothening
noisy image: gaussian noise with mea= 0.005 & vari= 0.005
0
100
200
300
400
500
600
0
100
200
300
400
500
600
700

```
noisy_image = cv2.imread("noisy_img.jpg")
plt.imshow(noisy_image)
```

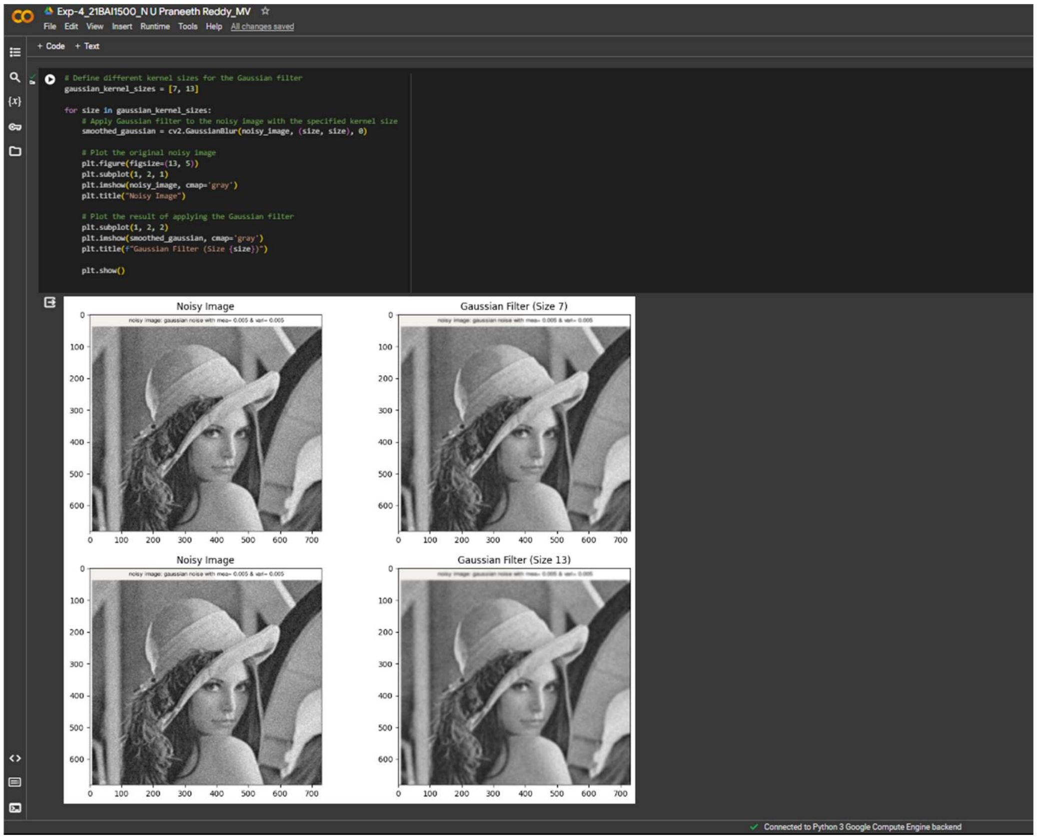
```
<matplotlib.image.AxesImage at 0x7f3b8f895ba0>
```

- Using Ideal filters and the kernel values at 3 and 5 with the ideal filter made of an array of ones


Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Noisy Image
Ideal Filter (Size 7)
Noisy Image
Ideal Filter (Size 13)
noisy image: gaussian noise with mea= 0.005 & var= 0.005
0
100
200
300
400
500
600
700

```python
# Define different kernel sizes for the ideal filter
kernel_sizes = [7, 13]

for size in kernel_sizes:
    # Create the ideal kernel for the specified size
    ideal_kernel = np.ones((size, size), np.float32) / (size * size)

    # Apply the ideal filter to the noisy image
    smoothed_ideal = cv2.filter2D(noisy_image, -1, ideal_kernel)

    # Plot the original noisy image
    plt.figure(figsize=(13, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(noisy_image, cmap='gray')
    plt.title("Noisy Image")

    # Plot the result of applying the ideal filter
    plt.subplot(1, 2, 2)
    plt.imshow(smoothed_ideal, cmap='gray')
    plt.title(f"Ideal Filter (Size {size})")

    plt.show()
```

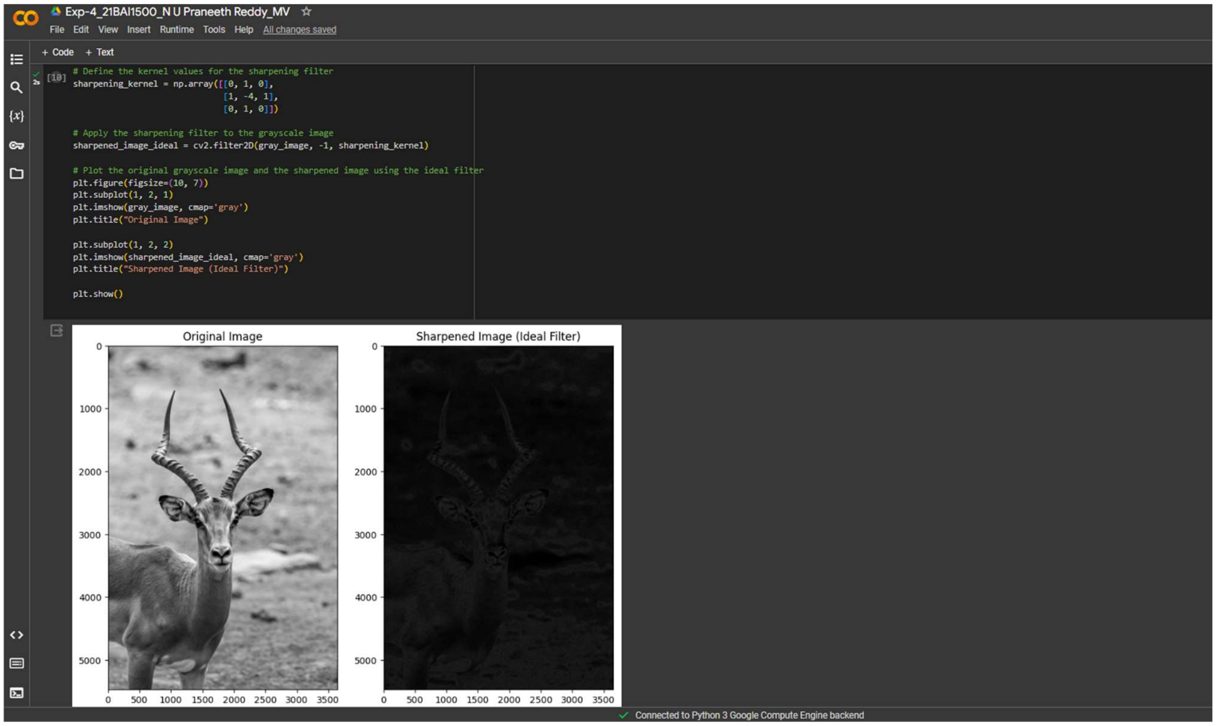
- Using gaussian filters and the kernel filters at sizes 3 and 5


Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Noisy Image
Gaussian Filter (Size 7)
Noisy Image
Gaussian Filter (Size 13)
noisy image: gaussian noise with mea= 0.005 & var= 0.005
0
100
200
300
400
500
600
700

```python
# Define different kernel sizes for the Gaussian filter
gaussian_kernel_sizes = [7, 13]

for size in gaussian_kernel_sizes:
    # Apply Gaussian filter to the noisy image with the specified kernel size
    smoothed_gaussian = cv2.GaussianBlur(noisy_image, (size, size), 0)

    # Plot the original noisy image
    plt.figure(figsize=(13, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(noisy_image, cmap='gray')
    plt.title("Noisy Image")

    # Plot the result of applying the Gaussian filter
    plt.subplot(1, 2, 2)
    plt.imshow(smoothed_gaussian, cmap='gray')
    plt.title(f"Gaussian Filter (Size {size})")

    plt.show()
```

✓ Connected to Python 3 Google Compute Engine backend

Task -3 : Perform image sharpening of an input image based on ideal filters (take different kernels values). Observe the results.



Exp-4_21BAI1500_N U Praneeth Reddy_MV
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[10]

```
# Define the kernel values for the sharpening filter
sharpening_kernel = np.array([[0, 1, 0],
                              [1, -4, 1],
                              [0, 1, 0]])

# Apply the sharpening filter to the grayscale image
sharpened_image_ideal = cv2.filter2D(gray_image, -1, sharpening_kernel)

# Plot the original grayscale image and the sharpened image using the ideal filter
plt.figure(figsize=(10, 7))
plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title("Original Image")

plt.subplot(1, 2, 2)
plt.imshow(sharpened_image_ideal, cmap='gray')
plt.title("Sharpened Image (Ideal Filter)")

plt.show()
```

Original Image
Sharpened Image (Ideal Filter)
0
1000
2000
3000
4000
5000
0 500 1000 1500 2000 2500 3000 3500
Connected to Python 3 Google Compute Engine backend

- We observe that the image has been sharpened.

Results: The given tasks have been done using programs in Python using CV2, Matplotlib and numpy libraries.

Conclusion: Python programs has been created to perform histogram equalization, to perform smoothening using ideal filters and gaussian filters, and to perform image sharpening with ideal filters and the results have been observed.

Google Collab Link :

https://colab.research.google.com/drive/1_lsEEmMikBZ6-GSZEDaQd7G1orWpDxnF?usp=sharing