CSE463\_02\_24241266\_PravakarDas

[Libraries: 2](#_qw3wlute8kmz)

[Folder Access: 2](#_678hnmeak08p)

[kernels & other functions: 2](#_bnj7jmtwx6bp)

[**TASK 1: 4**](#_rydkpl2qjxv4)

[CODE: 4](#_mzdez6j2wtbo)

[Screenshot: 4](#_5v90kai9gg8t)

[Explanation: 5](#_a5xnb2o938j5)

[**TASK 2: 5**](#_4wxrxta5ntty)

[CODE: 5](#_11i5c1qi14rp)

[Screenshot: 6](#_4x3vcn3kivbl)

[Explanation: 6](#_34wdz4d7sslg)

[**TASK 3: 6**](#_1o66nxkrnsh9)

[CODE: 6](#_qdcogb82o4qq)

[Screenshot: 7](#_t9op16ej42vn)

[Explanation: 7](#_l1gnghm55uw)

[**TASK 4: 8**](#_gjlkp1oul01f)

[CODE: 8](#_lt2rw5fd8kbp)

[Screenshot: 9](#_u60k3qgh1t2c)

[Explanation: 9](#_e27vvqjf8htm)

[**TASK 5: 9**](#_rulbty8nrh9f)

[CODE: 9](#_utnh8pmetl1)

[Screenshot: 10](#_pjbv1m9dyy6e)

[Explanation: 10](#_fgzq0osqcsva)

[**TASK 6: 10**](#_qwe681g29qva)

[CODE: 10](#_zemopkzfcu9)

[Screenshot: 11](#_vqkqi88a0so)

[Explanation: 11](#_51wvzpp7qfew)

[**TASK 7: 11**](#_8ascwrpumg71)

[CODE: 12](#_hagp3k9hsh4q)

[Screenshot:](#_lz5tu376vmq9) 12

[Explanation: 13](#_8cfe64kws6n4)

[**TASK 8.1: 13**](#_19e15qup056y)

[CODE: 13](#_732jnuqw9rrc)

[Screenshot: 14](#_eeg8zcup1x24)

[Explanation: 14](#_qf8e7doxyjav)

[**TASK 8.2: 14**](#_naeynw9vcpsr)

[CODE: 14](#_sm2pc3qqrgca)

[Screenshot: 15](#_g768ife041r7)

[Explanation: 15](#_qd2y4jk9g22j)

| Libraries: import numpy as np  import cv2  import matplotlib.pyplot as plt  import os |
| --- |
| Folder Access: folder = '/content/drive/MyDrive/CSE463-ComputerVision/24241266\_PravakarDas\_Lab2/24241266\_PravakarDas\_Lab02/'  images = [i for i in os.listdir(folder) if i.endswith('.jpg')]  img\_paths = {f'img{i+1}\_path': os.path.join(folder, images[i]) for i in range(len(images))}  print(img\_paths)  def resize\_image(image, new\_size=(512, 512)):  return cv2.resize(image, new\_size) |
| kernels & other functions: #kernels & other functions:  identity\_kernel = np.array([[0, 0, 0],  [0, 1, 0],  [0, 0, 0]], dtype=np.float32)  sharpen\_kernel = np.array([[0, -1, 0],  [-1, 5, -1],  [0, -1, 0]], dtype=np.float32)  def gaussian\_noise(image, mean=0, sigma=25):  noise = np.random.normal(mean, sigma, image.shape)  noisy\_image = np.clip(image + noise, 0, 255).astype(np.uint8)  return noisy\_image  vertical\_kernel = np.array([[-1, 0, 1],  [-1, 0, 1],  [-1, 0, 1]], dtype=np.float32)  horizontal\_kernel = np.array([[-1, -1, -1],  [0, 0, 0],  [1, 1, 1]], dtype=np.float32) |

# 

# TASK 1:

| CODE: #1. Apply Convolution with a Simple Kernel  img1 = cv2.imread(img\_paths['img1\_path'], cv2.IMREAD\_GRAYSCALE)  img1 = resize\_image(img1)  convolved\_img1 = cv2.filter2D(img1, -1, identity\_kernel)  plt.figure(figsize=(10,5))  plt.subplot(1,2,1)  plt.imshow(img1, cmap='gray')  plt.title('Original Image-1')  plt.subplot(1,2,2)  plt.imshow(convolved\_img1, cmap='gray')  plt.title('After Identity Kernel Convolution')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The output image remains almost identical to the input image cause the identity kernel preserves the original pixel values which means applying an identity filter does not alter the image's content, making it a useful baseline for comparison. |

# 

# TASK 2:

| CODE: #2. Custom Kernel Design  img2 = cv2.imread(img\_paths['img2\_path'], cv2.IMREAD\_GRAYSCALE)  img2 = resize\_image(img2)  sharpened\_img = cv2.filter2D(img2, -1, sharpen\_kernel)  plt.figure(figsize=(10,5))  plt.subplot(1,2,1)  plt.imshow(img2, cmap='gray')  plt.title('Original Image-2')  plt.subplot(1,2,2)  plt.imshow(sharpened\_img, cmap='gray')  plt.title('Sharpened Image')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The output image displays enhanced edge details and improved contrast due to the sharpening effect,but it also reveals some underlying noise. This occurs because the sharpening kernel accentuates high-frequency components, making both the edges and the inherent noise more prominent. |

# 

# TASK 3:

| CODE: #3. Experiment with Different Padding Techniques  img3 = cv2.imread(img\_paths['img3\_path'], cv2.IMREAD\_GRAYSCALE)  img3 = resize\_image(img3)  img3\_sharpen\_constant = cv2.filter2D(img3, -1, sharpen\_kernel, borderType=cv2.BORDER\_CONSTANT)  img3\_sharpen\_reflect = cv2.filter2D(img3, -1, sharpen\_kernel, borderType=cv2.BORDER\_REFLECT)  img3\_sharpen\_replicate = cv2.filter2D(img3, -1, sharpen\_kernel, borderType=cv2.BORDER\_REPLICATE)  plt.figure(figsize=(15,5))  plt.subplot(1,3,1)  plt.imshow(img3\_sharpen\_constant, cmap='gray')  plt.title('Sharpening with Constant Padding')  plt.subplot(1,3,2)  plt.imshow(img3\_sharpen\_reflect, cmap='gray')  plt.title('Sharpening with Reflect Padding')  plt.subplot(1,3,3)  plt.imshow(img3\_sharpen\_replicate, cmap='gray')  plt.title('Sharpening with Replicate Padding')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The output images with constant, reflect, and replicate padding look very similar cause the differences in border handling are minimal compared to the central part of the image, where most of the details lie. Padding defines how the filter deals with image edges during convolution, and here, even though each method treats the borders slightly differently, the overall effect on the noise-reduced, sharpened image is almost identical. |

# 

# TASK 4:

| CODE: #4. Adding Noise and Applying an Average Filter  img4 = cv2.imread(img\_paths['img4\_path'], cv2.IMREAD\_COLOR)  img4 = resize\_image(img4)  img4 = cv2.cvtColor(img4, cv2.COLOR\_BGR2RGB)  noisy\_img4 = gaussian\_noise(img4)  average\_filtered\_img4 = cv2.blur(noisy\_img4, (5,5))  # Plot the original, noisy, and filtered images  plt.figure(figsize=(15,5))  plt.subplot(1,3,1)  plt.imshow(img4)  plt.title('Original Image [4]')  plt.subplot(1,3,2)  plt.imshow(noisy\_img4)  plt.title('Noisy Image')  plt.subplot(1,3,3)  plt.imshow(average\_filtered\_img4)  plt.title('Average Filtered Image')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The noisy image exhibits significant noise, resulting in a contrasty and hazy appearance where individual pixels remain discernible due to the 250×250 resolution. In contrast, the average filtered image is heavily blurred, as the averaging process smooths out the high-frequency noise, leading to a loss of detail. |

# 

# TASK 5:

| CODE: #5. Gaussian Blur for Smoothing  gaussian\_blur\_sigma0 = cv2.GaussianBlur(noisy\_img4, (5,5), sigmaX=0)  gaussian\_blur\_sigma1 = cv2.GaussianBlur(noisy\_img4, (5,5), sigmaX=1)  gaussian\_blur\_sigma2 = cv2.GaussianBlur(noisy\_img4, (5,5), sigmaX=2)  plt.figure(figsize=(20,5))  plt.subplot(1,4,1)  plt.imshow(noisy\_img4)  plt.title('Noisy Image')  plt.subplot(1,4,2)  plt.imshow(gaussian\_blur\_sigma0)  plt.title('Gaussian Blur (sigma=0)')  plt.subplot(1,4,3)  plt.imshow(gaussian\_blur\_sigma1)  plt.title('Gaussian Blur (sigma=1)')  plt.subplot(1,4,4)  plt.imshow(gaussian\_blur\_sigma2)  plt.title('Gaussian Blur (sigma=2)')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The Gaussian blurred images show that as the sigma value increases, the image becomes increasingly unclear.At sigma 2, the image is almost entirely blurred, emphasizing how a higher standard deviation in the Gaussian kernel leads to stronger smoothing and a significant loss of detail. |

# 

# TASK 6:

| CODE: #6. Applying Laplacian Filter for Edge Detection  img5 = cv2.imread(img\_paths['img5\_path'], cv2.IMREAD\_GRAYSCALE)  img5 = resize\_image(img5)  laplacian\_img5 = cv2.Laplacian(img5, cv2.CV\_64F)  laplacian\_img5 = cv2.convertScaleAbs(laplacian\_img5)  plt.figure(figsize=(10,5))  plt.subplot(1,2,1)  plt.imshow(img5, cmap='gray')  plt.title('Original Image [5]')  plt.subplot(1,2,2)  plt.imshow(laplacian\_img5, cmap='gray')  plt.title('Laplacian Edge Detection')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The Laplacian filter effectively captures abrupt intensity transitions, highlighting edges and contours in the image.Interestingly, resizing the image before applying the filter can help unify the scale of details and reduce noise,making edges stand out more clearly—whereas on a large, high-resolution image, subtle intensity variations and noise can interfere with clean edge detection. |

# 

# TASK 7:

| CODE: #7. Estimating Horizontal and Vertical Gradients  img6 = cv2.imread(img\_paths['img6\_path'], cv2.IMREAD\_GRAYSCALE)  img6 = resize\_image(img6)  vertical\_edges = cv2.filter2D(img6, -1, vertical\_kernel)  horizontal\_edges = cv2.filter2D(img6, -1, horizontal\_kernel)  plt.figure(figsize=(15,5))  plt.subplot(1,3,1)  plt.imshow(img6, cmap='gray')  plt.title('Original Image [6]')  plt.subplot(1,3,2)  plt.imshow(vertical\_edges, cmap='gray')  plt.title('Vertical Edge Detection')  plt.subplot(1,3,3)  plt.imshow(horizontal\_edges, cmap='gray')  plt.title('Horizontal Edge Detection')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The vertical edge detection image highlights edges where intensity changes occur from left to right, the horizontal edge detection image emphasizes edges where intensity changes occur from top to bottom. By applying these two different gradient kernels, we can clearly see how each orientation of edges (vertical vs. horizontal) is captured in separate outputs. |

# 

# TASK 8.1:

| CODE: #8. Histogram Equalization for Contrast Enhancement  img7 = cv2.imread(img\_paths['img7\_path'], cv2.IMREAD\_GRAYSCALE)  img7=resize\_image(img7)  equalized\_img7 = cv2.equalizeHist(img7)  # Plot the original and equalized images  plt.figure(figsize=(10,5))  plt.subplot(1,2,1)  plt.imshow(img7, cmap='gray')  plt.title('Original Low-Contrast Image [7]')  plt.subplot(1,2,2)  plt.imshow(equalized\_img7, cmap='gray')  plt.title('Histogram Equalized Image')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The original low-contrast image exhibits a narrow range of gray values, making it appear somewhat flat. After histogram equalization, the intensity distribution is stretched, enhancing contrast and revealing more detail in both darker and lighter regions of the image. |

# 

# TASK 8.2:

| CODE: #8. Histogram Equalization for Contrast Enhancement  equalized\_img7\_first = cv2.equalizeHist(img7)  equalized\_img7\_second = cv2.equalizeHist(equalized\_img7\_first)  equalized\_img7\_third = cv2.equalizeHist(equalized\_img7\_second)  plt.figure(figsize=(15,5))  plt.subplot(1,4,1)  plt.imshow(img7, cmap='gray')  plt.title('Original Image [7]')  plt.subplot(1,4,2)  plt.imshow(equalized\_img7\_first, cmap='gray')  plt.title('1st Equalization')  plt.subplot(1,4,3)  plt.imshow(equalized\_img7\_second, cmap='gray')  plt.title('2nd Equalization')  plt.subplot(1,4,4)  plt.imshow(equalized\_img7\_third, cmap='gray')  plt.title('3rd Equalization')  plt.show() |
| --- |
| Screenshot: |
| Explanation: The image’s intensity distribution is re-stretched from an already modified range. As a result, the midtones and darker regions become more compressed or shifted, causing the image to appear darker yet more contrasty. So, it processes new details but also saturates certain areas, leading to potential artifacts or overly intense contrast. |