**Homework: Implement Figure 3.10 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('TkAgg')

**import** matplotlib.pyplot as plt

**import** os

**def** contrast\_stretch(input\_image):

    """

    Performs contrast stretching on the input image

    """

    r\_min **=** np.min(input\_image)

    r\_max **=** np.max(input\_image)

**if** r\_max **==** r\_min:

**return** input\_image.astype('uint8')

    input\_image\_float **=** input\_image.astype(float)

    output\_image **=** (input\_image\_float **-** r\_min) **\*** (255.0 **/** (r\_max **-** r\_min))

**return** output\_image.astype('uint8')

**def** threshold\_processing(input\_image, threshold\_value):

    """

    Performs grayscale thresholding on the image

    """

    output\_image **=** np.zeros\_like(input\_image)

    output\_image[input\_image >**=** threshold\_value] **=** 255

**return** output\_image.astype('uint8')

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    image\_path **=** "Fig0310(b)(washed\_out\_pollen\_image).tif"

    original\_image **=** imageio.imread(image\_path)

    stretched\_image **=** contrast\_stretch(original\_image)

    m **=** int(np.mean(original\_image))

    thresholded\_image **=** threshold\_processing(original\_image, m)

    # Visualization

    plt.figure(figsize**=**(18, 6))

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'Original Image')

    plt.axis('off')

    plt.subplot(1, 3, 2)

    plt.imshow(stretched\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'Contrast Stretch Result')

    plt.axis('off')

    plt.subplot(1, 3, 3)

    plt.imshow(thresholded\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'Threshold Processing Result')

    plt.axis('off')

    plt.tight\_layout()

    output\_path **=** "combined\_results.png"

    plt.savefig(output\_path, bbox\_inches**=**'tight')

    imageio.imwrite("Fig0310\_stretched.tif", stretched\_image)

imageio.imwrite("Fig0310\_thresholded.tif", thresholded\_image)

Output Images:



**Homework: Implement Figure 3.14 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**def** bit\_plane\_slice(input\_image, bit\_position):

    """

    Extracts a specific bit plane from an 8-bit grayscale image

    """

    bit\_plane **=** ((input\_image >> bit\_position) & 1) **\*** 255

**return** bit\_plane.astype('uint8')

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    image\_path **=** "./original\_image/Fig0314.png"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

**if** original\_image.dtype !**=** np.uint8:

**if** np.max(original\_image) > 255:

            original\_image **=** (original\_image **/** np.max(original\_image) **\*** 255).astype(np.uint8)

**else**:

            original\_image **=** original\_image.astype(np.uint8)

    bit\_planes **=** []

**for** i **in** range(8):

        plane **=** bit\_plane\_slice(original\_image, i)

        bit\_planes.append(plane)

    # Visualization

    plt.figure(figsize**=**(15, 15))

    plt.subplot(3, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('Original Image')

    plt.axis('off')

**for** i **in** range(8):

        bit\_plane\_idx\_for\_display **=** 7 **-** i

        plt.subplot(3, 3, i **+** 2)

        plt.imshow(bit\_planes[bit\_plane\_idx\_for\_display], cmap**=**'gray', vmin**=**0, vmax**=**255)

        plt.title(f'Bit-plane {bit\_plane\_idx\_for\_display}')

        plt.axis('off')

    plt.tight\_layout()

    combined\_output\_path **=** os.path.join(output\_dir, "Fig0314\_bit\_planes\_combined\_MSB\_first.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    print(f"Combined visualization (MSB first) saved to: {combined\_output\_path}")

**for** i **in** range(8):

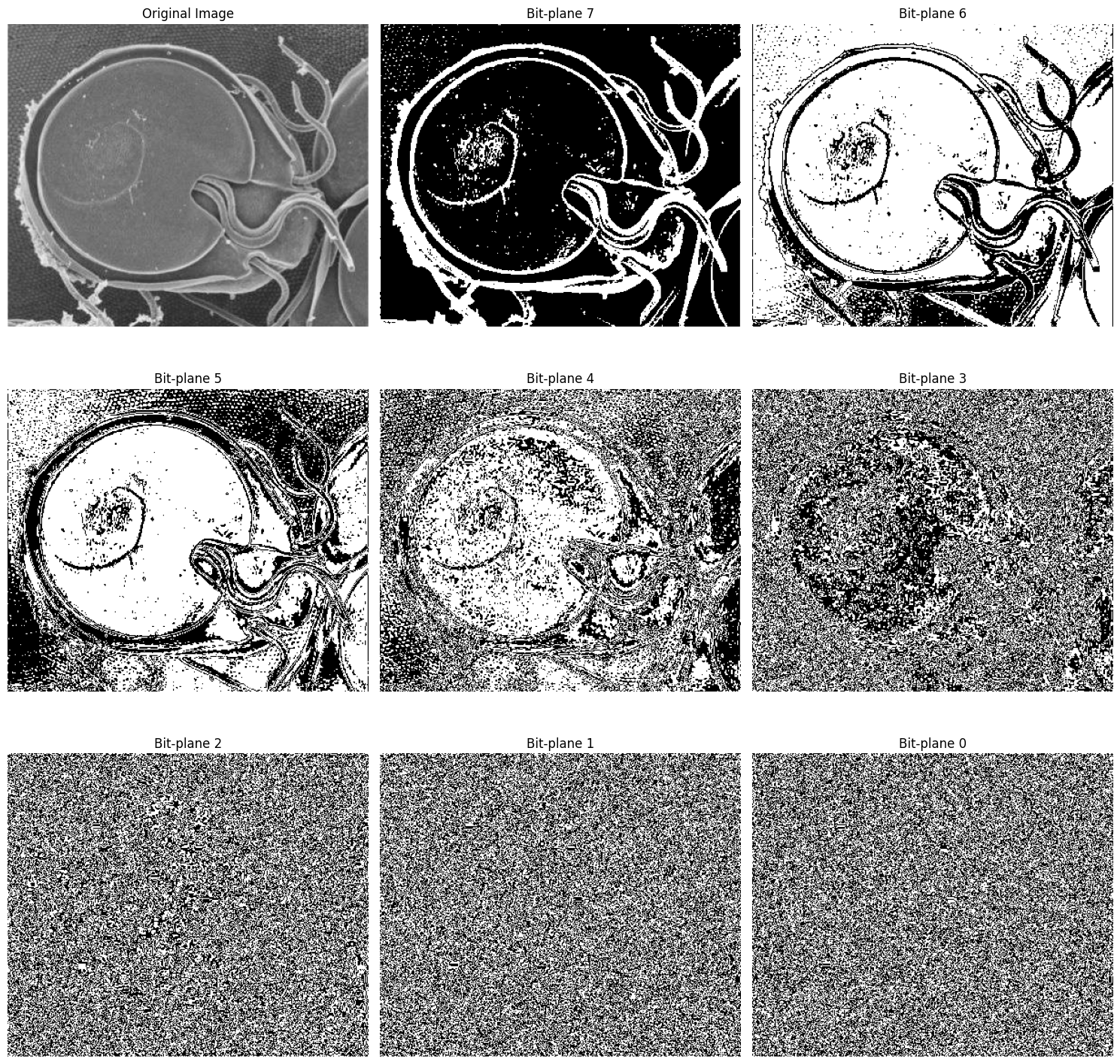
        bit\_plane\_idx\_for\_save **=** 7 **-** i

        individual\_output\_path **=** os.path.join(output\_dir, f"Fig0314\_bit\_plane\_{bit\_plane\_idx\_for\_save}\_MSB\_first.png")

        imageio.imwrite(individual\_output\_path, bit\_planes[bit\_plane\_idx\_for\_save])

        print(f"Bit-plane {bit\_plane\_idx\_for\_save} saved to: {individual\_output\_path}")

Output Images:



**Homework: Implement Figure 3.20 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**def** calculate\_histogram(image, bins**=**256):

    """

    Calculates the histogram of an 8-bit grayscale image.

    """

    hist, \_ **=** np.histogram(image.flatten(), bins, [0, bins])

**return** hist

**def** histogram\_equalization(image):

    """

    Performs histogram equalization on an 8-bit grayscale image.

    """

    hist **=** calculate\_histogram(image)

    cdf **=** hist.cumsum()

    cdf\_min **=** cdf.min()

    cdf\_max **=** cdf.max()

**if** cdf\_max **==** cdf\_min:

        equalized\_image **=** np.full\_like(image, 127, dtype**=**np.uint8)  # Set to mid-gray

**else**:

        cdf\_normalized **=** (cdf **-** cdf\_min) **\*** 255 **/** (cdf\_max **-** cdf\_min)

        equalized\_image **=** cdf\_normalized[image]

**return** equalized\_image.astype('uint8')

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filenames **=** [

        "Fig0320(1)(top\_left).tif",

        "Fig0320(2)(2nd\_from\_top).tif",

        "Fig0320(3)(third\_from\_top).tif",

        "Fig0320(4)(bottom\_left).tif"

    ]

**for** filename **in** image\_filenames:

        image\_path **=** os.path.join(input\_dir, filename)

        base\_name **=** os.path.splitext(filename)[0]

        original\_image **=** imageio.imread(image\_path)

        equalized\_image **=** histogram\_equalization(original\_image)

        original\_hist **=** calculate\_histogram(original\_image)

        equalized\_hist **=** calculate\_histogram(equalized\_image)

        # Visualization

        plt.figure(figsize**=**(12, 10))

        plt.suptitle(f'Histogram Equalization for {filename}', fontsize**=**16)

        plt.subplot(2, 2, 1)

        plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

        plt.title('Original Image')

        plt.axis('off')

        plt.subplot(2, 2, 2)

        plt.plot(original\_hist, color**=**'black')

        plt.title('Original Histogram')

        plt.xlabel('Gray Level')

        plt.ylabel('Pixel Count')

        plt.xlim([0, 255])

        plt.grid(True, linestyle**=**'--', alpha**=**0.6)

        plt.subplot(2, 2, 3)

        plt.imshow(equalized\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

        plt.title('Equalized Image')

        plt.axis('off')

        plt.subplot(2, 2, 4)

        plt.plot(equalized\_hist, color**=**'black')

        plt.title('Equalized Histogram')

        plt.xlabel('Gray Level')

        plt.ylabel('Pixel Count')

        plt.xlim([0, 255])

        plt.grid(True, linestyle**=**'--', alpha**=**0.6)

        plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

        combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_equalization\_results.png")

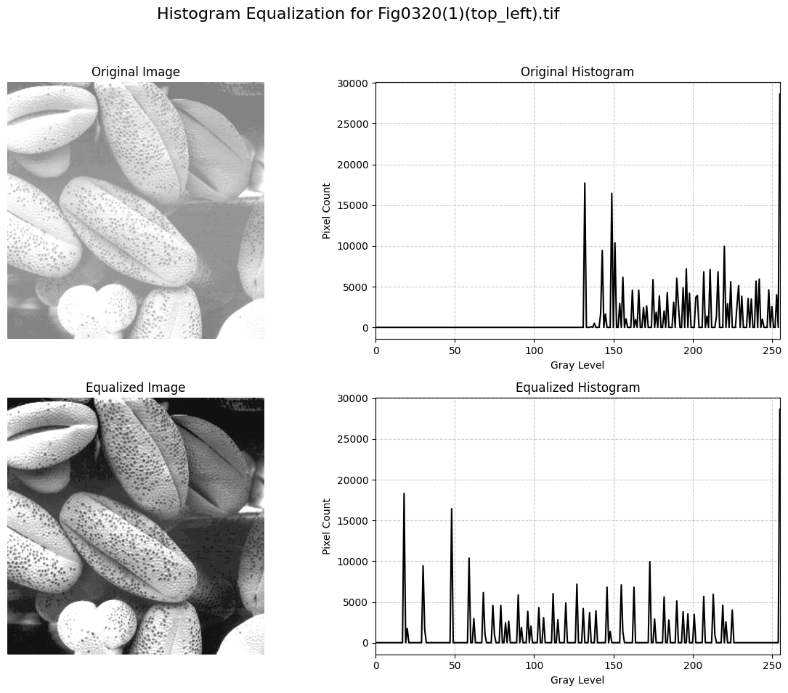
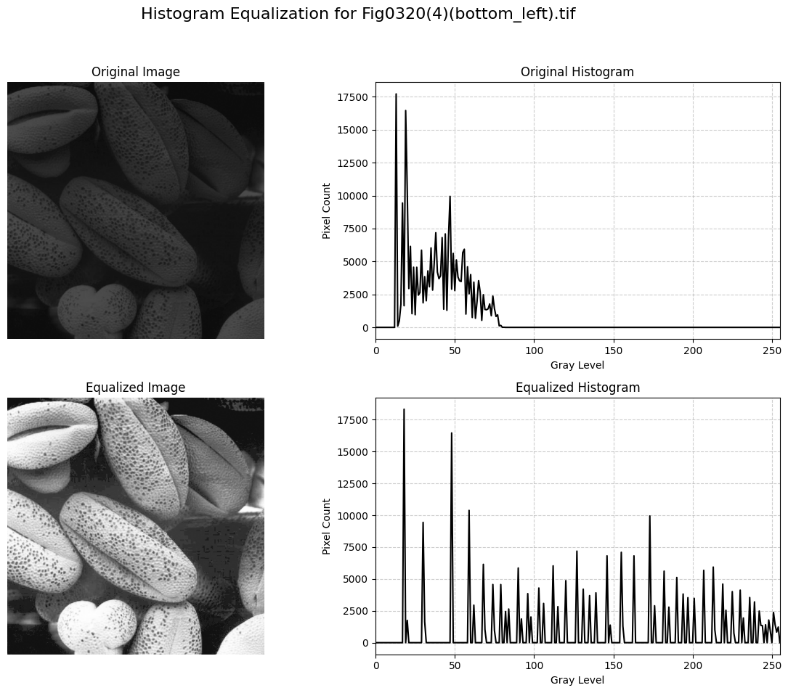
        plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

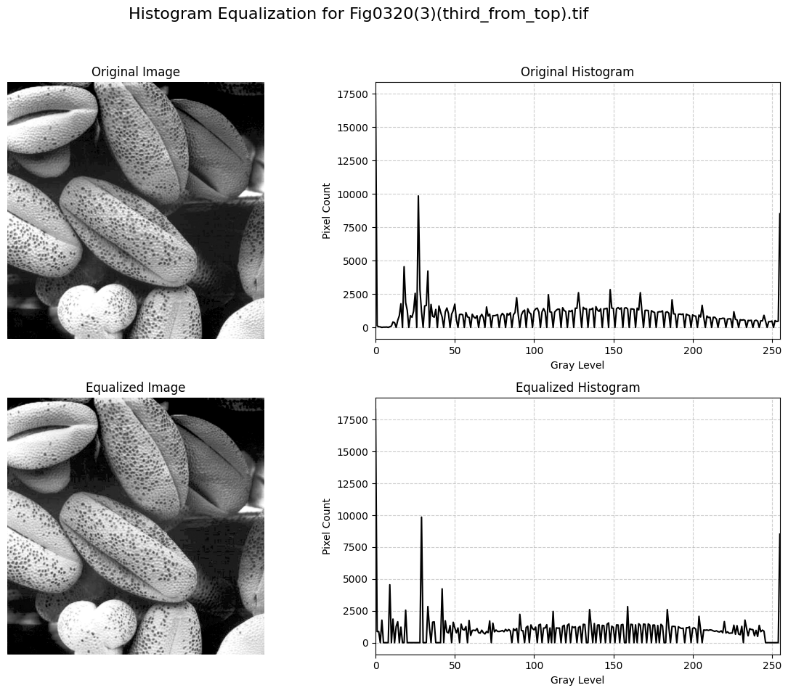
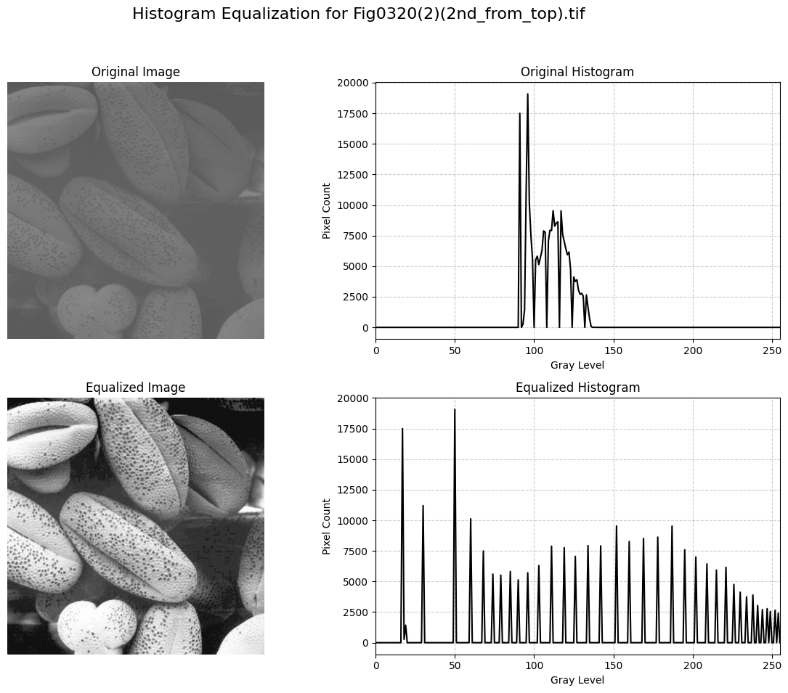
        plt.close()

        equalized\_image\_path **=** os.path.join(output\_dir, f"{base\_name}\_equalized.tif")

        imageio.imwrite(equalized\_image\_path, equalized\_image)

Output Images:





**Homework: Implement Figure 3.24 and 3.25 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**def** calculate\_histogram(image, bins**=**256):

    """

    Calculates the histogram of an 8-bit grayscale image

    """

    hist, \_ **=** np.histogram(image.flatten(), bins, [0, bins])

**return** hist

**def** histogram\_equalization(image):

    """

    Performs histogram equalization on an 8-bit grayscale image

    """

    hist **=** calculate\_histogram(image)

    cdf **=** hist.cumsum()

    cdf\_min **=** cdf.min()

    cdf\_max **=** cdf.max()

**if** cdf\_max **==** cdf\_min:

        equalized\_image **=** np.full\_like(image, 127, dtype**=**np.uint8)

        transform\_func **=** np.full(256, 127, dtype**=**np.uint8)

**else**:

        transform\_func **=** (cdf **-** cdf\_min) **\*** 255 **/** (cdf\_max **-** cdf\_min)

        transform\_func **=** transform\_func.astype('uint8')

        equalized\_image **=** transform\_func[image]

**return** equalized\_image, transform\_func

**def** histogram\_specification(input\_image, target\_histogram):

    """

    Performs histogram specification (matching) on an 8-bit grayscale image

    """

    input\_hist **=** calculate\_histogram(input\_image)

    input\_cdf **=** input\_hist.cumsum()

    input\_cdf\_normalized **=** input\_cdf **/** input\_cdf[**-**1]  # Normalize to [0, 1]

    target\_cdf **=** target\_histogram.cumsum()

    target\_cdf\_normalized **=** target\_cdf **/** target\_cdf[**-**1]  # Normalize to [0, 1]

    lookup\_table **=** np.zeros(256, dtype**=**np.uint8)

**for** i **in** range(256):

        j **=** np.searchsorted(target\_cdf\_normalized, input\_cdf\_normalized[i], side**=**'left')

        lookup\_table[i] **=** min(j, 255)

    specified\_image **=** lookup\_table[input\_image]

**return** specified\_image.astype('uint8'), lookup\_table

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0324.png"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    equalized\_image, equalization\_transform\_func **=** histogram\_equalization(original\_image)

    target\_histogram **=** np.zeros(256, dtype**=**float)

**for** i **in** range(256):

        target\_histogram[i] **=** np.exp(**-**i **/** 10.0)

    target\_histogram[0:10] **\*=** 5

    target\_histogram **=** target\_histogram **/** target\_histogram.sum() **\*** original\_image.size

    specified\_image, specification\_lookup\_table **=** histogram\_specification(original\_image, target\_histogram)

    original\_hist **=** calculate\_histogram(original\_image)

    equalized\_hist **=** calculate\_histogram(equalized\_image)

    specified\_hist **=** calculate\_histogram(specified\_image)

    # Visualization

    plt.figure(figsize**=**(18, 15))  # Adjust figure size for better display

    plt.subplot(3, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    plt.subplot(3, 3, 2)

    plt.plot(original\_hist, color**=**'black')

    plt.title('b) Original Histogram')

    plt.xlabel('Gray Level')

    plt.ylabel('Pixel Count')

    plt.xlim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.subplot(3, 3, 3)

    plt.plot(target\_histogram, color**=**'red')

    plt.title('Target Histogram $p\_z(z)$')

    plt.xlabel('Gray Level')

    plt.ylabel('Pixel Count')

    plt.xlim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.subplot(3, 3, 4)

    plt.imshow(equalized\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('c) Equalized Image')

    plt.axis('off')

    plt.subplot(3, 3, 5)

    plt.plot(equalized\_hist, color**=**'black')

    plt.title('d) Equalized Histogram')

    plt.xlabel('Gray Level')

    plt.ylabel('Pixel Count')

    plt.xlim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.subplot(3, 3, 6)

    plt.plot(np.arange(256), equalization\_transform\_func, color**=**'blue')

    plt.plot([0, 255], [0, 255], 'k--', alpha**=**0.7, label**=**'Identity')

    plt.title('e) Equalization Transform $T(r)$')

    plt.xlabel('Input Gray Level $r$')

    plt.ylabel('Output Gray Level $s$')

    plt.xlim([0, 255])

    plt.ylim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.legend()

    plt.subplot(3, 3, 7)

    plt.imshow(specified\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('f) Specified Image')

    plt.axis('off')

    plt.subplot(3, 3, 8)

    plt.plot(specified\_hist, color**=**'black')

    plt.title('g) Specified Histogram')

    plt.xlabel('Gray Level')

    plt.ylabel('Pixel Count')

    plt.xlim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.subplot(3, 3, 9)

    target\_cdf\_normalized\_for\_plot **=** target\_histogram.cumsum() **/** target\_histogram.cumsum()[**-**1] **\*** 255

    plt.plot(np.arange(256), target\_cdf\_normalized\_for\_plot, color**=**'green', label**=**'$G(z)$ (Target CDF)')

    plt.plot(np.arange(256), specification\_lookup\_table, color**=**'purple', linestyle**=**'--', label**=**'$G^{-1}(s)$ (Mapping)')

    plt.plot([0, 255], [0, 255], 'k--', alpha**=**0.7, label**=**'Identity')  # Identity line

    plt.title('h) Specification Transforms')

    plt.xlabel('Input Gray Level')

    plt.ylabel('Output Gray Level')

    plt.xlim([0, 255])

    plt.ylim([0, 255])

    plt.grid(True, linestyle**=**'--', alpha**=**0.6)

    plt.legend()

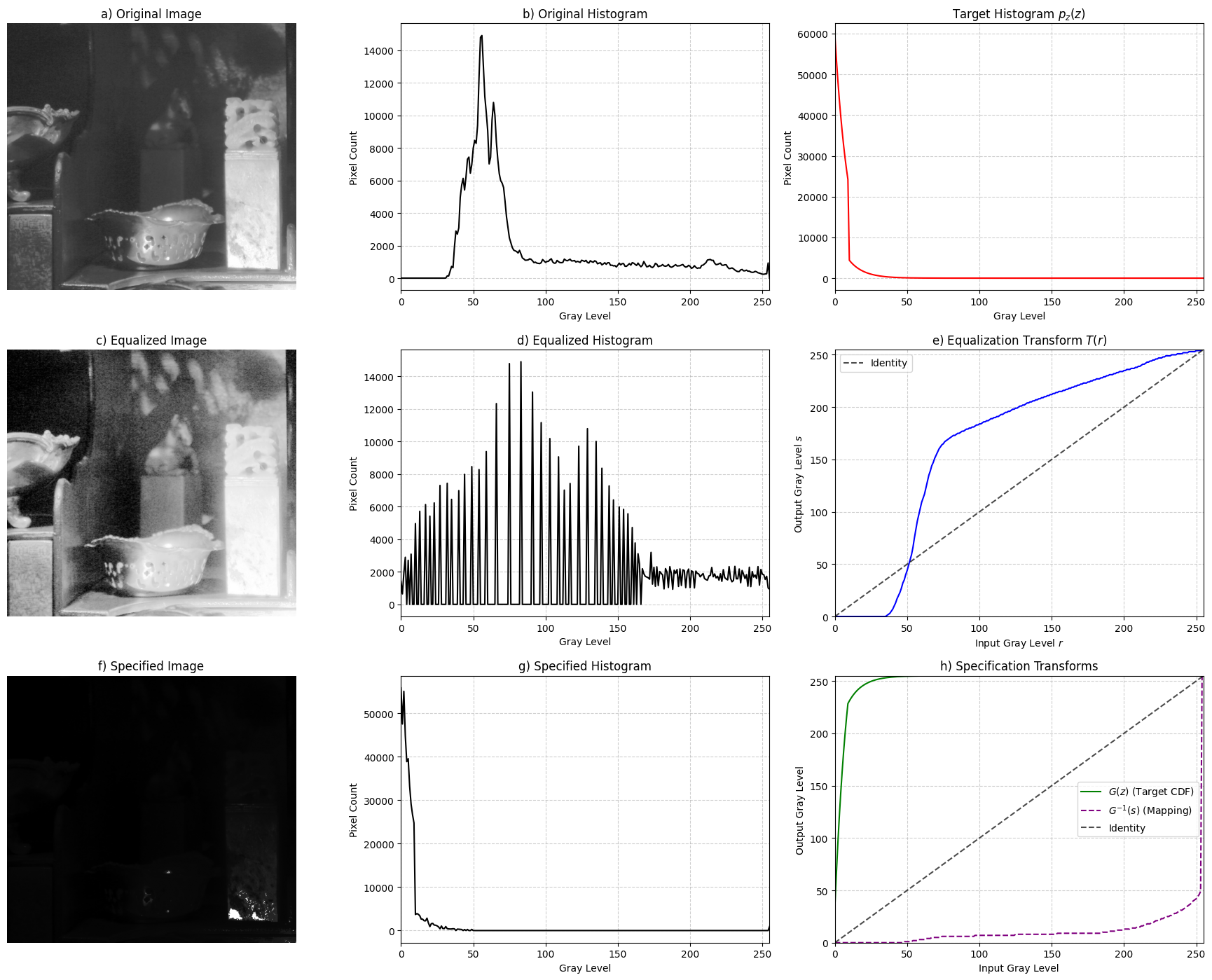
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.97])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_hist\_comparison\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images:



**Homework: Implement Figure 3.26 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**def** histogram\_equalization\_global(image):

    """

    Performs global histogram equalization on an 8-bit grayscale image

    """

    hist, \_ **=** np.histogram(image.flatten(), bins**=**256, range**=**[0, 256])

    cdf **=** hist.cumsum()

    cdf\_min **=** cdf.min()

    cdf\_max **=** cdf.max()

**if** cdf\_max **==** cdf\_min:

        equalized\_image **=** np.full\_like(image, 127, dtype**=**np.uint8)

**else**:

        transform\_func **=** (cdf **-** cdf\_min) **\*** 255 **/** (cdf\_max **-** cdf\_min)

        transform\_func **=** transform\_func.astype('uint8')

        equalized\_image **=** transform\_func[image]

**return** equalized\_image

**def** local\_histogram\_equalization(image, kernel\_size**=**3):

    """

    Performs local histogram equalization on an 8-bit grayscale image

    """

    H, W **=** image.shape

    output\_image **=** np.zeros\_like(image, dtype**=**np.uint8)

    pad\_size **=** kernel\_size **//** 2

    padded\_image **=** np.pad(image, pad\_size, mode**=**'edge')

**for** r **in** range(H):

**for** c **in** range(W):

            neighborhood **=** padded\_image[r: r **+** kernel\_size, c: c **+** kernel\_size]

            local\_hist, \_ **=** np.histogram(neighborhood.flatten(), bins**=**256, range**=**[0, 256])

            local\_cdf **=** local\_hist.cumsum()

            cdf\_min **=** local\_cdf.min()

            cdf\_max **=** local\_cdf.max()

**if** cdf\_max **==** cdf\_min:

                output\_image[r, c] **=** image[r, c]

**else**:

                pixel\_value **=** image[r, c]

                s **=** (local\_cdf[pixel\_value] **-** cdf\_min) **\*** 255 **/** (cdf\_max **-** cdf\_min)

                output\_image[r, c] **=** np.clip(s, 0, 255).astype(np.uint8)

**return** output\_image

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0326(a)(embedded\_square\_noisy\_512).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image**=** imageio.imread(image\_path)

    global\_equalized\_image **=** histogram\_equalization\_global(original\_image)

    print("Performed global histogram equalization.")

    kernel\_size **=** 3

    local\_equalized\_image **=** local\_histogram\_equalization(original\_image, kernel\_size**=**kernel\_size)

    # Visualization

    plt.figure(figsize**=**(18, 6))

    plt.suptitle(f'Histogram Equalization Comparison for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    plt.subplot(1, 3, 2)

    plt.imshow(global\_equalized\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Global Histogram Equalization')

    plt.axis('off')

    plt.subplot(1, 3, 3)

    plt.imshow(local\_equalized\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'c) Local Histogram Equalization ({kernel\_size}x{kernel\_size} neighborhood)')

    plt.axis('off')

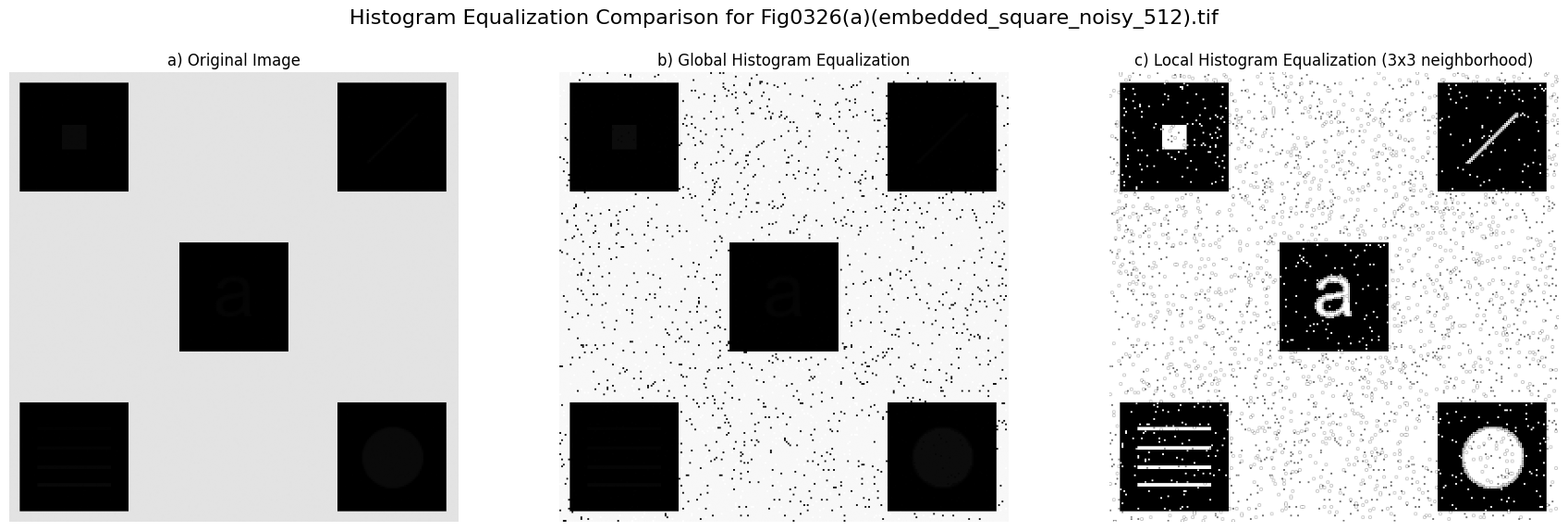
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_hist\_eq\_comparison.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images:



**Homework: Implement Figure 3.27 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**def** local\_contrast\_enhancement(image, kernel\_size**=**3, k0**=**0.4, k1**=**0.02, E**=**4.0):

    """

    Performs local contrast enhancement based on local mean and standard deviation

    """

    H, W **=** image.shape

    output\_image **=** np.zeros\_like(image, dtype**=**np.uint8)

    pad\_size **=** kernel\_size **//** 2

    padded\_image **=** np.pad(image, pad\_size, mode**=**'edge')

    M **=** np.mean(image)

    sigma **=** np.std(image)

    image\_float **=** image.astype(float)

    padded\_image\_float **=** padded\_image.astype(float)

**for** r **in** range(H):

**for** c **in** range(W):

            neighborhood **=** padded\_image\_float[r: r **+** kernel\_size, c: c **+** kernel\_size]

            m\_xy **=** np.mean(neighborhood)

            sigma\_xy **=** np.std(neighborhood)

            f\_xy **=** image\_float[r, c]

**if** m\_xy <**=** k0 **\*** M **and** sigma\_xy <**=** k1 **\*** sigma:

                g\_xy **=** E **\*** f\_xy

**else**:

                g\_xy **=** f\_xy

            output\_image[r, c] **=** np.clip(g\_xy, 0, 255).astype(np.uint8)

**return** output\_image

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0326(a)(embedded\_square\_noisy\_512).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    kernel\_size **=** 3

    k0 **=** 0.4

    k1 **=** 0.02

    E **=** 4.0

    # Visualization

    plt.figure(figsize**=**(12, 6))  # Adjust figure size for 1x2 layout

    plt.suptitle(f'Local Contrast Enhancement (Example 3.10) for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(enhanced\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'b) Enhanced Image (k0={k0}, k1={k1}, E={E})')

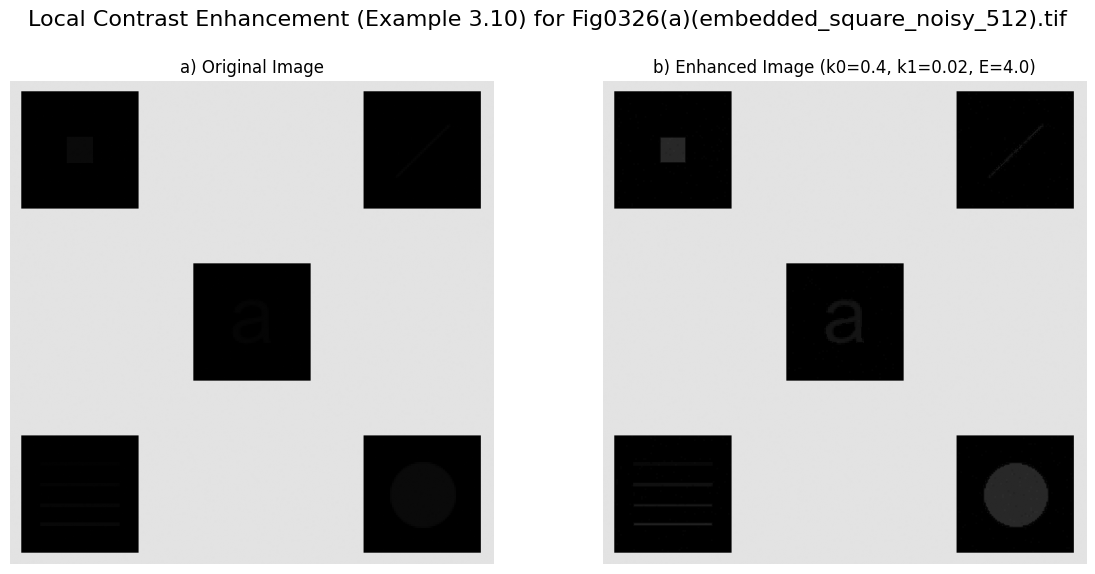
    plt.axis('off')

    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])  # Adjust layout to prevent title overlap

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_local\_enhancement\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

Output Images:



**Homework: Implement Figure 3.33 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**from** scipy.ndimage **import** convolve

**def** apply\_box\_filter(image, kernel\_size):

    """

    Applies a box (average) filter to the image using zero padding

    """

    kernel **=** np.ones((kernel\_size, kernel\_size), dtype**=**float) **/** (kernel\_size **\*** kernel\_size)

    image\_float **=** image.astype(float)

    filtered\_image\_float **=** convolve(image\_float, kernel, mode**=**'constant', cval**=**0.0)

    filtered\_image **=** np.clip(filtered\_image\_float, 0, 255).astype(np.uint8)

**return** filtered\_image

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0333(a)(test\_pattern\_blurring\_orig).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    kernel\_sizes **=** [3, 11, 21]

    filtered\_images **=** {}

**for** k\_size **in** kernel\_sizes:

        filtered\_img **=** apply\_box\_filter(original\_image, k\_size)

        filtered\_images[k\_size] **=** filtered\_img

    # Visualization

    plt.figure(figsize**=**(12, 12))

    plt.suptitle(f'Box Filtering Results for {image\_filename}', fontsize**=**16)

    plt.subplot(2, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image')

    plt.axis('off')

    plt.subplot(2, 2, 2)

    plt.imshow(filtered\_images[3], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Box Filter (3x3)')

    plt.axis('off')

    plt.subplot(2, 2, 3)

    plt.imshow(filtered\_images[11], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('c) Box Filter (11x11)')

    plt.axis('off')

    plt.subplot(2, 2, 4)

    plt.imshow(filtered\_images[21], cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('d) Box Filter (21x21)')

    plt.axis('off')

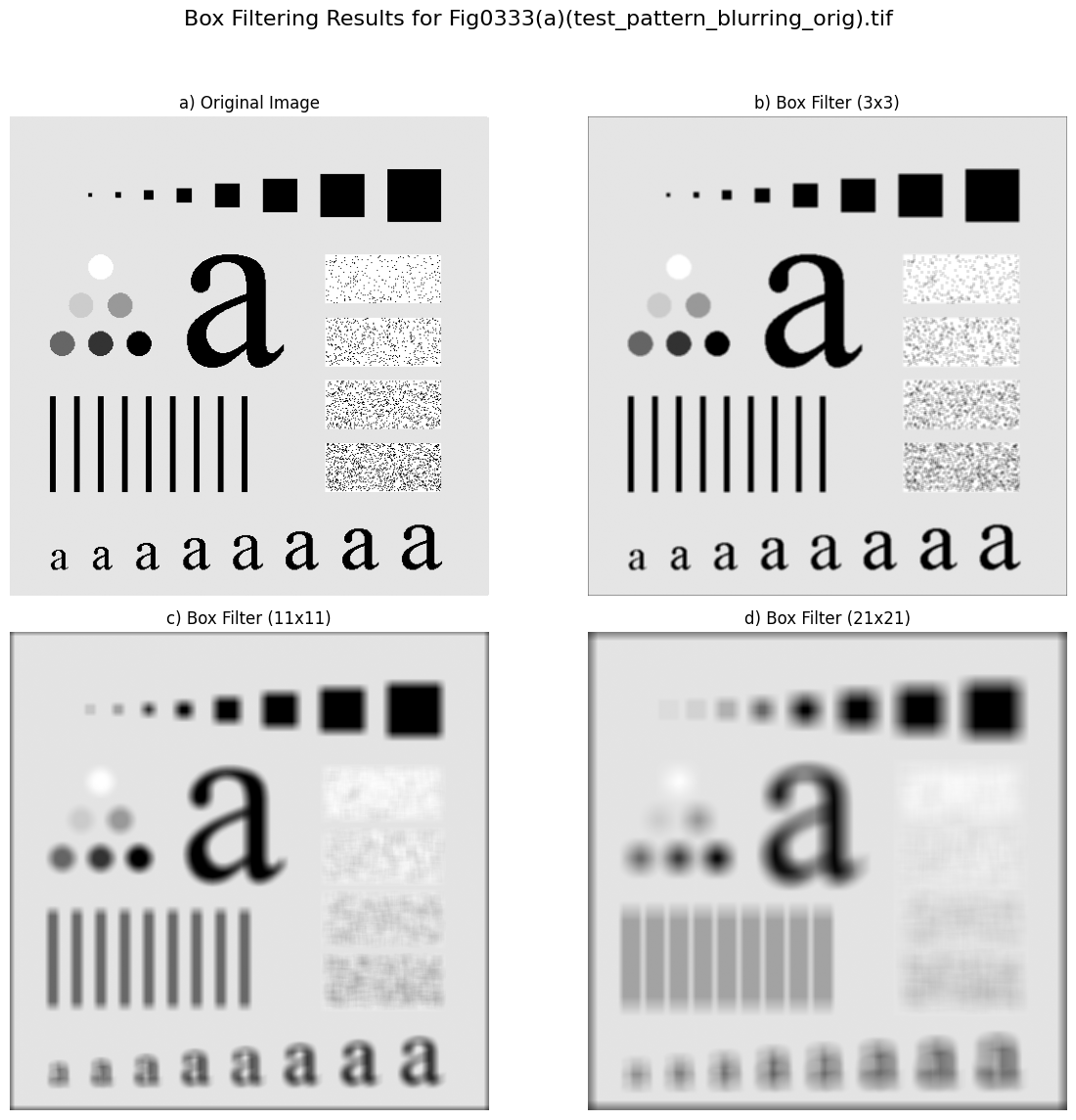
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_box\_filter\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images:



**Homework: Implement Figure 3.33 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**from** scipy.ndimage **import** gaussian\_filter, median\_filter

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0335(a)(ckt\_board\_saltpep\_prob\_pt05).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    gaussian\_sigma **=** 3

    gaussian\_filtered\_image **=** gaussian\_filter(original\_image, sigma**=**gaussian\_sigma)

    gaussian\_filtered\_image **=** np.clip(gaussian\_filtered\_image, 0, 255).astype(np.uint8)

    median\_kernel\_size **=** 7

    median\_filtered\_image **=** median\_filter(original\_image, size**=**median\_kernel\_size)

    # Visualization

    plt.figure(figsize**=**(18, 6))

    plt.suptitle(f'Noise Filtering Comparison for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 3, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Image (Salt-and-Pepper Noise)')

    plt.axis('off')

    plt.subplot(1, 3, 2)

    plt.imshow(gaussian\_filtered\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'b) Gaussian Filter ($\\sigma={gaussian\_sigma}$)')

    plt.axis('off')

    plt.subplot(1, 3, 3)

    plt.imshow(median\_filtered\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title(f'c) Median Filter ({median\_kernel\_size}x{median\_kernel\_size})')

    plt.axis('off')

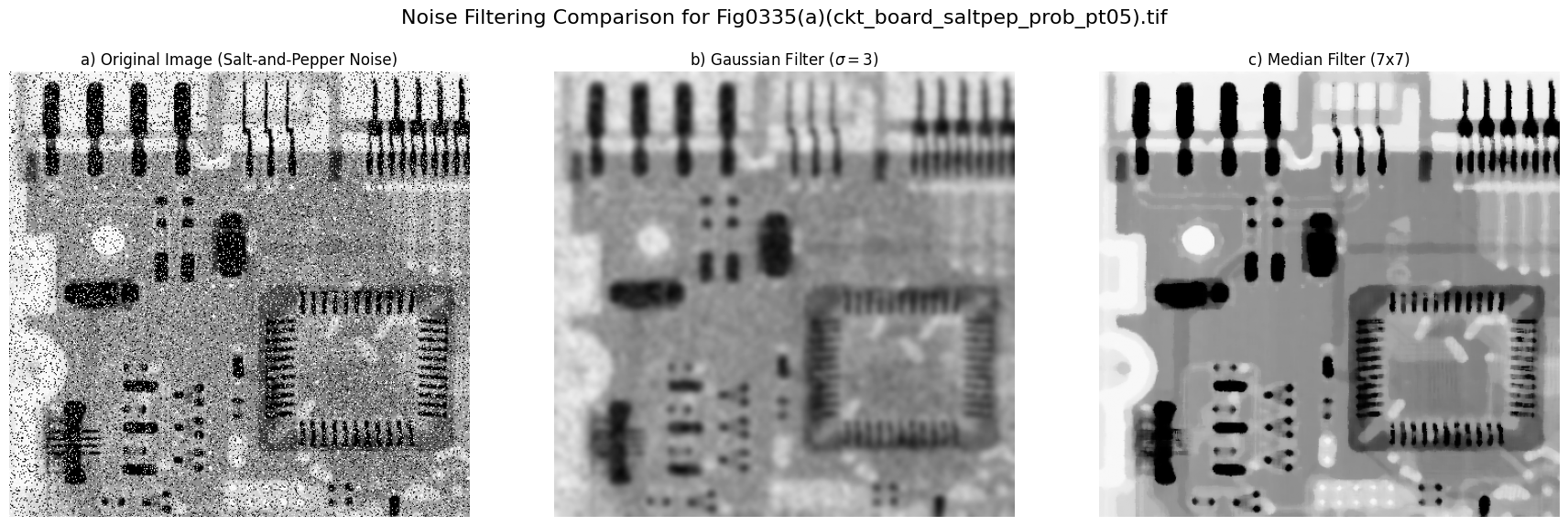
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_noise\_filtering\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images:



**Homework: Implement Figure 3.46 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**from** scipy.ndimage **import** convolve

**def** apply\_laplacian\_sharpening(image, laplacian\_kernel, c**=-**1):

    """

    Applies a Laplacian filter to an image and then sharpens it

    """

    laplacian\_response **=** convolve(image\_float, laplacian\_kernel, mode**=**'nearest', cval**=**0.0)

    laplacian\_image\_display **=** np.clip(laplacian\_response, 0, 255).astype(np.uint8)

    sharpened\_image\_float **=** image\_float **+** c **\*** laplacian\_response

    sharpened\_image **=** np.clip(sharpened\_image\_float, 0, 255).astype(np.uint8)

**return** laplacian\_image\_display, sharpened\_image

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0338(a)(blurry\_moon).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    laplacian\_kernel\_a **=** np.array([

        [0, 1, 0],

        [1, **-**4, 1],

        [0, 1, 0]

    ], dtype**=**float)

    laplacian\_kernel\_b **=** np.array([

        [1, 1, 1],

        [1, **-**8, 1],

        [1, 1, 1]

    ], dtype**=**float)

    laplacian\_img\_a\_display, sharpened\_img\_a **=** apply\_laplacian\_sharpening(original\_image, laplacian\_kernel\_a, c**=-**1)

    laplacian\_img\_b\_display, sharpened\_img\_b **=** apply\_laplacian\_sharpening(original\_image, laplacian\_kernel\_b, c**=-**1)

    # Visualization

    plt.figure(figsize**=**(15, 12))  # Adjust figure size for 2x2 layout

    plt.suptitle(f'Laplacian Sharpening for {image\_filename}', fontsize**=**16)

    plt.subplot(2, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Blurry Moon Image')

    plt.axis('off')

    plt.subplot(2, 2, 2)

    plt.imshow(laplacian\_img\_a\_display, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Laplacian Image (Kernel a, clipped)')

    plt.axis('off')

    plt.subplot(2, 2, 3)

    plt.imshow(sharpened\_img\_a, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('c) Sharpened Image (Kernel a, $c=-1$)')

    plt.axis('off')

    plt.subplot(2, 2, 4)

    plt.imshow(sharpened\_img\_b, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('d) Sharpened Image (Kernel b, $c=-1$)')

    plt.axis('off')

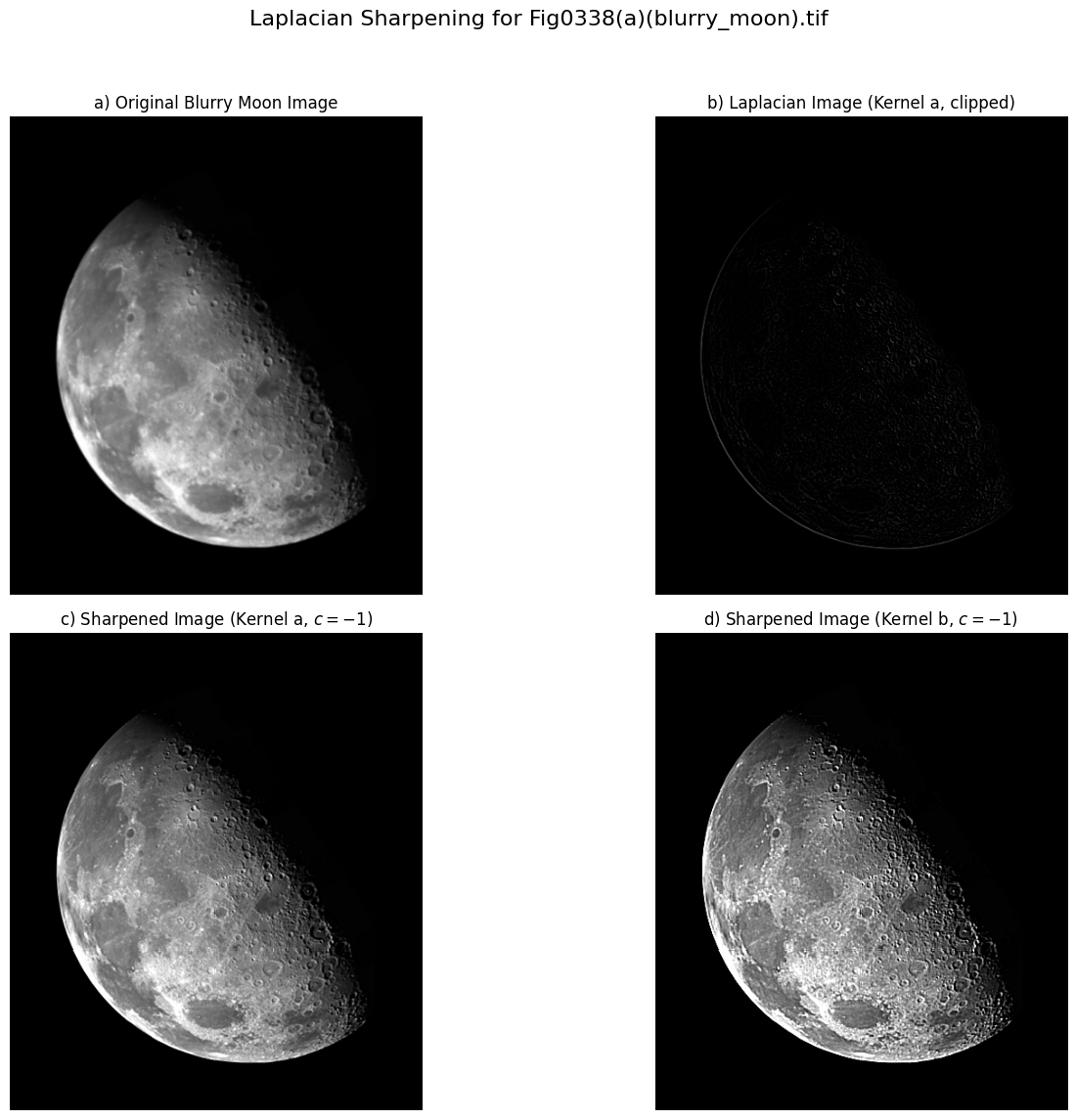
    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95])

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_laplacian\_sharpening\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images:



**Homework: Implement Figure 3.46 by programming with Python**

Python Code:

**import** numpy as np

**import** imageio.v2 as imageio

**import** matplotlib

matplotlib.use('Agg')

**import** matplotlib.pyplot as plt

**import** os

**from** scipy.ndimage **import** convolve

**def** apply\_sobel\_gradient(image):

    """

    Applies Sobel operators to compute the gradient magnitude of an image

    """

    sobel\_x **=** np.array([

        [**-**1, 0, 1],

        [**-**2, 0, 2],

        [**-**1, 0, 1]

    ], dtype**=**float)

    sobel\_y **=** np.array([

        [**-**1, **-**2, **-**1],

        [ 0,  0,  0],

        [ 1,  2,  1]

    ], dtype**=**float)

    image\_float **=** image.astype(float)

    Gx **=** convolve(image\_float, sobel\_x, mode**=**'nearest')

    Gy **=** convolve(image\_float, sobel\_y, mode**=**'nearest')

    gradient\_magnitude **=** np.sqrt(Gx**\*\***2 **+** Gy**\*\***2)

    max\_val **=** np.max(gradient\_magnitude)

**if** max\_val > 0:

        scaled\_gradient\_magnitude **=** (gradient\_magnitude **/** max\_val **\*** 255)

**else**: # Handle case where image is uniform (no edges)

        scaled\_gradient\_magnitude **=** np.zeros\_like(gradient\_magnitude)

    final\_gradient\_image **=** np.clip(scaled\_gradient\_magnitude, 0, 255).astype(np.uint8)

**return** final\_gradient\_image

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    # Define input and output directories

    input\_dir **=** "original\_image"

    output\_dir **=** "output\_image"

    os.makedirs(output\_dir, exist\_ok**=**True)

    image\_filename **=** "Fig0342(a)(contact\_lens\_original).tif"

    image\_path **=** os.path.join(input\_dir, image\_filename)

    base\_name **=** os.path.splitext(image\_filename)[0]

    original\_image **=** imageio.imread(image\_path)

    gradient\_image **=** apply\_sobel\_gradient(original\_image)

    # Visualization

    plt.figure(figsize**=**(12, 6))

    plt.suptitle(f'Gradient Edge Enhancement for {image\_filename}', fontsize**=**16)

    plt.subplot(1, 2, 1)

    plt.imshow(original\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('a) Original Contact Lens Image')

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(gradient\_image, cmap**=**'gray', vmin**=**0, vmax**=**255)

    plt.title('b) Sobel Gradient Magnitude')

    plt.axis('off')

    plt.tight\_layout(rect**=**[0, 0.03, 1, 0.95]) # Adjust layout to prevent title overlap

    combined\_output\_path **=** os.path.join(output\_dir, f"{base\_name}\_gradient\_enhancement\_results.png")

    plt.savefig(combined\_output\_path, bbox\_inches**=**'tight')

    plt.close()

Output Images”

