符昕宇 202364810311

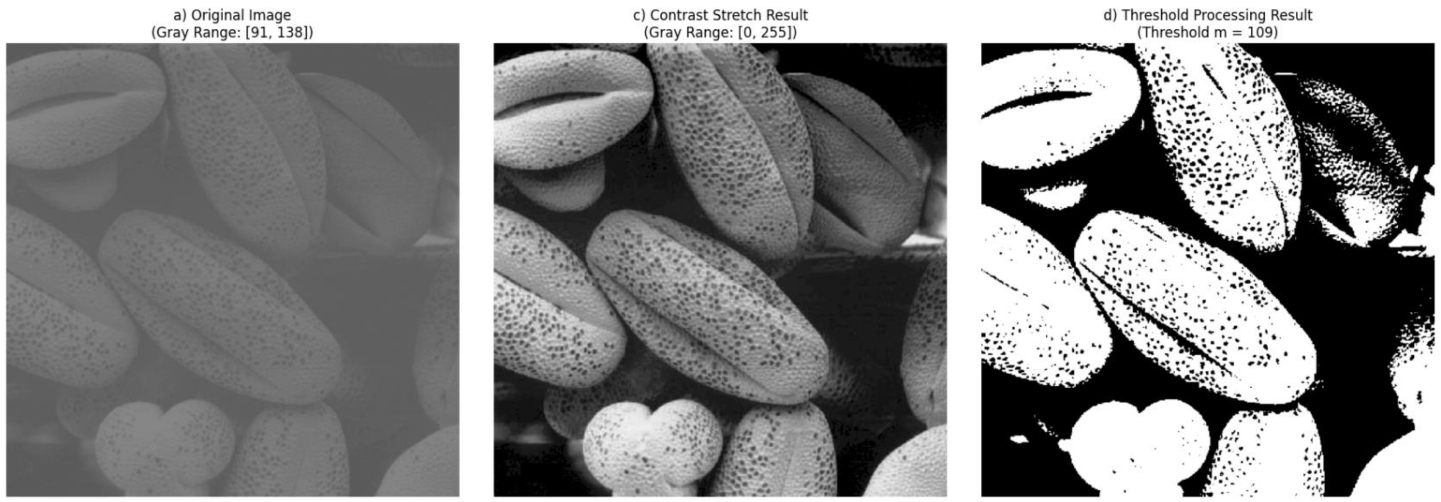
Homework: Implement Figure 3.10 by programming with Python

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))
```

```python
    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:

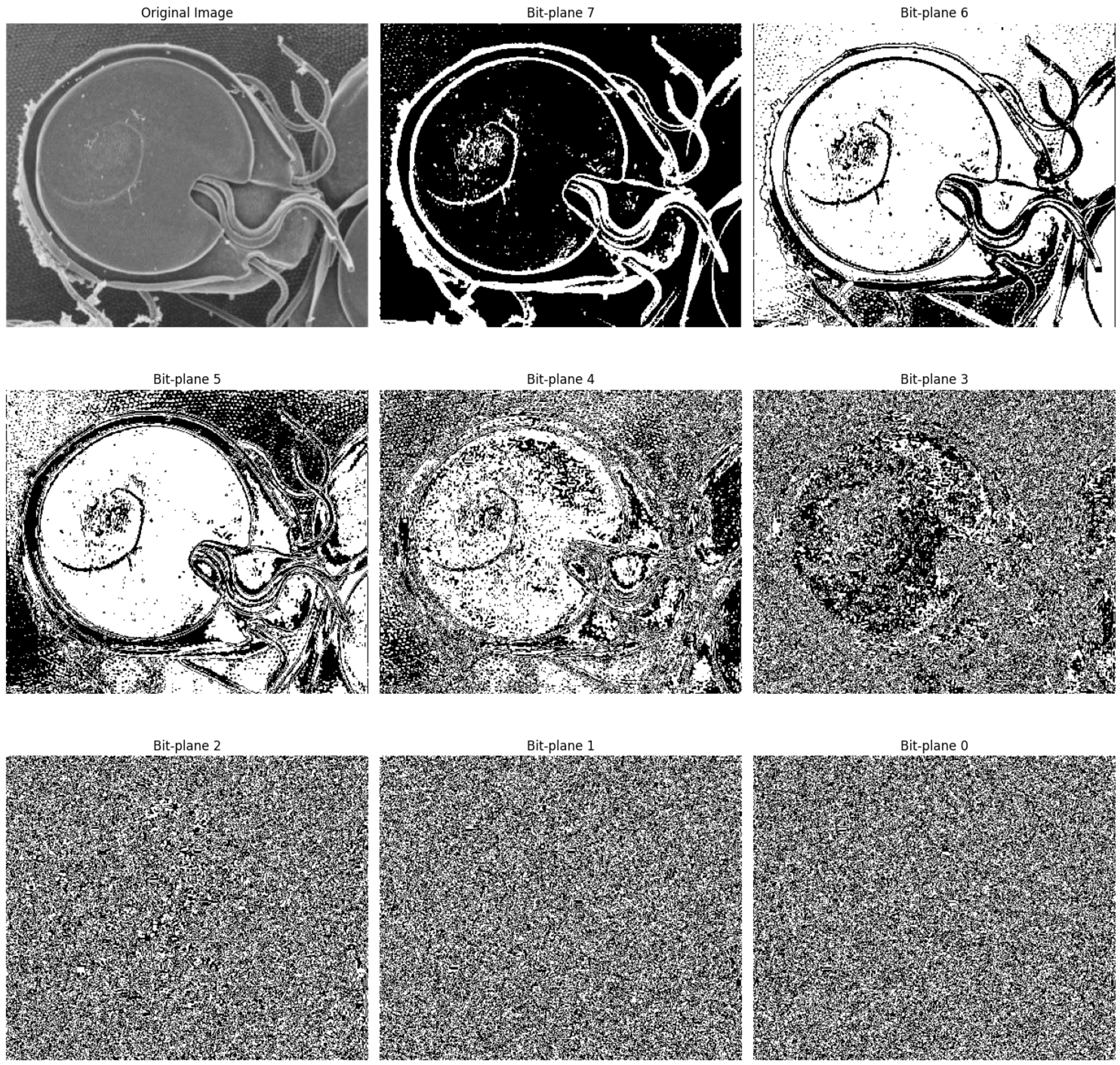


Homework: Implement Figure 3.14 by programming with Python

Python Code:

```python
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:



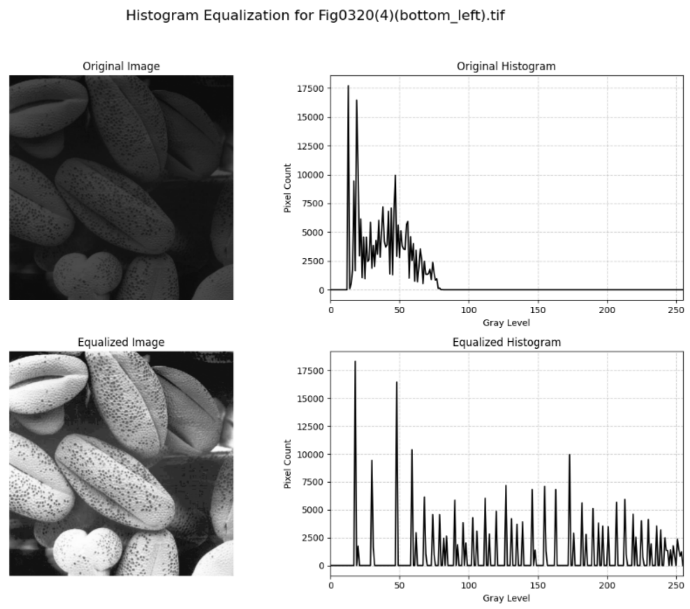
Homework: Implement Figure 3.20 by programming with Python

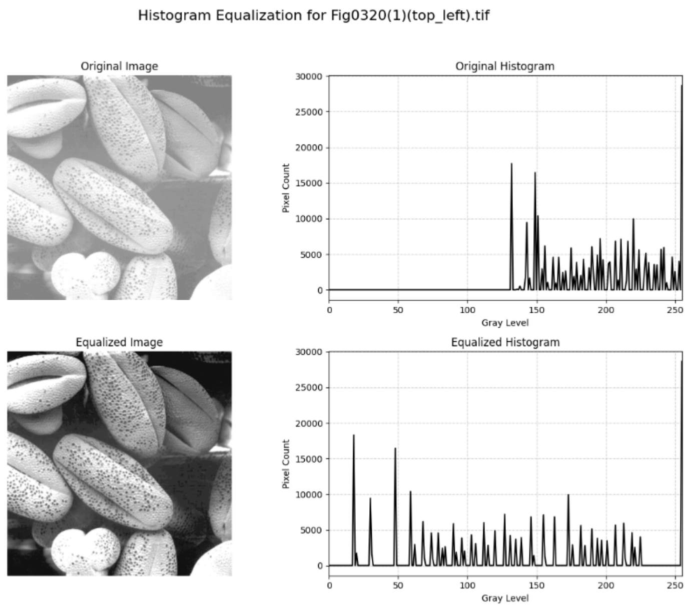
Python Code:

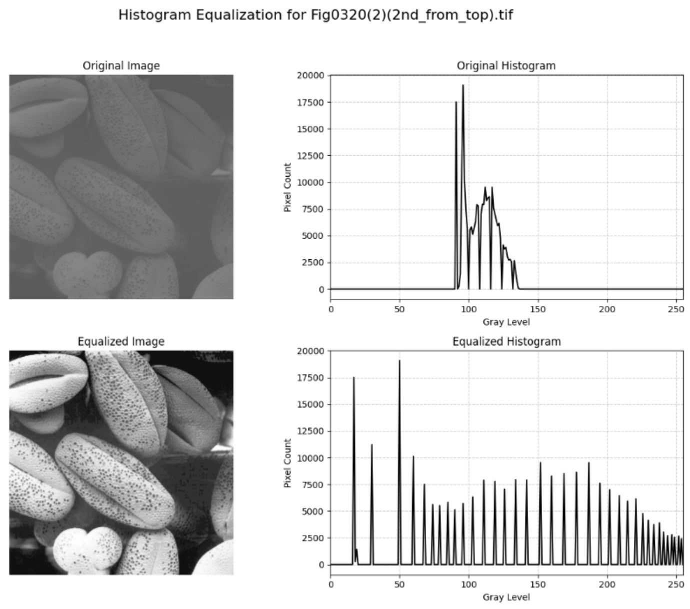
```
import numpy as np
import imageio.v2 as imageio
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import os


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)
    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"
    ]
```

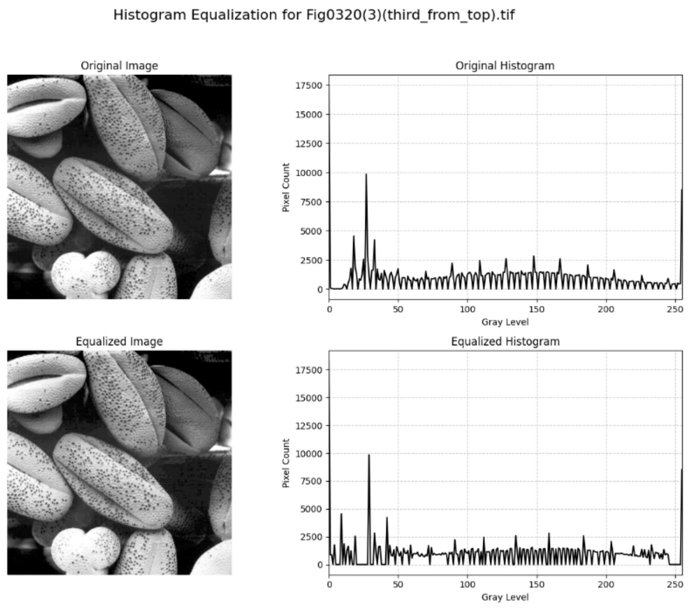
```python
for filename in image_filenames:
    image_path = os.path.join(input_dir, filename)
    base_name = os.path.splitext(filename)[0]

    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:









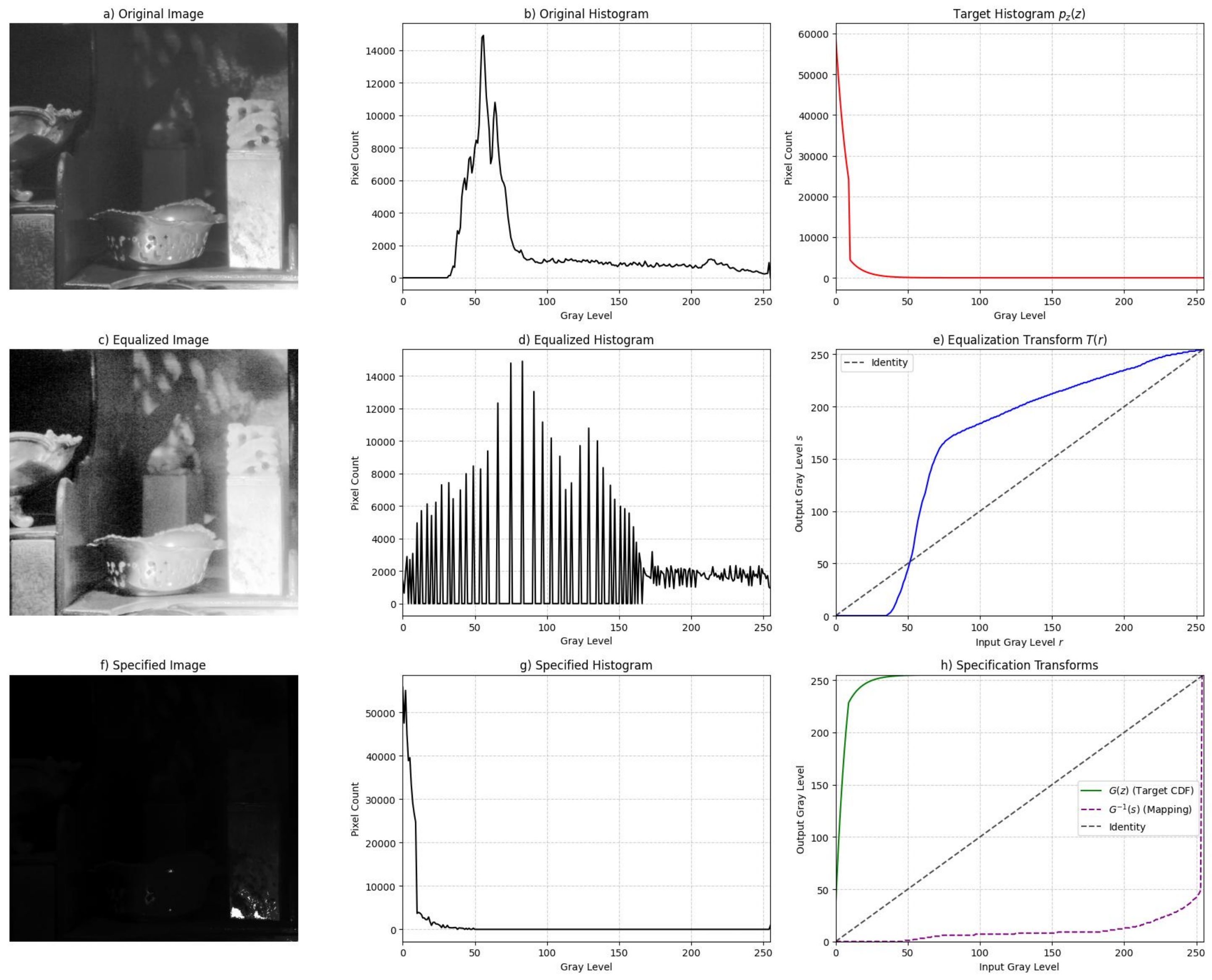
Homework: Implement Figure 3.24 and 3.25 by programming with Python

Python Code:

```python
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]
    target_cdf = target_histogram.cumsum()
    target_cdf_normalized = target_cdf / target_cdf[-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))

    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')
    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:

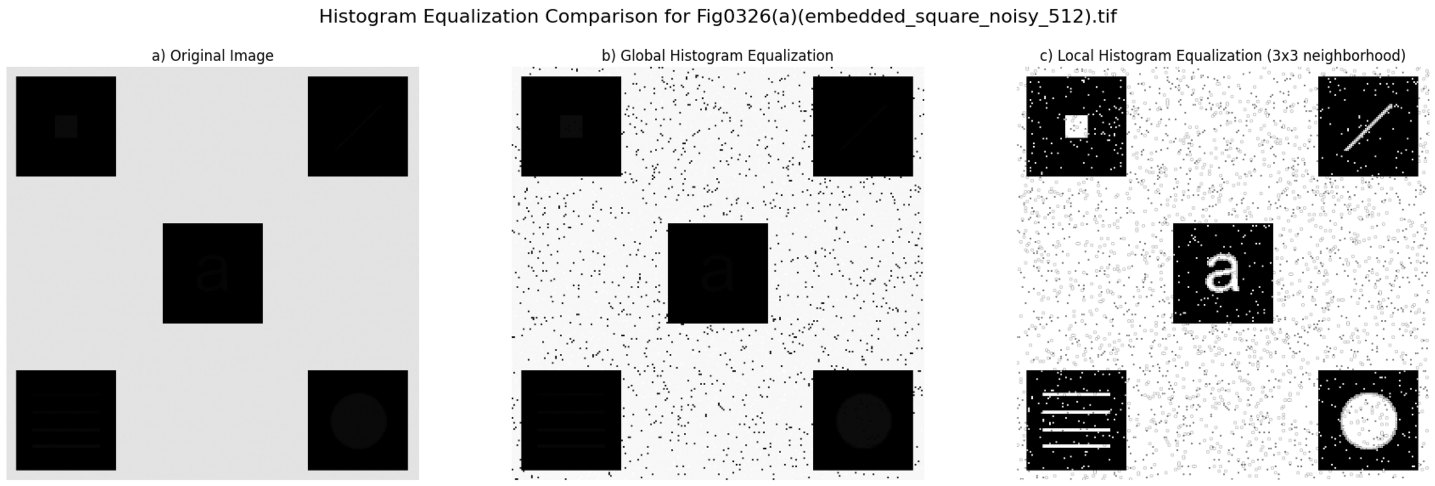


Homework: Implement Figure 3.26 by programming with Python

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:

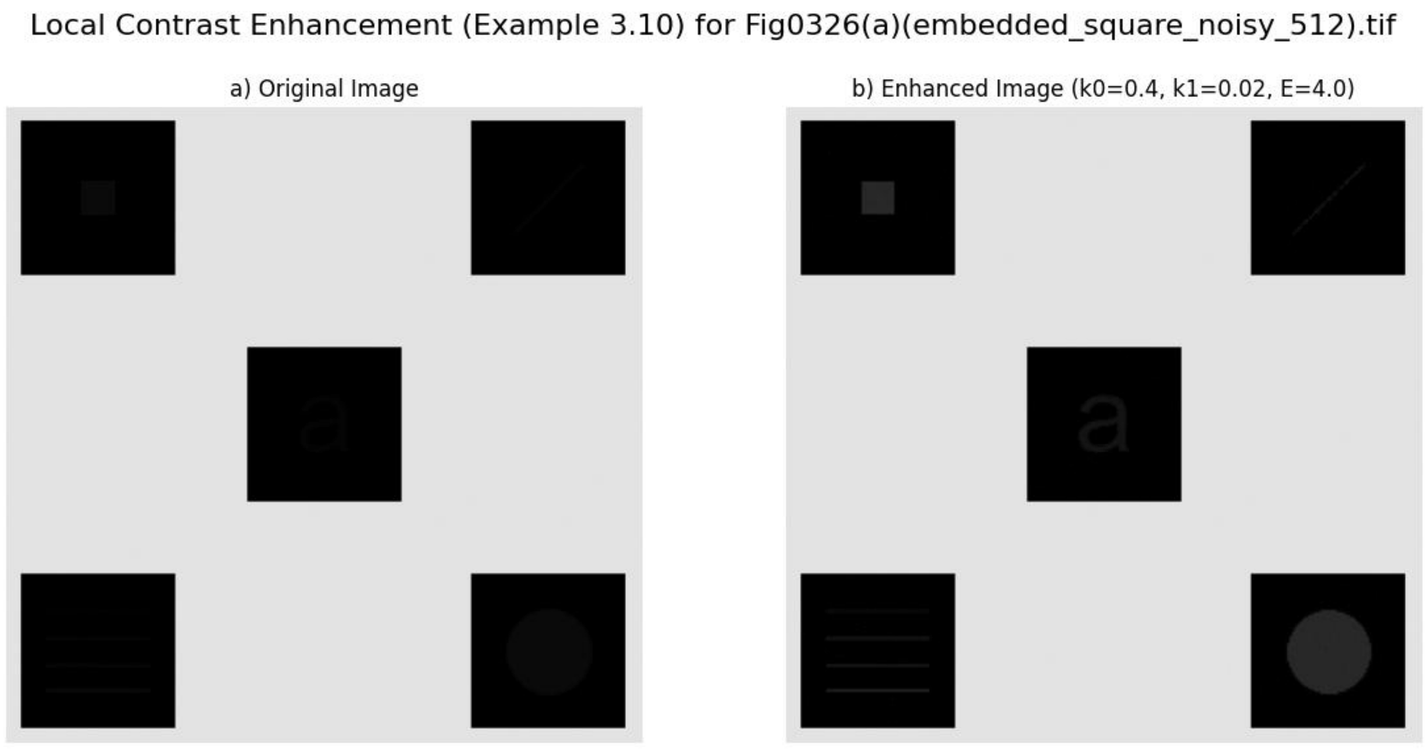

Histogram Equalization Comparison for Fig0326(a)(embedded_square_noisy_512).tif
a) Original Image
b) Global Histogram Equalization
c) Local Histogram Equalization (3x3 neighborhood)

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))
    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])
    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:


Local Contrast Enhancement (Example 3.10) for Fig0326(a)(embedded_square_noisy_512).tif
a) Original Image
b) Enhanced Image (k0=0.4, k1=0.02, E=4.0)
a

Homework: Implement Figure 3.33 by programming with Python

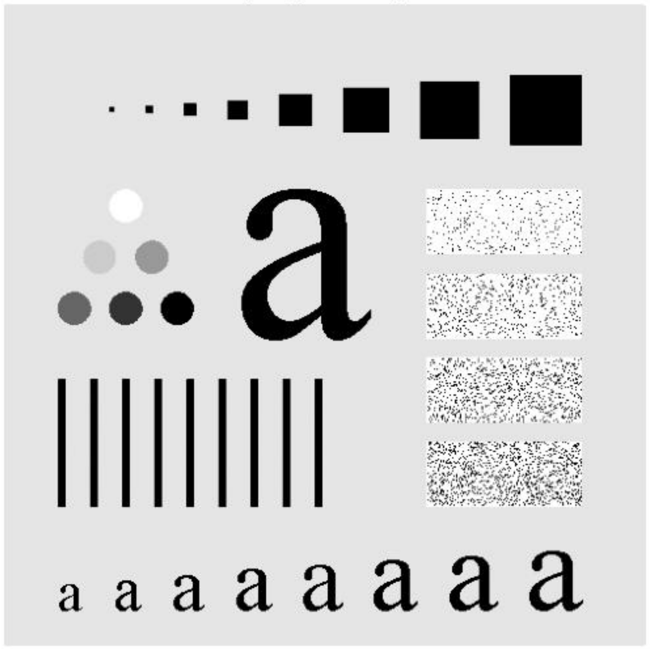
Python Code:

```python
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()
```

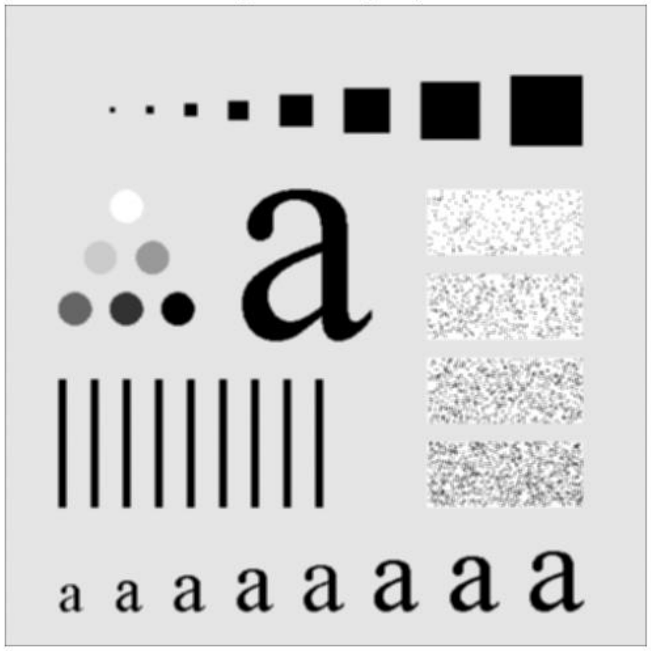
Output Images:

Box Filtering Results for Fig0333(a)(test_pattern_blurring_orig).tif

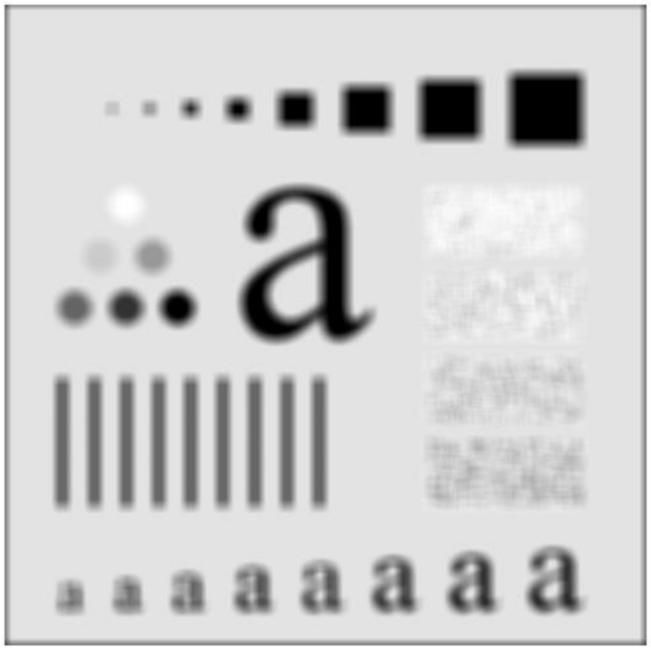
a) Original Image



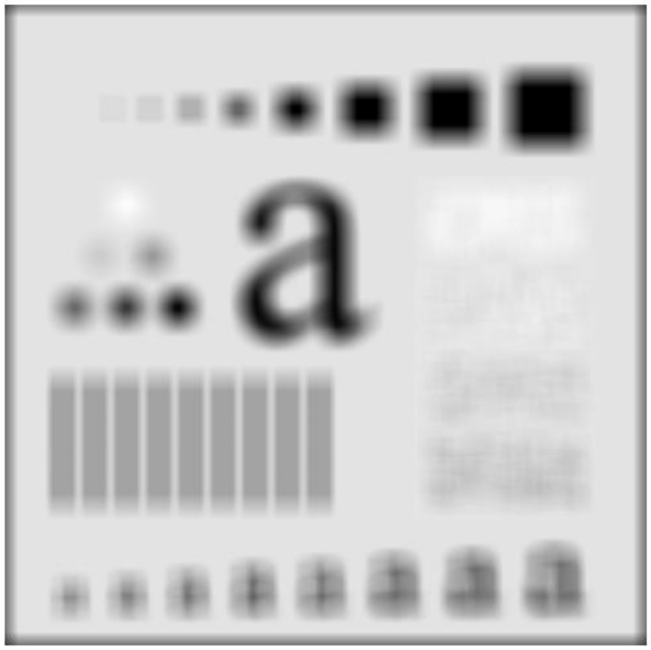
b) Box Filter (3x3)



c) Box Filter (11x11)



d) Box Filter (21x21)

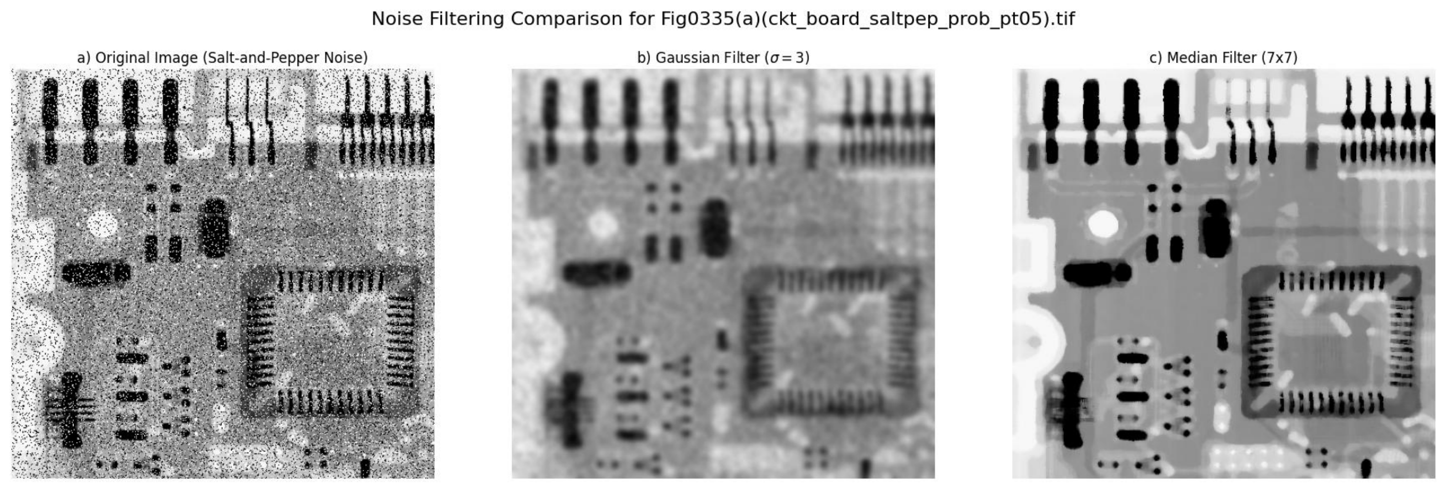


Homework: Implement Figure 3.33 by programming with Python

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:


Noise Filtering Comparison for Fig0335(a)(ckt_board_saltpep_prob_pt05).tif
a) Original Image (Salt-and-Pepper Noise)
b) Gaussian Filter ($\sigma = 3$)
c) Median Filter (7x7)