computer-vision-1

February 25, 2024

0.0.1 CV Assignment 4

0.0.2 Varun Agrawal

0.0.3 MDS202251

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

[2]:
```
### Q!
def display_colored_channel_histogram(image, tit):
    """Display colored histogram for each color channel of an image."""
    plt.figure(figsize=(12, 6))
    colors = ('b', 'g', 'r')
    for i, color in enumerate(colors):
        channel_hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(channel_hist, color=color)
        plt.xlim([0, 256])
    plt.title(f'Color Histogram (BGR) for {tit}')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()

def display_channel_histogram(image, tit):
    """Display histogram for each color channel of an image."""
    plt.figure(figsize=(12, 6))
    colors = ('b', 'g', 'r')
    for i, color in enumerate(colors):
        plt.subplot(2, 3, i + 1)
        plt.hist(image[:, :, i].ravel(), bins=256, color=color, alpha=0.5)
        plt.title(f'Channel {color.upper()} for {tit}')
        plt.xlim(0, 255)
    plt.tight_layout()
    plt.show()
```

[3]:
```
def equalize_intensity_counts(counts):
    """Equalize intensity counts."""
```

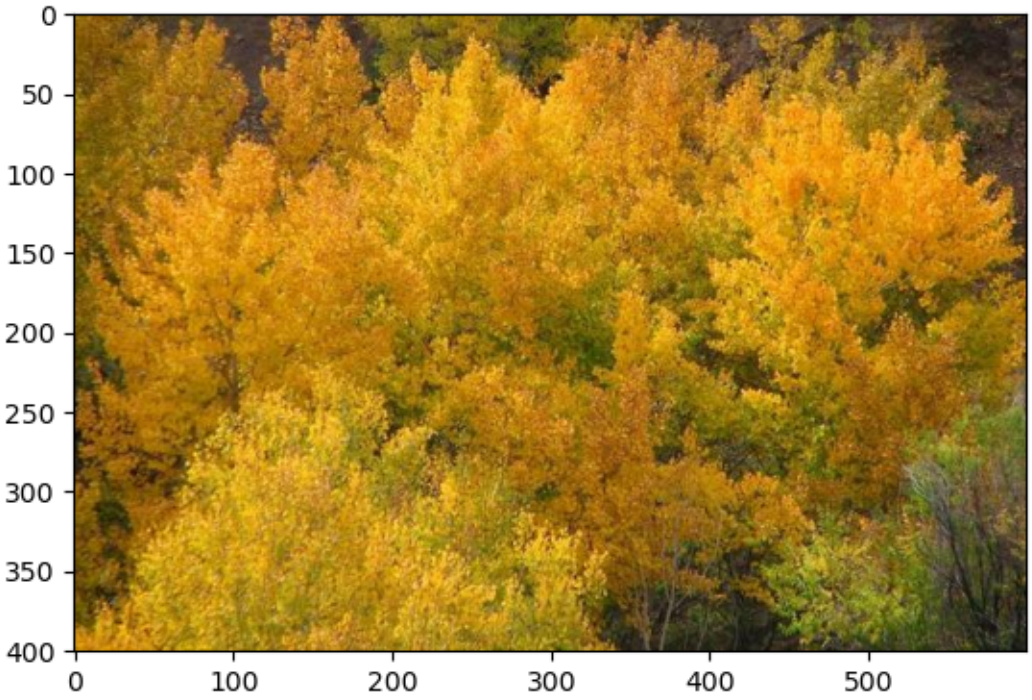
```
    total_pixels = sum(counts)
    sk = [round((cumulative / total_pixels) * 255) for cumulative in np.
↪cumsum(counts)]
    return sk
```

[4]:
```
def equalize_channel(channel, sk):
    """Equalize channel using specified sk values."""
    h, w = channel.shape
    equalized_channel = np.zeros_like(channel)
    for i in range(h):
        for j in range(w):
            equalized_channel[i, j] = sk[channel[i, j]]
    return equalized_channel
```

[5]:
```
def histogram_matching(input_image_path, reference_image_path):
    """Perform histogram matching from input image to reference image."""
    input_image = cv2.imread(input_image_path)
    reference_image = cv2.imread(reference_image_path)


    # Equalize intensity counts for each channel of both input and reference␣
↪images
    input_counts = [cv2.calcHist([input_image], [i], None, [256], [0, 256]).
↪flatten() for i in range(3)]
    reference_counts = [cv2.calcHist([reference_image], [i], None, [256], [0,␣
↪256]).flatten() for i in range(3)]

    input_eq_counts = [equalize_intensity_counts(counts) for counts in␣
↪input_counts]
    reference_eq_counts = [equalize_intensity_counts(counts) for counts in␣
↪reference_counts]

    # Perform histogram matching
    final_sk_list = []
    for i in range(3):
        temp_sk = []
        for j in range(256):
            temp_array = [x for x in reference_eq_counts[i] if x >=␣
↪input_eq_counts[i][j]]
            if temp_array:
                new_sk = min(temp_array)
            else:
                new_sk = 255
            new_zk = np.where(np.array(reference_eq_counts[i]) == new_sk)[0][0]
            temp_sk.append(new_zk)
        final_sk_list.append(temp_sk)
```

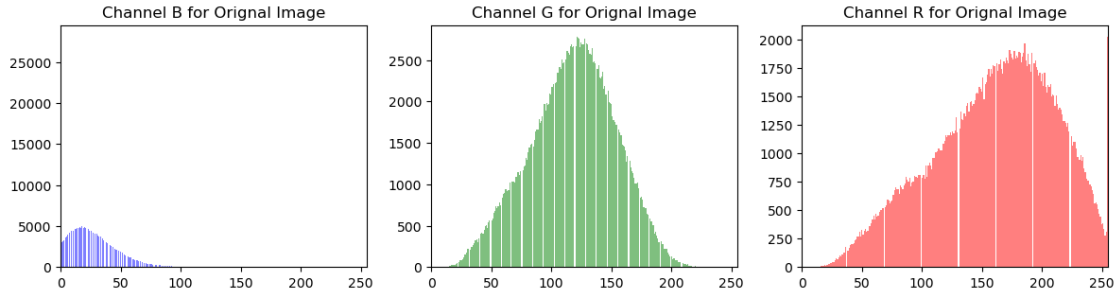
```
    # Reconstruct the new image
    new_image = np.zeros_like(input_image)
    for ch in range(3):
        new_image[:, :, ch] = equalize_channel(input_image[:, :, ch],␣
↪final_sk_list[ch])

    # Display histograms for each color channel of input and reference images
    display_channel_histogram(input_image, 'Orignal Image')
    display_channel_histogram(reference_image, 'Referance Image')
    display_channel_histogram(new_image, 'New Matched Image')
    display_colored_channel_histogram(input_image,'Orignal Image')
    display_colored_channel_histogram(reference_image, 'Referance Image')
    display_colored_channel_histogram(new_image,'New matched Image')

    # Display comparison between original and new images
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
    plt.title('Histogram Matched Image')
    plt.show()
```

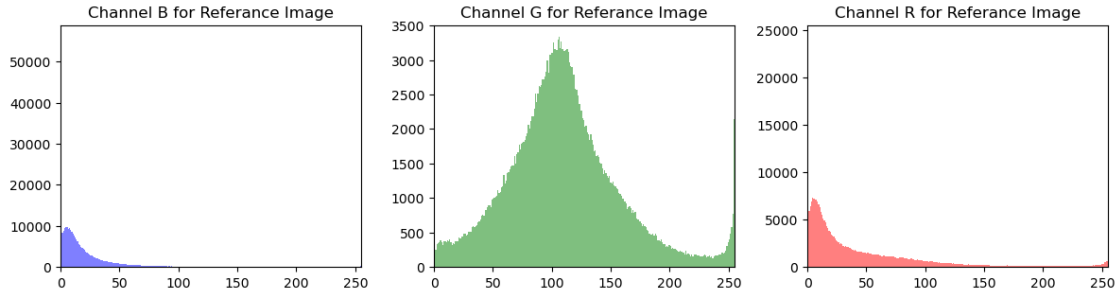
[6]:
```
# Images
image1 = cv2.imread("C:/Users/VARUN/Downloads/image1 (1).jpg")
image2 = cv2.imread("C:/Users/VARUN/Downloads/image2 (1).jpg")
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
plt.imshow(image1)
plt.show()
plt.imshow(image2)
plt.show()
```

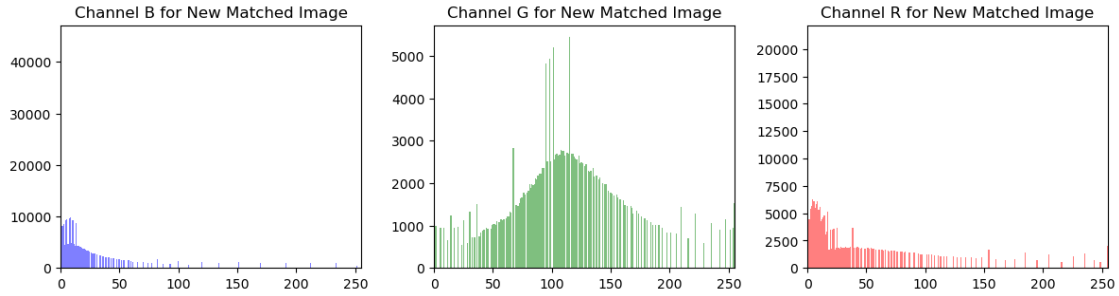

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500

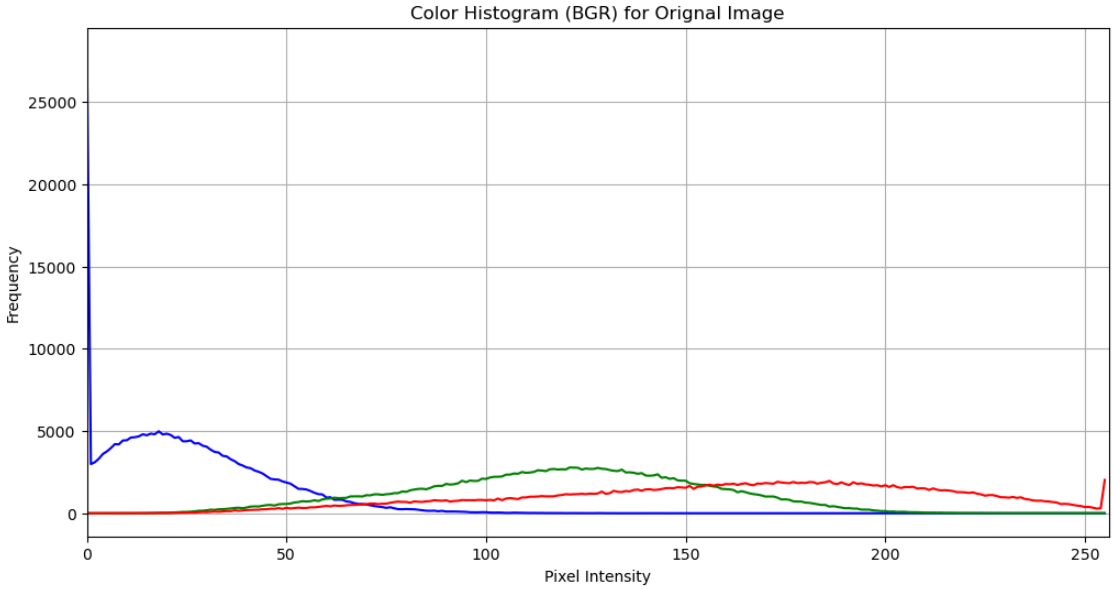

0
100
200
300
400
0
100
200
300
400
500

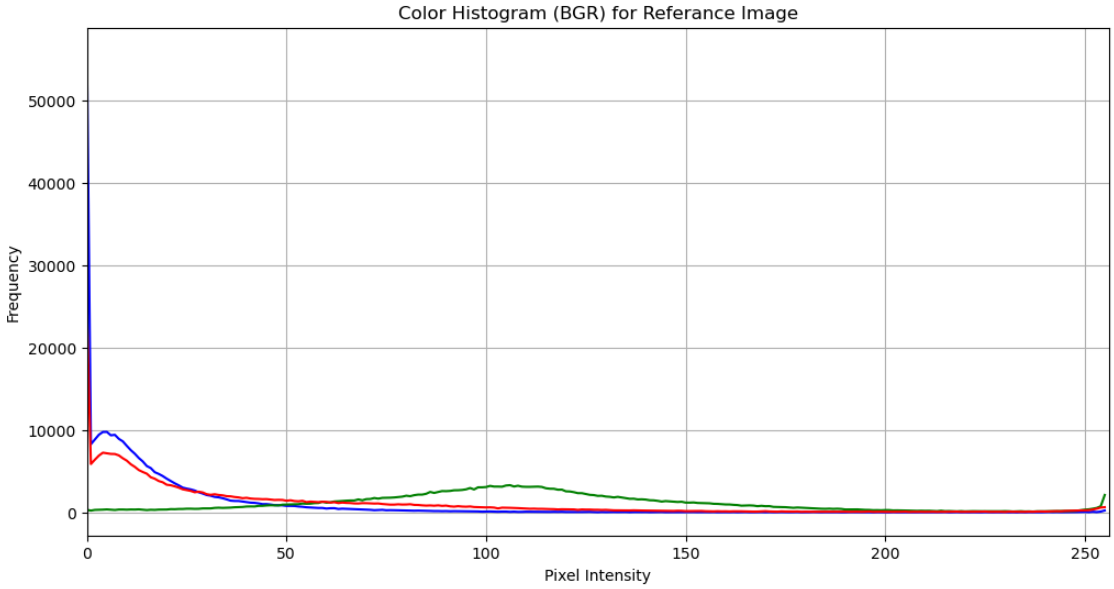
[7]:
```
histogram_matching("C:/Users/VARUN/Downloads/image1 (1).jpg", "C:/Users/VARUN/
↪Downloads/image2 (1).jpg")
```

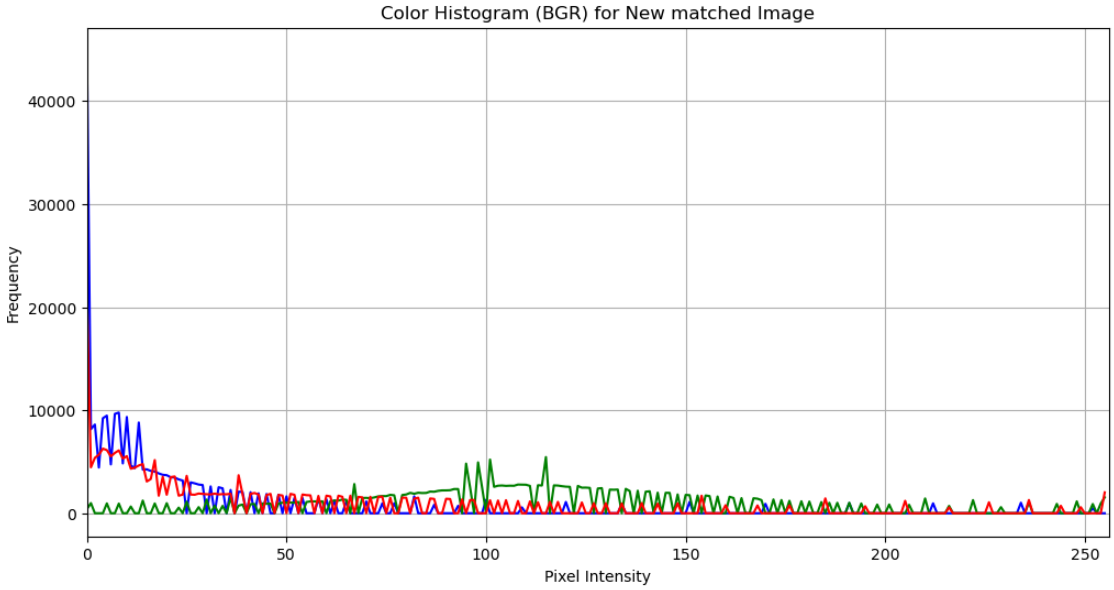


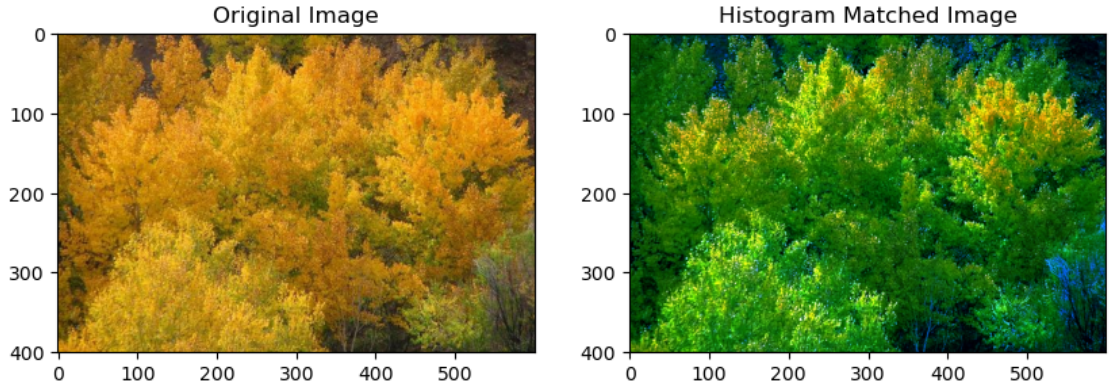





Color Histogram (BGR) for Orignal Image
Frequency
Pixel Intensity
0
5000
10000
15000
20000
25000
0
50
100
150
200
250


Color Histogram (BGR) for Referance Image
Frequency
Pixel Intensity
0
10000
20000
30000
40000
50000
0
50
100
150
200
250





[8]:
```
from skimage import data
import numpy as np
import matplotlib.pyplot as plt

########## Q3
# (a and b)
def intensity_slicing(image, low, high, highlight=None, set_rest_to=None,␣
 ↪set_high_to=255):
    """Perform intensity slicing on the image."""
    # Apply intensity slicing transformation
    if highlight:
        result = np.where((image >= low) & (image <= high), image, 0)
        transformation_name = f'Intensity Slicing ({low}-{high}) Highlighted'
```

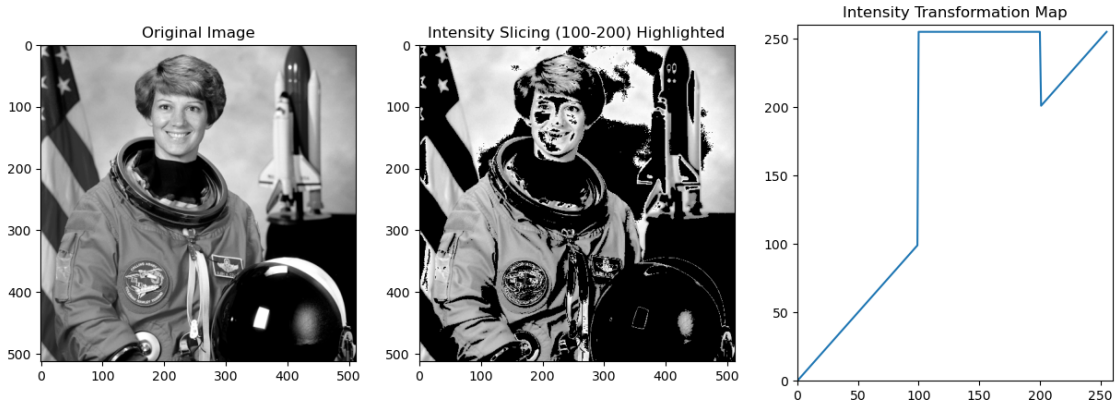
```
    else:
        if set_rest_to == "eq":
            result = np.where((image >= low) & (image <= high), set_high_to,␣
 ↪image)
        elif set_rest_to == "zero":
            result = np.where((image >= low) & (image <= high), set_high_to, 0 )
        transformation_name = f'Intensity Slicing ({low}-{high}) Unchanged'

    # Generate the intensity transformation map
    x_range = list(range(256))
    map_y_x = []
    for intensity in x_range:
        if intensity >= low and intensity <= high:
            map_y_x.append(set_high_to)
        else:
            if set_rest_to == "eq":
                map_y_x.append(intensity)
            elif set_rest_to == "zero":
                map_y_x.append(0)

    return result, transformation_name, x_range, map_y_x


def plot_intensity_transformation(image, transformed_image,␣
 ↪transformation_name, x_range, map_y_x):
    """Plot the intensity transformation."""
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')

    plt.subplot(1, 3, 2)
    plt.imshow(transformed_image, cmap='gray')
    plt.title(transformation_name)

    plt.subplot(1, 3, 3)
    plt.plot(x_range, map_y_x)
    plt.ylim(0, 260)
    plt.xlim(0, 260)
    plt.title('Intensity Transformation Map')

    plt.show()

########## Q4
# a)
def bit_plane_slicing(image):
```

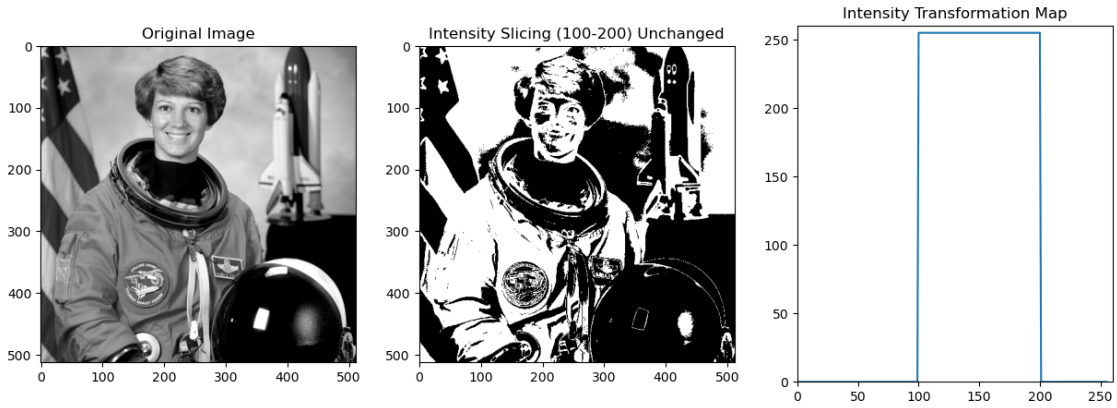
```
    """Perform bit-plane slicing on the image."""
    # Convert image to numpy array if it's not already
    if not isinstance(image, np.ndarray):
        image = np.array(image, dtype=np.uint8)
    # Ensure image is of type uint8 and between 0 and 255
    image = np.clip(image, 0, 255).astype(np.uint8)

    # Perform bit-plane slicing
    bit_planes = [np.bitwise_and(image, 2**i) for i in range(8)]
    return bit_planes


def plot_bit_planes(bit_planes):
    """Plot all the bit planes."""
    plt.figure(figsize=(12, 8))
    for i, plane in enumerate(bit_planes):
        plt.subplot(3, 3, i + 1)
        plt.imshow(plane, cmap='gray')
        plt.title(f'Bitplane {i}')
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# b)
def bitplane(image, plane, ret_lut=False):
    """
    Extract a specified bit plane from the image.

    Parameters:
        image (numpy.ndarray): The input image.
        plane (int or list of int): The bit plane(s) to extract.
        ret_lut (bool, optional): Whether to return the lookup table. Defaults␣
 ↪to False.

    Returns:
        numpy.ndarray or tuple: The extracted bit plane image, optionally with␣
 ↪the lookup table.
    """
    # Define a filter function based on the specified bit plane(s)
    if isinstance(plane, int):
        mask = 0b00000001 << plane
        filter_func = lambda x: ((x & mask) and True) * np.uint8(255)
    else:
        mask = 0b00000000
        for p in plane:
            mask = mask | (0b00000001 << p)
        filter_func = lambda x: x & mask
```

```
    # Create a lookup table with the filter for intensity transformation
    lut = np.array([filter_func(i) for i in range(256)], dtype=np.uint8)

    # Apply the transformation using the lookup table
    img = cv2.LUT(np.uint8(image), lut)

    if ret_lut:
        return img, lut

    return img


def plot_bit_plane_transformation_function(image, selected_planes):
    """Plot the intensity transformation of selected bit planes."""
    plt.figure(figsize=(12, 4))
    for i, plane_idx in enumerate(selected_planes):
        transformed_image, lut = bitplane(image, plane_idx, ret_lut=True)
        plt.subplot(1, len(selected_planes), i + 1)
        plt.plot(np.arange(256), lut, drawstyle="steps")
        plt.title(f'Bitplane {plane_idx} Intensity Transformation')
        plt.xlabel('Pixel Intensity')
        plt.ylabel('Transformed Intensity')
    plt.tight_layout()
    plt.show()


# c)
def reconstruct_image(bit_planes, num_planes):
    """Reconstruct the image with a given number of bit planes."""
    reconstructed_image = sum(bit_planes[8-num_planes:])
    return reconstructed_image
#### Results####
# Load grayscale astronaut image
astronaut_gray = data.astronaut()

# Convert to grayscale if needed
if astronaut_gray.ndim == 3:
    astronaut_gray = astronaut_gray.mean(axis=2)
############
print("Q3 part a and b")
# Intensity slicing
low = 100
high = 200
set_high_to = 255
```

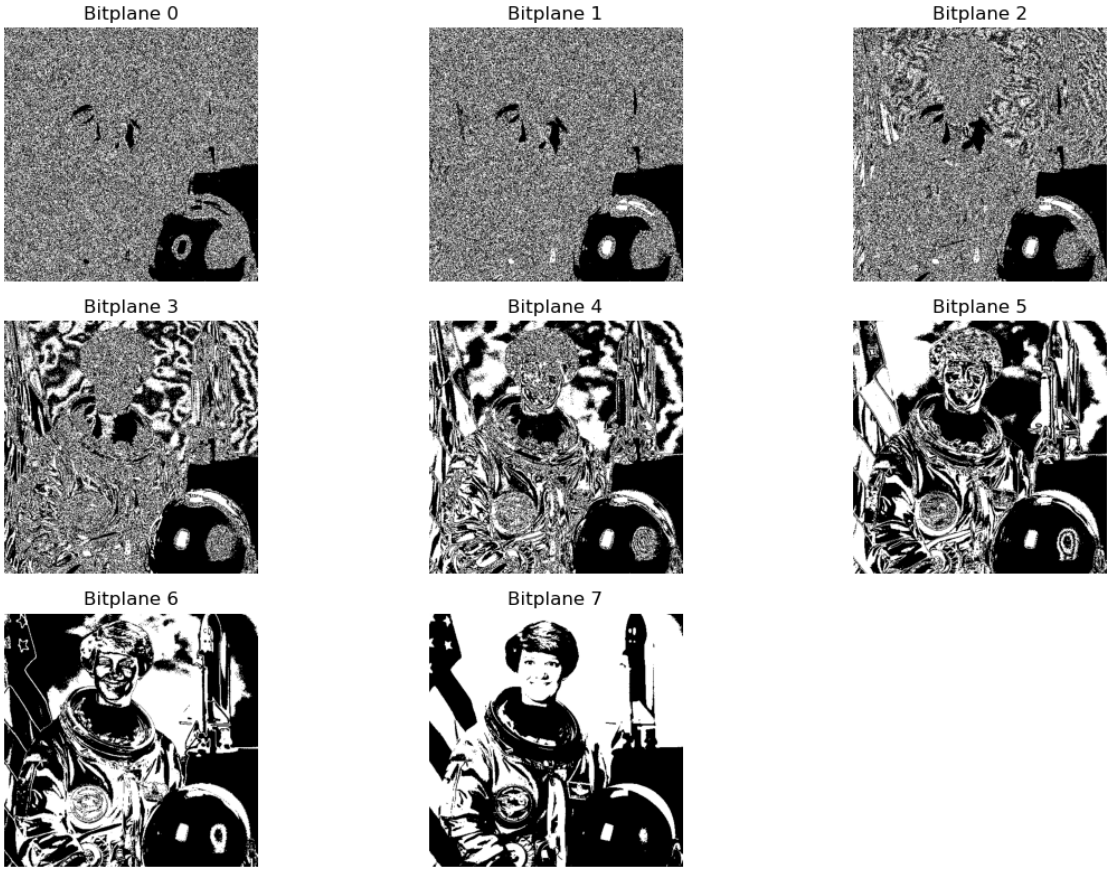
```
highlighted_image, highlighted_name, x_range, map_y_x =␣
 ↪intensity_slicing(astronaut_gray, low, high, highlight=True,␣
 ↪set_rest_to='eq', set_high_to=set_high_to)
unchanged_image, unchanged_name, x_range_u, map_y_x_u =␣
 ↪intensity_slicing(astronaut_gray, low, high, highlight=False,␣
 ↪set_rest_to='zero', set_high_to=set_high_to)

# Plot intensity slicing results
plot_intensity_transformation(astronaut_gray, highlighted_image,␣
 ↪highlighted_name, x_range, map_y_x)
plot_intensity_transformation(astronaut_gray, unchanged_image, unchanged_name,␣
 ↪x_range_u, map_y_x_u)

print("Q4 part a")
# Bit-plane slicing
bit_planes = bit_plane_slicing(astronaut_gray)

# Plot all bit planes
plot_bit_planes(bit_planes)


print("Q4 part b")
# Tranformaion functions
selected_planes = [0, 3, 7,[0,3,7]]  # Bitplane0, Bitplane3, Bitplane7
plot_bit_plane_transformation_function(astronaut_gray, selected_planes)


print("Q4 part c")
# Reconstruct the image with fewer bit planes
for i in range(8):
    num_planes_to_keep = i+1
    reconstructed_image = reconstruct_image(bit_planes, num_planes_to_keep)
    lis = []
    lis = [i for i in range(8-num_planes_to_keep,8)]
    # Plot reconstructed image
    plt.imshow(reconstructed_image, cmap='gray')
    plt.title(f'Reconstructed Image with {lis} Bit Planes')
    plt.axis('off')
    plt.show()
```

Q3 part a and b

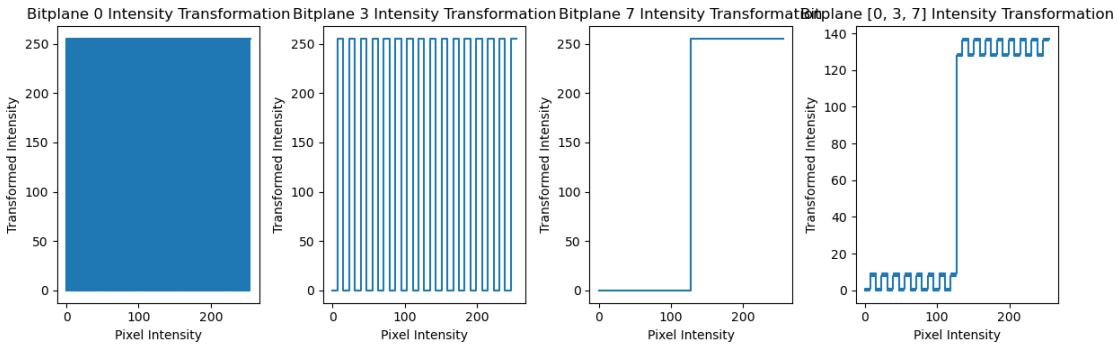

Original Image
Intensity Slicing (100-200) Highlighted
Intensity Transformation Map


Original Image
Intensity Slicing (100-200) Unchanged
Intensity Transformation Map

Q4 part a



Q4 part b



Q4 part c

Reconstructed Image with [7] Bit Planes

Reconstructed Image with [6, 7] Bit Planes

Reconstructed Image with [5, 6, 7] Bit Planes

Reconstructed Image with [4, 5, 6, 7] Bit Planes

Reconstructed Image with [3, 4, 5, 6, 7] Bit Planes

Reconstructed Image with [2, 3, 4, 5, 6, 7] Bit Planes

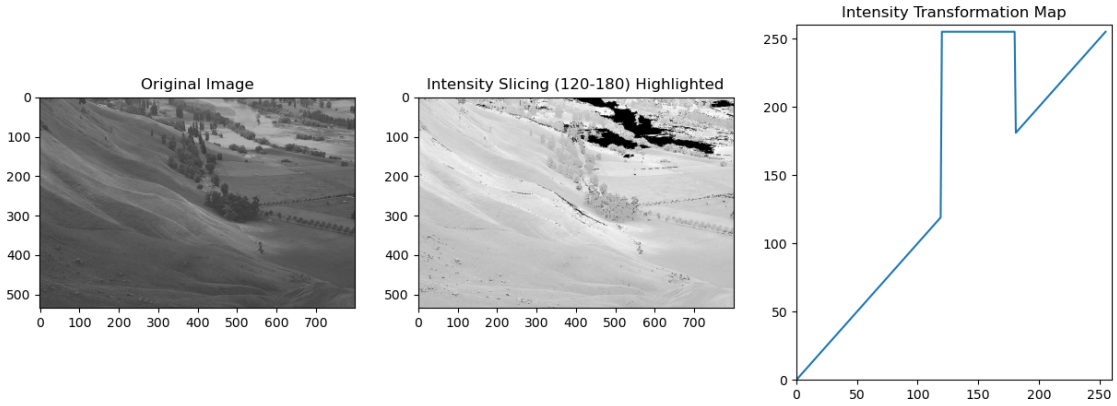
Reconstructed Image with [1, 2, 3, 4, 5, 6, 7] Bit Planes

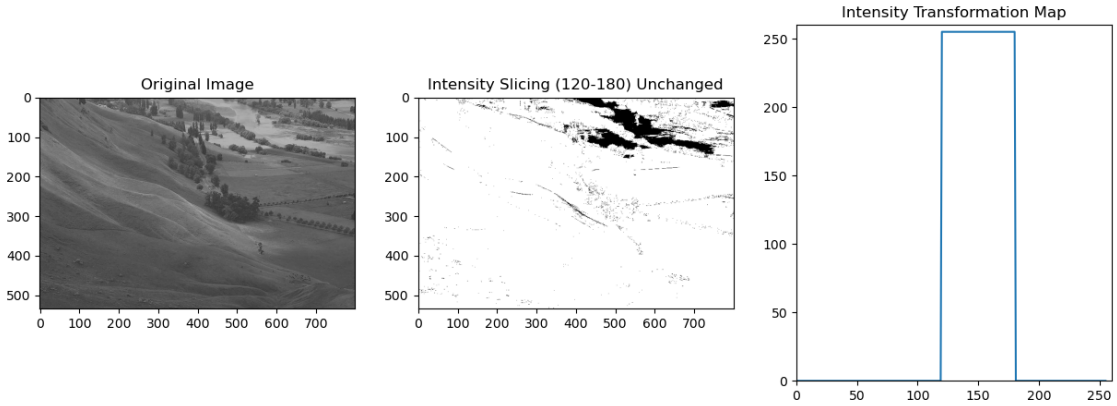
Reconstructed Image with [0, 1, 2, 3, 4, 5, 6, 7] Bit Planes

[9]:
```
# Load grayscale HawkesBay
Hawkes = cv2.imread("C:/Users/VARUN/Downloads/HawkesBay.jpg")

# Convert to grayscale if needed
if Hawkes.ndim == 3:
    Hawkes = Hawkes.mean(axis=2)

print("Q3 part a and b")
# Intensity slicing
low = 120
high = 180
highlighted_image, highlighted_name, x_range, map_y_x =␣
 ↪intensity_slicing(Hawkes, low, high, highlight=True, set_rest_to='eq',␣
 ↪set_high_to=set_high_to)
unchanged_image, unchanged_name, x_range_u, map_y_x_u =␣
 ↪intensity_slicing(Hawkes, low, high, highlight=False, set_rest_to='zero',␣
 ↪set_high_to=set_high_to)

# Plot intensity slicing results
plot_intensity_transformation(Hawkes, highlighted_image, highlighted_name,␣
 ↪x_range, map_y_x)
plot_intensity_transformation(Hawkes, unchanged_image, unchanged_name,␣
 ↪x_range_u, map_y_x_u)
# Bit-plane slicing
print("Q4 part a")
bit_planes = bit_plane_slicing(Hawkes)

# Plot all bit planes
plot_bit_planes(bit_planes)
print("Q4 part b")
# Tranformaion functions
selected_planes = [0, 3, 7,[0,3,7]]  # Bitplane0, Bitplane3, Bitplane7
plot_bit_plane_transformation_function(Hawkes, selected_planes)
print("Q4 part c")
# Reconstruct the image with fewer bit planes
for i in range(8):
    num_planes_to_keep = i+1
    reconstructed_image = reconstruct_image(bit_planes, num_planes_to_keep)
    lis = []
    lis = [i for i in range(8-num_planes_to_keep,8)]
    # Plot reconstructed image
    plt.imshow(reconstructed_image, cmap='gray')
    plt.title(f'Reconstructed Image with {lis} Bit Planes')
    plt.axis('off')
```

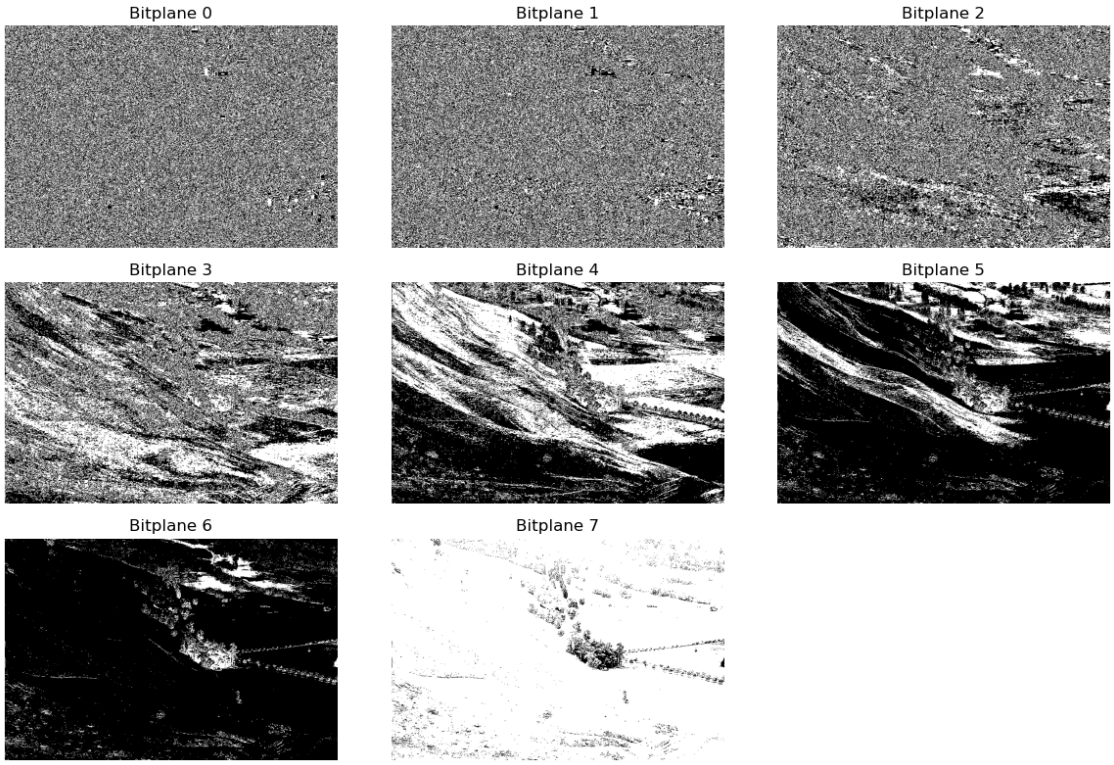
```
    plt.show()
```

Q3 part a and b

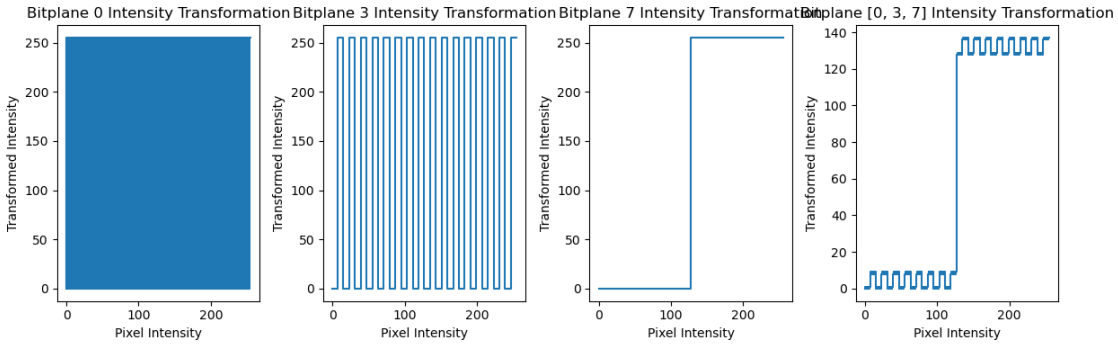




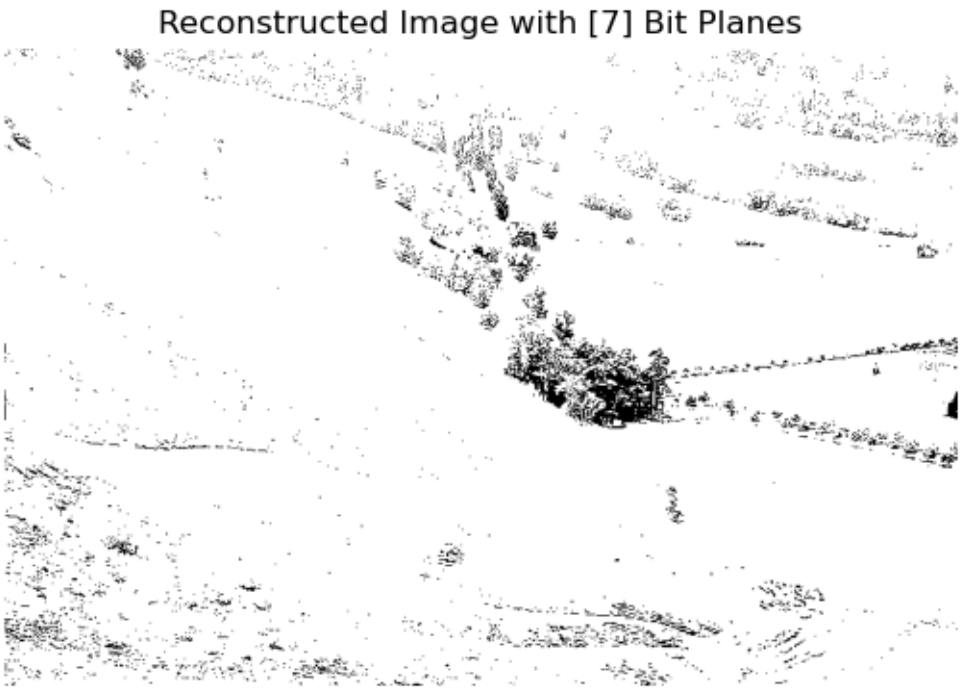
Q4 part a

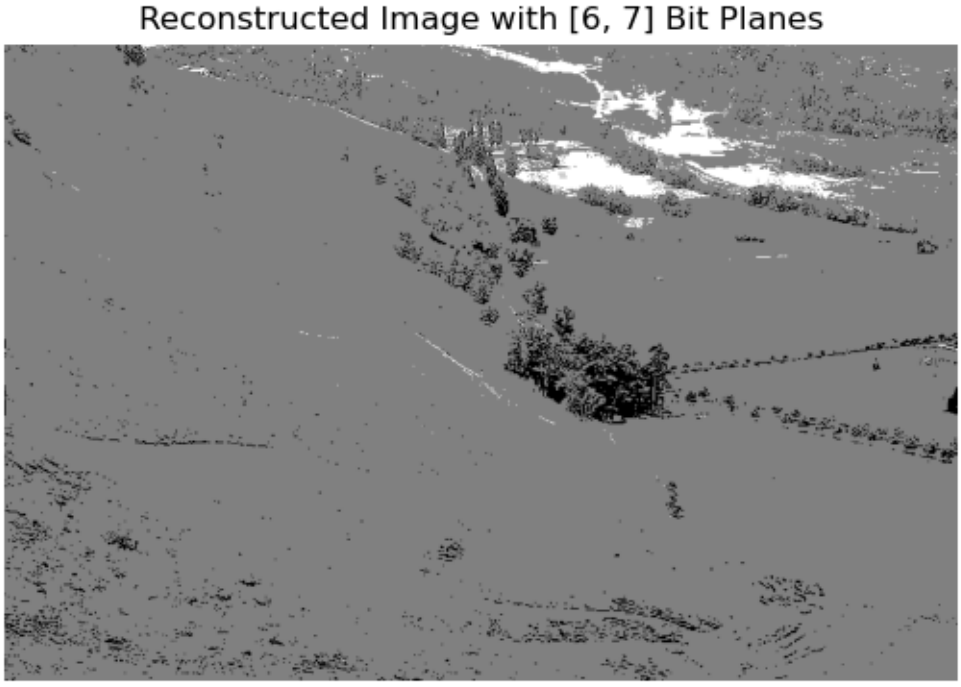


Q4 part b



Q4 part c


Reconstructed Image with [7] Bit Planes


Reconstructed Image with [6, 7] Bit Planes

Reconstructed Image with [5, 6, 7] Bit Planes

Reconstructed Image with [4, 5, 6, 7] Bit Planes

Reconstructed Image with [3, 4, 5, 6, 7] Bit Planes

Reconstructed Image with [2, 3, 4, 5, 6, 7] Bit Planes

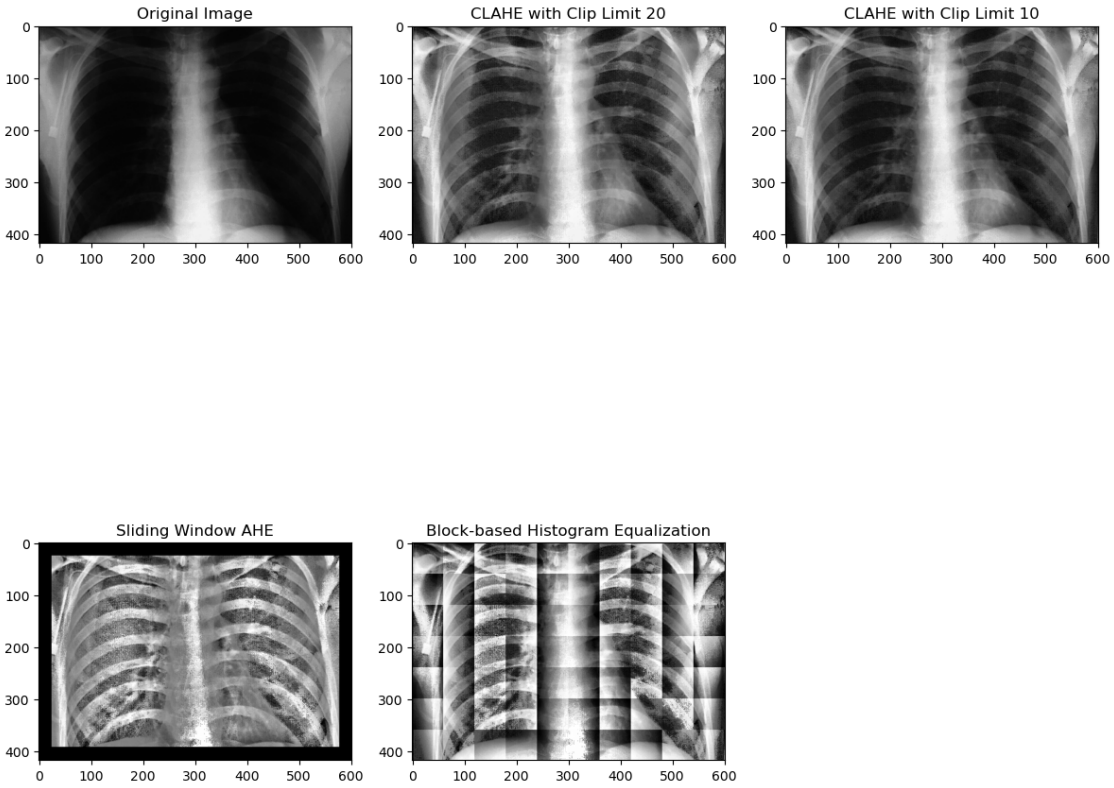
Reconstructed Image with [1, 2, 3, 4, 5, 6, 7] Bit Planes

Reconstructed Image with [0, 1, 2, 3, 4, 5, 6, 7] Bit Planes

Astronaut with 4,5,6,7 bit planes looks decent enough so 4 bit planes and HawkesBay with 3,4,5,6,7 looks good so 5 bit planes for that

[10]:
```
##### Q4
# Load the image
image = cv2.imread("C:/Users/VARUN/Downloads/chestxray1.png", 0)

# (a) CLAHE
def apply_clahe(image, clip_limit):
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=(8, 8))
    return clahe.apply(image)

# Experiment with different clip limits
clahe_image_1 = apply_clahe(image, clip_limit=20)
clahe_image_2 = apply_clahe(image, clip_limit=10)

# (b) SWAHE
def sliding_window_ahe(image, k):
    h, w = image.shape
    result = np.zeros_like(image)
    pad = k // 2

    for i in range(pad, h-pad):
        for j in range(pad, w-pad):
            neighborhood = image[i-pad:i+pad+1, j-pad:j+pad+1]
            hist, _ = np.histogram(neighborhood.flatten(), bins=256, range=(0,␣
 ↪255))
            cdf = hist.cumsum()
            cdf_normalized = cdf / cdf[-1]
            result[i, j] = cdf_normalized[image[i, j]] * 255

    return result

swahe_image = sliding_window_ahe(image, k=51)

# (c) Block-based histogram equalization
def block_histogram_equalization(image, block_size):
    h, w = image.shape
    result = np.zeros_like(image)

    for i in range(0, h, block_size):
        for j in range(0, w, block_size):
            block = image[i:i+block_size, j:j+block_size]
            eq_block = cv2.equalizeHist(block)
            result[i:i+block_size, j:j+block_size] = eq_block

    return result

block_hist_eq_image = block_histogram_equalization(image, block_size=60)
```

```
# Displaying results inline
plt.figure(figsize=(12, 12))

plt.subplot(231)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(232)
plt.title('CLAHE with Clip Limit 20')
plt.imshow(clahe_image_1, cmap='gray')

plt.subplot(233)
plt.title('CLAHE with Clip Limit 10')
plt.imshow(clahe_image_2, cmap='gray')

plt.subplot(234)
plt.title('Sliding Window AHE')
plt.imshow(swahe_image, cmap='gray')

plt.subplot(235)
plt.title('Block-based Histogram Equalization')
plt.imshow(block_hist_eq_image, cmap='gray')

plt.tight_layout()
plt.show()
```



CLAHE with a higher clip limit tends to produce more contrast-enhanced images, but excessively high clip limits may lead to artifacts. Sliding Window AHE and block-based histogram equalization techniques tend to enhance local details but may introduce block artifacts, especially when block size is large. Sliding Window AHE can be computationally expensive, especially for larger images and larger neighborhood sizes. Block-based histogram equalization is suitable for scenarios where local contrast enhancement is needed without significant computational overhead.

The CLAHE method enhances overall contrast, making both ribs and lungs visible. SWAHE emphasizes ribs and spine more than the lungs. Block-based AHE can isolate specific areas like the spinal cord for abnormalities. Choose the method based on the desired focus in your X-ray image.