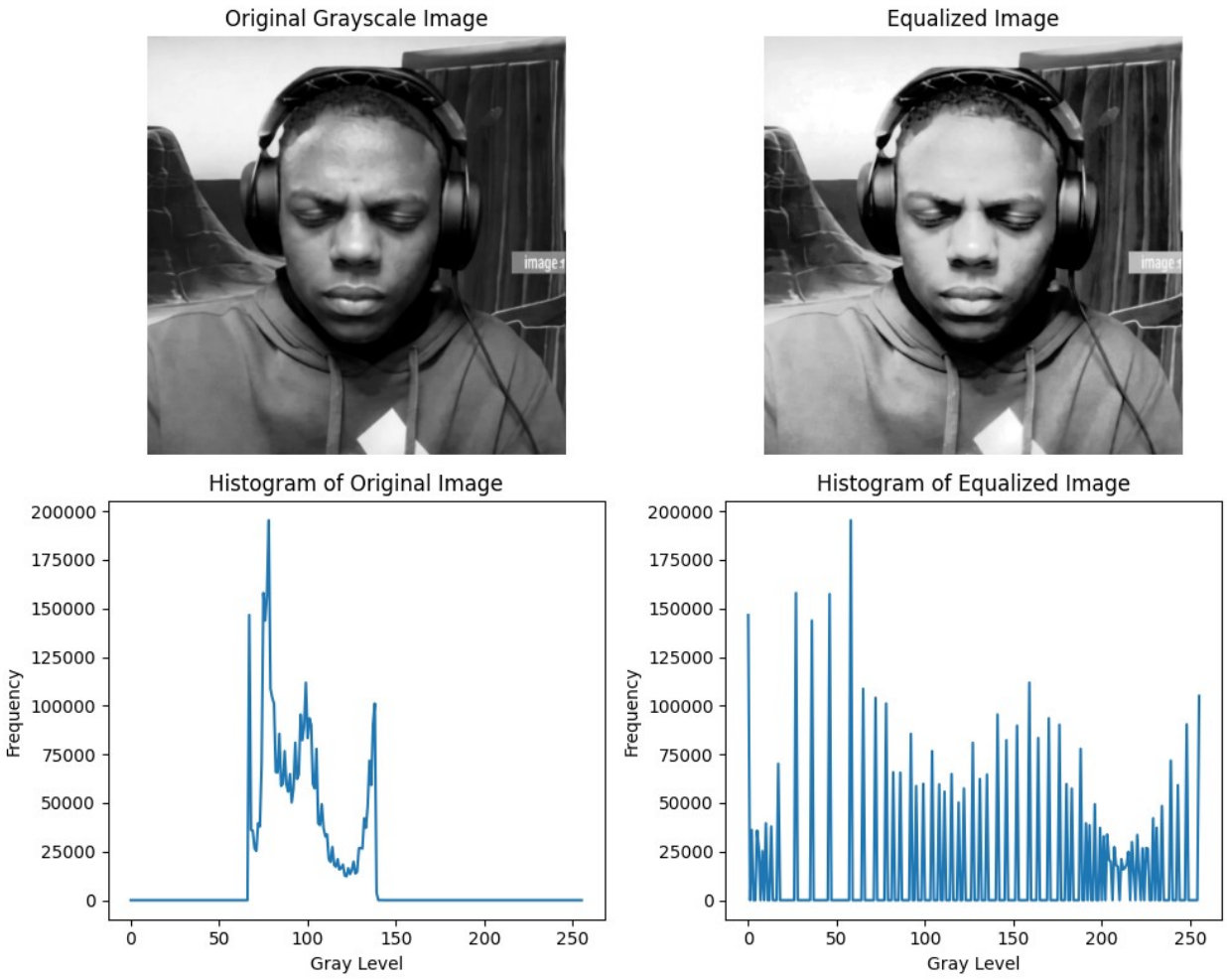
Part 1

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread(r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\
low_contrast_image.png", cv2.IMREAD_GRAYSCALE)

hist_original = cv2.calcHist(
    [img],
    [0],
    None,
    [256],
    [0, 256]
)

equalized_img = cv2.equalizeHist(img)

hist_equalized = cv2.calcHist(
    [equalized_img],
    [0],
    None,
    [256],
    [0, 256]
)

plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title("Original Grayscale Image")
plt.imshow(img, cmap='gray')
plt.axis("off")

plt.subplot(2, 2, 2)
plt.title("Equalized Image")
plt.imshow(equalized_img, cmap='gray')
plt.axis("off")

plt.subplot(2, 2, 3)
plt.title("Histogram of Original Image")
plt.plot(hist_original)
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(2, 2, 4)
plt.title("Histogram of Equalized Image")
plt.plot(hist_equalized)
plt.xlabel("Gray Level")
```

```
plt.ylabel("Frequency")

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



Part 2

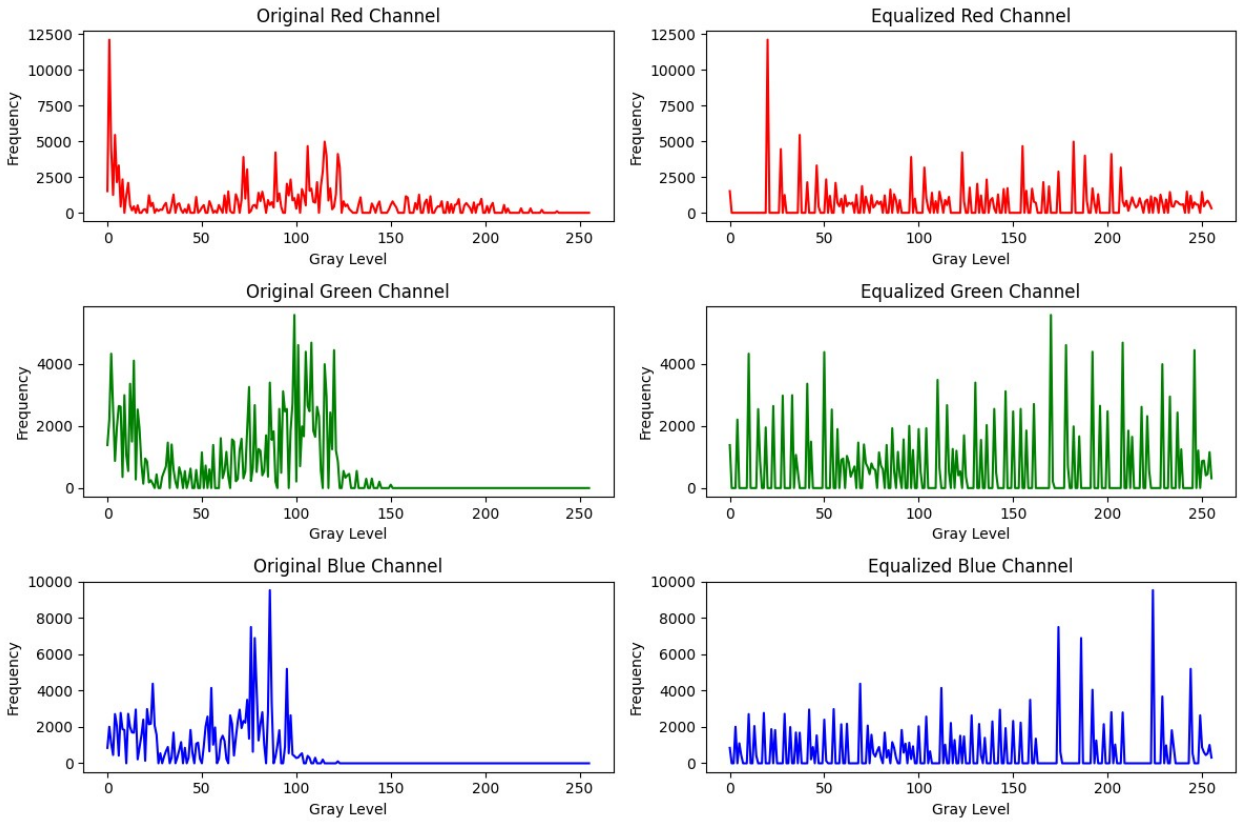
```
img_rgb = cv2.imread(r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\
charlie-kirk-charlie-kirk-meme.gif")
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB)

R, G, B = cv2.split(img_rgb)

hist_R = cv2.calcHist([R], [0], None, [256], [0,256])
hist_G = cv2.calcHist([G], [0], None, [256], [0,256])
hist_B = cv2.calcHist([B], [0], None, [256], [0,256])
```

```
R_eq = cv2.equalizeHist(R)
G_eq = cv2.equalizeHist(G)
B_eq = cv2.equalizeHist(B)

img_rgb_eq = cv2.merge((R_eq, G_eq, B_eq))

hist_R_eq = cv2.calcHist([R_eq], [0], None, [256], [0,256])
hist_G_eq = cv2.calcHist([G_eq], [0], None, [256], [0,256])
hist_B_eq = cv2.calcHist([B_eq], [0], None, [256], [0,256])


plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.title("Original RGB")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(1,2,2)
plt.title("Equalized RGB")
plt.imshow(img_rgb_eq)
plt.axis("off")
plt.show()

plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)
plt.title("Original Red Channel")
plt.plot(hist_R, color='r')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(3, 2, 2)
plt.title("Equalized Red Channel")
plt.plot(hist_R_eq, color='r')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(3, 2, 3)
plt.title("Original Green Channel")
plt.plot(hist_G, color='g')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(3, 2, 4)
plt.title("Equalized Green Channel")
plt.plot(hist_G_eq, color='g')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(3, 2, 5)
```

```
plt.title("Original Blue Channel")
plt.plot(hist_B, color='b')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(3, 2, 6)
plt.title("Equalized Blue Channel")
plt.plot(hist_B_eq, color='b')
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

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

Original RGB

Equalized RGB



Convert the RGB image to HSI

```
img_hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)

H, S, V = cv2.split(img_hsv)

hist_V = cv2.calcHist([V], [0], None, [256], [0,256])

V_eq = cv2.equalizeHist(V)

hist_V_eq = cv2.calcHist([V_eq], [0], None, [256], [0,256])

img_hsv_eq = cv2.merge((H, S, V_eq))
```

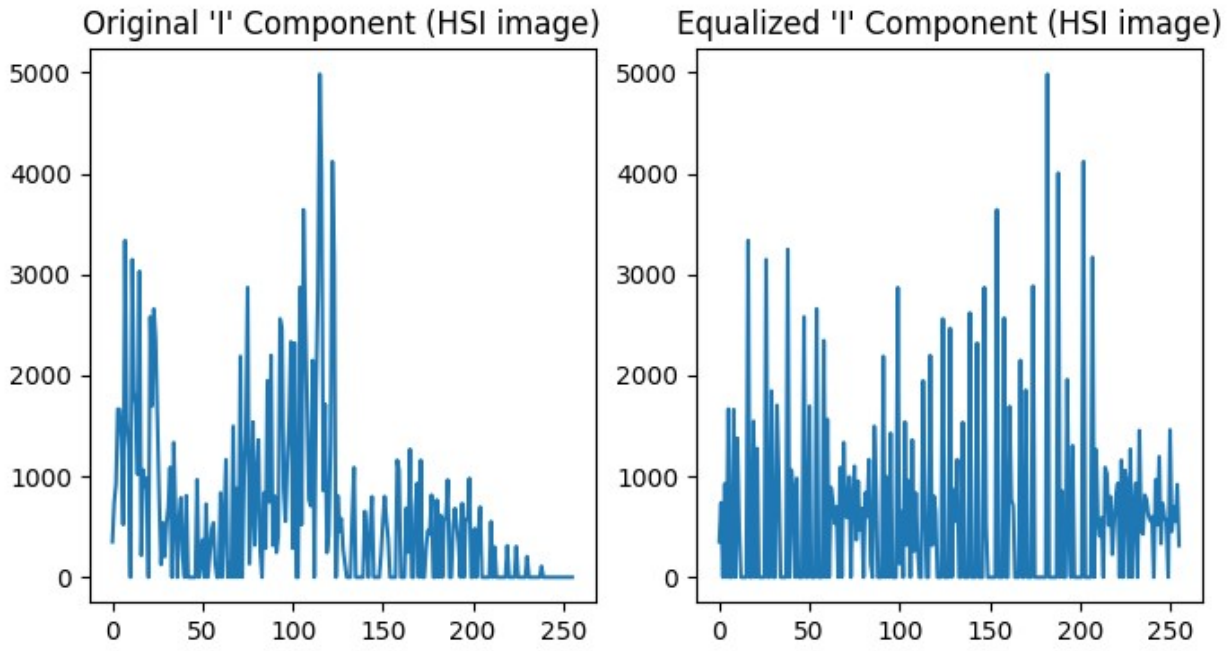
Convert the HSI image back to RGB

```
img_rgb_hsi_eq = cv2.cvtColor(img_hsv_eq, cv2.COLOR_HSV2RGB)

plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.title("Original RGB")
plt.imshow(img_rgb)
plt.axis("off")

plt.subplot(1,2,2)
plt.title("HSI (I Equalized)")
```

```
plt.imshow(img_rgb_hsi_eq)
plt.axis("off")
plt.show()

plt.figure(figsize=(8,4))
plt.subplot(1,2,1)
plt.title("Original 'I' Component (HSI image)")
plt.plot(hist_V)

plt.subplot(1,2,2)
plt.title("Equalized 'I' Component (HSI image)")
plt.plot(hist_V_eq)

plt.show()
```

Original RGB

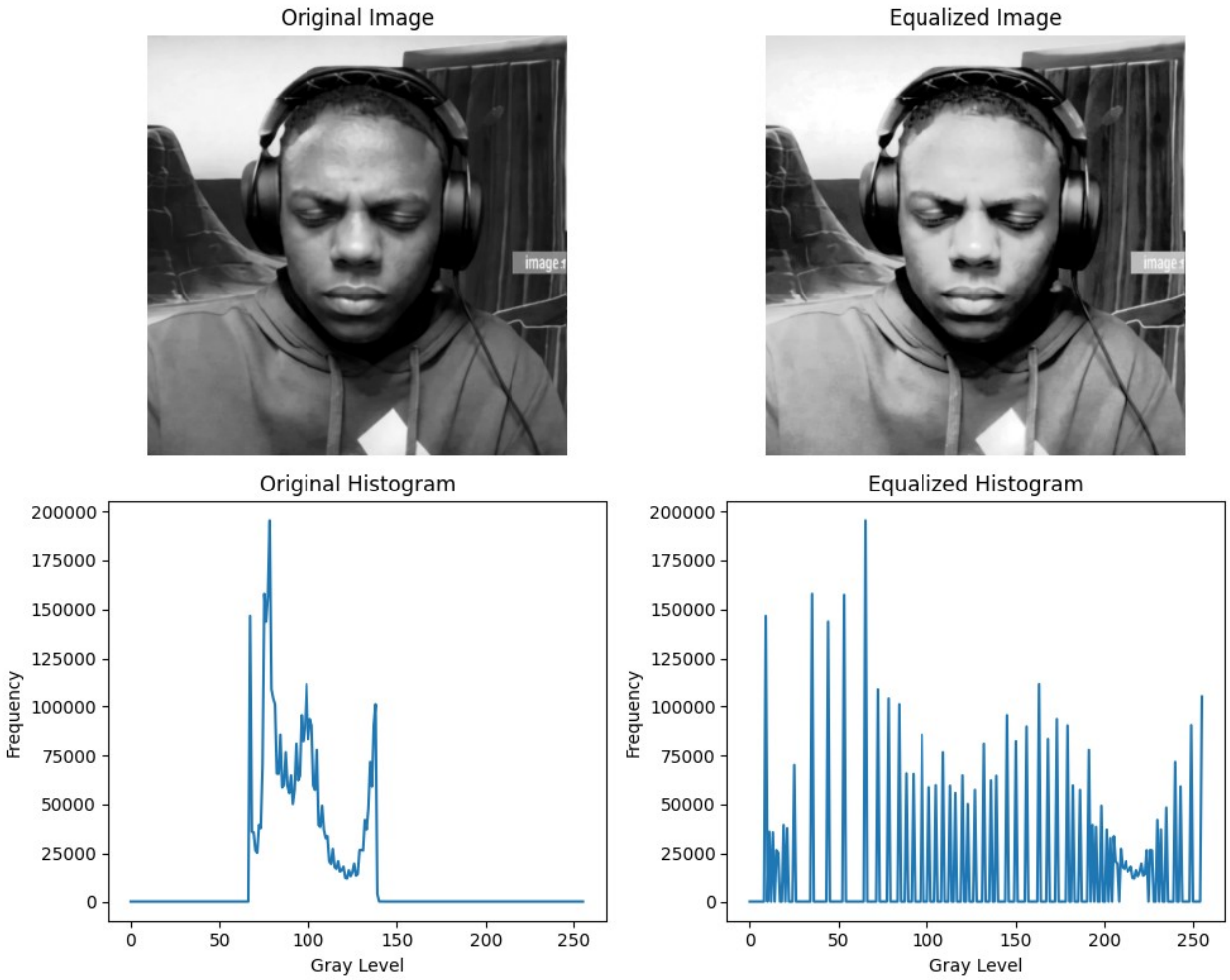
HSI (I Equalized)



Calculating the histogram and equlizing the image manually

```
img = cv2.imread(r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\
low_contrast_image.png", cv2.IMREAD_GRAYSCALE)

rows, cols = img.shape
n = rows * cols
L = 256

hist = np.zeros(L, dtype=int)
for i in range(rows):
    for j in range(cols):
        hist[img[i, j]] += 1

cum_hist = np.zeros(L, dtype=int)
cum_hist[0] = hist[0]
for k in range(1, L):
    cum_hist[k] = cum_hist[k - 1] + hist[k]

cdf = cum_hist / n
mapping = np.round(cdf * (L - 1)).astype(np.uint8)

equalized_img = np.zeros_like(img)
for i in range(rows):
    for j in range(cols):
        equalized_img[i, j] = mapping[img[i, j]]

hist_eq = np.zeros(L, dtype=int)
for i in range(rows):
    for j in range(cols):
```

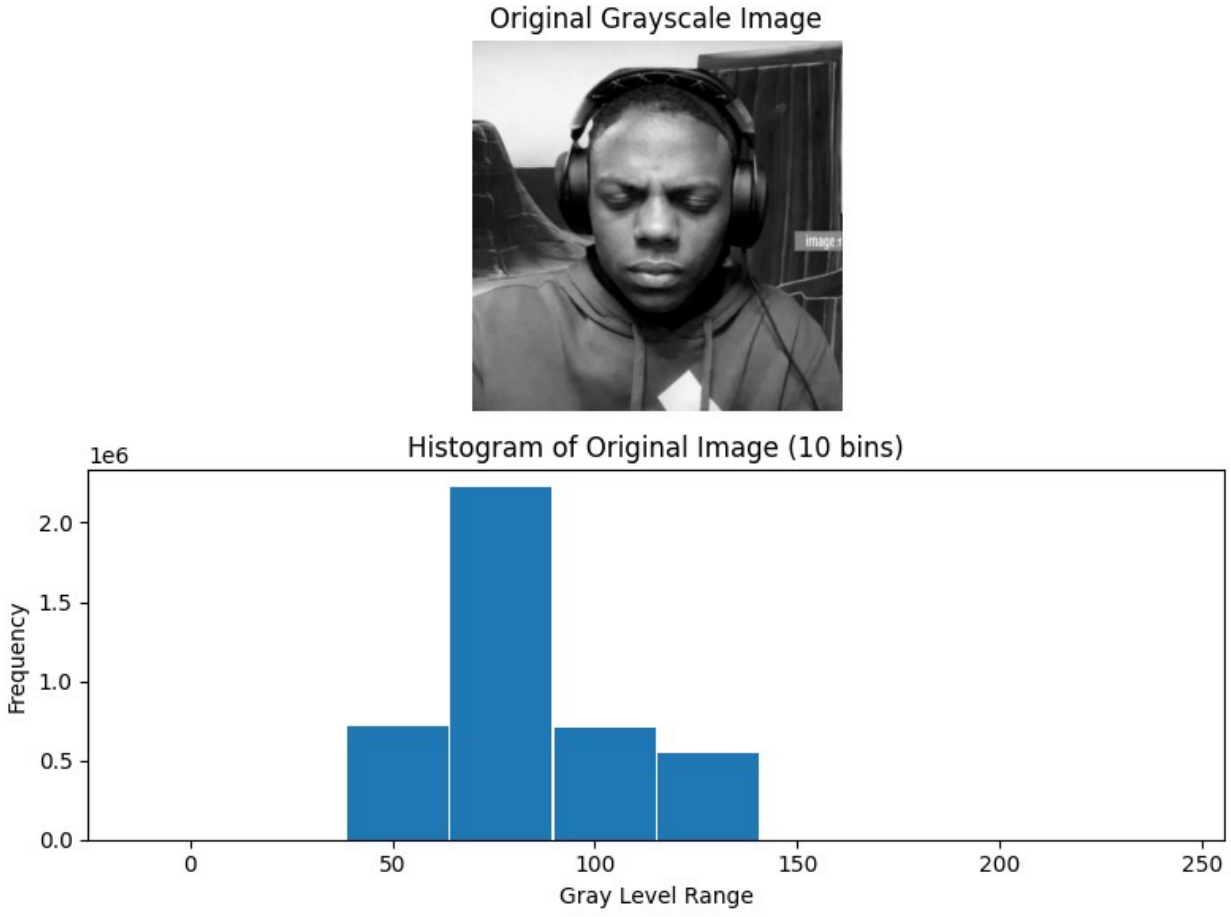
```
        hist_eq[equalized_img[i, j]] += 1

plt.figure(figsize=(10, 8))

plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(img, cmap="gray")
plt.axis("off")

plt.subplot(2, 2, 2)
plt.title("Equalized Image")
plt.imshow(equalized_img, cmap="gray")
plt.axis("off")

plt.subplot(2, 2, 3)
plt.title("Original Histogram")
plt.plot(hist)
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

plt.subplot(2, 2, 4)
plt.title("Equalized Histogram")
plt.plot(hist_eq)
plt.xlabel("Gray Level")
plt.ylabel("Frequency")

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



Changing the bin component in the calcHist() function

```
hist_original = cv2.calcHist(
    [img],
    [0],
    None,
    [10],
    [0, 256]
)

bin_edges = np.linspace(0, 256, 11)

plt.figure(figsize=(8, 6))

plt.subplot(2, 1, 1)
plt.title("Original Grayscale Image")
plt.imshow(img, cmap='gray')
plt.axis("off")

plt.subplot(2, 1, 2)
```

```
plt.title("Histogram of Original Image (10 bins)")
plt.bar(bin_edges[:-1], hist_original.flatten(), width=25)
plt.xlabel("Gray Level Range")
plt.ylabel("Frequency")

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

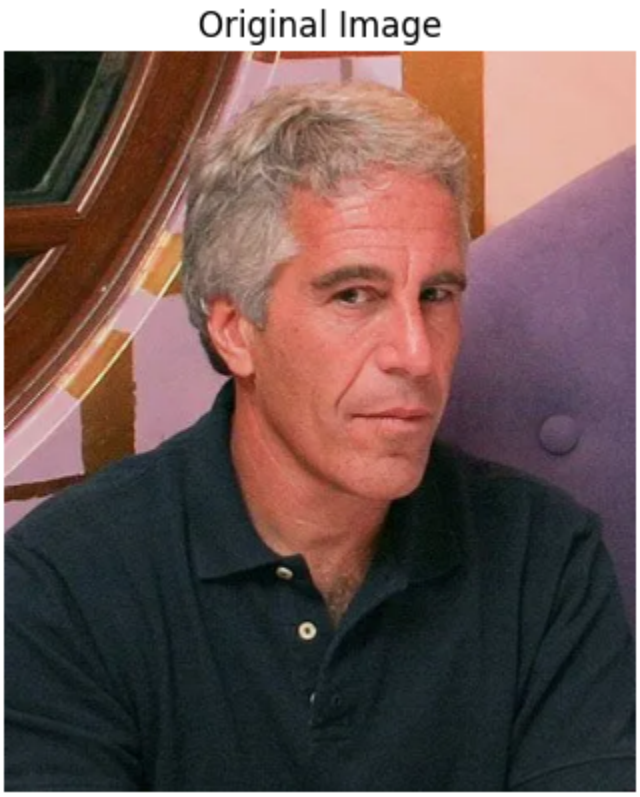


In [21]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Input Image

In [22]:
```
img = cv2.imread(r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Jeffrey-Epstein-S


img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img_rgb)
plt.title("Original Image")
plt.axis("off")
```

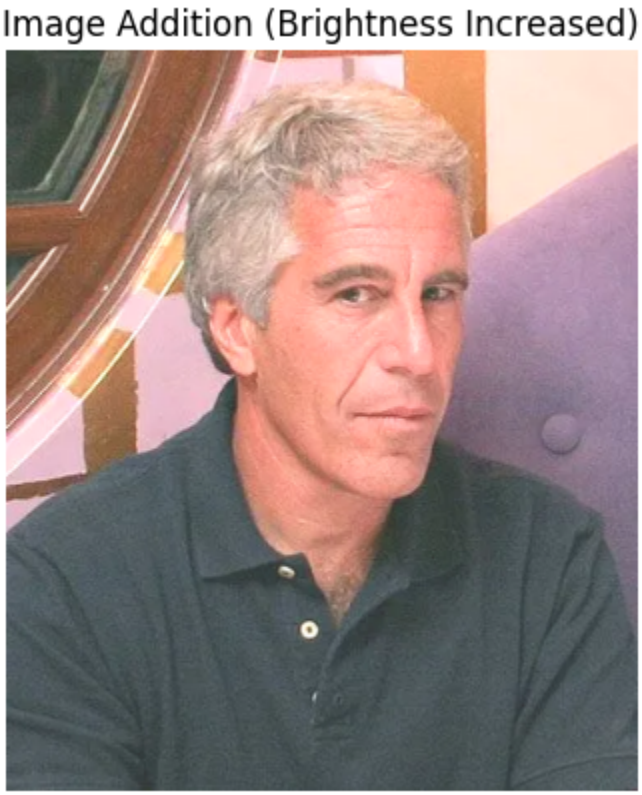
Out[22]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



Arithmetic Operations

In [23]:
```
bright = cv2.add(img, np.ones(img.shape, dtype='uint8') * 50)
bright_rgb = cv2.cvtColor(bright, cv2.COLOR_BGR2RGB)

plt.imshow(bright_rgb)
plt.title("Image Addition (Brightness Increased)")
plt.axis("off")
```

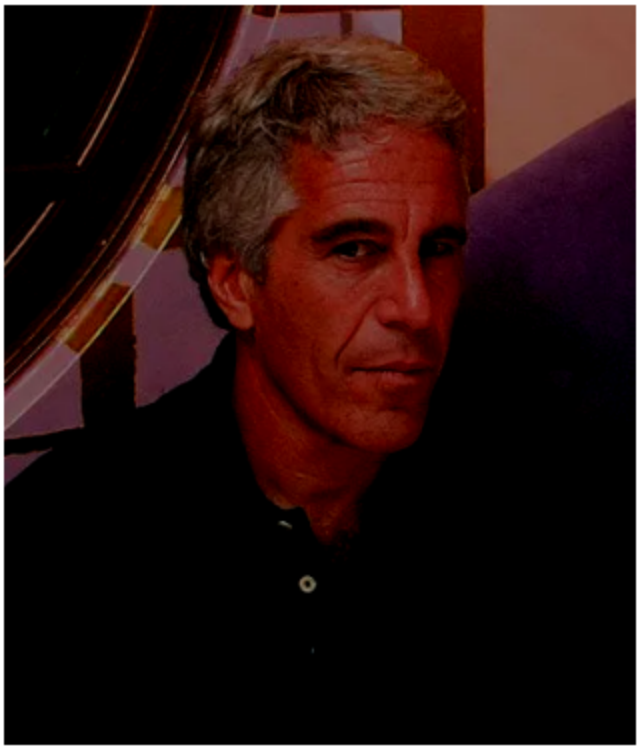
Out[23]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))



In [24]:
```
dark = cv2.subtract(img, np.ones(img.shape, dtype='uint8') * 100)
dark_rgb = cv2.cvtColor(dark, cv2.COLOR_BGR2RGB)

plt.imshow(dark_rgb)
plt.title("Image Subtraction (Brightness Decreased)")
plt.axis("off")
```

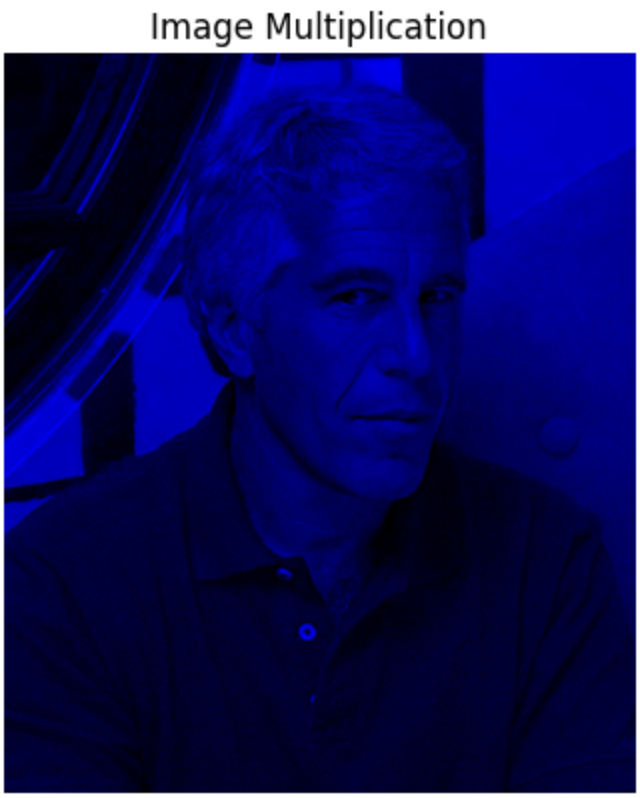
Out[24]: (np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))

Image Subtraction (Brightness Decreased)



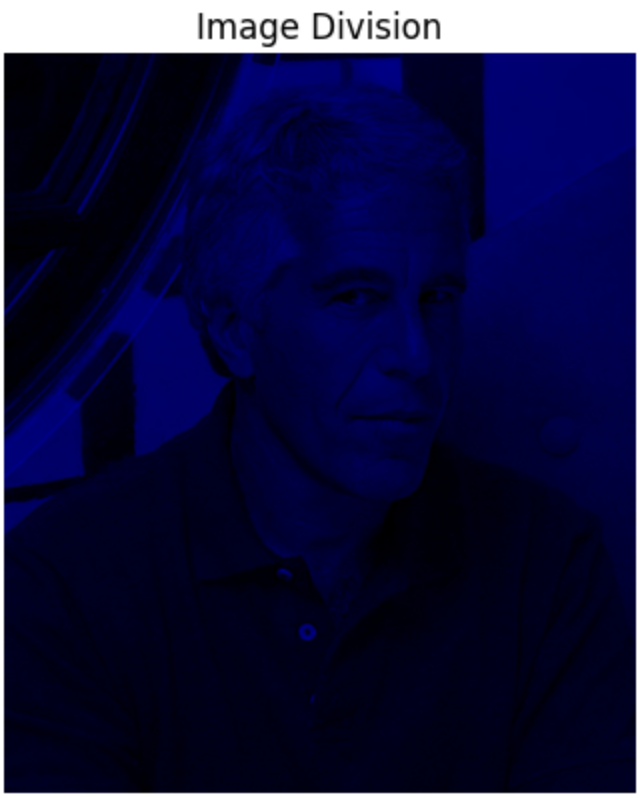
In [25]:
```
mul = cv2.multiply(img, np.array([1.2]))
mul_rgb = cv2.cvtColor(mul, cv2.COLOR_BGR2RGB)

plt.imshow(mul_rgb)
plt.title("Image Multiplication")
plt.axis("off")
```

Out[25]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



In [26]:
```
div = cv2.divide(img, np.array([1.5]))
div_rgb = cv2.cvtColor(div, cv2.COLOR_BGR2RGB)

plt.imshow(div_rgb)
plt.title("Image Division")
plt.axis("off")
```

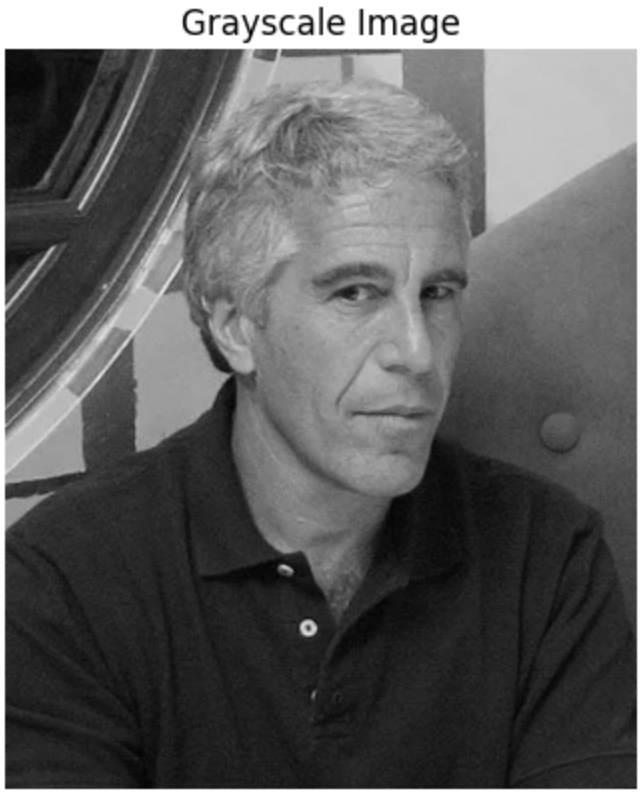
Out[26]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



Logical Operations

In [27]:
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray, cmap='gray')
plt.title("Grayscale Image")
plt.axis("off")
```

Out[27]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

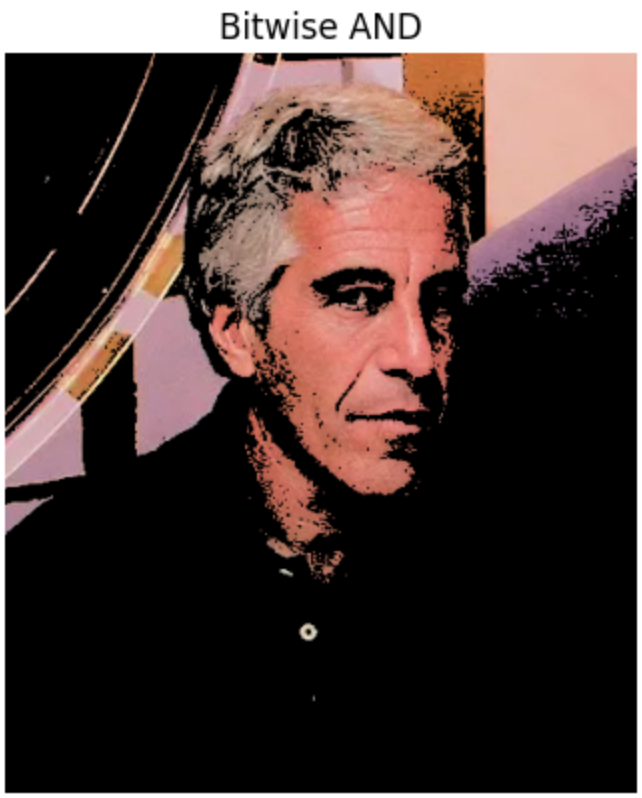


In [28]:
```
_, mask = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
plt.imshow(mask, cmap='gray')
plt.title("Binary Mask")
plt.axis("off")
```

Out[28]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

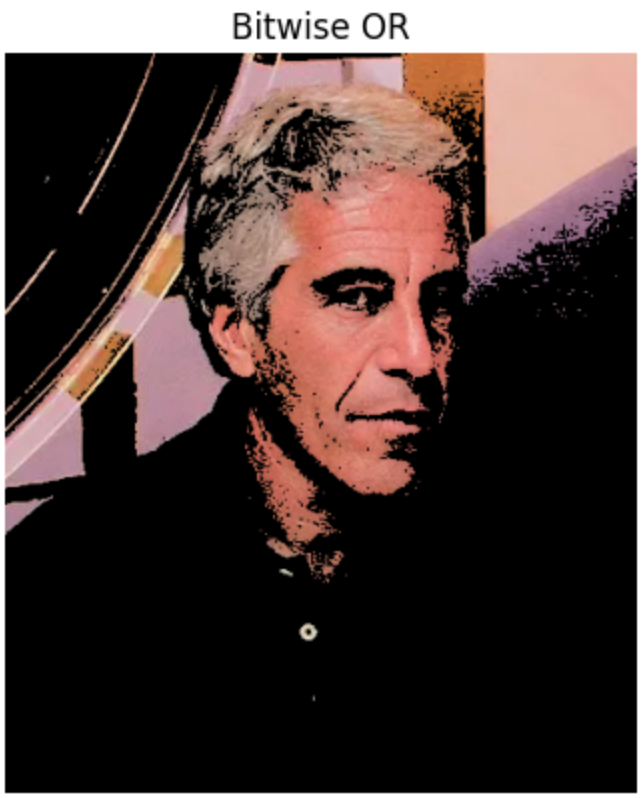
In [29]:
```
bit_and = cv2.bitwise_and(img, img, mask=mask)
bit_and_rgb = cv2.cvtColor(bit_and, cv2.COLOR_BGR2RGB)

plt.imshow(bit_and_rgb)
plt.title("Bitwise AND")
plt.axis("off")
```

Out[29]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



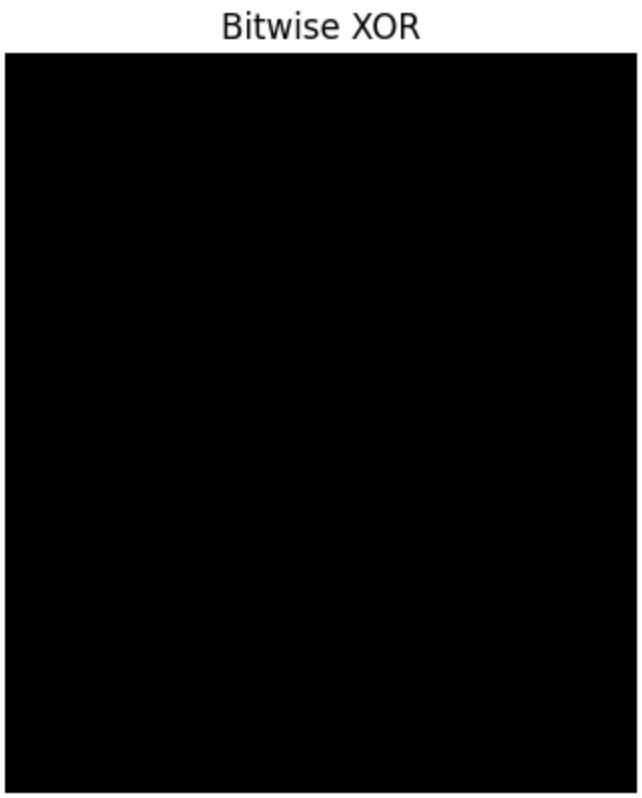
In [30]:
```
bit_or = cv2.bitwise_or(img, img, mask=mask)
bit_or_rgb = cv2.cvtColor(bit_or, cv2.COLOR_BGR2RGB)

plt.imshow(bit_or_rgb)
plt.title("Bitwise OR")
plt.axis("off")
```

Out[30]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



In [31]:
```
bit_xor = cv2.bitwise_xor(img, img, mask=mask)
bit_xor_rgb = cv2.cvtColor(bit_xor, cv2.COLOR_BGR2RGB)

plt.imshow(bit_xor_rgb)
plt.title("Bitwise XOR")
plt.axis("off")
```

Out[31]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



In [32]:
```
bit_not = cv2.bitwise_not(img)
bit_not_rgb = cv2.cvtColor(bit_not, cv2.COLOR_BGR2RGB)

plt.imshow(bit_not_rgb)
plt.title("Bitwise NOT (Negative Image)")
plt.axis("off")
```

Out[32]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

Image Resizing

In [33]:
```
resized = cv2.resize(img, (300, 300), interpolation=cv2.INTER_LINEAR)
resized_rgb = cv2.cvtColor(resized, cv2.COLOR_BGR2RGB)

plt.imshow(resized_rgb)
plt.title("Resized Image")
plt.axis("off")
```

Out[33]:
```
(np.float64(-0.5), np.float64(299.5), np.float64(299.5), np.float64(-0.5))
```

Resized Image

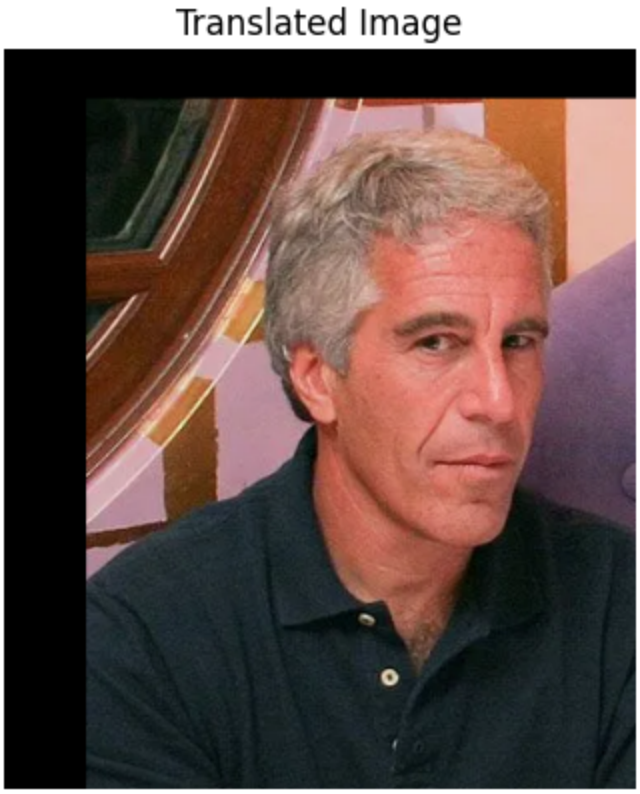
Rotation and Translation

In [34]:
```
(h, w) = img.shape[:2]
center = (w//2, h//2)

matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(img, matrix, (w, h))
rotated_rgb = cv2.cvtColor(rotated, cv2.COLOR_BGR2RGB)

plt.imshow(rotated_rgb)
plt.title("Rotated Image (45 Degrees)")
plt.axis("off")
```

Out[34]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

In [35]:
```
matrix_t = np.float32([[1, 0, 50],
                       [0, 1, 30]])

translated = cv2.warpAffine(img, matrix_t, (w, h))
translated_rgb = cv2.cvtColor(translated, cv2.COLOR_BGR2RGB)

plt.imshow(translated_rgb)
plt.title("Translated Image")
plt.axis("off")
```

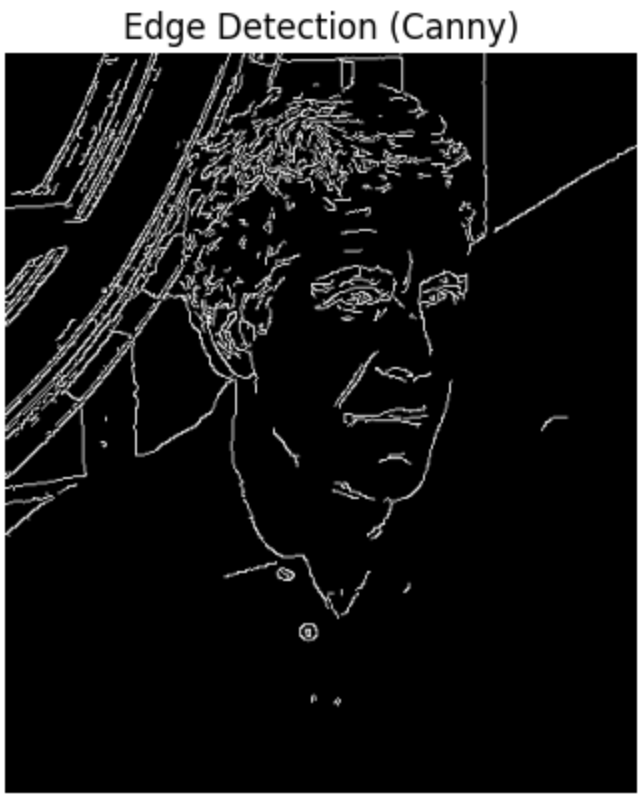
Out[35]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



Additional Basic Operations

In [36]:
```
edges = cv2.Canny(gray, 100, 200)
plt.imshow(edges, cmap='gray')
plt.title("Edge Detection (Canny)")
plt.axis("off")
```

Out[36]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```



In [37]:
```
blur = cv2.GaussianBlur(img, (7,7), 0)
blur_rgb = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)

plt.imshow(blur_rgb)
plt.title("Gaussian Blurred Image")
plt.axis("off")
```

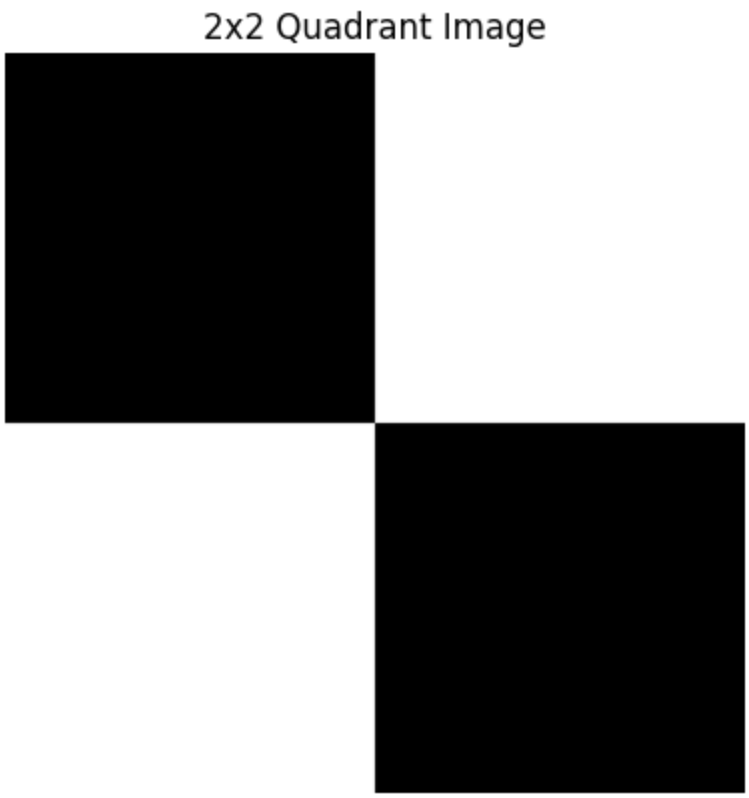
Out[37]:
```
(np.float64(-0.5), np.float64(383.5), np.float64(449.5), np.float64(-0.5))
```

In [39]:
```
img1 = np.zeros((256, 256), dtype=np.uint8)

for i in range(256):
    for j in range(256):
        if j > i:
            img1[i, j] = 255   # White
        else:
            img1[i, j] = 0     # Black

plt.imshow(img1, cmap='gray')
plt.title("Diagonal Half Black–White Image")
plt.axis("off")
```

Out[39]:
```
(np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

In [40]:
```
img2 = np.zeros((256, 256), dtype=np.uint8)

img2[0:128, 0:128] = 0
img2[0:128, 128:256] = 255
img2[128:256, 0:128] = 255
img2[128:256, 128:256] = 0
plt.imshow(img2, cmap='gray')
plt.title("2x2 Quadrant Image")
plt.axis("off")
```

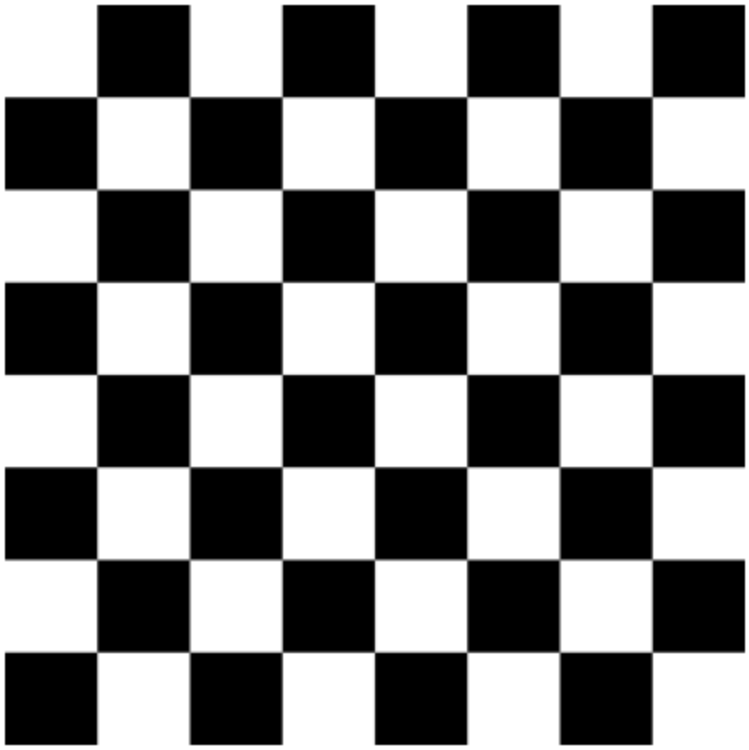
Out[40]:
```
(np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```



In [41]:
```
img3 = np.zeros((256, 256), dtype=np.uint8)
block = 32

for i in range(0, 256, block):
    for j in range(0, 256, block):
        if (i//block + j//block) % 2 == 0:
            img3[i:i+block, j:j+block] = 255

plt.imshow(img3, cmap='gray')
plt.title("Chessboard Pattern")
plt.axis("off")
```

Out[41]:
```
(np.float64(-0.5), np.float64(255.5), np.float64(255.5), np.float64(-0.5))
```

Chessboard Pattern



Algorithm: Piecewise Linear Transformation

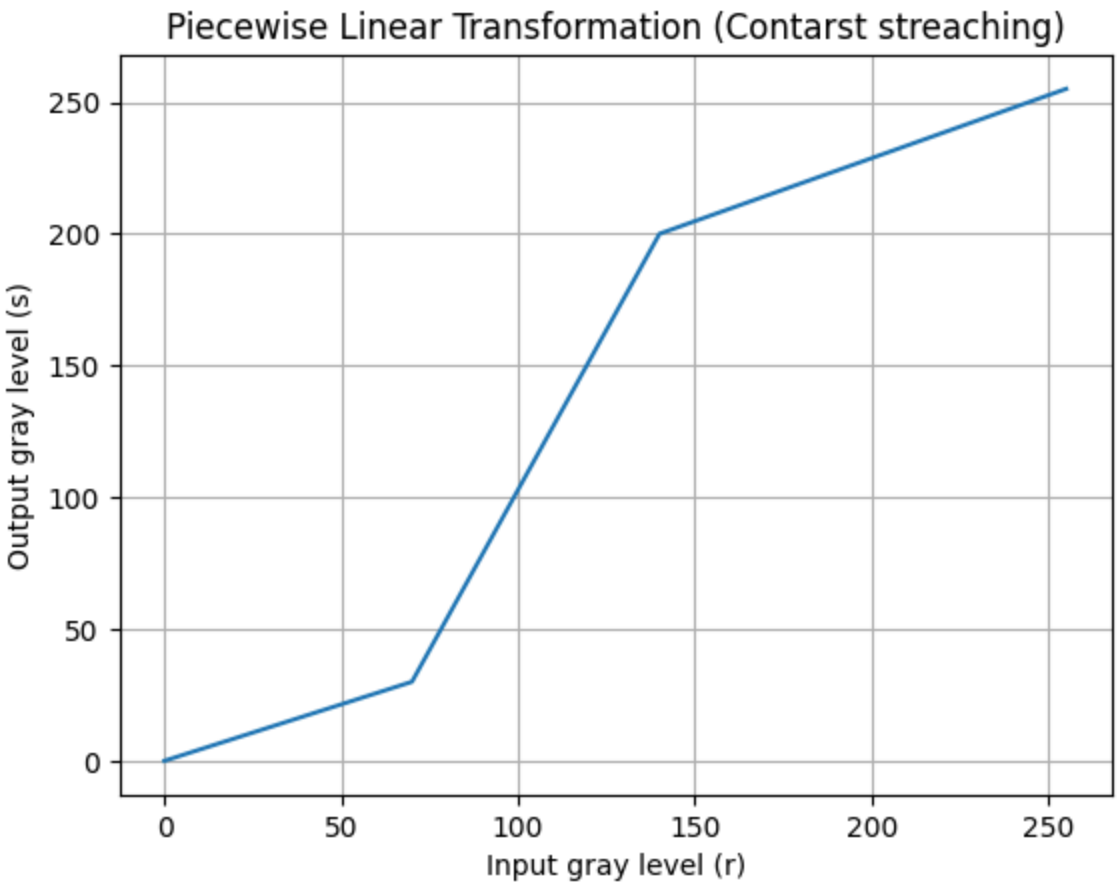
In [67]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [68]:
```
r = np.arange(256)
s = np.zeros(256)
```

In [69]:
```
r1 , r2 = 70, 140
s1 , s2 = 30 , 200
```

In [70]:
```
for i in range(256):
    if r[i] < r1:
        s[i] = (s1/r1) * r[i]
    elif r[i] <= r2:
        s[i] = ((s2 - s1)/(r2 - r1)) * (r[i] - r1) + s1
    else:
        s[i] = ((255 - s2)/(255 - r2)) * (r[i] - r2) + s2
```

In [71]:
```
plt.plot(r, s)
plt.xlabel("Input gray level (r)")
plt.ylabel("Output gray level (s)")
plt.title("Piecewise Linear Transformation (Contarst streaching)")
plt.grid()
plt.show()
```



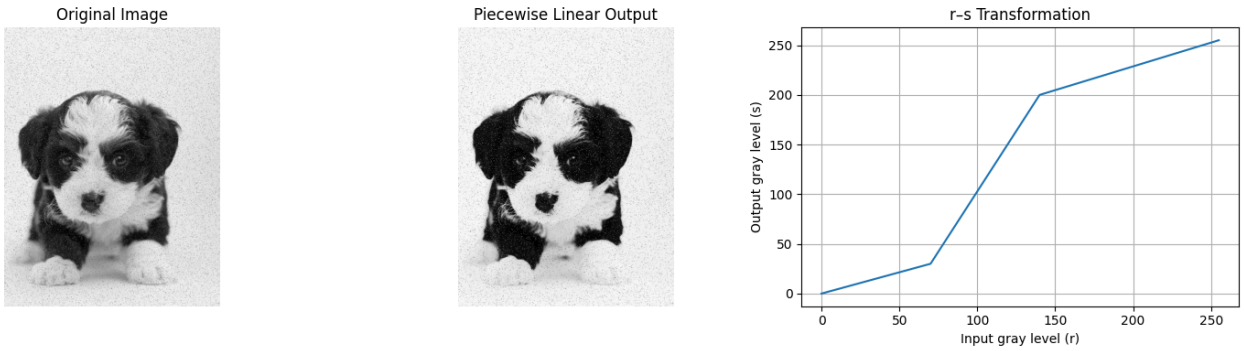
In [72]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\puppy-with-gray-scale-noise.we
    0
)
```

Piecewise Linear Transformation (ON IMAGE)

In [83]:
```
r_vals = np.arange(256)
s_vals = np.zeros(256)

r1, r2 = 70, 140
s1, s2 = 30, 200

for i in range(256):
    if r_vals[i] < r1:
        s_vals[i] = (s1 / r1) * r_vals[i]
    elif r_vals[i] <= r2:
```

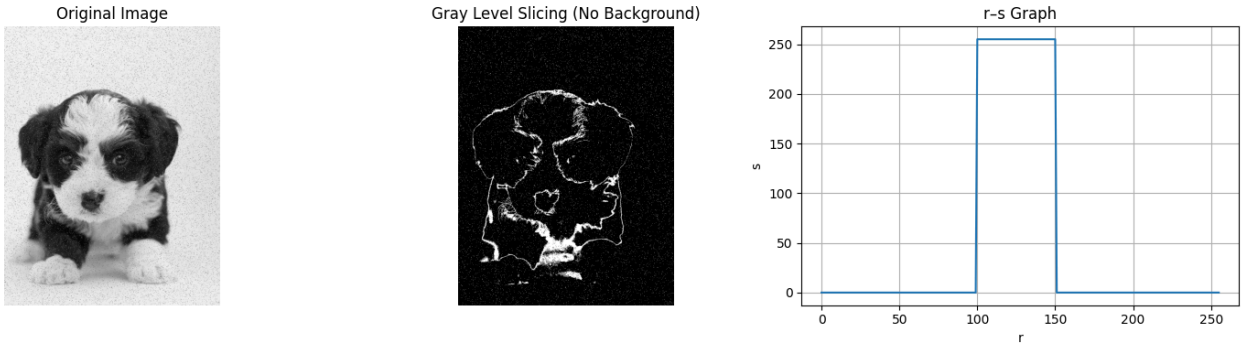
```
        s_vals[i] = ((s2 - s1) / (r2 - r1)) * (r_vals[i] - r1) + s1
    else:
        s_vals[i] = ((255 - s2) / (255 - r2)) * (r_vals[i] - r2) + s2



plt.figure(figsize=(15, 4))


plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")


plt.subplot(1, 3, 2)
plt.imshow(piecewise, cmap='gray')
plt.title("Piecewise Linear Output")
plt.axis("off")


plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("Input gray level (r)")
plt.ylabel("Output gray level (s)")
plt.title("r–s Transformation")
plt.grid()

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



Gray Level Slicing WITHOUT Background

In [85]:
```
r_vals = np.arange(256)
s_vals = np.zeros(256)

r1, r2 = 100, 150

for i in range(256):
```

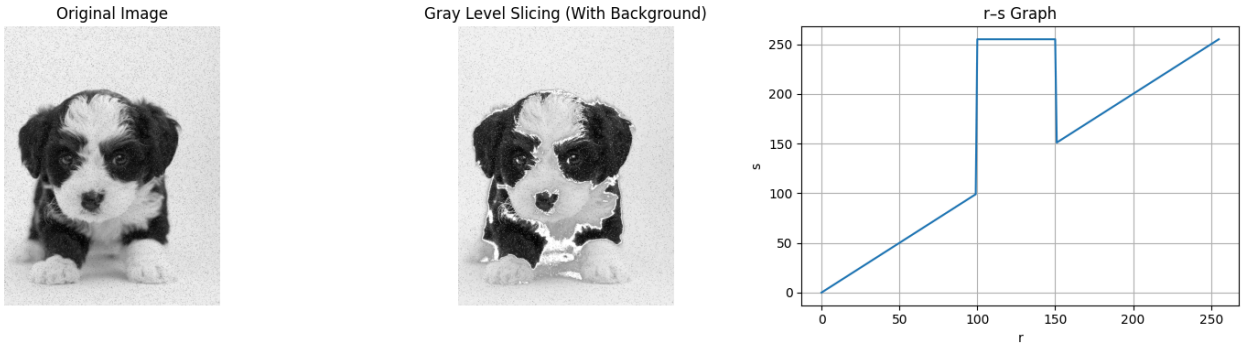
```
    if r1 <= r_vals[i] <= r2:
        s_vals[i] = 255
    else:
        s_vals[i] = 0
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(slice_no_bg, cmap='gray')
plt.title("Gray Level Slicing (No Background)")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r–s Graph")
plt.grid()

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



Gray Level Slicing WITH Background

In [86]:
```
r_vals = np.arange(256)
s_vals = np.zeros(256)

for i in range(256):
    if r1 <= r_vals[i] <= r2:
        s_vals[i] = 255
    else:
        s_vals[i] = r_vals[i]
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
```

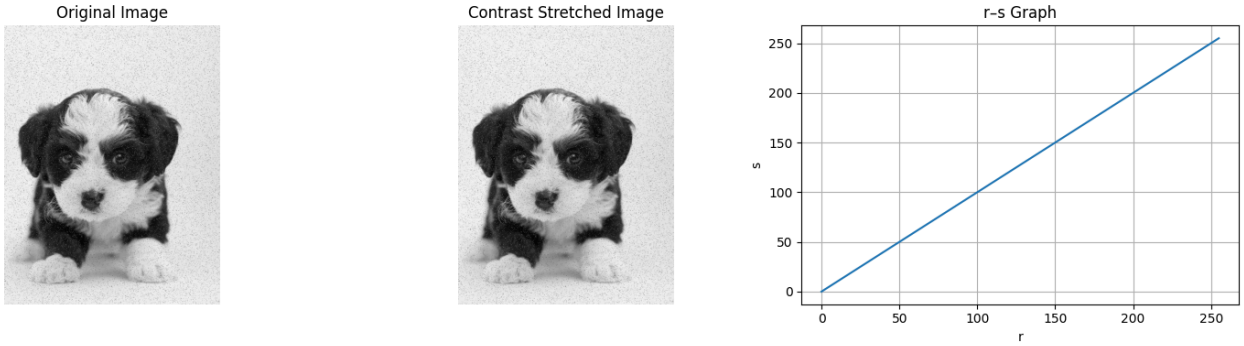
```
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(slice_with_bg, cmap='gray')
plt.title("Gray Level Slicing (With Background)")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r–s Graph")
plt.grid()

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



Contrast Stretching

In [87]:
```
r_vals = np.arange(256)
s_vals = ((r_vals - r_min) / (r_max - r_min)) * 255
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(contrast_stretch, cmap='gray')
plt.title("Contrast Stretched Image")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r–s Graph")
plt.grid()
```

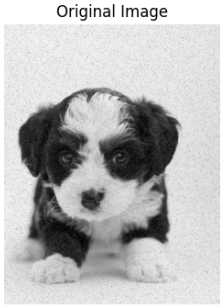
```
plt.tight_layout()
plt.show()
```



Thresholding

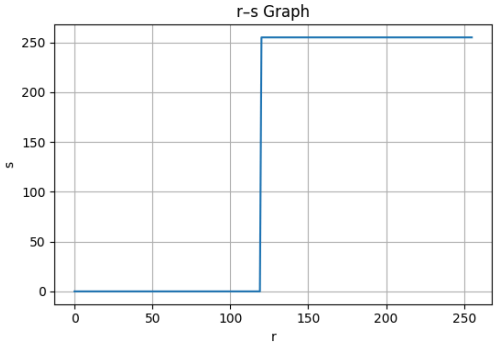
In [88]:
```
T = 120
r_vals = np.arange(256)
s_vals = np.where(r_vals >= T, 255, 0)
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(threshold_img, cmap='gray')
plt.title("Thresholded Image")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.plot(r_vals, s_vals)
plt.xlabel("r")
plt.ylabel("s")
plt.title("r–s Graph")
plt.grid()

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

Original Image



Thresholded Image

r–s Graph



Uzair Shaikh
1032240100

IPPR

* Exp-1 8/1/26 22/1

* Post lab Questions.

1) What do you mean by gray level?

→ In image processing, a grey level refers to the intensity value of a pixel in a grayscale image. It tells you how bright or dark pixel is.

2) Write the expression to find the number of bits to store a digital image.

→ The expression to store digital image using number of bits is.

Num of bits = M x N x K

M - number of rows.
N - number of columns.
K - number of bits per pixel (bpp)

3) Name types of resolutions w.r.t a digital image.

→ The main types of resolution w.r.to a digital image are.

1) Spatial resolution
2) Grey level (intensity) Resolution
3) Temporal resolution.
4) Physical level resolution.

4) Specify the elements of the DIP system.

→ A typical Digital Image processing system includes.

1) Image Acquisition
2) Image storage
3) Image Processing
4) Image Display
5) Image Transmission
6) Image Interpretation

Figure

5) Write any four application of DIP

→ Common application of DIP are.

1) Medical Imaging (e.g, CT, MRI)
2) Remote sensing (satellite)
3) Robotics & Machine vision
4) Bio metrics (Face, fingerprint).

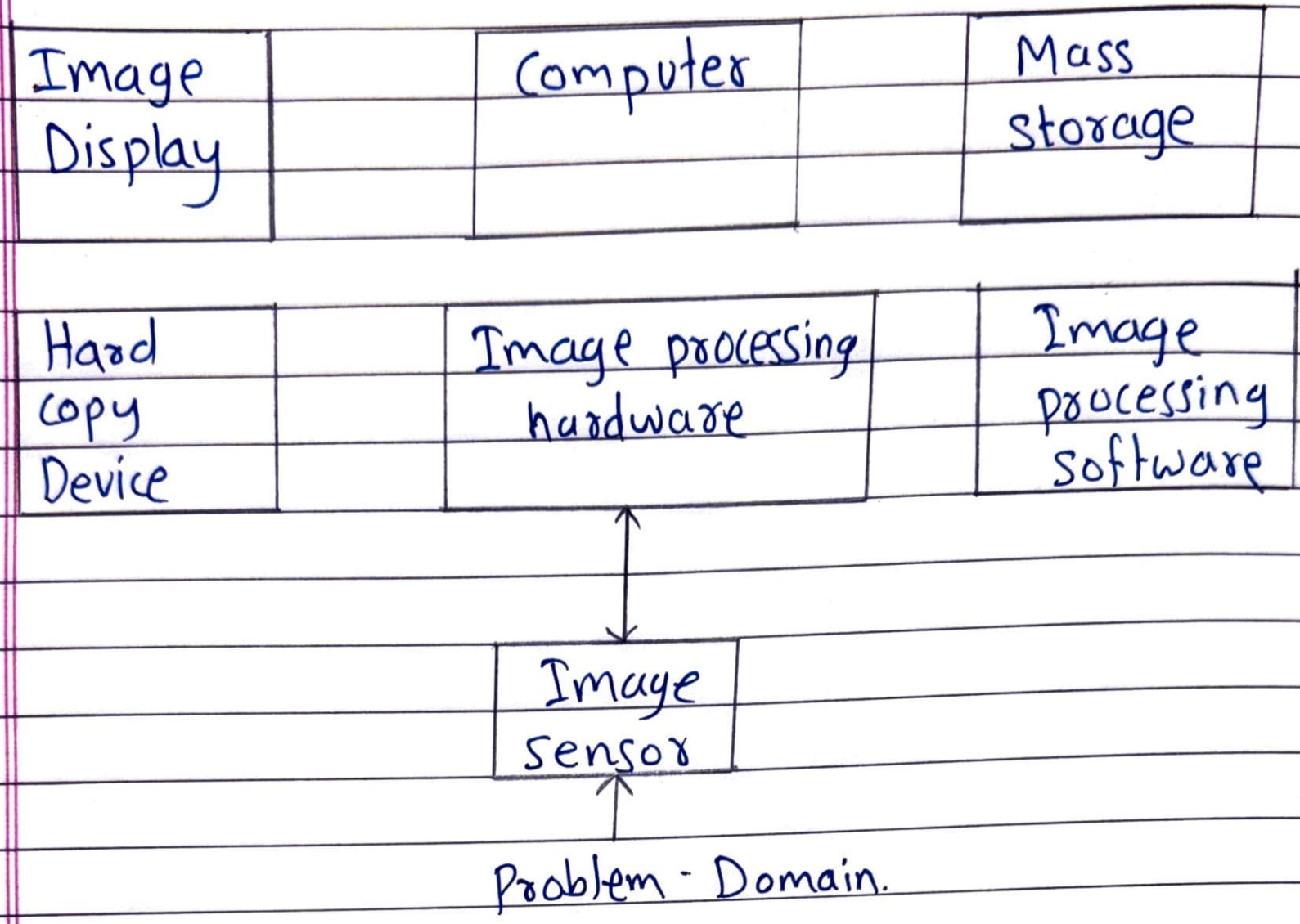


Image Display
Computer
Mass storage
Hard copy Device
Image processing hardware
Image processing software
Image sensor
Problem - Domain.

Part 1

In [43]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
```

In [44]:
```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Image-Processing-and-Pattern-R
    cv2.IMREAD_GRAYSCALE
)

if img is None:
    print("Image not found or path is incorrect!")
else:
    print("Image loaded successfully!")
    plt.imshow(img, cmap='gray')
    plt.title("Original Grayscale Image")
    plt.axis("off")
```

```
Image loaded successfully!
```

Gaussian noise

In [45]:
```
mean = 0
std = 25
```

```
gaussian_noise = np.random.normal(mean, std, img.shape)
noisy_img = img.astype(np.float32) + gaussian_noise
noisy_img = np.clip(noisy_img, 0, 255).astype(np.uint8)

# Display noisy image
plt.imshow(noisy_img, cmap='gray')
plt.title("Image with Gaussian Noise")
plt.axis("off")
```

Out[45]: (np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))

Add Salt & Pepper Noise

In [46]:
```
salt_pepper_noisy = img.copy()
prob = 0.02

rand = np.random.rand(*img.shape)

salt_pepper_noisy[rand < prob/2] = 0
salt_pepper_noisy[rand > 1 - prob/2] = 255

plt.imshow(salt_pepper_noisy, cmap='gray')
plt.title("Salt & Pepper Noise")
plt.axis("off")
```

Out[46]: (np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))

Salt & Pepper Noise

Add Uniform Noise

In [47]:
```
low = -30
high = 30

uniform_noise = np.random.uniform(low, high, img.shape)
uniform_noisy = img + uniform_noise

uniform_noisy = np.clip(uniform_noisy, 0, 255).astype(np.uint8)

plt.imshow(uniform_noisy, cmap='gray')
plt.title("Uniform Noise")
plt.axis("off")
```

Out[47]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

In [48]:
```
import cv2

kernel_size = 5
avg_filtered_img = cv2.blur(noisy_img, (kernel_size, kernel_size))
import matplotlib.pyplot as plt

plt.imshow(avg_filtered_img, cmap='gray')
plt.title("Averaging Filtered Image")
plt.axis("off")
```

Out[48]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

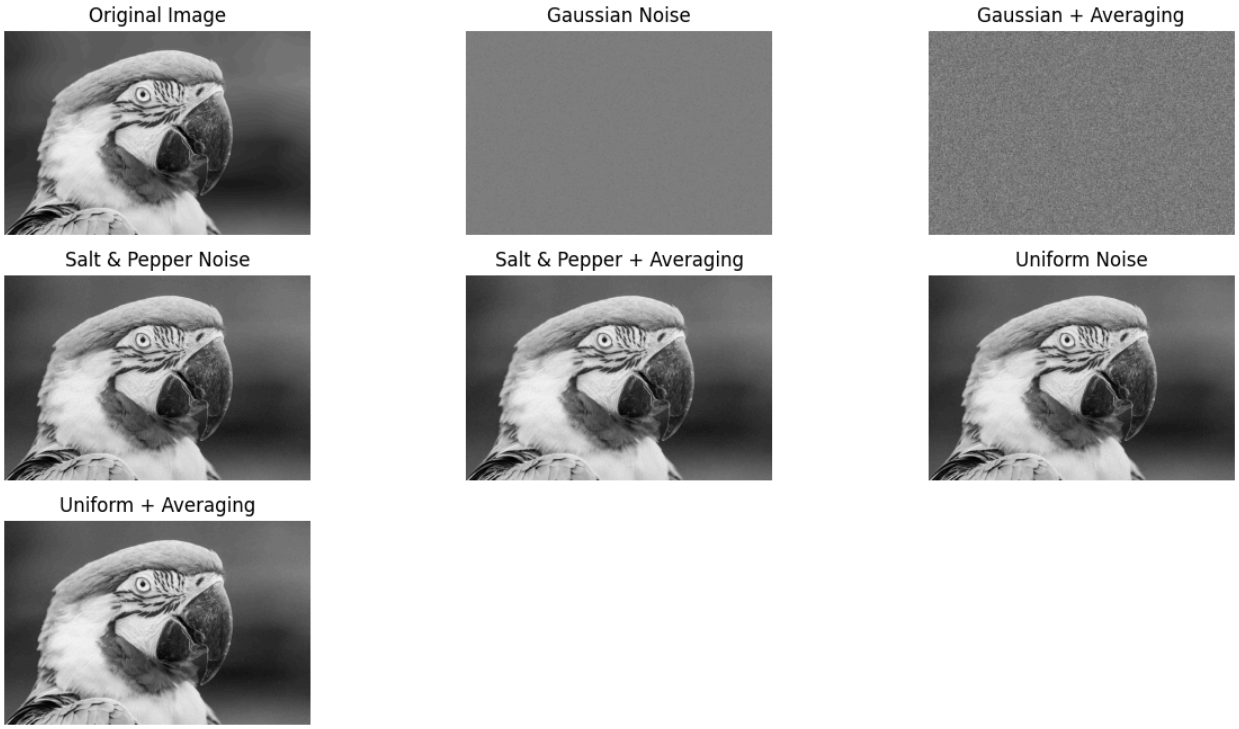
Averaging Filtered Image

Apply avg filter

In [49]:
```
kernel_size = 5

gaussian_avg = cv2.blur(gaussian_noise, (kernel_size, kernel_size))
salt_pepper_avg = cv2.blur(salt_pepper_noisy, (kernel_size, kernel_size))
uniform_avg = cv2.blur(uniform_noisy, (kernel_size, kernel_size))
```

In [50]:
```
plt.figure(figsize=(15,8))

plt.subplot(3,3,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(3,3,2)
plt.imshow(gaussian_noise, cmap='gray')
plt.title("Gaussian Noise")
plt.axis("off")

plt.subplot(3,3,3)
plt.imshow(gaussian_avg, cmap='gray')
plt.title("Gaussian + Averaging")
plt.axis("off")

plt.subplot(3,3,4)
plt.imshow(salt_pepper_noisy, cmap='gray')
plt.title("Salt & Pepper Noise")
plt.axis("off")
```

```
plt.subplot(3,3,5)
plt.imshow(salt_pepper_avg, cmap='gray')
plt.title("Salt & Pepper + Averaging")
plt.axis("off")

plt.subplot(3,3,6)
plt.imshow(uniform_noisy, cmap='gray')
plt.title("Uniform Noise")
plt.axis("off")

plt.subplot(3,3,7)
plt.imshow(uniform_avg, cmap='gray')
plt.title("Uniform + Averaging")
plt.axis("off")

plt.show()
```



Compare Median Filtered Image with Averaging Filtered Image

In [51]:
```
median_filtered_img = cv2.medianBlur(noisy_img, 5)


plt.figure(figsize=(12,4))

plt.subplot(1,3,1)
plt.imshow(noisy_img, cmap='gray')
plt.title("Noisy Image")
plt.axis("off")
```

```
plt.subplot(1,3,2)
plt.imshow(avg_filtered_img, cmap='gray')
plt.title("Averaging Filtered Image")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(median_filtered_img, cmap='gray')
plt.title("Median Filtered Image")
plt.axis("off")

plt.show()
```

Masking

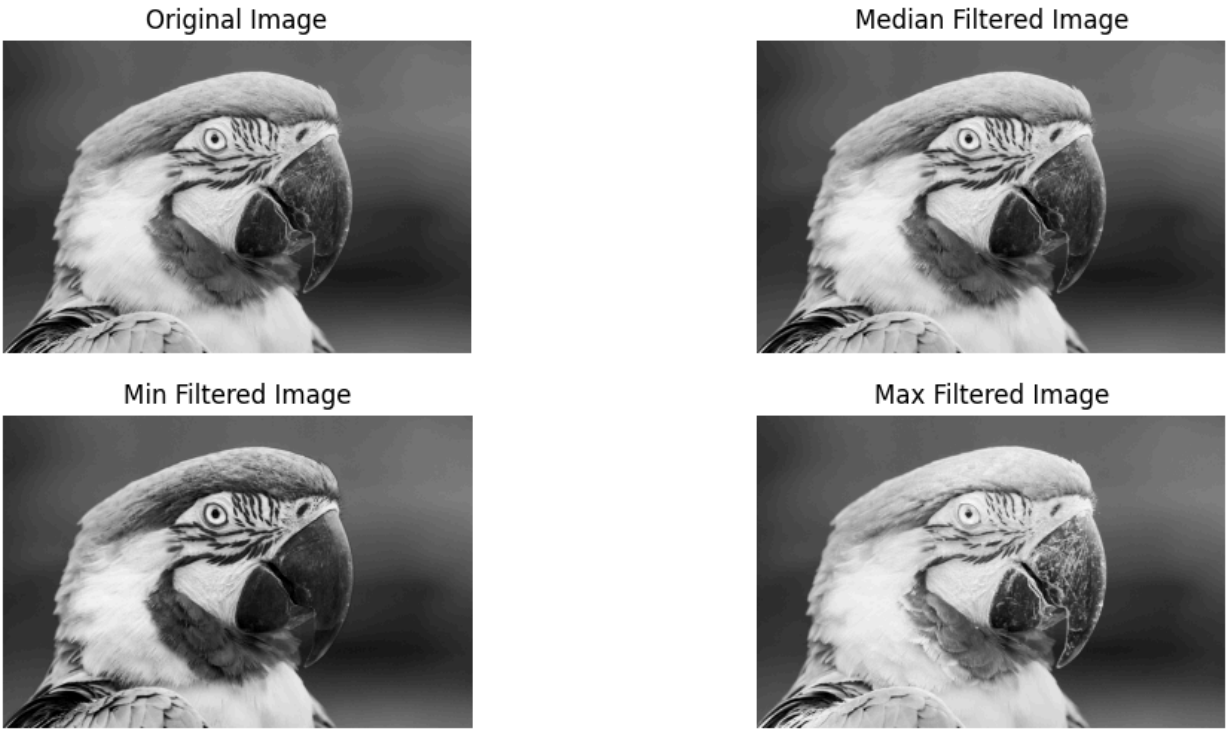
In [52]:
```
mask_sizes = [5, 7, 9, 11, 13, 15]

plt.figure(figsize=(15,8))


plt.subplot(2,4,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

for i, k in enumerate(mask_sizes):
    smoothed = cv2.blur(img, (k, k))

    plt.subplot(2,4,i+2)
    plt.imshow(smoothed, cmap='gray')
    plt.title(f"Averaging {k}×{k}")
    plt.axis("off")

plt.show()
```

Apply Nonlinear Filters (Median, Min, Max)

In [53]:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np


k = 5


median_img = cv2.medianBlur(img, k)


kernel = np.ones((k, k), np.uint8)
min_img = cv2.erode(img, kernel)


max_img = cv2.dilate(img, kernel)
plt.figure(figsize=(12,6))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(2,2,2)
plt.imshow(median_img, cmap='gray')
plt.title("Median Filtered Image")
plt.axis("off")

plt.subplot(2,2,3)
plt.imshow(min_img, cmap='gray')
plt.title("Min Filtered Image")
plt.axis("off")
```

```
plt.subplot(2,2,4)
plt.imshow(max_img, cmap='gray')
plt.title("Max Filtered Image")
plt.axis("off")

plt.show()
```



In [54]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt


mean = 0
std = 25
gaussian_noise = np.random.normal(mean, std, img.shape)
gaussian_noisy = img.astype(np.float32) + gaussian_noise
gaussian_noisy = np.clip(gaussian_noisy, 0, 255).astype(np.uint8)


salt_pepper_noisy = img.copy()
prob = 0.02
rand = np.random.rand(*img.shape)
salt_pepper_noisy[rand < prob/2] = 0
salt_pepper_noisy[rand > 1 - prob/2] = 255


uniform_noise = np.random.uniform(-30, 30, img.shape)
uniform_noisy = img.astype(np.float32) + uniform_noise
uniform_noisy = np.clip(uniform_noisy, 0, 255).astype(np.uint8)
```

In [56]:

```
k = 5
kernel = np.ones((k, k), np.uint8)


gaussian_median = cv2.medianBlur(gaussian_noisy, k)
salt_pepper_median = cv2.medianBlur(salt_pepper_noisy, k)
uniform_median = cv2.medianBlur(uniform_noisy, k)


gaussian_min = cv2.erode(gaussian_noisy, kernel)
salt_pepper_min = cv2.erode(salt_pepper_noisy, kernel)
uniform_min = cv2.erode(uniform_noisy, kernel)


gaussian_max = cv2.dilate(gaussian_noisy, kernel)
salt_pepper_max = cv2.dilate(salt_pepper_noisy, kernel)
uniform_max = cv2.dilate(uniform_noisy, kernel)

plt.figure(figsize=(12,14))


plt.subplot(5,3,1); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a
plt.subplot(5,3,2); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a
plt.subplot(5,3,3); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a


plt.subplot(5,3,4); plt.imshow(gaussian_noisy, cmap='gray'); plt.title("Gaussi
plt.subplot(5,3,5); plt.imshow(salt_pepper_noisy, cmap='gray'); plt.title("Sal
plt.subplot(5,3,6); plt.imshow(uniform_noisy, cmap='gray'); plt.title("Uniform


plt.subplot(5,3,7); plt.imshow(gaussian_median, cmap='gray'); plt.title("Gauss
plt.subplot(5,3,8); plt.imshow(salt_pepper_median, cmap='gray'); plt.title("S&
plt.subplot(5,3,9); plt.imshow(uniform_median, cmap='gray'); plt.title("Unifor

plt.subplot(5,3,10); plt.imshow(gaussian_min, cmap='gray'); plt.title("Gaussia
plt.subplot(5,3,11); plt.imshow(salt_pepper_min, cmap='gray'); plt.title("S&P
plt.subplot(5,3,12); plt.imshow(uniform_min, cmap='gray'); plt.title("Uniform


plt.subplot(5,3,13); plt.imshow(gaussian_max, cmap='gray'); plt.title("Gaussia
plt.subplot(5,3,14); plt.imshow(salt_pepper_max, cmap='gray'); plt.title("S&P
plt.subplot(5,3,15); plt.imshow(uniform_max, cmap='gray'); plt.title("Uniform

plt.show()
```

Part 2

In [ ]:

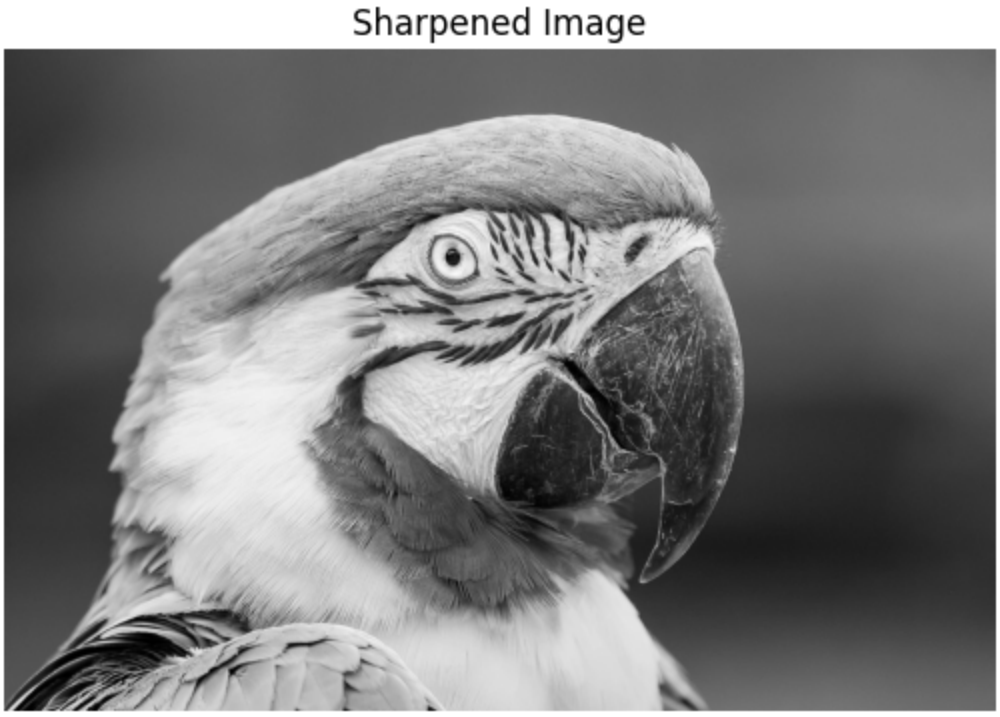
```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Image-Processing-and-Pattern-R
    cv2.IMREAD_GRAYSCALE
```

```
)

if img is None:
    print("Image not found!")
else:
    plt.imshow(img, cmap='gray')
    plt.title("Original Grayscale Image")
    plt.axis("off")
```

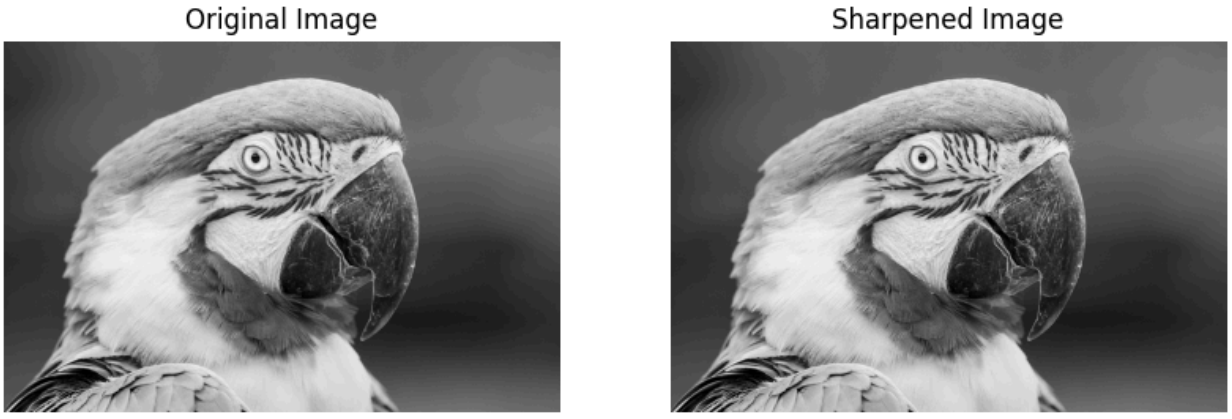
Apply Sharpening Filter on the Given Image

In [58]:
```
import numpy as np


sharpen_kernel = np.array([[ 0, -1,  0],
                           [-1,  5, -1],
                           [ 0, -1,  0]])


sharpened_img = cv2.filter2D(img, -1, sharpen_kernel)


plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")
```

Out[58]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

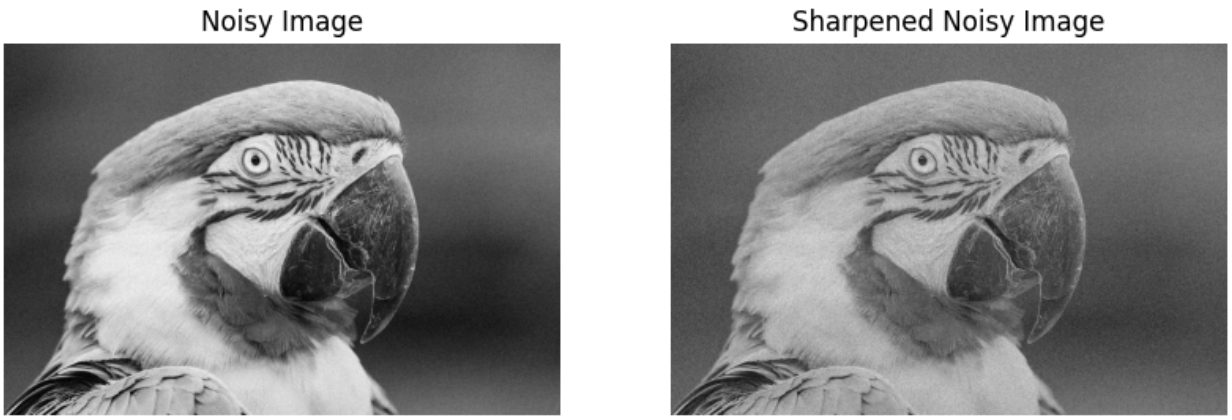


In [59]:
```
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")

plt.show()
```



Apply sharpening filter on the noisy image.

In [61]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt


sharpen_kernel = np.array([[ 0, -1,  0],
                           [-1,  5, -1],
                           [ 0, -1,  0]])
sharpened_noisy_img = cv2.filter2D(noisy_img, -1, sharpen_kernel)
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(noisy_img, cmap='gray')
plt.title("Noisy Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(sharpened_noisy_img, cmap='gray')
plt.title("Sharpened Noisy Image")
plt.axis("off")

plt.show()
```



Display original and sharpened images.

In [62]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))


plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")
```

```
plt.subplot(1,2,2)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")

plt.show()
```

Mean Squared Error (MSE)

In [63]:
```
import numpy as np

mse = np.mean((img.astype(np.float32) - sharpened_img.astype(np.float32)) ** 2

print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 340.6078
```

Peak Signal-to-Noise Ratio (PSNR)

In [64]:
```
import math

if mse == 0:
    psnr = float('inf')
else:
    max_pixel = 255.0
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))

print("Peak Signal-to-Noise Ratio (PSNR):", psnr, "dB")
```

```
Peak Signal-to-Noise Ratio (PSNR): 22.808257868711635 dB
```

IPPR

Name: Uzair Shaikh
PRN : 10322140100.

6/2

* Experiment-3

* Post lab Quetions.

1) Write a note on sharpening Filters.

→ Sharpening filters are image processing techniques used to enchance edge and fine details in an image, making it appear clearer and more defined. They work by emphasizing high-frequency components, which represent rapid changes in intensity such as edges. Common sharping methods inculdes laplacin, sobal, Prewitt, and unsharp masking, each used to highlight important image feacture. Shaping filters are widley applied in medical imaging, remote sensing, and digital photography, but excessive use can amplify noise and cause visual artifacts.

Advantages:

1) Enhances edges and find details
2) Improves visual quality.
3) Useful for feauture extraction.

Disadvantages:

1) Amplifies Noise
2) over-sharpning can cause artifacts and unnatural appierence.

5.

Example:-

10	10	210	210	210
10	10	210	210	210
10	10	210	210	210
10	10	210	210	210
10	10	210	210	210

Mask

0	-1	0
-1	4	-1
0	-1	0

Pixel (row 2, col 2)

$= (4 \times 10) + (10 \times (-1)) + (10 \times (-1)) + (10 \times (-1)) + (210 \times (-1))$

$= -200$

new image

10	10	210	210	210
10	-200	200	0	210
10	-200	200	0	210
10	-200	200	0	210
10	10	210	210	210

2. Consider an input image row [4 3 2 1] with inter sity values in range 0 to 15. Determine negative of the image.

→ [4 3 2 1]

Negative pixel = $(L-1) - f(x, y)$

For 4 → 15 - 4 = 11

For 3 → 15 - 3 = 12

for 2 → 15 - 2 = 13

For 1 → 15 - 1 = 14

new row

[11 12 13 14]

3. Determine the following value of the central pixel of the following image by applying a 3×3 size.

→
10	11	11
10	255	11
12	12	11

a) Mean filter

$$\text{Mean} = \frac{1}{n} \sum f(x_i y_i)$$

$$= \frac{10+10+12+11+11+11+12+255}{9}$$

$$= 38.11 \approx 38$$

new image

10	11	11
10	38	11
12	12	11

b) Median filter

10, 10, 11, 11, 11, 11, 12, 12, 255

median = 11

new image:

10	11	11
10	11	11
12	12	11

c) Mode filter

count →

- 10 → 2 times
- 11 → 4 times
- 12 → 2 times
- 255 → 1 times

most frequent → 11

10	11	11
10	11	11
12	12	11

Part 1

In [43]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
```

In [44]:
```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Image-Processing-and-Pattern-R
    cv2.IMREAD_GRAYSCALE
)

if img is None:
    print("Image not found or path is incorrect!")
else:
    print("Image loaded successfully!")
    plt.imshow(img, cmap='gray')
    plt.title("Original Grayscale Image")
    plt.axis("off")
```

```
Image loaded successfully!
```

Gaussian noise

In [45]:
```
mean = 0
std = 25
```

```
gaussian_noise = np.random.normal(mean, std, img.shape)
noisy_img = img.astype(np.float32) + gaussian_noise
noisy_img = np.clip(noisy_img, 0, 255).astype(np.uint8)

# Display noisy image
plt.imshow(noisy_img, cmap='gray')
plt.title("Image with Gaussian Noise")
plt.axis("off")
```

Out[45]: (np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))

Add Salt & Pepper Noise

In [46]:
```
salt_pepper_noisy = img.copy()
prob = 0.02

rand = np.random.rand(*img.shape)

salt_pepper_noisy[rand < prob/2] = 0
salt_pepper_noisy[rand > 1 - prob/2] = 255

plt.imshow(salt_pepper_noisy, cmap='gray')
plt.title("Salt & Pepper Noise")
plt.axis("off")
```

Out[46]: (np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))

Salt & Pepper Noise

Add Uniform Noise

In [47]:
```
low = -30
high = 30

uniform_noise = np.random.uniform(low, high, img.shape)
uniform_noisy = img + uniform_noise

uniform_noisy = np.clip(uniform_noisy, 0, 255).astype(np.uint8)

plt.imshow(uniform_noisy, cmap='gray')
plt.title("Uniform Noise")
plt.axis("off")
```

Out[47]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

In [48]:
```
import cv2

kernel_size = 5
avg_filtered_img = cv2.blur(noisy_img, (kernel_size, kernel_size))
import matplotlib.pyplot as plt

plt.imshow(avg_filtered_img, cmap='gray')
plt.title("Averaging Filtered Image")
plt.axis("off")
```

Out[48]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

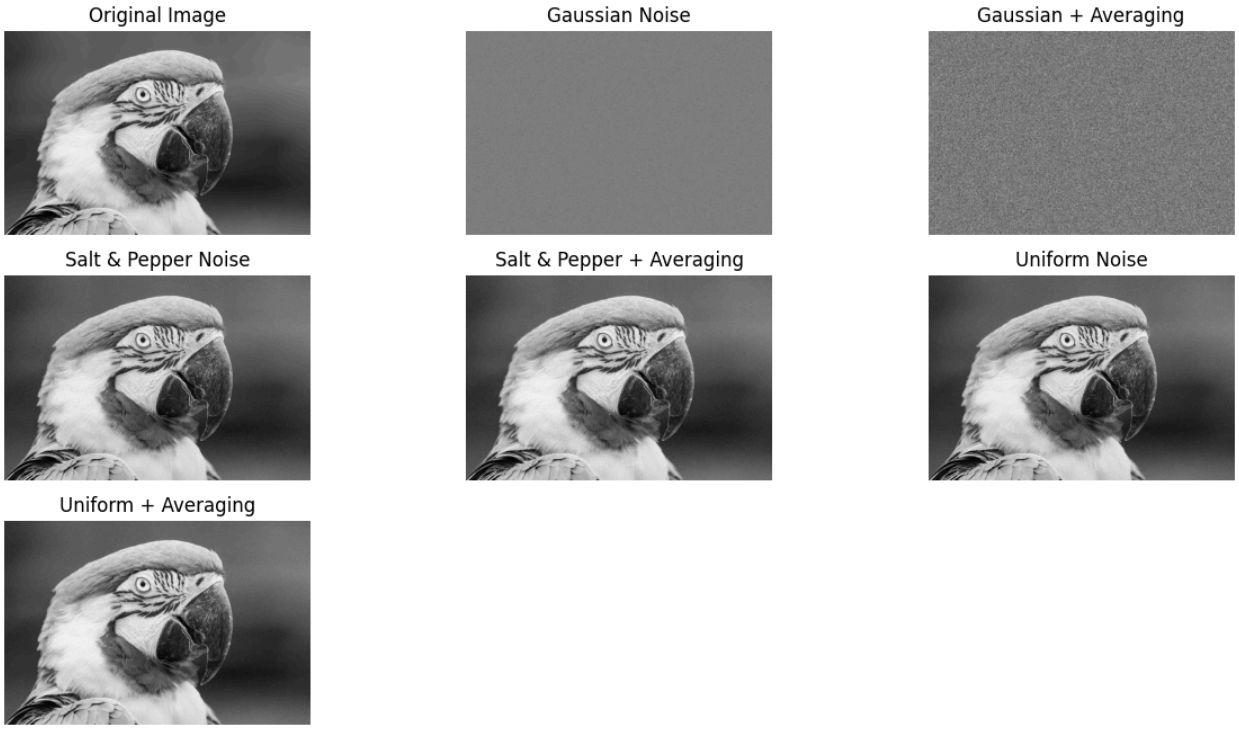
Averaging Filtered Image

Apply avg filter

In [49]:
```
kernel_size = 5

gaussian_avg = cv2.blur(gaussian_noise, (kernel_size, kernel_size))
salt_pepper_avg = cv2.blur(salt_pepper_noisy, (kernel_size, kernel_size))
uniform_avg = cv2.blur(uniform_noisy, (kernel_size, kernel_size))
```

In [50]:
```
plt.figure(figsize=(15,8))

plt.subplot(3,3,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(3,3,2)
plt.imshow(gaussian_noise, cmap='gray')
plt.title("Gaussian Noise")
plt.axis("off")

plt.subplot(3,3,3)
plt.imshow(gaussian_avg, cmap='gray')
plt.title("Gaussian + Averaging")
plt.axis("off")

plt.subplot(3,3,4)
plt.imshow(salt_pepper_noisy, cmap='gray')
plt.title("Salt & Pepper Noise")
plt.axis("off")
```

```
plt.subplot(3,3,5)
plt.imshow(salt_pepper_avg, cmap='gray')
plt.title("Salt & Pepper + Averaging")
plt.axis("off")

plt.subplot(3,3,6)
plt.imshow(uniform_noisy, cmap='gray')
plt.title("Uniform Noise")
plt.axis("off")

plt.subplot(3,3,7)
plt.imshow(uniform_avg, cmap='gray')
plt.title("Uniform + Averaging")
plt.axis("off")

plt.show()
```



Compare Median Filtered Image with Averaging Filtered Image

In [51]:
```
median_filtered_img = cv2.medianBlur(noisy_img, 5)


plt.figure(figsize=(12,4))

plt.subplot(1,3,1)
plt.imshow(noisy_img, cmap='gray')
plt.title("Noisy Image")
plt.axis("off")
```

```
plt.subplot(1,3,2)
plt.imshow(avg_filtered_img, cmap='gray')
plt.title("Averaging Filtered Image")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(median_filtered_img, cmap='gray')
plt.title("Median Filtered Image")
plt.axis("off")

plt.show()
```

Masking

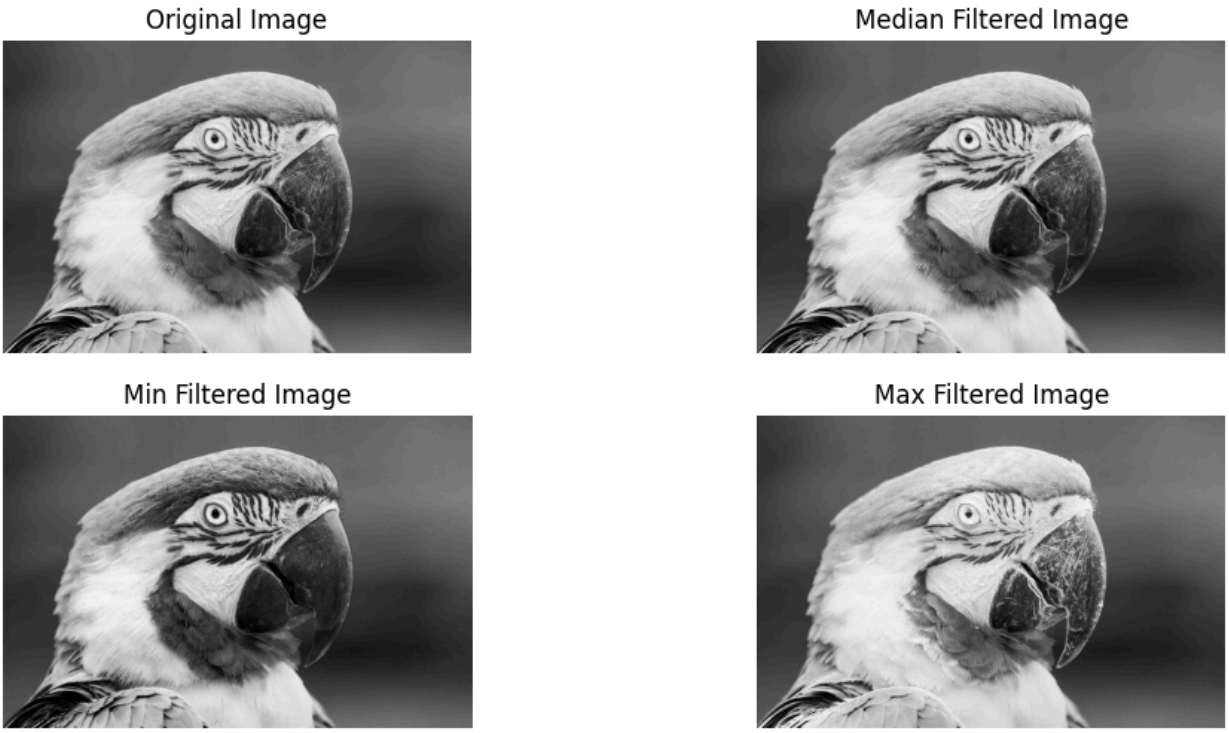
In [52]:
```
mask_sizes = [5, 7, 9, 11, 13, 15]

plt.figure(figsize=(15,8))


plt.subplot(2,4,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

for i, k in enumerate(mask_sizes):
    smoothed = cv2.blur(img, (k, k))

    plt.subplot(2,4,i+2)
    plt.imshow(smoothed, cmap='gray')
    plt.title(f"Averaging {k}×{k}")
    plt.axis("off")

plt.show()
```

Apply Nonlinear Filters (Median, Min, Max)

In [53]:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np


k = 5


median_img = cv2.medianBlur(img, k)


kernel = np.ones((k, k), np.uint8)
min_img = cv2.erode(img, kernel)


max_img = cv2.dilate(img, kernel)
plt.figure(figsize=(12,6))

plt.subplot(2,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(2,2,2)
plt.imshow(median_img, cmap='gray')
plt.title("Median Filtered Image")
plt.axis("off")

plt.subplot(2,2,3)
plt.imshow(min_img, cmap='gray')
plt.title("Min Filtered Image")
plt.axis("off")
```

```
plt.subplot(2,2,4)
plt.imshow(max_img, cmap='gray')
plt.title("Max Filtered Image")
plt.axis("off")

plt.show()
```



In [54]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt


mean = 0
std = 25
gaussian_noise = np.random.normal(mean, std, img.shape)
gaussian_noisy = img.astype(np.float32) + gaussian_noise
gaussian_noisy = np.clip(gaussian_noisy, 0, 255).astype(np.uint8)


salt_pepper_noisy = img.copy()
prob = 0.02
rand = np.random.rand(*img.shape)
salt_pepper_noisy[rand < prob/2] = 0
salt_pepper_noisy[rand > 1 - prob/2] = 255


uniform_noise = np.random.uniform(-30, 30, img.shape)
uniform_noisy = img.astype(np.float32) + uniform_noise
uniform_noisy = np.clip(uniform_noisy, 0, 255).astype(np.uint8)
```

In [56]:

```
k = 5
kernel = np.ones((k, k), np.uint8)


gaussian_median = cv2.medianBlur(gaussian_noisy, k)
salt_pepper_median = cv2.medianBlur(salt_pepper_noisy, k)
uniform_median = cv2.medianBlur(uniform_noisy, k)


gaussian_min = cv2.erode(gaussian_noisy, kernel)
salt_pepper_min = cv2.erode(salt_pepper_noisy, kernel)
uniform_min = cv2.erode(uniform_noisy, kernel)


gaussian_max = cv2.dilate(gaussian_noisy, kernel)
salt_pepper_max = cv2.dilate(salt_pepper_noisy, kernel)
uniform_max = cv2.dilate(uniform_noisy, kernel)

plt.figure(figsize=(12,14))


plt.subplot(5,3,1); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a
plt.subplot(5,3,2); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a
plt.subplot(5,3,3); plt.imshow(img, cmap='gray'); plt.title("Original"); plt.a


plt.subplot(5,3,4); plt.imshow(gaussian_noisy, cmap='gray'); plt.title("Gaussi
plt.subplot(5,3,5); plt.imshow(salt_pepper_noisy, cmap='gray'); plt.title("Sal
plt.subplot(5,3,6); plt.imshow(uniform_noisy, cmap='gray'); plt.title("Uniform


plt.subplot(5,3,7); plt.imshow(gaussian_median, cmap='gray'); plt.title("Gauss
plt.subplot(5,3,8); plt.imshow(salt_pepper_median, cmap='gray'); plt.title("S&
plt.subplot(5,3,9); plt.imshow(uniform_median, cmap='gray'); plt.title("Unifor

plt.subplot(5,3,10); plt.imshow(gaussian_min, cmap='gray'); plt.title("Gaussia
plt.subplot(5,3,11); plt.imshow(salt_pepper_min, cmap='gray'); plt.title("S&P
plt.subplot(5,3,12); plt.imshow(uniform_min, cmap='gray'); plt.title("Uniform


plt.subplot(5,3,13); plt.imshow(gaussian_max, cmap='gray'); plt.title("Gaussia
plt.subplot(5,3,14); plt.imshow(salt_pepper_max, cmap='gray'); plt.title("S&P
plt.subplot(5,3,15); plt.imshow(uniform_max, cmap='gray'); plt.title("Uniform

plt.show()
```

Part 2

In [ ]:

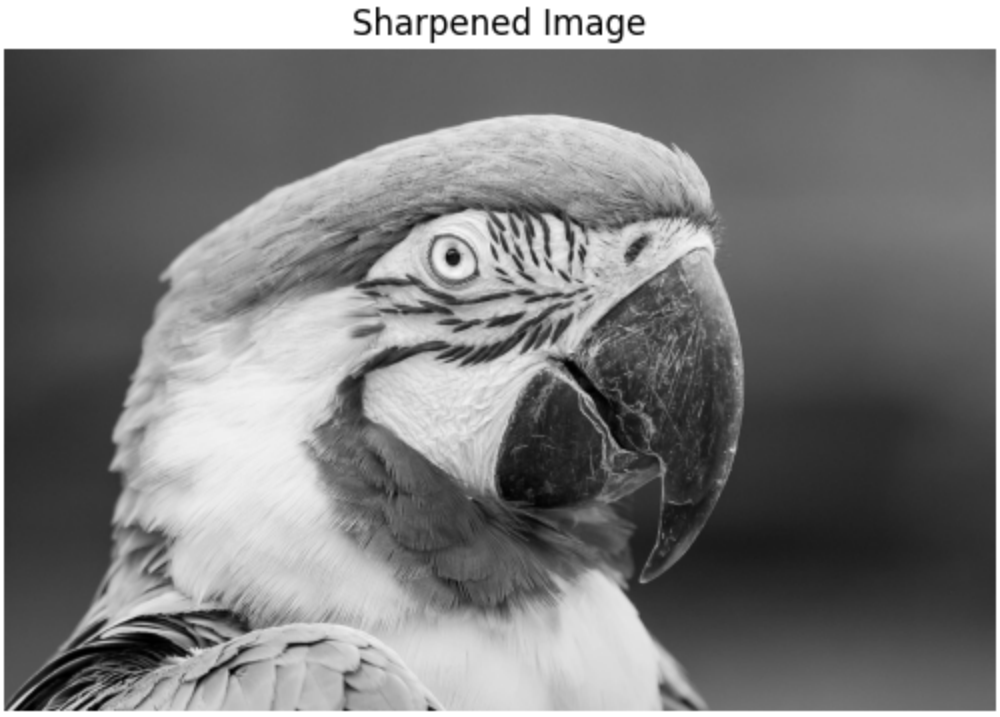
```
import cv2
import matplotlib.pyplot as plt
img = cv2.imread(
    r"D:\MIT WPU B.Tech Data\3nd Year Data\IPPR\Image-Processing-and-Pattern-R
    cv2.IMREAD_GRAYSCALE
```

```
)

if img is None:
    print("Image not found!")
else:
    plt.imshow(img, cmap='gray')
    plt.title("Original Grayscale Image")
    plt.axis("off")
```

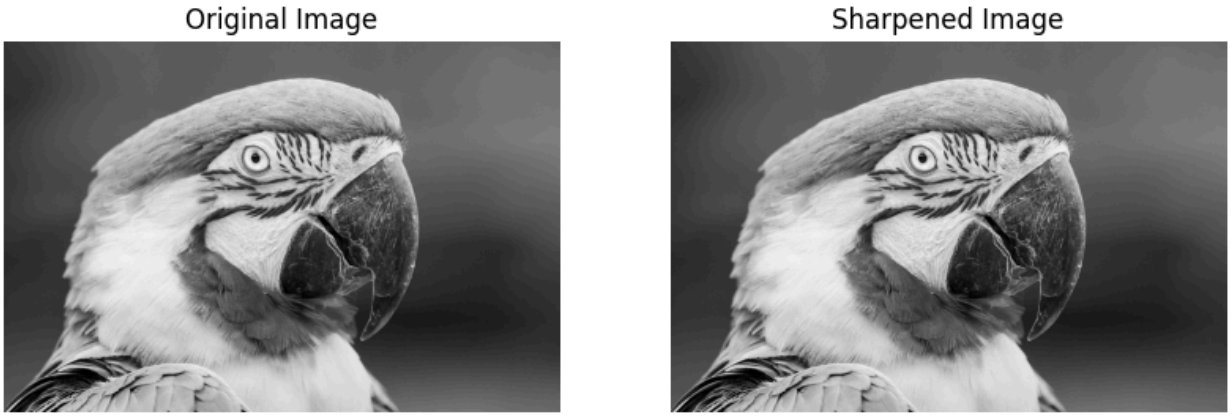
Apply Sharpening Filter on the Given Image

In [58]:
```
import numpy as np


sharpen_kernel = np.array([[ 0, -1,  0],
                           [-1,  5, -1],
                           [ 0, -1,  0]])


sharpened_img = cv2.filter2D(img, -1, sharpen_kernel)


plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")
```

Out[58]:
```
(np.float64(-0.5), np.float64(2930.5), np.float64(1953.5), np.float64(-0.5))
```

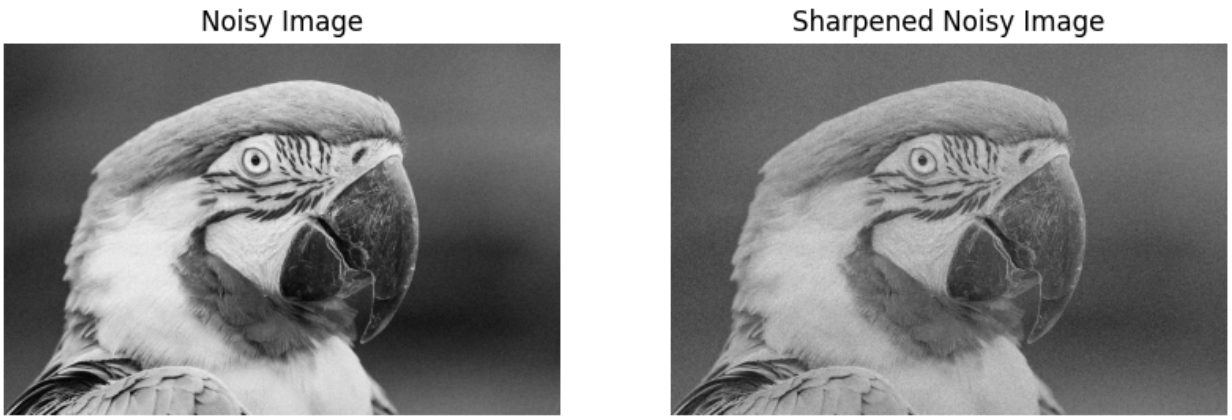


In [59]:
```
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")

plt.show()
```



Apply sharpening filter on the noisy image.

In [61]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt


sharpen_kernel = np.array([[ 0, -1,  0],
                           [-1,  5, -1],
                           [ 0, -1,  0]])
sharpened_noisy_img = cv2.filter2D(noisy_img, -1, sharpen_kernel)
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(noisy_img, cmap='gray')
plt.title("Noisy Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(sharpened_noisy_img, cmap='gray')
plt.title("Sharpened Noisy Image")
plt.axis("off")

plt.show()
```



Display original and sharpened images.

In [62]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))


plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")
```

```
plt.subplot(1,2,2)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis("off")

plt.show()
```

Mean Squared Error (MSE)

In [63]:
```
import numpy as np

mse = np.mean((img.astype(np.float32) - sharpened_img.astype(np.float32)) ** 2

print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 340.6078
```

Peak Signal-to-Noise Ratio (PSNR)

In [64]:
```
import math

if mse == 0:
    psnr = float('inf')
else:
    max_pixel = 255.0
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))

print("Peak Signal-to-Noise Ratio (PSNR):", psnr, "dB")
```

```
Peak Signal-to-Noise Ratio (PSNR): 22.808257868711635 dB
```