# EC7212 – COMPUTER VISION AND IMAGE PROCESSING

**ASSIGNMENT 01**


**NAME** : ABEYSEKARA P.K.
**REG NO** : EG/2020/3799
**SEMESTER** : 07
**DATE** : 21/06/2025

**GitHub Repository Link**
https://github.com/PasanAbeysekara/EC7212-Assignment1

# Task Implementations and Results

## Task 1: Intensity Level Reduction

Code Implementation:

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

IMAGE_PATH = 'sanath.jpg'

def display_image(image, title="Image", cmap=None):
    plt.figure(figsize=(6, 6))
    if cmap:
        plt.imshow(image, cmap=cmap)
    elif len(image.shape) == 2 or image.shape[2] == 1:
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show(block=True)

try:
    img_color_original = cv2.imread(IMAGE_PATH)
    if img_color_original is None:
        raise FileNotFoundError(f"Image not found at {IMAGE_PATH}")
    img_gray_original = cv2.cvtColor(img_color_original, cv2.COLOR_BGR2GRAY)
except FileNotFoundError as e:
    print(e)
    exit()
except Exception as e:
    print(f"Error loading image: {e}")
    exit()

def reduce_intensity_levels(image, num_levels):
    if not (num_levels > 1 and (num_levels & (num_levels - 1) == 0) and
    num_levels <= 256):
        raise ValueError("Number of levels must be a power of 2 between 2 and
    256.")

    image_float = image.astype(float)
    if num_levels == 1:
         output_image = np.zeros_like(image_float)
    else:
        output_image = np.floor(image_float * (num_levels / 256.0)) * (255.0 / (
    num_levels - 1.0))

    output_image = np.clip(output_image, 0, 255).astype(np.uint8)
    return output_image

if __name__ == "__main__":
    print("--- Task 1: Reduce Intensity Levels ---")
    display_image(img_gray_original, "Original Grayscale Image")

    desired_levels = [128, 64, 32, 16, 8, 4, 2]
    for levels in desired_levels:
        try:
            reduced_img = reduce_intensity_levels(img_gray_original.copy(),
    levels)
```

```
            display_image(reduced_img, f"Grayscale Image with {levels} Intensity
    Levels")
        except ValueError as e:
            print(e)
    print("Task 1 Completed.")
```

Listing 1: Intensity Level Reduction Function

**Results:**

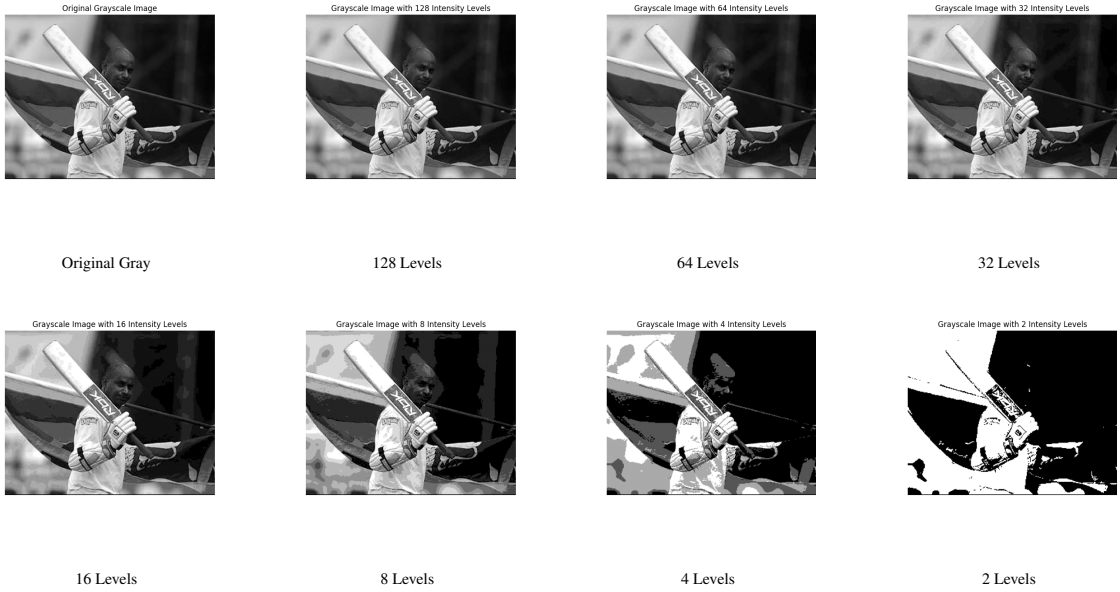


Figure 1: Task 1 - Intensity Level Reduction Results.

## Task 2: Spatial Averaging

Code Implementation:

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

IMAGE_PATH = 'sanath.jpg'

def display_image(image, title="Image", cmap=None):
    plt.figure(figsize=(6, 6))
    if cmap:
        plt.imshow(image, cmap=cmap)
    elif len(image.shape) == 2 or image.shape[2] == 1:
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show(block=True)

try:
    img_color_original = cv2.imread(IMAGE_PATH)
    if img_color_original is None:
        raise FileNotFoundError(f"Image not found at {IMAGE_PATH}")
    img_gray_original = cv2.cvtColor(img_color_original, cv2.COLOR_BGR2GRAY)
except FileNotFoundError as e:
```

```
    print(e)
    exit()
except Exception as e:
    print(f"Error loading image: {e}")
    exit()

def spatial_average(image, kernel_size_tuple):
    blurred_image = cv2.blur(image, kernel_size_tuple)
    return blurred_image

if __name__ == "__main__":
    print("--- Task 2: Spatial Averaging ---")
    display_image(img_color_original, "Original Color Image")

    kernel_sizes = [(3, 3), (10, 10), (20, 20)]
    for k_size in kernel_sizes:
        blurred_img = spatial_average(img_color_original.copy(), k_size)
        display_image(blurred_img, f"Color Image Blurred with {k_size[0]}x{k_size
    [1]} Kernel")

    print("\nApplying to Grayscale for demonstration:")
    display_image(img_gray_original, "Original Grayscale Image")
    for k_size in kernel_sizes:
        blurred_gray_img = spatial_average(img_gray_original.copy(), k_size)
        display_image(blurred_gray_img, f"Grayscale Image Blurred with {k_size
    [0]}x{k_size[1]} Kernel")

    print("Task 2 Completed.")
```

Listing 2: Spatial Averaging Function

**Results:**

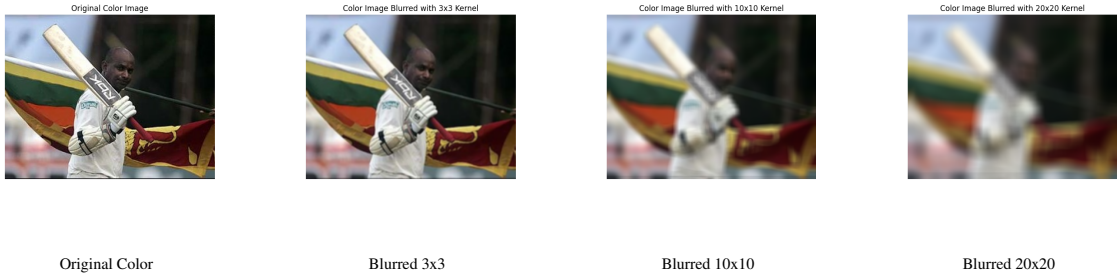


Figure 2: Task 2 - Spatial Averaging Results.

## Task 3: Image Rotation

Code Implementation:

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

IMAGE_PATH = 'sanath.jpg'

def display_image(image, title="Image", cmap=None):
    plt.figure(figsize=(6, 6))
    if cmap:
        plt.imshow(image, cmap=cmap)
    elif len(image.shape) == 2 or image.shape[2] == 1:
```

```
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show(block=True)

try:
    img_color_original = cv2.imread(IMAGE_PATH)
    if img_color_original is None:
        raise FileNotFoundError(f"Image not found at {IMAGE_PATH}")
except FileNotFoundError as e:
    print(e)
    exit()
except Exception as e:
    print(f"Error loading image: {e}")
    exit()

def rotate_image(image, angle):
    height, width = image.shape[:2]
    center = (width / 2, height / 2)

    if angle % 90 == 0:
        if angle == 90 or angle == -270:
            return cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)
        elif angle == 180 or angle == -180:
            return cv2.rotate(image, cv2.ROTATE_180)
        elif angle == 270 or angle == -90:
            return cv2.rotate(image, cv2.ROTATE_90_COUNTERCLOCKWISE)
        elif angle % 360 == 0:
            return image.copy()
        else:
            return image.copy()
    else:
        rad = math.radians(angle)
        sin_a = abs(math.sin(rad))
        cos_a = abs(math.cos(rad))

        new_width = int((height * sin_a) + (width * cos_a))
        new_height = int((height * cos_a) + (width * sin_a))

        M = cv2.getRotationMatrix2D(center, angle, 1.0)

        M[0, 2] += (new_width / 2) - center[0]
        M[1, 2] += (new_height / 2) - center[1]

        rotated_image = cv2.warpAffine(image, M, (new_width, new_height), flags=
    cv2.INTER_LINEAR)
        return rotated_image

if __name__ == "__main__":
    print("--- Task 3: Rotate Image ---")
    display_image(img_color_original, "Original Color Image")

    angles_to_rotate = [45, 90]
    for ang in angles_to_rotate:
        rotated_img = rotate_image(img_color_original.copy(), ang)
        display_image(rotated_img, f"Color Image Rotated by {ang} Degrees")

    print("Task 3 Completed.")
```

Listing 3: Image Rotation Function

**Results:**


Original Color Rotated 45° Rotated 90°

Figure 3: Task 3 - Image Rotation Results.

## Task 4: Block Averaging

Code Implementation:

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

IMAGE_PATH = 'sanath.jpg'

def display_image(image, title="Image", cmap=None):
    plt.figure(figsize=(6, 6))
    if cmap:
        plt.imshow(image, cmap=cmap)
    elif len(image.shape) == 2 or image.shape[2] == 1:
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show(block=True)

try:
    img_color_original = cv2.imread(IMAGE_PATH)
    if img_color_original is None:
        raise FileNotFoundError(f"Image not found at {IMAGE_PATH}")
    img_gray_original = cv2.cvtColor(img_color_original, cv2.COLOR_BGR2GRAY)
except FileNotFoundError as e:
    print(e)
    exit()
except Exception as e:
    print(f"Error loading image: {e}")
    exit()

def block_average_resolution(image, block_size_val):
    height, width = image.shape[:2]
    output_image = image.copy()

    is_color = len(image.shape) == 3

    for r_idx in range(0, height - height % block_size_val, block_size_val):
        for c_idx in range(0, width - width % block_size_val, block_size_val):
            if is_color:
                block = image[r_idx:r_idx+block_size_val, c_idx:c_idx+
   block_size_val, :]
```

```
                if block.size > 0:
                    avg_b = np.mean(block[:,:,0])
                    avg_g = np.mean(block[:,:,1])
                    avg_r = np.mean(block[:,:,2])
                    output_image[r_idx:r_idx+block_size_val, c_idx:c_idx+
    block_size_val, 0] = avg_b
                    output_image[r_idx:r_idx+block_size_val, c_idx:c_idx+
    block_size_val, 1] = avg_g
                    output_image[r_idx:r_idx+block_size_val, c_idx:c_idx+
    block_size_val, 2] = avg_r
            else: # Grayscale
                block = image[r_idx:r_idx+block_size_val, c_idx:c_idx+
    block_size_val]
                if block.size > 0:
                    average = np.mean(block)
                    output_image[r_idx:r_idx+block_size_val, c_idx:c_idx+
    block_size_val] = average

    return output_image.astype(np.uint8)

if __name__ == "__main__":
    print("--- Task 4: Block Averaging (Spatial Resolution Reduction) ---")
    display_image(img_gray_original, "Original Grayscale Image")

    block_sizes_list = [3, 5, 7]
    for b_size in block_sizes_list:
        res_reduced_img = block_average_resolution(img_gray_original.copy(),
    b_size)
        display_image(res_reduced_img, f"Grayscale Image with {b_size}x{b_size}
    Block Averaging")

    print("\nApplying to Color for demonstration:")
    display_image(img_color_original, "Original Color Image")
    for b_size in block_sizes_list:
        res_reduced_color_img = block_average_resolution(img_color_original.copy
    (), b_size)
        display_image(res_reduced_color_img, f"Color Image with {b_size}x{b_size}
     Block Averaging")

    print("Task 4 Completed.")
```

Listing 4: Block Averaging Function

**Results:**

**Grayscale Image:**

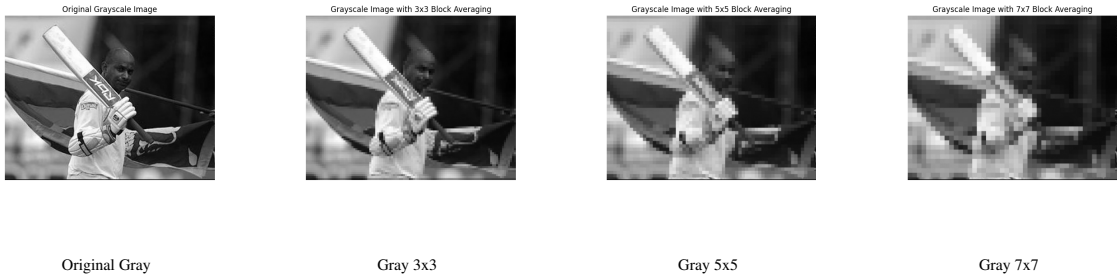


Figure 4: Task 4 - Block Averaging Results (Grayscale).

**Color Image:**


Original Color | Color 3x3 | Color 5x5 | Color 7x7

Figure 5: Task 4 - Block Averaging Results (Color).