

EC7212 Computer Vision and Image Processing

TAKE HOME ASSIGNMENT 1

Index No : EG/2020/4307

Date : 06. 21. 2025

CONTENTS

1. Introduction	4
2 Software and Tools Used	4
3 Tasks and Results.....	5
3.1 Task 1 – Intensity Level Reduction	5
3.2 Task 2 – Average Filtering.....	10
3.3 Task 3 – Image Rotation	14
3.4 Task 4 – Block Averaging	17
4. GitHub Link.....	21

List of Figures

Figure 1: Screenshot of the Python program code for intensity level reduction.	5
Figure 2 : Input image used for intensity level reduction.	6
Figure 3: Output image after reducing intensity levels to 2 levels.	7
Figure 4: Output image after reducing intensity levels to 4 levels.	8
Figure 5: Output image after reducing intensity levels to 8 levels.	9
Figure 6: Screenshot of the Python program code for average filtering.	10
Figure 7: Output image after applying a 3×3 average filter.	11
Figure 8: Output image after applying a 10×10 average filter.	12
Figure 9: Output image after applying a 20×20 average filter.	13
Figure 10: Screenshot of the Python program code for image rotation.....	14
Figure 11: Output image rotated by 45 degrees.....	15
Figure 12: Output image rotated by 90 degrees.....	16
Figure 13: Screenshot of the Python program code for block averaging.	17
Figure 14: Output image after 3×3 block averaging.	18
Figure 15: Output image after 5×5 block averaging.	19
Figure 16: Output image after 7×7 block averaging.	20

1. Introduction

This assignment is on the implementation of the basic image processing techniques using python

These primary tasks include

- Reducing the number of intensity levels
- Applying average(mean) filters
- Rotation images
- Block averaging for resolution reduction

These tasks demonstrate foundational concepts in image quantization, smoothing, rotation, and spatial resolution manipulation

2 Software and Tools Used

Item	Version/Details
Python Version	3.13.3
Libraries	numpy, opencv-python (cv2), matplotlib
IDE/Notebook	VS Code
Operation System	Windows 11

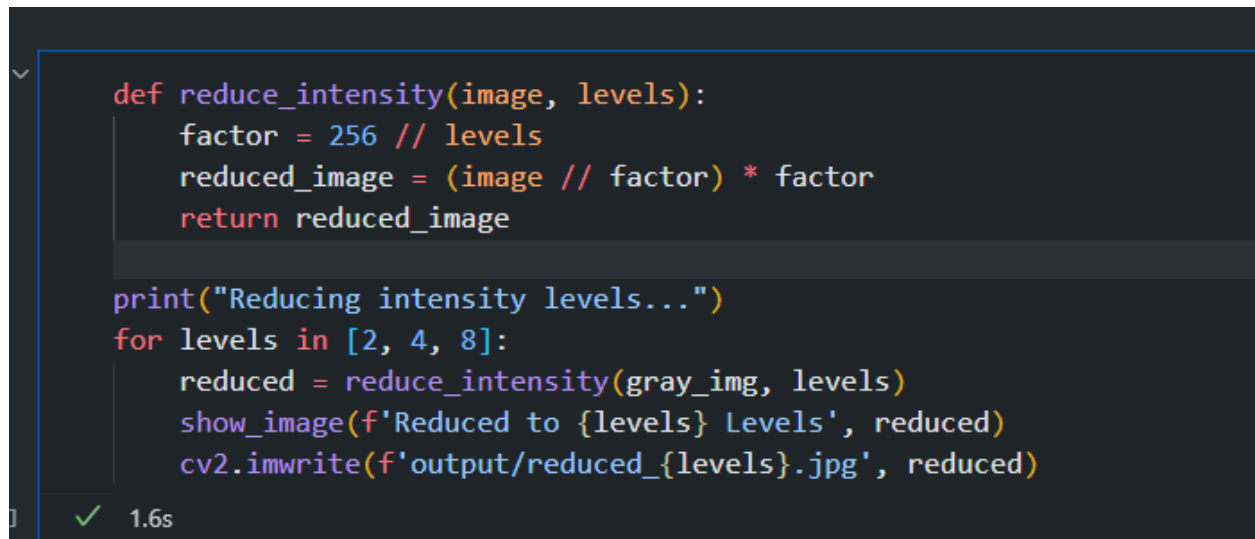
3 Tasks and Results

3.1 Task 1 – Intensity Level Reduction

To reduce the number of intensity levels in an image from 256 to 2, in integer powers of 2. The desired number of intensity levels needs to be a variable input to your program.

Method:

- Load image in grayscale
- Apply integer division and multiplication to quantize pixel values

A screenshot of a Python code editor with a dark background. The code defines a function 'reduce_intensity' that takes an image and a number of levels as input. It calculates a factor by dividing 256 by the number of levels, then uses integer division and multiplication to reduce the image's intensity. The function is called in a loop for 2, 4, and 8 levels. The code is syntax-highlighted with colors: def (blue), reduce_intensity (blue), image (blue), levels (blue), factor (blue), 256 (blue), // (blue), levels (blue), reduced_image (blue), (image // factor) (blue), * (blue), factor (blue), return (red), reduced_image (blue). The code is followed by a print statement and a for loop. The for loop iterates over [2, 4, 8]. Inside the loop, it calls reduce_intensity, shows the image, and writes it to a file. The code is executed successfully, as indicated by a green checkmark and '1.6s' in the bottom left corner.

```
def reduce_intensity(image, levels):  
    factor = 256 // levels  
    reduced_image = (image // factor) * factor  
    return reduced_image  
  
print("Reducing intensity levels...")  
for levels in [2, 4, 8]:  
    reduced = reduce_intensity(gray_img, levels)  
    show_image(f'Reduced to {levels} Levels', reduced)  
    cv2.imwrite(f'output/reduced_{levels}.jpg', reduced)
```

Figure 1: Screenshot of the Python program code for intensity level reduction.

Input Image:



Figure 2 : Input image used for intensity level reduction.

Output Images



Figure 3: Output image after reducing intensity levels to 2 levels.



Figure 4: Output image after reducing intensity levels to 4 levels.



Figure 5: Output image after reducing intensity levels to 8 levels.

Observation:

As intensity levels decrease, the image loses detail and appears more posterized or cartoon-like.

3.2 Task 2 – Average Filtering

Load an image and then perform a simple spatial 3x3 average of image pixels. Repeat the process for a 10x10 neighborhood and again for a 20x20 neighborhood.

Method:

- Use OpenCV's `cv2.blur()` function with varying kernel sizes

```
def average_filter(image, ksize):  
    return cv2.blur(image, (ksize, ksize))  
  
print("Applying average filters...")  
  
for size in [3, 10, 20]:  
    filtered = average_filter(gray_img, size)  
    show_image(f'Average Filter {size}x{size}', filtered)  
    cv2.imwrite(f'output/average_{size}x{size}.jpg', filtered)
```

Figure 6: Screenshot of the Python program code for average filtering.

Output Images:



Figure 7: Output image after applying a 3×3 average filter.



Figure 8: Output image after applying a 10×10 average filter.



Figure 9: Output image after applying a 20×20 average filter.

Observation:

- Small kernel: slight smoothing, edges mostly preserved.
- Large kernel: stronger blur, finer details smoothed out.

3.3 Task 3 – Image Rotation

Rotate an image by 45 and 90 degrees.

Method:

Use OpenCV's `getRotationMatrix2D` and `warpAffine`.

```
def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w//2, h//2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated

print("Rotating image...")
rotated_45 = rotate_image(color_img, 45)
rotated_90 = rotate_image(color_img, 90)

show_image('Rotated 45°', cv2.cvtColor(rotated_45, cv2.COLOR_BGR2RGB), cmap_type=None)
show_image('Rotated 90°', cv2.cvtColor(rotated_90, cv2.COLOR_BGR2RGB), cmap_type=None)

cv2.imwrite('output/rotated_45.jpg', rotated_45)
cv2.imwrite('output/rotated_90.jpg', rotated_90)
```

Figure 10: Screenshot of the Python program code for image rotation.

Output Images:



Figure 11: Output image rotated by 45 degrees.



Figure 12: Output image rotated by 90 degrees.

Observation:

The image is rotated around its center while maintaining its size.

3.4 Task 4 – Block Averaging

For every 3×3 block of the image (without overlapping), replace all the corresponding 9 pixels by their average. This operation simulates reducing the image spatial resolution. Repeat this for 5×5 blocks and 7×7 blocks.

Method:

- Divide the image into blocks (3×3 , 5×5 , 7×7).
- Replace block pixels with the block's mean.

```
def block_average(image, block_size):
    """
    Reduce image resolution by block averaging.
    Works for both grayscale and color images.
    """
    output = image.copy()
    h, w = image.shape[:2]

    for y in range(0, h, block_size):
        for x in range(0, w, block_size):
            y_end = min(y + block_size, h)
            x_end = min(x + block_size, w)
            block = image[y:y_end, x:x_end]

            # For color or grayscale: average over all axes except the first two
            avg_color = block.mean(axis=(0, 1)).astype(np.uint8)

            output[y:y_end, x:x_end] = avg_color

    return output

print("Applying block averaging...")

for size in [3, 5, 7]:
    block_avg = block_average(color_img, size)

    # Display: convert BGR to RGB for Matplotlib
    show_image(f'Block Average {size}x{size}', cv2.cvtColor(block_avg, cv2.COLOR_BGR2RGB), cmap_type=None)

    # Save: keep BGR for saving
    cv2.imwrite(f'output/block_{size}x{size}.jpg', block_avg)

print("✅ All tasks completed! All images are displayed above and saved in 'output/' folder.")
```

✓ 7.6s

Figure 13: Screenshot of the Python program code for block averaging.

Output Images:



Figure 14: Output image after 3×3 block averaging.



Figure 15: Output image after 5×5 block averaging.



Figure 16: Output image after 7×7 block averaging.

Observation:

Larger block sizes result in more noticeable pixelation, resembling low-resolution images.

4. GitHub Link

GitHub repo : <https://github.com/srijithyaparathna/EC-7212-Computer-Vision-and-Image-Processing-TakeHomeAssignment1/tree/main>