EC7212: COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT 1

NAME : SUNDARASEKARA G.O.

REG No. : EG/2020/3943

SEMESTER : 07

DATE : 21/06/2025

Table of Contents

List of Figures

List of Tables

1 Introduction

This assignment involves implementing four image processing operations. They are carried out using Python and OpenCV.

The tasks cover:

- Intensity Level Reduction
- Spatial Averaging (Blurring)
- Image Rotation
- Block Averaging for resolution reduction

The goal is to understand image manipulation techniques and visualize their effects on some sample images.

2 Methodology (Code Overview)

2.1 Task 1: Intensity Level Reduction

The program reduces the number of intensity levels in an image from 256 to 2, in integer powers of 2. The desired number of intensity levels are specified(input) by the user. This is achieved by dividing pixel values into discrete intervals and mapping them to their average level.

Process:

- The program checks if the input value given by the user is one of the allowed powers of 2. If not, an error message is displayed.
- The 256 possible intensity values are grouped into equal intervals depending on the desired number of levels given by the user.
- The quantization factor is calculated using:

$$factor = \frac{256}{desired\_levels}$$

- An integer division is performed by the factor for each pixel, effectively grouping it into a level.
- Then it is multiplied back by the factor to map it to a representative intensity value for that level.
- The image is first converted to grayscale in order to ensure intensity operations are applied on a single channel.
- The output image is saved with a filename that includes the number of levels used.

```python
import cv2
import numpy as np
import os

def reduce_intensity_levels(img, levels):
    if (levels < 2 or levels > 256):
        raise ValueError("Levels must be between 2 and 256 inclusive.")

    desired_levels = [2 ** i for i in range(1, 9)]

    if levels not in desired_levels:
        raise ValueError("Levels must be one of the following: 2, 4, 8, 16, 32, 64, 128, 256")

    factor = 256 // levels
    reduced_img = (img // factor) * factor
    return reduced_img

if __name__ == "__main__":
    img = cv2.imread("../images/task1.jpg", cv2.IMREAD_GRAYSCALE) # loaded as a numpy array

    if img is None:
        print("Error: Image not found.")
        exit(1)

    print("Original Image Shape: ", img.shape)

    levels = int(input("Enter the number of intensity levels (2-256, in integer powers of 2): "))

    try:
        output_img = reduce_intensity_levels(img, levels) # applies the division to all pixels at once
        os.makedirs("../results", exist_ok=True)
        output_filename = f"../results/task1_reduced_intensity_levels_{levels}.jpg"
        cv2.imwrite(output_filename, output_img)
    except ValueError as e:
        print(e)
```

FIGURE 1: TASK 1: CODE FOR INTENSITY LEVEL REDUCTION

Execution:

```
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1/task
s# python3 task_1_intensity_reduction.py
Original Image Shape:  (256, 256)
Enter the number of intensity levels (2-256, in integer powers of 2): 2
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1/task
s# python3 task_1_intensity_reduction.py
Original Image Shape:  (256, 256)
Enter the number of intensity levels (2-256, in integer powers of 2): 8
```

FIGURE 2: PROGRAM EXECUTION

2.2 Task 2: Spatial Averaging (Blurring)

The program loads an image and then performs a simple spatial 3x3 average of image pixels. The process is repeated for a 10x10 neighborhood and again for a 20x20 neighborhood. This is achieved using OpenCV's cv2.blur() function to smooth the image.

Process:

- Load color image (this can be done to a grayscale image too). Spatial averaging is applied to all color channels simultaneously.
- A spatial averaging filter replaces each pixel's value with the average of its surrounding pixels.
- The size of the neighborhood (kernel) determines how many surrounding pixels are considered:
- The averaging is applied using cv2.blur().
- The same function is executed three times with different kernel sizes.
- Each of the three resulting images is saved with a filename indicating the filter size used.

```python
import cv2
import os

os.makedirs("../results", exist_ok=True)
img = cv2.imread('../images/task2.jpg')
if img is None:
    print("Image not found!")
    exit(1)

blur_3x3 = cv2.blur(img, (3, 3))
output_filename = f"../results/task2_blur_3x3.jpg"
cv2.imwrite(output_filename, blur_3x3)

blur_10x10 = cv2.blur(img, (10, 10))
output_filename = f"../results/task2_blur_10x10.jpg"
cv2.imwrite(output_filename, blur_10x10)

blur_20x20 = cv2.blur(img, (20, 20))
output_filename = f"../results/task2_blur_20x20.jpg"
cv2.imwrite(output_filename, blur_20x20)

print("Blurred images saved successfully in the 'results' directory.")
```

FIGURE 3: TASK 2: SPATIAL AVERAGING (BLURRING)

Execution:

```
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1/tasks# python3 task_2_spatial_averaging.py
Blurred images saved successfully in the 'results' directory.
```

FIGURE 4: PROGRAM EXECUTION

2.3 Task 3: Image Rotation

The program rotates an image by 45 and 90 degrees. This is done by using OpenCV's affine transform and rotate functions.

Process:

- Load image in its original color format (this can be done to a grayscale image too).
- Extract height and width of the image.
- Calculate the center of the image to serve as the rotation point.
- Rotation by 45 degrees is not a multiple of 90, so a rotation matrix is explicitly computed using OpenCV's getRotationMatrix2D(). If the angle given as the argument is positive, it will result in an counterclockwise rotation. If the angle given is negative, a clockwise rotation will be resulted. Here, an counterclockwise rotation is performed.
- The matrix is then applied using cv2.warpAffine() to produce the rotated image.
- Since 90 degrees is a multiple of 90, OpenCV provides a built-in function cv2.rotate() which directly rotates the image clockwise. To rotate counterclockwise, give the argument: cv2.ROTATE_90_COUNTERCLOCKWISE.
- Both rotated images are saved in the results directory.

```python
import cv2
import os

os.makedirs("../results", exist_ok=True)
img = cv2.imread("../images/task3.jpg")
if img is None:
    print("Image not found!")
    exit(1)

# Get the image center
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
print("Image Shape: ", img.shape)

# 45 is a custom angle, so we need to use this method
rotation_matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated_img_45 = cv2.warpAffine(img, rotation_matrix, (w, h))

# 90 is an exact multiple of 90, so we can use this method
rotated_img_90 = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)

output_filename = f"../results/task3_rotated_img_45.jpg"
cv2.imwrite(output_filename, rotated_img_45)

output_filename = f"../results/task3_rotated_img_90.jpg"
cv2.imwrite(output_filename, rotated_img_90)

print("Rotated images saved successfully in the 'results' directory.")
```

FIGURE 5: TASK 3: IMAGE ROTATION

Execution:

```
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Imag
e-Processing-Take-Home-Assignme
nt-1/tasks# python3 task_3_image_rotation.py
Image Shape:  (640, 640, 3)
Rotated images saved successfully in the 'results' directory.
```

FIGURE 6: PROGRAM EXECUTION

2.4 Task 4: Block Averaging

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

Process:

- Load the grayscale or color image.
- Iterate over the image in steps of the block size (3, 5, and 7).
- For each block, average the pixel values.
- Replace all pixels within that block with the computed average.
- Edge blocks that are smaller than the block size (due to image boundaries) are also included and averaged using adjusted dimensions.
- The implementation supports both grayscale and color images by checking the number of channels. Comment or uncomment the code line relevant to loading a grayscale or color image.

```python
import cv2
import numpy as np
import os

def get_block_average(img, block_size):
    h, w = img.shape[:2]
    output = np.zeros_like(img) # Create an empty output image of the same shape

    if len(img.shape) == 2:  # Grayscale image
        print("Grayscale image detected.")
        for y in range(0, h, block_size):
            for x in range(0, w, block_size):
                y_end = min(y + block_size, h) # Define the block boundaries (ensure no overflow at edges)
                x_end = min(x + block_size, w)

                block = img[y:y_end, x:x_end] # Extract the block
                avg = np.uint8(np.mean(block))

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

    elif len(img.shape) == 3:  # Color image
        print("Color image detected.")
        channels = img.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)
                for c in range(channels): # Process each channel separately
                    block = img[y:y_end, x:x_end, c]
                    avg = np.uint8(np.mean(block))

                    output[y:y_end, x:x_end, c] = avg

    else:
        raise ValueError("Unsupported image format.")

    return output
```

FIGURE 7: TASK 4: BLOCK AVERAGING – GET_BLOCK_AVERAGE FUNCTION

```python
def main():
    os.makedirs("../results", exist_ok=True)
    img = cv2.imread("../images/task4.jpg") # Uncomment to load as color image
    # img = cv2.imread("../images/task4.jpg", cv2.IMREAD_GRAYSCALE) # Uncomment to load as grayscale image

    if img is None:
        print("Image not found!")
        exit(1)

    for block_size in [3, 5, 7]:
        # result = get_block_average(img, block_size)
        result = get_block_average(img, block_size)
        if len(img.shape) == 2:
            filename = f"../results/task4_block_average_for_grayscale_{block_size}.jpg"
        elif len(img.shape) == 3:
            filename = f"../results/task4_block_average_for_color_{block_size}.jpg"
        else:
            raise ValueError("Unsupported image format.")
        cv2.imwrite(filename, result)
    print("Block averaged images saved successfully in the 'results' directory.")


if __name__ == "__main__":
    main()
```

FIGURE 8: TASK 4: BLOCK AVERAGING – MAIN FUNCTION

Execution:

```
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1/t
asks# python3 task_4_block_averaging.py
Grayscale image detected.
Grayscale image detected.
Grayscale image detected.
Block averaged images saved successfully in the 'results' directory.
root@LAPTOP-MOFFKGDV:~/computer_vision_and_image_processing/take_home_assignment_1/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1/t
asks# python3 task_4_block_averaging.py
Color image detected.
Color image detected.
Color image detected.
Block averaged images saved successfully in the 'results' directory.
```

FIGURE 9: PROGRAM EXECUTION

3 Results and Observations

3.1 Task 1: Intensity Level Reduction

Results:

TABLE 3.1: RESULTS FROM TASK1 PROGRAM EXECUTION

Input Image	Reduced Levels = 2	Reduced Levels = 8
		

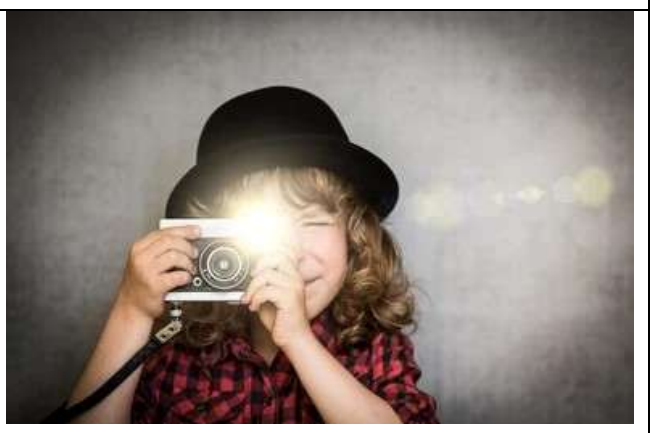
Observations:

With 2 intensity levels, the image appears posterized with visible banding and loss of detail. As the number of levels increases, with 8 levels, the image retains more grayscale detail and looks closer to the original.

3.2 Task 2: Spatial Averaging (Blurring)

Results:

TABLE 3.2: RESULTS FROM TASK2 PROGRAM EXECUTION

Neighbourhood Size	Result
Input Image	
3x3	
10x10	

20x20	

Observations:

The 3×3 blur slightly smooths the image, which may help reduce minor noise. The 20×20 blur strongly blurs the image.

3.3 Task 3: Image Rotation

Results:

TABLE 3.3: RESULTS FROM TASK3 PROGRAM EXECUTION

Angle	Result
Input Image	
45^0 Counterclockwise	

90^0 Clockwise	

Observations:

45° rotation:

The image is rotated diagonally.

Rotated image is not fully fit in the original image dimensions. So some parts appear cropped. The black borders around the image are visible due to transformation on a fixed-size canvas.

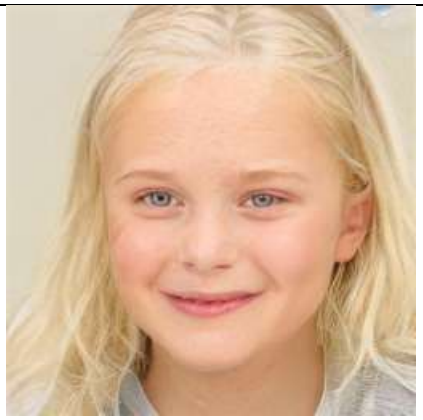
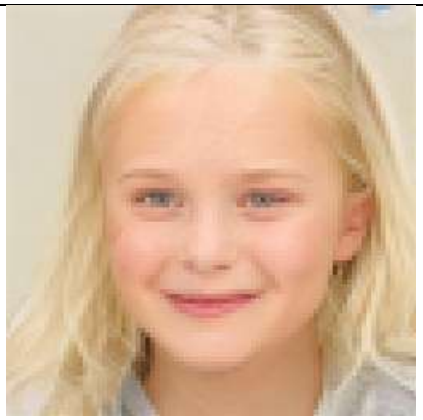
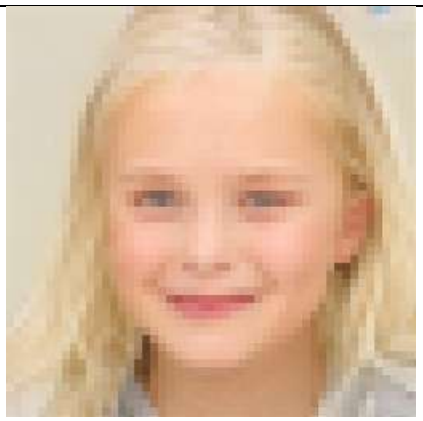
90° rotation:

The image is cleanly rotated clockwise. No black borders or cropping, as OpenCV handles the reorientation perfectly for multiples of 90 degrees.

3.4 Task 4: Block Averaging

Results:

TABLE 3.4: RESULTS FROM TASK4 PROGRAM EXECUTION FOR COLOR IMAGE

Block size	Result
Input Image	
3x3	
5x5	

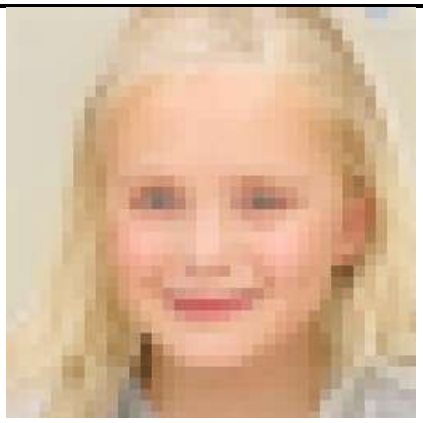
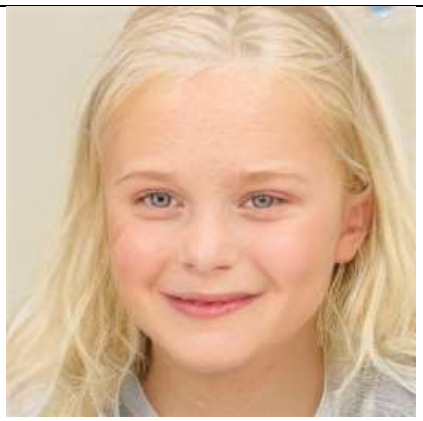
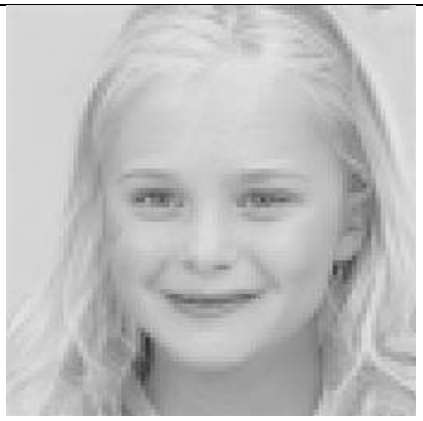
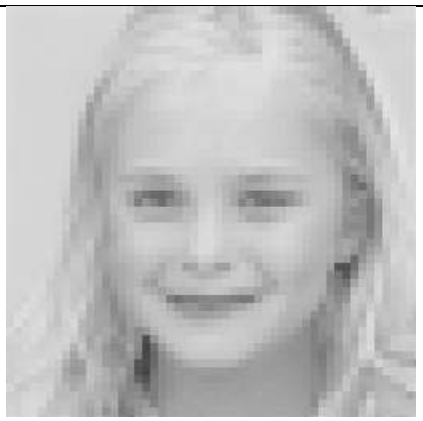
7x7	

TABLE 3.5 RESULTS FROM TASK4 PROGRAM EXECUTION FOR GRAYSCALE IMAGE

Block size	**Result**
Input Image	
3x3	
5x5	

7x7	

Observations:

As the block size increases, the block-averaged image appears increasingly blurred. Visual details from 7×7 blocks are lesser than 3×3. Edge blocks are handled smoothly without cropping or leaving unprocessed borders.

4 GitHub link

The complete code and results are available at:

https://github.com/Githmi123/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-1.git

5 Conclusion

This assignment helped learn fundamental image processing techniques including pixel intensity manipulation, spatial filtering, geometric transformations, and resolution reduction using block averaging.