EC7212 - Computer Vision and Image Processing

Reg. No: EG/2020/3829

Name: ARIYARATHNA P.G.R.V.D

1 Introduction

This report presents the solutions for the Take Home Assignment 1 of the EC7212 Computer Vision and Image Processing course. The assignment involves implementing four image processing tasks using Python with OpenCV and NumPy libraries. The tasks include reducing intensity levels, performing spatial averaging, rotating images, and reducing spatial resolution.

The code is available in a GitHub repository: https://github.com/pgvda/computer-vision-assignment-1.

The following sections describe each task, the implementation approach, and the results.

2 Task 1: Reducing Intensity Levels

This task involves reducing the number of intensity levels in a grayscale image from 256 to a specified number of levels, which must be a power of 2. The image is first converted to grayscale and then quantized using the formula: (image // factor) * factor, where factor = 256 // levels.

3 Task 2: Spatial Averaging

This task applies spatial averaging to an image using 3x3, 10x10, and 20x20 kernels. The filter replaces each pixel with the average of its neighborhood to smooth the image.

4 Task 3: Image Rotation

This task rotates an image by 45 and 90 degrees. It uses a rotation matrix and resizes the output image to avoid cropping.

5 Task 4: Reducing Spatial Resolution

This task simulates reducing the spatial resolution by replacing each non-overlapping block (3x3, 5x5, 7x7) with its average pixel value.

6 Source Code And Output

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
import io
from PIL import Image

uploaded = files.upload()
```

```
import cv2
import matplotlib.pyplot as plt


image_path = 'input.jpg'


image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(image)
plt.axis('off')
plt.show()
```

```
def convert_img_to_grayscale(image):
    if len(image.shape) == 3:
        img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        return img
    else:
        return image
```

```
def reduce_spatial_resolution(image, block_size):

    if block_size < 1:
        raise ValueError("Block size must be a positive.")

    image =convert_img_to_grayscale(image)
```

```python
    # Get image dimensions
    h, w = image.shape
    # Initialize output image
    output = np.zeros((h, w), dtype=np.uint8)

    # Process each block
    for i in range(0, h - block_size + 1, block_size):
        for j in range(0, w - block_size + 1, block_size):
            # Extract the block
            block = image[i:i+block_size, j:j+block_size]
            # Compute the average
            block_mean = np.mean(block).astype(np.uint8)
            # Assign the average to all pixels in the block
            output[i:i+block_size, j:j+block_size] = block_mean

    return output
```

```python
def reduce_intensity_level(image, levels):
    if levels < 2 or levels > 256 or (levels & (levels - 1)) != 0:
        raise ValueError("Number of levels must be a power of 2
between 2 and 256.")

    image = convert_img_to_grayscale(image)
    factor = 256 // levels
    reduced_img = (image // factor) * factor
    return reduced_img
```

```python
def simple_spatial_average(image, kernal_size):
  image = convert_img_to_grayscale(image)

  averaged = cv2.blur(image, (kernal_size, kernal_size))

  return averaged.astype(np.uint8)
```

```python
def rotate_img(image, angle):
  (h, w) = image.shape[:2]
  center = (w // 2, h // 2)

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

  cos = np.abs(M[0, 0])
```

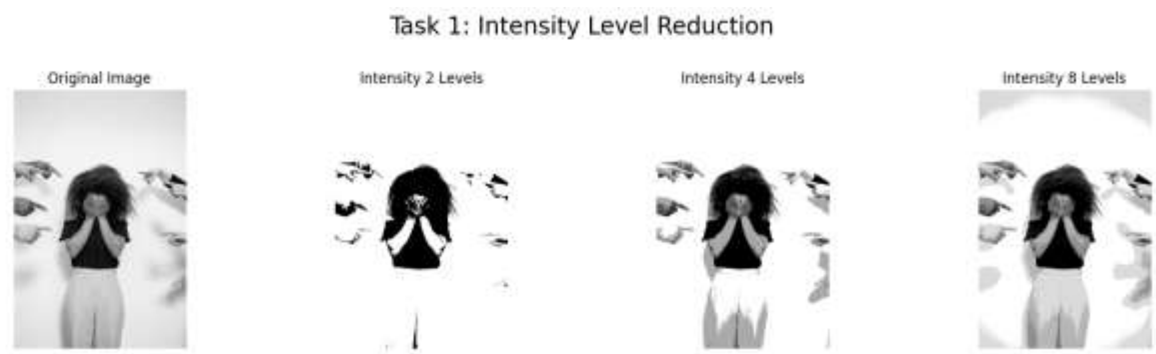
```
  sin = np.abs(M[0, 1])
  new_w = int((h * sin) + (w * cos))
  new_h = int((h * cos) + (w * sin))

  M[0, 2] += (new_w / 2) - center[0]
  M[1, 2] += (new_h / 2) - center[1]

  rotated_image = cv2.warpAffine(image, M, (new_w, new_h))
  return rotated_image
```

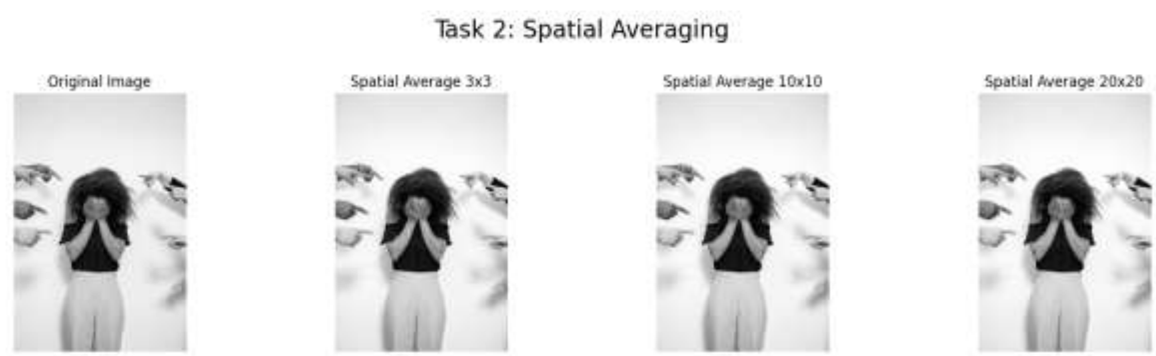
```
def show_result(original_img, results_dict, title = "Output
Images"):
    n_results = len(results_dict) + 1  # +1 for original
    cols = 4
    rows = (n_results + cols - 1) // cols

    plt.figure(figsize=(16, 4 * rows))

    # Display original image
    plt.subplot(rows, cols, 1)
    plt.imshow(original_img, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')

    # Display processed results
    for idx, (name, image) in enumerate(results_dict.items(), 2):
        plt.subplot(rows, cols, idx)
        plt.imshow(image, cmap='gray')
        plt.title(name)
        plt.axis('off')

    plt.suptitle(title, fontsize=20, y=1.02)
    plt.tight_layout()
    plt.show()
```

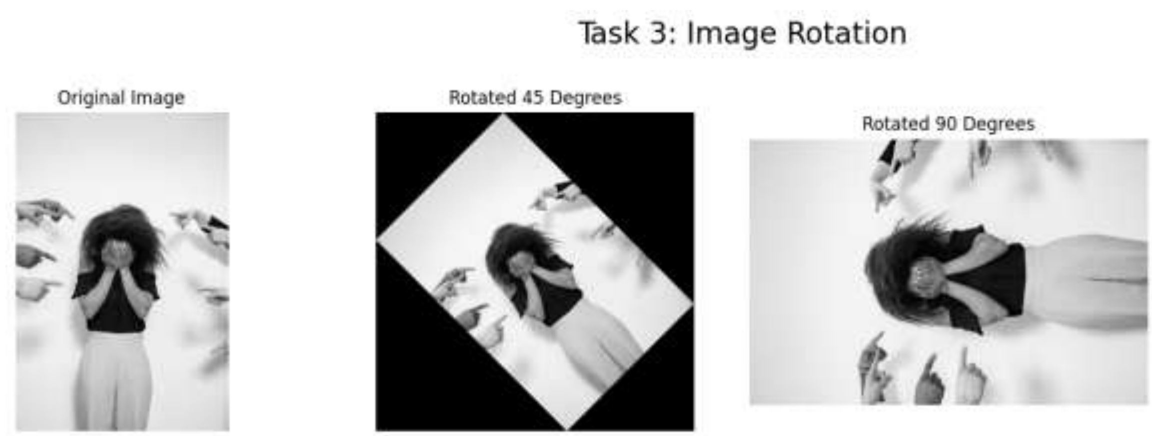
```
def main():
  input_img = cv2.imread('input.jpg')

  if input_img is None:
    raise FileNotFoundError("Input image not found. Please run
relavant code section")

  original_gray = convert_img_to_grayscale(input_img)
```

```
#task : 1
task_1 = {}
intensity_levels = [2, 4, 8]

for levels in intensity_levels:
  reduced = reduce_intensity_level(input_img, levels)
  task_1[f'Intensity {levels} Levels'] = reduced
  cv2.imwrite(f'reduced_intensity_{levels}_levels.jpg', reduced)
show_result(original_gray, task_1, title="Task 1: Intensity Level
Reduction")

#task : 2
task_2 = {}
kernel_sizes = [3, 10, 20]
for k in kernel_sizes:
    averaged = simple_spatial_average(input_img, k)
    task_2[f'Spatial Average {k}x{k}'] = averaged
    cv2.imwrite(f'spatial_average_{k}x{k}.jpg', averaged)
show_result(original_gray, task_2, title="Task 2: Spatial
Averaging")

# task : 3
task_3 = {}
angles = [45, 90]
for angle in angles:
    rotated = rotate_img(input_img, angle)
    # Convert to grayscale for display consistency
    rotated_gray = cv2.cvtColor(rotated, cv2.COLOR_BGR2GRAY) if
len(rotated.shape) == 3 else rotated
    task_3[f'Rotated {angle} Degrees'] = rotated_gray
    cv2.imwrite(f'rotated_{angle}_degrees.jpg', rotated)
show_result(original_gray, task_3, title="Task 3: Image Rotation")

# task : 4
task_4 = {}
block_sizes = [3, 5, 7]
for b in block_sizes:
    reduced_res = reduce_spatial_resolution(input_img, b)
    task_4[f'Reduced Resolution {b}x{b}'] = reduced_res
    cv2.imwrite(f'reduced_resolution_{b}x{b}.jpg', reduced_res)
show_result(original_gray, task_4, title="Task 4: Spatial
Resolution Reduction")
```

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







7 Conclusion

The assignment successfully implemented four key image processing tasks using Python and OpenCV. The functions are modular, handle edge cases, and produce the desired visual outcomes. All code and output results are available in the following GitHub repository:

https://github.com/pgvda/computer-vision-assignment-1