EE7204: COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT - 01

NAME	:	LAKSHAN D.P.W.K.
REG. NO	:	EG/2019/3649
SEMESTER	:	07
DATE	:	15/03/2024

Contents

List of Figures

1 GitHub Link

https://github.com/KalinduLakshan/Assignment_01.git

2 Coding Answers

2.1 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.

```
import cv2
import numpy as np


# Define the reduce_intensity_levels function
def reduce_intensity_levels(image, q_levels):
    # Calculate the scaling factor
    scale = 255 / (q_levels - 1)

    # Quantize pixel values
    quantized_image = np.round(image / scale) * scale

    # Convert to uint8
    quantized_image = quantized_image.astype(np.uint8)

    return quantized_image


# Loading the image file, assigned the image file name to a variable for a
gray scale or color image input
# identification
image_path = 'sample.jpg'

isGray = int(input("Do you need to convert the original image into a gray
scale image (Put 1='yes', 0='no'): "))

if isGray:
    input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check whether the image can be read or not
    if input_image is None:
        print("Sorry: Your given image could not read.")
    else:
        # Enter the number of intensity levels as a power of 2
        Q_levels = int(input("Enter the number of intensity levels (power of
2): "))

        # Check whether the number of intensity levels is a power of 2 or not
        if Q_levels & (Q_levels - 1) != 0:
            print("Error: Number of intensity levels must be a power of 2.")
        else:
            # Reduce the intensity levels by changing the quantization levels
            output_image = reduce_intensity_levels(input_image, Q_levels)

            # Display the original and quantized images
            cv2.imshow('Original Image', input_image)
            cv2.imshow(f'{Q_levels} Levels Reduced Intensity Image',
output_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

else:
    input_image = cv2.imread(image_path)

    # Check whether the input image can be read or not
    if input_image is None:
        print("Sorry: Your given image could not read.")
    else:

        # Enter the number of intensity levels as a power of 2
        Q_levels = int(input("Enter the number of intensity levels (power of
2): "))

        # Check whether the number of intensity levels is a power of 2 or not
        if Q_levels & (Q_levels - 1) != 0:
            print("Error: Number of intensity levels must be a power of 2.")
        else:
            # Reduce the intensity levels by changing the quantization levels
            output_image = reduce_intensity_levels(input_image, Q_levels)

            # Display the original and quantized images
            cv2.imshow('Original Image', input_image)
            cv2.imshow(f'{Q_levels} Levels Reduced Intensity Image',
output_image)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2.2 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.

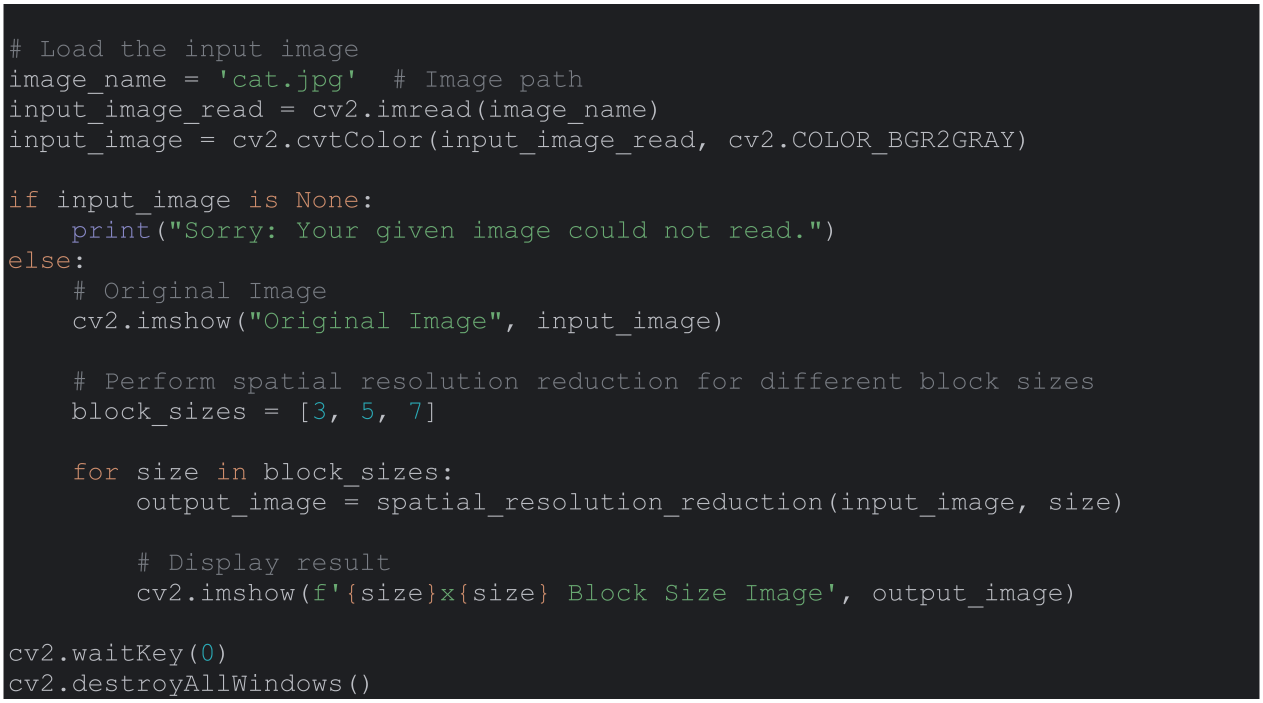
```
import cv2
import numpy as np


# Define the simple_spatial_average function
def simple_spatial_average(image, neighborhood_size):
    # Pad the image to handle the borders of the image
    padded_image = cv2.copyMakeBorder(image, neighborhood_size//2,
neighborhood_size//2, neighborhood_size//2, neighborhood_size//2,
cv2.BORDER_CONSTANT)

    # Initialize the output image
    output_image = np.zeros_like(image)

    # Apply the spatial averaging for pixels
    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            # Extract the current neighborhood
            neighborhood = padded_image[y:y+neighborhood_size,
x:x+neighborhood_size]

            # Calculate average and assign it to the output pixel
            output_image[y, x] = np.mean(neighborhood)

    return output_image


# Load the input image
image_path = 'sample.jpg'
input_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if input_image is None:
    print("Sorry: Your given image could not read.")
else:
    cv2.imshow('Original Image', input_image)
    # Perform spatial averaging for 3, 10, 20 neighborhood sizes
    neighborhood_sizes = [3, 10, 20]

    # Loop through different neighborhood sizes
    for size in neighborhood_sizes:
        output_image = simple_spatial_average(input_image, size)

        # Display the result
        cv2.imshow(f'{size}x{size} Neighborhood Image', output_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2.3 Rotate an image by 45 and 90 degrees.

```
import cv2

# Loading the image file
input_file_name = 'cat.jpg'
input_image = cv2.imread(input_file_name)

# Check whether the input image can be read or not
if input_image is None:
    print("Sorry: Your given image could not read.")
else:
    # Original Image
    cv2.imshow("Original Image", input_image)

    # Rotate the image by 90 degrees clockwise
    output_90_img = cv2.rotate(input_image, cv2.ROTATE_90_CLOCKWISE)
    cv2.imshow("Image rotated by 90 clockwise", output_90_img)

    # Rotate the image by 90 degrees counterclockwise or 270 clockwise
    output_counter90_img = cv2.rotate(input_image,
cv2.ROTATE_90_COUNTERCLOCKWISE)
    cv2.imshow("Image rotated by 90 counterclockwise", output_counter90_img)

'''
45 degrees rotation(arbitrary rotation) cannot be performed using
cv2.rotate() method.Here we need to perform the general rotation by giving
the angle and the scale factor.Need to create a rotation matrix using
cv2.getRotationMatrix2D() and apply it to image with cv2.warpAffine()
'''

    # Rotate the image by 45 degrees clockwise
    (height, width, channel) = input_image.shape
    center = (width/2, height/2)

    clockwise_angle = 45
    scale = 1.0

    img_clockwise_matrix = cv2.getRotationMatrix2D(center, clockwise_angle,
scale)
    output_45_img = cv2.warpAffine(input_image, img_clockwise_matrix, (width,
height))
    cv2.imshow("Image rotated by 45 clockwise", output_45_img)

    # Rotate the image by 45 degrees clockwise
    counterclockwise_angle = -45

    img_counterclockwise_matrix = cv2.getRotationMatrix2D(center,
counterclockwise_angle, scale)
    output_counter_45_img = cv2.warpAffine(input_image,
img_counterclockwise_matrix, (width, height))
    cv2.imshow("Image rotated by 45 counterclockwise", output_counter_45_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

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

```
import cv2
import numpy as np


# Define the spatial_resolution_reduction
def spatial_resolution_reduction(image, block_size):
    # Get the dimensions of the input image
    height, width = image.shape[:2]

    # Calculate the number of blocks in both dimensions
    num_blocks_x = width // block_size  # Integer value is taken
    num_blocks_y = height // block_size

    # Initialize the output image
    output_res_reduction_image = np.zeros_like(image)

    # Iterate over each block
    for y in range(num_blocks_y):
        for x in range(num_blocks_x):
            # Define the coordinates of the current block
            block_top = y * block_size
            block_bottom = (y + 1) * block_size
            block_left = x * block_size
            block_right = (x + 1) * block_size

            # Extract the current block from the image
            current_block = image[block_top:block_bottom,
block_left:block_right]

            # Calculate the average pixel value of the block
            average_value = np.mean(current_block)

            # Assign the average value to all pixels in the current block
            output_res_reduction_image[block_top:block_bottom,
block_left:block_right] = average_value

    return output_res_reduction_image


'''
Here I have used grayscale for better understandings because while reading
images using OpenCV, some color values
are manipulated.
Therefore the original color image has converted into a grayscale image for
better understanding
'''
```



```python
# Load the input image
image_name = 'cat.jpg'  # Image path
input_image_read = cv2.imread(image_name)
input_image = cv2.cvtColor(input_image_read, cv2.COLOR_BGR2GRAY)

if input_image is None:
    print("Sorry: Your given image could not read.")
else:
    # Original Image
    cv2.imshow("Original Image", input_image)

    # Perform spatial resolution reduction for different block sizes
    block_sizes = [3, 5, 7]

    for size in block_sizes:
        output_image = spatial_resolution_reduction(input_image, size)

        # Display result
        cv2.imshow(f'{size}x{size} Block Size Image', output_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3 Results (Captured Images)

3.1 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.

3.1.1 Gray-Scale Results

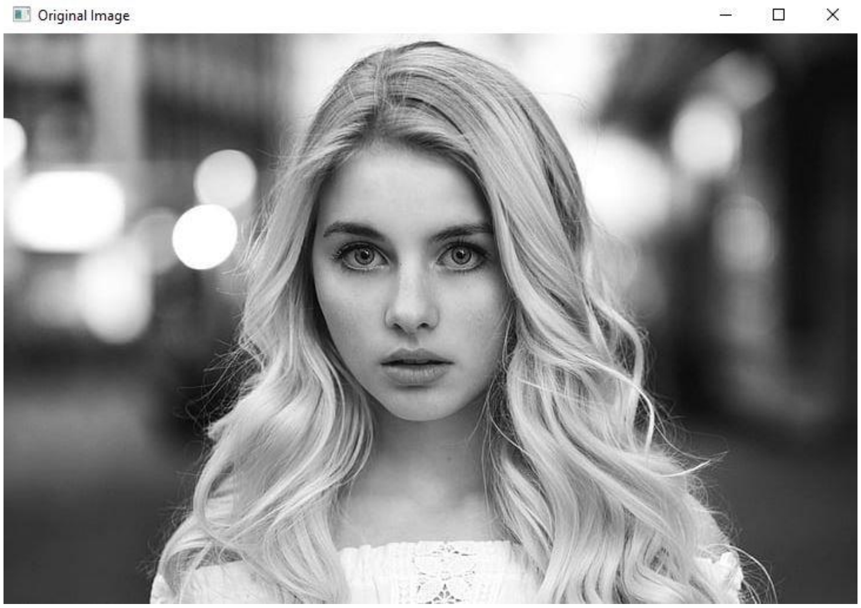


FIGURE 3.1: ORIGINAL GRAYSCALE IMAGE

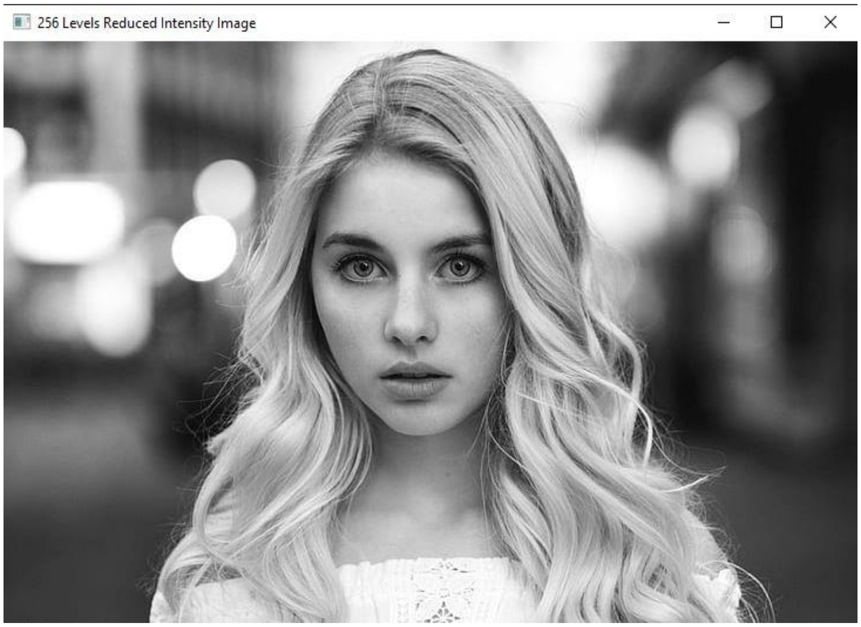


FIGURE 3.2: 256 LEVELS REDUCED INTENSITY IMAGE

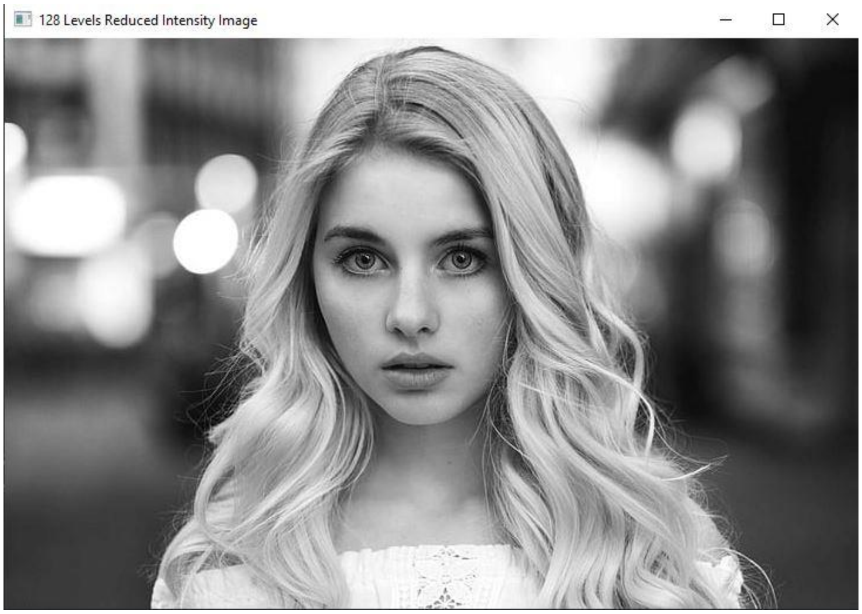


FIGURE 3.3: 128 LEVELS REDUCED INTENSITY IMAGE

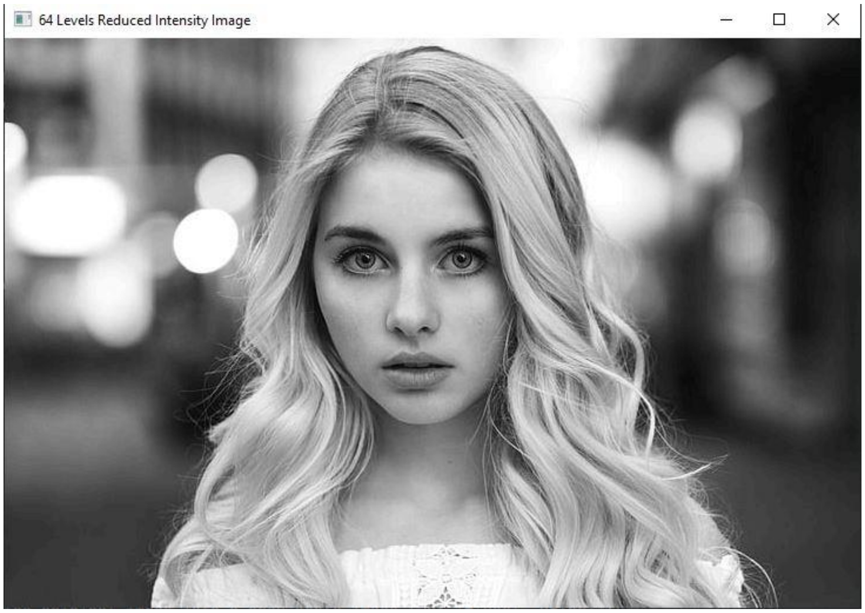


FIGURE 3.4: 64 LEVELS REDUCED INTENSITY IMAGE

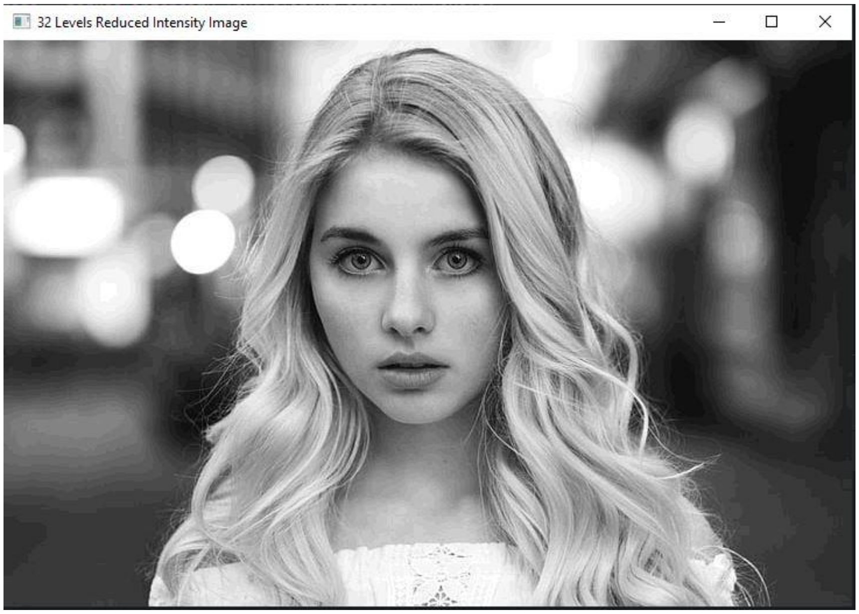


FIGURE 3.5: 32 LEVELS REDUCED INTENSITY IMAGE

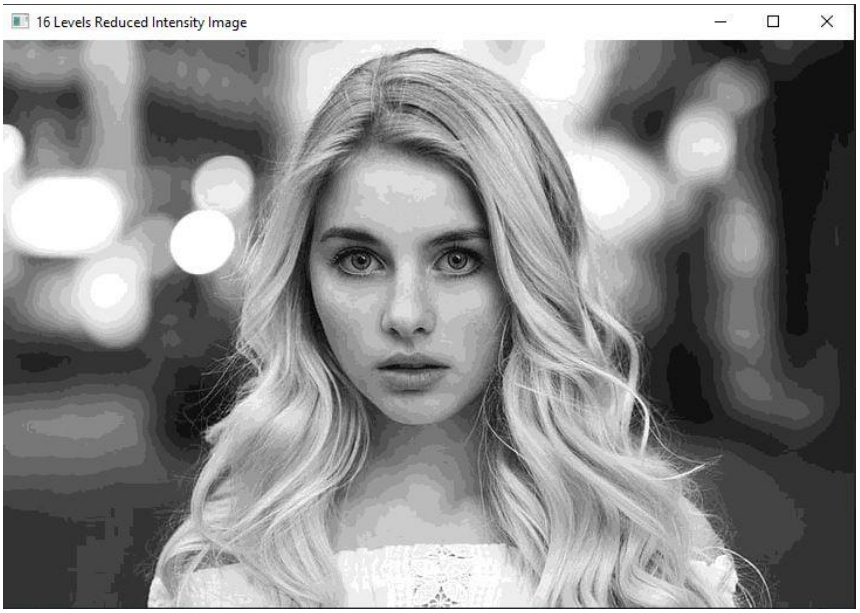


FIGURE 3.6: 16 LEVELS REDUCED INTENSITY IMAGE

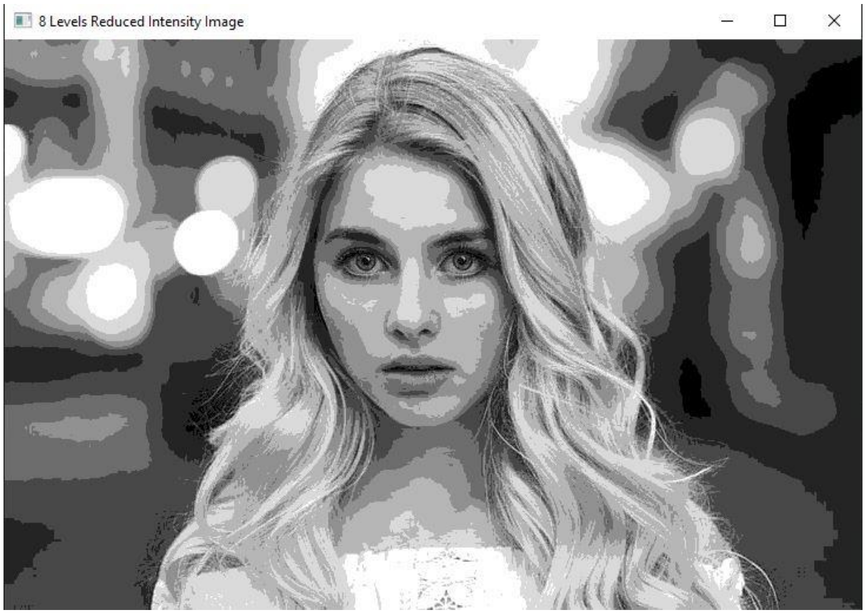


FIGURE 3.7: 8 LEVELS REDUCED INTENSITY IMAGE

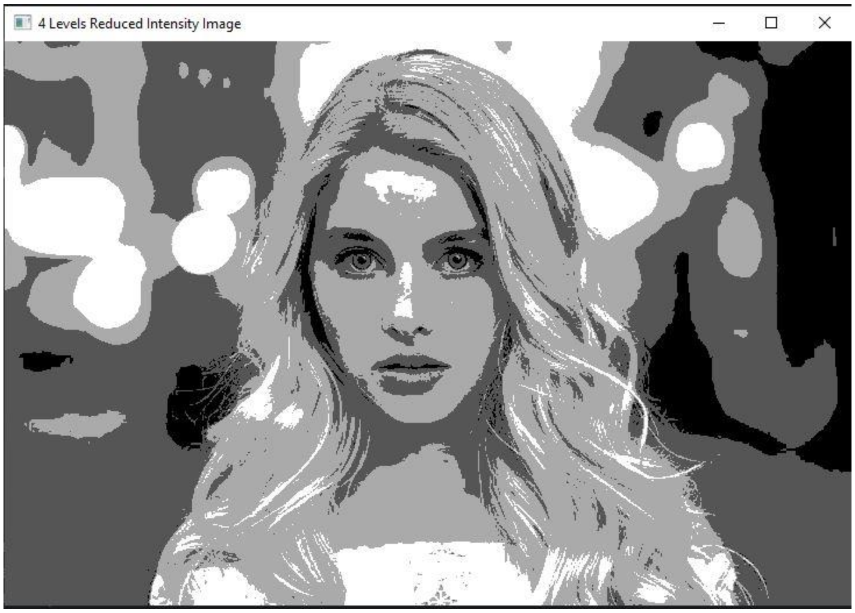


FIGURE 3.8: 4 LEVELS REDUCED INTENSITY IMAGE

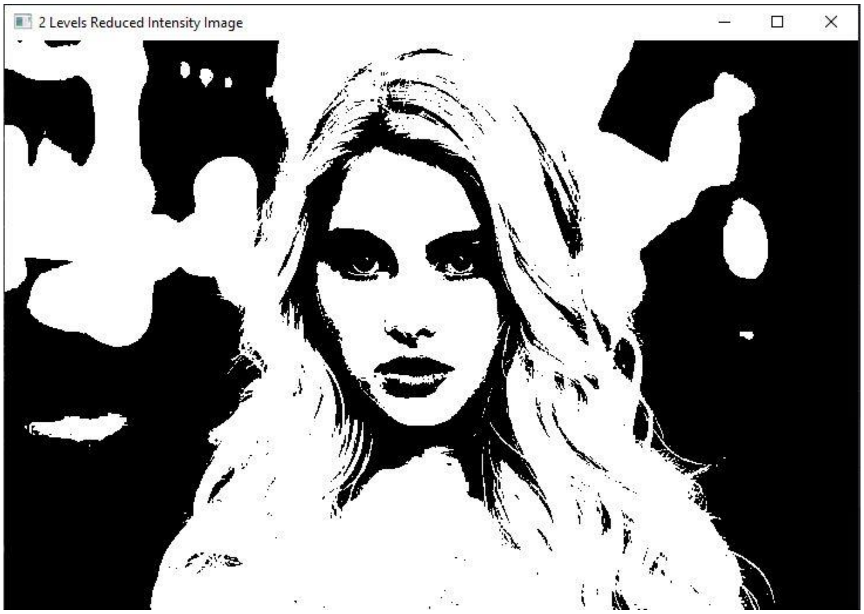


FIGURE 3.9: 2 LEVELS REDUCED INTENSITY IMAGE

3.1.2 Color Results

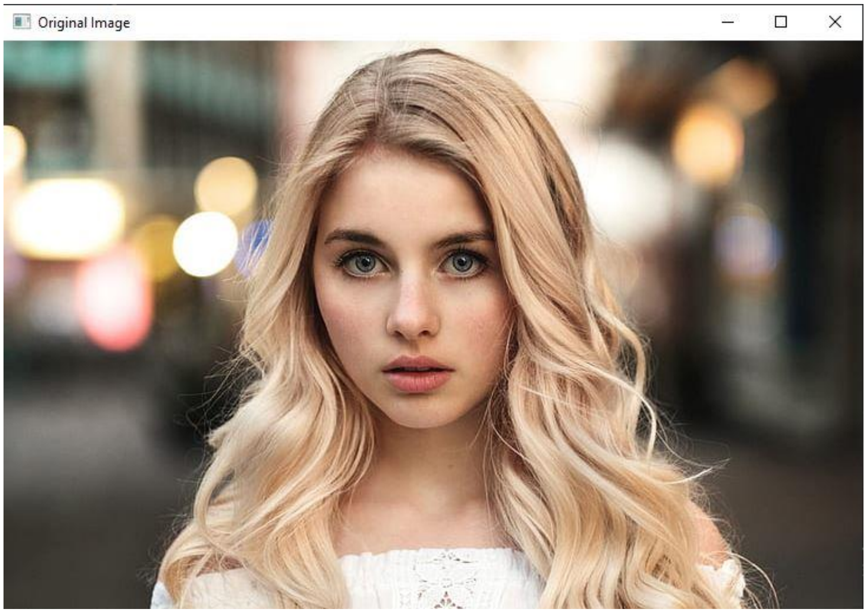


FIGURE 3.10: ORIGINAL IMAGE

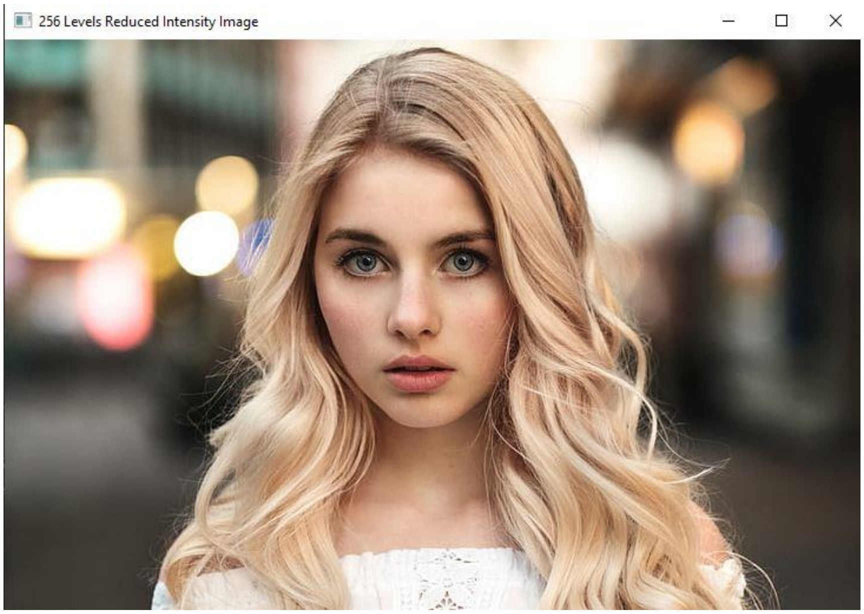


FIGURE 3.11: 256 LEVELS REDUCED INTENSITY IMAGE

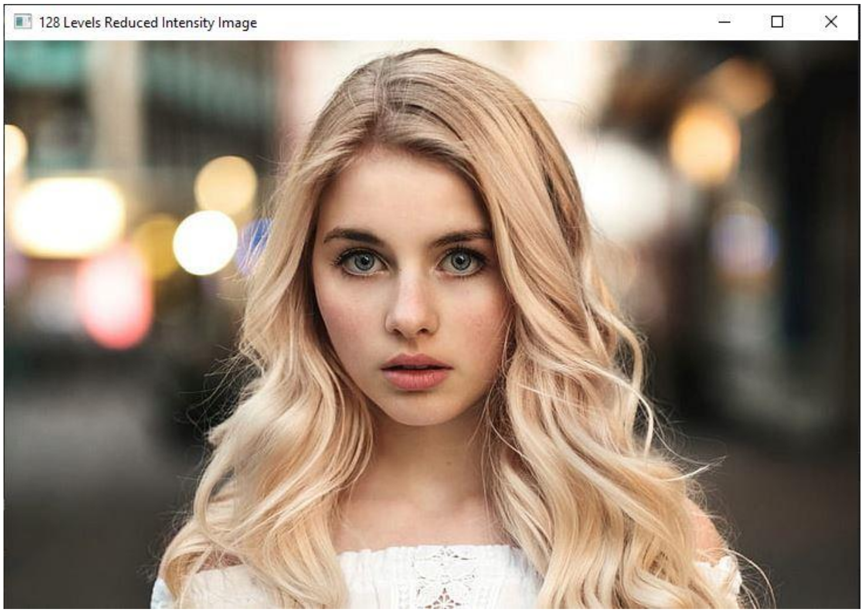


FIGURE 3.12: 128 LEVELS REDUCED INTENSITY IMAGE

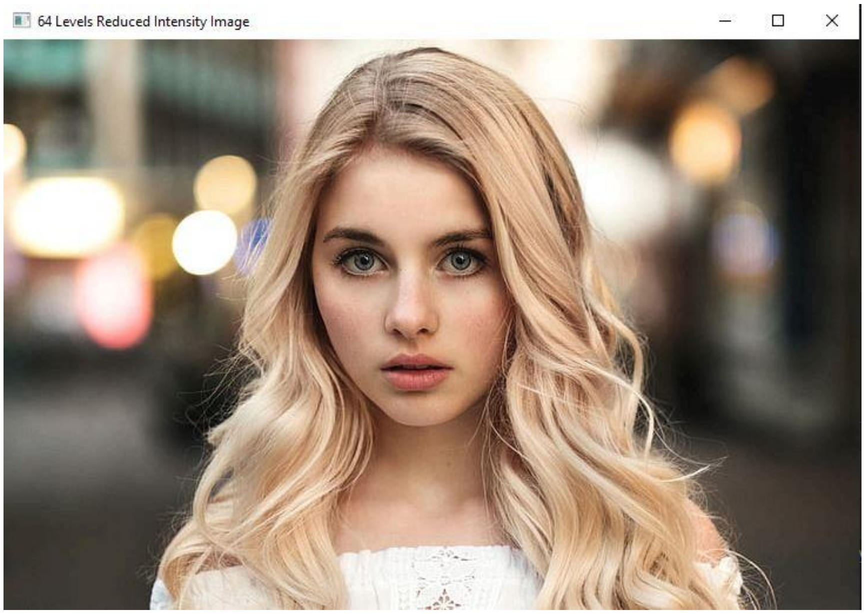


FIGURE 3.13: 64 LEVELS REDUCED INTENSITY IMAGE

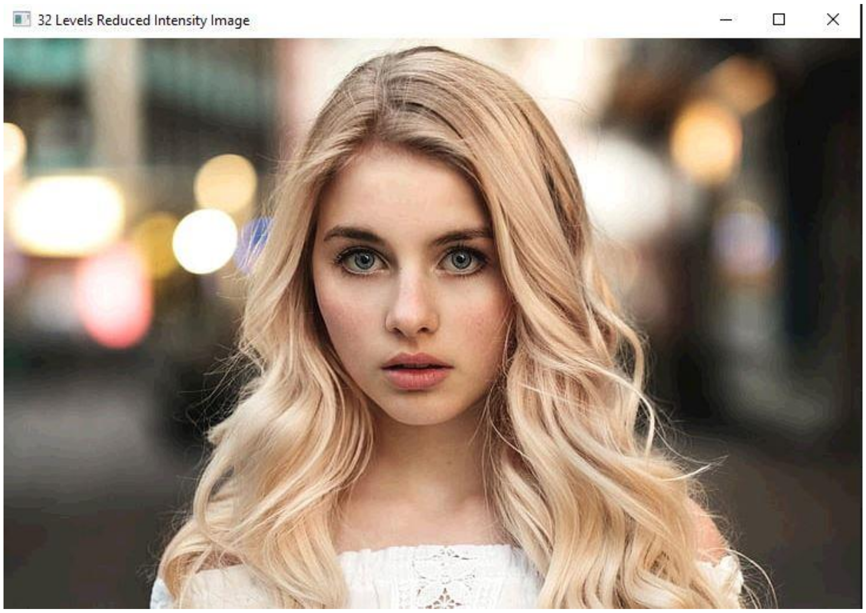


FIGURE 3.14: 32 LEVELS REDUCED INTENSITY IMAGE

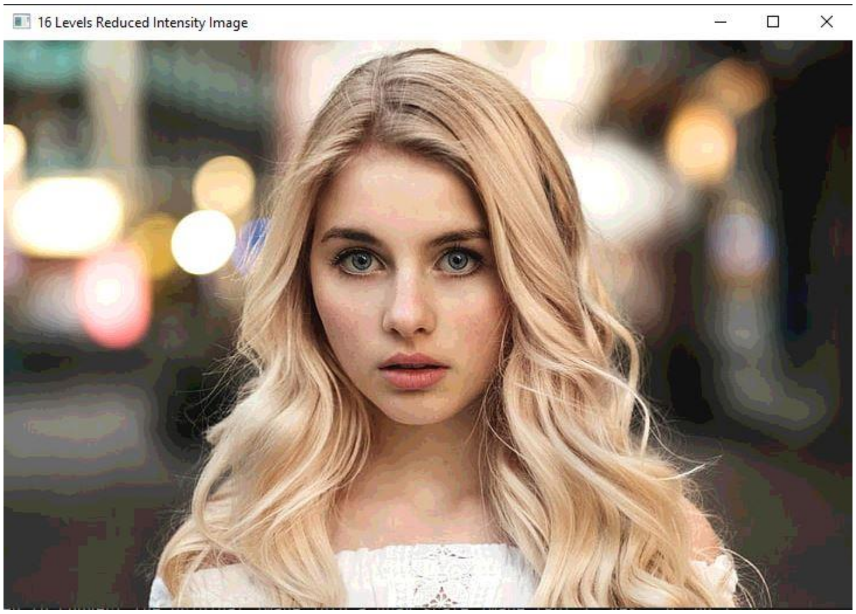


FIGURE 3.15: 16 LEVELS REDUCED INTENSITY IMAGE

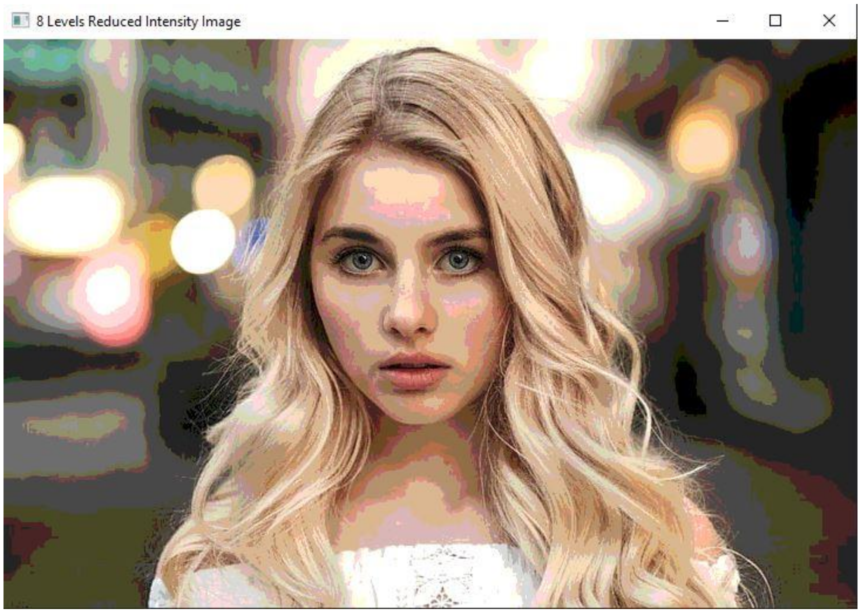


FIGURE 3.16: 8 LEVELS REDUCED INTENSITY IMAGE

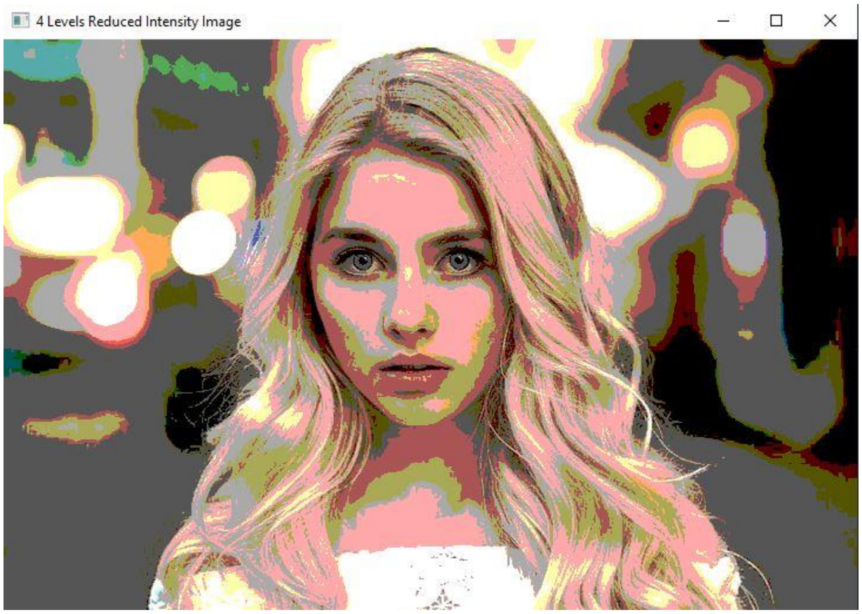


FIGURE 3.17: 4 LEVELS REDUCED INTENSITY IMAGE

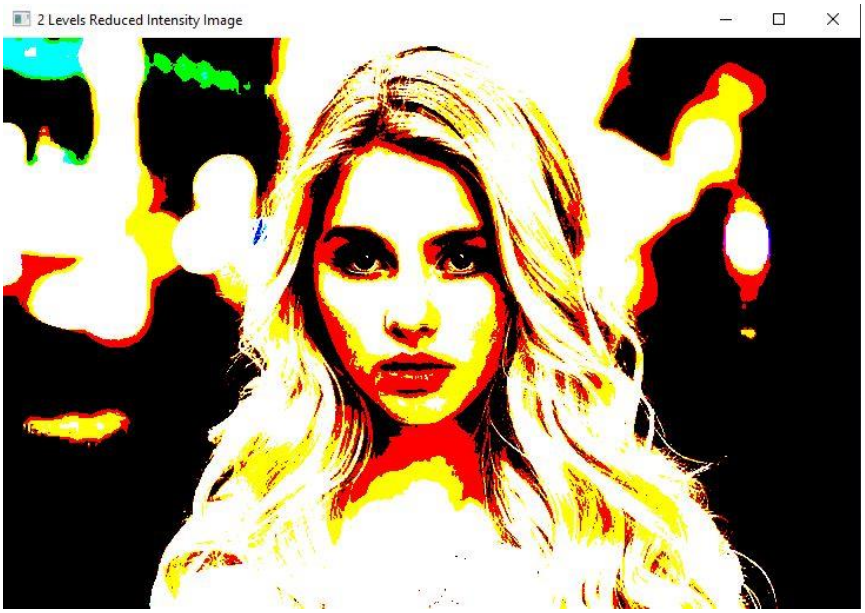


FIGURE 3.18: 2 LEVELS REDUCED INTENSITY IMAGE

3.2 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.

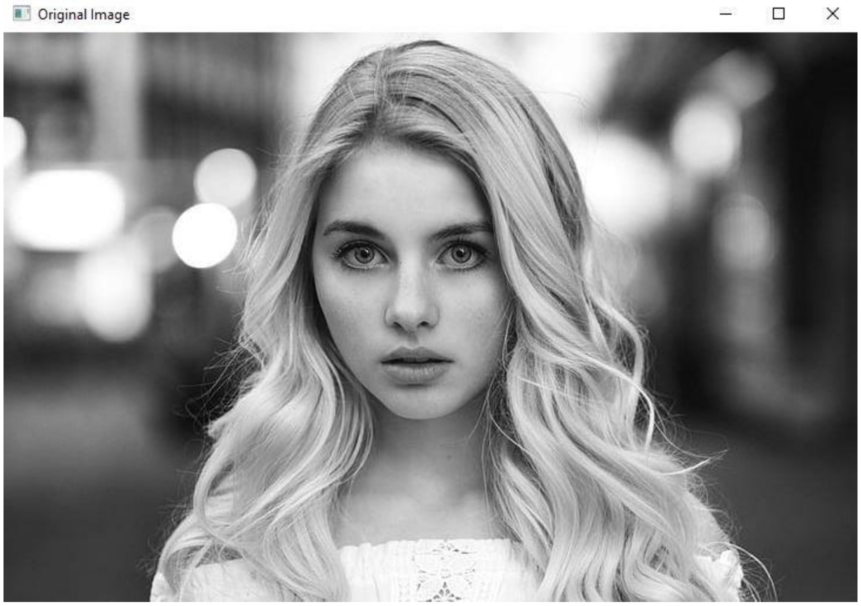


FIGURE 3.19: ORIGINAL IMAGE (IN GRAYSCALE)

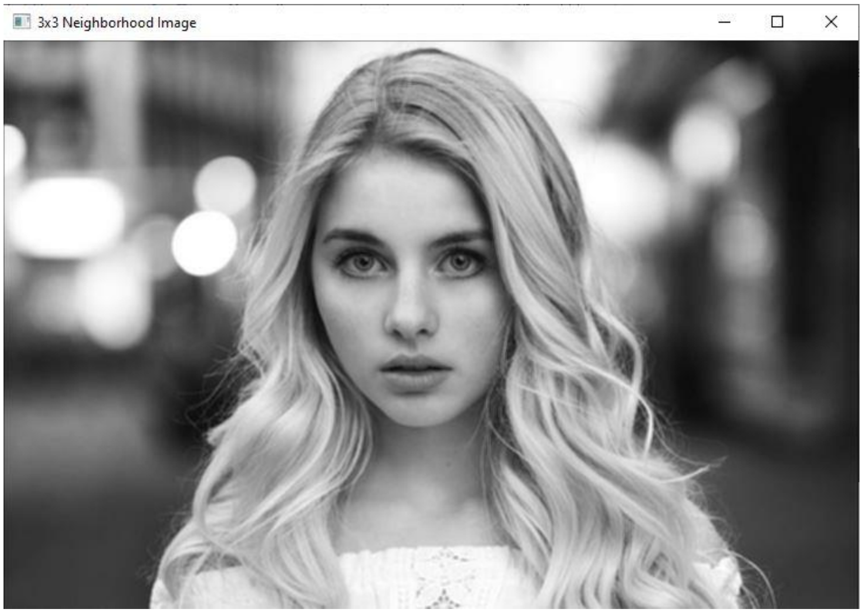


FIGURE 3.20: 3 X 3 SPATIAL AVERAGE IMAGE

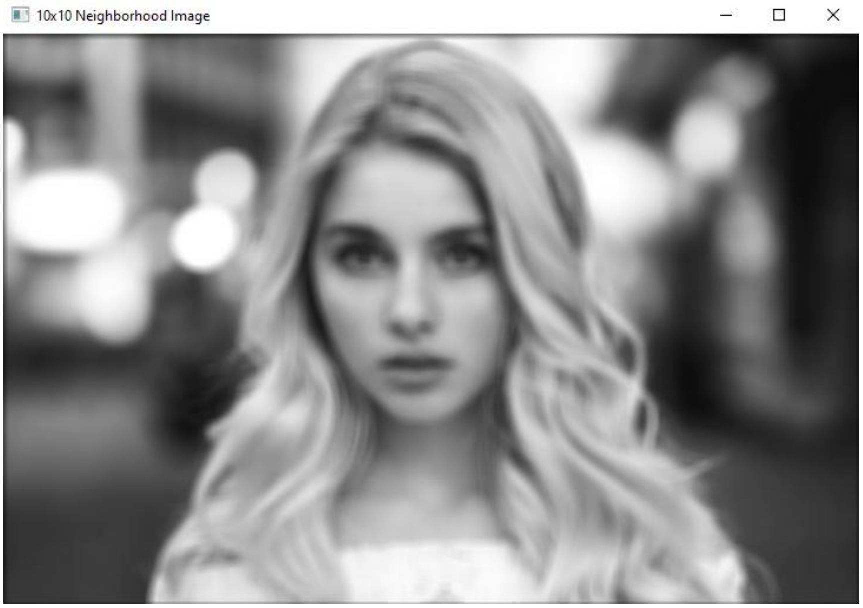


FIGURE 3.21: 10 X 10 SPATIAL AVERAGE IMAGE

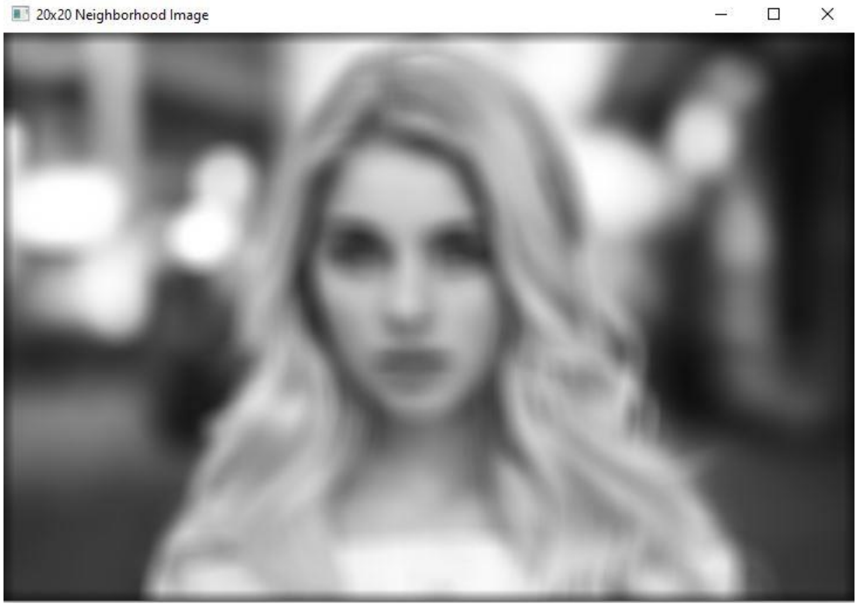


FIGURE 3.22: 20 X 20 SPATIAL AVERAGE IMAGE

3.3 Rotate an image by 45 and 90 degrees.

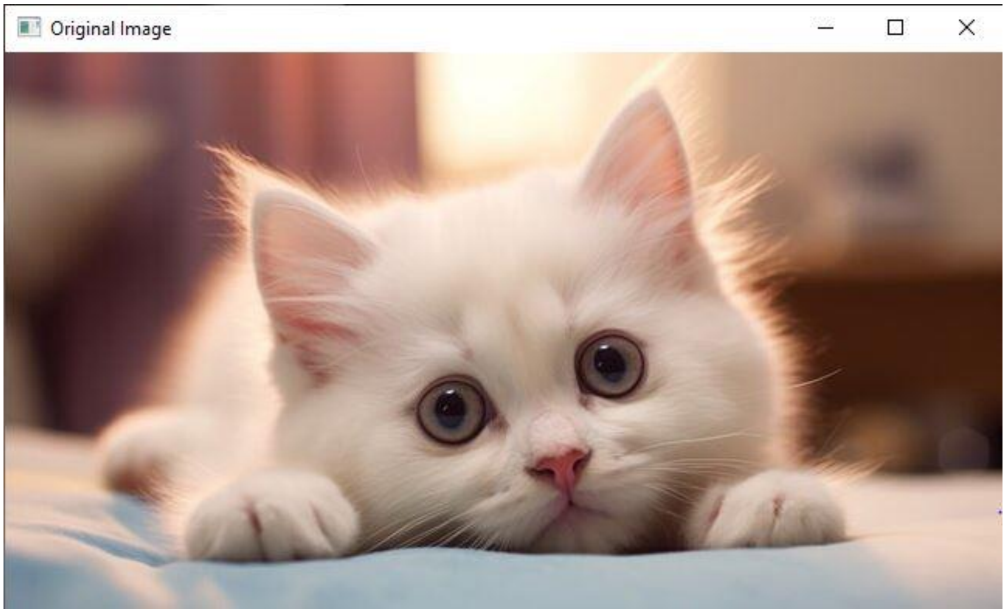


FIGURE 3.23: ORIGINAL IMAGE

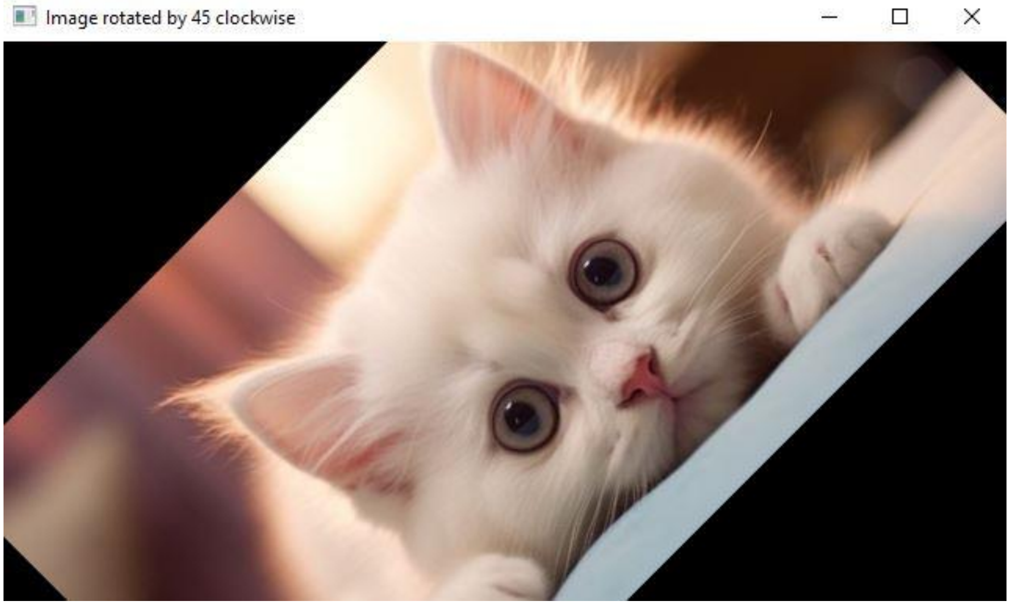


FIGURE 3.24: IMAGE ROTATED BY 45 DEGREES CLOCKWISE

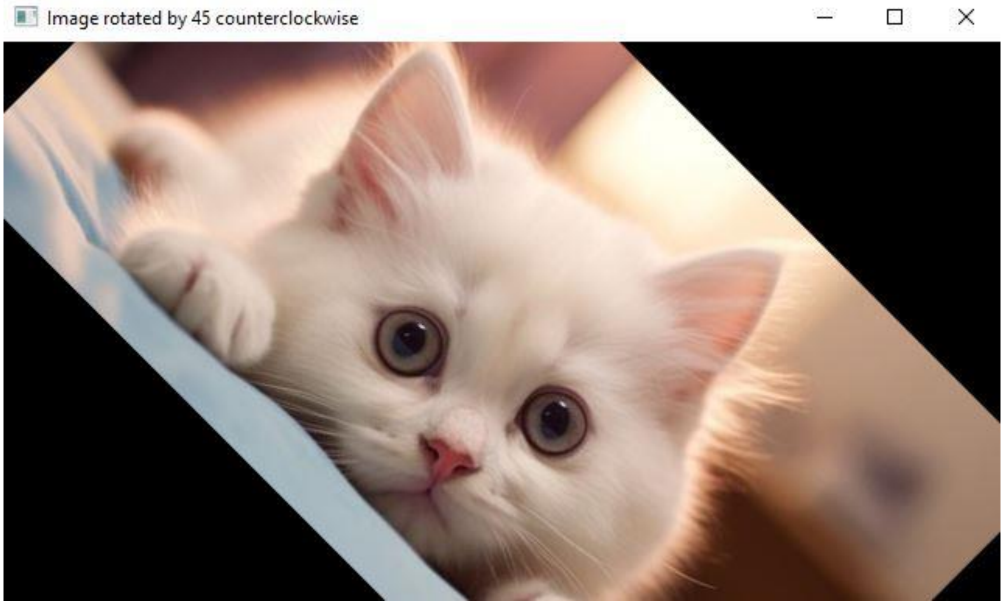


FIGURE 3.25: IMAGE ROTATED BY 45 DEGREES COUNTERCLOCKWISE

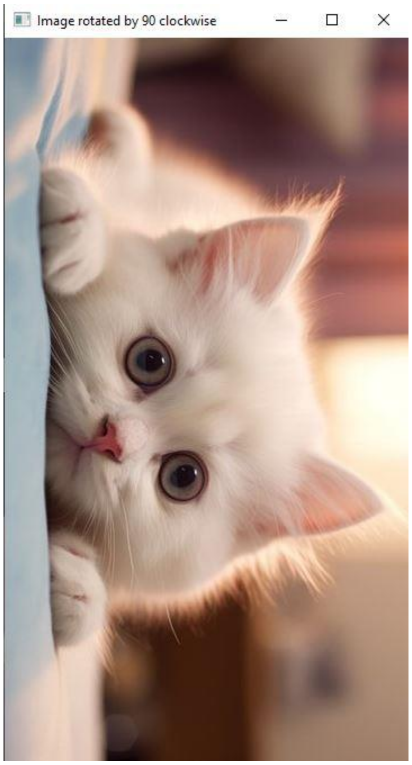


FIGURE 3.26: IMAGE ROTATED BY 90 DEGREES CLOCKWISE

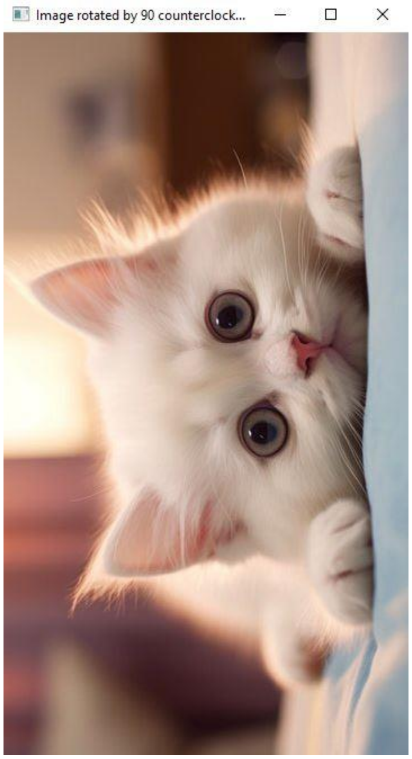


FIGURE 3.27: IMAGE ROTATED BY 90 DEGREES COUNTERCLOCKWISE

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

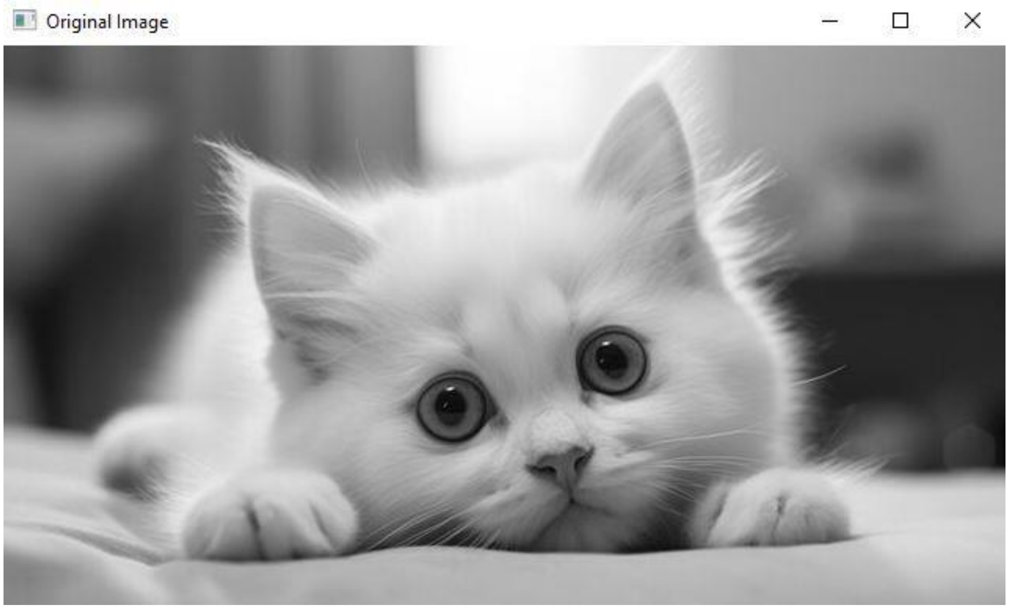


FIGURE 3.28: ORIGINAL IMAGE (IN GRAYSCALE)

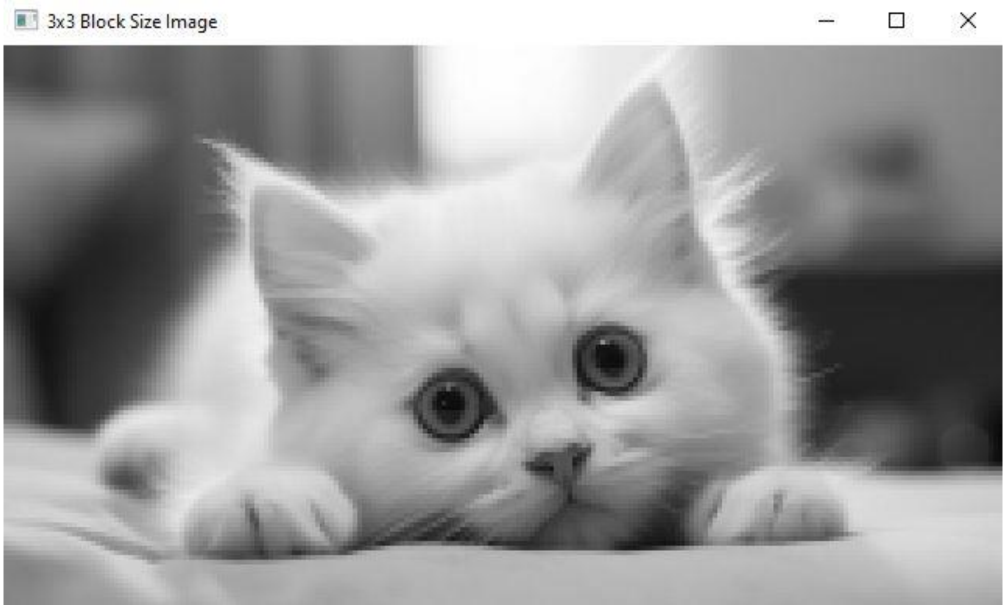


FIGURE 3.29: 3 X 3 BLOCK OF THE IMAGE (WITHOUT OVERLAPPING)

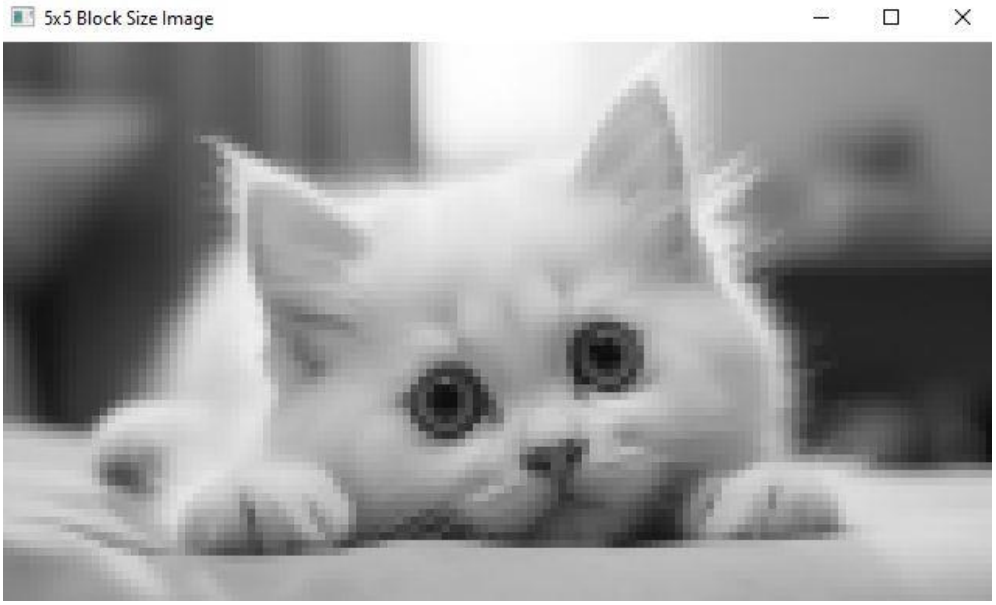


FIGURE 3.30: 5 X 5 BLOCK OF THE IMAGE (WITHOUT OVERLAPPING)

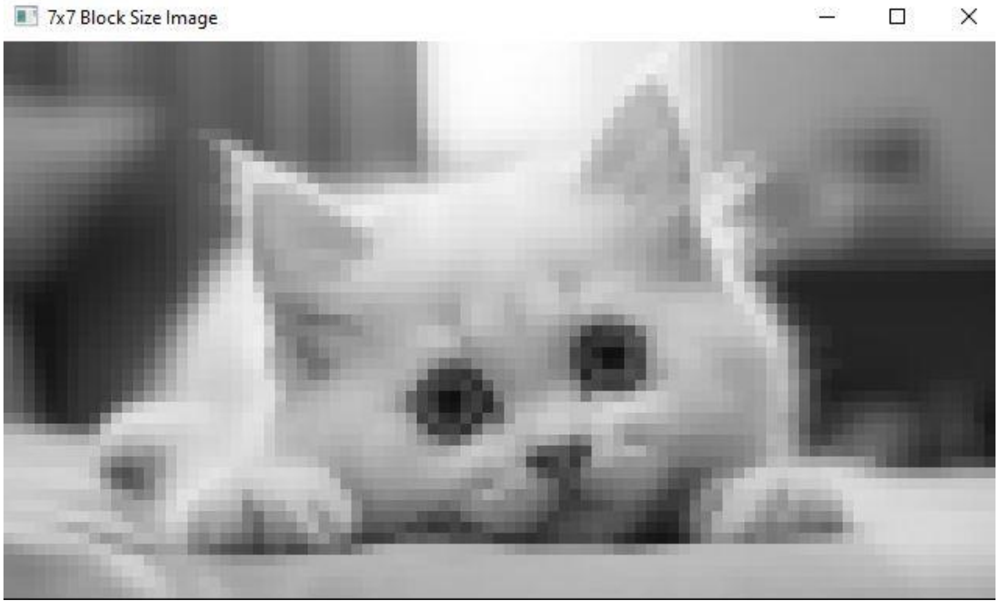


FIGURE 3.31: 7 X 7 BLOCK OF THE IMAGE (WITHOUT OVERLAPPING)