EC7212 : COMPUTER VISION AND IMAGE PROCESSING

ASSIGNMENT 01

NAME	:	MINHAJ M.H.A
REG NO.	:	EG/2020/4076
SEMESTER	:	07
DATE	:	21/06/2025

Table of Contents

List of Figures

GitHub Link

https://github.com/AhamedMinhaj456/Image-processing-techniques-using-python

Questions

Q-01. 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.

Q-02. 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.

Q-03. Rotate an image by 45 and 90 degrees.

Q-04. 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.

Original Image

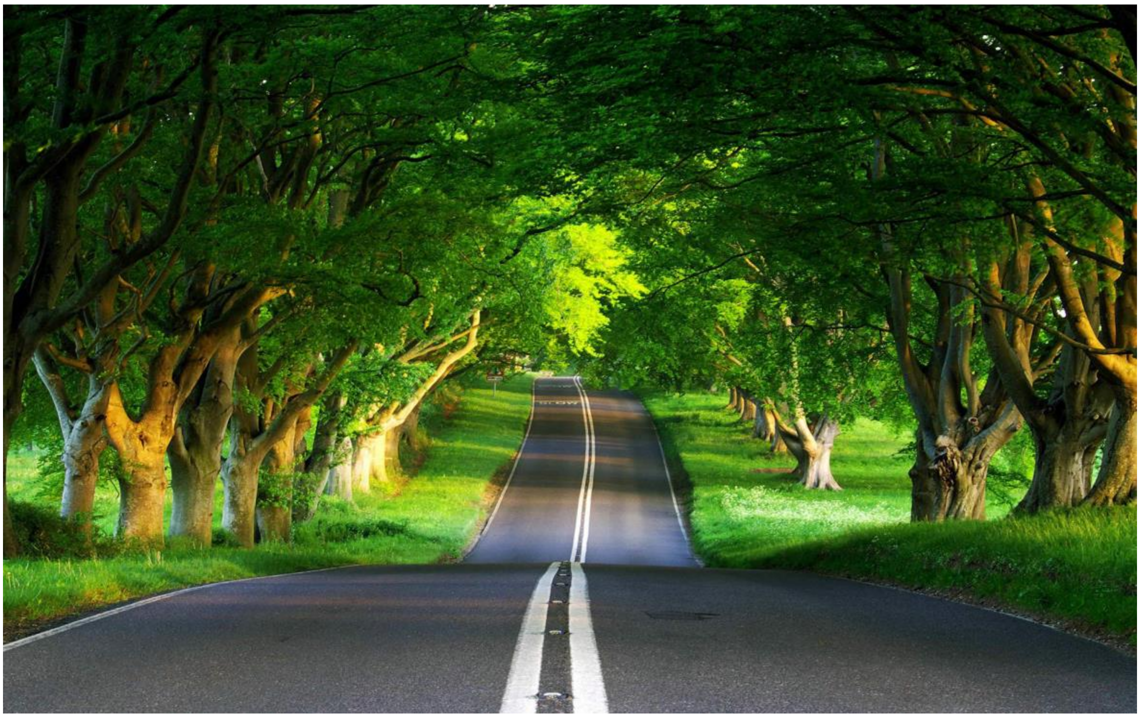

Figure 1: Original Image taken for processing

Q-01

Output:

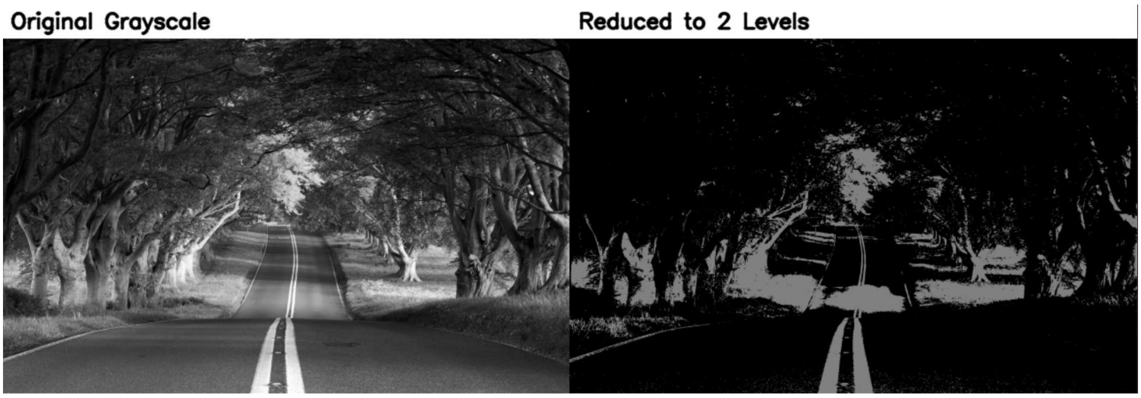

Figure 2: Grayscale Intensity Comparison of Reduce Level 2

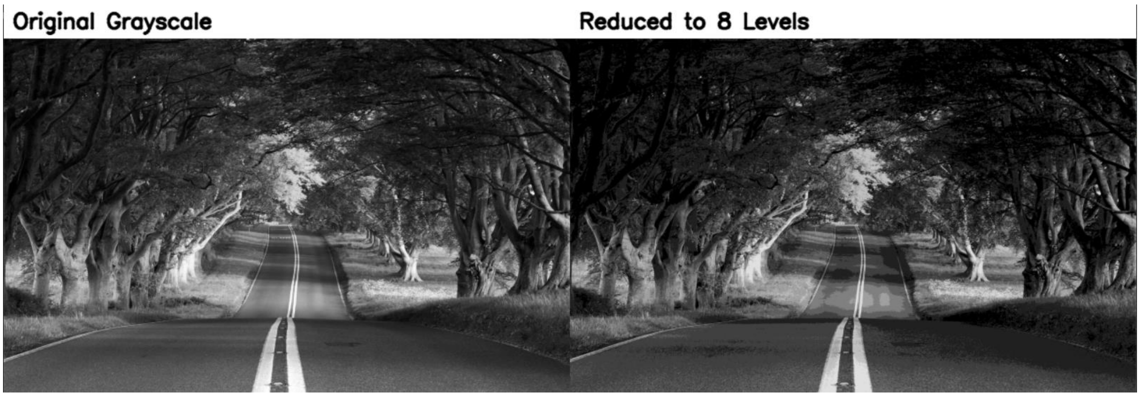

Figure 3: Grayscale Intensity Comparison of Reduce Level 8

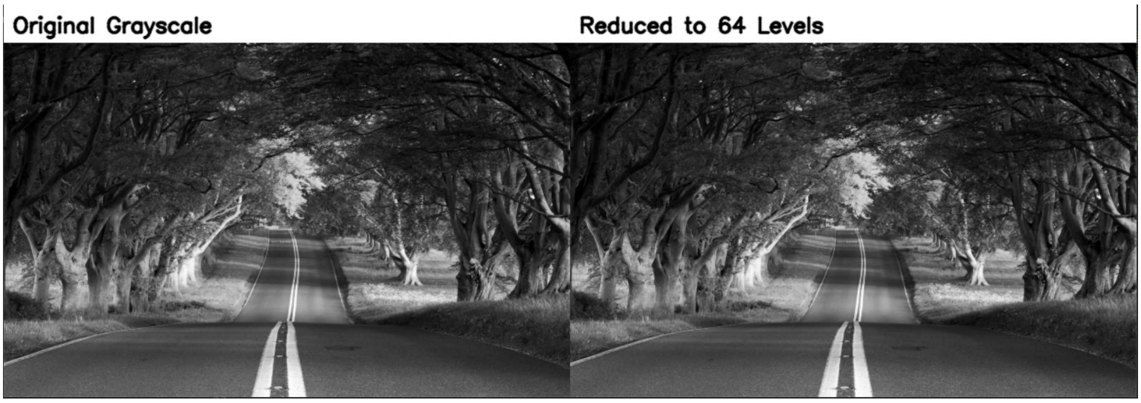

Figure 4: Grayscale Intensity Comparison of Reduce Level 64

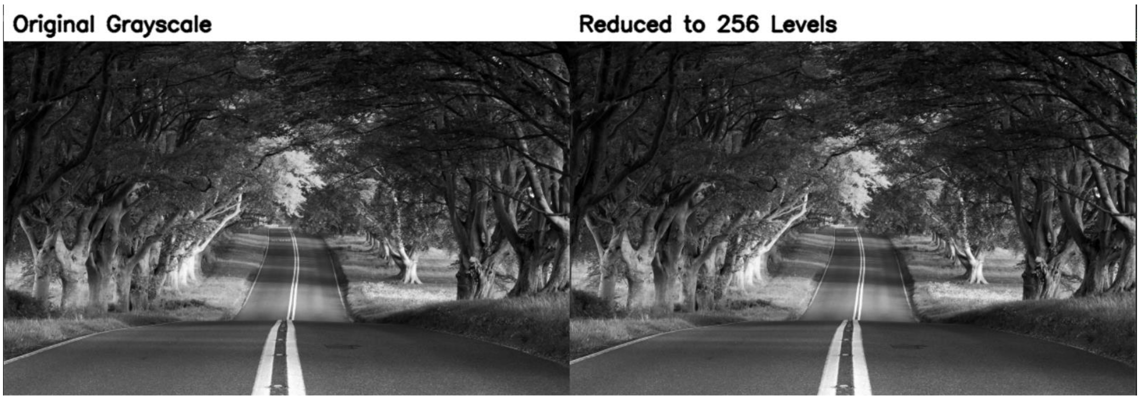

Figure 5: Grayscale Intensity Comparison of Reduce Level 256

Code:

```
import cv2
import numpy as np

def is_power_of_two(n):

    return n >= 2 and n <= 256 and (n & (n - 1)) == 0

def quantize_image(img, levels):
    step = 256 // levels
    return np.floor(img / step) * step

def add_label_to_image(image, label):

    # Adds a label above the image with a white header
    label_height = 40
    header = np.full((label_height, image.shape[1], 3), 255, dtype=np.uint8)
    cv2.putText(header, label, (10, 28), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2, cv2.LINE_AA)
    return np.vstack((header, image))

def resize_to_fit(img, max_height=400):

    # Resize image to fit a specific height while keeping aspect ratio
    h, w = img.shape[:2]
    scale = max_height / h
    new_size = (int(w * scale), max_height)
    return cv2.resize(img, new_size, interpolation=cv2.INTER_AREA)

def process_image(path, levels):

    if not is_power_of_two(levels):

        raise ValueError("Levels must be a power of 2 between 2 and 256.")

    # Load original image in color
    color_img = cv2.imread(path)
    if color_img is None:
        raise FileNotFoundError("Image not found or invalid path.")

    # Convert to grayscale
    gray_img = cv2.cvtColor(color_img, cv2.COLOR_BGR2GRAY)

    # Reduce intensity levels
    reduced_img = quantize_image(gray_img, levels).astype(np.uint8)

    # Convert grayscale images to BGR for color labeling
    gray_bgr = cv2.cvtColor(gray_img, cv2.COLOR_GRAY2BGR)
    reduced_bgr = cv2.cvtColor(reduced_img, cv2.COLOR_GRAY2BGR)

    # Resize both to fit nicely
    gray_bgr_resized = resize_to_fit(gray_bgr)
    reduced_bgr_resized = resize_to_fit(reduced_bgr)

    # Add headers
    labeled_gray = add_label_to_image(gray_bgr_resized, "Original Grayscale")
    labeled_reduced = add_label_to_image(reduced_bgr_resized, f"Reduced to {levels} Levels")

    # Match height before stacking
    final_height = min(labeled_gray.shape[0], labeled_reduced.shape[0])
    labeled_gray = cv2.resize(labeled_gray, (labeled_gray.shape[1], final_height))
    labeled_reduced = cv2.resize(labeled_reduced, (labeled_reduced.shape[1], final_height))

    # Combine side-by-side
    combined = np.hstack((labeled_gray, labeled_reduced))
    return combined

def main():

    try:

        path = input("Enter image path: ").strip()
        levels = int(input("Enter intensity levels (power of 2 between 2 and 256): "))

        combined_image = process_image(path, levels)

        # Show the result
        cv2.imshow("Grayscale Intensity Comparison", combined_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    except Exception as e:
        print("Error:", e)

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

Q-O2

Output:

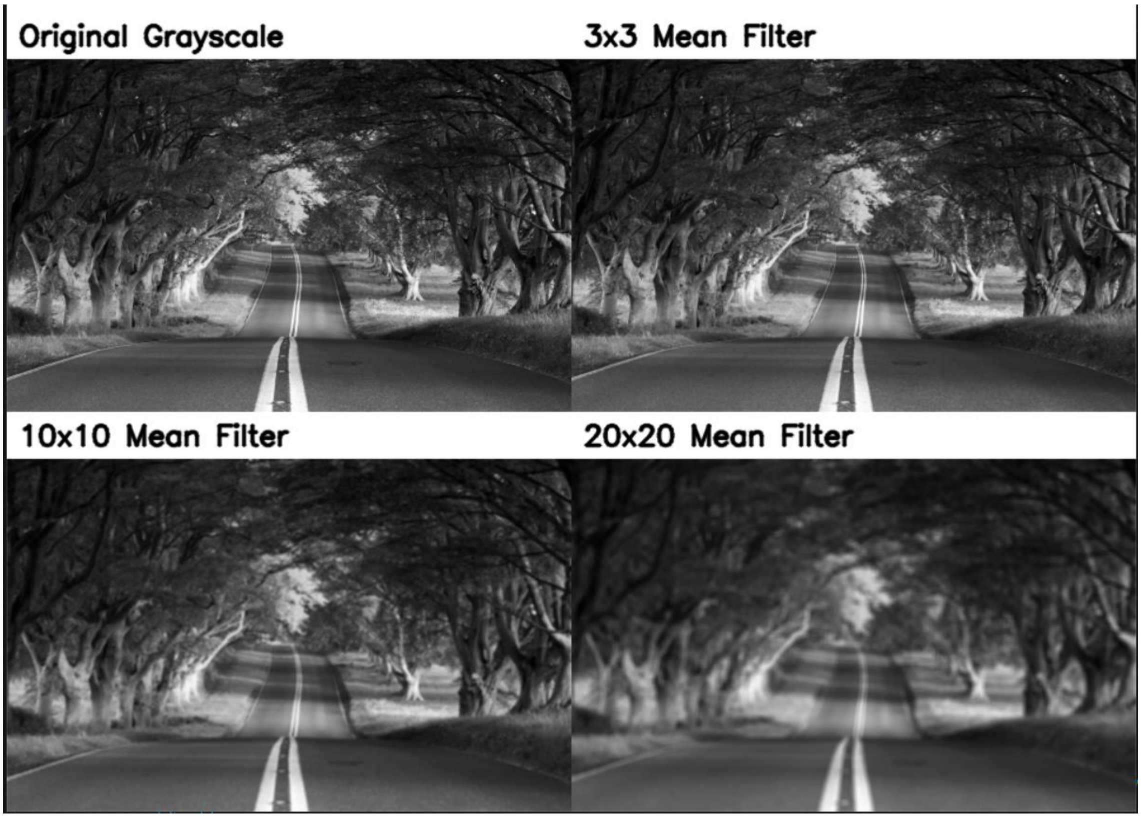

Figure 6: Mean Filter Comparison of 3x3, 10x10 and 20x20

Code:

```
import cv2
import numpy as np

def apply_mean_blur(img, kernel_size):

   # Apply average (mean) blur
   return cv2.blur(img, (kernel_size, kernel_size))

def add_label_to_image(image, label):

   # Add a label above the image
   label_height = 40
   header = np.full((label_height, image.shape[1], 3), 255, dtype=np.uint8)
   cv2.putText(header, label, (10, 28), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2,
cv2.LINE_AA)
   return np.vstack((header, image))

def resize_to_fit(img, target_height=300):

   # Resize image to target height keeping aspect ratio
   h, w = img.shape[:2]
   scale = target_height / h
   new_size = (int(w * scale), target_height)
   return cv2.resize(img, new_size, interpolation=cv2.INTER_AREA)
```

```
def main():

    try:

        path = input("Enter image path: ").strip()

        original = cv2.imread(path)
        # Check if the image was loaded successfully
        if original is None:
            raise FileNotFoundError("Image not found or path incorrect.")

        gray = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)

        # Apply filters
        blur_3x3 = apply_mean_blur(gray, 3)
        blur_10x10 = apply_mean_blur(gray, 10)
        blur_20x20 = apply_mean_blur(gray, 20)

        # Convert all to BGR for consistent display
        gray_bgr = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
        blur3_bgr = cv2.cvtColor(blur_3x3, cv2.COLOR_GRAY2BGR)
        blur10_bgr = cv2.cvtColor(blur_10x10, cv2.COLOR_GRAY2BGR)
        blur20_bgr = cv2.cvtColor(blur_20x20, cv2.COLOR_GRAY2BGR)

        # Resize for consistent display
        height = 300  # Set target height
        gray_bgr = resize_to_fit(gray_bgr, height)
        blur3_bgr = resize_to_fit(blur3_bgr, height)
        blur10_bgr = resize_to_fit(blur10_bgr, height)
        blur20_bgr = resize_to_fit(blur20_bgr, height)

        # Add labels
        labeled_gray = add_label_to_image(gray_bgr, "Original Grayscale")
        labeled_3x3 = add_label_to_image(blur3_bgr, "3x3 Mean Filter")
        labeled_10x10 = add_label_to_image(blur10_bgr, "10x10 Mean Filter")
        labeled_20x20 = add_label_to_image(blur20_bgr, "20x20 Mean Filter")

        # Arrange in 2x2 grid
        top_row = np.hstack((labeled_gray, labeled_3x3))
        bottom_row = np.hstack((labeled_10x10, labeled_20x20))
        grid = np.vstack((top_row, bottom_row))

        # Show result
        cv2.imshow("Mean Filtering - 2x2 Layout", grid)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    except Exception as e:
        print("Error:", e)

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

Q-03

Output:

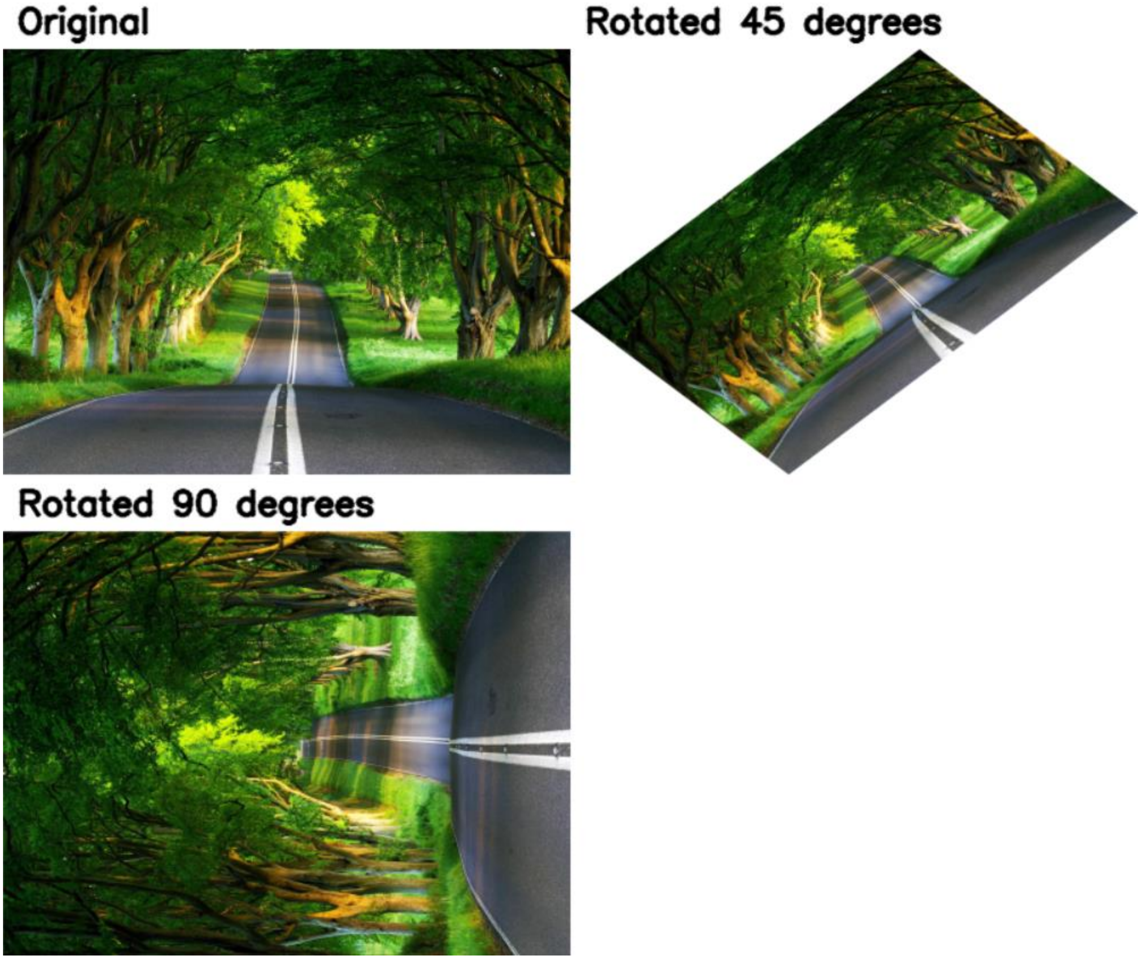

Figure 7: Image Rotation of 45 and 90 degree

Code:

```
import cv2
import numpy as np

def rotate_image(image, angle):

    # Rotate the image around its center without cropping
    h, w = image.shape[:2]
    center = (w // 2, h // 2)

    # Get the rotation matrix
    matrix = cv2.getRotationMatrix2D(center, angle, 1.0)

    # Compute the bounding box of the rotated image
    cos = np.abs(matrix[0, 0])
    sin = np.abs(matrix[0, 1])
    new_w = int((h * sin) + (w * cos))
    new_h = int((h * cos) + (w * sin))

    # Adjust the rotation matrix to take into account translation
    matrix[0, 2] += (new_w / 2) - center[0]
    matrix[1, 2] += (new_h / 2) - center[1]

    # Perform the rotation
    return cv2.warpAffine(image, matrix, (new_w, new_h), borderValue=(255, 255, 255))

def add_label_to_image(image, label):

    # Add a label above the image
    label_height = 40
    header = np.full((label_height, image.shape[1], 3), 255, dtype=np.uint8)
    cv2.putText(header, label, (10, 28), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2, cv2.LINE_AA)
    return np.vstack((header, image))

def resize_to_fit(img, target_height=300):

    # Resize image to a target height keeping aspect ratio
    h, w = img.shape[:2]
    scale = target_height / h
    new_size = (int(w * scale), target_height)
    return cv2.resize(img, new_size, interpolation=cv2.INTER_AREA)

def resize_to_size(img, size=(400, 300)):
    return cv2.resize(img, size, interpolation=cv2.INTER_AREA)

# Main function to handle image processing
def main():

    try:
        path = input("Enter image path: ").strip()

        image = cv2.imread(path)
        if image is None:
            raise FileNotFoundError("Image not found or path incorrect.")

        rotated_45 = rotate_image(image, 45)
        rotated_90 = rotate_image(image, 90)

        size = (400, 300)  # fixed size for all images (width, height)
        original_resized = resize_to_size(image, size)
        rotated_45_resized = resize_to_size(rotated_45, size)
        rotated_90_resized = resize_to_size(rotated_90, size)

        labeled_original = add_label_to_image(original_resized, "Original")
        labeled_45 = add_label_to_image(rotated_45_resized, "Rotated 45 degrees")
        labeled_90 = add_label_to_image(rotated_90_resized, "Rotated 90 degrees")

        blank = np.full_like(labeled_original, 255)
        top_row = np.hstack((labeled_original, labeled_45))
        bottom_row = np.hstack((labeled_90, blank))
        grid = np.vstack((top_row, bottom_row))

        cv2.imshow("Image Rotation - 45° and 90°", grid)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    except Exception as e:
        print("Error:", e)

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

Q-04

Output:

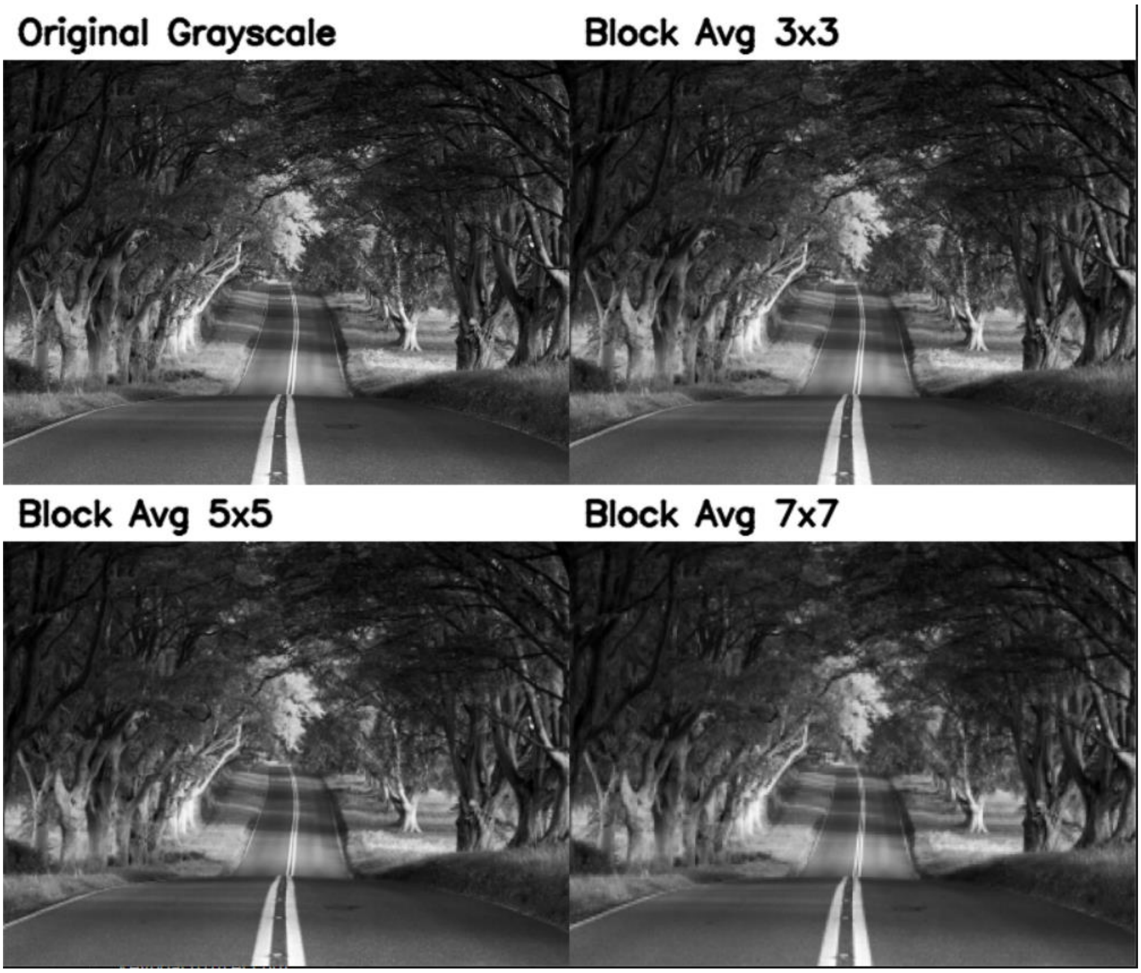

Figure 8: Block Average Down sampling of 3x3, 5x5 and 7x7

Code:

```
import cv2
import numpy as np

def block_average_downsample(img, block_size):

    """
    Replace each non-overlapping block_size x block_size block with its average.
    Works for grayscale images.
    """
    h, w = img.shape

    # Crop image to make dimensions multiples of block_size
    h_crop = h - (h % block_size)
    w_crop = w - (w % block_size)
    img_cropped = img[:h_crop, :w_crop].copy()

    for row in range(0, h_crop, block_size):
        for col in range(0, w_crop, block_size):
            block = img_cropped[row:row+block_size, col:col+block_size]
            avg_val = int(np.mean(block))
            img_cropped[row:row+block_size, col:col+block_size] = avg_val

    return img_cropped

def add_label(image, label):

    # Add a white label bar with text above the image
    label_height = 40
    width = image.shape[1]
    header = np.full((label_height, width, 3), 255, dtype=np.uint8)
    cv2.putText(header, label, (10, 28),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2, cv2.LINE_AA)
    return np.vstack((header, image))

def resize_to_fixed_size(img, size=(400, 300)):

    # Resize image to fixed size (width, height)
    return cv2.resize(img, size, interpolation=cv2.INTER_AREA)

# Main function to handle image processing
def main():

    try:
        path = input("Enter image path: ").strip()
        original = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        if original is None:
            raise FileNotFoundError("Image not found or path incorrect.")

        # Block average downsampling
        avg_3 = block_average_downsample(original, 3)
        avg_5 = block_average_downsample(original, 5)
        avg_7 = block_average_downsample(original, 7)

        # Convert grayscale images to BGR for display and labeling
        orig_bgr = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
        avg3_bgr = cv2.cvtColor(avg_3, cv2.COLOR_GRAY2BGR)
        avg5_bgr = cv2.cvtColor(avg_5, cv2.COLOR_GRAY2BGR)
        avg7_bgr = cv2.cvtColor(avg_7, cv2.COLOR_GRAY2BGR)

        # Resize all images to the same fixed size
        fixed_size = (400, 300)  # width x height
        orig_bgr = resize_to_fixed_size(orig_bgr, fixed_size)
        avg3_bgr = resize_to_fixed_size(avg3_bgr, fixed_size)
        avg5_bgr = resize_to_fixed_size(avg5_bgr, fixed_size)
        avg7_bgr = resize_to_fixed_size(avg7_bgr, fixed_size)

        # Add labels above images
        labeled_orig = add_label(orig_bgr, "Original Grayscale")
        labeled_3 = add_label(avg3_bgr, "Block Avg 3x3")
        labeled_5 = add_label(avg5_bgr, "Block Avg 5x5")
        labeled_7 = add_label(avg7_bgr, "Block Avg 7x7")

        # Create 2x2 grid
        top_row = np.hstack((labeled_orig, labeled_3))
        bottom_row = np.hstack((labeled_5, labeled_7))
        grid = np.vstack((top_row, bottom_row))

        # Show the final result
        cv2.imshow("Block Average Downsampling (2x2 grid)", grid)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    except Exception as e:
        print("Error:", e)

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