EC7212 : COMPUTER VISION AND IMAGE PROCESSING

ASSIGNMENT 01

NAME : MORAIS MNS

REG NO : EG/2020/4077

SEMESTER : 07

DATE : 21/06/2025

Table of Contents

List Of Figures

GitHub Link

https://github.com/MoraisMNS/Image-processing-using-python

Original Image

Figure 1: original image

Question 01

Code:

```
import cv2
import numpy as np

def validate_levels(level_count):

    # Ensure level count is a valid power of 2 between 2 and 256
    return level_count >= 2 and level_count <= 256 and (level_count & (level_count - 1)) == 0

def reduce_levels(grayscale_img, level_count):

    # Reduce grayscale intensity levels to the specified count
    factor = 256 // level_count
    return (grayscale_img // factor) * factor

def overlay_title(img, text):

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

def scale_image(img, target_height=400):

    # Scale an image to a given height while keeping aspect ratio
    h, w = img.shape[:2]
    ratio = target_height / h
    new_dims = (int(w * ratio), target_height)
    return cv2.resize(img, new_dims, interpolation=cv2.INTER_AREA)

def prepare_display_images(image_path, level_count):

    # Process the image and prepare display-ready original and reduced images
    if not validate_levels(level_count):
        raise ValueError("Intensity levels must be a power of 2 between 2 and 256.")

    original = cv2.imread(image_path)

    if original is None:
        raise FileNotFoundError("Invalid image path.")

    gray = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
    reduced = reduce_levels(gray, level_count).astype(np.uint8)

    # Convert for display
    gray_disp = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
    reduced_disp = cv2.cvtColor(reduced, cv2.COLOR_GRAY2BGR)

    # Resize and label
    gray_disp = scale_image(gray_disp)
    reduced_disp = scale_image(reduced_disp)

    gray_disp = overlay_title(gray_disp, "Original Grayscale")
    reduced_disp = overlay_title(reduced_disp, f"Quantized to {level_count} Levels")

    # Match height for horizontal stack
    min_height = min(gray_disp.shape[0], reduced_disp.shape[0])
    gray_disp = cv2.resize(gray_disp, (gray_disp.shape[1], min_height))
    reduced_disp = cv2.resize(reduced_disp, (reduced_disp.shape[1], min_height))
    return np.hstack((gray_disp, reduced_disp))
def main():
```

```
    try:

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

        cv2.imshow("Intensity Level Reduction", result)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

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

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

.

Output:

Figure 2: Intensity level reduction of 2

Figure 3: Intensity level reduction of 4

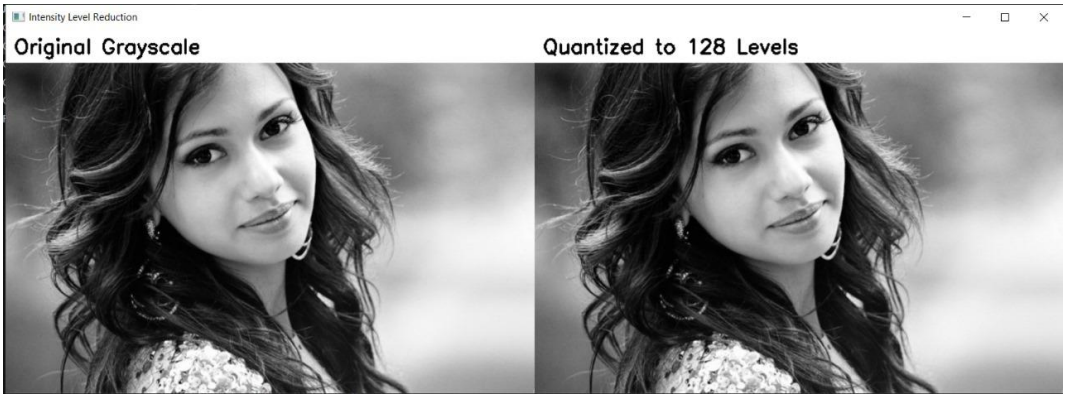

Figure 4 Intensity level reduction of 128

Question 02

Code:

```
import cv2
import numpy as np

def mean_filter(image, size):

    # Apply a mean filter of specified kernel size
    return cv2.blur(image, (size, size))

def label_image(image, title):

    # Add a white label bar with title above an image
    bar_height = 40
    label_bar = np.ones((bar_height, image.shape[1], 3), dtype=np.uint8) * 255
    cv2.putText(label_bar, title, (10, 28), cv2.FONT_HERSHEY_SIMPLEX,
                0.8, (0, 0, 0), 2, cv2.LINE_AA)
    return np.vstack((label_bar, image))

def resize_image(img, height=300):

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

def process_filters(image_path):

    # Process the grayscale image with multiple mean filters and return a 2x2 grid
    original = cv2.imread(image_path)

    if original is None:
        raise ValueError("Invalid image path or unable to load image.")

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

    # Apply different mean filter sizes
    filtered_images = {
        "Original Grayscale": gray,
        "3x3 Mean Filter": mean_filter(gray, 3),
        "10x10 Mean Filter": mean_filter(gray, 10),
        "20x20 Mean Filter": mean_filter(gray, 20)
    }

    # Convert to BGR and prepare labeled images
    labeled_blocks = []

    for label, img in filtered_images.items():
        bgr = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        resized = resize_image(bgr)
        labeled = label_image(resized, label)
        labeled_blocks.append(labeled)

    # Combine into a 2x2 grid
    top = np.hstack((labeled_blocks[0], labeled_blocks[1]))
    bottom = np.hstack((labeled_blocks[2], labeled_blocks[3]))
    return np.vstack((top, bottom))

def main():

    try:
        img_path = input("Enter the path to the image: ").strip()
        result_grid = process_filters(img_path)
```

```
        cv2.imshow("Mean Filtering Comparison", result_grid)
        cv2.waitKey(0)
        cv2.destroyAllWindows()  # ✅ Properly ends the try block

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

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

Output:

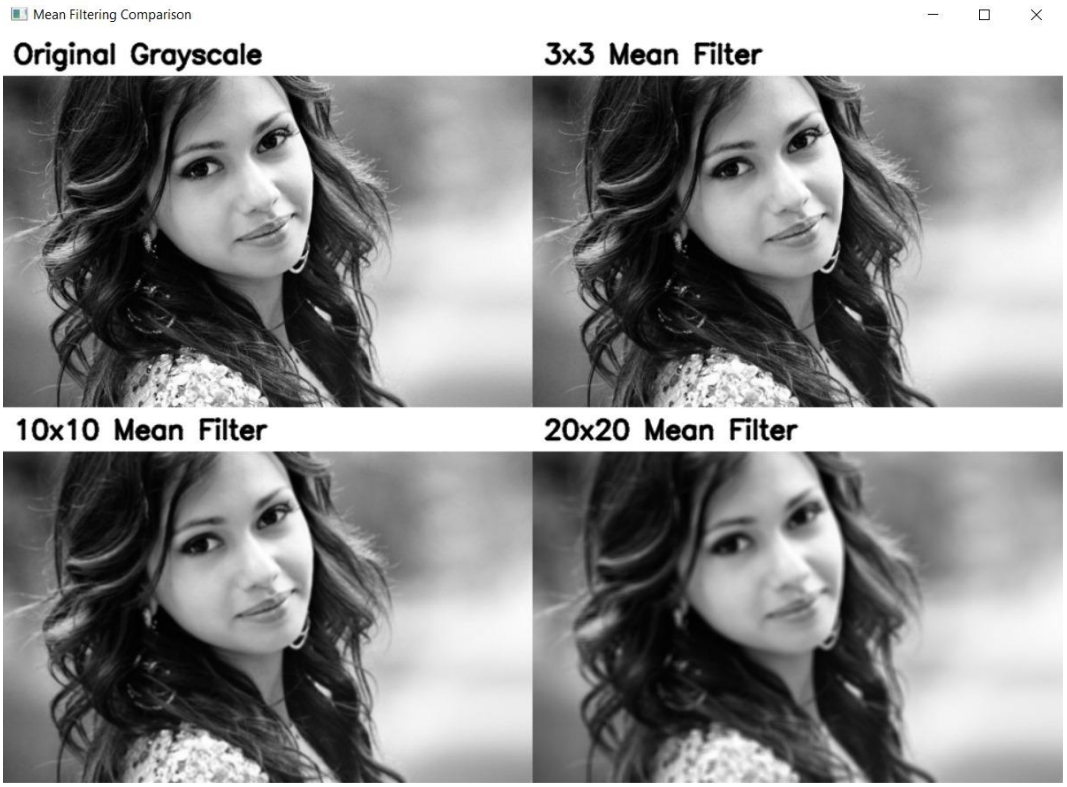

Figure 5: Mean Filtering comparison

Question 03

Code:

```
import cv2
import numpy as np

def rotate(img, angle_deg):

    # Rotate the image by a given angle without cropping, adding white background
    h, w = img.shape[:2]
    center = (w // 2, h // 2)

    rot_matrix = cv2.getRotationMatrix2D(center, angle_deg, 1.0)

    cos_a = abs(rot_matrix[0, 0])
    sin_a = abs(rot_matrix[0, 1])

    new_w = int((h * sin_a) + (w * cos_a))
    new_h = int((h * cos_a) + (w * sin_a))

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

    return cv2.warpAffine(img, rot_matrix, (new_w, new_h), borderValue=(255, 255, 255))

def label(img, text):
    # Add a white label bar on top of the image with text
    label_bar = np.ones((40, img.shape[1], 3), dtype=np.uint8) * 255
    cv2.putText(label_bar, text, (10, 28), cv2.FONT_HERSHEY_SIMPLEX,
                0.8, (0, 0, 0), 2, cv2.LINE_AA)
    return np.vstack((label_bar, img))

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

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

def build_display_grid(images_with_labels, grid_shape=(2, 2)):

    # Arrange images into a labeled grid layout
    row_imgs = []

    for i in range(grid_shape[0]):

        row = images_with_labels[i * grid_shape[1]:(i + 1) * grid_shape[1]]
        while len(row) < grid_shape[1]:
            row.append(np.full_like(row[0], 255))  # Add blank space if needed
        row_imgs.append(np.hstack(row))
    return np.vstack(row_imgs)

def main():

    try:
        img_path = input("Enter the image path: ").strip()
        original = cv2.imread(img_path)
        if original is None:
            raise FileNotFoundError("Unable to load image.")

        rotated_45 = rotate(original, 45)
        rotated_90 = rotate(original, 90)

        fixed_size = (400, 300)
        images = [
            label(resize_fixed(original, fixed_size), "Original"),
```

```
            label(resize_fixed(rotated_45, fixed_size), "Rotated 45 degree"),
            label(resize_fixed(rotated_90, fixed_size), "Rotated 90 degree")
        ]

        grid_img = build_display_grid(images, grid_shape=(2, 2))

        cv2.imshow("Image Rotation Comparison", grid_img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

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

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

Output:

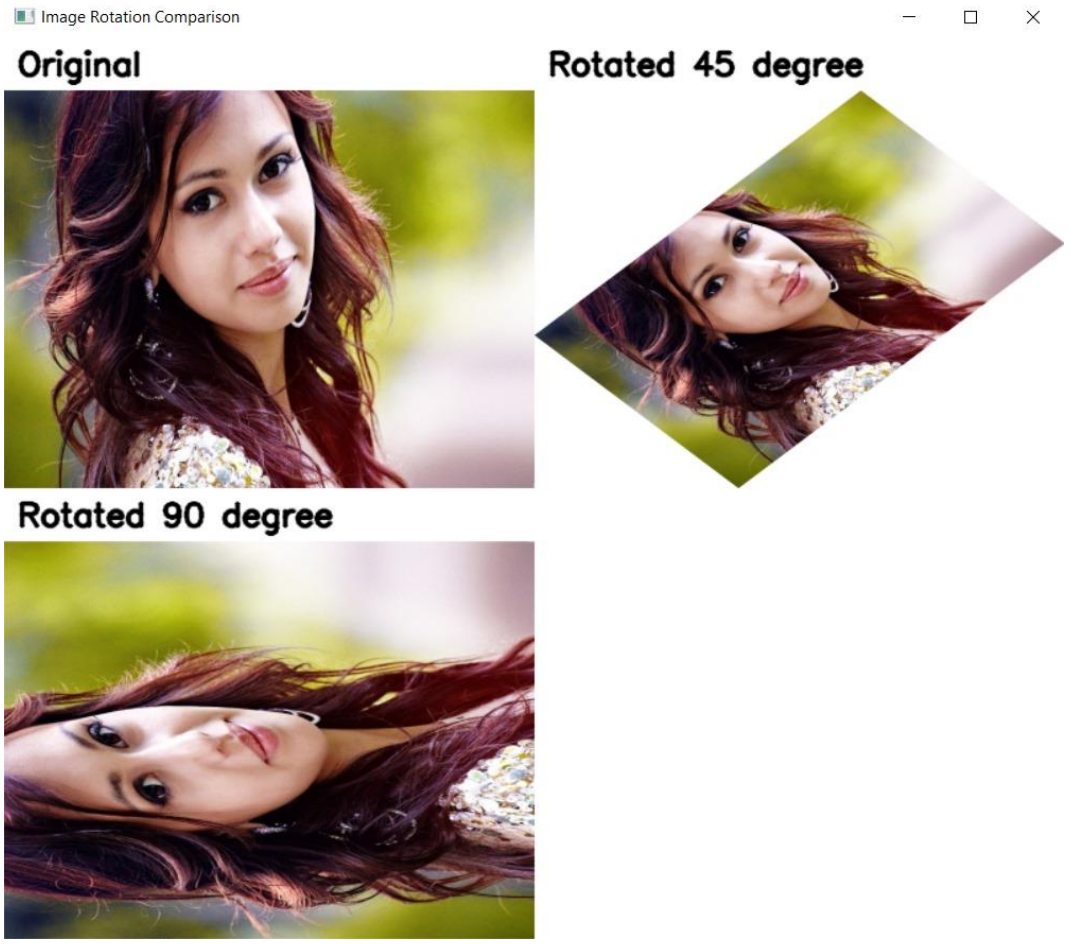

Figure 6: Image rotation Comparison

Question 04

Code:

```
import cv2
import numpy as np

def average_pooling(img, ksize):


    # Reduce image resolution by replacing non-overlapping blocks with their average.
    # Works on grayscale images.

    height, width = img.shape
    h_trim = height - (height % ksize)
    w_trim = width - (width % ksize)
    cropped = img[:h_trim, :w_trim].copy()

    for i in range(0, h_trim, ksize):
        for j in range(0, w_trim, ksize):
            block = cropped[i:i+ksize, j:j+ksize]
            avg = int(np.mean(block))
            cropped[i:i+ksize, j:j+ksize] = avg

    return cropped

def add_caption(img, caption):

    # Overlay a text label above the image
    caption_bar = np.ones((40, img.shape[1], 3), dtype=np.uint8) * 255
    cv2.putText(caption_bar, caption, (10, 28),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2)
    return np.vstack((caption_bar, img))

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

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

def process_and_prepare(image_path):

    # Process the image and prepare labeled versions with reduced resolutions
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise FileNotFoundError("Unable to load image from the specified path.")

    # Apply average pooling for multiple block sizes
    versions = {
        "Original Grayscale": img,
        "Block Avg 3x3": average_pooling(img, 3),
        "Block Avg 5x5": average_pooling(img, 5),
        "Block Avg 7x7": average_pooling(img, 7),
    }

    fixed_size = (400, 300)
    labeled_images = []

    for label, image in versions.items():
        bgr_img = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
        resized = resize_uniform(bgr_img, fixed_size)
        labeled = add_caption(resized, label)
        labeled_images.append(labeled)

    return labeled_images
```

```
def arrange_grid(images, rows=2, cols=2):

    # Arrange labeled images into a grid of given rows and columns
    grid_rows = []
    for i in range(rows):
        row = images[i*cols:(i+1)*cols]
        if len(row) < cols:
            row += [np.full_like(row[0], 255)] * (cols - len(row))
        grid_rows.append(np.hstack(row))
    return np.vstack(grid_rows)

def main():

    try:
        path = input("Enter image path: ").strip()
        images = process_and_prepare(path)
        final_grid = arrange_grid(images, rows=2, cols=2)

        cv2.imshow("Spatial Resolution Reduction", final_grid)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    except Exception as err:
        print("Error:", err)

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

Output:

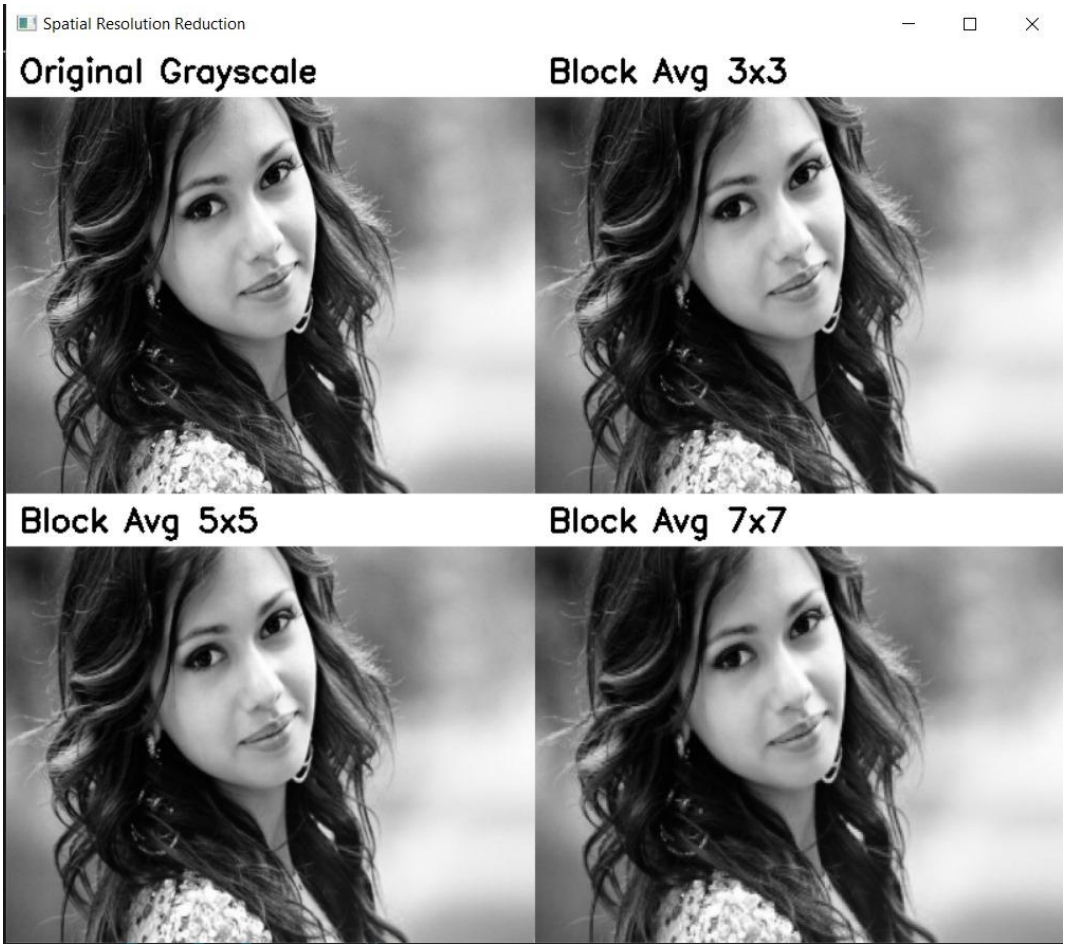

Figure 7: Spatial Resolution reduction