EE604 - Assignment 1

Lohit P Talavar

September 26, 2025

Q1: Text-to-Image Generation

Code

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt

# --- Part A: Text-to-Image Generation ---

# A simple bitmap font dictionary. Each character is represented by an 8x6 matrix.
# 1 represents a pixel to be drawn (black), 0 represents background (white).
BITMAP_FONT = {
    'A': np.array([
        [0, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'B': np.array([
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
```

```
    'C': np.array([
        [0, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'D': np.array([
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'E': np.array([
        [1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'F': np.array([
        [1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'G': np.array([
        [0, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 1, 1, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
```

```
        [0, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'H': np.array([
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'I': np.array([
        [1, 1, 1, 1, 1, 1],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [1, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'J': np.array([
        [0, 0, 0, 1, 1, 1],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 1, 0],
        [1, 0, 0, 0, 1, 0],
        [1, 0, 0, 0, 1, 0],
        [0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'K': np.array([
        [1, 0, 0, 0, 1, 0],
        [1, 0, 0, 1, 0, 0],
        [1, 0, 1, 0, 0, 0],
        [1, 1, 0, 0, 0, 0],
        [1, 0, 1, 0, 0, 0],
        [1, 0, 0, 1, 0, 0],
        [1, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'L': np.array([
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
```

```
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'M': np.array([
        [1, 0, 0, 0, 0, 1],
        [1, 1, 0, 0, 1, 1],
        [1, 0, 1, 1, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'N': np.array([
        [1, 0, 0, 0, 0, 1],
        [1, 1, 0, 0, 0, 1],
        [1, 0, 1, 0, 0, 1],
        [1, 0, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'O': np.array([
        [0, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'P': np.array([
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'Q': np.array([
```

```
        [0, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 1, 0, 1],
        [1, 0, 0, 0, 1, 0],
        [0, 1, 1, 1, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'R': np.array([
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [1, 0, 0, 1, 0, 0],
        [1, 0, 0, 0, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    'S': np.array([
        [0, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [0, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'T': np.array([
        [1, 1, 1, 1, 1, 1],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    'U': np.array([
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 1],
        [0, 1, 1, 1, 1, 0],
```

```
            [0, 0, 0, 0, 0, 0]
        ]),
        'V': np.array([
            [1, 0, 0, 0, 0, 1],
            [1, 0, 0, 0, 0, 1],
            [1, 0, 0, 0, 0, 1],
            [0, 1, 0, 0, 1, 0],
            [0, 1, 0, 0, 1, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 0, 0, 0, 0]
        ]),
        'W': np.array([
            [1, 0, 0, 0, 0, 1],
            [1, 0, 0, 0, 0, 1],
            [1, 0, 0, 0, 0, 1],
            [1, 0, 0, 0, 0, 1],
            [1, 0, 1, 1, 0, 1],
            [1, 1, 0, 0, 1, 1],
            [1, 0, 0, 0, 0, 1],
            [0, 0, 0, 0, 0, 0]
        ]),
        'X': np.array([
            [1, 0, 0, 0, 0, 1],
            [0, 1, 0, 0, 1, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 1, 0, 0, 1, 0],
            [1, 0, 0, 0, 0, 1],
            [0, 0, 0, 0, 0, 0]
        ]),
        'Y': np.array([
            [1, 0, 0, 0, 0, 1],
            [0, 1, 0, 0, 1, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 1, 1, 0, 0],
            [0, 0, 0, 0, 0, 0]
        ]),
        'Z': np.array([
            [1, 1, 1, 1, 1, 1],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 1, 0, 0],
            [0, 0, 1, 0, 0, 0],
```

```
        [0, 1, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1],
        [0, 0, 0, 0, 0, 0]
    ]),
    ' ': np.zeros((8, 6), dtype=int),
    '.': np.array([
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    '!': np.array([
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ]),
    '?': np.array([
        [0, 1, 1, 1, 1, 0],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0]
    ])
}

def text_to_image(text_input, char_height=8, char_width=6, padding
    =10):
    """
    Renders a string of text to an image using a custom bitmap font.

    Args:
        text_input (str): The string to render.
        char_height (int): The height of a single character in
            pixels.
```

```
        char_width (int): The width of a single character in pixels.
        padding (int): The padding around the text in the final image.

    Returns:
        numpy.ndarray: The rendered image as a NumPy array.
    """
    # Filter out any characters not in our font map
    lines = text_input.upper().split('\n')
    valid_lines = []
    for line in lines:
        valid_line = "".join([char for char in line if char in BITMAP_FONT])
        valid_lines.append(valid_line)

    # Calculate image dimensions
    num_lines = len(valid_lines)
    max_line_length = 0
    if valid_lines:
        max_line_length = max(len(line) for line in valid_lines)

    img_height = num_lines * char_height + 2 * padding
    img_width = max_line_length * char_width + 2 * padding

    # Create a white canvas (255 is white in grayscale)
    canvas = np.full((img_height, img_width), 255, dtype=np.uint8)

    # Render each character pixel by pixel
    y_cursor = padding
    for line in valid_lines:
        x_cursor = padding
        for char in line:
            char_map = BITMAP_FONT[char]

            # Define the region on the canvas to draw the character
            y_start, y_end = y_cursor, y_cursor + char_height
            x_start, x_end = x_cursor, x_cursor + char_width

            # Iterate through the bitmap and draw pixels
            for row_idx, row in enumerate(char_map):
                for col_idx, pixel_val in enumerate(row):
                    if pixel_val == 1:
                        # Set pixel to black (0)
                        canvas[y_start + row_idx, x_start + col_idx] = 0

            x_cursor += char_width
```

```
            y_cursor += char_height

    return canvas


if __name__ == '__main__':
    # --- Example Usage ---
    input_text = (
        "HELLO WORLD!\n"
        "THIS IS A TEST OF THE\n"
        "TEXT TO IMAGE SYSTEM.\n"
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ\n"
        "1234567890?.\n"
        "THIS IS ME LOHIT P TALAVAR\n"
        "THIS IS IMAGE PROCESSING"
    )

    # Generate the image
    rendered_image = text_to_image(input_text)

    # --- Display the Result ---
    plt.figure(figsize=(10, 8))
    plt.imshow(rendered_image, cmap='gray')
    plt.title('Rendered Text')
    plt.axis('off') # Hide axes for a cleaner look
    plt.tight_layout()
    plt.show()

    # cv2.imwrite('rendered_text.png', rendered_image)
```

Listing 1: Code for Text-to-Image Generation (‘a.py‘)

Results

Q2: Leopard Spot Removal

Code

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def remove_leopard_spots(image_path):
    """
    Removes 'leopard spots' (dark regions) from an image using HSV
       color segmentation
```

Figure 1: Output of the text-to-image generation system.

```
    and inpainting techniques.

    Args:
        image_path (str): The path to the input image file.

    Returns:
        None: Displays the original, mask, and inpainted images.
    """
    # --- Image Loading and Preprocessing ---
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Convert to RGB for matplotlib display later
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Convert to HSV color space for better color segmentation
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # --- Step 1: Create a mask for the spots ---
    # This is the trickiest part and will likely require fine-tuning
       .
    # We're looking for dark brown/black spots.
    # Define a range for dark colors in HSV. These values are
       approximate
    # and might need adjustment based on the specific image.

    # Lower bound for dark brown/black (adjust these!)
```

```
    lower_spot_hsv = np.array([0, 0, 0]) # Hue, Saturation, Value (V = brightness)
    upper_spot_hsv = np.array([180, 255, 70]) # Max Hue, Saturation, a relatively low Value for darkness

    # Create a mask for the spots
    spot_mask = cv2.inRange(hsv, lower_spot_hsv, upper_spot_hsv)

    # Refine the mask with morphological operations
    # Dilation to make spots slightly larger for inpainting (ensures edges are covered)
    kernel = np.ones((3,3), np.uint8)
    spot_mask = cv2.dilate(spot_mask, kernel, iterations=1)
    # Erosion to remove small noise, if any (uncomment if needed)
    # spot_mask = cv2.erode(spot_mask, kernel, iterations=1)

    # --- Step 2: Use inpainting to fill the masked areas ---
    # Inpainting reconstructs the masked area from the surrounding pixels.
    # For natural images, INPAINT_TELEA often gives good results.
    inpainted_img = cv2.inpaint(img, spot_mask, 3, cv2.INPAINT_TELEA)

    # Convert inpainted image to RGB for display
    inpainted_img_rgb = cv2.cvtColor(inpainted_img, cv2.COLOR_BGR2RGB)

    # --- Display Results ---
    plt.figure(figsize=(15, 7))

    plt.subplot(1, 3, 1)
    plt.imshow(img_rgb)
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(spot_mask, cmap='gray')
    plt.title('Identified Spots Mask')
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(inpainted_img_rgb)
    plt.title('Spots Removed (Inpainted)')
    plt.axis('off')

    plt.show()

```

```
    # --- Optional: Save the Result ---
    # cv2.imwrite('leopard_spots_removed.jpg', inpainted_img)
    # print("Result saved as 'leopard_spots_removed.jpg'")


if __name__ == '__main__':
    # --- Example Usage ---
    # Save your provided image as 'image.jpg' in the same directory
       as this script,
    # or provide the full path to your image file.
    remove_leopard_spots('image.jpg')
```

Listing 2: Code for Leopard Spot Removal (‘b.py‘)

Results

Figure 2: Input image for the leopard spot removal task.

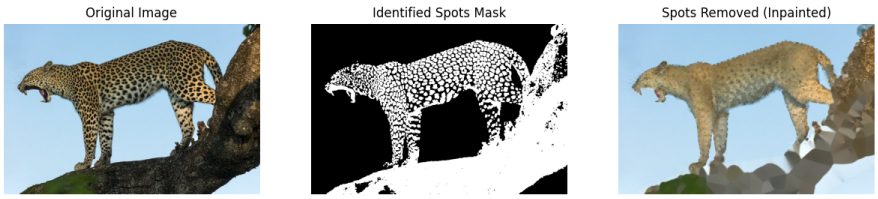

Figure 3: Leopard spot removal process: Original input image, identified spots mask, and inpainted image.