

# EE604 - Assignment 1

Lohit P Talavar

September 26, 2025

## Q1: Text-to-Image Generation

### Code

```
1 import numpy as np
2 import cv2
3 from matplotlib import pyplot as plt
4
5 # --- Part A: Text-to-Image Generation ---
6
7 # A simple bitmap font dictionary. Each character is represented by
8 # an 8x6 matrix.
9 # 1 represents a pixel to be drawn (black), 0 represents background
10 # (white).
11 BITMAP_FONT = {
12     'A': np.array([
13         [0, 1, 1, 1, 1, 0],
14         [1, 0, 0, 0, 0, 1],
15         [1, 0, 0, 0, 0, 1],
16         [1, 1, 1, 1, 1, 1],
17         [1, 0, 0, 0, 0, 1],
18         [1, 0, 0, 0, 0, 1],
19         [1, 0, 0, 0, 0, 1],
20         [0, 0, 0, 0, 0, 0]
21     ]),
22     'B': np.array([
23         [1, 1, 1, 1, 1, 0],
24         [1, 0, 0, 0, 0, 1],
25         [1, 0, 0, 0, 0, 1],
26         [1, 1, 1, 1, 1, 0],
27         [1, 0, 0, 0, 0, 1],
28         [1, 0, 0, 0, 0, 1],
29         [1, 1, 1, 1, 1, 0],
30         [0, 0, 0, 0, 0, 0]
31     ]),
32 }
```

```

30     'C': np.array([
31         [0, 1, 1, 1, 1, 1],
32         [1, 0, 0, 0, 0, 0],
33         [1, 0, 0, 0, 0, 0],
34         [1, 0, 0, 0, 0, 0],
35         [1, 0, 0, 0, 0, 0],
36         [1, 0, 0, 0, 0, 0],
37         [0, 1, 1, 1, 1, 1],
38         [0, 0, 0, 0, 0, 0]
39     ]),
40     'D': np.array([
41         [1, 1, 1, 1, 1, 0],
42         [1, 0, 0, 0, 0, 1],
43         [1, 0, 0, 0, 0, 1],
44         [1, 0, 0, 0, 0, 1],
45         [1, 0, 0, 0, 0, 1],
46         [1, 0, 0, 0, 0, 1],
47         [1, 1, 1, 1, 1, 0],
48         [0, 0, 0, 0, 0, 0]
49     ]),
50     'E': np.array([
51         [1, 1, 1, 1, 1, 1],
52         [1, 0, 0, 0, 0, 0],
53         [1, 0, 0, 0, 0, 0],
54         [1, 1, 1, 1, 1, 0],
55         [1, 0, 0, 0, 0, 0],
56         [1, 0, 0, 0, 0, 0],
57         [1, 1, 1, 1, 1, 1],
58         [0, 0, 0, 0, 0, 0]
59     ]),
60     'F': np.array([
61         [1, 1, 1, 1, 1, 1],
62         [1, 0, 0, 0, 0, 0],
63         [1, 0, 0, 0, 0, 0],
64         [1, 1, 1, 1, 1, 0],
65         [1, 0, 0, 0, 0, 0],
66         [1, 0, 0, 0, 0, 0],
67         [1, 0, 0, 0, 0, 0],
68         [0, 0, 0, 0, 0, 0]
69     ]),
70     'G': np.array([
71         [0, 1, 1, 1, 1, 1],
72         [1, 0, 0, 0, 0, 0],
73         [1, 0, 0, 0, 0, 0],
74         [1, 0, 0, 1, 1, 1],
75         [1, 0, 0, 0, 0, 1],
76         [1, 0, 0, 0, 0, 1],

```

```

77         [0, 1, 1, 1, 1, 0],
78         [0, 0, 0, 0, 0, 0]
79     ]),
80     'H': np.array([
81         [1, 0, 0, 0, 0, 1],
82         [1, 0, 0, 0, 0, 1],
83         [1, 0, 0, 0, 0, 1],
84         [1, 1, 1, 1, 1, 1],
85         [1, 0, 0, 0, 0, 1],
86         [1, 0, 0, 0, 0, 1],
87         [1, 0, 0, 0, 0, 1],
88         [0, 0, 0, 0, 0, 0]
89     ]),
90     'I': np.array([
91         [1, 1, 1, 1, 1, 1],
92         [0, 0, 1, 1, 0, 0],
93         [0, 0, 1, 1, 0, 0],
94         [0, 0, 1, 1, 0, 0],
95         [0, 0, 1, 1, 0, 0],
96         [0, 0, 1, 1, 0, 0],
97         [1, 1, 1, 1, 1, 1],
98         [0, 0, 0, 0, 0, 0]
99     ]),
100    'J': np.array([
101        [0, 0, 0, 1, 1, 1],
102        [0, 0, 0, 0, 1, 0],
103        [0, 0, 0, 0, 1, 0],
104        [0, 0, 0, 0, 1, 0],
105        [1, 0, 0, 0, 1, 0],
106        [1, 0, 0, 0, 1, 0],
107        [0, 1, 1, 1, 0, 0],
108        [0, 0, 0, 0, 0, 0]
109    ]),
110    'K': np.array([
111        [1, 0, 0, 0, 1, 0],
112        [1, 0, 0, 1, 0, 0],
113        [1, 0, 1, 0, 0, 0],
114        [1, 1, 0, 0, 0, 0],
115        [1, 0, 1, 0, 0, 0],
116        [1, 0, 0, 1, 0, 0],
117        [1, 0, 0, 0, 1, 0],
118        [0, 0, 0, 0, 0, 0]
119    ]),
120    'L': np.array([
121        [1, 0, 0, 0, 0, 0],
122        [1, 0, 0, 0, 0, 0],
123        [1, 0, 0, 0, 0, 0],

```

```

124         [1, 0, 0, 0, 0, 0],
125         [1, 0, 0, 0, 0, 0],
126         [1, 0, 0, 0, 0, 0],
127         [1, 1, 1, 1, 1, 1],
128         [0, 0, 0, 0, 0, 0]
129     ]),
130     'M': np.array([
131         [1, 0, 0, 0, 0, 1],
132         [1, 1, 0, 0, 1, 1],
133         [1, 0, 1, 1, 0, 1],
134         [1, 0, 0, 0, 0, 1],
135         [1, 0, 0, 0, 0, 1],
136         [1, 0, 0, 0, 0, 1],
137         [1, 0, 0, 0, 0, 1],
138         [0, 0, 0, 0, 0, 0]
139     ]),
140     'N': np.array([
141         [1, 0, 0, 0, 0, 1],
142         [1, 1, 0, 0, 0, 1],
143         [1, 0, 1, 0, 0, 1],
144         [1, 0, 0, 1, 0, 1],
145         [1, 0, 0, 0, 1, 1],
146         [1, 0, 0, 0, 0, 1],
147         [1, 0, 0, 0, 0, 1],
148         [0, 0, 0, 0, 0, 0]
149     ]),
150     'O': np.array([
151         [0, 1, 1, 1, 1, 0],
152         [1, 0, 0, 0, 0, 1],
153         [1, 0, 0, 0, 0, 1],
154         [1, 0, 0, 0, 0, 1],
155         [1, 0, 0, 0, 0, 1],
156         [1, 0, 0, 0, 0, 1],
157         [0, 1, 1, 1, 1, 0],
158         [0, 0, 0, 0, 0, 0]
159     ]),
160     'P': np.array([
161         [1, 1, 1, 1, 1, 0],
162         [1, 0, 0, 0, 0, 1],
163         [1, 0, 0, 0, 0, 1],
164         [1, 1, 1, 1, 1, 0],
165         [1, 0, 0, 0, 0, 0],
166         [1, 0, 0, 0, 0, 0],
167         [1, 0, 0, 0, 0, 0],
168         [0, 0, 0, 0, 0, 0]
169     ]),
170     'Q': np.array([

```

```

171         [0, 1, 1, 1, 1, 0],
172         [1, 0, 0, 0, 0, 1],
173         [1, 0, 0, 0, 0, 1],
174         [1, 0, 0, 0, 0, 1],
175         [1, 0, 0, 1, 0, 1],
176         [1, 0, 0, 0, 1, 0],
177         [0, 1, 1, 1, 0, 1],
178         [0, 0, 0, 0, 0, 0]
179     ]),
180     'R': np.array([
181         [1, 1, 1, 1, 1, 0],
182         [1, 0, 0, 0, 0, 1],
183         [1, 0, 0, 0, 0, 1],
184         [1, 1, 1, 1, 1, 0],
185         [1, 0, 0, 1, 0, 0],
186         [1, 0, 0, 0, 1, 0],
187         [1, 0, 0, 0, 0, 1],
188         [0, 0, 0, 0, 0, 0]
189     ]),
190     'S': np.array([
191         [0, 1, 1, 1, 1, 1],
192         [1, 0, 0, 0, 0, 0],
193         [1, 0, 0, 0, 0, 0],
194         [0, 1, 1, 1, 1, 0],
195         [0, 0, 0, 0, 0, 1],
196         [0, 0, 0, 0, 0, 1],
197         [1, 1, 1, 1, 1, 0],
198         [0, 0, 0, 0, 0, 0]
199     ]),
200     'T': np.array([
201         [1, 1, 1, 1, 1, 1],
202         [0, 0, 1, 1, 0, 0],
203         [0, 0, 1, 1, 0, 0],
204         [0, 0, 1, 1, 0, 0],
205         [0, 0, 1, 1, 0, 0],
206         [0, 0, 1, 1, 0, 0],
207         [0, 0, 1, 1, 0, 0],
208         [0, 0, 0, 0, 0, 0]
209     ]),
210     'U': np.array([
211         [1, 0, 0, 0, 0, 1],
212         [1, 0, 0, 0, 0, 1],
213         [1, 0, 0, 0, 0, 1],
214         [1, 0, 0, 0, 0, 1],
215         [1, 0, 0, 0, 0, 1],
216         [1, 0, 0, 0, 0, 1],
217         [0, 1, 1, 1, 1, 0],

```

```

218         [0, 0, 0, 0, 0, 0]
219     ]),
220     'V': np.array([
221         [1, 0, 0, 0, 0, 1],
222         [1, 0, 0, 0, 0, 1],
223         [1, 0, 0, 0, 0, 1],
224         [0, 1, 0, 0, 1, 0],
225         [0, 1, 0, 0, 1, 0],
226         [0, 0, 1, 1, 0, 0],
227         [0, 0, 1, 1, 0, 0],
228         [0, 0, 0, 0, 0, 0]
229     ]),
230     'W': np.array([
231         [1, 0, 0, 0, 0, 1],
232         [1, 0, 0, 0, 0, 1],
233         [1, 0, 0, 0, 0, 1],
234         [1, 0, 0, 0, 0, 1],
235         [1, 0, 1, 1, 0, 1],
236         [1, 1, 0, 0, 1, 1],
237         [1, 0, 0, 0, 0, 1],
238         [0, 0, 0, 0, 0, 0]
239     ]),
240     'X': np.array([
241         [1, 0, 0, 0, 0, 1],
242         [0, 1, 0, 0, 1, 0],
243         [0, 0, 1, 1, 0, 0],
244         [0, 0, 1, 1, 0, 0],
245         [0, 0, 1, 1, 0, 0],
246         [0, 1, 0, 0, 1, 0],
247         [1, 0, 0, 0, 0, 1],
248         [0, 0, 0, 0, 0, 0]
249     ]),
250     'Y': np.array([
251         [1, 0, 0, 0, 0, 1],
252         [0, 1, 0, 0, 1, 0],
253         [0, 0, 1, 1, 0, 0],
254         [0, 0, 1, 1, 0, 0],
255         [0, 0, 1, 1, 0, 0],
256         [0, 0, 1, 1, 0, 0],
257         [0, 0, 1, 1, 0, 0],
258         [0, 0, 0, 0, 0, 0]
259     ]),
260     'Z': np.array([
261         [1, 1, 1, 1, 1, 1],
262         [0, 0, 0, 0, 1, 0],
263         [0, 0, 0, 1, 0, 0],
264         [0, 0, 1, 0, 0, 0],

```

```

265         [0, 1, 0, 0, 0, 0],
266         [1, 0, 0, 0, 0, 0],
267         [1, 1, 1, 1, 1, 1],
268         [0, 0, 0, 0, 0, 0]
269     ]),
270     ' ': np.zeros((8, 6), dtype=int),
271     '.': np.array([
272         [0, 0, 0, 0, 0, 0],
273         [0, 0, 0, 0, 0, 0],
274         [0, 0, 0, 0, 0, 0],
275         [0, 0, 0, 0, 0, 0],
276         [0, 0, 0, 0, 0, 0],
277         [0, 0, 1, 1, 0, 0],
278         [0, 0, 1, 1, 0, 0],
279         [0, 0, 0, 0, 0, 0]
280     ]),
281     '!': np.array([
282         [0, 0, 1, 1, 0, 0],
283         [0, 0, 1, 1, 0, 0],
284         [0, 0, 1, 1, 0, 0],
285         [0, 0, 1, 1, 0, 0],
286         [0, 0, 1, 1, 0, 0],
287         [0, 0, 0, 0, 0, 0],
288         [0, 0, 1, 1, 0, 0],
289         [0, 0, 0, 0, 0, 0]
290     ]),
291     '?': np.array([
292         [0, 1, 1, 1, 1, 0],
293         [1, 0, 0, 0, 0, 1],
294         [0, 0, 0, 0, 1, 0],
295         [0, 0, 0, 1, 0, 0],
296         [0, 0, 1, 0, 0, 0],
297         [0, 0, 0, 0, 0, 0],
298         [0, 0, 1, 1, 0, 0],
299         [0, 0, 0, 0, 0, 0]
300     ])
301 }
302
303 def text_to_image(text_input, char_height=8, char_width=6, padding
304 =10):
305     """
306     Renders a string of text to an image using a custom bitmap font.
307
308     Args:
309         text_input (str): The string to render.
310         char_height (int): The height of a single character in
311             pixels.

```

```

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

312
313     Returns:
314         numpy.ndarray: The rendered image as a NumPy array.
315     """
316     # Filter out any characters not in our font map
317     lines = text_input.upper().split('\n')
318     valid_lines = []
319     for line in lines:
320         valid_line = "".join([char for char in line if char in
            BITMAP_FONT])
321         valid_lines.append(valid_line)
322
323     # Calculate image dimensions
324     num_lines = len(valid_lines)
325     max_line_length = 0
326     if valid_lines:
327         max_line_length = max(len(line) for line in valid_lines)
328
329     img_height = num_lines * char_height + 2 * padding
330     img_width = max_line_length * char_width + 2 * padding
331
332     # Create a white canvas (255 is white in grayscale)
333     canvas = np.full((img_height, img_width), 255, dtype=np.uint8)
334
335     # Render each character pixel by pixel
336     y_cursor = padding
337     for line in valid_lines:
338         x_cursor = padding
339         for char in line:
340             char_map = BITMAP_FONT[char]
341
342             # Define the region on the canvas to draw the character
343             y_start, y_end = y_cursor, y_cursor + char_height
344             x_start, x_end = x_cursor, x_cursor + char_width
345
346             # Iterate through the bitmap and draw pixels
347             for row_idx, row in enumerate(char_map):
348                 for col_idx, pixel_val in enumerate(row):
349                     if pixel_val == 1:
350                         # Set pixel to black (0)
351                         canvas[y_start + row_idx, x_start + col_idx]
                            = 0
352
353                 x_cursor += char_width

```



```

354         y_cursor += char_height
355
356     return canvas
357
358
359 if __name__ == '__main__':
360     # --- Example Usage ---
361     input_text = (
362         "HELLO WORLD!\n"
363         "THIS IS A TEST OF THE\n"
364         "TEXT TO IMAGE SYSTEM.\n"
365         "ABCDEFGHIJKLMNOPQRSTUVWXYZ\n"
366         "1234567890?.\n"
367         "THIS IS ME LOHIT P TALAVAR\n"
368         "THIS IS IMAGE PROCESSING"
369     )
370
371     # Generate the image
372     rendered_image = text_to_image(input_text)
373
374     # --- Display the Result ---
375     plt.figure(figsize=(10, 8))
376     plt.imshow(rendered_image, cmap='gray')
377     plt.title('Rendered Text')
378     plt.axis('off') # Hide axes for a cleaner look
379     plt.tight_layout()
380     plt.show()
381
382     # cv2.imwrite('rendered_text.png', rendered_image)

```

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

## Results

### Q2: Leopard Spot Removal

#### Code

```

1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 def remove_leopard_spots(image_path):
6     """
7     Removes 'leopard spots' (dark regions) from an image using HSV
        color segmentation

```

HELLO WORLD!  
 THIS IS A TEST OF THE  
 TEXT TO IMAGE SYSTEM.  
 ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 ?  
 THIS IS ME LOHIT P TALAVAR  
 THIS IS IMAGE PROCESSING

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

```

8   and inpainting techniques.
9
10  Args:
11      image_path (str): The path to the input image file.
12
13  Returns:
14      None: Displays the original, mask, and inpainted images.
15  """
16  # --- Image Loading and Preprocessing ---
17  img = cv2.imread(image_path)
18  if img is None:
19      print(f"Error: Could not load image from {image_path}")
20      return
21
22  # Convert to RGB for matplotlib display later
23  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
24
25  # Convert to HSV color space for better color segmentation
26  hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
27
28  # --- Step 1: Create a mask for the spots ---
29  # This is the trickiest part and will likely require fine-tuning
30  # We're looking for dark brown/black spots.
31  # Define a range for dark colors in HSV. These values are
32  # approximate
33  # and might need adjustment based on the specific image.
34  # Lower bound for dark brown/black (adjust these!)

```

```

35     lower_spot_hsv = np.array([0, 0, 0]) # Hue, Saturation, Value (V
        = brightness)
36     upper_spot_hsv = np.array([180, 255, 70]) # Max Hue, Saturation,
        a relatively low Value for darkness
37
38     # Create a mask for the spots
39     spot_mask = cv2.inRange(hsv, lower_spot_hsv, upper_spot_hsv)
40
41     # Refine the mask with morphological operations
42     # Dilation to make spots slightly larger for inpainting (ensures
        edges are covered)
43     kernel = np.ones((3,3), np.uint8)
44     spot_mask = cv2.dilate(spot_mask, kernel, iterations=1)
45     # Erosion to remove small noise, if any (uncomment if needed)
46     # spot_mask = cv2.erode(spot_mask, kernel, iterations=1)
47
48     # --- Step 2: Use inpainting to fill the masked areas ---
49     # Inpainting reconstructs the masked area from the surrounding
        pixels.
50     # For natural images, INPAINT_TELEA often gives good results.
51     inpainted_img = cv2.inpaint(img, spot_mask, 3, cv2.INPAINT_TELEA
        )
52
53     # Convert inpainted image to RGB for display
54     inpainted_img_rgb = cv2.cvtColor(inpainted_img, cv2.
        COLOR_BGR2RGB)
55
56     # --- Display Results ---
57     plt.figure(figsize=(15, 7))
58
59     plt.subplot(1, 3, 1)
60     plt.imshow(img_rgb)
61     plt.title('Original Image')
62     plt.axis('off')
63
64     plt.subplot(1, 3, 2)
65     plt.imshow(spot_mask, cmap='gray')
66     plt.title('Identified Spots Mask')
67     plt.axis('off')
68
69     plt.subplot(1, 3, 3)
70     plt.imshow(inpainted_img_rgb)
71     plt.title('Spots Removed (Inpainted)')
72     plt.axis('off')
73
74     plt.show()
75

```

```

76     # --- Optional: Save the Result ---
77     # cv2.imwrite('leopard_spots_removed.jpg', inpainted_img)
78     # print("Result saved as 'leopard_spots_removed.jpg'")
79
80
81 if __name__ == '__main__':
82     # --- Example Usage ---
83     # Save your provided image as 'image.jpg' in the same directory
      as this script,
84     # or provide the full path to your image file.
85     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.

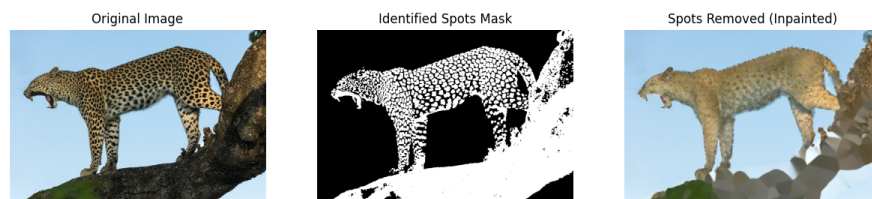


Figure 3: Leopard spot removal process: Original input image, identified spots mask, and inpainted image.