1 Problem 1

Print out and colour the matrix pictureMe.png. A fun exercise.

114		132	102 146	1 19 1 19	77 127	65 95	78 147	43 114	94 94	/84 /81	65 103	76 64	84	99 102	Fabi	1677	173		90		102	<u>an</u>	83	/58 91	30	41	23 /	ſ	00 08
122 115	86 100	105 92	137 99	59 32	67 79	118 147	124 101	130 148	32 113	84 113	125 119	82		/98	in.	a ca		(60) 97	fills this	12.	1.000	79 70	27	46 29	66	82	63	36	52 11
140 104	104 111	59 105	88 98	44 52	31 66	57 53	114 139	120	146	124 127	96 88	121 108		104 84	476 126	93		71 106	67 111	78 78	94	102	92 52	69 44	89 122	45	49 56		76
78	152 110	48 136	84 76	55 74	39 45	74 45	88 56	129 61	146 92	126 126		113 83	101 75	64 104	98 //39	102	65 79	77 87	63 58	88 102	122 88	63 89	38 70	44 94	72 72	106	50 126	40	26
64 70	91 40	53 56	68 59	81 58	65 57	62 65	38 47	51 32	44 61	95 26	97 121	139 123	93 44	96 64	141 101	139 37	145 47	111	141 102	83 82	70 84	66 75	55 79	19 66	49 47	37 33	32 28	19 38	52 71
41	55	47	47	44	75	55	54	28	30	71 70	81 70	134	99 54	36 116	64 59	63 88	62 62	50	63 78	68 62	29 60	27 59	79 52 57 27 18	23 45	23 72	37 36	12 30	29 15	65 43
70	51	65 60	44	70	53	62	65	54	48	82	45	60 27	87	87 54	48	39	40	80 70	76	66 23	32 40	54 33	27 18	18 21	19 24	24 22	19 21	20 29	30 17
75	51	54	51	77	33	47	71	57	54	54 57	52	37	24	61 70	53	57	46	85 63	67 82	51	54 46	46 41	50 49	34 76	59 42	48 25	22 55	12 31	11 26
52	27	115	58	39	68 64	40	67	38	42 59	45	39	50	44	122	20	36	80	112	112	39	43	39 26	51 22	56 37	37 42	34 42	45 48	41 62	19 30
119	39	106	63	102	32	43	62	55	50	40	44	45	37	106	136	55	31	34	31	29	21	21	19 21	27 99	65 54	60 33	59 31	114 51	67 66
75 71 52 109 119 27 103 157 115 50 161	76	18	64 73	153 157	5/ 71	58 76	46	74 74	45 73	59 58	48 59	35	34	31	100	92 59		un y acyt	10P	1060 1060 7784			149	104	50	30	40	91 197	37
115	16. 145	56 91	59 90	1 16	109 169	70 48	48 81	66 69	68 71	57 66	59 61	32 47	34 74	61 39	59 84	<u>75</u> 104	127	76	i Pa	137	108	57	72	48	137	126	110	94	40
Contraction of the	DATES IN	4-7-5		The set	A street	78 71	75 86	58 46	52 57	68 68								*****							90 81			31	50
149		174	139				<u>55</u> 89	57 82	51 48	56 55	65 50	65	74	44	37 71	27	53	80	47	41	103	22	51 63		70		31 70	25 69	20 77
	157	107	85	128			130	54 69	64		and the second second	60	79	57		47	56	70	65	93	51	38	44 32				41 36	37	39 39
149 117 28	127 76	94 48	54 74	110 96	Ē.	1.5		67	53	73	55	42	64	52	85	76	38	68	81	77	83	40 84	45 48	21 56	56 31	53 72	38 72 97	33 69	87 66
28 119	5 105	14 103	33 1867	141					63 66	38 89	35 26	58 45	50 (1)	81 80	63	59	66	98	26	같은 집	i i	76	31	37	56	91	97	99; 9	2
119 139	137			20. 188	169 170		1.6	1		24	41;	31	54		23	94 75	3/	00	42	20	1953		78 67				派星		
55	60	86	155	175		186 186	11-1-1-1			88	49 9	26 23 29	102 47		86 135	57 31	58 56		67 145	30 9	67	1. Cla		N.H.S.	42 23	120 66			
			di.							ust	76	44	32 29			29	54	127 109		29 7	37 47		iv s	1 1 1	5 10	17 23	1.1.1		
							541 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	r - C Ta		575 575		22	30	89 43		16	44		a en	39 14	57 15	Ma sis		n i Ro	9 15	31 53			and the second
1212									in time			135	13	27		24	45 44	131 Q/	109 50	47		松济		118 82	31 148	57 90	Service Services		
	i i i							1.0.2		4 4 4	100 A	05	25	17	Haile	24	34		90	103	71	157 54		63 100	68			Cl. Cl.	
東京	1133 		iki ili.							95	58	69	41	61	79	49	43	95	76	55	83			Ţ,					
			-	102	88	76	95	89 89	87 100	99 89	101	10 86	104	31	79 88 66	89		33. - 19						6.5					
91 91 103	1.74	81	97	86,		91	95	88	1091	95	105	11:	6/			A	- 44												
103 85	97 87	96 85	93 96	104 90	97 74	103 99	102 102	84 134	72. 167	101		Katta		dinte	in l	i k Vini	aut	al al al al	<u>i derte</u>	0.55		in in	il	10 10 10 10 10 10		mture			and and

Figure 1: Coloured pictureMe.png

I used 4 different grade graphite pencils to colour the image. Included uncoloured squares (white) this made me split up the interval [0, 255] into 5 linearly spaced sections, each corresponding to a white or the 4 shades of graphite.

I realised that most of the numbers were low, which correspond to darker pixels. Since I wanted to colour less, I corresponded high values with black and low values with white, and then inverted the final image (as you can see in Figure 1).

2 Problem 2

 (a) Read and show the image, and then capture the screenshot of the window showing the image. Save the captured screenshot as [your_unityid]_screenshot.png (e.g., twu19_screenshot.png)

Using the Python Image Library (PIL), I read in the image from its directory and displayed it using matplotlib.pyplot.imshow(). I used PIL instead of OpenCV or matplotlib to read in the file because OpenCV was not in RGB order and matplotlib was not in uint8 format ($\in [0, 255]$), whereas PIL did both without me having to reorder anything

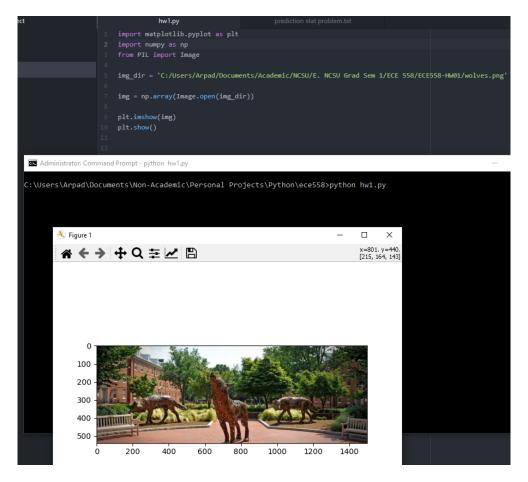


Figure 2: Screenshot of image being displayed

(b) i. Find the digit signature of your unity id. First, convert each non-digit character in your unity id to ASCII value (e.g., 'twu19' corresponds to 116, 119, 117, 19).

My Unity ID is aavoros, therefore it corresponds with ASCII numbers

aavoros = [97, 97, 118, 111, 114, 111, 115]

Regardless of having no number within my Unity ID, the program will still properly convert all characters to this 'digital signature' as given in the problem statement.

ii. Second, count the number of occurrence of each of the digit number in each color channel of the image (the number of occurrence could be zero, but definitely less than the total number of pixels).

The number of instances is given in Table 1 below. There is a redundancy in count for the characters **a** and **o** due to being duplicated.

Character	ASCII	# in R	# in G	# in B
a	97	3063	3472	2807
v	118	2506	3035	1910
0	111	2662	3149	2202
r	114	2639	3080	2057
s	115	2534	3170	2096

Table 1: Pixel counts for unique aavoros ASCII characters in wolves.png

iii. Third, change to 255 the pixel values of the 5 by 5 sub-image (if valid) centered at each occurrence and then show the result image. After you changed all the occurrence, save the result image as [your_unityid]_signature.png (e.g., twu19_signature.png)

Iterating through each channel, row, column, and ASCII value to change the value of a 5×5 box to 255 is susceptible to overwriting potential pixels which equal our ASCII values. Meaning, if there pixels with our ASCII values within 2-4 pixels of one another and in the same channel, whichever ASCII value sis checked first will overwrite the other ASCII value to 255. This results in not every ASCII pixel their 5×5 square. A way around this would be to create a map of every pixel position where an ASCII value exists, and then proceed to colour a copy of the image w.r.t. the map, rather than colouring the image w.r.t. itself. However, I have been informed that w.r.t. grading, the order does not matter and overwriting is allowed. Therefore I have simply kept the old version of my program. The order of checking pixel values to colour pixel sub-images hierarchically follows the following

1. Loop through an ordered list of each ASCII value of all unique characters

- 2. Loop through each RGB channel
- 3. Loop through each row
- 4. Loop through each column
- 5. Check if coloured & if ASCII value equals pixel value. If so, colour the 5×5 box to 255

Here is the final digital signature for **aavoros**

Figure 3: 'Digital Signature' for my Unity ID: aavoros

Below is the listing for hw1.py. To summarize:

- PIL is used to read in the wolves.png image and is converted to a numpy array
- A character array holds a UnityID. This variable is fed through an algorithm which extracts the ASCII values for characters, and the integers remain
- All instances of the ASCII values within the wolves.png using binary operations
- The image in copied to output the Digital Signature. A mask of already altered values is created to check pixels faster. Each pixel in the (row, column, channel) position is compared to each ASCII value. If they equal, then the 5×5 box is recoloured and the mask is appropriately updated.

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
# function to find bounds while painting. considers boundary conditions
# can change boundary size, but default is 5 (as said in hw1)
def get_bounds(r, c, num_r, num_c, box_l = 5):
    # O based indexing, this is max row/col value
   num_r -= 1
   num_c -= 1
    # init min/max values
   r_floor = 0
    c_floor = 0
   r_ceil = num_r
    c_ceil = num_c
    if not box_1 % 2:
        # want box length to be odd, so the pixel value at r/c is perfectly in
                                                  the
        # center of the square
        raise ValueError('box length must be odd: box_l = %d' % (box_l))
    else:
        delta = int((box_1 - 1) / 2)
        if r + delta < r_ceil:</pre>
            r_{ceil} = r + delta
        if r - delta > r_floor:
```

```
r_floor = r - delta
        if c + delta < c_ceil:</pre>
           c_ceil = c + delta
        if c - delta > c_floor:
            c floor = c - delta
    return np.array(range(r_floor, r_ceil)), np.array(range(c_floor, c_ceil))
# get the wolf image
folder_dir = 'C:/Users/Arpad/Documents/Academic/NCSU/E. NCSU Grad Sem 1/ECE 558/
                                         HW01/'
img_dir = folder_dir + 'wolves.png'
# display the wolf image
img = np.array(Image.open(img_dir))
plt.imshow(img)
plt.show()
# image characteristics
img_shape = np.shape(img)
num_r = img_shape[0]
num_c = img_shape[1]
# unity id, used in calculating the 'digital signature'
my_uid = 'aavoros'
# get the ASCII numbers of the unity id
ascii_uid = []
id_len = len(my_uid)
id_idx = 0
while id_idx < id_len:</pre>
    chr = my_uid[id_idx]
    if chr.isnumeric():
        offset = 1
        while my_uid[id_idx:id_idx + offset].isnumeric() and (id_idx + offset) <</pre>
                                                  = id_len:
            offset += 1
        ascii_uid.append(int(my_uid[id_idx:id_idx + offset - 1]))
        id_idx += offset - 2
    else:
        ascii_uid.append(ord(chr))
    id_idx += 1
ascii_uid = np.array(ascii_uid)
# print unity id with ASCII result
print(my_uid)
print(ascii_uid)
# find the pixel count of each ASCII code
num_red = np.zeros(np.shape(ascii_uid))
num_gre = np.zeros(np.shape(ascii_uid))
num_blu = np.zeros(np.shape(ascii_uid))
chr_idx = 0
for chr in ascii_uid:
    num_red[chr_idx] = int(np.sum(img[:, :, 0] == chr))
    num_gre[chr_idx] = int(np.sum(img[:, :, 1] == chr))
    num_blu[chr_idx] = int(np.sum(img[:, :, 2] == chr))
    chr_idx += 1
# print the counts
print("Number of red instances per ASCII value:")
print(num_red)
print("Number of green instances per ASCII value:")
```

```
print(num_gre)
print("Number of blue instances per ASCII value:")
print(num_blu)
# create a copy of the image to paint the digital signature
ds_img = np.copy(img)
\ensuremath{\texttt{\#}} each channel, RGB
channel = [0, 1, 2]
# create a binary mask to skip already painted pixels
unchecked_mask = np.ones(img_shape, dtype = bool)
# find each pixel, paint the digital signature
for chr in np.unique(ascii_uid):
    for chnl in channel:
         for r in range(num_r):
             for c in range(num_c):
                  if unchecked_mask[r, c, chnl] and ds_img[r, c, chnl] == chr:
                      r_range, c_range = get_bounds(r, c, num_r, num_c)
                      ds_img[r_range[:, None], c_range[None, :], chnl] = 255
unchecked_mask[r_range[:, None], c_range[None, :], chnl] =
                                                                   False
# save the image and display it
Image.fromarray(ds_img).save(folder_dir + my_uid + '_signature.png')
plt.imshow(ds_img)
plt.show()
```