Lab8

April 24, 2022

0.1 Lab 8

0.1.1 Submitted By: Manav Doda

0.1.2 Roll No.: 195057

0.2 Importing Necessary modules

[1]:

```
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

0.3 Objective:

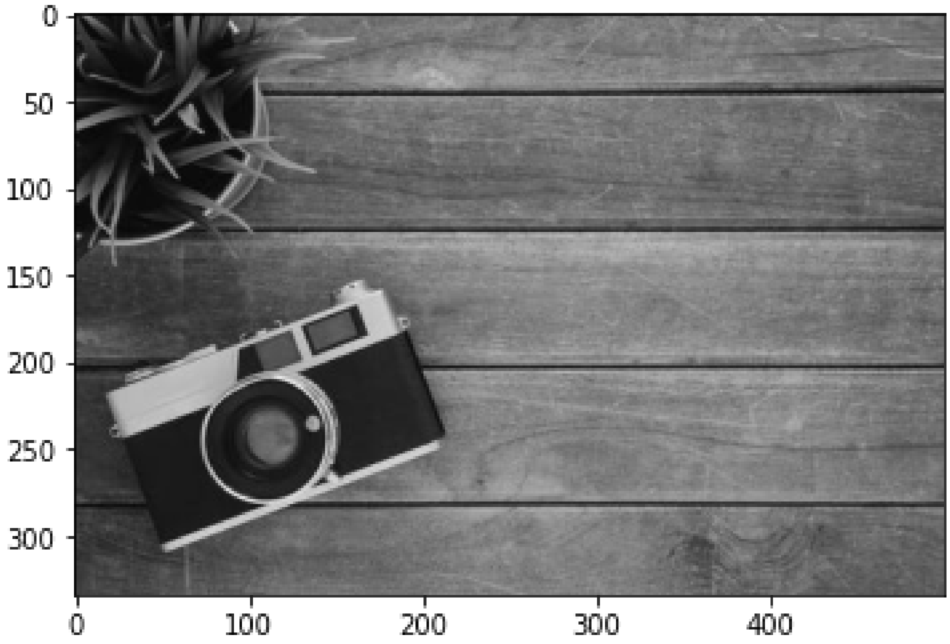
0.3.1 Compressing Image size using Huffman Coding

[2]:

```
image = cv2.imread("testImage.jpeg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_GRAY2RGB))
```

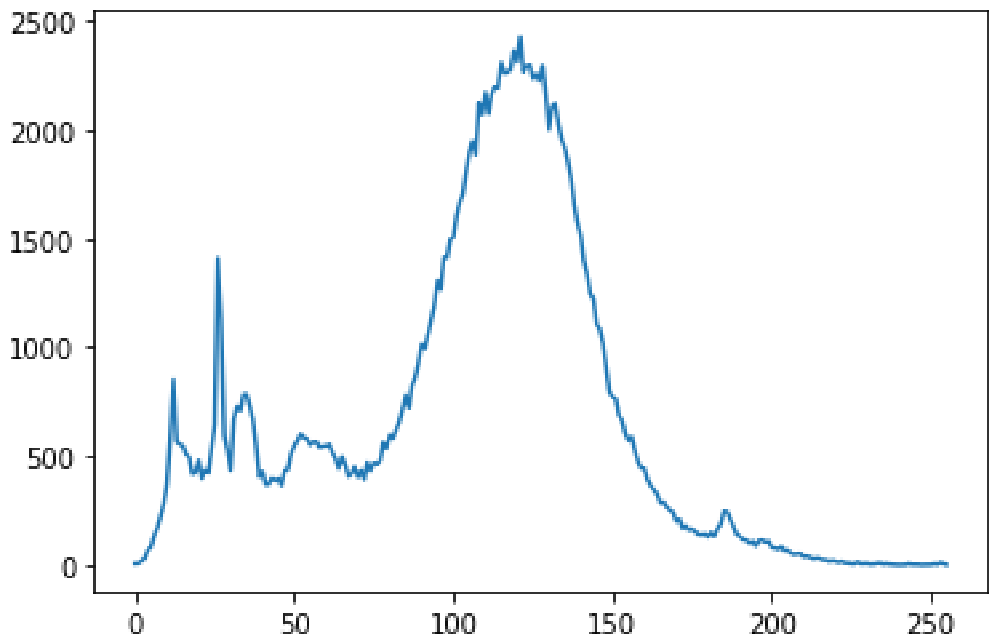
[2]:

```
<matplotlib.image.AxesImage at 0x112f70af0>
```



[3]:
```
hist = []
for i in range(256):
    hist.append(0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        hist[image[i][j]] += 1
plt.plot(range(256), hist)
```

[3]:
```
[<matplotlib.lines.Line2D at 0x1221822e0>]
```



[4]:

```python
class node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''

def printNodes(node, val=''):
    newVal = val + str(node.huff)
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")
```

[5]:

```python
nodes = []
for i in range(len(hist)):
    nodes.append(node(hist[i], i))
while len(nodes) > 1:
    nodes = sorted(nodes, key=lambda x: x.freq)
    left = nodes[0]
    right = nodes[1]
    left.huff = 0
    right.huff = 1
    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
    nodes.remove(left)
    nodes.remove(right)
    nodes.append(newNode)
printNodes(nodes[0])
```

```
65 -> 00000000
233 -> 00000001000000
237 -> 000000010000010
232 -> 000000010000011
218 -> 0000000100001
3 -> 000000010001
215 -> 000000010010
214 -> 000000010011
180 -> 0000000101
168 -> 000000011
90 -> 0000001
133 -> 000001
147 -> 0000100
16 -> 00001010
49 -> 00001011
109 -> 000011
111 -> 000100
62 -> 00010100
29 -> 00010101
204 -> 00010110000
220 -> 0001011000100
222 -> 0001011000101
238 -> 0001011000110000
244 -> 0001011000110001
245 -> 0001011000110010
246 -> 0001011000110011
252 -> 0001011000110100
241 -> 00010110001101010
242 -> 00010110001101011
248 -> 00010110001101100
249 -> 00010110001101101
250 -> 00010110001101110
255 -> 0001011000110111100
240 -> 00010110001101111010
247 -> 00010110001101111011
239 -> 000101100011011111
2 -> 0001011000111
182 -> 0001011001
190 -> 0001011010
6 -> 0001011011
157 -> 00010111
```

```
131 -> 000110
132 -> 000111
108 -> 001000
92 -> 0010010
79 -> 00100110
24 -> 00100111
58 -> 00101000
167 -> 001010010
4 -> 00101001100
205 -> 00101001101
178 -> 0010100111
146 -> 0010101
129 -> 001011
112 -> 001100
110 -> 001101
59 -> 00111000
60 -> 00111001
15 -> 00111010
50 -> 00111011
114 -> 001111
113 -> 010000
38 -> 01000100
55 -> 01000101
145 -> 0100011
127 -> 010010
61 -> 01001100
14 -> 01001101
93 -> 0100111
125 -> 010100
177 -> 0101010000
189 -> 0101010001
9 -> 010101001
78 -> 01010101
27 -> 0101011
126 -> 010110
116 -> 010111
117 -> 011000
122 -> 011001
56 -> 01101000
13 -> 01101001
57 -> 01101010
165 -> 011010110
179 -> 0110101110
202 -> 01101011110
212 -> 011010111110
210 -> 011010111111
118 -> 011011
123 -> 011100
```

```
128 -> 011101
124 -> 011110
51 -> 01111100
155 -> 01111101
54 -> 01111110
81 -> 01111111
115 -> 100000
120 -> 100001
11 -> 10001000
53 -> 10001001
156 -> 10001010
166 -> 100010110
181 -> 1000101110
176 -> 1000101111
119 -> 100011
94 -> 1001000
28 -> 10010010
80 -> 10010011
121 -> 100101
52 -> 10011000
154 -> 10011001
144 -> 1001101
143 -> 1001110
82 -> 10011110
223 -> 10011111000000
224 -> 10011111000001
234 -> 10011111000010
225 -> 100111110000110
226 -> 100111110000111
211 -> 100111110001
201 -> 10011111001
174 -> 1001111101
164 -> 100111111
96 -> 1010000
25 -> 10100010
83 -> 10100011
95 -> 1010010
200 -> 10100110000
5 -> 10100110001
175 -> 1010011001
172 -> 1010011010
183 -> 1010011011
153 -> 10100111
142 -> 1010100
37 -> 10101010
31 -> 10101011
203 -> 10101100000
195 -> 10101100001
```

```
7 -> 1010110001
163 -> 101011001
152 -> 10101101
141 -> 1010111
26 -> 1011000
97 -> 1011001
98 -> 1011010
33 -> 10110110
84 -> 10110111
188 -> 1011100000
173 -> 1011100001
162 -> 101110001
86 -> 10111001
32 -> 10111010
36 -> 10111011
46 -> 101111000
219 -> 1011110010000
228 -> 101111001000100
251 -> 101111001000101
253 -> 10111100100011
217 -> 1011110010010
216 -> 1011110010011
254 -> 101111001010000
0 -> 101111001010001
221 -> 10111100101001
229 -> 101111001010100
236 -> 101111001010101
243 -> 101111001010110
231 -> 1011110010101110
235 -> 1011110010101111
207 -> 101111001011
184 -> 1011110011
41 -> 101111010
42 -> 101111011
99 -> 1011111
100 -> 1100000
140 -> 1100001
10 -> 110001000
44 -> 110001001
151 -> 11000101
150 -> 11000110
85 -> 11000111
34 -> 11001000
35 -> 11001001
139 -> 1100101
72 -> 110011000
161 -> 110011001
149 -> 11001101
```

```
21 -> 110011100
45 -> 110011101
43 -> 110011110
170 -> 1100111110
193 -> 11001111110
208 -> 110011111110
209 -> 110011111111
101 -> 1101000
70 -> 110100100
39 -> 110100101
87 -> 11010011
138 -> 1101010
67 -> 110101100
18 -> 110101101
68 -> 110101110
187 -> 1101011110
171 -> 1101011111
102 -> 1101100
23 -> 110110100
19 -> 110110101
12 -> 11011011
103 -> 1101110
40 -> 110111100
198 -> 11011110100
194 -> 11011110101
199 -> 11011110110
213 -> 1101111011100
227 -> 11011110111010
1 -> 110111101110110
230 -> 110111101110111
206 -> 110111101111
88 -> 11011111
74 -> 111000000
30 -> 111000001
47 -> 111000010
71 -> 111000011
22 -> 111000100
48 -> 111000101
64 -> 111000110
160 -> 111000111
137 -> 1110010
159 -> 111001100
8 -> 1110011010
196 -> 11100110110
192 -> 11100110111
148 -> 11100111
104 -> 1110100
69 -> 111010100
```

```
66 -> 111010101
89 -> 11101011
136 -> 1110110
76 -> 111011100
75 -> 111011101
73 -> 111011110
197 -> 11101111100
191 -> 11101111101
169 -> 1110111111
107 -> 1111000
105 -> 1111001
135 -> 1111010
158 -> 111101100
20 -> 111101101
77 -> 111101110
63 -> 111101111
106 -> 1111100
134 -> 1111101
186 -> 1111110000
185 -> 1111110001
17 -> 111111001
91 -> 11111101
130 -> 1111111
```

[6]:

```
def countBits(node, val=''):
    totalBitsRequired = 0
    # huffman code for current node
    newVal = val + str(node.huff)
    # if node is not an edge node
    # then traverse inside it
    if(node.left):
        totalBitsRequired+=countBits(node.left, newVal)
    if(node.right):
        totalBitsRequired+=countBits(node.right, newVal)

        # if node is edge node then
        # display its huffman code
    if(not node.left and not node.right):
        totalBitsRequired += hist[int(node.symbol)-1] * len(newVal)
    return totalBitsRequired

bitSizeAfterCompression = countBits(nodes[0])
```

[7]:

```
newPixelSize = bitSizeAfterCompression/(image.shape[0]*image.shape[1])
print("Image size before compression: ", image.shape[0]*image.shape[1]*8)
print("Image size after compression:  ", bitSizeAfterCompression)
```

```python
print("Number of bits reduced:        ", image.shape[0]*image.
 ↪shape[1]*8-bitSizeAfterCompression)
print("Average pixel size:            ", newPixelSize)
print("Compression Ratio:             ", float(8/newPixelSize))
```

```
Image size before compression:  1336000
Image size after compression:   1211223
Number of bits reduced:         124777
Average pixel size:             7.252832335329341
Compression Ratio:              1.1030173634417444
```

```
[ ]:
```