dsp-assignment-2

February 10, 2020

1 Assigment 2

by Katon D. R. Wibowo (ID: 51934701)

This created by Jupyter Notebook. To run this code, please visit

https://github.com/katondr/dsp/blob/master/dsp-assignment-2.ipynb

(Github will run the code) Importing required module

[139]:
```
import numpy as np
import heapq
from PIL import Image as Pimage
```

1.1 Part 1: Huffman coding

1.1.1 a. Huffman coding function

[140]:
```
def frequency_table(data):
    freq_table = np.array(np.unique(data, return_counts=True))
    freq_table = np.transpose(freq_table)
    return dict(freq_table)

def codebook_generator(frequency):
    heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        low = heapq.heappop(heap)

        high = heapq.heappop(heap)

        for value in low[1:]:

            value[1] = '0' + value[1]

        for value in high[1:]:

            value[1] = '1' +value[1]
```

```
        heapq.heappush(heap, [low[0] + high[0]] + low[1:] + high[1:])

    return dict(sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p)))

def huffman_encoding(data, codebook):
    code = ''
    for i in data:
        code = code + codebook[i]
    return code
```

1.1.2 b. Huffman decoding function

[141]:
```
def huffman_decoding(encoded_data, codebook):
    data = encoded_data
    codebook = {value:key for key, value in codebook.items()}
    decoded_data = []
    c = 0
    l = 1
    while len(data) != 0:
        if data[c:l] in codebook:
            decoded_data.append(codebook[data[c:l]])
            data = data[l:]
            l = 1
        else:
            l = l + 1
    return decoded_data
```

1.1.3 c. Demonstration

[142]:
```
# sample data & convertion to numpy array
sample = [114, 20, 114, 114, 110, 12, 117, 85, 114, 118, 114, 114, 114, 6, 70,␣
 ↪71, 93, 102, 72, 114, 114, 53, 117, 74, 117, 114, 114, 14, 102, 14, 117, 114]
data = np.array(sample)

# create frequency table
frequency = frequency_table(data)

# generating codebook
codebook = codebook_generator(frequency)

# encoding data
encoded_data = huffman_encoding(data, codebook)
```

```
# decoding data
decoded_data = huffman_decoding(encoded_data, codebook)

# verify function
print('Frequency Table')
print(frequency)
print('')
print('Codebook')
print(codebook)
print('')
print('Encoded Data')
print(encoded_data)
print('')
print('Is data converted back corectly?')
print(decoded_data == sample)
```

```
Frequency Table
{6: 1, 12: 1, 14: 2, 20: 1, 53: 1, 70: 1, 71: 1, 72: 1, 74: 1, 85: 1, 93: 1,
102: 2, 110: 1, 114: 12, 117: 4, 118: 1}

Codebook
{114: '11', 117: '00', 14: '0101', 102: '1010', 6: '01000', 12: '01001', 20:
'01100', 53: '01101', 70: '01110', 71: '01111', 72: '10000', 74: '10001', 85:
'10010', 93: '10011', 110: '10110', 118: '10111'}

Encoded Data
11011001111101100100100100101110111111111010000111001111100111010100001111011010
0100010011110101101001010011

Is data converted back corectly?
True
```

1.2 Part 2: Differential and RL coding

1.2.1 a. Differential coding:

1.2.2 encoding

[143]:
```
def diff_encoding(data):
    encoded = []
    encoded.append(data[0])
    while len(data) != 1:
        if data[1] - encoded[0] > 0:
            encoded.append(257)
            encoded.append(data[1] - encoded[0])
        else:
```

```
            encoded.append(258)
            encoded.append(encoded[0] - data[1])
        data = data[1:]
    return encoded
```

1.2.3 decoding

[144]:
```
def diff_decoding(code):
    decoded = []
    data = code
    while len(data) != 0:
        if data[0] == 257:
            decoded.append(code[0] + data[1])
            data = data[2:]
        elif data[0] == 258:
            decoded.append(code[0] - data[1])
            data = data[2:]
        else:
            decoded.append(data[0])
            data = data[1:]
    return data
```

1.2.4 b. Run-length coding:

1.2.5 encoding

[145]:
```
def rl_encoding(data):
    rl_encoded = []
    while len(data) != 2:
        c = 0
        l = 1
        r = 1
        while data[c] == data[l]:
            l = l + 1
            r = r + 1
        if r > 3:
            rl_encoded.append(256)
            rl_encoded.append(data[c])
            rl_encoded.append(r)
        elif r <= 3:
            for i in data[c:l]:
                rl_encoded.append(i)
        data = data[l:]
    for i in data:
```

```
        rl_encoded.append(i)
    return rl_encoded
```

1.2.6 decoding

[146]:
```
def rl_decoding(data):
    decoded = []
    while len(data) != 0:
        if data[0] == 256:
            for i in range(0, data[2]):
                decoded.append(data[1])
        else:
            decoded.append(data[0])
        data = data[1:]
    return decoded
```

1.2.7 c. Demonstration

1.2.8 Differential encoding + Huffman encoding

[147]:
```
sample = [114, 20, 114, 114, 110, 12, 117, 85, 114, 118, 114, 114, 114, 6, 70,␣
↪71, 93, 102, 72, 114, 114, 53, 117, 74, 117, 114, 114, 14, 102, 14, 117, 114]
data = np.array(sample)

# differential encoding
data_211 = diff_encoding(data)

# huffman encoding
frequency_table_211 = frequency_table(data_211)
codebook_21 = codebook_generator(frequency_table_211)
data_212 = huffman_encoding(data_211, codebook_21)
```

1.2.9 RLE + Huffman encoding

[148]:
```
# run-length encoding
data_221 = rl_encoding(data)

# huffmann (encoding
frequency_221 = frequency_table(data_221)
codebook_22 = codebook_generator(frequency_221)
data_222 = huffman_encoding(data_221, codebook_22)
data_222
```

[148]:
```
'1101100111111011001001001001011101111111110100001110011111001110101000011110110100100010011111010110100101001 1'
```

1.2.10 Diffrential encoding + RLE + Huffman encoding

[149]:
```
# differential encoding
data_231 = diff_encoding(data)

# run-length encoding
data_232 = rl_encoding(data_231)

# huffman encoding
frequency_23 = frequency_table(data_232)
codebook_23 = codebook_generator(frequency_23)
data_233 = huffman_encoding(data_232, codebook_23)
print(len(data_233))
print(len(encoded_data))
```

```
193
108
```

2 Part 3: Image Coding

[150]:
```
# image
img = 'sample.png'
#img = 'Lhotse_Mountain_8-Bit_Grayscale.jpg'

# reading image / convert to numpy array
im = np.asarray(Pimage.open(img))
im = im.flatten()

# differential encoding
data_31 = diff_encoding(im)

# run-length encoding
data_32 = rl_encoding(data_31)

# huffman encoding
frequency_3 = frequency_table(data_32)
codebook_3 = codebook_generator(frequency_3)
data_3 = huffman_encoding(data_32, codebook_3)
#im.shape
len(data_3)
```

```
/home/katon/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5:
RuntimeWarning: overflow encountered in ubyte_scalars
  """
/home/katon/.local/lib/python3.6/site-packages/ipykernel_launcher.py:7:
RuntimeWarning: overflow encountered in ubyte_scalars
  import sys
```

[150]: 382