THE UNIVERSITY OF TEXAS AT ARLINGTON

Project Report
On
Data Compression: Lossless Versus Lossy

Instructor:
Prof. Qilian Liang

By-
Deepti Saxena (1001937586)

DEPARTMENT OF ELECTRICAL ENGINEERING
SPRING 2023

Table of Content

Sr.No	Topics	Page Number
1.	Introduction	3
2.	Project Description	5
3.	Huffman Coding	6
4.	Shannon Coding	11
5.	Compressive Sensing	16
6.	Conclusion	20
7.	References	20

INTRODUCTION

Data Compression

Data Compression means reducing the size of files. The advantages of using compression that it is used to reduce data needed to store in hardware, time it is taking for data compression, and bandwidth at which data is communicating.

Data Compression Techniques

There 2 different techniques through which data can be compressed :

- Lossless Data Compression technique

  All of the original data is kept same in lossless data. In this compression technique we compress the file while make sure to keep the information required to its original size when decompression is performed. The Data compression is used various applications, files and documents, and audio or image or video formats used by photographers.

  1) Huffman Coding
  2) Shannon Coding

- Lossy Data Compression techinque

  Lossy compression is another compression technique. This Compression algorithm where data which lost details are not much observable and is been considered by user. The applications where Lossy compression is used includes music, movies, videos, and images.

  1) Compressive Sensing

Compression techniques are used for following Data:

Text Data: In text data the compression technique is used to identify the text which is not much needed and compressing them to shorter symbols. The compression on such text data is determined by the text data type, algorithm and compression ratio used.

Image Data: If compression technique is used for image than it used to count number of occurrence of pixels and make them an one unit and based on compression technique and compression ration the final output come.

Audio Data: The compression technique used for audio files is lossy data compression technique where background noise and white noise are removed from files.

Video Data: The compression technique used for video files is lossy data compression technique where the file contain many images and audio data and they both are used for the compression and make the file of shorter size.

PROJECT DESCRIPTION

Used Python (Jupyter) programming performed the following things:

1. Performed Huffman coding to the two images respectively, and compared the coded bits length with their entropy

2. Performed Shannon coding to the two images respectively, and compared the coded bits length with their entropy

3. Performed Compressive Sensing to the two images, and plotted the decompressed images with different compression ratio.

4. Compared the above three approaches, and drawn the conclusions.

5. The one image is uncompressed data, and other image is processed data.

HUFFMAN CODING

- Huffman coding is a lossless data compression algorithm that assigns binary codes to frequently occurring symbols and long codes to infrequently occurring symbols.

- This works best for data with uneven distribution of symbols, where some symbols occur more frequently than others.

- Huffman coding can achieve high compression rates for such data, but requires prior knowledge of the symbol frequency distribution**.**

- The entropy is a measure of the average uncertainty in the variable. It is the number of bits on required to describe the random variable.
  $-\sum(n*\log_2(n))$

Code

```
#Libraries imported for the huffman coding

import cv2

import math

#Image 1 - Reference Image

Im1 = cv2.imread('ReferImage.png')


#Image 2 - Distorted Image

Im2 = cv2.imread('DistoredImage.png')

import numpy as np


# Entropy Calculation in Huffman coding
```

```
def Entropy(Im):
    histo = np.histogram(Im, bins=256)[0]
    probabili = histo / np.sum(histo)
    probabili = probabili[np.nonzero(probabili)]
    Entropy = -np.sum(probabili * np.log2(probabili))
    return Entropy
# Created Function for Huffman Coding- lossless Compression
from collections import Counter

def huffman_coding(Im):

    # Freq Calculation of [pixels]

    freq = Counter(Im.flatten())    # Created counter for calculation of
pixels

    # Code for the Huffman tree

    tree = [[weight, [symbol, ""]] for symbol, weight in freq.items()]

    while len(tree) > 1:

        tree.sort(key=lambda x: x[0])
        RT = tree[1]
```

```
        LT = tree[0]
        for pair in RT[1:]:
            pair[1] = '1' + pair[1]
        for pair in LT[1:]:
            pair[1] = '0' + pair[1]
        tree = [[LT[0] + RT[0]] + LT[1:] + RT[1:]] + tree[2:]

    # Dict of Codes
    codes = dict(sorted(tree[0][1:], key=lambda x: (len(x[-1]), x)))

        # Convertion to string to calculate the coded bit length
    BinStr = ""
    for pixel in Im.flatten():
        BinStr += codes[pixel]

    return BinStr
    # Huffman coding Image1
BinStr1 = huffman_coding(Im1)

# len of Coded Bits
lengthOfBits1 = len(BinStr1)

#Title
print("UNCOMPRESSED IMAGE:")
```

```
#Entropy of Image 1
Entropy_Im1 = Entropy(Im1)

# Printing the Entropy of Image 1
print("Entropy : %.2f " % (Entropy_Im1))

# Output of Coded Bit length

print("Coded Bit Length: %d \n" % (lengthOfBits1))



# Huffman coding Image2
BinStr2 = huffman_coding(Im2)

# len of Coded Bits
lengthOfBits2 = len(BinStr2)

#Title
print("DISTORTED IMAGE:")

#Entropy of Image 2
Ent_Im2 = Entropy(Im2)

# Printing the Entropy of Image 2
print("Entropy : %.2f" % (Ent_Im2))
```

Output of Coded Bit length

print("Coded Bit Length of : %d" % (lengthOfBits2))

Output

```
UNCOMPRESSED IMAGE:
Entropy : 7.89
Coded Bit Length: 18683469

DISTORTED IMAGE:
Entropy : 7.68
Coded Bit Length of : 18187099
```

The Entropy and Coded Bit length of the Uncompressed Image is greater than Compressed Image

Length of Coded bit is greater than the entropy.

Distored Image is not much compressed as we can see from the entropy of both.

SHANNON CODING

- Shannon coding, is a variable-length coding, the technique in information theory which is used for lossless data compression.

- In Shannon coding, each symbol in input is assigned a variable-length code. The codes are assigned in such a way that no code is a prefix of any other code.

- To assign the codes, the symbols are first sorted in descending order of their probability of occurrence.

Code:

```
#Libraries Imported needed for Shannon Coding
import numpy as np
import cv2            #Cv2-python to open the image file
import math           # Math funtions
from collections import Counter
#Image 1
Im1 = cv2.imread('ReferImage.png')
img11 = np.array(Im1)


image_pixels = Counter(img11.ravel())   #count of pixels in image1
probab = [count / sum(image_pixels.values()) for count in
image_pixels.values()]
```

```
#Image 1 Entropy
entropy = sum([-p * math.log2(p) for p in probab])

print("UnCompressed Image")
print("Entropy of ReferenceImage/UncompressedImage:", entropy)

#Image2
Im2 = cv2.imread('DistoredImage.png')
img11 = np.array(Im2)

image_pixels = Counter(img11.ravel())  #count of pixels in image2
probab = [count / sum(image_pixels.values()) for count in
image_pixels.values()]

#Image 2 Entropy
entropy = sum([-p * math.log2(p) for p in probab])

print("Distorted Image")
print("Entropy of DistoredImage/ProcessedImage:", entropy)

#Calculation of probability on Reference Image 1

probabi1 = np.zeros(256)
for a in range(Im1.shape[0]):
```

```
    for b in range(Im1.shape[1]):
        probabi1[Im1[a][b]] = probabi1[Im1[a][b]] + 1
probabi1 = probabi1 / (Im1.shape[0] * Im1.shape[1])
probabi1_sor = np.sort(probabi1)[::-1]                #sort the probaility



#Calculation of probability on Distorted Image 2
probabi2 = np.zeros(256)
for a in range(Im2.shape[0]):
    for b in range(Im2.shape[1]):
        probabi2[Im2[a][b]] = probabi2[Im2[a][b]]+ 1
probabi2 = probabi2 / (Im2.shape[0] * Im2.shape[1])
probabi2_sor = np.sort(probabi2)[::-1]                #sort the probaility

#defined the function to calcuate the Shannon Coding

def ShannonCoding(prob):
    if len(prob) == 1:
        return {'0': prob[0]}
    sp = 1
    diff = abs(sum(prob[:sp]) - sum(prob[sp:]))
    while abs(sum(prob[:sp+1]) - sum(prob[sp+1:])) <= diff and sp <
len(prob)-1:
        sp += 1
```

```
        diff = abs(sum(prob[:sp]) - sum(prob[sp:]))
    code = {}
    for key, value in ShannonCoding(prob[:sp]).items():
        code['0' + key] = value
    for key, value in ShannonCoding(prob[sp:]).items():
        code['1' + key] = value
    return code
# Calculation of Coded Bit length on Image 1 and Image 2

code1 = ShannonCoding(probabi1_sor)
len1 = 0
for key, value in code1.items():
    len1 = len1 + len(key) *
probabi1_sor[np.where(list(code1.values())==value)[0][0]]



code2 = ShannonCoding(probabi2_sor)
len2 = 0
for key, value in code2.items():
    len2 = len2 + len(key) *
probabi2_sor[np.where(list(code2.values())==value)[0][0]]
# Print length  of Coded Bit for Reference Image

print("UnCompressed Image")
print("Coded Bit Length : %.2f" % (len1))
```

Print length of Coded Bit for Distorted Image

print("Distorted Image")

print("Coded Bit Length : %.2f" % (len2))

Output:

```
UnCompressed Image
Entropy of ReferenceImage/UncompressedImage: 7.894773129230776
Distorted Image
Entropy of DistoredImage/ProcessedImage: 7.682354520811326

UnCompressed Image
Coded Bit Length : 26.61
Distorted Image
Coded Bit Length : 26.04
```

In Shannon Coding the Entropy and Coded Bit length of the Uncompressed Image is greater than Compressed Image

length of Coded bit length is greater than the entropy in Shannon Coding.

Compressive Sensing

- Compression sensing is a well-known method for signal/image collection with a comparatively low sample rate .

- The rate of data collection, the necessity for sensory equipment, the amount of memory that must be stored, and the power required for those devices are all significantly reduced by compressive sensing, which is based on a sparse assessment of the information sources.

- Image and audio compression, tomography, and medical imaging are all applications of the Compressive Sensing technique in signal processing.

- It has also found use in wireless sensor networks, where it can reduce energy consumption and improve network lifetime by reducing the number of measurements needed for signal reconstruction.

Code:

```
#Importing libraries
import numpy as np
import cv2          #Cv2-python used to open the image file
```

```
# Reference/Uncompressed Image 1
Im1 = cv2.imread('ReferImage.png',)

# Distorted/Processed Image 2
Im2 = cv2.imread('DistoredImage.png',)
#Importing libraries for compressive sensing

import matplotlib.pyplot as plt
import pywt
# defined function for Compressive Sensing on Image 1

def CompSens(Im, CompRatio):

    Coeff = pywt.dwt2(Im, 'db2', mode='periodization')
    CoeffSor = np.sort(np.abs(Coeff[0].flatten()))[::-1]
    Index = int(np.floor(CompRatio * len(CoeffSor)))
    thres = CoeffSor[Index]
    Coeff[0][np.abs(Coeff[0]) < thres] = 0
    ImageComp = pywt.idwt2(Coeff, 'db2',
mode='periodization')
    return ImageComp
# Different Compression Ratios are defined
CompressRatios = [0.01, 0.04, 0.2, 0.3, 0.7, 0.8]
```

```
# Defined the no of rows, coloumns and size of each figure in plot
fig, axs = plt.subplots(2, len(CompressRatios), figsize=(20, 8))

# This will go for each compression ratio defined above for Image 1 and Image 2
for a, CompressRatio in enumerate(CompressRatios):
    # dECOMPRESS iMAGE1
    compressed_img1 = CompSens(Im1, CompressRatio)
    axs[0, a].imshow(compressed_img1)
    axs[0, a].axis('off')
    axs[0, a].set_title(f'CR = {CompressRatio:.2f}')

    # dECOMPRESS iMAGE2
    compressed_img2 = CompSens(Im2, CompressRatio)
    axs[1, a].imshow(compressed_img2)
    axs[1, a].axis('off')
    axs[1, a].set_title(f'CR = {CompressRatio:.2f}')

# defined Plot to show image
plt.tight_layout()
plt.show()
```

Output:

Image 1:

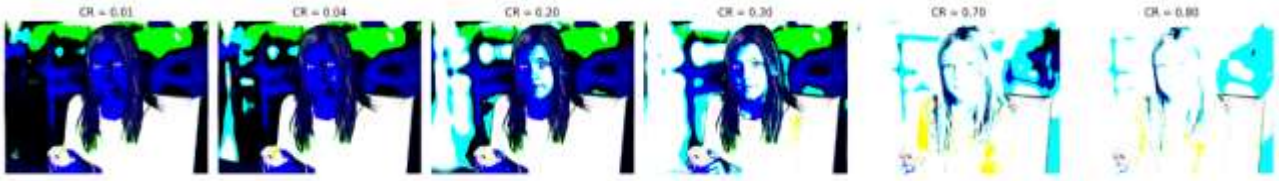

Image 2:

CONCLUSION

All the above three approaches are used for the various application.

But from the above output I can find that Huffman Coding results are optimal and has reduced code bit length compared to Shannon Coding.

Also, I found that its better than Compressive sensing because the CS is used in specific applications where the signal being compressed is sparse unlike Huffman code which are used in wide applications.

So From the above three approaches according to me Huffman coding is better.

REFERENCES

- **Lecture Notes**
- **https://www.python.org/about/help/**