CS517

Assignment Report

LZW Coding

-Yashasav Prajapati(2021CSB1143)

Introduction

Lempel-Ziv-Welch (LZW) coding is a lossless data compression algorithm that is used to compress data in a way that can be decompressed to its original form without losing any information.

Approach

LZW Coding: The basic idea of LZW coding is to replace frequently occurring sequences of data with a single code. This is done by building a dictionary of all possible sequences of input data and assigning a unique code to each sequence. As the input data is processed, the algorithm identifies sequences that are already in the dictionary and replaces them with their corresponding code. When a new sequence is encountered, it is added to the dictionary with a new code. Here is the code block describing it, the code is working by breaking the image into several nxn blocks and then applying the LZW Coding on it.

```python
def lzw_encode(img):
    rows, cols = img.shape[0],img.shape[1]
    # initialize the dictionary
    codeDict = {}
    for i in range(256):
        codeDict[i] = str(i)
        codeDict[str(i)] = i

    # initialize other variables needed for the table
    encodedSequence = np.array([], dtype=np.uint32)
    recognizedSequence = ""
    currentlyProcessingPixel = ""

    index = 256
    for i in range(0, rows):
        for j in range(0, cols):
            # at the beginning, we don't have any recognized sequence, so we simply get the curre
            if recognizedSequence=="":
                # set currentlyProcessingPixel to the current pixel we are at
                currentlyProcessingPixel = str(img[i][j])
                # add this currentlyProcessingPixel to the recognizedSequence which was earlier e
                recognizedSequence = currentlyProcessingPixel
            else:
                # set currentlyProcessingPixel to the current pixel we are at
                currentlyProcessingPixel = str(img[i][j])
                # if the sequence exists in our dict, great, we can simply work further
                if f"{recognizedSequence}_{currentlyProcessingPixel}" in codeDict:
                    # update the recognized sequence
                    recognizedSequence = recognizedSequence + '_' + currentlyProcessingPixel
                else: # sequence does not exist in codeDict
                    # add the recognized sequence to the dictionary and it's index should be the
                    newSequence = f"{recognizedSequence}_{currentlyProcessingPixel}"
                    codeDict[newSequence] = index
                    codeDict[index] = newSequence
                    index+=1
                    # add the code of the recognized sequence to the encoded sequence
                    codeDictKey = str(recognizedSequence)
                    encodedSequence = np.append(encodedSequence, codeDict[codeDictKey])

                    # reset the recognized sequence
                    recognizedSequence = currentlyProcessingPixel
            if(i+1==rows and j+1==cols):
                codeDictKey = str(recognizedSequence)
                encodedSequence = np.append(encodedSequence, codeDict[codeDictKey])
                break
    # codeDict = getCorrectCodeDict(codeDict)
    newCodeDict = {}
    for i in encodedSequence:
        newCodeDict[i] = codeDict[i]

    # print("Done Encoding!")
    return encodedSequence, newCodeDict
```

LZW Decoding: LZW decoding works by reversing the LZW encoding algorithm to retrieve the original data from the compressed data. It builds a dictionary of all possible codewords and uses it to decode the encoded data.

The decoder reads the encoded data and replaces the codes with the corresponding entries in the dictionary, adding new entries in the dictionary as it goes. This continues until all the encoded data has been decoded, resulting in the original uncompressed data.

Here is the code snippet describing it.

```
how.py > lzw_decode
def lzw_decode(encodedSequence, blockSize):
    decodedSequence = []
    imgBlock = np.zeros((blockSize, blockSize), dtype='int8')
    # initialize the dictionary
    codeDict = {}
    # fill the dictionary with the first 256 values
    for i in range(256):
        codeDict[i] = str(i)
        codeDict[str(i)] = i
    # get the first pixel from the encoded sequence and them remove it from th
    currentPixel = str(encodedSequence[0])
    # add the first pixel to the decoded sequence
    decodedSequence.append(currentPixel)
    encodedSequence = encodedSequence[1:]
    NewEntryIndex = 256
    for item in encodedSequence:
        newEntry = ""
        # if the item is in the dictionary, then add it to the decoded sequenc
        if item in codeDict:
            newEntry = codeDict[item]
        elif(item == NewEntryIndex): # if the item is equal to the NewEntryInd
            newEntry = currentPixel + '_' + currentPixel.split('_')[0]
        else:# else there is an error
            raise ValueError("This is not a valid encoded array, item does not
        # add the new entry to the decoded sequence
        decodedSequence.append(newEntry)
        stringToAdd = currentPixel + '_' + newEntry.split('_')[0]
        # add the new entry to the dictionary
        codeDict[NewEntryIndex] = stringToAdd
        codeDict[stringToAdd] = NewEntryIndex
        # increment for the next entry
        NewEntryIndex+=1
        # update pixel value
        currentPixel = newEntry
    decodedSequence = [int(j) for i in decodedSequence for j in i.split('_')]
    # now we need to convert the decoded sequence into an image
    CurrIndex = 0
    for i in range(blockSize):
        for j in range(blockSize):
            imgBlock[i,j] = decodedSequence[CurrIndex]
            CurrIndex +=1

    return imgBlock
```

Analysis

What was analyzed?

- **Compression Ratio**: It is the ratio of the number of original bits used to send the information with the number of bits sent when the data was compressed.
- **Entropy**: It is the measure of how much information is present in that image. A higher entropy means that the image has several changes in intensity/pixel values and a lower entropy means the image has a low amount of changes in the pixel/intensity values. It is given by $\sum_{all\ p} (-\ p\ *\ log(p))$
- **Average Length of Encoded pixels**: $len(encodedSequence)/totalPixelsInImage$
- **Maximum Number of codes used**: It is the maximum value of all the encoded values, this describes how many codes we have used for the coding.

Here are the results for **Compression Ratio, Entropy, Average length of encoded pixels, Maximum number of codes used** that were obtained for the images provided on 8x8 blockSize:

For **book-cover.tif**

```
CompressionRatio for file book-cover.tif =  1.426951286789643
Entropy for file book-cover.tif =  7.425825181235831
AvgLengthOfEncodedPixels for file book-cover.tif =  44.85086533261222
MaxCode for file book-cover.tif =  316
```

For **checkerboard1024.tif**

```
CompressionRatio for file checkerboard1024.tif =  5.818181818181818
Entropy for file checkerboard1024.tif =  1.0
AvgLengthOfEncodedPixels for file checkerboard1024.tif =  11.0
MaxCode for file checkerboard1024.tif =  264
```

For **Fig81a.tif**

```
CompressionRatio for file Fig81a.tif =  4.565695973247875
Entropy for file Fig81a.tif =  1.6613969600735903
AvgLengthOfEncodedPixels for file Fig81a.tif =  14.017578125
MaxCode for file Fig81a.tif =  280
```

For **zoneplate.tif**

```
CompressionRatio for file zoneplate.tif =  1.2927936914421685
Entropy for file zoneplate.tif =  6.842690750389012
AvgLengthOfEncodedPixels for file zoneplate.tif =  49.01137777777778
MaxCode for file zoneplate.tif =  317
```

For **Fig81b.tif**

```
CompressionRatio for file Fig81b.tif =  2.0
Entropy for file Fig81b.tif =  8.0
AvgLengthOfEncodedPixels for file Fig81b.tif =  32.0
MaxCode for file Fig81b.tif =  285
```

For **fingerprint.tif**

```
CompressionRatio for file fingerprint.tif =  1.345993626423931
Entropy for file fingerprint.tif =  6.56790603015178
AvgLengthOfEncodedPixels for file fingerprint.tif =  47.33083333333333
MaxCode for file fingerprint.tif =  304
```

For **Fig81c.tif**

```
CompressionRatio for file Fig81c.tif =  1.8572278742879813
Entropy for file Fig81c.tif =  1.813469721025499
AvgLengthOfEncodedPixels for file Fig81c.tif =  34.4599609375
MaxCode for file Fig81c.tif =  294
```

For **lena.tif**

```
Analysis for whole image as 1 block
CompressionRatio for file lena.tif = 1.1204219362394163
Entropy for file lena.tif = 7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif = 57.121337890625
MaxCode for file lena.tif = 317
```

For **matches-random.tif**

```
CompressionRatio for file matches-random.tif =  1.0755772264448586
Entropy for file matches-random.tif =  7.425424823672172
AvgLengthOfEncodedPixels for file matches-random.tif =  59.50293333333333
MaxCode for file matches-random.tif =  317
```

For **matches-aligned.tif**

```
CompressionRatio for file matches-aligned.tif =  1.0700337060617409
Entropy for file matches-aligned.tif =  7.346316371461064
AvgLengthOfEncodedPixels for file matches-aligned.tif =  59.8112
MaxCode for file matches-aligned.tif =  317
```

Here is the analysis for blockSize of **4x4**, **8x8**, **16x16**, **32x32**, **64x64**, **128x128**, **256x256** on **lena.tif**

4x4

```
Analysis for 4x4 blocksize
CompressionRatio for file lena.tif =  1.0487396033781269
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  15.25640869140625
MaxCode for file lena.tif =  269
```

8x8

```
Analysis for 8x8 blocksize
CompressionRatio for file lena.tif =  1.1204219362394163
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  57.121337890625
MaxCode for file lena.tif =  317
```

16x16

```
Analysis for 16x16 blocksize
CompressionRatio for file lena.tif =  1.2251666152566296
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  208.951171875
MaxCode for file lena.tif =  495
```

32x32

```
Analysis for 32x32 blocksize
CompressionRatio for file lena.tif =  1.3665219227140273
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  749.34765625
MaxCode for file lena.tif =  1162
```

64x64

```
Analysis for 64x64 blocksize
CompressionRatio for file lena.tif =  1.5441035276931867
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  2652.671875
MaxCode for file lena.tif =  3455
```

128x128

```
Analysis for 128x128 blocksize
CompressionRatio for file lena.tif =  1.7604545118765405
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  9306.6875
MaxCode for file lena.tif =  10661
```

256x256

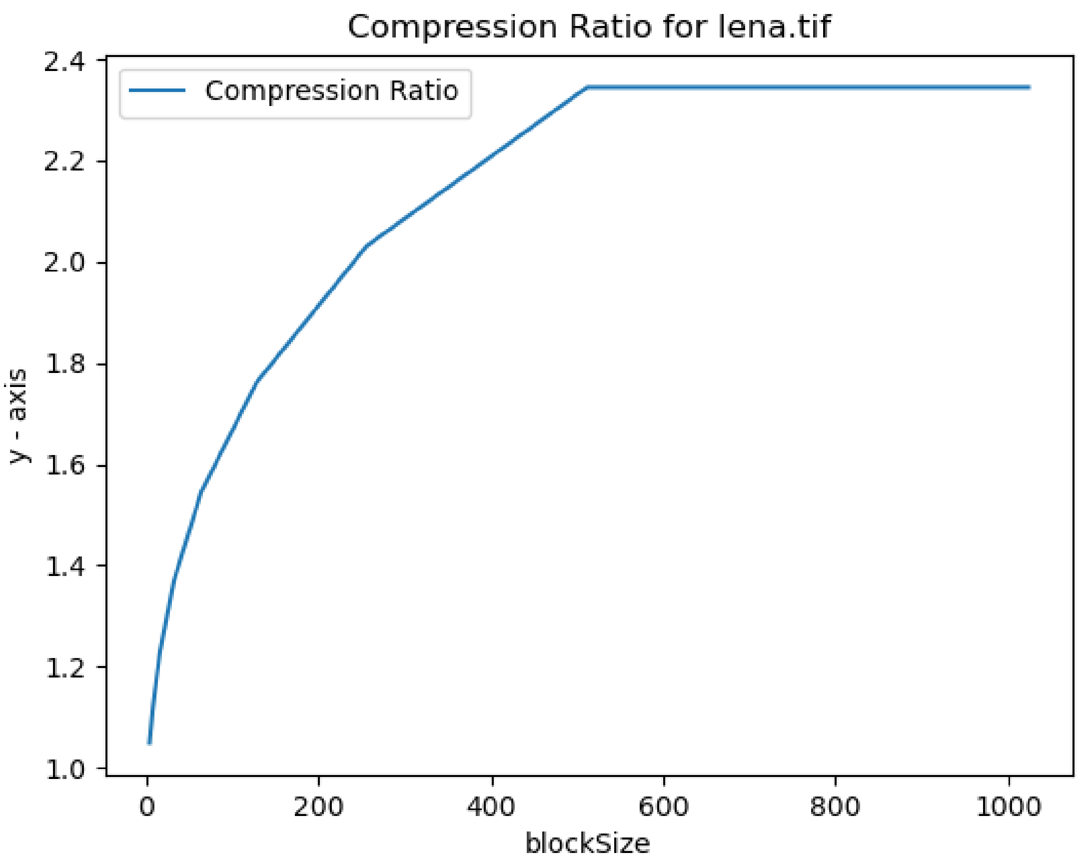
```
Analysis for 256x256 blocksize
CompressionRatio for file lena.tif =  2.031006190391335
Entropy for file lena.tif =  7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif =  32267.75
MaxCode for file lena.tif =  34414
```

Entire Image as 1 block

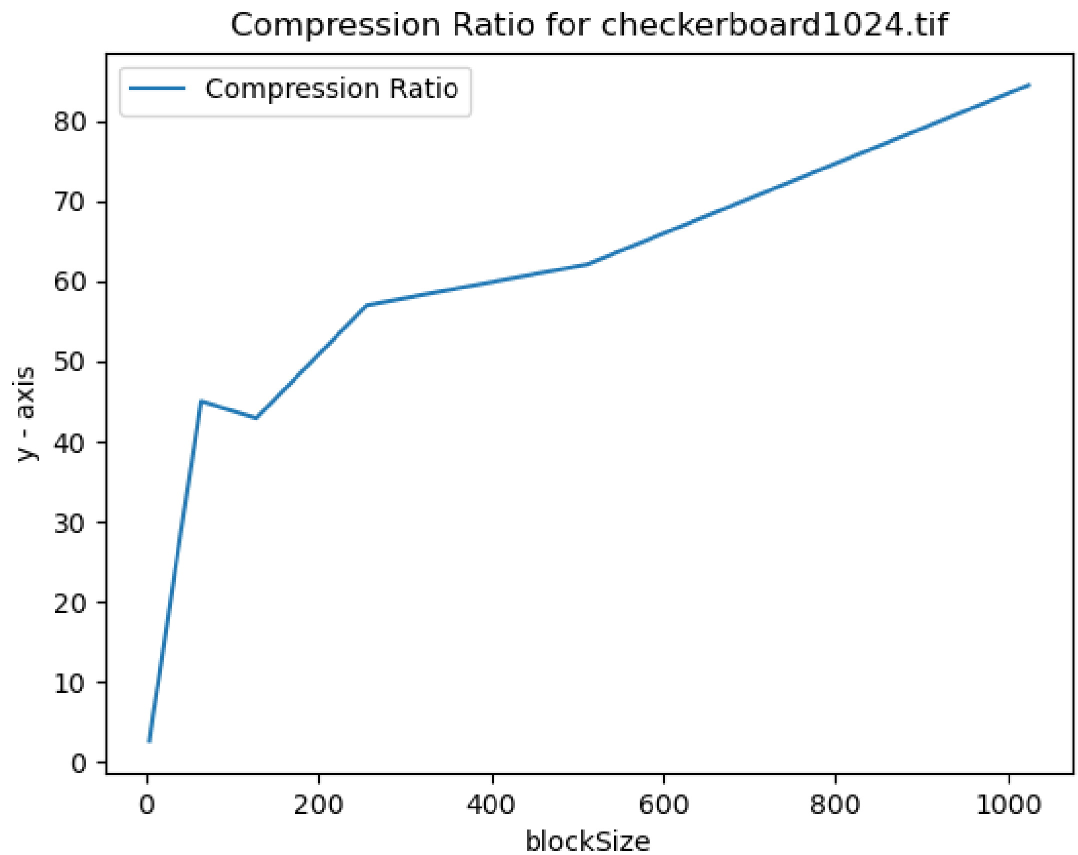
```
Analysis for whole image as 1 block
CompressionRatio for file lena.tif = 2.3450521532213338
Entropy for file lena.tif = 7.383779807515338
AvgLengthOfEncodedPixels for file lena.tif = 111786.0
MaxCode for file lena.tif = 111844
```

Graph between Compression Ratio and Block Size

For **lena.tif**



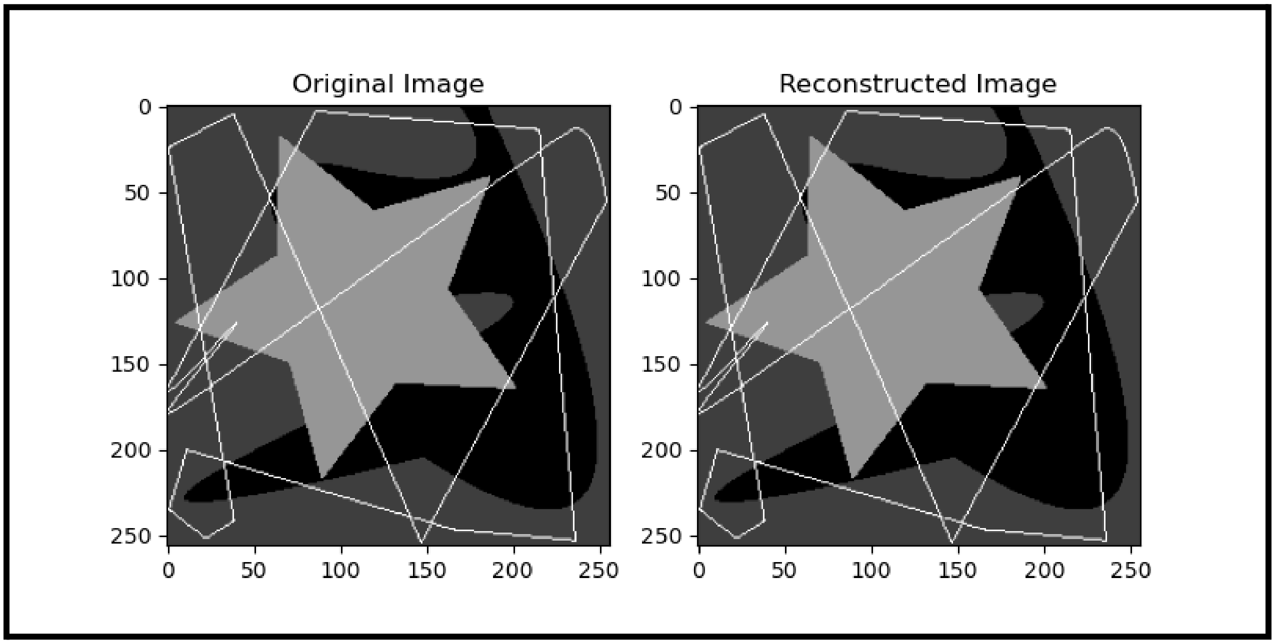
For **checkerboard1024.tif**



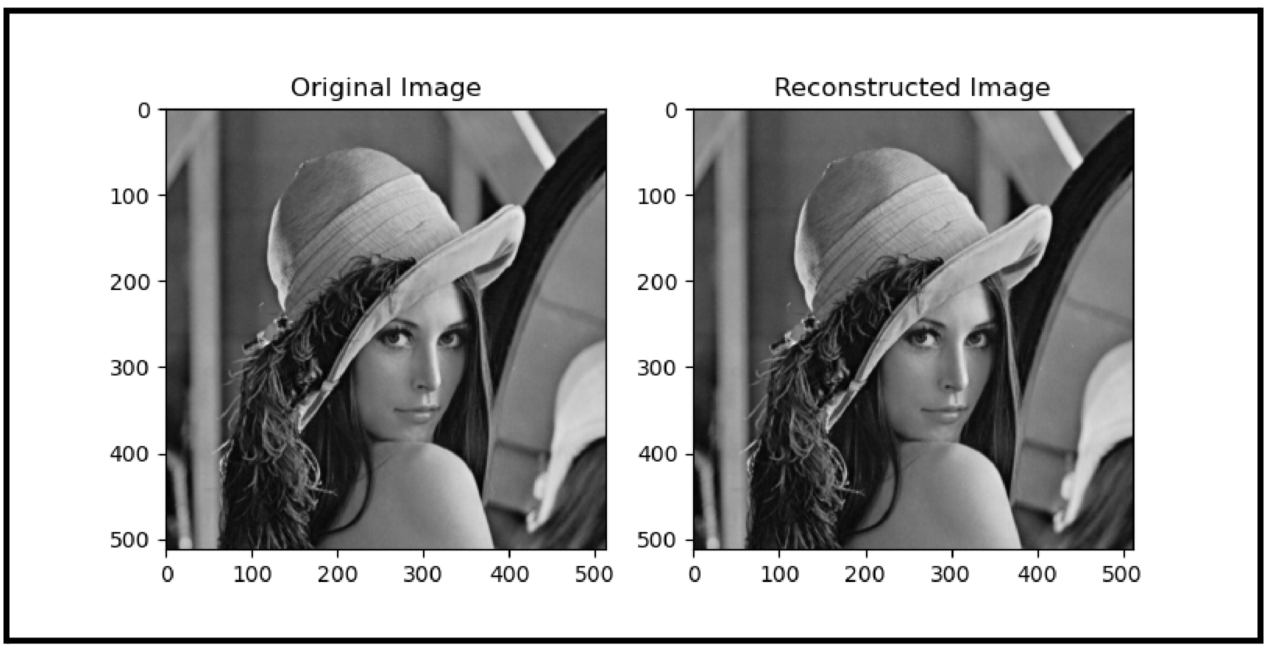
Results

Here are the results for various images before and after compression. Since the LZW compression is lossless type of compression, both the before and after images are identical.

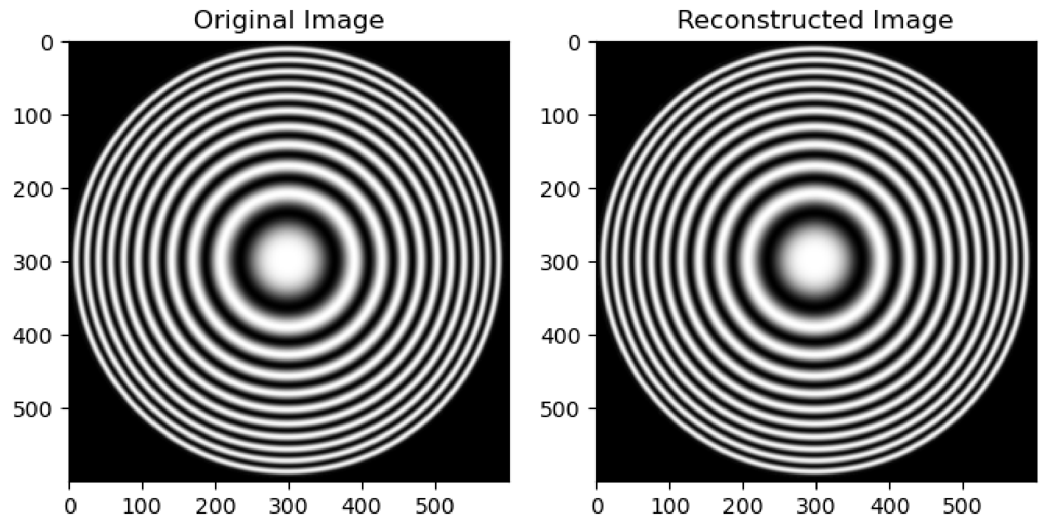
For **Fig81a.tif**



For lena.tif



For zoneplate.tif


Original Image
Reconstructed Image
0
100
200
300
400
500
0
100
200
300
400
500

Maximum Compression Achievable by reducing the number of bits required:

These are the maximum values for the grayscale images after compression, i.e. after creating the encoded sequence for 8x8 blocksize.

```
MAX CODE: 280
MAX CODE: 317
MAX CODE: 304
MAX CODE: 264
MAX CODE: 317
MAX CODE: 317
MAX CODE: 294
MAX CODE: 316
MAX CODE: 317
MAX CODE: 285
```

Clearly the maximum value of all these is **317**, which can be represented in **9 bits**. So we can encode all the values in 9 bits as **every other value will be either 317 or less than 317. This is to reduce the number of bits that we will transfer.**

Observations:

1. In **checkerboard1024.tif**, the entropy is **1** which is clear as there are not many changes in the pixel values.
2. Higher entropy images have lower compression ratios as compared to lower entropy images.
3. As evident from the graphs, **as the block size increases, compression ratio also increases**. This is quite trivial to understand, as the blocksize increases, the dictionary will use the already existing patterns instead of remaking them again and again in other dictionaries and it will find longer patterns which will ultimately reduce the overall compression.

Conclusion:

In this I learnt about how LZW Coding and Decoding works, how we implement its dictionaries. I also gained an understanding of how the LZW algorithm can be implemented in Python. In Python, we can implement the LZW algorithm using built-in data structures such as dictionaries and lists. I learned how to create these data structures and update them during the compression and decompression processes. I learnt how the blocksize affects the compression ratio and also about different terms like Entropy. Overall, this project provided me with a deeper understanding of data compression techniques and how they can be implemented using different programming languages. The LZW algorithm is a powerful and efficient compression technique that can be used to reduce the size of data files without losing any information.