Multimedia Final Task

Option [1]

Compression Algorithm

Name: Mahmoud Amr Mohamed Refaat

Student ID: 9181351

Sec: 2

BN: 25

Email: [mahmou.m.gody@gmail.com](mailto:mahmou.m.gody@gmail.com)

Table of Contents

[Case tackled for compression 3](#_Toc41354941)

[Best existing algorithms 3](#_Toc41354942)

[Algorithm Idea 3](#_Toc41354943)

[Algorithm 4](#_Toc41354944)

[Encoding 4](#_Toc41354945)

[Decoding 4](#_Toc41354946)

[Experiments Description 5](#_Toc41354947)

[Software Environment 5](#_Toc41354948)

[My Code 5](#_Toc41354949)

[LBG algorithm 5](#_Toc41354950)

[Hardware Environment 5](#_Toc41354951)

[Benchmarks 6](#_Toc41354952)

[Notes about the algorithms 8](#_Toc41354953)

[Parameters used 8](#_Toc41354954)

[Notes about the experiments 8](#_Toc41354955)

[Results 9](#_Toc41354956)

[Compression ratio 9](#_Toc41354957)

[Notes 9](#_Toc41354958)

[Encoding Time (seconds) 11](#_Toc41354959)

[Decoding Time (seconds) 12](#_Toc41354960)

[Quality Measure (PSNR) 13](#_Toc41354961)

[Decoded Images 14](#_Toc41354962)

# Case tackled for compression

The case tackled for compression is specifically grey scale images since they have only 256 levels and, in most images, adjacent pixels have close values for levels. Moreover, the difference between the close levels in greyscale is not obvious to the human eye for example the difference between levels 1 and 2.

# Best existing algorithms

* Linde-Buzo-Gray (LBG) algorithm using vector quantization

# Algorithm Idea

The idea of the algorithm is a **lossy** compression technique based on the ideas of quantization and run length encoding that depends mainly on the fact that adjacent pixels in a grey scale image have very close values. The algorithm has multiple stages. First, the image is prepared for run length encoding by pixel value by combining multiple levels in greyscale to one level for example (2, 4, 8, 16, 32, 64, 128). The more levels combined, the larger the compression ratio and the more the distortion in the image. Image quality starts to drop apparently at 16 or 32 levels combined depending on the image. By flattening the image and doing one pass over the image vector to replace the levels with the combined levels of greyscale, the probability of finding runs in the image increases. Next, a similar approach to run length encoding but depending on the pixel values (grey scale levels) is applied by doing another pass on the image vector storing the value of each greyscale level in an array and the value of the count (number of adjacent pixels with the same level) in another array. There will be a case that the count of a run is one (no adjacent pixel having the same level) which would be stored in two bytes one for the level and the other for the count which will be one in that case. To prevent such case depending on the idea that when combining multiple levels of the greyscale together we are uniting a number of the most significant bits of the greyscale depending on the number of levels combined leaving the least significant bit always at zero (even numbers). We can use the least significant bit to indicate that the length of the run was one by adding one (it becomes odd) to the greyscale level and storing the level only without the count in one byte. Finally, Huffman encoding is applied to both the levels array and the counts array separately to improve the compression ratio.

# Algorithm

## Encoding

* Take the Path of the image to be compressed as an input
* Take the number of greyscale levels to combine as an input
* Reshape the image into a one-dimensional vector
* Do a pass over the image and replace the image levels with the available greyscale levels after combining them where if the current level is not divisible by the number of levels to combine subtract the remainder from the current level
* Apply Run length encoding based on the pixel values as follows:
  + If the count of the run is more than one store the grey scale level in the levels array and store the count indicating the number of adjacent pixels in the counts array
  + If the count of the run is one add one to the value of the level and store it in the levels array, without storing the count in the counts array
* Use Huffman encoding to encode both the produced levels and count arrays
* Save the output of Huffman encoding as two binary files

## Decoding

The decoder should receive the following input:

* The original image dimensions
* The two code tables used by Huffman encoder to encode the levels and the counts array
* The two coded binary strings representing the levels and count arrays
* The original image file extension

Decoding algorithm:

* Decode the binary strings of the levels and counts arrays using the given code table
* Loop on both the levels array and the counts array to restore the image vector as follows:
  + If the level from the levels array is even, read the corresponding count from the counts array and add the level to the decoded stream count times
  + If the level is odd indicating a one length sequence, read the level from the levels array only, subtract one from it and add it to the decoded stream without incrementing the corresponding count index

Using the original image dimensions, reshape the vector back to a 2D matrix

* Using the given image file extension write the decoded image to the disk as “DecodedImage.extension”

# Experiments Description

## Software Environment

### My Code

|  |  |
| --- | --- |
| **Programming language** | Python 3.7 |
| **OS** | Windows 10 Home |
| **IDE** | PyCharm Community Edition 2019.3.3 |
| **Project Requirements** | * Python **3.7** * OpenCV-python **4.2.0.34** * NumPy **1.18.4** |

### LBG algorithm

|  |  |
| --- | --- |
| Programming language | MATLAB |
| Source | https://github.com/aashishkr/Image-Compression-Using-Vector-Quantization-with-LBG-algorithm |

## Hardware Environment

|  |  |
| --- | --- |
| **Processor** | Intel® Core™ i7-4510 CPU |
| **RAM** | 8 GB |

## Benchmarks

Source of benchmarks: <http://links.uwaterloo.ca/Repository.html>

|  |  |
| --- | --- |
| Image | Details |
| A close up of a monkey  Description automatically generatedmandrill.tif | **Row:** 512  **Cols:** 512  **Size on disk:** 262.27 kB  **Color:** 8 bit color |
| A close up of a fruit  Description automatically generatedpeppers2.tif | **Row:** 512  **Cols:** 512  **Size on disk:** 262.27 kB  **Color:** 8 bit color |
| A picture containing outdoor, person, object, water  Description automatically generatedcameraman.tif | **Row:** 256  **Cols:** 256  **Size on disk:** 65.13 kB  **Color:** 8 bit color |
| A close up of text on a white background  Description automatically generatedtext.tif | **Row:** 256  **Cols:** 256  **Size on disk:** 65.13 kB  **Color:** 8 bit color  **Note:** This image is the best case for the algorithm as it includes two greyscale levels only |

## Notes about the algorithms

### Parameters used

All experiments are run with the following parameters:

**My algorithm**

|  |  |
| --- | --- |
| Number of combined levels of grey scale(taking b MSB from grey scale) | Each 8 levels of grey scale are combined as one level in quantization phase(Ex: levels from 0 : 7 are replaced by 0 and so on) |

**LBG algorithm**

|  |  |
| --- | --- |
| **K (number of means for the codebook)** | 256 each is a 4-dimensional vector according to the blocksize |
| **Block Size** | 2 \* 2 (4-dimensional vectors) |

### Notes about the experiments

1. I applied my algorithm using two approaches:

* The first one is the one delivered as source code which involved fixing the count before applying Huffman encoding to the levels and the counts separately. Fixing the count means if the length of the count of a run is one store the level as an odd number without storing a corresponding count depending on the idea that after quantization all the available levels are even numbers. By that way single element runs will be distinguishable to the decoder and will only require one byte to store them
* The second one is without fixing the count before applying Huffman which involves calculating the levels and the run counts for the whole images.

1. By decreasing the number of combined levels of grey scale levels , the rate of the image increases, but the distortion decreases

# Results

## Compression ratio

### Notes

Compression ratio for **my algorithm** is calculated as follows:

Where:

* **image size on disk** is mentioned in the **Experiments Description** section
* **huff\_levels** is a NumPy file produced containing the binary string of the Huffman encoding of the grey scale levels present in the image
* **huff\_counts** is a NumPy file produced containing the binary string of the Huffman encoding of the run counts present in the image
* **cnts\_levels\_list, cnts\_prob\_list, lvl\_levels\_list, lvl\_prob\_list** are NumPy files including the levels and their probabilities in binary passed the Huffman decoder to be able to rebuild the Huffman tree

Compression ratio for **LBG algorithm** is calculated as follows:

Where:

* **k** is the number of codewords in the codebook trained on the same image
* **n** is square the dimension of the chosen block size
* **codebook** is stored as double(64 bit)
* **the indices array** is the index representing each block according to the closest vector in the codebook

|  |  |  |  |
| --- | --- | --- | --- |
| image | My algorithm  With Count Fix | My algorithm  Without Count Fix | LBG Algorithm |
| mandrill.tif | 1.907 | 1.826 | 4 |
| peppers2.tif | 2.296 | 2.281 | 4 |
| cameraman.tif | 2.767 | 2.733 | 3 |
| text.tif | 10.679 | 9.964 | 3 |

## Encoding Time (seconds)

|  |  |  |  |
| --- | --- | --- | --- |
| image | My algorithm  With Count Fix | My algorithm  Without Count Fix | LBG Algorithm |
| mandrill.tif | 7.6569 | 9.1831 | 30.3721 |
| peppers2.tif | 6.3011 | 5.2411 | 23.4333 |
| cameraman.tif | 1.2809 | 0.9894 | 4.2866 |
| text.tif | 0.7238 | 0.4878 | 1.2689 |

## Decoding Time (seconds)

|  |  |  |  |
| --- | --- | --- | --- |
| image | My algorithm  With Count Fix | My algorithm  Without Count Fix | LBG Algorithm |
| mandrill.tif | 0.8797 | 1.5441 | 0.9058 |
| peppers2.tif | 0.5678 | 0.8371 | 0.9382 |
| cameraman.tif | 0.2919 | 0.1039 | 1.0875 |
| text.tif | 0.1559 | 0.0399 | 0.4908 |

## Quality Measure (PSNR)

|  |  |  |  |
| --- | --- | --- | --- |
| image | My algorithm  With Count Fix | My algorithm  Without Count Fix | LBG Algorithm |
| mandrill.tif | 35.6901 | 35.6901 | 30.7147 |
| peppers2.tif | 35.6675 | 35.6675 | 35.5755 |
| cameraman.tif | 35.7495 | 35.7495 | 31.9144 |
| text.tif | Infinite  . as image is restored the same which is a very special case | Infinite | 45.4128 |

## Decoded Images

|  |  |  |
| --- | --- | --- |
| Original | My algorithm | LBG |
| **A close up of a monkey  Description automatically generated** | A close up of a monkey  Description automatically generated | A close up of a monkey  Description automatically generated |
| **A close up of a fruit  Description automatically generated** | A close up of a fruit  Description automatically generated | A close up of a fruit  Description automatically generated |
| **A picture containing outdoor, person, object, water  Description automatically generated** | A picture containing outdoor, person, object, man  Description automatically generated | A picture containing outdoor, person, object, water  Description automatically generated |
| **A close up of text on a white background  Description automatically generated** | A close up of text on a white background  Description automatically generated | A close up of text on a white background  Description automatically generated |