Final Report

Presented to:

Dr. Ahmed Hamdy

By:

Ahmed Walid Fathy

9180205

ahmedwa1999@gmail.com

1. INTRODUCTION

There are three types of methods we can use to compress an image. The first method is exploiting inter-pixel redundancies, in which the pixels of the image are not independent of each other, so we can predict accurately the value of some pixels knowing some other pixels. The second method is to take advantage of coding redundancies. The third method is to avail human limited ability to spot the difference between close colors. The later one is the we are going to use.

2. ALGORITHM

Mine is a lossy algorithm for grayscale image compression. The idea is totally mine and I didn't develop an existing algorithm.

I exploited the fact that we cannot spot the difference between close colors, so if many pixels are close to each other I encode them in a tuple (number of occurrences, value)

The algorithm is as the following:

1-Transform the image to an array.

2-define a "delta" and set a counter to 1.

3-start with the first element in the array and compare it to element after it.

4-If the absolute value of the difference is less than delta, I will increase the counter and compare the first element with the third, and so on and so forth. Till I find an element that does not satisfy the condition then I save the value of the pixel and the number of times it has occurred and continue.

2.1 EXAMPLE

Let delta equal to 5. And we will try to compress the following array using our algorithm.

-Data: [30 34 29 27 26 36 35 32 30 29 30]

-Compressed data: [(5,30), (3,36), (3,30)]

-Compression ratio = 1.83

*If we change the delta a little to be 6

-Compressed data: [(11,30)]

-Compression ratio = 5.5

2.2 ALGORITHM PARAMETERS

There is one parameter for this algorithm which is delta. As delta increases the size of the compressed file decreases. But also, as delta increases the distortion in the decoded image increases. So, the user of this algorithm should try different values of delta and choose the largest delta that gives an acceptable quality of decoded image. However, a value of 7 is safe, it gives a good quality of the decoded image and a good compression ratio for most images.

2.3 COMPLEXITY

The complexity of this algorithm is $O(n)$ for both decoding and encoding which makes it by far faster than most compression algorithms.

3. HARDWARE AND SOFTWARE

- Python 3

- Processor: Intel® Core ™ i7-8750H CPU @ 2.20GHz

- Windows 10 Home

- RAM: 8.00GB

4. EXPERIMENTS

4.1 EXPERIMENT 1

-Original size = 2.54 MB

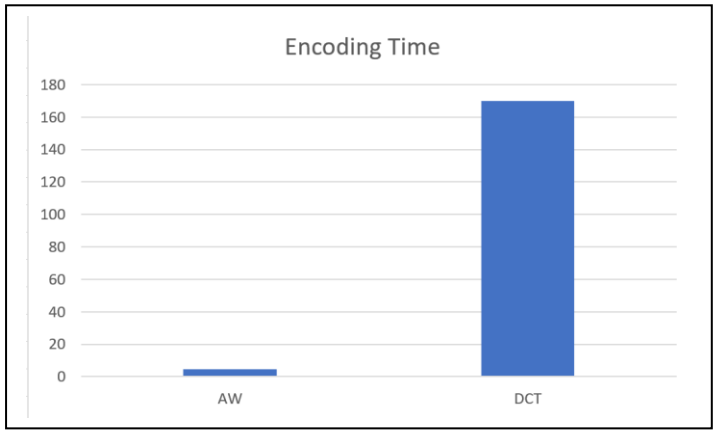
-Coded size = 856 KB

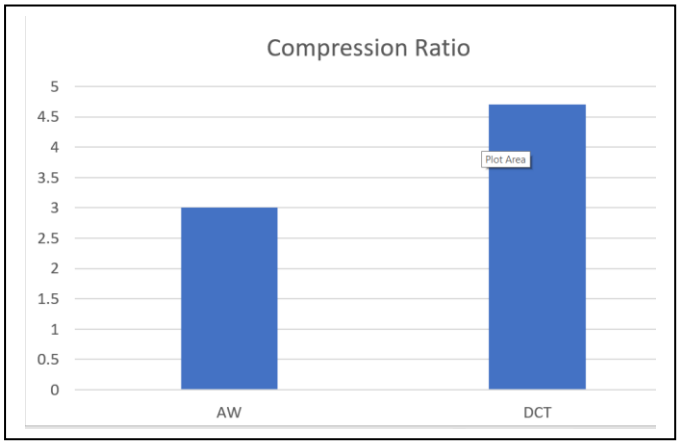
- delta = 15

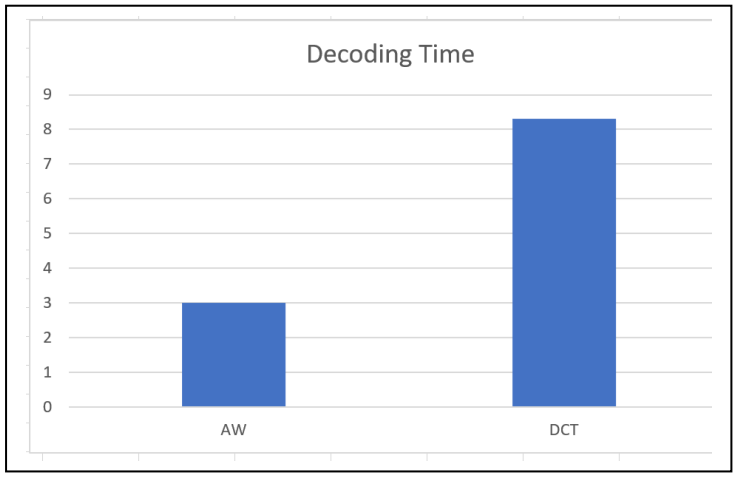
Original Image

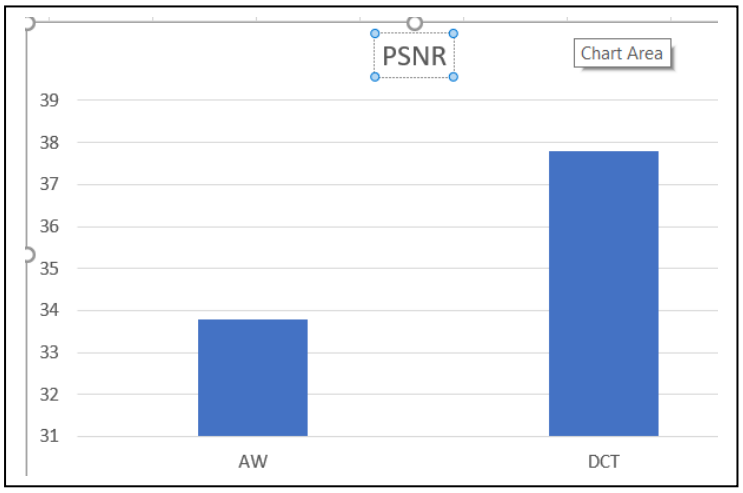
Decoded Image

	AW (mine)	**DCT**
Coding Time	4.7 Sec	170 Sec
Decoding Time	3 Sec	8.3 Sec
Compression Ratio	3	4.7
PSNR	33.8 dB	37.8 dB









4.3 EXPERIMENT 2

- delta = 30

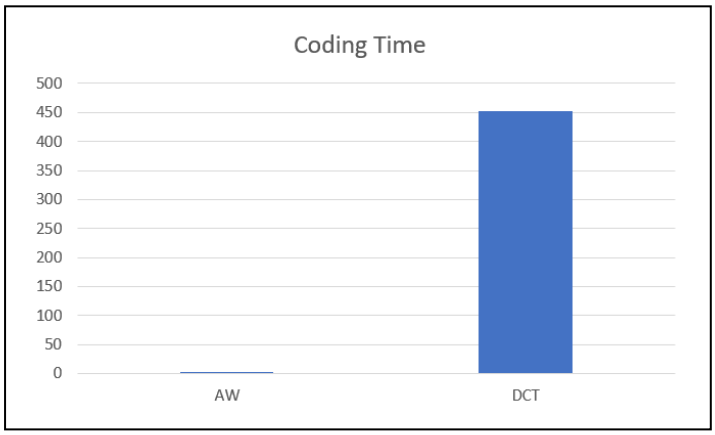
-Original size = 3.56 MB

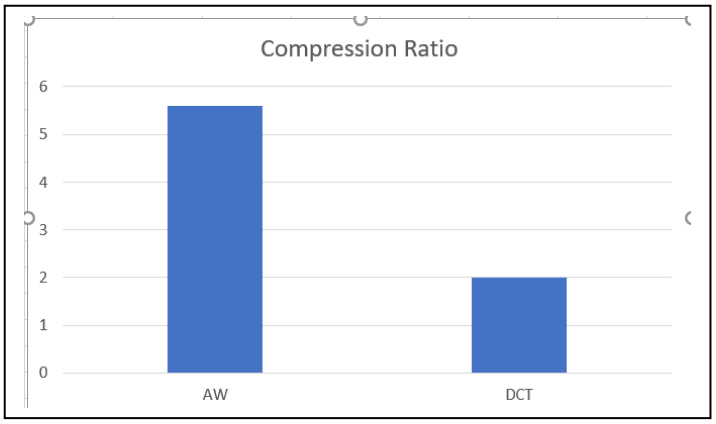
-Coded size = 637 KB

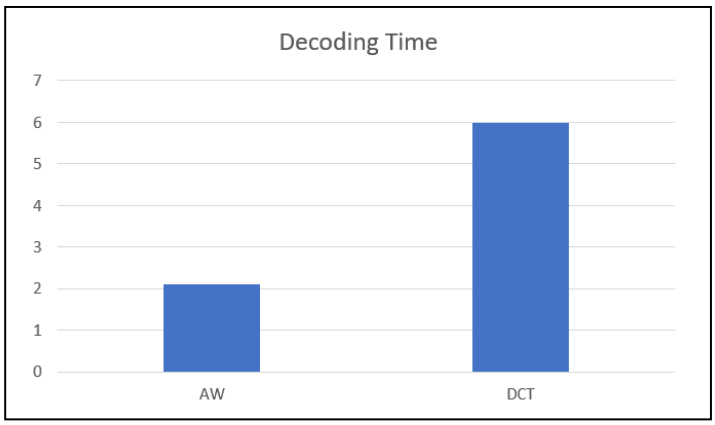
Original picture

Decoded Image

	AW (mine)	**DCT**
Coding Time	3.4 Sec	452 Sec
Decoding Time	2.1 Sec	6 Sec
Compression Ratio	5.6	2
PSNR	31.4 dB	28 dB


Coding Time
500
450
400
350
300
250
200
150
100
50
0
AW
DCT


Compression Ratio
6
5
4
3
2
1
0
AW
DCT


Decoding Time
7
6
5
4
3
2
1
0
AW
DCT

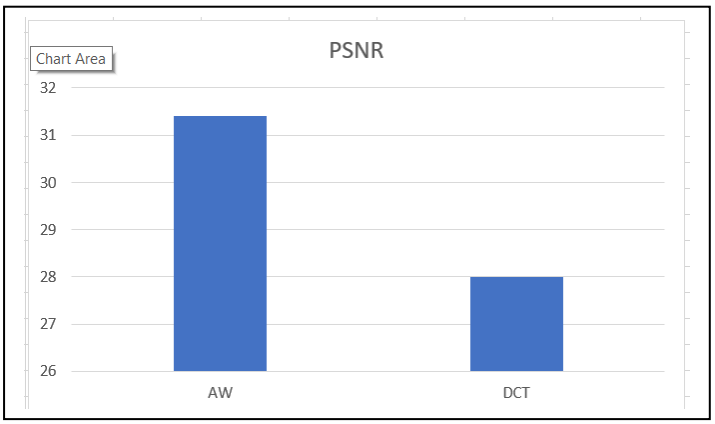


Chart Area
PSNR
32
31
30
29
28
27
26
AW
DCT