Data Structures and Algorithms Project

Illustration of effects of Quantizer and frequency coefficients cut-off in Lossy Image Compression

Presentation By: Bishal Khanal **st122221**

Image Compression Motivation

Size of image from a DSLR- 4898 × 3265, with 3 color channel, 8 bits per pixel (i.e. 256 quantization level) .

Total bits = 383,807,280

Total MB = 48 (nearly)

20 such Photos needs 1GB of storage.

In Next slide, I am going to show you two images, one is 9 MB size and other one is 1.7 MB size. I know you won't see the difference because of the much higher compression done by zoom app itself, but trust me, I don't see the difference too. But we can see the huge difference in the file size. The original raw size is around 30MB.


GG69
GG10
GG53
GG56
GG55


GG69
GG10
GG53
GG56
GG55

Image Compression

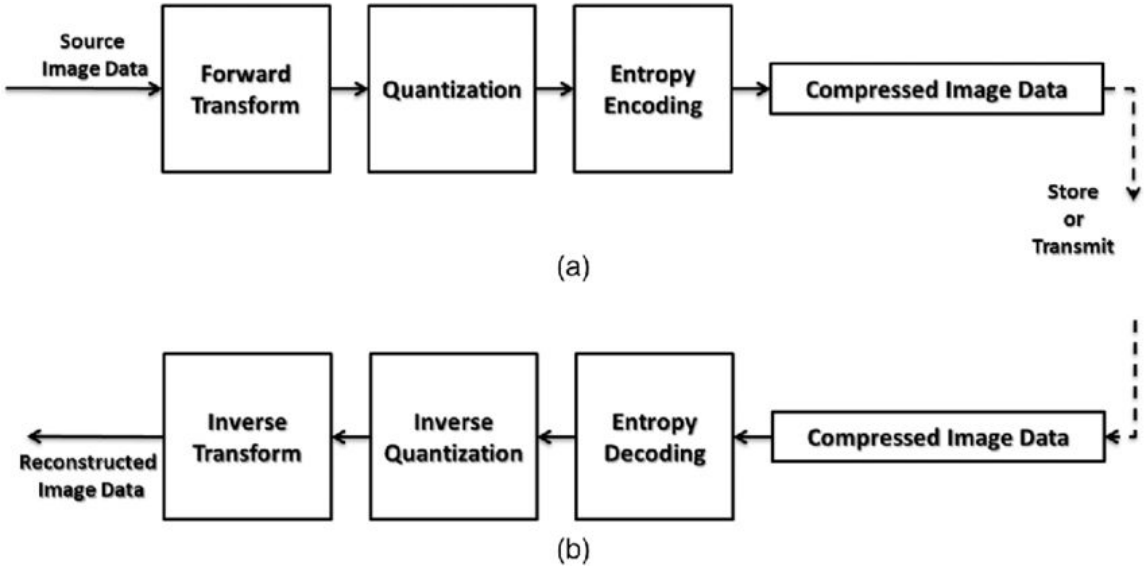
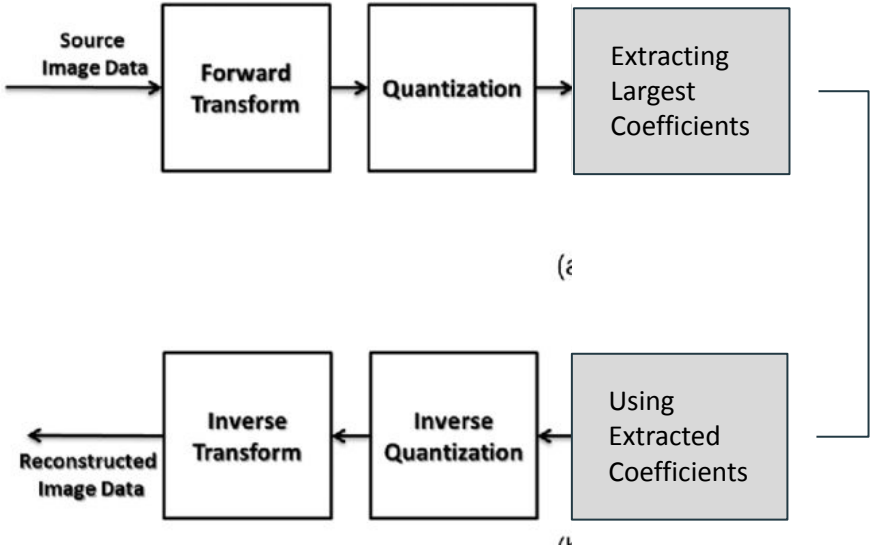


Image Compression (my project)



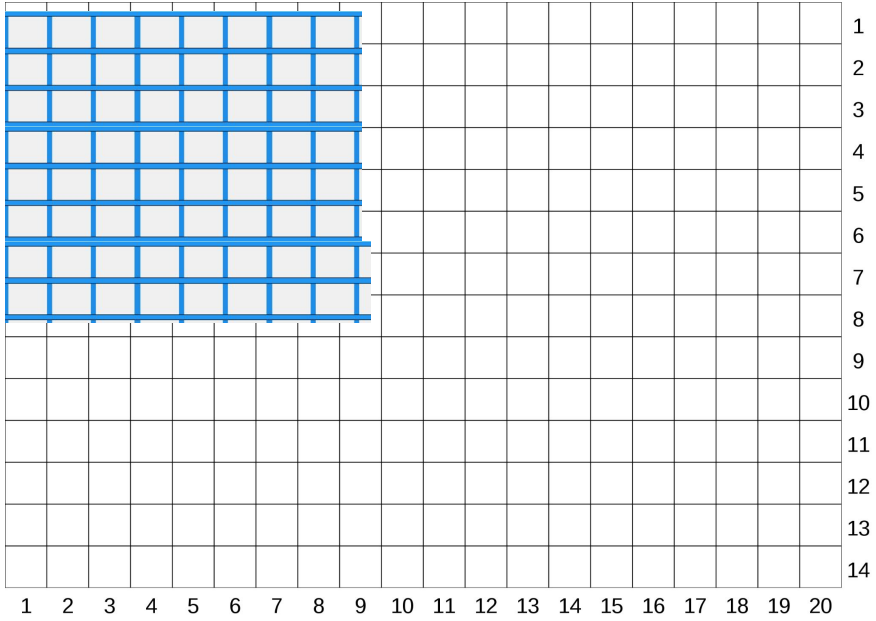
Since the main goal of this project is to show the effect of irrecoverable or lossy blocks effect that is used while compressing image file, I am going to skip the encoding block (lossless block).

Algorithm or Steps involved:

- Receive an input image
- Convert this RGB image into YCbCr image and take only Y for the purpose of demonstration.
  - YCbCr represents color as brightness and two color difference signals.
  - Y is brightness (luma), Cb and Cr provides color information.
- Divide the image into non-overlapping 8x8 blocks.
- Compute the DFT (discrete fourier transform) on each block.
- Quantize each block by dividing each coefficients by some number, round the result to the nearest integer, and multiply back by that same number. Try changing the value to see the effect.

Algorithm or Steps involved:

Dividing image
into 8 x 8 blocks



Algorithm or Steps involved:

- Try preserving the 8 largest coefficients (out of the total of 8x8=64). Or try decreasing and increasing the number of coefficients to preserve in order to see the effect.
    a. We can achieve high compression (lossy) in this block. Out of 64 coefficients, you can preserve only top 8 coefficients, achieving the maximum compression ratio. Meaning, if your image size is 8x8 you just can store at least 8 coefficients to reconstruct the image.
- In JPEG compression, we actually perform symbol encoding and save our file. But since we are not doing this for storage, we will now visualize the result.
- Perform the Inverse DFT.
- Visualize the results.

Performance Dictator

As from the steps in previous slide, the following two steps dictates the performance of the compressor:

- Discrete Fourier Transform Computation &
- Inverse Discrete Fourier Transform Computation

Since sorting is done only for 64 elements only whichever algorithm is okay (either stable or unstable).

Discrete Fourier Transform (1D)

The 1D DFT of a signal f(n) is calculated by following equation:

$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} f(n) \exp\left(-2\pi i \frac{nk}{N}\right)$$

Where,

f(n) is one dimensional digital signal

F(k) - Fourier Coefficients

N is size of f(n) and F(k) - same size

If implemented using the *for loop*:

- Computational Complexity = O(N^2)

```
F = 0
for k=0 to N-1:
        for n=0 to N-1
                F[k] = F[k] + f[n] * exp(-2.pi.i.nk/N)
```

Inverse Discrete Fourier Transform (1D)

The corresponding IDFT of the Fourier coefficients is calculated by following equation:

$$f(n) = \sum_{k=0}^{N-1} F(k) \exp\left(2\pi i \frac{nk}{N}\right)$$

Where,

F(k) - Fourier Coefficients

f(n) - Digital signal

As we can see, the computation complexity of DFT and IDFT of signal is the same.

DFT - 2D

Discrete Fourier Transform of 2D image.

$$F(k,l) = \Sigma_{i=0}^{M-1} \Sigma_{j=0}^{N-1} f(i,j) \exp^{-\mathbf{i}2\pi(\frac{ki}{M} + \frac{lj}{N})}$$

If implemented using the *for loop*:

- Computational Complexity = O(N^2 x M^2), My God!!

```
F = 0
for k=0 to M-1:
    for l=0 to N-1
        for i=0 to M-1
            for j=0 to N-1
                F[k,l] = F[k,l] + f[i,j] * exp(-2i.pi.(ik/M + lj/N)
```

Lets call this **Naive (implementation) dft**

IDFT - 2D

Corresponding Inverse Fourier transform of the 2D image signal:

$$f(a,b) = \frac{1}{N * M} \Sigma_{k=0}^{M-1} \Sigma_{l=0}^{N-1} F(k,l) \exp^{i2\pi(\frac{ka}{M} + \frac{lb}{N})}$$

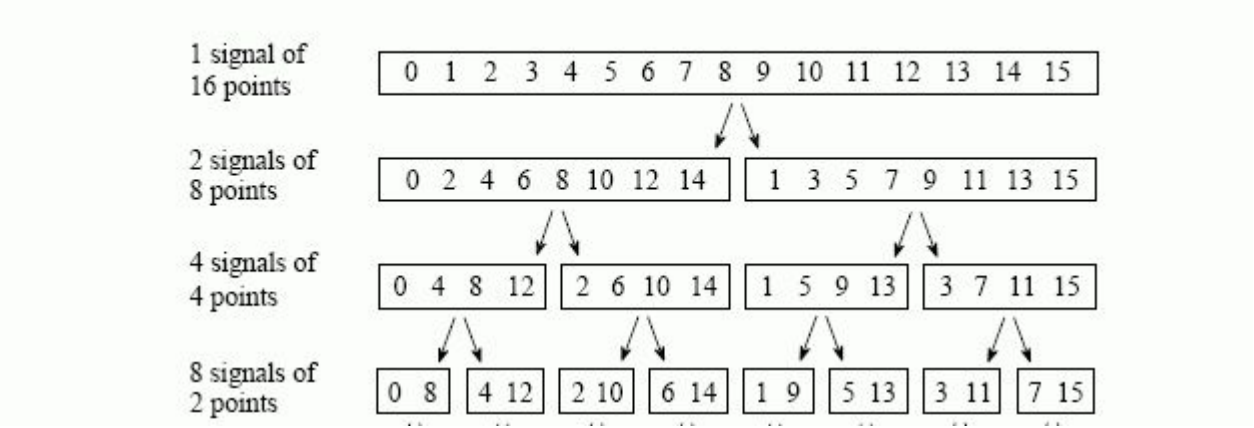
The Computational Complexity of idft is similar to the dft, i.e. O(M^2 x N^2). The Computational Efficiency of both dft and idft can be increased by using Matrix data structure to perform the computation. (Let's call this **smart dft**)

Fast Fourier Transform (FFT)

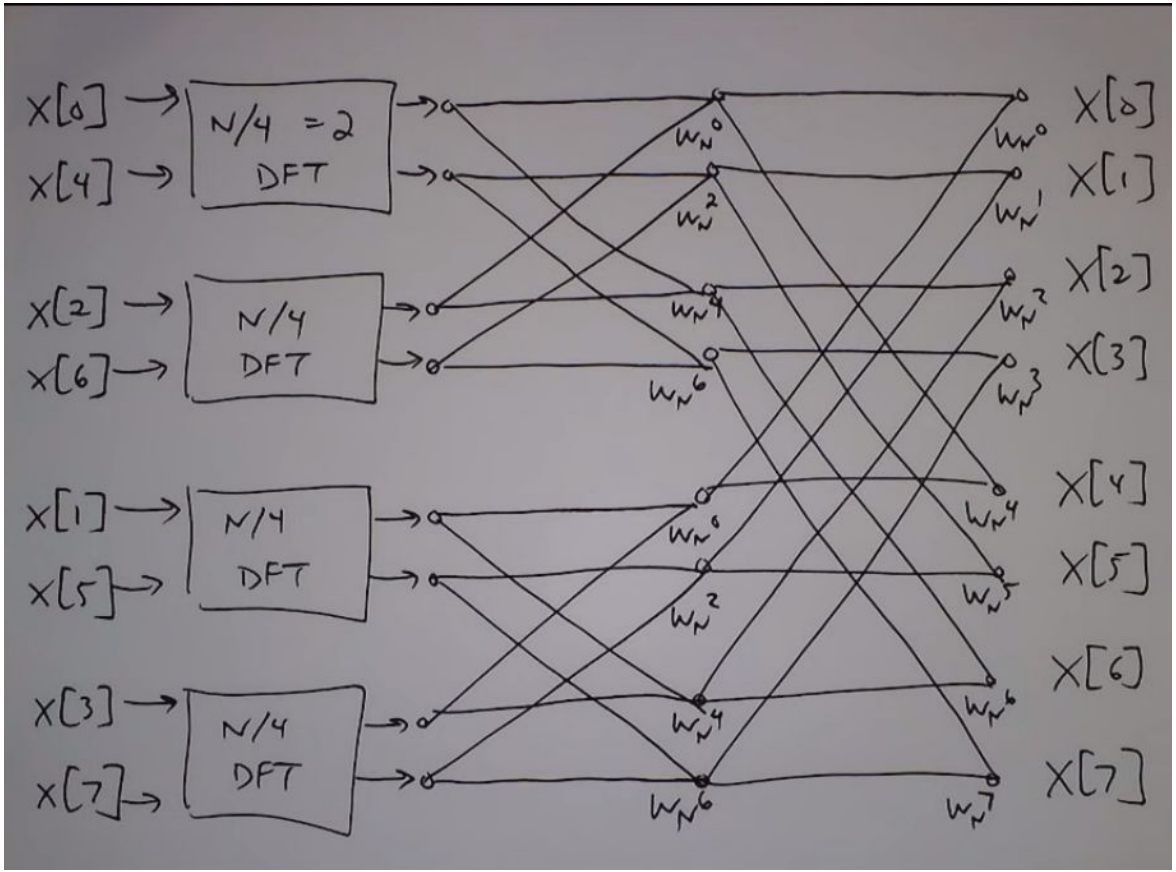
- Another trick to improve the computation complexity.
- This technique to compute DFT reduces the multiplication complexity to **N log (N)** for 1D signal.
- Based on Divide and Conquer Strategy

It computes DFT dividing N point DFTs to two N/2 point DFTs (even and odd samples).Similarly, again N/2 point DFTs is divided into two N/4 point DFTs (even and odd samples), until we reach 2 point DFTs and now we compute the 2 point DFT using the normal DFTs. This is the Fast Fourier Transform.

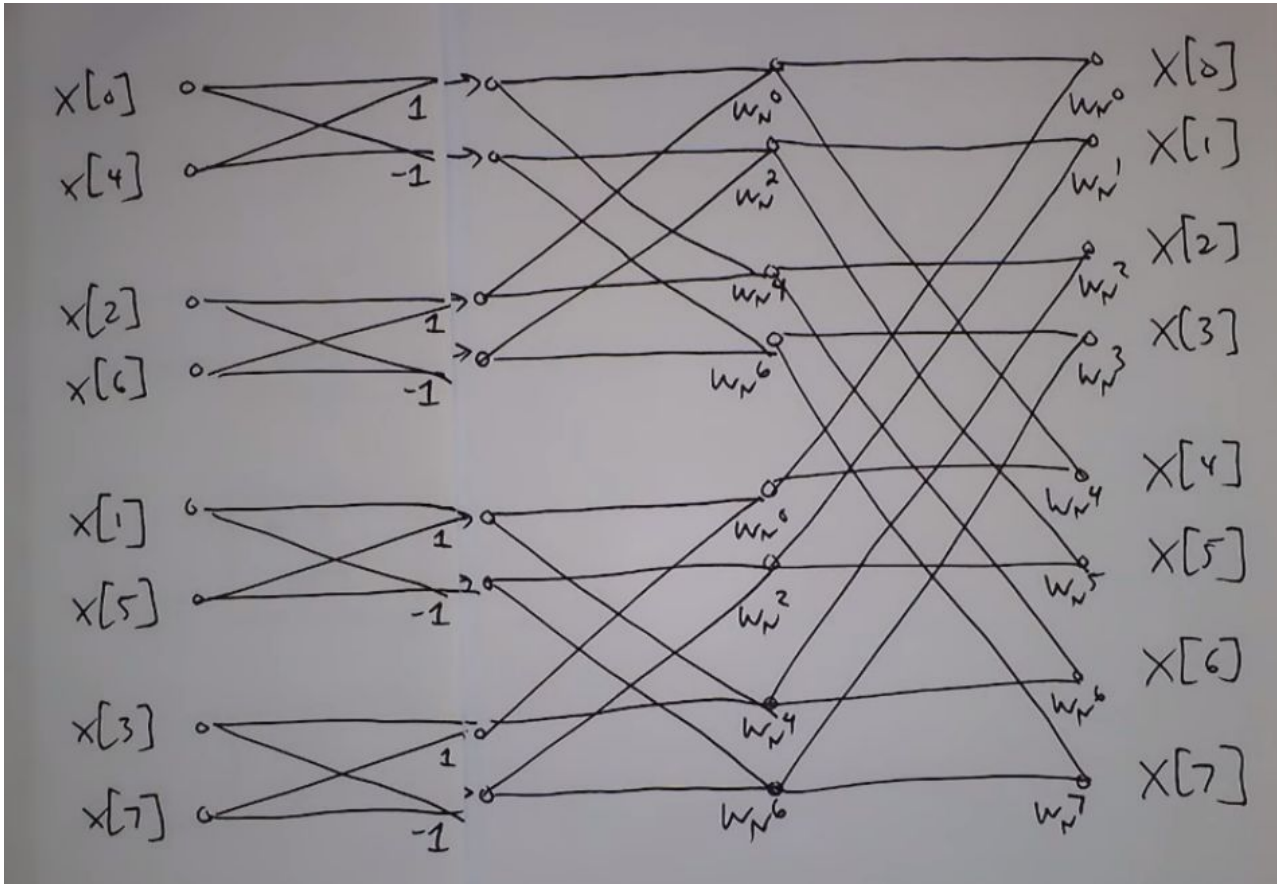
Fast Fourier Transform (FFT)



Source: https://www.dspguide.com/



Source: Prof. Rich Radke



Source: Prof. Rich Radke

Discrete Cosine Transform (DCT)

Its main plus point is it yields a real valued output image (unlike previous one that yields complex output) and that it is a fast transform and this is actually used in image compression.

$$C(k,n) = \alpha(k,n) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cos\left(\frac{(2i+1)k\pi}{2N}\right) \cos\left(\frac{(2j+1)n\pi}{2N}\right)$$

with

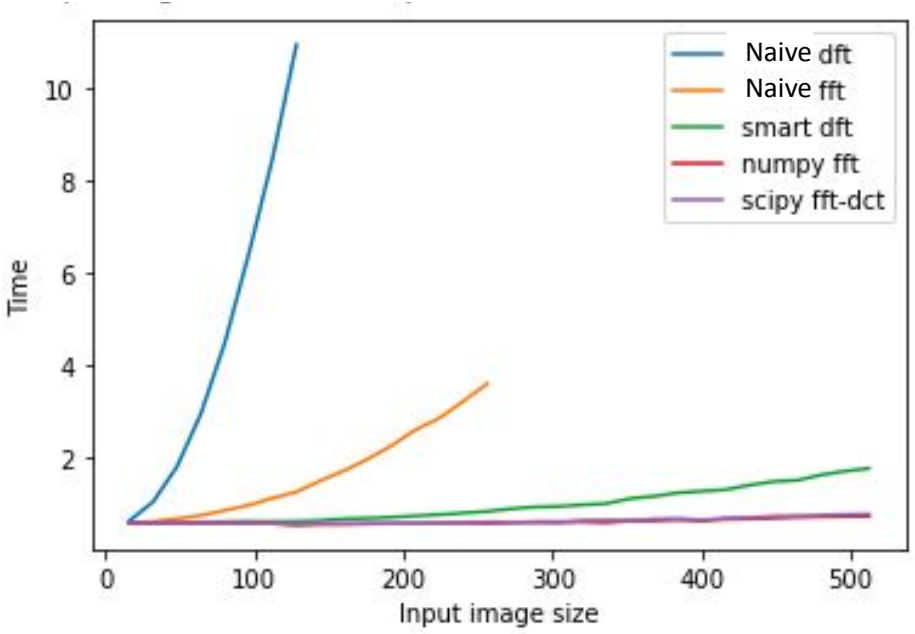
$$\alpha(k,n) = \begin{cases} \frac{1}{N} & \text{for } k,n = 0 \\ \frac{2}{N} & \text{for } k,n = 1,2,..,N\text{-}1 \end{cases}$$

Comparisons

The graph showing computational time -vs- input image dimension (Complexity graph).

The difference in slope for each case is because of the choice of:

- Data Structures and
- Algorithm



Limitations

- As we can see, the faster implementation to compute DFT is FFT, the complexity of which is Nlog(N) for a 1-D signal. Since it is used to compress images which, normally, is of dimension 3xNxN (one color and the other two spatial dimension), the complexity grows exponentially. Hence, a lot more efficient computations techniques and algorithms need to be developed to compute DFTs. There are some methods already developed to compute multidimensional DFTs and some are still in research.

Takeaway

- Creating some algorithm isn’t always enough (DFT in image processing)
- You need to develop efficient algorithms (FFT)
- Try to choose data structures which makes the computation faster in machine (Using vectors and arrays as data structures)

Naive dft - Just following algorithm without caring about data structures

Naive fft - Just following better version of the algorithm without exploiting advantages of data structures

Smart dft - Following slow version of the algorithm choosing better data structures

Numpy fft / scipy fft-dct - Following fast version of the algorithm choosing better data structures