RollNo:5117060

EXPERIMENT 05

Aim: To implement Fast Fourier Transform.

Theory:

If you have a background in electrical engineering, you will, in all probability, have heard of the Fourier Transform. In layman's terms, the Fourier Transform is a mathematical operation that changes the domain (x-axis) of a signal from time to frequency. The latter is particularly useful for decomposing a signal consisting of multiple pure frequencies. For more details have a look at the following video.
The application of the Fourier Transform isn't limited to digital signal processing. The Fourier Transform can, in fact, speed up the training process of convolutional neural networks. Recall how a convolutional layer overlays a kernel on a section of an image and performs bit-wise multiplication with all of the values at that location. The kernel is then shifted to another section of the image and the process is repeated until it has traversed the entire image.
The Fourier Transform can speed up convolutions by taking advantage of the following property. The above equation states that the convolution of two signals is equivalent to the multiplication of their Fourier transforms. Therefore, by transforming the input into frequency space, a convolution becomes a single element-wise multiplication. In other words, the input to a convolutional layer and kernel can be converted into frequencies using the Fourier Transform, **multiplied once** and then converted back using the inverse Fourier Transform. There is an overhead associated with transforming the inputs into the Fourier domain and the inverse Fourier Transform to get responses back to the spatial domain. However, this is offset by the speed up obtained from performing a single multiplication instead of having to multiply the kernel with different sections of the image.

Discrete Fourier Transform
The Discrete Fourier Transform (DTF) can be written as follows.

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

To determine the DTF of a discrete signal x[n] (where N is the size of its domain), we multiply each of its value by e raised to some function of n. We then sum the results obtained for a given n. If we used a computer to calculate the Discrete Fourier Transform of a signal, it would need to perform N (multiplications) x N (additions) = $O(N^2)$ operations.

As the name implies, the Fast Fourier Transform (FFT) is an algorithm that determines Discrete Fourier Transform of an input significantly faster than computing it directly. In computer science lingo, the FFT reduces the number of computations needed for a problem of size N from O(N^2) to O(NlogN).

Fast Fourier Transform Algorithm

Suppose, we separated the Fourier Transform into even and odd indexed sub-sequences

$$\begin{cases} n = 2r & \text{if } even \\ n = 2r+1 & \text{if } odd \end{cases}$$

$$\text{where} \quad r = 1, 2, ..., \frac{N}{2} - 1$$

After performing a bit of algebra, we end up with the summation of two terms. The advantage of this approach lies in the fact that the even and odd indexed sub-sequences can be computed concurrently.

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$

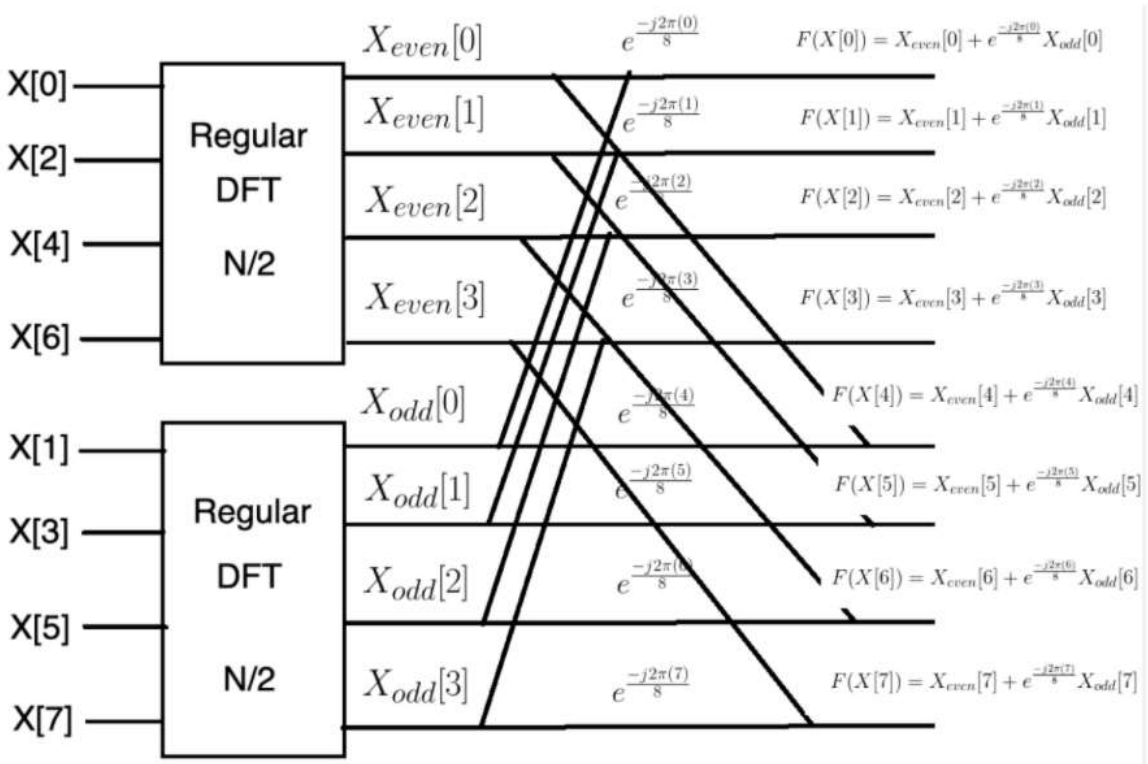
$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(2r)}{N}} \quad + \quad x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(2r+1)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(2r)}{N}} \quad + \quad x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(2r)}{N}}$$

$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r] e^{\frac{-j2\pi k(r)}{N/2}} \quad + \quad x[k] = e^{\frac{-j2\pi k}{N}} \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] e^{\frac{-j2\pi k(r)}{N/2}}$$

$$x[k] = x_{even}[k] \; + \; e^{\frac{-j2\pi k}{N}} x_{odd}[k]$$

Suppose, N = 8 , to visualize the flow of data with time, we can make use of a butterfly diagram. We compute the Discrete Fourier Transform for the even and odd terms simultaneously. Then, we calculate x[k] using the formula from above.



We can express the gains in terms of Big O Notation as follows. The first term comes from the fact that we compute the Discrete Fourier Transform twice. We multiply the latter by the time taken to compute the Discrete Fourier Transform on half the original input. In the final step, it takes N steps to add up the Fourier Transform for a particular k. We account for this by adding N to the final product.

DFT on N/2 elements

2 DFT

$$x[k] = x_{even}[k] + e^{\frac{-j2\pi k}{N}} x_{odd}[k]$$

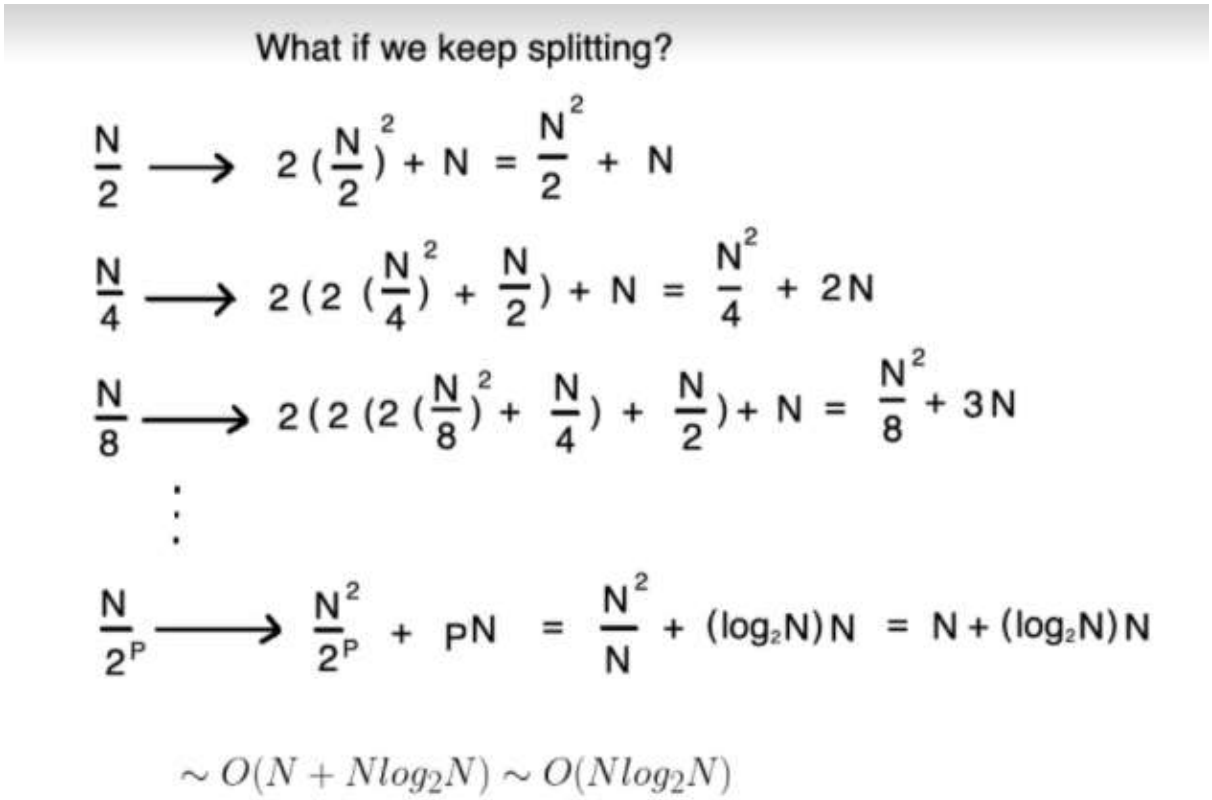
$$2 \times \left(\frac{N}{2}\right)^2 + N$$

$$= 2 \times \frac{N^2}{4} + N$$

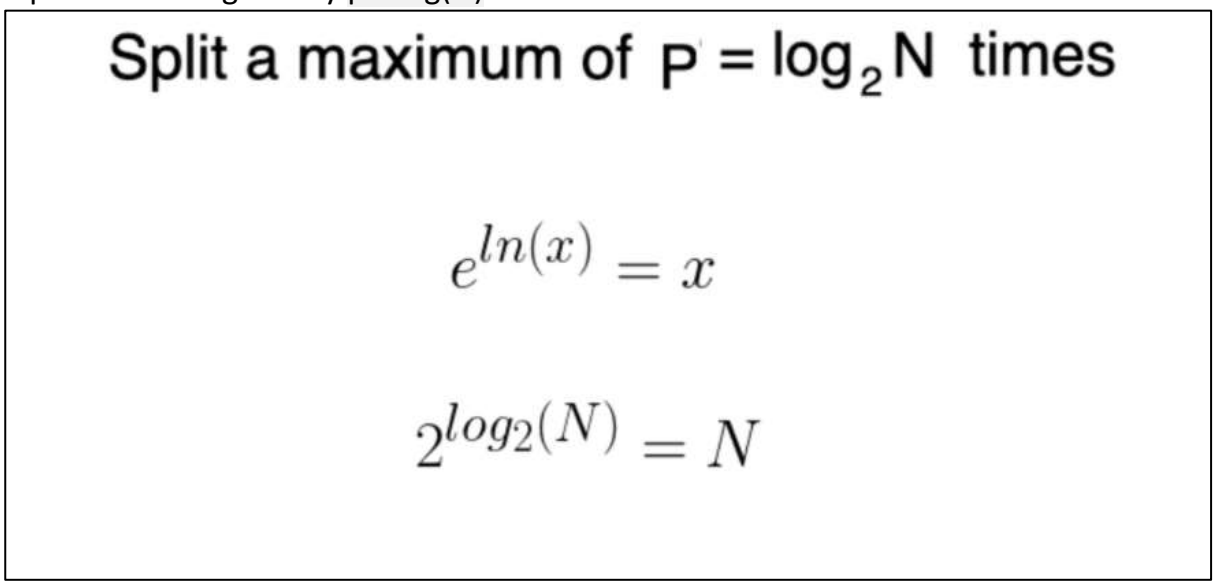
$$= \frac{N^2}{2} + N$$

$$O(\frac{N^2}{2} + N) \sim O(N^2)$$

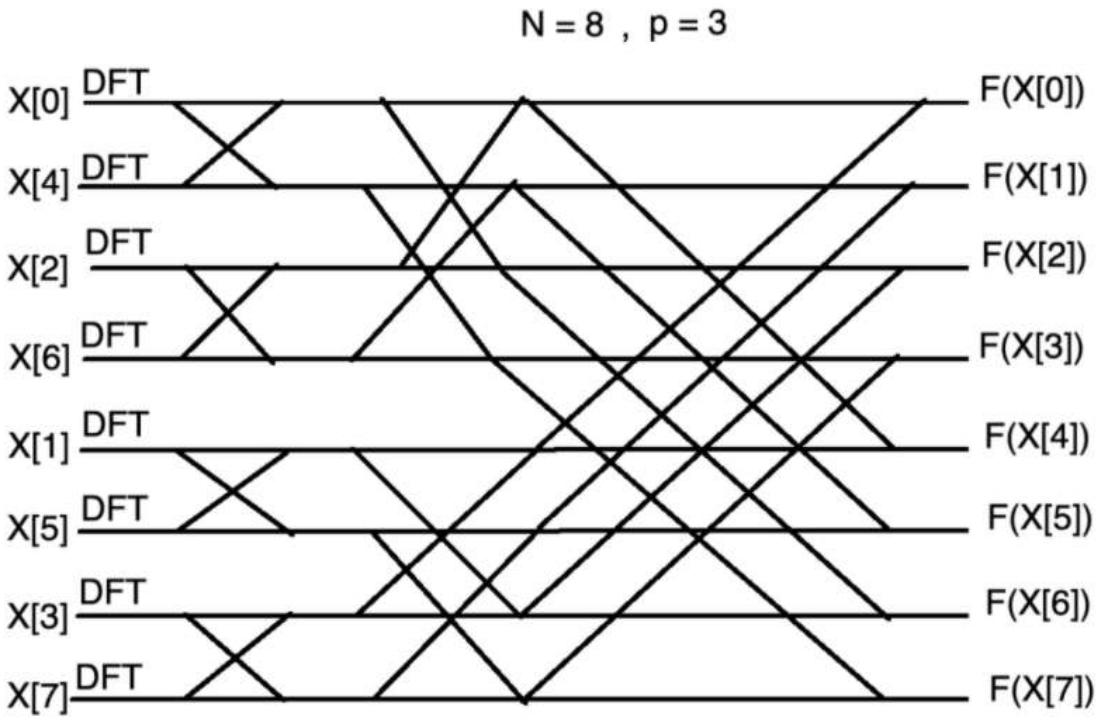
Notice how we were able to cut the time taken to compute the Fourier Transform by a factor of 2. We can further improve the algorithm by applying the divide-and-conquer approach, halving the computational cost each time. In other words, we can continue to split the problem size until we're left with groups of two and then directly compute the Discrete Fourier Transforms for each of those pairs.



So long as N is a power of 2, the maximum number of times you can split into two equal halves is given by p = log(N).



Here's what it would look like if we were to use the Fast Fourier Transform algorithm with a problem size of N = 8. Notice how we have p = log(8) = 3 stages.



Conclusion:
Thus we have successfully implemented Fast Fourier Transform

Code:

```
//Program to Compute the FFT of given Sequence
x[n]=[1,0,0,0,0,0,0,0].

clear;

clc ;

close ;

x = [1,0,0,0,0,0,0,0];

//FFT Computation

X = fft (x , -1);

disp(X,'X(z) = ');
```

Output:

```
X(z) =

    1.    1.    1.    1.    1.    1.    1.    1.
```