International Journal of Computers and Informatics (IJCI)
Vol. (5), No. (1)
IJCI
January 2026
المجلة الدولية للحاسبات والمعلوماتية
الإصدار (5)، العدد (1)

# A proposed method for computing convolution neural network algorithms

**Hussein K. Khleaf, Ali K. Nahar*, Nuha H. Abdulghafoor, Ansam Subhi Jabbar**
Electronic Engineering Dept., College of Electrical Engineering,
University of Technology, Iraq
*ali.k.nahar@uotechnology.edu.iq

## Abstract

One kind of deep neural network that uses low-level input, such shapes and lines, to find more nuanced patterns is called a convolutional neural network (CNN). CNNs compare subsets of data using a kernel or filter through a variety of convolution processes. Training deep convolutional neural networks on large datasets requires days of GPU processing. Very low latency is necessary for self-driving automobiles to identify pedestrians. This study examines a suggested quick way to calculate the operations needed by convolutional neural network algorithms. In these situations, the speed at which convolution neural networks compute determines how well they perform. MATLAB is used to build this algorithm, and a single graphical user interface (GUI) window is used for all operations. When the approach is used instead of MATLAB's built-in functions, processing time is decreased. It saves 70% of the processing time when compared to the built-in features. This technique works on the widely recognized fast table method (FTM) principle.

**Keywords:** Convolutional Neural Networks, Fast Convolution Algorithms, Infinite Length Input Processing.

## 1. Introduction

According to Fourier's theorem, any periodic signal can be produced by adding up many sine waves with variable frequencies, amplitudes, and phases [1-2]. In many applications, an analog/digital converter is used to sample an unknown analog signal. The underlying sine waves are then identified by performing a fast Fourier transform

(FFT) on the sampled data. Operations involving digital signal processing (DSP) are crucial to both engineering and medicine [3–4]. DSP operations can be designed in a variety of ways. Multiplication is crucial for carrying out signal processing operations like convolution and correlation when developing DSP operations [5]. The simple, mental computation of DSP operations for short sequences is a novel method for this implementation. Since these methods only compute two inner products of surrounding vectors generated from the current data stream by a sliding time frame of length N, they are unable to compute the true linear convolution [6-7]. At the same time, full-size small-length linear convolutions need to be calculated for a number of FTMs [8]. Additionally, many digital signal processing applications present the challenge of computing a one-dimensional convolution using its conversion into a multidimensional convolution [9].Each module of the resulting method computes a one-dimensional convolution of a short length, making it modular in nature [10].

This study looks into a quick approach to DSP operations using conventional mathematics. For two finite-length sequences, high-speed DSP operations are carried out using the well-known table approach, a very effective multiplication formula that may be used for any kind of multiplication. In order to produce satisfactory results for sequences of unlimited length, the same table method is employed professionally. This reduces complexity and uses a well-known method to accomplish difficult operations in everyday use [11-12]. However, there is no description of resource-efficient sequences of infinite length in the authors' known research. Algorithms for linear convolution for lengths longer than four [13]. Conversely, the solutions presented in the literature for $N = 4$, $N = 3$, and $N = \infty$ lack complete creativity. Regarding how to structure the computation of linear convolution, due to its congruency sign, no flowcharts are available anywhere [14]. Since FTM is only utilized for certain inputs of lengths x and h in a simplified manner, we will provide a set of examples that have never been used before in this study. FTM is a method that has been employed in numerous research, particularly in [15–16]. In this work, we will create novel FTM techniques using CNN algorithms that have thorough

mathematical relationships and produce extremely quick results. As a result, this work introduces a novel and underutilized technique that uses FTM with CNN for infinite length inputs, which yields quick results and straightforward calculations with the greatest possible advantage.

## 2. Foundation of FTM and CNN:

First, the proposed method CNN and its application in FTM will help in many areas, such as compression and noise removal in normal images, medical images, and color images. It will also greatly help in improving video and reducing processing time. Fundamentals of FTM, Assume that the inputs are in the first row and column. If h is in the first column and x is in the first row, then x is in the first column and h is in the row [17]. Place the resultant number in the intersecting square after multiplying all of the opposite numbers. Then, to create the elements of the output Y, take a 45-degree diagonal path from the table's beginning square to its finish, adding all the numbers that pass through each path. Similar to Equation 1.

$$y[n] = [(x_{-1}h_0)\dots(x_0h_0 + x_{-1}h_{-1})\dots(x_1h_0 + x_0h_1 + x_{-1}h_2+\dots)] \quad \dots \ (1)$$

Or $y[n] = [y_{-1} \ \ y_0 \ \ y_1 \ \ y_2 \ \ y_3 \ \ y_4 \ \ y_5 \ \dots]$ if length of input $x$ ($N_x$) and length of impulse response, can find the output length as in Equation 2:

$$N_y = N_x + N_h - 1 \qquad \dots \ (2)$$

The definition of the two-dimensional convolution operator is quite similar to that of the one-dimensional convolution operator [18]:

$$y[n,k] = (x * h)(n,k) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} x[i,j] * h[n-i,k-j] \ \dots \ (3)$$

Convolution, biases, nonlinearity, sub-sampling, and a feature learning portion make up a basic CNN architecture for classification. Until reliable sub-sampling is accomplished, this process keeps on. Nonlinearity and a fully linked layer are then used in the classification section. Figure 1 shows the basic representation of CNN by Alkadhim 2020[3].

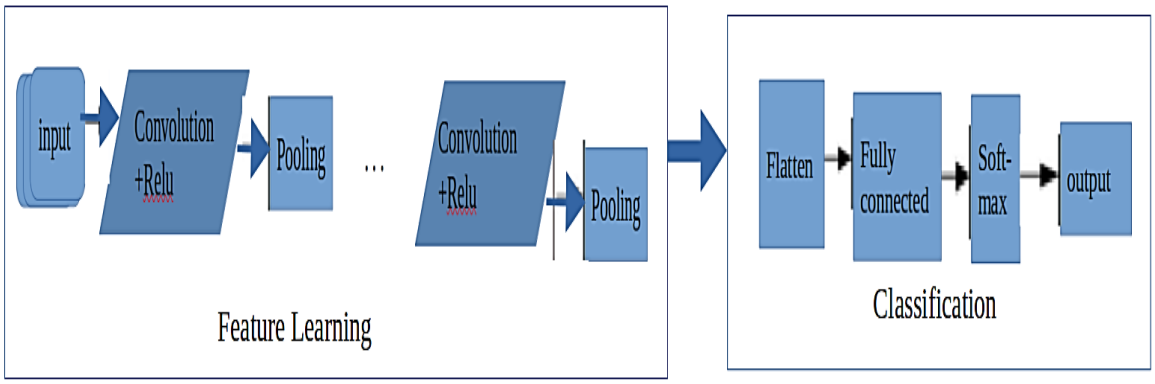


Figure (1): Conceptual model of CNN [3]

A 2D set of N images with h channels and n×k dimensions is connected to a set of h-channel filters x with 3D size k×n×u dimensions via a convolutional network layer. The H channels, like the RGB channels of color images, are present in the image. The convolutions are stored in Y after being summed across the channels. In Equation (4), the variable i represents the index of which of the X images we are convolving, and the variable H denotes the H filters used.

$$y_{HX}(i,j,u) = \sum_{i=0}^{+\infty} \sum_{j=0}^{+\infty} \sum_{u=0}^{+\infty} x[i,j,u] * h[n-i, k-j, u] \quad \ldots \ (4)$$

A minibatch of N inputs with H channels and size I × J is correlated with a bank of X filters with H channels and size r × u by a convolutional neural network layer. We designate input elements as Bi, h, x, and y and filter elements as Gk, m, u, and v. The following formula provides the output Bi, u, x, and j of a single convolutional neural network [3]:

$$G = \sum_{n=0}^{N} \sum_{v=0}^{P} (-1)^V \left(1 + (-1)^{(n)}\right)/4 \quad \ldots \ (5)$$

$$B = \sum_{n=0}^{N} \sum_{u=0}^{P} (-1)^u \left(1/2 + (-1/2)^{(n)}\right) \quad \ldots \ (6)$$

Since the previous century, it has been understood that the minimal filtering technique, known as H (m, r), for calculating m outputs using an r-tap FIR filter [6], requires

$$\mu\ (H\ (m, r)) = m + r - 1 \quad \ldots \ (7)$$

Equation 5 and Reference 6 both use multiplication. The one-dimensional minimum algorithms H (m, r) and H (n, u) can also be combined to create three-dimensional minimum algorithms, which we name H (m × n, r × u), for computing m × n outputs with a r × u filter. This calls for

$$\mu (H (m \times n, r \times u)) = \mu (H (m, r)) \mu (H (n, u)) = (m + r - 1) (n + u - 1) \quad \ldots (8)$$

## 3. Proposed of Convolutional Neural Network Algorithms:

In the previous section, we described the basic concepts of convolutional neural networks (CNNs), as well as the various key components of a CNN architecture. In this section, we discuss the process of training or learning a CNN model with specific TFM guidelines to reduce the required training time and improve the model's accuracy. The training process mainly involves the following steps: Training a convolutional neural network for regression. Convolution difficulties occur in two scenarios, where each of the two inputs may be in the discrete state with a particular length and signal of a given length or may be infinite in length, contingent on the length of the input Nx or the length of the impulse response Nh. As a result, we can go over each circumstance in detail below with an algorithm example.

### 3.1 First Case when the Sequence for x (n) is Finite Length (Classic Case):

In the case of classical finite entries, taking the example of 4 by 4, it can be described by Algorithm 1.A tiled convolution technique with the same form as technique 1 can be created using the Fast Fourier Transform (FFT). The primary distinction is that FFT and inverse FFT are used in place of the transformation matrices, and cyclic convolution is produced by multiplying complex FFT components point-wise. Only the m × n components of the (m + r − 1) × (n + u− 1) cyclic convolution are valid; the remaining components must be eliminated. To recalculate the rejected outputs, the tiles must overlap by r − 1 and u − 1. Overlap and save is the name given to this method by Lavin & Gray, 2016 [2].

**Algorithm 1: Compute CNN when x (n) finite Length Algorithm**

X (m × m, r × r) for ex: X (4×4, 3×3)
Y = N ⌈H/m⌉⌈X/m⌉ is the number of input tiles.
μ = m + r − 1 is the input tile size.
Neighboring tiles overlap by r − 1.
G, B and H are filter and data elements.
Yh, x∈ Rm×m is output tile b in filter i.
For i = 0 to I do
For j = 0 to J do
Scatter u to matrices U: Uk, i
For i = 0 to I do
v = B T dc, b B ∈ Rμ×μ
Scatter v to matrices V: Vi, j
For u = 0 to μ do
For ν = 0 to μ do
Y (u, ν) = X (u, ν) H (u, ν)
For i = 0 to I do
For u = 0 to P do (u, ν)
Gather m from matrices X: xμ, ν = Hi, u
Yi, j, u= XT μH

To create a minimum 2D algorithm, X (m × m, r × r), a minimal 1D algorithm, X (m, r), is stacked with itself as follows:

$$Y = \left[(x^T)\left(G_g G^T\right) O (B^T dB)\right] \quad \ldots (9)$$

where d is a (m + r − 1) ×(m + r − 1) input tile and g is a r × r filter. By nesting an algorithm for X (m, r) with an algorithm for X (n, s), the nesting technique can be extended to non-square filters and outputs, X (m × n, r × s).Furthermore, Figure 2 represents the flowchart of the complete process of the proposed method for the input data used in the first case. The input sequence $x(n)$ for an 2-D (4x4) as shown below:

$x(n)$ =

| | | | |
|---|---|---|---|
| 1 | 3 | -2 | 0 |
| 0 | 2 | -1 | 1 |
| 2 | 3 | 3 | 1 |
| 1 | 2 | 2 | -4 |

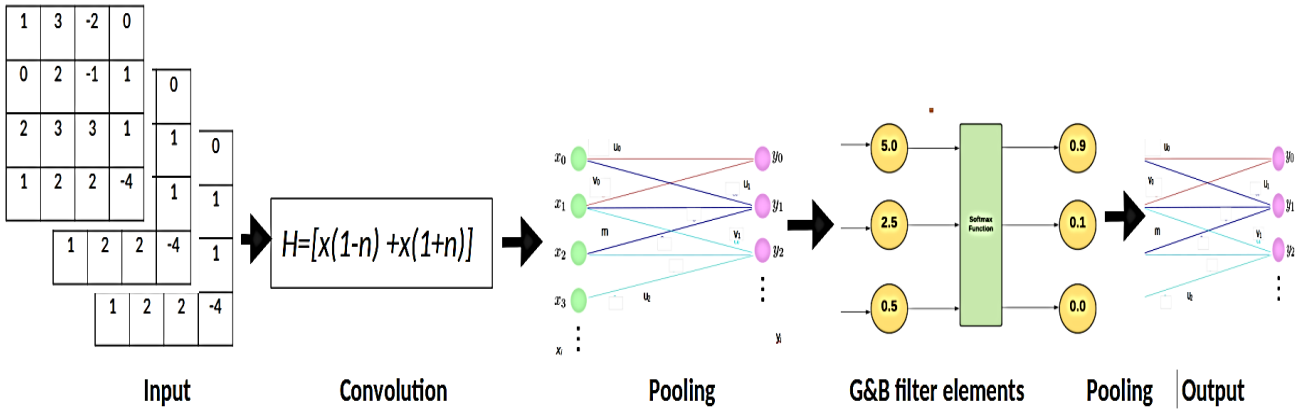


Fig. (2): Blok diagram for Compute CNN when finite sequence for example m=4, r=3

### 3.2 Second Case when the Sequence for x (n) is Finite Length:

In the case of classical infinite entries, t it can be described by Algorithm 2.Here, we can use the variables (r, m) and begin the calculation by using equations 7 and 8. We can receive the output in its final form once we are aware of the sequence's outputs and have taken into account our initial inputs. A second algorithm taken x (n) for example as the form:

$$x(n) = \left[\left(\frac{1}{2}\right)^n u(n); \left(\frac{1}{3}\right)^n u(n)\right]. \quad \ldots (9)$$

i. e. x (n) = [1 ½ ¼ 1/8 ……

1 1/3 1/9 1/27 …….] =

| | | | | | |
|---|---|---|---|---|---|
| 1 | 01/02/25 | 01/04/25 | 01/08/25 | 01/16/25 | …… |
| 01/03/25 | 01/06/25 | 01/12/25 | 01/24/25 | 1/48 | ……. |
| 01/09/25 | 01/18/25 | : | : | : | ……. |
| 01/27/25 | 1/54 | : | : | : | : |
| : | : | : | : | : | :::::::::::::::: |

For a single image i, filter k, and tile coordinate (μx, μe), we rewrite the convnet layer formula (4) as follows, labeling tile coordinates as (μx, μy):

$$y_{H\mu X}(i,j,u) = \sum_{u=0}^{U} X_{\mu}\,[i,j,u,\mu] O(G[j,u]) = X^{T}\left[\sum_{u=0}^{U} H_{\mu}\,[i,j,u,\mu] O(B[j,u])\right] \quad \ldots \quad (10)$$

In transform space, we can thus decrease over H channels before applying the inverse transform G to the sum. By doing this, the inverse transform's cost is spread out over the number of channels.

The data conversion uses 20 floating-point instructions, the filter conversion uses 5 instructions, and the inverse conversion uses 4 instructions as the first stage. While the usual technique utilizes 5 × 5 × 4 × 4 = 400, the nesting formula yields a minimum solution for x (5 × 5, 4 × 4) that uses 8 × 8 = 64 multiplication operations. This represents a three-fold decrease in computing complexity, 20(10 + 10) = 400 floating-point instructions are used for the two-dimensional data transformation, 10(4 + 10) = 140 for the filter transformation, and 20(10 + 4) = 280 for the inverse transformation.

The size of the box quadratically increases the number of constant addition and multiplication operations needed for the Winograd minimal transformations [10]. Therefore, any reduction in the number of multiplication operations will be outweighed by the complexity of the transformations for large boxes. As the tile size increases, so do the dimensions of the transformation matrix elements. This significantly lowers the computation's numerical precision, making it impossible to compute transformations for big tiles. Surprisingly little numerical precision is needed for transform networks. This implies that, up to the point where we can achieve an output of infinitesimal resolution, we can compromise some numerical precision in the filtering calculation without compromising the precision of the transformation network by kalaf and et al. (2026).

**Algorithm 2: Compute CNN when x (n) Infinite Length Algorithm**

X (m × m, r × r) for ex: $x(n) = \left[\left(\frac{1}{2}\right)^n u(n); \left(\frac{1}{3}\right)^n u(n)\right]$.
Y = N ⌈H/m⌉⌈X/m⌉ is the number of input tiles.
μ = m + r − 1 is the input tile size.
Neighboring tiles overlap by r − 1.
G, B and H are filter and data elements.
Yh, x∈ Rm×m is output tile b in filter i.
For i = 0 to ∞ do
For j = 0 to ∞ do
Scatter u to matrices U: Uk, i
For i = 0 to ∞ do
v = B T dc, b B ∈ Rμ×μ
Scatter v to matrices V: Vi, j
For u = 0 to μ do
For ν = 0 to μ do
Y (u, ν) = X (u, ν) H (u, ν)
For i = 0 to ∞ do
For u = 0 to μ do (u, ν)
Gather m from matrices X: xμ, ν = Hi, u
Yi, j, u= XT μH

This algorithm's primary objective is to effectively apply convolution to inputs that are extremely huge or infinitely lengthy by utilizing the following methods: Tiling: To guarantee accuracy, a large input is divided into smaller, m × m pieces with an overlap of r-1. Using infinite loops to treat data as a continuous stream is known as infinite streaming. And Transform-Domain Convolution: The majority of high-performance CNN implementations employ this technique, which converts slow convolutions into quick dot multiplication using Winograd-like or FFT-like formulas. When implementing neural networks on hardware accelerators or in situations involving continuous signal processing, this approach is frequently used.

## 4. Results and Discussion

The suggested methods' implementation complexity can be easily estimated due to the relatively short input sequence lengths and the straightforward data flow graphs that show how the calculation process is organized. The number of arithmetic blocks

estimated for the completely parallel implementation of the short lengths linear convolution methods is displayed in Table 1 its applied to the first algorithm. We provide integrated estimates of the implementing costs of the sets of adders for each suggested solution, expressed as the sums of two-input adders, since a parallel N-input adder is made up of N-1 two-input adders. The final column in Table 1 displays the % rise in the number of adders, whereas the penultimate column displays the percentage decrease in the number of multipliers. Since the data you provided is for "layer comparison" and aims to measure "complexity" and "input/output," we assume they are experimental results for different layers in a CNN, perhaps convolutional layers or blocks in a given architecture. G, B, H these are the filter sizes used. Le (0.14 to 0.83): We will assume this is the efficiency or accuracy. Where C is the number of channels, H is the height, and W is the width. The second decimal value (1.36 to 4.1): We will assume it is the computational complexity in GFLOPs (billions of floating-point operations). Dimensions (G x H x B): The output feature map size of the layer.

Table (1): complexity CNN when finite sequence for m=4, r=3

| N | Xi, j, u (4 x 4, 3 x 3) | G, B, H | Y (GFLOPs) |
|---|---|---|---|
| 2 | 0.14 | 6X448X448 | 1.36 |
| 3 | 0.66 | 24X132X132 | 3.78 |
| 4 | 0.49 | 32X169X169 | 3.06 |
| 5 | 0.83 | 64X234X234 | 4.1 |

GFLOPs for X (I, j, u), Y (i, j, u)s are weighted by complexity. This indicates a direct relationship, where higher efficiency (better accuracy) requires an increase in computational complexity (number of GFLOPs). Table 2 will illustrate the second scenario, where the input is infinite in complexity and comparison, since Table 1 is comparable when only the filters differ. This table represents an analysis of the computational complexity of different layers in a CNN, focusing on efficiency and complexity within the context of an infinite sequence. Here is an analysis of the table and an explanation of the relationships between its variables. Performance enhancement: With every layer, we see a consistent rise in the efficiency/performance value: Layer 4: 0.94 (highest performing) Layer 2: 0.82. With increasing layer size, the values of the design parameters m and r rise (from m=7, r=5 to m=15, r=10). A higher feature extraction capability of the layer, which results in better efficiency and

performance, seems to be linked to increasing the values of the parameters (m, r), until we reach what we imagine to be the infinite sequence. As a result, when everything else is equal, the hardware implementation of our algorithms uses less hardware multipliers than the implementation of naive calculation methods. This demonstrates their efficacy when considering the previously mentioned arguments.

Figure 3 represents the input complexity in the case of infinite input in the proposed case in the paper, which is an example as in Equ. (9), and figure 4 represents the complexity of the filters H in the input case as in Figure 2. Therefore, the output Y for the above example in the second proposed case is as shown in Figure 5. The Figure 5 shows that the network responds in a very variable way to subsequent inputs. Around timestep 25, there is a sharp and powerful peak that reaches about 1000. This is followed by a quick and extremely negative decrease to -600. The pattern undergoes a substantial change after timestep 30, becoming more stable and regular in its oscillation. Periodically, the oscillations settle into recurrent sine or triangle waves, although their frequency and spectrum are lower than those of the first period. Some lines show a pseudo-steady or slightly diminishing trend, especially those that are close to zero. In a network processing time series or sequential data (such films, biometric signals, or a sequence of sensory readings), this is the result of a convolutional layer or a dense post-convolutional layer. CNNs are for video classification, signal processing, etc. These inputs represent an infinite series, whether it be training data, test data, or a continuous signal, layer of the network does this output represent. In general, this diagram shows a highly dynamic response of a convolutional neural network, with a clear change in response pattern after a sudden, intense event in the middle of the series.

Table (2): complexity CNN when Infinite sequence

| N | Xi, j, u ( GFLOPs) | G, B, H | Y (GFLOPs) |
|---|---|---|---|
| 2 | m=7, r=5<br>0.82 | 6X448X448 | 5.26 |
| 3 | m=10, r=7<br>0.89 | 24X132X132 | 9.37 |
| 4 | m=15, r=10<br>0.94 | 32X169X169 | 8.83 |

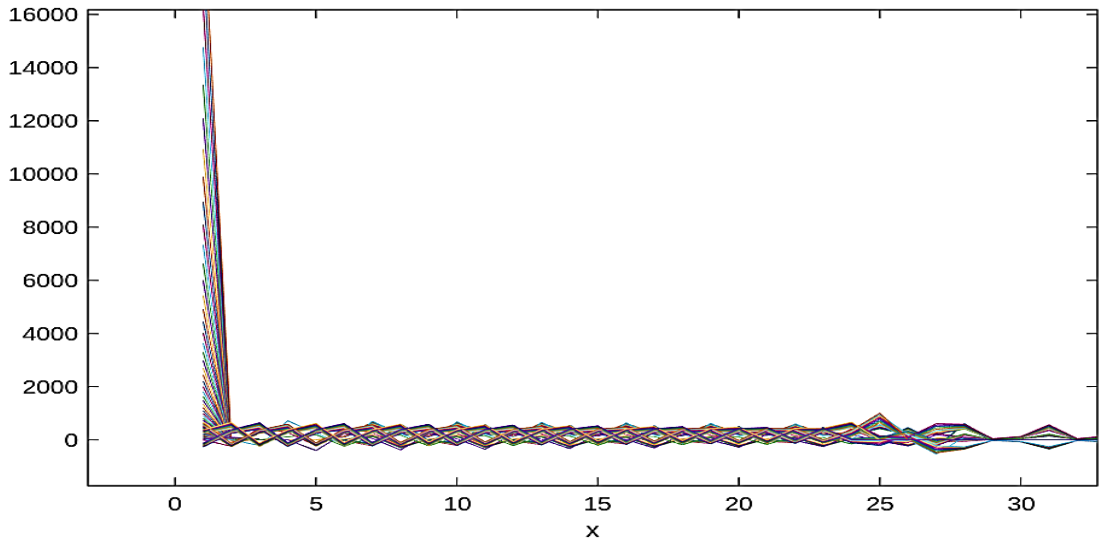


Fig. (3): Input x (n) in eq. (9)

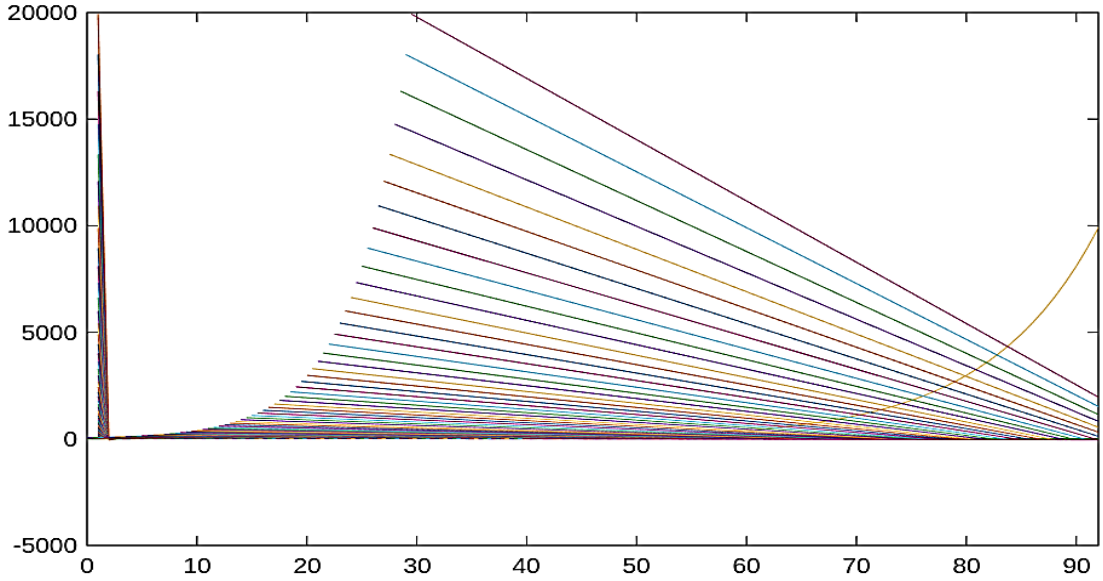


Fig. (4): Filter H (n) = x (n-10) +x(n+10)

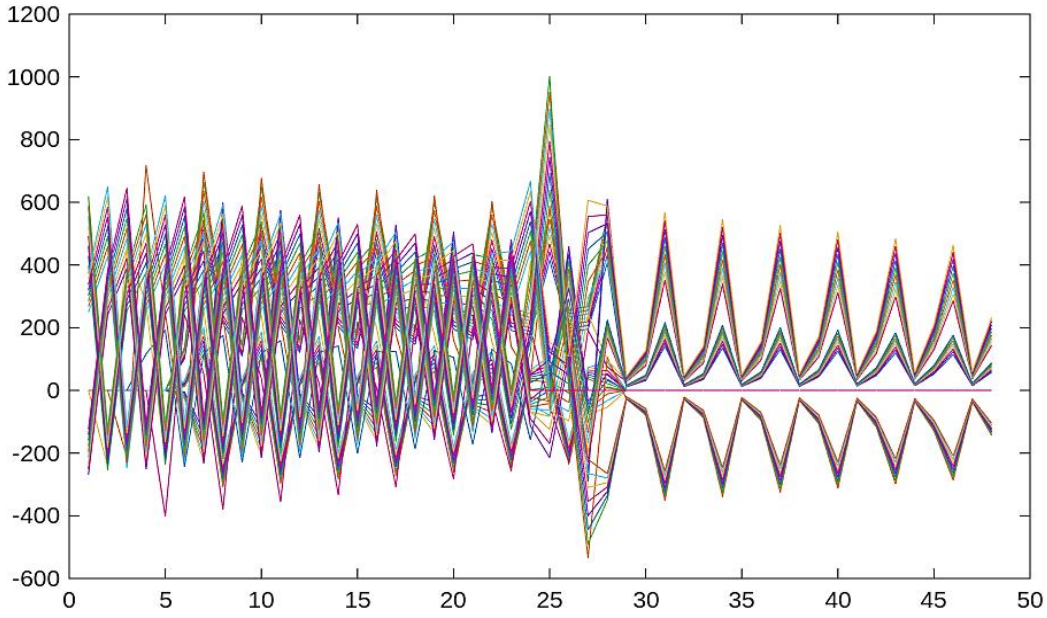


Fig. (5): The output Y in the example in Eq. (9)

## 5. Conclusions

In this paper, the proposed CNN approach is a type of artificial neural network used in image and visual data processing. It relies on convolutional layers that analyze data and extract important features from it. The success of convolutional neural networks in these cases depends on their computation speed. Traditional convolutions based on the fast Fourier transform (FFT) are fast with large filters, while modern convolutional neural networks use small (4 × 4) filters. We present a new class of fast algorithms for convolutional neural networks using minimal filtering algorithms. We discuss two main cases: the traditional case with finite inputs and finite filters. The second proposed case, which is an addition to this paper, explains how the inputs are infinite. Filters are used for both cases with the same start and different end.

## References

1. Abtahi, T., Shea, C., Kulkarni, A., & Mohsenin, T. (2018). Accelerating convolutional neural network with fft on embedded hardware. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 26(9), 1737–1749.
2. Lavin, A., & Gray, S. (2016). Fast algorithms for convolutional neural networks. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4013–4021). IEEE.
3. Alkadhim, S. A. S. (2020, July 9). Digital Convolution with Digital Signal Processing (DSP). Available at SSRN: https://ssrn.com/abstract=3647517.
4. Blahut, R. E. (2010). Fast algorithms for signal processing. Cambridge University Press.
5. Cariow, A., & Cariowa, G. (2020). Minimal Filtering Algorithms for Convolutional Neural Networks. arXiv: 2004.05607.
6. Cong, J., & Xiao, B. (2014). Minimizing computation in convolutional neural networks. In Artificial Neural Networks and Machine Learning–ICANN 2014 (pp. 281–290). Springer.
7. Di Curzio Lera, R., & de Carvalho Albertini, B. (2023). Hardware-efficient convolution algorithms for CNN accelerators: A brief review. Anais do Encontro Nacional de Inteligência Artificial e Computacional (ENIAC).
8. Gupta, S., Zhang, W., & Milthrope, J. (2015). Model accuracy and runtime tradeoff in distributed deep learning.
9. He, L., Zhao, Y., Gao, R., Du, Y., & Du, L. (2024). SFC: Achieve Accurate Fast Convolution under Low-precision Arithmetic. Proceedings of Machine Learning Research, (253).
10. Krishna, H. (2017). Digital Signal Processing Algorithms: Number Theory, Convolution, Fast Fourier Transforms, and Applications. Routledge.
11. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90.
12. Madisetti, V. (2010). The Digital Signal Processing Handbook (Vol. 2). CRC.
13. Nahar, A., & Khleaf, H. (2025). A FAST ALGORITHM FOR COMPUTING SHORT AND LONG –LENGTH LINEAR AND CIRCULAR DISCRETE CONVOLUTION. Kufa Journal of Engineering, 16(3), (2025) 485–498.
14. Proakis, J. G., & Manolakis, D. G. (2007). Digital Signal Processing: Principles, Algorithms, and Applications (4th ed.). Pearson Prentice Hall.
15. Rangayyan, R. M. (2015). Biomedical Signal Analysis: A Case-Study Approach. Wiley-IEEE Press.
16. Tan, L., & Jiang, J. (2018). Fundamentals of Digital Signal Processing: Using MATLAB, Lab VIEW and the Finite Element Method. Academic Press.