University of Dublin

TRINITY COLLEGE

Sparse Data Structures and Algorithms for Convolution Layers in Deep Neural Networks

Sudhansh Mehta
B.A.I Computer Engineering
Final Year Project April 2019
Supervisor: Dr.David Gregg

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

_____

Sudhansh Mehta

Permission to Lend

I agree that the library and other agents of the College may lend or copy this thesis upon request.

_____

Sudhansh Mehta

Acknowledgements

I would like to thank my supervisor Dr.David Gregg for the enormous wealth of advice and encouragement he has given me throughout the project.

Finally I would like to thank my friends and family for all of their help and support during this project.

Contents

1 Abstract

Convolution Neural Networks(CNN's) deliver state of the art results in Image Processing and Video Processing leading to diverse applications in numerous fields however training of these networks is a resource intensive process.This work tries to solve this problem by creating Novel Sparse Data Structures which utilize the inherent Sparse Nature of the Kernel Matrix(collection of 2-Dimensional filters).15 of the possible 24 different variations of the Data structure created are studied and their time and space performance for performing Sparse Convolution is analyzed on the basis of the two design parameters of the Sparse Convolution System (M(output channels) and D(input channels)).The study proves that depending on the magnitude of D and M certain arrangements of sparse and dense dimensions can be picked for Sparse Convolution as they give better performance on space and time.As Time and Space are the two main resource considerations of Neural Networks picking the right arrangements of sparse and dense dimensions for the Novel Data Structure created for performing Sparse Convolution will aid the process of training of resource-intensive Neural Networks.

2 Motivation

Convolution Neural Networks(CNN's) are inspired by biological processes in living beings and the layout of the Convolution Neural Networks(CNN's) resembles the layout of visual cortex in animals. Convolution Neural Networks(CNN's) deliver state of the art results in Image Processing and Video Processing leading to diverse applications in fields such as medical image analysis(in Radiology), predicting the steering angle in self-driving cars (Autonomous Driving Vehicles) to name a few. The major task/resource intensive task is the training of these Networks which require GPU(Graphical Processing Unit) farms for training the networks. This resource intensive training process is one of the major reasons that some applications of the Convolutional Neural Networks are yet untapped such as in the field of embedded systems which face the problems of limited computing resources and energy supply. The problem of resource intensive computation can be dealt with and the training process can be optimized as

- The Training Process tries to optimize the weights of the Convolutional Neural Networks i.e the weights connecting the neurons and thus minimize the error derivative of the output with respect to the weights.
- During the training process of the Convolutional Neural Network dropouts and regularization techniques are applied to tune the training process and avoid the over-fitting of the data during training process. The regularization techniques(like Tikhonov regularization) and dropout techniques achieve this by pulling the weights down to zero.
- Thus many filter weights are pulled down to zero during a pass of the training process (do not contribute to the training process).This knowledge can be utilized to store only those filter weights which contribute to the training process.
- This leads to a Sparse nature of the Kernel Matrix as not all weights associated with filters are significant.

This work is focused on creating Novel Sparse Data Structures which utilize the Sparse Nature of the Kernel Matrix(a collection of 2-Dimensional filters). The 4-Dimensional Kernel matrix of filters is made to go through a Sparsify Algorithm (Sparsify_4d function) which creates a Data Structure storing only the dense values from the 4-Dimensional Kernel Matrix of filters.This data structure is then made to convolve over the Input Channels of the Input 3-Dimensional matrix giving the Output 3-Dimensional matrix.There are 24 permutations/ways(4*3*2*1 for number of permutations possible without replacement) in which the 4-Dimensional Matrix can be traversed. Exploiting this information(based on the ways in which the 4-Dimensional Matrix can be traversed) the study focuses on evaluating the performance of 15 of the total 24 possible traversals for performing Sparse Convolution and a detailed analysis of factors affecting Time and Space Requirement is presented.

3 Background and Literature Review

3.1 Multi-Channel Convolution

Convolution serves as a feature detector in images and the Kernel's(filters) define the type of feature that is desired to be extracted. When the kernels(filters) convolve over the Input images(Input 3-Dimensional Matrix) they try to look for those features in different positions in the Input Channels of the Input 3-Dimensional Matrix.Tasks as complex as image classification, object detection and segmentation need to be looking at more than one feature at each layer of the network.The richer the feature/ kernels(filters) are the greater is the information extracted from the Input Images. Hence State of the art Convolutional Neural Networks like LeNet, AlexNet, VGG, GoogLeNet, ResNet have a very rich and deep Kernel 4-Dimensional Matrix of filters or feature detectors. The Figure 3.1 below is a graphical representation of the Input 3-Dimensional matrix, Kernel 4-Dimensional matrix and the Output 3-Dimensional matrix with values of design parameters of the complete system

- Depth(D) of Input channels (Depth of channels from the Input 3-Dimensional Matrix(W*H*D) and also serves as one of the dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D)) = 3
- M (One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which also determines the number of channels in the Output 3-Dimensional matrix(W*H*M)) = 4

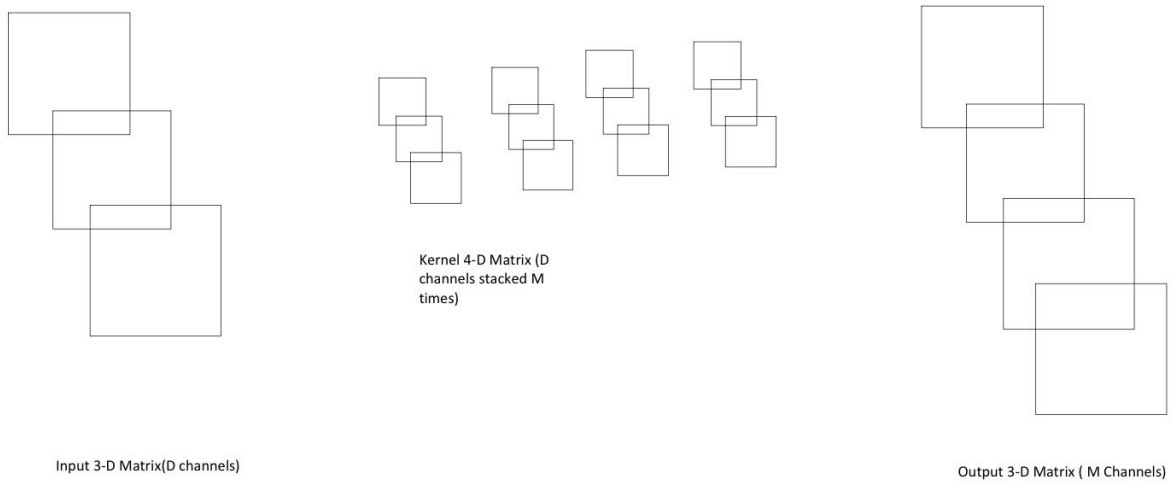

Figure 3.1: Multi Channel Convolution

3.2 Kernel 4D Matrix

The 4-Dimensional Kernel matrix(which stores the weights of the Neural Network) is the focus of the study as the 4-Dimensional Kernel Matrix is the only Sparse part(which can be sparsified i.e compressed) in the system(consisting of Input 3-Dimensional Matrix, Kernel 4-Dimensional Matrix and Output 3-Dimensional Matrix).Hence for speeding up the otherwise dense convolution(where even the values of weights which have turned to zero are stored in the Dense Kernel Matrix) different Sparse Data Structures are created for the Kernel matrix which store the Non-Zero values of Weights only. The Figure 3.2(Accelerator-Aware Pruning for Convolutional Neural Networks,Hyeong-Ju Kang,2018)) [1] below shows a dense kernel 4-Dimensional matrix with C as the 3rd Dimension(2-Dimensional Matrix stacked along the 3rd Dimension) and the whole structure thus formed is replicated and stacked in the 4th Dimension(4th dimension being M).

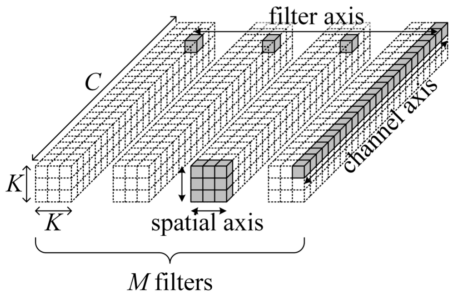

Figure 3.2: Kernel 4D Matrix(Hyeong-Ju Kang, IEEE,2018) [1]

3.3 Need for Randomly Generating Kernel 4-Dimensional Matrix

There are 24 permutations/ways(4*3*2*1 for number of permutations possible without replacement) in which the 4-Dimensional Matrix can be traversed. Exploiting this information(based on the ways in which the 4-Dimensional Matrix can be traversed) the study focuses on evaluating the performance of 15 of the total 24 possible traversals for performing sparse convolution.To compare the different orders/traversals and their performance the Kernel 4-Dimensional Matrix has to be randomly generated so that there is no bias and all the possible 24 traversals of the 4-Dimensional Matrix are equally dense(or equally sparse).

The different possible arrangements/order of traversal of the 4-Dimensional matrix are listed below

- M,D,X,Y
- D,M,X,Y
- X,M,D,Y
- M,X,D,Y
- D,X,M,Y
- X,D,M,Y
- X,D,Y,M
- D,X,Y,M
- Y,X,D,M
- X,Y,D,M
- D,Y,X,M
- Y,D,X,M
- Y,M,X,D
- M,Y,X,D
- X,Y,M,D
- Y,X,M,D
- M,X,Y,D
- X,M,Y,D
- D,M,Y,X
- M,D,Y,X
- Y,D,M,X
- D,Y,M,X
- M,Y,D,X
- Y,M,D,X

3.4 Sparse Matrix Representations

3.4.1 List of Lists

List of Lists is an algorithm which stores one list per row of the 2-Dimensional filter, with each entry containing the column index(in the 2-Dimensional filter) and the non-zero(dense) value. From an implementation point of view generally these entries are kept sorted by column index(in the 2-Dimensional filter) for faster look-up. A list of pointers to the heads of the individual lists is maintained hence the name List of Lists.The Figure 3.3 below is a graphical representation of the Data Structure for List of Lists.

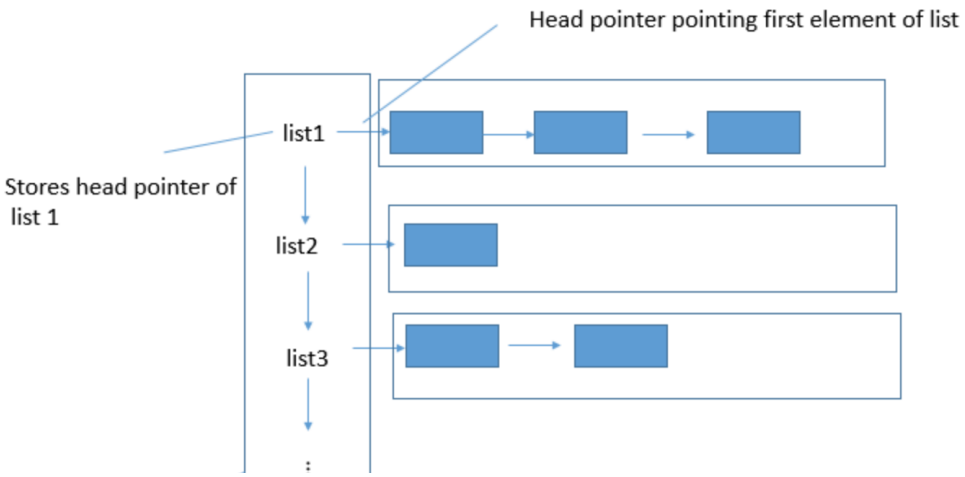

Figure 3.3: List of Lists

3.4.2 Dictionary of Keys(DOK)

Dictionary of Keys(DOK) utilizes dictionary(Hash-tables) as a Data Structure mapping (row,column) pairs to the value of the non-zero elements(dense values stored). The Data Structure Dictionary of Keys(DOK) is good for incrementally constructing a sparse matrix in random order.However Hash-tables do not preserve ordering be it natural ordering or order of insertion and performing a sort at time of iteration can be costly.

3.4.3 Compressed Sparse Row(CSR) and Compressed Sparse Column(CSC) Technique

Compressed Sparse Row (CSR) technique is used to compress a sparsely filled 2-Dimensional matrix (filter) into a set of three 1-Dimensional lists (containing only dense values from the 2-Dimensional matrix(filter)):

- A: Contains all the dense values from the 2-Dimensional matrix(filter)
- JA: Contains the column index for each of the dense values in the A list
- IA: Contains the information for number of elements in each row of the 2-Dimensional matrix(filter)

Alternatively Compressed Sparse Column Technique can be used which like the Compressed Sparse Row(CSR) technique is also a set of three 1-Dimensional lists (containing only dense values from the 2-Dimensional matrix(filter)) but with the modification:

- A: Contains all the dense values from the 2-Dimensional matrix(filter)
- JA: Contains the row index for each of the dense values in the A list
- IA: Contains the information for number of elements in each column of the 2-Dimensional matrix(filter)

From an implementation point of view a array of three pointers to the heads of the three lists is maintained in case of CSR(Compressed Sparse Row) or CSC(Compressed Sparse Column).The Figure 3.4 below is a graphical representation of the Data Structure for CSR(Compressed Sparse Row) or CSC(Compressed Sparse Column) algorithm.

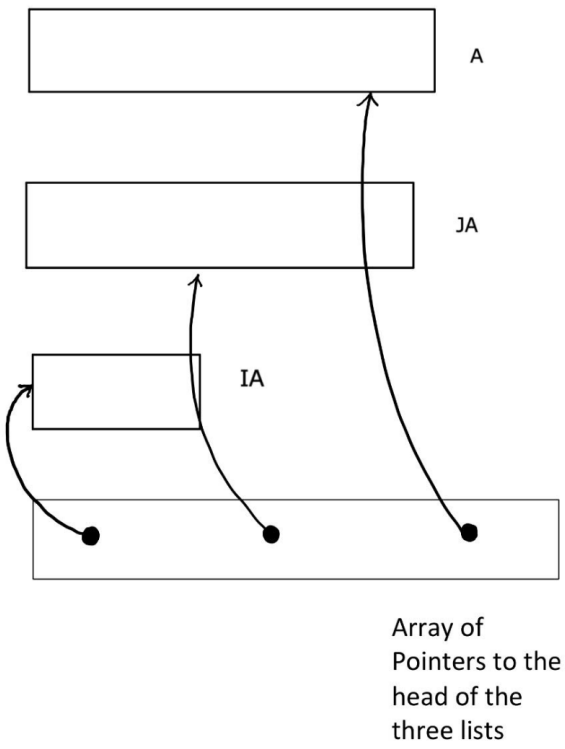

Figure 3.4: Data Structure for CSR or CSC algorithm

3.5 Related Work

- Faster CNN's With Direct Sparse Convolutions And Guided Pruning (Jongsoo Park,Sheng Li,Wei Wen,Ping Tak Peter Tang,Hai Li,Yiran Chen and Pradeep Dubey at ICLR 2017) [2]:- talks about pruning of the layers of the network as an approach to speed the process of Convolution and the paper provides important insights for training/pruning and talks about how sparse convolution is not useful for certain layers, in which case skipping pruning of those layers can provide more room for sparsity in the other layers.

- Pruning Filters for efficient Convnets (Hao Li,Asim Kadav,Igor Durdanovic,Hanan Samet, Hans Peter Graf at ICLR 2017) [3]:- talks about an acceleration method for Convolutional Neural Networks where they recommend pruning whole filters with their connecting feature maps from Convolutional Neural Networks(CNN's) that are identified as having a small effect on the output accuracy which leads to the computation costs being reduced significantly.

- Hybrid Pruning: Thinner Sparse Networks for Fast Inference on Edge Devices (Xiaofan Xu, Mi Sun Park, Cormac Brick ,Movidius, AIPG, Intel,2018) [4] :- talks about applying a combination of Coarse Grained Pruning (pruning whole filters) with their connecting feature maps from CNNs and Fine Grained Pruning (pruning only certain values in a filter) termed as Hybrid Pruning to speed the process of Convolution.The paper also introduces a sensitivity test for pruning channels revolving around the idea that about 3-5 percent loss in accuracy is acceptable as it can be recovered by fine-tuning of the network however damaging the network beyond that may lead to introducing Sparsity at the expense of accuracy of the network.

- Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (Song Han,Huizi Mao, William J. Dally at ICLR 2016) [5] :- The work introduces a 3-Stage pipeline for compressing the Neural Network. The 3-Stage compression pipeline being composed of

  - Pruning the small weight connections (Network Pruning)
  - Quantization Process (storing less bits per weight)
  - Huffman Encoding Algorithm for final stage of the pipeline and providing further data compression

4 Randomly Generated Sparse Kernel 4-Dimensional Matrix

The 4-Dimensional Matrix can be thought of as an array of 2-Dimensional Matrices stacked in the third dimension and the whole Data Structure then stacked in the 4th Dimension. So the 2-Dimensional Matrix can be thought of as the atomic element of the kernel 4-Dimensional Matrix.To Traverse a 2-Dimensional Array there is a need for two indexes(one referring to rows and one referring to columns)i.e array[row][column].The implementation uses two Randomly Generated Index Arrays (Arrays containing Randomly Generated Indexes) for :

- First Array(list) contains randomly generated indexes for randomly indexing into the rows of the 2-Dimensional array
- Second Array(list) contains randomly generated indexes for randomly indexing into the columns of the 2-Dimensional array

Hashmaps(unordered_maps in C++ implementation) are used to make sure that the random generator does not repeat indices in the either of the row indices or the column indices arrays. The randomly generated Row Indices array is then matched against the randomly generated column indexes array to get the complete address into the 2-Dimensional array.

4.1 Drawback of the Implementation

This drawback is due to the fact that the Randomly Generated Row Indexes array manages to get a random offset into the 2-Dimensional Array however when it has to iterate and match against the randomly generated column indices ,each index from the row indexes array has to match against all the entries in the generated column indices array which leads to a clustering problem. The Figure 4.1 below demonstrates the clustering problem for a sample input.

```
 Enter Width of channels: 7

 Enter Depth of channels: 10

 Enter Height of channels: 7

 ENTER THE NUMBER OF KERNEL MATRIX (M) : 1

 ENTER THE SIZE OF A FILTER (K): 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 20 14 8 20 10 13 13 7 14 8 16 18 11 0 0 17 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 19 3 12 6 13 4 9 0 0 0 0 0 0 0 17 2 7 5 18 19 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0
0 0 0 0 0 0 0 4 6 4 1 7 10 7 16 19 20 13 15 7 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 6 11 14 3 8 18 16 0 0 0 0 0 0 10 16 12 17 14 13 19 0 0 0 0 0 0 0
4 12 12 16 20 10 15 0 0 0 0 0 0 0 0 0 0 0 0 10 0 1 4 10 18 1 6 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 17 18 16 14 19 17 18 7 15 12 19 18 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 4 20 18 8 8 16 0 0 0 0 0 0 3 0 0 0 0 0 0 0 19 14 4 4 6 9 20
0 0 0 17 0 0 0 12 1 12 6 19 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 19 5 1 5 5 17
18 9 15 6 2 3 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20 8 15 11 1 12 8
0 0 0 0 0 0 0 0 0 12 0 0 0 0 0 0 0 0 0 0 0 2 9 6 17 12 12 12 0 0 0 0 0 0 0 11 14 12 1 1 19 8 0 0 0 0 0 0 0
```

Figure 4.1: Randomly Generated Kernel 4D Matrix Clustering Problem

5 Randomly Generated Sparse Kernel 4-Dimensional Matrix(with Uniform Distribution)

The problem of clustering is dealt with an alternate approach where instead of traversing the 4-Dimensional Matrix(considering it visually a list of 2-Dimensional arrays stacked in the third and the fourth dimensional) , the 4-Dimensional matrix is traversed as a list of 1-Dimensional arrays(4-Dimensional matrix thought of as 1-Dimensional arrays stacked in the 2nd dimension to give a 2-Dimensional array and then stacking the 2-Dimensional array in the 3rd and the 4th Dimension to give a 4-Dimensional kernel matrix). Sparsifying the 1-Dimensional matrix with a certain sparse percentage/proportion of the width of the 1-Dimensional list(array) gives a Uniform distribution of Dense elements throughout the whole 4-Dimensional Kernel Matrix. Figure 5.1 demonstrates a Uniform(Uniformly distributed dense values) 4-Dimensional Kernel matrix.

```
Enter Width of channels: 5

Enter Depth of channels: 10

Enter Height of channels: 5

ENTER THE NUMBER OF KERNEL MATRIX (M) : 10

ENTER THE SIZE OF A FILTER (K): 7
4 0 10 0 0 0 0 10 20 0 0 0 0 0 5 0 0 0 17 0 0 12 0 0 0 0 0 5 0 0 0 12 9 0 0 3 0 0 0 0 0 7 0 0 0 20 0 16 0
0 0 0 5 11 0 0 0 0 0 19 0 0 16 9 0 0 0 16 0 0 0 4 0 0 10 0 0 5 0 0 9 0 0 0 19 0 0 0 0 20 0 0 17 0 6 0 0 0
0 12 0 0 0 18 0 0 16 0 0 18 0 0 0 0 0 8 20 0 0 1 0 18 0 0 0 0 17 0 0 0 0 0 5 0 0 0 0 6 0 1 0 0 0 2 0 11 0
0 0 0 0 0 6 15 0 17 0 0 0 0 12 0 8 16 0 0 0 0 0 0 0 0 1 2 0 0 0 0 7 17 0 0 0 0 0 1 0 8 0 0 11 13 0 0 0 0
0 8 0 0 0 0 17 0 17 0 6 0 0 0 0 0 0 0 18 0 15 9 20 0 0 0 0 0 4 0 0 11 0 0 0 0 12 0 0 0 12 0 11 0 0 5 0 0 0
0 0 11 2 0 0 0 0 11 0 0 0 0 20 0 0 15 0 0 0 5 0 0 0 4 0 2 0 0 0 17 0 15 0 0 0 20 0 14 0 0 0 0 0 0 0 8 17 0
0 0 0 18 4 0 0 0 3 0 5 0 0 0 0 17 0 0 0 20 0 0 0 0 7 0 4 0 0 0 16 0 0 0 2 0 0 4 0 0 0 0 13 0 0 0 1 0 0 12
19 0 0 0 0 1 0 0 2 0 0 0 0 20 10 0 0 0 0 11 0 0 0 0 0 10 11 0 0 12 0 0 0 0 2 12 0 0 4 0 0 0 0 0 0 0 1 9 0
0 0 0 0 0 17 15 0 0 11 0 0 14 0 0 0 0 10 14 0 0 0 14 0 0 0 0 7 0 17 9 0 0 0 0 7 0 16 0 0 0 0 0 0 0 16 12 0 0
0 0 5 16 0 0 0 0 0 0 6 0 8 0 0 0 0 0 0 6 0 8 0 20 7 0 0 0 0 0 0 0 0 9 8 0 0 0 20 0 0 0 0 15 0 0 16 0 0 14 0 0
19 0 0 20 0 0 0 0 0 0 10 18 0 0 0 18 0 0 0 0 0 16 14 0 0 3 0 0 0 0 0 0 0 3 4 0 0 0 0 0 0 0 11 0 5 0 0 0 0 12 11 0
0 0 3 0 0 0 18 4 15 0 0 0 0 0 0 0 16 8 0 0 0 0 0 0 0 0 0 19 2 0 0 1 0 0 0 12 0 18 0 0 0 0 0 4 0 0 0 0 4 0 0 11
0 0 16 0 0 0 15 20 0 0 0 0 0 0 2 0 0 0 15 8 0 0 0 0 18 0 13 0 0 0 0 0 0 0 0 2 15 16 0 0 0 0 4 0 0 11 0 15 0 0 0
0 18 18 0 0 0 0 0 5 0 0 0 8 0 0 0 0 0 0 13 20 0 2 7 0 0 0 0 0 0 3 0 0 0 0 0 0 4 0 3 0 0 0 4 0 0 0 0 0 0 14 0 18
0 0 6 0 0 20 0 2 0 0 0 10 0 0 7 0 0 0 13 0 0 17 10 0 0 0 0 0 0 0 12 0 0 0 18 0 3 0 0 0 0 0 0 11 0 17 0 0 0 15 0
0 2 0 16 0 0 0 0 0 0 0 0 16 0 19 0 1 0 0 0 11 0 12 0 0 0 0 0 0 18 0 3 0 0 0 18 0 0 11 0 15 0 0 0 0 20 0 5 0 0 0
0 20 1 0 0 0 0 0 0 0 0 9 10 0 0 15 0 0 0 0 15 0 0 0 0 0 17 16 0 2 0 16 0 0 0 0 0 0 0 0 0 9 0 5 0 19 0 0 0 0 6 0
0 0 0 0 0 3 15 0 15 0 0 0 0 4 0 6 0 9 0 0 0 14 0 0 4 0 0 0 0 0 0 12 0 6 0 0 0 0 0 2 4 0 0 0 0 0 0 0 0 20 2 0
0 6 0 0 0 0 16 0 0 0 0 0 16 0 12 0 0 0 0 0 10 0 0 14 0 14 0 0 0 0 3 0 0 9 0 0 16 0 0 0 0 0 19 0 12 9 0 0 0 19 0 0
12 0 0 6 0 0 0 0 0 0 0 0 3 11 0 0 0 3 0 0 0 8 0 18 0 0 0 0 3 0 0 0 11 0 0 18 6 10 0 0 0 0 0 0 0 0 0 18 0 0 9
0 13 0 0 6 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0 0 0 20 0 10 0 0 0 12 0 0 0 0 0 5 0 0 2 0 0 0 5 0 0 8 0 16 0 0 0 0 5
0 1 0 0 0 3 0 11 0 0 0 0 6 0 10 0 0 0 0 20 0 0 0 14 0 0 0 8 0 20 0 0 0 0 12 0 0 10 0 20 0 0 1 0 0 0 0 20 0
0 11 0 0 0 16 0 13 0 0 7 0 0 0 0 5 0 0 0 12 0 0 19 0 7 0 0 0 12 0 0 10 0 0 0 15 0 0 0 0 0 17 0 0 0 0 0 0 9 2
0 0 7 3 0 0 0 0 7 0 0 0 0 2 0 0 0 0 0 16 18 11 0 20 0 0 0 0 13 0 0 17 0 0 0 0 0 7 0 0 15 0 0 0 0 12 19 0 0
0 6 0 0 0 10 0 0 0 6 0 0 12 0 20 0 0 19 0 0 0 0 0 0 0 0 14 11 0 0 0 0 8 0 6 0 9 0 8 0 0 0 0 6 0 16 0 0 0
0 13 0 16 0 0 0 0 0 0 0 0 17 4 0 0 0 0 6 0 3 0 0 10 0 15 0 0 0 12 0 15 0 0 0 0 18 7 0 0 0 0 0 0 5 0 0 2 0
0 14 12 0 0 0 0 0 0 0 0 3 20 0 0 20 0 0 16 0 0 17 0 0 0 0 0 12 7 0 0 0 15 0 0 14 0 0 0 19 0 0 0 0 0 5 16 0 0
0 0 0 13 0 5 0 0 0 8 0 0 5 0 16 15 0 0 0 0 0 0 11 0 16 0 0 0 16 5 0 0 0 0 0 0 0 0 17 0 11 0 0 0 9 0 0 9 0
0 0 13 0 0 0 8 0 0 0 0 0 5 13 0 0 0 2 0 20 0 0 0 0 0 0 7 20 0 0 0 12 0 16 0 13 0 0 0 17 0 0 0 14 0 11 0 0 0
10 0 0 1 0 0 0 0 0 16 0 0 0 13 0 15 0 0 0 0 19 0 5 4 0 0 0 0 0 0 0 0 0 3 0 10 0 0 0 0 0 0 3 14 0 0 0 14 6 0 0
13 0 13 0 0 0 0 0 3 0 0 9 0 0 1 15 0 0 0 0 0 0 0 0 0 15 17 0 0 0 2 13 0 0 0 0 8 0 0 0 6 0 0 0 18 8 0 0 0 0 0
0 0 7 0 0 6 0 0 0 0 0 20 0 16 0 17 14 0 0 0 0 0 19 0 9 0 0 0 0 0 0 11 0 0 0 11 0 0 0 10 0 0 0 0 8 0 0 0 16 0 0 16 0
20 0 14 0 0 0 0 0 0 15 0 4 0 0 0 0 0 18 19 0 0 0 0 0 0 0 0 13 0 0 4 0 3 9 0 0 0 0 0 0 0 0 0 10 0 11 0 13 8 0 0 0 0 0
0 2 18 0 0 0 0 6 0 0 0 0 1 0 0 4 14 0 0 0 0 0 0 0 0 9 0 3 0 5 3 0 0 0 0 0 0 0 4 0 0 0 1 0 0 15 9 0 0 0 0
0 0 12 7 0 0 0 0 0 0 0 0 8 8 0 0 0 0 0 0 14 0 0 1 0 9 0 4 0 0 0 0 10 0 0 0 0 0 0 0 18 20 0 0 0 0 0 7 0 0 4 0 20 0 0
0 15 0 0 16 0 0 0 0 0 0 14 0 8 0 11 0 0 18 0 0 0 0 7 0 0 0 5 0 0 12 0 0 0 0 19 0 0 11 0 0 0 5 0 0 16 0 4 0 0
2 0 0 13 0 0 0 0 0 12 0 0 0 3 0 0 17 0 13 0 0 0 0 0 0 0 0 1 17 0 0 0 18 3 0 0 0 0 0 0 14 0 0 17 0 3 0 0 0 0 5 0
0 0 0 12 0 3 0 0 0 0 0 0 0 4 18 0 0 0 0 0 0 7 7 0 0 0 20 0 0 4 5 0 0 0 0 0 0 6 0 0 4 0 0 15 0 0 0 0 0 0 10 11 0
0 0 16 0 0 19 0 9 0 0 0 0 18 0 0 0 0 0 0 12 0 0 6 1 0 0 2 0 0 0 0 0 0 11 0 0 0 14 7 0 0 0 0 13 0 0 3 0 0 13 0 0
0 0 0 3 0 0 7 17 0 0 0 12 0 0 0 9 0 0 16 0 0 10 0 10 0 0 0 0 0 0 0 0 9 0 5 0 0 0 10 0 0 12 0 0 0 0 0 0 10 16
0 2 0 0 10 0 0 0 0 12 0 13 0 0 0 18 0 0 0 3 0 0 0 0 0 0 8 3 0 0 0 0 0 0 0 18 17 4 0 0 0 0 13 0 0 0 0 0 17 0 19
0 0 0 0 2 14 0 9 0 0 0 0 0 10 0 0 0 0 0 0 16 0 18 0 19 0 0 0 20 0 0 0 0 0 0 2 0 14 0 0 18 0 0 0 10 0 0 0 17 1 0 0
```

Figure 5.1: Uniform Randomly Generated Kernel 4D Matrix

6 Data Structure for 2-Dimensional Filter(Based on Compressed Sparse Row(CSR) Technique)

The study and Data Structures formulated use Compressed Sparse Row technique for compressing the 4-Dimensional Kernel matrix(which can be thought of as a collection of 2-Dimensional matrices(filters) stacked in the 3rd and in the fourth dimension). Compressed Sparse Row (CSR) technique is used to compress a sparsely filled 2-Dimensional matrix (filter) into a set of three 1-Dimensional lists (containing only dense values from the 2-Dimensional matrix(filter)):

- A: Contains all the dense values from the 2-Dimensional matrix(filter)
- JA: Contains the column index for each of the dense values in the A list
- IA: Contains the information for number of elements in each row of the 2-Dimensional matrix(filter)

The three 1-Dimensional lists are implemented as Vectors from the Standard Template Library (STL) in C++ as the vectors allow the 1-Dimensional list to grow dynamically based on number of elements in the 2-Dimensional matrix(filter).The choice of vectors over traditional 1-Dimensional C++ arrays also allows to access the wide variety of functions provided in the Standard Template Library(STL) with a constant order time complexity (O(1) using Big-O notation for complexity). The three vectors thus obtained for a 2-Dimensional matrix(filter) are then encapsulated in an object. The Figure 6.1 below diagrammatically represents the object encapsulating the three Compressed Sparse Row vectors.

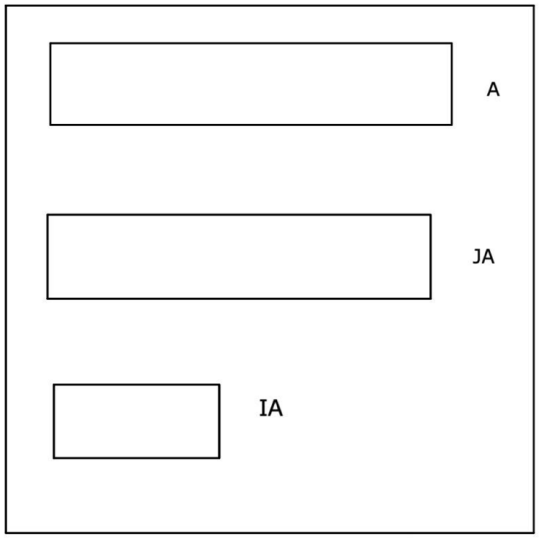

Figure 6.1: Object encapsulating the three vectors A,IA,JA

7 Data Structure for collection of 2-Dimensional Filters

The CSR (Compressed Sparse Row) technique deals with a 2-Dimensional matrix and compressing the data contained in the 2-Dimensional matrix. As there are a total of four dimensions for accessing a element in a 4-Dimensional space matrix, this leaves us with two dimensions that have not been sparsified.To summarize there are two sparse dimensions and two dense dimensions(dense dimensions are ones which have not been used for compression).Thus in implementation the collection of Compressed Sparse Row objects from the 4-Dimensional Kernel matrix is stacked in a 2-Dimensional array of objects which can be indexed on the basis of the dense parameters serving as row index and column index. The Figure 7.1 below is a diagrammatic representation of the data structure for the 4-Dimensional Kernel Matrix.

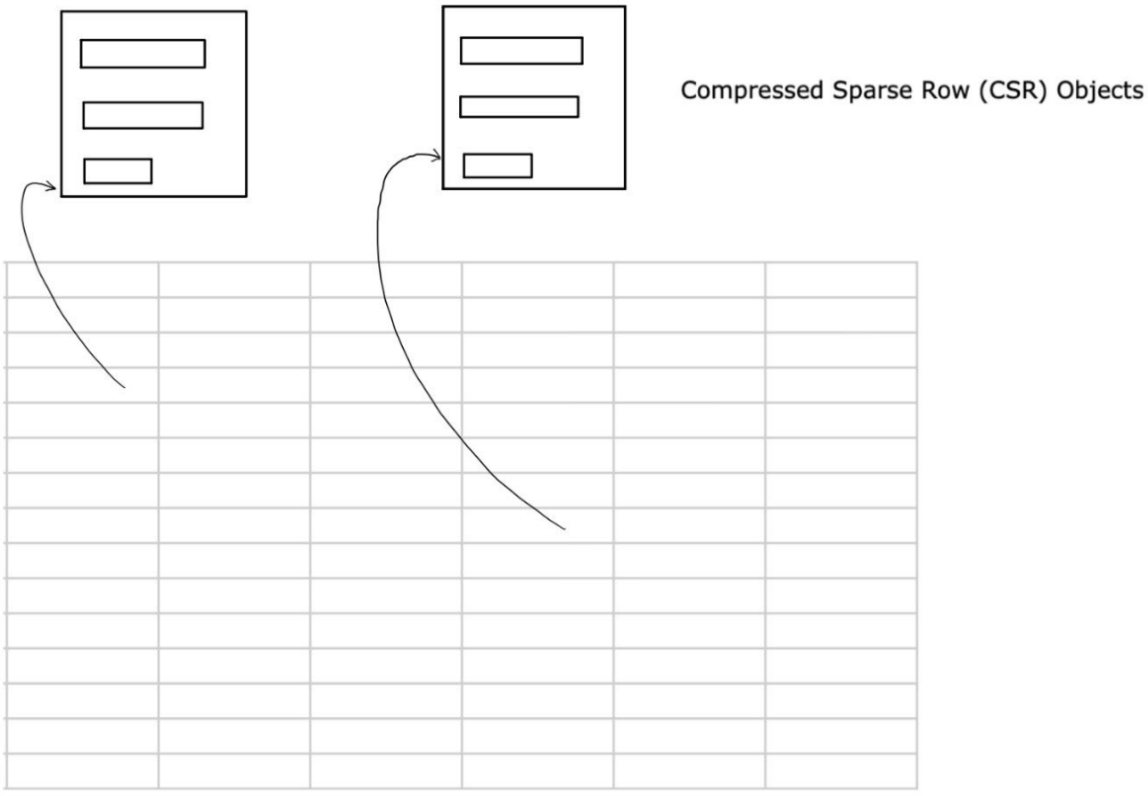

Figure 7.1: 2-Dimensional Array of CSR(Compressed Sparse Row) objects

8 Algorithms

8.1 Sparsify_4d function

Sparsify_4d function is responsible for creating the final data structure for performing sparse convolution. The main function passes the dense and the sparse dimensions as arguments to the Sparsify_4d function which in turn places the dense dimensions in the outermost loop for iterating over the 2-Dimensional Object array. For every iteration inside the nested loop(of the 2-Dimensional Object array) the 4-Dimensional kernel matrix is traversed in one of the 24 ways possible(there are 24 permutations/arrangements(4*3*2*1) in which a 4-Dimensional matrix can be traversed) and a 2-Dimensional array (array_to _csr) is created. This 2-Dimensional matrix created(inside every iteration of the nested loop of the 2-Dimensional Object array) is then passed to creating_csr function which is responsible for populating the A,IA,JA vectors (based on the Compressed Sparse Row algorithm) and encapsulating the three vectors (namely A,IA and JA) and passing the object back to the Sparsify_4d function. The function then is responsible for placing the object by indexing into the 2-Dimensional array of CSR(Compressed Sparse Row Objects) Objects based on the dense dimensions (dense dimensions serve as the row and column indexes into the 2-Dimensional Object array). The Figure 8.1 below is a code snippet of sparsify_4d function for ydxm order traversal (y(row index) and d(column index) serving as dense dimensions for indexing into the 2-Dimensional Object array).

```
//Y,D,X,M ORDER TRAVERSAL
void sparsify_4d_22(ConvolutionNode **array_nodes,int m,int depth,int k1,int k2)//sparsify's the 4D kernel matrix
{
    for(int object_row=0;object_row<k2;object_row++)//Y DENSE
    {
      for(int object_column=0;object_column<depth;object_column++) //D DENSE
      {
        int array_to_csr[150][150];//TO BE PASSED TO creating_csr() FOR EVERY ITERATON

        for(int i=0;i<k1;i++)//X SPARSE
        {
          for(int c=0;c<m;c++)//M SPARSE
          {
              array_to_csr[i][c]=kernel[c][object_column][i][object_row];
          }
        }
        //after filling up one of the 2D arrays from the 4D matrix pass it to the creating_csr() function
        //creating_csr will create three vectors for each 2D matrix, put them in an object
        //and pass the object back to sparsify_4d
        //put this object in the 2D array array_nodes[][]
        array_nodes[object_row][object_column]=creating_csr(array_to_csr,k1,m);
       }
    }
}
```

Figure 8.1: Sparsify_4d function for YDXM Order traversal

The Figure 8.2 below is a flowchart of Sparsify_4d algorithm with 2-Dimensional Array of CSR(Compressed Sparse Row) objects as the final data structure and the output of the Sparsify_4d algorithm.

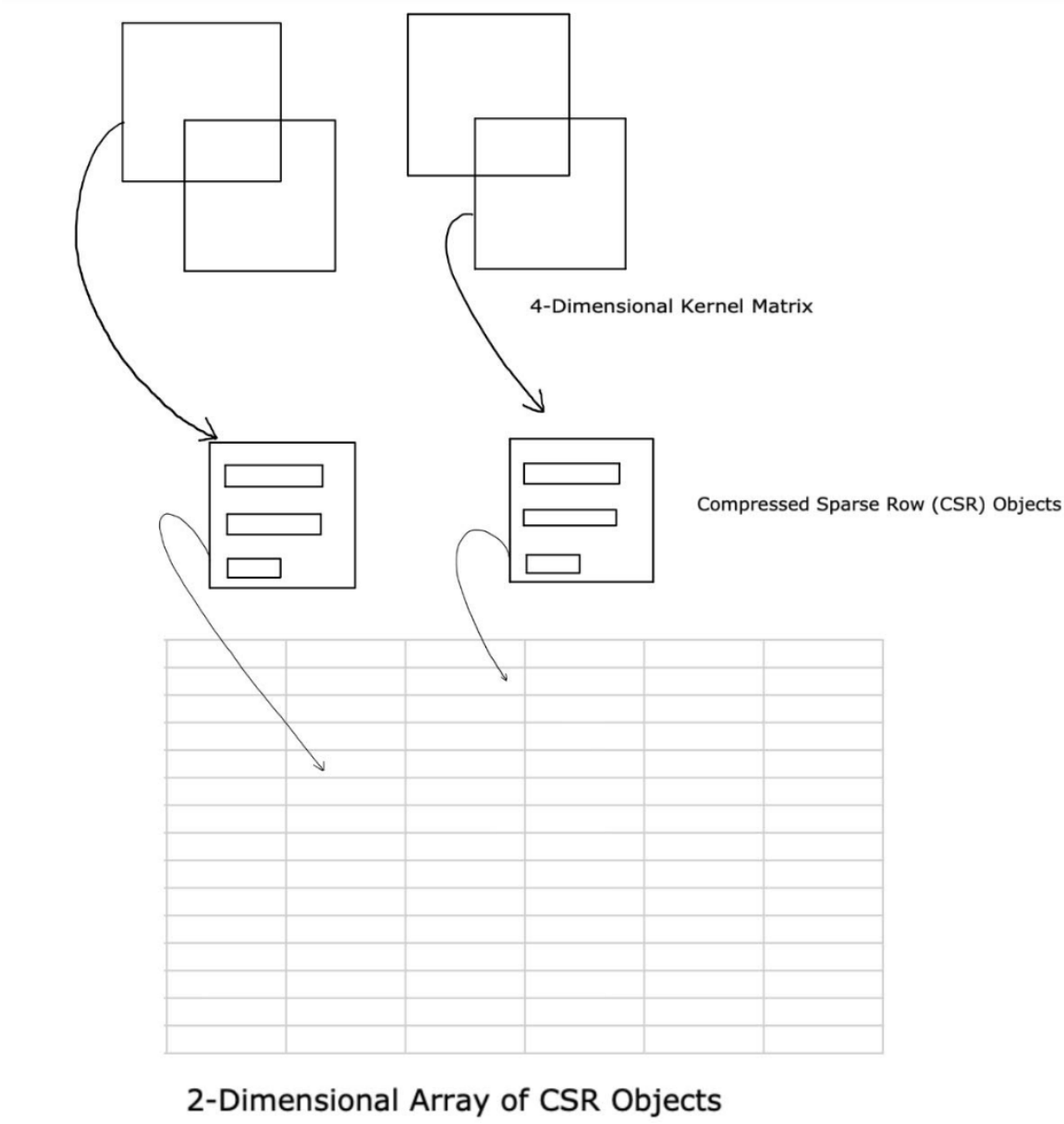

Figure 8.2: Flowchart of Sparsify_4d algorithm

8.2 Creating_csr function

Creating_csr algorithm is responsible for creating a compressed representation for the 2-Dimensional matrices(filters) passed to it by the Sparsify_4d function. The Creating_csr function achieves compression by compressing the 2-Dimensional matrix received into three vectors:

- A : iterating over the 2-Dimensional matrix an element is pushed into the vector A only if it is a non-zero element.Hence after iterating the 2-Dimensional matrix the vector A contains all the dense values from the 2-Dimensional matrix(filter).
- JA: As the non-zero elements are pushed into the A vector their corresponding column indexes (column index in the 2-Dimensional filter) are pushed into the JA vector.Hence after iterating the 2-Dimensional matrix the vector JA contains the column indexes to all the dense values from the 2-Dimensional matrix(filter).
- IA: A counter variable is kept which keeps a count on the number of non-zero elements up to that particular point. At the end of each iteration of the row the count(value of the counter variable) is pushed into the IA vector.Thus the IA vector contains number of elements in rows in a cumulative order and the number of elements in any row has to be extracted by getting the difference of two consecutive entries in the vector IA i.e

  $$number(n) = IA[n+1] - IA[n] \tag{1}$$

  where n denotes the index of the row in the 2-Dimensional matrix and number(n) denotes the number of elements in the row indexed at n.

After populating the three vectors(A,IA and JA) the Creating_csr algorithm encapsulates the three vectors in an object and passes the object to the Sparsify_4d function.

8.3 Sparse Convolution(sparse_convolution function)

This algorithm is responsible for performing sparse convolution with one of the 24 different variations of the final data structure from the Sparsify_4d algorithm. The array_nodes is a pointer to the final data structure(from the Sparsify_4d algorithm) and is received as one of the passed parameters from the main function. Along with the pointer to the final data structure the main function also passes all the dense parameters(four dense parameters passed in total) for convolution (two of the passed four dense parameters are needed for indexing into the final data structure). The four dense parameters are nested in outermost loops and the corresponding sparse parameters (two sparse parameters) are placed in loops inside the four nested dense loops. The innermost loops(associated with the two sparse parameters) are thus used for extracting the non-zero values (from the original 4-Dimensional Kernel Matrix) for performing convolution with the Input 3-Dimensional matrix.

8.3.1 Innermost Sparse Loops

The information for number of rows (in the original 2-Dimensional filter before being sparsified by the Sparsify_4d algorithm) in the [dense_parameter_1][dense_parameter_2] positioned object is obtained by

$$array\_nodes[dense\_parameter\_1][dense\_parameter\_2].ia.size() - 1 \tag{2}$$

as ia vector contains the number of non zero elements in each row of the original 2-Dimensional filter it inherently contains the number of rows in the original 2-Dimensional filter. Using the STL(Standard template library function size()) the number of elements in the ia vector is extracted which gives us the number of rows in the original 2-Dimensional filter.However the CSR(Compressed Sparse Row) implementation from Creating_csr function has a cumulative number of elements

$$numberofelements[row] = numberofelements[row] + numberofelements[row-1] \tag{3}$$

and the first element in the vector ia is kept as zero to indicate there are zero cumulative elements to begin with. Owing to this implementation there is an extra index in the ia vector and thus to get the number of rows in the 2-Dimensional filter that extra index is subtracted from the size of the ia vector.

As the CSR(Compressed Sparse Row) implementation from Creating_csr function has a cumulative number of elements for each row the number of elements in the original 2-Dimensional filter is extracted by

$$\begin{aligned} numberofelements[row] = {} & array\_nodes[dense\_parameter\_1][dense\_parameter\_2].ia[row+1] \\ & - array\_nodes[dense\_parameter\_1][dense\_parameter\_2].ia[row] \end{aligned} \tag{4}$$

The column number(in the original 2-Dimensional filter) for the non zero element is extracted from the data structure by

$$array\_nodes[dense\_parameter\_1][dense\_parameter\_2].ja[element\_iterator] \tag{5}$$

where ja vector from the Creating_csr implementation stores the column index of the non zero elements in the original 2-Dimensional filter.

The non zero value (value) is extracted by

$$array\_nodes[dense\_parameter\_1][dense\_parameter\_2].a[element\_iterator] \quad (6)$$

Finally using the information extracted above the value for two sparse parameters is obtained as

- row_num (Row number:by extracting row index)
- col_num (Column number: by extracting column index)

Thus the sparse parameter values obtained are substituted in the convolution equation for e.g

$$output[w][h][row\_num] += input[w + x][h + y][col\_num] * value \quad (7)$$

where the above equation is the Sparse Convolution equation for xymd arrangement. m and d being the sparse parameters are extracted and substituted as row_num and col_num respectively. output represents the output 3-Dimensional matrix and input represents the input 3-Dimensional matrix. The value variable is the non zero value from the data structure.

8.3.2 Edge Case of Convolution

The Edge case of convolution arises when the filter is made to convolve over the input matrix and the span of the filter matrix is outside the bounds of the input matrix. In this case the program has to deal with this as a special case as the values of the filter which lie outside the boundaries of the input matrix would convolve with values which do not belong to the input matrix.The Figure 8.3 below depicts the edge case

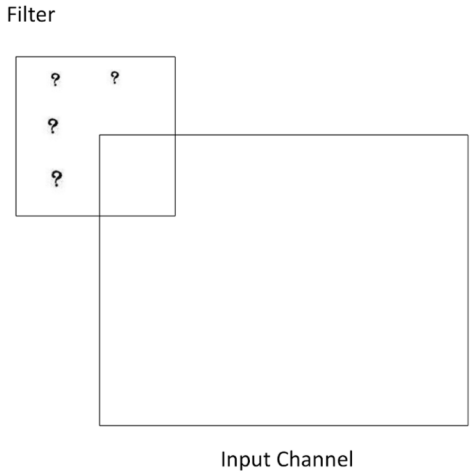

Figure 8.3: Edge Case of Convolution

8.3.3 Dealing with Edge Case of Convolution

The current implementation deals with the problem of Edge Case by initializing the whole 3-Dimensional Input matrix at all zeroes before populating the 3-Dimensional matrix and then shifting the 3-Dimensional coordinate system (or the vertex of the 3-Dimensional Input matrix) by a constant. This provides a solution to the edge case as the values of the filter which lie outside the boundaries of the input matrix are now made to convolve with zeroes hence do not wrongly skew the output of convolution. This solution is the most appropriate and time efficient solution as otherwise the program has to place a conditional check on the values of the filter so they do not span and convolve over values which do not belong the input matrix which would increase the time complexity of the solution. The Figure 8.4 below is a graphical representation of the solution designed for dealing with the Edge Case of the convolution.

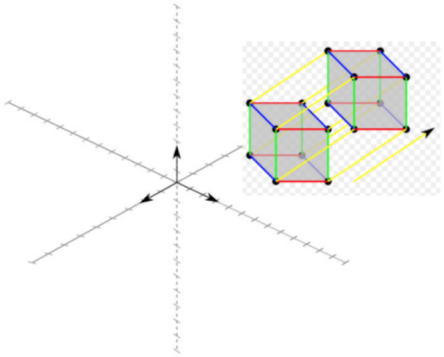

Figure 8.4: Shifting Vertex of Input: Dealing with Edge Case of Convolution

8.3.4 General Equation for Sparse Convolution

Taking into consideration the edge case of convolution the final equation for Sparse Convolution is of the format

$$output[w][h][row\_num] += input[w+x+constant][h+y+constant][col\_num+constant] * value \tag{8}$$

where the above equation is the final equation for Sparse Convolution equation for xymd arrangement and constant is the magnitude of displacement of the 3-Dimensional Input matrix. m and d being the sparse parameters are extracted from the data structure and substituted as row_num and col_num respectively. output represents the output 3-Dimensional matrix and input represents the input 3-Dimensional matrix. The value variable is the non zero value from the data structure.

8.3.5 Time Performance of Sparse Convolution

Timing performance of Sparse Convolution recorded in nanoseconds with the help of high_resolution_clock from chrono library defined in the std namespace. A instance of clock is placed at the start of the sparse convolution function and one at the end of the sparse convolution function. The absolute difference of the two clock instances gives us the time for performing Sparse Convolution in nanoseconds.

8.4 Space Performance of Sparse Data Structure

The space required by the Sparse Data Structure (2-Dimensional Array of CSR (Compressed Sparse Row Objects) is the space required by summing up the space required by each object of the 2-Dimensional array of CSR(Compressed Sparse Row) objects.This is calculated by iterating over the 2-Dimensional array of objects and maintaining a summation variable for accumulating the results from each object. The space required by each object of the 2-Dimensional array of CSR objects is given by

$$\begin{aligned} &spaceobject[dense\_parameter1][dense\_parameter2] = \\ &array\_nodes[dense\_parameter1][dense\_parameter2].ia.size()+ \\ &array\_nodes[dense\_parameter1][dense\_parameter2].ja.size()+ \\ &array\_nodes[dense\_parameter1][dense\_parameter2].a.size() \end{aligned} \tag{9}$$

Where space object denotes the space occupied by one object at index [dense_parameter1][dense_parameter2]. Summing up the space required by all the objects in the 2-Dimensional Array of CSR (Compressed Sparse Row) objects thus gives us the total units of space required by the Sparse Data Structure. The current Kernel matrix is assumed to be composed of integer values. Thus by multiplying the units obtained with the compiler specific space allocated for integers we can obtain the number of bytes allocated for the Sparse Data Structure on the heap (as the 2-Dimensional array of objects is created using new operator and the data members of the Class are three vectors, the entire memory is allocated during run-time on the heap memory).

9 Analysis of Sparse Convolution performed with different arrangements of dense and sparse dimensions : Factors affecting Time and Space requirement

There are 4 Dimensions to the Sparse Kernel Matrix which gives a total of 24 possible permutations/arrangements (4*3*2*1 : arrangements possible without replacement) which can be exploited to derive 24 different variations of the final data structure from the Sparsify_4d algorithm. 15 of these 24 different variations of the data structure from the Sparsify _4d algorithm are then used to perform Sparse Convolution and their time and space performance is analyzed. There are 4 parameters which affect the timing performance of a Sparse Convolution those being:

- Width(W) of Input Channel (Width of channels from the Input 3-Dimensional Matrix(W*H*D))
- Height(H) of Input Channel (Height of channels from the Input 3-Dimensional Matrix(W*H*D))
- Depth(D) of Input channels (Depth of channels from the Input 3-Dimensional Matrix(W*H*D) and also serves as one of the dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D))
- M (One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which also determines the number of channels in the Output 3-Dimensional matrix(W*H*M))

There are two parameters which affect the Space required by the Sparse Data Structure (from the Sparsify_4d algorithm) which is used for performing sparse convolution :

- M (One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which also determines the number of channels in the Output 3-Dimensional matrix(W*H*M))
- Depth(D) (One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) and also serves as Depth of channels from the Input 3-Dimensional Matrix(W*H*D))

The size of filters(K*K) is a constant and typical sizes of filters are 3*3 or 5*5. Due to the size of filters being very small in comparison to M,D and the fact that size of filters is a constant these two dimensions do not affect the Space Complexity of Sparsify_4d algorithm whose end product is the data structure(2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution.

Thus there are two parameters in common which affect both the time and space requirement for performing Sparse Convolution :

- M : One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which is the input to Sparsify _4d algorithm giving the final data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution and also determines the number of channels in the Output 3-Dimensional matrix(W*H*M)
- Depth(D) : One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which is the input to Sparsify _4d algorithm giving the final data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution and also serves as Depth of channels from the Input 3-Dimensional Matrix(W*H*D)

This section focuses on what affect the changing/varying of these two parameters(M and D) has on the timing and space performance of the data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution with different arrangements of dense and sparse dimensions.

Analysis done on basis of nature of relationship between the parameters (parameters being M and D) and performance of Sparse Convolution (based on Time and Space Performance) will help draw inferences on the design of the Multi-channel convolution system comprising of :

- Input 3-Dimensional Matrix (W*H*D)
- Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects)
- Output 3-Dimensional Matrix (W*H*M)

Experimental evaluation is done at two different levels of Sparsity of the Kernel 4-Dimensional matrix (which is the input to the Sparsify_4d algorithm for creating Data Structure for Sparse Convolution):

- 60 percent Sparsity (or 40 percent Dense)
- 80 percent Sparsity (or 20 percent Dense)

Where Sparsity of Kernel 4-Dimensional matrix is defined as what percentage of the total data inside the Kernel 4-Dimensional matrix is Sparse (numerically zero).

Timing performance for Sparse Convolution is recorded in nanoseconds and Space performance of the Data Structure used for Sparse Convolution is recorded in number of memory units occupied(based on the data type). The current implementation has integer data type so multiplying the memory units by number of bytes occupied by one integer (based on compiler) will give the number of bytes occupied by the Data Structure (in a particular arrangement) used for performing Sparse Convolution.

9.1 mdxy Arrangement for Sparse Convolution

m(row index) and d(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). x and y are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.1.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.1 and Figure 9.2 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

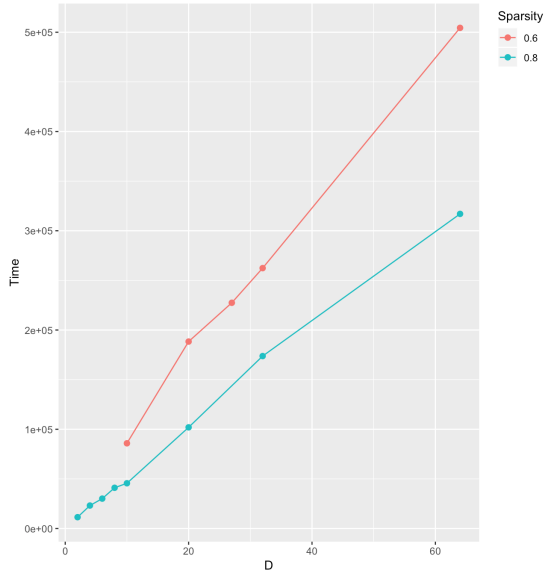

Figure 9.1: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

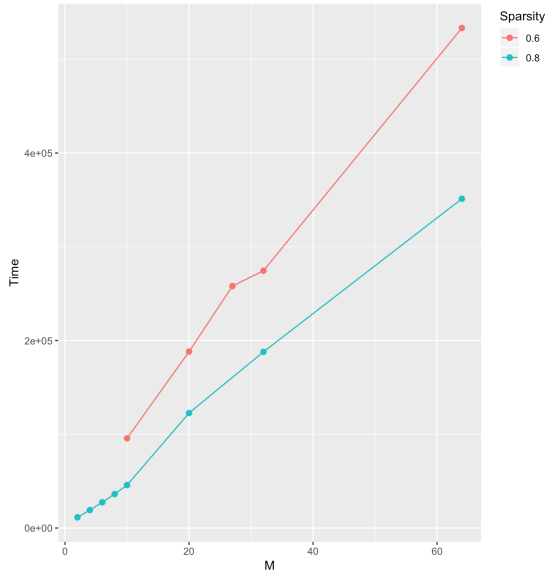

Figure 9.2: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.1.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.3 and Figure 9.4 it is observed that the slope of Space vs M graph and the Space vs D graph is equal and positive.

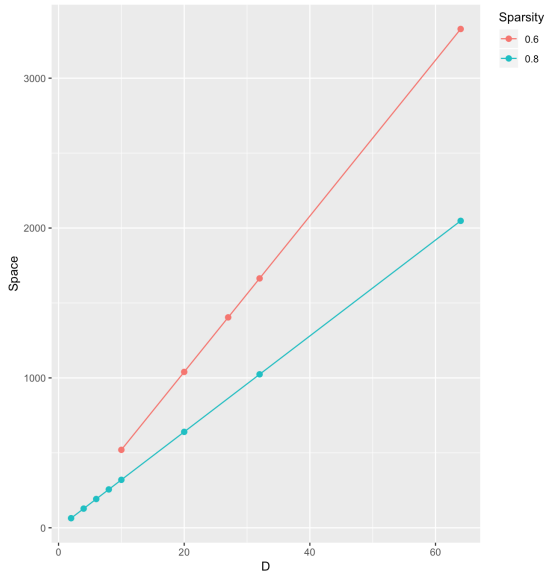

Figure 9.3: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

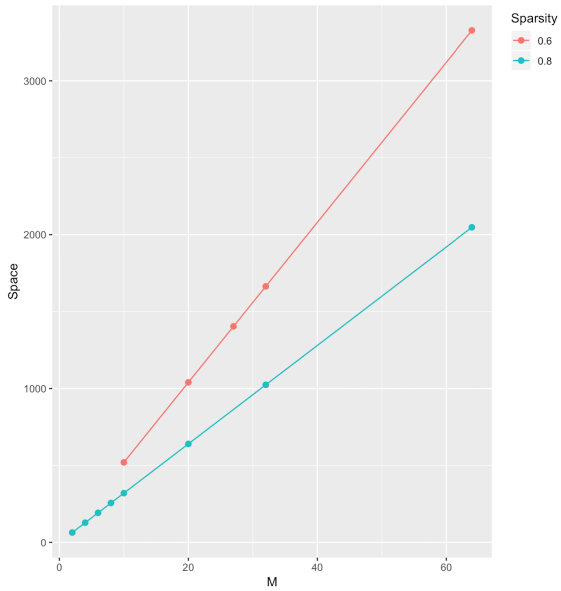

Figure 9.4: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.1.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement mdxy(with m and d being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance with space performance being same for both design parameters).

9.2 xymd Arrangement for Sparse Convolution

x(row index) and y(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). m and d are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.2.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.5 and Figure 9.6 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

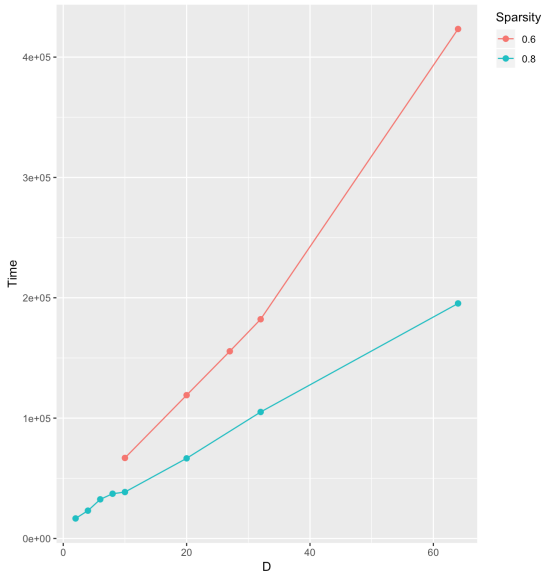

Figure 9.5: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

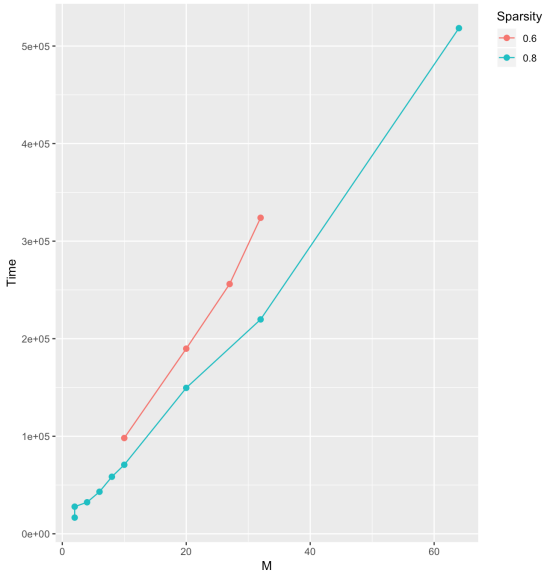

Figure 9.6: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.2.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.7 and Figure 9.8 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

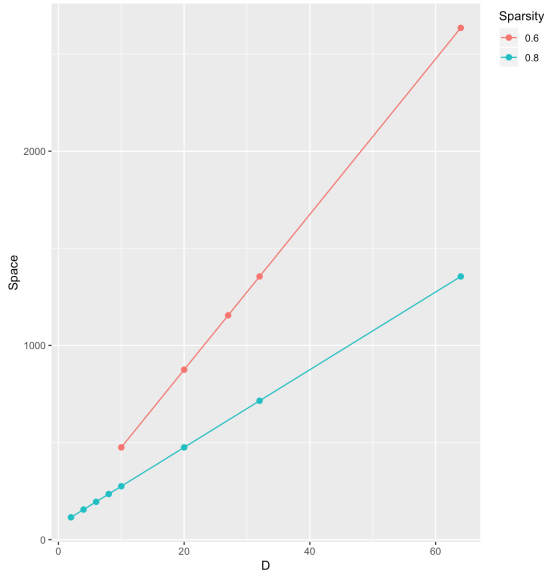

Figure 9.7: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

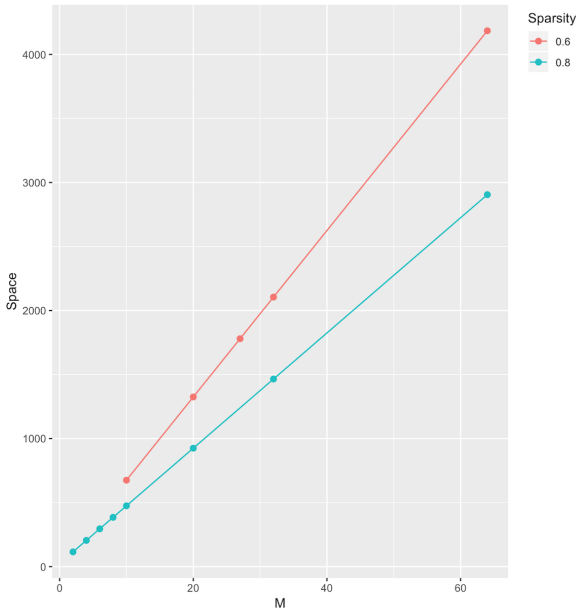

Figure 9.8: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.2.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement xymd(with x and y being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.3 dymx Arrangement for Sparse Convolution

d(row index) and y(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). m and x are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.3.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.9 and Figure 9.10 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

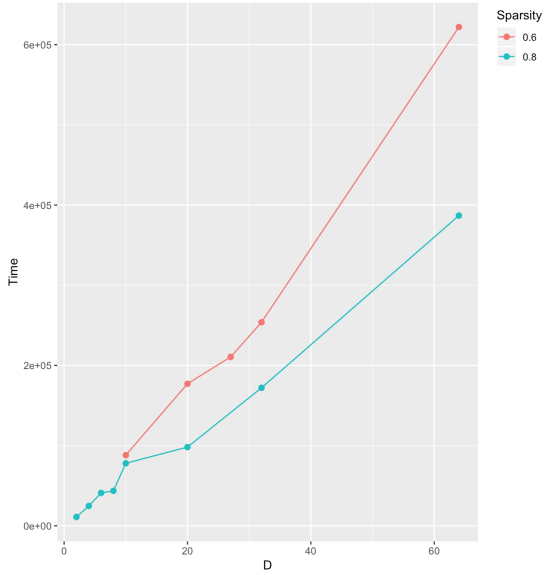

Figure 9.9: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

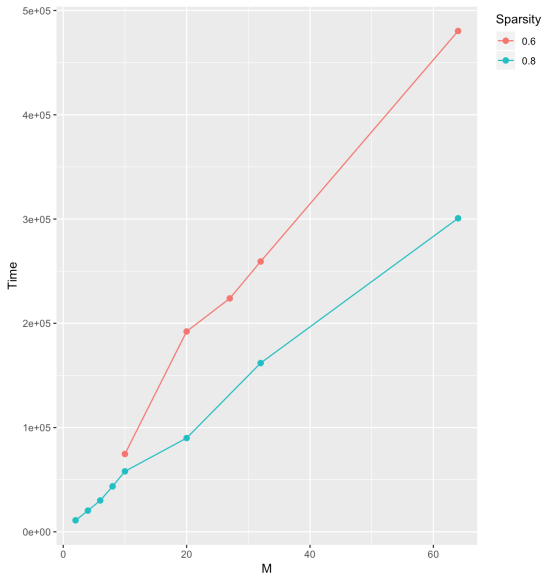

Figure 9.10: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.3.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.11 and Figure 9.12 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

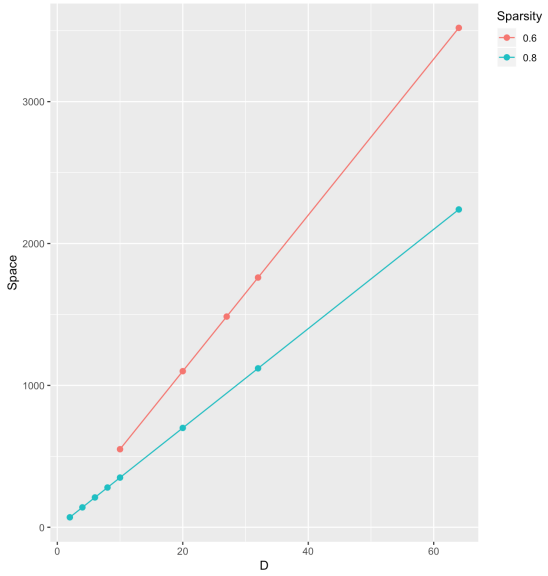

Figure 9.11: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

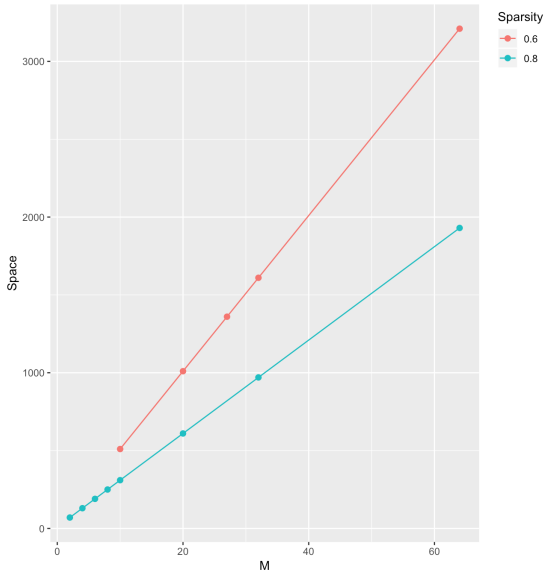

Figure 9.12: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.3.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement dymx(with d and y being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.4 mxdy Arrangement for Sparse Convolution

m(row index) and x(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). d and y are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.4.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.13 and Figure 9.14 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

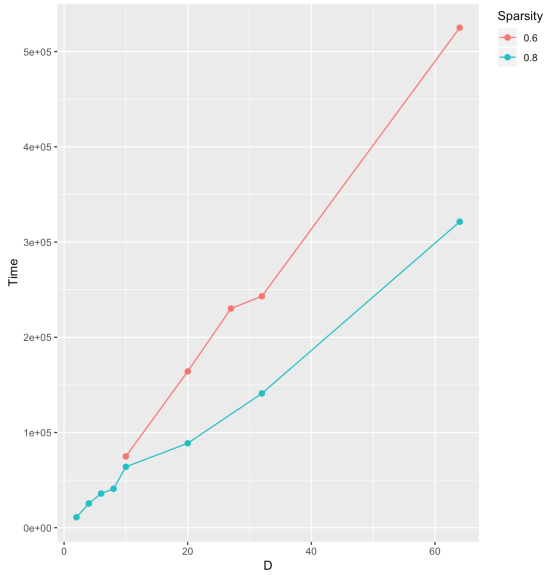

Figure 9.13: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

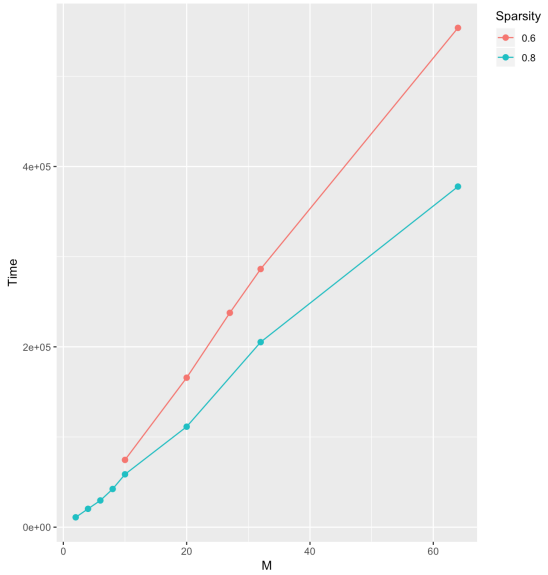

Figure 9.14: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.4.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.15 and Figure 9.16 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

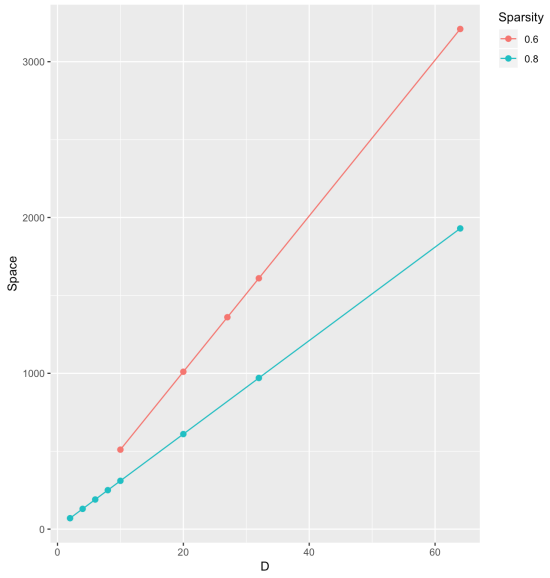

Figure 9.15: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

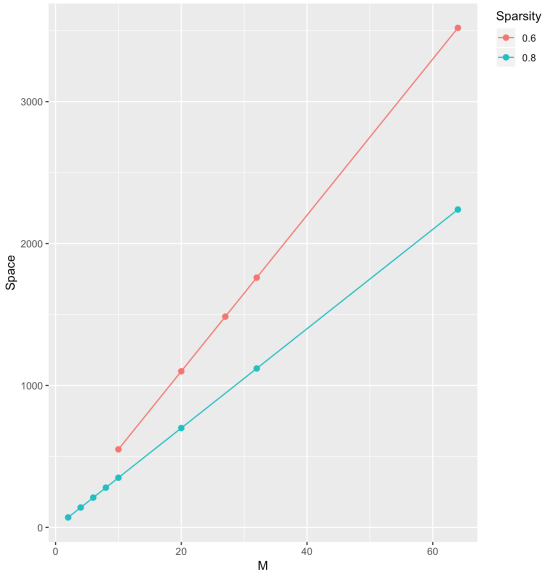

Figure 9.16: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.4.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement mxdy(with m and x being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.5 mxyd Arrangement for Sparse Convolution

m(row index) and x(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). y and d are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.5.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.17 and Figure 9.18 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

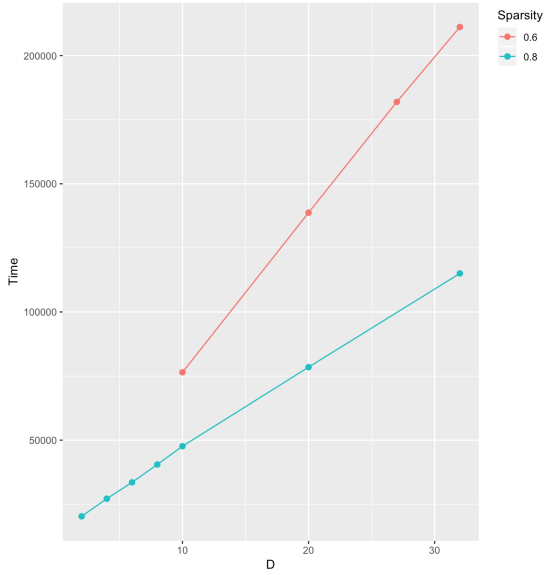

Figure 9.17: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

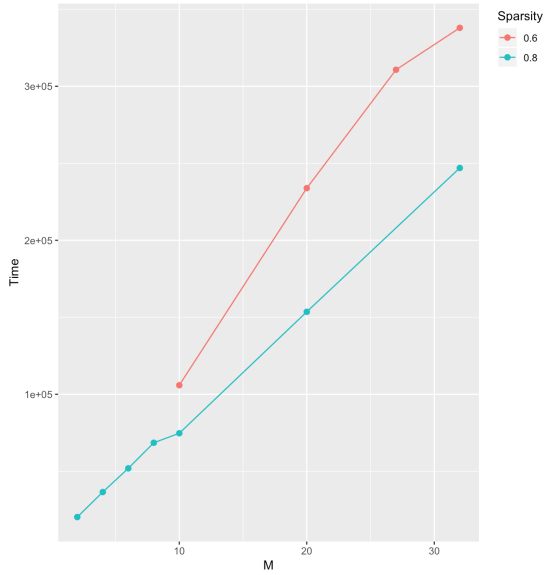

Figure 9.18: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.5.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.19 and Figure 9.20 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

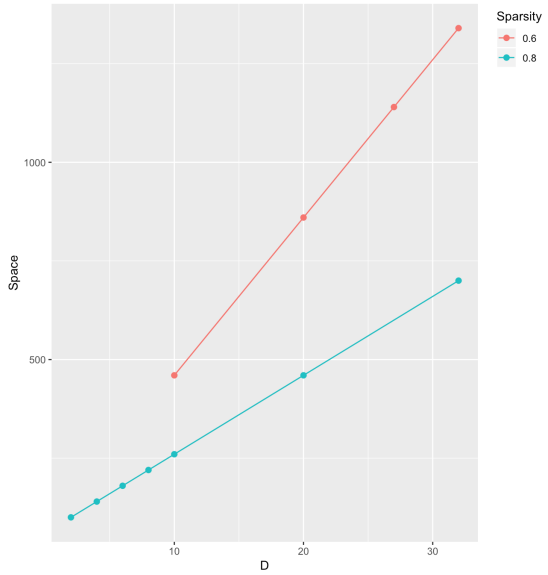

Figure 9.19: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

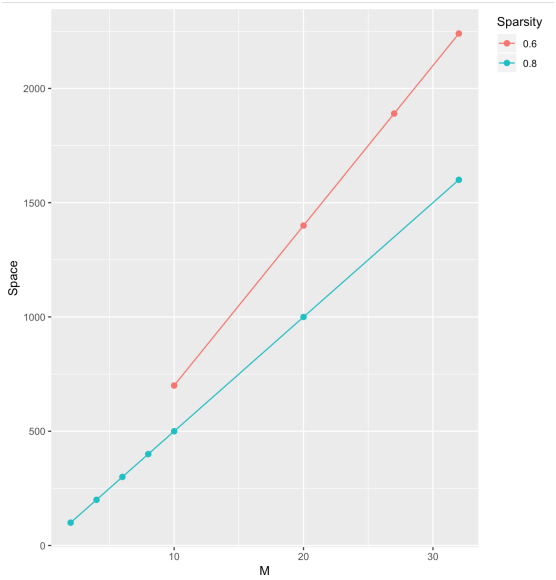

Figure 9.20: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.5.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement mxyd(with m and x being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.6 xmyd Arrangement for Sparse Convolution

x(row index) and m(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). y and d are the dimensions which have been sparsified by the Sparsify_4d algorithm.

9.6.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

In Figure 9.22 for Time vs M plot it is observed that the slope for sparsity 0.6 is much greater in magnitude in comparison to the slope for sparsity 0.8 whereas in Figure 9.21 for Time vs D plot it is observed that the slopes for sparsity 0.6 and sparsity 0.8 are close in magnitude. This leads to the impression from the graphical visualization that Time vs D has a greater positive slope than Time vs M (for sparsity 0.8) which is not true and this can be verified by comparing the numerical values for Time at the Y-axis.Thus comparing the experimental results from Figure 9.21 and Figure 9.22 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

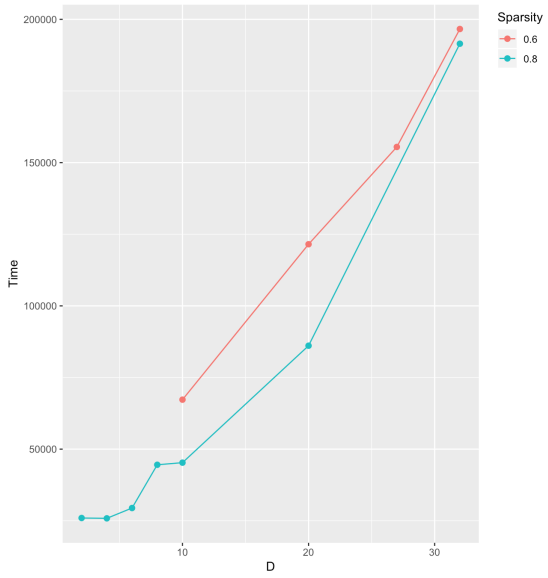

Figure 9.21: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

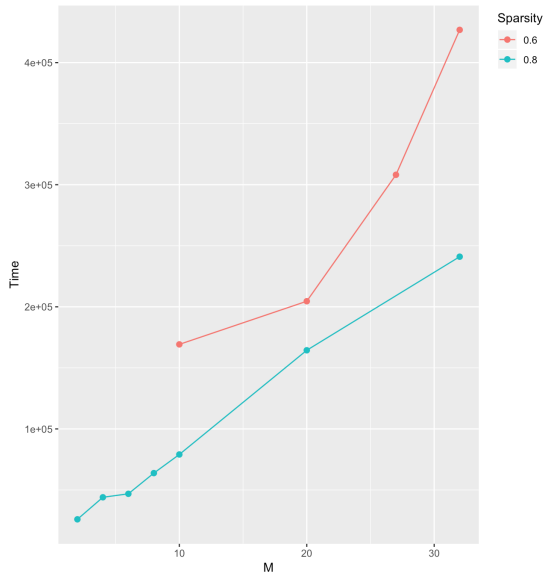

Figure 9.22: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.6.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.23 and Figure 9.24 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

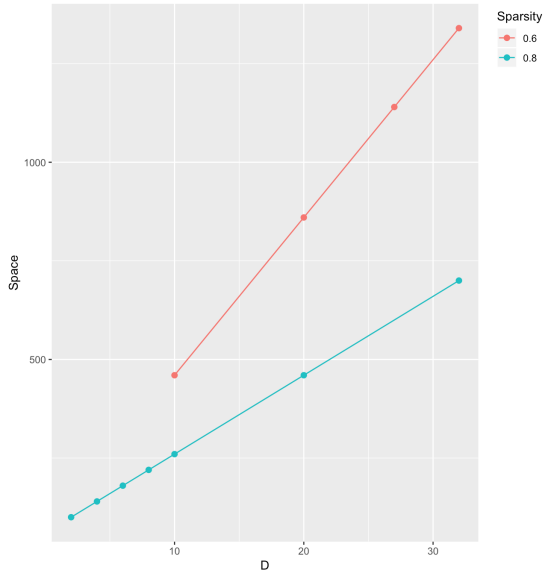

Figure 9.23: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

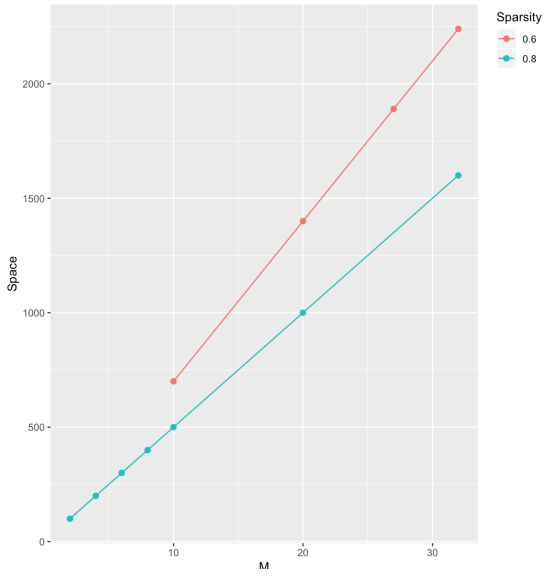

Figure 9.24: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.6.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement xmyd(with x and m being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.7 ymxd Arrangement for Sparse Convolution

y(row index) and m(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). x and d are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.7.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.25 and Figure 9.26 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

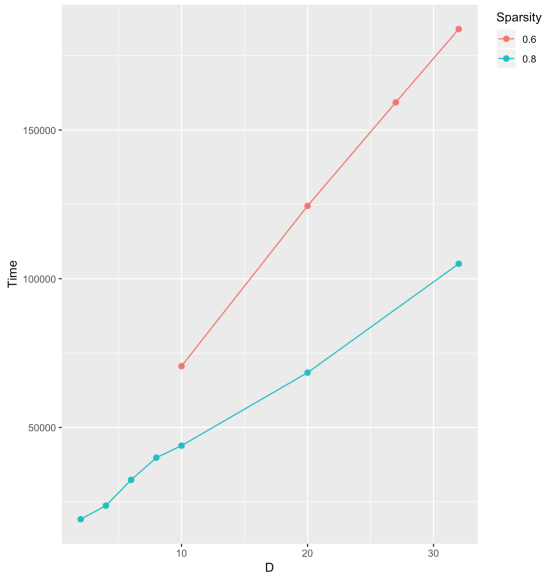

Figure 9.25: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

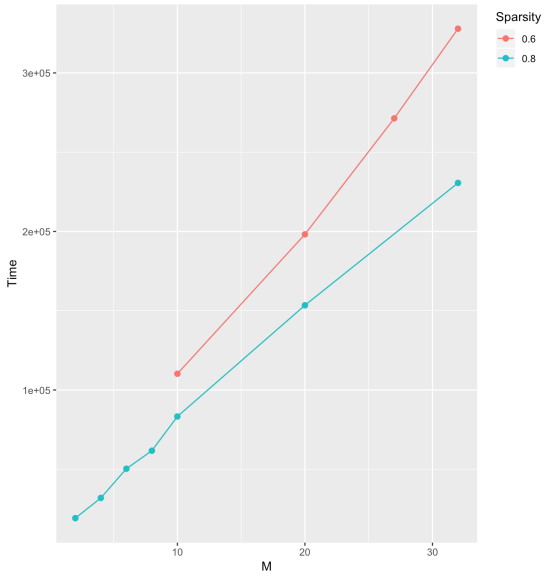

Figure 9.26: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.7.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.27 and Figure 9.28 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

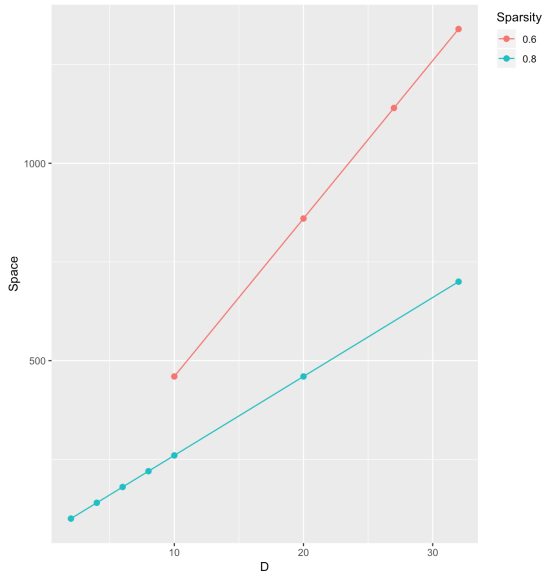

Figure 9.27: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

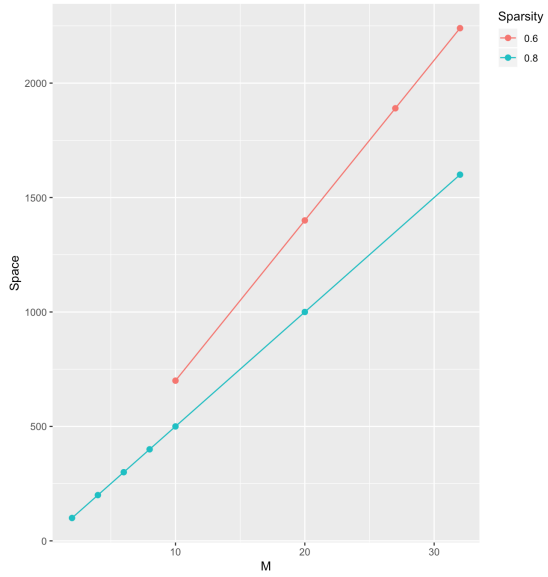

Figure 9.28: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.7.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement ymxd(with y and m being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.8 myxd Arrangement for Sparse Convolution

m(row index) and y(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). x and d are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.8.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.29 and Figure 9.30 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

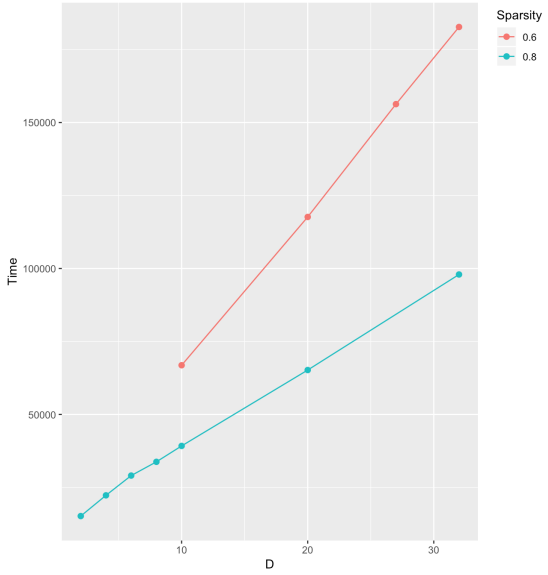

Figure 9.29: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

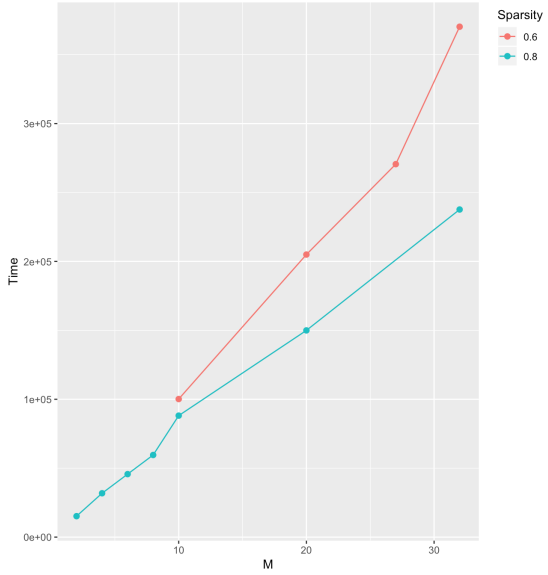

Figure 9.30: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.8.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.31 and Figure 9.32 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

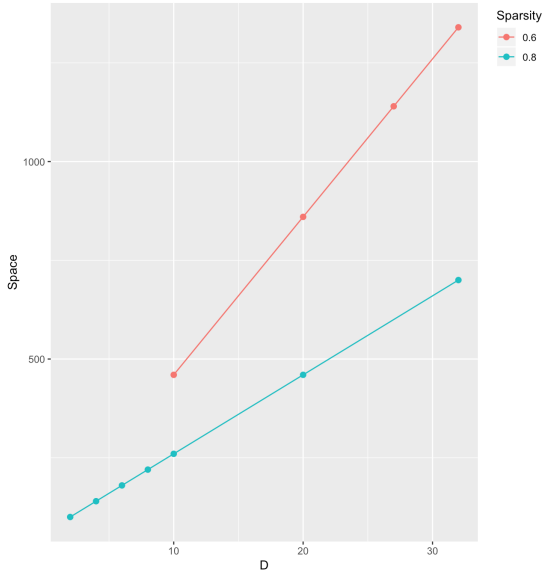

Figure 9.31: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

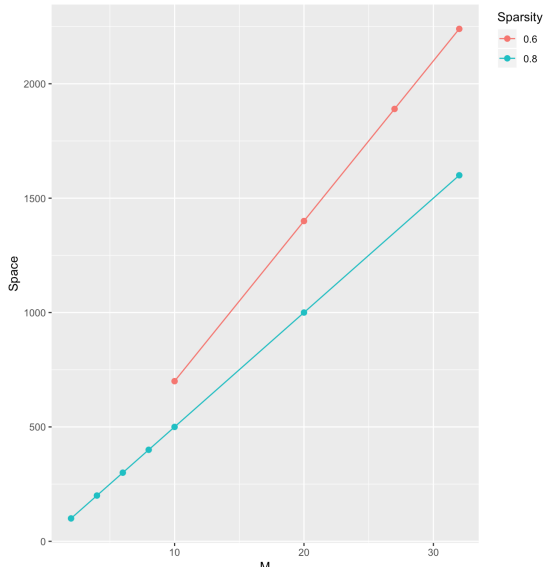

Figure 9.32: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.8.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement myxd(with m and y being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.9 yxmd Arrangement for Sparse Convolution

y(row index) and x(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). m and d are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.9.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.33 and Figure 9.34 it is observed that Time vs M graph has a greater positive slope compared to the Time vs D graph.

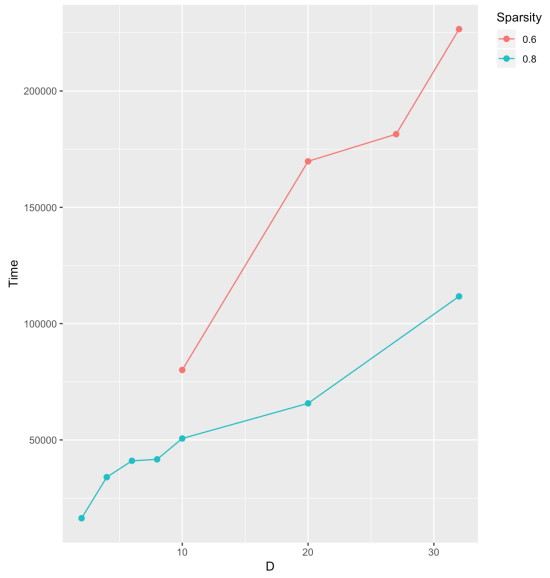

Figure 9.33: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

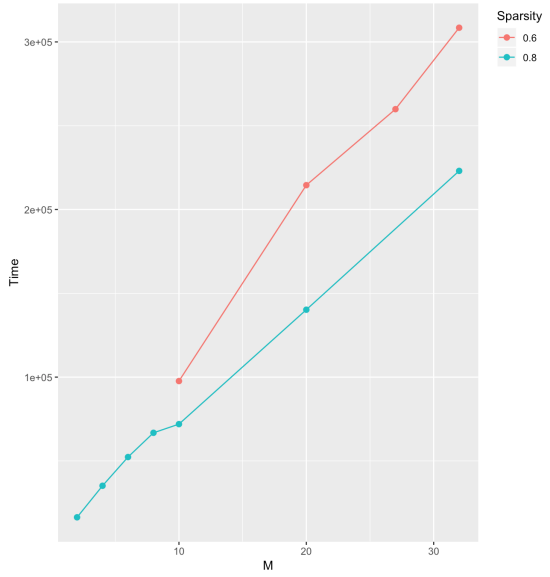

Figure 9.34: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.9.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.35 and Figure 9.36 it is observed that Space vs M graph has a greater positive slope compared to the Space vs D graph.

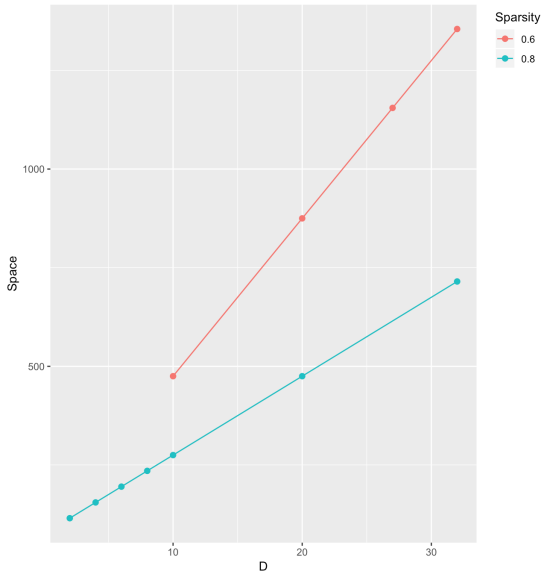

Figure 9.35: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

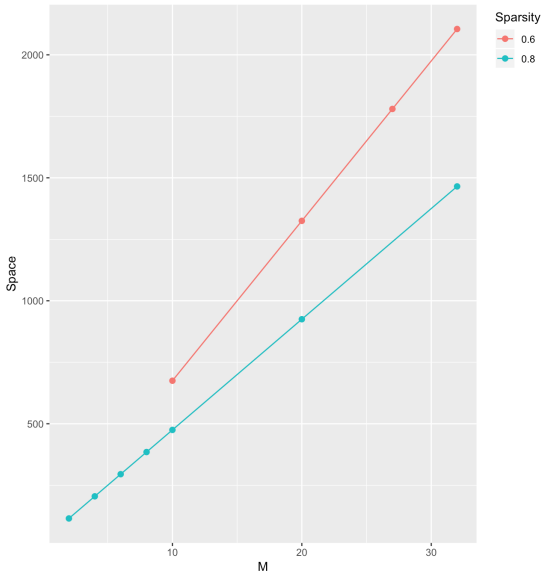

Figure 9.36: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.9.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement yxmd(with y and x being dense parameters) for performing Sparse Convolution benefits if the design parameter D is kept greater than design parameter M (considering both time and space performance).

9.10 yxdm Arrangement for Sparse Convolution

y(row index) and x(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). d and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.10.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.37 and Figure 9.38 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

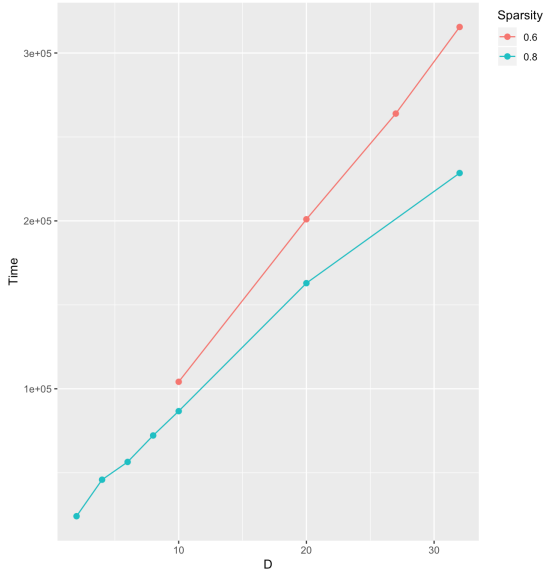

Figure 9.37: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

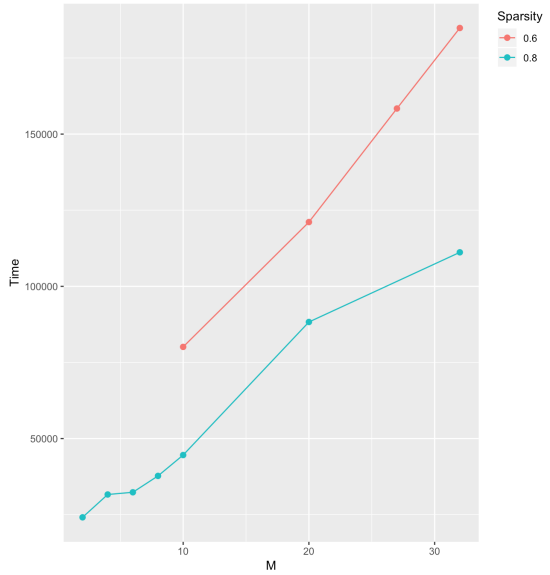

Figure 9.38: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.10.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.39 and Figure 9.40 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

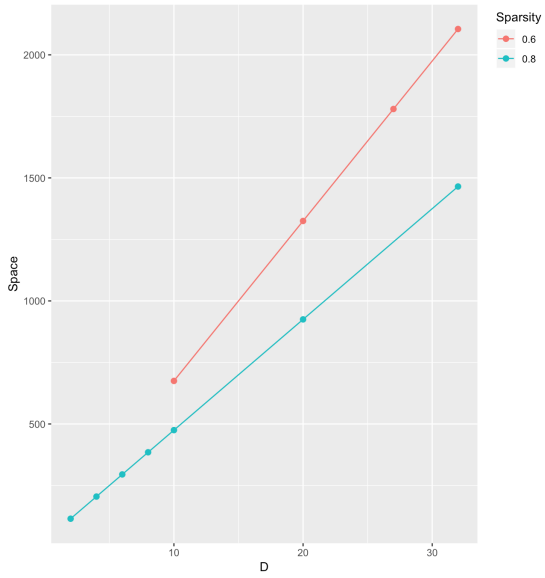

Figure 9.39: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

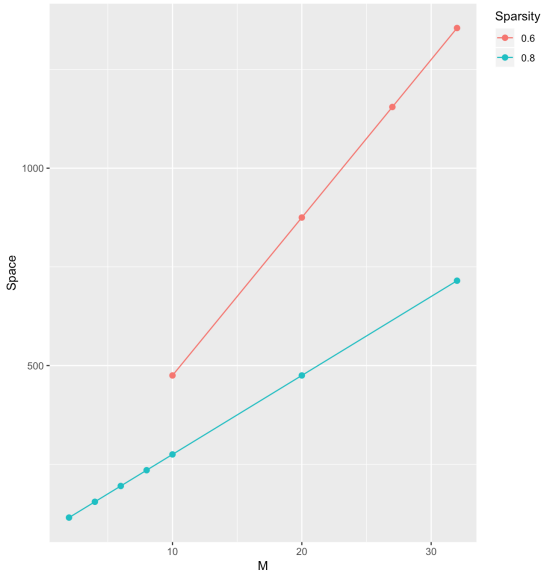

Figure 9.40: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.10.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement yxdm(with y and x being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.11 xydm Arrangement for Sparse Convolution

x(row index) and y(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). d and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.11.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.41 and Figure 9.42 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

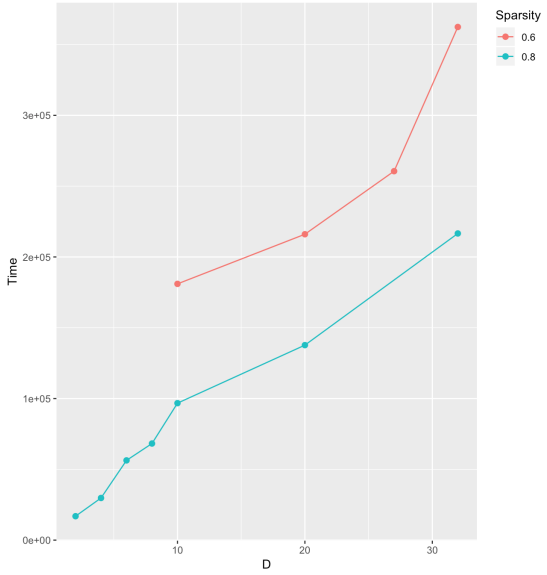

Figure 9.41: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

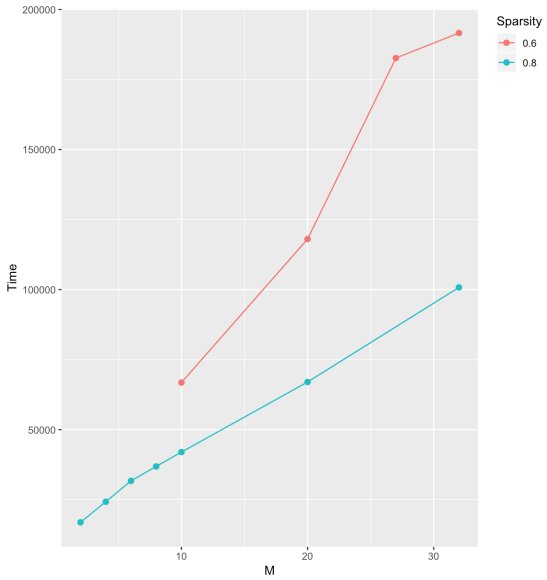

Figure 9.42: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.11.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.43 and Figure 9.44 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

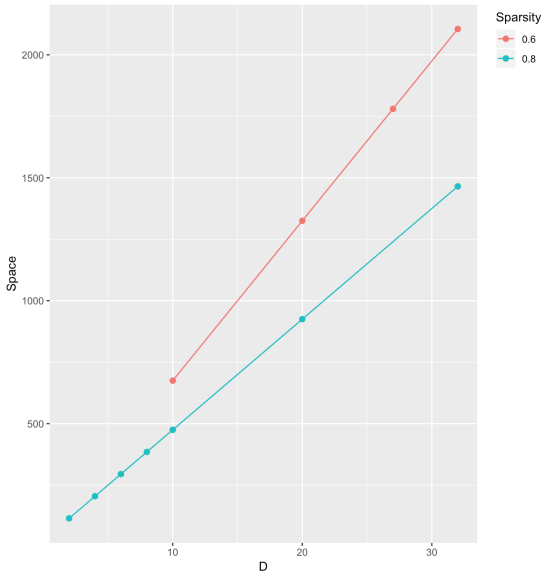

Figure 9.43: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

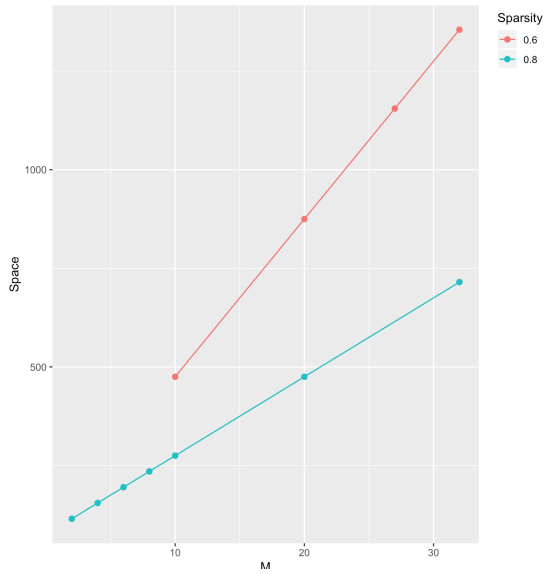

Figure 9.44: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.11.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement xydm(with x and y being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.12 dyxm Arrangement for Sparse Convolution

d(row index) and y(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). x and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.12.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.45 and Figure 9.46 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

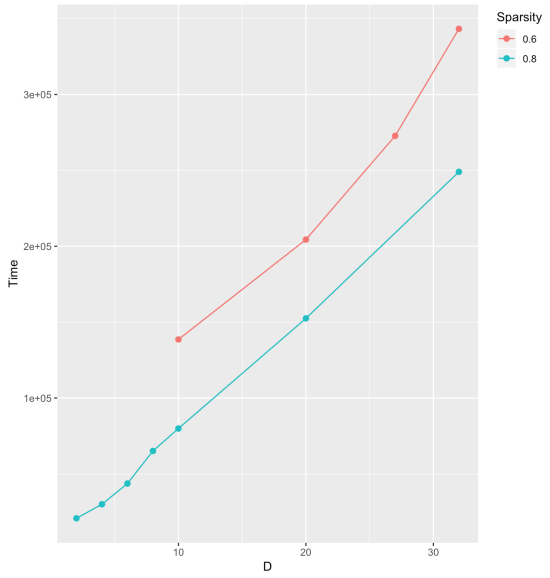

Figure 9.45: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

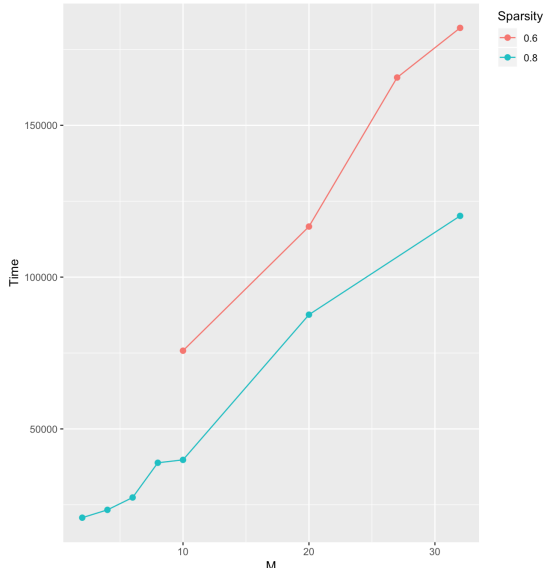

Figure 9.46: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.12.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.47 and Figure 9.48 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

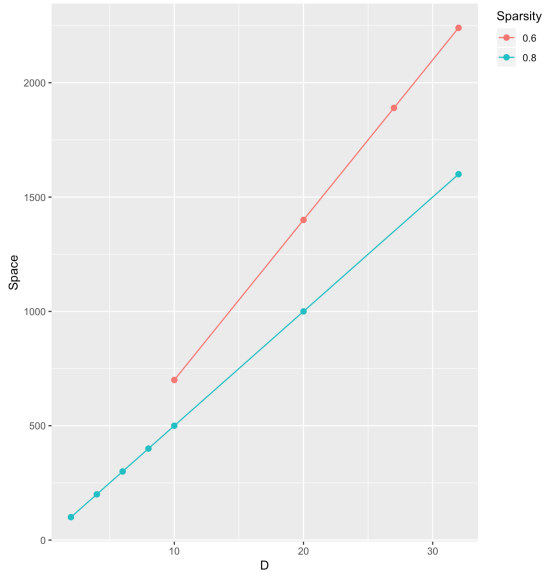

Figure 9.47: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

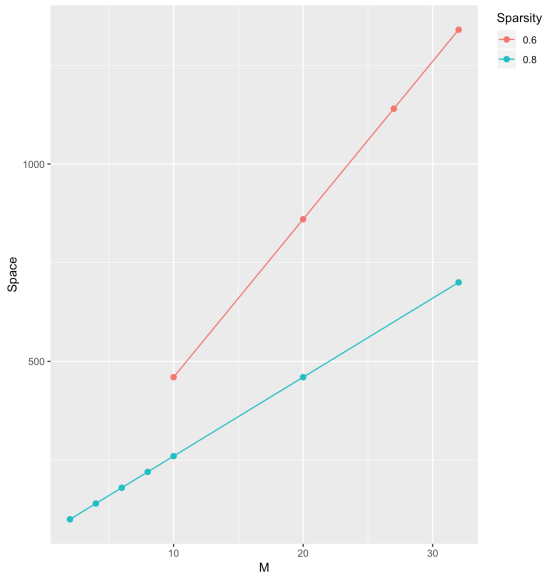

Figure 9.48: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.12.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement dyxm(with d and y being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.13 ydxm Arrangement for Sparse Convolution

y(row index) and d(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). x and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.13.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.49 and Figure 9.50 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

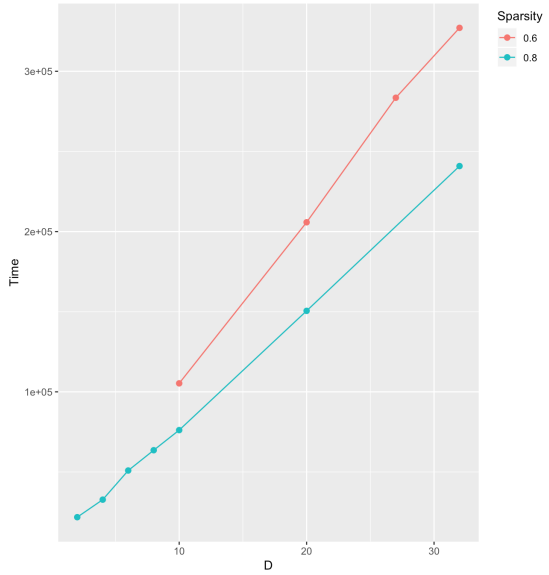

Figure 9.49: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

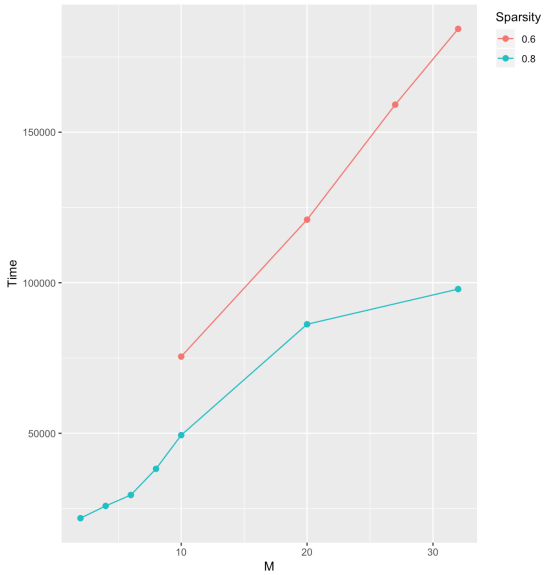

Figure 9.50: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.13.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.51 and Figure 9.52 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

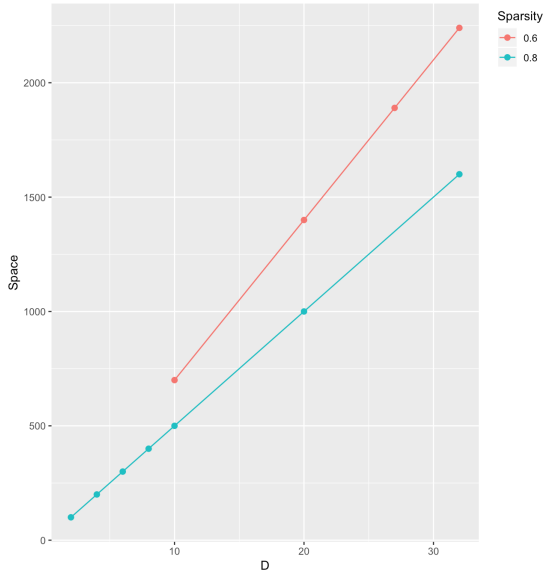

Figure 9.51: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

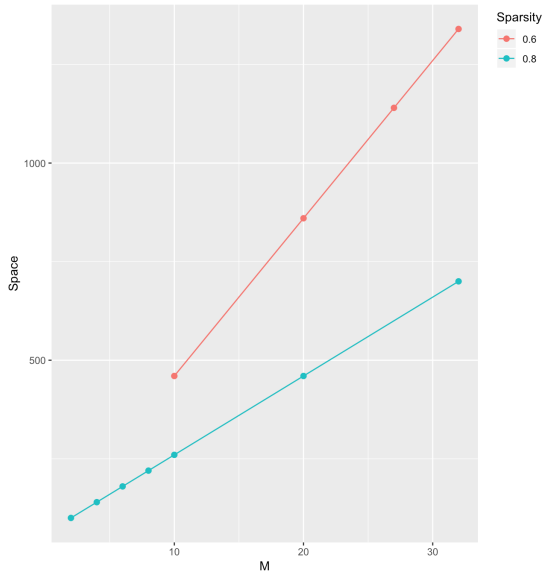

Figure 9.52: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.13.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement ydxm(with y and d being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.14 xdym Arrangement for Sparse Convolution

x(row index) and d(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). y and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.14.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.53 and Figure 9.54 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

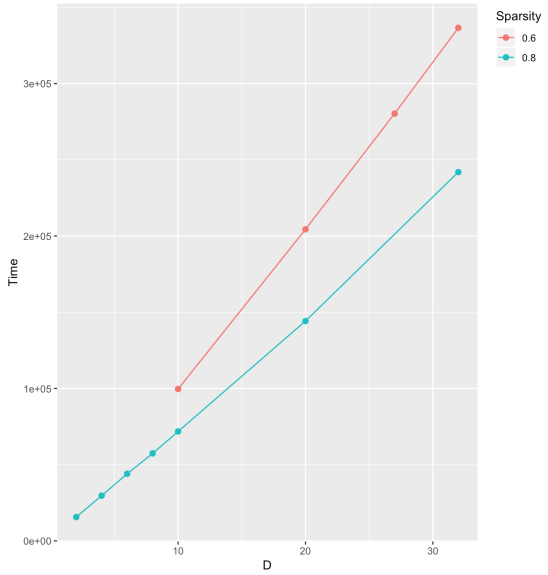

Figure 9.53: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

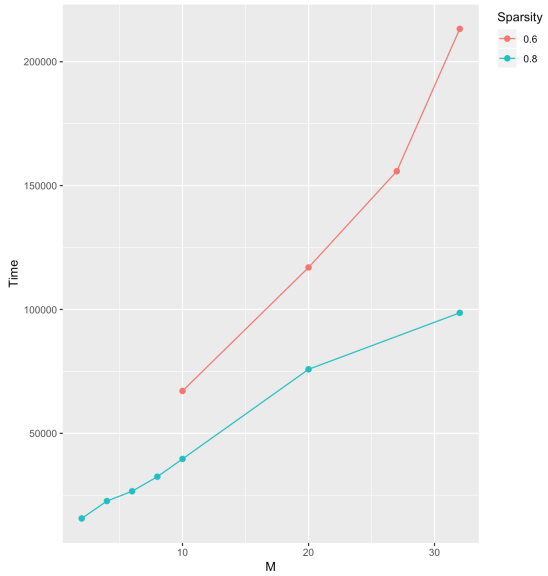

Figure 9.54: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.14.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.55 and Figure 9.56 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

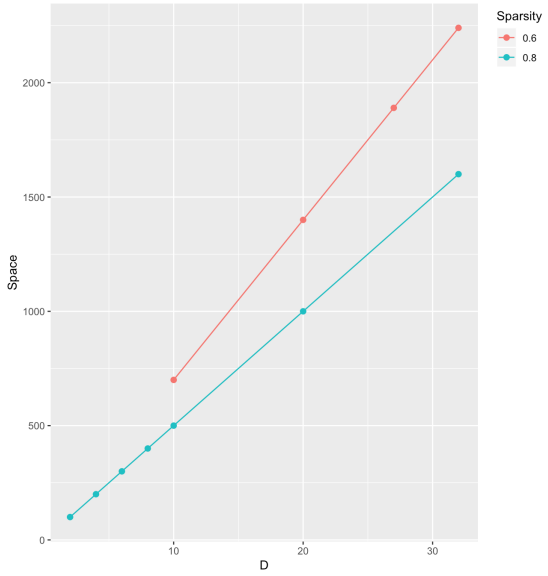

Figure 9.55: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

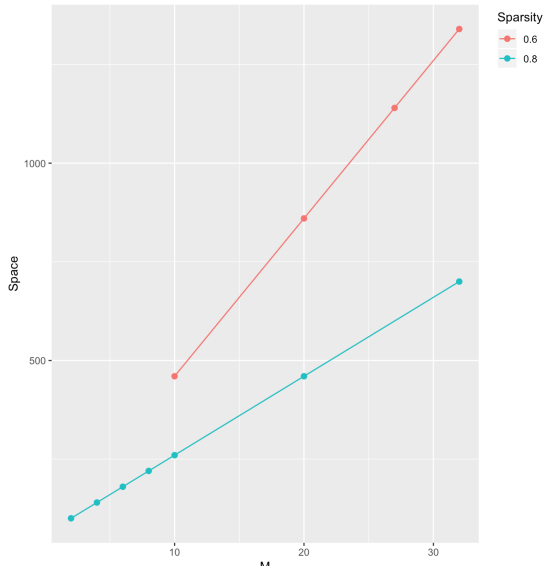

Figure 9.56: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.14.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement xdym(with x and d being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

9.15 dxym Arrangement for Sparse Convolution

d(row index) and x(column index) are the dense parameters and serve as the indexes into the Data Structure for Sparse Convolution (2-Dimensional array of CSR (Compressed Sparse Row) objects). y and m are the dimensions which have been sparsified by the Sparsify _4d algorithm.

9.15.1 Time Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.57 and Figure 9.58 it is observed that Time vs D graph has a greater positive slope compared to the Time vs M graph.

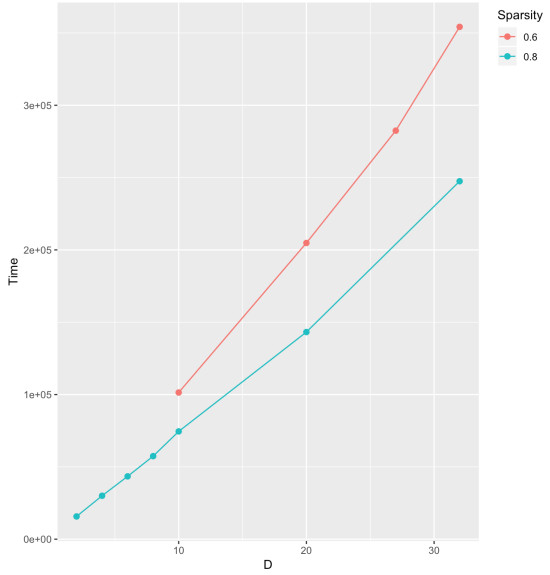

Figure 9.57: Time(nanosecond) vs D plot for Sparsity levels(0.6 and 0.8)

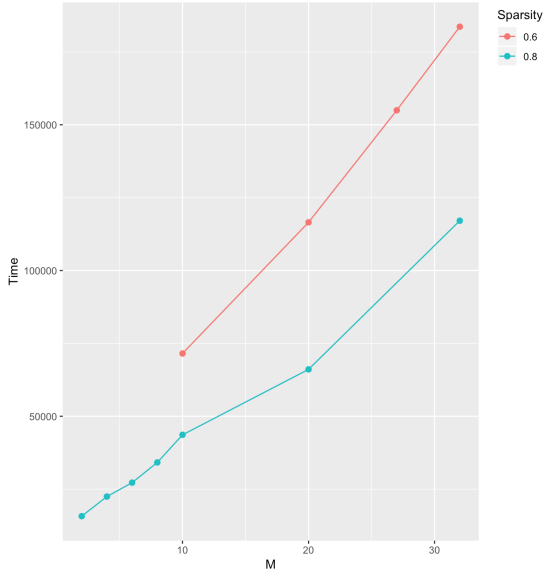

Figure 9.58: Time(nanosecond) vs M plot for Sparsity levels(0.6 and 0.8)

9.15.2 Space Performance w.r.t Multi-Channel Convolution Design Parameters(M,D)

Comparing the experimental results from Figure 9.59 and Figure 9.60 it is observed that Space vs D graph has a greater positive slope compared to the Space vs M graph.

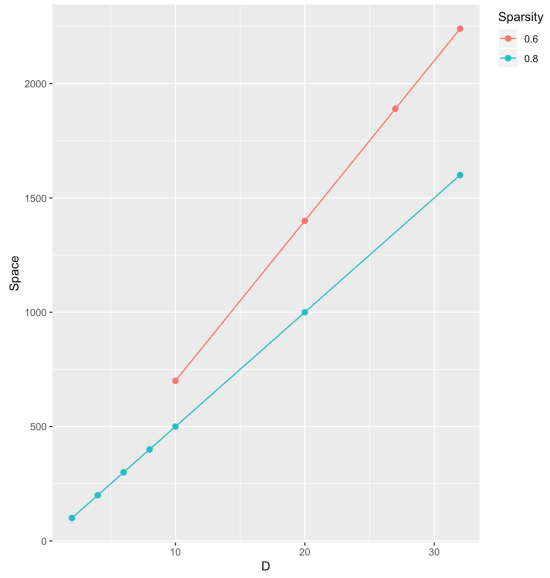

Figure 9.59: Space(number of memory units) vs D plot for Sparsity levels(0.6 and 0.8)

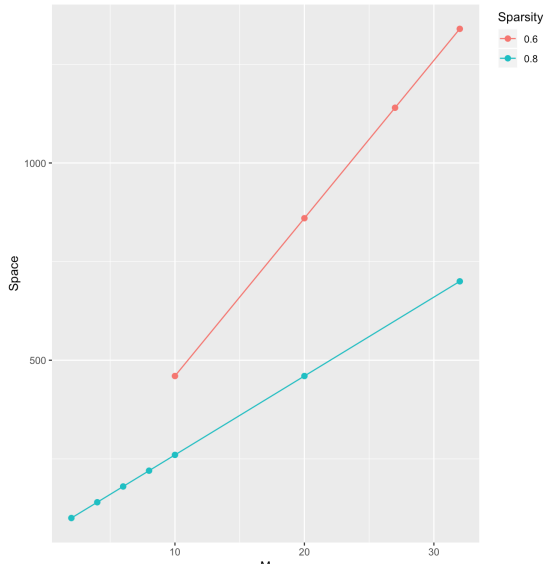

Figure 9.60: Space(number of memory units) vs M plot for Sparsity levels(0.6 and 0.8)

9.15.3 Conclusion

Based on the experimental results it is observed that the Data Structure with arrangement dxym(with d and x being dense parameters) for performing Sparse Convolution benefits if the design parameter M is kept greater than design parameter D (considering both time and space performance).

10 Conclusion and Future Work

15 of the possible 24 different variations of the Data structure from the Sparsify _4d algorithm are used to perform Sparse Convolution and their time and space performance is analyzed.There are two parameters(termed as the design parameters of the Sparse Convolution System) in common which affect both the time and space requirement for performing Sparse Convolution :

- M : One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which is the input to Sparsify _4d algorithm giving the final data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution and also determines the number of channels in the Output 3-Dimensional matrix(W*H*M)

- Depth(D) : One of the four dimensions of the Kernel 4-Dimensional Matrix(K*K*M*D) which is the input to Sparsify _4d algorithm giving the final data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution and also serves as Depth of channels from the Input 3-Dimensional Matrix(W*H*D)

The work tries to analyze what affect the changing/varying of these two parameters(M and D) has on the timing and space performance of the data structure (2-Dimensional array of CSR(Compressed Sparse Row Objects)) for performing Sparse Convolution with different arrangements of dense and sparse dimensions.

It is observed that some of the orders of traversals/ arrangements perform better when the design parameter D is greater than M while some of them perform better when the design parameter M is greater than D. This leads to concluding that depending on the magnitude of D and M certain orders of traversals /arrangements are better than the others for performing Sparse Convolution. The Table 10 below summarizes the results and should be used to pick certain orders of traversals over others depending on the magnitude of M and D.

The current implementation takes the two sparse dimensions in the innermost loops(inside the nested dense dimension loops) for performing Sparse Convolution. The Sparse Dimensions can be placed in a different arrangement for e.g the dense loops nested inside the loops for Sparse Dimensions and results can be compared to the current implementation.

The current implementation uses a single stage of compression. A multi-stage pipeline of compression can be applied to the Data in the filters to obtain a better performance on Time and Space required compared to the current implementation.

Table 10		
Order	Time Performance(Better Parameter)	Space Performance(Better Parameter)
MDXY	D	D
XYMD	D	D
DYMX	M	M
MXDY	D	D
MXYD	D	D
XMYD	D	D
YMXD	D	D
MYXD	D	D
YXMD	D	D
YXDM	M	M
XYDM	M	M
DYXM	M	M
YDXM	M	M
XDYM	M	M
DXYM	M	M

As Time and Space are the two main resource considerations of Neural Networks picking the right order of traversal for the Novel Data Structure created under Sparsify_4d algorithm will aid the process of training of resource-intensive Neural Networks.

References

[1] Hyeong-Ju Kang ,*Accelerator-Aware Pruning for Convolutional Neural Networks*,2018.

[2] Jongsoo Park,Sheng Li,Wei Wen,Ping Tak Peter Tang,Hai Li,Yiran Chen and Pradeep Dubey ,*Faster CNN's With Direct Sparse Convolutions And Guided Pruning*, ICLR 2017.

[3] Hao Li,Asim Kadav,Igor Durdanovic,Hanan Samet, Hans Peter Graf ,*Pruning Filters for efficient Convnets*, ICLR 2017.

[4] Xiaofan Xu, Mi Sun Park, Cormac Brick ,*Hybrid Pruning: Thinner Sparse Networks for Fast Inference on Edge Devices*, Movidius, AIPG, Intel,2018.

[5] Song Han,Huizi Mao, William J. Dally ,*Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, ICLR 2016.