Computation Reduction for Convolutional Neural Network Acceleration

B. Tech Project Report Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology

by

Dyuti Mangal and S Roshan

(190101035, 190101079)

under the guidance of

Dr. Hemangee K. Kapoor

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, GUWAHATI

Contents

Chapter 1

Introduction

1.1 Motivation

In the booming field of artificial intelligence, algorithms that can learn and make predictions, for instance, by creating models from training datasets, are used. Many machine learning (ML) methods, including Convolutional Neural Networks (CNNs), have recently demonstrated promise in various application areas, including speech recognition and image categorisation.

Recent improvements in the creation of ML models have significantly raised their computational and memory needs. Although deeper and bigger models perform well across the board [1], they take time, energy, and memory to execute. The efficiency (in terms of energy and execution time) of hardware accelerators in executing ML models has been recently showcased. A low inference time is crucial for use cases requiring low latency, like autonomous driving. A key factor for edge devices is energy efficiency.

Most tensor operations in ML models are simple operations. Appropriate

accelerator designs can facilitate efficient processing of these operations.

By significantly lowering the amount of computing, communication, and memory needed, ML models may be executed more efficiently. This is possible through inducing and leveraging the following:

- **Zero values (sparsity) [2]:** CNNs are often over-parameterised and thus have sparse weight matrices. Inference time and energy usage are decreased by detecting the zeros and only conducting multiply-accumulate(MAC) operations on the non-zeros.

- **Effective MAC units:** Since several billion MAC computations result in extremely high energy consumption for logical operations in CNNs, several works concentrate on the design of MAC units.

- **Early Computation Termination by Output Prediction:** Ineffective computations can be decreased through early output prediction. They examine if computations impact the subsequent layer's effective inputs. If not, these calculations are stopped.

1.2 Problem Statement

Computation reduction for acceleration of Convolutional Neural Network inference by exploiting sparsity.

1.3 Organization of this Document

The report is organised into five chapters. Chapter 2 provides a brief background on the sparsity and computation heaviness of CNNs. Chapter 3 is a literature survey that discusses acceleration techniques pertaining to sparsity exploitation and predictive computation termination. Chapter 4 goes into our proposed algorithm and the baselines to which it is compared. Chapter 5 contains the results of our approach. After that, we list the references.

Chapter 2

Background

2.1 CNNs

Convolutional neural networks (CNNs) are utilised widely in computer vision tasks. Convolutional layers (CONV) comprise the majority of a CNN architecture. Early convolutional layers extract simple features like vertical and horizontal lines, which are used by later layers to build high-level features, such as faces. The classifier, sometimes called the fully connected(FC) layer, finally decides the object's class [1].

The execution of many models is dominated by common and simple operations (e.g., general matrix multiplication, convolution, multilayer perceptron). For instance, convolutions dominate runtime and energy consumption, accounting for nearly 90% of all processing.

2.2 Sparsity

Sparsity in the context of hardware acceleration refers to inputs (weights, activations, gradients) having a value of zero.

2.2.1 Sources of Sparsity

- The ReLU activation function in CNNs zeroes out negative inputs.
- Dropout layers that are used to avoid overfitting.
- Since just a small portion of the vocabulary is contained in each document, text corpus in Natural Language Processing(NLP) applications results in significant sparsity.
- Object detection has sparse inputs since only some parts of the input image frame are relevant.
- Pruning strategies eliminate pointless weights, reduce model overfitting, and preserve classification accuracy.

It is possible to eliminate more than 60% of the weights in CONV layers and 90% of the weights in FC layers. Compact models like MobileNetV2 and EfficientNetB0 have weight sparsities ranging from 50% to 93%, which reduces the number of MAC operations by approximately 3.3×. Pruning can eliminate 80% of the weights in NLP tasks [3].

2.2.2 Advantages of Sparsity

Sparsity allows:

- Removing useless calculations, i.e., reduces energy overhead and execution time by skipping processing of zero values.
- Storage is reduced by recording just NZ values, thus reducing the frequency of SRAM accesses which are incredibly energy-intensive.
- Improving speedup due to reduced communication requirements.

Utilizing all types of sparsity is the aim of sparsity exploitation, which aims to significantly minimise computation, transmission, and storage of zeros while optimising area and energy costs [4].

2.3 Ineffective Computations

Most sparsity-aware accelerators perform ineffective computations in the convolution operation. ANT[5] demonstrates that outer product-based accelerators compute Redundant Cartesian Products that do not correspond to any output pixel value. Most recent SOTA CNNs use the ReLU activation function which filters out negative inputs. Thus, convolution operations that lead to a negative input need not be computed. Similarly, max-pooling layers filter out non-maximal input values. Thus, skipping such computations can potentially lead to performance gains.

2.4 Dataflow

A dataflow is the way a model layer is executed over space and time on the hardware resources. It has a significant impact on resource utilisation, data

reuse, etc. and consequently, execution time and energy consumption. The standard way to classify dataflows is by noting what kind of data remains stationary in the PEs while other data iterates over it.

With this background, we shall explore some acceleration techniques in the next chapter.

Chapter 3

Literature Review

3.1 Sparsity Based Acceleration

3.1.1 SCNN

SCNN[6] employs the ReLU-generated zero-valued activations and pruned sparse weights to enhance performance and energy efficiency. SCNN achieves this by implementing an innovative dataflow approach that compresses and encodes the sparse weights and activations. This approach eliminates the need for additional data transfers and reduces storage requirements, thereby boosting efficiency and effectiveness. These weights and activations can also be delivered to the multiplier array efficiently due to the SCNN dataflow, where they are extensively reused. Additionally, a novel accumulator array is used to perform the accumulation of multiplication products. SCNN uses the Planar Tiled-Input Stationary-Cartesian Product-sparse (PT-IS-CP-sparse) dataflow to utilize the sparsity in both weights and activations. With the help of this

strategy, SCNN makes use of an outer product architecture that tries to augment input data reuse among a collection of distributed processing elements. Additionally, it enables the usage of a dense compressed representation for both activations and weights across the entire processing flow.

3.1.2 SparTen

SparTen[7] points out several issues in SCNN, and improvises. SCNN takes advantage of two-sided sparsity to outperform dense designs in terms of performance and energy. The sparse vector-vector dot product, which is a vital primitive in sparse vision models, becomes ineffective if the format used by SCNN is employed. To perform the primitive, sparse vectors must be combined using a dot product operation that links them based on position as the key and a single value field that locates and accesses non-zero elements in matching positions. However, SCNN is not preferable for strided convolutions due to overheads.

SparTen delivers efficient computation of the primitive by offering two-sided storage and sparse computation for both inputs. SparTen uses a software technique called greedy balancing to address load imbalance. Based on their sparsity, kernels are grouped in 2 ways: a software-hardware version that uses finer-grain density and a software version that uses whole-filter density. SparTen outperforms SCNN, one-sided sparse architecture, and dense architecture, by around 3×, 1.8×, and 4.7×, respectively.

3.1.3 Summary

While performing our literature survey, we noticed specific trends in how accelerators have changed over the years. The earliest works, such as Eyeriss, focused on measuring the energy and speedup values by fabricating chips instead of just simulation results. Early works only use weight sparsity as it is static and easy to exploit. Recently, exploitation of two-sided sparsity has gained attention. Structured sparsity has begun to gain attention along with sparse training algorithms to enforce structured sparsity because it has less area and power overheads. As opposed to training acceleration, inference acceleration is a well studied problem. However, training acceleration has come into focus lately. We have moved from scalar to tensor PEs as data naturally comes in blocks, so it makes sense to have PEs that process blocks as in tensors or vectors rather than numbers. Novel dataflows are also being proposed (e.g., early works were channel-last until SNAP came along and showed the benefits of channel-first). Also, as edge and mobile devices become more popular, people are focusing on accelerators for these applications that need low area and power.

The table below summarises sparsity based accelerators based on several key design parameters.

	Use-case	Sparsity Type	Sparsity Support	Data Compression	Dataflow	PE Type
SCNN[6]	Inference	Unstructured	Two-sided	RLC	Input Stationery	Vector
SparTen[7]	Inference	Unstructured	Two-sided	Bitmap	Output Stationery	Scalar
SNAP[8]	Inference	Unstructured	Two-sided	COO-1D	Channel-first	Vector
Eyeriss[9]	Inference	Unstructured	One-sided	RLC	Row Stationery	Scalar
CANDLES[10]	Inference	Unstructured	Two-sided	Pixel-first	Channel-first	Vector
S2TA[11]	Inference	Structured	Two-sided	Bitmap	Output Stationery	Tensor
CSP[12]	Inference	Structured	One-sided	CSR	IpOS/ IpWS	Vector/ Scalar
ANT[5]	Training	Unstructured	Two-sided	CSR	Dataflow Agnostic	Vector

Table 3.1: Overview of architectural parameters for the explored accelerators.

3.2 Prediction Based Acceleration

We now look at acceleration from another angle. Accelerators designed to handle sparse data aim to optimize performance by disregarding zero-valued activations and parameters, which are typically a result of the ReLU activation function. However, the alternate approach of skipping the ineffectual outputs is also an option. In fact, it is more advantageous to retain the zero-valued inputs and instead skip the ineffective outputs during the computation, since the number of zero-valued outputs tends to be higher than that of inputs. Many of the zero-valued outputs are automatically filtered out as the inputs are passed through the pooling layer in the subsequent computation. In particular, ineffective computations can be reduced via Early Computation Termination by Output Prediction. We review a popular prediction-based accelerator:

3.2.1 SnaPEA

This work takes the activation function into consideration (which is ReLU). This involves rearranging the weights in such a way as to know if the computation's overall result is negative at the earliest. This paper proposes a method that leads to reduction of those computations by taking into consideration both algorithmic structure and information about runtime of CNNs. Predictive Early Activation Technique has two modes, namely Exact mode and Predictive mode. We focus on the Exact mode. In this mode, the computation of convolutional operations are cut short if a particular iteration predicts the final output of a computation to be negative during runtime. In this way, the ineffectual computations are pruned. SnaPEA statically re-orders weights based on their signs. PEs of its SnaPEA architecture contain prediction activation units (with each MAC), which check the signbit of the partial summation and raise a termination signal to notify the controller. The algorithm is described below:

1. Perform convolutional operation on positive weights

2. Sort the negative weights in decreasing order of their magnitudes

3. In each cycle, perform convolutional computation using one negative weight in order. If at any stage, the predicted activation becomes negative then terminate.

When compared to Eyeriss, SnaPEA in the exact mode yields 28% speedup (maximum of 74%) and 16% (maximum of 51%) energy reductions across

various modern CNNs without affecting their classification accuracy. Even for SqueezeNet—a statically pruned convolutional neural network—SnaPEA yields 1.3× and 1.14×. These savings for SqueezeNetshow that static pruning techniques are complementary to the dynamic approach of SnaPEA.

3.3 Conclusion

Although there are several accelerators that exploit sparsity, and several that exploit computation reduction, these two approaches are treated orthogonally. ComPreEND[13], a prediction-based accelerator, remarks that applying the scheme of skipping multiplications for a zero input to the accelerator can be another future research direction. Existing literature suggests that these methods can augment each other and lead to even greater performance gains, which is the motivation behind our proposed algorithm.

Chapter 4

Baselines and Proposed Algorithm

4.1 Baseline Algorithms

In this section, we discuss the algorithms against which our proposals are evaluated. As our broad goal is to incorporate sparsity into predictive algorithms, our baselines cover all combinations of sparsity(zero-aware vs. non zero-aware) and optimisation approach(none vs. prediction-based vs. sparsity-based). Thus, we decided to use the baselines described below.

4.1.1 nn.conv2d

This is the familiar inbuilt PyTorch convolution function. It simply takes the image and filters, and convolves them using matrix multiplication, without any regard to zero values in the data.

4.1.2 Compressed Convolution

This algorithm takes image and filters, and then stores only the non-zero values in a compressed format. The computation is thus done only on the non-zero values. Compression is done by storing the non-zero values as a list, and an Index Vector that stores the index of each non-zero element. There are several compression formats; an analyis of their memory reduction is given in the below image.

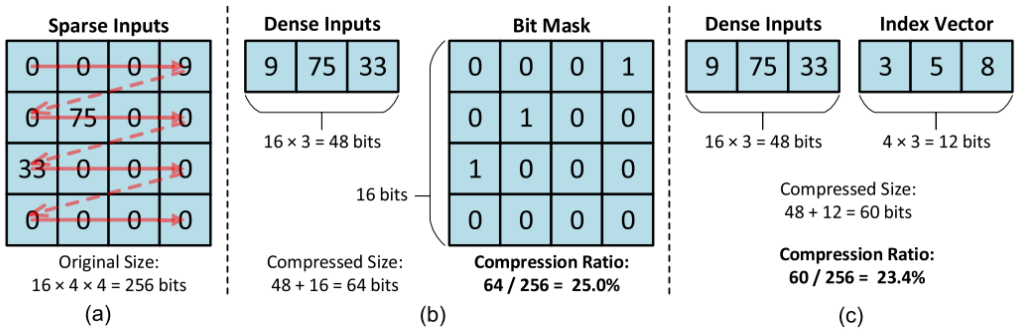


Fig. 4.1: Common Indexing Styles: (a) Original sparse data (b) Compressed data with Bitmask (c) Compressed data with Index vector

4.1.3 SnaPEA

This algorithm is the Exact Mode of SnaPEA(which theoretically leads to no loss in accuracy). It takes as input image and filters and applies the predictive algorithm of SnaPEA. It does not pay any consideration to zero values in the data.

4.2 Proposed Algorithms

Our proposal has two variations:

- **One-sided sparsity**: Computations are skipped only on the non-zero weights.

- **Two-sided sparsity**: Computations are skipped on the non-zero weights as well as non-zero activations.

As mentioned in the literature survey, whether to exploit sparsity in one or both inputs is a design parameter of accelerators. Implementing both variations gives us insight into whether exploiting sparsity in both inputs is beneficial as opposed to the one-sided version; it also helps us quantitatively make comparisons. Both our proposals are built on top of the 3^{rd} baseline, i.e., the Exact Mode of SnaPEA. Since the Exact mode does not result in accuracy loss, our proposals, too, should not result in any drop in classification accuracy.

As discussed in the previous chapter, combining the two most common lines of attack ie. prediction and sparsity, is an under-explored problem. We aim to explore that research direction in order to check if these two orthogonal approaches are compatible with each other.

4.2.1 Proposal 1

Algorithm 1: Predictive with One-sided Sparsity

```
Input: image, all_filters
Output: out[filter_count, height_out, width_out]
Data: stride, padding, bias
Function compute_pixel(comp_wt, r, c, bias):
    out_cell = 0
    pos_wt = comp_wt.get_pos_weights()
    neg_wt = comp_wt.get_neg_weights()
    sort(neg_wt)
    for w in pos_wt do
        out_cell += w*image[k][r+i][c+j]
        /* k, i, j are indices of weight value w                    */
    out_cell += bias
    for w in neg_wt do
        if out_cell < 0 then
            break
        out_cell += w*image[k][r+i][c+j]
        /* k, i, j are indices of weight value w                    */
    return out_cell

Function compute_filter_conv(comp_wt):
    Initialise out_channel[height_out, width_out]  // one channel of output
    for r ← 1 to valid_rows do
        for c ← 1 to valid_cols do
            Compute r_out, c_out                   // output pixel for (r,c)
            out_channel[r_out][c_out] = compute_pixel(comp_wt, r, c, bias)
    return out_channel

Function Main:
    for i ← 1 to filter_count do
        comp_wt = get_compressed_weights(all_filters[i])
        out[i] = compute_filter_conv(comp_wt)
    return out
```

4.2.2 Proposal 2

Algorithm 2: Predictive with Two-sided Sparsity

```
Input: image, all_filters
Output: out[filter_count, height_out, width_out]
Data: stride, padding, bias
Function compute_pixel(comp_wt, r, c, bias):
    out_cell = 0
    pos_wt = comp_wt.get_pos_weights()
    neg_wt = comp_wt.get_neg_weights()
    sort(neg_wt)
    for w in pos_wt do
        if image[k][r + i][c + j] == 0 then
            continue
        out_cell + = w*image[k][r+i][c+j]
        /* k, i, j are indices of weight value w                    */
    out_cell + = bias
    for w in neg_wt do
        if out_cell < 0 then
            break
        if image[k][r + i][c + j] == 0 then
            continue
        out_cell + = w*image[k][r+i][c+j]
        /* k, i, j are indices of weight value w                    */
    return out_cell

Function compute_filter_conv(comp_wt):
    Initialise out_channel[height_out, width_out]  // one channel of output
    for r ← 1 to valid_rows do
        for c ← 1 to valid_cols do
            Compute r_out, c_out                   // output pixel for (r,c)
            out_channel[r_out][c_out] = compute_pixel(comp_wt, r, c, bias)
    return out_channel

Function Main:
    for i ← 1 to filter_count do
        comp_wt = get_compressed_weights(all_filters[i])
        out[i] = compute_filter_conv(comp_wt)
    return out
```

Chapter 5

Results, Conclusion & Future Work

5.1 Implementation

In this section, we comprehensively list our attempts at implementation. All the coding was done in Python using the popular PyTorch framework. We initially tried to run pre-trained VGG16 on the ImageNet dataset. However, the accuracy was quite low, and retraining the model required computational resources that we do not have. Thus, we chose to run LeNet on the MNIST dataset and AlexNet on the CIFAR10 dataset. As AlexNet was designed to run on ImageNet, we used transfer learning to train it on CIFAR10 and obtained 81.91% accuracy. LeNet was designed for MNIST(a relatively simple dataset), so the accuracy was 98.8%. However, on analysing the weight sparsity of both models across various CONV layers, we observed that the weights were not sparse. To enable analysis across various sparsity levels, we carried out weight

pruning using PyTorch's pruning functionality. We experimented with two pruning settings:

1. **Local Pruning:** In this scheme, we specify a sparsity level. All CONV layers are individually pruned such that each layer reaches the specified sparsity level.

2. **Global Pruning:** In this scheme, we specify a sparsity level for the entire model. The weights across all CONV layers are collectively pruned so that the average sparsity reaches the specified level. However, each layer may have different sparsity.

In order to choose which scheme to use, we performed both kinds of pruning and measured the accuracy drop. It was observed that for both models, global pruning was better. Further, we gradually increased the level of sparsity in order to measure how much pruning is possible before the accuracy drops too much. Based on these experiments, we decided to go with 0%, 25%, 50% and 75% global sparsity. Several key quantities are measured: the activation sparsity across various layers; the accuracy of the models as the sparsity is increased and the algorithm is varied; the layer-wise number of multiplications. However, we do not include accuracy values for AlexNet, since doing so would require us to run the model on a sufficiently large number of images, which was not allowed by our computational capabilities.

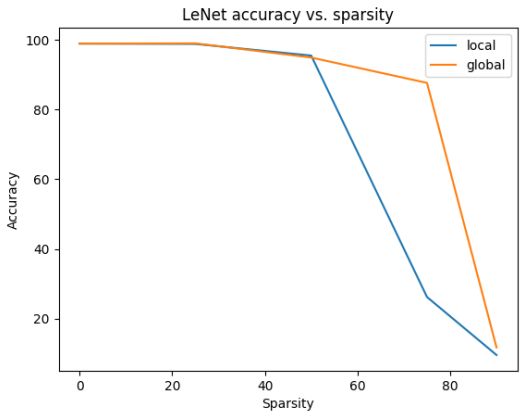


Fig. 5.1: LeNet

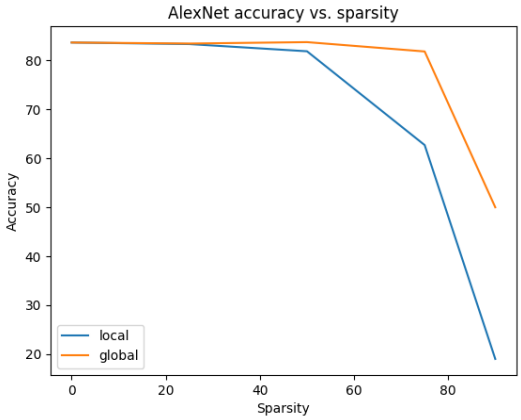


Fig. 5.2: AlexNet

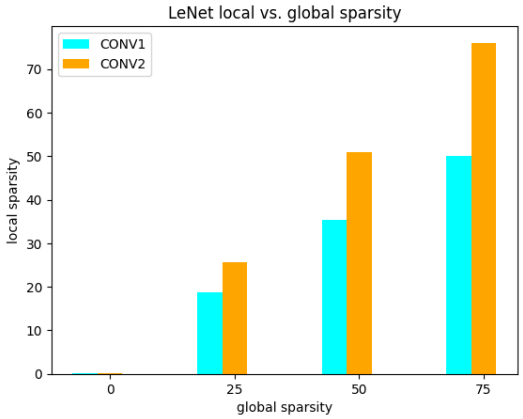


Fig. 5.3: LeNet

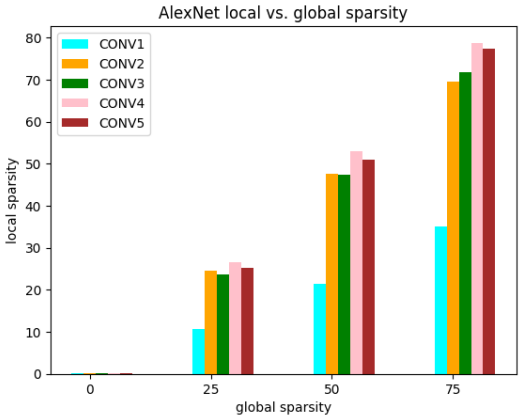
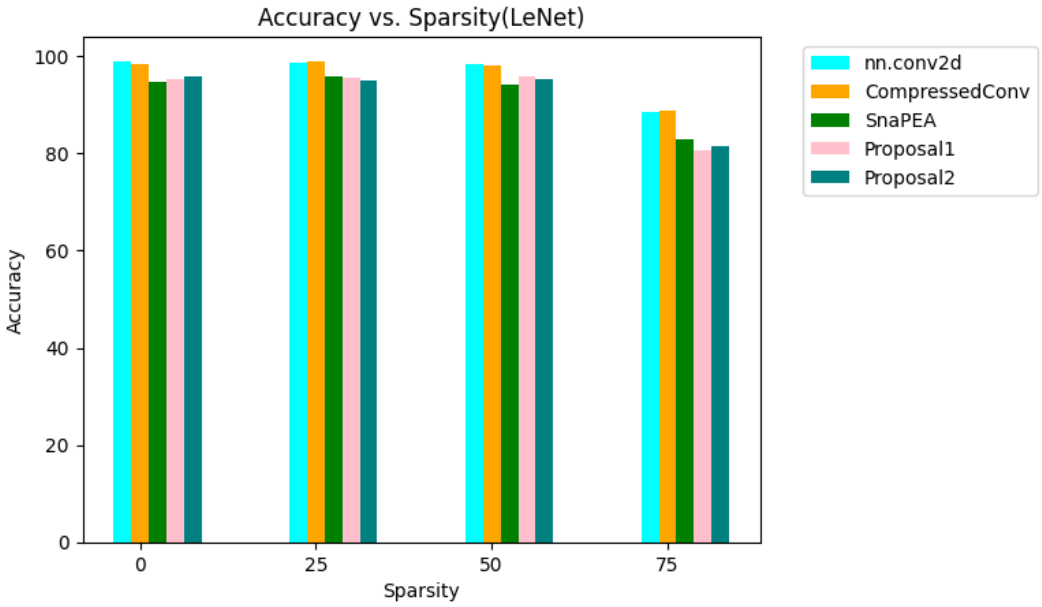


Fig. 5.4: AlexNet

5.2 Results

5.2.1 LeNet



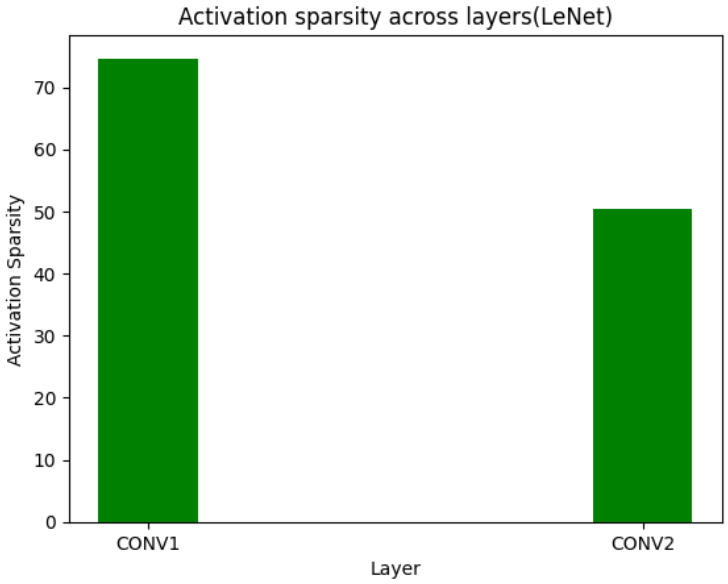


Fig. 5.5

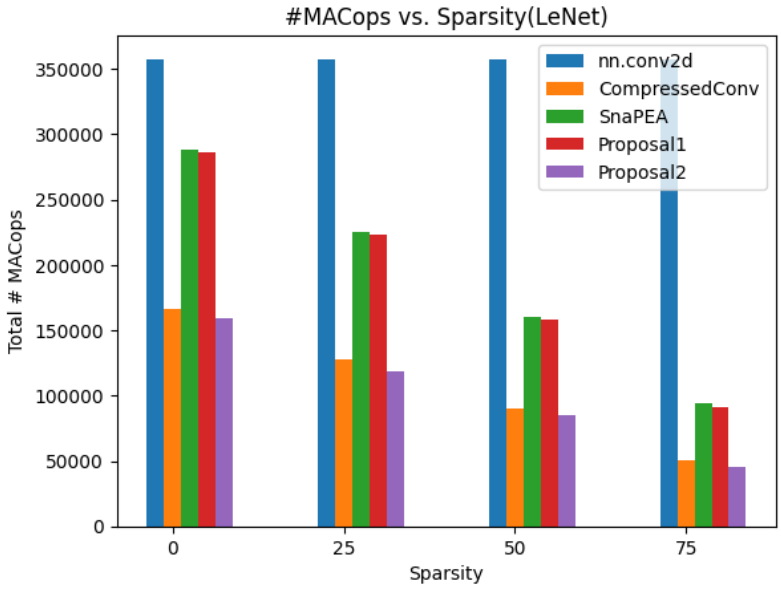


Fig. 5.6

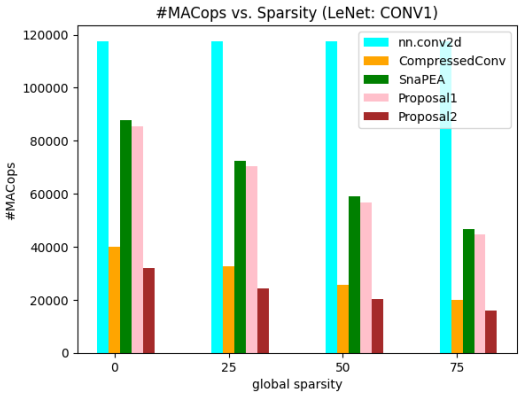


Fig. 5.7: CONV1

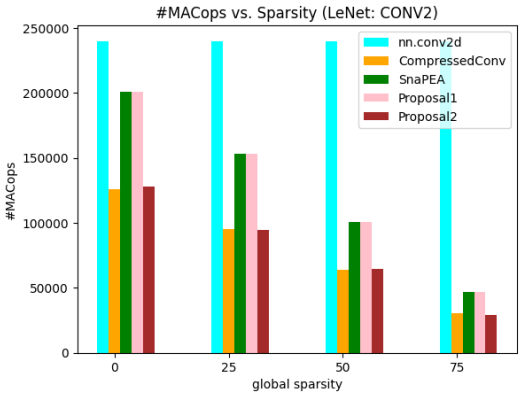


Fig. 5.8: CONV2

5.2.2 AlexNet

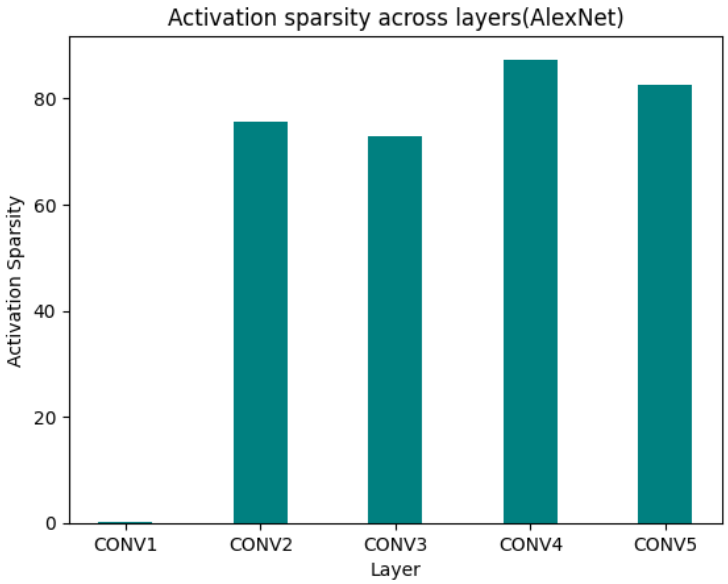


Fig. 5.9

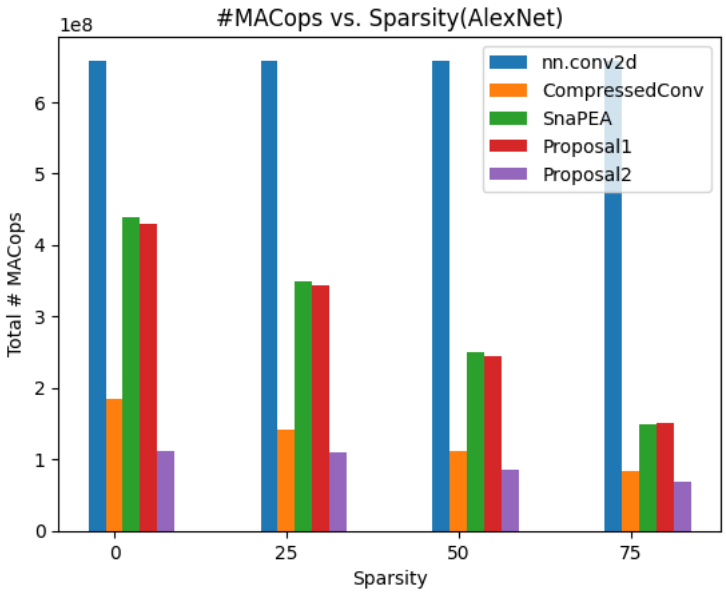


Fig. 5.10

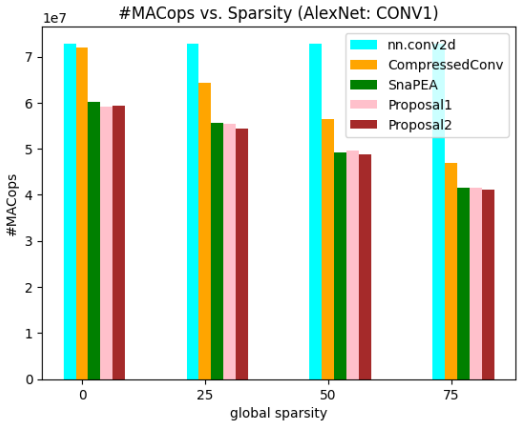


Fig. 5.11: CONV1

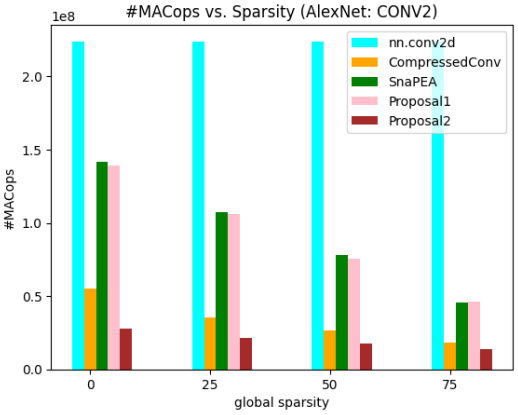


Fig. 5.12: CONV2

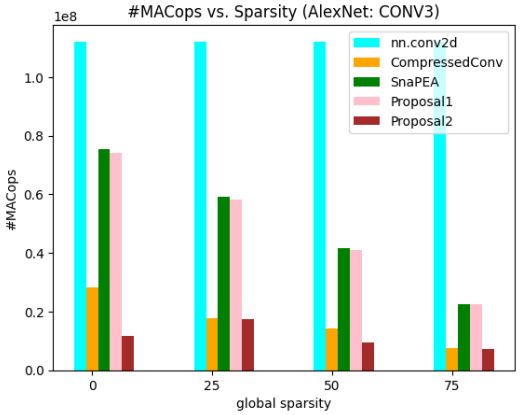


Fig. 5.13: CONV3

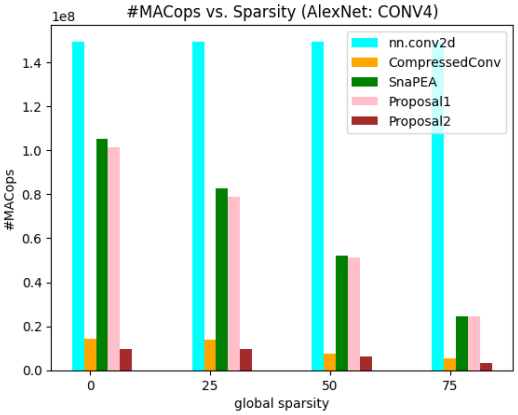


Fig. 5.14: CONV4

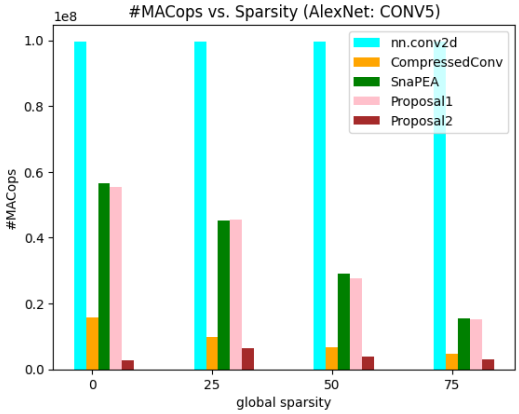


Fig. 5.15: CONV5

5.3 Observations

We notice that there are slight variations in the accuracies, which goes against our original notion that there should be no accuracy drop. However, this is due to two reasons: the first baseline is tested on the entire test dataset, whereas the rest are tested on random subsets. Further, LeNet has batch normalisation layers before the ReLU layers. This means that the output of the convolution operation is slightly modified before it is sent to ReLU. Outputs of the CONV layer that are positive, could become negative after normalisation, resulting in them getting filtered out. We also observe high values of activation sparsity for both models.

5.4 Conclusion

We hypothesised that combining sparsity and prediction would lead to greater gains(in terms of the number of MAC operations) than individually using these methods. Our experiments indicate that our hypothesis is true. We observe that our second proposal beats the first proposal across all layers and sparsity values for both AlexNet and LeNet. This confirms our intuition that exploiting two-sided sparsity enables high performance. In terms of the total number of multiplications across all layers, Proposal 2 beats all the baselines, outperforming nn.conv2d, CompressedConv, and SnaPEA by 5.26×, 1.23×, and 2.52× respectively.

5.5 Future Work

The compression scheme utilised; although memory efficient, leads to complex index matching logic. Further work could attempt to mitigate this issue.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," 2017. [Online]. Available: https://arxiv.org/abs/1703.09039

[2] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," 2016. [Online]. Available: https://arxiv.org/abs/1608.03665

[3] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," 2017. [Online]. Available: https://arxiv.org/abs/1710.01878

[4] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[5] J. S. Lew, Y. Liu, W. Gong, N. Goli, R. D. Evans, and T. M. Aamodt, "Anticipating and eliminating redundant computations in accelerated sparse training," in *Proceedings of the 49th Annual International*

Symposium on Computer Architecture, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 536–551. [Online]. Available: https://doi.org/10.1145/3470496.3527404

[6] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 27–40.

[7] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 151–165. [Online]. Available: https://doi.org/10.1145/3352460.3358291

[8] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "Snap: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 636–647, 2021.

[9] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[10] S. Gudaparthi, S. Singh, S. Narayanan, R. Balasubramonian, and V. Sathe, "Candles: Channel-aware novel dataflow-microarchitecture co-design for low energy sparse neural network acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 876–891.

[11] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 573–586.

[12] E. Hanson, S. Li, H. H. Li, and Y. Chen, "Cascading structured pruning: Enabling high data reuse for sparse dnn accelerators," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 522–535. [Online]. Available: https://doi.org/10.1145/3470496.3527419

[13] N. Kim, H. Park, D. Lee, S. Kang, J. Lee, and K. Choi, "Compreend: Computation pruning through predictive early negative detection for relu in a deep neural network accelerator," *IEEE Transactions on Computers*, vol. 71, no. 7, pp. 1537–1550, 2022.