CS498 Bachelor’s Thesis Project
Mid Semester Evaluation
Accelerator Design for Machine Learning

Supervisor: Prof. Hemangee K Kapoor

Saurabh Baranwal (180101072)
Amey Varhade (180101087)

Contents

- Domain and Problem Formulation
- Literature Survey
- Limitations
- Proposed Solution
- Further Work
- References

Domain and Problem Formulation

Accelerators

- Hardware acceleration is when specific computing tasks are run on specialized hardware components within the system, enabling greater efficiency than a general-purpose CPU alone.

- They usually have novel designs and typically focus on low-precision arithmetic, better dataflow architectures or in-memory computing capability.

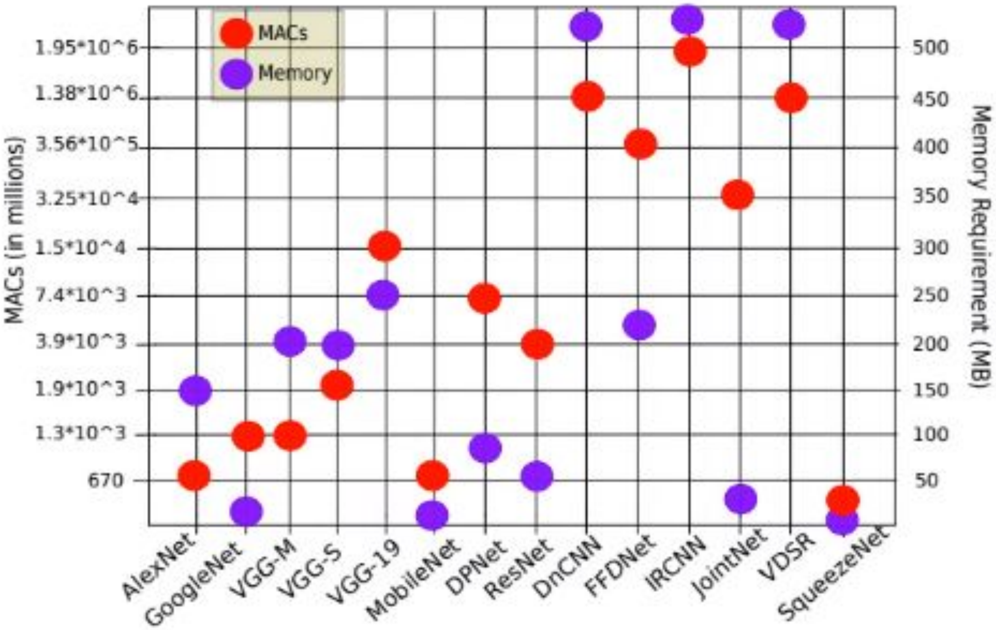
- Machine Learning Accelerators are the specialized hardware components designed to improve the efficacy of machine learning-based applications

Accelerator Design for Machine Learning

- Useful for but not restricted to computer vision, artificial neural networks, and other data-driven or sensor-driven tasks.
- Graphic Processing Units (GPUs) were traditionally designed to render 3D Graphics and perform other image, video operations, however they also serve as good machine learning accelerators.
- Google has introduced highly specialized Tensor Processing Units (TPUs) recently.

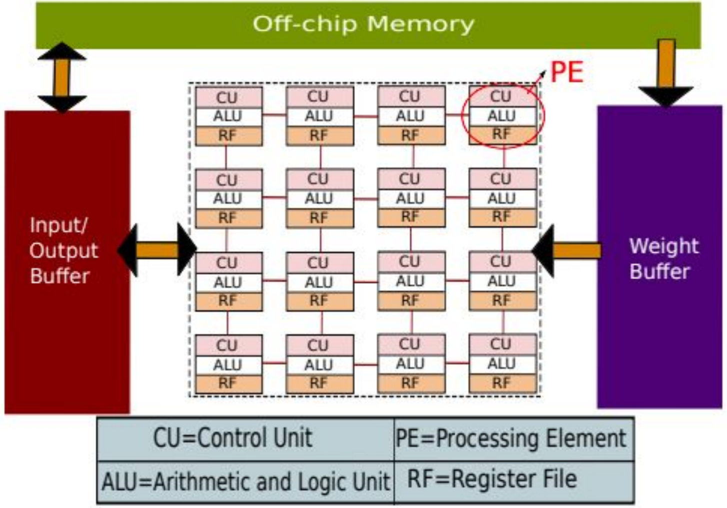
Motivation and Background

- ❏ The amount of data that needs to be processed is increasing at the rate of **1000x** per decade.
- ❏ Since 2012, 3000x increase in the computational time, and 10x increase in memory requirements.
- ❏ In CNNs, more than half of the computation time is of the **convolutional layers**.



Hence, the need to efficiently run these networks is urgent.

Motivation and Background



(a)

Memory Hierarchy	Data movement energy w.r.t ALU
RF (highest level)	1x
Inter-PE network	2x
Global Buffer	6x
DRAM (lowest level)	200x

(b)

Every CNN accelerator can be abstracted in the form of a vanilla accelerator architecture.

The image shows the generic CNN architecture and the approximate energy consumption. A typical optimization problem is as follows: Minimize the latency in a given CNN architecture subject to a constraint on the amount of on-chip memory.

Literature Survey

Existing Works

We surveyed existing methods based on improvements on two key efficiency metrics.
Reduction in Computation Time & Memory Footprint.

Reduction in Time

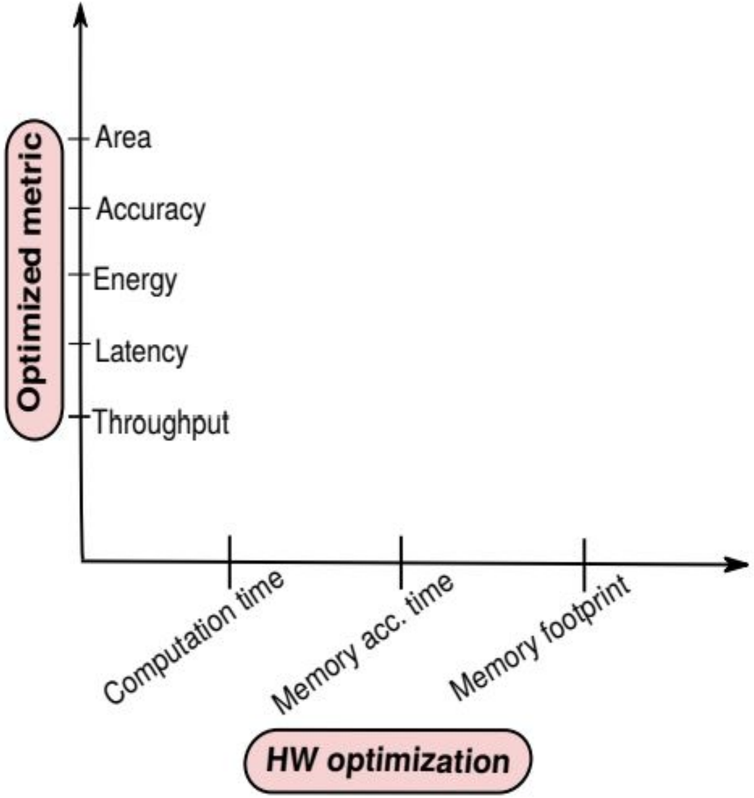
- A None-Sparse Inference Accelerator that Distills and Reuses the Computation Redundancy in CNNs. [1]
- Prediction Based Execution on Deep Neural Networks. [2]
- SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks. [3]
- Bit-Tactical: Exploiting Ineffectual Computations in Convolutional Neural Networks, Which, Why, and How. [4]
- FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks.[5]
- SCNN: An accelerator for compressed-sparse convolutional neural networks [6]

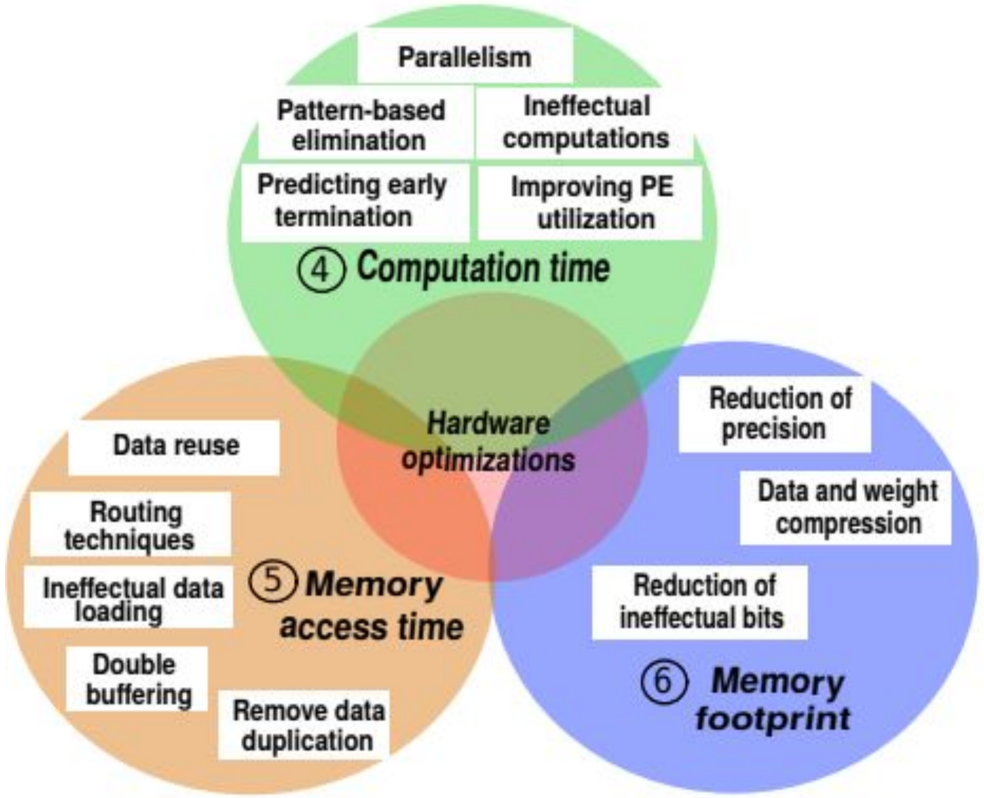
Existing Works

Reduction in Memory

- Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices. [7]
- Stripes: Bit-serial deep neural network computing. [8]
- Dynamic Stripes: Exploiting the Dynamic Precision Requirements of Activation Values in Neural Networks. [9]
- Diffy: A Déjà vu-free differential deep neural network accelerator. [10]
- Loom: exploiting weight and activation precisions to accelerate convolutional neural networks. [11]
- UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision. [12]

Architecture Classification Taxonomy





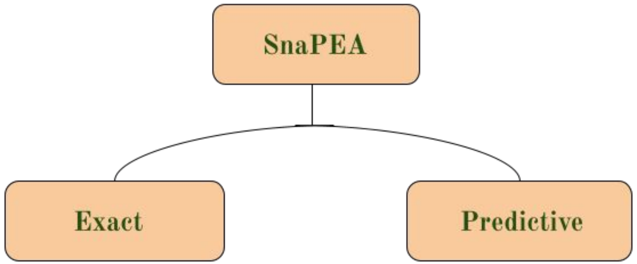
Reduction in Computation Time for CNNs

Our work would build upon the existing designs and develop algorithms and architecture based accelerators specifically for efficient inference of CNNs

- The problem we have chosen to work is reducing the **computation time** of CNN's in general.
- Paper we have chosen offers a solution that leverages a combination of **runtime information** and the **algorithmic structure** of CNNs.
- This technique cuts the computation of convolution operations short if it determines that the output will be negative.

SnaPEA[3] Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks

- Predictive Early Activation Technique has two modes, namely Exact mode and Predictive mode.
- These cut the computation of convolutional operations short if it predicts final output of a computation to be negative during runtime. In this way we **prune** the ineffectual computations.



Exact mode

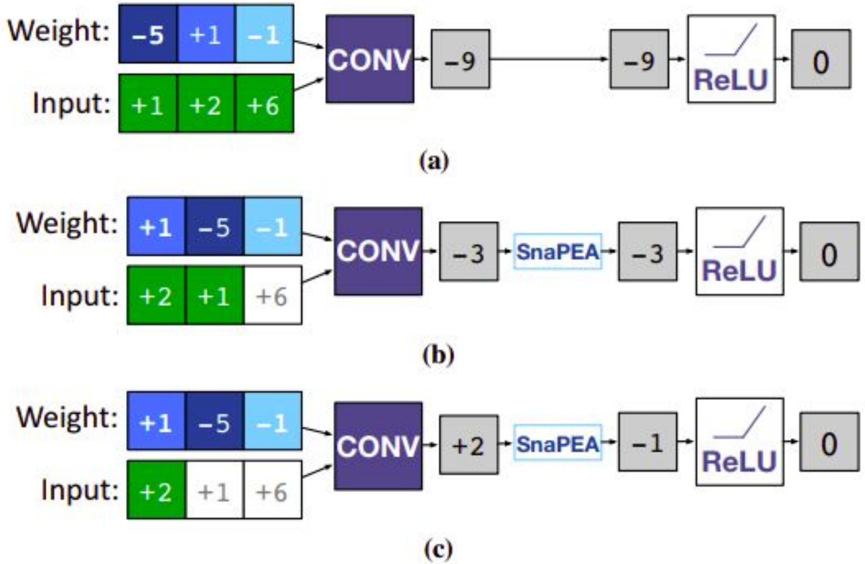
We rearrange weights in such a way to know if the computation overall results negative at the earliest.

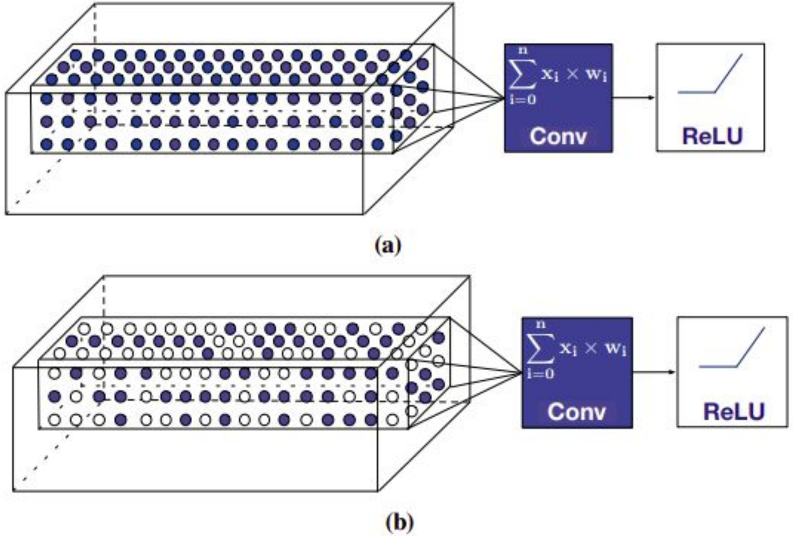
Algorithm

- ❏ Perform computation of add and multiply with positive weights
- ❏ Sort the negative weights in decreasing order of their absolute value
- ❏ In each cycle, perform computation using one negative weight in order.
  If at any stage, overall computation becomes negative then terminate.

(a) Original
(b) Exact
(c)Predictive

(a) Unaltered Convolution
(b) SnaPEA Convolution





Why move from Exact mode to Predictive mode?

- ❑ Selecting the weights with the larger magnitude ignores the contributions of inputs which are to a large degree, **random and data dependent**
- ❑ SnaPEA exact mode may not be effective for models where there are many negative weights with **similar magnitude.**

Predictive mode

Algorithm

- ❑ Sort the weights in ascending order, partition them into a number of smaller groups, and selects the weight with the largest magnitude from each group (called **predictive weights)**
- ❑ Then keep the remaining positive and negative weights in order.
- ❑ Computations are first done on predictive weights and then compared with a threshold value. If lesser than threshold, predict negative output and terminate.
- ❑ Else, terminate after computing for every weight value.

SnaPEA as an Optimisation Problem

$Th^k{}_l$ (threshold)

$N^k{}_l$ (number of groups)

Total number of weights of the kernel is ***$C_{in,l}$ × $D^k{}_l$ × $D^k{}_l$***

Then, the number of MAC operations required under different cases can be written as

$$\mathrm{Op}(o^d_{l,k}, \mathrm{Th}^k_l, \mathrm{N}^k_l) = \begin{cases} \mathrm{N}^k_l, & \text{if } \mathrm{PartialSum}_{\mathrm{N}^k_l} \leq \mathrm{Th}^k_l, \\ \mathrm{Idx}_{w^-}, & \text{if } \mathrm{PartialSum}_{\mathrm{N}^k_l} > \mathrm{Th}^k_l \, and \ \mathrm{PartialSum}_{w^-} \leq 0, \\ \mathrm{C}_{in,l} \times \mathrm{D}^k_l \times \mathrm{D}^k_l, & \text{otherwise} \end{cases}$$

SnaPEA as an Optimisation Problem

$$\min_{\mathrm{Th},\mathrm{N}} \sum_{d\in\mathscr{D}} \sum_{l\in L} \sum_{k\in K_l} \sum_{o\in O^d_{l,k}} \mathrm{Op}(o,\mathrm{Th}^k_l,\mathrm{N}^k_l)$$

$$\text{subject to} \quad \mathrm{Accuracy}_{CNN} - \mathrm{Accuracy}_{SnaPEA} \leq \varepsilon$$

ϵ : Acceptable accuracy loss
L : All layers in CNN
K : All kernels in layer l
D : Optimization dataset
$O^d_{l,k}$: Convolution output by kernel k in layer l for image d from D

Few Existing Limitations

General Limitations

- Details such as the number of memory buffers and the nature of the interconnect are crucial and also critical determinants of performance.
- When we reuse partial results across filters, this reduces the amount of computation at the cost of **increased storage space**.
- We need to balance the tradeoff between **reducing precision** (which leads to higher performance and reduced memory) and **accuracy loss.**

For CNNs, we can make these adjustments according to layers or type of layers.

Limitations in chosen problem [SnaPEA]

- ❏ SNAPEA model may not be effective for models where there are many negative weights with similar magnitude.
- ❏ Some **false positives** may occur. This will directly affect the accuracy, precision, recall and other metrics.
- ❏ Threshold acts a **control knob** for the accuracy (performance) and the computation involved. The stronger relationship for tradeoff between these needs to be established.

Proposed Solution and Work Done

Proposed Solutions

- ❏ Simplifying and formulating our own optimisation problem for the setup.
- ❏ Finding a better strategy (like **K-means clustering**) to find target weight from each group in weight list of SnaPEA predictive mode.
- ❏ Exploring possibilities of using **Dimensionality Reduction / Anomaly Detection** to identify weights.

Work Done

We have covered the following tasks uptil now.

❏ Conducted extensive literature Survey and reviewed various types of architectures and their implementations.

❏ Ideated on some select work and suggested possibilities.

❏ Learning Python Scientific Computing stack and **PyTorch** framework along with related libraries such as CUDA, CuDNN etc and trying out dummy implementations.

Further Work

Future plans

- ❑ Completing the implementation of the chosen existing work (SnaPEA)
- ❑ Implementing the proposed solutions and evaluating the results (**GPU based implementations primarily**.)
- ❑ Comparing the original results with our results on various hyperparameters and metrics. Figuring out limitations, analysing and overcoming them.
- ❑ An extension of this work could be to develop techniques for the training of the CNNs (If we are able to get good results on Inference in good time, we will start with training)

References

1. A None-Sparse Inference Accelerator that Distills and Reuses the Computation Redundancy in CNNs.
2. Prediction Based Execution on Deep Neural Networks
3. SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks.
4. Bit-Tactical: Exploiting Ineffectual Computations in Convolutional Neural Networks, Which, Why, and How.
5. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks.
6. SCNN: An accelerator for compressed-sparse convolutional neural networks
7. Re-architecting the on-chip memory sub-system of machine-learning accelerator for embedded devices.
8. Stripes: Bit-serial deep neural network computing

References

9.Dynamic Stripes: Exploiting the Dynamic Precision Requirements of Activation Values in Neural Networks.
10. Diffy: A Déjà vu-free differential deep neural network accelerator
11. Loom: exploiting weight and activation precisions to accelerate convolutional neural networks
12. UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision
13. [Stanford] Hardware Acceleration for Machine Learning
14. CNN Acceleration Survey

Github

Report

Thank You!

We are now open for questions and discussions.