Attention-based Activation Pruning to Reduce Data Movement in Real-time AI: A Case-study on Local Motion Planning in Autonomous Vehicles

Kruttidipta Samal, *Student Member, IEEE*, Marilyn Wolf, *Fellow, IEEE*, and Saibal Mukhopadhyay,*Fellow, IEEE*

***Abstract*—In state-of-the-art deep neural network (DNN), the layer-wise activation maps leads to significant data movement in hardware accelerators operating on real-time streaming inputs. We explore an architecture-aware algorithmic approach to reduce data movement and the resulting latency and power consumption. This paper presents an attention-based feedback for controlling input data, referred to as the activation pruning, that reduces activation maps in early layers of a DNN network which are critical for reducing data movement in real-time AI processing. The proposed approach is demonstrated for coupling RGB and Lidar images to perform real-time perception and local motion planning in autonomous systems. Lidar data is used to determine "Pixels of Interest"(PoI) in an RGB image depending on their distance from sensor, prune the RGB image to perform object detection only within the PoI, and use the detected objects to perform local motion planning. Experiments on sequences from KITTI dataset shows the activation pruning maintains quality of motion planning while increasing the sparsity of the activation maps. The sparsity-aware computing architectures is considered to leverage activation sparsity for improved performance. The simulation results show that proposed activation pruning algorithm reduces data movement (38.5%), computational load (30.1%), and memory latency (76.3%) in sparsity-aware compute architecture, leading to faster perception and lower energy consumption.**

***Index Terms*—robot vision, mobile autonomous system**

I. Introduction

Hardware accelerators for Artificial Intelligence (AI) models, in particular, deep neural network (DNN), are becoming pervasive in many real-time application [1] [2] [3]. An emerging application domain for AI hardware is the perception modules in various Autonomous Vehicles (AV), ranging from self-driving cars to mobile robots, that perform critical computer vision tasks (meta-tasks), such as object detection and tracking. The perception modules provide inputs for end tasks, such as motion planning [4], to generate control commands such as brake, accelerate, or turn.

The AI hardware in perception module need to process and integrate real-time data from multiple sensors including visible, infra-red (IR), and Lidar [5] [6]. Along with accuracy, the computational latency of the perception module is a key limiter of the reaction time and control precision in AV [7]. For example, object detection and tracking is a critical vision task.

Corresponding author: Kruttidipta Samal (email: ksamal3@gatech.edu).

Kruttidipta Samal and Saibal Mukhopadhyay are with School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA.

Marilyn Wolf is with Department of Computer Science and Engineering, University of Nebraska - Lincoln. Lincoln, NE 68588 USA

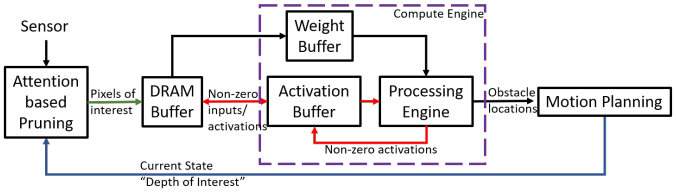
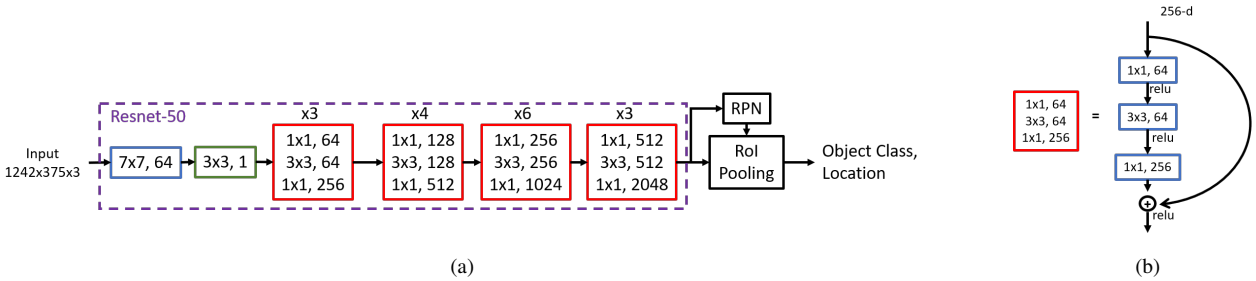


Fig. 1: Data movement in perception module of modern Autonomous System. Activation path(red) is the data path for input and network activations(focus of this work). Task based feedback(blue) is used to prune input and only "Pixels of interest"(PoI)(green) are used for processing. Weight and activation buffers are on-chip memory.

The DNN object detectors such as Faster-RCNN [8] with deep convolutional backbone (e.g. Resnet-50) requires 57 GFLOPs of computation; supporting a 25fps frame rate requires $\approx 1.71\ TFLOPs/S$ throughput. Moreover, the network includes 98 MB of weight and 751 MB of activation memory (see Section III-A for details). Operating on multiple modalities exacerbates the latency challenge [9]. For example, Voxelnet [10], a Lidar object detector, requires 735 GFLOPs, 76.3 MB of weight memory and 497 MB of activation memory. Hence, reducing computation, memory, and data movement of AI hardware, while ensuring accuracy, is critical for fast and efficient perception modules.

There have largely been two complementary approaches, namely, architectural and algorithmic, to reduce data movement in AI accelerators. Efficient weight, input, or output stationary data-flow models and architectures like systolic arrays have been developed to enhance data reuse and improve computational efficiency [11] [12] [13] [14] [15]. Near-memory computing techniques have been explored to reduce memory transfer between DRAM and processor [16] [17] [18] [19]. Likewise, compute-in-memory techniques based on SRAM, embedded DRAM, and embedded non-volatile-memory have been explored [18] [17] [20] [16]. The algorithmic approaches focus on decreasing number of weights in the model, often with graceful degradation of accuracy. For example, shallow object detectors such as SSD [21] and YOLO [22] have smaller models and hence, fewer computation, but are less accurate. The quantization, compression, and pruning of weights help reduce computation and data movement [23] [24] [25] [26] [27]. The hardware accelerators have been developed to leverage pruning for efficiency gains [28]. Efficient encoding techniques have been developed to exploit redundancy in activation maps and reduce data movement [29] [25] [30].

As mentioned before, activations in modern CNNs consume



(a) (b)

Fig. 2: Figure (a) shows the Architecture of the Faster RCNN Object Detection Network with Resnet-50 backbone. In this figure, Convolutional blocks have blue color, max pool layer have green color and Resnet block have red color. RPN and RoI pooling layers represent Region Proposal Network and Region of Interest pooling layers respectively. Each Resnet block is repeated several times, the number of repetition is mentioned on top of each block. The entire Resnet-50 backbone is represented in purple box. Figre (b) represents inner architecture of a Resnet block, in this figure we pick the first Resnet block and show its inner blocks and connections. Notice there is a skip connection which symbolizes residual connection in Resnet blocks.

much higher memory than weights. Consequently, pruning activations, rather than or along with weights, can provide more benefit in reducing data movement. Encoding techniques have been developed to exploit redundancy in activation maps [29] [25] [30]. As many activations are naturally "zero", recent accelerators can transform such sparsity to higher performance [31] [14] [28] [32]. We use the term ***"activation pruning"*** to indicate algorithmic approaches that can maximize the number of "zero" activations beyond the natural level. The activation pruning is more challenging than weight pruning as feature maps depend on time-varying inputs from multiple sensors and must be performed while ensuring adequate perception for the end task of the AV.

This paper explores a task-based knowledge to reduce data movement and computational load in AI hardware used in perception modules of AV. We present an ***"activation pruning"*** technique that intelligently couples RGB and Lidar images to explicitly use the real-time feedback from the end-task to perform attention-based pruning of the input images for motion planning applications (Figure 1). We couple the proposed activation pruning with recent advancements in sparsity-aware computing architectures resulting in improved performance and energy efficiency of the perception engine. We argue that in motion planning objects that are closer to the AV require more attention than the objects that are far away. The far-away objects have negligible impact on the trajectory planned in the current decision cycle. Hence, the attention-based feedback reduces data movement and computational workload by not perceiving the far-away objects.

At every decision cycle, we first compute a "Depth of Interest" based on the knowledge of the reaction time and current velocity of the AV (Current State in Figure 1). Next, we use real-time Lidar image of the scene to determine all the RGB pixels that are within this "Depth of Interest"(DoI) and these pixels are referred to as the "Pixels of Interest"(PoI)(green path in Figure 1). We have developed two methods- one uses connected component clustering of Lidar point cloud and the other uses 2D filtering of Lidar depth image to extract PoI from RGB image. The object detection is performed only within these PoI, instead of the entire RGB image. We show that the elimination of pixels outside of the PoI decreases the number of non-zero activation in CNN (inference) pipeline. Consequently, the proposed activation pruning reduces the number of non-zero activation data that must be (i) moved between compute and memory (less data movement) and (ii) operated on (less computation) during object detection. This results in reduced memory demand, less Floating Point Operations(FLOPs), and hence, improved energy/latency of the perception module.

Our experiments on sequences from KITTI dataset [33] show that proposed activation pruning using Lidar-assisted attention can eliminate 32.02% of activations per layer within RGB object detection CNN with virtually no impact on the quality of motion planning. We further observe a higher sparsity in activation maps of earlier layers of CNN, which can lead to 98% fewer DRAM access and 63% fewer floating point operations. A sparsity-aware compute engine leverages the reduced activation map resulting in 87% lower latency and 67.9% lower compute energy/frame.

The rest of the paper is organized as follows. Section II presents the background. Section III presents the proposed approach. Section IV presents the compute engine model. Section V presents the experimental platform. Section VI presents the results. Section VII presents discussion and future work. Finally, Section VIII concludes the paper.

II. Background

A. Object Detection Network

In this work, we focus on two-stage object detectors, namely, Faster RCNN [8] with Resnet-50 backbone (Figure 2 shows a detailed architecture diagram of this network), for perception. Two stage object detectors have very deep convolutional backbone (such as VGG-16, Resnet-50 or Resnet-101) for feature extraction. This is followed by a Region Proposal Network(RPN) that predicts regions in image that might be occupied by objects of interest, and a Classification & Regression Layer that process each region proposed by RPN to classify and predict a bounding box for the object under consideration. Data movement within the feature extraction layer is very high leading to performance degradation [34] of the entire network (more details in Section III-A). Therefore,

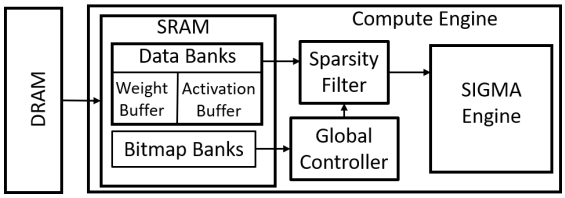
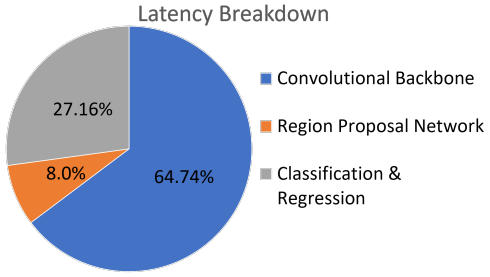


Fig. 3: High Level Architecture of Sigma Accelerator. Redrawn based on the architecture diagram from [14].

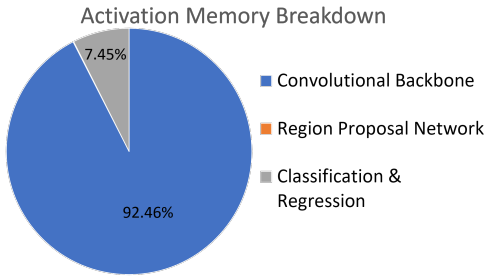
different techniques including weight pruning, quantization [25], short-cut connections [35], dense connections [36], and depth-wise separable convolution [37] have been proposed to reduce amount of computation and data movement within the backbone with graceful quality degradation. As an alternative, we propose to prune the input image based on task-level 'attention' to decrease network activation and thus reducing computational complexity(FLOPs) and data transferred, between memory and processor, within the backbone.

B. Hardware Acceleration of CNNs with Sparse Activation

The basic computational operation in CNN based Object Detection is convolution. Modern DNN accelerators and convolution kernels can eliminate multiplication operations if one of the inputs is zero [32] [38] [39]. In other words, advanced hardware accelerators for CNN can translate reduction in non-zero activation to improve performance/power consumption [14] [40], motivating different approaches to increase sparsity of activation maps of CNNs [41] [42]. Moreover, as layer activation maps have large memory footprint, they may not fit in on-chip SRAM, and have to be transferred to off-chip DRAM. The DRAMs can be 10x slower and 100x more power consuming than SRAM per memory access [43]. Hence, compression of activation maps have been explored [31]. While activation maps in deep layers in CNNs are naturally sparse, activation maps in early layers are dense and have much higher memory footprint. Most DNN software libraries convert convolution operation in a layer into 2D General Matrix Multiplication(GEMM) operation. As the same weights are multiplied with all input features, it leads to high memory re-use. Therefore modern accelerators such as Tensor Processing Units(TPU) [13] use systolic arrays and interconnects between Multiply-and-Accumulate(MAC) units which can leverage this memory re-use by loading either weight or activation matrix and streaming the other for efficient memory utilization. Qin et. al. [14] propose the Sigma accelerator that extends this systolic architecture to handle unstructured and irregular sparse GEMM operations. They introduce flexible interconnects and non-zero(NZ) data mapping to avoid transfer and computation of NZ data. Figure 3 shows the high level architecture of the Sigma accelerator. Here, activations and weights are stored in "Data Banks". Since only non-zero activations are used for computation, they compress the activations according to bitmap encoding. In this method, every element in activation matrix has a corresponding bit in bitmap to store whether its zero or non-zero. This leads to a constant sparsity compression



(a)



(b)

Fig. 4: Resource consumption breakdown. (a) shows latency attributed to different layers in a Faster RCNN object detection network with Resnet-50 backbone. (b) shows activation memory consumption attributed to different layers in a Faster RCNN object detection network with Resnet-50 backbone. Region Proposal Network consumes about 0.9% of total activation memory and hence is not visible in the pie-chart.

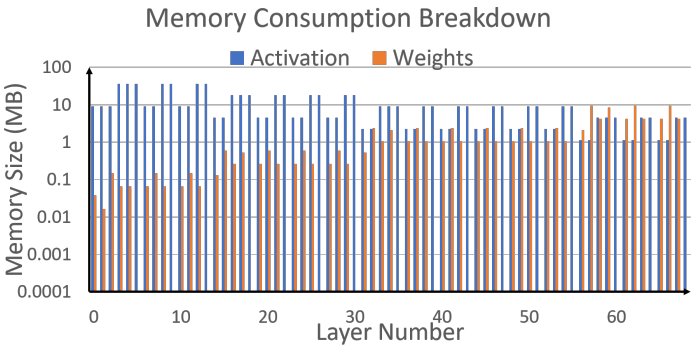


Fig. 5: Memory Consumption of Activation and Weights for different layers in Resnet-50 backbone in a Faster RCNN object detection network.

overhead. This bitmap is stored in "Bitmap Banks" in SRAM. Sigma accelerator [14] has "Global Controller" and "Sparsity Filter" that are responsible for accessing only non-zero data from memory according to bitmap encoding of the activation map. In this work, we show that attention-driven activation pruning can increase activation sparsity, and specially, in the early layers of CNNs which can lead to reduction in DRAM access and MACs in sparsity aware accelerator such as Sigma [14]

III. Proposed Approach

A. Motivation

In this section we explore our motivation for attention based "activation pruning". We begin by analyzing characteristics of

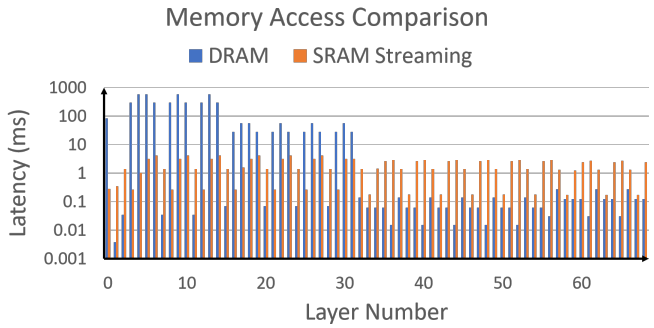


Fig. 6: Memory Latency attributed to DRAM and SRAM accesses for different layers in Resnet-50 backbone in a Faster RCNN object detection network. Here we are assuming DRAM access time=60ns, SRAM access time=10ns, SRAM Cache size=16MB. Memory access time values from "Network Systems Design Using Network Processors" [43]

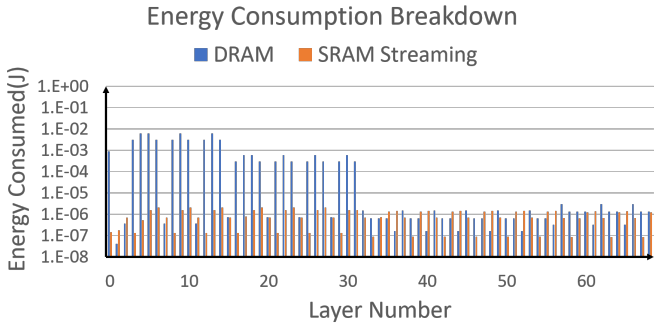


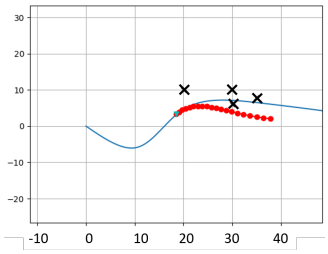
Fig. 7: Energy consumption attributed to DRAM and SRAM accesses for different layers in Resnet-50 backbone in a Faster RCNN object detection network.

different layers of a DNN based object detector. We executed a Faster RCNN [8] with Resnet-50 backbone from detectron2 [44] in a system with NVIDIA 1070 GPU, Intel i5 CPU and 16GB RAM. Figure 4a shows latency consumed and Figure 4b shows activation memory consumed by different layers of this object detection network. It can be observed that convolutional backbone consumes almost 65% of total latency and 92% of total activation memory.

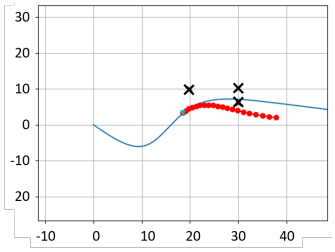
As the convolutional backbone consumes the most resources in terms of latency and memory(refer to Figure 4), we used the Resnet-50 backbone for subsequent analysis. We estimated data movement, latency, and energy of a simple systolic architecture(TPU-like systolic engine in Sigma Accelerator [14]) to process the Resnet-50 convolutional backbone of Faster RCNN object detector using an analytical model. The details of the model are given in Section IV. Figure 5 shows memory consumed by weights and network activations for this network. Here we can observe that activations have much higher memory footprint than weights and in early layers it can be more than cache size. In such cases activations have to be stored in DRAM. Thus increasing off-chip data movement. Figure 6 and Figure 7 show estimated DRAM and SRAM latency and energy consumption incurred at different layers of Resnet-50 backbone. Here we can observe that in earlier layers due to high activation memory, DRAM access is high, where as in later layers there is low DRAM access due to

TABLE I: Simulation Results.

Name	Baseline	Proposed
Avg. Obstacles per Frame	7.86	**3.70**
Average Velocity (Km/hr)	14.42	14.39
Time-Steps	93.85	94.28
No. of Failed Simulations	15	14



(a) Trajectory for $Baseline$



(b) Trajectory for $Proposed$

Fig. 8: Comparison of trajectory predicted by $Baseline$ and $Proposed$ methods. Here, blue line indicates Global Path, x indicate position of an obstacle, red dotted line indicates trajectory planned by the system at current time-step. Notice the obstacle located at far right in top image is not present in bottom image as it is beyond DoI.

less activation memory. Even though the number of SRAM access and SRAM latency is fairly constant throughout all layers, DRAM latency in earlier layers dominates the total latency incurred. Similarly, energy consumed in earlier layers is high due to high DRAM access. Therefore, if number of non-zero activations in earlier layers can be reduced, this will potentially lead to considerable reduction in DRAM access of the compute engine. Moreover, as discussed in Section II-B, less non-zero elements in the activation map will lead to less computations as well.

B. Basics of Motion Planning

Local motion planning is a major end task of perception module in AV. It takes inputs from odometry and mapping modules to create trajectory hypotheses that are maneuverable by the AV depending on its current inertia (velocity and acceleration) and other automotive properties (such as maximum acceleration, torque etc.). These trajectory hypotheses are evaluated according to their co-incidence to way-points from planned global path and probable collision risk with other objects in vicinity. The best hypothesis in terms of minimal deviation from global path and current inertia that is collision-free is selected for motion. This process is repeated at every time step to create a smooth and safe motion of

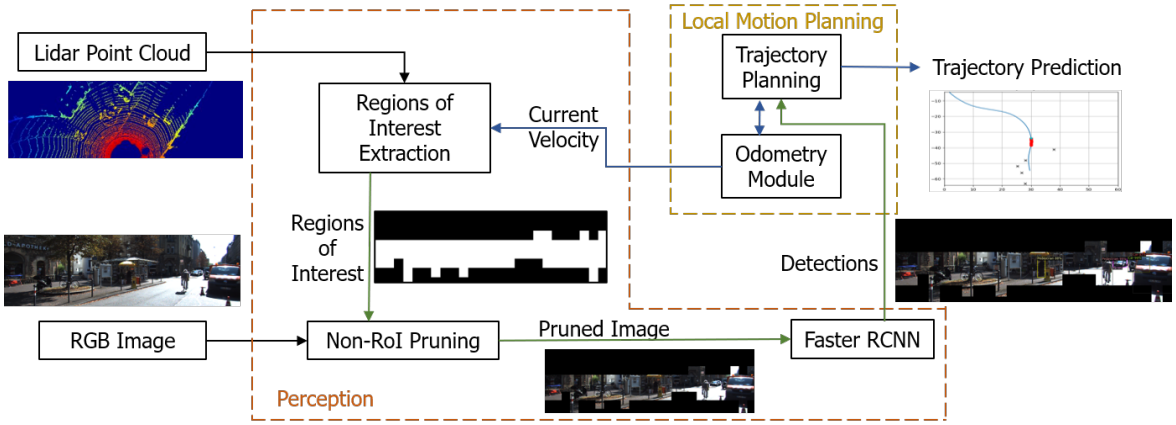


Fig. 9: Architecture of $Proposed$ system

an autonomous system. In order to evaluate collision risk of trajectory hypotheses, an obstacle map is created from objects detected by object detection module. This map contains locations of all detected objects.

C. Hypothesis

A key hypothesis of our approach is that obstacles beyond a certain distance can be neglected without any impact on trajectory planning. To test this hypothesis we experimented with Frenet Frame Trajectory Path planning [45] module in PythonRobotics [46]. By default this module is designed to plan motion of a robot with given way-points (global path), automotive properties (such as maximum velocity, acceleration and torque) and a static obstacle map (the $Baseline$ system). We augment this module (the $Proposed$ system) to ignore all obstacles that are beyond a "Depth of Interest"(DoI) at any given frame, where the DoI is computed as:

$$DoI(t) = Forecast\ Time \times Velocity(t) \tag{1}$$

Output paths created by the $Baseline$ and $Proposed$ system are compared considering the ability to create a viable trajectory (pass/fail), time to completion of the motion, average velocity and number of obstacles used for trajectory planning at every frame. This experiment was repeated 100 times with different obstacle maps.

Table I shows results of these simulations with $Forecast\ Time\ =\ 5\ Seconds$. The two systems show similar ability to find a trajectory. The $Proposed$ system considers less obstacles per frame to plan a path than $Baseline$ as it ignores all obstacles that are beyond DoI at any given time. There is negligible impact on time to completion and average velocity. Figure 8 shows trajectory of both $Baseline$ and $Proposed$ system at one time-step. $Proposed$ system's path in Figure 8b shows one of the obstacles that is located beyond DoI is not considered and yet there is no impact on planned trajectory.

D. Attention-driven Activation Pruning

As the quality of local motion planning is not impacted by ignoring far-away objects, we propose to remove regions in

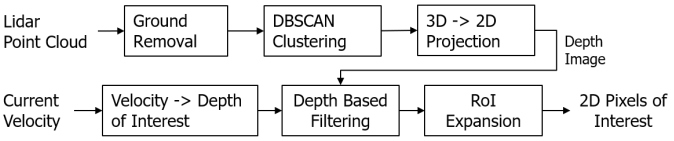


Fig. 10: Algorithmic Pipeline of $Proposed\_Cluster$ system.

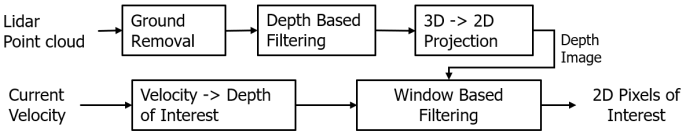


Fig. 11: Algorithmic Pipeline of $Proposed\_Window$ system.

RGB image that are located beyond DoI from the AV. The $Proposed$ system creates a feedback path between local motion planning module and object detection network (Figure 9). We use Lidar point cloud of a scene, a pre-set $Forecast\ Time$, and current velocity of the AV to create PoI within RGB image that represent the area within the DoI. All pixels that are outside PoI are turned to zero and the 'pruned' RGB image is used for object detection in a Faster RCNN network instead of original image.

E. Pixels of Interest Extraction

We develop two PoI extraction methods to prune an image.

1) Lidar Clustering: Architecture of this method is shown in Figure 10. It takes Lidar point cloud as input and uses Random Sample Consensus(RANSAC) to remove ground points. Ground point removal is a pre-processing step for Lidar data in autonomous systems, used for both SLAM(Simulataneous Localization and Mapping) and Object Detection [47]. Lidar point clouds have high point density at locations where objects are located. Therefore, we use euclidean connected component clustering like Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to detect high density point clusters. These clusters are considered as initial PoI proposals. While this method is prone to over- and under-clustering, our objective is not to create object proposals but PoI for RGB. Minimum distance between sensor and each cluster is obtained by calculating distance to all points within

a cluster and then sorting according to distance. Clusters which are located at distance less than DoI from sensor are projected on RGB image (using cross calibration matrix) to create PoI. Pixel values at regions that are outside PoI are turned to zero. Empirically we observed object detection networks need some background pixels for better detection. Therefore, the PoI from clusters are expanded by a preset factor to add contextual information for object detection.

2) Window Based Filtering: The DBSCAN is an iterative clustering algorithm. Also computational complexity of this algorithm is dependent on structure of Lidar point cloud of the scene under consideration, this can lead to non-deterministic computational load at run time. Even though most modern implementations of DBSCAN have max_iter parameter to limit the maximum number of iterations to reach optimal solution, it can lead to sub-optimal clustering. Due to these factors we develop a simple window based filtering method for PoI extraction. Architecture diagram of this method is shown in Figure 11. Similar to Lidar Clustering in Section III-E1, this method first ground points by RANSAC plane fitting. All points that are beyond DoI are removed from subsequent processing, this step is different from Section III-E1 as there is no point clustering prior to depth filtering. In the next stage, we project Lidar point cloud on RGB image plane using RGB-Lidar cross-calibration matrix. This projected image can be visualized as adding depth channel to existing three (3) color channels of RGB image, thus creating a four (4) channel RGBD image. As Lidar resolution is lower than RGB image, there are many RGB pixels that do not have any corresponding depth values. Therefore, we developed a "Window Based Filtering" operation that scans this 4D image in a strided manner and selects all windows that have one or more depth pixels within DoI as shown below:

$$out(x,y) = \begin{cases} in(x,y), \text{ iff } min(dm(w_x, w_y)) < DoI \\ \\ 0, \text{ else} \end{cases} \tag{2}$$

where, $(w_x, w_y) = \{(x+i, y+i)\}_{-W/2 \,<\, i \,<\, W/2}, \forall x, y$; dm is the depth map; DoI is the "Depth of Interest"; W is the Window Size, in is the Input Image; and out is the output image i.e. the pruned RGB image. This operation preserves context around objects helping improve object detection accuracy on the pruned RGB Image. We set the values of pixels in depth map that belong to ground (as detected by "ground removal" pre-processing) to be very high, thus the above filtering processing deletes ground points as well.

IV. Compute Engine Model

Estimating impact of the attention-based action pruning on latency and energy of an AI accelerator requires a model of the compute engine that can exploit sparsity of the activation map. As the paper focuses on the algorithmic technique, developing a cycle-level model of an architecture is beyond the scope. Therefore, in this section, we develop analytical models of a compute engine to estimate data movement, latency and energy savings. We consider two compute engine models in this work both use systolic array based processing, but one considers sparse-activation-aware processing engine and the other considers sparse-activation-unaware processing engine. The sparsity-unaware compute engine model is used for getting initial motivating results in Section III-A only, whereas sparsity-aware compute engine model is used for all subsequent analysis of both $Proposed$ and $Baseline$ systems. Since the convolutional backbone within object detection network is responsible for high latency and energy consumption (refer to Figure 4a and 4b), we used Resnet-50 convolutional backbone as computational load for this estimation. The Compute Engine in Figure 1(purple box) shows a high level architecture of this model. The size of Activation Buffer SRAM is 16MB and Weight Buffer SRAM is 1MB in our model. Floating point operations in all the computing models are single-precision 32 bit operations.

TABLE II: Value Table.

Name	Value
$SRAM^1_{acc}$	10ns
$DRAM^1_{acc}$	60ns
$SRAM^2_{epa}$	5pJ
$DRAM^2_{epa}$	640pJ
MAC^2_{epa}	4pJ
MAC^3_{acc}	2.8ns
PE	10000
i_x	1242
i_y	375
$stride$	10
ac	16MB

* In our estimation word size is 4 Bytes (size of a single precision floating point variable)
[1] Values from "Network Systems Design Using Network Processors" [43]
[2] Energy cost in a typical 45nm CMOS process [48] [39]
[3] Effective MAC unit latency in TPU [13]

A. Sparsity Unaware Compute Engine Model

We used the analytical model of TPU-like systolic engine presented in Sigma Accelerator [14] as sparsity-unaware compute engine. This model cannot leverage sparsity in activations for either data movement or computation.

In a typical systolic processing engine, weights are loaded once into the processing engine's registers and are reused in subsequent computations, whereas activations are loaded as per requirement in a 'streaming manner' [40] [14]. The time to load weights from SRAM buffer to processing element's register is significantly lower than the latency to stream activations into processing element [14]. Therefore, we used only streaming latency in subsequent analysis for calculating SRAM latency. SRAM memory transfer was estimated using the following equation from Sigma Accelerator [14]:

$$SRAM^l_{str} = M * ceil(K/\sqrt{PE}) * ceil(N/\sqrt{PE})$$

$Where,$

$$M = \frac{i^l_x . i^l_y . C^l_i}{f^l_x . f^l_y . C^l_f}$$
$$K = f^l_x . f^l_y . C^l_i$$
$$N = f^l_x . f^l_y . C^l_f \tag{3}$$

where, $SRAM^l_{str}$ represents the amount of activation data accessed from SRAM activation buffer. i^l_x and i^l_y represent the input spatial dimension and C^l_i represents the number of input channels. f^l_x, f^l_y, represent spatial dimensions of filters and C^l_f represents number of filters of layer l. PE is the total number of processing elements in processing engine (value shown in Table II).

Additionally, as DRAM memory transfer equations were not available in the analytical model, we developed following equations to estimate DRAM memory transfer:

$$DR^l = \begin{cases} W^l + i^l_x.i^l_y.C^l_i & \text{if } l = 0 \\ W^l + DW^{l-1} & \text{if } l > 0 \end{cases}$$

$$DW^l = o^l_x.o^l_y.C^l_f - ac$$

$$DIO^l = DW^l + DR^l \tag{4}$$

where, DW^l and DR^l represent the amount of data written to DRAM and read from DRAM respectively for layer l. W^l represents the size of Weights for layer l. Similarly, o^l_x, o^l_y represent the spatial dimension and C^l_f represent number of filters which is also same as the number of output channels of layer l. DIO^l is the total number of DRAM access in layer l and ac is the activation buffer cache size (value shown in Table II). Finally, we developed following equation to estimate SRAM and DRAM memory latency and energy consumption:

$$SRAM\_latency^l_{str} = SRAM^l_{str}.SRAM_{acc}$$

$$DRAM\_latency^l = DIO^l.DRAM_{acc}$$

$$SRAM\_energy^l_{str} = SRAM^l_{str}.SRAM_{epa}$$

$$DRAM\_energy^l = DIO^l.DRAM_{epa} \tag{5}$$

where, $SRAM\_latency^l_{str}$ is the latency incurred for SRAM streaming data transfer and $DRAM\_latency^l$ is the latency incurred for DRAM data transfer for layer l. $SRAM\_energy^l_{str}$ is the enrgy consumed by SRAM streaming data transfer and $DRAM\_energy^l$ is the energy consumed by DRAM data transfer for layer l. Values of SRAM access latency ($SRAM_{acc}$), DRAM access latency ($DRAM_{acc}$), SRAM access energy $SRAM_{epa}$ and DRAM access energy($DRAM_{epa}$) are shown in Table II.

B. Sparsity Aware Compute Engine Model

We used the analytical model of Sigma Engine [14] as sparsity aware compute engine. Figure 3 shows a high level architecture of this compute engine model. Here "Bitmap Banks" have a size of 2MB. This engine can leverage sparsity in activation maps for both data movement and computation. Equation 4 can be used to calculate DRAM and SRAM memory transfer for this set-up as well but DW and $SRAM^l_{str}$ are modified to take sparsity in activation maps into consideration [14]. Following is the equation for sparsity aware DW and $SRAM^l_{str}$ according to analytical model [14]:

$$\begin{aligned} DW^l &= s.o^l_x.o^l_y.C^l_f - ac \\ SRAM^l_{str} &= M.ceil(K.N.s^l/PE).ceil(N/\sqrt{PE}) \\ SRAM^l_{str} &= M.ceil(K.N.s^l/PE).ceil(N/\sqrt{PE}) \\ s^l &= \frac{NZ^l_{input}}{N^l_{input}} \end{aligned} \tag{6}$$

In the above equation, s^l represents sparsity of input, NZ^l_{input} is the amount of non-zero elements in input and N^l_{input} is the number of input elements for layer l. K, N, o^l_x, o^l_y, C^l_f, ac have the same definition as in Equation 4. In addition to data access, the analytical model also estimates SRAM access for sparsity overhead due to bitmap encoding of the activation maps, since bitmap metadata is stored only in SRAM, this overhead does not affect DRAM access. Following is the equation for sparsity overhead:

$$\begin{aligned} SRAM^l_{bm\_ip} &= i^l_x.i^l_y.C^l_i/32 \\ SRAM^l_{bm\_op} &= o^l_x.o^l_y.C^l_f/32 \\ SRAM^l_{bm} &= SRAM^l_{bm\_ip} + SRAM^l_{bm\_op} \\ SRAM\_latency^l_{bm} &= SRAM^l_{bm}.SRAM_{acc} \end{aligned} \tag{7}$$

In the above equation, $SRAM^l_{bm\_ip}$ is the amount of SRAM access required to load bitmap encoding of input activations and $SRAM^l_{bm\_op}$ is the amount of SRAM access required to store bitmap encoding of output activations for a layer l. i^l_x, i^l_y, C^l_i, o^l_x, o^l_y, C^l_f have the same definition as in Equation 4. $SRAM\_latency^l_{bm}$ is the total latency incurred for SRAM access of sparsity overhead metadata.

Following is the equation we developed for calculating total latency incurred and total energy consumed by DRAM, SRAM and Multiply And Accumulate (MAC) units.

$$\begin{aligned} SRAM_{latency} &= \sum_{l=0}^{L}(SRAM\_latency^l_{str} \\ &\qquad + SRAM\_latency^l_{bm}) \\ DRAM_{latency} &= \sum_{l=0}^{L} DRAM\_latency^l \\ MAC_{latency} &= \frac{MAC_{acc}}{PE}.\sum_{l=0}^{L}(s^l + \frac{1}{32}).f^l_x.f^l_y.C^l_i.(o^l_x.o^l_y.C^l_f) \\ SRAM_{energy} &= SRAM_{epa}.\sum_{l=0}^{L}(SRAM^l_{str} + SRAM^l_{bm}) \\ DRAM_{energy} &= DRAM_{epa}.\sum_{l=0}^{L} DIO^l \\ MAC_{energy} &= MAC_{epa}.\sum_{l=0}^{L}(s^l + \frac{1}{32}).f^l_x.f^l_y.C^l_i.(o^l_x.o^l_y.C^l_f) \end{aligned} \tag{8}$$

In the above equation, $SRAM_{latency}$, $DRAM_{latency}$ and $MAC_{latency}$ represent SRAM, DRAM and MAC unit latency respectively and $SRAM_{energy}$, $DRAM_{energy}$ and

MAC_{energy} represent energy consumed by SRAM, DRAM and MAC units while processing a single frame in a Resnet-50 convolutional backbone. Here, L is the total number of layers. $SRAM\_latency^l_{str}$ and s^l are defined in Equation 3. $DRAM\_latency^l$, DIO^l are defined in Equation 4. $SRAM\_latency^l_{bm}$ and $SRAM^l_{bm}$ are defined in Equation 7. f^l_x, f^l_y, C^l_i, o^l_x, o^l_y, C^l_f represent filter dimensions, number of input channels, output dimensions and number of filters for a layer l respectively. MAC_{acc} is the latency incurred and MAC_{epa} is the energy consumed for one word MAC operation (value shown in Table II). In the above equation there is a constant MAC overhead of $\frac{1}{32}$ for calculating sparsity bitmap for every layer. This is because bitmap size is always $\frac{1}{32}$ of input and output activations (each element in bitmap consumes one bit and corresponds to one element in activation map which consumes one word (4 bytes)).

C. Compute Engine model for Lidar overhead

We also developed a simple compute engine model to estimate latency and energy consumption for the RGB input image "pruning" operation using Lidar data for $Proposed_{Window}$ system. This operation loads the RGB and Lidar depth image into the processing engine and performs a filtering operation. As there is little data re-use in this operation, SRAM latency and energy consumption is assumed to be zero. DRAM and MAC latency and energy consumption is estimated using the following equation:

$$
\begin{aligned}
N^{ovd}_{mem} &= i_x.i_y.(4+3.s) \\
N^{ovd}_{mac} &= ceil(\frac{i_x}{stride}).i_y.(4+3*s) \\
latency^{ovd} &= N^{ovd}_{mem}.DRAM_{acc} + N^{ovd}_{mac}.\frac{MAC_{acc}}{PE} \\
energy^{ovd} &= N^{ovd}_{mem}.DRAM_{epa} + N^{ovd}_{mac}.MAC_{epa}
\end{aligned}
\tag{9}
$$

In the above equation, $latency^{ovd}$ and $energy^{ovd}$ represent latency and energy consumption overhead of the attention based input "pruning" operation. $stride$ represents stride of window filtering operation. s is the sparsity of the output image(out)(from Equation 2). Values of i_x, i_y, $stride$, MAC_{epa} and $DRAM_{epa}$ are shown in Table II. In this model we do not consider latency and energy consumed by RANSAC ground removal since it is a common pre-processing step utilized by other tasks in autonomous system such as SLAM.

V. Experimental Platform

A. Algorithm Evaluation Set-Up

We implement the $Proposed$ system on sequences from KITTI Tracking dataset. This required creation of obstacle map and global path. GPS co-ordinates(in Geographic co-ordinate system) of the vehicle at every frame was converted to North-East-Down(NED) co-ordinate system. As location of obstacles detected at any given frame are in 2D camera co-ordinate system, they are converted to 3D vehicle co-ordinate system using corresponding Lidar point cloud. Object locations in vehicle co-ordinate system are translated to NED co-ordinate system using rotation angle of the vehicle from East direction (yaw) at that frame.

Using the global path and per-frame obstacle map, trajectory planning was simulated using Frenet Frame Trajectory Path planning algorithm [45]. The object detector used for RGB object detection was Faster RCNN with Resnet-50 backbone from detectron2 [44], pretrained on Imagenet [49], COCO [50] and fine-tuned on KITTI object detection dataset [33]. The object detector and weights are common for both $Baseline$ and $Proposed$ systems, as our method does not require any retraining. $Baseline$ system uses entire RGB image whereas $Proposed$ systems uses pruned RGB image. Hereafter, the $Proposed$ system with Lidar Clustering PoI extraction (Section III-E1) is referred to as the $Propose\_Cluster$ and $Proposed$ system with Window Based Filtering(Section III-E2) is referred to as the $Proposed\_Window$.

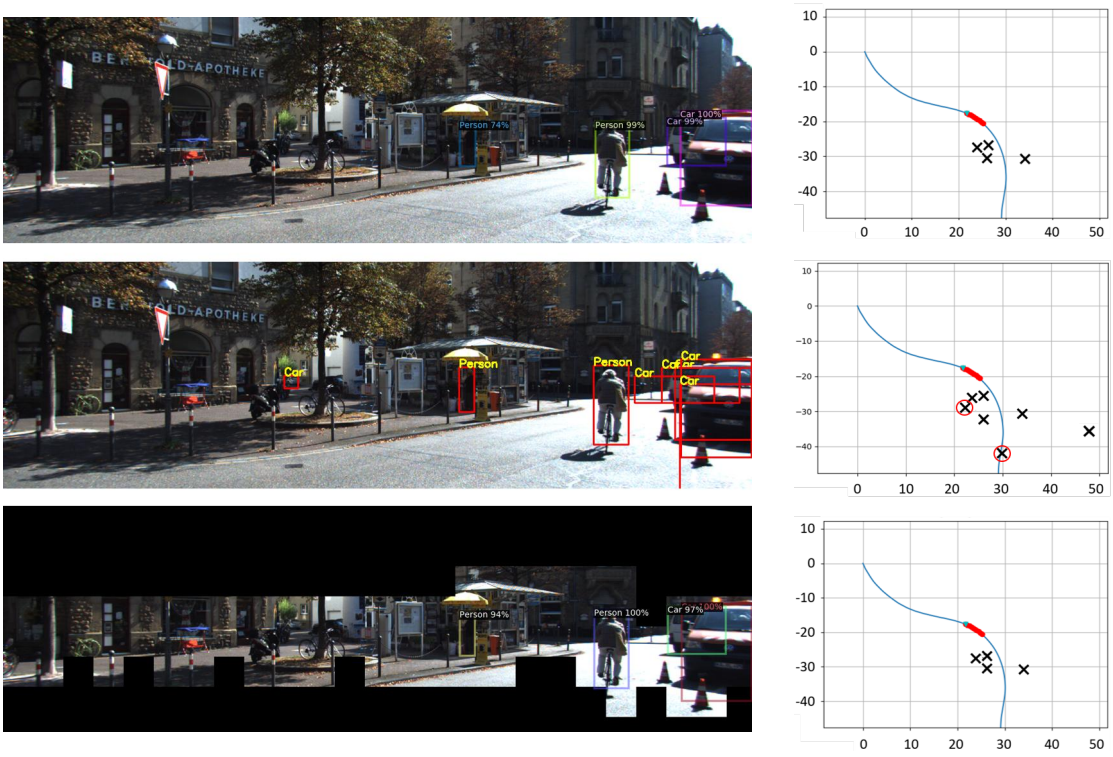
As a second baseline, we also create a $Baseline\_Fusion$ system that merges detections from a RGB object detector and a Lidar object detector to create the obstacle map. We use Voxelnet [10] which is a DNN based object detector that takes Lidar point cloud as input and gives 3D locations of detected objects as output. The Voxelnet used in the $Baseline\_Fusion$ system was trained on KITTI object detection dataset. The $Baseline\_Fusion$ represents a perception module that naively fuses multiple modalities. Also, the $Baseline\_Fusion$ is the only system that uses object detection in Lidar modality, neither $Baseline$ nor $Proposed$ systems detect objects in Lidar data.

Evaluation Criteria. Our objective is to decrease network activations of object detection network with least impact on trajectory planning. We use time to completion, average velocity per frame and number of obstacles considered for trajectory planning at every time step as parameters for trajectory evaluation.

B. Setup for Computation and Data Movement Analysis

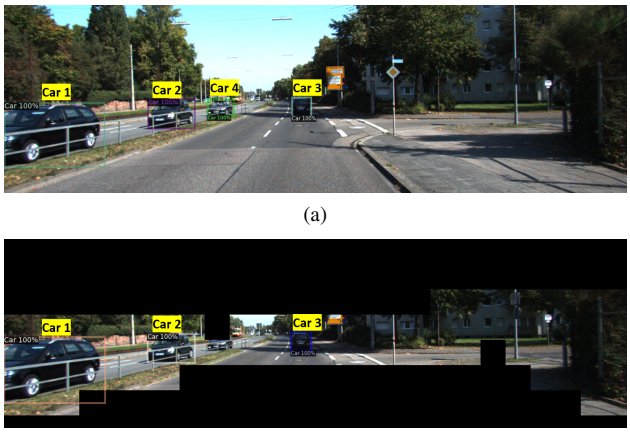
The data movement and computational efficiency is analyzed by coupling algorithm results (Section III) and computing engine models (Section IV). The first case, referred to as $Sparsity\ Unaware$, uses a sparsity un-aware compute engine (Section IV-A). The second case, referred to as the $Baseline$, uses sparsity-aware compute engine (Section IV-B) but does not consider activation pruning. Finally, we consider the $Proposed$ systems with activation pruning and sparsity-aware compute engine; but also includes the computing overheads associated with the Lidar processing (Section IV-C).

Evaluation Criteria. Data movement is estimated using number of non-zero activations inside convolutional backbone of the object detection network. The size of the non-zero activations represents both data movement and memory requirements. We also consider maximum non-zero feature map size of the network as indicator of worst case off-chip DRAM access. We estimate latency and energy consumption of different modules of compute engine for comparing data movement, latency, and energy dissipation of $Sparsity\ Unaware$, $Baseline$ and $Proposed$ systems.



(a) (b)

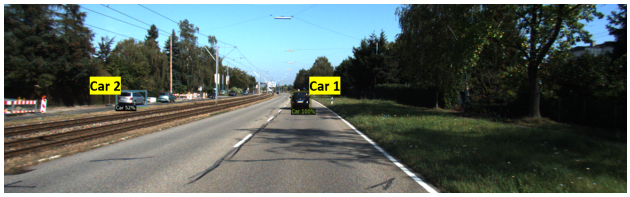
Fig. 12: Visualization of Object detection and obstacle maps. Figure (a) shows object detections by different systems and Figure (b) shows corresponding obstacle maps. (a, top) shows objects detected in $Baseline$ system, (a, middle) shows objects detected in $Baseline\_Fusion$ system and (a, bottom) shows objects detected in $Proposed$ system. (b, top) shows obstacle map corresponding to (a, top), (b, middle) shows obstacle map corresponding to (a, middle) and (b, bottom) shows obstacle map corresponding to (a, bottom). Objects detected by object detection are represented as 'X' in obstacle maps. False positives in object detection are circled 'red' in obstacle maps.



(a)

(b)

Fig. 13: Visualization of a scenario where input pruning fails. Figure (a) shows object detection on an unpruned input image and (b) shows object detection on the same image after pruning. Here, 'car 4' is not detected in pruned image.



(a)

(b)

Fig. 14: Visualization of a scenario where input pruning succeeds whereas not pruning fails. Figure (a) shows object detection on an unpruned input image and (b) shows object detection on the same image after pruning. Here, 'car 3' is detected in pruned image but not detected in unpruned image.

VI. Results

A. Visualization

1) Object Detection.: Object detection outputs of $Baseline$ and $Baseline\_Fusion$ systems are shown in Figure 12a (top) and 12a (middle) respectively. It can be observed that there are some false positives in the $Baseline\_Fusion$ system's output, this is due the lack of precision in Lidar object detection. Figure 12a (bottom) shows output of the $Proposed\_Window$ system. The input to the object detector of this system was pruned and therefore, all pixels that are outside PoI are inactive in this image. Such PoI pixels include both pixels that have depth greater than DoI and pixels that belong to the ground (refer to Section III-E for details).

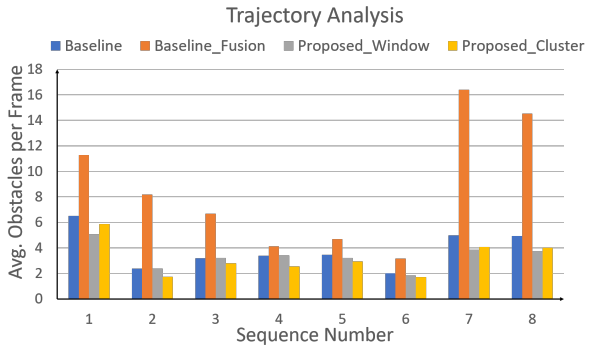


Fig. 15: Trajectory planning comparison between $Baseline$, $Baseline\_Fusion$, $Proposed\_Cluster$ and $Proposed\_Window$ systems. Columns indicate average number of obstacles encountered per frame for a given sequence.

TABLE III: Time Steps to Completion Table.

Sequence Number	Baseline	Baseline _Fusion	Proposed _Window	Proposed _Cluster
1	157	157	157	157
2	144	143	144	143
3	330	330	330	330
4	308	308	308	308
5	396	396	396	396
6	313	314	313	313
7	348	348	348	348
8	116	116	116	116

2) Obstacle Maps: Figure 12b shows the obstacle maps created by different systems at a particular frame in a sequence of KITTI dataset. From Figures 12b (top) and 12b (bottom) it can be observed that obstacle maps created by both $Baseline$ and $Proposed$ systems are same as all obstacles are within DoI. Whereas Figure 12b (middle) shows there are some additional obstacles detected by $Baseline\_Fusion$ system, these are false positives from Lidar object detector. We also encountered scenarios where $Proposed$ system was unable to detect objects within "Depth of Interest" due to aggressive pruning. Figure 13 shows one such scenario. This can happen if the Lidar points corresponding to an object are inaccurate, leading to inaccurate pruning. Contrarily we also observed that in some scenarios, $Proposed$ system was able to detect certain objects that were not detected in $Baseline$ system. Figure 14 shows such a scenario where pruning of input image allowed object detector to detect a car that was otherwise not detected. Such inconsistencies in detection are not prevalent and do not have an impact on end task as discussed in Section VI-B.

B. Trajectory Prediction

Figure 15 shows analysis of trajectory predicted by $Baseline$ and $Proposed$ systems on 8 sequences from KITTI object tracking dataset. Table III shows all systems take the same time to traverse the global path and Table IV shows all systems have similar velocity per frame, but they observe different number of obstacles in a given frame(Figure 15). As expected, both $Proposed$ systems encounter less number of obstacles than $Baseline$ system, as they ignore objects that are located beyond DoI. $Baseline\_Fusion$ system encounters highest number of obstacles per frame but most of these correspond to false detections. Similarity in time to completion and average velocity between $Baseline$ and $Proposed$ systems indicates that there is no noticeable change in autonomous system's motion due to $Proposed$ method. Simultaneously we did not observe any collision in any of the simulations. A video showing output of local motion planning with detection from the $Proposed\_Window$ system is available at [51].

TABLE IV: Average Velocity Table.

Sequence Number	Baseline (Km/hr)	Baseline _Fusion (Km/hr)	Proposed _Window (Km/hr)	Proposed _Cluster (Km/hr)
1	7.81	7.83	7.83	7.81
2	21.44	21.44	21.44	21.44
3	21.82	21.83	21.83	21.87
4	21.04	21.04	21.04	21.04
5	26.18	26.18	26.18	26.18
6	23.93	23.90	23.93	23.93
7	10.15	10.15	10.15	10.15
8	7.55	7.55	7.55	7.55

TABLE V: Data Demand Analysis.

Name	Total NZ Activations (MB)	Max NZ Activation Map (MB)
$Sparsity\ Unaware$*	732.4	35.7
$Baseline$	503.2	32.4
$Proposed\_Cluster$	371.4	23.1
$Proposed\_Window$	**280.3**	**16.8**

* All activations in $Sparsity\ Unaware$ are considered NZ Activations.

TABLE VI: Performance and Energy Analysis.

Name	Latency (ms)	Energy (mJ)
$Sparsity\ Unaware$	4900	365
$Baseline$	4160	254
$Proposed$*	**985**	**156**

*The $Proposed$ system refers to the $Proposed\_Window$ which has less activation than $Proposed\_Cluster$ (Table V).

C. Data Movement and Computational Efficiency Analysis

Table V shows the overall memory requirement of the activation maps for different cases. Table VI compares the total memory latency and energy consumption across all layers of the Resnet-50 backbone and Lidar overhead for different systems. These values include total latency from DRAM, SRAM and Multiply and Accumulate(MAC) units. In the $Sparsity\ Unaware$ and the $Baseline$ case, all computation is related to object detection; while for the $Proposed$ systems we account for the computation/memory overhead of the activation pruning algorithm. We first briefly compare the $Sparsity\ Unaware$ and $Baseline$ systems to study the effect of sparsity-aware compute architecture. The rest of the sub-sections perform detail comparison of $Baseline$ and $Proposed$ systems.

1) Effect of Sparsity-Aware Compute Architecture: Comparison between $Sparsity\ Unaware$ and $Baseline$ cases show that there exists appreciable natural sparsity in the activation map which, if exploited, can reduce data movement (Table V). This increase in activation memory and lack of ability to ignore Zero-input MAC operations leads to higher DRAM access and more MAC operations. Table VI shows the

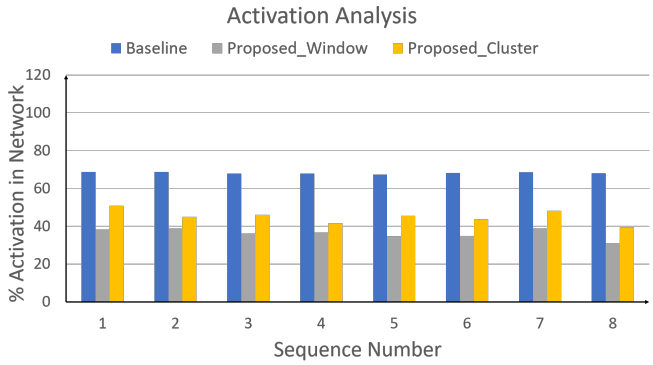


Fig. 16: Mean network activations in CNN convolutional backbone for 8 sequences in KITTI dataset are shown in columns.

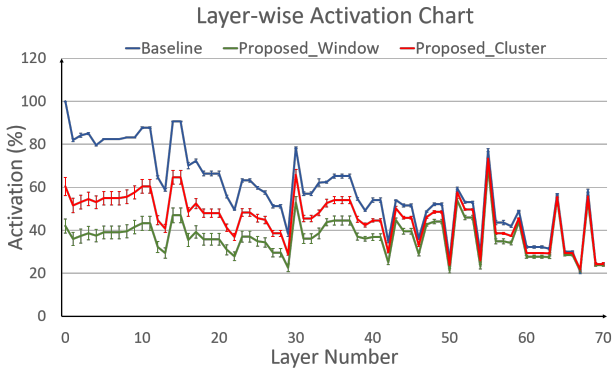


Fig. 17: Mean Layer-wise activation in convolutional backbone of *Baseline* and *Proposed* systems for 5 frames from the KITTI sequence. Error bars indicate standard deviation.

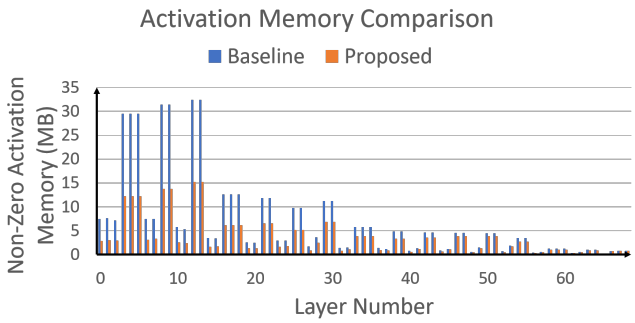


Fig. 18: Layer-wise Non-Zero Activation size comparison of *Baseline* and *Proposed* systems for a single frame.

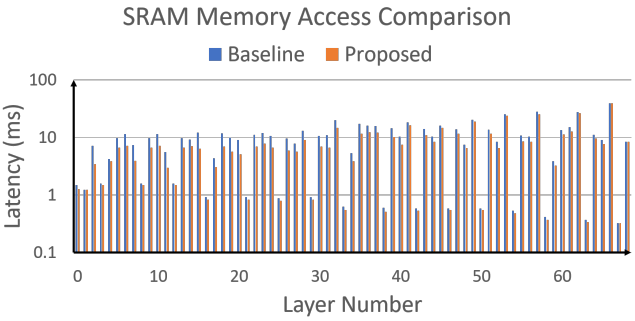


Fig. 19: Layer-wise SRAM Memory Access Latency comparison of *Baseline* and *Proposed* systems for a single frame.

overall latency and energy consumption of *Baseline* is lower than *Sparsity Unaware* by 28.3% and 30.4%, respectively.

2) Analysis of Data Demand: We compare data demand between the *Proposed* and *Baseline* systems. Figure 16 shows average non-zero activations within convolutional backbone(Resnet-50) of Faster RCNN object detector for each frame in 8 KITTI sequences. The *Proposed_Window* and *Proposed_Cluster* have 32.02% and 23.13% less non-zero network activations than *Baseline*, respectively. Figure 17 shows layer-wise non-zero activations in the Resnet-50 backbone within object detection network. In the first 15 layers of the network, the *Baseline* has 82.7% non-zero activation, while the same for the *Proposed_Cluster* and *Proposed_Window* are 55.3% and 39.4% respectively. Table V shows amount of non-zero memory consumed by activation maps in the *Baseline* and the *Proposed* system. The *Proposed_Window* system consumes 223 MB less non-zero memory than the *Baseline*. Also maximum non-zero memory consumed by an activation map in the *Proposed_Window* is 15.6 MB less than the *Baseline*. For subsequent analysis we refer to the *Proposed_Window* system as the *Proposed* system and compare it directly with *Baseline* system to showcase contrasting behavior. These results show that the *Proposed* systems have less non-zero activation leading to reduced memory footprint, off-chip DRAM access and overall data movement.

3) Savings in Memory Access Latency with Activation Pruning: In this section we perform a deeper analysis of memory access by both *Baseline* and *Proposed* systems within the convolutional backbone. Figure 18 shows the number of non-zero activation memory in every layer. In this chart, we can observe that earlier layers in *Baseline* system have the highest memory footprint. As sparsity of activations in *Proposed* system is high in earlier layers (refer to Figure 17), the savings in activation memory is also highest in these layers. Whereas in later layers, sparsity in activation map of both *Baseline* and *Proposed* systems is similar and their memory footprint is low. Therefore, the scope of memory savings is less in later layers, as shown in Figure 18. We observe that memory consumed by both *Baseline* and *Proposed* systems are almost same in later layers.

Figure 19 shows SRAM memory access latency for every layer of Resnet-50 backbone for a single frame. This includes SRAM latency due to both streaming and sparsity bitmap overhead. We observe that while SRAM latency of *Proposed* system is slightly less than *Baseline* for the earlier layers. This difference is due to higher sparsity in *Proposed* system's activation map compared to *Baseline*. Also, it can be observed that SRAM latency of later layers is higher than earlier layers, this is because filter size of later layers is higher than earlier layers.

Figure 20 shows layer-wise DRAM memory access latency for Resnet-50 backbone for a single frame. In some early layers we observe the DRAM latency for *Baseline* is 1000x

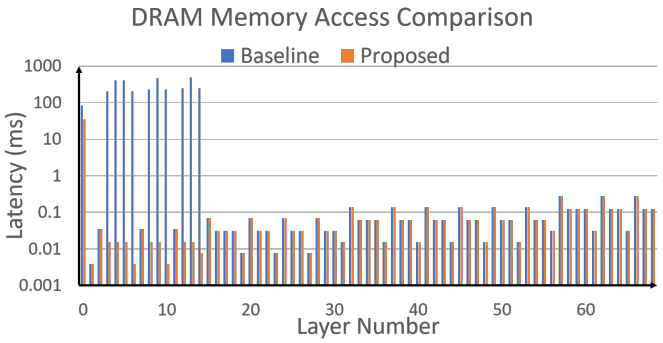


Fig. 20: Layer-wise DRAM Access Latency comparison of $Baseline$ and $Proposed$ systems for a single frame.

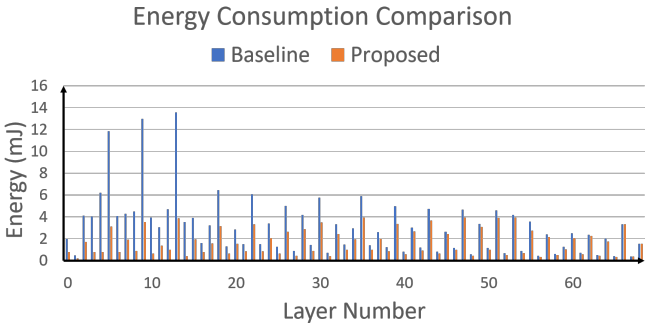


Fig. 21: Layer-wise Energy Consumption comparison of $Baseline$ and $Proposed$ systems for a single frame.

higher than $Proposed$. This is because $Proposed$ system has high activation sparsity in early layers, this leads to smaller non-zero activation map (refer Figure 18). Activation map that cannot fit within SRAM cache (Activation Buffer in Figure 1) is the primary source of off-chip DRAM access Therefore, smaller activation map leads to lower DRAM latency in $Proposed$ system.

4) Energy Consumption Savings with Activation Pruning: Figure 21 shows layer-wise energy consumption by SRAM, DRAM and Multiply and Accumulate(MAC) units for processing a single frame within Resnet-50 backbone. The energy consumed by $Baseline$ system is high in earlier layers. This can be attributed to low sparsity in activations leading to high DRAM access in these layers (refer to Figure 20). It can also be observed that $Proposed$ system has significantly lower energy consumption in earlier layers as attention based "activation pruning" increases sparsity in earlier layers leading to low DRAM access and therefore low energy consumption. In later layers, energy consumption is low for both $Baseline$ and $Proposed$ systems as DRAM access is low in these layers. Also, in later layers, there is no significant difference in the energy consumed by $Baseline$ and $Proposed$ systems, this is because both systems have similar memory access patterns in these layers (refer to Figure 19 and Figure 20).

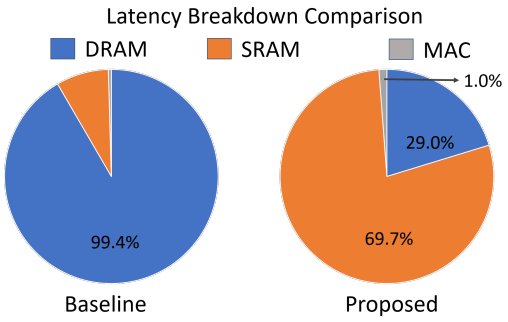


Fig. 22: Latency breakdown. $Baseline$ MAC latency $< 1\%$.

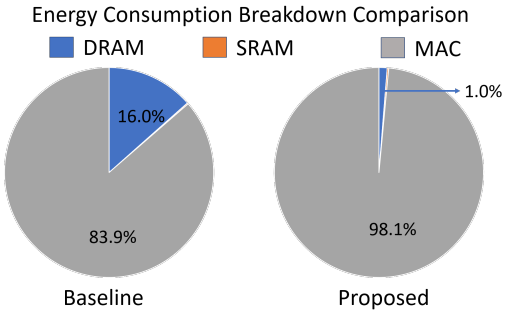


Fig. 23: Energy breakdown. SRAM energy $< 1\%$.

5) Savings in Other Parts: Region Proposal Network in Faster RCNN evaluates probability of object occurrence at every pixel of input image and only those pixels that have high probability of object occurrence are selected for subsequent processing. In both $Proposed$ systems, since all pixels outside PoI are turned to zero, computation of region proposals for these pixels can be safely avoided as well. Thus decrease in computational load will reflect in stages beyond convolutional backbone in object detection network.

6) Computational overhead of Lidar Processing: We calculated the latency and energy consumed by input "pruning" operation (refer to Equation 9). We observed an increase in latency by $147ms$. This can be attributed to increase in off-chip data movement. The window filtering operation (refer to Equation 2) requires loading of the entire RGB and Lidar depth image to the compute engine, thus increasing DRAM memory access which is a major cause of latency in the system (refer to Section VI-C3). Energy consumption overhead of the "pruning" operation is $1.57mJ$ which is nominal compared to energy consumed by the convolutional backbone (Figure 7).

D. Summary of latency and energy savings

The latency of the $Proposed$ system is 80.1% less than $Baseline$ system (Table VI). Figure 22 shows the latency breakdown of $Baseline$ and $Proposed$ systems. Latency of the $Baseline$ is dominated by DRAM. In the $Proposed$ system, DRAM latency contributes less while SRAM latency dominates. This is because "activation pruning" reduces off-chip data movement but does not significantly affect SRAM data movement(Figure 19). Similarly, we observe 38.5% reduction in the total energy consumed by the $Proposed$ system

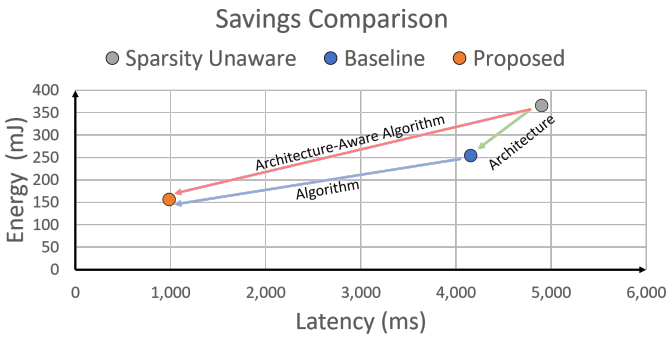


Fig. 24: Summary of latency and energy savings showing the benefits of algorithm and architectural techniques.

compared to $Baseline$ system (Table VI). The total energy in both systems is dominated by MAC energy (Figure 23). As Multiply and Accumulate(MAC) operations are not carried out for zero activations [14], the MAC energy for $Proposed$ systems is much lower than $Baseline$. In addition, DRAM energy contribution is negligible for the $Proposed$ system.

Figure 24 and Table VI summarize the overall benefit of the proposed task-based activation pruning in sparsity-aware architecture. The sparsity-aware compute architecture alone reduces latency (15.1%) and energy (30.4%) of $Baseline$ compared to $Sparsity\ Unaware$ architecture (green line in Figure 24). The $Proposed$ system couples sparsity-aware compute architecture with innovative attention-based activation pruning algorithm to decrease data movement resulting in further reduction in latency (76.3%) and energy (38.5%) than the $Baseline$ (blue line in Figure 24). In summary, the architecture-aware task-based activation pruning results in significant latency (79.8%) and energy (57.2%) savings in the $Proposed$ compared to the $Sparsity\ Unaware$ design (red line in Figure 24).

VII. Discussion And Future Work

A. Algorithm

The proposed algorithm prunes RGB input based on pixel distance which is sensed from a Lidar that is calibrated with the RGB camera. Currently only the pixels that have a distance less than depth of interest (DoI) and do not belong to ground are considered pixels of interest (PoI) and are used for subsequent processing by the object detection neural network. The DoI is calculated based on current velocity of the vehicle and a constant $Forecast\ Time$. It does not take the velocity of other objects in vicinity into consideration, this can lead to scenarios where the proposed algorithm might not detect an incoming fast moving vehicle if it is beyond the current depth of interest. Therefore, we have set the $Forecast\ Time$ at 5 seconds in our experiments which is higher than the average driver $Perception - Reaction\ Time$ (this is the time it takes for an average driver to perceive and react to a roadside hazard) of 1.64 secs [52]. Thus in highways where most high speed vehicles are encountered, a vehicle travelling at $Velocity = 60\ Km/hr$ with $Forecast\ Time = 5seconds$ leads to a $DoI = 80\ meters$, which is sufficiently higher than ideal stopping distance [53]. But ideally the $Forecast\ Time$ should be dynamic and depend on multiple factors such as vehicle brake dynamics, driving scenario, road conditions etc. Also the current logic can be extended to leverage Radar information which can sense both depth and relative velocity of different targets in the environment. We hope to explore these ideas in our future work.

We evaluated the proposed algorithm on sequences from the KITTI dataset which has been captured from ideal driving conditions in rural Germany. Therefore, the proposed algorithm could not be tested in challenging driving scenarios such as bad weather conditions or in scenarios with non-ideal target actors. Therefore, in future we plan to implement this algorithm in a real-world simulator such as Airsim [54] to prove its efficacy and safety.

B. Compute Engine

The proposed "activation pruning" reduces network activations within the object detection neural network. Using an analyical model of a sparse-aware architecture (Sigma Accelerator [14]) we observe that this reduction in network activations can lead to considerable reduction in latency and energy consumed by the network. The sparsity introduced due to the proposed "activation pruning" method is structured as regions in the input which are pruned tend to be spatially near each other. This can be observed in Figure 12a. Simultaneously the proposed algorithm leads to high sparsity in earlier layers of the network, which is different from existing works that depend on natural sparsity within the network (that exist in higher layers of the network) for improvement in performance. Therefore, in future we will explore architectures that can leverage the structured sparsity and high early layer activation sparsity features of the proposed algorithm. The latency and energy savings presented in Section VI are estimated based on an analytical model. But a cycle-level simulator would be better as it captures the various constraints of a micro-architecture. Therefore, we hope to synchronize a cycle-level simulator of a sparse-aware architecture executing the proposed algorithm with the real-world simulator mentioned in Section VII-A to create a holistic simulation system that captures both real-world dynamics and micro-architectural constraints of the hardware while executing the proposed algorithm.

VIII. Conclusion

We have presented an attention-based ***"activation pruning"*** algorithm that uses feedback from the local motion planning to intelligently prune the input, and hence, feature maps, to only focus on regions that are pertinent to the target end-task. The proposed activation pruning significantly reduces memory and data movement in the compute hardware. We further showed that a sparsity-aware compute architecture can translate this reduced data movement to significant energy and latency savings in a perception task. Therefore, we conclude that task-based activation pruning, executed on sparsity-aware architecture can provide substantial benefits in reducing data movement, latency, and energy dissipation in perception tasks in autonomous systems resulting in faster reaction time and lower energy consumption. The future work needs to consider

many different perception tasks in the AV and perform a detail micro-architectural design of the proposed system.

ACKNOWLEDGMENT

The research reported here was supported in part by the Defense Advanced Research Projects Agency (DARPA) under contract number HR0011-17-2-0045. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA.

REFERENCES

[1] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[3] L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and trends in signal processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[4] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Nov 2015.

[5] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 918–927.

[6] Z. Wang, W. Zhan, and M. Tomizuka, "Fusing bird's eye view lidar point cloud and front view camera image for 3d object detection," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, June 2018, pp. 1–6.

[7] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *ASPLOS*, 2018, pp. 751–766.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[9] H. Zhao *et al.*, "Towards safety-aware computing system design in autonomous vehicles," *arXiv preprint arXiv:1905.08453*, 2019.

[10] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4490–4499.

[11] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[12] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[14] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[15] NVIDIA, "Nvidia tesla v100 gpu architecture," 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[16] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 655–668.

[17] F. Schuiki, M. Schaffner, F. K. Gürkaynak, and L. Benini, "A scalable near-memory architecture for training deep neural networks on large in-memory datasets," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 484–497, 2018.

[18] P. Das, S. Lakhotia, P. Shetty, and H. K. Kapoor, "Towards near data processing of convolutional neural networks," in *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, 2018, pp. 380–385.

[19] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 380–392, 2016.

[20] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 420–434, 2018.

[21] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[23] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[24] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[25] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[26] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay, "Adaptive weight compression for memory-efficient neural networks," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 199–204.

[27] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[28] 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," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 27–40, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080254

[29] Y. 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, Jan 2017.

[30] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, 2020.

[31] G. Georgiadis, "Accelerating convolutional neural networks via activation map compression," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7085–7095.

[32] J. Albericio *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.

[33] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[34] H. Li, M. Bhargav, P. N. Whatmough, and H.-S. P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[36] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[37] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[38] J. Li *et al.*, "Squeezeflow: A sparse cnn accelerator exploiting concise convolution rules," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1663–1677, Nov 2019.

[39] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[40] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[41] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 806–814.

[42] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[43] D. Comer and L. Peterson, *Network Systems Design Using Network Processors*, 1st ed. USA: Prentice-Hall, Inc., 2003. [Online]. Available: https://npbook.cs.purdue.edu/agere/figs/figure-14.2.pdf

[44] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[45] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," 06 2010, pp. 987 – 993.

[46] A. Sakai *et al.*, "Pythonrobotics: a python code collection of robotics algorithms," *CoRR*, vol. abs/1808.10703, 2018.

[47] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 163–168.

[48] M. Horowitz, "Energy table for 45nm process, stanford vlsi wiki," accessed: 2020-08-11. [Online]. Available: https://sites.google.com/site/seecproject

[49] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

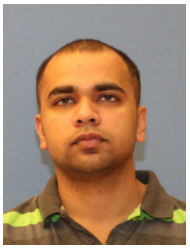
[50] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[51] "Demo video," accessed: 2020-08-11. [Online]. Available: https://drive.google.com/file/d/1QZeaCy9XdTi8VonhMofj0WxvHThCcWV/view?usp=sharing

[52] "Vehicle braking times - international clinical trials," accessed: 2020-08-11. [Online]. Available: https://internationalclinicaltrials.com/vehicle-braking-time.html

[53] "46.2-880. tables of speed and stopping distances," accessed: 2020-08-11. [Online]. Available: https://law.lis.virginia.gov/vacode/46.2-880/

[54] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065



Kruttidipta Samal received the B.Tech. degree in electronics and electrical enginnering from KIIT University, BBSR, India, in 2012, and the M.S. degree in electrical and computer engineering from Georgia Institute of Technology, Atlanta, GA, USA, in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Georgia Institute of Technology, Atlanta, GA, USA, under the supervision of Prof. S. Mukhopdhyay. Before starting his Ph.D., he was with the End-User Computing Team, VMWare, USA. His current research interests include multimodal computer vision, resource efficient perception systems.

Marilyn Wolf Marilyn Wolf is Elmer E. Koch Professor of Engineering and Chair of the Department of Computer Science and Engineering at the University of Nebraska Lincoln. She received her BS, MS, and PhD in electrical engineering from Stanford University in 1980, 1981, and 1984, respectively. She was with AT&T Bell Laboratories from 1984 to 1989. She was on the faculty of Princeton University from 1989 to 2007 and was Farmer Distinguished Chair at Georgia Tech from 2007 to 2019. Her research interests included embedded computing, embedded video and computer vision, and VLSI systems. She has received the IEEE Computer Society Goode Memorial Award, the ASEE Terman Award and IEEE Circuits and Systems Society Education Award. She is a Fellow of the IEEE and ACM and an IEEE Computer Society Golden Core member.

Saibal Mukhopadhyay received the B.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, in 2000, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2006. He is currently the Joseph M. Pettit Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. He has authored or coauthored more than 200 articles in refereed journals and conferences. He holds five U.S. patents. His current research interests include the design of energy-efficient, intelligent, and secure systems. His research explores a cross-cutting approach to design spanning algorithm, architecture, circuits, and emerging technologies. He was a recipient of the Office of Naval Research Young Investigator Award in 2012, the National Science Foundation CAREER Award in 2011, the IBM Faculty Partnership Award in 2009 and 2010, the SRC Inventor Recognition Award in 2008, the SRC Technical Excellence Award in 2005, the IBM Ph.D. Fellowship Award from 2004 to 2005, the IEEE Transactions on VLSI Best Paper Award in 2014, the IEEE TRANSACTIONS ON COMPONENT, PACKAGING, and MANUFACTURING TECHNOLOGY Best Paper Award in 2014, and multiple best paper awards in International Symposium on Low-power Electronics and Design in 2014, 2015, and 2016.