International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-10 Issue-6, August 2021

Memory Optimization Techniques in Neural Networks: A Review

Pratheeksha P, Pranav B M, Azra Nasreen

Abstract: ***Deep neural networks have been continuously evolving towards larger and more complex models to solve challenging problems in the field of AI. The primary bottleneck that restricts new network architectures is memory consumption. Running or training DNNs heavily relies on the hardware (CPUs, GPUs, or FPGA) which are either inadequate in terms of memory or hard-to-extend. This would further make it difficult to scale. In this paper, we review some of the latest memory footprint reduction techniques which would enable faster low model complexity. Additionally, it improves accuracy by increasing the batch size and developing wider and deeper neural networks with the same set of hardware resources. The paper emphasizes on memory optimization methods specific to CNN and RNN training.***

Keywords: Memory footprint reduction, Backpropagation through time (BPTT), CNN, RNN.

I. INTRODUCTION

Data being "the new oil of digital economy" is only accelerating in terms of volume, velocity, and variety. This valuable asset is exploited by modern technologies like machine learning, deep learning, data mining and big data analytics. Despite the fact that these techniques have become standard tools for handling any complicated problem in the domains of video analytics, image processing, speech recognition, and natural language processing, they still suffer from memory bottlenecks. Since many deep network models are resource-intensive, researchers struggle with the limited memory bandwidth of the devices used.

The most resource or compute-intensive phase of deep learning is the training phase. GPUs are commonly used to train these heavy machine learning or deep learning workloads for it offers several benefits over its non-specialized hardware counterparts. They are optimized for parallelizing training tasks, simultaneous compute operations (10-15x faster than CPUs), and sparing CPU for other jobs. However, the main motivation behind employing GPUs is due to its high memory bandwidth which also proves to be insufficient for larger networks. Table 1 shows memory and performance capacities of GPUs commonly used by scientists, researchers and enthusiasts for deep learning tasks.

Table- I: Memory and Performance capacities of commonly used GPUs

GPU Model	Memory (GB)	Cache (MB)	Performance (TFLOPS)
NVIDIA GeForce RTX 2080 Ti	11	6	120
NVIDIA Titan V (Standard Edition)	12	4.5	110
NVIDIA Titan RTX	24	6	130
NVIDIA GeForce RTX 2080 Ti	11	6	120

Table 2 shows GPU memory consumption (in GB) of state-of-the-art models when trained using some of the standard DL frameworks like Tensorflow, Pytorch and MXNet for an input size of 224x224x3 and batch size of 128.

Table- II: GPU memory consumption (in GB) of state-of-the-art models [1].

GPU Model	Memory (GB)	Cache (MB)	Performance (TFLOPS)
NVIDIA GeForce RTX 2080 Ti	11	6	120
NVIDIA Titan V (Standard Edition)	12	4.5	110
NVIDIA Titan RTX	24	6	130
NVIDIA GeForce RTX 2080 Ti	11	6	120

As depicted in Table 2, some of the networks like LSTM, ResNet and other simpler models can be trained on any of the GPUs mentioned above but networks like Inception v3 and VGG-16 hit memory bottlenecks for just a batch-size of 128. With increase in batch size and also the trend moving towards deeper and wider networks to improve training accuracy, the existing hardware proves to be restrictive.

Manuscript received on July 19, 2021.
Revised Manuscript received on July 25, 2021.
Manuscript published on August 30, 2021.
* Correspondence Author
Pratheeksha P, Department of Computer Science, R. V College of Engineering, Bengaluru (Karnataka), India. Email: pratheekshap.cs17@rvce.edu.in
Pranav B M*, Department of Computer Science, R. V College of Engineering, Bengaluru (Karnataka), India. Email: pranavbm.cs17@rvce.edu.in
Dr. Azra Nasreen, Assistant Professor, Department of Computer Science, R. V College of Engineering, Bengaluru (Karnataka), India. Email: azranasreen@rvce.edu.in

Over the previous five years, tensor processing in top-of-the-line GPUs has increased by 32 times, yet total accessible memory has only increased by 2.5 times. Hence, optimization techniques to reduce memory footprint in DNNs is indispensable. Dealing with memory bottleneck issues requires either changing the network architecture or scaling the training to multiple nodes. In this paper, a comprehensive review on the latest software based approaches developed for memory reduction in neural networks is carried out.

II. RELATED WORK

Memory consumption in DNNs can be categorized as memory footprint for DNN training and DNN inference. Modern neural network algorithms are difficult to deploy on limited hardware resources (such as mobile devices, IoT devices and other embedded systems) owing to their large memory requirements. Major sources of storage overhead in DNN inference are weights. Several prior compression techniques have resulted in a significant reduction in storage requirements and also to efficiently operate on the compressed networks with no accuracy loss [2,3].

As per [4], memory usage while training is classified as (i) model memory : to store weights and biases of each layer, (ii) optimizer memory : to store gradients and momentum buffers if any, and (iii) activation memory to store the activations of each layer in both forward and backward pass. Activation memory contributes to the majority of the memory usage while optimizer memory is 2-3x the model memory. Since memory for the training constitutes about twice as much as inference, approaches to optimize memory requirements for training are crucial to further improve model performance.

Prior works [4-8] effectively reduce memory footprint for training and minimizing computational overhead and it ensures efficient resource utilization [9]. [10] shows how heuristic algorithms for memory allocation reduces memory consumption and speeds up the training process. Recent approaches are progressing with the aim of accelerating DNNs to be more scalable and in a highly parallel architecture [11]. Although these methods prove to be efficient, they are generic. LSTM RNNs have tanh or sigmoid as their activation function and with their high runtime overhead for small layers cannot make use of optimization methods that are applicable to CNN training , thus necessitating neural network (CNNs, RNNs or GANs) specific optimization techniques.

In the following section, some of the latest memory footprint reduction techniques for CNN and RNN training are explored.

III. MEMORY FOOTPRINT REDUCTION TECHNIQUES

Efficient management of memory and reduction in memory requirements are the two main techniques in reducing the memory footprints. In this section, such state-of-the-art techniques employed in CNNs and RNNs are studied and their performance enhancements are noted.

A. *Convolutional Neural Networks (CNNs)*

CNN and its variants are known to be the most efficient approaches in deep learning models. Primarily, CNNs are used for computer vision and Image processing tasks and have achieved many state-of-the-art results. CNN often consists of multiple types of layers, which are basically categorized as convolutional (CONV) layers, subsampling or so-called pooling (POOL) layers, fully-connected (FC) layers, and activation (ACTV) layers.

1. Memory-Scheduling Strategy through virtualization

Shilje Li et al.[12] have proposed a novel memory scheduling strategy named mixed memory CNN (mmCNN). In this model, memory is virtualized as CNNs perform computations. In general, the programmers spend most of the time optimizing the memory while coding. But this model takes care of optimization and helps the developers to concentrate on network architecture. The essence of this model is in the concept of virtualization of memory. Virtualization is done by transferring the memory between host and device which might appear like a costly operation. But the model is sophisticated enough to decide the set of operations so that it does not affect performance.

The mmCNN framework is constructed using three main components.

(i) *Preprocessing module* - This module is used to fetch hardware and network configuration of the platform.

(ii) *Control module* - This module executes the proposed algorithm and hence is considered as the core module of the model.

(iii) *Feed-forward module* - This module is responsible for handling the data obtained from the control module.

In this algorithm, the initial starting point of all convolutional segments are found out. In the next step, the potential peak memory usage is calculated and it is denoted by Memmax. After calculating the Memmax, the available memory and maximum memory usage are updated. The Convolutional Part Number and Self Part Number indicates the convolutional layers to be prefetched and the number of parts the current layer's feature map should be divided into respectively. These numbers are calculated in the next step. When the current convolutional finishes the computation, the memory reserved by the layer is calculated and released. It is then added to available memory for the next iteration.

The experimental analysis shows that this method saves 98% memory as compared to traditional CNN. It also indicates that this methods saves more than 90% of memory compared to state-of-the-art related to Virtual DNNs (vDNNs) [13]

2. Designing Hardware Accelerators for CNNs

The approach used in [10] profiles on-chip memory size and off-chip memory bandwidth according to requirements of CNNs. This profiling helps in understanding the effect of memory system on accelerator design.

The memory requirements and memory bandwidth for various networks, and for various layers within a network, can differ by several orders of magnitude. Because of this fact, designing of fast and efficient hardware for all CNN applications becomes difficult. The system [10] thus proposes four heuristic design points that tries to optimize different data flow scenarios.

The CNN accelerator uses both on-chip memory and off-chip memory bandwidth. The on-chip memory is helpful in reducing the expensive off-chip memory accesses. The key constraint that has been dealt with is the determination of the amount of on-chip memory sufficient to guarantee that each activation or weight accessed off-chip utmost once per layer. Four memory schemes provided in the paper are as follows:

(i) Everything is present on-chip

(ii) Working set of activations and all filters with activations stored off-chip

(iii) Working set of filters and all activations with activations stored off-chip

(iv) Working set of both filters and activations with both stores off-chip

Here the working set refers to the number of filters/ activations that are computed in parallel and thus need to be stored on-chip.

There is a large variation across networks in terms of memory requirements, memory size, bandwidth, and between activation memory and weight memory. To minimize off-chip traffic for weight-intensive and activation-intensive networks, schemes 2 and 3 respectively offer a good heuristic design. Minimizing on-chip memory is desirable in some use cases and scheme 4 represents such a design point.

3. CPU offloading to reduce the memory cost of training CNNs

Training large CNNs demand huge amount of resources such as specialized Graphical Processing Units (GPU) and highly optimized implementations to get the most efficient performance out of hardware. During CNN training procedure, size of both inputs and model architecture are limited because of GPU memory which is a major bottleneck. The GPU memory contains the input activations of every layer and they are used in the backward pass to compute the layer weight gradients. As a result, during forward and backward computations across higher layers, the activations of lower layers are kept idle in GPU memory.

System proposed in [14] alleviates this memory bottleneck by leveraging an under-utilized resource of modern systems: the device to host bandwidth. Their method is termed CPU offloading, works by transferring hidden activations to the CPU upon computation, in order to free GPU memory for upstream layer computations during the forward pass. These activations are then transferred back to the GPU as needed by the gradient computations of the backward pass. However, this method faces a key challenge of efficiently overlapping data transfers and computations in order to minimize wall time overheads induced by the additional data transfers. The results show that there was a 35% decrease in memory requirements which added a wall time overhead of 21%.

While optimizations on computation have been extensively studied, off-chip memory accesses continue to restrict the energy efficiency of such accelerators because their energy cost is orders of magnitude higher than other operations. Minimizing off-chip memory access volume is thus the secret to improving energy efficiency even further. Xiaowei Li et al. [15] addresses the technique to overcome the problem in the prior state-of-the-art which used rigid data reuse patterns and is suboptimal for some, or even all, of the individual convolutional layers. They proposed an adaptive layer partitioning and scheduling scheme, called SmartShuttle, to minimize off-chip memory accesses for CNN accelerators. Smartshuttle dynamically matches different convolutional layers and fully connected layers by switching between different data reuse schemes and the corresponding tiling factor settings. Furthermore, SmartShuttle explores the effect of data reusability and sparsity on memory access volume in depth. SmartShuttle processes the convolutional layers at 434.8 multiply and accumulations (MACs)/DRAM access for VGG16 (batch size = 3) and 526.3 MACs/DRAM access for AlexNet (batch size = 4), outperforming the state-of-the-art method (Eyeriss) by 52.2 percent and 52.6 percent, respectively, according to the experimental results [15].

4. Heterogeneous Memory Management System for Optimization

Split Convolutional Neural Network (Split-CNN) [16] is a new type of CNN that is generated from the automatic transformation of current CNN models. Split-CNN differs from standard CNN in that it divides the input images into small patches and operates on these patches independently before proceeding to the next stage of the CNN model. They also present a novel heterogeneous memory management system (HMMS) that takes advantage of Split CNN's memory-friendly qualities. HMMS's five-step method of planning memory usage for computation graphs is as follows:

(i) The training model is split during the first stage. HMMS automatically converts a conventional convolutional neural network (CNN) to a Split-CNN with a splitting depth d determined by the percentage of convolutional layers to break apart and a 2-tuple of integers (h,w) specifying the number of splits in each spatial dimension (height and width).

(ii) Computations are serialized by topologically sorting compute nodes in the dataflow graph.

(iii) Later, each tensor in the computation graph is assigned with a tensor storage object.

(iv) Optimal offloading and prefetching scheme is derived to offload the most amount of memory without hurting the performance.

(v) In the final stage, three memory pools are created to accommodate the memory storage space requirements of the computation graph and its corresponding offloading or prefetching plan.

B. Recurrent Neural Networks (RNNs)

Recurrent neural networks are an important class of neural networks known for processing sequential data with its application ranging from sequence learning, neural machine translation, speech-to-text conversion, image-to-text translation to captioning videos, language translations, and many more. A typical RNN consists of stacks of input, hidden, and output layers. It works on the principle of saving the output of a particular layer and feeding it back to the input in order to predict the output of the layer.

With the introduction of LSTM, the vanishing gradient problem caused by RNNs was resolved to a certain extent but the need for an efficient network to process the ever-growing data in real-time was evident.

LSTMs, RNNs and their variants consume huge amounts of resources for training as well as deploying making it almost impossible to scale.

1. Compiler-based GPU Memory Footprint Reduction

Compiler-based automated optimization technique to reduce memory footprint in LSTM RNN based deep learning tasks [7] provides a comprehensive memory and runtime profile analysis of state-of-the-art NMT models. The model addresses two key challenges which were overlooked by prior works. These challenges are accurately estimating footprint reduction by adopting a selective recomputation strategy and non-conservatively estimating runtime overhead by deducing the data dependencies of the gradient operators [7]. Echo integrated with NNVM - a computation graph compiler (MXNet framework) is adjusted to collect relevant information (shape and datatype) prior to Echo pass. This allows for accurate estimation of footprint reduction.

The workflow begins with gradient node insertion and inferring shape and datatype of each tensor edge. In EdgeUseRef pass, the compiler traverses through the whole computation graph to compute the number of tensor references made by different operators. This is essential to account for the global effect of storage allocations in contrast to local which is a mere difference between memory released and allocated. Echo identifies targets for recomputation (backward pass), partitions the computation graph into disjoint subgraphs with compute-heavy layers as boundaries and eliminates those recomputation which do not contribute to the footprint reduction (forward pass). It also considers compute-heavy operators for recomputation if the runtime overhead is minimal. The backward-forward pass loop is repeated until all the inputs of the whole graph are covered. DeadNodeElimination as the name suggests, eliminates those nodes whose outputs are not referenced by any other node. The result of this method is that only those feature maps are saved in the memory whose recomputations prove to be costly. Scope of Echo is not just limited to RNNs but proves to be efficient across diverse machine learning models such as CNNs. Other factors that stand out for Echo is minimal/no programming/manual effort as it does not alter the training algorithms.

Echo when evaluated against four state-of-the-art models namely NMT, DeepSpeech2, Transformer and ResNet shows an incredible footprint reduction ratio of 3.13x, 1.59x, 1.56x, and 2.13x respectively. This reduction can be used to boost the training performance by either increasing the batch size or deepening the neural network.

2. Memory-Efficient Recurrent Neural Networks for language processing

A novel method to effectively reduce memory footprint and computational complexity in RNN models specific to language processing tasks is presented in [17]. Language modelling often involves a large vocabulary which limits the GPU memory capacity and as the size increases, compute-intensive operations like multiplication of output-embedding matrix with hidden state of a sequence can be quite laborious making it difficult to train and deploy. This issue is addressed by using 2-Component shared embedding where each word is characterized by a row and column vector. The model LightRNN [17] consists of 3 components : Bootstrap framework for word allocation, training the RNN model with 2C shared embedding, and MCMF algorithm for minimum weight perfect matching to refine the word allocation.

The model is evaluated on ACLW and BillionW datasets against state-of-the-art HSM and C-HSM algorithms with perplexity as a performance metric. With similar perplexity, LightRNN reduces the model size by a factor of 40-100 and improves the training speed by a factor of 2. The reduction in size allows for training the model with larger dimensions of embedding vectors which leads to better training accuracy.

3. Variants of Backpropagation through time algorithm

The following method [18] highlights multiple enhancements on the well-known backpropagation through time (BPTT) algorithm with a preset user-defined memory budget, allowing RNN training to fit on nearly any existing hardware resources. Three optimizations proposed are: (i) backpropagation through time with selective hidden state memorization (BPTT-HSM), (ii) backpropagation through time with selective internal state memorization (BPTT-ISM) and (iii) backpropagation through time with mixed state memorization (BPTT-MSM). These three strategies leverage memory reuse and dynamic programming to identify the optimal memory consumption policy with maximum computational performance.

In BPPT-HSM, initially, the hidden state (result of forward propagation) of any arbitrarily chosen RNN core (partitioning the sequence into two halves) is saved into one memory slot. Similar forward operation is then performed on the second half with the remaining memory slots. After computing the gradients (backpropagation) for the second half, the memory slots can be released and reused while backpropagating in the first half. It makes use of the divide-and-conquer approach to partition time steps and recursively solves each part. However, memory reuse and savings comes at the cost of increased recomputations. The cost of choosing the first hidden state to be saved is solved using dynamic programming and optimal state is chosen. BPPT-ISM follows the same approach as BPPT-HSM, but saves internal state instead of the hidden state. Thus, reducing one recomputation (forward operation during backpropagation) in every divide-and-conquer step. But BPPT-ISM requires more memory since the internal state takes up more space than the hidden state. BPPT-MSM is a combination of BPPT-HSM and BPPT-ISM where either internal or hidden state is saved depending on the optimal position and computational cost (equations solved using dynamic programming). For long sequences, BPPT-HSM performs better than BPPT-ISM, while BPPT-MSM significantly outperforms the other two approaches. When evaluated against standard BPTT, this method saves upto 95% of memory for a sequence length of 1000 but with 33% more training time per iteration.

Retrieval Number: 100.1/ijeat.F29910810621
DOI:10.35940/ijeat.F2991.0810621
Journal Website: www.ijeat.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication

IV. CONCLUSION

Various methods to reduce memory requirement to effectively manage memory in CNNs and RNNs are explored in this paper. It outlines the strategies that outperforms earlier state-of-the-art models. The popular approaches to memory optimization includes running operations such as activation functions 'in-place,' allowing the input data to be overwritten directly by the output.

It was found that memory can be reused by analysing data dependencies between network operations and allocating the same memory to operations that do not use it simultaneously. Memory in neural networks has been shown to be reduced by a factor of 2 to 3 when these methods are combined as seen in Echo and mmCNN. It is observed that these models improve memory requirements substantially as compared to others. Buffering and paging, model compression, memory sharing, and memory swapping are some of the techniques that can provide performance benefits when GPUs are used. Memory reduction frequently comes at the expense of recomputation, more training time, accuracy loss, or all of these. To identify the suitable technique, the trade-off between these factors must be carefully examined.

REFERENCES

1. Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, Mao Yang, "Estimating GPU Memory Consumption of Deep Learning Models", 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1342–1352, Nov 2020.
2. Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, William J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network", IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), vol. 44, no. 3, pp. 243-254, June 2016
3. Song Han, Huizi Mao, William J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding", arXiv:1510.00149 [cs.CV], Feb 2016.
4. Nimit S. Sohoni, Christopher R. Aberger, Megan Leszczynski, Jian Zhang, Christopher R´e, "Low-Memory Neural Network Training: A Technical Report", arXiv:1904.10631 [cs.LG], Apr 2019.
5. Aashaka Shah, Chao-Yuan Wu, Jayashree Mohan, Vijay Chidambaram, Philipp Krähenbühl, "Memory Optimization for Deep Networks", arXiv:2010.14501 [cs.LG], Oct 2020.
6. Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang and Gennady Pekhimenko, "Gist: Efficient Data Encoding for Deep Neural Network Training", IEEE 45th Annual International Symposium on Computer Architecture (ISCA), July 2018.
7. Bojian Zheng, Abhishek Tiwari, Nandita Vijaykumar, Gennady Pekhimenko, "Echo: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training", IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 1089–1102, May 2020.
8. Donglin Yang, Dazhao Cheng, "Efficient GPU Memory Management for Nonlinear DNNs", 29th International Symposium on High-Performance Parallel and Distributed Computing, pp. 185–196, June 2020M. Young, *The Techincal Writers Handbook.* Mill Valley, CA: University Science, 1989.
9. Minsoo Rhu; Natalia Gimelshein; Jason Clemons; Arslan Zulfiqar; Stephen W. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design", 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1-13, Oct 2016.
10. Sekiyama, T., Imamichi, T., Imai, H., & Raymond, R. (2018), "Profile-guided memory optimization for deep neural networks", arXiv preprint arXiv:1804.10001.
11. M. Imani, M. Samragh Razlighi, Y. Kim, S. Gupta, F. Koushanfar and T. Rosing, "Deep Learning Acceleration with Neuron-to-Memory Transformation," 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 1-14
12. S. Li, Y. Dou, J. Xu, Q. Wang and X. Niu, "mmCNN: A Novel Method for Large Convolutional Neural Network on Memory-Limited Devices," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2018, pp. 881-886
13. K. Siu, D. M. Stuart, M. Mahmoud and A. Moshovos, "Memory Requirements for Convolutional Neural Network Hardware Accelerators," IEEE International Symposium on Workload Characterization (IISWC), 2018, pp. 111-121
14. Hascoet, T., Zhuang, W.H., Febvre, Q., Ariki, Y. and Takiguchi, T. "Reducing the Memory Cost of Training Convolutional Neural Networks by CPU Offloading". Journal of Software Engineering and Applications, vol 12, pp. 307-320, 2019
15. J. Li et al., "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, pp. 343-348
16. Jin, Tian & Hong, Seokin, "Split-CNN: Splitting Window-based Operations in Convolutional Neural Networks for Memory System Optimization", Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 835-847
17. Xiang Li, Tao Qin, Jian Yang, Tie-Yan Liu, "LightRNN: memory and computation-efficient recurrent neural networks", 30th International Conference on Neural Information Processing Systems, pp. 4392–4400, Dec 2016.
18. Audrūnas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, Alex Graves, "Memory-Efficient Backpropagation Through Time", 30th International Conference on Neural Information Processing Systems, pp. 4132–4140, Dec 2016.
19. Wei R, Li C, Chen C, Sun G, He M. "Memory Access Optimization of a Neural Network Accelerator Based on Memory Controller". Electronics, vol. 10, no. 4, pp. 438, Feb 2021
20. Kim, H., Lyuh, C.-G. and Kwon, Y., "Automated optimization for memory-efficient high-performance deep neural network accelerators", ETRI Journal, vol 42, pp. 505-517, July 2020
21. S. Rajbhandari, J. Rasley, O. Ruwase and Y. He, "ZeRO: Memory optimizations Toward Training Trillion Parameter Models," SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1-16
22. Zhang, Junzhe, Sai Ho Yeung, Yao Shu, Bingsheng He, and Wei Wang. "Efficient memory management for gpu-based deep learning systems." arXiv preprint arXiv:1903.06631 (2019).

AUTHORS PROFILE

Pratheeksha P, is a data science enthusiast with a strong desire to solve complex system architecture challenges in critical applications with a real-world impact. She is currently pursuing her BE in Computer Science from RV College of Engineering. Bangalore. Her technical interests include Deep Learning, Cloud Computing and Big Data Analytics and has worked on multiple projects. Email: pratheekshap.cs17@rvce.edu.in

Pranav B M, is currently pursuing his BE in computer science from RV College of Engineering, Bangalore. He has worked in many start-ups to understand the real-world challenges and learn creative approaches in solving them. His technical interests include Deep learning, IoT, Cyber security and Networking. Email: pranavbm.cs17@rvce.edu.in

Dr. Azra Nasreen, has over 15 years of teaching and research experience. Presently, she is an Assistant professor at RV College of Engineering, Bangalore. She has guided many UG and PG projects and also published her works in various international journals and conferences. Her research interests include Deep learning, Video Analytics and High-Performance Computing. Email: azranasreen@rvce.edu.in