Gradient-Based Compression and Approximate Computing for Deep Network Optimization in Multimedia Data Processing

Vishal Krishna Singh, Jagpreet Singh, Rajkumar Singh Rathore, Devansh Nema

***Abstract*—Existing methods of compressed multimedia data processing in deep networks are constrained by inherent trade-offs such as low reconstruction accuracy, compression ratio, and high computational latency. With computational intensive tasks, the performance further deteriorates owing to high energy requirements and processing delays in real-time, large-scale multimedia applications. This work presents a deep learning framework that integrates gradient-based compression with approximate computing to optimize the in-network processing of multimedia data. Hyperparameter tuning is employed to systematically adjust bit-width, model size, and network depth, enabling fine-grained control over compression and computational efficiency. The proposed approach makes use of adaptive convolutional layers and dynamic learning rates for local gradient residue compression with the aim to improve the exploitation of low-rank structures and data sparsity. Extensive simulations are performed on publicly available datasets, including CIFAR-10, CIFAR-100, and MNIST, to validated the performance of the proposed method. Results demonstrate the effectiveness of the proposed method as it outperforms a set of state-of-the-art approaches achieving a classification accuracy of 99.09%, with almost real-time processing of the multimedia data.**

***Index Terms*—Approximate Computing, Data Compression, Deep Learning, Image Processing, Convolution Neural Network, MNIST.**

I. Introduction

THE era of Machine Learning (ML) and Computational Intelligence (CI) will be defined by the pivotal role of Deep Learning (DL) algorithms in several domains such as data transmission, in-network data processing, sensor networks, Internet of Things (IoT), big data analytics, pattern recognition, and many more. Specifically, the large volume of data and need for high accuracy and precision by the Neural Networks (NN) and Deep Neural Networks (DNNs) in resource-constrained networks impose additional overhead on the storage requirements, computational capacity, transmission bandwidth, and processing delays. These issues have been considered by several state-of-the-art methods, such as in [1], [2], [3]; however, the cost of in-network processing in deep models has escalated exponentially with the continually increasing volume of the data.

Interestingly, Approximate Computing (AC) presents an alternate computational paradigm for addressing the trade-off introduced by the large volumes of data in deep networks. Discarding the exactness of the traditional computing paradigm, AC achieves a significant edge in computational throughput without any major loss in accuracy or precision. The work presented in [4] shows the wide applications of AC to a variety of domains, where it has been phenomenal in reducing computational costs. A comprehensive analysis, presented in [4] and [5], proves that with AC not only the training cost of the DNN could be easily reduced to more than 50%, but also the overall execution time can be optimized up to 50% of the actual run time. The traction being received by AC for in-network data processing, especially in DL methods, comes naturally after the ground-breaking work of [6], [7] which was based on the Shannon-Nyquist sampling theorem. Since then, Compressed Sensing (CS) along with its various versions, have been the center of in-network data processing in resource hungry networks.

The literature supports that in-network processing has witnessed many approaches based on CS that make use of different transforms such as Wavelet, Discrete-Fourier and Discrete-Cosine [6], [7], [8]. Considering the wide range of algorithms developed for image data processing, they may be roughly categorized into three categories: *greedy algorithms, optimization-based algorithms*, and *learning-based algorithms* [9], [10]. However, naive application of CS in resource constrained networks, imposes extra overhead on the intermediate nodes and results in energy holes in the network [11]. Alternatively, various hybrid-CS methods have been used to optimize the load distribution and network lifetime in such resource constrained networks [12], [13]. It is empirical to note that in multimedia applications, large volume of data is simultaneously reported from multiple sources (example: video cameras), with each camera recording live at a sampling frequency of a few minutes, resulting in thousands of samples simultaneously being reported at any given time frame. Further, with the advent of $3D$ technologies, a steep proliferation in the data volume has been reported in such applications [14]. In-network processing, as such, becomes an imperative need when dealing with multimedia data being reported in real-time or almost real-time applications. Several methods have been proposed to achieve storage and in-network processing optimizations in recent years, such as dictionary-based approaches

Vishal Krishna Singh is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester, U.K. (E-mail: v.k.singh@essex.ac.uk

Jagpreet Singh is with the Indian Institute of Technology Ropar, Punjab, India. (E-mail: jagpreets@iitrpr.ac.in).

Rajkumar Singh Rathore is Department of Computer Science, School of Technologies, Cardiff Metropolitan University, United Kingdom. (E-mail: rsrathore@cardiffmet.ac.uk)

D Nema is with Intel India, Bengaluru, India. (E-mail: devanshnema14@gmail.com).

[15], [16], [17], the predictive coding methods [18], [19], [20], [21], [22], and component compression methods [23], [24], [25], [26].

A careful observation of these methods presents a series of interesting approaches to the in-network processing and analysis of stream image data. An example of such an approach is the work proposed in [27], where a CS-based method is used for the utilization of block cipher structures for encoding quantized information. These structures incorporate operations such as scrambling, blending, S-box, and chaotic matrix XOR. The seminal work by the authors in [28] introduced a novel hybrid approach to compression and encryption using CS, where a chaos index controls the measurement matrix constructed from the Hadamard matrix. The technique aims to ensure both rapid and secure transfer of photos over the internet. Another work, proposed in [29], suggested incorporating optical compression for simultaneous multiplexing and encoding of images, addressing the challenges associated with internet-based transmission. Furthermore, the authors in [30] proposed an innovative strategy that combines a hyper-chaotic network with $2D$ CS-based image compression and encryption, enabling concurrent execution of compression and encryption tasks while achieving data reduction and improved security. Our previous work on CS based in-network processing [31] and [13] stands out for efficient data processing and load balancing results. Alternatively, methods proposed in [32], [33] and [34] have used graph based methods and game theory [35] for in-network performance optimization.

These studies provide valuable insights and serve as relevant contributions to the literature in the field of DL and ML. However, it is pertinent to note that most of the mentioned DL and ML based approaches are highly dependent upon the need of highly interconnected GPUs for effective communication. During each iteration of the distributed processing, vast amounts of weight data must be synchronized across accelerators, which leads to substantial energy consumption and communication time. Understandably, with the high scaling distribution of data batches, due to the proliferation in the number of participating nodes, the communication bandwidth requirement increases exponentially. Considering the significant need for high data transfer rates, this work aims to address the issue of large number of hyper-parameters impacting the bandwidth consumption and in-network processing in deep networks. Optimizing the bit-width, model size and network depth through hyper-parameter tuning, the proposed work utilizes a DL framework to enhance model convergence and reduce computational overhead by implementing a gradient based compression method. The optimization of the local gradient residue allows maximum utilization of the low rank feature and sparsity in the data, leading to an improved accuracy, processing time, and energy consumption. This work exploits the advantages of the amalgamation of two approaches, i.e., AC and DL, with the aim of generating compounded benefits that are used to optimize storage and computation costs in deep networks. With the optimized in-network processing, the proposed method (hereafter termed as *Approximate Deep Learning (ADL)*), uses AC and Convolutional Neural Networks (CNN), for accurate feature extraction from the input data. The proposed *ADL*, achieves a significant edge, specifically in terms of accuracy, energy consumption, compression efficiency and end-to-end delay, as compared to state-of-the-art algorithms. The proposed *ADL* is able to optimize the substantial computational burden placed on the Fully Connected (FC) layers of DL networks. The improved model convergence and reduced computational overhead are validated on real-world image data sets. To define the scope of this study (without limiting the application of the proposed *ADL* to a specific domain), this work specifically considers the case of three widely used and publicly available *CIFAR-10*, *CIFAR-100 [36]* and *MNIST [37]* data sets, for testing and validation. The proposed *ADL* is also implemented on a real world test-bed and the results with varying batch sizes and learning rate (LR) are presented. The unique contributions of this work are summarized as below:

1) This work presents the novel *ADL* (AC based optimized DL framework), which amalgamates the major advantages of AC and CNNs for accurate feature extraction.
2) The proposed *ADL* utilizes a DL framework applied over a gradient-based compression method to optimize model convergence and computational overhead. The optimization of the local gradient residue allows maximum utilization of the low rank feature and sparsity in the data, leading to an improved in-network processing.
3) The proposed *ADL* is able to outperform most of the existing state-of-the-art methods, and the same is validated through a series of tests on publicly available *CIFAR-10*, *CIFAR-100 [36]* and *MNIST [37]* data sets.
4) The proposed *ADL* is also implemented on a real world test-bed and the results with varying batch sizes and LR are presented. Performance validation and comparative analysis prove that the proposed *ADL* has a significant edge, specifically in terms of accuracy, in-network processing, compression ratio and end-to-end delay, over the state-of-the-art algorithms.

The organization of the paper is as follows: Section II describes the specific problem targeted and the contributions of this work. It is followed by section III, which presents the system model and data set description. The section IV presents the proposed method followed by the section V which presents the details of the experimental and simulation environment. The results and detailed discussions are presented next in section VI followed by the concluding comments and future directions in section VII.

II. Problem Description and Contributions

In large-scale DL models, particularly those with billions of parameters, the communication of weight updates between distributed accelerators such as GPUs or TPUs becomes a significant bottleneck. During each iteration of distributed processing, vast amounts of weight data must be synchronized across accelerators, which leads to substantial energy consumption and communication time. As models grow in complexity, the number of weight updates increases exponentially, further amplifying communication overhead and reducing training efficiency. One approach to alleviate this problem

is to reduce the number of weight updates communicated between accelerators without sacrificing the model's accuracy.

By reducing the number of weight updates, techniques such as sparsification and pruning can significantly decrease the volume of data transferred between accelerators. These methods select only the most critical weights to be transmitted, leaving less significant updates out of the communication process. This reduces the communication time, which can be mathematically modeled as:

$$T_{comm} = \alpha + \beta \times \frac{n}{b} \tag{1}$$

where α is the communication latency (the fixed cost for initiating communication), β represents the time required to transmit a unit of data, n is the number of weight updates being transmitted, and b is the available bandwidth. By reducing n, the communication time decreases, which in turn minimizes the energy consumption. The energy consumption for communication can be expressed as:

$$E_{comm} = P_{comm} \times T_{comm} \tag{2}$$

where E_{comm} is the total energy consumed during communication, P_{comm} is the power consumed during communication, and T_{comm} is the communication time.

To ensure the accuracy and efficiency of the model are maintained, it must be taken into account that the available bandwidth limits the maximum rate at which data can be transmitted between accelerators. Reducing the number of weights helps alleviate bandwidth pressure, ensuring efficient data transfer, and is represented as $b \leq b_{max}$. The model's accuracy A must be preserved within acceptable limits, ensuring that pruning or sparsification does not degrade the model's accuracy beyond a predefined threshold $A_{min} \leq A \leq A_{max}$. The computational resources required for selecting and updating the weights must not exceed the total available resources. Efficient pruning and sparsification methods are necessary to ensure that the computational cost of reducing weight updates does not exceed the resources available, $R_{used} \leq R_{total}$. The overall communication latency must be kept within a predefined maximum limit, ensuring that communication delays do not compromise real-time or near-real-time performance, as represented by $T_{comm} \leq T_{max}$. By optimizing the number of weight updates and incorporating methods like pruning and sparsification, distributed deep learning systems can reduce communication time and energy consumption while maintaining high model performance. This allows for faster, energy-efficient training of large-scale models without overwhelming system resources or sacrificing accuracy.

III. System Model and Data Set Description

A. System Model

The system model outlines the design of an inter-accelerator communication framework using a 32-node cluster of Raspberry Pi $3B+$ devices. The cluster simulates a distributed DL environment where each Raspberry Pi functions as an individual accelerator. The proposed *ADL* is implemented on the motes and consists of five successively numbered layers of multiple deep compression layers. A convolution layer, batch normalization layer, and ReLU activation function make up the layer composition. Accordingly, the reconstruction layer is also composed of successive deep reconstruction layers. While layer 1 is designed to be fully connected, the layers from $2-5$ are considered to be de-convolution, and stride 2 design is considered for them. With the aim of obtaining a balanced robustness and efficiency ratio, a mini-batch gradient of size 32 is considered in this work. The proposed DL framework is applied over a gradient-based compression method to optimize model convergence and computational overhead. *ADL* is applied to every fully connected convolutional layer separately as an optimizer. The optimization is based on the idea to consider both input features and the accumulated residual gradient by dividing vectors into several fixed-length bins for each layer (algorithm 1). The neural network's final stages include a fully connected layer that employs matrix multiplication to convert it's input data into output features using a weight matrix.

B. Data Set Description

The proposed *ADL* along with all the compared state-of-the-art methods are implemented and the performance is compared on the following data sets:

1) CIFAR Dataset: The CIFAR-10 and CIFAR-100 [36] are popular labeled datasets with more than 50 million tiny images. The CIFAR-10 is divided into 10 classes with 6000 images per class, making it 60000 colour images. The 32×32 pixel images and are bifurcated in 5 training and 1 testing set, each containing 10000 images. The images in the test set are chosen randomly, with exactly 1000 images from each class. The CIFAR-100 is an extension of the CIFAR-10, with 100 classes, each comprising of 600 images. Each class is composed of 500 training images and 100 testing images, while the 100 classes together grouped to form a set of 20 super classes. For each image in the data set,'fine' and 'coarse' are used as the two labels for defining the class and super class, respectively.

2) MNIST Dataset: The MNIST dataset [37] is widely recognized as a benchmark in the fields of ML and computer vision (CV), specifically for tasks such as picture classification and digit recognition. This data set consists of $70,000$ gray scale images, each with a resolution of 30×30 pixels. The pixel intensity is represented by a gray scale value ranging from 0 to 128. The data set is divided into 3 subsets for different purposes: training, validation, and testing. The training set contains $50,000$ images, which are used to train the proposed *ADL* model in this work. The validation set comprises $10,000$ images and is utilized for hyper-parameter tuning and model evaluation during the training process. The remaining $10,000$ images from the test set are employed for the final evaluation of the trained model's performance. The images in the dataset exhibit variations in writing styles, forms, and sizes owing to the diverse sources for collection.

IV. Proposed Methodology

The DL model consists of a multi-layer architecture with *ReLU* as an activation function. The execution of the proposed *ADL* is defined by the following steps (shown in Figure 1):

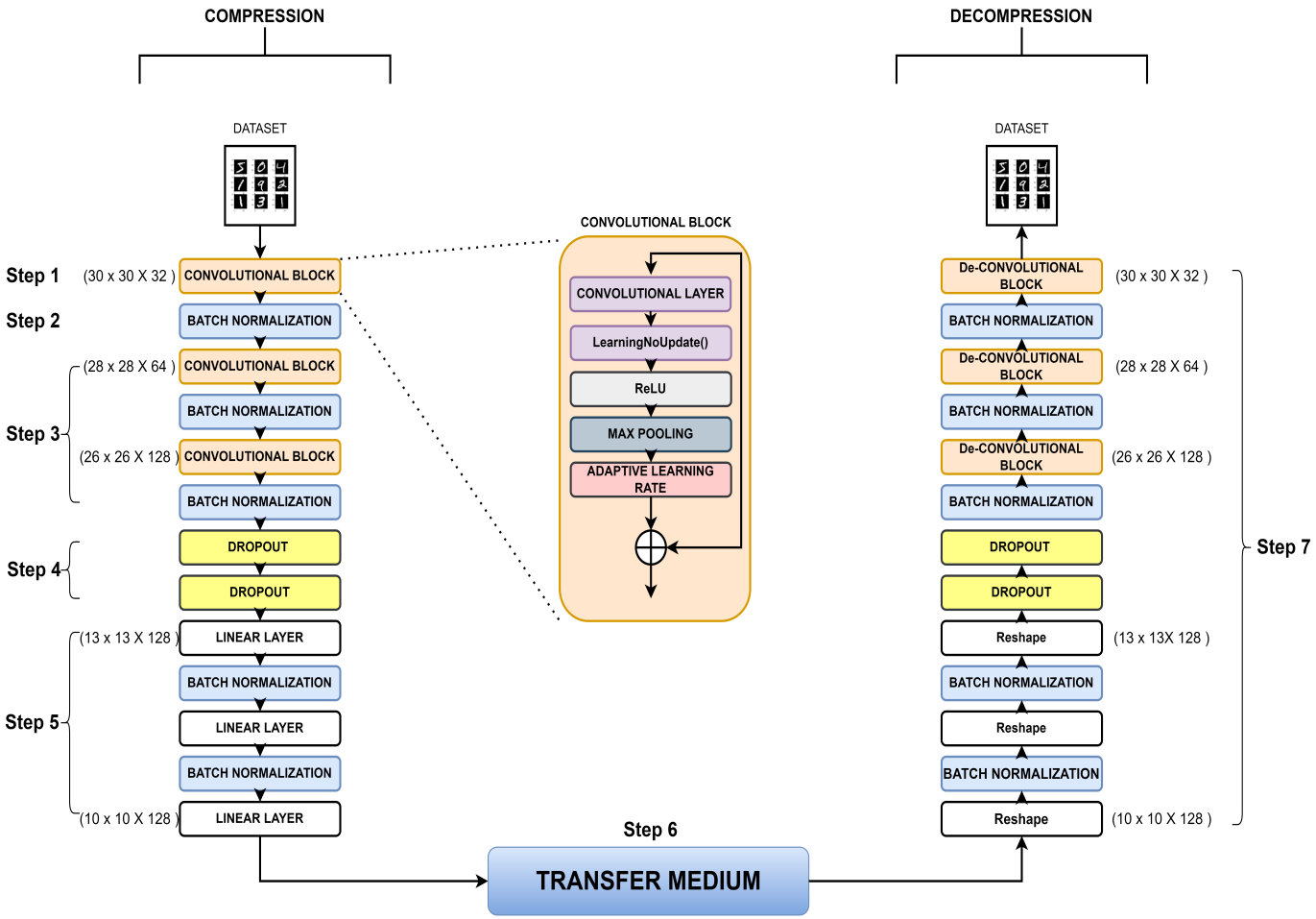


Fig. 1: System Architecture Diagram

1) *Convolutional Block:* The steps of execution are as follows:

   a) The input data is passed through the model, i.e. 1^{st} convolutional 2D layer. It adds a convolution layer with 32 filters that generate 2-dimensional feature maps to learn different aspects of the data with ReLU as an activation function.
   b) In the **learningNoUpdate()** phase, the algorithm learns the number update, followed by the **serializeGrad()**, which collects the gradient weightage of each layer in a vector to facilitate sequential weight update.
   c) A **Maxpooling2D** layer is used for feature extraction. The most relevant features are selected and are forwarded for further processing, while the remaining features are discarded.
   d) Adaptive learning rates are used to decrease the processing time, as shown in the equation 6, where *loss differential*, *learning rate* and *new learning rate*, are denoted by L_d, LR and $\hat{LR}$, respectively. The mathematical formulation to compute the values is inspired from the calculation presented in [38] and is optimized to fit the proposed *ADL*.

$$A_{y1} = \frac{1}{2} + \frac{L_d + \frac{1}{LR}}{5 \times 3.14} \tag{3}$$

$$A_{y21} = \frac{1}{4} + \frac{LR}{0.6 \times 3.14} \tag{4}$$

$$A_{y22} = \frac{1}{4} + \frac{L_d + \frac{1}{LR}}{5.5 \times 3.14} \tag{5}$$

$$\hat{LR} = \frac{LR}{2} + (L_d > 10^{-5}) \times LR \times A_{y1} + (L_d < 10^{-5}) \times LR \times (A_{y21} + A_{y22}) \tag{6}$$

2) *Batch normalization:* The output of the previous step is passed on to the batch normalization layer. Essentially, batch normalization widens the networks' intermediary layers, by normalizing to either the range of $[0, 1]$ or $[-1, 1]$ or to mean $= 0$ and variance $= 1$. The basic formula is :

$$a^* = \frac{(a - E[a])}{\sqrt{\text{var}(a)}}$$

where a^* represents the updated value of a single component, $E[a]$ represents it's batch mean, and $\mathrm{var}(a)$ represents it's batch variance.

3) The output of the previous step is iterated through two different convolutional blocks and batch normalization for further feature extraction and compression.
4) The output of the previous step is passed through 2 dropout layers which are applied at the rate of 0.25 and 0.5 respectively, to reduce the complexity of the model to prevent over fitting.
5) Furthermore, a batch normalizing layer is applied in between two separate linear layers. This enables the array to be flattened so that the convolution outputs (a matrix) may be fed into the fully connected layer (an array), allowing the compressed data to be properly transmitted from point A to point B.
6) The compressed data is then transmitted over the network and decompressed for reconstruction.
7) Reconstruction of original data is done with deconvolution block and reshape layers.

V. Experiment and Simulation Setup

The performance of the proposed *ADL* is validated through a series of experiments performed on a real test-bed of 32-node cluster of Raspberry Pi $3B+$ devices. The performance evaluation is further extended through extensive simulation analysis and the details of the experimental and simulation setup is presented below:

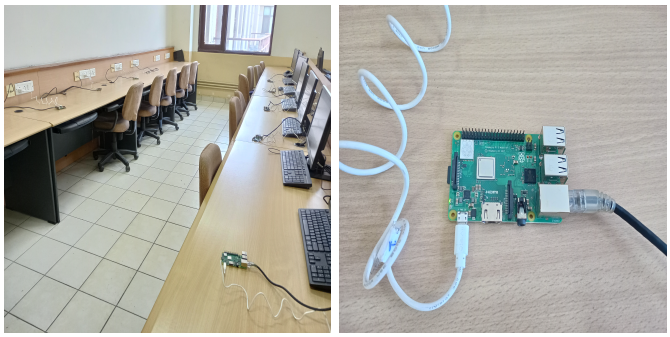
A. Experimental Setup

The experimental setup is deployed to replicate the framework for simulating the inter-accelerator communication using a 32-node cluster of Raspberry Pi $3B+$ devices. Each Raspberry Pi in the cluster, functions as a simulated accelerator within a distributed DL environment. The setup is designed to process the data independently on each node and the results are communicated and aggregated across the network. The Raspberry Pi $3B+$ devices, each equipped with a 1.4 GHz quad-core ARM Cortex-$A53$ processor, and have a 1 GB RAM with built-in ethernet. The motes are connected via ethernet cables to ensure low-latency communication. The cluster is powered by individual $5V/2.5A$ power supplies for each Raspberry Pi, and the devices are networked through a central network switch that supports all 32 nodes. The operating system for each Raspberry Pi is Raspbian OS, which is installed on microSD cards, providing a stable and familiar environment for development and execution. Coordinated execution is achieved through the use of libraries (MPI4Py, Dask, and Ray) for facilitating data distribution, parallel computation, and the synchronization of results among the nodes. The input data is distributed across the Raspberry Pi devices, and inter-accelerator communication is managed through message passing and synchronization protocols. Additionally, the model incorporates fault tolerance strategies like check-pointing and dynamic task reassignment to address potential node failures, as well as load balancing techniques to optimize resource utilization across the cluster. The results

Algorithm 1 Proposed *ADL*

Input : Original Image vector
Output : Re-Constructed image vector

procedure COMPRESS
  $Gr \leftarrow$ residue $+ Wt$
  $Z \leftarrow$ Residue $+ 2 \times Wt$
  Divide Gr into bins of size S
  for $i = 1$ to length$(Gr)/S$ **do**
    Calculate $Gr_{\max}(i)$
  end for
  for $i = 1$ to length$(Gr)/S$ **do**
    for $j = 1$ to S **do**
      index $\leftarrow (i-1) \times S + j$
      if $Z_{\text{index}} \geq Gr_{\max}(i)$ **then**
        Grq(index) $\leftarrow$ Quantize(Gr(index))
        Add Grq(index) to a compress vector
        residue(index) $\leftarrow Gr$(index) $\times Grq$(index)
      else
        residue(index) $\leftarrow Gr$(index)
      end if
    end for
  end for
  return compress_vector
end procedure
procedure DECOMPRESS(compress_vector)
  Initialize an empty array Gr of size length($compress\_vector$)
  Initialize an empty array Z of size length($compress\_vector$)
  Initialize S as the bin size used during compression
  Initialize Wt as the desired weight
  for $i = 1$ to length$(Gr)/S$ **do**
    Calculate $Gr_{\max}(i)$
  end for
  Initialize an empty array $residue$ of the same size as Gr
  for $i = 1$ to length$(Gr)/S$ **do**
    for $j = 1$ to S **do**
      index $\leftarrow (i-1) \times S + j$
      if $Z_{\text{index}} \geq Gr_{\max}(i)$ **then**
        Gr(index) $\leftarrow$ Dequantize($compress\_vector$[index])
        $residue$(index) $\leftarrow Gr$(index) $\times Gr$(index) $- Wt$
      else
        $residue$(index) $\leftarrow compress\_vector$[index] $- Wt$
      end if
    end for
  end for
  return $Reconstructed\_Image\_vector$
end procedure



(a) A cluster of Raspberry Pi 3B+ (b) Raspberry Pi 3B+

Fig. 2: Experimental Setup

TABLE I: Simulation Parameters and Metrics

Parameter	Value	Parameter	Formula
Average (γ) Correlation Coefficient	0.7	Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Test Batch Size	1000	Compression Ratio	$\frac{SIZE(initial)}{SIZE(compress)}$
Initial LR	0.005	Avg. Loss	$\frac{\sum_{i=1}^{n} \text{Loss}_i}{n}$
Stabilized (β) Training Loss	0.6	Proc. Time	$T_f - T_i$
Batch Size	32		
Epochs	100		
Dropout	0.25, 0.5		
Seed	50		
Log Interval	10		

are obtained for a series of evaluation parameters; namely in the terms of *Accuracy*, *Average Loss*, *Processing Time*, *Compression Ratio*, *Average Energy Consumption*, *End-to-End delay* and *Structural Similarity Index Measure (SSIM)*. The Figure 2 shows a magnified view of the experimental setup used for this study.

B. Simulation Setup

To validate the performance of the proposed *ADL*, extensive simulations were performed on a Dell Latitude 5420 with Intel core $I5-1145G7$ processor, frequency 2.60GHz, 16 GB Memory, and 256 GB SSD. The simulations were performed to replicate a distributed environment on a 64-bit Windows 10 operating system with Python 3.0 virtual environment for the execution of the code and for obtaining the results. For a fair analysis of the proposed *ADL*, the case with batch size $= 32$ and $LR = 0.005$, is simulated along with various optimizers, namely ADADelta [39], ADAGrad [40], Adam [41], ADAMax [41], and RMSprop [42]. The simulations are extended for implementing and comparing the out-cases of SGD [43], JPEG [44] [45], AWC [46] and FeCarNet [47] for a comparative evaluation of *ADL's* performance. The model is trained for 100 epochs with the seed value of 50 and a log interval of 10 to ensure reproducibility on publicly available *CIFAR-10*, *CIFAR-100 [36]* and *MNIST [37]* data sets. The results are obtained for a series of evaluation parameters; namely *Accuracy*, *Average Loss*, *Processing Time*, *Compression Ratio*, *Average Energy Consumption*, *End-to-End delay* and *SSIM*. The details of the simulation parameters and the required formulas for calculating the evaluation parameters, are presented in Table I.

VI. Results and Discussions

A. Experimental Results

The Table II presents a comprehensive performance analysis of the proposed *ADL* based on the experimental setup defined in section V-A. The outcomes of the proposed *ADL* are reported for varying batch sizes and LR. It is evident from the results that lower LRs (0.005 and 0.01) have a slower convergence because of the increased processing time and high energy consumption, as compared to the moderate LR (0.05), which provides an acceptable trade-off between time and network stability. The proposed *ADL* is able to show improved performance with higher LRs (0.5 and 1.0), as the training time is significantly reduced, resulting in improved convergence. Results show that the proposed *ADL* is able to significantly reduce the end-to-end delay as the minimum delay is reported to be 4.38 seconds, verifying the algorithms ability to improve the performance in almost real time applications. With the highest accuracy of 98.99%, highest SSIM at 96.79 and average energy consumption as low as 7.13 joules, the proposed *ADL* is able to achieve the desired objectives of improved performance in deep networks. Furthermore, the optimization of the local gradient residue allows maximum utilization of the low rank feature and sparsity in the data, leading to an improved energy performance and compression ratio by the *ADL* during the experiments.

B. Simulation Results

The performance assessment and comparative analysis of various optimization methods offer insightful information about their efficiency and viability. This section presents the findings from assessing the effectiveness of the proposed *ADL* and discusses how it stacks up against the existing state-of-the-art techniques. For the simulation scenario described in the section V-B, a comprehensive performance analysis is presented below. The simulation outcomes for the proposed *ADL* along with the compared methods on CIFAR-10, CIFAR-100 and MNIST datasets, are also presented in Table III.

*1) **Accuracy**:* The proposed *ADL* is able to outperform the other state-of-the-art optimizers and the SGD method proposed in [43], with a clear margin as the accuracy outcome is reported at 98.97% in comparison to the 98.81% for ADADelta, 98.68% for ADAGrad, 98.89% for Adam, 98.77% for RMSprop and 98.76% for SGD, at LR = 0.005 and batch size = 32. The findings, as reported in Table III and shown in Figure 3, are the outcomes of the proposed *ADL's* ability to allocate gradient weightage and sequential weight updation in the *learningNoUpdate()* and *serializeGrad()* phase. The results of efficient feature extraction through the Maxpooling 2D layer are seen in the algorithm's improved performance in terms of accuracy. A careful observation of the performance of the *ADL* on the CIFAR data sets shows that the method is equally

TABLE II: Experimental Performance of Proposed *ADL* with Varying Batch Size and Learning Rate

Batch Size	Learning Rate	Accuracy %	Average Loss	Proc. Time (sec)	Avg. Compression Ratio	Avg. Energy Consumption (J)	Delay (sec)	SSIM
32	0.003	98.82	0.04 ± 0.01	3.11	5.42	11.31	5.01	93.13
32	0.005	98.99	0.03 ± 0.01	3.56	5.60	12.67	5.99	96.47
32	0.01	98.69	0.03 ± 0.01	3.09	5.55	11.03	5.43	94.43
32	0.05	98.81	0.03 ± 0.02	2.99	5.51	10.09	5.01	95.03
32	0.5	98.93	0.04 ± 0.01	2.91	5.41	9.78	4.67	96.13
32	1	98.79	0.03 ± 0.01	2.82	5.50	7.13	4.38	96.79
64	0.005	98.86	0.03 ± 0.01	3.89	5.55	13.21	6.81	92.13
128	0.005	98.92	0.04 ± 0.02	4.95	5.42	14.87	7.20	92.13
256	0.005	98.93	0.05 ± 0.02	7.03	5.33	16.08	9.91	91.13

effective on coloured images, where the algorithm is able to achieve significantly improved accuracy of 89.19%, 59.11% on the CIFAR-10 and CIFAR-100 data sets, respectively.

*2) **Average Loss**:* The proposed *ADL* is able to preserve critical information through the series of CNN's applied through the proposed architecture, as shown in Figure 4. The impressively low average loss of the proposed *ADL* on MNIST data set, reported at 0.03, is the reason for improved data fidelity, transmission and data integrity. When compared to other state-of-the-art optimizers, such as ADADelta, ADAGrad, Adam, ADAMax, and RMSprop. As evident from Table III and Figure 4, the proposed *ADL* outperforms the compared approaches on all three data sets with a fair margin. The effect of adaptive LR in the proposed *ADL* is seen in terms of average loss when compared with the SGD [43] method, where the proposed *ADL* shows reduced average loss as the SGD [43] marks the average loss at 0.07, 1.95, 0.53 and FeCarNet [47] at 0.02, 1.68, 0.59 in comparison to the *ADL's* average loss of 0.03, 1.29 and 0.13 for the same parameters on MNIST, CIFAR-100 and CIFAR-10 data sets, respectively.

*3) **Processing Time**:* The applicability of the proposed *ADL* to real-time processing systems in bandwidth constrained networks, is marked by it's ability to strike a balance between optimization speed and accuracy, making it particularly suitable for large-scale ML applications where time-efficiency is critical. It can be clearly justified by the results of the processing time, reported in Table III, that the adaptive LR helps to enhance the overall stability and robustness of the model by preventing large fluctuations in the learning rate that can hinder convergence. It also dynamically adjusts the LR depending on the accuracy and other factors, at each iteration for better optimization of model parameters and weightage of each neuron links within interconnected convolution layers, that ultimately reduces the processing time of the proposed *ADL* without affecting the accuracy and also prevents it from underfitting or overfitting of the model. The processing of the proposed *ADL*, reported at 1.26 minutes for MNIST, 6.09 minutes for CIFAR-100 and 4.91 minutes for CIFAR-10, is significantly better than the state-of-the-art methods presented in Table III.

*4) **Compression Ratio**:* Figure 7 shows 16 randomly selected images from the MNIST data set. The selected images were subjected to compression through the proposed *ADL* along with the other identified optimizers. The outcomes of the experiment in terms of compression ratio, for respective inputs from Figure 7 are reported in Figure 8 and consistently demonstrate that the proposed *ADL* algorithm achieves competitive compression ratio across a variety of image resolutions. In addition to the above, a simple observation from the Table III reaffirms the algorithm's effectiveness in achieving higher compression ratios while preserving image details, as the proposed *ADL* consistantly maintains it's performance on simple grey scale (MNIST) as well as complex coloured data sets (CIFAR-10 and CIFAR-100). Through the integration of a convolutional layer combined with *learningNoUpdate()*, *serializeGrad()*, and the Max-pooling layer, the *ADL* algorithm surpasses by selectively retaining only the most important image pixels or features. This selective strategy enables the algorithm to accomplish higher compression ratio while preserving significant image details during subsequent image reconstruction. The results, as shown in the Table III, demonstrate that the *ADL* algorithm is ideally suited for a broad variety of applications where balancing compression ratios and image quality is of the utmost importance.

*5) **Energy Consumption**:* The results presented in Table III and Figure 5, show that the proposed *ADL* is able to achieve a superior performance in terms of average energy consumption. The impact of hyper-parameter optimization through the proposed *ADL*, is seen in the optimized computation and memory access requirements. The key hyper-parameters such as bit-width (precision of weights and activations), model size (number of parameters via pruning and sparsity), and network architecture (depth and number of filters) are optimized to ensure reduced energy consumption. The effect of bit-width precision, batch-size optimization, pruning, optimized loss rate and exploiting sparsity, reduces the number of non-zero weights, allowing for fewer operations, lower memory bandwidth requirements, and faster convergence, reducing the overall energy consumption. The model's adaptive learning ability and the integration of the convolutional layer combined with *learningNoUpdate()*, *serializeGrad()*, and the Max-pooling layer allows the algorithm to converge relatively quickly and minimises the processing time as well. The proposed *ADL* is able to achieve an average energy consumption of as low as 2247 joules on MNIST, 10475 joules on CIFAR-100 and 6693 joules on the CIFAR-10 data set. The dexterity of the proposed *ADL* is maintained over the compared state-of-the-art methods as the algorithm surpasses the existing methods with a minimum improvement of 23.01% in terms of average energy consumption. The proposed *ADL* has a reduced energy consumption rate as compared to traditional approaches such the JPEG [44] [45]. Such methods, in contrast to the

proposed *ADL*, use a fixed algorithmic pipeline with limited flexibility for energy reduction and lack tunable parameters to dynamically adjust computation or precision, making it less energy-efficient for specific tasks. Although JPEG itself is computationally simple and does not involve training, its use in deep learning pipelines introduces notable preprocessing overhead. In these systems, images compressed using JPEG require repeated decompression before being processed by the model, particularly during the training phase when large-scale datasets are involved. This continuous cycle of compression and decompression adds computational cost and significantly increases energy consumption, even when the JPEG algorithm is simple [45]. In contrast, the proposed *ADL* directly processes data without requiring intermediate compression-decompression steps, thus eliminating this overhead and reducing energy consumption.

6) ***End-to-End Delay****:* The performance of the proposed *ADL*, in terms of end-to-end delay, is presented for the three identified data sets in the Table III. Evidently, the proposed *ADL* outperforms the compared approaches and claims it's suitability for real-time and almost real-time processing systems. The results for the CIFAR-10 and CIFAR-100 proves the ability of the *ADL* to minimize the variations in the LR owing to the models capability for adaptive learning rate. This enhances the model's stability and robustness and ensures better convergence. Moreover, the ability to dynamically adjust the learning rate, based on neuron link weights, in interconnected convolution layers allows minimum end-to-end delay for the proposed *ADL*, without compromising the model's accuracy. The reported end-to-end delay of 3.69 minutes for MNIST, 7.07 minutes for CIFAR-100, and 5.11 minutes for CIFAR-10, shows the *ADL's* supremacy over the compared approaches.

7) ***Structural Similarity Index Measure****:* The SSIM is a normalized measure of the similarity of an image when compared to it's compressed version [48]. The SSIM is calculated on the basis of a window comparison method used for any two images of the same resolution such that the measure between the two windows of a and b is calculated as below:

$$SSIM(a,b) = \frac{(2\mu_a\mu_b + d_1)(2\sigma_{ab} + d_2)}{(\mu_a^2 + \mu_b^2 + d_1)((\sigma_a^2 + \sigma_b^2 + d_2)} \quad (7)$$

where μ_a and μ_b represent the average mean of 'a' and 'b' respectively and σ_a and σ_b are the average vairance of 'a' and 'b' respectively. Additionally, d_1 and d_2 are two variables. As discussed in the previous sections and validated through the results in Table III and Figure 6, the algorithm's high SSIM is another proof of the method's dexterity which is achieved through the integration of the convolutional layer combined with *learningNoUpdate()*, *serializeGrad()*, and the Max-pooling layer. The integration allows the *ADL* algorithm to not only maintain a high SSIM but also have a high compression ration without affecting the quality of the image.

VII. Conclusion and Future Directions

This work presents a DL architecture, termed as *ADL*, based on AC and CNN. Utilizing AC and hyper-parameter tuning, this work proposes a DL architecture for in-network performance optimization in large-scale image processing applications of deep networks. The proposed algorithm uses hyper-parameter tuning to optimize bit-width, model size and network depth of a deep learning framework, applied over a gradient-based compression method to optimize the local gradient residue, allowing maximum utilization of the low rank feature and sparsity in the data, leading to improved convergence and computational overhead. The performance of the proposed *ADL* is validated through a series of experiments performed on a real test-bed of 32-node cluster of Raspberry Pi $3B+$ devices. The performance evaluation is further extended through extensive simulation analysis on publicly available *CIFAR-10*, *CIFAR-100* and *MNIST* data sets. The proposed approach is able to outperform most of the state-of-the-art methods in terms of *Accuracy*, *Average Loss*, *Processing Time*, *Compression Ratio*, *Average Energy Consumption*, *End-to-End delay* and *SSIM*, achieving significantly good results for all the used data sets. The authors continue to extend this work by improving the training and validating the performance on more complex and diverse data sets.

References

[1] W. Jiang, Y. Zhang, H. Han, Z. Huang, Q. Li, and J. Mu, "Mobile traffic prediction in consumer applications: A multimodal deep learning approach," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 3425–3435, 2024.

[2] Q. Li, Z. Huang, W. Jiang, Z. Tang, and M. Song, "Quantum algorithms using infeasible solution constraints for collision-avoidance route planning," *IEEE Transactions on Consumer Electronics*, 2024.

[3] Z. Yang, W. Jiang, S. Huang, S. Chang, J. He, Y. Zhang, and Z. Feng, "Snr-enhanced automatic modulation classification in artificial intelligence of things for consumer electronics," *IEEE Transactions on Consumer Electronics*, 2025.

[4] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[5] F. Seiler and N. TaheriNejad, "Efficient image processing via memristive-based approximate in-memory computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 3312–3323, 2024.

[6] E. J. Candès *et al.*, "Compressive sampling," in *Proceedings of the international congress of mathematicians*, vol. 3. Madrid, Spain, 2006, pp. 1433–1452.

[7] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[8] H. Gan, Z. Guo, and F. Liu, "Nestd-net: Deep nesta-inspired unfolding network with dual-path deblocking structure for image compressive sensing," *IEEE Transactions on Image Processing*, vol. 33, pp. 1923–1937, 2024.

[9] Z. Xie, L. Liu, and Z. Chen, "Image compressed sensing: From deep learning to adaptive learning," *Knowledge-Based Systems*, vol. 293, p. 111659, 2024.

[10] M. Shen, H. Gan, C. Ma, C. Ning, H. Li, and F. Liu, "Mtc-csnet: Marrying transformer and convolution for image compressed sensing," *IEEE Transactions on Cybernetics*, vol. 54, no. 9, pp. 4949–4961, 2024.

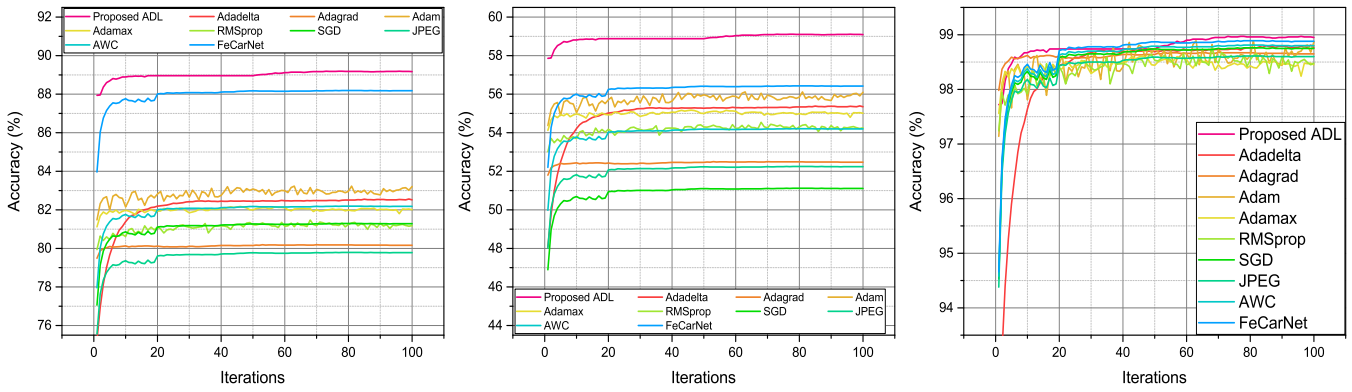
[11] V. K. Singh and M. Kumar, "A compressed sensing approach to resolve the energy hole problem in large scale wsns," *Wireless Personal Communications*, vol. 99, pp. 185–201, 2018.

[12] V. K. Singh, M. Kumar, and S. Verma, "Accurate detection of important events in wsns," *IEEE Systems Journal*, vol. 13, no. 1, pp. 248–257, 2017.

[13] ——, "Node scheduling and compressed sampling for event reporting in wsns," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 418–431, 2018.

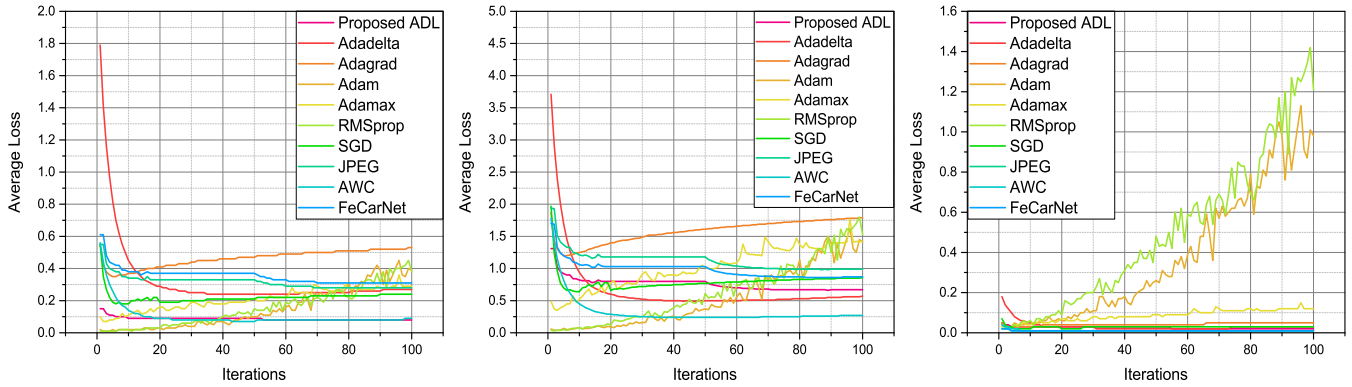
[14] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang, "Image and video compression with neural networks: A review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 6, pp. 1683–1698, 2019.

[15] J. Liang, M. Zhang, X. Zeng, and G. Yu, "Distributed dictionary learning for sparse representation in sensor networks," *IEEE Transactions on Image Processing*, vol. 23, no. 6, pp. 2528–2541, 2014.



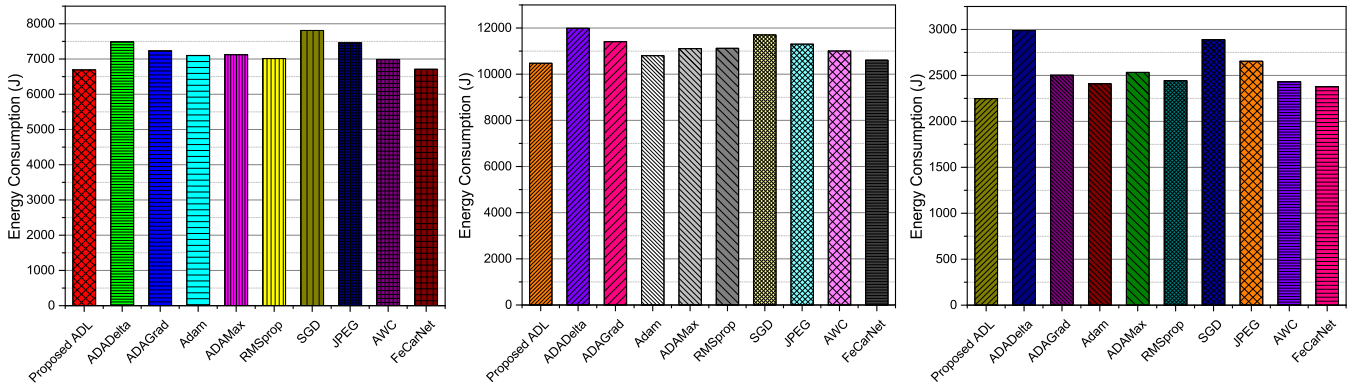
(a) Performance on CIFAR-10 (b) Performance on CIFAR-100 (c) Performance on MNIST

Fig. 3: Performance Evaluation - Accuracy of Proposed *ADL*



(a) Performance on CIFAR-10 (b) Performance on CIFAR-100 (c) Performance on MNIST

Fig. 4: Performance Evaluation - Average Loss of the Proposed *ADL*

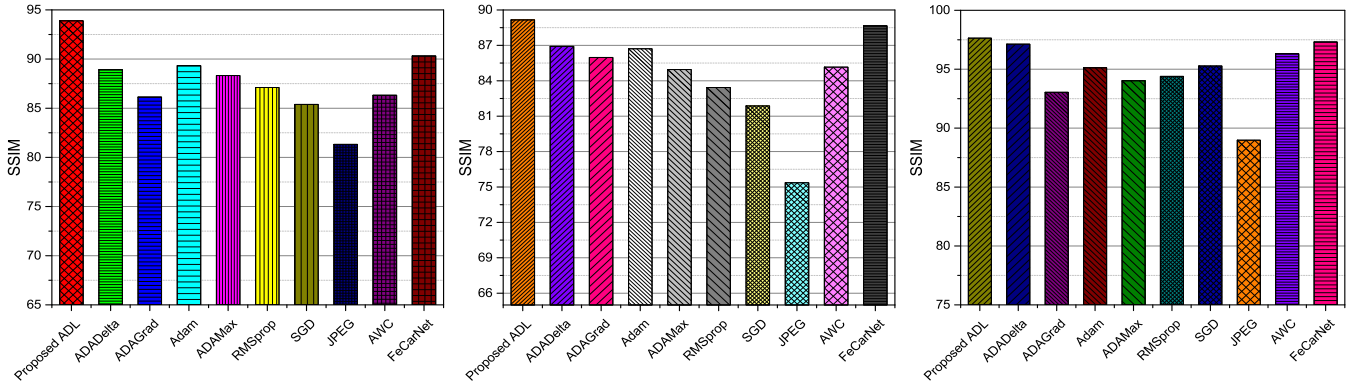


(a) Performance on CIFAR-10 (b) Performance on CIFAR-100 (c) Performance on MNIST

Fig. 5: Performance Evaluation - Energy Consumption of Proposed *ADL*

TABLE III: Simulation Performance Comparison of the Proposed *ADL* with State-of-the-Art Methods

Dataset	Algorithm	Accuracy %	Average Loss	Processing Time (min)	Compression Ratio	Avg. Energy Consumption (J)	Delay (min)	SSIM
CIFAR-10 [36]	**Proposed *ADL***	89.19	0.13 ± 0.02	4.91	3.97	6693	5.11	93.89
	ADADelta [39]	82.56	0.49 ± 0.06	5.10	3.67	7488	5.01	88.92
	ADAGrad [40]	80.19	0.53 ± 0.02	4.21	3.32	7231	5.43	86.13
	Adam [41]	83.23	0.43 ± 0.02	4.71	3.89	7102	5.71	89.32
	ADAMax [41]	82.17	0.34 ± 0.02	4.21	3.67	7123	5.93	88.32
	RMSprop [42]	81.47	0.43 ± 0.02	4.41	3.72	7011	6.43	87.11
	SGD [43]	81.29	0.53 ± 0.02	5.21	2.98	7809	6.91	85.39
	JPEG [44] [45]	79.19	0.53 ± 0.02	4.51	3.37	7459	5.32	81.32
	AWC [46]	82.19	0.54 ± 0.02	4.71	3.94	6987	6.39	86.32
	FeCarNet [47]	88.19	0.59 ± 0.02	5.01	3.95	6709	6.61	90.32
CIFAR-100 [36]	**Proposed *ADL***	59.11	1.29 ± 0.02	6.09	3.68	10475	7.07	89.16
	ADADelta [39]	55.39	1.12 ± 0.02	8.09	3.35	11988	8.89	86.92
	ADAGrad [40]	52.50	1.85 ± 0.02	7.53	3.27	11407	8.83	85.98
	Adam [41]	56.12	1.60 ± 0.02	7.45	3.54	10801	8.47	86.71
	ADAMax [41]	55.17	1.75 ± 0.02	7.27	3.61	11105	7.62	84.95
	RMSprop [42]	54.54	1.78 ± 0.02	7.12	3.52	11123	8.50	83.43
	SGD [43]	51.12	1.95 ± 0.02	7.81	2.71	11701	8.23	81.87
	JPEG [44] [45]	52.25	1.92 ± 0.02	7.46	3.12	11302	7.78	75.36
	AWC [46]	54.21	1.76 ± 0.02	7.17	3.36	11004	8.67	85.16
	FeCarNet [47]	56.43	1.68 ± 0.02	6.87	3.15	10605	8.32	88.65
MNIST [37]	**Proposed *ADL***	98.97	0.03 ± 0.01	1.26	4.86	2247	3.69	97.64
	ADADelta [39]	98.81	0.04 ± 0.01	2.27	4.11	2988	4.21	97.13
	ADAGrad [40]	98.68	0.03 ± 0.02	1.84	3.83	2503	3.82	93.04
	Adam [41]	98.89	1.01 ± 0.12	1.89	4.48	2409	4.87	95.12
	ADAMax [41]	98.61	0.15 ± 0.01	1.75	3.33	2533	4.86	94.02
	RMSprop [42]	98.77	1.23 ± 0.19	1.49	3.84	2443	3.79	94.39
	SGD [43]	98.76	0.07 ± 0.01	2.51	-	2889	4.32	95.28
	JPEG [44] [45]	98.61	0.02 ± 0.01	1.27	3.79	2654	3.94	88.98
	AWC [46]	98.81	0.04 ± 0.01	1.27	4.79	2431	3.73	96.31
	FeCarNet [47]	98.89	0.02 ± 0.01	1.37	4.79	2378	3.69	97.31



(a) Performance on CIFAR-10 (b) Performance on CIFAR-100 (c) Performance on MNIST

Fig. 6: Performance Evaluation - SSIM of the Proposed *ADL*

[16] C. Afef, C. Émilie, and D. Marc-André, "Fast dictionary-based approach for mass spectrometry data analysis," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 816–820.

[17] L. Shao, R. Yan, X. Li, and Y. Liu, "From heuristic optimization to dictionary learning: A review and comprehensive comparison of image denoising algorithms," *IEEE transactions on cybernetics*, vol. 44, no. 7, pp. 1001–1013, 2013.

[18] J. Anitha, P. Eben Sophia, and D. Jude Hemanth, "An optimized predictive coding algorithm for medical image compression," in *Artificial Intelligence: Second International Conference, SLAAI-ICAI 2018, Moratuwa, Sri Lanka, December 20, 2018, Revised Selected Papers 2*. Springer, 2019, pp. 315–324.

[19] Z. Yan, J. Wang, L. Sheng, and Z. Yang, "An effective compression algorithm for real-time transmission data using predictive coding with mixed models of lstm and xgboost," *Neurocomputing*, vol. 462, pp. 247–259, 2021.

[20] H. Latha and A. Rama Prasath, "Icpch: A hybrid approach for lossless dicom image compression using combined approach of linear predictive coding and huffman coding with wavelets," in *International Conference on Cognition and Recongition*. Springer, 2021, pp. 269–281.

[21] T. S. Shinde, "Efficient motion estimation and predictive coding methods for compression of spatio-temporal sequences." Ph.D. dissertation, Indian Institute of Technology Jodhpur, 2020.

[22] Z. Yin, Z. Wu, W. Shi, G. Hu, and W. Lin, "Video compressed sensing via wavelet residual sampling and dual-domain fusion," *IEEE Transactions on Multimedia*, 2025.

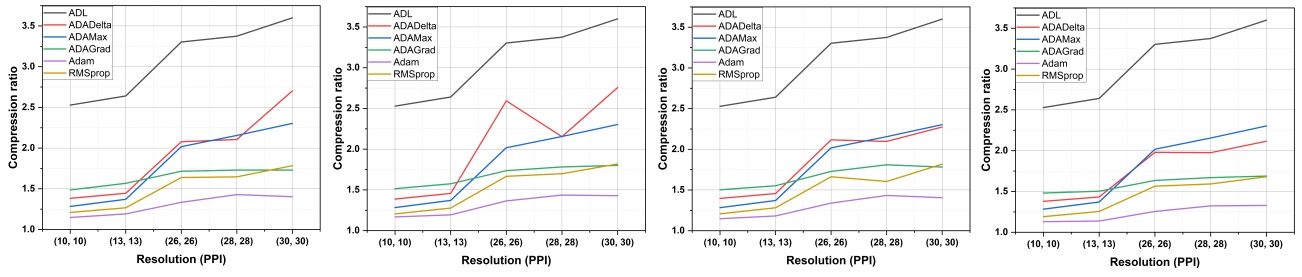
[23] H. Nuha, B. Liu, M. Mohandes, A. Balghonaim, and F. Fekri, "Seismic data modeling and compression using particle swarm optimization," *Arabian Journal of Geosciences*, vol. 14, pp. 1–11, 2021.

[24] W. X. Zhao, X. Zhang, D. Lemire, D. Shan, J.-Y. Nie, H. Yan, and J.-R. Wen, "A general simd-based approach to accelerating compression algorithms," *ACM Transactions on Information Systems (TOIS)*, vol. 33, no. 3, pp. 1–28, 2015.

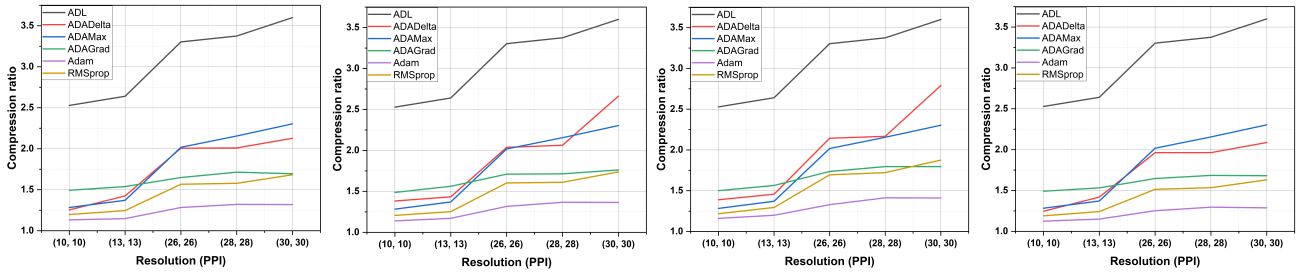
[25] S. Banerjee, R. Gupta, and J. Saha, "Compression of multilead electrocardiogram using principal component analysis and machine learning approach," in *2018 IEEE Applied Signal Processing Conference (ASPCON)*. IEEE, 2018, pp. 24–28.

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o) (p)

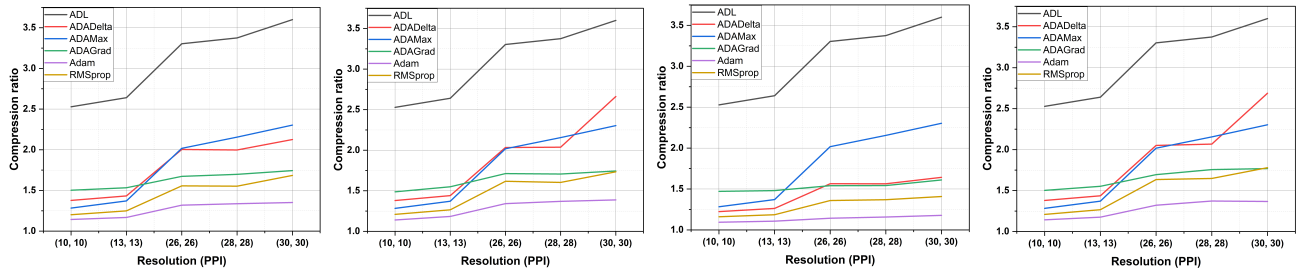
Fig. 7: Random 16 Images from MNIST Data Set Used for Compression Analysis



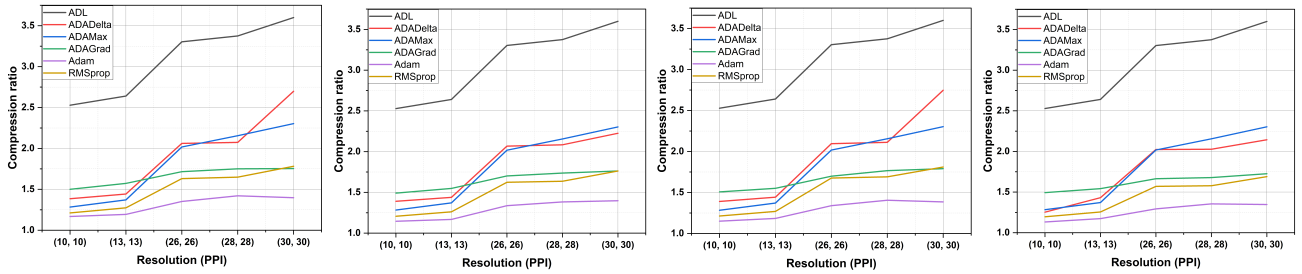
(a) Result of Fig. 7(a) (b) Result of Fig. 7(b) (c) Result of Fig. 7(c) (d) Result of Fig. 7(d)



(e) Result of Fig. 7(e) (f) Result of Fig. 7(f) (g) Result of Fig. 7(g) (h) Result of Fig. 7(h)



(i) Result of Fig. 7(i) (j) Result of Fig. 7(j) (k) Result of Fig. 7(k) (l) Result of Fig. 7(l)



(m) Result of Fig. 7(m) (n) Result of Fig. 7(n) (o) Result of Fig. 7(o) (p) Result of Fig. 7(p)

Fig. 8: Compression Ratio vs Resolution for 16 Randomly Selected Images

[26] X. Ma, Z. Huang, X. Li, Y. Qi, L. Wang, and Z. Zhu, "Multiobjectivization of single-objective optimization in evolutionary computation: a survey," *IEEE Transactions on Cybernetics*, 2021.

[27] R. Huang, K. Rhee, and S. Uchida, "A parallel image encryption method based on compressive sensing," *Multimedia tools and applications*, vol. 72, no. 1, pp. 71–93, 2014.

[28] N. Zhou, A. Zhang, J. Wu, D. Pei, and Y. Yang, "Novel hybrid image compression–encryption algorithm based on compressive sensing," *Optik*, vol. 125, no. 18, pp. 5075–5080, 2014.

[29] A. Alfalou and C. Brosseau, "Optical image compression and encryption methods," *Advances in Optics and Photonics*, vol. 1, no. 3, pp. 589–636, 2009.

[30] N. Zhou, S. Pan, S. Cheng, and Z. Zhou, "Image compression–encryption scheme based on hyper-chaotic system and 2d compressive sensing," *Optics & Laser Technology*, vol. 82, pp. 121–133, 2016.

[31] V. K. Singh, B. Nathani, and M. Kumar, "Weed-mc: Wavelet trans-

form for energy efficient data gathering and matrix completion," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1066–1073, 2019.
[32] A. Ali, I. Ullah, S. K. Singh, W. Jiang, F. Alturise, and X. Bai, "Attention-driven graph convolutional networks for deadline-constrained virtual machine task allocation in edge computing," *IEEE Transactions on Consumer Electronics*, 2025.
[33] V. K. Singh, N. Jain, G. Tripathi, and S. Sahani, "Correlated channeled spatio temporal graph attention network model for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2025.
[34] V. K. Singh, N. Anand, A. Pal, A. Chowdhury, and A. Srivastava, "Anomaly detection in dynamic graphs using multiple encoding strategies via transformers," *IEEE Transactions on Consumer Electronics*, 2025.
[35] W. Jiang, A. Liu, Y. Zhang, H. Han, J. Mu, S. Liu, W. Gu, S. Huang, and Z. Feng, "Towards global metaverse accessibility with rsma-based satellite-terrestrial integrated networks: A game theoretic approach," *IEEE Transactions on Consumer Electronics*, 2025.
[36] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[37] Y. Raut, T. Tiwari, P. Pande, and P. Thakar, "Image compression using convolutional autoencoder," in *ICDSMLA 2019: Proceedings of the 1st International Conference on Data Science, Machine Learning and Applications*. Springer, 2020, pp. 221–230.
[38] F. Ni, J. Zhang, and M. N. Noori, "Deep learning for data anomaly detection and data compression of a long-span suspension bridge," *Computer-Aided Civil and Infrastructure Engineering*, vol. 35, no. 7, pp. 685–700, 2020.
[39] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
[40] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.
[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[42] D. P. Williams, R. Hamon, and I. D. Gerg, "On the benefit of multiple representations with convolutional neural networks for improved target classification using sonar data," *Training*, vol. 2008, no. 29280, p. 2912, 2013.
[43] G. Yan, T. Li, S.-L. Huang, T. Lan, and L. Song, "Ac-sgd: Adaptively compressed sgd for communication-efficient distributed learning," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 9, pp. 2678–2693, 2022.
[44] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.
[45] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan, "Deepn-jpeg: A deep neural network favorable jpeg-based image compression framework," in *Proceedings of the 55th annual design automation conference*, 2018, pp. 1–6.
[46] 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*. IEEE, 2017, pp. 199–204.
[47] H. Chen, X. He, H. Yang, L. Qing, and Q. Teng, "A feature-enriched deep convolutional neural network for jpeg image compression artifacts reduction and its applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 1, pp. 430–444, 2021.
[48] I. F. Ince, F. Bulut, I. Kilic, M. E. Yildirim, and O. F. Ince, "Low dynamic range discrete cosine transform (ldr-dct) for high-performance jpeg image compression," *The Visual Computer*, vol. 38, no. 5, pp. 1845–1870, 2022.

Vishal K. Singh received his bachelor's degree in Information Technology, in 2010, the master's degree in Computer Technology and Application, in 2013, and PhD degree in Information Technology from Indian Institute of Information Technology, Allahabad, India in 2018. He is currently working as a Lecturer and is associated with the Networks and Communications Research Group at School of Computer Science and Electronics Engineering, University of Essex, Colchester, U.K. His research interests include Internet of Things, Wireless Sensor Networks, In-Network Inference, Machine Learning and Data Analytics.

Jagpreet Singh received the B.Tech. degree in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2003, the M.S. degree in software systems from the Birla Institute of Technology and Sciences, Pilani, in 2009, and the Ph.D. degree in computer science and engineering from the Indian Institute of Technology Ropar, India, in 2015. He was an Assistant Professor with the Indian Institute of Information Technology Allahabad, from 2015 to 2022. He has been an Assistant Professor with the Indian Institute of Technology Ropar, since 2022. His research interests include parallel and distributed systems, scheduling theory, high-performance computing, and wireless sensor networks.

Dr. Rajkumar Singh Rathore (Senior Member IEEE) is working as Head of Cyber Security of Connected and Autonomous Systems, CINC, Head of Cyber Physical and Networks Systems, CeRISS & Programme Director for MSc Computing and IT in Department of Computer Science at Cardiff Metropolitan University's School of Technologies, United Kingdom. His research works were fully supported by Nottingham Trent University, United Kingdom and Manchester Metropolitan University, United Kingdom.

Devansh Nema received the bachelor's degree in Computer Science and Engg. in 2019 from SRMIST, Chennai, India and M.tech in Computer Science in 2023. His research interests include Machine Learning, Internet of Things, Image Processing, and Data Analytic.