IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 15 November 2022, accepted 9 December 2022, date of publication 15 December 2022, date of current version 5 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3230008

RESEARCH ARTICLE

Resource-Restricted Environments Based Memory-Efficient Compressed Convolutional Neural Network Model for Image-Level Object Classification

ZAHRA WAHEED[1], SHEHZAD KHALID[1], SYED MURSLEEN RIAZ[2], SAJID GUL KHAWAJA[2], AND RIMSHA TARIQ[2]

[1]Department of Computer Engineering, Bahria University, Islamabad 44000, Pakistan
[2]Department of Computer Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Rawalpindi 46000, Pakistan

Corresponding author: Zahra Waheed (engg.zahra@gmail.com)

ABSTRACT In the past decade, Convolutional Neural Networks (CNNs) have achieved tremendous success in solving complex classification problems. CNN architectures require an excessive number of computations to achieve high accuracy. However, these models are deficient due to the heavy cost of storage and energy, which prohibits the application of CNNs to resource-constrained edge-devices. Hence, developing aggressive optimization schemes for efficient deployment of CNNs on edge devices has become the most important requirement. To find the optimal approach, we present a resource-limited environment based memory-efficient network compression model for image-level object classification. The main aim is to compress CNN architecture by achieving low computational cost and memory requirements without dropping system's accuracy. To achieve the said goal, we propose a network compression strategy, that works in a collaborative manner, where Soft Filter Pruning is first applied to reduce the computational cost of the model. In the next step, the model is divided into No-Pruning Layers (NP-Layers) and Pruning Layers (P-Layers). Incremental Quantization is applied to P-Layers due to irregular weights distribution, while for NP-Layers, we propose a novel Optimized Quantization algorithm for the quantization of weights up to optimal levels obtained from the Optimizer. This scheme is designed to achieve the best trade-off between compression ratio and accuracy of the model. Our proposed system is validated for image-level object classification on LeNet-5, CIFAR-quick, and VGG-16 networks using MNIST, CIFAR-10, and ImageNet ILSVRC2012 datasets respectively. We have achieved high compression ratio with negligible accuracy drop, outperforming the state-of-the-art methods.

INDEX TERMS Memory-efficient network compression, pruning, quantization, image-level object classification, resource-restricted edge-devices.

I. INTRODUCTION

Deep Learning made impressive success in the field of artificial intelligence including many computer vision and pattern recognition tasks and has been under intensive research [1]. It is known for decades due to the vital factors contributing to its emerging success such as the use of big data, robust feature extraction, and powerful computational resources. Amongst deep learning algorithms, Convolutional Neural Networks (CNNs) have evolved as state-of-the-art and have accomplished record-breaking results in applications of computer vision such as image classification, object detection, and natural language processing [2]. The superior performance of deep networks is closely related to the depth and width of the network, such that, the networks with a large number of layers and millions of trainable parameters achieve excellent

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

inference accuracy [3]. Regardless of the excellent performance of deep networks, the promising results of CNNs are credited to millions of parameters, computations (FLOPs) and storage requirements [2]. However, this in turn lays a heavy burden on memory and computational resources. For example, AlexNet has about 61 million parameters in the ILSVRC 2012 classification challenge, which is 100 times that of the LeNet model. Similarly, ResNet-152 needs to perform 11.3 billion FLOPs for the classification of an image of 224 $\times$ 224 with a capacity of 230MB in the ImageNet classification challenge in 2015 [2]. Since more parameters demand more storage requirements, energy consumption, and more floating-point operations (FLOPs), hence exceeding the computing power of the edge devices. Therefore, it becomes extremely challenging to deploy deep CNNs on resource-constrained edge devices with low computational and power budgets. This has made it essential to downsize CNN to have low computational costs while having high performance.

Although CNN models usually require a large number of parameters to ensure their high performance, there is a lot of redundancy in their parameterization, resulting in high deployment costs and limited application scenarios [4]. In this context, network compression has gained increasing attention for reducing deployment costs in both memory and computation. To obtain a lightweight network with low storage and computational costs, researchers have put a considerable amount of effort into the compression of CNN architectures [2]. Various methods are presented to achieve compression while maintaining system performance, including Pruning, Quantization, Low-Rank Decomposition, weight sharing, etc. [5]. Amongst these existing methods, Network Pruning has attracted great attention from researchers in the past recent years.

Pruning is considered the most reasonable method used to reduce the model size (number of parameters) and floating-point operations (FLOPs) by removing non-critical or redundant parameters from the network. Moreover, pruning not only reduces the network's complexity but also tends to reduce over-fitting and improves generalization [6]. Figure 1 illustrates the effect of pruning on CNN architecture. In addition, quantization is also a widely accepted network compression method, which attempts to directly compress the full-precision model and reduce a lot of memory [2]. Figure 2 demonstrates an example of quantization where data is divided into finite levels and each level is assigned a specific value to reduce high precision parameters to low bit-width.

It has been observed in research that pruning and quantization can be very effective in reducing the number and precision of parameters in deep networks [7]. Additionally, pruning can reduce the number of weights by up to 90%, as described by Han et a. [8]. Despite the success of these two methods, there is still a need to design an efficient compression model that can reduce memory consumption while preserving the system's performance [9]. This idea will facilitate the application of deep CNNs on resource-constrained

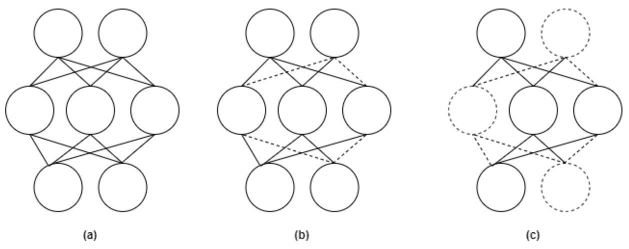


FIGURE 1. Effect of pruning on neural network architecture (a) Original Unpruned Network, (b) Unstructured pruned network (c) Structured pruned network.

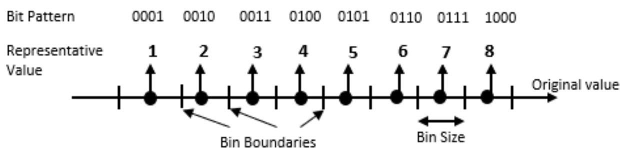


FIGURE 2. Quantization example.

edge devices. In this context, deep compression methods have been under investigation for the past recent years, that combine two or more compression techniques to run deep networks on resource-restricted devices with fast inference and low memory and energy requirements. Deep compression methods intend to achieve a high compression ratio by making good utilization of pruning and quantization both while preserving model's performance. This would result in a small storage requirement, that all trainable parameters can easily be stored on-chip with less energy consumption

However, this paper aims to provide a memory-efficient network compression framework to optimize the memory utilization of the CNN model. The main target of providing this system is to overcome memory and power limitations while applying CNN on resource-restricted devices. The proposed framework is a combination of two broadly used network compression techniques; pruning and quantization. It makes efficient use of these two methods while maintaining the system's overall performance. First, the original model undergoes pruning to eliminate parameter redundancy from the network while in the next step, weights are quantized for reducing the number of bits required to represent weight parameters. This not only reduces the number of computations but also reduces the overall memory requirement of the model, hence, making the deployment of CNNs on edge devices more feasible. Some of the major contributions of the proposed framework are as follows,

1) A memory-efficient network compression framework is proposed by integrating network pruning and quantization methods to obtain a low-memory and high-performance CNN model for resource-restricted edge devices.
2) We propose a novel pipelined network compression scheme that works in two stages. In the first stage,

we compress the model by applying Soft-Filter Pruning (SFP). In the second stage, different parts of the network are quantized using two different quantization methods. The proposed Optimized Quantization method is applied to starting layers (named as *NP-layers*)of the network while Incremental Quantization (INQ) is applied to the latter layers (named as *P-layers*) of the network. To the best of the author's knowledge, this is the first attempt to integrate pruning and quantization in this manner.

3) We propose a novel quantization algorithm Optimized Quantization (OQ) for the starting trainable layers of the network. Unlike traditional quantization methods that attempt to quantify parameters randomly up to selected *levels*, this proposed method first obtains optimal *levels* to which quantization has to be carried out using the proposed Optimizer.
4) This work proposes a novel optimizer structure aimed at obtaining optimal quantization *levels* with minimal accuracy loss and memory requirements. This Optimizer works in a pipeline, where quantization and Canonical Huffman coding is performed to get reduced memory and minimum accuracy loss. In the next step, regression is applied to determine the relationship of memory and error with other parameters of the model. Finally, the Genetic Algorithm is used to get the best *levels* with minimum fitness value. This is the first network compression model where Optimizer is used with the unique cause of obtaining optimal *levels* for the quantization process.
5) We have introduced two preference variables α and β for the selection of the final compressed model after applying OQ. α is related to accuracy loss, while β refers to the amount of memory saved, where the sum of the two variables equals 1. Based on user preference, we decide the value of each variable, for example, if accuracy loss and memory saved are equally critical, then we set $\alpha = \beta = 0.5$.

The rest of the paper is organized as follows, and Section II discusses various network compression methods based on pruning and quantization. We introduce our proposed network compression framework and corresponding algorithms in Section III. Datasets and CNN models details along with extensive experimental results are given in Section IV while Section V concludes the paper.

II. RELATED WORK

Over the past few years, various methods have been used for the acceleration and optimization of CNNs. Considering the burden of increasing computational complexity of CNNs, network compression is the most widely used CNN optimization method. Network compression basically aims at building a compressed CNN model that is fast and flexible when implemented on embedded systems. In this context, different CNN compression methods have been investigated, and in this section, we mainly focus on two widely used network compression techniques, namely network pruning and quantization. We have summarized some of the prominent network compression methods used in the literature.

A. NETWORK COMPRESSION USING PRUNING METHODS

Pruning is considered one of the pioneer network compression methods which have been applied broadly in the past decade. Weight pruning is the most popular unstructured pruning method that has been used for network compression. The first weight pruning method i.e. Optimal Brain Damage (OBD) is introduced in [10]. It removes weights based on their saliency measure in an iterative manner. It results with 60% pruning of LeNet when it is trained on MNIST dataset with accuracy higher than the original model. The main shortcoming that comes with this method is that OBD did not consider larger network like AlexNet and ResNet in its work. Considering this problem, authors have come up with new methods where different and better regularization techniques are used for measuring weight importance for larger networks. As in [11], an improved weight pruning method is presented which is known as Optimal Brain Surgeon (OBS). It has used Taylor expansion to select the least important weights which need to be pruned. Results have shown that this method outperforms OBD method but more computational cost is added due to fine-tuning. Furthermore, an enhanced version of weight pruning is presented in [12], which is based on Incremental Pruning Less Training. This method eliminates least important weights from filters rather than eliminating the entire filter from the network. This method works in two steps; in first step, weights pruning is carried out based on L2-norm regularization with less training which hampers the overall performance of the system. In second step, pruned network is retrained to recover the accuracy loss. System has achieved substantial improvement in accuracy i.e. more than 90% for both VGG-lite and LeNet models when they are tested on MNIST and CIFAR datasets in 20 epochs which is less than the time consumed by conventional methods. T-H. Chen et al. [13] have performed weight pruning based on statistical analysis of weights distributions. Threshold required to prune least important weights is obtained by applying Gaussian function due to gaussian spread of weights. Mask is determined by thresholding whose dot product is taken with original weights. This results with least important weights whose values are close to zero, which are then pruned to get a sparse network. 1/3 of weights are pruned for ResNet and AlexNet models when tested on CIFAR dataset with an accuracy drop of around 0.8% in 100 epochs. S. Moon et al. [14] have proposed a novel memory-reduced multiple accuracy pruning method. This method is combination of multiple CNN optimization techniques. In first step, different pruning ratios is used to get a stacked-CNN architecture where upper part is accuracy-aware obtained by reviving pruned weights from lower energy-aware part of the network. Weights of most accurate networks are stored only which has reduced memory utilization. In addition, multi-indexing technique is used to store indices of surviving weights that belong to the pruned network. The best energy-accuracy trade-off has been achieved in edge-level devices by using this method.

On the other side, different methods have been studied under the category of structured pruning, which are aimed to remove redundant structure (i.e. filter, channel or layer) from the network despite of removing parameters from a structure. In this context, a filter-based pruning method is proposed in [15]. It globally finds the importance of all filters and dynamically prunes them, followed by fine-tuning process to recall the filters which have been pruned mistakenly. The system is tested for three large networks i.e. AlexNet, VGG-16 and ResNet on ImageNet2012 dataset. Results have shown that 2.12x speedup and 1.15% top-5 accuracy loss has been achieved for AlexNet, that is superior to existing methods. Another filter pruning method based upon two fuzzy membership functions is given by authors in [6]. Two fuzzy functions are used to compute the degree of importance of filters. The decision of filter pruning is made by incrementally alpha-cut the least important filters. This method is tested for VGG-16 and VGG-19 models on MNIST and CIFAR datasets. It is observed that 74% and 77% storage reduction has occurred for VGG-16 and VGG-19 respectively. FLOPs reduction occurred around 63% and 39% for VGG-16 and VGG-19 respectively with 40% filter reduction. This has significantly reduced the overall computational complexity of the network. Xu Han et al. [8] has proposed a Parasitic-Mechanism (PAM) based filter pruning. In first step, Parasitic Layer is constructed which intelligently learns unimportant filters during training process for pruning. The parasitic layer is constructed on the basis of convolutional layer in two steps; first filters are soft-pruned under PAM algorithm using the defined criterion while in the second step, the network is tuned to compensate accuracy loss, hence ended up in promising results than traditional methods. An improved filter pruning method is proposed by M. M. Pasandi [16]. It gradually prunes weak filters based on L1-norm and STD which are used as pruning criterions. It recovers weak filters back into the network from attenuation. This novel concept of not eliminating filters completely and recovering them from attenuation would help to avoid degradation in system performance that may occur due to pruning. It is found that high accuracy i.e. greater than 90% is achieved when system is tested on CIFAR-10 dataset for VGG-16 model. A soft-mask filter pruning method is given by authors in [17]. It prunes output feature map of least important filters based on L2-norm regularization while preserving filter weights. This results in a compact network with an accuracy of 99% and FLOPs reduction of 41% when it is tested on CIFAR dataset for ResNet model. It is found that fine-tuning is considered the core step of pruning which takes good amount of time to carry out the whole process, hence increasing the computational cost of the network. In [18], authors have presented a No Fine-tuning method which takes off the computation burden of fine-tuning by adding network slimming process. Network slimming basically set a buffer to store a contribution value against each filter. This method has successfully reduced 34% parameters and 37% FLOPs when system is tested for ResNet-50 model. Furthermore, another CNN compression method is presented in [15], which is based on channel pruning. This method is Dynamic Channel Pruning (DCP) which dynamically prunes the least important channels and multiplies rest of the channels with weights in CONV layers. DCP has shown promising results when it is tested on ILSVCRC2012 and CIFAR datasets for VGG-16 and ResNet-50 models, achieving 3x speedup and 1.96% accuracy loss. Another channel pruning method via gradient of mutual information is given in [19]. This method outperforms the state-of-the-art pruning methods in terms of parameters and FLOPs when it is tested on CIFAR dataset for VGG-16, ResNet-20 and DenseNet-40 models. A dynamic channel pruning method is presented in [20], which adds channel-wise sparsity at the output of each convolutional layer by introducing Learning Kernel-Activation switch module (LKAM). This module activates and de-activates each kernel dynamically depending upon the input content. This whole process allows the relevant kernels to perform all required computation while stopping the not relevant kernels from performing any computation using LKAM. This has significantly reduced computational complexity. System has shown promising results when implemented on embedded environments for CaffeNet and SqueezeNet models. Moreover, an improved method of channel pruning is presented in [21]. In this method, channel is pruned based on sparsity index which is calculated by using the defined sparsity function. Results have shown that memory reduction of 4.32MB, 3.69MB and 3.9MB occurred when system is tested on CIFAR for VGG-16, GoogleNet and ResNet respectively in 200 epochs with accuracy drop under 3%. A layer pruning method is presented in [22]. It prunes least important layer by using Partial Least Square (PLS) projection as estimation criterion. It is found that later layers of the network contribute less to the classification accuracy so layer pruning is performed starting from end of the network. Whereas rest of the layers are pruned using filter based pruning method. This method has achieved a significant FLOPs reduction of 62.69% which is superior than state-of-the-art methods when tested on CIFAR dataset for ResNet-20, ResNet-56 and ResNet-110 models.

B. NETWORK COMPRESSION USING QUANTIZATION METHODS

Quantization-based network pruning has also been in studies in the past decade, which is used to off-load computational burden on large CNNs in order to speedup inference process. A power-efficient parameter quantization technique has been presented in [23]. This method performs quantization followed by fine-tuning process and then integrated with hardware accelerator to further speed up inference. For weight quantization, gaussian nature of weights distribution is considered. Using that, few samples for large-valued weights are chosen while more samples of small-valued weights are chosen. In the next step, fine-tuning is performed to recover accuracy loss. The minimum error of 0.24% and 0.39% is achieved for VGG-16 and AlexNet respectively with 8-bits representation using Tiny ImageNet dataset. This quantized model is given to multiplier-less hardware to execute MAC operations and a power reduction of up to 14.2% has achieved. In [24], a hybrid weight quantization method is given which is focused to apply CNN on embedded environments. This method also aims to minimize the burden of re-training which is performed to recover accuracy loss that occurs after quantization. The method works in two steps; firstly, uniform quantization of weights is carried out while in the second step quantized weights undergo k-mean clustering to further quantize weights to the optimal number of bits. This hybrid quantizer gives high accuracy with 4 and 5 bits when tested for AlexNet to perform object detection on ImageNet dataset. S. Kim and H. Kim [25] presented an adaptive quantization method which is a mixture of two types of data quantization i.e. deterministic and stochastic quantization. Deterministic quantization converts weights to the nearest sampling point without considering probability while stochastic quantization is based on the probability of how close a weight value is to its adjacent points. Stochastic quantization is applied to a certain portion of weights while deterministic is applied to the rest of the weight values. The method is tested for image classification using VGG-16 and object detection using YOLOV3 models on CIFAR and COCO datasets respectively. It is found that higher accuracy is achieved when mixture of both methods is used rather than using each method sole for 4-bits quantization of weights into fixed-point format. Another adaptive weights quantization method is used in [26] in order to facilitate the deployment of CNN on edge devices. The quantization interval is determined using double exponential probability and then converted to fixed-point 8-bit representation. 87% of accuracy is achieved by testing system on chest X-Rays images for the detection of pneumonia. It is also found that the model reduced up to 3.9x than original with minimum accuracy loss using 255 quantization intervals. Z. Bao et al. [27] has presented a hardware-aware quantization method for weights and activation on the basis of the learnable clipping method (LSFQ). This method has multiple stages where in first step, low-bit width weights and activations quantization is performed using linear symmetric quantization followed by soft-clipping using a learnt clipper. In next step, FC and CONV layers are fused with the Batch Normalization layer as CONV-BN and FC-BN in order to further reduce inference latency by eliminating additional computational overhead. This process is further enhanced by incorporating LSFQ with specialized hardware to further support inference process. System is evaluated for VGG7, VGG7-tiny2 and mobileNet-v2 models on CIFAR10 and CIFAR100 datasets. Further increase in accuracy is observed when LSFQ is tested on DAC-SDC dataset adding more practicability of the system. In [28], a computational complexity-aware regularization method is proposed to speedup inference process by determining optimal bit allocation using gradient descent algorithm. It actually computes the product of computational complexity in terms of MAC operations and computing performance as MACxbit metric. Results have revealed that 21% reduction in inference time is achieved when the method is applied on ImageNet for ResNet18, ResNet50 and MobileNetv2 with improved accuracy. S. Cho and S. Yoo [29] have discussed a novel Per-Channel Quantization Level Allocation (PCQLA) method to minimize the accuracy loss that occurs when 2-bit quantization is applied on ResNet-18/50. Results have shown that 68.9% and 74.9% of accuracy are achieved for 2-bit activation and 1-bit weights quantization using ResNet-18 and ResNet-50 respectively. In [30], the training-aware deep compression method is presented in three steps. In first step, first layer of network is kept standard as it contains the significant information that is critical to feature extraction. For middle layers, depth-wise separable convolution is applied in place of traditional convolution to reduce number pf parameters. This is viewed as training-aware pruning. In the last step, extreme binary depth-wise separable convolution is implemented to reduce memory utilization. This is viewed as training-aware quantization. This scheme is different from the conventional deep compression method as it takes one round of process and has shown better results than traditional methods.

It is found from the above discussion that there comes a significant improvement in the performance when multiple compression techniques are used at the same time. However, it is observed in the literature that most of the authors attempt to apply a single compression method for achieving network compression. Moreover, most of the methods include the step of fine-tuning in both pruning and quantization to re-train compressed networks to recover accuracy loss. This adds an extra burden on computational cost and resources, hence degrading the overall system's performance. In addition, the authors have restricted their research to the use of small networks, which are not sufficient enough to assess the performance of compression methods with the perspective of applying them to resource-restricted edge devices. Keeping these research gaps in mind, we propose a novel memory-efficient compression framework that involves two broadly deployed compression techniques, pruning and quantization. It performs filter pruning followed by weight quantization to reduce the number of parameters and bit-width of each weight parameter to minimize memory utilization while maintaining the system's high performance. Therefore, making the application of CNNs on resource-constrained devices more realistic.

III. PROPOSED SYSTEM

We propose a memory-efficient network compression model of convolutional neural networks for resource-restricted edge devices. The general overview of the proposed system is illustrated in Figure 3. It takes the original CNN model as input to be compressed. Initially, all layers of the model undergo pruning to eliminate redundant parameters resulting in a Sparse Network that is more error resilient and easier

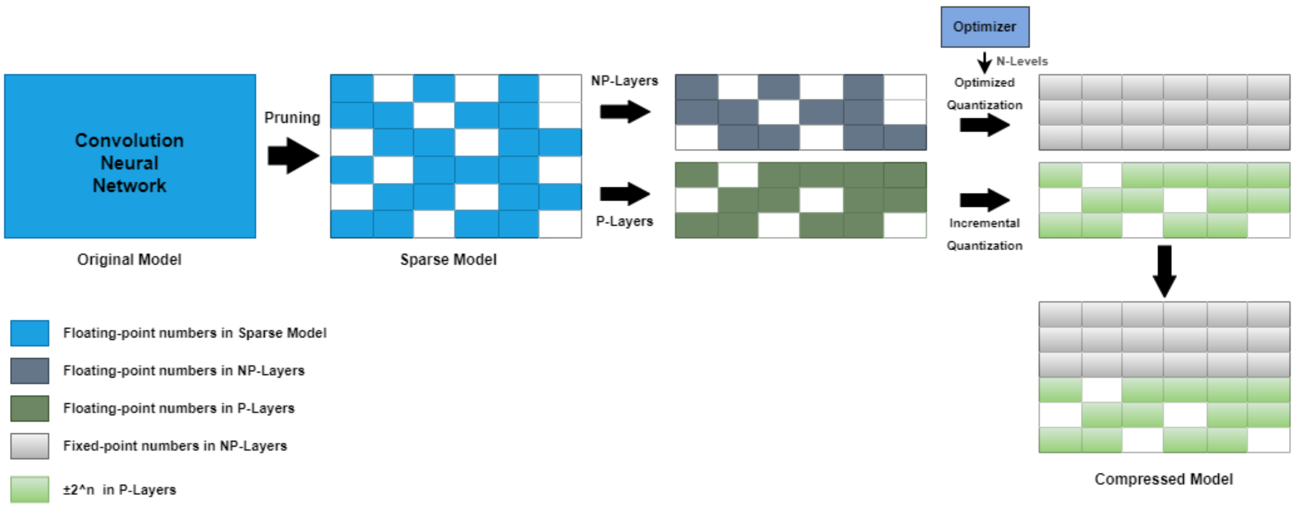

Optimizer
N-Levels
Optimized Quantization
Convolution Neural Network
Pruning
NP-Layers
P-Layers
Incremental Quantization
Original Model
Sparse Model
Compressed Model
Floating-point numbers in Sparse Model
Floating-point numbers in NP-Layers
Floating-point numbers in P-Layers
Fixed-point numbers in NP-Layers
±2^n in P-Layers

FIGURE 3. General overview of the proposed compression Framework: The output compressed network is obtained by applying network pruning, layers partitioning into NP-layers and P-layers, Quantization on both types of layers. parameters are converted from floating point 32-bit representation into low-bit width representation.

to converge. In the next step, we divide layers of the network into two categories; Pruning Layers (P-Layers) and No Pruning Layers (NP-Layers). The reason for dividing the layers is as the front layers are considered less error resilient and they contain significant information which is required to extract low-level features like edges and lines that are critical to subsequent feature extraction. Therefore, front layers are kept as NP-layers which include CONV layers only.

However, the weight distribution of the last layers becomes sparse as we move deep into the network, so the last layers are P-layers, which mostly include FC layers or FC and CONV layers both. First, all layers are pruned to obtain a sparse weight distribution for the model. In the next step, quantization is applied to both types of network layers. NP-layers undergo Optimized Quantization (OQ) whereas Incremental Quantization (INQ) is applied on P-layers of the network. We propose a novel optimized quantization algorithm that quantizes the weight parameters to lower bit widths up to the best level obtained from the optimizer. This quantization process initially trains the pruned network for NP-layers which turns the sparse distribution of weights into regular distribution and then converts parameters in floating-point representation into the fixed-point format as can be seen in Figure 3. While on the other side, an Incremental quantization (INQ) technique is applied on P-layers due to its sparse structure to convert parameters in floating-point representation into $\pm 2^n$ format which further reduces the memory requirement. In the end, we obtain a Compressed CNN Model which is front regular and back irregular having less number of parameters and less memory requirement than that of the original model.

Our main goal is to reduce the storage required to run inference faster on large networks so that they can be deployed on mobile devices. To achieve this goal, we present a two-stage pipelined network compression method that aims to compress the network to an extreme without losing the system's accuracy. The pipelined structure of the proposed method is given in Figure 4. First, we perform pruning on the original network by applying the Soft Filter Pruning (SFP) algorithm to reduce redundant parameters while keeping only the most critical ones in the network. In the second stage, the pruned network is divided into P-layers and NP-layers as explained earlier. Next, quantization is applied to both types of layers. We propose an Optimized Quantization algorithm for NP-layers. Quantization is carried out up to optimal *N-levels* obtained from the optimizer. We have also proposed a novel structure of Optimizer which is input to Optimized Quantization. The optimizer compresses the weights of the NP-layers using a quantizer and canonical Huffman coding. Next, we feed the results to a regression algorithm followed by Genetic Algorithm (GA), to obtain the *N-levels* needed for optimized quantization, resulting in NP-net. In parallel, Incremental Quantization (INQ) is applied on P-layers which results in P-net. Finally, the union of both P-net and NP-net gives the required compressed network which is more robust than the original one.

A. SOFT FILTER PRUNING

In the beginning, we applied Soft Filter Pruning (SFP) [31] on all layers of the network, which dynamically prunes the filters of each layer in a soft manner. Unlike the conventional hard pruning methods, which directly eliminate the filters from the network and dramatically degrade the system's performance, the soft pruning method intends to remove the least important filters dynamically during training. This approach allows the pruned filters to be updated during the training process which helps to maintain system performance, and the model's capacity. Further, it accelerates the system by taking off the

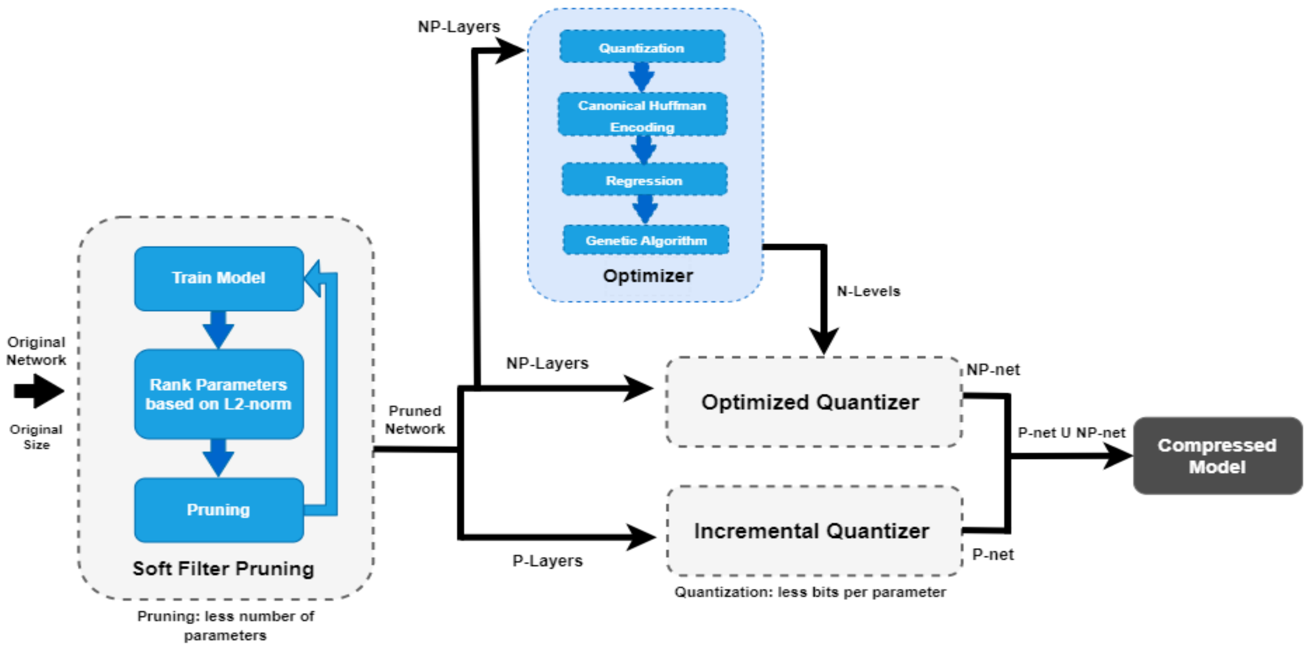


FIGURE 4. **Two-Staged pipelined Network compression scheme. Stage 1: Pruning of original network to obtain sparse network. Stage 2: Different quantization methods to apply on NP-layers and P-Layers separately to reduce number of bits required to represent parameters. Optimizer being part of stage-2 obtains optimal levels to quantify parameters of NP-layers in OQ. While INQ is applied to quantify parameters of P-layers into $\pm 2^n$ representation.**

Algorithm 1 Soft Filter Pruning

Input: Training data X, pruning ratio **(P_i)**,models weight parameters and filters of layer **L(W_L and F_L)**, **epochmax,** threshold_error
Output: Compact Model with less #filters per layer **F_L*** and less weight parameters **W*** such that **F_L*** < **F** and **W*** < **W**
Procedure:

```
1. for epoch = 1; epoch < epochmax; epoch++ do
2. Traning/Reconstruction: Train and update model parameters.
   if (error > threshold_error)
3.   for i=l; i < L; i++ do
4.     Filter Ranking and Selection: Calculate L2-norm for each filler
       || $F_{ij}$ $||_2$, $1 \leq j \geq N_{i+1}$ and rank them based on their importance.
5.     Filter Pruning: Set least important $N_{i+1}$ $P_i$ filters to 0.
6.   end for
7. end for
8. Return Compact Model with less number of filters and weight parameters.
```

burden of fine-tuning after pruning as it has been performed in conventional methods.

The SFP is an iterative filter pruning method, which performs filter selection, pruning, and retraining iteratively until the system converges. It avoids greedy layer-by-layer pruning of filters rather it enables pruning in all layers at the same time. More specifically, it prunes the filters during the training process. In each training epoch, it first trains the network, then the L2-norm of all filters in each layer is computed and all filters are ranked based on their L2-norm values. The least important filters are selected for pruning and set to 0, followed by the next training epoch. The whole process is repeated until convergence.

Let's assume a CNN network that is parameterized by $W_L(i)$ representing the matrix of weight connections for the *ith* layer where L indicates a total number of layers. The details of SFP are given in **Algorithm 1**, which is divided into the following four steps.

1) FILTERS RANKING AND SELECTION

We first obtain the importance of filters for each layer using the L2-norm. In general, convolutional results of filters with small L2-norm are more likely to have lower activation values, and thus have less impact on the overall prediction of the model. Therefore, these filters with low L2-norm have a high priority to be pruned than those with high L2-norm. Particularly, we use *Pi%* as the pruning rate which determines how many least important filters to prune from the *ith* layer. In other words, if there are N_{i+1} number of filters in *ith*-layer, then *Pi%* removes $N_{i+1}P_i$ least important filters from each layer as indicated by blue in Figure 5a. In practice, the L2-norm is used for weights ranking and selection. It is represented as given in (1) where $F_{i,j}$ represents *jth* filter of *ith* layer.

$$||F_{i,j}||_p = \sqrt{\sum_{n=1}^{N}\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}|F_{i,j}(n,k_1,k_2)|^p} \tag{1}$$

2) WEIGHT PRUNING

Once we obtain ranks of filters based on the L2-norm, we prune *Pi%* filters from the *ith* layer by setting $N_{i+1}Pi$

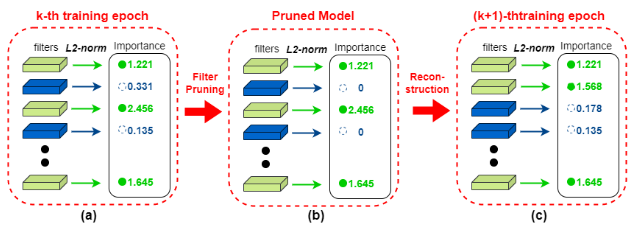


FIGURE 5. Overview of SFP. At every training epoch, we rank filters based on L2-norm and select least important filters as indicated by blue ones in (a). Then we prune the selected filters and set them 0 as indicated in (b). After pruning, filters undergo reconstruction where pruned filters are updated to non-zero as indicated in (c).

filters to zero as given in Figure 5b. This can temporarily remove their contribution to the output of the network. While in the next training stage, we allow these pruned filters to be updated to keep the model's capacity and performance high. In this step of filter pruning, filters of all weighted layers are pruned at the same time, which would save computational costs. Further, we keep our hyper-parameter i.e. pruning rate same for all layers as $P_i = P$ to maintain a balance between acceleration and accuracy. This can further avoid inconvenient hyper-parameter analysis.

3) RECONSTRUCTION

After pruning, we go to the next training epoch to reconstruct our pruned filters. This can be easily observed in figure 5c, where the pruned filter is updated to a non-zero value by backpropagation at the next epoch, as shown in blue. This step would allow the system to maintain the model's capacity, unlike hard pruning methods. Thus, we integrate the pruning step with the normal training process, and fine-tuning step is not necessary for SFP.

Algorithm 2 Incremental Quantization (INQ)

```
Input: P-Layers, Weight matrix $\mathbf{W_L}$ of layer L, grouping ratio r
Output: Quantized Network for P-Layers (P-net)
Procedure:
 1. for all layers ∈ P do
 2.     Compute Mask $\mathbf{T_L}$ for each $\mathbf{W_L}$ using Equation
 3. end for
 4. set grouping ratio r
 5. for all layers ∈ P do
 6.     Divide index of 1s in $\mathbf{T_L}$ into two groups $\mathbf{A_1}$ and $\mathbf{A_2}$ Using
         grouping ratio r
 7.     Set $\mathbf{T_L(i)}$ to 0 (i ∈ $A_2$)
 8.     Quantify $\mathbf{W_L(i)}$ (i ∈ $A_2$) using Equation
 9. end for
 10. Retrain and update weights by Equation.
 11. Go to Step 4 and repeat until all weights in P-layers have been
     quantified.
 12. Return Quantized Network (P-net)
```

4) OBTAINING COMPACT MODEL

SFP iterates over weight selection, weight pruning, and reconstruction steps. After the system converges to the optimal solution, iteration stops, and we obtain a compact model containing many "zero filters." Each "zero filter" corresponds to the output feature map. The resulting compact model is a sparse network with less number of parameters than the original one.

B. INCREMENTAL QUANTIZATION (INQ)

We divide layers of the pruned network into two categories: P-layers and NP-layers as has been discussed earlier in this section. Incremental Quantization [2] is applied on P-layers to further compress the model. This algorithm quantifies weights into low bit-width dynamically. It takes a compact network obtained in the previous step of pruning as input. For layer L, first, we determine a binary array T_L of the same size as W_L (weights array of layer L). T_L is computed using equation (2) mentioned below,

$$T_L(i) = \begin{cases} 0 & W_L \ = 0 \\ 1 & W_L \neq 0 \end{cases} \tag{2}$$

T_L is obtained for all P-layers where 0 and 1 correspond to zero and non-zero weights respectively. At the time of quantization, 1s in the T_L array are randomly divided into two groups: A_1 and A_2 using ratio r as has been mentioned in **Algorithm 2**. All entries of T_L for A_2 indexes are set to 0 indicating that corresponding weights will be quantified. The weight quantization in P-layers is represented by (3):

$$W_L(i) = \begin{cases} +2^{|log_2|W_L(i)||} & W_L \ > 0 \\ -2^{|log_2|W_L(i)||} & W_L \ \leq 0 \end{cases} \tag{3}$$

By using this quantization, weights will be quantified to lower bit-width of $\pm 2^n$ format which only required sign bit and exponent to be stored for direction and number of bits to shift respectively. It replaces the original multiplication operation in MAC with shift operation. After quantization, the model has been retrained to compensate for the accuracy loss that occurs due to quantization, and weights are updated as represented by line-10 of the **Algorithm 2** using equation (4).

$$W_L(i) \leftarrow W_L(i) - \eta \frac{\partial E}{\partial W_L(i)} T_L(i) \tag{4}$$

where η is the learning rate, E is the objective function and $T_L(i)$ denotes the mask of weights, which determines whether the weight must be updated. Corresponding weight will not be updated by zero entries of T_L indicating that it is already zero or has been quantified as illustrated in Figure 6. This whole process continues until all weights in P-layers have been quantified. This algorithm results in a compressed model having all weights of P-layers in $\pm 2^n$ format requiring less amount of memory due to low bit-width and small quantity. Moreover, the multiplication operation is replaced by to shift operation which further reduces the resource requirement of the network. This would be benefiting the model to have high performance when implemented on hardware.

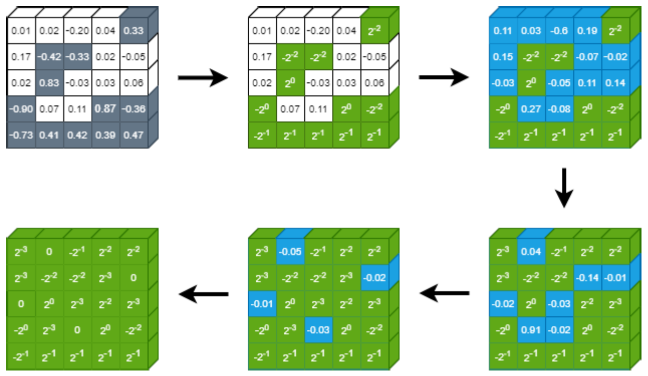


FIGURE 6. **Illustration of INQ Results. *1st row* shows three operations occur in first iteration. Left cube of first row corresponds to weight partitioning into two disjoint groups A_1 and A_2 using ratio r. Second cube corresponds to results after applying weight quantization on A_2 as shown in green cells. Third (right) cube shows the results of retraining non-quantified weights (A_1) of second cube as shown by blue cells. *2nd row* corresponds to results of next iteration until all weights get quantified.**

C. OPTIMIZED QUANTIZATION

We propose a novel Optimized Quantization method to compress weights in NP-layers. Optimizer is considered as the main component of this algorithm as it determines optimal levels of quantization *N-levels* to perform optimized quantization as illustrated in Figure 4. Optimizer is composed of data compression, regression, and Genetic Algorithm (GA). The data compression module is further made up of a quantizer and Huffman encoder. The complete structure of Optimizer is given in Figure 7, which takes trained weights of NP-Layers as input. The first task is to collect data for three different types of quantization methods (*Q-types*) followed by a Huffman encoder. Data to collect includes new memory obtained after applying the Huffman encoder, error computed after implementing three *Q-types*, and model parameters. The obtained data file is given to the regressor to find a relation between accuracy loss that may occur during quantization, parameters, and NP-layers of the model. Similarly, the relationship is found between said parameters with the memory of encoded weights. The regression equations found at the end of the regression process are fed into GA in the next step. GA takes all required parameters as input to determine *N-levels* across each Q-type and returns the best *N-levels* based on minimum fitness value. Finally, these optimized *N-levels* obtained from the Optimizer are given to Optimized Quantization to carry out basic quantization on the weights of NP-layers.

1) OPTIMIZER

As has been discussed above, Optimizer works in a pipeline manner including three main steps: 1) Data Compression using Quantization and Canonical Huffman Encoding, 2) Regression and 3) Genetic Algorithm as shown in Figure 7.

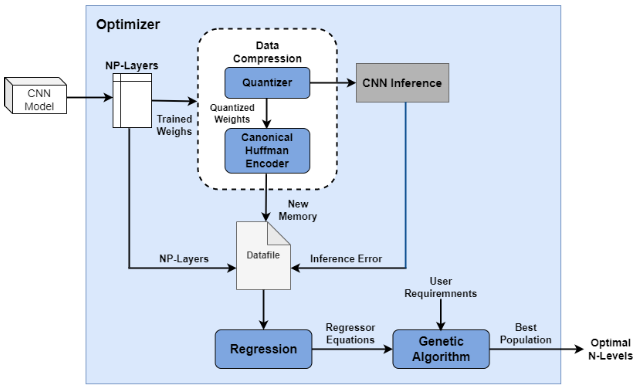


FIGURE 7. **Inside Structure of the proposed Optimizer. It includes three steps, i) Data Compression using Quantization and Canonical Huffman Endcoding, ii) Regression and iii) Genetic Algorithm.**

a: DATA COMPRESSION

In our proposed structure of Optimizer, Quantization and Canonical Huffman Encoding are used to perform data compression on pre-trained weights of NP-layers taken as input. Firstly, weights of NP-layers are quantified for different levels of quantization (N-levels) using three Q-types including uniform quantization, non-uniform quantization, and asymmetric quantization. In the next step of data compression, Canonical Huffman Coding [32] is applied to previously quantified weights we have obtained for all three Q-types. Encoding is done to further compress weights into lower bits to further reduce memory size. We calculate new memory for encoded weights to be saved in the data file as it is illustrated in Figure 4. Furthermore, the compression ratio and percentage of saved memory for all values of N-levels have been computed using equations 5 and 6 to be stored in the data file.

$$CompressionRatio = \frac{OriginalMemory}{CompressedMemory} \tag{5}$$

$$Memory_{saved} = \frac{|mem_{new} - mem_{orig}|}{mem_{orig}} \tag{6}$$

b: REGRESSION

Once all required data has been stored in the data file, regression is applied to fit our data perfectly for memory and accuracy loss as depicted in Figure 4. We apply different regressions and then propose an algorithm to select the best regression based on the least mean square error (MSE). This algorithm takes the data file and list of regressors we want to apply as input and MSE is computed for all regressors as given in **Algorithm 3**. This results in the best regressor with the one having the least average MSE along with the regression equations for accuracy loss and memory.

c: GENETIC ALGORITHM

As discussed earlier and illustrated in Figure 7, the final step in Optimizer is applying Genetic Algorithm [33] to determine

Algorithm 3 Selection Best Regression Equation

```
Input: Datafile, FR (Regressor function array)
Output: out[Ft, MSE]
Procedure:
    t←0
    Re ← Rm ← $\varphi$ # Re: regressor error
                         # Rm: regressor memory
    train_m, test_m ← split(shuffle(Datafile))
    train_e, test_e ← split(shuffle(Datafile))
    While (t == length(FR) −1) do:
        Eq1 ← FRt(train_e) # for error
        Eq2 ← FRt(train_m) # for memory
        Ye = Eq1. predict(test_e)
        Ym = Eq1. Lpredict(test_m)
            $E \leftarrow \frac{1}{n}\sum_{n}^{0}(Ye, Test\_e)^2$   (7)
            $M \leftarrow \frac{1}{n}\sum_{n}^{0}(Ym, Test\_m)^2$   (8)
        Re U E U FRt, Rm U M U FRt, Rm U M U FRt
    End
    FR error ← Min(Re, E)
    FR mem ← Min(Rm, M)
    Return FR error, FR me
```

optimal quantization levels (*N-levels*) to be used in Optimized Quantization. It takes user inputs along with the regression equations obtained in the previous step, model parameters, and layers while returning best *N-levels* for quantization. The inputs required by GA are; Population size required to initialize population (*size*), user acceptable error (*er*), percentage of memory to be saved (*mr*), number of generations (*gen*), threshold (*threh*), preferable parameters (α, β), regression equations for error and memory (*reg_e*,*reg_m*), CNN parameters and layers (*par*, *lay*). However, the outputs returned by the GA are; the fitness of final populations(*mse*), the fitness of each individual in the final population (*Find*), and the final population (*Po*).

The complete workflow of GA is given in **Algorithm 4**. In the first iteration, the population is initialized using the population initialization function which is then passed to the fitness function. Next, said population is modified according to individual fitness values in the population modification stage. To terminate the GA, one of two conditions must be satisfied i.e. either the MSE value of the current population is less than the user-defined threshold or the number of generations is completed. GA keeps on iterating if any of these conditions are not fulfilled. GA is divided into three steps as discussed below.

i) POPULATION INITIALIZATION

This is used just once to randomly initialize the population of individuals to enable the processing of GA. The size of the population defines the number of individuals in the population and is defined by the user depending upon the given problem. For the given system, our population is composed of different quantization levels each representing an integer and

Algorithm 4 Genetic Algorithm

```
Input   :size, er, mr, gen, thresh, a, 0, reg_e, reg_r, par, lay
Output  :F ind, Mse, Po Procedure:
    t←0
    Po ← initialize(size)
    While(t < gen) do:
    F indt,Mset ← Fitness(Po, er, mr, $\alpha$, $\beta$, reg e, reg r, par, lay,)
    if (M set <= thresh)do:
        break
    Pot ← Modification(F_ind, Pot)
    End
    Return F_ind, Mse, Po
```

we need to determine the best quantization levels (*N-levels*). We have defined *N-levels* as the initial population of GA, its format is given below,

$$P_o = array([c1], [c2], \ldots\ldots\ldots.[cM]) \tag{9}$$

ii) FITNESS EVALUATION

A fitness function is an objective function that determines how close a particular population is to the user requirements using a single fitness value. To achieve the said goal, previously obtained regression equations are used. Fitness values for each individual in the population and fitness for the entire population are calculated. Based on these computed fitness values, the population is modified to generate a new population. This keeps on computing fitness for all generations until GA is terminated. **Algorithm 5** is designed to calculate fitness.

The proposed fitness function returns a single fitness value for the entire population and individual fitness values for all members of a population. It takes population (*Po*), regression equations, user acceptable error (*er*), user-defined saved memory(*mr*) and preferable variables (α and β) as input. The role of preferable variables is to signify which user-defined parameter is critical. In the given method, α refers to accuracy loss while β refers to the percentage of memory to be saved. To understand the role of these parameters, let's assume an example, for instance, if accuracy loss is more critical than memory saved, then weights assigned α are greater than β ones, whereas the sum of both α and β is always 1. However, if both parameters are equally critical then both α and β will be assigned a value of 0.5.

Once the inputs to the algorithm are known, let's move into the working of the Fitness Evaluation algorithm as stated in **Algorithm 5**. Firstly, accuracy loss and memory to be saved are determined using regression equations obtained in the previous step. Next, MSEs is computed for accuracy loss (E) and memory to be saved (M) with user-defined parameters (*er* and *mr*). The final fitness value (*FP*) of populations is the sum of both MSEs (E and M).

Further, to estimate individual fitness for all individuals in a population, absolute error (*AE*) is used, which is defined in equations (10) and (11) of the algorithm. These equations are also used to find predicted accuracy loss and saved memory for each individual and saved in (F_ind) array with respect to

Algorithm 5 Fitness Evaluation

```
Input: Po, er, mr, Svmrbf, Im, a and p
Output: FP, F_ind
Procedure:
      t←0
      F_ind ← $\varphi$
      Ye = Im.predict(Po)
      Ym = SVMrbf.predict(Po)
          $E \leftarrow \frac{1}{n}\sum(Ye, er)^2 \times \alpha$ #MSE        (10)
          $M \leftarrow \frac{1}{n}\sum_n^0(Ym, mr)^2 \times \beta$ #MSE      (11)
      FP = E + M
      While (t< length (Po)) do:
          # Absolute error of each individual
          Er ← abs( Yet − Ep) × $\alpha$
          (14b)
          Mr ← abs(Ymt − Mp) × $\beta$
          (15 b)
          F_ind U (Er + Mr)
          t ← t + 1
      End
      Return FR error, FR me
```

their indexes in population. This individual fitness (F_ind) is used in the population modification process for updating the current population.

iii) POPULATION MODIFICATION

It is the process of selecting two or more individuals from the current population to generate new offspring and updating them into the current population using the following three methods.

1. Parent Selection: In this process, individuals with the least fitness values i.e. largest (AE) are selected and passed over to the next process. Firstly, individual population fitness (F_ind) is copied to another array (F_n) to avoid loss of data. In the second step, individuals with the highest (AE) are determined and their index position is saved to the selected parent list (Ps). In the next step, the individual fitness of the selected individual in the array (F_n) is set to the maximum integer value. The whole process keeps on repeating until the next parent is found.

2. Crossover: In the crossover, selected parents are used to generate one or more off-springs. To achieve this purpose, selected parents are binarized with the same bit length. In this work, one-point crossover is performed for generating new offspring. A random index within the bit length of binarized parents is produced. For generating the first offspring the first part of parent 1 in (Ps) is joined to the second part of parent 2 after that random index. The same procedure is repeated for generating the second offspring but with parent 2 as the first part and parent 1 as the second part. These newly generated offspring should be different from all individuals present

Algorithm 6 Optimized Quantization (OQ)

```
Input:Q-levek(bestpopulation), $\alpha$, $\beta$, Q-types, P-net
        (P-Laycrs), NP-Layers
Output:Compressed Model, NP-net, best Quantizer, best
        level, errorf(E'), memory(M'), FV'
Procedure:
  1. Apply each Q-type separately using required inputs
          $E_i$, $M_i$ = Q-type(model, Q-levels) (i ∈ Uni-Q
          or NonUm-Q or Aysm-Q)
  2. Compute fitness function across each quantizer
          $Er_i \leftarrow abs(E) \times \alpha$
          $Mr_i \rightarrow abs(M) \times \beta$
          $FV_i = Er_i + Mr_i$
  3. Determine best level and best quantizer by taking minimum of all
     fitness values FV' = min($FV_i$)
  4. Quantify weights of NP-layers to best level using best quantizer for
     different value of $\alpha$ and $\beta$ Take the best out of it.
  5. Return quantified model (NP-net)
  6. Return compressed model (NP-net)If (P-nel)
```

in the current population. If any of the offspring overlaps with the current population then that offspring is passed to the mutation process to generate new offspring. Once these offspring are finalized, they are converted to the decimal point and replaced with selected parents (Ps) using their index values ($indx$).

3. Mutation: The mutation is a process of creating genetic diversity in newly generated offspring. Mutation can only be performed when there are newly generated offspring existing in the current population. The mutation is carried out by randomly selecting and flipping one or more bits. For example, if the randomly selected bit is 0 then it is flipped to 1 and vice versa.

Once unique offspring are generated, they are replaced with the selected parents using the index values in the array ($indx$). The updated population is again processed with GA until it terminates on one of two conditions i.e. fitness value reaches the threshold or number of generations are completed. Once the algorithm terminates, the current updated population is returned as output along with the fitness values of the entire population and the fitness value of each individual in the population. At the end of the optimizer, we select the best population (*N-levels*) with the one having minimum fitness value (MSE) across each *Q-type* which is then fed into the next step of the Optimized Quantizer for quantization process.

2) OPTIMIZED QUANTIZER

Best population obtained in previous step is given to three quantizers i.e. uniform quantization, non-uniform quantization and asymmetric quantization with all required inputs. Error and memory are computed for each quantizer based upon the preference parameters α and β which basically measures the importance of two critical factors error and memory. α is associated with error while β is for memory. Fitness value is computed across each *Q-type* based on error

TABLE 1. Structure of CNN Networks used. It demonstrates the total layers included in each network with their respective parameters.

Network	CNN Structure	Parameters
LeNet-5	(CONV + AVGPOOL) x 2 + CONV + FC x 2	61,706
CIFAR-Quick	(CONV + MAXPOOL) x 3 + FC x 2	211,818
VGG16	(CONV x 2 + MAXPOOL) x 2 + (CONV x 3 + MAXPOOL) x 3 + FC x 2	40,416,074

TABLE 2. Datasets Statistics including size of each image, total number of images with the total training and test images. Also number of classes in each network.

Dataset	Image Size	No. of Training Images	No. of Test Images	Total Images	No. of classes
MNIST	28x28	60,000	10,000	70,000	10
CIFAR-10	32x32x3	50,000	10,000	60,000	10
ImageNet ILSVRC2012	224x224x3	9469	3925	13394	10

TABLE 3. Hyperparameters used while model training.

Hyperparameter	Value
Loss Function	Categorical_crossentropy
Optimizer	Adam
Learning Rate	0.01
No. of Epochs	100
Batch Size	64

and memory using said preference parameters as been given in **Algorithm 6**. The best level and best *Q-type* with minimum fitness value is determined and corresponding model is selected as the final compressed model.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we evaluate the effectiveness of our proposed system under comprehensive experimentation. We test our proposed framework for image-level object classification problem on Google Colab Pro. Our system is deployed on three widely used networks i.e. LeNet-5, CIFAR-Quick, and VGG-16 using MNIST, CIFAR-10, and ImageNet ILSVRC 2012 datasets respectively. We discuss an extensive performance analysis for Soft Filter Pruning (SFP) and two Quantization algorithms presented in the proposed network compression framework along with the results of Optimizer in this section. In addition, we also perform a comparative analysis of the proposed system with state-of-the-art network compression methods existing in the literature.

A. NETWORKS AND DATASETS

1) NETWORKS

LeNet-5 [34] is one of the simplest CNN models which is composed of 5 trainable layers including 3 convolutional layers and 2 fully connected (FC) layers. CIFAR-quick [35] is a fast-learning CNN model which also consists of 5 trainable layers having 3 convolutional layers and 2 FC layers. However, VGG-16 [36] is a deeper network having 13 convolutional layers and 2 FC layers making it total of 15 trainable layers. These networks are representative and the detailed structure of each model is illustrated in Table 1.

2) DATASETS

We conduct different experiments to test the validity of our proposed framework on MNIST [37], CIFAR-10 [38] and ImageNet ILSVRC2012 [39] datasets. We first evaluate the generalization performance of the system on small-sized samples using MNIST and CIFAR-10 datasets. MNIST is a collection of labeled data of handwritten 70,000 grayscale images with each image of size 28×28 having 10 classes. CIFAR-10 is having 60,000 colored images with 10 classes where each image size is $32 \times 32 \times 3$. However, ImageNet ILSVRC2012 is a large dataset that contains around 14 million images, each of size $224 \times 224 \times 3$ with 1000 classes. We have used a subset of this dataset, more details about each dataset used are demonstrated in Table 2.

B. EXPERIMENTAL SETUP

1) IMPLEMENTATION DETAILS

We conduct all experiments on Google Colab pro with GPU Tesla P100-PCIE-16GB in Python 3.5. In our experiments, training is performed under the specifications mentioned in Table 3. In addition, values of all hyperparameters have been kept the same for all models whereas Adam optimizer is used for LeNet-5 and CIFAR-quick while RMSprop is used for VGG-16 in our experiments.

2) EVALUATION PROTOCOLS

We assess resource-efficiency of the proposed system in terms of multiple evaluation protocols including *top-1 and top-5 classification accuracy*, *accuracy drop* and memory consumption in terms of *number of parameters*. In addition, we have also determined the amount of compression we have achieved in terms of *compression ratio* and the amount of memory saved after applying compression using the formulas given in equations 5 and 6. We take the original non-compressed trained model as *baseline model* and its accuracy is used as ground truth to carry out a comparative analysis.

C. EVALUATION OF SOFT FILTER PRUNING (SFP) ALGORITHM

We adopt training settings given in Table 3 and train our original models for 100 epochs. We compute classification accuracy, number of parameters, and memory required to store these parameters for the baseline model. This whole process is carried out before applying any compression method.

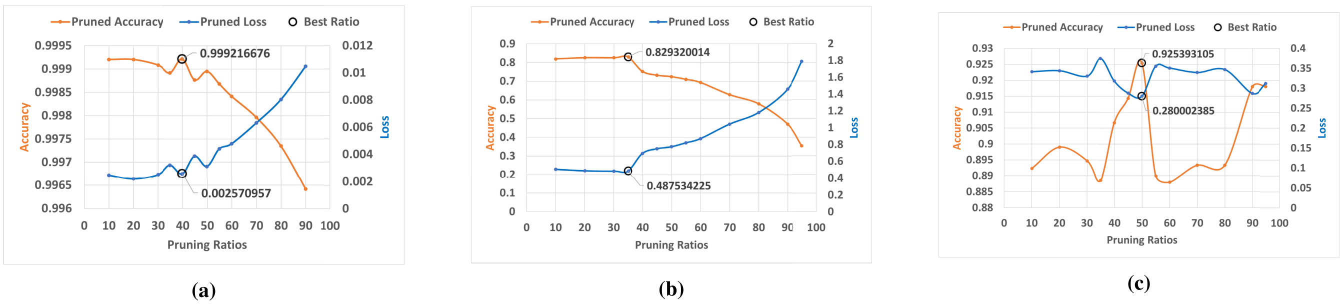


FIGURE 8. Selection of Best Pruning Ratio based on highest accuracy and minimum error. Learning Curves at different pruning ratios are given. (a) LeNet-5 performs best at *P=40%*, (b) CIFAR-quick performs best at *P=35%* and (c) VGG-16 performs best at *P=50%*.

TABLE 4. Memory Optimization based Comparison of Pruned Model with Baseline Model for LeNet-5, CIFAR-quick and VGG-16. Results of pruned model outperforms baseline model and saved significant amount of memory by downsizing each network using SFP.

		Accuracy (%)			No. of Parameters			Memory		
Network	**Dataset**	**Baseline Model**	**Pruned Model**	**Drop**	**Baseline Model**	**Pruned Model**	**Drop** (%)	**Baseline Model**	**Pruned Model**	**Memory Saved** (%)
LeNet-5 (at P=40%)	**MNIST**	99.88	99.92	-0.04	61706	41323	20383	1974592	1322336	33.03
CIFAR-quick (at P=35%)	**CIFAR-10**	86.40	82.93	3.47	211818	182871	28947	6778176	5851872	13.67
VGG-16 (at P=50%)	**ImageNet ILSVRC2012**	84.10	92.53	-8.43	27828042	14047914	13780128	890497344	449533248	49.5

With the proposed network compression scheme, we first compress network using Soft Filter Pruning (SFP) and evaluate system for different values of pruning ratios (*P*%) including 10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95. The model's accuracy and error have been determined across each pruning ratio and their learning curves are shown in Figure 8. LeNet-5 has performed best at pruning ratio = 40%. Similarly, CIFAR-quick and VGG-16 have performed best at pruning ratios of 35% and 50% respectively. Hence, we carry out the rest of the experimentation for these selected pruning ratios for all networks.

Table 4 shows the performance comparison of the pruned model with the baseline model and it is found that up to 33% (20383 out of 61706) number of parameters have been dropped with a slight increase in accuracy from 99.92% to 99.88% for LeNet-5. This saves up to 33% of the model's memory. Similarly, 13.67% of memory is saved with a slight decrease in accuracy for the CIFAR-quick model. However, SFP performs best for the VGG-16 model and achieved a significant increase in accuracy from 84.10% to 92.53% by saving almost half of the model's memory which is around 49.5%.

D. EVALUATION OF INCREMENTAL QUANTIZATION

With the proposed compression scheme, next, we divide network layers into NP-Layers and P-Layers categories. We empirically choose the 60:40 ratio for layers division. In LeNet-5, we have a total of 5 trainable layers (CONV and FC), out of which all 3 CONV layer comes in the NP-layers category while 2 FC layers are P-layers. Similarly, in CIFAR-quick using the said ratio, 3 CONV layers are grouped into NP-layers while the rest of the network layers (i.e. 2 FC layers) come under P-layers. However, this division becomes more interesting in larger networks as in VGG-16, where the first 9 CONV layers are grouped into the NP-layers category while the remaining 6 layers including both CONV and FC layers are grouped into P-layers.

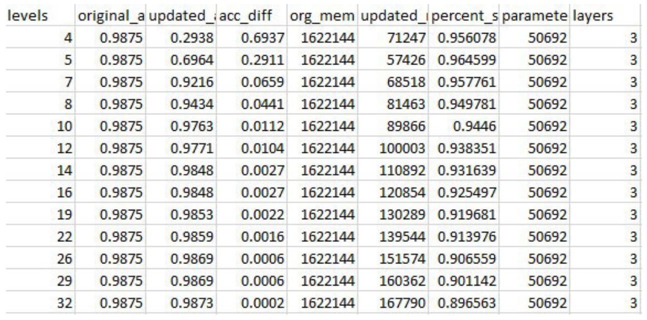
To further compress the model after pruning, we apply the INQ algorithm on P-layers of the network. As it is mentioned in Figure 6, INQ includes weight partitioning, group-wise weights quantization and re-training. For weight partitioning, we set partitioning ratio *r* as 70:30 to divide weights into two disjoint groups A1 and A2. Using the said ratio, weights are quantified in the accumulated portion of 30% −> 51% −>66%−>76%−>83%−>88%−>92%−> 94%−>96%−>97%−>98%−>99%−>100%.

We analyze the effectiveness of INQ by performing many experiments. In Table 5, we perform a memory optimization-based comparison of the compressed model with the baseline model. We perform two different experiments, wherein the first case INQ is applied on P-Layers of the original model while in the second case, INQ is applied on P-Layers of Pruned Model that we have obtained in the previous section. It is clearly shown in Table 5 that the results of the compressed model (P+INQ) outperform the baseline model and compressed model (using INQ only). Moreover, it has reduced the bit-width of weight parameters from 32-bits to lower bit-width representation. The notation of [6,5] in LeNet-5 refers to two P-layers with the first layer having weights quantified to 6-bits while weights of the second P-layer are quantified into 5-bits. Similarly, in CIFAR-quick we have quantified two P-layers into 6 and 5 bits respectively. However, in case-1

TABLE 5. Memory optimization based comparison of compressed model with baseline model for LeNet-5, CIFAR-quick and VGG-16. Case 1: compressed model (INQ only) and Case 2: compressed model (P+INQ) with original model as baseline model in both cases. Bit-width represents number of bits used to represent weight parameters. Column 1 of bit-width refers to bits for baseline model, its Column 2 refers to bits for NP-Layers of compressed model and Column 3 refers to bits for P-Layers of compressed model.

Cases	Network	Dataset	No. of param-eters		Bit-width			Memory		M_{saved} (%)	CR
			M_{base}	M_{comp}	M_{base}	M_{base} (NP-Layers)	M_{comp} (P-Layers)	M_{base}	M_{comp}		
Case 1	LeNet-5	MINIST	61706	61706	32	32	[6,5]	1974592	1687378	14.54	1.17x
	CIFAR-quick	CIFAR-10	211818	211818	32	32	[6,5]	6778176	3332146	50.84	2.03x
	VGG-16	ImageNet ILSVRC2012	27828040	27828040	32	32	[6,6,6,7,6,5]	890497280	378325540	57.51	2.35x
Case 2	LeNet-5	MINIST	61706	41323	32	32	[6,5]	1974592	1035122	47.57	1.97x
	CIFAR-quick	CIFAR-10	211818	182871	32	32	[6,5]	6778176	2405842	64.50	2.81x
	VGG-16	ImageNet ILSVRC2012	27828040	14047914	32	32	[6,6,6,6,6,5]	890497280	183540786	79.38	4.85x

of VGG-16 with six P-Layers, out of which the first three layers are quantified into 6-bits and the next three layers are quantified into 7-bits, 6-bits, and 5-bits respectively. While in case-2 of VGG-16, all layers are quantified into 6-bits except the last layer which is quantified into 5-bits. Results have shown that the compressed model (P+INQ) has saved up to the highest memory of 79% in VGG-16, while 64.5% and 47.5% in CIFAR-quick and LeNet-5 respectively and has achieved higher compression ratio than other compressed models (using INQ only).



levels	original_a	updated_	acc_diff	org_mem	updated_	percent_s	paramete	layers
4	0.9875	0.2938	0.6937	1622144	71247	0.956078	50692	3
5	0.9875	0.6964	0.2911	1622144	57426	0.964599	50692	3
7	0.9875	0.9216	0.0659	1622144	68518	0.957761	50692	3
8	0.9875	0.9434	0.0441	1622144	81463	0.949781	50692	3
10	0.9875	0.9763	0.0112	1622144	89866	0.9446	50692	3
12	0.9875	0.9771	0.0104	1622144	100003	0.938351	50692	3
14	0.9875	0.9848	0.0027	1622144	110892	0.931639	50692	3
16	0.9875	0.9848	0.0027	1622144	120854	0.925497	50692	3
19	0.9875	0.9853	0.0022	1622144	130289	0.919681	50692	3
22	0.9875	0.9859	0.0016	1622144	139544	0.913976	50692	3
26	0.9875	0.9869	0.0006	1622144	151574	0.906559	50692	3
29	0.9875	0.9869	0.0006	1622144	160362	0.901142	50692	3
32	0.9875	0.9873	0.0002	1622144	167790	0.896563	50692	3

FIGURE 9. Snippet of Sample datafile(.csv) obtained from data compression for LeNet-5 using uniform quantization. Col-1 indicates quantization levels (up to 1024), Col-2 is original accuracy, Col-3 & Col-4 represent accuracy after quantization and accuracy difference respectively. Col-5 is original memory required to store original parameters given in Col-8 while Col-6 is updated memory obtained after data compression. Col-9 shows percent of memory saved by applying data compression. Last column represents total number of NP-layers used to carry out the whole process.

E. EVALUATION OF OPTIMIZED QUANTIZATION

In this section, we analyze the results of Optimizer and Optimized Quantizer for NP-layers and study the impact of Optimized Quantization on the overall performance of the system. With the proposed network compression scheme, the next step is to perform Optimized Quantization using optimal *N-levels* obtained from Optimizer. In the process of obtaining *N-levels*, we first get datafiles by applying data compression on weight files of NP-layers. In data compression, we apply three quantizers i.e. uniform, non-uniform and asymmetric up to empirically selected 2^{10}=1024 quantization levels. Accuracy loss and original memory are determined through the quantization process across each quantizer. However, new memory is obtained by applying a canonical Huffman encoder on quantified weights to further reduce the memory. All these parameters obtained from both quantization and Huffman encoder are saved into data files. We have got a total of 3 data files for all three quantizers and perform this process for LeNet-5, CIFAR-quick, and VGG-16 networks. This results in a total of 9 datafiles, sample datafile is given in Figure 9.

Using the datafiles we have obtained previously, we applied many regressors to find regression equations for accuracy loss and saved memory. To achieve this goal, we test some different regressors including Linear Regression, Lasso Regression, Bayesian Ridge Regression, TheilSen Regression, SVM Linear, SVM Polynomial, SVM RBF, Sigmoid Regression, Huber Regression, and RANSAC Regression for accuracy loss and memory separately using **Algorithm 5**.

To determine the best regressor, MSEs obtained for regressors are averaged and a regressor with a minimum averaged MSE is selected. Bar graphs obtained for both accuracy loss and saved memory are given in Figure 10. It is found that the least average MSE is obtained for linear regression and it is selected for both accuracy loss and memory saved as shown with the yellow bar in Figure 10. In the next step, we input the regression equations for memory and accuracy loss obtained in the previous step from the best regressor to GA to get optimal quantization levels (*N-levels*). We test the performance of the GA algorithm for three different cases as given in Table 6 for all three quantizers (*Q-types*). In each case, a population array including different quantization levels of length 10 along with its fitness value (MSE) is determined across each quantizer. It is observed that for each quantizer, the population array from the best case is selected based on the least fitness value. For LeNet-5(NP-layers), the best case found is Case 1 for all quantizers, similarly for CIFAR-quick

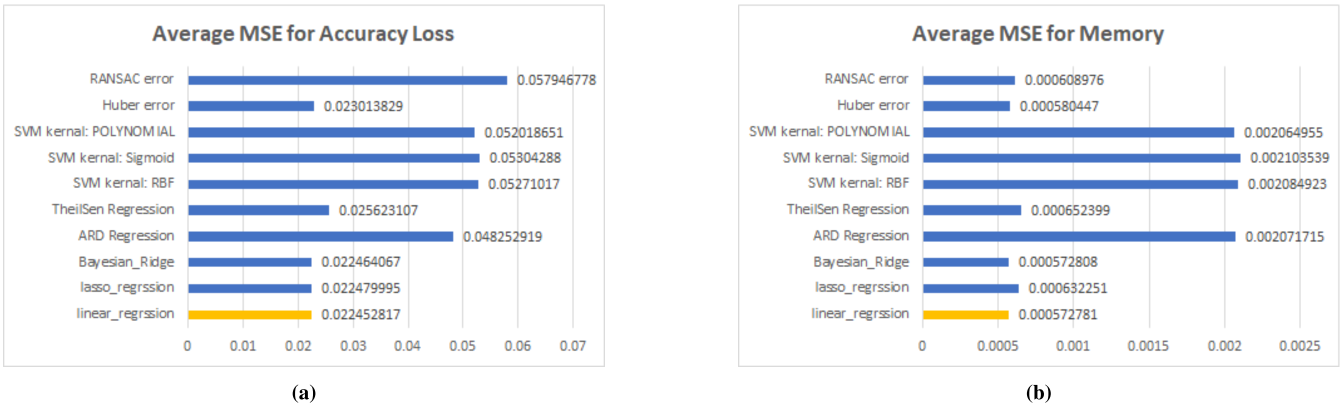


FIGURE 10. **Results of Regression Selection. (a) Average MSE of accuracy loss for all regressors. (b) Average MSE of saved memory for all regressors. Yellow bar in each graph indicates the best regressor with least average MSE.**

TABLE 6. **Parameters setting of three different cases for applying Genetic Algorithm.**

Parameters	Case 1	Case 2	Case 3
Iterations	100	200	100
Size	10	10	10
Crossovers	2	2	3
User Error	0.01	0.1	0.05
User Memory	0.4	0.35	0.5
α	0.6	0.6	0.5
β	0.4	0.4	0.5
Threshold	0.05	0.1	0.07

(NP-layers), the minimum fitness value is obtained for case-1 across all quantizers. However, for VGG-16 (NP-layers) the best case selected is case 2 with minimum fitness value. Best cases with respective populations and fitness values are highlighted in Table 7, 8 and 9.

Using the parameters setting of the best cases we have obtained previously, we analyze the impact of Optimized Quantization on the overall performance of the system. First, we use best population which contains *(N-levels)* and apply all *Q-types* to determine best quantization level (*Q-level*) for NP-layers of the network. To achieve this goal, we have chosen three randomly selected values for preference variables (α) and (β) where α measures the importance of accuracy loss and β is for saved memory. Depending upon user requirements, we set different values for said variables to determine the best level out of the 10 *N-levels* resulting from the Optimizer. We have used (α,β) = (0.5,0.5),(0,1),(1,0) and compute fitness value (MSE) for each quantization level against all *Q-type* using the given expression in 12.

$$fitnessvalue = \alpha \times accuracy_{loss} + \beta \times memory_{saved} \quad (12)$$

We have obtained 10 fitness values referring to 10 quantization levels for each population and the best *Q-level* and the best *Q-type* are selected based on the least fitness value. We repeat this for all values of α and β. Table 10 shows the results of the best level and best quantizer we have found for different values of α and β. Further, results have shown, that for LeNet-5, when we consider accuracy loss and saved memory equally critical (α=β=0.5) then it saves up to 94% memory with negligible accuracy drop. Further, when we consider memory more critical than accuracy loss (α=0, β=1), then memory saved is even more but the accuracy drop is beyond acceptance. And finally, when we consider accuracy loss more critical than saved memory (α=1, β=0) then accuracy remains the same while amount of saved memory drops significantly as given in Table 10. Similarly, the impact of different values of α and β on overall accuracy drop and saved memory is also given in Table 10 for CIFAR-quick and VGG-16 respectively. It is found that in the case of VGG-16, accuracy drops drastically to a non-acceptable level for α=0, β=1 with a significant drop in memory while for the other two cases of α and β values, a good amount of memory is saved with negligible accuracy drop. Further, the best *Q-type* and best *Q-level* are also shown for each case with minimum fitness value.

We further analyze the memory optimization achieved by applying the proposed system and compare the system's resource efficiency with the baseline model. Using the results from previous tables, we also determine the performance measures for the overall proposed system using all techniques as been proposed including pruning, INQ for P-layers, and OQ for NP-layers at different values of α and β. We analyze the memory optimization achieved by applying the proposed system and a comparison is made with the baseline model based on the parameter's bit-width, memory, and compression ratio. Remarkable improvement in results is shown using a proposed system that has reduced bit-width of weight parameters from 32-bits to low bits for both NP-layers and P-layers as given in Table 11. Considering the accuracy loss and memory equally critical i.e. α=0.5 and β=0.5, it is found that a significant amount of memory has been saved after applying the proposed compression method, which is 91% with a compression ratio of 10.6x for LeNet-5. Similarly, 6% compression has been achieved by saving memory up to 83% for CIFAR-quick, and compression of 5x has been achieved for VGG-16 with memory saved up to 80%.

TABLE 7. End Results of Optimizer for LeNet-5 indicating final populations obtained across each case by applying Genetic Algorithm for all quantizers. Each population contains 10 individuals representing 10 different quantization levels. Best population(*N-levels)* is highlighted across each quantizer based on least fitness value (MSE).

	Case1			Case 2			Case 3		
Q-type	Iteration	Fitness Value (MSE)	Final Population	Iteration	Fitness Value (MSE)	Final Population	Iteration	Fitness Value (MSE)	Final Population
Uniform	**100**	**0.078125**	**99, 55, 95, 2253, 18, 9, 134, 5, 2333, 131**	200	0.098051	99, 55, 95, 1281, 18, 9, 134, 769, 137, 131	100	0.091673	99, 55, 95, 129, 18, 9, 134, 2457, 2193, 131
Non-Uniform	**100**	**0.080861**	**54, 20, 46, 2581, 19, 21, 13, 46, 2709, 86**	200	0.099891	117, 20, 401, 109, 258, 118, 13, 174, 1297, 142	100	0.090608	117, 20, 4083, 109, 258, 118, 13, 174, 2995, 142
Asymmetric	**100**	**0.074117**	**37, 144, 262, 89, 246, 150, 75, 193, 3276, 3276**	200	0.097472	37, 144, 262, 89, 246, 150, 75, 193, 786, 786	100	0.091904	37, 144, 6, 89, 3494, 150, 75, 193, 2478, 24

TABLE 8. End Results of Optimizer for CIFAR-quick indicating final populations obtained across each case by applying Genetic Algorithm for all quantizers. Each population contains 10 individuals representing 10 different quantization levels. Best population(*N-levels)* is highlighted across each quantizer based on least fitness value (MSE).

	Case1			Case 2			Case 3		
Q-type	Iteration	Fitness Value (MSE)	Final Population	Iteration	Fitness Value (MSE)	Final Population	Iteration	Fitness Value (MSE)	Final Population
Uniform	**100**	**0.090505**	**3973, 152, 109, 134, 138, 90, 127, 118, 161, 3861**	200	0.098356	183, 152, 213, 134, 138, 3547, 127, 2459, 161, 177	100	0.101982	183,152, 213, 134, 138, 1696, 127, 1696, 161, 177
Non-Uniform	**100**	**0.086918**	**228, 134, 3650, 108, 192, 118, 74, 264, 3652, 253**	200	0.098628	228, 134, 2336, 108, 192, 118, 74, 3328, 152, 253	100	0.100753	228, 134, 3797, 108, 192, 118, 74, 264, 3831, 253
Asymmetric	**100**	**0.091058**	**96, 141, 63, 66, 168, 2979, 72, 168, 151, 2867**	200	0.099545	96, 141, 63, 66, 168, 197, 72, 168, 2999, 2999	100	0.104994	96, 141, 63, 66, 168, 197, 72, 168, 3832, 4049

Also, we have performed a comparative analysis of all compression models used in the proposed system based on top-1 accuracy, top-5 accuracy, and compression ratio in Table 12. This is done to investigate the contribution of each model to the overall performance of the proposed system. It is evident from results that the highest compression of 11x, 6x, and 5x are achieved for LeNet-5, CIFAR-quick, and VGG-16 respectively by using (P+INQ+OQ), which is larger than using any of the compression methods separately. It is also found that a negligible accuracy drop has occurred for LeNet-5 whereas a little rise in accuracy is observed for VGG-16 while the accuracy drop for CIFAR-quick is slightly higher than the other two models but it is in an acceptable range. Figure 11 also demonstrates these results in the form of bar graphs, where Figure 11a is comparison of Top-1 accuracy for all compression models, Figure 11b shows comparison of Top-5 accuracy for all compression models while Figure 11c refers to comparison of compression ratios achieved for all compression models. These results also validate the fact that highest compression ratio has achieved with a compression model (P+INQ+OQ) without losing system's accuracy. This has proved the effectiveness of the proposed system in downsizing the resource requirement of the system for resource-restricted embedded systems with a negligible drop in accuracy.

At the end, we compare our proposed system with state-of-the-art network compression methods, illustrated in Table 13. In order to carry out the extensive comparison,

TABLE 9. End Results of Optimizer for VGG-16 indicating final populations obtained across each case by applying Genetic Algorithm for all quantizers. Each population contains 10 individuals representing 10 different quantization levels. Best population(*N-levels)* is highlighted across each quantizer based on least fitness value (MSE).

	Case1			**Case 2**			**Case 3**		
Q-type	**Iteration**	**Fitness Value (MSE)**	**Final Population**	**Iteration**	**Fitness Value (MSE)**	**Final Population**	**Iteration**	**Fitness Value (MSE)**	**Final Population**
Uniform	100	0.384994	3, 3348, 3348, 59, 8, 50, 172, 142, 58, 11	**200**	**0.338636**	**3, 52, 4062, 59, 8, 50, 3804, 142, 58, 11**	100	0.340685	3, 60, 3703, 59, 8, 50, 3695, 142, 58, 11
Non-Uniform	100	0.384347	58, 50,27, 130, 102, 3718, 17, 3654, 44,133	**200**	**0.337673**	**58, 50, 27, 34, 102, 3885, 17, 43, 44,3389**	100	0.339611	58, 50, 27, 3833, 102, 3800, 17, 162, 44, 133
Asymmetric	100	0.385278	78, 222, 2601, 30, 30, 34, 2, 134, 45, 1577	**200**	**0.33706**	**78, 222, 267, 3550, 3423, 34, 2, 134, 45,243**	100	0.339039	78, 222, 267, 3018, 4074, 34, 2, 134, 45, 243

TABLE 10. Results of Optimized Quantization (OQ), demonstrating best *Q-level* and best *Q-type* for LeNet-5, CIFAR-quick and VGG-16 networks at three different values of α and β. Fitness value is the MSE computed using α and β, whereas Q-level and Q-type specify the best quantizer and best quantization level selected based on least fitness value. Original accuracy is the accuracy obtained by training original model while new accuracy refers to accuracy after applying OQ. Similarly, original memory is the memory required to store parameters of NP-layers in original model whereas new memory is the memory required by NP-layers parameters after applying OQ.

Network	α	β	**Best Q-type**	**Fitness Value**	**Best Levels**	**Original Accuracy**	**New Accuracy**	**Original Memory**	**New Memory**
LeNet-5 MNIST	0.5	0.5	Non-Uniform	0.033124	13	0.9875	0.9825	1622144	99354
	0	1	Uniform	0.035401	5	0.9875	0.6964	1622144	57426
	1	0	Asymmetric	0	2581	0.9875	0.9875	1622144	470883
CIFAR-quick CIFAR-10	0.5	0.5	Non-Uniform	0.054636	72	0.7243	0.7159	1612192	162626
	0	1	Uniform	0.096737	63	0.7243	0.1278	1612192	155959
	1	0	Asymmetric	0.0001	3650	0.7243	0.7244	1612192	371201
VGG-16 ImageNet ILSVRC2012	0.5	0.5	Non-Uniform	0.048668	3550	0.888917	0.889172	122164224	11859955
	0	1	Uniform	0.019323	2	0.888917	0.101656	122164224	2360632
	1	0	Asymmetric	0.000255	3550	0.888917	0.889172	122164224	11859955

TABLE 11. Demonstration of memory optimization achieved by applying proposed system (P+INQ+OQ) for LeNet-5, CIFAR-quick and VGG-16 networks. Parameters bit-width, memory and compression ratio(CR) are represented at different values of α and β. M_{base} refers to baseline (original) model while M_{comp} represents proposed compressed model.

Network			**Bit-width**			**Memory**					
	α	β	M_{base}	M_{comp} (Np-layers)	M_{comp} (P-layers)	M_{base}	M_{comp} (Np-layers)	M_{comp} (P-layers)	Total M_{comp}	**Memory Saved (%)**	**CR**
LeNet-5 MNIST	0.5	0.5	32	4	[6,5]	1974592	121236	65234	186470	90.5	10.6x
	0	1		3			90927		156161	92	12.6x
	1	0		12			363708		428942	78.27	4.6x
CIFAR-Quick CIFAR-10	0.5	0.5	32	7	[6,5]	6778176	352667	793650	1146317	83.1	6x
	0	1		6			302286		1095936	83.83	6.1x
	1	0		12			604572		1398222	79.37	4.84x
VGG-16 ImageNet ILSVRC 2012	0.5	0.5	32	12	[6,6,6,6,6,5]	890497280	45811584	133997092	179808676	79.8	4.95x
	0	1		1			3817632		137814724	84.5	6.46x
	1	0		12			45811584		179808676	79.8	4.95x

along with LeNet-5 and VGG-16 CNN models, we have also tested the performance of our system for AlexNet [9] and ResNet-50 [2] models using ImageNet ILSVRC2012 dataset. Along with the subset of ImageNet mentioned in Table 2, we have also analyzed system's performance for another subset of ImageNet ILSVRC2012 consisting of 1lac images

TABLE 12. Performance comparison of all compression models presented in paper based on Top-1 Accuracy, Top-5 Accuracy and Compression Ratio.

Compression Model	Network	Top-1 Accuracy		Top-5 Accuracy		CR
		Baseline Model	Compressed Model	Baseline Model	Compressed Model	
Model-1 (Pruning only)	**LeNet** (MNIST)	99.88	99.92	100	100	1.49x
	CIFAR-quick (CIFAR-10)	86.40	82.93	99.75	99.44	1.15x
	VGG-16 (ImageNet ILSVRC2012)	84.10	92.53	98.7	99.76	1.98x
Model-2 (Pruning + INQ)	**LeNet** (MNIST)	99.88	98.82	100	99.97	1.97x
	CIFAR-quick (CIFAR-10)	86.40	82.60	99.75	99.39	2.81x
	VGG-16 (ImageNet ILSVRC2012)	84.10	90.43	98.7	99.13	4.85x
Model-3 (Pruning + INQ + OQ) (α=0.5, β=0.5)	**LeNet** (MNIST)	99.88	98.25	100	99.94	10.6x
	CIFAR-quick (CIFAR-10)	86.40	71.60	99.75	97.05	6x
	VGG-16 (ImageNet ILSVRC2012)	84.10	88.92	98.7	99.23	4.95x

TABLE 13. Comparison of Proposed System with state-of-the-art Network Compression Methods. Increase in Top-1 and Top-5 accuracy with significant decrease in bit-width of weight parameters as compared to existing methods. In [12/6, 5] notation of bit-width for our method, 12 refers to reduced bit-width for NP-layers of the network while [6, 5] refers to 6-bits and 5-bits representation of weight parameters of different layers in P-layers.

Network	Compression Method	Weights Bit-width	Top-1 Accuracy	Top-5 Accuracy	CR
LeNet-5 (MNIST)	[40]	8	93.9	N/A	N/A
	[12]	N/A	99.45	N/A	N/A
	[40]	32	98.15	N/A	2.5x
	Ours	**[12/6,5]**	**98.25**	**99.94**	**10.6x**
VGG-16 (ImageNet ILSVRC2012)	[2]	5	70.82	90.3	N/A
	[41]	14	N/A	88.4	N/A
	[15]	N/A	70.32	89.42	49x
	[42]	32	72.5	90.9	N/A
	Ours (1)	**[12/6,5]**	**88.92**	**99.23**	**∼5x**
	Ours (2)	**[12/6,5]**	**87.52**	**97.45**	**∼5x**
AlexNet (ImageNet ILSVRC2012)	[9]	N/A	57.22	80.3	35x
	[43]	N/A	57.23	80.33	9x
	[24]	5	53.93	77.81	N/A
	[44]	N/A	55.3	78.5	32x
	Ours (1)	**[12/6,5]**	**56.91**	**80.01**	**∼8.5x**
	Ours (2)	**[12/6,5]**	**56.10**	**79.45**	**∼8.5x**
ResNet-50 (ImageNet ILSVRC2012)	[2]	5	74.81	92.45	N/A
	[45]	N/A	75.34	92.43	N/A
	[29]	3	74.7	92.1	N/A
	[44]	N/A	76.0	93.0	11.4x
	[46]	N/A	74.66	92.1	N/A
	Ours (1)	**[12/6,5]**	**75.63**	**92.47**	**∼7.5x**
	Ours (2)	**[12/6,5]**	**74.95**	**91.34**	**∼7.5x**

(such that 60,000 images for training and 40,000 images for testing) on VGG-16, AlexNet and ResNet-50 models. Therefore, two results of the proposed method are mentioned in Table 13 for these models, the former one is for ImageNet subset mentioned in Table 2, while the latter results are for the subset of ImageNet having 1lac images. It is found

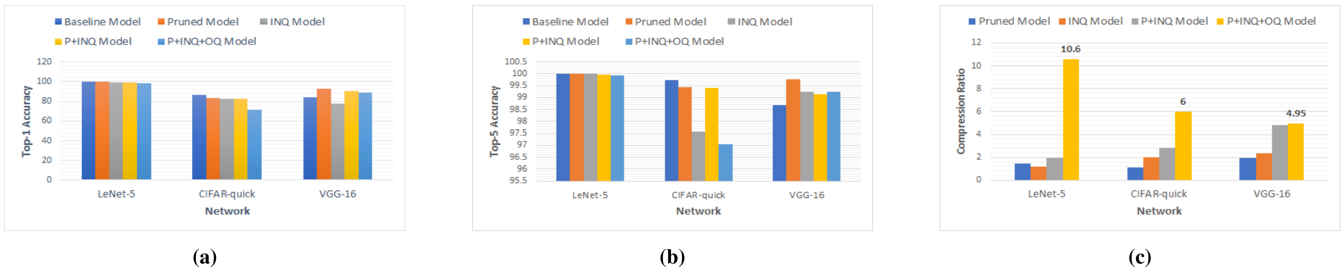


FIGURE 11. Performance Comparison of all Compressed Models used in this paper with the Baseline Model on the basis of (a) Top-1 Accuracyl (b) Top-5 Accuracy and (c) Compression Ratio for LeNet-5, CIFAR-quick and VGG-16 networks.

that consistent results have been achieved for both subsets. Also, it is noted that authors have used different types of methods to perform network compression, some have used pruning to downsize the network in terms of parameter count, others have applied quantization-based methods to reduce the bit-width of parameters while few authors have proposed methods based on both pruning and quantization to carry out network compression to reduce the overall memory requirement of the system by decreasing parameters count and bit-width both. We consider all types of methods in our comparison to determine the impact of compression on the performance of the system in terms of Top-1 accuracy, Top-5 accuracy and compression ratio. Table 13 shows the superior performance of our method in terms of resource efficiency with a significant reduction in bit-width of parameters by achieving a good compression rate and having top-1 accuracy and top-5 accuracy higher than other methods. Since the proposed method is a combination of pruning and quantization both, therefore, it is capable of achieving high compression while preserving the system's accuracy. This leads to an increase in the applicability of CNN for resource-restricted edge devices.

V. CONCLUSION

In this paper, we have proposed a memory-efficient based network compression framework for resource-restricted edge devices. This approach attempts to compress the network for reducing the overall memory requirement of the model while not losing the system's high performance. The proposed framework is a combination of both pruning and quantization, which works collaboratively. It first performs filter pruning to decrease the parameter count by setting the least significant parameter to zero. In the next step, the layers of the networks are divided into two categories as NP-layers (front layers) and P-layers (latter layers). Further, Incremental Network Quantization is applied to P-layers of the network due to sparse distribution of weights while Optimized Quantization is proposed for NP-layers of the network. In addition, a novel structure of an optimizer is presented which involves basic quantization, canonical Huffman coding, regression, and genetic algorithm to determine optimal quantization levels required to carry out quantization in OQ. We have carried out extensive experimentation to assess our system's effectiveness for image-level object classification on LeNet-5, CIFAR-quick, and VGG-16 using MNIST, CIFAR-10, and ImageNet datasets respectively. Results have shown that proposed network compression (P+INQ+OQ) has achieved a high compression rate of 11x for LeNet-5(MNIST) by saving up to 91% of model memory. It has reduced parameters bit-width to lower bits and parameters count to almost half with negligible accuracy drop. This effectively reduces the memory requirement of the system and enables CNN deployment in resource-restricted environments by offering good memory-accuracy trade-offs.

REFERENCES

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2019.

[2] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*.

[3] J. Rong, X. Yu, M. Zhang, and L. Ou, "Soft Taylor pruning for accelerating deep convolutional neural networks," in *Proc. 46th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2020, pp. 5343–5349.

[4] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–15.

[5] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey," 2020, *arXiv:2005.04275*.

[6] W. B. Zhao, Y. Li, and L. Shang, "Fuzzy pruning for compression of convolutional neural networks," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jun. 2019, pp. 1–5.

[7] Y. Wen and D. Gregg, "Exploiting weight redundancy in CNNs: Beyond pruning and quantization," 2020, *arXiv:2006.11967*.

[8] X. Han, L. Xue, Y. Xu, and K. Huang, "A parasitic mechanism-based filter pruning method for deep convolutional neural networks," in *Proc. IEEE Int. Conf. Inf. Technol., Big Data Artif. Intell. (ICIBA)*, Nov. 2020, pp. 45–48.

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

[10] R. Pilipovic, P. Bulic, and V. Risojevic, "Compression of convolutional neural networks: A short survey," in *Proc. 17th Int. Symp. INFOTEH-JAHORINA (INFOTEH)*, Mar. 2018, pp. 1–6.

[11] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon: Extensions and performance comparisons," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 6, 1993, pp. 1–13.

[12] S. Sarkar, M. Agarwalla, S. Agarwal, and M. P. Sarma, "An incremental pruning strategy for fast training of CNN models," in *Proc. Int. Conf. Comput. Perform. Eval. (ComPE)*, Jul. 2020, pp. 371–375.

[13] T.-H. Chen, C.-H. Huang, Y.-S. Chu, and B.-C. Cheng, "Towards efficient neural network on edge devices via statistical weight pruning," in *Proc. IEEE 9th Global Conf. Consum. Electron. (GCCE)*, Oct. 2020, pp. 192–193.

[14] S. Moon, Y. Byun, J. Park, S. Lee, and Y. Lee, "Memory-reduced network stacking for edge-level CNN architecture with structured weight pruning," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 4, pp. 735–746, Dec. 2019.

[15] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, p. 8.

[16] M. Mousa-Pasandi, M. Hajabdollahi, N. Karimi, S. Samavi, and S. Shirani, "Convolutional neural network pruning using filter attenuation," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 2905–2909.

[17] N. J. Kim and H. Kim, "Mask-soft filter pruning for lightweight CNN inference," in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2020, pp. 316–317.

[18] R. Liu, J. Cao, P. Li, W. Sun, Y. Zhang, and Y. Wang, "NFP: A no fine-tuning pruning approach for convolutional neural network compression," in *Proc. 3rd Int. Conf. Artif. Intell. Big Data (ICAIBD)*, May 2020, pp. 74–77.

[19] M. K. Lee, S. Lee, S. H. Lee, and B. C. Song, "Channel pruning via gradient of mutual information for light-weight convolutional neural networks," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 1751–1755.

[20] F. Nikolaos, I. Theodorakopoulos, V. Pothos, and E. Vassalos, "Dynamic pruning of CNN networks," in *Proc. 10th Int. Conf. Inf., Intell., Syst. Appl. (IISA)*, Jul. 2019, pp. 1–5.

[21] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1389–1397.

[22] A. Jordao, M. Lie, and W. R. Schwartz, "Discriminative layer pruning for convolutional neural networks," *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 4, pp. 828–837, May 2020.

[23] E. Kalali and R. van Leuken, "A power-efficient parameter quantization technique for CNN accelerators," in *Proc. 24th Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2021, pp. 18–23.

[24] S. Seo and J. Kim, "Hybrid approach for efficient quantization of weights in convolutional neural networks," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Jan. 2018, pp. 638–641.

[25] S. Kim and H. Kim, "Mixture of deterministic and stochastic quantization schemes for lightweight CNN," in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2020, pp. 314–315.

[26] S. Gheorghe and M. Ivanovici, "Model-based weight quantization for convolutional neural network compression," in *Proc. 16th Int. Conf. Eng. Modern Electr. Syst. (EMES)*, Jun. 2021, pp. 1–4.

[27] Z. Bao, K. Zhan, W. Zhang, and J. Guo, "LSFQ: A low precision full integer quantization for high-performance FPGA-based CNN acceleration," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS)*, Apr. 2021, pp. 1–6.

[28] K. Nakata, D. Miyashita, J. Deguchi, and R. Fujimoto, "Adaptive quantization method for CNN with computational-complexity-aware regularization," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[29] S. Cho and S. Yoo, "Per-channel quantization level allocation for quantizing convolutional neural networks," in *Proc. IEEE Int. Conf. Consum. Electron.-Asia (ICCE-Asia)*, Nov. 2020, pp. 1–3.

[30] C. Liu and H. Lu, "A highly efficient training-aware convolutional neural network compression paradigm," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jul. 2020, pp. 1–6.

[31] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," 2018, *arXiv:1808.06866*.

[32] R. Patel, V. Kumar, V. Tyagi, and V. Asthana, "A fast and improved image compression technique using Huffman coding," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Mar. 2016, pp. 2283–2286.

[33] J. H. Holland, "Genetic algorithms," *Sci. Amer.*, vol. 267, no. 1, pp. 66–73, 1992.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[35] Y. Zhou, S. Song, and N.-M. Cheung, "On classification of distorted images with deep convolutional neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 1213–1217.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[37] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[38] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on CIFAR-10," *Unpublished Manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[39] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[40] J. Lee and S. Lee, "Robust CNN compression framework for security-sensitive embedded systems," *Appl. Sci.*, vol. 11, no. 3, p. 1093, Jan. 2021.

[41] J. Wang, J. Lin, and Z. Wang, "Efficient hardware architectures for deep convolutional neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Nov. 2017.

[42] J. Du, X. Zhu, M. Shen, Y. Du, Y. Lu, N. Xiao, and X. Liao, "Model parallelism optimization for distributed inference via decoupled CNN structure," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1665–1676, Jul. 2021.

[43] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–12.

[44] S. I. Young, W. Zhe, D. Taubman, and B. Girod, "Transform quantization for CNN compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5700–5714, Sep. 2022.

[45] X. Ruan, Y. Liu, C. Yuan, B. Li, W. Hu, Y. Li, and S. Maybank, "EDP: An efficient decomposition and pruning scheme for convolutional neural network compression," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 10, pp. 4499–4513, Oct. 2021.

[46] A. Alqahtani, X. Xie, M. W. Jones, and E. Essa, "Pruning CNN filters via quantifying the importance of deep visual representations," *Comput. Vis. Image Understand.*, vols. 208–209, Jul. 2021, Art. no. 103220.

ZAHRA WAHEED received the B.S. degree in computer engineering from Bahria University, Islamabad, Pakistan, in 2013, and the M.S. degree in computer engineering from the College of Electrical and Mechanical Engineering, NUST, Rawalpindi, Pakistan, in 2015. She is currently a Ph.D. Scholar with the Department of Computer Engineering, Bahria University. She has published more than eight research papers, including both journals and conferences. Her research interests include image processing and pattern recognition, signal processing, digital system design, machine learning, and deep learning algorithms.

SHEHZAD KHALID received the B.S. degree in computer system engineering from the GIKI, Pakistan, in 2000, the M.S. degree in software engineering from the NUST, Islamabad, Pakistan, in 2003, and the Ph.D. degree in computer vision and machine learning from The University of Manchester, U.K., in 2009. He is currently a Professor with the Department of Computer Engineering, Bahria University, Islamabad. He worked on various projects as a PI/Co-PI amounting to more than Rs. 60 Million. He has published more than 85 impact factor journal publications along with many other HEC recognized and international conference publications. His research interests include computer vision, machine learning, medical multimedia analytics, natural language processing, and signal processing. He got many distinctions during his research and academic career. He has been awarded the Best University Teacher Award by HEC, in 2014, and have been awarded the Best Researcher Award by Bahria University for five consecutive years. He has also been a reviewer of number of impact factor journals.

SYED MURSLEEN RIAZ received the B.S. degree in computer engineering from the University of Engineering and Technology (UET), Taxila, Pakistan, in 2020. He is currently pursuing the M.S. degree in computer engineering with the College of Electrical and Mechanical Engineering, NUST, Rawalpindi, Pakistan. His research interests include machine learning-based application design, deep learning models for reconfigurable architectures, compression of neural networks, and approximate computing.

SAJID GUL KHAWAJA received the B.S., M.S., and Ph.D. degrees in computer engineering from the College of Electrical and Mechanical Engineering (CE&ME), National University of Sciences and Technology (NUST), Islamabad. He is currently an Associate Professor with the Department of Computer and Software Engineering, CE&ME, NUST. His research interests include signal processing, embedded systems, digital system design, and machine learning algorithms.

RIMSHA TARIQ received the B.S. degree in computer engineering from COMSATS University Islamabad, Lahore, Pakistan, in 2019, and the M.S. degree in computer engineering from the College of Electrical and Mechanical Engineering, NUST, in 2022. She is currently working as a Research Associate with the VISPRO Laboratory, Information Technology University. Her research interests include image processing, approximate computing, machine learning systems, and 3D scene reconstruction.