IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 10 June 2023, accepted 26 June 2023, date of publication 28 June 2023, date of current version 7 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3290620

RESEARCH ARTICLE

LCRM: Layer-Wise Complexity Reduction Method for CNN Model Optimization on End Devices

HANAN HUSSAIN[1], **P. S. TAMIZHARASAN**[1], **(Member, IEEE), AND PRAVEEN KUMAR YADAV**[2], **(Member, IEEE)**
[1]Department of Computer Science and Engineering, BITS Pilani, Dubai Campus, DIAC, Dubai, United Arab Emirates
[2]Atlastream Pte Ltd., Singapore 208833

Corresponding author: P. S. Tamizharasan (tamizharasan@dubai.bits-pilani.ac.in)

The work of Hanan Hussain was supported by the Ph.D. Research Fellowship with BITS Pilani, Dubai Campus.

ABSTRACT The increasing significance of state-of-the-art convolutional neural network (CNN) models in computer vision tasks has led to their widespread use in industry and academia. However, deploying these models in resource-limited environments, such as IoT devices or embedded GPUs, presents challenges due to increased complexities and resource consumption. This research paper proposes an optimization algorithm called Layer-wise Complexity Reduction Method (LCRM) to address these challenges by converting accuracy-focused CNNs into lightweight models. It evaluates the standard convolution layers and replaces them with the most efficient combination of substitutional convolutions based on the output channel size. The primary goal is to reduce the computational complexity of the parent models and the hardware requirements. We assess the effectiveness of our framework by evaluating its performance on various standard CNN models, including AlexNet, VGG-9, U-Net, and Retinex-Net, for different applications such as image classification, optical character recognition, image segmentation, and image enhancement. Our experimental results show up to a 95% reduction in inference latency and up to 93% reduction in energy consumption when deployed on GPU. Furthermore, we compare the LCRM-optimized CNN models with state-of-the-art CNN optimization methods, including pruning, quantization, clustering, and their four cascaded optimization methods, by deploying them on Raspberry Pi-4. The profiling experiments performed on each model demonstrate that the LCRM-optimized CNN models achieve comparable or better accuracy than the parent models while providing added benefits such as a 62.84% reduction in inference latency on end devices with significant memory compression and complexity reductions.

INDEX TERMS Convolution neural network, efficient models, optimization techniques, IoT applications, performance metrics, Raspberry pi.

I. INTRODUCTION

Convolution neural networks (CNN)s are mainly designed to process data in multidimensional arrays like images and videos. The basic structure of a CNN consists of two main parts called (a) feature extraction, where the convolution techniques are used to extract the intrinsic representations of the input images and (b) classification, in which the studied features are used to predict the class of the given image. Different layers like convolutional, pooling and fully connected layers are usually stacked to form a CNN. Additional layers like dropout or batch normalization layers are added to regularize or reduce overfitting.

CNNs' can be further divided into two types: i) accuracy-oriented models and ii) lightweight models, based on their number of parameters and the complexity of the computations [1]. Accuracy-oriented models focus mainly on accuracy measurement without considering the complexity of the underlying operations. One of the earliest examples in this category is AlexNet [2], which is a base model for many other CNN architectures like VGG16 and VGG19 with sixteen to nineteen more extended layers [3]. Some recent models include the Inception series [4], DenseNet [5], and ResNet [6]. These models are computationally expensive

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

and typically run on resource-rich environments like cloud GPUs or servers. The second category comprises lightweight CNN models that operate in resource-constrained environments like embedded devices, IoT, or edge devices [7]. The depth size of these models is less than accuracy-oriented models, which reduces the computations and the number of underlying parameters. Some of the examples of such CNN models are SqueezeNet [8], MobileNet v1 and v2 [9], ShuffleNets [10], and EfficientNets [11].

CNNs have proven to be highly effective in various applications. They have achieved state-of-the-art performance in image classification, object detection and recognition tasks [12]. In one-dimension data, CNNs have demonstrated success in tasks such as speech processing [13], time series classification tasks [14], and intrusion detection system [15]. The versatility and superior performance of CNNs have significantly impacted industries and advanced the field of artificial intelligence.

The main challenge in the portability of different types of CNN models between resource-rich and resource-limited environments is maintaining the balance between the required performance and the available resources like memory, energy, and computation power [16]. Hence, the optimization study must consider the resources mentioned above, complexity factors affecting the resources such as the number of parameters, floating-point operations, and other performance metrics like throughput and inference latency. Additionally, the most recent models are mainly optimized for inference rather than for the training phase. Even though the inference efficiency will help the model to accelerate its performance, understanding the training efficiency of a model is also essential as it unlocks the possibility of on-device learning on resource-limited devices. Thus, to overcome such limitations, we propose the Layer-wise Complexity Reduction Method (LCRM), which redesign the existing CNN with the following objective: (i) to reduce the computational complexity of the model, (ii) to improve the hardware resource utilization without affecting the performance.

The key contributions of this paper are as follows:

1) We introduce a novel optimization framework known as the layer-wise complexity reduction method (LCRM), which effectively reduces the computational complexity of accuracy-oriented CNN models in section III-A. Our proposed framework addresses the challenge of optimizing CNN models by specifically targeting resource-constrained end devices.
2) We analyze the training efficiency and testing efficiency of the LCRM-optimized CNN models in terms of (i) Complexity by calculating the reduction in parameters and floating point operations IV-B. (ii) Performance metrics according to the application like accuracy, inference latency, precision, PSNR etc. (iii) resource utilization including CPU and GPU utilization, energy consumption, memory consumption and memory compression in section IV-C.
3) We evaluate the effectiveness of the LCRM framework by optimizing standard CNN models, including AlexNet, VGG-9, U-Net, and Retinex-Net, for four different applications: image classification, optical character recognition, image segmentation, and image enhancement in section IV-D. Through these evaluations, we demonstrate the versatility and applicability of our framework in enhancing the performance of CNN models across diverse tasks.
4) We assess the performance of the LCRM technique against existing state-of-the-art optimization techniques such as quantization (Q), pruning (P), clustering (C) and cascaded optimization techniques, including sparsity preserving clustering (PC), sparsity preserving quantization (PQ), cluster preserving quantization (CQ), and sparsity and cluster preserving quantization (PCQ) in section IV-E. By comparing the results, we demonstrate how LCRM outperforms these techniques regarding model optimization and efficiency.

We assess the feasibility of performing inference by deploying models to the Raspberry Pi and GPU platforms, highlighting the practicality and efficiency of the LCRM technique.

The remainder of the paper is arranged as follows: In section II, we reviewed related works in the literature, followed by the methodology section of the proposed algorithm, complexity derivations, and profiling methodology of different performance metrics. Section IV explains the experimental setup and hardware parameters' profiling results. Section V answers all four research questions defined in the paper and discusses some of the shortcomings of the methodology. Finally, we will conclude the work in section VI while discussing possible future directions.

Our experiments and the framework are available at the following code repository: https://github.com/hana-an/LCRM.

II. RELATED WORKS

This section discusses related works about the techniques that improve the efficiency of CNN models and the profiling of different CNN models.

(i) The efficiency of a deep learning model can be achieved in many ways, but most of the techniques in the literature consider the following four criteria [17]:

(a) Common optimization techniques involving quantization, clustering, or pruning are used to compress the network layers with a minor trade-off in accuracy [18]. The TensorFlow framework provides an API that can use the above techniques. The total parameters of any CNN model involve both trainable and non-trainable parameters. Trainable parameters are values modified according to their gradient (the error/loss/cost derivative relative to the parameter). In contrast, non-trainable parameters are those values that are not optimized according to their gradient. Applying the classic optimization techniques increases the amount of non-trainable parameters considerably.

Other ensemble optimization techniques include cascaded PCQ, which involves quantization, clustering, and pruning [19]. Additionally, deep compression techniques involve pruning, quantization, and Huffman coding to compress and optimize model parameters [20]. Another approach involves compressing deep CNNs in the frequency domain using techniques like Discrete Cosine Transform (DCT), clustering, and quantization [21]. These methods employ hybrid optimization techniques that involve a combination of optimization techniques to compress the final CNN model.

However, our proposed framework focuses on the channel-wise reduction in total computation by separating the standard convolution into two distinct steps. The first step involves depthwise convolution, which performs a separate convolution for each input channel. The second step involves pointwise convolution, which applies a 1×1 convolution to combine the output channels. This architectural design choice based on the input and output channels of the parent model layers allows efficient computation while maintaining the representational power of the CNN.

(b) Knowledge distillation techniques like transfer learning use fewer data so that models can converge faster with less prediction error [22]. The previously learned knowledge from the data is used for training the new model, thereby reducing the training time [23].

(c) Hyperparameter search using automated algorithms to search the hyperparameters such as epochs, number of layers, or optimizers until the model returns the sequence with the highest accuracy [24]. Examples include grid search or parameter sweep and random search [25]. Similar algorithms like neural architecture search (NAS) return the model architecture itself [26]. An example of the NAS returned model is the EfficientNet-B_0.

(d) Efficient architectures developed from scratch to change the parent model significantly. A simple example is the introduction of convolution layers in computer vision applications instead of fully connected layers. The convolution layer made parameter sharing easier and reduced the requirement to learn separate weights for each pixel.

(ii) Profiling and performance evaluation of the deep learning models:

In [27], the authors studied the training performance, resource utilization, and power efficiency of sixteen deep learning models implemented on embedded GPU [28]. However, optimization techniques required for the efficient processing of the model are not explored. In contrast, our work focuses on an optimization technique along with deep learning models, training profiling, and the profiling of CNN inference. Many works in the literature analyze the performance and hardware utilization of matrix multiplication and 2D convolution operations in a high-end environment with multiple GPUs and TPUs [29], [30], [31].

Recent work in [32] analyzed the forward and backward pass of the CNN training algorithm and measured peak memory consumption. In contrast, the authors Ren et al. [33] considered both CNNs and BERTs [34] to analyze throughput by changing different optimization techniques and workloads. Li et al. [35] profiled the performance of two CNN-based models for image classifications on NVIDIA TITAN RTX. They found a significant latency throughput trade-offs exist when parameters like activation functions, loss functions, optimizers, epochs, or batch size are changed. Unfortunately, the above works only study the quality of CNNs. On the contrary, we profile and analyze the quality parameters like accuracy and the resource utilization of the models.

III. METHODOLOGY

In this section, we discuss the proposed LCRM algorithm that converts the parent model to an optimized version in different scenarios. We also describe the types of substitutional convolutions applied in the algorithm and calculate their computational complexity for comparison. The rationale behind substituting the standard convolution layers with other types of convolution layers (or a combination of different convolution layers) is to reduce the number of model parameters and resource requirements such that they are easily deployable on end devices [9]. Finally, we introduce a profiling workflow methodology of different performance metrics to evaluate the resource reductions during the training and inference process.

A. LCRM METHODOLOGY

The proposed LCRM algorithm optimizes each convolution layer of the parent CNN model to generate the optimized set of new layers. The algorithm requires the standard convolution layer L, input channel size M, and output channel size N as input. Based on the input and output channel sizes M & N, the algorithm analyzes and replaces the L with a set of substitutional layers except for the initial and final classification layers. The replacement is done with the most optimized available choices of convolution layers L so that its output shape remains the same while reducing the number of parameters and operations. The algorithm's output gives the optimized layer L' and changes the number of parameters δP after the layer-wise optimization.

There are three cases of the LCRM algorithm (Algorithm1), namely up-scaling, down-scaling, and no scaling:

1) UPSCALING (CASE 1)

Upscaling occurs when the input channel size M of the convolution layer L is less than the output channel size N. There are two cases in this section, they are:

- Case 1.1: When M mod N is zero, the algorithm calculates the multiplier m1 (such that $m1 * M = N$) and replaces the existing standard convolution with depthwise separable convolution. The *replacedws*() function takes the CNN layer $L, m1$ and returns the depthwise convolution layer L' by keeping all other hyperparameters of the L unchanged.

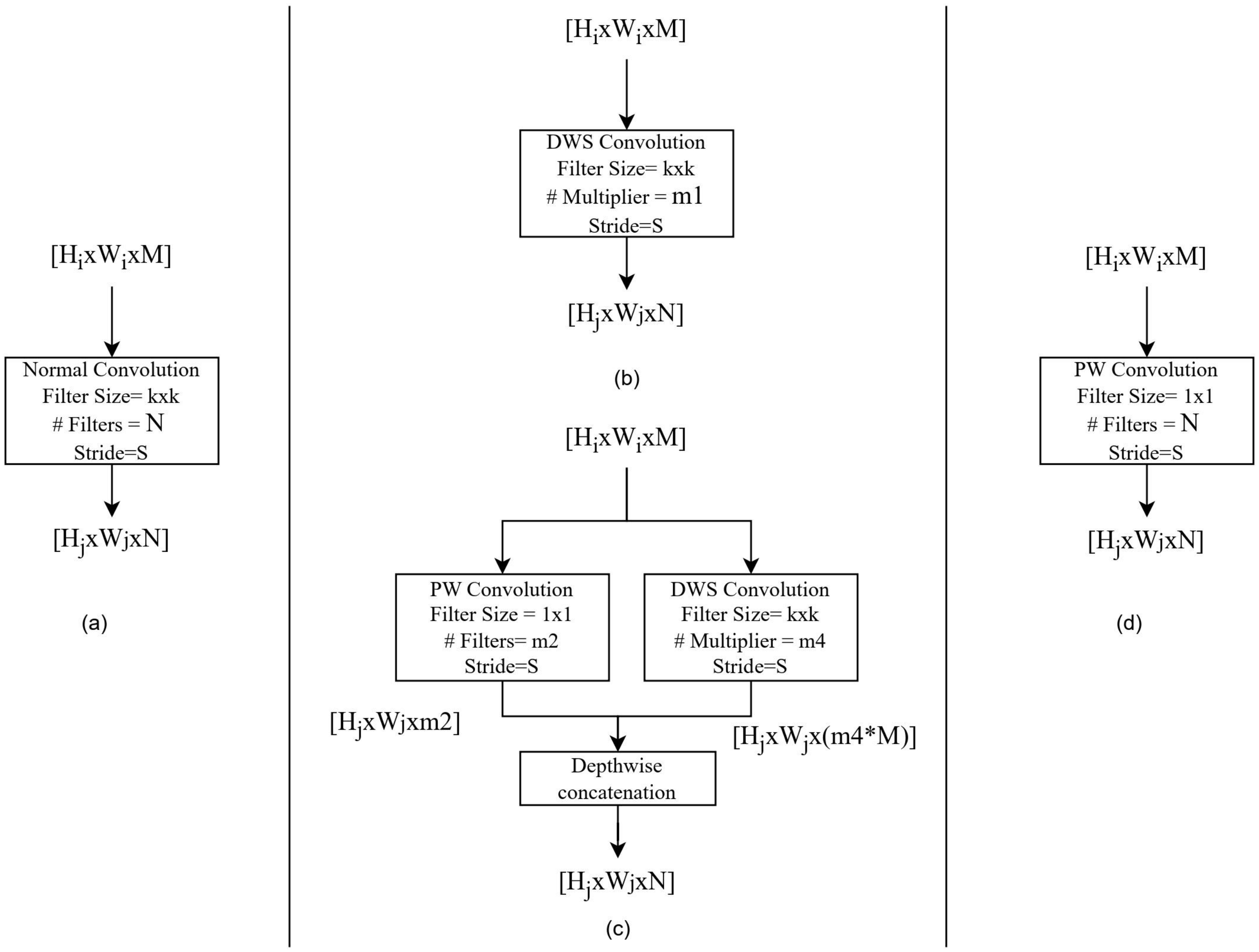


FIGURE 1. (a) standard convolution from parent model. (b) case 1.1, upscaling: replacing (1.a) with depthwise separable convolution. (c) case 1.2, replacing (1.a) with a combination of depthwise separable and pointwise convolution. (d) case 2, downscaling: replacing (1.a) by point-wise convolution.

- Case 1.2: When M mod N is not equal to zero: Here, the algorithm replaces input layer L with (a) pointwise convolution layer and (b) depthwise separable convolution layers, followed by depthwise concatenation. The substitution requires two functions, *replacedws*() and *replacepws*(), to be applied on layer L to return depthwise-separable convolution and pointwise convolution, respectively. The algorithm also evaluates function parameters $m2$, $m4$ required to pass to the function using simple calculations given in the algorithm. The values are $m2$=($N\%M$), $m3$=$N-m2$) and $m4$, where $M * m4 = m3$. Both functions return a set of layers, and the final function *concatenate*() performs a depthwise-concatenation operation.

2) DOWNSCALING (CASE 2)

Downscaling occurs when the input channel size M of the convolution layer L is greater than the output channel size N. Here, the algorithm uses *replacepws*() function to replace the current convolution with a pointwise convolution layer and keeps all the hyperparameters of layer L as such, except the kernel parameters. The size of the kernel is $[1*1*M]$, and the number of kernels $m5$ is N in number. The newly added L' is returned from the function and saved for later computations.

3) NO SCALING (CASE 3)

No scaling requires when the input channel size M of the convolution layer L is equal to the output channel size N. This case can be considered similar to upscaling (case 1.1) when multiplier m1 equals 1, where the input standard convolution layer L is replaced with depthwise separable layer L' having the same number of channels.

All the cases are diagrammatically represented in Figure 1 and given in Algorithm 1. The variables used in the Algorithm are summarized in Table 1 For all the cases, the proposed method calculates the total parameter of the input standard convolution layer L and the output convolution layer L' and finds their difference to return as the output from the LCRM.

Algorithm 1 LCRM

```
1:  $LCRM(L, M, N)$                              ▷ inputs to the algorithm
2:  if $M <= N$ then                  ▷ upscaling channels or no scaling
3:      if $(N\%M) = 0$ then                          ▷ case 1.1, case 3
4:          m1= N /M
5:          $L' \leftarrow replacedws(L, m1)$
6:      else                                                  ▷ case 1.2
7:          $m2 \leftarrow (N\%M)$
8:          $l1 \leftarrow replacepw(L, m2)$
9:          $m3 \leftarrow (N - m2)$
10:         $m4 \leftarrow m3/M$
11:         $l2 \leftarrow replacedws(L, m4)$
12:         $L' \leftarrow concatenate(L, l2)$
13:     end if
14: else                                  ▷ downscaling channels case 2
15:     $L' \leftarrow replacepw(L, m5)$
16: end if
17: $P1 \leftarrow calcparam(L)$
18: $P2 \leftarrow calcparam(L')$
19: $\delta P \leftarrow P1 - P2$
20: $return\ L', \delta P$                    ▷ output from the algorithm
```

TABLE 1. Variables and their corresponding descriptions.

Variable Name	Description
L	Standard convolution layer
L'	Optimized output layer
$l1$	Point-wise convolution layer
$l2$	Depth-wise separable convolution layer
H_i	Height of the feature map
W_i	Width of the feature map
M	Size of the input channel
N	Size of the output channel
$m1, m4$	Multipliers used in the DWS convolution layer
$m2, m5$	Multipliers used in the PW convolution layer
$m3$	Variable for storing intermediate result
P	Number of parameters in a convolution layer
F	Number of multiplication in a convolution layer
$k * k$	Size of the kernel (height x width)
D	Dimension of the feature map (D=H=W)

Rationale Behind the Selection of Substitutional Layers: The grid search and linear search algorithm resolve the challenge of selecting the architecture type of substitutional layer(s) from various convolutions and hyperparameters available in the literature. The search algorithm execution returns seven sets of optimized models with different substitutional convolutions. The most optimized model is selected based on two factors: (i) reduced computational complexity and (ii) less reduction of test accuracy. We have described the grid search, hyperparameter search, different substitutional layers of returned optimized models, their computational complexity, and the finalization of LCRM substitutional layers in the Appendix A.

B. CONVOLUTION TECHNIQUES

In this section, we discuss the parameters calculation and complexity of the computations in substitutional convolutions layers used in the LCRM algorithm. The layers are (a) standard convolution, (b) depthwise separable convolution containing: depthwise convolution, and pointwise convolution or point-by-point convolution in different combinations.

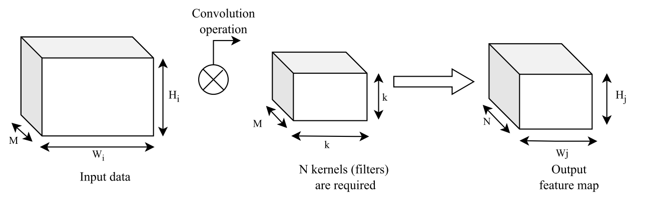


FIGURE 2. Standard convolution.

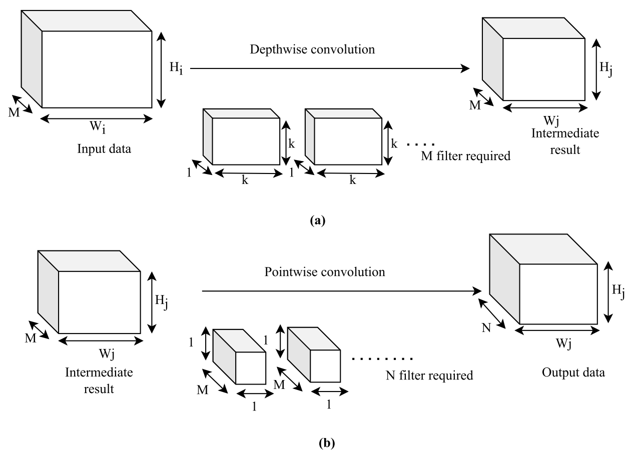


FIGURE 3. Depthwise separable convolution.

1) STANDARD CONVOLUTION

In a standard convolution, if the input data is of size $[H_i * W_i * M]$, it implies that the input image is of dimension W_i and H_i with M channels. Any convolution operation requires N filters or kernels to generate an output feature map of size $[H_j * W_j * N]$. The kernels are convolved with input data to form the output feature map [2]. Consider Figure 2. Here the number of multiplications in 1 convolution operation is the size of filter $= k * k * M$. Since the process involves N such filters that stride over the input data vertically and horizontally in H_j and W_j times, the total multiplications now become $N * H_j * W_j *$(multiplication per convolution).

Therefore, the total no of multiplications in standard convolution $= N * H_j * W_j * (k * k * M)$.

2) DEPTHWISE SEPARABLE CONVOLUTION

Depthwise separable convolution is a combination of depthwise convolution and pointwise convolution [36].

(a) Depthwise convolution: In depth-wise convolution, the convolution operation takes each channel separately for processing. Given M channels in the input data, we need M filters for convolution, as the output will be of size $[H_j * W_j * M]$ as shown in Figure 3 (a). Thus filters should have size $[k * k * 1]$ with M numbers. Here the single convolution operation requires $k * k * 1$ multiplications, and the filter strides over by $H_j * W_j$ times across all the M channels.

Therefore, the total number of multiplications in depthwise convolution $= M * Hj * Wj * k * k$

(b) Pointwise convolution. A pointwise convolution executes a 1×1 convolution operation on the M input channels, shown in Figure 3 (b). So the filter should be of size $[1*1*M]$ and N. Such filters are required to generate an output of size $[H_j * W_j * N]$. A single convolution operation requires $1 * M$ multiplications, and the filter strides over by $H_j * W_j$ times. Thus the total number of multiplications becomes $1*M*H_j*W_j*$ number of filters.

Therefore, the total no of multiplications in pointwise convolution $= M * Hj * Wj * N$.

However, the complete depthwise separable requires the sum of depth-wise convolution multiplications + Pointwise convolution multiplications.

Therefore, the total multiplications,

$= (M * H_j * W_j * k * k) + (M * Hj * Wj * N)$

$= M * Hj * Wj * (k^2 + N)$

C. PARAMETER AND COMPLEXITY CALCULATION OF LCRM CONVOLUTIONS

This section evaluates the parameter and complexity reduction for each case of LCRM by comparing it with the standard convolution from the equations derived from the previous sections (B.1 and B.2) to show the efficacy of the proposed algorithm.

Let the total number of trainable parameters in the standard convolution be P_0 (without considering bias values), and the number of floating-point calculations is F_0 in a standard convolution process.

$$P_0 = k * k * M * N \tag{1}$$

$$F_0 = k * k * M * N * H_j * W_j \tag{2}$$

The following subsection describes the different cases of the LCRM algorithm and their parameter and floating-point operation calculation reduction with regard to standard convolution.

1) CASE 1: UPSCALING

Upscaling is of two types: cases 1.1 and 1.2.

(a) Case 1.1 in upscaling occurs when $M < N$ and M mod N is zero (or, $m1 * M = N$). Here the algorithm replaces standard convolution with a depthwise separable convolution. As already discussed, depthwise separable convolution consists of both depthwise and pointwise convolution. Thus, the following simplified equation of the number of parameters $P_{1.1}$ and floating-point operation $F_{1.1}$ will have the terms from both the convolutions mentioned above.

$$P_{1.1} = (k^2 * M) + (M * N) \tag{3}$$

$$F_{1.1} = (k^2 * H_j * W_j * M) + (M * N * H_j * W_j) \tag{4}$$

We derive the reduction in the number of parameters and floating-point operation with respect to the standard convolution by calculating the ratio between equations (3) and (1) and the ratio between (4) and (2), respectively.

$$\frac{P_{1.1}}{P_0} = \frac{1}{N} + \frac{1}{k^2} \tag{5}$$

$$\frac{F_{1.1}}{F_0} = \frac{1}{N} + \frac{1}{k^2} \tag{6}$$

The simplified results from (5) and (6) show that the number of parameters and floating-point calculations of the depth-wise separable convolution is only $\frac{1}{N} + \frac{1}{k^2}$ times the standard convolution. Thus there is a lessening in the parameter and computing cost of the proposed model in case 1.1.

(b) case 1.2 in upscaling happens when $M < N$ and M mod N is not equal to zero. Here the algorithm replaces standard convolution with a pointwise convolution with m2 filters and a depthwise separable convolution with $m4$ times M filters), followed by a depth-wise concatenation to form a new feature map. Here $P_{1.2}$ and $F_{1.2}$ represent the parameters and operations of both pointwise convolutions (RHS term1) and depthwise separable (term 2 and term 3), respectively.

$$P_{1.2} = (M * m2) + (k^2 * M) + (M * (M * m4)) \tag{7}$$

$$F_{1.2} = H_j * W_j[(M * m2) + (k^2 * M) + (M * (M * m4))] \tag{8}$$

Reduction in the number of parameters and floating-point operations compared to the standard convolution:

$$\frac{P_{1.2}}{P_0} = \frac{1}{k^2 N}(m2 + M * m4) + \frac{1}{N} \tag{9}$$

As seen in Figure 1, (c), The algorithm derives a new feature map of the shape $[H_j * W_j * N]$ by using the depthwise concatenation of two feature maps of size $[Hj * Wj * m2]$ and $[H_j * W_j * (M * m4)]$. The term $(m2 + M * m4)$ is equal to N as the algorithm ensures that the input and output shape of feature maps and channels remains the same as that of the parent model. Therefore, by replacing the above term in equation 9, we get:

$$\frac{P_{1.2}}{P_0} = \frac{1}{N} + \frac{1}{k^2} \tag{10}$$

Similarly, the ratio between the operations is:

$$\frac{F_{1.2}}{F_0} = \frac{1}{N} + \frac{1}{k^2} \tag{11}$$

The inference from the above results shows a considerable reduction in parameters and operations identical to case 1.1.

2) CASE 2: DOWNSCALING

Downscaling occurs when $M > N$. In this case, the algorithm replaces the standard convolution with a pointwise convolution with N filters of size $[1 * 1 * M]$. The number of parameters and operations in pointwise convolutions is:

$$P_2 = 1 * 1 * M * N \tag{12}$$

$$F_2 = M * N * H_j * W_j \tag{13}$$

We calculate the ratio of equations (12) and (1) to find the reduction in the number of parameters and the ratio

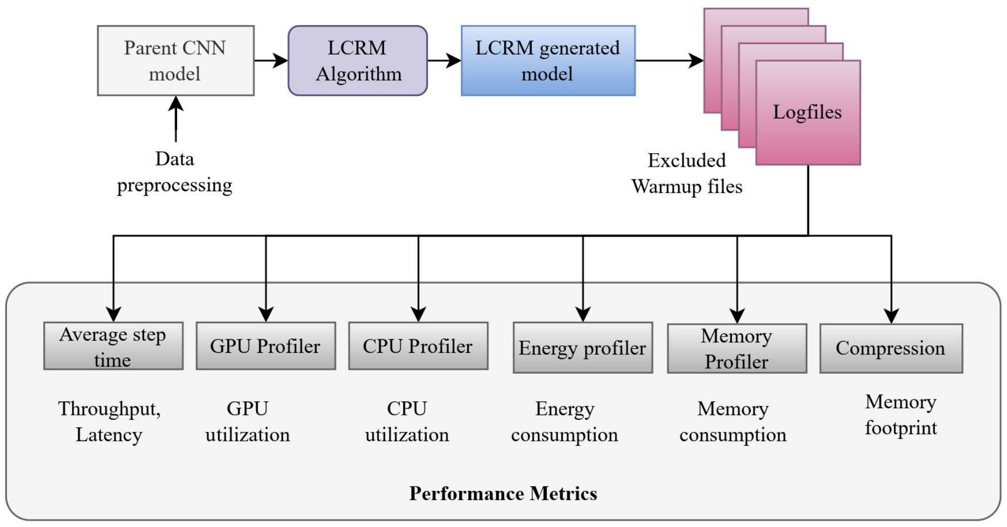


FIGURE 4. Profiling workflow.

TABLE 2. Comparison between the complexities of standard convolution and types of convolution operations in LCRM along with their ratios.

Type of convolution	Complexity	Ratio (R)
Standard convolution	$K^2 * M * N * D^2$	1
LCRM case 1.1	$(k^2 + N)M * D^2$	$\frac{1}{N} + \frac{1}{k^2}$
LCRM case 1.2	$(k^2 + N)M * D^2$	$\frac{1}{N} + \frac{1}{k^2}$
LCRM case 2	$M * N * D^2$	$\frac{1}{k^2}$
LCRM case 3	$(k^2 + N)M * D^2$	$\frac{1}{N} + \frac{1}{k^2}$

between (13) and (2) to find the reduction in the number of floating-point operations of pointwise convolution with respect to the standard convolution. The ratio between parameters is:

$$\frac{P_2}{P_0} = \frac{1}{k^2} \tag{14}$$

Similarly, the ratio between the operations is:

$$\frac{F_2}{F_0} = \frac{1}{k^2} \tag{15}$$

Here the P_2 and F_2 values are reduced up to $\frac{1}{k^2}$ times of standard convolution parameters and operations.

3) CASE 3: NO SCALING

No scaling occurs when $M = N$ is a particular case of 1.1. when M=N, the multiplier m1 defined in the algorithm is 1 for the term $M * m1 = N$. Hence, the calculated parameters, operations, and ratio remain the same as equations (5) and (6).

We summarise the complexity for standard convolution and each case of the LCRM algorithm along with the ratio of complexity for different cases of LCRM with the standard convolution in Table 2. We can see an apparent reduction in the complexity of all the cases.

D. PROFILING METHODOLOGY

In this section, we present profiling workflow, different performance metrics, tools for evaluating training and inference of the base model and LCRM generated model as shown in Figure 4. The profiling method involves the execution of an optimized LCRM model and its log file generation during the training and inference process. During the process, we ignore warmup files (log files generated during the first three epochs) from the profiler and use the remaining log files for further study on training efficiency and testing efficiency. The following subsections discuss the performance metrics with examples and their respective profilers.

1) THROUGHPUT AND LATENCY

The throughput metric evaluates the efficiency of the training process and is defined as the number of images processed during the training per second. However, latency is more suitable for assessing inference and is defined as the time required to classify a single image and is represented as seconds per inference [37]. For example, when we train the base model using a batch size of 16 and an average step time of 200 milliseconds (0.2 seconds), the training throughput is 16/0.2 = 80 images per second. If the same model takes 100s to classify the test dataset of 10000 images, then the inference latency is 100/100000 = 0.01 seconds per image inference.

2) MODEL ACCURACY

The accuracy is defined as the number of images correctly classified with respect to the total number of images in the dataset [38]. The equation for accuracy calculation is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{16}$$

3) ENERGY CONSUMPTION

A Python library called *PyJoules* profiles the GPU's energy consumption during training and inference at different batch sizes. It measures the energy footprint of a host machine and the execution of a piece of python code to return the execution duration and the energy consumed in microjoules. The profiling workflow calculates the energy consumptions at different workloads during model training and inference, [39].

4) HARDWARE UTILIZATION

Hardware utilization contains CPU utilization and GPU utilization [40]. CPU utilization is defined as the system's usage of processing resources or the amount of work handled by a CPU. The metric is estimated using the *psutil* library.

The GPU utilization is defined as how frequently GPU is active ($T_{GPUactive}$) during the model training (T_{total}). In contrast, the GPU idle ratio is the percentage of the training time that the core is idle. GPU utilization should be maximum, and the idle ratio should be low to increase throughput. The equation for calculation:

$$GPU_{utilization} = \frac{T_{GPUactive}}{T_{total}} * 100 \quad (17)$$

$$GPU_{IdleRatio} = 1 - GPU_{utilization} \quad (18)$$

5) PEAK MEMORY CONSUMPTION

A model's peak memory consumption (PMC) is the maximum memory used for training. For example, At batch size 256, MobileNet V2 (3.5M parameters) consumes 4.18 GiB of memory when trained from scratch. The TensorBoard memory profiler quantifies the peak memory consumption of the training process, whereas the Memprofiler library quantifies the memory utilization during the inference on Raspberry pi.

6) MODEL MEMORY FOOTPRINT

Memory requirement refers to the amount of secondary memory that a model requires to store or deploy them efficiently. The profiling workflow needs the optimized model and generated weights to convert into a compressed version for compression comparison. Moreover, we need TFLite models for deployment to targeted machines like IoT devices or end devices. The TensorFlow Lite framework helps in such tiny hardware device deployment. Hence, converting the TensorFlow model to a TFLite model is another option to calculate the memory footprint.

IV. EXPERIMENT EVALUATION

A. EXPERIMENTAL SETUP

- Hardware specifications: Intel Core i7-9750H, 16 GB RAM, NVIDIA Tesla K80 GPU (training), NVIDIA Geforce GTX 1650 (4 GB) (training and inference), Raspberry pi 4 Model B (inference only);
- Software requirements: Python 3.8, Tensorflow-GPU, Keras, Cuda Toolkit, CuDNN, PyJoules, Memprofiler;
- Datasets: CIFAR-10 for image classification, EMNIST [41] for optical character recognition (OCR), LOL (LOw-Light) Dataset [42] for image enhancement and Kaggle data science bowl 2018 dataset [43] for image segmentation.
- Parent models: AlexNet is selected to show how the LCRM algorithm works in all the defined conditions (case 1 to case 3) in the image classification application.

TABLE 3. Reduction in floating-point operations for each case given in the above example.

Layers	#Operations in parent model	#Operations in LCRM model	% Reduction in operations
D1	Initial layers are not considered		
D2: Case 1.1	276889600	12157184	95.61%
D3: Case 1.2	128065536	14561280	88.63%
D4: Case 3	193098816	21955680	88.63%
D5: Case 2	128065536	14229504	88.89%
D6	Final layers are not considered		

Other parent models considered include VGG-9 [3] for optical character recognition, Retinex-Net's decomposition net [42] for image enhancement and U-Net [44] for image segmentation.

B. LCRM RESULTS

To evaluate the LCRM algorithm on Alexnet, we divide all the layers into blocks D1 to D6, as shown in Table 4. We keep the middle convolution layers of AlexNet from D2 to D5 as such and modify only the initial convolution in D1 (such that LCRM case 1 applies to convolution in D2 for demonstration). We also modify the final classification layers D6 to fit into the CIFAR-10 dataset.

However, the modification of convolution layers in D1 and D6 will not come under any defined cases in the LCRM algorithm and has zero impact on optimizing calculation. Table 4 shows the parent models in different blocks and their corresponding substitution of LCRM layers, the total number of parameters, and percentage reduction. In summary, there is a reduction of 31.25% parameters, which is from 11.2 M (parent model) to 7.7 M (LCRM-generated model).

Table 3 further shows the number of floating-point operations of both the models and their percentage reduction, excluding bias terms. The results are calculated using the corresponding equations given for each case of the LCRM model. For example, to obtain the number of operations in cases 1.1, 1.2, 2, and 3, we are using equations 4, 8, 13, and 4, respectively. As observed in Layers D6, from the Table, our analysis excludes the final fully connected or dense layers of the CNN. Our specific focus is on reducing the layer-wise complexity of standard convolutional layers, which is relevant to accuracy-oriented network architectures and various computer vision applications.

C. PROFILING RESULTS

The profiling methodology ignores the first three training steps and their log files since the TensorFlow runtime system often uses initial steps to study underlying hardware architecture features like cache capacities or memory access latency to achieve optimization.

1) TRAINING THROUGHPUT

The training throughput of CNN models increases as the batch size increases, as seen in Table 5. As we change batch size from 16 to 256, the throughput changes from 131-124

TABLE 4. Comparison between the accuracy-oriented parent model and LCRM-generated optimized model.

Depth	Parent model layers	#Parameters in parent model (P0)	Input shape [HixWixM]	Output shape [HjxWjxN]	LCRM replaced layers	#Parameters after applying LCRM	% Reduction in number of parameters
D1: Initial Layers	Standard convolution: K=11x11, S=4 #Filters=64	23296	224x224x3	54x54x64	Initial layers are kept as such	Parameters remain the same as that of P1	0
	BN (Batch normalization)	256	54x54x64	54x54x64			
	ReLu	0	54x54x64	54x54x64			
	Max pooling K= 3x3 , S=2	0	54x54x64	26x26x64			
D2: Case 1.1: Upscaling (M is an exact multiple of N)	Standard Convolution: K = 5X5, S=1, #Filters=256	409856	26x26x64	26x26x256	DWS convolution: K= 5X5, S=1, #Multiplier = m1	DWC:6656 + PWC: 16640	94.316%
	BN	1024	26x26x256	26x26x256	No change in these layers	Parameters remain the same as that of P1	0
	ReLu	0	26x26x256	26x26x256			
	Max pooling: K= 3x3 , S=2	0	26x26x256	12x12x256			
D3: Case 1.2: Upscaling (N is not an exact multiple of M)	Standard Convolution: K= 3x3, S=1, #Filters=386 +BN +ReLu	885120 + 1536 + 0	12x12x256	12x12x386	PW Conv: K=1X1, S=1 #Filters= m2 +BN +ReLu	2560 +1024 +0	95.99%
			12x12x386	12x12x386	DWS conv K= 3x3, S=1 #Multiplier= m4 +BN+ReLu	32896 + 512 +0	0
					Concatenate (PWconv, DWSconv)	0	
D4: Case 3: No scaling (N =M)	Standard Convolution: K= 3x3, S=1, #Filters=386	1327488	12x12x386	12x12x386	DWS convolution: K= 3x3, S=1 #Multiplier = m1	3840	99.71%
	BN	1536	12x12x386	12x12x386	No change in these layers	Parameters remain the same as that of P1	0
	ReLu	0	12x12x386	12x12x386			
	Max pooling	0	12x12x386	12x12x386			
D5: Case 2: Downscaling (N <M)	Standard Convolution: K= 3x3, S=1, #Filters=256	884992	12x12x386	12x12x256	PW Conv: K=1x1, S=1 #Filters=m5	98560	88.86%
	BN	1024	12x12x256	12x12x256	No change in these layers	Parameters remains the same	0
	ReLu	0	12x12x256	12x12x256			
	Max pooling	0	12x12x256	6x6x256			
D6: Final Layers	Flatten	0	6x6x256	6400	No change in these layers	Parameters remain the same as that of P0	0
	Dense layer	6554624	6400	1024			
	Dropout	0	1024	1024			
	Dense layer	1049600	1024	1024			
	Dropout	0	1024	1024			
	Softmax layer	10250	1024	10			

and 555 - 792 images/ second in the parent and LCRM- - generated models, respectively. It is due to the increase in parallel computations and the batch size increase. The result implies that the LCRM algorithm speed up the throughput to 5x-6x times.

2) TESTING ACCURACY

The TensorFlow parent and LCRM-generated model obtain comparable testing accuracies, such as 0.8274 and 0.8279, respectively. The parent model converges at a 0.0001 learning rate for the Adam optimization algorithm. In contrast,

TABLE 5. Training throughput of CNN models at different batch size.

	Parent Model		LCRM Model	
Batch Size	Avg. Step time (ms)	Throughput (imgs/s)	Avg. Step time (ms)	Throughput (imgs/s)
16	141.4	113.1542	28.8	555.556
32	266.9	119.8951	47.8	669.456
64	525.8	121.7193	87	735.632
128	1039.6	123.1243	166.9	766.926
256	2064.4	124.007	323.2	792.079

TABLE 6. Inference latency at various batchsize on NVIDIA-GPU 1650.

	Parent model (AlexNet)		LCRM-AlexNet model	
Batch size	Inference time (s)	Latency (s/inference)	Inference time (s)	Latency (s/inference)
16	90.77	0.0091	5.16	0.00052
32	89.2	0.0089	4.2	0.00042
64	88.02	0.0088	3.9	0.00039
128	88.39	0.0088	3.86	0.00039
256	87.77	0.0088	3.81	0.00038

our proposed model requires more training epochs to converge the model with the same hyperparameters. At epoch 25, the parent model converges to 0.8274, whereas the LCRM-generated model reaches only 0.8011; however, at epoch 45, the model outperforms slightly to reach 0.8279. The point of convergence is found by setting the patience parameter as two. This setting makes the training process efficient by stopping the algorithm and updating the model's parameters to return the converged model when further iterations do not significantly improve the model's loss/accuracy values.

3) INFERENCE LATENCY

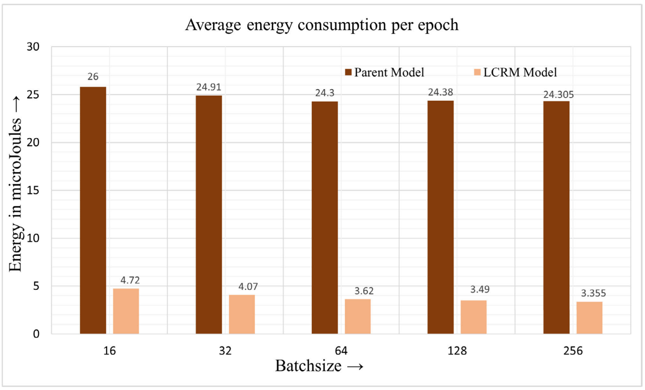
Inference latency per image decreases as we increase the batch size and remains the same at higher batch sizes for both CNN models. Additionally, the LCRM-generated model's inference latency was reduced to 94% -95%, as seen in Table 6. It is due to the reduction of floating operations in the new model compared to the parent model.

4) GPU ENERGY CONSUMPTION DURING TRAINING

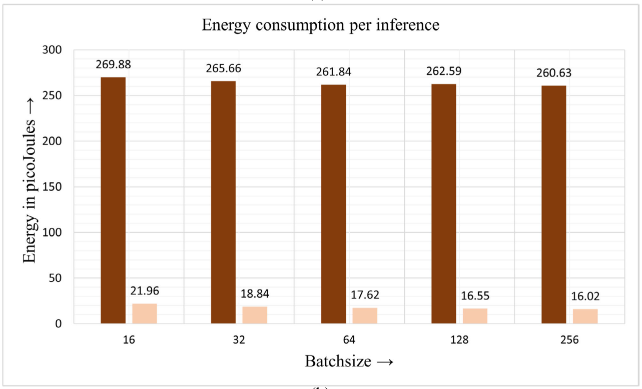
For the GPU-accelerated CNN models, as the batch size increases, there is a slight decrease in the GPU energy consumption. Figure 5 (a) shows the average energy consumed per epoch in microjoules at different batch sizes. Among the two models, the LCRM model reduced the energy consumption by 82% - 85% compared to the parent model. It is due to the reduction in the total time required to train the model concerning the increase in batch size.

5) ENERGY CONSUMPTION DURING INFERENCE

Similar to the training energy consumption of GPU, the energy consumption per inference also reduces as we increase the workload. Fig: 5 (b) shows the average energy consumed per epoch in picojoules at different workloads. Among the two models, the LCRM model has energy consumption per inference by 91% to 93% compared to the parent model.



(a)



(b)

FIGURE 5. (a) GPU energy consumption per epoch during the training process at different batch sizes and (b) Energy consumption per inference at the different batch sizes.

6) GPU UTILIZATION

GPU utilization increases as the batch size increases for all CNN models. The LCRM-generated model's GPU utilization seems inefficient compared to its parent model when the batch sizes are small. However, at batch size 256, both reach a comparable value and seem to be at the highest GPU utilization of 96.5%, and 97.8% for the parent and LCRM generated models, respectively, as shown in Table 7. The Table considers the GPU idle ratio (or $1 - GPU_{utilization}$) as equations 17 and 18. Another inference from the experiment shows a trade-off between batch size, PMC, and GPU utilization. Hence a suitable batch size must be considered while training to better PMC and GPU utilization.

According to Lu and Zhang [45], the depthwise separable convolutions (substitutional layers of the proposed model) cannot make effective use of the GPU parallelism when the batch size is small (< 256). We also observed similar results showing an ineffective use of GPU resources (memory and work distribution) leading to low GPU utilization in Table 7.

7) PEAK MEMORY CONSUMPTION

As already discussed, peak memory consumption (PMC) is one of the critical limitations while training a CNN model. Consider the Table 7, which quantifies the peak memory used by the model during the training period at the different workloads. As the workload increases, PMC also increases due to

TABLE 7. GPU utilization and PMC.

	Parent Model		LCRM Model		
Batch size	GPU idle ratio %	PMC (MiB)	GPU idle ratio %	PMC (MiB)	% reduction
16	34.2	302.28	78.4	200.06	33.82
32	12.1	566.06	68.4	444.28	21.51
64	11.6	1136.64	51.4	667.97	41.23
128	6.2	1495.04	35	919.15	38.52
256	3.5	2027.52	2.2	1525.76	24.75

the demand for storing more data while training. However, our proposed algorithm shows a reduction in the elevated memory consumption from 25% to 34% when compared to the parent model.

8) INFERENCE ON RASPBERRY PI

The TensorFlow models have formats like the saved model format, Keras, or KerasH5 format. However, resource-constrained devices like raspberry pi require TensorFlow Lite (TFLite) format for deployment. To convert the original TensorFlow model to a TFLite model (an optimized FlatBuffer format identified by the.tflite file extension), we use a TensorFlow Lite converter. A model evaluation is essential before attempting the conversion. The evaluation determines if the model's contents are compatible with the TFLite format. Also, it checks if the model is a good fit for mobile and edge devices in terms of the model's data, hardware processing requirements, and total size and complexity.

The experiment workflow ignores the classic optimization techniques like quantization and pruning before the conversion to avoid further test accuracy degradation. (i.e., the testing accuracy reported in the previous section for both models remains the same even after the TFLite conversion due to the absence of lossy compression-optimization techniques). The inference latency results show that the TFLite version of the parent model took 3310.49 s, and the TFLite - LCRM generated model took only 1230.91 s to infer 10000 test images on the selected IoT device. This outcome implies that the proposed optimization technique could reduce the end device inference latency up to 62.82%.

9) MEMORY FOOTPRINT

We summarize the memory requirement for storing different model formats like KerasH5 format, its compressed format, and TFLite versions in the Table 8. A helper-zip function in python compresses the parent and LCRM-generated models in saved kerasH5 format to generate the compressed (zipped) version. TFLite versions are essential when the original TensorFlow models are incompatible with resource-constrained devices, as many end devices do not support TensorFlow but support TFLite. The result from Table [46] depicts a reduction of 30% in the LCRM-generated TFLite version while comparing it with the TFLite version of the parent model.

TABLE 8. Memory footprint comparison.

Models	KerasH5 format	Compressed file	Tf Lite
Parent model	130.78 MB	41.55 MB	43.54 MB
LCRM model	91.39 MB	29.06 MB	30.41 MB

TABLE 9. Comparison between Decom-Net and LCRM generated Decom-Net for image enhancement application.

Performance metrics	Parent model(DecomNet)	LCRM-DecomNet
SSIM ↑	0.17	0.17
LPIPS ↓	0.25	0.26
NIQE ↓	5.37	5.35
PIQE ↓	11.27	11.8
BRISQUE ↓	21.05	18.12
# of parameters	0.19M	0.028M
Model size	4.33 MB	1.26 MB
Epochs trained	1800	1800

D. LCRM PERFORMANCE ANALYSIS ON DIFFERENT CNNs, APPLICATIONS AND DATASETS

This section explores the effectiveness of the LCRM algorithm on three different CNN models U-Net [44], VGG-9 [3] and Decomposition-Net [42], performing different image processing tasks such as image segmentation, image enhancement, and image classification, respectively, on the following datasets: Data Science Bowl [43], LOL [42], and EMNIST [41]. This analysis aims to assess the suitability of the LCRM algorithm for various image-processing tasks and determine its efficacy across different datasets and CNN models. This study will provide insights into the strengths and weaknesses of the LCRM algorithm and help identify potential areas for improvement.

1) IMAGE ENHANCEMENT

Image enhancement is the process of improving the visual quality of an image using various algorithms. It enhances image features, reduces noise and artifacts, and improves contrast and brightness. It is widely used in various medical imaging, surveillance, and photography applications. Here we consider poorly illuminated image enhancement using a Decomposition network from Retinex-Net (parent model) and its corresponding LCRM-optimized decomposition net called LCRM-DecomNet. We then evaluate both models on LOL (LOw-Light) dataset based on various performance metrics, including reference metrics like structure similarity index(SSIM) [47], learned perceptual image patch similarity (LPIPS) [48] and no-reference metrics like natural image quality evaluator (NIQE) [49], perception based image quality evaluator (PIQE) [50] and blind/referenceless image spatial quality evaluator (BRISQUE) [51].

Table 9 compares the original Decom-Net model and LCRM-DecomNet for an image enhancement application. It is evident that both models achieved similar results regarding SSIM and LPIPS, which are the common metric to measure the similarity between the original and enhanced images. LCRM-DecomNet achieved slightly better BRISQUE PIQE, and NIQE scores, indicating that it produced slightly better perceived and natural image quality, as seen in Figure 8.

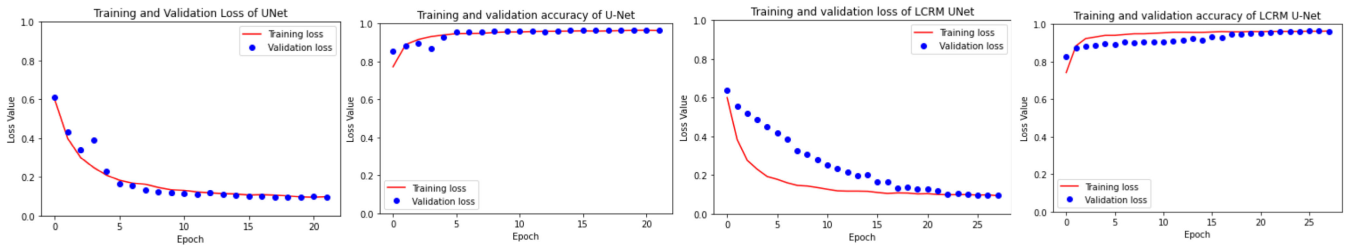


FIGURE 6. Training, validation loss and accuracy of U-Net and LCRM-UNet.

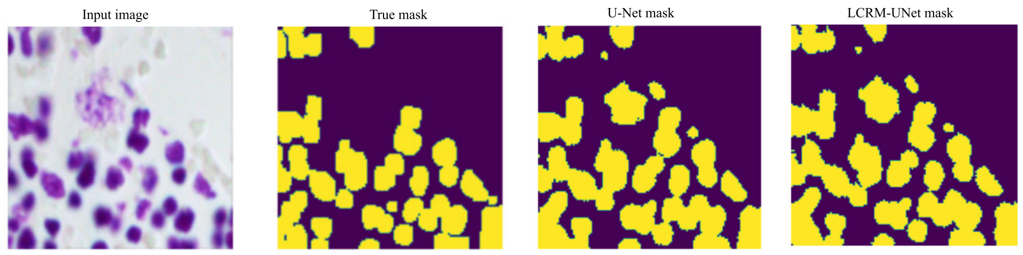


FIGURE 7. The segmentation output from U-Net and LCRM-UNet.

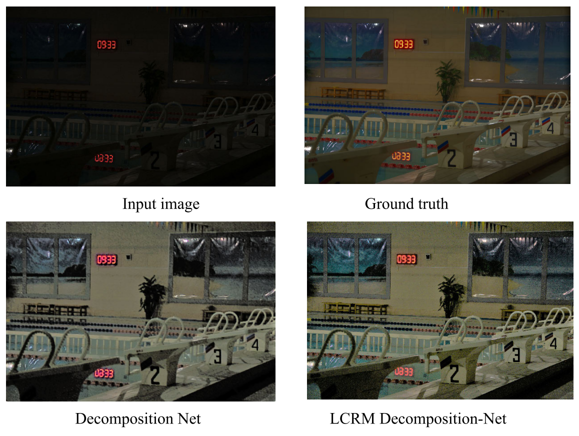


FIGURE 8. Image enhancement output from DecomNet and LCRM-DecomNet.

TABLE 10. Comparison between U-Net and LCRM generated U-Net for segmentation application.

Performance metrics	Parent model (U-Net)	LCRM-UNet
Accuracy	0.96	0.96
Point of convergence	22 nd epoch	28th epoch
Intersection over union(IoU)	0.854	0.842
Inference latency	27.8 ms	22.4 ms
Number of parameters	1.4 M	0.304M
Model size	16.2 MB	3.69 MB

Furthermore, LCRM-DecomNet had significantly fewer parameters and a smaller model size than the parent model, indicating its superior efficiency. Overall, LCRM-DecomNet is a promising modification of the original Decom-Net model, as it achieves similar SSIM and LPIPS, better NIQE and PIQE scores, and is more efficient in terms of parameters and model size.

2) IMAGE SEGMENTATION

Image segmentation is a method for partitioning an image into multiple segments or regions, each representing a meaningful part of the image. One of the approaches for tackling image segmentation is U-Net [44]. It is a CNN with an encoder-decoder structure and skips connections that preserve spatial information. We use the Kaggle Data Science Bowl dataset to evaluate the U-Net, which contains many nuclei images for segmentation [43]. The parent model U-Net is then Optimized using the LCRM algorithm to generate LCRM-UNet. We have trained both models from scratch with the same hyperparameter settings. Figure 6 shows the training and testing accuracy and loss.

The results and comparison between the original U-Net model and the LCRM-UNet for the segmentation application are in the Table: 10. The comparison is based on performance metrics, including accuracy, point of convergence, intersection over union (IoU), inference latency, number of parameters, and model size. Both models achieved the same accuracy of 0.96, but LCRM-UNet achieved this with significantly fewer parameters and a smaller model size than the parent model. LCRM-UNet had a lower inference latency of 22.4 ms/step, meaning it could perform the segmentation task faster than the parent model. Sample segmentations obtained from both models, along with ground truth and the input image, are given in Figure 7

However, the point of convergence for LCRM-UNet was later than the parent model, meaning it took longer to train the modified model to reach the desired level of accuracy. Since each epoch takes only 42 seconds for LCRM-UNet, the time required to prepare different epochs are negligible. Overall, LCRM-UNet is a promising modification of the original U-Net model, as it achieves similar levels of accuracy with

TABLE 11. Comparison between VGG-Net and LCRM generated VGG- Net for OCR application.

Performance metrics	Parent model(VGG9)	LCRM-VGG9
Accuracy	0.87	0.87
Point of conv.	5th epoch	10th epoch
Loss value	0.35	0.36
Inference latency	82 ms	66 ms
# of parameters	7.9 M	3.65 M
Model size	78.4 MB	30.4 MB

fewer parameters and smaller model sizes while also being faster at performing the segmentation task.

3) OPTICAL CHARACTER RECOGNITION

Optical character recognition (OCR) is a technique that converts scanned documents, images, and other digital files into searchable and editable text. Pattern recognition algorithms and machine learning techniques can efficiently identify and extract text from images or documents. Its applications are in publishing, healthcare, finance, and government industries. It is also used in automated systems such as self-driving cars and passport control systems.

A system needs optimized versions of existing algorithms to deploy such applications efficiently to a resource-constrained environment. Hence, for evaluating the proposed model, we consider VGG-9 and its optimized LCRM-VGG9 with the EMNIST dataset, which is then trained from scratch keeping all the hyperparameters intact. The models are then tested and evaluated based on different performance metrics, summarized in Table 11.

The Table shows that both models achieved the same accuracy of 0.87, which indicates that LCRM does not negatively affect model accuracy. However, the LCRM-generated VGG-Net model converged later than the parent VGG-Net model, with the point of convergence occurring at the 10th epoch compared to the 5th epoch of the parent model. The loss value of the LCRM-VGG9 model (0.36) is slightly higher than that of the parent VGG-Net model (0.35), and the difference is relatively small. It also has a faster inference latency of 66 ms than the parent VGG-Net model's 82 ms. This performance indicates that the LCRM-generated model is more efficient in processing data. The LCRM-VGG9 model has fewer parameters (3.65 M, 30.4 MB) than the parent VGG-Net model (7.9 M, 78.4 MB) and is more compact, and requires less memory and computation.

In summary, the LCRM-generated VGG-Net model performs similarly or better than the parent VGG-Net model in terms of accuracy, with the added benefits of lower inference latency, fewer parameters, and smaller model size. However, the LCRM-generated model takes a little longer epochs to converge and reach optimal value.

E. COMPARISON WITH THE STATE-OF-THE-ART OPTIMIZATION TECHNIQUES

To study the impact of the State-of-the-art (SOTA) optimization techniques like individual and cascaded optimization methods, we apply them to the parent model for evaluation and comparison [52]. The individual techniques include (i) quantization (Q), which reduces the precision of weights, (ii) Pruning (P), which sets weights to zero after a certain threshold, and (iii) weight clustering (C), which clusters similar weights together for better compression, similar to the deep compression method [20].

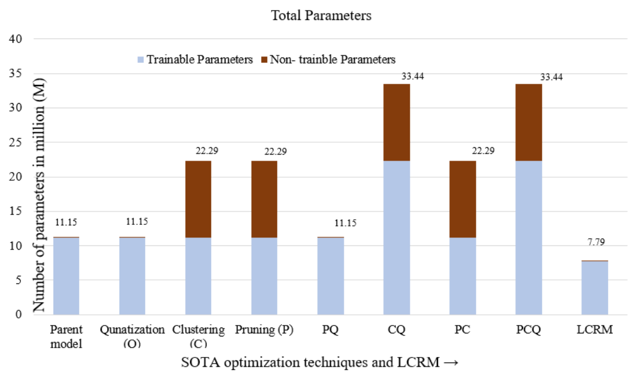


FIGURE 9. Comparison of the total parameters w.r.t the SOTA techniques.

TABLE 12. SOTA techniques and their corresponding evaluation metrics on Raspberry Pi B.

Optimization Techniques	Epoch Saturation	TFLite Accuracy	Inference Latency
Base model [2]	25	0.81	0.331
Clustering-C [20]	25+5	0.81	0.33
Pruning-P [54]	25+5	0.81	0.328
Quantization-Q [55]	25+5	0.81	0.299
P preserving Q (PQ) [53]	25+5	0.81	0.294
C preserving Q (CQ)[53]	25+1	0.83	0.266
P preserving C (PC) [53]	25+1	0.83	0.26
P,C preserving Q (PCQ) [53]	25+1	0.83	0.256
LCRM	45	0.83	0.123

The cascaded optimization techniques combine the classic techniques designed without nullifying each other's purposes. Examples include (iv) sparsity preserving clustering (PC), (v) sparsity preserving quantization (PQ), (vii) cluster preserving quantization (CQ), and (viii) sparsity and cluster preserving quantization (PCQ) [53]. The following subsection contains a detailed study of the optimization techniques and their comparison with the parent and LCRM-generated models. Two evaluation metrics for comparing the results are (I) performance metrics like inference latency and accuracy and computational complexity of the model and (II) resource utilization metrics like energy consumption during the inference, memory utilization, and CPU utilization.

1) PERFORMANCE METRICS

Trainable and non-trainable parameters: The total parameters of a CNN model consist of trainable and non-trainable parameters for different optimization techniques, as shown in Fig: 9.

The extra parameters present in C, P, PC, CQ, and PCQ are the non-trainable parameters of the clustering and pruning algorithms for calculating centroid values, indices values, or sparsity calculations. Quantization does not introduce any

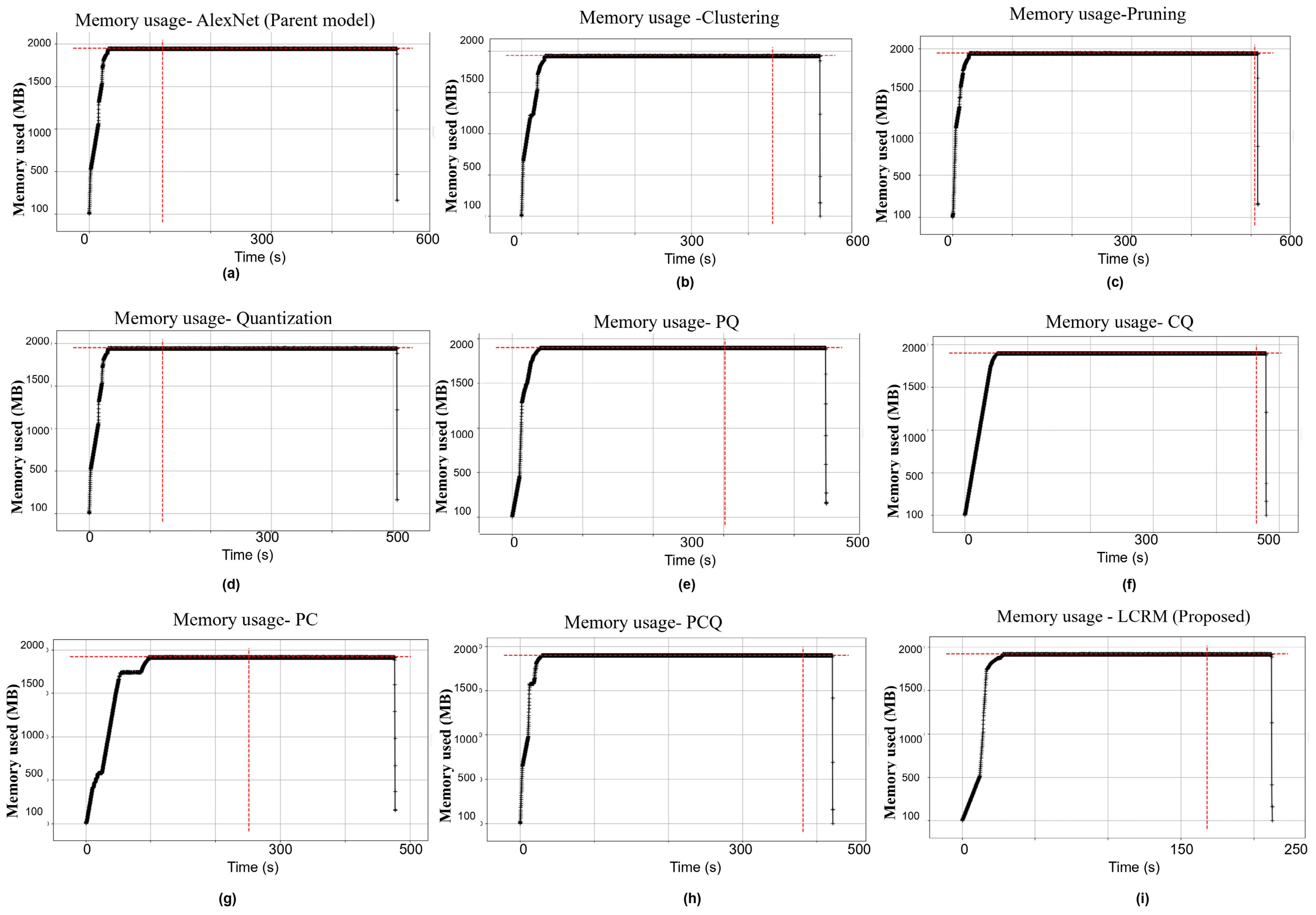


FIGURE 10. Memory utilization of Raspberry pi with (a) parent model, (b) Clustering-C, (c) Pruning-P, (d) Quantization-Q, (e) P preserving Q (PQ), (f) C preserving Q (CQ), (g) P preserving C (PC), (h) P, C preserving Q (PCQ) and (i) LCRM.

non-trainable parameters. In contrast, pruning and clustering, along with cascading the optimization techniques, increase trainable and non-trainable parameters, as seen in PCQ, resulting in the increased complexity of the model. However, the LCRM model has few non-trainable parameters, and the complexity and multiplication reductions are evident due to the involvement of substitutional layers.

Accuracy and Inference Latency : the TFlite accuracy of the models, the point at which they converge (epoch saturation) and their corresponding inference latency are given in Table 12. It shows that the cascaded optimization techniques (PQ, CQ, PC, PCQ) have better accuracy than the individual optimizing method (P, Cmodels'However, the LCRM method has comparable accuracy and excellent inference latency metrics compared to the existing classical optimization models. The latency is reduced up to 62.84% when compared to the parent model. In other words, The parent model requires 0.331 seconds to classify a single image, whereas the LCRM-optimized parent model requires only 0.123 seconds to do the same task.

2) RESOURCE UTILIZATION METRICS

The experiment involving CPU and memory utilization consists of 9 cases: the parent model, 7 SOTA techniques, and the LCRM-generated model. Fig: 10 represents the estimation during the inference of 1500 test images of the CIFAR-10 dataset during the first 600 seconds. The parent model and other compared models require CPU utilization between 68% and 72%, and the LCRM-optimized parent model only requires 45%.

Similarly, Fig: 11 represents the memory utilization of SOTA models and the proposed model with respect to the time of the execution of the first 600 seconds. The maximum memory requirement depends on the dataset used for the inference; hence, the values are almost similar, ranging from 1915.2 MB (LCRM) to 1942.7 (Parent model).

The inference energy consumption is much less due to the reduction in complexity and multiplications existing in the optimized version compared to the SOTA models. Fig: 12 shows the graphical representation of energy consumption per epoch against each SOTA technique.

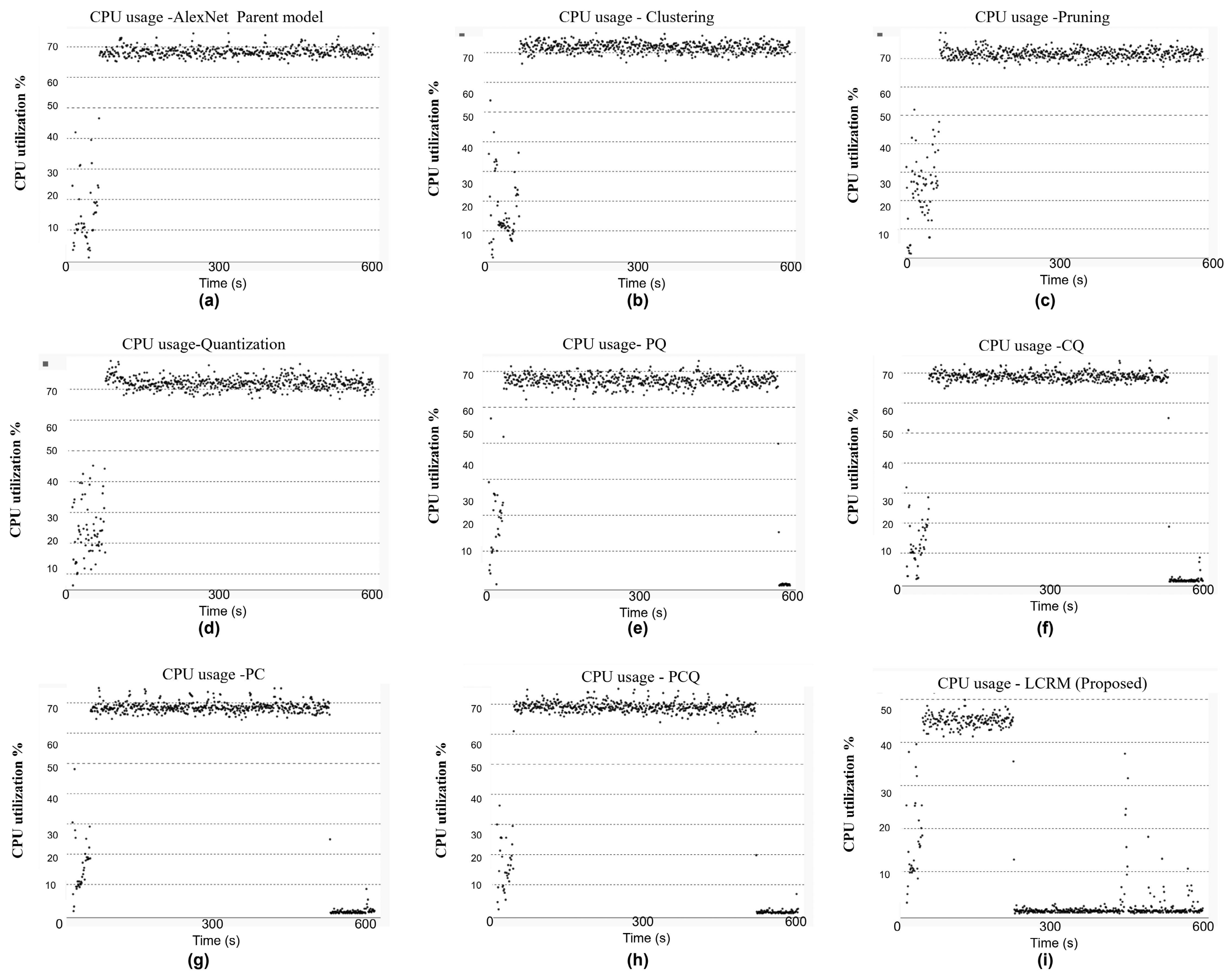


FIGURE 11. **CPU utilization of Raspberry pi with (a) parent model, (b) Clustering-C, (c) Pruning-P, (d) Quantization-Q, (e) P preserving Q (PQ), (f) C preserving Q (CQ), (g) P preserving C (PC), (h) P, C preserving Q (PCQ) and (i) LCRM, with respect to time (x-axis, 600 seconds and y-axis, CPU utilization %).**

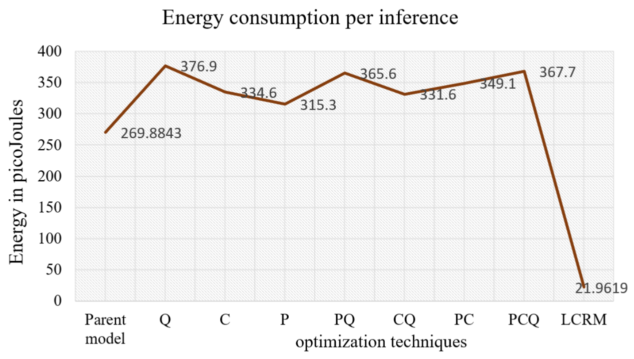


FIGURE 12. **Comparison of energy consumption per inference w.r.t the SOTA techniques.**

F. ABLATION STUDY

Ablation study evaluates the importance of different components in a model, such as layers, hyperparameters, or optimization techniques. By removing or disabling specific components, researchers can assess how they affect the model's performance, such as accuracy, loss, or inference time. This information can be used to optimize the model and improve its efficiency and effectiveness.

In this section, we conducted the ablation study on four scenarios of the LCRM algorithm on ALexNet and the LCRM-AlexNet optimized model. All five models are trained to 30 epochs, keeping the same hyperparameters such as loss function: SCE (sparse categorical entropy), optimization algorithm (Adam) and learning rate(0.0001). The test accuracy and loss results are in fig, 13. We also evaluate inference latency on the deployed GPU, the complexity of each model (as the number of parameters), and the memory required to store the model given in Table 13.

Scenario 1 (Upscaling When N>M, M%N=0): When the Scenario 1 module was disabled, the accuracy dropped from 0.802 to 0.71, and the loss function increased from

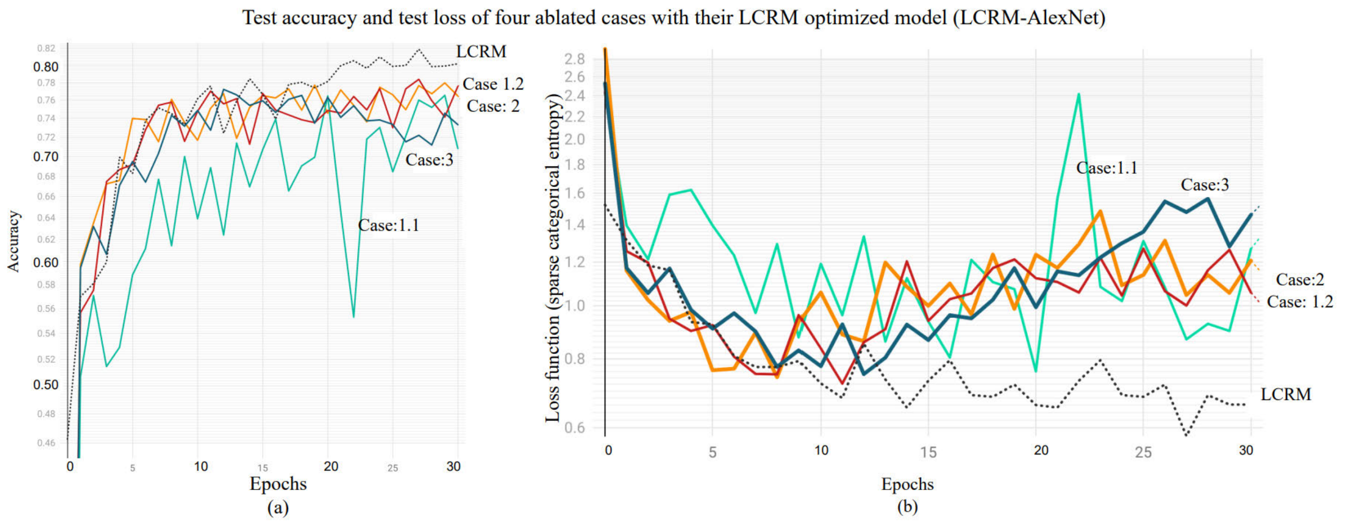


FIGURE 13. Test accuracy (a) and test loss (b) of ablation experiment for case:1.1 & 1.2 (Upscaling), Case 2: (no-scaling) and case 3: (Downscaling).

TABLE 13. Ablation study of different cases of LCRM deployed on Tesla K80 at batch-size 128.

Perf. metrics	Case 1.1	Case 1.2	Case 2	Case 3	LCRM
Accuracy	0.71	0.77	0.76	0.73	0.802
Loss fn (SCE)	1.27	1.06	1.21	1.46	0.66
Inf. latency	48ms	63ms	61ms	59ms	42ms
# parameter	8.19M	10.21M	11.47M	8.6M	7.7M
Size(MB)	87.7	108	121	91.6	83.3

0.66 to 1.27. The size and number of parameters were reduced slightly. This reduction suggests that Scenario 1 plays a vital role in improving the accuracy and reducing the loss of the LCRM algorithm and a minor role in contributing to the complexity.

Scenario 2 (Upscaling When N>M, M%N≠ 0): When the Scenario 2 module was removed, the accuracy dropped from 0.802 to 0.77, and the loss function increased from 0.66 to 1.06. This drop in accuracy and increase in loss function implies that Scenario 2 has a less significant impact on accuracy than Scenario 1, but it still helps to reduce the loss function. However, the number of parameters and size increased drastically, showing that Scenario 2 convolution modules are good at optimizing the overall complexity of the model.

Scenario 3 (No Scaling, N==M): When the Scenario 3 module was disabled, the accuracy dropped from 0.802 to 0.76, and the loss function increased from 0.66 to 1.21. This change in performance suggests that Scenario 3 significantly improves the accuracy and reduces the loss of the LCRM algorithm. Similar to Scenario 2, the model size and parameters have increased significantly.

Scenario 4 (Downscaling When M>N): When the Scenario 4 module was removed, the accuracy dropped from 0.802 to 0.73, and the loss function increased from 0.66 to 1.46. This change in performance indicates that Case 3 has a good impact on the accuracy and loss function of the LCRM algorithm.

Overall, the results of the ablation study suggest that all three cases play essential roles in improving the accuracy, reducing the loss, model size, parameters, and inference latency of the LCRM algorithm. However, the importance of each case may vary depending on the specific application or use case.

V. INFERENCES

In this section, we answer the four research questions of this paper based on the findings from the experiment evaluation. The RQs and answers are summarized below:

RQ1: What are the impacts of computational complexity while applying LCRM to different CNN models, and how do the optimized models result in a noticeable reduction in the number of parameters and floating point operations?

The application of LCRM on different parent CNN models resulted in a considerable decrease in the number of parameters and floating-point operations or multiplications, the size of the models, and inference latency. Table 4 shows the parameter reduction of both models and Table 2 shows the decrease in the number of multiplications for all 4 cases of the proposed algorithm in the image classification application.

The reduction in computation and complexity is also seen in different LCRM-CNNs and their applications that we studied in Table 10 for image segmentation applications. Similarly, in Tables 9 and 11 for image enhancement and optical character recognition applications, respectively.

RQ2: What are the performance impacts of training the LCRM-generated models?

We have evaluated training efficiency using performance metrics like throughput, accuracy, peak memory consumption, GPU energy consumption, and GPU utilization. The results show that the LCRM-generated model's throughput has increased by 5–6 times, energy consumption per epoch has reduced by 82% - 85%, and peak memory consumption by 25% - 34%. A significant limitation found was the reduced

GPU utilization profile during the execution of the proposed algorithm at a smaller batch size. Even though larger batch sizes give good GPU utilization, they also elevate PMC. Thus, a performance trade-off exists between batch sizes, PMC, and GPU utilization.

However, When LCRM is applied to any CNN model, the time taken to converge the algorithm while training is a little higher when compared to the parent model.

RQ3: How do the LCRM-generated models affect the inference efficiency?

We have considered metrics like inference latency, energy consumption per epoch, testing accuracy, and memory footprints to evaluate inference efficiency. In NVIDIA RX1650 GPU, The inference latency and energy consumption per inference have reduced to 94%-95% and 91% to 93%, respectively, compared to the parent model. The memory footprint of TensorFlow models has been reduced to 30% to make them suitable for end device deployment without compromising the overall testing accuracy.

We have also tested the feasibility of the optimization technique on Raspberry pi 4, and the results show that the TFLite version of the LCRM optimized model works well when deployed on the above device (i.e., there is a reduction of 62.84% in inference latency concerning the parent model). The results also show a reduction in CPU utilization and memory utilization.

The accuracy of various applications studied in this paper remained the same despite a considerable decrease in the computations, size and inference requirements, as seen in Table 10 for image segmentation application. Similarly, in Tables 9 and 11 for image enhancement and optical character recognition application, respectively.

RQ4: How does the LCRM technique outperform the existing state-of-the-art optimization techniques?

Out of the seven SOTA optimization techniques under section IV-D, the LCRM attains significantly fewer trainable parameters. It also outperforms in terms of reduced inference latency (62.84%) at lower energy consumption per epoch, memory, and CPU utilization with a comparable testing accuracy. Hence the end devices can use the LCRM-optimized models for performing time-critical applications where the speed of the model execution and resources matter.

1) ADVANTAGES OF THE LCRM FRAMEWORK

Two significant advantages of the proposed method are:

1:The LCRM-generated CNN optimization models can significantly reduce inference latency and resource requirement of parent CNNs, resulting in the faster and more efficient data processing. This gain makes the LCRM-optimized models well-suited for end-device deployment with limited computational and hardware resources. The experiments conducted on the state-of-the-art techniques in the section IV-E with the LCRM-generated model show that the proposed model results in reduced inference latency, better accuracy, and CPU utilization.

TABLE 14. Comparison between parent models and LCRM-generated models in terms of point of convergence (Epochs).

Application (CNN)	Parent model	LCRM-model
Image classification (AlexNet)	25 ep	45 ep
Image enhancement (Decomp.Net)	1800 ep	1800 ep
Image segementation (U-Net)	22 ep	28 ep
OCR (VGG-9)	5ep	10 ep

2: Another significant advantage of the LCRM-optimized models is their ability to reduce memory requirements to store the saved model compared to the parent model. A lesser memory requirement benefits applications such as mobile and edge computing environments. Additionally, the reduced memory requirements and complexity are optimized without compromising the model's overall performance. The results of experiments on various applications using CNN models generated by LCRM, as presented in Section IV-D, demonstrate that the proposed framework achieves reduced complexity by decreasing the number of parameters. This reduction in parameters leads to lower memory requirements while maintaining performance metrics.

2) LIMITATIONS OF THE LCRM FRAMEWORK

Two limitations of the proposed method are:

1: The LCRM-generate models require more training epochs to converge compared to the parent model due to their reduction in training parameters. The epochs required to converge both parent and LCRM-generated models are summarized in Table 14.

We also conclude that the number of epochs used for training the parent and LCRM models may vary depending on several factors, such as the model's complexity, the dataset's size, the availability of computational resources, and the desired level of performance.

2: Another limitation is that the optimized LCRM models are not capable of on-device training and are designed to be trained using a GPU. The minimum GPU requirement for training the model is 4 GB, meaning that users should ensure they have a compatible GPU before attempting to train it. While this may present a challenge for some users, it is essential to note that the optimized LCRM models offer significant improvements over parent models and are well worth the investment in time and resources. By setting up and training the model correctly, users can expect impressive results in different image processing tasks and obtain optimized models that can be executed on the end devices. We aim to address these limitations in our future research work.

VI. CONCLUSION AND FUTURE SCOPE

This paper proposes a layer-wise complexity reduction method (LCRM) for optimizing CNN models. The critical designs included are the LCRM algorithm, architecture, parameter estimation, complexity calculations, and the profiling of performance analysis. To our knowledge, this paper is the first work that proposes a layer-wise complexity reduction and optimization for accuracy-oriented CNN models (standard CNN models) that aims to: (i) reduce computational complexity, (ii) optimize hardware resource utilization, and (iii) study training and testing efficiencies, with respect to resource utilization and other performance metrics.

The feasibility of the proposed framework was tested on four different applications: image classification, image segmentation, image enhancement, and optical character recognition. Experimental results show that the proposed framework reduces inference latency, model compression, and parameter reduction without compromising accuracy for all tested CNNs, namely AlexNet, U-Net, VGG-9 and Retinex-Net (Decomposition-Net). The proposed framework is then tested with state-of-the-art techniques like pruning, quantization, clustering and their cascaded models by deploying them on an end device to show the proposed model's efficacy in latency reduction and hardware utilization.

Although there are a few limitations in the framework, such as the requirement for extra training epochs and the inability to perform on-device learning, these shortcomings can be addressed by incorporating the future directions of research such as:

(a) Introducing on-device training: One way to overcome the inability to perform on-device learning is by integrating methodologies like federated learning, incremental training, and accelerator arrays [16].
(b) Cascaded Optimization and resource utilization: To address the requirement for extra training epochs, the framework can incorporate advanced cascaded optimization techniques such as ensemble PCQ to improve resource utilization.
(c) Hardware and software co-design: Exploring hardware-aware designs for deep learning model optimizations holds promise and requires further effort to succeed [56]. By focusing on this aspect, the framework can leverage hardware capabilities and enhance its performance.

By incorporating these future directions of research, the limitations of the framework can be effectively balanced, leading to improved performance and efficiency.

APPENDIX A THE SELECTION OF SUBSTITUTIONAL LAYERS

Grid search is a process that searches exhaustively through a specified subset of the hyperparameter space of the targeted algorithm. For example, to search for an optimal learning algorithm of a CNN model, the subgroup has choices ranging from SGD, RMSprop, Adagrad, Adam, and Nadam. The search algorithm returns Adam as the best option from the subset when it attains better accuracy than others.

Similarly, the search algorithm returns the best substitutional layers for the LCRM algorithm from the subset of available convolution layers in the literature. The layers include standard convolution (Std Conv), transpose convolution (TrC), Groupwise convolution (GrC), depthwise convolution (DWC), pointwise convolution (PWC), and dilated convolution (DiC). The returned structures of the CNN models are in both series (single type of convolution) and parallel (combination of different convolution). The Table 15, included only the models from the above combination, with less than 15% accuracy reduction than the parent model. The experimental results from the Table also show a trade-off between the number of parameters and accuracy reduction. For example, in models 2,3,5 and 7, there is a reduction in parameters by 32%, resulting in an accuracy reduction. However, accuracy may or may not decrease when the parameter remains unchanged (e.g., models 1, 4, and 6).

As the objective of this paper is to reduce the number of parameters of CNN models using substitutional layers, only three models: (i) Group convolution (when the number of groups is G=32), (ii) Group convolution+ dilated convolution (when G=32 and dilation factor=2) and (iii) depthwise separable and pointwise convolution, met the condition. The rationale behind selecting the best model among the three is the percentage accuracy reduction, as seen in Table 15.

Models 2, 3, and 7 give a 31.25% reduction in the total number of parameters at 11.45%, 14.99%, and 0.2% reduction in accuracy, respectively. Hence, the substitutional convolution layers of model 7 (Depthwise separable and

TABLE 15. Comparison between different substitutional convolutions, number of parameters, type of architecture and change in accuracy with respect to parent model.

Model	Substitution layers	Architecture	Param.	% accuracy at (ep.24)
Model 0 (Base)	Std Conv	Serial	11.2 M	0.801
Model 1	DiC only	Serial	11.2 M	0.785
Model 2	GrC only	Serial	**7.7 M**	0.664
Model 3	PWC only	Serial	8 M	0.702
Model 4	TrC only	Serial	11.2 M	0.76
Model 5	GrC+ DiC	Parallel	**7.7 M**	0.633
Model 6	TrC + DiC	Parallel	11.2	0.764
Model 7	DWS+ PWC	Parallel	**7.7 M**	0.781

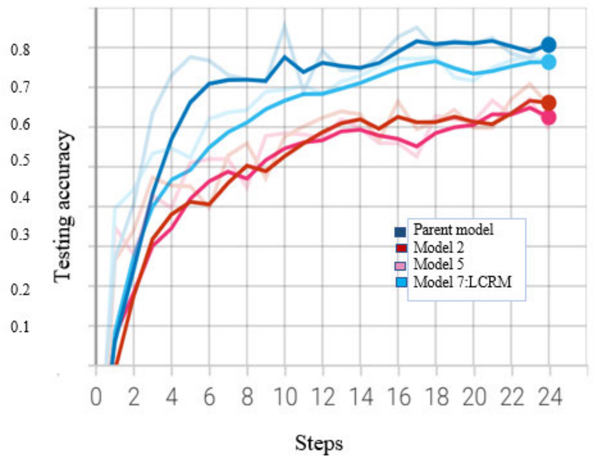


FIGURE 14. Test accuracy of the parent model and other three parameter reduced models.

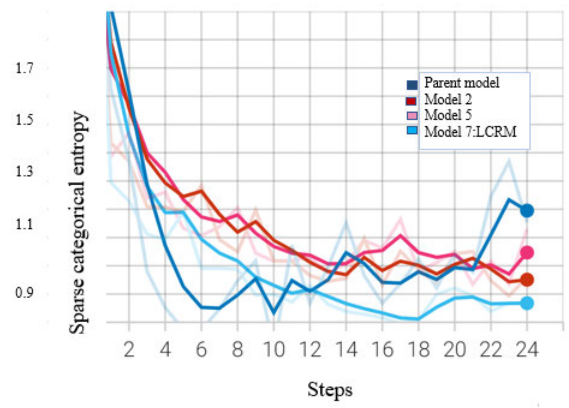


FIGURE 15. Test loss of the parent model and other parameter-reduced models.

TABLE 16. Hyperparameter results obtained from grid search.

Hyperparameter	Optimal choice	Choices given
Weight initializer	uniform	uniform, glorot_normal, glorot_uniform, he_normal, he_uniform
Batch size	128	16,32,64,128,256
Optimization algorithm	Adam	SGD, RMSprop, Adagrad, Adam, and Nadam
Learning rate	0.0001	0.01,0.001,0.0001
Activation function	ReLu	ReLu, tanh, sigmoid, softmax, linear

pointwise convolution layers) are optimal according to the experiments.

Additional information on the testing accuracy in Figure 14 and the loss function (sparse categorical entropy- SCE) in Figure 15 of all the three models, along with the parent model, are plotted for comparison with the help of the tensorboard. After finalizing the substitutional layers, a hyperparameter grid search is conducted to get the optimal results for the learning algorithm, learning rate, weight initialization, activation function, and batch size. The result from the hyperparameter search of the LCRM model is given in Table 16.

ACKNOWLEDGMENT

The work of Hanan Hussain was supported by the Ph.D. Research Fellowship with BITS Pilani, Dubai Campus. The authors would like to thank the Creative Laboratory and the Faculty Member, specifically Dr. Pranav Pawar with BITS Pilani, Dubai Campus, for their generous provision of the GPU and end devices used in our LCRM-CNN optimization experiments.

An earlier version of this paper was presented at the 20th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys '22), Boston, MA, USA, [DOI: 10.1145/3560905.3568299].

REFERENCES

[1] H. Hussain, P. S. Tamizharasan, and C. S. Rahul, "Design possibilities and challenges of DNN models: A review on the perspective of end devices," *Artif. Intell. Rev.*, vol. 55, no. 7, pp. 5109–5167, Oct. 2022, doi: 10.1007/s10462-022-10138-z.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

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

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269.

[7] V. Gupta, V. G. Panchal, V. Singh, D. Bansal, and P. Garg, "EmotionNet: ResNeXt inspired CNN architecture for emotion analysis on raspberry Pi," in *Proc. Int. Conf. Recent Trends Electron., Inf., Commun. Technol. (RTEICT)*, Aug. 2021, pp. 262–267.

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," in *Proc. CVPR*, 2016, pp. 1–13.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[10] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[11] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2019, *arXiv:1905.11946*.

[12] D. S. Joseph, P. M. Pawar, and R. Pramanik, "Intelligent plant disease diagnosis using convolutional neural network: A review," *Multimedia Tools Appl.*, vol. 82, pp. 21415–21481, Oct. 2022.

[13] T. Arias-Vergara, P. Klumpp, J. C. Vasquez-Correa, E. Nöth, J. R. Orozco-Arroyave, and M. Schuster, "Multi-channel spectrograms for speech processing applications using deep learning methods," *Pattern Anal. Appl.*, vol. 24, no. 2, pp. 423–431, May 2021.

[14] N. Lopac, F. Hržic, I. P. Vuksanovic, and J. Lerga, "Detection of non-stationary GW signals in high noise from Cohen's class of time-frequency representations using deep learning," *IEEE Access*, vol. 10, pp. 2408–2428, 2022.

[15] S. Mahadik, P. M. Pawar, and R. Muthalagu, "Efficient intelligent intrusion detection system for heterogeneous Internet of Things (HetIoT)," *J. Netw. Syst. Manag.*, vol. 31, no. 1, p. 2, Jan. 2023.

[16] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "On-device machine learning: An algorithms and learning theory perspective," 2019, *arXiv:1911.00623*.

[17] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Comput. Surv.*, vol. 55, pp. 1–37, 2021.

[18] B. Wang, F. Ma, L. Ge, H. Ma, H. Wang, and M. A. Mohamed, "Icing-EdgeNet: A pruning lightweight edge intelligent method of discriminative driving channel for ice thickness of transmission lines," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–12, 2021.

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "TensorFlow: A system for large-scale machine learning," 2016, *arXiv:1605.08695*.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," 2016, *arXiv:1510.00149*.

[21] Y. Wang, C. Xu, C. Xu, and D. Tao, "Packing convolutional neural networks in the frequency domain," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2495–2510, Oct. 2019.

[22] A. S. Aguiar, F. N. D. Santos, A. J. M. De Sousa, P. M. Oliveira, and L. C. Santos, "Visual trunk detection using transfer learning and a deep learning-based coprocessor," *IEEE Access*, vol. 8, pp. 77308–77320, 2020.

[23] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[24] L. Zahedi, F. G. Mohammadi, S. Rezapour, M. W. Ohland, and M. H. Amini, "Search algorithms for automated hyper-parameter tuning," 2021, *arXiv:2104.14677*.

[25] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," 2020, *arXiv:2003.05689*.
[26] C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. ECCV*, 2018, pp. 19–34.
[27] J. Liu, J. Liu, W. Du, and D. Li, "Performance analysis and characterization of training deep learning models on mobile device," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2019, pp. 506–515.
[28] A. A. Suzen, B. Duman, and B. Sen, "Benchmark analysis of Jetson TX2, Jetson nano and raspberry PI using deep-CNN," in *Proc. Int. Congr. Human-Comput. Interact., Optim. Robotic Appl. (HORA)*, Jun. 2020, pp. 1–5.
[29] Y. Wang, Q. Wang, S. Shi, X. He, Z. Tang, K. Zhao, and X. Chu, "Benchmarking the performance and power of AI accelerators for AI training," 2019, *arXiv:1909.06842*.
[30] I. Paul, W. Huang, M. Arora, and S. Yalamanchili, "Harmonia: Balancing compute and memory power in high-performance GPUs," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 54–65.
[31] C. A. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. D. Bailis, K. Olukotun, C. Re, and M. A. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, p. 102, 2017.
[32] A. Shah, C.-Y. Wu, J. Mohan, V. Chidambaram, and P. Krähenbuhl, "Memory optimization for deep networks," 2020, *arXiv:2010.14501*.
[33] Y. Ren, S. Yoo, and A. Hoisie, "Performance analysis of deep learning workloads on leading-edge systems," in *Proc. IEEE/ACM Perform. Model., Benchmarking Simul. High Perform. Comput. Syst. (PMBS)*, Nov. 2019, pp. 103–113.
[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[35] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022.
[36] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[37] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proc. 9th ACM Multimedia Syst. Conf.*, Jun. 2018, pp. 204–215.
[38] A. Agrawal and N. Mittal, "Using CNN for facial expression recognition: A study of the effects of kernel size and number of filters on accuracy," *Vis. Comput.*, vol. 36, no. 2, pp. 405–412, Feb. 2020.
[39] C. Yao, W. Liu, W. Tang, J. Guo, S. Hu, Y. Lu, and W. Jiang, "Evaluating and analyzing the energy efficiency of CNN inference on high-performance GPU," *Concurrency Comput., Pract. Exper.*, vol. 33, no. 6, p. e6064, Mar. 2021.
[40] X. Tang and Z. Fu, "CPU–GPU utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems," *IEEE Access*, vol. 8, pp. 58948–58958, 2020.
[41] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: An extension of MNIST to handwritten letters," 2017, *arXiv:1702.05373*.
[42] C. Wei, W. Wang, W. Yang, and J. Liu, "Deep Retinex decomposition for low-light enhancement," 2018, *arXiv:1808.04560*.
[43] J. C. Caicedo, A. Goodman, K. W. Karhohs, B. A. Cimini, J. Ackerman, M. Haghighi, C. Heng, T. Becker, M. Doan, C. McQuin, M. Rohban, S. Singh, and A. E. Carpenter, "Nucleus segmentation across imaging experiments: The 2018 data science bowl," *Nature Methods*, vol. 16, no. 12, pp. 1247–1253, Dec. 2019.
[44] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," 2015, *arXiv:1505.04597*.
[45] G. Lu, W. Zhang, and Z. Wang, "Optimizing depthwise separable convolution operations on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 70–87, Jan. 2022.
[46] B. Pal and S. Khaiyum, "Low memory footprint CNN models for end-to-end driving of autonomous ground vehicle and custom adaptation to various road conditions," *Int. J. Innov. Technol. Exploring Eng.*, vol. 9, no. 1, pp. 3252–3259, Nov. 2019.
[47] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.
[48] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018, *arXiv:1801.03924*.
[49] A. Mittal, R. Soundararajan, and A. C. Bovik, "Making a 'completely blind' image quality analyzer," *IEEE Signal Process. Lett.*, vol. 20, no. 3, pp. 209–212, Mar. 2013.
[50] N. Venkatanath, D. Praneeth, M. C. Bh, S. S. Channappayya, and S. S. Medasani, "Blind image quality evaluation using perception based features," in *Proc. 21st Nat. Conf. Commun. (NCC)*, Feb. 2015, pp. 1–6.
[51] A. Mittal, A. K. Moorthy, and A. C. Bovik, "No-reference image quality assessment in the spatial domain," *IEEE Trans. Image Process.*, vol. 21, no. 12, pp. 4695–4708, Dec. 2012.
[52] H. Hussain and P. S. Tamizharasan, "The impact of cascaded optimizations in CNN models and end-device deployment," in *Proc. 20th ACM Conf. Embedded Networked Sensor Syst.*, Nov. 2022, pp. 954–961, doi: 10.1145/3560905.3568299.
[53] *Cascaded Tensorflow Model Optimization (TFMOT)*. Accessed: May 11, 2022. [Online]. Available: https://www.tensorflow.org/model_optimization/guide
[54] G. Retsinas, A. Elafrou, G. Goumas, and P. Maragos, "Weight pruning via adaptive sparsity loss," 2020, *arXiv:2006.02768*.
[55] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017, *arXiv:1712.05877*.
[56] J. Song, X. Wang, Z. Zhao, W. Li, and T. Zhi, "A survey of neural network accelerator with software development environments," *J. Semiconductors*, vol. 41, no. 2, Feb. 2020, Art. no. 021403.

HANAN HUSSAIN received the bachelor's and master's degrees in computer science and engineering from the University of Calicut, India. She is currently a Ph.D. Researcher with BITS Pilani, Dubai Campus. With a strong background in the field, she was a Research Assistant with the Artificial Intelligence Research Centre (AIRC), Ajman University, United Arab Emirates. Her research interests include deep learning algorithms, edge-artificial intelligence, and computer vision applications. Her work aims to advance AI capabilities for CV-based applications by enabling enhanced performance and efficiency of edge intelligence.

P. S. TAMIZHARASAN (Member, IEEE) received the Ph.D. degree in computer science and engineering from the National Institute of Technology-Tiruchirappalli, India. He is currently an Assistant Professor with the Department of Computer Science, BITS Pilani, Dubai Campus. His research interests include high-performance computing and deep learning.

PRAVEEN KUMAR YADAV (Member, IEEE) received the Ph.D. degree in computer science and engineering from the National University of Singapore, Singapore. He is currently a Co-Founder and the CEO of Atlastream Pte Ltd. He is also a Consultant with the Panasonic Research and Development Centre, Singapore. His research interests include multimedia systems and signal processing, specifically focusing on 3D and video compression.