IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 8, 2021, accepted November 23, 2021, date of publication November 30, 2021,
date of current version December 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3131396

Low Latency Deep Learning Inference Model for Distributed Intelligent IoT Edge Clusters

SOUMYALATHA NAVEEN[1], (Student Member, IEEE),
MANJUNATH R. KOUNTE[2], (Senior Member, IEEE),
AND MOHAMMED RIYAZ AHMED[3], (Senior Member, IEEE)
[1]School of Computer Science and Engineering, REVA University, Bengaluru, Karnataka 560064, India
[2]School of Electronics and Communication Engineering, REVA University, Bengaluru, Karnataka 560064, India
[3]School of Multidisciplinary Studies, REVA University, Bengaluru, Karnataka 560064, India

Corresponding author: Soumyalatha Naveen (soumyanaveen.u@gmail.com)

ABSTRACT Edge computing is a new paradigm enabling intelligent applications for the Internet of Things (IoT) using mobile, low-cost IoT devices embedded with data analytics. Due to the resource limitations of Internet of Things devices, it is essential to use these resources optimally. Therefore, intelligence needs to be applied through an efficient deep learning model to optimize resources like memory, power, and computational ability. In addition, intelligent edge computing is essential for real-time applications requiring end-to-end delay or response time within a few seconds. We propose decentralized heterogeneous edge clusters deployed with an optimized pre-trained yolov2 model. In our model, the weights have been pruned and then split into fused layers and distributed to edge devices for processing. Later the gateway device merges the partial results from each edge device to obtain the processed output. We deploy a convolutional neural network (CNN) on resource-constraint IoT devices to make them intelligent and realistic. Evaluation was done by deploying the proposed model on five IoT edge devices and a gateway device enabled with hardware accelerator. The evaluation of our proposed model shows significant improvement in terms of communication size and inference latency. Compared to DeepThings for 5×5 fused layer partitioning for five devices, our proposed model reduces communication size by $\sim 14.4\%$ and inference latency by $\sim 16\%$.

INDEX TERMS Convolutional neural network, deep learning, distributed intelligence, edge computing, fog computing, heterogeneous devices, inference model, IoT clusters, low latency.

I. INTRODUCTION

Convergence of IoT with deep learning facilitates uninterrupted service for daily life and industrial applications that use communication technologies and advanced data analytics. However, the massive amount of data generated by these devices are processed in the cloud infrastructure. Since cloud computing is plagued with issues like long transmission time, demand for more bandwidth, latency between IoT devices and the cloud, Edge computing has been introduced. This paradigm helps for real-time predictions and improves scalability, latency, and privacy by processing the data locally at the source.

Across the globe, billions of cameras capture images and videos every day, which constitute Big data. Cloud infrastructure stores and processes these data using Intelligent image/video analytics to generate actionable insights. Figure 1 depicts one of the mission-critical applications of 'Geriatric Care'. Unfortunately, due to lack of mobility and proper balance, many elderly fall and succumb to death. While 'Fall' seems to be the primary reason, the fatality is due to the inability to recover from the fall, which leads to poor physical and cognitive conditions. Most of the literature supports the fact that if the elderly are attended within seven minutes of the fall and can recover, the physical consequences (such as brain injuries) following the fall can be avoided [1], thereby reducing morbidity and mortality among the ageing population. One can propose to incorporate a camera-based monitoring system for fall detection. However, capturing the images continuously, classifying them into normal (sleep) and abnormal(fall) positions, and raising the alarm in emergency need intelligent video analytics [2] within no time.

The associate editor coordinating the review of this manuscript and approving it for publication was Eyhab Al-Masri.

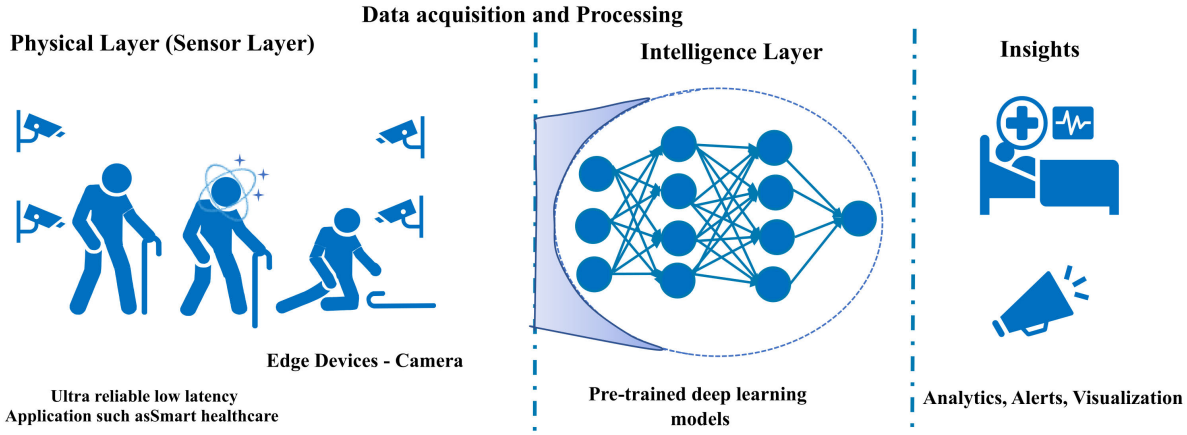

FIGURE 1. Illustration of 'Geriatric care' a time-critical application of IoT deployment. Edge computing strategy overview where the edge server is equipped with pre-trained DL models to identify the abnormal sleeping position (fall detection) with the least latency and high accuracy. The computing is independent of the cloud to avoid associated transmission delays.

We cannot afford to wait for the cloud to take care of such a huge data processing load in real-time. Cameras act as IoT devices that record data, process it, and deliver valuable insights on-site rather than sending it to the cloud, to reduce response time and latency. Processing such sensitive and personal data locally at the edge also satisfies privacy concerns.

The CAGR Edge computing market analysis report [3] estimates that the edge computing market is expected to reach USD 43.4 billion by 2027 due to numerous applications' growth of technologies such as 5G and data analytics. The unprecedented demand for smart edge devices has driven the industry to innovate and implement intelligent edge architectures for real-time, mission-critical applications which deal with large heterogeneous devices.

Any machine is intelligent if it mimics human behaviour such as perception, attention, cognition, and decision making. After a few winters of AI, machine learning [4] has brought momentum, and the proliferation of intelligent devices is driven via deep learning [5]. A convolution neural network is a deep learning algorithm widely used in computer vision, augmented reality, and virtual reality applications to process and classify images. An intelligent edge device is capable of handling data analytics via deep learning algorithms embedded in it. With Edge Intelligence [6], the aim is to push information processing load from the traditional cloud to edge devices to make them suitable for real-time applications.

Among all the deep learning models, CNN stands as a promising candidate for intelligent video analytics. Though there are many reasons for its massive adoption, the huge popularity can be owed to its ability to learn features without any human intervention and computational efficiency. As a result, many frameworks are being studied for the implementation of CNN. Among all the object detection algorithms, YOLO is the most preferred framework for object detection as it understands generalized object representation by looking only once [7]. This feature makes it super fast and can be run in real-time. In addition, YOLO is a fully convolutional network; it can process 45 frames per second.

Many attempts have been made to implement deep learning algorithms for edge intelligent applications [8]–[14]. The approach is either by training on the edge device [15] or the cloud [16]–[19] or local host [20], [21]. The training on the cloud is done by developing inference models [22], [23] on edge to provide a speedy and efficient prediction process [24]. However, few studies are reported to examine model compression techniques [25]–[28], knowledge distillation, network science-based knowledge partitioning algorithms [29], Early-Exit [30] and pruning algorithms [31]–[34] to reduce the memory footprint [35]. Table 1 presents state of art on DNN implementation at the edge and targeted performance metrics among recently published articles. List of important abbreviations in alphabetical order is found in Table 2.

In our extensive survey carried out for the DL implementation on edge devices, the following issues surfaced: 1) Deploying the deep learning model to edge devices such as mobile and other IoT devices require enormous computational resources and leave a vast memory footprint. 2) There is scarce research on inferencing techniques (for classification or prediction) in distributed heterogeneous IoT clusters. 3) Existing approaches are limited only to layer-based partitioning (to reduce the inference time) while completely neglecting the potential of compression techniques.

Generally, the previous works have tried to implement deep learning on edge devices using various frameworks for object detection, such as RCNN and SSD. YOLO is preferred for object detection as it is faster than RCNN. However, compared with SDD, the accuracy is less, has localization errors, and poor recall rate. Therefore, we chose YOLOv2, which is designed to overcome the limitations mentioned earlier.

Our work differs from previously published articles mentioned above in many ways; most of the earlier

TABLE 1. State of art on DNN implementations at the edge along with existing systems and frameworks including IoT devices, objectives, employed technologies, targeted performance metrics and effectiveness.

Ref.	IoT Devices	Model	Techniques used	Performance Metrics	Result
[36]	Raspberry Pi	Yolov2-16 layers	Used fused layer partitioning of convolutional layers along with work stealing approach	Minimize Latency	1.7X to 3.5X of speedup in CNN inference
[37]	Smart phones	Lenet, Inception-BN, VGG	Partitioning the trained DNN model across multiple mobile devices	Speed Latency	30.02% reduction in delivery time
[29]	Raspberry Pi, Odroid-XU4S	Resnet	Network science-based solution to the knowledge partitioning	Reduce the total latency	33X of reduction in total latency
[38]	Raspberry Pi	VGG-16, YOLOv2- 23 layers	Acceleration framework leveraging spatial partitioning techniques and parallelization	Improving the inference speed	1.9X 3.7X of speedup
[39]	Raspberry Pi	Yolov2, AlexNet, VGG-16, GoogleNet	Holistic optimization via Layer fusion and Partitioning the weight and fully connected layer	Speed up inference	1.52X of speed up
[40]	Raspberry Pi	GoogLeNet, ResNet	Collaborative and adaptive CNN inference system through scheduling	Reduced Inference Latency	Reduction upto 5.79X

TABLE 2. List of important abbreviations in alphabetical order.

Acronym	Definitions	Acronym	Definitions
5G	Fifth Generation	AE	Auto-Encoder
AI	Artificial Intelligence	ANN	Artificial Neural Network
AP	Access Point	AR	Augmented Reality
BS	Base Station	CC	Cloud Computing
CNN	Convolutional Neural Network	CV	Computer Vision
D2D	Device-to-device	DL	Deep Learning
DNN	Deep Neural Network	E2E	End-to-end
EC	Edge Computing	EH	Energy Harvesting
FC	Fog computing	FCNN	Fully Connected Neural Network
HetNets	Heterogenous Networks	IoT	Internet of Things
KD	Knowledge Distillation	KPI	Key Performance Indicator
LSTM	Long Short-Term Memory	M2M	Machine-to-machine
MC	Mobile Computing	MEC	Mobile Edge Computing
ML	Machine Learning	MLP	Multi-Layer Perception
NCS	Neural Compute Stick	NLP	Natural Language Processing
NN	Neural Network	NoC	Network-on-Chip
PCA	Principal Component Analysis	RFID	Radio Frequency Identification
RNN	Recurrent Neural Network	SDN	Software Defined Network
URLLC	Ultra-reliable and low latency communication	WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network	YOLO	You Only Look Once

work have focused on layer partitioning or compression. In contrast, we have used pruning algorithms even before layer partitioning. In addition, the majority of the works have aimed at implementation while we attempted to optimize the pre-trained model to make it faster and robust for real-time applications. Our objective in implementing DL for edge devices is threefold:

- **We propose an Edge-to-Device edge computing framework that facilitates optimization using the weight pruning method to deploy the model onto small smart devices devised for time-critical applications.** Our vision is to manifest a holistic distributed deep learning framework that orchestrates edge clusters to process the heterogeneous local data, take decisions (with the aid of a pre-trained DL model obtained from the cloud) and execute real-time applications with the least latency.
- **We formulate the latency minimization into a pruning problem and obtain an optimized pre-trained model by weight-pruning and fine-tuning.** We intend to build a horizontal collaborative CNN inference accelerating system. The input feature maps are partitioned and distributed among resource-constrained edge devices, such that memory footprint and latency are minimal.
- **The performance of our proposed framework and optimization techniques on a heterogeneous distributed edge network is evaluated** under a variety of system parameters.

The remainder of the paper is arranged as follows: Section II provides an account of a brief review of edge computing and a decent review of literature on using various DL frameworks for object detection in real-time. Section III presents the proposed model, while section IV presents the details of the experimental setup. Results and discussions are presented in section V, and the paper concludes in section VI.

II. BACKGROUND AND LITERATURE REVIEW

Any IoT implementation aims to develop a smarter environment with the least intervention. IoT's core is its computing framework [41]–[43], which processes data. The data processing can happen either at a cloud server called cloud computing or at the edge server called edge computing or

between them called fog computing. Cloud servers are powerful computing machines [44] with no energy constraint. However, since all data must be transmitted to the cloud, cloud computing finds its limitation in processing IoT data due to its inherent high latency, lack of privacy [45], [46] and demand for high connectivity.

Moreover, IoT data differs from Big data and is characterized by large amounts of streaming data, heterogeneity, time and spatial correlation, and high noise data. Beyond traditional data analytics, IoT data demands fast and streaming data analytics with time sensitivity [47]. Hence, obtaining inference from this kind of data is not a straightforward task. However, recent progress in hardware with superior computing speeds and advanced ML techniques opens the doors for shifting the computing load to the edge.

Several paradigms have been proposed for the workload distribution of data analytics among the cloud, edge and fog [48]. One approach is to have edge clusters (servers) that will pre-process and send the raw data to the cloud before training and send updates to the cloud after training. In addition, edge clusters will carry out the data cleansing, dimensionality reduction (PCA), noise removal (LDA) [49] and may also employ an autoencoder for extracting standard features. Therefore, having edge clusters will benefit in 2 ways: 1) alleviating the workload of the network 2) significantly reducing the network latency.

Edge computing is ideal for time-critical IoT applications as it is near the end-user and is dedicated, unlike cloud computing (shared), thereby avoiding delays incurred in transmission. In addition, edge computing costs less, requires less bandwidth, provides data privacy, and can be programmed for application-specific tasks [50]. The features like mobility management, geo-distribution, location awareness, scalability, and ultra-low latency make edge computing [51] more suitable for IoT data analytics. Mission-critical applications (such as driverless cars, fire prediction, and geriatric care) demand for latency less than 1msec [52] with reliability of more than 99.99%.

The challenge is how to serve the resource-intensive functionalities by resource-constrained [53], [54] edge devices. Edge computing needs to embrace intelligent processes to alleviate the burden of computation, communication, and storage. Incorporating intelligence at the edge refers to specialized control mechanisms for being context-aware and responsive while optimizing latency and energy efficiency parameters. Edge intelligence (union of EC and AI) strives to mimic human cognition by processing and learning from the data generated among heterogeneous edge devices (and servers) in the proximity. Beyond being fast, secure, and economical, edge intelligence exploits the potential of richer data to provide application-specific optimization [55]. Thus, edge infrastructure will pave a path for the democratization of AI [56]. Either intelligent process enables edge architecture or vice-versa, the end-users will be beneficiaries with reduced bottlenecks and enhanced scalability.

Advancements in ML algorithms capable of emulating human reasoning seems to be a step towards AI. DL, a subset of ML, is a state-of-art method of uncovering patterns and extracting valuable insights from a large chunk of data. As the name describes, deep learning learns from multiple levels to develop a model embedding complex relations among the data [57]. Neural networks mimic neurons in the brain. Deep Neural Networks is the core of machine learning techniques spanning from simple data analytics to natural language processing. A typical Deep Neural Network model contains layers of fully connected nodes (depicting neurons), and the process of passing forward the raw data from the input layer to the concerned category at the output layer is called model inference.

Deep Learning is independent of domain-specific training among all ML techniques, thereby accelerating inference via pre-trained models. However, DL implementation demands resources for computation, memory (and cache), and power. Leveraging the potential of DL models for inference on resource-constrained edge devices [58] is the crux of the matter. Hosting Artificial Intelligence on edge devices via DNN computations, i.e. deploying DNN models close to the users for fast real-time execution, has been focused on in the past few years. However, though the idea is excellent, the computational complexity creates a bottleneck for its implementation. As a result, resource-constrained edge devices have to redefine themselves to achieve it.

Transforming edge computing is possible either by hardware acceleration [5] or by software acceleration as described below:

1. Hardware transformations embrace DNN computation at the hardware level design for DL inference. Beyond usage of accelerators, they include hardware friendly optimizations such as matrix multiplication factorization, data path optimization, and parallel operations.
2. Software transformations involve defining novel DNN structures by finding the trade-off between accuracy and computation. Then, the desired accuracy with moderate computations in the resource-constrained devices can be achieved by compressing the DNN models via Pruning. Other techniques like quantization and approximation are also employed.
3. Complementary to the above two approaches, one can focus on run-time management. This includes DL model partitioning and offloading [59] the computational load between the cloud, edge and device to accelerate diverse embedded applications. The run-time management can be applied over and above the two transformations mentioned earlier.

Hardware accelerators find limitations in providing storage space for the large pre-trained model. They may also fail to achieve the expected inference rate, which will worsen when the accelerator executes other parallel tasks. Therefore, software acceleration must complement hardware accelerators to create an ecosystem for achieving inference models with excellent performance. In this regard, many attempts

TABLE 3. Important definitions re-defined according to the edge computing domain.

Terminologies	Definitions
IoT devices	IoT encompasses sensors to sense the event in the physical world, processing networks for the processing and preparing data, data analytics to get insights from data, and system monitoring for data sharing between devices and servers or among themselves.
Edge Learning	**Edge** devices or edge servers perform prior learning at the edge of the network by pre-processing the raw data to reduce the network traffic and speed up the computation at the data centers. or Edge learning enables edge devices to send the context based raw data to the edge server for inference.
Gateway	Gateway is a device to interface edge devices to the internet via various protocols.
Cloud Computing	In cloud computing, users can access plenty of computational resources over the internet on an on-demand basis with little management effort and minimal interaction from service providers.
Offloading	**Sharing** the workload among the cloud, edge and fog
Fog computing	Many distributed, heterogeneous distributed devices collaboratively cooperate to perform the tasks and store the data in a fog server/cloud server.
Edge Computing	Computing at the edge without passing anything to the cloud, works for offline mode.
Computing Frameworks	To process the data collected by the IoT, different computing frameworks are used depending on the applications. Computing frameworks are generally categorized as Fog computing, Edge computing and Cloud computing.
Task Parallelization	**Task Parallelization**is an approach to maximize resource utilization for the dynamic task by exploiting concurrency by executing the task parallel by adopting various techniques.
Network Science	Deals with representing certain complex phenomena as networks an then create models that can be used to understand and predict the desired phenomena
Model Inference	**DNN** model consists of series of inter-connected layers. The process of passing forward raw data from the input layer to one of the category at output layer is called Model inference
Model Compression	Model compression refers to a class of techniques that reduce size and computations of DNN without losing accuracy. It enables to run the model on tiny devices either through pruning(Weight, filter, layer) or by Quantization(Lower precision- Fewer bits per weight)
Pruning	**Pruning** is one of the compression techniques used in the deep neural network to reduce the model's parameters through weight or channel pruning to produce a simpler model without losing accuracy.
Quantization	Quantization is an approach that can be used in machine learning for model compression to reduce the memory footprint. Conventional deep learning models are trained with 32-bit floating-point weights and activations; this approach reduces the number of bits used for representing weights and activation functions.
Knowledge Distillation	Train a significantly smaller student network to mimic a larger teacher model (model compression by replacing teacher model by smaller student model)
Model Partitioning	**Layer Partitioning:** To distribute the work among the edge devices for faster inference and to reduce the memory footprint, each layer input feature map is partitioned and assigned to the edge device. **Fused Layer Partitioning:** Deep neural networks model's stacked layers are fused and dividing vertically and assign each partition to edge devices without any off-chip movement.
Latency	Latency is measured from the moment the request is initiated to the time the response is received, which includes transmission, waiting, and processing time. **Network Latency:** During the data transmission, the time taken for transferring the data from the device to the cloud is called Network Latency **Service latency** is measured as the time taken by the processing device for computation based on the current workload.
Jitter	**Jitter** is variation in latency and is an important measure for any applications.
Accuracy	Accuracy indicates the frequency at which predictions match the labels.

have been reported to optimize the DNN models at edge devices [60], [61]. While Communication load, communication overhead, cost, memory, processing speed, network bandwidth, jitter, complexity are a few performance parameters, much of the preliminary research has focused on low-latency and energy-efficient computations.

Tailoring the pre-trained DL models to suit the specific application by creating hardware and software accelerators is the need of the hour. Along with hardware accelerators, DL model partitioning and distribution for inference is the key to exploiting the full potential of edge computing. Several frameworks have been proposed to leverage the capabilities of edge infrastructure enabled with hardware accelerator and embedded with DNN models. The aim is to have fast multimodal data analytics in smaller scale platforms. DNN partitioning was conceived in a bid to accommodate resource-intensive computations on resource-constrained edge devices. The DNN model is partitioned into multiple parts and shared among devices, thereby collaboratively computing for low latency DNN inference.

Layer partitioning is done in two ways: layer wise parallelization, where each layer is independently parallelized while selecting appropriate techniques for each layer to obtain the best performance. Another method is a fused layer parallelization where the output of one layer is fed as input to the next layer without any off-chip data movement (i.e. without going to the memory). This work considers parallelizing by multiple fused layers instead of a single layer individually because it is scalable, network bandwidth-efficient, and has less memory footprint.

Table 3 provides the important definitions re-defined according to deep learning based edge computing environment.

Table 4 outlines the architectures, critical performance metrics, enabling approaches, and DL Models and frameworks for DNN model inference at the edge.

Our work focuses on reviewing, analysing, and implementing DL model partitioning and distribution to accelerate the inference task at the edge. The model assumes that a pre-trained DL model is already obtained from the cloud

TABLE 4. Overview of systems and frameworks for edge computing model inference, including the adopted architecture, the DL model and the enabling technologies.

Aspects	Various (Our considerations in bold)
Architecture	Device, Edge, **Device-Edge**, Edge-Cloud
Performance Indicators	**Inference Latency**, Memory, Energy, Communication cost, Communication overhead, Network Bandwidth
Enabling Technolgies	**Model compression (Pruning), Model Partitioning (Fused Tile partiotioning)**, Quatization
Optimization	Resource efficient (Application specific optimization)
Frameworks	**DeepThings, ModNN**
DL Models	Yolo, **Yolov2**, VGG, AlexNet

and the entire execution is offline. While considering all the trade-offs, we intend to optimize this pre-trained model via Pruning. Parallelization is achieved through Fused layer partitioning [62]. The model optimization via Pruning is employed even before the model partitioning and distribution is performed. We use a Raspberry Pi 3B plugged with Intel Movidius Neural Compute Stick (NCS) as a hardware accelerator to build a DL inference system. We report the pruning approach, model partitioning and distribution methods, experimental results, and improvements in performance achieved. In summary, we intend to build a horizontal collaborative CNN inference accelerating system in which the feature maps are partitioned and distributed among resource-constrained edge devices, such that memory footprint and latency are minimal.

III. PROPOSED MODEL

As CNNs are resource-intensive, the deployment of CNN on resource-constrained IoT devices is challenging to make this approach realistic for real-time applications. Therefore, we propose a CNN inference model for distributed heterogenous Edge clusters to minimize the communication size and inference latency.

Table 5 presents the summary of notations used in this article.

A. SYSTEM MODEL

Our proposed system consists of set of Edge devices, Edge gateway and host machine. Edge devices and Edge gateway device is interconnected to share the information among each other and host machine is used to trigger the edge device and gateway device to process the input feature map.

All the IoT devices in the network are denotes as D, and C represents the communication Edges between the devices, ED represents Edge device and EG represents Edge Gateway which is presented as.

Overall network is denoted as

$$N = [D, C] \tag{1}$$

Set of devices in the network is represented as

$$D = [ED_i, EG_i, Host] \tag{2}$$

TABLE 5. Summary of key notations used in this article that are relevant to the algorithm and the theoretical analysis.

Symbol	Menaning
D	Set of devices
ED	Edge devices
n	Number of devices
C	Connection
EG	Edge Gateway
λ	Latency
λ_{nl}	Network latency
λ_{cl}	Computational latency
λ_{il}	Inference latency
α	Task assigning indicator
y_i	Computational Complexity
z_i	Computational Capability
f	Number of Filters
W^s_{f*}	Optimized pre-trained model of the source model
W^s_f	pre-trained model with f number of filters optimized for s source
f	Number of Filters
D_s	Source data
D_t	Target data
£()	Loss function during the optimization of the network
£()	Loss function during the optimization of the network
W	Weights of the pre-trained model
B	Bias of the pre-trained model
f^*	The optimized filter output after pruning
q	Pruning ratio

Communication Edges of the network are represented by C, and the value of |C| depends on the number of IoT devices and edge computing devices.

If |D| = n, then maximum the values of |C| is given as below in Equ

$$|C| = 0 < |C| < \frac{n(n-1)}{2} \tag{3}$$

Several authors have attempted to model the latency in deep learning based Edge computing environment. To simplify the analysis, let us denote latency as λ and this contains Computational latency (λ_{cl}) and network latency (λ_{nl}) [63].

$$\lambda = \lambda_{cl} + \lambda_{nl} \tag{4}$$

For our framework network latency (λ_{nl}) = 0 as data are not transmitted to the cloud network. Hence Latency for our framework

$$\lambda = \lambda_{cl} \tag{5}$$

The Computational latency (λ_{cl}) majorly depends on processing the input and in turn depends on workload of each processor's of IoT devices. Computational Latency is calculated as below at time t considering the task assigning indicator (α), is estimated as

$$\lambda_{cl} = \alpha(\frac{y_i}{z_i}) \tag{6}$$

where y_i is the computational complexity of the CNN used at k and z_i is the computational capability at k.

Hence the Inference latency(λ_{il}) [64] estimation for the IoT edge clustering network is given by

$$\lambda_{il} = \lambda_{cl} \tag{7}$$

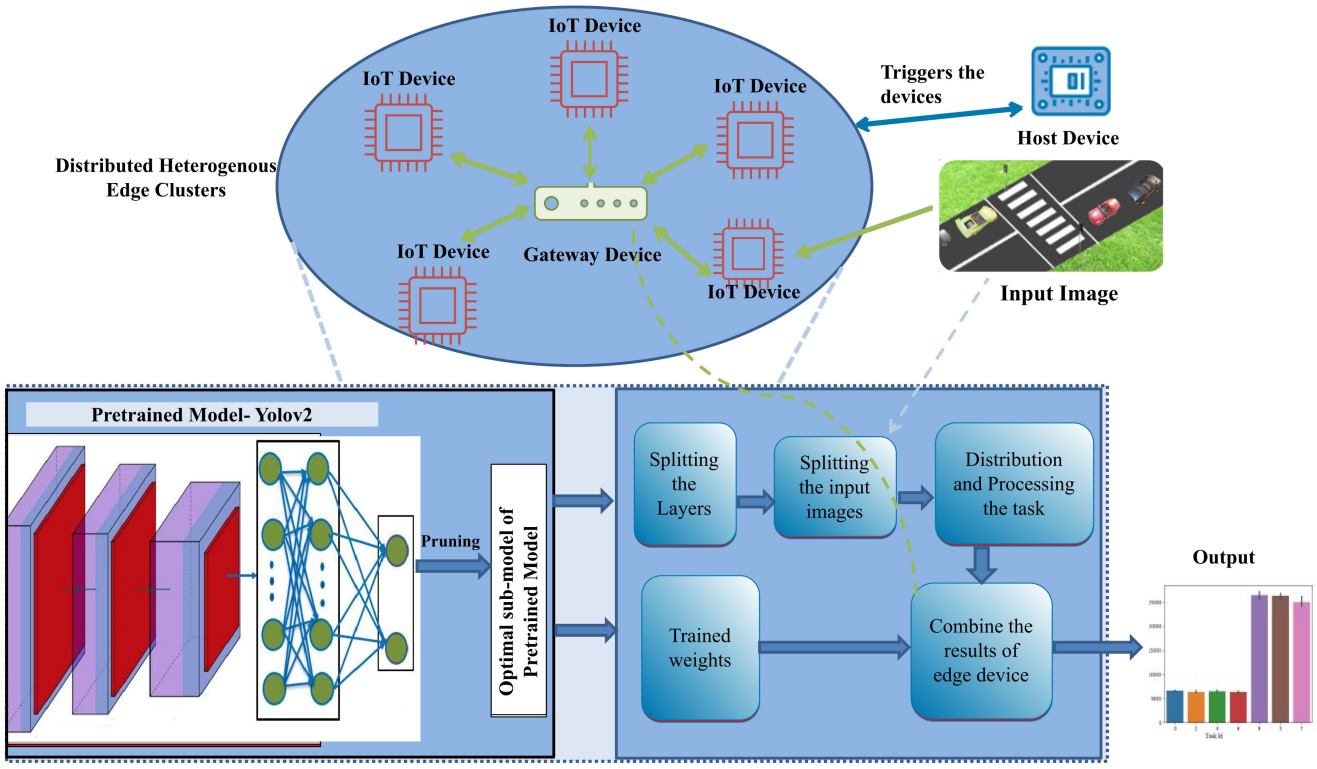

FIGURE 2. Proposed CNN based Inference Model for Edge-device inference. Initially, the framework takes the pre-trained Yolov2 model as input. This model is compressed by weight pruning. In each iteration, the pruning approach removes the unimportant weights based on the threshold value. The removed weights are updated in the back-propagation. Finally optimized sub-model of the pre-trained model is fed to the fused tile partitioning and distribution module to split the model via fused layer and distribute each partition to each edge device for feature extraction and for minimizing the inference latency.

We can envision that the proposed framework works well for real-time intelligent device applications such as smart homes, smart agriculture, intelligent surveillance, and automated vehicles, in which devices of varying capabilities need to cooperate to make an inference. Hence, edge devices such as intelligent devices with a wide range of computational capabilities are considered for processing input and performing collaborative inference tasks.

These devices are deployed with optimal sub-model of the pre-trained model of Yolov2 to make the devices intelligent and make the decision quickly. Figure 2 represents the proposed optimal pre-trained CNN based Yolov2 model for edge devices. The proposed model consists of four modules. The first module involves establishing heterogeneous edge clusters for IoT. The second module involves applying the pruning to the pre-trained model to obtain the pre-trained model's optimal sub-model while retaining significant accuracy. Finally, as the third step, we will load the optimized pre-trained CNN based Yolov2 model onto the Edge clusters. In the fourth step, fusion layer partitioning is applied, and the task is distributed to IoT edge devices, then processed. Lastly, the gateway device combines the results from all the edge devices to perform collective inference and display the total latency for the inference task. Each of the modules is explained in detail in the following subsections.

B. HETEROGENOUS EDGE CLUSTERS

The distributed heterogenous Edge clusters set up involves forming the interconnected distributed IoT device. IoT devices are resource constraints in nature; to perceive the limited computing resource, we have selected Raspberry Pi 3B+, Raspberry Pi 3B, Raspberry Pi 4 and Neural computing stick-2(NCS2) to form Edge Clusters. These devices are of different computing capabilities and also memory sizes. Furthermore, to speed up the inference task and to accelerate processing, the gateway device is plugged with NCS2. The detailed experimental setup is discussed in section IV.

C. CONVOLUTIONAL NEURAL NETWORK

This paper focuses on CNN, a deep learning algorithm that comprises an input layer, output, and multiple hidden layers. We have chosen CNN based Yolov2 object detection and classification CNN model as shown in Figure 3. Yolov2 network consists of 24 convolutional layers followed by a fully connected feed-forward neural network. With the convolutional operations, features are extracted from the image. In our experimental setup, we have used an image size of $608 \times 608 \times 3$, which is width, height and three RGB channels, respectively, as input to Yolov2. As shown in Table 6 Yolov2 detection network, layer1 uses 32 filters and hence convolutional layer computes output as $608 \times 608 \times 32$.

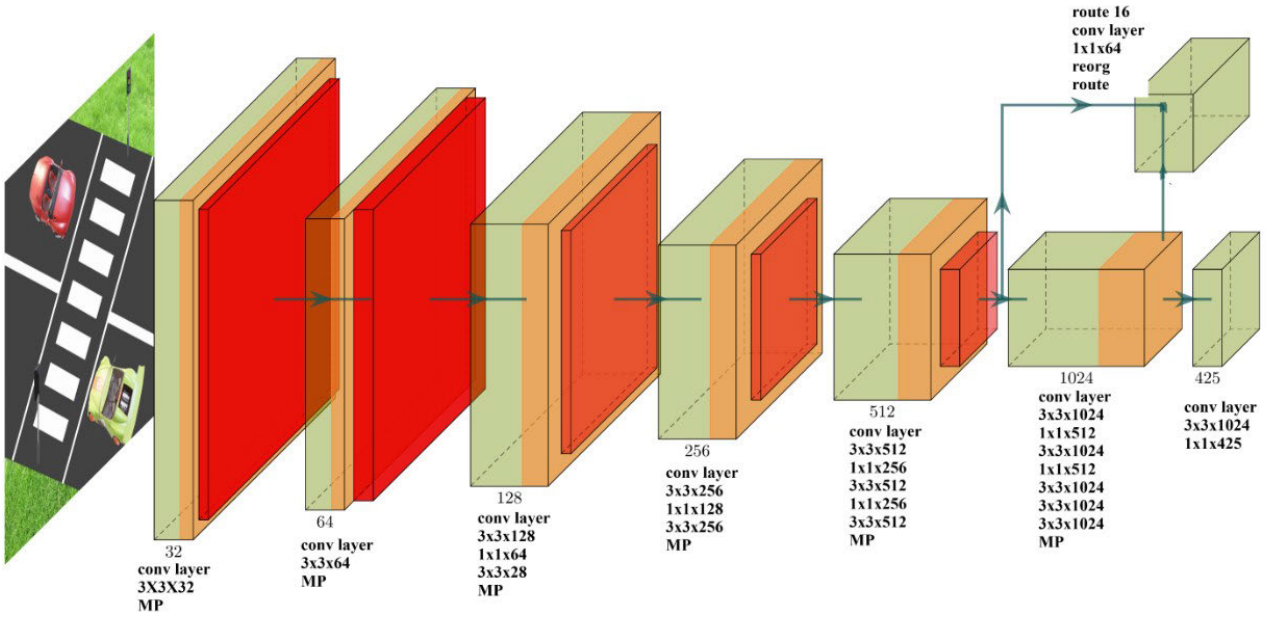

FIGURE 3. Yolov2 Architecture with the input image size 608 × 608. CNN based inference workflow for image classification tasks consists of a series of convolution, max pooling, batch normalization and activation function for feature map extraction. Layer1 computes output as 608 × 608 × 32 for 32 filters and Maxpooling reduces the image size to reduce the computation cost. In the feature extraction stage, the input image is processed to generate hidden features followed by a fully connected layer for classification. Feature extractor takes large inference latency leading to the bottleneck of CNN inference.

Furthermore, layer 2 Maxpooling performs a down sampling operation to reduce the image size, and hence the number of learnable parameters reduces the computation cost.

The Yolov2 detection network uses filters of size 32,64,128 and so on. These are two- or three-dimensional arrays applied across the input data through the sliding window through the element-wise dot product to produce the feature map. Stride represents the number of pixels shifting during the convolutional process. Value of Stride of one indicates the movement of 1 pixel at a time.

For better accuracy in Yolov2 architecture, the Reorg layer combines features from layers of middle level and high level. The 28th route layer uses the 27th and 24th layers to append the features of the previous layer and obtains the output as 19 × 19 × 1024. Three channels- R, G, B are considered with a momentum of 0.9 with a learning rate of 0.001, the stride of 1 and leaky activation function is used.

The role of the fully connected layer is to make the final decision by executing an activation function on the overall sum of the linear combination of the input.

D. PRUNING

In the literature, several compression techniques [29] such as pruning, quantization, and knowledge distillation have been used to optimize the convolutional neural network-based model without sacrificing accuracy.

Pruning is applied to minimize the network parameters to reduce the model's size and make it suitable for deployment on to resource constraint IoT devices for faster predictions. Pre-trained models are trained with large scale datasets. Transfer learning allows the model to be reused for similar tasks.

Due to the significant results shown by the pre-trained models, Transfer learning improves performance by optimizing the existing model on other related model designs. Pruning the trained model yields better results for efficient inference. The standard performance metrics used to evaluate the model after pruning are prediction accuracy, model size in terms of bytes/kilobytes and computation time through Floating point operations (FLOP) or memory utilization. The CNN can be optimized by either weight pruning, layer pruning, or filter pruning.

The most effective approach is to apply weight pruning. In our work, the pre-trained model of Yolov2, which is trained with large scale datasets, is tuned to accelerate by removing unimportant weights through weight pruning. The computation time is reduced with a smaller number of model parameters, and the model performs better with greater accuracy.

Formulation of pruning as an optimization problem [18] is represented below. The objective of pruning is to generate the optimized sub-model of the pre-trained model. Here, W^s_{f*} represents the optimized pre-trained model of the source model.

W^s_f indicates the pre-trained model with f number of filters optimized for s source.

Consider $D_s = X_0, X_1, \ldots X_n$ and $D_t = Y_0, Y_1, \ldots . Y_n$ as source data and target data respectively.

£() represents the loss function during the optimization of the network.

TABLE 6. Detailed Yolov2 detection network architecture for feature extraction. In our work Yolov2 pre-trained model is used. Yolov2 uses Darknet-19 architecture an open source neural network framework. In the table the repeated 1 × 1 convolution enables to reduce the number of parameters. In the table Conv-layer indicates the convolutional layer.

Layer	Filter	SizeStride	Input	Output
Conv-layer	32	3 X 3/1	608 X 608 X 3	608 X 608 X 32
Maxpooling		2 X 2/2	608 X 608 X 32	304 X 304 X 32
Conv-layer	64	3 X 3/1	304 X 304 X 32	304 X 304 X 64
Maxpooling		2 X 2/2	304 X 304 X 64	152 X 152 X 64
Conv-layer	128	3 X 3/1	152 X 152 X 64	152 X 152 X 128
Conv-layer	64	1 X 1/1	152 X 152 X 128	152 X 152 X 64
Conv-layer	128	3 X 3/1	152 X 152 X 64	152 X 152 X 128
Maxpooling		2 X 2/2	152 X 152 X 128	76 X 76 X 128
Conv-layer	256	3 X 3/1	76 X 76 X 128	76 X 76 X 256
Conv-layer	128	1 X 1/1	76 X 76 X 256	76 X 76 X 128
Conv-layer	256	3 X 3/1	76 X 76 X 128	76 X 76 X 256
Maxpooling		2 X 2/2	76 X 76 X 256	38 X 38 X 256
Conv-layer	512	3 X 3/1	38 X 38 X 256	38 X 38 X 256
Conv-layer	256	1 X 1/1	38 X 38 X 512	38 X 38 X 256
Conv-layer	512	3 X 3/1	38 X 38 X 256	38 X 38 X 512
Conv-layer	256	1 X 1/1	38 X 38 X 512	38 X 38 X 256
Conv-layer	512	3 X 3/1	38 X 38 X 256	38 X 38 X 512
Maxpooling		2 X 2/1	38 X 38 X 512	19 X 19 X 512
Conv-layer	1024	3 X 3/1	19 X 19 X 512	19 X 19 X 1024
Conv-layer	512	1 X 1/1	19 X 19 X 1024	19 X 19 X 512
Conv-layer	1024	3 X 3/1	19 X 19 X 512	19 X 19 X 1024
Conv-layer	512	1 X 1/1	19 X 19 X 1024	19 X 19 X 512
Conv-layer	1024	3 X 3/1	19 X 19 X 512	19 X 19 X 1024
Conv-layer	1024	3 X 3/1	19 X 19 X 1024	19 X 19 X 1024
Conv-layer	1024	3 X 3/1	19 X 19 X 1024	19 X 19 X 1024
route 16				
Conv-layer	64	1 X 1/1	38 X 38 X 512	38 X 38 X 64
reorg		/2	38 X 38 X 64	19 X 19 X 256
route	27 & 24			
Conv-layer	1024	3 X 3/1	19 X 19 X 1280	19 X 19 X 1024
Conv-layer	425	1 X 1/1	19 X 19 X 1024	19 X 19 X 425
detection				

The parameters such as $W = W_0, W_1, W_2, W_3 \ldots..W_n$ and $B = B_0, B_1, B_2, B_3 \ldots..B_n$ are weights and bias of the pre-trained model. These are optimized to minimize the loss function with respect to source data

Pruning percentage denoted by q% varies from 20% up to 80% on source data. The threshold is calculated using percentile over all the weights and the pruning percentage. Accuracy of 90% is maintained during the pruning process for producing the sub-optimized model of the pre-trained model.

We define loss function as

$$£(D_s|W_f^s) \tag{8}$$

While optimizing the pre-trained model, Loss function £() is defined as minimizing the loss on source data D_s of pre-trained model W_f^s to loss on source data of optimized pre-trained model with respect to accuracy.

To generate the optimized sub-model W_{f*}^s with minimal loss function with respect to source data D_s, pruning has to be performed with q% of pruning ratio.

Let f represents the number of filters optimized for source data set, f^* represents the optimized filter output after pruning.

$$f^* = f * q\% \tag{9}$$

W_{f*}^s, the optimized pre-trained model of the source model is evaluated as

$$W_{f*}^s = min|(\$(D_s|W_f^s) - \$(D_s|W_{f*}^s))| \tag{10}$$

Algorithm 1 Pruning the Pre-Trained Model, Dataset

Input: Pre-trained model, pre-trained weight

Output: Optimized sub model of pre-trained model

Initialization: Pre-trained model W_f^s, pre-trained weights W, source data D_s, loss function £(), Optimized sub model of pre-trained model W_{f*}^s

```
pruningPerc=0.30
threshold=percentile(sum(W), pruningPerc)
WeightPrune(W_f^s, pruningPerc)
for i = l to modelweight do
    if (i ≥threshold) then
        i =i
    else
        i = 0
    end if
end for
Retrain the pruned model until
Calculate the loss function using equation 10
if (Lossfunction ≥T) then
    break
else
    W_{f*}^s=W_f^s
end if
return optimizedmodel W_{f*}^s
```

Algorithm 1 provides steps for pruning the Yolov2 pre-trained model to optimize using weight pruning.

In Algorithm 1, the weights are pruned to lower the size of the yolov2 model while maintaining considerable accuracy without compromising the original task. Next, the model is pruned iteratively based on the weights and thresholds set. If the weights are less than the threshold, they are set to zero. Finally, the model is retrained based on the loss function. The optimized sub-model of the pre-trained model is obtained after achieving a minimum loss function and significant accuracy.

E. PARTITIONING THE FUSED LAYERS

In this work, we employ fused layer partitioning [36], [62] for CNN inference models for multiple resource constraints edge devices. The primary purpose of fused layer partitioning is to reduce inference latency and communication size by distributing the input across multiple devices.

CNN consists of convolutional layers, pooling layers and activation functions for feature extraction and classifier to classify. In CNN, input to the layer purely depends on the output of the previous layer. The optimized pre-trained model

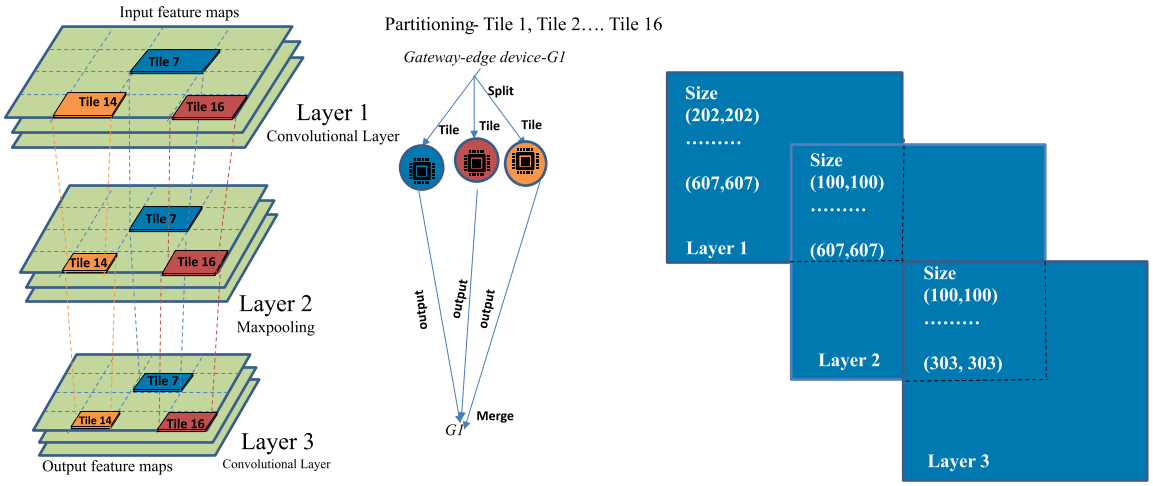

FIGURE 4. CNN consists of multiple convolutional and max-pooling layers. In Fused Tile partitioning, CNN is divided vertically into tiled stacks of fused convolutional and pooling layers. As shown in the figure, Feature maps of each layer are divided into small tiles. Fused Tiles across the layer are executed independently by the devices parallelly to reduce the latency and communication overhead. This concept distributes sizeable deep learning models that do not fit on a single memory-constraint IoT device.

parameters are split into multiple fused partitions to distribute among multiple devices to reduce the latency, as shown in Figure 4, Each partition in each device generates output feature maps. Finally, partial output from each device is collated by the gateway device to form the final output. As single input is distributed across numerous devices, this method minimizes computation latency and communication size as communication happens between the devices only during the partitioning in the beginning and at the end for the merging process.

Consider CNN with M layers. Each layer takes N set of feature maps in an input. Each convolution operation in each layer of m $= 1, 2, \ldots$ M with input dimension $Wm - 1 \times Hm - 1$ with set of Km filters with dimension $Dm \times Dm$ are used to slide across input with Stride Sm. Summation of result of multiplication of filter values with the set of input feature map produces output feature map. Process continues for Km filters. Among convolutional operation, maxpooling, activation function and fully connected layers, convolutional layer consumes more computational cost compared to other layers.

Hence to optimize the computational resources, the first 16 layers of yolov2 are vertically partitioned into tiles. Then the corresponding tiles from each layer are fused vertically to form a single unit. Finally, the input features map (input data region) for that particular tile is loaded from memory and subsequently, the intermediate layers compute intermediate values.

The process of fused layer partitioning and distribution is shown in Figure 5. The figure illustrates the fused tile partitioning of two layers. First Layer is partitioned into N sets of 5 × 5 tiles. Thus, only 5 × 5 × N input data are brought from memory. Next, the convolution operation is performed on Layer1 with a 3 × 3 × N filter producing

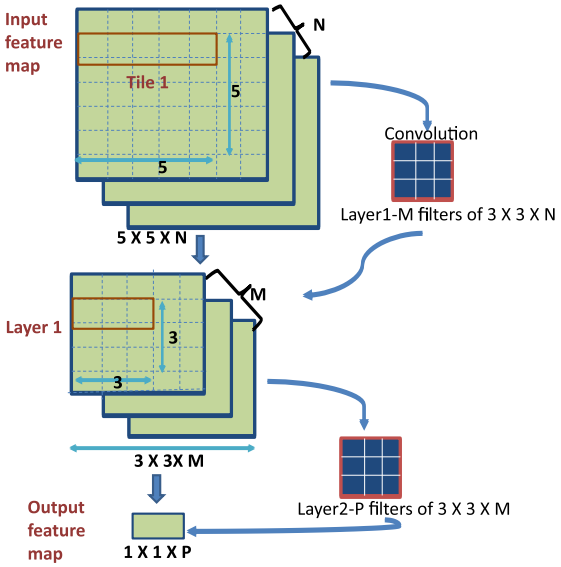

FIGURE 5. Illustrates fusion of two convolutional layers refer as layer 1 and layer 2. Layer1 receives the input feature map as input. These inputs comprise N different 5 × 5 feature maps. This feature map is convolved with 3 × 3 kernels for the stride of one and extends the same process down for all feature maps. The layers create a computational pyramid across the layers of feature maps.

a 3 × 3 × M intermediate feature map. Finally, the input is obtained to Layer 2, which convolves with P filters, and produces 1 × 1 × P output feature maps.

IV. EXPERIMENTAL SETUP

The proposed system has been evaluated with the well-known CNN based Yolov2 model. An optimized sub-model of the pre-trained model has been obtained after pruning. Parameters of the optimized model are fed into the framework, and fused layer partitioning and distribution modules have been

TABLE 7. List of hyperparameters considered for pruning the model, parameters of fused tile partitioning approach and detailed experimental parameters.

Parameters	Values
Pruning parameters	
Pruning percentage(q%)	10%, 20%, 30%,...80%
batch-size	32
conf-thresh	0.005
stride	1
pad	1
Fused Tile partitioning	
Deep Learning model	Yolov2
FTP partitioning	3X3, 4X4, 5X5
Experiment setup	
Number of Edge devices	5
Number of Gateway devices	5
Local Host	1
Edge device configuration	Raspberry Pi with the network connectivity
Gateway device configuration	Raspberry Pi with the network connectivity plugged with NCS2
Inference engine	Darknet Neural Network
Software Accelerator	NNPACK

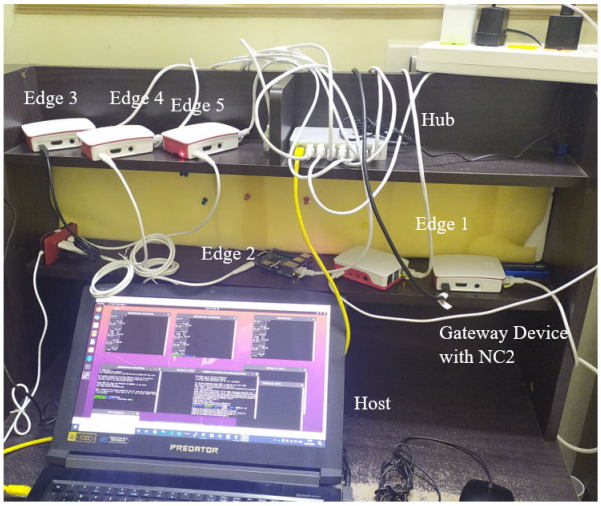

FIGURE 6. Hardware experimental setup of heterogeneous distributed collaborative IoT clusters consists of five raspberry pi's as edge devices, one Hub, and one raspberry pi plugged with NCS2 as a gateway device. We use an input image of size 608 × 608.

implemented to distribute the task into multiple IoT devices for faster predictions. These modules are implemented in C and C- based Darknet neural network libraries used as CNN inference engines. As IoT devices are resource constraints in terms of Processing capabilities, Darknet with NNPACK has been used for accelerating the CNN performance on IoT devices. The experimental setup is as shown in Figure 6.

TCP/IP socket APIs are used to communicate distributed edge clusters between hosts, gateway devices, and edge devices. An optimized sub-model of the pre-trained model is deployed on a set of IoT devices. To mimic the real-world scenario, which usually has resource constraint IoT devices, in our experiment, we have opted for 5 Raspberry Pi devices as Edge devices and one Gateway device accelerated by

TABLE 8. List of different heterogenous configurations we considered in our study. Focus on heterogeneity in computational ability(different device speeds) in order to evaluate the robustness of our approach and speed of inference.

Device	Frequency	RAM	Core
Raspberry Pi 3 Model B+	1.4GHz	1GB	Cortex-A53,ARM v8
Raspberry Pi 4 Model B	1.5GHz	2GB	quad-core, Cortex-A72,ARM v8
Raspberry Pi 3 – Model B	1.2GHz	1GB	Quad-Core,ARM Cortex-A53
Raspberry Pi 4 Model-B	1.5GHz	4GB	quad-core,Cortex-A72 (ARM v8)
Intel Neural Compute Stick 2	Intel Movidius Myriad X Vision Processing Unit, Supports Ubuntu, Windows, Raspbian		

NCS2 and a Host machine. The edge devices used for evaluation are Raspberry Pi 3 Model B+ with 1GB RAM, Cortex-A53 1.4GHz, Raspberry Pi 4 Model B with 2 GB RAM, quad-core, Cortex-A72, 1.5GHz and Raspberry Pi 3 – Model B with 1GB RAM, Quad-Core, 1.2GHz and Intel Neural Compute Stick 2 has been used to accelerate the Gateway devices for faster inference. Assumptions considered in designing the framework and experimental setup are listed below.

- Input data is considered as an image of any size.
- Computing capabilities such as varying processor speed and size of RAM is considered as heterogeneous IoT devices.
- A single input data frame has been processed by multiple devices.
- Each device has sufficient memory to load the trained weights and to perform the task.

The experiment involves testing the impact of varying the number of devices, starting from one device up to five devices, on communication size and latency for distributed heterogeneous edge clusters.

Table 8 lists the different heterogeneous configurations considered for the experiment. In this experiment, we have focused on heterogeneous IoT infrastructure to understand the inconsistency and the impact on inference speed.

The experiment is conducted for splitting the fused layer into 3 × 3, 4 × 4, 5 × 5 partitions by considering the first 16 layers of Yolov2. We started the evaluation of the model with the host device, a gateway device and a single edge device. We noted the communication size and inference latency required for a single data frame with a single edge device. Similarly, we evaluated the model by increasing the number of devices for each partition and observed the output.

Communication size depends on the number of partitions and communication between input and output data. Distributed IoT edge clusters involve communication between the partitioned layers; the proposed system produces better results for the more exemplary partitioning, as shown in Figure 7.

With the varying number of partitions from 3 × 3 to 5 × 5 and an increasing number of devices, we measured the com-

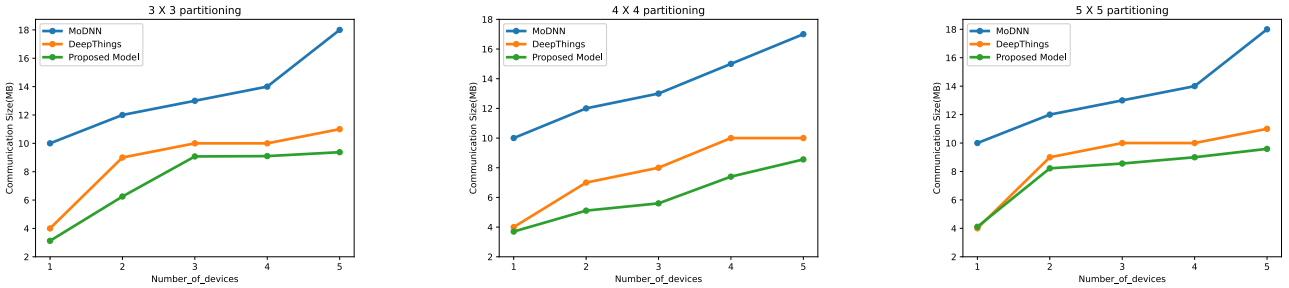

FIGURE 7. Impact of Communication Size on 3 × 3, 4 × 4 and 5 × 5 partitioning. Communication overhead depends on FTP- partitioning and distribution approach. As in FTP only input and output data of each fused tile needs to be communicated. As shown in figure for finer partitions like 3X3 partitioning,communication overhead is slightly larger because of additional overlapped input data during FTP process. In case of MoDNN , in each layer intermediate data is transferred to gateway device further leads to linear increase in communication overhead as number of devices increases from 1 to 5. In case of DeepThings, partitioning method in FTP reduces the communication overhead compared to MoDNN.In general as our proposed model are optimized with reduced number of parameters in the model , it still reduces the communication overhead and maintains an averge of 9.175MB.

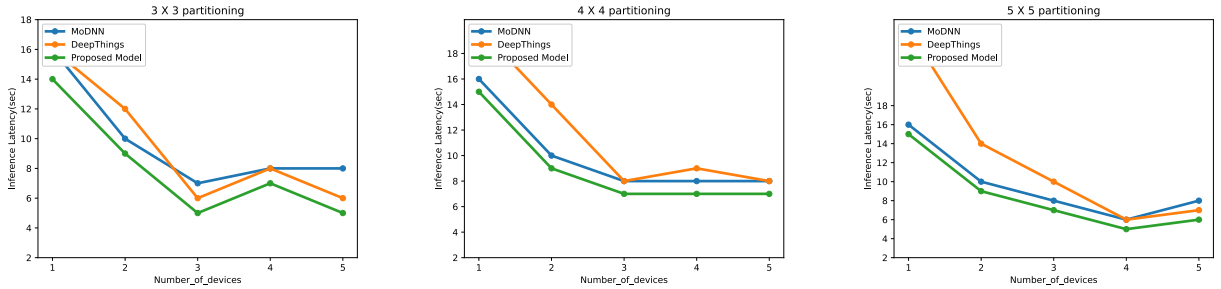

FIGURE 8. Impact of Inference latency on 3 × 3, 4 × 4 and 5 × 5 partitioning. In the case of MoDNN, due to centralized data distribution and process synchronization communication overhead increases and as computation time is inversely proportional, latency will be reduced and increase after third devices. In the case of DeepThings, it adaptively explores the available communication bandwidth and for finer partitioning, up to three devices due to the overlapped partitioning latency being higher and saturated thereafter. Our proposed model performs the best as it does not depend on layer-wise synchronization and is due to the optimized pre-trained model.

munication size. Increasing the number of edge devices will improve the performance. However, it impacts the execution time. As more devices are involved for intelligent computation, the amount of time required for processing is reduced and the lighter way it increases the communication size. The proposed model achieved an accuracy of 92% for the edge devices varying from one to five Raspberry Pi devices.

We can observe from Figure 8. the inference latency on different partitions. The figure depicts that as we increase the number of devices, computation is shared by more devices, which reduces the execution time. The inference latency is compared with DeepThings [36] and MODNN [37] on a single data frame. Our proposed model outperforms inference latency for 3 × 3, 4 × 4 and 5 × 5 partitioning due to the reduced computation as the pre-trained model is optimized. In MODNN, computation time is slightly higher as the number of devices increases due to centralized data distribution, layer-wise synchronization, and processing. In DeepThings, communication size and inference latency are more significant due to partitioning; processing occurred without optimizing the model.

As we have optimized the model and deployed it onto the IoT devices, communication size and inference latency are reduced due to the reduced number of parameters. As a result, the proposed model achieves a minimum communication size of 8.56MB∼9.59MB and inference latency of 5sec∼7sec for 3 × 3 to 5 × 5 fused layer partitioning for up to 5 devices.

TABLE 9. Summarizes the key difference between the present work and existing literature on CNN based edge intelligence from the perspective of model compression and heterogeneous distributed device inference.

Performance metrics	Related work	Techniques
Inference Latency	[36]	FTP, Work sharing and Stealing
Latency and communication Size	[37]	Partitioning the trained DNN model across multiple mobile devices and Parallelization
Latency and communication Size	Our work	Optimized pre-trained model through pruning and CNN portioning and distributing

V. RESULTS AND DISCUSSIONS

To evaluate and validate the proposed system, we have compared the results with few existing models similar to our setup. Table 8 describes the experimental setup, and Table 9 depicts the related work and techniques adopted in the recent research. During the comparison, inference latency is considered for a single data frame as input. Paper [36]

involves fusing of few convolutional layers for reducing the communication overhead. Our proposed model involves optimizing the pre-trained model and then adopting the fusion of layers for the heterogeneous Clusters of edge devices. Finally, in the paper [37] author examined parallelizing deep neural networks on IoT edge devices. However, it includes the transfer of a large number of feature maps across the devices.

The inference latency and communication size are compared with MoDNN and DeepThings. Both the framework does not explore model compression techniques and distributed heterogeneous IoT environment, where each actual physical IoT device may have different operating frequencies and memory size. MoDNN considered smartphones as edge devices, and Lenet, Inception-BN, VGG; deep learning models are considered for evaluation and achieved a 30.02% reduction in delivery time. DeepThings fused only the initial 16 layers of CNN and obtained 1.7× to 3.5× of speedup in CNN inference.

Compared to these two frameworks, our proposed work performs partition and distribution of fused layer for the optimized pre-trained model and obtains communication size of 8.56MB∼9.59MB and inference latency 5sec∼7sec for 3×3 to 5×5 fused layer partitioning for five devices. The proposed framework of optimized heterogeneous edge clusters enables CNN inference on multiple heterogenous devices; hence can be used in the applications such as smart homes, smart cities, competent healthcare, intelligent traffic signal, autonomous vehicles, intelligent drone, and computer vision applications where-in deep learning models are deployed on edge devices such as camera, mobile or any other intelligent devices.

Consider the case study to envision the adoption of the proposed framework for innovative home applications. Smart home consists of multiple intelligent cameras in various places to capture the image. Rather than sending the image to the cloud, the proposed framework will perform on-device computation in real-time. This framework not only reduces communication overheads but can provide faster inference even for distributed and heterogenous devices—the work between the other smart cameras to process the image, reducing the latency in real-time applications.

VI. CONCLUSION AND FUTURE WORK

From our brief review of the background, we learnt that while focusing on Low latency, we could face many demands and challenges. For example, the demand for increased bandwidth can be addressed using mmWave, while parallel and coded computing can address the demand for computing power and task dependency. Furthermore, shifting the computing to edge devices and making them intelligent (capable of machine learning) will reduce propagation delay and prediction delay, respectively. Finally, proactive computing is a promising approach towards reducing propagation delay and making the system energy efficient. In our work, we have shifted computing to the edge, implemented DNN based ML on the edge devices while parallelizing (partitioning) the task and distributing among the edge clusters to minimize propagation delay, prediction delay and power requirement.

In this paper, we have examined a hardware-based prototype and a software framework to optimize the pre-trained model and have designed a lightweight, optimized sub-model of the pre-trained CNN based Yolov2 model. The optimized sub-model is partitioned, and inference is distributed among multiple resource constraint edge devices and IoT gateway devices. Initial layers of CNN highly contribute to overall communication size and inference latency. Hence the first 16 layers of the sub-optimized CNN layers are split into multiple stacks of executable tasks and assigned to multiple IoT devices. The proposed framework obtains the sub-optimal pre-trained model and then splits and distributes CNN parameters into multiple heterogeneous IoT devices. In this process, each partition produces a set of partial output. Finally, partial outputs are collated by the gateway device to produce the final output. Hence for a single input, this approach achieves less inference latency.

To recreate the realistic scenario of clusters of heterogeneous devices (- i.e. resource-constrained IoT devices) in the lab, our model is demonstrated by deploying the model on 5 Raspberry Pi boards with different core frequencies for real-time on-device inference. Evaluation result has shown that our proposed optimized model achieved significant improvement in the result and has got minimum communication size of 8.56MB∼9.59MB, thereby reducing the communication size by ∼ 14.4% and inference latency of 5sec∼7sec, which is a reduction by ∼16% compared to DeepThings for 3×3 to 5×5 fused layer partitioning for five devices. Thus, our model outperforms DeepThings and ModNN with improved inference latency while maintaining significant accuracy and minimal communication size for the popular Yolov2 CNN model.

We further propose to explore federated learning and channel intermittency to achieve reliability and scalability in the proposed work. In future, we intend to define a hardware-software ecosystem capable of hardware aware hyperparameter tuning while being sensitive to the DNN containerization and fault-tolerant. In addition, future researchers can explore potential advantages of Neuromorphic computing such as in-memory computing and event-based spiking NN. This framework is suitable for smart homes, healthcare applications. However, processing various data generated by the devices, such as audio, video, or sensor readings, poses a hurdle. Furthermore, IoT devices running on different operating systems introduces new research challenges.

ACKNOWLEDGMENT

The authors are greatly indebted to the anonymous reviewers whose thought-provoking and encouraging comments have motivated them to modify significantly and update the paper. They also like to express their gratitude to REVA University for extending research facilities to carry out this research. Special thanks to B. U. V. Prashanth for his assistance in the study.

REFERENCES

[1] E. S. Kumar, P. Sachin, B. P. Vignesh, and M. R. Ahmed, "Architecture for IoT based geriatric care fall detection and prevention," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2017, pp. 1099–1104.

[2] R. Rajavel, S. K. Ravichandran, K. Harimoorthy, P. Nagappan, and K. R. Gobichettipalayam, "IoT-based smart healthcare video surveillance system using edge computing," *J. Ambient Intell. Hum. Comput.*, vol. 5, pp. 1–13, Mar. 2021.

[3] (May 2021). *Edge Computing Market Worth $61.14 Billion By 2028*. Accessed: Jul. 15, 2021. [Online]. Available: https://www.grandviewresearch.com/press-release/global-edge-computing-market

[4] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," 2019, *arXiv:1908.00080*.

[5] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 553–559.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[7] J. Lee and K.-I. Hwang, "YOLO with adaptive frame control for real-time object detection applications," *Multimedia Tools Appl.*, 2021, doi: 10.1007/s11042-021-11480-0.

[8] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[9] Y. Li, F. Qi, Z. Wang, X. Yu, and S. Shao, "Distributed edge computing offloading algorithm based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 85204–85215, 2020, doi: 10.1109/ACCESS.2020.2991773.

[10] L. Hu, G. Sun, and Y. Ren, "CoEdge: Exploiting the edge-cloud collaboration for faster deep learning," *IEEE Access*, vol. 8, pp. 100533–100541, 2020, doi: 10.1109/ACCESS.2020.2995583.

[11] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[12] L. Zhou *et al.*, "Distributing deep neural networks with containerized partitions at the edge," in *Proc. 2nd USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2019. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/zhou

[13] Z. Zhao, K. Wang, N. Ling, and G. Xing, "EdgeML: An AutoML framework for real-time deep learning on the edge," in *Proc. Int. Conf. Internet-of-Things Design Implement.*, May 2021, pp. 133–144.

[14] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *Proc. Int. Conf. Embedded Comput. Syst.* Cham, Switzerland: Springer, 2019, pp. 77–90.

[15] D. Gupta, O. Kayode, S. Bhatt, M. Gupta, and A. S. Tosun, "Learner's dilemma: IoT devices training strategies in collaborative deep learning," in *Proc. IEEE 6th World Forum Internet Things (WF-IoT)*, Jun. 2020, pp. 1–6.

[16] I. Jang, H. Kim, D. Lee, Y.-S. Son, and S. Kim, "Knowledge transfer for on-device deep reinforcement learning in resource constrained edge computing systems," *IEEE Access*, vol. 8, pp. 146588–146597, 2020, doi: 10.1109/ACCESS.2020.3014922.

[17] G. White and S. Clarke, "Urban intelligence with deep edges," *IEEE Access*, vol. 8, pp. 7518–7530, 2020, doi: 10.1109/ACCESS.2020.2963912.

[18] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported Internet of Things," *IEEE Access*, vol. 7, pp. 69194–69201, 2019, doi: 10.1109/ACCESS.2019.2919736.

[19] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 2407–2416.

[20] H. Yi, H. Jung, and S. Bae, "Deep neural networks for traffic flow prediction," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2017, pp. 328–331.

[21] C.-H. Yu, C.-N. Chou, and E. Chang, "Distributed layer-partitioned training for privacy-preserved deep learning," in *Proc. IEEE Conf. Multimedia Inf. Process. Retr. (MIPR)*, Mar. 2019, pp. 343–346.

[22] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Apr. 2018, pp. 184–190.

[23] Y. Li, Z. Han, Q. Zhang, Z. Li, and H. Tan, "Automating cloud deployment for deep learning inference of real-time online services," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 1668–1677.

[24] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 1049–1062.

[25] M. Z. Khan, S. Harous, S. U. Hassan, M. U. G. Khan, R. Iqbal, and S. Mumtaz, "Deep unified model for face recognition based on convolution neural network and edge computing," in *IEEE Access*, vol. 7, pp. 72622–72633, 2019, doi: 10.1109/ACCESS.2019.2918275.

[26] S. U. Amin and M. S. Hossain, "Edge intelligence and Internet of Things in healthcare: A survey," *IEEE Access*, vol. 9, pp. 45–59, 2021, doi: 10.1109/ACCESS.2020.3045115.

[27] Y. Yan, Q. Pei, and H. Li, "Privacy-preserving compressive model for enhanced deep-learning-based service provision system in edge computing," *IEEE Access*, vol. 7, pp. 92921–92937, 2019, doi: 10.1109/ACCESS.2019.2927163.

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–13.

[29] K. Bhardwaj, C.-Y. Lin, A. Sartor, and R. Marculescu, "Memory- and communication-aware model compression for distributed deep learning inference on IoT," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–22, Oct. 2019.

[30] E. Baccarelli, M. Scarpiniti, A. Momenzadeh, and S. S. Ahrabi, "Learning-in-the-fog (LiFo): Deep learning meets fog computing for the minimum-energy distributed early-exit of inference in delay-critical IoT realms," *IEEE Access*, vol. 9, pp. 25716–25757, 2021, doi: 10.1109/ACCESS.2021.3058021.

[31] 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*.

[32] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–17.

[33] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-Step pruning," 2019, *arXiv:1903.03472*.

[34] B. Liu, Y. Cai, Y. Guo, and X. Chen, "TransTailor: Pruning the pre-trained model for improved transfer learning," 2021, *arXiv:2103.01542*.

[35] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[36] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018, doi: 10.1109/TCAD.2018.2858384.

[37] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1396–1401, doi: 10.23919/DATE.2017.7927211.

[38] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, Nov. 2019, pp. 195–208.

[39] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "DeeperThings: Fully distributed CNN inference on resource-constrained edge devices," *Int. J. Parallel Program.*, vol. 49, pp. 1–25, Apr. 2021, doi: 10.1007/s10766-021-00712-3.

[40] Q. Liu, T. Han, N. Zhang, and Y. Wang, "DeepSlicing: Deep reinforcement learning assisted resource allocation for network slicing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6, doi: 10.1109/GLOBECOM42002.2020.9322106.

[41] S. A. Hossain, M. A. Rahman, and M. A. Hossain, "Edge computing framework for enabling situation awareness in IoT based smart city," *J. Parallel Distrib. Comput.*, vol. 122, pp. 226–237, Dec. 2018.

[42] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for Internet of Things data analysis: A survey," *Digit. Commun. Netw.*, vol. 4, no. 3, pp. 161–175, Aug. 2018.

[43] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, Jun. 2018.

[44] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, Nov. 2014.
[45] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge AI: Algorithms and systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2167–2191, 2020.
[46] H. Zeyu, X. Geming, W. Zhaohang, and Y. Sen, "Survey on edge computing security," in *Proc. Int. Conf. Big Data, Artif. Intell. Internet Things Eng. (ICBAIE)*, Jun. 2020, pp. 96–105.
[47] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, Apr. 2020.
[48] A. Morshed, P. P. Jayaraman, T. Sellis, D. Georgakopoulos, M. Villari, and R. Ranjan, "Deep osmosis: Holistic distributed deep learning in osmotic computing," *IEEE Cloud Comput.*, vol. 4, no. 6, pp. 22–32, Nov. 2017.
[49] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–2.
[50] Q. D. La, M. V. Ngo, T. Q. Dinh, T. Q. S. Quek, and H. Shin, "Enabling intelligence in fog computing to achieve energy and latency reduction," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 3–9, Feb. 2019.
[51] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.
[52] C. She, R. Dong, Z. Gu, Z. Hou, Y. Li, W. Hardjawana, C. Yang, L. Song, and B. Vucetic, "Deep learning for ultra-reliable and low-latency communications in 6G networks," *IEEE Netw.*, vol. 34, no. 5, pp. 219–225, Sep./Oct. 2020.
[53] J. Shao and J. Zhang, "Communication-computation trade-off in resource-constrained edge inference," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 20–26, Dec. 2020.
[54] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[55] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Hierarchical quantized federated learning: Convergence analysis and system design," 2021, *arXiv:2103.14272*.
[56] C. Garvey, "A framework for evaluating barriers to the democratization of artificial intelligence," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 1–2.
[57] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2595–2621, 2018.
[58] F. M. C. D. Oliveira and E. Borin, "Partitioning convolutional neural networks to maximize the inference rate on constrained IoT devices," *Future Internet*, vol. 11, no. 10, p. 209, Sep. 2019.
[59] T. Yang, H. Feng, S. Gao, Z. Jiang, M. Qin, N. Cheng, and L. Bai, "Two-stage offloading optimization for energy–latency tradeoff with mobile edge computing in maritime Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5954–5963, Jul. 2019.
[60] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, "Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems," *IEEE Access*, vol. 8, pp. 212177–212193, 2020, doi: 10.1109/ACCESS.2020.3039714.
[61] A. Nazir, R. N. Mir, and S. Qureshi, "Exploring compression and parallelization techniques for distribution of deep neural networks over edge–fog continuum—A review," *Int. J. Intell. Comput. Cybern.*, vol. 13, no. 3, pp. 331–364, Jun. 2020.
[62] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12, doi: 10.1109/MICRO.2016.7783725.
[63] Veeramanikandan, S. Sankaranarayanan, J. J. P. C. Rodrigues, V. Sugumaran, and S. Kozlov, "Data flow and distributed deep neural network based low latency IoT-edge computation model for big data environment," *Eng. Appl. Artif. Intell.*, vol. 94, Sep. 2020, Art. no. 103785.
[64] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. H. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Commun. Mag.*, vol. 58, no. 10, pp. 88–93, Oct. 2020.

SOUMYALATHA NAVEEN (Student Member, IEEE) received the M.Tech. degree from Visvesvaraya Technological University, India. She is currently a Research Scholar with the School of Computer Science and Engineering and an Assistant Professor with the School of Multidisciplinary Studies, REVA University, Bengaluru, India. Her main research interests include edge computing, the Internet of Things, deep learning, intelligent IoT systems, and optimization at resource constraint IoT device.

MANJUNATH R. KOUNTE (Senior Member, IEEE) received the bachelor's degree in electronics and communication engineering and the master's degree in computer network engineering from Visvesvaraya Technological University, Belgaum, India, and the Ph.D. degree in the domain of machine vision from JAIN University, Bengaluru, in 2017. He is currently serving as an Associate Professor and the Head for the Department of Electronics and Computer Engineering, School of Electronics and Communication Engineering, REVA University, Bengaluru. He has more than 13 years of teaching, research, and administrative experience. He has published one book and more than 40 peer reviewed publications, including 12 journals. He is a reviewer of various reputed international journals. He is also the Head of the Machine Learning and Blockchain Laboratory, REVA University, India. His research interests include machine learning, video processing, IoT edge computing, and blockchain technologies.

MOHAMMED RIYAZ AHMED (Senior Member, IEEE) received the B.E. and M.Tech. degrees in electronics and communication engineering and computer networking from Visvesvaraya Technological University, Belgaum, India, in 2007 and 2010, respectively, and the Ph.D. degree in electronics and communication engineering from JAIN University, Bengaluru, India, in 2016.

Since 2011, he has been with REVA University, Bengaluru, where he is currently an Associate Professor and an Assistant Director with the School of Multidisciplinary Studies. He is an Advisor of IEEE EMBS Student Chapter and a Mentor to IEEE ComSoc Student Chapter at REVA University. He is also the founder and the PI of the CPS Laboratory, REVA University. He is an advisor and the consultant to a number of higher educational institutes for their transformation into an Entrepreneurial University for knowledge economy. He is involved in investigation and teaching of design thinking as an enabler for individual, institutional, and international sustainable development. He has published more than 100 research papers in refereed journals and international conferences and as an invited speaker. His research interests include the computational cognitive neuroscience, WSNs, 5G and beyond, genomic data sequencing, tribology, neuromorphic engineering, spintronics, the IoT and edge computing, and green technologies. He has served as the TPC Member for IEEE ICAECC Conference and a technical reviewer for seven IEEE journals and transactions along with numerous achieved journals and conference proceedings.