LOW LATENCY DEEP LEARNING INFERENCE MODEL FOR DISTRIBUTED INTELLIGENT IOT EDGE CLUSTERS

SOUMYALATHA NAVEEN\textsuperscript{1}, (Student Member, IEEE), MANJUNATH R KOUNTE\textsuperscript{2}, (Senior Member, IEEE) MOHAMMED RIYAZ AHMED\textsuperscript{3}, (Senior Member, IEEE)

\textsuperscript{1}School of Computer Science and Engineering, REVA University, Bangalore, Karnataka, India (e-mail: soumyanaveen.u@gmail.com)
\textsuperscript{2}School of Electronics and Communication Engineering, REVA University, Bangalore, Karnataka, India (e-mail: manjunath.kounte@gmail.com)
\textsuperscript{3}School of Multidisciplinary Studies, REVA University, Bangalore, Karnataka, India (e-mail: informriyaz@gmail.com)

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

\textbf{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 X 5 fused layer partitioning for five devices, our proposed model reduces communication size by \(\sim 14.4\%\) and inference latency by \(\sim 16\%\).

\textbf{INDEX TERMS} Convolutional Neural Network, Deep Learning, Distributed Intelligence, Edge Computing, Fog Computing, Heterogeneous devices, Inference Model, IoT Clusters, Low Latency

I. INTRODUCTION

\textbf{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 I depicts one of the mission-critical application of 'Geriatric Care'. Unfortunately, due to the 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. We
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.

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 localhost [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 study 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 requires enormous computational resources and leaves 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 men-
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 3.3X 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 |
| [39] | 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 5.97X |

tioned above in many ways; most of the works 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 discussion 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 [59] 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 [60] 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

---

**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                 |
| FD      | 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                |

---

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see [https://creativecommons.org/licenses/by/4.0/]
create an ecosystem for achieving inference models with excellent performance. In this regard, many attempts have been reported to optimize the DNN models at edge devices [61], [62]. 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.

The 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 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 [63]. 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

\[ N = [D,C] \]  

(1)

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

| Symbol | Meaning |
|--------|---------|
| D      | Set of devices |
| ED     | Edge devices |
| n      | Number of devices |
| C      | Connection |
| EG     | Edge Gateway |
| \( \lambda \) | Latency |
| \( \lambda_{nl} \) | Network latency |
| \( \lambda_{cl} \) | Inference latency |
| \( \alpha \) | Task assigning indicator |
| \( l_t \) | Computational Complexity |
| \( z_t \) | Computational Capability |
| f      | Number of Filters |
| \( W^s \) | Optimized pre-trained model of the source model |
| \( W^f \) | Pre-trained model with f number of filters optimized for source |
| \( f^* \) | The optimized filter output after pruning |
| q      | Pruning ratio |

Set of devices in the network is represented as

\[ D = [ED_t, EG_t, Host] \]  

(2)

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

\[ |C| = 0 < |C| < \frac{n(n-1)}{2} \]  

(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 \( \lambda \) and this contains Computational latency \( \lambda_{cl} \) and network latency \( \lambda_{nl} \) [64].

\[ \lambda = \lambda_{cl} + \lambda_{nl} \]  

(4)

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

\[ \lambda = \lambda_{cl} \]  

(5)

The Computational latency \( \lambda_{cl} \) 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 (\( \alpha \)), is estimated as

\[ \lambda_{cl} = \alpha \frac{l_t}{z_t} \]  

(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(\( \lambda_{il} \)) [65] estimation for the IoT edge clustering network is given by

\[
\lambda_{il} = \lambda_{cl}
\]  

(7)

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. HETEROGENEOUS EDGE CLUSTERS

The distributed heterogeneous 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 X 608 X 3, which is width, height and three RGB channels, respectively, as input to Yolov2. As shown in Table 6 Yolov2 detection network, layer 1 uses 32 filters and hence convolutional layer computes output as 608 X 608 X 32. Furthermore, layer 2 Maxpooling performs a downsampling 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 X 19 X 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.

#### TABLE 6. Detailed Yolov2 detection network architecture for feature extraction.

In our work, Yolov2 pre-trained model is used. Yolov2 uses Darknet-19 architecture a open source neural network framework. In the Table the repeated 1 X 1 convolution enables to reduce the number of parameters. In the Table Conv-layer indicates the Convolutional Layer.

| Layer      | Filter | Size X Side  | 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 |
| Maxpooling |        | 2 X 2/2      | 152 X 152 X 64 | 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 256 |
| Conv-layer | 256    | 3 X 3/1      | 76 X 76 X 256 | 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 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 256 | 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 1024 |
| Conv-layer | 1024   | 3 X 3/1      | 19 X 19 X 1024 | 19 X 19 X 1024 |
| Conv-layer | 512    | 1 X 1/1      | 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 |
| Conv-layer | 512    | 1 X 1/1      | 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 |
| Conv-layer | 512    | 1 X 1/1      | 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 |
| Conv-layer | 512    | 1 X 1/1      | 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 |

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
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 reduce the network parameters to reduce the model’s size and make it suitable for deployment on 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, $W_{sf}$ represents the optimized pre-trained model of the source model. $W^f_s$ 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. $\ell()$ represents the loss function during the optimization of the network. 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

$$\ell(D_s|W^f_s)$$
While optimizing the pre-trained model, Loss function $\mathcal{L}()$ is defined as minimizing the loss on source data $D_s$ of pre-trained model $W^*_s$ to loss on source data of optimized pre-trained model with respect to accuracy.

To generate the optimized sub-model $W^*_{f_\cdot}$ 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.

$$f^* = f \cdot q\%$$ (9)

$W^*_{f_\cdot}$, the optimized pre-trained model of the source model is evaluated as

$$W^*_{f_\cdot} = \min\{(\mathcal{L}(D_s|W^*_s) - \mathcal{L}(D_s|W^*_{f_\cdot}))\}$$ (10)

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 \cite{63} 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 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 $m=1,2,...M$ with input dimension $W_{m-1} \times X$
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 $W_f$, pre-trained weights $W$, source data $D_m$, loss function $\mathcal{L}()$, Optimized sub model of pre-trained model $W'_f$

1. pruningPerc=0.30
2. threshold=percentile(sum(W), pruningPerc)
3. WeightPrune($W'_f$, pruningPerc)
4. for $i = 1$ to modelweight do
5. \hspace{1em} if ($i \geq$ threshold) then
6. \hspace{2em} $i = i$
7. \hspace{1em} else
8. \hspace{2em} $i = 0$
9. \hspace{1em} end if
10. end for
11. Retrain the pruned model until
12. Calculate the loss function using equation 10
13. if ($\mathcal{L}$ function $\geq$T) then
14. \hspace{1em} break
15. else
16. $W'_f=W_f$
17. end if
18. return optimized model $W'_f$.

$HM-1$ 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 $5 \times 5$ tiles. Thus, only $5 \times 5$N input data are brought from memory. Next, the convolution operation is performed on Layer1 with a $3 \times 3$N filter producing a $3 \times 3$M intermediate feature map. Finally, the input is obtained to Layer 2, which convolves with $P$ filters, and produces $1 \times 1$M 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 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
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 5X5 feature maps. This feature map is convolved with 3X3 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.

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 NCS2 and a Host machine. The edge devices used for evaluation are Raspberry Pi 3 Model B+ with 1GB RAM, Raspberry Pi 4 Model B with 2GB 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 as shown in Figure 6.

### 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          |

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 Computer Stick 2 has been used to accelerate the Gateway devices for faster inference.
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 X 3, 4 X 4, 5 X 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 X3 to 5 X 5 and an increasing number of devices, we measured the communication 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 MOДNN [37] on a single data frame. Our proposed model outperforms inference latency for 3 x 3, 4 x 4 and 5 x 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 3X3 to 5X5 fused layer partitioning for up to 5 devices.

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 1.7X to 3.5X 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 X 3 to 5 X 5 fused layer partitioning for five devices. The proposed framework of optimized heterogeneous edge clusters enables CNN inference on multiple heterogeneous 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.
multiple resource constraint edge devices and IoT gateway sub-model is partitioned, and inference is distributed among the pre-trained CNN based Yolov2 model. The optimized model and have designed a lightweight, optimized sub-model that minimizes the overall communication size and inference latency. This allows for the efficient distribution of tasks and predictions among edge devices, with a focus on reducing propagation delay and prediction latency, respectively. Finally, proactive computing is a promising approach towards reducing propagation delay and prediction delay, respectively. 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 the 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 X 3 to 5 X 5 fused layer partitioning for five devices. Thus, our

VI. CONCLUSION AND FUTURE WORK

From our brief review in the background section, we learnt that while focusing on Low latency, we 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 the 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 the single input, this approach achieves less inference latency.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3131396, IEEE Access

VOLUME x, xxxx
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 the 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 further 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. The authors also like to express their gratitude to REVA University for extending the research facilities to carry out this research. Special thanks to Mr B U V Prashanth for his assistance in the study.

REFERENCES
[1] Kumar, E. Sathish, et al. "Architecture for IOT based geriatric care fall detection and prevention." 2017 International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2017.
[2] Rajavel, Rajkumar, et al. "IoT-based smart healthcare video surveillance system using edge computing." Journal of Ambient Intelligence and Humanized Computing (2021): 1-13.
[3] Edge Computing Market Worth $61.14 Billion By 2028, Updated on: May 2021 and Accessed on: 15-07-2021. [Online]. Available: https://www. grandviewresearch.com/press-release/global-edge-computing-market
[4] M.G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, Faraz Hussain, “Machine Learning at the Network Edge: A Survey”, arXiv, May 2021
[5] Marchisio, Alberto, et al. "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges." 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2019.
[6] Zhou, Zhi, et al. "Edge intelligence: Paving the last mile of artificial intelligence with edge computing." in Proceedings of the IEEE, 107.8 (2019): 1738-1762.
[7] Jeonghun Lee, Kwang il Hwang, "YOLO with adaptive frame control for real time object detection applications", Springer, September 2021.
[8] Wang, Xiaofei, et al. "Convergence of edge computing and deep learning: A comprehensive survey." IEEE Communications Surveys & Tutorials 22.2 (2020): 869-904.
[9] Y. Li, F. Qi, Z. Wang, X. Yu and S. Shao, "Distributed Edge Computing Offloading Algorithm Based on Deep Reinforcement Learning," in IEEE Access, vol. 8, pp. 85204-85215, 2020, doi: 10.1109/ACCESS.2020.2991773.
[10] L. Hu, G. Sun and R. Ren, "CoEdge: Exploiting the Edge-Cloud Collaboration for Faster Deep Learning," in IEEE Access, vol. 8, pp. 100533-100541, 2020, doi: 10.1109/ACCESS.2020.2995583.
[11] Chen, Jiasi, and Xukan Ran. "Deep Learning With Edge Computing: A Review." Proc. IEEE 107.8 (2019): 1655-1674.
[12] Zhou, Li, et al. "Distributing deep neural networks with containerized partitions at the edge." 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019.
[13] Zhuo Zhao, Kai Wang, Weiren Ling, Gaoliang Xing, “EdgeML: An AutoML Framework for Real-Time Deep Learning on the Edge” IOTDI ’21: Proceedings of the International Conference on Internet-of-Things Design and Implementation, May 2021
[14] Stahl, Rafael, et al. "Fully distributed deep learning inference on resource-constrained edge devices." International Conference on Embedded Computer Systems. Springer, Cham, 2019.
[15] Gupta, Deepii, et al. "Learner’s Dilemma: IoT Devices Training Strategies in Collaborative Deep Learning." 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). IEEE, 2020.
[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,” in 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,” in 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," in IEEE Access, vol. 7, pp. 69194-69201, 2019, doi: 10.1109/ACCESS.2019.2919736.
[19] Wang, Ji, et al. "Not just privacy: Improving performance of private deep learning in mobile cloud." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2018.
[20] Yi, Hongsuk, HeeJin Jung, and Sanghoon Bae. "Deep neural networks for traffic flow prediction." 2017 IEEE international conference on big data and smart computing (BigComp). IEEE, 2017.
[21] Yu, Chun-Hsien, Chun-Nan Chou, and Emily Chang. "Distributed layer-partitioned training for privacy-preserved deep learning." 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR). IEEE, 2019.
[22] Guo, Tian. "Cloud-based or on-device: An empirical study of mobile deep inference." 2018 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2018.
[23] Li, Yang, et al. "Automating cloud deployment for deep learning inference of real-time online services." IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020.
[24] Zhang, Chengliang, et al. "Mark: Exploiting cloud services for cost-effective, sloppy machine learning inference serving." 2019 USENIX Annual Technical Conference (USENIXATC 19). 2019.
[25] M. Z. Khan, S. Harous, S. U. Hassan, M. U. Ghani Khan, R. Iqbal and S. Mumtaz, "Deep Neural Network and Edge Computing", in IEEE Access, vol. 7, pp. 72622-72633, 2019, doi: 10.1109/ACCESS.2019.2918275.
[26] I. Amin and M. S. Hossain, "Edge Intelligence and Internet of Things in Healthcare: A Survey," in IEEE Access, vol. 9, pp. 45-59, 2021, doi: 10.1109/ACCESS.2020.3045115.
[27] Y. Yan, Q. Pei and H. Li, "Privacy-Preserving Convex Model for Enhanced Deep-Learning-Based Service Provision System in Edge Computing," in IEEE Access, vol. 7, pp. 92921-92937, 2019, doi: 10.1109/ACCESS.2019.2927163.
[28] Hao Li, Asim Kadav, Igor Durandovic, Hanan Samet, Hans Peter Graf, "Pruning Filters for Efficient Convnets", 5th International Conference on Learning Representations(ICLR),2017
[29] Kartikeya Bhardwaj, Ching-Yi Lin, Anderson Sartor, Radu Marculescu, "Memory- and Communication-Aware Model Compression for Distributed Deep Learning Inference on IoT", in ACM Transactions on Embedded Computing Systems., article no.82, vol.18, no.5s, Oct 2019.
[30] E. Baccarelli, M. Scarpiniti, A. Momenzadeh and S. S. Abrabi, "Learning-in-the-Fog: Deep Learning Meets Fog Computing for the Minimum-Energy Distributed Early-Exit of Inference in Delay-Critical IoT Realms," in IEEE Access, vol. 9, pp. 25716-25725, 2021, doi: 10.1109/ACCESS.2021.3058021.
[31] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).
[32] Pavlo Molchanov, Stephen Tyree, Teno Karras, Timo Aila, Jan Kautz,"Pruning Convolutional Neural Networks for Resource Efficient Inference", 5th International Conference on Learning Representations(ICLR),2017
Mach, Pavel, and Zdenek Becvar. "Mobile edge computing: A survey on
Shao, Jiawei, and Jun Zhang. "Communication-computation trade-off in
Huang, Yutao, et al. "When deep learning meets edge computing." 2017
Merenda, Massimo, Carlo Porcaro, and Demetrio Iero. "Edge machine
Zeyu, Huang, et al. "Survey on edge computing security." 2020 Inter-
Shi, Yuanming, et al. "Communication-efficient edge AI: Algorithms and
Hossain, SK Alamgir, Md Anisur Rahman, and M. Anwar Hossain. "Edge
Li Zhou, Mohammad Hossein Samavatian, Anys Bacha, Saikat Majumdar,
Stahl, R., Hoffman, A., Mueller-Gritschneider, D. et al., "DeeperThings:
Li, Xiao, C. Yang, W. Wang. "Slicing: Deep Reinforcement Learning Assisted Resource Allocation for Network Slicing," in GLOBE-
Qiu, Jun, Tuan, N. Zhang and Wang. "DeepSlicing: Deep Reinforcement Learning Assisted Resource Allocation for Network Slicing," in GLOBE-
Hossain, SK Alamgir, Md Ansar Rahman, and M. Anwar Hossain. "Edge computing framework for enabling situation awareness in IoT based smart
eh, Mohammad Saeid, et al. "Machine learning for Internet of Things data analysis: A survey." Digital Communications and Networks 4.3 (2018): 161-175.
Mehrdi, et al. "Deep learning for IoT big data and streaming analytics: A survey." IEEE Communications Surveys & Tutorials 20.4 (2020): 2923-2960.
Barbosa, Sergio, Stefania Sardellitti, and Paolo Di Lorenzo. "Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks." IEEE Signal Processing Magazine 31.6 (2014): 45-55.
Shi, Yuanming, et al. "Communication-efficient edge AI: Algorithms and systems." IEEE Communications Surveys & Tutorials 22.4 (2020): 2167-
Zeyu, Huang, et al. "Survey on edge computing security." 2020 Intern-
Merenda, Massimo, Carlo Porcaro, and Demetrio Iero. "Edge machine learning for ai-enabled iot devices: A review." Sensors 20.9 (2020): 2533.
Morshed, Ahsan, et al. "Deep osmosis: Holistic distributed deep learning in osmotic computing." IEEE Cloud Computing 4.6 (2017): 22-32.
Huang, Yutao, et al. "When deep learning meets edge computing." 2017 IEEE 15th international conference on network protocols (ICNP). IEEE, 2017.
La, Quang Duy, et al. "Enabling intelligence in fog computing to achieve energy and latency reduction." Digital Communications and Networks 5.1 (2019): 3-9.
Li, He, Kaoru Ota, and Mianxiong Dong. "Learning IoT in edge: Deep learning for the Internet of Things with edge computing." IEEE network 32.1 (2018): 96-101.
She, Changyang, et al. "Deep learning for ultra-reliable and low-latency communications in 6G networks." IEEE Network 34.5 (2020): 219-225.
Shao, Jiawei, and Jun Zhang. "Communication-computation trade-off in resource-constrained edge inference." IEEE Communications Magazine 58.12 (2020): 20-26.
Mach, Pavel, and Zdenek Becvar. "Mobile edge computing: A survey on architecture and computation offloading." IEEE Communications Surveys & Tutorials 19.3 (2017): 1628-1656.
Liu, Lumin, et al. "Hierarchical Quantized Federated Learning: Conver-
Garvey, Colin. "A framework for evaluating barriers to the democratization of artificial intelligence.” Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
Mao, Qian, Fei Hu, and Qi Hao. "Deep learning for intelligent wireless networks: A comprehensive survey." IEEE Communications Surveys & Tutorials 20.4 (2018): 2595-2621.
Martins Campos de Oliveira, Fabiola, and Edson Borin. "Partitioning convolutive neural networks to maximize the inference rate on constrained IoT devices." Future Internet 11.10 (2019): 209.
Marchisio, Alberto, M. Baggio, L. Franceschini, and V. Prandoni. "Federated learning for edge computing: Current trends, cross-layer optimizations, and open research challenges.” 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2019.
Yang, Tingting, et al. "Two-stage offloading optimization for energy–latency tradeoff with mobile edge computing in maritime Internet of Things." IEEE Internet of Things Journal 7.3 (2019): 5954-5963.
Y. Matsubara, D. Callegaro, S. Baduya, M. Levorato and S. Singh, “Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems,” in IEEE Access, vol. 8, pp. 212177-212193, 2020, doi: 10.1109/ACCESS.2020.3039714.
Nazir, Azra, Roozie Niaz Mir, and Shaima Qureshi. “Exploring compression and parallelization techniques for distribution of deep neural networks over Edge–Fog continuum—a review.” International Journal of Intelligent Computing and Cybernetics (2020).
M. Alwani, H. Chen, M. Ferdman and P. Milder, “Fused-layer CNN accelerators,” in 2016 49th Annual IEEE/ACM International Symposi-
um on Microarchitecture (MICRO), 2016, pp. 1-12, doi: 10.1109/MIC-
CRO.2016.7783725.
Sankaranarayanan, Suresh, et al. “Data Flow and Distributed Deep Neural Network based low latency IoT-Edge computation model for big data environment.” Engineering Applications of Artificial Intelligence, 94 (2020): 103785.
Khun, Latif U., et al. “Federated learning for edge networks: Resource op-
timization and incentive mechanism.” IEEE Communications Magazine, 58.10 (2020): 88-93.

SOUMYALATHA NAVEEN received the M.Tech degree from Visvesvaraya Technological University, India. She is an Assistant professor in School of Multidisciplinary Studies, REVA University, Bangalore, India. Her main research interest includes edge computing, Internet of Things, Deep learning, Intelligent IoT Systems, Optimization at resource constraint IoT device.

MANJUNATH R KOUNTE (Senior Member, IEEE) received his Bachelors in Electronics and Communication Engineering from Visvesvaraya Technological University, Belgaum, India and master’s in Computer Network Engineering from Visvesvaraya Technological University, Belgaum, India and PhD in the domain of Machine Vision from JAIN University, Bengaluru in 2017. He is currently serving as Associate Professor and Head, Department of Electronics and Computer Engineering in REVA University, Bengaluru. He has over 13+ years of Teaching, Research and Administrative experience. He has published one book and 40+ peer reviewed publications including 12 Journals. He is reviewer of various reputed International Journals. He is also heading Machine Learning and Blockchain Lab, REVA University, India. His research interests include Machine learning, Video processing, IoT Edge Computing, Blockchain technologies.
MOHAMMED RIYAZ AHMED received the B.E. and M. Tech. degree in electronics and communication engineering and computer networking from Visvesvaraya Technological University, Belgaum, India, 2007 and 2010, respectively. He received the Ph.D. degree in electronics and communication engineering from the JAIN University, Bengaluru, India, in 2016. Since 2011, he has been with REVA University, Bengaluru, India, where he is currently an Associate Professor and Assistant Director of School of Multidisciplinary Studies. He is advisor of IEEE EMBS student chapter and mentor to IEEE ComSoc student chapter at REVA University. He has served as TPC member of IEEE ICAECC conference, and technical reviewer for 7 IEEE journals and transactions along with numerous achieved journals and conference proceedings.

His research interests include the computational cognitive neuroscience, WSNs, 5G and beyond, Genomic data sequencing, tribology, neuromorphic engineering, spintronics, IoT and edge computing, and green technologies. 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, international conferences, and as invited speaker. He is also the founder and PI of CPS lab at REVA University. Riyaz is the advisor and the consultant to a number of Higher Educational Institutes for their transformation into an Entrepreneurial University for Knowledge Economy.

***