Designing and Training of Lightweight Neural Networks on Edge Devices using Early Halting in Knowledge Distillation

Rahul Mishra, Member IEEE and Hari Prabhat Gupta, Senior Member IEEE

***Abstract*—Automated feature extraction capability and significant performance of Deep Neural Networks (DNN) make them suitable for Internet of Things (IoT) applications. However, deploying DNN on edge devices becomes prohibitive due to the colossal computation, energy, and storage requirements. This paper presents a novel approach, EarlyLight, for designing and training lightweight DNN using large-size DNN. The approach considers the available storage, processing speed, and maximum allowable processing time to execute the task on edge devices. We present a knowledge distillation based training procedure to train the lightweight DNN to achieve adequate accuracy. During the training of lightweight DNN, we introduce a novel early halting technique, which preserves network resources; thus, speedups the training procedure. Finally, we present the empirically and real-world evaluations to verify the effectiveness of the proposed approach under different constraints using various edge devices.**

***Index Terms*—Deep neural networks, early halting, edge devices, knowledge distillation.**

I. Introduction

Internet of Things (IoT) applications use sensors that generate a large amount of sensory data to perform a given task of real-time monitoring and detection [1], [2]. In time-critical IoT applications such as fire or gas leakage detection in industrial warehouses, the sensory data processing must be completed within a specific time limit from its occurrence. Such time interval is referred to as Maximum Allowable Processing (MAP) time. Further, the edge devices are usually battery operated and smaller in size, having limited storage and processing capacity. Due to the limited storage and processing, such edge devices delayed the task execution [3], [4]. It creates a vulnerable research challenge to execute a time-critical task within MAP time on a resource constraint edge device.

Moreover, the high accuracy and automated feature extraction capability of Deep Neural Networks (DNN) make them suitable for IoT applications. However, the deployment of DNN on edge devices becomes prohibitive due to the excessive demand for resources [5]. Generally, the resources include storage and processing capacity. Next, a DNN compression technique transforms large-size DNN to lightweight for edge devices without significantly reducing performance [6]. A lightweight DNN has fewer parameters and can run on an edge device within limited storage. Further, lightweight DNN also reduces the inference time. Most of the existing compression techniques compress DNN up to a certain percentage without simultaneously considering the available resources of edge devices, desired accuracy, and MAP time of the task.

Knowledge Distillation (KD) is a concept that improves the performance of the lightweight DNN using the generalization ability of the large-size DNN [7]. KD uses keywords teacher and student for large-size and lightweight DNN, respectively, where prior transfers knowldege to the latter. It trains a student under the guidance of a teacher. Most of the existing KD approaches utilized the knowledge limited to the pre-trained teacher model and did not consider the knowledge from the training process of the teacher model. Different from the existing work, Zhou *et al.* [8] employed the concept of using two teachers, *i.e.,* scratch and pre-trained. Scratch teacher compels the student to follow an optimal path towards achieving final logits. A pre-trained teacher helps in avoiding the loss due to random initialization. The authors in [9] proposed a framework, where a large-size DNN supervised the whole training process of lightweight DNN. The lightweight DNN shared parameters with large-size DNN to get low-level representation from the large-size. The main limitation of the prior work [8], [9] were not to considered the constraints of the edge devices while designing and training lightweight DNN. Furthermore, using multiple teachers [8] throughout the training of the student consumes huge resources.

In this paper, we assume a given large-size DNN that can process a task successfully. However, it requires higher storage and processing time. Therefore, we propose an approach, namely EarlyLight, to design a lightweight DNN using a large-size DNN that can process the task in MAP time on edge devices. Next, to achieve higher accuracy using lightweight DNN, we present a knowledge distillation based lightweight DNN training scheme. The scheme introduces a novel early halting technique that significantly reduces training time and required resources. Specifically, we address the problem of designing and training a lightweight DNN using a given large-size DNN, where trained lightweight DNN satisfy the α and β constraints. α and β are the maximum available memory on edge devices and MAP time, respectively.

Major contributions and novelty of the work: To the best of our knowledge, this is the first work to address the problem of designing lightweight DNN by considering α and β constraints of the edge devices. Along with this, the major contributions and novelty of this work are as follows:

This research has been supported by I-DAPT HUB Foundation IIT (BHU) under Project Grant R&D/SA/I-DAPT IIT(BHU)/CSE/22-23/06-446), supervised by Dr. Hari Prabhat Gupta, Dept. of CSE, IIT (BHU) Varanasi.

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, India (e-mail: rahulmishra.rs.cse17@iitbhu.ac.in; hariprabhat.cse@iitbhu.ac.in;)
(Corresponding author: Rahul Mishra.)

Transforming large-size to lightweight DNN: The first contribution is to obtain a lightweight DNN from a large-size DNN. To do this, we dropout the unimportant units followed by reducing resource consumption from the large-size DNN using weight factorization and the minimal gated units on different layers of dropout DNN. The novel contributions lie in consideration of the number of connections in the given large-size DNN and maximum iteration runs for dropout. None of the existing work considers both in the dropout. These novel considerations speed up the procedure of estimating the updated dropout rate. In addition, considering α and β during dropout and reducing the resources (through weight factorization and minimal gated unit) makes our work different.

Train the lightweight DNN: We present a knowledge distillation based technique to train lightweight DNN (student) where, we incorporate two large-size DNN (teachers) with the same structural configuration, *i.e.,* un-trained teacher and pre-trained teacher. We introduce a novel *early halting technique*, where the student and the un-trained teacher are simultaneously trained up to certain (*i.e.,* halting) epochs under the guidance of the pre-trained teacher. Afterwards, the student training is propagated under the guidance of the pre-trained teacher. Such a novel mechanism of early halting saves the resources; therefore, speedups the training. The proposed training procedure transfers the knowledge from trained large-size DNN to lightweight DNN by minimizing the loss. The proposed training procedure transfers the knowledge from trained large-size DNN to lightweight DNN by minimizing the loss and improves its performance. We also propose an iterative algorithm to determine the optimal and trained lightweight model. Apart from neural architecture search [10], the proposed algorithm required a limited steps due to α and β constraints.

Experimental evaluations: We verify the effectiveness of the EarlyLight on the existing large-size DNN [11]–[16], datasets, and edge devices. The results show that the proposed work can significantly improve performance and minimize latency. We also demonstrate real-world evaluation for locomotion mode recognition and evaluate the performance.

Paper Organization. In the next section, we briefly discuss the existing literature. Section III presents the preliminary and overview of the solution. We propose an EarlyLight approach to train and design a lightweight DNN in Section IV. The further two sections present the empirically and real-world evaluations. Finally, the paper concludes in Section VII.

II. Background and motivation

Dropout in DNN: The prior studies used random [17], fixed [18], [19], or optimal [20], [21] dropout methods for reducing resources of DNN. Authors in [17] highlighted the concept of random dropout to handle the overfitting problem in DNN. Such random dropout deteriorated the DNN structure. To mitigate the random dropout problem, Han *et al.* in [18] proposed a mechanism of pruning and splicing side-by-side. The connection pruned during training can be spliced in back-propagation. They established a quadratic relation between the number of connections and neurons on the layers of DNN. To obtain lightweight DNN, the authors in [19] disassembled a large DNN into small ones. They further estimated the gradients of smaller models. These gradients are compared to obtain the most reliable model. The fixed dropout [18], [19] hampered the opportunities to improve the accuracy of the compressed DNN. Thus, the authors in [20] proposed a DNN compression technique that has incorporated the estimation of optimal dropout rather than a fixed value.

Reducing resource requirements of DNN: The existing work reduced the resource requirements of DNN by reducing the complexity of computing units [22]–[28], weights and biases [26], [27], and filter pruning [29]. The authors in [22] utilized the concept of layer factorization to reduce floating-point operations of fully connected layer and convolutional filter of DNN. The authors in [23] performed DNN compression using weight quantization and layers pruning. The authors not considered the quantization scheme for convolutional and fully connected layers. Next, to reduce the massive resource demand and high complexity of neural architecture search. The authors in [25] proposed the concept of the once-for-all (OFA) network. The authors decoupled training and architecture search stages with minimal accuracy compromise. The authors claimed to get a sub-network from OFA with no additional training cost. Similarly, the authors in [26] proposed the concept of iteratively shrinking and expanding DNN, utilizing sparsifying regularizer and multiplicative factor, respectively.

Training of lightweight DNN using KD: KD improves the performance of lightweight DNN using large-size DNN [7]–[9], [30]–[34]. Authors in [7] proposed a KD technique, where the generalization ability of a pre-trained teacher is transferred to the student to improve its recognition performance. The logits of teacher and student are compared to estimate the distillation loss that should be minimized during the training of the student. The authors in [30] introduced the concept of simultaneous training of scratch teacher and student. It provided a soft target of logits for estimating the distillation loss between teacher and student. Next, Zhou *et al.* [9] presented a mechanism to share some initial layers of student and scratch teacher to improve the recognition accuracy. Further, the authors in [31] examined the effectiveness of KD by exploiting the impact of various factors on KD's performance, shedding light on its strengths and limitations. Similarly, the authors in [32] determined the limitations of KD and identifies certain classes of data that are challenging to distill effectively. However, it focuses primarily on the identification of undistillable classes without providing concrete solutions or alternative approaches to overcome this limitation. Finally, authors in [8] introduced the concept of pre-trained teacher and scratch teacher where, both teachers simultaneously guide student model.

Motivation: This work is motivated by the following limitations, as noted in the existing literature. The prior work on the dropout technique in DNN [17]–[19] used a fixed or random value of dropout. It leads to the pruning of important connections having lower weights, which results in significant accuracy compromise. Moreover, the work [17], [18], [21], [35] do not guarantee the pruning of computing units in the recurrent neural network that consumes colossal resources.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3297026

Next, the work in [22], [23], [28], [29], [36], [37] reduced the size of DNN. However, the authors did not consider the constraints for edge device while compressing the DNN.

Further, to obtain a compressed (or lightweight) DNN using neural architecture search [38] is cost-ineffective, energy-consuming, and requires substantial resources for its execution. Finally, the existing literature on knowledge distillation [7]–[9], [30]–[32], [34] adopted mechanisms to improve the performance of the lightweight DNN. However, none-of-the existing work emphasized reducing resources during the training of lightweight DNN and maintaining significant accuracy.

III. Preliminary and Overview of Solution

This section describes the terminologies and notations used in this work. We also discuss an overview of the solution to design a lightweight DNN from a large-size for a given edge device. Table I illustrates the list of notations used in this work.

A. Preliminary

Let $\mathcal{D}$ denotes a dataset having n instances and k class labels, containing sensory measurements of p different sensors. An instance i of dataset $\mathcal{D}$ is denoted as $\mathbf{x}_i$, $\forall i \in \{1, \cdots, n\}$. Each instance $\mathbf{x}_i$ holds values of all p sensors and corresponds to one class label l of k available classes, where, $l \in \{1, \cdots, k\}$. Let the large-size and lightweight DNN are denoted by M^t and M^s, respectively.

Definition 1 (Knowledge distillation). *Knowledge distillation refers to a process for improving the performance of a lightweight DNN (M^s). Here, the knowledge (or generalization ability) of a large-size DNN (M^t) is utilized for training M^s, so the model M^s can mimic a similar output pattern as M^t. This training from M^t to M^s is sometimes referred as **student-teacher training** [7] in knowledge distillation.*

The training of student M^s using knowledge distillation from teacher M^t incorporates the comparison of their logits. The logits are the output features vector obtained at one layer before the softmax layer (output layer). Let $\mathbf{t}_i$ denote the logit vector of M^t for i^{th} training instance of dataset $\mathcal{D}$, where, $1 \leq i \leq n$. Let t_{ij} $(1 \leq j \leq k)$ is an element of $\mathbf{t}_i$, which can be estimated as $t_{ij} = w_{ij}x_{ij} + b_j$, where, $x_{ij} \in X$, $w_{ij} \in W^T$, and $b_j \in \mathbf{b}$ represent an element of feature matrix, weight matrix, and bias vector of teacher model, respectively. Similarly, we can estimate student logit vector $\mathbf{s}_i$ for i^{th} training instance of $\mathcal{D}$.

Definition 2 (Maximum Allowable Processing time). *A task in time-critical applications must be processed within a predefined time interval. Such time interval is known as Maximum Allowable Processing (MAP) time, denoted by β. Let an edge device processes x FLOPs per unit time. A task of y FLOPs can successfully process on an edge device if $y/x \leq \beta$. MAP, β, comprises processing capacity (in FLOPs) and memory access latency (estimated using relative memory model).*

B. Problem statement and overview of solution

Consider an edge device that can provide a maximum α space to store and process a task of β MAP time. In this work, we investigate the following problem: *how to design a lightweight DNN using a given large-size DNN such that the trained lightweight DNN can successfully process a task on an edge device with given α and β constraints?*

We propose the EarlyLight approach that first designs a lightweight DNN for edge devices. The approach trains the lightweight DNN using KD. Section IV-A1 and Section IV-A2 present procedures to design a lightweight DNN from large-size using dropout and reducing the parameters of the computationally complex units, respectively. While designing lightweight DNN, we consider the given constraints α and β of the edge device. We next present a procedure to train the designed lightweight DNN incorporating knowledge of pre-trained and un-training large-size DNN. We further introduce a novel early halting technique to accelerate the training of lightweight DNN, discussed in Section IV-B.

TABLE I: List of notations used in this work.

Symbol	Description	Symbol	Description
$\mathcal{D}$	Dataset	n	Instances in $\mathcal{D}$
Q_i	Neurons at layer i	M^t	Teacher model
M^s	Student model	$\mathbf{\Pi}^s$	Student classifier
$\mathcal{L}_{DL}$	Distillation loss	$\mathcal{L}_{CE}$	Cross entropy loss
$\mathcal{L}_{AL}$	Attention loss	$\mathbf{x}_{te}$	Testing instance
y_{te}	Testing label	d	Dropout

IV. EarlyLight: Lightweight neural networks on edge devices using Early halting

This section proposes an approach to design and train lightweight DNN on edge devices using early halting in knowledge distillation, acronymed as *EarlyLight*. The approach comprises mainly two phases: 1) designing of lightweight DNN for edge device and 2) training of the designed DNN. The designing phase involves the transformation of a given large-size DNN into a lightweight, considering the α and β constraints of the edge device. We assume that a dataset $\mathcal{D}$ and a large-size DNN M^t are given prior to this transformation. Later, the training phase introduces the technique of early hating in KD. The halting simultaneously reduces training time and improves the accuracy of the designed lightweight DNN. Fig. 1 illustrates the overview of the EarlyLight approach.

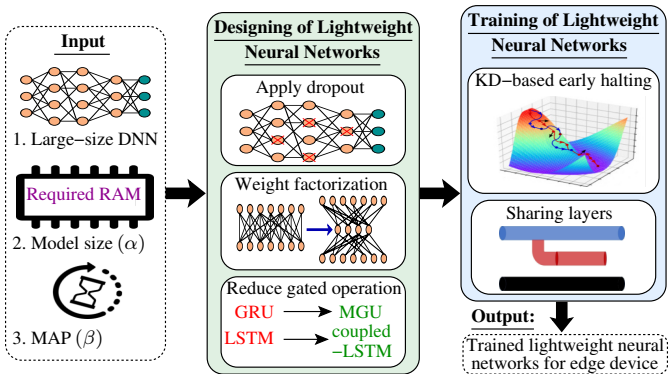


Fig. 1: An overview of EarlyLight approach. MAP: Maximum Allowable Processing, GRU: Gated Recurrent Unit, MGU: Minimal Gated Unit, LSTM: Long Shot Term Memory, KD: Knowledge Distillation.

A. Designing of lightweight DNN

This section describes the technique of designing lightweight DNN for edge devices, satisfying α and β constraints. We initially assume a large-size DNN (M^t) that is transformed into a lightweight DNN. First, we define the expression for execution time and memory consumption of lightweight DNN. Using the defined expressions, we deduce an optimization problem to minimize memory consumption and execution time for given constraints α and β, respectively. We next introduce the technique of estimating optimal dropout to reduce the resource requirement of M^t, which results in the dropout DNN. Later, the resources of the dropout DNN is minimized via weight factorization (convolutional and fully connected layers) and reduction in gated operations (recurrent layers). The resultant DNN is a lightweight neural network that satisfies the edge device's constraints α and β.

Let b_e and e_m denote the memory and time requirements for executing single FLOP, respectively. Such b_e and e_m depend on the hardware capacity of the edge devices. We deduce the expression for temporary memory consumption ($T_{mem.}$) and execution time ($T_{exec.}$) to run the lightweight DNN on a given edge device. The expressions are given as:

$$T_{mem.} = b_e \sum_{i=1}^{L} F_i, \quad T_{exec.} = e_m \sum_{i=1}^{L} F_i,$$

where, F_i denotes number of FLOPs for layer i of regular large-size or reduced lightweight model, given in Table II.

Finally, the objective function of a lightweight DDN M^s for a given edge device with the average available space α and MAP time β is given as:

$$\begin{aligned} &\min \ \Omega T_{mem.} + (1-\Omega) T_{exec.} \\ &\textit{s.t.,}\ \mathbf{c_1}: \ \ \mathbb{T}_{mem.} \leq \alpha, \mathbf{c_2}: \ \ \mathbb{T}_{exec.} \leq \beta, \end{aligned} \tag{1}$$

where, Ω ($0 \leq \Omega \leq 1$) is used to neutralize the mismatch between units of execution time and memory consumption. Solving Eq. 1 is tedious as the available resources on the edge devices changes dynamically. Therefore, we use a heuristic-based solution to apply dropout on large-size DNN and further reduce the resources requirement of dropout DNN.

1) Applying dropout on the large-size DNN: We first apply the dropout over given large-size DNN (M^t) to curtail unimportant or inferior connections. The resultant dropout DNN is equivalent to a lightweight DNN with weights scaled with a given dropout rate. The dropout over M^t reduces the required memory and execution time. Moreover, high and low dropout rates cause under-fitting and over-fitting of the DNN, respectively. A dropout over given large size model M^t is most not likely to incur overfit, if M^t itself does not overfit the data. The prior studies [17]–[21], the dropout is mainly used to prevent overfitting by modifying the network itself. The primary purpose of this work is to reduce the size of large-size networks using dropout. A low dropout rate requires considerable resources with minimal or no accuracy compromise. However, high dropout rate leads to substantial accuracy compromise. This work estimates the optimal dropout that best suits our resources and accuracy requirements. To initialize the selection of optimal dropout, we set a dropout rate (denoted by d) preferably with a higher value like $d = 0.5$ for hidden units and $d = 0.8$ for input units [17]. Let Q_b and Q_a denote the number of connections, before and after dropout, respectively. Let $\max_{iteration}$, and c are the maximum iteration runs for the dropout and a hyper-parameter, respectively. The updated dropout rate is given as follows: $d' \leftarrow d \times \max\left\{\sqrt{\frac{Q_b}{Q_a}}, \left(1 - \frac{iteration}{c \times \max_{iteration}}\right)\right\}$.

Furthermore, the initial value of Q_b is the nothing but the number of connections in M^t. The M^t can be represented as $\{W_i, Z_i : 1 \leq i \leq L\}$, where, Z_i is a binary matrix that indicates the state of the network connection at layer i. It holds the information about a weight that retains or discarded on a given dropout. The binary matrix Z_i is determined using discriminative function $f(\cdot)$ as, $Z_i^{(j,k)} = f(W_i^{(j,k)}), \forall (j,k) \in \mathcal{I}$, where, $\mathcal{I}$ denotes the set of indices of W_i at layer i. The function $f(\cdot)$ generates output 1 if connection $Q_i^{j,k}$ remains after training and 0 otherwise. Let M^t denotes a deep learning model, which can be represented as $\{W_i, Z_i : 1 \leq i \leq L\}$, where, Z_i is a binary matrix that indicates the state of the network connection at layer i and L is the number of layers in M^t. It holds the information about a weight that retains or discarded on a given dropout. The binary matrix Z_i is determined using discriminative function $f(\cdot)$ as, $Z_i^{(j,k)} = f(W_i^{(j,k)}), \forall (j,k) \in \mathcal{I}$, where, $\mathcal{I}$ denotes the set of indices of W_i at layer i. The function $f(\cdot)$ generates output 1 if connection $Q_i^{j,k}$ remains after training and 0 otherwise. Specifically, to apply dropout on layer i of a large-size DNN M^t, we randomly generate a binary mask Z_i of same shape with i, inspired from [39]. Afterwards, we scale the value of binary mask Z_i and replace i with following:

$$Dropout(i, Z_i) = i \otimes \Big(\frac{Size(Z_i)}{Sum(Z_i)}.Z_i\Big). \tag{2}$$

The above equation (Eq. 2) is iteratively called to meet the memory and execution time requirement of the resource constraint device without much degradation of performance. The steps involved in the selection of optimal dropout is illustrated in Procedure 1. The weight W in *SGD_function*() is updated using gradient descent with learning rate η. $\mathcal{L}(\cdot)$ is a cross-entropy loss associated with the DNN.

Procedure 1: Applying dropout on large-size DNN.

Input: M^t $\{W_i, Z_i : 1 \leq i \leq L\}$ with connection Q_a, learning rate η, loss of M^t as $\mathcal{L}$, $l \in L$;
Initialize $d^{(0)} \leftarrow 0.5$, $i \leftarrow 0$, $Q_a \leftarrow$ connections of M^t;
do
  Dropout $d^{(i)}$ on l of M^t with Q_a connections;
  Estimate loss: $Loss_i \leftarrow$*SGD_function* (W_i, Z_i);
  $Q_b \leftarrow$ reduced connections after dropout, $i \leftarrow i + 1$;
  Updating dropout using following formula:
  $d^{(i)} \leftarrow d^{(i-1)} \times \max\{\sqrt{\frac{Q_b}{Q_a}}, (1 - \frac{iteration}{c \times \max_{iteration}})\}$;
  $Q_a \leftarrow Q_b$ /*Updating connections*/;
while *(*$Loss_i \leq \mathcal{L}$*)*;
return Dropout model with connections Q_b;

2) Reducing resources of dropout DNN: Next, we describe the technique to reduce the resource requirements of the dropout DNN in terms of Floating Point Operations (FLOPs) and parameters. This reduction enforces the designing of lightweight DNN that satisfies the α and β constraints of the edge device. Apart from the prior work to reduce the resources of either convolutional or recurrent layers. This work introduces the technique to shrink the resource requirements of DNN layers, including convolutional, fully connected, and recurrent (Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU)). We apply weight factorization to reduce the resource requirements of the convolutional and fully connected layers. Further, we eliminate the gates of the recurrent units to suppress the resources of LSTM and GRU. Procedure 2 summarizes the steps involved in reducing the dropout DNN, satisfying α and β constraints of the edge devices.

Let I_i and O_i denote input and output dimensions of layer i for dropout DNN, where i may be Convolutional (Conv), Fully Connected (FC), LSTM, or GRU. The filter size, input channels, output channels of a convolutional layer i is represented as $f_i \cdot g_i$, h_i, and w_i, respectively. Further, s, L_g, and G_g denote step count, LSTM gates and GRU gates, respectively. The parameters (P_i) and required FLOPs (F_i) at layer i of DNN are given in Table II(a).

TABLE II: Number of parameters and FLOPs at layer i.

(a) **Regular layer**

Layer	Parameter (P_i)	FLOPs (F_i)
Conv	$(I_i \cdot (f_i \cdot g_i) \cdot O_i) + O_i$	$(f_i \cdot g_i) \cdot (I_i \cdot O_i) \cdot (h_i \cdot w_i)$
FC	$(I_i \cdot O_i) + O_i$	$(2I_i - 1) \cdot O_i$
LSTM	$L_g O_i \cdot (I_i + O_i + 1)$	$(2L_g O_i (I_i + O_i) + 4O_i)s$
GRU	$G_g O_i \cdot (I_i + O_i + 1)$	$(2G_g O_i \cdot (I_i + O_i) + 5O_i)s$

(b) **After factorization and using minimal gated unit**

Conv	$(I_i \cdot (f_i \cdot g_i) \cdot R_i) + R_i$	$((f_i \cdot g_i) \cdot (h_i \cdot w_i) + 1 + O_i) R_i$
FC	$(I_i \cdot R_i) + R_i$	$((2I_i - 1) + O_i) \cdot R_i$
LSTM	$L_g{}' O_i \cdot (I_i + O_i + 1)$	$(2L_g{}' O_i^l \cdot (I_i + O_i) + 4O_i)s$
GRU	$G_g{}' O_i \cdot (I_i + O_i + 1)$	$(2G_g{}' O_i \cdot (I_i + O_i) + 5O_i)s$

(a) *Weight factorization:* This paper uses the weight factorization technique [22] to reduce the parameters and FLOPs involved in convolutional and fully connected layers. The weight factorization technique introduces an intermediate multiplexing layer between two layers of the dropout DNN. This factorization of layers (convolutional or fully connected) reduces the computation requirement: if the size of the intermediate layer (denoted by R_i) $\not\geq \frac{I_i \times O_i}{I_i + O_i}$ [22]. R_i is obtained using a heuristic approach, where we start our factorization with $R_i < \frac{I_i \times O_i}{I_i + O_i}$ and estimate weight reconstruction error. Next, we iteratively decrease R_i and estimate reconstruction error at each iteration. Finally, we obtain R_i with minimum reconstruction error upon successful execution of this heuristic approach. The number of parameters and FLOPs at layer i after the weight factorization are given in Table II(b).

(b) *Reducing gated operations:* The parameters and FLOPs involved in the LSTM and GRU directly depend upon the gated operations. Therefore, we use the concept of MGU inspired from [40] to reduce the resource requirements of DNN. MGU relies on the basic principle that the gated units play a significant role in achieving higher performance, whereas incorporating several gated operations increases computation complexity. Hence, a wiser selection of gates that persist in the network leads to comparable accuracy and low execution complexity. Let $L_g{}'$ and $G_g{}'$ denotes the reduced gates in LSTM and GRU, respectively. We replace LSTM with coupled LSTM and GRU with MGU to reduce the gated operations.

Procedure 2: Reducing resource of dropout DNN.

Input: Dropout model with Q_b connections;
Initialize: $i \leftarrow 1$, $j \leftarrow 0$, dropout applied on l layers;
do
  for $i, j \in \{1 \leq i, j \leq l\}$ *and* $j \leftarrow i + 1$*;*
  do
    if i = *Conv.* ***or*** i = *FC* **then**
      Estimate reduced dimension R_i;
      Perform factorization ;
      Parameters and FLOPs using Table II(b);
    if i=*LSTM* **then**
      Replace LSTM with coupled LSTM;
      Estimate parameters and FLOPs (Table II(b)) with updated gate $L_g{}'$;
    else
      Replace the GRU cells with MGU;
      Estimate parameters and FLOPs (Table II(b)) with updated gate $G_g{}'$;
    Solve optimization problem in Eq. 1;
    $i \leftarrow i + 1$
while *(Eq. 1 is not satisfied)*;
return M^s with reduced parameters and FLOPs;

B. Training of lightweight DNN

This section covers the details about the training of lightweight DNN (or student) obtained in Section IV-A. The student (M^s) is trained using the knowledge distillation technique, where we involve two teachers, *i.e.,* a pre-trained large-size DNN (M^{tr}) and an un-trained large-size DNN (M^{te}). Further, we introduce the early halting technique for reducing the resource requirements for training M^s. Finally, this section derives the expression for optimal loss functions involved in the training. The main components of the training are: 1) knowledge distillation using early halting technique and 2) sharing layers of student and teacher.

1) Training M^s using knowledge distillation with early halting: Knowledge distillation from M^{tr} to M^s, while training of M^s on raw data improves its generalization ability. This improvement helps in enhancing the performance of M^s. In KD, the fine-tuned logits of M^{tr} is compared against the logits of M^s, which are generated from raw data. Thus, the logits of M^{tr} become a hard target for M^s. It also hinders the sufficient improvement in the performance of M^s. Thus, it could be beneficial to train an un-trained teacher (M^{te}) alongside M^s, where logits of M^{te} is a soft target for M^s . It provides soft-target during logits comparison. M^{tr} and M^{te} have same structural configuration. However, the un-trained teacher may sometimes undergo wrong random initialization, which leads to performance deterioration of M^s.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3297026

Moreover, if we use both M^{tr} and M^{te} during training of M^s then the problems, *i.e.,* hard logits target and performance diminution due to random initialization is solved [8]. It also leads to significant improvement in the performance of M^s. Despite the successful training of M^s due to appropriate matching of student and teachers logits, the simultaneous consideration of M^s, M^{tr} and M^{te} during training of M^s demands colossal resources. As it requires three models to be trained, *i.e.,* M^{tr} followed by M^{te} and M^s. The shared architecture reduces the training epochs and also ensures the student training follows the optimal loss propagation direction by avoiding poor initialization of the lightweight student.

We introduce the technique for *early halting* of M^{te} training after halting epoch h, where $h < E$ and E denotes the total required epochs for the training, to reduce the resource consumption during training of M^s. This work randomly assigns a higher value of E, which follows the following hypothesis, where E starts with a value three times the number of columns in the dataset. If we find the model is still improving after all epochs are complete, we try again with a higher value. Else if we find the model stopped improving before the final epoch, try again with a lower value. Further, the early halting saves the device's resources during training of M^s and therefore fasten the training. Hereafter, the training of M^s continues only under the guidance of trained M^{tr}, Fig. 2(a). The early halting technique uses cross-entropy loss $\mathcal{L}_{CE}(\cdot)$, attention loss $\mathcal{L}_{AL}(\cdot)$, and distillation loss $\mathcal{L}_{DL}(\cdot)$, as shown in Fig. 3. These loss functions are discussed in *supplementry file* in detail [41]. The performance of M^s can be improved in the supervision of trained M^{tr} that compares output at each epoch. The combined loss ($\mathcal{L}_{cb}(\cdot)$)is defined as:

$$\mathcal{L}_{cb}(\cdot) = \begin{cases} \lambda_1\mathcal{L}^s_{CE}(\cdot) + \lambda_2\mathcal{L}_{AL}(\cdot) + \lambda_3\mathcal{L}_{DL}(\cdot) + \lambda_4\mathcal{L}^{te}_{CE}(\cdot), \\ \text{till training of un-trained } M_i, \\ \lambda_1\mathcal{L}^s_{CE}(\cdot) + \lambda_2\mathcal{L}_{AL}(\cdot) + \lambda_3\mathcal{L}_{DL}(\cdot). \end{cases} \tag{3}$$

where λ_1, λ_2, λ_3, and λ_4 are the fractional contribution of different loss functions, $0 < \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\} \leq 1$. We only optimize the combined loss of M^s, as the contribution of the loss of untrained M^{te} is uniform through the training of M^s.

$$\min \ \mathcal{L}^s_{cb}(\cdot) \tag{4a}$$

$$s.t., \quad \lambda_1 + \lambda_2 + \lambda_3 = 1, \ 0 < \{\lambda_1, \lambda_2, \lambda_3\} < 1. \tag{4b}$$

To achieve the halting epoch $h < E$, we check the variance of combined loss $L^s_{comb}(\cdot)$ after each epoch. If the variance is nearly equal or shows a marginal change then the training of M^{te} un-trained teacher is halted.

Lemma 1. *The optimization problem in Eq. 4 holds a near optimal solution.*

Proof. The first second order derivatives of $\mathcal{L}^s_{cb}(\cdot)$ (Eq. 4).

$$\frac{\mathrm{d}\mathcal{L}^s_{cb}(\cdot)}{\mathrm{d}x_{ij}} = \lambda_1\frac{\mathrm{d}\mathcal{L}^s_{CE}(\cdot)}{\mathrm{d}x_{ij}} + \lambda_2\frac{\mathrm{d}\mathcal{L}_{AL}(\cdot)}{\mathrm{d}x_{ij}} + \lambda_3\frac{\mathrm{d}\mathcal{L}_{DL}(\cdot)}{\mathrm{d}x_{ij}}. \tag{5}$$

$$\frac{\mathrm{d}^2\mathcal{L}^s_{CE}(\cdot)}{\mathrm{d}^2x_{ij}} = \frac{\mathbb{1}(\cdot)w_{ij}^2 e^{(w_{ij}x_{ij}+b_j)}}{n\sum_{j=1}^{k} e^{(w_{ij}x_{ij}+b_j)}}\Big(1 - \frac{e^{(w_{ij}x_{ij}+b_j)}}{\sum_{j=1}^{k} e^{(w_{ij}x_{ij}+b_j)}}\Big).$$

$$\frac{\mathrm{d}^2\mathcal{L}_{AL}(\cdot)}{\mathrm{d}^2x_{ij}} = \frac{2}{n}\Big(\frac{w_{ij}}{||\mathcal{T}^s(\mathcal{F}^s_{ij})||}\Big)^2, \frac{\mathrm{d}^2\mathcal{L}_{DL}(\cdot)}{\mathrm{d}^2x_{ij}} = \frac{2}{n}(w_{ij})^2.$$

As $\frac{\mathrm{d}^2\mathcal{L}_{CE}(\cdot)}{\mathrm{d}^2x_{ij}} > 0$, $\frac{\mathrm{d}^2\mathcal{L}_{AL}(\cdot)}{\mathrm{d}^2x_{ij}} > 0$, and $\frac{\mathrm{d}^2\mathcal{L}_{DL}(\cdot)}{\mathrm{d}^2x_{ij}} > 0$, therefore, $\mathcal{L}_{CE}$, $\mathcal{L}_{AL}$, and $\mathcal{L}_{DL}$ are independently convex. However, their results are calculated from a feed-forward network that involves layer-by-layer non-linear activation, which lacks convexity. The proof for linear convergence is discussed in supplementary file [41].

□

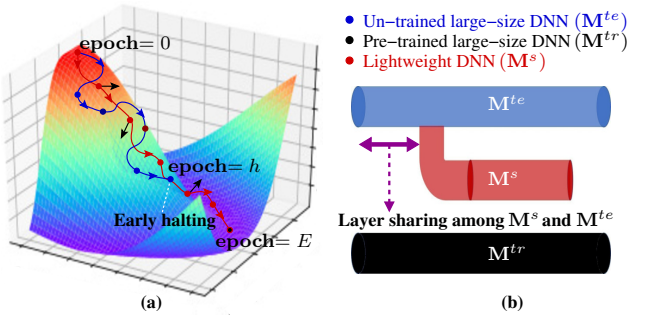


Fig. 2: Training of lightweight DNN (M^s) : (a) early halting of M^{te} training and (b) layer sharing among M^s and M^{te}.

2) Sharing layers of teacher and student: Inspired by the concept of layer sharing among student and teacher, as discussed in [9]; in this work, we share the first i layers of M^s and M^{te}. In other words, first, i layers of M^s and M^{te} are the same, as shown in Fig. 3. Layer sharing can be better visualized using Fig. 2(b), where, M^s is derived from M^{te}. The layer sharing improves the performance of the trained M^s and provides less variation in its output predicted probabilities. The layer sharing also preserve resources during training of M^s. In this work, we use sensory data for DNN training; thus, the complexity of large-size DNN is low. This lower complexity helps in obtaining lightweight DNN with minimal compression of large-size DNN. Through experimental analysis, we obtained that even at 50% layer sharing, the resource constraints of edge devices are satisfied. Thus, we are using 50% common layers of M^s and M^{te}.

C. EarlyLight algorithm

Algorithm 1 illustrates different steps involved in the EarlyLight approach to design and train lightweight DNN satisfying α and β constraints of edge devices. The complexity of the proposed EarlyLight algorithm depends upon the complexity of Procedure 1, Procedure 2, epoch E, and halting epoch h. To determine the complexity of Procedure 1, let us consider m_1 as the maximum iteration run of the procedure and then its time complexity of order $O(m_1)$. Similarly, let m_2 denotes the maximum steps required for solving Eq. 1 in Procedure 2; thus, its time complexity is of order $O(m_2)$. Further, let $O(X^a)$ and

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3297026

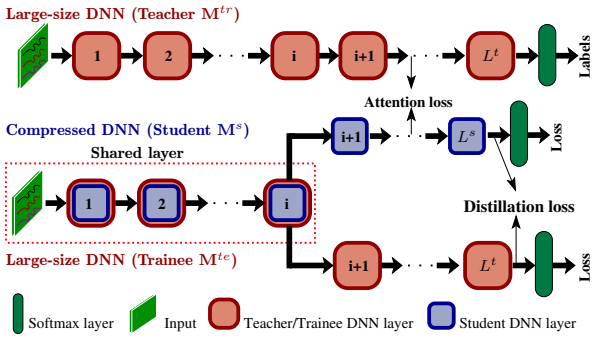


Fig. 3: Training of M^{te} and M^s under guidance of M^{tr}.

Algorithm 1: EarlyLight algorithm.

Input: Dataset D, M^{tr}, M^{te}, available space α, MAP time β, halting epoch h, epoch E;

Select a DNN with Q_a connections and L layers;
Identify L' layers that are not to be shared;
$\delta_L \leftarrow 0$, $l \leftarrow L'/2$; /*50% of non-shared layers*/
do
  $l \leftarrow l + \delta_L$;
  Call **Procedure 1**;
  Apply dropout on l layers of M^{te};
  Obtain dropout model with Q_b connections;
  Call **Procedure 2**;
  Solve optimization problem in Eq. 1;
  Obtain compressed model M^s from dropout;
  for *epoch* $e \leq E$ **do**
    if $e \leq h$ **then**
      Train M^s using M^{te} and M^{tr};
    else
      Train M^s using M^{tr};
  Solve optimization problem in Eq. 4;
  Obtain optimal value of λ_1, λ_2, and λ_3;
  $\mathcal{P} \leftarrow append(\mathcal{L}_{cb}(\cdot))$, preserve M^s, $\delta_L \leftarrow L'/10$;
while $(l \leq L')$;
$a \leftarrow \arg\min\{\mathcal{P}\}$;
Obtain lightweight model M^s for $\mathcal{L}_{cb}(\cdot)$ at $\mathcal{P}[a]$;
return Optimal lightweight model M^s;

$O(Y^b)$ denote the complexity of training M^{te} and M^s for one epoch. Hence, the complexity of the EarlyLight algorithm can be given as $O((m_1 + m_2 + X^a + Y^b) * h) + O((m_1 + m_2 + X^a) * (E - h)) = O((m_1 + m_2 + X^a) * E + (Y^b) * h)$.

V. Empirical Evaluation

This section evaluates the proposed work on publicly available datasets, existing large-size DNN, and edge devices.

A. Evaluation setup

1) Large-size DNN M^t architectures: We considered six existing DNN, including DeepZero [11], DeepFusion [12], DeepSense [13], DT-MIL [15], MFAP [14], and Human Activity Recognition using Multiple sensors fusion (HARM) [16], as shown in Fig. 4. These large-size DNN use sensory data for recognizing locomotion modes and human activities with high accuracy but require colossal parameters and FLOPs during their execution, as given in Table III.

TABLE III: FLOPs and parameters (prms) of large-size DNN, $(A, B) = A \times 10^B$.

DNN	Number of DNN layers				FLOPs	Prms
	Conv	FC	LSTM	GRU		
DeepZero [11]	15	5	2	—	(1.2, 10)	(3.8, 7)
DeepFusion [12]	18	3	—	1	(8.5, 11)	(5.4, 8)
DeepSense [13]	12	—	—	2	(7.5, 11)	(1.0, 7)
DT-MIL [15]	—	20	—	1	(2.4, 7)	(2.9, 4)
MFAP [14]	—	2	2	—	(3.7, 7)	(1.2, 7)
HARM [16]	6	2	—	—	(7.6, 10)	(1.2, 9)

2) Datasets: We select four publicly available sensory datasets, which are typically used in IoT applications, *e.g.*, locomotion mode recognition (LMR) [42], driving behaviour (DB) [43], river pollution monitoring (RPM) [44].

3) Edge devices for running lightweight M^s: We consider five different edge devices for deploying the lightweight DNN, *i.e.* trained student models (M^s), to verify the performance of the proposed approach. The devices include Intel edition kit ($\mathbf{d_1}$), Raspberry Pi 2 ($\mathbf{d_2}$), Raspberry Pi 3 ($\mathbf{d_3}$), Huwaie smartphone ($\mathbf{d_4}$), and Samsung smartphone ($\mathbf{d_5}$). The processing speed of the devices $\mathbf{d_1}$ to $\mathbf{d_5}$ are 11×10^8, 3×10^9, 5×10^{10}, 18×10^{10}, and 29×10^{10} FLOPs/second, respectively.

4) Baseline schemes for ablation studies: Table IV summarizes the architecture of different lightweight M^s and the training process for the ablation studies. $\mathbf{S}_5$ is the same as the proposed technique $\mathbf{S}_6$ with no early halting while training.

TABLE IV: Baseline schemes for ablation studies, where, M^s (student) is lightweight DNN, and M^{tr} (teacher) and M^{te} (trainee) are large-size DNN.

Scheme	Description
$\mathbf{S}_1$ [7]	M^s and M^{tr} are independent
	Training of M^s guided by pre-trained M^{tr}
$\mathbf{S}_2$ [30]	M^s and M^{te} are independent
	M^s and M^{te} are trained simultaneously
$\mathbf{S}_3$ [9]	M^s is sub-model derive from M^{te}
	M^s and M^{te} are trained simultaneously
$\mathbf{S}_4$ [8]	M^s is independent of M^{tr} and M^{te}
	M^s and M^{te} are trained under guidance of M^{tr}
$\mathbf{S}_5$	M^s is sub-model derive from M^{tr} or M^{te}
	M^s and M^{te} are trained under guidance of M^{tr}
$\mathbf{S}_6$ (**proposed**)	M^s is sub-model derive from M^{tr} or M^{te}
	M^s and M^{te} are trained under guidance by M^{tr} up to some epochs then M^{te} **training is halted**

5) Implementation details: For implementing the lightweight DNN transformed from large-size DNN, as illustrated in Fig. 4, we incorporated the sequential model and functional API of deep learning library Keras in Python language. Next, Algorithm 1 and all the procedures are implemented in Python. We adopt the differential evolution technique for estimating the fractional contributions of the different loss functions, *i.e.,* λ_1, λ_2, and λ_3. In the experimental analysis, we randomly divide the datasets into two sub-datasets, *i.e.,* training and testing with 70% and 30% data instances, respectively, using the function $sklearn.model\_selection.train\_test\_split()$ in Sklearn model selection. We repeat each experiment 100 times and calculate the average value. Further, α (memory size) is the

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3297026

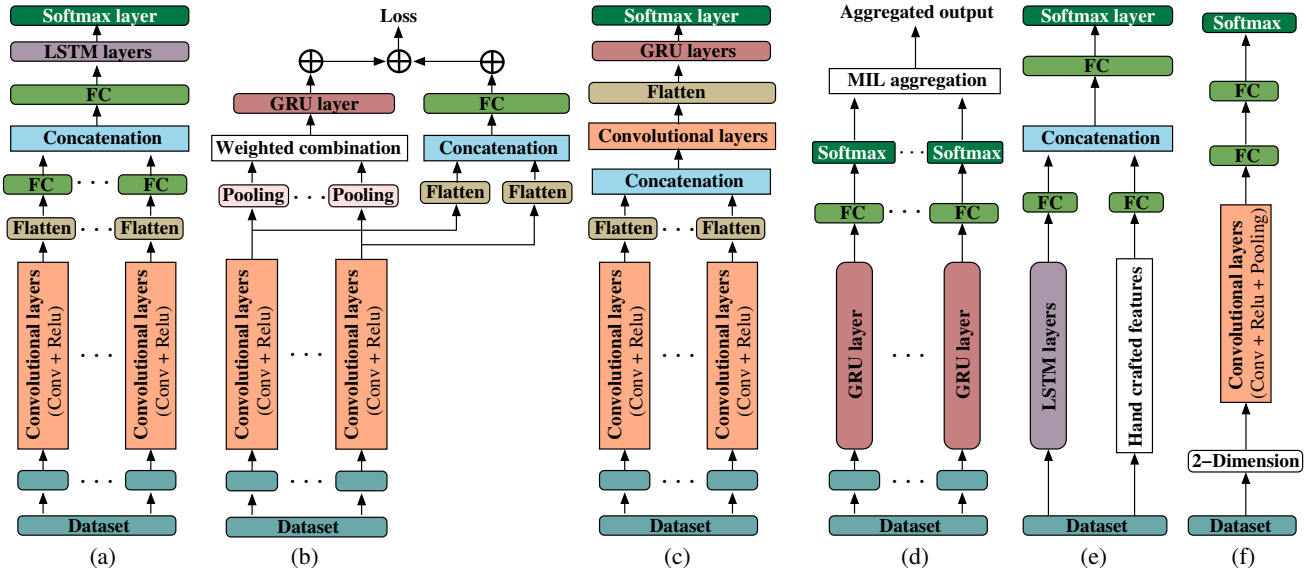


Fig. 4: Illustration of different DNN architectures incorporating sensory data for recognizing locomotion modes and human activities. (a) DeepZero [11], (b) DeepFusion [12], (c) DeepSense [13], (d) DT-MIL [15], (e) MFAD [14], and (f) HARM [16].

maximum available memory on the edge while requesting the appropriate compressed DNN. β (maximum allowable processing time) is estimated by analyzing the data processing history of the devices. In other words, β is determined in accordance with the processing speed (FLOPs/seconds) achieved by the device in the past event of data processing. $\beta \geq$ prior device processing speed $\times$ current FLOPs. The lightweight DNN can be trained on a considered device or server. To estimate the MAP time we use FLOPs and relative memory modfel (DDR4 to TPU/accelerator FLOPs), detailed in supplementary file [41]. We opt cross-validation technique to achieve the best performance by tuning the dropout ratio. We have also tuned the dropout separately for each layer and also during training to achieve proper loss on a higher dropout value. Furthermore, we have used TensorFlow Lite for Microcontrollers (TFLM) library, which provides optimized DNN operations for microcontrollers.

B. Validation metrics

This work used standard classification metrics to evaluate and compare the performance: F_1 score and accuracy. Let a given dataset consists of a set of $\mathcal{A}$ classes, and $|\mathcal{A}|$ represents the number of classes. Let TP_i, TN_i FP_i, and FN_i are the true positive, true negative, false positive, and false negative counts of a class $i \in \mathcal{A}$, respectively. The *accuracy* metric is computed as: $\frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|}\frac{TP_i+TN_i}{TP_i+TN_i+FP_i+FN_i}$. Next, the F_1 *score* is computed as: $\frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|}\frac{2\times TP_i}{2\times TP_i+FP_i+FN_i}$.

C. Experimental results

1) Impact of different schemes on accuracy of DNN: First, we performed the experiment to estimate the performance of lightweight DNN train using different schemes $\mathbf{S}_1$-$\mathbf{S}_6$. We considered different large-size DNN and $\mathbf{d_1}$ to $\mathbf{d_5}$ edge devices. The value of constraints $\alpha = 0.65$ and $\beta = 197$ ms. Table V illustrates the configuration of lightweight DNN based on the available resources on edge. The result depict that the accuracy of the lightweight DNN transform using scheme $\mathbf{S}_6$ is almost equal to the scheme $\mathbf{S}_5$. However, $\mathbf{S}_5$ requires a large number of FLOPs and parameters during the training of lightweight DNN. Due to the early halting of the trainee model training, the scheme $\mathbf{S}_6$ achieves a significant reduction in training time of lightweight DNN M^s. It is interesting to observe that the transformed lightweight DNN from large-size DNN using scheme $\mathbf{S}_6$ achieves high accuracy within the constraints of edge device, take less time during training of lightweight model and therefore saves the energy and resources of training machine.

2) Impact of the loss functions on accuracy of DNN: Table VI illustrates the fractional contribution of different loss functions on the performance of the lightweight DNN M^s using different edge devices. We used the devices and available resources as shown in previous results. We considered DeepZero as a large-size DNN (teacher model). We can observe from the result that with the increase in the device's resources, the accuracy and F_1 score of the lightweight DNN M^s improved, and the contribution of distillation loss (λ_3) increases. It is because when the difference between M^s and M^{te} is significant then simultaneous training deviates the M^s from achieving optimal convergence point due to random initialization of M^{te}.

3) Impact of the training time on accuracy: In this experiment, we determine the training time and accuracy achieved under different schemes for M^s. We consider DeepZero as large-size DNN M^{tr}, whose lightweight variant is deployed on device $\mathbf{d_3}$. Table VII illustrates the training time and accuracy of different schemes for device $\mathbf{d_3}$ excluding the training time of the pre-trained teacher model. As shown in the previous result, $\mathbf{S}_5$ and $\mathbf{S}_6$ give the high accuracy as compared with others. $\mathbf{S}_5$ trains both M^s and M^{te} simultaneously and therefore needs more FLOPs. The proposed $\mathbf{S}_6$ early halts the training of M^{te} and needs fewer resources. Therefore, an

TABLE V: Illustration of accuracy (%) achieved by schemes ($\mathbf{S}_1$-$\mathbf{S}_6$) on device specific student from teacher (DeepZero [11], DeepFusion [12], DeepSense [13], DT-MIL [15], MFAD [14], and HARM [16]). fConv and Conv are factorized and unfactorized Convolutional layers, fFC = factorized FC layer, cLSTM = coupled LSTM, MGU= Minimal Gated Unit.

	Device	Student model specification			Accuracy (%) in scheme					
		Number of DNN layers	FLOPs	Parameters	$\mathbf{S}_1$ [7]	$\mathbf{S}_2$ [30]	$\mathbf{S}_3$ [9]	$\mathbf{S}_4$ [8]	$\mathbf{S}_5$	$\mathbf{S}_6$ (**Proposed**)
DeepZero [11]	$\mathbf{d_1}$	fConv = 5, fFC = 3, cLSTM = 2	2.8×10^9	1.1×10^7	75.43	79.27	82.23	87.27	89.41	**90.53**
	$\mathbf{d_2}$	fConv = 7, fFC = 5, cLSTM = 2	4.3×10^9	1.5×10^7	79.42	83.19	85.19	89.13	**91.21**	91.08
	$\mathbf{d_3}$	fConv = 12, Conv = 3, fFC = 5, LSTM = 1, cLSTM= 1	6.5×10^9	1.8×10^7	82.31	85.11	87.93	90.97	**92.54**	92.32
	$\mathbf{d_4}$	fConv = 1, Conv = 14, fFC = 3, FC = 2, LSTM = 1, cLSTM = 1	8.4×10^9	2.3×10^7	85.41	87.29	88.71	92.23	**93.51**	93.33
	$\mathbf{d_5}$	fConv = 1, Conv = 14, fFC = 1, FC = 4, LSTM = 1, cLSTM = 1	9.3×10^9	2.6×10^7	86.21	87.45	88.92	92.71	93.44	**93.57**
DeepFusion [12]	$\mathbf{d_1}$	fConv = 6, fFC = 2, MGU = 1	3.1×10^{11}	2.3×10^8	85.93	88.81	89.78	90.07	91.13	**91.19**
	$\mathbf{d_2}$	fConv = 6, fFC = 3, GRU = 1	4.3×10^{11}	2.9×10^8	86.17	89.43	90.03	90.56	**92.83**	92.16
	$\mathbf{d_3}$	fConv = 7, fFC = 3, GRU = 1	5.7×10^{11}	3.4×10^8	88.29	90.97	91.07	92.43	**94.07**	93.23
	$\mathbf{d_4}$	fConv = 12, Conv = 3, fFC = 3, MGU = 1	6.6×10^{11}	4.1×10^8	89.31	91.34	92.23	92.71	**94.93**	94.57
	$\mathbf{d_5}$	fConv = 10, Conv = 5, fFC = 3, MGU = 1	7.3×10^{11}	4.8×10^8	89.73	91.47	92.31	92.97	**95.03**	94.63
DeepSense [13]	$\mathbf{d_1}$	fConv = 8, GRU = 1	3.3×10^{11}	3.6×10^6	80.22	80.83	84.21	84.89	89.12	**89.47**
	$\mathbf{d_2}$	fConv = 6, Conv = 4, GRU = 1, MGU = 1	4.1×10^{11}	4.2×10^6	81.29	81.92	87.20	87.62	**92.53**	92.21
	$\mathbf{d_3}$	fConv = 6, Conv = 4,GRU = 2	4.9×10^{11}	5.7×10^6	82.06	82.81	88.17	88.97	92.91	**93.03**
	$\mathbf{d_4}$	fConv = 4, Conv = 8, GRU = 1, MGU = 1	5.5×10^{11}	7.3×10^6	82.72	83.09	88.51	89.23	93.09	**93.20**
	$\mathbf{d_5}$	fConv = 4, Conv = 8,GRU = 2	6.3×10^{11}	8.1×10^6	82.91	83.93	89.02	89.91	93.37	**93.44**
DT-MIL [15]	$\mathbf{d_1}$	fFC = 12, MGU = 1	1.0×10^7	1.2×10^4	73.93	78.61	79.53	84.21	**87.23**	87.07
	$\mathbf{d_2}$	fFC = 12, FC = 2, MGU = 1	1.2×10^7	1.6×10^4	74.27	78.83	80.27	84.61	**88.73**	88.26
	$\mathbf{d_3}$	fFC = 12, FC = 6, MGU = 1	1.5×10^7	1.9×10^4	75.29	81.44	81.93	85.07	**89.45**	89.03
	$\mathbf{d_4}$	fFC = 12, FC = 8, MGU = 1	1.8×10^7	2.2×10^4	75.89	81.87	82.29	85.27	90.83	**90.85**
	$\mathbf{d_5}$	fFC = 8, FC = 12, GRU = 1	2.1×10^7	2.6×10^4	76.39	82.91	83.17	85.59	**91.21**	91.07
MFAD [14]	$\mathbf{d_1}$	fFC = 1, FC = 1, LSTM = 1	2.1×10^7	7.6×10^6	79.17	82.08	83.07	84.21	**85.37**	85.16
	$\mathbf{d_2}$	fFC = 1, FC = 1, LSTM = 1	2.1×10^7	7.6×10^6	79.29	82.31	83.26	84.53	**85.81**	85.23
	$\mathbf{d_3}$	fFC = 1, FC = 1, LSTM = 1, cLSTM = 1	2.4×10^7	9.1×10^6	80.17	83.23	84.11	85.38	**87.47**	87.13
	$\mathbf{d_4}$	fFC = 1, FC = 1, LSTM = 1, cLSTM = 1	2.4×10^7	9.1×10^6	80.41	83.63	84.61	85.83	**87.65**	87.29
	$\mathbf{d_5}$	fFC = 1, FC = 1, LSTM = 2	3.2×10^7	1.0×10^7	81.01	84.33	85.07	86.51	**88.27**	88.02
HARM [16]	$\mathbf{d_1}$	fConv = 1, fFC = 2	4.7×10^{10}	6.2×10^8	76.71	80.49	82.96	87.81	90.09	**90.13**
	$\mathbf{d_2}$	fConv = 1, fFC = 2	4.7×10^{10}	6.2×10^8	77.39	81.37	84.76	89.03	**90.39**	90.23
	$\mathbf{d_3}$	fConv = 2, fFC = 2	5.4×10^{10}	7.1×10^8	80.01	83.23	85.59	89.47	**91.92**	90.93
	$\mathbf{d_4}$	fConv = 4 fFC = 2	6.1×10^{10}	8.5×10^8	80.22	83.63	85.81	89.93	**92.17**	91.71
	$\mathbf{d_5}$	fConv = 4, Conv = 1 fFC = 2	6.8×10^{10}	1.0×10^9	80.71	83.61	86.21	90.21	92.19	**92.34**

TABLE VI: Fractional contributions (λ_1, λ_2, and λ_3) of different loss functions on the performance of M^s with $\mathbf{d_1}$ to $\mathbf{d_5}$ devices.

Device	Fractional weights			Accuracy	F_1 score
	λ_1	λ_2	λ_3		
$\mathbf{d_1}$	0.5117	0.3972	0.0911	90.53%	91.21%
$\mathbf{d_2}$	0.4919	0.4523	0.0563	91.08%	92.74%
$\mathbf{d_3}$	0.3208	0.3563	0.3227	92.32%	93.98%
$\mathbf{d_4}$	0.3700	0.2965	0.3334	93.33%	94.90%
$\mathbf{d_5}$	0.3922	0.2717	0.3361	93.57%	95.13%

TABLE VII: Training time and accuracy on different schemes for device $\mathbf{d_3}$ on large-size DNN (DeepZero). TT = Training Time and Acc = accuracy.

Schemes	$\mathbf{S}_1$	$\mathbf{S}_2$	$\mathbf{S}_3$	$\mathbf{S}_4$	$\mathbf{S}_5$	$\mathbf{S}_6$
Part (a): Accuracy v/s required TT (in minutes)						
Acc (in %)	82.31	85.11	87.93	90.97	92.54	92.32
TT ± 5	95	231	206	251	217	185
Part (b): Accuracy on a given TT = 197 minutes						
FLOPs$\times 10^{13}$	1.09	2.37	2.37	2.37	2.37	2.15
Acc (in %)	82.03	69.95	82.03	70.88	80.47	92.32

edge device with limited resources takes more time to train $\mathbf{S}_5$ as compared to the proposed $\mathbf{S}_6$. Table VII illustrates $\mathbf{S}_6$ has the best accuracy within the given training time. This is because other schemes either lack sufficient training or design lightweight DNN randomly, resulting in lower accuracy.

4) Compression ratio of large-size DNN: In this section, we illustrate the impact of the compression ratio (size of M^s/size of M^t (shared layer)) on the accuracy of the lightweight DNN. Here, we consider 50% shared layers between teacher and student models and assume only shared layers to determine the model size. The compression ratio depends on the available resources of the edge device. An edge device requires a high compression ratio with low processing speed (FLOPs) on a fixed MAP time β. Table VIII illustrates the various compression ratios of DeepZero. As expected, the high compression ratio gives low accuracy and F_1 score. An interesting observation from this result is that the accuracy and F_1 score go down sharply after a fixed compression ratio. The lightweight DNN for the given compression has very few layers and gated units. Thus, it shows incompetence in successfully classifying the given classes. Additionally, F_1 score is higher than the achieved accuracy due to the uneven distribution of class labels.

TABLE VIII: Different compression ratio of large-size DNN (DeepZero).

Compression ratio	×60	×50	×43	×18	×13	×4.8
Accuracy	67.43	70.17	73.22	87.21	90.23	92.16
F_1 score	69.09	71.83	74.88	88.87	91.89	93.82
FLOPs ($\times 10^9$)	2.03	2.40	2.79	6.67	9.23	25.21

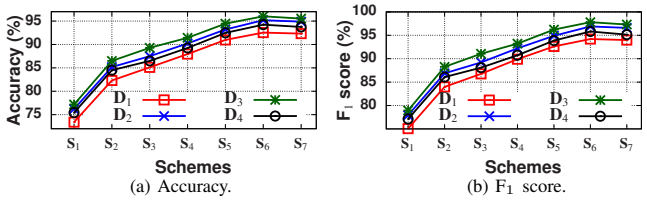
5) Impact of halting of M^{te} on performance of M^s: Further, we depict the impact of the halting time of M^{te} on the performance of lightweight M^s. We used the SHL dataset and the DeepZero model. Table IX illustrates that after a fixed duration of training of M^{te}, the progress in the accuracy and F_1 score of M^s is almost constant. However, the required resources for continuous training of M^{te} is increased with time. We, therefore, conclude that training of M^{te} and M^s till the end of processing does not provide high accuracy. They only consume more resources. Fig. IX illustrates the saturation in accuracy and F_1 score after a certain epochs (60 for DeepZero). After this saturation point, we can quickly halt the training of M^{te} without compromising accuracy of M^s.

TABLE IX: Impact of halting training of trainee model on accuracy and F_1 score achieved by M^s of DeepZero [11] for device $\mathbf{d_2}$.

Epochs	40	50	**60**	70	80	90
Training time M^{te} (in min.)	161	174	**185**	202	213	225
Accuracy M^s (in %)	78.51	83.02	**91.08**	91.37	91.53	91.97
F_1 score M^s (in %)	80.17	84.68	**92.74**	93.03	93.19	93.63
FLOPs ($\times 10^{13}$)	2.53	2.77	**2.93**	3.09	3.29	3.56

6) Impact of datasets on the accuracy of DNN: Finally, we study the performance achieved by different schemes ($\mathbf{S}_1$-$\mathbf{S}_6$) on selected datasets (SHL, VDB, DBD, and RWM). Fig. 5(a) illustrates the average accuracy (in %) achieved on schemes $\mathbf{S}_1$-$\mathbf{S}_6$. The result illustrates that for the DBD dataset, the accuracy under each scheme is highest. It is due to the least number of classes in the DBD dataset. Similarly, SHL achieves the lowest accuracy due to the presence of a maximum 8 classes. Next, $\mathbf{S}_5$ and $\mathbf{S}_6$ achieve the highest accuracy on all the datasets (*i.e.,* SHL, VDB, DBD, and RWM). Scheme $\mathbf{S}_5$ slightly supersedes the accuracy of scheme $\mathbf{S}_6$, as it incorporates training of trainee for all epochs. However, $\mathbf{S}_5$ consumes higher resources than $\mathbf{S}_6$. Further, a similar variation in F_1 score is observed under schemes $\mathbf{S}_1$-$\mathbf{S}_6$ on different datasets, as illustrated in Fig. 5(b). F_1 score is higher than the accuracy for all datasets. The class labels are not uniformly distributed among all class labels in the considered datasets. An interesting observation from the result is that if the number of class labels in the dataset is small then the achieved accuracy will be higher. Additionally, if the distribution of class labels is non-uniform then F_1 score will be higher in contrast with accuracy.

7) Comparison with existing work: Table X illustrates that the proposed EarlyLight approach and its modules (dropout and reducing resources) individually outperform the existing state-of-the-art techniques in terms of achieved performance. During the experiment, we use the SHL dataset, DeepZero model, device $\mathbf{d_4}$, and same lightweight DNN for evaluating all KD-based approaches. Such consideration also makes the FLOPs and parameters same for the student model in KD-based technique, as shown in Table X. The optimal dropout and resource requirement improve the accuracy of the standalone modules. We also observed that the standalone module for reducing resources requires least FLOPs due to simultaneous consideration of convolutional, fully connected, and recurrent layers. Moreover, the existing dropout technique may reduce FLOPs and parameter more than the optimal dropout but compromises accuracy. Finally, we observed that on computing optimal dropout with reducing resources decreases both FLOPs and parameter. Additionally, KD improves the obtained lightweight DNN.



(a) Accuracy. (b) F_1 score.

Fig. 5: Impact of datasets (SHL, VDB, DBD, and RWM) on accuracy and F_1 score of different schemes.

TABLE X: Illustration of FLOPs , parameters, accuracy (%) of student model on different approaches using SHL [42] dataset.

Module	**Existing work**	**FLOPs**	**Parameter**	**Accuracy**
Dropout	[20]	9.6×10^9	3.6×10^7	73.23%
	[21]	$\mathbf{9.1 \times 10^9}$	$\mathbf{3.0 \times 10^7}$	70.62%
Proposed work **(Only optimal dropout)**		9.4×10^9	3.1×10^7	**77.23%**
Reducing Resources	[22]	8.8×10^9	2.9×10^7	76.47%
	[23]	8.9×10^9	3.0×10^7	78.59%
Proposed work **(Only resource reduction)**		8.7×10^9	2.8×10^7	**81.73%**
KD-based training (using same lightweight DNN)	[7]	8.4×10^9	2.3×10^7	85.41%
	[30]	8.4×10^9	2.3×10^7	87.29%
	[9]	8.4×10^9	2.3×10^7	88.71%
	[8]	8.4×10^9	2.3×10^7	92.23%
Proposed work **(Overall EarlyLight)**		8.4×10^9	2.3×10^7	**93.33%**

VI. Real-world evaluation

A. Hardware and software

The prototype hardware is based on the NodeMCU ESP32 as data collection and processing unit. It is powered by ESP8266 module that can wirelessly transmit data using WiFi. Next, we attach the inertial sensors to measure angular rate, force and magnetic field and transfer to the NodeMCU. Further, the data is processed on NodeMCU using deployed lightweight DNN to predict class labels (locomotion modes). These labels are transferred to the server using the GSM module. NodeMCU has 512 KB SRAM (with 4 MB flash storage); therefore a high order DNN compressed is needed. We considered two scenarios of locomotion mode recognition, *i.e.,* identifying locomotion modes using sensors deployed in the shoes of the kids and the wrist band of a person.

We used DeepZero [11] as the large-size DNN upon which transform is performed. Fig. 4(a) illustrates the architecture of

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3297026

the DeepZero. The lightweight DNN of DeepZero is exported and loaded into the flash memory of the NodeMCU. The large-size DeepZero is trained on the Dell PC with 32 GB RAM with a clock speed of 2.4 GHz. The pre-trained model is further transformed using the proposed scheme and deployed on NodeMCU. Besides, the compressed DNN (from DeepZero) is trained on the server, as the available memory on NodeMCU is limited to store training data on its primary storage.

B. Data collection

We collected the sensory data of different locomotion modes including bicycle ($\mathbf{a_1}$), bike ($\mathbf{a_2}$), car ($\mathbf{a_3}$), auto rickshaw ($\mathbf{a_4}$), bus ($\mathbf{a_5}$), and train ($\mathbf{a_6}$). To facilitate the data collection, we developed an android application that uses the Inertial Measurement Unit (IMU) sensor of the smartphone. The sampling rate of IMU is set to 100 Hz to record 6000 data points per minute. We used an android smartphone, Samsung Galaxy Alpha for collecting data against each locomotion modes. The data was collected by the 10 volunteers (5 males and 5 females). The android application consists of a menu through which volunteers can select a locomotion mode. The measurements of the IMU sensor is recorded for 60 seconds.

C. Evaluation methods

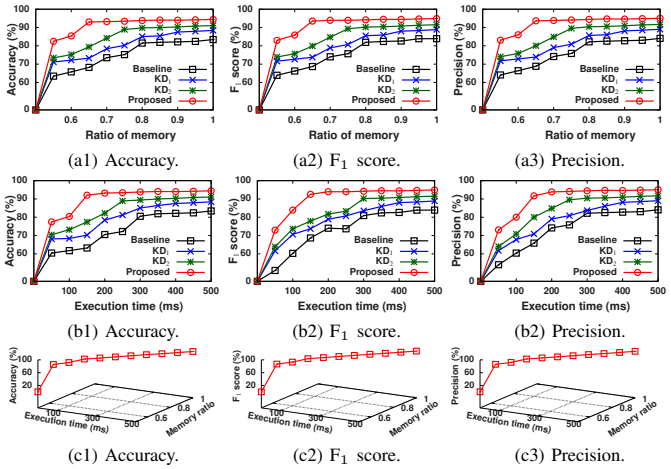
We considered the following four evaluation methods for verifying the effectiveness of the proposed approach in the real world scenario. First, we used **Baseline** method that is a lightweight version of DeepZero [11] and NodeMCU ESP32 as edge device. Next, we used $\mathbf{KD}_1$ an extension of the baseline method, where the lightweight DNN is trained under the guidance of the pre-trained DeepZero model using the knowledge distillation technique discussed in [7]. Next, $\mathbf{KD}_2$ method where some initial layers of lightweight DNN and standard DeepZero are shared to improve the performance of lightweight DNN. The lightweight and standard models are trained simultaneously using the knowledge distillation technique discussed in [9]. Finally, we used the proposed approach, named as **Proposed** in the results. Apart from the existing methods, we adopt an early halting technique for training the lightweight DNN under the guidance of pre-trained and untrained DeepZero. The initial layers of the compressed DNN are shared with the large-size DeepZero.

D. Validation metrics

• *F_1 score and accuracy:* The description of the validation metrics F_1 score and accuracy are discussed in Section V-B.
• *Precision:* Precision of a DNN is defined as the ratio of correct positive observation to the total correctly predicted observation, *i.e.*, $\mathbf{P}_3 = \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|}\frac{TP_i}{TP_i+FP_i}$.
• *Leave-one-out test:* This validation metric trains the DNN for all class labels except for one randomly chosen class label. However, during testing, the unseen class label is also supplied for predicting the output. Thus, it evaluates the performance of the classifier for unseen class labels.

E. Result 1: Impact of memory and execution time

We first study the impact of memory and execution time on the validation metrics. Fig. 6(a1) illustrates the accuracy achieved by the different methods with the change in memory ratio. The memory ratio is the ratio of required memory to the available memory of edge device. We can observe from the results that the proposed work outperforms the existing methods and achieves significantly higher accuracy with minimal energy consumption. The proposed work achieves accuracy around 94% when memory ratio is just 0.65. Similar observations can be made for other validation metrics, *i.e.,* F_1 score, and precision, as shown in parts (a2)-(a3) of Fig. 6, respectively. Next, parts (b1)-(b3) of Fig. 6 illustrate the impact of execution time on the validation metrics. The results depict that the execution time also follows a similar pattern as memory consumption, where the proposed work outperforms the existing methods and achieve maximal accuracy in minimum execution time. It requires 197 ms to achieve the performance of more than 93%. It is because of the involvement of multiple teachers (teacher and trainee) and layer sharing in the proposed scheme. Last, we study the impact of simultaneous change in memory ratio and execution time on the validation metrics as shown in parts (c1)-(c3) of Fig. 6. Similar as previous results, the results demonstrate that the proposed work can achieve accuracy of around 93% when the execution time (β) is just 197 ms, and memory ratio (α) is 0.65.



(a1) Accuracy. (a2) F_1 score. (a3) Precision.
(b1) Accuracy. (b2) F_1 score. (b2) Precision.
(c1) Accuracy. (c2) F_1 score. (c3) Precision.

Fig. 6: Impact of memory, execution time, and simultaneous change in memory ratio and execution time on validation metrics.

F. Result 2: Class-wise accuracy

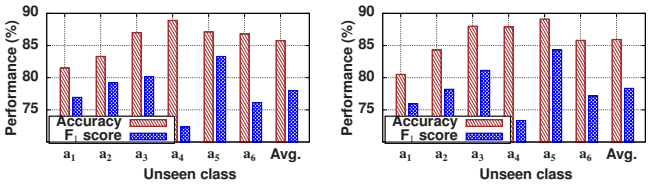
In this result, we fixed the value of edge constraints $\alpha = 0.65$ and $\beta = 197$ ms for estimating the class-wise accuracy of different methods that were considered for real world evaluation. Table XI illustrates the class-wise accuracy of different methods in the real world evaluation. We can observe from the result that the proposed method outperforms all existing methods in achieving accuracy against each class. Additionally, the class-wise accuracy of class $\mathbf{a_2}$ (bike) is highest, as it holds the most identifiable features in the dataset. Moreover, the number of instances for class a_2 is highest.

TABLE XI: Confusion matrix of [$\mathbf{KD}_1$,$\mathbf{KD}_2$,proposed] in %.

$\mathbf{a}_1$	$\mathbf{a}_2$	$\mathbf{a}_3$	$\mathbf{a}_4$	$\mathbf{a}_5$	$\mathbf{a}_6$
[70,73,89]					
	[76,84,96]				
		[71,80,94]			
			[72,78,92]		
				[75,79,91]	
					[75,82,93]

G. Result 3: Accuracy and F_1 with unseen class

Finally, we study the performance of the proposed scheme, when one class is unseen. In other words, we perform the leave-one-out test in this result. Fig. 7(a) illustrates the accuracy and F_1 score achieved by the proposed scheme, where the instance of the given class is missing from the training dataset, and the ratio of memory consumed (α) is 0.65 . Here, we observe that the accuracy and F_1 score decreases when one class is missing. It is because, the built classifier does not hold the features associated with the missing class. Further, the impact of one unseen class varies over another because the number of data instance that generates the most identifiable feature by a classifier changes with the change in the unseen class. Similarly, Fig. 7(b) illustrates the accuracy and F_1 score when instances of the given class are missing from the training dataset and execution time (β) is 197 ms.



(a) Ratio of memory ($\alpha = 0.65$). (b) Execution time ($\beta = 197$ ms).

Fig. 7: Accuracy and F_1 score with one unseen class.

VII. Conclusion

In this paper, we proposed an approach to design and train a lightweight DNN using a large-size DNN, where trained lightweight DNN satisfied the α and β constraints of the edge devices, acronymed as EarlyLight. The approach used optimal dropout selection and factorization for DNN compression. The EarlyLight approach also incorporated knowledge distillation to improve the performance of the lightweight DNN. Further, we introduced an early halting technique to train lightweight DNN, which saved resources; therefore, it speedups the training procedure. We also carried out several experiments to validate the effectiveness of the EarlyLight. The results showed that the approach achieved high accuracy on edge devices.

In this future work, we aim to expand the evaluations beyond a limited set of edge devices, recognizing the potential for extension. Furthermore, we acknowledge the need to consider the dynamic workload experienced by edge devices, as this is a common phenomenon that affects the determination of compressed model sizes. Including dynamic load-based compression as an additional dimension is an important aspect of our future research. Moreover, our work lays the groundwork for exploring a DNN compression technique capable of handling noise in datasets caused by faulty sensors, which presents an interesting direction for future development.

References

[1] R. Mishra, H. P. Gupta, and T. Dutta, "Teacher, trainee, and student based knowledge distillation technique for monitoring indoor activities," in *Proc. ACM SenSys*, 2020, pp. 729–730.

[2] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1745–1759, 2019.

[3] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things," *IEEE Comm. Mag.*, vol. 50, no. 12, pp. 144–149, 2012.

[4] X. He, X. Wang, Z. Zhou, J. Wu, Z. Yang, and L. Thiele, "On-device deep multi-task inference via multi-task zipping," *IEEE Trans. Mobile Comput.*, pp. 1–1, 2021, doi: 10.1109/TMC.2021.3124306.

[5] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Comput.*, vol. 16, no. 3, pp. 82–88, 2017.

[6] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv preprint arXiv:2010.03954*, 2020.

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

[8] H. Zhao, X. Sun, J. Dong, C. Chen, and Z. Dong, "Highlight every step: Knowledge distillation via collaborative teaching," *IEEE Trans. Cybern.*, pp. 1–12, 2020, doi: 10.1109/TCYB.2020.3007506.

[9] F. Y. C. R. B. W. Z. X. . G. K. Zhou, G., "Rocket launching: A universal and efficient framework for training well-performing light net," in *Proc. AAAI*, 2018, pp. 1–8.

[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[11] R. Mishra, A. Gupta, H. P. Gupta, and T. Dutta, "A sensors based deep learning model for unseen locomotion mode identification using multiple semantic matrices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 799–810, 2022.

[12] H. Xue, W. Jiang, C. Miao, Y. Yuan, F. Ma, X. Ma, Y. Wang, S. Yao, W. Xu, A. Zhang *et al.*, "Deepfusion: A deep learning framework for the fusion of heterogeneous sensory data," in *Proc. ACM Sensys*, 2019, pp. 151–160.

[13] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. WWW*, 2017, pp. 351–360.

[14] Z. Chen, C. Jiang, S. Xiang, J. Ding, M. Wu, and X. Li, "Smartphone sensor-based human activity recognition using feature fusion and maximum full a posteriori," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 7, pp. 3992–4001, 2020.

[15] V. M. Janakiraman, "Explaining aviation safety incidents using deep temporal multiple instance learning," in *Proc. ACM SIGKDD*, 2018, pp. 406–415.

[16] F. M. Noori, M. Riegler, M. Z. Uddin, and J. Torresen, "Human activity recognition from multiple sensors data using multi-fusion representations and cnns," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, no. 2, pp. 1–19, 2020.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. NIPS*, 2015, pp. 1135–1143.

[19] I. Jindal, M. Nokleby, and X. Chen, "Learning deep networks from noisy labels with dropout regularization," in *Proc. IEEE ICDM*, 2016, pp. 967–972.

[20] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. ACM SenSys*, 2017, pp. 1–14.

[21] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. ICML*, 2017, pp. 2498–2507.

[22] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proc. ACM SenSys*, 2016, pp. 176–189.

[23] J. Chauhan, J. Rajasegaran, S. Seneviratne, A. Misra, A. Seneviratne, and Y. Lee, "Performance characterization of deep learning models for breathing-based authentication on resource-constrained devices," *Proc. ACM Sensys*, vol. 2, no. 4, pp. 1–24, 2018.

[24] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *arXiv preprint arXiv:2006.05525*, 2020.

[25] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. ICLR*, 2020, pp. 1–15.

[26] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proc. CVPR*, 2018, pp. 1586–1595.

[27] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proc. CVPR*, 2019, pp. 11 398–11 407.

[28] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proc. ACM SenSys*, 2018, pp. 278–291.

[29] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proc. ICCV*, 2017, pp. 5058–5066.

[30] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *Proc. ICLR*, 2018, pp. 1–17.

[31] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proc. IEEE/CVF*, 2019, pp. 4794–4802.

[32] Y. Zhu, N. Liu, Z. Xu, X. Liu, W. Meng, L. Wang, Z. Ou, and J. Tang, "Teach less, learn more: On the undistillable classes in knowledge distillation," in *Proc. NIPS*, 2022.

[33] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[34] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proc. AAAI*, vol. 34, no. 04, 2020, pp. 5191–5198.

[35] R. V. Babu *et al.*, "Single-step adversarial training with dropout scheduling," *arXiv preprint arXiv:2004.08628*, 2020.

[36] S. Lee and S. Nirjon, "Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems," in *Proc. ACM Sensys*, 2019, pp. 138–152.

[37] X. Liu, W. Li, J. Huo, L. Yao, and Y. Gao, "Layerwise sparse coding for pruned deep neural networks with extreme compression ratio." in *Proc. AAAI*, 2020, pp. 4900–4907.

[38] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[39] H. Pham and Q. Le, "Autodropout: Learning dropout patterns to regularize deep networks," in *Proc. AAAI*, vol. 35, no. 11, 2021, pp. 9351–9359.

[40] G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou, "Minimal gated unit for recurrent neural networks," *Int. J. Autom. Comput.*, vol. 13, no. 3, pp. 226–234, 2016.

[41] Supplementry_EarlyLight, 2023. [Online]. Available: https://sites.google.com/view/rahulmishracse/earlylight

[42] SHL Challenge, 2022. [Online]. Available: http://www.shl-dataset.org/activity-recognition-challenge/

[43] Y. G. Yong Zhang, Junjie Li, "Vehicle driving behavior," 2022. [Online]. Available: http://dx.doi.org/10.21227/qzf7-sj04

[44] Water-to-Cloud, 2022. [Online]. Available: http://thoreau.uchicago.edu/