Poster Abstract: Efficient Knowledge Distillation to Train Lightweight Neural Network for Heterogeneous Edge Devices

Preti Kumari
NUS, Singapore
preti@nus.edu.sg

Hari Prabhat Gupta
IIT (BHU) Varanasi, India
hariprabhat.cse@iitbhu.ac.in

Biplab Sikdar
NUS, Singapore
bsikdar@nus.edu.sg

ABSTRACT

This poster presents a novel approach that harnesses large-sized deep neural networks to craft lightweight variants, addressing constraints in storage, processing speed, and task execution time on heterogeneous edge devices. Knowledge distillation is employed to refine the training of lightweight deep neural networks, and a novel early termination technique is introduced to optimize resource utilization and expedite the training process. This approach yields satisfactory accuracy while accommodating diverse heterogeneous edge device constraints.

KEYWORDS

Deep neural network, heterogeneity, knowledge distillation, sensors

ACM Reference Format:
Preti Kumari, Hari Prabhat Gupta, and Biplab Sikdar. 2023. Poster Abstract: Efficient Knowledge Distillation to Train Lightweight Neural Network for Heterogeneous Edge Devices . In *The 21st ACM Conference on Embedded Networked Sensor Systems (SenSys '23), November 12–17, 2023, Istanbul, Turkiye.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3625687.3628409

1 INTRODUCTION

Internet of Things (IoT) applications rely on sensory data for real-time monitoring and detection tasks [3]. Rapid data processing within Maximum Allowable Processing (MAP) time constraints is crucial in time-sensitive IoT applications. Deep Neural Networks (DNNs) are incredibly well-suited for such IoT applications since they have the specific benefit of great accuracy. Their implementation on edge devices, however, proves to be a challenging task because of a number of resource constraints, including restrictions on processing and storage capacity. Knowledge Distillation (KD) transfers knowledge from larger to smaller models, a valuable technique when resources are limited for larger models [2].

This poster considers a scenario with N number of heterogeneous edge devices where the memory space of the devices may not be equal. Let a device n consists of α_n memory space, and $\alpha = \min\{\alpha_1, \alpha_2, \cdots, \alpha_N\}$ represents the minimum space among all N devices. Given β as MAP time, we focus on this challenge: *How can we create a lightweight DNN from a larger one, ensuring successful task processing on each edge device under a given constraints α and β?* We propose an approach that leverages KD to design and train lightweight DNNs tailored for a diverse set of edge devices. Initially, this approach creates a lightweight DNN from the larger model using dropout techniques, taking into account the constraints of α and β. Subsequently, we outline a training procedure for designed lightweight DNN, leveraging knowledge from both pre-trained and untrained large DNNs. An early termination method is also given to fasten the training of the lightweight DNN.

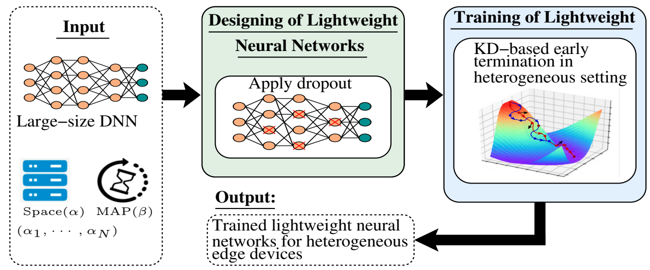


Figure 1: An overview of the proposed approach.

2 LIGHTWEIGHT DNN

This poster first converts a large DNN into a lightweight using dropout techniques considering the edge device's constraints. Assuming we have a dataset $\mathcal{D}$ and a large DNN denoted as M_l. Subsequently, we employ an early termination method in KD to expedite training and enhance precision in the resulting lightweight DNN.

2.1 Lightweight DNN using Dropout technique

Let m and t_n be the memory and processing time to complete one Floating Point Operation (FPO) of edge device n, respectively. Define $T_{mem} = m \sum_{i=1}^{L} F_i$ and $T^n_{exec} = t_n \sum_{i=1}^{L} F_i$ as the memory usage and processing time, respectively, where F_i represents the number of FPOs needed to execute for layer i in the lightweight DNN. Let $T_{exec} = \max\{T^1_{exec}, T^2_{exec}, \cdots, T^N_{exec}\}$, then the objective function of a lightweight DNN (M_s) with the given constraints as

$$\begin{aligned} &\min \quad \omega T_{mem} + (1-\omega) T_{exec}, \\ &\text{subject to} \quad c_1 : T_{mem} \le \alpha, c_2 : T_{exec} \le \beta, c_3 : Acc \ge Acc_{th}, \end{aligned} \quad (1)$$

where ω and Acc_{th} are weight factor and desired accuracy, respectively. To solve Eq. 1, we apply a heuristic-based dropout on M_l to prune unimportant connections, yielding a lightweight DNN with scaled weights. We estimate an optimal dropout rate (d) for our resources and accuracy needs. Define Q_b and Q_a as the connections count before and after dropout. Then the dropout rate is given as $d' = d \times \max\{\sqrt{\frac{Q_b}{Q_a}}, (1 - \frac{itr}{c \times itr_{max}})\}$, where itr_{max} and c are the required number of iterations for dropout and a hypermeter, respectively. Let $M_l = \{W_i, Z_i : 1 \le i \le L\}$, where Z_i is a binary matrix indicating network connection states at layer i and L is the total

SenSys '23, November 12–17, 2023, Istanbul, Turkiye

ACM ISBN 979-8-4007-0414-7/23/11…$15.00
https://doi.org/10.1145/3625687.3628409

number of layers in M_l. To apply dropout in a large DNN M_l to layer i, we create a binary mask Z_i matching its shape. We then scale the mask values and replace i with $Dropout(i, Z_i) = i \times \left(\frac{Size(Z_i)}{Sum(Z_i)}.Z_i\right)$. This process is iterated to meet memory and execution constraints on resource-limited devices, minimizing performance impact.

2.2 Training of lightweight DNN

The lightweight DNN (M_s) undergoes training via the KD technique, employing the guidance of a pre-trained (M_l^p) and an untrained (M_l^u) large DNN. Incorporating M_l^p and M_l^u boosts M_s performance, addressing challenges with hard logit targets and mitigating performance deterioration resulting from random initialization.

We propose an early termination technique for M_l^u at epoch h, with $h < E$, to save resources during M_s training, where E is the cumulative epochs needed for the training process. The choice of h can vary due to differences in memory space on heterogeneous edge devices, where $\alpha = \min\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$. After h epochs, the training of M_l^u terminates and M_s continues only under the guidance of trained M_l^p. On each epoch, we compare the performance of M_s using the combined loss ($\mathcal{L}_{cb}(\cdot)$) of M_s and M_l^u which includes cross-entropy loss $\mathcal{L}_{CE}(\cdot)$, attention loss $\mathcal{L}_{AL}(\cdot)$, and distillation loss $\mathcal{L}_{DL}(\cdot)$. The combined loss is defined as $\lambda_1\mathcal{L}_{CE}^s + \lambda_2\mathcal{L}_{AL} + \lambda_3\mathcal{L}_{DL} + \lambda_4\mathcal{L}_{CE}^{l^u}$. After h epoch, the comparison is conducted using the loss ($\mathcal{L}_{cb}^s$) of M_s which is $\lambda_1\mathcal{L}_{CE}^s + \lambda_2\mathcal{L}_{AL} + \lambda_3\mathcal{L}_{DL}$. Here, λ_i (for $1 \le i \le 4$) represents the fractional contribution of different loss functions, with $0 \le \lambda_i \le 1$. The optimization formulation of the loss is defined as

$$\begin{aligned} \min \quad & \mathcal{L} = x(\mathcal{L}_{cb}) + (1-x)(\mathcal{L}_{cb}^s), \\ \text{subject to} \quad & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1, 0 < \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\} < 1, \end{aligned} \tag{2}$$

where $x = 1$ till the training of untrained model M_l^u, else $x, \lambda_4 = 0$. Given varying edge device capabilities, the degree of model compression will differ. When the model undergoes substantial compression, it needs to be trained for a certain number of epochs to attain a reasonable level of accuracy. Let $size_s$ and $size_l$ be the sizes of lightweight and large DNNs, respectively. The minimum number of epochs to which the model must be trained is $e = E\left(1 - \frac{size_s}{size_l}\right)$. After e epochs, we assess the variance of $\mathcal{L}$ following each epoch. The training of M_l^u is stopped if the variance is essentially constant or shows only minor variations. The procedures to construct and train a lightweight DNN are shown in Algorithm 1.

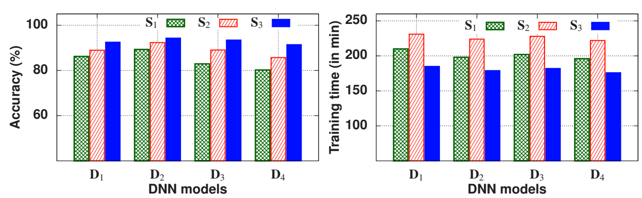
Algorithm 1: Design and training of lightweight DNN.

```
Input: D, M_l^p, M_l^u, α, β, h, E, λ_1, ···, λ_4;
Output: Optimal lightweight model M_s;
Estimate α = min{α_1, α_2, ···, α_N};
Train teacher model (M_t^p) on D;
while not converge do
    Apply dropout and obtain M_s with Q_a connections;
    Train M_s using M_l^u and M_l^p for e epochs;
    for epoch e + 1 ≤ E do
        if e + 1 ≤ h then
            Train M_s using M_l^u and M_l^p;
        else
            Train M_s using M_l^p;
    Solve Eq. 2 and Obtain optimal value of λ_1, ···, λ_4;
    P ← append(L), preserve M_s;
Obtain M_s for L at arg min{P};
return Optimal lightweight model M_s;
```

3 PERFORMANCE EVALUATION

This section assesses the proposed research using openly available datasets, pre-existing large-scale DNNs, and a variety of heterogeneous edge devices (Raspberry Pi, Samsung, and Huawei smartphones). We consider four established DNNs, namely DeepZero [3], DeepFusion [4], DeepSense [5], and DT-MIL [1], designated as D_1 through D_4. Additionally, we have analyzed three different approaches: S_1[2], S_2[6], and our proposed method S_3.

Figure 2(a) and Figure 2(b) demonstrate the DNNs' accuracy and training time, respectively. Employing schemes S_1 and S_2 demands a substantial number of FPOs and parameters. Scheme S_3 stands out, significantly reducing training time for the lightweight DNN M_s. The transformed lightweight DNN using S_3 maintains high accuracy within edge device constraints, accelerating training and conserving energy and resources.



(a) Accuracy. (b) Training time.

Figure 2: Impact of DNNs on different schemes.

4 CONCLUSION

This poster introduced an approach that designs and trains a lightweight DNN using a large-size counterpart, meeting heterogeneous edge device constraints. It employs optimal dropout along with KD to enhance performance. An early termination technique is introduced to accelerate training and conserve resources. Experimental validation demonstrates the proposed approach's high accuracy on edge devices.

REFERENCES

[1] V. M. Janakiraman. 2018. Explaining Aviation Safety Incidents using Deep Temporal Multiple Instance Learning. In *Proc. ACM SIGKDD*. 406–415.

[2] A. Mishra and D. Marr. 2018. Apprentice: Using Knowledge Distillation Techniques To Improve Low-Precision Network Accuracy. In *Proc. ICLR*. 1–17.

[3] R. Mishra, A. Gupta, H. P. Gupta, and T. Dutta. 2022. A Sensors Based Deep Learning Model for Unseen Locomotion Mode Identification using Multiple Semantic Matrices. *IEEE Trans. Mobile Comput.* 21, 3 (2022), 799–810.

[4] H. Xue, W. Jiang, C. Miao, Y. Yuan, F. Ma, X. Ma, Y. Wang, S. Yao, W. Xu, A. Zhang, et al. 2019. DeepFusion: A Deep Learning Framework for the Fusion of Heterogeneous Sensory Data. In *Proc. ACM Sensys*. 151–160.

[5] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. 2017. Deepsense: A Unified Deep Learning Framework for Time-series Mobile Sensing Data Processing. In *Proc. WWW*. 351–360.

[6] H. Zhao, X. Sun, J. Dong, C. Chen, and Z. Dong. 2020. Highlight Every Step: Knowledge Distillation via Collaborative Teaching. *IEEE Trans. Cybern.* (2020), 1–12. doi: 10.1109/TCYB.2020.3007506.