Tree DNN: A Deep Container Network

Brijraj Singh
Data Science
Sony Research India
brijraj.08@gmail.com

Swati Gupta
Department of Computer Science
JIIT, Noida, India
swatigupta.iitr@gmail.com

Mayukh Das
Microsoft Research, India
mayukhdas@microsoft.com

Praveen Doreswamy Naidu
On Device AI
Samsung Research Institute, Bangalore, India
praveen.dn@samsung.com

Sharan Kumar Allur
On Device AI
Samsung Research Institute, Bangalore, India
sharan.allur@samsung.com

Abstract

Multi-Task Learning (MTL) has shown its importance at user products for fast training, data efficiency, reduced overfitting etc. MTL achieves it by sharing the network parameters and training a network for multiple tasks simultaneously. However, MTL does not provide the solution, if each task needs training from a different dataset. In order to solve the stated problem, we have proposed an architecture named TreeDNN along with it's training methodology. TreeDNN helps in training the model with multiple datasets simultaneously, where each branch of the tree may need a different training dataset. We have shown in the results that TreeDNN provides competitive performance with the advantage of reduced ROM requirement for parameter storage and increased responsiveness of the system by loading only specific branch at inference time.

1 Introduction

Multitask learning (MTL) has proved its importance by facilitating the execution of multiple tasks with just one DNN (Deep Neural Network) model(1). MTL is achieved by exploiting the generalized features extracted from the back-bone module of the model where each task is differentiated at the last layers of the DNN network. In general, MTL works well if all the tasks are related to each other where generic features are exploited by the backbone layers and task-specific features are extracted with the help of classification layers(2).

In many commercial systems, it is often required to load multiple DNN models on working memory to perform sequential tasks. Application of camera models in mobile phones is one of the use-cases, where a user first goes with the basic camera and then switches among multiple options like night-mode, selfie, depth, sports, food, make-up etc. All categories of cameras use their respective DNN models for their corresponding tasks. A point to notice here is that, in general, all tasks are mutually exclusive and only one is performed at a time. As either a camera will work for the portrait features or it will work for night mode. As functional requirements are different hence, the training to their corresponding DNN model needs to be different as well. Therefore, an input sample should be plugged-in each DNN model exclusively unlike a typical MTL where same input is considered for multiple tasks.

In order to solve the stated problem, we are proposing here a deep container model named as TreeDNN. TreeDNN can be considered as inverted tree, which has a trunk and multiple branches (one branch for each task). A TreeDNN is deployed to support multiple tasks requirement where each task needs training from different dataset. To train a TreeDNN model, we need to perform a joint training

36th Conference on Neural Information Processing Systems (NeurIPS 2022), New Orleans, USA.

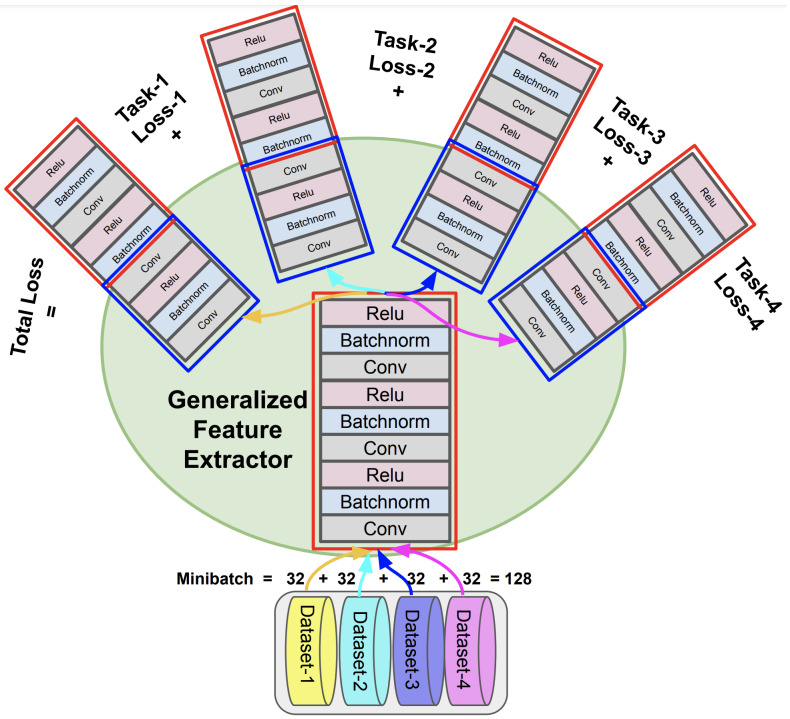

Figure 1: Training of TreeDNN

of all the branches followed by specialized training of each branch individually. A joint training is performed with the help of federated minibatch, which is prepared by considering all the datasets. The federated minibatch is inferenced through the trunk and all the branches of tree. Loss values are calculated at each branch and are aggregated for cumulative loss. In this work we have considered MobileNet-V2 (1), (3) as a backbone architecture.The cumulative loss is back-propagated through the branches and trunk of the tree to provide the generalized training to the model. Afterwards, the specialized training is given to each branch of the tree individually.

This work is proposed for the camera system of mobile phones, where multiple cameras are switched frequently. TreeDNN helps in reducing the model latency thereby improving the system responsiveness. It requires only 50% of the total number of parameters for the storage requirement along with competitive predicting performance.

1.1 Related work

MTL provides the facility to solve multiple tasks using one DNN model. Few MTL studies share parameters among their tasks and are known as shared trunk methodology(2) (4) (5). All the MTL methods work in a similar manner, where input sample is inferenced through multiple operations and operations are shared among tasks(6). This way it reduces efforts in training a task from scratch by utilizing the trunk (7). There are multiple applications like (8) proposed a MTL-based framework for identifying each plant's type and disease by considering raw image and transformed feature-maps from another pretrained DNN.

However, MTL methods do not provide solution, if each task needs training from a different dataset.

1.2 Proposed Solution

TreeDNN is a container network that is a collection of multiple DNN models prepared to serve the tasks independently. As the name suggests a portion of the DNN model is shared by all the tasks and is referred as trunk of the tree as shown in Figure 1. Each branch of the tree is dedicated to a particular task. In this work, TreeDNN utilizes the ops of vision model. Trunk of the tree is designed to receive generalized training through all the datasets. TreeDNN is designed to speed up the switching among

the tasks, which are different branches of the tree therefore, branches of the tree need to be lighter (on scale of parameters). The speedup is attained because of keeping trunk of the tree on the working memory and consuming the time only in loading the particular branch of the tree. Whole process can be segregated among a) Model designing b) Dataset preparation and c) Training.

(a) Model Designing: A typical vision model is a combination of operations like Convolution, BatchNorm, Relu, Pooling, Linear, Fully connected layers etc. A TreeDNN architecture contains a trunk, branches and both are designed with the above mentioned vision operations. In this work, we have considered MobileNet-V2 as a backbone architecture therefore trunk of the Tree is mainly a collection of inverted residual blocks. Branch of the TreeDNN is prepared from the second half portion of Mobilenet-V2 and the number of layers are reduced/ increased based on the complexity of the task performed at that particular branch. The process of model creation is defined through line 25-30, in Algorithm 1. It is interesting to realize that $M_i : Trunk \rightarrow Branch_i$ is a fully-functioning DNN model capable to serve $Task_i$.

Algorithm 1 TREE_DNN($D_1, D_2, \ldots, D_k$)

1: $\vec{\mathbb{M}} \leftarrow$ MODEL_CREATION($Trunk, Branch_1, Branch_2, \ldots, Branch_k$)
2: **procedure** FEDERATED_BATCH_PREPARATION($D_1, D_2, \ldots, D_k$)
3: **for** $i \in (0, len(D_1)/B)$ **do** ▷ $Batch\_size = B$
4: $Fed\_Batch_i = D_1[i * (B/k) : (i+1) * B/k] || \ldots || D_k[i * (B/k) : (i+1) * B/k]$
5: **end for**
6: **end procedure**
7: **procedure** GENERALIZED TRAINING($\vec{\mathbb{D}}, Max\_epoch$)
8: **for** $epoch \in Max\_epoch$ **do**
9: **for** $i \in k$ **do** ▷ Loss from each branch
10: **for** $Fed\_Batch \in \vec{\mathbb{D}}$ **do** ▷ $\vec{\mathbb{D}} = D_1 || D_2 \ldots || D_k$
11: $Loss[i]$=$\vec{\mathbb{M}}[i](Fed\_Batch)$
12: **end for**
13: **end for**
14: $Net\_Loss = \Sigma_{j=1}^{k}(W_i * Loss[j])$
15: $Net\_Loss.backward()$ ▷ Back-propagate with Net_Loss
16: **end for**
17: **end procedure**
18: **procedure** SPECIALIZED_TRAINING($\vec{\mathbb{M}}, \vec{\mathbb{D}}$)
19: $gradients(Trunk) \leftarrow False$
20: **for** $i \in k$ **do**
21: **for** $epoch \in Max\_epoch$ **do**
22: **for** $(data, label) \in \vec{\mathbb{D}}[i]$ **do**
23: $Branch_i \leftarrow loss(\vec{\mathbb{M}}[i](data), label).backward()$ ▷ Backpropagation
24: **end for**
25: **end for**
26: **end for**
27: **end procedure**
28: **procedure** MODEL_CREATION($Trunk, Branch_1, Branch_2, \ldots, Branch_k$)
29: $Trunk \leftarrow Arrangement(Conv, Relu, BatchNorm)$
30: **for** $task \in total\_tasks$ **do**
31: $Branch_{task} \leftarrow Arrangement(Conv, Relu, BatchNorm, FC, Linear)$
32: **end for**
33: **end procedure**

(b) Dataset Preparation: TreeDNN requires joint training through multiple datasets. In order to maintain trade-off between bias and variance while performing training at each branch, it is required to design the minibatch considering all the datasets. Since minibatch is prepared with the participation from each of the datasets, we refer to it as Fed_Batch (federated batch). The preparation of the federated batch is shown in line:3, Algorithm 1.

(c) Training TreeDNN is trained in two steps: i) Generalized Training ii) Specialized Training.

- **Generalized Training:** In first step, all the branches and trunk of the tree are trained together. During training, the samples are inferenced through the trunk and branches of the tree. This way, loss is calculated at each branch and then all the losses are aggregated as

Table 1: Case study of camera model, containing 8 DNN models

Parameters	Existing Solution	Proposed Solution
Memory	120 MB	68 MB
Response Time	228 ms	120 ms

Table 2: Result of TreeDNN in comparison with dedicated backbone model

Tree	Dataset	Samples	TreeDNN	SOTA (Mobilenet-V2)
Branch-1	CIFAR10	50000	89%	91%
Branch-2	CIFAR100	50000	66.9%	68%
Branch-3	CALTECH101	9146	83.5%	65%
Branch-4	CALTECH256	30,607	49.2%	40%
Branch-5	SVHN	600,000	92.6%	93.3%

shown in equation 1. Cumulative loss is then backpropagated to the network through branch and the trunk equation 2. This step helps in training the trunk of the tree. Where L is the loss value, $\hat{y}_j$ is the observed output at each branch and y_j is the label at branch j.

$$J(w^T, b) = \sum_{i=1}^{n} L_{Task:1}(\hat{y}_1^i, y_1^i) + \sum_{i=1}^{n} L_{Task:2}(\hat{y}_2^i, y_2^i) + \ldots + \sum_{i=1}^{n} L_{Task:k}(\hat{y}_k^i, y_k^i) \quad (1)$$

$$J(w^T, b).backward() \quad (2)$$

- **Specialized Training:** Now, it is required to tune each branch corresponding to a specific task. This step is performed by backpropagating the loss back to the network while keeping the gradient of trunk False equation 3. Specialized training is applied at each branch independently as shown in equation 4. Where I is the input samples.

$$\hat{v} = Trunk(I)\ , \qquad gradient(Trunk) = False \quad (3)$$

$$\hat{y}_j = branch_j(\hat{v})\ , \qquad Jj(w^T, b) = \sum_{i=1}^{n} L_{Task:j}(\hat{y}_j^i, y_j^i) \quad (4)$$

1.3 Results and Discussion

TreeDNN is developed with the agenda of solving the issues raised by loading and inferencing through multiple camera models or related systems. TreeDNN is a type of multi-task learning which supports the training through multiple datasets in a joint fashion. TreeDNN provides the flexibility of keeping the trunk of the tree on the working memory and because of that only a branch of the tree is required to be invoked at the time of execution. Hence, it reduces the loading time of the model, which can be witnessed in Table 1. The trunk of the tree works as the initial layers for any of the tasks and is shared among all the tasks therefore it helps in reducing the storage requirement of the initial layers' parameters. Table 1 showcases the case study of camera models and the performance of the TreeDNN when utilized in a real environment having 8 DNNs. The specified tasks for these models could be De-mosaic, Auto-Focus, Auto-Exposure, Auto-White balance, Auto-ISO, Color correction, Brightness enhancement etc.

The performance of the TreeDNN in the present experiment can be seen in Table 2. We have considered CIFAR10, CIFAR100, Caltech101, Caltech256, SVHN datasets to train the TreeDNN with 5 branches. SOTA results are the performance of MobileNet-V2 on aforementioned datasets when trained separately. It can be realized from the results that difficult datasets like Caltech256 and Caltech101 has shown improvement in performance because of utilizing the generalized trunk in the tree which works as regularization while training. However, there is a minute drop in the performance of relatively easy datasets. TreeDNN can be deployed as a single model for all the tasks which also opens the direction of mounting new pretrained branch in an already grown TreeDNN to increase the scalability of the model for new tasks. Where the decision about selection of most appropriate branch can be taken by considering the task similarity.

References

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[2] K. Lu, J. Huang, J. Li, J. Zhou, X. Chen, and Y. Liu, "Mtl-ffdet: A multi-task learning-based model for forest fire detection," *Forests*, vol. 13, no. 9, p. 1448, 2022.

[3] B. Singh, D. Toshniwal, and S. K. Allur, "Shunt connection: An intelligent skipping of contiguous blocks for optimizing mobilenet-v2," *Neural Networks*, vol. 118, pp. 192–203, 2019.

[4] Z. Xie, J. Chen, Y. Feng, K. Zhang, and Z. Zhou, "End to end multi-task learning with attention for multi-objective fault diagnosis under small sample," *Journal of Manufacturing Systems*, vol. 62, pp. 301–316, 2022.

[5] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1930–1939.

[6] F. Ott, D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler, "Joint classification and trajectory regression of online handwriting using a multi-task learning approach," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2022, pp. 266–276.

[7] J. G. Santiago, F. Schenkel, and W. Middelmann, "Self-supervised multi-task learning for semantic segmentation of urban scenes," in *Image and Signal Processing for Remote Sensing XXVII*, vol. 11862. SPIE, 2021, pp. 115–122.

[8] A. S. Keceli, A. Kaya, C. Catal, and B. Tekinerdogan, "Deep learning-based multi-task prediction system for plant disease and species detection," *Ecological Informatics*, p. 101679, 2022.