Directed Acyclic Graph-based Neural Networks for Tunable Low-Power Computer Vision

Abhinav Goel, Caleb Tung, Nick Eliopoulos, Xiao Hu, George K. Thiruvathukal*, James C. Davis, Yung-Hsiang Lu
Purdue University, *Loyola University Chicago
USA

ABSTRACT

Processing visual data on mobile devices has many applications, e.g., emergency response and tracking. State-of-the-art computer vision techniques rely on large Deep Neural Networks (DNNs) that are usually too power-hungry to be deployed on resource-constrained edge devices. Many techniques improve DNN efficiency of DNNs by compromising accuracy. However, the accuracy and efficiency of these techniques cannot be adapted for diverse edge applications with different hardware constraints and accuracy requirements. This paper demonstrates that a recent, efficient tree-based DNN architecture, called the hierarchical DNN, can be converted into a Directed Acyclic Graph-based (DAG) architecture to provide tunable accuracy-efficiency tradeoff options. We propose a systematic method that identifies the connections that must be added to convert the tree to a DAG to improve accuracy. We conduct experiments on popular edge devices and show that increasing the connectivity of the DAG improves the accuracy to within 1% of the existing high accuracy techniques. Our approach requires 93% less memory, 43% less energy, and 49% fewer operations than the high accuracy techniques, thus providing more accuracy-efficiency configurations.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Hardware** → **Power and energy**;

ACM Reference format:
Abhinav Goel, Caleb Tung, Nick Eliopoulos, Xiao Hu, George K. Thiruvathukal*, James C. Davis, Yung-Hsiang Lu. 2022. Directed Acyclic Graph-based Neural Networks for Tunable Low-Power Computer Vision . In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design, Boston, MA, USA, August 1–3, 2022 (ISLPED '22),* 6 pages.
https://doi.org/10.1145/3531437.3539723

1 INTRODUCTION

Deep Neural Networks (DNNs) are the state-of-the-art techniques to process visual data [8, 7]. In many applications, visual data needs to be processed on edge devices due to latency, privacy, or network bandwidth constraints. However, the significant computational requirements of DNNs limit their deployability on most resource-constrained edge devices [10]. In such scenarios, low-power DNN inference techniques are used to improve efficiency [19].

Most existing low-power techniques optimize DNNs to increase efficiency at the expense of accuracy [10, 5]. In some edge applications, e.g., biometric authentication, the best possible accuracy is required, whereas in applications like forest surveillance, long battery life is essential. However, with the existing techniques, the compromise between accuracy and efficiency cannot be tuned, thereby resulting in DNNs that are either too inaccurate or too inefficient for the target application [17, 1].

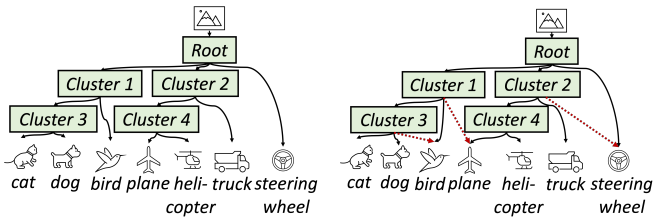

Figure 1: Toy example highlighting the contributions of this work. (a) Existing hierarchical DNNs: multiple small DNNs in the form of a tree. DNNs along a single root-leaf path are used to process an image. A misclassification at a node cannot be corrected by a subsequent node [5, 4, 13]. (b) Proposed method: Additional connections (red dotted lines) to build a Directed Acyclic Graph (DAG). With multiple root-leaf paths, a misclassification can be corrected to improve accuracy.

To address the lack of flexibility of the existing low-power techniques, some existing studies use gating networks to enable or disable certain DNN layers [17, 1]. The conditional inference allows these techniques to decrease the overhead of existing large DNNs. These techniques cannot be applied to improve the accuracy of existing low-power DNNs.

In this work, we present methods to provide varying accuracy-efficiency tradeoff configurations of a recent low-power DNN architecture, known as the hierarchical DNN [5, 6]. The hierarchical DNN architecture (cf. Section 2.1.1) is depicted in Figure 1(a); this technique employs a divide-and-conquer strategy to improve efficiency. Each input image follows a single root-leaf path. The main reason for the accuracy loss in hierarchical DNNs is because a mistake by a DNN (puts image into a wrong branch) cannot be corrected by its descendants. For example, in Figure 1(a), if an image of a plane gets misclassified to Cluster 1 instead of Cluster 2, there is no way to direct the image to the correct output.

This paper proposes a method that adds connections between nodes of a hierarchical DNN to convert the tree into a Directed Acyclic Graph (DAG). Because there are multiple paths from the root to the leaves, the DAG-based methods improve the accuracy. Figure 1(b) highlights our approach. The red dotted lines are the new connections added to convert the tree into a DAG. For example, planes and birds may look similar, but in Figure 1(a), if an image of a plane is misclassified by the Root into Cluster 1, it cannot later be corrected, whereas in Figure 1(b), it can. Adding new connections, however, increases the memory requirement of the hierarchical

DNN. Thus, the proposed method identifies the connections that should be added to provide the required improvement in accuracy with only a small increase in memory. By varying the connectivity of the DAG this method can tune the accuracy-efficiency tradeoff provided by hierarchical DNNs.

To evaluate this approach, our experiments compare the image classification accuracy and efficiency of the different techniques. This paper considers different computer vision datasets with varying input resolutions and tree structures to show that this method can vary the accuracy-efficiency tradeoff for different workload types. We also deploy the models using popular embedded devices, Raspberry Pi 4B and NVIDIA Jetson Nano, to compare the energy consumption and latency. Using the proposed approach, we observe an ~3% improvement in accuracy when compared with hierarchical DNNs. We find that, even with the extra connections, the memory, FLOPs, energy, and latency are still 93%, 49%, 43%, and 43% lower than the techniques with state-of-the-art accuracy, respectively. Thus, this method bridges the accuracy gap between hierarchical DNNs and the state-of-the-art DNNs.

2 BACKGROUND AND RELATED WORK

2.1 Efficient DNN Inference

2.1.1 Hierarchical Deep Neural Networks. The hierarchical DNN is a recently proposed tree-based DNN architecture that compromises accuracy for efficiency [5, 4, 13, 12]. As seen in Figure 1(a), hierarchical DNNs use a tree of small DNNs, where each DNN performs an intermediate classification between clusters of similar categories. Hierarchical DNNs are ~50% more efficient than conventional DNNs because each input gets processed by a subset of the small DNNs.

Our prior work finds that although hierarchical DNNs are efficient, they are less accurate than conventional DNNs. They accuracy decreases by ~4% [5] because a misclassification at a node puts the input on an incorrect branch that cannot be corrected by the offsprings. Thus, the errors at each level of the tree compound.

2.1.2 Other DNN Optimizations. There are other DNN optimization techniques that increase DNN efficiency [19, 18, 3]. Quantization and pruning [10] reduce the memory requirement of DNNs. MobileNet [14] is a DNN micro-architecture that uses bottleneck layers to reduce the number of parameters.

In these efficient DNN inference techniques, the compromise between accuracy and efficiency is fixed and cannot be tuned. We propose a method to add new connections to a hierarchical DNNs to build Directed Acyclic Graph-based DNNs. By varying the number of connections, we can tune the accuracy and efficiency of hierarchical DNNs.

2.2 Adaptive DNN Workloads

Some techniques re-configure the DNN structure based on the input. In these techniques, different DNN layers and connections are deactivated to vary the workload. The Slimmable Network [22] is an example of one such technique; it adjusts the DNN's width by using a switchable batch normalization layer. GaterNet [1] is another technique that uses a *gating network* to deactivate certain DNN connections. Throttleable Neural Networks [17] use a context-aware reinforcement learning controller that can manipulate the DNN workload. These techniques are effective in reducing the DNN workload while also lowering the accuracy. They cannot be used to increase the accuracy of an existing low-power DNN, e.g., hierarchical DNN. Our method focuses on increasing the accuracy of existing hierarchical DNNs.

Neural Architecture Search (NAS) uses a cost function and optimally tunes the DNN performance [21]. There are two ways to use NAS to tune the accuracy and efficiency of hierarchical DNNs: (1) convert the tree to a DAG by adding connections, and (2) build larger DNNs for each node of the tree. This paper focuses on the first technique, and relegates the latter to future work.

2.3 Our Contributions

(1) This is the first method to improve the accuracy of hierarchical DNNs by converting the tree into a directed acyclic graph. (2) We develop methods to select the connections to add to a hierarchical DNN that results in the desired accuracy improvement with low memory overhead. Our connection selection method does not need to evaluate all possible combinations of connections, thus making it feasible for large hierarchies. (3) Experiments show that our method outperforms existing state-of-the-art techniques in terms of memory, number of operations, and energy.

3 DIRECTED ACYCLIC GRAPH-BASED NEURAL NETWORK

This section describes our proposed method to convert hierarchical DNNs into Directed Acyclic Graphs (DAG). We describe our method for identifying the new connections that should be added to existing hierarchical DNNs for accuracy improvements with low overhead.

The tree structure is the main reason for the accuracy losses observed in existing hierarchical DNNs [5]. The tree structure allows only a single path from the root to every leaf. Thus, a misclassification made at a node cannot be corrected by its subsequent offspring nodes. Even with large DNNs at every node, no known solution achieves 100% accuracy. Therefore, the test error gets compounded deeper in the tree. Figure 2 shows a part of the tree constructed for the EMNIST handwriting dataset, along with the percentage (red text) of (a) lower-case *n* images and (b) upper-case *R* images, that are classified onto different branches. When classifying images of lower-case *n*, we see that 13.7% of images are classified as the number "9". Similarly, when classifying upper-case *R* images, 11.5% of images are misclassified into Cluster 1 instead of Cluster 2.

By converting the tree into a DAG, we create multiple paths from the root to the leaves, as depicted with red dotted lines in Figure 2(c,d). Here, mislassifications can be corrected by subsequent nodes to improve accuracy. For example, using the connection depicted in Figure 2(d), the 11.5% of upper-case *R* images that are classified into Cluster 1, instead of Cluster 2, can still reach the correct output.

In this section, we present a method to find connections that improve accuracy with only a small overhead. We first highlight the impact of adding new connections, to the overall accuracy and memory requirement of the hierarchical DNN. Then we present a method that decides which connections should be added to the tree to achieve a required accuracy improvement with low overhead.

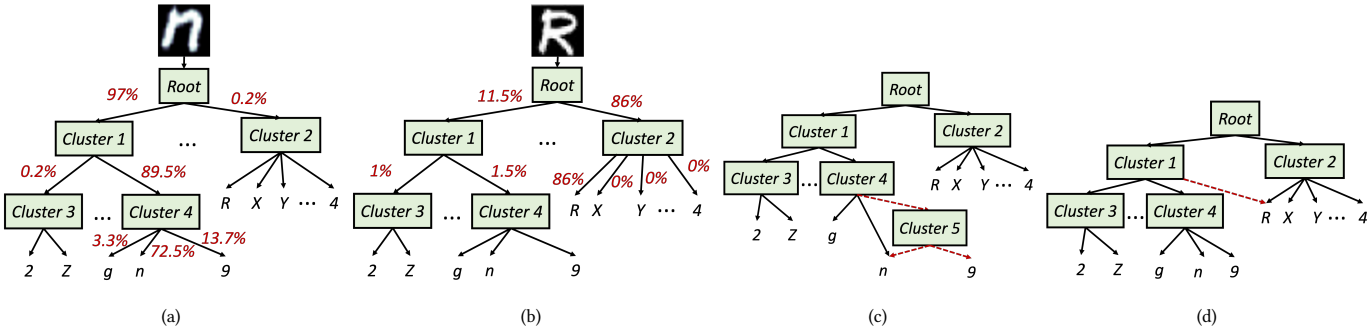

Figure 2: Portion of the hierarchical DNN constructed (using methods in Goel et al. [5]) for the EMNIST dataset. The percentages of (a) lower-case *n* and (b) upper-case *R* images, that are misclassified are depicted in red font. (c) Connections added to correct misclassifications of (c) lower-case *n* at Cluster 4 (*leaf connection*) and (d) upper-case *R* at the Root (*cluster connection*).

3.1 Impact of New Connections on Accuracy and Memory

To understand the overhead associated with adding different connections, we first analyze the types of connections that can be added to a tree. There are two types of connections, depending on the branch where images of an object category are misclassified. (1) *Leaf connections*, added when images are misclassified as another leaf in the tree. For example, in Figure 2(a), lower-case *n* is often misclassified as the number "9" (a leaf of the tree). In this case, a new cluster needs to be added to the tree, e.g., Cluster 5 in Figure 2(c). (2) *Cluster connections*, added when images are misclassified as another cluster. For example, in Figure 2(b), upper-case *R* is misclassified into Cluster 1. Here, new clusters are not required, e.g., the connection between Cluster 1 and *R* in Figure 2(d).

Adding new connections improves the accuracy of the hierarchical DNN. The accuracy improvement for each connection can be estimated by analyzing the misclassifications of the object categories. The misclassification analysis, depicted in Figure 2(a,b), identifies the frequency with which images of the different object categories are classified onto the different branches of the tree. For example, in Figure 2(b), 11.5% of upper-case *R* images are misclassified into Cluster 1. A connection between Cluster 1 and upper-case *R* enables a Maximum Accuracy (*MA*) increase of 11.5% on upper-case *R* images. Because upper-case *R* images are 2.12% of the entire dataset, the maximum accuracy increase is $MA = 2.12\% \times 11.5\% = 0.24\%$. The estimated accuracy improvement also depends on the DNN Accuracy (*DA*) of the node where the new connection is being added. This is because not all images will be classified correctly onto the new connection. For example, if the DNN at Cluster 1 classifies 90% of images correctly ($DA = 90\%$), then we estimate the accuracy improvement obtained by adding a connection between Cluster 1 and upper-case *R* as $0.24\% \times 90\% = 0.225\%$. Thus, the estimated accuracy improvement with each connection is $MA \times DA$.

Table 1: The average DNN model sizes (in MB) required in a hierarchical DNN as the fan-out changes. When constructing the hierarchical DNN, we vary the fan-out to find the relationship between the fan-out and DNN model size for a given dataset. As the fan-out increases, larger models are needed to maintain the classification accuracy.

Fan-out	2	4	8	16	32
C-256	0.20	0.25	0.34	0.45	3.00
EMNIST	0.07	0.12	0.16	0.21	1.50

Each connection offers a different accuracy improvement. Similarly, as a new connection either adds a new cluster (leaf connection) or increases the fan-out (cluster connection), the connections increase the overall memory requirement by different amounts. When a cluster is added, the overall memory requirement increases by the new DNN's model size. When the fan-out of a cluster increases, the associated DNN needs to process and distinguish more images, thus a larger DNN is required to maintain the same accuracy.

To estimate the increase in memory requirement, we calculate the average size of DNNs constructed with different fan-outs. To vary the fan-out, we use hierarchical K-Means Clustering [20] with different numbers of clusters. We follow the techniques used in prior studies and perform K-Means Clustering on the feature vectors of images extracted from a pre-trained DNN. The average model sizes for different fan-outs obtained with the Caltech-256 (C-256) and EMNIST datasets are presented in Table 1. For example, for the Caltech-256 dataset, as the fan-out increases from 16 to 32, the average DNN model size increases from 450 KB to 3 MB. This super-linear growth follows trends in DNNs: accurately classifying between many types of inputs requires DNNs with many layers and channels [7].

Using the information in Table 1, we estimate the memory costs of the different connections. For a *leaf connection*, the memory requirement of the new DNN is estimated by using the fan-out of the new cluster. When adding a *cluster connection*, the memory requirement increase is estimated using the percentage by which the average memory requirement increases with the new fan-out.

The accuracy improvement and memory requirement increase also depend on the content of images and the position of DNN within the hierarchy. To accelerate training, the estimates presented in this section only consider the number of incorrectly classified images and the fan-out. Our experiments show that these estimates are useful for comparing candidate connections when building DAG-based DNNs with varying accuracy-efficiency tradeoffs.

3.2 Selecting the Connections to Add

Adding connections to a hierarchical DNN converts it to a Directed Acyclic Graph and improves accuracy. However, as discussed in the

Table 2: Mean Squared Error (MSE) between the memory requirement obtained with the different techniques when compared with the optimal solution. Exhaustive search obtains the optimal solution by satisfying the constraint in eqt. (1), and thus the MSE on the memory requirement with exhaustive search is zero. We show the MSE with our proposed *improved greedy* search method for different values of n, where $|C|$ is the total number of candidate connections. We also compare the number of connection combinations that need to be evaluated with each technique.

Technique	Exhaustive	Greedy	Proposed Improved Greedy				
			$n = 0.2\times\|C\|$	$n = 0.4\times\|C\|$	$n = 0.6\times\|C\|$	$n = 0.8\times\|C\|$	$n = \|C\|$
Mean Squared Error	0.000	3.523	5.710	0.041	0.026	0.006	0.000
# Evaluations	3×10^7	625	32	1024	3×10^4	1×10^6	3×10^7

previous section, adding connections also increases the memory requirement and consequently reduces the efficiency of hierarchical DNNs. In this section, we describe and compare three methods to decide which subset of connections should be added for a given accuracy requirement or memory constraint: (1) exhaustive search; (2) greedy selection; and (3) our proposed improved greedy method.

For a tree with K nodes, there are $K \times (K-1)$ possible connections between nodes. Candidate connections are the subset of connections that offer a significant accuracy improvement. Given all possible candidate connections C, and the target accuracy requirement increase Δa_t, exhaustive search finds the optimal set of connections $E \subset C$ to add to the hierarchical DNN. The constraints for the exhaustive search is presented in eqt. (1). Here, a_i is the estimated accuracy improvement and m_i is the estimated memory increase obtained by adding connection $e_i \in E$.

$$\text{Minimize: } \sum_{i=0}^{|E|} m_i, \text{ subject to: } \sum_{i=0}^{|E|} a_i \geq \Delta a_t \tag{1}$$

Exhaustive search always finds the set of connections E that obtains the required Δa_t accuracy improvement with the least memory overhead. The constraint can also be modified to find the DAG with the maximum accuracy, given a memory limit, Δm_t. However, with either optimization constraint, there are $2^{|C|}$ possible combinations of connections that need to be evaluated. Thus, the exhaustive search is feasible only for small hierarchies where the number of candidate connections is also small. For the Caltech-256 dataset, the tree structure has 13 nodes, and 25 candidate connections. Because of the small number of candidate connections, we use this dataset in Table 2 to compare the relative memory overhead and the computation costs (# Evaluations) associated with the different techniques.

The greedy selection assigns a score, s, to each candidate connection and sorts the connections based on this score. This method iteratively selects the connections with the highest score until the target accuracy increase Δa_t (or target memory requirement Δm_t) is reached. Prior studies on greedy techniques suggest that the score should be the ratio of the value of a connection to its associated cost [2]. Thus, the score assigned to each candidate connection is given by a_i/m_i. As mentioned in Section 3.1, the estimated accuracy increase (a_i) is given by the product of the connection's DNN accuracy and the maximum accuracy increase; the memory requirement (m_i) is obtained using the hierarchical DNN characterization in Table 1. The greedy selection method reduces the computational costs associated with selecting connections to add to the hierarchical DNN, but may result in a model with a larger memory requirement. This can be understood with the following example, if a 0.50% accuracy improvement is needed, a connection with a 0.90% accuracy improvement and 900 KB overhead will be selected before a connection with 0.50% accuracy improvement and 550 KB memory overhead. Table 2 shows that the greedy technique has lower computation costs to select connections that meet the accuracy requirement, but generally results in larger models.

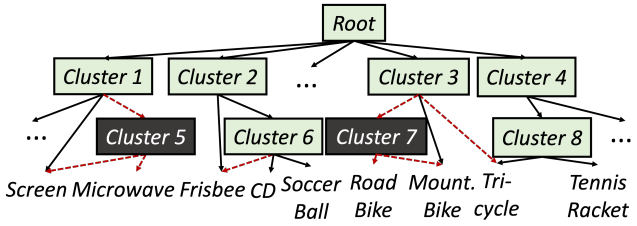

Figure 3: Portion of the Directed Acyclic Graph constructed for the Caltech-256 dataset with our proposed method. The red dotted lines depict the new connections. The darker clusters are new clusters added for the leaf connections.

Finally, we propose an improved greedy method that uses the connection ranking from greedy selection to reduce the overhead of exhaustive search. Here, the greedy selection is first used to rank the candidate connections. Instead of performing exhaustive search on every single candidate connection, our method performs exhaustive search on the n-highest ranked connections, where n is a hyper-parameter. By only searching for subsets of a set N, where $|N| = n$ and $N \subset C$, the computation cost decreases. As seen in Table 2, when we consider the top ~40% connections, the improved greedy approach has results similar to exhaustive search, while requiring fewer evaluations. Figure 3 shows a portion of the DAG constructed for the Caltech-256 dataset using our method. A combination of leaf (e.g., Cluster 5 and *screen*) and cluster (e.g., Cluster 6 and *frisbee*) connections are added.

4 EXPERIMENTS AND RESULTS

This section shows that a PyTorch implementation of the proposed method improves the accuracy of existing hierarchical DNNs. Our method offers multiple accuracy-efficiency tradeoff options.

4.1 Datasets Used

We use three image datasets to evaluate our method: Extended MNIST (EMNIST), Caltech-256, and Tiny-ImageNet. We select these datasets because they have varied tree structures and input resolutions. The tree structure contains 13 nodes for Caltech-256, 14

nodes for EMNIST, and 31 nodes for Tiny-ImageNet. We use a subset of Caltech-256 to be consistent with prior work on hierarchical DNNs [5, 9]. EMNIST contains 28×28 gray scale images of English characters and numbers. Caltech-256 and Tiny-ImageNet have color images of size 224×224 and 64×64, respectively.

4.2 Experimental Setup

4.2.1 Metrics. The accuracy is the percentage of the images classified correctly from the test set. To compare the efficiency, we report the memory requirement (model size) and numbers of operations (FLOPs) per image. These are found using the *torchinfo* library. In the baseline hierarchical DNN and our proposed method, we report average memory requirement and FLOPs over all test set images, because the root-leaf paths have varying lengths. We use a Yokogawa WT310E Power Meter to measure the energy consumption of the techniques on a Raspberry Pi 4B and NVIDIA Jetson Nano.

4.2.2 Directed Acyclic Graph-based Model Construction. We use the methods described in Goel et al. [5] to construct and train the efficient baseline hierarchical DNN. Our technique does not change the DNN architecture used at each node of the tree (as described in Section 2). During the construction of the hierarchical DNN, we vary the fan-out and observe the changes to the model size and accuracy of the DNNs at each node. This is used to collect the information in Table 1 for the evaluation of the candidate connections. The accuracy improvement and memory overhead estimates are used to evaluate the connections to reduce the DAG construction time. We use the *improved greedy* approach (Section 3.2) to select the connections that create the DAG.

With this approach, we first use experiments to decide the number of candidate connections (n) to evaluate. In most cases, we find that $n = 40\% \times |C|$ is appropriate for finding connections. For every new connection added, we use the PyTorch implementation of categorical cross entropy and back-propagation. Only the DNNs where new connections are added and their corresponding sub-trees are re-trained. The other DNNs of the tree are unchanged. We report results of DAG-based DNNs with different numbers of connections in Tables 3 and 4.

4.2.3 Comparison with state-of-the-art. To show that different configurations of the Direct Acyclic Graph-based method can tune the accuracy and efficiency of low-power computer vision, we compare with representative low-power computer vision techniques that are focused on: (1) high efficiency—hierarchical DNN [5]; and (2) high accuracy—MobileNetV2 [14], VGG-5 [16], and ResNet [7, 15]. In particular, we compare with: VGG-5 and ResNet9 on the EMNIST dataset; MobileNetV2 [14] on the subset of Caltech-256; MobileNetV2 [14] and ResNet50 (as described in Jeevan et al. [15]) on the Tiny-ImageNet dataset. To maintain a fair comparison, we use the same dataset augmentation used in prior studies [15].

4.2.4 Evaluation of Improved Greedy Technique. We create a DAG constructed by adding connections at random to show the value of our method to evaluate the candidate connections. The reported results are based on an average of five runs. For each run, different connections, selected at random, are added to the DAG.

Table 3: Comparison of different variants of the proposed DAG-based DNN (DAG-Net) with the hierarchical DNN baseline (HDNN) [5] and state-of-the-art techniques.

Dataset	Technique	Accuracy (%)	Model Size (MB)	FLOPs ($\times 10^6$)
EMNIST	HDNN [5]	91.20	0.25	2.13
	DAG-Net 1	91.30	0.27	2.14
	DAG-Net 2	91.70	0.28	2.17
	DAG-Net 3	92.00	0.29	2.79
	DAG-Net 4	92.14	0.32	3.21
	DAG-Net 5	92.15	0.37	3.45
	VGG-5 [16]	92.59	15.00	161.24
	ResNet9 [7]	92.00	26.00	636.71
C-256	HDNN [5]	88.90	0.58	147.90
	DAG-Net 1	90.35	0.59	148.50
	DAG-Net 2	90.65	0.60	148.83
	DAG-Net 3	91.10	0.61	149.19
	DAG-Net 4	91.80	0.62	149.46
	DAG-Net 5	92.00	0.64	149.90
	DAG-Net 6	92.05	0.68	151.00
	MobileNetV2 [14]	93.30	10.00	300.00
Tiny ImageNet	HDNN [5]	47.19	0.47	11.60
	DAG-Net 1	48.30	0.48	11.69
	DAG-Net 2	48.90	0.49	11.74
	DAG-Net 3	49.41	0.51	11.91
	DAG-Net 4	49.90	0.53	12.12
	DAG-Net 5	50.01	0.56	12.46
	ResNet50 [15]	48.77	95.67	334.11
	MobileNetV2 [14]	52.00	9.95	24.74

4.3 Results

Table 3 compares the accuracy, model size, and number of operations (FLOPs) of the different techniques. We present results of configurations (with increasing numbers of connections) of our proposed Direct Acyclic Graph-based method (denoted "DAG-Net i" in Table 3) till the accuracy improvement saturates. When using the Caltech-256 dataset, different configurations of our method achieve classification accuracy in the range of 90.35% – 92.05% with 0.59MB – 0.68MB memory requirement per image. The total memory requirement (sum of all DNN model sizes) of our DAG-based method ranges between 2.2MB and 5.8MB for the Caltech-256 dataset; still considerably smaller than the 10MB required by MobileNetV2. By having different numbers of connections in the Direct Acyclic Graph-based method, we also vary the number of operations performed during inference. In the EMNIST dataset, the average numbers of operations per image performed by our method lie in the range 2.14×10^6 – 3.45×10^6. These results show that our proposed method provides a tunable accuracy-efficiency tradeoff between hierarchical DNNs and the more accurate methods.

Table 4 shows the processing time and energy consumption of the different techniques on two commonly used embedded devices. The results are reported after averaging over all images in the testing set. When compared with MobileNetV2 on the Raspberry Pi 4B, our technique requires 43% ($1 - \frac{7.52}{13.36}$) less energy and processing time. The processing time of EMNIST dataset is higher on the NVIDIA

Table 4: Processing time (sec/img) and energy consumption (J/img) comparison on two embedded devices.

Dataset	Technique	Raspberry Pi 4B		NVIDIA Jetson Nano	
		Latency	Energy	Latency	Energy
EMNIST	HDNN	0.053	0.28	0.320	2.46
	DAG-Net 1	0.057	0.30	0.320	2.47
	DAG-Net 3	0.062	0.32	0.322	2.49
	DAG-Net 5	0.066	0.35	0.322	2.49
	VGG-5	0.431	2.27	4.041	31.15
Tiny ImageNet	HDNN	1.299	6.79	0.341	2.63
	DAG-Net 1	1.315	6.92	0.347	2.67
	DAG-Net 3	1.362	7.20	0.350	2.69
	DAG-Net 5	1.425	7.52	0.355	2.72
	MobileNetV2	2.501	13.36	2.007	15.43

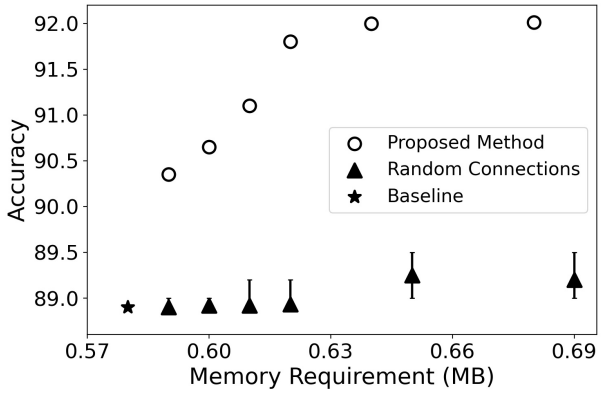

Figure 4: Evaluation of the proposed *improved greedy* approach to select connections to convert a tree into a DAG. The proposed method consistently achieves higher accuracy than a method that selects connections at random, on the Caltech-256 dataset. The errors bars indicate the maximum and minimum accuracy values obtained in the random trials.

Jetson Nano because too few GPU cores are used when the input resolution is small, thus leading to high overhead [11]. Due to space constraints, we only present results for the EMNIST and Tiny ImageNet datasets. We observe similar results with Caltech-256.

We evaluate the effectiveness of our *improved greedy* method used to convert a hierarchical DNN to a DAG, by comparing it with a method that selects connections at random. Figure 4 shows the classification accuracy obtained with the two techniques for varying memory constraints. As the memory capacity increases, our proposed method is able to improve accuracy. The random selection method does not always select connections that result in substantial increases of accuracy.

5 CONCLUSIONS

Existing hierarchical DNNs use a divide-and-conquer approach to improve the efficiency of computer vision at the expense of accuracy. In this paper, we present a novel method to convert existing hierarchical DNNs to Directed Acylic Graphs (DAGs) with tunable accuracy-efficiency tradeoff configurations. Our approach identifies the connections to add between nodes of a hierarchical DNN, obtaining the required accuracy improvement with a small overhead. This paper is among the first to develop a method that estimates the accuracy improvement and the memory cost of each possible additional connection in a hierarchical DNN. We then develop an *improved greedy* approach that finds the combination of connections that varies the accuracy-efficiency tradeoff. This technique does not need to evaluate every possible combination of connections, allowing our method to scale to large hierarchical DNNs. After evaluating on different datasets, we show that our method successfully bridges the accuracy gap between existing hierarchical DNNs and the more accurate DNNs. Our experiments on two varied embedded devices, Raspberry Pi 4B and NVIDIA Jetson Nano, also confirm that the proposed method is more energy-efficient than the existing techniques with state-of-the-art accuracy.

ACKNOWLEDGEMENT

This project was supported in part by NSF CNS-1925713, NSF OAC-2107230, NSF OAC-2104709, and NSF OAC-2107020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Z. Chen et al. 2019. You look twice: gaternet for dynamic filter selection in cnns. In *IEEE CVPR*.
[2] G. B. Dantzig. 1957. Discrete-variable extremum problems. In *Operations Research*.
[3] A. Goel et al. 2020. A survey of methods for low-power deep learning and computer vision. In *IEEE WF-IoT*.
[4] A. Goel et al. 2020. Low-power object counting with hierarchical neural networks. In *ACM ISLPED*.
[5] A. Goel et al. 2020. Modular neural networks for low-power image classification on embedded devices. In *ACM TODAES*.
[6] D. Roy et al. 2020. Tree-CNN: A Hierarchical Deep Convolutional Neural Network for Incremental Learning. In *Neural Networks*.
[7] K. He et al. 2016. Deep Residual Learning for Image Recognition. In *IEEE CVPR*, 770–778.
[8] K. Simonyan et al. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, (Sept. 2014).
[9] P. Panda et al. 2017. FALCON: Feature Driven Selective Classification for Energy-Efficient Image Recognition. In *TCAD*.
[10] S. Han et al. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *arXiv:1510.00149*.
[11] B. Fu et al. 2017. Parallel video processing using embedded computers. In *IEEE GlobalSIP*.
[12] A. Goel et al. 2022. Efficient computer vision on edge devices with pipeline-parallel hierarchical neural networks. In *ASP-DAC*.
[13] A. Goel et al. 2021. Low-power multi-camera object re-identification using hierarchical neural networks. In *IEEE/ACM ISLPED*.
[14] A. Howard et al. 2019. Searching for mobilenetv3. In *IEEE ICCV*.
[15] P. Jeevan et al. 2022. Wavemix: resource-efficient token mixing for images. In *arXiv 2203.03689*.
[16] HM Kabir et al. 2020. Spinalnet: deep neural network with gradual input. In *arXiv 2007.03347*.
[17] H. Liu et al. [n. d.] 2020 dynamically throttleable neural networks (TNN). In *arXiv 2011.02836*.
[18] D. Marculescu. 2021. When climate meets machine learning: edge to cloud ml energy efficiency. In *IEEE/ACM ISLPED*.
[19] G. K. Thiruvathukal et al. 2022. Low-Power Computer Vision: Improve the Efficiency of Artificial Intelligence. In CRC Press.
[20] A. Vedaldi et al. 2010. Vlfeat: an open and portable library of computer vision algorithms. In *ACM MM*.
[21] B. Wu et al. 2019. Fbnet: hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE CVPR*.
[22] J. Yu et al. 2019. Slimmable neural networks. In *ICLR*.