Modular Neural Networks for Low-Power Image Classification on Embedded Devices

Abhinav Goel
Purdue University

Sara Aghajanzadeh
Purdue University

Caleb Tung
Purdue University

Shuo-Han Chen
Academica Sinica, Taipei, Taiwan

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

See next page for additional authors

Recommended Citation

Abhinav Goel, Sara Aghajanzadeh, Caleb Tung, Shuo-Han Chen, George K. Thiruvathukal, and Yung-Hsiang Lu. 2020. Modular Neural Networks for Low-Power Image Classification on Embedded Devices. ACM Trans. Des. Autom. Electron. Syst. 26, 1, Article 1 (October 2020), 35 pages.

Modular Neural Networks for Low-Power Image Classification on Embedded Devices

ABHINAV GOEL, SARA AGHAJANZADEH, and CALEB TUNG, Purdue University
SHUO-HAN CHEN, National Taipei University of Technology
GEORGE K. THIRUVATHUKAL, Loyola University Chicago
YUNG-HSIANG LU, Purdue University

Embedded devices are generally small, battery-powered computers with limited hardware resources. It is difficult to run deep neural networks (DNNs) on these devices, because DNNs perform millions of operations and consume significant amounts of energy. Prior research has shown that a considerable number of a DNN's memory accesses and computation are redundant when performing tasks like image classification. To reduce this redundancy and thereby reduce the energy consumption of DNNs, we introduce the Modular Neural Network Tree architecture. Instead of using one large DNN for the classifier, this architecture uses multiple smaller DNNs (called *modules*) to progressively classify images into groups of categories based on a novel visual similarity metric. Once a group of categories is selected by a module, another module then continues to distinguish among the similar categories within the selected group. This process is repeated over multiple modules until we are left with a single category. The computation needed to distinguish dissimilar groups is avoided, thus reducing redundant operations, memory accesses, and energy. Experimental results using several image datasets reveal the effectiveness of our proposed solution to reduce memory requirements by 50% to 99%, inference time by 55% to 95%, energy consumption by 52% to 94%, and the number of operations by 15% to 99% when compared with existing DNN architectures, running on two different embedded systems: Raspberry Pi 3 and Raspberry Pi Zero.

CCS Concepts: • **Computing methodologies → Neural networks**; *Computer vision*; • **Computer systems organization → Embedded systems**; • **Hardware → Power and energy**;

Additional Key Words and Phrases: Low-power, image classification

ACM Reference format:
Abhinav Goel, Sara Aghajanzadeh, Caleb Tung, Shuo-Han Chen, George K. Thiruvathukal, and Yung-Hsiang Lu. 2020. Modular Neural Networks for Low-Power Image Classification on Embedded Devices. *ACM Trans. Des. Autom. Electron. Syst.* 26, 1, Article 1 (October 2020), 35 pages.
https://doi.org/10.1145/3408062

This project was supported in part by NSF OAC-1747694, OAC-1535108, CNS-1925713, Google, Facebook, Intel, and Argonne National Laboratory. 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.
Authors' addresses: A. Goel, S. Aghajanzadeh, C. Tung, and Y.-H. Lu, Purdue University, MSEE 222, School of Electrical and Computer Engineering, Materials and Electrical Engineering Building, 501 Northwestern Ave., West Lafayette, Indiana 47907-2035; emails: {goel39, saghajan, tung3, yunglu}@purdue.edu; S.-H. Chen, Department of Computer Science and Information Engineering, National Taipei University of Technology, Room 331, Technology Building, 1, Sec. 3, Chung-Hsiao E. Rd., Taipei 106, Taiwan, Republic of China; email: qoolili@iis.sinica.edu.tw; G. K. Thiruvathukal, Loyola University Chicago, Doyle 301, Doyle Center, Lake Shore Campus, 1052 W Loyola Ave, Chicago, IL 60626; email: gkt@cs.luc.edu.

1084-4309/2020/10-ART1 $15.00
https://doi.org/10.1145/3408062

1 INTRODUCTION

Cameras are widely deployed in many embedded systems (called *Internet of Video Things* [1]), and there is a need to improve the efficiency of computer vision running on embedded devices with limited hardware resources [2–6]. This article analyzes and tackles the problems associated with performing image classification on embedded devices. Image classification is a supervised learning problem: assigning a single label from a set of categories (objects to identify in images) to every input image, such as a dog or a car. Recent advances in deep neural networks (DNNs) trained on millions of images have achieved high accuracy for image classification [7]. However, most DNNs are designed for scenarios where computational resources are abundant [8]. They are not suitable for devices where energy efficiency is critical, such as drones and wearable devices [2, 9–12].

Most DNNs, like VGG [13] and ResNet [7], have monolithic architectures as seen in Figure 1. Such an architecture is a single DNN responsible for identifying and processing all features associated with all categories to make decisions. The DNN has to perform many different tasks that require a large number of neurons and layers. However, when processing each image, only a small number of these neurons have significant activations [14], thus leading to redundancies. These redundancies increase the energy consumption of the DNN significantly. We propose the Modular Neural Network Tree (MNN-Tree) architecture as a method to reduce these redundancies and perform image classification on embedded devices.

The proposed method first finds the visual similarity between different categories using a novel similarity metric. Similar categories are grouped into entities called *super-groups*. Similar super-groups are then grouped into larger super-groups, creating a hierarchy in the form of a tree. The MNN-Tree architecture uses several small DNNs, called *modules*, responsible for classifying between different super-groups. For an input image, once a module selects a super-group, another module further classifies among the children of the super-group. The modules associated with other super-groups are not used during the inference of that image. By doing so, only a small subset of the modules are used during inference, thus avoiding redundant operations. Figure 2 illustrates the MNN-Tree architecture, where the categories are *dog*, *cat*, *car*, and *truck*. *Dog* and *cat* form a super-group, called *animals*. *Car* and *truck* are grouped into another super-group for *vehicles*.

We propose a novel method to measure the visual similarity between categories of a dataset, called the *averaged softmax likelihood* (ASL). The similarity metric computes the output (softmax) of a DNN for a category X, averaged over all input images belonging to another category Y. The DNN's softmax output is used to quantify the confusion between categories. A high softmax output for category X (when inputs are from category Y) indicates that the DNN frequently gets confused between the two visually similar categories. ASL groups all categories that are visually similar into a single super-group automatically, whereas existing hierarchical clustering techniques are limited to grouping a fixed number of categories at each level. Our experiments show that ASL can be used to build hierarchies with the MNN-Tree architecture for lower energy consumption and faster image classification. We show that MNN-Tree built using ASL achieves 4.2% to 17.2% higher accuracy than existing hierarchical image classifiers. The proposed MNN-Tree is also evaluated against monolithic DNN architectures such as VGG, ResNet, and DenseNet on different embedded devices. Experimental results show that the MNN-Tree architecture has a 50% to 99% smaller DNN model size, 52% to 94% lower energy consumption, 55% to 95% lower inference time, and 15% to 99% fewer operations when compared with existing monolithic DNN architectures on a Raspberry Pi 3 and a Raspberry Pi Zero. Furthermore, by testing the MNN-Tree on the Extended MNIST (EMNIST), SVHN, CIFAR-10, CIFAR-100, and ImageNet datasets, we see a negligible loss to

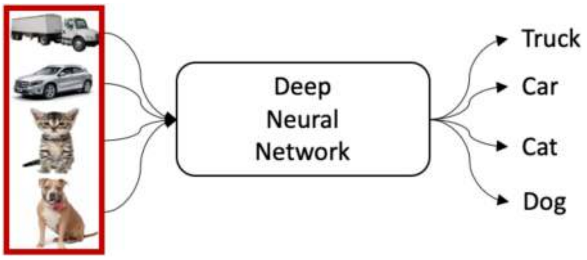


Fig. 1. Convolutional DNN. A single monolithic architecture classifies images into their corresponding categories.

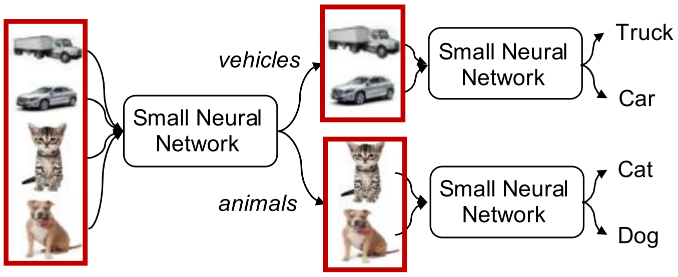


Fig. 2. The proposed solution (MNN-Tree architecture). The input image is processed incrementally using small DNNs. After detecting the type of images, finer classifications are made. If an animal is detected in the image, only the DNN for classifying cats and dogs is used. The computation for distinguishing trucks and cars is avoided.

the classification accuracy when compared with the state-of-the-art monolithic DNNs. This work makes the following contributions:

(1) The article proposes a method for constructing the MNN-Tree by grouping visually similar categories automatically. The method works consistently for different image datasets.
(2) We propose a novel visual similarity metric to find and group similar categories. This metric can build super-groups of different sizes in an MNN-Tree. This is an improvement on clustering techniques that are limited to grouping a fixed number of categories at each level of the hierarchy (e.g., the hierarchical K-means clustering algorithm).
(3) The MNN-Tree reduces redundant computation and memory accesses, thus saving energy when compared with existing monolithic DNN architectures.
(4) Our experiments demonstrate that the proposed architecture consistently outperforms other hierarchical architectures in terms of accuracy on popular image datasets. We also provide a detailed explanation of why existing solutions do not perform well.
(5) We implement the proposed method on two popular embedded devices and show consistent improvements in terms of lower energy consumption and shorter inference time.

The rest of the article is organized as follows. Section 2 provides background on DNNs (Section 2.1) and discusses the current techniques used to perform low-power computer vision (Section 2.2) and hierarchical image classification (Section 2.3). Section 3 presents the MNN-Tree architecture in greater detail. The motivation behind using the MNN-Tree is explained in Section 3.1. Section 3.2 describes the proposed similarity metric for finding visually similar categories. Section 3.3 describes the algorithm used for building the MNN-Tree. Section 3.4 explains the method

Table 1. Comparison of Different Techniques on the CIFAR-10 Dataset

Method	Decision Tree	Naive Bayes' Classifier	*K*-Nearest Neighbors	Random Forests	DNNs	
					ResNet	CondenseNet
Test Accuracy	0.272	0.290	0.417	0.491	0.931	0.966

Note: It can be seen that DNNs outperform the other techniques significantly in accuracy.

used to train each module. Section 3.5 describes how image classification is performed with the proposed technique. Section 4 presents the experimental results. Section 4.1 and Section 4.2 focus on dataset configurations and the experimental setup, respectively. Section 4.3 compares MNN-Tree with the existing monolithic DNNs, and Section 4.4 compares MNN-Tree with hierarchical image classifiers. Section 4.5 describes experiments on different embedded devices. Section 4.6 puts forward potential extensions of the proposed method. Section 5 concludes the article. The examples, source code, and DNN models are available on GitHub [15].

2 BACKGROUND AND RELATED WORK

This section provides a background on DNNs and discusses the existing methods for low-power and hierarchical computer vision. For clarity, we divide this section into three subsections. A short introduction to DNNs and a comparison with other image classification techniques is provided in Section 2.1. Section 2.2 describes different techniques to reduce the computation associated with monolithic DNNs. Section 2.3 explains related work with hierarchical image classification.

2.1 Deep Neural Networks

DNNs are a class of machine learning algorithms that can achieve high accuracy on many computer vision tasks [16, 17]. DNNs use the back-propagation algorithm to train parameters and require large amounts of training data to achieve high accuracy. DNNs generally contain several layers (convolution (conv) and fully connected (FC)). The output of each layer is called an *activation map* or a *feature map*. These feature maps are filtered versions of the input image. They highlight specific attributes of the image such as shapes, textures, and colors. Each layer performs matrix multiplications between the previous layer's feature map and the DNN parameters to create more complex feature maps. Because of the large number of feature maps constructed for each image, DNNs are generally computationally expensive and require high-performance computers for low latency. Consequently, embedded devices with limited resources are not ideal for performing image classification [2].

Decision trees [18], clustering algorithms [19], and Naive Bayes' classifiers [20] have significantly lower computational requirements when compared with DNNs [21]. However, our experimental results show that the accuracy obtained with these approaches is poor when compared with the accuracy obtained by DNNs, as reported in Table 1. The Decision Tree and Naive Bayes' classifiers obtain 27.2% and 29.0% accuracy, respectively. The low accuracy is due to overfitting, as the trained models do not generalize well to the testing set. *K*-Nearest Neighbors obtains 41.7% accuracy with $K = 30$ (K refers to the number of neighbors used in the algorithm), and Random Forests obtains 49.1% accuracy when using 512 trees. DNNs such as ResNet [7] and CondenseNet [22] obtain greater than 93% accuracy on the same dataset. This vast difference in accuracy is attributed to the depth of DNNs. As a result, the rest of this article focuses on reducing the energy consumption associated with running DNNs on embedded devices.

2.2 Low-Power DNNs

Bianco et al. [23] use *accuracy density* to measure the tradeoff between DNN accuracy and computational cost. Using this metric, the low-power DNN techniques can be broken into four major directions [24].

2.2.1 Parameter Pruning and Quantization. To reduce the energy consumption in DNNs, recent research has looked into the tradeoff between accuracy and the number of memory accesses. Binarized neural networks [25] constrain the parameters to either +1 or −1; each parameter is a single bit. Some techniques approximate or quantize DNN parameters to reduce the required amounts of memory [2, 26–28]. Jiang et al. [29] use approximate computing and parallel processing for lower DNN energy consumption.

2.2.2 Bottleneck Filters. Bottleneck filters replace large convolutional filters with compact blocks of small filters to improve the inference speeds. SqueezeNet [30] and MobileNet [31] are micro-architectures that use bottleneck layers to reduce the number of parameters. Similarly, Szegedy et al. [32] use two 1×1 filters to approximate 3×3 filters in large DNNs.

2.2.3 Low-Rank Factorization. Sironi et al. [33], Denton et al. [34], and Jaderberd et al. [35] use tensor decompositions to estimate the informative parameters to obtain lower memory requirements. Zhong et al. [36] and Li et al. [37] use reduced matrix representations with hardware and software accelerators for low-latency applications. Low-rank factorizations are difficult to implement as they involve decomposition operations that are computationally expensive on low-power devices [38].

2.2.4 Knowledge Distillation. Knowledge distillation techniques build smaller DNNs with the information from a larger DNN using a student-teacher training paradigm [39]. Ba et al. [40] compress a large DNN by training each layer of a small DNN to mimic the activations of the large DNN.

All of the techniques reduce the memory requirements, but they are still monolithic architectures with redundancies (i.e., using a single DNN to classify all categories at once). In contrast, this work proposes a hierarchy of several small DNNs to eliminate redundancies for image classification.

2.3 Hierarchical Image Classifiers

This section describes the approaches for hierarchical image classification. There are four major techniques to quantify the similarity between categories for building hierarchies for computer vision. Some of the techniques are summarized in Table 2. However, they fail to achieve high classification accuracy or high inference speed. To understand the reasons, we conduct several experiments in Section 4.4.

2.3.1 Distances Between Feature Vectors. Guérin et al. [41] use DNNs to extract features from image categories. Furthermore, Euclidean distances between the centroids of feature vectors of different categories can be used to find the similarity between categories [42]. Some techniques attempt to find similarities between images by using DNNs to learn a hash function that links the pairwise Hamming distances with the pairwise similarity [43, 44]. In the aforementioned techniques, two categories are merged into a super-group—for instance, they share a parent node in the tree if the distance (Euclidean or Hamming) between categories is smaller than a threshold. However, our experiments discover that finding the correct threshold for each level of the hierarchy and each dataset requires a significant amount of trial and error. An incorrect threshold can lead to imbalanced trees that are either too wide or too tall. Thus, using a threshold on the

Table 2. Different Techniques to Build Hierarchies for Image Classification

Techniques	Description and Deficiencies
Distances between feature vectors [41, 42]	Use the distances between the centroids of feature vectors to find similarity. All groups of categories lying within a radius of each other are grouped. Difficult to determine the optimal radius.
Hierarchical clustering [45–49]	Use the distances between the centroids of feature vectors to find similarity. The k closest categories are grouped at every stage. Fixing the value k produces poor accuracy and degenerated hierarchies.
HSV and Gabor features [50]	Use texture and color information from categories to find similarities. All categories sharing the same color template or textures are grouped. Images in the same category may have different colors and textures.
Semantic similarity [51–54]	Use semantic information from sources like WordNet to quantify the similarity. Categories that share more semantic details are grouped together. Semantic and visual similarities often do not correlate.
MNN-Tree (proposed method)	Use DNN's softmax output for a category, averaged over all input images belonging to other categories to find similarity. The softmax output is an indication of the confusion between categories. Categories that are frequently confused are grouped together.

distances between feature vectors is not ideal to build balanced hierarchical classifiers. Instead, this article presents a method for building the hierarchy without the need for finding the threshold through trial and error.

2.3.2 Clustering Techniques. Some methods build trees of classifiers using different clustering techniques [45–47]. They first find the distances between the centroids of categories and then group two or more closest categories into a single super-group [57]. There is no requirement for a threshold. The number of categories in each super-group must be fixed before clustering. Some techniques use support vector machines [48, 49] at each node of the hierarchy to obtain high inference speeds. However, as clustering techniques group a fixed number of categories into a single super-group, they do not obtain high accuracy. This is because it is not always possible to separate visually similar categories into fixed-sized super-groups, especially higher up in the tree [50]. Our experiments show that clustering techniques usually result in degenerated hierarchies because a single centroid is not enough to represent a very large cluster of many categories. Degenerated trees are usually tall and require many classifiers to make a decision, leading to a lower inference speed and higher energy consumption. In contrast, this work uses ASL because it does not require centroids of feature vectors to find the similarity between categories and thus can be used to build more balanced hierarchies. Moreover, the proposed method can build hierarchies with different-sized super-groups.

2.3.3 Gabor and HSV Transformation Techniques. A 2D Gabor filter [58] is a Gaussian function modulated by a sinusoidal plane wave and is used for texture analysis. These filters can identify frequency content in specific orientations or directions. To capture all of the textures, usually several Gabor filters with different frequencies and orientations are used. HSV (Hue-Saturation-Value) [59] is a linear color transformation on the RGB (Red-Green-Blue) color space to get feature vectors corresponding to different color components. Panda et al. [50] propose a method that uses Gabor and HSV transformations to identify common textures and colors to group categories that share similar textures and colors. However, all objects belonging to the same category may not always have the same textures or colors, as shown in Section 4.4. This work builds hierarchies that more closely account for visual similarities to obtain significantly higher accuracy in classification without using textures and colors.

2.3.4 Semantic Similarities. Some techniques [51–54] generate hierarchies using semantic similarity between categories or use pre-defined semantic hierarchies like WordNet [61]. WordNet is a large lexical database of the English language where nouns, verbs, adjectives, and adverbs are linked to one another using conceptual-semantic and lexical relations. Some techniques [62–64] use semantic information to build hash functions to quantify the similarity between different images. However, semantic similarities do not always correspond to visual similarities—for example, airplanes and birds are commonly seen in the sky (visually similar) but are semantically distant. Using semantic similarities can lead to misclassifications and poor accuracy in image classification. In contrast, this work uses visual similarity, not semantic similarity, to build the hierarchy.

2.3.5 Other Hierarchical Techniques for Computer Vision. Wang et al. [65] treat the building of a hierarchy as a bin-packing problem to minimize the number of nodes in the tree. This is done to obtain high accuracy, at the expense of increased computational requirements (2× of a monolithic DNN). Tree-CNN [66] is an incremental learning technique to train DNNs on previously unseen categories without the requirement of extensive retraining. Our method may also be used to take advantage of incremental learning as suggested in Tree-CNN [66], but this study does not consider that. Given a hierarchy of linear classifiers, Sun et at. [67] use a branch-and-bound algorithm to improve the accuracy of tree-based methods. This is achieved by traversing down multiple paths of the hierarchy simultaneously. These methods do not concentrate on reducing redundancies or the energy consumption of DNNs.

3 MODULAR NEURAL NETWORK-TREE

This section first explains the motivation behind using the MNN-Tree and describes the challenges associated with this technique. The proposed metric for quantifying the similarities between different categories of a dataset is described in Section 3.2. Then, Section 3.3 explains the algorithm that uses the similarity metric to build the MNN-Tree. The method to train the modules is explained in Section 3.4. Finally, Section 3.5 describes how image classification is performed with the MNN-Tree.

3.1 Motivation for the MNN-Tree

DNNs use many neurons in multiple layers to model complex features associated with different categories. When performing inference on a single image, only a small fraction of the neurons have significant activations (contribute meaningfully to the output). This leads to redundancies in both conv layers and FC layers [14, 68]. By reducing these redundancies, the MNN-Tree makes the DNN smaller and faster while maintaining classification accuracy.

We propose the MNN-Tree architecture to reduce the redundant operations by using only a subset of the neurons in the DNN. The MNN-Tree architecture consists of several small DNNs (called *modules*) responsible for classifying between groups of similar categories, called *super-groups*. The number and the sizes of the super-groups are not fixed and are selected dynamically based on the training data. The MNN-Tree architecture is depicted in Figure 3. The input image is processed by the root module first (analogous to the root of a tree) and is classified into one of several super-groups (dotted lines in Figure 3). The image is then processed by the module associated with the chosen super-group. This process continues until one of the leaves of the MNN-Tree is reached. The leaf modules contain the original categories of the dataset. It is ensured that, during inference, each image takes a single path from the root to one of the leaves. The unused modules that are not

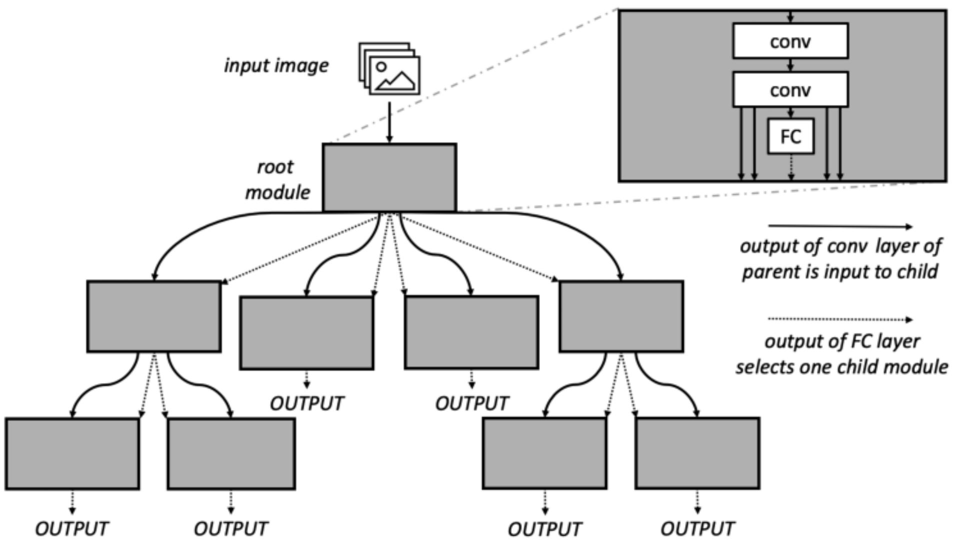

input image
root module
conv
conv
FC
output of conv layer of parent is input to child
output of FC layer selects one child module
OUTPUT
OUTPUT
OUTPUT
OUTPUT
OUTPUT
OUTPUT

Fig. 3. The image is first processed by the root module. The FC layer of a parent module classifies between its children (dotted lines). The solid lines show how the output of the conv layer (i.e., the feature map) of a parent module is used as input to its child module. This process is repeated along a path from the root to a leaf module. The leaf modules may have different distances from the root (i.e., the tree is not necessarily balanced). In addition, different parents may have different numbers of child modules (i.e., this is not necessarily a binary tree).

on the path from the root to the leaf are never loaded into memory. This reduces the memory and computation requirements, as well as energy consumption.

It should be noted that the output of the conv layers (feature map) from the parent module is the input to its child modules (solid line in Figure 3). This ensures that the operations already performed by the parent are not repeated in the child module, to further reduce redundancies. Therefore, by reusing the feature maps, the MNN-Tree acts like a DNN with a large number of layers (broken into small modules), with fewer redundant operations and high classification accuracy. This is one of the advantages of the MNN-Tree architecture when compared with Tree-CNN [66], where the original image is used as the input to every DNN. Furthermore, the MNN-Tree is a multi-level hierarchy with multiple levels of small DNN modules, for reducing the memory and energy requirement. In contrast, the Tree-CNN is not designed for low-power inference on embedded systems and the DNNs used in Tree-CNN are significantly larger (in terms of memory usage and number of operations) than the modules of the MNN-Tree. The DNNs in the Tree-CNN require 20 MB of memory for the CIFAR-10 dataset and perform close to 100M operations. The MNN-Tree requires only 0.8 MB of memory and performs 33M operations for the same dataset. The experimental setup and details are provided in Section 4.

A few challenges are associated with using the MNN-Tree. The first challenge is finding the visual similarity between the categories is crucial. We propose ASL, which detects visual similarities by using the output of a DNN to quantify the confusion between categories. The second challenges involves developing a systematic method for choosing the DNN hyper-parameters of each module and subsequently building the tree based on visual similarity to achieve high accuracy and inference speed. The third challenge is that each module of the MNN-Tree needs to be trained such that the parent is trained before its children. This ensures that the feature maps of the parent can be

Table 3. Comparison of ASL on an Untrained (U) and Trained (T) DNN for the CIFAR-10 Dataset

	In	Plane	Auto.	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
U	Cat	0.094	0.108	0.097	0.098	0.097	0.103	0.092	0.092	0.110	0.105
	Truck	0.099	0.103	0.098	0.101	0.107	0.101	0.104	0.105	0.092	0.086
T	Cat	0.011	0.026	**0.103**	**0.303**	**0.105**	**0.200**	**0.102**	**0.104**	*0.010*	0.032
	Truck	0.060	***0.193***	0.018	0.024	0.017	0.022	0.012	0.041	0.060	**0.550**

Note: This data is obtained by using the DNN model described in Section 3.3.1. The untrained DNN's outputs are similar (close to $\frac{1}{10}$). After training, the categories that are similar to the input category have high softmax outputs, such as *truck* and *auto* (0.193). The dissimilar categories, such as *cat* and *ship*, have low softmax outputs (0.010).

used as inputs to the child to eliminate redundancies. The fourth challenge is that after training, the modules of the MNN-Tree need to be used for hierarchical image classification.

3.2 Similarity Metric: Averaged Softmax Likelihood

Instead of using techniques like threshold on the distances between feature vectors, hierarchical clustering, or semantic similarities, this article proposes using the ASL method to quantify the similarity between categories. The softmax layer is the last layer of a DNN. This layer assigns a probability to each category in a classification problem. Let M be the number of categories in a dataset—for example, in the CIFAR-10 dataset, $M = 10$. Suppose A and B are two categories in the training data. Equation (1) describes how the ASL is computed: the term $\text{softmax}_A(B)$ denotes the value obtained at the output (softmax) layer of the module corresponding to category B, when the input actually belongs to category A. For example, $softmax_{cat}(dog)$ refers to the softmax output of the category *dog*, when the input image belongs to the category *cat*. $|A|$ represents the number of input samples labeled with the category A. In other words, Equation (1) is the module's average output for category B, when the inputs belong to category A. If the $L_A(B)$ is large, it means the classifier is confused about the two categories A and B.

$$L_A(B) = \frac{\sum_{i=0}^{|A|} \text{softmax}_A(B)}{|A|} \tag{1}$$

On an untrained DNN, the ASL for every category is approximately the same: $\frac{1}{M}$ ($M = 10$ for CIFAR-10) as seen in Table 3. This is because the DNN cannot distinguish between the categories. When we compute the ASL on a trained DNN, the distribution of the probabilities changes and they rise significantly close to the correct label of the input image. However, from this experiment, we also observe that after a DNN has been trained, the DNN remains confused between a few groups of categories. The softmax probabilities of certain categories after training (boldface in Table 3) is greater than the probabilities before training. For example, when the input image is a *cat*, the category *dog* has an ASL of 0.200 after training. This confusion arises due to the visual similarity between the categories. Figure 4 shows examples of images from the CIFAR-10 dataset. The categories *cat* and *dog* look similar and are visually distinct from categories like *automobile* and *truck*.

The misclassification matrix in Table 4 shows that the softmax confusion is related to the visual similarity. For a given input category, the categories that have a high ASL (boldface) are the categories to which images are frequently misclassified. For example, *truck* is most frequently misclassified as *automobile* and is rarely misclassified as other categories. This means that the categories of *truck* and *automobile* are visually similar. Thus, ASL uses the confidence of the DNN to find groups of visually similar categories without any threshold.

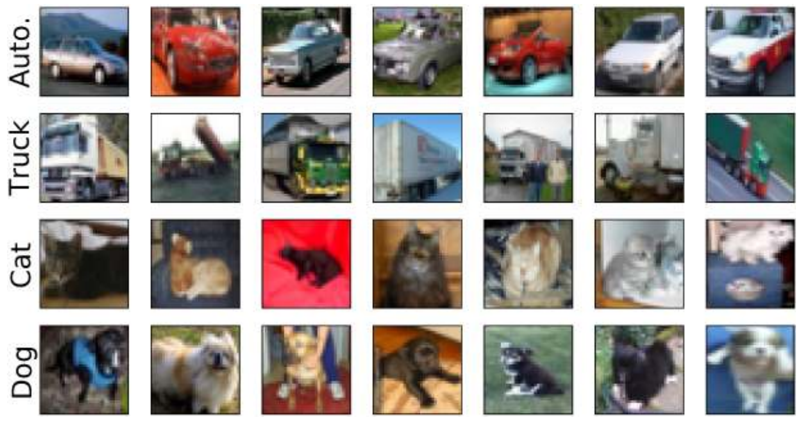


Fig. 4. Examples of images of different categories from the CIFAR-10 dataset. Our method using ASL indicates that *cat* and *dog* are visually similar.

Table 4. The DNN Confusion Matrix Shows the Number of Images Classified into Each Category from the CIFAR-10 Dataset

Input	Plane	Ship	Bird	Cat	Deer	Dog	Frog	Horse	Auto.	Truck
Plane	**360**	**75**	22	6	4	4	9	7	23	20
Ship	**48**	**370**	4	12	3	2	7	4	23	20
Bird	20	7	**248**	**42**	**46**	**30**	**32**	**32**	7	3
Cat	10	9	**31**	**204**	**31**	**98**	**43**	**30**	9	7
Deer	20	10	**43**	**37**	**267**	**29**	**35**	**44**	1	9
Dog	2	7	**32**	**96**	**28**	**256**	**27**	**30**	5	4
Frog	3	8	**35**	**46**	**25**	**27**	**354**	**24**	4	2
Horse	9	3	**30**	**27**	**39**	**36**	**28**	**319**	8	13
Auto.	22	20	6	4	5	2	4	6	**406**	**35**
Truck	23	23	3	6	3	6	8	13	**91**	**340**

Note: The categories that have been grouped together by the ASL method (boldface) are frequently misclassified into one another because they are visually similar. In this table, Auto. and Truck are examples of visually similar categories.

ASL uses Equation (1) to express the similarity between each pair of categories A and B in the dataset. The categories that have high softmax outputs are visually similar. However, making groups of visually similar categories is not a well-defined operation that can be obtained using a fixed threshold. For example, if we set the threshold to the value $\frac{1}{10}$ (softmax value obtained from untrained DNN, assuming $M = 10$), then a category with an ASL of 0.099 will not be selected. However, a category with an ASL of 0.101 will always be selected. To avoid this problem and to eliminate the need for a fixed threshold, the proposed technique uses concepts from *fuzzy sets* [69, 70]. The sigmoidal membership function is used to quantify the degree of memberships of each category: the output of the membership function gives the likelihood that a particular category exists in a super-group. The sigmoidal membership function is shown in Equation (2).

$$\sigma(x) = \frac{1}{1 + \exp(-\frac{x-\mu}{s})} \tag{2}$$

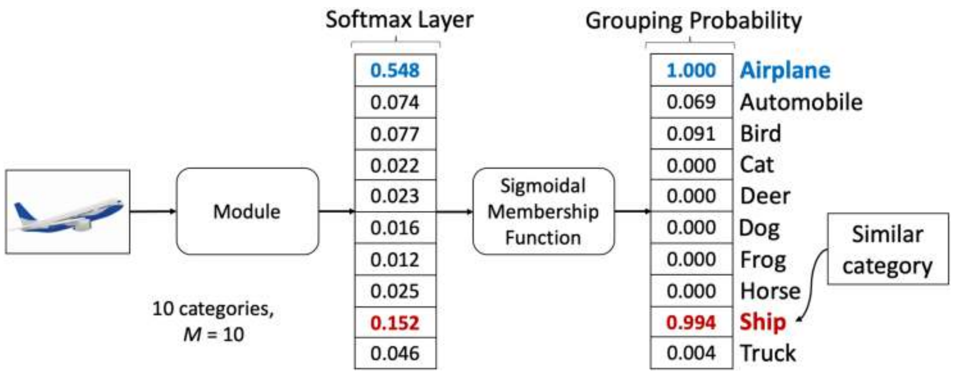


Fig. 5. The ASL for the root module of the CIFAR-10 dataset. To group categories that are visually similar to *airplane*, we use $\sigma(L_{\text{airplane}}(\text{category}))$ (Equation (3)). The sigmoidal membership function is used to avoid trial and error associated with finding a good fixed threshold.

The output of the membership function gives us the probability of grouping two categories together into a single super-group. This work uses $\mu = \frac{1}{M}$ and $s = \frac{1}{10M}$ in Equation (2) to scale the function for datasets of different sizes. This ensures that a category B with a high $L_A(B)$ is visually similar and has a high probability ($\approx$1) to be grouped with category A. Visually dissimilar categories have $L_A(B) \ll \frac{1}{M}$, and the probability of forming a super-group is very low ($\approx$0). Even though it is possible that dissimilar categories may be grouped, it is very unlikely when using the sigmoidal membership function. The use of this technique is justified by the fact that we can build super-groups of visually similar categories without the need for an optimal threshold. The probability of grouping two categories A and B, $p(A \sim B)$ is defined in Equation (3). Equation (3) is the expansion of Equations (1) and (2).

$$p(A \sim B) = \sigma(L_A(B)) = \frac{1}{1 + \exp(-10M \times (L_A(B) - \frac{1}{M}))} \tag{3}$$

Figure 5 is an example of how the ASL finds similar categories without a threshold. A high softmax output indicates that the DNN gets confused between the categories. In Figure 5, when the input image is an *airplane*, the *ship* category has a high softmax value, indicating that the categories are visually similar. The *ship* category has a high probability (0.994) of being grouped with *airplane* (given by Equation (3)). Visually dissimilar categories, such as *cat*, have a very small probability (0.000) of getting grouped with *airplane*.

3.3 Building the MNN-Tree

To build the MNN-Tree, we need to choose the appropriate DNN size for each module and then find the ASL to group categories. The MNN-Tree is constructed in a root-down fashion. The process to build a trained MNN-Tree is described in Algorithm 1. The inputs to the algorithm are the training dataset, and a hyper-parameter τ to select the size of each module (explained in Section 3.3.1). The algorithm's output is the trained MNN-Tree. Initially, we start with a single super-group that contains all categories of the dataset. In step (2) of the loop, we select the size of the DNN by using the techniques described in Section 3.3.1. The softmax output matrix, similar to Table 3 for the trained DNN, is obtained in step (4). In step (6), the ASL is used as the similarity metric to group categories (described in Section 3.3.2). If new children super-groups are formed, in step (7) the module is retrained to classify between the new super-groups. The training of the modules of the MNN-Tree is described in Section 3.4. The loop is repeated until all super-groups' modules are trained. We explain the steps of this algorithm in this section.

3.3.1 Selecting Neural Network Sizes and Hyper-Parameters. Here, we explain steps (1) through (3) of Algorithm 1. The number of layers and the size (configuration) of each module need to be chosen. We use a new metric called the *change in accuracy density* (ΔAD). It measures the accuracy gain per unit increase in the model size between two DNN models, $\Delta AD_{ij} = \frac{VA_i - VA_j}{MS_i - MS_j}$, where VA is the validation accuracy obtained during image classification, and MS is the model size of the DNNs. Here, i and j refer to the number of layers in the two DNN models. In this work, i is always $j + 1$.

Table 5 shows the effectiveness of using ΔAD as a metric. After a certain DNN depth (4 for CIFAR-10, 3 for SVHN, and 2 for EMNIST), there is no appreciable increase in accuracy when the model sizes increase. The accuracy is calculated for each module of the MNN-Tree before the children are grouped into super-groups. For example, ΔAD for the root module of an MNN-Tree is computed on accuracy obtained for classifying between all categories of the dataset, and the ΔAD for any other module in the MNN-Tree is computed while classifying between all of its children categories. By using ΔAD, we can determine the DNN configurations that are efficient and can distinguish categories with high accuracy. This approach can achieve a negligible loss in accuracy when compared with the large monolithic DNNs, because the small DNNs only need to classify among a few groups of visually similar categories and not all categories of the dataset.

ALGORITHM 1: Building the MNN-Tree

```
inputs: Training dataset $(x, y)$ $x$: images and $y$: labels; threshold $\tau$.
output: MNN-Tree structure $T$.
$CATEGORIES = \{c: c \text{ is a unique label} \in y\}$   // set of all categories
$TREE$ = {untrained root DNN}   // set holding the tree structure
$\mathbb{S} = CATEGORIES$   // all categories of the dataset come under the root
while ($\exists$ an untrained DNN in $TREE$)
    (1) $lc = 0$   // layer_count: keeps track of the number of layers in the DNN
    (2) do
            $lc = lc + 1$
            initialize a DNN $D_{lc}$, with $lc$ convolutional layers.
            train $D_{lc}$ to classify all categories of $\mathbb{S}$.
            if $lc \neq 1$ then calculate $\Delta AD_{lc,lc-1} = \frac{VA_{lc} - VA_{lc-1}}{MS_{lc} - MS_{lc-1}}$
        while ($lc = 1$ or $\Delta AD_{lc,lc-1} > \tau$)
    (3) $DNN_{config} = D_{lc-1}$   // select the DNN configuration for the module
    (4) $SOFTMAX\_MATRIX$ = softmax output of $DNN_{config} \forall c \in \mathbb{S}$   // softmax output matrix
    (5) $CHILD\_GROUPS = \phi$   // keeps track of newly formed children super-groups
    (6) for each $c \in \mathbb{S}$   //use the ASL
            find set $\mathbb{V}$, s.t. $\{\mathbb{V} \subset \mathbb{S} : \text{prob}(c \sim v) = sgmf(SOFTMAX\_MATRIX[c, v]), \forall v, c \in \mathbb{V}\}$
            add untrained DNN corresponding to $\mathbb{V}$ into $TREE$
            $CHILD\_GROUPS = CHILD\_GROUPS \cup \mathbb{V}$ // similar categories added to set of children
    (7) train $DNN_{config}$ to classify between elements of $CHILD\_GROUPS$
    (8) $\mathbb{S}$ = children of next untrained DNN in $TREE$
return tree
```

Table 5. Model Size (in Megabytes), VA, and ΔAD_{ij} Are Computed for Three Different Image Datasets with DNNs of Increasing Depth

Layer Count	1			2			3		
Dataset	Model Size	VA	ΔAD_{10}	Model Size	VA	ΔAD_{21}	Model Size	VA	ΔAD_{32}
SVHN	0.085	0.865	NA	0.106	0.893	1.333	0.174	0.969	1.118
EMNST	0.103	0.750	NA	0.161	0.845	1.638	**0.358**	**0.849**	**0.020**
CIFAR-10	0.085	0.570	NA	0.106	0.620	2.381	0.174	0.780	2.353
Layer Count	**4**			**5**			**6**		
Dataset	Model Size	VA	ΔAD_{43}	Model Size	VA	ΔAD_{54}	Model Size	VA	ΔAD_{65}
SVHN	**0.225**	**0.970**	**0.020**	0.332	0.972	0.017	0.910	0.981	0.015
EMNIST	0.561	0.852	0.015	0.766	0.854	0.010	1.350	0.856	0.003
CIFAR-10	0.225	0.895	2.255	**0.341**	**0.900**	**0.043**	0.915	0.910	0.017

Note: As the number of layers in a DNN increases, the accuracy increases, and the model size also increases. After a certain DNN depth, there is no appreciable increase in accuracy for the increase in model size. Boldface font is used to indicate the DNN configuration where ΔAD_{ij} is negligible. This work uses four layers for CIFAR-10, three layers for SVHN, and two layers for EMNIST.

It is known that increasing the depth of a DNN is more effective than increasing the width of the DNN to obtain better accuracy [71]. This is the reason Table 5 starts with only one convolutional layer and increases the number of layers by one each time. Each DNN configuration is trained until the validation accuracy (VA) saturates, then ΔAD_{ij} is computed. We use the hyper-parameter $\tau = 0.1$ on ΔAD_{ij} between two consecutive DNN models. If ΔAD_{ij} is below this value (boldface in Table 5), there is an insignificant gain in accuracy for a deeper DNN. The depth of the DNN is set to j layers. For example, in Table 5 for the EMNIST dataset, when the module has one layer, the model size is 0.103 MB, and the accuracy obtained is 75%. With two layers, the model size increases to 0.161 MB, with an accuracy of 84.5%. Thus, $\Delta AD_{21} = \frac{0.845-0.750}{0.161-0.103} = 1.638$. When three layers are used, $\Delta AD_{32} = \frac{0.849-0.845}{0.358-0.161} = 0.02$. Since $\Delta AD_{32} < \tau = 0.1$, the accuracy increase is marginal for the increase in the model size. Hence, for the EMNIST dataset, we use a DNN with two layers at the root module.

The value τ is a hyper-parameter that needs to be selected before constructing the MNN-Tree. A large τ leads to tall MNN-Trees that contain multiple small DNN modules (each module has few layers). Tall hierarchies have higher latency and require more energy because multiple levels of modules need to be loaded into memory. A small τ leads to short-wide hierarchies with large DNNs. The DNNs in such hierarchies resemble the monolithic DNNs and contain many redundancies. It is important to find an intermediate value of τ to avoid these drawbacks. Table 6 evaluates the tradeoff for selecting the value of τ. When $\tau = 5$ (a large value), the size of each DNN module is restricted (one layer in each module). This is because a deeper DNN is chosen only when a significant increase in accuracy is possible. Small modules cannot distinguish between categories effectively, thus leading to the formation of a small number of super-groups under each parent (only two super-groups under each parent). Each module performs few operations and thus consumes little energy (0.358 J). However, because many modules are loaded into memory and used (height of the MNN-Tree = 6 when $\tau = 5$), the entire MNN-Tree consumes a significant amount of energy (2.152 J). The methods to measure the energy requirement and classification accuracy are described in Section 4.2. Moreover, tall trees have high test error (0.231). However, a smaller value of τ results in a short and wide MNN-Tree, where each module consists of many convolutional layers (~9 layers in each DNN module when $\tau = 0.001$). Such modules resemble the monolithic DNN

Table 6. Tradeoff Between Module Size, Energy Consumption, and Test Error

τ	Average No. of Layers	Average No. of Children per Module	Average Energy Consumption per Module (J)	Height of MNN-Tree	Total Energy Consumption of MNN-Tree (J)	Test Error
0.001	8.50	5.0	1.584	2	3.168	0.067
0.100	3.33	2.5	0.511	3	1.533	0.079
5.000	1.00	2.0	0.358	6	2.152	0.231

Note: When τ is small, each module is large and consumes more energy, but the tree obtains low test error. When τ is large, the modules are small and test error is high. Because many levels of modules are required, the energy consumption is also high. The methods to measure the energy requirement and test error are described in Section 4.2.

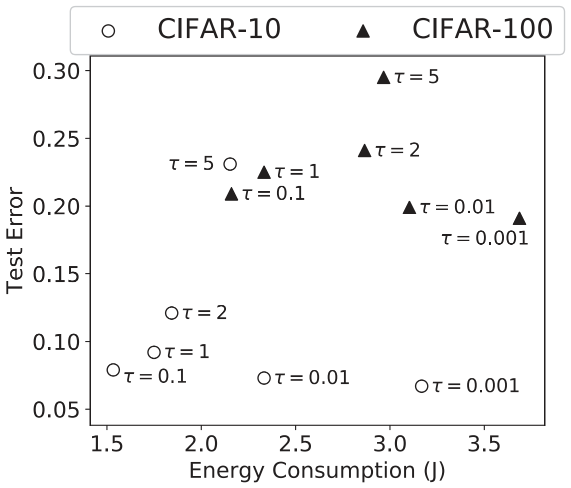


Fig. 6. Plot of energy consumption versus test error obtained from MNN-Trees built with different values of τ for the CIFAR-10 and CIFAR-100 datasets. Large values of τ result in tall MNN-Trees that consume more energy and have a high error. A small τ results in a short-wide MNN-Tree where each module has several convolutional layers—consuming more energy but obtaining high accuracy.

architectures, and the corresponding MNN-Tree consumes more energy (3.168 J) but achieves low test error (0.067). Figure 6 plots the test error and energy consumption of the MNN-Tree with different values of τ (5, 2, 1, 0.1, 0.01, and 0.001) for the CIFAR-10 and CIFAR-100 datasets. For small values of τ (0.01 and 0.001), we get high energy consumption and low test error. For intermediate values of τ (~0.1), we get significantly lower energy consumption, with a marginal increase in test error. As τ continues to increase, the energy consumption begins to increase and test error increases. This empirical evidence tells us that $\tau = 0.1$ is a good selection for low-power and high accuracy to construct MNN-Trees.

The size of the DNN is calculated for every module of the MNN-Tree, and hence different modules may have different DNN sizes. Table 7 shows the DNN architecture obtained for each module of the MNN-Tree of the SVHN dataset. These architectures are obtained by using the ΔAD metric and $\tau = 0.1$. The input images of size $32 \times 32 \times 3$ (32 pixels tall and wide, three color channels corresponding to R, G, and B) are fed into the root module. Here, three conv layers and one max pooling layer are used to build the feature map. The selection between the children is made at the end of the FC layer (there are six children: two super-groups and four categories). Depending on the output of the root module, the feature map (of size $16 \times 16 \times 32$) is used as an input to one

Table 7. Architecture Used in Each Module of the MNN-Tree for the SVHN Dataset

Root Module		Child Module 1		Child Module 2	
Layers	Output Size	Layers	Output Size	Layers	Output Size
Input	32 × 32 × 3	Input	16 × 16 × 32	Input	16 × 16 × 32
Conv1 3 × 3 × 16 Stride = 1 Padding = 1 BatchNorm ReLu	32 × 32 × 16	Conv1 3 × 3 × 32 Stride = 1 Padding = 1 BatchNorm ReLu	16 × 16 × 32	Conv1 3 × 3 × 32 Stride = 1 Padding = 1 BatchNorm ReLu	16 × 16 × 32
Conv2 3 × 3 × 32 Stride = 1 Padding = 1 BatchNorm ReLu	32 × 32 × 32	Conv2 3 × 3 × 64 Stride = 1 Padding = 1 BatchNorm ReLu	16 × 16 × 64	Conv2 3 × 3 × 32 Stride = 1 Padding = 1 BatchNorm ReLu	16 × 16 × 32
Conv3 3 × 3 × 32 Stride = 1 Padding = 1 BatchNorm ReLu	32 × 32 × 32	Max Pooling 2 × 2 Stride = 2	8 × 8 × 64	Max Pooling 2 × 2 Stride = 2	8 × 8 × 32
Max Pooling 2 × 2 Stride = 2	16 × 16 × 32	FC	4096 × 4	FC	2048 × 2
FC	8192 × 6				

Note: The output feature map ofthe last conv layer from the root module is used as inputs to the child modules.

of the child modules. Since each DNN is very small, most of the hyper-parameters (kernel size, number of channels, layer width, batch normalization, weight-decay, etc.) can be the same and do not require significant fine tuning for high accuracy.

It should be kept in mind that the output feature map from the parent module is used as inputs to the child module. This ensures that the operations performed by the parent are not repeated in the child. Furthermore, by reusing the feature map, the child acts as a specialized extension of the parent. This makes each path of the MNN-Tree act like a different DNN with several layers. This is the reason the modules lower in the MNN-Tree also have small DNNs even though they are classifying between visually similar categories.

3.3.2 Identifying Super-Groups and Building Hierarchy. In this part, we explain steps (4) through (6) of Algorithm 1. The root of the hierarchy is created first. After the size of the DNN module is selected, we use the ASL to compute the similarity between all categories of the dataset to obtain the first level of super-groups (children of the root module). For each child of the root that is a super-group, the process is repeated: first the module's DNN size is selected (Section 3.3.1), then the ASL is used to find the similarities between the categories and build the super-groups, and finally the module is trained to classify between the newly formed children super-groups (Section 3.4). This process is depicted in Figure 7 and is performed for each child module until the original categories of the dataset can be classified without forming a super-group (i.e., individual

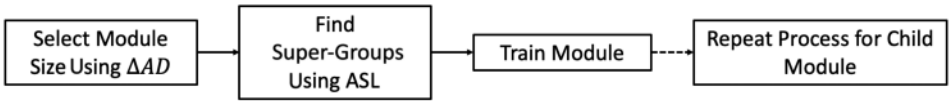

Select Module Size Using ΔAD
Find Super-Groups Using ASL
Train Module
Repeat Process for Child Module

Fig. 7. The method to build the MNN-Tree. First, the module's DNN size is selected, then using the ASL, the super-groups are found. The module is trained to classify between its children super-groups. The dotted line shows us that the process is repeated for the next module.

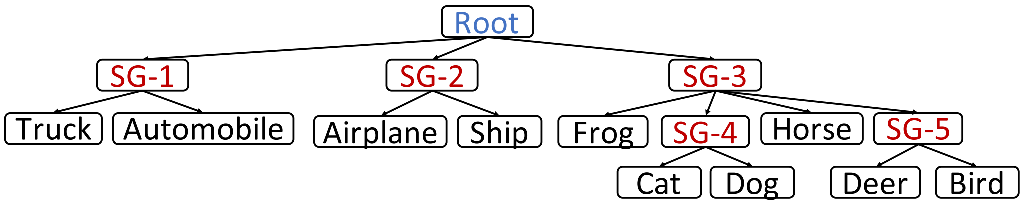


Fig. 8. The MNN-Tree obtained for the CIFAR-10 dataset. SG refers to a super-group.

categories are all leaf modules). This technique is used to create the MNN-Tree seen in Figure 8 for CIFAR-10. The tree follows intuitive ideas of visual similarity. The categories *dog* and *cat* are similar, and they are less similar to *horse* and very dissimilar to *ship*. It is worth noting that in the CIFAR-10 dataset, the category *automobile* refers to cars and SUVs, and does not include trucks or other types of vehicles.

Furthermore, it should be noted that the number of children under each parent varies because the proposed method groups visually similar categories dynamically. This is an improvement on existing techniques that are limited to grouping a fixed number of categories at each level of the hierarchy.

3.4 Training the MNN-Tree

In this section, we explain step (7) of Algorithm 1. The conventional back-propagation algorithm is used to train each module of the MNN-Tree. The root module is trained first. The output feature maps of the images that have been classified correctly by a parent module are used to train its child modules. This prevents modules lower in the MNN-Tree from being trained with features not pertinent to them and their corresponding sub-trees. Furthermore, it also ensures that the training error in an ancestor module does not affect the training of its descendants. We consider two methods to train the MNN-Tree. Kontschieder et al. [72] compute the loss at the leaf nodes and use back-propagation through all nodes of the tree to minimize the loss. Roy et al. [73] use back-propagation in each module, individually. We use the latter technique because the modules at a given depth (sibling modules) are independent of one another. This allows us to train modules in parallel to reduce the training time. Small DNNs have fewer parameters and are less likely to overfit and can get high accuracy with little hyper-parameter tuning. The computational cost of training small DNNs is lower because small DNNs have considerably fewer parameters. This ensures that the building and training of the MNN-Tree can be completed quickly.

3.5 Image Classification with the MNN-Tree

Figure 9 describes how image classification is performed with the MNN-Tree. The root is the first module to process the image and select one of its children. The feature map of the root is used as the input to the selected child module, and the process is repeated until a leaf of the MNN-Tree is reached. The leaves of the MNN-Tree are the original categories of the dataset. Every module

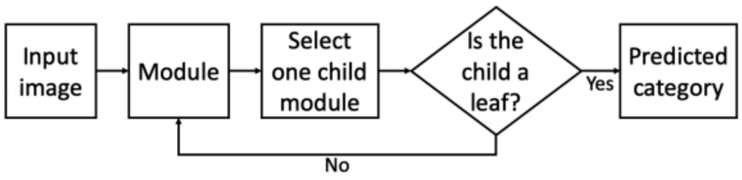


Fig. 9. Image classification with the MNN-Tree. First, the input image is processed by the root module. The image is processed by every module along a path from the root to a leaf.

selects only one child, and thus a significant portion of the MNN-Tree is pruned at every step, leading to a considerable reduction in the number of operations.

In Figure 8, the root module classifies among the super-groups *SG-1*, *SG-2*, and *SG-3*. Once a classification is made, the partially processed data (output feature map of the root) is then passed onto one of the children. If the *SG-3* super-group is chosen, the module responsible for classifying between *SG-4*, *SG-5*, *frog*, and *horse* is used. This process is repeated at every level of the tree until a leaf module is reached.

By using the output feature map of the parent module as the input to the child module, the MNN-Tree takes advantages of deep architectures for better accuracy. When using the MNN-Tree, at a given time, only a single module is loaded in memory. After the module is used, the memory is freed so that it can be used by its child module. This reduces the total memory required from the embedded device to run the MNN-Tree. Moreover, because only a small subset of the modules are used during inference, the total number of bytes loaded into memory is significantly reduced when compared with existing DNN architectures [7, 13, 74]. Details of the experiments are presented in the next section.

4 EXPERIMENTS AND RESULTS

The datasets used in the experiments are described in Section 4.1. The experimental setup is explained in Section 4.2. Section 4.3 compares the accuracy, memory requirement, and number of operations of the MNN-Tree and existing monolithic DNN architectures. The MNN-Tree is compared with the existing state-of-the-art hierarchical image classifiers in Section 4.4. We also explain the reasons the existing methods fail to achieve high accuracy and experimentally show that the MNN-Tree is superior. Section 4.5 compares the inference time and energy consumption of different architectures on embedded devices. Section 4.6 discusses potential extensions and future work.

4.1 Datasets Used

We use several image datasets in our experiments. CIFAR [75], SVHN [76], and EMNIST [77] contain centered and fixed-size images, with only one object in each image. Images in larger datasets like ImageNet 2012 [78] and Caltech-256 [79] are of different sizes and represent real-life images more closely. Examples of images from these datasets are provided in Figure 10.

The CIFAR datasets consist of 32×32 color images of 10 (CIFAR-10) and 100 (CIFAR-100) different categories. The training and test sets contain 50,000 and 10,000 images, respectively. We follow the common practice of using 5,000 images from the training set to form the validation set. The SVHN dataset contains 73,257 colored images of size 32×32 pixels in the training set and 531,131 images for additional training. While reporting the results of the SVHN dataset, we follow the common practice of using all training data without any data augmentation. A set with 6,000 images is used to verify the training results. EMNIST is an extension of the popular MNIST dataset. There are six configurations of the EMNIST dataset, and we use the EMNIST-Balanced configuration. It contains 131,600 grayscale images of size 28×28 pixels belonging to 47 categories. The

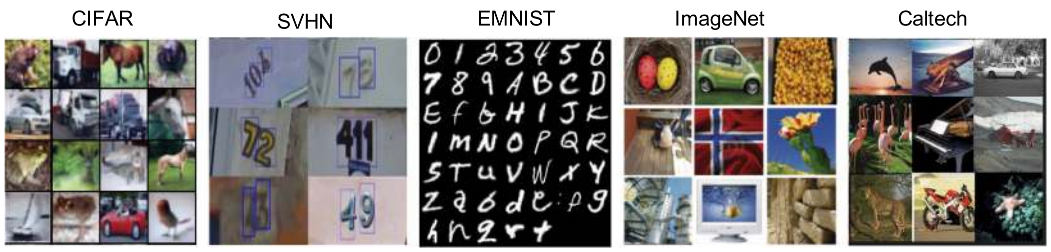


Fig. 10. Examples of images from the different datasets used in our experiments. It can be seen that images from different datasets vary significantly. Some images have background (e.g., grass and sky) in addition to the foreground objects.

Table 8. Details of the Datasets Used in the Experiments

Dataset	Image Size	No. of Training Images	No. of Test Images	No. of Categories
CIFAR 10	$32 \times 32 \times 3$	50,000	10,000	10
CIFAR 100	$32 \times 32 \times 3$	50,000	10,000	100
SVHN	$32 \times 32 \times 3$	604,388	26,032	10
EMNIST	$28 \times 28 \times 1$	112,800	18,800	47
ImageNet 2012	Varying	1,200,000	75,000	1,000
ImageNet 2012 (subset)	Varying	26,000	2,000	20
Caltech (subset)	Varying	2,000	400	11

Note: The output feature map of the last conv layer from the root module is used as inputs to the child modules.

ImageNet training set contains 1,000 categories with approximately 1,000 images each. ImageNet also contains a validation set and a testing set. We report the ImageNet top-1 accuracy in our experiments when trained without any data augmentation. We also use a subset of the ImageNet dataset with 20 categories to easily visualize the MNN-Tree and fully understand the hierarchy's details and properties. The Caltech dataset is used in our experiments as well. As suggested in Panda et al. [50], we use a subset of 11 categories from the Caltech dataset to maintain a fair comparison with existing work. Each category in the Caltech dataset has approximately 100 training images and 20 testing images. The datasets' parameters are described in Table 8.

4.2 Experimental Setup

Figure 11 is a flowchart explaining the method used to conduct the experiments. For each dataset, once the MNN-Tree is built and trained, the test set of the dataset is used to find the classification accuracy. The test set is used to avoid the effects of overfitting. The memory requirement and number of operations are found using the `torchsummary` and `thop` PyTorch libraries, respectively. A Yokogawa WT310E Power Meter [80] is used to measure the energy consumption for running the different DNN architectures on the embedded systems. The images for image classification are stored locally on the Raspberry Pi, and the PyTorch DataLoader [81] is set up before energy consumption is measured. The inference time and energy consumption are measured during image classification. Measurement is stopped after the classification of 500 images is complete. To monitor the effects of thermal throttling between measurements, we use the `vcgencmd` command for obtaining the temperature of the device. We only run the next experiment after the device has stopped thermal throttling its CPU. The output of the `vcgencmd get_throttled` command is used to identify if the Raspberry Pi is throttled.

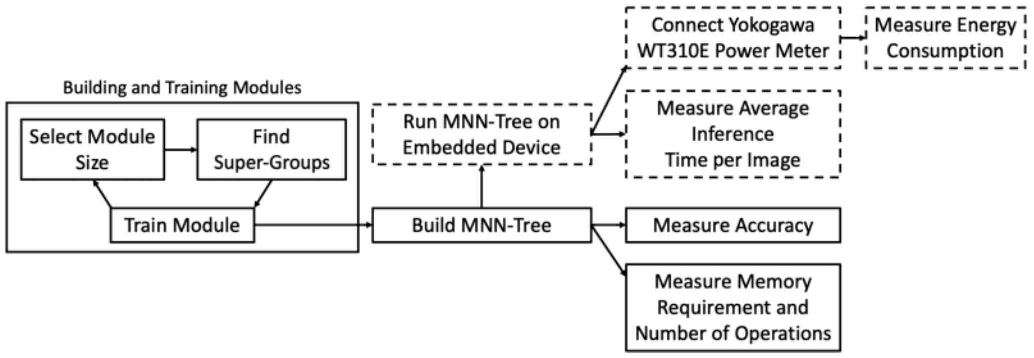

Building and Training Modules
Select Module Size
Find Super-Groups
Train Module
Build MNN-Tree
Run MNN-Tree on Embedded Device
Connect Yokogawa WT310E Power Meter
Measure Energy Consumption
Measure Average Inference Time per Image
Measure Accuracy
Measure Memory Requirement and Number of Operations

Fig. 11. Workflow of the method used to conduct experiments. First, the DNN size is selected, super-groups are found, and then the DNN is trained, for each module to build the MNN-Tree. The memory requirement of the modules and number of operations is measured. The MNN-Tree is then run on the embedded devices to measure energy consumption. The boxes in dashed lines are implemented on the embedded devices. The other processes are implemented on a GPU desktop machine.

On embedded systems, most image classification applications perform inference on one image at a time [2, 11, 23, 24, 82, 83]. Even when processing videos, inference can be performed on individual frames each time [84]. The MNN-Tree is designed specifically for deployment on embedded devices (e.g., Raspberry Pi). Such devices are not suitable for massive batch processing applications with high throughput requirements because embedded devices have limited memory and computing resources. Consequently, this work only compares the inference time when the batch size is 1.

All modules of the MNN-Tree are trained using the ADAM [85] learning rule. A batch size of 200 is used for all of the datasets with 150 epochs. An initial learning rate of 0.01 is used and is dropped by a factor of 10 at 50% and 75% of the total number of epoch. Since each module contains a small DNN, the number of parameters is considerably smaller than large DNNs, thus avoiding overfitting. Small DNNs do not require significant hyper-parameter tuning to achieve high accuracy.

The MNN-Tree for the ImageNet dataset takes the longest to train because it has the largest number of modules. Training time for the MNN-Tree architecture includes the time taken to build the MNN-Tree. When using an Nvidia TITAN X (Pascal) GPU, the total training time is approximately 29 hours for the ImageNet dataset. This training time is comparable to the training time required by the monolithic DNNs for the ImageNet dataset.

The examples, source code, and DNN models are available on GitHub [15]. The *ClassicalMachineLearningTechniques* folder in the GitHub repository includes source code for the different machine learning techniques (without DNNs) that are used for image classification (seen in Table 1). *GaborHSVSimilarity*, *ThresholdMethods*, and *HierarchicalClustering* folders contain analyses of different existing hierarchical techniques. The *MonolithicDNNs* folder contains implementation of different monolithic DNNs used in our experiments. The *MNNTree* folder contains training scripts, testing scripts, examples, and implementation of the ASL for different datasets. A video [86] capturing the experimental setup and energy consumption measurement has been uploaded to YouTube.

4.3 Comparison with Monolithic DNN Architectures

We compare the MNN-Tree architecture with several existing architectures. Table 9 compares the MNN-Tree with existing monolithic DNNs on various metrics. The MNN-Tree has the least memory requirement and number of operations. These performance gains come at a negligible

Table 9. Comparison of Memory Requirement, Number of Operations, and Test Error for Different Datasets and Techniques

Dataset	Technique	Model Size (KB)	No. of Operations	Test Error
CIFAR-10	VGG-16 [13]	78,410	313 M	0.067
	VGG-Pruned [27]	28,200	206 M	0.066
	DenseNet-190 [74]	102,000	9,388 M	0.070
	CondenseNet-160 [22]	43,000	1,084 M	0.034
	MobileNet V2 [31]	8,800	100 M	0.060
	MNN-Tree$^+$	806	28 M	0.079
CIFAR-100	VGG-16 [13]	78,590	207 M	0.295
	VGG-Pruned [27]	28,910	210 M	0.252
	DenseNet-190 [74]	103,000	9,400 M	0.171
	CondenseNet-160 [22]	44,500	1,080 M	0.184
	Wide ResNet-28,10 [87]	141,100	25,800 M	0.192
	MNN-Tree$^+$	832	34 M	0.209
SVHN	DenseNet-190 [74]	102,000	9,388 M	0.017
	Wide ResNet-16,4 [87]	11,000	1,935 M	0.016
	MNN-Tree$^+$	522	30 M	0.018
EMNIST	EDEN [88]	—	—	0.117
	MNN-Tree$^+$	363	4 M	0.078
ImageNet 2012 (subset)	VGG-16$^+$ [13]	528,000	15,300 M	0.076
	ResNet-34$^+$ [7]	84,000	3,640 M	0.081
	DenseNet-121$^+$ [74]	32,300	3,000 M	0.080
	SqueezeNet$^+$ [30]	5,120	837 M	0.146
	MobileNet v2$^+$ [31]	8,820	585 M	0.104
	MNN-Tree$^+$	1,872	605 M	0.124
ImageNet 2012	VGG-16 [13]	528,120	15,300 M	0.295
	ResNet-34 [7]	84,100	3,640 M	0.276
	DenseNet-121 [74]	32,400	3,000 M	0.230
	SqueezeNet [30]	5,330	837 M	0.425
	MobileNet v2 [31]	8,910	585 M	0.280
	MNN-Tree$^+$	2,515	713 M	0.313

Note: EDEN does not report the model size or the number of operations and hence is represented as a dash (—). We also use a subset of ImageNet to better analyze the different properties of the obtained MNN-Tree. The plus sign (+) refers to the techniques whose results are obtained by our experiments. The other results are obtained from the respective publications.

cost to the test error. The ResNet [7] and VGG [13] DNNs contain 54 and 16 layers, respectively. We also compare with the pruned and quantized version of VGG architecture: VGG-Pruned [27]. DenseNet [74] and CondenseNet-160 [22] use group convolutions to improve the parameter efficiency. Huang et al. [74] suggest that DenseNet-190 be used for CIFAR and SVHN datasets, and DenseNet-121 be used for ImageNet. We follow these suggestions in our experiments. We also compare the MNN-Tree with Wide ResNet-28,10 and Wide ResNet-16,4 [87]. Wide ResNet-x, y corresponds to an architecture with x layers and a growth rate of y. The growth rate is a hyper-parameter that determines the size of each layer in Wide ResNet. For the EMNIST dataset, we use EDEN [88] for comparisons. We also compare with SqueezeNet [30] and MobileNet [31].

Table 10. Comparison of the Test Error of MNN-Tree with Other Hierarchical Image Classification Techniques

Technique		Test Error		
		CIFAR-10	Caltech (subset)	ImageNet 2012
Hierarchical Clustering	Griffin et al. [45]	—	0.180	—
	Chen et al. [55]	—	0.174	—
	Marszalek et al. [56]	—	0.185	—
	Lukic et al. [57]	0.231	0.199	—
Gabor and HSV Filters	Panda et al. [50]	0.207	0.212	—
Semantic Similarity	Qu et al. [60]	—	—	0.452
Proposed Method	**MNN-Tree**	**0.079**	**0.111**	**0.313**

Note: A dash (—) indicates that data and/or source code is not available for the dataset. The MNN-Tree achieves the best accuracy.

These architectures contain large DNNs with inverted bottleneck filters to reduce the number of operations.

In Table 9, we see that the MNN-Tree has the smallest model size. When compared with VGG-Pruned on CIFAR-100, the MNN-Tree requires a model 97.12% ($1 - \frac{832}{28910} = 0.9712$) smaller. Similarly, when comparing the MNN-Tree with SqueezeNet on ImageNet, we observe a 52.81% ($1 - \frac{2515}{5330} = 0.5281$) reduction in the size. Smaller models require fewer memory accesses, achieve faster inference, and reduce energy consumption. The table shows the number of floating-point multiplications and additions performed during the inference of a single image. The reported model size and number of operations of the MNN-Tree is the sum of the values of the modules along the longest path from the root to a leaf. The number of operations for the MNN-Tree is 99.70% ($1 - \frac{28}{9388} = 0.9970$) lower than DenseNet and 97.41% ($1 - \frac{28}{1084} = 0.9741$) lower than CondenseNet for the CIFAR-10 dataset. There is a negligible difference in memory requirement and number of operations when comparing the MNN-Tree for ImageNet 2012 (subset) and the entire ImageNet 2012 dataset. This shows the scalability of the proposed technique when constructed using the methods presented in Section 3.3. The table does not report the model size and number of operations for EDEN because the data and source code is not openly available.

From Table 9, it can be seen that the MNN-Tree achieves the lowest error of 7.8% for the EMNIST dataset. The MNN-Tree's accuracy is comparable to the state of the art for CIFAR-10 and SVHN datasets. The MNN-Tree also outperforms SqueezeNet on the ImageNet dataset in top-1 classification accuracy. For CIFAR-100, the benchmark test error is 17.1%, and the MNN-Tree achieves 20.9%. It is worth noting that the test error achieved in the state-of-the-art monolithic DNN architectures is obtained after spending significant effort in performing hyper-parameter tuning.

4.4 Comparison with Hierarchical Image Classification Architectures

We compare the MNN-Tree with existing hierarchical image classification techniques. Griffin et al. [45] use hierarchical spectral clustering to create a binary tree of SVM classifiers. Hierarchical K-means is used in Chen at al. [55] and Marszalek et al. [56] to group SIFT feature vectors from different images to create a tree of classifiers. Lukic et al. [57] describe methods to use feature vectors from intermediate layers of DNNs to perform hierarchical clustering. Panda et al. [50] use Gabor and HSV filters to identify similar colors and textures among images to build groups of categories. The *WordNet Tree* described by Qu et al. [60] is the tree of categories that is built using the semantic information from WordNet.

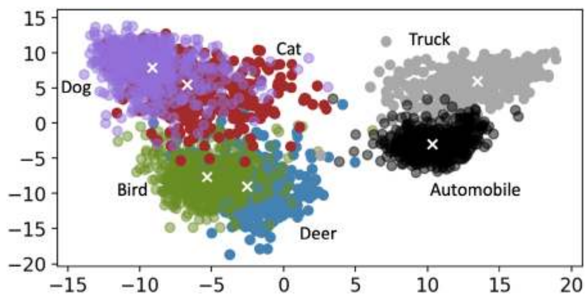


Fig. 12. Principal component analysis of the feature vectors obtained from VGG-16 trained on the CIFAR-10 dataset, along the two axes of maximum variance. The "×" marks the centroids of the clusters of the six different categories. It can be seen that some categories are closer than other categories. The distances between the centroids of the categories is shown later in Table 11.

Table 11. Euclidean Distances Between the Centroids of Categories of the Principal Component Analysis of the Feature Vectors Obtained from VGG-16 Trained on the CIFAR-10 Dataset

Categ.	Auto.	Truck	Cat	Dog	Bird	Deer
Auto.	0.00	2.29	17.23	14.70	15.80	14.70
Truck	2.29	0.00	19.40	20.45	17.35	15.86
Cat	17.23	19.40	0.00	1.73	8.82	9.55
Dog	18.34	20.45	1.73	0.00	8.15	9.09
Bird	15.80	17.35	8.82	8.15	0.00	3.59
Deer	14.70	15.86	9.55	9.09	3.59	0.00

Note: Some pairs of categories have a smaller distance than other pairs. It is difficult to select a threshold on the distances between categories to find groups of similar categories. Small changes in the threshold can lead to significantly different groups.

Table 10 shows that the proposed method achieves the best accuracy when compared with different hierarchical techniques for three popular image datasets: CIFAR-10, Caltech, and ImageNet. The test error obtained by the MNN-Tree on the Caltech dataset (11.1%) is significantly lower than other techniques. On the CIFAR-10 dataset, the MNN-Tree achieves an error of 7.9%, whereas the technique proposed by Panda et al. [50] achieves an error of 20.7%. Because the source code is not readily available, it is not possible to make a fair comparison in terms of energy consumption, memory requirement, or latency on the Raspberry Pi.

The remainder of this section explains why the MNN-Tree outperforms the other techniques using hierarchical image classification. Section 4.4.1 explains why threshold methods are not used in hierarchical image classifiers. Section 4.4.2 compares the MNN-Tree with techniques using Gabor and HSV filters. The MNN-Tree is compared with hierarchical clustering techniques in Section 4.4.3 and with trees based on semantic similarities in Section 4.4.4.

4.4.1 Comparison with Threshold Methods. Figure 12 is a principal component analysis of the feature vectors obtained for the CIFAR-10 dataset from a pre-trained VGG-16 DNN. Some categories like *cat* and *dog* have a small distance between them, whereas categories like *dog* and *truck* have a large distance between them. The Euclidean distances between the centroids of the different categories are tabulated in Table 11 for the root of the CIFAR-10 tree. It can be seen that *cat* and *dog* have a distance of 1.73, and *dog* and *truck* have a distance of 20.45. The threshold-based methods use a manually selected threshold on these distances to group similar categories. For example, if

Table 12. Analysis of the Euclidean Distances Between Centroids of Different Categories for Different Datasets

Dataset	Avg. Distance Between Clusters	Min Distance Between Clusters	Max Distance Between Clusters
CIFAR-10	15.59	12.41	21.38
CIFAR-100	15.77	6.12	21.74
SVHN	22.04	15.45	28.00
EMNIST	0.85	0.31	1.16

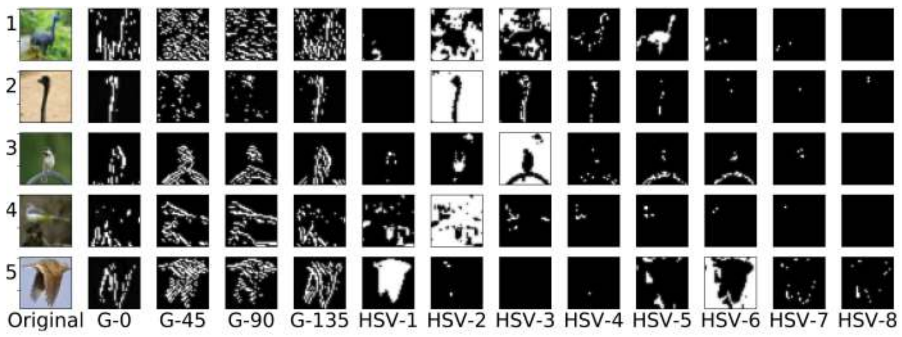
Note: The distances for different datasets vary significantly. Thus, it is not possible to select a single threshold for different datasets.

the threshold = 5, then there are three groups formed: *cat* and *dog*, *bird* and *deer*, and *automobile* and *truck*. If the threshold = 10, then there are two groups: *cat*, *dog*, *bird* and *deer*, and *automobile* and *truck*. Furthermore, if the threshold = 10 at the root, then a new threshold must be selected to divide *cat*, *dog*, *bird*, and *deer* into groups of similar categories at the child node. In other words, a new threshold needs to be selected for every node in the tree. This makes it difficult to build a hierarchy of classifiers based on thresholds. The problems with threshold-based methods are more pronounced in datasets with a large number of categories (e.g., ImageNet) because it is difficult to manually inspect the distances between all pairs of categories. The technique proposed in this article uses the process described in Section 3.2 to find visually similar categories. The proposed technique has the advantage of only requiring the DNN module's softmax output and the number of categories of the dataset to group categories in every node of the tree. No manual tuning of thresholds is required.

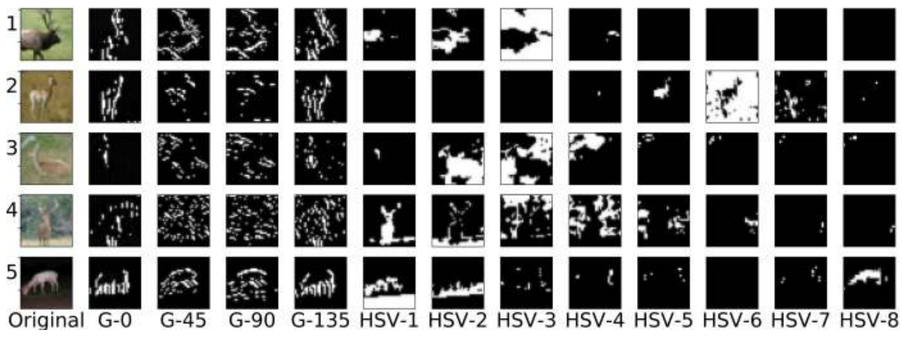
Table 12 analyzes the distances between the centroids of the categories for different datasets. For each category, the centroid of the feature vectors is obtained from a pre-trained VGG-16 DNN [41]. The feature vectors for all datasets are of the same dimension, and therefore it is possible to compare the distances between feature vectors across datasets. The average distance between the categories in the EMNIST and SVHN datasets are 0.85 and 22.04, respectively. This means that the different categories of the EMNIST dataset are more similar to each other (have more common visual characteristics) than the different categories of the SVHN dataset. This is intuitive because all images of EMNIST are normalized with a black background, whereas the images in the SVHN dataset are of different colors with varied backgrounds (see Figure 10 for reference). Because the distances between categories for different datasets vary significantly, the evidence indicates that we need different thresholds to build the MNN-Tree for different datasets. The threshold needs to be selected carefully for each dataset.

The proposed method dynamically builds super-groups without the need of a varying threshold. This is achieved because we use ASL, a new similarity metric, to group similar categories. As described in Section 3.2, ASL uses the output of the softmax layer to quantify the visual similarity between different categories. A sigmoidal membership function probabilistically determines if two categories belong to the same super-group (described in Equations (2) and (3)). This systematic method is used for all modules in the MNN-Tree and builds hierarchies without any manual tuning or thresholds. Furthermore, the selection of the hyper-parameter τ (used to select the size of the modules of the MNN-Tree) is obtained through experiments and does not need to be tuned for each individual module/dataset.

4.4.2 Comparison with Gabor and HSV Filters. Gabor and HSV filters are inconsistent for finding the similarities between categories in datasets. This is because different images belonging to a single category often have different colors and textures, meaning that it is difficult to find a fil-


1
2
3
4
5
Original G-0 G-45 G-90 G-135 HSV-1 HSV-2 HSV-3 HSV-4 HSV-5 HSV-6 HSV-7 HSV-8

(a) Bird Category from CIFAR-10


1
2
3
4
5
Original G-0 G-45 G-90 G-135 HSV-1 HSV-2 HSV-3 HSV-4 HSV-5 HSV-6 HSV-7 HSV-8

(b) Deer Category from CIFAR-10

Fig. 13. Images from the CIFAR-10 dataset belonging to categories *Bird* and *Deer* after Gabor and HSV transformations. There is no single texture or color that can represent every image of the category. G-0, G-45, G-90, and G-135 correspond to the feature maps obtained from Gabor filters oriented at 0°, 45°, 90°, and 135°, respectively. HSV-1 to HSV-8 correspond to the activations of eight HSV color components.

ter (single color or texture) that represents each category uniquely. Figure 13 shows examples of images of birds and deer from the CIFAR-10 dataset with their corresponding feature activations from Gabor filters G-0, G-45, G-90, and G-135 (for Gabor filters oriented at 0°, 45°, 90°, and 135°) and HSV color features HSV-1 to HSV-8 (for the eight HSV color components). Each feature activation detects and highlights (white regions) the presence of textures and colors in the input image. Certain feature activations represent the object of interest in the image accurately. For example, the bird in the image in row 1 of Figure 13(a) is accurately represented by HSV-5. The same bird does not contain features corresponding to the HSV-1 filter, and thus the corresponding feature activation does not effectively represent the bird.

As seen in Figure 13(a), no single filter uniformly represents all birds in the different images of the category. The bird in the image in row 1 is effectively represented by HSV-5, but no other bird (in the other images of the category) is activated by the filter HSV-5. Similarly, the birds in the images of row 2 and row 5 are the only ones to have a meaningful activation from G-0 and HSV-1, respectively. Different birds in the category contain different colors and textures. Hence, Gabor and HSV filter activations do not make a good similarity metric. A similar observation is seen in Figure 13(b) for the images belonging to the *Deer* category. This is the reason the hierarchical image classification technique described by Panda et al. [50] leads to poor classification accuracy on most datasets as seen in Table 10.

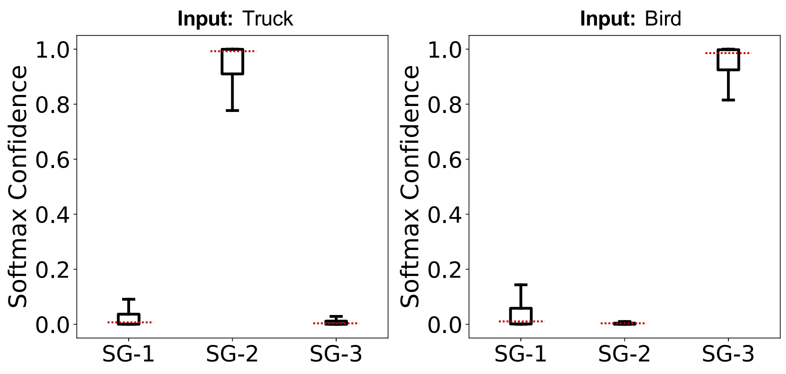


Fig. 14. Box plots of the softmax confidence of the root module of the CIFAR-10 MNN-Tree shown in Figure 8. Images are consistently classified into the correct super-group with high confidence. For reference, SG-1 contains *airplane* and *ship*, SG-2 contains *truck* and *automobile*, and SG-3 contains *dog*, *cat*, *bird*, *frog*, and *horse*. SG stands for super-group.

We conduct experiments to show that the MNN-Tree does not suffer from the problem seen in Figure 13. The proposed similarity metric works even when images of the same category have varying colors and textures. The box plots in Figure 14 represent the softmax confidence of the three super-groups at the root module of the CIFAR-10 MNN-Tree. The box plots are used to display the distribution of the softmax values obtained for each super-group (for all images belonging to the input category). Each box plot is based on a five-number summary: minimum (lower whisker), first quartile (bottom of the box), median (red dotted line), third quartile (top of the box), and maximum (upper whisker). The softmax confidence measures the certainty with which the images are classified into each super-group. The box plots show that the images belonging to the *bird* category are consistently classified into SG-3 (the correct super-group) with high confidence. The *bird* images have a low softmax confidence for the other two super-groups. A similar observation is seen for the *truck* category. The super-groups in the MNN-Tree closely represent the visual similarity and is the reason the MNN-Tree obtains high classification accuracy.

4.4.3 Comparison with Hierarchical Clustering. This article also compares the MNN-Tree with techniques that use hierarchical clustering [45, 55–57] to find similar categories. Hierarchical clustering techniques group a fixed number of categories in every level of the tree. Prior work has shown that these techniques obtain low classification accuracy because groups of similar categories cannot be formed with a fixed partition [50]. We experimentally show that hierarchical clustering generally results in degenerate hierarchies: a tree where each non-leaf node (super-group) has only one non-leaf child (another super-group) and one or more leaf children. Figure 15(a) depicts the tree obtained with hierarchical clustering when the two closest categories are grouped at every step [57]. The distance between the categories is the distance between the centroids of the feature vectors of the categories. The resulting tree is degenerated. Degenerate trees need many modules (i.e., long path from the root to a leaf node) to make a prediction. They have the same problems as tall MNN-Trees, where the error in each level of the tree is compounded, resulting in poor accuracy with a higher memory and energy requirement.

Degenerated trees are formed because the centroid of many categories (a large super-group) does not sufficiently represent the categories. In large super-groups, the constituent images are more spread out (high intra-cluster variance), causing the centroid to misrepresent the images. This misrepresentation of the images leads to a phenomenon called *inversion of distances* [89]. Inversion of distances causes the similarity between two dissimilar categories to increase (distances

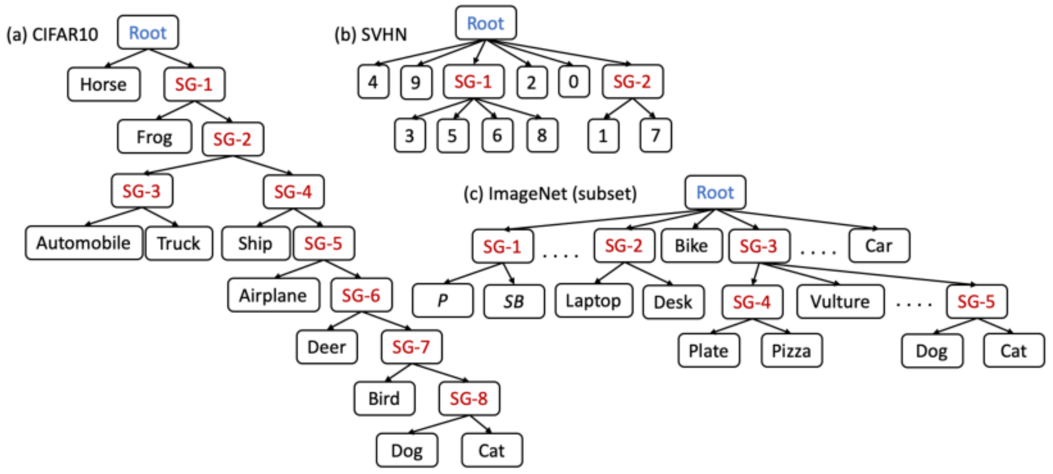


Fig. 15. Hierarchies obtained with different techniques and for different datasets. Well-balanced hierarchies that correspond to visual similarities achieve a better accuracy than other hierarchies. (a) Degenerate tree obtained with centroid-based hierarchical clustering [57] on the CIFAR-10 dataset. This tree has a high test error and consumes more energy. (b) MNN-Tree built for the SVHN dataset using the ASL similarity metric. This MNN-Tree is more balanced and achieves a low test error. (c) MNN-Tree built for the ImageNet 2012 (subset) dataset using the ASL similarity metric. Here, *P* and *SB* represent the categories *pomegranate* and *strawberry*, respectively. We can see that visually similar categories (although semantically dissimilar; e.g., *pizza* and *plate*) are grouped.

decrease) as more super-groups are formed. This leads to the formation of groups that are not intuitive. For example, in Figure 15(a), SG-5 contains visually dissimilar children: *airplanes* and SG-6 (a super-group containing animals). One method to overcome the inversion of distances problem is to use the average distance between every pair of feature vectors in the two categories (instead of distances between centroids of categories). However, as the numbers of categories and images increase, the computational cost associated with this similarity metric increases significantly.

Balanced hierarchies obtain a lower classification error and generally consume lesser energy than degenerate hierarchies. Figure 8 is the tree obtained by the proposed method for the CIFAR-10 dataset. It can be seen that it is significantly more balanced than the tree obtained with hierarchical clustering (Figure 15(a)). Figure 15(b) and Figure 15(c) show the balanced MNN-Trees obtained with the proposed method for the SVHN and ImageNet (subset) datasets, respectively. These more balanced trees are obtained without a significant increase in the computation costs when compared with the centroid-based hierarchical clustering method. Furthermore, such trees enable us to obtain significantly higher accuracy, as reported in Table 10.

4.4.4 Comparison with Semantic Hierarchy. Semantic hierarchies group categories that have related meanings even though they may look quite different. Here, birds and lions are grouped together as animals, and bicycles and trucks are together as vehicles. These categories may have low visual similarities. For example, bicycles and trucks look different. Figure 16 shows the relationship between the semantic similarities (measured by the Jiang-Conrath (JC) distance [90]) and the visual similarities (measured by the average distance between the centroids of the feature vectors of the categories) for every pair of the 20 categories in the ImageNet (subset) dataset. A smaller distance means the two pairs are similar. The red line in Figure 16 is the best fit line. Many pairs of categories (e.g., *Pizzas* and *Plates*) have high visual similarity and low semantic similarity.

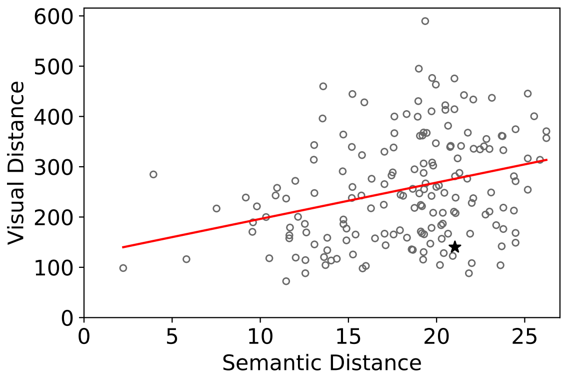


Fig. 16. Plot of semantic distances and visual distances between the categories in the ImageNet 2012 (subset) dataset. Visual similarities and semantic similarities have a low correlation. The red line indicates the best fit line for the given data points. The data point ⋆ corresponds to the categories *Pizza* and *Plate* that are visually similar but semantically dissimilar.

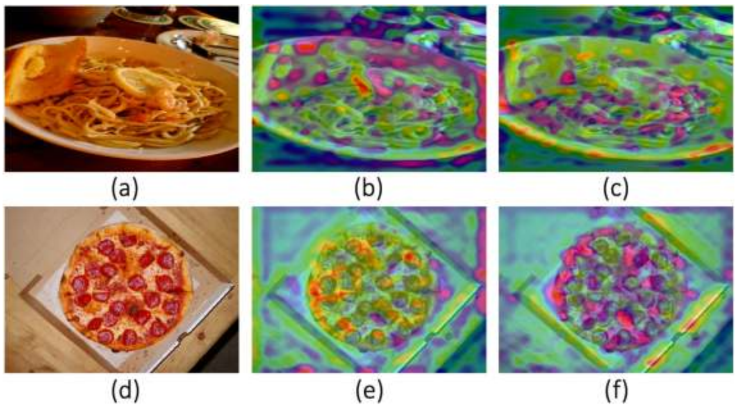


Fig. 17. Visualization of the differentiating features of *pizzas* and *plates* obtained with the GradCam [91] technique. (a) Original *plate* image. Heat map identifying important *plate* (b) and *pizza* (c) features in the plate image. (d) Original *pizza* image. Heat map identifying important *plate* (e) and *pizza* (f) features in the pizza image.

The JC distance [90], given in Equation (4), quantifies the distances between two objects, a and b, in a semantic taxonomy. Here, LCA is the lowest common ancestor of the two objects and p() corresponds to the probability of the object's occurrence.

$$D(a,b) = 2 \times log(\text{p}(\text{LCA}(a,b))) - (log(\text{p}(a)) + log(\text{p}(b))) \tag{4}$$

Semantic and visual similarities do not always correlate. An example is *plate* and *pizza*. They are semantically different (plate is not edible) but share similar visual features (e.g., round or oval shapes). Figure 17 shows an example to explain why they are commonly confused by DNNs. Here, the GradCam technique [91] is used to identify the differentiating features that are important for identifying a particular category. GradCam generates a heat map on the original image depicting the important differentiating features of a particular category. The features are color coded: the regions marked in red are the most important, followed by yellow, blue, and then green. For example, Figure 17(a) shows the original input image to the GradCam technique, and Figure 17(b) and

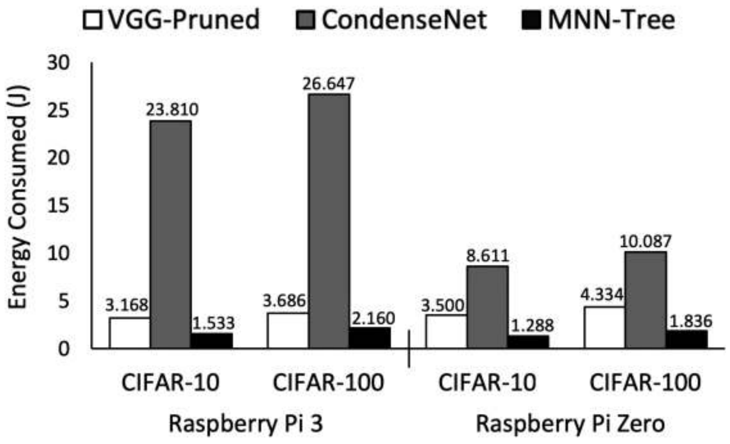


Fig. 18. Average energy consumption comparison for processing one image (averaged over 500 images) on a Raspberry Pi 3 and a Raspberry Pi Zero. The MNN-Tree consumes the least energy on both devices.

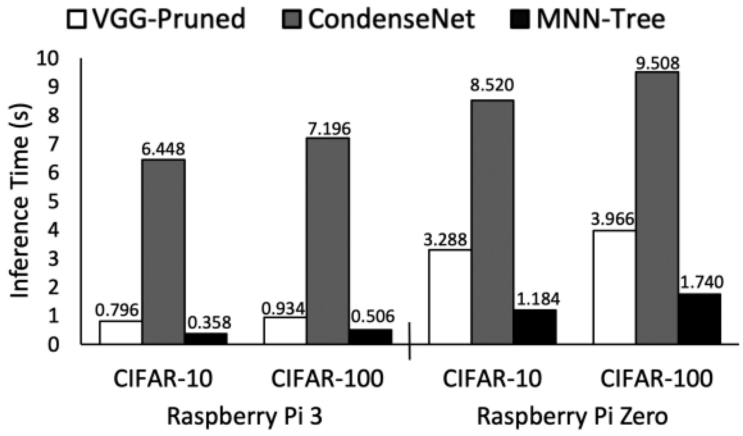


Fig. 19. Average time taken for processing one image (averaged over 500 images) on a Raspberry Pi 3 and a Raspberry Pi Zero. The MNN-Tree performs the fastest inference on both devices.

(c) show the heat maps marking the important features used by DNNs to identify plates and pizzas, respectively. The presence of red regions in both Figure 17(b) and (c) show that the differentiating features belonging to *plates* and *pizzas* are found in the original image of a plate. Similarly, the red regions in Figure 17(e) and (f) indicates that images of pizzas have several features resembling *plates* and *pizzas*. This example explains why semantic similarity and visual similarity are different. This work uses visual similarity for classification.

4.5 Performance Evaluation on Embedded Systems

4.5.1 Energy Consumption and Inference Time. The MNN-Tree consumes significantly less energy and performs inference faster than the existing techniques. Figure 18 and Figure 19 show the energy consumption and inference time of the different DNN architectures on a Raspberry Pi 3 and a Raspberry Pi Zero, respectively. For the MNN-Tree, the worst-case energy consumption (i.e., images are processed by the DNN modules along the longest path from the root to a leaf) is reported. The energy consumption is measured using a Yokogawa WT310E Power Meter. Since

Table 13. Average Time Taken (in Seconds) for Different Tasks Per Image

Time	VGG-Pruned	CondenseNet	MNN-Tree
Load	0.468	1.619	0.070
Execution	0.328	4.829	0.288

Note: Results are obtained for the CIFAR-10 dataset, measured on a Raspberry Pi 3. The MNN-Tree requires the least amount of memory and performs the fewest operations.

Table 14. Analysis of the Average Power Consumption When Running Different DNN Architectures on Different Devices

Device	VGG-Pruned	CondenseNet	MNN-Tree
Raspberry Pi Zero	1.064 W	1.010 W	1.088 W
Raspberry Pi 3	3.979 W	3.692 W	4.283 W

Note: On the Raspberry Pi Zero, the average power consumed is approximately the same. On the Raspberry Pi 3, the MNN-Tree has the highest average power because it does not trigger thermal throttling.

the MNN-Tree architecture reduces redundancies by using a small subset of the modules for every input, the energy consumption for the MNN-Tree is 52% lower than VGG-Pruned and 93.5% lower than CondenseNet. The MNN-Tree architecture requires 55% to 95% less time to classify an image on the Raspberry Pi 3. Similarly, the MNN-Tree requires 63% to 85% less energy and 64% to 86% less time on the Raspberry Pi Zero. It is not possible to run large architectures such as DenseNet, WideResNet, ResNet, MobileNet, and SqueezeNet on a Raspberry Pi 3 or a Raspberry Pi Zero. When we run these architectures on the embedded devices, we encounter a segmentation fault (we suspect that this occurs because the device runs out of memory). Moreover, the source code for models using the SVHN and EMNIST datasets is not readily available. This is why Figure 18 and Figure 19 compare the MNN-Tree with only the smaller architectures, VGG-Pruned and CondenseNet, on the CIFAR datasets.

Table 13 breaks down the running time while performing image classification on a Raspberry Pi 3. The MNN-Tree uses only a subset of the modules for each image and thus requires the least time to load the DNN models into memory and perform inference. CondenseNet performs significantly more operations and requires the highest execution time.

4.5.2 Average Power Consumption and Thermal Throttling. The Raspberry Pi Zero is a significantly smaller device and consumes less energy than the Raspberry Pi 3. The Raspberry Pi Zero contains a 1-GHz single-core CPU, and the Raspberry Pi 3 contains a 1.2-GHz quad-core CPU. The more powerful CPU is the reason the Raspberry Pi 3 performs inference significantly faster but also consumes more energy than the Raspberry Pi Zero. We report the average power consumed by these devices when running different DNN architectures in Table 14. On the Raspberry Pi Zero, the average power consumption remains almost constant across techniques. However, on the Raspberry Pi 3, the CondenseNet architecture has the lowest average power consumption, followed by VGG-Pruned. The MNN-Tree has the highest average power consumption. This observation seems counter-intuitive, as the MNN-Tree consumes the least energy per image. This can be explained by the thermal throttling of the Raspberry Pi 3. Thermal throttling causes a reduction in CPU and memory frequencies to avoid damage when the different components generate heat. This increases the time taken to process each image. This can be seen in Figure 20. When running the VGG-Pruned and CondenseNet architectures on the Raspberry Pi 3, thermal throttling

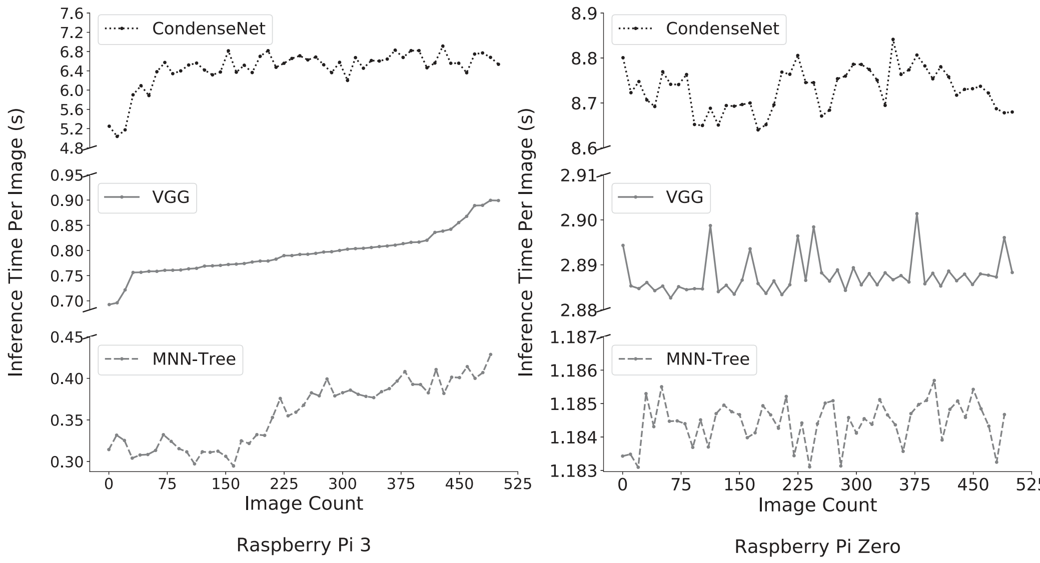


Fig. 20. Analyzing the effects of thermal throttling on the inference time on the Raspberry Pi 3 and Raspberry Pi Zero. As more images are processed, the time taken to classify each image increases. On the Raspberry Pi 3, the MNN-Tree is unaffected by thermal throttling for much longer than the other solutions. There is no significant thermal throttling on the Raspberry Pi Zero. Please note the breaks in the *y*-axis in the plots.

is observed almost immediately. The time taken to process one image steadily increases as more images are processed sequentially. The MNN-Tree encounters thermal throttling only after ≈170 images are processed. The MNN-Tree architecture is processed at the full CPU capacity for longer, ensuring a shorter running time. Since the average power is measured as $\frac{\text{energy}}{\text{time}}$, the smaller inference time is the reason for the higher average power consumption of the MNN-Tree. As explained in Section 4.2, the `vcgencmd get_throttled` command is used to check for thermal throttling. The MNN-Tree is least affected by thermal throttling on the Raspberry Pi 3. The inference time (per image) increases from 0.29 seconds to ≈0.42 seconds. For CondenseNet, the inference time increases from ≈5 seconds per image to ≈6.8 seconds after thermal throttling. Thus, the MNN-Tree is better suited for such embedded devices, because it is possible to run a DNN with the least amount of thermal throttling.

It is uncommon to observe thermal throttling on the Raspberry Pi Zero, because it contains a 1-GHz single-core CPU that does not dissipate much heat. The Raspberry Pi Zero's CPU temperature increases to approximately 50°C (measured with the `vcgencmd` command) under a heavy workload, and the CPU is designed to throttle only when the temperature exceeds 85°C. This is why the inference time per image for the three architectures on the Raspberry Pi Zero does not increase significantly with time. Even on the Raspberry Pi Zero, the MNN-Tree performs inference faster than the other architectures.

4.5.3 Module Analysis of the MNN-Tree. Depending on the number of modules that are used during inference, the energy consumption of the MNN-Tree varies. For the CIFAR-10 dataset, some categories require only two modules, whereas the other categories need three modules before the final classification is made. Figure 21 shows the energy consumption as the depth of the MNN-Tree increases. If a leaf node is closer to the root, image classification consumes less

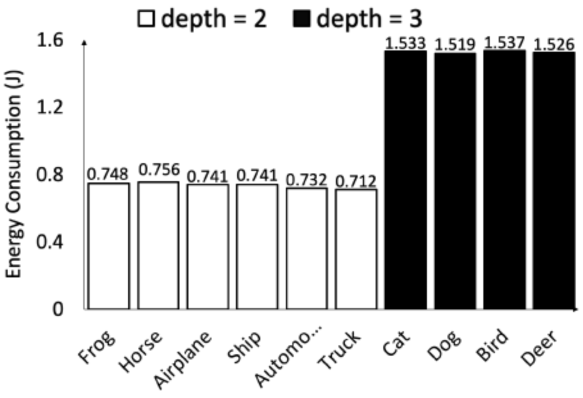


Fig. 21. Energy consumption per category of the CIFAR-10 dataset, for processing one image on a Raspberry Pi 3. The categories are organized by their depth in the MNN-Tree, as seen in Figure 8.

Table 15. Comparison of the Inference Time, Energy Consumption, Average Power, and Memory Requirement When a Single DNN Module, Two Levels of DNN Modules, and all DNNs Modules of the MNN-Tree (Seen in Figure 8) Are Loaded into Memory at a Time

Levels of Modules	CIFAR-10				CIFAR-100			
	Time (s)	Energy (J)	Average Power (W)	Memory (KB)	Time (s)	Energy (J)	Average Power (W)	Memory (KB)
1	0.358	1.533	4.283	806	0.506	2.160	4.268	832
2	0.356	1.684	4.732	1,466	0.504	2.325	4.614	2,692
3 (all modules)	0.344	1.716	4.981	1,802	0.502	2.476	4.933	3,512

Note: Results reported are per image, averaged over 500 images on a Raspberry Pi 3.

energy. For example, images from categories like *cat*, *dog*, *bird*, and *deer* require approximately 1.52 J of energy. Images from *frog* require only 0.748 J of energy.

It is possible to get a higher inference speed when multiple levels of modules are loaded into memory simultaneously. This analysis is presented in Table 15. When a single module is loaded into memory at a time (one child module is loaded into memory based on the output of the parent module), the memory requirement is the least. This also has the lowest inference speed. When two levels of the MNN-Tree are loaded into memory simultaneously (one module and all of its child modules), the inference speed increases. When all modules (the entire MNN-Tree) are loaded, the memory requirement and the inference speed increase further. However, when multiple modules are loaded into memory, a significant amount of energy is spent in performing memory operations. For example, in the CIFAR-10 dataset, the average power consumed by the Raspberry Pi 3 is about 4.23 W when a single module is loaded at a time. This value increases to close to 5 W when all modules of the MNN-Tree are loaded into memory for the inference of a single image.

4.6 Extensions and Future Work

4.6.1 Custom Hardware Accelerators. Only a small fraction of DNN parameters and activations contain non-zero values. The input density and parameter density measure the percentage of non-zero activations and parameters, respectively. The input and parameter densities of different DNN architectures are tabulated in Table 16. VGG-16 is the most sparse (has the least input and parameter density). The MNN-Tree architecture is less sparse compared with VGG-16 and AlexNet but is more sparse than DenseNet-190. To improve the inference time of sparse DNNs, some custom

Table 16. Input Density, Parameter Density, and Number of Parameters of Different DNN Architectures Compared with the MNN-Tree for the CIFAR-10 Dataset

Technique	Input Density	Parameter Density	No. of Parameters
AlexNet [92]	41.20%	46.20%	23,270 K
VGG-16 [13]	39.84%	35.23%	14,720 K
DenseNet-190 [74]	71.57%	92.37%	18,100 K
MNN-Tree	54.50%	84.50%	107 K

hardware accelerators have been proposed [93–95]. These accelerators perform sparse vector dot products with efficient vector inner-joins to avoid arithmetic operations with zeros. These accelerators decrease the inference time of sparse DNNs like VGG-16 and AlexNet considerably. They do not lead to significant speedups when used with the MNN-Tree and DenseNet. However, when deployed on general-purpose CPUs (e.g., Raspberry Pi), dense DNNs (the MNN-Tree and DenseNet) perform fewer redundant operations because fewer multiplications with zero are performed. This is an acceptable tradeoff because the MNN-Tree is designed for inference on general-purpose embedded devices like the Raspberry Pi.

4.6.2 MNN-Tree Inference with Large Batch Sizes. When we perform inference with batch size > 1 using the MNN-Tree, the different images in the batch would be able to traverse different branches of the tree. That could lead to greater memory and computation requirements. This problem, however, is outside of the scope of this article and could be investigated in our future work. By adding an image buffer to the DNN modules of the MNN-Tree, we can potentially solve this problem. The modules will buffer the images that are classified into their corresponding super-group by their parents instead of processing the images immediately. When this buffer is full (i.e., a batch of images has been classified into the corresponding super-group), the module will process all images in its buffer. This will enable DNN model reuse across several images, thus reducing the memory accesses. Such a strategy will increase the throughput but at the expense of increased latency.

5 CONCLUSION

We propose the MNN-Tree architecture as an improvement over monolithic DNNs by eliminating redundant computation and memory accesses to support fast, energy-efficient inference on embedded devices. The MNN-Tree architecture utilizes several small DNNs (called *modules*) in the form of a tree that work together to classify an image. Building a tree that corresponds to the visual similarity between categories of a dataset is a significant challenge associated with any hierarchical image classification technique. We propose a systematic and automatic method to build MNN-Trees, where each non-leaf node corresponds to a group of visually similar categories. This is done by defining a novel similarity metric: ASL. ASL achieves this task automatically for several different image datasets. Through experiments, we demonstrate that the MNN-Tree is more balanced and thus significantly outperforms the existing hierarchical classifiers in terms of accuracy. When compared with existing monolithic DNN architectures, the MNN-Tree architecture selectively uses modules to avoid redundant computation and memory accesses. This enables the MNN-Tree to operate with significantly reduced memory requirement, energy consumption, number of operations, and inference time, with only negligible impact on classification accuracy. We quantitatively evaluate the performance of various DNN architectures on a Raspberry Pi 3 and a Raspberry Pi Zero, and show the advantages of using the MNN-Tree over a single large DNN for image classification.

REFERENCES

[1] A. Mohan et al. 2017. Internet of Video Things in 2030: A world with many cameras. In *Proceedings of IEEE ISCAS 2017*. 1–4.

[2] S. Han et al. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. ArXiv:1510.00149 [cs].

[3] H. Zhao et al. 2018. Thermal-sensor-based occupancy detection for smart buildings using machine-learning methods. *ACM Transactions on Design Automation of Electronic Systems* 23, 4 (2018), Article 54, 21 pages.

[4] H. Huang et al. 2018. Distributed machine learning on smart-gateway network toward real-time smart-grid energy management with behavior cognition. *ACM Transactions on Design Automation of Electronic Systems* 23, 5 (2018), 1–26.

[5] S. Aghajanzadeh et al. 2020. Camera placement meeting restrictions of computer vision. In *Proceedings of IEEE ICIP 2020*.

[6] Y. Lu. 2019. Low-power image recognition. *Nature Machine Intelligence* 1, 4 (2019), 199.

[7] K. He et al. 2016. Deep residual learning for image recognition. In *Proceedings of IEEE CVPR 2016*. 770–778.

[8] M. Amir et al. 2018. Switching predictive control using reconfigurable state-based model. *ACM Transactions on Design Automation of Electronic Systems* 24, 1 (2018), Article 2, 21 pages.

[9] R. Fallahzadeh et al. 2018. Trading off power consumption and prediction performance in wearable motion sensors: An optimal and real-time approach. *ACM Transactions on Design Automation of Electronic Systems* 23, 5 (2018), Article 67, 23 pages.

[10] S. Anup et al. 2017. Visual positioning system for automated indoor/outdoor navigation. In *Proceedings of IEEE TENCON 2017*.

[11] S. Alyamkin et al. 2019. Low-power computer vision: Status, challenges, and opportunities. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 411–421.

[12] K. Gauen et al. 2018. Three years of low-power image recognition challenge. In *Proceedings of IEEE DATE 2018*.

[13] K. Simonyan et al. 2014. Very deep convolutional networks for large-scale image recognition. ArXiv:1409.1556 [cs].

[14] Y. Cheng et al. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of IEEE ICCV 2015*.

[15] A. Goel. 2019. Modular Neural Networks. Retrieved August 8, 2020 from https://github.com/abhinavgoel95/Modular_Neural_Networks.

[16] I. Ghodgaonkar et al. 2020. Observing responses to the COVID-19 pandemic using worldwide network cameras. ArXiv:2005.09091.

[17] C. Szegedy et al. 2013. Deep neural networks for object detection. In *Proceedings of Advances in NeurIPS 2013*. 2553–2561.

[18] J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (1986), 81–106.

[19] T. Cover et al. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (1967), 21–27.

[20] N. Friedman et al. 1997. Bayesian network classifiers. *Machine Learning* 29 (1997), 131–163.

[21] S. Kaski. 1998. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Proceedings of IEEE IJCNN 1998 and IEEE WCCI1998*, Vol. 1. 413–418.

[22] G. Huang et al. 2018. CondenseNet: An efficient DenseNet using learned group convolutions. In *Proceedings of IEEE CVPR 2018*.

[23] S. Bianco et al. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE Access* 6 (2018), 64270–64277.

[24] A. Goel et al. 2020. A survey of methods for low-power deep learning and computer vision. In *Proceedings of IEEE WF-IoT 2020*.

[25] M. Rastegari et al. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of ECCV 2016*. 525–542.

[26] H. Albalawi et al. 2017. Training fixed-point classifiers for on-chip low-power implementation. *ACM Transactions on Design Automation of Electronic Systems* 22, 4 (2017), 69:1–69:18.

[27] H. Li et al. 2016. Pruning filters for efficient ConvNets. ArXiv:1608.08710 [cs].

[28] A. Goel et al. 2018. CompactNet: High accuracy deep neural network optimized for on-chip implementation. In *Proceedings of IEEE Big Data2018*.

[29] L. Jiang et al. 2019. Energy-efficient and quality-assured approximate computing framework using a co-training method. *ACM Transactions on Design Automation of Electronic Systems* 24, 6 (2019), Article 59, 25 pages.

[30] F. N. Iandola et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. ArXiv:1602.07360 [cs].

[31] M. Sandler et al. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of IEEE CVPR 2018*. 4510–4520.

[32] C. Szegedy et al. 2017. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Proceedings of ACM AAAI 2017*. 4278–4284.

[33] A. Sironi et al. 2015. Learning separable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), 94–106.

[34] E. Denton et al. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of Advances in NeurIPS 2014*. 1269–1277.

[35] M. Jaderberg et al. 2014. Speeding up convolutional neural networks with low rank expansions. ArXiv:1405.3866.

[36] G. Zhong et al. 2019. Synergy: An HW/SW framework for high throughput CNNs on embedded heterogeneous SoC. *ACM Transactions on Embedded Computing Systems* 18, 2 (2019), Article 13, 23 pages.

[37] J. Li et al. 2018. SynergyFlow: An elastic accelerator architecture supporting batch processing of large-scale deep neural networks. *ACM Transactions on Design Automation of Electronic Systems* 24, 1 (2018), 1–27.

[38] Y. Cheng et al. 2017. A survey of model compression and acceleration for deep neural networks. ArXiv:1710.09282 [cs].

[39] G. Hinton et al. 2015. Distilling the knowledge in a neural network. ArXiv:1503.02531 [cs, stat].

[40] J. Ba et al. 2014. Do deep nets really need to be deep? In *Proceedings of Advances in NeurIPS 2014*. 2654–2662.

[41] J. Guérin et al. 2017. CNN features are also great at unsupervised classification. ArXiv:1707.01700 [cs].

[42] V. Di Gesú et al. 1999. Distance-based functions for image comparison. *Pattern Recognition Letters* 20 (1999), 207–214.

[43] R. Zhang et al. 2015. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing* 24, 12 (2015), 4766–4779.

[44] H. Zhu et al. 2016. Deep hashing network for efficient similarity retrieval. In *Proceedings of AAAI 2016*. 2415–2421.

[45] G. Griffin et al. 2008. Learning and using taxonomies for fast visual categorization. In *Proceedings of IEEE CVPR 2008*. 1–8.

[46] J. Deng et al. 2011. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Proceedings of Advances in NeurIPS 2011*.

[47] A. Beygelzimer et al. 2009. Conditional probability tree estimation analysis and algorithms. In *Proceedings of ACM UAI 2009*. 51–58.

[48] X. Yuan et al. 2006. Automatic video genre categorization using hierarchical SVM. In *Proceedings of ICIP 2006*. 2905–2908.

[49] M. Rastegari et al. 2012. Attribute discovery via predictable discriminative binary codes. In *Proceedings of ECCV 2012*. 876–889.

[50] P. Panda et al. 2017. FALCON: Feature driven selective classification for energy-efficient image recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 12 (2017), 2017–2029.

[51] A. Torralba et al. 2008. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30 (2008), 1958–1970.

[52] J. Redmon et al. 2016. 2016. YOLO9000: Better, faster, stronger. ArXiv:1612.08242 [cs].

[53] A. Zweig et al. 2007. Exploiting object hierarchy: combining models from different category levels. In *Proceedings of IEEE ICCV 2007*.

[54] V. Peluso et al. 2018. Scalable-effort ConvNets for multilevel classification. In *Proceedings of IEEE/ACM ICCAD 2018*. 1–8.

[55] S. Chen et al. 2015. Discriminative hierarchical k-means tree for large-scale image classification. *IEEE Transactions on Neural Networks and Learning Systems* 26, 9 (2015), 2200–2205.

[56] M. Marszalek et al. 2008. Constructing category hierarchies for visual recognition. In *Proceedings of ECCV 2008*. Vol. 5305. 479–491.

[57] Y. Lukic et al. 2016. Speaker identification and clustering using convolutional neural networks. In *Proceedings of IEEE MLSP 2016*. IEEE, Los Alamitos, CA, 1–6.

[58] A. K. Jain et al. 1997. Object detection using Gabor filters. *Pattern Recognition* 30 (1997), 295–309.

[59] H. Levkowitz et al. 1993. GLHS: A generalized lightness, hue, and saturation color model. *CVGIP: Graphical Models and Image Processing* 55 (1993), 271–285.

[60] Y. Qu et al. 2017. Joint hierarchical category structure learning and large-scale image classification. *IEEE Transactions on Image Processing* 26, 9 (2017), 4331–4346.

[61] G. A. Miller. 1995. WordNet: A lexical database for English. *Communications of the ACM* 38 (1995), 39–41.

[62] R. Xia et al. 2014. Supervised hashing for image retrieval via image representation learning. In *Proceedings of AAAI 2014*. 2156–2162.

[63] F. Shen et al. 2018. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 12 (2018), 3034–3044.

[64] J. Wang et al. 2010. Semi-supervised hashing for scalable image retrieval. In *Proceedings of IEEE CVPR 2010*. 3424–3431.
[65] Z. Wang et al. 2018. Learning fine-grained features via a CNN tree for large-scale classification. *Neurocomputing* 275 (2018), 1231–1240.
[66] D. Roy et al. 2018. Tree-CNN: A hierarchical deep convolutional neural network for incremental learning. *ArXiv:1802.05800.*
[67] M. Sun et al. 2013. Find the best path: An efficient and accurate classifier for image hierarchies. In *Proceedings of IEEE ICCV 2013*. 265–272.
[68] Y. Guo et al. 2016. Dynamic network surgery for efficient DNNs. In *Proceedings of Advances in of NeurIPS 2016*. 1379–1387.
[69] L. A. Zadeh. 1965. Fuzzy sets. *Information and Control* 8, 3 (1965), 338–353.
[70] N. D. Singpurwalla et al. 2004. Membership functions and probability measures of fuzzy sets. *Journal of the American Statistical Association* 99 (2004), 867–889.
[71] M. Tan et al. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of ICML 2019*. 6105–6114.
[72] P. Kontschieder et al. 2015. Deep neural decision forests. In *Proceedings of ICCV 2015*. 1467–1475.
[73] A. Roy et al. 2016. Monocular depth estimation using neural regression forest. In *Proceedings of IEEE CVPR 2016*. 5506–5514.
[74] G. Huang et al. 2017. Densely connected convolutional networks. In *Proceedings of IEEE CVPR 2017.*
[75] A. Krizhevsky et al. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report TR-2009. University of Toronto.
[76] Y. Netzer et al. 2011. Reading digits in natural images with unsupervised feature learning. In *Proceedings of the NeurIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning.*
[77] G. Cohen et al. 2017. EMNIST: An extension of MNIST to handwritten letters. ArXiv:1702.05373 [cs].
[78] J. Deng et al. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of IEEE CVPR 2009*. 248–255.
[79] G. Griffin et al. 2007. *Caltech-256 Object Category Dataset*. Technical Report. Available at http://authors.library.caltech.edu/7694.
[80] Yokogawa. 2017. WT310E/TW310EH/WT332E/WT333E Digital Power Meter: User's Manual. Retrieved August 9, 2020 from https://cdn.tmi.yokogawa.com/IMWT310E-01EN.pdf.
[81] PyTorch. 2019. Torch.utils.data. Retrieved August 9, 2020 from https://pytorch.org/docs/stable/data.html.
[82] A. Canziani et al. 2016. An analysis of deep neural network models for practical applications. ArXiv:1605.07678
[83] B. Chen et al. 2018. Introducing the CVPR 2018 On-Device Visual Intelligence Challenge. *Google AI Blog*. Retrieved August 9, 2020 from http://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html.
[84] A. Mordvintsev et al. 2013. Getting Started with Videos: OpenCV-Python Documentation. Retrieved August 9, 2020 from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html.
[85] D. P. Kingma et al. 2014. Adam: A method for stochastic optimization. ArXiv:1412.6980 [cs].
[86] A. Goel. 2019. Image Classification with the Modular Neural Network Tree. Retrieved August 9, 2020 from https://youtu.be/gdae-v-ZyVs.
[87] S. Zagoruyko et al. 2016. Wide residual networks. ArXiv:1605.07146 [cs].
[88] E. Dufourq et al. 2017. EDEN: Evolutionary deep networks for efficient machine learning. In *Proceedings of PRASA-RobMech 2017.*
[89] T. Hastie et al. 2001. *The Elements of Statistical Learning*. Springer.
[90] J. J. Jiang et al. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of ROCLING 1997*. 19–33.
[91] R. R. Selvaraju et al. 2016. Grad-CAM: Visual explanations from deep networks via gradient-based localization. ArXiv:1610.02391 [cs].
[92] A. Krizhevsky et al. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of NeurIPS 2012.*
[93] A. Gondimalla et al. 2019. SparTen: A sparse tensor accelerator for convolutional neural networks. In *Proceedings of ACM/IEEE MICRO 2019*. 151–165.
[94] J. Albericio et al. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of ACM/IEEE ISCA 2016.*
[95] A. Parashar et al. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of ACM ISCA 2017.*

Received November 2019; revised April 2020; accepted June 2020