

Received 28 June 2024, accepted 8 July 2024, date of publication 10 July 2024, date of current version 19 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3426350

## APPLIED RESEARCH

# Optimized Visual Recognition Through a Deep Convolutional Neural Network With Hierarchical Modular Organization

ADE CLINTON SITEPU<sup>1</sup> AND CHUAN-MING LIU<sup>2</sup>, (Member, IEEE)

<sup>1</sup>College of Electrical Engineering and Computer Science, National Taipei University of Technology, Taipei 10608, Taiwan

<sup>2</sup>Department of Computer Science and Information Engineering, National Taipei University of Technology, Taipei 10608, Taiwan

Corresponding author: Chuan-Ming Liu (cmliu@ntut.edu.tw)

This work was supported in part by the National Science and Technology Councils, Taiwan, under Grant NSTC 112-2221-E-027-106.

**ABSTRACT** Artificial Intelligence, including machine learning and deep convolutional neural networks (DCNNs), relies on complex algorithms and neural networks to process and analyze data. DCNNs for visual recognition often require access to high-performance hardware, such as GPUs and cloud-based computing resources, to perform tasks efficiently. Visual recognition requires DCNNs with numerous layers. Fully connected layers in DCNNs are often the most computationally intensive. These layers connect every neuron in one layer to every neuron in the next layer, resulting in a large number of parameters to compute. To mitigate redundancy and make DCNNs more efficient, this article implements and demonstrates the concept to identifying and removing redundant or low-impact connections from fully connected layers using convolution neural network with hierarchical modular organization. The modularity of the DCNNs is built based on the cluster hierarchy of the similar image. These clusters are created based on a novel similarity metric, which measures how closely related images are to each other. The architecture uses multiple smaller DCNNs, referred to as modules, designed to progressively classify images into super clusters according to their similarity. Experimental results using popular image datasets show that the proposed DCNNs model to optimized number of operations by 49% to 99% and keeps its performance comparable.

**INDEX TERMS** Visual recognition, mean softmax likelihood similarity, sigmoidal membership function, hierarchical modular deep convolution neural network.

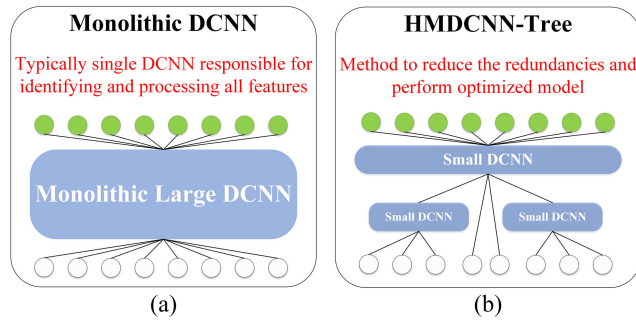
## I. INTRODUCTION

Artificial Intelligence (AI) tasks such as natural language processing, visual recognition, and autonomous decision-making require substantial computational power beside its superior performance. The need for AI, especially Deep Convolution Neural Networks (DCNNs), in image processing is substantial and has rapidly grown in today's era. Convolutional Neural Networks (CNNs) have demonstrated remarkable capabilities in visual recognition and understanding, making them indispensable in a wide range of applications. DCNNs techniques have brought about an advance impact in the field of artificial intelligence (AI) and show remarkable success in visual recognition (computer

vision) domains. In computer vision, DCNNs trained on large-scale image datasets have achieved high accuracy [1]. However, the exceptional performance of DCNNs comes with the considerable cost of computational complexity.

Most DCNNs, such as VGG architecture [2] and ResNet [3] architecture, typically adopt monolithic structures. In this type of architecture, as illustrated in Fig. 1 (a), a single DCNN is responsible to recognize and process features related to all classes, making decisions across various tasks. This design requires a significant number of neurons and layers in the DCNN. However, during the processing of each image, only a portion of these neurons show significant activation, resulting in fundamental redundancies. These redundancies contribute significantly to the DCNN's efficiency. In Fig. 1 (b), by reducing the irregular and entangled path of the DCNNs, we try to reduce the computational

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Hossein Moayeri<sup>1</sup>.



**FIGURE 1.** Types of DCNN architectures where (a) DCNNs are usually over-parameterized; and (b) HMDCNN-Trees architecture removes the irregular and entangled in DCNNs to reduce the model size and computational cost.

cost without losing its performance significantly. To address this issue and enhance visual recognition, we propose the Hierarchical Modular Deep Convolution Neural Network Tree (HMDCNN-Tree) architecture.

The HMDCNN-Tree architecture aims to address these challenges by reducing redundant operations inherent in monolithic DCNNs. Traditional evolutionary algorithms used in DCNNs often result in irregular and entangled structures. HMDCNNs offer an energy-accuracy trade-off by processing inputs along a single path from the root to a leaf, thus aiming to reduce the model size and computation cost compared to traditional DCNNs. The HMDCNN-Tree constructs several small DCNNs (referred to as modules) designed to classify between clusters of similar classes, known as “super-clusters”.

Initially, the root module processes the image. The parent module’s fully connected (FC) layer classifies among its children, as denoted by the dotted lines in Fig. 2. The solid lines illustrate how the output of the convolutional layer in a parent module is utilized as input to its corresponding child module. The child module avoids duplicating operations performed by the parent, minimizing redundancies. Through the reuse of feature maps, the HMDCNN-Tree operates similarly to a DCNN with numerous layers distributed across small modules. This approach is effective in reducing redundant operations and maintaining a low error classification rate. The procedure is iterated from the root to the child module, resulting in leaf modules (output) that may be positioned at different distances from the root. The tree may not be balanced; for instance, in Fig. 2, the leaf (output) distance of super-cluster 1 from the root is longer than the super-cluster 2.

There are a few challenges when using the HMDCNN-Tree. The first task involves finding the visual similarity that may exist between the classes. To detect visual similarities and quantify confusion between classes, we utilize Mean Softmax Likelihood, which uses the output of a DCNN. The second challenge is to devise a systematic method for selecting hyperparameters for each DCNN module and then constructing the tree based on visual similarity to achieve a

balance between low error and efficient inference speed. The third challenge is that every module in the HMDCNN-Tree requires training sequentially, with the parent being trained first. The parent’s feature maps can be used as inputs for the child, resulting in fewer redundancies. After training, the HMDCNN-Tree modules must be used for hierarchical image classification, which is the fourth challenge.

The proposed architecture begins by identifying visual similarities among varied classes over a novel similarity metric. In our work, we apply the Mean Softmax Likelihood (MSL) techniques to find the visual similarity between classes within a dataset [1]. Images within the same class of similarity are grouped into “super-clusters,” and these super-clusters are further organized into larger super-clusters, forming a hierarchical tree structure. The HMDCNN-Tree architecture operates multiple smaller DCNNs, which we called as “modules”, in order to classify various of each super-clusters. A super-cluster is chosen by a module upon receiving an image, and another module then conducts classification within the subclasses of that selected super-cluster. Moreover, the modules associated with other super-clusters remain inactive during the inference process for that particular image. This selective activation ensures that only a subset of modules is utilized during inference, effectively avoiding redundant operations.

We present and validate using a metric image similarity in this article, Mean Softmax Likelihood (MSL), which is utilized to identify and group similar classes. The MSL metric facilitates the creation of super-clusters of varying sizes within an HMDCNN-Tree. This similarity metric involves computing the mean output (softmax) of a DCNN for a specific class  $X$ , based on input images belonging to another class  $Y$ . The softmax output of the DCNN presents to quantify the confusion between classes, with a high softmax output for class  $X$  (when inputs are from class  $Y$ ) indicating frequent confusion between the visually similar classes. Additionally, our experiments aim to determine the performance differences of MSL compared to other monolithic architectures in terms of accuracy across popular image datasets.

This article provides the following contributions:

- Introducing the HMDCNN-tree, a hierarchical tree structure composed of interconnected nodes, where nodes with children are termed super-clusters. Each node, except leaf nodes, contains a module trained to classify inputs into its children, preventing unnecessary activations and enhancing visual recognition efficiency.
- Utilizing the concept of fuzzy sets and sigmoid membership function (sgmf), the creation of modules in the HMDCNN-tree is based on visual similarities between new and old classes, allowing for dynamic adaptation without relying on fixed optimal thresholds.
- The trade-off between accuracy and DCNN architecture size is addressed by using the metric change in efficiency gain ( $\Delta EG$ ) and fixed optimal threshold values, enabling the selection of optimal configurations for each module

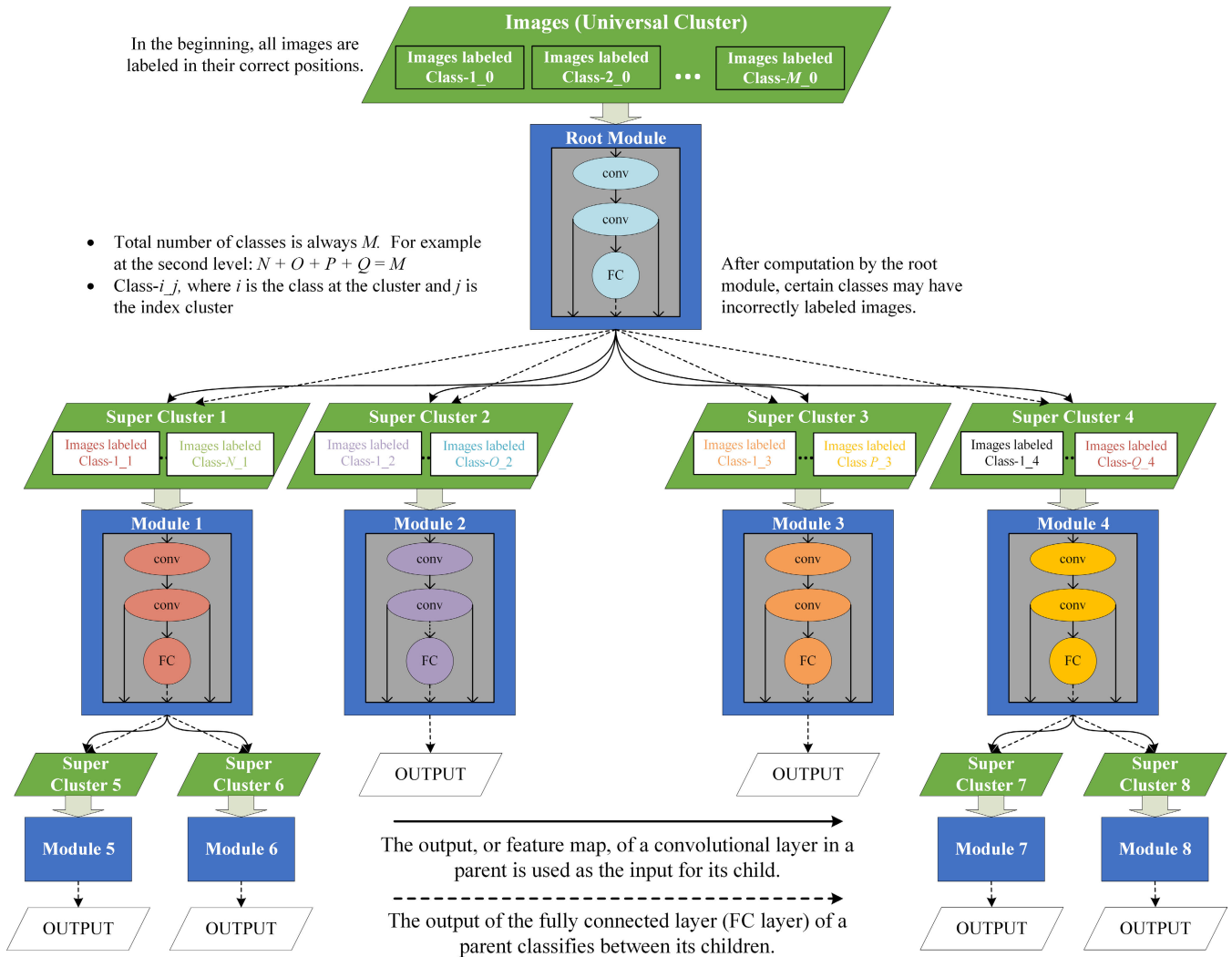


FIGURE 2. The HMDCNN-Tree architecture.

and significantly reducing DCNN size without losing its performance across popular image datasets.

II. BACKGROUND AND RELATED WORK

A. TRADITIONAL DCNN ON VISUAL RECOGNITION

Visual recognition is integral to various image processing tasks across applications such as image enhancement [28], hyperspectral image classification [29], [30], image segmentation [31], and object detection [32]. Traditionally, visual recognition relied on two-stage methods: feature extraction followed by classification using handcrafted features and classifiers. These methods, though effective for specific tasks, struggled with generalization and robustness due to their inability to capture complex semantic information.

The introduction of deep learning, particularly Deep Convolutional Neural Networks (DCNNs) [1], revolutionized visual recognition by integrating feature extraction and classification into end-to-end trainable models. DCNNs, comprising layers like convolutional (Conv) and fully

TABLE 1. Cifar-10 test accuracy by machine learning and DCNNs techniques.

Dataset	Method	Test Error
CIFAR-10	K-Nearest Neighbors [11]	0.583
	Naive Bayes' Classifier [12]	0.703
	Decision Tree [14]	0.733
	Random Forest [12]	0.505
	ResNet (DCNNs) [3]	0.064
	CondenseNet (DCNNs) [3]	0.034

connected (FC), excel in learning hierarchical features [10]. These networks produce activation maps that highlight image features crucial for recognition tasks. However, their computational demands limit deployment on edge devices. Table 1 contrasts the computational efficiency of DCNNs with traditional machine learning (ML) techniques like Decision Trees [13], Naive Bayes' classifiers [12], and clustering algorithms [12] on the CIFAR-10 dataset [11].

DCNNs like ResNet and CondenseNet [3] significantly outperform traditional methods due to their ability to learn complex representations.

AlexNet [4], which won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), marked a pivotal moment in DCNNs' success. Subsequent architectures like VGG nets [2] and GoogleNet [5] introduced deeper networks and innovative module structures (e.g., Inception) that improved performance and efficiency [6], [7], [8]. However, deeper networks introduced challenges like gradient vanishing, addressed by ResNet [3] through residual learning. EfficientNet [9], with its scalable depth, width, and resolution, further optimized model efficiency without compromising accuracy.

**B. MODULARITY IN CONVOLUTION NEURAL NETWORKS**

In the realm of convolution neural networks, modularity refers to the characteristic of a network that allows it to be broken down into several subnetworks according to connectivity patterns. It can be contended that the transition toward functional modularity in the brain and biological neural networks represents a significant advancement in neuroscience, similar to the impact of the neuron doctrine.

Modularizing neural networks can be achieved through various techniques that operate at different levels of abstraction. These techniques are applied to specific levels to introduce modularity into the network topology. M. Amer's (2019) provides a taxonomy of these techniques, categorized by the abstraction level they utilize to achieve modularity. Each technique is analyzed, detailing its fundamental rationale, advantages, and disadvantages (as shown in Table 2), and showcasing prominent use cases from the literature [15]. The primary levels at which these modularization techniques operate are complementary. A modular neural network (MNN) is created using a sequence of techniques known as a *modularization chain*. It consists of various levels of the neural network environment, each contributing to the development of the modular neural network. Therefore, each modularization chain is associated with a specific type of MNN.

A modularization chain begins with the optional step of partitioning the "domain" and it can be optional. After that, choose a modular "topological" structure for the model. Then, choose the "integration" and "formation" techniques for constructing the model and integrating its various modules. For instance, when developing an MNN for EMNIST dataset for classification, a typical modularization chain would proceed as follows [15]:

- 1) "Domain": We may choose to augment the dataset by applying a specific image processing function to create a copy of each image, extracting particular information. Subsequently, we consider both the original and processed images as distinct subdomains.
- 2) "Topology": We may select for a multi-path topology, where one path of the network takes the original image as input, while the others take the processed ones.

**TABLE 2. The advantages and disadvantages of techniques in MNN.**

Technique	Advantages	Disadvantages
<i>A. Domain</i>		
1. Manual	<ul style="list-style-type: none"> <li>• Prior knowledge integration</li> <li>• Fine control over partitions</li> </ul>	<ul style="list-style-type: none"> <li>• Partitions are hard to define</li> <li>• Relation between decomposition and solution is not straightforward</li> <li>• Separation of variation factors is hard</li> </ul>
2. Learned	<ul style="list-style-type: none"> <li>• Capture useful relations not tractable by human designer</li> </ul>	<ul style="list-style-type: none"> <li>• Computational cost and extra step of learning the decomposing model</li> </ul>
<i>B. Topology</i>		
1. HCNR	<ul style="list-style-type: none"> <li>• Sparse connectivity</li> <li>• Short average path</li> </ul>	<ul style="list-style-type: none"> <li>• Complex structure</li> <li>• Hard to analyse and adapt to problems</li> <li>• Formation difficulty</li> </ul>
2. Repeated block		
2.1. Multi-path	<ul style="list-style-type: none"> <li>• Parallelizable</li> <li>• Suitable for multi-modal integration</li> </ul>	<ul style="list-style-type: none"> <li>• Additional hyperparameters</li> <li>• Currently lacks theoretical justification</li> </ul>
2.2. Modular node	<ul style="list-style-type: none"> <li>• Computational capability with relatively fewer parameters</li> <li>• Can be adapted for hardware implementation</li> </ul>	<ul style="list-style-type: none"> <li>• Additional hyperparameters</li> </ul>
2.3. Sequential	<ul style="list-style-type: none"> <li>• Deep composition</li> </ul>	<ul style="list-style-type: none"> <li>• Hard training</li> <li>• Excessive depth is arguably unnecessary</li> </ul>
2.4. Recursive	<ul style="list-style-type: none"> <li>• Readily adaptable to recursive problems</li> <li>• Deep nesting with short paths</li> </ul>	<ul style="list-style-type: none"> <li>• Excessive depth is arguably unnecessary</li> </ul>
3. Multi-Architectural	<ul style="list-style-type: none"> <li>• Better collective performance</li> <li>• Error tolerance</li> </ul>	<ul style="list-style-type: none"> <li>• Computationally complex</li> </ul>
<i>C. Formation</i>		
1. Manual	<ul style="list-style-type: none"> <li>• Prior knowledge integration</li> <li>• Fine control over components</li> </ul>	<ul style="list-style-type: none"> <li>• Hard in practice</li> </ul>
2. Evolutionary	<ul style="list-style-type: none"> <li>• Adaptable way for modularity formation</li> <li>• Suitable for HCNR formation</li> </ul>	<ul style="list-style-type: none"> <li>• Lengthy and computationally complex</li> </ul>
3. Learned	<ul style="list-style-type: none"> <li>• Dynamic formation of modularity</li> <li>• Sample from large set of models</li> </ul>	<ul style="list-style-type: none"> <li>• Computational complexity</li> <li>• In implicit learned variant, networks are densely connected</li> </ul>
<i>D. Integration</i>		
1. Arithmetic-logic	<ul style="list-style-type: none"> <li>• Prior knowledge integration</li> <li>• Loosely coupled modules</li> </ul>	<ul style="list-style-type: none"> <li>• Difficult in practice</li> </ul>
2. Learned	<ul style="list-style-type: none"> <li>• Captures complex relations</li> </ul>	<ul style="list-style-type: none"> <li>• Computationally complex</li> <li>• Tightly coupled modules</li> </ul>

- 3) "Formation": We may apply an evolutionary algorithm to construct the multi-path topology, with a constraint requiring exactly two paths. Alternatively, manual formation requires expertise in design and involves a process of trial and error.
- 4) "Integration": We may integrate the outputs of each path into the final system output, either through the evolutionary process itself or through a post-formation learning (or fine-tuning) algorithm.

We can conclude the modularization chain technique that used in Hierarchical Modular Deep Convolution Neural Networks (HMDCNN) Tree as: learned domain, multi path topology and manual formation without using integration. Hierarchical MNNs use multiple small DCNNs in the form of a tree, as seen in Fig. 2. Each small DCNN specializes in an intermediate classification between groups of similar classes. In each level of the hierarchy, a small DCNN uses the activation map of its parent and makes an intermediate classification into progressively smaller groups, until a leaf DCNN provides the final output.

TABLE 3. Notation.

Symbol	Definition
$\exp(x)$	Standard exponential function. $\exp(x)$ is $e^x$ , where $e = 2.718281828\dots$ , the Euler's number
$\sigma$	Softmax function
$\vec{z}$	Input data derived from the fully-connected output values
$z_i$	The $i$ -th element of vector $\vec{z}$
$(\vec{z})_i$	A softmax value of $z_i$ element
$L(A)_B$	The mean softmax value (MSL) output for class $A$ , when the inputs belong to class $B$
$ A $	The current node's actual total number of images labeled as $A$ .
$\mu$	Sigmoidal Membership Function (sgmf)

### III. DESIGN CONCEPT

The construction of the HMDCNN-Tree requires selecting the suitable size for each DCNN module and then calculating the MSL for class clustering. The construction of the HMDCNN-Tree begins with a top-down manner that starts from the root. This section will explain the idea of creating super-cluster based on the input similarity and also the design concept of the HMDCNN-Tree.

#### A. IMAGE METRIC SIMILARITY: MEAN SOFTMAX LIKELIHOOD (MSL)

A DCNN ends with the softmax layer. In a classification problem, this layer assigns a probability to each class. The softmax function, which stands  $\sigma$ , is calculated using the element-wise method outlined in [17]:

$$\sigma(\vec{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)} \quad (1)$$

Here,  $\vec{z}$  is defined as an input vector  $\vec{z} = (z_1, z_2, \dots, z_n)$ , comprising raw outputs from a DCNN. Each input represents the value of its corresponding class. As a result, the number of elements  $n$  in  $\vec{z}$  corresponds to the total number of classes at the node. Consider the  $i$ -th entry in the softmax output vector  $\sigma(\vec{z})$  as the probability of the test input belonging to class  $i$ . In Equation (1),  $\sum_{j=1}^n \exp(z_j)$  is the sum of the exponential of all elements in the input vector.

Let's consider an image labeled as class  $A$ . The term  $\sigma(\vec{z})_A$  represents the softmax value indicating the probability of the image belonging to class  $A$ . For comparison, the DCNN also calculates  $\sigma(\vec{z})_B$ , representing the softmax value for the image belonging to class  $B$ . If  $\sigma(\vec{z})_A > \sigma(\vec{z})_B$ , then the image is correctly classified as  $A$ . However, there are instances when the value of  $\sigma(\vec{z})_B$  is greater than  $\sigma(\vec{z})_A$ , leading to incorrect classification.

Mean Softmax Likelihood (MSL) is a method used to quantify the similarity between classes. From Equation (1), MSL function is defined as:

$$L(A)_B = \frac{\sum_{i=0}^{|A|} \sigma(\vec{A})_B}{|A|} \quad (2)$$

The equation (2) describes how the MSL is computed: the vector  $\vec{A}$  represents an input vector of raw outputs from

a DCNN, where  $\vec{A} = (A_0, A_1, \dots, A_{|A|})$ , and it has been misclassified as class  $B$ .  $|A|$  is the total number of input images labeled as class  $A$  in the node.  $\sigma(\vec{A})_B$  represents all the softmax values of instances from class  $A$  that are misclassified as class  $B$ .

Let's consider a hypothetical example using samples from the CIFAR-10 dataset. For example,  $\sigma(\text{truck})_{\text{automobile}}$  refers to the softmax output for the class "truck" when the input image is misclassified as the class "automobile". The MSL output for class  $A$  is represented by Equation (2) when the inputs are actually from class  $B$ . If the value of  $L(A)_B$  is high, it suggests that the classifier is uncertain or confused about the distinction between the two classes,  $A$  and  $B$ .

However, creating clusters of visually similar classes is not easily achieved with a fixed threshold. For instance, establishing the value at  $\frac{1}{47}$ , which represents the softmax value obtained from the trained DCNN, taking 47 classes as the denominator, implies that a class with an MSL of 0.01 will not be selected, whereas a class with an MSL of 0.02 will always be included. Using the *fuzzy sets* technique, it is possible to overcome the limitation of needing a fixed threshold [18]. The *sigmoidal membership function* (sgmf) assesses the degree of membership for each class, with the output of the membership function indicating the likelihood of a specific class's presence in a super-cluster. With  $a$  as the slope parameter and  $b$  as the center parameter, which determines the point of maximum membership in the context of fuzzy logic, the sgmf can be defined as:

$$\mu(o) = \frac{1}{1 + \exp(-a(o - b))} \quad (3)$$

The symbol  $\mu(o)$  denotes the membership function, representing the degree to which an element  $o$  belongs to a fuzzy set. The probability of clustering two categories into a single super-cluster is obtained from the output of the symbol  $\mu$ . Consider  $M$  as the total number of image classes within a node. For instance, in the EMNIST dataset, at the root  $M$  is 47.

$$p(A \sim B) = \mu(L(A)_B) = \frac{1}{1 + \exp\left(-10M \times \left(L(A)_B - \frac{1}{M}\right)\right)} \quad (4)$$

This article adopts  $a = 10M$  and  $b = \frac{1}{M}$  in Equation (3) to standardize the function across datasets with varying sizes. This technique is justified because it allows us to create superclusters of visually similar classes without requiring an optimal threshold. By extending equations (2) and (3), the probability of clustering two classes  $A$  and  $B$  can be denoted as  $p(A \sim B)$ .

Fig. 3 provides an example illustrating how the MSL identifies similar classes without relying on a threshold. The DCNN getting confused between the classes can be detected by a high softmax output. When the input image is a truck in Fig. 3, the MSL value for the automobile class is high, indicating their similarity. The automobile class has a high

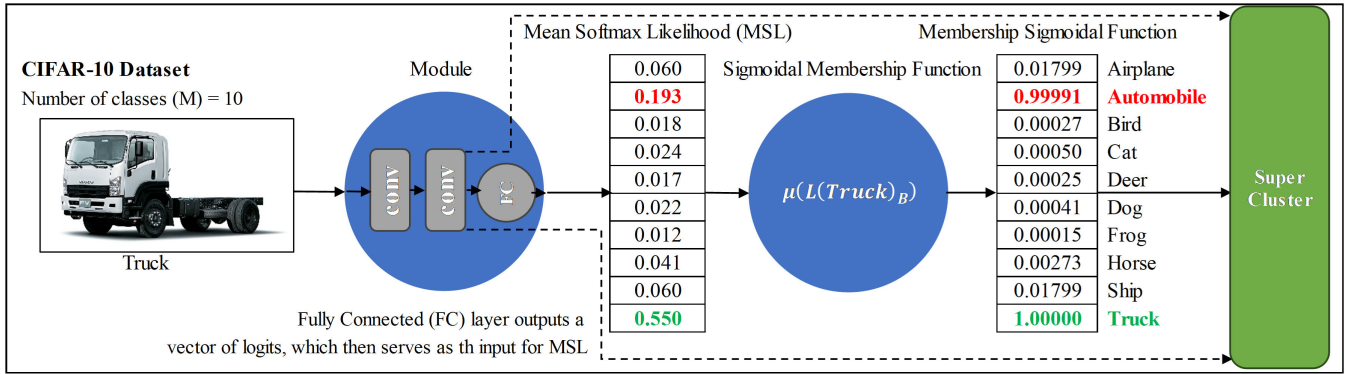


FIGURE 3. Visual similarity measurement using MSL at the root module on CIFAR-10 dataset.

probability (0.99991) of being clustered with the truck class, as given by Equation (4). Frog class, for instance, has a very low (0.000015) chance of being clustered with truck class based on their visual dissimilarity.

**Algorithm 1** Building and Configuring HMDCNN-Tree

**inputs:** Image or output feature map from the parent module

**output:** Trained HMDCNN-Tree structure  $T$

**initialization:**

$T \leftarrow$  Untrained root DCNN  $\triangleright$ Set containing the tree structure

**algorithm:**

$\mathbb{C} \leftarrow$  set of all classes at the root

**while** ( $\exists$  a node in  $T$  containing an untrained DCNN)

1)  $l_{conv} \leftarrow 1$   $\triangleright$ number of conv layers

2) **do**

•  $l_{conv}' \leftarrow l_{conv} - 1$

• Initialize a DCNN  $D_{l_{conv}}$

• Train  $D_{l_{conv}}$  to classify all classes of  $\mathbb{C}$

• **if**  $l_{conv} > 1$  **then** calculate:

$$\Delta EG_{l_{conv}, l_{conv}'} = \frac{VA_{l_{conv}} - VA_{l_{conv}'}}{MS_{l_{conv}} - MS_{l_{conv}'}}$$

•  $l_{conv} \leftarrow l_{conv} + 1$

**while** ( $l_{conv}' == 2 \parallel \Delta EG_{l_{conv}, l_{conv}'} > \tau$ )

3)  $DCNN_{config} \leftarrow D_{l_{conv}'}$   $\triangleright$  Select previous DCNN configuration for module

4)  $SOFTMAX \leftarrow$  Softmax output of  $DCNN_{config} \forall c \in \mathbb{C}$   
 $\triangleright$  Softmax output matrix

5)  $CLUSTER \leftarrow \phi$   $\triangleright$ Track new children's clusters

6) **for each**  $c \in \mathbb{C}$   $\triangleright$ Use the MSL

• Find set  $\mathbb{H}$ , such that  $\mathbb{H} \subset \mathbb{C}$ :

$$\text{prob}(c \sim h) = \mu(SOFTMAX_{[c, h]}), \forall h, c \in \mathbb{H}$$

• Add untrained DCNN corresponding to  $\mathbb{H}$  into  $T$

•  $CLUSTER \leftarrow CLUSTER \cup \mathbb{H}$   $\triangleright$ Add similar classes to the children's set

7) Train  $DCNN_{config}$  to classify between elements of  $CLUSTER$

8)  $T \leftarrow T \cup$  trained  $DCNN_{config}$

9)  $\mathbb{C} \leftarrow$  set of all classes at next node

**return**  $T$

**B. ALGORITHM SELECTING THE SIZE OF CONVOLUTION NEURAL NETWORKS AND HYPER-PARAMETERS**

Algorithm 1 has step (1) through (3) explain how to select the size of the DCNN. It is essential to decide on the configuration and the number of convolution layers for each DCNN. Let  $l_{conv}$  represent the number of convolution layers, and define  $l_{conv}' = l_{conv} - 1$  as the number of previous convolution layers in the DCNN models. In this article, a metric called the "change in efficiency gain" ( $\Delta EG$ ) is used. This metric measures the improvement in efficiency as we move from one model size to another. Mathematically, it is defined as  $\Delta EG_{l_{conv}, l_{conv}'} = \frac{VA_{l_{conv}} - VA_{l_{conv}'}}{MS_{l_{conv}} - MS_{l_{conv}'}}$ , where  $VA$  denotes the validation accuracy achieved when classified the image, and  $MS$  denotes the DCNN's model size.

For each node of the HMDCNN-Tree, the validation accuracy is computed before clustering the children into a super-cluster. The accuracy obtained for classifying all classes in the dataset is used to calculate the  $\Delta EG$  for the root module of an HMDCNN-Tree. In the case of any other module within the HMDCNN-Tree,  $\Delta EG$  is calculated during classifying among all of its children classes, which is a smaller number of classes. Hence, each module may have a different DCNN size.

By using  $\Delta EG$ , we can identify efficient DCNN configurations that are capable of distinguishing classes with low error. This method can achieve a small loss of accuracy when compared to large monolithic DCNNs. The reason lies in the fact that small DCNNs only need to classify among a few clusters of visually similar classes rather than all classes in the dataset.

Enhancing the depth of a DCNN is widely recognized to be frequently more effective than increasing its width in order to achieve improved accuracy [9]. The result is that  $DCNN_{config}$  starts with only one convolutional layer ( $l_{conv}$ ) and incrementally adds one layer at a time. Until the validation accuracy ( $VA$ ) saturates, each DCNN configuration is trained, and then ( $\Delta EG$ ) is calculated.

The choice of  $\tau$  is a crucial hyperparameter in the construction of the HMDCNN-Tree. Choosing a higher  $\tau$  leads to the creation of taller HMDCNN-Trees, featuring

numerous small DCNN modules, each with a limited number of layers. However, these small modules may face challenges effectively distinguishing between classes, leading to the creation of a limited number of super-clusters under each parent and the formation of tall HMDCNN-Trees.

On the contrary, opting for a smaller  $\tau$  produces short and wide hierarchies featuring large DCNNs, with each module comprising numerous convolutional layers. In such hierarchies, the DCNNs exhibit similarities to monolithic DCNNs, potentially harboring redundant components. Therefore, selecting an appropriate value for  $\tau$  becomes crucial to mitigate these potential drawbacks.

### C. DETECTING SUPER-CLUSTERS AND CONSTRUCTING A HIERARCHY

In this section, we clarify the procedures outlined in steps (4) through (7) of Algorithm 1, focusing on the identification of super-clusters and the construction of the hierarchy. The hierarchy begins with the creation of the root. Once the size of the DCNN module is determined, the MSL is used to assess the similarity among all classes of the dataset, establishing the initial level of super-clusters (children of the root module). This process recurs for every root child, which serves as a super-cluster. Here, the module's DCNN size is chosen, MSL is used to identify class similarities and construct super-clusters, and the module is subsequently trained to classify among the newly formed children super-clusters.

The softmax output matrix for the trained DCNN is acquired in step (4). Subsequently, in step (6), the MSL serves as the similarity metric to cluster classes. If new child super-clusters are formed, step (7) involves retraining the module to classify between these newly created super-clusters. An illustrative example of how the HMDCNN-Tree is constructed can be found in Fig. 4. This process iterates until all modules associated with super-clusters are appropriately trained.

## IV. EXPERIMENTAL RESULTS

In the experiments, we evaluate the results, effectiveness of the proposed method and compare it with monolithic DCNN.

### A. DATASET USED

In our experiments, we utilize several image datasets, including CIFAR-10 [19], SVHN [20], and EMNIST [21]. These datasets contain centered and fixed-size images, each containing only one object. As shown in Table 4, CIFAR datasets consist of color images with 3 color channel and dimensions of 32 pixels in height (H) and  $32 \times 3$  pixels in width (W), with CIFAR-10 containing images from 10 different classes. Respectively, the training set and test set consist of 50,000 and 10,000 images. We adhere to the standard practice of creating a validation set by utilizing 5,000 images from the training set. The SVHN dataset includes 73,257 images, each having 3 color channels ( $C = 3$ ), and a size of 32 pixels in width (W) by 32 pixels in height (H) in the training set. Additionally, there are 531,131 images

TABLE 4. Dataset specifications for the experiments.

Dataset	Image Size ( $W \times H \times C$ )	No. of Training Images	No. of Test Images	No. of Classes
CIFAR 10	$32 \times 32 \times 3$	50,000	10,000	10
EMNIST	$28 \times 28 \times 1$	112,800	18,800	47
SVHN	$32 \times 32 \times 3$	604,388	26,032	10

available for further training. When presenting the results of the SVHN dataset, we adhere to the common approach of using the entire training data without applying any data augmentation. The SVHN dataset containing 6,000 images is used to validate the training results. The EMNIST dataset is an extension of the well-known MNIST dataset. Among the six configurations of the EMNIST dataset, we specifically utilize the EMNIST-Balanced configuration. This dataset includes 131,600 images, each having 1 color channels ( $C = 1$ ), and a size of  $28 \times 28$  pixels, distributed across 47 classes.

### B. EXPERIMENTAL SETUP

After constructing and training the HMDCNN-Tree for each dataset, the classification accuracy is assessed using the respective test set. The utilization of the test set aims to mitigate the impact of overfitting. The memory requirements and number of operations, million multiply-accumulate (MMac), are determined using the `torchsummary` and `thop` PyTorch libraries, respectively. Training for all modules of the HMDCNN-Tree is conducted using the ADAM learning rule. A uniform batch size of 200 is employed across all datasets with a total of 150 epochs. The initial learning rate is set at 0.01 and is reduced by a factor of 10 at 50% and 75% of the total number of epochs.

As each module consist of a compact DCNN, the number of parameter is significantly reduced compared to large DCNNs, effectively mitigating the risk of overfitting. Small DCNNs exhibit high accuracy without the need for major hyperparameter tuning. In our workflow, we leverage the computing power of either Google Colab or Kaggle GPU runtime environments for the purpose of training the HMDCNN-Tree and obtaining a well-trained model. The utilization of these platforms allows us to harness the advantages of their powerful Graphics Processing Unit (GPU) resources, which significantly accelerates the training process compared to conventional CPU-based computations. Google Colab and Kaggle provide convenient and accessible cloud-based environments that facilitate the training of complex DCNN models by offering GPU support as part of their runtime options. This strategic combination of Colab and Kaggle resources ensures efficient model training, enabling us to achieve optimal performance and accuracy in our DCNN tasks. The source code of our works are available in GitHub.<sup>1</sup>

<sup>1</sup><https://github.com/adeclintonsitepu/HMDCNN-Tree>

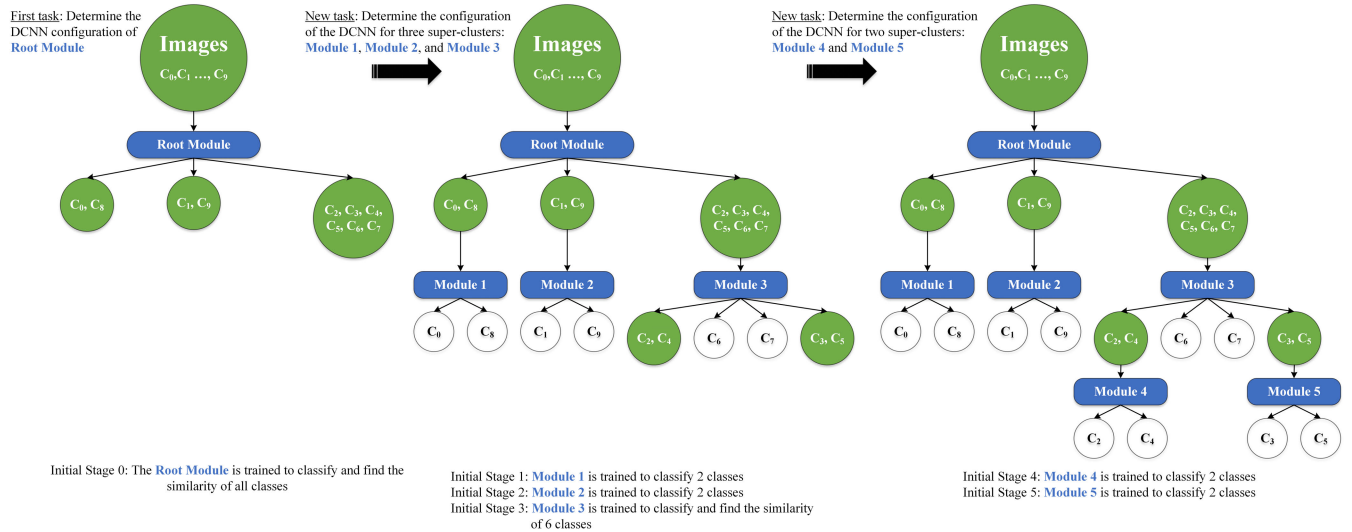


FIGURE 4. An example of constructing and training the HMDCNN-Tree on the CIFAR-10 dataset.

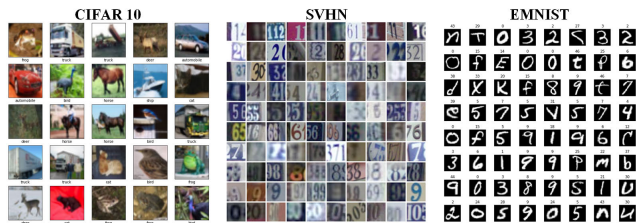


FIGURE 5. Selected images from the different datasets used in the experiments.

C. DCNN CONFIGURATION AND ARCHITECTURE ANALYSIS

The initial setup commences with a single convolutional layer and incrementally raises the number of layers by one in each iteration. Training continues for each DCNN configuration until the validation accuracy (VA) reaches a saturation point, following which  $\Delta EG_{lconv, lconv'}$  is calculated. The hyperparameter  $\tau = 0.1$  is used for evaluating  $\Delta EG_{lconv, lconv'}$  between two consecutive DCNN models. If  $\Delta EG_{lconv, lconv'}$  is lower than this specified threshold (marked with an underline in Table 5), the accuracy improvement for a deeper DCNN is minimal. We fix the depth of the DCNN based on the count of  $lconv'$  layers. As an illustration, consider the CIFAR-10 dataset in Table 5, where the module has a single layer, resulting in a model size of 87.0 KB and an accuracy of 57%. When one more layer is added, the model size rises to 108.5 KB, and the VA reaches 62%. Consequently,  $\Delta EG_{21} = \frac{62-57}{108.5-87.0} = 0.233$ . At current calculation, which five layers are used ( $lconv = 5$ ), so  $\Delta EG_{54} = \frac{90-89.5}{349.1-230.4} = 0.004$ . As  $\Delta EG_{54} < \tau = 0.1$ , the improvement in accuracy is minimal compared to the growth in model size. Therefore, a DCNN with four layers ( $lconv' = 4$ ) is applied at the root module for the CIFAR-10 dataset. For every module in

TABLE 5. Calculation for DCNN configuration at the root for each dataset.

Dataset	Size (KB)	Val Acc. (%)	$\Delta EG$	No. Layer
CIFAR10	87.040	57.0	NA	1
SVHN	87.040	86.5	NA	
EMNIST	105.472	75.0	NA	
CIFAR10	108.544	62.0	0.233	2
SVHN	108.544	89.3	0.130	
EMNIST	164.864	84.5	0.160	
CIFAR10	178.186	78.0	0.230	3
SVHN	178.186	96.9	0.109	
EMNIST	366.592	84.9	0.002	
CIFAR10	230.400	89.5	0.220	4
SVHN	230.400	97.0	0.002	
EMNIST	574.464	85.2	0.001	
CIFAR10	349.184	90.0	0.004	5
SVHN	339.978	97.2	0.002	
EMNIST	784.384	85.4	0.001	
CIFAR10	936.960	91.0	0.002	6
SVHN	931.840	98.1	0.002	
EMNIST	1,382.400	85.6	0.000	

the HMDCNN-Tree, the DCNN configuration is determined, leading to diverse DCNN sizes across modules.

D. THE CLUSTERS AND HMDCNN-TREE RESULT

The HMDCNN-Tree for CIFAR-10 starts with three super-clusters as shown in Fig. 7. The DCNN configuration at the root is trained to classify input images using Fig. 6 configuration. In super-cluster 1, the initial classes of CIFAR-10 are clustered into Automobile and Truck, super-cluster 2, the initial classes of CIFAR-10 are clustered into Ship, Airplane, and super-cluster 3, the initial classes of CIFAR-10 are clustered into Bird, Dog, Cat, Frog, Deer and Horse.

The size of the DCNN is individually calculated for every module in the HMDCNN-Tree, resulting in varied sizes among modules. The DCNN architecture for each



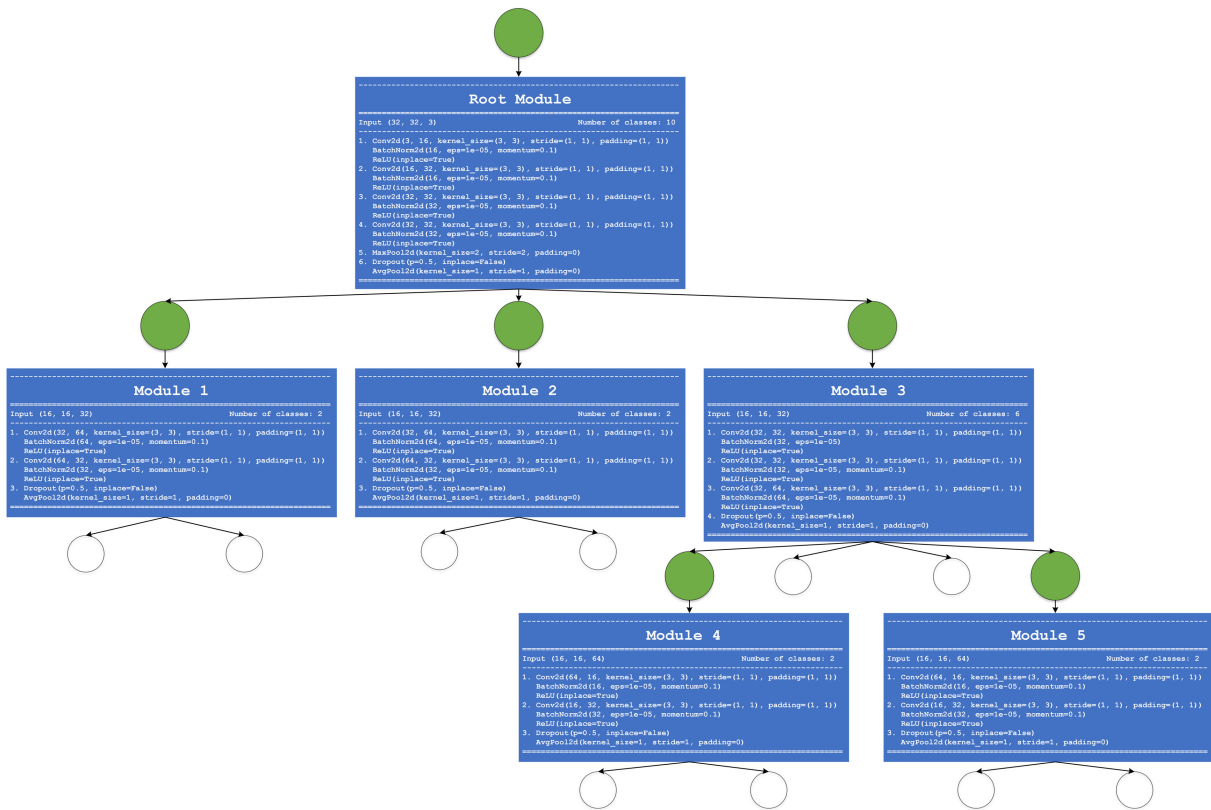


FIGURE 6. The configuration of the architecture within each module of the HMDCNN-Tree for the CIFAR-10 dataset.

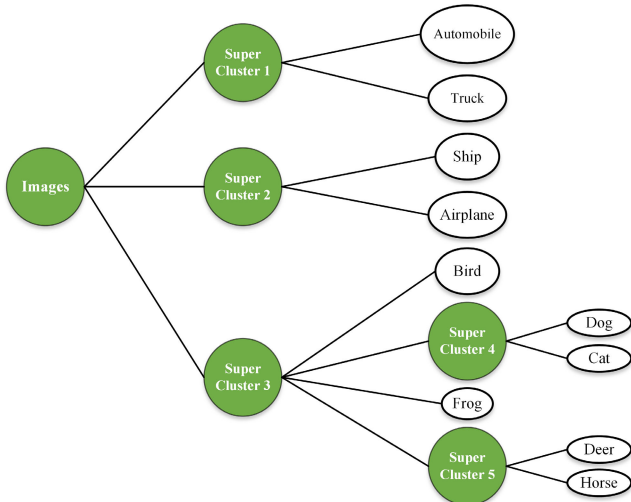


FIGURE 7. The HMDCNN-Tree architecture achieved for the CIFAR-10 dataset.

module of the HMDCNN-Tree in the CIFAR-10 dataset is showed in Fig 6. The  $\Delta EG$  metric with a threshold of  $\tau = 0.1$  is used to derive these architectures. The root module receives input images sized  $32 \times 32 \times 3$ , with 32 pixels in height and width and three color channels (R, G, B). The architecture here consists of four convolutional layers and one max-pooling layer, which together form the feature

map. The decision between the children is determined at the conclusion of the fully connected (FC) layer, offering six possibilities: two super-clusters and four individual classes. Based on the output of the root module, the child module is determined, using the feature map of dimensions  $16 \times 16 \times 32$ . Due to the small size of each DCNN, many hyperparameters, channel count, kernel size, weight decay, batch normalization, and layer width, can remain consistent without the need for extensive fine-tuning to achieve high accuracy.

It should be noted that the child node receives input from the output of the parent node, which is a feature map value. This guarantees that the computations carried out by the parent are not duplicated in the child. Moreover, through the reuse of the feature map, the child module functions as a specialized extension of the parent. Consequently, every pathway in the HMDCNN-Tree operates as if it were an independent DCNN equipped with multiple layers. This clarifies why the lower modules in the HMDCNN-Tree utilize small DCNNs, even as they classify between classes with visual similarities.

E. TRAINING

Training for each module in the HMDCNN-Tree follows the traditional back-propagation algorithm, starting with the root module. The training of child modules within the HMDCNN-

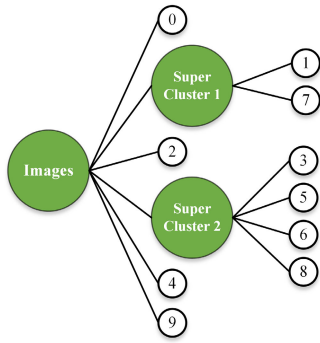


FIGURE 8. The HMDCNN-Tree architecture achieved for the SVHN dataset.

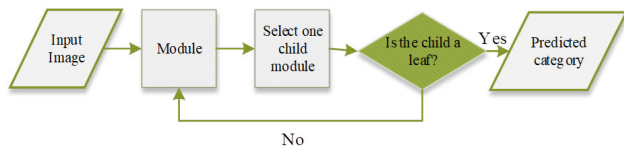


FIGURE 9. Image classification with the HMDCNN-Tree.

Tree includes the utilization of feature maps derived from correctly classified images by the parent module, ensuring relevance to lower modules and their respective sub-trees. Additionally, it guarantees that the training errors in a higher-level module don't impact the training of its lower-level descendants. We explore two approaches to train the HMDCNN-Tree. The method involves the calculation of loss at the leaf nodes, followed by the application of back-propagation through all nodes of the tree to minimize this loss. In the work by Roy and Todorovic [22], back-propagation is used individually in each module. We choose this approach due to the independence of modules at a given depth (sibling modules). By adopting this approach, we can train modules simultaneously, leading to a reduction in the overall training time. With fewer parameters, small DCNNs are less possible to overfitting, allowing them to attain high accuracy with minimal hyperparameter tuning. The computational cost associated with training small DCNNs is lower, given their considerably reduced number of parameters. This guarantees that the HMDCNN-Tree is constructed and trained quickly.

F. CLASSIFYING IMAGES USING THE HMDCNN-TREE

In the image classification procedure with the HMDCNN-Tree, as illustrated in Figure 9, the root module is the first to handle the image, making a selection from among its children. The selected child module receives the feature map from the root as input, and this procedure is iterated until reaching a leaf in the HMDCNN-Tree. The leaves within the HMDCNN-Tree represent the initial classes present in the dataset. At each step, each module chooses only a single child, resulting in substantial pruning of the HMDCNN-Tree, and consequently, a significant reduction in the number of operations.

TABLE 6. Investigating variations in test error and number of operations across various datasets and techniques.

Dataset	Method	MMac	Size (MB)	Test Error (%)
CIFAR-10	VGG-16 [2]	313	76.6	6.7
	DenseNet-121 [23]	59	99.6	7.0
	MobileNet V2 [24]	101	8.6	6.0
	*HMDCNN-Tree	<b>28</b>	<b>0.8</b>	<b>7.9</b>
SVHN	DenseNet-121 [23]	59	99.6	1.7
	Wide ResNet-16,4 [26]	1,935	10.7	1.6
	*HMDCNN-Tree	<b>30</b>	<b>0.5</b>	<b>1.8</b>
EMNIST	EDEN [25]	23	-	11.7
	Supervised SNN [27]	18,700	-	14.4
	*HMDCNN-Tree	<b>4</b>	<b>0.3</b>	<b>7.8</b>

Let's examine the results depicted in Fig. 8. The root module performs classification among super cluster-1 and super cluster-2. After making a classification, the output feature map of the root, which includes partially processed data, is then transmitted to one of the children. The activation of the module, which classifies 3, 5, 6, and 8, occurs when super cluster-2 is selected. The repetition of this procedure occurs sequentially at all levels of the tree until it reaches a leaf module. Optimized accuracy is achieved in the HMDCNN-Tree because the input for the child module comes from the output feature map of the parent module.

The HMDCNN-Tree ensures that, at any given time, only a singular module is loaded into memory. Once the module has used its purpose, the memory is freed up for use by its corresponding child module. This decreases the overall memory utilization on the device while executing the HMDCNN-Tree. Furthermore, because only some of the modules is used during inference, the total memory loading is considerably decreased compared to monolithic DCNN architectures.

G. COMPARISONS WITH TRADITIONAL EVOLUTIONARY DCNNs

In our evaluation, we examine how the HMDCNN-Tree architecture compared to several monolithic DCNN architectures. In Table 6, a comparison between the HMDCNN-Tree and existing monolithic DCNNs is presented, covering various metrics. In terms of memory usage or computational operations, the HMDCNN-Tree outperforms the other models. These improvements in performance are achieved with minimal impact on the test error. The VGG [2] and ResNet [3] DCNNs, are composed of 16 and 54 layers, respectively. We also evaluate DenseNet [23], which integrates group convolutions to improve parameter efficiency, in our comparison. We also compare HMDCNN with Wide ResNet-16,4 [26]. In the context of Wide ResNet, the notation Wide ResNet- $x, y$  is the architecture which has  $x$  number of layers with  $y$ . Here, the growth rate serves as a hyperparameter determining the size of each layer in the Wide ResNet. In the case of the EMNIST dataset, we utilize Supervised SNN [27] for comparative analysis. Additionally, we conduct comparisons with EDEN [25].

As shown in Table 6, the HMDCNN-Tree exhibits the smallest number of operation. In comparison to VGG-16 on CIFAR-10, the HMDCNN-Tree requires a model that is 99.90% smaller ( $1 - \frac{28}{28910} = 0.9990$ ). Likewise, when HMDCNN-Tree is compared with DenseNet on SVHN, a decrease of 49.15 in MMac (million multiply-accumulate operations) is observed ( $1 - \frac{30}{59} = 0.4915$ ). Lower MMac can be considered more efficient. MMac is a measure of computational efficiency, and a lower value indicates that fewer multiply-accumulate operations are required, which can lead to faster inference and reduced computational cost. Reduced memory accesses, faster inference, and lower energy consumption are associated with smaller MMac.

The HMDCNN-Tree demonstrates the lowest error, with error rate at 7.8%, on the EMNIST dataset. Additionally, its accuracy compares favorably with state-of-the-art results for SVHN and CIFAR-10 datasets, despite employing fewer MMac operations.

## V. CONCLUSION

This article introduces a novel approach to improve the processing throughput of DCNNs on low-resource platforms. Unlike traditional evolutionary algorithms, our approach partitions hierarchical DCNNs into sets of layers with balanced loads and decreased communication costs. We observed that building the tree and running the training stage required more effort because the code is split depending on the number of clusters created from each cluster. Unlike monolithic DCNNs, where the architecture undergoes training in a single code program, training our architecture necessitates separate code programs due to the different sizes of DCNN modules. Our method offers the best trade-off between accuracy and the complexity of the architecture when compared against a monolithic DCNN. The structure of the HMDCNN-Tree incorporates multiple small DCNNs, denoted as modules, organized in a tree-like fashion, collaborating to classify an image. In contrast to traditional monolithic DCNN architectures, the HMDCNN-Tree architecture strategically uses modules, minimizing redundant computation and memory access. This allows the HMDCNN-Tree to function with markedly reduced memory demands and fewer operations, with only minimal impact on classification accuracy.

However, our current experiments focused on datasets with classes that have distinct visible features, making them relatively easy to separate. This simplification was necessary to establish the baseline effectiveness of our proposed architecture. We acknowledge that in real-world applications, classes often have similar features, which poses a more significant challenge for classification algorithms. To address the limitations identified in this research, we plan to extend our research to include classes with similar features. This involves implementing data augmentation to enhance the dataset, ensuring a more balanced analysis and improving the robustness of our model when dealing with visually similar classes. Additionally, the HMDCNN-Tree architecture will be adapted to handle the increased complexity associated

with similar-looking classes. This adaptation may involve developing new modules or enhancing existing ones to better capture subtle differences between classes. Another strategy includes conducting extensive experiments focused on evaluating the performance of the HMDCNN-Tree on these more challenging datasets.

## REFERENCES

- [1] A. C. Sitepu and C.-M. Liu, "Efficient computer vision inference using modular neural network techniques," in *Proc. VTS Asia-Pacific Wireless Commun. Symp. (APWCS)*, Aug. 2023, pp. 1–2, doi: [10.1109/apwcs60142.2023.10234066](https://doi.org/10.1109/apwcs60142.2023.10234066).
- [2] L. Tang, "Image classification based on improved VGG network," in *Proc. IEEE 6th Int. Conf. Signal Image Process. (ICSIP)*, Nanjing, China, Oct. 2021, pp. 316–320, doi: [10.1109/ICSIP52628.2021.9688778](https://doi.org/10.1109/ICSIP52628.2021.9688778).
- [3] C. Zhang, P. Benz, D. M. Argaw, S. Lee, J. Kim, F. Rameau, J.-C. Bazin, and I. S. Kweon, "ResNet or DenseNet? Introducing dense shortcuts to ResNet," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2021, pp. 3549–3558.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.
- [5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [6] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
- [7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.
- [8] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inceptionv4, inception-ResNet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, vol. 31, no. 1, pp. 4278–4284.
- [9] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
- [10] N. Manakitsa, G. S. Maraslidis, L. Moysis, and G. F. Fragulis, "A review of machine learning and deep learning for object detection, semantic segmentation, and human action recognition in machine and robotic vision," *Technologies*, vol. 12, no. 2, p. 15, Jan. 2024.
- [11] S. Aslam and A. B. Nassif, "Deep learning based CIFAR-10 classification," in *Proc. Adv. Sci. Eng. Technol. Int. Conferences (ASET)*, Dubai, United Arab Emirates, Feb. 2023, pp. 1–4, doi: [10.1109/aset56582.2023.10180767](https://doi.org/10.1109/aset56582.2023.10180767).
- [12] S. Dahiya, R. Tyagi, and N. Gaba, "Comparison of ML classifiers for image data," *EasyChair, Tech. Rep.* 3815, 2020.
- [13] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *Social Netw. Comput. Sci.*, vol. 2, no. 3, p. 160, May 2021.
- [14] M. Znalezniak, P. Rola, P. Kaszuba, J. Tabor, and M. Śmieja, "Contrastive hierarchical clustering," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2023, pp. 627–643.
- [15] M. Amer and T. Maul, "A review of modularization techniques in artificial neural networks," *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 527–561, Jun. 2019.
- [16] W. Li, M. Chu, and J. Qiao, "Design of a hierarchy modular neural network and its application in multimodal emotion recognition," *Soft Comput.*, vol. 23, no. 22, pp. 11817–11828, Nov. 2019.
- [17] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3280–3296, Sep. 2022, doi: [10.1109/TSE.2021.3087402](https://doi.org/10.1109/TSE.2021.3087402).
- [18] P. Witold, "An introduction to computing with fuzzy sets," in *Proc. IEEE ASSP Mag.*, vol. 190, 2021, pp. 225–247.
- [19] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Tech. Rep., 2009.
- [20] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NeurIPS*, 2011, pp. 1–9.

- [21] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2921–2926.
- [22] A. Roy and S. Todorovic, "Monocular depth estimation using neural regression forest," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5506–5514.
- [23] B. Chen, T. Zhao, J. Liu, and L. Lin, "Multipath feature recalibration DenseNet for image classification," *Int. J. Mach. Learn. Cybern.*, vol. 12, no. 3, pp. 651–660, Mar. 2021.
- [24] M. Ayi and M. El-Sharkawy, "RMNv2: Reduced mobilenet V2 for CIFAR10," in *Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Las Vegas, NV, USA, Jan. 2020, pp. 0287–0292, doi: [10.1109/CCWC47524.2020.9031131](https://doi.org/10.1109/CCWC47524.2020.9031131).
- [25] E. Dufourq and B. A. Bassett, "EDEN: Evolutionary deep networks for efficient machine learning," in *Proc. Pattern Recognit. Assoc. South Afr. Robot. Mechatronics (PRASA-RobMech)*, Nov. 2017, pp. 110–115.
- [26] M. S. Hanif and M. Bilal, "Competitive residual neural network for image classification," *ICT Exp.*, vol. 6, no. 1, pp. 28–37, Mar. 2020.
- [27] R. Vailla, J. Chiasson, and V. Saxena, "A deep unsupervised feature learning spiking neural network with binarized classification layers for the EMNIST classification," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 6, no. 1, pp. 124–135, Feb. 2022, doi: [10.1109/TETCI.2020.3035164](https://doi.org/10.1109/TETCI.2020.3035164).
- [28] W. Zhang, L. Zhou, P. Zhuang, G. Li, X. Pan, W. Zhao, and C. Li, "Underwater image enhancement via weighted wavelet visual perception fusion," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 4, pp. 2469–2483, Apr. 2024, doi: [10.1109/TCSVT.2023.3299314](https://doi.org/10.1109/TCSVT.2023.3299314).
- [29] W. Zhang, W. Zhao, J. Li, P. Zhuang, H. Sun, Y. Xu, and C. Li, "CVANet: Cascaded visual attention network for single image super-resolution," *Neural Netw.*, vol. 170, pp. 622–634, Feb. 2024.
- [30] W. Zhang, Z. Li, G. Li, P. Zhuang, G. Hou, Q. Zhang, and C. Li, "GACNet: Generate adversarial-driven cross-aware network for hyperspectral wheat variety identification," *IEEE Trans. Geosci. Remote Sens.*, vol. 62, 2024, Art. no. 5503314, doi: [10.1109/TGRS.2023.3347745](https://doi.org/10.1109/TGRS.2023.3347745).
- [31] C. Wang, S. Dong, X. Zhao, G. Papanastasiou, H. Zhang, and G. Yang, "SaliencyGAN: Deep learning semisupervised salient object detection in the fog of IoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2667–2676, Apr. 2020, doi: [10.1109/TII.2019.2945362](https://doi.org/10.1109/TII.2019.2945362).
- [32] D. Zhang, D. Meng, and J. Han, "Co-saliency detection via a self-paced multiple-instance learning framework," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 5, pp. 865–878, May 2017, doi: [10.1109/TPAMI.2016.2567393](https://doi.org/10.1109/TPAMI.2016.2567393).



natural language processing (NLP), and neuromorphic computing.

**ADE CLINTON SITEPU** received the master's degree from the Department of Computer Science, Universitas Potensi Utama, Indonesia, in 2021. Currently, he is pursuing the Ph.D. degree in computer science with the National Taipei University of Technology (Taipei Tech), Taiwan. He joined the Applied Computing Laboratory, in 2021. His research interests include machine learning, deep neural networks optimization technique, especially in data science includes visual recognition,



**CHUAN-MING LIU** (Member, IEEE) received the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, USA, in 2002. In 2010 and 2011, he held visiting appointments with Auburn University, Auburn, AL, USA, and Beijing Institute of Technology, Beijing, China. He is currently a Professor with the Department of Computer Science and Information Engineering, National Taipei University of Technology (Taipei Tech), Taiwan. In addition to his association with several journals, conferences, and societies, he has published more than 100 papers in numerous prestigious journals and international conferences. His research interests include big data management and processing, uncertain data management, data science, spatial data processing, data streams, ad hoc and sensor networks, and location-based services.

• • •