




Confusion matrix-based modularity induction into pretrained CNN

Salman Ahmad¹ · Shahab U. Ansari¹ · Usman Haider¹  · Kamran Javed² · Jalees Ur Rahman¹ · Sajid Anwar¹

Received: 20 June 2021 / Revised: 5 January 2022 / Accepted: 18 January 2022 /
Published online: 18 March 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Structurally and functionally, the human brain's visual cortex inspires convolutional neural networks (CNN). The visual cortex consists of different connected cortical regions. When a cortical area receives an input, it extracts meaningful information and forwards it to its neighboring region. CNN imitates the hierarchical structure of the visual cortex by multiple feature extraction layers. In neurosciences, it is believed that the modular structure of the human brain is the source of its cognitive abilities. This work contributes to the problem of domain decomposition, information routing control in the network, and module integration for image classification by proposing a novel framework to induce modularity in a pretrained CNN. We decompose the input domain of the CNN by employing novel Confusion Matrix driven Centroid Based Clustering (CMCBC) to create functional modules comprised of different pathways. CMCBC is an unsupervised clustering technique that utilizes the k-Medoid algorithm. This approach uses a confusion matrix to find similarities between each pair of classes and medoid for every cluster instead of using a distance function. The proposed framework is evaluated on two benchmark datasets, MNIST and CIFAR10, and the results achieved are promising. On the MNIST dataset, we achieved 98.51% accuracy using our proposed Modular CNN compared to the baseline accuracy of 99.39%. But at the same time, we saved 53% multiplications in the network, which significantly reduced the complexity. Similarly, on the CIFAR10 dataset, our model achieves 78.01% accuracy, 6% less than the baseline accuracy (84%). But when we retrain the network to align the weights further, our model outperformed the baseline model accuracy by 2.78% and achieved 86.78% accuracy.

Keywords CNN · Pruning · Confusion matrix · Modularity · Clustering

1 Introduction

Convolutional neural network (CNN) is state-of-the-art for a wide range of computer vision problems such as object recognition, image classification, and semantic segmentation [28].

Sajid Anwar deceased

✉ Usman Haider
usman.haider@giki.edu.pk

Extended author information available on the last page of the article.

Moreover, CNN has also been proven effective for natural language processing and forecasting problems [12, 18]. CNN is a subclass of representation learning techniques or algorithms. The output of each hidden layer in the network is considered a representation of the original data. CNN constantly changes the presentation of data by modifying its weights throughout its learning process. In general, a system is said to be modular if it can be divided into several independent subsystems, called modules. A modular convolutional neural network (MCNN) is a CNN that incorporates the concepts and practices of modular design. Although the modular design is achieved with an additional cost, it is still preferred over a monolithic system. Each module in a modular system is designed to solve an isolated sub-problem. This structure of loosely coupled modules coordinating with each other enhances the system's fault tolerance. The modular design also facilitates parallelism and scaling up the system to add additional functionality with no interference to the existing functions. In addition, to facilitate scaling up the system, modularity also assists in the reallocation of the functional units to new tasks [41].

Major concerns in deep learning models are latency and resource consumption. In literature many techniques have proposed for deep model compression [6, 16, 17, 25]. Modhej et al. [31] proposed a novel approach based on the computational function of the dentate gyrus of the hippocampus for pattern separation. One of the prominent features of the proposed network is the employment of two excitation steps and two inhibition steps to activate or deactivate a node. The numerical simulation results indicate that the proposed network requires fewer training iterations to achieve comparable accuracy because weak nodes are silenced in different steps of the proposed network.

Practically, deep neural network-based solutions have a high computational cost and high power consumption, making it challenging for real-time applications. Studies have shown that a trained deep neural network (DNN) does not require all its weights to perform a task [20]. It has been proved that removing unwanted weights can reduce the computation cost of neural nets and improve their performance. The process of compressing neural networks by removing one or more of their core structural elements is called pruning. Different pruning algorithms imply other selection criteria that rank network elements intended to be pruned. The pruning can be done at runtime or offline.

Runtime pruning allows temporarily pruning the network for a single iteration and restoring it to its original state for the next iteration. In combination with proper domain decomposition, it can induce modularity in a CNN. Modular CNN has more resemblance with biological neural networks than monolithic CNN. Furthermore, the modular structure of CNN helps in understanding the overall working of the network and can learn additional tasks without damaging the already known information. This behavior encourages using a pre-trained network for learning even a heterogeneous task while retaining the old data. Our contributions in this paper are summarized as follow:

- We induced modularity in a pre-trained CNN model by utilizing the information learned by the network during training.
- Based on learned knowledge, we decompose the input domain.
- The decomposed input domain is utilized to achieve modularity.

Based on the literature, we hypothesize that the learned knowledge of a network provides us with enough information that can be used for input domain decomposition. Furthermore, the domain decomposition can raise modularity in a neural network if supported by an information routing control mechanism.

The rest of the paper is organized as follows: the literature review is discussed in Section 2, in Section 3, we discuss the proposed methodology in detail, then in Section 4, we discuss experimental setup and results, and finally, in Section 5, we conclude our findings.

2 Literature review

Modularity in ANN has been a center of interest for researchers since the 1980s [43]. The next decade follows the trend, and several fundamental techniques for MNN have been introduced for various machine learning problems [15, 21, 33]. A modularization technique can contribute to one or more of the mentioned categories. Domain refers to the input domain of an MNN, which defines the problem addressed. Topology is the overall architecture of an MNN where formation corresponds to path selection for input inside an MNN. It is the process that helps to attain modularity in the network. However, integration algorithms integrate the output of different modules that contribute to the network's final decision.

The domain consists of all the data that a neural network process and learn from to generalize unseen data. It defines the problem that needs to be addressed. Domain modularization is based on the rationale that a problem can be divided into pieces, each acting on a separate subdomain. Consequently, a module in MNN that is constructed by other techniques at different modularization levels can process one of the subspaces instead of the entire domain.

Dimensional domain partitioning is another type that occurs per data sample. In this type of partitioning, a data instance is decomposed based on its features. A different set of features are assigned to various modules of an MNN. For example, in [11] dimensional domain partitioning is used by applying additional filters and processing the output by other modules. Manual domain partitioning is done by partitioning the domain in multiple subspaces based on some analytical solution or expert knowledge. Each subspace in the domain corresponds to different subproblems. Therefore, manual domain partitioning methods for one problem do not necessarily work for another problem. Recently, the age recognition algorithms based on facial features have been proposed [4, 5]. For example, a survey on feature selection methods for face recognition problems in [8] concludes that feature wise manual decomposition of a face dataset is analytically not feasible.

Learned domain partitioning used learning algorithms for data partitioning. It clusters data based on complex representations usually invisible to experts due to complex mathematical structures or incomplete understanding of the problem. In literature, different learning algorithms are used for clustering input domains. For example, [48] use a three-factor selection method, [34] use fuzzy clustering and [46] utilize Divide and Conquer Learning (DCL) to cluster input domain.

The network's topology is the connectivity between different nodes and modules that produce the overall structure of a network. Modularity in the topology of a network creates clusters of nodes called modules. Nodes inside a module are densely connected while sparsely connected with nodes of other modules. The intra-module and inter-module connectivity patterns define modularity in a modular structure. Structural modularity in deep learning also provides a solution for the problems in monolithic neural network architectures, such as overfitting and vanishing gradient problems, to name a few. In [45] authors show that inducing structural modularity in a neural network improves the generalization error of the network. Moreover, the work in [45] produces modularity in CNN by hierarchical clustering the feature vectors of the hidden layer and obtaining a short path for gradient

flow in the backpropagation phase. Similarly, effects of modularity are also reported in [35] where the flow of information in the network is controlled by specialized units called gating units. The authors refer to their proposed network architecture as a highway network. Another widely used group of modularization techniques in neural networks is repeated block. Recurrent neural networks (RNN) [30] is a well-known architecture that uses the repeated blocks method. An RNN has several long short-term memory (LSTM) [26]. In sequential topology, the structural units of architecture consist of whole modules. The very famous Inception networks [37] and Xception networks [9] are multi-path topology-based networks composed of sequentially arranged convolution modules. The Highway networks described in [35] are also built around sequential composition principles.

Formation in modular neural networks refers to the process that generates modularity by path selection for input inside a modular network. In a given set of available modules in a network, the formation technique selects that will process the input. A formation can either be manual or automatic. In manual formation, human experts utilize heuristics and intuition to define modularity in a network. Automated techniques use evolutionary and machine learning algorithms for formation. Authors in [24] effectively use connection cost minimization as a formation technique to produce a modular network. It is concluded in the mentioned work that adding connection cost to HyperNEAT network and its variations [42] results in a significant improvement in its performance and its modular structure as well. Moreover, evolutionary algorithms are used to combine networks for knowledge transfer in [7]. Dropout [36], a widely used technique for regularization in neural networks, is an implicit formation technique that randomly drops nodes during the learning process. Drop-Circuit [32] is also a dropout technique used in parallel circuits, which is a multipath neural network.

Integration defines how the final output from different modules is calculated. It could be either cooperative or competitive. Only one out of all the modules' output is selected to contribute to the final output in competitive integration. However, cooperative integration leads to the contribution of all the modules to the final output. To the best of our knowledge, integration techniques combine modules output by arithmetic logic or learning algorithms. [19] use competitive integration with a multi-network architecture. Similarly, in [44] the output of two CNN networks is integrated logically for character recognition where one CNN detects characters in an image, and the other recognizes the detected characters. Integration through learning algorithms provides an optimal combination of modules with the best overall performance. For example, [29] utilizes fuzzy logic to integrate different neural networks of the MNN trained for the image recognition task. Neural networks are evaluated by a fuzzy inference system and integrated the output by Sugeno integral. Likewise, a work proposed in [1] adds new modules to a pre-trained network to achieve transfer learning. A similar approach for integration is also adapted in [40] where the network is trained for multi-task learning by adding a new module.

Researchers have proposed various ways to accelerate inference with deep convolutional networks in the literature. These methods speed up convolutions without critical degradation of the accuracy of the models, which are Factorization and Decomposition of convolution's kernels [39], Separable Convolutions [10], and Pruning [2]. Researchers are also proposing different techniques and algorithms for accelerating the CNNs, which include [27, 50].

Zhao et al. [50] presented a technique that is based on the Strassen algorithm and Winograd minimal algorithm for filtering. They showed a theoretically colossal reduction in computational complexity and also, in practice, proposed algorithms providing optimal performance. Still, the problem is that these are very expensive implementation-wise and

cannot run on embedded devices and in real-time systems. Also, it increased parallelizing difficulty for an acceleration of hardware. New classes of fast techniques for CNN are introduced by [27] which used minimal Winograd filtering algorithms. The method used small tiles, which reduced computational complexity.

Spatial separable convolutions are also used later on in [23], MobileNets, followed by [49], which introduced highly computationally efficient CNN architecture known as ShuffleNet. It is specially designed for mobile devices and requires limited computing power of 10–15 MFLOPs. Zhang et al. [49] used new operations of channel shuffle and point-wise group convolutions. ShuffleNet achieved a speedup of $13\times$ over AlexNet without affecting the accuracy. They lowered top-1 error to 7.8% absolute than [23] on ImageNet classification problem. Another slimmer model was introduced specially for mobile phones and embedded systems [14], known as EffNet. The EffNet outperformed MobileNet and ShuffleNet in accuracy and computational burden.

Pruning is another promising technique for highly efficient CNN keeping low complexity. Pruning in CNN is a technique that removes less significant neurons, feature maps, kernels, or possibly layers in large-sized networks. Random pruning can affect the accuracy of a network significantly. In pursuance of deep neural network acceleration, channel pruning was firstly proposed by [47]. A channel pruning method has been introduced by [22]. A two-step repetitive algorithm for pruning has been performed of some trained models of the CNN. This technique accelerated networks like Xception [10] and ResNet [38] up to $2\times$ speedup but the model accuracy is reduced by 2%. Also, the training time of the method is drastically high, and they have used the off-the-shelf libraries for the networks.

3 Proposed methodology

In this paper, a novel technique to induce modularity in convolution neural networks for image classification problems. This work contributes to three categories, i.e., domain decomposition, formation, and integration. We decompose the input domain to a CNN into multiple groups or clusters using the information learned by the CNN. The topology of modular neural networks remains the same; however, we utilize runtime pruning for modular topology formation. The results of a module are integrated into other modules.

3.1 Clustering input domain

Clustering the input domain (classes) is one of the main contributions of this work. It is a crucial step that enables us to achieve modularity in neural networks. Modularity aims to process input data in an organized way that similar input follows similar paths throughout the network. A modular neural network needs to group similar data based on some similarity index. Therefore, clustering input plays a crucial role in our proposed framework. We also propose Confusion Matrix driven Centroid Based Clustering (CMCBC) for clustering. CMCBC is an unsupervised clustering technique that utilizes k -Medoid algorithm. However, there is no distance function involved in the proposed method. Instead, it uses a confusion matrix to find similarities between each pair of classes and medoid for every cluster.

3.1.1 Confusion matrix driven centroid based clustering (CMCBC)

Generally, clustering algorithms start by computing the distance between every pair of units to be clustered. However, we use a confusion matrix obtained from a trained neural network as a distance matrix in this work. Although any confusion matrix does not follow the symmetry rule, we present further in this section that how it can be used effectively as a distance matrix in combination with *k*-Medoid clustering algorithm. The confusion matrix obtained from the CNN model trained on the MNIST dataset is shown in Fig. 1. The confusion matrix indicates the correlation between different dataset classes concerning the neural network. It demonstrates that the CNN confuses three class 0 with class 6 and 2 samples with class 8. It indicates some degree of correlation among different samples of different classes that can be exploited to cluster the dataset. In order to use confusion matrix for clustering, it needs to be preprocessed. Preprocessing confusion matrix in this work consists of two steps:

- Normalization
- Distance calculation

Normalization The confusion matrix is normalized just before using it for distance calculation. All the values in the matrix are transformed within the range of 0–1 using (1) where *cm* is the confusion matrix, *i* is used for row and *j* for column. *Sum* function returns the sum of the *i*th row of the confusion matrix obtained using *cm*^{*i*}. Figure 2 is the normalized version of the confusion matrix, shown in Fig. 1 calculated using (1).

$$cm^i_j = \frac{cm^i_j}{sum(cm^i)} \tag{1}$$

Distance calculation The normalized confusion matrix can be viewed as a similarity matrix where the highest values are at the diagonal. In order to convert it to a distance matrix, we

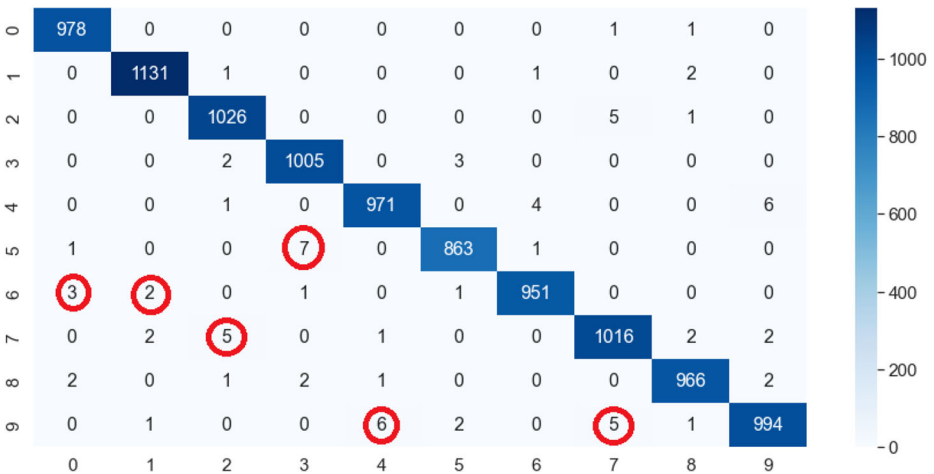


Fig. 1 MNIST dataset confusion matrix

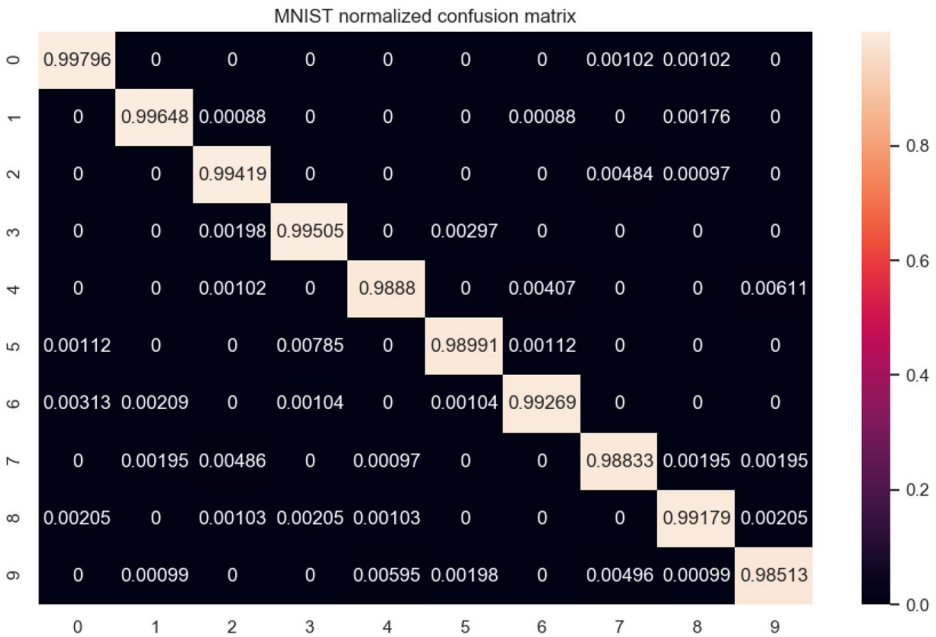


Fig. 2 MNIST model normalized matrix

have used (2) where x_j^i is a placeholder for a value at row i and column j of normalized confusion matrix.

$$y_j^i = 1 - x_j^i \tag{2}$$

Figure 3 depicts a distance matrix for MNIST dataset calculated from the normalized confusion matrix shown in Fig. 2. It can be seen that the MNIST distance matrix is asymmetrical. The values in the lower triangle of the matrix do not match the corresponding values in the upper triangle. In other words, the distance between x and y is not equal to the distance between y and x . Furthermore, the diagonal values are not equal to 0. Even though our distance matrix does not comply with the given rules of a distance matrix, it still can be used effectively in combination with k -Medoid for clustering. The pseudocode is presented as Algorithm 1.

Algorithm 1 Confusion matrix driven centroid based clustering algorithm.

Result: Clusters c
Input: Confusion matrix cm

```

1 for  $i \leftarrow 0$  to  $cm_{cols}$  do
2   for  $j \leftarrow 0$  to  $cm_{rows}$  do
3      $cm[i][j] \leftarrow 1 - cm[i][j]/sum(cm[i])$ 
4   end
5 end
6  $c \leftarrow kmedoid(cm)$  return  $c$ 

```

k -Medoid is a greedy algorithm that iteratively makes greedy choices. It compares the distances between a data point and medoids and clusters it with the closest medoid. The process repeats until an optimal state of the overall configuration is achieved. Provided the

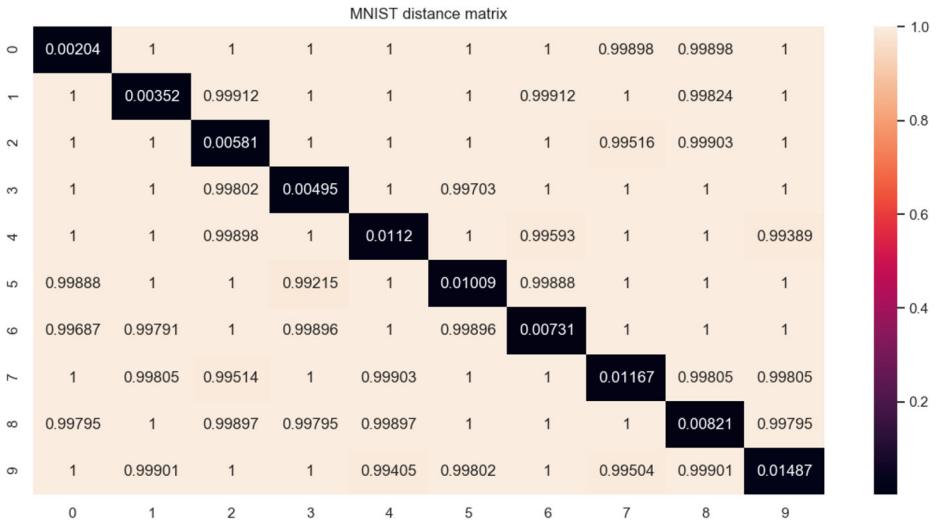


Fig. 3 MNIST dataset distance matrix

distance matrix in Fig. 3, the k -Medoid selects the smallest distance between two competing distances. The asymmetrical property of any distance matrix calculated from a confusion matrix does not affect the performance of k -Medoid clustering algorithm.

3.2 Inhibition mask based training

Runtime pruning is a type of network pruning in which the network is pruned dynamically. Kernels, feature maps, layer nodes, and channels can be pruned at run time. Models pruned with static pruning methods may permanently lose a significant amount of information. Runtime pruning solves this problem by temporarily removing information. We believe that this behavior of runtime pruning can be used effectively to attain modularity in neural networks. In this method, we have used inhibition mask-based runtime feature map pruning. It is a training process in which a pre-trained model is retrained to get a pruned model. Feature map pruning is enforced during retraining by an inhibition mask at each target layer. Target layers are those layers that are subject to pruning. The hit and trial method decides an optimal number of target layers. We use the term modular layer interchangeably also with the term target layer. The inhibition mask itself is a binary mask that works as a filter. It allows only a certain number of feature maps to pass to the next layer in each feedforward pass. Inhibition mask for each layer is designed based on intuition. Figure 4 shows a high-level design of the process.

3.2.1 Clusters

Clusters obtained using k -medoid algorithm in the previous step are given as input. It provides a base to select a mask for each input image so that all the input images that belong to the same cluster pass through the same module in a layer. It means that the number of clusters at a target layer equals the number of modules at that layer. Mathematically, suppose there are L_T target layers out of the network’s total L_N layers. In that case, the number of clusters C , for the entire neural network equals the sum of the number of clusters in each

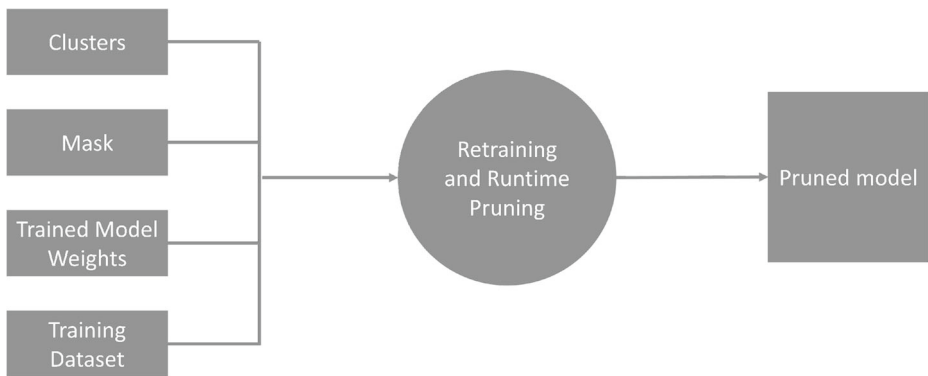


Fig. 4 High level diagram of inhibition mask based training process

target layer. The number of modules in the i^{th} target layer of the network is denoted as M_T^i . Similarly, the total number of modules in the network, represented by M , equals C . Thus, the number of clusters per target layer equals the number of modules in that specific layer.

3.2.2 Mask

Mask is a 3D binary tensor that is used for pruning. It works as a filter that allows some feature maps to move forward in feedforward pass while blocking the rest. The size of the mask for an overall network is equal to the sum of total modules at each target layer, as shown in (3). The sparsity degree and the pattern of the mask are decided empirically.

$$M_N = \sum_{i=0}^{L_T} M_i \tag{3}$$

Equation (3) indicates a sub-mask with a 2D shape for each target layer in a network. Each layer has a different number of modules and various units. Both of these terms define the shape of the sub-mask. The shape of each sub mask at a specific target layer can be calculated as the number of units in that layer times the total number of modules assigned to the layer.

3.2.3 Trained model

Weights of the trained model are used to initialize another identical model, which is then trained with runtime pruning enabled. This initialization process makes the new model retain the knowledge of the pre-trained model that we used to obtain clusters and gives a good point for the new model to start training. Initializing a new model from a pre-trained model is essential as we want to keep the old weights for further experiments.

3.2.4 Retraining and runtime pruning

The new model, initialized with old weights, is fine-tuned, taking all the above input parameters. The retraining phase enables the model to support pruning to achieve modularity. The difference between this retraining process and the standard training is that this process also utilizes our feature map pruning Algorithm 2. The algorithm receives feature maps from the

target layer as input and clusters them to one of the predefined clusters provided as a parameter. Based on the clustering result, a sequence of predefined masks is generated. Mask is a sequence of binary values with the same length as the number of modules in the layer. Each sub mask corresponding to one module has a length equal to the number of filters in the layer. We have implemented our framework so that the feature maps corresponding to the true values of a sub mask are zeroed. We are not removing the entire feature map from the list to maintain its shape for the next layer. The effect of pruning can be seen in the backward pass when gradients are propagated backward. Gradients calculated for the filters corresponding to the pruned feature maps are zero. Figure 5 graphically represents this whole idea.

Algorithm 2 Feature map pruning algorithm.

```

Result: Pruned feature maps
Input: Input feature maps  $F$ , their corresponding labels  $Y$ , binary mask for the layer  $M$ , and clusters  $C$ 
1  $indices \leftarrow$  arrange integers from 0 to the number of units in the layer as an array;
2 for  $i \leftarrow 0$  to  $batchSize$  do
3   for  $j \leftarrow 0$  to  $len(C)$  do
4     if  $y[i]$  in  $C[j]$  then
5        $maskIndex \leftarrow j$ ;
6       break;
7     end
8   end
9    $mask \leftarrow M[maskIndex]$ ;
10   $MI \leftarrow$  select each value from the list  $indices$  at every index where  $mask$  is  $True$ ;
11  for  $k$  in  $MI$  do
12     $F[i] \leftarrow 0$ 
13  end
14 end
15 return  $F$ ;

```

3.3 Inhibition masked based prediction

Using the previous step, the modular CNN can outperform the non-modular CNN of the same architecture. In non-modular convolutional neural networks, the prediction on the test

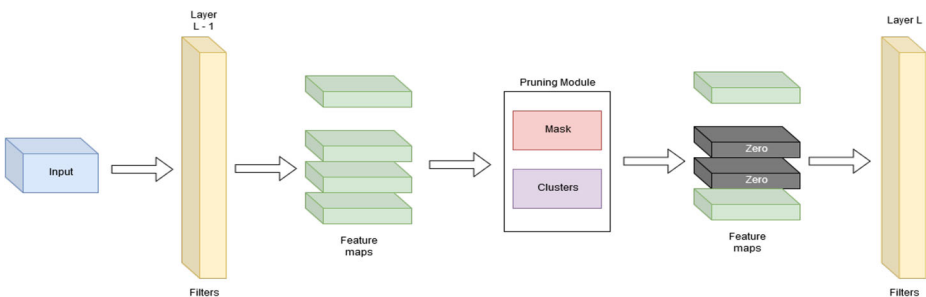


Fig. 5 Convolutional layer operations

dataset consists of only a feedforward pass. The forward pass of our modular CNN also includes a runtime pruning algorithm. The pruning algorithm uses the true labels provided with the training dataset to cluster input at each modular layer. A sub mask from a list of predefined masks is selected based on this clustering result. Each layer has a different number of units and additional modules, so the mask list for each modular layer is different. The selected mask activates only one module in each modular layer while deactivating the rest. In other words, all the modular layer units that do not belong to the active module are pruned. Inhibition mask-based pruning is a soft pruning technique. The Algorithm 2 works fine for training, but it cannot be embedded as it is in the feedforward pass of the testing phase. The problem that we have in the testing phase is that the system has no prior knowledge about the actual labels of the test data. Of course, labels are provided with test data, but that can only be used to evaluate the final performance of a trained network. To solve this problem, we present Algorithm 3 which is an extended version of Algorithm 2. The extended pruning algorithm utilizes neural networks to cluster inputs. Since every cluster corresponds to the mask, we call these neural network mask prediction models. One mask prediction model is trained for each modular layer. The model predicts the mask index in the mask list for input feature maps.

Algorithm 3 Extended feature map pruning algorithm.

Result: Pruned feature maps
Input: Input feature maps F , mask prediction model m , binary mask for the layer M , and clusters C

- 1 $indices \leftarrow$ arrange integers from 0 to the number of units in the layer as an array;
- 2 **for** $j \leftarrow 0$ **to** $batchSize$ **do**
- 3 $i \leftarrow m(F)$;
- 4 $mask \leftarrow M[i]$;
- 5 $MI \leftarrow$ select each value from the list $indices$ at every index at which $mask$ is $True$;
- 6 **for** i **in** MI **do**
- 7 $F[i] \leftarrow 0$;
- 8 **end**
- 9 **end**
- 10 **return** F ;

Training mask prediction model Mask or cluster prediction for runtime feature map pruning problem in the testing phase is solved by deploying additional neural networks. The number of mask prediction models in the modular neural network equals the number of modular layers. Each mask prediction model is deployed after its corresponding modular layer. It accepts feature maps from its preceding layer and returns the index of a cluster. Thus, additional computation for mask prediction models is the only overhead in our modular CNN. For training mask prediction models, we conducted two experiments. In one experiment, we trained the models on pre-activation of the modular layer, while in the other experiment, we used post-activations. Pre-activations are the output of the modular layer before the activation function is applied, and post-activation is the output of the activation function. The concept is depicted in Fig. 6.

Preparing dataset The datasets for training mask prediction models is prepared using the trained modular neural network, its input dataset, and the clustering information. The

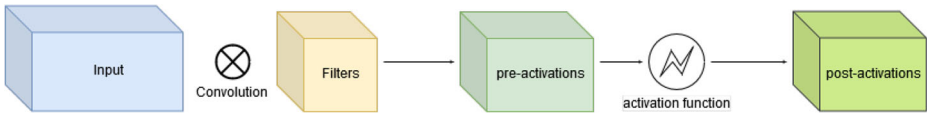


Fig. 6 Convolutional layer operations

trained modular CNN is used to save the layer activations for which the mask prediction model is trained. For this purpose, all three sections (train, test, and validation) of the input dataset are passed to the modular CNN. Activations of train data are used to train the mask prediction models, validation activations are used for validation, and test data is used to evaluate model performance after training. Labels for activations are generated using clustering information.

Training For the MNIST dataset, we have used fully connected neural networks (FCNN), while CNN models are used for the CIFAR10 dataset. In order to reduce the computational complexity of FCNN, a preprocessing step precedes the training. We have divided each $n \times n$ feature map into an $m \times m$ grid. The sum of every cell in the grid is normalized and fed to the input layer. This method reduces the computational complexity by $(n/m)^2$ times. Further details about the experiments are shared in the experiments section.

Deploying mask prediction models Each mask prediction model work as a subprocess of our modular neural network. Each process is called in every iteration of data on a modular neural network. The output of the mask prediction models is used for mask selection which is then applied to feature maps for dynamic pruning. One of the disadvantages of using sub-models is extra computation, while the other is that the loss of these sub-models is added to the loss of MCNN. Thus, it potentially reduces the overall accuracy of the MNIST model by less than 1%. However, it results in a 2% increase in the CIFAR10 model.

3.4 Single shot training for modular CNN

The framework presented in this section to induce modularity in CNN is a multi-step and time-consuming process. Training all the models or networks individually and manually integrating them is a tedious task. Another drawback is that feature maps that are used to train mask prediction models often need a massive amount of memory storage. It occupies ample space and consumes time to load into memory. We have experimented with multi-output CNN with an additional custom layer to solve these problems. A multi-output network is a neural network that consists of one or more sub-networks. Each network returns its output. All the sub-nets can take input either directly from the input layer or any other layer of any sub-network. All the models are trained simultaneously in a single training loop. In the case of supervised learning, labels for every sub-model with output are provided to the base model with a tag attached to it. The tag identifies which data is intended for which model. All sub-models can share the basic configurations (loss function, optimizer function, etc.) or can be configured independently. However, the loss of every sub-model accumulates to the loss of the base model. We have integrated our framework into a single training process using a multi-output model design concept. Also, all the data processing has been moved outside the main training loop. With this approach, we need to prepare data

only before the training. The overall model structure seems like the mask prediction models trained separately are now embedded inside the base model. So we call it Embedded Modular CNN (EMCNN) shown in Fig. 7.

Algorithm 4 Runtime pruning layer algorithm.

Result: Sparse feature maps

Input: Input feature maps F , mask prediction model output m_{out} , binary mask for the layer M

- 1 $cluster_index \leftarrow \text{argmax}(m_{out}, num_clusters)$;
- 2 $module_mask \leftarrow M[cluster_index]$;
- 3 $sparse_fm \leftarrow \text{multiply}(F, module_mask)$;
- 4 **return** $sparse_fm$;

3.4.1 Runtime pruning layer algorithm

The runtime pruning layer has no trainable parameters. It receives output from a mask prediction model and input feature maps from its preceding layer indicated by m_{out} and F respectively in the Algorithm 4. The mask M in the input is a 3-dimensional binary mask for the modular layer. If the mask’s shape is represented by $w \times h \times t$, then w and h are equal to the width and height of feature maps F passed to the algorithm, and t is equal to the number of modules in the layer. All the feature maps corresponding to the zero matrices in

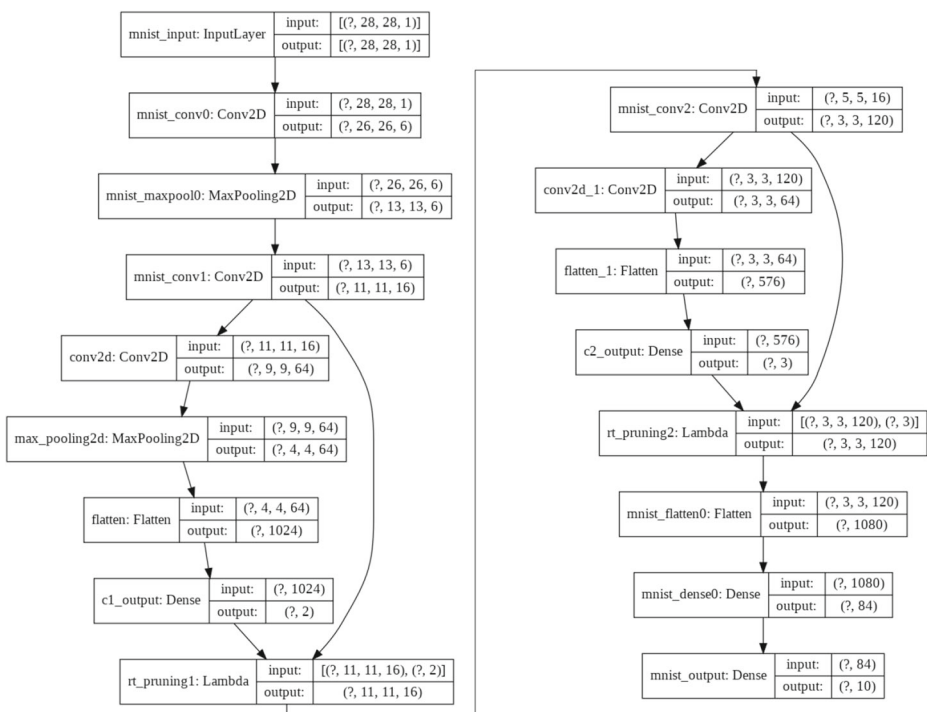


Fig. 7 Embedded CNN for MNIST

the mask get pruned by the layer. The *argmax* function decodes the one-hot encoded output of the sub-network where the *multiply* function performs pointwise multiplication on its arguments. The returned sparse feature maps represented by *sparse_fm* are forwarded to the next layer in the network.

4 Experiments and results

4.1 Preparing datasets

We tested our method on two benchmark datasets: MNIST and CIFAR10. The MNIST dataset contains a total of 70,000 28×28 grayscale images of handwritten digits. The dataset is divided into 60,000 training samples and 10,000 test samples which are categorized into ten classes. CIFAR10 consists of 60,000 32×32 RGB images. All the samples are categorized into ten different classes. The dataset is divided into 50,000 training samples and 10,000 test samples. Furthermore, the proposed method is evaluated on the Facial Aging database, IFDB. The IFDB contains around 1200 facial images along with age information.

4.2 Training MNIST and CIFAR10 networks

First, we discuss the experimental setup and network architecture for the MNIST dataset. The network architecture is outlined in Table 1 as CNN_{MNIST} . The network has three convolution layers followed by two fully connected layers. Each of the first two convolution layers precedes a max-pooling layer with 2×2 filter size and 1×1 stride. CIFAR10 experiment is conducted using $CNN_{CIFAR10}$ architecture outlined in Table 1. We use the default parameter settings as reported in *coarse_filter_pruning*. The network architecture is reported by authors with an alphanumeric string [3]. The $(2 \times 128C3)$ depicts two consecutive convolution layers with 3×3 convolution kernels and 128 feature maps. $MP2$ represents a single overlapped max-pooling layer with a kernel size of 3×3 and stride size of 2 for both x and y axes.

4.3 MNIST experiment

4.3.1 Clustering result for MNIST

In this step, we first cluster the input domain using *k*-Medoid and MNIST distance matrix calculated in Section 3.1.1. Column 1 of the table shows the number of clusters. While column 2, named “Clusters” represents items of each cluster. For example, the second row shows that all the labels are grouped in two clusters such that 0, 5, 6, and 8 consists of one

Table 1 CNN architecture details

Network	Architecture	Baseline MCR (%)
CNN_{MNIST}	$6(C5) - MP - 16(C5) - MP - 120(C5) - 84FC - 10Softmax$	0.61
$CNN_{CIFAR10}$	$2 \times 128C3 - MP2 - 2 \times 128C3 - MP2 - 2 \times 256C3 - 256FC - 10Softmax$	16

Table 2 MNIST clusters

Num of clusters	Clusters									
1	0 1 2 4 5 6 7 8 9									
2	0 5 6 8						1 2 3 4 7 9			
3	6 0 1 2				5 3		9 4 7 8			
4	0 1 6 8					5 3		9 4		7 2
5	2 1 7				2 5 8		4 9		0 6	
6	2 1 7				0 8		3 5		9 4 6	
7	7 2 9				6 4		0		1 3 5 8	
8	0 6		1 8		2	4	5	7	9	3
9	0 6		1		5	2	7	3	8	4 9
10	1		2		0	3	9	4	8	5 7 6

cluster and 2, 1, 3, 4, 7, and 9 consists of the other cluster. The first element of each cluster in the table is used as the medoid of the cluster (Table 2).

4.3.2 Inducing modularity in MNIST network

CNN_{MNIST} is extended to modular CNN ($MCNN_{MNIST}$ by marking 2 of its convolution layers as modular layers. Configuration details for $MCNN_{MNIST}$ are given in Table 3. The configuration table shows that there are seven layers in the MNIST model. Two layers at index 2 and 4 are modular layers. The first modular layer has six convolution units and the second layer has 120 convolution units. The clusters or modules are 2 and 3, respectively, for both layers. Modularity has been induced in the second and third convolution layers for the 2nd and 3rd modules. The feature maps of the second convolution layer are clustered into 2 clusters, while feature maps of the third layer are clustered into 3 clusters according to Table 2.

Inhibition mask for $MCNN_{MNIST}$ The binary inhibition mask for $MCNN_{MNIST}$ is designed intuitively. The mask for L_T^1 has a shape of 2×16 since it has two modules and 16 feature maps, while the mask for L_T^2 has a shape of 3×120 . The masks $m_1^{i=1,2}$ induce 50% sparsity in the layer, and both masks are mutually exclusive. Similarly, the masks $m_2^{i=1,2,3}$ induce 50% sparsity in the layer but are mutually inclusive. m_2^1 and m_2^2 are mutually exclusive to one another but 50% inclusive to m_2^3 . Table 4 makes the idea more clear by presenting the structure of inhibition masks for $MCNN_{MNIST}$. The table shows that the mask for the first module ($i = 1$) in the first modular layer ($t = 1$), m_1^1 prunes the first eight feature maps from 1-8 (both 1 and 8 are inclusive) for the input images that belong to cluster C_1^1 . The mask m_1^2 does the opposite. It prunes the last eight features of the layer from 9-16 for the input images that belong to cluster C_1^2 . Inhibition mask in the second modular layer where

Table 3 $MCNN_{MNIST}$ configuration

L_N	7	L_T	2
L_T^i	$i = 1$	$i = 2$	
L_N^i	$i = 3$	$i = 4$	
L_{FM}^i	16	120	
M_i	2	3	
C_i	2	3	
M	5	C	5

Table 4 Inhibition mask structure for $MCNN_{MNIST}$

Mask	Pruned feature maps	Non-pruned feature maps
m_1^1	1-8	9-16
m_1^2	9-16	1-8
m_2^1	1-60	61-120
m_2^2	61-120	1-60
m_3^3	1-30 and 91-120	31-90

$t = 2$, all the three masks induce 50% sparsity in the layer by pruning 60 out of 120 feature maps.

Mask prediction models for $MCNN_{MNIST}$ The Mask prediction model is trained for each modular layer in the network. For MNIST, we have trained fully connected networks with different architectures. Feature maps of training, validation, and testing datasets obtained from each modular layer feed FCN models. All the three datasets are preprocessed by dividing each channel of feature maps into 2×2 sub-matrices followed by calculating the sum of each sub-matrix. The preprocessed feature maps are normalized and flattened in row-major order. Since each feature map has 16 channels, we get 784 values as input. For the second modular layer, feature maps have a shape of $7 \times 7 \times 120$, giving us 1080 values for input after preprocessing. Table 5 show details about the experiment. It is evident in the table that the first mask prediction model is performing well in both aspects, i.e., accuracy and computational complexity.

Based on these results, we selected the first network as mask prediction models for their corresponding modular layer. The first network architecture achieves the highest accuracy and lowest computational complexity among the given networks for L_T^1 . However, it reaches the third-highest accuracy for L_T^2 , but the difference between the top 3 accuracies, in this case, is negligible.

4.3.3 Modular CNN for MNIST results

Using the CNN_{MNIST} model specified at Table 1, we achieved 99.39% accuracy on test dataset. We induced modularity for 2 clusters and 3 clusters in two consecutive convolution

Table 5 MNIST mask prediction models details

Input		Relu layers	Output (one-hot)		Accuracy	
L_T^1	L_T^2		L_T^1	L_T^2	L_T^1	L_T^2
784	1080	64 - 64	2	3	99.3	99.47
		64 - 64 - 128			98.01	99.41
		64 - 32 - 64 - 64			98.97	99.37
		64 - 16 - 32 - 64 - 64			98.9	99.18
		64 - 64 - 64 - 64 - 120			98.86	99.53
		64 - 64 - 64 - 64 - 64 - 64			98.89	99.47
		64 - 64 - 32 - 64 - 64 - 64 - 120			98.9	99.18

layers. After retraining the model with runtime pruning enabled, we calculated test accuracy in two ways, by simulating mask prediction models with 100% accuracy and deploying actual mask prediction models. Furthermore, we evaluated the effect of pruning at each layer by enabling pruning at each layer individually and performed test data iterations. For simulating mask prediction models with 100% accuracy, we cluster input to a modular convolution layer by directly reading its label. The feature maps are not forwarded to any of the mask prediction models.

In the Table 6, the column “Mask Prediction Model” indicates whether mask prediction models are deployed or not. If a column value is unchecked at a specific row, the accuracy is calculated by simulating mask prediction models. The test accuracy for all the cases in the table shows that the loss of the mask prediction models contributes to the loss of the modular base model.

Furthermore, the abrupt drop of accuracy mentioned in the last row of the table where the pruned modular CNN is evaluated with runtime pruning disabled validate our hypothesis that our framework enforces a set of specific convolution units to fit a particular input group. It enables different kernels to specialize in different sub-domains of the input by allowing only a subset of filters to training at a modular layer and precluding the others. We can call it targeted training. The results in the table are also illustrated in the bar graph Fig. 8. The blue bars represent our modular CNN accuracy on test data while simulating mask prediction models. In contrast, the other bars represent test accuracy reported in the table with the mask prediction model column set to enabled.

4.4 CIFAR10 experiment

4.4.1 Cluster result for CIFAR10

In this step, we first cluster the input domain based on the distance matrix using the k -Medoid algorithm. First we take the normalized confusion matrix of our trained $CNN_{CIFAR10}$ model which has 84% accuracy, show in Fig. 9. It can be observed that 123 sample dogs are confused as a cat by the model, the truck is confused with the airplane, and deer are mostly confused with birds. As mentioned in Section 3, we calculated the distance matrix as shown in Fig. 10, from the CIFAR10 normalized confusion matrix. Next, we applied k -Medoid algorithm on the distance matrix to cluster the CIFAR10 dataset (Table 7).

Table 6 Modular convolution neural network for MNIST results

Model	Pruning	Mask prediction model	Pruning at L_T^1	Pruning at L_T^2	Test accuracy (%)
CNN_{MNIST}	✗	✗	✗	✗	99.39
$MCNN_{MNIST}$	✓	✗	✓	✗	94.65
	✓	✗	✗	✓	96.79
	✓	✗	✓	✓	99.83
	✓	✓	✓	✗	94.19
	✓	✓	✗	✓	91.75
	✓	✓	✓	✓	98.51
	✗	✗	✗	✗	95.3

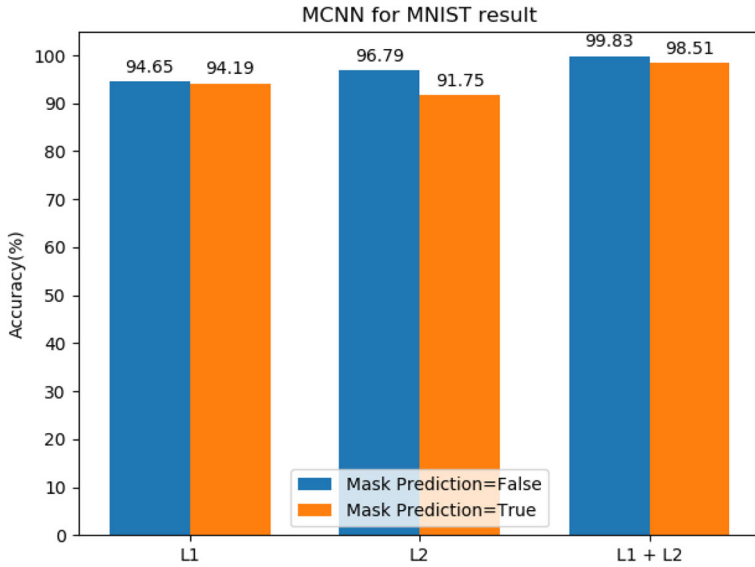


Fig. 8 Modular CNN for MNIST results bar graph

4.4.2 Inducing modularity in CIFAR10 network

$CNN_{CIFAR10}$ is extended to modular CNN ($MCNN_{CIFAR10}$) by marking 3 of its convolution layers as modular layers. Configuration details for $MCNN_{CIFAR10}$ are given in Table 8. The configuration settings depicted by the table indicates that there is a total of 10 layers (L_N) in $CNN_{CIFAR10}$ 3 of which are modular layer denoted by L_T . The modular layers lie at indexes 2, 4, and 7 of the $MCNN_{CIFAR10}$ respectively (indexes start from 1).

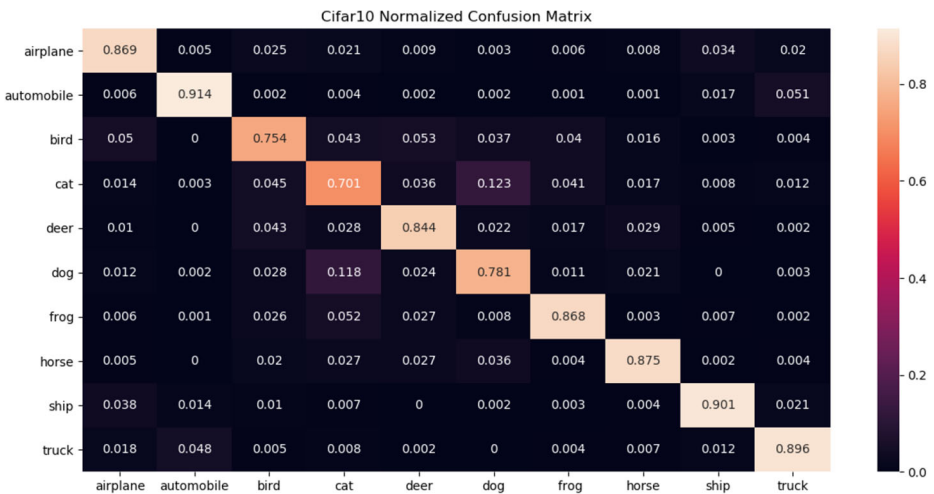


Fig. 9 CIFAR10 normalized confusion matrix

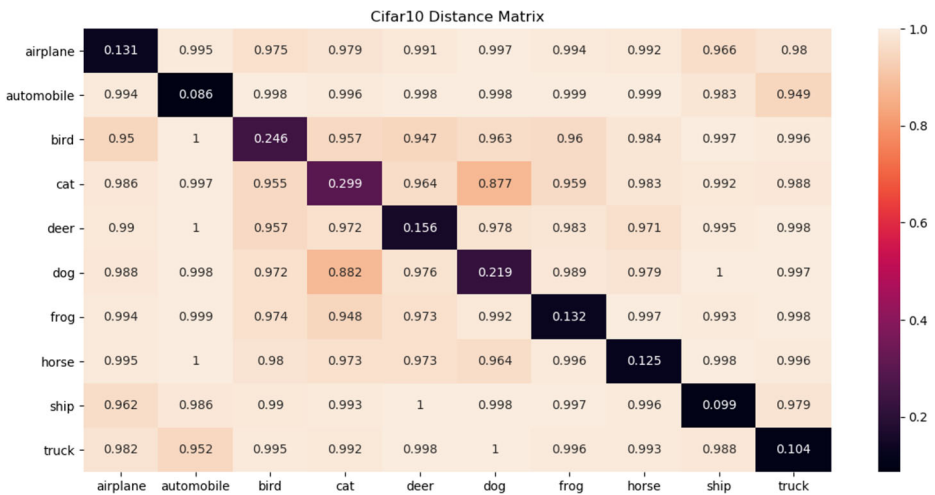


Fig. 10 CIFAR10 distance matrix

L_{FM}^i denotes the total number of feature maps produced by i^{th} layer of the network. M and C denote a total number of modules and clusters, respectively, in the entire modular CNN.

Inhibition mask for $MCNN_{CIFAR10}$ We have defined a binary inhibition mask for each modular layer of $MCNN_{CIFAR10}$. As shown in Table 8, the 3 modular layers has 128, 128 and 256 ReLu units and 2, 3 and 4 modules respectively. These two factors define the shape of the inhibition mask for its corresponding modular layer. The first mask (inhibition mask for the first modular layer) has a shape of 2×128 , the second mask has a shape of 3×128 , and the third mask has a shape of 4×256 . Table 9 explains the structure of each of the masks. All the masks induce 50% sparsity in their corresponding modular layer at runtime. Only $m_1^{i=1,2}$ are mutually exclusive. The sparsity of the mask determines the mutual exclusion property of the mask. One must be traded off for another. For example, to obtain a mutual exclusive mask for M_7^3 the sparsity must be greater than or equal to 75%. We have used masks with 50% sparsity in all our experiments for this work.

Mask prediction models for $MCNN_{CIFAR10}$ Mask prediction models specified in Table 10 are convolution neural networks. The “ReLU Layers” column refers to the number of hidden convolution layers and the number of convolution filters in each layer. All the layers have a 3×3 filter and ReLu as activation function. The model input comes from preceding modular

Table 7 CIFAR10 clusters

Number of clusters	Clusters									
1	deer airplane automobile bird cat dog frog horse ship truck									
2	ship airplane automobile truck					cat bird deer dog frog horse				
3	horse bird cat deer dog					automobile truck		ship airplane frog		
4	ship airplane		automobile truck			cat bird deer dog horse				
5	ship airplane		horse automobile			bird cat deer dog frog				
6	ship	airplane	frog cat			deer bird dog		truck automobile		horse
7	ship	horse deer		truck automobile		frog	dog	cat		airplane bird
8	automobile		bird deer		truck	cat frog	ship	airplane	dog	horse
9	ship	airplane	horse	frog	bird	dog	cat	automobile	dog	truck
10	ship	airplane	horse	dog	cat	bird	automobile	truck	deer	frog

Table 8 MCNN_{CIFAR10} configuration

L_N	10	L_T	3
L_T^i	$i = 1$	$i = 2$	$i = 3$
L_N^j	$j = 2$	$j = 4$	$j = 7$
L_{FM}^i	128	128	256
M_i	2	3	4
C_i	2	3	4
M	9	C	9

layers, and the output of the models is used for mask selection for runtime pruning. Adam optimizer is used with an initial learning rate of 0.001 besides with early stopping callback to stop the training at a convergence point. The table specifies all the model architectures we trained on the feature maps of their respective modular layers. The model in row 1 is selected for L_T^1 and L_T^2 , and the model in row 2 is used the layer L_T^3 due to their high accuracy on the corresponding layer feature maps and lower computation complexity.

4.4.3 Modular CNN for CIFAR10 results

The modular CNN for CIFAR10 results is compiled in Table 11. We have collected results for different scenarios. $MCNN_{CIFAR10}$ has been evaluated with both simulated mask prediction models and actual mask prediction models. Simulated mask prediction models mean that we have predicted masks for a set of feature maps based on its associated label even during inference. It enables us to observe base model performance without any additional loss due to the mask prediction model. The tests which incorporate “Pruning” and do not incorporate the “Mask Prediction Model” uses this technique. The fourth row of the table shows that our modular convolutional neural network has 88.86% accuracy. This is the maximum accuracy this model can achieve if mask prediction models are improved. The last row of the table report 78.01% accuracy with the best trained model and is about 6% less than the base model accuracy (84%). Based on our experiments, we hypothesized that the weights of mask prediction models are not aligned with the base model weights, resulting in a significant drop in accuracy. To make the alignments, we retrained our $MCNN_{CIFAR10}$ with all the mask prediction models enabled in the entire training phase. After retraining, our modular CNN outperforms the baseline model accuracy by 2.78% and achieves 86.78% accuracy. We refer to this retraining as weight alignment training in Table 11. The results in

Table 9 Inhibition mask structure for $MCNN_{MNIST}$

Mask	Pruned feature maps	Non-pruned feature maps
m_1^1	1-64	65-128
m_1^2	65-128	1-64
m_2^1	1-64	65-128
m_2^2	65-128	1-64
m_2^3	1-32 and 97-128	33-96
m_3^1	1-128	129-256
m_3^2	129-256	1-128
m_3^3	1-64 and 193-256	65-192
m_3^4	65-192	1-64 and 193-256

Table 10 CIFAR10 mask prediction models details

Input	Relu layers			Output (one-hot)			Accuracy		
	L_T^1	L_T^2	L_T^3	L_T^1	L_T^2	L_T^3	L_T^1	L_T^2	L_T^3
$30 \times 30 \times 128$	$28 \times 28 \times 128$	$13 \times 13 \times 128$	64 - 64	2	3	4	89.50	94.98	99.49
			64 - 64 - 128				89.52	95.04	99.94
			64 - 32 - 64 - 64				89.60	94.89	99.10
			64 - 16 - 32 - 64 - 64				89.39	95.06	99.93
			64 - 64 - 64 - 64 - 120				90.40	95.09	99.91
			64 - 64 - 64 - 64 - 64				90.71	95.35	99.95
			64 - 64 - 32 - 64 - 64 - 64 - 120				90.23	95.27	99.79

Table 11 Modular convolution neural network for CIFAR10 results

Model	Pruning	Mask prediction model	Pruning at L_T^1	Pruning at L_T^2	Pruning at L_T^3	Test accuracy (%)
$CNN_{CIFAR10}$	X	X	X	X	X	84.0
$MCNN_{CIFAR10}$	✓	X	✓	X	X	78.16
	✓	X	✓	✓	X	79.52
	✓	X	✓	✓	✓	88.9
	✓	✓	✓	X	X	78.24
	✓	✓	✓	✓	X	77.27
	✓	✓	✓	✓	✓	74.14
	X	X	X	X	X	79.26
Weight alignment training						86.38

the table are also illustrated in Fig. 11. The blue bars represent our modular CNN accuracy on test data while simulating mask prediction models. In contrast, the other bars represent test accuracy reported in the table with the mask prediction model column set to enabled.

4.5 Modular CNN for MNIST: single shot training results

We trained our model represented in Fig. 7 using Keras functional API with Adam as an optimizer and early stopping algorithm with 0.0001 minimum delta, six epochs patience, and the restore best weights flag enabled. The base model layers are initialized with MNIST base model (CNN_{MNIST}) weights while mask prediction sub-models are initialized with the default algorithm used by Keras. Furthermore, the MNIST training data provided to the

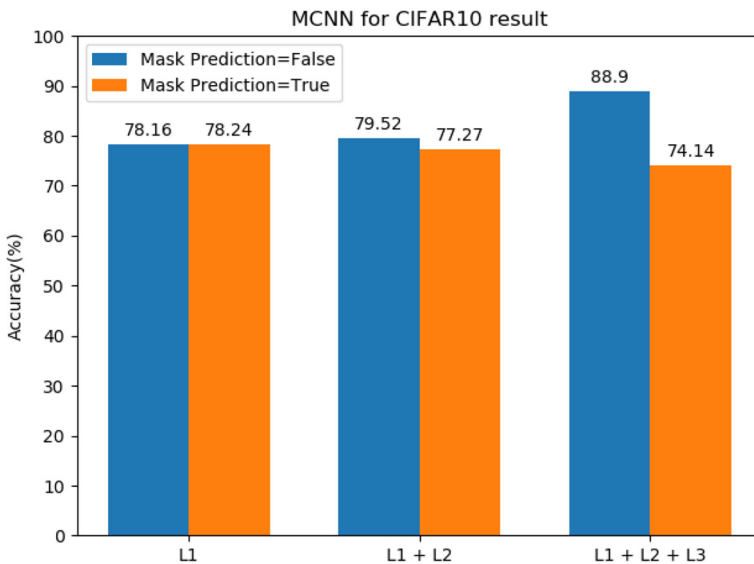


Fig. 11 Modular CNN for CIFAR10 results bar graph

model has been split into 5,000 validation images and 45,000 training images. We have achieved 98.69% test accuracy on the base model with all the above. The mask prediction models embedded into the base model for L_T^1 and L_T^2 have 99.12% and 99.36% accuracy. The loss of the overall mechanism is recorded to be 0.90, which is the Sum of MNIST base model loss (0.45) and the two mask prediction models loss (0.25, 0.20). It can be concluded from the results that the drop in the MNIST base model accuracy is due to the performance of mask prediction models. Trained mask prediction models add a certain level of overhead to the base model. We theoretically calculated the overhead and computational complexity reduction by runtime pruning in terms of multiplications. Table 12 represents the number of multiplications that can be avoided per sample iteration through the model. The first column represents the layer's name; the second column represents the total number of multiplications performed. The third column shows the number of multiplications that can potentially be avoided due to the sparsity added to feature maps. The 16Conv layer is the first modular layer, and the 120Conv is the second modular layer. The amount of complexity reduction is directly proportional to the sparsity of the pruning mask. The table further indicates that pruning at one layer also affects its next layer. Although the inhibition mask for the 120 Conv layer mentioned in Table 4 has a 50% sparsity ratio, we get 75% sparsity in the layer. The additional 25% sparsity is added due to channel reduction in its preceding layer, 16Conv. Similarly, pruning at 12Conv influences its succeeding fully connected layer named 84FC in the table. Despite the sparsity added to the model, mask prediction models add additional computations to the base model. The mask prediction models add 289,920 and 107,712 multiplications in total for L_T^1 and L_T^2 layers, respectively. Considering the sparsity in modular and fully connected layers, we get a net overhead of 333,504 multiplications. The mask prediction model architecture directly influences this overhead. Multiple cost reduction techniques can use to reduce the overhead.

Our proposed model achieved accuracy near the baseline model for the MNIST dataset. In the case of CIFAR10, our approach outperformed the baseline model when the weighted alignment (i.e., retraining of the network) was performed. But without weight alignment, the performance was almost 6% less than the baseline performance. In this work, at one end, we compromised the performance of the models with a slight fraction, but at the same time, we saved around 53% multiplications in the Convolution Layers. Keeping these multiplications helped to achieve low computational complexity.

4.6 Experiments of IFDB dataset

In order to further validate our approach, we evaluated it on the IFDB database and compared the results with the method proposed in [13]. The data was divided into four different age groups as shown in Table 13. There is accuracy loss due to the samples per class. Since

Table 12 $MCNN_{MNIST}$
Modular layers complexity
calculations

Layer	Total multiplications	Saved multiplications	Percentage
16Conv	11,616	5,808	50%
120Conv	17,280	12,960	75%
84FC	90,720	45,360	50%
Total	119,616	64,128	53%

Table 13 Results on IFDB dataset

	0-2	3-39	40-59	60-100
Dehshibi et al.	80	82.9	75.92	72.72
Hornig et al. DB	92.24	83.21	78.13	64.72
Ours	68.76	69.21	66.36	62.21

we have used the same models, the models are not too generalized to adapt to every problem. The same models are used for fair comparison and observed accuracy loss. But at the same time, the computational time was reduced.

5 Conclusion

CNN delivers state-of-the-art performance on various computer vision problems. Its early success for image classification problems developed the interest of researchers in the field. The hierarchical structure of CNN is inspired by the cortical region of the human brain. The hidden convolution layers pull out useful information from input images and forward it to succeeding layers. CNN has a monolithic structure and works as a black box despite the hierarchical processing mechanism. In this work, we proposed a framework to instigate modularity in a pre-trained convolution neural network for image classification. We exploit the information available in the confusion matrix of the model. We hypothesized that the confusion matrix of a trained or frozen CNN provides enough information to cluster the input domain. After the initial composition of modules, we utilized the confusion matrix as a distance matrix for domain clustering. The clustering divides the datasets into several groups or clusters.

In module composition, we manually selected the layers to induce modularity and composed the shape of the modules in each layer. We deactivate several modules by inhibiting mask-based runtime feature map pruning using artificial neural networks to route a group of input images through a specific path formed by modules. We call the routing networks to mask prediction models. The mask prediction models accept feature maps from a modular layer in the CNN and classify them to one of the available clusters. We select the inhibition mask for the cluster and produce zero sparsity in the input feature maps based on the result. To train the mask prediction models for modular layers, we prepared the train, validation, and test datasets to save the corresponding layers' feature maps and the available clustering and module configuration information.

Moreover, we use arithmetic integration of the module's output which operates in a competitive environment. The proposed framework is evaluated on two benchmarks datasets, MNIST and CIFAR10. On the MNIST dataset, we achieved 98.51% accuracy using our proposed Modular CNN compared to the baseline accuracy of 99.39%. But at the same time, we saved 53% multiplications in the network, which significantly reduced the complexity. Similarly, on the CIFAR10 dataset, our model achieves 78.01% accuracy, 6% less than the baseline accuracy (84%). But when we retrain the network to align the weights further, our model outperformed the baseline model accuracy by 2.78% and achieved 86.78% accuracy. Modularity produces sparsity in the network, but the computation overhead added to mask prediction models exceeds the sparsity produced. Also, modularity adds additional hyper-parameters to tune. However, we consider computational overhead reduction for future work. In the future, we will explore methods to predict the optimal configuration for overall

modular network structure, such as the number of modules and composition of modules and inhibition mask. We also consider adding modules to a pre-trained network for a heterogeneous task and profound hierarchical domain decomposition, and expanding modularity in the entire network. Furthermore, we intend to evaluate the capability of our modular CNN for knowledge distillation and transfer learning.

Author Contributions S.A.: Study design of neural network architecture and its implementation. S.U.A.: Supervision, writing article, article review. U.H.: Study design of neural network architecture, writing article. K.J.: Helped in developing neural network architecture, article review. J.R.: Study design, article review. S.A.: Supervision, writing article, article review.

Funding No funds or grants were received.

Data Availability The data used in this paper is publicly available. The link to the datasets is as follow:

- MNIST (<http://yann.lecun.com/exdb/mnist/>)
- CIFAR10 (<https://www.cs.toronto.edu/~kriz/CIFAR.html>)

Declarations

Conflict of Interests The authors declare no conflict of interest.

References


1. Anderson A, Shaffer K, Yankov A, Corley CD, Hodas NO (2016) Beyond fine tuning: a modular approach to learning on small data. arXiv:1611.01714
2. Anwar S, Hwang K, Sung W (2017) Structured pruning of deep convolutional neural networks. *ACM J Emerg Technol Comput Syst (JETC)* 13(3):32
3. Anwar S, Hwang K, Sung W (2017) Structured pruning of deep convolutional neural networks. *ACM J Emerg Technol Comput Syst (JETC)* 13(3):1–18
4. Bastanfard A, Bastanfard O, Takahashi H, Nakajima M (2004) Toward anthropometrics simulation of face rejuvenation and skin cosmetic. *Computer Animation and Virtual Worlds* 15(3):347–352
5. Bastanfard A, Takahashi H, Nakajima M (2004) Toward e-appearance of human face and hair by age, expression and rejuvenation. In: 2004 International conference on cyberworlds, pp 306–311
6. Blakeney C, Li X, Yan Y, Zong Z (2020) Parallel blockwise knowledge distillation for deep neural network compression. *IEEE Transactions on Parallel and Distributed Systems* 32(7):1765–1776
7. Braylan A, Hollenbeck M, Meyerson E, Miikkulainen R (2015) Reuse of neural modules for general video game playing. arXiv:1512.01537
8. Chihaoui M, Elkefi A, Bellil W, Ben Amar C (2016) A survey of 2d face recognition techniques. *Computers* 5(4):21
9. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1251–1258
10. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1251–1258
11. Ciregan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: 2012 IEEE conference on computer vision and pattern recognition. IEEE, pp 3642–3649
12. Cui Z, Henrickson K, Ke R, Wang Y (2019) Traffic graph convolutional recurrent neural network: a deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*
13. Dehshibi MM, Bastanfard A (2010) A new algorithm for age recognition from facial images. *Signal Process* 90(8):2431–2444
14. Freeman I, Roese-Koerner L, Kummert A (2018) Effnet: an efficient structure for convolutional neural networks. In: 2018 25th IEEE international conference on image processing (ICIP). IEEE, pp 6–10
15. Fritsch J (1996) Modular neural networks for speech recognition. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, Tech. Rep.

16. Gheorghe T, Ivanovici M (2021) Model-based weight quantization for convolutional neural network compression. In: 2021 16th International conference on engineering of modern electric systems (EMES). IEEE, pp 1–4
17. Ghosh S, Srinivasa SK, Amon P, Hutter A, Kaup A (2019) Deep network pruning for object detection. In: 2019 IEEE international conference on image processing (ICIP). IEEE, pp 3915–3919
18. Goldberg Y (2016) A primer on neural network models for natural language processing. *J Artif Intell Res* 57:345–420
19. Gradojevic N, Gençay R., Kukolj D (2009) Option pricing with modular neural networks. *IEEE Transactions on Neural Networks* 20(4):626–637
20. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv:1510.00149
21. Happel BL, Murre JM (1994) Design and evolution of modular neural network architectures. *Neural Netw* 7(6-7):985–1004
22. He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE international conference on computer vision, pp 1389–1397
23. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861
24. Huizinga J, Clune J, Mouret J-B (2014) Evolving neural networks that are both modular and regular: hyperneat plus the connection cost technique. In: Proceedings of the 2014 annual conference on genetic and evolutionary computation, pp 697–704
25. Jain S, Hamidi-Rad S, Racapé F (2021) Low rank based end-to-end deep neural network compression. In: 2021 Data compression conference (DCC). IEEE, pp 233–242
26. Karim F, Majumdar S, Darabi H, Chen S (2017) Lstm fully convolutional networks for time series classification. *IEEE Access* 6:1662–1669
27. Lavin A, Gray S (2016) Fast algorithms for convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4013–4021
28. LeCun Y, Bengio Y et al (1995) Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* 3361(10):1995
29. Melin P, Mendoza O, Castillo O (2011) Face recognition with an improved interval type-2 fuzzy logic sugeno integral and modular neural networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41(5):1001–1012
30. Mikolov T, Kombrink S, Burget L, Černocký J, Khudanpur S (2011) Extensions of recurrent neural network language model. In: 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 5528–5531
31. Modhej N, Bastanfard A, Teshnehlab M, Raiesdana S (2020) Pattern separation network based on the hippocampus activity for handwritten recognition. *IEEE Access* 8:212 803–212 817
32. Phan KT, Maul TH, Vu TT, Lai WK (2018) Dropcircuit: a modular regularizer for parallel circuit networks. *Neural Process Lett* 47(3):841–858
33. Ronco E, Gawthrop P (1995) Modular neural networks: a state of the art. Rapport Technique CSC-95026, Center of System and Control, University of Glasgow. <http://www.mech.gla.ac.uk/control/report.html>
34. Ronen M, Shabtai Y, Guterman H (2002) Hybrid model building methodology using unsupervised fuzzy clustering and supervised neural networks. *Biotech Bioeng* 77(4):420–429
35. Srivastava RK, Greff K, Schmidhuber J (2015) Highway networks. arXiv:1505.00387
36. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. *The J Mach Learn Res* 15(1):1929–1958
37. Szegedy C, Ioffe S, Vanhoucke V, Alemi A (2016) Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv:1602.07261
38. Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4 inception-resnet and the impact of residual connections on learning. In: Thirty-first AAAI conference on artificial intelligence
39. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016) Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2818–2826
40. Terekhov AV, Montone G, O'Regan JK (2015) Knowledge transfer in deep block-modular neural networks. In: Conference on biomimetic and biohybrid systems. Springer, pp 268–279
41. Tseng MM, Wang C (2014) Modular design, pp 895–897. Springer, Berlin
42. Verbancsics P, Stanley KO (2011) Constraining connectivity to encourage modularity in hyperneat. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp 1483–1490
43. Waibel A (1989) Modular construction of time-delay neural networks for speech recognition. *Neural Comput* 1(1):39–46

44. Wang T, Wu DJ, Coates A, Ng AY (2012) End-to-end text recognition with convolutional neural networks. In: Proceedings of the 21st international conference on pattern recognition (ICPR2012). IEEE, pp 3304–3308
45. Watanabe C (2019) Interpreting layered neural networks via hierarchical modular representation. In: International conference on neural information processing. Springer, pp 376–388
46. Wei W, Wong Y, Du Y, Hu Y, Kankanhalli M, Geng W (2019) A multi-stream convolutional neural network for semg-based gesture recognition in muscle-computer interface. *Pattern Recogn Lett* 119:131–138
47. Wen W, Wu C, Wang Y, Chen Y, Li H (2016) Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems, pp 2074–2082
48. Yuan M, Lin Y (2006) Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68(1):49–67
49. Zhang X, Zhou X, Lin M, Sun J (2018) Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6848–6856
50. Zhao Y, Wang D, Wang L, Liu P (2018) A faster algorithm for reducing the computational complexity of convolutional neural networks. *Algorithms* 11(10):159

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Salman Ahmad¹ · Shahab U. Ansari¹ · Usman Haider¹  · Kamran Javed² · Jalees Ur Rahman¹ · Sajid Anwar¹

Salman Ahmad
ahmadsalman145@gmail.com

Shahab U. Ansari
sansari@giki.edu.pk

Kamran Javed
kamranuettaxila@gmail.com

Jalees Ur Rahman
jalees@giki.edu.pk

Sajid Anwar
sajid@giki.edu.pk

¹ Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, District Swabi, Topi, 23640, Pakistan

² National Center of Artificial Intelligence (NCAI), Saudi Data and Artificial Intelligence Authority (SDAIA), Riyadh, Saudi Arabia