A Dissertation submitted in partial satisfaction of the requirements
for the degree of Doctor of Philosophy in Computer Science and Engineering
in the Graduate School of the University of Aizu

MS-NET: A Novel Approach to Boost the Accuracy of Deep Neural Networks Through Systematic Module Selection

by

Chowdhury Md Intisar

March 2022

The thesis titled

MS-NET: A Novel Approach to Boost the Accuracy of Deep Neural Networks Through Systematic Module Selection

by

Chowdhury Md Intisar

is reviewed and approved by:

Main referee		
Professor *ZHAO Qiangfu*	ZHAO, Qiangfu	Feb 10, 2022
Professor *LIU Yong*	Liu Yong	Feb. 10, 2022
Professor *MARKOV Konstantin*	K. Markov マルコフ	
Professor *TOMIOKA Yoichi*	Yoichi Tomioka	Feb. 10, 2022

THE UNIVERSITY OF AIZU

March 2022

Contents

List of Figures

List of Tables

List of Abbreviations

BWOD	Block-Wise Object Detection
CAM	Class Activation Mapping
CMNN	Coupled Modular Neural Network
DCNN	Deep Convolutional Neural Network
DNN	Deep Neural Network
EL	Ensemble Learning
FA	Feature Average
FC	Feature Concatenation
GAP	Global Average Pooled
ICC	Inter-Class Correlation
KD	Knowledge Distillation
LSA	Log-Softmax Average
MoE	Mixture of Experts
MS-Net	Modular Selective Network
NAS	Neural Architecture Search
O-MS-Net	Optimized MS-Net
ORR	On-Road-Risk
RR	Round Robin
SEED	Selective Expert Ensemble Distillation
SM	Softmax
SOTA	State-of-the-arts

List of Symbols

C	Number of classes
$\mathcal{D}$	Dataset
$\mathcal{T}$	Teacher Signal
N	Number of samples
S	Set of subsets
$\|S\|$	Cardinality of set of subsets
$\mathcal{E}$	Set of expert networks
$\mathcal{O}$	Final output of MS-Net
$\bar{\mathcal{E}}$	Set of expert networks selected by Router
R	Router network
ρ	Redundancy variable
K	Number of experts
$\mathcal{X}$	Bernoulli random variable
β	Branching factor
Q	vector of normalized logits
J	Inter-Class Correlation matrix

Dedicating my Ph.D. dissertation to my beloved Grandfather
Late Md. Ishaque Chowdhury.

Acknowledgment

At first, I would like to pay my earnest gratitude and thankfulness to my Almighty for giving me the strength and knowledge.

During my graduate study in the University of Aizu I was very fortunate to be able to attend a course titled *ITC05F- Machine Learning* which was taught by Professor Qiangfu Zhao. After attending this course my confidence, motivation and passion for research on Deep Learning and Computer vision grew stronger. Eventually I have decided to pursue in depth knowledge and experience through doctoral degree under the supervision of Professor Zhao. My gratefulness and thankfulness to Professor Zhao can not be expressed enough in words. Through out this three years PhD journey, through over thousand of emails and over hundred of meetings, Professor Zhao has provided me with plenty of professional guidance. Whenever I came up with a research idea Professor assisted me in distilling its core and fundamental essence and formulate it in a more professional way. Whenever I completed a research work Professor encouraged me to stay on track and improve based on our previous research. Whenever I wrote a very unstructured and terrible research draft Professor with utmost patience repeatedly corrected and guided me until I have perfected it. Professor Zhao did not only provide me with academic guidance, he also gave me opportunity to gain professional experience by participating in several collaborative research with well-reputed industries. This allowed me to understand the practicality of our research in the real-world field. Truth be told, Professor Zhao has transformed me from a confused, impatient researcher to a more consistent and focused researcher. To be very honest, I would have not been able to come this far without Professor Zhao's guidance and supervision.

I would like to thank Professor Yong Liu from the bottom of my heart, who has continuously provided me with constructive comments and advises from the very beginning of my PhD journey. Almost every week I had chances to discuss in detail about our research ideas and experimental results. In every single discussion Professor Liu always raised interesting issues and insightful observation from our experimental results (which is really inspiring!) which pushed me to improve continuously. A very important thing that I would like to add is that, the first time I have obtained in depth knowledge about the fundamental of Neural Networks was through a course taught by Professor Liu and Professor Zhao titled *CSA01 Neural Networks I: Fundamental Theory and Applications*. I still remember the way Professor Liu taught us how back-propagation works through those differential equations in the white board in a very easy way!

My earnest gratitude and thankfulness to Professor Konstantin Markov. I am grateful to Professor Markov for his calm and patience-ful teaching. I was in-fact one of his student in the course *ITA34 Practical Deep Learning*, where I have received a very strong foundation on state-of-the-art Deep Learning tools such as Pytorch Deep learning framework. The skills that I have learned from Professor Markov and his course have laid a very strong foundation for me during my PhD studies.

I am very grateful and thankful to Professor Yoichi Tomioka. I was really fortunate to be able to participate in collaborative project with Professor Tomioka from the first year of my PhD journey. Through our collaborative project meeting, Professor Tomioka provided me with several advises which assisted me in understating the practicality and scope of my research.

Through out my PhD journey I had the privilege to work with and learn from several brilliant minds. I would like to specially thank my colleagues whom I consider my brother Su kai (who also used to be my room-mate!), Wang Huitao (one of the kindest, hard-working and fast learner), and Xiasong (best athlete and football player, a brilliant researcher would loves to tackle difficult topic). They have generously supported me in academic and also my day to day life. I wish them best of luck and pray for their success. Su kai and Xiasong are PhD students now, and Huitao is working in AI industry. I firmly believe they will do amazing researches. I sincerely hope someday we can work together again.

I would like to thank my parents and my younger brother who always supported and encouraged me. It is indeed very difficult to stay far away from my family. But their unconditional love and prayer have given me the strength to work hard and make them proud.

Abstract

Image classification which is a sub-field of computer vision is now reliably performed by Deep Neural Networks (DNNs) in a near-human level accuracy or sometimes out-performing them. Succeeding in building an accurate image classification model will eventually produce success in downstream tasks such as, visual object localization; image segmentation; scene understanding and captioning; person-re identification; synthetic image generation and so on. Most of the networks leveraged for these tasks are currently monolithic. That is, a single neural network to perform classification on all possible classes. But, often datasets consist of several visually similar samples or classes where even these complex monolithic networks can fail or get confused. Such as, categorizing different breed of cats and dogs; different types of flowers; different types of mushrooms; different types of automobiles; and so on. Ensembles perform excellent in such situation, but they are expensive during both training and testing time. One of the promising solutions for such limitation is to make neural network modular, where several modules only concentrate and specialize on certain part of the dataset and perform collectively very good relatively to single complex model. In this dissertation, we conduct research on Modular Neural Networks (MNNs) and propose novel learning algorithms and architecture designs for the MNNs. We adopt image classification task for this study.

First in Chapter 2, for the multi-class image classification task we propose a modular architecture of Deep Neural Network known as the Modular Selective Network or MS-Net. The network primarily consists of a router network and a set of expert networks. The backbone for these router and experts is built with identical and simple Deep Convolutional Neural Networks (DCNNs). For a C class classification task, we show that the architecture has exactly C independent and decoupled expert neural networks. Moreover, for each class, MS-Net has ρ expert networks specializing in that particular class, where ρ is called the redundancy rate in this research. The concept of controlled redundancy rate in this architecture is novel. The research demonstrates that ρ plays a key role in the performance of MS-Net. Although these experts are light weight and weak learners alone, together they can match the performance of more complex DNNs for the whole task. We perform extensive empirical study and theoretical analysis for MS-Net on CIFAR-10, CIFAR-100 and F-MNIST datasets to demonstrate its performance and effectiveness.

Next, in Chapter 3, we propose optimization for MS-Net to improve the *accuracy and inference cost* trade-off per sample. We name this optimized model as the Optimized MS-Net or O-MS-Net. Original MS-Net is model agnostic in a sense that it does not require prior knowledge on the backbone network to construct the router and a set of expert networks. However, O-MS-Net takes into consideration about the router networks strength (or weakness) on the dataset. This information is obtained by performing Inter-Class-Correlation (ICC) analysis through calculating the joint-probability of co-occurrence of top-2 classes in routers prediction. From this calculated information K binary subset of datasets are prepared which are enriched in confusable set of classes. Value of K does not have any theoretical upper or lower bound. The value K is set based on computational resources and time available during training phase. Generally, the bigger the K the more experts we have, thus better performance. Unlike MS-Net, redundancy is not encouraged in these constructed subsets. Thus, during the inference phase,

besides router, only one expert neural network is leveraged per sample. Comparative study and performance evaluation of O-MS-Net is performed on the same datasets, i.e. CIFAR-10, CIFAR-100 and FMNIST.

Lastly in Chapter 4, we introduce more generalized version of MS-Net. MS-Net and O-MS-Net consist of router and expert networks which are decoupled and independent with no parameter sharing or re-use. In Chapter 4 we propose multi-branch neural network architecture where several modules (equivalent to experts of MS-Net) are closely coupled through parameter sharing and the network is end-to-end trainable. Hence instead of using the final output of the router, all the modules share the same feature map. There is no selective router or gating mechanism like MS-Net. Thus, we term our proposed network Coupled Modular Neural Network (CMNN). CMNN in our study consists of β closely coupled sub-networks, where β is the branching factor. We call the whole network super-graph and each sub-network within this super-graph as sub-graph. This kind of coupled modular design choice is indeed preferable as it makes efficient use of parameters and data. In order to effectively leverages the knowledge learned by this complex and powerful super-graph we propose a simple but easy-to-implement online Knowledge distillation based Round-Robin learning algorithm. The learning algorithm facilitates the sub-graph to effectively learn from the complex super-graph while encouraging diversity among them. We have shown that, such shared modular architecture and learning algorithm produces strong sub-graph network that can perform substantially better that its original baseline model. The framework can also produce strong ensemble performance when all sub-graphs are leveraged together. We validate our result through aforementioned datasets.

Chapter 1

Introduction

1.1 Deep Neural Networks

In order to introduce Deep Neural Network (DNN) [5] let us first consider the simplest form of a DNN, i.e a shallow neural network with a single hidden layer. Once we have a basic understanding of a single hidden layer neural network we will later generalize for the multi-layer networks or commonly known as the DNNs. Let us assume that we have an input sample x. Next, we have an input layer that takes the input sample x. In general, the dimension of both the input sample and the input layer are similar. If our input x is of Q elements the input layer of our neural network will have Q neurons. The input x next undergoes a linear transformation by W_1 (also known as the weight matrix) and b (known as the bias) to form intermediate form $xW_1 + b$. This transformation produces a row vector with D_1 elements (as we have D_1 neurons in the hidden layer). Next, an element-wise non-linearity denoted as $\sigma(.)$ is applied to the intermediate form, which completes the hidden layer step. There are several choices of non-linearity such as, Rectified Linear Unit (ReLU), Sigmoid, Tangent Hyperbolic (TanH), and so on. Finally, the output of the hidden layer is further linearly transformed by the second set of weight matrix W_2 to give the final output through the output layer. This output layer is also a row vector with D_2 computational units or neurons. Thus, from a single hidden layer neural network, we have two sets of weights, where W_1 is QxD_1 matrix, and W_2 is D_1xD_2 matrix. The bias b is a D_1 dimension vector. The final output of the network is depicted as,

$$\hat{y} = \sigma(xW_1 + b)W_2 \tag{1.1}$$

This simple neural network when equipped with more hidden layers, such as multiple linear transformations followed by element-wise non-linearity consecutively we have a deeper version of the neural network, which is known as the Deep Neural Networks (DNNs).

In order to leverage this neural network for performing prediction we need to define our dataset and an objective function. Let us assume that we have training dataset $\mathcal{D} = \{d_i | i = 1, .., N\}$, where N is the number of samples in that dataset. $\mathcal{T} = \{t_i | i = 1, .., N\}$ is the corresponding ground-truth labels, where t_i is associated with d_i for $i = 1, 2, ..., N$. Let us consider that our network performs classification task, and the output of the network for the input dataset $\mathcal{D}$ is

$$\hat{y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_N\}$$

Thus, for the classification task, we pass the output $\hat{y}$ through a SoftMax function defined in Eqn. (3.1), where we obtain the class probabilities as:

$$q_i = \frac{exp(\hat{y}_i)}{\sum_j exp(\hat{y}_j)}. \tag{1.2}$$

Once we have these output we formulate the loss function as follows:

$$\mathcal{L}(\mathcal{D}, \mathcal{T}) = \sum_{i=1}^{N} \log(q_i, t_i) \tag{1.3}$$

where, $t_i \in \{1, ..., C\}$ is ground truth class for corresponding input d_i.

The goal to train this neural network by minimizing the objective function defined in Eqn.1.3 w.r.t to the weights $W1, W2$ and b. This summarizes the basic principle of Deep Neural Networks or DNNs.

1.1.1 Deep Convolutional Neural Network

Deep Convolutional Neural Networks (DCNNs) are very similar to DNNs, such as they are made upon neurons with trainable weights and biases. However, DCNNs are much more versatile as they scale well for larger images with no significant increase in parameters. This has been possible because of the efficient use of convolution filters, pooling layers, and activation functions. DCNN has gained high preference and success mainly due to its ability to automatically learn discriminative features and complex input to output mapping through these convolution filters, pooling layers and activation functions using back-propagation algorithm [6]. Three properties of DCNN that make it very preferable are *local connectivity*, *parameter sharing* and *spatial arrangement*.

Each of the filter maps of CNN is panned on the entire image according to the kernel size and stride. This allows the filter to find matching patterns no matter where the pattern is located in the input image. Unlike the MLP, DCNN layers are sparsely and partially connected, which substantially reduces the number of parameters. Such properties of DCNNs [7, 8] have enable rapid and outstanding development and progress in computer vision task such as, object recognition [9–11]; object detection and localization [12, 13]; semantic image segmentation [14–17]; image generation [18–20]; neural style transfer [21, 22] and so on.

In the very beginning DCNNs networks (such as, [23] were simple and consisted of limited hyper-parameters and factors such as number of layers, number of channels per layer, and Sigmoid or TanH non-linearity. However, gradually these networks have evolved and started to get more complex, deeper, and more accurate with the introduction of templates such as fixed filter size with a large number of channels or feature maps [24], skip connection between non-consecutive layers [25], multi-path or branch design [26] and so on. Well known and most practiced networks such as ResNet [25], DenseNet [27], GoogleNet [28] and so on leverage the aforementioned templates. In Figure 1.6 we depict a simplified image of DCNN.

1.1.2 State-of-the-art Practices in Deep Neural Networks

State-of-the-arts (SOTA) DNNs are performing with excellent generalization capability and accuracy in the field of computer vision such as, visual object recognition [9–11]; scene segmentation [14–17]; person-re-identification [29]; realistic image generation [20]; natural language processing such as, language modeling [30]; playing games at human-level intelligence [31] and so on. In order to understand the reason for such outstanding performance we need to take a quick look at the last five years' progress in DNNs. Our survey reveals the following reasons for such as substantial progress in DNNS.

Scaling up DNNs has been known as an effective methodology to improve model capacity. Due to recent substantial improvement and progress in hardware infrastructure (Google TPUs, NVIDIA GPUs) for DNNs, DNNs are now scaled-up to an unprecedented level in terms of width and depth. To keep up with these scaled-up DNNs several parallelism libraries were proposed which allowed training DNNs with billions of parameters on multiple accelerators with no strict limitation for memory. Such as, Google GPipe [1], which is a scalable model-parallelism library

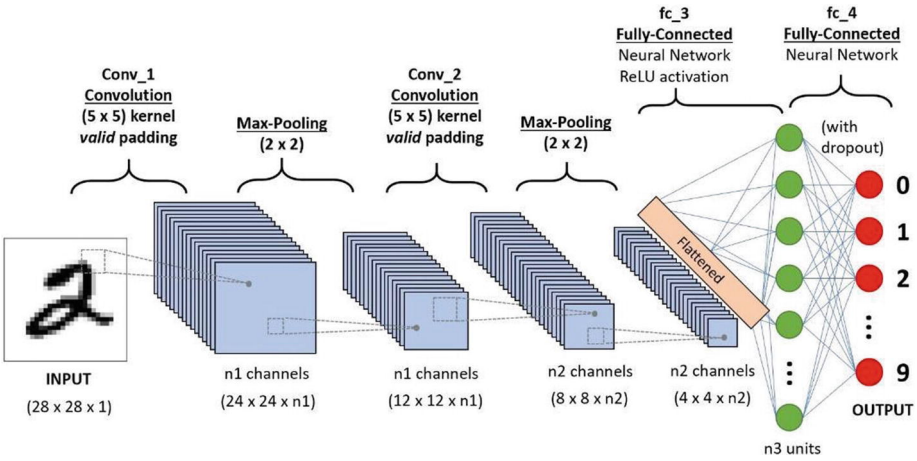


Figure 1.1: **Deep Convolutional Neural Network (DCNN)**

.

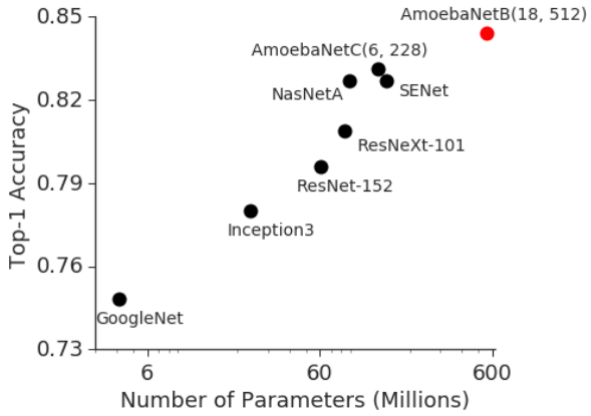


Figure 1.2: **Models trained by GPIPE framework.** [1]

.

for training Giant Neural Networks managed to scale up AmoebaNet [32] to 557 Millions (M) parameters and achieve 84.4% *top*-1 accuracy in ImageNet-2012. GPipe also allowed training a 6 billion parameters multilingual Transformer model on 103 languages achieving state-of-the-art results. Besides parallelism libraries, scaling strategy is also very important, as scaling DNNs without a proper strategy can result in over-parametric function thus, over-fitting. DNNs such as a series of networks from ResNet introduced a simple and brilliant concept of residual learning which allowed us to scale neural networks in-depth and achieve proportional performance gain. Similarly DenseNet [27] layer configuration alleviate the vanishing-gradient problem which allowed to scale up the depth without over-fitting.

Neural Architecture Search (NAS) a sub-field of Automated Machine Learning [33] (Or popularly known as AutoML) is recently dominating in achieving the best neural network architecture, in-fact architecture obtained through the NAS are currently outperforming human-engineered architecture (e.g [34]). The credit for AutoML goes to powerful hardware development and clever search algorithms and strategies [35]. Three key points that defined NAS are i) search space, ii) search strategy iii) performance evaluation. Since search space for the NAS task can be huge and which is basically an NP-Hard problem, prior knowledge and heuristic about typical properties of architecture are incorporated during the search to reduce search space [35]. Recent Neural Network architectures such as the *AmoebaNet-A (N=6, F=448)* [32], EfficientNet-B7 [36], NASNet-A [37] obtained through NAS have shown promising results in terms on accuracy and efficiency. However, these benefits come with a huge computational cost. Such as network obtained by reinforcement learning-based approach [38] required around *450 GPUS* for four days to perform a single experiment. Similar case is also true for the evolutionary based approach [34]. At present, EfficientNet which is obtained through evolutionary process and compound scaling (scaling in width, depth and resolution) method holds rank-1 in both visual object recognition and object detection [39]. EfficientNet employs a multi-objective NAS that optimizes network architecture both in terms of accuracy and FLOPS count.

Large Scale Transfer Learning enabled DNNs to achieve superior generalization capability even when exposed to a very small number of new training data. Transfer learning is referred to the situation where some tasks have been learned in one domain and are later exploited to improve generalization capability in another domain [5]. This has been an effective approach in reducing training time, hyper-parameter search. However, when this approach is leveraged with a scaled-up neural networks and coupled with a very large dataset the results obtained are outstanding. Such as, research [40] by Google Known as the Big-Transfer (BIT) demonstrated that training scaled up ResNet series networks (ResNet-50X1, ResNet-50X3, ResNet-101X3 and ResNet-152X4) on large scale dataset JFT-300M (300 million noisy labeled image dataset), ImageNet-21K and ILSVRC-2012, and later fine-tuning on dataset CIFAR-10 produced 99.4% test accuracy, which was the state-of-the-art score at that time. What is more interesting is that the same trained model when fine-tuned on CIFAR-10 with only 10 examples per class the network obtained 97.0% accuracy, which is almost comparable to a state-of-the-art score. Reaching this level of accuracy even with a large network by training from scratch on whole CIFAR-10 is really difficult and sometimes impossible. Attention-based networks known as the Transformer [30] and its variants [41,42] which have been dominating in the field of natural language processing has also started to take over in visual object recognition task. Again, research work by Google [43] demonstrated how pre-trained Transformer (with minimal modification in the original architecture) on large datasets such as ImageNet-21K and Google inhouse dataset JFT-300M achieves superior performance on ImageNet, CIFAR-100 dataset. All this evidence implies how powerful large-scale transfer learning is.

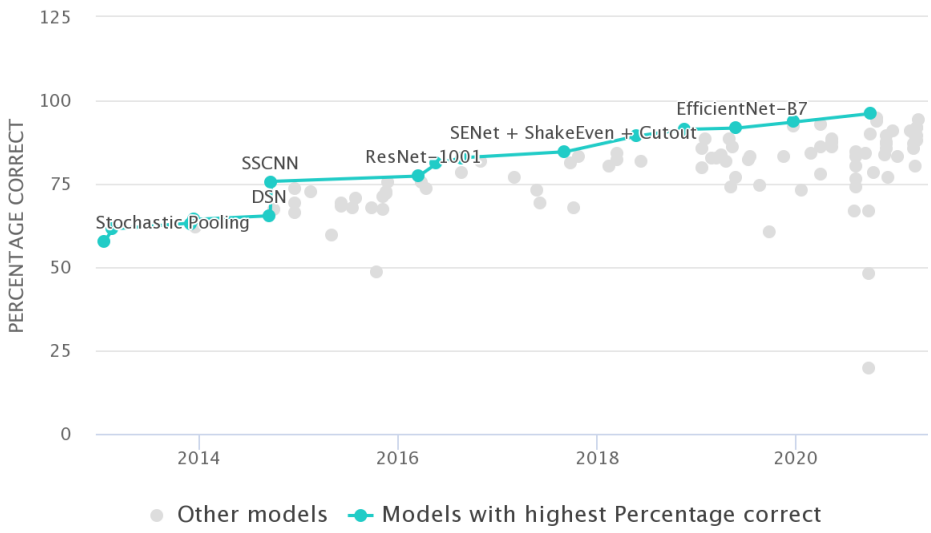


Figure 1.3: **Performance of State-of-the-art DNNs in Cifar-100 dataset**

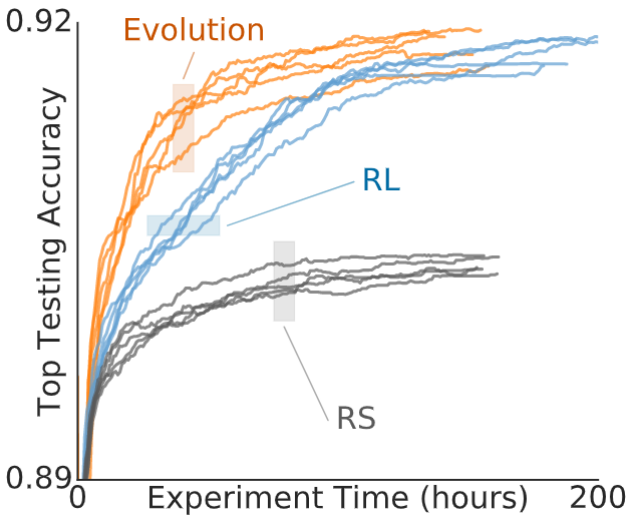


Figure 1.4: **Neural Architecture Search experiment time**

1.2 Modular Neural Networks

Modern (aforementioned) DNNs and practices are more or less monolithic approaches. That is, a single large neural network is constructed to solve a complex problem. As the input space gets bigger and more complex, we scale up the network accordingly to fit our network. In contrast to such an approach, a Modular Neural Network (MNN) is a special kind of network that consists of several modules where each module is responsible for learning one sub-task, and all modules function as one to solve one global task [2]. The global task can be any task that a neural network usually solves, such as, visual image classification [44, 45], speech recognition [46, 47], and so on. What makes MNN so interesting is the feature of the gating mechanism that dynamically selects only relevant specialist modules for the corresponding input. This allows each specialist module to be very simple while allowing them to learn complex non-linear relations from dataset collectively [48]. Before we provide a formal definition of Modular Neural Network let us first consider the key properties of Modular System for a high-level understanding.

The origin of the neural network was motivated by the Biological Neural Network [2]. The primary goal was to mimic the functionality of the actual human brain with a view to solving and modeling real-world complex problems. However, modern DNNs with backpropagation-based error calculation are not considered biological plausible [49], as we do not learn by backpropagation [50]. Survey [2] has clearly elaborated on some key properties of MNN that have certain biological plausibility and computational benefits.

First, **Specialization** property gives MNN the biological plausibility. Indeed, our brain is modular in different spatial scales [51]. Considering locality, our brain is composed of synapses which are clustered on dendrites [2, 51]. Considering a global scale, our brain is composed of several compartments or regions, where each region is responsible for performing different special tasks [2]. Such as, the visual cortex is responsible for processing color, the shape of objects, discriminative features of the visual scenes perceived through our eyes [52]. There are a certain group of neurons in the cortex region responding to important faces, memories, and taste [53]. Similarly, MNN has certain modules which are encouraged to specialize in certain concepts or tasks [54].

Second, **Fault tolerance**. The human brain Cortex when undergoes bilateral damage in a certain region, only parts of the ability such as sensation ability for color, pattern, or motion are disrupted. The rest of the abilities still remain functional. This inspires neural network design to be modular for situations where fault tolerance is very important. In MNN, each of the modules are specializing in a certain task, and the chances of mistakes by the modules are relatively smaller than a generalist module. Such as, in classification systems for medical purposes [55] or in self-driving road cars [56] we need a very accurate and precise neural network that we can trust.

Third, **Competition and Co-operation among modules**. This property gives MNN more stability, reliability, and equilibrium state. Considering our brain, the visual cortex employs certain connectivity and co-operation among cortex modules to perform the overall task of vision [52]. In general, competition and cooperation strategy can also be observed in nature, such as Ants foraging behaviour [57]. In MNN, competition and cooperation is important for reliable final decisions. Disagreement among modules can result in catastrophic module collapse and wrong prediction. Various systematic objective functions can be designed which can encourage cooperation through knowledge sharing [45, 58, 59] among the modules of MNN.

1.2.1 Formulation of Modular Neural Networks

MNN implementation and design can be of various kinds. Such as, modular network with gating mechanism or Mixture of Experts (MOE) [45, 60–62], different variants of popular ensemble learning [63–66] and so on. Here we will try to give a formal definition of MNN which

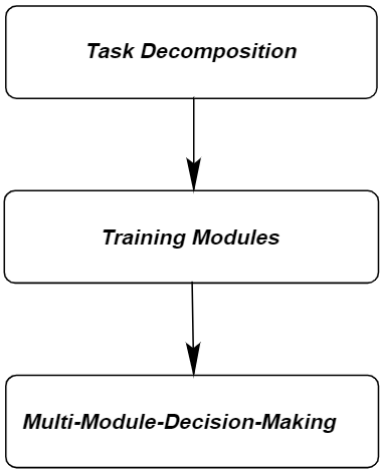


Figure 1.5: **The three main stages for designing MNN by [2]**

consists of the basic properties of MNN while ignoring other tiny details and variants.

First Stage in Figure 1.5: Let us consider we have set of dataset $\mathcal{D} = \{d_i | i = 1, .., N\}$, where N is the number of samples in that dataset. $\mathcal{T} = \{t_i | i = 1, .., N\}$ is the corresponding ground-truth labels, where t_i is associated with d_i for $i = \{1, 2, ..., N\}$. The goal is to partition the dataset $\mathcal{D}$ in to say K subsets based on some policy $\mathcal{P}$. We denote these subsets of dataset as $\mathcal{S} = \{\mathcal{D}_{sub_1}, .., \mathcal{D}_{sub_K}\}$. Here, partitioning policy $\mathcal{P}$ can be any pre-defined heuristics (by human) or based on some methods such as, clustering or projection of dataset into a latent space [67], predicting the confusing samples first [45] and so on. Moreover, partitioning can be done based on class, or simply grouping some set of samples with mixture of all classes in it.

Second Stage in Figure 1.5: Once we construct these K subsets of dataset we need to construct K expert neural networks say $\mathcal{E} = \{e_{sub_1}(), .., e_{sub_K}()\}$. $\mathcal{E}$ can be set of any neural networks of any size and types, such as multi-layer perceptron, DCNN and so on. The topology of each experts within the set can be different from one another depending on the tasks. Now, each of the expert e_{sub_i} will be trained on corresponding subset of data $\mathcal{D}_{sub_i}$, where $i = \{1, 2, ..., K\}$. As mentioned earlier, each of subset $\mathcal{D}_{sub_i}$ can either be subset of samples with all classes or subset of samples with selective classes.

Third Stage: Once we have these trained experts, inference will be performed by the aggregated decision of several experts, which will be selected by a router or gating network $\mathcal{R}()$. Let us assume that a set of experts $\mathcal{E}_r$ were selected by the router for input d, based on some selection policy, where $\mathcal{E}_r \in \mathcal{E}$. Simplified version of experts output aggregation will be as follows:

$$o_f = \frac{1}{|\mathcal{E}_r|} \sum_{e_r \in \mathcal{E}_r} e_r(d) * \mathcal{R}(d) \tag{1.4}$$

The aggregated output is weighted by the router's confidence. This kind of modular framework is decoupled modular network and is easier to parallelize as opposed to Mixture of Experts (MoE) [45]. There are also coupled MNNs that gained popularity for efficient parameter re-usage. The key design principle is to have a few shared intermediate layers among the expert modules and the router network [67]. This summarizes the preliminary understanding of MNN.

1.2.2 Task or Concept Partitioning for Modular Neural Network

In our earlier section, we stated that the dataset in the MNN framework is partitioned based on some policy $\mathcal{P}$. The performance and complexity of MNN largely depend on how this data partitioning policy is carried out. When the number of classes or distinct concepts to learn in a

dataset is huge, finding the right partitioning of concepts can sometimes be an exhaustive process. However, luckily with a few tricks and heuristics, it is possible to systematically partition the dataset and achieve good performance. Let us briefly illustrate some of the most common and classical data partitioning methods. Afterwards, we will introduce some modern practices for data partitioning for MNN.

Class Binarization (CB) is one of the most well-known data partitioning techniques in the modular learning framework. It can be considered as a special case of ensemble learning, where each binary module is assigned to learn or distinguish a single concept or class from the rest. Among different CB techniques, *ONE VS ALL* (un-ordered binarization) is the most commonly practiced technique in neural network [62], support vector machine [68], due to its computational efficiency and performance boost. The technique first appeared in the literature [69]. The method constructs C binary classifiers in total, where C is the total number of classes. Despite its simplicity, the method suffers from class imbalance, since the number of positive instances is smaller compared to the negative instances for each binary classifier. In addition, an ordered variant of the mentioned CB technique requires only $C - 1$ classifiers. However, the class imbalance short-coming was later resolved by the method *ONE VS ONE* which appeared in the literature *Separate-and-Conquer Rule Learning* [70]. A more systematic method for generating binary classifiers which is known as the *Round Robin learning* was introduced by the same author in the literature [61, 71, 72]. Due to its systematic method of creating a binary classifier, it carries more interpretability. The method has demonstrated that a total of $C\ (C-1)/2$ classifiers are constructed using the Round-Robin method. Each of these classifiers is a pair-wise-classifier, expert on two specific classes or concepts. Thus, the issue of class imbalance no longer prevails. In addition to that, authors have shown that this approach requires a relatively fewer amount of data during training as opposed to *ONE VS ALL* method. However, during the inference phase, all $C\ (C-1)/2$ classifiers require evaluation. With a view to resolving this computational issue. A relatively recent literature [60] proposed an efficient prediction algorithm for these ensembles, where pair-wise classifiers can be dynamically chosen without any drop in accuracy.

Data partitioning techniques discussed in the last paragraph are systematic, carry interpretability. However, these techniques are router agnostic. That is, partition policy does not take into consideration about the properties and difficulties of the samples according to the router. For instance, research [45] first evaluates the router network (also known as the generalist model in original literature) and performed clustering on the router's prediction co-variance matrix to pre-determine the confusing samples class. Next, expert networks are trained on those confusing set of classes. Each individual expert is a classifier of type *CONFUSABLE SUBSET VS REST*, where one part is the CONFUSABLE set of task and the rest ends up with single DUSTBIN class.

1.3 Structure of Thesis

The thesis is mainly divided into *6* chapters. The first chapter is the Introduction where preliminaries on DNNs and its recent trends; Modular Neural Networks and their formal definitions; and main motivation and contributions are briefly introduced. Chapters 2, 3, and 4 are novel contributions to this thesis. Each of these chapters is self-contained with background studies, literature review, and detailed experiments. Chapter 5 discusses ongoing open research studies, future research direction, and several supplementary experiments. Finally, I conclude this dissertation in Chapter 6.

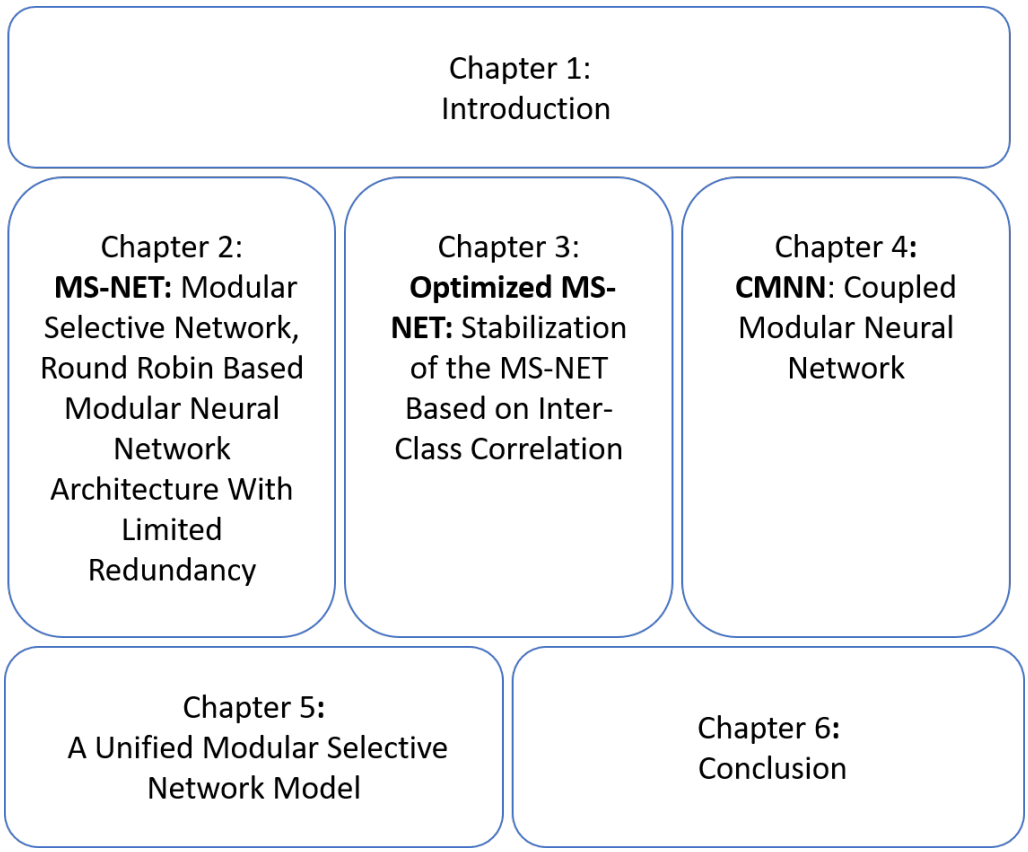


Figure 1.6: **Structure of Thesis**

.

1.4 Motivations

Two main motivations inspired us and will be reflected in this dissertation.

Motivation # 1: It is important to design neural network frameworks that is accurate and affordable in laboratory-level computers.

If we take a brief look at section 1.1.2 i.e. *State-of-the-art practices in Deep Neural Networks* for the visual object recognition task we can realize that i) Scaling up SOTA DNNs architecture will give superior classification performance, given that we have a proportional amount of training data and also powerful hardware at our disposal to train on them. ii) multi-objective NAS through different search strategies can produce compact, leaner and highly accurate neural networks. This strategy can bypass the need for scaling up DNNs and making them complex. However, *There is no free lunch*. NAS is an extremely expensive procedure and can take days or more even in a moderate amount of GPUS. iii) Large Scale transfer learning is also expensive.

Thus, our goal is to design a Modular Neural Network framework that can achieve performance comparable to the aforementioned SOTA DNNs practices while keeping computational cost tractable in both the training and testing phase.

Motivation # 2: MNN although introduced two decades ago, its popularity, contribution, and practice in visual object recognition with current SOTA DNNs are still sparse and have plenty of opportunities for improvement. As a result, we aim to contribute with a complete framework for MNN in the context of modern DNNs practice, and also shed light on several practical and potential research issues and opportunities for MNN.

The first complete framework of MNN with gating and expert modules appeared in the literature titled *Adaptive Mixtures of local Experts* [47] for the multi-speaker vowel recognition task. Later, during the early 90s several modular networks were proposed for the speech and phoneme

classification task [46, 62, 73]. MNNs for the vision task (such as visual image classification) was once again popularized by Hinton in the paper [45]. Later, a few researches [67, 74, 75] explored the image classification and labeling task by leveraging several variants of MNNs.

The dissertation approached with a view to overcoming the sparsity of MNNs practice in visual image classification task by implementing complete framework based on state-of-the-art practices.

1.5 Main Contributions

The dissertation is a report on my three years of research on **Modular Neural Networks architecture and framework design for the image classification task.** Chapter 2, 3, and 4 present our novel research works and experimental results. Chapter 2 first introduced our novel MNN architecture known as the Modular Selective Network (MS-Net). Chapter 3 proposed an Optimized-MS-Net with substantial improvement in parameter usage and classification accuracy. Chapter 4 proposed a more generalized version of MS-Net both in terms of design choice and learning framework. Finally, in Chapter 5 we discussed several open issues such as improving MS-Net training, unification of variants of MS-Net from Chapters 2, 3, and 4. Chapter 5 also includes several supplementary experiments that can guide us to both a promising direction and also directions to avoid. Our contributions to each chapter are summarized below:

1.5.1 Chapter 2

The chapter first introduced the novel Modular Neural Network framework known as the *Modular Selective Network or in short MS-Net for the visual multi-class classification task.* MS-Net introduced a novel systematic dataset partitioning technique for each expert neural network and an effective objective function to train expert neural networks. Summary of contributions of this chapter are as follows:

- The chapter proposed a novel data partitioning technique for the modular neural network based on the Round Robin (RR) method. The technique enables decomposition of the dataset into C subsets of class indices, where C is the total number of classes available in the dataset.
- The chapter introduced a concept called redundancy variable denoted as ρ. This variable allowed for controlled redundancy of any particular class or concept among the set of aforementioned C subsets. The chapter also provided a detailed theoretical and empirical study on the effect of redundancy variable ρ on MS-Net performance.
- Theoretically demonstrated that the MS-Net required no more than C expert networks to effectively specialize on the corresponding C subsets of classes.
- The chapter introduced a novel stochastic objective function to optimize the expert neural networks. The objective function enabled each expert network to specialize in their designated subset classes, while moderately generalizing on the rest of the classes.

1.5.2 Chapter 3

MS-Net of Chapter 2 achieved accuracy improvement by leveraging the concept of redundancy during inference. The network also achieved comparable performance relative to more complex monolithic DNNs. Redundancy in MS-Net enabled deployment of several expert networks for routers $top\text{-}n$ predictions in a systematic and controlled way. One simple question that we try to address in Chapter 3, *i) is it always necessary to evaluate multiple experts for a*

single sample? or in other words, can we reduce the number of expert evaluations per sample yet preserve modularity and performance comparable to original MS-Net?

The chapter thus performs research on these questions through the following contributions.

- Chapter 3 proposed another novel data partitioning policy for MS-Net based on Inter-Class Correlation (ICC) information. The ICC was calculated based on the top-n SoftMax prediction of the router network. Optimized MS-Net (O-MS-Net) of Chapter 3 does not explicitly advocate redundancy. As a result, fewer experts are required to train within this framework. These expert networks only specialize in the set of classes or tasks that are often mistaken or confused by the router network. Through the empirical study, the chapter substantiates the stability and effectiveness of data partitioning technique.

- Based on the proposed ICC-based subset construction method, the chapter also introduced an improved inference algorithm that leverages at best only one expert per sample. In the best-case scenario, no experts are leveraged as the prediction made by the router will suffice.

1.5.3 Chapter 4

MS-Net of Chapter 2 and Chapter 3 are MNNs with decoupled expert networks, where each experts are specialized on some certain subtasks. Later, router network is leveraged to direct to the relevant experts to perform inference for any input datum based on our proposed policies. Chapter 4 proposed more efficient, generalized and practical version of MS-Net where several modules are coupled through parameter sharing, hence the name Coupled Modular Neural Network (CMNN). As the name indicates there is no gating or ***S****elective* mechanism in CMNN. Like the MS-Net in Chapter 2 and Chapter 3, proposed generalized version of Chapter 4 has

- several modules or neural networks which are similar to architecture of MS-Net experts.

- input aggregator for several modules or neural networks to perform the final inference.

- Round-Robin training of modules.

However, unlike MS-Net of Chapter 2 and Chapter 3, proposed version of MS-Net of chapter 4 has the following improvements and generalization of features:

- CMNN does not have a selective router network, i.e. there is no gating mechanism to select relevant expert neural networks for input. Instead of a router, the network has a shared backbone that provides all the other modules' intermediate feature maps.

- Modules or networks are not decoupled, rather they are coupled through shared backbone encouraging primary level parameter re-use. Thus, more efficiency.

- Inference can be performed simply with a single module as opposed to MS-Net where multiple experts are used for inference. In other words, each module of CMNN is a stand-alone classifier with boosted performance. This performance boost is due to the knowledge shared by the collective efforts of several modules.

- However, when leveraging several modules for inference (like MS-Net), instead of aggregating final soft-max prediction produced by the individual modules, feature maps from individual modules are aggregated to make the final prediction.

- During training online distillation is leveraged as opposed to off-line distillation in MS-Net and O-MS-Net.

- Each module of MS-Net and its successor O-MS-Net is an expert neural network. However, modules (also known as the sub-graph) of the proposed generalized version are not an expert, rather a strong and boosted generalist network.

Thus the contribution of chapter 4 can be summarized as follows.

- In order to obtain boosted neural networks, a modular-based structural learning framework was proposed for the DCNNs. The framework was constructed based on the concept of sub-graph/super-graph design. All the sub-graphs are stand-alone neural networks that are trained by Knowledge Distillation (KD) from the complex super-graph.

- Three variants of super-graph architectures are proposed i.e. different ways of aggregating the output of all the sub-graphs while training the super-graph. The aggregation methods are i) concatenation of Global Averaged Pooled (GAP) features from all the sub-graphs, known as Feature Concatenation (FC) method. ii) averaging the GAP feature maps from all the sub-graphs known as Feature Averaging (FA) method and lastly iii) Log-SoftMax-Average (LSA) of output from individual sub-graphs.

- In order to effectively leverage the complex structure of the super-graph and obtain a strong sub-graph network, the chapter introduced a simple and effective knowledge-distillation-based Round-Robin training procedure. The effectiveness of this distillation-based training procedure was substantiated through empirical study, Class Activation Mapping (CAM) analysis of the sub-graph and super-graph networks.

In summary, MS-Net and its variants proposed in each Chapter is the successor of the previous Chapter with several incremental improvements, such as boosted accuracy and computational efficiency.

1.6 Publications

Research works and experiment results of Chapters 2, 3, and 4 have been published in the following journals and conferences.

Major Journals

1. **Chowdhury Md Intisar**, Kai Su, and Qiangfu Zhao. "MS-NET: modular selective network." *International Journal of Machine Learning and Cybernetics (IJMLC) 12.3 (2021): 763-781.*

2. **Chowdhury Md Intisar**, Qiangfu Zhao, Kai Su, and Yong Liu. "CMNN: Coupled Modular Neural Network." *IEEE Access 9 (2021): 93871-93891.*

Major Conferences

1. **Chowdhury Md Intisar**, and Qiangfu Zhao. "A selective modular neural network framework." *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST). IEEE, 2019.*

2. **Chowdhury Md Intisar**, Kai Su, Huitao Wang, and Qiangfu Zhao. "Mapping DCNN to a Three Layer Modular Architecture: A Systematic Way for Obtaining Wider and More Effective Network." *In 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), pp. 856-860. IEEE, 2020.*

3. **Chowdhury Md Intisar**, Kai Su, Huitao Wang, and Qiangfu Zhao. "Stabilization of the Modular Selective Neural Network Model Based on Inter-Class Correlation." *In 2021 5th IEEE International Conference on Cybernetics (CYBCONF), pp. 050-055. IEEE, 2021.*

4. Su, Kai, **Chowdhury Md Intisar**, Qiangfu Zhao. "Knowledge Distillation for Real-time On-Road Risk Detection." *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2020.*

Non-Major Conferences

1. Su, Kai, Huitao Wang, **Chowdhury Md Intisar**, Qiangfu Zhao, and Yoichi Tomioka. "You Only Look at Interested Cells: Real-Time Object Detection Based on Cell-Wise Segmentation." *In 2020 11th International Conference on Awareness Science and Technology (iCAST), pp. 1-6. IEEE, 2020.*

2. Wang, Huitao, Kai Su, **Chowdhury Md Intisar**, Qiangfu Zhao, and Yoichi Tomioka. "Comparison Between Block-Wise Detection and A Modular Selective Approach." *In 2020 11th International Conference on Awareness Science and Technology (iCAST), pp. 1-5. IEEE, 2020.*

Chapter 2

MS-Net: Modular Selective Network

The chapter proposes a modular architecture of Deep Neural Network (DNN) for multi-class classification task. The architecture consists of two parts, a router network and a set of expert networks. In this architecture, for a C-class classification problem, we have exactly C experts. The backbone network for these experts and the router are built with simple and identical DNN architecture. For each class, the modular network has a certain number ρ of expert networks specializing in that particular class, where ρ is called the redundancy rate in this study. We demonstrate that ρ plays a vital role in the performance of the network. Although these experts are light weight and weak learners alone, together they match the performance of more complex DNNs. We train the network in two phase wherein, first the router is trained on the whole set of training data followed by training each expert network enforced by a new stochastic objective function that facilitates alternative training on a small subset of expert data and the whole set of data. This alternative training provides an additional form of regularization and avoids over-fitting the expert network on subset data. During the testing phase, the router dynamically selects a fixed number of experts for further evaluation of the input datum. The modular nature and low parameter requirement of the network makes it very suitable in distributed and low computational environments. Extensive empirical study and theoretical analysis on CIFAR-10, CIFAR-100 and F-MNIST substantiate the effectiveness and efficiency of our proposed modular network.

2.1 Introduction

Deep Neural Networks (DNNs) in the last two decades have shown it's superiority in the field of visual object recognition [9–11]; image segmentation [14–17]; speech recognition and translation [76, 77]; natural language processing [78, 79]; reinforcement learning [31, 80, 81]; bio informatics [15]; educations [82, 83]; and so on. Despite their simple layered structures of neurons and connections, they have outperformed other machine learning models [84]. This superiority has been achieved due to its ability of complex non-linear mapping from input to output, automated rich and discriminate features learning as opposed to hand-engineered low-level features such as GABOR features [85], local binary patterns [86], SIFT [87] and so on. With the passage of time, we can notice that not only the performance is levitating dramatically, also networks are getting deeper [25, 88, 89] and wider [26]. As a result, these finer networks are lacking few important and desirable properties such as interpret-ability or comprehensibility, practical applicability in low computational devices and so on. In addition, problems such as catastrophic forgetting with the arrival of new data [90], lack of memory efficiency, have also started to arise. Fortunately, various novel approaches have been proposed to mitigate a few of these shortcomings. Recent notable approaches include knowledge distillation from the cumbersome models to smaller models [45]; compression of knowledge from ensemble to a single model [91]; pruning of neural networks [92–96]; efficient Neural Architecture Search

(NAS) [38,93,97]; modular neural network design [46,47,62,73,84]; and so on. There have been also significant advances in efficient hardware design architectures for DNNs. Intel Corporation has developed a neural computation stick powered by the Vision Processing Unit (VPU) which can accelerate the inference phase of complex DNN on a low computational device. Google has also recently developed small edge Tensor Processing Unit (TPU) for high-performance machine learning inference. These small ASIC devices for DNN can easily execute deep Convolutional Neural Networks (CNN), which make it one of the best alternatives for cloud-based service. Unfortunately, when it comes to state-of-the-art networks, these ASIC devices still face performance bottle-neck when executed in real-time scenarios. Thus, it is necessary to devote time and research on mitigating the above shortcomings of DNN.

In this chapter, we propose a novel modular neural network framework for multi-class classification, which is inherently simple and easy to implement. The key idea is to leverage a fixed number of experts, each with parameters as few as possible during the inference phase, while maintaining accuracy comparable to relatively complex and monolithic state-of-the-art DNNs. The proposed framework has a close resemblance to the model of the human brain depicted by Minsky in [98], where he described the human brain as a collection of specialist agents interconnected by nerve-bundles. Quoting from [98] *We're born with proto-specialists involved with hunger, laughter, fear and anger, sleep and sexual activity- and surely many other functions no one has discovered yet- each based upon a somewhat different architecture and mode of operation*. Analogous to this brain model, our framework consists of a countable set of expert agents and a router agent. In this literature, we term the expert agents as the expert networks and router agent as the router network. Each of these expert networks is expert on a specific subtask and their computation take place independently. Although they are not superior individually for a whole set of tasks, they outperform each individual network when they execute collectively. The router network moderates the execution of these expert networks. The concept of the modular neural network itself is not new. The key concept of modular connectionist goes back to the mid-1980s in [47]. A number of contributions such as [47,73,99,100] have approached the task of speech recognition using the modular connectionist theory. A majority of the proposed modular architectures are equipped with a gating network (analogous to our router network) and a set of expert networks. Despite the popularity of modular connectionist models during the 80s, the modular approach in recent DNNs (such as CNN, Recurrent Neural Network(RNN) and so on) era is relatively sparse, until recently Hinton and Vinyals have introduced the novel concept of knowledge distillation in neural network [45]. Our proposed modular neural network framework which we termed as the *MS-Net* has a close resemblance to [45,60–62] literature in the following key points: i) We divide the dataset into a number of subsets/subtasks/concepts. Afterward, we train a fixed number of expert neural networks on each of these subsets ii) The router module navigates us to those expert networks for further re-evaluation.

However, in addition to the above points and our previous work [101], the novelty of our contributions to this research can be summarized as follows:

1. We propose a simple data partitioning technique for the modular neural network based on the Round Robin method. This technique enables decomposition of the dataset into C subsets of class indices, where C is the total number of classes available in the dataset.

2. We provide a detailed theoretical and empirical study on effect of redundancy variable ρ on the complexity and performance of MS-Net.

3. We theoretically demonstrate that the proposed MS-Net requires no more than C expert networks to effectively specialize on the corresponding C subsets of classes.

4. We propose a new stochastic objective function to train the expert networks. The proposed objective function is composed of two terms, wherein the first one facilitates optimizing

the expert networks on data from its corresponding subset classes, and the second term optimizes networks on data from the whole set of classes.

2.2 Outline

We arrange the paper in the following order.

1. Section 4.2: Related works on modular neural networks and machine learning models, ensemble learning and so on.
2. Section 2.4: An overview of the MS-Net architecture.
3. Section 2.5: Detailed discussion on Round-Robin based dataset partition.
4. Section 2.6: Training procedure of expert networks, including algorithm.
5. Section 2.7: Inference phase on MS-Net, including algorithm.
6. Section 4.5: Detailed discussion about the datasets and experiment settings.
7. Section 2.9: Empirical analysis.
8. Section 2.10: Guidance for optimal hyper-parameters selection for the network.
9. Section 2.10.2 Effects of knowledge-distillation on MS-Net.
10. Section 2.11 Discussion on results and comparison to state-of-the-art DNNs.
11. Section 4.7: Conclusion and possible future works.

2.3 Prior Works

Modular architectures have been famous in neural networks or connectionist models for a long time. In addition to that, modularity has also been widely implemented in other traditional machine learning models. This approach has not only boosted the performance of these learning models, but also introduced virtues such as interpret-ability, training efficiency, distributed computation, reduction of parameters and so on [62]. In this section, we provide an overview on the neural networks and other machine learning models which exhibit modular behaviour.

Class Binarization (CB) is one of the most well-known method in the modular framework. It can be considered as a special case of ensemble learning, where each binary module is assigned to learn or distinguish a single concept or class from the rest. Among different CB techniques, *ONE VS ALL* (un-ordered binarization) is the most commonly practiced technique in neural network [62], support vector machine [68], due to its computational efficiency and performance boost. The technique first appeared in the literature [69]. The method constructs C binary classifiers in total, where C is the total number of classes. Despite its simplicity, the method suffers from class imbalance, since the number of positive instances is smaller compared to the negative instances for each binary classifier. In addition, an ordered variant of the mentioned CB technique requires only $C - 1$ classifiers. However, the class imbalance short-coming was later resolved by the method *ONE VS ONE* which appeared in the literature *Separate-and-Conquer Rule Learning* [70]. A more systematic method for generating binary classifiers which is known as the *Round Robin learning* was introduced by the same author in the literature [61, 71, 72]. Due to its systematic method of creating binary classifier, it carries more interpret-ability. The method has demonstrated that a total of $C\ (C-1)/2$ classifiers are constructed using the Round-Robin method. Each of these classifiers is a pair-wise-classifier,

expert on two specific classes or concepts. Thus, the issue of class imbalance no longer prevails. In addition to that, authors have shown that this approach requires relatively fewer amount of data during training as oppose to *ONE VS ALL* method. However, during the inference phase all $C\ (C-1)/2$ classifiers require evaluation. With a view to resolving this computational issue, a relatively recent literature [60] proposed an efficient prediction algorithm for these ensembles, where pair-wise classifiers can be dynamically chosen without any drop in accuracy.

Knowledge Distillation (KD) is a recent and very popular method for compression of complex and cumbersome DNNs. The method was first proposed by Hinton et al. [45]. This method is now widely implemented in deep learning research and industrial applications. Studies such as, [102, 103] have shown that, KD not only allows compression but also enables a relatively smaller student model to outperform its teacher model. The key idea is to train a student network to mimic the output features or the class probability distributions of the teacher network. The literature's [45] contribution was not only limited to KD, the authors have also proposed a modular network framework that has a very close resemblance to our proposed framework. The model consists of two main parts, a generalist network and a set of independent expert networks. Each of these expert networks is a simple CNN, which is trained on data that are often confused and misclassified by the generalist network. Thus, each individual expert is classifier of type *CONFUSABLE SUBSET VS ALL*, where one part is the CONFUSABLE set of task and the rest ends up with single DUSTBIN class. This notion implies that the generalist model needs to be evaluated first to obtain those CONFUSABLE set of classes. In-order to i) retain knowledge about the non-expert classes ii) avoid over-fitting and iii) solve the class imbalance problem the author initialized the expert networks with the weights of generalist network. The literature has shown that, as the number of expert networks increases, the accuracy increases proportionally. However, there have been no concrete indication and estimation on the number of experts covering those CONFUSABLE set of classes. In addition, the literature states that there can be situation where there are no expert networks covering a certain set of classes (since the generalist network is already confident on its prediction for those certain set of classes).

Recent research [59] titled *Deep Mutual Learning (DML)* which consists of cohort of student models resembles modular behavior. The DML enables a number of student models to mutually learn from one another by minimizing the Kullback Leibler (KL) Divergence between their predictions, which is a special case of KD [45]. The experiments have shown that the number of student networks in cohort during training can be extended to more than two. Moreover, empirical results show that, multiple student neural networks trained by the mutual learning out-perform single model network trained independently. This learning process has also shown to outperform the KD method.

Other notable recent research contribution on modular neural network includes the famous Generative Adversarial Network (GAN) [18], where two networks, discriminator and the generator network co-operate and compete against each other. There are also different variants of GAN which comprise of more than two networks [104]. Research [24] proposed modular like architecture that is build upon the existing state-of-the-art neural networks. In the literature, they re-configure the model parameters into several parallel branches where each branch is a stand-alone neural network. They have demonstrated that, the average of the log-probabilities of multiple parallel branches give better representation as opposed to the single independent branch.

In this chapter, our modular neural network framework has a very close similarity to the literature [45, 61, 62], such as presence of gating network and expert networks. But in contrast to *ONE VS ONE* and *ONE VS ALL*, our expert networks are not limited to binary classifiers. We introduce a simple Round-Robin based systematic data partition technique which enables us to train each expert on subset of multiple classes. A contrast to note that, unlike ensemble learning method such as well known AdaBoost [105],Bagging [63], Random Forest [106], Gradient boosting [107] and so on which requires the collective wisdom of all available classifiers, our

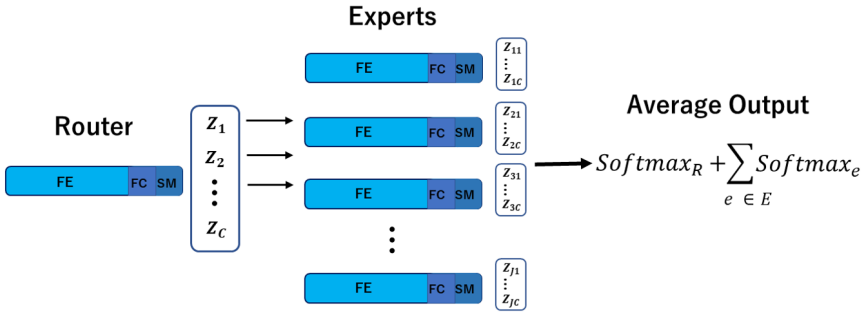


Figure 2.1: **Test Phase version of MS-Net. FE and FC depict Feature Extractor and Fully Connected layer of neural network respectively. E is the set of all experts dynamically selected by the router network R. The first block represents the router network which dynamically selects the expert networks based on its softmax (SM) confidence. The second part is the pool of expert networks further re-evaluating the router's top-n most likely predictions. Finally, the network aggregates the soft-max scores of router and selected experts.**

network does not require to run all the neural network models during inference. The novelty in our proposed framework is that, the router of the MS-Net extensively reduces the number of expert network evaluation during the inference phase. Since the partition of dataset is systematic, i) it gives us prior knowledge on which experts are specialist on which subsets, which also facilitates us to dynamically chose specific number of expert networks during inference. ii) it guarantees presence of multiple expert networks for a single concept or class, thus we have a certain degree of fault tolerance in case other experts or the router network fail to correctly classify the data.

2.4 Proposed network architecture

The network has two main modules, a router network, and a pool of expert networks. In the expert network pool there are C expert networks, where C is the total number of classes available in a given dataset. A simplified image of our modular framework is shown in Figure 2.1 . The expert networks and router network have the same network architecture. A very important issue is the size of the network. In our experiment, a cumbersome and computationally expensive network is not desirable. On the contrary, we also do not want the network to face performance bottle-neck due to simple architecture. There are many remarkable literature relating to the compact, efficient and accurate Neural Architecture Search (NAS) [35, 38] in recent times, but this topic is out of scope for this paper. However, the choice of architectures of any network are dependent on the complexity of the dataset. Considering the computational issue and memory efficiency, we chose *ResNet-20* [25] as the initial backbone network, which is of one the most minimalist and light weight network to our knowledge. We leverage the *Resnet-20* as the backbone of MS-Net to find the optimal hyper-parameters such as, the value for β, top-n and so on . After we obtain the optimal hyper-parameters through extensive empirical study with *ResNet-20* we train other complex DNNs which are, *GoogleNet* [88] and *MobileNet* [108] as the backbone network of MS-Net.

Table 2.1: **Backbone networks.**

Network	# of parameters (M)	MACs (G)
ResNet-20 [109]	0.269	0.041
MobileNet [108]	2.36	0.33
GoogleNet [88]	6.20	16.04

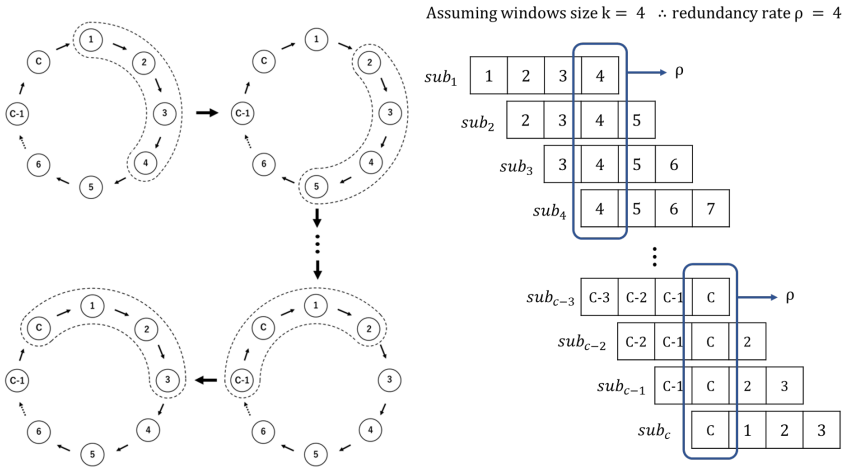


Figure 2.2: **Round Robin partition of the dataset. The left image depicts the sliding of the window over the classes. In each sliding operation, we have a subset. The sliding operation continues for C times. The right image illustrates the effect of size of the sliding window on the redundancy variable ρ. In the image we fix sliding window size to 4, hence we have each class occurring in exactly four subsets.**

2.5 Round Robin based Data-set Partition with sliding window

In this research, the redundancy rate plays a vital role in the performance of the framework. We denote the redundancy rate as ρ. The variable ρ has two main interpretations. First, ρ is the size of each subset of class indices. Second, each class index appears exactly in ρ subsets of class indices. In any sense, when ρ is larger more expert networks will get the chance to see the training data from any particular class. This is the reason why we called ρ redundancy rate.

In order to prove the above two points let us introduce several notations. First, we use $\mathcal{D} = \{d_i | i = 1, .., N\}$ to denote the whole training data set, where N is the total number of training data; and $\mathcal{T} = \{t_i | i = 1, .., N\}$ to denote the set of teacher signals, where t_i is associated with d_i for $i = 1, 2, ..., N$. To partition the subsets for training the expert networks, we leverage a sliding window of length k and stride s. Refer to Figure 2.2 for a graphical overview of dataset partition. In this figure, we arrange the indices of all classes in a ring-shaped manner. The sliding window length k is a positive integer less than C, which is the total number of classes. The redundancy rate ρ depends directly on k. Each time when we shift the sliding window with a stride s over the ring in a Round-Robin fashion (clockwise), we obtain a subset sub_i which contains k class indices. We use $S = \{sub_1, sub_2, ...\}$ to denote the set of all class index sets so far obtained. We can prove that, for any value of k and with stride $s = 1$, the cardinality of S is always equal to the total number of classes C. Since we have C target classes, and if we can prove $|S| = C$, we can conclude that our framework requires no more

than C experts.

Lemma 1 With stride $s = 1$ and for any value of k, the cardinality of S is always equivalent to the total number of class C available in the data-set.

Proof If k is the length of sliding window of any length, by the convolution arithmetic [110] we can state that the number of class index sets in S as:

$$|S| = \frac{(C-k)}{s} + k \tag{2.1}$$

Since we are using the Round Robin rotation, the later term k is added instead of 1. As, $s = 1$, Eq. (2.1) can be re-written as:

$$\begin{aligned} |S| &= C - k + k \\ &= C \end{aligned} \tag{2.2}$$

Thus, with stride 1, the total number of class index sets or the number of expert networks is always equal to the number of classes.

Lemma 2 If the length of sliding window is k and stride $s = 1$, the index for each class occurs exactly in k class index sets or in other words, we have exactly k experts related to each class.

This implies that the redundancy rate ρ is determined by the sliding window size k. This phenomenon also suggests that, k determines the fault tolerance of the proposed MS-Net. As the value of k increases, we have more experts for each particular class (Note that, the total number of experts remains constant i.e. C). On the contrary, as we decrease k, the redundancy rate or the number of experts specializing on that particular class decreases.

Proof Let us assume that the sliding window length is k, where $k < C$. After the $n-th$ $(n = 0, 1, ..., C-1)$ sliding operation, we obtain the following class index sets.

$$sub_{n+1} = \{n \bmod C + 1, ..., (n + k - 1) \bmod C + 1\}.$$

According to the definitions of the sliding window and the class index sets, $|sub_{n+1}| = k$. Since we are performing Round Robin rotation, we use the modulus operator for indices of each class.

Without loss of generality, we show that the index $(n + k - 1) \bmod C + 1$ exists in exactly k class index sets. During the Round-Robin partition, we shift each element of sub_{n+1} to the left of the sliding window with stride $s = 1$ as depicted in Figure 2.3. Thus, in each sliding operation we introduce a new class index to the right of the sliding window, which in the case of sub_{n+1} is $(n + k - 1) \bmod C + 1$. In the same way, for the next sliding operation, we have,

$$\begin{aligned} sub_{n+2} = \{&(n + 1) \bmod C + 1, ..., \\ &(n + k) \bmod C + 1\}. \end{aligned}$$

As we can observe in sub_{n+2}, the class index $n \bmod C + 1$ ceases to exist and a new index $(n+k) \bmod C+1$ arrives in the right most position. In addition, the index $(n+k-1) \bmod C+1$ shifts one position to the left. After the $n + k - 1$-th sliding operation, we have,

$$\begin{aligned} sub_{n+k} = \{&(n + k - 1) \bmod C + 1, ..., \\ &(n + 2k - 2) \bmod C + 1\}. \end{aligned}$$

The index $(n + k - 1) \bmod C + 1$ is now at the left most position. After the $n + k - th$ sliding operation, $(n + k - 1) \bmod C + 1$ will no longer exist in the subset sub_{n+k+1} because

$$\begin{aligned} sub_{n+k+1} = \{&(n + k) \bmod C + 1, ..., \\ &(n + 2k - 1) \bmod C + 1\}. \end{aligned}$$

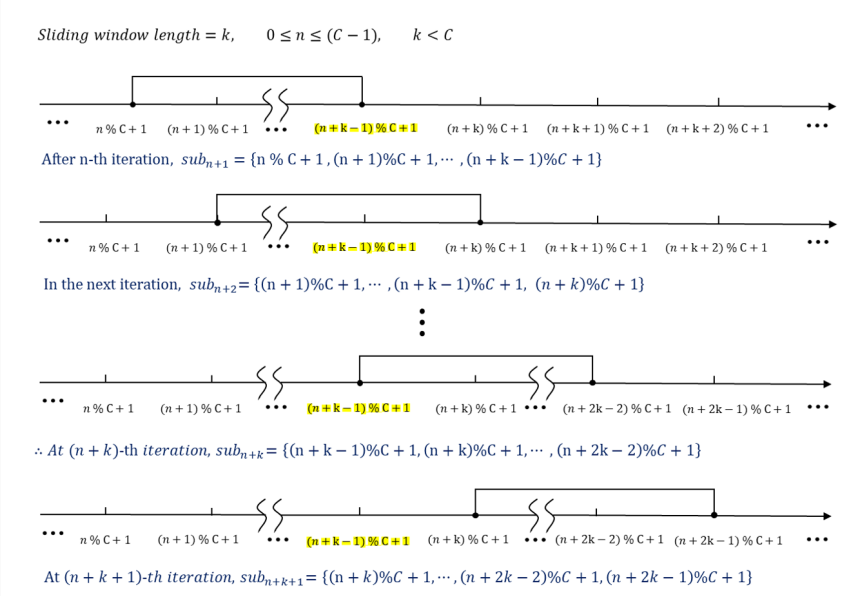


Figure 2.3: **Illustration of Lemma 2. The figure depicts that, with a sliding window length of k, each (in this figure, the highlighted class index $(n + k - 1) \bmod C + 1$ is shown to occur k times.) class index occurs in exactly k subsets. This also suggests that for each class in the dataset MS-Net has k expert networks.**

It is clear from above equation, $(n + k - 1) \bmod C + 1 \notin sub_{n+k+1}$ since after the $n + k - th$ sliding operation, the class index $(n + k - 1) \bmod C + 1$ slides out of the window. Thus, $(n + k - 1) \bmod C + 1$ occurs in $\{sub_{n+1}, sub_{n+2}, ..., sub_{n+k}\}$ or exactly in $(n + k) - (n + 1) + 1 = k - 1 + 1 = k$ class index sets, which also concludes we have k experts for the class $(n + k - 1) \bmod C + 1$.

2.6 Training Phase

We perform the training procedure in two steps. First, we train the router network on whole dataset. Second, we train C experts on the subsets which can be constructed based on the class index sets obtained in Round-Robin fashion depicted in Section 2.5. We denote the router network as $y = R(.) : \mathcal{D} \rightarrow \mathcal{T}$, where $\mathcal{D}$ and $\mathcal{T}$ are the dataset and the corresponding label set, respectively. The output of the router network is the softmax defined in Eq. (3.1), where we obtain the probabilities $q_1, ..., q_C$ for all C classes. Here, $z_1, ..., z_C$ are the logit scores for the corresponding classes.

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{2.3}$$

For our modular network framework, the top-1 score does not require to be strictly accurate. Since it is obvious that, the likeliness of the correct answer to be in top-n (as n increases) is higher than top-1, we take into consideration the top-n most probable answers. The role of the expert networks comes into play in this situation, where a set of experts further re-evaluate the router's top-n predictions. Thus, the accuracy of the experts have a significant effect on the MS-Net performance. Let us assume, we have a set of expert neural networks $\mathcal{E} = \{e(.)_1, ..., e(.)_C\}$. In order to ensure these experts effectively specialize on the subsets, we formulate a stochastic objective function which we depict in the Eq. (3.2). The objective function optimizes each of the expert network on its corresponding subset data $\{\mathcal{D}_{sub_i}, \mathcal{T}_{sub_i}\}$ using cross entropy loss function, where $\mathcal{D}_{sub_i} = \{d_j \in \mathcal{D} | t_j \in sub_i \wedge 1 \leq j \leq N\}$ and $\mathcal{T}_{sub_i} = \{t_j \in \mathcal{T} | t_j \in sub_i \wedge 1 \leq j \leq N\}$

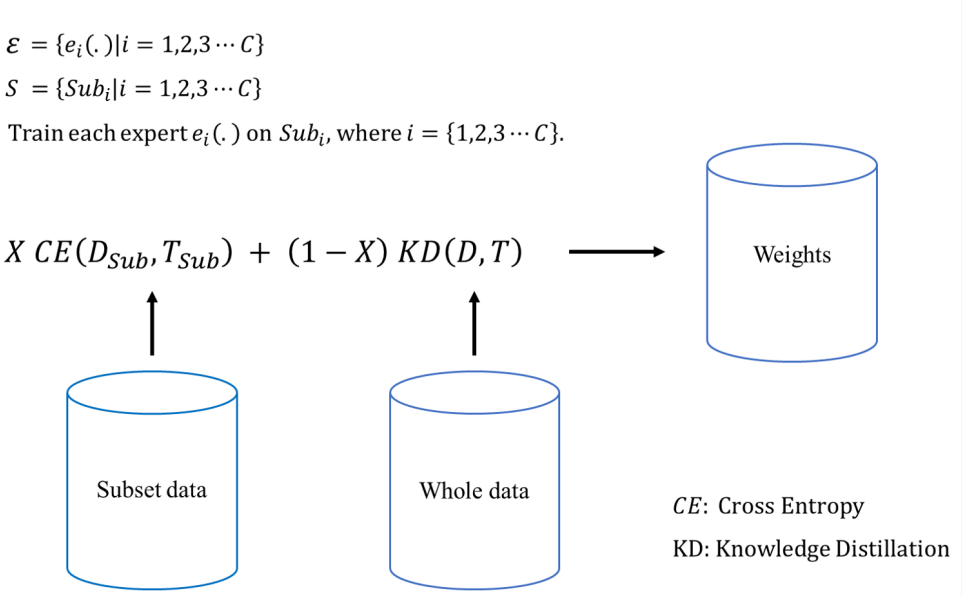


Figure 2.4: **Illustration of the training phase. This figure is the pictorial version of the Training phase section.**

and on the whole set of data $\{\mathcal{D}, \mathcal{T}\}$ using KD function, alternatively.

The knowledge is distilled from the router network. Thus the router is the teacher model. The alteration between two the loss terms in Eqn. 3.2 is controlled by the Bernoulli random variable $\mathcal{X}$ with the probability

$$Prob(\mathcal{X} = 1) = \beta.$$

The stochastic nature of the objective function for a certain range of β provides i) balanced training of networks and ii) better regularization. Again, the cardinality of each class index set sub_i is determined by the redundancy variable ρ. In our experiment we demonstrate the effectiveness and performance of the framework for $\rho = 2, 3$ and 4. We stress that, during the inference phase, as we increase ρ the number of expert network evaluation increases linearly. Due to the stochastic training of expert networks on whole dataset using KD (the second part of Eq. (3.2)), these networks are no longer limited to its corresponding subset data. Rather, each of the network is an expert on their own subset classes, in the meantime has certain generalization ability on the data of other classes.

In Eq. (3.2), the first term optimizes the expert network $e_i()$ on the classes defined by sub_i, weighted by Bernoulli random variable $\mathcal{X}$ which takes a value of 1 based on the probability β. The later term of Eq. (3.2) optimizes the network on the whole dataset weighted by $1 - X$ based on probability $1 - \beta$. Thus, β controls the trade-off between two loss terms in Eq. (3.2).

$$Loss_{e_i}^{kd} = \mathcal{X}\ l_{sub_i} + (1 - \mathcal{X})\ KD_{all} \tag{2.4}$$

where,

$$l_{sub_i} = -\sum_{l,t \in (\mathcal{D}_{sub_i}, \mathcal{T}_{sub_i})} \sum_{m \in sub_i} \delta(t, m) \log(P_{e_i}^m(l)) \tag{2.5}$$

and

$$KD_{all} = -\alpha \sum_{j=1}^{N} \sum_{k=1}^{C} P_{e_i}(d_j) \ \log \frac{P_{e_i}(d_j)}{P_R(d_j)} \\ -(1-\alpha) \sum_{j=1}^{N} \sum_{k=1}^{C} \delta(t_j, k) \log(P_{e_i}^{k}(d_j)) \tag{2.6}$$

In the above equations, $\delta(t, k)$ is the Kronecker delta function defined by

$$\delta(t,k) = \begin{cases} 1, & t = k, \\ 0, & t \neq k \end{cases}$$

The hyper-parameter α controls the trade-off between the KD and cross-entropy loss, where $0 < \alpha < 1$. The value of α during training depends on the performance of the teacher network. A high α value puts more weight on the distilled knowledge of teacher network and vice-versa. In our experiment, we aim to retain as much knowledge as possible from the router network (here the router network is the teacher network for experts) to the expert networks. In this way, we ensure that, the expert networks are at-least as good as the router network and if not, better. Thus in this literature, we fix the α value to 0.8. However, to learn more about the fine tuning of KD parameters we suggest to refer to the literature [45]. The purpose of leveraging KD in the loss function $Loss_{e_i}^{kd}$ is to simply retain all the knowledge of the router network in the experts. To illustrate the contrast, we construct another objective function depicted in Eq. (2.7) which is a variant of objective function in Eq. (3.2), but without knowledge distillation (wokd) term. We retrain all the experts using the loss function $Loss_{e_i}^{wokd}$ and illustrate performance gain by KD in the result discussion section.

$$Loss_{e_i}^{wokd} = \mathcal{X}\ l_{sub_i} + (1 - \mathcal{X})\ l_{all} \tag{2.7}$$

where,

$$l_{all} = -\sum_{j=1}^{N} \sum_{k=1}^{C} \delta(t_j, k) \log(P^{k}(d_j)) \tag{2.8}$$

Algorithm 4 illustrates the step by step training procedure of the MS-Net. In the Algorithm 4, line 1 through 4 performs the initialization of variable containers. In line 4 we obtain the subset class indices using the method discussed in section 2.4. Line 6 and 7 load the subset of training data corresponding to the class index sets. In the Line 9 we randomly sample training data which consist of all classes. Thus we have two set of training data available, one with classes exclusively from the class index sets and the other with all available classes. Line 10 and 11 perform the forward pass of the expert network $e_i(.)$ for the data from *all classes* and class index set respectively. However, the objective function defined in line 12 optimizes either of the term based on the state of the random variable X. Finally we perform the back-propagation of the loss term followed by parameter update for expert network. We carry out this procedure for rest of class index sets and expert networks.

2.7 Inference phase

During the inference phase of MS-Net, the cost or the model complexity is dependent on two key parameters, namely, n for top-n evaluation; and the redundancy rate ρ. In the testing phase, the input is first fed to the router. From the router, we obtain the probability scores for each class. Since the router is relatively small it is less likely that most of time the top-1 will be correct. But needless to say, the probability of obtaining a correct answer increases as the value

Algorithm 1: Training phase of MS-Net depicted in Figure 2.4

Input: Class index sets
Output: C trained expert networks

Dataset with all classes: $\mathcal{D} = \{d_i | i = 1, .., N\}$
Teacher signal: $\mathcal{T} = \{t_i | i = 1, .., N\}$
Expert networks: $\mathcal{E} = \{e()_i | i = 1, .., C\}$
Class index sets: $\mathcal{S} = \{sub_i | i = 1, .., C\}$
for *i, sub in* $enumerate(\mathcal{S})$ **do**
  $\mathcal{D}_{sub} = \{d_i \in \mathcal{D} | t_i \in sub_i \wedge 1 \leq i \leq N\}$
  $\mathcal{T}_{sub} = \{t_i \in \mathcal{T} | t_i \in sub_i \wedge 1 \leq i \leq N\}$
  for d_{sub}, t_{sub} *in* $enumerate(\mathcal{D}_{sub}, \mathcal{T}_{sub})$ **do**
    $\mathrm{d}_{all}, t_{all} = RandomSampler(\mathcal{D}, \mathcal{T})$
    $o = e_i(d_{all})$
    $o' = e_i(d_{sub})$
    $L_{e_i} = \mathcal{X}\, l_{sub_i}(o', t_{sub}) + (1 - \mathcal{X})\, l_{all}(o, t_{all})$ ▷$Pr(\mathcal{X} = 1) = \beta$ and $Pr(\mathcal{X} = 0) = 1 - \beta$
    $BackPropagation(L_{e_i})$
    $UpdateParameter(e_i)$
  end
end

of n increases. Thus, we select the *top-n* most likely classes or predictions $\mathcal{P} = \{p_1, .., p_n\}$ from the sorted softmax scores $q_1, .., q_n$ of the router. Next, for each predicted class p_i the router chooses ρ experts from the expert pool, where $i = \{1, .., n\}$. Thus, as ρ increases the number of expert evaluation for a particular class increases proportionally. For each element or prediction in $\mathcal{P}$, we select a set of experts using the following equation:

$$\bar{\mathcal{E}} = \bigcup_{p \in \mathcal{P}} \{e(.)_p \in \mathcal{E} | \exists sub \in \mathcal{S} \wedge p \in sub\} \tag{2.9}$$

where, $\mathcal{E}$ is the set of all experts whose cardinality $|\mathcal{E}| = C$ (refer to Lemma 1), and $\bar{\mathcal{E}}$ is a subset of experts available for a certain set of predictions $\mathcal{P}$ for a single input datum. In the proposed MS-Net we will always have C expert neural networks. This is shown by Lemma 1 and Lemma 2. However, during inference we do not leverage all C expert neural networks. Rather, the expert neural networks are selected based on ρ and n. For each input datum, the router selects n most likely classes for re-checking. For each class, we use ρ expert neural networks to provide information for making the final decision. Thus, MS-Net leverages at-most $(\rho * n)$ and at-least $(\rho + (n - 1))$ expert networks during the inference phase. The value of $(\rho * n)$ and $(\rho + (n - 1))$ are always smaller than C. In this chapter the maximum value for ρ and n are only 4 and 3 respectively. The prediction we obtain from the aggregated softmax of the set of selected experts $\bar{\mathcal{E}}$ for input x is presented in Eq. (2.10). For single input x, the softmax returns $\{q_1, .., q_C\}$, where each of the element q_i is the probability of x belonging to the class i.

$$\mathcal{O} = sm_r + \sum_{e \in \bar{\mathcal{E}}} sm_e(x) \tag{2.10}$$

where, sm_r and sm_e are the softmax scores by the router and experts, respectively. Finally, we take the most likely output label or the predicted class using Eq. (2.11)

$$prediction = \underset{j}{\text{arg-max}}(\mathcal{O}|q_j \in \mathcal{O}, 1 \leq j \leq C) \tag{2.11}$$

Algorithm 2: Testing phase of MS-Net

Input: Dataset $\mathcal{D}$
Output: accuracy: Accuracy of MS-Net

```
1  Empty list for top-n: preds = {}
2  Router network: R(.)
3  Expert networks: E = {e_i(.)|i = 1, .., C}
4  Flag dictionary: visited[sub_1 : sub_C] = False
5  Class index sets: S = {sub_i|i = 1, .., C}
6  Counter: c = 0
7  for d, t in enumerate(D, T) do
8      preds = R(d)
9      for p in preds do
10         for i, sub in enumerate(S) do
11             if p ∈ in sub & visited[sub] == False then
12                 loadExpertModel(e_i(.))
13                 asm = asm + softmax(e_i(d))
14                 ▷ asm is the Average Soft-Max count = count + 1
15                 visited[sub] = True
16             else
17                 continue
18             end
19         end
20         prediction = arg-max_i(asm[0 : C])
21         if prediction == t then
22             correct = correct + 1
23         end
24     end
25     accuracy = 100 * correct/|D|
26 end
```

Algorithm 3 represents the testing phase of the MS-Net. Line 1 through 6 initialize the variables and all the networks (router and expert networks). Initially, we pass the input to the router network in line 8. We select the top-n most probable predictions from the router whose further re-evaluation start from line 9. Based on the prediction of router we select a fixed number of expert networks. As discussed in the earlier section, the number of expert networks for inference is governed by the variable ρ and top-n. In the worst case scenario we will have to evaluate at-most $(\rho * n)$ expert networks and in best case $(\rho + (n - 1)$ expert networks. We aggregate the softmax of all the expert networks in line 13 and increment the count (so far evaluated expert networks). After all the expert networks are evaluated we take the corrected or re-evaluated output based on the highest softmax value in line 20. The final output is the accuracy of MS-Net. In Line 4 and 15 of the Algorithm 3 the Boolean dictionary list $visited[sub_1 : sub_C]$ ensures that we are not evaluating an expert for particular subset more than once. This optimization comes into play during situation where the index of two or more predictions of router are consecutive numbers.

2.8 Experiments

2.8.1 Datasets

To evaluate and validate the effectiveness of the network we leverage three public datasets, which are CIFAR-10 or C-10 (Canadian Institute For Advanced Research) [111], CIFAR-100 or C-100 [111], and F-MNIST (Fashion-Modified National Institute of Standards and Technology database). The CIFAR-10 dataset consists of 60,000 32X32 color images with 10 classes. Each class has 6,000 images. The dataset is divided into two parts with 50,000 images for training purposes and 10,000 images for testing [111]. The CIFAR-100 is just like CIFAR-10 but with 100 classes containing 600 images for each class. Among these 600 images for each class, 500 are for training and the rest 100 for testing. Moreover, the 100 classes are grouped into 20 super-classes. The F-MNIST database is a large database of fashion accessories. The database contains 60,000 training images and 10,000 testing images with 10 classes, where each image is 28X28 gray-scale image.

For saving the checkpoints for router and the expert networks we leverage the validation set that we construct from the original training set. The validation set consists of 10% of the samples from the original training set. The rest 90% from the original training set is leveraged to train the models.

2.8.2 Experiment settings

We implement MS-Net in the PyTorch framework [112], and perform all the experiments on single NVIDIA GeForce RTX 2080 GPU. The setting of hyper-parameters during training slightly vary across different datasets. However, for all datasets, we use Stochastic Gradient Descent(SGD) with momentum. We set the initial learning rate for all routers and experts to $lr = 0.1$ and momentum to 0.9. Hyper-parameters such as batch size, iterations and learning rate decay scheduler (γ) differ across routers, experts and datasets which are shown in the Table 2.2.

Table 2.2: **Training hyper-parameters for router and experts**

Network	Dataset	Batch size	Epochs	Steps
Router	C-10	32	300	50
	C-100	128	300	50
	F-MNIST	64	200	60
Experts	C-10	32	30	8
	C-100	16	25	8
	F-MNIST	64	30	10

2.9 Result Discussion

For the CIFAR-10 and CIFAR-100 dataset, we perform a detailed empirical study on the effect of variable β (of objective function Eq. (3.2)) on expert networks during the training phase. We also perform analysis on effect of ρ and n during the test phase. In addition, beside ResNet-20 we also provide performance of MS-Net with two well-known DNNs as backbone. However, in this chapter we perform all the empirical analysis and hyper-parameters search

with the backbone ResNet-20. Table 2.3 and 2.4 represent the performance of MS-Net (with ResNet-20 backbone) for CIFAR-10 and CIFAR-100, respectively.

In Table 2.7 we demonstrate the performance of individual expert network on subset class indices for dataset CIFAR-10 and F-MNIST. CIFAR-100 has 100 classes which make it difficult to interpret the performance of all 100 expert networks in a table. The table illustrates several key points about the MS-Net. Firstly, we observe that each of the expert network performs with remarkable score on its corresponding subset. That is, the performance of e_i on its corresponding subset sub_i, where $i = \{1, ..., C\}$, is very good (highlighted on Table 2.7). The performance of any expert networks on the whole set or on subsets assigned to other expert networks is relatively lower. Secondly, the performance of the Router R on each individual subset is significantly lower than that of the expert networks. However, when we execute the router and the expert networks together, they perform very well.

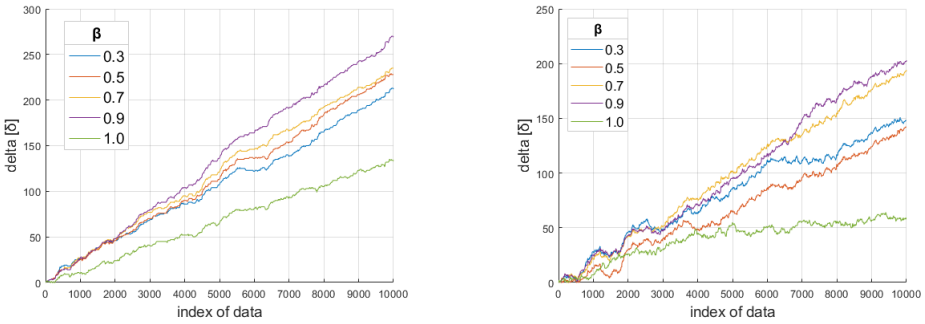
The empirical results for CIFAR-10 and CIFAR-100 suggest that, during training phase, fixing β to value 0.9 in the objective function tends to give relatively higher scores. To avoid redundant experiments, we perform rest of the training with β fixed to 0.9. Table 2.5 presents the performance of MS-Net for F-MNIST.

It is clear that with $\beta = 1$ in Eq. (3.2) we simply optimize the expert networks on training data sampled from subset class indices. On the contrary, with $\beta = 0$ we optimize the expert networks on the dataset comprising of all the available classes, which is analogous to the *naive Ensemble Learning (EL)* of DNNs. The optimal value for β has no theoretical bindings , rather it is dependent on the dataset. Expert networks trained with β in the range $0.3 \sim 0.9$ give near optimal classification scores. However, fixing β to either 0 or 1 during training degrades the performance scores, which implies that we should maintain a certain range for β while optimizing the proposed loss function. The variable n tells the experts up to how many top-n most probable prediction of router to further re-evaluate. For all the experiments, we re-evaluate up-to top-3 of router's prediction. The δ depicts the total number of samples correctly re-classified by the experts. *A positive δ value depicts the number of samples expert networks have correctly re-classified and a negative value for δ indicates the number of mis-classifications by the experts, or in other words, δ is the measurement of improvement in accuracy by our framework relative to the router network.* All the scores that we report In this chapter (figures and tables) are relative to the backbone network, which in this case is the router network. It is worth noting that, we use the online inference method during the testing. Thus for MS-Net, we make the prediction for a single observation at each iteration as oppose to batch processing. Due to modular nature of the framework, the online inference is the simplest implementation.

2.9.1 Performance on CIFAR-10

Table 2.3: **Performance on CIFAR-10 with variable probability distribution β. The backbone (ResNet-20) score is 92.68%, and the δ score depicts the number of samples correctly re-classified by MS-Net expert networks (relative to the backbone).**

β	ρ	n	acc. (%)	δ
0.3	2	2	93.70	+102
		3	93.65	+97
	3	2	94.74	+206
		3	94.60	+191
	4	2	94.80	+212
		3	94.85	+217
0.5	2	2	93.65	+97
		3	93.66	+98
	3	2	94.75	+207
		3	94.64	+196
	4	2	95.00	+228
		3	95.00	+228
0.7	2	2	93.60	+92
		3	93.58	+90
	3	2	94.83	+215
		3	94.75	+207
	4	2	95.03	+235
		3	95.10	+242
0.9	2	2	93.54	+86
		3	93.34	+66
	3	2	94.15	+147
		3	94.06	+138
	4	2	95.38	+270
		3	95.30	+261
1.0	2	2	93.30	+52
		3	93.09	+41
	3	2	93.85	+117
		3	93.83	+115
	4	2	94.01	+133
		3	93.90	+123



(a) Delta score for CIFAR-10

(b) Delta score for CIFAR-100.

Figure 2.5: **The objective function of MS-Net optimized with different probability distribution β. The *y*-axis depicts the δ scores (no. of samples correctly re-classified by experts). The *x*-axis represents the index of each data-points. Each point in the graph depicts the number of samples correctly re-classified (of the ResNet-20 router) by the experts till that particular data-index.**

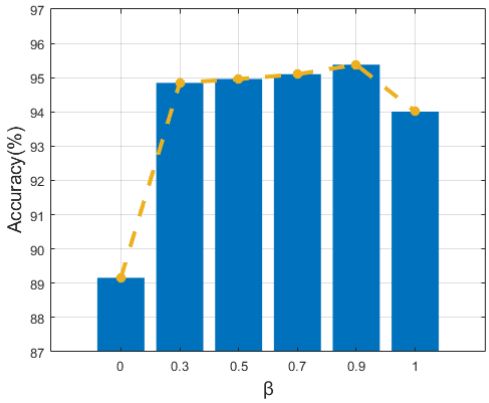


Figure 2.6: **Performance ($\%$) of MS-Net (with Resnet-20 backbone) on CIFAR-10 with variable distribution for β. It is evident that optimizing the objective function with two extreme values $\beta = 0$ or 1 does not provide with an optimal performance. Probability distribution ranging from 0.3 to 0.9 tends to give the near optimal performance.**

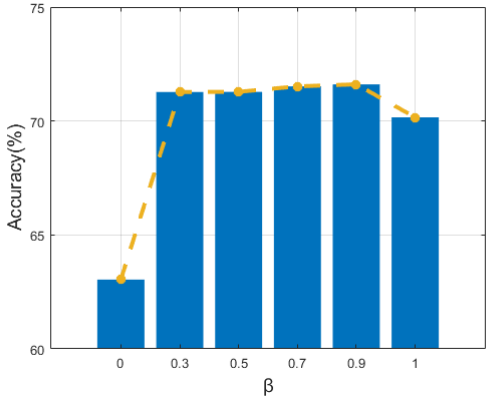
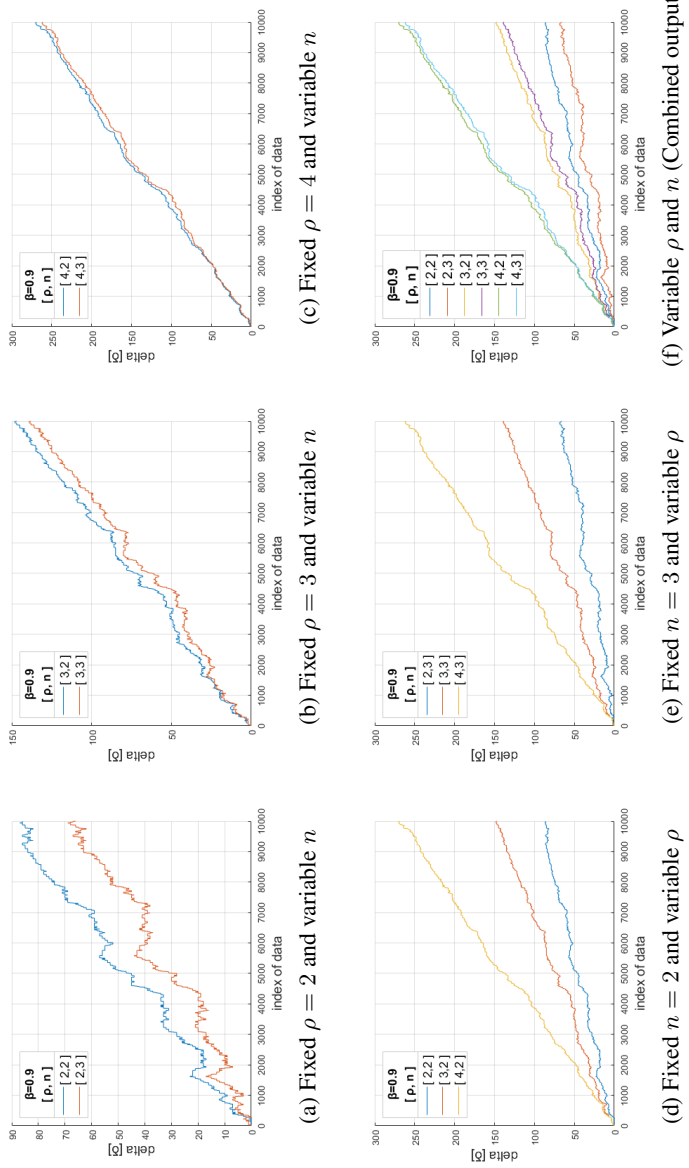


Figure 2.7: **Performance ($\%$) of MS-Net on CIFAR-100 (ResNet-20 backbone) with different distribution for β. The optimal score for distribution 0.3 to 0.9 holds for CIFAR-100 too.**

Table 2.4: **Performance on CIFAR-100 with variable probability distribution of β. The backbone (ResNet-20) score is 69.58%, and the δ score depicts the number of samples correctly re-classified by MS-Net expert networks (relative to the backbone).**

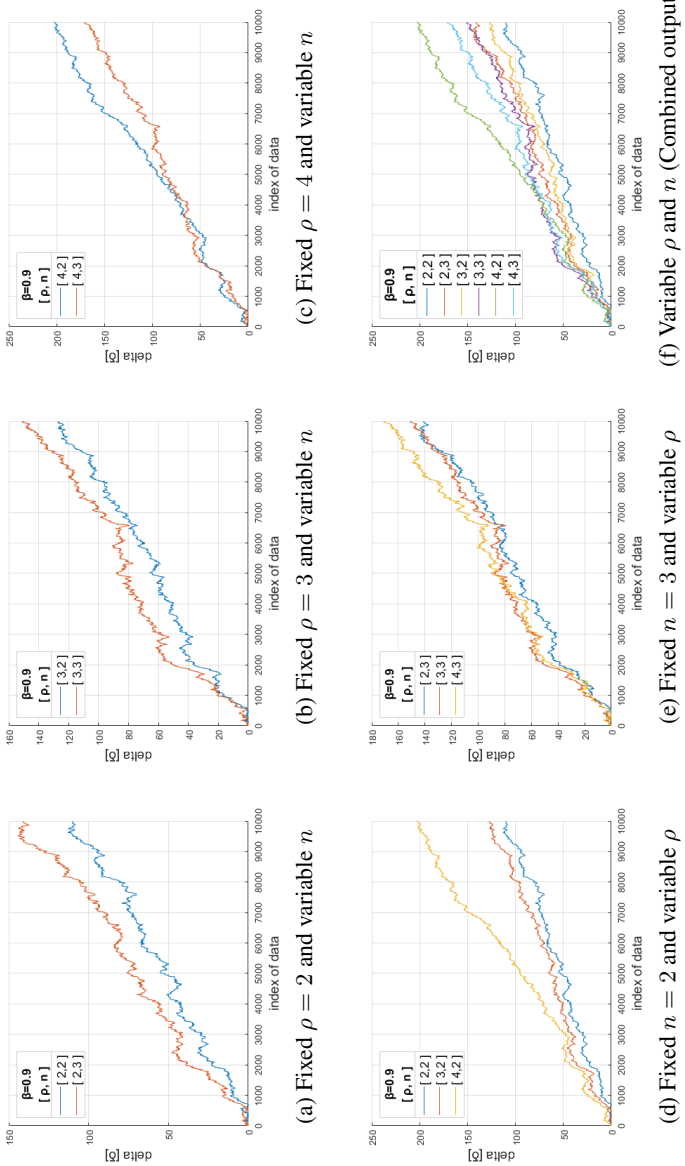
β	ρ	n	acc. (%)	δ
0.3	2	2	71.00	+132
		3	70.80	+127
	3	2	71.27	+170
		3	71.09	+152
	4	2	71.06	+150
		3	71.06	+150
0.5	2	2	71.07	+148
		3	71.05	+142
	3	2	71.10	+152
		3	71.28	+170
	4	2	71.05	+142
		3	71.25	+167
0.7	2	2	71.00	+136
		3	71.01	+142
	3	2	71.03	+144
		3	71.11	+152
	4	2	71.52	+193
		3	71.25	+167
0.9	2	2	70.68	+110
		3	71.00	+141
	3	2	70.85	+127
		3	71.09	+151
	4	2	71.61	+203
		3	71.28	+170
1.0	2	2	69.73	+15
		3	69.72	+14
	3	2	69.74	+16
		3	69.52	-5
	4	2	70.16	+58
		3	69.75	+17



(a) Fixed $\rho = 2$ and variable n (b) Fixed $\rho = 3$ and variable n (c) Fixed $\rho = 4$ and variable n

(d) Fixed $n = 2$ and variable ρ (e) Fixed $n = 3$ and variable ρ (f) Variable ρ and n (Combined output)

Figure 2.8: **The effect of variable n and ρ on CIFAR-10 during test phase: The first row (a,b and c) depicts the variation of δ score by keeping ρ fixed and changing variable n, i.e. it demonstrates how the network performs when we tweak the variable n. The second row (d and e) depicts performance of network keeping the n fixed while nudging variable ρ. The last Figure (f) summarizes all the δ score Figures (a, b, c, d and e) in a single graph for comparison.**



(a) Fixed $\rho = 2$ and variable n (b) Fixed $\rho = 3$ and variable n (c) Fixed $\rho = 4$ and variable n

(d) Fixed $n = 2$ and variable ρ (e) Fixed $n = 3$ and variable ρ (f) Variable ρ and n (Combined output)

Figure 2.9: **The effect of variable n and ρ on CIFAR-100 during test phase: The first row (a,b and c) depicts the variation of δ score while keeping ρ fixed and changing variable n, i.e. it demonstrates how the network performs when we tweak the variable n. The first two figures of second row depict the performance of network keeping n fixed while nudging variable ρ. The last Figure (f) combine all the Figures (a, b, c, d and e) for depicting the contrast clearly**

Table 2.3 represents the performance of MS-Net for CIFAR-10. From our experimental results, we can deduce the following key observations.

1. As we increase the value of ρ the accuracy increases. Refer to Figure 2.8 (d), (e) for graphical illustration of this phenomenon. However, for the case of CIFAR-10 increasing top-n beyond the value 2 does not improve the performance further (Figure 2.8 (a), (b) and (c)).

2. We observe gradual improvement in performance for the expert networks trained with increasing β which can be confirmed by Figure 2.5 (a) and 2.6. The score gets lowest when we train the expert networks with $\beta = 1$. This phenomenon suggests that training the expert networks solely on its subset classes ($\beta = 1$ i.e. clamping $\mathcal{X} = 1$ in the objective function during the whole training process) does not improve performance, rather degrades. This degradation of result occurs due to imbalanced logit value in the last layer since the expert networks do not encounter any training data from rest of the classes (classes apart from the subset classes) during the training phase. Training these experts on the whole set of data alternatively within the optimal range of probability distribution substantially improve the performance. This method acts as a very effective regularization, as it prevents the experts from over-fitting on the dataset from subset classes. A graphical overview of the effect of the probability distribution β is presented in the bar chart Figure 2.6.

3. In our experiment, we obtain the best score (with ResNet-20 backbone) for CIFAR-10 (95.38%) with $\rho = 4, n = 2$ and $\beta = 0.9$. The δ score with the mentioned parameters is $+270$, which means, integration of the expert networks with router further improves the performance by $+2.70\%$. In other words, the router with a backbone network ResNet-20 has a top-1 accuracy of 92.68% and by integrating the experts for further re-evaluation, we levitate the top-1 score by $+2.70$ i.e. 95.38%.

2.9.2 Performance on CIFAR-100

Table 2.4 represents the result for CIFAR-100. For CIFAR-100, the same hyper-parameters $\rho = 4, n = 2$ and $\beta = 0.9$ give relatively high score of 71.68%. We can observe from the Table 2.8 that router's top-1 performance (ResNet-20) for CIFAR-100 is only 69.58%, and with the integration of the experts the performance increases by 2.48%. This phenomenon suggests that as we increase ρ and n we are more likely to get higher accuracy. The scores in Table 2.8 depict that MS-Net has relatively lower score on CIFAR-100 compared to CIFAR-10 and F-MNIST. This phenomenon is also observable for other state-of-the-art DNN (refer to Table 2.9). The probable reason for such low performance is mostly due to fewer amount of data per class in CIFAR-100. While CIFAR-10 has 6000 samples per class, CIFAR-100 has only 600 samples. This problem has been mitigated to a certain extent recently by leveraging large scale Transfer Learning (ImageNet pre-trained) [40], learning data augmentation policy or Auto-Augment (AA) [113], task-specific NAS with Transfer Learning (TL) [36, 114], Neural Architecture through hybrid online TL with multi-objective evolutionary search procedure [115] and so on. The MS-Net proposed in this study also has a significant improvement in performance compared to the backbone networks. We may expect further improvement if we introduce TL and other techniques described above.

Table 2.5: **Performance on F-MNIST with $\beta = 0.9$. The backbone (ResNet-20) score for F-MNIST is 95.22%, and the δ score depicts the number of samples correctly re-classified by MS-Net's expert networks (relative to the backbone).**

Dataset	ρ	n	acc. ($\%$)	δ
FMNIST	2	2	95.80	+60
		3	95.96	+74
	3	2	95.80	+60
		3	96.77	+156
	4	2	96.02	+80
		3	96.77	+156

Table 2.6: **Performance of individual expert network on all the subsets for CIFAR-10. The highlighted parts depict the score of each expert on its corresponding subset class index obtained through the Round Robin partition. The last column S depicts the performance of experts on whole set of data. The row with R represents the performance of router network on individual subset. In this table, all the experts and the router network have *ResNet-20* backbone. The ρ is 4 which also depicts the cardinality of each subset. We train all experts with $\beta = 0.9$.**

Dataset	net \ sub	sub_0	sub_1	sub_2	sub_3	sub_4	sub_5	sub_6	sub_7	sub_8	sub_9	S
CIFAR-10	e_0	97.55	95.125	89.6	63.325	58.07	60.15	69.15	75.87	83.27	91.17	75.13
	e_1	87.52	98.22	80.22	71.6	62.27	57.57	66	62.32	72.23	76.38	72.65
	e_2	80.05	92	99.17	83.52	65.55	55.32	45.97	47.85	58.92	70.2	70.54
	e_3	76.8	82.2	83.77	97.55	90.2	87	84.47	79.07	81	74.7	83.05
	e_4	70.87	74.25	74.75	83	97.05	95.42	92.82	89.5	88	81	84.5
	e_5	62.52	62.65	68.7	76.62	92.57	98.7	92	86.2	79.2	71.47	78.88
	e_6	61.27	62.42	62.37	71.27	84.7	91.25	98.9	91.25	82.35	73.5	77.77
	e_7	73.57	71.27	71.87	77.27	87.65	92.62	96.52	98.27	91.72	82.7	83.43
	e_8	81	77.92	73.25	78.17	84.45	87.4	92.47	96.27	98.47	91.23	85.33
	e_9	87.97	78.62	72.47	65.57	73.12	75.05	81.6	91.23	92.47	98.8	81.73
	R	90.07	90.02	87.87	89.97	92.85	93.1	94.87	94.15	94.62	93.15	92.68

Table 2.7: **Performance of individual expert network on all the subsets for FMNIST. The highlighted parts depict the score of each expert on its corresponding subset class index obtained through the Round Robin partition. The last column S depicts the performance of experts on whole set of data. The row with R represents the performance of router network on individual subset. In this table, all the experts and the router network have *ResNet-20* backbone. The ρ is 4 which also depicts the cardinality of each subset. We train all experts with $\beta = 0.9$.**

Dataset	net \ sub	sub_0	sub_1	sub_2	sub_3	sub_4	sub_5	sub_6	sub_7	sub_8	sub_9	S
FMNIST	e_0	98.67	85.05	80.30	60.57	60.77	77.35	77.85	97.87	97.27	96.17	82.32
	e_1	90.52	97.10	95.35	75.92	76.40	76.45	77.27	86.40	86.62	87.52	84.48
	e_2	91.52	96.90	98.50	80.80	80.25	80.27	81.82	92.15	91.40	92.20	87.90
	e_3	60.90	80.52	80.70	98.72	97.35	93.97	93.05	74.80	74.55	60.52	80.11
	e_4	40.12	62.60	62.55	82.25	97.37	94.60	90.90	70.05	70.10	51.55	72.61
	e_5	65.67	70.70	71.60	83.22	88.52	98.32	96.65	82.55	81.05	68.70	80.50
	e_6	74.95	70.95	70.50	76.90	80.47	96.65	98.75	86.42	86.87	78.15	81.81
	e_7	81.97	77.47	74.30	67.45	77.97	81.55	84.77	98.55	98.12	92.12	83.23
	e_8	75.22	71.95	60.27	51.45	45.60	45.01	57.67	74.74	99.67	91.35	84.66
	e_9	90.10	85.35	83.95	80.20	87.87	91.20	92.60	96.22	98.32	99.87	90.57
	R	94.70	95.35	95.37	93.40	94.35	95.67	94.95	96.52	96.72	95.15	95.22

Table 2.8: **Performance of MS-Net for CIFAR-10 (C-10), CIFAR-100 (C-100) and F-MNIST. The first section depicts the score of backbone networks itself, which also indicates the performance of routers. The second section represents the performance of our proposed framework (MS-Net) equipped with different backbone networks. We train MS-Net with different backbone networks with exact same hyper-parameters.**

Type	Methods	C-10	C-100	F-MMNIST	# Param. (M)
Backbone	ResNet-20 [109]	92.68	69.58	95.22	0.269
	GoogleNet [88]	92.93	78.03	93.70	6.2
	MobileNet [108]	94.43	68.08	95.00	2.36
MS-Net framework	**MS-Net (ResNet-20)**	**95.38**	**71.61**	**96.77**	**2.95**
	MS-Net (GoogleNet)	**97.01**	**85.05**	**96.80**	**55.80**
	MS-Net (MobileNet)	**96.01**	**78.03**	**96.80**	**21.24**

Table 2.9: **Performance of state-of-the-art networks for CIFAR-10 (C-10), CIFAR-100 (C-100) and F-MNIST. The first section reports the score for relatively larger DNN, which we term as Type-I. The second section i.e. the Type-II are the networks that share relatively same computational capacity and parameters as MS-Net. Type-II section also includes automated learned architectures (without human intervention) through evolutionary search, reinforcement learning and so on. The table is divided for ease of comparison and contrast.**

Type	Methods	C-10	C-100	F-MMNIST	# Param. (M)
Type-I	GPIPE + TL [1]	99.00	91.30	-	556
	Shared WRN [116]	97.47	82.57	-	118
	SGDR WRN-28-10 3 runs $\times$ 3 snapshots [117]	96.75	83.36	-	329
	SGDR WRN-28-10 16 runs $\times$ 3 snapshots [117]	96.86	83.79	-	1752
Type-II	Res2net-29 [118]	-	83.44	-	36.90
	PyramidNet+ShakeDrop+ Fast AA [119]	98.30	88.30	-	-
	Wide GatedResNet [120] (4,10) + Dropout	96.35	81.73	-	36.50
	DenseNet [27]	96.54	82.62	95.40	20
	Inception [28]	-	77.19	-	22.30
	VGG16-BN [121]	92.64	72.93	93.50	34
	ResNet-101 [109]	93.75	77.78	94.9	42.70
	ResNet-152 [109]	95.38	77.61	-	58.30
	FractalNet [122]	95.40	76.27	-	38.60
	NAS V3 [38]	96.35	-	-	37.40
	NASNet-A (7 @ 2304) [37]	97.03	-	-	27.60
	EfficientNet-B7 + NAS + TL [36]	98.90	-	-	64

2.9.3 Performance on F-MNIST

Table 2.5 represents the result of MS-Net on F-MNIST. In order to avoid redundant experiment the same hyper-parameters that give optimal score for CIFAR-10 and CIFAR-100 are set during the training. The network achieves score of 96.77% on F-MNIST test set. The delta score is +156, which is relatively higher. To our knowledge, the best score for F-MNIST was 96.30%, reported by Wide-ResNet-28-10 with Random Erasing data-augmentation [123]. Thus, MS-Net achieves a state-of-the-art score for this particular dataset.

2.10 Hyper-parameter recommendation

2.10.1 Optimal value for ρ, top-n evaluation and β

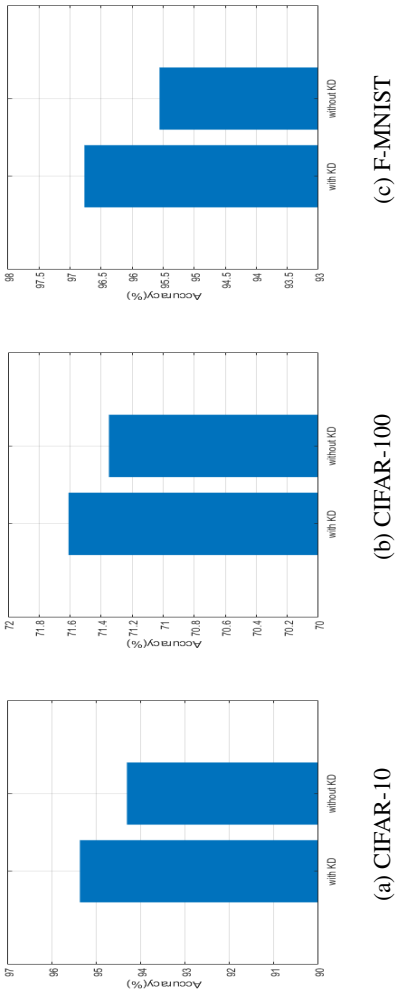
A very common intuition is that as we increase the n of the router we can score (at best) as good as the router's top-n prediction score. In practice, increasing n beyond the value 2 does not substantially improve the performance, however, increasing the value of ρ gradually improves the accuracy of the network. For numerical comparison please refer to the Table 2.3 and 2.4. In addition, Figure 2.6 and 2.7 represent the effect of ρ and n for CIFAR-10 and CIFAR-100 respectively. An interesting observation from the Figure 2.8 is that, as we increase ρ the performance levitates dramatically, on the contrary, increasing n does not increase accuracy with a big margin. This is also the case for the CIFAR-100 dataset. The graphs in Figure 2.9 indicate that, for a fixed value of ρ, increasing n further increases the accuracy, but not with a substantial margin. Although in this literature, our experiment is limited to $\rho \leq 4$, we can anticipate that for CIFAR-100 further increasing ρ will increase the accuracy. The reason is that CIFAR-100 is relatively difficult dataset with a large number of classes. Thus, from the above observations we can conclude with following guidelines for optimal parameters selection.

1. Evaluating till top-2 probable predictions of the router will suffice. This statement is true at-least for all the dataset we have explored so far.

2. Setting the redundancy rate variable ρ to 3 provides with a comparable classification score for all cases. We know that increasing ρ implies that we have more expert networks for each class. Thus, in situation where we have enough resource budgets, we can increase the variable ρ beyond 3 for more redundant expert networks and accuracy.

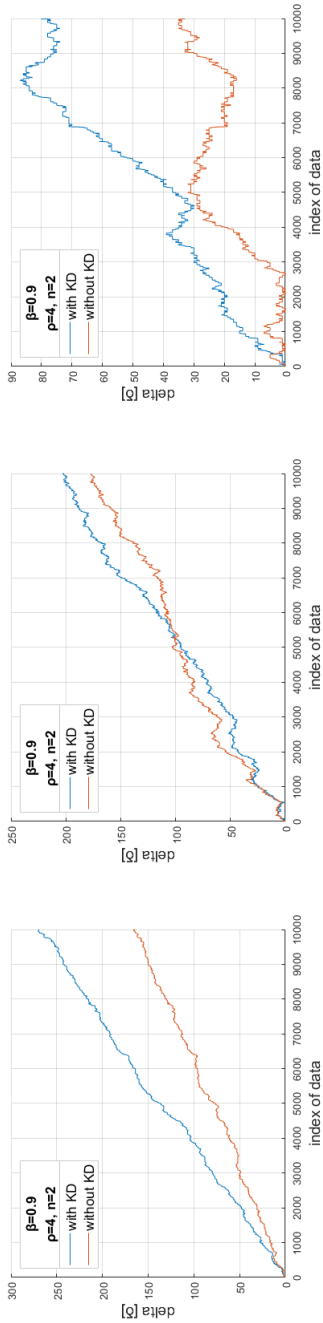
3. During the training phase the variable β of objective function (Eqn. (3.2)) plays a crucial role in performance. Although there are no fixed value or theoretical bindings for β, we recommend to avoid fixing β to two extreme values i.e. 0 and 1. Optimizing the expert networks keeping β in range of 0.3 to 0.9 tend to give optimal score.

Thus, to keep the experiments simple, the training of MS-Net (implementation with different backbone networks) in the rest of the paper will confine to the above mentioned hyper-parameters.



(a) CIFAR-10 (b) CIFAR-100 (c) F-MNIST

Figure 2.10: **Performance of the MS-Net trained with and without KD loss.**



(a) CIFAR-10

(b) CIFAR-100

(c) F-MNIST

Figure 2.11: **Contrast of δ scores: The figure represents the δ score difference for MS-Net trained with and without the KD loss.**

Table 2.10: **Contrast of performance: The table represents the score differences for MS-Net (ResNet-20 backbone) trained with and without the KD loss.**

Dataset	KD	Acc.(%)	Impr. delta (%)
CIFAR-10	YES	95.38	1.06
	NO	94.32	
CIFAR-100	YES	72.00	0.65
	NO	71.35	
FMNIST	YES	96.77	1.21
	NO	95.56	

2.10.2 Effect of Knowledge Distillation

KD plays a vital role in training the expert networks. Earlier in the section 2.6 we proposed a slight variant of objective function (Eq. (3.2)) where we replace the KD term with simple cross entropy loss term (Eq. (2.7)). We train MS-Net on CIFAR-10, CIFAR-100 and F-MNIST leveraging the loss $Loss^{wokd}$ depicted in Eq. (2.7). The hyper-parameters are exactly identical to the experiments done with KD loss. The results show that, expert networks of MS-Net optimized without KD loss drops in accuracy with a considerable margin. Figure 2.11 (a), (b) and (c) and Table 2.10 depict the contrast between the δ scores of MS-Net trained with and without the KD loss for CIFAR-10, CIFAR-100 and F-MNIST respectively. In Figure 2.10 and Table 2.10 we show the contrast for the classification accuracy. Using distillation in the loss term assists the expert networks in retaining the existing knowledge of the router. This also ensures the experts are at-least as good as the router network in the worst case scenario. In other words, the expert networks are less prone to mis-classify samples that are already correctly classified by the router networks.

2.11 Comparison to state-of-the-art results

In this section we provide a brief comparison of MS-Net to the performance of existing state-of-the-art DNNs on CIFAR-10, CIFAR-100 and F-MNIST dataset. For comparison we provide two tables, Table 2.8 and Table 2.9 , wherein, first table depicts the performance of backbone networks (routers) and MS-Net frameworks (with different backbone networks), and second table represents the performance of state-of-the-art DNNs. For the ease of comparison and illustration, we divide the benchmark Table 2.9 into two types, where Type-I represents the network with large number of parameters, Type-II with parameters and computational resource almost similar to our proposed framework. The Type-II networks also include several architectures learned by using computationally expensive methods (e.g. evolutionary search and reinforcement learning) equipped with transfer learning (TL). For bench-marking, we refer to the site [124].

We can observe from the Table 2.8 that, MS-Net framework elevates the classification accuracy with a significant margin relative to the backbone router. Comparing MS-Net framework with Type-I networks from Table 2.9, the network actually performs with a neck-and-neck scores. However, compared to Type-II networks (approximately similar parameter counts) MS-Net performs with high score relative to most of the networks. The highest score that we obtain

so far is with the backbone network *GoogleNet*, leveraging at most 55.80M parameters (Table 2.8). This high score and setup undeniably come with a trade-off of more computational resources and parameter budget.

Most recently, researchers have been trying to find the best structure using evolutionary algorithms, reinforcement learning algorithms, and so on, and some very interesting results have been obtained [37, 38, 93, 115]. For example, for the database CIFAR-10, the best performance obtained so far is 98.9% (refer to Type-II section of Table 2.9) and the model's training parameter number is 64M [36]. However, based on the *'no free lunch theorem'* [125], an optimal model is usually fine-tuned for some specific database, and the model may not be useful for solving other problems. Even for the same problem with more observed data, to preserve the best performance, we have to use a very expensive process to re-design the model. On the other hand, the MS-Net structure proposed in this study is very simple, and can leverage the performance of any existing state-of-the-art models by increasing the inference cost slightly. In this sense, MS-Net can be a good starting point for solving various problems.

2.12 Summary

In this chapter, we have proposed a modular neural network architecture termed as the MS-Net (Modular Redundant Network). For a C-class classification problem, the network consisted of a router network and C expert networks. In summary, the key idea of the research has been to further re-evaluate the top-n most probable predictions of the router by leveraging these expert networks. To effectively train these expert networks we have proposed a stochastic objective function equipped with the knowledge distillation technique that facilitates alternative training on a subset of expert data and whole set of data. This alternative training has been regulated by clamping a Bernoulli random variable to each of loss function term. We have constructed the subsets of data systematically by Round-Robin fashion. As a result, it has provided us with a mean to control the redundancy of each class in the set of subsets, which have also allowed us to know which expert network is a specialist on which subset (thus we have more interpret-ability). We have shown that, with a very limited parameter budget and simple DNN as backbone, our network has achieved performance comparable or sometimes equivalent to more complex DNNs.

Chapter 3

Optimized MS-Net: Stabilization of the MS-Net Based on Inter-Class Correlation

The chapter presents the successor of Chapter-2 MS-Net with several optimization implementation. In our original proposal in chapter 2, MS-Net is constructed based on a Round-Robin dataset partition with controlled redundancy among the subsets of classes. In this chapter, we propose a novel way for reducing the inference cost by performing Inter-Class-Correlation (ICC) analysis through calculating the joint-probability of appearance of top-2 pair of classes in router's prediction. Next, we construct subset of classes on the most frequently occurring class pairs from the ICC and train experts on those subsets. We do not explicitly enforce redundancy in these subsets, thus during inference, only one expert is leveraged per sample. In best case scenario we do not require experts as the predictions of router will suffice. We validate O-MS-Net on four popular datasets which are CIFAR-10, CIFAR-100, FMNIST and SVHN. With parameter budget of only $2.41M$ and ResNet-110 backbone O-MS-Net achieves 96.90%, 76.50%, 96.90% and 98.01% on CIFAR-10, CIFAR-100, FMNIST and SVHN respectively, which is comparable to the original MS-Net performance.

3.1 Introduction

Modular design in machine learning is an effective and common approach to boost the performance. Most common approaches to achieve modularity in learning systems are Random forest [106], Bagging [63], Boosting [107] and so on. These approaches are collectively known as the Ensemble [126]. While ensemble techniques with any backbone learning model have shown promising results and good generalization capability they are quite expensive when implemented with neural network backbone. Modern neural networks such as Convolutional Neural Networks (CNNs) are very deep and can be computationally prohibitive when leveraged in a large ensemble system. Moreover, the performance boost that we actually obtain through the ensemble does not provide us with a optimum trade-off between accuracy and computational resources. To overcome these limitations design properties such as sub-task partitioned based neural networks [127], adaptive neural network inference [128], gating mechanism [47,73,99,100] and so on were proposed. These techniques moderately reduce the number of neural networks to be evaluated with substantially less compromise in performance as oppose to full-blown ensemble method. Modern modular neural networks are not only limited to the modular design, these networks are also now equipped with powerful transfer-learning techniques such as, Knowledge Distillation (KD) [45], co-distillation [129], mutual distillation [59] and so on. These sort of transfer learning assist each of the module of the whole system to take advantage of each other

during the learning phase.

With all these virtues in mind we have proposed MS-Net discussed in the previous chapter. MS-Net is a modular neural network framework with two primary module known as router and experts. These design router/expert paradigm is not new and have been previously observed in various remarkable literature's [45, 47, 73, 99, 100]. However, MS-Net provides with two key contributions which make it suitable for any backbone network.

- A systematic and efficient way to partition the whole task into set of sub-task, where there is a limited redundancy of any class in those set of sub-task. This redundancy can be controlled explicitly by the redundancy variable ρ based on the computational availability.
- stochastic loss function optimizes the expert networks to perform with high accuracy on corresponding subset of classes while generalizing moderately of rest of the classes. The stochastic nature of the loss function acts as a powerful regularizer for the expert network.

Keeping these key properties of MS-Net in mind we propose an improvement for MS-Net. Below we present the contributions that address the limitations of MS-Net in this chapter:

1. *The experts of MS-Net does not take into consideration about router's weakness and strength.* This implies that, there are expert networks being evaluated for samples or input which are very easy and could have been easily handled by the router (i.e. visually easy samples). The chapter proposed an optimization technique for original MS-Net that addresses this issue and significantly reduces the number of expert network evaluation while maintaining performance close to the original MS-Net.

2. Another issue that the chapter tries to address, *is it always desirable to evaluate multiple expert network for a single prediction?* Addressing this question is important since it is intuitive that having multiple expert networks for a single class can sometimes have negative effects, such as conflict or disagreement among the experts. Needless to say, there are regularizer or objective function which allows co-learning (such as distillation [129]) among the experts to overcome this limitation. We would like to address this issue in our future work. However, in this chapter we tackle this issue by performing Inter-Class-Correlation (ICC) analysis through calculating the joint-probability of appearance of top-n classes in router's prediction. In this study we limit the value of n to 2 and 3. For the proposed optimized MS-Net, for every single input sample we evaluate only one expert network and achieve performance comparable to the original MS-Net.

3.2 Prior arts

In this section we will give a brief overview on MS-Net, and related modular neural networks. The concept of modular neural network architecture was introduced around two decades ago in literature [47] known as the mixture-of-experts. The architecture consisted of a tree like structure with a gating network at non-terminal node and set of experts sitting in leaf node. Modular architecture with gating mechanism for phonetic classification can also be observed in [73, 99, 100].

Modular network topology was popular and often practiced for speech recognition. However, modular network design approach for visual object recognition was relatively sparse until recently Hinton [45] proposed a special type of network ensemble which can learn to classify fine-grained classes. These fine-grained classes are often mis-classified by the generalist model (also known as the router network). In addition, unlike the mixture-of-expert the proposed framework can be trained in parallel.

A large and generalized modular framework was proposed in [130]. The network employed thousands of feed-forward sub-networks (or experts) with a trainable gating network. The output

of gating network is sparse as it select few experts during the inference. The network was primarily evaluated for language modeling and machine translation task. However, the design of the network allowed to be adopted for other domain as well.

Modular Selective Network or MS-Net [127] is also a modular network with router and expert module. The framework leverages the expert network to reevaluate routers top-n prediction. In the original literature the value of n varied from 2 to 3. The MS-Net framework first partitions the dataset into C number of subsets. Each of the subsets has a controlled overlapping of classes. This overlapping degree is termed as the redundancy rate ρ which can be explicitly controlled before training and testing as well . As we increase the value of ρ the occurrence of classes also increases proportionally in each of the subset. For each subset of classes the framework has an expert network (thus a total of C expert networks). During the inference phase, it is possible to control how many expert we want to deploy for the routers top-n predicted classes. Thus, during inference the prediction is always performed through expert networks weighted by routers confidence. The literature [127] provides a detailed ablation study on the effects of ρ, routers top-n and knowledge distillation on performance.

3.3 Optimizing MS-Net

3.3.1 Recapping MS-Net Data Partitioning Technique

In the original MS-Net the construction of subset of classes required no prior knowledge about router strength or weakness, thus it did not require router evaluation. The dataset or the concept was partitioned systematically in to several subsets through the Round-Robin method. A redundancy variable known as the ρ was introduced which explicitly controlled the redundancy of any particular class among the constructed subsets. In other words, as ρ increased the occurrence of dataset from any particular class will increased proportionally. MNNs implemented based on this data partitioning technique showed boosted accuracy.

Now, this approach is very suitable in situation where we do not have prior knowledge about the i) difficulty of dataset; ii) strength and weakness of the router network. In this chapter we put emphasis on the aforementioned factors and try to reduce number of expert networks during the inference phase, while maintaining performance comparable to the original MS-Net.

3.3.2 Key Idea

In case of Optimized MS-Net (O-MS-Net) we first evaluate the router to obtain a prior knowledge about the difficulty of dataset. This approach is not new. Research [45] have actually first leveraged the router (known as the generalist in original literature) to get prior knowledge about the confusable set of classes in dataset. The literature performed K-Means clustering on the co-variance matrix of the predictions of the generalist model. Later, several neural networks were trained on the obtained clusters of classes to obtain specialists. HydraNet [67] which is a special type of MNN with shared backbone leveraged the clustering technique to group visually similar classes, and assigned each group to some particular CNN branches.

Our approach is similar to prior research in a sense that, we try to group classes based on the visual similarity. However, we do not perform any clustering or projection of data on latent space. Clustering although is simple to implement and visually assuring it has several issues. First, deciding the number of clusters is difficult, and sometimes it can lead to imbalanced class partitioning. Cluster based partitioning can also welcome redundancy which we do not encourage in the current implementation motivation.

In our partitioning implementation we leverage simple softmax information from the router network to construct consusable subset of classes. Softmax is leveraged mainly for the multi-class problem where each class is assigned certain confidence (probability) based on the output logits of neural networks for the corresponding input. The class with the highest confidence is

assumed to be the correct answer. *What about classes with second highest confidence or the third?* Research [45] has demonstrated that the softmax output of any neural network is a very valuable piece of information. Softmax output of neural network implicitly depicts the visual similarity among the classes. Let us visually demonstrate this simple statement through the Figure 3.1. Lets consider the left example from the Figure 3.1. The ground truth label for this image is *dog*. However, the network predicted the image as *cat* with a 31% confidence, and with 29% the network assumes it a *dog*. Truth be told, both cat and dog actually shares various visual similarities which have been reflected in the networks softmax output. In this case, the network no only convey us the information about visual similarities, it also show its weakness in classifying this visually similar pair of classes. Thus, we can use this simple and valuable piece of information to construct pair-wise subset of dataset. This approach only considers those tuple of classes which are visually confusing to router. This summarize our key idea of optimization.

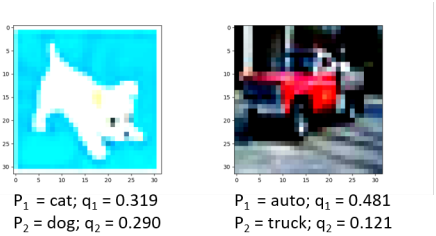


Figure 3.1: **Illustration of softmax output for two instances from CIFAR-10 dataset. We leverage ResNet-20 to generate the softmax output**

3.3.3 Inter-Class-Correlation based Data Partition

The section provides detailed procedure and formal explanation of the aforementioned data partition technique. We term this technique as Inter-Class-Correlation (ICC) based data partition.

Let us assume the router network as $y = R(.) : \mathcal{D} \rightarrow \mathcal{T}$, where $\mathcal{D} = \{d_i | i = 1, ..., N\}$ is the dataset and $\mathcal{T} = \{t_i | i = 1, ..., N\}$ is the corresponding teacher signal. Here N is the number of training data. The output of the router network $R(.)$ is the softmax defined in Eq. (3.1), where we obtain the probabilities say $Q = \{q_1, ..., q_C\}$ for all C classes as follows.

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}. \tag{3.1}$$

Here, $Z = \{z_1, ..., z_C\}$ are the logit scores for the corresponding classes. The output Q is assumed to be sorted and we take the corresponding sorted argument $\mathcal{P} = \{p_1, .., p_C\}$. These sorted arguments are the prediction class by the router ordered based on the confidence. Now our assumption is that, for any input, the top-2 (we consider $n = 2$ for top-n in this formal definition) predictions based on the softmax confidence will be visually similar or have close resemblance. Our target is to construct subsets of tuples from top-2 classes which are frequently mistaken or confused by the router network. To obtain these subsets we will leverage a set of validation data say $\mathcal{D}_{val}$ and corresponding ground truth $\mathcal{T}_{val}$ (data that the router has not encountered during the training phase). During validation, for any input d we count the co-occurrence of the top-2 predictions p_1 and p_2 to ultimately get $Pr((p_1, p_2)|(d, t))$, which depicts the join-probability of appearance of top-2 predictions p_1, p_2 by the router $R()$ given ground truth t associated with the input datum d. Here, p_1 and p_2 can take any value from $\{1, .., C\}$, where C is the number of classes available in the dataset. We also assume that either of p_1 or p_2 is a correct answer. Conveniently, we can also extend the ICC for the top-2 to top-3 by simply extending the calculation to $Pr((p_1, p_2, p_3)|(d, t))$. In the case of considering top-3 predictions,

we have higher probabilities of having correct answers in the chosen tuples. Moreover, for large datasets with large number of classes the number of confusable classes can be huge. In such case the straight forward solution is to increase n of the top-n while calculating the ICC. In this way, co-occurrence of more confusing classes (more than two) can be encoded. It should be noted that the more we increase the value of top-n the more classes the experts should focus on. Besides, the proportion of correct answer in the top-n varies with the difficulty of the dataset (refer to Table 3.4)

Once we complete calculating the join-probability of appearance of top-2 predictions for the validation set we construct the square matrix J of *join-probability* which takes shape of C, C. The matrix takes the following form:

$$J = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,C} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ c_{C,1} & c_{C,2} & \cdots & c_{C,C} \end{pmatrix}$$

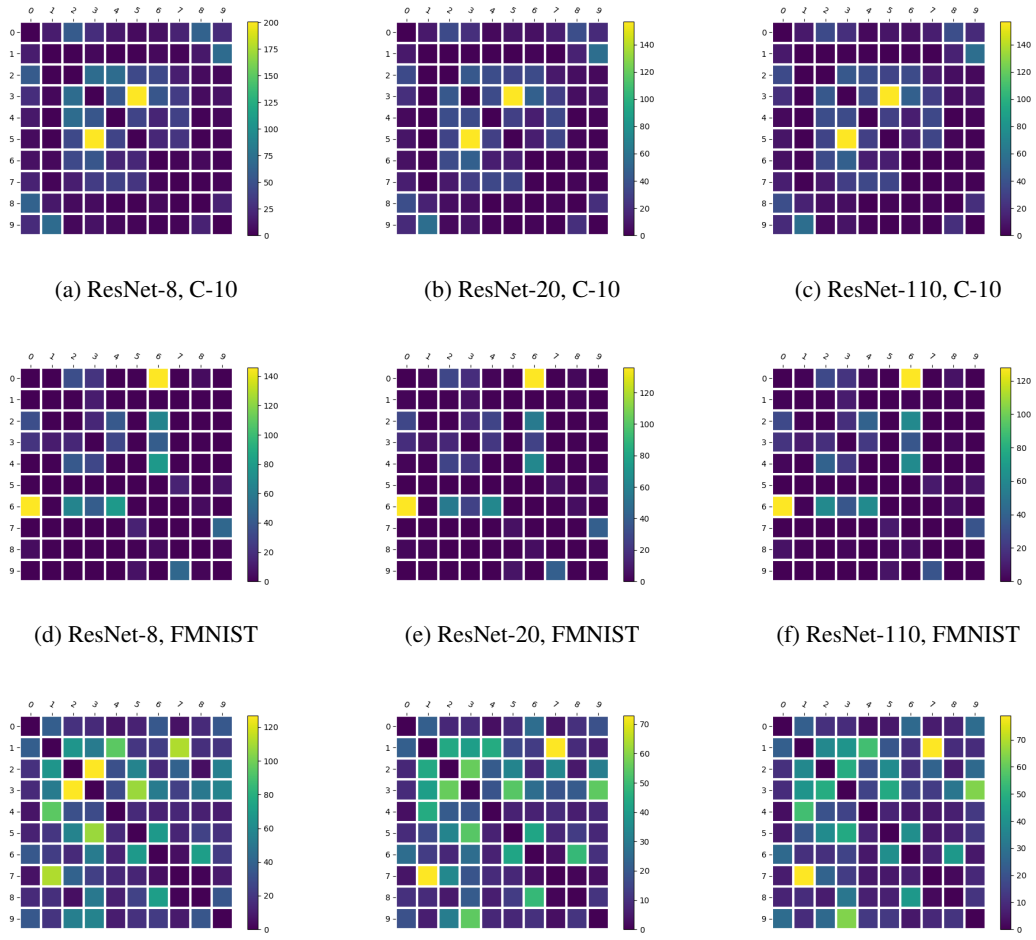
Each cell $c_{i,j}$ in J is basically the join-probability (un-normalised) of class-index i and j co-occurring together in router's top-2 prediction for input t. More formally, each cell $c_{i,j} = Pr(i, j|(d, t))$.

For ease of understanding let us take a visual example of matrix J. In Figure 3.2 we illustrate J for 3 well-known public dataset, CIFAR-10, FMNIST and SVHN produced by 3 variants of ResNet networks i.e. ResNet-8, 20 and 110. We will discuss more about these dataset in the later section. For now, let us discuss the un-normalized version of J calculated based on the output of ResNet-8 for CIFAR-10 dataset. From Figure 3.2 (a) it is quite easy to deduce that class index pair $\{3, 5\}$ are frequently mistaken, i.e. class index 3 is often confused with class index 5 and vice-versa. If we refer to Table 3.1 we can see that the class index pairs are cat, dog. This observation actually supports our discussion in previous section 3.3.2. We depict some of the confusable pairs of classes (for top-2) such as $\{\{3, 5\}, \{2, 3\}, \{2, 4\}, \{1, 9\}..\}$ that we obtain from Figure 3.2 (a), (b) or (c) in the Table 3.2. We also include instances of top-3 frequently co-occurring tuples of classes in the same Table 3.2. Such as, $\{\{\{3, 4, 5\}, \{2, 3, 5\}, \{3, 5, 7\}..\}$. A very important observation from Figure 3.2 is that, the confusable class pairs for any dataset are consistent regardless of network architecture. The only contrast is that, bigger networks make less mistakes relative to the smaller one on those confusable pair of classes. This is an implicit indication that ICC based data partition is consistent and can be reliably leveraged to obtain the subset classes.

Once we have these confusable pairs of classes, we prepare a set of subsets say, $S = \{sub_1, sub_2, .., sub_K\}$. Here K is the total number of subsets we consider. Each element of S is a confusable pair (in case of top-2) or tuple of class indexes (in case of top-3) arranged in sorted order (sorted based on their count of join probabilities). The cardinality of the element of S depends on the value of n of top-n (refer to Table 3.2). In addition, there are no theoretical support and definite lower or upper bound for the value of K, i.e. how many experts to construct. This value is set based on factors such as, the number of difficult classes within a dataset, amount of computational resource at our disposal during training and so on. In the latter section of the chapter we provide some detailed insights on the value of k.

3.3.4 Training Phase

For **router** and expert networks, original MS-Net leverages ResNet-20, GoogleNet and MobileNet as the backbone. However, GoogleNet and MobileNet can be over-parameterized, complex and heavy for O-MS-Net expert networks (expert networks in this research are specialist for only two class and generalist on rest). Thus, in this research we leverage ResNet-8 (the light-



(a) ResNet-8, C-10 (b) ResNet-20, C-10 (c) ResNet-110, C-10

(d) ResNet-8, FMNIST (e) ResNet-20, FMNIST (f) ResNet-110, FMNIST

(g) ResNet-8, SVHN (h) ResNet-20, SVHN (i) ResNet-110, SVHN

Figure 3.2: **Illustration of matrix J in un-normalized form. Each cell depicts the number of sample often confused for the corresponding class index. Each heatmap also has a color indicator depicting approximation on total number of samples mistaken by the router on validation set. For example, approximately 200 sample from class $\{3, 5\}$ by Router ResNet-8 is confused with each other.**

Table 3.1: **Classes of CIFAR-10 dataset.**

Class Index	Class Name
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

Table 3.2: **Few instances of confusable pair of class index for CIFAR-10 dataset.**

top-n	subset ID.	class pairs
2	sub_1	$\{3, 5\}$
	sub_2	$\{2, 3\}$
	sub_3	$\{2, 4\}$
	sub_4	$\{1, 9\}$
	sub_5	$\{0, 8\}$
3	sub_1	$\{3, 4, 5\}$
	sub_2	$\{2, 3, 5\}$
	sub_3	$\{3, 5, 7\}$
	sub_4	$\{1, 8, 9\}$
	sub_5	$\{4, 5, 7\}$

Table 3.3: **Performance of experts on the corresponding confusable subset classes for CIFAR-10 dataset. The backbone network in this Table is ResNet-8. Each row depicts the performance of expert (trained on that corresponding subsets) on subset classes, all classes, and performance of router on the corresponding subset classes**

subsets	expert on sub. (%)	expert on all (%)	router. on sub. (%)
sub_1	85.70	80.90	79.15
sub_2	91.6	78.17	79.10
sub_3	93.5	79.61	86.55
sub_4	95.50	79.96	93.70
sub_5	95.55	81.96	91.25

est ResNet with only 8 layers), ResNet-20 and ResNet-110 as they are very lightweight in terms of parameters and faster to train. We train ResNet-8 and ResNet-110 from scratch for the router network using the exact same hyper-parameters specified in the original MS-Net for ResNet-20 (we will discuss more about hyper-parameters in the later section of this chapter). We do not require training ResNet-20 as we use the same router model from previous study with MS-Net.

For **expert networks** training, we perform sequential training like the original MS-Net. We train each expert networks by optimizing the stochastic loss function that we have proposed for MS-Net in previous chapter. The number of expert is exactly equal to the number of subsets we construct. Thus, we have total K experts denoted as $\mathcal{E} = \{e_1, e_2, .., e_K\}$. We depict the simplified version of loss function as follows:

$$Loss_{e_i} = \mathcal{X}\ CE_{sub_i} + (1 - \mathcal{X})\ KD_{all} \tag{3.2}$$

In the Eqn. 3.2, $\mathcal{X}$ is a Bernoulli random variable which takes a value of 1 with probability $Prob(\mathcal{X} = 1)$. In this study we set $Prob(\mathcal{X} = 1) = 0.5$, which implies that experts are trained with subset ground-truth labels for 50% of time and in rest of time trained on the softened labels provided by the teacher network. As our previous study for MS-Net, here the teacher network is the router network. An important question that can rise is, *how many experts should we have in the framework?* For original MS-Net the answer is straight-forward, as $Lemma\ 1$ suggest the total number of expert network is always equal to the number of classes available in the dataset. Thus, to keep comparison and our study consistent we keep the number of expert almost similar to the original MS-Net. However, it is important to note that, increasing the number of experts in this proposed optimization has two effects, i) the more experts we have, the more confusing classes we are likely to correct; ii) increased training time.

3.3.5 Test Phase

During the test phase, for the original MS-Net the number of expert evaluation depends on the variable ρ and top-n. The framework has exactly ρ experts for each class (the parameter ρ is tun-able based on computational availability). Thus, during inference if we consider the top-n of the router, we will require ρ * n experts for one sample every time. This can get exhaustive and redundant for samples that are relatively easier for router. O-MS-Net only evaluates one expert neural network per sample. On top of that, O-MS-Net does not leverage experts for sample that can be easily classified by the router. We have already shown in section 3.3.3 how we can leverage the *join-probability of co-occurrence of confusing classes* to know for which samples experts should be deployed.

We provide a simple test phase procedure for O-MS-Net as follows. We assume the input sample is d.

- **Step #1** Router $R(.)$ takes the input sample d and provide with it top-2 (or top-3 depending on dataset and implementation) most likely predictions p_1 and p_2.
- **Step # 2** Check if the top-2 predictions p_1 and p_2 occur exactly in same order in our set of subsets S.
- **Step # 3** If the predictions do not occur in subsets S we assume router is confident and return its top-1 prediction.
- **Step # 4** Else if the prediction occurs in similar order in the S we load the relevant expert for that pair of confusing classes.
- **Step # 5** We return the $arg\text{-}max$ of the output of the expert say $e(.)$ weighted by the routers confidence as $e(d) * R(d)$.

Algorithm 3: Testing phase of Optimized-MS-Net (evaluating up to top-2 of routers prediction)

Input: Dataset
Output: accuracy
Router network: $\mathcal{R}(.)$
Routers top-2 **class prediction:** $pred_1, pred_2$
set of expert networks: $\mathcal{E} = \{e_i(.)|i = 1, .., K\}$
subsets by the ICC method $\mathcal{S} = \{sub_i(.)|i = 1, .., K\}$
Boolean: routerIsConfident = True
Counter: correct = 0
for *d, t in* $enumerate(\mathcal{D}, \mathcal{T})$ **do**
    $pred_1, pred_2 = R(d)$
    for *i, sub in* $enumerate(\mathcal{S})$ **do**
        if *(*$pred_1$ & $pred_2$ *)*$\in$ *in sub* **then**
            set routerIsConfident = False
        end
        if *routerIsConfident == True* **then**
            if $pred_1$ == *t* **then**
                correct = correct + 1
            end
        else
        end
        # We will go to expert if router prediction does not suffice
        for *i, sub in* $enumerate(\mathcal{S})$ **do**
            if *sub* $\cap \{pred_1, pred_2\} \neq \emptyset$ **then**
                $pred_e$ = $e_i(d)$
            end
        end
        if $pred_e$ == *t* **then**
            correct = correct + 1
        end
    end
end

Table 3.4: Top**-1, 2 and 3 performance of Routers for CIFAR-10, CIFAR-100, FMNIST, and SVHN**

Router	Dataset	top-1 (%)	top-2 (%)	top-3 (%)
ResNet-8	C-10	88.52	96.01	98.28
	C-100	60.5	73.74	80.28
	F-MNIST	94.23	98.58	99.60
	SVHN	93.98	97.75	98.64
ResNet-20	C-10	92.68	97.20	98.73
	C-100	69.58	79.13	84.83
	F-MNIST	95.22	98.92	99.76
	SVHN	96.45	98.54	99.17
ResNet-110	C-10	93.15	97.58	99.00
	C-100	71.44	82.13	87.21
	F-MNIST	95.50	98.83	99.70
	SVHN	96.70	98.57	99.50

3.4 Test Phase Algorithm

We provide more formal definition of test-phase in the Algorithm 3. Line 1-6 we initialize the data-loader, construct subset by ICC, initialize expert networks and so on. In line 8 we take the top-2 probable prediction by the router i.e. $\{pred_1, pred_2\}$. Next, we check if these top-2 predictions are in our confusing class subsets. If they exist in the confusing class subsets we simply assume router $R(.)$ is not confident. In the line 10 we check if $R(.)$ is confident or not, if confident, we take its prediction as our final prediction, thus we do not require to go to expert any longer. However, if Router is not confident we further go to line 19. The loop in line $20-21$ checks if our pre-constructed subset have non-empty intersection with the top-2 predictions of router. If we have non-empty intersection we perform inference through that corresponding expert network.

3.5 Experiments

3.5.1 Datasets

Datasets that we leverage to evaluate O-MS-Net are CIFAR-10, CIFAR-100, Fashion MNIST (FMNIST) and SVHN (The Street View House Numbers) . The CIFAR-10 dataset has 60,000 32X32 color images. 50,000 of them are for training and rest for the testing. CIFAR-100 also consists of 60,000 color images with 100 classes and 600 images per class. 50,000 of them are for training and rest 10,000 for the testing. The partition for train and test have been officially set by the original dataset provider [111]. FMNIST dataset is also moderately large dataset with fashion accessories consisting of 60,000 training images and 10,000 test images. The images of FMNIST are 28X28 gray-scale image. SVHN is a color image dataset with 73,257 training images and 26,032 test images provided officially. Each image is 32X32 in shape.

SVHN is relatively new dataset that we leverage for the first time in this dissertation to evaluate O-MS-Net along side aforementioned datasets. CIFAR-10 and CIFAR-100 are visually

Table 3.5: **Performance of Optimized MS-Net on CIFAR-10 based on the top-2 ICC analysis. The router and expert networks architecture are identical. The δ column depicts the improvement overall network achieves with corresponding experts relative to the router network, i.e. the number of samples correctly re-classified by the experts.**

Backbone	only router (%)	# of experts	with expert (%)	δ (%)
ResNet-8	88.52	3	89.50	+0.98
		5	90.00	+1.48
		10	92.20	+3.68
ResNet-20	92.68	3	93.93	+1.25
		5	95.30	+2.62
		10	95.90	+3.22
ResNet-110	93.15	3	94.83	+1.68
		5	95.54	+2.39
		10	96.90	+3.75

Table 3.6: **Performance of Optimized MS-Net on CIFAR-10 based on the top-3 ICC analysis.**

Backbone	# of experts	with expert (%)	δ (%)
ResNet-8	3	90.88	+2.36
	5	91.36	+2.84
	10	93.61	+5.09
ResNet-20	3	94.30	+1.62
	5	95.80	+3.12
	10	96.46	+3.78
ResNet-110	3	95.14	+1.99
	5	95.77	+2.62
	10	96.95	+3.80

Table 3.7: **Performance of Optimized MS-Net on CIFAR-100 based on the top-2 ICC analysis..**

Backbone	only router (%)	# of experts	with expert (%)	δ (%)
ResNet-8	60.50	10	62.93	+2.43
		30	65.55	+5.05
		50	66.04	+5.54
ResNet-20	69.58	10	71.15	+1.57
		30	72.50	+2.92
		50	72.81	+3.23
ResNet-110	71.44	10	73.50	+2.06
		30	75.94	+4.50
		50	76.50	+5.06

Table 3.8: **Performance of Optimized MS-Net on CIFAR-100 based on the top-3 ICC analysis.**

Backbone	# of experts	with expert (%)	δ (%)
ResNet-8	10	63.01	+2.51
	30	68.46	+7.96
	50	68.95	+8.45
ResNet-20	10	72.70	+3.12
	30	74.71	+5.13
	50	74.90	+5.32
ResNet-110	10	74.42	+2.98
	30	76.95	+5.51
	50	77.30	+5.86

Table 3.9: **Performance of Optimized MS-Net on FMNIST based on the top-2 ICC analysis..**

Backbone	only router (%)	# of experts	with expert (%)	δ(%)
ResNet-8	94.23	3	95.10	+0.87
		5	95.50	+1.27
		10	95.80	+1.57
ResNet-20	95.22	3	95.70	+0.48
		5	96.20	+0.98
		10	96.88	+1.66
ResNet-110	95.50	3	96.10	+0.60
		5	96.45	+0.95
		10	96.75	+1.25

Table 3.10: **Performance of Optimized MS-Net on FMNIST based on the top-3 ICC analysis.**

Backbone	# of experts	with expert (%)	δ (%)
ResNet-8	3	95.52	+1.29
	5	95.72	+1.49
	10	96.31	+2.08
ResNet-20	3	96.21	+0.99
	5	96.44	+1.22
	10	96.98	+1.76
ResNet-110	3	96.24	+0.74
	5	96.51	+1.01
	10	97.00	+1.78

Table 3.11: **Performance of Optimized MS-Net on SVHN.**

Backbone	only router (%)	# of experts	with expert (%)	δ (%)
ResNet-8	94.00	3	95.20	+1.20
		5	95.70	+1.70
		10	96.10	+2.10
ResNet-20	96.45	3	97.30	+0.85
		5	97.70	+1.25
		10	97.80	+1.35
ResNet-110	96.70	3	97.30	+0.6
		5	97.70	+1.00
		10	98.01	+1.31

very interpretable and reliable to understand the confusing set of classes and performance of experts on them. However, additionally we choose SVHN for several reasons. First, SVHN is a challenging dataset due to existence of several noise in the images. It is basically a digit dataset similar to MNIST. However, unlike MNIST, SVHN is real world digit images extracted from the house numbers in Google Street View images [131] which consists of several noise (such as blurry and incomplete images). Second, images of this dataset have several distraction. A single image can consist of several overlapping digits or multiple digits in a single frame. This requires classifiers to be pin-point accurate. The noise and distraction in this dataset make prediction for neural network challenging. Evaluating performance of experts for such sample can be ideal and a proper measurement of performance.

The aforementioned dataset mostly comes with official training and test set. Thus, to save the checkpoints we segregate the original training dataset into two parts with 10% for validation and checkpoint saving and rest for training experts and routers.

3.5.2 Implementation and Hyper-parameter Settings

We implement the networks of O-MS-Net in the PyTorch framework [112], and perform all the experiments on single NVIDIA GeForce RTX 2080 GPU. The setting of hyper-parameters such as, batch size, number of epochs and so on during training slightly vary across different datasets. However they have been kept consistent and similar to the original MS-Net. We optimize all the networks (router and experts) through Stochastic Gradient Descent(SGD) with momentum. We set the initial learning rate for all routers to 0.1. The rest of hyper-parameter for routers for different dataset is depicted in the Table 3.12.

Training Expert Network is slightly different than router. We do not encourage training the experts from scratch. One of the major reason for not doing so, is to avoid over-fitting the experts on the subset data. Each of subset is composed on two confusable set of classes. Thus, subset is composed of very few dataset and over-fitting on these dataset will be easier for these experts. So we adopt the solution for avoiding over-fitting from original literature [45]. The solution is to simply initialize expert networks with the weights of router network. In this way, we can preserve general knowledge about all the classes for the experts. The next step is exactly similar to the naive-fine-tuning task. We fine-tune each layers of expert networks with different learning rate. Such as, the initial learning rate for the first two CNN ResNet blocks (applicable to any ResNet networks) of experts is set to 0.0001. The rest of CNN blocks and fully connected layers have initial learning rate of 0.1. The goal was to preserve features learned by the earlier

layer (since they are learned from all class data). However, it is also possible to freeze earlier layers to preserve global features provided by Router. We have not tried it, but we anticipate similar performance. The rest of hyper-parameters are depicted in the Table 3.12. Training and testing of all the routers and experts are performed in single run. Performing multiple run of experiments will definitely provide us with more reliable measure of performance and better weights with substantially increased cost in training time.

Table 3.12: **Training hyper-parameters for router and experts**

Network	Dataset	Batch size	Epochs	Steps
Router	C-10	32	300	50
	C-100	128	300	50
	F-MNIST	64	200	60
	SVHN	64	300	80
Experts	C-10	32	30	8
	C-100	16	25	8
	F-MNIST	64	30	10
	SVHN	16	25	8

3.6 Results Discussion

Performance of router is crucial for MS-Net and also O-MS-Net. We train all routers from scratch with similar hyper-parameters (depicted in Table 3.12) with single run. Table 3.4 depicts the top-1 and top-2 performance of the router. Intuitions suggest, the top-1 is lower bound performance that we can at least expect from O-MS-Net (or MS-Net too) and the top-2 is the upper bound performance that we anticipate to achieve. An important and key observation about router is that, the top-1 accuracy is fluctuating (actually improving) slightly as architecture get more complex (refer to Table 3.4). However, the top-2 predictions are relatively consistent irrespective of the architecture and its complexity. On top of that, the heat-map for Joint-probability of occurrence of top-2 confusing class-pairs are consistent irrespective of training setup, architecture or any hyper-parameter. This suggests the top-1 performance of router does not need to be strictly consistent. The performance will suffice as long as top-2 is consistent, which is indeed consistent for all four datasets that we explored so far.

3.6.1 Performance on CIFAR10 and CIFAR100

We study O-MS-Net for **CIFAR-10** with three different variants of ResNet network as we depict in the Table 3.10. The number of experts we test the whole network with are 3, 5 an 10 for CIFAR-10 dataset. As we increase the number of experts we have consistent improvement in performance, which is expected. Because, as we increase number of experts we are considering more confusable set of classes to correct. The performance of the experts are significantly better than their router network on these confusable set of classes which we illustrate in Table 3.3. Moreover, Table 3.3 also demonstrates that these experts are not over-fit on the subset data, rather they have generalized quite well on all classes. This credit goes to the proposed stochastic objective function in the original MS-Net. Adding more experts requires more training time, while pushing O-MS-Net to close the gap with the upper bound target, that is the top-2 accuracy

of router. The best score we obtain so far is $96.90\%$with the ResNet-110 backbone with 10 experts. The total number of parameters is therefore 18.7M during training time. However, during inference only 1 expert is leveraged which make the total parameter count 3.4M per sample.

CIFAR-100 dataset is challenging for both MS-Net and O-MS-Net. One of the primary reason is large number of classes, yet with very few samples per class. The router itself suffers to achieve good top-2 performance (refer to Table 3.4). However, O-MS-Net still exhibits consistent and positive δ score as we increase experts. As number of classes is large we report scores starting from 10 experts and eventually increasing number of experts to 30. Table 3.7 represents the performance of O-MS-Net on CIFAR-100.

3.6.2 Performance on FMNIST and SVHN

Performance of O-MS-Net on FMNIST and SVHN is quite consistent like the CIFAR-10. As we leverage more expert networks, the performance gap gets closer to the upper bound (i.e. top-2)for respective datasets (refer to Table 3.4). Consistent performance on all these dataset convinces us that O-MS-Net data partition policy and performance of experts are effective as we increase more experts (thus covering more confusable set of classes). However, O-MS-Net is more favourable and tractable in-terms of training time where number of classes is relatively smaller. With increased number classes it might be ideal to increase the value of n for top-n evaluation or in order words to consider more confusable set of classes. This procedure will of-course require more training time as experts will increase.

3.6.3 Visual Demonstration and Discussion on Confusing Samples

The section demonstrates and discusses on visually difficult samples corrected by the expert networks. It is indeed desirable to understand what set of samples are mis-classified by the router and why?, and also samples that are corrected by the experts. This will assist in finding out the classification performance (subjective judgment) of experts for samples for the fine-grained confusable set of classes. We depict few cherry-picked samples corrected by experts for CIFAR-10 and SVHN in Figure 3.3. The depicted samples in the aforementioned figures are also the anticipated top-1 error by the router. Three human raters were involved in picking the samples depicted in Figure 3.3. The samples are color coded, where green color indicates that human raters agree with the expert networks prediction. Red indicates that human raters are not sure about the prediction (i.e. confusing for human raters). The first word for every sample is the expert prediction and the second word is routers prediction.

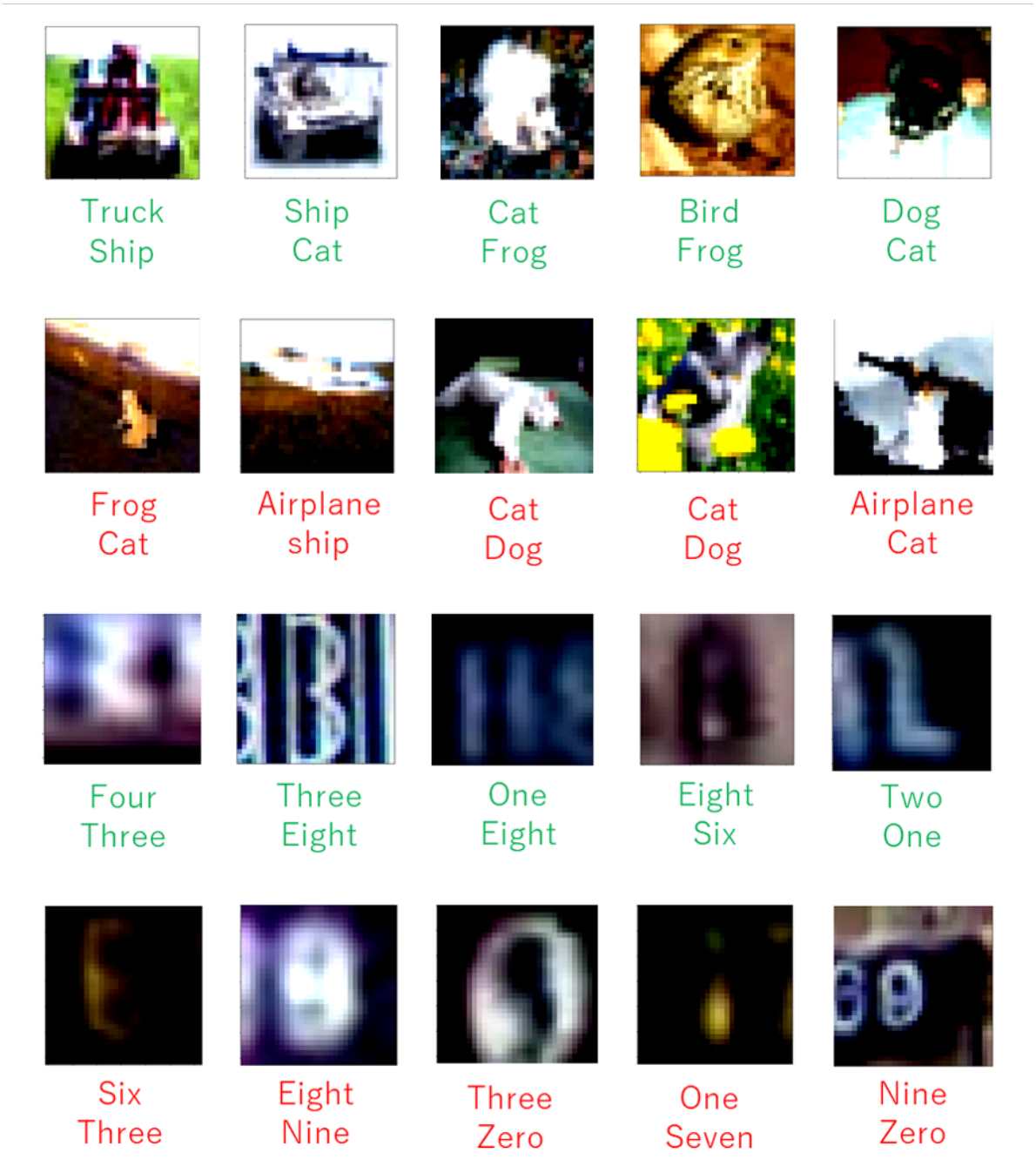


Figure 3.3: **Instances of samples from CIFAR-10 and SVHN that are corrected by the expert networks. First word for every sample is the expert prediction and second word is the routers prediction**.

Table 3.13: **Performance comparison for MS-Net and O-MS-Net for CIFAR-10, CIFAR-100, and FMNIST**

Network	C-10 (%)	C-100 (%)	FMNIST (%)	Train. Params. (M)	Test. Params. (M)	# of exp./sample
MS-Net-20 ($\rho = 2, n = 2$)	93.54	70.68	95.80	2.95 (27.16)	1.35	4
MS-Net-20 ($\rho = 3, n = 2$)	94.15	70.85	95.80	2.95 (27.16)	1.87	6
MS-Net-20 ($\rho = 4, n = 2$)	95.38	71.61	96.02	2.95 (27.16)	2.41	8
MS-Net-20 ($\rho = 2, n = 3$)	93.34	71.00	95.96	2.95 (27.16)	1.87	6
MS-Net-20 ($\rho = 3, n = 3$)	94.06	71.09	96.77	2.95 (27.16)	2.7	9
MS-Net-20 ($\rho = 4, n = 3$)	95.30	71.28	96.77	2.95 (27.16)	3.50	12
O-MS-Net-20 ($n = 2, K = 3, 10$)	93.93	71.15	95.70	1.07 (2.95)	0.53	1
O-MS-Net-20 ($n = 2, K = 5, 30$)	95.30	72.50	96.20	1.60 (8.07)	0.53	1
O-MS-Net-20 ($n = 2, K = 10, 50$)	95.90	72.81	96.88	2.95 (13.71)	0.53	1
O-MS-Net-110 ($n = 2, K = 3, 10$)	94.83	73.50	96.10	6.8 (18.70)	2.41	1
O-MS-Net-110 ($n = 2, K = 5, 30$)	95.54	75.94	96.45	10.20 (52.70)	2.41	1
O-MS-Net-110 ($n = 2, K = 10, 50$)	96.90	76.50	96.75	18.70 (85.00)	2.41	1
O-MS-Net-20 ($n = 3, K = 3, 10$)	94.30	72.70	96.21	1.07 (2.95)	0.53	1
O-MS-Net-20 ($n = 3, K = 5, 30$)	95.80	74.71	96.44	1.60 (8.07)	0.53	1
O-MS-Net-20 ($n = 3, K = 10, 50$)	96.46	74.90	96.98	2.95 (13.71)	0.53	1
O-MS-Net-110 ($n = 3, K = 3, 10$)	95.14	74.42	96.24	6.8 (18.70)	2.41	1
O-MS-Net-110 ($n = 3, K = 5, 30$)	95.77	76.95	96.51	10.20 (52.70)	2.41	1
O-MS-Net-110 ($n = 3, K = 10, 50$)	96.95	77.30	97.00	18.70 (85.00)	2.41	1

3.6.4 Comparison with MS-Net

Table 3.13 depict the performance comparison for MS-Net and O-MS-Net. MS-Net in chapter 2 performed detailed empirical and ablation study with ResNet-20 backbone. To make the comparison comparable and fair we depict all the scores of MS-Net with ResNet-20 with redundancy value $\rho = \{2, 3, 4\}$ and keeping top-n fixed to 2. For the O-MS-Net of this chapter we present scores with ResNet-20 backbone consisting of varying number of expert networks, i.e. varying the value of K. The value of K i.e. the number of experts for CIFAR-10, FMNIST are $k = \{3, 5, 10\}$. CIFAR-100 has more number of classes thus we present results of O-MS-Net with value of $K = \{10, 30, 50\}$. Training parameters thus varies greatly for CIFAR-100 dataset. Since CIFAR-100 training parameters can vary we depict the value within the bracket in the column *Train. Params.*. However, during inference irrespective of number of experts in our disposal we leverage one experts (or no experts at best case scenario when router is confident for that certain datum). Comparing MS-Net and O-MS-Net head to head is not straight-forward due to several contrast. To keep our comparison simple let us focus on number of parameters leveraged (or number of experts) per sample. In the following points we summarize our observation from Table 3.13.

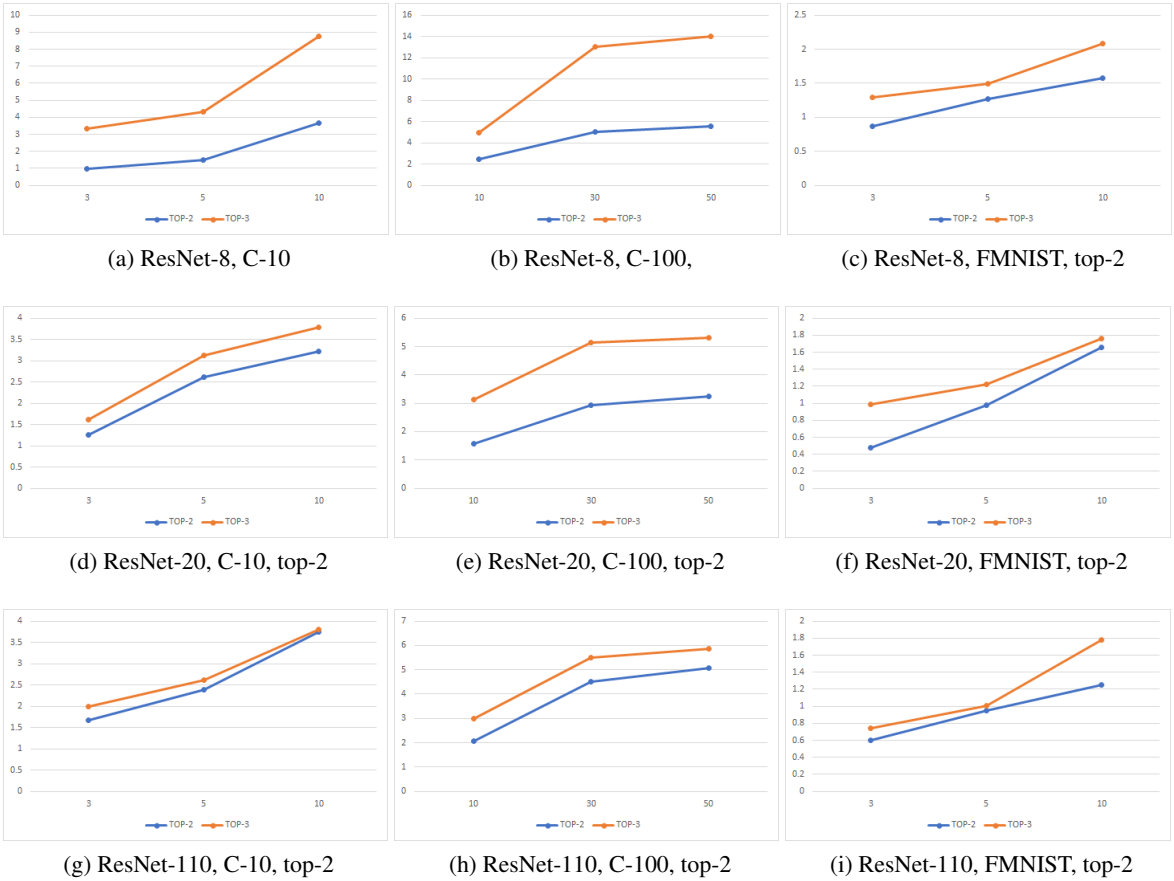
- O-MS-Net is substantially better than original MS-Net when we consider cost of inference per sample. Let us consider an example from the Table 3.13. We take the lightest MS-Net (first row) and lightest O-MS-Net (sixth row) and compare head to head. Performance of both model is very close for CIFAR-10 and FMNIST. The contrast in parameter usage and experts per sample is significant.
- MS-Net provides with systematic control and freedom to adjust redundancy. This can be helpful for situation where we need explicit redundancy for fault tolerance. O-MS-Net does not guarantee redundancy in our current implementation.
- Training cost of MS-Net is very high as it requires to train C experts within whole framework. O-MS-Net suffices with K experts, where K can be must smaller than C yet perform substantially better than MS-Net.
- With O-MS-Net we can afford to leverage much more complex DNNs as the backbone yet keep parameter count tractable and lower than MS-Net during both training and testing (Refer to row 3, 8 and 9 for comparison).

3.7 Training Setting Recommendations for O-MS-Net

There are several hyper-parameters that can substantially effect the performance of O-MS-Net. Proposed O-MS-Net is quite sensitive to hyper-parameter settings as it consists of training several decoupled expert networks and router network. Irrespective of hyper-parameter settings we can always anticipate positive δ relative to the baseline router network. However, for better and consistent performance in this section we provide several training guideline for O-MS-Net.

3.7.1 ICC with top-2 and top-3

An obvious intuition is that as we evaluate more top-n of the router we can correct more samples through experts. The empirical result reflects this intuition. We visualize this effect of top-2 versus top-3 performance (with varying backbone networks) for CIFAR-10, CIFAR-100 and FMNIST datasets in Figure 3.4. In original MS-Net as we increase value of n (of top-n) we required evaluating more expert neural networks. For O-MS-Net the number of expert neural network to be evaluated during the inference phase remains the same despite the increment in the value of n.

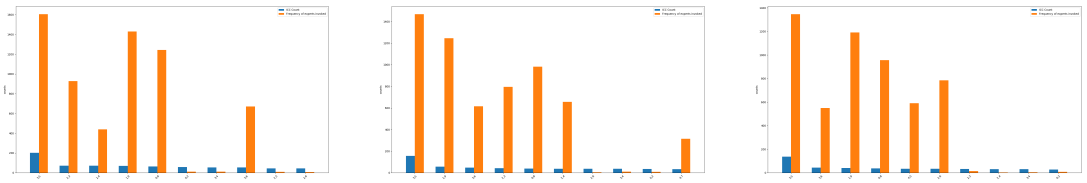


(a) ResNet-8, C-10 (b) ResNet-8, C-100, (c) ResNet-8, FMNIST, top-2

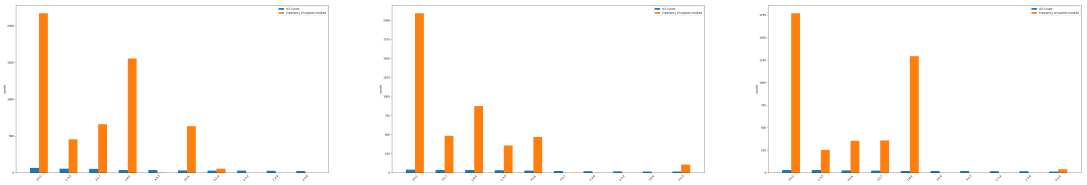
(d) ResNet-20, C-10, top-2 (e) ResNet-20, C-100, top-2 (f) ResNet-20, FMNIST, top-2

(g) ResNet-110, C-10, top-2 (h) ResNet-110, C-100, top-2 (i) ResNet-110, FMNIST, top-2

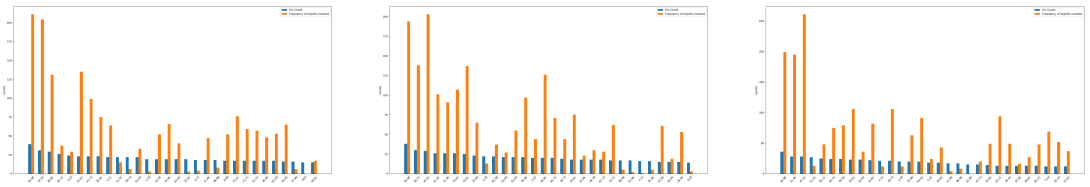
Figure 3.4: **Delta score of O-MS-Net when experts are constructed based on top-2 and top-3 based ICC. When we train experts based on ICC of routers top-2 the experts gain specialization on two classes and generalizes on rest of classes (refer to the loss function equation 3.2). When we increase to top-3 of router each experts specializes on three set of classes (based on ICC of top-3 triplet of classes). Since top-3 based ICC cover more class we get slightly better improvement in performance.**



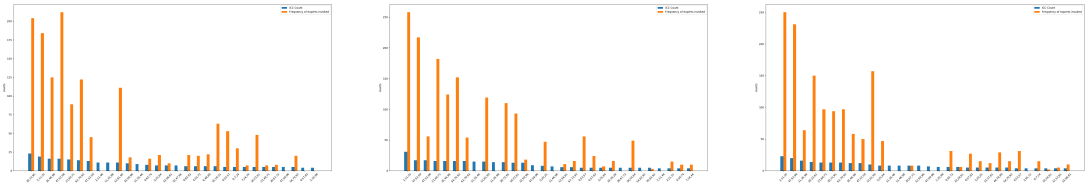
(a) ResNet-8, C-10, top-2 based ICC (b) ResNet-20, C-10, top-2 based ICC (c) ResNet-110, C-10, top-2 based ICC



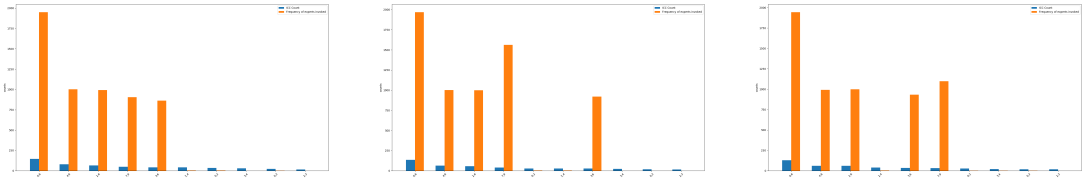
(d) ResNet-8, C-10, top-3 based ICC (e) ResNet-20, C-10, top-3 based ICC (f) ResNet-110, C-10, top-3 based ICC



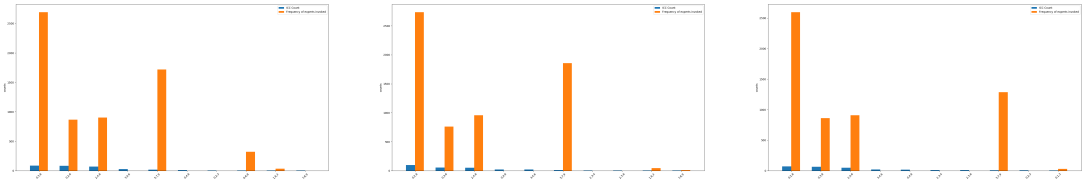
(g) ResNet-8, C-100, top-2 based ICC (h) ResNet-20, C-100, top-2 based ICC(i) ResNet-110, C-100, top-2 based ICC



(j) ResNet-8, C-100, top-3 based ICC (k) ResNet-20, C-100, top-3 based ICC(l) ResNet-110, C-100, top-3 based ICC



(m) ResNet-8, FMNIST, top-2 based ICC (n) ResNet-20, FMNIST, top-2 based ICC (o) ResNet-110, FMNIST, top-2 based ICC



(p) ResNet-8, FMNIST, top-3 based ICC (q) ResNet-20, FMNIST, top-3 based ICC (r) ResNet-110, FMNIST, top-3 based ICC

Figure 3.5: **Statistics on number of times each experts invoked during the inference phase on test data of CIFAR-10, 100 and FMNIST. The bar-charts are sorted based on the most frequently leveraged experts. It is clear from the figure that the dominant experts (frequently picked by the router) are for the difficult classes. The figures depict frequency of experts for both top-2 and top-3 cases.**

Increasing the value of n effects the size of subsets (of class indexes) on which the experts are trained on. Such as, with value of $n = 2$ and $n = 3$ the cardinality of subset of class indexes would be 2 and 3 respectively. This implies that as we increase value of n the experts requires focusing on more classes. As experts focuses on more classes the framework basically do not require too many experts to cover the confusable set of classes (since any expert itself covers more classes). We can depict this effect from the Figure 3.5. We observe that the number of expert networks leveraged by the top-2 version O-MS-Net is relatively higher than the top-3 version.

The effect of value n in terms of performance depends on the difficulty of dataset. Let us refer to Figure 3.4 where we can observe that the delta gap between top-2 and top-3 performance (for ResNet-8, 20, 110) for CIFAR-10 and FMNIST dataset is relatively smaller than CIFAR-100 dataset. This indicates that increasing top-n for difficult dataset like CIFAR-100 provides considerable performance gain. Thus we can recommend to consider evaluating more top-n for dataset with large number of classes, as smaller value of n for weak router will not suffice.

A question still remain, *how should we decide the value of top-n when we do not have prior knowledge on dataset?* Should we keep on training O-MS-Net with increasing value of n till we reach the bottleneck performance ? Then again, the obtained value of n in such case would be dataset specific, and needless to say very expensive. A simple approach to decide would be, to consider the accuracy gap between O-MS-Net-top-n' and router top-$(n' - 1)$. If the top-$(n' - 1)$ accuracy of router is considerably comparable (or better than) to the O-MS-Net top-n' (also for original MS-Net) performance we can decide to limit the value to n'. This is because, since router's top-$(n' - 1)$ performance is already better or equal to the O-MS-Net top-n' performance increasing value of n' will not provide any significant performance.

To understand this let us consider a simple example with the help of Table 3.4, 3.10 and 3.6. From Table 3.4 we find that the top-2 and top-3 performance of router (with Resnet-20 backbone) for CIFAR-10 dastaset are 97.20 % and 98.73 % respectively. The top-2 and top-3 version of O-MS-Net with same router and ten experts obtain accuracy of 95.90 % and 96.46 %. It is clear that O-MS-Net top-3 version has a slight performance gap with the router top-2 accuracy. This is actually a clear indication to limit value of n' to 3. In addition to that it should be noted that increasing n increases the cardinality of subset of classes index (obtained through ICC). As cardinality of respective subset increases expert will require focusing on more classes.

3.7.2 Value of K

Value of K is a very important hyper-parameter for O-MS-Net. Unlike the original MS-Net there is no theoretical upper or lower bound for the value of K. Thus our prior knowledge or heuristics were leveraged when deciding the value of K. Such as, for dataset with small number of classes (CIFAR-10, FMNIST, SVHN) we leverage at-most 10 experts. On the other hand for dataset like CIFAR-100 we leverage at-most 50 experts. It is intuitional that as we leverage more experts we cover more confusable set of classes, that is increment of delta in positive direction. However, for dataset with smaller number of classes the effect of positive delta starts to fade after we increase number of experts (i.e. value of k) beyond 5. This implies that for these particular dataset we might not require more than 5 experts in total.

For visual assurance let us refer to Figure 3.5. In Figure 3.5 the orange bar depicts the frequency of each experts leveraged for the test set. The blue bar is the ICC of the classes (classes on which corresponding experts were trained) based on validation set. We know based on the ICC on validation set we construct experts on corresponding classes. The Figure shows that ICC is fairly a good approximation of classes that requires experts. However, there are certain experts that are leveraged considerably more than we anticipated (refer to sudden spikes in the bar-chart of Figure 3.5). This is one of the key reason on why upper and lower bound of K is difficult to fix.

Table 3.14: **Performance of Optimized MS-Net on CIFAR-100 with small K.**

Backbone	router (%)	# of experts	with expert (%)	δ %
ResNet-8	60.50	1	61.05	+0.55
		3	62.00	+1.50
		5	62.53	+2.02
ResNet-20	69.58	1	69.80	+0.22
		3	70.43	+0.85
		5	71.16	+1.58
ResNet-110	71.44	1	72.00	+0.56
		3	72.80	+1.36
		5	73.22	+1.78

3.7.3 Weight Initialization for Expert Networks

Weight initialization is an important design choice for O-MS-Net and MS-Net. Weight initialization is used to define the initial values for the parameters in neural network models prior to training the models on a dataset. Current DNNs are so heavily parameterized and stochastic in nature that, the initial weight choice can make 1-1.5% performance difference in every single run. Thus, making the obtained results very difficult and uncertain to repeat. Moreover a proper weight initialization can assists in reaching optimum results in less number of epochs with certain degree of guarantee. Several heuristics are incorporated while initializing weights, such as, sampling Gaussian or uniformly distributed values for weights. Best practice is actually initializing neural networks with pre-trained weights if exists (for vision task generally ImageNet trained weights are used).

In this research we recommend initializing expert neural networks with routers weights (assuming that both router and expert networks have identical topology). This approach has been recommended earlier in literature [45]. Also it is not recommended to perform random initialization of expert networks when subset cardinality is smaller. Expert networks are trained on very small proportion of original dataset (few subset of classes). The amount of dataset are relatively smaller which are very easy to over-fit on. Moreover with such small amount of data the network does not learn good feature representation that can provide with proper generalization. Initializing with routers weight greatly assists in prevailing knowledge (originally known as the dark knowledge, a term coined by Prof. Hinton) on the rest of the class, and also provides with better feature preservation in the earlier layers. It is worth mentioning that, randomization of weight might be welcoming in situation where we have sufficient subset data to train the experts.

In order to demonstrate the above discussion we train O-MS-Net with random weight initialization for expert networks. We basically leverage Xavier Weight Initialization of Pytorch implementation [132]. We depict the performance of O-MS-Net with random initialization in Table 3.15, 3.16 and 3.17 for CIFAR-10, CIFAR-100 and FMNIST datasets respectively. The results in these tables are a clear indicator that randomly initialized expert networks when trained on subset data fails to maintain consistency for all the datasets. Below we highlight few important key points:

- **Randomly initialized expert networks are well-suited in situation when there are sufficient amount of samples per class.** Such as for CIFAR 10 (refer to Table 3.15) and FMNIST (refer to Table 3.17). These dataset have sufficient samples per class, as a result

Table 3.15: **Performance of Optimized MS-Net on CIFAR-10 with expert networks trained through random initialization.**

Backbone	only router (%)	# of experts	with expert (%)	δ (%)
ResNet-8	88.52	3	89.10	+0.52
		5	90.10	+1.58
		10	92.05	+3.53
ResNet-20	92.68	3	94.00	+1.32
		5	95.20	+2.52
		10	95.70	+3.02
ResNet-110	93.15	3	94.94	+1.79
		5	95.30	+2.15
		10	96.50	+3.35

Table 3.16: **Performance of Optimized MS-Net on CIFAR-100 with expert networks trained through random initialization.**

Backbone	only router (%)	# of experts	with expert (%)	δ (%)
ResNet-8	60.50	10	59.38	-1.12
		30	61.59	+1.09
		50	64.74	+4.24
ResNet-20	69.58	10	67.57	-2.01
		30	68.91	-0.67
		50	71.22	+1.64
ResNet-110	71.44	10	67.32	-4.12
		30	68.34	-3.1
		50	69.87	-1.57

Table 3.17: **Performance of Optimized MS-Net on FMNIST with expert networks trained through random initialization.**

Backbone	only router (%)	# of experts	with expert (%)	δ(%)
ResNet-8	94.23	3	95.25	+1.02
		5	95.57	+1.34
		10	95.76	+1.53
ResNet-20	95.22	3	95.68	+0.46
		5	95.90	+0.68
		10	96.35	+1.13
ResNet-110	95.50	3	96.15	+0.65
		5	96.30	+0.80
		10	96.54	+1.04

experts optimized with stochastic loss function achieve performance comparable to the router initialized networks. For both of these datasets the δ value is positive consistently.

- CIFAR-100 is already a challenging dataset due to large number of visually similar classes (many classes and sub-classes among them, such as, different types of vehicles, trees, Reptiles and so on). Moreover, number of samples per class is substantially low which does not help experts to learn rich features. This difficulty is easily depictable from Table 3.16, where δ value is negative for most of the combination of experts. **Thus, situation where number of sampler per class is sparse, it is recommended to i) initialize with router's weight (or any pre-trained weights such as, ImageNet); and ii) put more weight in KD terms in loss function when training the experts (recommended weight** $\alpha = 0.8$**).**

3.8 Summary

The chapter proposed optimization for MS-Net (Modular Selective Network). Proposed optimization significantly reduces the number of expert required to match original MS-Net performance. The key idea was to leverage the softmax property of neural networks, whose top-n probability distribution exhibits certain degree of visual similarity and proximity. The importance of this information have been first demonstrated first in literature [45]. We leverage and augment such information to calculate the join-probability of co-occurrence of certain visually similar or confusing class pairs from the routers performance on validation set. More importantly we demonstrate that this join-probability is very stable irrespective of network architecture, thus ensuring partition stability. From the calculated join-probability matrix we construct subset of datatset for the expert networks. Such data partitioning technique for the MS-Net is much more systematic, efficient, straight-forward and easy to implement as oppose to clustering based data partitioning. Performance of O-MS-Net through this data partitioning technique required leveraging at most one expert network during the inference. In the best case scenario O-MS-Net does not need any expert as the prediction of router will suffice. Empirical study on four well-known dataset demonstrated its superiority in-terms of parameter counts and performance over original MS-Net.

Chapter 4

CMNN: Coupled Modular Neural Network

The chapter proposes a generalized version of MS-Net where instead of having several decoupled modules like MS-Net, modules are coupled through shared backbone network. Thus, all the modules shares same intermediate feature maps through the backbone network. Thus the network architecture is named Coupled Modular Neural Network or CMNN.

A CMNN is a network consisting of β closely coupled sub-networks, where β is termed as the branching factor in this paper. We call the whole network a super-graph and each sub-network a sub-graph. Each sub-graph is a stand-alone neural network and shares a common block with other sub-graphs. To effectively leverage the super-graph we propose a simple but easy-to-implement Round-Robin-based learning algorithm. Each training iteration contains two phases. In the first phase, we choose a sub-graph in a Round-Robin fashion and train it using knowledge of the super-graph (distillation). In the second phase, we fine-tune the super-graph based on the updated sub-graphs. This algorithm produces a different copy of the super-graph at each iteration which acts as an improved teacher network for the sub-graph; and a different copy of one of the sub-graphs which functions as a new building block for the super-graph. To validate and test CMNN and the proposed algorithm, we conduct experiments on CIFAR-10, CIFAR-100, and a private On-Road-Risk (ORR) dataset. Empirical results on all these three datasets indicate that we not only obtain a strong sub-graph network, the learning framework can also produce strong ensemble performance which substantiates the diversity introduced throughout the learning framework.

4.1 Introduction

Since the introduction of Convolutional Neural Networks (CNNs) [7, 8] development and progress in computer vision tasks such as, object recognition [9–11]; object detection and localization [12, 13]; semantic image segmentation [14–17]; image generation [18–20]; neural style transfer [21, 22] and so on have boosted dramatically. CNNs have gained high preference and success mainly due to its ability to automatically learn both discriminative and constructive features; and can approximate complex input-output mappings through the convolutional and pooling layers using back-propagation algorithm [6]. In the very beginning CNNs (such as [23]) were simple and consisted of limited hyper-parameters and factors such as number of layers, number of channels per layer, and Sigmoid or Hyperbolic Tangent non-linearity. However, gradually these networks have evolved and started to get more complex, deeper, and accurate with the introduction of the template such as fixed filter size with a large number of channels or feature maps [24], skip connection between non-consecutive layers [25], multi-path or branch design [26] and so on. Well-known and most practiced networks such as ResNet, DenseNet, GoogleNet, and so on leverage the aforementioned templates. These networks have served as a

very strong backbone networks for object detection [12,133], classification task [25,27], and so on.

However to achieve near-human level accuracy usually it is preferred to leverage the deeper and heavier version of these networks such as *ResNet-1202, DenseNet-BC (L=190, k=40)*. In practice, these networks are too heavy to be deployed on mobile devices with limited computational resources. Needless to say, the lighter and shallow version of these networks can be leveraged in mobile devices but with a compromise for accuracy. Hence there is a growing need to make neural networks smaller and faster without hampering or compromising performance. Recently there have been several approaches to make neural networks affordable in low computational devices, such as i) making them compact and smaller by model compression techniques [45,59,91]; ii) different regularization techniques to assist smaller networks to generalize well [134,135]; iii) automated Neural Architecture Search (NAS) through different search procedures [36–38,136]; iv) redundant parameter pruning [92]; and v) modular or block-like design of neural networks [84,100].

Among all these approaches recently model compression through Knowledge Distillation (KD) has gained increased popularity due to its implementation simplicity and availability of strong teacher networks. Automated Machine Learning (AutoML) techniques such as network pruning and NAS are also widely practiced now and have shown promising results. In fact most of the recent state-of-the-art neural network architectures such as the AmoebaNet-A (N=6, F=448) [32], EfficientNet-B7 [36], NASNet-A [37] obtained through NAS have outperformed hand-crafted neural networks in terms on accuracy and efficiency. However, these benefits come with a huge computational cost. Such as network obtained by reinforcement learning-based approach [38] required around *450 GPUS* for four days to perform a single experiment. Similar case is also applicable for the evolutionary based approach [34]. Moreover, the topology of these networks are evolved for task or dataset specific purpose. It is simply very expensive to perform such a large-scale network topology search and training thousands of these models on lab-level computers or for day-to-day machine learning practitioners.

In order to make the search space for neural topology smaller and affordable research [97,137,138] introduced the concept of weight sharing among multiple neural networks. This approach avoids training each of the neural networks from scratch. The key concept is to first construct a strong and complex directed acyclic hyper-network. Afterwards, the search algorithm searches for optimal sub-networks within the same large hyper-network, thus reducing the search space within the hyper-network. The search strategy or sampling of sub-networks within the hyper-network can vary across different literature. Such as, research [139] leveraged evolutionary algorithm to adapt the sub-networks of corresponding super-network, research [140] employs a controller network trained by policy gradient based on reinforcement learning to search for the most efficient sub-networks within the whole network.

In this paper, we propose a modular structural learning framework for the Deep Convolutional Neural Networks (DCNNs) termed as the Coupled Modular Neural Network (CMNN) framework. CMNN has a very close resemblance to the research [97,137,138] in terms of network architecture. Our proposed framework consists of three primary blocks known as the *shared common block, multiple hidden blocks, and the aggregation block*. These three primary blocks together make the super-graph network. The super-graph consists of multiple sub-graphs where each sub-graph is a stand-alone neural network. Each of the stand-alone sub-graph networks shares weights with other sub-graph networks through the shared common block. A single sub-graph consists of a shared common block, a single hidden block, and finally a single linear layer. We depict the big picture of the network in Figure 4.1.

Our network architecture has a very close resemblance to the literature of NAS such as [97,138] in terms of weight sharing among multiple sub-nets within one hyper-net. However we do not employ any search strategy to search for optimal neural architecture. In our framework each of the sub-graph is prefixed and identical to one another in terms of network

architecture. The target of our framework is to leverage light weight neural networks as the sub-graph (needless to say multiple light weight sub-graphs together compose the super-graph) and boost the performance of respective sub-graph networks by leveraging the knowledge of complex super-graph through distillation.

We perform all the experiments with different ResNet networks as the backbone (Resnet-20, 56, and 110). In the experiment, we show that our proposed framework can substantially boost the performance of each sub-graph relative to the original backbone ResNet network. Once the network training and validation are completed, based on the computational availability either sub-graph, super-graph or ensemble of all sub-graphs can be leveraged for the inference task. We can easily sample the best performing sub-graph from the framework for the inference task. It is worth noting that, our primary goal is not focused on achieving an improved ensemble of neural networks, rather leverage the knowledge of the complex structure of the super-graph network to boost sub-graph performance. However, while doing so, we have actually obtained good ensemble performance. In this way, we preserve the simplicity of the end network while achieving performance boost. Our contribution to this article can be summarised as follows:

- We propose a simple modular-based structural learning framework for the DCNNs. We built the framework upon the concept of sub-graph/super-graph design. All the sub-graphs are stand-alone neural networks that we train by distilling knowledge from the complex super-graph.

- We explore three variants of super-graph architectures i.e. different ways of combining the output of all the sub-graphs while training the super-graph. The aggregation methods are i) concatenation of Global Averaged Pooled (GAP) features from all the sub-graphs, known as Feature Concatenation (FC) method; ii) averaging the GAP feature maps from all the sub-graphs known as Feature Averaging (FA) method; and iii) Log-Softmax Average (LSA) of output from individual sub-graphs. We provide an empirical study on these three variants of feature aggregation methods for CMNN on various datasets and conclude that FC-based CMNN provides the best super-graph and sub-graph networks.

- The third contribution is the novel part. In order to effectively leverage the complex structure of the super-graph and obtain a strong sub-graph network, we propose a simple and effective knowledge Distillation based Round-Robin training procedure. We confirm the effectiveness of this distillation-based training procedure through empirical study, Class Activation Mapping (CAM) analysis of the sub-graph and super-graph networks.

- We validate the implementation of CMNN on CIFAR-10, CIFAR-100 and Tiny ImageNet datasets. In addition to that, we also perform experiments where we use CMNN as the backbone network of On-Road Risk (ORR) classification system that we have designed for the mobility scooter in the literature [4].

We arrange the paper in the following order.

1. Section 4.2: This section covers key researches conducted for neural network compression through knowledge distillation, efficient architecture search, super-graph/sub-graph type architectures, and so on.

2. Section 4.3: An overview on Coupled Modular Neural Network (CMNN) architecture.

3. Section 4.4: Here we present an explanation on various aggregation methods of feature vectors from sub-graph networks while training super-graph. The explanation also includes the formulation of objective functions and step-by-step explanation of algorithm to train CMNN.

4. Section 4.5: This section includes implementation settings, dataset preparation, and all empirical result discussion through Tables and Figures.

5. Section 4.6: The section discusses on limitations of the CMNN. Later, few recommendations are provided to mitigate the limitations to a certain degree.

4.2 Literature Review

The task of obtaining an efficient and accurate neural networks has been approached from different points of view, such as through efficient architecture design and optimization, introducing regularization terms in loss function, and so on. In the section, we explore and explain research works on neural networks which share similar motivation as ours.

Knowledge Distillation (KD) is currently the most popular and simple to implement approach for compressing neural networks without significant loss of performance. The concept was first introduced in [91] depicted as *model compression*. The key idea of *model compression* [91] was to train a compact and lightweight network to mimic the complex function learned by a cumbersome ensemble [65]. Instead of training the compact neural network on an original and small set of datasets, the training of the network was carried out on a large set of a dataset that is pseudo-labeled by the huge and complex ensemble of neural networks. This approach pushes the smaller neural network prediction capability close to the set of ensemble neural networks. A more generalized version of model compression has been proposed in research [45], where instead of pseudo-labeling approach, knowledge transfer was carried out by teacher-student training set up. The key idea was to optimize a loss function where the logits (the inputs to the final softmax) of the student model are pushed to match the softened logits of the cumbersome teacher model (or ensemble of multiple teacher models). This training procedure thus encourages the student model into learning the class probability distributions by the teacher model.

Previously KD distillation strategy was off-line which consisted of two steps, first, the teacher model required to be trained, and second, the distillation of knowledge from the trained teacher to students. To mitigate this limitation, research works [59] proposed one step distillation process where both teacher and students mutually learn from each other during the training phase. In our proposed framework we leverage a similar variant of the online distillation objective function to optimize the sub-graph known as the *co-distillation* proposed in [129]. Originally, the co-distillation of [129] allows training multiple copies of independent (no weight sharing) neural networks in parallel by enabling each model to match the average prediction of other models. The distillation-based training was carried out even before the teacher model has fully converged. A similar distillation-based approach was proposed in [58] termed as the *"On the fly Native Ensemble (ONE)"*.

A key contrast between large scale co-distillation training [129] and *ONE* [58] is the neural structural design. In *ONE* framework the teacher model is a multi-branch network with identical auxiliary branches replicated for certain times. Each of the particular branches is referred to as a student model and is infused within the teacher network. Both teacher and student models are trained simultaneously. The student models (also known as the branch) in *ONE* are encouraged to mimic the output of the teacher network. The output of the teacher network is basically the weighted average of softened logits from all the branches. During inference, the best-performing branches (or student models) are selected.

Our proposed framework thus has a very close resemblance to [58, 129] in a sense that, the objective function pushes each of the sub-graphs to match the aggregated output of the complex super-graph. However, in contrast to [58, 129], i) our framework explores different aggregation methods of the sub-graph network output during training phase; and ii) the architecture design, backbone architectures, and training scheme for the whole framework are substantially

different from previous related researches. The training procedure that we present in this literature is simple but effective for coupled sub-graph type networks which we substantiate through extensive empirical study.

Ensemble Learning (EL) is regarded as the machine learning interpretation for 'Wisdom of the crowd' where a set of classifiers are trained individually [141], and during the inference phase prediction of each trained classifiers are combined to make the final prediction [126]. This method achieves very reliable fault-tolerant computation and suitable bias-variance trade-off. Some of the most commonly practiced EL methods are AdaBoost [142], Bagging [63], Random Forest [106], Gradient Boosting [107] and so on. Usually, most of the machine learning models (such as decision trees, support vector machines, multi-layer perceptron are the popular ones) can be implemented into these EL learning frameworks as the backbone classifier. However, modern neural networks have also been observed to benefit from ensemble techniques, particularly CNNs [143]. It is needless to say that the training and testing phase of the ensemble framework is very expensive. Building on the stated fact, the ensemble of neural networks can be much more resource-hungry, which can limit the practical applicability [101]. Research [144] proposed strategies to mitigate the tedious problem of training many neural networks in an ensemble framework by saving multiple copies of weight checkpoints while training a single neural network. Each of the checkpoints was assumed to be converging to several local minima on its optimization path. While this approach mitigates the training time problem, the inference and model size problem still prevails. Slightly similar work can be found in [117], where during the training phase periodic restarts were simulated by re-initializing the learning rate to some value and eventually scheduled to decrease. Afterward, all the models before and after restart are leveraged in the ensemble inference. *Our proposed learning framework is not an ensemble of neural networks. In our framework, we rather leverage the whole network (termed as the super-graph) to train and boost the performance of each individual sub-graph. Of course, given enough resource budget we can utilize our framework like the classical ensemble method where all the branches prediction are aggregated using several combination strategies.*

Multi-Branch and Multi-Column networks are very popular and common design paradigm for current state-of-the-art neural networks. This design paradigm advocates the usage of common template building blocks stacked in a sequential manner to form the complete model [24]. These template building blocks are built upon the concept of grouped convolution, where multiple kernels of varying sizes are present per layer. This results in a wider neural network with relatively fewer parameters and rich features. Such kind of approach can be termed as local level branching. On the other hand, [24] proposed multi-branch architecture which is branched in global level, i.e. multiple architectures of networks are coupled in parallel to form the final composite network model. Research [145] proposed hierarchical tree-structured neural networks where multiple neural networks are coupled together through intermediate layer sharing. During inference, depending on the computational budget more or less neural networks within the tree are evaluated. In addition, the paper also proposed a novel KD method to encourage diversity and boost the performance of individual networks within the tree of a neural ensemble. *In contrast to these remarkable approaches where multiple neural networks are leveraged for the prediction we focus on boosting the performance of a single neural network through constructing a large super-graph network consisting of multiple baseline sub-graph networks. In addition, proposed learning framework preserves the diversity among these sub-graph networks.*

Super-graph and sub-graph networks, also known as the hyper-net and sub-net in the literature, have proven to mitigate the expensive process of NAS by drastically reducing the search space. Moreover, networks with a super/sub-graph architecture can substantially reduce the number of parameters by encouraging coupling and parameter sharing across the sub-graphs. Research work [97] has proposed an efficient NAS that sampled optimal sub-graph network from a large computational graph by leveraging a reinforcement learning-based controller net-

work. It has been demonstrated that such approach can reduce the search cost of standard NAS process by a factor of $1000\times$, and yet achieve competitive performance for standard image classification tasks. One shot model architecture search by [138] proposed technique to accelerate the architecture search by constructing an auxiliary hyper-net (analogous to our super-graph) that generated the weights for several sub-nets. The sub-nets were later validated on the classification task, where it showed competitive performance. A recent work [139] proposed a novel concept of Neural Architecture Transfer (NAT), where a pre-trained multi-objective super-net was constructed, from which task-specific sub-nets were sampled without requiring any expensive additional training. The process was carried out in an iterative manner, where the super-net was repeatedly adapted while in the meantime, searching for task-specific sub-nets. Besides visual image classification task, sub-graph based deep networks have demonstrated success in the graph classification task. Such as, Research work [146] proposed novel sub-graph based self-attention network, where different sub-graphs learn different level of features. Similarly, research work [147] proposed sub-graph based neural network to learn disentangled sub-graph representations, which is considered relatively challenging task. Graph classification model known as the SUGAR [148] was proposed that performed through sub-graph level selection to learn discriminative sub-graph representation. *Although our proposed super-graph/sub-graph based network is mainly focused on visual image classification task, it is thus clear that, the concept of sub/super-graph can be a general scheme for solving problems of other domains.*

Modular design with gating mechanism for the neural networks has proven to be very efficient during the inference phase. Research [74] proposed a neural architecture that is composed of many additional modules besides the main module. Each main module delivers error gradients and tries to approximate the loss and output of the main module. Research [67] proposed a multi-branch network that is equipped with a gating network. Each branch was trained to specialize in the sub-task. The inference cost was optimized using the gating network. Research [45] proposed a modular type network that consisted of two main parts, a generalist network and a set of independent expert networks. The expert network is trained on the set of data that are often confused and mis-classified by the generalist model. Recently, in researches [44, 101] we proposed a modular neural network framework consisting of gating and expert networks. The gating network was leveraged to dynamically select expert neural networks for further inference. The expert neural networks were trained on data consisting of subset of classes that was prepared systematically in a Round Robin fashion.

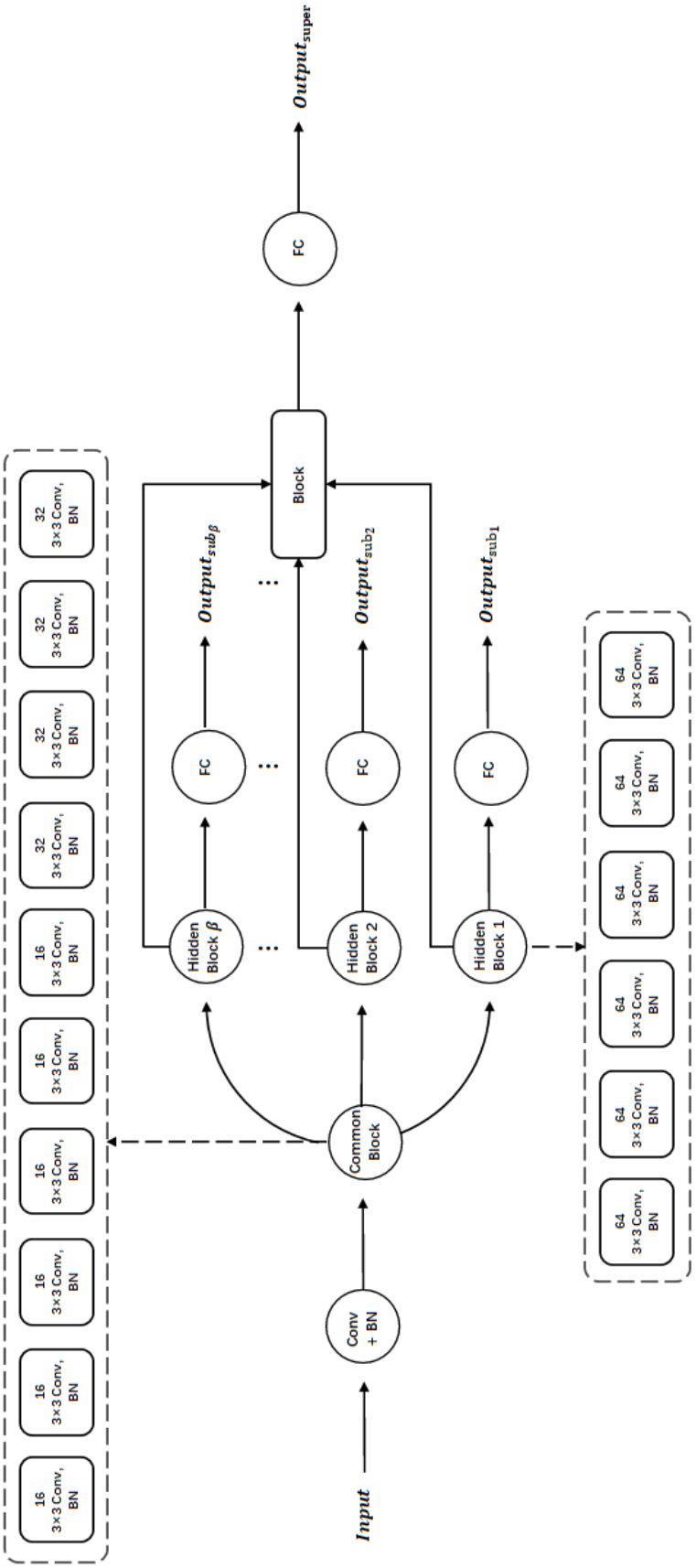


Figure 4.1: **Architecture of CMNN**

4.3 Coupled Modular Neural Network

The framework primarily consists of two parts, sub-graphs and super-graph. We depict the detailed architecture of the framework in Figure 4.1. In the following paragraph, we give a clear overview on each part of the framework.

- **Sub-graphs:** In the context of the proposed framework, a sub-graph is a building block of a super-graph and a stand-alone neural network. The sub-graph network is built up of a shared common block, a single hidden block followed by a fully connected linear layer for classification task. There are multiple sub-graph networks in the framework and the number of sub-graphs within a single super-graph is determined by the number of times each hidden block is replicated (refer to Figure 4.1 for clear demonstration). The term controlling this replication of hidden block is known as the branching factor which is denoted by β in this literature. Each block of sub-graph (such as common block, hidden block) consists of series of operations acting on the input such as convolutional operation, max or average pooling, batch normalization, Rectified Linear Unit (ReLU), and so on. Each of the hidden blocks should have an identical series of operations. The common block in this framework is shared among all the sub-graphs owing to the assumption that, initial layers are responsible for learning the primary common features [58]. In addition, this *shared weight* design method enables us to reduce the training cost [58]. The later part of the network i.e. the hidden blocks and respective linear layers allow the network to learn diverse features. At the end of the training, we sample the best sub-graph network for the inference based on its performance on the validation set.

- **Super-Graph** comprises of the shared common block, all the hidden blocks (excluding the linear layers attached to each hidden block for classification), aggregation block followed by a linear layer for classification. The aggregation block and the linear layer are the only components that make the super-graph different from its constituent unit (i.e. sub-graph). The implementation and definition of aggregation block can be different based on the choice of combining the output of hidden block units. After performing the aggregation the network passes aggregated features of hidden blocks to the linear layer which is responsible for classification and loss calculation. The width and complexity of the super-graph increases as we increase the variable β for the network.

An important issue in constructing such a network is the appropriate design choice for each particular block [67]. To overcome this issue we leverage series of ResNet neural network architecture as the backbone. ResNet networks are very well-known for mitigating the vanishing gradient problem by leveraging the skip connection layers known as the identity shortcut connection. As a result of this shortcut connection deeper network performs at least as good as the shallower counterparts and if not better. For our framework, the ResNet series network is suitable for its inherent compartment-like architecture. The ResNet series architectures (for CIFAR-10 and CIFAR-100) in general consist of $6n + 2$ layers where n can take values of $\{3, 5, 7, 9, 18\}$. These stacked $6n$ layers can be divided into three blocks, where each group of $2n$ convolutional layers consist of $\{16, 32, 64\}$ filter maps respectively [25]. Thus, this compartment-type architecture of ResNet based on the number of filter maps is suitable for designing each particular block of our framework. More precisely, we can assign each group of $2n$ CNN layers for each block of CMNN. However, there still rises a question on how many CNN groups should be assigned for the shared common block and the hidden blocks. Similar to the basic design choice of [58] the first convolutional layer and $4n$ layer group of ResNet (also known as the $conv2x$ and $conv3x$ blocks in the original literature [25]) are assigned for the **shared common block**. The assumption is that the earlier layer is mostly responsible for learning primary features which can be shared across all the hidden blocks, thus reducing the training cost and encouraging parameter reuse [58, 67].

We allocate the rest of the $2n$ (known as the $conv4x$ block) layers configuration of the ResNet to the **hidden block**. We replicate the last $2n$ layers β times where β is the branching factor. These hidden block parameters are independent of one another and initialized randomly. The assumption is each of these hidden blocks learns diverse features during the training phase, and we aggregate these features and later distill the knowledge back to every single sub-graph. We will discuss more on aggregation method in section 4.4.

Apart from the layer re-configuration for the CMNN we introduce a few implementation modifications to the ResNet architecture. Such as, we replace all the $ReLU$ activation with the $LeakyReLU$. $LeakyReLu$ takes negative slope angle as its parameter [132]. We adhere to the default parameter settings of PyTorch which is $1e^{-2}$. We also change the stride values of the CNN layers in $conv4x$ to 1 (originally it was 2). We adopt this approach to encourage the last block to learn fine-grained features.

4.4 Training Procedure of CMNN

The training phase for CMNN consists of three key steps, i) first pre-train the super-graph for a certain number of epochs (this step is crucial as it gives a very good starting point for the sub-graph networks), ii) train sub-graph network using the distillation loss where the teacher network is the pre-trained super-graph network and ii) next, fine-tune the super-graph at the end of training one sub-graph. We perform the later two steps alternatively, i.e choose one sub-graph network from many (in Round-Robin fashion), train the sub-graph by leveraging the super-graph network as the teacher and then fine-tune the super-graph to produce a different copy of teacher network for the next sub-graph training (refer to Figure 4.2 for a visual demonstration).

In the following paragraph, we introduce and explain all the important notations for network parameters, objective functions, and so on.

First, let us assume that we have training dataset $\mathcal{D} = \{d_i | i = 1, .., N\}$, where N is the number of samples in that dataset. $\mathcal{T} = \{t_i | i = 1, .., N\}$ is the corresponding ground-truth labels, where t_i is associated with d_i for $i = \{1, 2, ..., N\}$. Next, we denote the sub-graph networks which say takes an input d as $f_b^{sub}(d; \theta_b)$, where $b = \{1, 2, ..., \beta\}$. Thus, we have β sub-graph networks and there corresponding parameters are $\Theta = \{\theta_1, \theta_2, .., \theta_\beta\}$ respectively. Now, for the ease of explanation we further expand the parameter θ_b of each sub-graph as follows.

$$\theta_b = \{\theta_c, \theta_{b_h}, \theta_{b_f}\} \tag{4.1}$$

where, θ_c, θ_{b_h} and θ_{b_f} are shared common block, hidden block and fully connected layers parameters respectively. We can express the sub-graph network as $f^{sub}_{b_{lin}}(f^{sub}_{b_{hid}}(f_{com}(d; \theta_c); \theta_{b_h}); \theta_{b_f})$ which gives us the logit vector $\mathbf{z_b}$ as the output. Thus, logits $\mathbf{z_b} = [z_{b_1}, z_{b_2}, .., z_{b_C}]^\top$ where, the subscript b is the indicator stating from which sub-graph the logits came from. For ease of notation we denote the sub-graph as $f_b^{sub}(d; \theta_b)$ in short-form. For the super-graph let us assume the parameter sets as θ_s. Now θ_s will cover its own parameters and all the parameters of sub-graphs as well, depicted as follows.

$$\theta_S = \{\theta_c, \theta_{S_F}, \theta_{b_{h[1:\beta]}}\} \tag{4.2}$$

Thus, the super-graph network can be expressed as $f^{sup}_{lin}(f^{sub}_{b_{hid[1:\beta]}}(f_{com}(d; \theta_c); \theta_h); \theta_F)$ or in-short form as $f^{sup}(d; \theta_S)$. The super-graph $f^{sup}(d; \theta_S)$ outputs logit vector z_{comb} where, the subscript $comb$ actually depicts the way we obtain the logits which can be through Feature Concatenation (FC) (con), Feature Average (FA) (avg) or Log-Softmax Average (LSA). Each module of the sub-graph i.e. the common block $f_{com}(d; \theta_c)$ and hidden block $f^{sub}_{b_{hid}}(f_{com}(d; \theta_c); \theta_h)$ give us the high-level and lower-level feature maps respectively.

4.4.1 Aggregation method for hidden blocks

In this literature, ***we explore three different variants of combining methods of sub-graph's output to construct the super-graph***. These methods are:

1. **Feature Concatenation:** In this method we first obtain the sub-graph's common block feature and hidden block features from $f_{com}(d;\theta_c)$ and $f^{sub}_{b_{hidden}}(f_{com}(d;\theta_c);\theta_h)$ respectively , where $b = \{1, ..., \beta\}$. Let us denote the common block feature vector as o_{feat} and hidden block feature vectors for the sub-graphs as o_{b_h}. Next, these feature vectors go through the GAP layer where features get transformed to single linear vector of size 64. We concatenate each of this vector of length 64 to obtain final feature vector of length $64 * \beta$ which we depict as follows.

$$out_{con} = GAP(o_{feat})||GAP(o_{b_{h_1}})||..GAP(o_{b_{h_\beta}}) \tag{4.3}$$

Here, $||$ depicts the concatenation operation between consecutive feature vectors. These feature vectors once again pass through the GAP operation. Finally, we feed these concatenated and pooled feature vectors to the fully connected layer of the super-graph to obtain logits $\mathbf{z_{con}} = [z_{con_1}, z_{con_2}, .., z_{con_C}]^\top$. The fully connected layer of super-graph with concatenation combination method has a shape of $(64 * beta, C)$, where C is the total number of classes available in the dataset. It is worth noting that for the concatenation operation the length of the feature vector increases as we increase the value of β. This also assists in preserving more feature information from all the hidden blocks and common blocks which we will demonstrate through the Class Activation Mapping (CAM) analysis in the later part of the literature.

2. **Feature Averaging:** Feature Averaging in this literature is the operation of averaging the feature map from all the sub-graphs hidden block CNN layers. While all the operation is analogous to the previous mentioned FC method, the only contrast is the feature combining method which results in a feature vector of length only 64.

$$out_{avg} = \frac{1}{|\Theta|}\sum_{\theta_i \in \Theta} GAP(f^{sub}_{b_{hid}}(f_{com}(d;\theta_c);\theta_i)) \tag{4.4}$$

where, $\Theta = \{\theta_1, \theta_2, .., \theta_\beta\}$ and $|\Theta| = \beta$. The size of out_{avg} is 64 which implies that the fully connected layer of super-graph will have a shape of $(64, C)$ neurons.

3. **Log-Softmax Average:** This combination method averages the Log-Softmax confidence predicted by each sub-graph network for the target class. The combination method is quite similar to the Loss Average (LA) method of the multi-branch network of [24]. However, unlike the FC and FA method where we leverage the hidden block features of the sub-graph, the LSA method leverages the fully connected layers of the sub-graph networks. Thus, during the super-graph training phase, we combine and average the LogSoftmax confidence obtained from the logits of all the sub-graphs fully connected layers as follows.

$$p_{avg}(c|d) = \frac{1}{\beta}\sum_{b=1}^{\beta} log(p_b(c|d,\theta_b)) \tag{4.5}$$

where the probability distribution vector for target classes obtained by sub-graph networks through the softmax is defined as follows:

$$p_b(c|d,\theta_b) = SM(z_{b_c}) = \frac{exp(z_{b_c})}{\sum_{i=1}^{C} exp(z_{b_i})} \tag{4.6}$$

Where, $c \in \{1, .., C\}$, and C is the total number of classes available in the dataset $\mathcal{D}$ and, vector of logits $\mathbf{z_b} = [z_{b_1}, z_{b_2}, .., z_{b_C}]^\top$ for the branch b. For the LSA combination method, each of the sub-graph networks undergoes full-forward propagation during the super-graph training.

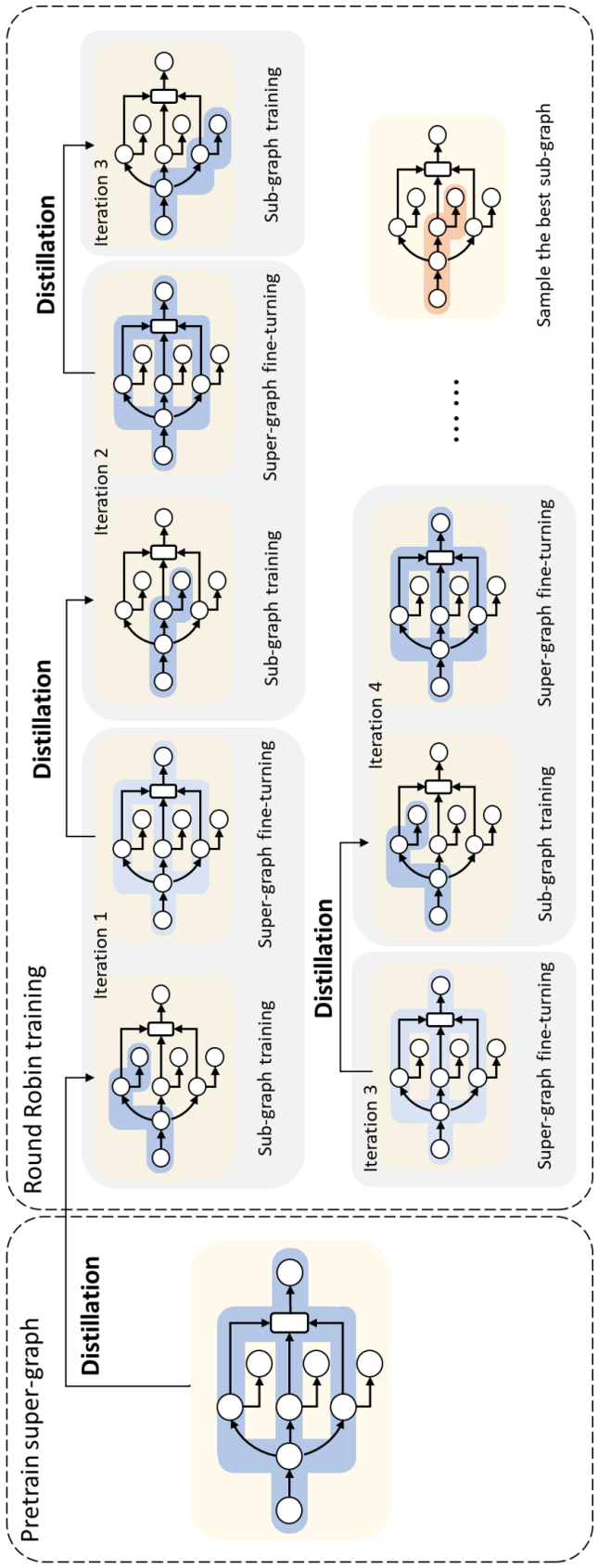


Figure 4.2: **Training Phase of CMNN. This figure depicts the novelty of the training algorithm for the CMNN. First, we train the super-graph network for certain epochs. Once we have a pre-trained super-graph the Round-Robin-based training of the sub-graph begins. The blue shade depicts the sub-graph that is under-going training in that certain epoch. While training the sub-graph the rest of the graph is frozen, i.e. error gradient is prohibited to flow through the rest of the graph. Second phase of Round-Robin training is fine-tuning, where the super-graph is unfrozen and fine-tuned with a low learning rate. After each fine-tuning phase we have slightly different copy of the super-graph network which acts as the teacher network for the next Round-Robin iteration.**

4.4.2 Objective Functions for Super-graph Training

The first objective is to achieve a strong super-graph network. From the previous section we observe that the super-graph network $f^{sup}(d;\theta_S)$ gives z_{comb} logits which are un-normalised log probability output by the network. Thus we can define,

$$p_{sup}(c|d,\theta_S) = SM(z_{comb_c}) = \frac{exp(z_{comb_c})}{\sum_{i=1}^{C} exp(z_{comb_i})} \tag{4.7}$$

Here again, the subscript $comb \in \{con, avg\}$, i.e. we can obtain logits from either FC or FA version of super-graph. We leverage the cross-entropy loss to train the super-graph network.

$$L_{comb}^{sup} = -\sum_{i=1}^{N}\sum_{c=1}^{C} \delta(t_i, c)\log(p_{sup}(c|d_i,\theta_S)) \tag{4.8}$$

where, $\delta(t_i, c)$ is the Kronecker delta function defined by

$$\delta(t_i, c) = \begin{cases} 1, & t_i = c, \\ 0, & t_i \neq c \end{cases}$$

However, for the LSA version of the super-graph network, we have a slight change in the objective function. In LSA combination method all the sub-graphs of the network undergo forward propagation to obtain the logits $\mathbf{z_b}$ and eventually we apply softmax on those logits as we depict in the Eqn. 4.6. Afterward, we average the LogSoftmax of all the sub-graph networks which we depict as p_{avg} in Eqn. 4.5. Once we obtain p_{avg} we can perform the cross-entropy loss as follows:

$$L_{avg}^{sup} = -\sum_{i=1}^{N}\sum_{c=1}^{C} \delta(t_i, c)\log(p_{avg}(c|d_i)) \tag{4.9}$$

4.4.3 Objective Function for sub-graph training

First, we perform warm-up pre-training of super-graph before we start full-scale training of sub-graph networks, as this procedure provides a good initialization for the sub-graph networks. Once we obtain the pre-trained super-graph the training procedure continues in a Round-Robin fashion, where we train each sub-graph through distillation followed by fine-tuning the super-graph (refer to the Figure 4.2). The loss term that we optimize for any sub-graph b is depicted in its simplified form as follows:

$$L_b^{sub} = \alpha L_{ce}^{sub} + (1-\alpha) L_{kd}^{sub} \tag{4.10}$$

where, $0 < \alpha < 1$. Here, α is the hyper-parameter that controls the weight and emphasis between KD and cross-entropy loss. As our core idea of constructing the super-graph from many sub-graphs is actually to boost the performance of each sub-graph we tend to keep α fixed to a very low value of 0.2. This depicts that throughout the training process of the sub-graph networks we emphasize distilled knowledge from the super-graph network. However, performing cross-validation of KD hyper-parameter α may give us better training set-up with an increase in the cost of extra model training. In this literature we do not perform a detailed empirical study on the effect of the hyper-parameter α, instead, we seek guidance from the previous study [45, 58, 149] for choosing the optimal value of α. In general, there is a lot of encouragement to leverage higher weights (i.e. low alpha value) when the teacher network is strong.

Now returning back to the Eqn. 4.10, we elaborate the equation which consists of the cross entropy term L_{ce}^{sub} and KD term L_{kd}^{sub}. The L_{ce}^{sub} calculates loss based on the output $p_b(c|d,\theta_b)$

of the sub-graph b as follows:

$$L_{ce}^{sub} = -\sum_{i=1}^{N}\sum_{c=1}^{C}\delta(t_i, c)\log(p_{sub}(c|d_i, \theta_b)) \tag{4.11}$$

The second term of the loss is the KD term which optimizes each of the sub-graph to bring them close to the performance of the complex super-graph. The calculation of KD for super-graph variants *FC* and *FA* differs slightly from the *LSA* method. Let us first formulate the KD term for the *FC* and *FA*.

To effectively enable KD we soften the probability distributions at temperature T (here T = 5 for all cases) for the teacher super-graph as:

$$\begin{aligned}\tilde{p}_{comb}^{sup}(c|d, \theta_S) &= SM(z_{comb_c}; T) \\ &= \frac{exp(z_{comb_c}/T)}{\sum_{i=1}^{C} exp(z_{comb_i}/T)}\end{aligned} \tag{4.12}$$

Now, if the choice of combing method for the super-graph teacher network is *LSA* we will require to soften the probability distribution obtained from each sub-graph network. Thus, we require a slight modification in the Eqn. 4.6 as follows:

$$\begin{aligned}\tilde{p}_b(c|d, \theta_b) &= SM(z_{b_c}; T) \\ &= \frac{exp(z_{b_c}/T)}{\sum_{i=1}^{C} exp(z_{b_i}/T)}\end{aligned} \tag{4.13}$$

Next, we take the average of the softened probability distribution from each individual sub-graph as follows:

$$\tilde{p}_{avg}^{sup}(c|d) = \frac{1}{\beta}\sum_{b=1}^{\beta} log(\tilde{p}_b(c|d, \theta_b)) \tag{4.14}$$

Here $\tilde{p}_{avg}(c|d)$ is the average value of all softened probability distribution produced by all the sub-graph networks. This averaging method is quite similar to the branch's prediction averaging method depicted in earlier literature [58, 145]. In this literature, we thus have three choices of super-graph as teacher network that we can leverage to train the sub-graph network. These three choices are $\tilde{p}_{comb}^{sup}(c|d, \theta_S)$ and $\tilde{p}_{avg}^{sup}(c|d)$. For the ease of formulating the KD loss term let us assume all these three types of teacher super-graph network as $\tilde{p}_t(c|d, \theta_S)$ Thus, the KD term stands as:

$$L_{kd}^{sub} = -\sum_{i=1}^{N}\sum_{b=1}^{\beta}\sum_{c=1}^{C}\tilde{p}_b(c|d_i, \theta_b)\log\frac{\tilde{p}_b(c|d_i, \theta_b)}{\tilde{p}_t(c|d_i, \theta_S)} \tag{4.15}$$

In this literature we leverage the *Kullback Leibler divergence* to bring the sub-graph probability distribution close to the super-graph. It is also possible to leverage other loss terms.

4.4.4 Explanation of Round-Robin training algorithm

In this section we provide a brief explanation of the training algorithm for CMMN. We depict the pseudo-code in the Algorithm 4 and visual depiction of the training in Figure 4.2.

In Algorithm 4, line 1 - line 3 perform dataset and network initialization. At the beginning of the training process, we first perform pre-training of the super-graph network. The pre-training steps are performed in lines 6-8. Value of $epoch_1$ is 50 for all datasets. Next, we start the sub-graph network training from line 13. Value of $epoch_2$ is 300 for all datasets. In line 15 we perform Round-Robin selection of a sub-graph. Sub-graph optimization begins from line 16.

After performing a single pass in sub-graph we set the learning rate to a very low value (lr = 0.0001 in this study for all cases of the datasets) for fine-tuning step (refer to line 18). After training a single sub-graph network, we fine-tune the super-graph network with the defined low learning rate (line 19).

4.4.5 Computational Complexity Analysis of CMNN

There are two key factors that contribute to the computational complexity of the CMNN. The first one is the depth of the backbone ResNet network, and the second is the value of the branching factor β. In the following paragraph we provide a simple analytical formulation for the computational complexity of CMNN, and later in Table 4.9 we provide the FLOP counts for both sub-graph and super-graph networks (cost per unit sample during test phase) for several ResNet backbone networks.

The complexity during training is proportional to the value of the branching factor β. As we increase the value of β we will subsequently have more sub-graph networks coupled to compose a complex super-graph. Although the complexity or the size of the common block (refer to Figure 4.1) remains constant for any value of β (because the common block is shared among all the sub-graphs), the number of hidden blocks increases proportionally. For ease of understanding and formulation, let us consider a function $\mathcal{K}$ that calculates the cost of any particular block. We can assume that, the function $\mathcal{K}$ can return either the number of trainable parameters or the FLOPs count, given any block of the CMNN as the argument or input. The block can be i) the shared common block (cb); ii) a hidden block (hb), or iii) a feature aggregation block (fab). The cost will include training a particular sub-graph network (that we choose by Round-Robin fashion) and fine-tuning the whole super-graph per epoch. Thus, during training the total estimated cost per epoch would be:

$$\mathcal{K}_{train}(CMNN) = 2 \times \mathcal{K}(cb) + \mathcal{K}(hb) \times (\beta + 1) + 2 \times \mathcal{K}(fab) \tag{4.16}$$

On the other hand, the estimated cost per epoch for training a single backbone network would be:

$$\mathcal{K}_{train}(backbone) = \mathcal{K}(cb) + \mathcal{K}(hb) + \mathcal{K}(fab) \tag{4.17}$$

From the Eqn. 4.16 and 4.17, ignoring the constant factors, it is indicating that branching factor β is the dominating part in determining the complexity of CMNN. Moreover, in order to provide equal training chances for all sub-graphs, the number of epochs needed to train the CMNN should be approximately β times larger than that needed for training the backbone network.

Complexity during the test phase depends on if we leverage i) a CMNN sub-graph, ii) a closely coupled super-graph, or iii) a loosely coupled ensemble of all sub-graphs. Since we are interested only in leveraging a single sub-graph in the current study, the test phase computational cost of the CMNN is basically similar to that of a stand-alone ResNet network. A generalized formulation for computational complexity for all cases would be:

$$\mathcal{K}_{test}(CMNN) = \mathcal{K}(cb) + \beta' \times \mathcal{K}(hb) + \mathcal{K}(fab) \tag{4.18}$$

When the value of $\beta' = 1$, we are leveraging only a single sub-graph network. For the cases ii) and iii), β' equals to the branching factor β. Based on the Eqn. 4.18 we provide computational complexity in the unit 10^8 FLOPs during the test phase for sub-graph and super-graph (per unit sample) with varying ResNet backbone networks and β' values in the Table 4.9. The Table also

depicts FLOPs count for rest of the models for comparison, which are directly illustrated from respective literature.

Algorithm 4: Training phase of CMNN depicted in Figure 4.2

Input: Dataset
Output: β branched Super-graph network.
Training Dataset: $\mathcal{D} = \{d_i | i = 1, .., N\}$
Teacher signal: $\mathcal{T} = \{t_i | i = 1, .., N\}$
Super-graph network: $f^{sup}()$
Sub-graph networks: $f_b^{sub}() \quad b = \{1, ..., \beta\}$
# First, pre-train super-graph network for $epoch_1$
for *i in* $enumerate(epoch_1)$ **do**
    for d, t *in* $enumerate(\mathcal{D}, \mathcal{T})$ **do**
        Train super-graph network $f^{sup}(d)$ by optimizing the Eqn.4.8 or Eqn. 4.9 based on the preferred feature aggregation methods.
    end
end
# sub-graph training starts from here
for *i in* $enumerate(epoch_2)$ **do**
    # select sub-graph in Round-Robin fashion
    $b = (b + 1)$ **mod** β
    for d, t *in* $enumerate(\mathcal{D}_{sub}, \mathcal{T}_{sub})$ **do**
        optimize $f_b^{sub}(d)$ using the Eqn.4.10
        Set learning rate of super-graph to lr = 0.0001
        fine-tune super-graph $f^{sup}(d)$
    end
end

4.5 Experiments

4.5.1 Datasets and Parameters settings

Datasets that we leverage to evaluate CMNN are CIFAR-10 (Canadian Institute For Advanced Research) [111], CIFAR-100 [111], Tiny ImageNet [150] and a private On-Road-Risk (ORR) dataset by [4]. CIFAR-10 dataset is a collection of 60,000 color images with 10 classes, where 50,000 images are for training and rest 10,000 for testing. CIFAR-100 data is similar to CIFAR-10 with 60,000 images. However, CIFAR-100 has 100 classes, thus containing 600 images per class. Tiny ImageNet is a subset of dataset from the original ImageNet. The dataset consists of 200 classes (subset of classes from 1000 classes of ImageNet) . There are total 100,000 images for training, and 10,000 images for validation. Thus, 500 images per class for training and 50 images for validation. The same validation set is actually leveraged as test set to report the performance of trained models in several literature. This is because the ground-truth for validation is publicly available. In addition, the dataset is down-sampled from 256×256 to 64×64 resolution with a view to make the dataset affordable in constrained computational environments [150].

Due to the very limited computing resources in a university laboratory, we have selected the above three public datasets for the visual object classification task evaluation. We also leverage the ORR dataset on the other hand to evaluate the performance of the network when integrated with the ORR classification system proposed by us in [4]. We will provide a brief overview and working mechanism of the ORR classification system in the later part of the literature

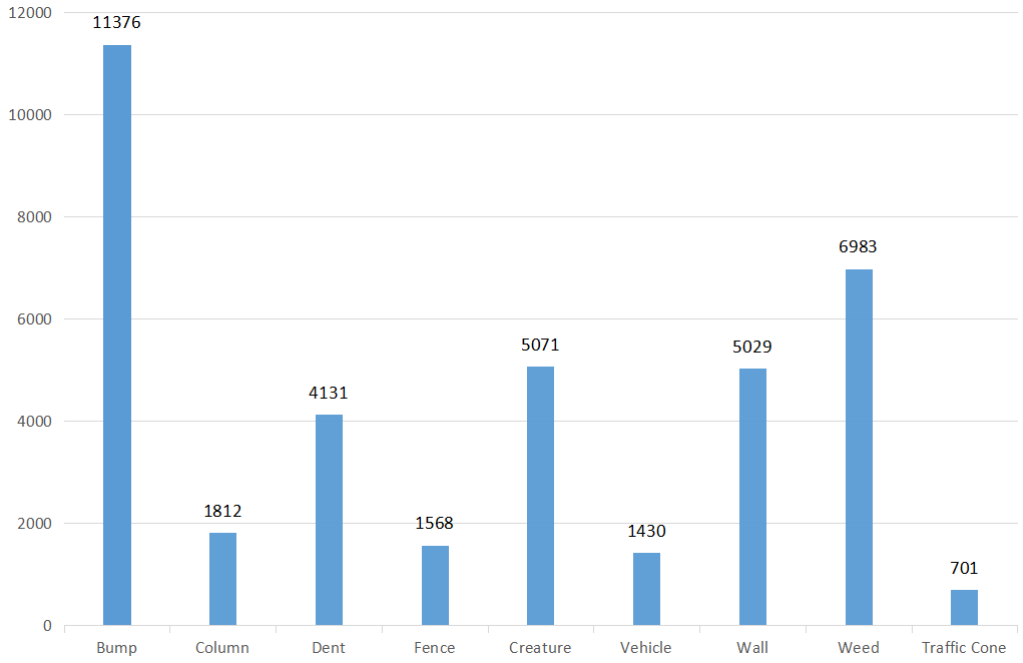


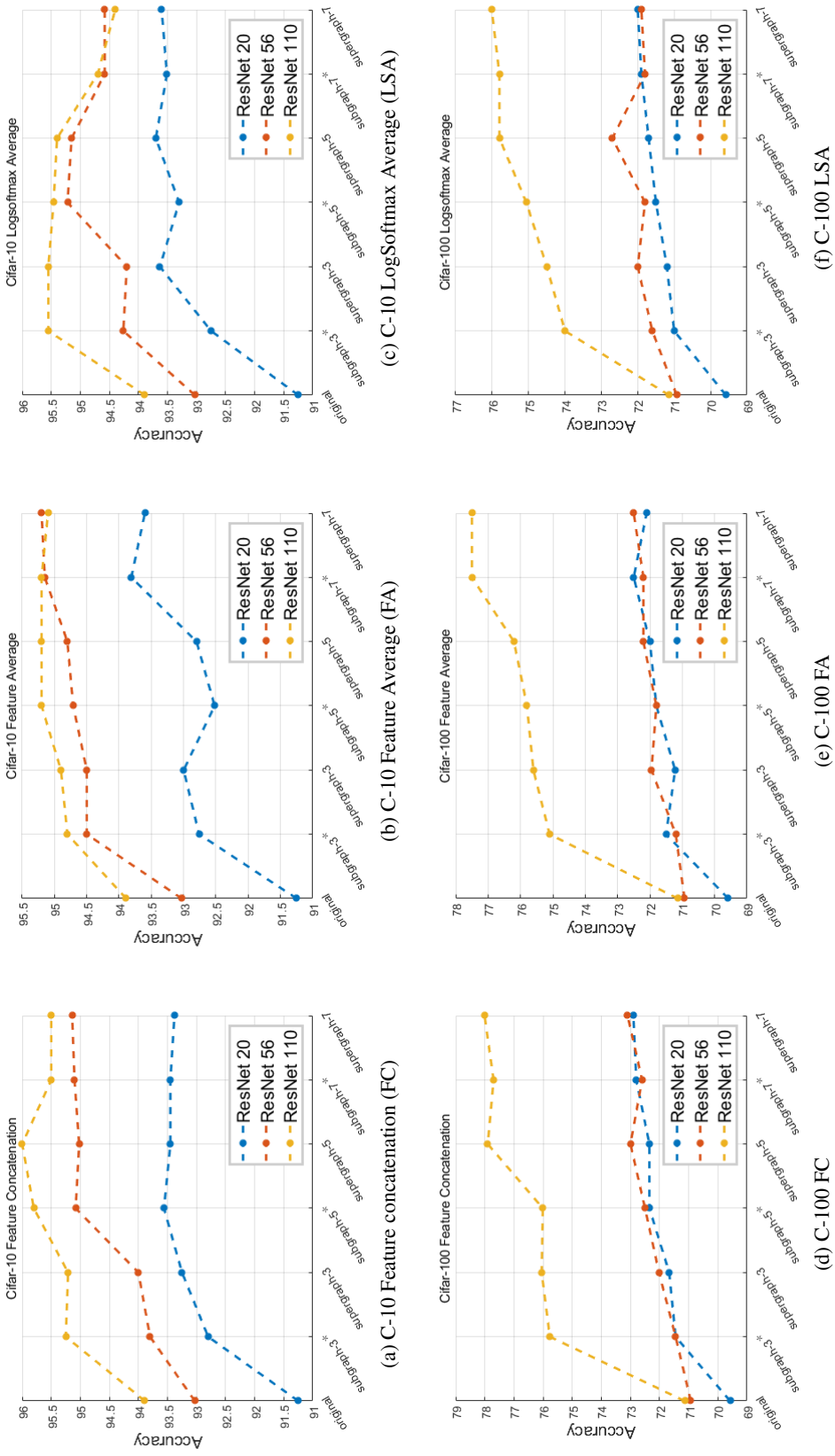
Figure 4.3: **Class statistics of ORR dataset. The image is directly depicted from the [3,4]** .

The ORR dataset by [4] is a private multi-class classification dataset which was constructed and leveraged to evaluate the ORR classification model. The dataset is a real-world on-road risk dataset that was collected using a *real-sense depth camera d435* mounted on the shoulder of a mobility scooter. Several videos were captured with this mounted camera which were afterward pre-processed, such as, conversion to still frames, selection of diverse images from a sequence of images, removing redundancy from the images, and so on. All these steps are taken to ensure that the dataset can provide a reliable measure of the performance of any classification model. The dataset primarily consists of 10 classes commonly encountered on the streets, which are, *bump, column, dent, fence, creature, vehicle, wall, weed, traffic cone, and normal* [4]. The class *normal* usually depicts safe situation. There are 5,774 images in the dataset. Each of these 5,774 images is further segmented into 22 small images known as the Cell of Interest (COI), resulting in a total of 127,028 images for fine-grained classification. All the images are labeled. For more detail, we recommend referring to the literature [4]. We provide a brief overview and statistics of the dataset in Figure 4.3. We will release the dataset soon to the public for academic purpose.

We implement **training and testing procedure** in PyTorch Deep Learning Framework [132]. We conduct all experiments on single *NVIDIA GeForce RTX 2080 GPU*. We keep most of the hyper-parameters during training consistent and similar for all the datasets. We use the Stochastic Gradient Descent (SGD) optimizer with momentum for training. The momentum is set to 0.9. As we mentioned earlier before the full-scale training starts we pre-train the super-graph network to provide each sub-graph network a good initialization point. Thus, we train the super-graph network for 50 epochs beforehand (applicable for all the datasets). Next, we begin the training of the sub-graph networks in a Round-Robin fashion. After each sub-graph network finishes an iteration the super-graph network is fine-tuned with a very small learning rate (lr). The initial learning rate for the training sub-graph network is set to 0.1 and fine-tuning learning rate for the super-graph is set to 0.0001. The total number of training epochs for all the dataset is 300 (for CMNN with $\beta = 3$). For larger CMNN with $\beta = \{5, 7\}$ we set training epochs to 500. Learning rate scheduler is set to change learning rate at epochs $\{80, 160, 250\}$ by a factor of 0.1.

For all the experiments, during the Round-Robin step (i.e. step 2) we leverage 80% of

the training datasets for training, while rest 20% for validation. During the step-2 process, since both sub-graph and super-graph will undergo training we need to reduce the training time (as we have many sub-graph to train). Thus, instead of feeding whole training data to each sub-graph network we perform training of each sub-graph on a small subset of training data consisting of all the classes. We prepare these subsets of dataset in a random fashion with replacement. Although this approach looks like Bagging [63], in practice it is not, because, during the fine-tuning of super-graph we eventually leverage all the training data. We save the best weight based on the performance on the validation set. In practice, various strategies and criteria can be adopted for saving the best weights, such as, saving weights based on the ensemble performance or based on super-graph performance. In this paper, we save weights based on the best performance of any of the sub-graph on the validation set. That is, we are saving weights based on the best accuracy obtained so far on the validation set by either of the sub-graph networks.



(a) C-10 Feature concatenation (FC) (b) C-10 Feature Average (FA) (c) C-10 LogSoftmax Average (LSA)

(d) C-100 FC (e) C-100 FA (f) C-100 LSA

Figure 4.4: **Empirical study of CMNN by varying the number of sub-graphs (1-7) and backbone networks (ResNet-20, 56, and 110) for CIFAR-10 (C-10) and CIFAR-100 (C-100) dataset. The first row depicts the performance on C-10 dataset and the next row on C-100. It is quite visible that as sub-graph increases the performance of super-graph increases (which is expected), also the corresponding sub-graph performance improves too due to distillation from teacher network.**

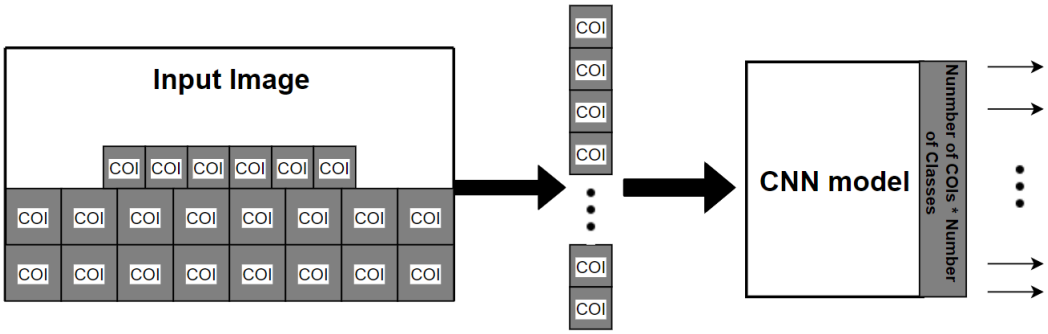


Figure 4.5: **Illustration of the ORR classification model. We directly depict this image from [4]. The input image was first segmented into certain pre-defined size patches. Each of patches depicts a certain location of the original image. These patches are next fed to the CNN classification model to predict the classes of patches. In this research, we set the backbone network i.e. the CNN-model with the sub-graph networks that we obtain from the our proposed framework. The sub-graph network after integration performs risk classification for the pre-defined cells or image patches. Any pre-defined cells if classified by the CNN network as "obstacle" is assumed to be risky for mobility scooter to proceed.**

.

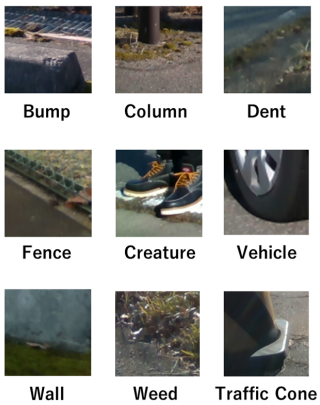
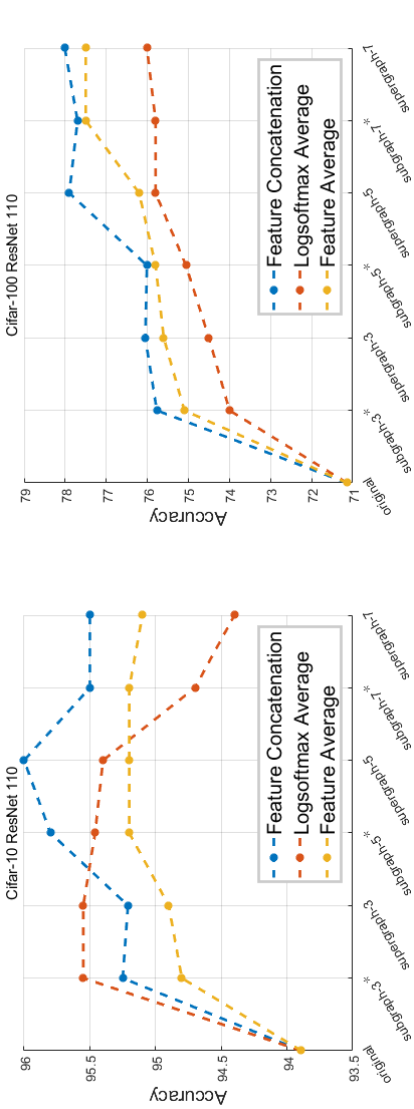


Figure 4.6: **Instances from ORR dataset**



(a) FC vs FE vs LSA for C-10 dataset

(b) FC vs FE vs LSA for C-100 dataset

Figure 4.7: **Performance comparison among FC, FE and LSA for C-10 and C-100 dataset**

Table 4.1: **Empirical study of CMNN super-graph and sub-graph networks on C-10 dataset with FC based training. The Table depicts only the best performing sub-graph that we represent with asterisk *. $|fV|$ is length of the final feature vector of the teacher network. 'Original' is the baseline accuracy of ResNet networks.**

Model	β	Sub* (%)	Sup. (%)	$\lvert fV \rvert$	Original (%)	Params. of sub (M)	Params. of Super (M)	δ (%)
CMNN-20	3	92.80	93.25	256	91.25	0.27	0.93	+1.55
	5	93.45	93.56	384			1.38	+2.20
	7	93.38	93.45	512			1.83	+2.13
CMNN-56	3	93.80	94.00	256	93.03	0.85	2.85	+0.77
	5	95.02	95.08	384			4.18	+2.00
	7	95.09	95.14	512			5.52	+2.06
CMNN-110	3	95.21	95.25	256	93.89	1.70	5.72	1.32
	5	95.70	96.00	384			8.39	+1.81
	7	95.40	95.50	512			11.05	+1.51

Table 4.2: **Empirical study of CMNN super-graph and sub-graph networks on C-10 dataset with FA based training. The super-graph parameter count is similar to the FC based super-graph which we depict in previous Table 4.1. For the original baseline score of the ResNet network refer to the previous Table 4.1.**

Model	β	Sub* (%)	Sup. (%)	$\|fV\|$	δ (%)
CMNN-20	3	92.76	93.00	64	+1.51
	5	92.52	92.80		+1.27
	7	92.85	93.60		+1.60
CMNN-56	3	94.50	94.60		+1.47
	5	94.68	94.80		+1.65
	7	95.15	95.20		+2.12
CMNN-110	3	94.80	94.90		+0.91
	5	95.20	95.20		+1.31
	7	95.20	95.10		+1.31

4.5.2 CIFAR-10 and CIFAR-100

We leverage CIFAR-10 and CIFAR-100 for detail empirical study on the effect of β. In addition, we also perform study on the effect of sub-graph combining method during the training phase for varying value of β. In order to keep the rest of experiments (with ORR and Tiny ImageNet datasets) simple and computationally tractable we adhere to the optimal training hyper-parameters (such as, smaller value of β, best sub-graph combining methods and so on) that we obtain from the study with CIFAR-10 and CIFAR-100.

Analysis of sub-graph combining method and the effect of the number of sub-graphs

In this section, we discuss about the performance of the proposed framework for three different sub-graph combination variants. The performance comparison for the three combination methods, *FC*, *FA* and *LSA* are depicted in Figure 4.7. In addition, Tables 4.1, 4.2 and 4.3 depict empirical study for the CIFAR-10 dataset. For the CIFAR-100 dataset we recommend referring to Tables 4.4, 4.5 and 4.6.

We train and evaluate CMNN with all proposed combination methods (backbone networks ResNet-20, 56 and 110) consisting of 3, 5 and 7 sub-graphs for CIFAR-10 and CIFAR-100 datasets. For simplicity we only represent the best sub-graph network score in all the Tables marked with the **asterisk** *. We depict the original performance of ResNet networks for CIFAR-10 and CIFAR-100 in Table 4.1 and 4.4 respectively under the column 'Original'. We also illustrate the performance improvement relative to the corresponding baseline ResNet network under the column 'δ'. δ is basically the performance difference between the best sub-graph and corresponding similar ResNet version, i.e. $\delta = Sub^* - Original$. Following points summarize our observation from the aforementioned Figures and Tables.

- **It is very important to construct an accurate super-graph network because an accurate super-graph can properly supervise a sub-graph network through distillation. It is quite intuitive that *FC* based super-graph network can provide a better teacher network for the sub-graph networks relative to the *FA* and *LSA* methods, since *FC***

Table 4.3: **Empirical study of CMNN super-graph and sub-graph networks on C-10 dataset with LSA based training.**

Model	β	Sub* (%)	Sup. (%)	$\|fV\|$	δ (%)
CMNN-20	3	92.75	93.64	64	+1.50
	5	93.30	93.75		+2.05
	7	93.50	93.60		+2.25
CMNN-56	3	94.20	94.26		+1.17
	5	94.68	94.80		+1.65
	7	95.15	95.20		+2.12
CMNN-110	3	94.80	94.90		+0.91
	5	95.20	95.20		+1.31
	7	95.10	95.20		+1.21

Table 4.4: **Empirical study of CMNN super-graph and sub-graph networks on C-100 dataset with FC based training.**

Model	β	Sub* (%)	Sup. (%)	$\|fV\|$	Original (%)	δ (%)
CMNN-20	3	71.45	71.67	256	69.00	+2.45
	5	72.35	72.35	384		+3.35
	7	72.44	72.90	512		+3.44
CMNN-56	3	71.45	72.02	256	69.30	+2.15
	5	72.50	73.00	384		+3.20
	7	72.60	73.10	512		+3.30
CMNN-110	3	75.77	76.05	256	71.14	+4.63
	5	76.00	77.90	384		+4.86
	7	77.70	78.00	512		+6.56

Table 4.5: **Empirical study of CMNN super-graph and sub-graph networks on C-100 dataset with FA based training**

Model	β	Sub* (%)	Sup. (%)	$\|fV\|$	δ (%)
CMNN-20	3	71.22	71.50	64	+2.22
	5	71.35	72.00		+2.35
	7	71.70	72.10		+2.70
CMNN-56	3	71.70	71.95		+2.40
	5	71.80	72.20		+2.50
	7	72.20	72.50		+2.90
CMNN-110	3	75.10	75.60		+3.96
	5	76.00	76.20		+4.86
	7	77.40	77.50		+6.26

Table 4.6: **Empirical study of CMNN super-graph and sub-graph networks on C-100 dataset with LSA based training**

Model	β	Sub* (%)	Sup. (%)	$\|fV\|$	δ (%)
CMNN-20	3	71.00	71.20	64	+2.00
	5	71.50	71.70		+2.50
	7	71.90	72.00		+2.90
CMNN-56	3	71.60	72.00		+2.3
	5	71.80	72.70		+2.5
	7	71.80	72.70		+2.5
CMNN-110	3	74.00	74.50		+2.86
	5	75.06	75.80		+3.92
	7	75.80	76.01		+4.66

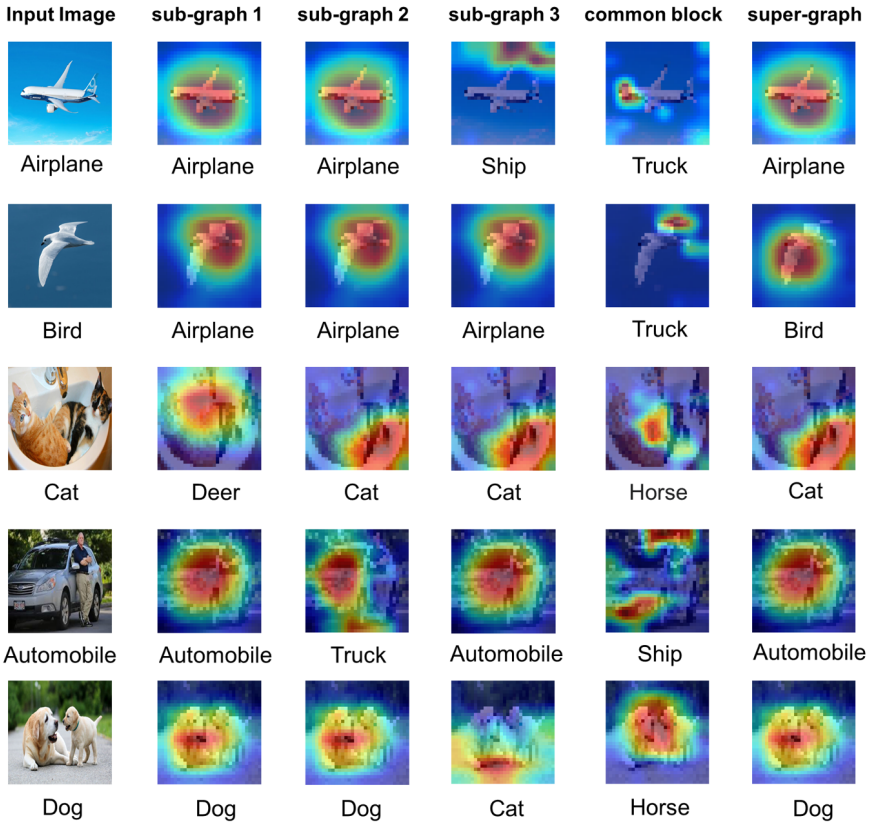


Figure 4.8: **Class Activation Mapping (CAM) analysis on CMNN super-graph, sub-graph, and common shared branches. It is expected that common branch CAM will not be precise in terms of quality. However, as we move to respective sub-graphs the CAM gets more focused on the object of interest region. In addition, the super-graph prediction and CAM are precise (as super-graph learns rich features from individual branches), and due to distillation during the training phase between sub-graph and super-graph, the sub-graph CAM is almost identical to the super-graph which is precise and of good quality.**

preserves more feature information from each sub-graph network. These rich features preservation come with slightly increased length in the final feature vector. As we can see from Table 4.1, as the number of branch increases the corresponding feature vector length $|fV|$ increases too. However, preserving these feature vectors assist in obtaining a relatively better and accurate sub-graph network. For example, CMNN super-graph with 5 coupled sub-graphs, with *FC* based combination method and with ResNet-110 backbone or in-short CMNN-FC-110-5 achieves 96.00% accuracy. Out of 5 sub-graph networks, the best sub-graph accuracy we obtain is 95.70%. This observation is also consistent and similar for CIFAR-100 dataset. For the CIFAR-100 dataset, the best sub-graph performs with 78.00% accuracy obtained through the *FC* based super-graph network.

- **All the sub-graph networks that we obtain through distillation from super-graph (type *FC, FA, and LSA*) improve substantially in-terms of accuracy relative to its corresponding baseline ResNet model (refer to the δ scores in all the empirical study Tables).** We can observe this performance improvement from the Figure 4.7. In the Figure 4.4 and 4.7, X-axis with the value 1 depicts the performance of the original ResNet networks. The subsequent labels such as $subgraph3^*$, $subgraph5^*$ and $subgraph7^*$ represent the performance of sub-graph networks that we obtain from their corresponding 3, 5 and 7 branched super-graph networks (denoted as $supergraph3$, $supergraph5$ and $supergraph7$) respectively.

- **Sub-graph networks that we train through *FC* and *FE* based super-graph networks show a consistent increment in performance as opposed to *LSA* method**. *LSA* has shown a slight inconsistency for the CIFAR-10 and CIFAR-100 datasets in terms of performance relative to both *FC and FA*. Our experiment findings from the *LSA* based super-graph network are actually consistent with the findings of [143]. It has been shown that averaging the logits or probabilities of multiple neural networks during the training phase has the negative effect of reducing the diversity in the gradients during the back-propagation through the ensemble.

- **As we increase the value of β for CMNN with any ResNet backbones (such as, ResNet-20, 56 or 110) there are increasing trend in performance for *FC* and *FE* variants**. This phenomenon is true for both the super-graph and the sub-graph networks as we illustrate in Figure 4.4 and 4.7. However, the performance saturates as the number of branches reaches beyond 5.

Comparison with Other Multi-branch networks

CIFAR-10 and CIFAR-100 datasets are commonly leveraged datasets for evaluating Image classification models. Due to its frequent use and practice, the number of benchmark scores for these datasets are also abundant. In the last couple of years, neural networks have achieved outstanding performance in these datasets. Such outstanding performance are the results of i) development of large scale transfer learning [40]; ii) unprecedented level neural network scaling in terms of factors such as depth, width, and learn-able parameters [151]; iii) efficient and accurate data specific neural network topology search through evolutionary algorithms [34], reinforcement learning [38].

Although these methodologies have proven to provide human-level accuracy (or sometimes surpassing human level) for recognition tasks they are expensive to carry out in lab-level resources. Thus, for the bench-marking and comparison, we adhere to neural networks which have close resembles to our approach in terms of factors such as, topology or architecture design, network size, and so on.

In Table 4.9 we depict the best performing sub-graph networks that we sample from the super-graph network. Each of the sub-graphs is a stand-alone ResNet network that we train

through our learning framework depicted in Figure 4.2. For ease of comparison, we represent sub-graph networks with 3 different backbone networks i.e. ResNet-20, 56, and 110. We present all the sub-graph networks obtained from the *FC*-based super-graph. We also present the performance of two large networks i.e. best performing super-graph networks and ensemble of its sub-graphs. We perform the ensemble by simply averaging the predictions of each sub-graph network within the super-graph.

From the Table 4.9 for the CIFAR-10 dataset sub-graph from *FC* based super-graph or in-short CMNN-FC-Sub with any backbone improves accuracy relative to *ONE-1* (i.e. with single branch) and *HNE* considerably while keeping parameter count very low as the original ResNet network. More precisely, CMNN-FC-Sub with ResNet-56 and 110 backbone perform with neck and neck score with both large and small multi-branch *HNE* network. Now, when we consider our largest networks, which are super-graph and the ensemble of sub-graphs, we have a substantial performance gain while maintaining moderate parameter counts. Such as, CMNN-FC-Ensemble-5 (i.e. CMNN with 5 coupled sub-graphs) which has around 8.39M parameters perform with 96.00% accuracy on test set. This performance is on par with ensembles by *ONE-1* [58] and Coupled Ensembles-6/8 [24].

For the CIFAR-100 dataset, our network has a good performance improvement when we consider the trade-off between network complexity and accuracy. Such as, CMNN-FC-Sub with ResNet-110 backbone or in-short CMNN-FC-Sub-110 has considerable improvement over most of the small multi-branch *HNE* [145] and multi-branch *HydraNet* [67]. *ONE* [58] on the other-hand obtained better testing accuracy for CIFAR-100 relative to the CMNN-FC-Sub-110. The ensemble of our sub-graphs has slightly better performance relative to the *ONE* ensemble. Compare to the large *HNE* [145] we achieve comparable performance.

From this comparative study, we can simply conclude two key points

- First, the sub-graph that we obtain through the super-graph actually performs with a competitive score while maintaining original parameter count.
- Second, our learning framework (visualized in Figure 4.2 and algorithm illustrated in Algorithm 4) encourages each of the sub-graph in the super-graph to be diverse from one another. A good improvement in ensemble accuracy supports this claim. Moreover, performing ensemble with this lightweight network is still affordable and simple when we compare it with a single complex model, thanks to the concept of shared backbone which encourages parameter re-usage [67].

Table 4.7: **Performance on ORR dataset**

Model	β	Comb.	Sub* (%)	Sup. (%)	δ (%)
CMNN-20	3	FC	92.95	91.30	+0.54
		FA	92.24	92.21	-0.17
		LSA	92.68	92.61	+0.27
CMNN-56		FC	93.10	92.88	+0.14
		FA	92.89	92.98	-0.07
		LSA	93.40	93.44	+0.44
CMNN-110		FC	93.50	92.48	+0.30
		FA	91.26	91.82	-1.94
		LSA	93.36	93.36	+0.16

Table 4.8: **Classification performance comparison for ORR dataset**

Source	Backbone	Acc. (%)	Params. (M)	FPS
ORR [3]	ResNet-50	90.51	23.52	23.46
	ResNet-101	90.50	58.15	15.84
	ResNet-152	89.02	60.19	12.03
Vanilla	ResNet-20	92.41	0.27	99.00
	ResNet-56	92.96	0.85	36.95
	ResNet-110	93.20	1.70	20.25
CMNN-FC (Ours)	ResNet-20	92.95	0.27	99.00
	ResNet-56	93.10	0.85	36.95
	ResNet-110	93.50	1.70	20.25

Table 4.9: **Performance comparison for C-10 and C-100 datasets. The numeric after each model name depicts the number of networks coupled within that framework. A value of one usually depicts a single stand-alone network. Value more than one either depicts ensemble or super-graph network. FLOPs are presented in the unit 10^8 FLOPs. For all the reported benchmarks the FLOPs count are measured per unit sample during test phase.**

Model	Backbone	C-10 (%)	C-100 (%)	Params. (M)	FLOPS ($\times 10^8$)
HNE-4 (small) [145]	ResNet	93.20	73.40	-	≈ 0.40
HNE-8 (small) [145]	ResNet	94.10	74.50	-	≈ 0.60
HNE-16 (Small) [145]	ResNet	94.30	75.30	-	≈ 0.80
HNE-4 (large) [145]	ResNet	95.10	78.00	-	-
HNE-8 (large) [145]	ResNet	95.50	79.00	-	-
HNE-16 (large) [145]	ResNet	95.60	79.80	-	-
ONE-1 [58]	ResNet-32	94.01	73.39	0.50	1.38
ONE-1 [58]	ResNet-110	94.58	78.38	1.70	5.05
ONE-1 [58]	ResNext-29 (8×64d)	96.55	83.93	34.40	-
ONE Ensemble-3 [58]	ResNet-110	-	78.97	5.10	8.29
Coupled Ensembles-6 [24]	DenseNet-BC L= 76, k= 33	97.08	84.24	25	-
Coupled Ensembles-8 [24]	DenseNet-BC L= 64, k= 35	96.87	84.05	25	-
Hydra-Res-d1-4 [67]	ResNet	-	65.81	1.28	0.52
Hydra-Res-d5-4 [67]	ResNet	-	73.84	7.59	3.17
Hydra-Res-d9-4 [67]	ResNet	-	75.26	13.19	5.81
CMNN-FC-Sub-1 (Ours)	ResNet-20	93.38	72.44	0.27	0.80
CMNN-FC-Sub-1 (Ours)	ResNet-56	95.14	72.60	0.85	2.50
CMNN-FC-Sub-1 (Ours)	ResNet-110	95.70	77.00	1.70	5.05
CMNN-FC-Super-5 (Ours)	ResNet-110	96.00	77.90	8.39	11.80
CMNN-FC-Super-7 (Ours)	ResNet-110	95.50	78.00	11.05	15.20
CMNN-Ensemble-3 (Ours)	ResNet-110	96.65	79.27	5.10	8.29
CMNN-Ensemble-7 (Ours)	ResNet-110	96.80	79.50	11.05	15.20

Table 4.10: **Performance of three branched CMNN super-graph and sub-graph networks on Tiny ImageNet dataset with FC based training. The Table depicts only the best performing sub-graph that we represent with asterisk** *.

Model	Sub.*(%)	Sup.(%)	Orig. (%)	δ (%)
CMNN-20	54.50	55.10	53.10	+1.40
CMNN-56	59.30	60.50	58.20	+1.10
CMNN-110	61.80	62.10	59.48	+2.32

4.5.3 Tiny ImageNet

Overview

Tiny ImageNet is relatively a difficult dataset compared to the CIFAR-10 and CIFAR-100. First, the dataset consists of twice more classes than CIFAR-100, second, the dataset has large number of fine-grained classes which are visually very similar (subset of ImageNet dataset). The images of this dataset are down-sampled thus loosing several fine details, resulting in fine-grained classification difficulty. The dataset has currently gained increased attention due to its reduced size and resolution, which allow experiments to be performed under constrained environment with in a limited time and computational budget. Most of SOTA literature that report benchmark scores are usually pre-trained on large ImageNet dataset. Later, fine-tuning was performed on the down-sampled Tiny-ImageNet dataset to achieve competitive score. For instance, recent visual image classification model known as the *Vision Transformer (VIT)* [43] fine-tuned a *large ImageNet pre-trained transformer* to achieve state-of-the-art accuracy of 84.65% on the ImageNet dataset.

Performance of CMNN and its backbone

For maintaining the consistency throughout the paper and in-order to keep the super-graph and sub-graph networks light-weight, we adhere to the similar architectures for experiments with Tiny ImageNet, i.e. the ResNet-20, 56, and 110 (same backbone networks that are designed for CIFAR-10/100 experiments). Now, it is true that for dataset like ImageNet and Tiny ImageNet, networks with larger number of feature maps and channels are preferable (such as ImageNet version ResNet-18, 34, 50, 101, 152 and so on), as demonstrated in previous experiments [153] and literature [154, 155]. It is practically possible to integrate ImageNet version ResNet to CMNN to construct more heavier and powerful super-graph and sub-graph networks. Also, theoretically, this approach will boost performance to a good extent (as we have seen in our earlier experiments that larger super-graph network provides better teacher networks). But, this can make computation very expensive, specially training phase which consists of periodic super-graph and sub-graph training. However, through this work we demonstrate that even with light weight ResNet networks as the backbone of CMNN, we can alleviate the performance and perform with accuracy comparable to most of the deeper networks.

First of all, we train the backbone networks ResNet-20, 56, and 110 on Tiny ImageNet with training hyper-parameters similar to CIFAR-10/100. In this experiment we perform all the training from scratch, i.e. we do not leverage any extra training data for pre-training the models. Now, these scores will serve us as the baseline reference to calculate the δ. Next, we train three branched *FC* based (with $\beta = 3$) CMNN with ResNet 20, 56, and 110 backbones. We depict the performance of CMNN and its respective backbones on Tiny ImageNet in Table 4.10. In Table 4.10 the fourth column depicts the performance of backbone networks ResNet-

Table 4.11: **Performance comparison of three branched CMNN (FC based super-graph training) with other multi-model networks on Tiny-ImageNet.**

Model	Acc. (%)	Params. (M)
DenseNet + ResNet [152]	60.00	-
Snapshot Ensemble-ResNet-110-6 (α_0=0.1) [144]	59.46	10.20
Snapshot Ensemble-ResNet-110-6 (α_0=0.2) [144]	60.60	10.20
Single Cycle Ensembles-ResNet-110-6 [144]	57.40	10.20
ResNet-20	53.10	0.27
ResNet-56	58.20	0.85
ResNet-110	59.48	1.70
ResNet-20 (Vanilla-Distillation)	54.00	0.27
ResNet-56 (Vanilla-Distillation)	59.10	0.85
CMNN-20-FC-sub-1 (ours)	54.50	0.27
CMNN-56-FC-sub-1 (ours)	59.30	0.85
CMNN-110-sub-1 (ours)	61.80	1.70
CMNN-110- FC-Super-3 (ours)	62.10	5.10
CMNN-110-Ensemble-3 (ours)	62.87	5.10

20, 56, and 110 with naive training settings. Next, the third column represents the performance of three branched CMNN super-graph with varying ResNet backbone, which we train through leveraging the proposed learning framework. It is quite clear that the three branched super-graphs of CMNN have considerable performance gain. Such as, super-graph with ResNet-110 backbone scores 62.10%. The best performing sub-graphs sampled from these super-graphs are depicted in the second column. Sub-graph networks (which are stand-alone ResNet networks) gain around $\{+1.40, +1.10, +2.32\}\%$ accuracy relative to the performance of its exact baseline architectures ResNet-20, 56, and 110 respectively. The improvement in performance relative to the respective baseline are depicted in the last column of Table 4.10. Such improvement is due to the effect of distillation and co-operative learning from the multi-branch teacher network, which in this case is the super-graph network. It is conclusive that, as sub-graph network gets deeper (or complex) super-graph network eventually gets stronger, which in the later step serves as a strong teacher network.

Comparison with Other Models

For the demonstration of efficacy and comparative study for CMNN on the Tiny-ImageNet dataset we consider several ensemble or multi-model type networks performance in Table 4.11. For the balanced comparison we mostly adhere to the ensemble system with similar ResNet backbones. In addition, to demonstrate the effect of knowledge distillation that is induced through our proposed learning system we also represent performance of the backbone networks that we train with vanilla distillation, where the ResNet-110 is the teacher network.

Snapshot Ensemble [144] which consists of six identical ResNet-110 models during the inference performs with an accuracy of 60.60% (when α_0=0.1) and 59.46% (when α_0=0.2). In contrast, the super-graph and the ensemble of CMNN-110 with only three branches (parameter budget of only 5.10M) achieve 62.10% and 62.87% accuracy respectively. Now, when we consider a single sub-graph from the super-graph of CMNN-110 we achieve an accuracy of 61.80%. Needless to say, the sub-graph is basically equivalent to the ResNet-110 network, which has a parameter count of only 1.70M. Moreover, performance of the sampled sub-graph (of CMNN-110) has a considerable performance improvement relative to both its baseline (around +2.32%, refer to row 15 of Table 4.10 and also Table 4.11). Similarly, sub-graphs of CMNN-20 and CMNN-56 (row 18, 19) have slightly better performance relative to the exact same baseline models trained with (vanilla distillation) and without distillation.

Thus, it is clear from the study with Tiny ImageNet that, i) the distillation effect in CMNN is quite effective and dominant in producing improved sub-graph performance relative to its baseline version; ii) ensemble with fewer sub-graphs (β=3 in this case) is very lightweight yet provides improvement in accuracy.

In this comparative study we have mostly adhered to ensemble or multi-model networks with parameter budget approximately similar to CMNN. However, it is important to note that there are currently several contributions that have achieved remarkable accuracy on the Tiny ImageNet dataset with relatively heavier ImageNet version backbone networks. Such as, the Deep Ensembles (DE) of PreActResNet-18 (width=2) [156] with parameter counts 89.80M and 134.70M achieve accuracy 68.06% and 69.05% respectively. The backbone of this framework is ResNet-18 which is itself sufficiently parameterized relative to the light weight baseline in our experiments (i.e. ResNet-20, 56 and so on). As a result, as the PreActResNet-18 and its DE versions are scaled up there are substantial growth in parameters [156].

4.5.4 On Road Risk Classification

Overview

In this section, we give a brief overview on the ORR classification model. To learn more about this model we recommend referring to our original literature [4].

State-of-the-art object detection models such as YOLO [133] and SSD [13] perform one-stage object detection by predicting the bounding box through regression and classifying the object within the bounding box. This task is performed on the feature maps produced by the CNN backbone and neck networks. Bounding box prediction and classification are not only limited to one feature map, rather multiple feature maps are leveraged for fine-grained object localization of various shapes and sizes. Moreover, these detection networks are composed of multiple blocks of networks, such as backbone, neck, and head networks for dense prediction. For the purpose of ORR dataset, we take a different approach in literature [4] with a view to obtaining high-speed performance. Although *YOLO* [133] provides superior performance, even the lightest version of these networks are sometimes very expensive for us to run on the mounted hardware of the mobility scooter. Moreover, for mobility scooter the risk detection task needed to be performed on a very specific predefined location for which [13, 133] can be redundant and overuse.

Thus, ORR classification model (refer to Figure 4.5) was proposed which concentrates on a very specific predefined Cell of Interest (COI) and classifies any anomaly or risky events only on those pre-defined cells. The core working mechanism of the network is in the systematic segmentation of the frame into certain number of image patches. The input frame is basically segmented into two kinds of sub-region. The first sub-region has six small squares or cells, each of size (68,68), and the second sub-region has 16 large cells, each of size (106,106). Next, each of 22 cells (or COI) is feed to the CNN model to predict (i.e. classification) the class within that cell. The CNN basically performs multi-label classification for the corresponding cell. Thus, the whole task boils down to only concentrating on the individual COI through CNN. This property makes the proposed ORR classification model very suitable for mobile devices. Moreover, any CNN network of preference can be integrated with the system depending on computational availability. Usually, the heavier and complex the network is, the better is the performance of ORR classification model. It is very important to design a backbone network for the block-wise detection network which is lightweight in terms of parameter count and also performs well in terms of accuracy.

CMNN as the backbone of ORR classification model

Implementing CMNN as the backbone of the ORR classification model can assist in measuring its practicality both in terms of speed and accuracy. First, we leverage the ORR dataset to train a 3-branched CMNN-FC network. The ORR dataset is a large dataset, thus we consider the simpler 3-branched CMNN for the experiment (as it is efficient and quick to train). The CMNN-FC version has shown consistent performance for the CIFAR-10 and CIFAR-100 datasets. We can also see similar performance for the ORR in Table 4.7. Next, we sample the best performing sub-graph from the 3-branched CMNN-FC and integrate it with the ORR classification model as the backbone network. In addition, for the comparison, we collect all the benchmark results on the ORR dataset depicted in the paper [4], and we obtain trained ResNet-20, 56, and 110 without CMNN framework. We represent the comparison score in Table 4.8. In Table 4.8 under the source column, 'ORR' depicts the highest accuracy obtained so far in the original literature [4]. The 'W/O CMNN' depicts training original ResNet networks without our learning framework. Lastly, we present the performance of networks when trained with the CMNN learning framework. CMNN based learning framework produces a network which is light weight inherently and performs with better accuracy relative to performance depicted in

original literature [4].

Thus CMNN learning framework can be adopted in the scenario where accuracy and network complexity are very important trade-off. The performance gain relative to the original network (naive training) is not substantial, however, the gain is consistent for the smaller versions of the network. That is, smaller networks are more likely to gain from the CMNN learning framework.

4.6 Limitations and Recommendations

In this section we briefly discuss about limitations of the CMNN, later we provide a few implementation recommendations in-order to minimize and mitigate the effect of these limitations to a certain extent.

The first limitation that we can easily realize is the training cost of CMNN. It is quite clear from the Algorithm 4 that, during the second stage of training, for each epoch two networks undergo training. First the sub-graph training, followed by β branched super-graph fine-tuning. Thus, the training cost is higher, and the reasons are 1) cost per epoch is $(\beta + 1)$ times larger; and 2) number of epochs needed to train the CMNN should be approximately β times larger that that needed for training the backbone network (to provide equal training chances for all the sub-graphs). For instance, a single run experiment (with hyper-parameters mentioned earlier) with three branched CMNN on CIFAR-10/100 datasets takes approximately three days on a single *NVIDIA 2080 RTX GPU*.

Mitigation of the first limitation might require training implementation improvement. Given enough computational budget (specifically more GPU memory) several sub-graphs can be optimized in parallel during the training phase. In addition, the super-graph can also under-go fine-tuning in parallel. This implementation although will require relatively more GPU memory, the training time might get reduced considerately.

The second limitation is related to the sensitivity of CMNN to initial state of the super-graph network. We know that the training procedure of CMNN is sequential (Round-Robin fashion), where the super-graph (teacher) and sub-graph (student) are optimized alternatively. Thus, insufficiently trained (or no pre-training) super-graph can result in either longer convergence time for sub-graph (not guaranteed) or training collapse. This is because, the distillation of knowledge from poorly trained super-graph will not be effective as the sub-graph will repeatedly try to mimic the un-converged output of super-graph network.

Mitigation of the second limitation will require well trained teacher network. Thus, for a stable training and to avoid collapse it is recommended to sufficiently pre-train the super-graph network (refer to the first procedure of the Algorithm 4). This approach will not only provide a well initialized teacher network, but also a good initialization for the sub-graph networks. Besides stability, we can anticipate that leveraging a third teacher network (assuming very accurate) to distill its knowledge to the super-graph can provide better generalization and also good accuracy. We believe this is something that is worth exploring in the future work.

4.7 Summary

In this chapter, we have proposed Coupled Modular Neural Network (CMNN) framework. The framework was built based on the concept of sub-graph and super-graph design pattern. Each of the sub-graph networks is a standalone ResNet neural network coupled with other sub-graphs through a shared common backbone. To effectively leverage this complex super-graph network we propose a Knowledge Distillation based Round-Robin training procedure. In summary, the training procedure first started by warming up the super-graph network through pre-training for certain epochs. This step provided all the sub-graph networks with a good weight

initialization and also a sub-optimal teacher network. Next, the Round-Robin training of the sub-graph started. During the Round-Robin training phase, each of the sub-graphs gets selected to be trained through the distillation, where the super-graph was the teacher network. At the end of sub-graph training, the super-graph gets fine-tuned. The fine-tuning phase produced a slightly different set of weights for super-graph (teacher network) for the next sub-graph training. Our extensive empirical study confirms two things, first, through this super-graph/sub-graph design, we obtain a boosted sub-graph network in terms of accuracy which performs substantially better than its naive baseline version and sometimes outperforming its deeper version. Second, our proposed novel learning algorithm for this framework introduced diversity among the sub-graph networks which has been reflected through the ensemble performance. The coupling among sub-graphs through weight sharing, Round-Robin based training and fine-tuning introduced an additional form of regularization

Super-graph-Sub-graph (or also known as hyper-net, sub-net in several literature) based architectures design are now getting more attention due to several advantages, such as enabling online distillation, limiting search space for finding efficient and strong sub-network, and so on. One of the research direction we would like to take in our future work is that, instead of feeding a random batch of the dataset to the sub-graph networks during the training phase which consists of all classes, we would like to feed the dataset with a subset of classes constructed in a systematic way (Such as our previous work on modular MS-Net [44]). Thus, each sub-graph can gain specialty in certain aspects of the dataset. HydraNet [67] was implemented based on this principle, however, HydraNet employs conditional execution of selective branches. We are curious to explore how each specialized sub-graph can leverage co-distillation to mutually learn from one another and performs on whole during inference.

Chapter 5

A Unified Modular Selective Network Model

The chapter mainly discusses several open issues and supplementary experiments on MS-Net. First, we shed a light on how several variants of MS-Net of Chapters 2, 3, and 4 proposed in this dissertation can be unified to have a self-contained framework. Next, we discuss open issues such as designing expert-aware routers, improving coordination and consistency among experts through distillation, and so on. In the later part, we provide a list of some possible future works on MNN.

5.1 Unified MS-Net

Decoupled MNNs are very feasible for leveraging parallel training and testing as stated in [45] and demonstrated in [130]. But such a system still does not make full and proper utilization of parameters. Such as, features learned by the earlier layers by either router or experts are more or less similar and primary. If these features can be shared among these experts the system can be made much more efficient. Chapter 4 actually leveraged such a system where a common block has been shared by several modules. However, these modules were not explicitly enforced to gain any specialty on a certain subset of data, unlike MS-Net and O-Ms-Net. A promising research direction would be to unify MS-Net and its variant from all previous chapters to produce a coupled version of MS-Net, where a router of MS-Net can be fused into CMNN to select relevant features maps (instead of relevant expert networks). Instead of boosting each sub-graph network of CMNN in all classes, each sub-graph can be encouraged to specialize in subset data (confusable set of data). It would be interesting to experiment on how different expert branches of CMNN leverage the Online-distillation to cooperatively learn and perform on whole.

Such unified framework would basically be a ***Coupled Modular Selective Network or CMS-Net*** with following properties:

- O-MS-Net with shared backbone network, i.e. router and all the experts will share a common backbone like CMNN. Each branch of CMS-Net will be a specialist in a certain confusable subset of classes and generalist on rest. This although looks intuitively promising, several issues should be considered. Such as proper implementation of an objective function, learning dynamics among the branches, and so on.

- Instead of aggregating the feature vectors from all the sub-graphs (like the CMNN), aggregation can be performed only on selective feature vectors through a router network. Thus, the router network should learn to select relevant features from several branches of the network for the input data (or datum).

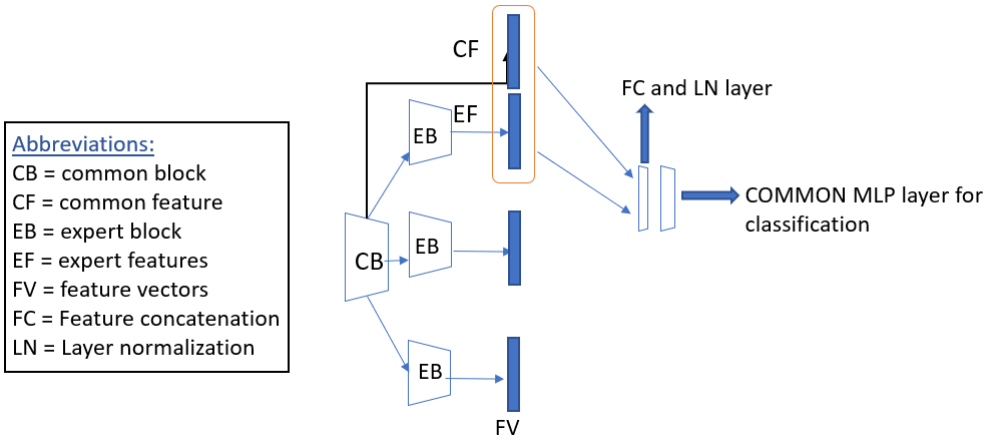


Figure 5.1: **An illustration of Unified MS-Net.**

- Such a network can still be end-to-end trainable (as several modules are coupled) and can make efficient use of dataset and parameters.
- Controlled redundancy should exist in such a system. Chapter 2 has already shown how redundancy can significantly improve performance. This property will also increase the chance of the router choosing at least one correct expert branch (or network).
- Besides ResNet series more state-of-the-art networks should be explored and leveraged to test the versatility of Coupled MS-Net.

5.2 Open Issues and Experiments

5.2.1 Expert Aware Router (EAR)

The performance of MS-Net or O-MS-Net is dependent on how well the router can route the input data to the set of trained experts. The upper-bound performance that we can anticipate is the router's top-2 accuracy (in case we choose to evaluate only the top-2). However, if the router makes a mistake in routing the input to its relevant expert networks there is no way of correction. This is why we introduce controlled redundancy in Chapter 2 in MS-Net, where we assume if one wrong expert is chosen, we still have chances to correct the prediction through other redundant expert networks. The choice of experts has been based on the top-n predictions of the router. But the routing task can also be cast as explicitly choosing expert networks. That is, instead of relying on routers top-n prediction we can train the router to directly map to the relevant expert networks. To be more precise, we train a router to score or vote for the most relevant experts to be leveraged (during the inference phase). Thus, instead of having a SoftMax output from the router, we leverage the Sigmoid layer providing multiple outputs (for choosing multiple experts). Thus the proposed router is termed as the **Expert Aware Router**, as in this case router is more aware of the experts rather than classes of dataset.

Model	PSF	top-2	CBO
O-MS-Net-20	72.15	79.13	77.89

Table 5.1: **Performance So Far (PSF), top-2, Can be Obtained (CBO) for CIFAR-100 dataset with O-MS-Net, ResNet-20 backbone**

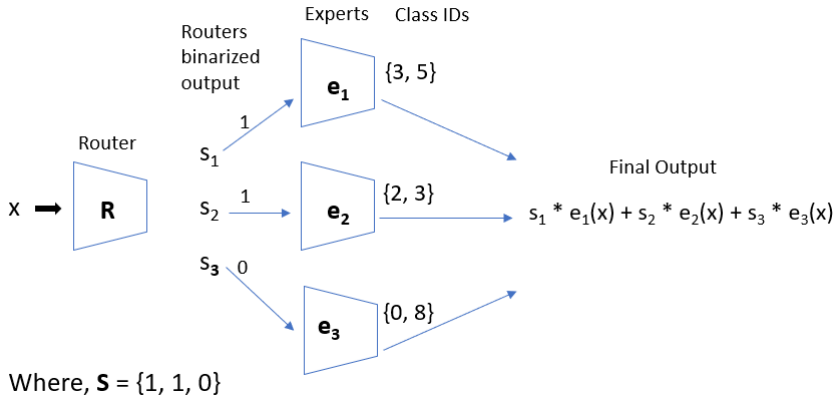


Figure 5.2: **Demonstration of inference phase of EAR. In this figure the input datum x has class ID 3 (i.e the ground truth label). The router is trained to put more confidence on experts that has non-empty intersection with its set of class ID. In this case router output *True* for experts with class ID tags $\{3, 5\}$ and $\{2, 3\}$. The final output is average of selected experts based on the routers output vector S.**

Motivation of EAR

If we look at the empirical results, we will see that gap between the upper-bound performance (i.e. the top-2 performance of router) and MS-Net performance is still substantially big. On the other hand performance of individual expert networks on their corresponding subset is very accurate. Hence, a more reliable router can assist in reducing this gap. Such router should be a **Expert Aware Router**, where router will have prior knowledge on performance of experts. Let us consider a simple case in Table 5.1. Value under **PSF** column is the performance that we have achieved so far by O-MS-Net-20. **top-2** is the upper-bound score that can be obtained at best if and only if the routing and expert accuracy are 100% (which is theoretically impossible). Lastly, **CBO** is the score that can be obtained given that the router accuracy is 100%. The practical goal is to achieve **CBO** upper bound.

Formulation of EAR

We formulate the EAR as the multi-label classification problem. For the original MS-Net and its variant the router $R(.)$ output logit vector $\mathbf{Z} = \{z_1, z_2, .., z_C\}$ is run through Softmax function (refer to equation 3.1) to obtain the probability vector $\mathbf{Q} = \{q_1, q_2, .., q_C\}$. Based on this probability vector we select top-n experts to further re-evaluate the input datum. It is clear that the router in this case is a simple C class classifier, thus it has C output nodes. In the case of EAR we have K output nodes, where K is the total number of experts we have in our disposal. It should be noted that the value of K equals C for MS-Net. For the stable MS-Net (chapter 3) the value of K can vary depending on the difficulty of dataset.

The logit vector from EAR say $\mathbf{Z} = \{z_1, z_2, .., z_K\}$ is run through the Sigmoid function to obtain score vector $\mathbf{S} = \{s_1, ..., s_K\}$. The score vector is later binarized based on certain threshold (we set threshold to 0.5) to select relevant expert. Thus, if output s_i equals to value 1 the i_{th} expert is chosen, where $s_i \in \mathbf{S}$. Several s_i can take the value 1 in which case several experts can be chosen (this is applicable when we have redundant experts). The implementation of router requires the teacher signal to be transformed to one-hot encoded vector. Let us elaborate through a simple example (refer to Figure 5.4). Suppose we have total 3 experts (i.e. value of $K = 3$). Each of the three experts specializes on class $\{3, 5\}$, $\{2, 3\}$ and $\{0, 8\}$ respectively. Now, if an input signal x has ground truth label 3 the transformed ground truth label for the

Table 5.2: **Performance of EAR framework on CIFAR-100.**

Backbone	baseline	EAR prec. (%)	EAR rec. (%)	EAR whole (%)	O-MS-Net (%)
ResNet-8	69.50	78.07	64.00	70.30	72.51
ResNet-20	71.44	81.00	70.00	73.25	75.96

EAR during training would be $\{1, 1, 0\}$, since class ID 3 exist in the $\{3, 5\}$ and $\{2, 3\}$. This is one-hot encoded vector which is basically a mapping from the input to corresponding relevant experts. Hence the task of router boils down to mapping the input to relevant experts from K choices. While we train the EAR on these transformed dataset the expert training are identical to the MS-Net or O-MS-Net.

Experiments with EAR To train and test EAR we leverage our previously pre-trained expert networks of O-MS-Net (refer to the Chapter 3). The backbone network for router and experts for this experiment are ResNet-20 and 110. We train the EAR (only the router) on the one-hot encoded training data for 300 epochs with initial learning rate set to 0.1. We use the Stochastic Gradient Descent (SGD) optimizer which is scheduled to decrease the learning rate by factor 0.1 at steps 80, 160, and 240. We leverage the CIFAR-100 dataset to train and test the EAR framework. A total of 30 experts are integrated into the framework. Table 5.2 represents the brief experiment with EAR.

In Table 5.2 column *baseline* depicts the accuracy of ResNet-20 and 110 on CIFAR-100 test set. The third and fourth columns respectively depict the router's precision and recall scores in selecting the correct experts for the test data. We report the precision and recall score at threshold of 0.5 in this experiment. Encouraging high precision in this case will force router to be very selective in picking at least one right expert. Whereas a high recall value encourages redundancy as router will try to select multiple relevant experts for a single input data. Now as we perform inference with EAR along with all the experts, we achieve classification score of 70.30 % and 73.25% for ResNet-20 and 110 backbones respectively on the CIFAR-100 dataset. EAR framework achieves a considerable positive delta compared to the baseline scores.

However, the performance of EAR framework is still fairly lower than the O-MS-Net (consisting of same number of experts and parameters). The performance on EAR framework is mainly dependent on the accuracy of the router network in selecting the relevant expert. From Table 5.2 it is clear that the performance of router is not satisfactory. Thus router performance is the bottleneck in the EAR framework. Apart from the ResNet series network, we have also explored several multi-layer perceptron (MLP) networks backbone for the router. There is still a performance gaps compared to the original MS-Net.

5.2.2 Conflict and Co-ordination of Expert Modules

The router of the original MS-Net selects n expert networks for each datum during the inference phase based on its top-n predictions. It is true that inference through several expert neural networks based on top-n probable predictions encourages fault tolerance. However, a major limitation of having several experts is the conflict and competition among the networks. The effect of this conflict is not empirically significant but it holds back the network from achieving the anticipated performance. Such as performance close to routers top-n accuracy. We know that each of the experts is biased towards its own subset class (because we sample more training data from the subset classes during the training phase). In other words, the experts compete against each other by putting more softmax confidence on its own designated classes. We are currently exploring two key ideas to resolve this conflicting behaviour of experts. The first approach is the simplest one which we term *One Class MS-Net*. It is a data-centric approach. The

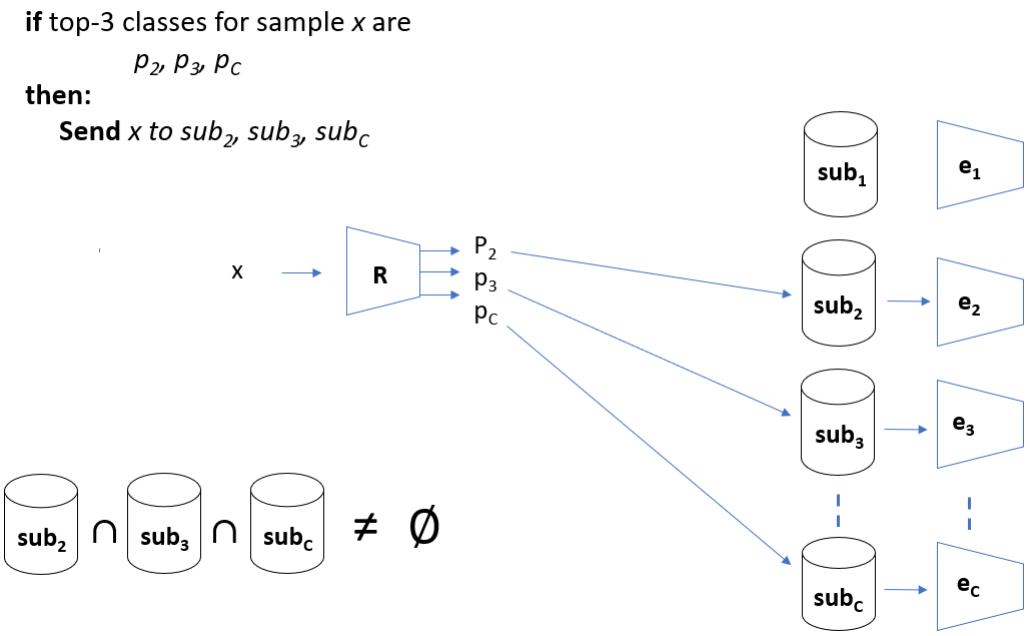


Figure 5.3: **Dataloader construction for the One-Class-MS-Net. Data x is fed to router which is assigned certain confidence by the router network. The data x is afterward assigned to n dataloaders based on top-n prediction by router.**

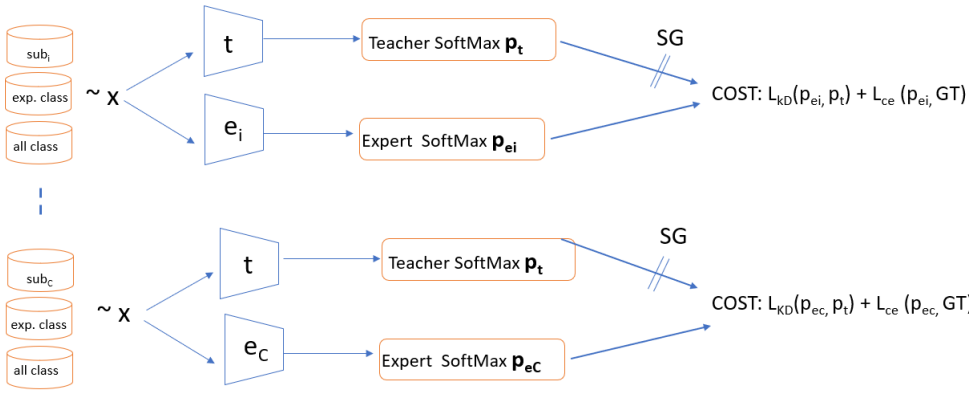


Figure 5.4: **Distillation based training of expert neural networks. The original MS-Net (chapter 2) and O-MS-Net (chapter 3) trains each experts in stochastic matter on samples drawn from the subset classes (or known as the expert class indexes) and all classes. In this approach samples are drawn from three dataloaders, where dataloader sub_i encourages co-ordinance among experts.**

Table 5.3: **Performance of One-Class MS-Net with varying backbone and top-n.**

Backbone	1-C-MS-Net-2 (%)	1-C-MS-Net-3 (%)	O-MS-Net-2 (%)	O-MS-Net-3 (%)
ResNet-8	93.04	93.37	92.20	93.61
ResNet-20	95.50	96.70	95.90	96.45

second approach is distillation-based approach where we introduce a novel dynamic learning system known as the Selective Expert Ensemble Distillation or in short SEED. In the following sub-section, we will briefly explain them.

Data-Centric Approach

The key idea is to train a set of experts on common subset of training data such that these set of experts have consistent agreement during the test phase. We prepare these subset of training data by leveraging the router network. The idea is very similar to ICC based data partition where we first take the routers top-n prediction labels for an input datum. For the ease of understanding let us assume that the top-n predictions are p_1, p_2 and p_3. Next, we make n copies of the input datum and assign each of the copy to n dataloaders respectively. We denote the dataloaders as $S = \{sub_1, sub_2, .., sub_3\}$, where each sub_i consists of samples designated to the corresponding expert e_i.

After we construct subset of samples (it should be noted that each subset $s_i \in S$ in this experiment is composed of training samples not classes indexes) we train the experts on training data sampled uniformly from these three dataloaders, i.e. sub_{all}, sub_i, sub_{exp}. As a result of these training method experts are more coherent with each other that is reflected in the brief experiments in Table 5.3. Previously MS-Net struggled with conflicting experts as we increased the number of expert evaluation. In this proposed approach the conflicting behaviour is moderately resolved, as we can see the 1-Class-MS-Net (with 3 experts per sample) almost matches performance with O-MS-Net (with 1 expert per sample). In addition, increasing top-n also exhibit improvement in performance.

Although the current data centric approach resolved conflict among experts, this is still a open research where we can improve the coordination among experts to extract more performance from the MS-Net framework. The efficacy and versatility of this data-centric approach for modular neural networks is yet to be explored in detail. We are currently running more experiments.

Distillation Approach

In our studies so far, distillation was one of the key strategies in training modular neural networks. Indeed, we have proposed few variants of distillation-based cost function to train and regularize the expert networks. Let us briefly summarize the distillation-based approaches before we introduce the new idea.

- In chapter 2 and 3 for the expert training we leverage a static router network as the teacher network. The router is relatively weak as it shares the same architecture as the expert network. However, the main task of distillation is to preserve the routers generalized knowledge in the experts besides gaining expertise on its subset data. Because of the distillation term in loss function the experts can avoid over-fitting on the subset data.

- In chapter 4 we introduce a different form of distillation, where the teacher network is not static. During each iteration the teacher network weight is perturbed slightly. Thus, the

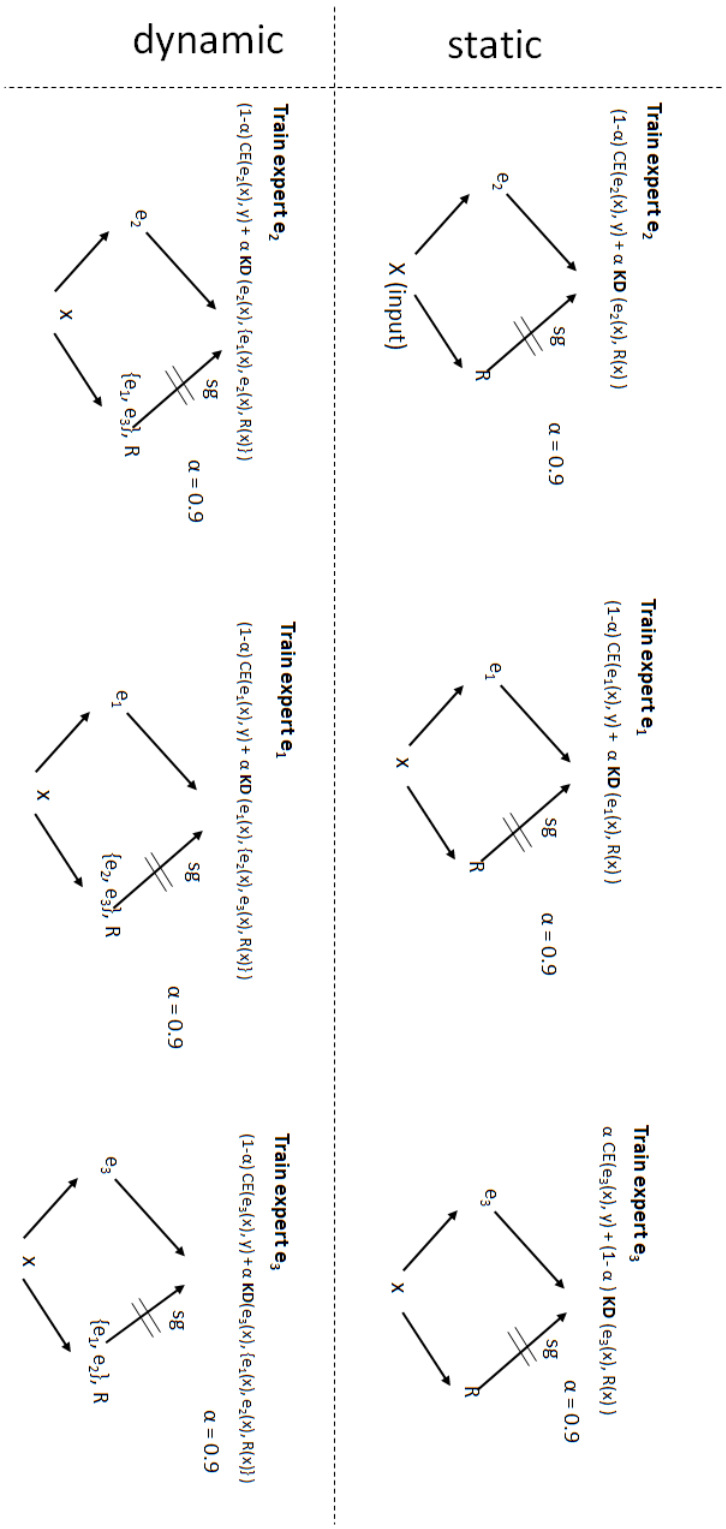


Figure 5.5: **Training demonstration of expert networks with static vs. dynamic teachers. Here teacher network is the ensemble of several experts. Ensemble of experts (teacher) can change depending on which expert (student) we are training. In this figure x is the input and y is the ground-truth label, sg is the stop gradient operation.**

teacher network is slightly different at each iteration, hoping that it encourages diversity during training.

Now, one of the key limitations of chapter 2 and 3 distillation-based approach is that the teacher network is static and distillation only occurs from the static teacher to experts. There are no *expert to expert* communication during training, or in order words distillation from expert to expert. A simple question that may arise, *why do experts need co-distillation among themselves?* The answer is, during the inference phase the aggregated SoftMax confidence by several experts are not calibrated. That is, the output by the individual experts can compete. Moreover, experts are not aware of each other's confidence. Thus, we propose a new variant of co-distillation loss that encourages consistency among experts. In the following section we provide a detailed overview.

Selective Expert Ensemble Distillation

The key idea is, instead of leveraging the output of static teacher we leverage the ensemble of experts and router (aggregated softmax) as the teacher signal. The experts that participate in the distillation term depend on the current expert we are training. For clear understanding let us refer to a visual example. In figure 5.5 the row *static* depicts the distillation approach adopted in chapter 2 and 3. In case of static teacher, the teacher is the router network (R). In dynamic case, we leverage ensemble of experts together with the router network. For instance, in figure 5.5 based on the ICC we assume that experts e_1, e_2 and e_3 co-occur together frequently. Thus, experts e_1, e_2 and e_3 participate together during the training phase. When we are training expert e_2 we leverage ensemble of e_3, e_1 and R as the teacher signal. The error signal during training is back-propagated through e_3 only, while rest part of the graph is detached from back-propagation calculation. The rest of experts e_1 and e_2 are trained in similar way in the next iteration.

In figure 5.6 we provide a complete framework of the proposed selective ensemble distillation. In the first step, we train expert e_i through distillation from the aggregated output of selected ensemble of experts (by the MS-Net router). In the second step, once training of e_i is completed the trained expert is pushed back to the expert pool (which also acts as one of the teacher network in certain iteration), and next expert e_{i+1} in the queue is pop out. In third step, expert e_{i+1} is trained in similar fashion as step 1. These three steps should be performed in Round-Robin fashion until all the C experts are sufficiently trained or converged. Once we complete training all the experts in queue we will obtain C trained weights. The rest of procedure (inference phase) is identical to MS-Net.

An important question is, *what do we except from experts trained in such fashion?* In short, the proposed distillation based training encourages the SoftMax confidence or the output distribution of frequently participating experts to be coherent and stable. The outcome of this approach is actually implicitly similar to the data-centric approach. *In data-centric approach frequently co-occurring experts observe or pay attention to similar data to achieve consistent output. In case of distillation based approach frequently co-occurring experts shares the SoftMax distribution through co-distillation to achieve consistent output.*

Experiments with SEED

Table 5.4 represent brief experiments with the proposed selective expert ensemble distillation. We would like to state that the results reported in table 5.4 are not performed through carefully selected (or grid searched) hyper-parameters. Moreover, we adhere to single run for quick experiments. We train SEED using the 1-Class-MS-Net strategy, thus we have C experts in total for C classes. For demonstrating the effect of teacher network we depict the performance of O-MS-Net with similar backbone networks and same number of expert networks. SEED-1-Class model performs considerately better than O-MS-Net for both top-2 and top-3 based router

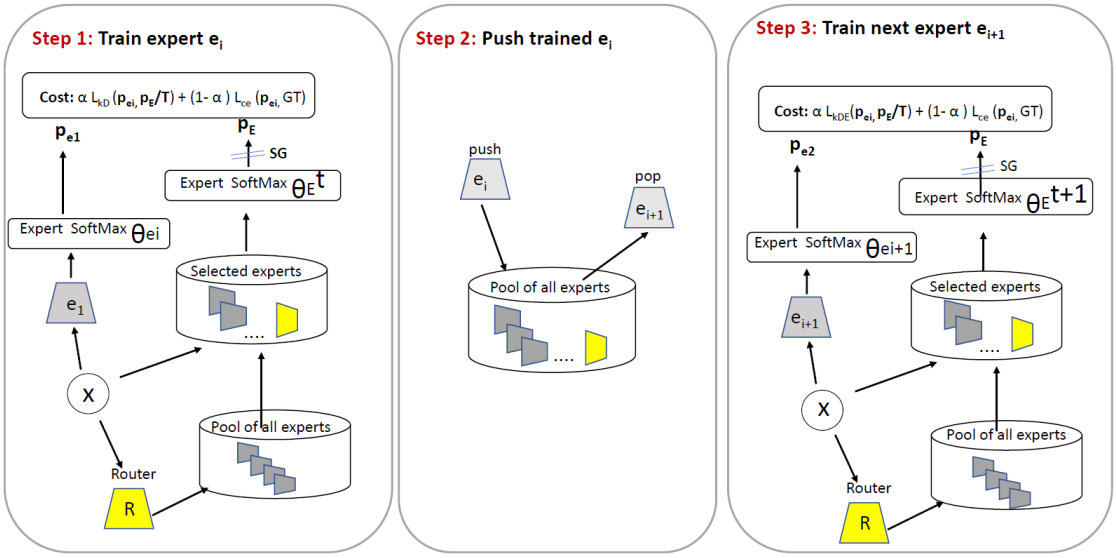


Figure 5.6: **Selective Expert Ensemble Distillation. In step 1, train current experts through distillation from the ensemble of experts (selected through the router). In step 2, push back the trained expert and pop out a new expert. In step 3, train the pop out expert. We perform step 1 to step 3 in iterative fashion until we are satisfied with the collective performance of the experts.**

evaluation. SEED-1-Class requires leveraging more experts than O-MS-net. Thus, we can anticipate slightly better results. However, leveraging several experts can introduce conflict, thus hurting performance. The ensemble distillation assists in resolving this conflict. We can observe that increasing top-n evaluation now consistently improve performance.

5.3 Future Works

Modular Neural Networks are now getting high preference in neural computation as it opens up the possibility to substantially increase the capacity of neural networks without any explosion in parameters during the inference phase. The inherent modularity gives freedom to explore a range of efficient design possibilities and use cases. Such as, efficient design of router and experts' architecture; learning routing policy to smartly co-ordinate experts; incremental learning or life-long learning; multi-modal modular architecture; efficient implementation on hardware

Table 5.4: **Performance of Selective Expert Ensemble Distillation (SEED) framework (refer to figure 5.6) on CIFAR-10 dataset (with ResNet-8 as backbone). The meaning of SEED-1-Class is Selective Expert Ensemble Based Distillation with 1-Class MS-Net variant. 1-Class depicts that each expert in the framework is expert on one class only, and generalist on rest of classes. Hence, total C experts in the framework.**

Model	teacher	top-n	acc. (%)
SEED-1-Class	selective ensemble	2	93.50
SEED-1-Class	selective ensemble	3	94.54
O-MS-Net	static router	2	92.20
O-MS-Net	static router	3	93.61

and so on. In the following bullet points we list some future works that we plan to explore (or anticipate future researchers to consider) as follows:

- The architecture of experts and routers leveraged in our studies are not necessarily the most appropriate one. Both router and experts share the same architecture in our study. However, constructing architecture of experts based on its designated dataset (subset of expert data) will be more appropriate. Such architecture can be designed either through leveraging human heuristics and prior knowledge or through automated search (such as, evolutionary search, random search, R-4 rule and so on). Searching for appropriate architecture can further simplify expert networks for easy tasks and complicated (deeper) architecture for difficult tasks. Optimization of router network is also important.

- We leverage routers SoftMax confidence to route the input datum to the pre-trained experts. However, the choice of routing policy is still open research and several routing policies can be adopted. Such as, learning routing task end-to-end through back-propagation [130]; casting the task of routing as a reinforcement learning problem [157] (the reinforce algorithm optimizes to learn policy that encourages only sparse activation of few units in whole network); learning routing task and expert through optimizing maximum log likelihood function [158] and so on. These strategies are yet to be explored in the context of designing Modular Selective Network.

- As we know experts of MS-Net (of chapter 2 and 3) are independent, which implies that these networks can be trained and tested in parallel. Training of MS-Net in multiple GPU in parallel is an interesting implementation challenge that we look forward to exploring. In addition, optimization of sub-graph networks in parallel can also be achieved for Chapter 3. The learning dynamics of several sub-graphs together with super-graph in parallel is a research direction worth exploring.

Chapter 6

Conclusion

So far, we have seen that modular design for Deep Neural Networks (DNNs) is a promising direction to achieve an accurate and efficient image classification system. Such as, making neural networks modular requires very few parameters to learn complex tasks as opposed to single complex networks. Moreover, it is computationally expensive to train a deep and complex neural network in a single GPU machine. Loading a large DNN in a single lab-level GPU (say NVIDIA GTX 1080) indeed sometimes runs out of memory (error such as *RuntimeError: CUDA out of memory.*). On the other hand, when we have a modular neural network, training each module sequentially in a single GPU is easily manageable. This is because each of the modules is a lightweight DNN with substantially low parameters (and layers). It is true that as the number of modules in the network increases training time increases proportionally. But during test time the number of modules leveraged and the total number of parameter costs per sample are still substantially lower than deeper and complex networks.

In this dissertation, we have proposed a novel modular neural network architecture termed Modular Selective Neural or MS-Net. Through several theoretical analyses and empirical studies with MS-Net and its subsequent variants, we have shown that modular neural networks achieve performance comparable (or sometimes outperform) to the complex monolithic DNNs. In the following paragraphs, we conclude the dissertation in a chapter-wise manner.

In Chapter 2, we have first introduced MS-Net for the visual object recognition task. For a C-class classification problem, the network consisted of one router and exactly C expert networks. The key idea of the research has been to further re-evaluate the top-n most probable predictions of the router by leveraging these expert neural networks. A novel systematic Round-Robin-based data partitioning technique was introduced for MS-Net, where we could explicitly control the redundancy of classes occurring in the constructed subset of the dataset. This property has given the overall network with boosted performance with no significant parameter cost. The research has also proposed a stochastic objective function for the expert networks which has been equipped with the Knowledge-Distillation (KD) term that facilitated alternative training on subset data and the whole set of data. This alternative switching during the training phase has been regulated by clamping a Bernoulli Random variable to each loss function term. Optimizing this loss function has resulted in a well generalized expert network. In summary, we have shown that, with a very limited parameter budget and simple Deep Neural Network (DNN) backbone, our network has achieved performance comparable or sometimes equivalent to more complex monolithic DNNs.

Chapter 3 is the successor of the originally proposed MS-Net where we have optimized the network by constructing expert neural networks for only a set of classes that are often confused or mistaken by the router network. These confusable set of classes have been obtained by leveraging the SoftMax information of the router, whose top-n probability distribution exhibit a certain degree of visual similarity. An interesting observation is that these confusable sets of classes are consistent irrespective of the architecture of the router networks. During the infer-

ence phase, experts were leveraged when routers top-n predictions have non-zero overlapping with confusing classes. Thus, only relevant experts were utilized during inference. This approach has substantially reduced per sample expert usage, yet performing with neck-and-neck accuracy with original MS-Net. O-MS-Net has not leveraged any redundancy during inference as the network directly addresses the confusable set of classes. However, the study did not advocate getting rid of redundancy. In fact, redundancy is very important to build a more reliable and error-tolerant neural network system. In chapter 5 we actually discussed how well-regularized redundancy can make such a network more accurate.

Lastly, in Chapter 4, we demonstrated that a modular neural network is best utilized when coupled through a shared backbone. This allowed efficient use of datasets and parameters. Thus a *generalized and efficient version of MS-Net* has been proposed. The generalized version encouraged parameter sharing among several modules through a shared common backbone, hence the network has been termed Coupled Modular Neural Network (CMNN). The framework has been built based on the concept of super-graph and sub-graph design, where each of the sub-graphs is a standalone neural network coupled with other sub-graph networks within the framework. All the sub-graphs together form a complex and wide super-graph network. To effectively leverage this complex super-graph network an online knowledge-distillation-based Round-Robin training procedure was introduced. In conclusion, the training procedure first started by warming up the super-graph network through pre-training for certain epochs. This step ensured that all the sub-graph networks' weights are properly initialized. The next stage is the Round-Robin training of the sub-graphs. During the Round-Robin training phase, each of the sub-graphs gets selected to be trained through the co-distillation, where the super-graph was the teacher network. At the end of sub-graph training, the super-graph gets fine-tuned. The fine-tuning phase produced a slightly different set of weights (different modes in parameter space) for the super-graph (teacher network) for the next sub-graph training. Extensive empirical studies confirm two things, first, through this super-graph/sub-graph design, we obtained sub-graph networks which performed substantially better than its naive baseline version and sometimes outperformed its deeper version. Second, the proposed novel learning algorithm introduced diversity among the sub-graph networks which produced strong ensemble performance.

A big picture of our contribution has been depicted in Figure 6.1. The Figure represents important common properties and contrasts among the versions of MS-Net of the aforementioned Chapters.

Chapter and Model	Router	Expert module	Aggregation method
Chapter 2: MS-Net	Has a router that selects ρ*n experts per sample, where the product of ρ and n is smaller than C	Total C experts are designed with each experts specializing on certain subset of classes. Framework requires no more than C experts (refer to Lemma 1 of Chapter 2)	Performed averaging of the selected experts softmax prediction and finally weighted by the routers confidence.
Chapter 3: O-MS-Net	Has a router that selects only one expert per sample.	Total K experts. As K increased more samples have been corrected from confusable set of classes. No theoretical upper or lower bound for K. Usually the more the better with increased training cost.	Computed the final softmax prediction of expert network weighted by the router softmax confidence.
Chapter 4: CMNN	Instead of having a decoupled router and experts, all the modules share the same "feature map" generated by the shared common backbone (here shared common block is analogous to router of original MS-Net). The shared common block does not perform selective operation for CMNN, rather prevented re-calculation of intermediate feature map every time for all the modules.	Total β modules or sub-graphs, where each modules are stand-alone neural networks performing considerably well on all classes. The framework does not explicitly enforce specialization for each modules. Rather, all modules are trained on all classes through online distillation in Round-Robin fashion. A single module can be leveraged for performing classification on all task, or all modules can be collectively leveraged to boost accuracy further.	Several aggregation methods can be leveraged in this generalized version of MS-Net. Such as, feature concatenation, feature averaging or softmax averaging from the obtained output of sub-graph (modules). Aggregated features are later feed to linear layer for performing prediction. Naïve ensemble can also be performed where individual prediction of modules are averaged.
Chapter 5: C-MS-Net: Unified MS-Net (Open issue and Future work)	Will have a router, that will select feature-map from the relevant branch or sub-graph. Number of sub-graph to be selected for an input will depend on several factors such as, redundancy rate, total number of sub-graphs (or confusable subsets) and so on.	The network will have a shared common block like CMNN, with several **EXPERT** sub-graphs specializing on certain subset of dataset (subsets can be obtained through proposed ICC method or RR based method). Router will be trained to pick the suitable sub-graph to further calculate feature-map for the input data.	Selected expert sub-graphs will produce feature-maps which can be aggregated like procedure introduced in Chapter 4.

Figure 6.1: **Big picture of the properties of MS-Nets and its successors.**

References

[1] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in Neural Information Processing Systems*, 2019, pp. 103–112.

[2] G. Auda and M. Kamel, "Modular neural networks: a survey," *International Journal of Neural Systems*, vol. 9, no. 02, pp. 129–151, 1999.

[3] H. Wang, K. Su, I. M. Chowdhunry, Q. Zhao, and Y. Tomioka, "Comparison between block-wise detection and a modular selective approach," in *2020 11th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2020, pp. 1–5.

[4] K. Su, C. M. Intisar, Q. Zhao, and Y. Tomioka, "Knowledge distillation for real-time on-road risk detection," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2020, pp. 110–117.

[5] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[6] R. Rojas, "The backpropagation algorithm," in *Neural networks*. Springer, 1996, pp. 149–182.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[8] A. El-Sawy, E.-B. Hazem, and M. Loey, "Cnn for handwritten arabic digits recognition based on lenet-5," in *International conference on advanced intelligent systems and informatics*. Springer, 2016, pp. 566–575.

[9] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[12] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[15] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[17] H. Wu, J. Zhang, K. Huang, K. Liang, and Y. Yu, "Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation," *arXiv preprint arXiv:1903.11816*, 2019.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[19] X. Wang and A. Gupta, "Generative image modeling using style and structure adversarial networks," in *European conference on computer vision*. Springer, 2016, pp. 318–335.

[20] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[21] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[22] J. Kim, M. Kim, H. Kang, and K. Lee, "U-gat-it: unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation," *arXiv preprint arXiv:1907.10830*, 2019.

[23] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[24] A. Dutt, D. Pellerin, and G. Quénot, "Coupled ensembles of neural networks," *Neurocomputing*, 2019.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[26] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[29] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Deep metric learning for person re-identification," in *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, pp. 34–39.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[32] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[33] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[34] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen, "A survey on evolutionary neural architecture search," *arXiv preprint arXiv:2008.10937*, 2020.

[35] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[36] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[39] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[40] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," *arXiv preprint arXiv:1912.11370*, 2019.

[41] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[43] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[44] I. M. Chowdhury, K. Su, and Q. Zhao, "Ms-net: modular selective network," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 3, pp. 763–781, 2021.

[45] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[46] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[47] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton *et al.*, "Adaptive mixtures of local experts." *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[48] A. Makkuva, P. Viswanath, S. Kannan, and S. Oh, "Breaking the gridlock in mixture-of-experts: Consistent and efficient algorithms," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4304–4313.

[49] B. Illing, W. Gerstner, and J. Brea, "Biologically plausible deep learning—but how far can we go with shallow networks?" *Neural Networks*, vol. 118, pp. 90–101, 2019.

[50] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.

[51] A. Kelkar and J. D. Medaglia, *Evidence of Brain Modularity*. Cham: Springer International Publishing, 2018, pp. 1–10. [Online]. Available: https://doi.org/10.1007/978-3-319-16999-6_2422-1

[52] S. Zeki and S. Shipp, "The functional logic of cortical connections," *Nature*, vol. 335, no. 6188, pp. 311–317, 1988.

[53] T. Hrycej, *Modular learning in neural networks: a modularized approach to neural network classification*. John Wiley & Sons, Inc., 1992.

[54] R. A. Jacobs, M. I. Jordan, and A. G. Barto, "Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks," *Cognitive science*, vol. 15, no. 2, pp. 219–250, 1991.

[55] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, "Medical image classification with convolutional neural network," in *2014 13th international conference on control automation robotics & vision (ICARCV)*. IEEE, 2014, pp. 844–848.

[56] J. Zhao, B. Liang, and Q. Chen, "The key technology toward the self-driving car," *International Journal of Intelligent Unmanned Systems*, 2018.

[57] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[58] X. Zhu, S. Gong *et al.*, "Knowledge distillation by on-the-fly native ensemble," in *Advances in neural information processing systems*, 2018, pp. 7517–7527.

[59] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4320–4328.

[60] S.-H. Park and J. Fürnkranz, "Efficient prediction algorithms for binary decomposition techniques," *Data Mining and Knowledge Discovery*, vol. 24, no. 1, pp. 40–77, 2012.

[61] J. Fürnkranz, "Round robin ensembles," *Intelligent Data Analysis*, vol. 7, no. 5, pp. 385–403, 2003.

[62] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka, "Efficient classification for multiclass problems using modular neural networks," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 117–124, 1995.

[63] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[64] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural computation*, vol. 12, no. 8, pp. 1869–1887, 2000.

[65] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 18.

[66] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: many could be better than all," *Artificial intelligence*, vol. 137, no. 1-2, pp. 239–263, 2002.

[67] R. Teja Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8080–8089.

[68] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[69] P. Clark and R. Boswell, "Rule induction with cn2: Some recent improvements," in *European Working Session on Learning*. Springer, 1991, pp. 151–163.

[70] J. Fürnkranz, "Separate-and-conquer rule learning," *Artificial Intelligence Review*, vol. 13, no. 1, pp. 3–54, 1999.

[71] ——, "Round robin classification," *Journal of Machine Learning Research*, vol. 2, no. Mar, pp. 721–747, 2002.

[72] ——, "Pairwise classification as an ensemble technique," in *European Conference on Machine Learning*. Springer, 2002, pp. 97–110.

[73] J. B. Hampshire II and A. Waibel, "The meta-pi network: Building distributed knowledge representations for robust multisource pattern recognition," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 7, pp. 751–769, 1992.

[74] H. Lee and J.-S. Lee, "Local critic training for model-parallel learning of deep neural networks." *arXiv preprint arXiv:1805.01128*, 2018.

[75] S. Gross, M. Ranzato, and A. Szlam, "Hard mixtures of experts for large scale weakly supervised vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6865–6873.

[76] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[77] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[78] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[79] R. Socher, Y. Bengio, and C. Manning, "Deep learning for nlp," *Tutorial at Association of Computational Logistics (ACL)*, 2012.

[80] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[81] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[82] K. Warburton, "Deep learning and education for sustainability," *International Journal of Sustainability in Higher Education*, vol. 4, no. 1, pp. 44–56, 2003.

[83] C. M. Intisar and Y. Watanobe, "Classification of online judge programmers based on rule extraction from self organizing feature map," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2018, pp. 313–318.

[84] C. Watanabe, K. Hiramatsu, and K. Kashino, "Modular representation of layered neural networks," *Neural Networks*, vol. 97, pp. 62–73, 2018.

[85] A. K. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.

[86] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face recognition with local binary patterns," in *European conference on computer vision*. Springer, 2004, pp. 469–481.

[87] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features." in *iccv*, vol. 99, no. 2, 1999, pp. 1150–1157.

[88] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[89] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical data augmentation with no separate search," *arXiv preprint arXiv:1909.13719*, 2019.

[90] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013.

[91] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[92] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[93] Q. Zhao and T. Higuchi, "Evolutionary learning of nearest-neighbor mlp," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 762–767, 1996.

[94] Q. Zhao, "Stable online evolutionary learning of nn-mlp," *IEEE transactions on neural networks*, vol. 8, no. 6, pp. 1371–1378, 1997.

[95] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[96] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[97] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[98] M. Minsky, *Society of mind*. Simon and Schuster, 1988.

[99] H. C. Leung and V. W. Zue, "Applications of error back-propagation to phonetic classification," in *Advances in neural information processing systems*, 1989, pp. 206–214.

[100] A. Waibel, H. Sawai, and K. Shikano, "Modularity and scaling in large phonemic neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, pp. 1888–1898, 1989.

[101] C. M. Intisar and Q. Zhao, "A selective modular neural network framework," in *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2019, pp. 1–6.

[102] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar, "Born again neural networks," *arXiv preprint arXiv:1805.04770*, 2018.

[103] C. Yang, L. Xie, S. Qiao, and A. Yuille, "Knowledge distillation in generations: More tolerant teachers educate better students," *arXiv preprint arXiv:1805.05551*, 2018.

[104] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1857–1865.

[105] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.

[106] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[107] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[108] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[109] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[110] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[111] [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[112] N. Ketkar, "Introduction to pytorch," in *Deep learning with python*. Springer, 2017, pp. 195–208.

[113] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 113–123.

[114] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "Xnas: Neural architecture search with expert advice," in *Advances in Neural Information Processing Systems*, 2019, pp. 1977–1987.

[115] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," in *Advances in Neural Information Processing Systems*, 2018, pp. 8356–8365.

[116] P. Savarese and M. Maire, "Learning implicitly recurrent cnns through parameter sharing," *arXiv preprint arXiv:1902.09701*, 2019.

[117] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[118] S. Gao, M.-M. Cheng, K. Zhao, X.-Y. Zhang, M.-H. Yang, and P. H. Torr, "Res2net: A new multi-scale backbone architecture," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[119] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast autoaugment," *arXiv preprint arXiv:1905.00397*, 2019.

[120] P. H. Savarese, L. O. Mazza, and D. R. Figueiredo, "Learning identity mappings with residual gates," *arXiv preprint arXiv:1611.01260*, 2016.

[121] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[122] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[123] Zalandoresearch, "zalandoresearch/fashion-mnist," Aug 2019. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

[124] ——, "zalandoresearch/fashion-mnist," Aug 2019. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

[125] Y.-C. Ho and D. L. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *Journal of optimization theory and applications*, vol. 115, no. 3, pp. 549–570, 2002.

[126] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999.

[127] I. M. Chowdhury, K. Su, and Q. Zhao, "Ms-net: modular selective network," *International Journal of Machine Learning and Cybernetics*, pp. 1–19, 2020.

[128] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *International Conference on Machine Learning*. PMLR, 2017, pp. 527–536.

[129] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," *arXiv preprint arXiv:1804.03235*, 2018.

[130] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[131] [Online]. Available: ufldl.stanford.edu/housenumbers.

[132] "Pytorch," https://pytorch.org/.

[133] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[134] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 727–10 737.

[135] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[136] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[137] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[138] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[139] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," 2020.

[140] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Faster discovery of neural architectures by searching for paths in a large model," 2018.

[141] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[142] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[143] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks," *arXiv preprint arXiv:1511.06314*, 2015.

[144] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.

[145] A. Ruiz and J. Verbeek, "Distilled hierarchical neural ensembles with adaptive inference cost," *arXiv preprint arXiv:2003.01474*, 2020.

[146] H. Peng, J. Li, Q. Gong, Y. Ning, S. Wang, and L. He, "Motif-matching based subgraph-level attentional convolutional network for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5387–5394.

[147] E. Alsentzer, S. Finlayson, M. Li, and M. Zitnik, "Subgraph neural networks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 8017–8029. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/5bca8566db79f3788be9efd96c9ed70d-Paper.pdf

[148] Q. Sun, J. Li, H. Peng, J. Wu, Y. Ning, P. S. Yu, and L. He, "Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism," *arXiv preprint arXiv:2101.08170*, 2021.

[149] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4794–4802.

[150] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[151] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[152] Z. Abai and N. Rajmalwar, "Densenet models for tiny imagenet classification," *arXiv preprint arXiv:1904.10429*, 2019.

[153] Github, "Tiny-image-net benchmarks," https://github.com/meet-minimalist/TinyImageNet-Benchmarks.

[154] J. Wu, Q. Zhang, and G. Xu, "Tiny imagenet challenge," 2017.

[155] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, p. 7, 2015.

[156] A. Rame, R. Sun, and M. Cord, "Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks," *arXiv preprint arXiv:2103.06132*, 2021.

[157] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," *arXiv preprint arXiv:1511.06297*, 2015.

[158] L. Kirsch, J. Kunze, and D. Barber, "Modular networks: Learning to decompose neural computation," *arXiv preprint arXiv:1811.05249*, 2018.