CONDITIONAL COMPUTATION TECHNIQUES IN DEEP NEURAL NETWORKS WITH CONDITIONAL INFORMATION GAIN

by

Ufuk Can Bicici

B.S., Computer Engineering, Marmara University, 2009

M.S., Computer Engineering, Boğaziçi University, 2012

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2024

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Prof. Lale Akarun, for her unending support and patience during the preparation of this thesis. Without her unwavering guidance and encouragement, I could not have imagined bringing this thesis to its conclusion. She always believed in this work, sometimes even more than I did, and continually encouraged me with her optimism. I also want to thank the members of my defense committee for their reviews and feedback.

I would like to thank my parents, Atila and Sukran Bicici, for their support throughout this long journey. They were always there to listen to my concerns and worries during difficult times. I also want to thank my other family, Suleyman Buyukeren, Gulseren Buyukeren, and Burcin Buyukeren, for their support.

Many people have offered their support during the years spent preparing the work contained in this thesis. Although the list is not exhaustive, I want to thank Deniz Durmus, Eray Berger, Cem Keskin, Alp Kindiroglu, and Ogulcan Ozdemir for their ideas, input, and emotional support. A special thank you goes to Tuna Han Salih Meral, who contributed significantly to this work with his coding.

Finally, a very special and heartfelt "thank you" goes to my lovely wife, Burcu Buyukeren Bicici. Without her never-ending love and patience, I would never have been able to complete this work. Her optimism and endurance always inspired me to move forward, even in the darkest times.

This work was partially supported by the Turkish Ministry of Development under the TAM Project, number 2007K120610. Some of the experiments in this work have been conducted using Bogazici University TETAM GPUs.

Some figures in this thesis were published in Pattern Recognition, Vol. 120, Bicici, U. C. and L. Akarun, "Conditional Information Gain Networks as Sparse Mixture of Experts", p. 108151 Copyright Elsevier (2021) and Pattern Recognition Letters, Bicici, U. C., T. H. S. Meral and L. Akarun, "Conditional Information Gain Trellis", Copyright Elsevier (2024).

The images that emerged within the scope of this thesis study and whose copyrights were transferred to the publishing houses, were used in the thesis book following the publication policies valid for the re-use of the text and graphics produced by the author, which can be found on the publishing houses' web pages.

ABSTRACT

CONDITIONAL COMPUTATION TECHNIQUES IN DEEP NEURAL NETWORKS WITH CONDITIONAL INFORMATION GAIN

Recently, deep neural networks, particularly convolutional neural networks, have excelled in computer vision tasks such as image classification, object detection, and semantic segmentation. Their high performance stems from numerous layers and learnable parameters. However, this complexity poses challenges for efficient inference, especially on devices with limited computing power, like edge devices. Among numerous similar approaches in the literature to address this issue, conditional computing is an efficient inference method where parts of a deep neural network are used or skipped based on the properties of the input. In this thesis, we develop two main conditional computing approaches: "Conditional Information Gain Networks", where a neural network is designed in the shape of a tree, and the samples are routed based on network elements that are trained with information gain. The second one is the "Conditional Information Gain Trellis", which describes a trellis-shaped network that again allows the routing of the samples based on the decisions of routing units trained by information gain objectives. We develop loss functions, training methodologies, and regularizers for both models. For both models, we develop inference methodologies that allow the routing of samples over more than one route in these networks, where we try to achieve a balance between additional model performance and extra computational burden. These multiple-path routing approaches, which we call "Sparse Mixture of Experts" inference, are implemented using algorithms such as Bayesian Optimization, Cross-Entropy Entropy Search, and Reinforcement Learning. We show the results of these model designs with various experiments.

ÖZET

DERİN SİNİR AĞLARINDA KOŞULLU BİLGİ KAZANIMI İLE KOŞULLU HESAPLAMA YÖNTEMLERİ

Son zamanlarda derin sinir ağları, özellikle evrişimli sinir ağları, görüntü işleme, nesne algılama ve anlamsal bölümleme gibi birçok bilgisayarlı görme probleminde büyük performans elde etmiştir. Bu ağlar, yüksek sayıda katman ve birçok öğrenilebilir parametre kullanarak bu performans seviyesine ulaşabilmektedir. Bu durum, bu ağların uç cihazlar gibi sınırlı bilgi işlem kapasitesine sahip ortamlarda kullanımı gerektiğinde, verimli çıkarım için sorunlar yaratır. Literatürde bu konuyu ele alan çok sayıda benzer yaklaşım arasında koşullu hesaplama, girdinin özelliklerine bağlı olarak derin bir sinir ağının bazı bölümlerinin kullanıldığı veya atlandığı bir verimli çıkarım yöntemidir. Bu tezde iki ana koşullu hesaplama yaklaşımı geliştirdik: Birincisi, bir sinir ağının ağaç şeklinde tasarlandığı ve örneklerin bilgi kazanımıyla eğitilmiş ağ elemanlarına göre yönlendirildiği "Koşullu Bilgi Kazanım Ağları" yöntemidir. İkincisi ise yine bilgi kazanım hedefleri doğrultusunda eğitilmiş yönlendirme birimlerinin kararlarına dayalı olarak örneklerin yönlendirilmesine olanak sağlayan kafes şeklinde bir ağı tanımlayan "Koşullu Bilgi Kazanımı Kafesi" yöntemidir. Her iki model için de kayıp fonksiyonları ve eğitim algoritmaları geliştiriyoruz. Her iki model için de örneklerin bu ağlarda birden fazla rota üzerinden yönlendirilmesine izin veren çıkarım metodolojileri geliştiriyoruz ve burada ek model performansı ile ekstra hesaplama yükü arasında bir denge kurmaya çalışıyoruz. "Uzmanların Seyrek Karışımı" çıkarımı olarak adlandırdığımız bu çok yollu yönlendirme yaklaşımları, Bayesci Optimizasyon, Çapraz Entropi Entropi ile Arama ve Pekiştirmeli Öğrenme gibi algoritmalar kullanılarak uygulanmaktadır. Bu model tasarımlarının sonuçlarını çeşitli deneylerle gösteriyoruz.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i
ABSTRACT iii
ÖZET iv
LIST OF FIGURES viii
LIST OF TABLES xiii
LIST OF SYMBOLS xvi
LIST OF ACRONYMS/ABBREVIATIONS xviii
1. INTRODUCTION 1
2. RELATED WORK 9
2.1. Conditional Computing 10
2.2. Mixture of Experts 15
2.3. Neural Network - Decision Tree Hybrids 18
2.4. Network Pruning 23
2.5. Neural Architecture Search 27
3. CONDITIONAL INFORMATION GAIN NETWORKS 30
3.1. Conditional Information Gain Networks 30
3.2. Architecture 31
3.3. Differentiable Local Information Gain 35
3.4. Load Balancing 39
3.5. Routing Errors - During the Training 40
3.6. Sparse Mixture of Experts with CIGN 42
3.6.1. Motivation 42
3.6.2. Sparse Mixture of Experts Model 43
3.6.3. Bayesian Optimization for Optimal Threshold Learning 44
3.7. Experiments 47
3.7.1. Experiment Design 47
3.7.2. MNIST Experiments 49
3.7.3. Fashion-MNIST Experiments 53

3.7.4. CIFAR-100 Experiments . . . 56
3.7.5. USPS Dataset Experiments . . . 60
3.7.6. Comparison with Pruning Methods . . . 61
3.7.7. Semantic Partitioning of Data . . . 63
3.7.7.1. MNIST . . . 63
3.7.7.2. Fashion-MNIST . . . 65
3.7.7.3. CIFAR-100 . . . 67
3.7.8. Statistical Tests for Experiments . . . 72
4. CONDITIONAL INFORMATION GAIN TRELLIS . . . 74
4.1. Conditional Information Gain Trellis . . . 74
4.2. Architecture . . . 78
4.3. Local Information Gain Losses . . . 80
4.4. The Problem with Stochastic Calculations . . . 82
4.5. Training Algorithms for CIGT . . . 86
4.5.1. Approximate Training . . . 86
4.5.2. The Full Training Using Gumbel-Softmax Relaxation . . . 88
4.5.3. Information Gain Based Routers In Full Training . . . 90
4.6. Sparse Mixture of Experts with CIGT . . . 92
4.6.1. Motivation . . . 92
4.6.2. Cross-Entropy Search . . . 93
4.6.3. Cross-Entropy Search for CIGT Threshold Determination . . . 95
4.7. Experiments . . . 101
4.7.1. Experiment Design . . . 102
4.7.2. MNIST Experiments . . . 102
4.7.3. Fashion-MNIST Experiments . . . 105
4.7.4. CIFAR-10 Experiments . . . 110
4.8. Semantic Partitioning of Data . . . 115
4.8.1. MNIST . . . 116
4.8.2. Fashion-MNIST . . . 119
4.8.3. CIFAR-10 . . . 122
5. REINFORCEMENT LEARNING BASED MULTI-PATH ROUTING . . . 126

5.1. Motivation 126
5.2. Brief Reinforcement Learning Recapture 127
5.3. Interpreting CIGT Routing a Markov Decision Process 129
5.4. Policy Gradients Based Routing 136
5.5. Supervised Q Networks Based Routing 138
5.6. Experiments 140
5.6.1. Experiment Design 140
5.6.2. Fashion-MNIST and CIFAR-10 Experiments 141
5.6.3. Comparison of Cross-Entropy Search and Reinforcement Learning for CIGT Multi-Path Routing 146
5.6.4. Effect of Multi-Path Routing on Model Interpretability 149
6. CONCLUSION 155
REFERENCES 159

LIST OF FIGURES

Figure 2.1. Conditional computing. 10

Figure 2.2. Mixture of experts. 15

Figure 2.3. Neural trees. 18

Figure 2.4. Network pruning. 24

Figure 2.5. Neural architecture search. 27

Figure 3.1. The semantic partition of the data via information gain-based routing system. The samples are from the Fashion-MNIST dataset [29]. Each row of images represents a separate class. 32

Figure 3.2. Illustrated are the routing patterns of various CIFAR-100 test images using our CIGN model. The first row shows the images with labels. The next two rows display routing probabilities at the first and second CIGN levels. The fourth row shows the leaf node posterior probability. The final row presents the root-to-leaf nodes in the CIGN structure for each image. 43

Figure 3.3. LeNet architectures for MNIST. The left side depicts the thin baseline with root-to-leaf networks and IG-based routing, while the right side shows the thick baseline. 50

Figure 3.4. The performance of bayesian optimization in terms of extra average MAC per sample against different λ values (the balance parameter in bayesian optimization) for the MNIST, Fashion-MNIST, and CIFAR-100 datasets. 52

Figure 3.5. The network architectures for the Fashion-MNIST dataset. The left side shows the thin baseline (root-to-leaf network) with IG-based routing, while the thick baseline is on the right. 55

Figure 3.6. We adopt the ResNet-145 architecture as our baseline for the [2,2] CIGN. Each node at the same depth in the CIGN incorporates an identical sub-network. 57

Figure 3.7. The specifications of the ResNet-145 architectures for both the thin and thick baselines employed in our experiments are outlined below. The columns represent the layer group names, feature map sizes, kernel sizes, and the number of channels, with parentheses denoting the bottleneck layers typically utilized in ResNet. 58

Figure 3.8. The arrangement of MNIST test data across the leaf nodes of the CIGN model is illustrated. The blue bars and accompanying percentage values indicate the frequency of the corresponding class in each node. For visual clarity, the histograms are truncated at a 90% threshold. 64

Figure 3.9. The allocation of each MNIST class among four leaf nodes in our CIGN model with information gain routing. 64

Figure 3.10. The distribution of Fashion-MNIST test data across the leaf nodes of the CIGN model is illustrated in the figure. The blue bars and corresponding percentage values depict the frequency of each class in the respective node. For visual simplicity, the histograms are truncated at a 90% threshold. 66

Figure 3.11. The allocation of each Fashion-MNIST class among four leaf nodes in our CIGN model with information gain routing. 66

Figure 3.12. Distribution for the first leaf node of $[2, 2]$ CIFAR100-CIGN. . . . 67

Figure 3.13. Distribution for the second leaf node of $[2, 2]$ CIFAR100-CIGN. . . 68

Figure 3.14. Distribution for the third leaf node of $[2, 2]$ CIFAR100-CIGN. . . . 68

Figure 3.15. Distribution for the fourth leaf node of $[2, 2]$ CIFAR100-CIGN. . . 69

Figure 3.16. The allocation of each CIFAR-100 class among four leaf nodes in our CIGN model with information gain routing. 70

Figure 3.17. Distribution of CIFAR-100 coarse labels on the four leaf nodes. . . 71

Figure 4.1. An example of the proposed CIGT architecture. In our terminology, this corresponds to a CIGT-[1,2,4] structure. The router units H_i direct the samples into the computation units of the following layers, namely $F_{i+1,j}$. 76

Figure 4.2. The effect of the variance maximization on the distribution of routing probabilities. 100

Figure 4.3. Statistics that describe the behavior of the routers in the Fashion-MNIST CIGT model. Each image is generated solely with outputs of a specific computation unit. 109

Figure 4.4. Statistics that describe the behavior of the routers in the CIFAR-10 CIGT model. Each image is generated solely with outputs of a specific computation unit. 114

Figure 4.5. Distribution of MNIST samples into CIGT routing units. 117

Figure 4.6. MNIST test set sample distribution in block 1. 118

Figure 4.7. MNIST test set sample distribution in block 2. 118

Figure 4.8. Distribution of Fashion-MNIST samples into CIGT units. 120

Figure 4.9. Fashion-MNIST test set sample distribution in block 1. 121

Figure 4.10. Fashion-MNIST test set sample distribution in block 2. 121

Figure 4.11. Distribution of CIFAR-10 samples into CIGT routing units. 123

Figure 4.12. CIFAR-10 test set sample distribution in block 1. 124

Figure 4.13. CIFAR-10 test set sample distribution in block 2. 124

Figure 5.1. $a^0 = 0$ is selected. 133

Figure 5.2. $a^0 = 1$ is selected. 134

Figure 5.3. Layer 2 nodes are selected at layer 1 following the MDP action trajectory $a^0 = 1$ and $a^1 = 2$. Nodes in CIGT layer 2 are color-coded: red for nodes selected by route $(0, 0), (1, 0)$, green for nodes selected by route $(0, 0), (1, 1)$, and gradient colors for nodes selected by both routes. 134

Figure 5.4. Routing behavior for 10 CIFAR-10 samples with the least routing entropy. There is one sample per class. The routers make highly confident decisions with low entropies. Visually, each sample can be recognized as a member of the coarse class group in the leaf nodes, the decisions are interpretable. 150

Figure 5.5. Routing behavior for 10 Fashion-MNIST samples with the least routing entropy. There is one sample per class. All router decisions have very low entropy. Each sample is easily distinguished as a member of the coarse class groups; routing decisions are highly interpretable. 151

Figure 5.6. Routing behavior for three random CIFAR-10 samples with high routing entropy. Samples are routed into multiple paths. Model behavior is not easily explainable. 152

Figure 5.7. Routing behavior for three random Fashion-MNIST samples with high routing entropy. Each sample is routed into multiple paths. Routing decisions are not easily explainable. 153

LIST OF TABLES

Table 3.1. MNIST test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers. 51

Table 3.2. Fashion-MNIST test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers. 54

Table 3.3. CIFAR-100 test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers. 59

Table 3.4. USPS test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers. 61

Table 3.5. CIGN vs structured pruning methods comparison (On MNIST and LeNet). Each sample in the CIGNs visits a network equivalent to (*) plus routers. 62

Table 3.6. MNIST - CIGN vs baselines statistical comparison. 72

Table 3.7. Fashion-MNIST - CIGN vs baselines statistical comparison. 72

Table 3.8. CIFAR-100 - CIGN vs baselines statistical comparison. 72

Table 3.9. USPS - CIGN vs baselines statistical comparison. 72

Table 4.1. MNIST Results. Each sample in the CIGTs visits a LeNet (slim) network with router blocks. CIGT-(R*) uses a CIGT-[1,2,4] architecture with random routings. 103

Table 4.2. CIGT vs structured pruning methods comparison (on MNIST and LeNet). 103

Table 4.3. Fashion-MNIST results. Each sample in the CIGTs visits a network equivalent to LeNet (slim) plus router blocks. CIGT-(R*) network uses an equivalent CIGT-[1,2,4] architecture but routings are made randomly. AT means approximate training and FT means full training. 106

Table 4.4. CIFAR-10 results. Each sample in the CIGTs visits a network equivalent to R-110 (slim) plus router blocks. CIGT-(R*) network uses an equivalent CIGT-[1,2,4] architecture but routings are made randomly. AT means approximate training and FT means full training. 111

Table 5.1. CIFAR-10 results with reinforcement learning based multi-path routing models. The extra computation sourced by both routes and policy/Q-learning models is included in the MAC measurement. All CIGT models have the $[1, 2, 4]$ architecture. s-CIGT means sMoE-CIGT. 143

Table 5.2. Fashion-MNIST results with reinforcement learning based multi-path routing models. The extra computation sourced by both routes and policy/Q-learning models is included in the MAC measurement. All CIGT models have the $[1, 2, 4]$ architecture. s-CIGT means sMoE-CIGT. 144

Table 5.3. Significance comparison and p-values of model performances for reinforcement learning (Q-CIGT), cross-entropy search (s-CIGT), and vanilla routing (CIGT) using bootstrap t-test [148, 170]. (© 2024 IEEE). 148

Table 5.4. Average increase in MAC per sample for reinforcement learning (Q-CIGT) and cross-entropy search (s-CIGT) compared to vanilla routing (CIGT) [170]. (© 2024 IEEE). 149

LIST OF SYMBOLS

A^i	Action space of a CIGT in $i.$ layer.
$A_{i,j}$	Raw activation matrix for the routing probabilities, from the CIGT computation unit (i, j) to their child nodes in the next layer.
B_l	Gumbel-Softmax path selection variable for $l.$ CIGT layer.
D_{KL}	Kullback-Leibler Divergence.
$\mathbb{E}$	Expected value.
$E_{n,i}$	The routing probability entropy for the decision node n and sample i.
$F_{i,j}$	j.th computation unit in i.th CIGT layer.
G	A Gumbel-Softmax variable.
$\mathbb{H}$	Entropy of a probability distribution.
H_i	Routing unit in i.th CIGT layer.
J_C	Classification loss.
J_{IG}	Routing loss.
K_i	Total number of computation units in i.th CIGT layer.
L_{class}	Classification loss in Sparse Mixture of Experts.
L_{MAC}	Multiplication and Accumulation loss in Sparse Mixture of Experts.
L_{opt}	Total Sparse Mixture of Experts total loss.
$MAC(x)$	Multiplication and Accumulation count for path x.
n_i	Path selection variable for CIGNs.
P^i	Policy network or Q-Network for the $i.$ CIGT layer.
$s(.)$	Cross-Entropy Search score function.
Var_n	The variance of the $E_{n,i}$ values for the decision node n.
W_F	Learnable parameters of the CIGN network units.
W_H	Learnable parameters of the CIGN router units.
Z_l	One-hot routing vector for l.th CIGT layer.
Z	One-hot routing matrix for the whole CIGT.

$\hat{Z}_l$	Gumbel-Softmax routing vector for l.th CIGT layer.
$\hat{Z}$	Gumbel-Softmax routing matrix for the whole CIGT.
λ_F	Weight decay parameter for network units.
λ_H	Weight decay parameter for routing units.
γ_t	Cross-Entropy Search score threshold for t.th iteration.
ϕ	Learnable parameters of the CIGT routing units.
ρ	Routing probability thresholds.
θ	Learnable parameters of the CIGT network units.
τ	Temperature parameter for Softmax and Gumbel-Softmax routing functions.
$\pi(a_t\|s_t)$	Policy distribution at time step t.
Q^π	State-Action table under the policy π.
π^*	The optimal policy.

LIST OF ACRONYMS/ABBREVIATIONS

AIS	Adaptive Inference Graphs
AT	Approximate Training
CBAM	Convolutional Block Attention Modules
CIGN	Conditional Information Gain Networks
CIGT	Conditional Information Gain Trellis
CNN	Convolutional Neural Network
CSGS	Channel Selection with Gumbel Softmax
DAG	Directed Acyclic Graph
DNDF	Deep Neural Decision Forests
EM	Expectation-Maximization
FT	Full Training
GMM	Gaussian Mixture Model
GS	Gumbel-Softmax
HMoE	Hierarchical Mixture of Experts
IG	Information Gain
LSTM	Long-Short Term Memory
MAC	Multiplication and Accumulation
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MoE	Mixture of Experts
NAS	Neural Architecture Search
Q	Q-Learning or Q-Table
PG	Policy Gradients
RL	Reinforcement Learning
sMoE	Sparse Mixture of Experts

1. INTRODUCTION

Extensive success in image classification tasks has been attained by deep convolutional neural networks (CNN), thanks to their utilization of numerous parameters. However, the substantial computational demands of these intricate models make them impractical for edge devices such as mobile phones and embedded platforms. Conditional computing offers a remedy by selectively deactivating a portion of the entire model, using only a constrained segment for forward calculations and/or training updates, depending on the provided sample. The first idea regarding the use of conditional computation in deep neural networks has been presented in [1] by Bengio. Later, some different approaches followed this seminal idea. A certain group of methods opt for sampling smaller structures in a larger network based on the router's operations. These implicitly conditional methods involve dropping complete layers, blocks, or units [2–5]. Some other approaches follow a similar path; however, they first define a supernet structure and use subsets of this structure based on routing decisions taken by routers or similar mechanisms. The so-called "Slimmable Networks" and their variants constitute such approaches. In a different group of conditional models, pre-made structures are placed inside a larger architecture, and samples are used to decide which sub-structures to run. This way, there are no implicit sampling operations [6], [7], and [8]. Finally, some methods use the idea of early exits: Whenever a sample is deemed to exhibit enough information for a correct prediction, an early classifier processes it and gives the output earlier, compared to executing the whole network [9, 10].

Similar works that can be counted among conditional execution are Mixture of Experts systems, based on the idea of the seminal paper of Jordan et al. [11]. These models use routers or gating mechanisms to route samples to different experts in an ensemble-like system. Such models are especially challenging to define and train in the domain of deep neural networks, given that even though training a single expert network is an expensive operation, these algorithms work on the collaboration of many experts. Different routing, gating, and training mechanisms have been proposed and

analyzed for such works [12–14].

When it comes to conditional execution, a tree structure is a perfect candidate. One can skip sub-trees and route a sample to the exact desired location. This will result in conditional execution without any special effort. However, by design, decision trees and neural networks have very different characteristics. There is a large literature of different works that try to bring the best of these two approaches together [15–19]. It should be noted that most efforts to combine the neural networks with tree structures are motivated by the interpretability of the trees and the black box characteristics of the neural networks, but the conditional execution characteristics of the trees are also of great interest.

Removing parts of a neural network explicitly and converting to a more compact version while retaining much of the generalization power at the same time is the aim of pruning models. Pruning models are performed by eliminating layers, channels, neurons, and sometimes whole blocks from the neural networks to reduce the model size and lead to increased efficiency. In our case, the structured pruning methods are the most important. They get rid of channels or subsets of layers in a structured way, such as by channel granularity, so that the remaining method can still work with hardware acceleration [20–22].

Finally, not very close to the idea of efficient computation but bearing occasional similarities, are the methods in the domain of Neural Architecture Search (NAS). Such methods usually aim to learn the model structure from the data as well as the model weights. The reason that these approaches are relevant to conditional computing is that they also aim to find subsets in a larger search space of larger models to work on. In the case of conditional computing, a sample dynamically selects a sub-network; however, NAS methods seek a sub-network from a larger search space during the training and retain the structure during inference time [23–25].

Pursuing the idea of conditional computation, we designed deep neural network architectures that are explicitly tree- and trellis-shaped. With the information from the samples, both of these structures can choose parts of the neural network architecture ([26], [27], and [28]). We call our tree-shaped neural networks "Conditional Information Gain Networks", shortened as CIGNs ([26], [27]). These tree-structured feed-forward neural networks can choose optimal subsets of their architecture depending on the provided sample. In the CIGN model, every non-leaf node makes a routing decision by maximizing a local information gain objective, thereby reducing entropy in the subsequent child nodes. The information-gain-based routers in the non-leaf nodes also provide the ability to group semantically similar classes. The main goal of doing such a grouping is to delegate the task of distinguishing between semantic class groups to simple routers, thereby facilitating the relatively difficult task of classification between semantically similar classes, which is left to the deep classifiers in the leaf nodes.

A CIGN model contains two kinds of transformations: the first transformation (F) drives the classification objective, and the second transformation (H) controls the routing through an information gain objective. In addition to the usual classification cost, each non-leaf node has a local information gain objective that is optimized using gradient descent-based methods. This objective is used to control the routing process. Maximizing the information gain results in lower entropy at the deeper levels of the neural tree since the deeper nodes get data batches with higher concentrations of certain classes. This creates expert sub-networks, each of which is fine-tuned to discriminate between semantically similar classes. This may allow the sub-trees in the network to learn the finer differences between classes.

The specialized sub-networks are then tasked with a simplified classification, facilitating a decrease in the number of parameters within the child nodes while maintaining comparable accuracy with an unconditional, thicker model. This configuration naturally lends itself to conditional computing because information gain objectives direct each sample to the most likely path, resulting in sparse updates and executions.

The routing mechanisms in the CIGN models do not work perfectly. For some of the challenging images, a sample can be misrouted into a subtree that does not belong to its semantic group. To address this issue, we suggest introducing probability thresholds during the inference phase for each parent-child node connection. If the routing probability exceeds these thresholds, a sample can be directed into more than one path at each decision node. Samples with high entropy in the decision nodes, indicating difficulty in classification, are directed into multiple leaf nodes. The final classification result is computed as the average of the posteriors in these leaves, constituting our Sparse Mixture of Experts approach. We determine optimal routing thresholds through Bayesian optimization on a validation set, minimizing a weighted loss that incorporates classification accuracy and the average number of MAC operations per sample. Through extensive tests on MNIST, Fashion-MNIST [29], and CIFAR-100 [30] datasets, we demonstrate the effectiveness of CIGN models enhanced with the Sparse Mixture of Experts approach. These models maintain high generalization performance, comparable to a thick unconditional model, while operating with the same computational efficiency as a much thinner model. The Sparse Mixture of Experts further improves generalization at the cost of a tunable additional computation load during the inference phase. We compare our CIGN-based methods with structural neural network pruning methods from the literature, showing that our proposed methods achieve higher classification performance at similar sparsification levels.

In addition to the tree-shaped CIGN models, we also designed trellis-shaped conditional models, which we name "Conditional Information Gain Trellis" (CIGT) networks. One problem with the tree structure of CIGN models is that misrouted samples can end up in sub-trees that are not specialized to classify them. We introduce a novel conditional computation model designed in a trellis shape to address the issue of misrouting. Unlike a tree, where there is only one path between two nodes, a trellis structure offers multiple paths between nodes. If a sample is misrouted to an incorrect computational unit, it retains the opportunity to choose the correct path in the subsequent layer. CIGT adopts the information gain-based routing mechanism from [26], implementing it within a Directed Acyclic Graph (DAG) or Trellis structure.

Our goal is to establish expert paths within the Trellis structure to accommodate various semantic groups in the provided dataset. We partition the computational units in each Trellis layer (or interchangeably, a Trellis block) into N parallel groups, referred to as F units, each specializing in learning essential representations for the data. Additionally, there is a single H unit, termed a router, that learns a probability distribution across the F computation units in the next Trellis layer (block). The local probability distributions within each block's F units collectively form a global probability distribution along each root-to-leaf path in the Trellis structure. Consequently, the model can be viewed as a hierarchical mixture of experts model [11].

Due to the large number of experts—each root-to-leaf expert being a convolutional neural network—the EM training for the mixture of experts model required by the CIGT model is challenging. Conducting a maximization step for each expert entails a highly resource-intensive neural network optimization operation. Instead of employing an Expectation Maximization-based training approach, we introduce routing mechanisms based on differentiable information gain objectives [26, 27, 31], that are similar to the ones used in CIGN models. These mechanisms enable sparse execution of the computational blocks and facilitate the grouping of samples belonging to semantically similar classes. It is expected that the deep network along the path will naturally learn to tell the difference between similar samples with small semantic differences when they follow the same root-to-leaf paths in the DAG structure. This will make the original multi-class problem easier to solve. The probability distributions over each block's F units are then trained using the corresponding local information gain losses from the previous block. We posit that the Trellis structure inherently offers greater robustness as a topology compared to trees [26]. This is because, in the event of a sample being misrouted within a specific block, there exists the potential for recovery in subsequent blocks. Due to the mixture of experts structure of the CIGT models, we identify structural problems with the vanilla gradient-descent training procedure and propose a Gumbel-Softmax relaxation-based alternative training algorithm that allows the use of full gradients from the mixture of experts loss function [32, 33]. For the CIGT networks, we conduct experiments on MNIST, Fashion-MNIST, and

CIFAR-10 [30] datasets.

Although the trellis structure is more well-behaved in comparison to trees regarding the misrouting behavior of difficult samples, these errors will still exist in CIGT models. It is natural to look for ways to exploit the knowledge in multiple root-to-leaf expert networks for samples that we deem difficult. Like in the tree-shaped CIGN networks, we introduce probability thresholds to the routing units in every CIGT layer. Whenever a sample produces routing probabilities that are larger than these thresholds, the sample is routed into multiple paths. This multiple-path routing behavior will result in the evaluation of multiple root-to-leaf expert networks in the CIGT structure. The classification probabilities from these experts are averaged to obtain a similar Sparse Mixture of Experts behavior as in the CIGNs [27]. We set up an optimization problem on a validation set to find the best routing thresholds for the routing units. The goal is to find the sweet spot between a higher level of accuracy and the least amount of extra computation (as measured by the MAC metric, which stands for "multiplication and accumulation"). The optimization problem is solved with the Cross-Entropy search algorithm [34], which is a black-box global optimization method for functions that are not easy to maximize numerically. We experimentally show that attaining elevated classification performance is possible with a small increase in the computational burden.

The multiple path routing behaviors during the inference phase that we mentioned so far depend on determining optimal routing thresholds that are global, meaning that once determined, they are used for all the test samples after the optimization. Ideally, a more elegant multiple-path routing mechanism might involve determining different thresholds or routing behaviors dependent on individual samples. In our conditional execution scheme, we interpret the multi-path routing behavior during the inference as a Reinforcement Learning problem. After interpreting a sample´s routing decisions at every level in a conditional network in the framework of a Markov Decision Process (MDP), the optimal policy would entail routing decisions (resulting in a single path or multiple paths) that increase the overall model accuracy while causing as

little extra computation burden due to multi-path inference as possible. We use two approaches to solve this problem: The first is a Policy Gradients approach that uses the REINFORCE [35] algorithm to determine the optimal routing policy. The original REINFORCE algorithm proposes a method for evaluating the policy gradients that need to be calculated for policy optimization in reinforcement learning algorithms. The second method is based on the idea of Q-learning. However, instead of relying on a design based on a full Deep Q Network (DQN) [36], we observe two things: First, the episodes of an agent in this particular MDP are always finite (equal to the number of levels or blocks in our conditional networks). Second: In practical usage, the depth of the conditional neural networks in our work does not exceed three, and the number of computation units at each level is at most four. These two observations mean that we can fully enumerate the optimal state-action tables for every sample without suffering from a combinatorial explosion of action space sizes. This allows us to convert the Q-Table learning problem into a supervised regression one. Hence, our second approach is based on this supervised Q-learning method. Based on this motivation, we develop a Policy Gradients and a supervised Q-Learning-based multi-path routing method and apply these to our trellis-based CIGT models, since the global optimization-based Sparse Mixture of Experts multi-path routing algorithms in the context of CIGT models resulted in inferior performance compared to our CIGN models. Our goal was to design a more successful Sparse Mixture of Experts approach for these trellis-shaped models. We use them on the same models that we have trained in our prior CIGT work.

In summary, we can list the contributions of this thesis as follows:

- We design a tree-shaped conditional deep neural network that allows routing of the samples in the tree structure with information-gain-based routers. We show the effectiveness of this routing mechanism with numerous experiments, demonstrating it can attain better or comparable accuracy in comparison to larger, unconditional baseline models. We call these models Conditional Information Gain Networks (CIGNs).

- Observing that samples that make uncertain routing decisions tend to be misclassified, we define probability thresholds at routing units and develop a Bayesian optimization algorithm for learning the optimal thresholds. This Sparse Mixture of Experts inference time behavior of the CIGN networks improves accuracy at a modest extra computation cost.
- Since tree structure is susceptible to misrouting problems, we define a Trellis-structured conditional neural network, which we call a Conditional Information Gain Trellis (CIGT). In addition to a vanilla training mode similar to the CIGN models, we define a Gumbel-Softmax relaxation-based full training mode for the mixture of experts loss function induced by the CIGT structure.
- Like in CIGNs, we introduce probability thresholds for the routers in CIGT models, that allow the evaluation of multiple paths in the trellis structure during the inference time. We design a Cross-Entropy search-based algorithm to determine the thresholds that improve the accuracy at the cost of minimum extra computation cost.
- We design inference time multi-path routing algorithms based on Reinforcement Learning. These algorithms effectively provide sample-dependent multi-path routing behavior, contrasting to global probability thresholds determined with Bayesian Optimization or Cross Entropy Search. We apply these algorithms to our Trellis-based CIGT models and show that they are performing favorably in comparison to the global threshold searching algorithms in the context of Trellis models.

The thesis is structured as follows: In section 2, we present a review of the literature on conditional execution in deep network architectures. In Section 3, we formulate the CIGN model and its Sparse Mixture of Experts extension for multi-path inference. Section 4 details the CIGT model. The CIGT model definition, its vanilla, and Gumbel-Softmax-based training algorithms are presented. We also show the multi-path inference algorithms for the CIGT model. In Chapter 5, we explain our Reinforcement Learning based multi-path inference algorithms for our CIGT models. In Chapter 6 we give our conclusion remarks and conclude the thesis.

2. RELATED WORK

Deep neural networks have demonstrated significant success in various machine learning tasks, particularly in image classification. A notable example is AlexNet [37], which marked the first instance of a convolutional neural network (CNN) achieving state-of-the-art performance on the ImageNet dataset. After this groundbreaking work, other successful CNN models like [38], [39], [40] and [41–43] have introduced innovative mechanisms to facilitate the training of deeper models. Despite the enhanced classification performance on diverse datasets offered by these sophisticated networks, their demanding memory and computational requirements make them impractical for deployment on resource-constrained platforms, such as mobile phones and embedded systems.

We designed the CIGN and CIGT models intending to reduce the burden of computational computation for complex deep neural networks and keep the model performance on par with unmodified methods, at the same time. Both the CIGN and CIGT models are linked to various domains within the neural network literature. They fall under the category of conditional computation models and represent a variation of the Hierarchical Mixture of Experts (HMoE) model. CIGN, specifically, being inherently a decision tree-neural network hybrid architecture, contributes to the relevant literature in this regard. Additionally, these models can be examined in comparison to network pruning methods, as both approaches aim to alleviate the computational and memory demands of neural networks. Another area of research, which has similarities with conditional computation is "Neural Architecture Search" (NAS) [44]. In NAS, the aim is to learn the optimal neural network structure for a given task. This involves learning the number of computational units, like the number of convolutional layers, their filter sizes, the channel count, how these are connected, and what type of non-linearities are used.

2.1. Conditional Computing

Conditional computation in neural computing serves two primary objectives. The first goal is to decrease the computational expense during both evaluation and training by selectively activating only a subset of the network. The second goal is to establish expert networks tailored for specific subsets of the input space, ensuring that the model's performance remains comparable to that of unconditional models.

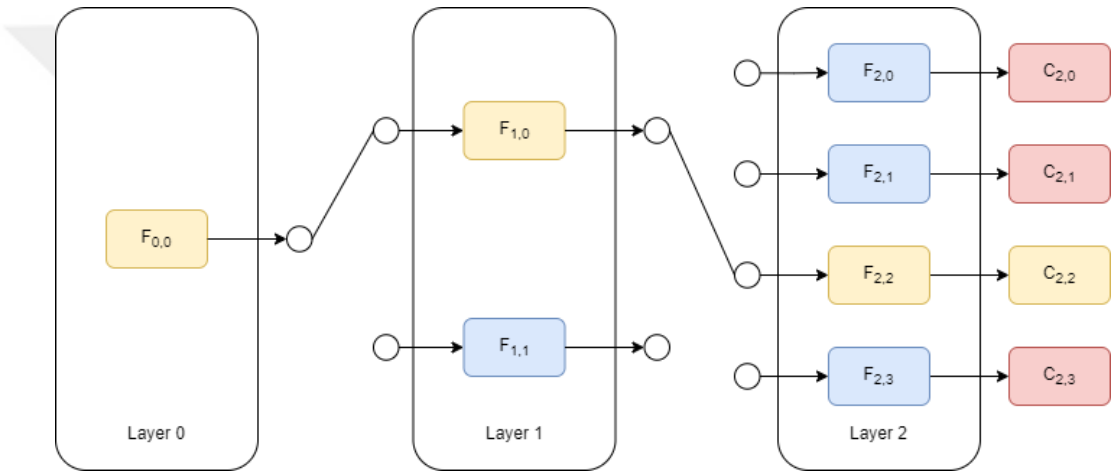


Figure 2.1. Conditional computing.

The first work, including a discussion on conditional computation, is by Bengio, in [1]. A seminal work that proposes a simple yet very effective way to deal with differentiability issues that come with conditional execution due to the search space becoming discontinuous is given by Bengio et al. [45]. They present "Straight-Through" estimators designed for handling discontinuous operators, as well as suggest stochastic neurons capable of toggling between an on or off state based on their outputs, allowing specific segments of the network to be activated or deactivated. The study of Murdock et al. [2] presents a modification of the well-known Dropout regularizer [46], known as the "Blockout" method. In this technique, random blocks of the weight matrices in a neural network are stochastically set to zero. Unlike Dropout, the probabilities of dropping follow trainable Bernoulli distributions. To learn these Bernoulli parameters, Straight-Through estimators are employed, as outlined in [45]. Similarly, Li et al. [47]

apply the probabilistic disabling of the neural network elements to different modalities in multi-modal applications. In the work by Bengio et al. [48], binary masks are employed on the weight matrices of the neural network. Similarly, these binary masks are sampled from Bernoulli distributions. The key distinction lies in the training approach; they use the REINFORCE algorithm [35] and the policy gradients method to learn the Bernoulli parameters.

The "Blockdrop" algorithm, introduced by Wu et al. [3], introduces an auxiliary network that learns to selectively omit certain convolutional layers within a ResNet architecture. They employ reinforcement learning to identify the layers to be omitted. On the other hand, Veit [4] presents adaptive inference graphs for deep neural networks, utilizing a categorical distribution to decide whether to skip the current layer. Its noteworthy contribution lies in the use of the Gumbel-Softmax approximators for discrete sampling behavior in neural networks [32, 33]. In their work, McGill et al. [49] put forth three distinct training methods for dynamically directing samples in a convolutional neural network with multiple columns and layers. Meanwhile, "SkipNet", as presented in [50] by Wang et al., employs gating functions to probabilistically decide whether to execute a layer or skip it. The training of both the gating functions and the backbone network involves a hybrid supervised-reinforcement learning algorithm. For achieving conditional computation, Herrmann et al. [5] use Gumbel-Softmax-based channel selection in convolutional neural networks. This is different from [4, 50] in the sense that both of these methods apply conditional computation to the granularity of complete layers, whereas [5] works on the scope of individual filters. Krishna et al. use the ideas of channel selection, as presented in [5], in the domain of self-supervised deep neural networks [51]. Selecting computation units based on a router system has also been extended to visual transformers by Wojcik et al. in [10]. The routers for the transformer units are trained separately from the actual network by using a heuristic that is based on the distance between the activated units´ generated outputs and the complete output of all units. Only a minimum number of units that provide a distance smaller than a predefined threshold is kept as the router target.

Recently, an approach called "Slimmable Neural Networks" has attracted interest in the domain of conditional computation [52]. In this approach, neural networks with lower complexities, which are primarily measured by the number of layer widths (number of convolution channels, hidden units in dense layers), are embedded into a larger network, usually called the "Supernet". According to the sample complexity, the network selects the appropriate smaller network in the Supernet. This selection usually occurs by masking the unused channels or excluding them with indexing operations in deep neural network libraries [53]. Later, some additional work extended this idea. Yu et al. [54] introduced "Sandwich Rule" and "In-Place Distillation" algorithms for effectively training the slimmable networks. The rule pertains to effectively training a batch of networks with different complexity in a supernet. This rule observes that training the smallest network, the largest network, and several networks that have randomly sampled widths from the Supernet in a unified loss function leads to better model performance compared to individually training models of different complexity. In-place distillation trains the lower-complexity models with the soft targets of the largest network during the training. There are also suggestions regarding the modification of Batch-Normalization [55] for the Slimmable Networks context. Since models of different widths share the same superstructure, calculating the population statistics and the scale and location parameters of the batch normalization layers proves to be problematic due to the conditional switching in these layers. The learned statistics for different channels may be incompatible. A calibration step after training networks has been proposed to tackle this issue in [54]. Li et al. extend the idea of network slimming to additional dimensions of the networks like kernel widths, padding lengths, and dilation step sizes [56]. The same authors later extend their work in [53] to incorporate new training strategies for slimmable networks, like "In-Place Bootstrapping", where the in-place distillation works on the soft targets of an exponential moving average version of the largest network to prevent fluctuations in the soft targets. Moreover, they introduce dynamic slimming, which involves selecting different-sized networks based on router structures. Chin et al. [57] propose a framework for jointly optimizing weights and widths of slimmable networks instead of using a set of predefined widths. Jiang et al. use a dynamic slimmable neural network for image denoising [58]. Similar

to [53] they use decoupled training steps for first training the supernetwork, the slimmer networks, and the gating mechanisms. Using the ideas that have been proposed in slimmable networks, Yang et al. utilize the concept of having a supernet that can be partitioned into smaller sub-networks in the domain of Continual-Learning, where the aim is to prevent a neural network from forgetting its old knowledge regarding previous tasks it has been trained on when it is introduced to new data and a new task [59]. They propose "ScrollNet", where a subgroup of parameters in a supernetwork is assigned different importance weights based on the different tasks to which the network is introduced.

These methods differ from our work in an important aspect: we do not have an explicit supernet structure in our conditional networks; the smaller networks only partially share certain computation blocks according to the tree and trellis structures, and these blocks are used to their full extent by these smaller networks. There is no explicit layer sharing between different complexities; hence, we are not affected by the problems caused by Batch Normalization layers.

The methods discussed so far implicitly involve sampling different network structures per sample, typically achieved by dropping or freezing specific parts of the neural network. Another approach is to pre-determine an explicit hierarchical structure and employ routing mechanisms to activate subsets of that hierarchy. In [6] by Ioannou et al., small routing networks are utilized to assign weights to each neural computation path in a Directed Acyclic Graph (DAG) structure. Subsequently, these paths are merged through weighted averaging, and during the inference phase, only paths with the top-k weights are selected by the routers. In a different approach, Xiong et al. [60] employ a binary tree-structured neural network with a routing mechanism involving split losses attached to the outputs of convolutional layers in each non-leaf node. This loss aims to maximize the distances of data means from a splitting hyperplane and does not utilize any supervision signal. Another routing-based mechanism is employed by Liu et al. in [7]. Similar to [6], they define routing networks responsible for sending samples to different subnetworks, but in this case, the entire network and the routing

mechanisms are trained using Q-learning.

Genetic algorithms have also been utilized to learn connections between different neural modules. Fernando et al. build a DAG structure of neural networks, and their optimal connectivity is learned by a genetic algorithm [61]. Denoyer et al. build a similar, tree-structured network that consists of computation and routing units [62]. The routers and the computational units are trained using the REINFORCE algorithm, and therefore it bears similarities with the approach in [7]. The "Branchynet" model, as introduced in [9] by Teerapittayanon et al., incorporates early exit branches that furnish an initial classification outcome when the entropy score of the posterior probability reaches a satisfactory low level. Expanding on the concept of early exits, Han et al. introduce in [63] a weight learning scheme for samples exiting early and late. The aim is to reconcile the differences in behavior between the training and testing phases of these models. Considering a computation system that partially lives in a cloud and partially in an edge environment, Pacheco et al. show an efficient partitioning algorithm for an early exit network, deciding how the exits should be placed concerning the cloud and edge environments [64]. They convert the problem into a shortest-path problem, which is solvable in polynomial time. Chiang et al. [65] also worked on the problem of optimally placing the early exits in an early exit setting. Formulating the branch location problem as an optimization, they first show the problem is NP-complete, and then they design an approximate placement algorithm that is built on dynamic programming, which effectively can find optimal locations for exit networks in polynomial time. Another work that tries to improve the idea of early exits is proposed by Ayyat et al. [66]. They focus on specific critical classes that are determined before the execution of the early exit model and allocate specific early exits solely for such classes. This is helpful for cases where the detection of some classes is more time-critical compared to the rest. Jeon et al. propose an early-exit mechanism [67] where they learn the optimal locations of the early-exit classifiers and layer-wise mixed precision for quantization as a joint optimization problem. Yang et al. approach the problem of early exit from the image resolution perspective [68]. They note that, for easy samples, low-resolution versions of an input image are enough to correctly predict the label.

However, for images where distinguishing properties are hidden in the fine details, higher resolutions might be needed. Based on this observation, they first feed the lower-resolution images to a lightweight neural network. If this network cannot produce highly confident results, then a higher-resolution version of the image, in addition to the feature maps extracted by the lightweight network, is fed to a larger network. This operation can be cascaded for multiple layers of networks.

2.2. Mixture of Experts

In the context of the hierarchical mixture of experts model (HMoE), gating mechanisms assign probabilistic weights to various experts from their mixture. A Bayesian treatment of this setting is also available in [69]. However, in the domain of deep neural networks, the Mixture of Experts-based approaches is less common due to the challenges in training and evaluating such models [70]. Consequently, various approximate or sparse adaptations to the original HMoE model have been proposed for deep neural networks.

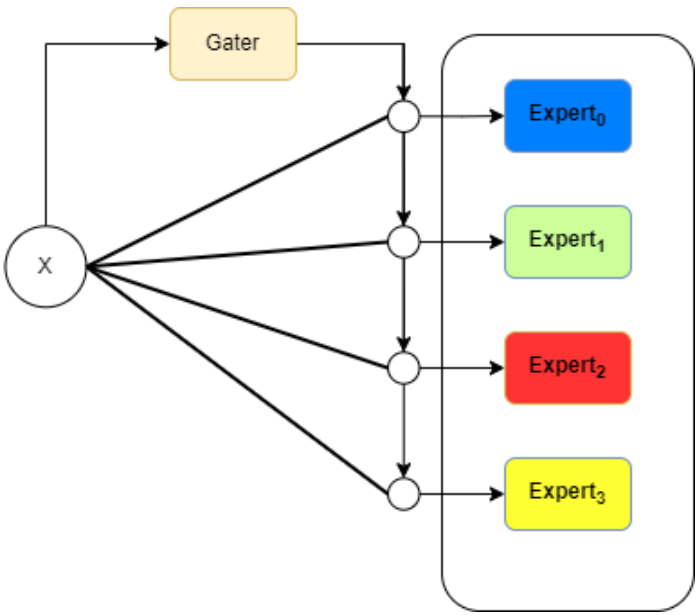


Figure 2.2. Mixture of experts.

Eigen et al. incorporate the mixture of experts model by stacking multiple layers of expert and gater networks in a feedforward deep neural network [14]. Shazeer et al. present a sparsely gated mixture of experts model [12], where the sparse selection and evaluation of experts make training and inference more manageable. This involves picking the top-k experts during both inference and training, and they suggest methods for load-balancing data among the expert networks. In [71], Lee et al. show different ways of ensembling convolutional neural networks as strong learners, beyond simple averaging of independently trained experts, have been investigated. It has been shown that ensembling strategies involving parameter sharing and ensemble-aware losses outperform simple averaging of independent networks.

Another method is HD-CNN, which was first described by Yan et al. in [8]. It first finds large class groups using spectral clustering and then builds a tree-structured network. Similarly, Ahmed et al. [17] propose a comparable architecture with generalist and specialist networks. In this instance, the generalist network first trains to divide the data into coarse categories, and then the expert networks train for each coarse category. Subsequently, these expert networks are concatenated and fine-tuned. However, this system does not involve conditional execution. For speech enhancement, Chazan et al. [72] employ a deep mixture of experts network, utilizing a modified version of the original Expectation-Maximization (EM) approach for training. The "Born-Again Neural Networks" proposed by Furlanello et al., describe a way to build multi-path neural network structures where an ensemble of similarly structured networks is created iteratively, with every newly added network generated from the intermediate previous one, with knowledge distillation [73]. Finally, these networks are used in ensemble fashion. While it does not involve conditional computing, this work is an important attempt to build multi-path architectures in a principled way.

Kirsch et al. in [74] treat a deep neural network as a MoE model and train it using a Viterbi-EM scheme. They especially focused on the problem of "mode collapse", in which a high percentage of the samples chose only a particular expert in an MoE scheme. They claim that the Viterbi EM training also handles the load balancing

inherently but they have reported mode collapse problems in their training processes nevertheless.

Recently, there have been attempts to formally analyze the behavior of Mixture of Experts systems and their sparse activation-conditional execution behaviors from a theoretical perspective. Baykal et al. introduce a mathematical model that represents sparsely activated deep neural networks, where the activations are decided by locality-sensitive hashing functions [13]. Locality-sensitive hashing maps points in a metric space to buckets; hence, sparse activation is created based on the bucket membership. They show that such models can approximate a limited set of functions, which are Lipschitz functions, as well as dense networks, but in a much more efficient way due to sparse executions. While the real-world data entails much more complex functions in general, the theoretical insights put forward in this work are valuable. In [75], Dikkala et al. provide another experimental and theoretical perspective on the behavior of Mixture of Experts systems that are trained on synthetic data with well-defined properties. For example, when the data is coming from a Mixture of Gaussians, the MoE models can learn to assign samples to different experts that correspond to components of the mixture models. They also experimentally show that router functions that are learned from the data lead to more performant models compared to predefined routing functions. In their theoretical results, they showed that under certain conditions, the routers can learn to disregard uninformative dimensions in the data and base their routing decisions on the subspaces where the actual data-generating manifold lives. Provided that the data comes from spherical Gaussians and for every row of the router matrix a randomly sampled data point is used, for every cluster, they also show how many experts would be needed to assign at least one expert to every data cluster.

The large-scale Transformer models are lately the focus of very intense research due to their capacity to learn very complex data structures [76, 77]. Mixture of Experts systems are also getting more attention for these kinds of models because they can benefit from the efficient inference properties that can be provided by network structures that run only when certain conditions are met. “Switch transformers” pro-

posed by Fedus et al. [78] apply the idea of Mixture of Experts and routing into expert computation units via a trainable router in the field of Transformer networks.

2.3. Neural Network - Decision Tree Hybrids

Hybrid models that combine decision trees and neural networks exhibit a wide range of combinations from both domains. Neural Trees, which induce oblique decision trees with perceptrons or MLPs as splitting functions in non-leaf nodes, have been explored by various studies [79–83]. These algorithms, akin to common ID3 and C4.5 decision trees [84], recursively grow a decision tree, minimizing class impurity criteria like Gini index and Shannon entropy at each split node through exhaustive search or random sampling of split parameters. In Neural Trees, this induction algorithm is modified with an MLP or Perceptron, trained using conventional numerical optimization methods to find an optimal split.

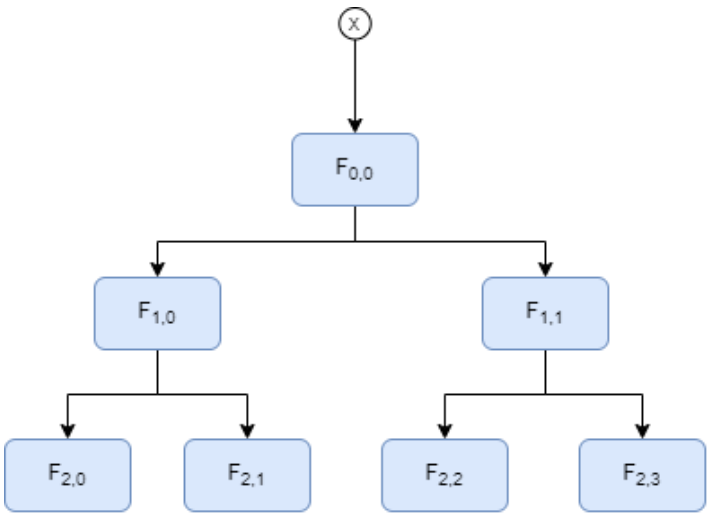


Figure 2.3. Neural trees.

The methods within the family of Neural Tree approaches often differ in defining neural split functions and optimizing them. Despite variations, they commonly find the best splits locally during tree induction, and the discovered split functions and parameters remain fixed throughout the rest of the training process. Recent approaches

have introduced the use of deep neural networks in such hybrid structures. For instance, the "Deep Decision Network" by Murthy et al. [85] employs a recursive architecture based on spectral clustering. They classify data with a root network, perform spectral clustering on misclassified samples, and train new networks for each misclassified cluster while keeping previous networks fixed.

The "Deep Neural Decision Forests" (DNDF) by Kontschieder et al. [15] replace conventional softmax classifier layers in deep neural networks with decision trees. Each split node of these decision trees connects to the last fully connected layer of the root neural network, acting as a soft router for child nodes. Categorical class distributions are held in the leaf nodes, and an iterative algorithm learns both the categorical distributions and the neural network parameters. Different researchers have used the DNDF framework for various purposes. In [86] Shen et al. modified this framework for age estimation. Similar to [15], an iterative algorithm first fixes the leaf nodes and then learns the split nodes and the CNN backbone parameters. Afterward, the leaf nodes for the regression are learned by using a variational bound, holding the rest of the network fixed. Another DNDF-based application has been put forward in [87] by Roy et al. . In this work, the framework is used for monocular depth estimation. The trees that constitute the forest contain CNNs that apply filtering and routing of the sample patches down the tree. The leaf nodes contain Gaussian distributions that act as weak regressors. Shen et al. in [88] use the DNDF framework for label distribution learning. A similar work, as [15], is given the name "Neural Backed Decision Tree" (NBDT) in [18] by Wan et al. . Similar to DNDFs, they replace the last classification layer, which is usually a fully connected layer, with a decision tree. However, they build the tree hierarchy based on the trained network parameters instead of data-based measures. They want to make a soft routing system using the rows of the last fully connected layer. Each row of the fully connected layer is a tree node, and an agglomerative clustering algorithm is used to build the tree's hierarchy from these row vectors. While this induced soft classifier works better than alternatives in terms of model performance, it does not have an immediate conditional execution effect. Hazimeh et al. [89] propose a differentiable tree ensemble layer for deep neural networks that replaces the conven-

tional dense (fully connected) layers. They build the design of the tree layer's routing structure on top of a soft tree-based routing system like [15]. However, they design a smooth-step activation function instead of a regular sigmoid that is exactly zero outside a predefined interval around the origin, and it is still first-order continuously differentiable at the interval points. Inside the interval, they are using a cubic polynomial. With this activation function, they can execute sparse forward and backward passes and conduct conditional execution, in contrast to similar approaches. However, this conditionality is only limited to the proposed ensemble layer. Another work that utilizes tree architecture in the context of deep neural networks is in [90] by Song et al. The authors inspect the decision-making process for classification in a knowledge distillation setting [70] and empirically show that it is realized in a coarse-to-fine way that is distributed among layers. The early layers capture salient features for distinguishing coarse classes, whereas the final layers learn more fine-grained differences. Since this hierarchical decision-making process is akin to a tree structure, the authors design a "Tree-like Decision Distillation" algorithm. Instead of using teacher network outputs conventionally for knowledge distillation, they incorporate the tree-like decision hierarchy of the teacher network into the distillation procedure. Tree-based architectures are popular for interpretability in machine learning since a tree structure outputs a sequence of decisions that allows a human user to closely inspect the model behavior. This is why, in addition to conditional computing, it is common in the field of interpretable machine learning to combine tree structures with more advanced machine learning algorithms, such as deep neural networks and ensemble models, such as random decision forests or gradient-boosting trees. Pioneering work in this area that involves building a tree-structured deep neural network is by Frosst and Hinton, [91]. They train a soft decision tree, whose intermediate nodes contain trainable feature vectors and leaf nodes contain linear classifiers, to apply knowledge distillation from a pre-trained deep neural network. The inner nodes of the soft decision tree learn how the deep network makes senseful decisions when trained on soft targets made by a deep network that has already undergone training. Using soft trees as interpretable models has been used in [92] by Yoo et al. as well. The authors propose the "Ensemble to Distilled Tree" algorithm. Claiming that large ensemble models like random forests

obfuscate the decision process as deep neural networks do, they construct a variant of soft decision trees used in [91]. Their variants called "Compact Soft Decision Trees" use only a subset of the features in the tree nodes and later simplify the model by pruning, namely removing redundant branches in the tree, leading to improved accuracy over vanilla soft decision trees. To build an interpretable deep neural network that has a tree shape, Ouyang et al. [93] first train a conventional deep neural network. For a D layered neural network, they build a binary tree of depth $D-1$. The pre-trained network layers are replicated to constitute the transformation nodes at every tree layer. The routers are affine transformations of the transformation node outputs, going through a sigmoid function to create binary distributions on the child nodes. The training involves first learning the router parameters by fixing the router parameters and then fine-tuning the whole network. The decisions of the learned routers can be used for model interpretability afterward. Integrating trees as interpretable structures into deep neural networks has also been investigated in [94] by Nauta et al. . They insert a tree structure in the final layer of a convolutional neural network, in which every split node contains a trainable "prototype", that corresponds to specific parts of an input image. The similarity between these prototypes and certain subsections of the images guides the routing in the tree structure later. It is hypothesized that these prototypes would learn to emulate structures that are specific to particular classes, like the beaks of certain bird species. Hence, the decisions made in the router tree would be analogous to a human decision-making process. Ji et al. [95] also use a tree structure for tackling the fine-grained visual categorization problem. Like other works that use neural trees with soft routing, they create a full binary tree-shaped neural network where differentiable neural units decide the likelihood of routes in the tree. A weighted sum of the leaf nodes is what determines the classification. They also use attention-based transformers to increase the discriminative capacity of the network. It should be noted that no conditional computing takes place in such soft routing-based tree-neural network hybrid designs. Tree-structured neural networks have also been used outside the image classification domain. Fu et al. [96] propose a shot-level video summarization architecture that utilizes a binary neural tree as a hierarchical regressor, where each root-to-leaf path corresponds to a regression function that evaluates

the given shot sequence from a particular perspective. The binary tree contains neural regressors in the non-leaf nodes that assign probability weights to each root-to-leaf regression network in the model. The final score is calculated as a weighted sum of regressor predictions. Again, this approach is focused on the discriminative power of the tree topology instead of conditional computing. Another example of a neural network-tree hybrid that focuses on non-image data is given by Luo et al. [19]. They propose a hierarchical, differentiable neural regression tree that they call "Decision Tree Regressors". This model is designed for tabular data The routing and prediction mechanisms are connected to neural units. A weighted average of the tree's leaf nodes performs the regressor prediction.

In contrast to the neural network interpretability approaches, which use an explicit tree structure to enforce explainable neural network operations, Wu et al. [97] use a regional tree regularization term in addition to the usual loss function of a deep neural network. This method encourages the model to have good approximations in different regions of its input space by using different decision trees. This allows a better generalization compared to confining the whole model into a single decision tree. Vidal and Schiffer propose "Born-again tree ensembles", where they define some rules and use knowledge distillation to construct a single decision tree that is interpretable from an ensemble of trees. The rules they define constrain the constructed tree to give the same outputs for the same inputs in the training set, as the teacher tree ensemble [98].

Another avenue in hybrid models is to interpret the weight connections of a neural network as a decision tree [99]. Baek et al.'s work [16] focuses on converting deep convolutional neural networks into decision jungles [100] by defining layer-wise entropy losses on probability distributions derived from response map activation strengths. Saxena et al. [101] propose "Convolutional Neural Fabrics", where a trellis-shaped network structure is designed between convolutional units of each layer. While no conditional execution has been considered in this work, it is an attempt to introduce tree/trellis structures into neural models. "Self-born wiring for Neural Trees" has been proposed by Chen et al. [102]. In this method, they first disconnect all individual

layers in a conventional deep neural network like VGG-13. Then these layers are combined into computational groups at every layer. The connections between these layers are then learned explicitly with a Hierarchical Mixture of Experts-based loss function. If the trained mixture model gives these connections a low likelihood of routing, compared to predetermined thresholds, the connections between computation units in later layers are cut down. The resulting neural tree is then fine-tuned for the target dataset. Aytekin in [103] shows that a pre-trained deep neural network has an equivalent tree representation that can be utilized for interpretability.

In their paper, Wang et al. [104, 105] suggest using a tree-structured neural network. In each split node, the distance from the splitting hyperplane is added to the original feature. This lets the whole tree-structured network be optimized on a global level, since adding up hyperplane distances makes it easier for gradient signals to change hyperplane parameters across the network. This stands out as the only neural network-decision tree hybrid algorithm allowing global optimization. Yang et al. [106] propose Deep Neural Decision Trees that change the greedy splitting with a gradient descent scheme, similar to [104]. However, they are using the Gumbel-Max trick in their difference. This model is designed particularly for tabular data.

While not strictly a neural algorithm, Zhou et al.'s "gcForest" [107] is an intriguing method comprising stacked layers of random forests, lacking differentiable neural modules. Outputs are concatenated at the end of each layer and fed into the next layer of random forests. In recent work, "Adaptive Neural Trees" Tanno et al. [108] apply the idea of probabilistic routing to tree-shaped networks inspired by [15], dynamically growing a neural tree using a validation set.

2.4. Network Pruning

A recent set of techniques for accelerating neural networks involves pruning [109]. While network pruning approaches aim for a similar goal of efficient network execution, they are strictly different from conditional computing methods. Pruning is a once-and-

for-all operation. It is either applied as a post-processing step after model training or it is integrated into the training step, such that the model learns how to prune itself during the usual gradient-based training procedure [109]. After pruning has been applied to a model, it will stay fixed for different samples. In conditional computation, no explicit changes are applied to the model. For each sample, a specific subset of the network is selected and executed. How to select this subset per sample is the main area of distinction among the conditional computation techniques. However, pruning approaches to optimize network execution efficiency bear similarities with conditional computation; hence, it is worthwhile to review some of the prominent examples in this domain.

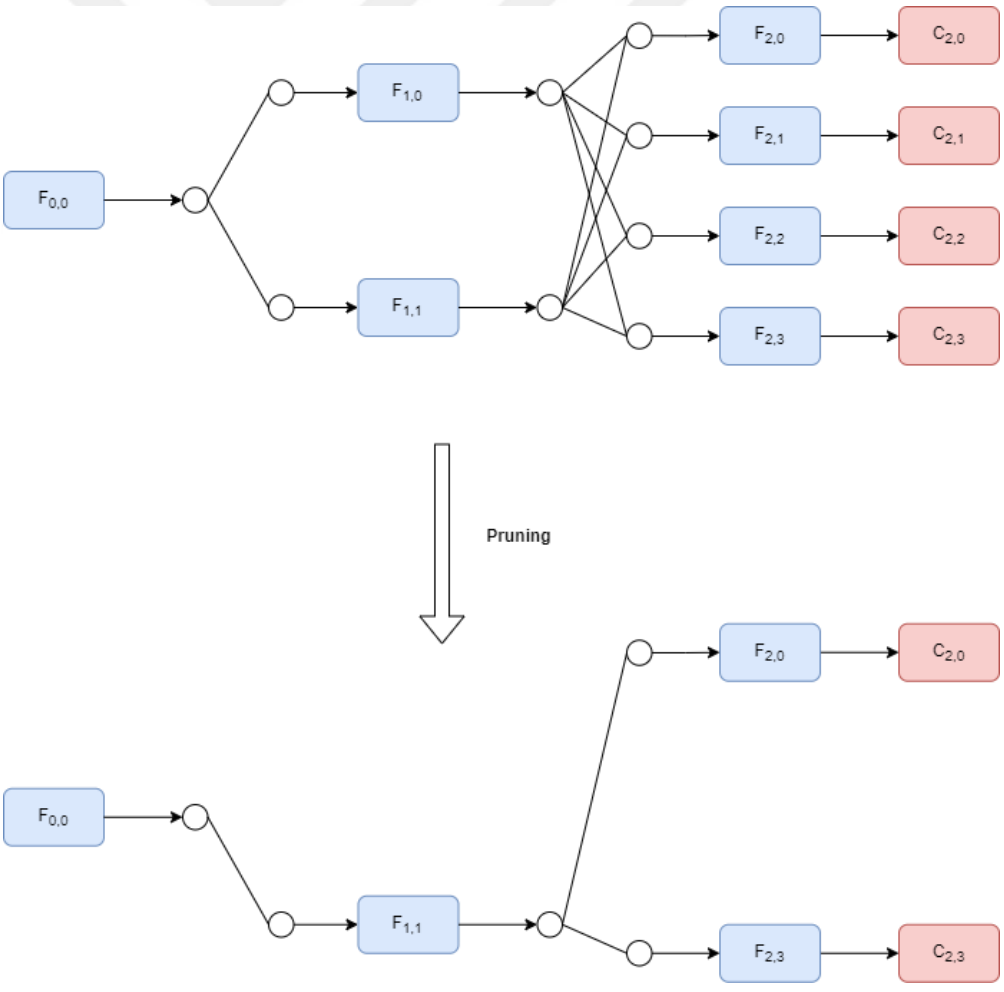


Figure 2.4. Network pruning.

Pruning seeks to alleviate the computational and memory load of neural networks by removing unessential weights and/or filters. These methods differ in how they determine which weights and filters to eliminate.

Categorized by whether retraining is required, in [109] Neill classifies pruning methods into four types: Magnitude-Based Pruning (MBP), where weights with the lowest absolute values are removed based on a threshold; methods introducing a special regularization term into the loss function to enforce smaller weights; methods eliminating weights based on loss sensitivity concerning a specific weight; and search-based approaches aiming to learn optimal links or paths for the given task. Additionally, a distinction is made between structured and unstructured pruning techniques. Structured pruning considers groups of weights and filters, aligning with hardware constraints to achieve faster models. Conversely, unstructured pruning provides models with increased flexibility, potentially leading to better generalization.

The CIGN and CIGT models function as conditional computation methodologies. Instead of engaging in the pruning of weights or filters, they choose to opt for the selection of the most advantageous root-to-leaf path within the context of tree or trellis-structured networks for each sample. This strategy diverges from pruning methodologies that aim to decrease computational complexity by discarding inconsequential filters. It is pertinent to highlight that both CIGN and CIGT inherently exhibit a pruning effect when restricting the selection to a limited number of root-to-leaf paths. Both CIGN and CIGT can be thought of in this way as similar to structured pruning algorithms, since the routing mechanism leaves out whole deep neural network structures. Consequently, related structured pruning approaches will be reviewed in the following.

The approach detailed in [21] by Molchanov et al. employs the Taylor expansion to characterize the impact of weight groups on the cost function. Subsequently, the groups with minimal effect on the cost function undergo pruning. Meanwhile, Mallya et al. in [110] introduce a masking method applied to the weights of an original network,

adaptable for new tasks through an end-to-end learning process. For filter removal across channels, shape, and depth, We et al. [111] employ a group lasso regularization, whereas Lebedev in [112] utilizes group-wise sparse regularizers to eliminate groups of filters and weights. In contrast, Molchanov et al. [22] adopt Variational Dropout, employing variational inference to learn individual dropout rates for each parameter in the network. This involves minimizing the bound between a variational Gaussian and a regular prior on the parameters.

Lin et al. in [113] introduce GAN principles into the field of network pruning, aiming to find a generalizable subset network through a minimax objective between the outputs of the baseline network and the pruned network. On the other hand, Louizos et al. in [114] use sparsity-inducing hierarchical Bayesian priors on network units like weights and filters, learning dropout rates through variational inference, and then getting rid of units with high dropout rates. This approach's advantage over the similar Variational Dropout method lies in its capacity to operate on groups of weights, making it advantageous from a hardware perspective. Additionally, Dai et al. [115] utilize the information bottleneck principle for compression [116], seeking to minimize redundancy between adjacent layers. This is achieved by penalizing the mutual information between each consecutive layer, and a variational inference framework is proposed for this minimization objective. Tang et al. in [117], propose "Sparse Connectivity Learning". This approach reparameterizes a network weight as the element-wise multiplication between a trainable variable and a binary mask. They also define that a proxy straight-through estimator for a binary mask element should produce positive gradients for mask variables to converge. They construct a suitable straight-through estimator in that regard and learn sparse connections via binary masks.

The approach called the "Lottery Ticket Hypothesis" and its numerous variations have gained increased attention recently [118]. This work empirically states that smaller subsets of larger networks, given the same weight and connection configuration, can achieve similar accuracy as the source network. Another idea for achieving structured pruning is to use a Neural Architecture Search framework that is designed for finding

efficient networks, which is the idea applied in [119] by Noy et al. . Another approach to pruning uses Reinforcement Learning to learn optimal policies that lead to an efficient neural network design [20].

2.5. Neural Architecture Search

An area of research that has a slight similarity to conditional computation is "Neural Architecture Search" (NAS). In NAS, the aim is to learn the optimal neural network structure for a given task. This involves learning the number of computational units, like the number of convolutional layers, their filter sizes, the channel count, how these are connected, and what type of non-linearities are used. The similarity between neural architecture search methods and conditional computing is sourced mainly from the following fact: conditional computing samples or explicitly chooses specific sub-architectures in a larger structure, which bears similarity to searching for an optimal architecture a priori like in NAS methods; however, in the case of conditional computing, it is focused on a single sample.

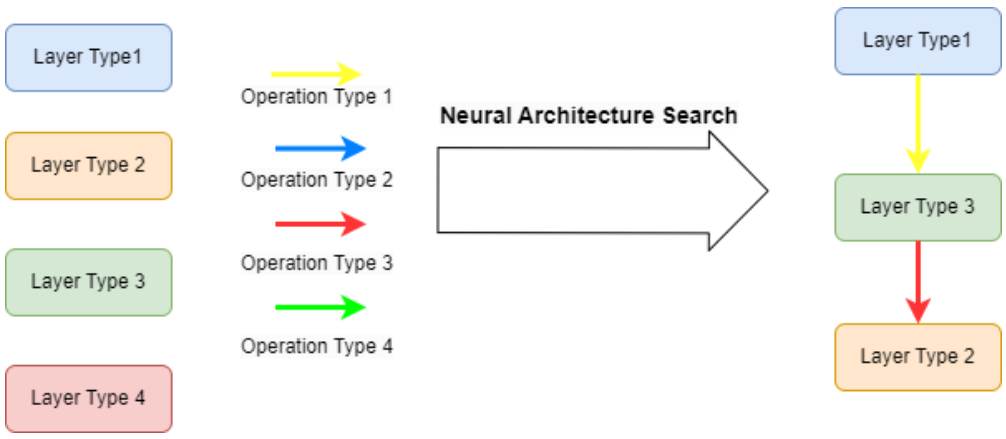


Figure 2.5. Neural architecture search.

It is common to use reinforcement learning or genetic algorithms to learn the connectivity between a fixed number of computational units in NAS applications [61]. Zoph et al. [44] and [120] use reinforcement learning in the form of REINFORCE algorithm [35] and policy optimization. In such algorithms, usually a proxy network

(for example, an RNN) is trained, which represents the policy generating the structure of a CNN. The reward signal is the classification accuracy on a validation set. Similarly, Zhong et al in [24] and Baker et al. in [121] learn the structure of the CNN by Q-learning. Cai et al. [122] use policy gradients, but instead of searching a discrete architecture space, it either widens an existing layer or adds a new layer to the network. This requires an initial model that goes through transformations according to the policy gradient training. Elsken et al. [123] use a hill-climbing search to learn the appropriate transforms on an initial model. Another popular strategy in neural architecture search is to use evolutionary algorithms (EA). In EA approaches, the network structure is encoded with a certain representation, and this representation goes through combinations and random mutations as the genetic algorithm evolves in a discrete search space. The quality of the combinations and/or mutations is measured on a validation set. [124], [125], [126], and [61] can be considered examples of evolutionary approaches to NAS. Different works do not use either RL or EA methods. For example, Liu et al. [127] use a heuristic search in the cell space. According to their claim, a search space on the CNN cell components is more efficient than a search space on complete models. Liu et al. [128] also propose a slightly different approach, called "Differentiable Neural Architecture Search" (DARTS). In DARTS, the computation of the model is represented as a directed acyclic graph. The authors introduce a continuous relaxation to the otherwise combinatorial search space, and they use straightforward gradient descent to learn the connections between various nodes in the computation graph. Specifically, the connections between the nodes in the DAG are represented as a discrete probability distribution, and the most likely connection is retained after the training finishes. Zhang et al. in [129] generate the topology and the weights of a model by using a graph neural network. Finally, Sirui et al. in [23] formulate the architecture search as the optimization of the parameters of a joint distribution on the search space for network cells. Similar to [128], its most promising aspect is coming up with a differentiable architecture search framework.

While NAS methods contain similarities with learning conditional computation in a Trellis or DAG structure, their main difference is that they learn a final model

for a specific task at hand, independent of how they formulate their search procedure. The only exception we are aware of is the InstaNAS paper by Cheng et al. [25], in which a controller is trained with a policy gradient, which builds a distribution of the possible architectures to pick from. During the inference phase, the controller selects the most suitable architecture from this distribution according to the input. From this perspective, this is the most closely related work compared to ours.

3. CONDITIONAL INFORMATION GAIN NETWORKS

3.1. Conditional Information Gain Networks

Our first attempt to design a conditional execution framework involving deep neural networks is the model we call "Conditional Information Gain Networks" (CIGN). The CIGN model is inherently a tree-shaped neural network, designed with computational efficiency in mind. This tree-shaped network consists of "split nodes" and "leaf nodes". A "node" is an abstract construct that contains a set of neural computational layers. When a sample is routed into a computational layer, that means it is fed into a composition of neural layers that transform the input and output into an intermediate representation. After providing a sample to the network, it chooses a particular sub-tree in the architecture. Skipping the rest of the network provides a computational advantage and makes up the foundation of the conditional execution idea in the context of this particular model. However, the way of selecting a particular sub-tree is of utmost importance, as we will show later in this chapter. The selection should be made in an informed manner, such that the selected sub-tree can provide the optimal representation for the sample. The likelihood that the neural network will correctly classify the sample increases with the selection of the appropriate subtree. Our core contribution to the CIGN architecture design is a router system based on a differentiable version of information gain terms found in conventional decision tree induction algorithms [84, 100]. We will experimentally show that routing decisions that are based on routers trained with these specifically designed objectives help the network attain comparable or better accuracy in comparison to thicker baselines that do not include conditional operations. We will also show that this design choice is significantly better than purely random routing decisions. An effect that we readily obtain with information-gain-based routing is that samples that are semantically similar to each other get grouped. In other words, the routing mechanisms we use result in "coarse classes". The leaf nodes of the tree structure contain cross-entropy-based classifiers. These classifiers receive samples that come from semantically grouped coarse

class groups. Hence, they can focus on hard-to-distinguish classes. Our experiments show that this effect helps with the elevated accuracy levels we obtain with the CIGN structure.

We have observed that the difficult samples result in uncertain router decisions: Information-gain-based routers would generate routing probabilities with a large entropy in such cases. The natural result of this is the misrouting of samples into leaf nodes (classifiers) that are not focused on the sample´s coarse class group. To recover such cases, we design a multi-path inference scheme into the CIGN structure. Based on the observation that such difficult samples incur uncertain routing probabilities, we determine network-wide probability thresholds for every split node and its corresponding router. When the routing probabilities are larger than these thresholds, a sample is routed into the corresponding child node. This routing mechanism results in the evaluation of multiple classifiers at the leaf nodes. The classification result is obtained by averaging the classification probabilities of the evoked leaf classifiers. We call this the "Sparse Mixture of Experts (sMoE)" mode of inference. The routing probability thresholds are determined by using Bayesian optimization on a validation set. The Bayesian Optimization algorithm minimizes a cost function that involves a mixture of model accuracy and extra expected MAC (multiplication and accumulation) that is caused by the multi-path routing. With such a loss function, we aim to find a compromise between increased model performance and additional computational burden. We will show the benefits of this Sparse Mixture of Experts design through experiments.

3.2. Architecture

In the CIGN model, consisting of a set of split nodes N_s and leaf nodes N_l, each split node may have two or more child nodes. For each split node, a sample undergoes two kinds of transformations. The first, F, involves learning optimal representations for the classification task, encompassing conventional CNN operations such as convolution, pooling, nonlinearities, residual connections [39], Dropout, and Batch Normalization. Meanwhile, the second set of transformations, H, functions as routers and is linked to

the information gain losses. These routers may also include common CNN layers, but for efficiency, they should be as lightweight as possible. By default, they connect to the F transformations and can share parameters with representation learning stages to reduce complexity.

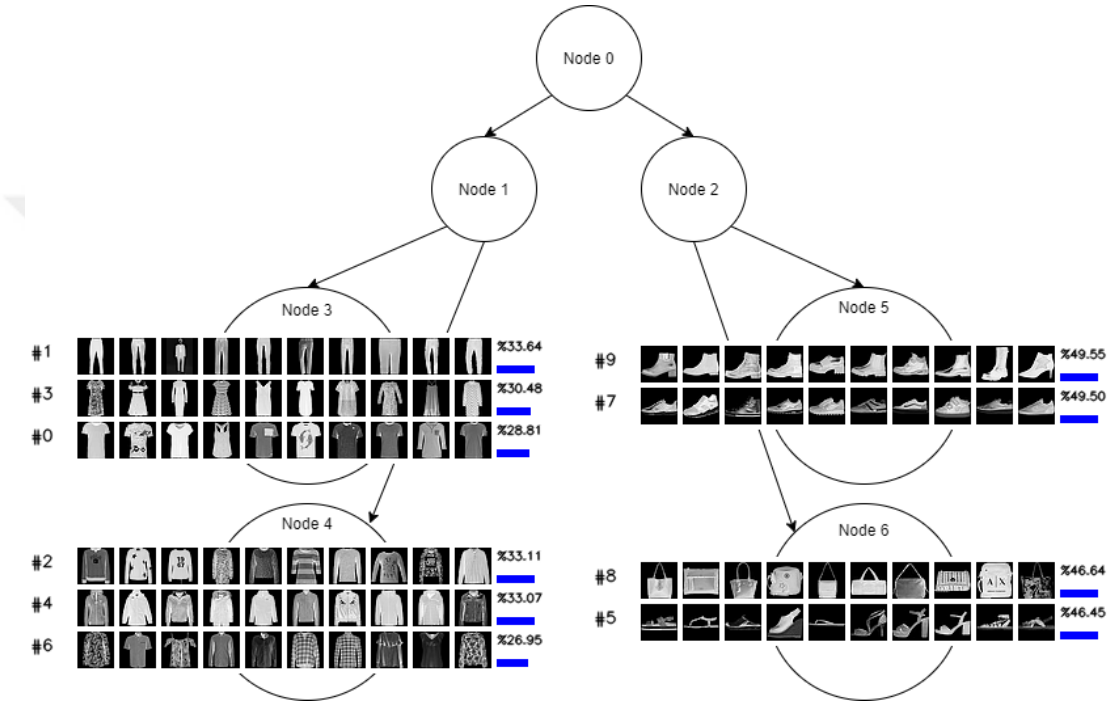


Figure 3.1. The semantic partition of the data via information gain-based routing system. The samples are from the Fashion-MNIST dataset [29]. Each row of images represents a separate class.

When CIGN is used, each router network sets up a probability distribution among the child nodes of the split node. The parameters of this distribution are learned from gradient signals that come from the information gain objectives. The probability distribution on the child nodes is conditioned upon the current sample, guiding the sample to traverse the branch with the highest probability.

The parameters of the classification transformations are learned through the training process involving both the classification and information gain objectives when they reside in the split nodes. In the tree structure, the leaf nodes exclusively host

the classification transformations, and they generate logits for utilization with softmax classifiers. The illustration in Figure 3.1 depicts a binary CIGN, highlighting how the information gain-based routers partition a sample dataset into semantically similar subsets.

The general loss function of the CIGN is given as

$$J_C = -\mathbb{E}_{p(x,y)} \left[\log \sum_{j=1}^{|N_l|} p(l_j|x; W_H, W_F) p(y|l_j, x; W_F) \right]. \tag{3.1}$$

In this equation, W_F and W_H represent the parameters for the F and H transformations, respectively. The variable l_j encompasses all paths from the root node to each of the $|N_l|$ leaf nodes. Each path that starts from the root node of the tree and ends in a leaf node is called a "root-to-leaf" expert. Such networks are equivalent to a single, unconditional deep neural network. Such experts share computational units with other root-to-leaf networks in the architecture due to the tree structure. The term $p(y|l_j, x; W_F)$ denotes the posterior probability given the sample x over the path l_j. This formulation of loss aligns with the well-known HMoE loss [11]. Each distinct value of l_j corresponds to a distinct root-to-leaf expert in the Mixture of Experts framework, which is an unconditional neural network that starts in the tree´s root node and ends in one of the N_l leaf nodes. The information gain objective is expressed as

$$J_{IG} = -\lambda_{IG} \sum_{i=1}^{|N_s|} IG_i. \tag{3.2}$$

Here, IG_i represents the local information gain at the ith split node. The parameter λ_{IG} serves as the weight term for the information gain loss, striking a balance with the classification loss. The comprehensive expression for the global objective function of a CIGN can be formulated as

$$L_{CIGN} = J_C + J_{IG} + \lambda_F ||W||^2 + \lambda_H ||W_H||^2, \tag{3.3}$$

where λ_F and λ_H are the coefficients for the usual weight decay regularizers. We separate the weight decay coefficients of the classification parameters and router parameters explicitly since in some settings it could prove useful to treat these two mechanisms distinctly.

We establish a functional $\psi(\cdot)$ across the set P of probability distributions $p(l)$, where $l = 1, \ldots, k$. $p(l)$ assigns a probabilistic weight to each of the N_l root-to-leaf networks in the usual Mixture of Experts system [11, 130].

Mathematically, $\psi : P \mapsto \mathbb{R}^k$. The functional ψ employs one-hot encoding for any given probability distribution, assigning the value of 1 to the largest entry of $p(l)$ and setting all other entries to 0. This behavior reflects the conditional computation approach. A sample always follows the branch of a split node that has been assigned the largest probability by the router until it reaches the leaf node.

According to the functional definition, the selection probability of an expert l_j is

$$p(l_j|x; W_H, W_F) = \prod_{i \in l_j} \psi(p_i(n_i|x)). \tag{3.4}$$

In this context, the variable i represents the indices of the split nodes along the root-leaf path l_j. The term n_i denotes the paths from node i to its children, making $p_i(n_i|x)$ the probability distribution over the child nodes of i, given the sample x. The distribution p_i is generated as the final layer of the router network H_i within node i. By applying a one-hot transformation to these local branching distributions, we acquire a "hard" mixture of experts model. For a given sample x, only a single path $l_j = \hat{l}$ is active, with the probability $p(l_j = \hat{l}|x) = 1$, and $p(l_j = l|x) = 0$ for all $l \neq \hat{l}$. This hard mixture of experts model inherently leads to sparse evaluation of a trained CIGN model: At each split node, the sample is directed down the path with the highest branching probability. This mechanism also enables sparse updates of the classification parameters $w \in W_F$ situated on the path $l_j = \hat{l}$. Given a sample x, this becomes equivalent to updating a convolutional neural network on the path $l_j = \hat{l}$. The gradient of a weight on the root-to-leaf expert $l_j = \hat{l}$ becomes

$$\begin{aligned} \frac{\partial J_C(x)}{\partial w} &= -\frac{\partial \left(\sum_{j=1}^{|N_l|} p(l_j|x; W_H, W_F) p(y|l_j, x; W_F) \right) / \partial w}{\sum_{j=1}^{|N_l|} p(l_j|x; W_H, W_F) p(y|l_j, x; W_F)} \\ &= -p(y|l_j = \hat{l}, x; W_F)^{-1} \frac{\partial p(y|l_j = \hat{l}, x; W_F)}{\partial w}. \end{aligned} \tag{3.5}$$

In practice, the above formulation leads to sparse execution. When a batch of samples is input into the network, one-hot routing probabilities correspond to applying binary masks to the batch members. Each subset of the batch samples gets routed into the computation node that is assigned the largest routing probability. This also allows for sparse updates in the training phase as well. The gradient information is accumulated in the parent node and recursively conveyed up to the root node of the tree. This design prevents an exponential increase in the computation load in the training and inference phases.

3.3. Differentiable Local Information Gain

Decision trees and algorithms that are based on weak decision tree-based classifiers are well established in the machine learning literature [84, 131–134]. When constructing a decision tree, new nodes are added to it based on an impurity minimization loss, which is the basis of the well-known ID3 algorithm [84]. Such algorithms use a greedy search approach in the feature space. An optimal split is searched for that would minimize the used impurity measure, like information gain or the Gini index [135]. The information gain is usually calculated by introducing a splitting hyperplane into the feature space and then counting the data points that fall on the opposing sides of that hyperplane [136]. When treated in the deep neural network domain, such losses are problematic since the counting operation is by its nature, not continuously differentiable [31].

Using the terminology as [31] in the following, a labeled set of samples is defined as $S = (x, v(x))$ where $x \in R^D$ are the sample vectors and $v(.)$ is a function that assigns a sample x to a class label as $y \in [1, \ldots, c]$. The Shannon entropy of the class labels is then defined as $\mathbb{H}[p(y)] = -\sum_{y=1}^{C} p(y) \log p(y)$. The class probability function $p(y)$ just counts the samples with the correct label: $p(y) = \frac{\sum_{i=1}^{N}[v(x_i) - y]}{N}$. $[.]$ is the indicator function, that outputs 1 when [0] and 0 otherwise. [31] defines a weak learner, $h_\psi(x, \theta)$, which is parameterized by θ, by a hyperplane in the feature space.

The explicit definition of $h_\psi(x, \theta)$ can be given as

$$h_\psi(x,\theta) = \begin{cases} 0, \psi(x,\theta) <= 0 \\ 1, \psi(x,\theta) > 1 \end{cases} . \tag{3.6}$$

ψ, in this case, can be treated just as a simple point versus plane comparison in the feature space, as $\psi(x,\theta) = [x^T : 1]\theta = \sum_{i=1}^{D} x_i\theta_i + \theta_{D+1}$, where $[x^T : 1]$ means concatenating 1 as the final dimension to x, bringing it to the $D+1$ dimensional homogeneous coordinates. The sign of this operation will indicate whether the point is in front of the hyperplane or if it is on the other side [135]. Based on this weak hyperplane-based learner, the data is split into two different sets: $S^L = \{x \in S | h_\psi(x,\theta) = 1\}$ and $S^R = \{x \in S | h_\psi(x,\theta) = 0\}$, where L and R are used abbreviations for left and right, considering a binary tree. Consecutively, the number of samples on both sides of the decision hyperplane corresponds to $N^L = \sum_{i=1}^{N} h_\psi(x_i,\theta)$ and $N^R = N - N^L$. After the partition has been made by a weak classifier h, the label distribution probability on the left side, $p(y|S^L)$, is defined as

$$p(y|S^L) = \frac{\sum_{i=1}^{N}[v(x) - y]h_\psi(x_i,\theta)}{N^L} = \frac{\sum_{i=1}^{N}[v(x) - y]h_\psi(x_i,\theta)}{\sum_{i=1}^{N} h_\psi(x_i,\theta)}. \tag{3.7}$$

An equivalent definition for the right side, $p(y|S^R)$ can also be given. The entropy of the label distribution on the left side after the split is then described as

$$\mathbb{H}[p(y|S^L)] = -\frac{\sum_{i=1}^{N}[v(x) - y]h_\psi(x_i,\theta)}{\sum_{i=1}^{N} h_\psi(x_i,\theta)} \log\left(\frac{\sum_{i=1}^{N}[v(x) - y]h_\psi(x_i,\theta)}{\sum_{i=1}^{N} h_\psi(x_i,\theta)}\right). \tag{3.8}$$

The entropy on the right side, $\mathbb{H}[p(y|S^L)]$, is also defined similarly. The information gain that the impurity minimization algorithms try to increase is then defined as the difference between the label distribution entropy without splitting the data and the expected label entropies on both sides of the hyperplane after the split. This is defined as

$$\begin{aligned} IG_{S|h} &= \mathbb{H}[p(y)] - \frac{1}{N}\left(N^L\mathbb{H}[p(y|S^L)] + N^R\mathbb{H}[p(y|S^R)]\right) \\ &= \mathbb{H}[p(y)] - \frac{1}{N}\left(\sum_{i=1}^{N}[v(x_i) - y]h_\psi(x_i,\theta)\right)\log\left(\frac{\sum_{i=1}^{N}[v(x_i) - y]h_\psi(x_i,\theta)}{\sum_{i=1}^{N} h_\psi(x_i,\theta)}\right) \\ &\quad - \frac{1}{N}\left(\sum_{i=1}^{N}[v(x_i) - y]\left(1 - h_\psi(x_i,\theta)\right)\right)\log\left(\frac{\sum_{i=1}^{N}[v(x_i) - y]\left(1 - h_\psi(x_i,\theta)\right)}{\sum_{i=1}^{N}\left(1 - h_\psi(x_i,\theta)\right)}\right). \end{aligned} \tag{3.9}$$

If we think of the information gain term in 3.9 as a function that needs to be maximized for the hyperplane parameters θ in neural networks, then we will need to look at its derivatives. As can be seen, θ is always related to $IG_{S|h}$ via the classifier function h_ψ. Using the chain rule, the derivative of the information gain term with respect to θ is given as

$$\frac{\partial IG_{S|h}}{\partial \theta} = \frac{\partial IG_{S|h}}{\partial h_\psi} \frac{\partial h_\psi}{\partial \theta}. \tag{3.10}$$

For a complete derivation of this derivative expression, we refer the reader to [31]. Since the weak classifier function h_ψ is piecewise constant, the derivative is almost always zero, hence a gradient-based optimization is not directly possible. A natural way to circumvent this and convert this operation into a differentiable function is to use a sigmoid function instead of a hard "zero or one" operation. [31] proposes the following sigmoid classifier

$$h(x, \theta) = \exp\left(\frac{\sum_{i=1}^{D} x_i \theta_i + \theta_{D+1}}{\sigma}\right)^{-1}. \tag{3.11}$$

σ is used as a smoothness parameter that controls the error surface; it usually starts with a large positive value and is annealed down to close to zero. As $\sigma \to 0$, the function approaches a step function.

Inspired by this differentiable relaxation of the information gain objective by Montillo et al. [31], we incorporate differentiable information gain as the objective function for our router networks. We consider a split node i, with l_{-i} representing the split nodes on the path from the root to node i. We let $p(x)$ denote the original data distribution. The data distribution at node i is expressed as

$$p(x|l_{-i}) = \frac{p(x)p(l_{-i}|x)}{\int p(x)p(l_{-i}|x)dx} = \frac{p(x)\prod_{j\in l_{-i}} \psi(p_j(n_j|x))}{\int p(x)\prod_{j\in l_{-i}} \psi(p_j(n_j|x))dx}. \tag{3.12}$$

This conditional data distribution is a reweighting of the samples reaching node i; since the ψ functions will assign 1 only to the samples that reach the current node. For this

partial data that reaches node i, the following joint distribution is then defined

$$p_i(x, y, n_i) = p_i(x)p(y|x)p_i(n_i|x). \tag{3.13}$$

Here, $p_i(x) := p(x|l_{-i})$ is the conditional data distribution as defined in 3.12 and $p(y|x) := [c(x) - y]$ where $c(x)$ gives the ground truth class label for x. $p_i(n_i|x)$ defines a probability distribution on the child nodes that are connected to the current node i. We define it with a softmax function as

$$p_i(n_i = k|x) = \frac{\exp\left(\frac{w_k^T h^i(x) + b_k}{\tau}\right)}{\sum_{j=1}^{K} \exp\left(\frac{w_j^T h^i(x) + b_j}{\tau}\right)}. \tag{3.14}$$

The output feature of the router network at node i is denoted as $h^i(x)$, typically derived from a fully connected layer. The decision hyperplane for the j-th path is represented by (w_j, b_j). The parameter τ is employed to smooth the distribution; as τ approaches infinity, $p_i(n_i|x)$ approaches the uniform distribution, and as τ approaches 0, it converges to the argmax function [33]. Initialization with a large value and annealing during training is a common practice to prevent the routers from being excessively confident in their initial decisions [31].

Utilizing the joint distribution (3.13), we characterize the information gain objective as the disparity between the entropy of the class distribution at node i and the expected entropy of the conditional class distributions following branching, as

$$IG_i = \mathbb{H}\left[p_i(y)\right] - \mathbb{E}_{p_i(n_i)}\left[\mathbb{H}\left[p_i(y|n_i)\right]\right]. \tag{3.15}$$

We define the entropy as: $\mathbb{H}\left[p(x)\right] = -\sum_x p(x) \log p(x)$. Using the Kullback-Leibler divergence, we can define the same quantity as the mutual information between the class label y and the branch that is being taken, indicated by n_i

$$IG_i = D_{KL}(p_i(y, n_i)||p_i(y)p_i(n_i)). \tag{3.16}$$

The mutual information perspective of this router objective function makes intuitive sense: We encourage particular class labels to convey information about a particular child node, such that each class follows certain paths in the tree-shaped network. This leads to the automatic generation of coarse class groups, in turn.

The derivative of the information gain with respect to network parameters is

$$\frac{\partial IG}{\partial W} = \sum_{n_i=1}^{K} \sum_{y=1}^{C} \mathbb{E}\left[p(y|x) \frac{\partial p_i(n_i|x)}{\partial W}\right] \log \mathbb{E}\left[p(y|x) p_i(n_i|x)\right] - \sum_{n_i=1}^{K} \mathbb{E}\left[\frac{\partial p_i(n_i|x)}{\partial W}\right] \log \mathbb{E}\left[p_i(n_i|x)\right], \tag{3.17}$$

where the expectations are over $p_i(x)$. When used in a minibatch setting, this derivative has a biased estimator of the complete data version. In practice, we did not encounter a problem regarding that. The information-gain objectives converge with the rest of the network regularly.

It should be noted that we use information gain objectives at every split node of a CIGN model in a unified loss function. The CIGN structure is determined a priori before training the model on a dataset. This is in contrast to the approach proposed in [31], where the greedy optimization of the impurity measure is changed with gradient descent for tree induction. The addition of new nodes to a tree continues until a stopping condition like maximum depth or sample count has been reached. Only a single loss function is optimized at the same time.

3.4. Load Balancing

The information gain objective within a router has the potential to reach a local minimum, leading to a scenario where the probability of directing samples to a specific child node is significantly higher than others. In situations where a majority of the samples are processed through a root-to-leaf network, convergence may occur at a slower pace, possibly resulting in overfitting for root-to-leaf paths trained with a limited subset of the data. While the CIGN framework is not a generative structure, this problem is similar to the "mode collapse" phenomenon encountered in the domain of Generative Adversarial Networks (GANs) [137–139]. A similar problem has been observed in a similar Mixture of Experts-based deep neural network architecture in [74]. In that work, it has been claimed that a Viterbi-EM scheme, where during the

maximization phase only the most likely path is selected in the MoE framework, is inherently sufficient for preventing a "mode collapse" scenario. Going one step further, we analyze the mathematical properties of the information gain objective function that we use for training our routers and design a modification that encourages a more balanced distribution of samples to the child nodes in a split node of the CIGN tree.

We first observe that the information gain objective can be decomposed as

$$IG_i = \mathbb{H}\left[p_i(y)\right] + \mathbb{H}\left[p_i(n_i)\right] - \mathbb{H}\left[p_i(y, n_i)\right]. \tag{3.18}$$

It is apparent that the term $\mathbb{H}\left[p_i(n_i)\right]$ is the entropy of the distribution of samples routed into the child nodes. A high value for this term indicates a more balanced distribution of the samples into the child nodes. Considering this, we define a hyper-parameter, that is $\lambda_{balance} > 1$, which we add into the equation 3.18 as a coefficient for the entropy term $\mathbb{H}\left[p_i(n_i)\right]$. A higher value for this coefficient will reward settings with a close number of samples routed into all possible child nodes; hence, the mode collapse scenario is avoided. With that addition, the objective becomes

$$IG_i^{balanced} = \mathbb{H}\left[p_i(y)\right] + \lambda_{balance}\mathbb{H}\left[p_i(n_i)\right] - \mathbb{H}\left[p_i(y, n_i)\right]. \tag{3.19}$$

The coefficient $\lambda_{balance}$ acts as a load-balancing regularizer and facilitates a more stable training of the overall structure.

3.5. Routing Errors - During the Training

A hierarchical structure in the form of a tree is susceptible to routing errors, potentially causing misclassification if a sample is directed to a path that is not specialized for its class. One strategy to address this problem involves enhancing the accuracy of the routing process, while another approach aims to boost the classification performance of the experts for samples that are misrouted to them. We establish a threshold, denoted as ρ, such that if $p_i(n_i = k|x) \geq \rho$, the sample x is directed to the k-th child. The threshold ρ takes values in the interval $[0, 1/K]$, initially starting at 0 and gradually increasing to $1/K$ as the training proceeds. This allows all root-to-leaf experts to process the entire dataset initially. As ρ approaches $1/K$, only samples as-

sociated with the focused classes and those semantically similar to them, as perceived by the current root-to-leaf expert, surpass the threshold.

Here, the aim is to give the expert an idea about the semantically similar samples with their focus classes and prevent confusing such samples during the inference. This can be thought of as a form of specific Transfer Learning for the CIGN architecture, where the expert networks are trained with a broad range of classes first and then fine-tuned on a specific (semantically similar) subset.

The information-gain-based routers are designed to work with a semantically filtered subset of the data that reaches them. When we use the aforementioned thresholding method and route samples into multiple possible paths during the training phase, the semantic distinction between class groups becomes vague. While this helps the classifiers in the leaf nodes generalize better for misrouted samples, it may cause difficulties for information gain-based routers for convergence. To address this issue, we maintain and update two versions of a data minibatch during the training phase: D_ρ constructed using thresholding and $D_{\frac{1}{K}}$, which strictly results from directing each sample to the most likely route so that it is always $D_\rho \supseteq D_{\frac{1}{K}}$. For the forward pass and gradient calculation of the information gain objectives, we utilize $D_{\frac{1}{K}}$, ensuring that the correct semantically filtered version of the data reaches routers, enabling the information gain objectives to function as intended.

Furthermore, for batch normalization operations, we compute the running average mean and variance based on $D_{\frac{1}{K}}$. We apply normalization to D_ρ by using the mean and variance calculated from $D_{\frac{1}{K}}$. In practical implementation, we do not explicitly store two distinct minibatches; instead, we maintain sample indices corresponding to D_ρ and $D_{\frac{1}{K}}$. These indices are updated using thresholded and non-thresholded binary masks after each tree split.

3.6. Sparse Mixture of Experts with CIGN

3.6.1. Motivation

CIGN effectively clusters semantically similar samples, and at the leaf nodes, the classification occurs between hard-to-distinguish classes, which have low inter-class variance. Illustrated in Figure 3.2 is an example binary model of depth two, which shows the actual routing and classification behaviors of an actual, trained CIGN model. The top row displays the routing behaviors of different test images from the CIFAR-100 dataset. Rows two and three depict the corresponding routing probabilities. Notably, the left-side images are confidently routed to the indicated leaf nodes in the bottom row. Consequently, the correct classification decisions in row four are achieved for samples belonging to classes #81 (train) and #53 (orange).

However, certain challenging samples are misrouted to experts lacking expertise in these classes, leading to classification errors. This scenario is exemplified by the samples on the right-hand side of Figure 3.2. Samples corresponding to classes #57 (pear) and #44 (lizard) pose difficulty even for human observers due to their visual vagueness. The routing probabilities are low, resulting in high routing entropies, and both have been misclassified in the greedy information gain-based routing scheme, in which each sample is routed only into the most likely child node that is being determined by the routers. For these specific samples, routing them into more than one leaf node and obtaining a consensus result from multiple experts may aid in recovering the correct classification.

Based on our observations of routing behaviors and classification performance, we hypothesize that when a sample is semantically similar to multiple data partitions in the leaf nodes, causing uncertain routing, it is preferable to let the ensemble of experts at these leaves classify the sample. This led to the design of the Sparse Mixture of Experts mode for CIGN inference. We call our CIGN models that are using this inference approach as “sMoE-CIGN”s.

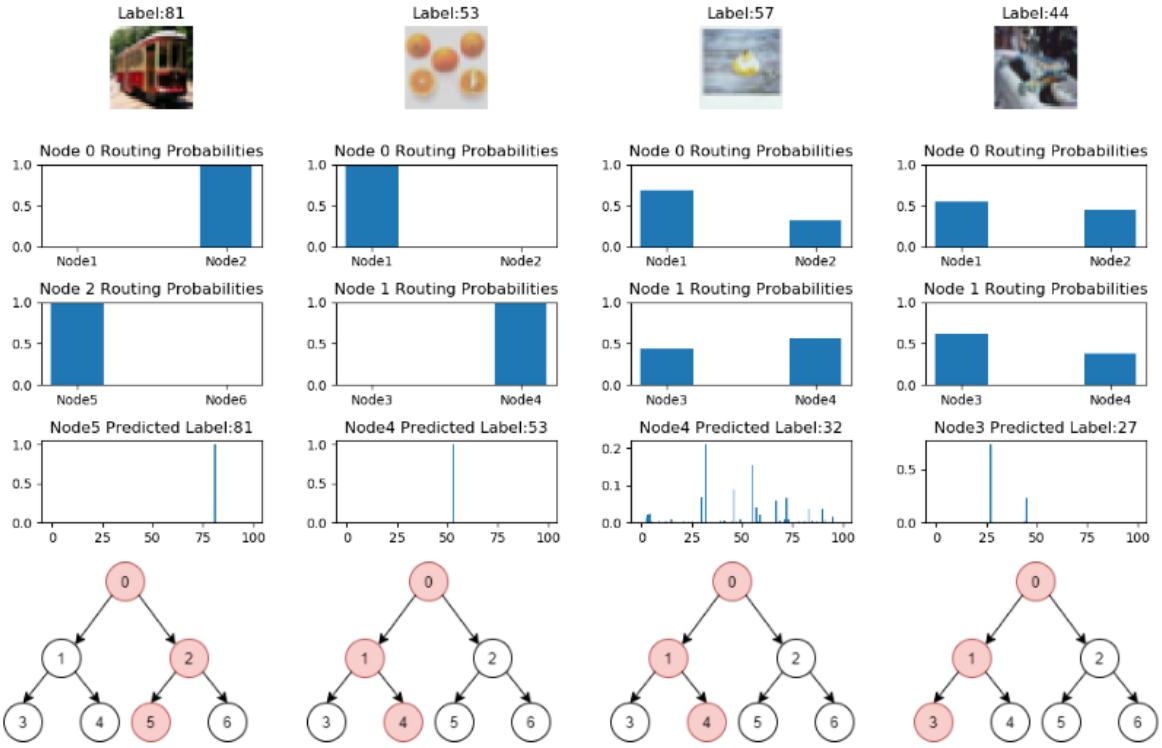


Figure 3.2. Illustrated are the routing patterns of various CIFAR-100 test images using our CIGN model. The first row shows the images with labels. The next two rows display routing probabilities at the first and second CIGN levels. The fourth row shows the leaf node posterior probability. The final row presents the root-to-leaf nodes in the CIGN structure for each image.

3.6.2. Sparse Mixture of Experts Model

To address the classification of challenging samples that may be misrouted, we devise a Sparse Mixture of Experts inference approach built upon trained CIGNs. During inference, if the routing probability at a decision node i exceeds a threshold ρ_i (i.e., $p_i(n_i = k|x) > \rho_i$), the sample is directed to the k-th child. This way, the sample reaches $|N_l'|$ out of a total of $|N_l|$ leaf nodes. Utilizing the posterior probabilities obtained at the assessed leaf nodes, we compute the final classification probability as

$$p(y|x) = \frac{1}{|N_l'|} \sum_{l \in N_l'} p(y|x, l). \tag{3.20}$$

This strategy transforms CIGN into a collection of root-to-leaf expert networks, specifically for samples considered challenging to classify within a particular expert's focus set. This inference approach involves a trade-off between computational efficiency and accuracy. We refer to this modified inference model within the CIGN structure as the Sparse Mixture of Experts.

3.6.3. Bayesian Optimization for Optimal Threshold Learning

We aim to determine optimal thresholds $\{\rho_{i,j}\}_{j=1}^{K}$ for each decision node i and its K children, with the goal of directing challenging samples to multiple leaves. Routing more samples to more than one path during inference may make the model work better, but it also means that multiple paths have to be run more often, which means that more multiplication and accumulation (MAC) operations have to be done. To tackle this, we formulate the objective function for the optimal threshold selection as

$$L_{opt}(\rho) = \lambda L_{class}(\rho) + (\lambda - 1)\mathbb{E}_{p(x)}[L_{MAC}(x, \rho)]. \tag{3.21}$$

The vector ρ encompasses all distinct thresholds specific to the given CIGN structure. The hyperparameter $\lambda \in [0, 1]$ signifies the relative emphasis on model performance compared to computational load. L_{class} denotes the model performance (accuracy) of an already trained CIGN model with the specified ρ values. $L_{MAC}(x, \rho)$ is defined as

$$L_{MAC}(x, \rho) = \frac{MAC(x, \rho)}{MAC(\hat{l})}. \tag{3.22}$$

This ratio quantifies the rise in the number of multiplication and accumulation operations (MAC) for the particular sample, given the threshold vector ρ. The expectation $\mathbb{E}_{p(x)}[L_{MAC}(x, \rho)]$ is computed across the entire validation set. We use the method described in [140] to measure the MAC loads caused by the convolution and fully connected layers, which make up most of the computational work in deep convolutional neural networks.

To optimize $L_{opt}(\rho)$ with respect to ρ, traditional gradient-based methods are unsuitable due to the non-differentiability of components, $L_{class}(p)$ and $\mathbb{E}_{p(x)}[L_{MAC}(x, \rho)]$, with respect to ρ. Consequently, we resort to black-box optimization methods, specif-

ically leveraging the Bayesian optimization framework [141]. This framework employs an approximate surrogate model, $p(x)$, mimicking the actual black-box function $f(x)$. The model $p(x)$ encapsulates a prior belief about the function's structure. The second component is the "acquisition" function, which, based on the current state of the surrogate model $p(x_{T+1}|x_{1:T})$ and the sampled data $x_{1:T}$ so far, proposes a new point x_{T+1}. The surrogate model is then updated with the newly sampled data. This iterative process concludes either when the samples meet a specific criterion or a predetermined number of iterations have transpired.

A commonly employed option for surrogate models involves the Gaussian Process [141, 142]. This process establishes a joint Gaussian distribution over a set of points $y(x_1), \ldots, y(x_N)$, where $x \in \mathbb{R}^d$; expressed as $p(y_{1:N}|x_{1:N}) = N(y_{1:N}|\mu(x_{1:N}), K)$ [130]. The matrix K is a Gram matrix with entries $K_{ij} = k_\theta(x_i, x_j)$, denoting a kernel function with trainable hyperparameters θ. Typically, $\mu(x_{1:N})$ is considered as zero, and a noise term ϵ_i is introduced, such that $t_i = y_i + \epsilon_i$ with $\epsilon_i \sim N(\epsilon|0, \beta^{-1})$, to accommodate the inherent noise in the data generation process, resulting in target values t_i. The marginal distribution of target values $t_{1:N}$ given a data $x_{1:N}$ can then be expressed as

$$p(t_{1:N}|x_{1:N}) = \int p(t_{1:N}|y_{1:N})p(y_{1:N}|x_{1:N})dy_{1:N}. \tag{3.23}$$

Upon the arrival of a new sample x_{N+1}, the predictive distribution $p(t_{N+1}|t_{1:N}, x_{1:N+1})$ can be derived in a closed analytical form [130]. This iterative update-predict scheme also entails optimizing the hyperparameters of the selected kernel function $k_\theta(x, x')$ through the maximum likelihood principle [130].

The cyclic process of updating and predicting within the Gaussian Process renders it a suitable surrogate model for the underlying $f(x)$ function in Bayesian Optimization. In our experiments, we employ a Gaussian Process with a Matern kernel [143]. Additionally, an acquisition function is necessary for sampling new x points, and we opt for the expected improvement (EI), defined as

$$EI(x) = \int_{f(x^*)}^{+\infty} p(f|x, f_{1:N}, x_{1:N})(f - f(x^*))df. \tag{3.24}$$

In 3.24, x^* represents the best sample observed to date, satisfying $f(x^*) \geq f(x)$ for all $x \in x_{1:N}$. The term $p(f|x, f_{1:N}, x_{1:N})$ refers to the present surrogate model, which, in our case, is the Gaussian Process predictive posterior for the prospective candidate x. The expected improvement measures the surrogate model's ability to produce values better than $f(x^)$, given x as the new value. An x value that increases the probability mass above $f(x^)$ results in a higher expected improvement score. Our Bayesian Optimization framework for threshold optimization in CIGNs is shown in Algorithm 1.

Algorithm 1: Bayesian Optimization For Routing Thresholds

Result: Optimal thresholds ρ^*

Determine a validation set D;

$N := TopologicalSort(ParentNodes)$;

// Sample n sets of random thresholds initially;

while $i = 1, \ldots, n$ **do**

  $\rho_i = []$;

  for *node in* N **do**

    $c := node.childCount$;

    $\rho_i^{node} = \left[\rho_{i,1}^{node}, \ldots, \rho_{i,c}^{node}\right] \sim Uniform\left(0, \frac{1}{c}\right)$;

    $\rho_i = \rho_i \cup \rho_i^{node}$;

    $l_i = \mathbb{E}_{(x,y)\sim D}[\lambda L_{classification}(x, y, \rho_i) + (\lambda - 1)L_{MAC}(x, \rho_i)]$;

  end

end

Set $S = \{(\rho_1, l_1), \ldots, (\rho_n, l_n)\}$;

while $t = 1, \ldots, T$ **do**

  Fit Gaussian Process regressor using known thresholds and results so far:

  $p(l|\rho, S)$;

  $\rho_t = argmax_\rho EI(\rho, p(l|\rho, S))$;

  $l_t = \mathbb{E}_{(x,y)\sim D}[\lambda L_{classification}(x, y, \rho_t) + (\lambda - 1)L_{MAC}(x, \rho_t)]$;

  $S = S \cup (\rho_t, l_y)$;

end

$\rho^* = argmax_{\rho,(\rho,l)\in S}[l]$;

3.7. Experiments

We assessed the effectiveness of our models across four distinct image classification datasets: MNIST [144], Fashion-MNIST [29], CIFAR-100 [30] and USPS [145].

3.7.1. Experiment Design

Our experimental approach involves taking a conventional, fully parameterized CNN architecture without conditional execution (the vanilla form) and subsequently transforming it into a CIGN. This conversion is achieved by introducing a tree structure with relevant routing mechanisms in the decision nodes, guided by information-gain objectives. In our CIGN structures, each root-to-leaf path represents a thinned version of the original baseline, implying a reduction in the number of feature maps, filters in the convolutional layers, and a decrease in hidden layer dimensions in the fully connected layers. To ensure fair comparisons with the unconditional baselines, we maintain identical layer compositions in the CIGN models, avoiding any surplus or reduced number of layers compared to the thicker baselines.

As part of our experiments, we change a standard CNN architecture that is fully parametrized and unconditional into a CIGN by adding a tree structure with relevant routing mechanisms in the decision nodes. These are driven by information-gain losses. Each root-to-leaf path in our CIGN structures constitutes a thinned version of the thicker baseline, implying that it has fewer feature maps and filters in the convolutional layers and fewer hidden layer dimensions in the fully connected layers. For fair comparisons with the unconditional baselines, we leave the layer compositions the same as in the thicker baselines, meaning no surplus or reduced number of layers exists in the generated CIGN models.

When analyzing a CIGN model, we use the following versions of the CIGN and the baselines in all of our experiments:

(i) Thick Baseline: We initially compared the CIGN models with the original thicker baseline, characterized by more parameters and MAC operations per sample.

(ii) Thin Baseline: The thin baseline corresponds to a root-to-leaf expert in our CIGN models, with nearly equivalent MAC operations per sample on average.

(iii) CIGN Model - No Annealing: In this variant, a sample is routed solely to the most likely path. No annealing strategy is used during the training of the model to facilitate the generalization of the classifiers.

(iv) CIGN Model - Annealing: This model involves temperature-annealing during training. Initially, all samples are routed to each possible path, followed by a focus on subsets of the data for each root-to-leaf path as described in Section 3.5. During inference, only greedy information gain (IG)-based routing is employed.

(v) sMoE-CIGN - Bayesian Optimized Routing: This variant allows a sample to be routed to more than one expert based on the thresholds learned through Bayesian optimization. For learning, we evenly split the test set into two halves. Bayesian optimization is applied using one half as a validation set, and accuracy is measured with the Bayesian-optimized thresholds in the other half. This process is repeated with 10 different splits, and averages are reported. When n different training runs are done for a specific dataset, the results of the Bayesian optimization are the averages of $10n$ different calculations, where for each separate training procedure, 10 Bayesian optimizations are calculated.

(vi) CIGN Model - Random Routing: We use this version of the CIGN to show the effectiveness of the information gain-based routing over training and evaluation with random routing of the samples. In this case, routing is performed based on uniform sampling over the child nodes at every split node. We expect the information gain-based routing to perform better compared to this baseline since there is no semantically meaningful partitioning of the data.

(vii) CIGN Model - All paths: This variant is a CIGN model that is trained by routing every sample to every possible path in every decision node of the tree. This variant does not include any conditionality in the training phase, but the information gain losses are still trained as they were receiving a subset of the data according to the routing decision of the ancestor nodes, with the masking method we mentioned

in the section 3.5. The training procedure for this case can be considered one with probability thresholds during the training; however, with thresholds set to 0 during the whole training, each sample is routed into every possible path from the start to the end. With this variant, we want to assess the value of the information gain-based routing mechanism from the standpoint of the various root-to-leaf experts who have received training from samples of semantically related classes. Intuitively, classifiers of semantically similar class groups should perform better since they work on a relatively easy problem compared to classifying all the data indiscriminately.

3.7.2. MNIST Experiments

MNIST [144] comprises 10 classes representing each digit, with 60,000 training and 10,000 test samples. Our baseline network is a LeNet model with ReLUs instead of sigmoid nonlinearities. The thick LeNet baseline involves two convolutional layers with 20 and 50 feature maps, respectively, using 5×5 kernel sizes. A 500-dimensional fully connected layer and the softmax layer follow these.

The thin version maintains the same overall architecture as the standard model but modifies specific parameters to reduce complexity. It incorporates 15 feature maps in the second convolutional layer, which is fewer than in the standard version, and has a final fully connected layer with a dimension of 25. Both versions of the model include routers that feature a Global Average Pooling layer after each convolutional block. This is followed by a 20-dimensional fully connected layer, Batch Normalization, and the calculation of Information Gain, which helps in making more informed routing decisions. Additionally, it's important to note that no data augmentation techniques were applied to the MNIST dataset in our experiments. This means that the training and testing data were used in their original forms, without any modifications such as rotations, scaling, or flips that are often used to artificially increase the size and variability of the dataset. Figure 3.3 shows a detailed diagram for the thin and thick baselines. We employ a CIGN model in the form of a binary tree with a depth of two (a

[2, 2] CIGN). The thin baseline serves as a root-to-leaf expert network in our structure. Additionally, we present results for different CIGN models as outlined in Section 3.7.1.

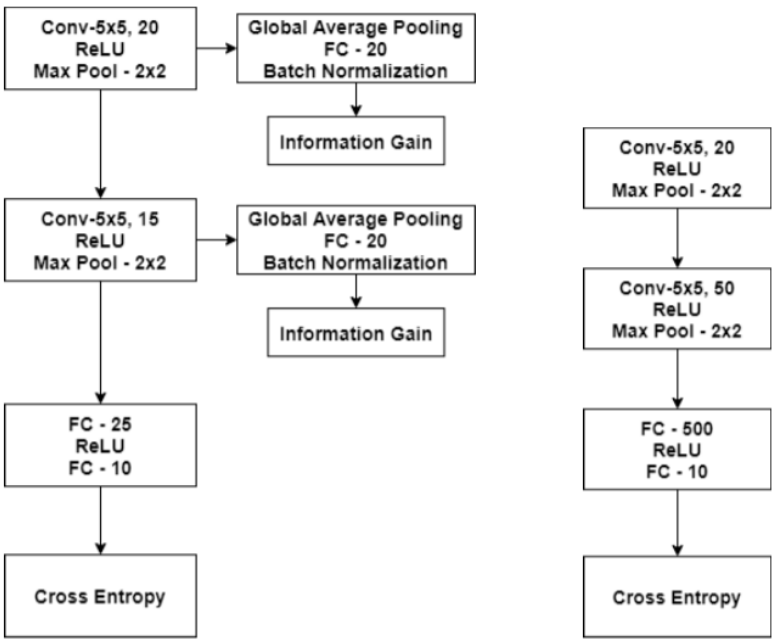


Figure 3.3. LeNet architectures for MNIST. The left side depicts the thin baseline with root-to-leaf networks and IG-based routing, while the right side shows the thick baseline.

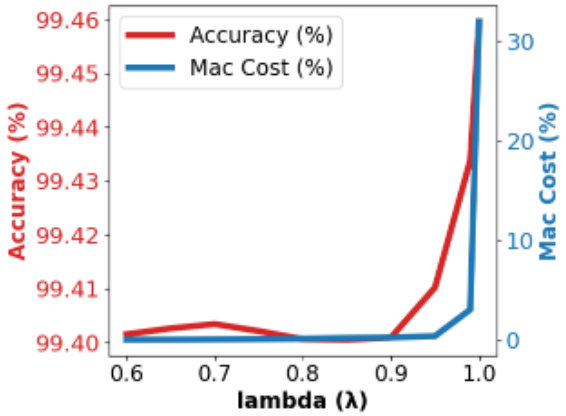
The outcomes of the MNIST experiments are detailed in Table 3.1. All results represent the average of 5 independent training runs with consistent hyperparameters. The thick baseline achieves an accuracy of 99.25% with an average of approximately 4.018×10^6 MAC operations, aligning with prior findings [15, 144]. Optimizing the regularization hyperparameter allows the thin baseline to approach the thick version's accuracy, reaching 99.20%. This is expected as MNIST is considered a relatively straightforward dataset [26]. The CIGN model, utilizing random routing, attains an accuracy of 99.28%, surpassing the thick baseline. This can be attributed to the ensemble behavior of the CIGN's multi-exit architecture, where random partitioning mimics the training of an ensemble with random data subsets, reducing variance and overfitting, similar to a bagging scenario [146, 147].

Table 3.1. MNIST test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers.

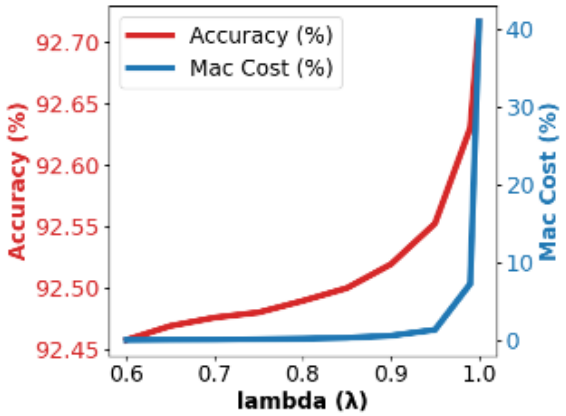
Model	**Mean Acc.**	**Avg # MAC**	**# Params**
LeNet Thin Baseline(*)	99.20%	1.057×10^6	26695
LeNet Thick Baseline	99.25%	4.018×10^6	1256080
CIGN-Random Routing	99.28%	1.057×10^6	92276
CIGN-All Paths	99.27%	1.057×10^6	92276
CIGN-Greedy IG Routing - No Annealing	99.32%	1.057×10^6	92276
CIGN-Greedy IG Routing + Annealing	99.37%	1.057×10^6	92276
sMoE-CIGN Optimized Routing-1 $\lambda = 1.0$	99.46%	1.394×10^6	92276
sMoE-CIGN Optimized Routing-2 $\lambda = 0.99$	99.43%	1.088×10^6	92276
sMoE-CIGN Optimized Routing-3 $\lambda = 0.95$	99.41%	1.060×10^6	92276
Adaptive Neural Trees [108]	99.36%	approx. 5.0×10^6	100596
Deep Neural Decision Forest [15]	99.30%	approx. 4.0×10^6	544600

The alternative baseline, directing all samples to every path during training but applying information-gain-based routing, achieves comparable accuracy to random routing. However, without semantically similar data partitioning, the root-to-leaf experts struggle with the more complex task of distinguishing all classes. The CIGN model with information gain-based routing (applied during both training and testing) without annealing achieves 99.32% accuracy. Enabling annealing of the routing thresholds during training further boosts accuracy to 99.37%, indicating that exposing each expert to the entire dataset initially and then focusing on semantic partitions enhances generalization.

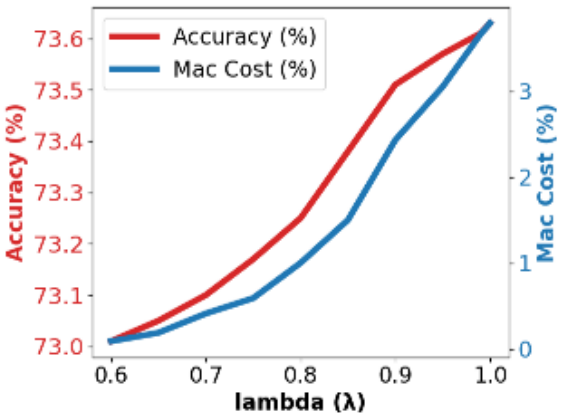
Enhanced generalization is achievable through the sMoE model. By setting $\lambda = 1.0$ and disregarding the additional computational load in Equation 3.21, we achieve an accuracy of 99.46% with an extra 32% average MAC operations per sample compared to the thin baseline. Figure 3.4 illustrates a sharp decline in extra computation between $\lambda = 1.0$ and $\lambda = 0.99$. As λ decreases to ≤ 0.95, the extra computation load becomes negligible, yet accuracy remains consistently above 99.40%. At $\lambda = 0.95$, the accuracy stands at 99.41%, representing a favorable trade-off point where extra computation is still negligible. This underscores the effectiveness of our sMoE-CIGN inference approach. Significance testing using the Bootstrap t-test [148] reveals that all CIGN versions, except the No Annealing, are significantly superior to all baselines, with very small or zero p-values at a 0.05 significance level. The No-Annealing version is only significantly better compared to the thin baseline.



(a) MNIST



(b) Fashion MNIST



(c) CIFAR-100

Figure 3.4. The performance of bayesian optimization in terms of extra average MAC per sample against different λ values (the balance parameter in bayesian optimization) for the MNIST, Fashion-MNIST, and CIFAR-100 datasets.

We also compare our MNIST results with two recent approaches that employ decision tree-neural network hybrids. The Deep Neural Decision Forest algorithm by [15] utilizes a LeNet-5 backbone and replaces the softmax layer with a differentiable decision forest. They achieve an accuracy of 99.30% with the LeNet-5 backbone, using approximately four times the average MAC operations and more than five times the parameters compared to our sMoE-CIGN model. On the other hand, the Adaptive Neural Trees algorithm presented in [108] reports a higher accuracy of 99.36% on MNIST compared to [15], but at an inference cost of approximately 5.0×10^6 MAC operations on average. It employs nearly $100K$ trainable parameters, which is comparable to our sMoE-CIGN model. In general, our sMoE-CIGN model outperforms both algorithms in terms of accuracy, with significantly lower computational costs and a reduced number of trainable parameters.

3.7.3. Fashion-MNIST Experiments

Fashion-MNIST, designed to be more challenging than MNIST, shares the same number of classes and data size with it. Each image is 28×28 pixels, and the dataset consists of 60,000 training and 10,000 test samples, featuring 10 clothing categories: T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle Boot.

In our Fashion-MNIST experiments, we employed a LeNet-style architecture, comprising 3 convolutional and 2 fully connected layers. For the thick baseline, the convolutional layers used 32, 64, and 128 feature maps with 5×5, 5×5, and 1×1 kernels, respectively. The fully connected layers had dimensions of 1024 and 512. Dropout was applied to the last two fully connected layers for regularization. In the thin baseline and CIGN models, we reduced the feature map count to 32 in each convolutional block, and the fully connected layer dimensions were downsized to 128 and 64, respectively. Routers incorporated a Global Average Pooling layer after the first and second convolutional blocks, followed by a 128-dimensional fully connected layer, batch normalization, and information gain calculation. The detailed layer composition for the thin and thick baselines can be seen in Figure 3.5. Training hyperparameters

for both baselines and CIGN models followed the procedures outlined in [26]. For every model we report, the optimal drop probability is found with grid search in the interval $[0.0, 0.5]$ with a grid size of 0.05. We use binary trees of depth 2 for the CIGN models ($[2, 2]$). The root-to-leaf networks correspond to a thinned version of the thick baseline. During the training, no augmentation has been used on the training set. We use a SGD optimizer with a momentum of 0.9. All models are trained for 100 epochs, using a minibatch size of 125. The learning rate starts as 0.01. It is halved at 15000. and 30000. iterations and multiplied with 0.1 at 40000. iteration. Following [26] we use $\lambda_{balance} = 5$ for balancing the distribution of the samples in the decision nodes. We use the same annealing processes both for τ and ρ as in the MNIST experiments.

Table 3.2. Fashion-MNIST test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers.

Model	**Mean Acc.**	**Avg # MAC**	**# Params**
Fashion-MNIST Thin Baseline(*)	91.65%	3.145×10^6	102090
Fashion-MNIST Thick Baseline	92.25%	8.609×10^6	2688522
CIGN-Random Routing	91.90%	3.154×10^6	368814
CIGN-All Paths	91.97%	3.154×10^6	368814
CIGN-Greedy IG Routing - No Annealing	92.27%	3.154×10^6	368814
CIGN-Greedy IG Routing + Annealing	92.37%	3.154×10^6	368814
sMoE-CIGN Optimized Routing-1 $\lambda = 1.0$	92.72%	4.448×10^6	368814
sMoE-CIGN Optimized Routing-2 $\lambda = 0.99$	92.63%	3.387×10^6	368814
sMoE-CIGN Optimized Routing-3 $\lambda = 0.9$	92.52%	3.165×10^6	368814

The outcomes of the Fashion-MNIST experiments are detailed in Table 3.2. These results are derived from the average of five independent training runs utilizing the specified hyperparameters.

The thick baseline network, when employed for Fashion-MNIST, achieves an accuracy of 92.25% with an average MAC operation count of approximately 8.609×10^6. On the other hand, the thin baseline, equivalent to a root-to-leaf network, attains an accuracy of 91.65% with an average MAC count of 3.145×10^6. Random routing and all-paths approaches yield accuracies of 91.90% and 91.97%, respectively, demonstrating that information gain-based routing significantly outperforms these alternatives.

When utilizing information gain-based routing without annealing during training, an accuracy of 92.27% is achieved. The average MAC cost is 3.154×10^6, where the routers introduce an additional computation cost of approximately 0.3% compared to the thin baseline. Enabling annealing during training further improves accuracy to 92.37% without incurring any extra computation or parameter cost during inference.

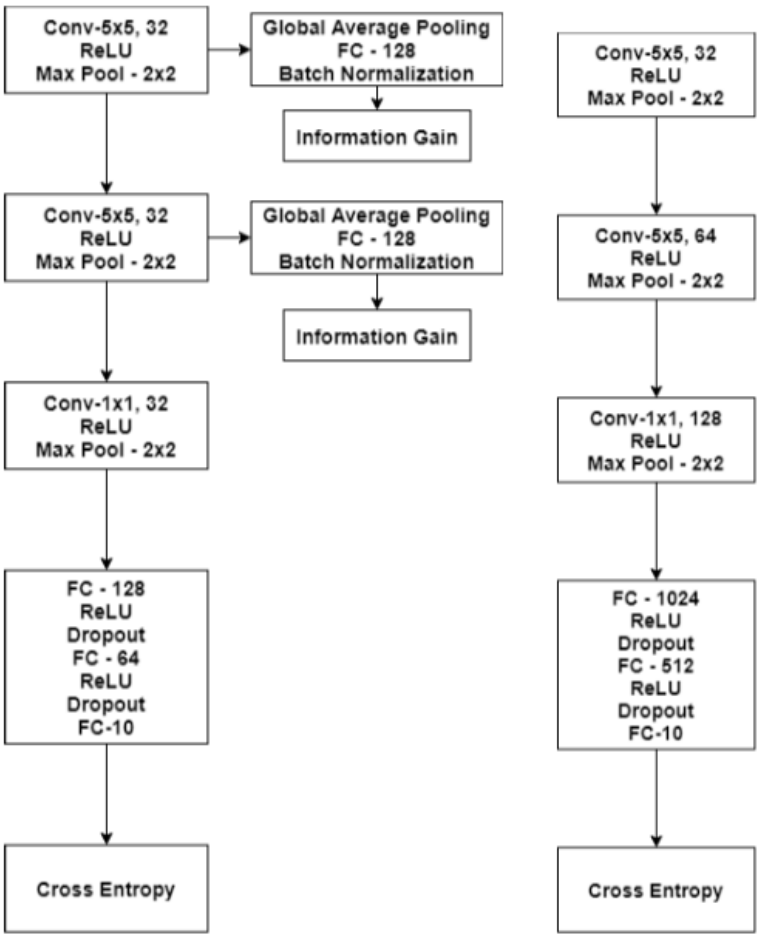


Figure 3.5. The network architectures for the Fashion-MNIST dataset. The left side shows the thin baseline (root-to-leaf network) with IG-based routing, while the thick baseline is on the right.

Utilizing Bayesian Optimization to determine optimal routing thresholds, we achieve an accuracy of 92.72% at $\lambda = 1.0$, accompanied by an additional 41% MAC cost on average. Figure 3.4 illustrates the impact of varying λ values. Similar to MNIST, a

notable reduction in the average MAC is observed within the range $\lambda \in [1.0, 0.99]$. At $\lambda = 0.99$, the accuracy reaches 92.63%, with the extra MAC cost decreasing to 7.3%.

The extra MAC cost becomes nearly negligible at $\lambda = 0.9$, amounting to 0.64%, while maintaining a mean accuracy of 92.52%. These findings demonstrate that sMoE-CIGN delivers an accuracy boost with a customizable additional inference cost. Applying the Bootstrap t-test to the experiment results reveals that all sMoE-CIGN versions outperform the baselines, with very small or zero p-values (utilizing a 0.05 significance level). In the absence of sMoE, the differences in performance between the thick baseline and CIGN models are not significant.

3.7.4. CIFAR-100 Experiments

The CIFAR-100 dataset comprises 50,000 training and 10,000 test images distributed across 100 distinct classes, organized within a two-level hierarchy [30]. Each RGB image has dimensions of 32×32 pixels. The first level of the hierarchy consists of "superclasses", which represent broad semantic class groupings such as aquatic mammals, fish, flowers, vehicles, insects, reptiles, and trees. There are 20 of these coarsely grouped classes. For instance, the "aquatic mammals" group encompasses specific class labels for beaver, dolphin, otter, seal, and whale.

To classify the CIFAR-100 data, we employ a ResNet architecture with 145 layers as our baseline, following the design in [39]. Our implementation mirrors the approach in [39], incorporating three different layer blocks with feature map sizes of $32 \times 32, 16 \times 16, 8 \times 8$, respectively. Within each feature map size, we employ 16 bottleneck blocks, each consisting of three layers. The bottleneck blocks include a 1×1 reduction layer decreasing feature maps from $4n$ to n, a 3×3 layer applying a nonlinear transformation, and a 1×1 expansion layer restoring the feature maps from n to $4n$. The bottleneck layers reduce MAC operations. We apply Batch Normalization before each convolutional layer and use Leaky ReLU (0.1 leak rate) as the nonlinearity.

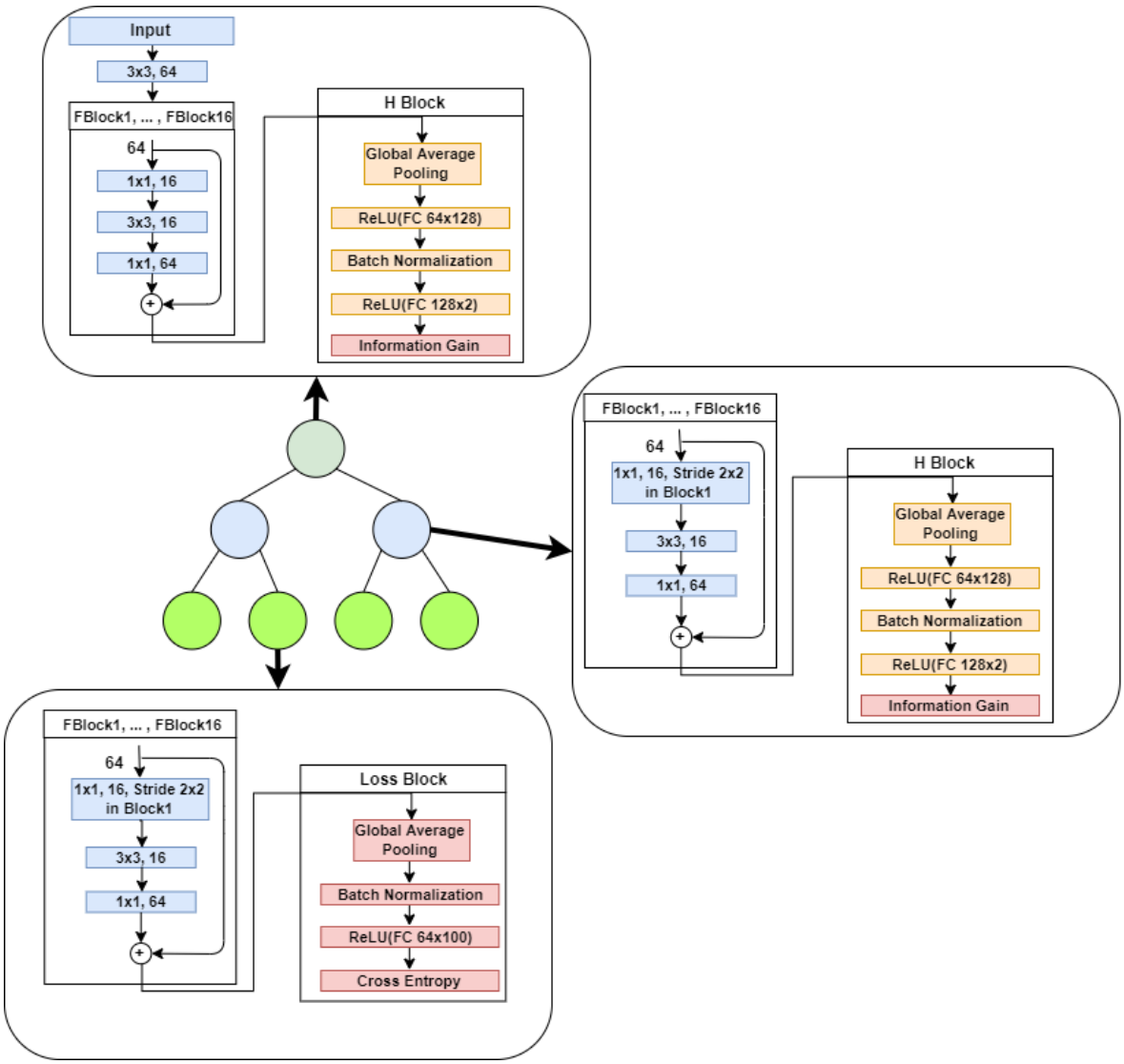


Figure 3.6. We adopt the ResNet-145 architecture as our baseline for the [2,2] CIGN. Each node at the same depth in the CIGN incorporates an identical sub-network.

As an unconditional baseline model, we employ two variations of the ResNet-145 model. The thin model consists of 16 bottleneck blocks in the conv2x, conv3x, and conv4x sections, with respective feature map sizes of $32 \times 32, 16 \times 16, 8 \times 8$. This thin model corresponds to a root-to-leaf network in our CIGN models and incorporates 64 feature maps in each conv2x, conv3x, and conv4x part. In contrast, the thick model comprises 64, 116, and 120 feature maps in these respective parts. We selected the feature map numbers in the thick model to ensure that the total number of trainable parameters closely matches that of our CIGN models. Following two distinct fully

connected layers, the features input into the routers (H blocks) undergo global average pooling to reduce complexity. Compared to the rest of the network, the routers contribute negligibly to the additional computation (MAC). Further details of the ResNet model structures can be observed in Figures 3.6 and 3.7.

Layer Name	Output Size	Thin ResNet	Thick ResNet
conv1	32×32	$3 \times 3, 16$, stride 1	
conv2_x	32×32	$\begin{pmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{pmatrix} \times 16$	
conv3_x	16×16	$\begin{pmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{pmatrix} \times 16$	$\begin{pmatrix} 1 \times 1, 29 \\ 3 \times 3, 29 \\ 1 \times 1, 116 \end{pmatrix} \times 16$
conv4_x	8×8	$\begin{pmatrix} 1 \times 1, 16 \\ 3 \times 3, 16 \\ 1 \times 1, 64 \end{pmatrix} \times 16$	$\begin{pmatrix} 1 \times 1, 30 \\ 3 \times 3, 30 \\ 1 \times 1, 120 \end{pmatrix} \times 16$
fc1	1×1	Global Average Pooling, 100-d fc, Softmax	

Figure 3.7. The specifications of the ResNet-145 architectures for both the thin and thick baselines employed in our experiments are outlined below. The columns represent the layer group names, feature map sizes, kernel sizes, and the number of channels, with parentheses denoting the bottleneck layers typically utilized in ResNet.

We employ an SGD optimizer with a momentum of 0.9. The initial learning rate is set to 0.1 and is scaled by a factor of 0.1 at 40000, 70000, and 100000 iterations. Each model undergoes training for 120000 epochs. The minibatch size is 256 for unconditional models, while for CIGN models, a batch size of 512 is utilized to ensure sufficient data for leaf nodes. Each model configuration incorporates an L2 norm penalty regularizer, and the optimal value is determined through grid search optimization for each configuration independently. The CIGN models employ binary trees of depth 2 ([2,2] trees).

Table 3.3 showcases the results of various models on the CIFAR-100 dataset. Each mean accuracy represents the outcome of five distinct model runs with the best hyperparameters. The thin baseline achieves a mean accuracy of 70.03% on the dataset, while the thicker baseline performs at 74.55%. Utilizing only the greedy information

gain routing ($\rho = 0.5$), the CIGN model attains a performance of 72.12%, employing nearly the same number of MAC operations as the thin baseline. Incorporating annealing during training, where all samples are sent to all paths for the initial 35000 epochs ($\rho = 0$), followed by setting the threshold $\rho = 0.4$, boosts the accuracy to 72.84% without increasing the number of parameters or computational complexity during inference.

Table 3.3. CIFAR-100 test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers.

Model	**Mean Acc.**	**Avg # MAC**	**# Params**
ResNet-145 Thin Baseline(*)	70.03%	87.86×10^6	233524
ResNet-145 Thick Baseline	74.55%	133.42×10^6	590120
CIGN-Random Routing	68.89%	87.86×10^6	587110
CIGN-All Paths	71.14%	87.86×10^6	587110
CIGN-Greedy IG Routing - No Annealing	72.12%	87.86×10^6	587110
CIGN-Greedy IG Routing + Annealing	72.84%	87.86×10^6	587110
sMoE-CIGN Optimized Routing-1 $\lambda = 1.0$	73.63%	91.19×10^6	587110
sMoE-CIGN Optimized Routing-2 $\lambda = 0.99$	73.61%	91.08×10^6	587110
sMoE-CIGN Optimized Routing-2 $\lambda = 0.9$	73.51%	90.00×10^6	587110

The sMoE model, employing Bayesian optimization, enhances performance at the expense of a slight increase in the average number of operations. In Figure 3.4, we present the accuracy-computation trade-off for various values of λ (Equation 3.21). As anticipated, as λ decreases, assigning more weight to the L_{MAC} term, Bayesian optimization sacrifices accuracy for a reduced average MAC per sample.

When $\lambda = 1.0$, the additional MAC per sample cost is entirely disregarded, resulting in an average of 3.8% MAC load per example, a relatively modest increase in computation load. With this configuration, we ignore the cost of extra computations and incur an extra 3.8% average MAC during inference, achieving a performance gain of approximately 0.8% over the greedy IG-based routing, totaling 73.63%. Although this is still lower than the thicker baseline, we achieve a 3.6% accuracy increase over the thin baseline, using fewer computation resources on average.

Additionally, as part of ablation studies, we compare our greedy IG and sMoE-based routing with the two other CIGN baselines mentioned earlier: a fully randomly

routed CIGN and a CIGN where each sample is sent to every path. Both of these baselines significantly underperform compared to our CIGN models (68.89% and 71.14%). We see that randomly routing the samples significantly underperforms compared to CIGN routing mechanisms. The baseline, where we send each sample to every path, performs better than random routing since the routing during inference is still based on information gain. But with an accuracy of 71.14%, this alternative baseline is still inferior to our CIGN models. Significance tests using Bootstrap t-testing demonstrate that all CIGN versions are significantly better than the baselines with zero p-values, except for the thick baseline.

3.7.5. USPS Dataset Experiments

The USPS dataset from the UCI repository comprises 9298 samples of 16×16 sized digit images distributed across 10 classes [145]. Our selection of this dataset aims to explore the impact of the CIGN approach on small datasets with a limited number of features. To adapt to fully connected layers, we vectorize the input images. As the thick baseline, we employ a three-layered MLP with hidden layer dimensions of $256, 192, 128$. We utilize binary CIGN models with a depth of two, where each root-to-leaf path has dimensions $32, 24$, and 16, respectively. This three-layered MLP also serves as our thin baseline. The information gain routers connect to the ends of 8 and 4 dimensional MLP layers in the first and second tree layers, respectively, which, in turn, link to the primary MLP layers in the decision nodes. All MLP layers utilize ReLU nonlinearities. The training of all CIGN models and unconditional baselines is carried out using the Adam optimizer.

In Table 3.4, the sMoE-CIGN method outperforms the baselines with an accuracy of 94.66%. Bootstrap t-testing indicates that these differences are statistically significant. Given the already minimal extra computation load at $\lambda = 1.0$ (only 0.5%), tests with smaller λ values are omitted. Additionally, we compare the CIGN method with a recently published, similar algorithm, "Random Forest Inspired Neural Networks" (NNRFs) [104].

Table 3.4. USPS test results. Each sample in the CIGNs visits a network equivalent to (*) plus routers.

Model	Mean Acc.	Avg # MAC	# Params
MLP Thin Baseline(*)	94.03%	9504	9586
MLP Thick Baseline	94.51%	140544	141130
CIGN-Random Routing	94.23%	9880	12622
CIGN-All Paths	94.18%	9880	12622
CIGN-Greedy IG Routing + Annealing	94.27%	9880	12622
sMoE-CIGN Optimized Routing $\lambda = 1.0$	94.66%	9931	12622
sMoE-CIGN-Forest (3 Trees) $\lambda = 1.0$	95.53%	29806	37866
NNRF [104] (150 Trees)	96.01%	49200	192600

In this approach, the authors construct a random forest using small neural trees and report results on the USPS dataset. To facilitate a fair comparison, we build an ensemble of 3 sMoE-CIGNs. While achieving an accuracy of 95.53% compared to NNRF's 96.01%, we do so at an average 60% of their MAC cost per sample and utilize only 20% of their parameter count. We attribute the accuracy difference mainly to NNRF's larger ensemble size. NNRF is designed with a focus on bagging and ensembling, whereas in CIGN, our emphasis is on a single network without random feature selection.

3.7.6. Comparison with Pruning Methods

As additional experiments, we conduct comparison tests between the CIGN and state-of-the-art pruning approaches, a domain where MNIST and LeNet are commonly used as the baseline dataset and architecture. These pruning methods typically evaluate the achieved sparsification and accuracy post-pruning in comparison to the original network. We specifically compare the results of our sMoE-CIGN with these algorithms using MNIST.

Table 3.5 showcases the accuracy, expected MAC count, and the number of parameters used for state-of-the-art pruning methods alongside sMoE-CIGN. Typically, these methods initiate with the same LeNet architecture, applying pruning to its convolutional filters and fully connected layer neurons, followed by a fine-tuning step in the post-pruning phase. Pruning methods systematically reduce the original LeNet's

parameters, whereas the CIGN method starts with a thinned baseline and further sparsifies the network during runtime by converting it into a tree structure. For CIGN, the number of used parameters is calculated based on the expected number of parameters during inference under the computed probability thresholds. Variational Dropout (VB) [22] and Variational Information Bottleneck (VIB) [115] achieve substantial sparsification but at the cost of lower model performance. In comparison, the sMoE-CIGN algorithm achieves a maximum sparsification of only 2.27%, but it attains the best accuracy among all pruning algorithms, owing to its semantic routing mechanisms. Furthermore, since CIGN is not designed for filter-wise or neuron-wise sparsification, it is complementary to these pruning methods, allowing for their combined use to achieve additional model compression. It should be also noted that since these pruning methods work on a much finer granularity, they can theoretically achieve a much smaller sparsification rate compared to our CIGN architecture. CIGN is not designed as a pruning method, it is inherently a conditional computation approach. The smallest level of sparsification that CIGN can achieve by design, is the execution of a single root-to-leaf path when the sMoE approach is not used.

Table 3.5. CIGN vs structured pruning methods comparison (On MNIST and LeNet). Each sample in the CIGNs visits a network equivalent to (*) plus routers.

Model	**Mean Acc.**	**Avg # MAC**	**# Used Params**	**Dense%**
LeNet Thick Baseline	99.25%	4.018×10^6	1256080	100%
LeNet Thin Baseline(*)	99.20%	1.057×10^6	26695	2.13%
CIGN-All Paths	99.27%	1.057×10^6	92276	7.35%
sMoE-CIGN-1	**99.46%**	1.394×10^6	46237	3.68%
sMoE-CIGN-2	99.43%	1.088×10^6	30220	2.41%
sMoE-CIGN-3	99.41%	1.060×10^6	28565	2.27%
SSL-1 [111]	99.20%	0.780×10^6	473534	37.70%
SSL-2 [111]	99.00%	0.432×10^6	300500	23.92%
GBD 4% Density [112]	94.00%	$\mathbf{0.074 \times 10^6}$	54588	4.35%
GBD 8% Density [112]	98.00%	0.152×10^6	103766	4.35%
VD [22]	99.0%	0.137×10^6	**4486**	**0.36**%
BC [114]	99.0%	0.408×10^6	35231	2.80%
VIB [115]	99.0%	0.244×10^6	10425	0.83%
GAL-0.01 [113]	99.14%	0.667×10^6	151743	12.08%
GAL-0.05 [113]	99.10%	0.266×10^6	79835	6.36%
GAL-0.1 [113]	98.99%	0.183×10^6	79903	6.36%

3.7.7. Semantic Partitioning of Data

In this section, we show the semantic partitioning effect of CIGN´s information gain-based routing. We posited that employing information gain-based routing would automatically organize the data into semantically meaningful groups. While previous works, such as [8], achieve this through pre-processing methods like spectral clustering, we observe that our information gain-based routing achieves a similar outcome without the need for additional preprocessing. Although we introduce a prior grouping to the data by specifying a tree topology, our information gain-based routing effectively partitions the data into groups where inter-group variance is high, and intra-group variance is low.

3.7.7.1. MNIST. In Figure 3.8, we present the distribution of test samples into leaf nodes, employing greedy information gain for routing. At each decision node, each sample strictly chooses the most likely path. The frequency of class labels in each node is displayed in ascending order, with the histograms representing only at most 90% of the samples in each leaf node. Each row showcases 10 randomly selected sample images from the corresponding class. In leaf 3, samples of digits 3 and 5 are grouped together, demonstrating a similar drawing style in the bottom halves of these two digits—a rightward bulged curve. This classifier effectively distinguishes between classes based on their top parts. In leaf 4, classes 2, 8, and 6 are grouped, as these classes share similar round and curved features in their digits. Leaf node 5 encompasses the classes 9, 4, and 0, where digits 9 and 4 exhibit notable similarities, particularly when the upper part of 9 has a cornered shape instead of being curved. The digit 0 and the upper part of 9 have compatible shapes when 9 is drawn with a curve. The digit 8 also has round features, leading to about 14% of 0 samples being routed to leaf node 4. Leaf node 6 is easier to analyze, dominated by classes 1 and 7, both having linear features and rarely round curves. In Figure 3.9, we illustrate the distribution of the MNIST digit test set across the leaf nodes of the CIGN model, as utilized in Figure 3.8.

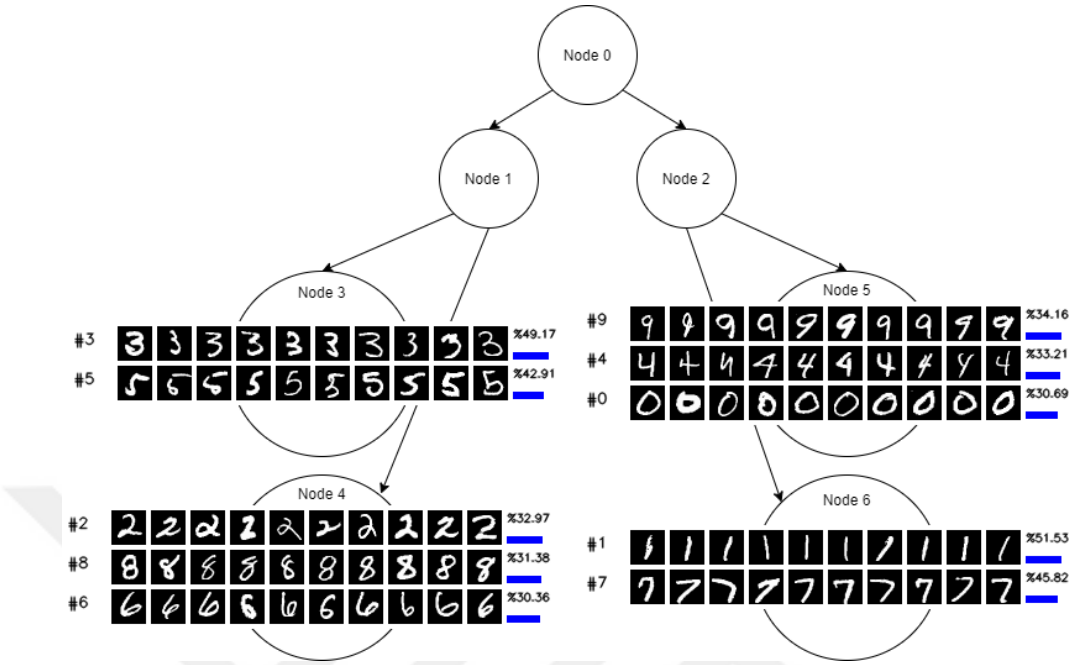


Figure 3.8. The arrangement of MNIST test data across the leaf nodes of the CIGN model is illustrated. The blue bars and accompanying percentage values indicate the frequency of the corresponding class in each node. For visual clarity, the histograms are truncated at a 90% threshold.

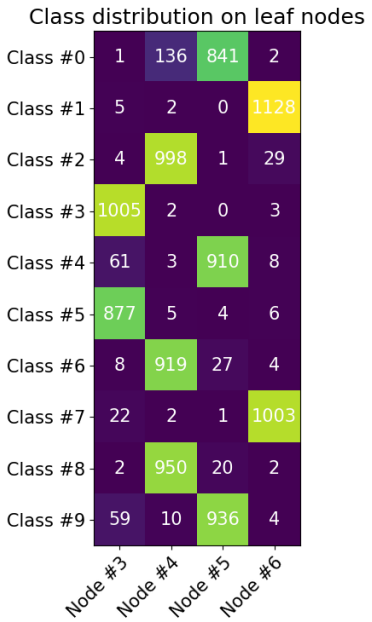


Figure 3.9. The allocation of each MNIST class among four leaf nodes in our CIGN model with information gain routing.

As previously mentioned, class 0 exhibits a tendency to be routed to more than one leaf node, likely due to both leaf nodes 4 and 5 containing samples with round figures. Conversely, other classes predominantly show a clear tendency to be grouped in a single leaf node, with more than 90% of the samples directed into the same node for each class.

3.7.7.2. Fashion-MNIST. Fashion-MNIST lacks an explicit class hierarchy like CIFAR-100, but it inherently exhibits modality, with three footwear classes, six clothing classes, and the Bag class that doesn't strictly align with either group. In Figure 3.10, we present the distribution of test set samples across the leaf nodes of our CIGN model. Each row displays 10 random samples from a class, along with their frequency distributions in the corresponding node. In leaf node 3, samples from the Trouser, T-Shirt, and Dress classes are predominantly accumulated. Trouser and Dress images typically feature long, thin blobs cutting the image into two halves, and both T-Shirt and Dress images predominantly contain short-armed samples, leading to the semantic grouping of these classes.

Leaf node 4 groups the Pullover, Coat, and Shirt classes, all characterized by garments with long sleeves. Some Shirt samples with short sleeves are routed to leaf node 3 due to visual similarity to the T-Shirt class. Leaf node 5 groups the Sneaker and Ankle Boot classes, showing clear semantic similarity. Leaf node 6 groups the Bag and Sandal classes, attributed to the similarity in thin, curved features like bag handles and sandal straps.

In Figure 3.11, we present the distribution of the Fashion-MNIST test set across four leaf nodes. It is observed that a minor portion of T-Shirt and Shirt samples tends to be routed into each other's predominant leaf nodes. This phenomenon is due to the visual similarity of certain samples between these classes. Conversely, most other classes are directed to their primary nodes with minimal misrouting. This outcome shows effective data partitioning into semantically similar subgroups through information gain-based routing.

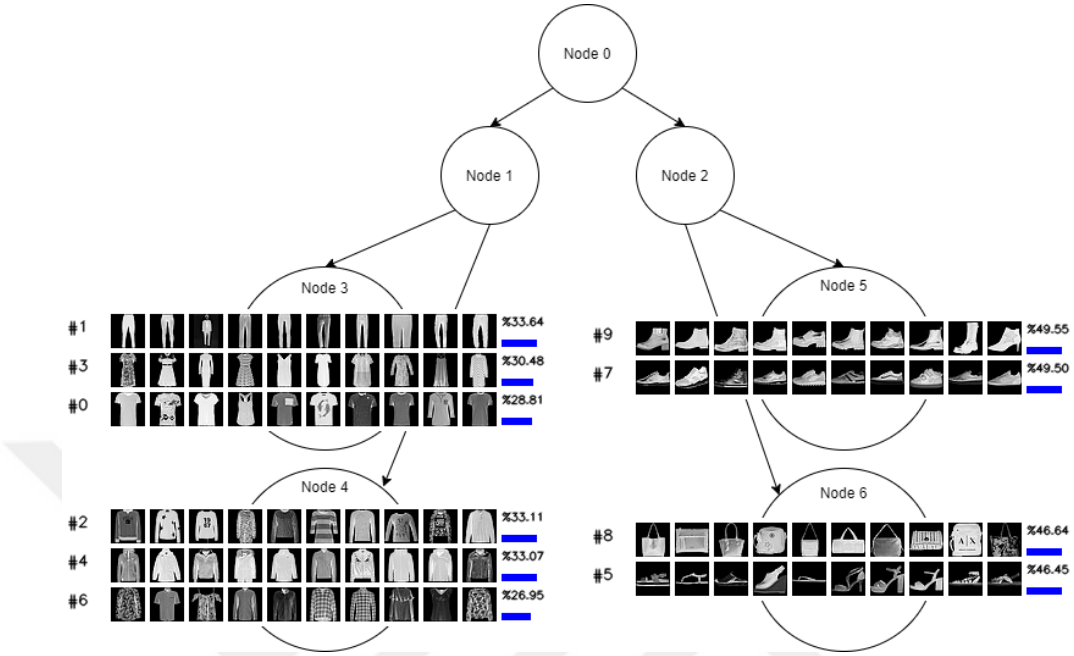


Figure 3.10. The distribution of Fashion-MNIST test data across the leaf nodes of the CIGN model is illustrated in the figure. The blue bars and corresponding percentage values depict the frequency of each class in the respective node. For visual simplicity, the histograms are truncated at a 90% threshold.

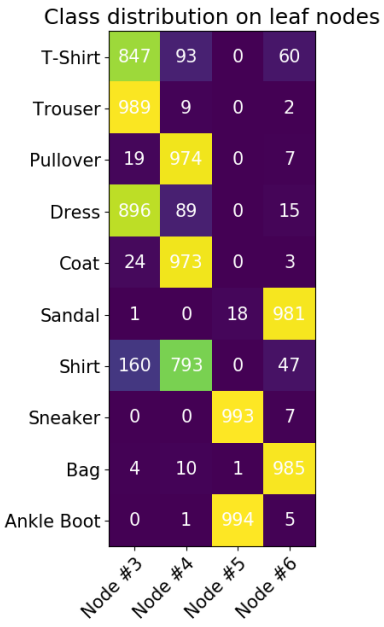


Figure 3.11. The allocation of each Fashion-MNIST class among four leaf nodes in our CIGN model with information gain routing.

3.7.7.3. CIFAR-100. We present the outcomes of one of our experiments using the information gain routed CIGN models. Figures 3.12, 3.13, 3.14, and 3.15 depict the distribution of CIFAR-100 classes among the four leaf nodes of the depth-2 binary CIGNs. Each row showcases five random samples from the corresponding class, accompanied by the percentage of that class present in the node. All samples are from the test set, arranged in descending order of frequency from the top left to the bottom right. Histograms are truncated when the cumulative frequency exceeds the 85% threshold.

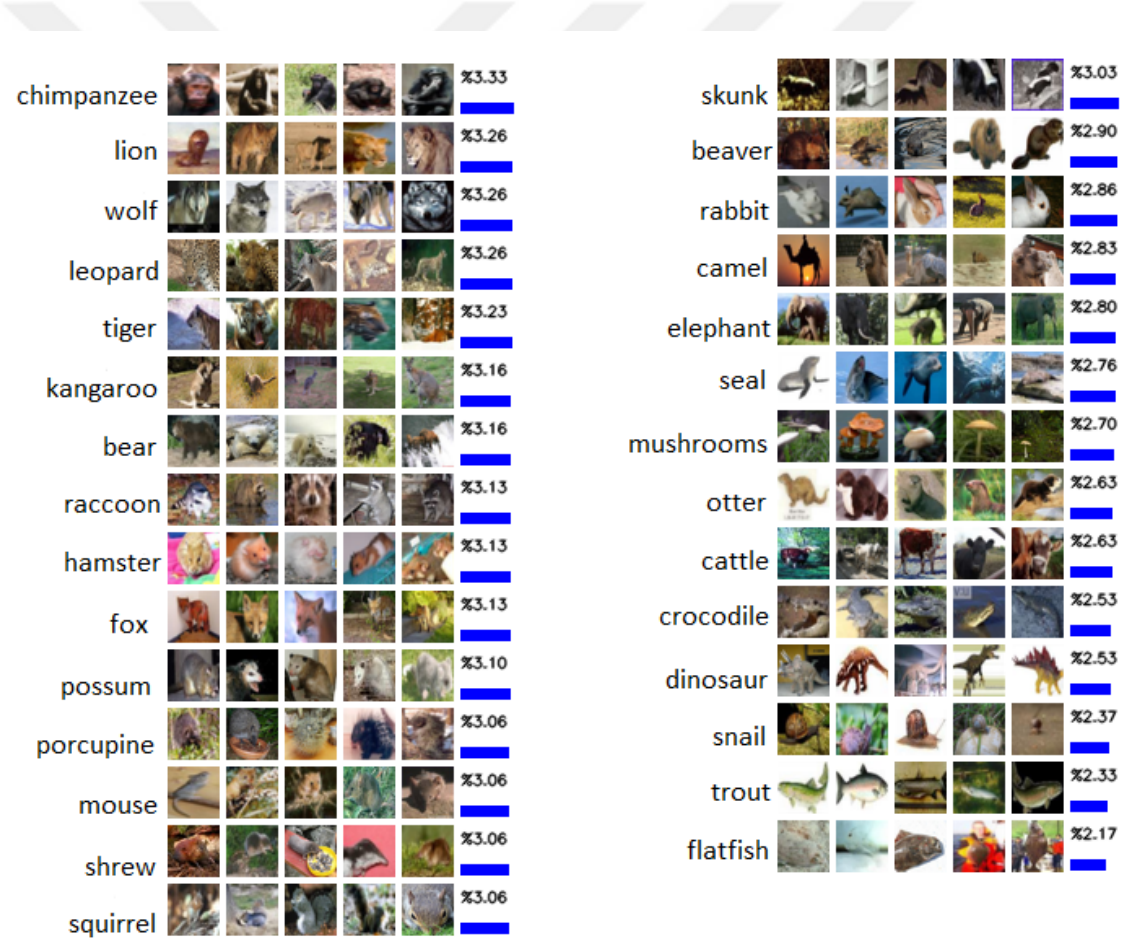


Figure 3.12. Distribution for the first leaf node of [2, 2] CIFAR100-CIGN.

In Figure 3.12, mammals, birds, certain reptiles, and fish cluster together, making up 85% of the sampling frequency at a leaf node. In Figure 3.13, the second leaf groups fruits, flowers, people, motorcycles, bikes, and other reptiles.

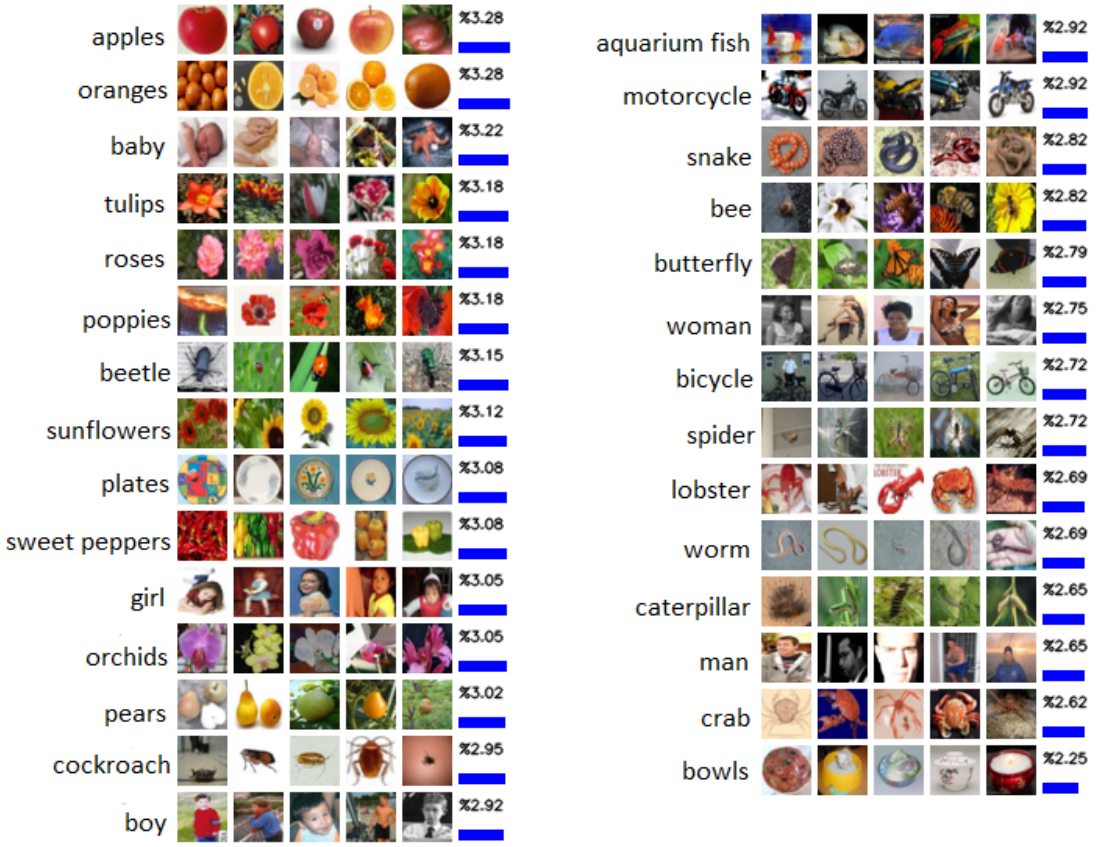


Figure 3.13. Distribution for the second leaf node of [2, 2] CIFAR100-CIGN.

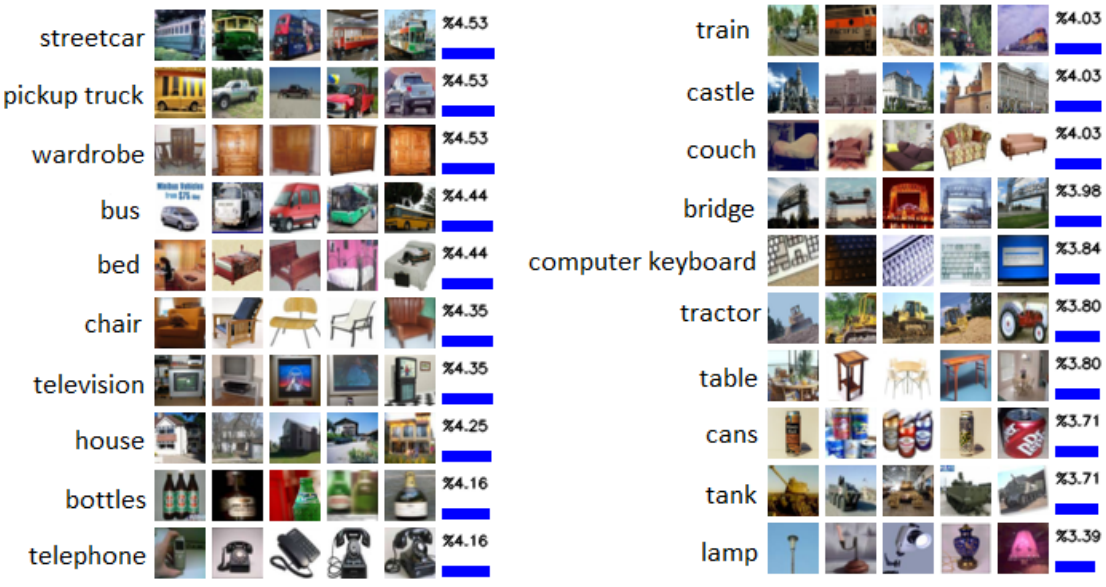


Figure 3.14. Distribution for the third leaf node of [2, 2] CIFAR100-CIGN.

These objects typically exhibit round features and are characterized by predominantly red-yellowish colors (such as apples, oranges, flower centers, rounded snake figures, motorcycle tires, and human faces).

In Figure 3.14, the broader class grouping comprises artificial objects such as furniture, cars, trains, electronic devices, buildings, and beverage containers. The dominant brown-yellowish color of the wood in furniture and the shared shapes of different pieces of furniture might contribute to their grouping. Similarly, the presence of common elements like tires and windows in trucks, cars, trains, and minibuses aligns with their grouping in the same leaf node.

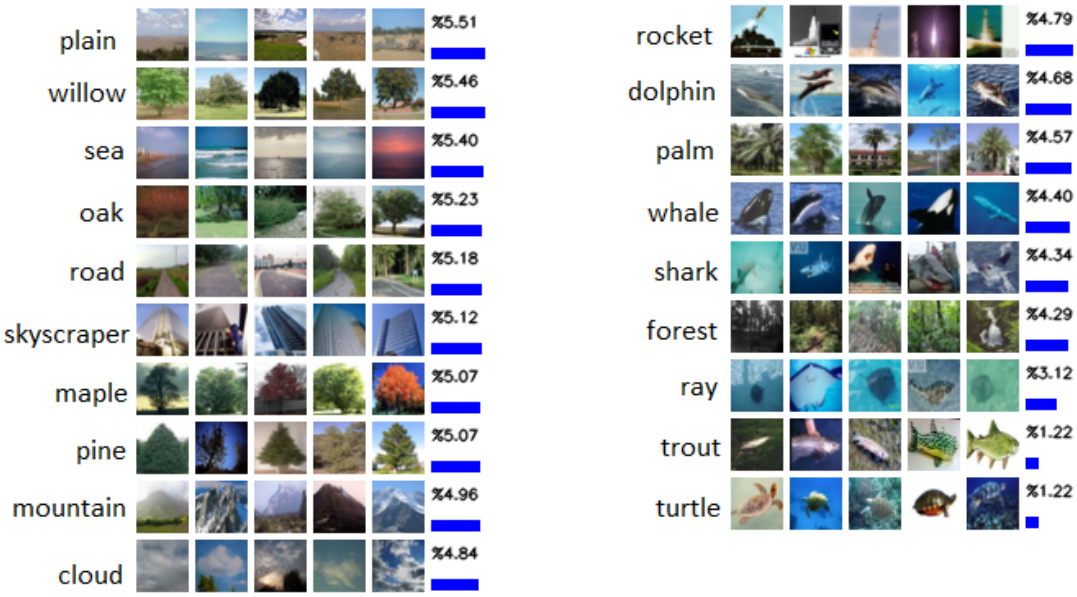


Figure 3.15. Distribution for the fourth leaf node of [2, 2] CIFAR100-CIGN.

In Figure 3.15, landscapes, various types of trees, skyscrapers, rockets, undersea creatures, and images featuring the sky are clustered together. Pictures depicting undersea scenes, the sky, and landscapes predominantly exhibit a bluish hue. The router identified semantically similar characteristics in images of rockets and skyscrapers, both of which involve tall objects. Additionally, all distinct classes of trees are predominantly grouped in this node.

In Figure 3.16, we present the distribution of each class in the test set across the four leaf nodes, derived from the same model used to generate Figures 3.12, 3.13, 3.14, and 3.15. This figure illustrates that classes with similar semantic characteristics are consistently directed to the same leaf nodes, reinforcing the observations made in Figures 3.12, 3.13, 3.14, and 3.15. Certain classes with substantial intra-class variance, such as "lizard" or "turtle", do not exhibit concentration in a single leaf. In cases where a singular root-to-leaf expert might struggle to achieve optimal performance due to high intra-class variance, the routing probabilities exhibit high entropy. This indicates uncertainty in the model about the correct path for these samples. The thresholds identified through Bayesian optimization help mitigate potential errors by directing such samples to multiple leaf nodes. This approach increases the chances of correctly classifying difficult samples by leveraging the combined expertise of several nodes.

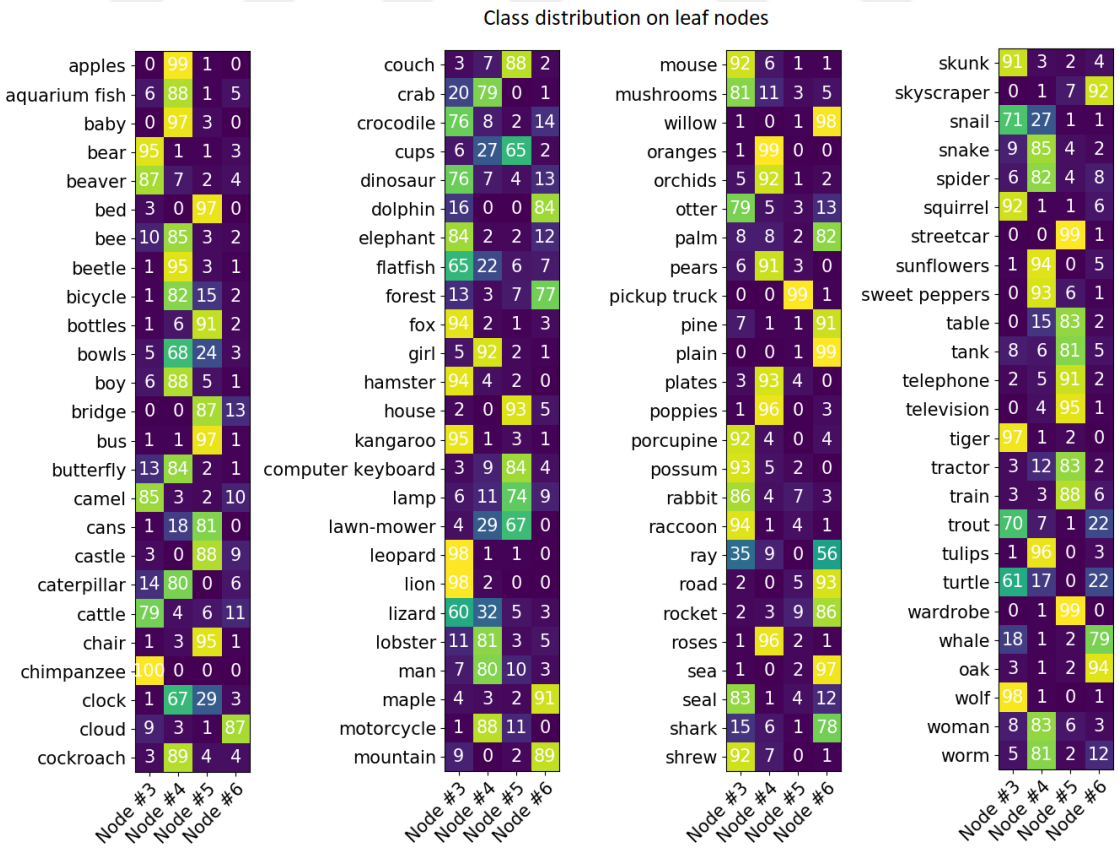


Figure 3.16. The allocation of each CIFAR-100 class among four leaf nodes in our CIGN model with information gain routing.

Figure 3.17 illustrates the distribution of the coarse CIFAR-100 categories among the leaf nodes. Despite not incorporating the coarse class labels during training, it is evident that half of the manually determined coarse class groups align precisely with the semantic groupings of the CIGN model. The remaining half of the coarse groups typically consist of two clusters characterized by high visual cluster intra-variances. The "fruit and vegetables" group includes classes like apples, mushrooms, oranges, pears, and sweet peppers. Apples, oranges, sweet peppers, and pears are largely routed to the second leaf node due to their distinct round shapes and red-yellowish colors. Conversely, the mushroom class is mainly routed to the first leaf node due to dark-greenish backgrounds and the diverse shapes and colors of mushrooms. This shows how the model's semantic understanding aligns with human-defined categories, even without explicit coarse class labels during training.

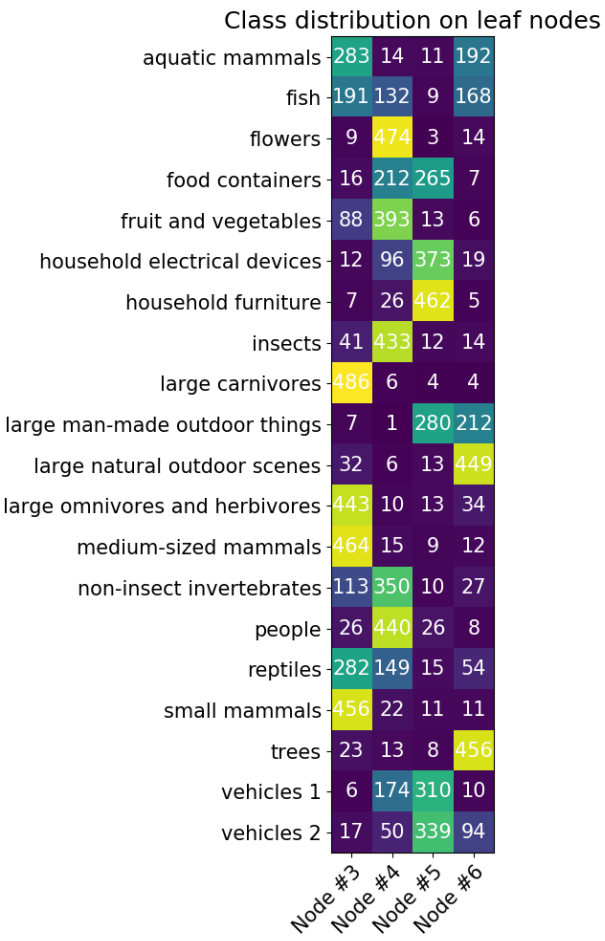


Figure 3.17. Distribution of CIFAR-100 coarse labels on the four leaf nodes.

3.7.8. Statistical Tests for Experiments

In this section, we present the outcomes of the Bootstrap t-tests conducted between the baselines and CIGN models. We will also outline the Bootstrap t-Test algorithm utilized, as proposed by [148].

Table 3.6. MNIST - CIGN vs baselines statistical comparison.

p-Values	Thin Baseline	Thick Baseline	Random*	All Paths*
No Annealing*	(+) $\mathbf{4.3 \times 10^{-3}}$	(-) 7.3×10^{-2}	(-) 2.2×10^{-1}	(-) 9.1×10^{-2}
Annealing*	(+) $\mathbf{2.0 \times 10^{-4}}$	(+) **0**	(+) **0**	(+) **0**
sMoE-1*	(+) $\mathbf{5.0 \times 10^{-4}}$	(+) **0**	(+) **0**	(+) **0**
sMoE-2*	(+) $\mathbf{7.0 \times 10^{-4}}$	(+) **0**	(+) **0**	(+) **0**
sMoE-3*	(+) $\mathbf{9.0 \times 10^{-4}}$	(+) **0**	(+) **0**	(+) **0**

Table 3.7. Fashion-MNIST - CIGN vs baselines statistical comparison.

p-Values	Thin Baseline	Thick Baseline	Random*	All Paths*
No Annealing*	(+) **0**	(-) 4.2×10^{-1}	(+) $\mathbf{1.0 \times 10^{-4}}$	(+) $\mathbf{5.0 \times 10^{-4}}$
Annealing*	(+) $\mathbf{4.0 \times 10^{-4}}$	(-) 1.6×10^{-1}	(+) $\mathbf{5.0 \times 10^{-3}}$	(+) $\mathbf{2.1 \times 10^{-2}}$
sMoE-1*	(+) **0**	(+) **0**	(+) **0**	(+) **0**
sMoE-2*	(+) **0**	(+) $\mathbf{1.0 \times 10^{-4}}$	(+) **0**	(+) **0**
sMoE-3*	(+) **0**	(+) $\mathbf{2.4 \times 10^{-3}}$	(+) **0**	(+) **0**

Table 3.8. CIFAR-100 - CIGN vs baselines statistical comparison.

p-Values	Thin Baseline	Thick Baseline	Random*	All Paths*
No Annealing*	(+) **0**	(+) $\mathbf{2.0 \times 10^{-4}}$	(+) **0**	(+) **0**
Annealing*	(+) **0**	(+) $\mathbf{4.0 \times 10^{-4}}$	(+) **0**	(+) **0**
sMoE-1*	(+) **0**	(+) $\mathbf{8.8 \times 10^{-3}}$	(+) **0**	(+) **0**
sMoE-2*	(+) **0**	(+) $\mathbf{6.3 \times 10^{-3}}$	(+) **0**	(+) **0**
sMoE-3*	(+) **0**	(+) $\mathbf{2.9 \times 10^{-3}}$	(+) **0**	(+) **0**

Table 3.9. USPS - CIGN vs baselines statistical comparison.

p-Values	Thin Baseline	Thick Baseline	Random*	All Paths*
Annealing*	(+) $\mathbf{4.8 \times 10^{-2}}$	(+) **0.0**	(-) 1.2×10^{-1}	(-) 1.2×10^{-1}
sMoE-1 ($\lambda = 1.0$)*	(+) $\mathbf{1.4 \times 10^{-3}}$	(+) $\mathbf{1.1 \times 10^{-2}}$	(+) **0**	(+) **0**

The results of each hypothesis test are organized in a table, with columns representing baselines and rows representing our CIGN models. The entries in each cell denote the resulting p-value between the models in the corresponding column and row. Green coloring indicates that the CIGN model outperforms the corresponding baseline, while red denotes the opposite. Models with (*) signify the use of a CIGN structure.

In the table, a (+) indicates the rejection of the null hypothesis, and a (-) signifies otherwise. A significance level of 0.05 has been applied across all experiments. Algorithm 2 shows the Bootstrap t-test we have applied to our experiment results. We have decided to use this particular test since it is particularly suitable when the sample sizes are small. The test itself tries to accommodate that by drawing bootstrap samples from the already existing ones and applying a simulation of a conventional t-test, which is usually applied to larger samples.

Algorithm 2: Bootstrap t-Test For Comparing Means

Result: p-value

Two samples x and y of sizes n_x and n_y given.

The null hypothesis is $H_0 : \mu_x = \mu_y$.

Calculate sample means and variances: $\overline{x}, \overline{y}, \sigma_x^2, \sigma_y^2$;

Calculate t-statistic: $t = \dfrac{\overline{x} - \overline{y}}{\sqrt{\dfrac{\sigma_x^2}{n_x} + \dfrac{\sigma_y^2}{n_y}}}$;

Create a combined sample z such that $z = concatenate(x, y)$. Let its mean be $\overline{z}$;

Create two modified data sets x' and y', which satisfies H_0:

$x' : \{x_i - \overline{x} + \overline{z} | \forall x_i \in x\}$;

$y' : \{y_i - \overline{y} + \overline{z} | \forall y_i \in y\}$;

Set N to boostrap simulation count (for example $N = 10000$);

Set $B = 0$;

while $i = 1, \ldots, N$ **do**

  Draw bootsrap samples $x_i^* \sim x'$ and $y_i^* \sim y'$ with sizes n_x and n_y, with replacement;

  Calculate simulated t-statistic: $t_i^* = \dfrac{\overline{x_i^*} - \overline{y_i^*}}{\sqrt{\dfrac{\sigma_{x_i^*}^2}{n_x} + \dfrac{\sigma_{y_i^*}^2}{n_y}}}$;

  Check if t_i^* is at least extreme as t; where $\mathbb{I}$ is the indicator function:;

  $B \leftarrow B + \mathbb{I}\left[t_i^* \notin [-t, t]\right]$;

end

Estimate the simulated p-value:;

$p = \dfrac{B}{N}$

4. CONDITIONAL INFORMATION GAIN TRELLIS

4.1. Conditional Information Gain Trellis

In the previous chapter, we introduced our "Conditional Information Gain Networks" model, that describes a tree-structured neural network that allows conditional execution in deep neural network architectures [26, 27]. CIGN establishes a tree-structured network with router elements in non-leaf nodes, using router mechanisms trained with local information gain objectives to maximize information gain. In the context of CIGN, the routers, trained with local information gain objectives, aim to maximize information gain by learning an optimal split. This split ensures that data routed into subtrees becomes purer, grouping semantically similar classes. The convolutional filters in these subtrees then specialize in learning discriminative representations for their respective data groups, resulting in a reduction of parameters in the tree branches.

A tree structure is a natural way to implement conditional execution. It has also been used in different works to introduce interpretability [91, 93, 94, 96, 97, 149]. A tree structure induces a hierarchical coarse-to-fine refining procedure to identify the properties of the input, which is analogous to a human decision-making process. For this reason, it is preferable in the context of the interpretable and explainable AI domain. However, when it comes to conditional execution, the tree structure has an important flaw: When the routers in the non-leaf tree nodes mistakenly route a sample into a subtree that is not optimized for the coarse semantic group for that particular input, the likelihood of making prediction errors gets high. We have demonstrated this effect with the tree-based CIGN models in Chapter 3. With CIGNs, we have proposed some countermeasures to remedy this problem:

- We have introduced thresholds during the training time for the routing probabilities, such that samples with routing probabilities larger than these thresholds

would route into multiple paths in the tree structure. During the training, these thresholds are increased with an annealing procedure. This helped by allowing each leaf classifier to see a large portion of data from the start of the training and then slowly focusing on their target coarse group. Since each leaf classifier had seen samples from different classes during the whole training, in cases of misrouting, this helped with a higher number of correct predictions.

- The Sparse Mixture of Experts inference system increased the performance further by introducing probability thresholds in the inference phase. We saw that samples that generate uncertain routing probabilities tend to get misclassified. To cover such cases, we have learned probability thresholds on a validation set, such that when routing probabilities are larger than these thresholds during the training, the sample would follow multiple paths and the classification would take place by averaging the results from multiple leaf experts.

While these measures provide performance increases, they still do not address the inherent problem with the tree topology: In the case of misrouting, a sample does not have a chance to recover the correct path in the conditional structure. A trellis structure, instead of a tree, helps with this disadvantage. In a trellis architecture, every node in a previous layer is connected to every other node in the next layer. There is always a route that exists between every combination of nodes in every layer: Assuming a trellis structure that contains N_l nodes at each layer with a total of L layers, this amounts to a total of $\prod_{l=}^{L} N_l$ paths in the whole structure. There isn´t a single route between two nodes in the architecture, like in the tree structure. However, evaluating each of these routes is intractable due to the combinatorial complexity of the route count. A conditional execution method that aims to utilize the benefits of the trellis structure also needs to take this into account.

In this chapter, we introduce a novel trellis-shaped model that enables the recovery of misrouted samples. If a sample is misrouted to the wrong computational unit, it will still have a chance to select the correct path in the next layer. We call this architecture “Conditional Information Gain Trellis”, abbreviated as CIGT in the rest of

the text. Similar to our tree-based CIGN models, differentiable information gain-based losses in each layer of the trellis structure are used to implement conditional execution. With these, we aim to create expert paths in the Trellis structure for different semantic groups in the given data. Given a convolutional or fully connected layer or a single block of such layers, at every level of trellis structure, we divide the computational units into N_l parallel groups (called F units) for every layer l, whose aim is to learn required representations for the data and a single H_l unit for that layer, called a router, which learns a probability distribution over the computation units F in the following layer. The probability distribution on each layer's F units ends up defining a global probability distribution on each root-to-leaf path in the Trellis structure, and hence the model can be treated as a hierarchical mixture of experts model [11].

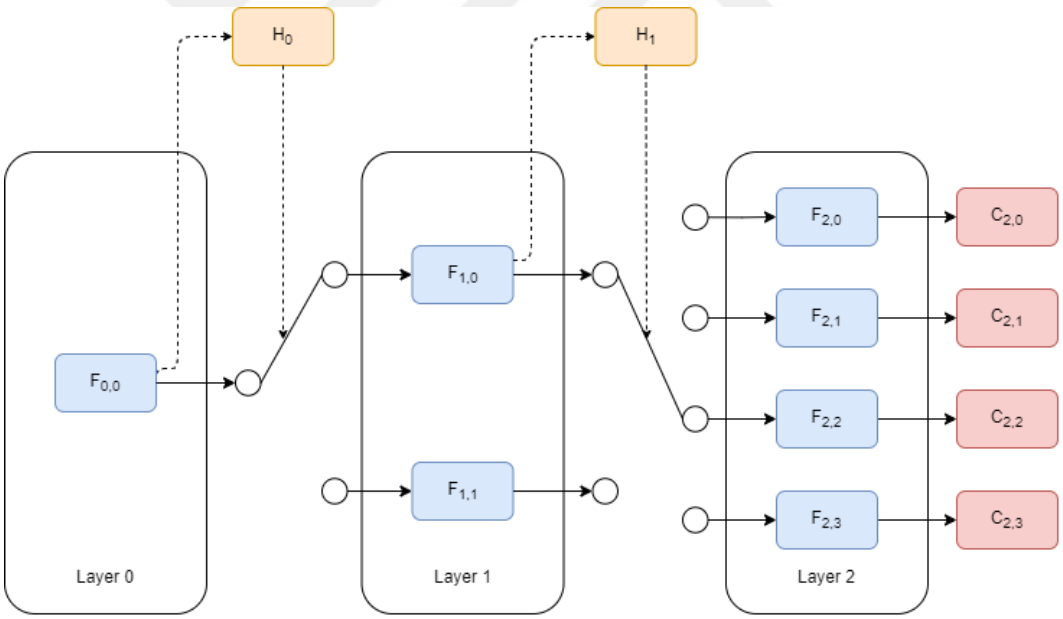


Figure 4.1. An example of the proposed CIGT architecture. In our terminology, this corresponds to a CIGT-[1,2,4] structure. The router units H_i direct the samples into the computation units of the following layers, namely $F_{i+1,j}$.

The structure of a general CIGT architecture can be seen in Figure 4.1. The CIGT model in this figure corresponds to a CIGT-[1,2,4] model, which means the model has three layers, and at each layer, there are 1, 2, and 4 parallel units of computation,

respectively. According to the figure, the first layer contains a single unit $F_{0,0}$ that feeds input into the router H_0. The output of $F_{0,0}$ is fed into the computation unit in the next layer, which H_0 decides. In this case, $F_{1,0}$ has been selected. Then, the output of layer $F_{1,0}$ is used as input to router H_1. H_1 directs the sample into the computation unit $F_{2,2}$, where the classification also takes place, driven by the loss $C_{2,2}$. It should be noted that, in this particular example, the units $F_{0,0}, F_{1,0}, F_{2,2}$ determine a single root-to-leaf deep neural network as a composite function $F_{2,2} \circ F_{1,0} \circ F_{0,0}(x)$, which is ideally a slimmer version of a thicker network. The purpose is to be able to select the best of possible root-to-leaf nodes entailed by this system by virtue of the conditional information gain-based routers, H_0 and H_1. Again, for the particular network of the type CIGT[1,2,4] there are 8 such possible root-to-leaf experts. By defining a single router unit for every layer, we only need to learn two probability distributions whose complexity is linear with the depth of the trellis, avoiding combinatorial complexity that will make the training of such a model intractable.

While the CIGT model is characterized as a mixture of experts architecture, the application of the Expectation-Maximization (EM) training is not straightforward due to the extensive number of experts, each represented as a convolutional neural network from root-to-leaf. This would mean a new deep neural network optimization procedure at every maximization step of the EM algorithm for every expert, which is already a very costly operation by itself. This complexity necessitates a novel optimization procedure for CIGTs. We replace the naive EM algorithm with routing systems based on differentiable information gain goals [26, 27, 31]. These systems make it possible for computational units to run sparsely at each trellis layer and for samples from classes that are semantically similar to be grouped together. By having comparable samples with subtle semantic distinctions traverse identical root-to-leaf paths in the trellis structure, it is posited that the deep network constituting the path will naturally distinguish the finer details of such samples, effectively transforming the original multi-class problem into a more manageable one. The probability distributions over each block's F units are trained with the previous block's corresponding local information gain losses.

In this chapter, we will be presenting the CIGT architecture in more detail. We will cover the basic architecture and two different approaches to training the hierarchical mixture of experts model. We will also show a Sparse Mixture of Expert inference algorithm, as we did in the CIGN models in the previous chapter. We will learn optimal probability thresholds as a post-processing step for a trained CIGT model, such that samples that generate uncertain routing probabilities could be routed into multiple paths in the trellis structure. We will show the benefits of this model and the Sparse Mixture of Experts (sMoE) extension through various experiments.

4.2. Architecture

The CIGT algorithm builds upon a Trellis structure of computational units. Each ith layer of the Trellis structure contains K_i parallel, identical computational units. These computational units can be parts of any widely used neural network architecture, like residual convolutional layers or dense layers. They can have common deep neural network operations like Batch Normalization [55], Dropout [46], and different types of non-linear activations.

We call a single trellis layer as F_i and the parallel computation units in this layer as $F_{i,j}$ where $1 \leq j \leq K_i$. Each computational unit $F_{i,j}$ contains $1/K_i$ times lower parameters (filters for convolutional layers, hidden units for dense layers). To extract low-level abstractions shared by all layers, we have a certain amount of constant, unconditional computation in block 0, F_0.

As an instance of the composition of a layer i in the CIGT, consider first an unconditional computational block, with a convolutional layer producing an output shape of $W \times H \times C$, where W and H represent the width and height of the feature map, and C is the channel count. Following this, we assume that there is a dense layer with a dimension of D, after a global average pooling or flattening layer. If we have a value of $K_i = 2$ for that CIGT layer, then each of the aforementioned operations is subdivided into two parallel units, with dimensions $W \times H \times C/2$ and $D/2$ respectively.

While the i.th layer as a whole is called F_i, the two parallel computational units that have been generated in the mentioned way are called $F_{i,0}$ and $F_{i,1}$. A sample in the CIGT structure will select either of these two units based on the router decision from the previous CIGT layer. (With the possible exception of the sMoE version of the model.). It's worth noting that while blocks may have diverse compositions, we have chosen to employ identical units for each block in this study.

When a sole $F_{l,i}$ unit is chosen in each CIGT layer, a sequential arrangement of computational nodes $(F_{L,i_L} \circ \cdots \circ F_{2,i_2} \circ F_{1,i_1} \circ F_0)(x)$ constitutes a single root-to-leaf expert. Such a root-to-leaf expert establishes a posterior distribution $p(y|Z, x, \theta)$ for the given data point (x, y). Here, Z denotes the matrix indicating expert selection, with row l of Z representing the expert selection configuration for the corresponding CIGT layer. Layer 0 always contains a single, shared computational unit. We can consider Z a matrix of the size $(L+1) \times max\{1, K_1, \ldots, K_L\}$ for the aforementioned CIGT architecture. The layers with a smaller number of computational units than $max\{1, K_1, \ldots, K_L\}$ can be assumed to contain zeros for the locations of the missing units. Setting $K_{max} = max\{1, K_1, \ldots, K_L\}$, the lth row, denoted as Z_l, takes the form of a binary vector, $Z_l \in \{0,1\}^{K_l} \cup \{0\}^{K_{max}-K_l}$. For instance, if $K_l = 3$ at layer l, $K_{max} = 4$, and $Z_l = [0, 1, 0, 0]$, it signifies that the computational unit $F_{l,1}$ is chosen for the current sample, while $F_{l,0}$ and $F_{l,2}$ are omitted. Also, since $F_{l,3}$ does not exist, the last entry of Z_l will always be 0.

θ encompasses all neural network parameters, such as convolutional weights or matrices for fully connected layers, that govern the behavior of the F units. The routers are denoted as H units, typically linked to the intermediate outputs of the F units within the same layer, and they parameterize a probability distribution $p(Z_{l+1}|x, \theta, \phi)$ for the subsequent layer of F units. ϕ encapsulates all parameters responsible for transforming the intermediate input into a probability distribution within the H unit. Since a H unit processes the intermediate outputs of F units, it is dependent on the θ parameters. The training of H units involves the utilization of information-gain objectives. These layer-wise distributions on the F units collectively form a probability

distribution across all root-to-leaf expert paths in the model, as

$$p(Z|x,\theta,\phi) = \prod_{l=1}^{L} p(Z_l|x,\theta,\phi). \tag{4.1}$$

To maintain simplicity in notation, we use the symbol Z to represent all Z_l in each layer throughout the text, unless specified otherwise. In the context of a dataset $\{x_i, y_i\}_{i=1}^{N}$, the primary classification objective function of the mixture of experts model can be expressed as

$$\mathbb{E}_{p_{data}(x,y)}\left[\log p(y|x,\theta,\phi)\right] = \frac{1}{N}\sum_{i=1}^{N} \log \sum_{Z} p(y_i|Z,x_i,\theta) \prod_{l=1}^{L} p(Z_l|x_i,\theta,\phi). \tag{4.2}$$

Instead of the objective outlined in Equation (4.2), we employ the subsequent lower bound at Equation (4.3), akin to the REINFORCE approach [35]. This lower bound is preferred as it removes the summation from the logarithm, resulting in improved behavior for the computation of stochastic gradient estimates. This is given as

$$\begin{aligned} J_C(\theta,\phi) =& \frac{1}{N}\sum_{i=1}^{N}\sum_{Z} \log p(y_i|Z,x_i,\theta) \prod_{l=1}^{L} p(Z_l|x_i,\theta,\phi) \\ \leq \hat{J}_C(\theta,\phi) :=& \frac{1}{N}\sum_{i=1}^{n} \log \sum_{Z} p(y_i|Z,x_i,\theta) \prod_{l=1}^{L} p(Z_l|x_i,\theta,\phi), \end{aligned} \tag{4.3}$$

which is the result of Jensen's Inequality. Additionally, we incorporate information gain losses, denoted as IG_l, which guide the data distribution to computational units in each layer l. As a result, our overall loss is defined as

$$L_{CIGT} = -J_C(\theta,\phi) - \lambda_{IG}\sum_{l=1}^{L} IG_l + \Omega(\theta,\phi). \tag{4.4}$$

Here, λ_{IG} controls the intensity of the collective information gain losses, and Ω serves as a regularizer on the network parameters.

4.3. Local Information Gain Losses

The H units establish probability distributions $p(Z_l|x,\theta,\phi)$ over the F units in the subsequent layers. The router parameters ϕ and network parameters θ undergo training through information gain losses formulated based on these local probability distributions. We have structured the information gain losses in a manner consistent

with our earlier CIGN works [26, 27]. In each layer of CIGT, we define the following joint probability distribution

$$p(y, Z_l, x|\theta, \phi) = p(y, x)p(Z_l|x, \theta, \phi). \tag{4.5}$$

In this context, $p(y, x)$ represents the empirical distribution of training data. We express $p(Z_l|x, \theta, \phi)$ as a softmax function applied to the potential values of the one-hot vector Z_l, as

$$p(Z_l = \text{onehot}(k)|x, \theta, \phi) = \frac{\exp\{(w_k^T h_{l-1}(x) + b_k)/\tau\}}{\sum_{i=1}^{K_l} \exp\{(w_i^T h_{l-1}(x) + b_i)/\tau\}}. \tag{4.6}$$

The hyperplanes denoted by w_k, b_k establish the decision boundaries at the routers, while $h_{l-}(x)$ signifies the output of transformations applied to the intermediate representation of the sample x in the previous layer, generated by the selected F unit in that layer.

The parameter τ serves as a hyperparameter governing the smoothness of the probability distribution. As τ approaches infinity, the probability distribution tends toward uniformity, while as τ approaches 0, it converges to

$$\text{onehot}\left(\arg\max_i \left(w_i^T h_{l-1}(x) + b_i\right)\right). \tag{4.7}$$

This results in a one-hot vector, where the entry corresponding to the highest logit is set to one and the others are set to zero. This is used for the same purpose with the same τ parameter in the CIGN routers; it is slowly annealed to 0 from a larger value during the training to force the routers to make more certain decisions.

The local information gain at layer l is then given as

$$IG_l = \mathbb{H}\left[p(y)\right] - \mathbb{E}_{p(Z_l|\theta,\phi)}\left[\mathbb{H}\left[p(y|Z_l, \theta, \phi)\right]\right]. \tag{4.8}$$

Here $\mathbb{H}\left[p(x)\right]$ is defined as the entropy of the probability distribution $p(x)$: $\mathbb{H}\left[p(x)\right] = -\sum_x p(x) \log p(x)$. The derivative of the information gain term concerning a network parameter is equivalent to the corresponding equation in our CIGN works; it is stated in Chapter 3. Similar to CIGN models, in order to provide a balanced distribution of the samples, we again use the mathematical properties of the information gain term.

The information gain can be decomposed as

$$IG_l = \mathbb{H}\left[p(y)\right] + \mathbb{H}\left[p(Z_l|\theta,\phi)\right] - \mathbb{H}\left[p(y,Z_l|\theta,\phi)\right]. \tag{4.9}$$

The maximization of information gain occurs when $\mathbb{H}\left[p(Z_l|\theta,\phi)\right]$ reaches its maximum and $\mathbb{H}\left[p(y,Z_l|\theta,\phi)\right]$ reaches its minimum. It is important to note that a higher value of $\mathbb{H}\left[p(Z_l|\theta,\phi)\right]$ implies a more evenly distributed allocation of data to the F units in block l. To amplify the significance of this entropy term, we introduce a hyperparameter $\lambda_{balance} > 1$, as implemented in [26]. This results in the information gain objective by taking the form

$$IG_l^{balanced} = \mathbb{H}\left[p(y)\right] + \lambda_{balance}\mathbb{H}\left[p(Z_l|\theta,\phi)\right] - \mathbb{H}\left[p(y,Z_l|\theta,\phi)\right]. \tag{4.10}$$

This adjusted cost function favors minima, characterized by more evenly distributed data across the computation nodes. The information gain objective offers an elegant solution to address this load balancing issue (referred to as "mode collapse" in [74]), as compared to the supplementary proxy balancing losses found in works such as [12, 16, 48].

4.4. The Problem with Stochastic Calculations

The derivatives of the lower bound in Eq. (4.3) with respect to θ and ϕ are given as

$$\begin{aligned}\frac{\partial J_C}{\partial \theta} =& \frac{1}{N}\sum_{i=1}^{N}\sum_{Z}\frac{\partial \log p(y_i|Z,x_i,\theta)}{\partial\theta}\prod_{l=1}^{L}p(Z_l|x_i,\theta,\phi) \\ &+ \sum_{i=1}^{N}\sum_{Z}\log p(y_i|Z,x_i,\theta)\frac{\partial\prod_{l=1}^{L}p(Z_l|x_i,\theta,\phi)}{\partial\theta},\end{aligned} \tag{4.11}$$

and

$$\frac{\partial J_C}{\partial \phi} = \frac{1}{N}\sum_{i=1}^{N}\sum_{Z}\log p(y_i|Z,x_i,\theta)\frac{\partial\prod_{l=1}^{L}p(Z_l|x_i,\theta,\phi)}{\partial\phi}. \tag{4.12}$$

While the first term of $\frac{\partial L_C}{\partial \theta}$ can be straightforwardly approximated using a simple Monte Carlo approach by drawing samples $x_i \sim p(x)$ and $Z_i \sim p(Z|x_i, \theta, \phi)$, the second term and the derivative $\frac{\partial L_C}{\partial \phi}$ do not involve an expectation with respect to a specific distribution. Consequently, computing them through Monte Carlo simulations is not feasible. Generally, gradients of the form $\frac{\partial}{\partial w}\mathbb{E}_{p(x|w)}[f(x)]$ pose challenges and various strategies have been proposed in the literature to address such problematic terms. This type of differentiation is commonly referred to as "differentiation through stochastic nodes" [150], and because it includes a sampling operation, its definition is not well-established. Two main approaches are commonly employed to tackle this issue.

The first approach is based on REINFORCE-type score function estimators [35]. The core idea of this method involves using the identity

$$\frac{\partial}{\partial w}p(x|w) = p(x|w)\frac{\partial}{\partial w}\log p(x|w). \tag{4.13}$$

This transformation converts the challenging term

$$\frac{\partial}{\partial w}\mathbb{E}_{p(x|w)}[f(x)] \tag{4.14}$$

into

$$\mathbb{E}_{p(x|w)}[f(x)]\frac{\partial}{\partial w}\log p(x|w), \tag{4.15}$$

allowing the gradient to be estimated once again through Monte Carlo simulation. However, a drawback of this approach is its high practical variance in the estimator [32].

Numerous studies in the literature aim to diminish the variance of the score function estimator, primarily employing the control variate technique. The process involves introducing a second random variable to the original estimate, maintaining the mean of the original variable while reducing its variance [35]. Essentially, when we have the Monte Carlo estimate $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$ with $x_i \sim p(x)$ and a variance of σ_f^2, we introduce a new function $b(x)$ with a known mean μ_b. We then construct a Monte Carlo estimate for the expectation of the random variable $z = f(x) - \alpha(b(x) - \mu_b)$

as $\hat{z} = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - \alpha b(x_i)) + \alpha\mu_b$. Here, α is a scalar variable determined for maximum variance reduction. The variance of the new expectation becomes $\sigma_z^2 = \sigma_f^2 - 2\alpha C_{f,b} + \alpha^2\sigma_b^2$, where $C_{f,b}$ is the covariance between $f(x)$ and $b(x)$. The optimal α is given by $\frac{C_{f,b}}{\sigma_f^2}$. The corresponding variance then becomes $\sigma_z^2 = \sigma_f^2 - \frac{C_{f,b}^2}{\sigma_b^2}$ or $\sigma_z^2 = \sigma_f^2(1 - \rho_{f,b})$, where $\rho_{f,b}$ is the correlation between $f(x)$ and $b(x)$. The last equation underscores that selecting a $b(x)$ with a high positive correlation with $f(x)$ is crucial for significant variance reduction. $C_{f,b}$ and σ_f^2 are typically estimated from the sample. Distinct estimators for the score function differ in the way they construct the control variates $b(x)$, also known as "baseline" [35].

Some common approaches involve building $b(x)$ from moving averages, calculating the Taylor expansion around $f(x)$, and using neural networks to model the baseline [151–154].

Another method to circumvent the ill-posed gradient problem is to use the LOTUS ("Law of the Unconsicuous Statistician") theorem [155]. This theorem asserts that when given an expectation $E_{p(x)}[f(x)]$ and there exists a deterministic relationship between x and another random variable ϵ, such as $x = g(\epsilon)$, we can express $E_{p(x)}[f(x)]$ as $E_{p(\epsilon)}[f(g(\epsilon))]$. Leveraging this concept, the "Reparametrization Trick" is introduced in [156]. For a continuous variable x, a reparametrization is constructed as $x = g(\epsilon, \theta)$, where typically $\epsilon \sim N(0, 1)$. This representation transforms x into a deterministic function of a noise variable ϵ with a known distribution, incorporating learnable parameters θ. It decouples the parameters from the stochastic variable and therefore leads to a convenient solution for back-propagating the derivatives through sampling operations. The derivative $\frac{\partial}{\partial w}\mathbb{E}_{p(x|w)}[f(x)]$ is then reformulated as $\frac{\partial}{\partial w}\mathbb{E}_{p(\epsilon)}[f(g(\epsilon, w))]$. This revised form once again enables a manageable Monte Carlo estimation.

The discrete stochastic variables in the computational graph of a neural network pose challenges for the reparametrization trick, as they have zero derivatives almost everywhere. To address this issue, equivalent solutions under the name "Gumbel-Softmax Trick" or "Concrete Distribution" have been proposed independently by [32]

and [33]. These approaches are based on the "Gumbel-Max Trick", initially introduced in [157], which offers a practical method for sampling from a categorical distribution. A categorical distribution with k classes can be represented using k-dimensional one-hot vectors, where the index of every non-zero entry indicates the selected class. Assuming the categorical distribution has parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$ with $\sum_{i=1}^{k} \alpha_i = 1$, a scalar g follows the Gumbel(0,1) distribution if, through inverse transform sampling, we have $g = -\log(-\log(u))$ with $u \sim U(0,1)$. We can generate k-dimensional one-hot vectors z, distributed according to the categorical distribution parameterized by $\alpha_1, \alpha_2, \ldots, \alpha_k$, if the one-hot condition is met, as

$$z = \text{onehot}\left(\arg\max_i \left(g_i + \log \alpha_i\right)\right). \tag{4.16}$$

Here, $g_1, g_2, \ldots, g_k$ are drawn from $Gumbel(0,1)$. This trick is useful in the sense that the randomness and sampling are pushed from the categorical distribution to the parameterless $Gumbel(0,1)$ variables. Any expectation involving the categorical distribution can be substituted with an expectation over the Gumbel variables using the reparametrization of z provided by the formula above. However, the outcome still poses challenges, as the onehot operation has zero derivatives almost everywhere. The "Gumbel-Softmax" or "Concrete" Distribution alleviates the limitations of the Gumbel-Max trick by introducing a continuous distribution. This distribution enables the sampling of continuous probability vectors y on the $k-1$ dimensional probability simplex Δ^{k-1}

$$y_i = \frac{\exp\left(\left(\log \alpha_i + g_i\right)/\lambda\right)}{\sum_{j=1}^{k} \exp\left(\left(\log \alpha_j + g_j\right)/\lambda\right)}, \tag{4.17}$$

for $i = 1, \ldots, k$. The parameter λ represents a temperature, and as $\lambda \to 0$, the continuous distribution approaches the categorical distribution in probability, where α_i are the parameters. Typically, an annealing mechanism for λ is employed, gradually reducing it to near-zero values during training. Both [32] and [33] note that lower temperatures, despite being closer to the actual categorical distribution, introduce more variance. Therefore, it is generally not recommended to initiate training at very small temperatures. Substituting the occurrence of the categorical variable $z \sim Categorical(\alpha_{1:k})$

with $y \sim Concrete(\alpha_{1:k}, \lambda)$ and replacing any expectation over z with expectation over g, we obtain a continuous, approximate reparametrization of the categorical distribution that mimics the categorical distribution as $\lambda \to 0$. Since the expectation now involves the parameterless Gumbel distribution, we can easily construct Monte Carlo estimators for the loss function.

The Gumbel-Softmax (Concrete) distribution has some useful mathematical properties, which have been shown in [33]. Let $x \sim Concrete(\alpha, \lambda)$ where α are location parameters with $\alpha \in (0, \infty)^n$ and λ is temperature parameter $\lambda \in (0, \infty)$, then the Concrete distribution has the following properties:

- Reparameterization: $g_k \sim Gumbel(0, 1) \to x_i = \dfrac{\exp((\log \alpha_i + g_i)/\lambda)}{\sum_{j=1}^{k} \exp((\log \alpha_j + g_j)/\lambda)}$
- Rounding: $p(x_k > x_i, k \neq i) = \dfrac{\alpha_k}{\sum_{i=1}^{n} \alpha_i}$
- Zero Temperature: $\lim_{\lambda \to 0} p(x_k = 1) = \dfrac{\alpha_k}{\sum_{i=1}^{n} \alpha_i}$

We will be using these properties actively in the next part, where we will be introducing the training algorithms of our CIGT models. We want to note that in the remaining text, we might use the naming of Gumbel-Softmax and Concrete Distributions interchangeably for the same probability distribution we now described.

4.5. Training Algorithms for CIGT

4.5.1. Approximate Training

In the CIGT framework, we adopt a distinct approach for training the Mixture of Experts loss that incorporates stochastic nodes. Specifically, we disregard $\dfrac{\partial J_C}{\partial \phi}$ in Eq. (4.12) and the second part of $\dfrac{\partial J_C}{\partial \theta}$ in Eq. (4.11) entirely, denoting the reduced gradient as $\dfrac{\partial \hat{J}_C}{\partial \theta}$. The parameters of the H units are exclusively updated based on the information gain losses at each block, given by $-\lambda_{IG} \sum_{l=1}^{L} \dfrac{\partial}{\partial \phi} IG_l$. Similarly, the parameters of the computation units (F) are updated through information gain objectives.

Algorithm 3: Approximate CIGT Training Algorithm

Result: Posteriors $p(y|Z, x, \theta)$, information gain losses IG_l and derivatives: $\frac{\partial \hat{J}_C}{\partial \theta}$, $\frac{\partial IG_l}{\partial \theta}$ and $\frac{\partial IG_l}{\partial \phi}$

while *epoch* $< N_{warmup}$ **do**

  Use uniform sampling at the routers;

  Train the rest of the network like a conventional CNN;

end

while *epoch* $>= N_{warmup}$ **do**

  Sample a minibatch $\{x_i, y_i\}_{i=1}^{B} \sim p(x, y)$;

  for $1 \leq i \leq B$ **do**

    $\hat{x}_0^i = F_0(x_i)$;

    Sample next layer's F units: $Z_1^i \sim p(Z_1^i|\hat{x}_0^i, \theta, \phi)$;

  end

  Evaluate information gain:

  $IG_1 = \mathbb{H}[p(y)] + \lambda_{balance}\mathbb{H}[p(Z_1|\theta, \phi)] - \mathbb{H}[p(y, Z_1|\theta, \phi)]$;

  Compute gradients: $\frac{\partial IG_1}{\partial \theta}$ and $\frac{\partial IG_1}{\partial \phi}$;

  while $1 \leq l \leq L$ **do**

    for $1 \leq i \leq B$ **do**

      $j = \arg\max\{Z_l^i\}$;

      $\hat{x}_l^i = F_{l,j}(\hat{x}_{l-1}^i)$;

      Sample next layer's F units: $Z_{l+1}^i \sim p(Z_{1+1}^i|\hat{x}_l^i, \theta, \phi)$;

    end

    Evaluate information gain:

    $IG_{l+1} = \mathbb{H}[p(y)] + \lambda_{balance}\mathbb{H}[p(Z_{l+1}|\theta, \phi)] - \mathbb{H}[p(y, Z_{l+1}|\theta, \phi)]$;

    Compute gradients: $\frac{\partial IG_l}{\partial \theta}$ and $\frac{\partial IG_l}{\partial \phi}$;

  end

  for $1 \leq i \leq B$ **do**

    Evaluate posterior: $p(y_i|Z_i, \hat{x}_L^i, \theta)$;

    Compute gradient: $\frac{\partial \log p(y_i|Z_i, \hat{x}_L^i, \theta)}{\partial \theta}$;

  end

end

In each CIGT layer l, the router units utilize the outputs of the computational units to construct and sample from the probability distributions $p(Z_{1+1}|\hat{x}_l, \theta, \phi)$, where $\hat{x}_l$ represents the outputs of each corresponding F unit for a particular sample in the current CIGT layer. Based on the one-hot vector samples Z_{l+1}, the corresponding F units in the next block are activated. This process is equivalent to sampling a root-to-leaf path for every sample, evaluating it, and subsequently back-propagating the corresponding error signal along the traversed path. Given that we only partially utilize the gradients of the main loss and rely on the information gain losses to train the router nodes, we term this training algorithm "Approximate Training". Additionally, we allow the routers to undergo uniform sampling for a certain number of epochs before introducing the information-gain objectives. This initial phase ensures that each route is trained with every possible sample. This is similar to our CIGN models, which use annealed thresholds during training. Uniform sampling instead of solely information-gain-based routing helps root-to-leaf experts acquire insight about the whole dataset before focusing on their coarse class groups.

In the evaluation phase, the F unit with the highest likelihood is chosen from the output of the corresponding H unit for a given sample x. This expert path is deemed the most suitable as it is trained with samples that are semantically similar to x, thereby possessing a high discriminative capability. Inference involves evaluating only a single path, rendering the algorithm more efficient than unconditional baselines.

4.5.2. The Full Training Using Gumbel-Softmax Relaxation

Alongside the approximate training, we present a training approach that we call "Full Training". In this strategy, we use the entire gradient obtained from the main loss function by utilizing the Gumbel-Softmax distribution. As a first step, we need to ensure that the router outputs that are to be passed into the Gumbel-Softmax layer are strictly positive due to the distribution´s definition. To this end, we transform the

router outputs we used in Equation 4.6 with a conventional softmax operation

$$\beta_{l,k}(x) = \frac{\exp\{(w_k^T h_l(x) + b_k)\}}{\sum_{i=1}^{K_{l+1}} \exp\{(w_i^T h_l(x) + b_i)\}}. \tag{4.18}$$

After defining the positive weights for every computational unit in the next CIGT layer as $\beta_{l,k}$ terms, we define continuous selection vectors $\hat{Z}_l$, which are the continuous relaxation of the one-hot vectors Z_l that we have used in the Approximate Training. These vectors are Gumbel-Softmax variables. When we are at CIGT layer l, the next layer´s selection vector, Z_{l+1}, is defined as

$$\hat{Z}_{l+1,k}(x) = \frac{\exp\{(\log \beta_{l,k}(x) + G_{l,k})/\tau\}}{\sum_{i=1}^{K_l} \exp\{(\log \beta_{l,i}(x) + G_{l,i})/\tau\}}. \tag{4.19}$$

The temperature parameter τ that we used for smoothing the information gain value for the Approximate Training is now used as the temperature parameter of the Gumbel-Softmax distribution as well. During the training, in each layer l, $\hat{Z}_l$ will act as a weighting parameter among $F_{l,k}$ units, which is similar to attention mechanisms in modern architectures.

The lower bound for the main loss (4.2) is now an expectation over the Gumbel-Softmax variables $\hat{Z}_l$ at each layer, and we write it as

$$\hat{J}_C(\theta, \phi) = -\frac{1}{N} \sum_{i=1}^{N} \int_{\Delta^{K_l - 1}} \log p(y_i|\hat{Z}, x_i, \theta) \prod_{l=1}^{L} p(\hat{Z}_l|x_i, \tau, \theta, \phi) d\hat{Z}. \tag{4.20}$$

The integration spans $\Delta^{K_l - 1}$, representing the probability simplex of dimension $K_l - 1$, and we have explicitly indicated the dependence on the temperature. Now, by reparametrizing $\hat{Z}$ with Gumbel samples G, we can express the equivalent loss as

$$\hat{J}_C(\theta, \phi) = -\frac{1}{N} \sum_{i=1}^{N} \int_{G} \log p(y_i|\hat{Z} = g(G, \tau, \theta, \phi), x_i, \theta) p(G) dG. \tag{4.21}$$

In this new form of the loss, the expectation is pushed onto the parameterless Gumbel variables and it allows us to calculate Monte Carlo estimates of the gradients

$$\frac{\partial L_C}{\partial \theta} = -\frac{1}{N} \sum_{i=1}^{N} \int_{G} \left(\frac{\partial \log p(y_i|\hat{Z} = g(G, \tau, \theta, \phi), x_i, \theta)}{\partial \hat{Z}} \frac{\partial \hat{Z}}{\partial \theta} + \frac{\partial \log p(y_i|\hat{Z} = g(G, \tau, \theta, \phi), x_i, \theta)}{\partial \theta} \right) p(G) dG, \tag{4.22}$$

and

$$\frac{\partial L_C}{\partial \phi} = -\frac{1}{N}\sum_{i=1}^{N}\int_G \frac{\partial \log p(y_i|\hat{Z} = g(G,\tau,\theta,\phi), x_i, \theta)}{\partial \hat{Z}}\frac{\partial \hat{Z}}{\partial \phi}p(G)dG. \tag{4.23}$$

4.5.3. Information Gain Based Routers In Full Training

The transition from categorical Z_l variables to continuous Gumbel-Softmax $\hat{Z}_l$ variables for the information gain losses at each layer is not straightforward. One option is to relax the Z_l variables in the information gain loss and replace expectations over Z_l with the probability density functions of Gumbel-Softmax variables $\hat{Z}_l$ (as defined in both [32, 33]). However, this approach requires the use of differential entropy in calculating the information gain. Differential entropy doesn't provide the same intuitive results as discrete entropy, which is necessary for grouping data into semantically similar coarse classes.

The router objective that ideally needs to be optimized is the information gain between the class labels y and the hard-route selection variable, the one-hot vector Z_l. Given a sample x, Z_l has a Categorical distribution

$$Z_l|x \sim Categorical\left(\frac{\beta_{l,1}(x)}{\sum_{i=1}^{K_l}\beta_{l,i}(x)}, \ldots, \frac{\beta_{l,K_l}(x)}{\sum_{i=1}^{K_l}\beta_{l,i}(x)}\right). \tag{4.24}$$

We have defined these weights, $\beta_{l,k}$, as in Equation 4.18. They are strictly positive real numbers. In the case of Full Training, we are not able to work directly with the one-hot routing vectors Z_l anymore since we are now using their continuous Gumbel-Softmax counterparts $\hat{Z}_l$ for routing. Therefore, we need to devise an objective function for routing using $\hat{Z}_l$ variables that will ideally still help increase the information gain between the class labels and hard-route selections.

In this case, we define a new random variable at the CIGT layer l, which we call $B_l|x,\tau$, that is distributed, given a sample x and the temperature τ, as

$$B_l|x,\tau \sim Categorical\left(\mathbb{E}[\hat{Z}_{l,1}|x,\tau], \ldots, \mathbb{E}[\hat{Z}_{l,K_l}|x,\tau]\right). \tag{4.25}$$

Here, $\mathbb{E}[\hat{Z}_{l,K_l}|x,\tau]$ is defined as the expectation of kth element of the Gumbel-Softmax

vector $\hat{Z}_l$, given x and the temperature τ. The explicit mathematical expression for this expectation is given in the following, first with the Gumbel-Softmax density and then in the second line, using the "Reparametrization Trick", defining the expectation over Gumbel variables $G = \{G_1, \ldots, G_{K_l}\}$, using the definition of the Gumbel-Softmax variable per [33], as

$$\begin{aligned}\mathbb{E}[\hat{Z}_{l,k}|x,\tau] &= \int_0^1 \hat{Z}_{l,k} p(\hat{Z}_{l,k}|x,\tau) d\hat{Z}_{l,k} \\ &= \int_{\mathbb{R}^{K_l}} \frac{\exp((\log \beta_{l,k}(x) + G_k)/\tau)}{\sum_{i=1}^{K_l} \exp((\log \beta_{l,i}(x) + G_i)/\tau)} p(G) dG, \end{aligned} \tag{4.26}$$

where $p(\hat{Z}_{l,k}|x,\tau)$ is the density of the Gumbel-Softmax variable as defined in [32, 33] and $p(G)$ is the joint distribution of the i.i.d. $Gumbel(0,1)$ variables $\{G_1, \ldots, G_{K_l}\}$, as $p(G) = \prod_{i=1}^{K_l} p(G_i)$. We dropped the dependency on the neural network parameters θ, ϕ for clarity. The justification for our decision for the definition of $B_l|x,\tau$ lies in the behavior of the expected value $\mathbb{E}[\hat{Z}_l|x,\tau]$, as the temperature τ approaches 0: As $\tau \to 0$, it can be shown that $\mathbb{E}[\hat{Z}_{l,k}|x,\tau]$ approaches to $\frac{\beta_{l,k}(x)}{\sum_{k=1}^{K_l} \beta_{l,k}(x)}$ as

$$\begin{aligned}\mathbb{E}[\hat{Z}_{l,k}|x,\tau \to 0] &= \int_{\mathbb{R}^{K_l}} \lim_{\tau \to 0} \frac{\exp((\log \beta_{l,k}(x) + G_k)/\tau)}{\sum_{i=1}^{K_l} \exp((\log \beta_{l,i}(x) + G_i)/\tau)} p(G) dG \\ &= \int_{\mathbb{R}^{K_l}} \mathbb{I}[k = \arg\max_i \{\log \beta_{l,i}(x) + G_i\}] p(G) dG \\ &= \frac{\beta_{l,k}(x)}{\sum_{k=1}^{K_l} \beta_{l,k}(x)}. \end{aligned} \tag{4.27}$$

In Equation 4.27, in the second line, we used the property of the Softmax function, which behaves as an Argmax operation in the limit of $\tau \to 0$ [33]. That means, whenever the term $\{\log \beta_{l,k}(x) + G_k\}$ is the largest one among all other $\{\log \beta_{l,i}(x) + G_i\}$, $(1 \leq i \leq K_l)$ terms, the Softmax in the limit $\tau \to 0$ will return 1 and 0 otherwise. $\mathbb{I}$ is the indicator function that returns 1 whenever the condition inside is met. The third line follows from the "Gumbel-Max" trick. This trick states that given a set of real numbers $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ where $\alpha_k \in (0, \infty)$ and $(G_1, G_2, \ldots, G_n)$ where G_i are i.i.d. Gumbel(0,1) variables, the probability that the index k maximizes the term

$\{\log \alpha_i + G_i\}$ is equal to $\frac{\alpha_k}{\sum_{i=1}^{n} \alpha_i}$ [157].

So, in the case of Full Training with Gumbel Softmax variables $\hat{Z}_l$ we have shown that in the limit $\tau \to 0$, using the expected values of $\hat{Z}_l$ vectors as the parameters of the Categorical distribution of $B_l|x, \tau$ route variables will recover the information gain maximization procedure between the class variables and selected routes as in the Approximate Train. It is also worth noting that the expected values $\mathbb{E}[\hat{Z}_{l,k}|x, \tau]$ are easy to approximate with straightforward Monte Carlo simulations since we can easily sample $\hat{Z}_l$ variables from Gumbel-Softmax-based routers.

The approximate information gain term is then calculated as

$$IG_l = \mathbb{H}\left[p(y)\right] + \lambda_{balance}\mathbb{H}\left[p(B_l|\theta, \phi)\right] - \mathbb{H}\left[p(y, B_l|\theta, \phi)\right]. \tag{4.28}$$

4.6. Sparse Mixture of Experts with CIGT

4.6.1. Motivation

As in the tree-based CIGN models, it is natural to design a multi-path inference process for the trellis-based CIGT models. The trellis model is naturally better at handling situations where a hard sample leads to uncertain routing probabilities and the data is sent to the wrong subgraphs in the CIGT. There will be a chance to recover a misrouted sample in the following layers, which is not possible with a tree structure. However, there may still be samples that consistently get misrouted at every CIGT layer due to uncertain decisions. Introducing a Sparse Mixture of Experts (sMoE) inference mode, as in the CIGNs, can help recover such cases by allowing the invocation of more than one root-to-leaf path. Similar to the CIGN models, we introduce probability thresholds for each router H_l in the CIGT architecture. Again, as in the CIGN approach, these probability thresholds are learned from a validation set, using the loss function in Equation 3.21 that balances the extra accuracy and the extra computational burden. In contrast to the CIGN approach, we will be focusing

on the Cross-Entropy Search method [34], which is similar to Bayesian Optimization in being a black-box global optimization algorithm that does not require derivatives. While we will be developing this multi-path inference mechanism for the CIGT models, the benefit that these trellis-shaped models can obtain from such a sparse mixture of experts inference approach might be more limited than tree-shaped CIGN models. The primary reason for this would be that CIGT is particularly designed with a more robust routing capability in mind. It should, by design, already handle misrouted samples more gracefully than the CIGN models do. So, the room for improvement that comes with a sparse mixture of experts inference could be humbler.

4.6.2. Cross-Entropy Search

Cross-Entropy Search is a black box optimization algorithm that requires no derivative information for the function to be optimized. The only requirement is that the target function could be evaluated in its support space with ease. The Cross-Entropy Search was originally formulated as a Rare-Event Simulation Approach [34]. A score function $S(x)$ that is defined on a search space $x \in X$ is to be optimized. This function has an optimum value of $\gamma^* = S(x^*)$. Secondly, a family of probability density functions, $f(x, w)$, parameterized with a vector of learnable parameters, $w \in W$, is defined on the search space X. The basic insight of the cross-entropy search method is that when a set of sample points $X_1, X_2, \ldots, X_N$ are sampled from an initial density function $f(x, u)$, the number of acceptable samples that are to generate values from the score function that are close enough to the optimum value γ^* is low. In other words, when we define an acceptability threshold γ_t, such that $\gamma^* - \gamma_t < \epsilon$ where $\epsilon > 0$ is an arbitrarily small difference, the likelihood of generating acceptable samples from the density function $f(x, u)$ is small as well, like

$$P_u(s(x) > \gamma_t) = \int_X f(x, u)\mathbb{I}[s(x) > \gamma_t]dx = \mathbb{E}_{f(x,u)}\left[\mathbb{I}[s(x) > \gamma_t]\right]. \tag{4.29}$$

A natural idea for obtaining many acceptable samples x, that produce scores larger than the threshold γ_t, is to optimize the density function $f(x, .)$ and obtain a parameter vector w^* such that the probability for obtaining acceptable samples becomes

much larger, namely $P_{w^*}(s(x) > \gamma_t) \gg P_u(s(x) > \gamma_t)$. It should be noted that we only have access to the density function $f(x, u)$ with the parameter vector u at the start of the procedure. Given these constraints, the maximum likelihood optimization problem can be solved with respect to the parameter vector w to obtain the optimum parameters that will generate acceptable samples with the highest likelihood possible [34]

$$\begin{aligned} w^* &= \arg\max_v \int_X f(x, u)\mathbb{I}[s(x) > \gamma_t] \log f(x, v)dx \\ &= \arg\max_v \mathbb{E}_{f(x,u)} \left[\mathbb{I}[s(x) > \gamma_t] \log f(x, v)\right]. \end{aligned} \tag{4.30}$$

This cost function means that for every x, that we can produce via our initial density function $f(x, u)$, we are going to pick the ones that satisfy the acceptability criterion $s(x) > \gamma_t$ and for these particular samples, we are going to learn a new density function such that their log-likelihood will be maximized. In practice, we are going to obtain samples drawn from $f(x, u)$, such that $X_1, X_2, \ldots, X_N \sim f(x, u)$ and we will optimize a finite approximation of the expectation in Equation 4.30, for an approximation of the optimal parameter w^* [34] as

$$\hat{w}^* = \arg\max_{\hat{v}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[s(X_i) > \gamma_t] \log f(X_i, v). \tag{4.31}$$

The primary problem with this basic approach is that since by definition it is a rare event to obtain a sample that satisfies $s(x) > \gamma_t$, we are going to need to have a very large number of samples to learn the parameter $\hat{w}^*$ in a reliable way. To overcome this problem, a two-step iterative algorithm has been proposed. A number of N samples are generated from the initial density function, $f(x, u)$. These samples are sorted based on their scores obtained from the score function $s(x)$. A $(1 - \rho)$ quantile of the best-performing samples is selected then, where the cut-off score for that quantile is named as γ_1. The optimization problem in Equation 4.31 is solved, and a vector of intermediate optimum parameters is obtained with the name w_1. In the next step, this time by using w_1, a new set of samples is generated from the density function $f(x, w_1)$. This iterative procedure continues until either a set number of iterations is reached or the $(1 - \rho)$ quantile cut-off score stops improving for a certain number of iterations, indicating that an approximation of the optimal parameters may have been reached.

Algorithm 4: The Cross-Entropy Search Algorithm

Number of total iterations : T

Number of iterations without improvement: M

Initial parameters : w_1

Quantile : ρ

Score function : $s(.)$

Result: Approximately optimum parameters $\hat{w}^*$

while $i < T$ **do**

Sample $X_1^i, X_2^i, \ldots, X_N^i \sim f(x, w_i)$;

Sort samples according to the scores: $s(X_1^i), s(X_2^i), \ldots, s(X_N^i)$;

Determine (1-ρ) quantile performance γ_i;

Solve the optimization problem:

$w_{i+1} = \arg\max_{\hat{v}} \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[s(X_j^i) > \gamma_i] \log f(X_j^i, v)$;

Break if γ_i has not been improved in the last M iterations;

end

It should be noted that the optimization step is akin to Maximum Likelihood Estimation, where the estimation is only made for the subset of samples that satisfy the $s(X_j^i) > \gamma_i$ criterion [34]. The success of the algorithm depends on the structure of the score function $s(.)$ and the quantile parameter ρ, primarily. If $s(.)$ is a very expensive function to evaluate, the algorithm might prove to be infeasible since the number of samples needed per iteration might be high to execute the Maximum Likelihood Estimation step reliably.

4.6.3. Cross-Entropy Search for CIGT Threshold Determination

We adopt the Cross-Entropy search paradigm for determining the optimum thresholds in the CIGT setting. For every parent node-child node connection in the trellis structure, we require a threshold parameter. For example, considering the CIGT-[1,2,4] structure in Figure 4.1, we would need 10 parameters; 2 are required from the root computation unit $F_{0,0}$ to the $F_{1,0}$ and $F_{1,1}$ connections. 4 is required from the connection from the unit $F_{1,0}$ to $F_{2,0}, F_{2,1}, F_{2,2}, F_{2,3}$, and similarly, 4 is also needed from the

unit $F_{1,1}$ to the next layer units $F_{2,0}, F_{2,1}, F_{2,2}, F_{2,3}$. Another option to keep the search space small might be using shared thresholds for each layer. Namely, when we have K_l computation units at layer l and K_{l+1} units in the following layer, we would need to learn $K_l \times K_{l+1}$ thresholds. When sharing thresholds, all K_l units will share the same thresholds for the connection into the next layer.

Our search space for the cross-entropy search algorithm would consist of closed intervals since we are looking for probability thresholds; that would be only sensible when in the interval $[0, 1]$. The dimension of the search space is equal to all thresholds that are to be used in the CIGT algorithm, depending on whether the computation units will share the same set of thresholds for every CIGT layer. When a CIGT layer has K_l computation units, in our algorithm we use the interval $[0, 1/K_l]$ for the thresholds corresponding to the connections into these units. An important element of the cross-entropy search method is to determine a density function $f(x, w)$. In our case, the probability density function is going to be used for sampling the CIGT routing probability thresholds. We have two desirable properties for the probability density function family we will be using:

- The density should be multi-modal. We consider this important since the complex parent-child node connections in the trellis structure of the CIGT might not be sufficiently representable with a uni-modal distribution. Ideally, the density function could be a mixture of uni-modal densities to be able to capture the complexity.
- The density functions should be continuous, and their support should be finite to be compatible with the intervals of our probability thresholds.

Our first consideration was using a mixture of beta distribution densities. The beta distribution is defined on the interval $[0, 1]$ hence it is suitable for use with probability thresholds. However, it turns out that the mixture model of beta distributions suffers from a couple of problems. Firstly, the maximum likelihood estimation of individual beta distributions is known to suffer from singularity problems around the

interval ends and stability issues where an EM-based optimization of the mixture of beta distributions aggravates these problems [158]. Secondly, even though precautions are taken for the numerical problems in the maximization algorithm, the maximization itself is based on complex numerical optimization algorithms since no closed-form solution of the beta distribution parameters exists in a maximum-likelihood setting [158].

Our second consideration for density functions is Gaussian Mixture Models. Gaussian distributions have closed-form solutions for their mean and standard deviation parameters under a maximum likelihood setting that facilitates the maximization step of the EM algorithm [130]. No expensive numerical optimizations are needed for Gaussian mixtures. While they are numerically and algorithmically well-behaved, the problem with Gaussian mixtures is that they have the whole real number line as their support. This problem can be overcome by applying a transformation to the variables sampled from a GMM, which maps the real line into a finite interval. The sigmoid function is a suitable candidate for this purpose.

For the cross-entropy setting, we decided to use Gaussian Mixture Models, whose outputs are going through transformations that map the Gaussian variables into the real line. Namely, we sample a vector x from a Gaussian Mixture Model as

$$x \sim \sum_{n=1}^{K} \pi_n N(x|\mu_n, \Sigma_n). \tag{4.32}$$

Assuming our search space is D-dimensional, for every dimension of x, we define a sigmoid-based transformation that takes x into a predefined finite interval

$$y_i = u_i(x_i) = (b_i - a_i)\frac{1}{1 + \exp\{-x_i\}} + a_i. \tag{4.33}$$

In Equation 4.33, (a_i, b_i) pairs are the probability threshold interval endpoints for every dimension. We define a vector-valued function, which is to be applied to the GMM vector x, which takes its elements from the real line and maps them into predetermined intervals as

$$y = u(x) = (u_1(x_1), u_2(x_2), \ldots, u_D(x_D)). \tag{4.34}$$

In this case, the vector y will be our set of thresholds for Sparse Mixture of Experts-based routing mechanism. To use y in the Cross-Entropy Search framework, we need to calculate its probability density function (pdf). Assuming a random variable x with a known pdf, $f(x)$, goes through an invertible transformation $y = u(x)$, such that $v(u(x)) = x$, we have the pdf for the variable y

$$f(y) = |J| f(v_1(y_1), v_2(y_2), \ldots, v_D(y_D)). \tag{4.35}$$

$|J|$ is the determinant of the Jacobian matrix, with entries $J_{i,j} = \dfrac{\partial v_i(y_1, y_2, \ldots, y_D)}{\partial y_j}$. In our case, the inverse functions v_i and their derivatives with respect to y_i are defined as

$$\begin{aligned} v_i(y_i) &= \log\left(\frac{y_i - a_i}{b_i - y_i}\right) \\ \frac{dv_i}{dy_i}(y_i) &= \frac{a_i - b_i}{(y_i - a_i)(y_i - b_i)}. \end{aligned} \tag{4.36}$$

Finally, observing that the Jacobian matrix J is square, diagonal, and such that its determinant is equal to the product of its diagonal elements, we can write the full expression for the pdf of the y variables as

$$\begin{aligned} f(y) = &\left(\prod_{i=1}^{D} \frac{a_i - b_i}{(y_i - a_i)(y_i - b_i)}\right) \times \\ &\left(\sum_{k=1}^{K} \pi_k N\left(\log\left(\frac{y_1 - a_1}{b_1 - y_1}\right), \log\left(\frac{y_2 - a_2}{b_2 - y_2}\right), \ldots, \log\left(\frac{y_D - a_D}{b_D - y_D}\right) | \mu_k, \Sigma_k\right)\right). \end{aligned} \tag{4.37}$$

The logarithm of the probability density function $f(y)$ leads to simple maximization during Cross-Entropy search iterations: The Jacobian term does not depend on the GMM parameters; hence, it can be left out. The remaining logarithm term with the actual GMM pdf can be efficiently optimized with the EM algorithm by converting the y variables back into the original x variables.

We apply a preprocessing operation before conducting the Cross-Entropy search for the optimal probability threshold values. Without loss of generality, assuming a CIGT-[1,2,4] architecture as in Figure 4.1, three sets of routing probabilities are

generated by the network on a dataset of size N: A $N \times 2$ matrix, $A_{0,0}$ for routing probabilities of the samples from the node $F_{0,0}$ into $F_{1,0}$ and $F_{1,1}$. This is followed by a $N \times 4$ sized probability matrix from $F_{1,0}$, $A_{1,0}$, into $F_{2,0}, F_{2,1}, F_{2,2}, F_{2,3}$, and likewise a $N \times 4$ sized probability matrix for the node $F_{1,1}$, $A_{1,1}$, again into $F_{2,0}, F_{2,1}, F_{2,2}, F_{2,3}$. Note that $A_{i,j}$ matrices can also contain raw activations that would generate the routing probabilities after a softmax is applied to them, row-wise: $\hat{A}_{i,j} = Softmax\left(A_{i,:}\right)_j$. In the following discussion, we are going to consider the $A_{i.j}$ matrices as such raw activation matrices for routing. We observe that the routing probabilities, on average for the whole dataset, can be very certain for the path they are choosing: the selected path´s probability becomes almost 1 while other probabilities become 0. This causes numerical difficulties when trying to determine probability thresholds with the Cross-Entropy search since the actual search space would be confined around very small neighborhoods of 0 and 1. Overall, if the routing probabilities could be more balanced on average without changing the routing decisions, this could facilitate the Cross-Entropy optimization algorithm. We first define the routing probability entropy for a decision node n and a particular sample i as

$$E_{n,i}(T) = -\sum_{k=1}^{K} \left(\frac{a_{i,k}/T}{\sum_{j=1}^{K} a_{i,j}/T} \right) \log \left(\frac{a_{i,k}/T}{\sum_{j=1}^{K} a_{i,j}/T} \right). \tag{4.38}$$

In this equation, $a_{i,k} \in A_n$ indicates the activation for the sample i for the k.th child node in the next CIGT layer. These values are fed into Softmax layers in the CIGT architecture, and the actual routing probabilities are obtained. T is the temperature parameter that determines the certainty of the routing decision. This should not be confused with the τ temperature that we used in the training of CIGT; T is only used during the Cross-Entropy search for optimal probability thresholds; hence, it is a post-processing term.

Next, we calculate the variance of the $E_{n,i}(T)$ terms for all samples in the dataset for the node n

$$Var_n(T) = \frac{1}{N} \sum_{i=1}^{N} (E_{n,i}(T) - \mathbb{E}[E_{n,i}(T)])^2. \tag{4.39}$$

When we maximize the variance of the routing entropies $Var_n(T)$ with respect to the temperature T, as $T^* = \arg\max_T Var_n(T)$, we obtain routing probabilities that are overall less certain and more spread out. This later helps with the Cross-Entropy search that follows. We can see the effect of this operation in Figure 4.2. Before the optimization, nearly the whole dataset was making completely certain routing decisions at the current node. After maximizing the variance using the temperature variable T, the entropies become much more spread. This also means that the routing probabilities are distributed in a more balanced way and are not snapped to 0 and 1 completely. We deem this a necessary step before applying the Cross-Entropy search for threshold optimization.

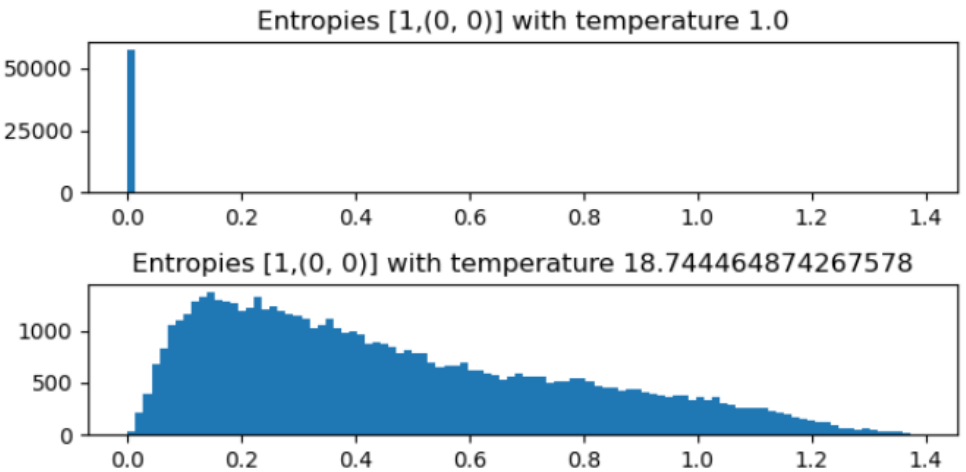


Figure 4.2. The effect of the variance maximization on the distribution of routing probabilities.

For the score function, we use the same term as in sMoE-CIGNs, provided in Equation 3.21. This function strikes a balance between accuracy and additional computation based on a hyperparameter λ. The hyperparameter λ helps to control the trade-off, ensuring that the model achieves high accuracy without incurring excessive computational costs. Finally, the Cross-Entropy search algorithm for determining the probability threshold is formulated as shown in Algorithm 5.

Algorithm 5: The Cross-Entropy Search Algorithm for sMoE-CIGT

Decision Nodes of CIGT : F_n

Validation Dataset : D

Number of total iterations : T

Number of iterations without improvement: M

Initial parameters : w_1

Quantile : ρ

Score function : $s(.)$

Probability Density Function : $f(y|w_1)$, Sigmoid GMM as in Equation 4.37

Result: Optimum routing thresholds ρ_o

Calculate variance maximizing temperatures for every decision node, by maximizing the term in Equation 4.39: $T_n = \arg\max_v Var_n(v)$ for every F_n;

Apply T_n to every routing probability before start;

while $i < T$ **do**

  Sample $\rho_1, \rho_2, \ldots, \rho_N \sim f(y|w_i)$;

  $S_i = []$;

  for $\rho_j \in \{\rho_1, \rho_2, \ldots, \rho_N\}$ **do**

    Evaluate the sampled threshold:

    $s(\rho_j) = \mathbb{E}_{(x,y)\sim D}[\lambda L_{classification}(x, y, \rho_j) + (\lambda - 1)L_{MAC}(x, \rho_j)]$;

    $S_i.add(s(\rho_j))$

  end

  Sort samples according to the scores: $s(\rho_1), s(\rho_2), \ldots, s(\rho_N)$;

  Determine (1-ρ) quantile performance γ_i;

  Solve the optimization problem:

  $w_{i+1} = \arg\max_{\hat{v}} \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[s(\rho_j) > \gamma_i] \log f(y|w_i)$;

  Break if γ_i has not been improved in the last M iterations;

end

4.7. Experiments

We carried out experiments on the MNIST [144], Fashion-MNIST [29], and CIFAR-10 [30] datasets. LeNet-based models, as described in [26], were employed for

the MNIST and Fashion-MNIST datasets. For CIFAR-10, a ResNet110-based model was utilized, comparable to those in [4] and [5].

4.7.1. Experiment Design

Our experimental approach involves establishing baseline architectures and transforming these structures into CIGT architectures by incorporating information-gain routing blocks and dividing convolutional layers into routes. In each forward pass, our CIGT network corresponds to a slimmer version of the baseline network and the routing blocks. To ensure a fair comparison, we have maintained identical layer configurations in the CIGT networks as those in the baseline architectures.

4.7.2. MNIST Experiments

For MNIST, we adopt a LeNet baseline featuring 20 feature maps with 5×5 convolutional kernels, succeeded by 50 feature maps of the same kernel size, followed by 500 and 10-dimensional dense layers. Our root-to-leaf experts closely resemble those in [26], wherein we reduce the feature map count of the second convolutional layer to 15 and decrease the dimensionality of the subsequent dense layer to 25.

The CIGT structure incorporates three routing layers. The first layer contains the baseline's first convolutional layer in its entirety. The second CIGT layer comprises two units of convolutions with 15 feature maps. The third block consists of four units of dense layers. We refer to this configuration as $CIGT - [1, 2, 4]$, denoting the number of routing units. The routers are 16-dimensional dense layers. We utilized the SGD optimizer with a momentum of 0.9, a batch size of 125, and a total of 100 epochs for each training.

We select $\lambda_{balance} = 2$. The remaining hyperparameters are employed following the values in [26], except for the absence of a distinct weight decay term for routers. We use the same annealing procedure that has been used for the temperature parameter

in [26] for the Full Training mode. In our evaluations, we compare CIGT against the thick baseline, the single root-to-leaf expert (slim), and a CIGT variant with random routing instead of information gain. We also present the results of CIGT models with Approximate and Full Training modes. The results for all our CIGT experiments represent the average of five distinct runs.

Table 4.1. MNIST Results. Each sample in the CIGTs visits a LeNet (slim) network with router blocks. CIGT-(R*) uses a CIGT-[1,2,4] architecture with random routings.

Method	Avg Acc.	Avg # MAC	Parameters
LeNet (Thick)	99.25%	4.018×10^6	1256080
LeNet (Slim)	99.20%	1.057×10^6	26695
CIGT-(R*)	99.29%	1.057×10^6	92276
CIGT-[1,2,4] - Approx. Training	**99.40%**	1.057×10^6	92276
CIGT-[1,2,4] - Full Training	**99.33%**	1.057×10^6	92276
CIGN [26, 27]	99.36%	1.057×10^6	92276
ANT [108]	99.36%	$\approx 5.0 \times 10^6$	-
DNDF [15]	99.30%	$\approx 4.0 \times 10^6$	-

Table 4.2. CIGT vs structured pruning methods comparison (on MNIST and LeNet).

Model	**Mean Acc.**	**Avg # MAC**	**Dense%**
LeNet Thick Baseline	99.25%	4.018×10^6	100%
CIGT-[1,2,4] - Approx. Training	**99.40** %	1.057×10^6	5.96%
SSL-1 [111]	99.20%	0.780×10^6	37.70%
SSL-2 [111]	99.00%	0.432×10^6	23.92%
GBD 4% Density [112]	94.00%	$\mathbf{0.074 \times 10^6}$	4.35%
VD [22]	99.0%	0.137×10^6	**0.36**%
BC [114]	99.0%	0.408×10^6	2.80%
VIB [115]	99.0%	0.244×10^6	0.83%

Our results that are presented in Table 4.1 suggest that our method achieves improved accuracy not only by reducing overfitting with fewer parameters, as evidenced by the slim version of the baseline failing to achieve comparable results. Moreover, our experiments with random routing demonstrate that our learned routing mechanism goes beyond functioning as a mere dropout mechanism for convolutional filters. Our method outperforms random routing since classes with similar meanings tend to follow similar paths in our CIGT network.

CIGT, in the case of MNIST, outperforms our analogous tree-based conditional model, CIGN [26]. Additionally, we compare CIGT with Adaptive Neural Trees [108] and Deep Neural Decision Forests [15], both employing the same LeNet model as their backbone on MNIST. CIGT achieves superior performance compared to these models while utilizing fewer MAC operations per sample.

We conducted a comparison between the CIGT model and several widely recognized structured network pruning algorithms. The LeNet architecture and the MNIST dataset are commonly used as test benchmarks for network pruning algorithms. In our tests, we compare CIGT to other methods by looking at the number of average multiplication and accumulation (MAC) operations per sample, the model's accuracy, and the degree of sparsification, which is shown by the ratio of the parameters that were used to the thicker baseline.

The outcomes presented in Table 4.2 depict a comparison with various related structured pruning algorithms. These pruning algorithms initiate the process with a thicker architecture, conducting pruning at the level of convolutional filters and neurons in the dense layers. Furthermore, fine-tuning is necessary for the resulting pruned model. In contrast, CIGT commences with predefined routes in the original network, with each root-to-leaf route corresponding to a thinner network.

The results reveal that methods like Variational Dropout (VB) [22] and Variational Information Bottleneck (VIB) [115] achieve substantial levels of sparsification,

albeit at the expense of reduced model performance. CIGT achieves a comparatively lower level of sparsification at 5.96% when employing 2 routing layers of 2 units each. In this setup, a single root-to-leaf route serves as the natural lower bound for sparsification, as it doesn't involve pruning at the granularity of filters or neurons. Despite this, CIGT achieves the highest accuracy among the pruning methods, even surpassing the original thick model. Notably, the routing mechanisms within CIGT are independent of pruning methods and can be utilized in conjunction for additional sparsification.

An interesting result of the MNIST experiments is that the Approximate Training approach performed better compared to Full Training. Our hypothesis about this realization is that MNIST is a very easy dataset, and it is easy to overfit on it. In the Full Training mode, we backpropagate additional error signals from the mixture of experts loss, which could have caused the model to overfit due to an increased amount of information about the data. We did not conduct sMoE experiments with MNIST, since, as we have experienced in CIGN models, the benefit of multi-path inference with the MNIST dataset is meager at best, and moreover, we are anticipating lower benefits of this approach for CIGT models.

4.7.3. Fashion-MNIST Experiments

For the Fashion-MNIST tests, we utilized a LeNet CNN model as our baseline, comprising 3 convolutional and 3 fully connected layers. The convolutional layers have 32, 64, and 128 filters, with 5×5 filters and 2×2 max-pooling layers following each convolutional layer. The dense layer sizes are 1024, 512, and 10, respectively. The CIGT-[1,2,4] setting, similar to the MNIST experiments, was employed. The first convolutional layer, with 32 feature maps, serves as the initial block. The second layer, featuring 64 convolutional feature maps, is divided into two routes, each with 32 feature maps. The number of feature maps for the third convolutional layer is adjusted to 32, and the dimensions of the subsequent two fully connected layers are set to 128 and 64. These layers and the final classification layer constitute each of the four routing units in the final block. Throughout the experiments, we aimed to maintain

the network corresponding to a single path as similar as possible to the root-to-leaf networks in [27], considering computation (number of MACs) and parameter count for comparison purposes.

The decision networks receive input from the outputs of the first and second convolutional layers. Global Average Pooling is applied to these outputs, followed by a 128-dimensional fully connected layer that generates the input feature for the information gain routers.

Table 4.3. Fashion-MNIST results. Each sample in the CIGTs visits a network equivalent to LeNet (slim) plus router blocks. CIGT-(R*) network uses an equivalent CIGT-[1,2,4] architecture but routings are made randomly. AT means approximate training and FT means full training.

Method	Avg Acc.	Avg # MAC	Parameters
CNN (Thick)	92.25%	6.522×10^6	1771018
CNN (Slim)	91.65%	2.606×10^6	73418
CIGT-(R*)	92.09%	2.659×10^6	294702
CIGT-[1,2,4] AT	**92.52**%	2.659×10^6	294702
CIGT-[1,2,4] FT	**92.55**%	2.659×10^6	294702
CIGN [27]	92.37%	3.154×10^6	368814
sMoE-CIGN $\lambda = 1.0$ [27]	92.72%	4.448×10^6	368814
sMoE-CIGN $\lambda = 0.99$ [27]	92.63%	3.387×10^6	368814
sMoE-CIGN $\lambda = 0.9$ [27]	92.52%	3.165×10^6	368814
sMoE-CIGT-[1,2,4] FT $\lambda = 1.0$	**92.65**%	3.201×10^6	294702
sMoE-CIGT-[1,2,4] FT $\lambda = 0.99$	**92.63**%	2.751×10^6	294702
sMoE-CIGT-[1,2,4] FT $\lambda = 0.9$	**92.53**%	2.698×10^6	294702

We conducted experiments with the Approximate Training mode, Full Training mode, and sMoE inference modes. For optimization, we utilized an SGD optimizer with a momentum of 0.9. Each model underwent training for 125 epochs with a batch size of 125, starting with a learning rate of 0.01. In the initial 25 epochs, the network

was trained without conditional routing as a warm-up phase, akin to the annealing approach in [27]. The learning rate was halved at iterations 27000 and 42000 and multiplied by 0.1 at iteration 52000. Dropout was employed after each fully connected layer for regularization, with a chosen probability of 0.3 and $\lambda_{balance} = 2$. Empirically, we set the routing loss coefficient λ_{IG} to 0.7 to balance the magnitudes of classification and routing losses around the same magnitudes. The Softmax temperature decay procedure remained the same as in the MNIST experiments.

All results related to CIGT in Table 4.3 represent the averages of 10 different runs with no augmentation. The presence of random routing within the CIGT structure provides a regularization effect. However, the use of actual information-gain routing yields superior results over random routing, highlighting that the performance improvement is not solely attributed to the reduction of overfitting. Like the MNIST experiments, we were not able to see a significant difference between Approximate Training and Full Training approaches, the difference between the two is negligibly small. This tells us that the extra signals that are backpropagated from the Mixture of Experts loss do not contribute to generalization. From another point of view, this can be attributed to the effectiveness of information-gain-based routers: They already convey enough discriminative information to the system such that the ignored gradients do not contribute an extra amount of information.

We additionally assess the CIGT model in comparison with the CIGN and sMoE-CIGN models presented in [27], utilizing similar architectures on Fashion-MNIST. While sMoE (Sparse Mixture of Experts) models allow for multiple-path routing during inference when routers exhibit higher entropy than a specified threshold, our conventional CIGT models adhere strictly to a single root-to-leaf path [27]. Despite this constraint, the Trellis network achieves performance equivalent to the sMoE-CIGN model with multi-path inference capability at a lower inference cost. In Fashion-MNIST experiments, we also involved the Sparse Mixture of Experts inference for CIGT models, where we learned the thresholds from a validation set. For the Cross-Entropy algorithm we used in this procedure, we have used Sigmoid Gaussian Mixtures with 3

components. The optimization loop in the Algorithm 5 has been iterated 10 times. At every iteration, we sampled 10,000 thresholds, and the quantile variable ρ was set to 0.05. As the validation set, we again used the training set that we have used to train the CIGT model. However, we added Cut-Out augmentation to the images during the Cross-Entropy optimization this time, at a cut ratio of 0.25 [159]. Cut-out randomly selects a rectangular area of an input image with a size that is determined by the cut ratio and either sets this area to 0 or fills it with random noise. By using such a powerful augmentation, we wanted to artificially lower the accuracy of the trained model on the training set, which is around 98% at the end of the training. This is already a high percentage compared to the test accuracy, meaning most of the samples are already classified correctly by the usual single-path execution. In that case, the network might not have to benefit from the multiple-path inference.

The results show that the improvement from using an extra Sparse Mixture of Experts mode, allowing multiple paths during inference, is minimal compared to single-path inference. The sMoE-CIGT and sMoE-CIGN architectures achieve similar accuracies at the same λ level, balancing accuracy and computation. However, at $\lambda = 1.0$, the tree-based sMoE-CIGN model performed better. Multi-path routing benefits the trellis structure less than the tree structure because the trellis architecture already has a more powerful routing capability. CIGT models without sMoE performed better than CIGN models without sMoE. Additionally, in our experiments with Cross-Entropy Search on the Fashion-MNIST dataset, the routing units of trained CIGT models showed very large activation magnitudes, resulting in highly certain routing decisions. This can be seen in Figure 4.3. In this figure, each image has been generated solely by the outputs of a specific computational unit. In the respective order from top to bottom, the first image shows the routing statistics of the computational unit $F_{0,0}$ when each sample is processed by it and fed into the router H_0. Similarly, the second image shows the routing statistics when every sample follows the computational units $F_{0,0}$ and $F_{1,0}$ in CIGT layers 0 and 1 and then gets fed into the router H_1. The third image shows the statistics of the routers when each sample follows the path of $F_{0,0}$ and $F_{1,1}$. (For reference, the network topology is the same as the Figure in 4.1).

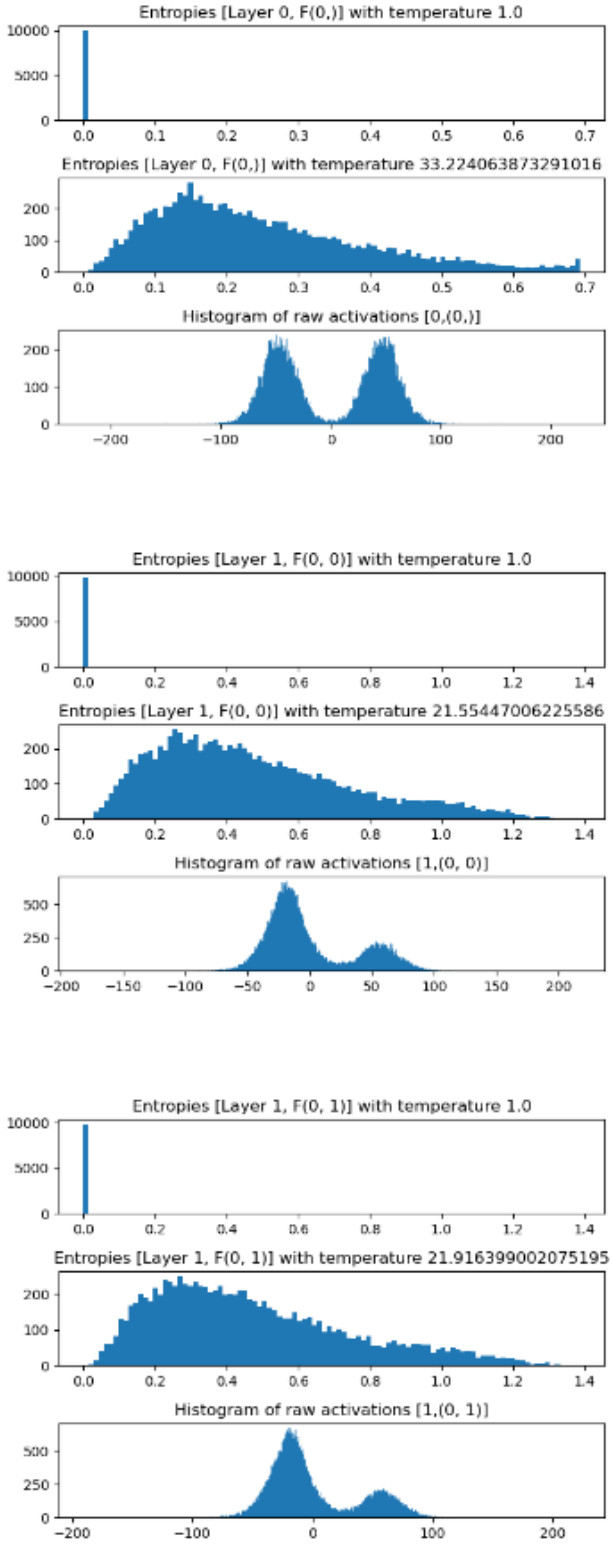


Figure 4.3. Statistics that describe the behavior of the routers in the Fashion-MNIST CIGT model. Each image is generated solely with outputs of a specific computation unit.

In every image, the first row shows the distribution of the entropies of the routing probabilities for every sample in the test set in the corresponding router unit. Namely, this graph shows the distribution of the entropy terms defined in Equation 4.38. The second row shows the distribution of the entropies after an optimal temperature value is found by solving the optimization problem in Equation 4.39. Finally, the third row shows the distribution of the unnormalized routing activations for every sample. The first rows show that the routing decisions were taken very confidently. This is also apparent in the third row, which shows a high concentration of activations that are far from 0. This can be thought of as a sort of overfitting, where router units become too confident, even for hard-to-classify samples. This conflicts with our primary insight when building the Sparse Mixture of Experts inference approach; hence, the multi-path inference fails to identify difficult samples. A possible solution might be a regularization term that discourages the routing probabilities from becoming too certain; in other words, the variance term in Equation 4.39 can be explicitly added to the total neural network loss function as a further regularizer for the routing units.

4.7.4. CIFAR-10 Experiments

In our CIFAR-10 experiments, we adopted the ResNet110 backbone employed in previous works such as [4, 5]. The CIGT networks were configured with [1,2,2] and [1,2,4] routing options. In the first routing block (1 unit), we retained 9 ResNet blocks, each comprising two convolutional layers with 3×3 kernels, followed by the addition of the block input via shortcut connection. No bottleneck layer was utilized in these blocks [39]. Each ResNet block contained 16 feature maps. The second routing block featured 2 units, each containing 27 ResNet blocks. The first 9 blocks had 12 feature maps, and after these, a 2×2 stride was applied, reducing the feature map dimensions. The subsequent 18 ResNet blocks maintained 16 feature maps. The final routing block, with either 2 or 4 units based on the CIGT structure, comprised units with 18 ResNet blocks. In the initial ResNet block of these units, a 2×2 stride was applied. For routers, we employed Convolutional Block Attention Modules (CBAM), introduced in [160]. After passing through any of the units in a routing block, the sample underwent 3

layers of CBAM with a 4-reduction rate and 4×4 average pooling. Following the CBAM layers, the information gain for the sample was calculated.

Table 4.4. CIFAR-10 results. Each sample in the CIGTs visits a network equivalent to R-110 (slim) plus router blocks. CIGT-(R*) network uses an equivalent CIGT-[1,2,4] architecture but routings are made randomly. AT means approximate training and FT means full training.

Method	Avg Acc.	Avg # MAC	Parameters
R-110 (Thick)	93.39%	186.8×10^6	1730714
R-110 (Slim)	91.76%	78.2×10^6	235218
CIGT-(R*)	92.32%	78.2×10^6	644478
CIGT-[1,2,4] AT	**93.81**%	78.8×10^6	644478
CIGT-[1,2,4] FT	**93.93**%	78.8×10^6	644478
CIGT-[1,2,2] AT	**93.26**%	78.8×10^6	475112
CIGT-[1,2,2] FT	**93.38**%	78.8×10^6	475112
sMoE-CIGT-[1,2,4] AT $\lambda = 1.0$	**94.11**%	83.3×10^6	644478
sMoE-CIGT-[1,2,4] AT $\lambda = 0.99$	**94.09**%	81.24×10^6	644478
sMoE-CIGT-[1,2,4] AT $\lambda = 0.9$	**94.05**%	79.5×10^6	644478
AIG-110 [4]	94.24%	153.3×10^6	$\approx 1.74 \times 10^6$
CSGS-110 [5]	94.36%	121.5×10^6	$\approx 1.74 \times 10^6$

We use average pooling and the CBAM reduction rate parameter to keep the routers' processing overhead as small as possible. The motivation to use CBAM layers is to utilize their attention mechanisms for increased router performance.

In our experiments with ResNet-110-based CIGT models, we employed minibatches with a size of 1024. To enhance the generalization of routers, each minibatch was split into two parts: The first half of each minibatch underwent standard CIFAR-10 augmentations, including random crop and random horizontal flip, and was utilized for training the classifiers. The second half of the minibatch was subjected to RandAugment [161] augmentations, selecting two random augmentations per image from

a predefined pool. These heavily augmented samples were employed for training the routers using the information gain loss. We utilized an SGD optimizer with an initial learning rate of 0.1. At the 950th and 1350th epochs, the learning rate was reduced by a factor of 0.1. The entire training process spanned 1750 epochs, with the first 350 epochs serving as a warm-up phase. During this warm-up period, routers were continually trained, but routing decisions were made randomly. This initial phase ensured that each path in the Trellis structure encountered samples from every class. A weight decay term of 0.0005 was applied. The balance coefficient $\lambda_{balance}$ for the CIGT-[1,2,2] model remained at 1, while for the CIGT-[1,2,4] model, we adjusted it to $\lambda_{balance} = 5$ since routing into four paths in the last block could potentially lead to load-balancing challenges during training. Throughout all experiments, λ_{IG} was set to 1. The above procedure has been used for Approximate Training and Full Training options, where the temperature decay term, τ is used for the Gumbel-Softmax variables in the case of the Full Training.

The results presented in Table 4.4 demonstrate a significant performance improvement of CIGT-[1,2,4] over the Thick ResNet-110 baseline, achieving superior results while utilizing only 41.8% of the computation load and 37% of the total parameters. It's noteworthy that the CIGT-[1,2,2] variant exhibits a performance drop compared to CIGT-[1,2,4]. In addition to the evident extra routing capability in the latter model, setting $\lambda_{balance} = 5$ in CIGT-[1,2,4] appears to contribute to stabilizing the training process, leading to further performance gains.

When compared to Adaptive Inference Graphs (AIG) [4] and Channel Selection with Gumbel Softmax (CSGS) [5], both of which present results with ResNet-110-based models on CIFAR-10, our best model falls short. This performance gap can be attributed to the different granularities of the competitor methods, as both AIG and CSGS operate with greater flexibility in terms of skipping computations.

AIG possesses the capability to skip individual ResNet blocks, and CSGS goes a step further by selecting specific feature maps to ignore using Gumbel-Softmax sam-

pling. In contrast, CIGT skips entire blocks of computation, offering a modular computation approach suitable for scenarios where each block is processed on a distributed computation server. Although this approach constrains the search space for model sparsity, it enhances inference efficiency. AIG-110 utilizes 82% and CSGS-110 uses 65% of the computation load of the thick model, whereas CIGT requires only 41.8% and achieves superior performance. Additionally, CIGT models tend to be more compact; both AIG-110 and CSGS-110 leverage the original thick model solely for adapting it to conditional computation during inference. Therefore, they don't allow any memory footprint reduction. However, CIGT models are designed to be compact.

Our top-performing Approximate Training model demands only 37% of the parameters compared to the original ResNet-110. Furthermore, these techniques, including CIGT, are orthogonal to each other. In theory, one can potentially implement the Gumbel-Softmax Channel Selection approach proposed in [5] within a Trellis-shaped CIGT model.

In the case of CIFAR-10 and Resnet 110, using Full Training instead of Approximate Training provided a slight advantage to generalization performance. Using error signals from the whole of the Mixture of Experts loss function seems to contribute to model performance, even though at a relatively small amount. A possible explanation of this can be that since Resnet 110 is a much larger network compared to LeNet-based ones, the model has the capability to acquire extra useful discriminative information.

Another interesting observation concerns the application of the Sparse Mixture of Experts approach. Similar to the Fashion-MNIST experiments, we derive the thresholds directly from the training set. However, in this case, we enhance the training data by applying the Cut-Out augmentation technique, which involves masking out random sections of the images to improve model robustness and generalization. For this approach, we utilized Sigmoid Gaussian Mixtures with three components. We trained the Cross-Entropy Search algorithm for 10 iterations, optimizing it to effectively handle the augmented dataset and refine the thresholds for better performance.

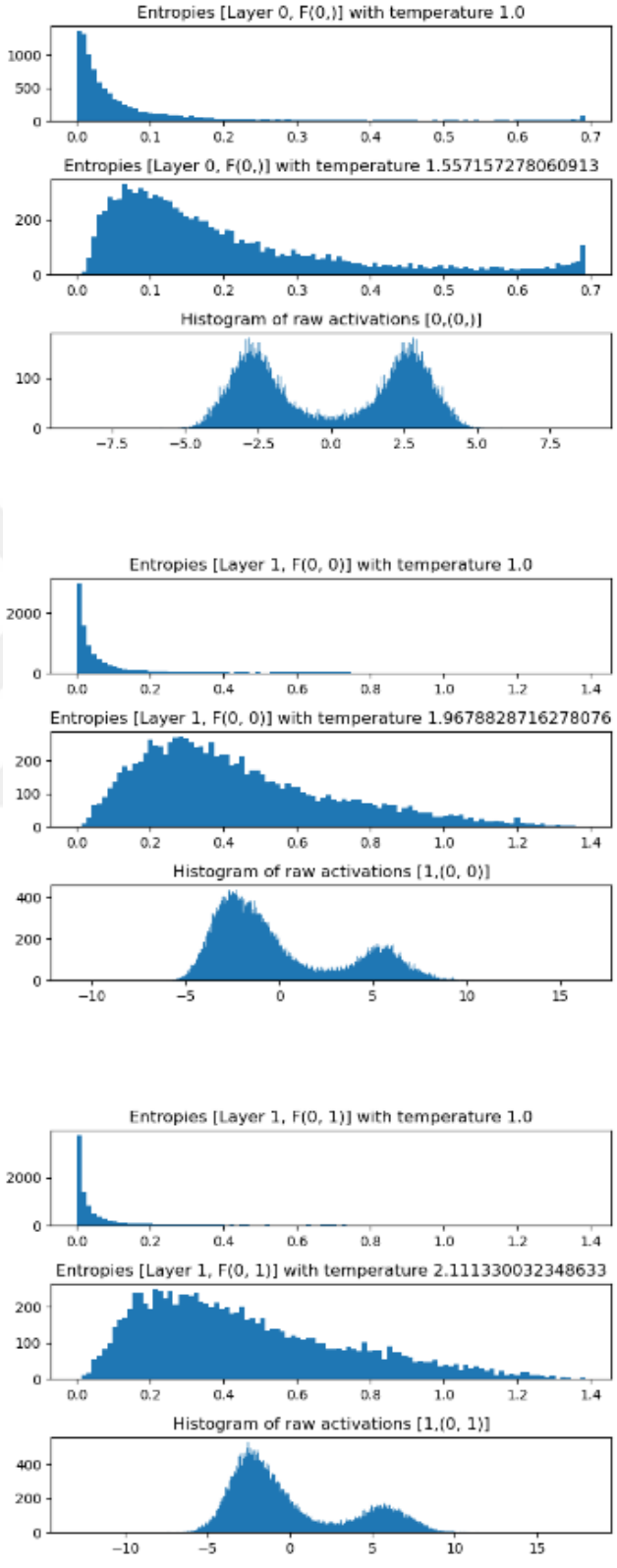


Figure 4.4. Statistics that describe the behavior of the routers in the CIFAR-10 CIGT model. Each image is generated solely with outputs of a specific computation unit.

It should be noted that we always use a pre-trained network for the Cross-Entropy algorithm in all of our experiments. We use 10000 samples and use a quantile of $\rho = 0.05$. The Cut-out ratio is kept at 0.25. We observe that the gains from the sMoE approach for Resnet 110 and CIFAR-10 are more generous compared to Fashion-MNIST. At the level of $\lambda = 1.0$, where the Cross-Entropy algorithm completely ignores the MAC term, we can attain 0.3% more accuracy, at the cost of an extra 5.8% computation load. While this is still not as large as the improvements seen when used in the sMoE mode by the CIGN models trained with Resnet145 on CIFAR100, this is understandable due to the more powerful routing capability of CIGT models.

In Figure 4.4, as we did with the Fashion-MNIST experiments, we gave statistics about the router behaviors. Compared to Fashion-MNIST experiments, we can see that routers produce more smooth routing distributions that have higher entropies. Moreover, the overall distribution of the routing unit activations is clustered much closer to 0, in contrast with the Fashion-MNIST models. This more well-behaved structure of the router units can facilitate the improved success of the sMoE approach for the Resnet110 models on CIFAR-10.

4.8. Semantic Partitioning of Data

The information gain-based routing in CIGT is designed to group samples from semantically similar classes. When comparable samples with subtle semantic distinctions traverse the same root-to-leaf paths in the DAG structure, the deep network along the path can learn to distinguish between the finer details of these samples, effectively transforming the original multi-class problem into a more manageable one. In this section, we illustrate how information gain-based routing divides data into semantically meaningful sub-parts through histograms computed on the test sets of Fashion-MNIST, CIFAR-10, and MNIST datasets.

For each dataset, we display the utilized routing structure and present histograms associated with the most prevalent classes routed to that specific block. The histograms

are truncated when the cumulative probability surpasses the 85% threshold, ensuring concise representation while capturing the predominant routing patterns.

In the images of routing units, each row showcases five random samples from the corresponding class, alongside their frequencies. This visual representation offers insight into the diversity and distribution of samples within each class.

The subsequent set of figures provides the precise number of test set samples routed to the respective blocks, offering detailed quantitative analysis of the routing distribution across different classes and blocks.

4.8.1. MNIST

The CIGT routing for the MNIST dataset on a [1,2,4] architecture is depicted in Figure 4.5. In handwritten digits, there isn't a distinct visual grouping of class groups. However, the CIGT routing effectively divides the classes into groups based on certain low-level features of the handwritten shapes. For instance, classes 1 and 7 are directed into the same route in both blocks, likely due to the linear structure shared by both numbers. Similarly, classes 3 and 5 are routed together, showcasing a similar curvature structure, especially in the lower half of the images. This pattern is also observed for classes 4 and 9, where the lower part of both classes is drawn similarly. This suggests that the CIGT model is able to discern similarities in shape characteristics, aiding in effective classification despite the lack of distinct visual groupings in the dataset.

Figures 4.6 and 4.7 provide a quantitative distribution of test samples in both routing blocks. Notably, samples from class 0 are predominantly directed to Block(2,2), with significant numbers also routed to Block(2,0). In Block(2,2), we observe the presence of classes 8 and 6, while Block(2,0) includes class 9. All these classes share circular features akin to class 0, which can elucidate the inclination of class 0 samples to be routed into these specific blocks. This distribution highlights how certain shared visual characteristics influence the routing decisions within the CIGT model.

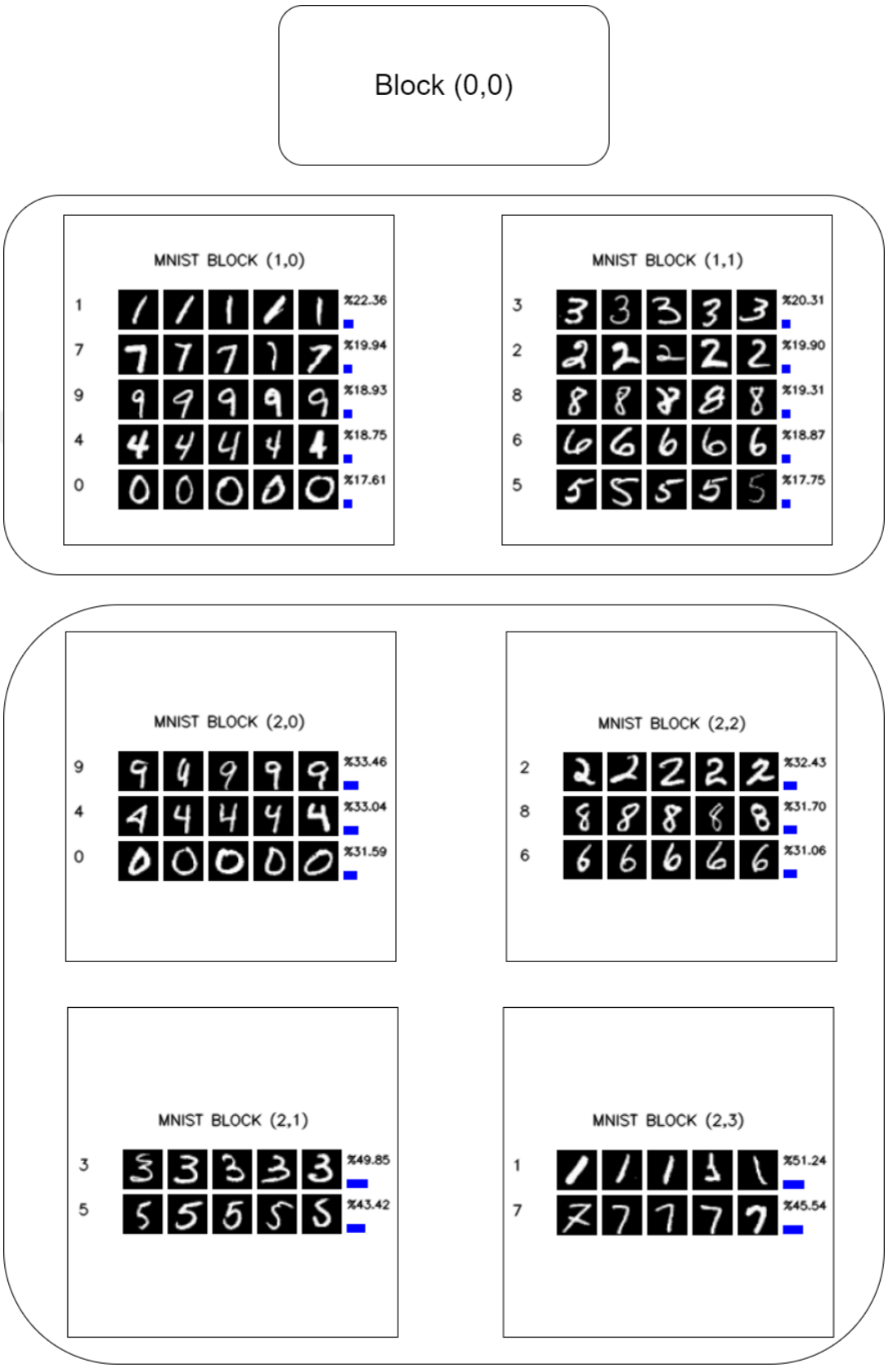


Figure 4.5. Distribution of MNIST samples into CIGT routing units.

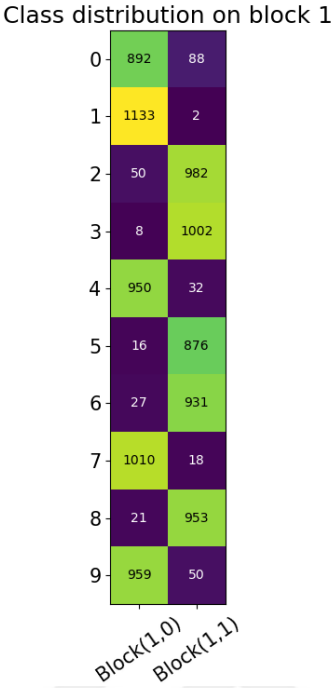


Figure 4.6. MNIST test set sample distribution in block 1.

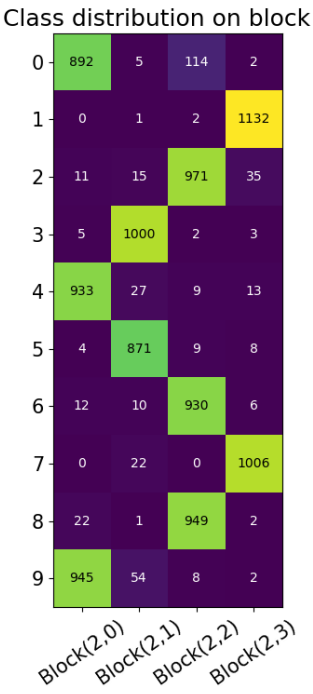


Figure 4.7. MNIST test set sample distribution in block 2.

4.8.2. Fashion-MNIST

Fashion-MNIST exhibits a distinct separation between three main groups: three classes for footwear, six classes for clothing, and the Bag class, which doesn't strictly align with the former two groups. The [1,2,4] structure of CIGT models, as illustrated in Figure 4.8, effectively groups these classes. Histograms computed on the test set depict the routing distributions, showcasing the separation of six clothing classes from the three footwear classes and the Bag class in both Block(1,0) and Block(1,1). In the subsequent routing block, Trouser, Dress, and T-Shirt classes are routed to Block(2,0). This grouping makes sense visually, as both Trouser and Dress samples often resemble a rectangle with greater height and smaller width, while T-Shirt samples share visual similarities with the upper body parts of the Dress category. Block(2,1) predominantly contains the Bag and Ankle Shoe categories. Although these categories may not share detailed visual similarities, their broader resemblance—larger width and shorter height rectangles—could lead to this specific grouping. In Fashion-MNIST experiments, a $\lambda_{balance}$ value of 2 is utilized, encouraging routers to achieve a balanced distribution of samples. This might contribute to grouping Bag with Ankle Boots instead of grouping all footwear classes while leaving the Bag class isolated. Block(2,2) is dominated by the Coat, Pullover, and Shirt classes, logically grouping upper body-related clothing samples with generally longer arm parts. Finally, Block(2,3) contains the Sandals and Sneakers classes, both being footwear categories with shorter ankle parts, providing a plausible reason for their clustering. These examples demonstrate that the CIGT structure generates groups with meaningful semantic explanations.

Figure 4.9 and 4.10 present the numerical distribution of test samples from the same experiment across various computation units. An interesting observation is the slight tendency of the Shirt class to be routed into Block(2,0) as well. This could potentially be explained by the visual distinction between shorter and longer arms in Shirt images, an inter-class variance effect.

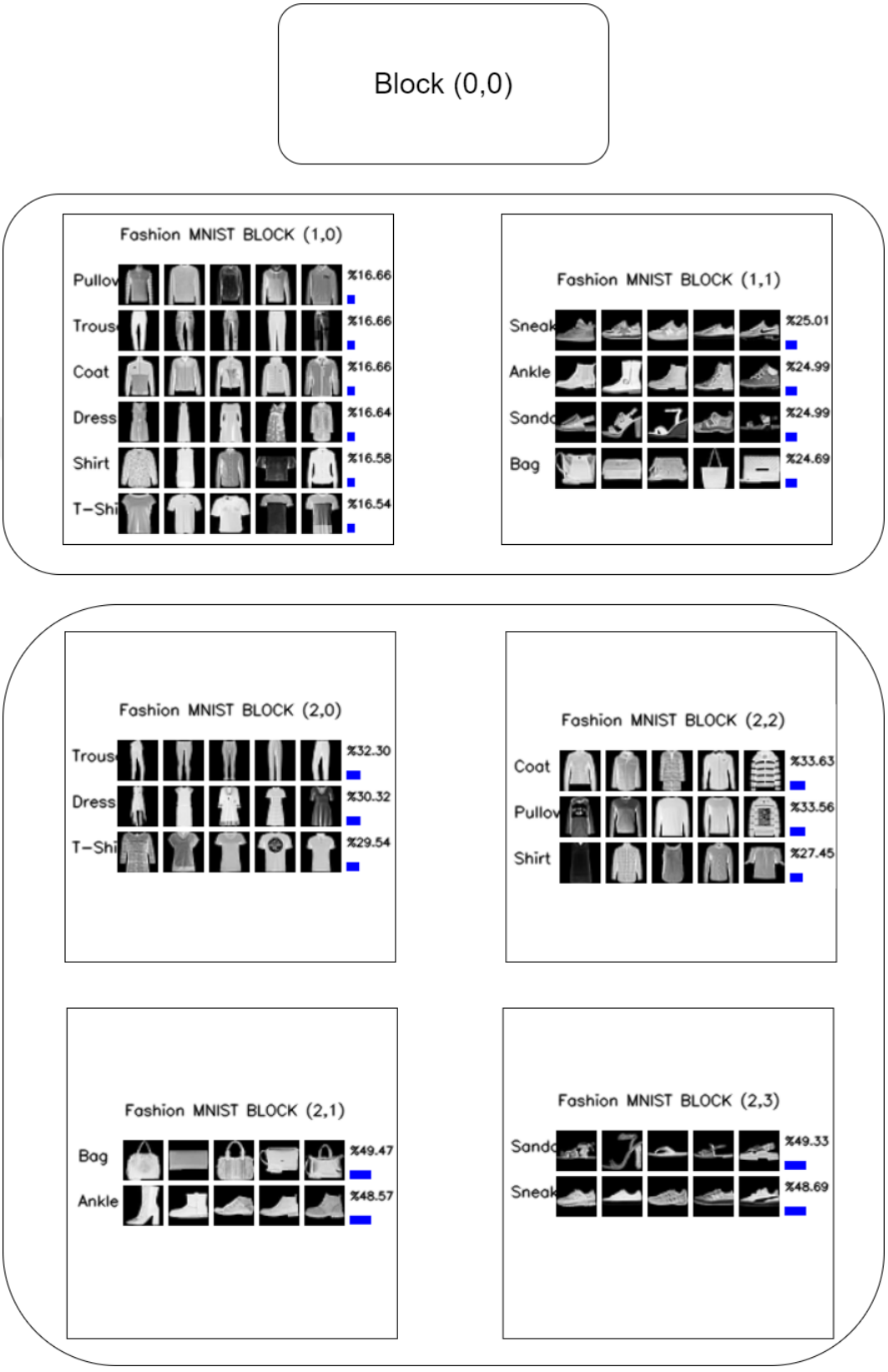


Figure 4.8. Distribution of Fashion-MNIST samples into CIGT units.

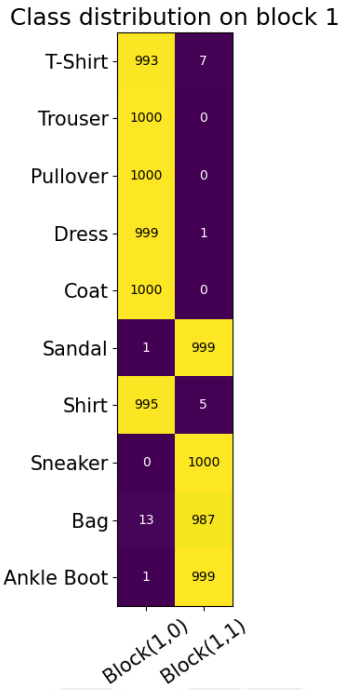


Figure 4.9. Fashion-MNIST test set sample distribution in block 1.

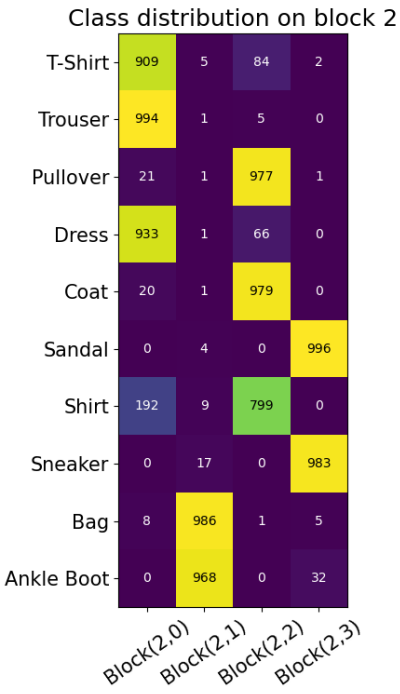


Figure 4.10. Fashion-MNIST test set sample distribution in block 2.

4.8.3. CIFAR-10

The categorization of CIFAR-10 test samples driven by information gain-based routing in the [1,2,4] CIGT model is presented in Figure 4.11.

In Block 1, a distinct separation is evident between the two groups. Block(1,0) encompasses animals, while human-made objects (automobile, truck, ship, airplane) are directed to Block(1,1). Visual features within animal classes exhibit greater similarity, particularly among four-legged animals like dogs, cats, and deer, in contrast to the diversity found in human-made objects. It's also natural that road vehicles, such as automobiles and trucks, are grouped into the same computation unit due to evident similarities, like the presence of wheels. Similarly, ships and airplanes, sharing a background of sky and sea, exhibit visual similarities, making them more akin. A noteworthy observation is that a greater number of bird samples are directed into Block(1,1) compared to other animals. One plausible explanation is that certain bird images feature the sky as the background, and images with spread wings share some visual resemblance with airplane samples. Within Block 4, there are four distinct routes. Block (2,0) predominantly comprises four-legged animals such as horses, dogs, and cats, making it easily interpretable. Block (2,1) consists mainly of the frog and deer classes. Although these classes are not inherently similar, there might be some visual cues in the background, such as a greenish-grassy appearance. In our CIFAR-10 experiments, we employed a balance coefficient, that is $\lambda_{balance} = 5$, which exerted pressure on the routers to uphold a balanced distribution of samples among the computation units. This may have influenced the router to learn a specific distribution where both deer and frog classes are directed into the same route. Semantically, it might be more meaningful to allocate deer samples to Block (2,0) alongside other four-legged animals, while designating the frog class as the sole dominant class for Block (2,1). In Block (2,2), the predominant classes are ship, airplane, and bird. We attribute the routing of bird samples to this unit to the previously mentioned similarity between birds and airplanes. Lastly, Block (2,3) predominantly contains automobile and truck classes, which can be easily interpreted semantically.



Figure 4.11. Distribution of CIFAR-10 samples into CIGT routing units.

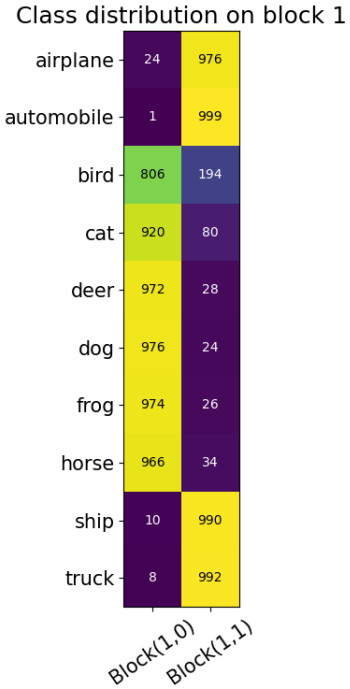


Figure 4.12. CIFAR-10 test set sample distribution in block 1.

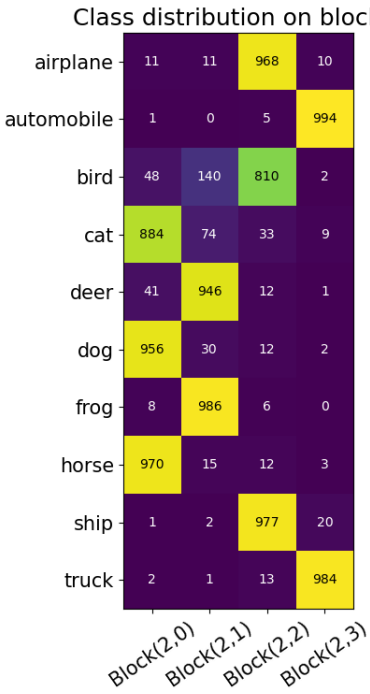


Figure 4.13. CIFAR-10 test set sample distribution in block 2.

In Figures 4.12 and 4.13, we present the distribution of test samples across various routes, displaying actual numerical values in matrix form. These numbers align with the earlier observations, confirming that classes with semantic similarities are consistently directed to the same routes.

5. REINFORCEMENT LEARNING BASED MULTI-PATH ROUTING

5.1. Motivation

In both of our CIGN and CIGT models, we have added an inference module that we call Sparse Mixture of Experts to our algorithms, which acts as a post-processing operation on a pre-trained model. The purpose of this algorithm is to run on a validation set after training the backbone conditional network and determining optimal probability threshold values for the routers, such that some samples that we deem as hard and that produce uncertain routing probabilities could be sent into multiple paths in the model. Hence, more than one root-to-leaf expert model would calculate predictions on such difficult samples and possibly recover any possible error that a misrouted expert would cause solely by its prediction. We have observed in our experiments that this post-processing approach contributes to CIGN models in terms of considerably higher accuracy, in Chapter 3. We also saw in Chapter 4 that a similar Sparse Mixture of Experts approach did not show the same improvement with our trellis-based CIGT models. We have hypothesized that since the trellis shape already has the capability of routing samples more robustly compared to the tree structure, the room for improvement using the probability thresholds for a whole dataset might be small. However, we still think there is potential in improving the performance of CIGT models with a post-processing step and enabling Sparse Mixture of Experts inference with an improved approach compared to finding global routing probability thresholds. With global, we mean that these thresholds are used for all samples, independent of the particular samples once they have been calculated. In this chapter, we are going to focus on a new Sparse Mixture of Experts method for the CIGT models that we designed to improve the performance of these models further.

While we see that the Cross-Entropy search for CIGT models gives some accuracy increase with an acceptable extra computation load, it is optimized for the whole

validation set. Intuitively, a better approach would be to have a mechanism that finds the optimal routing probability thresholds for every sample, uniquely, compared to other samples. Theoretically, we expect that this approach will work better; since given $L(x, \rho)$ is the target function for x and ρ, we have

$$\max_{\rho_1,\ldots,\rho_N} \sum_{i=1}^{N} L(x_i, \rho_i) \geq \max_{\rho} \sum_{i=1}^{N} L(x_i, \rho). \tag{5.1}$$

If we can find optimal ρ_i for every sample x_i uniquely, then we can obtain an accuracy and computation load on average over the whole data that is at least as good as the global threshold optimization approach. With this motivation, our target now is to find a method that will provide us with ρ_i as a function of the sample x_i in the CIGT models. For this purpose, we turn to reinforcement learning approaches.

5.2. Brief Reinforcement Learning Recapture

Reinforcement Learning is widely used for the solution of optimal control problems [162]. The decision-maker in the RL environment is called the agent. The agent interacts with the environment, which includes everything outside the agent. At a given time step t, the environment is in a certain state, s_t. Assuming that the agent is informed of the exact state of the environment, it takes an action a_t and possibly changes the state of the environment. The environment also provides a reward, r_{t+1} to the agent due to its action, a_t, in the environment's current state, s_t. $s_t \in S$ denotes all the possible states the agent can be in at time step t, and $a_t \in A(s_t)$ denotes the possible actions for the state s_t. In general, when the agent is in the state s_t, it takes an action according to a policy $a_t \sim \pi(a_t|s_t)$. Then the next state of the environment is sampled: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. The agent also receives a reward as the result of its action: $p(r_{t+1}|s_t, a_t)$. The system only depends on the recent state and action while sampling new values, so this type of modeling is called the Markov Decision Process (MDP). The value of a policy π, $V^{\pi}(s_t)$ is the expected cumulative reward that will be received while the agent follows the policy, starting from the state s_t. In the finite horizon model, the agent tries to maximize the expected reward for the next T

steps [135]

$$V^{\pi}(s_t) = \mathbb{E}[r_{t+1} + r_{t+2} + \cdots + r_{t+T}|s_t] = \mathbb{E}\left[\sum_{i=1}^{T} r_{t+i}|s_t\right]. \tag{5.2}$$

When we have an infinite number of steps to take, we use the infinite horizon model, where a discount rate of $0 < \gamma < 1$ is introduced to make the value function a finite amount

$$V^{\pi}(s_t) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+T} + \ldots|s_t] = \mathbb{E}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}|s_t\right]. \tag{5.3}$$

The optimal policy π^* is defined as

$$\pi^* = \arg\max_{\pi} V^{\pi}(s_t), \forall s_t. \tag{5.4}$$

Similarly, we have the Q function as $Q^{\pi}(s_t, a_t)$ where

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[r_{t+1} + r_{t+2} + \cdots + r_{t+T}|s_t, a_t] = \mathbb{E}\left[\sum_{i=1}^{T} r_{t+i}|s_t, a_t\right], \tag{5.5}$$

and

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+T} + \ldots|s_t, a_t] = \mathbb{E}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}|s_t, a_t\right]. \tag{5.6}$$

So we have

$$V^{\pi}(s_t) = \sum_{a_t} \pi(a_t|s_t) Q^{\pi}(s_t, a_t). \tag{5.7}$$

An important result in Reinforcement Learning is the Bellman equation, which relates the value or Q-function at step t with the next time step $t+1$, for a given optimal policy π^* [135] as

$$Q^{\pi^*}(s_t, a_t) = \sum_{s_{t+1}} \sum_{r_{t+1}} p(r_{t+1}|s_t, a_t) p(s_{t+1}|s_t, a_t) \left[r_{t+1} + \gamma \max_{a_{t+1}} Q^{\pi^*}(s_{t+1}, a_{t+1})\right]. \tag{5.8}$$

The fundamental problem in reinforcement learning approaches, very broadly, is either learning an approximation to the optimal Q tables or learning an approximation to the optimal policy, π^*. Learning Q tables can be done with algorithms like Value or Policy Iteration and Deep Q Learning methods, where direct learning of the optimal policy is the target of the Policy Gradients algorithm and its numerous variants [36, 135, 162–166]. We would like to point out that our aim here is not a broad review of state-of-the-art reinforcement learning techniques and their implementations in the

field of deep learning, which is clearly out of the scope of this work, but rather to apply basic ideas from this field of machine learning to our particular problem for solving the optimal multi-path routing problem for the CIGT models. We use two ideas for our CIGT multi-path learning problem. The first is a straightforward implementation of the REINFORCE algorithm for Policy Gradients in our multi-path routing framework [35]. The second strategy is a Q-Learning one in which we turn the reinforcement learning problem into a supervised one that is permissible due to the finite nature of the potential routes in our CIGT models.

5.3. Interpreting CIGT Routing a Markov Decision Process

In this section, we will start by recapturing the idea of our Sparse Mixture of Experts approach in the case of CIGT models. Then we will show we can interpret this as a Markov Decision Process, which can then lead to the application of ideas from the reinforcement learning algorithms.

After a CIGT model has been trained regularly, we take this model and freeze its weights. For every possible route in the CIGT model, we determine probability thresholds. Assuming a CIGT-[1,2,4] model, as in Figure 4.1, let us describe how the multi-path routing and Sparse Mixture of Experts inference are executed, with a concrete example:

A sample first passes through the $F_{0,0}$ unit with a path in the network as $F_{0,0}(x)$ and goes into the H_0 unit, which will generate a probability distribution on the following two computation units, $F_{1,0}$ and $F_{1,1}$ in the next layer. Let us further assume that the sample generates the routing probabilities $p_0(n_0 = 0|x) = 0.4$ and $p_0(n_0 = 1|x) = 0.6$, and we have probability thresholds that we have optimized with the use of Cross-Entropy Search, as told in Chapter 4, that are $\rho^0_{(0,0)} = 0.45$ and $\rho^1_{(0,0)} = 0.3$. The underscored $(0, 0)$ means that the samples have followed the route $F_{0,0}$ so far, and the overscore 0 and 1 means that thresholds are defining the connections to the nodes 0.th

and 1.th nodes in the next CIGT layer, respectively. Comparing these thresholds with the probabilities we calculated, it is apparent that the sample will only get routed into the node $F_{1,1}$ in the next CIGT layer. The sample will follow the route $F_{1,1} \circ F_{0,0}(x)$ and will go into the router unit H_1. This unit will now produce four probabilities for the units in the next CIGT layer. As an example, assume we obtained the probabilities $p_1(n_1 = 0|x) = 0.3$, $p_1(n_1 = 1|x) = 0.2$, $p_1(n_1 = 2|x) = 0.45$, $p_1(n_1 = 3|x) = 0.05$ and we have the thresholds $\rho^0_{(0,0),(1,1)} = 0.2$ and $\rho^1_{(0,0),(1,1)} = 0.1$, $\rho^2_{(0,0),(1,1)} = 0.15$ and $\rho^3_{(0,0),(1,1)} = 0.225$. Since the probabilities for the child nodes $0, 1, 2$ have larger values than the learned thresholds, that means the sample will now traverse into the three corresponding nodes in the next CIGT layer: $F_{2,0}$, $F_{2,1}$, $F_{2,2}$. So, in total, three root-to-leaf paths will be executed: $F_{2,0} \circ F_{1,1} \circ F_{0,0}(x)$, $F_{2,1} \circ F_{1,1} \circ F_{0,0}(x)$ and $F_{2,2} \circ F_{1,1} \circ F_{0,0}(x)$. The final prediction will be made based on these three root-to-leaf paths

$$\begin{aligned} p(y|x) =& \frac{1}{3} p(y|x, l = \{(0,0),(1,1),(2,0)\}) + \frac{1}{3} p(y|x, l = \{(0,0),(1,1),(2,1)\}) + \\ & \frac{1}{3} p(y|x, l = \{(0,0),(1,1),(2,2)\}). \end{aligned} \tag{5.9}$$

In this architecture, assuming we learn an independent probability threshold for every possible route connection, we are going to learn 10 thresholds from the Cross-Entropy algorithm:

- For the route $(0,0)$: $\rho^0_{(0,0)}, \rho^1_{(0,0)}$
- For the route $(0,0),(1,0)$: $\rho^0_{(0,0),(1,0)}, \rho^1_{(0,0),(1,0)}, \rho^2_{(0,0),(1,0)}, \rho^3_{(0,0),(1,0)}$
- For the route $(0,0),(1,1)$: $\rho^0_{(0,0),(1,1)}, \rho^1_{(0,0),(1,1)}, \rho^2_{(0,0),(1,1)}, \rho^3_{(0,0),(1,1)}$

The cross-entropy search in Algorithm 5 optimizes as a cost function

$$L_{opt}(\rho) = \lambda L_{class}(\rho) - (1 - \lambda)\mathbb{E}_{p(x)}[L_{MAC}(x, \rho)], \tag{5.10}$$

where ρ is the vector of all probability thresholds.

Based on the description of the CIGT multi-path machinery above, we observe that the sequence induced in a CIGT is equivalent to a finite horizon process in a rein-

forcement learning setting. Without loss of generality, based on our CIGT architecture given in Figure 4.1, we can define a Markov Decision Process as follows:

- We begin in the first layer of the CIGT, where we only have the node $F_{0,0}$. The original sample x goes through the transformations in this node 0, which becomes $F_{0,0}(x)$. These computation units are usually convolutional layers, but for the sake of clarity, assume they are linearized and we have our state space in the first layer defined as $s^0 = F_{0,0}(x)$, $s^0 \in \mathbb{R}^{L_1}$.
- Assume that the router in the first CIGT layer, H_0, has generated two probabilities, $p^0_{(0,0)}(x), p^1_{(0,0)}(x)$. We define our action space as $A^0 = \{0, 1\}$. The actions are defined as:
  (i) $a^0 = 0$, which means we route the sample only to the node that is assigned the largest probability by the router H_0.
  (ii) $a^0 = 1$, which means we route the sample only into the nodes that have been assigned the two largest probabilities by the router H_0. Since we already have two nodes in the next layer, this amounts to selecting both of the nodes in the second layer.

  A policy network P_0 for the CIGT layer 0 $\pi(a_0|s_0)$ is defined as a small neural network that takes in the input produced by $s^0 = F_{0,0}(x)$ and outputs a probability distribution for the action space A^0. Given a hypothetical CIGT-[1,2,4] structure where the probabilities $p^0_{(0,0)}(x) = 0.4$ and $p^1_{(0,0)}(x) = 0.6$ for a given sample, selecting the different actions leads to the following results in Figures 5.1 and 5.2. Note that the reward in the intermediate layers is defined as 0, hence $R_0(s_0, a_0) = 0$.
- Based on the first layer's actions, either one or both units in the next layer are selected. If $a^0 = 1$ is chosen from the policy network P_0 in the previous step, the sample is routed into both $F_{1,0}$ and $F_{1,1}$. Both of these networks are executed, and their results are accumulated into a single state space vector. For the accumulation, we simply take the average of the two outputs: $s^1 = \frac{1}{2}F_{1,0} \circ F_{0,0}(x) + \frac{1}{2}F_{1,1} \circ F_{0,0}(x)$. If a computation unit is not selected due to the previous layer's action, we use a zero vector of the appropriate shape instead of its output.

- Since both $F_{1,0}$ and $F_{1,1}$ have been executed, their intermediate representations are fed into the router H_1, and we obtain two sets of routing probabilities for the final layer of the CIGT:

  (i) For the route $F_{1,0} \circ F_{0,0}(x)$ we have the probabilities

  $$p^0_{(0,0),(1,0)}(x), p^1_{(0,0),(1,0)}(x), p^2_{(0,0),(1,0)}(x), p^3_{(0,0),(1,0)}(x). \tag{5.11}$$

  (ii) For the route $F_{1,1} \circ F_{0,0}(x)$ we have the probabilities

  $$p^0_{(0,0),(1,1)}(x), p^1_{(0,0),(1,1)}(x), p^2_{(0,0),(1,1)}(x), p^3_{(0,0),(1,1)}(x). \tag{5.12}$$

  The accumulated state space vector s^1 is input into a second policy network $\pi(a_1|s_1)$, denoted as P_1. In a general scenario, let F_1 represent the set of activated routes in CIGT layer 1. In this specific example, F_1 is defined as $F_1 = \{F_{1,0} \circ F_{0,0}, F_{1,1} \circ F_{0,0}\}$. Here, $F_{1,0}$ and $F_{1,1}$ denote activated routes within the first layer of the CIGT, each combined with $F_{0,0}$, which represents the activated route in the preceding layer, layer 0.

- The policy network P_1 outputs a probability distribution on a state space of four, $A^1 = \{0, 1, 2, 3\}$. In this case, the actions represent the following:

  (i) $a^1 = 0$, for every route $r \in F_1$, only the computation unit in the next CIGT layer with the largest probability routing probability is activated.

  (ii) $a^1 = 1$, for every route $r \in F_1$, only the computation units in the next CIGT layer with the two largest probability routing probabilities are activated.

  (iii) $a^1 = 2$, for every route $r \in F_1$, only the computation units in the next CIGT layer with the three largest probability routing probabilities are activated.

  (iv) $a^1 = 3$, for every route $r \in F_1$, only the computation units in the next CIGT layer with the four largest probability routing probabilities are activated. This also means that all possible routes will be executed.

  Only the child nodes of routers already executed in the CIGT can be activated. This sequential activation process enables the execution of computational nodes at subsequent levels, prioritizing those with higher routing probabilities based on the confidence of information gain-based routers. Figure 5.3 illustrates a hypothetical representation of this operational sequence with example values, demonstrating how nodes are prioritized and activated based on routing confidence in

the CIGT structure.

- The action $a1$ entails the final combination of root-to-leaf paths, where a set $r \in F_2$ represents all activated routes in the final layer. The class prediction of CIGT would be then: $\hat{y} = \arg\max_y \left(\frac{1}{|F_2|} \sum_{r \in F_2} p(y|x, l = r) \right)$. If the prediction is correct, $I[\hat{y} = y] = 1$, then we will get a reward of $\alpha > 0$. If the prediction is wrong, $I[\hat{y} = y] = 0$, then we don't get a reward, or we get a negative reward: $\beta \leq 0$. Also taking into account all the extra paths that have been calculated during the sample´s navigation in the CIGT, we also calculate the extra computation that has been done, denoted as $L_{MAC}(x, \{F_0, F_1, F_2\})$ where we calculate the total computation cost over all paths that have been executed. The final reward term is then given as

$$R_2 = \alpha^{I[\hat{y}=y]} \beta^{1-I[\hat{y}=y]} + \lambda L_{MAC}(x, \{F_0, F_1, F_2\}). \tag{5.13}$$

With that, a single episode for a single sample has been completely executed in the CIGT.

In the Algorithm 6, we give a generalized version of the Markov Decision Process for CIGT multi-path routing.

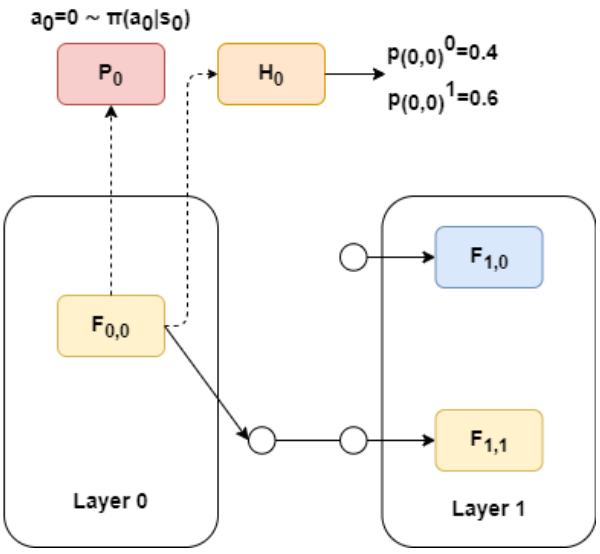


Figure 5.1. $a^0 = 0$ is selected.

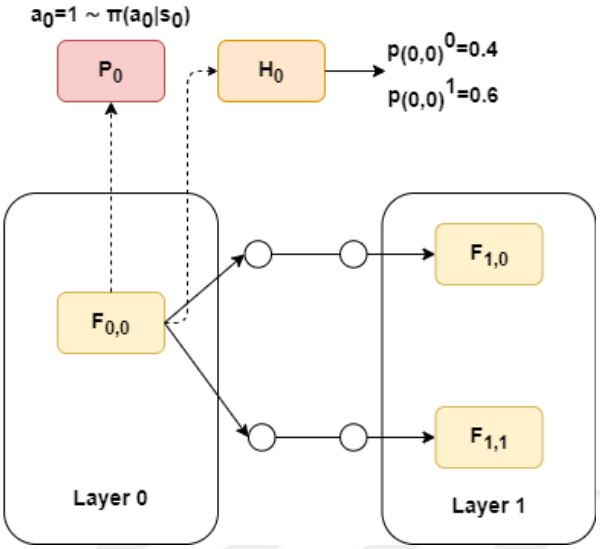


Figure 5.2. $a^0 = 1$ is selected.

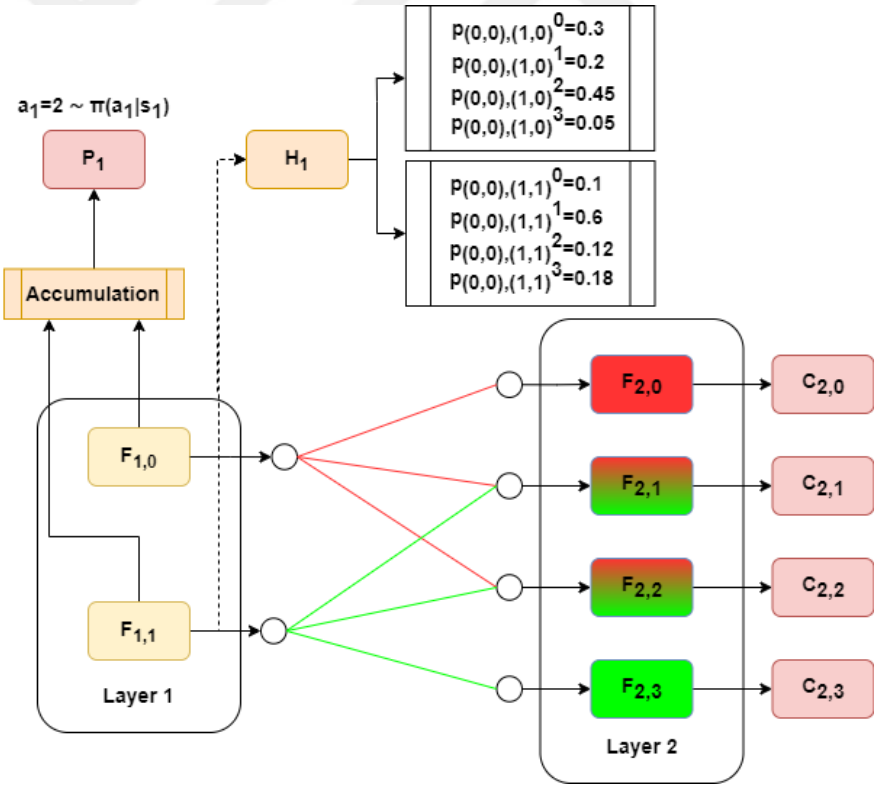


Figure 5.3. Layer 2 nodes are selected at layer 1 following the MDP action trajectory $a^0 = 1$ and $a^1 = 2$. Nodes in CIGT layer 2 are color-coded: red for nodes selected by route $(0, 0), (1, 0)$, green for nodes selected by route $(0, 0), (1, 1)$, and gradient colors for nodes selected by both routes.

Algorithm 6: The Markov Decision Process for CIGT Multi-path Routing

Sample : (x,y)

Computation Units : $F_{0,0}, \ldots F_{L-1,K_{L-1}-1}$

Routing Units : $H_0, \ldots H_{L-1}$

Policy Networks : $P_0, \ldots P_{L-1}$

MDP Hyperparameters : α, β, λ

Result: Multi-Path Routing Reward: R_L

Keep the record of executed routes in set F, initialize F with the root unit $F_{0,0}$:
$F = \{F_{0,0}\}$;

Set L to the number of non-leaf CIGT levels;

for *t in [0:L]* **do**

  $F_{next} = \{\}$;

  $S = \{\}$;

  Keep routing probabilities to the next CIGT level in A, for every current route $r \in F$: $A = []$;

  for *r in F* **do**

    Execute the route and store its result: $r(x) \rightarrow s_r$;

    $S = S \cup s_r$;

    Execute the router unit, and sort the next level nodes based on routing probabilities: $A\,[r] = sort(H_t(s_r))$;

  end

  Set the state vector, with an accumulation operation: $s_t = accum(S)$;

  Sample action a_t from the policy network: $a_t \sim P_t(s_t)$;

  for *r in F* **do**

    Select next-level nodes, with Python-style indexing:
    $F_{next} = F_{next} \cup \{n \in A[r][: (a_t + 1)]\}$;

  end

  $F_{next} \rightarrow F$;

end

Calculate the final reward: $\hat{y} = \arg\max_y \left(\frac{1}{|F|} \sum_{r \in F} p(y|x, l = r) \right)$;

$R_L = \alpha^{I[\hat{y}=y]} \beta^{1-I[\hat{y}=y]} + \lambda L_{MAC}(x, F)$;

Based on this definition of the Markov Decision Process, in the case of a CIGT network, we are going to define two algorithms that will train the policy networks P_l at every level of CIGT networks.

5.4. Policy Gradients Based Routing

Policy Gradients is a classic model-based reinforcement learning algorithm whose aim is to learn the optimal policy function $\pi_\theta(a_t|s_t)$ given a state vector [162, 163]. It should be noted that since CIGT routing is a finite process, we will always use finite-horizon definitions of the basic algorithm. The objective for optimization is to increase the probability of trajectories that would bring the highest total reward on average as

$$\arg\max_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} r_t\right] = \sum_\tau p(\tau;\theta)\left(\sum_{t=0}^{T-1} r_t\right), \tag{5.14}$$

where the state, action, and reward sequences τ are defined as

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1})\,. \tag{5.15}$$

The optimization problem seeks the optimal θ to maximize the expected reward. The probability term $p(\tau;\theta)$ typically includes state transition and action generation probabilities, with policy functions and rewards as constant functions of state and action. The expectation can then be explicitly written as

$$\mathbb{E}\left[\sum_{t=0}^{T-1} r_t\right] = \sum_\tau p(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)\prod_{t=0}^{T-1} p(s_{t+1}|s_t,a_t)\left(\sum_{t=0}^{T-1} r_t\right). \tag{5.16}$$

Optimizing this term with gradients that are calculated with respect to θ lies at the heart of the classic REINFORCE algorithm [35]. Based on the identity $\nabla_\theta\pi_\theta(x) = \pi_\theta(x)\nabla_\theta\log\pi_\theta(x)$, the REINFORCE algorithm calculates the policy gradients with respect to θ as

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_\theta\log\pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t=0}^{T-1} r(s_{i,t},a_{i,t})\right), \tag{5.17}$$

where the summation is over a batch of samples N and each sequence is of length T, however, the sequences can be of different lengths in practice.

We interpret the CIGT multi-path routing scheme as a policy gradient problem and adapt the REINFORCE algorithm to solve for the optimal routing policies for each sample in the architecture. In the case of CIGT, the action spaces are defined as in the previous section and in Algorithm 6. It should be noted that in our case, both the action space and the action space are changing at every new step of the algorithm. For example, in the CIGT-[1,2,4] model, in the first level, our action space will consist of two actions: $A^0 = \{0, 1\}$, and in the second step of the algorithm, it will become $A^1 = \{0, 1, 2, 3\}$. The state space also changes since the inputs are coming from different levels of the CIGT network, where they are going through different dimension-changing transformations, like max pooling operations. For this reason, we project the intermediate output of the CIGT layers into a shared, same-sized state space. For the action spaces, we are using policy networks P_l for every CIGT level l that convert an intermediate CIGT layer output into a discrete action.

Like in the case of cross-entry search-based probability threshold optimization, we use a pre-trained CIGT network for policy optimization. The CIGT networks will behave as feature extractors; only the attached policy networks P_l are trained by the REINFORCE algorithm. For a CIGT-[1,2,4] model of ours, we calculate a policy gradient term

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{2} \nabla_\theta \log P_\theta(a_{i,t}|CIGT(s_i, t)) \right) (R_2(s_i) - B_t)\,. \tag{5.18}$$

The $CIGT(s_i, t)$ term represents the accumulated state space vector, which is calculated from the combination of routes that a sample goes through until the level t. The accumulation is simply an average of the state space vectors for all the routes in the t. CIGT level. B is the baseline term, which we calculate as an exponential moving average of the final rewards we acquire after every iteration as

$$B_{t+1} = \epsilon * B_t + (1 - \epsilon) \left(\frac{1}{N} \sum_{i=1}^{N} R_2(s_i) \right). \tag{5.19}$$

The baseline is just a control variate term that is purposed to be positively correlated with the reward terms, which will reduce the variance of the REINFORCE policy gradients. It is a well-established way of controlling the otherwise problematic stochastic

gradient variance in the case of the REINFORCE algorithm [162, 163]. While there are more advanced ways to approach the policy gradient algorithm in the literature, like the Actor-Critic method, Proximal Policy Optimization, and Deterministic Policy Gradients [166, 167], our aim in this context is to determine routing behaviors based on individual samples and not the application of an advanced reinforcement learning algorithm, and therefore we opted for using the REINFORCE method.

5.5. Supervised Q Networks Based Routing

The CIGT structure helps us define a finite-horizon Markov Decision Process. The episodes have a length strictly equal to the depth of the CIGT architecture. While in general, the number of all possible routes in a CIGT network grows exponentially, in practice we only make use of these networks with a depth of at most three. Hence, the number of all possible routes in each level of a CIGT-[1,2,4] model is $1, 2, 8$ respectively. In such a setting, we can calculate and enumerate the reward tables exactly and feasibly. Hence, we can define a Q-Learning setting where the optimal Q-tables can be precisely defined for every time step and the whole problem can be formulated as a regression problem, in contrast with complex game-playing scenarios that need experience-replay tables for deep Q networks [36] and where the regression targets are constantly changing as the game progresses. However, in our supervised Q-learning setting, regression targets are always constant.

Again, without loss of generality, using the previous CIGT-[1,2,4] example, we start from the final, second layer and define our first regression problem. Then we will iterate into the first layer and define our second regression problem. This process can be applied to any CIGT, no matter how deep or how many degrees it has. However, for very long CIGT structures, the exponential space complexity of the generated Q-tables might be a problem.

- In the final layer, for a given sample x, assume that $\hat{y}(x, a_0, a_1)$ is the predicted label after the sample x followed the actions a_0 and a_1 and the routes that have

been entailed due to these decisions, as defined in our MDP framework for CIGTs. Then the optimal Q-table is equal to the obtained rewards

$$Q_2(CIGT_2(x,a_0),a_1) = \lambda a^{I[\hat{y}(x,a_0,a_1)=y]} b^{1-I[\hat{y}(x,a_0,a_1)=y]} + (1-\lambda) L_{MAC}((CIGT_2(x,a_0),a_1)). \tag{5.20}$$

Here, $CIGT_2(x,a_0,a_1)$ means the state space of the sample x that went through all possible routes after executing the actions a_0 and a_1 at every intermediate CIGT level and gets accumulated into the shared state space for that particular CIGT level. $CIGT_2(x,a_0)$ is defined analogously. $a^{I[\hat{y}(x,a_0,a_1)=y]} b^{1-I[\hat{y}(x,a_0,a_1)=y]}$ is the correct prediction reward as defined previously in this chapter. The computation burden penalty term, $L_{MAC}(CIGT_2(x,a_0),a_1)$, represents the extra computation cost of the sample after taking all the routes indicated by the (a_0,a_1) action sequence. What we do is estimate the optimal Q-table with a neural network P_2, which takes as input the state vector $CIGT_2(x,a_0)$ and outputs 4 heads per the CIGT-[1,2,4] architecture, where every i.th head will predict $Q_2(CIGT_2(x,a_0),a_1=i)$.

Then we have the regression problem to solve

$$L_{Q_2} = \mathbb{E}_{a_1\sim\pi_1^*(a_1|CIGT_2(x,a_0)),x\sim p(x)} \left[\left(P_2(CIGT_2(x,a_0),a_1) - Q_2(CIGT_2(x,a_0),a_1) \right)^2 \right]. \tag{5.21}$$

Note that since we know the optimal Q table, we can calculate the optimal policy and sample from it as

$$\pi_1^*(a_1=i|CIGT_2(x,a_0)) = \frac{\exp\{Q_2(CIGT_2(x,a_0),a_1=i)\}}{\sum_{j=1}^{|A^2|}\exp\{Q_2(CIGT_2(x,a_0),a_1=j)\}}. \tag{5.22}$$

- In the first layer, the optimal Q-table can be calculated by using the Bellman equation since we have the optimal Q-tables as

$$Q_1(CIGT_1(x),a_0) = \max_{a_1} Q_2(CIGT_2(x,a_0),a_1). \tag{5.23}$$

This formula does not have an explicit reward term since all intermediate rewards except the final CIGT layer in our setting are set to 0. Note that all state transitions are deterministic in our structure as well. We can extend this inductive process to each CIGT layer. After obtaining the optimal Q-table for the first layer,

the problem can be written again as a regression problem. We estimate Q_1 with a neural network P_1. We minimize the loss

$$L_{Q_1} = \mathbb{E}_{a_1 \sim \pi_1^*(a_0|CIGT_1(x)), x \sim p(x)} \left[\left(P_1(CIGT_1(x), a_0) - Q_1(CIGT_1(x), a_0) \right)^2 \right]. \tag{5.24}$$

Again, we build the optimal policy to sample actions, as

$$\pi_1^*(a_0 = i|CIGT_1(x)) = \frac{\exp\{Q_1(CIGT_1(x), a_0 = i)\}}{\sum_{j=1}^{|A^1|} \exp\{Q_1(CIGT_1(x), a_0 = j)\}}. \tag{5.25}$$

- In general, the MDP-based routing loss for a whole, generic CIGT structure can be written as

$$L_Q = \sum_{i=1}^{d} L_{Q_i}, \tag{5.26}$$

where d is the CIGT depth.

Normally, this supervised approach to Q-Table learning is only constrained when the CIGT levels are low enough and the number of nodes at every CIGT layer is not very large. However, we don´t run into this problem since our trellis-shaped networks contain at most three layers in our practice.

5.6. Experiments

We carried out experiments on the Fashion-MNIST [29] and CIFAR-10 [30] datasets, since we have experimented with the Cross-Entropy search with these datasets already. Our aim in these experiments is to see if the reinforcement learning approaches we have designed for multi-path routing help with increased accuracy as we hypothesize, and if not, to analyze their potential shortcomings.

5.6.1. Experiment Design

Our experimental approach involves utilizing the same LeNet-based and Resnet110-based models employed in our CIGT model experiments. For supervised Q-Learning

experiments, we integrate Q-Table regressors at the conclusion of each CIGT computation layer. Similarly, in policy gradient experiments, policy networks are appended to each CIGT layer. It's important to note that we strictly employ pre-trained CIGT networks in these experiments. These CIGT networks solely generate feature representations that are subsequently fed into either the policy networks or Q-Table regression networks.

During training, optimization is focused solely on the policy networks or Q networks. To augment the training set, we utilize Cut-Out augmented versions of the original training sets used for training the foundational CIGT networks. This augmentation strategy is intended to reduce the accuracy of the pre-trained networks on the training set. By potentially lowering their accuracy, we encourage these networks to explore and adopt multipath execution strategies aimed at enhancing overall accuracy. This approach is particularly beneficial as single root-to-leaf path routing based on information gain already performs well on training sets. Therefore, learning additional multipath routing strategies can further optimize model performance in varied scenarios.

5.6.2. Fashion-MNIST and CIFAR-10 Experiments

Since the experiments on the Fashion-MNIST and CIFAR-10 datasets are very similar, we will be giving them in the same section. In both the CIFAR-10 and Fashion-MNIST experiments, we employed policy networks and Q-networks with similar structures. For each dataset, we utilized the respective CIGT Resnet110 and CIGT LeNet networks, which were previously trained using our established training methodologies tailored for trellis methods. These models, structured as CIGT-[1,2,4], have been optimized to handle the specific characteristics and complexities inherent to CIFAR-10 and Fashion-MNIST datasets.

In CIFAR-10, after applying a 2×2 max pooling on the intermediate features from CIGT, we integrate three consecutive CBAM [160] layers. The outputs from

these CBAM layers are flattened using global average pooling and then projected to a dimension size of 128 through a linear layer. Subsequently, the outputs from the two inner CIGT layers are inputted into an LSTM [168] network to process the sequence while maintaining temporal coherence. The LSTM networks in our CIGT-[1,2,4] models have hidden units of size 128, and they generate 2 and 4 action heads, respectively, for each model variant.

For the LeNet-based CIGT model, we utilize a single CBAM layer following the CIGT outputs. This is followed by a 3-dimensional LSTM module to handle the sequential nature of the data effectively.

Both the Policy Gradients and Q-Networks are trained using an Adam Optimizer [169] initialized with a learning rate set to 0.0001. To ensure stable convergence over the course of training, this learning rate undergoes linear decay at each iteration. Additionally, a weight decay rate of 0.00005 is consistently applied across both datasets during the training of policy and Q-networks. This regularization technique helps mitigate overfitting, thereby enhancing the models' ability to generalize effectively across diverse scenarios presented in CIFAR-10 and Fashion-MNIST datasets.

For both datasets, the training process spans 20 epochs on the training set. This extended duration allows the models ample opportunities to iterate over the dataset, gradually refining their parameters and learning to navigate the complexities inherent in CIFAR-10 and Fashion-MNIST. By training over multiple epochs, the models can capture intricate patterns and variations within the data, improving their ability to generalize and perform well on unseen examples during evaluation.

For training these reinforcement learning networks, we used the same original dataset as for the backbone networks. However, we used the Cut-out optimizer to prevent the backbone network from producing representations that achieve high accuracy without multi-path routing.

Table 5.1. CIFAR-10 results with reinforcement learning based multi-path routing models. The extra computation sourced by both routes and policy/Q-learning models is included in the MAC measurement. All CIGT models have the $[1, 2, 4]$ architecture. s-CIGT means sMoE-CIGT.

Method	Avg Acc.	Avg # MAC	+ % MAC	Parameters
CIGT AT	**93.81**%	78.8×10^6	0.0%	644478
CIGT FT	**93.93**%	78.8×10^6	0.0%	644478
s-CIGT AT $\lambda = 1.0$	**94.11**%	83.3×10^6	5.7%	644478
s-CIGT AT. $\lambda = 0.99$	**94.09**%	81.24×10^6	3.1%	644478
s-CIGT AT. $\lambda = 0.9$	**94.05**%	79.5×10^6	0.9%	644478
Q-CIGT AT. $\lambda = 1.0$	**94.25**%	104.0×10^6	32.5%	644478
Q-CIGT AT. $\lambda = 0.999$	**94.25**%	105.5×10^6	33.9%	644478
Q-CIGT AT. $\lambda = 0.995$	**94.26**%	104.0×10^6	32.0%	644478
Q-CIGT AT. $\lambda = 0.99$	**94.27**%	91.3×10^6	15.8%	644478
Q-CIGT AT. $\lambda = 0.95$	**94.13**%	80.5×10^6	2.2%	644478
Q-CIGT AT. $\lambda = 0.9$	**94.15**%	80.2×10^6	1.8%	644478
Q-CIGT AT. $\lambda = 0.85$	**94.11**%	80.5×10^6	2.2%	644478
PG-CIGT AT. $\lambda = 1.0$	**94.05**%	109.0×10^6	38.4%	644478
PG-CIGT AT. $\lambda = 0.999$	**94.08**%	94.1×10^6	19.4%	644478
PG-CIGT AT. $\lambda = 0.995$	**94.08**%	81.1×10^6	2.9%	644478
PG-CIGT AT. $\lambda = 0.99$	**94.07**%	80.1×10^6	1.7%	644478
PG-CIGT AT. $\lambda = 0.95$	**93.84**%	79.1×10^6	0.5%	644478
PG-CIGT AT. $\lambda = 0.9$	**93.84**%	78.8×10^6	0.0%	644478
AIG-110 [4]	94.24%	153.3×10^6	94.5%	$\approx 1.74 \times 10^6$
CSGS-110 [5]	94.36%	121.5×10^6	54.2%	$\approx 1.74 \times 10^6$
SeBoW-A (SP) [102]	93.41%	$\approx 73 \times 10^6$	-7.3%	$\approx 0.7 \times 10^6$
SeBoW-A (MP) [102]	93.45%	$\approx 76 \times 10^6$	-3.5%	$\approx 1.0 \times 10^6$
SeBoW-B (SP) [102]	93.93%	$\geq 100 \times 10^6$	$\geq$ 26.9%	$\approx 1.6 \times 10^6$
SeBoW-B (MP) [102]	94.00%	$\geq 150 \times 10^6$	$\geq$ 90.4%	$\approx 2.7 \times 10^6$

Table 5.2. Fashion-MNIST results with reinforcement learning based multi-path routing models. The extra computation sourced by both routes and policy/Q-learning models is included in the MAC measurement. All CIGT models have the $[1, 2, 4]$ architecture. s-CIGT means sMoE-CIGT.

Method	Avg Acc.	Avg # MAC	+ % MAC	Parameters
CIGT AT.	**92.52**%	2.7×10^6	0.0%	294702
CIGT FT.	**92.55**%	2.7×10^6	0.0%	294702
CIGN [27]	92.37%	3.2×10^6	19.5%	368814
s-CIGN $\lambda = 1.0$ [27]	92.72%	4.4×10^6	62.9%	368814
s-CIGN $\lambda = 0.99$ [27]	92.63%	3.4×10^6	25.8%	368814
s-CIGN $\lambda = 0.9$ [27]	92.52%	3.2×10^6	18.5%	368814
s-CIGT FT. $\lambda = 1.0$	**92.65**%	3.2×10^6	18.5%	294702
s-CIGT FT. $\lambda = 0.99$	**92.63**%	2.8×10^6	3.7%	294702
s-CIGT FT. $\lambda = 0.9$	**92.53**%	2.7×10^6	0.0%	294702
Q-CIGT AT. $\lambda = 1.0$	**92.74**%	3.7×10^6	38.5%	294702
Q-CIGT AT. $\lambda = 0.999$	**92.74**%	3.7×10^6	38.2%	294702
Q-CIGT AT. $\lambda = 0.995$	**92.71**%	3.4×10^6	27.4%	294702
Q-CIGT AT. $\lambda = 0.99$	**92.71**%	3.2×10^6	19.0%	294702
PG-CIGT AT. $\lambda = 1.0$	**92.62**%	3.9×10^6	42.6%	294702
PG-CIGT AT. $\lambda = 0.999$	**92.58**%	3.8×10^6	39.6%	294702
PG-CIGT AT. $\lambda = 0.995$	**92.56**%	3.8×10^6	38.9%	294702
PG-CIGT AT. $\lambda = 0.99$	**92.51**%	3.7×10^6	35.4%	294702

The result for our CIFAR-10 experiments can be seen in Table 5.1. We can see that our Q-Learning-based multi-path learning is able to outperform the Cross-entropy-based sMoE-CIGTs with higher accuracy. It is also apparent that at higher levels of the λ balance parameter, the Q-Networks tend to execute extra paths more heavily compared to the cross-entropy search. This result can be attributed to the overfitting behavior of the neural networks, which greedily try to execute as many routes as possible to keep the accuracy high at low balance coefficients. Cross-entropy search, a black box optimization algorithm, is more sensitive to the level of balance

parameter. In the case of Q-Learning, there is a sharp drop in the amount of extra computation used, from 15.8% to 2.2% when switching the balance parameter from $\lambda = 0.99$ to $\lambda = 0.95$. At only 2.2% extra computation, the Q-learning-based CIGT model for CIFAR-10 can beat the best-performing cross-entropy-based sMoE-CIGT method by a small margin, which incurs 5.7% extra computation. In general, we can see that Q-Learning-based multi-path routing outperforms the cross-entropy search-based one in terms of accuracy and the extra computation burden. The results with the REINFORCE algorithm (denoted as PG-CIGT in Table 5.1) are, however, clearly inferior to the Q-Learning approach. The extra accuracy they bring is even inferior to the cross-entropy-based sMoE-CIGT models. This can have two reasons: While our Q-Learning approach directly learns a constant supervised target variable, the vanilla REINFORCE algorithm struggles with learning optimal paths and suffers from the common problems of reinforcement learning algorithms. For example, we have used baselines for variance reduction, but even this might not be sufficient when the gradients are working on complex error surfaces. Secondly, our Q-Learning approach readily learns from optimal Q-Tables, designed as a regression problem. The training directly minimizes the difference with the optimal policy. However, in the case of the REINFORCE algorithm, we don't have access to the optimal policy directly. Theoretically, the loss function for policy gradient algorithms directly optimizes towards the optimal policy; however, the optimization can get stuck with suboptimal policies, where a policy function can assign all the probability mass to a specific action, causing a kind of mode collapse. We have observed this in our experiments and introduced a policy entropy term to keep the training procedure from getting degraded to action selections. Despite that improvement, the REINFORCE algorithm was not able to perform as well as the supervised Q-Learning approach.

The results of the Fashion-MNIST experiments are unfortunately not as encouraging as the CIFAR-10 ones when using our reinforcement learning-based routing systems. We can see that our Q-Learning method increases the accuracy slightly compared to the cross-entropy-based approach; the maximum accuracy becomes 92.74% from 92.65%. However, this small increase in accuracy comes with a great increase

in the extra computation. The extra MAC percentage of our Q-CIGT models ranges between 19.0% to 38.5%. We don´t show results for the values where λ is smaller than 0.99 since in these cases, the accuracy degrades to a lower level than the one of the vanilla, single-path CIGT model. The REINFORCE algorithm performed even weaker than the Q-Learning approach.

The general weakness of the post-processing-based multi-path routing algorithms in the case of Fashion-MNIST and the trained LeNet-based CIGT models can be connected to the fact that the routers make very confident choices already, and the CIGT model in that case might have little to gain from multi-path routing. We have analyzed the specific behavior of these models in the previous chapter for CIGT algorithms. Even if the multi-path routing algorithm is more complex, the network has little room for improvement since it is not possible to distinguish between difficult and easy samples by looking at the entropies of the decision routers. As in the case of the Cross-entropy-based search for thresholds, we hypothesize that a regularizer that discourages the routers from being too confident and pushes them to smoother decisions might facilitate such post-processing-based multi-path routing algorithms.

5.6.3. Comparison of Cross-Entropy Search and Reinforcement Learning for CIGT Multi-Path Routing

In our experiments, we observed that, in general, the Reinforcement Learning method significantly outperforms the baseline Conditional Information Gain Trellis (CIGT) routing compared to the Cross-Entropy Search-based method. This is particularly evident with the ResNet110 architecture on the CIFAR-10 dataset. On average, for three balance levels (λ=1.0, λ=0.99, λ=0.9), the average accuracy for the Reinforcement Learning routing (Q-CIGT) method is 94.24%, whereas it is 94.08% for the Cross-Entropy-based routing (s-CIGT). The vanilla CIGT has a mean accuracy of 93.81%. The best-performing Q-CIGT at λ=0.99 achieves an accuracy of 94.27%, while s-CIGT reaches 94.11%. The performance difference between the Reinforcement Learning and Cross-Entropy Search methods at the best-performing λ levels is significant.

According to the Bootstrap t-test between both experiment results at a significance level of 0.05, we obtain a p-value of 2e-6 between the Reinforcement Learning and Cross-Entropy Search results.

For the Fashion-MNIST dataset and the LeNet-based smaller neural network, the Reinforcement Learning-based Q-CIGT network performs slightly better on average than the Cross-Entropy-Search-based method (s-CIGT). Q-CIGT achieves 92.73% accuracy across all balance levels (λ=1.0, λ=0.99, λ=0.9), whereas s-CIGT achieves 92.60%. The best-performing Q-CIGT at λ=0.99 reaches 92.74% accuracy, and the Cross-Entropy Search counterpart achieves 92.65% at λ=1.0. However, the Bootstrap t-test between the best-performing Q-CIGT and s-CIGT models gives a p-value of approximately 0.056, indicating no significant difference on average due to the high variance in Q-CIGT model results. Nonetheless, both models significantly outperform the vanilla CIGT model, according to the Bootstrap t-test with very small p-values.

Regarding computational efficiency, we observe that Reinforcement Learning routing leads to more additional computation compared to Cross-Entropy Search. For ResNet110 and the CIFAR-10 dataset, on average, across the three balance levels, Cross-Entropy Search induces an extra 3.2% computation, measured in MACs (Multiplications and Accumulations). The best-performing s-CIGT model at λ=1.0, which ignores penalties for extra computation, causes a 5.7% increase compared to the vanilla CIGT. The Reinforcement Learning approach leads to a 16.5% increase on average across all balance levels, with the best-performing model at λ=0.99 causing a 15.9% additional computational burden. For the Fashion-MNIST dataset and the LeNet-based network, the extra computation is higher. Across all balance levels, Cross-Entropy Search causes a 7.4% increase, with the best-performing model at λ=1.0 leading to an 18.5% increase. The Reinforcement Learning approach results in a 30.8% increase across all balance levels, and the best-performing model at λ=0.99 induces an additional 37% MAC count per sample.

In general, our experiments demonstrate that Reinforcement Learning can lead to higher model performance compared to Cross-Entropy Search, with the trade-off of higher additional computation. The extra computation in the case of Reinforcement Learning can be attributed to the overfitting behavior of the neural networks used as policy networks, which greedily try to execute as many routes as possible to maintain high accuracy at high balance coefficients. The Q-learning approach attempts to approximate optimal Q-tables, where at high λ levels, the penalty from extra computation becomes nearly negligible, and neural networks can easily converge to local optima that map as many inputs to the action that opens all paths for maximum accuracy. Additionally, Reinforcement Learning uses policy networks, albeit very lightweight, contributing to the extra computation. Cross-Entropy Search, on the other hand, directly optimizes a loss value incorporating a computation cost term by adjusting probability threshold values. This allows for more fine-tuned control of extra computation compared to Reinforcement Learning. Cross-Entropy Search also does not require additional neural units, as it is a black-box optimization algorithm working solely on neural network outputs to produce probability thresholds.

Table 5.3. Significance comparison and p-values of model performances for reinforcement learning (Q-CIGT), cross-entropy search (s-CIGT), and vanilla routing (CIGT) using bootstrap t-test [148, 170]. (© 2024 IEEE).

ResNet110 CIFAR-10	**s-CIGT** $\lambda = 1.0$	**CIGT**
Q-CIGT $\lambda = 0.99$	$p = 2 \times 10^{-6}$	$p = 1 \times 10^{-6}$
s-CIGT $\lambda = 1.0$	-	$p = 2 \times 10^{-5}$
LeNet Variant Fashion-MNIST	**s-CIGT** $\lambda = 1.0$	**CIGT**
Q-CIGT $\lambda = 0.99$	$p = 5.6 \times 10^{-2}$	$p = 0.0$
s-CIGT $\lambda = 1.0$	-	$p = 0.0$

Table 5.4. Average increase in MAC per sample for reinforcement learning (Q-CIGT) and cross-entropy search (s-CIGT) compared to vanilla routing (CIGT) [170]. (© 2024 IEEE).

ResNet110 CIFAR-10	**vs. CIGT**
Q-CIGT $\lambda = 0.99$	15.9%
s-CIGT $\lambda = 1.0$	5.7%
LeNet Variant Fashion-MNIST	**vs. CIGT**
Q-CIGT $\lambda = 0.99$	37.0%
s-CIGT $\lambda = 1.0$	18.5%

Regarding model performance, Reinforcement Learning is expected to provide higher accuracy since, in theory, it allows each sample to choose the optimal routing behavior, whereas Cross-Entropy Search determines global thresholds using the training set. This expectation is significantly observed with the ResNet110 and CIFAR-10 data. However, for the smaller LeNet variant model and Fashion-MNIST dataset, the performance gain of Reinforcement Learning over Cross-Entropy Search is more limited. The significance comparisons, made using the Bootstrap t-test are given in Table 5.3. The computation load difference between the best-performing multi-path methods is given in Table 5.4.

5.6.4. Effect of Multi-Path Routing on Model Interpretability

The original Conditional Information Gain Trellis (CIGT) method [28] demonstrated that routers trained with information gain objectives tend to route samples from semantically similar classes through the same paths. This improves model interpretability, making the decisions more explainable compared to unconditional network versions. Additionally, this routing strategy has been shown to enhance model performance [28].



Figure 5.4. Routing behavior for 10 CIFAR-10 samples with the least routing entropy. There is one sample per class. The routers make highly confident decisions with low entropies. Visually, each sample can be recognized as a member of the coarse class group in the leaf nodes, the decisions are interpretable.

In this thesis, we designed two multi-path routing approaches based on Cross-Entropy Search and Reinforcement Learning. Both methods are based on the idea that if a router unit struggles to confidently classify a sample's semantic group, the sample is likely to be misclassified. It may be better to route such samples through multiple paths and obtain the prediction as an ensemble of experts. Cross-Entropy Search identifies these samples based on high-entropy routing distributions that exceed determined thresholds, indicating difficult samples. The Reinforcement Learning approach indirectly applies the same principle by iteratively learning optimal Q-Tables. These Q-Tables quantify the advantages of multi-path routing, particularly benefiting samples that are challenging to classify using a single route. Through this method, the model enhances its ability to make effective routing decisions, especially for samples that exhibit complex or ambiguous features.



Figure 5.5. Routing behavior for 10 Fashion-MNIST samples with the least routing entropy. There is one sample per class. All router decisions have very low entropy. Each sample is easily distinguished as a member of the coarse class groups; routing decisions are highly interpretable.

Given the workings of the multipath routing mechanism, we can describe the relationship between model interpretability and sample routing: The interpretability of the Cross-Entropy Search or Reinforcement Learning model's decisions is inversely proportional to the number of paths a sample is routed through. Intuitively, if a sample is difficult for the routers to classify, it is routed through multiple paths, leading to multiple expert predictions and a more complex decision process, thus reducing model interpretability. This relationship is illustrated in Figures 5.4, 5.5, 5.6, and 5.7.

In Figures 5.4 and 5.5, we see samples from each CIFAR-10 and Fashion-MNIST class with the least total routing entropy for every router unit. For CIFAR-10, data is grouped into "artificial, man-made objects" and animals at the first level (units $F_{1,0}$ and $F_{1,1}$).

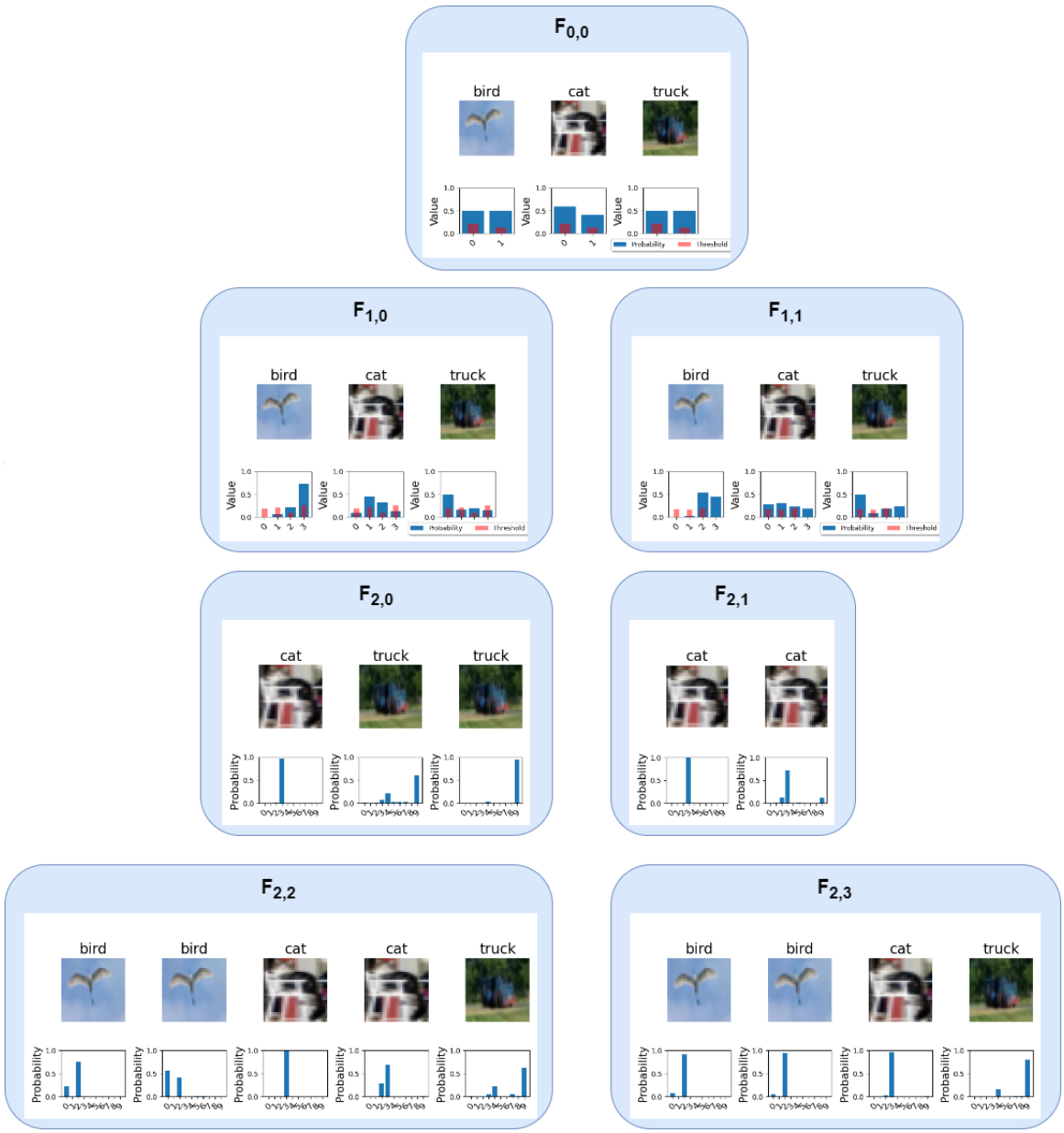


Figure 5.6. Routing behavior for three random CIFAR-10 samples with high routing entropy. Samples are routed into multiple paths. Model behavior is not easily explainable.

At the final level (units $F_{2,0}$, $F_{2,1}$, $F_{2,2}$, and $F_{2,3}$), the groupings include land devices ("automobile" and "truck"), two groups of animals ("cat", "dog", "horse", four-legged animals, and the rest), and air and sea devices ("airplane", "ship"), indicated by bluish backgrounds (Figure 5.4).

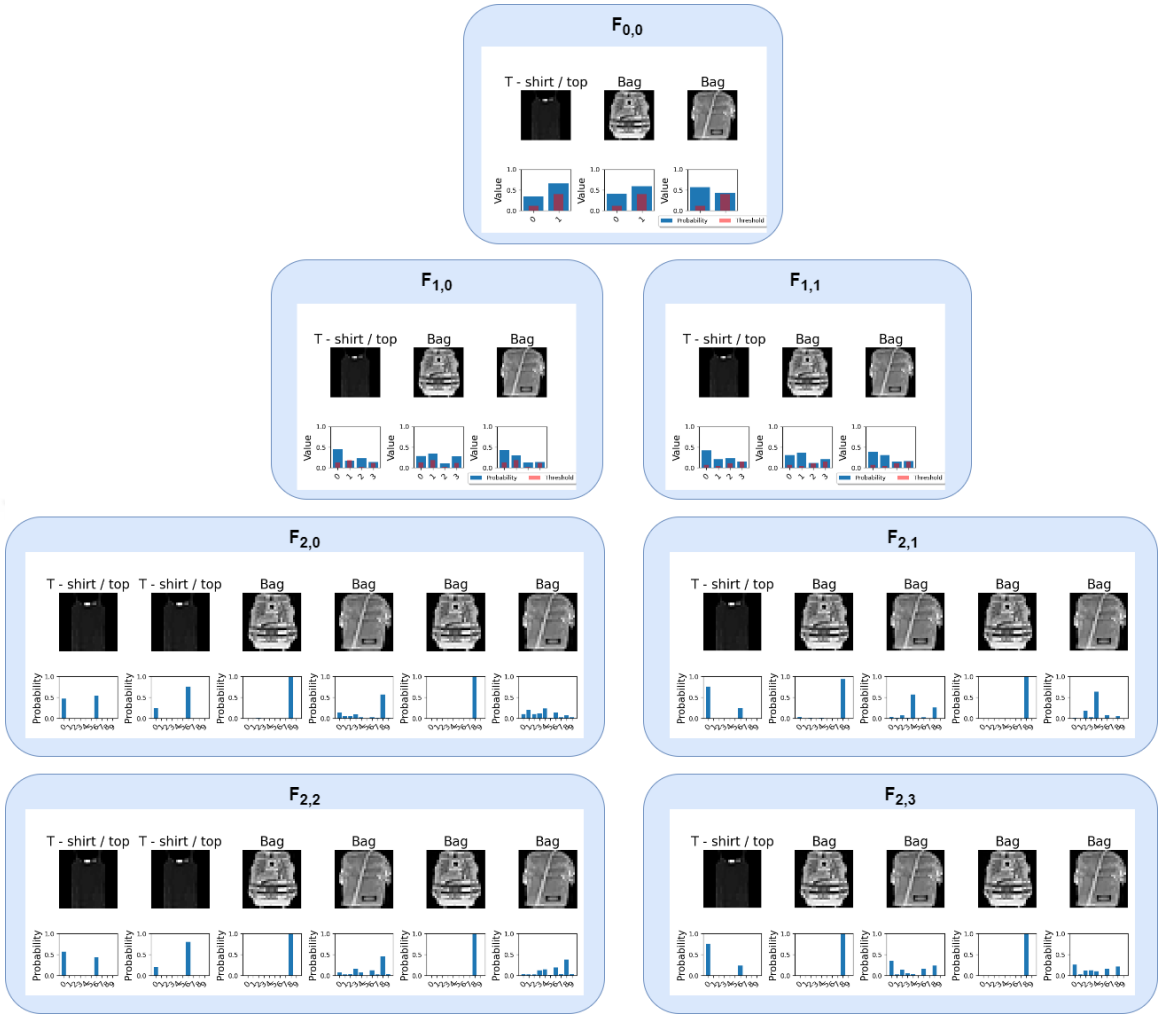


Figure 5.7. Routing behavior for three random Fashion-MNIST samples with high routing entropy. Each sample is routed into multiple paths. Routing decisions are not easily explainable.

For Fashion-MNIST, the first level groups include shoes and bags, and upper dresses and trousers. The second level groups include sandals and bags (with string or stripe-like handles), upper body garments with long sleeves ("Pullover", "Coat", "Shirt"), vertically long garments ("Trouser" and "Dress"), short-sleeved garments ("T-Shirt" and "Dress"), and closed footwear ("Sneaker" and "Ankle Boot") (Figure 5.5).

The routing decisions in Figures 5.4 and 5.5 have a very low entropy, with each sample following a single path in the trellis structure, making model behaviors easily

interpretable by the semantic affinities of the input samples. This affinity is visible in the 10 samples per dataset in Figures 5.4 and 5.5.

Figures 5.6 and 5.7 show three samples per CIFAR-10 and Fashion-MNIST datasets, randomly selected from 30 samples with the highest total routing entropy for each routing unit. In Figure 5.6, the "bird" image exemplifies a middle ground between easily and hardly interpretable routing decisions, looking similar to both a bird and an airplane. Birds and airplanes are grouped in units $F_{2,2}$ and $F_{2,3}$, and the image follows four routes from the root unit $F_{0,0}$ to these leaf units. These decisions remain interpretable as the model considers the possibility of the image belonging to either a bird or an airplane.

On the other hand, the "cat" and "truck" images in Figure 5.6 are less interpretable, with the "cat" image following six paths and visiting all leaf nodes. The absence of paths $F_{0,0} \rightarrow F_{1,0} \rightarrow F_{2,0}$ and $F_{0,0} \rightarrow F_{1,0} \rightarrow F_{2,3}$ suggests the sample is likely not a man-made device, but this is a weaker interpretation compared to a single path selection.

In Figure 5.7, the bag images are routed through all eight possible paths in the trellis structure. The second bag image (third one in $F_{0,0}$) visually resembles a coat due to arm-like structures on both sides, but this interpretation is only evident in the class probabilities of the leaf node outputs. With all possible paths executed by the routers, the interpretability of the CIGT model degrades to the level of an unconditional deep neural network model. These visual examples illustrate the inverse relationship between the number of paths taken and interpretability in the multi-path CIGT algorithms.

6. CONCLUSION

In this thesis work, we have analyzed the need for conditional computing in the domain of deep neural networks. Such models are usually very heavyweight, and it is not possible to deploy them to edge environments easily. Conditional computing is one approach to addressing this problem, where parts of a deep neural network are executed based on the properties of a given sample. To address this problem, we have designed two different algorithms. The first one is called Conditional Information Gain Networks. These are tree-structured feed-forward neural networks that lead a sample to a suitable sub-tree in the tree structure. The unique router structures we have created for conditional computing are what determine this suitable sub-tree. We chose to use information gain losses, similar to the heuristic-based information gain maximization techniques in conventional decision tree algorithms. These losses are usually not differentiable; we have converted them into differentiable neural network operations that are placed into the routing nodes in the CIGN tree structures. We have also designed a regularizer for load balancing during the training of the tree structure that keeps the tree from focusing on single root-to-leaf paths. We have done this by using the mathematical properties of the information gain term we have designed. We have additionally observed that samples that are difficult to classify get routed to experts who are not focused on their coarse class group, and such samples are highly likely to get misclassified. To address that, we have introduced the Sparse Mixtures of Experts mode of inference. After a CIGN is trained, usually with single-path execution, a search is applied for optimal routing probability thresholds. This search runs on a validation set, and it seeks to balance the extra accuracy with the extra computation burden that comes from executing multiple paths. We encoded this into a special loss function and solved this optimization problem with the machinery of Bayesian optimization since the loss function is not differentiable. In our experiments, we have shown that the CIGN algorithms are performing favorably compared to their thicker, unconditional baselines and compared to ad-hoc routing approaches like random routing. We also compared our algorithm with numerous similar conditional computation works and showed that

the CIGNs are either performing better or at least comparably with such methods.

We have observed that the tree structure found in CIGNs is susceptible to misrouting. Whenever a sample is misrouted to a subnetwork that is not optimized for its coarse class group, there is no chance to recover that particular sample. Even though we have designed countermeasures to remedy this problem in our CIGN work, a more elegant design would be to come up with a trellis-structured architecture that would allow the routing of samples between two computation nodes in more than a single way. That might help recover samples that follow erroneous paths in the network, even if they once make a routing mistake. For this reason, we have developed the Conditional Information Gain Trellis networks, which are based on a trellis shape where computation nodes have more than a single route between them. Similar to the CIGN networks, we have used information gain-based routers in this structure. The hierarchical mixture of experts loss induced by the trellis structure prevents easy back-propagation from the expert loss functions and router functions into the parameters. Identifying that, we have proposed two solutions to overcome this. The first one is a training technique we call the Approximate Training. In this training mode, only the signals that travel from the root-to-leaf experts´ classification heads and the signals that travel from the information gain-based router losses train the weights of the network. The second approach is called the Full Training. In this method, we relax the one-hot route selections with Gumbel-Softmax-based vectors and, hence, using reparametrization, change the expectation of the HMoE loss into one with parameterless Gumbel variables. While this approach helps with generating approximate gradients from the router component of the HMoE loss that we previously ignored, we run into the problem of redefining our information gain routers since they were originally designed for working with one-hot path selection vectors. By using the mathematical properties of the Gumbel-Softmax distribution, we modify the information gain losses to make them compatible with the Full Training. Like in the case of CIGNs, we identify that CIGT models can benefit from a Sparse Mixture of Experts type of multi-path routing-based inference, too. We design a Cross-Entropy search-based optimization framework for finding optimal probability thresholds for the CIGT routers, that allow an increase in accuracy while keeping

the computation burden small. For CIGT algorithms, in our experiments, we show that the vanilla CIGT method can outperform the vanilla CIGN algorithm based on its more robust routing structure. However, we have also seen during these experiments that the Sparse Mixture of Experts based routing, which works by finding optimal probability thresholds on a validation set, does not help CIGT models as much as CIGN models. While that could be anticipated since CIGT models have a more sophisticated routing structure that is less affected by misrouting, we wanted to further investigate a more sophisticated routing method. Analyzing that determining thresholds per sample, instead of determining global routing probability thresholds, theoretically will result in higher classification performance, we have turned to reinforcement learning approaches. For these methods, we turn our focus to our CIGT-based models, which need a more powerful multi-path routing mechanism in comparison to CIGN models. For this, we first interpret routing in the CIGT structure as a Markov Decision Process. Based on that, we developed two algorithms. The first one is based on the popular policy gradient algorithm REINFORCE, and the second is our design: Observing that in practical use cases, the CIGT models contain a tractable number of routes in contrast to an exponentially high number, we define optimal Q-Tables exactly in our CIGT routing framework and propagate them into the whole CIGT structure with the Bellmann Equation. After this point, we convert the multi-path routing into a supervised regression problem, which is understandably easier to solve than policy gradient-based approaches. Our tests with these new routing methods show that they work better than our global threshold-based multi-path routing methods. This is especially true for the CIFAR-10 dataset and the Resnet 110 backbone. At the same time, while we have again slightly better results in the Fashion-MNIST models, the improvement in that case with RL-based algorithms comes with a much higher computation burden than the cross-entropy-based routing. Analyzing the routing behaviors in this case, particularly the case of Fashion-MNIST and the LeNet-based model we have trained for it, we observed that the routers in this situation make very certain decisions, hence the distinction between hard and easy samples gets blurred. We hypothesize that this might be the cause of the inferior performance of multi-path routing techniques with the FashionMIST CIGT models and propose an entropy regularizer as a future work,

that we think can be helpful. With that, we are concluding this thesis.

REFERENCES

1. Bengio, Y., "Deep Learning of Representations: Looking Forward", A.-H. Dediu, C. Martín-Vide, R. Mitkov and B. Truthe (Editors), *Statistical Language and Speech Processing*, pp. 1–37, Springer Berlin Heidelberg, Berlin, Germany, 2013.

2. Murdock, C., Z. Li, H. Zhou and T. Duerig, "Blockout: Dynamic Model Selection for Hierarchical Deep Networks", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2583–2591, Las Vegas, Nevada, USA, 2016.

3. Wu, Z., T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman and R. Feris, "BlockDrop: Dynamic Inference Paths in Residual Networks", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8817–8826, Salt Lake City, Utah, USA, 2018.

4. Veit, A. and S. Belongie, "Convolutional Networks with Adaptive Inference Graphs", *International Journal of Computer Vision*, Vol. 128, No. 3, pp. 730–741, March 2020.

5. Herrmann, C., R. S. Bowen and R. Zabih, "Channel Selection Using Gumbel Softmax", A. Vedaldi, H. Bischof, T. Brox and J.-M. Frahm (Editors), *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 241–257, Springer International Publishing, Online due COVID, 2020.

6. Ioannou, Y., D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown and A. Criminisi, "Decision Forests, Convolutional Networks and the Models In-between", *arXiv preprint arXiv:1603.01250*, 2016.

7. Liu, L. and J. Deng, "Dynamic Deep Neural Networks: Optimizing Accuracy-Efficiency Trade-offs by Selective Execution", *Proceedings of the 32nd AAAI*

Conference on Artificial Intelligence, AAAI, pp. 3675–3682, AAAI press, New Orleans, Louisiana, USA, 2018.

8. Yan, Z., H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di and Y. Yu, "HD-CNN: Hierarchical Deep Convolutional Neural Networks for Large Scale Visual Recognition", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2740–2748, Santiago, Chile, 2015.

9. Teerapittayanon, S., B. McDanel and H. T. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks", *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, Cancun, Mexico, 2016.

10. Wójcik, B., A. Devoto, K. Pustelnik, P. Minervini and S. Scardapane, "Adaptive Computation Modules: Granular Conditional Computation for Efficient Inference", *arXiv preprint arXiv:2312.10193*, 2023.

11. Jordan, M. I. and R. A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm", *Neural Computation*, Vol. 6, No. 2, pp. 181–214, March 1994.

12. Shazeer, N., A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–19, Toulon, France, 2017.

13. Baykal, C., N. Dikkala, R. Panigrahy, C. Rashtchian and X. Wang, "A Theoretical View on Sparsely Activated Networks", S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh (Editors), *Advances in Neural Information Processing Systems*, Vol. 35, pp. 30071–30084, New Orleans, Louisiana, USA, 2022.

14. Eigen, D., M. Ranzato and I. Sutskever, "Learning Factored Representations in a Deep Mixture of Experts", *Proceedings of the International Conference on Learn-*

ing Representations (ICLR), Workshop Track, pp. 1–8, Banff, Canada, 2014.

15. Kontschieder, P., M. Fiterau, A. Criminisi and S. R. Bulo, "Deep Neural Decision Forests", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1467–1475, Santiago, Chile, 2015.

16. Baek, S., K. I. Kim and T. Kim, "Deep Convolutional Decision Jungle for Image Classification", *arXiv preprint arXiv:1706.02003*, 2017.

17. Ahmed, K., M. H. Baig and L. Torresani, "Network of Experts for Large-Scale Image Categorization", B. Leibe, J. Matas, N. Sebe and M. Welling (Editors), *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 516–532, Springer International Publishing, Amsterdam, The Netherlands, 2016.

18. Wan, A., L. Dunlap, D. Ho, J. Yin, S. Lee, H. Jin, S. Petryk, S. A. Bargal and J. E. Gonzalez, "NBDT: Neural-Backed Decision Trees", *Proceedings of the International Conference on Learning Representations (ICLR)*, pp. 1–12, Online due COVID, 2021.

19. Luo, H., F. Cheng, H. Yu and Y. Yi, "SDTR: Soft Decision Tree Regressor for Tabular Data", *IEEE Access*, Vol. 9, No. 1, pp. 55999–56011, 2021.

20. He, Y., J. Lin, Z. Liu, H. Wang, L.-J. Li and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices", V. Ferrari, M. Hebert, C. Sminchisescu and Y. Weiss (Editors), *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 815–832, Munich, Germany, 2018.

21. Molchanov, P., S. Tyree, T. Karras, T. Aila and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Inference", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–17, Toulon, France, 2017.

22. Molchanov, D., A. Ashukha and D. Vetrov, "Variational Dropout Sparsifies Deep

Neural Networks", *Proceedings of the 34th International Conference on Machine Learning, ICML*, pp. 2498–2507, Sydney, NSW, Australia, August 2017.

23. Sirui Xie, C. L., Hehui Zheng and L. Lin, "SNAS: Stochastic Neural Architecture Search", *Proceedings of the 7th International Conference on Learning Representations, ICLR*, pp. 1–17, New Orleans, Louisiana, USA, 2019.

24. Zhong Z., L. C., Yan J., "Practical Block-wise Neural Network Architecture Generation", *arXiv preprint arXiv:1708.05552*, 2018.

25. Cheng, A.-C., C. H. Lin, D.-C. Juan, W. Wei and M. Sun, "InstaNAS: Instance-aware Neural Architecture Search", *arXiv preprint arXiv:1811.10201*, 2018.

26. Bicici, U. C., C. Keskin and L. Akarun, "Conditional Information Gain Networks", *Proceedings of the 24th International Conference on Pattern Recognition (ICPR)*, pp. 1390–1395, Beijing, China, 2018.

27. Bicici, U. C. and L. Akarun, "Conditional Information Gain Networks as Sparse Mixture of Experts", *Pattern Recognition*, Vol. 120, No. 1, p. 108151, 2021.

28. Bicici, U. C., T. H. S. Meral and L. Akarun, "Conditional Information Gain Trellis", *arXiv preprint arXiv:2402.08345*, 2024.

29. Xiao, H., K. Rasul and R. Vollgraf, "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms", *Dataset, arXiv:1708.07747*, 2017.

30. Krizhevsky, A. and G. Hinton, "Learning Multiple Layers of Features from Tiny Images", *Dataset, Department of Computer Science, University of Toronto*, 2009.

31. Montillo, A., J. Tu, J. Shotton, J. Winn, J. E. Iglesias, D. N. Metaxas and A. Criminisi, *Entanglement and Differentiable Information Gain Maximization, In Book Decision Forests for Computer Vision and Medical Image Analysis*, pp. 273–293, Springer London, London, 2013.

32. Jang, E., S. Gu and B. Poole, "Categorical Reparameterization with Gumbel-Softmax", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–13, Toulon, France, 2017.

33. Maddison, C. J., A. Mnih and Y. W. Teh, "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–20, Toulon, France, 2017.

34. De Boer, P.-T., D. P. Kroese, S. Mannor and R. Y. Rubinstein, "A Tutorial on the Cross-Entropy Method", *Annals of Operations Research*, Vol. 134, No. 1, pp. 19–67, 2005.

35. Williams, R. J., "Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning", *Machine Learning*, Vol. 8, No. 3-4, pp. 229–256, 1992.

36. Mnih, V., 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, pp. 529–533, 2015.

37. Krizhevsky, A., I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *ACM Communications*, Vol. 60, No. 6, pp. 84–90, 2017.

38. Simonyan, K. and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *Proceedings of the 3rd International Conference on Learning Representations, ICLR*, pp. 1–14, San Diego, California, USA, 2015.

39. He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition (CVPR), pp. 770–778, Las Vegas, Nevada, USA, 2016.

40. Huang, G., Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, Honolulu, Hawaii, USA, 2017.

41. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, Boston, Massachusetts, USA, 2015.

42. Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, Las Vegas, Nevada, USA, 2016.

43. Szegedy, C., S. Ioffe, V. Vanhoucke and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 4278–4284, San Francisco, California, USA, 2017.

44. Zoph, B. and Q. V. Le, "Neural Architecture Search with Reinforcement Learning", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–16, Toulon, France, 2017.

45. Bengio, Y., N. Leonard and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation", *arXiv preprint arXiv:1308.3432*, 2013.

46. Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting.", *Journal*

of Machine Learning Research, Vol. 15, No. 1, pp. 1929–1958, 2014.

47. Li, F., N. Neverova, C. Wolf and G. Taylor, "Modout: Learning to Fuse Modalities via Stochastic Regularization", *Journal of Computational Vision and Imaging Systems*, Vol. 2, No. 1, pp. 1–3, 2016.

48. Bengio, E., P. Bacon, J. Pineau and D. Precup, "Conditional Computation in Neural Networks for Faster Models", *Proceedings of the 4th International Conference on Learning Representations, ICLR, Workshop Track*, pp. 1–12, San Juan, Puerto Rico, 2016.

49. McGill, M. and P. Perona, "Deciding How to Decide: Dynamic Routing in Artificial Neural Networks", D. Precup and Y. W. Teh (Editors), *Proceedings of the 34th International Conference on Machine Learning, ICML*, Vol. 70, pp. 2363–2372, Sydney, NSW, Australia, August 2017.

50. Wang, X., F. Yu, Z.-Y. Dou, T. Darrell and J. E. Gonzalez, "SkipNet: Learning Dynamic Routing in Convolutional Networks", *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 409–424, Munich, Germany, September 2018.

51. Krishna, T., A. K. Rai, Y. A. Djilali, A. F. Smeaton, K. McGuinness and N. E. O'Connor, "Dynamic Channel Selection in Self-Supervised Learning", *arXiv preprint arXiv:2207.12065*, 2022.

52. Yu, J., L. Yang, N. Xu, J. Yang and T. Huang, "Slimmable Neural Networks", *arXiv preprint arXiv:1812.08928*, 2018.

53. Li, C., G. Wang, B. Wang, X. Liang, Z. Li and X. Chang, "DS-Net++: Dynamic Weight Slicing for Efficient Inference in CNNs and Vision Transformers", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 45, No. 4, pp. 4430–4446, 2022.

54. Yu, J. and T. S. Huang, "Universally Slimmable Networks and Improved Training Techniques", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1803–1811, Seoul, South Korea, 2019.

55. Ioffe, S. and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *Proceedings of the 32th International Conference on Machine Learning, ICML*, pp. 448–456, Lille, France, 2015.

56. Li, C., G. Wang, B. Wang, X. Liang, Z. Li and X. Chang, "Dynamic Slimmable Network", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8607–8617, Nashville, Tennessee, USA, 2021.

57. Chin, T.-W., A. S. Morcos and D. Marculescu, "Joslim: Joint Widths and Weights Optimization for Slimmable Neural Networks", *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD*, pp. 119–134, Bilbao, Spain, 2021.

58. Jiang, Z., C. Li, X. Chang, L. Chen, J. Zhu and Y. Yang, "Dynamic Slimmable Denoising Network", *IEEE Transactions on Image Processing*, Vol. 32, No. 1, pp. 1583–1598, 2023.

59. Yang, F., K. Wang and J. van de Weijer, "ScrollNet: Dynamic Weight Importance for Continual Learning", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 3345–3355, Paris, France, 2023.

60. Xiong, C., X. Zhao, D. Tang, K. Jayashree, S. Yan and T. Kim, "Conditional Convolutional Neural Network for Modality-Aware Face Recognition", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 3667–3675, Santiago, Chile, 2015.

61. Fernando, C., D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel and D. Wierstra, "PathNet: Evolution Channels Gradient Descent in Super Neu-

ral Networks", *arXiv preprint arXiv:1701.08734*, 2017.

62. Denoyer, L. and P. Gallinari, "Deep Sequential Neural Network", *arXiv preprint arXiv:1410.0510*, 2014.

63. Han, Y., Y. Pu, Z. Lai, C. Wang, S. Song, J. Cao, W. Huang, C. Deng and G. Huang, "Learning to Weight Samples for Dynamic Early-Exiting Networks", *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 362–378, Tel Aviv, Israel, 2022.

64. Pacheco, R. G. and R. S. Couto, "Inference Time Optimization Using Branchynet Partitioning", *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, Rennes, France, 2020.

65. Chiang, C.-H., P. Liu, D.-W. Wang, D.-Y. Hong and J.-J. Wu, "Optimal Branch Location for Cost-effective Inference on Branchynet", *IEEE International Conference on Big Data (Big Data)*, pp. 5071–5080, Orlando, Florida, USA, 2021.

66. Ayyat, M., T. Nadeem and B. Krawczyk, "ClassyNet: Class-Aware Early Exit Neural Networks for Edge Devices", *IEEE Internet of Things Journal*, Vol. 11, No. 1, pp. 15113–15127, 2023.

67. Jeon, J.-Y., X. T. Nguyen, S. Ryu and H.-J. Lee, "USDN: A Unified Sample-Wise Dynamic Network With Mixed-Precision and Early-Exit", *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 646–654, Waikoloa, Hawaii, USA, 2024.

68. Yang, L., Y. Han, X. Chen, S. Song, J. Dai and G. Huang, "Resolution Adaptive Networks for Efficient Inference", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2369–2378, Seattle, Washington, USA, 2020.

69. Bishop, C. M. and M. Svensén, "Bayesian Hierarchical Mixtures of Experts",

arXiv preprint arXiv:1212.2447, 2012.

70. Hinton, G., O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network", *arXiv preprint arXiv:1503.02531*, 2015.

71. Lee, S., S. Purushwalkam, M. Cogswell, D. J. Crandall and D. Batra, "Why M Heads Are Better than One: Training a Diverse Ensemble of Deep Networks", *arXiv preprint arXiv:1511.06314*, 2015.

72. Chazan, S. E., J. Goldberger and S. Gannot, "Speech Enhancement Using a Deep Mixture of Experts", *arXiv preprint arXiv:1703.09302*, 2017.

73. Furlanello, T., Z. Lipton, M. Tschannen, L. Itti and A. Anandkumar, "Born Again Neural Networks", J. Dy and A. Krause (Editors), *Proceedings of the 35th International Conference on Machine Learning, ICML*, Vol. 80, pp. 1607–1616, Stockholm, Sweden, July 2018.

74. Kirsch, L., J. Kunze and D. Barber, "Modular Networks: Learning to Decompose Neural Computation", S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, Vol. 31, pp. 2408–2418, Montréal, Canada, 2018.

75. Dikkala, N., N. Ghosh, R. Meka, R. Panigrahy, N. Vyas and X. Wang, "On the Benefits of Learning to Route in Mixture-of-Experts Models", *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 9376–9396, Singapore, 2023.

76. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need", I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, Vol. 30, pp. 6000–6010, Long Beach, California, USA, 2017.

77. Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models Are Few-shot Learners", H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin (Editors), *Advances in Neural Information Processing Systems*, pp. 1877–1901, Vancouver, Canada, 2020.

78. Fedus, W., B. Zoph and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity", *The Journal of Machine Learning Research*, Vol. 23, No. 1, pp. 5232–5270, 2022.

79. Foresti, G. L. and G. Pieroni, "Exploiting Neural Trees in Range Image Understanding", *Pattern Recognition Letters*, Vol. 19, No. 9, pp. 869–878, 1998.

80. Sakar, A. and R. J. Mammone, "Growing and Pruning Neural Tree Networks", *IEEE Transactions on Computers*, Vol. 42, No. 3, pp. 291–299, 1993.

81. Utgoff, P. E., "Perceptron Trees: A Case Study in Hybrid Concept Representations", *Connection Science*, Vol. 1, No. 4, pp. 377–391, 1989.

82. Stromberg, J.-E., J. Zrida and A. Isaksson, "Neural Trees-Using Neural Nets in a Tree Classifier Structure", *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, pp. 137–140, Toronto, Canada, 1991.

83. Bulo, S. and P. Kontschieder, "Neural Decision Forests for Semantic Image Labelling", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 81–88, Columbus, Ohio, USA, 2014.

84. Quinlan, J. R., "Induction of Decision Trees", *Machine Learning*, Vol. 1, No. 1, pp. 81–106, March 1986.

85. Murthy, V. N., V. Singh, T. Chen, R. Manmatha and D. Comaniciu, "Deep Decision Network for Multi-class Image Classification", *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2240–2248, Las Vegas, Nevada, USA, 2016.

86. Shen, W., Y. Guo, Y. Wang, K. Zhao, B. Wang and A. L. Yuille, "Deep Regression Forests for Age Estimation", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2304–2313, Salt Lake City, Utah, USA, 2018.

87. Roy, A. and S. Todorovic, "Monocular Depth Estimation Using Neural Regression Forest", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5506–5514, Las Vegas, Nevada, USA, 2016.

88. Shen, W., K. Zhao, Y. Guo and A. L. Yuille, "Label distribution learning forests", I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, Vol. 30, pp. 834–843, Long Beach, California, USA, 2017.

89. Hazimeh, H., N. Ponomareva, P. Mol, Z. Tan and R. Mazumder, "The Tree Ensemble Layer: Differentiability Meets Conditional Computation", *Proceedings of the 37th International Conference on Machine Learning, ICML*, pp. 4138–4148, Online due COVID, July 2020.

90. Song, J., H. Zhang, X. Wang, M. Xue, Y. Chen, L. Sun, D. Tao and M. Song, "Tree-like Decision Distillation", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13488–13497, Nashville, Tennessee, USA, 2021.

91. Frosst, N. and G. Hinton, "Distilling a Neural Network into a Soft Decision Tree", *arXiv preprint arXiv:1711.09784*, 2017.

92. Yoo, J. and L. Sael, "Edit: Interpreting Ensemble Models via Compact Soft Decision Trees", *IEEE International Conference on Data Mining (ICDM)*, pp.

1438–1443, Beijing, China, 2019.

93. Ouyang, X. and C. Feng, "Interpretable Neural Network Construction: From Neural Network to Interpretable Neural Tree", *Journal of Physics: Conference Series*, Vol. 1550, No. 3, pp. 132–154, May 2020.

94. Nauta, M., R. Van Bree and C. Seifert, "Neural Prototype Trees for Interpretable Fine-grained Image Recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14933–14943, Nashville, Tennessee, USA, 2021.

95. Ji, R., L. Wen, L. Zhang, D. Du, Y. Wu, C. Zhao, X. Liu and F. Huang, "Attention Convolutional Binary Neural Tree for Fine-grained Visual Categorization", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10468–10477, Online due COVID, 2020.

96. Fu, H. and H. Wang, "Self-attention Binary Neural Tree for Video Summarization", *Pattern Recognition Letters*, Vol. 143, No. 1, pp. 19–26, 2021.

97. Wu, M., S. Parbhoo, M. Hughes, R. Kindle, L. Celi, M. Zazzi, V. Roth and F. Doshi-Velez, "Regional Tree Regularization for Interpretability in Deep Neural Networks", *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI"*, Vol. 34, pp. 6413–6421, New York, New York, USA, 2020.

98. Vidal, T. and M. Schiffer, "Born-again Tree Ensembles", *Proceedings of the 37th International Conference on Machine Learning, ICML*, pp. 9743–9753, Online due COVID, 2020.

99. Biau, G., E. Scornet and J. Welbl, "Neural Random Forests", *Sankhya A: The Indian Journal of Statistics*, Vol. 81, No. 2, pp. 347–386, December 2019.

100. Shotton, J., S. Nowozin, T. Sharp, J. Winn, P. Kohli and A. Criminisi, "Decision Jungles: Compact and Rich Models for Classification", *Proceedings of the 26th*

International Conference on Neural Information Processing Systems - Volume 1, pp. 234–242, Lake Tahoe, Nevada, USA, 2013.

101. Saxena, S. and J. Verbeek, "Convolutional Neural Fabrics", *Advances in Neural Information Processing Systems*, pp. 4053–4061, Barcelona, Spain, 2016.

102. Chen, Y., F. Mao, J. Song, X. Wang, H. Wang and M. Song, "Self-born Wiring for Neural Trees", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 5047–5056, Montréal, Canada, 2021.

103. Aytekin, C., "Neural Networks Are Decision Trees", *arXiv preprint arXiv:2210.05189*, 2022.

104. Wang, S., C. Aggarwal and H. Liu, "Random-Forest-Inspired Neural Networks", *ACM Transactions on Intelligent Systems and Technology*, Vol. 9, No. 6, pp. 1–25, October 2018.

105. Wang, S., C. Aggarwal and H. Liu, "Using a Random Forest to Inspire a Neural Network and Improving on It", *Proceedings of the SIAM International Conference on Data Mining*, pp. 1–9, Pittsburgh, Pennsylvania, USA, 2017.

106. Yang, Y., I. G. Morillo and T. M. Hospedales, "Deep Neural Decision Trees", *arXiv preprint arXiv:1806.06988*, 2018.

107. Zhou, Z.-H. and J. Feng, "Deep Forest: Towards an Alternative to Deep Neural Networks", *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 3553–3559, Melbourne, Australia, 2017.

108. Tanno, R., K. Arulkumaran, D. Alexander, A. Criminisi and A. Nori, "Adaptive Neural Trees", K. Chaudhuri and R. Salakhutdinov (Editors), *Proceedings of the 36th International Conference on Machine Learning, ICML*, Vol. 97, pp. 6166–6175, Long Beach, California, USA, 2019.

109. Neill, J. O., "An Overview of Neural Network Compression", *arXiv preprint arXiv:2006.03669*, 2020.

110. Mallya, A., D. Davis and S. Lazebnik, "Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights", *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82, Munich, Germany, 2018.

111. Wen, W., C. Wu, Y. Wang, Y. Chen and H. Li, "Learning Structured Sparsity in Deep Neural Networks", *Advances in Neural Information Processing Systems*, pp. 2082–2090, Barcelona, Spain, 2016.

112. Lebedev, V. and V. Lempitsky, "Fast ConvNets Using Group-Wise Brain Damage", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2554–2564, Las Vegas, Nevada, USA, 2016.

113. Lin, S., R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang and D. Doermann, "Towards Optimal Structured CNN Pruning via Generative Adversarial Learning", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2785–2794, Long Beach, California, USA, 2019.

114. Louizos, C., K. Ullrich and M. Welling, "Bayesian Compression for Deep Learning", *Advances in Neural Information Processing Systems*, pp. 3290–3300, Long Beach, California, USA, 2017.

115. Dai, B., C. Zhu, B. Guo and D. Wipf, "Compressing Neural Networks Using the Variational Information Bottleneck", J. Dy and A. Krause (Editors), *Proceedings of the 35th International Conference on Machine Learning, ICML*, Vol. 80, pp. 1135–1144, Stockholm, Sweden, July 2018.

116. Tishby, N. and N. Zaslavsky, "Deep Learning and the Information Bottleneck Principle", *Proceedings of the IEEE Information Theory Workshop (ITW)*, pp. 1–5, Jerusalem, Israel, 2015.

117. Tang, Z., L. Luo, B. Xie, Y. Zhu, R. Zhao, L. Bi and C. Lu, "Automatic Sparse Connectivity Learning for Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 34, No. 10, pp. 7350–7364, 2023.

118. Frankle, J. and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks", *Proceedings of the 7th International Conference on Learning Representations, ICLR*, pp. 1–42, New Orleans, Louisiana, USA, 2019.

119. Noy, A., N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes and L. Zelnik, "ASAP: Architecture Search, Anneal and Prune", *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, Vol. 108, pp. 493–503, Online due COVID, August 2020.

120. Zoph, B., V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, Salt Lake City, Utah, USA, 2018.

121. Baker B., N. N., Gupta O. and R. R., "Designing Neural Network Architectures Using Reinforcement Learning", *Proceedings of the 5th International Conference on Learning Representations, ICLR*, pp. 1–18, Toulon, France, 2017.

122. Cai, H., T. Chen, W. Zhang, Y. Yu and J. Wang, "Efficient Architecture Search by Network Transformation", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, pp. 2787–2794, New Orleans, Louisiana, USA, 2018.

123. Elsken, T., J.-H. Metzen and F. Hutter, "Simple and Efficient Architecture Search for Convolutional Neural Networks", *arXiv preprint arXiv:1711.04528*, 2017.

124. Miikkulainen, R., J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju,

H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving Deep Neural Networks", *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, Vol. 1, No. 1, pp. 293–312, 2019.

125. Real, E., S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le and A. Kurakin, "Large-scale Evolution of Image Classifiers", D. Precup and Y. W. Teh (Editors), *Proceedings of the 34th International Conference on Machine Learning, ICML*, Vol. 70, pp. 2902–2911, Sydney, NSW, Australia, August 2017.

126. Xie, L. and A. Yuille, "Genetic CNN", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397, Venice, Italy, 2017.

127. Liu, C., B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang and K. Murphy, "Progressive Neural Architecture Search", *arXiv preprint arXiv:1712.00559*, 2017.

128. Liu, H., K. Simonyan and Y. Yang, "Darts: Differentiable Architecture Search", *arXiv preprint arXiv:1806.09055*, 2018.

129. Liu, H., K. Simonyan and Y. Yang, "Graph HyperNetworks for Neural Architecture Search", *arXiv preprint arXiv:1810.05749*, 2018.

130. Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

131. Breiman, L., "Random Forests", *Machine Learning*, Vol. 45, No. 1, pp. 5–32, 2001.

132. Robnik-Šikonja, M., "Improving Random Forests", *European Conference on Machine Learning*, pp. 359–370, Pisa, Italy, 2004.

133. Segal, M. and Y. Xiao, "Multivariate Random Forests", *Wiley Interdisciplinary*

Reviews: Data Mining and Knowledge Discovery, Vol. 1, No. 1, pp. 80–87, 2011.

134. Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T.-Y. Liu, "Lightgbm: A Highly Efficient Gradient Boosting Decision Tree", I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Editors), *Advances in Neural Information Processing Systems*, Vol. 30, pp. 3149–3157, Long Beach, California, USA, 2017.

135. Alpaydin, E., *Machine Learning*, Mit Press, Cambridge, 2021.

136. Criminisi, A., J. Shotton, E. Konukoglu *et al.*, "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-supervised Learning", *Foundations and Trends® in Computer Graphics and Vision*, Vol. 7, No. 2–3, pp. 81–227, 2012.

137. Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets", *Advances in Neural Information Processing Systems*, Vol. 27, pp. 2672–2680, Montréal, Canada, 2014.

138. Goodfellow, I., "Nips 2016 Tutorial: Generative Adversarial Networks", *arXiv preprint arXiv:1701.00160*, 2016.

139. Creswell, A., T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, "Generative Adversarial Networks: An Overview", *IEEE Signal Processing Magazine*, Vol. 35, No. 1, pp. 53–65, 2018.

140. Sze, V., Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey", *Proceedings of the IEEE*, Vol. 105, No. 12, pp. 2295–2329, 2017.

141. Jones, D. R., M. Schonlau and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions", *Journal of Global Optimization*, Vol. 13, No. 4, pp. 455–492, Dec 1998.

142. Williams, C. and C. Rasmussen, "Gaussian Processes for Regression", *Advances in Neural Information Processing Systems*, Vol. 8, pp. 514–520, Denver, Colorado, USA, 1995.

143. Williams, C. K. and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, Vol. 2, MIT Press, Cambridge, 2006.

144. Lecun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

145. Hull, J. J., "A database for handwritten text recognition research", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 5, pp. 550–554, 1994.

146. Breiman, L., "Bagging Predictors", *Machine Learning*, Vol. 24, No. 2, pp. 123–140, 1996.

147. Breiman, L., "Pasting Small Votes for Classification in Large Databases and Online", *Machine Learning*, Vol. 36, No. 1, pp. 85–103, 1999.

148. Efron, B. and R. J. Tibshirani, *An Introduction to the Bootstrap*, Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993.

149. Confalonieri, R., L. Coba, B. Wagner and T. R. Besold, "A Historical Perspective of Explainable Artificial Intelligence", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 11, No. 1, pp. 1–21, 2021.

150. Schulman, J., N. Heess, T. Weber and P. Abbeel, "Gradient Estimation Using Stochastic Computation Graphs", *Advances in Neural Information Processing Systems*, Vol. 28, pp. 3528–3536, Montréal, Canada, 2015.

151. Mnih, A. and K. Gregor, "Neural Variational Inference and Learning in Belief

Networks", *Proceedings of the 31th International Conference on Machine Learning, ICML*, pp. 1791–1799, Beijing, China, 2014.

152. Gregor, K., I. Danihelka, A. Mnih, C. Blundell and D. Wierstra, "Deep Autoregressive Networks", *Proceedings of the 31th International Conference on Machine Learning, ICML*, pp. 1242–1250, Beijing, China, 2014.

153. Gu, S., S. Levine, I. Sutskever and A. Mnih, "Muprop: Unbiased Backpropagation for Stochastic Neural Networks", *arXiv preprint arXiv:1511.05176*, 2015.

154. Mnih, A. and D. Rezende, "Variational Inference for Monte Carlo Objectives", *Proceedings of the 33th International Conference on Machine Learning, ICML*, pp. 2188–2196, New York, New York, USA, 2016.

155. Casella, G. and R. L. Berger, *Statistical Inference*, Vol. 2, Duxbury Thomson Learning, Pacific Grove, California, USA, 2001.

156. Kingma, D. P. and M. Welling, "Auto-encoding Variational Bayes", *arXiv preprint arXiv:1312.6114*, 2013.

157. Gumbel, E. J., *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, Vol. 33, US Government Printing Office, Washington DC, USA, 1954.

158. Schröder, C. and S. Rahmann, "A Hybrid Parameter Estimation Algorithm for Beta Mixtures and Applications to Methylation State Classification", *Algorithms for Molecular Biology*, Vol. 12, No. 1, pp. 1–12, 2017.

159. DeVries, T. and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout", *arXiv preprint arXiv:1708.04552*, 2017.

160. Woo, S., J. Park, J.-Y. Lee and I. S. Kweon, "Cbam: Convolutional Block Attention Module", *Proceedings of the European Conference on Computer Vision*

(ECCV), pp. 3–19, Munich, Germany, 2018.

161. Cubuk, E. D., B. Zoph, J. Shlens and Q. V. Le, "Randaugment: Practical Automated Data Augmentation with a Reduced Search Space", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 702–703, Online due COVID, 2020.

162. Sutton, R. S. and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, 2018.

163. Sutton, R. S., D. McAllester, S. Singh and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", *Advances in Neural Information Processing Systems*, Vol. 12, pp. 1057–1063, Denver, Colorado, USA, 1999.

164. Fan, J., Z. Wang, Y. Xie and Z. Yang, "A Theoretical Analysis of Deep Q-learning", *Learning for Dynamics and Control*, pp. 486–489, Berkeley, California, USA, 2020.

165. Moerland, T. M., J. Broekens, A. Plaat, C. M. Jonker *et al.*, "Model-based Reinforcement Learning: A Survey", *Foundations and Trends® in Machine Learning*, Vol. 16, No. 1, pp. 1–118, 2023.

166. Lapan, M., *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero, and More*, Packt Publishing Ltd, Birmingham, United Kingdom, 2018.

167. Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, "Deterministic Policy Gradient Algorithms", *Proceedings of the 31th International Conference on Machine Learning, ICML*, pp. 387–395, Beijing, China, 2014.

168. Hochreiter, S. and J. Schmidhuber, "Long Short-term Memory", *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

169. Zhang, Z., "Improved Adam Optimizer for Deep Neural Networks", *IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–2, Banff, Canada, 2018.

170. Bicici, U. C. and L. Akarun, "Multi-Path Routing for Conditional Information Gain Trellis Using Cross-Entropy Search and Reinforcement Learning", *IEEE Access*, Vol. 12, No. 1, pp. 66319–66334, 2024.