Data-dependent Resource Allocation and Routing in Multi-path Neural Networks for Vision-based Learning

M H G Dumindu Tissera

(188013F)

Thesis submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy

Department Electronic and Telecommunication Engineering

University of Moratuwa
Sri Lanka

September 2023

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: UOM Verified Signature Date: 2023/09/10

The above candidate has carried out research for the PhD thesis under my supervision.

Signature of the Supervisor(s): Date: 10 September 2023

UOM Verified Signature

Dr. Ranga Rodrigo

Abstract

As the depth of a neural network increases, the non-linearity and more parameters allow it to learn more complex functions. While network deepening has been proven effective, there is still an opportunity for efficient feature extraction within a layer that will improve the overall performance for the complexity of the network. Widening networks by adding more filters to each layer is the naive approach towards strengthening layer-wise feature extraction. It is an inefficient scaling option, considering the number of parameters being quadratic with the number of filters employed per layer. In contrast, parallel extractors in each layer provide an efficient scaling option. However, without context-dependent input allocation among these processes, such parallel computations tend to learn similar features, collapsing to a single computation.

Thus, it is vital to study the parallel stacking of computations layer-wise and design a routing method that allocates incoming feature maps to these computations. The expected outcome is to group homogeneous feature maps in parallel layers and employ exclusive filter sets to each of the groups (paths) so that the filter sets of each path can specialize in extracting features exclusive to each group.

To allow the network input to be routed end-to-end over such parallel paths, we propose data-dependent parallel resource allocation methods layer-wise. Given a layer of parallel tensors, we first employ sub-networks that produce gating coefficients to weigh cross-connections to the next layer of parallel tensors. Then, the next layer's parallel tensors are constructed by getting summations of the current layer's tensors, each weighted by the corresponding gating coefficient. We demonstrate that our multi-path networks outperform previous widening and adaptive feature extraction, ensembles, and deeper networks with comparable complexity using image recognition challenges.

To further regularize gating sub-networks, we think of a gating network's path allocation as a soft clustering of its input feature maps. Thus, we propose a neural mixture model-based clustering objective to use as a regularization loss for the gating networks, which We first study as a standalone neural network-based clustering approach. The proposed clustering framework uses a neural network to learn cluster distributions in mixture modeling instead of tuning human-defined distributions. We adopt the Expectation-Maximization (EM) algorithm to train the network and perform batch-wise EM iterations where the forward pass acts as the E-step and the backward pass as the M-step. For image clustering, we use the mixture-based EM objective as the clustering objective, along with consistency optimization. Our networks outperform traditional and single-stage deep clustering methods that still depend on k-means.

Finally, we propose using this clustering objective to regulate gating networks to get distributed gating activation patterns. We show that the skewed gating patterns can be improved with such regularization loss as a local regularization. We further present the need for a global regularization method that takes the end task performance into account. We also suggest extending research towards sparse resource allocation, along with gating networks to handle more diversity.

***Keywords*- Multi-Path Networks, Data-Dependent Routing, Deep Clustering, Neural Mixture Models**

DEDICATION

To the resilient people of Sri Lanka,

who have faced the challenges of the country's financial crisis with unwavering determination and faith,

who continue to endure and strive for a better future for the nation.

This dissertation is a tribute to your strength and commitment to a brighter tomorrow.

ACKNOWLEDGMENTS

I would like to express my deep appreciation and gratitude to the following individuals for their invaluable contributions to the completion of my Ph.D. thesis.

First and foremost, I would like to thank my adviser, Dr. Ranga Rodrigo, for his unwavering support, guidance, and encouragement throughout my Ph.D. studies. His expertise, insight, and mentorship have been invaluable to my academic and personal growth.

I also extend my gratitude to Dr. Subha Fernando for her technical support, which has contributed significantly to the success of this research. Additionally, I am grateful to Prof. Sanath Jayasena and Dr. Jayathu Samarawickrama for their constructive suggestions and insightful comments, which have helped refine and improve this thesis's technical content.

I would like to acknowledge the generous support of CodeGen International (PVT) Ltd and its CEO Dr. Harsha Subashinghe, who provided funding for this research. I am grateful for their support, which has allowed me to pursue my academic goals.

I would also like to thank Prof. Dileeka Dias, Prof. Sanath Jayasena, Dr. Ranga Rodrigo, Dr. Subha Fernando, Dr. Jayathu Samarawickrama, Dr. Harsha Subhasinsghe, and CodeGen International (PVT) Ltd for their role in the establishment of QBITS Lab at the University of Moratuwa and the Ph.D. program.

I extend my sincere appreciation to Prof. Ruwan Udayanga and Dr. Prathapasinghe Dharmawansha, the Research Coordinators, for coordinating the program and providing feedback on technical content. I am also indebted to Dr. Charith Chiththraranjan, the chair of the progress review panel, for reviewing and providing valuable feedback on my work.

I thank my colleagues and collaborators, Rukshan Wijesinghe, Kasun Vithanage, Alex Xavier, and Pubudu Ekanayake, for their outstanding technical contributions and unwavering support throughout this research.

Finally, I thank the thesis examiners Prof Salman Khan, Dr. Sadeep Jayasumana and Dr. Shehan Perera, and PhD examination chair Prof. Gihan Dias for providing valuable suggestions.

Thank you for your invaluable contributions, guidance, and support in making this achievement possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Towards Improving Layer-wise Feature Extraction

The neural network performance in learning a particular task tends to increase with its depth [1–5] as deeper neural networks are able to approximate rather complex patterns. The ability to approximate such complex patterns arises from the high number of layers and non-linear activations which separate them. While depth has a validated impact on neural network performance, it is also intuitive to search for possible improvements in feature extraction within neural layers. However, there has been comparatively less exploration of efficient layer-wise resource usage. A carefully engineered resource allocation within each neural layer may improve the overall utility of the network., i.e., the performance with respect to the total number of parameters.

One could increase the number of filters in a convolutional layer or the number of nodes in a dense layer to improve the layer-wise feature extraction [6]. However, such a design results in inefficient scaling as the number of parameters is quadratic in terms of the number of filters per layer (or number of nodes per layer). It is more efficient to have parallel paths or operations in a given layer [5,7] instead, which results in an efficient scaling option where the number of parameters is linear in terms of the number of stacks per layer. Model ensembling [4,8] feeds the same image to several neural networks and accumulates those network responses to produce the final prediction. In addition, there have been efforts to obtain different versions of the same image by different image processing methods and feed these versions to multiple independent networks [9, 10] to produce the output. While all these works use parallel computations, they still lack an efficient method to allocate the input among these parallel computations in a context-dependent manner. Therefore, these parallel computations tend to learn redundant information, which is an inefficient use of parallel resources.

1.2 Usage of Parallel Stacks of Feature Maps (Paths)

A conventional neural layer contains a single set of feature maps on which a single family of learnable filters operates to produce the next layer feature map set. In contrast, let's consider a multi-path neural network that has several parallel computations in each layer. In a layer of a multi-path network, there are several parallel independent sets of feature maps (paths). Each set of feature maps (tensor or path) has a dedicated family of learnable filters to operate on them to produce the corresponding output set of feature maps, continuing that particular path. Let's assume that homogeneous feature maps which share similar contextual details are already grouped to parallel paths in a given layer. Now, each family of filters operating on its set of feature maps (path) can specialize to extract features in the corresponding context. Therefore, we can expect efficient use of parallel resources in a given layer which is dependent on the nature of the incoming sets of feature maps.

Overall, we can expect efficient use of parallel paths of the multi-path network which is dependent on the context of the input image. Such efficient usage of layer-wise parallel resources can result in improved network performance with respect to its capacity (number of parameters or complexity). A layer of multiple paths and families of dedicated filters for each path's context may perform a richer combination of features in total than a single larger set of feature maps (conventional widening), a layer with parallel paths but no such grouping of feature maps (existing parallel computations), and even a sequential stack of layers in a single path (deeper networks). Such effective use of parallel computations does not present in existing parallel architectures therefore those parallel computations tend to learn similar features.

1.3 Need for Layer-wise Routing Layers

How do we group homogeneous feature maps to parallel paths and allocate incoming parallel tensors to those paths? To do that, we need an algorithm that routes information between subsequent layers of parallel paths (tensors) in a data-dependent manner. The algorithm should first facilitate cross-connections between parallel tensors in adjacent layers. These connections must be further weighted in a data-dependent way to allocate the incoming tensors in an adaptive manner. The weighting of cross-connections in a particular layer would preferably be based on the context of the incoming parallel tensors. With such a careful design, the parallel paths would be able to allocate resources to incoming tensors efficiently.

It is possible to allocate inputs to the parallel paths at the first layer and then concatenate the outputs of those parallel paths at the end of the network. In such scenarios, there are no intermediate connections between parallel paths, which can route the feature maps before the final layer. Rather than using this approach, it is important to have intermediate routing along the whole depth of a multi-path network, preferably for each layer segment, because the image *context* is captured throughout the depth of the neural network, where each successive layer represents more abstract concepts than its predecessor layer. Hence, the groups of homogeneous feature maps can be dissimilar from each other. In our opinion, the context of an image is a cumulative detail that is not limited to the corresponding class, and the real image context that governs the given task may be different from the human interpretation [11]. In the lowest levels of a neural network, overall color and structure of edges, etc. may be captured as the context. On deeper levels, the contextual representations become much more abstract and can be constituted of information like body pose or even the class.

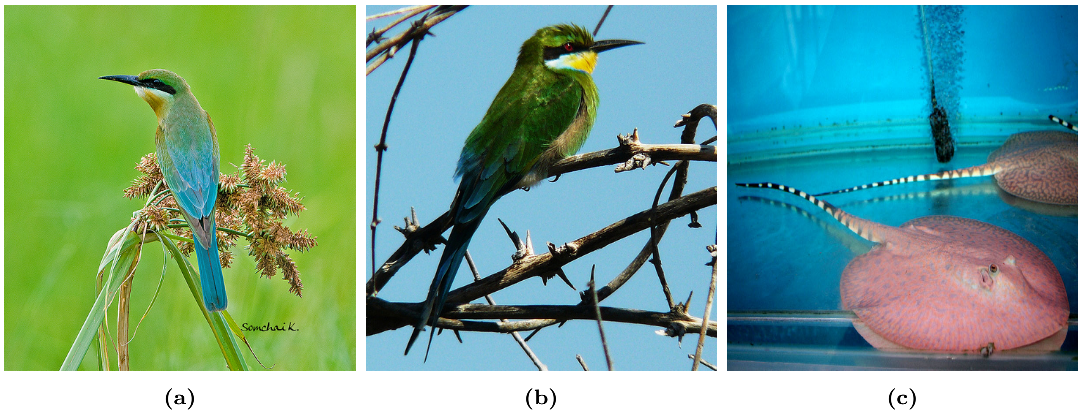


Figure 1.1: Need for routing throughout a network with parallel resources: Three samples from the 2012 [12] validation set of the ILSVRC. Two hummingbirds can be seen in the first two images, and an electric ray can be seen in the third. While images a and b contain similar abstract information like body stance, images b and c just share similar low-level attributes like the dominating color. Therefore, processing image b and c feature maps together in the initial layers of a multi-path network and image a and b feature maps together in the deeper levels may result in improved overall performance.

Thus, when a multi-path network with context-wise path routing learns a given task, the resource allocation patterns will vary along the depth of the network. For example, consider the images shown in Figure 1.1. These images are selected from the ILSVRC2012 [13] dataset. Where image 1.1a and image 1.1b depict hummingbirds sitting on branches, image 1.1c represents an electric ray in the water. The last two images share a blue background while the first two images share the same class of the bird. If we consider details that are usually extracted in shallow layers of a neural network (low-level details) like the overall color, image 1.1b and 1.1c are similar in nature to each other, whereas image 1.1a is different from the first two. However, if we consider more abstract details which are expected to be extracted at deeper layers, such as body pattern, image 1.1a and 1.1b are similar, and image 1.1c is different. Reflecting this phenomenon, images 1.1b and 1.1c can get equivalent path allocations in the initial layers of a multi-path network, while images 1.1a and 1.1b might get similar path allocations in latter stages. Hence, we need routing mechanisms along the whole depth of a network to facilitate this type of resource grouping according to the abstractness of the features in each layer.

1.4 Contributions

We explore novel, data-dependent mechanisms that can intelligently route information in a parallel path network end-to-end. We introduce such routing mechanisms layer-wise to allocate incoming parallel tensors (sets of feature maps) to a routing layer, to its output parallel paths. A routing layer facilitates cross-connections between the parallel tensors of adjacent layers and further weight these connections. We refer to such weighting coefficients as gate coefficients. The gate coefficients are produced from parametric functions which compute on the incoming feature maps. We insert such routing layers throughout the depth of the parallel path network to enable parallel resource allocation according to the level of context representing at that depth and end-to-end adaptive routing.

To further regulate the layer-wise routing process, we introduce a novel loss function which can cluster a sample space to a given number of categories without labels. We follow mixture modelling for clustering and propose to use a neural network to learn the cluster distributions and posterior assignments in mixture modelling. We show that following our approach, a neural network can successfully model complex cluster distributions which otherwise must be captured by hand-designed distributions with limited parameters. As a standalone end-to-end clustering approach, our framework shows superior performance to traditional clustering algorithms and deep clustering approaches which rely on k-means as the basic clustering technique.

Finally we incorporate this clustering loss in each cross-connection based routing layer in our multi-path networks to investigate the possible improvements to the regularization. We analyse the impact of such regularization on our multi-path network and its gating mechanism. We show that the clustering objective enables gates to be evenly activated minimizing skewed gate activations (e.g., only one gate mostly activating for all inputs). We further suggest the extension towards hard path allocation with gating networks and clustering objective governing the input allocation.

1.5 Summary

We emphasize the need for carefully engineered layer-wise feature extraction to achieve the overall utility of a network (performance in terms of the parameters/complexity employed). Exploiting this objective, we focus on employing parallel sets of feature maps in each layer and the need for data-dependent input allocation among these parallel paths. We also raise the need to include such allocation methods throughout the depth of the networks. Following these insights, we propose multi-path neural networks with data-dependent routing methods that route the traffic in multi-path networks end-to-end. We also propose a neural mixture modeling-based clustering approach that uses a neural network to learn cluster distributions and assignments concurrently. We validate the clustering objective in standard image clustering and further discuss the effect of such an objective to regularize gating in multi-path networks.

The rest of the dissertation is organized as follows. In Chapter 2, we provide the background and literature search on layer-wise feature extraction. Chapter 3, based on Tissera *et al.* 2019 [14], Tissera *et al.* 2020 [15] and Tissera *et al.* 2023 [16], describes the routing algorithms used and the performance evaluation of the proposed multi-path networks. Chapter 4 bases on Tissera *et al.* 2022 [17] and introduces neural network-based mixture modeling as a standalone deep clustering framework. In Chapter 5 we illustrate one use case of our clustering objective in regularizing the gating networks in our multi-path networks. Finally, in Chapter 6, we discuss the utilities, limitations and future directions.

Chapter 2

BACKGROUND

2.1 Neural Networks

Neural networks are parametric models that consist of multiple layers separated by non-linear activation functions. Given an adequate amount of hidden nodes, learning, and loss function, there is no theoretical limit to the approximation capability of a multi-layer perceptron [18]. A multi-layer perceptron [19, 20] employs several layers of neurons. One layer connects its input nodes to its output nodes via weights. The first layer's input nodes represent the input to the network, and the final layer's output nodes are the network's output. The intermediary layers carry hidden nodes. The weights are generally updated with backpropagation [21, 22], where the computed loss function during the forward pass is optimized by updating weights in the backward pass.

Neural network architectures have evolved, marking several significant milestones. Convolutional neural networks (CNNs) [4, 8, 22] are specifically designed to extract spatial features from images while reducing the complexity of employing a multi-layer perceptron. Instead of connecting all nodes in subsequent layers, CNNs maintain the 2D structure of the tensors and convolute them with a matrix of weights called filters to produce output feature maps of the layer. Recurrent neural networks (RNNs) [21, 23–25] are designed to capture the patterns distributed over data sequences. RNNs function similarly to a feed-forward network for each step of a sequence while also accommodating a memory component from previous steps. Transformers [26] parallelize the RNNs' sequential operations, which require less training time, and introduce multi-head attention, which is much more sophisticated in paying attention to the parts of the sequence than sequential memory in RNNs.

Neural Networks, with their immense capability of approximating complex functions and handling high dimensional data, have successfully marked the state-of-the-art in many domains including visual recognition [27–29], speech and audio processing [30, 31], time series prediction [32, 33], natural language processing [34, 35] and generative modeling [36–38]

2.2 Deepening Neural Networks

Neural networks perform better with more layers in the stack due to their ability to approximate rather complex patterns with more layers along the depth separated by non-linear activations. [4,39,40]. Besides, having more layers along the depth [4,5,8] has already shown better performance compared to shallow neural networks [20, 22]. However, Neural network deepening has its disadvantages, such as gradient vanishing/exploding [41], increased training time [6], performance degradation [1], and overfitting to training set [42].

Numerous solutions have been proposed to tackle such challenges of network deepening. Methods such as input normalization [22], intermediate tensor batch-normalization [43], and weight initialization [44, 45] help overcome gradient vanishing/exploding problems. In addition, shortcut connections that connect non-adjacent layers along depth [1, 2] enable stacking many layers without subjecting them to performance degradation. Furthermore, regularization methods [46, 47] prevent deep networks from overfitting to training data, and advanced optimizers [48, 49] also help mitigate issues with deep network training.

Overall, deepening neural networks is well explored and popular in application.

However, an optimum depth exists for a given task, beyond which merely adding layers might be inefficient regarding the performance gain due to the added complexity and training time [6]. Therefore, it is also intuitive to explore strengthening layer-wise feature extraction, which may lead to improved performance with respect to the employed complexity [50].

2.3 Enriching Layer-wise Feature Extraction—Widening

One way to enrich layer-wise feature extraction is to have more resources in each layer. Conventionally, this is achieved by increasing the filters/nodes in each layer [6]. Such naive widening is inefficient because the network parameters are quadratic in terms of the number of filters or nodes per layer. It is more intuitive to stack independent parallel operations in each layer [5, 7, 51, 52], which leads to an efficient scaling option where the number of network parameters is linear in terms of the number of stacks per layer. Still, without a mechanism to adaptively allocate incoming feature maps among such parallel computations in each layer, these parallel paths tend to learn redundant features and collapse to a single path [53].

In addition, there have been efforts to employ multi-path neural networks where each path learns in isolation. While model ensembles [4, 8] accumulate independently trained network responses for the same input to predict final output during inference, there are efforts to feed different versions of the same input (e.g., pre-processed with different techniques) to parallel paths of multi-path networks during training [9, 10]. However, these parallel paths do not connect with each other and learn in isolation until the final layer. Hence, there is no adaptive allocation of parallel resources distributed layer-wise. In addition, these methods are also subjected to the aforementioned feature redundancy among parallel paths [53].

To summarize, existing widening techniques are either inefficient in terms of added complexity, do not facilitate layer-wise connections among parallel paths and/or do not allocate inputs intelligently among parallel paths.

2.4 Multi-path Networks with Cross-connections

Enabling a multi-path network to allocate its parallel resources throughout the depth requires parallel paths to facilitate connections among them, preferably per a segment of layers. Multi-path networks with such cross-connections between parallel paths are widely used in the multi-task learning domain [54–56], where the model learns to perform multiple tasks on a single input (e.g., semantic segmentation and surface normal estimation).

In particular, Cross-Stitch Networks [57] first introduced cross-connections between parallel paths in a multi-path network. These cross-connections are weighted by additional coefficients, which are independently learned as regular weights. Therefore, those weighting coefficients are fixed during inference. There have been subsequent attempts to improve the intuition of such resource sharing [58, 59]. Such weighting intends to enable the multi-path network to learn the optimum mix of task-specific and shared parallel resources per layer segment to perform all given tasks.

However, we intend to route an input end-to-end in a multi-path network, where per each layer segment, an adaptive routing process shall take place to delegate incoming parallel tenors to next layer parallel tensors according to the nature of the incoming tensors. For such purpose, having the cross-connection weighting independently learned is insufficient; those weighting coefficients should be produced in a dynamic, input-dependent manner.

2.5 Adaptive Feature Extraction Methods

The idea of context-dependent resource allocation is inspired by existing adaptive feature extraction methods, which use additional mechanisms to make the primary feature extraction process dynamic and responsive to the input-context during inference. Such mechanisms are either parametric or non-parametric and usually compute on the input to the network [60] or input to a layer [61–73].

Hypernetworks [60] use a sub-network to predict the main network weights. Squeeze-and-excitation network (SENet) [63] re-calibrates each convolutional channel after a usual convolution using a parametric function computed on the convolutional output. This is commonly referred to as channel-wise attention, which has been subsequently adopted to improve existing networks via channel re-calibration [71, 72].

Highway Networks [68, 74] regulates the information flow along the depth of a network using a gating mechanism. Sharing a similar motivation to regulate information along the depth of a network, ConvNet-AIG [66], BlockDrop [67], and SkipNet [70] selectively use residual blocks in ResNets [1] to adjust the effective depth of the network according to the context of the input. Inspired by such adaptive mechanisms, we explore the adaptive allocation of parallel resources in a multi-path network layer-wise. In contrast to using a common single path, we use parallel paths and additional mechanisms to route among these paths per segment of layers so that the input is effectively routed end-to-end through the multi-path networks with a context-dependent soft combination of path activations.

2.6 Mixture of Experts

Our work and intuition is closely related with the Mixture of Experts domain. A mixture of experts [75, 76] partitions the input space into sub-spaces and picks experts to extract features in each sub-space. Initially, only complete models were employed as experts, but the following work introduced mixtures of experts layer-wise [77, 78]. Sparsely gated mixtures [53, 79] have achieved significant progress in fields such as natural language processing [78, 80, 81] and vision [82, 83]. However, sparse mixtures of experts require enormous amounts of data and a heavy reliance on network engineering among parallel devices during training. In contrast, we utilize soft allocation of parallel resources, which allows for single-device backpropagation.

2.7 Summary

We identify that despite the remarkable success of deepening neural networks, there is still room for layer-wise solid feature extraction. Exploiting this direction, we acknowledge existing widening methods and identify that stacking parallel operations is more efficient than conventional widening. However, we also note the need for an efficient input-dependent parallel resource allocation method. We also explore existing multipath networks in the multi-tasking domain where the coefficients weighting cross-connections are independently learned. To this end, we borrow from existing adaptive feature extraction methods that employ additional parametric functions to support the main task, leading to dynamic input-sensitive resource allocation. We thus move towards employing parallel stacks of feature maps in each layer and additional parametric functions to govern the cross-allocation of parallel layer resources.

Chapter 3

DATA-DEPENDENT END-TO-END ROUTING

We enable end-to-end routing in a multi-path network by introducing layer-wise routing algorithms. Such algorithms construct a set of parallel tensors (next layer) from a given set of parallel tensors (current layer) while accommodating all possible cross flows of information between the parallel paths of the two layers. An additional set of parametric functions take the incoming tensors as input and produces gating coefficients that weight these cross flows adaptively.

3.1 Cross-Prediction Based Routing

In the cross-prediction based routing algorithm, given a layer of parallel tensors, each tensor first predicts all parallel tensors in the next layers via independent convolutional or dense operations. In addition, each tensor also predicts a set of probabilities (coupling coefficients/ gates) each denoting the probability of routing to each tensor in the next layer. Given these predictions and gates, each tensor in the next layer is constructed by summing all corresponding gate-weighted predictions made by previous layer tensors to that particular tensor. Figure 3.1 depicts routing between two layers having m and n tensors. There, Figure 3.1a depicts one tensor in the first layer predicting second-layer tensors and coupling probabilities to them. Figure 3.1b depicts the construction of the second layer using the predictions and gates computed by previous layer tensors.

Let $[\mathbf{X}_{i=1,\dots,m}]$ be m parallel tensors in a given layer (say inputs to the routing

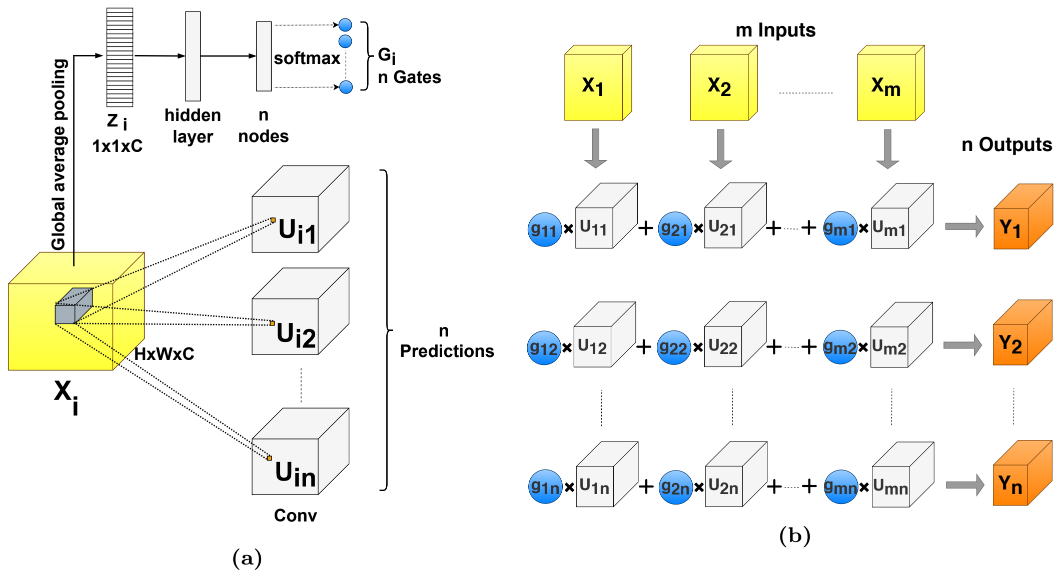


Figure 3.1: Routing layer of m inputs and n outputs. a) Input $\mathbf{X_i}$ predicting n outputs and associated gates. b) Constructing n outputs $\mathbf{Y}_{j=1,\ldots,n}$ using the predictions and gates computed by m inputs $\mathbf{X}_{i=1,\ldots,m}$. See Eq. 3.6.

layer) and $[\mathbf{Y}_{j=1,\ldots,n}]$ be the n tensors which the routing algorithm constructs as the next layer (say outputs of the routing layer). Each input $\mathbf{X}_i$ first predicts each output tensor via a linear parametric computation. $\mathbf{U}_{ij}$, the prediction made from $\mathbf{X}_i$ to $\mathbf{Y}_j$ is,

$$\mathbf{U}_{ij} = \mathrm{W}_{ij}\mathbf{X}_i + b_{ij}. \tag{3.1}$$

W_{ij} and b_{ij} are weights and biases of the linear transformation. This corresponds to a convolutional operation in case where $\mathbf{X}$ and $\mathbf{Y}$ are 3-dimensional. $\mathbf{X}_i$ also predicts n gating coefficients stacked as an n-dimensional vector $\mathbf{G}_i$,

$$\mathbf{G}_i = [g_{i1}, \ldots, g_{in}]. \tag{3.2}$$

These n coefficients are the soft probabilities of routing $\mathbf{X}_i$ to each of the next layer tensors. g_{ij} is the probability of $\mathbf{X}_i$ getting routed to $\mathbf{Y}_j$, i.e., the soft gate value in the connection between $\mathbf{X}_i$ and $\mathbf{Y}_j$.

A non-linear parametric calculation on $\mathbf{X}_i$, preferably two dense operations separated by ReLU activation, can be used to determine $\mathbf{G}_i$. However, if $\mathbf{X}_i$

is three-dimensional ($H \times W \times C$, where H, W and C correspond to height, width and number of channels in the tensor), a large number of parameters are occupied. Therefore, in this case, we first feed $\mathbf{X}_i$ through a global average pooling operation to obtain the latent channel descriptor $\mathbf{Z}_i$ of size $1 \times 1 \times C$ [63, 66]. Global average pooling produces a compressed descriptor that still contains the information about the presence of each feature since each channel in a set of convolutional feature maps reflects a distinct feature of the input that is sought by a particular filter. The c^{th} channel value $(z_i)_c$ of the channel descriptor $\mathbf{Z}_i$ is,

$$(z_i)_c = \frac{1}{H \times W} \sum_{a=1}^{H} \sum_{b=1}^{W} (x_i)_{a,b,c}, \tag{3.3}$$

where $(x_i)_{a,b,c}$ stands for the pixel of location (a, b, c) in tensor $\mathbf{X}_i$. Then, $\mathbf{Z}_i$ is sent into a non-linear computation that has two fully connected layers (weights W_1 and W_2) which are separated by ReLU activation. This process produces n latent relevance scores $\mathbf{A}_i$ ($[a_{i1}, \ldots, a_{in}]$).

$$\mathbf{A}_i = \mathrm{W}_2(\mathrm{ReLU}(\mathrm{W}_1 \mathbf{Z}_i)) \tag{3.4}$$

We finally use softmax activation on n relevance scores $\mathbf{A}_i$ to determine the gate probabilities $\mathbf{G}_i$.

$$\mathbf{G}_i = \mathrm{softmax}(\mathbf{A}_i), \text{ i.e., } g_{ij} = \frac{\mathrm{e}^{a_{ij}}}{\sum_{k=1}^{n} \mathrm{e}^{a_{ik}}}. \tag{3.5}$$

The resultung n scores represent the probabilities of $\mathbf{X}_i$ being routed to each output $\mathbf{Y}_{j=1,\ldots,n}$. Figure 3.1a depicts the processes a 3-dimensional tensor performs at a routing layer's input during the prediction stage.

With cross-predictions $\mathbf{U}_{ij}$ and gates $\mathbf{G}_i$ calculated, we then derive the routing layer outputs. To construct j^{th} output $\mathbf{Y}_j$, predictions made for $\mathbf{Y}_j$ ($\mathbf{U}_{ij}$, $i = 1, \ldots, m$) are weighted by corresponding gate values (g_{ij}, $i = 1, \ldots, m$) and added together. We further use ReLU activation on the constructed tensor.

$$\mathbf{Y}_j = \text{ReLU}\left(\sum_{i=1}^{m}(g_{ij} \times \mathbf{U}_{ij})\right) \tag{3.6}$$

This adaptive re-calibration of the input tensors' predictions that are used to build the output tensors bears a similar idea of attention to that presented in SENets [63]. Our aim, though, is soft-routing information in diverse directions. Algorithm 1 illustrates the routing between two layers in further detail.

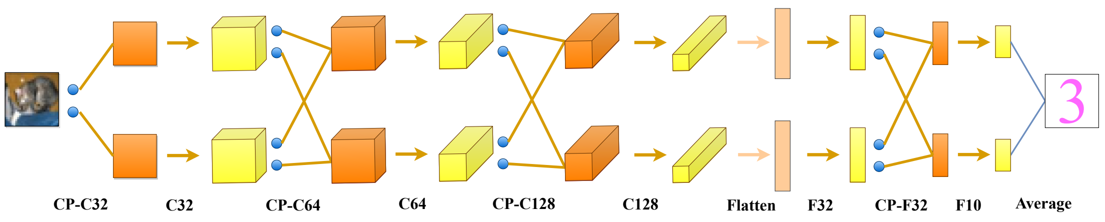


Figure 3.2: Insertion of cross-prediction-based routing layers to a two-path CNN in image classification task. CP-Cn stands for a routing layer of n-filter convolutional cross-predictions, whereas CP-Fn stands for a routing layer of n-node dense cross-predictions. Cn represents a forward layer with parallel n-filter convolutions, and Fn represents a forward layer with parallel n-node dense layers.

Algorithm 1 Cross-Prediction based routing between two layers.

```
Input: X {[X_i for $i = 1, 2, \dots, m$]}
Predictions from current layer:
    for $i = 1$ to $m$ do
      for $j = 1$ to $n$ do
          $\mathbf{U}_{ij} \leftarrow \mathrm{W}_{ij}\mathbf{X}_i + b_{ij}$
      end for
      Gate Computation on X_i:
      $\mathbf{Z}_i \leftarrow \text{global\_average\_pooling}(\mathbf{X}_i)$
      $\mathbf{A}_i = [a_{i1}, \dots, a_{in}] \leftarrow \mathbf{W}_2^i(\text{ReLU}(\mathbf{W}_1^i \boldsymbol{Z}_i))$
      $\mathbf{G}_i = [g_{i1}, \dots, g_{in}] \leftarrow \text{softmax}(\mathbf{A}_i)$
    end for
Construction of outputs:
    for $j = 1$ to $n$ do
      $\mathbf{Y}_j \leftarrow \text{ReLU}(\sum_{i=1}^{m}(g_{ij} \times \mathbf{U}_{ij}))$
    end for
Output: Y {[$\mathbf{Y}_j$ for $j = 1, 2, \dots, n$]}
```

We place these routing layers between a few different layers in multipath networks (Figure 3.2), allowing other layers to have their parallel independent paths to learn in isolation. Figure 3.2 depicts a two-path convolutional neural network with our routing inserted at specific spots referred to later as BaseCNN-2-CP. Due to the convolutional or dense nature of the cross-predictions, adding one routing layer raises the network's effective depth by one layer. Before feeding the parallel tensors to the subsequent feed-forward computation, we impose non-linear *ReLU* activation because the output layer tensors are mixtures of linear operations. Finally, the parallel feature maps are averaged to create a single output in the final layer.

The number of parameters used for the cross-predictions in the routing process between two layers is susceptible to a quadratic increase with the number of parallel paths because each tensor in a given layer predicts each tensor in the next layer in terms of convolution or dense operation. We, therefore, introduce cross-connection-based routing to reduce the routing layer to gate computation and directly weighting inputs with the gates to construct outputs. Such design will be much efficient for scaling in terms of the number of parameters.

3.2 Cross-Connection Based Routing

Cross-connection-based routing directly weighs the current layer tensors to create the next layer tensors rather than weighting the convolutional or dense cross-predictions. By doing this, the quadratic parameters addition by the cross-predictions in terms of the number of paths can be eliminated. Instead of cross-predictions contributing to the effective stack of the network, forward layers with parallel convolutional or dense operations solely focus on learning the main task. A routing layer is now only a cross-connecting layer and no longer carries weights that aid in learning the primary objective. Therefore, adding cross-connections between layers enables soft routing without deepening the network. The computations for the non-linear gates have contributed just a tiny number of parameters to the routing overhead.

We now explain how given a layer of m tensors $[\mathbf{X}_{i=1,\ldots,m}]$, cross-connecting-based routing produces the next layer of n tensors $[\mathbf{Y}_{j=1,\ldots,n}]$. Each $\mathbf{X}_i$ computes the gate vector $\mathbf{G}_i$ ($[g_{i1}, \ldots, g_{in}]$) as in cross-prediction-based routing (Eq. 3.3, Eq. 3.4 and Eq. 3.5). The algorithm next computes each next layer tensor $\mathbf{Y}_j$ by summing the previous layer tensors $[\mathbf{X}_{i=1,\ldots,m}]$ each weighted by the corresponding gate $g_{ij,i=1,\ldots,m}$:

$$\mathbf{Y}_j = \sum_{i=1}^{m} (g_{ij} \times \mathbf{X}_i). \tag{3.7}$$

The output tensor dimensions of a cross-connection-based routing layer are the same as its input tensors since the routing directly connects its inputs to create its outputs. In Figure 3.3, we present the cross-connecting procedure between two layers carrying two parallel tensors and the adaptive cross-connecting procedure by Algorithm 2. In Figure 3.4, we show the insertion of such routing layers at selected locations in a two-path CNN which is referred as BaseCNN-2-CC later.

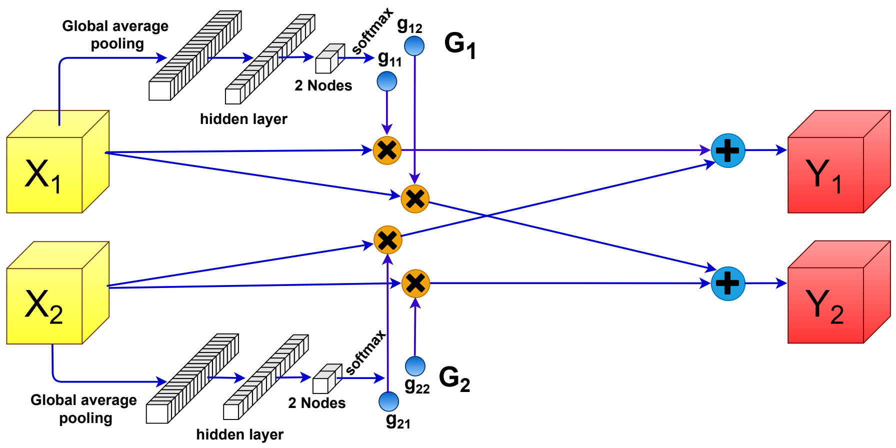


Figure 3.3: Cross-connecting two layers that have two parallel tensors on each layer. Learnable parametric computations use the input tensors to compute the gates that weigh the connections.

We also illustrate the pixel-wise operations of the cross-connecting process in matrix form. Consider a set of 3-dimensional input tensors $[\mathbf{X}_{i=1,\ldots,m}]$ to the routing layer and its output tensors $[\mathbf{Y}_{j=1,\ldots,n}]$. Lets denote the pixel value at the location (a, b, c) of $\mathbf{X}_i$ as $(x_i)_{a,b,c}$, and $\mathbf{Y}_j$ as $(y_j)_{a,b,c,}$. The set of output pixels at

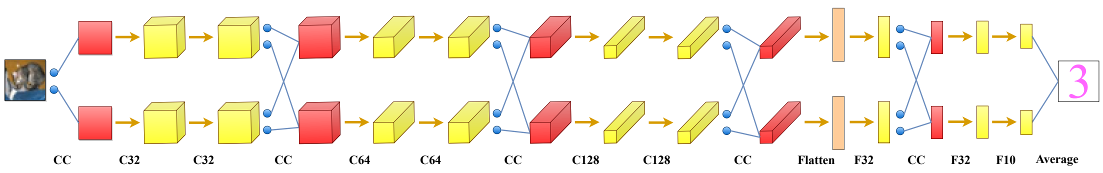


Figure 3.4: CNN with two paths and adaptive cross-connections placed at specific locations. CC stands for a cross-connecting layer, whose gates, connections, and outputs are represented by blue circles, blue edges and red boxes. Cn and Fn are forward convolutional and dense layers, respectively. Yellow boxes are used to represent the outputs of these forward layers. Adding cross-connecting layers does not increase the network's effective depth because the cross-connections are merely weighted connections.

Algorithm 2 Cross-connection-based routing between two adjacent layers with m and n sets of feature maps respectively.

```
Input:
    $\mathbf{X}$: inputs $\{[\mathbf{X}_i$ for $i = 1, \ldots, m]\}$
Calculating gate values:
    for $i = 1$ to $m$ do
      $\mathbf{Z}_i \leftarrow$ global_average_pooling$(\mathbf{X}_i)$
      $\mathbf{A}_i = [a_{i1}, \ldots, a_{in}] \leftarrow \mathbf{W}_2^i(\mathrm{ReLU}(\mathbf{W}_1^i \boldsymbol{Z}_i))$
      $\mathbf{G}_i = [g_{i1}, \ldots, g_{in}] \leftarrow \mathrm{softmax}(\mathbf{A}_i)$
    end for
Construction of outputs:
    for $j = 1$ to $n$ do
      $\mathbf{Y}_j \leftarrow \sum_{i=1}^{m}(g_{ij} \times \mathbf{X}_i)$
    end for
Return:
    $\mathbf{Y}$: outputs $\{[\mathbf{Y}_j$ for $j = 1, \ldots, n]\}$
```

(a, b, c) are therefore,

$$\begin{bmatrix} (y_1)_{a,b,c} \\ \vdots \\ (y_n)_{a,b,c} \end{bmatrix} = \begin{bmatrix} g_{11} & \cdots & g_{m1} \\ \vdots & \ddots & \vdots \\ g_{1n} & \cdots & g_{mn} \end{bmatrix} \begin{bmatrix} (x_1)_{a,b,c} \\ \vdots \\ (x_m)_{a,b,c} \end{bmatrix}. \tag{3.8}$$

This formulation is similar to Cross-Stitch Networks [57]. The coupling coefficients of Cross-Stitch networks g_{ij}, however, are trained independently. Such independently trained coefficients only enable learning the mix of shared and task-specific resource allocation to execute many tasks on a single input. Such a learned mix is fixed during inference. Our approach uses the channel-wise attention mechanism [63] to construct g_{ij}s through a parametric computation on the inputs $\mathbf{X}_i$. Such adaptive gate computation enables dynamic switching between context-specific and shared representations based on the nature of the various inputs.

3.3 Backpropagation through a Cross-Connection Layer

In Section 3.2, we presented how cross-connection enables soft routing in a context-specific way. Backpropagation of gradients through cross-connections is necessary for network training. However, a cross-connecting layer's backpropagation is not as simple as in Cross-Stitch networks where the coupling coefficient matrix comprises independently learned weights. In our instance, the routing inputs $\mathbf{X}$ are used to create the elements of the gating matrix $\mathbf{G}$. As a result, in addition to the direct gradient weighted by the gate element, the gradients flown to each routing input $\mathbf{X}_i$ include another component from the gate computation. Also, the weights that produce the gates are optimized rather than the actual gates.

For clarity, let us assume that the tensors $\mathbf{X}$ and $\mathbf{Y}$ are k-dimensional vectors and that the gate computation uses a straightforward fully-connected layer rather than Eq. 3.3 and Eq. 3.4. This simplified cross-connecting technique for two-parallel paths is shown in Figure 3.5. Calculating the relevance scores $\mathbf{A}_i$ from each $\mathbf{X}_i$ now reduces to,

$$\mathbf{A}_i = \mathbf{W}^i \mathbf{X}_i, \tag{3.9}$$

where $\mathbf{W}^i$ is a $n \times k$ matrix of weights. $\mathbf{G}_i$ is computed by taking $softmax$ of these logits as in Eq. 3.5 and next layer tensors $\mathbf{Y}_{j\,(j=1 \ldots n)}$ are constructed as in Eq. 3.7. We calculate the gradients w.r.t each $\mathbf{X}_{i\,(i=1 \ldots m)}$ and $\mathbf{W}^i_{(i=1 \ldots m)}$, given the gradients of loss w.r.t. each output $\mathbf{Y}_{j\,(j=1 \ldots n)}$. I.e., given $\frac{\partial L}{\partial \mathbf{Y}_j}_{j=1 \ldots n}$, we compute $\frac{\partial L}{\partial \mathbf{W}^i}_{i=1 \ldots m}$ and $\frac{\partial L}{\partial \mathbf{X}_i}_{i=1 \ldots m}$. Figure 3.5 shows the flow of gradients to $\mathbf{W}^1$ and $\mathbf{X}_1$

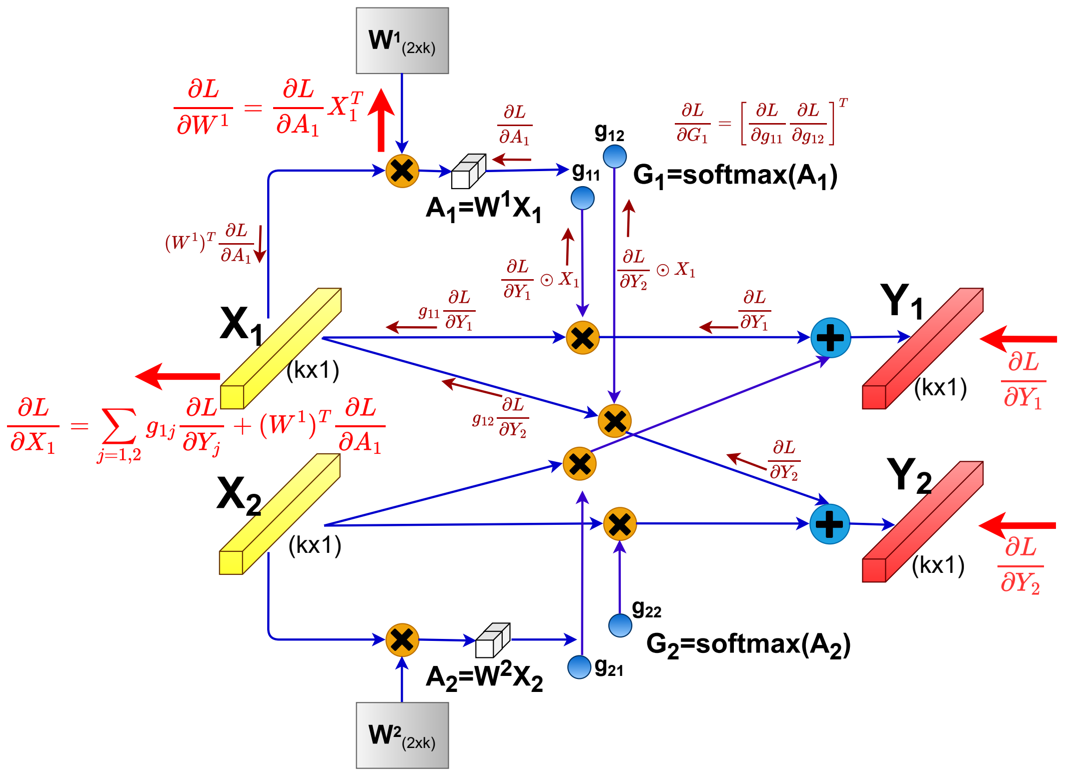


Figure 3.5: Backpropagation through the simplified cross-connecting layer between two successive layers each containing two parallel tensors. Gradient flows to the input layer's top tensor X_1 and its gate computation weight matrix W^1 are shown.

from $\mathbf{Y}_{j(j=1,2)}$ in a two-path cross-connecting layer that aids understanding the detailed gradient flow as explained below.

The incoming gradient must first be propagated to each g_{ij}. When creating $\mathbf{Y}_j$; the scalar g_{ij} multiplies each component of $\mathbf{X}_i$ (Eq. 3.7). Consequently, the partial derivative of loss relative to g_{ij} is the sum of the element-by-element multiplication between the gradient vector and $\mathbf{X}_i$,

$$\frac{\partial L}{\partial g_{ij}} = \sum_k \frac{\partial L}{\partial \mathbf{Y}_j} \odot \mathbf{X}_i. \tag{3.10}$$

With all such $\frac{\partial L}{\partial g_{ij}}_{(j=1,\ldots,n)}$, we form $\frac{\partial L}{\partial \mathbf{G}_i}$ as an n-dimensional column vector,

$$\frac{\partial L}{\partial \mathbf{G}_i} = \begin{bmatrix} \frac{\partial L}{\partial g_{i1}} & \cdots & \frac{\partial L}{\partial g_{in}} \end{bmatrix}^T. \tag{3.11}$$

By multiplying the gradients with respect to $\mathbf{G}_i$ by the partial derivative of gate values with respect to the relevance scores $\frac{\partial \mathbf{G}_i}{\partial \mathbf{A}_i}$, gradient is propagated to the relevance scores $\mathbf{A}_i$;

$$\frac{\partial L}{\partial \mathbf{A}_i} = \frac{\partial \mathbf{G}_i}{\partial \mathbf{A}_i}^T \frac{\partial L}{\partial \mathbf{G}_i} = \left(J_{\mathbf{A}_i}^{\mathbf{G}_i}\right)^T \frac{\partial L}{\partial \mathbf{G}_i}. \tag{3.12}$$

Here, $J_{\mathbf{A}_i}^{\mathbf{G}_i}$ stands for the Jacobian matrix of the derivative of softmax,

$$\frac{\partial \mathbf{G}_i}{\partial \mathbf{A}_i} = J_{\mathbf{A}_i}^{\mathbf{G}_i} = \begin{bmatrix} g_{i1}(1-g_{i1}) & \cdots & -g_{i1}g_{in} \\ \vdots & \ddots & \vdots \\ -g_{in}g_{i1} & \cdots & g_{in}(1-g_{in}) \end{bmatrix}. \tag{3.13}$$

By propagating the gradient w.r.t. $\mathbf{A}_i$ through Equation 3.9, it is now possible to determine the gradients of loss w.r.t. $\mathbf{W}^i$. Therefore,

$$\frac{\partial L}{\partial \mathbf{W}^i} = \frac{\partial L}{\partial \mathbf{A}_i}\mathbf{X}_i^T = \left(J_{\mathbf{A}_i}^{\mathbf{G}_i}\right)^T \frac{\partial L}{\partial \mathbf{G}_i}\mathbf{X}_i^T. \tag{3.14}$$

It is also important to calculate the gradient of loss w.r.t $\mathbf{X}_i$ since it is propagated to the previous layer.

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{X}_i} &= \sum_{j=1}^{n} g_{ij}\frac{\partial L}{\partial \mathbf{Y}_j} + (\mathbf{W}^i)^T \frac{\partial L}{\partial \mathbf{A}_i} \\ &= \sum_{j=1}^{n} g_{ij}\frac{\partial L}{\partial \mathbf{Y}_j} + \left(\mathbf{W}^i\right)^T \left(J_{\mathbf{A}_i}^{\mathbf{G}_i}\right)^T \frac{\partial L}{\partial \mathbf{G}_i} \end{aligned} \tag{3.15}$$

Here, the first term is the direct gradient flow from the multiplication between g_{ij} and $\mathbf{X}_i$, to $\mathbf{X}_i$. The second term is the portion of the gradient propagated to g_{ij} from that particular multiplication flowing back to $\mathbf{X}_i$. This residual gradient flow results from the attention-like production of g_{ij} from $\mathbf{X}_i$.

3.4 Experiments

We conduct various image recognition experiments to demonstrate the efficacy of parallel paths with data-dependent resource allocation. First, we study the effects of transforming regular convolutional neural networks into their equivalent multi-path variants, followed by experimenting with custom Residual Networks (ResNets) [1] with parallel paths that apply our routing algorithms. In each scenario, we assess how well our architectures perform compared to existing, more complex networks, adaptive feature extraction techniques, and deeper networks with comparable parametric complexity. Whenever we cannot discover existing approaches' variants that roughly match our models' complexity, we create new variants whose parametric complexity roughly resembles our models.

3.4.1 Conventional Convolutional Neural Networks with Parallel Paths

This section compares parallel-path convolutional neural networks with traditional network widening, network deepening, and other related networks. Table 3.1 illustrates the specifics of the networks we employ. As the baseline, we select BaseCNN, a 9-layer convolutional neural network with six convolutional layers and three dense layers. We expand BaseCNN by stacking parallel paths with routing layers to construct its multi-path networks.

BaseCNN-X-CP stands for an X-path network with cross-prediction-based routing, where each path is similar to a BaseCNN. The BaseCNN-2-CP design, which employs two parallel paths, is shown in Figure 3.2. Here, cross-prediction-based routing layers are used in place of the first, third, and fifth convolutional layers, as well as the second dense layer. Cross-predictions are convolutional or dense operations; therefore, one routing layer increases the network's effective depth by one layer. Therefore, to preserve the same depth as BaseCNN, we substitute the chosen layers in the parallel-path BaseCNN with the routing layers.

Table 3.1: Notations and details of the compared convolutional neural networks: Cn stands for a n-filter convolutional layer. Fn stands for a n-node fully connected layer.

Network	Structure
BaseCNN	$C32$ $C32$ $C64$ $C64$ $C128$ $C128$ $F32$ $F32$ $F10$
WideCNN	$C64$ $C64$ $C128$ $C128$ $C256$ $C256$ $F32$ $F32$ $F10$
DeepCNN	$C32$ $C32$ $C64$ $C64$ $C128$ $C128$ $C128$ $C256$ $C256$ $C256$ $F32$ $F32$ $F10$
BaseCNN-X	BaseCNN–X paths. No routing.
Base Ensemble	Ensemble of 3 BaseCNNs
All Ensemble	Ensemble of BaseCNN, WideCNN and DeepCNN
SEBaseCNN	SENet ([63]) on BaseCNN
SEDeepCNN	SENet ([63]) on DeepCNN
Cr-Stitch2	Cross-stitch network ([57]) with 2 parallel BaseCNNs
BaseCNN-X-CP	BaseCNN–X paths–cross-prediction based routing
BaseCNN-X-CC	BaseCNN–X paths–cross-connections

The final prediction is created by averaging the final layer parallel outputs.

BaseCNN-X-CC stands for an X-path network with adaptive cross-connections. The BaseCNN-2-CC design, which has two parallel routes, is shown in Figure 3.4. After the second, fourth, and sixth convolutions, as well as after the first dense layer, we add cross-connections. We also add a one-to-many connector to expand the input image to parallel routes at the beginning of the network (One-to-many connector is a cross-connecting layer but with one tensor in the input layer and the required number of tensors in the output layer). The effective depth of the network is not increased by adding a cross-connection-based routing layer because it only contains cross-connections and weighing coefficients. As a result, we can add such layers to the BaseCNN multi-path network without substituting any forward layers.

Next, we build comparable architectures representing conventional widening, deepening and adaptive feature extractors. We widen the BaseCNN conventionally and create WideCNN by doubling the filter size in each convolution. The DeepCNN architecture is created by deepening the BaseCNN with additional convolutional layers. We construct BaseCNN-X, an identical multi-path network to BaseCNN-X-CC but without intermediary routing. Here, X is the number of parallel BaseCNNs sharing the same input and output (averaging). We employ an ensemble of three individually trained BaseCNNs to compare with model ensembles (Base Ensemble). The individual BaseCNN outputs are averaged at inference to provide the Base Ensemble's output. We also create All Ensemble, an ensemble of BaseCNN, WideCNN, and DeepCNN. We incorporate SE operations [63] into the convolutional layers of BaseCNN and DeepCNN to produce SEBaseCNN and SEDeepCNN, respectively, to compare our multi-path networks with corresponding SENets. To create an equal two-path Cross-Stitch Network, Cr-Stitch2, we add cross-stitching procedures in place of BaseCNN-2-CC's adaptive cross-connections.

We first train these models for 200 epochs with a 128-batch size on the CIFAR10 dataset [84]. We employ stochastic gradient descent (SGD) with a momentum of 0.9 and an initial learning rate of 0.1, which decays by a factor of 10 after 80 and 150 epochs. To augment images, we add random horizontal flipping and pixel shifts in both directions, up to a maximum shift of 4 pixels. We display the results of this investigation in Table 3.2, where we present the best performance out of three trials for each model.

Our routing techniques enhance BaseCNN's performance when added with parallel paths to outperform traditional widening. In this scenario, the BaseCNN with two paths and routing (BaseCNN-2-CP/CC) outperforms the WideCNN, which has two times filters in each layer. WideCNN carries four times as many parameters as BaseCNN because of the quadratic increment of parameters in traditional widening, whereas adding two parallel routes doubles the number of parameters. Even with the additional routing overhead, BaseCNN-2-CP still has fewer parameters than WideCNN. BaseCNN-2-CC carries nearly the same number of parameters as two BaseCNNs due to the very minimal additional routing overhead added by cross-connection-based routing.

BaseCNN-3-CP/CC outperforms BaseCNN-3 (no routing) by a significant

Table 3.2: CIFAR10 CNN ablation study: Classification errors (%) BaseCNNs with parallel paths and routing outperform conventional widening, model ensembles, SENets, Cross-stitch networks, and even conventional deepening at equivalent or lower complexity. All networks are trained for 200 epochs. We also report multi-path networks' performance after training for 350 epochs to set the benchmark. * indicates performance reported in the relevant paper.

Network	Params (M)	Error% (200 epochs)	Error% (350 epochs)
BaseCNN	0.55	9.26	
WideCNN	1.67	8.96	
DeepCNN	2.0	8.49	
BaseCNN-3	1.5	9.41	
BaseCNN Ensemble	1.66	7.87	
All Ensemble	4.27	6.9	
SEBaseCNN	0.58	8.99	
SEDeepCNN	2.06	8.15	
Cr-Stitch2	1.11	7.89	
VGG16 [4]	14.9	6.98	
Capsule Nets* [64]	8.2	10.6	
Highway* [68, 74]	2.3	7.54	
BaseCNN-2-CP	1.3	7.24	**6.48**
BaseCNN-3-CP	2.23	**6.63**	**6.04**
BaseCNN-4-CP	3.34	**6.45**	**5.91**
BaseCNN-2-CC	1.11	7.03	**6.53**
BaseCNN-3-CC	1.67	**6.51**	**6.09**
BaseCNN-4-CC	2.22	**6.55**	**6.26**

margin. Moreover, BaseCNN-3-CP/CC outperforms the ensemble of 3 BaseCNNs as well as the ensemble of BaseCNN, WideCNN, and DeepCNN. This implies that our routing methods improve the performance of stacked parallel paths due to the data-dependent resource allocation. Furthermore, the DeepCNN, whose total number of parameters is more than three times that of the BaseCNN, is outperformed in this trial by BaseCNN-2-CP/CC. Our multi-path networks even outperform the VGG16 [4], which uses many parameters along with depth and width.

BaseCNN-2-CP/CC outperforms the cross-stitch network (Cr-Stitch2) with two paths, demonstrating that adaptive cross-routing is preferable to independently learned cross-stitching coefficients for learning a task while handling diversity. BaseCNN-2-CP/CC outperforms SE Nets created using the WideCNN and

DeepCNN, demonstrating the superiority of using parallel paths to re-calibrating a single path. Our method outperforms Highway networks [68] and Capsule Networks [64] among rich layer-wise feature extraction or adaptive feature extraction at a similar or lower complexity.

CIFAR10 studies show that BaseCNN-2-CP/CC, which adds a parallel path to BaseCNN, performs significantly better. However, the third parallel path (BaseCNN-3-CP/CC) addition yields less performance gain than the addition of the second path. Adding the fourth path, BaseCNN-4-CP/CC, provides little to no improvement. To acquire the optimum performance for the number of parameters used, it is crucial to carefully construct the number of parallel pathways according to the dataset. However, all deepening [1, 2] and widening [6, 7] procedures exhibit this phenomenon.

With more parallel paths, the multi-path networks with cross-connections (BaseCNN-X-CC) employ much fewer parameters than with cross-prediction-based routing (BaseCNN-X-CP). This is because adaptive cross-connections significantly reduce routing overhead by excluding cross-convolutional or cross-dense operations from cross-prediction-based routing. Furthermore, cross-connection-based routing yields similar performance as cross-prediction-based routing for less complexity, hence giving better performance with regard to model complexity.

We further set the benchmark for CNN-based multi-path networks: Our multi-path nets are retrained using the same parameters as before, but for 350 epochs, with the learning rate decay after 150 and 250 epochs. The last column of Table 3.2 displays the benchmark values.

3.4.2 Residual Networks with Parallel Paths

Next, we add parallel paths and our routing strategies to the residual networks (ResNets) [1]. First, we add parallel routes to the ResNet variations intended to learn from small-scale datasets (ResNet20, ResNet32, etc.). Following a first convolution, these models employ three successive stacks, each including multiple residual blocks (in ResNet20, three residual blocks). Each stack begins with a strided residual block, producing feature maps that have been down-sampled. Finally, a global average pooling layer and the last dense layer, which produces the class probabilities, mark the network's termination.

Extending ResNets with parallel paths and cross-prediction-based routing (ResNet-X-CP) is as follows. First, we replace the initial convolutional layer with a convolutional one-to-many routing layer. Then we add two additional routing layers before the second and third stacks. The final output is created by averaging the parallel dense layer outputs. This design increases the effective depth of the network by two extra levels. To create parallel-path ResNets with cross-connection-based routing (ResNet-X-CC), after the first convolution, we add a one-to-many router and three more cross-connection-based routers after the first, second, and third stacks. This architecture keeps the network's original depth because these cross-connections do not contain convolutions.

We employ a setup identical to the prior study to train our models on the CIFAR10 and CIFAR100 [84] datasets. We train our models for 350 epochs using a batch size of 64, with the learning rate decaying after 150 and 250 epochs. We run three trials for each model, and the best result is reported. The recorded classification errors of our models are displayed in Table 3.3, together with the classification errors of traditional ResNets and ResNet-based adaptive feature extractors.

WideResNet40-2 (WRN-40-2) has a depth of 40 layers and uses two filters in each convolutional layer. Despite using a smaller number of parameters, the Hyper Network [60] developed on top of WRN-40-2 (Hyper-WRN-40-2) performs worse than WRN-40-2. ResNet20 with three paths and routing (ResNet20-3-CP/CC) outperforms WRN-40-2. ResNet20 with two paths outperforms ResNet110 in CIFAR10, and ResNet20 with three paths outperforms ResNet110 in CIFAR100. This is remarkable because ResNet20 is much shallower than ResNet110,

Table 3.3: Comparison of ResNet variants. ResNet20-3 performs better than ResNet110. ResNet20-3/4 and ResNet32-3/4 perform as well as or better than existing adaptive designs, most of which are based on ResNet110.

Network	Params (M)	CIFAR10	CIFAR100
ResNet20 [1]	0.27	8.75	-
ResNet110	1.7	6.61	26.88
ResNet164	2.5	5.93	25.16
WRN-40-2 [6]	2.2	5.33	26.04
Hyper-WRN-40-2 [85]	0.15	7.23	-
SEResNet110 [63]	1.7	5.21	**23.85**
BlockDrop [67]	1.7	6.4	26.3
ConvNet-AIG [66]	1.78	5.76	-
ConvNet-AIG all [66]	1.78	5.14	-
ResNet20-2-CP	0.59	5.86	27.7
ResNet20-3-CP	0.92	**4.99**	**25.13**
ResNet20-4-CP	1.29	**4.81**	**23.82**
ResNet20-2-CC	0.55	5.5	27.36
ResNet20-3-CC	0.82	5.18	25.76
ResNet20-4-CC	1.1	**4.96**	**24.81**
ResNet32-2-CC	0.94	5.14	25.96
ResNet32-3-CC	1.41	**4.96**	**24.51**
ResNet32-4-CC	1.88	**4.59**	**23.52**

and even after parallel paths (2/3/4) are added, the total number of parameters is still lower than ResNet110.

Multi-path networks based on ResNet also outperform ResNet110-based adaptive feature extraction techniques. The ResNet110-based BlockDrop network [67] performs worse than all of our multi-path networks with CIFAR10 and just slightly better than ResNet20-2-CP/CC with CIFAR100. Compared to ConvNet-AIG [66], which bases on ResNet110, ResNet20-3/4-CP, ResNet20-4-CC, and ResNet32-3/4-CC perform better. With CIFAR10, except for ResNet20-2-CC/CP, all of our multi-path networks outperform the SENet [63] which builds on ResNet110 with identity mappings [2]. ResNet20-4-CP performs similarly to SEResNet110 in CIFAR100, whereas ResNet32-4-CC outperforms it. Other than ResNet32-4-CC, all of our multi-path networks have fewer parameters than

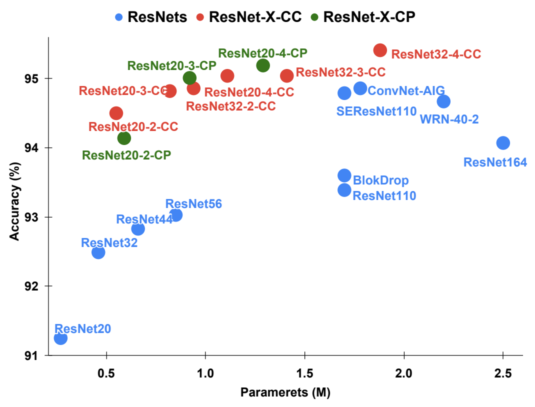
ResNet110-based networks.

It is important to note the effect of the depth in these experiments. While all the networks we compared ours with have at least 110 layers, our networks maximally contain 32 layers. The deeper networks we compared with generally performed better in CIFAR100 than in CIFAR10. This phenomenon is particularly evident in SEResNet110 as it performs inferior to all our networks except ResNet20-2-CC/CP with CIFAR10 but shows competent performance with our networks with CIFAR100. This observation adds that an optimum depth and non-linearity are suitable to approximate the solution for a given task at a given complexity. In this case, CIFAR100, being a more complex dataset than CIFAR10, can benefit from more layers in the stack than CIFAR10. Thus, it is essential to attribute the suitable depth for each task while strengthening layer-wise feature extraction with the chosen depth.

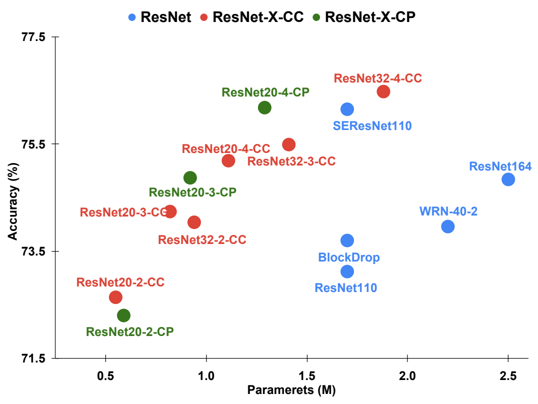
The CIFAR accuracy comparisons, along with the number of parameters in each network, are shown in Figure 3.6. These charts reveal that our multi-path networks exhibit the highest network utility for the chosen set of parameters. For a given depth, Multi-path ResNets gives the best performance with cross-prediction-based routing. However, we favor cross-connection-based multi-path ResNets because they have a more straightforward routing method that significantly reduces the widening's routing overhead.

3.4.3 Multi-path ResNets on ILSVRC2012

We further evaluate multi-path ResNets in the ILSVRC 2012 [12, 13], a 1000-class image dataset containing 1.3M training images and 50k validation images. We extend the residual networks [1] that are originally designed to learn in ILSVRC 2012 with parallel paths and routing. Similar conditions exist for each of these networks and the thin residual networks created for CIFAR learning. The ILSVRC ResNets begin with a 7×7 convolution with a stride of 2, then a



(a) CIFAR10



(b) CIFAR100

Figure 3.6: ResNet variant performance (accuracy) in CIFAR with the number of parameters (millions). Blue circles show conventional ResNets and ResNet-based adaptive networks. Green circles show multi-path ResNets with cross-prediction-based routing, while red circles correspond to multi-path ResNets with cross-connection-based routing. Our networks flocking to the top-left of the charts prove they show the best performance w.r.t the network complexity.

max-pooling operation. Then, four sequential stacks of residual blocks are used, each stack containing a certain number of residual blocks with the same feature map size. The first residual block of each stack begins with a stridden convolution that downsamples the feature maps by a factor of two. The response from the final residual block is input into a global average pooling block, followed by the final fully connected layer to output the class response.

We exclusively employ cross-connection-based routing when extending these models to parallel paths since it is less complex, consumes very little overhead, and still produces reasonably comparable results to cross-prediction-based routing. After the first convolution and max-pooling layer, we add a one-to-many adaptive router, which broadens the network to parallel pathways. We then add cross-connection-based routing layers following each stack of residual blocks with the same feature map size. The next step is to average the dense predictions from the final layer.

As explained above, we extend ResNet18 with two parallel paths and cross-connection-based routing, resulting in ResNet18-2-CC. We train ResNet18-2-CC in ILSVRC 2012 for 120 epochs with a batch size of 256. We employ an SGD optimizer with a momentum of 0.9. The initial learning rate of 0.1 decays by a factor of 10 every 30 epochs. We employ common data augmentation for ILSVRC 2012, rescaling the data to 256×256, selecting arbitrary 224×224 crop sizes, and randomly inverting the horizontal axis. We use a portion of the ILSVRC 2012 dataset that only includes the first 100 classes to further assess deeper models with parallel paths. There are 130k training images and 5k validation images in this subset. Finally, we extend ResNet50 with two paths and cross-connection-based routing and train with this subset (ResNet50-2-CC). Except for training the models for 90 epochs, our training setup is the same as it is for the full dataset. We also train ResNet50, WideResNet50-2, ResNeXt50-2-64, and ResNet101 to compare with ResNet50-2-CC. WideResNet50-2 contains two times filters in each layer, and ResNeXt50-2-64 contains two parallel operations in each layer.

In Table 3.4, we show the results of ILSVRC 2012 experiments. ResNet18-2-CC easily outperforms ResNet18 and shows on-par performance with ResNet34. It also surpasses WRN-18-1.5, the WideResNet18 with 1.5 times convolutional filters in each layer that still has more parameters. In the 100-class subset, ResNet50-2-CC outperforms its single path baseline (ResNet50), as well as WideResNet50-2 and ResNeXt50-2-64, demonstrating the superiority of our method over other widening at comparable complexity. ResNet50-2-CC even displays marginally superior results to ResNet101, which is twice as deep.

Overall, these tests confirm that our multi-path networks and adaptive routing algorithms effectively use the resources in each layer. Thus, our multi-path networks outperform conventional widening, other methods for rich layer-wise feature extraction, and even conventional deepening at an equivalent or lower complexity.

Table 3.4: ILSVRC 2012 Dataset: Single-crop and 10-crop validation error (%). ResNet18-2-CC comfortably outperforms ResNet18 and show on-par performance to ResNet34. It also surpasses WRN-18-1.5, which has 1.5 times filters. In the 100-class subset of ILSVRC2012, ResNet50-2-CC outperforms WideResNet, ResNext counterparts, and even twice deep ResNet101. * denotes reproduced results

Network	Params	Single-Crop		10-Crop	
		Top-1	Top-5	Top-1	Top-5
Full Dataset					
ResNet18 [6,86]	11.7M	30.4	10.93	28.22	9.42
ResNet34 [1,6]	21.8M	26.77	8.77	24.52	7.46
WRN-18-1.5 [6]	25.9M	27.06	9.0		
ResNet18-2-CC	23.4M	**26.48**	**8.6**	**24.5**	**7.34**
Subset of first 100 classes					
ResNet50*	23.71M	20.46	4.96	19.26	4.72
ResNet101*	42.7M	19.16	4.58	17.78	4.44
WideResNet50-2* [6]	62.0M	19.82	5.02	18.62	4.76
ResNeXt50-2-64* [7]	47.5M	20.26	5.06	19.0	4.84
ResNet50-2-CC	47.5M	**18.64**	**4.34**	**17.62**	**4.0**

3.5 Visualizations

In this section, we analyze the cross-connection-based routing scheme's gating patterns using a variety of visualizations. We employ a VGG13 [4] network with 256 nodes per dense layer and half as many convolutional filters (32, 64, 128, 256) in each layer. As with the multi-path networks in Section 3.2, we connect two of these networks using cross-connections, adding the routing layers after each pooling operation and the first dense layer. We train this network with the 100-class ILSVRC 2012 subset.

We begin by displaying the variations of routing patterns of the trained network at different depths. We further analyze the context of input which maximizes each gate. We present the images from the validation datasets that activate specific gating neurons the most. We also synthesize randomly initialized images that maximize those neurons. To comprehend the class-wise gate activation, we plot the gate activations of a few chosen classes. Finally, we show that each parallel path can learn different information by plots of weight histograms of the two parallel paths at selected layers.

3.5.1 Routing Visualization

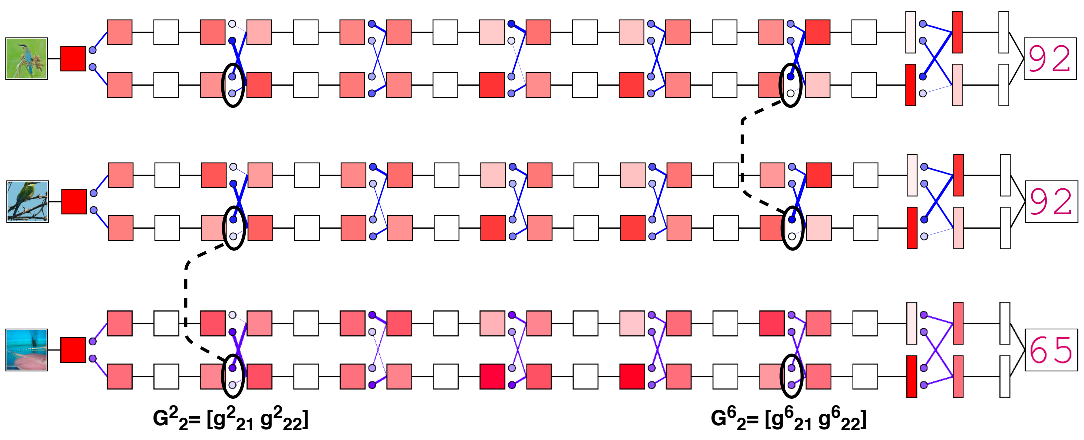

92
92
65
$G^2_2 = [g^2_{21}\ g^2_{22}]$
$G^6_2 = [g^6_{21}\ g^6_{22}]$

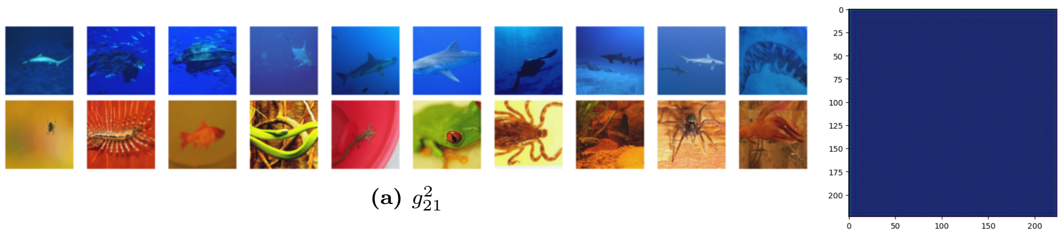
Figure 3.7: Route visualizations of VGG13-2-CC for the three images in Figure 1.1. The routing diagrams, top to bottom, correspond to Image 1.1a (Hummingbird in green background), Image 1.1b (Hummingbird in blue background), and Image 1.1c (Electric ray in water), respectively. In each routing layer, the gate strengths are indicated in connection thicknesses and blue intensities, and the relative strengths of the input and output tensors are displayed in red intensities. The gating vector G_2^2, in shallow layers, displays identical gating patterns for images 1.1b and 1.1c although they belong to two different classes. In contrast, gating vector G_2^6 that lies in deeper layers, displays similar gating patterns for the same class images 1.1a and 1.1b. The routing layers situated in different network depths allocate their inputs based on the level of the context (features) represented at that depth.

To comprehend the gating patterns, we display the routing flow through cross-connections of the trained 2-path network. Figure 3.7 displays such routing flows for the three images in Figure 1.1. Each routing layer has two parallel inputs, two parallel outputs, and gates that weigh the cross-connections. We show each routing layer's input and output tensors' relative activation and gate strengths. A tensor's relative activation strength is its average activation value normalized by all such values of the parallel tensors in that layer. We color each box representing a specific tensor with red intensities corresponding to these relative activation strengths. Each input's computed softmax gate values are directly transferred to blue intensities and thickness values, then used to color the edges and circles representing each gate, respectively. We use plain-colored boxes to represent the stacks of traditional forward layers. They do not do cross-computations but do contain sequential convolutions or dense operations that occur in parallel.

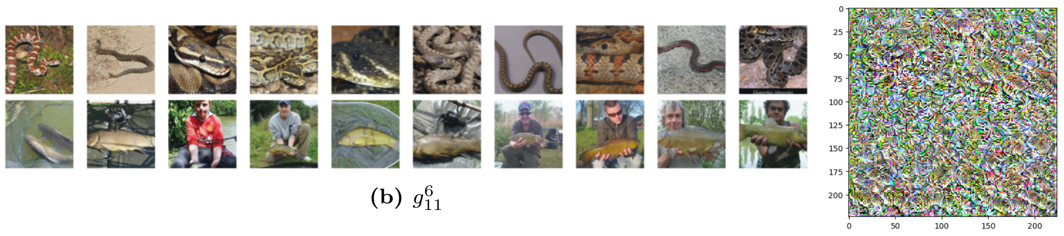
Let G_i^l ($[g_{i1}^l, g_{i2}^l]$) be the gating vector calculated from the l^{th} cross-connecting layer's i^{th} input tensor. We focus on the gating vectors G_2^2 ($[g_{21}^2, g_{22}^2]$) and G_2^6 ($[g_{21}^6, g_{22}^6]$) in these routing diagrams. Although they are from distinct classes, images 1.1b and 1.1c share comparable gating patterns with gating vector G_2^2 (maximized g_{21}^2), located in the network's first layers. Even though both images 1.1a and 1.1b are of hummingbirds, G_2^2 displays different gating patterns for them. However, gating vector G_2^6, which lies within a deeper layer, shows similar gating patterns to the two hummingbird images (maximized g_{21}^6) and noticeably different gating pattern for the electric eel. This behavior illustrates that the gating in a routing layer at a particular network depth is sensitive to the context/features represented at that depth. Also, it highlights the importance of having routing layers throughout the depth of the network since, at different depths, different mixes of parallel resource allocation are suitable. We further investigate which features maximize each gate next to comprehend the underlying causes of this behavior better.

3.5.2 Gate Maximization Patterns

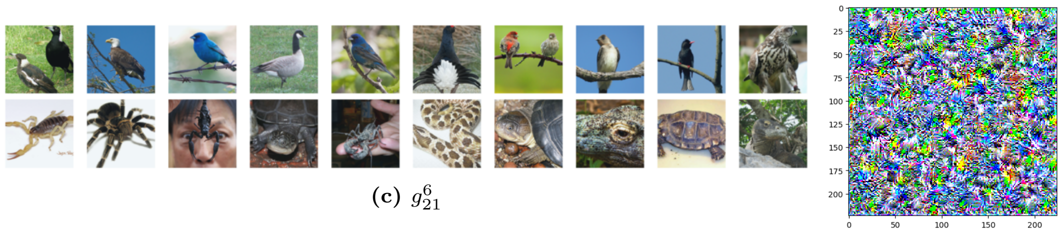
We show the images in the validation set that maximize and minimize each gating neuron to comprehend the features (context) that maximize each gate and, consequently, to describe the gating patterns mentioned above. We also synthesize the network input while freezing the trained network to maximize the specific gating neuron (prior to softmax activation). This is similar to the gradient ascent method that [87] introduced. For this visualization, we use four gating vectors, G_2^2, G_1^6, G_2^6 and G_1^7, where G_2^2 and G_2^6 are the gate vectors described in the previous visualization. Due to the softmax activation, one of a pair of gating neurons is inversely connected to the other; thus, increasing one gate has



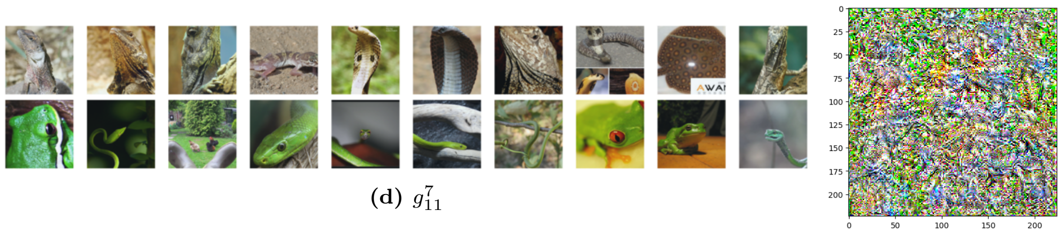
(a) g^2_{21}



(b) g^6_{11}



(c) g^6_{21}



(d) g^7_{11}

Figure 3.8: Features that maximize gates. Each sub-figure corresponds to a specific gate and shows the ten images that maximally activated the gate at the top, the ten images that minimally activated the gate at the bottom, and the synthesized image that maximizes the gate at the right. g^2_{21}, in shallow layers, maximizes for blue. Other gates which lie in deeper layers maximize for more abstract attributes: g^6_{11} for snake body patterns, g^6_{21} for bird patterns, and g^7_{11} for raised upper body patterns.

the opposite effect as reducing the other. Therefore, from each pair of gating neurons, we select just one for visualization—correspondingly, gating neurons g^2_{21}, g^6_{11}, g^6_{21} and g^7_{11} from each gating vector.

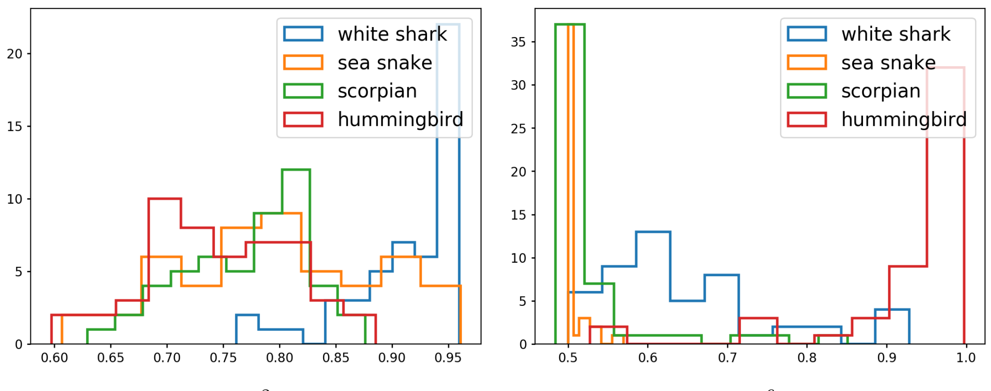
The results of this visualization are shown in Figure 3.8. The top left of each subfigure shows the 10 photos that produce the highest gate activation, while the bottom left shows the 10 images that give the lowest activation. The synthesized image that maximizes the gate neuron is presented to the right. Gate g^2_{21} (Fig 3.8a), which lies within the initial layers, maximizes for the blue color, which is a low-level detail. However, all other gates in the figure that lie within deeper layers maximize for complex patterns. For example, gate g^6_{11} maximizes for snake patterns, gate g^6_{21} maximizes for bird patterns, and gate g^7_{11} maximizes for animal poses with raised thorax. Each synthesized image displays patterns that maximize the related gate, and they both agree on the top 10 active images.

We interpret the gating patterns in Section 3.5.1 Based on the maximization patterns of g^2_{21} and g^6_{21}. Since the gate g^2_{21} maximizes for the blue color, it shows high activation for 1.1b and image 1.1c, which consist of backgrounds highly composed of blue. Meanwhile, it shows a lower activation to image 1.1a despite being in the same class as image 1.1b. Gate g^6_{21}, which lies deeper, maximizes for bird patterns. The two hummingbird images (image 1.1a and image 1.1b) highly activates this gate, whereas the gate shows a lower activation for the electric eel (image 1.1c). This behavior demonstrates how the task-related visual context is dispersed along with the trained network's depth. It is crucial to have routing layers within the network for each segment of layers because resource allocation in different stages of depth differs from one another depending on the level of context represented in that depth.

3.5.3 Class-Wise Gating Patterns

Our multi-path networks' resource distribution in each layer is based on the type of feature maps present at that depth. We plot the gate response of a few chosen classes for gates g^2_{21} and g^6_{21} to look into any potential influences of the class on gating patterns. For this, we select four classes in the ILSVRC 2012: scorpion, hummingbird, sea snake, and white shark. We record the gate responses for each class's images and summarize the gate activation histograms for the four classes Figure 3.5.3.

The class white sharks exhibits overall high g^2_{21} activation, with blue sea water being the predominant detail in most instances. However, the distribution of g^2_{21}



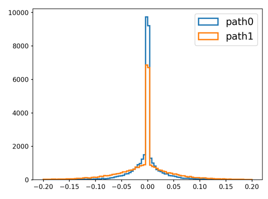
(a) g_{21}^{2} **(b)** g_{21}^{6}

Figure 3.9: ILSVRC 2012 validation set gate activation histograms for four selected classes. While other classes are similarly distributed, the white shark, with primarily blue seawater as the background, has a high activation for g_{21}^{2} overall. While other classes, which agree less with bird patterns, exhibit less g_{21}^{6} activation, hummingbird images often show a high activation. A class's members receive similar gating in that layer if the triggering pattern of any gate is frequently observed in that class.

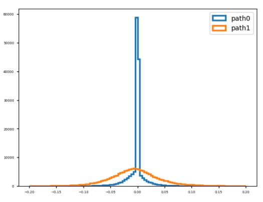
is evenly distributed among the other classes since those classes contain instances that may or may not contain dominating blue. Furthermore, class hummingbird, which exhibits bird poses and patterns, exhibits overall high activation for g_{21}^{6}, which activates for bird patterns. However, since they rarely agree on bird patterns, the other classes exhibit low g_{21}^{6} activations. The context of features, which is vital for the resource allocation in each layer, is a complicated detail that goes beyond merely the class. However, because of the distinctiveness of some classes, the majority of their members could have identical gating patterns in specific layers. This occurs when the class members make up most of the gate's triggering pattern.

3.5.4 Parallel Computation Weights

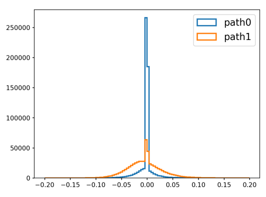
Grouping homogeneous feature maps into parallel paths and allowing the parallel filter sets of the same layer to learn various informational subsets is one of the goals of routing in multi-path networks. We present the weights histograms of the VGG13-2 selected layers with two parallel convolutions or dense operations on the two separate sets of feature maps to see whether our strategy was success-



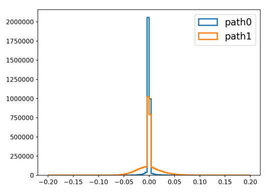
(a) layer 4



(b) layer 6



(c) layer 8



(d) layer 11

Figure 3.10: Parallel operations' weight histograms at particular layers. The two weight histograms in each sub-figure correspond to the two parallel paths at the particular layer. Histogram variations within the same layer demonstrate that distinct information is learned along parallel paths.

ful. The weight histograms for the two parallel processes at layers 4, 6, 8, and 11 are displayed in Figure 3.10. While the other layers are convolutional, layer ll is dense. The parallel pathways' discrete histograms demonstrate that they have acquired distinct information.

3.6 Conclusion

In this chapter, we introduced techniques for combining multiple parallel paths into a single network and intelligently routing data that depend on the input. Our approach consistently achieved better classification accuracy than existing widening methods with similar complexity. Our networks also outperformed existing adaptive learning strategies and even performed slightly better than thinner, deeper networks with a similar number of parameters. Through experimentation, we demonstrated the effectiveness of our routing mechanisms in handling input dependency and extracting unique features from parallel paths. Our multi-path networks can be viewed as a single adaptive model that smoothly switches between different sub-modules depending on the input.

Towards regularizing gating:

Our multi-path networks, including the sub-gating networks, learn from the end objective; in our experiments, classification. In the current design, the L2 regularization of weights and the global average pooling of input tensor function as the regularization methods for the gating network. However, there may be instances where a particular gating network learns to mostly activate one gate and leave other gates deactivated for all samples. E.g., a two-gate sub-gating network may learn to activate only one gate most of the time, leaving the other gate less activated for all samples.

The gate activations are conditioned on the input feature map set. A particular gating network takes one tensor among the parallel as input and computes the probabilities of that tensor being routed to each of the next layer parallel tensors. The decision of which path to forward the current tensor is based on the features of the current tensor. Thus, it is worthwhile to emphasize motivating the gating network to learn clustering probabilities in addition to learning from the end objective. Therefore, we explore gating regularization as a clustering objective. In the following section, we introduce neural mixture models, in which a neural network output nodes are trained to cluster its input into a number of clusters equal to the number of output nodes.

Chapter 4

NEURAL MIXTURE MODELS FOR CLUSTERING

4.1 Introduction

The gating networks introduced in the previous section govern resource allocation between parallel processes in a multi-path layer. While those gating sub-networks learn from the end objective, we focus on further regularizing them as there may be some gating vectors that mostly activate in a specific direction. To this end, we focus on using a clustering objective directly on the gating neurons, which enforces soft-cluster input feature maps and activates gates accordingly. We propose neural-based mixture modeling, which enforces a neural network to learn cluster distributions and posterior cluster assignments concurrently. This chapter proposes and evaluates our neural-based mixture modeling as a standalone clustering task.

Clustering groups together the data points with similar characteristics. Since the labels are absent, clustering algorithms must learn both cluster representations and member assignment. One way to accomplish this dual-objective learning is by iteratively switching between the two phases, following the Expectation-Maximization (EM) [88] algorithm: 1) assigning samples to clusters according to the degree of match to the cluster representations and 2) updating the current cluster representations based on the member assignments.

Mixture-model-based clustering follows this iterative optimization. They use distributions such as Gaussian and Bernoulli with pre-defined parameters (a prob-

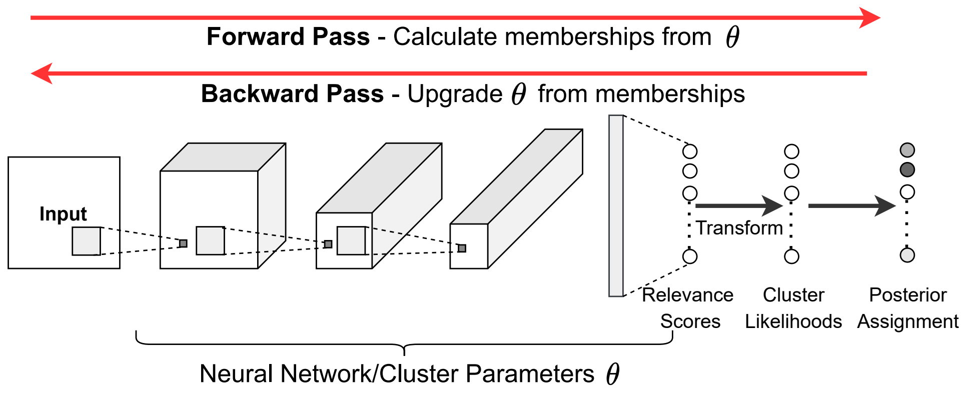

Forward Pass - Calculate memberships from θ
Backward Pass - Upgrade θ from memberships
Input
Transform
Relevance Scores
Cluster Likelihoods
Posterior Assignment
Neural Network/Cluster Parameters θ

Figure 4.1: Overview of mixture-EM formulation with a neural network : The cluster parameters are represented by θ, the network parameters. When the network produces cluster relevance scores during the forward pass, they are converted to cluster distribution outputs (likelihoods). The posterior membership probabilities are then computed using these distribution outputs. The computed memberships are used in the backward pass to update the parameters, θ.

ability, or mean and covariance) as cluster representations. The observed data points are fitted with a mixture of these distributions such that the total likelihood is maximized. The distribution parameters and posterior member assignments are iteratively updated, retaining one fixed at a time, following the EM algorithm.

We follow mixture modeling with EM and use a neural network to model cluster distributions and posterior member assignments. The parameter updates follow the EM algorithm batch-wise via backpropagation. Since a neural network can approximate complex functions [18], we can approximate complex cluster representations better than limited-parameter hand-designed distributions in the EM framework. Moreover, the network-learned distributions are not pre-selected and are adaptive to the data space. Also, such a mixture-EM objective can easily be plugged into a gating sub-network within a main network as a regularization loss.

First, each cluster distribution is approximated by a parametric function which is the neural network from its first layer to the respective final layer neuron, followed by an extra transformation. This additional transformation ensures that the altered final layer neuron behaves as a probability density function of the input, thus, preventing one cluster distribution from dominating other distributions to capture all datapoints. This transformation uses sigmoid activation on the final layer neurons that are normalized over the batch and further normalized by a constant (a temperature parameter) so that they have a batch mean of zero and fall within the most linear region of the sigmoid function. As demonstrated later, restricting the final-layer neurons to have a batch mean of zero and to fall within the most linear portion of the sigmoid causes all cluster distributions to possess the same integral over the sample space. Consequently, the approximated distributions serve as probability density functions of input, avoiding trivial solutions.

Second, we present a batch-based EM optimization for end-to-end network training. Using the estimated cluster likelihoods, we compute the posterior probabilities of cluster assignments. These posteriors are estimated by taking softmax of normalized final-layer outputs. Using the estimated posteriors and cluster likelihoods, we derive the EM loss for backpropagation. The optimization of the network consists of online batch-wise EM iterations. For each iteration, we input the network a batch of data. The forward pass corresponds to the E-step, which calculates the cluster likelihoods and posterior probabilities for the given batch from the network output, and derives the EM loss. The backward pass of the network corresponds to the M-step, which performs a gradient step toward optimizing the EM loss. Figure 4.1 depicts an overview of the suggested neural network-based mixture-EM optimization.

Finally, for image clustering, we combine EM optimization with consistency optimization between the original and transformed versions of data. It is vital to force the neural network to acquire generic semantically significant features and prevent itself from overfitting to the low-level features in images. Thus, we combine learning to generate a consistent response to the original and its augmented versions [89–91] with the mixture-EM optimization. Our consistency optimization minimizes the Kullback–Leibler (KL) divergence [92] between the responses of the model to the original images and the augmented images. This optimization is incorporated into the EM procedure, resulting in a two-step optimization. We demonstrate that this dual-optimization yields a more rapid and better convergence than only consistency optimization.

The proposed optimization conducts EM iterations in batches online, thereby eliminating the requirement to iterate over the whole data space for a single EM update. A neural network allows complicated cluster representations to be learned instead of limiting to manually-selected distributions. The transition of the last layer neurons to regulate the modeled cluster distributions as probability density functions avoids the model from falling to trivial solutions, hence removing the need for extra losses to offset this tendency. The implementation is simple, and the loss function may be effortlessly plugged in as a regularization loss for sub-gating networks.

4.2 Related Work

Clustering a data set typically involves learning cluster representations, which are the basis for grouping the data points into those clusters. Classes of techniques to discover cluster representations include centroid-based [93], connectivity-based [94], distribution-based [95], subspace-based [96], and density modeling-based [97] methods. Mixture modeling is a distribution-based clustering model in which a statistical distribution describes each cluster. GMM [98] models each cluster by a Gaussian distribution and iteratively improves the cluster distributions and assignments using the EM algorithm [88]. K-means [93] can be viewed as a specific instance of GMM in which the clusters are modeled as untilted spheres. In addition, there have been efforts to model sophisticated clusters using neural networks, as traditional clustering techniques' cluster representability is limited to a few hand-designed parameters [90, 99–103].

Notably, numerours deep image clustering algorithms still use k-means [100, 104–107]. These works either employ k-means in a learned latent space (abstract embedding space with lower number of dimensions than the inputs, but can represent the inputs meaningfully) [100, 104, 106] or synthesize k-means using neural networks [105]. The mapping from input space to latent space is learned via a self-supervised representation learning [100, 104, 106]. k-means is applied to either cluster the latent space [104] or to produce pseudo-labels for network training [100].

Although synthesizing k-means with a neural network has been explored [105], implementing mixture modeling with a neural network has been studied only infrequently, to the best of our knowledge. Neural Expectation-Maximization (N-EM) [108] presents differentiable EM-based clustering. The goal of N-EM is to learn the perceptual grouping of a given input by distinguishing the various conceptual entities in the input. N-EM employs a neural network to estimate the statistical parameters of a defined cluster distribution given object vectors (e.g., a probability if Bernoulli, mean, and variance if Gaussian). The cluster distribution outputs are computed using these statistical parameters. Then, the network parameters and the object vector are updated via backpropagation. Instead of using a neural network to predict the parameters of a human-defined cluster distribution, we use a neural network to estimate the cluster distribution function along with its parameters for different clusters.

A successful deep image clustering approach should acquire generic semantic information necessary for identifying abstract groups. Thus, allowing a neural network to extract rich semantic features while learning the clustering is essential. Typically, a self-supervised representation learning task is used to acquire a robust semantic representation of images that serves as a solid foundation for clustering [90, 91, 99, 101–104, 106, 107, 109, 110]. The feature representation is either learned prior to clustering and clustering is performed in the learned latent space [91, 104, 106, 107, 109, 110], or is simultaneously learned alongside clustering [90, 99, 101, 102]. In our case, we use consistency optimization as our feature learning task for image clustering and fuse it with mixture-EM clustering optimization to optimize both the objectives side-by-side.

The proposed mixture modeling employs a neural network as the cluster distribution estimator and its parameters as cluster distribution parameters. The neural mixture-EM optimization is a superior alternative to k-means and other conventional clustering algorithms. In addition, our clustering objective can be used in conjunction with other feature learning techniques, such as consistency optimization and pre-text task learning, and self-labeling fine-tuning approaches, to construct quite sophisticated end-to-end multi-stage clustering solutions. However, we only uncover the combination of consistency and mixture-EM optimization for image clustering, resulting in a two-fold single-stage training procedure.

4.3 EM Algorithm in Mixture Modeling

Expectation-Maximization (EM) [88] is an iterative algorithm often used where direct objective functions are difficult to optimize. Most commonly, EM is used to approximate solutions to the maximum likelihood estimate (MLE) and maximum a posterior estimation (MAP). Mixture modeling fits a collection of distributions to the sample space by maximizing the total likelihood of the composite distribution. Due to the difficulty of maximizing this likelihood, EM is commonly utilized to maximize an alternate lower bound. This section briefly discusses how EM can approximate MLE when fitting a mixture of distributions to a given dataset.

Let x represent an observation of the continuous random variable X in space $\mathcal{D}$, and let D represent a set comprising N such observations that are sampled from $\mathcal{D}$ ($D \subset \mathcal{D}$). Let θ denote the cluster-defining parameters. The space $\mathcal{D}$ must be clustered into K clusters. We create a discrete latent variable Z that can be viewed as the cluster assignment ($z \in [1, \ldots, K]$). The purpose of MLE is to discover θ such that the total likelihood marginalized over Z is maximized:

$$\mathcal{L}(\theta; D) = \prod_{x \in D} \sum_{z \in [1,\dots,K]} f(x, z \mid \theta). \tag{4.1}$$

$f(x, z \mid \theta)$ is the joint probability density of x and z for the parameters θ. It represents the likelihood of θ for the observed x and z: $l(\theta; x, z)$, in the likelihood function. Note that f stands for a continuous probability density while p stands for a discrete probability.

MLE finds optimum θ which maximizes this likelihood. Due to the difficulties in optimizing this product, the logarithm is often taken, leading to the log-likelihood;

$$\log \mathcal{L}(\theta; D) = \sum_{x \in D} \log \sum_{z \in [1\dots K]} f(x, z \mid \theta). \tag{4.2}$$

However, due to the logarithm of the summation, which arrives from the marginalization over Z, it is difficult to optimize this objective function. Therefore, the EM procedure optimizes a lower bound instead;

$$Q(\theta) = \sum_{x \in D} \sum_{z \in [1\dots K]} p(z \mid x, \theta) \log[f(x, z \mid \theta)]. \tag{4.3}$$

Here, $p(z \mid x, \theta)$ is the posterior probability of $Z = z$ given x.

The EM Algorithm switches between the E-step and the M-step while optimizing Equation 4.3. At the E-step in the t^{th} iteration, it derives the posteriors for the current parameters θ^t: $p(z \mid x, \theta^t)$ and computes the EM objective.

$$Q(\theta \mid \theta^t) = \sum_{x \in D} \sum_{z \in [1\dots K]} p(z \mid x, \theta^t) \log[f(x, z \mid \theta)], \tag{4.4}$$

The M-step optimizes this objective w.r.t. θ,

$$\theta^{t+1} = \arg\max_{\theta} Q(\theta \mid \theta^t). \tag{4.5}$$

4.4 Formulating Mixture-EM on a Neural Network

This section realizes the mixture modeling with the EM algorithm on a neural network to cluster a dataset end-to-end. Rather than maintaining a set of distributions from a hand-designed function (e.g., Gaussian) and optimizing their statistical parameters (e.g., means and covariances of each cluster Gaussian), we let the neural network freely model the cluster distributions along with their parameters for each cluster. The neural network parameters in all layers but the final layer of the network decide the shared representations of the clusters (e.g., abstract shape), while the parameters in the final fully connected layer learn the cluster-specific representations (e.g., dimensions of each cluster).

It is vital for all cluster distributions to act as probability density functions (PDFs) of the input. The distribution outputs should be continuous, positive, and share the same integral throughout the sample space. If all distributions do not share the same integral, a single cluster distribution could rapidly grow to contain all datapoints and dominate the other distributions. This results in the trivial allocation of all datapoints to a single cluster. We ensure the PDF behavior of the modeled distributions by normalizing the last layer neurons (each neuron corresponds to a single cluster) over the batch and constraining their sigmoid outputs to the most linear portion of the sigmoid. We show that such transformation enables the approximated cluster distributions to act as PDFs. The posterior class assignments are calculated using the softmax of the normalized final layer neurons. We then use the posteriors and the distribution outputs to calculate the EM objective in Eq. 4.4.

We carry out online EM optimization using batch-wise backpropagation. At one iteration, we feed the neural network a batch of observations (samples from input space) and conduct one EM iteration, corresponding to calculating and backpropagating the EM-based loss. In contrast to conventional EM, where the current EM loss is fully optimized in the M-step (Eq. 4.5), we take a single gradient descent step in optimizing the EM loss for the current batch. We feed the next batch of observations at the next iteration and repeat EM. These batch-wise online EM iterations avoid the inefficient collection of all observations for a single EM iteration.

Let x denote a sample in the continuous space X (an observation). Our task is to cluster this space to K number of clusters. We use K output nodes of the neural network to estimate K cluster distributions. Let g be a parametric function, a neural network that learns the cluster representations and member assignments. Then, the cluster parameters θ are reflected by the neural network parameters. The network's final layer's K nodes represents the relevance of input to the respective clusters. Given a batch of n observations $x_{i,i=1,\ldots,n}$, the network's final layer outputs K relevance scores a_i for each x_i:

$$a_i = [a_{i1}, \ldots, a_{iK}] = g_\theta(x_i) \tag{4.6}$$

Here, a_{ij} stands for the relevance of observation x_i to cluster j (j = 1,...,k). a_{ij} increases with the degree of membership of x_i to cluster j. We illustrate this relevance as the output of two composite functions g_{θ^s} and $g_{\theta^e_j}$:

$$a_{ij} = g_{\theta^s,\theta^e_j}(x_i) = g_{\theta^e_j}(g_{\theta^s}(x_i)). \tag{4.7}$$

Say, the neural network has L layers. g_{θ^s} represents the portion of the network from input to layer $L-1$. All clusters share the parameters of this network portion θ^s. The mapping from layer $L-1$ to the final layer neuron j is represented by $g_{\theta^e_j}$. The parameters θ^e_j here are exclusive for the j^{th} cluster.

4.4.1 Approximate Cluster Distributions

Here, we show how to estimate each cluster's distribution function h_θ, i.e., the probability density of a certain observation x given the cluster assignment z and its parameters θ: $f(x \mid z, \theta)$. If $h_{\theta^s,\theta^e_j}(x)$ is the cluster j distribution function,

$$f(x \mid Z = j, \theta) = h_{\theta^s,\theta^e_j}(x). \tag{4.8}$$

$h_{\theta^s,\theta^e_j}(x)$ is fundamentally a PDF of x and we intend to transform each batch's relevance score $g_{\theta^s,\theta^e_j}(x)$ to obtain $h_{\theta^s,\theta^e_j}(x)$. Thus, we first describe the necessary characteristics of $h_{\theta^s,\theta^e_j}(x)$:

1. $h_{\theta^s,\theta^e_j}(x)$ must be a continuous function x.
2. $h_{\theta^s,\theta^e_j}(x)$ must be positive for all x.
3. the integral of h_{θ^s,θ^e_j} over the space $\mathcal{D}$ should be 1.

Nonetheless, the integral of h_{θ^s,θ^e_j} being a constant shared by all clusters instead of being 1 still yields their behavior as PDFs because such a constant is irrelevant in the optimization process. This restriction of integral controls the cluster shapes and avoids only a few clusters dominating others (trivial solutions).

Although the output of the last layer j^{th} neuron: $a_{ij} = g_{\theta^s,\theta^e_j}(x_i)$ is linked to the degree of x_i's membership in cluster j, it is not yet suitable for approximating the cluster j distribution. Consequently, we turn this relevance into a form capable of representing the cluster distribution via a second transformation, H:

$$h_{\theta^s,\theta^e_j}(x) = H(g_{\theta^s,\theta^e_j}(x)). \tag{4.9}$$

This transformation is derived step-by-step to satisfy the criteria above. Initially, $g_{\theta^s,\theta^e_j}(x_i)$ (or a_{ij}) is a continuous function of x_i. However, it is the output of a final layer neuron prior to any activation; therefore, it can be negative. Simply taking the exponential of a_{ij} will transform it into a positive value. However, due to the exponentially growing nature of this function and having no fixed upper bound, we use a normalized form of the exponential, the sigmoid activation instead,

$$\text{sigmoid}(a_{ij}) = \frac{1}{1 + \exp(-a_{ij})}. \tag{4.10}$$

The final requirement is not yet met, as the integral of sigmoid(a_{ij}) over $x \in \mathcal{D}$ is not a constant value shared by all clusters. We enable this attribute by setting a constraint on the integrals of all cluster distributions in space $\mathcal{D}$. First, we normalize the sigmoid's input, which is the relevance score a_{ij}, across all such scores to cluster j in the batch.

$$a^*_{ij} = \frac{a_{ij} - \mu_j}{\sigma_j}, \quad \text{where} \quad \mu_j = \frac{1}{n}\sum_{i=1}^{n} a_{ij} \quad \text{and} \quad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(a_{ij} - \mu_j)^2. \tag{4.11}$$

a^*_{ij} denotes the normalized relevance of x_i to the j^{th} cluster. μ_j and σ_j are the batch's mean and standard deviation of a_{ij}. We again divide a^*_{ij} by an additional constant γ ($\gamma > 1$), that plays the role of a temperature parameter. Therefore, the cluster j distribution function that meets all the criteria is,

$$h_{\theta^s, \theta^e_j}(x_i) = \text{sigmoid}(g^*_{\theta^s, \theta^e_j}(x_i)/\gamma) = \text{sigmoid}(a^*_{ij}/\gamma). \tag{4.12}$$

The rationale for normalizing a_{ij} and dividing by γ are as follows. According to Figure 4.2, if we assume that the sigmoid's input is a standard normal variable (mean 0, variance 1), the sigmoid input is constrained to a small interval around zero, and distant values are rare (99.9% confidence interval of standard normal variable is $[-3.29, 3.29]$). Even in the absence of this assumption, when we normalize a set of n points by their own sample statistics, the attainable upper bound for the maximum Z score is proven to be $\frac{(n-1)}{\sqrt{n}}$ by Shiffler 1988 [111]. Consequently, $\frac{(n-1)}{\sqrt{n}}$ be the upper bound for the most deviated Z score. Similarly, the Z score cannot go beyond $-\frac{(n-1)}{\sqrt{n}}$ (This bound is derived from the edge case of $n-1$ samples being the same value and the n^{th} sample being distinct). Thus, we can conclude that the normalized points are bounded to the interval

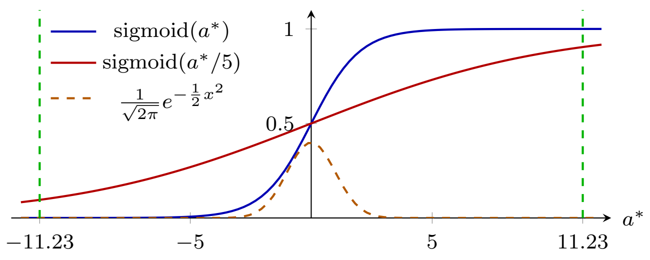


Figure 4.2: A standard normal variable is constrained to a narrow interval around 0 with high certainty. Moreover, the sigmoid function input a^* is bound to [-11.23,11.23] (green vertical lines) as a^* is normalized with a batch size of 128. We divide a^* by $\gamma = 5$ to maintain the sigmoid output for this interval in sigmoid's predominantly linear region. With sigmoid activation's zero mean linear region input, the overall function from input x to sigmoid output behaves as a PDF of x.

$\left[\frac{-(n-1)}{\sqrt{n}}, \frac{(n-1)}{\sqrt{n}}\right]$. For instance, if we normalize a_{ij} over 128 observations, the normalized a^*_{ij} is bounded to interval $[-11.23, 11.23]$. Moreover, we divide a^*_{ij} by γ ($\gamma > 1$) to maintain the sigmoid output within its predominantly linear zone for all bounded a^*_{ij}s. γ is linked to the batch size. For batch size 128, γ being 5 sufficiently ensures that the sigmoid output is predominantly linear for all a^*_{ij}, as shown in Figure 4.2.

Normalizing a_{ij} across the batch and subsequent division by γ imposes two significant constraints on the cluster distribution $h_{\theta^s,\theta^e_j}(x_i)$, as outlined below:

1. sigmoid activations's input: a^*_{ij}/γ or $g^*_{\theta^s,\theta^e_j}(x_i)/\gamma$, is zero on average over the batch:
$$\frac{1}{n}\sum_{i=1}^{n} a^*_{ij}/\gamma = 0. \tag{4.13}$$

2. For any x_i, sigmoid(a^*_{ij}/γ) restricts to sigmoid's predominantly linear region.

Due to these two conditions, we can estimate the batch average of sigmoid(a^*_{ij}/γ) by 0.5:
$$\frac{1}{n}\sum_{i=1}^{n} h_{\theta^s,\theta^e_j}(x_i) = \frac{1}{n}\sum_{i=1}^{n} \text{sigmoid}(a^*_{ij}/\gamma) = 0.5. \tag{4.14}$$

Using Monte Carlo integral estimate [112], we can verify that the j^{th} cluster distribution integral over $x \in \mathcal{D}$ is a fixed value shared by all clusters. Let's define I_j as the j^{th} cluster distribution integral over the space $\mathcal{D}$:

$$I_j = \int_{\mathcal{D}} h_{\theta^s,\theta^e_j}(x)\, dx. \tag{4.15}$$

Here, x is an m-dimensional observation from space $\mathcal{D}$ ($\mathcal{D} \subset \mathrm{R}^m$). We approximate the integral I_j by uniformly sampling n observations $x_{i|i=1,\ldots,n}$ from space $\mathcal{D}$ and using the Monte Carlo approach:

$$I_j \approx V \frac{1}{n} \sum_{i=1}^{n} h_{\theta^s,\theta^e_j}(x_i). \tag{4.16}$$

V denotes the volume of m-dimensional space $\mathcal{D}$: $V = \int_{\mathcal{D}} dx$. According to Eq. 4.14, the mean distribution output across n samples is 0.5. Consequently, the Monte Carlo estimate for the j^{th} cluster distribution integral across space $\mathcal{D}$ is,

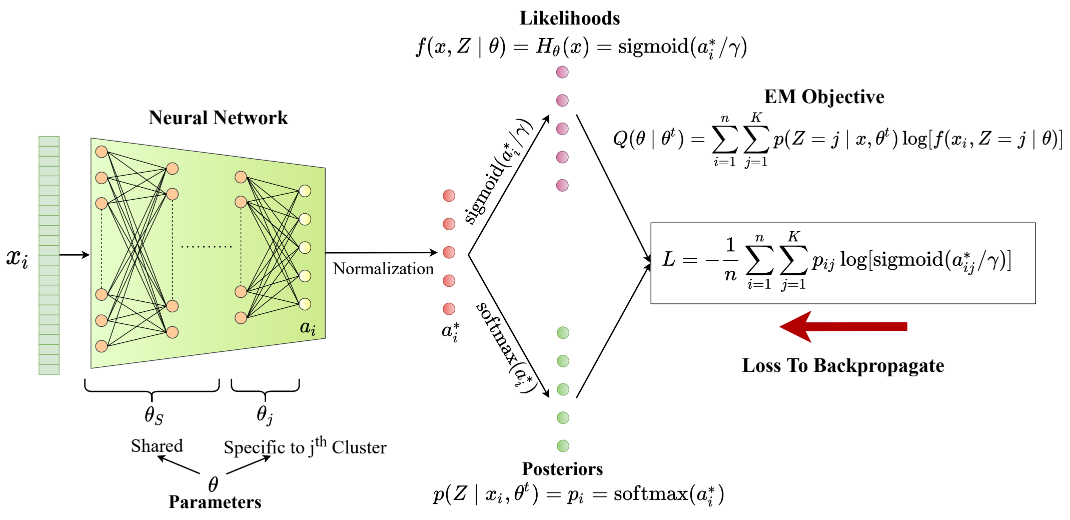
$$I_j \approx 0.5V. \tag{4.17}$$

All cluster distributions $h_{\theta^s,\theta^e_j}(x)|_{j=1,\ldots,K}$ share the same integral approximation as cluster j ($0.5V$). Thus, all distributions behave as PDFs with the same integral over space $\mathcal{D}$. This integral regularization avoids certain clusters from dominating other clusters.

4.4.2 Deploying EM Batch-Wise

Having defined the cluster distributions, we proceed to the batch-wise EM iterations. Consider the t^{th} iteration forward pass with a batch $x_{i,i=1,\ldots,n}$. We first derive the posterior probabilities $p(z \mid x, \theta^t)$ of cluster assignments. The probability that x_i is allocated to cluster j for current parameters θ is,

$$p(Z = j \mid x_i, \theta) = \frac{p(Z = j \mid \theta) f(x_i \mid Z = j, \theta)}{\sum_{k=1}^{K} p(Z = k \mid \theta) f(x_i \mid Z = k, \theta)}. \tag{4.18}$$



Likelihoods
$f(x, Z \mid \theta) = H_\theta(x) = \text{sigmoid}(a_i^*/\gamma)$
EM Objective
$Q(\theta \mid \theta^t) = \sum_{i=1}^{n}\sum_{j=1}^{K} p(Z=j \mid x, \theta^t)\log[f(x_i, Z=j \mid \theta)]$
Neural Network
x_i
a_i
Normalization
a_i^*
$\text{sigmoid}(a_i^*/\gamma)$
$\text{softmax}(a_i^*)$
$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} p_{ij}\log[\text{sigmoid}(a_{ij}^*/\gamma)]$
Loss To Backpropagate
θ_S
θ_j
Shared
Specific to jth Cluster
θ
Parameters
Posteriors
$p(Z \mid x_i, \theta^t) = p_i = \text{softmax}(a_i^*)$

Figure 4.3: Deploying Mixture-EM method for end-to-end training of a neural network for clustering. Normalized final layer relevance scores are used to determine cluster likelihoods and posteriors. Then, the EM-based objective is derived and backpropagated using the likelihoods and posteriors.

$p(z \mid \theta)$ denotes the priors of cluster allocation, which are derived from the previous step's posteriors. We set the prior for each cluster to $1/K$ as the datasets we use in this paper are uniformly distributed among K classes. Thus, the posterior for cluster j simplifies to the normalized cluster j distribution:

$$p(Z = j \mid x_i, \theta) = \frac{h_{\theta^s,\theta_j^e}(x_i)}{\sum_{k=1}^{K} h_{\theta^s,\theta_k}(x_i)}. \tag{4.19}$$

In consideration of the simplicity of numerical optimization, we reduce this ratio between sigmoids to the ratio between unnormalized exponentials, resulting in the softmax activation. Thus, the posterior $p(Z = j \mid x_i, \theta^t)$ is approximated by the softmax-activated a_{ij}^*. (Denoted by p_{ij} later):

$$p(Z = j \mid x_i, \theta^t) \approx \frac{e^{a_{ij}^*}}{\sum_{k=1}^{K} e^{a_{ik}^*}} = p_{ij}. \tag{4.20}$$

Then, we estimate the likelihood of parameters θ given observation x and cluster assignment z: $l(\theta; x, z)$, which is computed by taking the joint probability density of x and z for the current θ: $f(x, z \mid \theta)$. This likelihood is expandable to

$p(z \mid \theta) f(x \mid z, \theta)$. As we have specified the prior of cluster assignment $p(z \mid \theta)$ to be constant $1/K$, we can ignore it during optimization. Consequently, the joint density is represented by the conditional density $f(x \mid z, \theta)$, hence the cluster distribution:

$$f(x, z \mid \theta) = h_{\theta^s, \theta^e_j}(x_i) = \text{sigmoid}(a^*_{ij}/\gamma). \tag{4.21}$$

Once the posteriors and likelihoods of θ are estimated, we compute the loss for the current batch corresponding to the EM objective in Eq. 4.4:

$$L_{\theta|\theta^t} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{K} p_{ij} \log[\text{sigmoid}(a^*_{ij}/\gamma)]. \tag{4.22}$$

To improve the total likelihood for current posterior probabilities, we backpropagate this loss and update network parameters θ while holding p_{ij}s constant. Figure 4.3 shows the overall mixture-EM formulation with a neural network.

Batch-normalization of the relevance scores and division by γ is essential to constrain them to the predominantly linear region of the sigmoid with zero mean, enabling the sigmoid outputs' PDF behavior. γ is a hyperparameter that depends on the batch size. Since we use a batch size of 128, the normalized relevance scores are considerably dispersed around zero and bounded between [-11.23, 11.23]. γ to 5, ensuring that the sigmoid of these scores falls within the sigmoid linear region. Moreover, because our framework executes EM steps in batches, the optimizer sees a batch of data that reflects the whole sample space in each iteration. The higher the batch size, the more accurately it can reflect the sample space for every EM iteration. In the meantime, the smaller the batch size, the higher the addition of noise to the sample space approximation, which assists in regulating the optimization process.

4.4.3 Image Clustering with Consistency Optimization

Training a network with a clustering objective solely on the original data is insufficient for image clustering, as the network could overfit to irrelevant low-level features. We transform images during the learning process to enable the network to extract semantically meaningful features from images in conjunction with mixture-EM optimization. T transforms an original image x_i into its altered version x_i^{tr}: $x_i^{tr} = T(x_i)$. It comprises common data augmentations such as random cropping, shifting, rotating, and scaling, as well as random adjustments to image brightness, contrast, hue, and saturation. As before, we calculate relevance scores for the augmented images: $a_i^{tr} = g_\theta(x_i^{tr})$. We add an additional term to the loss in Eq. 4.22, the log-likelihood of the augmented image weighted by the original image's posterior. Therefore, the loss is,

$$L_{\theta|\theta^t} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} p_{ij}\left[\log[\sigma(a_{ij}^*/\gamma)] + \log[\sigma(a_{ij}^{tr*}/\gamma)]\right]. \qquad (4.23)$$

Optimization of the log-likelihoods of original and augmented images both in favor of the original image's posterior (p_{ij}) enables the neural network to preserve similar network outputs for both images. In continuation of this purpose, we utilize a similar approach to consistency regularization [89] to urge the network to produce similar outputs for the original and augmented images. We compute q_i, the posterior probabilities for the augmented input x_i^{tr}: $q_i = \text{softmax}(g_\theta^*(x_i^{tr}))$, and minimize the KL divergence between p_i and q_i:

$$D_{KL}(p_i \,||\, q_i) = \sum_{j=1}^{K} p_{ij}\log\frac{p_{ij}}{q_{ij}}. \qquad (4.24)$$

When minimizing the KL divergence, we hold the original image's posteriors p_i constant, thereby creating temporary soft labels for the altered image response q_i. We fuse the consistency optimization (Eq. 4.24) with the EM optimization (Eq. 4.23). For each batch, gradient steps in optimizing the two objectives are

Algorithm 3 Two-fold optimization for batch $[x_{i,i=1\ldots n}]$ and transformed batch $[x^{tr}_{i,i=1,\ldots,n}]$ with the network parameterized by θ

```
EM Optimization
    Calculate cluster relevances scores
    $a_i = [a_{i1}, \ldots, a_{ik}] = g_\theta(x_i)$. Similarly $a^{tr}_i = g_\theta(x^{tr}_i)$
    Normalize relevance scores
    $a^*_{ij} = \frac{a_{ij} - \mu_j}{\sigma_j}$. Similarly $a^{tr*}_{ij}$.
    Calculate posterior probabilities
    $p_i = \text{softmax}(a^*_i)$.
    Get clustering loss: $L_{\theta|\theta^t}$,
    $-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} p_{ij}\left[\log[\sigma(a^*_{ij}/\gamma)] + \log[\sigma(a^{tr*}_{ij}/\gamma)]\right]$ (Eq. 4.23)
    Backpropagate loss and update parameters $\theta$
    $\theta^t \rightarrow \theta^{t+1}$
Consistency Optimization for Augmented Images
    Calculate posteriors $p_i$ with updated $\theta$
    $p_i = \text{softmax}(g^*_\theta(x_i))$
    Calculate posteriors for augmented images
    $q_i = \text{softmax}(g^*_\theta(x^{tr}_i))$
    Step in optimize the KL Divergence ($p_{ij}$ constant)
    $D_{KL}(p_i \mid\mid q_i) = \sum_{j=1}^{K} p_{ij}\log\frac{p_{ij}}{q_{ij}}$ (Eq. 4.24)
```

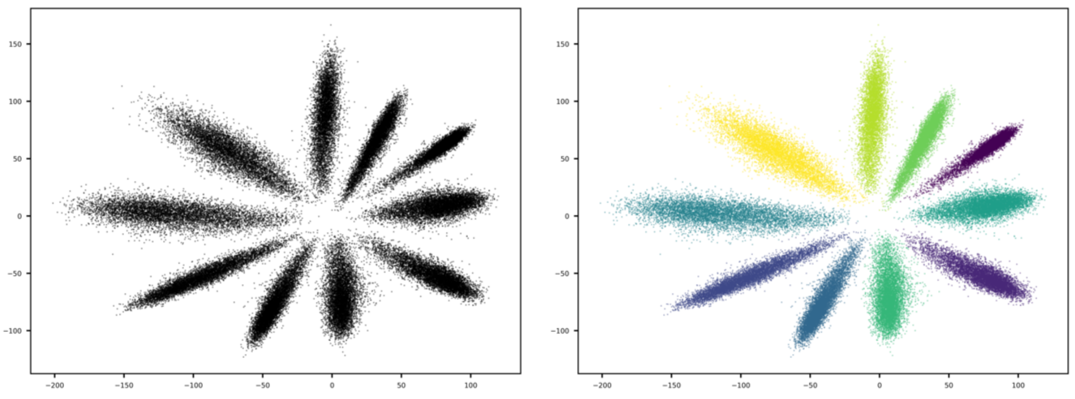
performed sequentially using different optimizers. This results in a two-step optimization procedure, as seen in Algorithm 3.

After one gradient step of EM optimization, the network parameters are modified in favor of an original image's current posteriors p_i, resulting in an improved posterior. The consistency objective encourages the network to keep its response to the altered image q_i close to the original response p_i. It is inefficient to optimize these two goals simultaneously. Because while the original image's posteriors p_i update to more accurate values, the consistency goal requires the altered image's posteriors q_i to remain closer to the old p_i. Therefore, we alternately optimize these two objectives. After one step of EM optimization, we recalculate the original image posteriors to optimize the consistency objective because the network has been updated.

4.5 Experiments

4.5.1 Two-Dimensional Space

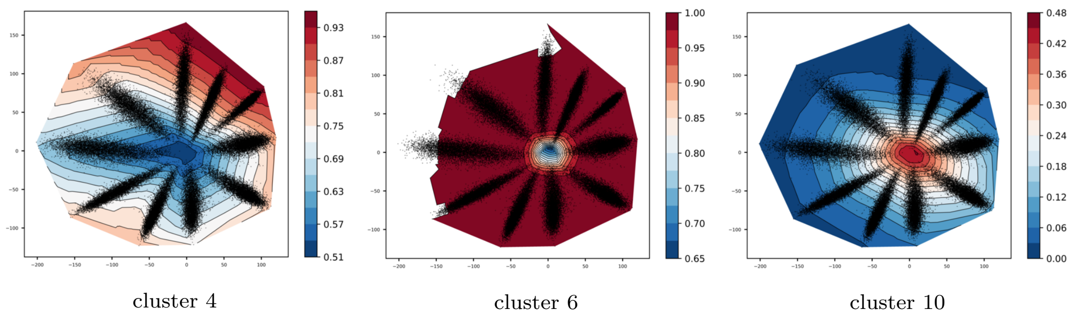
We first perform a small-scale clustering study with a 2-d MNIST [113] dataset-generated space to investigate cluster assignments and the learned cluster distributions. We supervise train a CNN with the MNIST dataset to generate 2-dimensional data. The network comprises a bottleneck layer of two nodes preceding the final layer of ten nodes. Once the network has been trained, we retrieve the bottleneck layer output for all images, resulting in set D of two-dimensional points (See Figure 4.4a). This supervised setup for dimensionality reduction allows 2-d samples to disperse into observable clusters more effectively than unsupervised methods [114–116].



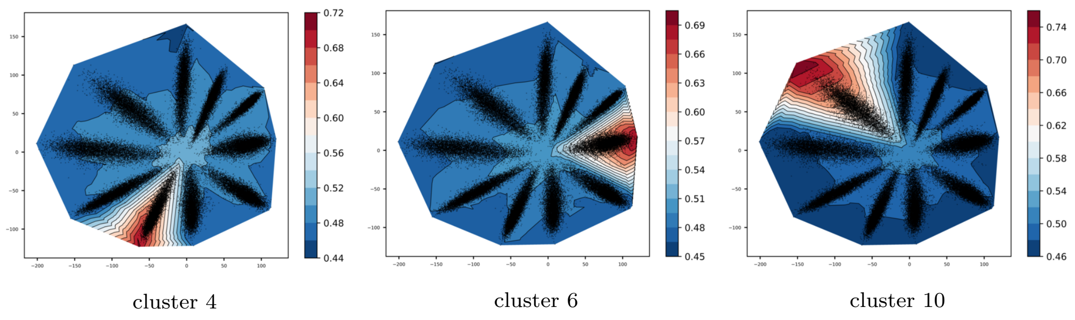
(a) Input Space **(b)** Categorized Space

Figure 4.4: The 2-dimensional sample space produced from the MNIST dataset and the clustered set following training with our approach. Our approach successfully captures the observable clusters.

We employ a three-layer perceptron that has two hidden layers with 32 nodes in each and a final layer with ten nodes (for ten clusters) to cluster this set. We employ the mixture-EM optimization (Eq. 4.22), which utilizes only the original data. We apply a batch size of 128, a γ value of 5, and the Adam optimizer [48] with a learning rate (LR) of 0.001 to train the network. Figure 4.4b depicts the clustered space after training for ten epochs, in which the network successfully captures the observable clusters. We use a running mean and a running standard deviation for normalizing relevance scores rather than using only each batch's statistics. Using running statistics smoothes the optimization and allows inference with varying batch sizes.



(a) Without normalizing relevance scores



(b) With normalized relevance scores

Figure 4.5: Effect of the relevance score normalization: Cluster distribution plots of selected clusters after training. (a) Without relevance score normalization, certain clusters dominate others by acquiring all data points (Here, cluster 6). (b) With relevance score normalization, cluster distributions behave as PDFs of observation with a shared integral across the input space. Each distribution shows a high likelihood for each observable group.

Figure 4.5 shows the cluster distribution ($f(x \mid Z = j, \theta)$ or $h_{\theta^s,\theta^e_j}(x)$) contour plots with input x. We select three clusters out of the ten that correspond to the final layer's fourth, sixth, and tenth nodes. Figure 4.5a depicts the selected cluster distributions after training with unnormalized relevance scores a_{ij}, in which the distributions do not adequately describe potential clusters. In addition, their space integrals do not appear to share the same value. The distribution of cluster 6 maximizes likelihood for all samples, but the distribution of cluster 10 is quite low for all samples. In this instance, the model falls to a trivial solution because all samples are assigned to cluster 6. The distributions in Figure 4.5b are from the same clusters but when trained using normalized relevance scores a^*_{ij}. The contour plots demonstrate that each distribution covers a visible cluster by displaying high likelihoods for those locations. Since all distributions empirically show a common integral over the observation space, they behave similarly to PDFs. The normalization of relevance scores is thus vital for clustering.

Table 4.1: Average cluster distribution outputs with a batch of 128 samples for the clusters shown in Figure 4.5. The average outputs reach arbitrary values without relevance score normalization. The average distribution output is closest to 1 for Cluster 6. All average distribution outputs reach 0.5 with relevance score normalization.

Cluster index j	4	6	10
$\frac{1}{n}\sum_{i=1}^{n} h_{\theta^s,\theta^e_j}(x_i)$ with a_{ij}s (un-normalized)	0.736	0.999	0.108
$\frac{1}{n}\sum_{i=1}^{n} h_{\theta^s,\theta^e_j}(x_i)$ with a^*_{ij}s (normalized)	0.498	0.507	0.504

To validate the PDF behavior, we experimentally estimate the integrals of the cluster distributions for these three clusters. As indicated by Eq. 4.16, the cluster distribution h integral across space $\mathcal{D}$ is proportional to the mean of h in n uniform samples. We further showed that because of the relevance score normalization and further restriction to the linear region of sigmoid, the average cluster distribution output of n samples reaches 0.5 (Eq. 4.14). To show this phenomenon empirically, we estimate the mean cluster distribution outputs $\frac{1}{n}\sum_{i=1}^{n} h_{\theta^s,\theta^e_j}(x_i)$ for the three clusters across 128 samples and show the results in Table 4.1. Without relevance score normalization, the batch's average cluster distribution outputs have varying values, with dominant cluster 6 showing an average approaching 1. When relevance score normalization is used, the average distribution output of all three clusters is close to 0.5 for the batch. We can conclude that they behave as PDFs of x since they share a common integral of $0.5V$ over the space $\mathcal{D}$ (V: the volume of $\mathcal{D}$ according to Eq. 4.16).

4.5.2 Image Clustering

We test our algorithm on the four image datasets, MNIST [113], CIFAR10, CIFAR100 [84] and STL10 [117]. CIFAR100 comprises 100 classes that are further abstracted into 20 superclasses (Each superclass contains five classes). We cluster CIFAR100 into its superclasses, whereas other datasets are into their usual number of classes. STL10 has 13k labeled samples and 100k unlabeled samples. We only use the labeled set for clustering omitting their labels during training.

For the STL10 dataset, we employ a 9-layer CNN; for CIFAR10/100, a 7-layer CNN; and for MNIST, a 5-layer CNN (Table 4.2 outlines these network architectures). Before sending RGB images to the network, they are converted to grayscale with a single channel. Then, we apply horizontal and vertical Sobel filters to the single-channel images. Therefore, the input to the network is a stack of two planes of horizontal and vertical edges which share the same height and width as the original ($2 \times H \times W$). This preprocessing prevents the network from overfitting to color and helps the network to learn broad features.

Table 4.2: Network architectures used. Cn stands for a convolution that carry n filters, M denotes Max-pooling, and Fn stands for a dense layer that has n output nodes.

Dataset	**Architecture**	**Params**
MNIST	C64 M C128 M C256 F32 F10	0.8M
CIFAR	C64 C64 M C128 C128 M C256 C256 Fn	1.3M
STL10	C64 C64 M C128 C128 M C256 C256 M C256 C256 F10	2.7M

We employ our two-fold optimization (Section 4.4.3 and Algorithm 3), keeping two Adam [48] optimizers, one for EM optimization (LR = 5e-5) and one for consistency optimization (LR = 1e-4). Assigning a higher learning rate for consistency optimization is vital because the learned clusters shall be more accurate if learning generic semantic features takes precedence over clustering. The models are trained for 250 epochs with batch size 128 and $\gamma = 5$ using the whole

Table 4.3: Clustering accuracy (%) comparison. † - methods that rely on k-means. ∗ - DeepCluster [100] and ADC [99] figures reported by Ji *et al.* , [90]. Our two-fold optimization outperforms all conventional clustering algorithms, all deep algorithms that continue to rely on k-means, and in some circumstances, even state-of-the-art single-stage deep clustering techniques.

Approach	**STL10**		**CIFAR10**		**CIFAR100**		**MNIST**	
	Acc	NMI	Acc	NMI	Acc	NMI	Acc	NMI
Random Network	10.01		10.35		5.16		21.15	
K-means [118]	19.2	12.5	22.9	8.7	13.0	8.4	57.2	50.0
Spectral Clustering [119]	15.9	9.8	24.7	10.3	13.6	9.0	69.6	66.3
JULE [120]	27.7	18.2	27.2	19.2	13.7	10.3	96.4	91.3
Triplets [121] †	24.4	-	20.5	-	9.94	-	52.5	-
AE [122] †	30.3	25.0	31.4	23.4	16.5	10.0	81.2	72.6
Sparse AE [123] †	32.0	25.2	29.7	24.7	15.7	10.9	82.7	75.7
Denoising AE [124] †	30.2	22.4	29.7	25.1	15.1	11.1	83.2	75.6
Var. Bayes AE [125] †	28.2	20.0	29.1	24.5	15.2	10.8	83.2	73.6
SWWAE [126] †	27.0	19.6	28.4	23.3	14.7	10.3	82.5	73.6
GAN [127] †	29.8	21.0	31.5	26.5	15.1	12.0	82.8	76.4
DEC [104] †	35.9	27.6	30.1	25.7	18.5	13.6	84.3	77.2
K-meansNet [105]	-	-	20.23	6.87	-	-	87.76	78.70
DeepCluster [100] †	33.4*	-	37.4*	-	18.9*	-	65.6*	-
DECCA [91]	-	-	-	-	-	-	96.37	0.9087
SCAE [128]	-	-	33.48	-	-	-	**99.0**	-
DAC [101]	47.0	36.6	52.2	40.0	23.8	18.5	97.8	93.5
ADC [99]	53.0	-	32.5	-	16.0*	-	**99.2**	-
IIC [90]	59.8	49.6	**61.7**	**51.1**	25.7	**22.5**	**99.2**	-
IIC [90] our setting	47.12	41.02	44.17	34.89	16.18	9.88	95.72	93.96
EM Optimization (Eq. 4.23)	49.61	41.99	49.53	39.59	19.36	12.23	98.44	95.67
Two-Fold Optimization	**63.84**	**50.3**	57.97	47.03	**25.94**	19.72	**98.88**	**96.74**
	± 2.6	± 2.13	± 3.03	± 2.04	± 0.8	± 0.41	± 0.07	± 0.16

datasets, with the exception of STL10, for which we only utilize the labeled set.

Table 4.3 contrasts our methodology with conventional and state-of-the-art single-stage deep clustering techniques. Here, we analyze the clustering accuracy and normalized mutual information (NMI) metrics. Following the Hungarian approach [129], we match the predicted cluster index to the actual label using linear sum assignment [90]. We present our model performance when trained with the mixture-EM optimization using original and transformed images (Eq. 4.23), as well as the two-fold optimization (Algorithm 3). We show the average accuracy across six trials for each dataset, and for the two-fold optimization, we further present the margin of error using a 95% confidence interval.

Our models, when trained with the mixture-EM optimization using original and altered images, outperform all conventional algorithms and deep clustering approaches that still rely on k-means, including DeepCluster [100] and Deep Embedded Clustering (DEC) [104]. The two-fold optimization considerably enhances the mixture-EM optimization, in most cases outperforming end-to-end deep clustering methods such as IIC [90], DECCA [91], ADC [99], DAC [101], and SCAE [128]. IIC [90] employs both labeled and unlabeled segments of STL10, achieving **49.9%** when trained in only labeled segment. In addition, IIC uses many transformed samples in a single batch by repeatedly augmenting each sample (5 times). With once-per-batch augmentation, IIC only reaches **47%** for STL10. IIC also uses multiple clustering heads and overclustering strategies.

In contrast, when trained with only the labeled section and with once per-batch augmentation, our architecture achieves **57.93%** in STL10. We demonstrate that IIC, in our training environment, with a single head and once a batch augmentation, yields worse performance to mixture-EM and two-fold optimization. Keeping similar responses to the original and altered images, the consistency objective in the two-fold optimization has the same concept as IIC. Therefore, we can conclude that the two-fold optimization is superior to consistency optimization alone. Figure 4.6 demonstrates the learning curves of IIC (in our environment) and the two-fold optimization, proving this point.

Notably, several deep image clustering approaches report superior performance to ours [102, 103, 109, 130].. The majority of methodologies, however, are multi-stage approaches that include initialization methods, numerous losses, and fine-tuning. For instance, the majority of SCAN's [109] improved performance can be attributed to the pre-text task learned before clustering and self-labeled fine-

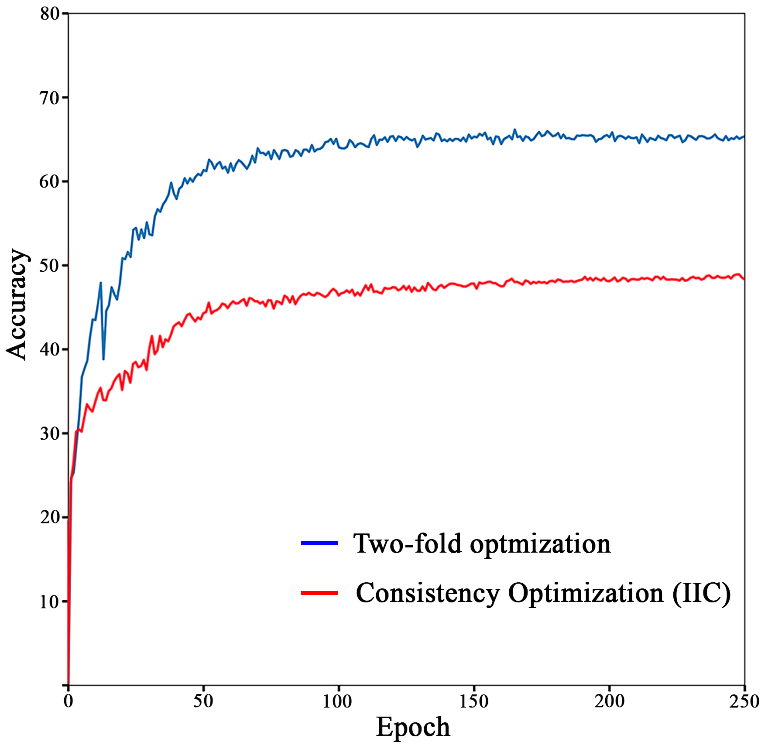


Figure 4.6: Learning curves in STL10. Blue curve represents our two-fold optimization, and red curve depicts the IIC [90] consistency optimization in our environment. With a combination of mixture-EM and consistency optimization, our two-fold optimization demonstrates a considerable improvement above sole consistency optimization.

tuning post-clustering. While our mixture-EM formulation can also be expanded to include pre-text learning, stronger augmentation, and self-labeling, we ignore these improvements in this research.

4.5.3 Visualizations

Here, we empirically analyze cluster modeling and neural network's feature extraction using an STL10 clustered network. Figure 4.7 displays the model response prior to softmax in two-dimension for a 2560 images-subset of STL10. Using T-SNE algorithm [115], we translate the model response to 2-d while maintaining the original dimension relationship between vectors. The network response for a randomly initialized network is depicted in Figure 4.7a. Figure 4.7b depicts the network's response after training with our two-fold optimization but

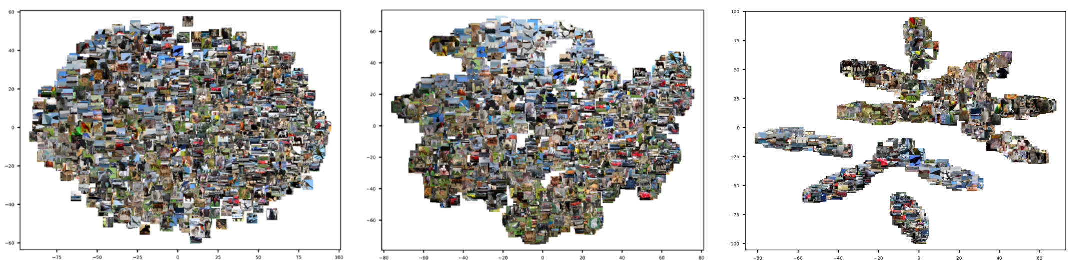

(a) Random Network **(b)** No Normalization **(c)** Normalization

Figure 4.7: Two-dimensional mapping of network response for STL10 subset. a) Random network: Has no knowledge of potential clusters. b) Network trained without relevance score normalization: falls to a trivial solution of one cluster. c) Network trained with relevance score normalization: meaningful clusters have been detected.

without relevance score normalization. Figure 4.7c depicts the network response after training with relevance score normalization. The randomly initialized network lacks knowledge regarding a potential clustering basis. Our approach trains such a network to successfully cluster the sample space into meaningful groups with visible borders. The relevnce score normalization plays a key role in our formulation and prevents trivial convergence in which a single cluster is produced with no members in other clusters.

Figure 4.8 shows the ten images with the highest relevance scores for five selected clusters, as well as the image synthesized to maximize the particular final layer node. We create the synthesized image by conducting gradient ascent on a randomly initialized input to maximize the node's response prior to activation while freezing network parameters [87]. The visualizations that result demonstrate that the model groups images with similar high-level features. Each cluster's synthesized image correlates to the top member images' high-level patterns. For instance, cluster 4 is predominantly activated for dogs, and the synthesized image displays leg patterns that match. Cluster 7's finest images are primarily of cats, and the synthesized image contains matching dot patterns found in the 10 images. The majority of the top images in Cluster 8 depict deer viewed from the side, and the synthesized image matches their body form. This experiment demonstrates the trained network's capacity to model rich clusters end-to-end.

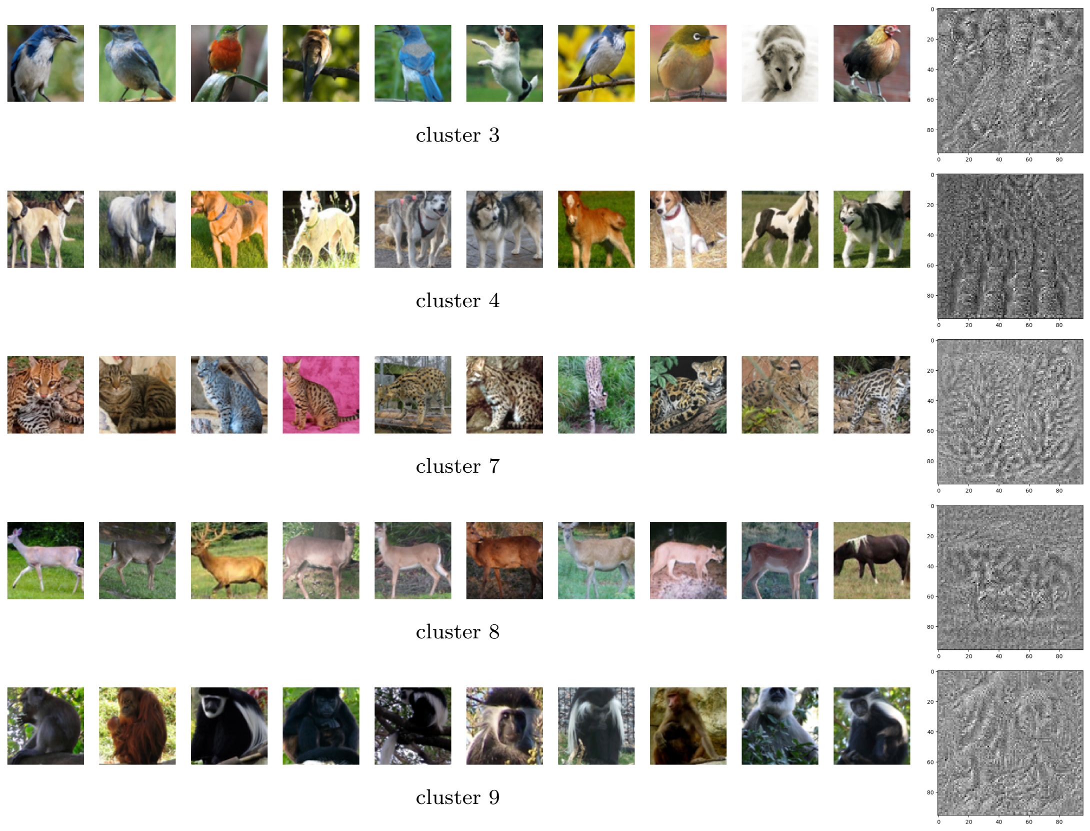
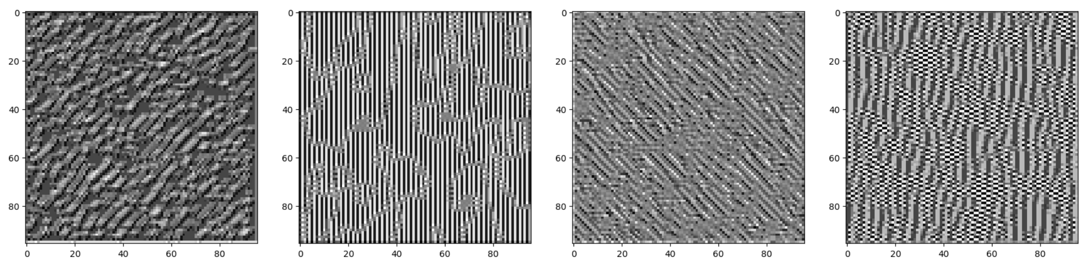


Figure 4.8: STL10 clustered network. Five selected clusters, each represented by a particular neuron in the final layer. Each row displays the ten images which show the highest relevance score for the respective neuron, followed by the image synthesized to maximize that neuron. Images containing similar high-level features have been grouped together. Similar bodily shapes and patterns are displayed in the synthesized images.

Finally, we evaluate the convolutional feature extraction by visualizing the model's convolutional filters. To visualize a filter, we optimize a randomly initialized image while keeping the network parameters constant to maximize the filter's output [87]. We plot those visualizations of selected filters in the first and last convolutional layers in Figure 4.9. The shallow filters from the first layer have learned to extract low-level features (Figure 4.9a), whereas the deep final convolutional layer filters have learned high-level features (Figure 4.9b). Thus, it is evident that CNN learns features that are dispersed with network depth, increasing complexity with the depth. Consistency optimization, which encourages the network to maintain similar responses to original and altered images, enables the CNN filters to learn such features, producing an informative base for clustering.


(a) Initial convolutional layer filters

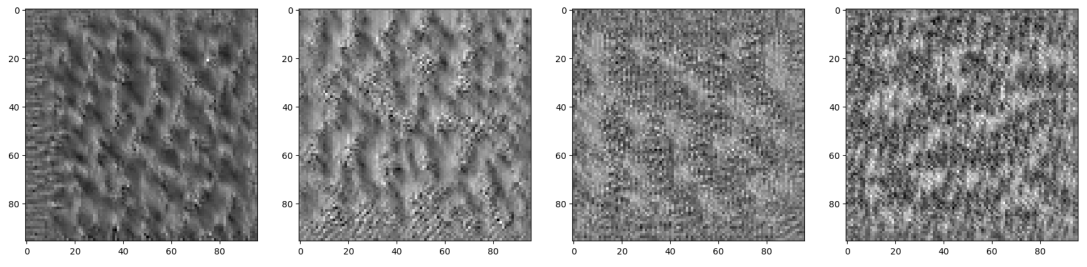

(b) Final convolutional layer filters

Figure 4.9: Visualization of the first and last convolutional layers' selected filters. The first layer filters display simple features, but the deep filters display intricate patterns. Overall, the convolutional filters are learning features with climbing the abstraction level.

4.6 Conclusion

Our mixture-EM formulation trains a network to learn cluster distributions and cluster assignments simultaneously. Normalizing the cluster relevance scores across batches allows the approximated cluster distributions to behave as PDFs of the observation, hence avoiding trivial solutions. Visualizations empirically confirmed meaningful cluster modeling and rich convolutional feature extraction. We experiment our mixture-EM clustering objective without substantial data augmentation, alternative representation learning approaches such as pre-text tasks, deeper networks, sophisticated initialization or fine-tuning methods. Despite this, it is worthwhile to research the possibilities of further enhancements. While we suggest the sigmoid activation in conjunction with normalized relevance scores to represent cluster distributions, it would be interesting to investigate more complex activations or techniques to create better distributions.

Chapter 5

REGULARIZE ROUTING WITH CLUSTERING LOSS

In this chapter, we discuss the potential of the clustering objective introduced in Chapter 4 as a regularization loss for the gating modules in our multi-path networks and possible extensions toward regulating spare-path allocation methods. We first incorporate this clustering loss in each cross-connection-based routing layer in our multi-path networks and observe its regularization effect.

5.1 Regularization of Cross-connection based Routing

When the multi-path networks are trained with only the end-objective, we identified that certain gating vectors showed skewed responses over the entire dataset, i.e., among a gating vector, a particular gate predominantly activates for all samples, leaving other gates with lesser activations. We observed that the clustering objective mitigates this behavior, enabling gates to be more evenly activated. We add the mixture-EM-based objective, as in Equation 4.4, to the outputs of each gating network to encourage the gating to cluster their input spaces in addition to updating from the end task loss. We weigh the clustering objective with respect to the end-task loss to tune the priority of regularization over the main task. We maintained the weighting coefficient of the clustering objective with respect to the primary loss of 0.01 for our experiments.

Figure 5.1 shows a chosen gate's response in a two-path network (Note that a gate's value can vary between 0-1). The corresponding gating vector has two gates since there are two possible pathways to forward the input tensor. The top figure shows the gate response when the network is trained with only the end objective. In contrast, the bottom figure shows its response when trained with the clustering objective as an additional gating regularization. Without any explicit regularization, the gate is less activated all the time. As a result, the gate's output lies below 0.4 for all inputs. However, with our clustering objective employed as a regularization loss for gating, the gate's response stretches above 0.5, better activating it for certain samples.

However, we note that this clustering objective on each gating vector functions as a gate-wise local regularization for the multi-path networks as it focuses on clustering the input space to each gating vector only. Thus, the impact of regularization should be tuned via the weighting coefficient of the clustering objective. In particular, we observed that the clustering objective has to be given lesser weight with the increased number of parallel paths as it solely focuses on clustering its respective input space to the given number of clusters. Using the prior probabilities in the EM objective is also important since the gate clustering shall be treated as an un-even clustering task. Also, we note the importance of further updating the clustering objective to find the optimum number of clusters for the given gating vector, which will also decide the optimum number of paths for a given layer.

5.2 Towards Global Regularization

The clustering objective forces each gating network to distribute its inputs and functions as a local regularization for the gating function. Such local regularization does not consider the network's end outcome. Thus, it is interesting to exploit further updating the clustering objective to consider the entire layer with parallel paths and multiple gating functions into account. Furthermore, we encourage the exploration of alternative regularization methods for the gating

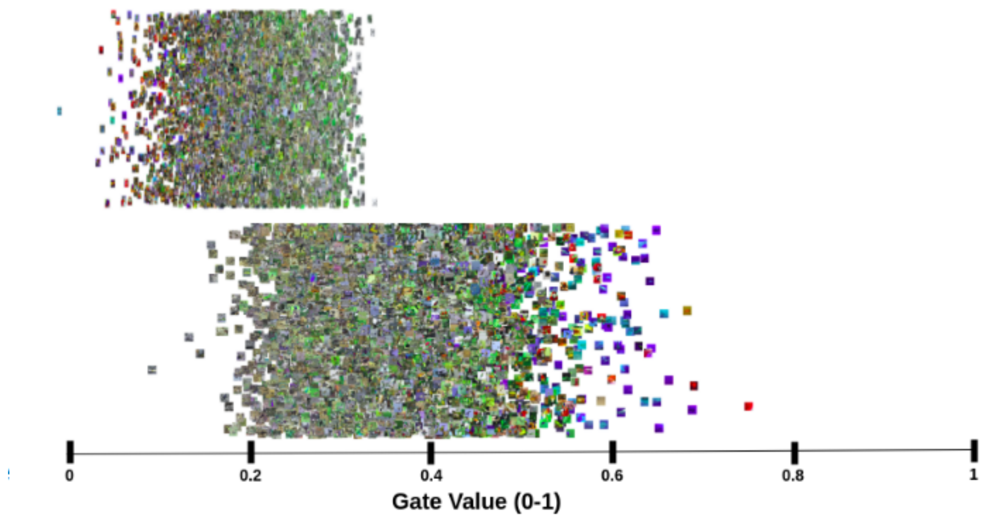


Figure 5.1: A chosen gate's response in a two-path network for a sample set of input space. The corresponding gating vector has two gates whose activations sum to 1. The top image shows the gate response without regularization loss, and the bottom image shows the gate response with regularization loss. Without regularization, the gate is mostly deactivated for all samples, whereas with regularization, the gate's response stretches above 0.5, better activating it for certain samples.

function that looks at the function of all gates in the primary network throughout the depth overall.

5.3 Towards Sparse Multi-path Networks

The proposed multi-path networks contain soft routers, i.e., all paths are activated for a single sample with different soft weights. Also, one layer contains multiple gating functions, each employed in one of the parallel paths, which leads to difficulties in regularizing the routing layers and interpreting each gating function's purpose. Thus, it is worthwhile to simplify the routing process further to allocate incoming feature maps to parallel paths, do parallel path computation, and accumulate the results for the next layer. Such a design will simplify the path allocation process, and studying the effect of clustering objectives for path allocation is easier.

Such simplified multi-path allocation (divide to paths, compute, and accumulate layer-wise) will be more suitable for a sparse-path allocation system where only one or several paths are activated. Such a sparse-path allocation method can handle more diversity among the input space and can even be extended toward learning from multiple domains. Thus, it is worthwhile to extend such routing mechanisms toward regulating sparse multi-path networks. For example, a clustering-based router can route an incoming tensor to one or a few next-layer paths based on the clusters it falls under. This can be considered an alternative clustering-based path allocator for existing sparse multi-path networks [78,80–83].

Unlike in soft routing, such a sparse routing network would allow more diversity in the dataset to be addressed and allow efficient scalability as, despite how large the network, only a portion of it will be used for one sample. However, modeling and training such a sparse-path network would meet several challenges, such as the engineering of the network with sparse activations using multiple devices and parallelization for parallel paths, optimization, and the need for large-scale data.

Chapter 6

CONCLUSIONS

Deep networks require a substantial amount of resources. Thus, it is vital to attain the best performance for the given number of parameters with appropriate depth and width. In this dissertation, we explored strengthening layer-wise feature extraction by stacking parallel paths and introducing novel mechanisms to route the input among such parallel paths end-to-end intelligently. The proposed multi-path networks behave as a composite network of parallel paths with layer-wise soft path allocation methods.

The neural mixture-EM formulation trains a network to model advanced cluster distributions and cluster the input space accordingly. It is a superior alternative to traditional clustering algorithms such as k-means or GMM; one can substitute traditional clustering in any application with a neural network and mixture-EM optimization. Using a neural network with appropriate depth and a straightforward training procedure makes our clustering technique easy to adopt for any clustering problem with different cluster modeling complexity. Furthermore, the neural mixture-EM optimization can be easily plugged into gating-like sub-networks within a complex network which forces the sub-networks to cluster their input spaces while also learning from the end objective (Attention). Such cluster-based regularizing method's utility will be significant with sparse resource allocation methods.

As a standalone end-to-end neural network-based clustering approach, the proposed neural mixture modeling can benefit from sophisticated augmentation methods, initialization methods, representation learning methods, and deeper networks. Furthermore, improving the current clustering framework by tuning the mix of cluster-specific and shared layers is worthwhile. For our experiments, we use a single-column neural network for clustering, leading to the final layer's parameters being cluster-specific and all other parameters being shared between clusters. Thus, it is interesting to see the effect of cluster distribution modeling with more layers in the network allocated as cluster-exclusive layers (By splitting into multiple columns after a certain number of layers). Exploring advanced transformations to approximate cluster distributions and posterior probabilities from final layer neurons is also important. Furthermore, we note the importance of advanced arrangements of layer computation, including layer activation, to extend towards clustering datasets other than distance/distribution-based spaces, such as connectivity-based clustering spaces. We also note the potential direction towards updated formulation to cluster un-even datasets accommodating prior probabilities and updating towards dynamic online clustering, which can also capture emerging clusters.

Once employed as a local regularization for each gating network, the clustering objective was able to enforce each gating network to distribute its input space better, mitigating always activated or deactivated gates. However, we note the importance of tuning the weight of the clustering-based regularization with respect to the primary loss function based on the number of paths in the network, as the clustering objective in each gating network functions as a local regularization. It is also important to extend such regularization methods towards global regularization techniques, considering all the gating in one routing layer or all the gating vectors in the network as whole. Also, we note the importance of having prior probabilities calculated and used in the EM formulation that supports uneven cluster assignments in multi-path network gating. We also note the future direction towards using the clustering objective to decide the optimum number of paths for a given layer.

The proposed multi-path networks are soft routers; hence, all neurons will be used during training and inference. It is worthwhile to extend this mechanism towards sparse resource allocation along with the clustering objective to govern the behavior of gating networks. A sparse resource allocation method can handle more diversity in the input space. Only a portion of the network is used for each sample, leading to improved scalability.

LIST OF PUBLICATIONS

Articles Accepted in International Conferences and Journals

1. **D. Tissera**, K. Vithanage, R. Wijesinghe, K. Kahatapitiya, S. Fernando, R. Rodrigo, "Feature-dependent cross-connections in multi-path neural networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 4032-4039.
2. **D. Tissera**, K. Vithanage, R. Wijesinghe, A. Xavier, S. Jayasena, S. Fernando, R. Rodrigo, "Neural mixture models with expectation-maximization for end-to-end deep clustering," in *Neurocomputing*, vol. 505, pp. 249-262, 2022.
3. **D. Tissera**, R. Wijesinghe, K. Vithanage, A. Xavier, S. Fernando, R. Rodrigo, "End-to-end data-dependent routing in multi-path neural networks," in *Neural Computing and Applications*, 2023 - Accepted.

BIBLIOGRAPHY

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[2] ——, "Identity mappings in deep residual networks," in *European Conference on Computer Vision (ECCV).* Springer, 2016, pp. 630–645.

[3] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *Proceedings of International Conference on Learning Representations(ICLR)*, 2015.

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

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[6] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2016, pp. 87.1–87.12.

[7] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1492–1500.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[9] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3642–3649.

[10] M. Wang, "Multi-path convolutional neural networks for complex image classification," *arXiv preprint arXiv:1506.04701*, 2015.

[11] K. Kahatapitiya, D. Tissera, and R. Rodrigo, "Context-aware automatic occlusion removal," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 1895–1899.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[14] D. Tissera, K. Kahatapitiya, R. Wijesinghe, S. Fernando, and R. Rodrigo, "Context-aware multipath networks," *arXiv preprint arXiv:1907.11519*, 2019.

[15] D. Tissera, K. Vithanage, R. Wijesinghe, K. Kahatapitiya, S. Fernando, and R. Rodrigo, "Feature-dependent cross-connections in multi-path neural networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 4032–4039.

[16] D. Tissera, R. Wijesinghe, K. Vithanage, A. Xavier, S. Fernando, and R. Rodrigo, “End-to-end data-dependent routing in multi-path neural networks,” *Neural Computing and Applications*, pp. 1–20, 2023.

[17] D. Tissera, K. Vithanage, R. Wijesinghe, A. Xavier, S. Jayasena, S. Fernando, and R. Rodrigo, “Neural mixture models with expectation-maximization for end-to-end deep clustering,” *Neurocomputing*, vol. 505, pp. 249–262, 2022.

[18] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[19] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, p. 533, 1986.

[21] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, “Learning internal representations by error propagation,” 1985.

[22] Y. LeCun, 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.

[23] M. I. Jordan, “Serial order: A parallel distributed processing approach,” in *Advances in psychology.* Elsevier, 1997, vol. 121, pp. 471–495.

[24] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using

rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[27] M. Zhou, Y. Bai, W. Zhang, T. Zhao, and T. Mei, "Look-into-object: Self-supervised structure modeling for object recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 774–11 783.

[28] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh *et al.*, "Symbolic discovery of optimization algorithms," *arXiv preprint arXiv:2302.06675*, 2023.

[29] P. Wang, S. Wang, J. Lin, S. Bai, X. Zhou, J. Zhou, X. Wang, and C. Zhou, "One-peace: Exploring one general representation model toward unlimited modalities," *arXiv preprint arXiv:2305.11172*, 2023.

[30] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," *arXiv preprint arXiv:2010.10504*, 2020.

[31] P.-Y. Huang, V. Sharma, H. Xu, C. Ryali, H. Fan, Y. Li, S.-W. Li, G. Ghosh, J. Malik, and C. Feichtenhofer, "Mavil: Masked audio-video learners," *arXiv preprint arXiv:2212.08071*, 2022.

[32] T. Zhou, Z. Ma, Q. Wen, L. Sun, T. Yao, W. Yin, R. Jin *et al.*, "Film: Frequency improved legendre memory model for long-term time series forecasting," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 677–12 690, 2022.

[33] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are transformers effective for time series forecasting?” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 9, 2023, pp. 11 121–11 128.

[34] T. Brown, 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,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[35] OpenAI, “Gpt-4 technical report,” 2023.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Adv. in Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[37] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[38] S. Gao, P. Zhou, M.-M. Cheng, and S. Yan, “Masked diffusion transformer is a strong image synthesizer,” *arXiv preprint arXiv:2303.14389*, 2023.

[39] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[41] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[42] T. Dietterich, “Overfitting and undercomputing in machine learning,” *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 326–327, 1995.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[44] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[46] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," *Advances in neural information processing systems*, vol. 4, 1991.

[47] N. Srivastava, G. 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, pp. 1929–1958, 2014.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Procedings of International Conference on Learning Representations (ICLR)*, 2015.

[49] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[50] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning.* PMLR, 2019, pp. 6105–6114.

[51] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conference on Artificial Intelligence*, 2017.

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

[53] Z. Chen, Y. Deng, Y. Wu, Q. Gu, and Y. Li, “Towards understanding mixture of experts in deep learning,” *arXiv preprint arXiv:2208.02813*, 2022.

[54] R. Caruana, “Multitask learning,” *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[55] K.-H. Thung and C.-Y. Wee, “A brief review on multi-task learning,” *Multimedia Tools and Applications*, vol. 77, no. 22, pp. 29 705–29 725, 2018.

[56] M. Crawshaw, “Multi-task learning with deep neural networks: A survey,” *arXiv preprint arXiv:2009.09796*, 2020.

[57] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, “Cross-stitch networks for multi-task learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3994–4003.

[58] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, “Latent multi-task architecture learning,” in *Proceedings of AAAI Conference of Artificial Intelligence*, February 2019, pp. 4822–4829.

[59] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, “Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3205–3214.

[60] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” in *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[61] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with EM routing,” in *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[62] J. Hu, L. Shen, S. Albanie, G. Sun, and A. Vedaldi, “Gather-excite: Exploiting feature context in convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9401–9411.

[63] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 7132–7141.

[64] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3856–3866.

[65] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, “Eca-net: Efficient channel attention for deep convolutional neural networks,” *arXiv preprint arXiv:1910.03151*, 2019.

[66] A. Veit and S. Belongie, “Convolutional networks with adaptive inference graphs,” in *European Conference on Computer Vision*, 2018, pp. 3–18.

[67] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, “Blockdrop: Dynamic inference paths in residual networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8817–8826.

[68] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” *arXiv preprint arXiv:1505.00387*, 2015.

[69] Y. Rao, J. Lu, J. Lin, and J. Zhou, “Runtime network routing for efficient image classification,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 10, pp. 2291–2304, 2018.

[70] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, “Skipnet: Learning dynamic routing in convolutional networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 409–424.

[71] B. Chen, T. Zhao, J. Liu, and L. Lin, "Multipath feature recalibration densenet for image classification," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 3, pp. 651–660, 2021.

[72] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha *et al.*, "Resnest: Split-attention networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2736–2746.

[73] K. Yu, X. Wang, C. Dong, X. Tang, and C. C. Loy, "Path-restore: Learning network path selection for image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[74] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

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

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

[77] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," *arXiv preprint arXiv:1312.4314*, 2013.

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

[79] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.

[80] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional

computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[81] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," 2021.

[82] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby, "Scaling vision with sparse mixture of experts," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8583–8595, 2021.

[83] L. Wu, M. Liu, Y. Chen, D. Chen, X. Dai, and L. Yuan, "Residual mixture of experts," *arXiv preprint arXiv:2204.09636*, 2022.

[84] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[85] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[86] Facebook, "fb.resnet.torch." [Online]. Available: https://github.com/facebookarchive/fb.resnet.torch

[87] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[88] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[89] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *arXiv preprint arXiv:1610.02242*, 2016.

[90] X. Ji, J. F. Henriques, and A. Vedaldi, "Invariant information clustering for unsupervised image classification and segmentation," in *Proceedings of*

the International Conference on Computer Vision (ICCV), 2019, pp. 9865–9874.

[91] B. Diallo, J. Hu, T. Li, G. A. Khan, X. Liang, and Y. Zhao, “Deep embedding clustering based on contractive autoencoder,” *Neurocomputing*, vol. 433, pp. 96–107, 2021.

[92] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[93] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[94] S. C. Johnson, “Hierarchical clustering schemes,” *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[95] G. J. McLachlan, S. X. Lee, and S. I. Rathnayake, “Finite mixture models,” *Annual review of statistics and its application*, vol. 6, pp. 355–378, 2019.

[96] J. A. Hartigan, “Direct clustering of a data matrix,” *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 123–129, 1972.

[97] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[98] C. M. Bishop, “Pattern recognition and machine learning: springer new york,” 2006.

[99] P. Haeusser, J. Plapp, V. Golkov, E. Aljalbout, and D. Cremers, “Associative deep clustering: Training a classification network with no labels,” in *German Conference on Pattern Recognition.* Springer, 2018, pp. 18–32.

[100] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[101] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, “Deep adaptive image clustering,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5879–5887.

[102] J. Wu, K. Long, F. Wang, C. Qian, C. Li, Z. Lin, and H. Zha, “Deep comprehensive correlation mining for image clustering,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 8150–8159.

[103] S. Han, S. Park, S. Park, S. Kim, and M. Cha, “Mitigating embedding and class assignment mismatch in unsupervised image classification,” in *16th European Conference on Computer Vision, ECCV 2020*. Springer, 2020.

[104] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in *International Conference on Machine Learning (ICML)*, 2016, pp. 478–487.

[105] X. Peng, I. W. Tsang, J. T. Zhou, and H. Zhu, “k-meansnet: When k-means meets differentiable programming,” *arXiv preprint arXiv:1808.07292*, 2018.

[106] O. Kilinc and I. Uysal, “Learning latent representations in neural networks for clustering through pseudo supervision and graph-based activity regularization,” in *International Conference on Learning Representations*, 2018.

[107] Y. Tao, K. Takagi, and K. Nakata, “Clustering-friendly representation learning via instance discrimination and feature decorrelation,” in *International Conference on Learning Representations*, 2020.

[108] K. Greff, S. Van Steenkiste, and J. Schmidhuber, “Neural expectation maximization,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6691–6701.

[109] W. Van Gansbeke, S. Vandenhende, S. Georgoulis, M. Proesmans, and L. Van Gool, "Scan: Learning to classify images without labels," in *European Conference on Computer Vision*. Springer, 2020, pp. 268–285.

[110] T. W. Tsai, C. Li, and J. Zhu, "Mice: Mixture of contrastive experts for unsupervised image clustering," in *International Conference on Learning Representations*, 2020.

[111] R. E. Shiffler, "Maximum z scores and outliers," *The American Statistician*, vol. 42, no. 1, pp. 79–80, 1988.

[112] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[113] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[114] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[115] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research (MLR)*, vol. 9, pp. 2579–2605, 2008.

[116] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[117] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[118] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering," in *Advances in Neural Information Processing Systems*, 2005, pp. 1601–1608.

[119] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized cartesian k-means," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 180–192, 2014.

[120] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5147–5156.

[121] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," in *Advances in Neural Information Processing Systems*, 2004, pp. 41–48.

[122] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, 2007, pp. 153–160.

[123] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[124] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of Machine Learning Research (MLR)*, vol. 11, no. 12, 2010.

[125] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[126] J. Zhao, M. Mathieu, R. Goroshin, and Y. Lecun, "Stacked what-where auto-encoders," *arXiv preprint arXiv:1506.02351*, 2015.

[127] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[128] A. Kosiorek, S. Sabour, Y. W. Teh, and G. E. Hinton, “Stacked capsule autoencoders,” in *Advances in Neural Information Processing Systems*, 2019, pp. 15 512–15 522.

[129] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[130] S. Park, S. Han, S. Kim, D. Kim, S. Park, S. Hong, and M. Cha, “Improving unsupervised image clustering with robust learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 278–12 287.