Embedding Convolutional Mixture of Experts into Deep Neural Networks for Computer Vision Tasks

Master Thesis

by

Lukas Struppek

Degree Course: Industrial Engineering and Management M.Sc.
Matriculation Number: 1954638

Institute of Applied Informatics and Formal
Description Methods (AIFB)
KIT Department of Economics and Management
&
FZI Research Center for Information Technology

Advisor:	Prof. Dr.-Ing. Johann Marius Zöllner
Second Advisor:	Prof. Dr. Andreas Oberweis
Supervisors:	Dipl.-Math. oec. Christian Hubschneider
	M.Sc. Svetlana Pavlitskaya
Submitted:	November 26, 2020

Embedding Convolutional Mixture of Experts into Deep Neural Networks for Computer Vision Tasks

by
Lukas Struppek

Master Thesis
November 2020

FZI

Master Thesis, FZI
Department of Economics and Management, 2020
Reviewers: Prof. Dr. J. M. Zöllner, Prof. Dr. A. Oberweis

Department Technical Cognitive Systems
FZI Research Center for Information Technology

Assertion

I hereby declare that I produced this thesis without external assistance, and that no other than the listed references have been used as sources of information. Passages taken literally or analogously from published or non published sources is marked as this.

Karlsruhe, November 26, 2020 Lukas Struppek

Abstract

In many real situations, it is beneficial to seek the opinion of different experts and combine their expertise. One approach that the machine learning community introduced to adopt this idea is the Mixture of Experts (MoE) concept. It relies on several expert networks trained in different domains or tasks. Then a meaningful combination of their predictions is learned by a separate gating network. However, traditional MoE approaches have a few drawbacks: no end-to-end training is possible, and it is also dependent on suitable datasets to achieve individual expert specializations. Furthermore, the utilization of distinct experts varies considerably and often results in only a small number of experts being considered.

We tackle these drawbacks by generic MoE layers embedded into various convolutional neural network architectures. This thesis extends previous works on MoE layers to computer vision applications and investigates the impact on image classification and object detection. We further present different constraint-based approaches to enable a stable training process concerning expert utilization.

Our results for models with embedded MoE layers match the performance of vanilla networks and, in some cases, even exceed those. Additionally, we demonstrate that even with a single dataset and end-to-end training, experts can implicitly focus on individual sub-domains of the input space. Thereby, additional insights into the decision-making process within neural networks are provided.

Zusammenfassung

In vielen realen Situationen ist es von Vorteil, die Meinung verschiedener Experten einzuholen und deren Erfahrungen zu kombinieren. Diese Idee wurde von der Machine Learning Community unter anderem in Form des Mixture of Experts (MoE) Konzepts umgesetzt. Dieses basiert auf einer Menge an Experten, die jeweils für verschiedene Bereiche oder Aufgaben trainiert werden. Anschließend wird ein sogenanntes Gating Network eingesetzt, welches eine sinnvolle Kombination der Vorhersagen der Experten lernt. Traditionelle MoE-Ansätze besitzen allerdings einige Nachteile: So ist kein Ende-zu-Ende-Lernen der Architekturen möglich und weiterhin werden passende Datensätze benötigt, um eine Spezialisierung der individuellen Experten zu gewährleisten. Außerdem schwankt die Inanspruchnahme der einzelnen Experten stark und führt häufig dazu, dass nur eine kleine Anzahl an Experten während der Entscheidungsfindung überhaupt beachtet wird.

Diese Arbeit löst die genannten Probleme durch den Einsatz von generischen MoE-Schichten, die in verschiedenen Varianten von Convolutional Neural Networks eingebettet werden. Dabei werden bestehende Arbeiten für MoE-Schichten um Anwendungen im Bildverarbeitungsbereich erweitert und speziell der Einfluss auf Bildklassifizierung und Objekterkennung untersucht. Darüber hinaus werden verschiedene auf Restriktionen basierende Ansätze vorgestellt, um stabile Trainingsprozesse hinsichtlich der Verwendung von Experten zu erreichen.

Die untersuchten Modelle mit eingesetzten MoE-Schichten liefern eine vergleichbare Performance wie die zugrundeliegenden Neuronalen Netze und übertreffen diese in einzelnen Fällen sogar. Es wird gezeigt, dass selbst mit einem einzelnen Datensatz und Ende-zu-Ende-Lernen eine Spezialisierung der einzelnen Experten auf individuelle Teilbereiche aus dem Input-Space möglich ist. Hierdurch werden zusätzliche Einblicke in den Entscheidungsprozess innerhalb von Neuronalen Netzen ermöglicht.

Contents

Abbreviations

AP Mean Average Precision

AR Mean Average Recall

BBox Bounding Box

CNN Convolutional Neural Network

COCO Common Objects in Context

EM Expectation-Maximization

FPN Feature Pyramid Network

IoU Intersection-over-Union

KL divergence Kullback–Leibler divergence

MAC Multiply-Accumulate operation

MoE Mixture of Experts

MoE layer Sparsely-Gated Mixture-of-Experts Layer

NLP Natural Language Processing

NMS Non-Maximum Suppression

PCA Principal Component Analysis

ReLU Rectified Linear Unit

ResNet Residual Neural Network

SSD Single Shot Detector

SVM Support Vector Machine

t-SNE t-distributed Stochastic Neighbor Embedding

YOLO You Only Look Once

List of Symbols

General Symbols and Statistics

A	Number of anchor boxes
Acc	Model accuracy
C	Number of classes
$cov(X, Y)$	Covariance between variables X and Y
CV	Coefficient of variation
μ	Arithmetic mean
$p_{j\|i}$	Conditional probability of high-dimensional data points in t-SNE
p_{ij}	Joint probability of high-dimensional data points in t-SNE
q_{ij}	Joint probability of low-dimensional data points in t-SNE
$R(x_i)$	Rank of sample x_i
$\bar{R}(x)$	Arithmetic mean of ranks $R(x_1), ..., R(x_n)$
$\rho_{X,Y}$	Pearson correlation coefficient
ρ_S	Spearman's rank correlation coefficient
σ	Standard deviation
t_i	Parameterized coordinates of a BBox prediction
t_i^*	Parameterized coordinates of a ground-truth BBox
X	Batch
$\|X\|$	Batch size of batch X

Mixture of Experts

E	Set of experts $E_1, ..., E_N$
$e_i(x)$	Output of expert E_i for input x
$F_{MoE}(x)$	Output of an Mixture of Experts model for input x
G	Gating network
$G(x)$	Weight vector computed by gating network G for input x
$g_i(x)$	Weight for expert E_i computed by gating network G for input x
$\bar{g}_i$	Averagely assigned sparse weights to expert E_i during evaluation
$H(x)$	Raw logits of gating network G for input x
$H^{Noise}(x)$	Noisy logits of gating network G for input x
k	Number of activated experts in an MoE layer
k_{reg}	Number of activated experts in a regressor MoE layer
k_{cls}	Number of activated experts in a classifier MoE layer
N	Number of experts in set E
$p(c\|E_i, x)$	Probability of class c given expert E_i and input x
$p(E_i\|x)$	Probability to select expert E_i given input x

Loss Functions and Constraints

$D_{KL}(P \parallel Q)$	Kullback–Leibler divergence from Q to P
$I(X)$	Importance vector for batch X
$I_i(X)$	Importance of expert Ei for batch X
$I_i^{rel}(t)$	Relative importance of expert E_i for batch X_t
$\bar{I}_i(t)$	Mean importance of expert E_i for batch X_t
$\mathcal{L}_{balanced\,focal}$	α-balanced Focal loss
$\mathcal{L}_{ce}$	Categorical cross-entropy loss
$\mathcal{L}_{focal}$	Focal loss
$\mathcal{L}_{importance}$	Importance loss
$\mathcal{L}_{kl}$	Kullback–Leibler divergence loss
$\mathcal{L}_{reg}$	Smooth L_1 loss
$\mathcal{L}_{var}$	Variance loss
m_{mean}	Threshold for mean importance constraint
m_{rel}	Threshold for relative importance constraint
$W_i(t)$	Sum of weights assigned to expert E_i up to time step t
$\bar{W}(t)$	Average sum of weights $W_i(t)$ for all experts in E
w_{aux}	Weighting factor of auxiliary losses
$w_{importance}$	Weighting factor of $\mathcal{L}_{importance}$
w_{kl}	Weighting factor of $\mathcal{L}_{kl}$
w_{var}	Weighting factor of $\mathcal{L}_{var}$

1 Introduction

In many areas of our modern society, we rely on experts specialized in distinct fields. For example, hospitals have medical specialists for anesthesiology, ophthalmology, surgery, and many other fields. Each of these experts has a general medical background and additional domain-specific knowledge. Depending on the case, one or many of these experts are asked for advice. In some cases, a single expert might be sufficient to make a reliable diagnosis. In more complex cases, a group of experts is consulted to determine the treatment. Even if several people are involved in the decision, some experts' opinions have more weight than others since more competence is attributed to them. Comparable expert systems can be found in companies, universities, ministries, and many more areas. Focusing on specialists instead of individual generalists has proven its worth in the past.

The machine learning community adapted the idea of expert systems in the form of Mixture of Experts (MoE) architectures in the early 1990s. These architectures rely on a set of expert models, each previously trained in a distinct domain or for a specific task. An additional model, the gating network, then decides input-dependent to what extent it relies on the individual experts' predictions. Traditional MoE architectures using neural networks are trained in two steps: we first train each distinct expert model, and then the gating network to combine the predictions. End-to-end training of the entire architecture is not directly applicable. We also have to provide different data sets for the experts, either manually prepared or using clustering algorithms. Besides, all experts have to perform partially redundant steps to extract information without a shared feature extractor. This costs valuable computing time, which is especially important in real-time applications such as autonomous driving or robotics. Furthermore, MoE architectures often suffer from under-utilization of distinct experts and a fixation on a small expert subset.

To overcome these problems, we pursue an approach to embed the idea into generic neural network layers. We insert these MoE layers into Convolutional Neural Networks (CNNs) and apply the approach to the computer vision tasks of image classification and object detection. All our models with embeddings are trained end-to-end with standard optimizers and a single training set. We present various soft and hard constraints to tackle the problem with uneven expert utilization during training. The models further provide a hyperparameter to adjust the number of active experts in each forward pass and, thereby, the computational complexity. It allows us to train models with large capacity but, at the same time, maintain a beneficial computational complexity. Besides, embedded MoE layers enable us to interpret and gain insights into the neural networks' decision-making process.

1.1 Overview of Contents

Chapter 2 introduces the theoretical foundations for Mixture of Expert (MoE) models. Besides traditional MoE approaches, we introduce embedded MoE layers as a realization of conditional computation. We further explain our approaches for balancing expert utilization with soft and hard constraints.

Various network architectures for image classification and object detection are presented in chapter 3. Our applications rely on Residual Neural Networks (ResNets) for image classification and RetinaNet for object detection. We also describe famous one-stage and two-stage architectures for object detection, namely R-CNN, SSD, and YOLO.

We start our experimental studies in chapter 4 with a few traditional MoE architectures for image classification on MNIST. Simple CNNs are trained in various dataset split settings and investigate different gating network and label decoding approaches. These experiments are proofs-of-concept and demonstrate some drawbacks of traditional MoE architectures.

In chapter 5, we embed MoE layers into Residual Neural Networks and train our models for image classification on CIFAR-100. Large ResNet blocks are replaced with MoE layers using deep experts. We narrow down the expert networks to maintain a comparable computational complexity with the vanilla model. Furthermore, we conduct an extensive analysis of the expert utilization, sample assignment process, and domain specialization.

Chapter 6 extends the application of embedded MoE layers to object detection with RetinaNet and the COCO dataset. Our focus lies on the behavior of MoE in the final network parts responsible for bounding box regression and object classification. We investigate the MoE process visually by comparing the bounding box predictions of distinct experts with the combined MoE prediction. We further analyze the model's detection behavior quantitatively for different feature map levels and distinct experts. Experiments with pre-trained experts and embeddings in the feature extractor complete our object detection applications.

A summary in chapter 7 concludes this thesis with our main findings. We demonstrate the importance of our work and highlight the most meaningful results. Furthermore, open topics in the field of embedded MoE layers are pointed out for future research.

2 Theoretical Foundations

We introduce in this chapter the theoretical foundations on which we base our work. We first explain the traditional Mixture of Experts (MoE) concept in section 2.1. Besides its basic properties, we compare MoE to other ensemble learning algorithms and present an overview of further developments and modifications. In section 2.2, we describe the concept of conditional computation. This concept characterizes models that activate only a part of their architecture during inference. At the same time, these models maintain a much larger capacity in terms of learned parameters. Conditional computation enables a beneficial ratio of model capacity to computational complexity and allows us to build deep neural networks with reduced computational costs.

We then introduce in section 2.3 the embedded Sparsely-Gated Mixture-of-Experts Layers (MoE layers) as a realization of conditional computation. These layers integrate the MoE concept into a generic layer and can be used in various types of neural networks with end-to-end training. Most of our experiments in this thesis are based on this architecture and develop it further. Finally, we present in section 2.4 different approaches to balance expert utilization. MoE architectures suffer from a problem to which we refer to as dying experts. During training MoE architectures, the models often utilize a few experts significantly more than others. The utilization of some experts may completely stay off, and the effective number of experts decreases. We introduce different soft and hard training constraints to overcome the dying experts problem.

2.1 Mixture of Experts

Jacobs et al. [1] introduced Mixture of Experts models in 1991. The authors address multi-task learning problems and argue that training a single multilayer network simultaneously on different tasks produces strong interference effects that negatively affect learning and generalization. To overcome this problem, they suggest training independent *expert networks* (experts) on datasets from different tasks or input space regions. An additional model, the *gating network* (gate), then combines the expert models and selects the most useful models for the current use-case.

Classic MoE models [1] consist of a variable number of expert models and a single gating network to combine them. At first, the expert models are trained separately on different subsets. In a second step, a gating network is trained to combine the outputs of the experts. Separating experts' training on different tasks or datasets can reduce interference effects between them.

From an algorithmic view, an MoE architecture is a divide-and-conquer algorithm. Each expert solves a subproblem, and the gating network combines their solutions to solve the initial problem [2]. In an efficient MoE model, the gating network should implicitly identify experts that are specialized in the current task or domain and produce more accurate predictions. Figure 1 illustrates a simplified MoE for classification with two experts. Each expert makes perfect predictions in one subspace but lacks in the remaining input space. The gating network divides the input space into two subspaces and assigns each expert to one of them. This example is highly simplified, and experts are responsible for distinct input spaces. In more advanced MoE architectures, the gating network may rely on more than one expert's prediction.

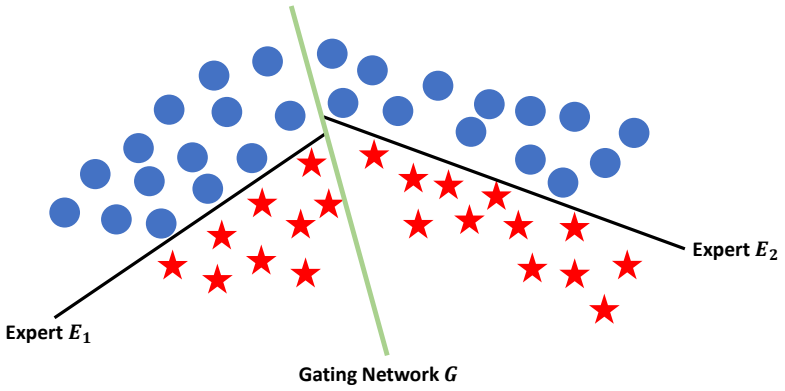


Figure 1: A simple MoE classification example with two linear experts E_1 and E_2 and a gating network G. Each expert makes perfect predictions in a distinct part of the input space but lacks in the remaining space. The gating network identifies the subspaces in which each expert performs best and partitions the input space accordingly. Consequently, expert E_1 is responsible for the left space and expert E_2 for the right. Note that the gating network assigns all weights to one specific expert in this simplified example. The representation is inspired by [3].

Figure 2 shows a classic MoE architecture. In its basic version for classification, an MoE model consists of a set of expert networks E with size $|E| = N$ that contains expert models $E_1, ..., E_N$. For a given input x, each expert E_i produces an output $e_i(x)$ that maps x to one of C classes. A gating network G computes a weight vector $G(x)$ with length N for the set of experts E. Note that inputs of experts and gating network may differ. For simplicity, we assume that both inputs are identical. The model applies a softmax function to turn the gating network's logits $H(x)$ into weights $g_i(x), ..., g_N(x)$ for N experts. The softmax function is defined as $softmax(z)_i = \frac{exp(z_i)}{\sum_{j=1}^{n} exp(z_j)}$ for $i = 1, ..., n$ [4]. Because we compute $G(x) = softmax(H(x))$ by a softmax function, the sum of weights amounts to $\sum_{i=1}^{N} g_i(x) = 1$. The final MoE output is computed by a convex combination of the experts' outputs, according to Equation 2.1. MoE approaches for regression problems are practically identical with differences in the expert outputs [1, 5].

$$F_{MoE}(x) = \sum_{i=1}^{N} g_i(x) e_i(x) \tag{2.1}$$

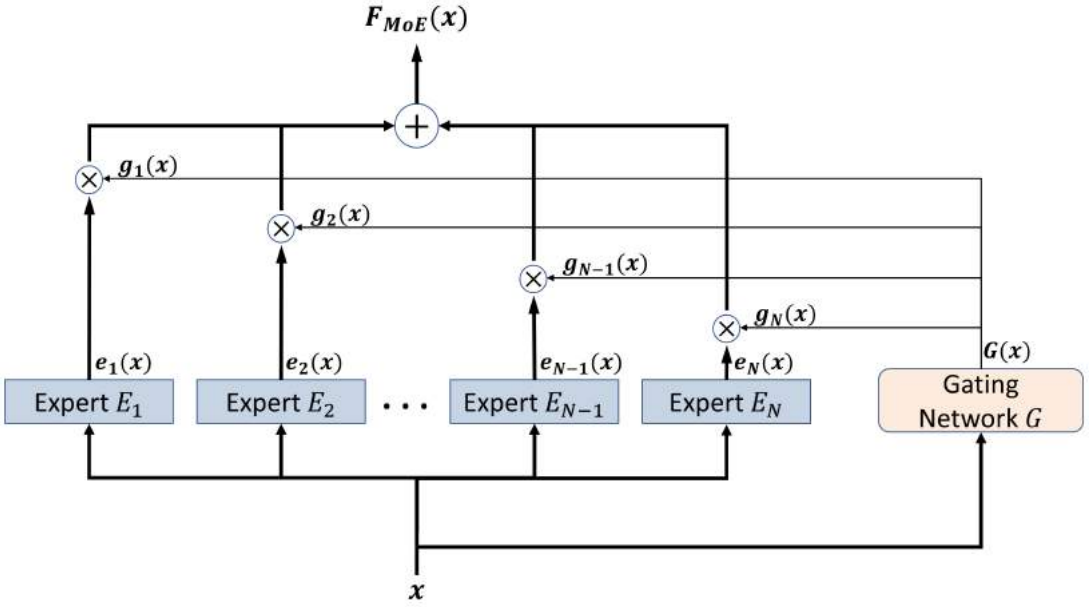


Figure 2: Traditional Mixture of Experts (MoE) architecture. Input x is processed by a set of experts E. Each expert receives the same input x. Gating network G computes a weight vector $G(x)$ using a softmax function for normalizing its outputs. Expert outputs $e_i(x)$ are then multiplied by corresponding weights $g_i(x)$. The weighted outputs are finally summed up to $F_{MoE}(x)$. The representation is inspired by [1].

To measure the utilization of experts, we define an importance vector $I(X) = \sum_{x \in X} G(x)$ for each batch of training samples X, and the importance of a single expert E_i as $I_i(X) = \sum_{x \in X} g_i(x)$ [6]. Therefore, the importance corresponds to the sum of weights assigned to a particular expert E_i for a single batch. Importance helps to compare the utilization of different experts. We use it to compute auxiliary loss functions and constraints (see section 2.4) and analyze the weight assignment process during inference.

We can also interpret an MoE model as a probability model in which class probabilities are marginalized over experts' selection. Each weight $g_i(x)$ can be seen as the probability $p(E_i|x)$ to select a specific expert for a given input. Outputs of each expert quantify the probability $p(c|E_i, x)$ of each class $c \in C$. We can rewrite Equation 2.1 to compute a model's output as in Equation 2.2 by setting $g_i(x) = p(E_i|x)$ and $e_i(x) = p(c|E_i, x)$ [5]:

$$F_{MoE}(x) = \sum_{i=1}^{N} p(E_i|x)p(c|E_i, x) = p(c|x) \tag{2.2}$$

During training, the gating network allocates samples to the different experts. If their outputs are incorrect, the gate decreases their weighting and vice versa. In this process, experts' weightings are decoupled from each other, and no direct interference between them occurs. Still, some indirect coupling exists when the gating network changes weightings for different experts. A gating network may alter one expert's responsibility due to others' behavior [1].

We can train MoE models using an Expectation-Maximization (EM) algorithm or a gradient-based optimizer [7]. While earlier work focuses on training by EM (e.g. [2], [8]), recent work applies standard optimizers for neural networks (e.g. [6], [9]). In this thesis, we investigate the behavior of deep neural networks. All of our models are trained with the gradient-based Adam optimizer [10].

MoE architectures are a variant of ensemble learning that refers to combinations of multiple learning algorithms or models to improve the overall performance. Stacked generalization [11], as an ensemble learning algorithm, is similar to an MoE. In a stacked generalization setting, we first train a set of base learners on a shared training set. In a second step, we fit a meta-learning model that uses the base learners' predictions as input and tries to learn their outputs' best aggregation. Unlike MoE, a stacked generalization model does not explicitly learn how to weight the base learners' outputs, but aggregation takes place in the meta-learning model itself.

Another similar approach is ensemble averaging [12] that takes outputs of different models and averages them. Accordingly, it is like an MoE model with N experts and the gating network replaced by a fixed weighting vector $g_i = \frac{1}{N}$. Unlike an MoE, ensemble averaging approaches cannot efficiently benefit from specialized models since specific experts are fading between other models with an increasing number of experts.

Since their introduction in 1991, many MoE architecture variants have been published, demonstrating the basic concept's high variability. Apart from simple linear models ([1]), other expert architectures like SVMs ([13], [14]), Gaussian processes ([15], [16]), or deep CNNs ([17], [18]) are presented. One approach introduces a tree-structured hierarchical mixture of experts, similar to a decision tree [2]. Another work extends the model by a gating network for an infinite number of experts [19].

Recent approaches reuse traditional MoE architectures to gain insights into the decision making process in neural networks [20]. The authors argue that we can achieve additional interpretability while maintaining comparable prediction quality by analyzing the agreement and disagreement between distinct expert networks and the combined MoE output. Note that these additional insights come at the costs of running large expert networks at the same time. Real-time applications, as needed for self-driving cars, are hard to realize with this approach. Our work and analysis are inspired by that work and extend their findings.

2.2 Conditional Computation

The basic idea of conditional computation in machine learning is to activate only a part of a model depending on its current input. It reduces computational complexity and speeds up inference. Decision trees like CART [21] implement such behavior naturally since only a single subtree is activated during inference, while most parts of the model remain inactive. A decision tree with binary splits and a depth of N computes $O(N)$ nodes out of a pool of up to $O(2^N)$ nodes during inference. Consequently, the ratio of model capacity in terms of the number of parameters to computation is beneficial, and model expansions do not negatively affect inference time as long as the tree depth stays constant [22].

For neural networks, no natural conditional computation exists. In a classic deep learning model, each part of the network is activated for each input resulting in a capacity to computation ratio of one. While techniques like dropout [23] randomly deactivate parts of a model, in conditional computation, the deactivation is non-stochastic and learned during training time. Bengio proposed in 2013 an approach of conditional computation in neural networks to tackle the problem of scaling computations [24]. The work of [25] expands that idea further.

Conditional computation promises various advantages. It speeds up information propagation through neural networks and reduces training and inference time. This enables an increase of network capacity in terms of parameters and maintains a comparable runtime complexity to a model with full network utilization. Additionally, conditional computation produces sharper gradients for the activated parts of a network, making optimization faster and easier [24]. It also acts as a form of regularization since only a fraction of the network is available at each forward pass [26]. We show a simple, theoretical example for conditional computation in a fully-connected neural network in Figure 3: two neurons are inactive during the forward pass but may be activated for another model input. Consequently, the model performs fewer mathematical operations, and computational costs decrease.

We measure the computational complexity in terms of Multiply-Accumulate operations (MACs) required to process a specific network input. Each MAC operation computes the product of two numbers b and c and then adds the result to an accumulator a. A MAC operation corresponds mathematically to $a \leftarrow a + (b \cdot c)$. Since our networks are relatively large, we state complexity in Giga MAC (GMAC). MACs allow a hardware-independent analysis of computational costs for different model architectures.

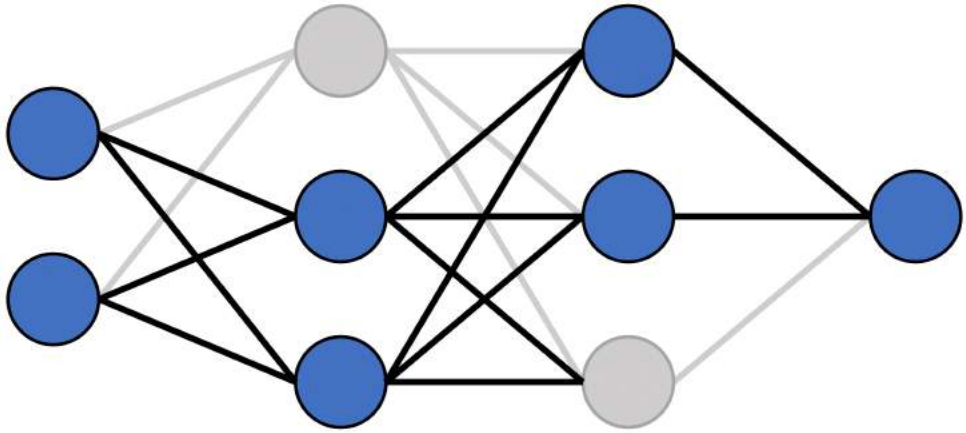


Figure 3: Theoretical example of conditional computation in neural networks. Only a part of the network's neurons is activated (blue) in each forward pass, so the number of computations decreases. Simultaneously, the network's total capacity stays the same since it still contains the inactive neurons and their weights. A separate mechanism decides input-dependent which neurons to activate during inference.

In an MoE context, we can achieve conditional computation by deactivating some experts according to the gating network's output. Due to multiplicative connections between experts and the gating network's weightings, zero-weighted experts' predictions do not influence the final output. Therefore, we can omit the activation of these experts during the current inference. This gating process is comparable to the selection of a subtree in a decision node in decision trees. However, in MoE architectures, more than one expert should be activated to enable a combination of different experts and avoid limited local generalization [24].

Various practical approaches for conditional computation based on MoE have been proposed in recent years. They all follow the idea to increase a model's capacity by maintaining its computational cost. An early approach [26] uses reinforcement learning to optimize conditional computation policies in a neural network. Their model combines fully connected layers with stochastic per-layer policies that activate or deactivate units input-dependent.

[27] proposes a ResNet-based network architecture that dynamically chooses the number of residual units and implements an input-dependent early stopping mechanism. During each forward pass, the model computes a halting score for each residual unit. After the cumulative sum of this halting score reaches a threshold, all following units are skipped.

Dynamic Deep Neural Networks (D^2NN), introduced by [28], augment feed-forward neural networks with control modules. Each control module is a subnetwork that decides which other modules to execute depending on the input. It improves computational efficiency since the model skips unnecessary computations. D^2NNs also allow to set and restrict computation budgets for large networks. In each forward pass, the network only executes a subset of its available neurons.

[9] explore a deep dynamic routing approach for CNNs. A multi-headed gating network selects and re-weights the channels in each convolutional layer. The authors aim to build deep neural networks in a sparsely-gated MoE model. Their work is based on [6] that proposes a sparsely-gated mixture-of-experts layer that only activates a predefined number of experts in each forward pass. They demonstrated a 1000x increase in model capacity while maintaining computational efficiency with only minor losses. The following section gives a more detailed introduction to this approach on which our work is mainly based.

2.3 Embedded Mixture of Experts Layers

In 2017, Shazeer et al. from Google Brain published the generic Sparsely-Gated Mixture-of-Experts Layer (MoE layer) [6]. Their work combines the ideas behind MoEs and conditional computation and integrates the MoE concept into a generic layer for neural networks. The authors point out that no earlier work demonstrates massive practical improvements in capacity, quality, or training time in neural networks through conditional computation. However, their models with MoE layers achieve remarkable results on language modeling and translation datasets with substantial model capacity improvements and minor computational efficiency losses.

As classic MoE architectures, the proposed MoE layer consists of a variable number of experts and a gating network to select a sparse combination of the experts. Unlike classic MoE approaches, embedded MoE layers form only a part of the entire model and are trained end-to-end with other network parts by backpropagation. Figure 4 illustrates the basic concept of an MoE layer with N experts, of which the gating network selects and activates two experts for each forward pass. It then convexly combines the experts' outputs to produce the MoE layer's output.

The MoE layer activates experts only if their weights $g_i(x)$ are non-zero to reduce computational complexity. It allows the network to have a large number of different experts in one layer to increase the model's capacity and, at the same time, preserve sparse computations.

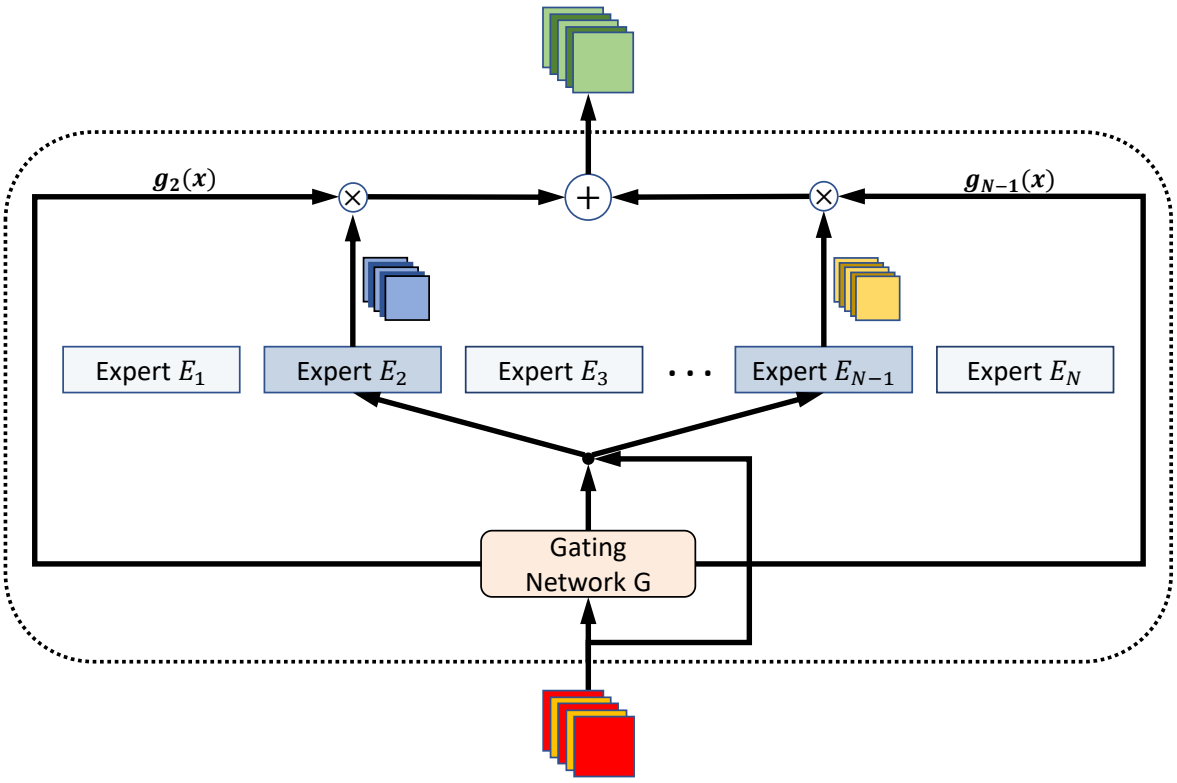


Figure 4: Embedded MoE layer illustrated in a convolutional neural network. At first, the gating network computes weights for a set of N experts. It then selects the top k weighted experts and passes the input to them. Unlike classic MoE architecture, only a subset of experts is activated. In this case, the gating network dynamically selects the top $k = 2$ out of N experts. Outputs of active experts are then fused in a convex combination. The representation is inspired by [6].

To further increase sparsity, the MoE layer only activates the top-weighted experts in each step, controlled by hyperparameter k. The model sets all but the top k gating network's logits $H(x)$ to minus infinity, as stated in Equation 2.3, to deactivate other experts. On the resulting vector, the gating network applies a softmax function to compute the weight vector $G(x)$, as in Equation 2.4.

$$KeepTopK(h,k)_i = \begin{cases} h_i & \text{if } h_i \text{ is in the top } k \text{ elements of } h. \\ -\infty & \text{otherwise.} \end{cases} \tag{2.3}$$

$$G(x) = softmax(KeepTopK(H(x),k)) \tag{2.4}$$

The authors also add Gaussian noise at training time as second component to the gating function. According to the authors, the noise supports a smooth estimator of the number of samples assigned to each expert and helps with load balancing. Be $Noise(x)$ the amount of noise computed and learned by an additional single fully connected layer without bias. A $softplus(x) = ln(1 + e^x)$ function is then applied to smoothly approximate a rectifier $f(x) = max(x, 0)$. In this thesis, we train our models only on single GPUs.

Hence, we do not consider load balancing. However, we found the noise term to act as a kind of regularization and be beneficial to overall training stability and expert utilization. The gating network's logits $H(x)$ in Equation 2.4 are therefore replaced by H^{noise}, as stated in Equation 2.5.

$$H^{noise}(x)_i = H(x)_i + Z \cdot softplus(Noise(x)_i), \qquad Z \sim N(0,1) \tag{2.5}$$

In the original paper [6], the authors apply their proposed MoE layer on the Natural Language Processing (NLP) tasks of language modeling and machine translation, embedded in Long Short-Term Memory layers [29]. Other use-cases are not stated and, to the best of our knowledge, not conducted in other research publications. For NLP experiments, experts and gating networks consist of single trainable weight matrices that correspond to single fully connected layers without biases. Due to the sparse gating function, model capacity could be increased massively with only small computational efficiency losses.

The authors train neural networks with up to 137.7 billion parameters, which show significant performance boosts on the benchmarks compared to earlier state-of-the-art architectures. Note that models in their experiments need a large number of experts to achieve comparable results. The models are trained on clusters of 16 to 128 Tesla K40 GPUs. In this thesis, we further develop their approach, adapt it to the computer vision domain, and combine the concept of MoE layers with CNNs. We also examine to what extent we can train models with MoE layers on single-GPU systems.

2.4 Balancing Expert Utilization

A common problem with MoE architectures is that gating networks tend to focus only on a small subset of all available experts. The weights assigned to other experts are permanently zero or negligibly small. The problem arises in neural network MoE architectures from training with gradient descent algorithms. In particular, during the first few training iterations, some experts perform better than others, highly depending on the number of training samples seen. Hence, the gating network increases the probabilities of these experts to be activated. Consequently, these few experts improve above average, and the gating network assigns even higher weights to them. This self-reinforcing process continues, and the optimizer ends up in a local minimum [5].

We not only observe this behavior during the first few training steps but also in later training epochs. We refer to this problem as dying experts, analogous to the dying ReLU problem [30]. We count experts as dead in our experiments if they receive less than 1% average importance on the corresponding test set. Note that average importance is identical to the average weighting an expert receives on a single forward pass.

Other authors also observe the dying experts problem and propose different approaches to address the problem. Eigen et al. [5] propose a hard constraint applied during training on the relative gating assignments to each expert. They sum up the assigned weights to each expert over all training samples $x_{t'}$ as $W_i(t) = \sum_{t'=1}^{t} g_i(x_{t'})$. It is similar to the sum of the expert's importance $\sum_{t'=1}^{t} I_i(X_{t'})$ over all batches $X_{t'}$ until time step t.

Minor differences between both equations arise from weights assigned to samples in the current batch just before the current sample $x_{t'}$. These samples are not considered in the importance equation since importance only takes full batches into account. The authors then compute the average gating assignments $\bar{W}(t) = \frac{1}{N} \sum_{i=1}^{N} W_i(t)$. Weights for expert E_i are set to zero if $W_i(t) - \bar{W}(t) > m$ for a threshold m. The remaining positive weights in vector $G(x_t)$ are then recomputed by a softmax function to maintain a convex combination of the experts.

Based on the work of Eigen et al. [5], we propose two additional hard constraints on importance. Be $I_i^{rel}(t)$ the relative importance of expert E_i for batch X_t at time step t, computed accordingly to Equation 2.6. Note that the mean importance $\mu_{I(X_t)}$ is equivalent to the ratio between batch size $|X_t|$ and the number of experts N.

$$I_i^{rel}(t) = \frac{I_i(X_t) - \mu_{I(X_t)}}{\mu_{I(X_t)}} = \frac{I_i(X_t) - \frac{|X_t|}{N}}{\frac{|X_t|}{N}} \tag{2.6}$$

Our first constraint is based on the running relative importance $\sum_{t'=1}^{t} I_i^{rel}(t')$ of expert E_i at time step t. We set $g_i(x) = 0$ for $x \in X_{t+1}$ if $\sum_{t'=1}^{t} I_i^{rel}(t') > m_{rel}$ for a threshold m_{rel}. Recent values have the most substantial impact on the constraint because we sum up relative importance values. Earlier relative importance values compensate each other or are compensated by exceeding the threshold m_{rel} and deactivating experts. We refer to this constraint as *relative importance*.

The second constraint we propose is based on the mean importance $\bar{I}_i$ for expert E_i, computed following Equation 2.7. Similar to the previous constraint, we set $g_i(x) = 0$ for $x \in X_{t+1}$ if $\bar{I}_i(t) - \mu_{I(X_t)} > m_{mean}$ for a threshold m_{mean}. We refer to this constraint as *mean importance*. Whereas the relative importance constraint takes a stronger focus on the recent past, the absolute importance approach takes a holistic view. All past importance values have the same impact on the constraint.

$$\bar{I}_i(t) = \frac{1}{t} \sum_{t'=1}^{t} \frac{I_i(X_t)}{|X_t|} \tag{2.7}$$

Both hard constraints are only active during training and deactivate experts for an entire batch. This fact may theoretically result in an unstable expert utilization, constantly exchanging active and inactive experts between batches. However, we observed a stabilization process of expert utilization in practice after a few training iterations and with a decreased learning rate. As we will show in chapter 5, our constraint based on relative importance manages expert utilization more reliable than the mean importance constraint.

Shazeer et al. [6] present a soft constraint approach introducing an auxiliary *importance loss* $\mathcal{L}_{importance}$, following Equation 2.8. The loss function computes the squared coefficient of variation of importance vector $I(X)$ for batch X. A hyperparameter $w_{importance}$ weights the result. The coefficient of variation is defined as $CV = \frac{\sigma}{\mu}$ with mean μ and standard deviation σ. $\mathcal{L}_{importance}$ encourages equal importance for all experts during training. The number of training samples per expert may still vary since importance is used instead of the mean number of samples per expert. We compute σ as the squared sample standard deviation that uses Bessel's correction to avoid a bias in the estimation. The sample standard deviation is generally defined as $\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$.

$$\mathcal{L}_{importance} = w_{importance} \cdot CV\left(I(X)\right)^2 = w_{importance} \cdot \left(\frac{\sigma_{I(X)}}{\mu_{I(X)}}\right)^2 \tag{2.8}$$

In addition, we propose another soft constraint that takes a probabilistic view on the experts' importance. According to Equation 2.2 in section 2.1, we interpret the gating network's output as a discrete probability distribution P with probability $P(E_i|\mathcal{X})$ that expert E_i is selected for input $\mathcal{X}$.[1] In expectation, the gating network should assign each expert E_i out of N experts the same average weighting equal to $\mathbb{E}_X[P(E_i|\mathcal{X})] = \frac{1}{N}$. The expected weight assignment corresponds to a discrete uniform distribution Q with probability $Q(E_i|\mathcal{X}) = Q(E_i) = \frac{1}{N}$.

We define an auxiliary *KL divergence loss* $\mathcal{L}_{kl}$ in Equation 2.9 as the Kullback–Leibler divergence (KL divergence) D_{KL} between P and Q, weighted by hyperparameter w_{kl}. We compute $P(E_i|\mathcal{X} = X) = \frac{I_i(X)}{|X|}$ as the average importance per sample in batch X with batch size $|X|$.

$$\begin{aligned} \mathcal{L}_{kl} = w_{kl} \cdot D_{KL}(P \parallel Q) &= w_{kl} \cdot \sum_{i=1}^{N} P(E_i|X) \cdot ln\left(\frac{P(E_i|X)}{Q(E_i)}\right) \\ &= w_{kl} \cdot \sum_{i=1}^{N} \frac{I_i(X)}{|X|} \cdot ln\left(\frac{I_i(X) \cdot N}{|X|}\right) \end{aligned} \tag{2.9}$$

[1] Note that $\mathcal{X}$ corresponds to a random variable for input x and does not refer to batch X.

We plot the importance and KL divergence auxiliary loss functions for models with two experts in Figure 5 and three experts in Figure 6. We set the batch size $|X| = 1$, vary the weight assigned to expert E_1 as $g_1 \in [0, 1]$, compute the second weight $g_2 = 1 - g_1$, and plot the resulting losses in Figure 5. Similarly, we vary $g_1, g_2 \in [0, 1]$, set $g_3 = 1 - g_1 - g_2$ and plot the resulting losses for models with three experts in Figures 6a (importance loss) and 6b (KL divergence loss). We omit data points to which $g_1 + g_2 > 1$ applies since these are no valid weight assignments. Note that for batch size $|X| = 1$, weight vectors and importance vectors are identical, and $I_i(X) = \frac{I_i(X)}{|X|} = g_i(x), x \in X$. Both loss functions evaluate the share of total assigned importance and are therefore independent of the batch size.

The loss functions take their maximum if a single expert receives all of the weights or importance. The maximum value for the importance loss corresponds to the number of experts N in the model. For the KL divergence loss, the maximum value corresponds to the natural logarithm of the number of experts.

Importance loss assigns higher losses to an unequal importance distribution than the KL divergence loss. The larger the inequality becomes, the more the loss functions diverge from each other. Consequently, importance loss penalizes inequality in importance distribution harder than KL divergence loss. Importance loss would be more useful if we aim to an equal expert utilization. However, we might want a higher variance in the expert utilization for some models but still avoid dying experts. In this case, the KL divergence loss would be the better choice to allow larger deviations from an equal importance distribution.

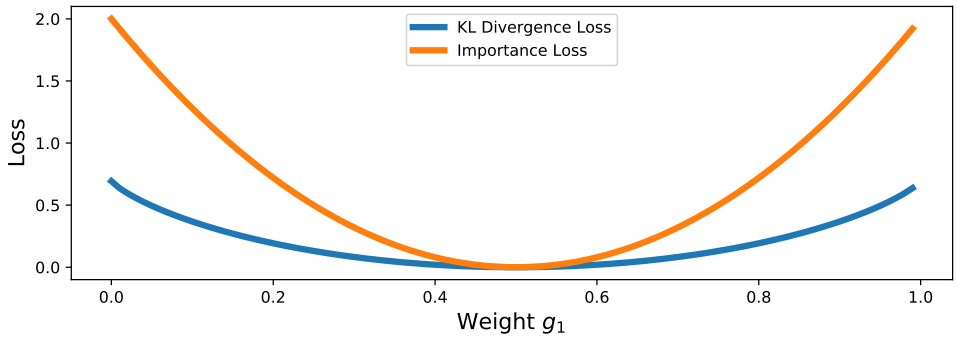
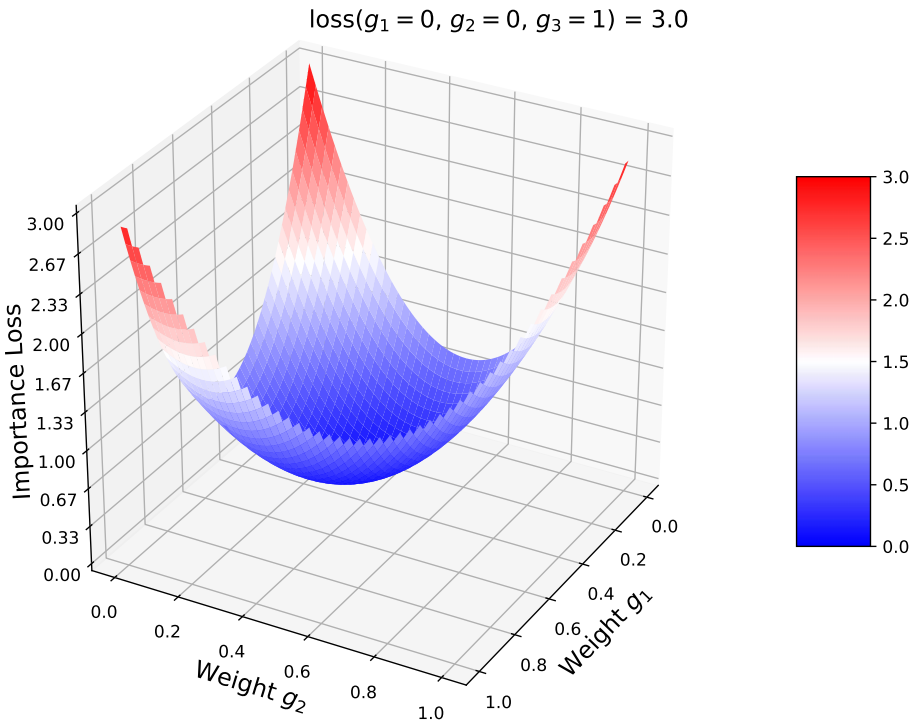
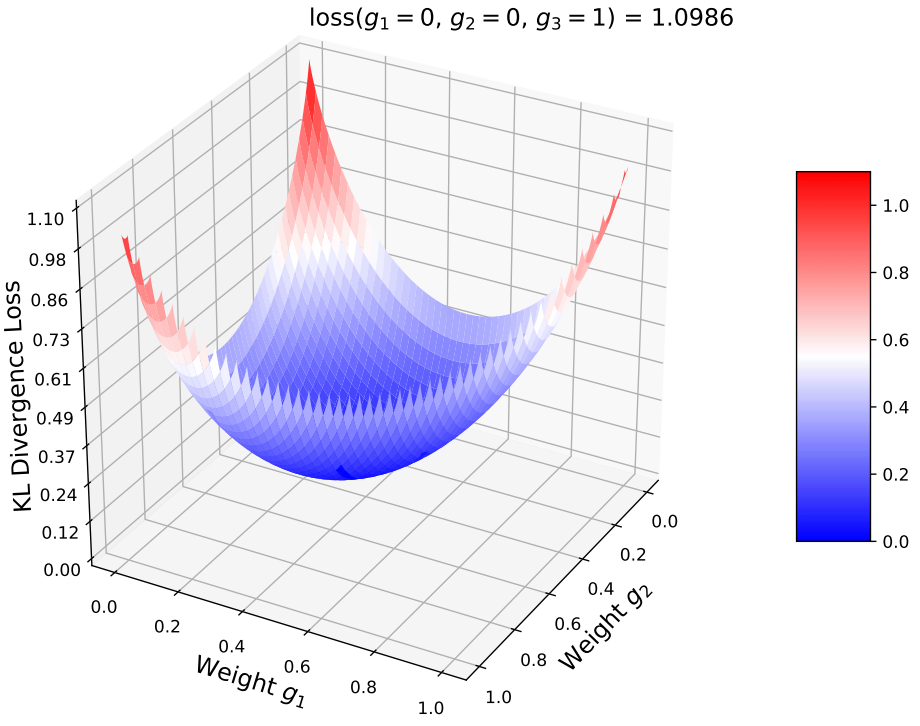


Figure 5: Visualization of importance and KL divergence loss functions for models with two experts. We set the batch size $|X| = 1$, so per-expert importance corresponds to individually assigned weights g_i. Importance loss assigns higher values to larger deviations from an equal expert utilization in terms of assigned importance than KL divergence loss. Figure 6 visualizes the same functions for models with three experts.



(a) Importance loss function $\mathcal{L}_{importance}$ for three experts.



(b) KL divergence loss function $\mathcal{L}_{kl}$ for three experts.

Figure 6: Visualization of importance and KL divergence loss functions for models with three experts. We set the batch size $|X| = 1$, so per-expert importance corresponds to individually assigned weights g_i. Importance loss assigns higher values to larger deviations from an equal expert utilization in terms of assigned importance than KL divergence loss.

3 Architectures for Classification and Detection

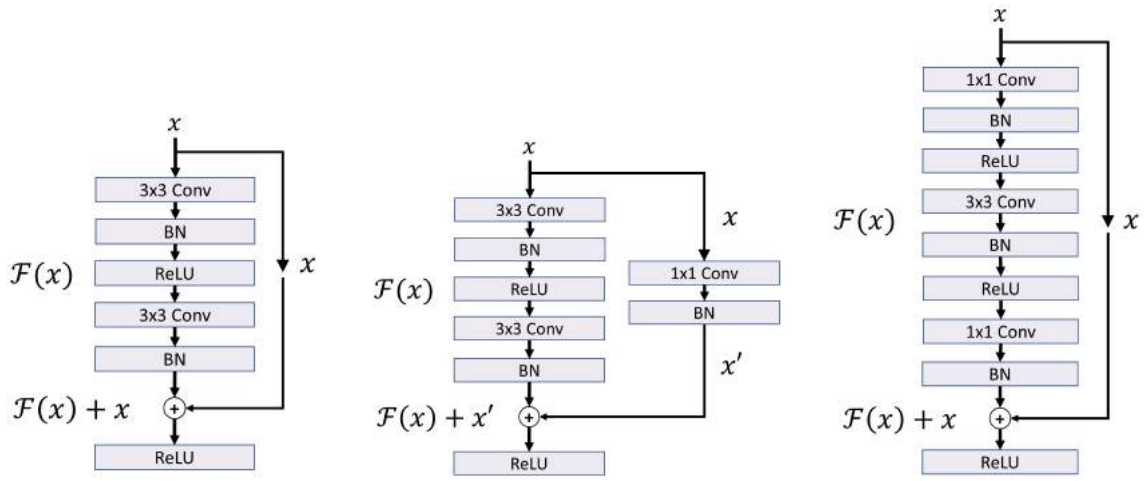
We present in this chapter various common neural network architectures for computer vision tasks. First, we introduce Residual Neural Networks (ResNets) in section 3.1. This kind of neural network uses shortcut connections to ease the learning process of deep neural networks. The basic variants of ResNets and some further developments are introduced. Our classification experiments with embedded MoE layers in chapter 5 rely on these models. We then introduce and compare two classes of object detectors in the following sections. We first take a look into two-stage detectors in section 3.2. Our focus lies on the R-CNN framework and its variations. Two-stage object detectors first generate region proposals and then make predictions based on these proposals.

On the other hand, one-stage object detectors directly process input images and eliminate explicit region proposals to improve inference speed. We explain YOLO and SSD as representative one-stage detectors in section 3.3. Finally, we take a detailed look at the one-stage object detector RetinaNet in section 3.4. RetinaNet has a straightforward network architecture and uses a ResNet backbone as feature extractor. We use RetinaNet in chapter 6 for experiments with embedded MoE layers for object detection.

3.1 Residual Neural Networks

He et al. [31] introduced Residual Neural Networks (ResNets) for image recognition in 2015. Deep CNNs implicitly learn different levels of features that get enriched with increased network depth. Besides common problems with vanishing and exploding gradients, deep neural networks show a degrading behavior when they start to converge. With increasing depth, training accuracy gets saturated and degrades rapidly afterward. The authors show that the problem is not caused by overfitting and even increases by adding more layers to an existing network. He et al. introduce shortcut connections, also called skip connections, to overcome the degradation problem in deep neural networks' training.

The authors consider that a deep neural network should perform at least comparable to an identical shallow neural network with fewer layers. Additional layers may theoretically learn identity mappings and, consequently, produce the same network output as networks with fewer layers. However, the degradation problem suggests that learning a mapping function is not trivial and hard to achieve with current optimizers. The authors add shortcut connections to let layers explicitly learn identity mappings in deeper networks.



(a) Basic building block with identity shortcut.

(b) Basic building block with projection shortcut.

(c) Bottleneck building block with identity shortcut.

Figure 7: ResNet building block variants. 1x1 convolutional layers with stride 2 increase dimensions in 7b. ResNet architectures with size ≥ 50 use bottleneck building blocks, as in 7c. Representations are inspired by [31].

He et al. present ResNet architectures that consist of so-called building blocks. Each building block implements shortcut connections for residual learning. Proposed models combine these blocks in four superior ResNet-blocks with a varying number of building blocks. Figure 7a shows a basic building block. A basic building block consists of two convolutional layers with batch normalization and a Rectified Linear Unit (ReLU) after the first layer and produces output $\mathcal{F}(x)$. By applying a shortcut connection, input x is then added element-wise to $\mathcal{F}(x)$. After that, another ReLU is applied to the sum of x and $\mathcal{F}(x)$.

$\mathcal{H}$ denotes the desired underlying mapping of the building block. Without shortcut connections, the layers have to fit $\mathcal{F}(x) = \mathcal{H}(x) = x$ to learn in identity mapping, resulting in the degradation problem. With shortcut connections, the mapping of the building block is $\mathcal{F}(x) + x$. Therefore, the layers fit a residual mapping $\mathcal{H}(x) - x$, which is easier to optimize for learning an identity mapping because it is easier to fit $\mathcal{F}(x) = 0$ than $\mathcal{F}(x) = x$. Simple shortcut connections add no additional parameters, keep computational complexity constant, and are trained end-to-end with standard optimizers.

Dimensions of x and $\mathcal{F}(x)$ have to be equal to enable element-wise addition. If they differ, we need to up- or downsample projections of shortcut connections. The authors propose projection shortcuts, as shown in Figure 7b, which apply 1×1 convolutions to match dimensions. ResNet architectures with 50 or more layers use bottleneck building blocks, as shown in Figure 7c.

Level	Block	Output size	ResNet-18	ResNet-50
1	-	112×112	7×7, 64, stride 2	
2	1	56×56	3×3 max pool, stride 2	
			$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
3	2	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
4	3	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
5	4	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
Output	-	1×1	global average pool	
			1000-d fully connected	
			softmax	

Table 1: Architecture details of ResNet-18 and ResNet-50. Square bracket blocks correspond to building blocks (see Figure 7). The first convolution layers in blocks 2, 3, and 4 use stride 2 to downsample dimensions. ResNet-18 uses basic building blocks, while ResNet-50 uses bottleneck building blocks. Refer to [31], on which this table is based, for details on other ResNet architectures.

These bottleneck building blocks first reduce the number of channels by 1×1 convolution, apply a 3×3 convolution, and then restore the original number of channels by another 1×1 convolution. The authors use bottleneck building blocks to reduce the computational complexity in deeper networks. We show an overview of ResNet-18 and ResNet-50 architectures in Table 1. This thesis relies on ResNet-18 for image classification and ResNet-50 as RetinaNet backbone for object detection.

In recent years, many high performing image classification networks are based on residual neural networks, for example, ResNeXt [32], DenseNet [33], and ResNeSt [34].[2] The creators of ResNet developed residual networks further into ResNeXt and increase their width through multiple parallel pathways to which they refer to as cardinality. They repeat residual blocks that process the same input in parallel. The model then adds the results element-wise and adds a shortcut connection afterward.

[2]For image classification on ImageNet [35], the models achieve 78.57% (ResNet-152), 80.90% (ResNeXt-101), 77.85% (DenseNet-264) and 84.5% (ResNeSt-269) top-1 accuracy. Since different test sets are used, the results are not directly comparable.

The authors suggest increasing a model's capacity by increasing cardinality is more effective than increasing the networks' depth or layer width [32]. We visualize a simple ResNeXt block in Figure 8. ResNeXt blocks are similar to MoE layers with convolutional expert models. Unlike an MoE, ResNeXt blocks aggregate pathways by element-wise addition and not by a weighted convex combination. ResNeXt also activates all pathways during inference, whereas the MoE layer only uses a sparse selection of experts. ResNeXt follows a similar approach as GoogleLeNet [36] that introduces inception modules. These inception modules also offer multiple parallel pathways, but with varying filter sizes in each pathway.

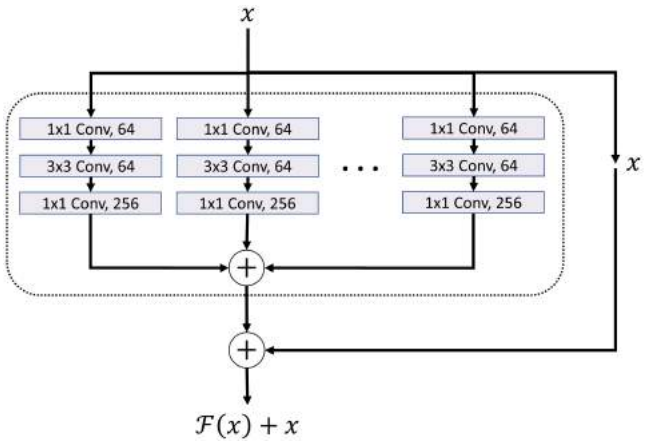


Figure 8: Simple ResNeXt block with multiple parallel pathways. The model sums the results of different pathways together. After that, a shortcut connection adds the input feature map to the aggregation. We do not show activation functions and batch normalization in the image for clarity. The representation is inspired by [32].

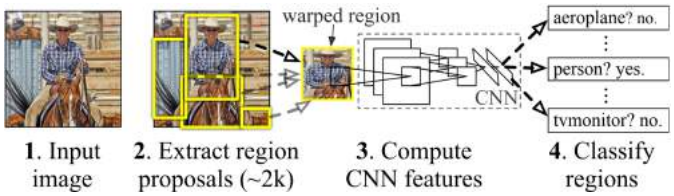
DenseNet extends the concept of shortcut connections and connects each convolutional layer to each subsequent layer. Therefore, each layer uses the outputs of all preceding layers as input and, vice versa, provides its output as input for every subsequent layer. This method allows the net to become narrow as it needs fewer filters and reduces the number of parameters compared to vanilla ResNets. The additional shortcut connections also support a more robust gradient flow. They ease the training of deeper networks and add a regularization effect.

ResNeSt [34] extends the modular ResNet architecture by Split-Attention blocks that enable feature-map attention across different feature-map groups. Each block divides feature-maps along the channel dimension into groups and subgroups. A weighted combination of its splits' representations then determines the feature representation of each group. Within each group, the model computes attention weights across all splits. The model multiplies the feature maps with the corresponding attention weights and adds the results element-wise. ResNeSt is similar to hierarchical MoE architectures [2]. Experts correspond to feature-map groups and expert weightings to attention weights.

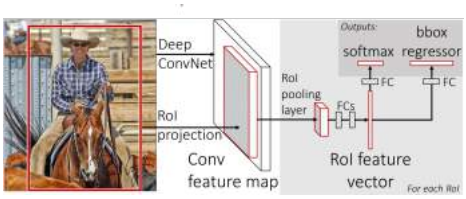
3.2 Two-Stage Object Detectors

Object detection describes the computer vision task in which algorithms try to identify objects in images, locate and annotate them with a surrounding Bounding Box (BBox), and classify each distinct instance. It is a combination of regression for BBoxes and classification of objects. Besides traditional approaches like SIFT [37], HOG [38], or Haar-like features [39], recent approaches focus on the application of deep neural networks. We distinguish between two types of neural network-based detectors: One-stage and two-stage detectors. In this section, we focus on two-stage detectors and introduce one-stage detectors in section 3.3.

Many commonly known two-stage detectors are based on the R-CNN framework [40]. The first stage of these models generates class-agnostic region proposals that likely contain an object. A second stage model distinguishes between objects and background parts for each proposal and predicts the object class and BBox. The original R-CNN framework [40] uses a CNN as feature extractor for each region proposal and Support Vector Machines (SVMs) for classification. An additional regression model refines the BBox predictions of each region proposal. Selective Search [41], which is a bottom-up hierarchical grouping algorithm, generates the region proposals. The procedure starts with class-independent initial regions. In each step, the algorithm merges two neighboring regions with the highest similarity. The process of recalculating similarities and merging repeats until only one single region exists. The object detector then extracts possible object locations generated during the Selective Search.



(a) R-CNN architecture.



(b) Fast R-CNN architecture.

Figure 9: Comparison between R-CNN and Fast R-CNN architectures. R-CNN processes each region proposal separately. The model uses SVMs for classification and an additional regression model for BBox prediction. Fast R-CNN processes each input image once by a deep CNN to generate a feature map. A region of interest pooling layer then computes a feature vector for each region proposal. The model uses fully connected layers for classification instead of SVMs. Both models use Selective Search to generate region proposals. Image sources: [40] (R-CNN), [42] (Fast R-CNN).

Since the processing of about 2,000 region proposals per image separately is computationally expensive and runtime inefficient, R-CNN's authors propose *Fast R-CNN* [42]. The model uses a fully convolutional network to process the entire input image at once instead of each region proposal separately. Figure 9 compares both model architectures.

A region of interest pooling layer extracts a feature vector for each region proposal out of its feature map. Classifier and BBox regressor further process these feature vectors. The authors integrate both parts as fully connected neural networks to enable end-to-end training using a multi-task loss. The main differences between R-CNN and its successor result from the treatment of region proposals and processing of the input image.

Subsequently, [43] introduces *Faster R-CNN* that uses a region proposal network instead of Selective Search to identify region proposals and speed-up inference. The region proposal network uses anchor boxes instead of pyramids of images. We explain the concept of anchor boxes in section 3.2. The whole network, including the region proposal network, is trained end-to-end. *Faster R-CNN* reduces inference time to 200ms per image, compared to the original *R-CNN* with up to 47 seconds per image. Figure 10 visualizes the network architecture, which comes without an additional region proposal algorithm.

Mask R-CNN [44] extends Faster R-CNN and adds a branch for segmentation mask prediction to enable instance segmentation. Two-stage detector models achieve high precision results on standard benchmarks like COCO [45] or VOC [46]. However, a significant drawback of many two-stage detectors is their lack of real-time inference speed and high computational costs due to their complex pipeline. However, real-time inference is necessary for many applications, such as autonomous driving and robotic systems.

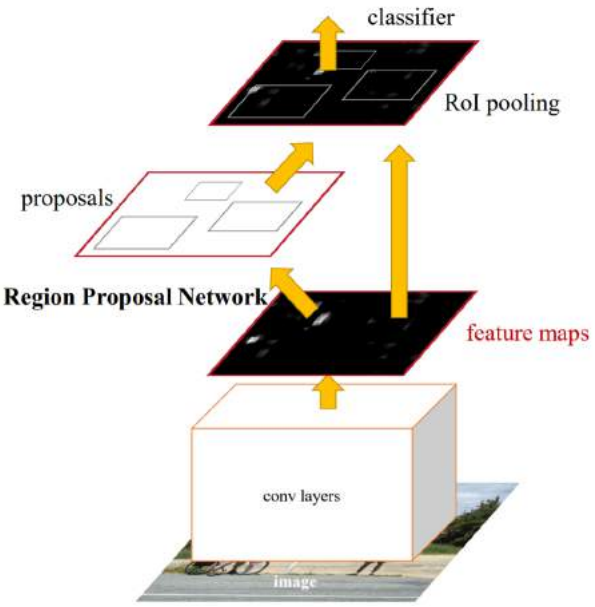


Figure 10: Faster R-CNN architecture uses a single, unified network for object detection. A region proposal network instead of Selective Search computes region proposals. The model reduces the inference time to 200ms per image. Image source: [43].

3.3 One-Stage Object Detectors

One-stage detectors eliminate explicit region proposals and directly process input images using a single convolutional network architecture. Since one-stage detectors process input images only once, they are also called single-shot detectors. Processing an entire image in a single forward pass increases runtime efficiency compared to two-stage approaches. The first well-known one-stage detectors are Single Shot Detector (SSD) [47] and You Only Look Once (YOLO) [48]. Both detectors, except the first version of YOLO, rely on fully convolutional networks using only convolutional layers and do not include any fully connected layers.

SSD and YOLO (v2 and later) rely on the concept of anchor boxes, as introduced by Faster R-CNN [43]. One major challenge for object detectors is predicting absolute BBox coordinates for different kinds of object shapes in images with many objects. Problems arise because a convolutional filter predicts similar absolute coordinates for similar objects independently of their location. This may lead to an unstable training process and inferior prediction quality. Anchor boxes help to stabilize training. They are predefined reference boxes with various aspect ratios to roughly fit most objects' shapes in a dataset. The bounding box regressor then computes a BBox for each anchor box at each position on a feature map. Coordinates are predicted as relative offsets to anchor boxes and mapped to exact positions in the input image. An essential property of anchor boxes is that they are translation-invariant, and the model can apply the same anchor box to different locations. Figure 11 shows an example of three different anchor boxes applied to a specific location. The red anchor box fits the object best while the other anchor boxes differ from the object's shape.

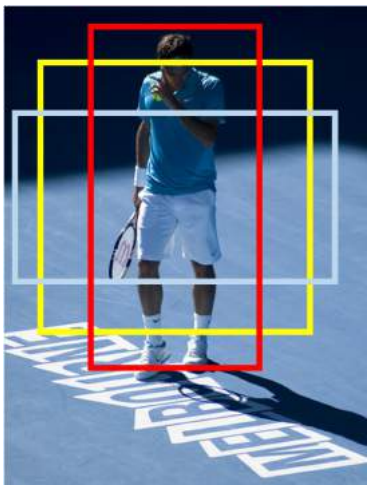
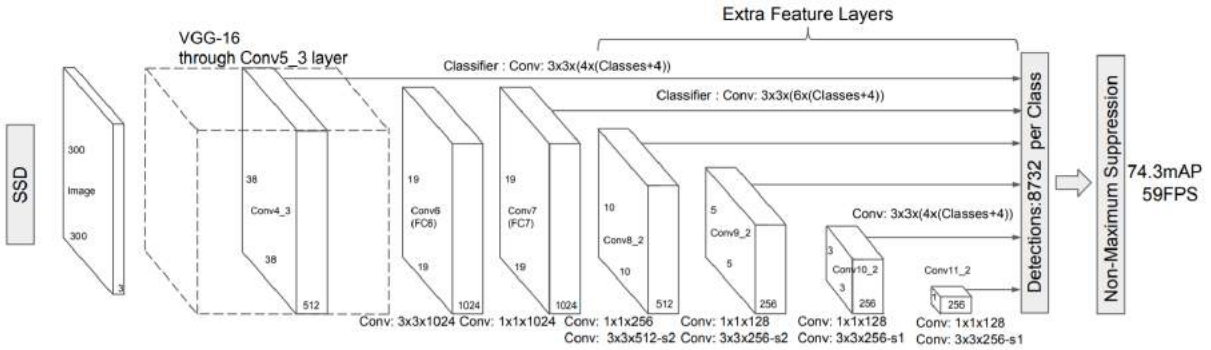
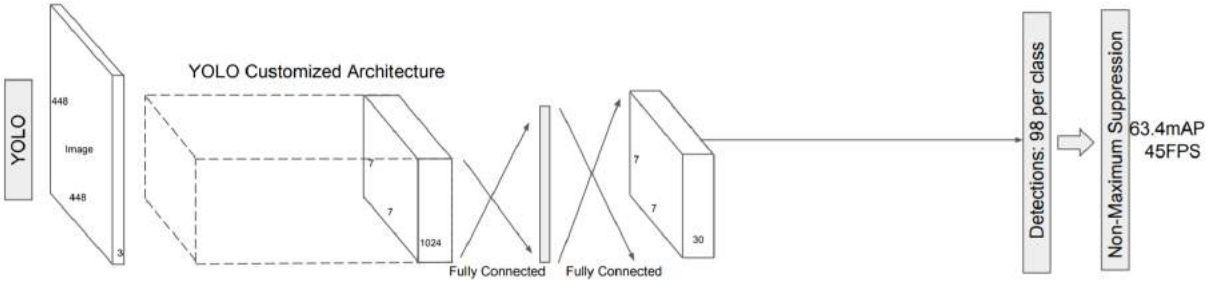


Figure 11: Example of three different anchor boxes with aspect ratios 1:1, 1:2, and 2:1. The red anchor box shows the highest IoU with the ground-truth and is, therefore, matched with the object during training. Raw image source: [45].

Assignment between anchor boxes and ground-truth BBoxes during training is based on Intersection-over-Union (IoU), also known as the Jaccard index. We compute the IoU by $\frac{|A \cap B|}{|A \cup B|}$. It responds to the ratio between the number of elements in the intersection of two sets and the number of elements in the union. For objects in images, the IoU describes how well a predicted BBox overlaps the ground-truth. SSD [47] assigns the anchor box to the ground-truth BBox with the highest IoU to ensure every ground-truth box has a corresponding anchor box. We match each anchor box to a ground-truth if IoU > 0.5 to simplify the learning problem. Otherwise, we assign an anchor to the background class. This enables the model to make predictions on multiple overlapping anchor boxes instead of only the anchor box with the highest IoU.



(a) SSD [47] architecture with a VGG-16 [49] feature extractor. The authors add extra convolutional feature layers to improve the detection of larger objects. Detection is basically done using 3×3 convolutional filters.



(b) YOLOv1 [48] architecture with a modified GoogleLeNet [36] as feature extractor and additional fully connected layers for detection.

Figure 12: Comparison between SSD and YOLOv1 architectures. Mean average precision and inference speed are measured on VOC2007 [46] test data. Input image dimensions are 300×300 for SSD and 448×448 for YOLOv1. Image source: [47].

Figure 12a shows the SSD architecture. The network uses a pre-trained VGG-16 [49] to extract feature maps at multiple scales from input images. The detector then applies small 3×3 convolutional filters for detection. The authors add six convolutional layers to the vanilla VGG-16 model to further decrease feature map sizes. Resulting feature maps with lower resolutions have larger receptive fields and enable detections at larger scales.

SSD applies $(C+4)A$ filters for prediction with C being the number of classes and A the number of anchor boxes. The model computes a score for each distinct class and takes the class with the highest score to predict the current object. SSD consequently produces $(C+4)A \times m \times n$ outputs for an $m \times n$ feature map.

YOLOv1 [48] divides an input image into a 7×7 grid. Each grid cell is responsible for detecting an object if the object's center falls into the grid cell. For each grid cell, the model predicts a predefined number of BBoxes and confidence scores. However, it is only able to detect one object per cell. The confidence score $P(Object) \cdot IoU$ indicates how confident the model is that the predicted BBox contains an object and how precise the BBox captures it. The confidence score is defined as the product of the probability $P(Object)$ that the BBox contains an object, and the IoU between the predicted and ground-truth BBoxes. Each grid cell further predicts for each class a probability $P(c_i|Object)$ that an object in the grid cell belongs to a specific class c_i. YOLOv1 uses a modified GoogleLeNet as feature extractor and adds two fully connected layers for prediction. Figure 12b visualizes the network architecture. Problems in YOLO v1 arise for images with many small objects close to each other since the model can only detect one object per grid cell.

The authors made many improvements over the years to the first version of YOLO. The second version of YOLO [50] introduces batch normalization and anchor boxes, among other modifications, into the model. It uses anchor boxes for each grid cell to make class predictions independent from spatial location and increase the model's recall. Unlike SSD running detection on feature maps of different scales, YOLOv2 uses a passthrough layer similar to projection shortcut connections in ResNets. The passthrough layer concatenates the final feature map with earlier feature maps to detect fine-grained features. YOLOv2 also exchanges its feature extractor for Darknet-19 to speed-up inference.

YOLOv3 [51] incrementally improves its predecessor with small design changes. The model relies on a feature extractor with more layers called Darknet-53. The authors introduce a concept similar to Feature Pyramid Networks (FPNs) (see section 3.4) to predict object BBoxes at three different scales. YOLOv3 also introduces multi-label detection by using independent logistic classifiers instead of a softmax function. Consequently, the model can identify and assign objects with multiple labels, for example, woman and person. Another research team took on YOLO in 2020 and developed YOLOv4 [52]. They introduce various generic CNN improvements into the model and push the evaluation results even further.

3.4 RetinaNet

RetinaNet is a one-stage detector and consists of four components, as shown in Figure 13: a feature extraction network, a Feature Pyramid Networks (FPNs), and two subnets for classification and BBox regression, respectively. We rely on RetinaNet [53] for our object detection experiments in chapter 6 due to its simple and straightforward architecture. Our models use ResNet-50 as feature extractor, consisting of four superior ResNet-blocks using bottleneck building blocks (see section 3.1). The network doubles the number of feature maps in each superior ResNet-block in its bottom-up pathway, while dimensions of each feature map are halved. Refer to Table 1 in section 3.1 for architecture details.

FPNs are introduced by [54] and build a top-down architecture on an existing feature extractor network by adding lateral connections. Generally, feature pyramid approaches compute features on different image scales and enable independent predictions on each scale. While traditional approaches are computational and memory inefficient, FPNs produce feature pyramids with marginal extra computational costs. RetinaNet's FPN is similar to the FPN introduced in [54], with minor changes applied to improve detection on larger scales and increase performance.

We stick to the notation of [54] and denote the ResNet's intermediate output feature maps as $\{C_2, C_3, C_4, C_5\}$ and the final set of feature maps as $\{P_3, P_4, P_5, P_6, P_7\}$. Numbers respond to different ResNet levels, as stated in Table 1 in section 3.1. The FPN uses the output feature map of the last ResNet block C_5 as input. A lateral connection applies 1×1 convolutions to reduce the number of channels to 256, resulting in feature map P_5. Nearest neighbor upsampling by factor two increases the feature maps in the top-down connection. An identical lateral connection processes feature map C_4 and then adds the output element-wise to the upsampled feature map P_5, resulting in feature map P_4. Feature map P_3 is created in the same way. Outputs of C_2 are left out to increase computational performance.

To further improve large object detection, the authors of RetinaNet add feature maps P_6 and P_7. A 3×3 convolution with stride 2 computes feature map P_6 out of C_5. P_7 is then created by applying a ReLU and another 3×3 convolution with stride 2 on P_6. The FPN applies further 3×3 convolutions on feature maps P_3, P_4 and P_5 to reduce the aliasing effect of upsampling. Eventually, the network feeds the final feature maps P_3 to P_7 separately into the classifier and BBox regressor subnets.

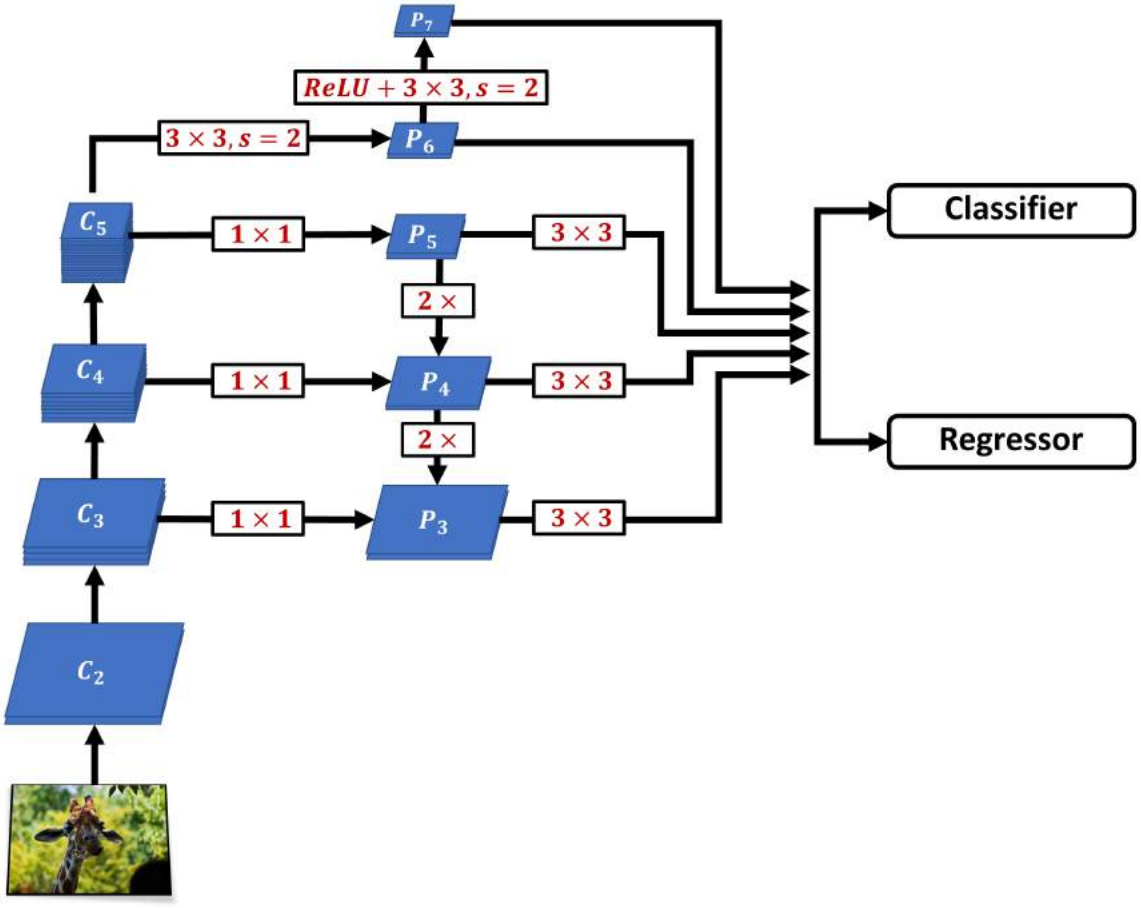


Figure 13: RetinaNet [53] architecture consisting of a ResNet-50 feature extractor, a feature pyramid network (FPN), and two subnets for classification and BBox regression, respectively. The bottom-up pathway is identical to a standard ResNet-50. Lateral connections apply 1×1 corresponding convolutions to set the number of channels to 256. Additional top-down connections use $\times 2$ nearest neighbor upsampling. Two 3×3 convolutions with stride $s = 2$ compute feature maps P_6 and P_7. The model applies additional 3×3 convolutions to P_3, P_4, and P_5 to reduce the aliasing effect of upsampling.

RetinaNet uses anchor boxes for prediction, with areas amounting from 32^2 to 512^2 on pyramid levels P_3 to P_7, respectively. Their aspect ratios are {1:2, 1:1, 2:1}. Anchor boxes at each level are scaled by $\{2^0, 2^{1/3}, 2^{2/3}\}$ to increase scale coverage. So the number of anchor boxes per level results in $A = 9$. The anchor boxes cover a scale range of 32 - 813 pixels across levels P_3 to P_7 with respect to the input image dimensions.

The network assigns two vectors for classification and BBox regression to each anchor. The classification vector has length C with C being the number of classes that are one-hot encoded. The length of the BBox vector is four. During training, we assign anchor boxes to ground-truth object BBoxes if IoU ≥ 0.5. In this case, the responding entry in the one-hot encoded classification vector is set to 1, while every other entry is set to 0. We assign an anchor to the background if its IoU < 0.4. Anchor boxes with IoU $\in [0.4, 0.5)$ are ignored during training. For BBox regression, the model computes offsets between each assigned anchor and ground-truth BBox.

Classification and regression subnets are separated from each other and do not share parameters. Both networks process inputs by four 3×3 convolutions, each followed by a ReLU activation. The number of channels stays constant at 256. The classification subnet adds another 3×3 convolutional layer with $A \cdot C$ filters. A sigmoid function is applied to the output feature map to predict each class in a binary way. The shape of resulting feature maps is $W \times H \times AC$ with W and H corresponding to the input feature map's width and height.

The regression subnet uses a 3×3 convolutional layer with $4 \cdot A$ filters to predict relative offsets (x, y, w, h) between anchor boxes and ground-truth BBoxes. The regression is class-agnostic since it is separated from the classification subnet. That results in a reduced number of parameters compared to non-class-agnostic approaches. The output shape of the regression subnet is $W \times H \times 4A$.

During inference, we only take predictions with confidence above a threshold of 0.05 into account. To further increase speed, the model decodes only 1,000 top-scoring BBox predictions. The remaining predictions are then merged on all levels and Non-Maximum Suppression (NMS) is applied. NMS [55] is a greedy algorithm to filter location proposals based on confidence. The algorithm takes a set of proposed BBoxes as input. We put the proposal with the highest confidence score into a set of final proposals. Next, we compute the IoU between this proposal and all other proposals from the input set. We remove proposals with an IoU larger than a threshold from the input set. We repeat this process until the input set is empty. RetinaNet sets the threshold to 0.5. One drawback of NMS is that the algorithm also removes proposals for neighboring objects if present in the overlap threshold. As a solution, [56] propose soft-NMS that decays confidence scores of overlapping proposals proportional to the IoU instead of removing proposals. To keep our model close to the original RetinaNet, we stay with the standard NMS.

RetinaNet uses a multi-task loss for classification (focal loss) and regression (L_1 loss). The authors introduce focal loss to tackle large class imbalance problems between the background class and actual objects in the foreground. Focal loss reshapes standard cross-entropy loss to down-weight easy samples and focuses on easily misclassified samples. In object detection, these hard samples refer to actual objects that are falsely classified as background. Focal loss assigns more weight to these hard samples and down-weights easy samples. So the contribution of hard samples to the total loss increases, while easy samples' contribution decreases.

For each anchor box, the authors define focal loss as

$$\mathcal{L}_{focal} = -(1-p_t)^{\gamma}\, log(p_t) \quad \text{with} \quad p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases} \tag{3.1}$$

with $y \in \{\pm 1\}$ determining the ground-truth class and $p \in [0, 1]$ the predicted probability for the class with label $y = 1$. Hyperparameter $\gamma \geq 0$ adjusts the rate at which easy samples are down-weighted. The authors also add a weighting factor α_t to balance the importance of positive and negative samples. Factor α_t is based on another hyperparameter $\alpha \in [0, 1]$ and is computed analogously to p_t. The α-balanced variant of the focal loss for each anchor and C classes is then defined as

$$\mathcal{L}_{balanced\,focal} = -\alpha_t(1-p_t)^{\gamma}\, log(p_t) \tag{3.2}$$

The authors state best practical results for $\gamma = 2$ and $\alpha = 0.25$. Figure 14 visualizes focal loss for varying values of γ. [57] present an automated version of focal loss that substitutes γ by a parameter that is automatically adapted during training. The authors demonstrate a significant faster training convergence. We stay with $\gamma = 2$ in our experiments to keep results comparable to a vanilla RetinaNet. We calculate the total focal loss per sample during training as the sum of focal loss over all anchor boxes. The result is then normalized by the number of anchor boxes assigned to a ground-truth BBox.

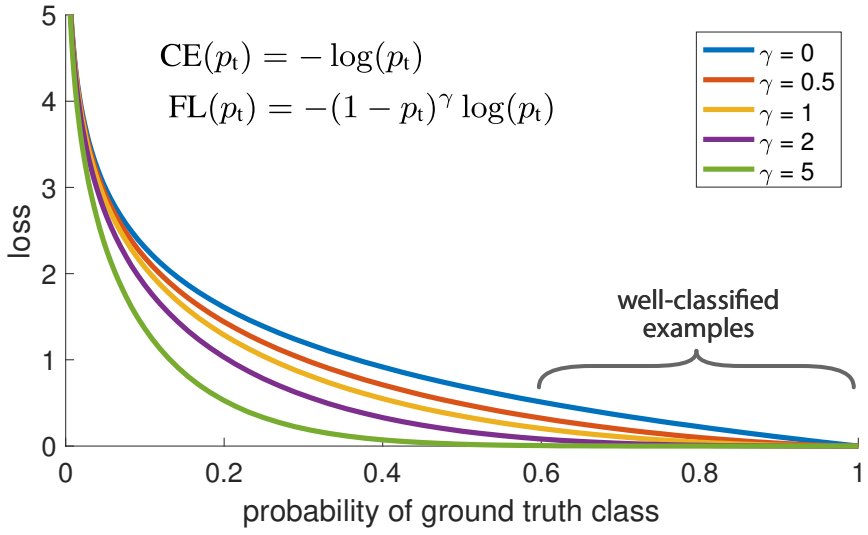


Figure 14: Visualization of focal loss for different values of γ. For $\gamma = 0$, focal loss is equivalent to cross-entropy loss for binary classification. The higher γ is set, the more easy samples are down-weighted. The authors state the best results for $\gamma = 2$. Image source: [53].

For training the BBox regression subnet, RetinaNet uses a smooth L_1 loss, as introduced by Faster R-CNN [43]. Let (x, y, w, h) denote the BBox center's coordinates, width, and height. Further, x, x_a, and x^* refer to the predicted BBox, anchor box, and ground-truth BBox, respectively. The notations apply analogously for y, w, h. Be t_i a vector representing the parameterized coordinates of a predicted BBox and t_i^* the ground-truth BBox. We compute the values as follows:

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a}, \quad t_w = log\left(\frac{w}{w_a}\right), \quad t_h = log\left(\frac{h}{h_a}\right) \tag{3.3}$$

$$t_x^* = \frac{x^* - x_a}{w_a}, \quad t_y^* = \frac{y^* - y_a}{h_a}, \quad t_w^* = log\left(\frac{w^*}{w_a}\right), \quad t_h^* = log\left(\frac{h^*}{h_a}\right) \tag{3.4}$$

Based on these values, we calculate regression loss $\mathcal{L}_{reg}$ as

$$\mathcal{L}_{reg} = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i - t_i^*) \tag{3.5}$$

with $smooth_{L_1}$ being the smooth L_1 loss function, which is less sensitive to outliers than the L_2 loss. Smooth L_1 loss is defined by [42] as

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}. \tag{3.6}$$

4 Basic Mixture of Experts Concepts

This chapter introduces basic MoE concepts for image classification and evaluates them experimentally as a proofs-of-concept. We perform experiments without paying respect to model complexity or conditional computation. However, our experiments in chapters 5 and 6 take complexity into account and introduce sparsity into MoE models. All experiments in this chapter use the default MNIST dataset and a Swiss remake of it. We introduce both datasets in section 4.1. Hardware and software specifications for our experiments are stated in Appendix A.

Our network architectures and label decoding approaches are presented in section 4.2. In section 4.3, various baseline models are trained for comparison. We then analyze three different variants of MoE architectures. Section 4.4 covers MoE models in a dataset split setting. We train one expert on each dataset and then combine individual expert predictions. Next, we train in section 4.5 two experts, each on half a set of labels. Since each expert is agnostic of some labels, two decoding approaches for expert predictions are compared. In our third experiment in section 4.6, we train an expert on each 2-combination of labels with a total of 45 experts. We analyze the weight assignment process and demonstrate the dying expert problem.

All experiments in this chapter consist of two training phases: In the first phase, we train the expert models. In the second phase, all experts' weights are frozen and gating networks are trained to combine the experts' predictions. Training sets are identical for both training phases, although depending on the experimental setting, we use a subset to train the experts.

We measure our models' performance as classification accuracy, the rate of correct classifications on an independent test set. We Gating networks and experts are trained with cross-entropy loss, which combines a softmax function with a negative log-likelihood loss[3]. Be y the one-hot encoded target vector and $\hat{y}$ the network's output logits. We then define the loss function for a single sample following Equation 4.1 [4, p.222]. We do not use other loss functions in this chapter and do not constrain expert utilization.

$$\mathcal{L}_{ce} = -y^T log\left(softmax(\hat{y})\right) = -log\left(\frac{e^{\hat{y}_{correct}}}{\sum_{i=1}^{C} e^{\hat{y}_i}}\right) \tag{4.1}$$

[3]PyTorch implements cross-entropy loss to accept raw logits as input. Softmax is then applied in the loss function, as stated in Equation 4.1. TensorFlow implementation, for comparison, expects a probability distribution instead of raw logits per default.

4.1 MNIST Datasets and Preprocessing

We introduce section the datasets for our proof-of-concept experiments in this section. The models are trained on the MNIST handwritten digits dataset [58]. The default MNIST dataset contains 60,000 training samples and 10,000 test samples. Each sample represents a grey-level image with 28x28 pixels. We show samples of the default MNIST dataset in Figure 15a.

To increase data variation, we also use a Swiss remake of the MNIST dataset [59]. This second dataset contains 16,393 handwritten digits, divided manually into a distinct training set (14,056 samples) and a test set (2,337 samples). Each sample represents an RGB image with 28x28 pixels. We ignore additional meta information like the age or gender of the persons who wrote the numbers. Figure 15b visualizes samples of the Swiss MNIST. We refer to this dataset as *S-MNIST*.

We transform the S-MNIST samples into grayscale values during preprocessing, which we then invert to match the MNIST data. Samples of both datasets are standardized using $\mu = 0.1307, \sigma = 0.3015$ for MNIST and $\mu = 0.0349$, $\sigma = 0.0954$ for S-MNIST, respectively. We calculate the statistics on each training set. For experiments with both datasets, we subsample the MNIST training and test sets to match their sizes to the S-MNIST datasets.

When comparing both datasets visually in Figure 15, a clear difference between them is notable. While MNIST digits are centered and written with bold lines, S-MNIST counterparts are more faded, illegible, and shifted. Therefore, we expect our models to perform slightly worse on the Swiss dataset because its data is more inconsistent and harder to generalize.

(a) Default MNIST samples (b) Swiss MNIST (S-MNIST) samples

Figure 15: Visual comparison between both MNIST datasets before preprocessing. We transform samples of S-MNIST into grayscale values and standardize samples of both datasets using statistics corresponding to each dataset. Image sources: [58] (MNIST), [59] (S-MNIST).

4.2 Implementation Details for Expert and Gating Networks

This section presents the architectures and their implementation details for our experiments. We rely on a simple CNN, referred to as *MnistNet*, for experiments on both MNIST datasets. The model consists of two convolutional layers with 3×3 filters and ReLU nonlinearity. A max-pooling layer with stride 2 follows each convolutional layer to reduce dimensions. The number of output channels amounts to 32 and 64.

In the next step, the feature maps are flattened and combined by two fully connected layers with 128 neurons and 10 neurons, respectively. The first fully connected layer also applies a ReLU activation. For regularization reasons, we apply dropout with $p = 0.5$ on the first fully connected layer. The network consists of 421,642 learnable parameters. We use bias terms in each layer. Figure 16 states a more detailed view of the architecture. We denote layers with stride 2 in this and other network architecture visualizations by /2. If no stride size is stated, the stride corresponds to 1.

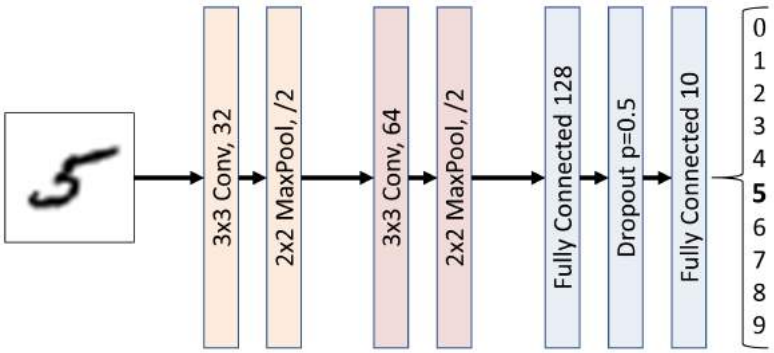


Figure 16: *MnistNet* architecture consisting of two convolution blocks, each followed by a max-pooling layer with stride 2 for downsampling. Two fully connected layers then process the outputs. We apply dropout with $p = 0.5$ to the first fully connected layer.

We reuse the MnistNet architecture for our first gating network, to which we refer as *MnistNetGate*. The only change in architecture we make is to set the number of output neurons equal to the number of experts N and apply a softmax function to the output logits. The number of trainable parameters sums up to $420,352 + 129N$. The final softmax layer produces weights g_i for the i-th expert network's logits $e_i(x)$.

Furthermore, we define a second gating network architecture, referred to as *FMGate*. The gating network takes concatenated feature maps of the second max-pooling layer of MnistNet-based experts as input. Consequently, its number of input channels varies depending on the number of experts and amounts to $64N$. The gating network processes the input feature maps in a single 3×3 convolutional layer.

Two fully connected layers, identical to MnistNetGate, then process the flattened feature maps. We compute the MoE output as $F_{MoE}(x) = \sum_{i=1}^{N} g_i(x_i')e_i(x)$ with max-pooling output x_i' of expert E_i. The number of trainable parameters amounts to $819,584+147,585N$. Figure 17 visualizes the full network architecture.

For experiments in sections 4.5 and 4.6, we train our experts only on a subset of labels without information about the full set. Therefore, our experts return predictions for less than ten labels and we need to decode their outputs in our MoEs. We use two different decoding approaches. Our first approach is a simple mapping function that maps each expert's logits into a larger vector with size ten and sets the values for unknown labels to zero.

Our second approach inflates each expert's output by a single fully connected layer with ten output neurons and no bias. This increases the total number of trainable parameters by $\sum_{i=1}^{N} 10 \cdot |O_i|$ with $|O_i|$ being the number of output neurons of expert i. We combined these approaches with our gating networks MnistNetGate and FMGate, and denote the models with *Mapping* and *FC*, respectively. We also tried to add an *unknown* class to each expert and add a comparable number of samples from the other domains to the training sets. Since the resulting models perform similarly to experts without an additional class, we do not follow that approach further.

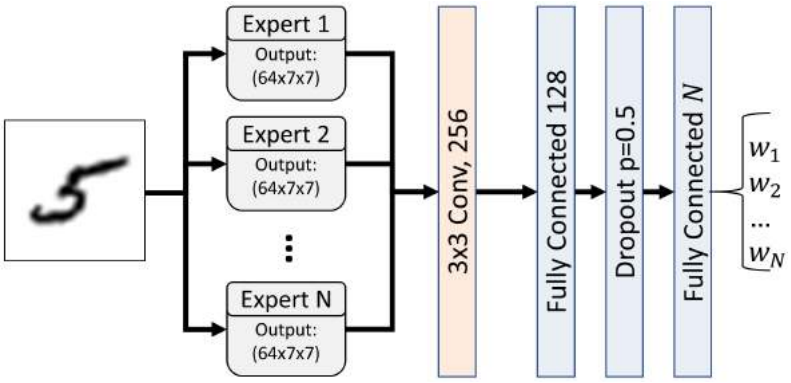


Figure 17: *FMGate* architecture that takes feature maps extracted from a fixed number N of expert networks as input. Feature maps correspond to the output of the second max-pooling layer in MnistNet-based experts. The gating network processes the concatenated feature maps with a single 3×3 convolutional layer with stride 1. The following fully connected layers are identical to an MnistNetGate. The number of output neurons corresponds to the number of experts.

4.3 MnistNet Baselines

We provide a separate MnistNet baseline model for each dataset. We train *Baseline-D* on default MNIST and *Baseline-S* on S-MNIST. A third baseline, *Baseline-C*, is trained on a combination of both datasets using a subsampled MNIST to avoid biases due to the different dataset sizes. We use a fixed learning rate for all experiments since decreasing it during training does not significantly improve accuracy. Further experimental details are stated in Appendix B.1.

Our evaluation results, stated in Table 2, show nearly perfect prediction accuracy for our Baseline-D with 99.34%. This result illustrates the low complexity of MNIST. Baseline-S achieves slightly worse results with 97.63% accuracy. Note that Baseline-S performs better on the MNIST test set than the other way around, although we train the model on much less training data. We assume the model generalizes better because S-MNIST is more complex than MNIST. Furthermore, Baseline-C slightly outperforms Baseline-S but falls behind Baseline-D. It is most likely because the combined training set is twice as large as the S-MNIST training set and about half the MNIST training set size.

Model	Combined	MNIST	S-MNIST
Baseline-D		$99.34\% \pm 0.03$	$67.1\% \pm 0.15$
Baseline-S		$81.83\% \pm 1.07$	$97.63\% \pm 0.31$
Baseline-C	$98.42\% \pm 0.26$	$98.91\% \pm 0.20$	$97.93\% \pm 0.32$

Table 2: Mean accuracy and sample standard deviation for MnistNet baselines. We train a model on MNIST (Baseline-D), S-MNIST (Baseline-S), and a combination of both datasets (Baseline-C). We omit the evaluation results of Baseline-D and Baseline-S on the combined test set since the larger MNIST training set would induce a strong bias into the results. All results are averaged over three runs.

4.4 Experiments with Dataset Split MoE

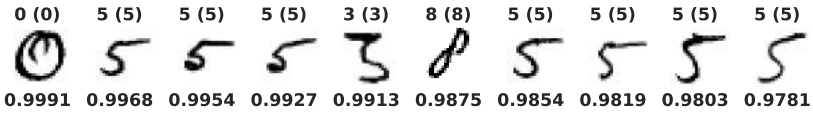
We build a simple MoE architecture with a dataset split setting. First, we train a MnistNet-based expert on a subsampled MNIST and reuse the already trained S-MNIST baseline as a second expert. After training for 25 epochs, we fix the experts' weights and train a gating network to combine their outputs. We train both gating network variants, FMGate and MnistNetGate, on the same expert networks. Experimental details are stated in Appendix B.2.

We list our test results in Table 3. Unsurprisingly, the results on the combined test set are identical to the mean of its subsets. Baseline-C, which we train on both MNIST training sets, yields comparable results to both MoE variants. Similar to our baseline results stated in Table 2, accuracy on the S-MNIST is generally lower than on MNIST because S-MNIST samples are more inconsistent and, therefore, harder to generalize. The MnistNetGate performs slightly better than FMGate on the S-MNIST test set, but the difference is not significant. Note that MoE models and Baseline-C achieve results similar to the experts on their specific domain. We conclude that both gating networks are capable of identifying underlying structures in the data to weight experts appropriately.

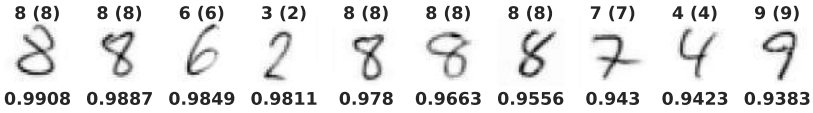
Model	Combined	MNIST	S-MNIST
Baseline-C	**98.42**% ± 0.26	**98.91**% ± 0.20	**97.93**% ± 0.32
Expert MNIST	78.66% ± 0.21	98.83% ± 0.05	58, 49% ± 0.43
Expert S-MNIST	89.73% ± 0.47	81.83% ± 1.07	97.63% ± 0.31
MoE FMGate	98.17% ± 0.07	**98.89**% ± 0.05	97.45% ± 0.13
MoE MnistNetGate	98.29% ± 0.13	**98.90**% ± 0.07	97.68% ± 0.27

Table 3: Mean accuracy and sample standard deviation for different MoE architectures, experts, and Baseline-C on MNIST and S-MNIST test sets, and a combination of both. We highlight the best results on each test set.

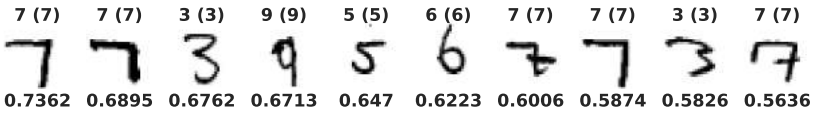
We further analyze the average weight assignment: both gating networks assign slightly more weighting to the S-MNIST expert with an average weighting of 0.5079 (FMGate) and 0.5215 (MnistNetGate) on the combined test set. Accordingly, the gating networks rely slightly more on the S-MNIST expert. We assume this is because the S-MNIST experts have to generalize more widely due to the higher data complexity. The gating networks consequently assign hard MNIST samples to the expert trained on S-MNIST.



(a) *FMGate* - MNIST samples assigned to the S-MNIST expert.



(b) *FMGate* - S-MNIST samples assigned to the MNIST expert.



(c) *MnistNetGate* - MNIST samples assigned to the S-MNIST expert.

(d) *MnistNetGate* - S-MNIST samples assigned to the MNIST expert.

Figure 18: Test samples that the gating networks assign to the expert trained on the other domain with the highest weights. We state the resulting MoE prediction on top of each sample. The numbers in brackets correspond to the ground-truth labels. We specify the assigned weights below each sample. The visualized S-MNIST samples are already transformed into grayscale values and inverted.

We visually analyze the top-weighted samples of both test sets. Figure 18 states the top-weighted samples and their assigned weights for both gating network variants. FMGate assigns high weights also to experts trained on the respective other dataset, while MnistNetGate relies more on the experts trained on the same domain as the test samples. Note that MnistNetGate sees the raw input image, while FMGate only sees the experts' feature maps. So MnistNetGate can better identify the domain of each input sample and consequently weights the corresponding expert higher. Still, the predictions for top-weighted samples are almost always correct for both MoE models.

We conclude that both gating networks can identify the best expert for each case, even if the expert is trained in another domain. Both gating networks seem to rely on the MNIST expert for images of eights. Assignments for the S-MNIST expert differ more. While FMGate assigns the highest weights mainly for samples containing fives, MnistNetGate allocates a broader range of labels with high weightings.

4.5 Experiments with Semi Label Split MoE

Next, we analyze an MoE architecture for label splits instead of dataset splits. Experts only know a subset of all labels in a label split setting. Therefore, they are less generalized compared to experts trained on the whole dataset. In a first step, we divide the full MNIST dataset into two subsets with labels $[0, 4]$ and $[5, 9]$. Since S-MNIST is relatively small, we rely only on MNIST in the following experiments. We then train an expert based on MnistNet on each subset. In a second step, we train a gating network to combine both experts. Experimental details are stated in Appendix B.3.

Table 4 states our evaluation results on different test sets. Baseline-D achieves the highest accuracy of all models on the full MNIST test set. However, the expert for labels $[5, 9]$ significantly outperforms it in terms of accuracy on the corresponding test set. The MoE using MnistNetGate with decoding by mapping performs similarly to the expert for labels $[4, 9]$, but below expert $[5, 9]$. In all test cases, the MnistNetGate MoE exceeds the FMGate MoE. Results for MnistNetGate also show a smaller standard deviation between different runs. We conclude that a gating network can extract more meaningful information from raw images to select trustworthy experts than from the experts' feature maps. Also, note that results vary much more for MoE models than for the simple CNN baseline.

Model	Full MNIST	MNIST [0, 4]	MNIST [5, 9]
Baseline-D	**99.34**% ± 0.03	**99.66**% ± 0.08	99.00% ± 0.10
Expert [0, 4]		99.28% ± 0.07	
Expert [5, 9]			**99.78**% ± 0.06
MoE FMGate Mapping	98.19% ± 0.09	98.80% ± 0.37	97.54% ± 0.42
MoE FMGate FC	97.93% ± 0.50	98.32% ± 0.73	97.52% ± 0.32
MoE MnistNetGate Mapping	98.83% ± 0.03	99.31% ± 0.28	98.32% ± 0.29
MoE MnistNetGate FC	98.12% ± 0.20	98.33% ± 0.27	97.90% ± 0.16

Table 4: Mean accuracy and sample standard deviation for MoE models in a label split setting. The experts are only evaluated on one test set since they are unable to make predictions for other labels due to their network architecture. We highlight the best results for each test set.

FC decoding achieves reliable but always inferior results compared to mapping approaches. That is not surprising since models using FC decoding need to learn additional parameters. Even in a global optimum, it is unlikely that this decoding approach induces any other meaningful information compared to a mapping approach, which corresponds to ground-truth decoding. We still include FC decoding in our experiments because an explicit mapping is only feasible if an expert's domain and output semantic is known. In practice, situations are imaginable in which a gating network relies on several experts whose outputs cannot directly be combined as a convex combination.

4.6 Experiments with 2-Combinations Label Split MoE

We conduct another experiment on label splitting. This time, we train an MnistNet expert on each 2-combination of all labels resulting in $\binom{10}{2} = 45$ different experts. All models in this setting are trained on MNIST. Afterward, we train our gating network models to combine the expert outputs. Experimental details are stated in Appendix B.4.

We state our test results for different gating networks and decoding approaches in Table 5. We also state each expert's accuracy on corresponding test sets with only two labels in Figure 19. The heatmap provides insight into various experts' performances and shows which labels experts have problems distinguishing. The most difficult pairs of numbers to differentiate between seem to be (7, 9), (8, 9), and (3, 5). Best results are achieved on $(6, 7)$ and $(0, 1)$.

Model	Full MNIST
Baseline-D	**99.34**% ± 0.03
MoE FMGate Mapping	97.67% ± 0.39
MoE FMGate FC	97.42% ± 0.12
MoE MnistNetGate Mapping	99.05% ± 0.07
MoE MnistNetGate FC	99.01% ± 0.09

Table 5: Mean accuracy and sample standard deviation for a 2-combinations label split setting. We average results over three training runs. Our baseline achieves the best test accuracy. MoE models with MnistNetGate perform slightly worse than the baseline but significantly better than models with FMGate. Decoding approaches with fully connected layers perform similarly to mapping approaches.

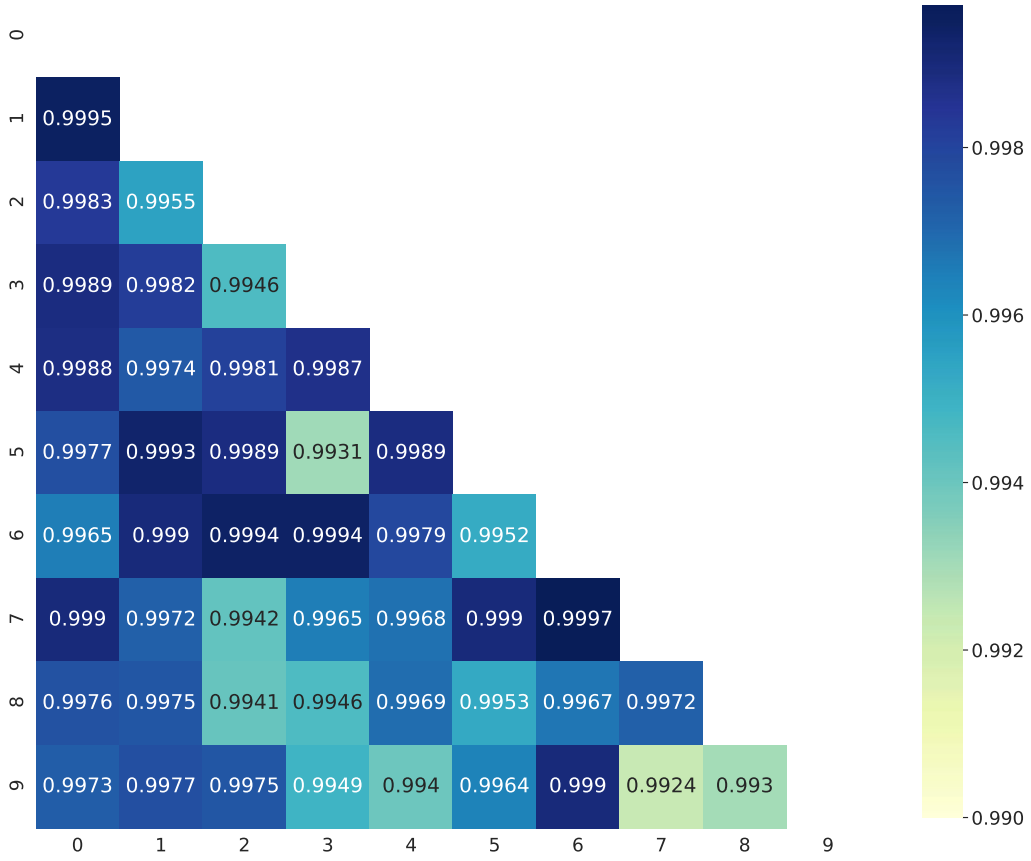


Figure 19: Accuracy for distinct MnistNet experts in a 2-combinations label split setting. We average results over three training runs. Experts for labels (6, 7) and (0, 1) achieve the best test results. The experts for labels (7, 9), (8, 9), and (3, 5) show the lowest test accuracy.

Like in previous experiments, Baseline-D outperforms every MoE model. Models with FMGate show inferior evaluation results compared to similar models trained in a semi label split dataset, as proposed in section 4.5. That may be because FMGate models have to handle a much larger input in the form of concatenated feature maps and fit significantly more parameters. That probably negatively affects their ability to generalize on the training data. MnistNetGate-based models, on the other hand, outperform their semi label split equivalents. Again, models using a mapping approach perform slightly better than models with additional fully connected layers for decoding.

We analyze expert utilization of different models using a discrete Lorenz curve. It is a tool used primarily by economists to analyze wealth and income distribution. A Lorenz curve visualizes inequality and shows the share v_j of total income that the bottom share u_j of households with the least income possess. We interpret the total weight assignment to each expert during test time as income and compute each model's Lorenz curve. Values on the abscissa correspond to the relative share v_j of total weights assigned to the bottom j experts with the least weights assigned.

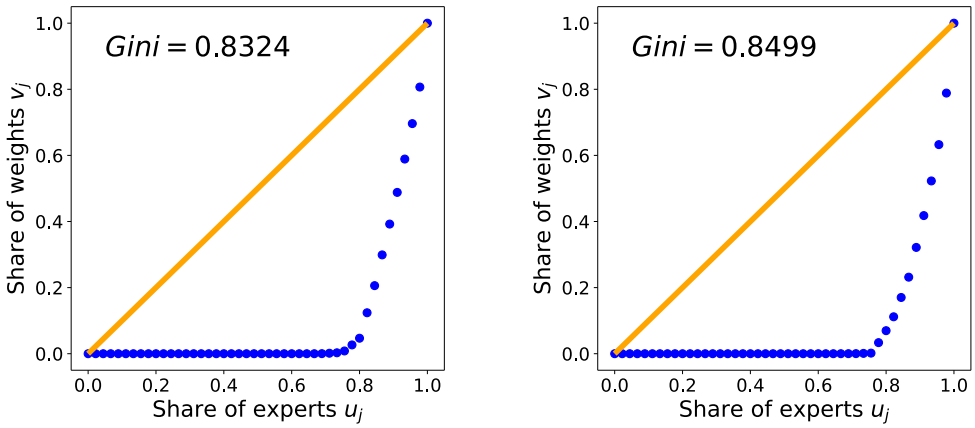
We first sort the sum of assigned weights to each expert in ascending order. We then calculate $v_j = \frac{\sum_{i=1}^{j} w_i}{\sum_{i=1}^{N} w_i}$ with w_i being the sum of weights assigned to expert E_i during test time. That is equivalent to the expert's importance computed on all test samples. $\sum_{i=1}^{N} w_i$ corresponds to the number of test samples since weights assigned to experts during a single forward pass sum up to one. Values u_j on the ordinate are equally spaced and calculated as $u_j = \frac{j}{N}$ with N being the total number of experts.

The resulting Lorenz curve is a monotonically increasing and convex function $L(u_j) = v_j$. The curve would be a straight line through points (0, 0) and (1, 1) for entire completely even weights. Lorenz curves for expert utilization visualize which share v_j of total weights are assigned to the share u_j of experts with the least weightings [60, pp.70-72]. To further compare different Lorenz curves quantitatively, we compute the Gini coefficient for each Lorenz curve. The higher the degree of inequality, the larger the area A between a Lorenz curve and the line at 45° becomes. The Gini coefficient quantifies the proportion between area A and the area between the 45° diagonal line, which covers an area of 0.5. We compute the Gini coefficient according to Equation 4.2. We then normalize the result by $Gini_{norm} = \frac{N}{N-1} Gini$ to ensure $0 \leq Gini_{norm} \leq 1$. The more concentrated the distribution of weights is, the greater the value of $Gini_{norm}$ becomes [60, pp.73-74].

$$Gini = \frac{A}{0.5} = 2A = 2 \frac{\sum_{i=1}^{N} i \cdot w_i - (N+1) \sum_{i=1}^{N} w_i}{N \cdot \sum_{i=1}^{N} w_i} \tag{4.2}$$

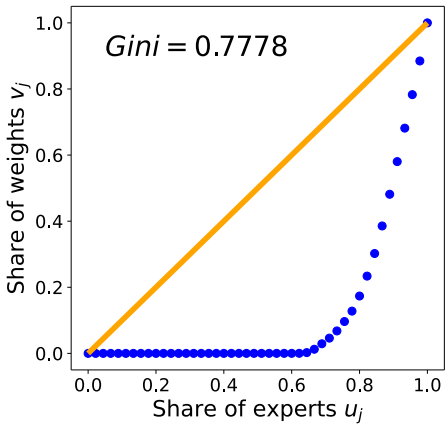
Figure 20 shows the resulting Lorenz curves and Gini coefficients for different MoE architectures. Table 6 further states the total weights assigned to the top-weighted experts. We compute the values on the corresponding models with the highest test accuracy. Each of these models shows a massive dying expert problem, independently of the MoE architecture. MnistNetGate-based models show a significantly lower $Gini_{norm}$ coefficient than the corresponding FMGate models. Differences mainly arise since MnistNetGate models assign total weights more equally to the highest weighted experts.

FMGate models, on the other hand, favor a single expert and assign significantly higher weights to that expert. Lorenz curves 20a and 20b illustrate that fact with a large gap between the last two data points. Furthermore, FMGates assign approximately 97% weights to the ten highest weighted experts. MnistNetGates allocate only 86 - 87% to the same group.

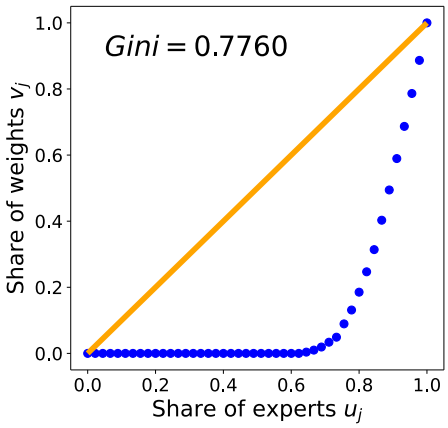


(a) MoE FMGate Mapping

(b) MoE FMGate FC



(c) MoE MnistNetGate Mapping



(d) MoE MnistNetGate FC

Figure 20: Lorenz curves (blue) for weight assignments during the evaluation on MNIST. The orange line represents a perfect uniform distribution of weights. Results visually demonstrate the dying expert problem of MoE models. Besides, we state the normalized Gini coefficient for each Lorenz curve. In every model, the gating networks assign only small weights to about two-thirds of the experts.

We further analyze how precise each gating network selects experts. Thus, we compute the assignment accuracy by summing up assigned weights to experts trained on the same label as the current test sample. We then put the sum in relation to the total amount of weights assigned during training, which corresponds to the number of samples in the test set. Gating networks with a mapping approach achieve an assignment accuracy of almost 98%. Gating networks using fully connected layers for decoding assign only 75% (FMGate FC) and 79% (MnistNetGate) to correct experts.

Rank	FMGate Mapping	FMGate FC	MnistNetGate Mapping	MnistNetGate FC
1	(4, 9): 1908.38	(0, 1): 2119.43	(1, 8): 1159.25	(1, 7): 1132.52
2	(1, 7): 1103.90	(3, 9): 1547.18	(6, 7): 1013.96	(4, 9): 1014.73
3	(7, 8): 1085.75	(1, 7): 1088.54	(0, 3): 1013.23	(7, 9): 997.70
4	(0, 3): 1016.40	(2, 6): 1049.50	(2, 5): 1012.26	(3, 8): 969.90
5	(5, 9): 943.48	(0, 4): 959.02	(0, 1): 988.55	(1, 6): 954.33
6	(0, 4): 936.78	(1, 5): 901.11	(2, 6): 957.75	(1, 5): 934.19
7	(6, 9): 922.26	(4, 7): 630.85	(5, 9): 832.41	(0, 1): 842.95
8	(0, 2): 806.27	(0, 5): 572.42	(0, 8): 691.62	(2, 4): 632.29
9	(1, 8): 780.32	(0, 7): 434.77	(5, 6): 617.79	(2, 6): 601.59
10	(5, 8): 195.30	(2, 4): 377.18	(4, 6): 461.32	(3, 7): 525.16
$\sum$	9,698.84	9,680.00	8,748.14	8,605.36
Assignment acc	0.9772	0.7477	0.9794	0.7900

Table 6: Top weight assignments of different gating networks in a 2-combinations label split MoE setting. Numbers in brackets refer to the label set on which an expert is trained. We state weight assignments as total weights assigned to a single expert during evaluation. Also, we state the assignment accuracy that refers to the proportion of weights assigned to experts trained on the test sample's domain. Note that the total amount of available weight assignment amounts to 10,000, the MNIST test set's size.

Nevertheless, both models with FC decoding achieve similar test accuracy compared to gating networks with a mapping approach. These results demonstrate that FC models rely not only on identifying experts of the same domain as the input sample but also give significant attention to experts of other domains. Even if some experts might not have seen samples of a specific domain, the gating networks can still derive correct predictions. Hence, the predictions of experts trained in one domain also contain useful information to make predictions for other domains.

5 Image Classification Enhancements with MoE Layers

After examining various classic MoE architectures, we now conduct experiments using embedded Sparsely-Gated Mixture-of-Experts Layers (MoE layers), as introduced in section 2.3. All models in this chapter are based on ResNet-18, which we modify with different MoE layer variants. We introduce in section 5.1 the CIFAR-100 dataset for image classification. An overview of the vanilla ResNet-18 and our modifications are stated in section 5.2. We rely on deep expert networks based on ResNet-18 blocks in most of our experiments in this chapter. Details on experts and our main gating network are stated in section 5.3.

We start our MoE layer experiments in section 5.4 with replacements of single convolutional layers. Our focus in this chapter lies on deep expert networks similar to vanilla ResNet-18 blocks. Therefore, we also replace distinct ResNet blocks with MoE layers using deep experts and analyze expert utilization for soft and hard constrained models. Section 5.5 discusses results for soft constrained models and section 5.6 for hard constrained models. The gating networks' sample assignment decisions are visually demonstrated with t-SNE and PCA plots in section 5.7.

We further analyze the behavior of distinct experts and the per-class prediction accuracy of these in section 5.8. Section 5.9 sheds light on the effect of varying the number of active experts per forward pass. Since all models are trained from scratch with the full CIFAR-100 dataset, no explicit expert domains are defined. We demonstrate in section 5.10 that the experts in embedded MoE layers specialize implicitly in distinct subdomains of the input space. Section 5.11 completes the chapter and presents some modifications to our gating network architecture and training schedules to further improve prediction accuracy. All models in this chapter are initialized with PyTorch default values using Kaiming initialization [61].

We denote our models with deep expert networks by the scheme *NarrowMoEConstraint-Position-NumberOfExperts*. Constraints are abbreviated as *KL* (KL divergence loss), *Imp* (importance loss), *Rel* (relative importance constraint), and *Mean* (mean importance constraint). We refer to the corresponding MoE layers with deep experts also as MoE blocks. The hardware and software specifications for our experiments are stated in Appendix A.

5.1 CIFAR-100 Dataset and Preprocessing

We use the CIFAR-100 dataset [62] for more complex classification experiments[4]. The dataset contains 60,000 RGB color images with pixel dimensions 32x32, which results in an overall input size of 3,072 values per sample. Each sample belongs to one of 100 distinct classes. CIFAR-100 does not support Multi-label classification. The authors further group the classes into 20 superclasses, each including five classes. We state an overview of all classes and superclasses in Appendix C. The dataset provides 500 training images and 100 test images per class. Consequently, the training set contains 50,000 images, and the test set 10,000 images. Figure 21 visualizes randomly selected samples from different classes.

We apply various data augmentation techniques during preprocessing. Each image is flipped horizontally with a probability of 50%. Besides, we randomly change the brightness, contrast, and saturation of each sample individually with jitter factors chosen uniformly and independently from the interval $[0.7, 1.3]$. Images are then padded on all four sides with an additional zero value and cropped at a random location so the input dimension of 32x32 is maintained. Finally, all images are standardized using $\mu = (0.5071, 0.4865, 0.4409), \sigma = (0.2009, 0.1984, 0.2023)$. Means and standard deviations are computed on the CIFAR-100 training set.

Figure 21: CIFAR-100 sample images. Each RGB color image consists of $32 \times 32 \times 3 = 3,072$ values and belongs to one of 100 distinct classes. Samples are shown before preprocessing and data augmentation. Image source: [62].

[4]Dataset is available at `https://www.cs.toronto.edu/~kriz/cifar.html`.

5.2 Implementation Details and Modifications for ResNet-18

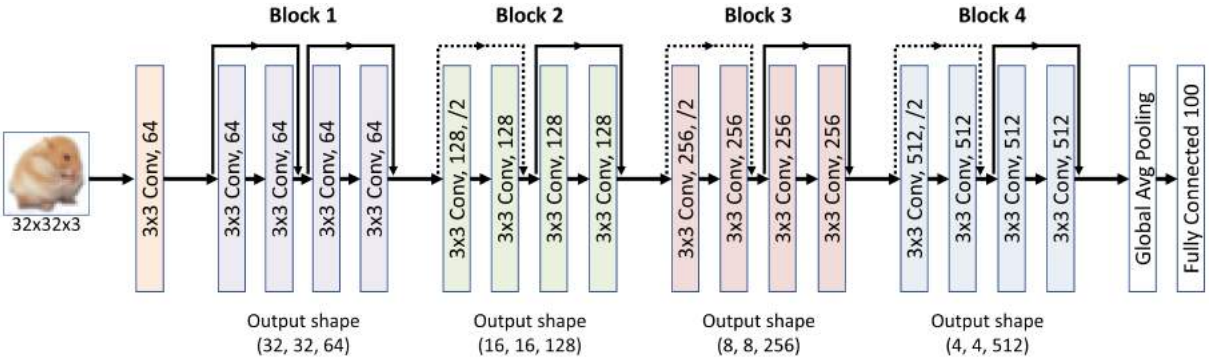


Figure 22: Our ResNet-18 with modified input and output layers for training on CIFAR-100. We state filter size and number of filters for each convolutional layer. We indicate downsampling with stride 2 by /2. Each ResNet block consists of two building blocks, as visualized in Figure 7 in section 3.1. Batch normalization and rectification are applied analogously. Dashed lines indicate projection shortcuts. We state output shapes as *(height, width, channels)*. The representation is inspired by [34].

We use a simple modified version of a vanilla ResNet-18[5] proposed by [31] and introduced in section 3.1. Our focus lies on the behavior of MoE layers and not on pushing state-of-the-art results further. We also trained variants of ResNet-34, ResNet-50, and ResNet-101 but found their capacity to be too large. A large vanilla model capacity would distort our results with MoE layers that also increase capacity. ResNet-18 also offers a decent training time in combination with MoE layers.

The basic ResNet-18 implementation uses a 7×7 convolutional layer as input layer, followed by a 3×3 max pooling. Both layers use stride 2 for dimension reduction. While the ResNet authors [31] train on ImageNet, we train our network using CIFAR-100 with an input dimension of $32 \times 32 \times 3$. Therefore, we reduce the input layer's filter size to 3×3 with stride 1 and remove max-pooling. ResNet-18 consists of four major ResNet blocks, each composed of two building blocks with convolutional layers. Each convolutional layer performs zero-padding of one pixel to each side to maintain the feature maps' size. Figure 22 provides an overview of our slightly modified ResNet-18 network architecture.

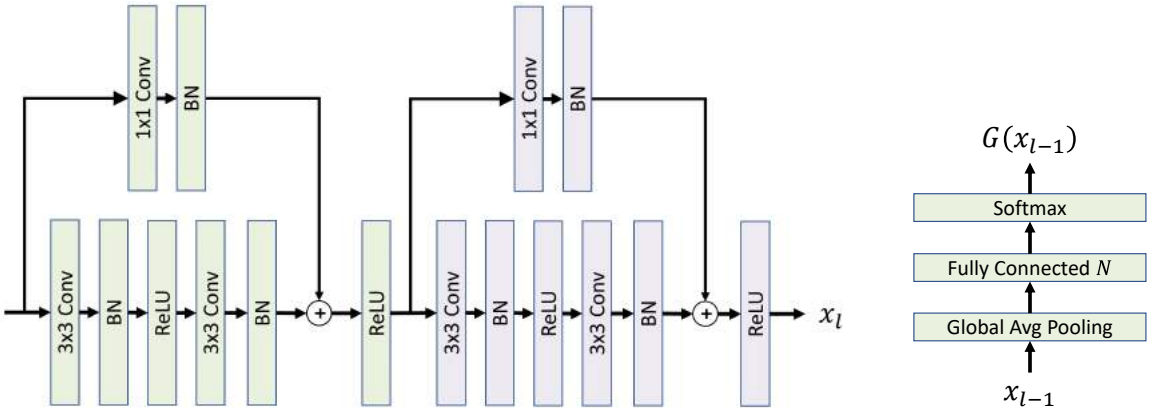
ResNet-18 applies batch normalization [63] after each convolutional layer. The number of channels doubles in each ResNet block. Simultaneously, the feature maps' size halves using a stride of 2 for the first convolutional layer in each ResNet block, except the first one. Shortcut connections with dashed lines in Figure 22 correspond to projection shortcuts, as stated in Figure 7b in section 3.1.

[5] Our models are based on the official PyTorch implementation available at
`https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py`

5.3 Implementation Details for Expert and Gating Networks

We integrate Mixture-of-Experts layers into our modified ResNet-18 architecture. We extend the concept of Shazeer et al. [6] by using deep convolutional experts based on the four ResNet blocks. We reduce the number of parameters in each expert network, referred to as *NarrowResNetExperts*, to maintain comparable computational complexity with vanilla ResNet-18. For this purpose, we adjust the number of filters of each expert in a bottleneck manner. Consequently, each expert needs to learn a slightly compressed representation of the input features, similar to undercomplete convolutional autoencoders.

We assume the bottleneck encourages experts to learn distinct feature spaces, combined by the gating network. Since we integrate MoE layers into vanilla ResNet-18 models, input and output dimensions need to stay identical to regular ResNet blocks. Another possible solution to reduce computational complexity are 1×1 convolutions to reduce the number of feature maps, as applied in GoogleLeNet [36]. We choose to stay as close as possible to the vanilla ResNet-18 and, with this in mind, do not apply 1×1 convolutions.



(a) NarrowResNetExpert (b) SimpleGate

Figure 23: Basic *NarrowResNetExpert* and *SimpleGate* architectures. Channel reduction in NarrowResNetExperts takes place in the first convolutional layer to reduce computational complexity. The third convolutional layer then increases the number of channels matching the number of output channels. Shortcut connections use 1×1 convolutions for adjusting the number of channels. The representation of Figure 23a is inspired by [31].

Figure 23a visualizes our basic NarrowResNetExperts architecture. Table 7 states further architectural details of the experts. The *Block* number refers to different ResNet blocks. More details on distinct blocks are stated in Table 1 in section 3.1. *In Channels* and *Out Channels* respond to the number of input and output channels of each expert. The first building block in each expert reduces the number of feature map channels. Both convolutional layers use a smaller number of filters corresponding to *Bottle Channels*. The second building block then increases the number of channels corresponding to *Out Channels*.

Block	In Channels	Bottle Channels	Out Channels	# Params	Complexity
Block 1	64	25	64	0.075 M	0.077 GMac
Block 2	64	50	128	0.267 M	0.068 GMac
Block 3	128	90	256	1.010 M	0.065 GMac
Block 4	256	180	512	4.037 M	0.065 GMac

Table 7: Overview of NarrowResNetExperts' parameters in different MoE blocks, replacing standard ResNet blocks. The number of channels refers to the number of input and output channels of each expert. Bottle Channels states the number of channels in the bottleneck. We measure complexity for input images with dimension $32 \times 32 \times 3$ and experts placed in a ResNet-18 architecture.

We keep the number of filters identical in each building block. Each convolutional layer's stride stays as in our vanilla ResNet-18 model, visualized in Figure 22 in section 5.2. We also propose two slightly modified MoE layer architectures. First, we add a projection shortcut that connects the MoE layer input with its output. We assume that MoE layers that are more deeply embedded in a model and have deeper expert networks will benefit from shortcuts similar to vanilla ResNets. We denote these models as *Shortcut* compared to *Basic* MoE layer architectures.

As a second modification, we no longer apply ReLU activation on each expert's output, right before different experts' results are combined. Instead, we remove the activation from every single expert and apply it to the combined feature maps. Our intuition for this decision is that without ReLU activation, different experts' feature maps can compensate each other by allowing negative values in each output feature map. It allows experts to add features to the combined result and, at the same time, express their rejection for specific features. We denote these models with postfix *ReLU* in result tables.

For our gating network, we choose a simple architecture to keep computational complexity low and facilitate convergence. We refer to it as *SimpleGate*. Our gating network first applies global average pooling, reducing each feature map to a single value. The resulting values are then processed by a single fully connected layer whose number of neurons corresponds to the number of experts. Therefore, the number of parameters increases in deeper block positions since the number of input channels increases. We set all biases to zero in the fully connected layer. A softmax function then calculates weights. Figure 23b visualizes our SimpleGate architecture. Computation budgets in MoE layers are controlled by hyperparameter k that determines the number of experts activated during each forward pass. We set $k = 2$ during training to keep computational complexity similar to vanilla ResNet-18.

We denote models using NarrowResNetExperts and SimpleGate by the naming scheme *NarrowMoE-Constraint-Position-NumberOfExperts*. We abbreviate constraints as *KL* (KL divergence loss), *Imp* (importance loss), *Rel* (relative importance constraint), and *Mean* (mean importance constraint). For example, we refer to a ResNet-18 model with an MoE layer replacing ResNet block 3 using four experts and KL divergence loss for training as *NarrowMoE-KL-3-4*.

5.4 Experiments with Single Layer MoE Embeddings

Before using *NarrowResNetExperts* and taking computational complexity into account, we first want to analyze simple MoE layers' behavior. We replace the last convolutional layer in different ResNet-18 blocks with an MoE layer containing four experts, so each model only contains a single MoE layer. We rely on our *SimpleGate* architecture as gating network. In each forward pass, the gating network activates the top $k = 2$ weighted experts. The position numbers refer to different ResNet blocks. Position 0 refers to the first convolutional layer, which directly processes the input image.

The number of trainable parameters per model increases with its position number since the number of filters in each ResNet block doubles. The computational complexity for different models stays the same since the number of feature maps halves with each ResNet block. Note that the MoE models in this section have slightly more computational complexity than our vanilla ResNet-18. We do not use pre-trained weights and train each model from scratch. Similar to NarrowMoE models, we refer to models in this section as *SingleMoE-Constraint-Position-NumberOfExperts*.

We set the weights for auxiliary losses to $w_{aux} = 0.5$. We further set thresholds for hard constraints to $m_{rel} = 0.5$ (relative importance) and $m_{mean} = 0.3$ (mean importance). Hyperparameters are selected based on a small grid search with respect to accuracy and stable expert utilization. Except for different constraints during training, the models are identical on the same level. Further experimental details are stated in Appendices B.5 (ResNet-18 baseline) and B.6 (SingleMoE models). Table 8 states test results on the CIFAR-100 test set. We highlight models with the best accuracy that do not suffer from dying experts. Experts are considered 'dead' if they receive less than 1% average weightings on the test set. As we will see in later sections, hard constrained models, particularly with mean importance, tend to suffer from the dying expert problem.

Model	Importance Loss	KL divergence Loss	Rel. Importance	Mean Importance
Baseline		$72.62\% \pm 0.29$		
SingleMoE-{}-0-4	$71.59\% \pm 0.25$	$72.10\% \pm 0.51$	**72.16**$\% \pm 0.14$	$72.06\% \pm 0.06$
SingleMoE-{}-1-4	$72.56\% \pm 0.04$	**72.72**$\% \pm 0.32$	$73.02\% \pm 0.40$	$72.51\% \pm 0.37$
SingleMoE-{}-2-4	$72.63\% \pm 0.13$	**73.00**$\% \pm 0.14$	$73.02\% \pm 0.29$	$73.09\% \pm 0.22$
SingleMoE-{}-3-4	**72.78**$\% \pm 0.20$	$72.57\% \pm 0.21$	$72.71\% \pm 0.45$	$72.92\% \pm 0.28$
SingleMoE-{}-4-4	$72.34\% \pm 0.4$	$72.40\% \pm 0.09$	**73.54**$\% \pm 0.30$	$73.37\% \pm 0.19$

Table 8: Mean accuracy and sample standard deviation for embedded MoE layers in a ResNet-18 architecture. Each expert consists of a single convolutional layer. MoE layers replace the last convolutional layer in various ResNet blocks. In all experiments, the gating network activates the top $k = 2$ weighted experts. We highlight models with the highest accuracy on each level.

For soft constrained models, SingleMoE-KL-2-4 achieves the best result with 73.00% test accuracy. Our vanilla ResNet-18 baseline accomplishes only 72.62%. MoE layers embedded in position 3 and 1, respectively, also beat the baseline slightly, but not significantly. Models using KL divergence loss perform better than importance loss models, except for position 3. However, differences are not significant. Our hard and soft constrained models achieve a comparable test accuracy for position 0 to 3.

For MoE layers embedded in position 4, both hard constraints outperform the soft constrained counterparts. They both beat the baseline model significantly with 73.54% and 73.37% accuracy, respectively. SingleMoE-Rel-4-4 also achieves the overall best test accuracy. Independently of the applied constraint, models with MoE layers in position 0 achieve the lowest test accuracy. We conclude that increased capacity and specialization using different experts do not benefit the low-level feature processing.

It is somewhat surprising that performance also decreases for soft constrained MoE layers in the last position. We expected that by increasing the capacity for high-level features with MoE layers, the models could improve feature extraction and make better predictions. On the other hand, hard-constrained models increase their accuracy with MoE layers replacing the last convolutional layer in the models. We assume that equal importance between experts as pursued by both auxiliary loss functions is not per se beneficial. It seems to increase accuracy when models utilize some experts more often than others. These experts are consequently activated more frequently, trained on more samples, and specialized in processing a broader range of high-level features. We believe that high-level features cannot be divided efficiently into groups of comparable size, in a way that each expert is responsible for a similar number of features. We further analyze the specialization of distinct experts in the following sections, particularly in section 5.10.

5.5 Experiments with Soft Constrained MoE Blocks

We now insert MoE blocks, as introduced in section 5.3, in our ResNet-18 architecture. All MoE layers in this section are trained with auxiliary losses. We analyze models trained with hard constraints afterward in section 5.6. All models are trained from scratch without pre-trained weights. In each model, we replace one complete ResNet block with an MoE layer. Since we use deep expert networks, we refer to MoE layers as MoE blocks. Other network parts remain the same.

We use NarrowResNetExperts and SimpleGate architectures in our MoE layers containing four and ten experts, respectively. For each sample, the top $k = 2$ weighted experts are activated to keep computational complexity comparable to a vanilla ResNet-18. As for experiments in section 5.4, we set $w_{aux} = 0.5$ for both soft constraints. Models with MoE blocks in the same network position are identical except for different gating constraints and possible shortcut connections. We train models with four experts for 150 epochs and models with ten experts for 180 epochs. Further experimental details are stated in Appendix B.7.

Model	Importance Loss		KL divergence Loss	
	Basic	Shortcut	Basic	Shortcut
Baseline	$72.62\% \pm 0.29$			
NarrowMoE-{}-1-4	$72.20\% \pm 0.24$	$72.24\% \pm 0.49$	$72.21\% \pm 0.29$	$\mathbf{72.72}\% \pm 0.36$
NarrowMoE-{}-1-10	$70.94\% \pm 0.14$	$\mathbf{71.51}\% \pm 0.23$	$70.76\% \pm 0.48$	$71.32\% \pm 0.60$
NarrowMoE-{}-2-4	$71.75\% \pm 0.22$	$72.18\% \pm 0.29$	$71.78\% \pm 0.37$	$\mathbf{72.25}\% \pm 0.17$
NarrowMoE-{}-2-10	$70.71\% \pm 0.25$	$71.76\% \pm 0.50$	$70.63\% \pm 0.44$	$\mathbf{71.87}\% \pm 0.30$
NarrowMoE-{}-3-4	$71.45\% \pm 0.24$	$\mathbf{71.65}\% \pm 0.43$	$71.45\% \pm 0.40$	$71.54\% \pm 0.22$
NarrowMoE-{}-3-10	$70.72\% \pm 0.30$	$\mathbf{71.47}\% \pm 0.29$	$70.88\% \pm 0.20$	$71.43\% \pm 0.12$
NarrowMoE-{}-4-4	$71.52\% \pm 0.09$	$\mathbf{71.95}\% \pm 0.39$	$71.37\% \pm 0.04$	$71.80\% \pm 0.23$
NarrowMoE-{}-4-10	$70.70\% \pm 0.30$	$71.61\% \pm 0.16$	$71.16\% \pm 0.40$	$\mathbf{71.99}\% \pm 0.23$

Table 9: Mean accuracy and sample standard deviation for embedded MoE blocks in a ResNet-18 architecture with soft constrained gating networks. In all experiments, the gating network activates only the top $k = 2$ experts to maintain computational complexity. We highlight models with the highest accuracy on each level.

Table 9 states the test results on the CIFAR-100 test set for Basic and Shortcut models. Moving the ReLU function after the MoE output, as proposed in section 5.3, does not show performance increases. Therefore, we do not state them in this section to improve readability. Instead, we present full test results in Appendix D.1.

Models with shortcut connections always outperform equivalent basic models. We conclude that shortcuts support the MoE learning process, and it is also easier for MoE blocks to fit residual mappings $\mathcal{H}(x) - x$ instead of $\mathcal{H}(x)$. Shortcut models with four experts outperform basic models by about 0.5 percentage points, except in position 3, where both variants are very close. For models with ten experts, the differences are even more significant. We, therefore, analyze only the MoE models with shortcut connections further in following sections.

We point out that models with four experts almost always outperform models with ten experts. This result seems unexpected since ten experts offer a much larger capacity and theoretically enable each expert to specialize in a smaller domain. We reason that models with MoE layers in lower positions do not benefit from an increased capacity because the range of low-level features is much smaller than for high-level features. This assumption is supported by the fact that differences between models with four and ten experts decrease the deeper we insert the MoE blocks.

Another reason may be that each of the ten experts only sees about half the number of samples during training compared to models with four experts[6]. We can partly compensate for this effect by increasing the training epochs and adapting the learning rate schedule. However, an increase in epochs so that each expert sees a comparable number of training samples is not beneficial since the whole model converges at a specific point. No further performance improvements are apparent in later epochs.

We notice that both auxiliary losses deliver similar results. Performance differences are not significant and probably due to random processes during training. Also, the baseline achieves higher test accuracy than most models with embedded MoE blocks. Only the shortcut model with KL divergence loss in position 1 beats the baseline slightly but not significantly.

We point out that MoE models perform better with embeddings in lower positions than higher positions. MoE blocks embedded in position 1 deliver the best performance. On the other hand, the worst evaluation results are achieved by models with embeddings in position 3. The decreasing performances with increased MoE layer positions are consistent with our results using single layer experts in section 5.4.

[6]Average samples per expert are computed by $\frac{k}{numberofexperts} * epochs$.

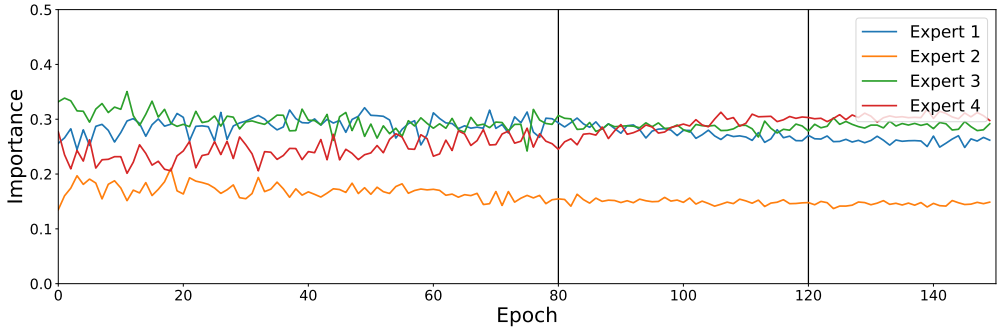
Model		Importance Loss			KL divergence Loss		
		$CV_{activation}$	$CV_{importance}$	Living	$CV_{activation}$	$CV_{importance}$	Living
NarrowMoE-{}-1-4	Default	7.92%	4.03%	4	5.65%	3.93%	4
	Shortcut	6.72%	3.82%	4	9.23%	6.36%	4
NarrowMoE-{}-1-10	Default	13.93%	13.22%	10	15.91%	15.26%	10
	Shortcut	10.69%	10.43%	10	14.93%	13.47%	10
NarrowMoE-{}-2-4	Default	7.09%	0.86%	4	6.75%	1.82%	4
	Shortcut	10.24%	2.44%	4	10.55%	2.99%	4
NarrowMoE-{}-2-10	Default	8.24%	7.15%	10	8.70%	7.19%	10
	Shortcut	7.25%	8.63%	10	8.62%	8.56%	10
NarrowMoE-{}-3-4	Default	5.75%	2.96%	4	7.06%	2.52%	4
	Shortcut	5.87%	2.17%	4	9.35%	2.21%	4
NarrowMoE-{}-3-10	Default	17.84%	15.71%	9.66	9.83%	6.66%	10
	Shortcut	6.43%	5.83%	10	10.51%	5.10%	10
NarrowMoE-{}-4-4	Default	8.15%	2.88%	4	10.00%	4.79%	4
	Shortcut	7.97%	2.86%	4	8.41%	2.79%	4
NarrowMoE-{}-4-10	Default	8.73%	7.34%	10	10.42%	6.94%	10
	Shortcut	4.80%	4.48%	10	8.95%	6.67%	10

Table 10: Expert utilization for different soft constrained models with embedded MoE blocks. We compute statistics during the evaluation on the CIFAR-100 test set. They state the coefficients of variation for the assigned number of samples and importance per expert during test time. The column *Living* states the number of experts that receive at least 1% of average weightings. We average all results over the three runs for each model setup.

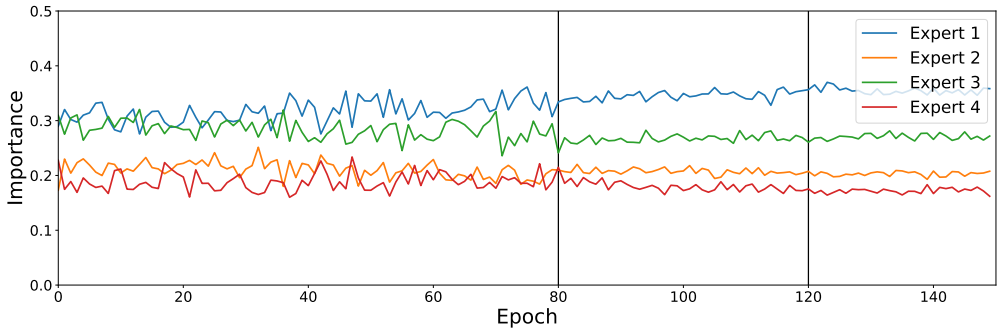
We also analyze expert utilization for different model setups. For this reason, we compute the coefficients of variation $CV = \frac{\sigma}{\mu}$ for the number of samples and total importance assigned to each expert during evaluation. The coefficient of variation is a measure of dispersion standardized by taking the mean into account. It enables analysis of the dispersion of frequency distributions with varying expectation values. A larger CV expresses higher variances in expert utilizations. Besides, we state the average number of living experts during evaluation. We classify an expert as alive if it receives at least 1% average weighting on the test set. We state expert utilization metrics in Table 10, measured during the evaluation on the CIFAR-100 test set.

Almost all models show full expert utilization with no dying experts. It demonstrates that both loss functions are generally able to overcome the dying expert problem. Only a single NarrowMoE-Imp-3-10 without a shortcut connection lets one expert die. That also distorts the coefficients of variation for this model setup. In most cases, models using KL divergence loss show higher values for $CV_{activation}$ than equivalent models using importance loss. $CV_{importance}$ results are with a few exceptions comparable for all models.

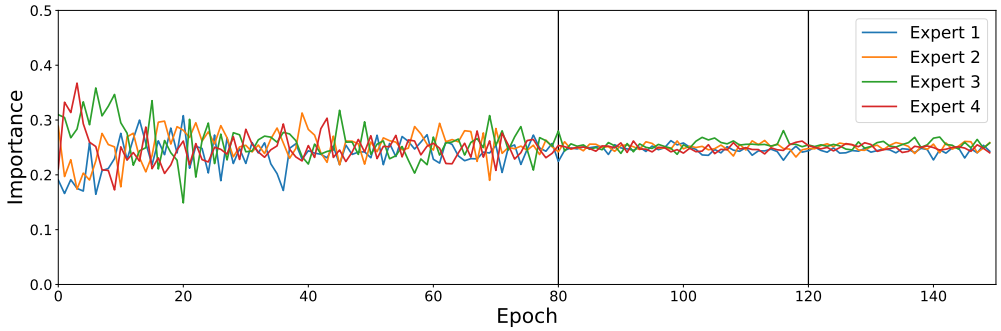
Note that both loss functions try to minimize $CV_{importance}$. The loss functions do not directly influence $CV_{activation}$ since equal importance may still allow very different numbers of samples assigned to each expert. $CV_{importance}$ values decrease for models with shortcut connections except position 2 and position 1 with KL divergence loss.



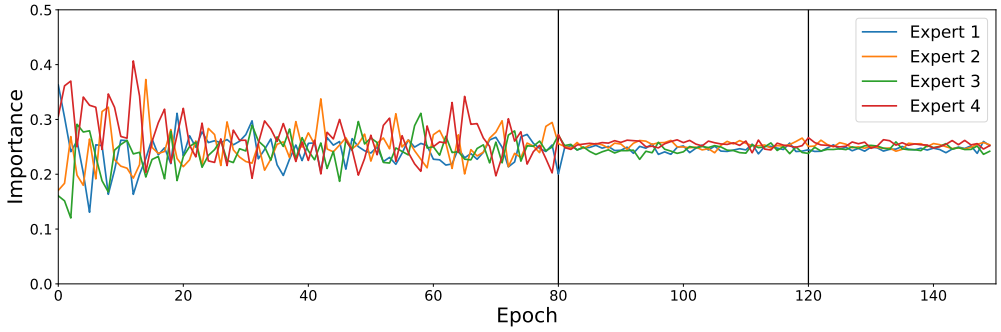
(a) NarrowMoE-Imp-1-4



(b) NarrowMoE-KL-1-4



(c) NarrowMoE-Imp-4-4



(d) NarrowMoE-KL-4-4

Figure 24: Average importance assignment on the CIFAR-100 test set at different training epochs. We train all models with auxiliary losses. The models contain embedded MoE layers with NarrowResNet-Experts in positions 1 and 4, respectively.

That may indicate that models with shortcut connections make it easier to optimize expert utilization by auxiliary losses based on importance. We also notice that both coefficients of variation are significantly larger for models with ten experts in position 1 than for models with four experts. Similar to accuracy values, coefficients of variation approach each other for embeddings in deeper positions.

We further illustrate the effect of the soft constrains during the training process in Figure 24. We measure expert utilization after each training epoch as the average assigned importance on the CIFAR-100 test set. We state the expert utilization for the models with the best test accuracy. Black vertical lines in the graphs mark the learning rate decreases. The figures demonstrate that both loss functions similarly handle expert utilization. No significant differences between the loss functions are recognizable.

The higher assignments to three of four experts in Figure 24a are no structural property of importance loss and do not occur in similar runs. The figures further confirm that expert utilization varies less for MoE blocks in deeper positions. Assigned importance stabilizes for reducing the learning rate from 10^{-3} to 10^{-4} after 80 epochs. Note that training with a reduced initial learning rate of 10^{-4} decreases the final models' test accuracy.

5.6 Experiments with Hard Constrained MoE Blocks

Next, we train models similar to section 5.5 with relative and mean importance constraints. Since models with shortcut connections significantly outperform other variants in section 5.5, we only train *Shortcut* models combined with hard constraints. We set thresholds to $m_{rel} = 0.5$ and $m_{mean} = 0.3$, equally to single layer MoE models in section 5.4. Further experimental details stay identical to training models with soft constraints and are stated in Appendix B.7.

Test results on the CIFAR-100 test set are stated in Table 11. Models constrained with mean importance perform better than other models, except for models with four experts and an MoE block in position 4. Note that higher accuracy does not come per se with a suitable expert utilization as we analyze in the next paragraph. While comparable soft constrained models show best performances in positions 1 and 2, hard constrained models perform mostly better for embeddings in positions 1 and 4. Similar to soft constrained models, MoEs with four experts almost always beat models with ten experts. Only models using mean importance with ten experts in position 4 form an exception. Mean importance models with four experts beat the baseline model for MoE blocks in positions 1 and 2, and with ten experts in position 4.

Model	Relative Importance	Mean Importance	Best Soft Const.
Baseline	$72.62\% \pm 0.29$		
NarrowMoE-{}-1-4	$72.21\% \pm 0.42$	**73.00**$\% \pm 0.40$,	$72.72\% \pm 0.36$
NarrowMoE-{}-1-10	$71.60\% \pm 0.09$	**72.28**$\% \pm 0.15$	$71.51\% \pm 0.23$
NarrowMoE-{}-2-4	$72.18\% \pm 0.48$	**72.95**$\% \pm 0.35$	$72.25\% \pm 0.17$
NarrowMoE-{}-2-10	$71.51\% \pm 0.44$	**72.08**$\% \pm 0.32$	$71.87\% \pm 0.30$
NarrowMoE-{}-3-4	$72.05\% \pm 0.15$	**72.61**$\% \pm 0.37$	$71.65\% \pm 0.43$
NarrowMoE-{}-3-10	$70.82\% \pm 0.16$	**72.05**$\% \pm 0.61$	$71.95\% \pm 0.39$
NarrowMoE-{}-4-4	**73.10**$\% \pm 0.25$	$72.57\% \pm 0.31$	$71.95\% \pm 0.39$
NarrowMoE-{}-4-10	$72.84\% \pm 0.30$	**73.09**$\% \pm 0.35$	$71.99\% \pm 0.23$

Table 11: Mean accuracy and sample standard deviation for embedded MoE blocks in a ResNet-18 architecture with hard constrained gating networks. In all experiments, top $k = 2$ experts are activated to maintain computational complexity. We highlight models with the highest accuracy on each level.

Relative importance constrained models with four experts significantly beat the baseline for embeddings in position 4. Like soft constrained models, the lowest performances are achieved in position 3. We assume that embedded MoE blocks work, constraint independently, better for positions early or late in a network.

We also analyze the expert utilization in Table 12. When comparing the number of living experts, we notice that both hard constraints have problems with dying experts. In particular, mean importance constrained models are unable to keep all experts alive in a single model. On the other hand, relative importance constraints manage to overcome the dying expert problem for models with four experts. However, the models still face problems with managing ten experts. Only NarrowMoE-Rel-4-10 shows improvements in the number of living experts and keeps all experts alive. Finetuning the threshold values depending on the model setup may improve expert utilization.

However, even in experiments with varying threshold values, mean importance models show large dying expert problems. When we also take expert utilization into account, the top performance by models with mean importance is caused by using only a part of all available experts. We assume that performance improvements are based on the fact that only a few experts are alive during training and consequently trained on a larger number of training samples than models with more experts alive.

Both hard constraints yield significantly higher values for $CV_{activation}$ and $CV_{importance}$ compared to soft constrained models, as stated in Table 10 in section 5.5. Dying experts favor high coefficients of variation since they receive only a small number of samples and importance. Vice versa, the assignments to alive experts and variances increase. We can eliminate this effect partially by only comparing models with all experts alive.

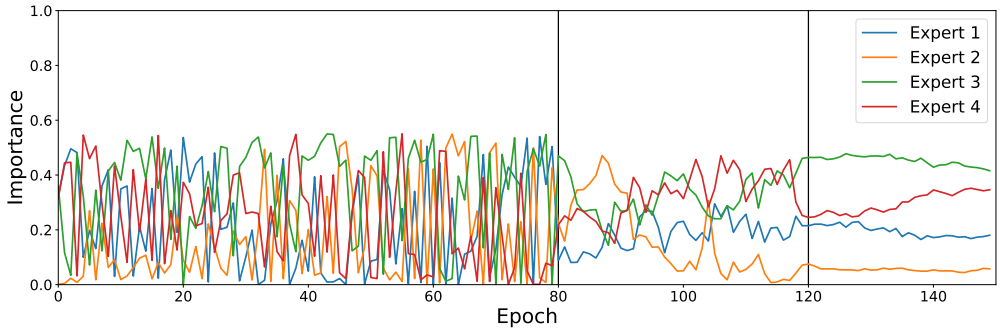
Model	Relative Importance			Mean Importance		
	$CV_{activation}$	$CV_{importance}$	Living	$CV_{activation}$	$CV_{importance}$	Living
NarrowMoE-{}-1-4	56.00%	56.34%	**4.00**	97.37%	103.23%	2.33
NarrowMoE-{}-1-10	160.60%	168.11%	5.33	199.77%	200.02%	2.00
NarrowMoE-{}-2-4	52.20%	51.61%	**4.00**	78.32%	97.60%	3.00
NarrowMoE-{}-2-10	165.66%	186.78%	5.67	170.56%	182.24%	3.00
NarrowMoE-{}-3-4	43.19%	25.21%	**4.00**	90.98%	98.32%	2.33
NarrowMoE-{}-3-10	125.13%	144.25%	6.67	166.65%	158.01%	3.33
NarrowMoE-{}-4-4	13.39%	2.91%	**4.00**	62.99%	95.53%	3.00
NarrowMoE-{}-4-10	10.61%	9.37%	**10.00**	147.22%	158.82%	3.67

Table 12: Expert utilization for different hard constrained models with embedded MoE blocks. We compute statistics during the evaluation on the CIFAR-100 test set. They state the coefficients of variation for the assigned number of samples and importance per expert during test time. The column *Living* states the number of experts that receive at least 1% of average weightings. Models with all experts alive are highlighted. We average all results over the three runs for each model setup.

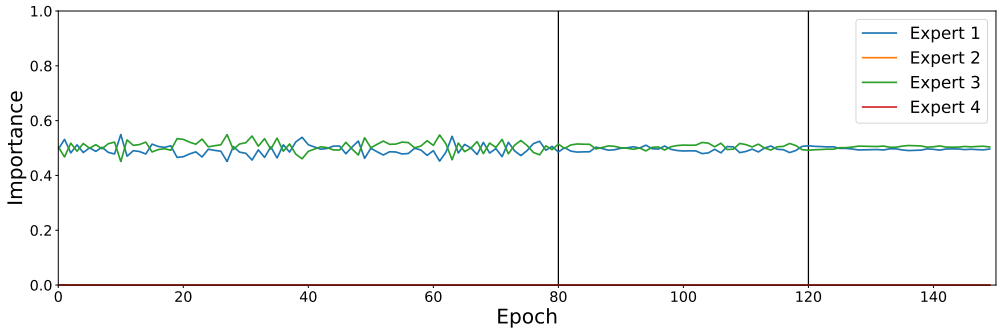
Still, relative importance models show significant higher variations compared to soft constrained models. It demonstrates that their expert utilization varies much more, and some experts are activated more frequently than others. Since mean importance models perform significantly better than soft constrained models with embeddings in deeper positions, it seems beneficial to allow the experts' importance to vary more during training instead of pushing importance to similar values by auxiliary losses.

Note that NarrowMoE-Rel-4-4 manages to reduce $CV_{importance}$ to a value comparable to soft constrained models. However, the model results in a higher $CV_{activation}$ value. Experts are assigned similar weights on average, but the number of assigned samples per expert still varies more for hard constrained models. One major drawback of hard constraints is their unstable expert utilization, especially compared to soft constraints. Since mean importance has significant problems with dying experts, we will exclude this constraint from some further experiments and concentrate on relative importance as hard constraints.

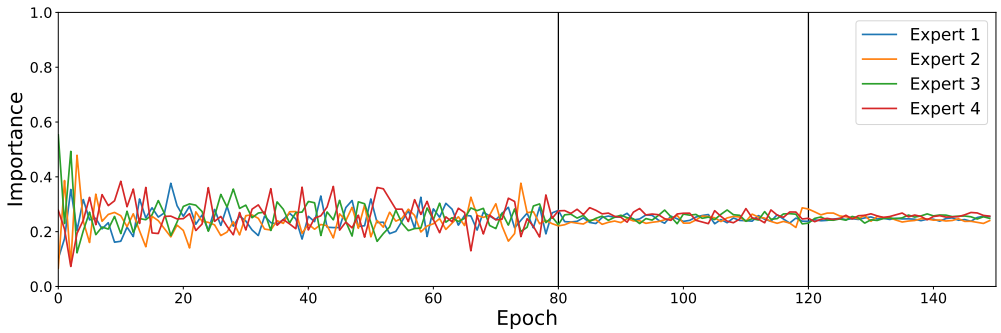
We plot the averagely assigned importance to each expert for different training epochs in Figure 25. Values are measured during evaluation on the CIFAR-100 test set. Note that we increased the value range of the ordinate from 0.5 to 1.0. For relative importance models, the effect of a decreasing learning rate is similar to soft constrained models and stabilizes expert utilization. NarrowMoE-Rel-4-4 shows a robust training process, which stabilizes after the first few training epochs. On the other hand, mean importance models seem to be very stable during training. However, the models only keep two and three experts alive, respectively. The number of living experts and the tendency of assigned importance are determined in the first epochs and do not change until the end. It supports our assumption that accuracy achievements by mean importance constraints are due to the higher number of training samples each living expert receives.



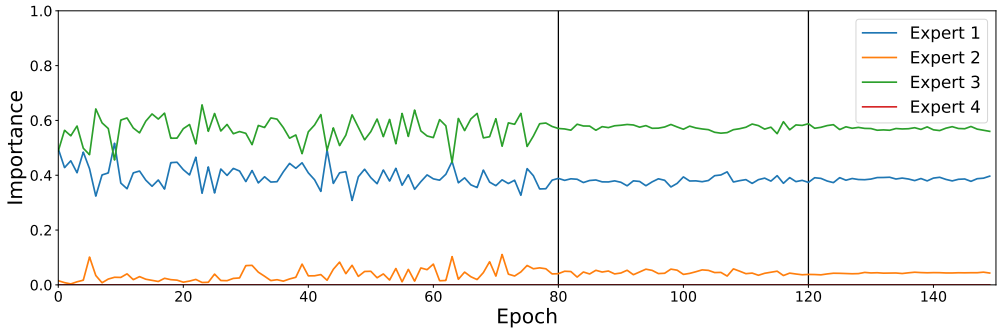
(a) NarrowMoE-Rel-1-4



(b) NarrowMoE-Mean-1-4



(c) NarrowMoE-Rel-4-4



(d) NarrowMoE-Mean-4-4

Figure 25: Average importance assignment on the CIFAR-100 test set at different training epochs. All models are trained with hard constraints. The models contain embedded MoE layers with NarrowResNetExperts in positions 1 and 4, respectively.

5.7 Visualization of Gating Network Decisions

This section analyzes how gating networks make decisions and assign weights to distinct experts. We use t-distributed Stochastic Neighbor Embedding (t-SNE) and Principal Component Analysis (PCA) to reduce the dimensionality of the gating network outputs to two dimensions. We then visualize results to gain insights into the decision process. We use soft and hard constrained models resulting from our experiments in the previous sections 5.5 and 5.6.

PCA [64] is a traditional dimensionality reduction technique computing a linear combination to project input vectors into a smaller subspace. Each principal component tries to capture as much unexplained variance of the higher dimensional representation as possible to minimize information loss during the transformation. Principal components are computed sequentially, each optimized on the previously unexplained variance.

In contrast to PCA, t-SNE [65] is a nonlinear dimensionality reduction technique. The method is based on Stochastic Neighbor Embeddings [66] that first converts the high-dimensional Euclidean distances between data points into conditional probabilities. These probabilities represent similarities between the data points and increase for nearby data points. The algorithm computes a low-dimensional conditional distribution for counterparts y_i and y_j of the high-dimensional datapoints x_i and x_j. For two distinct data points x_j and x_i the conditional probability $p_{j|i}$ is computed by $p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$ with σ_i being the variance of a Gaussian centered on data point x_i. The conditional probability $p_{j|i}$ could be interpreted as the probability that x_i would pick x_j if the neighbors were picked in proportion to their probability density given a Gaussian with center x_i.

The algorithm then minimizes the KL divergence $D_{KL}(P \parallel Q)$ between the two joint probability distributions P (high-dimensional space) and Q (low-dimensional space) using a gradient descent approach. Compared to basic Stochastic Neighbor Embeddings, t-SNE [65] offers a modified loss function and uses a Student t-distribution with a single degree of freedom to measure similarities between data points in the low-dimensional space. These changes ease optimization and improve visualization results. Joint probabilities are set to $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$ in the high-dimensional space. The corresponding joint probabilities in the low-dimensional space are then defined as $q_{ij} = \frac{(1+\|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1+\|y_k - y_l\|^2)^{-1}}$. Since the loss function is not convex, t-SNE results may differ for varying initializations.

We apply PCA and t-SNE on the gating networks' logits and reduce their dimensions to two. Additionally, we use t-SNE to reduce dimensions of resulting weight vectors, in which only two values are unequal zero since we activate $k = 2$ experts for each sample.

For the t-SNE algorithm, we set perplexity to 15, learning rate to 5, and the number of iterations to 1,500. Perplexity controls the variance σ_i of the Gaussian and is a smooth measure of the effective number of neighbors. Other hyperparameters of both algorithms are set to default values[7].

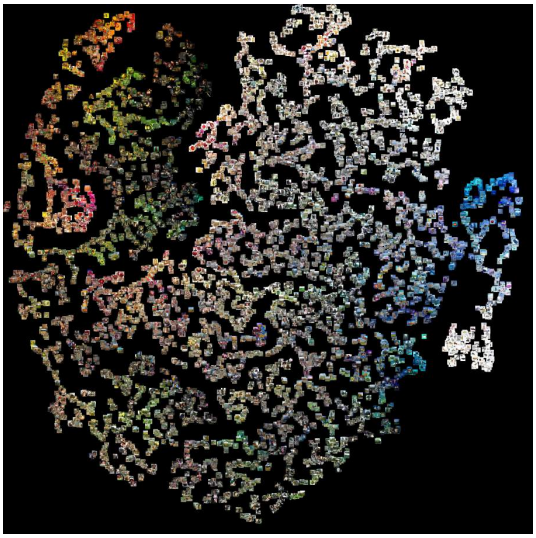
Figure 26 shows the t-SNE and PCA plots for the shortcut models NarrowMoE-Imp-1-4 (Figures 26a and 26b) and NarrowMoE-Imp-4-4 (Figures 26c and 26d). All four plots are computed on the raw gating network logits computed on the CIFAR-100 test set. We take the models with the highest test accuracy of each configuration. Plots for models trained with KL divergence loss lead to comparable results and are therefore not stated. We scale the resulting vectors v in both dimensions using min-max normalization $v' = \frac{v - v_{min}}{v_{max} - v_{min}}$ where v' is the rescaled value. We replace each data point in the resulting plot with its corresponding input image. Plots assigning different colors instead of images to each data point lack clarity due to the high number of classes and are therefore not stated. We provide larger plots of each Figure in Appendices D.2 (t-SNE plots) and D.3 (PCA plots).

Analyzing the t-SNE plot for NarrowMoE-Imp-1-4 in Figure 26a, it is noticeable that images with similar dominant colors are placed closer together than images of different colors. Vice versa, the gating network produces similar logits for images with similar dominant colors. The corresponding PCA plot in Figure 26b further shows that images with large shares of blue and orange/red, as well as white and green produce opposite outputs since images are placed in contrary corners.

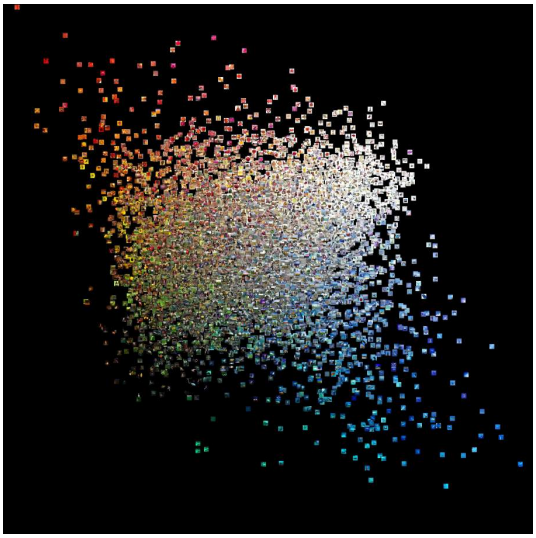
We conclude that gating networks close to the models' input layer assign weights mainly based on the input images' dominant colors. These results are not unexpected since our SimpleGate applies global average pooling on its input feature maps. Only low-level features have been computed to this point in the network, and colors dominate the feature maps. Note that only a single convolutional layer processes the input images before the gating network.

When comparing results to NarrowMoE-Imp-4-4 with an MoE block in position 4, a clear visual difference is recognizable. Images are no longer arranged primarily on their dominant colors. Still, there are regions identifiable in both the t-SNE (Figure 26c) and the PCA plot (Figure 26d) with dominant colors. However, the distinctions are much more faded, and the gating network's outputs rely less on dominant colors.

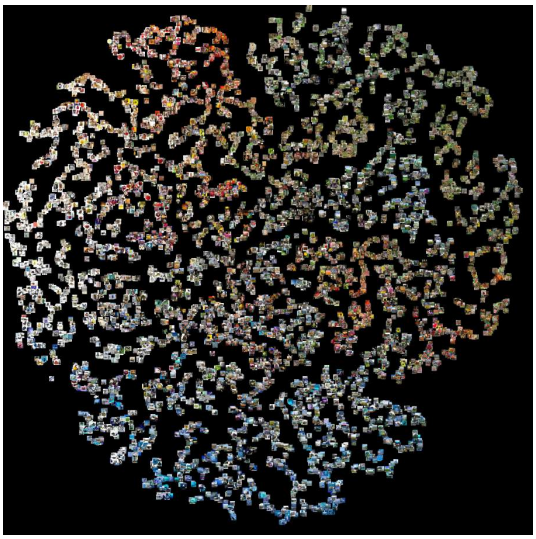
[7] See the API documentations for details on the hyperparameters:
`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`,
`https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`.



(a) t-SNE plot for NarrowMoE-Imp-1-4



(b) PCA plot for NarrowMoE-Imp-1-4

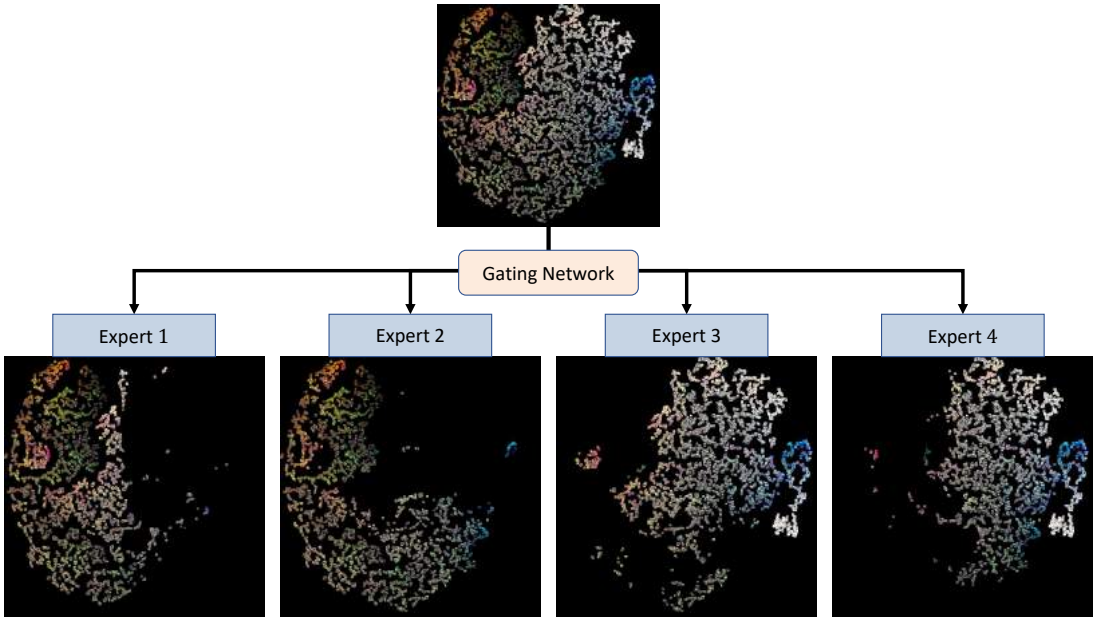


(c) t-SNE plot for NarrowMoE-Imp-4-4

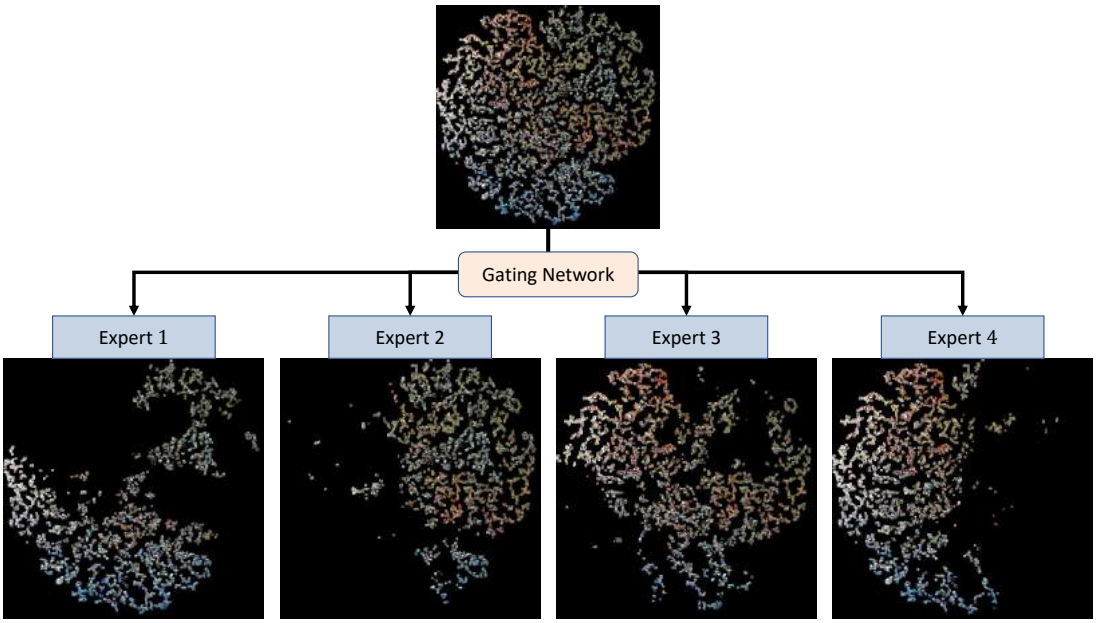
(d) PCA plot for NarrowMoE-Imp-4-4

Figure 26: Visualization of gating network logits for test samples using t-SNE and PCA. Images are created using models with the best test accuracy out of three runs. We use ResNet models with MoE block embeddings and four experts in positions 1 and 4, respectively. Models trained with KL divergence loss deliver similar visual results.

Figure 27 shows the resulting sample assignment to different experts for $k = 2$ active experts per forward pass. Figure 27a demonstrates for the NarrowMoE-Imp-1-4 model that the gating network divides the data into two major subdomains. Most images are either processed by experts 1 and 2, or experts 3 and 4. The gating network in NarrowMoE-Imp-4-4, on the other hand, varies more widely between different expert combinations. For example, images with strong blue tones placed at the bottom of the image are processed by all four experts similarly. The gating network in position 1 assigns blue and white images mainly to experts 3 and 4.



(a) NarrowMoE-Imp-1-4

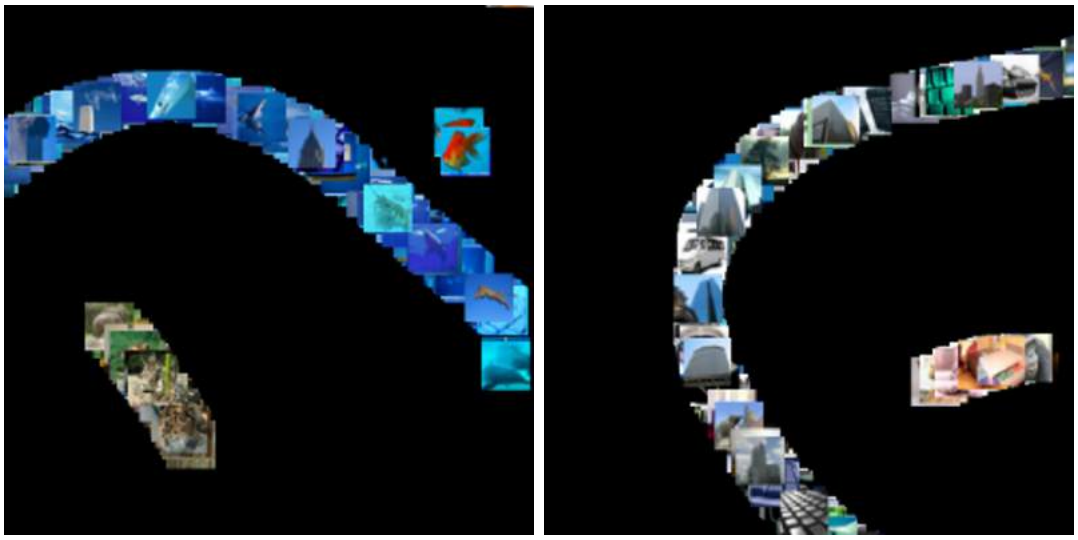


(b) NarrowMoE-Imp-4-4

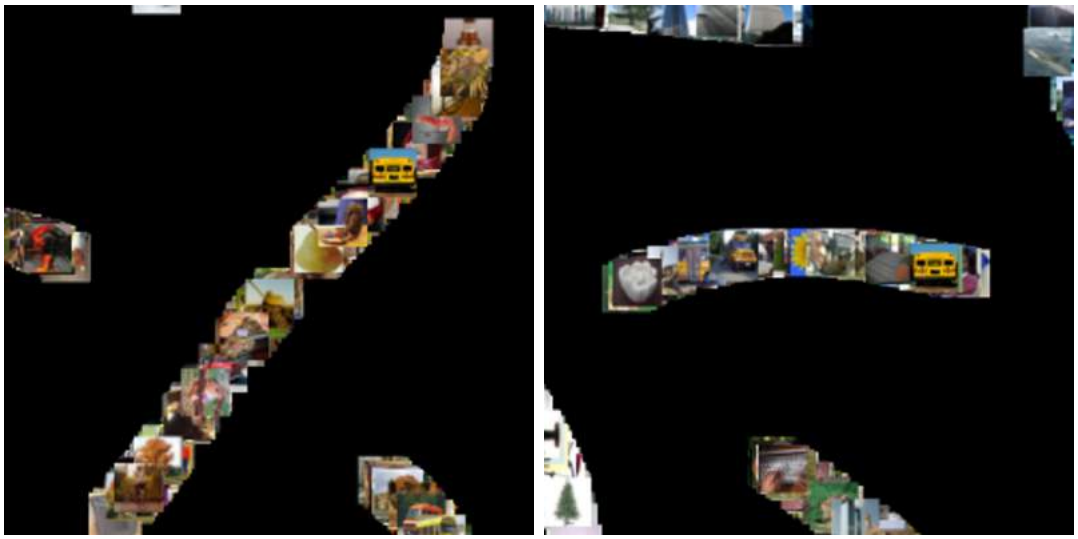
Figure 27: t-SNE plots that show the input samples' assignment to specific experts in soft constrained models. The gating network in position 4 (27b) shows a more diversified expert utilization compared to the gating network in position 1 (27a).

We conclude that weight assignments in deeper MoE layers is, in fact, based to a more considerable extent on high-level features and differentiate more strongly between experts. As we demonstrate in section 5.10, experts in position 4 are more specialized in specific subdomains than experts in position 1. So the gating process plays a crucial role in identifying reliable experts for each domain. Experts in position 1 are principally more generalized, so the selection process is less critical to extract meaningful features.

(a) NarrowMoE-Imp-1-4 (b) NarrowMoE-Imp-4-4



(c) NarrowMoE-Imp-1-4 (d) NarrowMoE-Imp-4-4



(e) NarrowMoE-Imp-1-4 (f) NarrowMoE-Imp-4-4

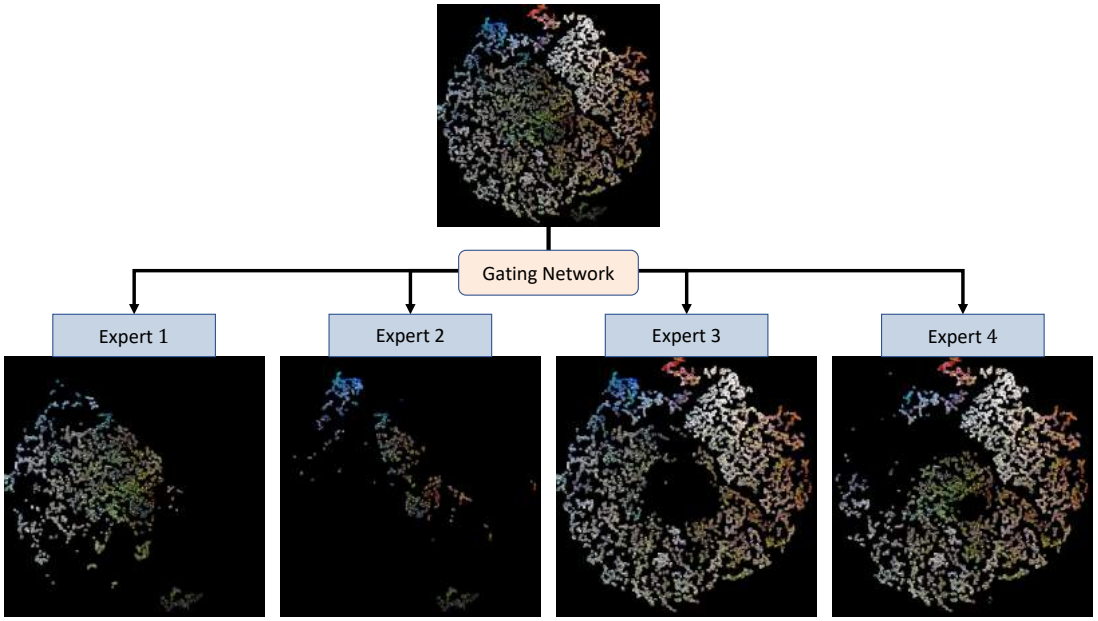
Figure 28: Selected excerpts of t-SNE plots for sparse weight vectors of soft constrained MoE models.

We apply the t-SNE algorithm on the resulting weight vectors of soft constrained models after setting all but top $k = 2$ weights to zero. We state in Figure 28 three hand-picked excerpts of both resulting plots with similar class labels and colors. The complete t-SNE plots are stated in Appendix D.4. Images in the left column belong to NarrowMoE-Imp-1-4, images on the right to NarrowMoE-Imp-4-4. It becomes clear that the gating network in position 4 puts a stronger emphasis on high-level features and similar image classes.

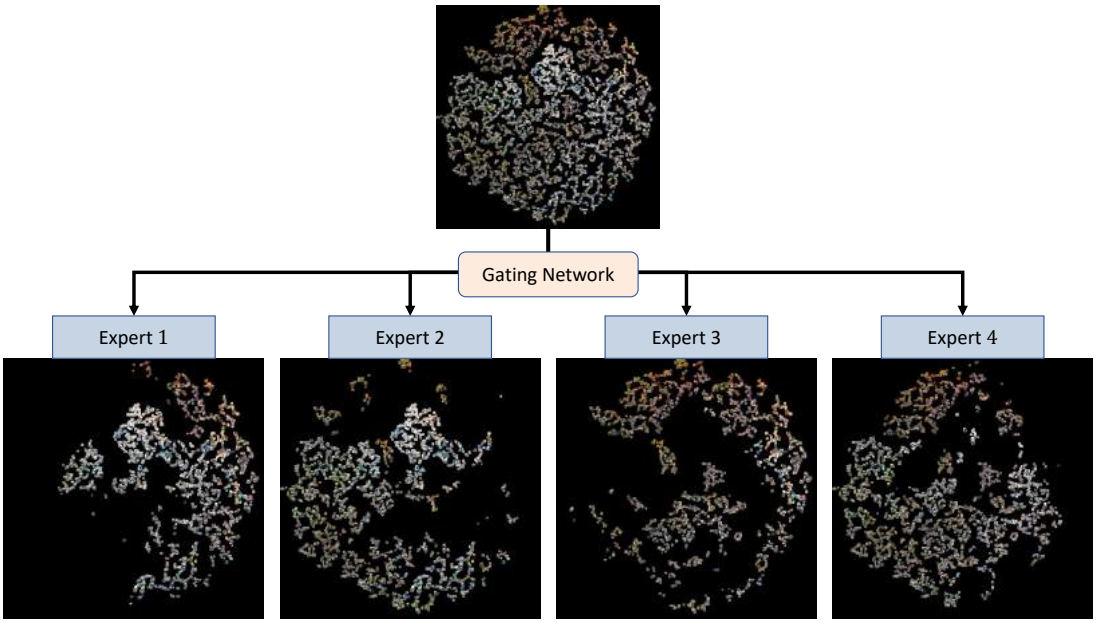
When comparing Figures 28a and 28b, both sample lines contain bikes, among others. Nevertheless, as for position 1, bikes are sporadically put together with images of other classes. In position 4, many bike images are put together with fewer other classes in between. Note that the gating network in position 4 still assigns similar weights to images of lobsters and other classes. However, the tendency to group similar classes and structures is more evident.

Similar results can be seen in images 28c and 28d, both containing images of skyscrapers. However, the gating network in position 4 assigns similar weights to images containing front views of skyscrapers from a pedestrian perspective. The corresponding gating network in position 1 assigns similar weights to skyscrapers mixed with different kinds of marine animals, both groups with dominant blue colors. Figures 28e and 28f in the last line demonstrate that the gating network in position 4 does not always assign similar weights according to similar high-level features. At least these features are not recognizable for humans. Still, there are a few images of school busses put next to each other, but mixed with flowers and other classes.

We also analyze gating visualizations for models trained with relative importance constraints. Sample arrangements in the plots based on gating network's logits are similar to soft constrained models. Plots for models with embeddings in positions 1 and 4 are stated in Appendices D.2 (t-SNE plots of logits), D.3 (PCA plots of logits), and D.4 (t-SNE plots of weights). Again, in position 1, clusters of images with similar dominant colors are identifiable in the t-SNE plot. However, the colors seem to be less clear separated than for soft constrained models. Similarly, plots for position 4 demonstrate that samples with similar structures or classes are also a bit more mixed. Visible structures are less significant compared to the soft constrained model.



(a) NarrowMoE-Rel-1-4



(b) NarrowMoE-Rel-4-4

Figure 29: t-SNE plots that show input samples' assignment to specific experts in hard constrained models. The gating network in NarrowMoE-Rel-1-4 mostly utilizes experts 3 and 4, while NarrowMoE-Rel-4-4 utilizes experts more equally in terms of the number of assignments.

We again plot the resulting sample assignments to different experts in Figure 29. In position 1, the hard constrained gating network mainly utilizes experts 3 and 4, as Figure 29a shows. Experts 1 and 2 are activated much less and are assigned only 36% and 12% of test samples, respectively. The gating network in position 4 shows better expert utilization, as Figure 29b demonstrates. Compared to soft constrained gating networks, the sample assignments seem more disturbed and irregular in the hard constrained gates.

Gating networks with ten instead of four experts produce similar visual results for soft and hard constraints. Except for minor differences in the assigning process, our analysis results in this section also apply to models with a varying number of experts. We exclude mean importance constrained models since they suffer from major dying expert problems. It prevents a sufficient analysis of gating decisions since all samples are assigned to only a few living experts.

5.8 Prediction Accuracy for Distinct Classes

After the visual analysis of the gating decisions, we examine the MoE behavior quantitatively. We compare the per-class accuracy for the baseline and MoE models with embeddings in positions 1 and 4. The complete class-wise test results for soft constrained models are stated in Appendix D.5 and for hard-constrained models in Appendix D.6. All results are averaged over three training runs. The models correspond to the experiments in sections 5.5 and 5.6. We highlight the results of MoE models that outperform the baseline models by at least three percentage points. We further state in Appendix D.7 the classes most frequently confused by the baseline models. We highlight falsely predicted labels that take a share of more than 10% of predictions in that specific class of test samples.

We first identify the classes for which the baseline models have the most problems to distinguish. We then analyze if the MoE models can differ better between these classes and consequently improve predictions. We identify the following groups of frequently confused classes: (baby, boy, girl, man, woman), (beaver, shrew, mouse), (bus, streetcar), (dolphin, shark), (bed, couch), (crab, lobster), (maple tree, oak tree, willow tree), (otter, seal), (poppy, tulip, rose) and (snake, worm).

NarrowMoE-Imp-4-4 shows performance improvements for classes (crab, lobster) and (maple tree, oak tree, willow tree) and outperforms the baseline on these classes. Note that the accuracy on the full test set is still almost 0.7 percentage points below the baselines. So the MoE blocks seem to enable the models to differentiate between these classes better, but at the cost of other class accuracies. NarrowMoE-KL-1-4 significantly outperforms the baselines for classes (snake, worm) and achieves comparable accuracy on the full test set. The soft constrained models show improved performances for some of the stated class combinations. However, there is no significant indication that MoE blocks improve the baseline model in a general way to differentiate better between visually similar classes.

For hard constrained models, NarrowMoE-Rel-4-4 shows improvements for classes (beaver, shrew, mouse). The models also show massive improvements for other classes like leopard, lizard, and possum and minor improvements for classes similar to these. Considering NarrowMoE-Mean-1-4, the model shows significant accuracy improvements for classes (beaver, mouse), and other animals. Otherwise, no further clear improvements for the class pairs are notable. Overall, NarrowMoE-Rel-4-4 and NarrowMoE-Mean-1-4 outperform the baseline models by about 0.5 percentage points.

Altogether, in some cases, MoE models can better differentiate between similar classes than baseline models and consequently increase prediction accuracy for these specific classes. However, this statement applies only to a few selected class-pairs and cannot explain why MoE models show better performance in some cases. We assume that the increased accuracy for some classes comes at the cost of decreased prediction accuracy for other classes. Each expert tends to predict some labels more frequently than others, resulting in an increased accuracy on these specific classes. A more precise weight assignment might overcome this problem and increase the overall test accuracy of the MoE models.

5.9 Varying Number of Active Experts

Next, we analyze how models perform for varying the number of activate experts k in each forward pass. Note, all models are trained with $k = 2$ and use shortcut MoE blocks. We evaluate performance on the CIFAR-100 test set and average the accuracy values on three runs. We compute metrics for models from sections 5.5 (soft constraints) and 5.6 (hard constraints). The results for MoE models with four experts are stated in Table 13.

The accuracy of most models increases for a larger number of active experts. Likewise, all models show decreased performances for only one active expert per forward pass. Except for mean importance constrained models, most models perform better for activating three experts in each forward pass. Activating the fourth expert in soft constrained models brings an additional accuracy boost. However, accuracy gains are smaller than for activating the 3rd expert. Accuracy differences between varying values for k are also more remarkable for models with MoE layers embedded in positions 3 and 4. Our results also demonstrate the effect of dying experts for mean importance constrained models. In all these models, at least one expert is assigned less than 1% importance. Consequently, activating three or four experts does not increase performances since additional experts are trained only on a few examples and cannot contribute to the overall prediction. Dead experts even decrease performances in lower positions since their predictions distort the combined MoE results.

Model	k=1	k=2	k=3	k=4	Averaging
Baseline			72.62%		
NarrowMoE-Imp-1-4	71.88%	72.24%	72.30%	**72.31%**	72.29%
NarrowMoE-Imp-2-4	71.59%	72.18%	72.19%	**72.22%**	71.62 %
NarrowMoE-Imp-3-4	70.17%	71.65%	**71.96%**	71.88%	70.05 %
NarrowMoE-Imp-4-4	69.24%	71.95%	72.48%	**72.73%**	71.14 %
NarrowMoE-KL-1-4	72.39%	**72.72%**	72.62%	72.61%	72.53 %
NarrowMoE-KL-2-4	71.79%	72.25%	**72.34%**	72.29%	71.68 %
NarrowMoE-KL-3-4	70.08%	71.54%	71.75%	**71.81%**	70.45 %
NarrowMoE-KL-4-4	69.38%	71.80%	72.29%	**72.41%**	70.29 %
NarrowMoE-Rel-1-4	72.15%	72.21%	**72.23%**	72.15%	72.18 %
NarrowMoE-Rel-2-4	72.14%	72.18%	**72.20%**	72.18%	72.12 %
NarrowMoE-Rel-3-4	71.97%	72.05%	**72.09%**	72.06%	71.89 %
NarrowMoE-Rel-4-4	72.01%	73.10%	73.35%	**73.44%**	**73.63%**
NarrowMoE-Mean-1-4	65.80%	**73.00%**	72.14%	53.30%	43.86 %
NarrowMoE-Mean-2-4	72.31%	**72.95%**	72.85%	72.76%	40.53 %
NarrowMoE-Mean-3-4	70.27%	**72.61%**	72.45%	72.16%	20.64 %
NarrowMoE-Mean-4-4	71.28%	72.57%	**72.81%**	72.80%	45.74 %

Table 13: Evaluation results for a varying number k of active experts on each forward pass in MoE models with four experts. Besides, we compute prediction results for activating all four experts and fix weights to 0.25 per expert to average their outputs.

Our results, excluding mean importance models, supports our assumption that experts in deeper positions learn more diversified high-level features and specialize in a more specific domain than experts in lower layers. We conclude that experts in positions 1 and 2 are all similarly able to extract meaningful low-level features. So it is less critical to have a high number of different experts next to the input layer. On the other hand, MoE blocks, embedded in deeper network positions, benefit from increasing the number of experts available. NarrowMoE-Rel-4-4, as the best performing model with four active experts, outperforms the baseline significantly by almost one percentage point. Interestingly, averaging the experts' outputs even increases the results slightly. It is the only case in which expert averaging performs better than using the computed gating weights for combination.

Changes in parameter k can bee seen as a trade-off between accuracy and computational complexity. We state in Table 14 the models' complexity in GMac for different positions and values of k . Note that vanilla ResNet-18 networks have a computational complexity of 0.56 GMac and are comparable to models using k=2 experts. So using *NarrowResNetExperts*, each additionally activated expert adds about 0.06 to 0.08 GMac.

We conduct the same analysis also on models with ten experts. The results are stated in Appendix D.8. The patterns are similar to models with four experts: performances for increasing k vary more for MoE layers in deeper positions. The best results are achieved for setting k between six and eight experts.

	k=1	k=2	k=3	k=4
Position 1	0.48 GMac	0.56 GMac	0.64 GMac	0.71 GMac
Position 2	0.49 GMac	0.56 GMac	0.63 GMac	0.7 GMac
Position 3	0.49 GMac	0.55 GMac	0.62 GMac	0.68 GMac
Position 4	0.49 GMac	0.55 GMac	0.62 GMac	0.68 GMac

Table 14: Computational complexity for NarrowMoE models with different embedding positions and a varying number k of active experts.

Again, soft constrained models show more extensive performance improvements for increasing k than comparable hard constrained models. Soft constrained models with embeddings in position 4 and $k \geq 5$ activated experts reach accuracy values above 73% with top values of 73.14% (NarrowMoE-Imp-4-4, k=9) and 73.22% (NarrowMoE-KL-4-4, k=6). These models perform significantly better than models with four experts and outperform the baseline models. When comparing test accuracy for models with $k = 2$ active experts, soft constrained models perform worse than hard constrained models. However, if we increase the number of active experts, performance differences get smaller. Still, hard constrained models with relative importance show comparable performances but achieve higher accuracy with less computational complexity.

We furthermore compare our results to models replacing the gating network with a static gating network that assigns equal weights to all experts. It is identical to an embedded ensemble averaging approach. Note that in this case, all four experts are activated on each forward pass. Results are also stated in Table 14 in the column averaging. For soft constrained models, expert averaging achieves similar results to $k = 2$ for position 1. For MoE layers in deeper positions, accuracy decreases and lies between results activating one and two experts. It is also notable that accuracy for mean importance constrained models collapses and falls below 46%. This is probably due to dying experts that strongly distort the results. Consequently, our MoE approach beats a simple averaging approach and illustrates the relevance of a trainable gating network.

For NarrowMoE-Rel-4-4, averaging increases the total accuracy and even shows the best performance of all analyzed models. The gating network is thus not able to combine different experts in the best possible way. Note that for relative importance constrained models, differences between an MoE with $k = 4$ active experts and averaging are smaller than in other models. We assume that due to the hard constraint approach deactivating specific experts for whole batches, experts are per se more generalized than experts in soft constrained models. Moreover, it is harder for the gating network to learn meaningful combinations of these experts. Since we average our result over three runs and sample standard deviation amounts to 0.32 and 0.29 for $k = 4$, improvements by averaging are not statistically significant.

5.10 Utilization of Distinct Experts

We further analyze how distinct experts of a single MoE model behave. For this reason, we only activate one specific expert during evaluation and assign all weights to it. We rely on NarrowMoE-Imp-4-4 and NarrowMoE-Rel-4-4 since both models achieve the best performances for k=4 and do not suffer from dying experts. Besides, we only take the models with the highest overall accuracy to avoid overlapping results by averaging over different training runs. Note that the models are the same we use in section 5.7 to create the t-SNE and PCA plots.

We state class-wise results for NarrowMoE-Imp-4-4 in Appendix D.10. For completion, we also state results for NarrowMoE-Imp-1-4 in Appendix D.9. To gain insights into the domains experts are specialized in, we state in Table 15 the classes for which distinct experts in NarowMoE-Imp-4-4 receive the most weights during evaluation. We further state the per-class test accuracy of distinct experts. Our results show that the gating network selects different experts for specific classes. Experts receiving the highest weights for a class also show high evaluation results. Note that experts perform only slightly worse for classes on which they do not receive the highest accuracy.

Furthermore, we can roughly assign each expert to a subdomain of classes: Expert 1 is focused on marine animals, vehicles, and outdoor scenes. Expert 2 is specialized in different animals ashore. Expert 3 focuses on flowers, fruits, and vegetables. And expert 4 performs best for furniture and household devices. Consequently, the MoE layer implicitly learned to create experts specialized in distinct subdomains. While traditional MoE architectures need prepared datasets for each subdomain, our models using an embedded MoE layer build these subdomains by itself.

We also investigate the reliability of expert selections. To get a better overview, we extract results for classes on which expert 1 is assigned the highest and lowest weights and state them in Table 16. For each expert E_i, we state accuracy Acc_i on the specific classes and the average assigned sparse weights $\bar{g}_i$ during test time. For each class, we highlight the highest accuracy achieved by a single expert. Additionally, the full MoE model results with k=2 experts activated are stated as Acc_{MoE}. We highlight results in this column if the MoE model outperforms every single expert.

Expert 1		Expert 2		Expert 3		Expert 4	
Class	Acc_1	Class	Acc_2	Class	Acc_3	Class	Acc_4
mountain	**87%**	leopard	**76%**	poppy	**74%**	chair	**89%**
dolphin	**65%**	forest	60%	orange	**87%**	cockroach	**85%**
shark	**56%**	porcupine	**69%**	rose	**73%**	telephone	**73%**
sea	**80%**	tiger	**78%**	tulip	**61%**	lamp	**62%**
whale	**74%**	mushroom	**71%**	sunflower	**88%**	bottle	**80%**
cloud	**80%**	raccoon	**76%**	hamster	**79%**	cup	**83%**
plain	**88%**	shrew	49%	apple	**89%**	wardrobe	**89%**
road	**90%**	kangaroo	**72%**	orchid	**71%**	clock	65%
bridge	**79%**	beaver	**58%**	sweet pepper	**62%**	plate	**67%**
skyscraper	**90%**	willow tree	**69%**	baby	**56%**	can	68%

Table 15: Comparison of weight allocation and expert performance in NarrowMoE-Imp-4-4. We measure the test accuracy with only specific experts activated on each evaluation run. We state results for the top ten classes for which the experts receive the most weightings. We highlight results in cases the expert outperforms other experts for the specific class.

Class	Acc_{MoE}	Acc_1	$\bar{g}_1$	Acc_2	$\bar{g}_2$	Acc_3	$\bar{g}_3$	Acc_4	$\bar{g}_4$
mountain	85%	**87%**	0.86	49%	0.08	27%	0.05	36%	0.01
dolphin	**67%**	**65%**	0.86	49%	0.05	18%	0.01	31%	0.09
shark	**59%**	**56%**	0.80	24%	0.05	38%	0.07	23%	0.08
sea	79%	**80%**	0.79	39%	0.01	31%	0.11	44%	0.09
whale	**74%**	**74%**	0.79	10%	0.02	17%	0.02	34%	0.17
orange	**89%**	67%	0.00	45%	0.02	**87%**	0.79	62%	0.20
hamster	**83%**	43%	0.01	65%	0.09	**79%**	0.68	72%	0.23
apple	87%	47%	0.01	35%	0.03	**89%**	0.62	77%	0.35
lion	**82%**	62%	0.01	**81%**	0.47	75%	0.39	69%	0.11
wardrobe	**92%**	82%	0.02	84%	0.04	86%	0.37	**89%**	0.58

Table 16: Evaluation results of distinct experts in NarrowMoE-Imp-4-4. The upper half shows classes for which expert 1 receives the highest average weightings. The lower half stated the classes for which expert 1 receives the lowest weights. We highlight the best performing expert for each class. Further, if the full MoE model performs better than each distinct expert, we also highlight these result.

The results in Table 16 demonstrate that expert 1 is assigned the highest weights for test samples of mountains and marine animals. For each of these classes, expert 1 outperforms all other experts significantly. So the gating network chooses the best performing expert for these domains. If we take all classes into account, the full MoE performs in 73 out of 100 classes at least as well as its best experts on this domain. That shows that the gating network can identify experts for distinct domains and suitable support experts with inferior performance. It is consequently able to reasonably combine the output feature maps to improve the whole model's predictions. Since in a quarter of all classes, the MoE performs worse than its best expert, there seems to be room for improvements in the weight assignment process. For this reason, we try to increase the gating network's complexity in section 5.11.

When we analyze the lower part of Table 16, expert 1 performs significantly worse for classes on that it receives the smallest weightings. Results also demonstrate that the gating network does not always assign most weights to the best performing models but can reasonably differentiate for each new sample. For instance, on test samples of the class wardrobe, the top-performing expert 4 receives an average weighting $\bar{g}_4 = 0.53$ and the slightly worse performing expert 3 an average weighting of $\bar{g}_4 = 0.37$. Note that all four experts achieve an accuracy of over 80%. The full MoE model then outperforms the distinct experts by at least three percentage points.

We further examine if the gating network assigns larger weights to experts for classes on which they perform better than other experts. Since that is crucial for high-level feature extracting experts, we restrict our calculations to NarrowMoE-Imp-4-4 and NarrowMoE-Rel-4-4. We compute the Pearson correlation coefficients[8] $\rho_{X,Y}$ to measure the linear correlation between two variables X and Y. It is defined as $\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$ and computes the standardized covariance between both variables. The Pearson correlation coefficient is a dimensionless measure with $-1 \leq \rho_{X,Y} \leq +1$.

A value of $\rho_{X,Y} = 1$ implies a perfect linear relationship between both variables, while $\rho_{X,Y} = 0$ signalizes no linear correlation [67, pp.309-310]. The Pearson correlation coefficient's distribution differs considerably from a 2-dimensional normal distribution for a smaller number of observations and larger absolute values $|\rho_{X,Y}|$. To compute the average over all classes, we use the Fisher z-transformation. This transformation normalizes approximately the correlation coefficient's distribution and enables us to compute statistics based on a set of correlation coefficients. Fisher's z-transformation of $\rho_{X,Y}$ is equal to $z = arctanh(\rho_{X,Y})$. Its corresponding inverse is $\rho_{X,Y} = tanh(z)$ [67, pp.740, 747].

We consider that the correlation between weight assignment and expert performance might be non-linear. For this reason, we also compute Spearman's rank correlation coefficient ρ_S. Spearman's ρ_S is a non-parametric measure of rank correlation and specifies the monotonic relationship instead of a linear relationship. It is based on the Pearson correlation coefficient and defined as

$$\rho_S = \frac{\sum_{i=1}^{n} \left(R(x_i) - \bar{R}(x)\right)\left(R(y_i) - \bar{R}(y)\right)}{\sqrt{\sum_{i=1}^{n} \left(R(x_i) - \bar{R}(x)\right)^2} \sqrt{\sum_{i=1}^{n} \left(R(y_i) - \bar{R}(y)\right)^2}} \tag{5.1}$$

with $\bar{R}(x) = \frac{1}{n} \sum_{i=1}^{n} R(x_i)$ and $\bar{R}(y) = \frac{1}{n} \sum_{i=1}^{n} R(y_i)$, respectively, as the arithmetic means of ranks $R(x_1), ..., R(x_n)$ and $R(y_1), ..., R(y_n)$ [68, pp.113-116].

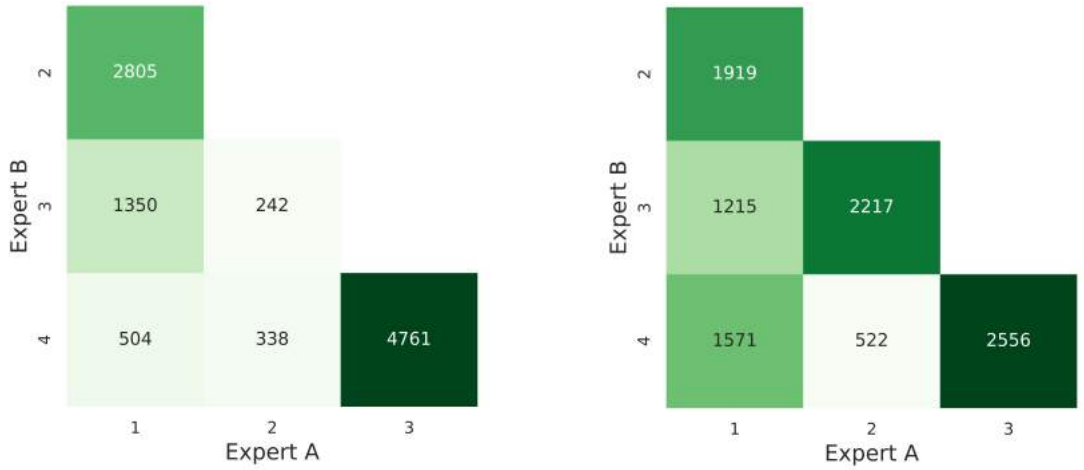
[8] also known as Pearson product-moment correlation coefficients

We again compute the average above all 100 classes using the Fisher z-transformation. For numerical reasons, we decrease correlation values of 1 to 0.99 before applying the Fisher z-transformation. Otherwise, perfect correlation values of 1 would induce a strong bias into the resulting average. We compute the correlations between the average assigned sparse weights and every single expert's test accuracy per class. The Pearson correlation results measuring the linear relationship between accuracy and assigned weights results to $\rho_{X,Y} = 0.8720$. It indicates a strong linear correlation between both values. The monotonic relationship with Spearman's correlation coefficient results in $\rho_S = 0.8870$, also indicating a strong relationship. We conclude that the gating networks assign larger weights to experts that provide better predictions on distinct domains.

Note that correlation does not imply a causal relationship between both variables. So we cannot prove statistically that the gating network assigns higher weights to experts because they surpass others. Particularly since we compute correlation measures on the test set not used for training the gating network, other factors may influence the weight assignment process. However, since the gating network is also optimized with cross-entropy loss, we assume that distinct experts' training accuracy plays a significant role in learning the weight assignments.

To complete the picture, we also compute correlation values for accuracy and weights computed by applying a softmax function to the gating network's raw outputs without deactivating experts. In this case, the averaged correlation coefficients result in $\rho_{X,Y} = 0.8705$ and $\rho_S = 0.8903$. Additionally, we compute the correlation between accuracy and the number of activations per expert. The number of activations is equal to the number of samples for which an expert receives one of the top-2 weights. The averaged results are $\rho_{X,Y} = 0.8501$ and $\rho_S = 0.8301$. These additional results underline the strong correlation between expert accuracy and its utilization during test time.

We state per-class test results for distinct experts in a NarrowMoE-Rel-4-4 model in Appendix D.11. Computing the averaged correlation coefficients on the sparse weighting results to $\rho_{X,Y} = 0.5827$, and $\rho_S = 0.5765$. Therefore, the hard constrained model shows a much smaller but still significant correlation between each expert's accuracy and received weights. Average correlation coefficients for non-sparse weightings, $\rho_{X,Y} = 0.5919$, and $\rho_S = 0.5823$ draw the same picture, but the monotonic correlation seems stronger. For the number of activations, correlation values are also significantly smaller: $\rho_{X,Y} = 0.5981$ and $\rho_S = 0.6172$. Note, both MoE variants show a significant positive correlation between expert utilization and performance.



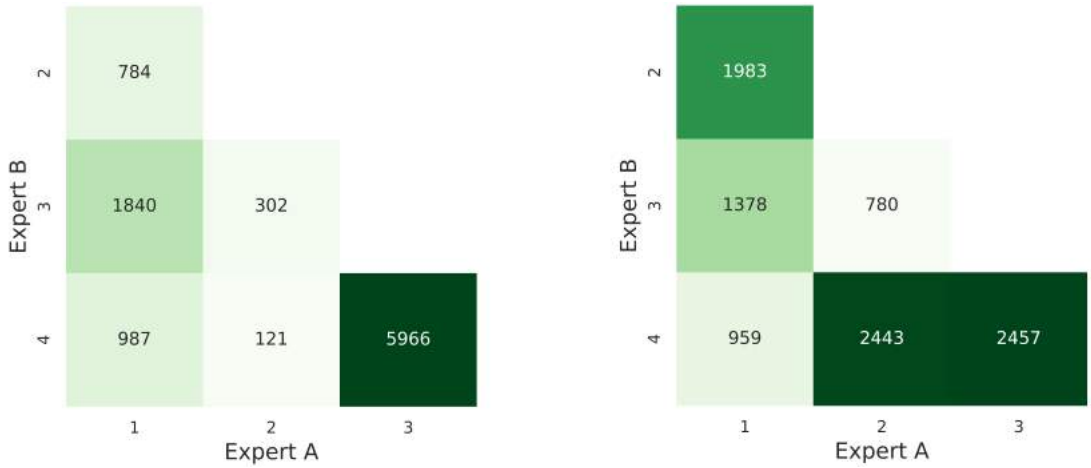
(a) NarrowMoE-Imp-1-4 (b) NarrowMoE-Imp-4-4

Figure 30: Number of activations on the CIFAR-100 test set for each 2-combination of experts. Results are stated for models with MoE layers in positions 1 and 4, respectively. Models are trained with importance loss and $w_{aux} = 0.5$.

Experts in NarrowMoE-Rel-4-4 are more generalized and do not show as large performance variations on different classes as NarrowMoE-Imp-4-4 experts. So the gating network seems to select different experts for different test samples of the same class instead of relying mostly on the same experts. Since we only compute correlations as per-class averages, it decreases the resulting correlation coefficients.

We also want to analyze if some expert combinations are activated more frequently than others. For this reason, we compute the number of activations for each 2-combination of experts on the test set. Results for NarrowMoE-Imp-1-4 and NarrowMoE-Imp-4-4 are stated in Figure 30. The results show that NarrowMoE-Imp-1-4 mainly uses expert combinations (3, 4) and (1, 2), which are selected in about three-quarters of all cases. Other expert combinations are utilized much less, (2, 3), for example, in only 2.5% of cases.

Expert utilization and combinations of experts are more evenly distributed in NarrowMoE-Imp-4-4. Still, some combinations are used much more frequently than others. These results demonstrate that those expert combinations are not randomly selected. In other words, the partner of the highest weighted expert is not chosen arbitrarily. It also shows that the gating network in position 1 has a stronger tendency to focus on two different expert combinations. In contrast, the gating network in position 4 varies more between different experts. These results quantitatively describe the sample assignments visualized in Figure 27 in section 5.7.



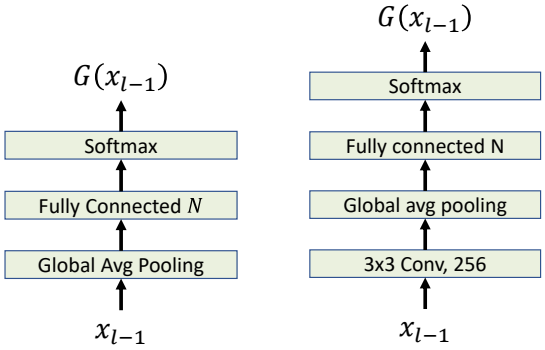
(a) NarrowMoE-Rel-1-4 (b) NarrowMoE-Rel-4-4

Figure 31: Number of activations on the CIFAR-100 test set for each 2-combination of experts. Results are stated for models with MoE layers in positions 1 and 4, respectively. Models are trained with relative importance constraints and $m_{rel} = 0.5$.

When we compare those results to models constrained by relative importance, a clear difference is notable. Results for NarrowMoE-Rel-1-4 and NarrowMoE-Rel-4-4 are stated in Figure 31. The gating network in position 1 is highly focused on the expert combination (3, 4) and uses this combination in about 60% of all cases. Figure 29a in section 5.7 provides the visualization of the gating decisions. As for soft constrained models, the gating network in position 4 shows a more diversified expert utilization comparable to NarrowMoE-Imp-4-4. We conclude that MoE layers embedded in deeper network positions utilize experts more generally and prevent a strong focus on single expert combinations. That is especially important for load balancing if expert networks are distributed on different computing units. In this case, a strong focus on a few experts may lead to an overload of some units while other units are underutilized.

5.11 Additional Training and Architecture Modifications

The previous sections demonstrate that the gating networks can reliably identify subdomains in the training data and select experts specialized in these subdomains. However, the results in section 5.10 also confirm that the gating networks do not always choose the best expert for individual input samples. So there is still room for improvement in the gating decision process. We assume that the architecture of our SimpleGate with only a single fully connected layer might be too simple.



(a) SimpleGate (b) ConvGate

Figure 32: Comparison between SimpleGate and ConvGate architectures. ConvGate adds a convolutional layer with 3×3 filters. The number of filters corresponds to the number of channels. For NarrowMoE-Imp-4-4 the layer learns 256 filters.

Therefore, we improve the sample assignment by increasing the gating networks' complexity. We focus on NarrowMoE-Imp-4-4 since we assume that improvements have a greater impact. That is due to more specialized experts in soft constrained models than in hard constrained ones. We rely on importance loss in this section since it behaves very similar to KL divergence loss.

We propose a new, more complex gating network architecture. We add a convolutional layer with filters size 3×3, stride 1, and ReLU activation to our SimpleGate architecture. The number of filters corresponds to the number of input channels. We refer to this gating network as *ConvGate*. Figure 32 visually compares the architectures of SimpleGate and ConvGate. Our first training runs with hyperparameters equal to the ones we use in section 5.5 suffer from the dying expert problem. Increasing w_{aux} also shows no significant improvements. Adjustments to the learning rate schedule eventually solve the problem for soft constrained models. We use a linearly increasing learning rate in the first 20 epochs to stabilize expert utilization and then continue with the same learning rate schedule as for MoEs blocks with SimpleGate networks.

Hard constrained models using ConvGate suffer massively from dying experts, even with a decreased learning rate. That demonstrates the more unstable expert utilization in hard constrained MoE models. We also train models without Gaussian noise in the gating network, denoted as *No Noise* in the results. Details on Gaussian noise are stated in section 2.3. As another experiment, we train the gating network with Gaussian noise for half the training time (75 epochs) and then freeze its weights. Therefore, the sample assignment stays constant, and experts learn on an identical subset of training samples in the remaining iterations. We denote these models with *Frozen Gate*. Experimental details for these experiments are stated in Appendix B.8.

Inspired by [26], we run experiments with an additional variance loss term L_{var} that aims to maximize the variance of distinct weight vectors. We define the loss function in Equation 5.2 as the negative average variance of raw weight vectors before deactivating experts. Variance is computed as the squared sample standard deviation for N experts. The average weight $\bar{g}$ is equal to $\frac{1}{N}$. This additional loss function aims to increase the variation between assigned expert weights g_i for each sample and avoids equal distributed weight assignments. We expect it to support distinct experts' specialization process since the selection of active experts per sample is more clearly. On the other hand, importance loss and KL divergence loss balance the expert utilization per batch and prevent dying experts. We weight the loss by $w_{var} = 0.01$ and denote the model with *Variance Loss*.

$$\mathcal{L}_{var} = -w_{var} \cdot \frac{1}{|X|} \sum_{x \in X} \left(\frac{1}{N-1} \sum_{i=1}^{N} (g_i - \bar{g})^2 \right) \tag{5.2}$$

Table 17 states the test results for all presented modifications. The increase in the gating networks' complexity improves the test accuracy by about 0.5 percentage points compared to the same model with SimpleGate. Still, the performance is slightly below the baseline model. The results also demonstrate that using Gaussian noise during training is beneficial to the model's accuracy. Besides, freezing the gating network after half the training time does not increase the models' performance and even decreases the accuracy slightly. Finally, adding variance loss also decreases the test accuracy slightly. We further increased w_{var}, but we find that to degrade the model's accuracy and, from a certain point on, to destabilize expert utilization resulting in dying experts.

Model	Importance Loss
Baseline	$72.62\% \pm 0.29$
NarrowMoE-Imp-4-4	$71.95\% \pm 0.39$
NarrowMoE-Imp-4-4 ConvGate	$72.42\% \pm 0.27$
NarrowMoE-Imp-4-4 ConvGate - No Noise	$72.24\% \pm 0.28$
NarrowMoE-Imp-4-4 ConvGate - Frozen Gate	$72.22\% \pm 0.32$
NarrowMoE-Imp-4-4 ConvGate - Variance Loss	$72.26\% \pm 0.42$

Table 17: Accuracy of NarrowMoE-Imp-4-4 models using ConvGate instead of SimpleGate. We also try some finetuning during training. We first train models without Gaussian noise in the gating network (No Noise). In another experiment, we freeze the gating network's weights after half the training time (Frozen Gate). As our last modification, we add variance loss during training (Variance Loss).

We take the best performing ConvGate model with Gaussian noise and compute again each distinct expert's test accuracy and utilization. The class-wise results are stated in Appendix D.12. Comparing results to NarrowMoE-Imp-4-4 using SimpleGate, see Appendix D.10, the distinct experts in the ConvGate model perform about four percentage points worse on average. Still, the full MoE model achieves 72.70% overall accuracy compared to 72.39% for the best soft constrained model with SimpleGate.

Also, the full ConvGate MoE model performs better than each distinct expert in 82 of 100 classes, compared to only 71 classes for the SimpleGate counterpart. The ConvGate model surpasses the SimpleGate model in 52 classes and is surpassed by it in 42 classes. Comparing the performance per class, we cannot detect a clear pattern or tendency for specific classes or domains. We assume the improved performance results from better expert selection and weight assignments for specific cases.

We visualize the assignment process with t-SNE plots, as described in section 5.7. Figure 33 shows the resulting plots. We can visually divide the space into four areas by drawing straight lines between opposite corners. Each distinct expert processes mainly samples from two coherent areas out of the four areas. Further, each area is processed by two different experts. When we compare the assignments to the results in section 5.7, the ConvGate subdivides the assignments much clearer. Consequently, the gating network produces more unambiguous weight vectors and selects experts more definitely.

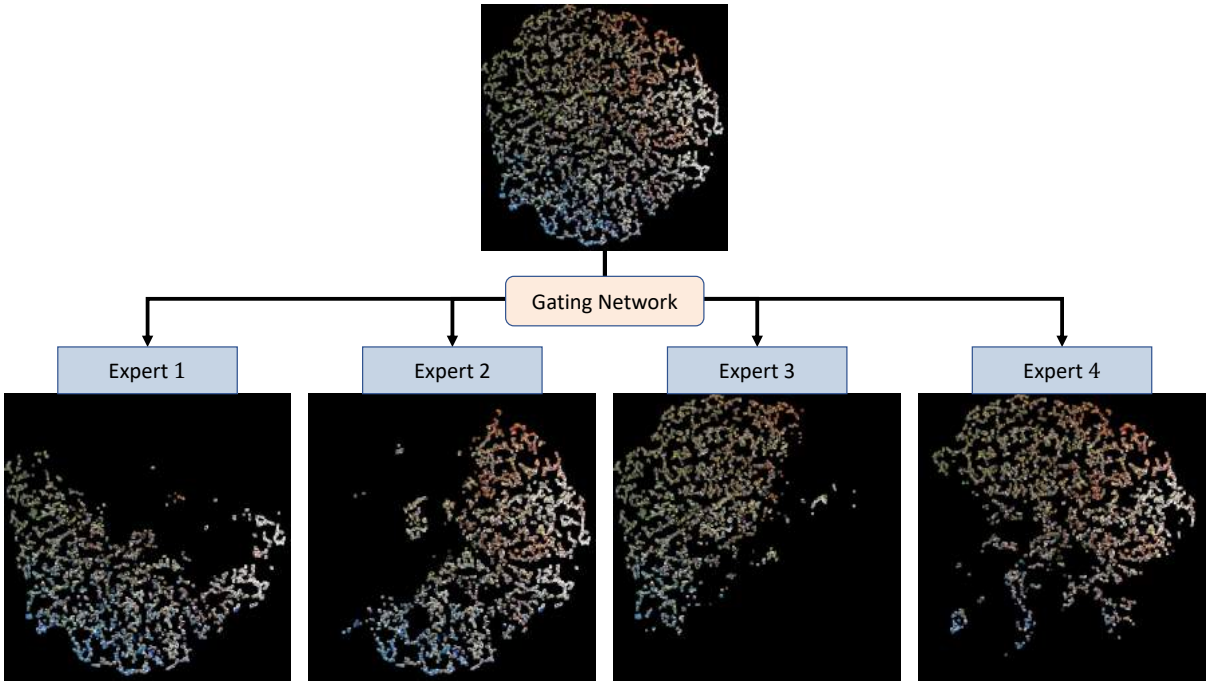


Figure 33: t-SNE plots that show the input samples' assignment to specific experts in NarrowMoE-Imp-4-4 ConvGate model. The model uses ConvGate as gating network that results in more clearly divided expert weights.

We finish this chapter with a short potential analysis. We compute two benchmarks that may be achieved by selecting the best experts for each sample. As our first benchmark, we take the best performing expert's accuracy and compute the average over all classes, which amounts to 69.50%. Note that our best ConvGate MoE model with $k = 2$ active experts already achieves 72.70% test accuracy and beats this benchmark significantly.

With our second benchmark, we consider that the gating network might place confidence in the wrong experts and better experts are available. In each case, the MoE model makes a false prediction, we analyze if activating a single distinct expert might correct it. Is this the case, we consider it a correct prediction. The resulting benchmark accuracy is 83.63%. It demonstrates that there is much room for improvement in the gating network decisions without any further changes in experts or basic network architecture.

Keeping this benchmark in mind, we try continuing training our model and freeze all weights but the gating network itself. We also set $w_{aux} = 0$ and train for 20 epochs with different learning rate schedules. However, no improvements in terms of test accuracy are visible. Since an omniscient gating network computes the second benchmark, it should be seen as a theoretical upper bound for an MoE models's performance. Also, that benchmark does not consider that even if all distinct experts make false predictions on their own, there still might be weighted combinations to achieve the correct prediction. Without experimental evidence, a theoretical omniscient gating network might even outperform this second benchmark.

6 Object Detection Enhancements with MoE Layers

We continue our experiments with embedded Sparsely-Gated Mixture-of-Experts Layers in this chapter. Our image classification experiments with CIFAR-100 in the previous chapter provide many valuable insights into MoE layers' behavior. However, those models have only limited practical applicability. Therefore, we change our focus to object detection, widely used in autonomous driving, robotics, and medical image computing. All of our models in this chapter are based on RetinaNet, introduced in section 3.4. We train all models on the COCO dataset, which we introduce in section 6.1. The officially associated COCO evaluation metrics are explained in section 6.2.

We rely on a reimplementation of RetinaNet and compare our MoE models to a pre-trained vanilla RetinaNet. Evaluation results for our baseline and other one-stage object detectors are stated in section 6.3. Our primary focus lies in replacing the regressor and classifier subnets with embedded MoE layers. We expect these MoE layers to have the most considerable effects in the decision part of a model. The resulting model architectures and their performances are then presented in section 6.4. The following section 6.5 provides a qualitative analysis of the prediction behavior of distinct experts in the regression subnet. We complete these analyses with a quantitative evaluation of the regressor MoE behavior in section 6.6. The classification counterpart is similarly analyzed in section 6.7. We further investigate in section 6.8 the effect of using pre-trained weights for expert models. The last section 6.9 concludes this chapter with a review of MoE layers embedded in the feature extractor network.

We denote models with MoE layers embedded in the regressor and classifier subnets as *DetectorMoE*. Similar to chapter 5, we state the training constraint and the number of experts in the models' names as *DetectorMoE-Constraint-NumberOfExperts*. Models only replacing the fourth convolution layer in both subnets are labeled with postfix *Conv4*. Similarly, we refer to models with MoE layers embedded into the feature extractor as *ExtractorMoE-Constraint-Position-NumberOfExperts*. We again abbreviate constraints as *KL* (KL divergence loss), *Imp* (importance loss) and *Rel* (relative importance constraint). We refer to the corresponding MoE layers with deep experts also as MoE blocks. The hardware and software specifications for our experiments are stated in Appendix A.

6.1 COCO Dataset and Preprocessing

We use the Microsoft Common Objects in Context (COCO) dataset for object detection [45]. All models are trained on the 2017 training set containing 118,287 images with 80 distinct object classes. We test our models on the COCO test-dev split from 2017, which has no public labels available. Metrics are computed by uploading detection results onto an evaluation server[9]. We also measure some metrics on the 2017 validation set containing 5,000 images due to the limited number of evaluations allowed on the evaluation server.

The authors built a dataset that shows objects in non-iconic views, for example, in the background or partially covered by other objects. COCO tries to reflect daily scenes with different objects in their natural environments. It enables algorithms to take contextual information into account. Classes are entry-level categories, for example, *dog* or *cat*, and inspired by indoor and outdoor objects most recognizable by children ranging from four to eight years. Subclasses like German shepherds are not part of the dataset. Figure 34 visualizes randomly selected samples from the dataset.

Figure 34: COCO sample images from the validation set 2017. Size, aspect ratio, and perspective vary between different images. Each image contains a variable number of objects from one or more classes. Image source: [45].

The COCO dataset is highly imbalanced since the number of images that contain an object of a specific class varies highly between different classes. About 54% of training samples contain a person, while the second most common object chair is only present in about 11%. We do not take class imbalance into account during training to see if MoE layers impact the detection of rare objects. Therefore, we apply no over- or undersampling to the dataset. Figure 40 in Appendix C.2 states a complete overview of the training data's class distribution.

Each image's true label consists of a variable number of sublabels, one for each object in the training sample. Each sublabel consists of five values: four values to define the BBox *(x, y, w, h)* and a fifth value specifying the object class. Image sizes and aspect ratios vary in both dimensions.

[9] COCO detection challenge for BBox detection (2019) is available at `https://competitions.codalab.org/competitions/20794`.

Because RetinaNet uses a ResNet backbone pre-trained on ImageNet [69], we consequently standardize all input images using the ImageNet statistics for $\mu = (0.485, 0.456, 0.406)$ and $\sigma = (0.229, 0.224, 0.225)$. We only use horizontal image flipping as data augmentation, as applied by the RetinaNet authors [53]. We then resize the images so that the smaller side has size 608. Exceeds the larger side a size of 1,024 after rescaling, the image resizing is reduced so that the larger side matches 1,024. After resizing, zero-padding is applied to each image's right and bottom, so dimensions are divisible by 32.[10]

6.2 COCO Evaluation Metrics

We use the twelve official COCO metrics to compare the performance of our object detector models[11]. All metrics are based on precision and recall values. First, precision and recall are computed separately for each class in the dataset. Predictions are assigned to a specific class if the confidence score exceeds a threshold. A BBox prediction is true positive (TP) if the IoU also exceeds a second threshold. Otherwise, the prediction is classified as false positive (FP).

If a ground truth is not detected because the prediction's confidence score is lower than the threshold, the detection is classified as false negative (FN). Precision and recall are then computed as $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$ for each class. Averages over all classes are called Mean Average Precision (AP) and Mean Average Recall (AR). AP and AR are also commonly abbreviated as mAP and mAR. We follow the notation of COCO and use AP and AR.

Six metrics are based on AP. Metrics $\text{AP}^{IoU=.50}$ and $\text{AP}^{IoU=.75}$ are the mean average precision over all classes using an IoU threshold of 0.5 and 0.75, respectively, during evaluation. Metric $\text{AP} = \text{AP}^{.50:.05:.95}$ averages over ten IoU thresholds between 0.50 and 0.95 with a step size of 0.05. This metric is the primary COCO challenge metric and is usually denoted just as AP[12]. Three additional metrics, AP^{small}, AP^{medium}, and AP^{large}, are available to compare detection performance for objects of different sizes. The metrics compute $\text{AP} = \text{AP}^{.50:.05:.95}$ for objects that cover areas smaller than 32^2 (AP^{small}), between 32^2 and 96^2 (AP^{medium}) and larger than 96^2 (AP^{large}).

[10] [53] uses size 600 for rescaling images. By using 608 as minimum size, padding can often be avoided.
[11] Metrics and descriptions are stated at `https://cocodataset.org/#detection-eval`
[12] The COCO challenge authors assume that different meanings of AP are clear from the context.

Metrics based on AR are also divided into two groups. The first three metrics $\text{AR}^{max=1}$, $\text{AR}^{max=10}$ and $\text{AR}^{max=100}$ compute the maximum recall given a fixed number of detections per image averaged over all classes and IoU thresholds. The last three metrics AR^{small}, AR^{medium} and AR^{large} compute the AR for different object sizes as stated before. All introduced metrics besides $\text{AR}^{max=1}$ and $\text{AR}^{max=10}$ allow a maximum of 100 top scoring detections per image across all classes.

6.3 RetinaNet Baseline

We train different RetinaNet models with embedded MoE layers. All models are based on an unofficial PyTorch reimplementation for RetinaNet [70][13]. We use a pre-trained model of the reimplementation as baseline. All models in this chapter use a ResNet-50 backbone and an image scale of 600. We choose to use ResNet-50 since it provides a proper balance between performance and training time. We train all models using $\gamma = 2$ and $\alpha = 0.25$ for focal loss.

Table 18 states the precision results of the baseline on the COCO test-dev2017 dataset, and compares it to the results of other object detection models. Note that not all papers include recall results. Differences between both RetinaNet probably result from different training settings. The reimplementation is optimized with Adam [10] instead of SGD, as stated in the paper. Besides, the learning rate schedule and batch size differ due to the number of GPUs available. We compare our following MoE models to the RetinaNet reimplementation since all our models are modified variants of it.

Model	AP	$AP^{IoU=.50}$	$AP^{IoU=.75}$	AP^{small}	AP^{medium}	AP^{large}
RetinaNet (ResNet-50) [53]	34.3%	53.2%	36.9%	16.2%	37.4%	47.4%
RetinaNet Reimplementation	35.0%	52.5%	37.3%	16.6%	37.5%	47.4%
SSD513 (ResNet-101) [47, 71]	31.2%	50.4%	33.3%	10.2%	34.5%	49.8%
YOLOv3 (Darknet-53) [51]	33.0%	57.9%	34.4%	18.3%	35.4%	41.9%
RetinaNet (ResNet-101) [53]	36.0%	55.2%	38.7%	17.4%	39.6%	49.7%

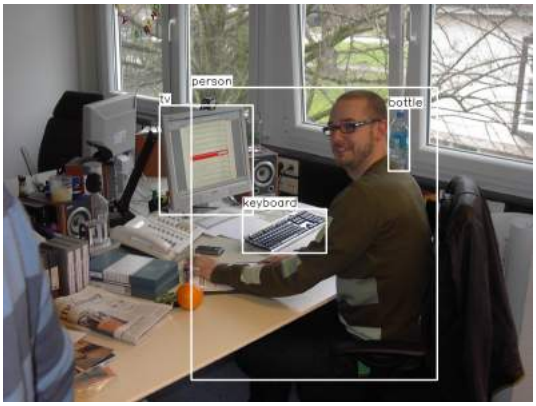
Table 18: Comparison of evaluation results for RetinaNet and comparable one-stage object detectors. All metrics are computed on the COCO test-dev2017 dataset. Differences between both RetinaNet variants probably arise from different training settings. RetinaNet results are stated for models with 600 pixels image scales. Note that most publications refer to evaluation results of RetinaNet models trained and tested on a scale of 800 pixels. The stated test results are taken from [51] and [53].

[13] Available at `https://github.com/yhenon/pytorch-retinanet` under Apache License 2.0.

We apply the baseline model to a few sample images to get an impression of the architecture's capabilities. All images are taken from Wikimedia Commons and are not part of the training data. Image sources and licenses are stated in Appendix C.3. We select images of different aspect ratios, resolutions, and domains. Detection results are then adjusted to the input image sizes. We plot results for four images in Figure 35. Since the model is trained on COCO, it detects 80 distinct classes.

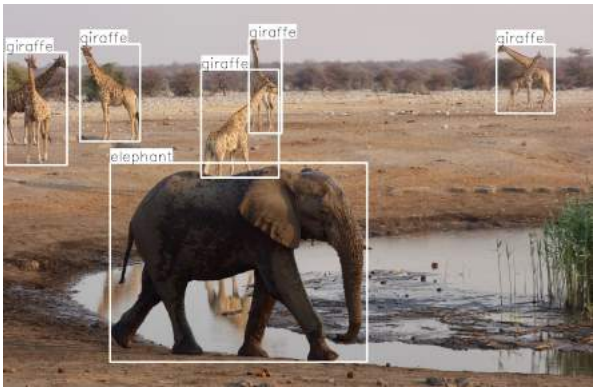
Most objects are detected precisely and assigned with the correct label. However, some objects and people are not detected, for example, the orange, chair, and cell phone in picture 35b. Also, overlapping giraffes in Figure 35d are detected as a single instance. The detection performance can be improved by using a deeper backbone like ResNet-101. Additional data augmentation techniques like position and color augmentation probably also improve detection results. Because we want to analyze the impact of embedded MoE layers, we stay with the data augmentation, as stated in the official paper, to avoid biases in our results.

(a)



(b)

(c)



(d)

Figure 35: Application of the vanilla RetinaNet baseline using a ResNet-50 backbone to different sample images of varying aspect ratios, resolutions, and domains. Input images are prepared as stated in section 6.1. Detection results are upscaled and inserted into the images. Raw image sources: C.3.

6.4 Experiments with Regressor and Classifier MoE

We modify the vanilla RetinaNet by replacing the regressor and classifier with two separate MoE blocks, referred to as *DetectorMoE*. We assume that the MoE concept provides the most significant benefit in the parts of a model responsible for decision making and processing high-level features. Specialized experts may interpret extracted features in different ways and draw appropriate conclusions for different domains. Even if this type of architecture is closer to a traditional MoE approach, the whole model is trained end-to-end. We re-use the vanilla classification and regression subnets as expert networks. No further modifications are applied to the expert networks. Due to restricted resources, we only train models with four experts in each MoE layer. The gating networks make assignment decisions independently for each feature map level P_3 to P_7. Section 3.4 states details on the RetinaNet architecture and their various levels. Consequently, on each of the five resulting feature maps, individual weightings emerge, and different experts are utilized. We state the number of active experts in the regressor and classifier MoE blocks as k_{reg} and k_{cls}, respectively.

Instead of constraining expert utilization in terms of importance for each feature map level P_3 to P_7 individually during training, we compute the importance values based on each level's weight vectors and images in a batch. So for a batch size of 4, importance is computed on a total of $5 * 4 = 20$ sparse weight vectors. It enables the model to balance expert utilization on a larger scale and, at the same time, allows different experts to specialize on distinct feature map levels. The models would similarly utilize experts on each level if we compute the importance for each level separately and use the sum of them for constraints. We argue that it is more reasonable to encourage similar expert usage per batch instead of per feature map level. We further freeze batch normalization layers since we only train with a small batch size of 4. Batch normalization may become unstable and degrades the training progress for small batch sizes.

We rely on the same expert utilization constraints as in chapter 5. We only exclude mean importance constraints since the models suffer from massive dying expert problems. We train models with both soft constraints, importance and KL divergence loss, with $w_{aux} = 0.25$. We reduce w_{aux} since expert utilization seems to become more stable in MoE layers that are embedded deeper in networks. Additionally, we use the relative importance constraint narrowing the threshold tighter to $m_{rel} = 0.3$ to avoid dying experts. Each model uses pre-trained weights from the baseline for the ResNet-50 feature extractor. We keep those weights frozen during training to increase training speed and reduce GPU memory usage.

We also train models with unfrozen weights but do not observe performance improvements. The gating network is identical to the ConvGate from section 5.11 to improve feature processing on different feature map levels. All experts are initialized with PyTorch default values using Kaiming initialization [61]. By that, we expect the experts to learn more diverse features compared to using pre-trained weights. Section 6.8 investigates the effects of experts with pre-trained weights.

Besides training models with two separate gating networks, we also train one MoE model with a single gating network shared between regressor and classifier MoE blocks. We train this model with KL divergence loss. Since both subnets make predictions on the same input feature maps, we assume that experts in each subnet are specialized in similar domains. A shared gating network further reduces the model's size, number of trainable parameters, and training time.

Experimental details are stated in Appendix B.9. Test results on COCO test-dev2017 for all models using $k = k_{reg} = k_{cls} = 2$ active experts in each MoE layer are stated in Tables 19 (precision) and 20 (recall). Metrics are computed on the COCO evaluation server since no public labels are available. We refer to models using two separate gating networks as *DetectorMoE-Constraint-NumberOfExperts*. Models with a shared gating network are marked as *Single*. Similar to chapter 5, we abbreviate constraints as *KL* (KL divergence loss), *Imp* (importance loss), and *Rel* (relative importance constraint).

None of the DetectorMoE models beats the baseline model in a single performance metric. Results are about one to two percentage points below the baseline results. Our MoE models show the most significant difference for AR^{large} with up to three percentage points below the baseline. The models perform similarly independent of the training constraint. Still, the relative importance model seems to perform slightly better than the soft constrained models. That may be due to random processes during training. Because of our limited hardware resources, we perform only a single training run per model. More runs would be needed to quantify statistically significant differences between the constraints. The results demonstrate that a model with only a single gating network performs only slightly worse compared to the equivalent model using two separate gating networks.

We further compare class-wise performances with $k = 2$ active experts on the test-dev2017 dataset. Results for the class-wise mean average precision AP are stated in Appendix E.1. None of the DetectorMoE models beats the baseline in terms of AP on any class. Hence, the models perform only slightly worse for most classes. For most classes, differences amount to less than three percentage points.

Model	AP	$AP^{IoU=.50}$	$AP^{IoU=.75}$	AP^{small}	AP^{medium}	AP^{large}
Baseline	**35.0%**	**52.5%**	**37.3%**	**16.6%**	**37.5%**	**47.4%**
DetectorMoE-KL-4	33.5%	50.9%	35.6%	15.5%	36.3%	44.8%
DetectorMoE-Imp-4	33.5%	50.9%	35.5%	15.5%	36.2%	44.4%
DetectorMoE-Rel-4	33.7%	51.0%	35.8%	15.7%	36.2%	45.1%
DetectorMoE-KL-4 Single	33.4%	50.9%	35.5%	15.5%	36.1%	44.8%

Table 19: RetinaNet classifier and regressor MoE precision metrics on COCO test-dev2017.

Model	$AR^{max=1}$	$AR^{max=10}$	$AR^{max=100}$	AR^{small}	AR^{medium}	AR^{large}
Baseline	**31.0%**	**48.3%**	**50.9%**	**27.6%**	**55.6%**	**69.7%**
DetectorMoE-KL-4	30.0%	46.6%	49.1%	25.3%	54.2%	66.9%
DetectorMoE-Imp-4	29.9%	46.6%	49.1%	25.7%	54.0%	66.9%
DetectorMoE-Rel-4	30.0%	46.7%	49.3%	25.7%	54.3%	67.3%
DetectorMoE-KL-4 Single	30.0%	46.6%	49.1%	25.7%	53.9%	66.7%

Table 20: RetinaNet classifier and regressor MoE recall metrics on COCO test-dev2017.

We also compare the results for different object sizes. In terms of AP^{Large}, the models outperform the baseline in a few cases. The most considerable difference is achieved by DetectorMoE-KL-4 that outperforms the baseline by nine percentage points for the class ball. Other differences are much smaller and mostly below two percentage points.

We compare analogously to section 5.9 evaluation metrics for a varying number of active experts per MoE computation. Since RetinaNet makes detections on five different scales, expert combinations may vary for the same input image on different levels. We vary k equally in both gating networks. Table 21a states selected evaluation metrics for DetectorMoE-KL-4. We highlight the best results for each metric. Similar to our experiments on CIFAR-100, increasing the number of active experts above $k = 2$ (training value), performance increases for most metrics. Hence, the additional precision and recall gains are significantly smaller for activating more than two experts. Each additional regression expert adds 25.835 GMac and each classification expert 34.279 GMac in computational complexity.

Test results for DetectorMoE-Imp-4 and DetectorMoE-KL-4 Single in Appendix E.2 draw a similar picture. Table 21b shows that DetectorMoE-Rel-4 improves its performance similarly with an increasing number of active experts. However, differences for varying values of k are smaller than for soft constrained models. The MoE models can still not beat the baseline model in a single metric, independently of the number of active experts. We further analyze the regressor MoE and classifier MoE separately in the next sections since both subnets are trained independently. We focus our analysis on DetectorMoE-KL-4 and DetectorMoE-Rel-4 because both soft constrained models show similar behavior.

Metric	k=1	k=2	k=3	k=4
AP	31.1%	33.5%	**33.6%**	**33.6%**
AP^{small}	14.2%	15.5%	15.6%	**15.7%**
AP^{medium}	34.8%	**36.3%**	36.2%	36.1%
AP^{large}	41.7%	44.8%	44.7%	**45.0%**
$AR^{max=100}$	46.3%	49.1%	**49.3%**	49.2%
AR^{small}	23.0%	25.3%	25.6%	**26.0%**
AR^{medium}	51.4%	54.2%	**54.5%**	54.3%
AR^{large}	63.2%	**66.9%**	**66.9%**	66.8%

(a) DetectorMoE-KL-4

Metric	k=1	k=2	k=3	k=4
AP	32.9%	33.7%	**33.9%**	**33.9%**
AP^{small}	15.3%	15.7%	15.8%	**15.9%**
AP^{medium}	35.5%	36.2%	**36.5%**	**36.5%**
AP^{large}	44.1%	45.1%	**45.5%**	**45.5%**
$AR^{max=100}$	48.6%	49.3%	**49.5%**	**49.5%**
AR^{small}	24.8%	25.7%	25.9%	**26.0%**
AR^{medium}	53.7%	54.3%	54.4%	**54.5%**
AR^{large}	66.4%	67.3%	**67.6%**	**67.6%**

(b) DetectorMoE-Rel-4

Table 21: COCO metrics on the test-dev2017 dataset for a varying number k of active experts.

We further compare the inference time between the vanilla RetinaNet and models with embedded MoE layers in the regressor and classifier subnets. We first warm up the GPU with ten predictions and synchronize the CUDA kernels. The inference time for a single image is measured in milliseconds (ms). The results also state the frames per second (FPS) for batch sizes of one, two, or four images. All results are averaged over 300 repetitions. We use the same image as in Figure 38f for our measurements. After resizing, the input size amounts to (3, 640, 928).

The results in Table 22 demonstrate the loss of inference speed for DetectorMoE models. The lower half of the table states inference times for models with an MoE layer replacing the fourth convolutional layer in each subnet. This model architecture is used in section 6.8. Note that our model implementations are not specifically designed for speed. Additional efficiency improvements are probably possible, especially in the processing of the network outputs. The official RetinaNet paper states an inference time of 98ms for models with a ResNet-50 backbone and an input image scale of 600 [53]. Those results are measured on the COCO test-dev2017 dataset with images of various sizes. We point out that other models, for example, YOLOv3 have a lower inference time. The authors state an inference time of 51 ms for YOLOv3 and an input image size of 608 [51]. We expect other object detection models with embedded MoE layers to have shorter inference times.

Model	Mean (ms)	Std	FPS @ BS=1	FPS @ BS=2	FPS @ BS=4
Baseline	72.21 ± 0.31	0.31	13.87	19.85	24.71
DetectorMoE, k=1	101.38 ± 0.84	0.84	9.78	14.93	19.71
DetectorMoE, k=2	124.94 ± 0.46	0.46	7.99	11.70	14.95
DetectorMoE, k=3	151.66 ± 0.74	0.74	6.60	9.49	11.96
DetectorMoE, k=4	180.18 ± 1.28	1.28	5.56	7.87	9.89
DetectorMoE Conv4, k=1	92.86 ± 0.67	0.67	10.69	16.07	21.09
DetectorMoE Conv4, k=2	98.05 ± 0.73	0.73	10.21	15.19	19.81
DetectorMoE Conv4, k=3	104.13 ± 0.56	0.56	9.60	14.40	18.67
DetectorMoE Conv4, k=4	108.55 ± 0.48	0.48	9.24	13.75	17.70

Table 22: Inference times for different model architectures with a varying number k of active experts. We use the same input image as in Figure 38f. We measure results as the average of 300 repetitions on a GeForce RTX 2080 Ti. Results are stated in ms for a single image and as frames per second (FPS) for batch sizes of 1, 2, and 4.

6.5 Qualitative Analysis of Regressor MoE

To gain insights into the weighting decisions of BBox regressor, we first analyze the behavior visually. For the whole chapter, we keep $k_{cls} = 2$ experts active in the classifier MoE and only analyze the regressor MoE. We visualize the different BBox predictions of DetectorMoE-KL-4 in Figure 36. The final prediction of our model is drawn with blue lines. Predictions made by single experts selected from the gating network are drawn with green lines. For each prediction, top $k_{reg} = 2$ experts are selected. Additionally, we plot predictions in red made by experts not selected by the gating network. These predictions do not contribute to the MoE prediction. Note that RetinaNet can detect multiple objects in an image, but we will only use images containing a single object for now. The gating network assigns the weights (0.5528, 0.4472) to experts 2 and 3. Without deactivating experts, the gating network would assign the weights (0.1510, 0.3531, 0.2857, 0.2102) to experts 1 to 4. It demonstrates that the gating network takes considers all experts before choosing the top-weighted experts. Hence, the assigned weights deviate from an equal distribution, and expert 2 is assigned more than twice as much weight as expert 1.

Figure 36 visually demonstrates the basic MoE concept. We exclude label predictions and focus on the predicted BBoxes. None of the BBox predictions of single experts fits the object accurately. Most BBoxes are too large or too small. Some sides are shifted too far right or left. The weighted output, on the other hand, fits the object precisely. Note that the inactive experts do not make significantly worse predictions. One of the red BBoxes is even very close to the combined output.

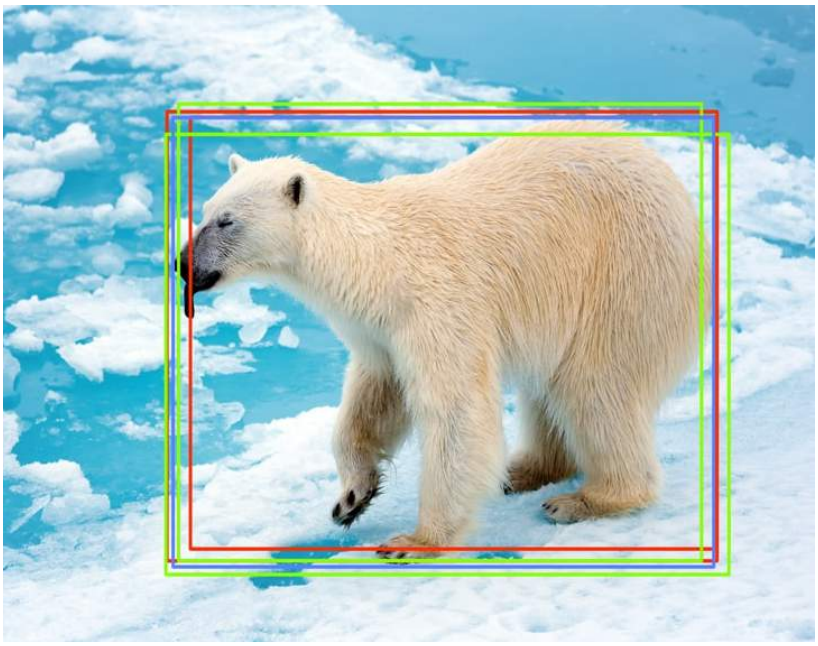
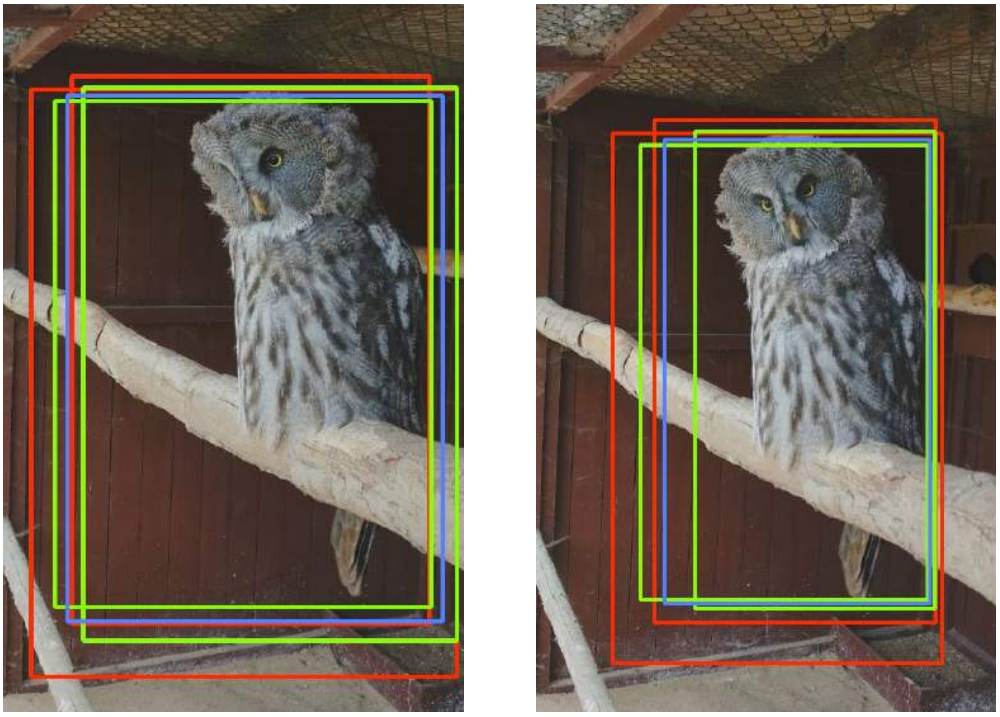


Figure 36: Visualization of BBox predictions of DetectorMoE-KL-4 and its distinct experts. The predictions of active experts ($k_{reg} = 2$) are green, predictions of inactive experts red. The final MoE prediction is blue. Raw image source: C.3.

However, since expert predictions are convex combined, using the inactive experts instead of the activate ones would result in BBox predictions more inaccurate on the left and bottom. Consequently, the gating network selects the best experts for these predictions, one that slightly estimates the BBox too low and one that estimates it too high. Hence, the selected experts would not make the best predictions on their own.

We want to point out that this image is just a single example that illustrates the gating network decision and expert combination. For many other input images, the single expert predictions vary much less and are sometimes not distinguishable. To complete the picture, we also plot expert predictions and final results for the other DetectorMoE models in Appendix E.3. All models achieve final predictions close to the baseline. Predictions made by distinct experts in the DetectorMoE-Rel-4 model vary less than for models trained with soft constraints. It confirms our results from section 5.10 that experts in relative importance models are more generalized and produce similar predictions.

Figure 37 compares BBox predictions on the same object from different angles. All predictions are again made by DetectorMoE-Kl-4. Assigned weights before deactivating experts amount to $(0.4368, 0.1518, 0.1934, 0.2180)$ (Figure 37a) and $(0.4383, 0.1503, 0.1938, 0.2176)$ (Figure 37b). The assigned weights are clearly not uniformly distributed even if all experts receive a significant share. After expert selection and re-standardization, resulting weights are $(0.6671, 0.3329)$ (Figure 37a) and $(0.6683, 0.3317)$ (Figure 37b).



(a) Sample with side view (b) Sample with front view

Figure 37: Comparison of BBox predictions for different views on the same object. Predictions vary less for clearer views on objects. In both cases, the same experts are selected. Assigned weights are (0.5634, 0.4366) (37a) and (0.5620, 0.4380) (37b). Raw image source: Own recording.

Figure 37a demonstrates that all four experts have problems estimating the precise object boundaries for atypical poses and situations. Predictions on objects with a clear front view vary less. The gating network is again able to choose eligible experts for the input in Figure 37b. Hence, the gating network assigns about twice as much weight to the inferior expert than the most precise one. It underlines the fact that the gating network is not always able to pick the single best expert. Still, the inactive experts would both overestimate the bottom of the bounding box, while the selected experts both predict the bottom closely.

For images in Figure 37, all four experts have problems estimating the object's precise boundaries. We also plot predictions of other MoE models in Appendix E.4. Note that the baseline produces no precise predictions for both images (Figures 54a and 54b). DetectorMoE-Rel-4 produces a significantly closer BBox prediction for the side view (Figure 54g) and a slightly better one for the front view (Figure 54h). DetectorMoE-Imp-4 captures the object most closely for the front view (Figure 54f), even if the gating network does not choose the best experts. Figure 54f also illustrates that sometimes inactive experts make better predictions than experts selected by the gating network. In this case, the owl's tail is more precisely taking into account by the inactive experts. Both inactive experts even give a better BBox prediction than the combined MoE output.

6.6 Quantitative Analysis of Regressor MoE

Next, we compute COCO metrics for distinct regression experts by setting $k_{reg} = 1$ and assigning all weights during evaluation to a distinct expert. Due to the restricted number of evaluations on the COCO evaluation server, we compute these metrics on the COCO validation set 2017. We also evaluate the baseline and other MoE models on the validation set 2017 to compare results. Therefore, the stated metrics deviate slightly from previous sections. Note that the validation set 2017 only contains 5,000 images compared to 20,288 in the test-dev2017 dataset. We do not modify the classifier MoE for these experiments and keep $k_{cls} = 2$. Our results for distinct experts in DetectorMoE-KL-4 and DetectorMoE-Rel-4 are stated in Table 23. Additional results for DetectorMoE-Imp-4 and DetectorMoE-KL-4 Single are stated in Appendix E.5. The gating networks for regressor and classifier in the DetectorMoE-KL-4 Single model are usually identical. We clone the gating network and keep $k_{cls} = 2$ for the classifier MoE to analyze the regression MoE.

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-KL-4	Baseline
AP	31.8%	31.8%	31.8%	**32.0%**	33.1%	34.5%
AP^{small}	14.9%	**15.0%**	14.9%	14.7%	15.3%	16.7%
AP^{medium}	35.6%	35.6%	**36.1%**	35.9%	36.7%	38.0%
AP^{large}	44.3%	44.5%	44.2%	**44.8%**	46.3%	48.6%
$AR^{max=100}$	46.5%	46.4%	46.8%	**46.9%**	48.1%	50.1%
AR^{small}	24.1%	24.1%	**24.2%**	24.1%	24.7%	28.0%
AR^{medium}	52.3%	52.2%	**53.1%**	52.6%	53.7%	55.1%
AR^{large}	63.3%	63.6%	63.8%	**64.3%**	66.0%	69.1%

(a) DetectorMoE-KL-4

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-Rel-4	Baseline
AP	32.8%	**33.1%**	**33.1%**	**33.1%**	33.3%	34.5%
AP^{small}	15.0%	**15.1%**	15.0%	**15.1%**	15.1%	16.7%
AP^{medium}	36.4%	**36.8%**	**36.8%**	36.7%	37.0%	38.0%
AP^{large}	46.4%	**46.7%**	**46.7%**	**46.7%**	47.0%	48.6%
$AR^{max=100}$	47.7%	48.1%	**48.2%**	48.1%	48.3%	50.1%
AR^{small}	**24.9%**	24.4%	24.4%	24.5%	24.4%	28.0%
AR^{medium}	53.2%	**53.8%**	**53.8%**	53.7%	54.0%	55.1%
AR^{large}	65.8%	**66.3%**	**66.3%**	**66.3%**	66.6%	69.1%

(b) DetectorMoE-Rel-4

Table 23: COCO metrics for DetectorMoE models with only distinct regression experts activated during evaluation on the validation set 2017. We keep $k_{cls} = 2$ during evaluation to observe the regression experts' behavior. We highlight the top-performing expert for each metric. For comparison, we state results for the regular model using $k_{reg} = 2$ experts for predictions, and the vanilla RetinaNet baseline.

The results show that every single expert of the same model performs similarly, independent of object scales. Still, the standard MoE models with $k_{reg} = 2$ active experts perform slightly better. That demonstrates that combining outputs of different experts improves predictions and the gating networks are able to select meaningful experts. Concerning the mean average recall AR on different scales, distinct experts' results vary more. Some of the experts receive higher scores for a specific scale than the other experts. This suggests that experts are, at least slightly, specialized on different object scales, as assumed in section 6.4. When comparing the class-wise results to our soft constrained models for CIFAR-100 classification from section 5.10, experts trained for BBox regression perform significantly better independently and are not specialized in specific classes like experts in simple ResNet-18 architectures.

We assume this is because experts in DetectorMoE models are trained for multi-object detection on images containing a varying number of objects of different classes. Since classes of different domains appear together in a single image, a specialization on different object groups may not be beneficial. Besides, the regressor subnet is class-agnostic and does not receive specific class labels during training. We verify that regression experts in our DetectorMoE-KL-4 model are not class-wise specialized by computing the difference between the best and worst performing experts per class in terms of mean average precision AP.

Most considerable differences in AP amount to 2.7 percentage points for the classes train, bed, and toaster. Results for $AR^{max=100}$ vary the most for the classes microwave, drier, and toaster with about 4.5 to 5.8 percentage points. Note that these classes are strongly underrepresented in the training and validation dataset, see Appendix C.2 for an overview of the distribution. For most other classes, the performance of experts varies only slightly. The results for the hard constrained DetectorMoE-Rel-4 draw a similar picture.

We further compute the averagely assigned weightings to each regression expert during the evaluation. We further break down the weight assignments into the different feature map levels P_3 to P_7. On each feature map level, the gating network computes separate weights and consequently selects experts independently of other levels. We compute the average weight assignments on the sparse weight vectors after selecting the $k = 2$ active experts. Our results for DetectorMoE-KL-4 and DetectorMoE-Rel-4 are stated in Table 24. Additional results for DetectorMoE-Imp-4 and DetectorMoE-KL-4 Single are stated in Appendix E.5.

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	**65.54%**	0.00%	0.16%	34.30%
P_4	**56.44%**	0.04%	0.52%	42.99%
P_5	2.32%	2.25%	47.43%	**48.00%**
P_6	0.01%	**55.83%**	43.94%	0.22%
P_7	0.04%	**66.49%**	33.37%	0.10%

(a) DetectorMoE-KL-4 Regressor

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	**67.75%**	2.62%	13.85%	15.79%
P_4	33.00%	16.23%	**47.13%**	3.64%
P_5	0.11%	**50.90%**	46.77%	2.21%
P_6	9.69%	29.81%	23.81%	**36.69%**
P_7	0.29%	**54.30%**	44.03%	1.38%

(b) DetectorMoE-Rel-4 Regressor

Table 24: Averagely assigned weights to each distinct regression expert during evaluation on the validation set 2017. Results are computed on the sparse vectors after deactivating all but $k_{reg} = 2$ experts. Levels respond to the feature map levels of RetinaNet. We highlight the top weights on each feature map level.

We also illustrate the results in Figure 38 by plotting the predicted BBoxes for different input images with objects of varying scales. The BBox captions state the indices of the two highest weighted experts, the weight assigned to the first expert, and the feature map level on which the detection is made. The results point out that the gating network selects experts depending on the feature map levels P_3 to P_7. For DetectorMoE-KL-4 (see Table 24), Expert 1 is mainly utilized on levels P_3 and P_4 that are used to detect small objects such as ties or chairs (see Figures 38b and 38f). Expert 2, on the other hand, is mainly used on levels P_6 and P_7 for detection of large objects like cars (see Figure 38c). Experts 3 and 4 are used on levels P_5 to P_7 and P_3 to P_5, respectively, with slightly fewer weights than the first two experts.

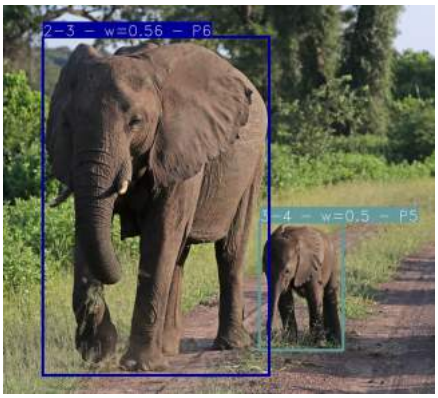
As the predictions in Figure 38 illustrate, weights between experts are assigned nearly equally distributed to both active experts most of the time. No direct correlation is notable when comparing the performance (see Table 23) and assigned weightings (see Table 24) for each distinct expert. Expert 4 performs best for large objects but is almost always inactive on the corresponding feature map levels. Similarly, expert 3 performs best for medium and small images but is not used on the corresponding feature maps.

When comparing results to DetectorMoE-Rel-4, the hard constrained gating network assigns weights more diversified. Still, experts tend to be mainly utilized for different feature map levels. Expert 1 is again mainly used for levels P_3 and P_4 and expert 2 for levels P_5 to P_7[14]. Hence, expert 3 is used on all scales from P_3 to P_7. Furthermore, expert 4 is mainly used on levels P_1 and P_6, consequently for predicting small and large objects.

[14]It is purely coincidental that expert numbers are the same as for the DetectorMoE-KL-4 model and has nothing to do with the training itself.

We conclude that distinct experts in DetectorMoE-Rel-4 perform better than DetectorMoE-KL-4 (see Table 23) because they are activated on a larger number of different feature map levels and are trained to compute BBoxes on different scales. Expert 1 is the only expert that is primarily used for detecting small objects. It outperforms the other experts on AR^{small} but, on the other hand, performs worse on the other metrics.

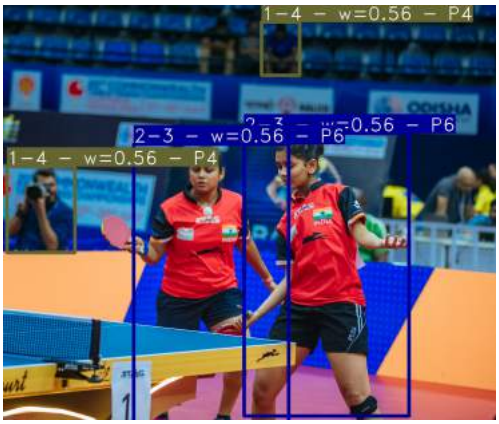
Altogether, we conclude that experts in regressor MoE layers do not specialize in predicting BBoxes for distinct classes. On the contrary, our analyses show that the gating network selects experts mainly depending on the input of different feature map levels. That is because we calculate constraints based on the importance across all levels. Moreover, we do not apply the constraints level-wise. Still, experts perform similarly accurate on detecting objects of different scales on different feature map levels. Improvements could be made by training specific experts on datasets particular for small or large datasets. For example, an expert mostly used for detection on levels $P3$ and $P4$ could be further trained with a dataset for small objects like toasters or hairdryers. This type of expert specialization allows improvements on specific object scales. Note that small objects refer to the object's size in the input image and not to the actual sizes in reality.



(a) Medium objects (P_5 - P_6)

(b) Small to medium objects (P_3 - P_5)

(c) Large Object (P_7)



(d) Small to medium objects (P_4 to P_6)

(e) Small to medium objects (P_4 - P_6)

(f) Small to medium objects (P_4 - P_6)

Figure 38: Predicted BBoxes by DetectorMoE-KL-4. The caption for each BBox states the index of the two highest weighted experts, the weight assigned to the top expert, and the feature map level on which the detection is made. Output images are partly cropped. Raw image sources: C.3.

6.7 Quantitative Analysis of Classifier MoE

Analogous to the analysis of the regressor MoE in the previous section, we now compute COCO metrics on the validation set 2017 for the classifier MoE. Likewise, we keep $k_{reg} = 2$ and assign all weights in the classifier MoE to a specific expert. Results for DetectorMoE-KL-4 and DetectorMoE-Rel-4 are stated in Table 25. We state additional results for DetectorMoE-Imp-4 and DetectorMoE-KL-4 Single in Appendix E.6.

For DetectorMoE-KL-4, evaluation results of specific classifier experts vary more than of regressor experts in section 6.6. We further compute the average assigned weights to each expert during evaluation and state results in Table 26. Expert 1 stands out and performs significantly worse than the other experts. The expert is mostly utilized on feature map level P_7 with about 90% of weight assignments. Consequently, the expert specializes in detecting large objects on this specific feature map size. It also receives about a quarter of weights for level P_6 but is not the top-weighted expert. On other levels is the expert not significantly utilized .

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-KL-4	Baseline
AP	19.5%	29.7%	30.7%	**31.0%**	33.1%	34.5%
AP^{small}	5.3%	13.1%	**14.6%**	13.9%	15.3%	16.7%
AP^{medium}	18.6%	34.9%	**35.1%**	34.4%	36.7%	38.0%
AP^{large}	29.6%	42.5%	42.3%	**43.6%**	46.3%	48.6%
$AR^{max=100}$	32.5%	46.4%	46.4%	**47.3%**	48.1%	50.1%
AR^{small}	14.8%	23.4%	23.6%	**25.4%**	24.7%	28.0%
AR^{medium}	32.9%	52.2%	51.6%	**52.9%**	53.7%	55.1%
AR^{large}	46.7%	**64.6%**	63.7%	64.4%	66.0%	69.1%

(a) DetectorMoE-KL-4

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-Rel-4	Baseline
AP	32.3%	32.3%	32.3%	**32.6%**	33.3%	34.5%
AP^{small}	13.8%	**14.8%**	13.9%	14.5%	15.1%	16.7%
AP^{medium}	**36.1%**	35.9%	35.5%	36.0%	37.0%	38.0%
AP^{large}	45.9%	44.9%	45.4%	**46.3%**	47.0%	48.6%
$AR^{max=100}$	47.8%	47.5%	47.6%	**48.2%**	48.3%	50.1%
AR^{small}	24.6%	24.2%	24.6%	**25.3%**	24.4%	28.0%
AR^{medium}	53.4%	53.0%	53.0%	**53.6%**	54.0%	55.1%
AR^{large}	65.9%	65.3%	65.7%	**66.4%**	66.6%	69.1%

(b) DetectorMoE-Rel-4

Table 25: COCO metrics for DetectorMoE models with only a distinct single classification expert activated during the evaluation on the validation set 2017. We keep $k_{rel} = 2$ during evaluation to observe the classifier experts' behavior. We highlight the top-performing expert for each metric. For comparison, we state results for the regular model using $k = 2$ experts for predictions and the vanilla RetinaNet baseline.

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	0.00%	0.05%	**59.36%**	40.59%
P_4	0.00%	4.44%	**57.12%**	38.44%
P_5	1.42%	**58.42%**	8.15%	32.01%
P_6	26.55%	**58.19%**	0.00%	15.26%
P_7	**90.02%**	7.17%	2.34%	0.47%

(a) DetectorMoE-KL-4 Classifier

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	0.06%	**69.68%**	11.67%	18.59%
P_4	15.54%	21.43%	18.57%	**44.46%**
P_5	**54.54%**	0.04%	23.78%	21.64%
P_6	15.74%	0.10%	32.83%	**51.33%**
P_7	**52.52%**	0.27%	26.60%	20.62%

(b) DetectorMoE-Rel-4 Classifier

Table 26: Averagely assigned weights to each distinct classification expert during the evaluation on the validation set 2017. Results are computed on the sparse vectors after deactivating all but $k_{cls} = 2$ experts. Levels respond to the feature map levels of RetinaNet. We highlight the top weights on each feature map level.

We conclude that expert 1 can detect large objects but, on the other side, is not able to detect and classify objects of smaller scales as precisely as the other experts. Therefore, the single performance of the expert is significantly worse in terms of COCO metrics. Note that metrics for large objects take all objects larger than 96^2 into account. With input images of a minimum size of 608×608, objects for these metrics may be detected on feature map levels below P_7.

When comparing the other three experts in DetectorMoE-KL-4, differences are still small but slightly larger than for the regressor experts. Experts in the DetectorMoE-Rel-4 model perform slightly better on their own and are closer to the MoE model utilizing $k_{cls} = 2$ experts. Note that some experts even perform better for the recall metrics on distinct scales than the full MoE models. In terms of precision, the full models outperform every single expert. It demonstrates that distinct experts may identify more relevant objects at the cost of an increasing number of false-positive predictions. The MoE, on the other hand, shows better accuracy and, therefore, can decrease the number of false-positive predictions at the cost of also missing some objects. It is a classic precision-recall trade-off.

Weight assignments in the classifier MoE of the DetectorMoE-KL-4 model are similar to the regressor MoE. Again, the gating network in DetectorMoE-KL-4 assigns weights for levels P_3 and P_4 to two specific experts. For level P_5 experts 2 and 4 are mainly used. Differences arise for levels P_6 and P_7. Additional results for DetectorMoE-Imp-4 and DetectorMoE-KL-4 Single are stated in Appendix E.6.

Class	Expert 1	Expert 2	Expert 3	Expert 4	ΔAP^{small}
toaster	0.0%	7.6%	43.2%	9.3%	33.9 pp
parking meter	3.4%	15.2%	20.6%	16.0%	4.6 pp
book	0.2%	4.9%	8.7%	4.0%	3.8 pp
oven	0.0%	0.0%	3.7%	0.3%	3.4 pp
person	23.3%	23.8%	26.6%	24.0%	2.6 pp
sheep	12.8%	16.8%	22.1%	19.5%	2.6 pp
bottle	8.4%	15.3%	17.1%	14.4%	1.8 pp
tie	0.0%	10.9%	12.5%	10.6%	1.6 pp
handbag	0.0%	6.4%	9.3%	7.7%	1.6 pp
remote	0.2%	8.9%	12.6%	11.3%	1.3 pp

Table 27: Class-wise AP^{small} evaluation results on COCO validation set 2017. We state the ten classes with the highest differences between classification expert 3 and the next best expert in DetectorMoE-KL-4. Expert 3 receives the highest average weighting on feature map levels P_3 and P_4 and is consequently used primarily for detecting small objects. We state the evaluation differences between expert 3 and the next best expert in percentage points in column ΔAP^{small}.

While the gating network in the regressor MoE focuses on two experts per level, the classifier's gating network utilizes three experts for P_6 and mostly a single expert for P_7. Expert 3, mainly utilized on levels P_3 and P_4, shows the best accuracy for small and medium objects. Expert 4, which is also used to detect small objects, shows the highest recall on these object scales. The hard constrained DetectorMoE-Rel-4 also assigns weights more diversified to its experts in the classifier MoE. On level P_4, all four experts are utilized to a significant portion, with expert 4 receiving about twice the average weights. Three of four experts are utilized on the other levels, with one primary expert receiving more than half of the weights. Only expert 2 is used primarily on levels P_3 and P_4.

Analyzing class-wise metrics for DetectorMoE-KL-4, expert 1 performs worse than other experts in all classes in terms of AP. Comparing results for AP^{large}, expert 1 slightly outperforms the other experts for the classes table, bed, and person. For most other evaluation metrics, expert 1 performs worse than the remaining experts. When comparing the remaining three experts, class-wise performances differ more than for regressor MoE experts. As an example, we state class-wise results for AP^{small} Table 27. We restrict the data to classes for which expert 3 outperforms other experts most significantly in terms of absolute differences.

As expected, expert 3 outperforms other experts for small objects like toasters, parking meters, or books. Note that differences to the second-best expert decreases rapidly and amounts for the tenth class to only 1.3 percentage points. Similar results can be observed for other metrics and scales. In summary, some experts perform better for specific objects of specific sizes. Still, the performance of different experts is not as class-dependent as for models trained for CIFAR-100 classification. These results also apply to other classifier subnets in DetectorMoE models.

We reuse the images from Figure 38, but this time we set $k_{reg} = 2$ and plot predictions using distinct classification experts. We state assigned class labels, predicted confidence scores, and the selected experts for each detection. Figure 39 compares the experts' predictions to the full MoE model on the same input image. The DetectorMoE-KL-4 model using $k_{cls} = 2$ experts for classification detects all relevant objects in the image. Note that all experts can detect these objects correctly but vary in their confidence. Experts 2 and 4 also falsely predict additional objects. Expert 4 detects a table tennis racket and classifies it as a standard tennis racket. Since the training data does not contain table tennis rackets, the expert's prediction might be rated positively.

Expert 2 detects all smaller objects correctly but falsely detects a tv and truck. Note that expert 2 achieves the highest AR^{large} score on the validation set 2017, but not the best AP^{large} score. This may be an indicator that the expert tends to predict more objects than other experts. The expert is also only assigned about 7% average weights on feature map level P_7 on which these predictions are made.

Expert 1 makes the overall best predictions with the highest confidence scores. Indeed, the gating network utilizes expert 1 to detect the two women in the foreground but assigns slightly higher weights to expert 2. For persons in the background, experts 2 and 3 are utilized even if expert 1 is more confident. This example demonstrates that the gating network does not always choose the best experts for specific feature map levels. That is due to the fact that experts are mainly utilized on the same feature map levels. Still, experts making false predictions are not activated on the corresponding feature map levels. The final outcome, therefore, avoids false object detections. We state more visual comparisons between different expert predictions in Appendix E.7.

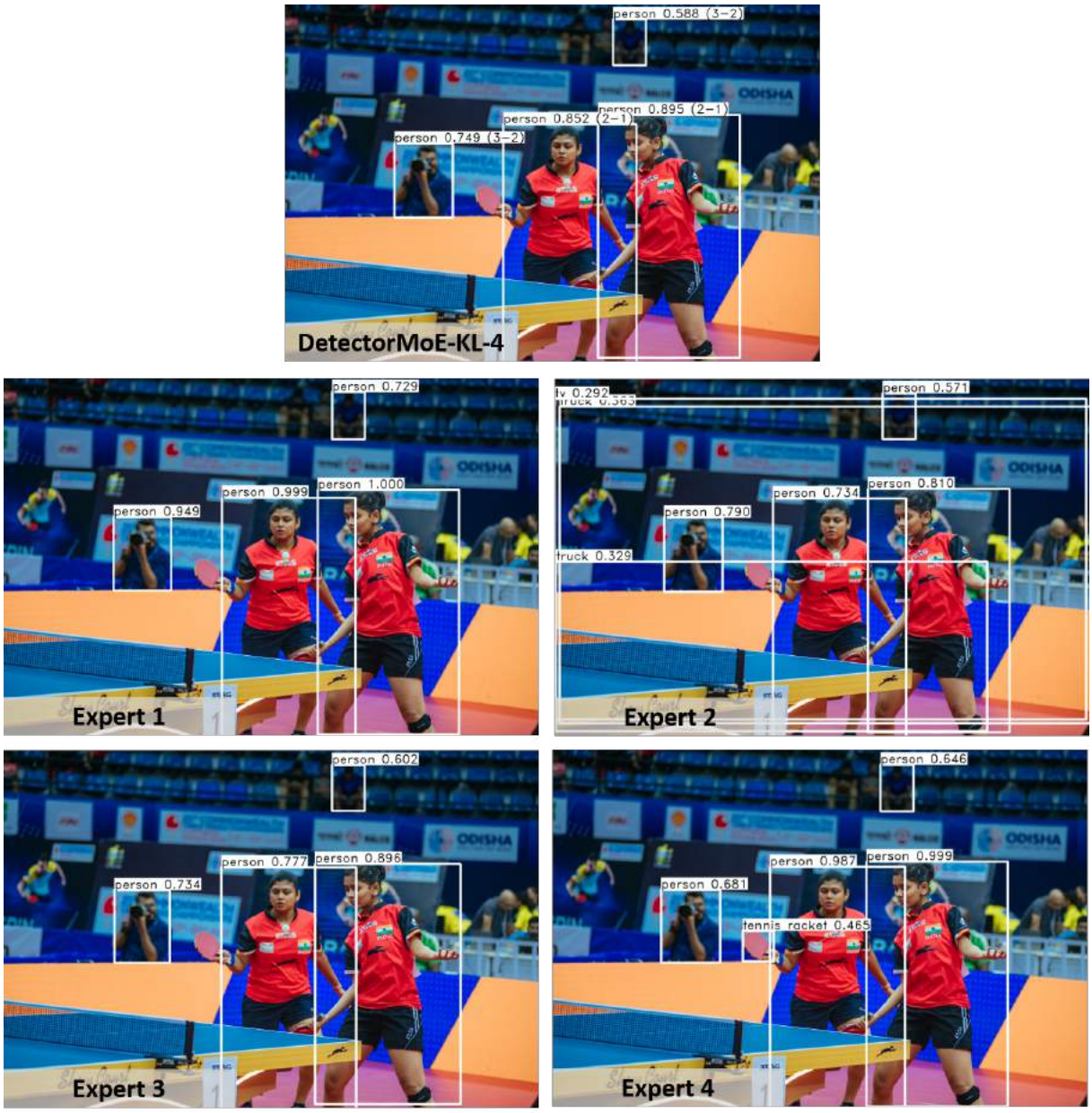


Figure 39: Comparison of different classifier expert predictions and the full MoE model. All detections are made by DetectorMoE-KL-4. For each BBox, we state the predicted class and the confidence score. We further state in brackets the two active classification experts for each prediction in the MoE. Raw image source: C.3.

6.8 Experiments with Pre-Trained Expert Networks

We also investigate the behavior of MoE models using pre-trained weights for expert networks. For this reason, we train models using pre-trained weights from the baseline model. Gaussian noise Z sampled from a zero-mean normal distribution $Z \sim N(\mu = 0, \sigma = 0.1)$ is added to the pre-trained weights to enable different experts' specialization. We assume that initializing all experts with identical weights may counteract against expert specialization and encourages the gating network to produce equal distributed weights. Furthermore, the learning rate and the number of training epochs are reduced. We also decrease the loss function's weighting to $w_{kl} = 0.05$ since each expert is already able to detect objects reliably. By reducing w_{kl}, we want to avoid forcing an equally distributed weight assignment. For the hard constrained DetectorMoE-Rel-4, we keep $m_{rel} = 0.3$ since it already allows the gating network to vary more between different experts. Our training details for DetectorMoE models are stated in Appendix B.9.

Besides, we train a RetinaNet model by only replacing the fourth convolutional layers in the regressor and classifier subnets with an MoE layer. Each expert consists of a single convolutional layer identical to the replaced layers. These models are denoted with $Conv4$. We also use pre-trained weights for the whole model and add noise to the expert networks. Because the basic DetectorMoE-KL-4 performs better for pre-trained weights than the hard constrained equivalent, we continue to use KL divergence loss with $w_{kl} = 0.05$ for the Conv4 models. We freeze the weights in the ResNet-50 backbone and only update the regressor and classifier subnets. For comparison, we also train Conv4 models without pre-trained weights. We state the training details for the Conv4 models in Appendix B.10. The evaluation metrics for all models computed on the COCO test-dev2017 dataset are stated in Tables 28 (precision) and 29 (recall).

Both KL divergence loss models with pre-trained weights outperform the baseline slightly in most metrics. The baseline performs only for AR^{large} better, but at the same time shows slightly worse performance for AP^{large}. All three models with pre-trained weights outperform the DetectorMoE models training regressor and classifier from scratch. Soft constrained models show more extensive improvements than the hard constrained model. We further analyze if the assigned weightings and expert performance show differences for models with pre-trained experts. We focus in this analysis on Conv4 models since they perform similar to our basic DetectorMoE models. However, they have less trainable parameters and a lower computational complexity in terms of GMac.

Model	AP	$AP^{IoU=.50}$	$AP^{IoU=.75}$	AP^{small}	AP^{medium}	AP^{large}
Baseline	35.0%	52.5%	37.3%	16.6%	37.5%	47.4%
DetectorMoE-KL-4	33.5%	50.9%	35.6%	15.5%	36.3%	44.8%
DetectorMoE-Rel-4	33.7%	51.0%	35.8%	15.7%	36.2%	45.1%
DetectorMoE-KL-4 Pre-trained	**35.1%**	**52.7%**	**37.4%**	**16.7%**	37.5%	**47.6%**
DetectorMoE-Rel-4 Pre-trained	34.4%	51.7%	36.6%	16.2%	36.8%	46.7%
DetectorMoE-KL-4 Conv4	33.6%	50.9%	35.8%	15.6%	36.1%	45.4%
DetectorMoE-Rel-4 Conv4	33.6%	50.9%	35.8%	15.6%	36.0%	45.1%
DetectorMoE-KL-4 Conv4 Pre-trained	**35.1%**	52.6%	**37.4%**	**16.7%**	**37.6%**	**47.6%**

Table 28: DetectorMoE precision metrics on COCO test-dev2017 using pre-trained weights for classifier and regressor experts. The results are compared to models without pre-trained weights in the regression and classification subnets.

Model	$AR^{max=1}$	$AR^{max=10}$	$AR^{max=100}$	AR^{small}	AR^{medium}	AR^{large}
Baseline	31.0%	48.3%	50.9%	27.6%	55.6%	**69.7%**
DetectorMoE-KL-4	30.0%	46.6%	49.1%	25.3%	54.2%	66.9%
DetectorMoE-Rel-4	30.0%	46.7%	49.3%	25.7%	54.3%	67.3%
DetectorMoE-KL-4 Pre-trained	**31.1%**	**48.4%**	51.0%	**27.8%**	55.9%	69.4%
DetectorMoE-Rel-4 Pre-trained	30.5%	47.3%	49.8%	25.8%	54.8%	68.4%
DetectorMoE-KL-4 Conv4	30.2%	47.0%	49.5%	25.9%	54.5%	67.6%
DetectorMoE-Rel-4 Conv4	30.1%	46.9%	49.5%	26.1%	54.5%	67.4%
DetectorMoE-KL-4 Conv4 Pre-trained	**31.1%**	**48.4%**	**51.1%**	27.7%	**56.1%**	69.6 %

Table 29: DetectorMoE recall metrics on COCO test-dev2017 using pre-trained weights for classifier and regressor experts. The results are compared to models without pre-trained weights in the regression and classification subnets.

We also measure the expert utilization as averagely assigned weights during the evaluation on the COCO validation set 2017 and state results for regressor and classifier MoE in Table 30. The gating network in the pre-trained regressor MoE acts similar to the gating network in the DetectorMoE-KL-4 with deep expert networks in Table 23a. Again, experts utilization is mainly dependent on the feature map levels. However, expert utilization is more scattered, and level boundaries are less rigid. Except expert 3, all experts are utilized on all levels with a notable amount of assigned weights. Note that we reduce the number of training epochs and lower the initial learning rate for models with pre-trained expert networks.

We assume that for longer training durations, the gating network would refer more strongly to one or two experts on each level, comparable to other DetectorMoE models. For the classifier MoE, assigned weightings are also more distributed between different feature map levels, compared to Table 26b for DetectorMoE-KL-4.

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	37.83%	**48.78%**	0.64%	12.76%
P_4	36.75%	**46.07%**	1.95%	15.23%
P_5	21.00%	**38.26%**	15.33%	25.40%
P_6	8.46%	24.40%	**38.56%**	28.58%
P_7	3.20%	30.58%	**50.67%**	15.55%

(a) DetectorMoE-KL-4 Conv4 Pre-trained regressor experts

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	0.01%	48.13%	**50.76%**	1.09%
P_4	0.75%	41.80%	**46.31%**	11.14%
P_5	29.08%	**29.53%**	22.19%	19.20%
P_6	**54.14%**	22.64%	8.07%	15.15%
P_7	**55.03%**	18.59%	9.42%	16.95%

(b) DetectorMoE-KL-4 Conv4 Pre-trained classifier experts

Table 30: Averagely assigned weights to each distinct expert during the evaluation of DetectorMoE-KL-4 pre-trained on the validation set 2017. Results are computed on the sparse vectors after deactivating all but $k = 2$ experts. Levels respond to the feature map levels of RetinaNet. We highlight top weights on each feature map level.

Here are differences more apparent than for the regressor MoE. For level P_7 expert 1[15] still receives most weights, but only about 55% for the Conv4 model compared to 90% in the DetectorMoE-KL-4 model. Also, all experts are utilized on all levels except expert 1. Again, the weight distribution in the Conv4 model might approach the distribution of the DetectorMoE-KL-4 model with a higher learning rate and longer training time.

When comparing the performance of each distinct expert in Table 31, no significant differences are notable. All experts achieve similar evaluation results with only minor differences. Our DetectorMoE-KL-4 models with deeper expert networks trained from scratch show a much more substantial variance in their performance, as analyzed in sections 6.6 (regressor MoE) and 6.7 (classifier MoE). Besides, distinct experts perform mostly as good as the MoE model with two active experts, and for some metrics perform even better. However, the results also

We also analyze the class-wise metrics, which shows that all experts perform similarly for each class without significant differences. It demonstrates that the gating network cannot select and combine different experts in a way, so that resulting features improve object detection. However, the results in the MoE model do not degenerate either. We assume that experts perform very similar at training begin and do not show significantly different performances on various domains or feature map levels. Therefore, the gating network cannot choose experts based on input domains and learn a meaningful combination during training time. It further prevents experts from specializing in a specific domain and explains their similar performances.

[15] Again, it is purely coincidental that expert numbers are the same as for the DetectorMoE-KL-4 model and has nothing to do with the training itself.

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-KL-4 Conv4	Baseline
AP	**34.7%**	34.6%	**34.7%**	**34.7%**	34.7%	34.5%
AP^{small}	**16.8%**	16.7%	16.7%	16.7%	16.7%	16.7%
AP^{medium}	**38.2%**	**38.2%**	**38.2%**	**38.2%**	38.2%	38.0 %
AP^{large}	**48.7%**	**48.7%**	**48.7%**	**48.7%**	48.7%	48.6 %
AR	**50.3%**	50.2%	**50.3%**	**50.3%**	50.2%	50.1%
AR^{small}	**27.9%**	**27.9%**	27.8%	27.7%	27.8%	28.0%
AR^{medium}	**55.5%**	55.4%	55.4%	55.4%	55.4%	55.1%
AR^{large}	69.3%	69.2%	69.3%	**69.4%**	69.3%	69.1%

(a) DetectorMoE-KL-4 Conv4 Pre-trained regressor MoE

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-Rel-4 Conv4	Baseline
AP	34.6%	**34.7%**	**34.7%**	**34.7%**	34.7%	34.5%
AP^{small}	**16.8%**	**16.8%**	**16.8%**	**16.8%**	16.7%	16.7%
AP^{medium}	38.1%	**38.2%**	**38.2%**	**38.2%**	38.2%	38.0 %
AP^{large}	48.7%	**48.9%**	48.8%	**48.9%**	48.7%	48.6 %
AR	**50.3%**	50.2%	**50.3%**	**50.3%**	50.2%	50.1%
AR^{small}	**28.0%**	27.8%	27.8%	27.9%	27.8%	28.0%
AR^{medium}	55.3%	55.4%	**55.5%**	55.4%	55.4%	55.1%
AR^{large}	**69.4%**	**69.4%**	**69.4%**	69.3%	69.3%	69.1%

(b) DetectorMoE-KL-4 Conv4 Pre-trained classifier MoE

Table 31: COCO metrics for the DetectorMoE-KL-4 Conv4 Pre-trained model with only distinct classification or regression experts activated during the evaluation on the validation set 2017. We keep $k = 2$ experts active for the corresponding other MoE layer. We highlight the top-performing experts for each metric. For comparison, the tables also state results for the regular model using $k = 2$ experts in both subnets, and the vanilla RetinaNet baseline.

Expert training with pre-trained weights increases the performance significantly compared to training from scratch but obstructs expert specializations. Consequently, the decision-making process of the MoE model becomes less transparent and interpretable. It seems to be a tradeoff between interpretability and expert specialization in the MoE architecture and the whole model's object detection performance. We assume that training expert weights from scratch with correct training parameters might achieve comparable evaluation results. Due to its long training time and our hardware restrictions, hyperparameter optimization requires a lot of time and goes beyond the scope of this thesis.

6.9 Experiments with Feature Extractor MoE

To complete the picture and this chapter, we introduce MoE blocks in the ResNet-50 feature extractor of our RetinaNet model. We reuse distinct ResNet blocks as experts, but unlike in chapter 5, we make no changes to their architecture to use pre-trained weights. The available hardware, especially the GPUs, strongly restricts these experiments. To relieve the GPU during training, we freeze all earlier layers before our embedded MoE layers. Models are trained with soft constraints since we observed a more stable training than with hard constraints, as analyzed in chapter 5. We set $w_{kl} = 0.25$ for all experiments. We allow weight updates for all layers deeper than the embedded MoE layer to adjust their feature processing to the MoE outputs. We use pre-trained weights for the full model architecture. Besides, we add Gaussian noise Z sampled from a zero-mean normal distribution $Z \sim N(\mu = 0, \sigma = 0.1)$ to the pre-trained expert weights, as in section 6.8.

We denote our models as *ExtractorMoE-Constraint-Position-NumberOfExperts* with position referring to the ResNet block number. Details on ResNet-50 are stated in section 3.1, particularly in Table 1. We train models with MoE layers replacing ResNet blocks two and four, respectively. Since we observe a performance drop for ExtractorMoE-KL-2-4 at the first few epochs, we train an identical model with a modified learning rate schedule, starting with a lower initial learning rate and a reduced number of epochs. We denote this model with *Short*. Experimental details are stated in Appendix B.11.

Model	AP	$AP^{IoU=.50}$	$AP^{IoU=.75}$	AP^{small}	AP^{medium}	AP^{large}
Baseline	**35.0%**	**52.5%**	**37.3%**	**16.6%**	**37.5%**	47.4%
ExtractorMoE-KL-2-4	33.4%	50.3%	35.6%	14.9%	35.5%	46.1%
ExtractorMoE-KL-2-4 Short	34.8%	52.3%	37.0%	16.2%	37.2%	**47.5%**
ExtractorMoE-KL-4-4	32.6%	49.4%	34.6%	14.3%	34.3%	44.9%

Table 32: ExtractorMoE precision metrics on COCO test-dev2017. Our models use pre-trained weights for the whole architecture. All layers before the MoE blocks are frozen.

Model	$AR^{max=1}$	$AR^{max=10}$	$AR^{max=100}$	AR^{small}	AR^{medium}	AR^{large}
Baseline	**31.0%**	**48.3%**	**50.9%**	**27.6%**	**55.6%**	**69.7%**
ExtractorMoE-KL-2-4	29.9%	45.8%	48.1%	24.7%	52.1%	66.9%
ExtractorMoE-KL-2-4 Short	30.8%	47.8%	50.4%	26.8%	55.2%	69.2%
ExtractorMoE-KL-4-4	29.5%	44.9%	47.1%	23.6%	50.7%	65.8%

Table 33: ExtractorMoE recall metrics on COCO test-dev2017. Our models use pre-trained weights for the whole architecture. All layers before the MoE blocks are frozen.

Evaluation results for the COCO test-dev2017 dataset are stated in Tables 32 (precision) and 33 (recall). Note, the output of the MoE layer in ExtractorMoE-KL-4-4 directly results in feature maps of level P_5, P_6, and P_7 on which mostly large and medium objects are detected. Due to top-down connections, the output also influences feature maps P_3 and P_4 and, consequently, the detection of small objects. It explains differences in the corresponding metrics compared to the baseline model. The MoE embedding in ResNet block 2 shows superior performance compared to ResNet block 4. A shorter training with a lower initial learning rate further improves performance.

Table 34 compares the averagely assigned weights to each expert. ExtractorMoE-KL-2-4 and ExtractorMoE-KL-4-4 both assign weights nearly equal distributed. ExtractorMoE-KL-2-4 Short, on the other hand, mainly utilizes three experts. One expert in the model only receives about 3.5% of all weights. This focus on three experts probably arises from the random modification of initial expert weights with noise combined with a small learning rate. We assume that one expert performs worse than the other three experts at the beginning of the training. Since the learning rate starts at 10^{-6} compared to 10^{-5} for the other models, the specific expert gets stuck in a local minimum. So the lower initial learning rate only allows a local parameter search on which the specific expert cannot perform equally well as the other three experts. Consequently, the gating network assigns only small weights to that expert.

Model	Expert 1	Expert 2	Expert 3	Expert 4
ExtractorMoE-KL-2-4	**26.38%**	23.93%	24.90%	24.79%
ExtractorMoE-KL-2-4 Short	03.49%	29.99%	**37.36%**	29.16%
ExtractorMoE-KL-4-4	24.20%	24.10%	**25.89%**	25.81%

Table 34: Average weight assignments to specific experts in ExtratorMoE models.

We assume embedded MoE layers in the feature extractor may demonstrate their true potential when trained from scratch with the whole model. Unfortunately, our experiments are restricted by the available hardware. Therefore, we do not pursue feature extractor MoEs further and close this chapter with a short summary. Our results in previous sections show that MoE layers embedded in the classifier and regressor subnets, the decision units in the network, enable us to gain insights into the models' decision processes. We can separately analyze each expert's single predictions and the resulting MoE prediction. Distinct expert networks specialize in detecting objects of specific sizes and provide the possibility to improve single experts with additional datasets. Most experiments in this chapter can be elaborated further with a more potent underlying hardware. Increases in the number of trainable parameters, batch sizes, and training duration might positively impact the prediction performance of our MoE models.

7 Conclusion

This thesis demonstrates that we can realize the Mixture of Experts idea as embeddings in standard CNNs for computer vision tasks. Compared to our early experiments with traditional MoE architectures, we only need a single training run to fit models with embedded MoE layers. End-to-end training with standard optimizers like Stochastic Gradient Descent or Adam is possible. No other optimization procedures like Expectation-Maximization or a multi-step training of distinct model parts are required. Thereby, it reduces the training effort and complexity of standard ensemble learning techniques.

We investigate the dying expert problem and present various soft and hard constraints to solve that problem. Both approaches tackle the problem from a different angle and lead to various MoE behavior. While soft constraints handle expert utilization better and support the expert specialization process, hard constraints mostly maintain generalized experts and increase the models' performance in many applications. Our approaches are not only applicable for embedded MoE layers but can also be used for balancing traditional MoE architectures with a two-step training process.

We underline that embedded MoE layers realize conditional computation in standard neural networks with modifications only in distinct network layers. These embedding allow us to increase the model capacity extensively and still keep an appropriate inference time by determining the number of active experts per forward pass. We illustrate this fact by a comparison between the vanilla ResNet-18 and models with embedded MoE layers. Both model categories achieve comparable test results. Some MoE models even outperform the ResNet-18 baseline and show accuracy improvements. The prediction performance of MoE models can be even further improved by increasing the number of active experts. Hence, we can balance the trade-off between model accuracy and computational complexity with a single parameter. It enables us to train models with many experts and parameters on powerful hardware and then scale the runtime complexity based on the deployment device.

Our results further indicate that experts trained end-to-end without predefined dataset splits are still able to specialize in distinct subdomains. While traditional MoE models need domain- or task-specific datasets, the experts in our models with MoE embeddings implicitly focus on distinct domains without suitable predefined datasets. We demonstrate that experts trained for CIFAR-100 image classification specialize in recognizing different domains such as sea animals or flowers without previous data clustering. Our experiments with RetinaNet and COCO furthermore point out that object detection experts can also specialize in detecting objects of distinct sizes.

Embedded MoE layers also shed light on the decision process of neural networks. We can reproduce sample assignments by the gating networks and evaluate the behavior of distinct experts. The MoE approach enables us to pinpoint wrong assignment decisions and their impacts on the prediction quality. Moreover, we may tweak the gating network and each distinct expert by training them individually by freezing some parts of the model and fine-tuning other parts with specific datasets. We assume that, in particular, object detection with embedded MoE layers benefits from continuing the training of experts for specific object scales with appropriate datasets.

Our results show the great potential of embedded MoE layers. We only conduct little hyperparameter optimization steps and still achieve performances similar to the baseline models. Our limited hardware resources also restricted the search for optimal hyperparameters. We assume that modifications of the learning rate schedules, weight factors for auxiliary losses, training duration, and the batch size improve the training process of MoE models and their overall performance. Particularly our object detection models suffer from limited resources and only little fine-tuning is done. Besides, simultaneous training of all model parts was impossible, and we had to freeze some layers. Future studies could fruitfully explore this issue by training all parts of a model together, either from scratch or with pre-trained weights.

We further believe that improvements in the gating network architecture may support the expert selection process during training. We expect a more meaningful expert specialization is achievable, which also influences the prediction quality and interpretation approaches positively. The assignment process might be further improved by replacing the fixed number of active experts with a threshold approach. So the number of active experts becomes input dependent and is no longer fixed by a hyperparameter. Moreover, for less complicated inputs or tasks, only a single expert needs to process the data.

Future research should further develop and confirm our findings by embedding MoE layers into more complex network architectures, such as YOLO, or other computer vision tasks, for example, semantic segmentation and image captioning. We expect that MoE layers' advantages enlarge with the model architecture, dataset, and task complexity. An increasing number of experts in each embedding or multiple MoE layers in a single model might bring more advantages. Also, hierarchical arrangements of MoE layers in CNNs are an exciting topic for future work.

A Hardware and Software Specifications

All of our experiments are performed on workstations with the following hardware specifications. A software update was conducted during this thesis. Influences on performance or results have not been observed.

Hardware	Value
CPU	Intel Core i9-9900K @ 3.60 GHz, 8 Cores
GPU	GeForce RTX 2080 TI (11 GB GDDR6, 14 Gbps)
Memory	64 GB
CUDA	Version: 10.1, 11.0
PyTorch	Version: 1.5.1, 1.6.0
OS	Ubuntu 18.04 LTS (Bionic Beaver) & 20.04 (Focal Fossa)

Table 35: Hardware and software specifications workstation variant 1.

Hardware	Value
CPU	AMD Ryzen Threadripper 2970WX @ 3.0 Ghz, 24 Cores
GPU	GeForce RTX 2070 Super (8 GB GDDR6, 14 Gbps)
Memory	64 GB
CUDA	Version: 10.1, 11.0
PyTorch	Version: 1.5.1, 1.6.0
OS	Ubuntu 18.04 LTS (Bionic Beaver) & 20.04 (Focal Fossa)

Table 36: Hardware and software specifications workstation variant 2.

B Experimental Details and Hyperparameters

In this chapter, we state the training details for our various experiments. All weights are initialized with Kaiming initialization [61]. Only experiments using RetinaNet rely on a pre-trained backbone, and some experiments also on pre-trained expert networks. Refer to the corresponding sections for further architecture details.

B.1 MnistNet Baselines

Experimental details for our MnistNet baseline models, as introduced in section 4.3. Table 37 shows the training details for the MnistNet baselines trained on a single MNIST dataset. Table 38 states the training details for the MnistNet baseline trained on a combination of both datasets.

Parameter	Value
Trainable parameters	$421,642$
Training samples	$60,000 \mid 14,056$
Test samples	$10,000 \mid 2,337$
Batch size	256
Learning rate	10^{-3}
Number of epochs	25
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 37: Training details for training the MnistNet baseline on MNIST (left) and S-MNIST dataset (right), respectively.

Parameter	Value
Trainable parameters	$421,642$
Training samples	$28,112$ $(14,056 + 14,056)$
Test samples	$4,674$ $(2,337 + 2,337)$
Batch size	256
Learning rate	10^{-3}
Number of epochs	25
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 38: Training details for training the MnistNet baseline on the combined MNIST dataset with subsampled default MNIST datasets. The number of samples from each dataset are stated in brackets.

B.2 Dataset Split MoE

Experimental details for our MoE models in a dataset split setting, as introduced in section 4.4. The training details for the expert networks are stated in Table 39. The gating networks' training details are shown in Table 40.

Parameter	Value
Trainable parameters	$421,642$
Training samples	$14,056$
Test samples	$2,337$
Batch size	256
Learning rate	10^{-3}
Number of epochs	25
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 39: Training details for MnistNet experts. One expert is trained on the S-MNIST training set and another one on a subsampled MNIST training set.

Parameter	Value
Trainable parameters	$420,610 \mid 1,114,754$
Training samples	$28,112$
Test samples	$4,674$
Batch size	256
Learning rate	10^{-4}
Number of epochs	25
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 40: Training details for MnistNetGate (left) and FMGate (right). Both gating networks combine two experts, one trained on each MNIST dataset.

B.3 Semi Label Split MoE

Experimental details for our MoE models in a label split setting with two distinct label subsets, as introduced in section 4.5. Table 41 states the training details for each expert network. Training details for the gating networks are stated in Table 42.

Parameter	Value
Trainable parameters	$420,997$
Training samples	$30,596 \mid 29,404$
Test samples	$5,139 \mid 4,861$
Batch size	256
Learning rate	10^{-4}
Number of epochs	25
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 41: Training details for MnistNet experts trained on default MNIST subsets with labels [0,4] (left) and [5,9] (right), respectively.

Parameter	Value
Trainable parameters	$420,610$ ($420,710$) $\mid 1,114,754$ ($1,114,854$)
Training samples	$60,000$
Test samples	$10,000$
Batch size	256
Learning rate	10^{-4}
Number of epochs	25 (50)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 42: Training details for MnistNetGate (left) and FMGate (right) which combine experts for a semi label split setting. Values are stated for gating networks with a mapping approach. Deviating values for gating networks with an FC decoding approach are stated in brackets.

B.4 2-Combinations Label Split MoE

Experimental details for our MoE models in a 2-combinations label split setting with 45 distinct experts, as introduced in section 4.6. Training details for expert networks are stated in Table 43. Note that the dataset sizes vary since MNIST is slightly unbalanced. Table 44 states the training details for the gating networks.

Parameter	Value
Trainable parameters	$420,610$
Training samples	$11,263 - 13,007$
Test samples	$1,850 - 2,167$
Batch size	64
Learning rate	10^{-4}
Number of epochs	10
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 43: Training details for MnistNet experts trained on MNIST subset labels in a 2-combinations label split setting.

Parameter	Value
Trainable parameters	$426,157$ ($427,057$) | $7,460,909$ ($7,461,809$)
Training samples	$60,000$
Test samples	$10,000$
Batch size	64
Learning rate	10^{-4}
Number of epochs	50 (75)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 44: Training details for MnistNetGate (left) and FMGate (right) which combine experts for a 2-combinations label split setting on default MNIST data. Values are stated for gating networks with Mapping decoding approach. Deviating values for gating networks with FC decoding approach are stated in brackets.

B.5 ResNet-18 Baseline

Experimental details for our ResNet-18 baseline, as introduced in section 5.2, are stated in Table 45.

Parameter	Value
Trainable parameters	$11.2M$
Complexity (GMac)	0.56
Training examples	$50,000$
Test examples	$10,000$
Batch size	128
Learning rate	$lr = \begin{cases} 10^{-3} & \text{if } epoch < 60 \\ 10^{-4} & \text{if } 80 \leq epoch < 100 \\ 10^{-5} & \text{otherwise} \end{cases}$
Number of epochs	150
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 45: Training details for our ResNet-18 baseline. We only modify the input layers to adapt the model to CIFAR-100.

B.6 ResNet-18 with Single Layer MoE

Experimental details for our ResNet-18 model with embedded MoE layers, as introduced in section 5.4. We replace only the last single convolutional layers in one ResNet block per model. Table 46 states the training details for these models with four experts.

Parameter	Value
Trainable parameters	$11.2M \mid 11,3M \mid 11.7M \mid 13.0M \mid 18.3M$
Complexity (GMac)	$0.56 \mid 0.6 \mid 0.6 \mid 0.6 \mid 0.6$ $(k = 2)$
Training examples	$50,000$
Test examples	$10,000$
Batch size	128
Number of Experts	4
Active experts k	2
Learning rate	$lr = \begin{cases} 10^{-3} & \text{if } epoch < 80 \\ 10^{-4} & \text{if } 80 \leq epoch < 120 \\ 10^{-5} & \text{otherwise} \end{cases}$
Number of epochs	150
Optimizer	Adam $(\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08)$
Experiment repetitions	3

Table 46: Training details for ResNet-18 with single embedded MoE layers. Values separated by vertical lines refer to the position of the embedding, starting with the input convolutional layer and followed by the distinct ResNet blocks.

B.7 ResNet-18 with MoE Blocks using SimpleGate

Experimental details for our ResNet-18 model with embedded MoE blocks and deep expert networks, as introduced in section 5.3 and trained in sections 5.5 (soft constraints) and 5.6 (hard constraints). We replace a complete ResNet block in each model with a similar MoE block. Table 47 states the training details for our models.

Parameter	Value
Trainable parameters	$11.3M \,\vert\, 11.7M \,\vert\, 13.2M \,\vert\, 19.1M$ (4 experts)
	$11.8M \,\vert\, 13.4M \,\vert\, 19.2M \,\vert\, 43.2M$ (10 experts)
Complexity (GMac)	$0.56 \,\vert\, 0.56 \,\vert\, 0.55 \,\vert\, 0.55$ $(k = 2)$
Training examples	$50,000$
Test examples	$10,000$
Batch size	128
Number of Experts	4 and 10
Active experts k	2
Learning rate	$lr = \begin{cases} 10^{-3} & \text{if } epoch < 80 \\ 10^{-4} & \text{if } 80 \leq epoch < 120 \\ 10^{-5} & \text{otherwise} \end{cases}$ (4 experts)
	$lr = \begin{cases} 10^{-3} & \text{if } epoch < 100 \\ 10^{-4} & \text{if } 100 \leq epoch < 150 \\ 10^{-5} & \text{otherwise} \end{cases}$ (10 experts)
Number of epochs	150 (4 experts), 180 (10 experts)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 47: Training details for ResNet-18 with embedded MoE blocks. Values separated by vertical lines refer to the ResNet block position replaced by an MoE block.

B.8 ResNet-18 with MoE Blocks using ConvGate

Experimental details for our RetinaNet-18 model with an embedded MoE block replacing the last block in a ResNet-18, as introduced in section 5.11. We increase the gating network's complexity by adding an additional 3×3 convolutional layer. Table 48 states the training details for our models.

Parameter	Value
Trainable parameters	$19.7M$
Complexity (GMac)	0.59 ($k = 2$)
Training examples	$50,000$
Test examples	$10,000$
Batch size	128
Number of Experts	4
Active experts k	2
Learning rate	$lr = \begin{cases} 4.95 \cdot 10^{-5} \cdot epoch & \text{if } epoch < 20 \\ 10^{-3} & \text{if } 20 \leq epoch < 80 \\ 10^{-4} & \text{if } 80 \leq epoch < 120 \\ 10^{-5} & \text{otherwise} \end{cases}$
Number of epochs	150
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	3

Table 48: Training details for NarrowMoE-Imp-4-4 with ConvGate. Note that the additional convolutional layer increases the model's complexity to 0.59 GMac.

B.9 RetinaNet with DetectorMoE

Experimental details for our RetinaNet model with MoE blocks replacing the regressor and classifier subnets, as introduced in section 6.4. We use a pre-trained ResNet-50 feature extractor and freeze its weights during training. We train our experts from scratch with randomly initialized weights. A second training is performed with pre-trained expert weights. We take the pre-trained weights into account by a shorter training duration. Table 49 states the training details for our models.

Parameter	Value
Total parameters	58.543M | 57.951M
Trainable parameters	27.037M | 26.445M
	Regression expert: 2.443M
	Classification expert: 4.02M
	Gating network: 0.592M
Complexity (GMac)	160.849 | 155.814 ($k = 2$)
	Regression expert: 25.835
	Classification expert: 34.279
	Gating network: 5.035
Training samples	$118,287$
Validation samples	$5,000$
Test samples	$20,288$
Batch size	4
Number of Experts	4, each for regressor and classifier subnet
Active experts k	2, each for regressor and classifier subnet
Learning rate	$lr = \begin{cases} 10^{-5} & \text{if } epoch < 15 \\ 10^{-6} & \text{if } 15 \leq epoch < 19 \\ 10^{-7} & \text{otherwise} \end{cases}$
	$lr = \begin{cases} 10^{-6} & \text{if } epoch < 5 \\ 10^{-7} & \text{otherwise} \end{cases}$ (pre-trained)
Number of epochs	20, 6 (pre-trained)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	1

Table 49: Training details for DetectorMoE models. The deviating number of parameters and complexity for MoE models with a shared gating network are stated after a vertical separator.

B.10 RetinaNet with DetectorMoE Conv4

Experimental details for our RetinaNet model with MoE layers replacing the fourth convolutional layers in the regressor and classifier subnets, as introduced in section 6.8. We use a pre-trained ResNet-50 feature extractor and freeze its weights during training. We train our experts from scratch with randomly initialized weights. A second training is performed with pre-trained expert weights. We take the pre-trained weights into account by a shorter training duration. Table 50 states the training details for our models.

Parameter	Value
Total parameters	42.693M
Trainable parameters	11.188M
	Regression expert: 0.59M
	Classification expert: 0.59M
	Gating network: 0.592M
Complexity (GMac)	104.66 ($k = 2$)
	Regression expert: 4.547
	Classification expert: 4.547
	Gating network: 4.551
Training samples	$118,287$
Validation samples	$5,000$
Test samples	$20,288$
Batch size	4
Number of Experts	4 each for regressor and classifier subnet
Active experts k	2 each for regressor and classifier subnet
Learning rate	$lr = \begin{cases} 10^{-5} & \text{if } epoch < 15 \\ 10^{-6} & \text{if } 15 \leq epoch < 19 \\ 10^{-7} & \text{otherwise} \end{cases}$
	$lr = \begin{cases} 10^{-6} & \text{if } epoch < 5 \\ 10^{-7} & \text{otherwise} \end{cases}$ (pre-trained)
Number of epochs	20, 6 (pre-trained)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	1

Table 50: Training details for DetectorMoE Conv4 models. We only replace the fourth convolutional layer in both detection subnets with an MoE layer.

B.11 RetinaNet with Feature Extractor MoE

Experimental details for our RetinaNet model with MoE blocks replacing the second and fourth ResNet block in the feature extractor, as introduced in section 6.9. We use a pre-trained ResNet-50 feature extractor and freeze all weights in layers previous to our MoE blocks during training. We rely on pre-trained RetinaNet weights for the whole model during these experiments. Table 51 states the training details for our models.

Parameter	Value
Total parameters	42.22M | 85.231M
Trainable parameters	41.994M | 76.687M
	Expert: 4.878 M | 14.965M
	Gating network: 0.592M | 2.368M
Complexity (GMac)	107.743 | 95.847 ($k = 2$)
	Expert: 7.637 | 5.975
	Gating network: 13.645 | 3.41
Training samples	$118,287$
Validation samples	$5,000$
Test samples	$20,288$
Batch size	2 | 3
Number of Experts	4
Active experts k	2
Learning rate	$lr = \begin{cases} 10^{-5} & \text{if } epoch < 8 \\ 10^{-6} & \text{if } epoch < 10 \\ 10^{-7} & \text{otherwise} \end{cases}$ $lr = \begin{cases} 10^{-6} & \text{if } epoch < 5 \\ 10^{-7} & \text{otherwise} \end{cases}$ (short)
Number of epochs	10 | 5 (short)
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08$)
Experiment repetitions	1

Table 51: Training details for ExtractorMoE-2-4 and ExtractorMoE-4-4. We reduce the batch size to enable training on our hardware. Values separated by vertical lines refer to models with MoE blocks replacing the second (left) and fourth (right) ResNet-50 block. We also train a second ExtractorMoE-2-4 model with a shorter training time. This learning rate schedule is denoted as *short*.

C Dataset Details

We provide additional information in this chapter to our training datasets. We further state in section C.3 the image sources and licenses for the images we use for evaluating our RetinaNet models.

C.1 CIFAR-100 Classes and Superclasses

Superclas	Classes				
aquatic mammals	beaver	dolphin	otter	seal	whale
fish	aquarium fish	flatfish	ray	shark	trout
flowers	orchids	poppies	roses	sunflowers	tulips
food containers	bottles	bowls	cans	cups	plates
fruit and vegetables	apples	mushrooms	oranges	pears	sweet peppers
household electrical devices	clock	keyboard	lamp	telephone	television
household furniture	bed	chair	couch	table	wardrobe
insects	bee	beetle	butterfly	caterpillar	cockroach
large carnivores	bear	leopard	lion	tiger	wolf
large man-made outdoor things	bridge	castle	house	road	skyscraper
large natural outdoor scenes	cloud	forest	mountain	plain	sea
large omnivores and herbivores	camel	cattle	chimpanzee	elephant	kangaroo
medium-sized mammals	fox	porcupine	possum	raccoon	skunk
non-insect invertebrates	crab	lobster	snail	spider	worm
people	baby	boy	girl	man	woman
reptiles	crocodile	dinosaur	lizard	snake	turtle
small mammals	hamster	mouse	rabbit	shrew	squirrel
trees	maple	oak	palm	pine	willow
vehicles 1	bicycle	bus	motorcycle	pickup truck	train
vehicles 2	lawn-mower	rocket	streetcar	tank	tractor

Table 52: Overview over CIFAR-100 classes and superclasses [62].

C.2 COCO Class Distribution

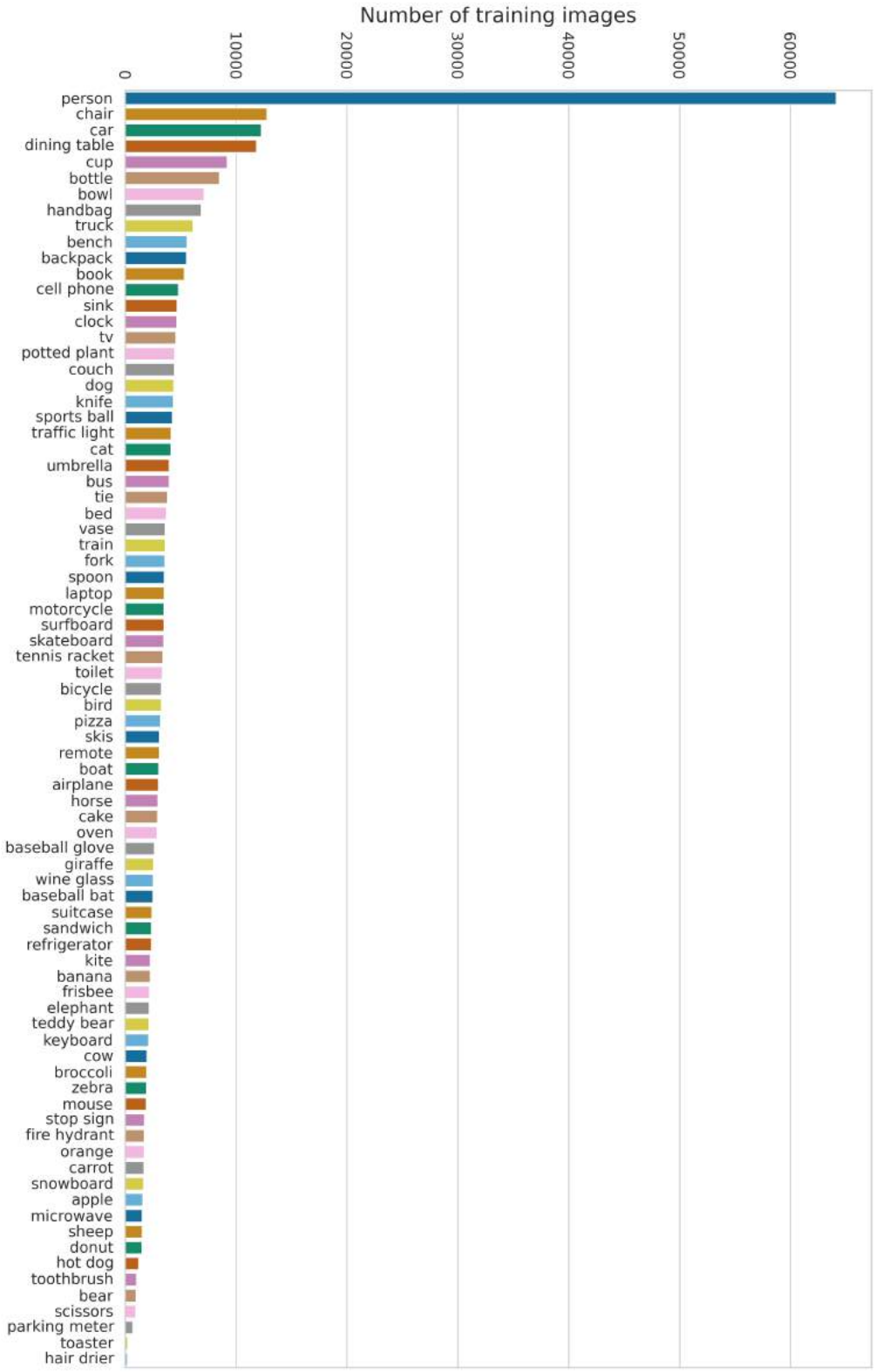


Figure 40: Distribution of object classes in the COCO training dataset. For each class, the number of images is stated that contain at least one instance. Results are sorted in descending order. The chart demonstrates a significant dataset imbalance.

C.3 Wikimedia Image Sources and Licenses

The images that we use for validating our detection models are provided by Wikimedia Commons. The website provides a large dataset of freely usable media files. License and Wikimedia page URLs of the images we use in this thesis are stated in Table 53. We point out that the licensor does not endorse our use. We add object predictions to the raw images. Some images are also cropped. Further changes are not made. We further refer to the license files in Table 54.

Figure	Wikimedia URL	License
Cover picture	File:Violet_sabrewing_(Campylopterus_hemileucurus_mellitus)_male_in_flight.jpg	CC BY-SA 4.0
35a	File:Skiing_in_Killington.jpg	CC BY-SA 4.0
35b	File:Rainer_Maria_Jilg20080317.jpg	CC BY-SA 3.0
35c	File:Veggie_burger_speed_eating_contest.jpg	CC BY 2.0
35d	File:African_bush_elephant_and_Giraffes_(23890761248).jpg	CC BY 2.0
36	File:Polar_Bear_with_its_tongue_sticking_out.jpg	CC BY 2.0
38a	File:African_bush_elephants_(Loxodonta_africana)_female_with_six-week-old_baby.jpg	CC BY-SA 4.0
38b	File:Moqueur_Lane_Living_Room_New_Orleans_2016.jpg	CC BY 2.0
38c	File:2007_BMW_635d_Sport_-_Flickr_-_The_Car_Spy_(22).jpg	CC BY 2.0
38d	File:263_254_mousomi_seerja_IND9.jpg	CC BY 4.0
38e	File:I_am_Dunking_this.jpg	CC BY-SA 4.0
38f	File:Press_conference_Nobel_Peace_Prize_2016_(1).jpg	CC BY-SA 4.0

Table 53: Image sources for Wikimedia Commons files of object detection validation images. URLs are a combination of https://commons.wikimedia.org/wiki/ and the paths provided in the table.

License	License URL
CC BY 2.0	`https://creativecommons.org/licenses/by/2.0/deed.en`
CC BY 4.0	`https://creativecommons.org/licenses/by/4.0/deed.den`
CC BY-SA 3.0	`https://creativecommons.org/licenses/by-sa/3.0/deed.en`
CC BY-SA 4.0	`https://creativecommons.org/licenses/by-sa/4.0/deed.en`

Table 54: URLs for Creative Commons license files.

D Additional Experimental Results for ResNet-18 MoE

In this chapter, we state additional results to our experiments in chapter 5. All experiments rely on ResNet-18 and use embedded MoE layers. Since we use deep expert networks, we refer to the layers also as MoE blocks. The results in this chapter are complements and mostly state our findings for other model architectures or training constraints. All models are trained on the CIFAR-100 dataset.

D.1 Accuracy for Soft Constrained MoE Blocks

The results in Table 55 complete our results for soft constrained models in section 5.5. We also state the test accuracy for models moving the ReLU function after the MoE output.

Model	Importance Loss			KL divergence Loss		
	Basic	ReLU	Shortcut	Basic	ReLU	Shortcut
Baseline	$72.62\% \pm 0.29$					
NarrowMoE-{}-1-4	$72.20\% \pm 0.24$	$72.36\% \pm 0.12$	$72.24\% \pm 0.49$	$72.21\% \pm 0.29$	$72.50\% \pm 0.69$	**72.72**$\% \pm 0.36$
NarrowMoE-{}-1-10	$70.94\% \pm 0.14$	$71.10\% \pm 0.37$	**71.51**$\% \pm 0.23$	$70.76\% \pm 0.48$	$70.34\% \pm 0.56$	$71.32\% \pm 0.60$
NarrowMoE-{}-2-4	$71.75\% \pm 0.22$	$71.68\% \pm 0.11$	$72.18\% \pm 0.29$	$71.78\% \pm 0.37$	$71.50\% \pm 0.20$	**72.25**$\% \pm 0.17$
NarrowMoE-{}-2-10	$70.71\% \pm 0.25$	$70.68\% \pm 0.44$	$71.76\% \pm 0.50$	$70.63\% \pm 0.44$	$70.73\% \pm 0.59$	**71.87**$\% \pm 0.30$
NarrowMoE-{}-3-4	$71.45\% \pm 0.24$	$71.45\% \pm 0.24$	**71.65**$\% \pm 0.43$	$71.45\% \pm 0.40$	$71.55\% \pm 0.25$	$71.54\% \pm 0.22$
NarrowMoE-{}-3-10	$70.72\% \pm 0.30$	$70.79\% \pm 0.68$	**71.47**$\% \pm 0.29$	$70.88\% \pm 0.20$	$70.89\% \pm 0.16$	$71.43\% \pm 0.12$
NarrowMoE-{}-4-4	$71.52\% \pm 0.09$	$71.42\% \pm 0.39$	**71.95**$\% \pm 0.39$	$71.37\% \pm 0.04$	$71.58\% \pm 0.28$	$71.80\% \pm 0.23$
NarrowMoE-{}-4-10	$70.70\% \pm 0.30$	$71.03\% \pm 0.41$	$71.61\% \pm 0.16$	$71.16\% \pm 0.40$	$71.10\% \pm 0.18$	**71.99**$\% \pm 0.23$

Table 55: Mean accuracy and sample standard deviation for embedded MoE blocks in a ResNet-18 architecture for different block positions, number of experts, and modifications to the experts. In all experiments, top $k = 2$ experts are activated to maintain computational complexity.

D.2 t-SNE Plots of Gating Network Logits

We provide larger versions of our t-SNE plots from section 5.7. Figures 41 and 42 plot the results for NarrowMoE-Imp-1-4 and NarrowMoE-Imp-4-4. The results for NarrowMoE-Rel-1-4 and NarrowMoE-Rel-4-4 are stated in Figures 43 and 44. All four plots are computed on the raw gating network logits resulting on the CIFAR-100 test set. We take the models with the highest test accuracy of each configuration. For further details refer to section 5.7.

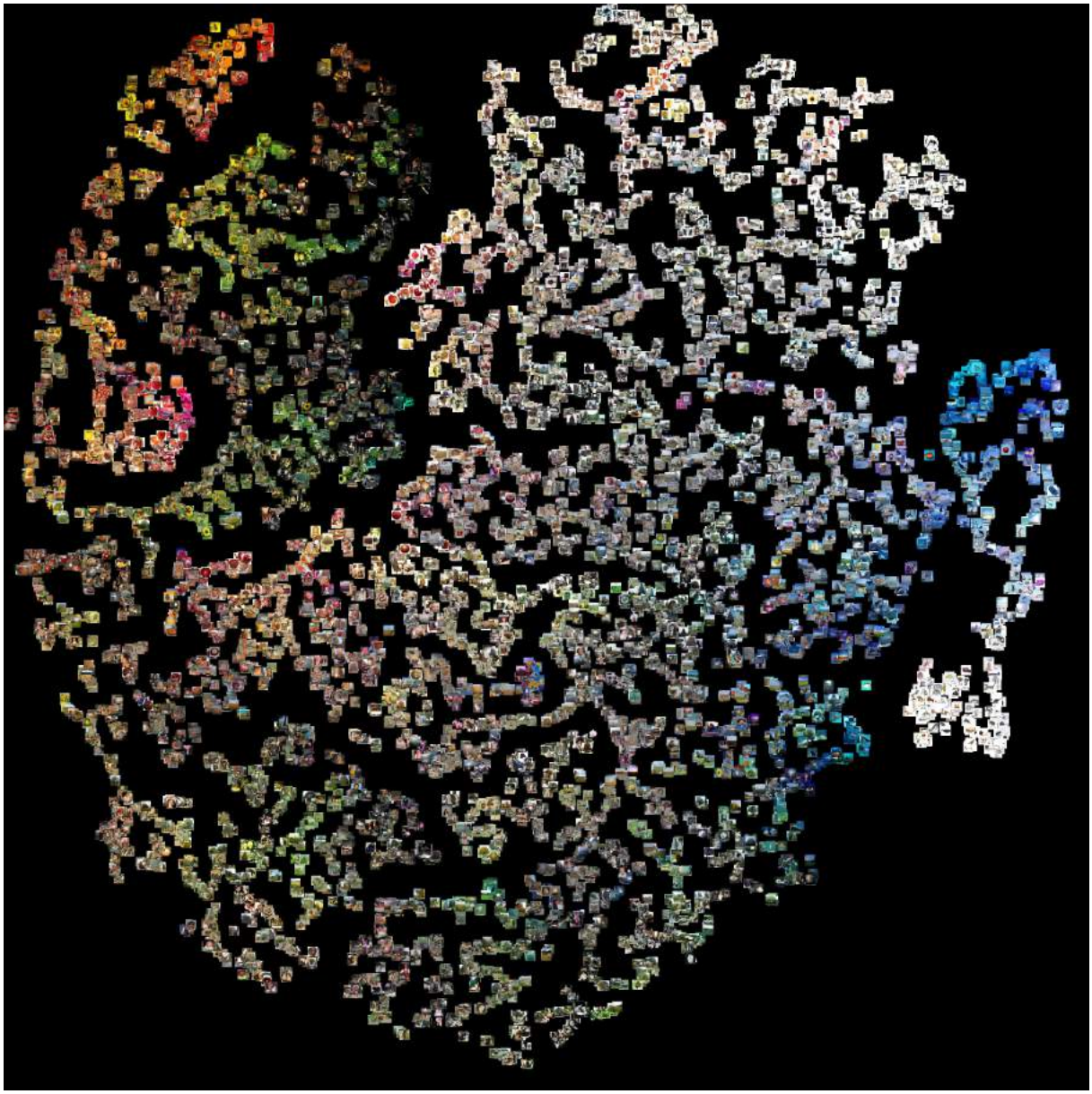


Figure 41: t-SNE of gating network logits for NarrowMoE-Imp-1-4.

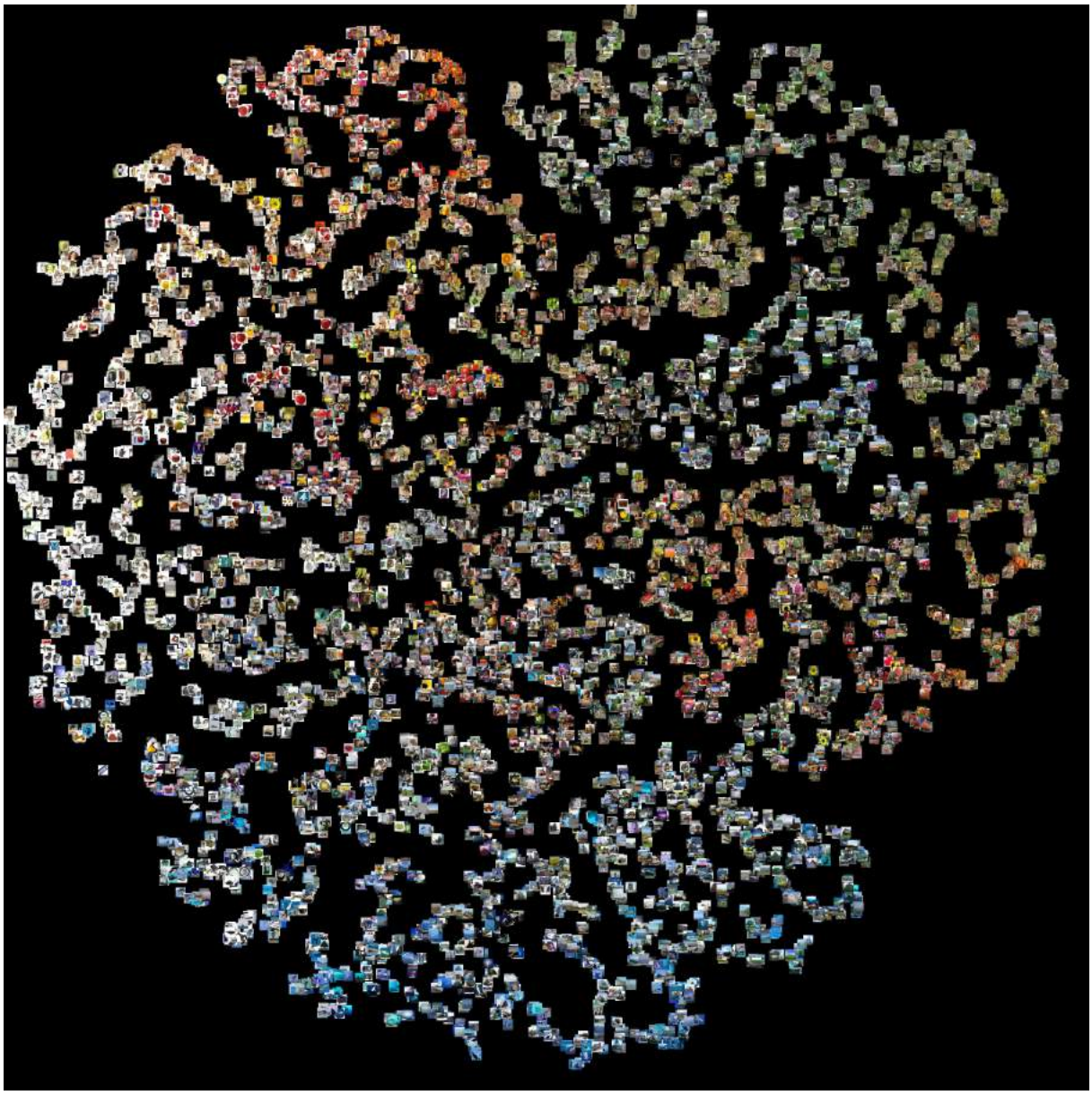


Figure 42: t-SNE plot of gating network logits for NarrowMoE-Imp-4-4.

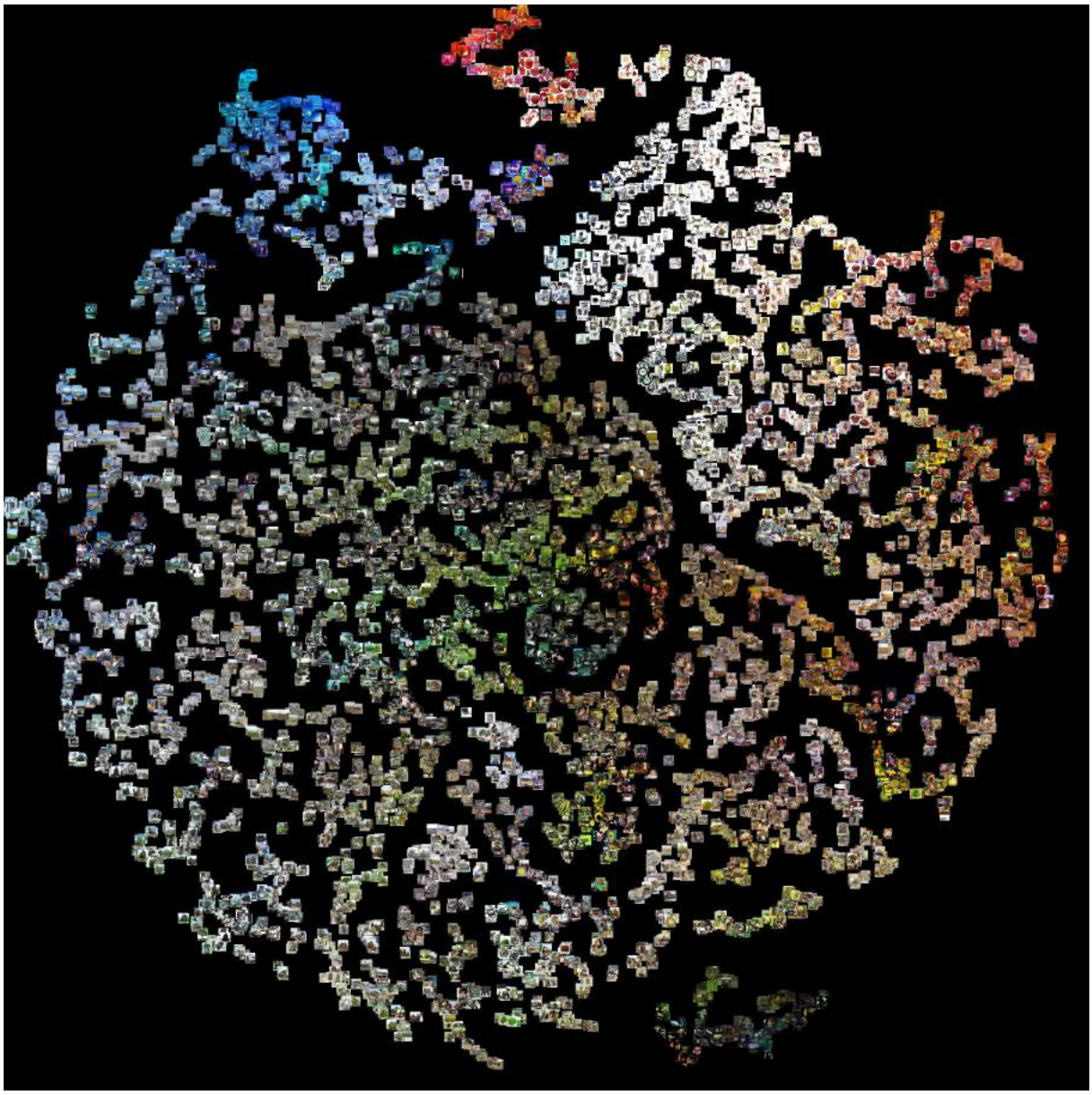


Figure 43: t-SNE plot of gating network logits for NarrowMoE-Rel-1-4.

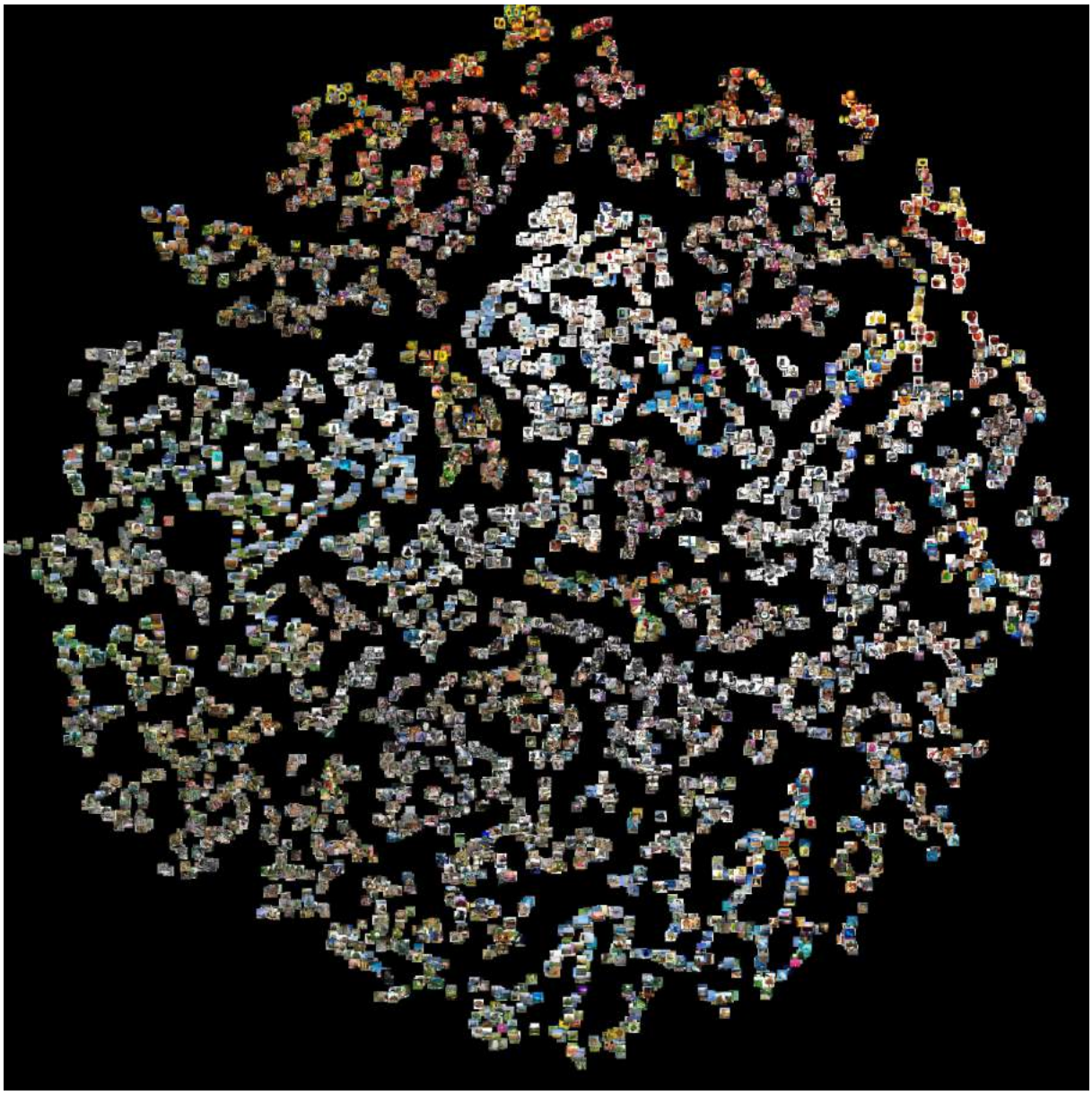


Figure 44: t-SNE plot of gating network logits for NarrowMoE-Rel-4-4.

D.3 PCA Plots of Gating Network Logits

We provide larger versions of our PCA plots from section 5.7. Figures 45 and 46 show the results for NarrowMoE-Imp-1-4 and NarrowMoE-Imp-4-4. The results for NarrowMoE-Rel-1-4 and NarrowMoE-Rel-4-4 are stated in Figures 47 and 48. All four plots are computed on the raw gating network logits resulting on the CIFAR-100 test set. We take the models with the highest test accuracy of each configuration. For further details refer to section 5.7.

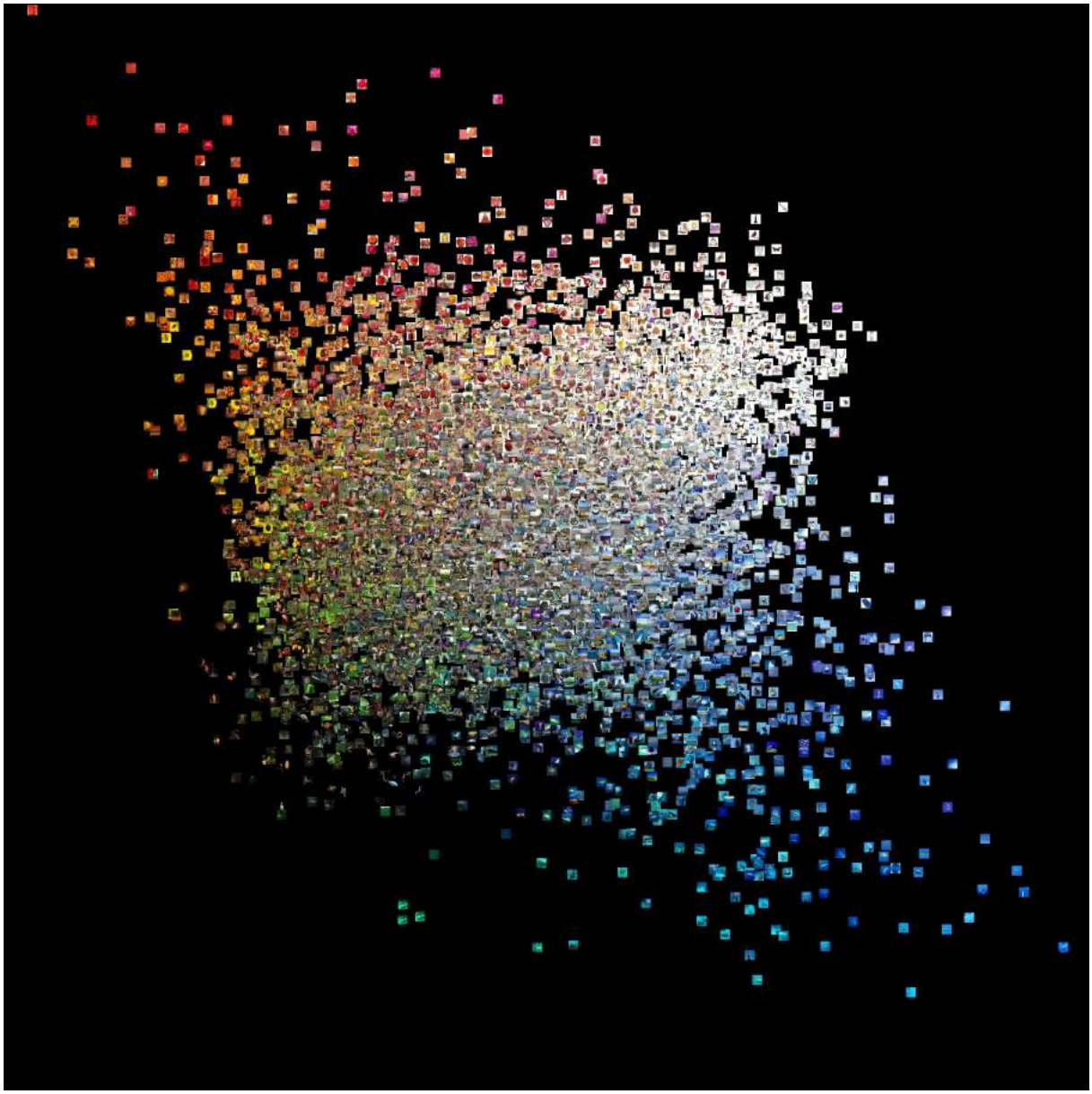


Figure 45: PCA plot of gating network logits for NarrowMoE-Imp-1-4.

Figure 46: PCA plot of gating network logits for NarrowMoE-Imp-4-4.

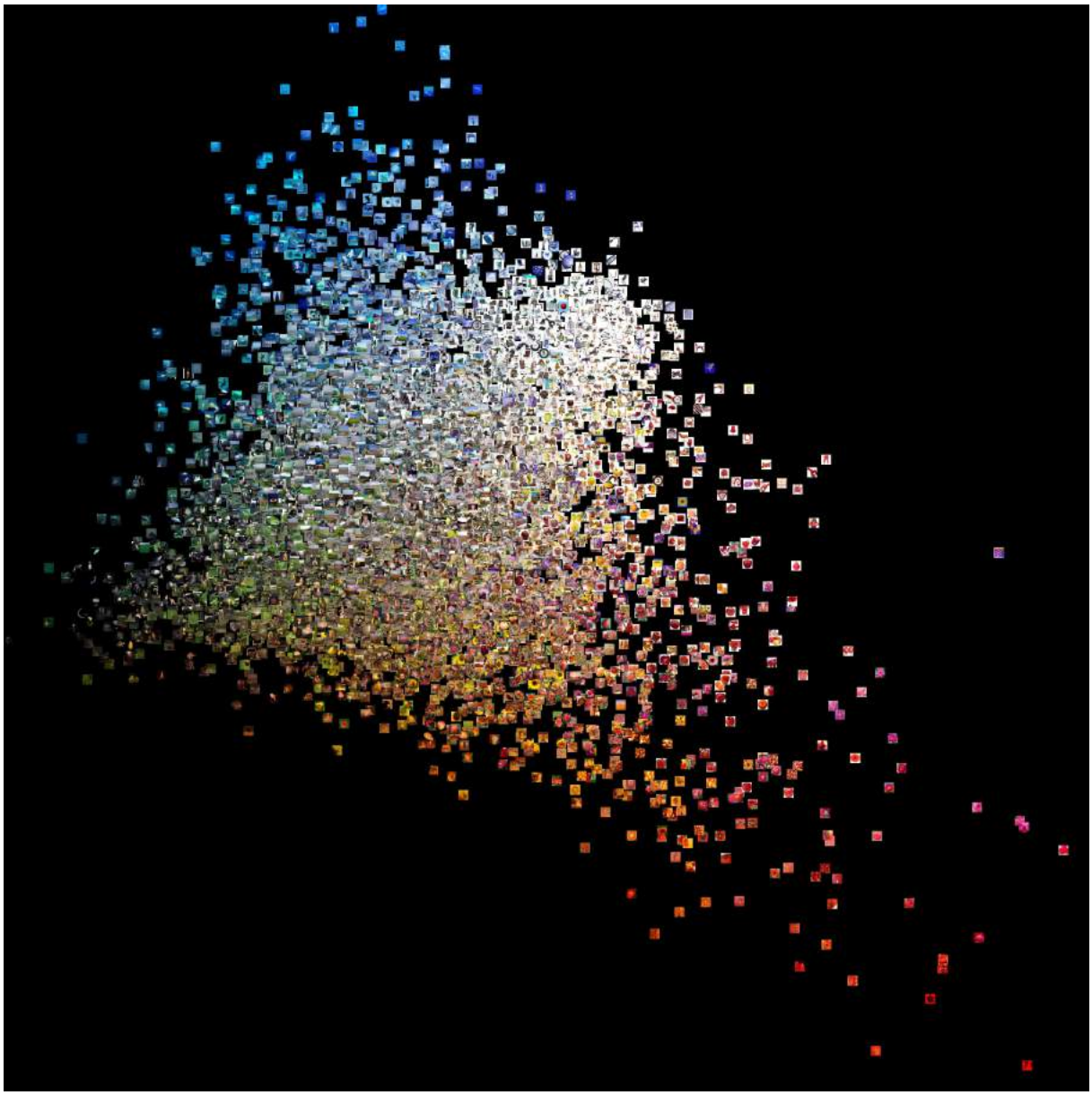


Figure 47: PCA plot of gating network logits for NarrowMoE-Rel-1-4.

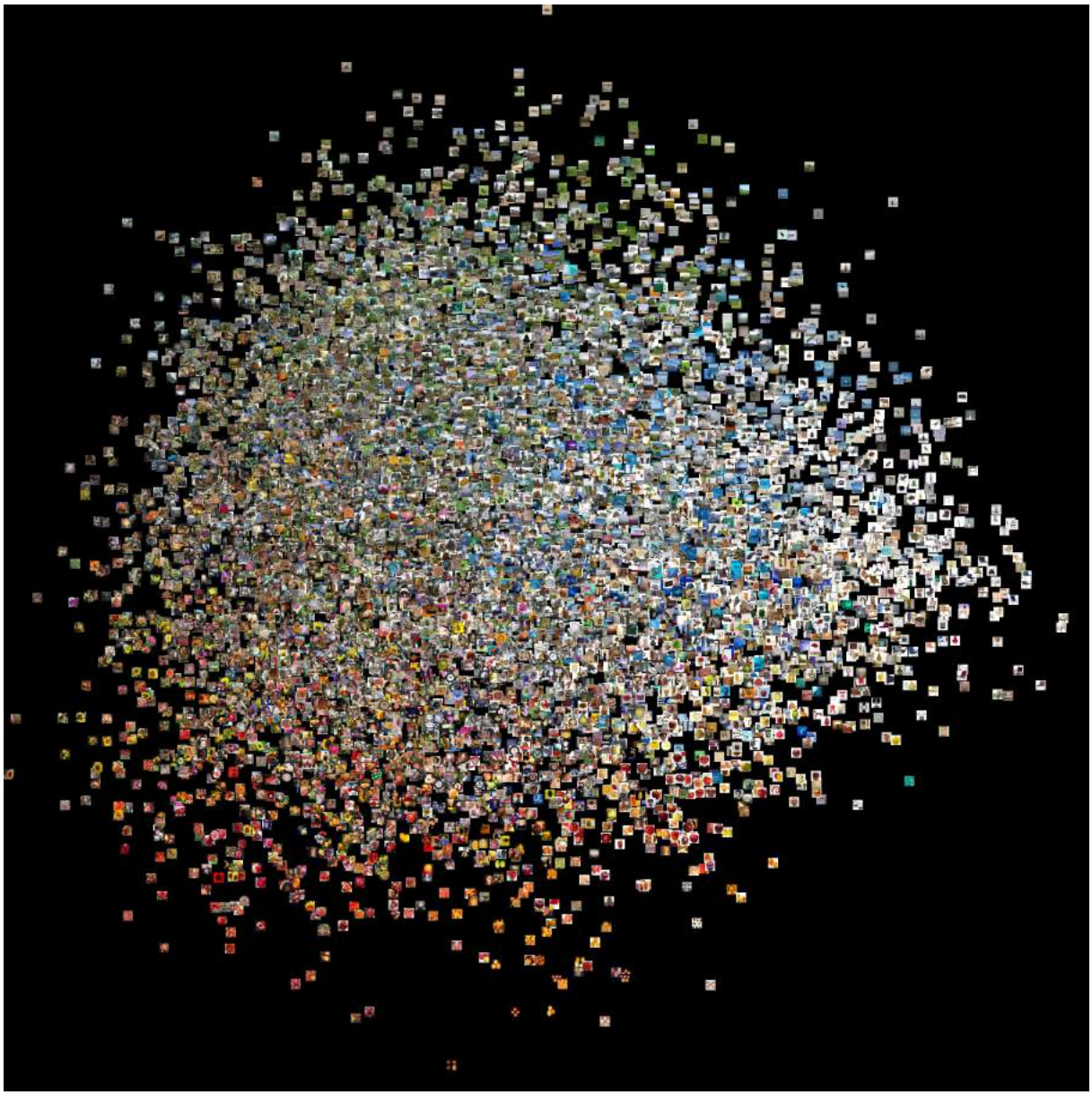


Figure 48: PCA plot of gating network logits for NarrowMoE-Rel-4-4.

D.4 t-SNE Plots of Sparse Weight Vectors

We provide larger versions of our t-SNE plots of the weight vectors from section 5.7. We apply t-SNE on the resulting sparse weight vectors of the MoE models after setting all but the top $k = 2$ weights to zero. Figures 49 and 50 plot the results for NarrowMoE-Imp-1-4 and NarrowMoE-Imp-4-4, respectively. The results for NarrowMoE-Rel-1-4 and NarrowMoE-Rel-4-4 are stated in Figures 51 and 52. All four plots are computed using the CIFAR-100 test set. We take the models with the highest test accuracy of each configuration. For further details refer to section 5.7.

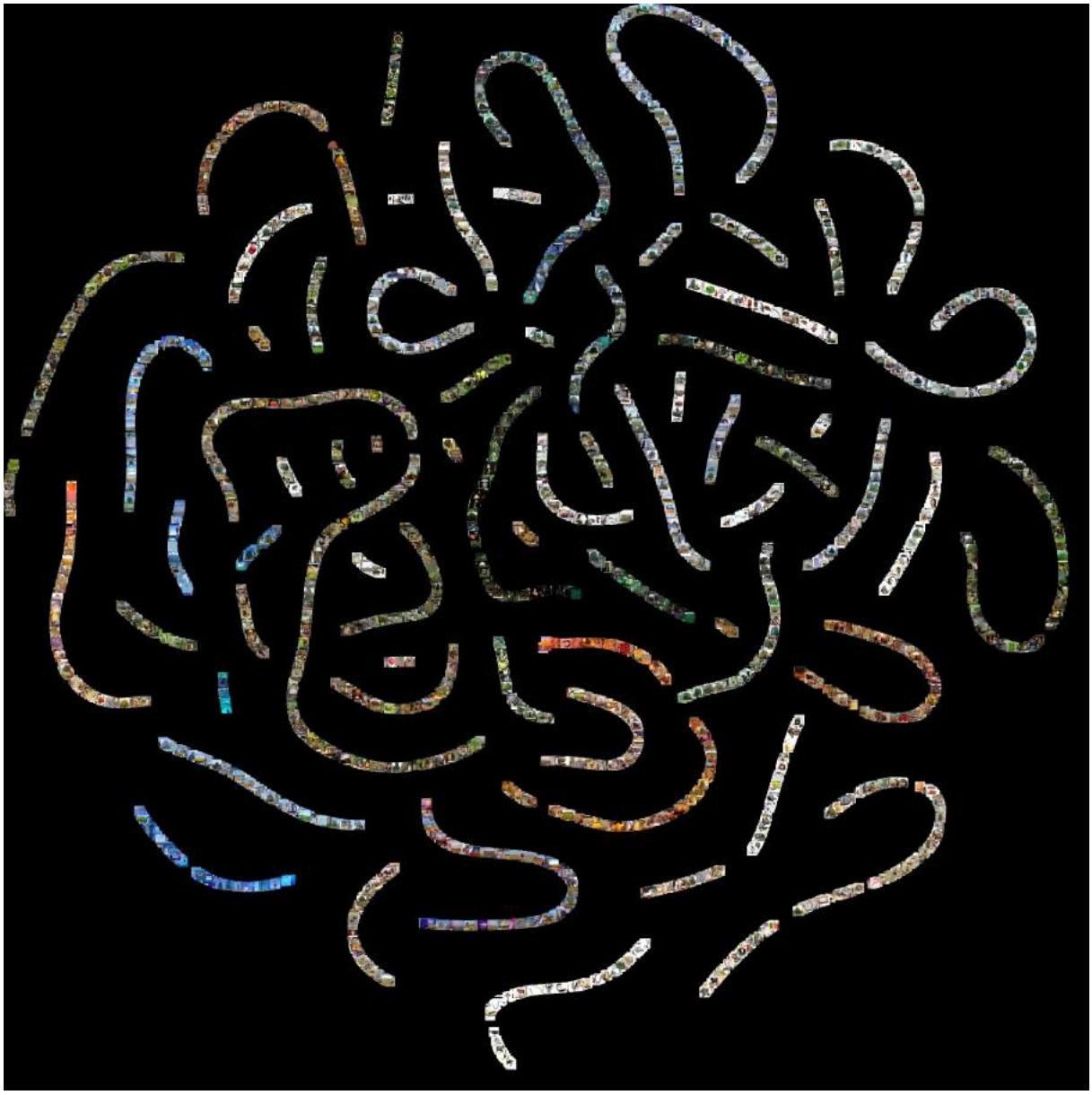


Figure 49: t-SNE plot of gating network weights for NarrowMoE-Imp-1-4.

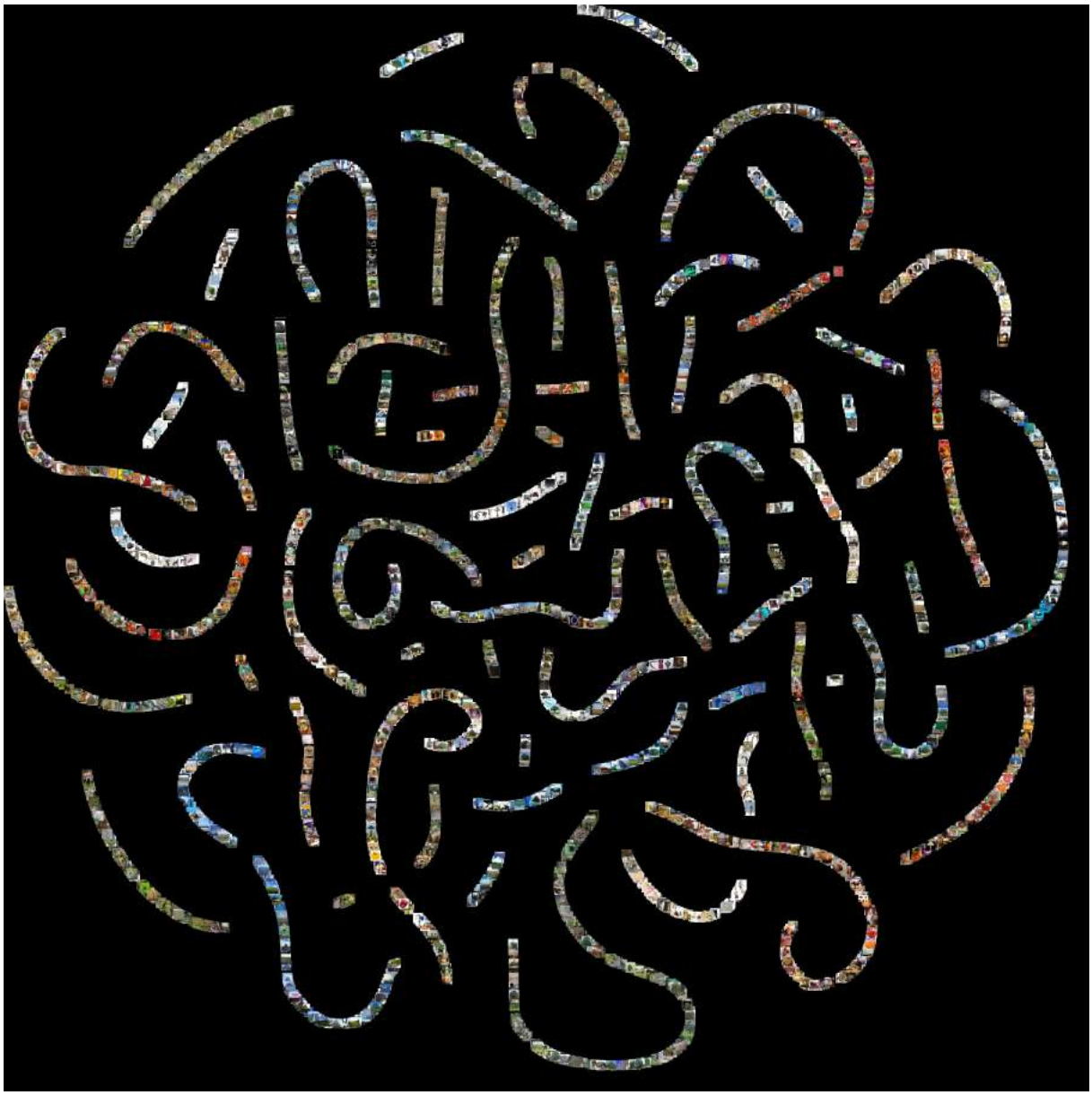


Figure 50: t-SNE plot of gating network weights for NarrowMoE-Imp-4-4.

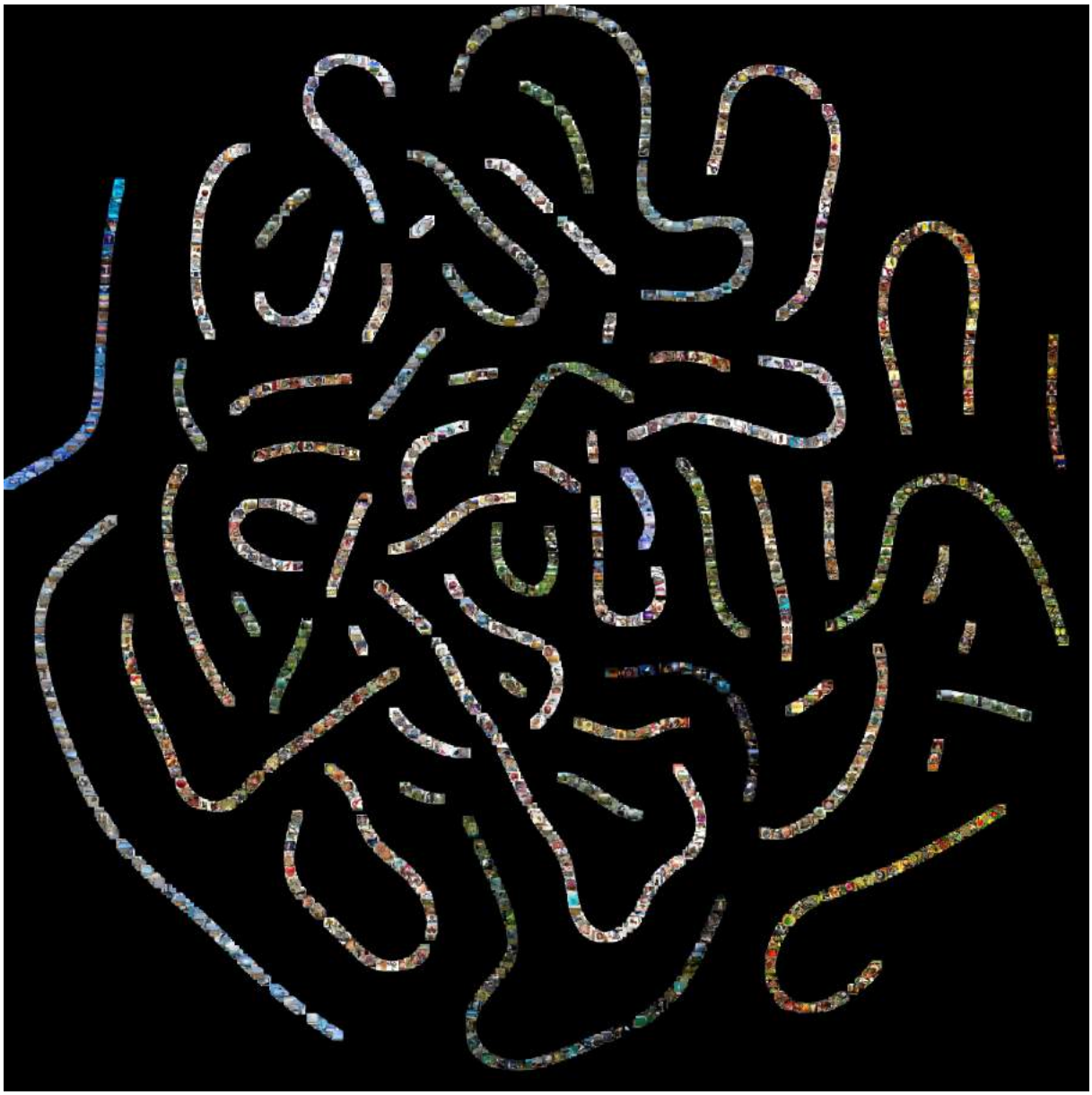


Figure 51: t-SNE plot of gating network weights for NarrowMoE-Rel-1-4.

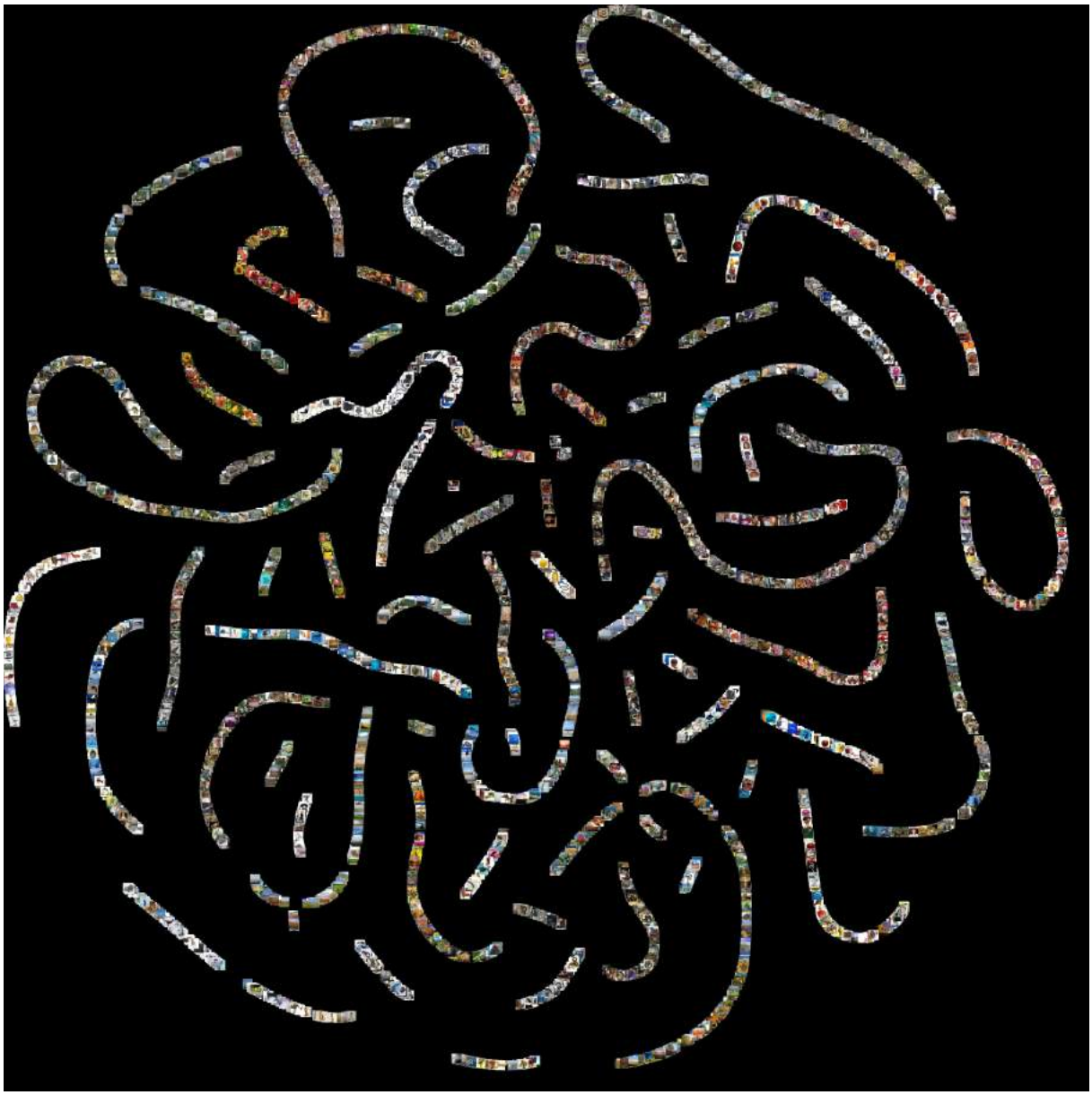


Figure 52: t-SNE plot of gating network weights for NarrowMoE-Rel-4-4.

D.5 Class-wise Accuracy of Soft Constrained MoE Models

Table 56 states the class-wise results for soft constrained models with embedded MoE blocks. Details on the experimental setup are stated in section 5.8. We measure the accuracy metrics for distinct classes on the CIFAR-100 test set. The accuracy for distinct classes are highlighted if the MoE model performs better than the baseline on that specific class.

Class	Baseline	NarrowMoE-Imp-1-4	NarrowMoE-Imp-4-4	NarrowMoE-KL-1-4	NarrowMoE-KL-4-4
apple	89.67%	86.33%	86.00%	86.33%	89.33%
aquarium fish	82.00%	84.00%	82.67%	82.67%	**87.00%**
baby	60.33%	60.33%	58.67%	61.33%	59.33%
bear	55.00%	54.00%	50.00%	55.67%	49.33%
beaver	55.67%	58.33%	**63.00%**	**65.33%**	**62.00%**
bed	77.67%	79.00%	77.67%	76.67%	74.00%
bee	79.67%	80.00%	78.67%	80.67%	81.00%
beetle	71.67%	73.67%	68.67%	71.67%	73.00%
bicycle	85.00%	85.67%	84.00%	85.33%	86.67%
bottle	84.33%	83.67%	81.33%	84.33%	81.33%
bowl	54.67%	57.33%	53.00%	53.00%	51.33%
boy	50.00%	45.33%	45.00%	50.00%	50.00%
bridge	83.67%	83.33%	81.00%	82.67%	78.00%
bus	68.33%	69.00%	66.33%	68.67%	67.00%
butterfly	62.00%	64.33%	61.67%	**65.33%**	**67.00%**
camel	78.00%	80.00%	78.33%	79.33%	75.33%
can	77.00%	73.00%	75.67%	72.33%	72.00%
castle	85.00%	84.33%	85.00%	85.67%	82.00%
caterpillar	67.67%	64.67%	62.00%	64.00%	62.33%
cattle	65.67%	64.00%	64.67%	66.00%	62.00%
chair	88.00%	87.67%	87.67%	87.00%	87.00%
chimpanzee	89.00%	88.33%	89.67%	87.00%	90.67%
clock	73.67%	68.67%	71.67%	71.00%	71.00%
cloud	80.67%	**83.67%**	76.67%	82.33%	79.67%
cockroach	82.67%	83.67%	83.33%	83.00%	83.67%
couch	62.67%	60.33%	60.33%	63.67%	60.00%
crab	65.33%	**70.00%**	**70.00%**	67.00%	66.33%
crocodile	59.00%	59.67%	58.00%	58.67%	**63.33%**
cup	80.67%	81.33%	81.00%	81.67%	80.00%
dinosaur	69.67%	70.67%	67.67%	72.33%	67.00%
dolphin	66.67%	67.00%	67.33%	67.00%	65.33%
elephant	72.33%	71.67%	70.33%	71.67%	72.67%
flatfish	69.33%	65.33%	64.67%	68.00%	63.67%
forest	65.33%	63.33%	62.00%	60.33%	58.33%
fox	78.00%	77.00%	79.00%	78.33%	75.67%
girl	56.67%	49.67%	49.33%	53.00%	53.33%
hamster	81.00%	81.33%	83.67%	80.67%	81.33%
house	78.33%	76.67%	79.33%	77.67%	77.33%
kangaroo	62.67%	**67.33%**	**67.67%**	**67.00%**	63.00%
keyboard	88.00%	85.67%	86.33%	86.67%	88.33%

lamp	64.00%	63.67%	66.00%	63.67%	61.67%
lawn mower	90.00%	89.33%	88.33%	89.33%	88.00%
leopard	69.33%	70.33%	**72.33%**	70.67%	71.67%
lion	80.33%	80.67%	81.00%	82.00%	81.67%
lizard	48.67%	50.33%	44.67%	48.33%	43.33%
lobster	60.67%	58.00%	62.00%	61.67%	59.33%
man	52.33%	50.67%	53.67%	51.00%	51.67%
maple tree	65.67%	66.00%	**68.67%**	**68.67%**	64.00%
motorcycle	94.00%	92.67%	93.67%	95.67%	94.33%
mountain	85.33%	85.33%	86.33%	**88.33%**	85.67%
mouse	50.33%	51.33%	49.33%	50.67%	49.33%
mushroom	72.00%	73.67%	72.33%	72.00%	74.33%
oak tree	69.67%	70.67%	70.67%	70.67%	71.33%
orange	89.33%	91.00%	91.33%	89.67%	89.67%
orchid	87.33%	82.33%	84.00%	83.67%	81.00%
otter	48.33%	47.67%	48.00%	49.00%	46.67%
palm tree	88.67%	88.00%	89.00%	88.00%	87.67%
pear	77.33%	78.00%	78.67%	**81.67%**	79.33%
pickup truck	83.67%	**86.67%**	84.67%	82.33%	84.33%
pine tree	68.33%	67.00%	68.33%	68.00%	68.67%
plain	89.67%	89.00%	86.33%	88.67%	88.33%
plate	69.67%	69.00%	68.67%	70.00%	68.67%
poppy	76.33%	74.33%	73.00%	77.00%	75.67%
porcupine	68.67%	69.00%	70.33%	67.33%	69.33%
possum	51.00%	51.00%	51.33%	50.33%	52.00%
rabbit	58.33%	54.00%	54.33%	58.33%	53.33%
raccoon	77.67%	76.33%	79.00%	78.00%	75.33%
ray	63.67%	63.33%	62.00%	60.33%	60.33%
road	94.00%	93.00%	92.33%	94.00%	92.00%
rocket	83.00%	80.33%	82.67%	79.00%	85.00%
rose	74.00%	73.00%	76.00%	73.67%	76.33%
sea	82.67%	84.00%	80.67%	81.67%	83.67%
seal	48.00%	45.00%	44.33%	48.33%	47.67%
shark	54.00%	52.67%	**59.33%**	55.67%	56.67%
shrew	55.67%	55.67%	57.67%	57.67%	54.67%
skunk	87.33%	85.33%	88.67%	87.00%	87.33%
skyscraper	90.00%	89.67%	90.00%	91.00%	91.33%
snail	67.67%	67.33%	65.67%	64.33%	68.33%
snake	63.00%	65.67%	62.67%	**67.00%**	63.67%
spider	78.00%	77.67%	75.00%	77.67%	74.33%
squirrel	56.33%	**59.33**	51.67%	58.00%	55.67%
streetcar	74.67%	74.67%	75.33%	77.00%	76.67%
sunflower	91.67%	91.33%	89.67%	91.00%	91.33%
sweet pepper	69.67%	65.00%	70.67%	68.67%	69.00%
table	68.67%	66.00%	66.33%	69.00%	69.00%
tank	83.00%	77.00%	81.33%	82.67%	82.00%
telephone	74.00%	73.00%	74.00%	72.33%	71.67%
television	83.00%	85.67%	82.00%	**87.00%**	83.67%
tiger	78.00%	78.67%	78.67%	79.33%	78.33%
tractor	89.67%	88.00%	86.33%	88.67%	88.33%

train	81.67%	82.67%	81.33%	81.33%	79.67%
trout	79.00%	80.33%	78.67%	81.33%	78.67%
tulip	61.33%	**65.00%**	59.67%	**64.33%**	60.67%
turtle	59.33%	56.00%	53.33%	57.67%	56.33%
wardrobe	94.33%	92.33%	93.67%	93.33%	92.67%
whale	70.67%	72.33%	**74.00%**	71.00%	69.00%
willow tree	65.67%	**70.00%**	**70.00%**	64.67%	67.00%
wolf	74.67%	74.33%	75.33%	74.00%	73.67%
woman	55.33%	53.67%	49.33%	53.67%	49.33%
worm	71.00%	70.67%	71.33%	**74.00%**	71.67%
average	72.62%	72.24%	71.95%	72.72%	71.80%

Table 56: Prediction accuracies for each distinct class in CIFAR-100 evaluated for soft constrained MoE models using importance loss (Imp) and KL divergence loss (KL).

D.6 Class-wise Accuracy of Hard Constrained MoE Models

Table 57 states the class-wise results for hard constrained models with embedded MoE blocks. Details on the experimental setup are stated in section 5.8. We measure the accuracy metrics for distinct classes on the CIFAR-100 test set. The accuracy for distinct classes are highlighted if the MoE model performs better than the baseline on that specific class.

Class	Baseline	NarrowMoE-Rel-1-4	NarrowMoE-Rel-4-4	NarrowMoE-Mean-1-4	NarrowMoE-Mean-4-4
apple	89.67%	85.33%	87.33%	86.67%	88.67%
aquarium fish	82.00%	83.00%	**86.67%**	84.67%	84.33%
baby	60.33%	58.33%	61.67%	**63.33%**	58.00%
bear	55.00%	54.67%	55.00%	54.67%	52.00%
beaver	55.67%	56.33%	57.33%	**59.33%**	58.00%
bed	77.67%	**80.67%**	77.67%	77.67%	72.33%
bee	79.67%	80.00%	78.33%	80.00%	78.00%
beetle	71.67%	**75.67%**	71.33%	**74.67%**	70.00%
bicycle	85.00%	85.67%	**88.00%**	86.33%	86.33%
bottle	84.33%	82.00%	85.00%	86.00%	85.00%
bowl	54.67%	52.67%	50.33%	55.33%	51.33%
boy	50.00%	46.33%	51.33%	48.67%	44.67%
bridge	83.67%	80.67%	80.33%	83.33%	79.67%
bus	68.33%	67.00%	66.33%	67.33%	70.33%
butterfly	62.00%	64.33%	**68.00%**	60.00%	63.00%
camel	78.00%	75.33%	79.00%	78.67%	80.67%
can	77.00%	75.33%	74.00%	77.33%	76.33%
castle	85.00%	84.67%	87.00%	87.33%	84.33%
caterpillar	67.67%	64.33%	69.67%	66.33%	63.67%
cattle	65.67%	63.33%	67.33%	63.67%	62.33%
chair	88.00%	88.00%	87.33%	87.33%	86.33%
chimpanzee	89.00%	90.33%	89.67%	89.33%	89.67%
clock	73.67%	72.00%	74.33%	72.33%	74.00%
cloud	80.67%	**85.33%**	82.67%	83.00%	80.33%
cockroach	82.67%	83.33%	80.00%	85.33%	85.33%
couch	62.67%	58.33%	63.33%	58.33%	61.33%
crab	65.33%	**68.33%**	**70.33%**	68.00%	**70.33%**
crocodile	59.00%	56.33%	60.00%	**63.33%**	59.67%
cup	80.67%	79.67%	82.67%	81.00%	79.33%
dinosaur	69.67%	69.00%	68.67%	72.33%	71.00%
dolphin	66.67%	**70.00%**	64.00%	64.33%	68.33%
elephant	72.33%	71.00%	71.00%	71.67%	73.00%
flatfish	69.33%	68.33%	65.00%	67.67%	64.67%
forest	65.33%	62.67%	67.33%	62.67%	63.67%
fox	78.00%	75.33%	78.33%	74.33%	79.33%
girl	56.67%	51.33%	52.67%	53.67%	55.33%
hamster	81.00%	80.67%	83.33%	83.00%	81.00%
house	78.33%	75.67%	77.33%	79.00%	80.33%
kangaroo	62.67%	64.33%	63.67%	**66.33%**	63.33%
keyboard	88.00%	88.33%	88.33%	88.67%	87.33%

lamp	64.00%	62.67%	62.67%	**67.67%**	65.33%
lawn mower	90.00%	90.00%	89.00%	91.67%	90.33%
leopard	69.33%	69.00%	**76.67%**	71.33%	70.33%
lion	80.33%	80.33%	79.33%	81.33%	80.00%
lizard	48.67%	49.00%	**54.00%**	48.67%	48.33%
lobster	60.67%	61.33%	62.00%	61.67%	**64.00%**
man	52.33%	49.33%	52.33%	52.67%	54.33%
maple tree	65.67%	68.33%	66.33%	67.00%	**68.67%**
motorcycle	94.00%	94.00%	91.67%	94.67%	94.33%
mountain	85.33%	87.33%	**89.33%**	85.67%	**88.33%**
mouse	50.33%	47.33%	**54.33%**	**54.33%**	51.67%
mushroom	72.00%	73.33%	**75.67%**	72.67%	72.67%
oak tree	69.67%	68.00%	66.00%	70.00%	66.67%
orange	89.33%	92.00%	91.67%	89.33%	91.67%
orchid	87.33%	81.67%	86.00%	86.67%	83.00%
otter	48.33%	49.67%	50.33%	49.00%	50.33%
palm tree	88.67%	88.33%	89.67%	91.00%	88.33%
pear	77.33%	76.33%	76.67%	77.00%	78.33%
pickup truck	83.67%	85.67%	85.33%	85.00%	86.00%
pine tree	68.33%	63.33%	65.33%	68.00%	64.67%
plain	89.67%	88.33%	91.33%	89.33%	89.33%
plate	69.67%	68.00%	68.00%	68.33%	69.33%
poppy	76.33%	76.33%	75.00%	76.67%	75.67%
porcupine	68.67%	70.00%	67.67%	69.00%	68.33%
possum	51.00%	50.33%	**58.33%**	**55.33%**	**56.00%**
rabbit	58.33%	59.33%	56.33%	**61.33%**	59.00%
raccoon	77.67%	78.33%	81.00%	76.67%	78.33%
ray	63.67%	60.00%	61.67%	60.00%	64.00%
road	94.00%	95.67%	94.00%	92.67%	94.67%
rocket	83.00%	82.00%	84.67%	80.33%	82.67%
rose	74.00%	72.33%	74.00%	74.00%	68.33%
sea	82.67%	80.67%	80.33%	82.33%	81.33%
seal	48.00%	43.33%	44.00%	46.00%	46.67%
shark	54.00%	55.33%	**57.67%**	55.67%	54.00%
shrew	55.67%	56.33%	**61.00%**	52.33%	55.67%
skunk	87.33%	89.33%	86.33%	89.33%	88.33%
skyscraper	90.00%	89.67%	91.67%	91.00%	89.33%
snail	67.67%	66.67%	65.67%	63.33%	66.33%
snake	63.00%	**66.00%**	63.67%	**66.67%**	60.33%
spider	78.00%	77.67%	76.33%	79.67%	77.67%
squirrel	56.33%	56.33%	58.33%	**60.67%**	58.00%
streetcar	74.67%	75.00%	77.67%	75.67%	77.33%
sunflower	91.67%	92.67%	91.00%	93.67%	92.67%
sweet pepper	69.67%	69.00%	72.33%	69.00%	71.00%
table	68.67%	**72.33%**	71.67%	68.33%	70.00%
tank	83.00%	77.67%	82.00%	82.67%	81.00%
telephone	74.00%	72.00%	75.67%	73.33%	72.33%
television	83.00%	83.67%	83.00%	84.67%	83.33%
tiger	78.00%	80.33%	78.67%	**81.33%**	**81.33%**
tractor	89.67%	86.33%	89.00%	88.33%	85.00%

train	81.67%	82.33%	82.67%	82.33%	84.33%
trout	79.00%	81.00%	80.33%	79.33%	79.33%
tulip	61.33%	62.67%	57.00%	61.00%	59.33%
turtle	59.33%	58.33%	**62.00%**	58.00%	57.67%
wardrobe	94.33%	92.00%	92.00%	92.67%	94.00%
whale	70.67%	72.00%	**73.67%**	71.00%	73.33%
willow tree	65.67%	67.00%	64.33%	65.33%	66.00%
wolf	74.67%	77.00%	76.67%	**78.00%**	73.00%
woman	55.33%	54.33%	55.33%	53.00%	58.00%
worm	71.00%	72.33%	71.67%	73.67%	**74.33%**
average	72.62%	72.21%	73.10%	73.00%	72.57%

Table 57: Prediction accuracies for each distinct class in CIFAR-100 evaluated for different hard constrained MoE models using relative importance (Rel) and mean importance (Mean) constraints.

D.7 Confusion Analysis of the Baseline

Table 58 states the top-3 predicted labels the baseline models predict for input samples of a specific class. We measure the values on the CIFAR-100 test set. The results provide insights into the most confused classes of the ResNet-18 baseline models. We refer to this data in section 5.8. We highlight falsely predicted classes if they have a share of at least 10%.

Class	Top Prediction	Share	2nd Top Prediction	Share	3rd Top Prediction	Share
apple	apple	89%	pear	5%	bowl	2%
aquarium fish	aquarium fish	83%	orchid	2%	lobster	2%
baby	baby	62%	**boy**	**11%**	girl	6%
bear	bear	56%	otter	5%	seal	5%
beaver	beaver	52%	**shrew**	**13%**	porcupine	7%
bed	bed	82%	couch	7%	table	4%
bee	bee	81%	spider	4%	beetle	2%
beetle	beetle	70%	cockroach	6%	spider	3%
bicycle	bicycle	86%	motorcycle	3%	bus	2%
bottle	bottle	86%	tractor	3%	pear	2%
bowl	bowl	55%	plate	7%	clock	5%
boy	boy	49%	**baby**	**13%**	**girl**	**12%**
bridge	bridge	84%	forest	2%	streetcar	2%
bus	bus	70%	**streetcar**	**11%**	pickup_truck	5%
butterfly	butterfly	63%	caterpillar	4%	lobster	3%
camel	camel	77%	lion	3%	pear	2%
can	can	83%	cup	2%	television	2%
castle	castle	82%	house	7%	mountain	3%
caterpillar	caterpillar	65%	lizard	7%	bee	6%
cattle	cattle	69%	camel	5%	elephant	3%
chair	chair	89%	couch	6%	crab	1%
chimpanzee	chimpanzee	91%	raccoon	1%	otter	1%
clock	clock	74%	plate	6%	telephone	3%
cloud	cloud	78%	mountain	6%	sea	5%
cockroach	cockroach	84%	beetle	5%	lobster	3%
couch	couch	59%	**bed**	**14%**	chair	6%
crab	crab	62%	**lobster**	**11%**	spider	2%
crocodile	crocodile	60%	lizard	6%	turtle	5%
cup	cup	80%	bowl	5%	can	4%
dinosaur	dinosaur	66%	lizard	3%	fox	2%
dolphin	dolphin	62%	**shark**	**12%**	whale	7%
elephant	elephant	66%	chimpanzee	4%	cattle	3%
flatfish	flatfish	68%	ray	4%	aquarium fish	3%
forest	forest	62%	willow tree	7%	mountain	3%
fox	fox	80%	camel	3%	seal	3%
girl	girl	58%	**boy**	**12%**	**woman**	**10%**
hamster	hamster	82%	mouse	5%	possum	3%
house	house	77%	streetcar	5%	bridge	3%
kangaroo	kangaroo	62%	fox	6%	crocodile	3%
keyboard	keyboard	85%	bed	2%	worm	1%

lamp	lamp	63%	cup	4%	can	3%
lawn mower	lawn mower	91%	bus	1%	tractor	1%
leopard	leopard	72%	tiger	6%	fox	4%
lion	lion	79%	camel	2%	fox	2%
lizard	lizard	51%	crocodile	8%	snake	6%
lobster	lobster	62%	crab	3%	snake	2%
man	man	51%	**woman**	**11%**	**boy**	**11%**
maple tree	maple tree	63%	**oak tree**	**17%**	willow tree	9%
motorcycle	motorcycle	95%	beetle	1%	pickup truck	1%
mountain	mountain	85%	sea	3%	whale	2%
mouse	mouse	50%	**shrew**	**10%**	rabbit	8%
mushroom	mushroom	70%	snail	6%	pine tree	2%
oak tree	oak tree	68%	**maple tree**	**18%**	pine tree	7%
orange	orange	93%	pear	3%	sweet pepper	2%
orchid	orchid	88%	snake	2%	butterfly	1%
otter	otter	51%	**seal**	**10%**	bear	3%
palm tree	palm tree	89%	pine tree	5%	sunflower	2%
pear	pear	77%	orange	4%	snail	2%
pickup truck	pickup truck	84%	bus	5%	bridge	2%
pine tree	pine tree	69%	**oak tree**	**13%**	willow tree	9%
plain	plain	90%	sea	5%	mountain	1%
plate	plate	74%	bowl	9%	clock	5%
poppy	poppy	74%	**tulip**	**11%**	rose	3%
porcupine	porcupine	66%	beaver	4%	shrew	4%
possum	possum	53%	mouse	8%	wolf	6%
rabbit	rabbit	51%	kangaroo	6%	mouse	4%
raccoon	raccoon	77%	mouse	3%	possum	3%
ray	ray	63%	shark	6%	seal	4%
road	road	94%	bridge	2%	sea	1%
rocket	rocket	84%	skyscraper	2%	lamp	2%
rose	rose	80%	tulip	5%	orchid	3%
sea	sea	78%	cloud	7%	plain	6%
seal	seal	48%	**otter**	**10%**	snail	5%
shark	shark	57%	dolphin	6%	ray	4%
shrew	shrew	57%	**mouse**	**12%**	porcupine	4%
skunk	skunk	85%	raccoon	2%	possum	2%
skyscraper	skyscraper	93%	bus	1%	otter	1%
snail	snail	69%	butterfly	3%	lizard	2%
snake	snake	61%	**worm**	**10%**	lizard	9%
spider	spider	78%	bee	3%	beetle	2%
squirrel	squirrel	62%	otter	3%	wolf	3%
streetcar	streetcar	78%	train	6%	bus	6%
sunflower	sunflower	94%	poppy	2%	tiger	1%
sweet pepper	sweet pepper	72%	pear	7%	orange	5%
table	table	69%	couch	4%	bed	3%
tank	tank	86%	pickup truck	2%	train	2%
telephone	telephone	74%	can	3%	keyboard	3%
television	television	81%	telephone	2%	can	2%
tiger	tiger	80%	fox	3%	lion	3%
tractor	tractor	86%	motorcycle	2%	camel	1%

train	train	82%	streetcar	9%	bus	3%
trout	trout	79%	caterpillar	3%	rocket	2%
tulip	tulip	63%	**rose**	**10%**	poppy	6%
turtle	turtle	57%	dolphin	4%	shark	3%
wardrobe	wardrobe	95%	rabbit	1%	bottle	1%
whale	whale	68%	**dolphin**	**11%**	shark	9%
willow tree	willow tree	64%	**maple tree**	**14%**	oak tree	7%
wolf	wolf	71%	kangaroo	3%	raccoon	3%
woman	woman	59%	**girl**	**16%**	boy	7%
worm	worm	67%	**snake**	**14%**	plate	4%

Table 58: Top-3 label predictions of baseline models on the CIFAR-100 test set. Values are computed across three runs. Falsely predicted labels with at least 10% share are highlighted.

D.8 Varying Active Experts for NarrowMoE Models

We analyze in section 5.9 how models perform for varying the number of activate experts k in each forward pass. We evaluate performances on the CIFAR-100 test set and average accuracy values on three runs. Table 59 states the results for models with ten experts. We further state the computational complexity for a varying number of active experts in Table 60.

Model	Constraint	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Baseline		72.62%									
Pos=1, Exp=10	Importance Loss	71.41%	71.51%	71.52%	71.54%	**71.55%**	71.53%	71.49%	71.47%	71.49%	71.48%
Pos=2, Exp=10	Importance Loss	71.52%	**71.76%**	71.69%	71.75%	71.74%	71.66%	71.66%	71.60%	71.60%	71.57%
Pos=3, Exp=10	Importance Loss	70.37%	71.47%	71.67%	71.72%	**71.74%**	71.71%	71.66%	71.64%	71.66%	71.62%
Pos=4, Exp=10	Importance Loss	68.52%	71.61%	72.58%	72.70%	73.00%	73.04%	73.09%	73.12%	**73.14%**	73.08%
Pos=1, Exp=10	KL Divergence Loss	71.05%	71.32%	71.37%	71.33%	71.37%	**71.40%**	71.39%	71.37%	71.35%	71.38%
Pos=2, Exp=10	KL Divergence Loss	71.69%	71.87%	**72.00%**	71.97%	71.97%	71.94%	71.88%	71.92%	71.89%	71.87%
Pos=3, Exp=10	KL Divergence Loss	70.35%	71.43%	71.69%	71.75%	71.79%	**71.84%**	71.84%	71.83%	71.83%	71.81%
Pos=4, Exp=10	KL Divergence Loss	69.38%	71.99%	72.91%	73.10%	73.17%	**73.22%**	73.20%	**73.22%**	73.21%	73.18%
Pos=1, Exp=10	Mean Importance	72.02%	72.28%	72.28%	**72.29%**	72.28%	72.27%	72.27%	72.26%	72.27%	72.28%
Pos=2, Exp=10	Mean Importance	71.89%	72.08%	**72.11%**	72.09%	71.99%	71.94%	71.85%	71.84%	71.86%	71.87%
Pos=3, Exp=10	Mean Importance	71.56%	72.05%	**72.23%**	72.06%	71.51%	70.64%	70.17%	69.79%	69.60%	69.60%
Pos=4, Exp=10	Mean Importance	71.52%	73.09%	**73.34%**	73.30%	73.21%	73.23%	73.17%	73.14%	73.16%	73.16%
Pos=1, Exp=10	Relative Importance	71.66%	71.60%	71.62%	71.61%	71.65%	71.64%	**71.69%**	71.66%	71.68%	71.66%
Pos=2, Exp=10	Relative Importance	71.52%	71.51%	**71.56%**	71.51%	71.49%	71.51%	71.52%	71.53%	71.53%	71.52%
Pos=3, Exp=10	Relative Importance	70.82%	70.82%	70.84%	**70.87%**	70.87%	**70.87%**	70.86%	70.85%	70.85%	70.85%
Pos=4, Exp=10	Relative Importance	72.56%	72.84%	73.08%	73.11%	73.14%	73.15%	73.14%	73.15%	**73.16%**	73.16%

Table 59: Evaluation results for NarrowMoE models with 10 experts and a varying number k of active experts. We highlight the best results for each model setup.

	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10
Pos=1	0.48 GMac	0.56 GMac	0.64 GMac	0.71 GMac	0.79 GMac	0.87 GMac	0.94 GMac	1.02 GMac	1.1 GMac	1.18 GMac
Pos=2	0.49 GMac	0.56 GMac	0.63 GMac	0.7 GMac	0.77 GMac	0.84 GMac	0.9 GMac	0.97 GMac	1.04 GMac	1.11 GMac
Pos=3	0.49 GMac	0.55 GMac	0.62 GMac	0.68 GMac	0.75 GMac	0.81 GMac	0.88 GMac	0.94 GMac	1.01 GMac	1.07 GMac
Pos=4	0.49 GMac	0.55 GMac	0.62 GMac	0.68 GMac	0.75 GMac	0.81 GMac	0.88 GMac	0.94 GMac	1.01 GMac	1.07 GMac

Table 60: Computational complexity for different NarrowMoE models with 10 experts.

D.9 Expert-wise Accuracy for NarrowMoE-Imp-1-4

Table 61 states the class-wise evaluation results for each expert in NarrowMoE-Imp-1-4. We only activate one specific expert during the evaluation and assign all weights to it. Further details on the experimental setup are stated in section 5.10. The accuracy for distinct classes are highlighted if the expert performs better than the baseline on that specific class. Additionally, we also state the averagely assigned weights $\bar{g}_i$ to distinct experts per class.

Class	Acc_{MoE}	Acc_1	$\bar{g}_1$	Acc_2	Imp2	Acc_3	$\bar{g}_3$	Acc_4	$\bar{g}_4$
apple	**88%**	**88%**	0.59	87%	0.09	85%	0.23	84%	0.09
aquarium fish	84%	81%	0.41	**85%**	0.30	84%	0.05	83%	0.23
baby	58%	59%	0.39	**60%**	0.07	**60%**	0.29	**60%**	0.24
bear	54%	**56%**	0.25	55%	0.23	50%	0.30	54%	0.23
beaver	58%	58%	0.33	58%	0.22	**63%**	0.24	61%	0.22
bed	**80%**	**78%**	0.30	77%	0.06	76%	0.38	77%	0.25
bee	79%	77%	0.49	76%	0.17	**80%**	0.18	78%	0.15
beetle	76%	**78%**	0.30	77%	0.16	71%	0.33	75%	0.21
bicycle	**87%**	86%	0.11	82%	0.08	82%	0.42	**87%**	0.39
bottle	84%	81%	0.23	83%	0.13	83%	0.36	**85%**	0.28
bowl	54%	53%	0.31	54%	0.10	**58%**	0.35	55%	0.25
boy	48%	47%	0.36	48%	0.12	45%	0.29	**50%**	0.23
bridge	84%	**87%**	0.07	83%	0.06	82%	0.32	84%	0.55
bus	66%	64%	0.14	66%	0.06	66%	0.38	**67%**	0.41
butterfly	67%	63%	0.36	**69%**	0.31	65%	0.22	67%	0.12
camel	79%	78%	0.33	78%	0.14	75%	0.30	**81%**	0.23
can	74%	70%	0.20	**77%**	0.06	73%	0.43	74%	0.31
castle	83%	81%	0.06	81%	0.02	82%	0.42	**85%**	0.50
caterpillar	64%	59%	0.36	**65%**	0.40	62%	0.12	**65%**	0.12
cattle	63%	63%	0.31	65%	0.24	64%	0.28	**67%**	0.17
chair	**89%**	88%	0.10	**89%**	0.02	86%	0.59	86%	0.29
chimpanzee	90%	**91%**	0.21	89%	0.30	89%	0.28	86%	0.20
clock	72%	**74%**	0.23	70%	0.07	67%	0.37	69%	0.33
cloud	82%	82%	0.09	**88%**	0.04	79%	0.26	79%	0.60
cockroach	**84%**	82%	0.09	**84%**	0.02	**84%**	0.55	**84%**	0.34
couch	58%	60%	0.28	54%	0.12	**61%**	0.32	58%	0.28
crab	**74%**	69%	0.35	**72%**	0.15	66%	0.28	69%	0.21
crocodile	**60%**	57%	0.31	56%	0.23	56%	0.24	**60%**	0.22
cup	**81%**	**80%**	0.15	**80%**	0.12	**80%**	0.37	79%	0.36
dinosaur	70%	71%	0.24	69%	0.16	66%	0.36	**72%**	0.24
dolphin	**70%**	68%	0.00	69%	0.05	**70%**	0.17	69%	0.78
elephant	67%	**68%**	0.23	66%	0.16	67%	0.36	67%	0.25
flatfish	66%	**67%**	0.16	64%	0.07	**67%**	0.36	64%	0.40
forest	63%	**67%**	0.32	58%	0.38	64%	0.12	66%	0.18
fox	**79%**	76%	0.38	73%	0.21	**79%**	0.24	75%	0.17
girl	44%	46%	0.36	**47%**	0.12	**47%**	0.32	43%	0.20
hamster	80%	80%	0.32	77%	0.09	**82%**	0.32	75%	0.26
house	74%	78%	0.09	**79%**	0.04	75%	0.42	78%	0.45
kangaroo	**68%**	64%	0.29	**67%**	0.20	65%	0.29	64%	0.21
keyboard	87%	**90%**	0.15	85%	0.07	87%	0.38	87%	0.40

lamp	**67%**	66%	0.22	**67%**	0.14	66%	0.31	65%	0.33
lawn mower	88%	88%	0.17	88%	0.13	88%	0.43	**89%**	0.28
leopard	69%	68%	0.32	66%	0.22	**70%**	0.25	68%	0.21
lion	83%	**85%**	0.55	79%	0.22	83%	0.16	81%	0.06
lizard	**55%**	**55%**	0.29	53%	0.16	52%	0.32	53%	0.22
lobster	61%	62%	0.39	**65%**	0.16	**65%**	0.23	62%	0.22
man	53%	48%	0.24	56%	0.17	51%	0.30	**59%**	0.29
maple tree	**69%**	66%	0.33	68%	0.17	66%	0.24	**69%**	0.27
motorcycle	**94%**	**94%**	0.17	93%	0.11	93%	0.41	**94%**	0.31
mountain	87%	86%	0.05	87%	0.05	85%	0.25	**88%**	0.66
mouse	48%	**50%**	0.33	49%	0.15	49%	0.31	**50%**	0.21
mushroom	**74%**	73%	0.42	70%	0.34	73%	0.16	**74%**	0.07
oak tree	68%	69%	0.07	65%	0.17	70%	0.32	**73%**	0.44
orange	91%	90%	0.71	92%	0.11	**93%**	0.15	91%	0.03
orchid	**86%**	**86%**	0.36	**86%**	0.19	**86%**	0.23	84%	0.22
otter	53%	**56%**	0.18	48%	0.16	50%	0.30	**56%**	0.35
palm tree	**88%**	85%	0.10	85%	0.10	86%	0.35	**87%**	0.45
pear	**81%**	**80%**	0.39	78%	0.18	**80%**	0.31	**80%**	0.12
pickup truck	87%	86%	0.22	88%	0.11	87%	0.33	**89%**	0.33
pine tree	**67%**	66%	0.11	66%	0.11	64%	0.33	**67%**	0.45
plain	88%	88%	0.13	**89%**	0.04	87%	0.30	87%	0.53
plate	69%	69%	0.11	67%	0.04	64%	0.45	**70%**	0.39
poppy	75%	75%	0.71	73%	0.14	**78%**	0.06	76%	0.08
porcupine	71%	**72%**	0.32	67%	0.29	69%	0.21	69%	0.18
possum	54%	53%	0.30	51%	0.26	54%	0.24	**55%**	0.20
rabbit	51%	**55%**	0.31	49%	0.27	54%	0.22	54%	0.19
raccoon	72%	**77%**	0.27	72%	0.24	70%	0.29	74%	0.21
ray	**66%**	64%	0.09	**65%**	0.12	63%	0.18	62%	0.60
road	**96%**	94%	0.04	93%	0.08	**95%**	0.36	94%	0.52
rocket	76%	71%	0.07	**78%**	0.05	73%	0.30	75%	0.58
rose	71%	70%	0.64	73%	0.16	**76%**	0.12	72%	0.08
sea	**86%**	**85%**	0.09	81%	0.04	83%	0.24	**85%**	0.63
seal	**50%**	47%	0.11	**49%**	0.12	**49%**	0.32	45%	0.45
shark	52%	50%	0.05	52%	0.15	**54%**	0.12	50%	0.68
shrew	**58%**	55%	0.38	56%	0.27	**58%**	0.22	56%	0.13
skunk	85%	86%	0.27	86%	0.29	82%	0.28	**87%**	0.16
skyscraper	86%	**90%**	0.03	86%	0.03	87%	0.29	86%	0.65
snail	67%	63%	0.32	62%	0.27	66%	0.24	**70%**	0.17
snake	**72%**	67%	0.31	**69%**	0.12	66%	0.33	**69%**	0.24
spider	**80%**	75%	0.23	**80%**	0.25	78%	0.27	79%	0.25
squirrel	59%	**60%**	0.29	57%	0.19	**60%**	0.28	59%	0.24
streetcar	76%	**80%**	0.19	77%	0.11	78%	0.34	77%	0.37
sunflower	92%	91%	0.56	90%	0.21	93%	0.11	**98%**	0.11
sweet pepper	**69%**	**69%**	0.58	68%	0.16	67%	0.19	67%	0.07
table	**67%**	65%	0.30	63%	0.09	64%	0.35	**66%**	0.26
tank	78%	84%	0.14	77%	0.08	78%	0.46	**86%**	0.33
telephone	**72%**	**72%**	0.07	71%	0.05	70%	0.50	69%	0.38
television	**88%**	**88%**	0.19	87%	0.14	**88%**	0.33	87%	0.34
tiger	**80%**	78%	0.40	**79%**	0.23	77%	0.24	77%	0.13
tractor	**87%**	83%	0.20	84%	0.10	**87%**	0.35	85%	0.35

train	**86%**	82%	0.14	82%	0.11	84%	0.32	**85%**	0.43
trout	82%	81%	0.19	**83%**	0.19	79%	0.35	78%	0.26
tulip	**71%**	**69%**	0.59	66%	0.20	65%	0.13	63%	0.07
turtle	**58%**	50%	0.10	57%	0.10	52%	0.29	**58%**	0.51
wardrobe	**93%**	92%	0.57	**93%**	0.11	**93%**	0.23	**93%**	0.08
whale	70%	**71%**	0.01	70%	0.01	68%	0.23	70%	0.74
willow tree	67%	**68%**	0.20	66%	0.26	67%	0.28	**68%**	0.26
wolf	74%	**80%**	0.25	73%	0.23	76%	0.29	76%	0.23
woman	54%	50%	0.36	50%	0.09	52%	0.33	**55%**	0.22
worm	75%	73%	0.19	76%	0.16	74%	0.31	**78%**	0.34
Average	**72.80%**	72.16%	0.2618	71.78%	0.1487	71.78%	0.2917	**72.58%**	0.2978
Non-Sparse Average Weighting			0.2612		0.1797		0.2683		0.2908
Average Activations			46.59%		33.85%		63.53%		56.03%

Table 61: Comparison of weight allocation and expert performance in NarrowMoE-Imp-1-4. We measure the test accuracy with a single expert activated on each evaluation run. We highlight results in cases the expert outperforms other experts for the specific class. We further state the averagely assigned weights to each expert $\bar{g}_i$.

D.10 Expert-wise Accuracy for NarrowMoE-Imp-4-4

Table 62 states the class-wise evaluation results for each expert in NarrowMoE-Imp-4-4. We only activate one specific expert during the evaluation and assign all weights to it. Further details on the experimental setup are stated in section 5.10. The accuracy for distinct classes are highlighted if the expert performs better than the baseline on that specific class. Additionally, we also state the averagely assigned weights $\bar{g}_i$ to distinct experts per class.

Class	Acc_{MoE}	Acc_1	$\bar{g}_1$	Acc_2	Imp2	Acc_3	$\bar{g}_3$	Acc_4	$\bar{g}_4$
apple	87%	47%	0.01	35%	0.03	**89%**	0.62	77%	0.35
aquarium fish	**86%**	74%	0.31	76%	0.14	**77%**	0.53	36%	0.01
baby	**56%**	24%	0.04	26%	0.06	**56%**	0.58	47%	0.32
bear	**55%**	40%	0.21	**50%**	0.50	31%	0.21	18%	0.08
beaver	**65%**	44%	0.14	**58%**	0.60	52%	0.14	56%	0.12
bed	**76%**	70%	0.14	61%	0.04	**72%**	0.36	70%	0.45
bee	**81%**	40%	0.04	**74%**	0.32	**74%**	0.41	67%	0.23
beetle	**71%**	36%	0.05	**68%**	0.38	38%	0.15	61%	0.41
bicycle	**84%**	70%	0.10	**82%**	0.31	79%	0.17	**82%**	0.43
bottle	79%	73%	0.06	19%	0.04	75%	0.27	**80%**	0.63
bowl	**54%**	40%	0.17	37%	0.05	43%	0.39	**52%**	0.39
boy	**44%**	25%	0.06	34%	0.09	**43%**	0.48	**43%**	0.37
bridge	**80%**	**79%**	0.66	71%	0.16	50%	0.05	73%	0.13
bus	**68%**	**65%**	0.53	51%	0.09	47%	0.34	40%	0.04
butterfly	**63%**	43%	0.06	53%	0.43	**57%**	0.27	**57%**	0.23
camel	**77%**	61%	0.07	**74%**	0.26	72%	0.36	65%	0.31
can	**73%**	47%	0.09	66%	0.03	**72%**	0.37	68%	0.52
castle	85%	**86%**	0.56	82%	0.29	73%	0.10	72%	0.04
caterpillar	**60%**	45%	0.19	**57%**	0.53	44%	0.11	54%	0.17
cattle	**63%**	50%	0.17	**60%**	0.48	56%	0.23	41%	0.12
chair	87%	43%	0.04	34%	0.01	28%	0.09	**89%**	0.85
chimpanzee	**88%**	**84%**	0.30	83%	0.35	82%	0.27	83%	0.08
clock	**74%**	**66%**	0.14	63%	0.03	64%	0.27	65%	0.56
cloud	77%	**80%**	0.73	71%	0.09	71%	0.12	53%	0.06
cockroach	84%	47%	0.04	48%	0.04	44%	0.06	**85%**	0.85
couch	55%	**57%**	0.29	25%	0.06	55%	0.33	46%	0.32
crab	**73%**	43%	0.09	47%	0.29	**56%**	0.31	46%	0.30
crocodile	**59%**	52%	0.26	**55%**	0.59	33%	0.06	37%	0.09
cup	81%	62%	0.17	45%	0.02	62%	0.19	**83%**	0.62
dinosaur	**71%**	49%	0.19	42%	0.26	38%	0.13	**59%**	0.42
dolphin	**67%**	**65%**	0.86	49%	0.05	18%	0.01	31%	0.09
elephant	**74%**	67%	0.21	**69%**	0.43	62%	0.22	53%	0.14
flatfish	**68%**	44%	0.22	33%	0.13	**61%**	0.43	47%	0.23
forest	**65%**	**61%**	0.16	60%	0.75	35%	0.06	32%	0.02
fox	**77%**	42%	0.04	**77%**	0.51	64%	0.30	49%	0.16
girl	**51%**	47%	0.04	18%	0.05	**50%**	0.46	41%	0.44
hamster	**83%**	43%	0.01	65%	0.09	**79%**	0.68	72%	0.23
house	**83%**	76%	0.48	**78%**	0.28	74%	0.17	69%	0.07
kangaroo	**74%**	44%	0.07	**72%**	0.62	44%	0.15	60%	0.17
keyboard	**88%**	**83%**	0.38	70%	0.23	73%	0.12	77%	0.26

lamp	**67%**	41%	0.12	26%	0.02	44%	0.22	**62%**	0.64
lawn mower	**89%**	69%	0.11	73%	0.16	74%	0.25	**84%**	0.48
leopard	75%	60%	0.06	**76%**	0.78	50%	0.10	43%	0.06
lion	**82%**	62%	0.01	**81%**	0.47	75%	0.39	69%	0.12
lizard	**45%**	17%	0.19	**42%**	0.43	33%	0.15	37%	0.23
lobster	**62%**	32%	0.10	41%	0.31	**51%**	0.36	37%	0.23
man	**54%**	30%	0.09	25%	0.06	**54%**	0.43	44%	0.42
maple tree	**68%**	41%	0.28	**59%**	0.37	53%	0.31	42%	0.04
motorcycle	**93%**	89%	0.20	90%	0.22	**91%**	0.36	90%	0.23
mountain	85%	**87%**	0.86	49%	0.08	27%	0.05	36%	0.01
mouse	**51%**	39%	0.09	**41%**	0.35	32%	0.24	39%	0.32
mushroom	**74%**	59%	0.02	**71%**	0.63	63%	0.27	41%	0.07
oak tree	70%	**73%**	0.55	60%	0.40	55%	0.03	24%	0.02
orange	**89%**	67%	0.00	45%	0.02	**87%**	0.79	62%	0.20
orchid	**83%**	68%	0.08	69%	0.19	**71%**	0.61	61%	0.13
otter	**52%**	34%	0.38	**37%**	0.31	27%	0.14	30%	0.16
palm tree	**89%**	87%	0.44	**88%**	0.26	73%	0.05	87%	0.25
pear	**79%**	44%	0.03	59%	0.07	77%	0.43	**78%**	0.47
pickup truck	83%	**86%**	0.45	74%	0.14	81%	0.38	49%	0.03
pine tree	**70%**	60%	0.45	**61%**	0.41	41%	0.04	37%	0.10
plain	**88%**	**88%**	0.73	69%	0.15	75%	0.09	72%	0.03
plate	**74%**	54%	0.14	46%	0.07	61%	0.24	**67%**	0.56
poppy	**75%**	59%	0.02	34%	0.11	**74%**	0.80	48%	0.07
porcupine	**70%**	24%	0.08	**69%**	0.72	47%	0.15	40%	0.05
possum	52%	40%	0.11	**57%**	0.52	41%	0.24	50%	0.14
rabbit	**52%**	29%	0.13	**46%**	0.34	38%	0.27	25%	0.25
raccoon	**78%**	60%	0.12	**76%**	0.63	64%	0.12	67%	0.13
ray	**64%**	**62%**	0.56	44%	0.19	22%	0.06	41%	0.19
road	**90%**	**90%**	0.71	81%	0.14	50%	0.03	67%	0.12
rocket	**82%**	78%	0.49	38%	0.02	51%	0.10	**79%**	0.39
rose	72%	56%	0.03	27%	0.14	**73%**	0.76	58%	0.07
sea	79%	**80%**	0.79	39%	0.01	31%	0.11	44%	0.09
seal	**46%**	**43%**	0.44	36%	0.23	38%	0.17	31%	0.17
shark	**59%**	**56%**	0.80	24%	0.05	38%	0.07	23%	0.08
shrew	**60%**	**50%**	0.09	49%	0.62	35%	0.15	25%	0.15
skunk	**88%**	72%	0.28	**83%**	0.58	63%	0.04	75%	0.10
skyscraper	**91%**	**90%**	0.62	60%	0.03	59%	0.04	82%	0.30
snail	**66%**	43%	0.07	**63%**	0.45	50%	0.26	47%	0.22
snake	64%	46%	0.09	**67%**	0.48	53%	0.15	57%	0.27
spider	**76%**	54%	0.13	**73%**	0.30	51%	0.11	68%	0.45
squirrel	51%	31%	0.11	**53%**	0.58	45%	0.20	39%	0.10
streetcar	**80%**	**76%**	0.39	67%	0.20	70%	0.40	62%	0.02
sunflower	**89%**	81%	0.05	86%	0.13	**88%**	0.71	78%	0.12
sweet pepper	**75%**	45%	0.04	59%	0.11	**62%**	0.60	58%	0.24
table	**64%**	45%	0.08	52%	0.14	**62%**	0.36	57%	0.41
tank	**81%**	**78%**	0.52	64%	0.29	72%	0.10	59%	0.09
telephone	**75%**	58%	0.24	31%	0.01	55%	0.09	**73%**	0.65
television	**85%**	77%	0.37	66%	0.03	80%	0.26	**81%**	0.34
tiger	**78%**	63%	0.04	**78%**	0.64	62%	0.25	69%	0.07
tractor	83%	**85%**	0.26	**85%**	0.37	77%	0.36	49%	0.02

train	**82%**	**80%**	0.58	68%	0.18	77%	0.21	56%	0.04
trout	**78%**	69%	0.30	58%	0.25	69%	0.12	**73%**	0.33
tulip	**63%**	33%	0.02	57%	0.18	**61%**	0.71	33%	0.09
turtle	**49%**	**49%**	0.45	36%	0.21	30%	0.09	40%	0.25
wardrobe	**92%**	82%	0.02	84%	0.04	86%	0.37	**89%**	0.58
whale	**74%**	**74%**	0.79	10%	0.02	17%	0.02	34%	0.17
willow tree	**71%**	60%	0.30	**69%**	0.59	55%	0.09	67%	0.01
wolf	76%	56%	0.12	73%	0.54	63%	0.20	65%	0.14
woman	49%	24%	0.07	28%	0.02	42%	0.42	55%	0.49
worm	72%	63%	0.26	35%	0.15	27%	0.10	61%	0.49
average	**72.39%**	**57.62%**	0.2430	56.45%	0.2588	56.66%	0.2580	56.52%	0.2402
Non-Sparse Average Weighting			0.2418		0.2558		0.2579		0.2445
Average Activations			47.05%		46.58%		59.88%		46.49%

Table 62: Comparison of weight allocation and expert performance in NarrowMoE-Imp-4-4. We measure the test accuracy with a single expert activated on each evaluation run. We highlight results in cases the expert outperforms other experts for the specific class. We further state the averagely assigned weights to each expert $\bar{g}_i$.

D.11 Expert-wise Accuracy for NarrowMoE-Rel-4-4

Table 63 states the class-wise evaluation results for each expert in NarrowMoE-Rel-4-4. We only activate one specific expert during the evaluation and assign all weights to it. Further details on the experimental setup are stated in section 5.10. The accuracy for distinct classes are highlighted if the expert performs better than the baseline on that specific class. Additionally, we also state the averagely assigned weights $\bar{g}_i$ to distinct experts per class.

Class	Acc_{MoE}	Acc_1	$\bar{g}_1$	Acc_2	Imp2	Acc_3	$\bar{g}_3$	Acc_4	$\bar{g}_4$
apple	89%	88%	0.41	**90%**	0.01	86%	0.57	86%	0.02
aquarium fish	**88%**	76%	0.06	77%	0.07	**87%**	0.68	82%	0.19
baby	60%	64%	0.18	**69%**	0.03	56%	0.65	47%	0.15
bear	**54%**	47%	0.06	**54%**	0.35	53%	0.19	46%	0.40
beaver	52%	37%	0.07	53%	0.37	**58%**	0.19	50%	0.37
bed	80%	**82%**	0.72	67%	0.08	64%	0.09	68%	0.11
bee	**82%**	79%	0.05	78%	0.24	**80%**	0.53	78%	0.18
beetle	65%	61%	0.25	**66%**	0.35	51%	0.18	58%	0.21
bicycle	90%	**91%**	0.38	84%	0.07	88%	0.07	**91%**	0.49
bottle	**83%**	**81%**	0.79	80%	0.06	78%	0.11	75%	0.05
bowl	50%	47%	0.39	49%	0.04	**51%**	0.47	44%	0.10
boy	**53%**	46%	0.31	48%	0.05	49%	0.53	**52%**	0.12
bridge	**82%**	76%	0.42	76%	0.30	**82%**	0.02	**82%**	0.26
bus	69%	**70%**	0.28	59%	0.10	65%	0.21	66%	0.41
butterfly	**67%**	59%	0.17	62%	0.11	62%	0.37	**66%**	0.35
camel	**79%**	**79%**	0.21	78%	0.30	74%	0.25	72%	0.24
can	75%	**76%**	0.68	70%	0.04	67%	0.24	70%	0.04
castle	**85%**	**84%**	0.07	**84%**	0.60	81%	0.00	**84%**	0.33
caterpillar	**71%**	66%	0.12	64%	0.38	**70%**	0.32	67%	0.18
cattle	**71%**	65%	0.13	**70%**	0.44	67%	0.12	69%	0.30
chair	**91%**	**91%**	0.96	86%	0.02	85%	0.01	86%	0.01
chimpanzee	**91%**	81%	0.03	83%	0.11	84%	0.39	**88%**	0.47
clock	**74%**	**68%**	0.45	59%	0.03	66%	0.32	64%	0.20
cloud	**81%**	80%	0.11	**81%**	0.61	79%	0.13	78%	0.14
cockroach	78%	82%	0.66	78%	0.26	**85%**	0.05	75%	0.04
couch	65%	59%	0.65	64%	0.11	43%	0.10	**66%**	0.15
crab	70%	61%	0.16	63%	0.13	65%	0.40	**72%**	0.31
crocodile	**58%**	45%	0.03	51%	0.47	56%	0.07	**58%**	0.43
cup	**85%**	**83%**	0.74	78%	0.03	82%	0.16	78%	0.07
dinosaur	68%	**72%**	0.29	**72%**	0.33	59%	0.11	65%	0.27
dolphin	66%	61%	0.34	**67%**	0.38	64%	0.09	**67%**	0.19
elephant	**72%**	**72%**	0.11	**72%**	0.39	66%	0.15	69%	0.34
flatfish	65%	**68%**	0.19	48%	0.13	**68%**	0.43	61%	0.24
forest	68%	57%	0.03	**70%**	0.68	54%	0.06	60%	0.23
fox	**82%**	68%	0.08	71%	0.36	75%	0.28	**77%**	0.28
girl	52%	44%	0.21	45%	0.04	**55%**	0.55	47%	0.20
hamster	**84%**	**83%**	0.07	82%	0.08	**83%**	0.60	79%	0.25
house	**80%**	75%	0.24	77%	0.28	76%	0.05	**80%**	0.42
kangaroo	**64%**	58%	0.12	**64%**	0.47	48%	0.08	63%	0.33
keyboard	**91%**	87%	0.34	86%	0.16	84%	0.11	**89%**	0.40

lamp	59%	58%	0.73	60%	0.07	**61%**	0.15	53%	0.04
lawn mower	92%	**93%**	0.49	91%	0.13	89%	0.12	91%	0.26
leopard	**78%**	76%	0.03	**77%**	0.42	**77%**	0.09	76%	0.46
lion	**81%**	74%	0.03	**81%**	0.23	76%	0.51	77%	0.22
lizard	**56%**	45%	0.13	**50%**	0.36	49%	0.19	49%	0.32
lobster	58%	51%	0.16	54%	0.15	52%	0.44	**60%**	0.25
man	53%	**54%**	0.28	30%	0.04	51%	0.51	51%	0.18
maple tree	64%	64%	0.03	64%	0.66	59%	0.20	**65%**	0.11
motorcycle	92%	89%	0.17	91%	0.04	86%	0.21	**93%**	0.58
mountain	**90%**	76%	0.05	88%	0.40	**89%**	0.19	88%	0.36
mouse	52%	43%	0.21	47%	0.14	47%	0.23	**56%**	0.42
mushroom	**78%**	54%	0.05	72%	0.28	**73%**	0.34	**73%**	0.34
oak tree	65%	64%	0.04	65%	0.79	**73%**	0.03	63%	0.15
orange	90%	87%	0.24	**92%**	0.02	90%	0.74	85%	0.00
orchid	**87%**	75%	0.11	80%	0.03	**86%**	0.48	83%	0.38
otter	**56%**	48%	0.16	**52%**	0.32	44%	0.19	48%	0.33
palm tree	88%	85%	0.16	88%	0.41	85%	0.10	**89%**	0.32
pear	73%	73%	0.49	71%	0.05	**75%**	0.43	71%	0.03
pickup truck	**89%**	85%	0.24	85%	0.15	85%	0.17	**89%**	0.44
pine tree	71%	**72%**	0.04	69%	0.65	58%	0.05	66%	0.26
plain	**89%**	88%	0.10	**89%**	0.87	86%	0.02	**89%**	0.01
plate	68%	70%	0.38	**72%**	0.04	65%	0.39	66%	0.20
poppy	74%	73%	0.03	**75%**	0.04	**75%**	0.86	74%	0.07
porcupine	64%	48%	0.02	63%	0.41	**66%**	0.15	65%	0.42
possum	**63%**	55%	0.04	54%	0.12	56%	0.25	**60%**	0.60
rabbit	**55%**	51%	0.20	51%	0.26	**52%**	0.20	45%	0.34
raccoon	**82%**	76%	0.06	77%	0.14	73%	0.11	**80%**	0.69
ray	**63%**	61%	0.29	**63%**	0.41	55%	0.15	62%	0.14
road	**95%**	93%	0.25	**95%**	0.54	89%	0.09	93%	0.13
rocket	**84%**	**83%**	0.70	82%	0.24	81%	0.02	78%	0.03
rose	**76%**	70%	0.07	70%	0.03	74%	0.80	**76%**	0.10
sea	79%	**82%**	0.34	80%	0.57	79%	0.05	71%	0.04
seal	44%	45%	0.18	**47%**	0.30	41%	0.16	43%	0.36
shark	**59%**	**56%**	0.39	**56%**	0.22	54%	0.21	55%	0.18
shrew	**60%**	47%	0.08	53%	0.26	**57%**	0.17	54%	0.49
skunk	86%	84%	0.11	**88%**	0.29	75%	0.04	86%	0.56
skyscraper	**93%**	**91%**	0.58	**91%**	0.18	87%	0.10	90%	0.14
snail	65%	65%	0.14	**69%**	0.24	61%	0.38	67%	0.24
snake	60%	56%	0.16	58%	0.19	**62%**	0.22	61%	0.43
spider	**74%**	70%	0.30	**73%**	0.31	70%	0.18	71%	0.20
squirrel	57%	53%	0.07	**58%**	0.34	54%	0.17	54%	0.41
streetcar	74%	74%	0.18	**77%**	0.13	73%	0.22	71%	0.47
sunflower	**93%**	89%	0.04	86%	0.12	**93%**	0.83	88%	0.01
sweet pepper	**73%**	68%	0.23	62%	0.07	**73%**	0.67	65%	0.03
table	68%	61%	0.54	**77%**	0.08	62%	0.13	64%	0.25
tank	**83%**	**82%**	0.07	81%	0.36	71%	0.02	81%	0.55
telephone	**75%**	**74%**	0.75	66%	0.02	71%	0.07	**74%**	0.16
television	**83%**	**83%**	0.79	77%	0.01	81%	0.10	82%	0.10
tiger	78%	**81%**	0.02	79%	0.25	79%	0.36	78%	0.36
tractor	**92%**	82%	0.08	**90%**	0.37	87%	0.13	**90%**	0.41

train	79%	76%	0.05	**81%**	0.25	79%	0.17	**81%**	0.53
trout	80%	74%	0.30	77%	0.26	76%	0.14	**84%**	0.30
tulip	56%	50%	0.06	**57%**	0.04	55%	0.77	56%	0.12
turtle	**66%**	**60%**	0.20	**60%**	0.26	58%	0.20	58%	0.33
wardrobe	92%	**93%**	0.80	88%	0.12	89%	0.06	90%	0.02
whale	74%	**75%**	0.52	72%	0.28	65%	0.06	69%	0.14
willow tree	**65%**	60%	0.01	**64%**	0.82	63%	0.03	60%	0.14
wolf	**78%**	72%	0.02	76%	0.07	75%	0.18	**78%**	0.73
woman	55%	56%	0.33	39%	0.04	53%	0.49	**57%**	0.14
worm	**74%**	69%	0.61	**70%**	0.09	**70%**	0.16	**70%**	0.14
average	**73.30%**	69.31%	0.2547	70.25%	0.2404	69.36%	0.2483	**70.37%**	0.2565
Non-Sparse Average Weighting			0.2541		0.2427		0.2534		0.2497
Average Activations			43.20%		52.06%		46.15%		58.59%

Table 63: Comparison of weight allocation and expert performance in NarrowMoE-Rel-4-4. We measure the test accuracy with a single expert activated on each evaluation run. We highlight results in cases the expert outperforms other experts for the specific class. We further state the averagely assigned weights to each expert $\bar{g}_i$.

D.12 Expert-wise Accuracy for NarrowMoE-Imp-4-4 Complex

Table 64 states the class-wise evaluation results for each expert in NarrowMoE-Rel-4-4 with a ConvGate as gating network. We only activate one specific expert during the evaluation and assign all weights to it. Further details on the model setup are stated in section 5.11. The accuracy for distinct classes are highlighted if the expert performs better than the baseline on that specific class. Additionally, we also state the averagely assigned weights $\bar{g}_i$ to distinct experts per class.

Class	Acc_{MoE}	Acc_1	$\bar{g}_1$	Acc_2	Imp2	Acc_3	$\bar{g}_3$	Acc_4	$\bar{g}_4$
apple	**89%**	34%	0.01	79%	0.49	42%	0.03	**82%**	0.46
aquarium_fish	**84%**	71%	0.23	27%	0.03	73%	0.24	**76%**	0.50
baby	**59%**	25%	0.03	38%	0.24	24%	0.08	**57%**	0.65
bear	48%	23%	0.17	25%	0.07	**49%**	0.54	35%	0.22
beaver	**57%**	29%	0.13	36%	0.11	**54%**	0.59	30%	0.17
bed	**77%**	66%	0.16	**75%**	0.58	25%	0.03	65%	0.22
bee	**73%**	42%	0.05	55%	0.11	64%	0.38	**65%**	0.47
beetle	**71%**	23%	0.04	61%	0.32	**64%**	0.47	34%	0.17
bicycle	**88%**	81%	0.15	83%	0.51	**84%**	0.23	80%	0.12
bottle	**79%**	65%	0.10	**76%**	0.61	18%	0.03	67%	0.26
bowl	**53%**	36%	0.11	**50%**	0.46	26%	0.08	44%	0.35
boy	43%	21%	0.06	36%	0.34	41%	0.10	**45%**	0.50
bridge	81%	**82%**	0.70	57%	0.16	48%	0.11	24%	0.03
bus	**67%**	**67%**	0.54	62%	0.24	55%	0.08	41%	0.15
butterfly	**74%**	39%	0.03	51%	0.18	**66%**	0.51	49%	0.28
camel	**81%**	**74%**	0.23	71%	0.23	65%	0.26	58%	0.28
can	**78%**	66%	0.07	**74%**	0.57	52%	0.01	69%	0.34
castle	86%	**87%**	0.79	55%	0.11	64%	0.08	27%	0.02
caterpillar	57%	45%	0.14	52%	0.06	**59%**	0.60	45%	0.20
cattle	**62%**	**60%**	0.26	51%	0.14	59%	0.43	42%	0.17
chair	85%	11%	0.04	**87%**	0.90	14%	0.00	16%	0.05
chimpanzee	**90%**	75%	0.08	69%	0.11	**83%**	0.44	81%	0.37
clock	68%	53%	0.09	**69%**	0.63	50%	0.06	49%	0.23
cloud	**83%**	**83%**	0.78	40%	0.02	78%	0.08	77%	0.12
cockroach	88%	54%	0.09	**91%**	0.78	47%	0.06	50%	0.07
couch	**63%**	44%	0.21	**60%**	0.55	38%	0.04	44%	0.20
crab	**71%**	25%	0.06	45%	0.33	**60%**	0.33	47%	0.29
crocodile	**65%**	53%	0.22	27%	0.07	**61%**	0.65	26%	0.07
cup	**81%**	62%	0.17	**81%**	0.60	33%	0.02	68%	0.22
dinosaur	**69%**	47%	0.17	**62%**	0.44	48%	0.27	42%	0.12
dolphin	**68%**	**67%**	0.82	29%	0.07	45%	0.10	16%	0.01
elephant	**75%**	56%	0.26	59%	0.16	59%	0.34	**65%**	0.24
flatfish	**67%**	53%	0.27	46%	0.27	28%	0.15	**55%**	0.32
forest	**64%**	41%	0.28	18%	0.01	**57%**	0.64	30%	0.08
fox	**83%**	26%	0.05	30%	0.10	**73%**	0.55	68%	0.30
girl	**48%**	43%	0.04	32%	0.27	20%	0.08	**46%**	0.60
hamster	**79%**	49%	0.05	72%	0.18	60%	0.13	**76%**	0.65
house	**81%**	**74%**	0.61	66%	0.18	68%	0.13	66%	0.09
kangaroo	**70%**	34%	0.09	46%	0.11	**65%**	0.63	53%	0.17
keyboard	**89%**	77%	0.27	**81%**	0.38	61%	0.16	76%	0.18

lamp	62%	46%	0.18	**63%**	0.59	22%	0.02	52%	0.21
lawn_mower	**86%**	62%	0.12	**86%**	0.55	75%	0.18	61%	0.15
leopard	73%	33%	0.06	46%	0.05	**77%**	0.77	35%	0.11
lion	81%	41%	0.03	61%	0.05	**84%**	0.48	71%	0.45
lizard	**48%**	11%	0.12	29%	0.18	**41%**	0.51	30%	0.19
lobster	**66%**	24%	0.10	29%	0.24	36%	0.31	**37%**	0.35
man	**55%**	25%	0.10	43%	0.33	30%	0.11	**52%**	0.46
maple_tree	**69%**	**63%**	0.57	38%	0.04	**63%**	0.22	62%	0.17
motorcycle	**95%**	73%	0.09	**92%**	0.47	**92%**	0.20	86%	0.24
mountain	87%	**89%**	0.87	20%	0.02	29%	0.08	17%	0.03
mouse	**43%**	27%	0.06	31%	0.29	35%	0.38	**40%**	0.27
mushroom	**71%**	51%	0.05	28%	0.07	**65%**	0.60	61%	0.27
oak_tree	69%	**75%**	0.77	27%	0.05	35%	0.18	9%	0.00
orange	90%	30%	0.00	72%	0.28	32%	0.01	**91%**	0.70
orchid	**86%**	61%	0.03	57%	0.15	69%	0.24	**85%**	0.58
otter	**44%**	**31%**	0.35	22%	0.14	27%	0.37	27%	0.14
palm_tree	**92%**	**87%**	0.51	63%	0.17	74%	0.25	76%	0.06
pear	**76%**	55%	0.04	71%	0.50	44%	0.07	**75%**	0.39
pickup_truck	**88%**	**88%**	0.45	80%	0.30	66%	0.10	83%	0.16
pine_tree	**67%**	**59%**	0.65	28%	0.08	53%	0.24	31%	0.03
plain	**88%**	**88%**	0.88	64%	0.06	36%	0.04	47%	0.02
plate	**71%**	41%	0.06	**66%**	0.60	38%	0.10	56%	0.25
poppy	**76%**	62%	0.03	16%	0.07	44%	0.15	**75%**	0.74
porcupine	**67%**	26%	0.06	39%	0.04	**66%**	0.75	47%	0.16
possum	**57%**	31%	0.05	26%	0.09	**47%**	0.55	40%	0.32
rabbit	**54%**	38%	0.12	29%	0.21	**43%**	0.37	42%	0.30
raccoon	**79%**	56%	0.04	60%	0.09	**73%**	0.64	60%	0.23
ray	**62%**	**54%**	0.52	42%	0.14	44%	0.29	22%	0.05
road	**95%**	**95%**	0.73	59%	0.09	69%	0.17	58%	0.01
rocket	**86%**	**84%**	0.68	66%	0.23	47%	0.03	60%	0.07
rose	71%	47%	0.03	48%	0.09	41%	0.22	**77%**	0.66
sea	**80%**	**79%**	0.85	13%	0.08	19%	0.01	52%	0.05
seal	**46%**	41%	0.40	41%	0.14	35%	0.28	**43%**	0.18
shark	**55%**	**51%**	0.75	26%	0.07	24%	0.11	31%	0.07
shrew	**57%**	21%	0.04	20%	0.12	**54%**	0.66	20%	0.18
skunk	**91%**	48%	0.15	67%	0.14	**88%**	0.63	54%	0.08
skyscraper	**90%**	**88%**	0.75	72%	0.20	46%	0.04	56%	0.02
snail	**70%**	36%	0.06	46%	0.13	**69%**	0.49	51%	0.31
snake	62%	38%	0.05	**72%**	0.26	61%	0.51	50%	0.18
spider	**77%**	60%	0.16	60%	0.28	**74%**	0.39	65%	0.16
squirrel	55%	33%	0.11	44%	0.09	**57%**	0.58	44%	0.21
streetcar	**75%**	**71%**	0.54	56%	0.13	61%	0.12	54%	0.21
sunflower	**92%**	78%	0.13	77%	0.11	82%	0.15	**91%**	0.61
sweet_pepper	**69%**	36%	0.02	54%	0.24	43%	0.13	**60%**	0.60
table	**64%**	42%	0.14	**62%**	0.49	48%	0.09	58%	0.28
tank	**87%**	**82%**	0.48	70%	0.29	75%	0.19	58%	0.04
telephone	**77%**	35%	0.10	**77%**	0.80	27%	0.01	39%	0.09
television	**83%**	76%	0.24	**82%**	0.53	78%	0.03	67%	0.21
tiger	**82%**	61%	0.06	51%	0.07	**80%**	0.57	68%	0.29
tractor	85%	**88%**	0.40	70%	0.17	80%	0.26	74%	0.18

train	**84%**	**84%**	0.65	61%	0.13	57%	0.14	51%	0.08
trout	**80%**	66%	0.26	**69%**	0.37	63%	0.30	53%	0.07
tulip	**61%**	25%	0.03	10%	0.07	36%	0.21	**58%**	0.69
turtle	**52%**	40%	0.39	34%	0.23	**46%**	0.29	21%	0.09
wardrobe	**93%**	76%	0.04	**91%**	0.53	61%	0.01	81%	0.42
whale	**77%**	**74%**	0.75	37%	0.20	22%	0.03	4%	0.02
willow_tree	**71%**	61%	0.50	**70%**	0.03	69%	0.42	31%	0.05
wolf	**74%**	55%	0.06	65%	0.06	**74%**	0.52	53%	0.36
woman	**57%**	21%	0.07	**56%**	0.39	30%	0.04	49%	0.50
worm	**76%**	51%	0.18	**70%**	0.52	54%	0.18	37%	0.12
total	**72.70%**	53.33%	0.2532	**53.48%**	0.2487	52.90%	0.2577	52.24%	0.2405
Non-Sparse Average Weighting			0.2581		0.2502		0.2549		0.2368
Average Activations			47.64%		49.30%		47.08%		55.98%

Table 64: Comparison of weight allocation and expert performance in NarrowMoE-Imp-4-4 ConvGate. We measure the test accuracy with a single expert activated on each evaluation run. We highlight results in cases the expert outperforms other experts for the specific class. We further state the averagely assigned weights to each expert $\bar{g}_i$.

E Additional Experimental Results for RetinaNet MoE

In this chapter, we state additional results to our experiments in chapter 6. All experiments rely on RetinaNet and use embedded MoE layers. Since we use deep expert networks, we refer to the MoE layers also as MoE blocks. The results in this chapter are complements and mostly state our findings for other model architectures or training constraints. Our primary focus lies in replacing the regressor and classifier subnets with embedded MoE layers. All models are trained on the COCO dataset.

E.1 Class-wise Mean Average Precision of DetectorMoE models

Table 65 states the class-wise mean average precision AP for DetectorMoE models. Details on the experimental setup are stated in section 6.4. We measure the AP for distinct classes on the COCO test-dev2017 using the official evaluation server.

Class	Baseline	DetectorMoE-KL-4	DetectorMoE-Imp-4	DetectorMoE-Rel-4	DetectorMoE-KL-4 Single
airplane	52.4%	49.6%	49.8%	50.5%	49.4%
apple	22.8%	21.3%	21.1%	21.7%	21.3%
backpack	14.0%	13.1%	13.4%	12.7%	13.0%
ball	31.0%	29.9%	30.1%	29.7%	29.9%
banana	21.7%	20.7%	20.4%	21.2%	20.5%
bat	22.6%	20.7%	21.4%	21.6%	21.2%
bear	36.7%	34.8%	34.8%	35.2%	33.9%
bear	69.2%	67.1%	66.9%	67.3%	66.5%
bed	43.1%	41.8%	42.1%	42.0%	41.3%
bench	19.3%	18.1%	18.3%	18.2%	17.8%
bicycle	27.9%	26.6%	26.8%	26.9%	26.7%
bird	32.1%	31.6%	31.2%	31.4%	31.4%
boat	19.0%	17.4%	17.2%	17.6%	17.3%
book	8.1%	7.3%	7.3%	7.3%	7.4%
bottle	30.0%	28.2%	28.3%	28.5%	28.7%
bowl	33.4%	32.1%	31.5%	32.2%	31.8%
broccoli	25.6%	24.8%	24.9%	25.3%	24.5%
bus	62.8%	61.8%	61.8%	61.6%	61.6%
cake	26.4%	24.6%	25.0%	25.0%	24.7%
car	34.7%	33.7%	33.7%	33.6%	33.6%
carrot	16.0%	14.9%	14.7%	15.4%	14.6%
cat	58.6%	55.2%	55.8%	56.2%	55.2%
chair	22.6%	21.5%	21.6%	21.6%	21.4%
clock	46.1%	45.0%	44.9%	45.1%	45.3%
couch	37.0%	36.4%	36.2%	36.2%	36.5%
cow	46.5%	45.1%	44.8%	45.5%	44.8%
cup	34.6%	33.3%	33.5%	33.5%	33.7%
dog	54.9%	52.8%	53.1%	52.7%	53.2%
donut	47.0%	45.0%	44.9%	46.0%	45.1%
drier	0.5%	0.3%	0.3%	0.7%	0.1%

elephant	66.0%	65.1%	64.6%	65.4%	65.1%
fork	20.0%	18.1%	17.2%	18.1%	18.1%
frisbee	47.9%	46.1%	46.7%	46.2%	46.4%
giraffe	68.5%	67.5%	67.2%	67.3%	67.0%
glass	32.1%	30.6%	30.7%	30.8%	31.0%
glove	32.0%	30.6%	30.7%	30.8%	30.8%
handbag	10.7%	9.9%	9.7%	9.6%	9.9%
horse	54.3%	52.4%	52.7%	52.9%	52.3%
hot dog	23.9%	21.3%	22.6%	22.5%	22.6%
hydrant	58.7%	57.2%	56.9%	57.5%	56.8%
keyboard	41.3%	38.7%	38.8%	38.6%	39.2%
kite	35.3%	33.7%	33.4%	33.5%	33.5%
knife	9.4%	9.0%	8.5%	8.8%	8.5%
laptop	55.4%	53.3%	53.5%	53.5%	53.0%
light	21.9%	21.1%	21.1%	20.9%	20.9%
meter	35.4%	33.8%	32.8%	32.7%	32.8%
microwave	54.1%	52.2%	51.1%	51.9%	51.7%
motorcycle	37.6%	36.2%	35.8%	36.3%	36.0%
mouse	49.3%	47.2%	47.0%	46.9%	47.5%
orange	26.6%	25.2%	25.2%	25.5%	24.5%
oven	34.0%	32.3%	31.9%	31.9%	31.5%
person	47.2%	46.1%	46.1%	46.2%	46.1%
phone	25.2%	23.6%	23.5%	23.8%	23.8%
pizza	54.1%	52.5%	51.9%	52.0%	51.8%
plant	20.4%	19.5%	19.5%	19.7%	19.6%
racket	42.4%	40.4%	40.9%	40.9%	40.8%
refrigerator	43.3%	41.6%	41.9%	41.7%	42.2%
remote	22.8%	21.2%	20.7%	21.1%	21.0%
sandwich	31.8%	30.7%	30.5%	31.5%	30.7%
scissors	20.7%	18.3%	18.6%	18.9%	17.8%
sheep	43.8%	42.8%	42.4%	42.8%	42.3%
sign	65.8%	63.8%	63.5%	63.7%	63.5%
sink	31.3%	30.0%	30.4%	30.1%	29.8%
skateboard	40.0%	38.2%	37.7%	37.7%	37.3%
skis	14.0%	12.5%	12.3%	12.7%	12.4%
snowboard	21.8%	19.1%	18.9%	19.3%	19.6%
spoon	9.1%	8.1%	8.0%	8.1%	8.3%
suitcase	29.7%	28.9%	28.5%	28.8%	28.5%
surfboard	24.9%	22.8%	22.7%	22.7%	22.5%
table	28.6%	26.5%	26.3%	26.7%	26.3%
tie	25.2%	23.0%	23.8%	23.5%	23.0%
toaster	15.9%	11.8%	11.6%	11.5%	14.1%
toilet	56.3%	54.3%	54.0%	55.0%	53.8%
toothbrush	8.0%	7.7%	8.1%	7.6%	7.6%
train	59.9%	58.3%	58.7%	58.9%	58.1%
truck	31.6%	30.1%	30.2%	30.1%	29.8%
tv	51.7%	50.2%	49.9%	50.3%	49.5%
umbrella	31.1%	29.8%	30.0%	29.4%	29.6%
vase	34.2%	31.8%	31.9%	31.8%	32.3%
zebra	61.9%	60.7%	60.7%	61.5%	60.3%
AP	35.0%	33.5%	33.5%	33.7%	33.4%

Table 65: Class-wise Mean Average Precision *AP* for DetectorMoE models and a vanilla RetinaNet.

E.2 Varying Active Experts for DetectorMoE Models

We compare in section 6.4 the results for a varying number of active experts per MoE computation. Since RetinaNet makes detections on five different feature size levels, expert combinations may vary for the same image on different levels. We vary k in both gating networks, likewise. We additional state results for DetectorMoE-Imp-4 in Table 66a and for DetectorMoE-KL-4 Single in Table 66b.

Metric	k=1	k=2	k=3	k=4
AP	31.3%	33.5%	**33.6%**	33.5%
AP^{small}	14.4%	**15.5%**	**15.5%**	**15.5%**
AP^{medium}	35.1%	36.2%	**36.3%**	36.0%
AP^{large}	41.8%	44.4%	44.7%	**45.2%**
$AR^{max=100}$	46.9%	49.1%	**49.3%**	49.1%
AR^{small}	23.6%	**25.7%**	**25.7%**	**25.7%**
AR^{medium}	52.2%	54.0%	**54.5%**	54.3%
AR^{large}	63.7%	66.9%	**67.0%**	66.6%

(a) DetectorMoE-Imp-4

Metric	k=1	k=2	k=3	k=4
AP	31.7%	33.4%	**33.6%**	33.5%
AP^{small}	14.3%	15.5%	**15.6%**	15.5%
AP^{medium}	34.7%	36.1%	**36.3%**	35.9%
AP^{large}	42.6%	44.8%	45.2%	**45.3%**
$AR^{max=100}$	46.9%	49.1%	**49.3%**	49.1%
AR^{small}	23.0%	25.7%	25.9%	**26.0%**
AR^{medium}	51.9%	53.9%	**54.4%**	54.2%
AR^{large}	64.6%	66.7%	**67.0%**	66.8%

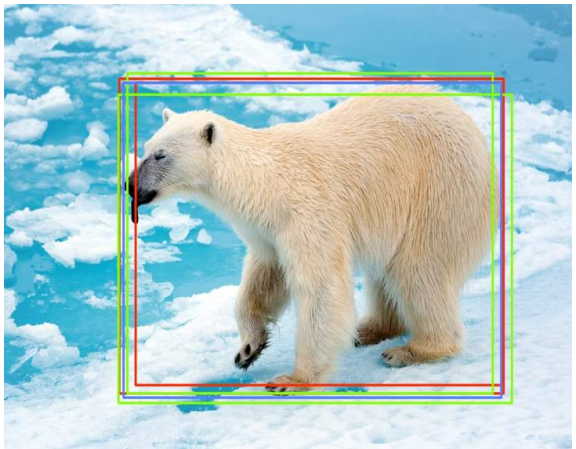
(b) DetectorMoE-KL-4 Single

Table 66: COCO metrics on the test-dev2017 dataset for a varying number k of active experts.

E.3 Visualization of MoE Behavior

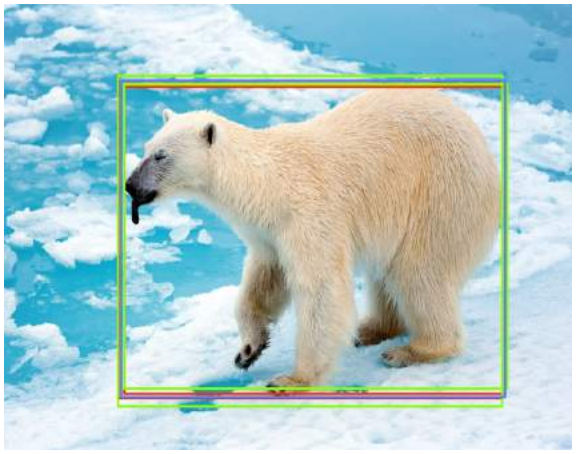
We state additional visualize BBox predictions of different MoE models and their experts in Figure 53 analog to section 6.5. BBox predictions of active experts ($k = 2$) are green, predictions of inactive experts red. We state the distinct expert predictions in the left images. The final MoE predictions are stated in the right images. Predictions are made using different RetinaNet models with MoE layers replacing the classifier and regressor subnets. We state the raw weight vectors and the sparse weight vectors for each DetectorMoE model. Raw image source: C.3

(a) Baseline



(b) DetectorMoE-KL-4
weights (0.1510, **0.3531**, **0.2857**, 0.2102)
sparse weights (0.5528, 0.4472)

(c) DetectorMoE-KL-4 final BBox

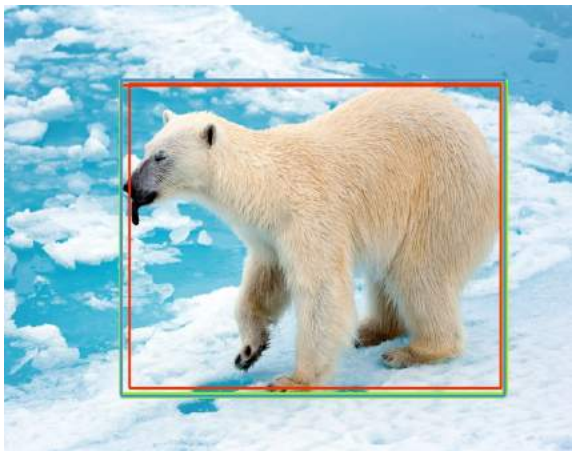


(d) DetectorMoE-Imp-4
weights (0.1850, 0.1571, **0.2404**, **0.4174**)
sparse weights (0.3655, 0.6345)

(e) DetectorMoE-Imp-4 final BBox

(f) DetectorMoE-Rel-4
weights (**0.4376**, **0.1891**, 0.1867, 0.1866)
sparse weights (0.6983, 0.3017)

(g) DetectorMoE-Rel-4 final BBox



(h) DetectorMoE-KL-4 Single
weights (**0.2929**, 0.2430, **0.3792**, 0.0849)
sparse weights (0.4358, 0.5642)

(i) DetectorMoE-KL-4 Single final BBox

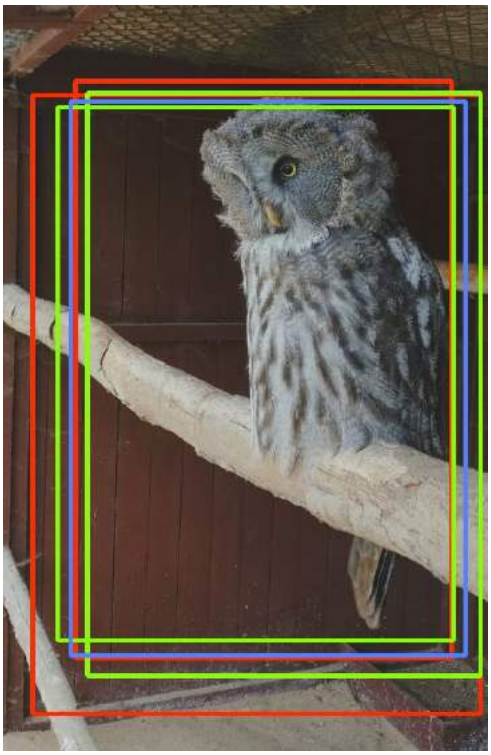
Figure 53: Comparison of BBox predictions by distinct experts in different MoE architectures.

E.4 MoE Behavior for different Object Views

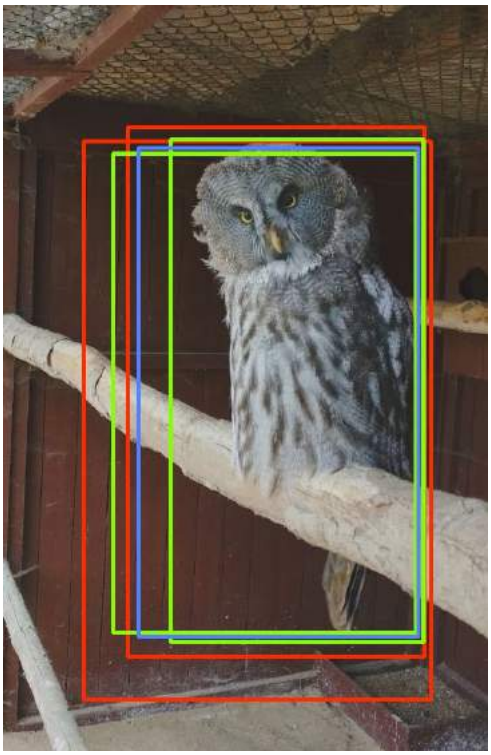
We visualize BBox predictions of different MoE models and their experts in Figure 54 analog to section 6.5. We compare the BBox predictions for different views on the same object. BBox predictions of active experts ($k = 2$) are green, predictions of inactive experts red. The final MoE predictions are drawn with blue lines. Predictions are made using different RetinaNet models with MoE layers replacing the classifier and regressor subnets. We state the raw weight vectors and the sparse weight vectors for each DetectorMoE model.

(a) Baseline

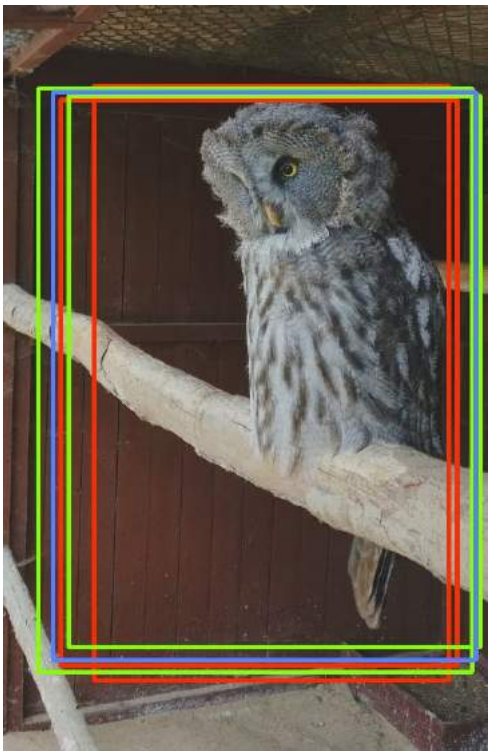
(b) Baseline



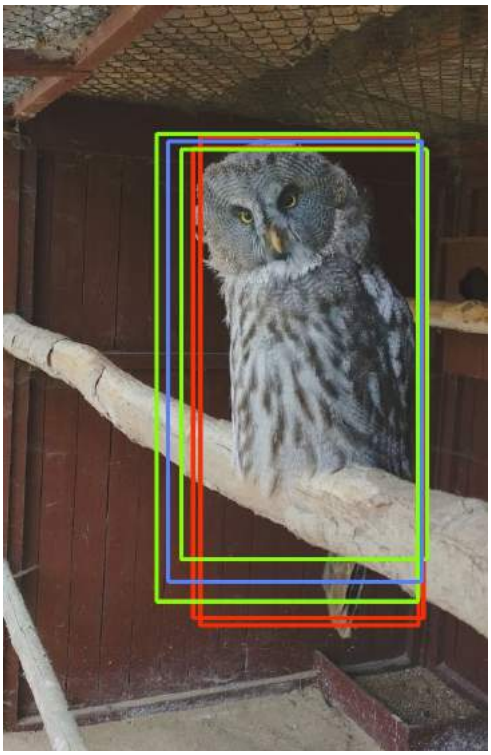
(c) DetectorMoE-KL-4
weights (**0.4368**, 0.1518, 0.1934, **0.2180**)
sparse weights (0.6671, 0.3329)



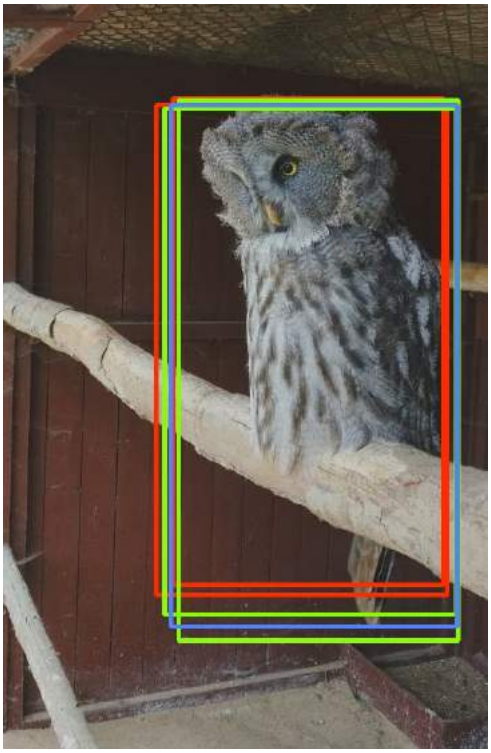
(d) DetectorMoE-KL-4
weights (**0.4383**, 0.1503, 0.1938, **0.2176**)
sparse weights (0.6683, 0.3317)



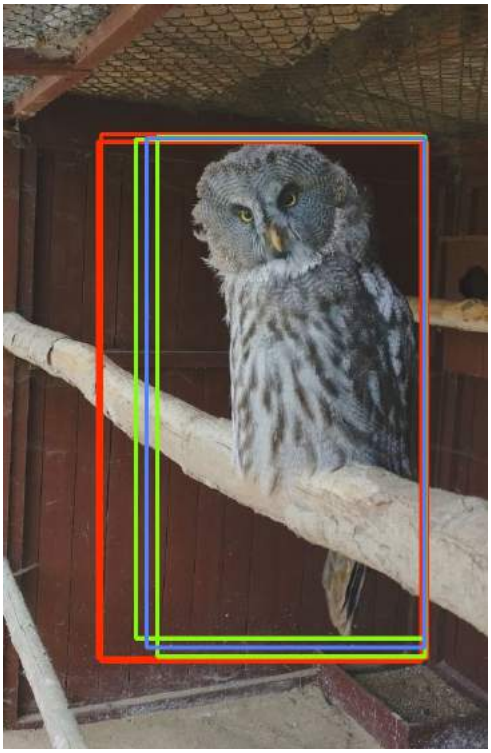
(e) DetectorMoE-Imp-4
weights (**0.2917**, **0.3326**, 0.2141, 0.1616)
sparse weights (0.4672, 0.5328)



(f) DetectorMoE-Imp-4
weights (0.1809, 0.1503, **0.2328**, **0.4360**)
sparse weights (0.3481, 0.6519)



(g) DetectorMoE-Rel-4
weights (**0.4550**, 0.1779, 0.1828, **0.1842**)
sparse weights (0.7118, 0.2882)



(h) DetectorMoE-Rel-4
weights (**0.2611**, 0.2399, **0.2648**, 0.2343)
sparse weights (0.4965, 0.5035)

Figure 54: Comparison of BBox predictions by distinct experts for different views on the same object.

E.5 Quantitative Analysis of Regressor MoE

We state additional results for section 6.6 using models not considered in the section. We compute COCO metrics for distinct experts setting $k_{reg} = 1$ and assigning all weights during evaluation to a distinct regressor expert. The results are stated in Table 67. We further compute the averagely assigned weightings per regression expert during the evaluation. We breakdown weights to different feature map levels P_3 to P_7. We state these results in Table 68.

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-Imp-4	Baseline
AP	31.9%	32.0%	**32.1%**	31.9%	33.1%	34.5%
AP^{small}	14.7%	14.6%	14.6%	**14.8%**	15.1%	16.7%
AP^{medium}	36.1%	35.7%	**36.3%**	36.0%	36.9%	38.0%
AP^{large}	44.2%	**44.6%**	44.4%	44.2%	46.0%	48.6%
$AR^{max=100}$	46.9%	47.0%	**47.1%**	46.8%	48.4%	50.1%
AR^{small}	24.7%	24.5%	24.6%	**24.9%**	25.5%	28.0%
AR^{medium}	**53.0%**	52.4%	**53.0%**	52.6%	53.9%	55.1%
AR^{large}	63.5%	64.2%	**64.3%**	63.4%	65.8%	69.1%

(a) DetectorMoE-Imp-4

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-KL-4 Single	Baseline
AP	32.3%	**32.6%**	32.2%	30.7%	33.1%	34.5%
AP^{small}	14.8%	**14.9%**	14.7%	13.9%	15.1%	16.7%
AP^{medium}	**36.5%**	36.4%	36.2%	34.2%	36.9%	38.0%
AP^{large}	44.6%	**45.2%**	44.6%	42.7%	45.9%	48.6%
$AR^{max=100}$	47.3%	**47.5%**	47.0%	45.1%	48.2%	50.1%
AR^{small}	**24.9%**	24.8%	24.7%	23.0%	25.7%	28.0%
AR^{medium}	53.3%	**53.4%**	53.0%	50.4%	53.9%	55.1%
AR^{large}	63.6%	**64.2%**	63.3%	61.2%	65.0%	69.1%

(b) DetectorMoE-KL-4 Single

Table 67: COCO metrics for DetectorMoE models with only a distinct regression expert activated during the evaluation on the validation set 2017. We keep $k_{cls} = 2$ during evaluation to observe the regression experts' behavior. We highlight the top-performing expert for each metric.

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	0.03%	0.00%	35.24%	**64.73%**
P_4	0.51%	0.01%	41.19%	**58.28%**
P_5	**50.10%**	1.60%	47.07%	1.23%
P_6	47.34%	**52.59%**	0.07%	0.00%
P_7	27.36%	**71.28%**	1.27%	0.09%

(a) DetectorMoE-Imp-4

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	43.37%	0.17%	**56.46%**	0.00%
P_4	44.31%	3.42%	**52.26%**	0.01%
P_5	38.10%	**56.70%**	3.74%	1.46%
P_6	2.88%	**60.16%**	13.90%	23.07%
P_7	0.29%	10.14%	0.62%	**88.96%**

(b) DetectorMoE-KL-4 Single

Table 68: Averagely assigned weights to each distinct regression expert during evaluation on the validation set 2017. Results are computed on the sparse vectors after deactivating all but $k = 2$ experts. We highlight top weights on each feature map level.

E.6 Quantitative Analysis of Classifier MoE

We state additional results for section 6.7 using models not considered in the section. We compute COCO metrics for distinct experts setting $k_{cls} = 1$ and assigning all weights during evaluation to a distinct classifier expert. The results are stated in Table 69. We further compute the averagely assigned weightings per classification expert during the evaluation. We breakdown weights to different feature map levels P_3 to P_7. We state these results in Table 70.

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-Imp-4	Baseline
AP	30.9%	**31.1%**	29.0%	21.1%	33.1%	34.5%
AP^{small}	**14.3%**	14.1%	12.7%	6.1%	15.1%	16.7%
AP^{medium}	34.4%	**35.4%**	34.8%	19.9%	36.9%	38.0%
AP^{large}	42.7%	**43.1%**	42.2%	31.6%	46.0%	48.6%
$AR^{max=100}$	**47.0%**	**47.0%**	46.0%	34.3%	48.4%	50.1%
AR^{small}	**25.1%**	24.0%	22.2%	15.6%	25.5%	28.0%
AR^{medium}	52.1%	**52.2%**	**0.522**	35.1%	53.9%	55.1%
AR^{large}	64.7%	**65.5%**	64.4%	48.7%	65.8%	69.1%

(a) DetectorMoE-Imp-4

Metric	Expert 1	Expert 2	Expert 3	Expert 4	DetectorMoE-KL-4 Single	Baseline
AP	31.0%	29.2%	**31.1%**	15.8%	33.1%	34.5%
AP^{small}	**14.3%**	12.4%	14.1%	4.2%	15.1%	16.7%
AP^{medium}	34.9%	34.8%	**35.4%**	13.8%	36.9%	38.0%
AP^{large}	42.9%	42.1%	**43.3%**	24.5%	45.9%	48.6%
$AR^{max=100}$	**47.5%**	45.8%	46.4%	27.6%	48.2%	50.1%
AR^{small}	**25.5%**	22.7%	23.1%	11.2%	25.7%	28.0%
AR^{medium}	**52.8%**	52.0%	52.3%	26.8%	53.9%	55.1%
AR^{large}	**64.6%**	63.8%	64.1%	39.1%	65.0%	69.1%

(b) DetectorMoE-KL-4 Single

Table 69: COCO metrics for DetectorMoE models with only a distinct single classification expert activated during the evaluation on the validation set 2017. We keep $k_{cls} = 2$ during evaluation to observe the classification experts' behavior. We highlight the top-performing expert for each metric.

Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	43.27%	**56.70%**	0.03%	0.00%
P_4	39.70%	**58.04%**	2.26%	0.01%
P_5	31.15%	9.40%	**58.17%**	1.28%
P_6	10.42%	0.00%	**58.58%**	31.00%
P_7	0.51%	0.27%	9.03%	**90.19%**

(a) DetectorMoE-Imp-4

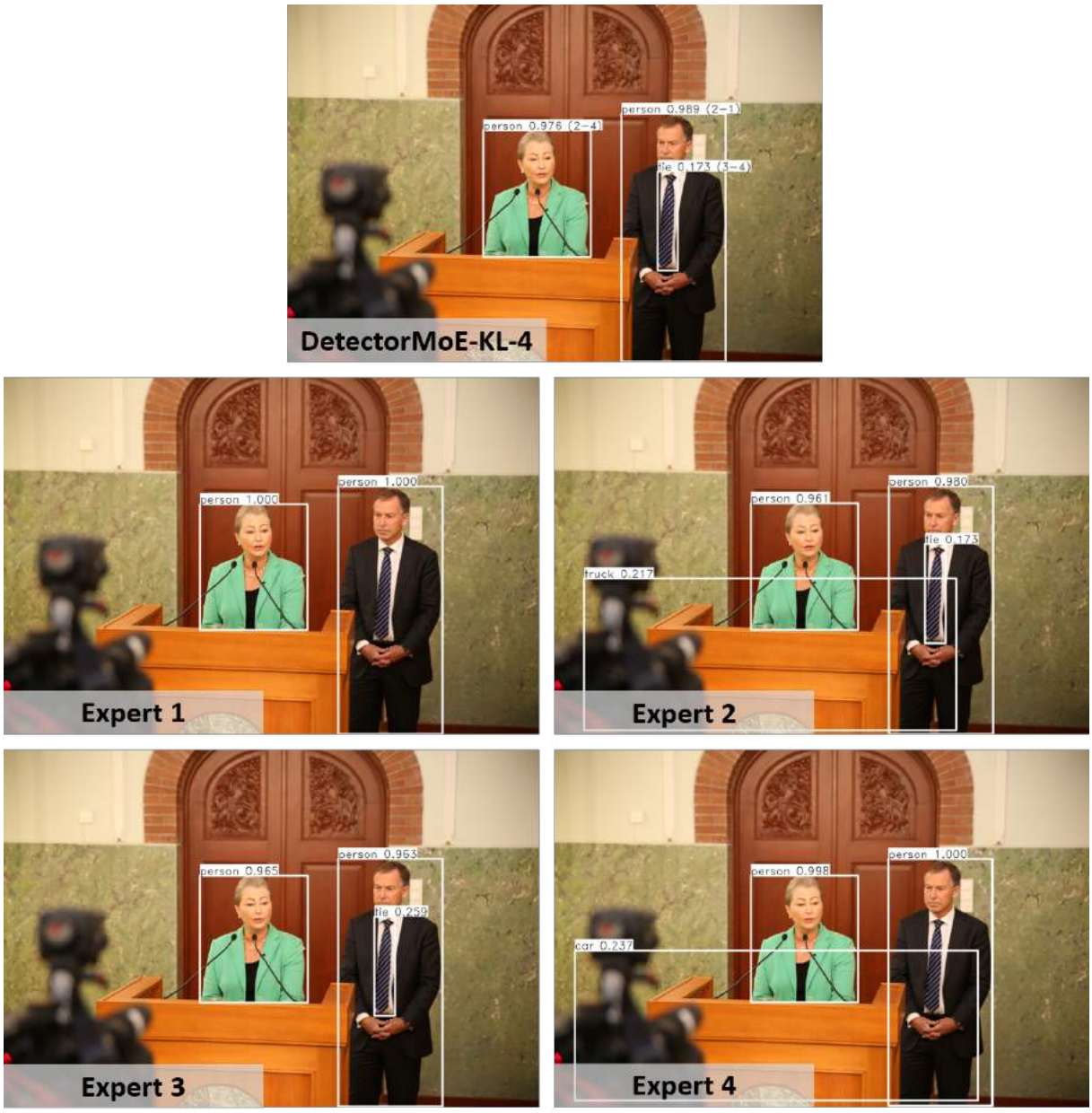
Level	Expert 1	Expert 2	Expert 3	Expert 4
P_3	43.37%	0.17%	**56.46%**	0.00%
P_4	44.31%	3.42%	**52.26%**	0.01%
P_5	38.10%	**56.70%**	3.74%	1.46%
P_6	2.88%	**60.16%**	13.90%	23.07%
P_7	0.29%	10.14%	0.62%	**88.96%**

(b) DetectorMoE-KL-4 Single

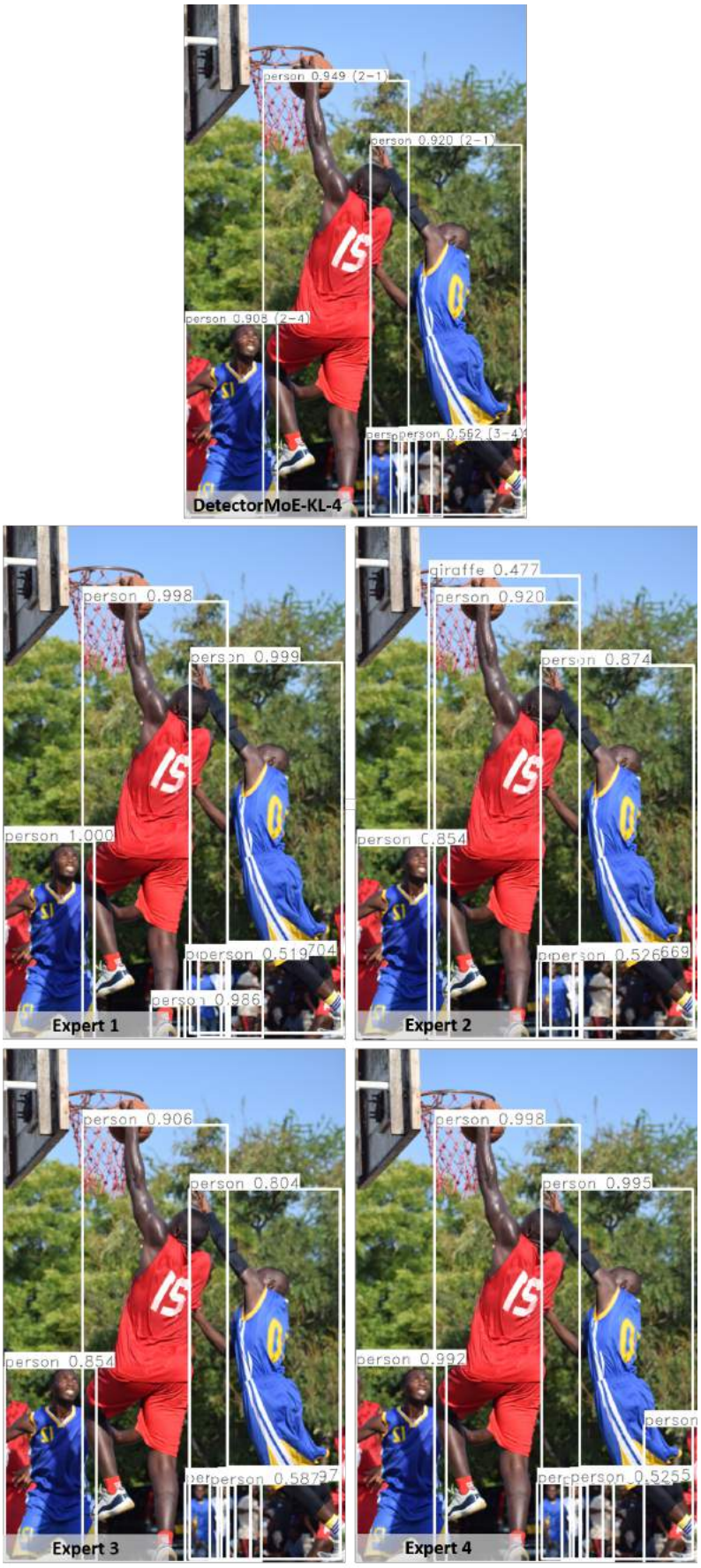
Table 70: Averagely assigned weights to each distinct classification expert during evaluation on the validation set 2017. Results are computed on the sparse vectors after deactivating all but $k = 2$ experts. We highlight top weights on each feature map level.

E.7 Distinct Classifier Expert Predictions

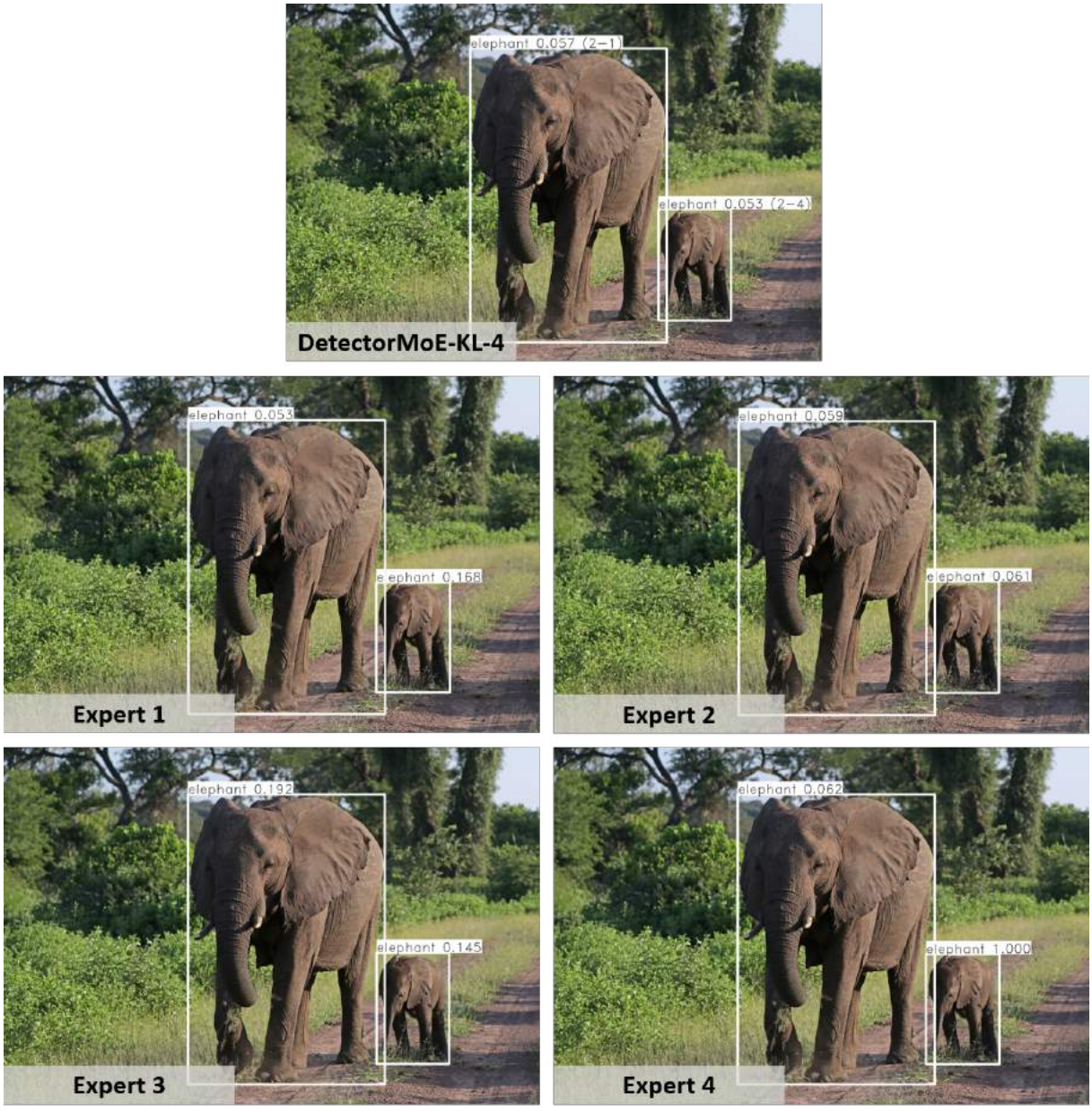
We state additional visual results to section 6.7 for distinct classification experts using DetectorMoE-KL-4. We set $k_{reg} = 1$ and plot predictions using a single classification expert. The plots in Figure 55 state the assigned class label, the predicted confidence score, and the selected experts for each predicted BBox. Distinct experts' predictions are compared to the combined MoE prediction.



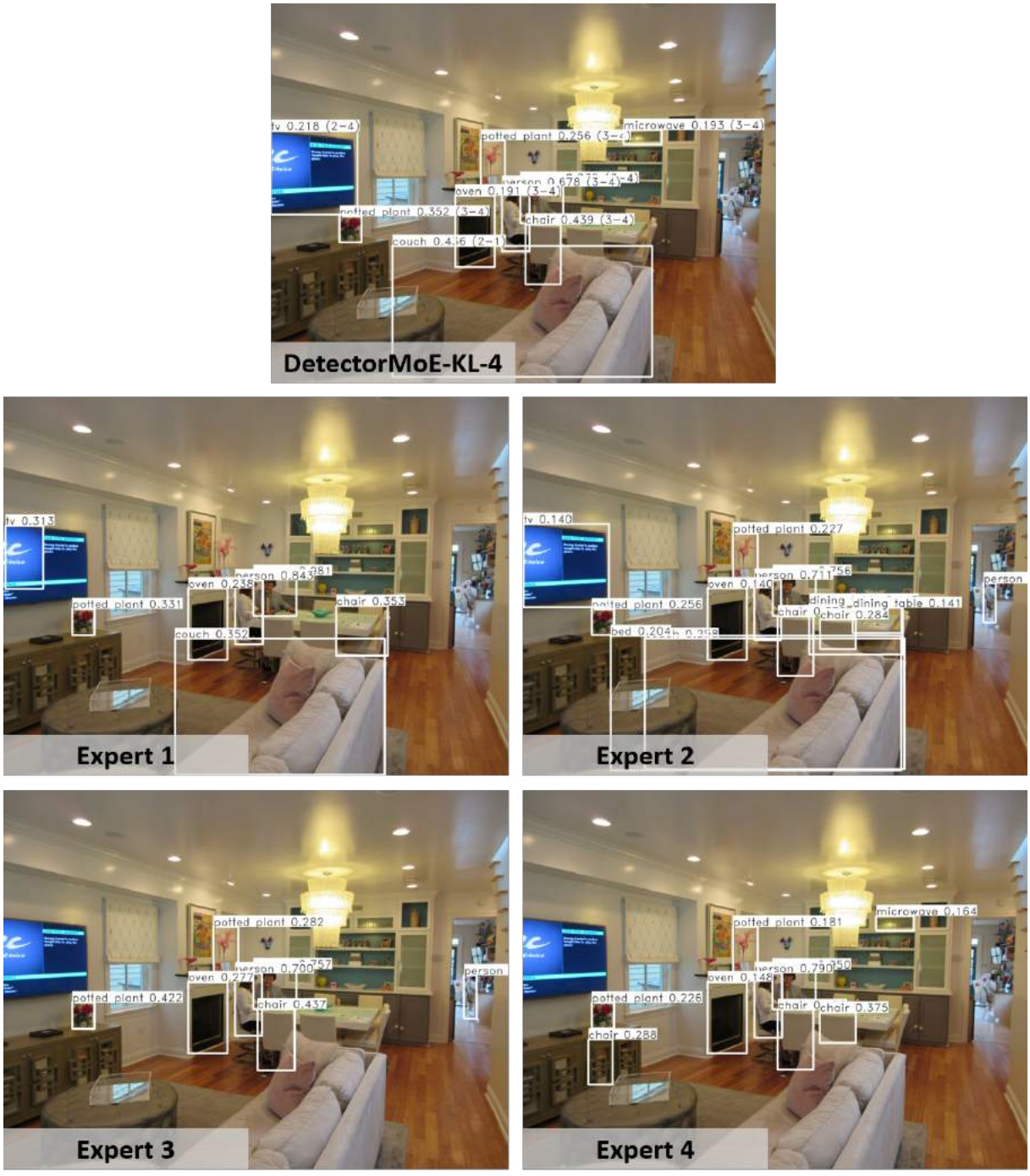
(a) Small to medium objects (P_4 - P_6)



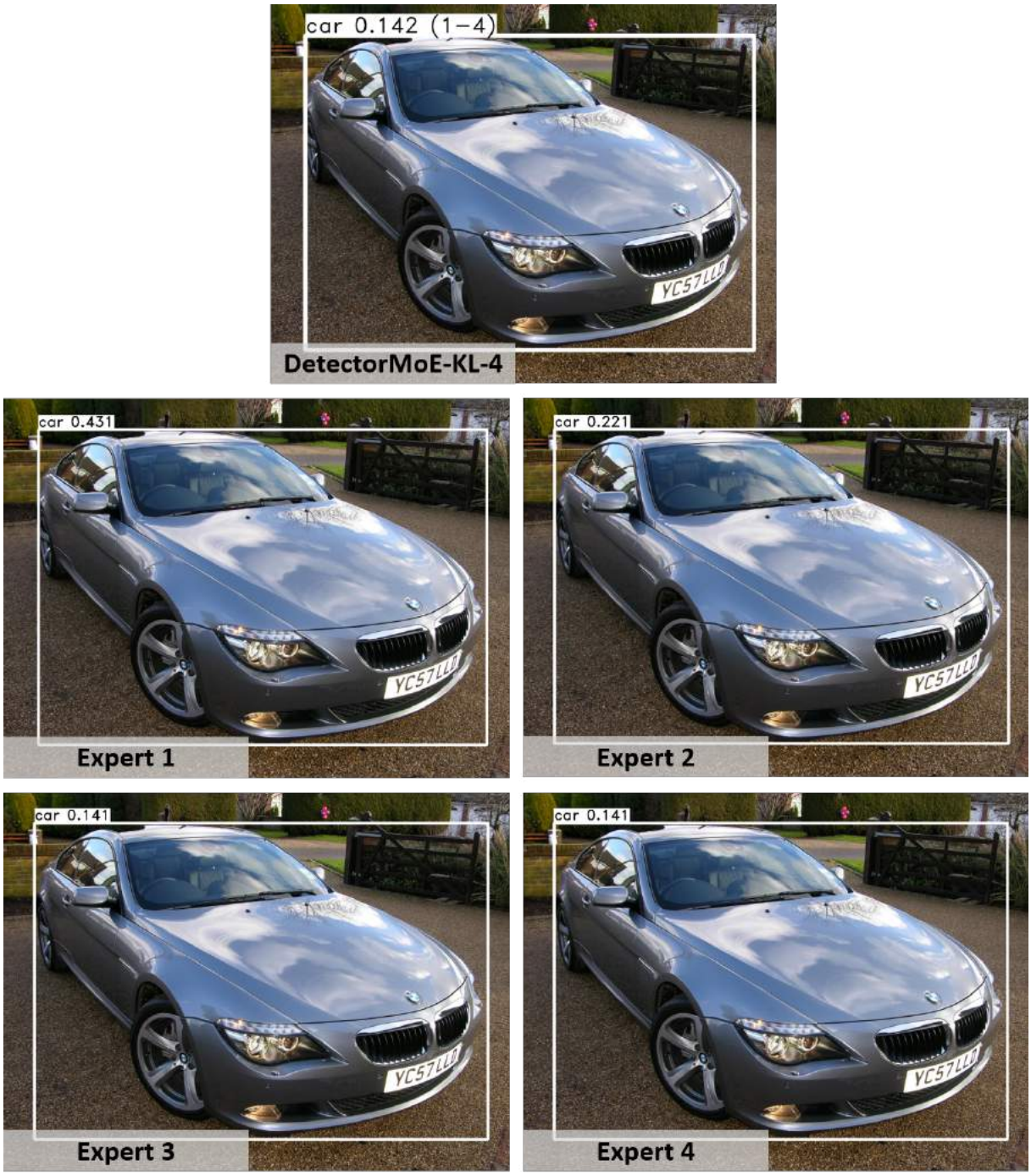
(b) Small to medium objects (P_4 - P_6)



(c) Medium objects (P_5 - P_6)



(d) Small to medium objects (P_3 - P_5)



(e) Large Object (P_7)

Figure 55: Comparison of different classifier expert predictions and the full MoE model. All detections are made by DetectorMoE-KL-4. For each BBox, we state the predicted class and the confidence score. We further state two active experts for each MoE prediction in brackets. Raw image sources: C.3

List of Figures

List of Tables

References

[1] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3, 1991.

[2] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Proceedings of 1993 International Conference on Neural Networks*, 2, 1993.

[3] S. E. Yuksel, J. N. Wilson, and P. D. Gader. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23, 2012.

[4] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[5] D. Eigen, M.A. Ranzato, and I. Sutskever. Learning factored representations in a deep mixture of experts. 2014. arXiv:1312.4314.

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

[7] K. P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Massachusetts, 2013.

[8] L. Xu, M. I. Jordan, and G. E. Hinton. *An Alternative Model for Mixtures of Experts*. MIT Press, 1995.

[9] X. Wang, F. Yu, R. Wang, Y. Ma, A. Mirhoseini, R. Darrell, and E. J. Gonzalez. Deep mixture of experts via shallow embedding. 2018. arXiv:1806.01531.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2014. arXiv:1412.6980.

[11] D. H. Wolpert. Stacked generalization. *Neural networks*, 5, 1992.

[12] W. P. Lincoln and J. Skrzypek. Synergy of clustering multiple back propagation networks. *Advances in Neural Information Processing Systems*, 2, 1990.

[13] J. Tin-Yau Kwok. Support vector mixture for classification and regression problems. *Proceedings. Fourteenth International Conference on Pattern Recognition*, 1, 1998.

[14] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, 2001.

[15] C. Yuan and C. Neubauer. Variational mixture of gaussian process experts. *Advances in Neural Information Processing Systems 21*, 2009.

[16] V. Tresp. Mixtures of gaussian processes. *Advances in Neural Information Processing Systems 13*, 2001.

[17] A. Valada, A. Dhall, and W. Burgard. Convoluted mixture of deep experts for robust semantic segmentation. 2016.

[18] O. Mees, A. Eitel, and W. Burgard. Choosing smartly: Adaptive multimodal fusion for object detection in changing environments. 2016. arXiv:1707.05733.

[19] C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of gaussian process experts. *Advances in Neural Information Processing Systems*, 14, 2002.

[20] S. Pavlitskaya, C. Hubschneider, M. Weber, R. Moritz, F. Hüger, P. Schlicht, and J. M. Zöllner. Using mixture of expert models to gain insights into semantic segmentation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020.

[21] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[22] K. Cho and Y. Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. 2014. arXiv:1406.7362.

[23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

[24] Y. Bengio. Deep learning of representations: Looking forward. 2013. arXiv:1305.0445.

[25] A. Davis and I. Arel. Low-rank approximations for conditional feedforward computation in deep neural networks, 2014. arXiv:1312.4461.

[26] E. Bengio, P. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. 2015. arXiv:1511.06297.

[27] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. 2016. arXiv:1612.02297.

[28] L. Liu and J. Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. 2017. arXiv:1701.00299.

[29] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9, 1997.

[30] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying relu and initialization: Theory and numerical examples. 2020. arXiv:1903.06733.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 2015. arXiv:1512.03385.

[32] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. 2016. arXiv:1611.05431.

[33] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. 2017. arXiv:1608.06993.

[34] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, Z. Zhang, H. Lin, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, and A. J. Smola. Resnest: Split-attention networks. 2020. arXiv:2004.08955.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. 2015. arXiv:1409.4842.

[37] D. G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision*, 2, 1999.

[38] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[39] P. Viola and Michael J. Robust real-time object detection. *International Journal of Computer Vision*, 2001.

[40] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2013. arXiv:1311.2524.

[41] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[42] R. B. Girshick. Fast R-CNN. 2015. arXiv:1504.08083.

[43] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. 2015. arXiv:1506.01497.

[44] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. 2017. arXiv:1703.06870.

[45] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. 2014. arXiv:1405.0312.

[46] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111, 2015.

[47] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. 2015. arXiv:1512.02325.

[48] J. Redmon, S. Kumar Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. 2015. arXiv:1506.02640.

[49] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. arXiv:1409.1556.

[50] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. 2016. arXiv:1612.08242.

[51] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. 2018. arXiv:1804.02767.

[52] A. Bochkovskiy, C. Wang, and H. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[53] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. 2017. arXiv:1708.02002.

[54] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. 2016. arXiv:1612.03144.

[55] A. Rosenfeld and M. Thurston. Edge and curve detection for visual scene analysis. *IEEE Transactions on Computers*, 20, 1971.

[56] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms - improving object detection with one line of code. 2017. arXiv:1704.04503.

[57] M. Weber, M. Fürst, and J. M. Zöllner. Automated focal loss for image based object detection. 2019. arXiv:1904.09048.

[58] Y. LeCun, C. Cortes, and C. J. Burges. Mnist handwritten digit database. 2, 2010. ATT Labs. available at http://yann.lecun.com/exdb/mnist, last accessed: 25.06.2020.

[59] A. Schroeder. Numbers - swiss mnist dataset. 2017. available at https://github.com/kensanata/numbers, last accessed: 27.06.2020.

[60] B. Auer and H. Rottmann. *Statistik und Ökonometrie für Wirtschaftswissenschaftler: Eine anwendungsorientierte Einführung.* Springer Fachmedien Wiesbaden, 2011.

[61] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015. arXiv:1502.01852.

[62] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012.

[63] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. arXiv:1502.03167.

[64] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24, 1933.

[65] L. van der Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2008.

[66] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2003.

[67] J. Hedderich. *Angewandte Statistik : Methodensammlung mit R.* Springer Spektrum, Berlin, Heidelberg, 2016 edition, 2016.

[68] E. Cramer and U. Kamps. *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik - Eine Einführung für Studierende der Informatik, der Ingenieur- und Wirtschaftswissenschaften.* Springer-Verlag, Berlin Heidelberg New York, 2020.

[69] 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. 2015. arXiv:1409.0575.

[70] Y. Henon. Pytorch retinanet reimplementation. 2020. available at https://github.com/kensanata/numbers under Apache License 2.0, last accessed: 26.10.2020.

[71] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. DSSD : Deconvolutional single shot detector. 2017. arXiv:1701.06659.