Supplemental Material to BMVC-#736 "Revisiting Single-gate Mixtures of Experts"

1 Datasets Detailed Description

We perform experiments on three datasets which we describe below:

- CIFAR-100 [23] (32x32 images, 50k training samples, 10k test samples, 100 classes) is a classic benchmark for small-scale image classification.
- tiny-ImageNet [26] (64x64 images, 100k training samples, 10k test samples, 200 classes) is a downscaled subset of ImageNet. It is significantly more challenging than CIFAR-100, which, combined with its reasonable scale, allows us to perform comprehensive ablation experiments in this study.
- ILSVRC2012 [41] (224x224 images, 1.28M training samples, 50k test samples, 1000 classes) is the de-facto benchmark for large-scale classification.

2 Training hyperparameters

In this section, we report the hyperparameters we used to train our baselines and run our experiments.

CIFAR-100. We follow the training pipeline of [9] to train our CIFAR-100 baselines. Each model is trained for **200** epochs with an initial learning of **0.1**; The learning rate is decayed by a factor of **5** at epoch **60**, **120** and **160**. The model is trained with SGD with a momentum of **0.9**. Finally, we use a batch size of **128**. For training the experts, we use the same hyperparameters, but train them with batch size **512** (4 times fewer training iterations), starting from pretrained weights from the base models.

tiny-ImageNet. We follow the training pipeline of [27] to train our tiny-ImageNet baselines. Each model is trained for **400** epochs with an initial learning of **0.2**; The learning rate is decayed by a factor of **10** at epoch **200** and **300**. The model is trained with SGD with a momentum of **0.9**. Finally, we use a batch size of **256**. For training the experts, we use the same hyperparameters, but only train for **100** epochs and the same batch size of 256, starting from pretrained weights from the base models.

ImageNet. We use the torchvision default pretrained models, whose training hyperparameters details can be found in torchparams. We train all our experts with the same hyperparameters except **(i)** We use cosine learning rate decay and **(ii)** we use fewer training iterations (**45** epochs with batch size **2048** for our experts, versus **90** epochs with batch size **32** in the original ResNet18, and **600** epochs with batch size **128** in the original MobileNetv3-small).

3 Qualitative Analysis of Experts specialization

In this section, we report on a few qualitative results to highlight how the experts specialize during training. **First**, we perform a simple analysis on trained experts: For each sample, we record which expert reaches minimal cross-entropy loss on that given sample. We then display the resulting class distribution across experts of the whole training set. We report the results in Figure 4: We observe that the experts do end up specializing to specific subsets of the data, although, in contrast to hierarchical classification, many classes are still clearly split across several experts. Furthermore, most of these specialization patterns are consistent across the number of experts. Finally, while some experts clearly account for more classes than others, no expert is ever fully inactive.

Second, we visualize the initial sample-to-expert assignment from the gate g_0, following the K-means clustering step on the base model. We report some of these visualizations on ImageNet in Figure 5. The initial gate does find meaningful groupings in the dataset, following the base model's pretrained embeddings; Furthermore, the mapping does not exactly respect the dataset's labels as most classes are distributed across more than one expert. Finally, we observe that there are different levels of "density" across the clusters: E.g. the 9-th expert is very self-contained and contains almost entirely all dog breeds. In contrast the 6-th expert contains many more varied classes, and not always fully, mostly composed of common household objects.

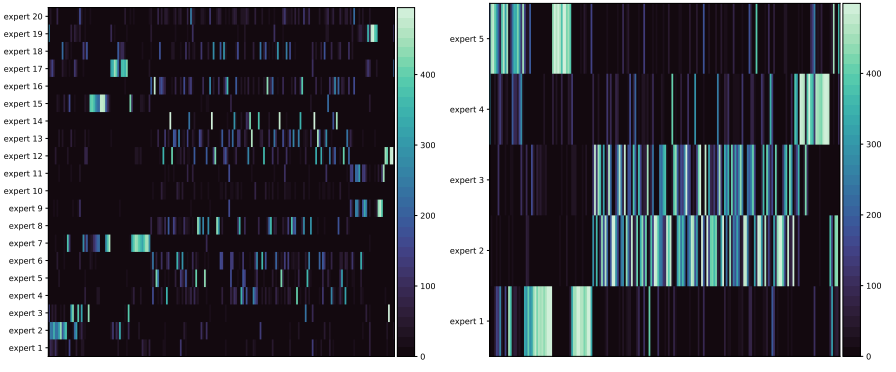
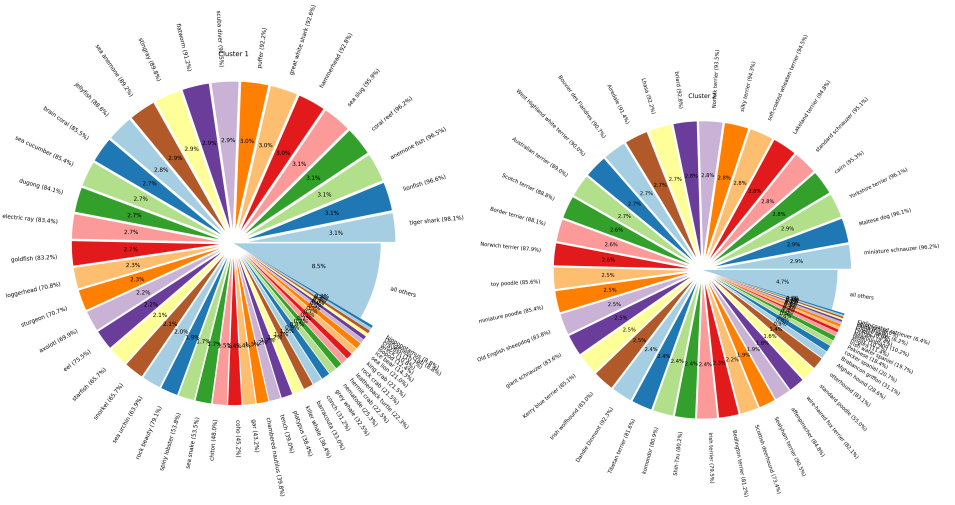


Figure 4: Specialization pattern of the trained experts for 20 experts (*left*) and 5 experts (*right*) on the tiny-ImageNet dataset. The x-axis represents the 200 classes, and the y-axis represents the experts. The cells' colors represent the number of training samples of said class and for which said expert yields the lowest cross-entropy loss

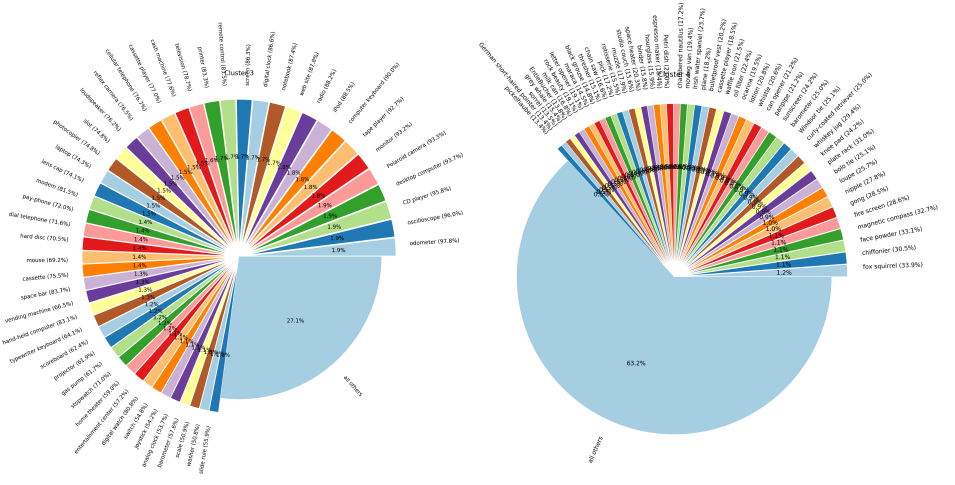
4 Analysis of the EM Training Scheme

As described using the EM algorithm allows us to jointly train the gate alongside the experts, while preserving some training stability. In particular, a determining factor is N_E, the number of E steps update. The higher this number is, the more often the gate is updated based on experts feedback. Our asynchronous training pipeline (Algorithm ') can be seen as a special case $N_E = 0$.

Given a fixed number of total epochs, n_e we vary the number of EM iterations, N_E. For our proposed training pipelines ($N_E = 0$) this means that the experts are trained for n_e epochs



(a) Example of two self-contained / highly specialized experts



(b) Example of complex experts with much higher class variation

Figure 5: Visualization of some sample-to-expert mappings from the precomputed initial gate g_0 on ImageNet for 20 experts and the ResNet18 base model. Each pie chart represents an expert; For readability, only the slices of the top-k classes are displayed individually. The percentage *inside* a slice is the percentage of this class among the samples mapped to this expert. Conversely, the percentage next to the class label is the percentage among all samples of the class (across all experts): A high percentage means that the class is almost entirely contained in one cluster. Best seen zoomed in.

with the fixed initial gate g_0. For $N_E > 0$, the experts are also trained for n_e epochs, but every $\frac{n_e}{N_E+1}$ epoch, the gate weights are updated based on the current experts' performance using the E step formula in **Equation 2**.

We report quantitative results on tiny-ImageNet in Figure 6 (left), and observe that joint training does result in more accurate models. However, this makes training cumbersome as it requires synchronization across experts every time the E step is computed, i.e., a total of N_E times.

For the case of $N_E = 40$, we further analyse the difference between the initial gate g_0 and the learned gate g at the end of training, as shown in Figure 6 : The two gates differ on roughly 14% of the training dataset, and the assignment differences are not random but follow a specific pattern across classes.

N_E	top-1-acc (w/o ensembling)	top-1 acc (w/ ensembling)
0 **(ours)**	63.11	65.72
10	63.46	65.74
20	63.58	65.84
40	**64.48**	**66.15**

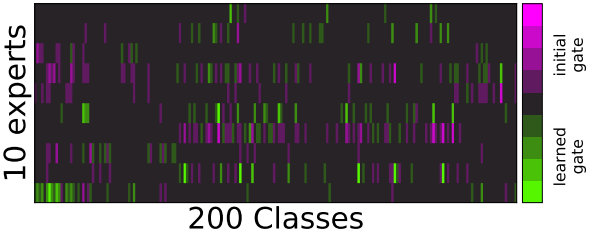


Figure 6: (*left*) Impact of increasing the number of E steps, N_E, when jointly training the gate and experts, using the setup of Table **??** (tiny-ImageNet, 10 experts). The total training budget is the same for all runs, only the number of updates differ. (*right*) We plot the heatmap of the difference between the class-to-expert assignments of the **initial gate** g_0 and the **final trained gate** for $N_E = 40$.

We also detail the five most consistent differences in assignment between these two gates below: The samples assigned to a different expert by the learned gate g are semantically meaningful: g often re-assigns the samples to an expert that already contained a large proportion of its ground-truth class.

✓ **40** samples of the class `steel arch bridge` and **105** samples of the class `triumphal arch` originally mapped to expert **4** by g_0 are mapped to expert **5** by the learned gate instead.

✓ **96** samples of the class `espresso` originally mapped to expert **6** by g_0 are mapped to expert **7** by the learned gate instead.

✓ **81** samples of the class `European Fire Salamander` originally mapped to expert **8** by g_0 are mapped to expert **1** by the learned gate instead.

x **59** samples of the class `bikini` originally mapped to expert **2** by g_0 are mapped to expert **9** by the learned gate instead.

For reference, and to better understand each expert's specialty, the most often assigned expert to each class by the original g_0 gate in this experiment was as follows:

- **Expert 1:** European fire salamander, bullfrog, tailed frog, American alligator, boa constrictor, trilobite, scorpion, tarantula, centipede, brain coral, slug, sea slug, spiny lobster, dugong, cockroach, sea cucumber, snorkel, coral reef
- **Expert 2:** academic gown, apron, bikini, bow tie, cardigan, Christmas stocking, fur coat, kimono, miniskirt, neck brace, plunger, poncho, potter's wheel, punching bag, sock, sombrero, sunglasses, swimming trunks, teddy, vestment, ice lolly
- **Expert 3:** goose, koala, Chihuahua, Yorkshire terrier, golden retriever, Labrador retriever, German shepherd, standard poodle, tabby, Persian cat, Egyptian cat, cougar, lion, brown bear, guinea pig, hog, ox, bison, bighorn, gazelle, Arabian camel, orangutan, chimpanzee, baboon, African elephant, lesser panda

- **Expert 4:** abacus, altar, bannister, barbershop, brass, cash machine, chest, computer keyboard, confectionery, desk, dining table, freight car, organ, parking meter, payphone, refrigerator, rocking chair, scoreboard, sewing machine, space heater, turnstile, comic book

- **Expert 5:** black stork, albatross, barn, beacon, birdhouse, cannon, cliff dwelling, crane, dam, flagpole, fountain, gondola, lifeboat, obelisk, picket fence, pole, projectile, steel arch bridge, suspension bridge, thatch, triumphal arch, viaduct, water tower, alp, cliff, lakeside, seashore

- **Expert 6:** American lobster, frying pan, wok, plate, guacamole, ice cream, pretzel, mashed potato, cauliflower, bell pepper, orange, lemon, banana, pomegranate, meat loaf, pizza, potpie

- **Expert 7:** goldfish, jellyfish, king penguin, backpack, barrel, bathtub, beaker, beer bottle, binoculars, broom, bucket, candle, CD player, chain, drumstick, dumbbell, gasmask, hourglass, iPod, lampshade, magnetic compass, nail, oboe, pill bottle, pop bottle, reel, remote control, sandal, stopwatch, syringe, teapot, water jug, wooden spoon, espresso

- **Expert 8:** black widow, snail, ladybug, fly, bee, grasshopper, walking stick, mantis, dragonfly, monarch, sulphur butterfly, spider web, mushroom, acorn

- **Expert 9:** basketball, butcher shop, jinrikisha, lawn mower, maypole, military uniform, rugby ball, torch, umbrella, volleyball

- **Expert 10:** beach wagon, bullet train, convertible, go-kart, limousine, moving van, police van, school bus, sports car, tractor, trolleybus

5 Derivation of the EM Algorithm

We denote by X and Y the random variables associated to the input data (images and associated class labels respectively). We denote by Z the hidden variable associated to the index of the expert that image X is most likely associated to.

We are interested in the two following probabilty densities:

- $p(Z|X)$: The gate network routing a sample to an expert. This corresponds to the gate, $g(k|x)$, in our model.

- $p(Y|X,Z=k)$: The probability distribution over class labels output by the k-th expert. This corresponds to the expert, $e_k(y|x)$, in our model.

We are interested in maximizing the total likelihood of the model, $p(y|x)$ using Expectation-Maximization (**EM**). We first derive the Evidence lower bound (ELBO) on $p(y|x)$ using the standard variational Bayesian framework; Introducing the variational distribution $q(Z|X,Y)$, we have:

$$KL(q(z|x,y)\|p(z|x,y)) = \mathbb{E}_q \log(q(z|x,y)) - \mathbb{E}_q \log p(z|x,y) \tag{4}$$

$$= \mathbb{E}_q \log(q(z|x,y)) - \mathbb{E}_q \underbrace{\log p(z,y|x)}_{\log p(z|x)+\log p(y|z,x)} + \underbrace{\mathbb{E}_q \log p(y|x)}_{=\log p(y|x)} \tag{5}$$

$$= KL(q(z|x,y)\|p(z|x)) - \mathbb{E}_q \log p(y|x,z) + \log p(y|x) \tag{6}$$

Reordering the terms around, we have:

$$\log p(y|x) = \underbrace{\mathbb{E}_q \log p(y|x,z) - KL(q(z|x,y)\|p(z|x))}_{ELBO} + \underbrace{KL(q(z|x,y)\|p(z|x,y))}_{\geq 0} \tag{7}$$

The underlying idea of EM is a two step process in which we (i) minimize the difference between $\log p(y|x)$ and the ELBO (i.e., minimize the right-hand term in (11)) and (ii) maximize the ELBO (left-hand terms in (11)), usually until convergence. We then reiterate these two steps until satisfied.

E step: Minimize the difference between ELBO and the likelihood
For this we only have to compute the posterior distribution, i.e., set q to:

$$q(z|x,y) \leftarrow p(z|x,y) = \frac{p(z|x)p(y|z,x)}{p(y|x)} = \frac{p(z|x)p(y|z,x)}{\sum_z' p(z'|x)p(y|z',x)} \tag{8}$$

$$q(z|x,y) \leftarrow \frac{g(z|x)e_z(y|x)}{\sum_z' g(z'|x)e_{z'}(y|x)} \tag{9}$$

M - step: Maximize ELBO with the fixed q

$$ELBO = \mathbb{E}_{z\sim q(z|x,y)} \log p(y|x,z) - KL(q(z|x,y)\|p(z|x)) \tag{10}$$

Using our model's notations (and a sum over the experts rather than an expectation), this means we have to maximize:

$$\sum_z [q(z|x,y) \log e_z(y|x) - KL(q(z|x,y)\|g(z|x))] \tag{11}$$

Thus we exactly recover **Equations 2** and **3** from the main text

Equivalence to Backpropagation. The separation of the E and M step is crucial for training stability: In the M step, the updates for the parameters of the gating function and the experts become decoupled entirely, hence can be trained independently. In fact, if we perform the E and M steps on the same batch of data, then we have by definition that $q(z|x,y) = p(z|x,y)$ on the current batch. Therefore, the right-hand KL divergence term in **Equation 11** is equal to 0, and the ELBO is equal to the total log-likelihood. Therefore, the EM algorithm simply becomes equivalent to directly minimizing the total log-likelihood $\log p(y|x)$ via backpropagation.

In other words, performing the E and M steps simultaneously is equivalent to directly training the gate and experts jointly with standard backpropagation. However, we observed many training instabilities using direct backpropagation (e.g. gate collapse), which also

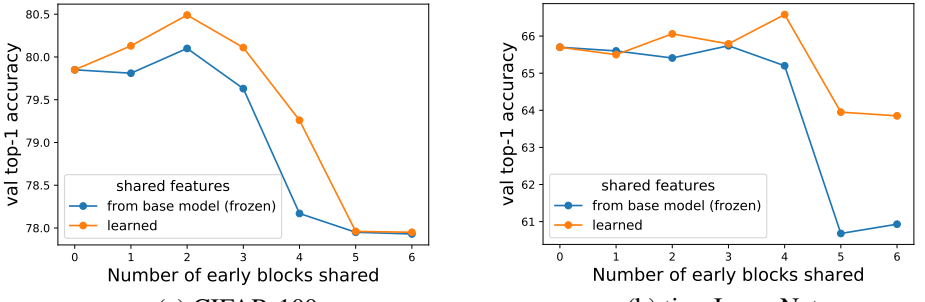
seems to be the case in the literature as most works introduce ad-hoc losses (e.g. balancing loss term) when implementing joint training via backpropagation.

Intuitively, this means that EM essentially implements a delay in the parameter updates: We estimate the probability of the data-points belonging to each expert in the E step, and then we update the gate and expert functions following that estimate for a few epochs. The algorithm thus incurs one extra hyperparameter on top of standard backpropagation; That is, how many times we go through the E step to update the expert assignments estimates (we denote this hyperparamter as N_E in the main text).

6 Ablation: Expert Entry Layer

In the main paper, we always set the number of early layers shared by the experts to be about half the architecture (i.e. 3 layers for ResNet-based models, and 8 for MobileNetv3-small).

Performing a full ablation study on this parameter shows that it could be better optimized for further accuracy gain: For instance in Figure 7, we observe that sharing 2 layers instead of 3 yields higher final accuracies on CIFAR-100. In both CIFAR-100 and tiny-ImageNet, we also see that there is a clear drop when the experts are reduced to only being linear layers. In general, we observe that the early features from the tiny-ImageNet base model are useful to the expert until a higher depth than the CIFAR ones. Automatically tuning this parameter for a given dataset and architecture is a research direction we consider for future work.



(a) CIFAR-100 (b) tiny-ImageNet

Figure 7: Validation accuracy when varying the number of early layers shared by the experts. All experiments were performed using 10 experts in the `tr18-tr18` configuration on CIFAR-100 (*left*) and tiny-ImageNet (*right*).

7 Ablation: γ Hyperparameter

In this section, we analyze the effect of the γ hyperparameter that we use to smooth the gate weights during training the experts, as described in **Section 2.2**: γ can be interpreted as a non-zero weight given to all "negative samples" (those which the gate does not map to the current expert) while training an expert. In Figure 8, we report accuracy results when varying γ. We observe that too low values of γ are detrimental to accuracy, both with and without ensemblers. However the model seems to be more robust to higher values of γ. In practice, we use a value $\gamma = 0.05$ for all our main experiments.

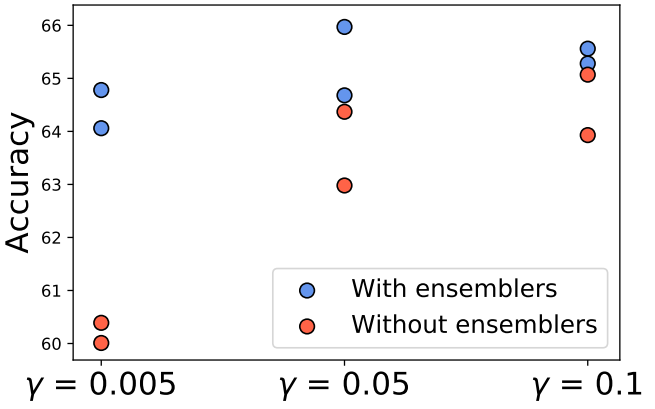


Figure 8: Effect of varying the γ hyper-parameter (lower bound of the gate clipping operation Γ when training the experts). Results are displayed for two random seeds, and obtained on tiny-ImageNet, for a `tr18-tr18` model with 10 experts.

8 Ablation: Early-exiting based on the gate or base model

Thresholding based rule. Our framework allows for two approaches to threshold-based early-exiting: (i) thresholding the gate's confidence; and (ii) thresholding the base model's confidence. Thresholding the gate's confidence is based on the intuition that samples which are not routed confidently are also more likely to be classified wrong. On the other hand, base model thresholding directly builds on the intuition that the base model predicts correct samples with high confidence while samples with lower confidence are more likely to be classified wrong. Based on the reliability diagrams in Figure 9, we can see that, while not perfectly calibrated, there is a strong correlation between confidence and accuracy for the base model, while the same is not the case for the gate module. Therefore, we use the base model's confidence rather than the gate's as our guide for early-exiting in the main text and all experiments.

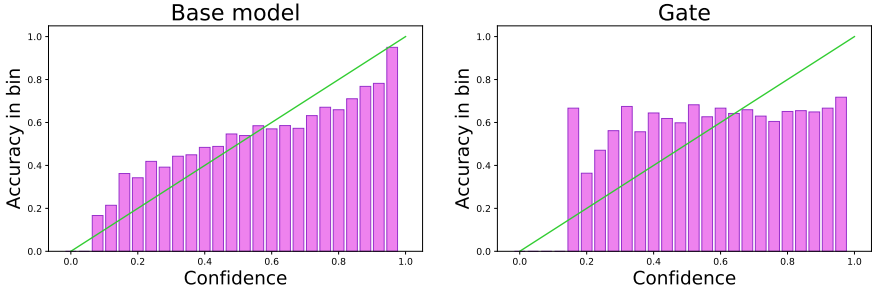
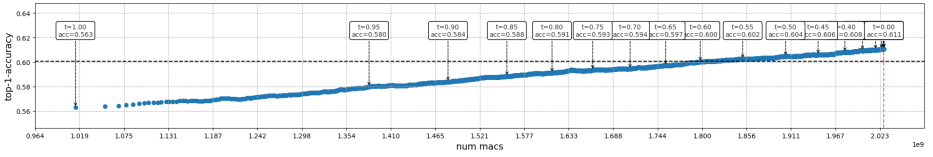
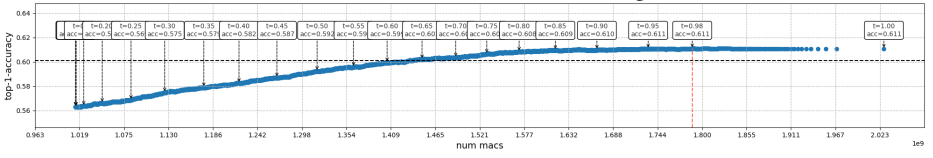


Figure 9: Reliability pattern for `tr10` on tiny-ImageNet (20 `tr`-10 experts). The bins on the x-axis correspond to the confidence of the base model (*left*) and the gate (*right*). The y-axis corresponds to the accuracy on the samples in each bin.

To confirm this observations further, we also plot early-exiting results for both gate's confidence thresholding and base model's confidence thresholding for one of our models in Figure 10. We observe that base model thresholding indeed yields higher accuracy vs early-exiting ratio trade-offs.



(a) Gate's confidence thresholding



(b) Base model's confidence thresholding

Figure 10: Comparison of early-exiting by thresholding either the gate's confidence or the base model's confidence for `tr10-base` on tiny-ImageNet (20 `tr-10` experts). The x-axis corresponds to number of MACs at different threshold values. The y-axis corresponds to accuracy of model with early-exiting. Dashed *black* line corresponds to 1% accuracy drop. Dashed *red* line shows result for the best performing early-exiting threshold. Boxes above the blue curve show the different threshold values along with accuracy for each threshold.

Selecting the best threshold. We further report results for the best performing threshold in each setting in Table 7. We present results for `tr10-base` on tiny-ImageNet (20 `tr-34` experts) and `tr34-base` on tiny-ImageNet (20 `tr-10` experts). Looking at the whole range of thresholds allows us to find a model that fits the required computational budget. And selecting the best threshold by looking at the validation set can be seen as the upper bound. However, ideally, setting the optimal threshold properly and fairly would require an external validation holdout set. As an alternative, we use a subset of the training data to investigate whether we can set a threshold that generalizes well to inference time using training data only. We report the corresponding results in Table 7.

method	base - expert - num experts	baseline acc, % (no early-exiting)	early-exit ratio, %	top-1 acc, %	MACs (x 1e9)	#params (x 1e7)
validation-selection	tr10-tr34-20	66.22	20.11	66.27	4.7	2.2
	tr34-tr10-20	66.36	29.07	66.36	5.3	2.5
subset-selection	tr10-tr34-20	66.22	37.40	65.87	3.9	1.8
	tr34-tr10-20	66.36	75.07	65.18	4.9	2.3
gate-learning	tr10-tr34-20	66.22	10.64	65.54	5.1	2.4
	tr34-tr10-20	66.36	14.63	65.61	5.5	2.5

Table 7: Best performing thresholds for different approaches to early exiting as discussed in Appendix 8. `Validation-selection` stands for selecting optimal threshold on a validation set. `Subset-selection` stands for selection threshold on a subset of training data and using that for validation set. `Gate learning` stands for learning a separate gate for early exiting.

Learning the early-exiting gate. Alternatively, we can try to *train* the early-exiting behavior for a given early-exiting budget. This approach requires slightly changing model the architecture by adding another output to the gate g. We rewrite the early-exiting criterion to include this new output:

$$ee(x) = \mathbb{1}(arg\max_k g(k|x) = K+1) \tag{12}$$

where $\mathbb{1}$ is the indicator function, K is the number of experts, and the $K+1$ index corresponds to the newly added early-exiting output. The drawback of such approach is that it requires to train additional parameters. However, since the rest of the network is kept fixed, training only requires a very small number of epochs and does not increase the complexity of the entire approach much.

Similar to previous works [6, 42], we use a supervised approach to train the new gate. However, unlike previous papers which use auxiliary classifiers to label each sample as being fit for early-exiting or not, we use a simple approach to label the data from the solution of an integer linear programming (**ILP**) problem: Specifically, we first fix an efficiency budget $\tau \in [0,1]$, which is our target early exiting ratio on the training set. We then solve the optimal early exit assignment for budget τ on the training set by solving the following discrete optimization problem:

$$\text{maximize} \sum_{(x,y)} \left[ee(x)\ \phi(y|x) + (1-ee(x))\ \sum_{k=1}^{K} g(k|x)\ e'_k(y|x) \right]$$

subject to:

$$\frac{1}{N}\sum_x ee(x) = \tau$$
$$\forall x,\ ee(x) \in \{0,1\}$$

where N is the total number of samples, the assignment $ee(x) \in \{0,1\}$ is 1 when sample x should early exit, and 0 otherwise. This is a simple ILP problem with binary variables, that directly solves the optimal assignment on the training set. Solving this problem yields binary labels $ee(x)$ which we then use to learn the early-exiting gate. We report accuracy results of the learned gate in Table 7. Overall, this approach seems to perform worse than selecting a simple threshold based on the training set, as the learned early exiting behavior does not generalize well at inference time.