MIXTUREENSEMBLES: LEVERAGING PARAMETER SHARING FOR EFFICIENT ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Ensembles are a very effective way of increasing both the robustness and accuracy of a learning system. Yet they are memory and compute intensive; in a naive ensemble, n networks are trained independently and n networks must be stored. Recently, BatchEnsemble (Wen et al., 2020b) and MIMO (Havasi et al., 2020) has significantly decreased the memory footprint with classification performance that approaches that of a naive ensemble. We improve on these methods with MixtureEnsembles, which learns to factorize ensemble members with shared parameters by constructing each layer with a linear combination of templates. Then, each ensemble member is defined as a different set of linear combination weights. By modulating the number of templates available, MixtureEnsembles are uniquely flexible and allow easy scaling between the low-parameter and high-parameter regime. In the low parameter regime, MixtureEnsembles outperforms BatchEnsemble on both ImageNet and CIFAR, and are competitive with MIMO. In the high-parameter regime, MixtureEnsembles outperform all baselines on CIFAR and ImageNet. This flexibility allows users to control the precise performance-memory cost trade-off without making any changes in the backbone architecture. When we additionally tune the backbone architecture width, we can outperform all baselines in the low-parameter regime with the same inference FLOP footprint.

1 INTRODUCTION

Ensembles are one of the most commonly used tools in predictive modelling; they frequently are a critical component of data science competitions (Atwood et al., 2020; Ostyakov & Nikolenko, 2019) because the averaged predictions of multiple models will typically be more accurate than any single ensemble member. Ensembles of neural networks are particularly effective; the stochasticity in the initialization and training of the neural network results in differing models which have decorrelated errors (Fort et al., 2019). They have been shown to increase robustness, calibration, and accuracy (Lakshminarayanan et al., 2016; Dietterich, 2000). Nevertheless, they are memory and compute intensive, because the multiple models need to be trained, stored, and used for inference. Even just storing the resulting models could become a burden for edge devices, which often have specially designed parameter-efficient networks (Howard et al., 2017).

Several methods have been proposed to make ensembles more parameter efficient, such as BatchEnsemble (Wen et al., 2020b), HyperBatchEnsemble (Wenzel et al., 2020), and TreeNet (Lee et al., 2015). All of these methods leverage cross-ensemble parameter sharing in some form, in order to decrease the parameter footprint of the ensemble. Having ensemble networks share parameters can save the storage requirements of the overall ensemble. MIMO (Havasi et al., 2020) minimizes both parameter count and inference time by ensembling subnetworks within a single backbone. While network size can be reduced in other ways, such as with pruning (LeCun et al., 1990; Gale et al., 2019; Lee et al., 2019) or weight quantization (Nagel et al., 2019; Jin et al., 2020), such reduction methods are complementary to parameter sharing methods. In this work, we focus on finding more efficient and flexible ways for ensembles to share parameters.

Savarese & Maire (2019) and Plummer et al. (2020) find that parameters can be shared *across* layers of the same network with little performance penalty. This is accomplished by creating weights through a linear combination of templates between identical layers in a network. This allows for

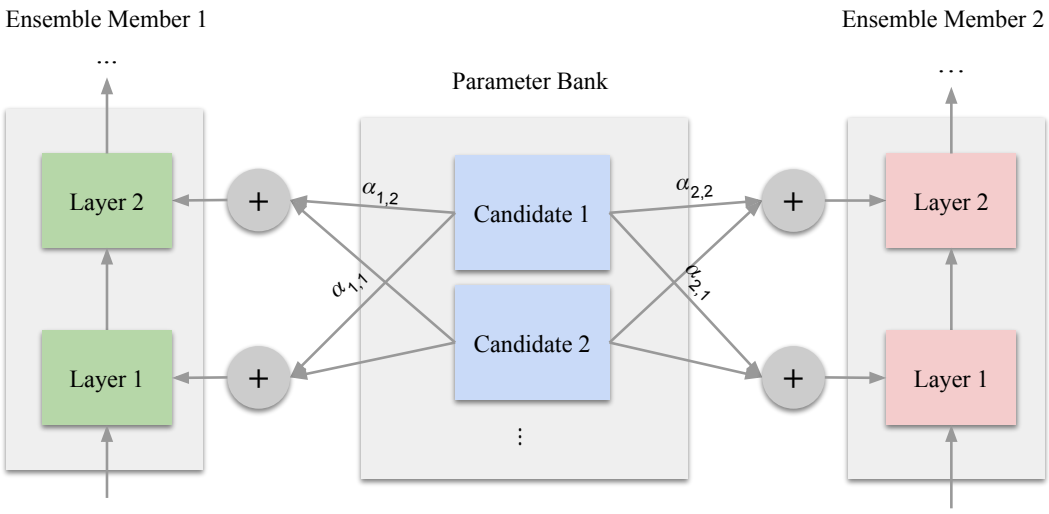

Figure 1: An overview of our MixtureEnsembles method. The parameter bank holds templates shared by all ensemble members. The layer weights of each ensemble member are composed as a linear combination of the template parameters. The exact combination strategy is determined by the linear combination weights $\boldsymbol{\alpha}$ and is learned automatically together with the weights. Therefore, each ensemble member only costs a small number of additional parameters – the size of "combiner weights" $\boldsymbol{\alpha}$ and the member-specific BatchNorm parameters, making this a memory efficient ensembling method.

effective parameter re-use across layers, enabling much smaller models to achieve competitive accuracy with the fully-parameterized model. However, these prior works have only addressed sharing within a single network and have not studied sharing in network ensembles. An optimal sharing scheme in an ensemble maximizes not only individual member accuracy but also ensemble diversity.

We leverage the following guiding principle in our method: we want to learn ensemble members which are related in parameter space, enabling parameter sharing, but are diverse in function space, enabling effective ensembling. Based on this principle, we propose a novel ensembling method, MixtureEnsembles. MixtureEnsembles define each ensemble member by a unique set of linear coefficients that combine shared templates, in addition to independent Batch normalization (BN) layers. (An overview of our MixtureEnsembles can be found on Figure 2.) This results in a diverse ensemble which gives competitive results. Further, this formulation allows us to *precisely tune* the total number of parameters in the ensemble, since it is just a hyper-parameter and is not tied to the network architecture. As the number of parameters goes up, so does the accuracy of the model. This allows the user to easily trade off overall accuracy for memory, and for us to increase parameter count without prohibitive training time increases.

In short, our contributions are as follows:

1. We generalize cross-layer parameter sharing techniques to create a new efficient ensembling technique which allows parameter sharing both between layers and ensemble members.
2. We show this allows for precise control over the memory-accuracy trade-off by decoupling parameter count from architecture, increasing usability of efficient ensembling methods. This allows to decrease our parameter count with more graceful degradation than baselines (Figure 3), and also allows us to increase the number of parameters without prohibitive training time increases. Our method is competitive in the low-parameter regime, and outperforms even naive ensembles in the high-parameter regime. When we additionally tune the backbone width, we can outperform all existing methods for a given memory and inference FLOP footprint.
3. We demonstrate that ensemble member redundancy is minimized through a diversity analysis of our trained ensembles despite MixtureEnsembles sharing parameters.

2 Related Work

Efficient Ensembles: Ensembles have long been known to improve predictive accuracy and uncertainty estimates of classification models (Dietterich, 2000; Lakshminarayanan et al., 2016). More

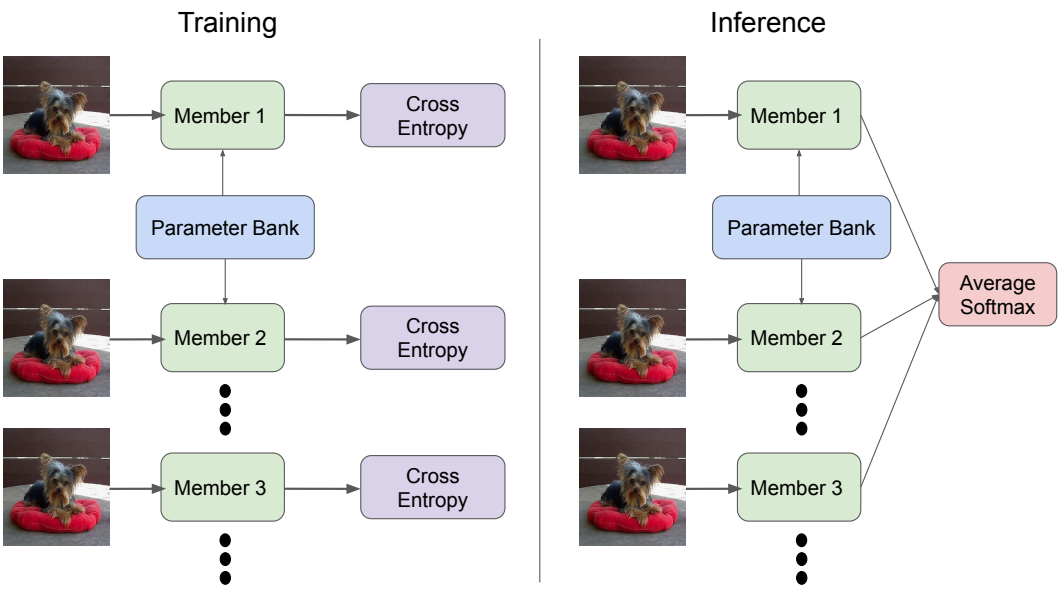

Figure 2: The MixtureEnsemble training and infeerence pipeline. Each ensemble member is instantiated from a shared parameter bank, and is trained independently on the same samples with a cross-entropy loss. At inference, we average predictions from the members.

recently, Fort et al. (2019) characterized ensembles by measuring diversity of the learned functions; the authors find that differently initialized neural networks converge to functions that have independent predictions. This decorrelation is what enables deep ensembles to outperform single models.

There has been much recent interest in efficient implementations of ensembles. MC-dropout uses dropout at inference time to construct ensemble members (Gal & Ghahramani, 2016). Multiheaded networks have been studied as a form of computationally cheaper ensembling in Teterwak & Torresani (2014) and Lee et al. (2015); where a shared initial trunk then bifurcates into different classification heads. These classification heads are considered as different ensemble members. Because of the shared trunk, this significantly decreases computational and memory costs of the ensemble at a small accuracy penalty. A clever riff on this shared trunk idea is MIMO (Havasi et al., 2020), which concatenates several samples in the input layer and has to classify each with a multi-headed output. Therefore, the network has to learn M subnetworks to classify the images, and this functions as an effective ensemble. Most similar to our work is BatchEnsemble (Wen et al., 2020b), which has introduced a factorization scheme which constructs each layers' parameters as a Hadamard product between a full set of weights shared between all ensemble members and a simple rank-1 matrix which is different for each ensemble member. This significantly decreases the memory overhead of the network. In our work we develop a similar, but more flexible and more performant framework.

Parameter Sharing: For single models, there has been much interest in cross-layer sharing for efficient computation. Savarese & Maire (2019) devise a cross-layer parameter sharing scheme where parameters are linear combinations of templates, and templates are shared amongst layers of same shape and size. Plummer et al. (2020) devise a more flexible version of this model called ShapeShifterNetworks. These models reduce parameter footprints of models, reducing their memory usage and communication cost, at a very small penalty to predictive performance. In this work we leverage such techniques to construct more efficient and flexible ensembles. Another model of this flavor is the Isometric Neural Network (Sandler et al., 2019; Zhmoginov et al., 2021), where all layers are the same shape and size, and therefore can very efficiently share layers.

Parameter Pruning: A different approach to parameter efficiency is parameter pruning (Hoefler et al., 2021). Gale et al. (2019) show that pruning can decrease the parameter count with only small performance penalties, with only a few tricks. Frankle & Carbin (2018) call these pruned networks a *lottery ticket*, and show that if you rewind weights of the pruned topology to their initialization, one can train a highly-performant network. However, sparse operations are not well suited to current hardware accelerators. In order to take advantage of dense operations, Li et al. (2016) proposed to prune entire filters. Still, pruning methods cannot leverage parameter sharing.

Parameter Sharing for Multi-task learning: Cross-model sharing has also been explored in the context of multi-task learning. There, the intuition is that different tasks might share the same

information. One basic form of this is a neural network with multiple heads, one for each task as Zhang et al. (2014) do for facial landmark detection and Collobert & Weston (2008) do for NLP. Moving away from hard sharing as with a shared trunk towards a soft sharing, methods like Cross-stitch Networks (Misra et al., 2016) and AdaShare (Sun et al., 2019) learn what they should share between tasks. In this work we aim to learn what to share between two independent models trained on the same task, instead of multiple tasks.

3 METHOD: MIXTUREENSEMBLES

In this section, we describe our method for template sharing across layers and ensemble members, allowing for more parameter efficient ensembles. We build on the parameter sharing techniques that Savarese & Maire (2019) and Plummer et al. (2020) introduced for single models in order to create ensembles of models using shared parameters.

3.1 MODEL ARCHITECTURE

Defining a Single Ensemble Member: For a single model, as in Savarese & Maire (2019), we can group layers of the same type and shape together. Each one of these groups is assigned a set of parameters, which we call a Parameter Group, which is fraction f of the overall parameter blob P. This fraction f is determined by the proportion of parameters used by layers in that group in a standard, non-factorized network. For example, if there are 4 identical residual blocks in a network which together make up 25 percent of the networks parameters, f would be 0.25. Within a parameter group g, the parameters are split into a set of templates T of size n, each with the shape of the layer of the group. Then, each layer is defined as a linear combination of a subset these templates:

$$\theta_j = \sum_{l \in t} \alpha_l T_l \tag{1}$$

where t is a subset of templates T, θ_j are the weights of layer j, and α is a learned scalar multiplier associated with template l. If t and T are unequal sets, then each layer belonging to a particular parameter group g is assigned templates to linearly combine in a round robin fashion, as described in the next section. It is important to note that we are combining *parameters* instead of *features*. This allows for each ensemble member to learn a function that is significantly different from the other members. Finally, when fewer parameters have been allocated to a group than needed (or when there is only a single layer of a particular shape), a single traditional (non-factorized) layer is constructed, to be shared amongst all ensemble members in order to conserve the parameter budget.

Assigning templates to layers: When the parameter group g contains more parameters than is needed for the templates t, it would be inefficient to not use the excess parameters. Therefore, as in Plummer et al. (2020), the templates are assigned to layers within a group in a round robin fashion. This allows for the most efficient use of parameters, without excessive parameter sharing.

Introducing More Ensemble Members: Up to this point, we have described how to construct a *single* ensemble member, which leverages *cross-layer* parameter sharing. However, our goal is to also enable *cross ensemble-member* parameter sharing.

In MixtureEnsembles, additional ensemble members are simply an additional set of linear combination weights α. More formally, the parameters for layer j in ensemble member i are:

$$\theta_{j,i} = \sum_{l \in t} \alpha_{l,i} T_l \tag{2}$$

The α parameters are randomly initialized, seperately for each ensemble member. In this way, each ensemble member gets a pseudo-independent set of weights. We find that a simple random initialization is sufficient to generate a diverse ensemble.

Additionally, each ensemble member gets its own set of Batch Normalization (BN) layers (Ioffe & Szegedy, 2015); since the feature statsitics of each ensemble member are likely to be substantially different. Critically, each additional ensemble member has a very small memory overhead. For

Method	Params	Accuracy	NLL	ECE
Baseline (Havasi et al., 2020)	36.5M	96.0	0.159	2.3 %
BatchEnsembles (Havasi et al., 2020)	36.5M	96.2	0.143	2.1 %
BatchEnsembles + EnsembleBN (Wen et al., 2020a)	36.5M	96.2	n/a	1.8 %
MIMO (Havasi et al., 2020)	36.5M	**96.4**	0.123	1.0 %
MixtureEnsembles (**Ours**)	36.5M	96.3	**0.120**	**0.8** %
Naive Ensemble	146M	**96.6**	**0.114**	1.0 %
MixtureEnsembles (**Ours**)	120M	**96.6**	0.119	**0.8 %**

Table 1: CIFAR-10. For Accuracy, higher is better. For NLL and ECE, lower is better. All models use the WRN-28-10 architechture. We find that in the low parameter regime, MixtureEnsembles outperforms the baseline on 2 out of 3 metrics. In the high parameter regime, we match or exceed naive ensembles on all metrics with 17% fewer parameters.

example, for a WRN-28-10 as is used for our CIFAR-100 and CIFAR-100-C experiments, each ensemble member costs a few thousand parameters: the 105 linear combination weights plus the BN parameters.

3.2 Training and Inference

Training the Ensemble: We train the ensemble simply by feeding the same input batch to each ensemble member. Each ensemble member is trained independently with a standard cross-entropy loss. We found that that the system is otherwise plug-and-play with a vanilla classification network. Because each ensemble member has a different initialization of α parameters and BN layers, the members become independent and form an effective ensemble.

Inference: As in a naive ensemble, at inference time, each input sample gets propagated through each ensemble member. Then, we average the softmax outputs of each member to get a final probability distribution. Since each ensemble member learns a different function, it makes its classification decision based on slightly different input features. Then, averaging the outputs allows the ensemble to make a robust decision. Although we do need to forward-propagate through each ensemble member, this can be trivially parallelized on the inference device as long as it has sufficient accelerator hardware, because ensemble members can be run in parallel.

Parameter Efficiency and Flexibility: The number of parameters in the model is simply a hyper-parameter. This makes for a very *flexible* system. If the user is constrained by memory, they can simply make the parameter groups used for constructing templates smaller. On the other hand, if the target system has fewer constraints and requires more predictive performance, the user can train with more parameters. This flexibility is a significant advantage over competing methods, such as BatchEnsemble and MIMO. To adapt the number of parameters in those methods, one must adjust the network architecture such as increasing the number of feature channels. Architecture adjustments have side-effects such as compute cost (number of FLOPS) and learning dynamics. No other hyper-parameters, or parts of the training procedure, are changed when we adjust the the number of parameters; we treat MixtureEnsemble as a plug-and-play system.

Computational Costs: The training and inference cost of MixtureEnsembles is comparable to baselines, while improving accuracy. Although at first glance MIMO is more computationally efficient than MixtureEnsembles, at train time it actually repeats each batch 4 times, making it computationally comparable to other baselines. At inference time MIMO appears to be more efficient than other efficient ensembling methods. However, MIMO does not share parameters between ensemble members; it is better characterized as method which finds smaller subnetworks which function as an ensemble within a larger network. We can explicitly make our subnetworks smaller by decreasing the width of the ensemble members, decreasing the number of inference FLOPS. Normalizing by inference FLOPS, MixtureEnsembles perform better than MIMO.

MixtureEnsembles also have a memory storage advantage; for a given parameter count MixtureEnsembles have better performance than baselines.

Method	Params	Clean	Corrupt	FLOPS
Baseline (Havasi et al., 2020)	36.5M	79.8	51.4	1x
BatchEnsembles (Havasi et al., 2020)	36.5M	81.5	54.1	4x
BatchEnsembles + EnsembleBN (Wen et al., 2020a)	36.5M	81.9	54.1	4x
MIMO (Havasi et al., 2020)	36.5M	82.0	53.7	1x
MixtureEnsembles (**Ours**)	36.5M	82.2	52.9	4x
Thin MixtureEnsembles (**Ours**)	36.5M	**82.3**	**54.2**	1x
Naive Ensemble	146M	82.7	54.1	4x
MixtureEnsembles (**Ours**)	120M	**82.9**	**54.7**	4x

Table 2: Accuracy on CIFAR-100 (clean) and CIFAR-100-C (corrupted). All ensembles were of size 4 and use the same WRN-28-10 architechture, excpet for thin MixtureEnsembles, which use WRN-28-5. We find that our WRN-28-10 model is competitive with MIMO at 36.5 million parameters, but our WRN-28-5 model outpeforms MIMO with the same inference footprint. With more parameters our MixtureEnsembles WRN-28-10 achieves the highest performance overall while reducing parameter count by 17% relative to naive ensembles. Baseline corresponds to a single WRN-28-10 model trained with cross-entropy. FLOPS refers to inference-time FLOPS, relative to the baseline.

Method	Parameters	NLL	ECE	FLOPS
Baseline (Havasi et al., 2020)	36.5M	0.875	8.6%	1x
BatchEnsembles (Havasi et al., 2020)	36.5M	0.740	5.6%	4x
BatchEnsembles + EnsembleBN (Wen et al., 2020a)	36.5M	n/a	2.8%	4x
MIMO (Havasi et al., 2020)	36.5M	0.690	**2.2%**	1x
MixtureEnsembles (**Ours**)	36.5M	0.702	2.7%	4x
Thin MixtureEnsembles (**Ours**)	36.5M	**0.689**	2.4%	1x
Naive Ensemble	146M	**0.666**	**2.1%**	4x
MixtureEnsembles (**Ours**)	120M	**0.666**	2.2%	4x

Table 3: Negative Log-Likelihood and Expected Calibration Error on the CIFAR-100 test set. Lower is better. Once again we are competitive with MIMO at lower parameter counts, and are competitive with naive ensembles at 17% lower parameter count. Baseline corresponds to a single WRN-28-10 model trained with cross-entropy.

4 Benchmark Experiments

We evaluate our method on three standard benchmarks: CIFAR-10, CIFAR-100 and ImageNet. Additionally, as one of the advantages of ensembles is their robustness to out-of-distribution data, we evaluate the robustness of our method on CIFAR-100-C.

4.1 CIFAR-10, CIFAR-100, and CIFAR-100-C

We showcase CIFAR results in Tables 1, 2, 3, and 4. For all CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009) models except *Thin Mixture Ensembles*, we train a WRN-28-10 network architecture. *Thin Mixture Ensembles* are WRN-28-5 models. These thinner models match the inference-time FLOPS of MIMO, yet have higher accuracy. Trying to make MIMO thinner reduces performance (see Figure 3). Additional training details are in the Appendix.

We compute accuracy for both CIFAR-100 and CIFAR-100-C. CIFAR-100-C (Hendrycks & Dietterich, 2019) is a verson of the CIFAR-100 test set which is corrupted with distortions such as Gaussian blur and JPEG compression at various severities. We evaluate accuracy in addition to the calibration metrics of Negative Log-Likelihood (NLL) and Expected Calibration Error (ECE) (Naeini et al., 2015; Guo et al., 2017).

We find that our WRN-28-10 model is competitive with BatchEnsemble and MIMO at 36.5 million parameters, yet outperforms even naive ensembles with higher numbers of parameters. The only difference in training was a single hyper-parameter; the number of parameters available to the model in the parameter store. Neither BatchEnsemble's nor MIMO have a way of interpolating between the low-parameter and high-parameter regime without changing network architecture. A larger archi-

Method	Parameters	NLL	ECE	FLOPS
Baseline (Havasi et al., 2020)	36.5M	2.70	23.9%	1x
BatchEnsembles (Havasi et al., 2020)	36.5M	2.49	19.1%	4x
BatchEnsembles + EnsembleBN (Wen et al., 2020a)	36.5M	n/a	19.1%	4x
MIMO (Havasi et al., 2020)	36.5M	2.28	12.9%	1x
MixtureEnsembles (**Ours**)	36.5M	2.17	10.3%	4x
Thin MixtureEnsembles (**Ours**)	36.5M	**2.09**	**9.7%**	1x
Naive Ensemble	146M	2.27	13.8%	4x
MixtureEnsembles (**Ours**)	120M	**2.00**	**10.3%**	4x

Table 4: NLL and ECE for CIFAR-C dataset of *corrupted* images from CIFAR. Lower is better. We find that overparameterized MixtureEnsembles outperform even Naive Ensembles, and that our WRN-28-5 model outperforms all competitors with the same FLOP footprint. Baseline corresponds to a single WRN-28-10 model trained with cross-entropy.

tecture dramatically increases training time; for example scaling MIMO to 120M parameters would result in over 3x the training FLOPS of our MixtureEnsembles. We further explore the behavior of the model at various parameter operating points in Section 5.1.

In their paper the authors of MIMO demonstrated their benefits could not be achieved using an ensemble of thin networks, but this is not true of our MixtureEnsembles. When we change the trained model from WRN-28-10 to WRN-28-5 (*Thin MixtureEnsembles*), we can can outperform MIMO at the same inference FLOP footprint. This experiment is motivated by the fact that MIMO doesn't share parameters between ensemble members, and instead learns subnetworks within a wider network which become ensemble members. We can accomplish this explicitly by simply making each ensemble member thin. Note that MixtureEnsembles can still perform inference on single thin ensemble member when computational budgets are low unlike MIMO, further illustrating the greater flexibility of our approach. We further note that we compare against two versions of BatchEnsemble cited in the literature; with and without independent BatchNorm layers for each ensemble member.

4.2 IMAGENET

In order to test the system on a larger scale dataset, we train our model on ImageNet (Deng et al., 2009). We use the standard ResNet-50 architecture. Additional training details are in the appendix.

To encourage maximum ensemble diversity for our ImageNet experiments, we assign disjoint templates within the parameter bank greedily. Once all disjoint templates have been assigned, templates start being reused. Note that we found this unnecessary with CIFAR, and assigned the same templates to corresponding layers in different ensemble members. More details are in Section 3.1.

The results are in Table 4.2. In the low parameter regime, we find that results are competitive with BatchEnsemble and MIMO, demonstrating that MixtureEnsembles improve over the ResNet-50 baselines even on large scale visual recognition tasks. When we overparamterize MixtureEnsembles, we once again outperform Naive Ensembles with only **half** of the parameters.

Method	Parameters	# Ensemble Members	Top-1 Accuracy
Baseline	25.6M	1	76.4%
BatchEnsemble	25.6M	4	76.7%
MIMO	25.6M	3	**77.5%**
MixtureEnsemble (**Ours**)	25.6M	4	77.2%
Naive Ensemble	102.4M	4	77.5%
MixtureEnsemble (**Ours**)	51.2M	4	**77.9%**

Table 5: ImageNet Results on ResNet-50. MixtureEnsembles are competitive with other methods in the low-parameter regime. In the high-parameter regime, we outperform Naive Ensembles with **50%** fewer parameters.

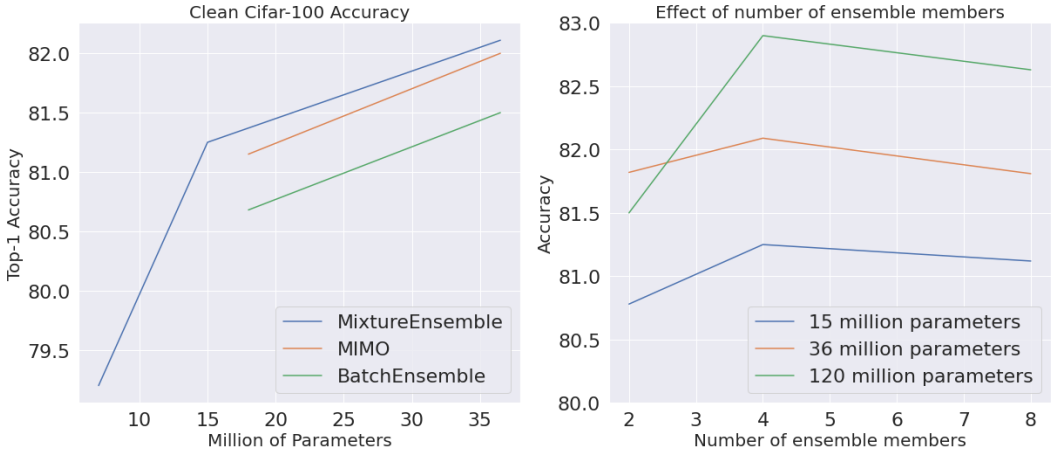

Figure 3: On the left, we show the effect of scaling parameters for different efficient ensembling methods. MixtureEnsembles scale down most gracefully. On the right, we show the effect of number of ensemble members for CIFAR-100. We find that across a number of parameter sizes, ensemble size 4 is best.

5 Analysis

5.1 Scaling Properties: Parameters

One fundamental feature of our model is the ease with which we can scale the number of parameters. Therefore, it is natural to study how one can trade off performance and memory requirements in our system. We run experiments that show how, as we increase the number of parameters, the system becomes more and more competitive in both accuracy and uncertainty metrics.

We find that as we increase the number of parameters, accuracy improves. This allows the user to choose an operating point for their use case. In the extremely low parameter regime; we only lose a little bit of performance compared to the baseline. In the high parameter regime, we even outperform naive ensembles. Results are shown in Figure 3.

We train an ensemble of size 4 at each of the following points: 7 million parameters, 15 million parameters, and 36.5 million parameters. For the 7 million parameter case, we need to make a slight modification to the training procedure; instead of grouping parameters by layer type we instead have a single parameter group for all layers. As in Plummer et al. (2020), we then reshape the single parameter blob separately for each layer, to form templates for the layer. There will sometimes be extra parameters in the blob not needed for the templates for a particular layer; in that case parameters are selected in a round-robin fashion for subsequent layers.

5.2 Scaling number of ensemble members

In addition to varying the number of parameters, we also varied the number of ensemble members across several ensemble parameter sizes. We see that across all sizes, 4 ensemble members is the optimal (see Figure 3). We hypothesize that sharing templates is also a form of regularization, since the update of one member also perturbs the parameters of another member. Furthermore, as the number of ensemble members grows, the number of parameters per member becomes more constrained, decreasing the per-member accuracy.

5.3 Interpolation Analysis of Diversity

Inspired by similar analysis in Fort et al. (2019) and Neyshabur et al. (2020), in Figure 4 we interpolate between two ensemble members in parameter space to see if the models indeed are in different optimization basins. We accomplish this by interpolating parameters, and leaving Batch Normalization (Ioffe & Szegedy, 2015) in train mode because accumulated statistics are not meaningful at interpolated points. If the interpolates have high accuracy, this indicates the ensemble members

Method	Params	Accuracy	Diversity
Naive Ensemble (Havasi et al., 2020)	146M	82.7	0.88
MixtureEnsembles (**ours**)	36.5M	82.1	0.78
MixtureEnsembles (**ours**)	120M	82.9	0.85
BatchEnsembles + EnsembleBN (Wen et al., 2020a)	36.5M	81.9	0.4
MIMO (Havasi et al., 2020)	36.5M	82.0	0.91

Table 6: Diversity. MixtureEnsembles are more diverse than BatchEnsembles, the other shared parameter model. Although our 120M model is slightly less diverse than Naive Ensembles, the performance is higher, indicating that average member accuracy is higher. This suggests that MixtureEnsembles are also an effective regularizer.

landed in the same optimization basin and therefore are not as diverse as they could be. We find that interpolates have much decreased accuracy compared to the end points, supporting the idea that our model learns independent ensemble members.

We can see that although all networks experience some degree of accuracy drop between ensemble members, the accuracy drop for the low-parameter model is significantly lower. This seems to indicate that as the function space becomes constrained with a lower number of parameters in the parameter bank, the ensemble diversity starts to suffer.

We additionally use a diversity metric introduced in Fort et al. (2019), which measure the fraction of differing predictions by two ensemble members, normalized the by the error of one of them. We present these results in Table 6; we can see that Mixture Enesmbles are more diverse than all other shared parameter methods. One interesting finding is that 120M parameter MixtureEnsembles outperform Naive Ensembles, yet have lower diversity. Looking deeper, the individual model accuracies are improved for MixtureEnsembles (average naive 79.89% vs average Mixture Ensemble 80.36%) Therefore, it seems like MixtureEnsembles are also an effective regularizer. This could come from the fact that the parameter factorization limits the space of possible weights, or from the parameter sharing. This is especially interesting because WRN-28-10 is already a highly optimized model from a hyper-parameter perspective. Of note is that one of the ensemble members from MixtureEnsembles gives higher performance (80.49% for the best model) than a single naive model (80.1%). Therefore one could use MixtureEnsembles to train a highly performant single model.

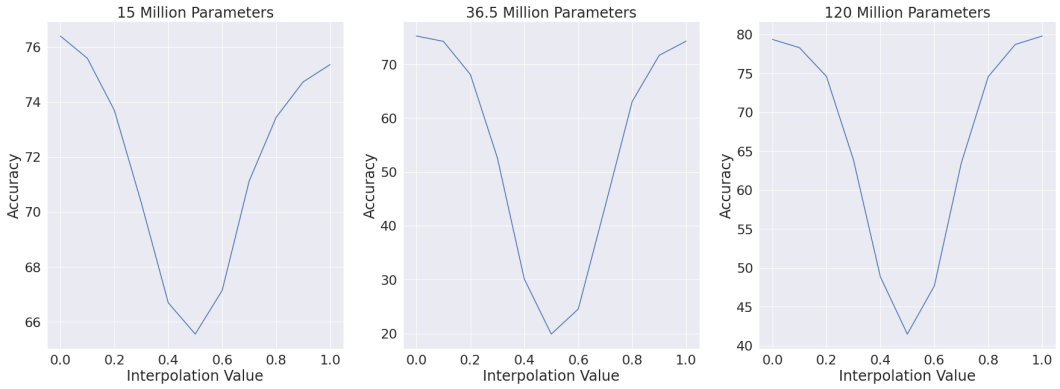

Figure 4: Linear interpolations in parameter space, with different numbers of parameters in an ensemble of size 4. We plot accuracy vs interpolation point. Because the accuracy takes a significant dip, we can tell that the ensemble members are independent and find different local minima.

6 Conclusion

In this work, we propose MixtureEnsembles, a novel and efficient ensembling method. MixtureEnsembles leverage both cross-layer and cross-ensemble member sharing. MixtureEnsembles well across a wide range of paramter size operating points, allowing for a user to easily choose performance-memory usage tradeoff. They give state-of-the-art results on both CIFAR and ImageNet datasets, and we show that the ensemble members are indeed independent.

7 ETHICS STATEMENT

Ensembles are more robust and better calibrated than single models, which can translate directly to fairer and safer deployments. For example, the Inclusive Images Challenge (Atwood et al., 2020) allowed competitors to train on images from one geographical region (e.g. the global North) and evaluate on a different geographical region (e.g. the global South). The winning solution by Ostyakov & Nikolenko (2019) uses a very large ensemble, illustrating the great potential of ensembles. Decreasing parameter cost of ensembles, as our method does, makes training and deploying ensembles more accessible.

Although efficient ensembling does indeed increase robustness of models, we caution readers that even more robust and accurate models should not be trusted blindly. It is prudent to thoroughly audit all models before and during deployment, and carefully consider how building and deploying them may affect the lives of others. We additionally note that the ImageNet dataset may contain personally identifiable information. However, it remains a commonly used benchmark and therefore is an important comparison.

8 REPRODUCIBILITY STATEMENT

We provide all training details in the Appendix, and provide code in the Supplementary Materials. At acceptance, we will provide GitHub repository for readers.

REFERENCES

James Atwood, Yoni Halpern, Pallavi Baljekar, Eric Breck, D Sculley, Pavel Ostyakov, Sergey I Nikolenko, Igor Ivanov, Roman Solovyev, Weimin Wang, et al. The inclusive images competition. In *The NeurIPS'18 Competition*, pp. 155–186. Springer, 2020.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pp. 1321–1330. PMLR, 2017.

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2019.

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3994–4003, 2016.

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1325–1334, 2019.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*, 2020.

Pavel Ostyakov and Sergey I Nikolenko. Adapting convolutional neural networks for geographical domain shift. *arXiv preprint arXiv:1901.06345*, 2019.

Bryan A Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. Neural parameter allocation search. *arXiv preprint arXiv:2006.10598*, 2020.

Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

Pedro Savarese and Michael Maire. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019.

Piotr Teterwak and L Torresani. Shared roots: Regularizing neural networks through multitask learning. *TR2014-762, Dartmouth*, 2014.

Yeming Wen, Ghassen Jerfel, Rafael Muller, Michael W Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran. Combining ensembles and data augmentation can harm your calibration. *arXiv preprint arXiv:2010.09875*, 2020a.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020b.

Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *arXiv preprint arXiv:2006.13570*, 2020.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pp. 94–108. Springer, 2014.

Andrey Zhmoginov, Dina Bashkirova, and Mark Sandler. Compositional models: Multi-task learning and knowledge transfer with modular networks, 2021. URL `https://openreview.net/forum?id=paUVOwaXTAR`.

A Appendix

A.1 CIFAR Training Details

For CIFAR-100 experiments, we use the common WRN-28-10 Zagoruyko & Komodakis (2016) architecture. We train for 200 epochs with an initial learning rate of 0.1. We use SGD with a Nestorov momentum value of 0.9. We use a weight decay value of $5e{-}4$ on all parameters except weight coefficients. We decay the learning rate by a factor of 0.2 at 60,120,and 180 epochs. We use a batch size of 128, and use asynchronous BatchNorm across two devices (so BatchNorm batch size is 64). We pad images by 4 pixels and crop to 32 x 32 pixels, and also randomly flip and normalize such that it is zero-mean for training. At test time, we just normalize with no other image transformations.

For *Thin BatchEnesmble*, we use WRN-28-5 and train for 1000 epochs to match the train FLOPS of MIMO. We decay learning rate at epochs 300,600, and 900. We also use parameter banks even when there is only a single layer within a model which has a particular shape in a group (see Section 3.1), reducing implicit regularization for the narrower model. Otherwise training is the same as for WRN-28-10 models.

A.2 ImageNet Training Details

For ImageNet experiments, we use the ResNet-50 architecture He et al. (2016). We train for 90 epochs with a learning rate of 1.6. We use SGD with non-Nestorov momentum value of 0.9. We use a batch size of 1024. We decay the learning rate at epochs 30, 60, and 80 by a factor of 0.1. We use a label smoothing value of 0.1, and a weight deay of 0.0001. The weight decay is not applied to batch norm parameters. We do a linear warmup for the first 10 epochs. We use standard Inception-style data augmentation.