PACKED-ENSEMBLES FOR EFFICIENT UNCERTAINTY ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Ensembles (DE) are a prominent approach achieving excellent performance on key metrics such as accuracy, calibration, uncertainty estimation, and out-of-distribution detection. However, hardware limitations of real-world systems constrain to smaller ensembles and lower capacity networks, significantly deteriorating their performance and properties. We introduce Packed-Ensembles (PE), a strategy to design and train lightweight structured ensembles by carefully modulating the dimension of their encoding space. We leverage grouped convolutions to parallelize the ensemble into a single common backbone and forward pass to improve training and inference speeds. PE is designed to work under the memory budget of a single standard neural network. Through extensive studies we show that PE faithfully preserve the properties of DE, e.g., diversity, and match their performance in terms of accuracy, calibration, out-of-distribution detection and robustness to distribution shift.

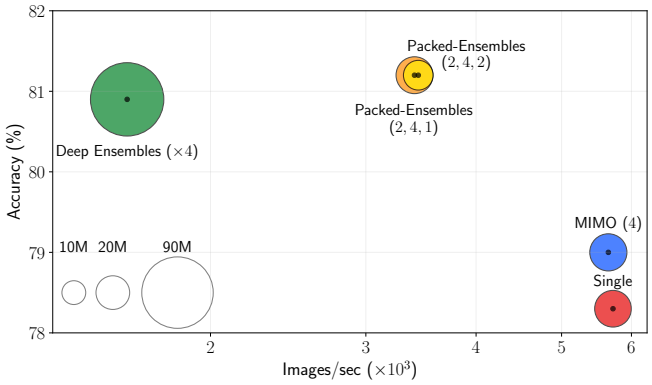


Figure 1: **Evaluation of computation cost and performance trade-offs for multiple uncertainty quantification techniques on CIFAR-100.** The y-axis shows the accuracy and the x-axis the inference time in images per second. The circle area is proportional to the number of parameters. Optimal approaches should be closer to the top-right corner. Packed-Ensembles strikes a good balance between predictive performance and speed.

1 INTRODUCTION

Real-world safety-critical machine learning decision systems such as autonomous driving (Levinson et al., 2011; McAllister et al., 2017) impose particularly high reliability and performance requirements over a broad range of metrics: accuracy, calibration, robustness to distribution shifts, uncertainty estimation, and computational efficiency under limited hardware resources. Although their performance across all key dimensions has been dramatically improved in the last years, vanilla Deep Neural Networks (DNNs) still have several shortcomings, notably the overconfidence on both correct and wrong predictions (Nguyen et al., 2015; Guo et al., 2017; Hein et al., 2019). Deep Ensembles (DE) (Lakshminarayanan et al., 2017) arise as a prominent approach to address these challenges by leveraging predictions from multiple high-capacity neural networks. By averaging predictions or by voting, DE achieve high accuracy and robustness since potentially unreliable predictions are exposed via the disagreement between individuals. Thanks to the simplicity and effec-

tiveness of the ensembling strategy (Dietterich, 2000), DE have become largely used and dominate performance across various benchmarks (Ovadia et al., 2019; Gustafsson et al., 2020).

DE tick all boxes in the requirements list for real-world applications, except for computational efficiency. In fact, DE are in practice computationally demanding (required memory storage, number of operations, inference time) both for training and testing as their costs grow linearly with the size of the ensembles. Their computational costs are therefore prohibitive under tight hardware constraints. This limitation of DE has inspired numerous approaches proposing computationally efficient alternatives: multi-head networks (Lee et al., 2015), architectures with ensemble-imitating layers (Wen et al., 2019; Havasi et al., 2020; Ramé et al., 2021), multiple forwards on different weight subsets of the same network (Gal & Ghahramani, 2016; Durasov et al., 2021), ensembles of smaller networks (Kondratyuk et al., 2020; Lobacheva et al., 2020), computing ensembles from a single training run (Huang et al., 2017; Garipov et al., 2018), efficient Bayesian Neural Networks (Maddox et al., 2019; Franchi et al., 2020). These approaches typically improve storage usage, train cost or inference time at the cost of lower accuracy and lower diversity in the predictions.

An essential property of ensembles that enables improved predictive uncertainty estimation is related to the diversity of predictions in the ensemble. Perrone & Cooper (1992) have shown that the independence of individuals is critical to the success of ensembling. Fort et al. (2019) argue that the diversity of DE due to randomness from weight initialization, data augmentation and batching, stochastic gradient updates, etc., is superior to various other efficient ensembling approaches, in spite of their predictive performance boosts. Few works manage to mirror this property of DE in a computationally efficient manner close to a single DNN (in terms of memory usage, number of forward passes, image throughput, etc.).

In this work, we aim to design a DNN architecture that closely mimics properties of ensembles, in particular having a set of independent networks, in a computationally efficient manner. Previous works propose ensembles composed of small models (Kondratyuk et al., 2020; Lobacheva et al., 2020) and achieve performances comparable to a single large model. We build upon this idea and devise a strategy based on small networks towards matching the performance of an ensemble of large networks. To this end we leverage *grouped convolutions* (Krizhevsky et al., 2012) to delineate multiple subnetworks within the same network. The parameters of each subnetworks are not shared across subnetworks, leading to independent smaller models. This enables fast training and inference times, while predictive uncertainty quantification is close to DE (Figure 1).

In summary, our contributions are the following:

- We propose *Packed-Ensembles* (PE), an efficient ensembling architecture relying on grouped convolutions, as a formalization of structured-sparsity for Deep Ensembles;
- We extensively evaluate PE in terms of accuracy, calibration, OOD detection, and dataset shift on classification and regression tasks. We demonstrate that PE achieves state-of-the-art predictive uncertainty quantification.
- We thoroughly study and discuss the properties of PE (diversity, sparsity, stability, behavior of subnetworks) and release our PyTorch implementation.

2 Background

In this section, we present the formalism for this work and offer a short background on grouped convolutions and ensembles of DNNs.

2.1 Background on convolutions

The convolutional layer (LeCun et al., 1989) consists in a series of cross-correlations between feature maps $\mathbf{h}^j \in \mathbb{R}^{C_j \times H_j \times W_j}$ regrouped in batches of size B and a weight tensor $\boldsymbol{\omega}^j \in \mathbb{R}^{C_{j+1} \times C_j \times s_j^2}$ with C_j, H_j, W_j three integers that represent respectively the number of channels, the height and the width of $\mathbf{h}^j$. C_{j+1} and s_j are also two integers corresponding respectively to the number of channels of $\mathbf{h}^{j+1}$ (the output of the layer) and the kernel size. Finally, j is the layer's index and will be fixed in the following formulae. The main notations are detailed in Table 3, in the supplementary material. The bias of convolution layers will be omitted in the following for simplicity. Hence the output value of the convolution layer, denoted $\circledast$, is:

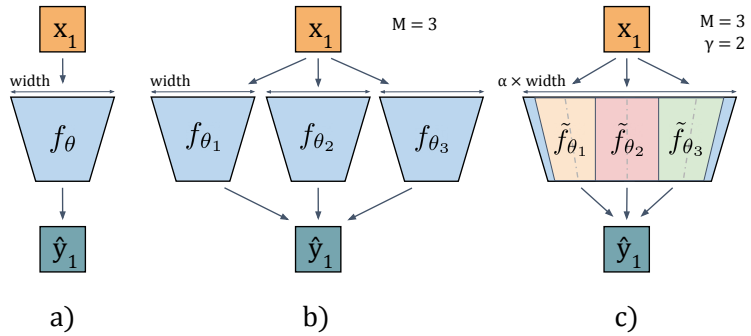


Figure 2: **Overview of the considered architectures:** (*left*) baseline vanilla network; (*center*) Deep Ensembles; (*right*) Packed-Ensembles-$(\alpha, M = 3, \gamma = 2)$.

$$\mathbf{z}^{j+1}(c,:,:) = (\mathbf{h}^j \circledast \boldsymbol{\omega}^j)(c,:,:) = \sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k,:,:) \star \mathbf{h}^j(k,:,:) \tag{1}$$

where $c \in [\![0, C_{j+1}-1]\!]$ is the index of the considered channel of the output feature map, $\star$ is the classical 2D cross-correlation operator, and $\mathbf{z}^j$ is the pre-activation feature map such that $\mathbf{h}^j = \phi(\mathbf{z}^j)$ with ϕ an activation function.

To embed an ensemble of subnetworks, we leverage grouped convolutions, already used in ResNext (Xie et al., 2017) to train several DNN branches in parallel. The grouped convolution operation with γ groups and weights $\boldsymbol{\omega}^i_\gamma \in \mathbb{R}^{C_{j+1} \times \frac{C_j}{\gamma} \times s_j^2}$ is given in (2), γ dividing C_j for all layers. Any output channel c is produced by a specific group (set of filters), identified by the integer $\left\lfloor \frac{\gamma c}{C_{j+1}} \right\rfloor$, which only uses $\frac{1}{\gamma}$ of the input channels.

$$\begin{aligned}\mathbf{z}^{j+1}(c,:,:) &= (\mathbf{h}^j \circledast \boldsymbol{\omega}^j_\gamma)(c,:,:) \\ &= \sum_{k=0}^{\frac{C_j}{\gamma}-1} \boldsymbol{\omega}^j_\gamma(c,k,:,:) \star \mathbf{h}^j\left(k + \left\lfloor \frac{\gamma c}{C_{j+1}} \right\rfloor \frac{C_j}{\gamma}, :, :\right)\end{aligned} \tag{2}$$

The grouped convolution layer is mathematically equivalent to a classical convolution where the weights are multiplied element-wise by the binary tensor $\text{mask}_m \in \{0,1\}^{C_{j+1} \times C_j \times s_j^2}$ such that $\text{mask}^j_m(k,l,:,:) = 1$ if $\left\lfloor \frac{\gamma l}{C_j} \right\rfloor = \left\lfloor \frac{\gamma k}{C_{j+1}} \right\rfloor = m$ for each group $m \in [\![0, \gamma-1]\!]$. The complete layer mask is finally defined as $\text{mask}^j = \sum_{m=0}^{\gamma-1} \text{mask}^j_m$ and the grouped convolution can therefore be rewritten as:

$$\mathbf{z}^{j+1} = \mathbf{h}^j \circledast \left(\boldsymbol{\omega}^j \circ \text{mask}^j\right) \tag{3}$$

where $\circ$ is the Hadamard product.

2.2 Background on Deep Ensembles

For a classification problem, let us define a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{|\mathcal{D}|}$ containing $|\mathcal{D}|$ pairs of samples $\mathbf{x}_i \in \mathbb{R}^{H_j \times W_j}$ and one-hot-encoded labels $\mathbf{y}_i \in \mathbb{R}^{N_C}$ modeled as the realisation of a joint distribution $\mathcal{P}_{(X,Y)}$ where N_C is the number of classes in the dataset. The input data $\mathbf{x}_i$ is processed via a neural network $f_{\boldsymbol{\theta}}$ which is a parametric probabilistic model such that $\hat{\mathbf{y}}_i = f_{\boldsymbol{\theta}}(\mathbf{x}_i) = P(Y = \mathbf{y}_i | X = \mathbf{x}_i; \boldsymbol{\theta})$. This approach consists in considering the prediction $\hat{\mathbf{y}}$ as parameters of a Multinoulli distribution.

To improve the accuracy of predictions but also the accuracy of the predicted uncertainties and the detection of OOD samples, Lakshminarayanan et al. (2017) have proposed to ensemble M randomly

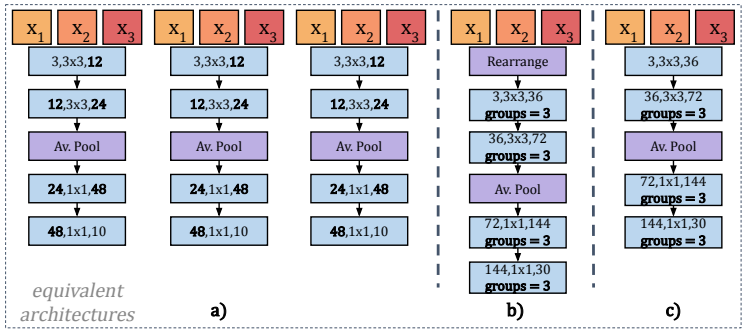


Figure 3: **Equivalent architectures for Packed-Ensembles. (a)** corresponds to the first sequential version, **(b)** to the version with the rearrange operation and grouped convolutions and **(c)** to the final version beginning with a full convolution.

initialized deep Neural Networks as a large predictor called Deep Ensembles. These ensembles can be seen as a discrete approximation of the intractable bayesian marginalization on the weights, according to Wilson & Izmailov (2020). If we note $\{\boldsymbol{\theta}_m\}_{m=0}^{M-1}$ the set of trained weights for the M DNNs, Deep Ensembles consists in averaging the predictions of these M DNNs as in equation (4).

$$P(\mathbf{y}_i|\mathbf{x}_i, \mathcal{D}) = \frac{1}{M} \sum_{m=0}^{M-1} P(\mathbf{y}_i|\mathbf{x}_i, \boldsymbol{\theta}_m) \tag{4}$$

3 Packed-Ensembles

In this section, we present how to train multiple subnetworks efficiently using grouped convolution. Then, we explain how our new architectures are equivalent to training several networks in parallel.

3.1 Revisiting Deep Ensembles

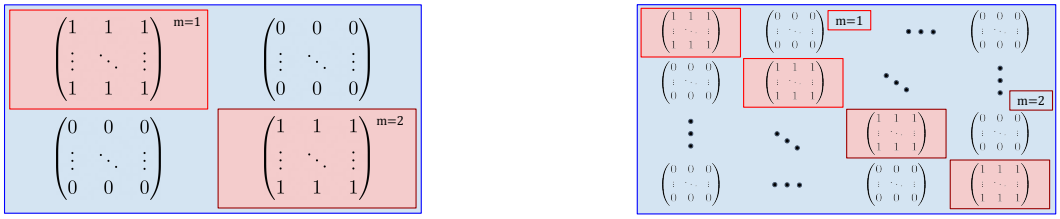
Although Deep Ensembles provide undisputed benefits, they also come with the major drawback that the training time and the memory usage in inference increase linearly with the number of networks. To alleviate these problems, we propose assembling small sub-neural networks, which are DNNs with fewer parameters. Moreover, while ensembles to this day have been trained sequentially, we propose to leverage grouped convolutions to accelerate massively their training and inference computations thanks to their smaller size. The propagation of grouped convolutions with M groups, M being the number of subnetworks in the ensemble, ensures that the sub-neural networks are trained independently while dividing their encoding dimension by a factor M. More details on the usefulness of grouped convolutions to train ensembles can be found in subsection 3.3.

To create Packed-Ensembles (illustrated in Figure 2), we build on small subnetworks but compensate the dramatic decrease of the model capacity by multiplying the width by the hyperparameter α, which can be seen as a expansion factor. Hence, we propose *Packed-Ensembles*-$(\alpha, M, 1)$ as a flexible formalization of ensembles of small subnetworks. For an ensemble of M subnetworks, Packed-Ensembles-$(\alpha, M, 1)$ therefore modifies the encoding dimension by a factor $\frac{\alpha}{M}$ and the inference of our ensemble is computed with the following formula:

$$\hat{\mathbf{y}} = \frac{1}{M} \sum_{m=0}^{M-1} P(\mathbf{y}|\boldsymbol{\theta}_{\alpha,m}, \mathbf{x}) \text{ with } \boldsymbol{\theta}_{\alpha,m} = \{\boldsymbol{\omega}_\alpha^j \circ \text{mask}_m^j\}_j, \tag{5}$$

where $\boldsymbol{\omega}^{j,\alpha}$ is the weight of the layer j of dimension $(\alpha C_{j+1}) \times (\alpha C_j) \times s_j^2$.

In the following, we also add another hyperparameter γ which corresponds to the number of groups of each subnetworks of the Packed-Ensembles and creates another level of sparsity to our ensembles. These groups are also called "subgroups" and are applied to the different subnetworks. Formally, we denote our technique *Packed-Ensembles*-(α, M, γ), where the hyperparameters are in the parentheses. In this work, we consider the case of a constant number of subgroups across the layers and therefore γ divides αC_j for all j.



(a) $M = 2, \gamma = 1$

(b) $M = 2, \gamma = 2$

Figure 4: Diagram representation of a subnetwork mask: $\texttt{mask}^j$, with $M = 2$, j an integer corresponding to a fully connected layer

3.2 Computational cost

In a convolutional setting, the number of parameters in a layer involving C_j input channels, C_{j+1} output channels, kernels of size s_j and γ subgroups is equal to $M \times \left[\frac{\alpha C_j}{M} \frac{\alpha C_{j+1}}{M} s_j^2 \gamma^{-1}\right]$.

The same formula applies to dense layers as 1×1 convolutions. If architectures of the estimators are fully convolutional or dense, two specific cases emerge. If $\alpha = \sqrt{M}$: the number of parameters in the ensemble is equal to the number of parameters in a single model and if $\alpha = M$, each subnetwork corresponds to a single model (and their ensemble is therefore equivalent to Deep Ensembles).

3.3 Implementation details

In this paper, we propose a simple way of designing efficient ensemble convolutional layers using grouped convolutions. To take advantage of the parallelization capabilities of GPUs in training and inference, we replace the sequential training architecture, **(a)** on Figure 3, by the parallel implementations **(b)** and **(c)**. Figure 3 summarizes different equivalent architectures for a simple ensemble of $M = 3$ DNNs with 3 convolutional layers and a final dense layer (equivalent to a 1×1 convolution) with $\alpha = \gamma = 1$.

In **(b)**, we propose to stack the feature maps on the channel dimension (denoted as the "rearrange"[1] operation). This yields a feature map $\mathbf{h}^j$, of size $M \times C_i \times H_j \times W_j$ regrouped by batches of size only $\frac{B}{M}$, with B the batch size of the ensemble. One solution to keep the same batch size is to repeat the batch M times so that its size is equal to B after the rearrangement. Using convolutions with M groups and γ subgroups per subnetworks, each feature map is convoluted separately by each subnetworks and yields its own independent output. Grouped convolutions are propagated until the end to ensure that gradients stay independent between sub-neural networks. Other operations such as Batch Normalization (Ioffe & Szegedy, 2015) can be applied directly as long as they can be grouped or have independent actions on each channel. Figure 4a illustrates the mask used to code Packed-Ensembles in the case where $M = 2$.

Finally, **(b)** and **(c)** are also equivalent. It is indeed possible to replace the "rearrange" operation and the first grouped convolution by a standard convolution if the same images are to be provided at the same time to all the subnetworks. We confirm in the Appendix F that this procedure is not detrimental to the performance of the ensemble and we take advantage from this property to provide this final optimization and simplification. Figure 4b shows the mask used to code Packed-Ensembles in the case where $M = 2$ and $\gamma = 2$.

4 Experiments

To validate the performance of our method, we conduct experiments on classification tasks and we measure the effect of the parameters α and γ. Regression tasks are detailed in Appendix N.

[1] See https://einops.rocks/api/rearrange/

4.1 DATASETS AND ARCHITECTURES

First, we demonstrate the efficiency of Packed-Ensembles on CIFAR-10 and CIFAR-100 (Krizhevsky, 2009), showing how the method adapts to different data complexities. As we propose to replace a single model architecture with several subnetworks, we study the behavior of Packed-Ensembles on architectures with various sizes: ResNet-18 and ResNet-50 (He et al., 2016), and Wide ResNet28-10 (Zagoruyko & Komodakis, 2016).We compare it against Deep Ensembles (Lakshminarayanan et al., 2017) and two other approximated ensembles from the literature: BatchEnsemble (Wen et al., 2019) and MIMO (Havasi et al., 2020).

Secondly, we report our results for Packed-Ensembles on ImageNet, which we compare against Deep Ensembles. We run experiments with Resnet-50 and Resnet-50x4. All training runs are started from scratch.

4.1.1 METRICS

We evaluate the overall performance of the models in classification tasks using the accuracy (Acc) in % and Negative Log-Likelihood (NLL). We choose the classical Expected Calibration Error (ECE) (Naeini et al., 2015) for the calibration of uncertainties[2], and measure the quality of the OOD detection using the Areas Under the Precision/Recall curve (AUPR) and Under the operating Curve (AUC), as well as the False Positive Rate at 95% recall (FPR95), all expressed in % similarly to Hendrycks & Gimpel (2017)

We use accuracy as the validation criterion (i.e., the final trained model is the one with the highest accuracy). During inference, we average the softmax probabilities of all subnetworks and consider the index of the maximum in the output vector to be the predicted class of the ensemble. We define the prediction confidence as the value of this maximum (also called maximum softmax probability).

For the out-of-distribution detection tasks on CIFAR-10 and CIFAR-100, we use the SVHN dataset (Netzer et al., 2011) as an out-of-distribution dataset and transform the initial classification problem into a binary classification between in-distribution (ID) data and OOD data using the maximum softmax probability as OOD criterion. We discuss the different OOD criteria in appendix E. For ImageNet, we use two out-of-distribution datasets: ImageNet-O (IO) (Hendrycks et al., 2021b) and Texture (T) (Wang et al., 2022) and use the Mutual Information (**MI**) as criterion for the ensembles techniques (see Appendix E for **MI**) and the maximum softmax probability for the single model. To measure the robustness under distribution shift, we use ImageNet-R (Hendrycks et al., 2021a) and evaluate the Accuracy, ECE, and NLL, denoted respectively rAcc, rECE, and rNLL on this dataset.

We implement our models using the PyTorch-Lightning framework built on top of PyTorch. Both are open-source Python frameworks. Appendix B and Table 4 detail the hyper-parameters used in our experiments across architectures and datasets. Most trainings are completed on a single Nvidia RTX 3090 except on ImageNet for which we use 4 to 8 Nvidia A100-80GB.

4.1.2 RESULTS

Table 1 presents the average performance over five runs on the classification task using the hyper-parameters on Table 4. We showcase that in the particular setting of $\alpha = 2$ and $\gamma = 2$, Packed-Ensembles yields similar results to Deep Ensembles while having a lower memory cost than the single model. On CIFAR-10 (Krizhevsky, 2009), we notice that its relative performance compared to Deep Ensembles seems to increase as the original architecture gets bigger. For ResNet-18, the method matches Deep Ensembles on OOD detection metrics but shows slightly worse performance on the others. Using ResNet-50, both models seem to perform equivalently, while Packed-Ensembles slightly outperforms Deep Ensembles in terms of classification performance with Wide ResNet28-10.

On CIFAR-100, Packed-Ensembles is outperformed by Deep Ensembles on ResNet-18. Yet, we argue that ResNet-18 architecture does not have enough representation capacity to be divided into subnetworks for CIFAR-100. Indeed, when we look at the results of ResNet-50, we can see that

[2]Note that the benchmark conducted at https://github.com/google/uncertainty-baselines uses only ECE to measure the calibration quality of the models

Table 1: **Performance comparison (averaged over 5 runs) on CIFAR-10/100 using ResNet-18 (R18), ResNet-50 (R50), and Wide ResNet28-10 (WR) architectures.** All ensembles have $M = 4$ subnetworks, we highlight the best performances in bold. For our method, we consider $\alpha = \gamma = 2$, except for WR where $\gamma = 1$. *Mult-Adds* corresponds to the inference cost, i.e., the number of giga multiply-add operations for a forward pass which is estimated with Torchinfo.

Method	Data	Net	Acc ↑	NLL ↓	ECE ↓	AUPR ↑	AUC ↑	FPR95 ↓	Params (M) ↓	Mult-Adds ↓
Single Model	C10	R18	94.0	0.238	0.035	94.0	89.7	33.8	11.17	0.56
BatchEnsemble	C10	R18	92.9	0.257	0.031	92.4	87.8	32.1	11.21	2.22
MIMO ($\rho = 1$)	C10	R18	94.0	0.228	0.033	94.4	90.2	28.6	11.19	2.24
Masksembles	C10	R18	94.0	0.188	0.009	93.6	89.5	27.8	11.19	2.22
Packed-Ensembles	C10	R18	94.3	0.178	**0.007**	**94.7**	**91.3**	23.2	**8.18**	**0.48**
Deep Ensembles	C10	R18	**95.1**	**0.156**	0.008	**94.7**	**91.3**	**18.0**	44.70	2.22
Single Model	C10	R50	95.1	0.211	0.031	95.2	91.9	23.6	23.52	1.30
BatchEnsemble	C10	R50	93.9	0.255	0.033	94.7	91.3	20.1	23.63	5.19
MIMO ($\rho = 1$)	C10	R50	95.4	0.197	0.030	95.1	90.8	26.0	23.59	5.22
Masksembles	C10	R50	95.3	0.175	0.019	95.7	92.2	22.1	23.6	5.19
Packed-Ensembles	C10	R50	95.9	0.137	**0.008**	**97.3**	**95.2**	**14.4**	**14.55**	**1.00**
Deep Ensembles	C10	R50	**96.0**	**0.136**	**0.008**	97.0	94.7	15.5	94.08	5.19
Single Model	C10	WR	95.4	0.200	0.029	96.1	93.2	20.4	36.49	5.95
BatchEnsemble	C10	WR	95.6	0.206	0.027	95.5	92.5	22.1	36.59	23.81
MIMO ($\rho = 1$)	C10	WR	94.7	0.234	0.034	94.9	90.6	30.9	36.51	23.82
Masksembles	C10	WR	94.0	0.186	0.016	97.2	95.0	14.5	36.53	23.82
Packed-Ensembles	C10	WR	**96.2**	**0.133**	**0.009**	**98.1**	**96.5**	**11.1**	**19.35**	**4.06**
Deep Ensembles	C10	WR	95.8	0.143	0.013	97.8	96.0	12.5	145.96	23.82
Single Model	C100	R18	75.1	1.016	0.093	88.6	79.5	55.0	11.22	0.56
BatchEnsemble	C100	R18	71.2	1.236	0.116	86.0	75.4	60.2	11.25	2.22
MIMO ($\rho = 1$)	C100	R18	75.3	0.962	0.069	89.2	80.7	52.9	11.36	2.24
Masksembles	C100	R18	74.2	1.054	0.061	86.7	76.3	59.8	11.19	2.22
Packed-Ensembles	C100	R18	76.4	0.858	0.041	88.7	79.8	57.1	**8.18**	**0.48**
Deep Ensembles	C100	R18	**78.2**	**0.800**	**0.018**	**90.2**	**82.4**	**50.5**	44.88	2.22
Single Model	C100	R50	78.3	0.905	0.089	87.4	77.9	57.6	23.70	1.30
BatchEnsemble	C100	R50	66.6	1.788	0.182	85.2	74.6	60.6	23.81	5.19
MIMO ($\rho = 1$)	C100	R50	79.0	0.876	0.079	87.5	76.9	64.7	24.33	5.22
Masksembles	C100	R50	78.5	0.832	0.046	90.3	81.9	52.3	23.6	5.19
Packed-Ensembles	C100	R50	**81.2**	**0.703**	**0.020**	**90.0**	**81.7**	56.5	**15.55**	**1.00**
Deep Ensembles	C100	R50	80.9	0.713	0.026	89.2	80.8	**52.5**	94.82	5.19
Single Model	C100	WR	80.3	0.963	0.156	81.0	64.2	80.1	**36.49**	**5.95**
BatchEnsemble	C100	WR	82.3	0.835	0.130	**88.1**	**78.2**	**69.8**	36.59	23.81
MIMO ($\rho = 1$)	C100	WR	80.2	0.822	**0.028**	84.9	72.0	72.8	36.51	23.82
Masksembles	C100	WR	74.4	0.937	0.063	76.1	60.0	75.1	36.53	23.82
Packed-Ensembles	C100	WR	**83.9**	**0.678**	0.089	86.2	73.2	80.7	36.62	**5.96**
Deep Ensembles	C100	WR	82.5	0.903	0.229	81.6	67.9	71.3	145.96	23.82

Packed-Ensembles has better results than Deep Ensembles. This demonstrates that, given a sufficiently large network, Packed-Ensembles is able to match Deep Ensembles with only 16% of its parameters. In appendix F, we discuss the influence of the representation capacity.

Based on the results in Table 2, we can conclude that Packed-Ensembles improves uncertainty quantification for OOD and distribution shift on ImageNet compared to Deep Ensembles and Single model and that it improves the accuracy with a moderate training and inference cost.

4.1.3 Study on the parameters α and γ

Table 1 reports results only for $\alpha = 2$ and $\gamma = 2$, however depending on the task, the architecture used and the available memory, one might want to tune those parameters to fit more precisely its needs. Figures 6 and 7 showcase the performance evolution of the Packed-Ensembles as the α values varies.

5 Discussions

We have shown that Packed-Ensembles has interesting properties, especially by providing a similar quality of Uncertainty Quantification as Deep Ensemble while using a reduced architecture and computing cost. Several questions can be raised and we conducted some studies, detailed in the Appendix section, in order to bring possible answers.

Table 2: Performance comparison on ImageNet (Deng et al., 2009) with ResNet-50 (R50) and ResNet-50x4 (R50x4) architectures. All ensembles have $M = 4$ subnetworks and $\gamma = 1$. We highlight the best performances in bold. For OOD tasks, we use ImageNet-O (IO) (Hendrycks et al., 2021b) and Texture (T) (Wang et al., 2022), for distribution shift we use ImageNet-R (Hendrycks et al., 2021a). We choose the results corresponding to the best OOD criterion among Max Probability and Mutual Information.

Method	Net	Acc	ECE	AUPR (T)	AUC (T)	FPR95 (T)	AUPR (IO)	AUC (IO)	FPR95 (IO)	rAcc	rNLL	rECE
Single Model	R50	77.8	0.1206	18.0	80.9	68.6	3.6	50.8	90.8	23.5	5.187	0.0822
BatchEnsemble	R50	65.7	0.1221	10.6	76.5	70.7	3.5	49.5	90.3	16.5	5.830	0.0658
MIMO	R50	77.6	0.1465	18.4	81.6	66.8	3.7	52.2	90.6	23.4	5.115	0.0585
Masksembles	R50	73.6	0.2093	13.6	79.7	68.3	3.31	47.7	87.7	21.2	5.139	0.0107
Packed-Ensembles $\alpha = 3$	R50	77.9	0.1796	**35.1**	**88.2**	**43.7**	**9.9**	**68.4**	**80.9**	23.8	4.978	0.0221
Deep Ensembles	R50	**79.2**	0.2326	19.6	83.4	62.1	3.7	52.5	85.5	**24.9**	**4.879**	**0.0182**
Single Model	R50×4	80.2	0.0221	20.5	82.6	63.9	4.9	60.2	87.4	26.0	5.190	0.1721
BatchEnsemble	R50×4	/	/	/	/	/	/	/	/	/	/	/
MIMO	R50×4	80.3	0.0150	19.3	82.5	66.1	4.9	60.7	86.4	25.8	5.278	0.1886
Masksembles	R50×4	/	/	/	/	/	/	/	/	/	/	/
Packed-Ensembles $\alpha = 2$	R50×4	81.3	0.1034	**34.6**	**88.1**	**50.3**	**9.6**	**69.9**	**79.2**	26.6	4.848	**0.0750**
Deep Ensembles	R50×4	**82.1**	0.0534	23.0	85.6	58.1	5.0	62.7	81.9	**28.2**	**4.789**	0.1048

Discussion on the sparsity As described in section 3, one could interpret Packed-Ensembles as leveraging group convolutions to approximate Deep Ensembles with a mask operation applied to some components. In appendix C, by using a simplified model, we are able to bound the approximation error based on the Kullback-Leibler divergence between the Deep Ensemble and its pruned version. This bound depends on the density of ones in the mask p, and more specifically, depends on the terms $(p-1)$, $p(p-1)$ and $(p-1)^2/p^2$. By manipulating these terms, corresponding to modifying the number of estimators M, the number of groups γ and the dilatation faction α, we could theoretically be able to control the approximation error.

On the sources of stochasticity Diversity is essential in ensembles and is usually obtained by exploiting two main sources of stochasticity: the random initialization of the model's parameters and the shuffling of the batches. A last source of stochasticity is introduced during the training phase by the non-deterministic behaviour of the backward algorithms. In Appendix F, we study the function space diversities which arise from every possible combination of these sources. It follows that only one of these sources is often sufficient to generate diversity, and no peculiar pattern seem to emerge to predict the best combination. Especially, we highlight that even the only use of non-deterministic algorithms introduces enough diversity between each estimator of the ensemble.

Ablation study We perform ablations studies in order to assess the impact of the parameters M, γ, and α on the performances of Packed-Ensembles. The details of this study are given in Appendix D. No explicit behavior appears from the results we obtained. A trend seems to show that a higher number of subnetworks is helpful in obtaining a better OOD detection, but the improvement in terms of AUPR is not significant.

Training speed Depending on the chosen hyperparameters α, M, and γ, PE may have fewer parameters than the single model, as shown in Table 1. This translates into an expected lower number of operations. However, a study of the training and inference speeds, developed in Appendix H, shows that using PE-(2,4,1) does not significantly increase the training and testing times compared to the single model while improving accuracy and uncertainty quantification performances. Yet, this also hints that the actual speedup is not optimal despite the important acceleration offered by 16 bits floating points.

OOD criteria The criterion for discriminating OOD elements is based on the maximum softmax probability. However, this criterion is not unique, and others can be used, such as the Mutual Information, the maximum logit, or the Shannon entropy of the mean prediction. Although no relationship is expected between this criterion and the method of Packed-Ensembles, we obtained different performances in OOD detection according to the selected criterion. The results are detailed in appendix E, and they show that an approach based on the maximum logit seems to give the best results in detecting OOD. It should be noted that this discussion focuses primarily on CIFAR-100 and that the notion of OOD depends on the training distribution. Such a discussion does not necessarily gen-

eralize to all datasets. Indeed, preliminary results have shown that Mutual information outperforms the other criteria for our method applied to the ImageNet dataset.

6 Related Work

Ensembles and uncertainty quantification. Bayesian Neural Networks (BNNs) (MacKay, 1992; Neal, 1995) are the cornerstone and main source of inspiration for uncertainty quantification in deep learning. In spite of the progress enabled by variational inference (Jordan et al., 1999; Blundell et al., 2015), BNNs remain difficult to scale and train for large DNN architectures (Dusenberry et al., 2020). DE (Lakshminarayanan et al., 2017) arise as a practical and efficient instance of BNNs, coarsely but effectively approximating the posterior distribution of weights (Wilson & Izmailov, 2020). DE are currently the best performing approach for both predictive performance and uncertainty estimation (Ovadia et al., 2019; Gustafsson et al., 2020).

Efficient ensembles. The appealing properties in performance and diversity of DE (Fort et al., 2019), but also their major downside related to computational cost have inspired a large cohort of approaches aiming to mitigate it. BatchEnsemble (Wen et al., 2019) spawns an ensemble at each layer thanks to an efficient parameterization of subnetwork specific parameters trained in parallel. MIMO (Havasi et al., 2020) shows that a large network can encapsulate multiple subnetworks using a multi-input multi-output configuration. A single network can be used in ensemble-mode by disabling different sub-sets of weights at each forward pass (Gal & Ghahramani, 2016; Durasov et al., 2021). Liu et al. (2022) leverage the sparse networks training algorithm of Mocanu et al. (2018) to produce ensembles of sparse networks. Ensembles can be computed from a single training run by collecting intermediate model checkpoints (Huang et al., 2017; Garipov et al., 2018), by computing the posterior distribution of the weights by tracking their trajectory during training (Maddox et al., 2019; Franchi et al., 2020), by ensembling predictions over multiple augmentations of the input sample (Ashukha et al., 2020). However most of these approaches require multiple forward passes.

Neural network compression. The most intuitive approach for reducing the size of a model is to employ DNNs that are memory-efficient by-design, relying on e.g., channel shuffling (Zhang & Yang, 2021), point-wise convolutional filters (Liang et al., 2021), weight sharing (Bender et al., 2020), or a combination of them. Some of the most popular architectures the leverage such models are: SqueezeNet (Iandola et al., 2016), ShuffleNet (Zhang et al., 2018b), MobileNet-v3 (Howard et al., 2019). Some approaches conduct automatic model size reduction, e.g., network sparsification (Molchanov et al., 2017; Louizos et al., 2018; Frankle & Carbin, 2018; Tartaglione et al., 2022). These approaches aim at removing as many parameters as possible from the model to improve memory and computation efficiency. Similarly, quantization approaches (Han et al., 2016; Lin et al., 2017) avoid or minimize the computation cost of floating point operation and optimize the use of the much more efficient integer computation.

Grouped convolutions. To the best of our knowledge, grouped convolutions (group of convolutions) were introduced by Krizhevsky et al. (2012). Enabling the computation of several independent convolutions in parallel, they developed the idea of running a single model on multiple GPU devices. Xie et al. (2017) demonstrate that using grouped convolutions lead to accuracy improvements and model complexity reduction. So far, grouped convolutions have been used primarily for computational efficiency. Here, we re-purpose them for delineating multiple subnetworks within a network and for efficiently training an ensemble of such subnetworks.

7 Conclusions

We propose a new ensemble framework: Packed-Ensembles, which is able to approximate Deep Ensembles in terms of uncertainty quantification and accuracy. Our work provides several new findings: first, we show that ensembling independent small neural networks can be equivalent to ensembling independent deep neural networks. Secondly, we demonstrate that not all sources of diversity are needed to improve the diversity of the ensemble. Thirdly, we show that Packed-Ensembles is more stable than a single DNN. Fourthly, we highlight that there is a trade-off between the accuracy and the size of the parameters, and Packed-Ensembles allows us to have flexible and efficient ensembling. In the future, we intend to explore Packed-Ensembles for more complex tasks.

8 REPRODUCIBILITY

Alongside this paper, we provide the source code of Packed-Ensembles layers. Moreover, two notebooks enable the training of Packed-Ensembles based on the ResNet-50 architecture, both on CIFAR-10 and CIFAR-100 (public datasets). To ensure reproducibility, we report the performance given a specific random seed with a deterministic training process. Furthermore, it should be noted that the source code contains two PyTorch Module classes to efficiently produce Packed-Ensembles. The idea would indeed be to release a Python package to provide easier access to Packed-Ensembles layers. A readme file at the root of the project details how to install and run experiments. In addition, we showcase how to get Packed-Ensembles from LeNet (LeCun et al., 1998).

9 ETHICS

The purpose of this paper is to provide a method for a better estimation of Deep Learning model uncertainty. Nevertheless, we acknowledge their limitation, which could become critical when applied to safety systems for instance. While this work aims to improve the reliability of Deep Neural Networks, this approach is not ready for deployment in safety-critical systems. We show the limitations of our approach in several experiments. Many more validation and verification steps would be need before real-world deployment to ensure robustness to various unknown situations, corner-cases, adversarial attacks, biases.

REFERENCES

Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *ICLR*, 2020. 9

William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *CVPR*, 2018. 20

Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V. Le. Can weight sharing outperform random architecture search? An investigation with TuNAS. In *CVPR*, 2020. 9

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, 2015. 9

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR*, 2020. 16

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 8, 24

Thomas G Dietterich. Ensemble methods in machine learning. In *IWMCS*, 2000. 2

Nikita Durasov, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Masksembles for uncertainty estimation. In *CVPR*, 2021. 2, 9, 17

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In *ICML*, 2020. 9

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019. 2, 9

Gianni Franchi, Andrei Bursuc, Emanuel Aldea, Séverine Dubuisson, and Isabelle Bloch. Tradi: Tracking deep neural network weight distributions. In *ECCV*, 2020. 2, 9

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018. 9, 21

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016. 2, 9, 24

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018. 2, 9

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017. 1

Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. Evaluating scalable bayesian deep learning methods for robust computer vision. In *CVPR Workshops*, 2020. 2, 9

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 9

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *ICLR*, 2020. 2, 6, 9

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6, 16

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019. 1

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019. 22, 23

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017. 6

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *CVPR*, 2021a. 6, 8

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *CVPR*, 2021b. 6, 8

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*, 2015. 24

Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3. In *ICCV*, 2019. 9

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get M for free. In *ICLR*, 2017. 2, 9

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 9

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 1999. 9

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NeurIPS*, 2017. 24

Dan Kondratyuk, Mingxing Tan, Matthew Brown, and Boqing Gong. When ensembling smaller models is more efficient than single large models. *arXiv preprint arXiv:2005.00570*, 2020. 2

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, MIT, 2009. 6, 22

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 2, 9

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, 2017. 1, 3, 6, 9, 19, 24

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989. 2

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 10

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. 2

Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J. Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, Michael Sokolsky, Ganymed Stanek, David Stavens, Alex Teichman, Moritz Werling, and Sebastian Thrun. Towards fully autonomous driving: Systems and algorithms. In *IV*, 2011. 1

Feng Liang, Zhichao Tian, M. Dong, Shuting Cheng, Li Sun, Hai Helen Li, Yiran Chen, and Guohe Zhang. Efficient neural network using pointwise convolution kernels with linear phase constraint. *Neurocomputing*, 2021. 9

Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *NeurIPS*, 2017. 9

Shiwei Liu, Tianlong Chen, Zahra Atashgahi, Xiaohan Chen, Ghada Sokar, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Deep ensembling with no overhead for either training or testing: The all-round blessings of dynamic sparsity. In *ICLR*, 2022. 9

Ekaterina Lobacheva, Nadezhda Chirkova, Maxim Kodryan, and Dmitry Vetrov. On power laws in deep ensembles. In *NeurIPS*, 2020. 2

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018. 9

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 1992. 9

Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. In *NeurIPS*, 2019. 2, 9

Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *IJCAI*, 2017. 1

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 2018. 9

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, 2017. 9

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *AAAI*, 2015. 6

Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. A modern take on the bias-variance tradeoff in neural networks. *arXiv preprint arXiv:1810.08591*, 2018. 23

Radford M Neal. Bayesian learning for neural networks. *PhD thesis, University of Toronto*, 1995. 9

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshops*, 2011. 6

A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, 2015. 1

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? Uncovering how neural network representations vary with width and depth. In *ICLR*, 2020. 21

D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *ICNN*, 1994. 24

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *NeurIPS*, 2019. 2, 9, 22

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 15

Michael P Perrone and Leon N Cooper. When networks disagree: Ensemble methods for hybrid neural networks. Technical report, Brown University, 1992. 2

Alexandre Ramé, Rémy Sun, and Matthieu Cord. Mixmo: Mixing multiple inputs for multiple outputs via deep subnetworks. In *ICCV*, 2021. 2

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 16

Enzo Tartaglione, Andrea Bragagnolo, Attilio Fiandrotti, and Marco Grangetto. Loss-based sensitivity regularization: towards deep sparse neural networks. *Neural Networks*, 2022. 9

Torchinfo. Torchinfo. https://github.com/TylerYep/torchinfo. Version: 1.7.1. 7, 24

Haoqi Wang, Zhizhong Li, Litong Feng, and Wayne Zhang. ViM: Out-of-distribution with virtual-logit matching. In *CVPR*, 2022. 6, 8

Yeming Wen, Dustin Tran, and Jimmy Ba. BatchEnsemble: an alternative approach to efficient ensemble and lifelong learning. In *ICLR*, 2019. 2, 6, 9

Ross Wightman, Hugo Touvron, and Herve Jegou. Resnet strikes back: An improved training procedure in timm. In *NeurIPS 2021 - Workshop ImageNet PPF*, 2021. 16

Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *NeurIPS*, 2020. 4, 9

Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 3, 9, 24

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *CVPR*, 2019. 16

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 6

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018a. 16

Qing-Long Zhang and Yubin Yang. SA-Net: Shuffle attention for deep convolutional neural networks. In *ICASSP*, 2021. 9

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018b. 9

TABLE OF CONTENTS

A Notations **15**

B Implementation details **15**

B.1 General Considerations 15

B.2 Masksembles 16

C Discussion on the Sparsity **17**

D Ablation Study **19**

E Discussion about OOD criteria **19**

F Discussion about the sources of stochasticity **20**

G Discussion about the subnetworks **21**

H Discussion about the training velocity **22**

I Distribution shift **22**

J Stabilization of the performance **23**

K On the Equivalence between sequential training and Packed-Ensembles **23**

L Using groups is not sufficient to provide equivalent results to Packed-Ensembles **24**

M Efficiency of the networks trained on ImageNet **24**

N Regression **24**

A NOTATIONS

We summarize the main notations used in the paper in Table 3.

B IMPLEMENTATION DETAILS

B.1 GENERAL CONSIDERATIONS

Our code is implemented in PyTorch (Paszke et al., 2019) using the PyTorch Lightning framework. The code will be made publicly available after the anonymity period.

To ensure that the layers convey sufficient information and are not weakened by groups, we have chosen to set a constant minimum number of channels per group to 64 for all experiments presented in the paper. Experiments in which this minimum could play an important part and lead to confusions (see for instance PE-$(1, 4, 4)$ in Table 5) have not been conducted.

Table 3: **Summary of the main notations of the paper.**

Notations	**Meaning**
$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{\lvert\mathcal{D}\rvert}$	The set of $\lvert\mathcal{D}\rvert$ data samples and the corresponding labels
j, m, L	The index of the current layer, the current subnetwork, and the number of layers
$\mathbf{z}^j$	The preactivation feature map and output of the layer $(j-1)$/input of layer j
ϕ	The activation function (considered constant throughout the network)
$\mathbf{h}^j$	The feature map and output of layer j, $\mathbf{h}^j = \phi(\mathbf{z}^j)$
H_j, W_j	The height and width of the feature maps and output of layer $j-1$
C_j	The number of channels of the feature maps and output of layer $j-1$
n_j	The number of parameters of layer j
B	The batch size of the training procedure
mask_m^j	The mask corresponding to the layer j of the subnetwork m
$\lfloor\cdot\rfloor$	The integer part function
$\star$, $\circledast$, $\circ$	The 2D cross-correlation, the convolution and the Hadamard product
s_j	The size of the kernel of the layer j
M	The number of subnetworks in an ensemble
$\hat{\mathbf{y}}_i^m$	The prediction of the subnetwork number m concerning the input $\mathbf{x}_i$
$\hat{\mathbf{y}}_i$	The prediction of the ensemble concerning the input $\mathbf{x}_i$
α	The width-augmentation factor of Packed-Ensembles
γ	The number of subgroups of Packed-Ensembles
$\boldsymbol{\theta}_{\alpha,m}$	The set of weights of the subnetwork m with a width factor α
$\boldsymbol{\omega}_{\alpha,\gamma}^j$	The weights of layer j with γ groups and a width factor α

Table 4: **Hyperparameters for image classification.**

Dataset	**Networks**	max_epochs	batch_size	lr	momentum	weight_decay	lr_gamma	milestones	data augmentation
C10	R18	75	128	0.05	0.9	5e-4	0.1	25, 50	HFlip
C10	R50	200	128	0.1	0.9	5e-4	0.2	60, 120, 160	HFlip
C10	WR28-10	200	128	0.1	0.9	5e-4	0.2	60, 120, 160	HFlip
C100	R18	75	128	0.05	0.9	1e-4	0.2	25, 50	HFlip
C100	R50	200	128	0.1	0.9	5e-4	0.2	60, 120, 160	HFlip
C100	WR28-10	200	128	0.1	0.9	5e-4	0.2	60, 120, 160	Medium

Table 4 summarizes all the hyperparameters used in the paper for CIFAR-10 and CIFAR-100. HFlip denotes the classic horizontal flip. The BatchEnsemble based on ResNet-50 uses a smaller learning rate of 0.08 instead of 0.1 for stability. The medium data augmentation corresponds to a combination of mixup (Zhang et al., 2018a) and cutmix (Yun et al., 2019) using timm's augmentation classes[3], with alphas respectively 0.5 and 0.2. We also use RandAugment (Cubuk et al., 2020) with $m = 9$, $n = 2$, and $mstd = 1$ and a label-smoothing (Szegedy et al., 2016) of intensity 0.1 in this case.

For ImageNet, we use the A3 procedure from Wightman et al. (2021) for all models.

Please note that the hyperparameters of the training procedures have not been optimized for our method and have been taken directly from the literature (He et al., 2016; Wightman et al., 2021).

B.2 MASKSEMBLES

[3]available at github.com/rwightman/pytorch-image-models

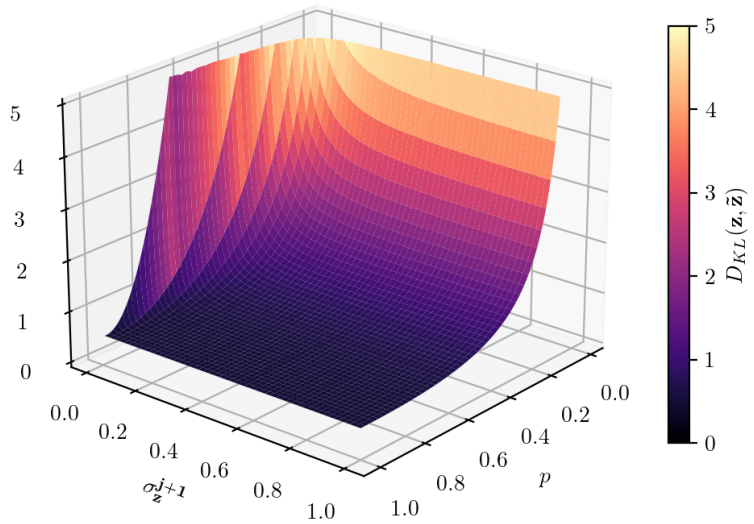


Figure 5: KL divergence for different values of p and $\boldsymbol{\sigma_z^{j+1}}$, with $\boldsymbol{\mu^j}(k) = 0.1\ \forall j, k$ and $w^j(c, k) = 0.1\ \forall j, c, k$.

Concerning Masksembles (Durasov et al., 2021), we use the code proposed by the authors[4]. We modified the mask generation function using binary search as proposed by the authors, since it was unable to build masks for ResNet50x4. We note that the code implies performing batch repeats at the start the forward passes. All the results regarding this technique are therefore computed with this specification. The ResNet implementations are built using Masksemble2D layers with M=4 and a scale factor of 2 after each convolutions.

C Discussion on the Sparsity

In this section, we are going to provide an estimation of the expected distance between a dense, fully-connected layer, and a sparse one. For simplicity, we are here assuming to operate with a fully-connected layer. First, let us write our first proposition:

Proposition C.1. *Given a fully connected layer $j+1$ defined by:*

$$\mathbf{z}^{j+1}(c) = \sum_{k=0}^{C_j - 1} \boldsymbol{\omega}^j(c, k)\mathbf{h}^j(k) \tag{6}$$

and its approximation defined by:

$$\tilde{\mathbf{z}}^{j+1}(c) = \sum_{k=0}^{C_j - 1} (\boldsymbol{\omega}^j(c, k) mask^j(k, c))\mathbf{h}^j(k) \tag{7}$$

Under the assumption that the j follows a Gaussian distribution $\boldsymbol{h}^j \sim \mathcal{N}(\boldsymbol{\mu^j}, \Sigma^j)$, where Σ^j is the covariance matrix and $\boldsymbol{\mu}^j$ the mean vector, the Kullback–Leibler divergence between the layer and its approximation is bounded by:

$$D_{\mathrm{KL}}(\mathbf{z}, \tilde{\mathbf{z}})(c) \leq \frac{1}{2}\left\{p + \frac{1}{p} - 2 + \frac{p \cdot (1-p)\sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)^2 \boldsymbol{\mu^j}(k)^2}{(\boldsymbol{\sigma_z^{j+1}})^2(c)} + \frac{\left[(1-p) \times \boldsymbol{\mu_z^{j+1}}(c)\right]^2}{p(\boldsymbol{\sigma_z^{j+1}})^2(c)}\right\} \tag{8}$$

where $p \in [0; 1]$ is the fraction of the parameters of $\mathbf{z}^{j+1}(c)$ included in the approximation $\tilde{\mathbf{z}}^{j+1}(c)$.

[4]available at github.com/nikitadurasov/masksembles

A plot for (8) is provided in Figure 5.

Proof. To prove Prop. C.1, we state first that, since $\mathbf{h}^j(k)$ follows a Gaussian distribution, and considering that $\boldsymbol{\omega}^j$ at inference time is constant and linearly-combined with a gaussian random variable, $\boldsymbol{z}^{j+1}$ will be as well gaussian-distributed.
From the property of linearity of expectations, we know that the mean for $\mathbf{z}^{j+1}(c)$ is:

$$\boldsymbol{\mu}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) = \sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)\boldsymbol{\mu}^{\boldsymbol{j}}(k) \tag{9}$$

and the variance is:

$$(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) = \sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)\left[\boldsymbol{\omega}^j(c,k)\Sigma(k,k) + 2\sum_{k'<k} \boldsymbol{\omega}^j(c,k')\Sigma(k',k)\right]. \tag{10}$$

If we assume $\Sigma(i,k) = 0\ \forall\ i \neq k$, (10) simplifies into:

$$(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) = \sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)^2\Sigma(k,k). \tag{11}$$

Let us now consider the case with the mask, similarly as presented in (3):

$$\tilde{\mathbf{z}}^{j+1}(c) = \sum_{k=0}^{C_j-1} (\boldsymbol{\omega}^j(c,k)\text{mask}^j(k,c))\mathbf{h}^j(k) \tag{12}$$

We assume here that $\text{mask}^j \sim Ber(p)$ where p is the probability of the Bernoulli (or 1-pruning rate). In the limit of large C_j, we know that $\tilde{\mathbf{z}}^{j+1}(c)$ follows a Gaussian distribution defined by a mean and a variance equal to:

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) = \sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)\boldsymbol{\mu}^{\boldsymbol{j}}(k)p \tag{13}$$

$$(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) = \sum_{k=0}^{C_j-1} p\boldsymbol{\omega}^j(c,k)^2\left[\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2(1-p) + \Sigma(k,k)\right] \tag{14}$$

Hence, we have:

$$\tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) = p \times \boldsymbol{\mu}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) \tag{15}$$

$$(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) = p\left[(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + (1-p)\sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2\right] \tag{16}$$

In order to assess the dissimilarity between $\mathbf{z}$ and $\tilde{\mathbf{z}}$, we can write the Kullback–Leibler divergence:

$$D_{\text{KL}}(\mathbf{z},\tilde{\mathbf{z}})(c) = \frac{1}{2}\left\{\log\left[\frac{(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)}{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)}\right] + \frac{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + \left[\boldsymbol{\mu}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) - \tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)} - 1\right\} \tag{17}$$

Straightforwardly we can write the inequality:

$$D_{\text{KL}}(\mathbf{z},\tilde{\mathbf{z}})(c) \leq \frac{1}{2}\left\{\frac{(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)}{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)} - 1 + \frac{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + \left[\boldsymbol{\mu}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) - \tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{(\tilde{\boldsymbol{\sigma}}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)} - 1\right\} \tag{18}$$

According to (16) we can write:

$$D_{\text{KL}}(\mathbf{z},\tilde{\mathbf{z}})(c) \leq \frac{1}{2}\left\{\frac{p\left[(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + (1-p)\sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2\right]}{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c)} - 1 + \right.$$
$$\left. + \frac{(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + \left[\boldsymbol{\mu}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c) - \tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{p\left[(\boldsymbol{\sigma}_{\boldsymbol{z}}^{\boldsymbol{j+1}})^2(c) + (1-p)\sum_{k=0}^{C_j-1} \boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2\right]} - 1\right\} \tag{19}$$

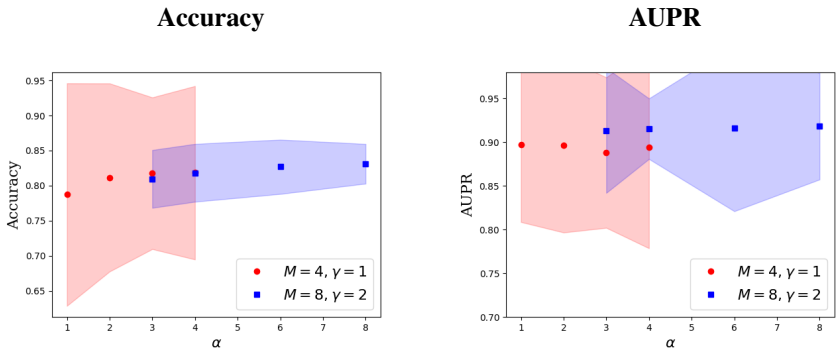


Figure 6: Accuracy and AUPR of Packed-Ensembles with ResNet-50 on CIFAR-100 depending on α.

Table 5: **Performance (Acc / ECE / AUPR) of Packed-Ensembles for various α and γ values with ResNet-50 on CIFAR-100 and $M = 4$.**

γ \ α	1	2	3	4
1	0.7872 / 0.0165 / 0.8969	0.8116 / 0.0203 / 0.8966	0.8187 / 0.0201 / 0.8825	0.8183 / 0.0230 / 0.8939
2	0.7857 / 0.0185 / 0.9024	0.8103 / 0.0295 / 0.9115	0.8186 / 0.0197 / 0.9127	0.8242 / 0.0190 / 0.9088
4	/	0.8119 / 0.0180 / 0.9066	0.8182 / 0.0236 / 0.9140	0.8225 / 0.0226 / 0.9229

Since we know that $\frac{(\boldsymbol{\sigma_z^{j+1}})^2(c)+\left[\boldsymbol{\mu_z^{j+1}}(c)-\tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{p\left[(\boldsymbol{\sigma_z^{j+1}})^2(c)+(1-p)\sum_{k=0}^{C_j-1}\boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2\right]} \leq \frac{(\boldsymbol{\sigma_z^{j+1}})^2(c)+\left[\boldsymbol{\mu_z^{j+1}}(c)-\tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{p(\boldsymbol{\sigma_z^{j+1}})^2(c)}$ we can also write:

$$D_{\mathrm{KL}}(\mathbf{z},\tilde{\mathbf{z}})(c) \leq \frac{1}{2}\left\{p-1+\frac{p\cdot(1-p)\sum_{k=0}^{C_j-1}\boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2}{(\boldsymbol{\sigma_z^{j+1}})^2(c)}+\right.$$
$$\left.+\frac{(\boldsymbol{\sigma_z^{j+1}})^2(c)+\left[\boldsymbol{\mu_z^{j+1}}(c)-\tilde{\boldsymbol{\mu}}_{\boldsymbol{z}}^{\boldsymbol{j+1}}(c)\right]^2}{p(\boldsymbol{\sigma_z^{j+1}})^2(c)}-1\right\} \tag{20}$$

Finally, according to: (15)

$$D_{\mathrm{KL}}(\mathbf{z},\tilde{\mathbf{z}})(c) \leq \frac{1}{2}\left\{p+\frac{1}{p}-2+\frac{p\cdot(1-p)\sum_{k=0}^{C_j-1}\boldsymbol{\omega}^j(c,k)^2\boldsymbol{\mu}^{\boldsymbol{j}}(k)^2}{(\boldsymbol{\sigma_z^{j+1}})^2(c)}+\frac{\left[(1-p)\times\boldsymbol{\mu_z^{j+1}}(c)\right]^2}{p(\boldsymbol{\sigma_z^{j+1}})^2(c)}\right\}$$

finding back (8).

□

D Ablation Study

Our algorithm mainly depends on three hyperparameters. M represents the number of subnetworks in the ensemble, α controls the power of representation of the DNN, and γ is an extra parameter that controls the sparsity degree of the DNN. To evaluate the sensitivity of Packed-Ensembles to these parameters, we train 5 ResNet-50 on CIFAR-10 similarly to the protocol explained in section 4.1. Figures 6 and 7 show that the more we add subnetworks increasing M, the better the performances, in terms of accuracy and AUPR. We also note that the results are stable with γ. Moreover, we notice that the resulting accuracy tends to increase with α until it reaches a plateau. These statement are confirmed by the results in Table 5.

E Discussion about OOD criteria

Deep Ensembles (Lakshminarayanan et al., 2017) and Packed-Ensembles are ensembles of DNNs that can be used to quantify the uncertainty of the DNNs prediction. Similarly to Bayesian Neural

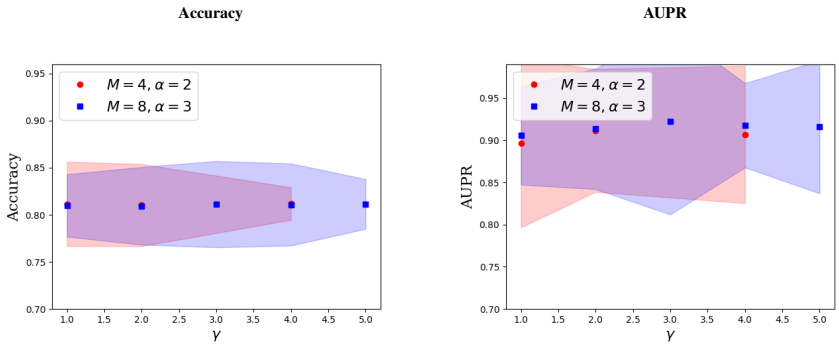


Figure 7: Accuracy and AUPR of Packed-Ensembles with ResNet-50 on CIFAR-100 depending on γ.

Table 6: Comparison of the effect of the different uncertainty criteria for OOD on CIFAR-100 with different sets of parameters for Packed-Ensembles.

Criterion	OOD eval	$\alpha = 2, \gamma = 1$ $M = 4$	$\alpha = 3, \gamma = 1$ $M = 8$	$\alpha = 4, \gamma = 2$ $M = 8$	$\alpha = 6, \gamma = 4$ $M = 8$	$\alpha = 8, \gamma = 1$ $M = 16$
MSP	AUPR (↑)	0.8952 ± 0.0132	0.9055 ± 0.0034	0.9153 ± 0.0012	0.9149 ± 0.0071	0.9141 ± 0.0057
ML	AUPR (↑)	**0.9183 ± 0.0098**	**0.9175 ± 0.0044**	**0.9285 ± 0.0012**	**0.9265 ± 0.0070**	**0.9268 ± 0.0068**
Ent.	AUPR (↑)	**0.9105 ± 0.0138**	**0.9152 ± 0.0035**	**0.9260 ± 0.0016**	**0.9237 ± 0.0066**	**0.9252 ± 0.0060**
MI	AUPR (↑)	**0.8649 ± 0.0061**	**0.9139 ± 0.0077**	**0.9157 ± 0.0072**	**0.9196 ± 0.0109**	**0.9245 ± 0.0091**
v	AUPR (↑)	0.8404 ± 0.0071	0.8746 ± 0.0056	0.8827 ± 0.0033	0.8842 ± 0.0102	0.8931 ± 0.0072
MSP	AUC (↑)	0.8056 ± 0.0260	0.8204 ± 0.0101	0.8408 ± 0.0033	0.8432 ± 0.0134	0.8387 ± 0.0094
ML	AUC (↑)	**0.8562 ± 0.0194**	**0.8421 ± 0.0115**	**0.8665 ± 0.0027**	**0.8621 ± 0.0144**	**0.8607 ± 0.0114**
Ent.	AUC (↑)	**0.8361 ± 0.0271**	**0.8427 ± 0.0095**	**0.8662 ± 0.0027**	**0.8617 ± 0.0136**	**0.8614 ± 0.0096**
MI	AUC (↑)	**0.7711 ± 0.0064**	**0.8312 ± 0.0135**	**0.8402 ± 0.0116**	**0.8468 ± 0.0163**	**0.8513 ± 0.0120**
v	AUC (↑)	0.7305 ± 0.0153	0.7799 ± 0.0129	0.7943 ± 0.0082	0.7999 ± 0.0166	0.8092 ± 0.0113

Network, one can take the softmax outputs of posterior predictive distribution, which define the **MSP** $= \max_{y_i}\{P(\mathbf{y}_i|\mathbf{x}, \mathcal{D})\}$. The MSP can also be used for classical DNN, yet in this case, we use the conditional likelihood instead of the posterior distribution.

One can also use the Maximum Logit (ML) as an uncertainty criterion and the entropy of the posterior predictive distribution as an uncertainty criterion, which is defined by **Ent.** $= \mathcal{H}(P(\mathbf{y}_i|\mathbf{x}, \mathcal{D}))$ with $\mathcal{H}$ being the entropy function. Another metric is the mutual information (MI) between two random variables, which is defined by: **MI** $= \mathcal{H}(P(\mathbf{y}_i|\mathbf{x}, \mathcal{D})) - \frac{1}{M}\sum_{m=0}^{M-1}\mathcal{H}(P(\mathbf{y}|\boldsymbol{\theta}_{\alpha,m}, \mathbf{x}))$. It represents a measure of the ensemble entropy, which is the entropy of the posterior minus the average entropy over predictions.

The last metric used in active learning is the variation ratio (Beluch et al., 2018) which measures the dispersion of a nominal variable, and is calculated as the proportion of predicted class labels that are not the modal class prediction. It is defined by: $\mathbf{v} = 1 - \frac{f_i}{M}$, where f_i is the number of predictions falling into the modal class category.

In Table 6, the results for the different metrics are reported. We note that **ML** seems to be the best metric to detect OOD. This metric is followed by **Ent.** and then **MI**. Note that **v**, which is widely used in active learning, does not seem to be effective in detecting OOD. This shows us that it is essential to use a good criterion in addition to good ensembling.

F Discussion about the sources of stochasticity

As written in the introduction of the paper, diversity is essential to the success of ensembling. Three main sources can induce weight diversity and therefore diversity in the function space. These sources are the initialization of the weights, the composition of the batches and the use of non-deterministic backpropagation algorithms [5]. On Table 7, we measure the mean performance over 5 experiments

[5] see https://docs.nvidia.com/deeplearning/cudnn/api/index.html

Table 7: Comparison of the performance wrt. the different sources of stochasticity - CIFAR-100. **ND** corresponds to the use of Non-deterministic backpropagation algorithms, **DI** to different initializations, and **DB** to different compositions of the batches

Stochasticity			ResNet-18				ResNet-50			
ND	**DI**	**DB**	**Acc** (↑)	**ECE** (↓)	**AUPR** (↑)	**MI**	**Acc** (↑)	**ECE** (↓)	**AUPR** (↑)	**MI**
-	-	-	71.7	0.0497	87.3	-	77.6	0.0825	89.2	-
✓	-	-	75.8	**0.0365**	**89.5**	0.1945	80.9	0.0179	90.2	0.1513
-	✓	-	76.2	0.0419	**89.5**	**0.2011**	81.0	0.0202	91.1	0.1524
-	-	✓	76.1	0.0434	88.7	0.1987	80.9	**0.0178**	90.8	0.1505
✓	✓	-	76.1	0.0424	88.7	0.1995	**81.2**	0.0210	**91.7**	**0.1584**
✓	-	✓	76.2	0.0433	88.9	0.2032	81.1	0.0200	90.4	0.1503
-	✓	✓	76.1	0.0437	89.2	0.1943	81.1	0.0186	90.9	0.1521
✓	✓	✓	**76.3**	0.0445	89.0	0.1954	81.1	0.0198	90.7	0.1534

in accuracy, NLL, calibration and OOD-Detection for all major configurations on CIFAR 100 for Packed-Ensembles. Results on Table 7 show that the use of non-deterministic algorithms can be sufficient to generate diversity. It was noted that this effect does not always appear depending on the selected architecture and the precision used (`float16`, `float32`). In practice, we use non-deterministic backpropagation algorithms as well as different inializations to ensure enough stochasticity but also for programming convenience and training speed. The accuracy, ECE and AUPR are given as a proxy of the quality of the diversity while the mutual information provides a quantification of this diversity. We confirm that using deterministic backpropagation algorithm, the same initialization and the same batches lead to the same weights and therefore the same values in the function space.

G Discussion about the subnetworks

The discussion between the width and depth of deep neural networks is an essential topic of research. Researchers have focused on the correct approaches to increase the depth of DNN, which in turn increases the accuracy. Nguyen et al. (2020) demonstrate that the width and depth are connected to the model capacity and enable the DNN to learn block structures which lead to good accuracy. Hence, by splitting the width, chances are that we could decrease the model capacity.

Deep neural networks are models with huge amounts of parameters, works like the lottery ticket hypothesis (Frankle & Carbin, 2018) suggest that these DNNs are heavily overparameterized. Thus it is possible to kill up to 80% of the neurons without losing too much performance. MIMO builds on this assumption to ensure that each network drives several networks simultaneously. However, unlike in our work, the boundaries between the networks are not clearly defined. Thus, the same neuron can be used for multiple different subnetworks (DNNs) in the same ensemble. In our case, we clearly assign each neuron to a DNN from the ensemble. This way, the DNNs do not get mixed up and can learn an independent representation. However, as in MIMO, we rely on the fact that not all neurons are helpful, so we split the width of the initial DNNs into a set of DNNs. This decomposition may seem crude. However, it allows us to better parallelize Packed-Ensembles for training and inference. To overcome the fact that our networks are not wide enough if M is too large, we have added an alpha hyperparameter that can increase the width of the subnetworks. In Figure 8, we study the influence of the width of the subnetworks.

First of all, we notice that the accuracy increases with the width, while the AUPR seems relatively constant. This seems to confirm the importance of the alpha parameter in balancing the width of the DNN. Furthermore, reducing the width does not appear to reduce the accuracy significantly. Thus, this justifies our choice to split the width of the DNN to produce several subnetworks since the uncertainty quantification will be constant, and the accuracy will not drop drastically. Moreover, the addition of the alpha parameter allows us to add a new degree of freedom to our ensemble and enables it to increase its accuracy.

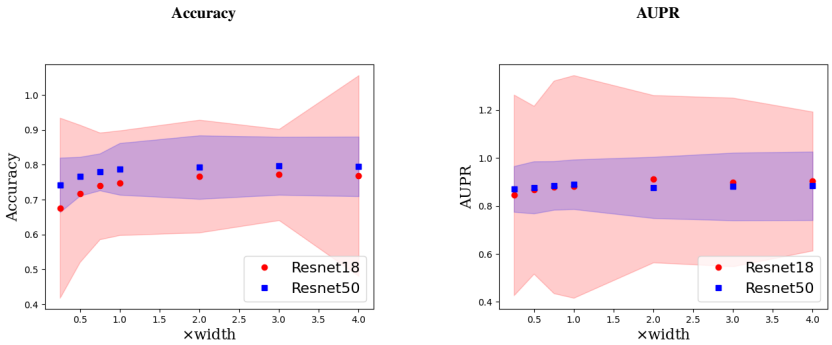


Figure 8: Accuracy and AUPR curves of ResNet-18 in red and ResNet-50 in blue on CIFAR-100 with different widths. When the width is equal to 1, it corresponds to the original ResNet; when the width is equal to x, the width of every layer is multiply by x.

Table 8: Comparison of training and inference times of different ensemble techniques using torch1.12.1+cu113 and an RTX 3090.

models *RESNET-50 CIFAR-100*	**f32 Precision** **Training** ↓ *s/epoch*	**Inference** ↑ *im/s*	**f16 Precision** **Training** ↓ *s/epoch*	**Inference** ↑ *im/s*
Single Model	37.06	3709	22.42	5718
Packed-Ensembles-(2,4,1)	179.50	1381	51.20	3406
Packed-Ensembles-(2,4,2)	175.10	1501	52.11	3440
Deep Ensembles M=4	145.30	1001	84.86	1609
MIMO M=4	37.90	1048	24.44	1801
BatchEnsemble M=4	/	/	181.10	774

H DISCUSSION ABOUT THE TRAINING VELOCITY

Our experiments show that grouped convolutions are not as fast as they could theoretically be, and confirm the statements made by many PyTorch and TensorFlow users [6]. Following the idea that grouped convolutions are bandwidth-bound, we advise readers to leverage Native Automatic Mixed Precision (AMP) and cuDNN benchmark flags when training an Packed-Ensembles to reduce the bandwidth bottleneck compared to the baseline. AMP will also divide the VRAM usage by two while yielding equally good results. Future improvements of PyTorch grouped convolutions should help this method develop its full potential, increasing its current assets. We note in Table 8 that using `float16`, Packed-Ensembles is only $1.6\times$ slower than the single model during inference. Furthermore, Packed-Ensembles is only $2.3\times$ slower during training than the single model, making it an efficient model capable of training four models in half the time of a Deep Ensembles.

I DISTRIBUTION SHIFT

In this section, we evaluate the robustness of Packed-Ensembles under dataset shift. We use models trained on CIFAR-100 (Krizhevsky, 2009) and shift the data using corruptions and perturbations proposed by (Hendrycks & Dietterich, 2019) to produce CIFAR-100-C. There are 5 levels of perturbation called "severity", from 1, the weakest to 5, the strongest. In real world scenario, distributional shift is crucial as explained by (Ovadia et al., 2019), and it is crucial to study how much a model prediction shifts from the original training data distribution. Thanks to Figure 9 we notice that Packed-Ensembles achieves the highest accuracy and lowest ECE under distributional shift, leading to a method robust against this uncertainty.

[6]For instance https://github.com/pytorch/pytorch/issues/75747

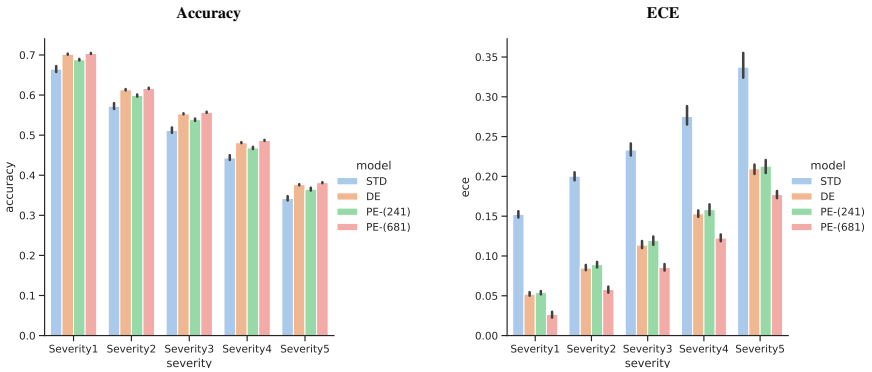


Figure 9: Accuracy and Calibration under distributional shift. Comparison of the accuracy and ECE under all types of corruptions on (a) CIFAR-100-C (Hendrycks & Dietterich, 2019) with different levels of severity.

Table 9: Comparison between the results obtained with Packed-Ensembles and a similar ResNeXt-50 on CIFAR-10

Network	Acc	NLL	ECE	AUPR	AUC	FPR95	Params (M)
PE ResNet-50	**96.0**	**0.1367**	**0.0087**	**97.1**	**94.9**	**14.5**	23.6
ResNeXt-50	90.4	0.4604	0.0709	90.4	82.5	63.4	**23.0**

J STABILIZATION OF THE PERFORMANCE

We perform five times each training task on CIFAR-10 and CIFAR-100 to estimate a better value and be able to compute a variance. Let us first note that the standard deviation for the single DNN on CIFAR-100 with a ResNet-50 architecture amounts to 0.68%. The use of ensemble strategies shrinks the standard variation to 0.43% for Deep Ensembles and 0.19% for Packed-Ensembles. Thus it seems that Packed-Ensembles make DNN predictions more stable in addition to improving accuracy and uncertainty quantification. This result is interesting as it seems to contradict (Neal et al., 2018) who claim that wider DNNs have a smaller variance. This stability might come from the ensembling.

K ON THE EQUIVALENCE BETWEEN SEQUENTIAL TRAINING AND PACKED-ENSEMBLES

The sequential training of Deep Ensembles differs significantly from the training procedure of Packed-Ensembles. The main differences lie in the subnetworks' batch composition and the best models' selection.

Concerning Packed-Ensembles, the batches are strictly the same for all subnetworks, thus removing one source of stochasticity compared to sequential learning. Yet, in practice, we show empirically that random initialization and stochastic algorithms are sufficient to get diverse subnetworks (see Appendix F for more details).

For the selection of models, Packed-Ensembles considers subnetworks as a whole (i.e., maximize the ensemble accuracy on the validation set) and therefore selects the best ensemble at a given epoch. On the other hand, sequential training selects the best networks individually, possibly on different epochs, which does not guarantee that the best ensemble is selected but ensures the optimality of subnetworks over the epochs.

Table 10: **Comparison of the efficiency of the networks trained on ImageNet (Deng et al., 2009). All ensembles have $M = 4$ subnetworks. *Mult-Adds* corresponds to the inference cost, i.e., the number of giga multiply-add operations for a forward pass which is estimated with Torchinfo.**

Method	**Net**	**Params** (M) ↓	**Mult-Adds** (G) ↓
Single Model	R50	25.6	4.09
BatchEnsemble	R50	25.7	16.36
MIMO	R50	31.7	4.45
Masksembles	R50	25.7	16.36
Packed-Ensembles ($\alpha = 3$)	R50	59.1	9.29
Deep Ensembles	R50	94.0	16.36
Single Model	R50x4	383.6	70.0
BatchEnsemble	R50x4	384.4	256.0
MIMO	R50x4	408.3	65.4
Masksembles	R50x4	384.0	256.0
Packed-Ensembles ($\alpha = 2$)	R50x4	392.0	64.47
Deep Ensembles	R50x4	1534.4	280.0

L USING GROUPS IS NOT SUFFICIENT TO PROVIDE EQUIVALENT RESULTS TO PACKED-ENSEMBLES

To make sure that our results cannot be not simply explained by the use of groups, we compare on CIFAR-10 our Packed-Ensembles to a single ResNeXt-50 (32×4d) proposed by Xie et al. (2017) in Table 9. The ResNeXt-50 is fairly equivalent to our method but does not propagate groups, only used in the middle layer of each block, which are therefore not independent. The training optimization procedures are the same as well as the data-augmentation strategies and are all detailed in Appendix B.

M EFFICIENCY OF THE NETWORKS TRAINED ON IMAGENET

Table 10 provides the efficiency of the networks trained on ImageNet-1k (see section 4.1.3), in number of parameters and multiply-additions.

N REGRESSION

To generalize our work, we propose to study regression tasks. We replicate the setting developed by Hernández-Lobato & Adams (2015), Gal & Ghahramani (2016), and Lakshminarayanan et al. (2017).

For the training in the one-dimensional regression setting, we minimize the gaussian NLL (21) using networks with two outputs neurons which estimate the parameters of a heteroscedastic gaussian distribution (Nix & Weigend, 1994; Kendall & Gal, 2017). One output corresponds to the mean of the predicted gaussian distribution, and the softmax applied on the second one is its variance. The ensemble mean $\bar{\mu}_\theta(\mathbf{x}_i)$ is computed using the empirical mean over the estimators and the variance using the formula of a mixture $\bar{\sigma}_\theta(\mathbf{x}_i)^2 = M^{-1} \sum_m \left(\sigma_{\theta_m}(\mathbf{x}_i)^2 + \mu_{\theta_m}(\mathbf{x}_i)^2\right) - \bar{\mu}_\theta(\mathbf{x}_i)$ (Lakshminarayanan et al., 2017).

$$\mathcal{L}\left(\mu_{\theta_m}(\mathbf{x}_i), \sigma_{\theta_m}(\mathbf{x}_i)^2, y_i\right) = \frac{(y_i - \mu_{\theta_m}(\mathbf{x}_i))^2}{2\sigma_{\theta_m}(\mathbf{x}_i)^2} + \frac{1}{2}\log \sigma_{\theta_m}(\mathbf{x}_i)^2 + \frac{1}{2}\log 2\pi \tag{21}$$

We compare Packed-Ensembles-$(2, 3, 1)$ and Deep Ensembles on the UCI datasets in Table 11. The subnetworks of these methods are based on a multi-layer perceptron with a single hidden layer, containing 400 neurons for the bigger Protein dataset and 200 for the others, and a ReLU non-linearity. The results show that Packed-Ensembles and Deep Ensembles provide equivalent results on most datasets.

Table 11: Comparison between the results obtained with Packed-Ensembles and Deep Ensembles on regression tasks

Datasets	RMSE		NLL	
	Packed-Ensembles	Deep Ensembles	Packed-Ensembles	Deep Ensembles
Boston housing	**2.218 ± 0.099**	**2.219 ± 0.098**	**2.028 ± 0.034**	2.047 ± 0.028
Concrete	**5.092 ± 0.225**	5.167 ± 0.234	**2.854 ± 0.028**	2.885 ± 0.032
Energy	**1.675 ± 0.085**	1.712 ± 0.067	**1.543 ± 0.072**	**1.553 ± 0.060**
Kin8nm	**0.058 ± 0.003**	**0.058 ± 0.003**	-1.442 ± 0.010	**-1.452 ± 0.010**
Naval Propulsion Plant	**0.002 ± 0.000**	**0.002 ± 0.000**	**-4.835 ± 0.066**	-4.833 ± 0.097
Power Plant	3.127 ± 0.018	**3.097 ± 0.020**	2.607 ± 0.007	**2.600 ± 0.007**
Protein	3.476 ± 0.030	**3.412 ± 0.017**	2.472 ± 0.033	**2.442 ± 0.015**
Wine	**0.482 ± 0.006**	**0.483 ± 0.006**	0.622 ± 0.014	**0.611 ± 0.013**
Yacht	**1.949 ± 0.215**	2.511 ± 0.283	**2.023 ± 0.075**	**2.023 ± 0.074**