6. Appendix

The sections in this Appendix follow a similar order to their related sections in the main paper. We first illustrate the reweighting of the loss components in Appendix 6.1. Appendix 6.2 elaborates on our analysis of filters activity. Appendix 6.3 clarifies our framework generalization with $M > 2$ subnetworks. We describe in greater details our implementation in Appendix 6.4, and then our evaluation setting in 6.5. Appendix 6.6 showcases training dynamics. We provide a quick refresher on common MSDA techniques in Appendix 6.7. Appendix 6.8 studies the importance of α. Appendix 6.9 is a preliminary study of MixMo on ImageNet. Appendix 6.10 analyzes ensembles of Cut-MixMo with CutMix that reach state of the art. Finally, we provide a pseudocode in Algorithm 1.

6.1. Weighting function w_r

As outlined in Section 3.3, the asymmetry in the mixing mechanism leads to asymmetry in the relative importance of the two inputs. Thus we reweight the loss components with function w_r, defined as $w_r(\kappa) = 2\frac{\kappa^{1/r}}{\kappa^{1/r}+(1-\kappa)^{1/r}}$. It rescales the mixing ratio κ through the use of a $\frac{1}{r}$ root operator. In the main paper, we have focused on $r = 3$.

Fig. 11 illustrates how w_r behaves for $r \in \{1, 2, 3, 4, 10\}$ and $r \to \infty$. The first extreme $r = 1$ matches the diagonal $w_r(\kappa) = 2\kappa$, without rescaling of κ, similarly to what is customary in MSDA. Our experiments in Section 4.3.3 justified the initial idea to shift the weighting function closer to the horizontal and constant curve $w_r(\kappa) = 1$ with higher r. In the other experiments, we always set $r = 3$.

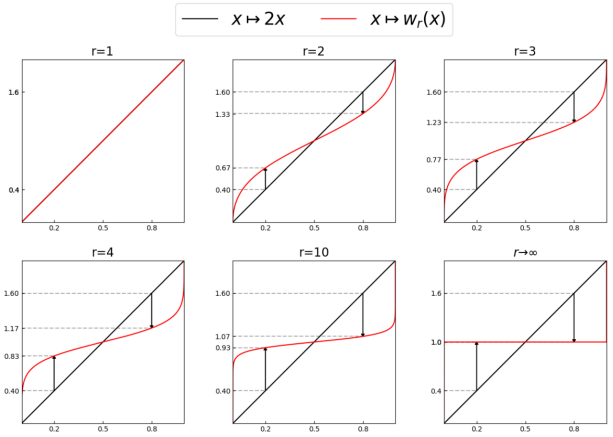

Figure 11: **Curves of the reweighting operation** that projects κ to the flattened ratio $w_r(\kappa)$

6.2. Filters activity

We argued in Section 3.4 that MixMo better leverages additional parameters in wider networks. Concretely, a larger proportion of filters in large networks really help for classification as demonstrated in Fig. 4a and 4b in the main paper. Following common practices in the structured pruning literature [47], we used the l_1-norm of convolutional filters as a proxy for importance. These 3D filters are of shape $n_i \times k \times k$ with n_i the number of input channels and k the kernel size. In Fig. 4b, we arbitrarily defined a filter as active if its l_1-norm is at least 40% of the highest filter l_1-norm in that filter's layer. We report the average percentage of active filters across all filters in the core network $\mathcal{C}$, for 3 learning strategies: vanilla, CutMix and Cut-MixMo.

Table 7: **Proportion (%) of active filters** in core network vs. width w for a WRN-28-w on CIFAR 100 and different activity thresholds t_a.

Method	Width	$t_a = 0.2$	$t_a = 0.3$	$t_a = 0.4$	$t_a = 0.5$
Vanilla	2	98.9	98.8	97.8	93.3
	3	97.3	96.4	93.2	87.5
	4	96.5	95.2	91.2	81.6
	5	95.1	91.7	85.7	73.3
	7	92.6	88.2	81.0	69.5
	10	87.8	80.4	71.5	57.3
	14	83.9	74.0	61.6	46.8
CutMix	2	99.2	99.0	97.8	95.3
	3	98.7	98.5	97.2	93.4
	4	98.1	97.4	94.0	87.3
	5	97.0	96.1	90.7	80.6
	7	95.8	94.0	86.2	74.6
	10	93.5	88.4	81.3	67.0
	14	89.4	81.9	70.3	50.9
Cut-MixMo	2	100.0	100.0	99.4	97.3
	3	99.8	99.8	99.7	98.7
	4	99.7	99.7	99.6	98.7
	5	99.3	99.3	98.9	97.4
	7	98.9	98.8	98.0	95.2
	10	98.5	98.2	96.8	92.4
	14	97.5	96.3	93.1	82.6

The threshold $t_a = 0.4$ was chosen for visualization purposes. Nevertheless, the observed trend in activity proportions remains for varying thresholds in Tab. 7. For example, for the lax $t_a = 0.2$, CutMix uses 93.5% of filters vs. 98.5% for Cut-MixMo.

6.3. Generalization to $M > 2$ heads

We have mostly discussed our MixMo framework with $M = 2$ subnetworks. For better readability, we referred to the mixing ratios κ and $1 - \kappa$ with $\kappa \sim \text{Beta}(\alpha, \alpha)$. It's equivalent to a more generic formulation $(\kappa_0, \kappa_1) \in \text{Dir}_2(\alpha)$ from a symmetric Dirichlet distribution with concentration parameter α. This leads to the alternate equations $\mathcal{L}_{\text{MixMo}} = \sum_{i=0,1} w_r(\kappa_i)\mathcal{L}_{\text{CE}}(y_i, \hat{y}_i)$, where $w_r(\kappa_i) = 2\frac{\kappa_i^{1/r}}{\sum_{j=0,1}\kappa_j^{1/r}}$.

Now generalization to the general case $M \geq 2$ is straightforward. We draw a tuple $\{\kappa_i\}_{0 \leq i < M} \sim \text{Dir}_M(\alpha)$

and optimize the training loss:

$$\mathcal{L}_{\text{MixMo}} = \sum_{i=0}^{M-1} w_r(\kappa_i)\mathcal{L}_{\text{CE}}\left(y_i, \hat{y}_i\right), \tag{4}$$

where the new weighting naturally follows:

$$w_r(\kappa_i) = M\frac{\kappa_i^{1/r}}{\sum_{j=0}^{M-1}\kappa_j^{1/r}}, \forall i \in \{0, \dots, M-1\}. \tag{5}$$

The remaining point is the generalization of the mixing block $\mathcal{M}$, that relies on the existence of MSDA methods for $M > 2$ inputs. The linear interpolation can be easily expanded as in Mixup:

$$\mathcal{M}_{\text{Linear-MixMo}}\left(\{l_i\}\right) = M\sum_{i=0}^{M-1}\kappa_i l_i, \tag{6}$$

where $l_i = c_i(x_i)$. However, extensions for other masking MSDAs have only recently started to emerge [40]. For example, CutMix is not trivially generalizable to $M > 2$, as the patches could overlap and hide important semantic components. In our experiments, a soft extension of CutMixMo performs best: it first linearly interpolates $M-1$ inputs and then patches a region from the M-th:

$$\begin{aligned}\mathcal{M}_{\text{Cut-MixMo}}\left(\{l_i\}\right) = M[&\mathbb{1}_{\mathcal{M}}\odot l_k + \\ &(\mathbb{1} - \mathbb{1}_{\mathcal{M}})\odot\sum_{i=0, i\neq k}^{M-1}\frac{\kappa_i}{1-\kappa_k}l_i],\end{aligned} \tag{7}$$

where $\mathbb{1}_{\mathcal{M}}$ is a rectangle of area ratio κ_k and k sampled uniformly in $\{0, 1, \dots, M-1\}$. However, it has been less successful than $M = 2$, as only two subnetworks can fit independently in standard parameterization regimes. Future work could design new framework components, such as specific mixing blocks, to tackle these limits.

6.4. Implementation details

We first used the popular image classification **datasets** CIFAR-100 and CIFAR-10 [42]. They contain 60k 32×32 natural and colored images in respectively 100 classes and 10 classes, with 50k training images and 10k test images. At a larger scale, we study Tiny ImageNet [10], a downsampled version of ImageNet [12]. It contains 200 different categories, 100k 64×64 training images (*i.e.* 500 images per class) and 10k test images.

Our code was adapted from the official MIMO [30] implementation[1]. For CIFAR, we re-use the **hyperparameters** from MIMO [30]. The optimizer is SGD with learning rate of $\frac{0.1}{b} \times \frac{\text{batch-size}}{128}$, batch size 64, linear warmup over 1 epoch, decay rate 0.1 at steps $\{100, 200, 225\}$, l_2 regularization 3e-4. We follow standard MSDA practices [2, 41, 83] and set the maximum number of epochs to 300. For Tiny ImageNet, we adapt PreActResNet-18-w, with $w \in \{1, 2, 3\}$ times more filters. We re-use the hyperparameters from Puzzle-Mix [41]. The optimizer is SGD with learning rate of $\frac{0.2}{b}$, batch size 100, decay rate 0.1 at steps $\{600, 900\}$, 1200 epochs maximum, weight decay 1e-4. Our experiments ran on a single NVIDIA 12Go-TITAN X Pascal GPU. All results without a † were obtained with these training configurations. We will soon release our code and pre-trained models to facilitate reproducibility.

Batch repetition increases performances at the cost of **longer training**, which may be discouraging for some practitioners. Thus in addition to $b = 4$ as in MIMO [30], we often consider the quicker $b = 2$. Note that most of our concurrent approaches also increase training time: DE [43] via several independent trainings, Puzzle-Mix [41] via saliency detection ($\approx \times 2$), GradAug [82] via multiple subnetworks predictions ($\approx \times 3$) or Mixup BA [36] via 10 batch augmentations ($\approx \times 7$ with our hardware on a single GPU).

MixMo operates in the features space and is complementary with **pixels augmentations**, *i.e.* cropping, AugMix. The standard vanilla pixels data augmentation [31] consists of 4 pixels padding, random cropping and horizontal flipping. When combined with CutMix, notably to benefit from multilabel smoothing, the input may be of the form: $(m_x(x_i, x_k, \lambda), x_j)$, where x_k is randomly chosen in the whole dataset, and not only inside the current batch[2]. Moreover, $\mathcal{M}_{\text{Cut-MixMo}}$ modifies by $\mathbb{1}_{\mathcal{M}}$ the visible part from mask $\mathbb{1}_m$ (of area λ). We thus modify targets accordingly: $(\lambda' y_i + (1-\lambda')y_k, y_j)$ where $\lambda' = \frac{\sum \mathbb{1}_m \odot \mathbb{1}_{\mathcal{M}}}{\sum \mathbb{1}_{\mathcal{M}}}$. To fully benefit from b, we force the repeated x_i to remain predominant in its b appearances: *i.e.*, we swap x_i and x_k if $\lambda' < 0.5$. We see CutMix as a perturbation on the main batch sample.

Distributional uncertainty measures help when there is a mismatch between train and test data distributions. Thus [34] introduced **CIFAR-100-c** on which **AugMix** performs best. AugMix sums the pixels from a chain of several augmentations and is complementary to our approach in features. We use default parameters[3]: the severity is set 3, the mixture's width to 3 and the mixture's depth to 4. We exclude operations in AugMix which overlap with CIFAR-100-c corruptions: thus, [equalize, posterize, rotate, solarize, shear_x, shear_y, translate_x, translate_y] remain. We disabled the Jensen-Shannon Divergence loss between predictions for the clean image and for the same image AugMix augmented: that would otherwise triple the training time. For comparison of out-of-domain uncertainty estimations, we report NLL as in [30, 58]: indeed, the recommendation of [2] to apply TS only stands for in-domain test set.

[1] https://github.com/google/edward2/

[2] Following https://github.com/ildoonet/cutmix

[3] https://github.com/google-research/augmix/blob/master/cifar.py

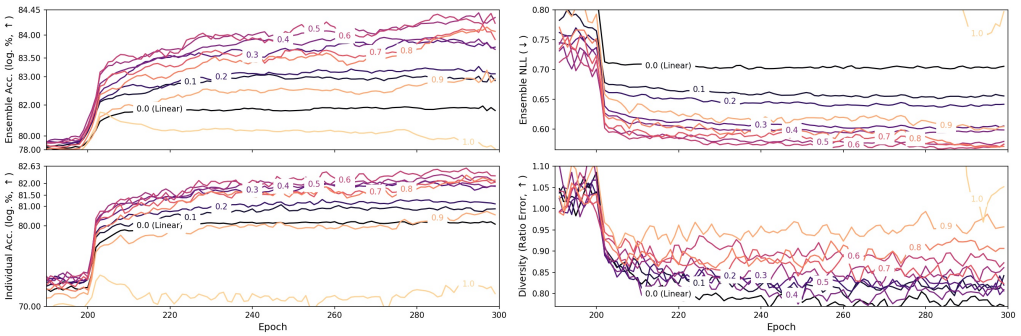

Figure 12: **Training dynamics**. Higher probability p of binary mixing via patches increases diversity (lower right), and also subnetworks accuracy (lower left) but only up to $p = 0.6$. Around this value, we obtain best ensemble performances, in terms of accuracy (upper left) or uncertainty estimation (upper right). $b = 2$, $r = 3$, $\alpha = 3$ with WRN-28-10 on CIFAR-100.

Table 8: **WRN-28-10 on CIFAR** without early stopping.

Dataset		CIFAR-100					CIFAR-10			
Approach	Time Tr./Inf.	Top1 %,↑	Top5 %,↑	NLL_c 10^{-2},↓	NLL 10^{-2},↓	ECE 10^{-2},↓	Top1 %,↑	NLL_c 10^{-2},↓	NLL 10^{-2},↓	ECE 10^{-2},↓
Vanilla	1/1	81.47	95.57	73.6	76.2	6.47	96.31	12.5	14.1	1.95
Mixup	1/1	83.15	95.75	66.3	67.3	**1.62**	97.00	11.3	11.5	0.97
Hard PatchUp†	1/1	83.87	-	-	66.0	-	97.47	-	11.4	-
CutMix	1/1	83.74	96.18	65.4	66.1	4.95	97.21	9.7	10.8	1.51
Puzzle-Mix†	2/1	84.05	96.08	66.9	68.1	2.76	-	-	-	-
GradAug†	3/1	83.98	96.28	-	-	-	-	-	-	-
+ CutMix†	3/1	85.25	96.85	-	-	-	-	-	-	-
Mixup BA†	7/1	84.30	-	-	-	-	**97.80**	-	-	-
DE (2 Nets)	2/2	83.15	96.30	66.0	67.2	5.15	96.58	11.1	12.2	1.82
+ CutMix	2/2	85.46	96.90	57.4	57.5	3.62	97.51	8.7	9.0	1.16
MIMO ($M = 2$)	2/1	82.04	95.75	69.1	72.4	6.32	96.33	12.1	13.4	1.89
Linear-MixMo	2/1	81.88	95.97	67.8	70.3	6.20	96.55	11.4	12.5	1.67
+ CutMix	2/1	84.55	96.95	57.4	57.5	2.54	97.34	8.9	9.3	1.34
Cut-MixMo	2/1	84.07	96.97	56.6	57.9	4.19	97.26	8.7	9.1	0.98
+ CutMix	2/1	85.17	97.28	54.4	54.5	2.13	97.33	8.5	8.6	**0.88**
MIMO ($M = 2$)	4/1	82.74	95.90	67.0	74.0	7.56	96.66	11.5	13.6	1.98
MIMO† ($M = 3$)	4/1	82.0	-	-	69.0	2.2	96.4	-	12.3	1.0
Linear-MixMo	4/1	82.53	96.08	65.8	68.5	6.64	96.78	10.8	11.8	1.80
+ CutMix	4/1	85.24	96.97	56.3	56.4	3.53	97.53	8.8	8.6	1.19
Cut-MixMo	4/1	85.32	97.12	53.6	54.8	4.53	97.42	8.1	8.4	1.15
+ CutMix	4/1	**85.59**	**97.33**	**53.2**	**53.3**	1.95	97.70	**8.0**	**8.2**	0.98

6.5. Evaluation setting and metrics

We reproduce the **experimental setting** from CutMix [83], Manifold Mixup [76] and other works such as the recent state-of-the-art ResizeMix [63]: in absence of a validation dataset, results are reported at the epoch that yields the best test accuracy. For fair comparison, we apply this early stopping for all concurrent approaches. Nonetheless, for the sake of completeness, Table 8 shows results without early stopping on the main experiment (CIFAR with a standard WRN-28-10). We recover the exact same ranking among methods as in Table 1.

Following recent works in ensembling [9, 50, 64], we have mainly focused on the NLL_c **metric** for in-domain test set. Indeed, [2] have shown that "comparison of [...] ensembling methods without temperature scaling (TS) [25] might not provide a fair ranking". Nevertheless in Table 8, we found that *Negative Log-Likelihood* (NLL) (without TS) leads to similar conclusions as NLL_c (after TS).

The TS even mostly seems to benefit to poorly calibrated models, as shown by the calibration criteria *Expected Calibration Error* (ECE, ↓, 15 bins). ECE measures how confidences match accuracies. MixMo attenuates over-confidence in large networks and thus reduces ECE. In our case, combining ensembling and data augmentation improves calibration [79]. Note that the appropriate measure of calibration is still under debate [56]. Notably, [2] have also stated that, despite being widely used, ECE is biased and unreliable: we can confirm that we found ECE to be dependant to hyper-parameters and implementation details. Due to space constraints and these pitfalls, we have not included this controversial metric in the main paper.

6.6. Training dynamics

Fig. 12 showcases training dynamics for probability $p \in [0, 1]$ of patch mixing (see Section 4.3.2). In the remaining $1 - p$, we interpolate features linearly. For $p = 0$, we recover our Linear-MixMo; for $p = 0.5$, we recover our Cut-MixMo. In all approaches, p is linearly reduced towards 0 beyond the $\frac{11}{12}$ of the training epochs, *i.e.* from epoch 275 to 300 on CIFAR. As we sum at inference, this reduces the train-test distribution gap and slightly increases individual accuracy during the final epochs (lower left in Fig. 12).

Diversity is measured by the ratio-error, the ratio between the number of samples on which only one of the two predictor is wrong, divided by the number of samples on which they are both wrong. It is positively correlated with p. However, individual accuracies first increase with p until $p = 0.6$, then the tendency is reversed. Overall, best ensemble performances in terms of accuracy (Top1) and uncertainty (NLL) estimation are obtained with $p \in [0.5, 0.6]$. Most importantly, we note that the performance gaps are consistent and stable along training.

6.7. Mixed sample data augmentations

We have drawn inspiration from MSDA techniques to design our mixing block $\mathcal{M}$. In particular, Section 4.3.2 compared different $\mathcal{M}$ based on recent papers. Fig. 13 provides the reader a visual understanding of their behaviour, which we explain below.

MixUp [86] linearly interpolates between pixels values: $m_x(x_i, x_k, \lambda) = \lambda x_i + (1-\lambda)x_k$. The remaining methods fall under the label of **binary MSDA**: $m_x(x_i, x_k, \lambda) = \mathbb{1}_m \odot x_i + (\mathbb{1} - \mathbb{1}_m) \odot x_k$ with $\mathbb{1}_m$ a mask with binary values $\{0, 1\}$ and area of ratio λ. They diverge in how this mask is created. The **horizontal concatenation**, also found in [68], simply draws a vertical line such that every pixel to the left belongs to one sample and every pixel to the right belongs to the other. Similarly, we define a **vertical concatenation** with an horizontal line. **PatchUp** [17] adapted DropBlock [24]: a canvas C of patches is created by sampling for every spatial coordinate from the Bernoulli distribution $\text{Ber}(\lambda')$ (where λ' is a recalibrated value of λ): if the drawn binary value is 1, a patch around that coordinate is set to 1 on the final binary mask $\mathbb{1}_m$. PatchUp was designed for in-manifold mixing with a different mask by channels. However, duplicating the same 2D mask in all channels for $\mathcal{M}$ performs better in our experiments. **FMix** [29] selects a large contiguous region in one image and pastes it onto another. The binary mask is made of the top-λ percentile of pixels from a low-pass filtered 2D map G drawn from an isotropic Gaussian distribution. **CowMix** [20, 21] selects a cow-spotted set of regions, and is somehow similar to FMix with a Gaussian filtered 2D map G. **CutMix** [83] was inspired by CutOut [13]. Formally, we sample a square with edges of length $R\sqrt{\lambda}$, where R is the length of an image edge. Note that this sometimes leads to non square rectangles when the initially sampled square overlaps with the edge from the original image. We adjust our λ a posteriori to fix this boundary effect. Regarding the hyper-parameters, we use in $\mathcal{M}$ those provided in the seminal papers, except for sampling of κ where we set $\alpha = 2$ in all setups.

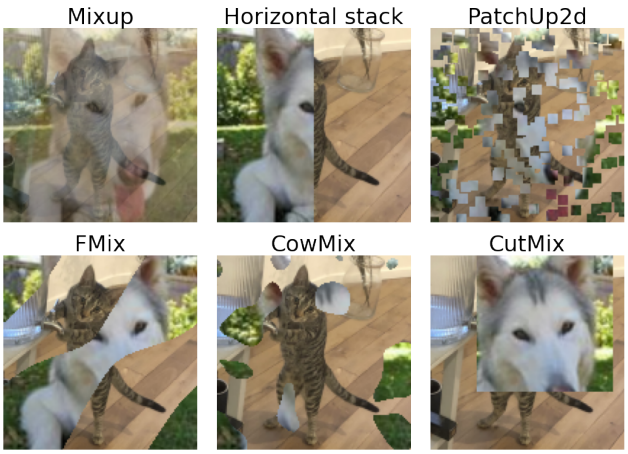

Figure 13: **Common MSDA procedures with** $\lambda = 0.5$.

Note we consider both versions of MixUp (in-pixel and manifold) in this paper, but only the in-pixel version of CutMix. Indeed, the manifold version of CutMix was shown in the seminal CutMix paper [83] to be inferior to the standard in-pixel variant.

6.8. Hyper-parameter α

In Fig. 14, we study the impact of different values of α, parameterizing the sampling law for $\kappa \sim \text{Beta}(\alpha, \alpha)$. For high values of α, the interval of κ narrows down around 0.5. Diversity is therefore decreased: we speculate this is because we do not benefit anymore from lopsided updates. The opposite extreme, when α=1, is equivalent to uniform distribution between 0 and 1. Therefore diversity is increased, at the cost of lower individual accuracy due to less stable training. For simplicity, we set α=2. Manifold-Mixup [76] selected the same value on CIFAR-100. However, this value could be fine tuned on the target task: *e.g.* in Fig. 14, α=4 seems to perform best for Cut-MixMo on CIFAR-100 with WRN-28-10 with r=3, p=0.5 and b=2.

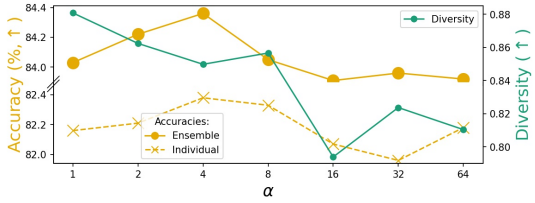

Figure 14: **Diversity/accuracy** as function of α.

6.9. Preliminary ImageNet experiments

To further prove MixMo's ability to scale to more complex problems, we also conduct a preliminary study of its behavior on the larger scale ImageNet dataset [12]. Following the protocol outlined in the seminal MIMO paper [30], we consider variations on the standard ResNet-18 in the form of ResNet-18-w networks where w is multiplicative width factor.

These first experiments confirm that MixMo performs well when networks are overparameterized. For values of $w \geq 5$, our network at the end of training outperforms both Vanilla and CutMix baselines. For example, with a ResNet-18-5 backbone, Cut-MixMo (78.20% Top1, 0.867 NLLc) improves over Vanilla (76.47%, 1.121) and CutMix (77.40%, 1.263). This remains the case for a ResNet-18-7 backbone with Cut-MixMo (78.55% Top1, 0.846 NLLc) outperforming Vanilla (76.86%, 1.100) and CutMix (77.18%, 1.190).

Table 9: **Summary**: WRN-28-w on CIFAR-100. $b = 4$.

Width w	Approach CutMix	1-Net -	1-Net ✓	2-Nets -	2-Nets ✓	Linear-MixMo -	Linear-MixMo ✓	Cut-MixMo -	Cut-MixMo ✓	2-Cut-MixMos -	2-Cut-MixMos ✓
2	Top1	76.44	78.06	79.16	80.81	75.82	76.36	75.66	75.17	76.98	76.11
	NLL_c	0.921	0.815	0.776	0.695	0.841	0.824	0.824	0.846	0.7661	0.798
	# params	1.48M		2.95M		1.49M				2.99M	
3	Top1	77.95	80.70	80.85	83.14	78.51	80.74	79.81	79.85	80.78	81.20
	NLL_c	0.862	0.750	0.738	0.644	0.760	0.696	0.693	0.702	0.635	0.650
	# params	3.31M		6.62M		3.33M				6.66M	
4	Top1	78.84	81.55	81.48	83.93	80.43	81.66	81.68	81.69	82.57	82.58
	NLL_c	0.824	0.711	0.711	0.609	0.712	0.656	0.646	0.635	0.590	0.588
	# params	5.87M		11.74M		5.89M				11.79M	
5	Top1	79.75	82.55	82.18	84.60	80.95	83.06	83.11	83.34	83.97	84.31
	NLL_c	0.813	0.686	0.693	0.596	0.703	0.617	0.598	0.591	0.549	0.546
	# params	9.16M		18.32M		9.19M				18.39M	
7	Top1	81.14	83.71	82.94	85.52	82.4	84.51	84.32	84.94	85.50	85.90
	NLL_c	0.764	0.648	0.673	0.573	0.675	0.581	0.562	0.543	0.516	0.498
	# params	17.92M		35.85M		17.97M				35.94M	
10	Top1	81.63	84.05	83.17	85.74	83.08	85.47	85.40	85.77	86.04	86.63
	NLL_c	0.750	0.644	0.668	0.571	0.656	0.558	0.535	0.524	0.494	0.479
	# params	36.53M		73.07M		36.60M				73.21M	
14	Top1	82.01	84.31	83.47	85.80	83.79	86.05	85.76	86.19	86.58	87.11
	NLL_c	0.730	0.645	0.656	0.569	0.648	0.545	0.527	0.518	0.488	0.473
	# params	71.55M		143.1M		71.64M				143.28M	

6.10. Ensemble of Cut-MixMo with CutMix

Fig. 15 plots performance for different widths w in WRN-28-w and varying number of ensembled networks N: two vertically aligned points have the same parameter budget. Indeed, the total number of parameters in our architectures has been used as a proxy for model complexity, as in [9, 50]. The increase in the total number of weights in MixMo is visually almost unnoticeable. Precisely, with WRN-28-10, MixMo (M=2) has 36.60M weights vs. 36.53M standardly (+0.2%). Moreover, the number of flops is 5.9571G Flops for MixMo vs. 5.9565G Flops standardly (+0.01%). That's why we state we achieve ensembling (almost) "for free".

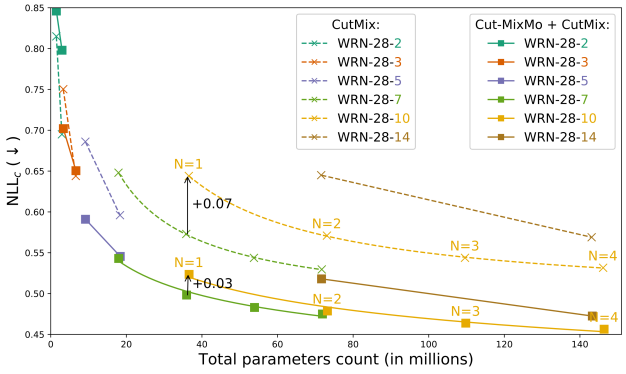

Figure 15: **Ensemble effectiveness** (NLL_c/#params). We slide the width in WRN-28-w and numbers of members N. CutMix data augmentation. Interpolations through power laws [50] when more than 2 points are available.

We compare ensembling with CutMix rather than standard pixels data augmentation, as previously done in Fig. 6 from Section 4.6. CutMix induces additional regularization and label smoothing: empirically, it improves all our approaches. For a fixed memory budget, a single network usually performs worse than an ensemble of several medium-size networks: we recover the **Memory Split Advantage** even with CutMix. However, Cut-MixMo challenges this by remaining closer to the lower envelope. In other words, parameters allocation (more networks or bigger networks) has less impact on results. This is due to Cut-MixMo's ability to better use large networks.

In Table 9, we summarize several experiments on CIFAR-100. Among other things, we can observe that large vanilla networks tend to gain less from ensembling [50]: *e.g.* 2 vanillas WRN-28-10 (83.17% Top1, 0.668 NLL_c)

do not perform much better than 2 WRN-28-7 (82.94%, 0.673). This remains true even with CutMix: (85.74%, 0.571) vs. (85.52%, 0.573). We speculate this is related to wide networks' tendency to converge to less diverse solutions, as studied in [55]. Contrarily, **MixMo improves the ensembling of large networks**, with (86.04%, 0.494) vs. (85.50%, 0.517) on the same setup. When additionally combined with CutMix, we obtain state of the art (86.63%, 0.479) vs. (85.90%, 0.498). This demonstrates the importance of Cut-MixMo in cooperation with standard pixels data augmentation. It attenuates the drawbacks from over-parameterization This is of great importance for practical efficiency: it modifies the optimal network width for real-world applications.

6.11. Pseudo Code

Finally, the pseudocode in Algorithm 1 describes the procedure behind Cut-MixMo with $M = 2$.

Algorithm 1: Procedure for Cut-MixMo with $M = 2$ subnetworks

/* Setup */

Parameters: First convolutions $\{c_0, c_1\}$, dense layers $\{d_0, d_1\}$ and core network $\mathcal{C}$, randomly initialized.

Input: Dataset $D = \{x_i, y_i\}_{i=1}^{|D|}$, probability p of applying binary mixing via patches, reweighting coefficient r, concentration parameter α, batch size b_s, batch repetition b, optimizer g , learning rate l_r.

/* Training Procedure */

for *epoch* ***from*** 1 ***to*** *#epochs* **do**

  for *step* ***from*** 1 ***to*** $\frac{|D| \times b}{b_s}$ **do**

    /* Step 1: Batch creation */

    Randomly select $\frac{b_s}{b}$ samples // Sampling

    Duplicate these samples b times to create batch $\{x_i, y_i\}_{i \in B}$ of size b_s // Batch repetition

    Randomly shuffle B with π to create $\{(x_i, x_j), (y_i, y_j)\}_{i \in B, j = \pi(i)}$ // Shuffling

    /* Step 2: Define the mixing mechanism at the batch level */

    if *epoch* $> \frac{11}{12} \times$ *#epochs* **then**

      $p_e = p \frac{\text{\#epochs} - \text{epoch}}{\frac{1}{12} \times \text{\#epochs}}$ // Linear descent to 0 over the last twelfth of training

    else

      $p_e = p$

    Sample $1_{binary} \sim \text{Ber}(p_e)$ from Bernoulli distribution // Whether we apply binary or linear mixing

    Sample $1_{outside} \sim \text{Ber}(0.5)$ // Whether the first input is inside or outside the rectangle

    /* Step 3: Forward and loss */

    for $i \in B$ **do**

      Sample $\kappa_i \sim \text{Beta}(\alpha, \alpha)$

      $l_i^0 = c_0(x_i)$ and $l_i^1 = c_1(x_{\pi(i)})$

      if 1_{binary} **then**

        Sample $\mathbb{1}_{\mathcal{M}}$ a rectangular binary mask with average κ_i (as in CutMix)

        if $1_{outside}$ **then**

          $\mathbb{1}_{\mathcal{M}} \leftarrow \mathbb{1} - \mathbb{1}_{\mathcal{M}}$ // Permute the rectangle and its complementary

          $\kappa_i \leftarrow 1 - \kappa_i$

        $l_i = 2\left[\mathbb{1}_{\mathcal{M}} \odot l_0 + (\mathbb{1} - \mathbb{1}_{\mathcal{M}}) \odot l_1\right]$ // Apply binary mixing

      else

        $l_i = 2\left[\kappa_i l_0 + (1 - \kappa_i) l_1\right]$ // Apply linear interpolation

      Extract features $f_i \leftarrow \mathcal{C}(l_i)$ from core network

      Compute predictions $\hat{y}_i^0 \leftarrow d_0(f_i)$ and $\hat{y}_i^1 \leftarrow d_1(f_i)$

      Compute weights $w_i \leftarrow 2 \frac{\kappa_i^{1/r}}{\kappa_i^{1/r} + (1 - \kappa_i)^{1/r}}$

      Compute loss $\mathcal{L}_i \leftarrow w_i \mathcal{L}_{\text{CE}}\left(y_i, \hat{y}_i^0\right) + (2 - w_i) \mathcal{L}_{\text{CE}}\left(y_{\pi(i)}, \hat{y}_i^1\right)$

    Average loss $\mathcal{L}_{\text{MixMo}} \leftarrow \frac{1}{|B|} \sum \mathcal{L}_i$

    /* Step 4: Back propagation */

    $c_0, c_1, \mathcal{C}, d_0, d_1 \leftarrow g\left(\text{gradient} = \nabla \mathcal{L}_{\text{MixMo}}, \text{learning rate} = \frac{l_r}{b}\right)$

/* Test Procedure */

Data: Inputs $\{x_i\}_{i=1}^{T}$ // Test Data

for $i \in \{1, \ldots, T\}$ **do**

  Extract features $f_i = \mathcal{C}\left(c_0(x_i) + c_1(x_i)\right)$

  Output: $\frac{1}{2}\left[d_0(f_i) + d_1(f_i)\right]$