MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks

Alexandre Ramé[*†1], Rémy Sun[*1,2] and Matthieu Cord[1,3]

[1]Sorbonne Université, CNRS, LIP6, Paris, France
[2]Optronics & Missile Electronics, Land & Air Systems, Thales
[3]Valeo.ai

Abstract

Recent strategies achieved ensembling "for free" by fitting concurrently diverse subnetworks inside a single base network. The main idea during training is that each subnetwork learns to classify only one of the multiple inputs simultaneously provided. However, the question of how to best mix these multiple inputs has not been studied so far.

In this paper, we introduce MixMo, a new generalized framework for learning multi-input multi-output deep subnetworks. Our key motivation is to replace the suboptimal summing operation hidden in previous approaches by a more appropriate mixing mechanism. For that purpose, we draw inspiration from successful mixed sample data augmentations. We show that binary mixing in features - particularly with rectangular patches from CutMix - enhances results by making subnetworks stronger and more diverse.

We improve state of the art for image classification on CIFAR-100 and Tiny ImageNet datasets. Our easy to implement models notably outperform data augmented deep ensembles, without the inference and memory overheads. As we operate in features and simply better leverage the expressiveness of large networks, we open a new line of research complementary to previous works.

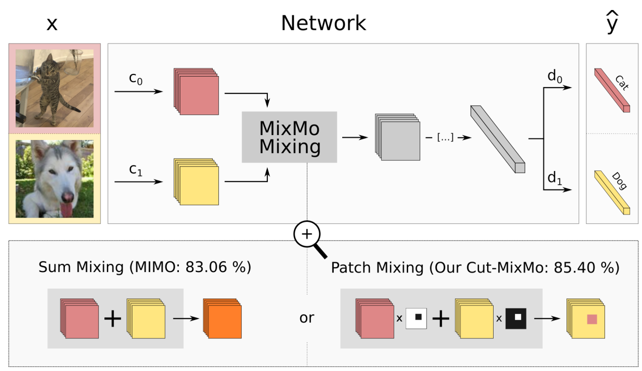


Figure 1: **MixMo overview**. We embed $M = 2$ inputs into a shared space with convolutional layers (c_1, c_2), mix them, pass the embedding through further layers and output 2 predictions via dense layers (d_1, d_2). The key point of our MixMo is the mixing block. Mixing with patches performs better than basic summing: 85.40% vs. 83.06% (MIMO [30]) on CIFAR-100 with WRN-28-10.

1. Introduction

Convolutional Neural Networks (CNNs) have shown exceptional performance in computer vision tasks, notably classification [42]. However, among other limitations, obtaining reliable predictions remains challenging [34, 58]. For additional robustness in real-world scenarios or to win Kaggle competitions, CNNs usually pair up with two practical strategies: data augmentation and ensembling.

Data augmentation reduces overfitting and improves generalization, notably by diversifying training samples [51]. Traditional approaches are label-preserving. In contrast, recent **mixed sample data augmentation** (MSDA) create artificial samples by mixing multiple inputs and their labels proportionally to a ratio λ. The seminal work Mixup [86] linearly interpolates pixels while Manifold Mixup [76] interpolates latent features in the network. Binary masking MSDAs [21, 29, 41] such as CutMix [83] have since diversified mixed samples by pasting patches from one image onto another in place of interpolation.

Aggregating predictions from a diverse set of neural networks (*i.e.* with different failure cases) strongly improves generalization [14, 28, 43], notably uncertainty estimation [2, 27, 58]. An ensemble of several small networks usually performs better than one large network empirically [9, 50]. Yet, unfortunately, ensembling is costly in time and memory both at training and inference: this often limits applicability.

[*]Equal contribution.
[†]Correspondence to alexandre.rame@lip6.fr

In this paper, we propose **MixMo**, a new generalized multi-input multi-output framework: we train a base network with $M \geq 2$ inputs and outputs. This way, we fit M independent subnetworks [23, 30, 66] defined by an input/output pair and a subset of network weights. This is possible as large networks only leverage a subset of their weights [19]. Rather than pruning (ie, eliminating) inactive filters [44, 47], we seek to fully use the available neurons and over parameterization through multiple subnetworks.

The challenge is to prevent homogenization and enforce diversity among subnetworks with no structural differences. Thus, we consider M (input, label) pairs at the same time in training: $\{(x_i, y_i)\}_{0 \leq i < M}$. M **images are treated simultaneously**, as shown on Fig. 1 with $M = 2$. The M inputs are encoded by M separate convolutional layers $\{c_i\}_{0 \leq i < M}$ into a shared latent space before being mixed. The representation is then fed to the core network, which finally branches out into M dense layers $\{d_i\}_{0 \leq i < M}$. Diverse subnetworks naturally emerge as d_i learns to classify y_i from input x_i. At inference, the same image is repeated M times: we obtain ensembling "for free" by averaging M predictions.

The key divergent point between MixMo variants lies in the **multi-input mixing block** that seeks features independence. Should the merging be a basic summation or a concatenation, we would recover MIMO [30] or respectively Aggregated Learning [66] - which both featured this multi-input multi-output strategy.

Our main intuition is simple: we see summing as a balanced and restrictive form of Mixup [86] where $\lambda = \frac{1}{M}$. By analogy, we draw from the considerable MSDA literature to design a more appropriate mixing block. In particular, we leverage binary masking methods to ensure subnetworks diversity. Our framework allows us to create a new Cut-MixMo variant inspired by CutMix [83], and illustrated in Fig. 1: a patch of features from the first input is pasted into the features from the second input.

This asymmetrical mixing also raises new questions regarding information flow in the network's features. We tackle the imbalance between the multiple classification training tasks via a new weighting scheme. Conversely, MixMo's double nature as a new mixing augmentation in features yields important insights on traditional MSDA.

In summary, our contributions are threefold:

1. We propose a general framework, MixMo, connecting two successful fields: mixing samples data augmentations & multi-input multi-output ensembling.

2. We identify the appropriate mixing block to best tackle the diversity/individual accuracy trade-off in subnetworks: our easy to implement Cut-MixMo benefits from the synergy between CutMix and ensembling.

3. We design a new weighting of the loss components to properly leverage the asymmetrical inputs mixing.

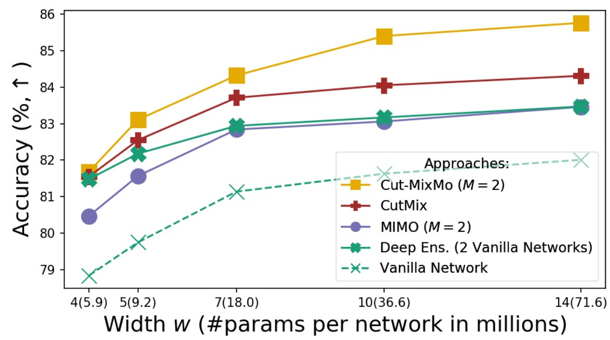


Figure 2: **Main results**. CIFAR-100 with WRN-28-w. Our Cut-MixMo variant (patch mixing and $M = 2$) surpasses CutMix and deep ensembles (with half the parameters) by leveraging over-parameterization in wide networks.

We demonstrate excellent accuracy and uncertainty estimation with MixMo on CIFAR-10/100 and Tiny ImageNet. Specifically, Cut-MixMo with $M = 2$ reaches state of the art on these standard datasets: as exhibited by Fig. 2, it outperforms CutMix, MIMO and deep ensembles, at (almost) the same inference cost as a single network.

2. Related work

2.1. Data augmentation

CNNs are known to memorize the training data [85] and make overconfident predictions [25] to the detriment of generalization on new test examples. **Data Augmentation** (DA) inflates the training dataset's size by creating artificial samples from available labeled data. Beyond slight perturbations (*e.g.* rotation), recent works [11, 35] apply stronger transformations [33]. CutOut [13] randomly deletes regions of images in training and prevents models from focusing on a single pixels region, similarly to how regularizations like Dropout [67] or DropBlock [24] force networks to leverage multiple features.

Mixed Sample Data Augmentation (MSDA) recently expanded the notion of DA. From pairs of labeled samples $\{(x_i, y_i), (x_k, y_k)\}$, they create virtual samples: $(m_x(x_i, x_k, \lambda), \lambda y_i + (1-\lambda) y_k)$ where $\lambda \sim \text{Beta}(\alpha, \alpha)$. [48] shows that mixing the targets differently than this linear interpolation may cause underfitting and unstable learning. Indeed, approaches mainly focus on developing the most effective input mixing m_x. In [38, 72, 73, 86], m_x performs a simple linear interpolation between pixels: *e.g* in Mixup [86], $m_x(x_i, x_k, \lambda) = \lambda x_i + (1-\lambda) x_k$. Theoretically, it regularizes outside the training distribution [5, 26, 87] and applies label smoothing [53, 61].

CutMix draws from Mixup and CutOut [13] by pasting a patch from x_k onto x_i: $m_x(x_i, x_k, \lambda) = \mathbb{1}_m \odot x_i +$

$(\mathbb{1} - \mathbb{1}_m) \odot x_k$ where $\odot$ represents the element-wise product and $\mathbb{1}_m$ a binary mask with average value λ. CutMix randomly samples squares, which often leads to rectangular masks due to boundary effects. Such **non-linear binary masking** improves generalization [68, 70] by increasing dataset: it creates new images with usually disjoint patches [29]. [3, 17] seek more diverse transformations via arbitrarily shaped masks: proposals range from cow-spotted masks [21] to masks with irregular edges [29]. As masking of discriminative regions may cause label misallocation [26], [41, 74] try to alleviate this issue with costly saliency heatmaps [65]. Yet, ResizeMix [63] shows that they perform no better than random selection of patch locations.

In addition to Manifold Mixup [76], only a few works [17, 46, 81, 83] have tried to mix intermediate **latent features** as we do. Our goals and methods are however quite different, as shown later in Section 3.4. In brief, they mix deep features to smooth the decision boundaries, while we mix shallow features only so that inputs can remain distinct.

2.2. Ensembling

Like [79], we explore combining DA with another standard technique in machine learning: ensembling [14, 28]. For improved performances, aggregated members should be both *accurate* and *diverse* [57, 62, 64]. Deep ensembles [43] (DE) simultaneously train multiple networks with different random initializations converging towards different explanations for the training data [18, 80].

Ensembling's fundamental drawback is the inherent **computational and memory overhead**, which increases linearly with the number of members. This bottleneck is typically addressed by sacrificing either *individual* performance or *diversity* in a complex *trade-off*. Averaging predictions from several checkpoints on the training process, *i.e.* snapshot ensembles [37, 39], fails to explore multiple local optima [2, 18, 80]. So does Monte Carlo Dropout [22]. The recent BatchEnsemble [16] is parameter-efficient, yet requires multiple forward passes. TreeNets [45, 69] reduce training and inference cost by sharing low-level layers. MotherNets [78] share first training epochs between members. However, sharing reduces diversity.

Very recently, the multi-input multi-output MIMO [30] achieves **ensemble almost "for free"**: all of the layers except the first convolutional and last dense layers are shared ($\approx +1\%$ #parameters). [66] motivated a related Aggregated Learning to learn concise representations with arguments from information bottleneck [71]. The idea is that over-parameterized CNNs [19, 52, 60] can fit multiple subnetworks [75]. The question is how to prevent homogenization among the simultaneously trained subnetworks. Facing a similar challenge, [23] includes stochastic channel recombination; [15] relies on predefined binary masks; in GradAug [82], subnetworks only leverage the first channels up to a given percentage. In contrast, MIMO does not need structural differences among subnetworks: they learn to build their own paths while being as diverse as in DE.

3. MixMo framework

We first introduce the main components of our MixMo strategy, summarized in Fig. 3: we mix multiple inputs to obtain multiple outputs via subnetworks. We highlight the key mixing block combining information from inputs, and our training loss based on a dedicated weighting scheme.

We mainly study $M = 2$ subnetworks here, both for clarity and as it empirically performs best in standard parameterization regimes. For completeness, we straightforwardly generalize to $M > 2$ in Section 3.5.

3.1. General overview

We leverage a training classification dataset D of i.i.d. pairs of associated image/label $\{x_i, y_i\}_{i=1}^{|D|}$. We randomly sample a subset of $|B|$ samples $\{x_i, y_i\}_{i\in B}$ that we randomly shuffle via permutation π. Our training batch is $\{(x_i, x_j), (y_i, y_j)\}_{i\in B, j=\pi(i)}$. The loss $\mathcal{L}_{\text{MixMo}}$ is averaged over these $|B|$ samples: the networks' weights are updated through backpropagation and gradient descent.

Let's focus on the training sample $\{(x_0, x_1), (y_0, y_1)\}$. In MixMo, both inputs are **separately encoded** (see Fig. 1) into the shared latent space with two different convolutional layers (with 3 input channels each and no bias term): x_0 via c_0 and x_1 via c_1. To recover a strictly equivalent formulation to MIMO [30], we simply sum the two encodings: $c_0(x_0) + c_1(x_1)$. Indeed, MIMO merges inputs through channel-wise concatenation in pixels: MIMO's first convolutional layer (with 6 input channels and no bias term) hides the summing operation in the output channels.

Explicitly highlighting the underlying mixing leads us to consider a **generalized multi-input mixing block** $\mathcal{M}$. This manifold mixing presents a unique opportunity to tackle the ensemble diversity/individual accuracy trade-off and to improve overall ensemble results (see Section 3.2). The shared representation $\mathcal{M}\left(c_0(x_0), c_1(x_1)\right)$ feeds the next convolutional layers. We note κ the **mixing ratio** between inputs.

The core network $\mathcal{C}$ handles features that represent both inputs simultaneously. The dense layer d_0 predicts $\hat{y}_0 = d_0\left[\mathcal{C}\left(\mathcal{M}\left\{c_0(x_0), c_1(x_1)\right\}\right)\right]$ and targets y_0, while d_1 targets y_1. Thus, the **training loss** is the sum of two cross-entropies $\mathcal{L}_{\text{CE}}$ weighted by parametrized function w_r (defined in Section 3.3) to balance the asymmetry when $\kappa \neq 0.5$:

$$\mathcal{L}_{\text{MixMo}} = w_r(\kappa)\mathcal{L}_{\text{CE}}\left(y_0, \hat{y}_0\right) + w_r(1-\kappa)\mathcal{L}_{\text{CE}}\left(y_1, \hat{y}_1\right). \quad (1)$$

At inference, the same input x is repeated twice: the core network $\mathcal{C}$ is fed the sum $c_0(x) + c_1(x)$ that preserves maximum information from both encodings. Then, the diverse predictions are averaged: $\frac{1}{2}\left(\hat{y}_0 + \hat{y}_1\right)$. This allows us to benefit from ensembling in a single forward pass.

3.2. Mixing block $\mathcal{M}$

The mixing block $\mathcal{M}$ - which combines both inputs into a shared representation - is the cornerstone of MixMo. Our main intuition was to analyze MIMO as a simplified Mixup variant where the mixing ratio κ is fixed to 0.5. MixMo generalized framework encompasses a wider range of variants inspired by MSDA mixing methods. Our first main variant - Linear-MixMo - fully extends Mixup. The mixing block is $\mathcal{M}_{\text{Linear-MixMo}}(l_0, l_1) = 2[\kappa l_0 + (1-\kappa) l_1]$, where $l_0 = c_0(x_0)$, $l_1 = c_1(x_1)$ and $\kappa \sim \text{Beta}(\alpha, \alpha)$ with α the concentration parameter. The second and more effective variant **Cut-MixMo** adapts the patch mixing from CutMix:

$$\mathcal{M}_{\text{Cut-MixMo}}(l_0, l_1) = 2\left[\mathbb{1}_{\mathcal{M}} \odot l_0 + (\mathbb{1} - \mathbb{1}_{\mathcal{M}}) \odot l_1\right], \quad (2)$$

where $\mathbb{1}_{\mathcal{M}}$ is a binary mask with area ratio $\kappa \sim \text{Beta}(\alpha, \alpha)$, valued at 1 either on a rectangle or on the complementary of a rectangle. In brief, a patch from $c_0(x_0)$ is pasted onto $c_1(x_1)$, or vice versa. This binary mixing in Cut-MixMo advantageously replaces the linear interpolation in MIMO and Linear-MixMo: subnetworks are more accurate and more diverse, as shown empirically in Fig. 7.

First, binary mixing in $\mathcal{M}$ trains stronger **individual** subnetworks for the same reasons why CutMix improves over Mixup. In a nutshell, linear MSDAs [76, 86] produce noisy samples [5] that lead to robust representations. As MixMo tends to distribute different inputs on non-overlapping channels (as discussed later in Fig. 4a), this regularization hardly takes place anymore in $\mathcal{M}_{\text{Linear-MixMo}}$. On the contrary, by masking features, we simulate common object occlusion problems. This spreads subnetworks' focus across different locations: the two classifiers are forced to find information relevant to their assigned input at disjoint locations. This occlusion remains effective as the receptive field in this first shallow latent space remains small.

Secondly, linear interpolation is fundamentally ill-suited to induce diversity as full information is preserved from both inputs. CutMix on the other hand explicitly increases dataset diversity by presenting patches of images that do not normally appear together. Such benefits can be directly transposed to $\mathcal{M}_{\text{Cut-MixMo}}$: binary mixing with patches increases randomness and **diversity between the subnetworks**. Indeed, in a similar spirit to bagging [4], different samples are given to the subnetworks. By deleting asymmetrical complementary locations from the two inputs, subnetworks will not rely on the same region and information. Overall, they are less likely to collapse on close solutions.

3.3. Loss weighting w_r

Asymmetries in the mixing mechanism can cause one input to overshadow the other. Notably when $\kappa \neq 0.5$, the predominant input may be easier to predict. We seek a weighting function w_r to **balance the relative importance**

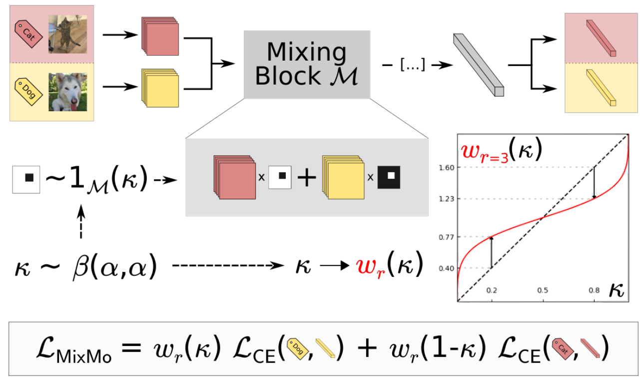


Figure 3: **Cut-MixMo training**. We sample a mixing mask given κ, and balance the losses with $w_r(\kappa)$ from Eq. 3.

of the two $\mathcal{L}_{\text{CE}}$ in $\mathcal{L}_{\text{MixMo}}$. This weighting modifies the effective learning rate, how gradients flow in the network and overall how mixed information is represented in features. In this paper, we propose to weight via the parametrized:

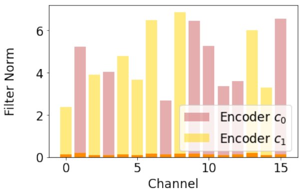
$$w_r(\kappa) = 2\frac{\kappa^{1/r}}{\kappa^{1/r} + (1-\kappa)^{1/r}}. \quad (3)$$

This defines a family of functions indexed by the parameter r, visualized for $r = 3$ in red on Fig. 3. See Appendix 6.1 for complementary visualizations. This power law provides a natural relaxation between **two extreme configurations**. The first extreme, $r = 1$, $w_1(\kappa) = 2\kappa$, is in line with linear label interpolation in MSDA. The resulting imbalance in each subnetwork's contribution to $\mathcal{L}_{\text{MixMo}}$ causes lopsided updates. While it promotes diversity, it also reduces regularization: the overshadowed input has a reduced impact on the loss. The opposite extreme, $r \rightarrow \infty$, $w_\infty(\kappa) \rightarrow 1$, removes reweighting. Consequently, w_r inflates the importance of hard under-represented inputs, *à la* Focal Loss [49]. However, minimizing the role of the predominant inputs destabilizes training. Overall, we empirically observe that moderate values of r perform best as they trade off pros and cons from both extremes.

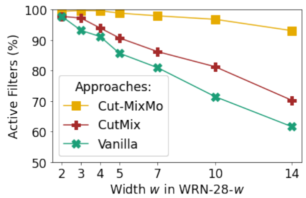
Interestingly, the proper weighting of loss components is also a central theme in **multi-task learning** [6, 8]. While it aims at predicting several tasks from a shared input, MixMo predicts a shared task from several different inputs. Beyond this inverted structure, we have similar issues: *e.g.* gradients for one task can be detrimental to another conflicting task. Fortunately, MixMo presents an advantage: the exact ratios κ and $1 - \kappa$ of each task are known exactly.

3.4. From manifold mixing to MixMo

We have discussed at length how we extend multi-input multi-output frameworks by borrowing mixing protocols from MSDA. Now we reversely point out how our **MixMo diverges from MSDA schemes**. At first glimpse, the idea



(a) Filters l_1-norms of the input encoders c_0 and c_1.



(b) Proportion of active filters in the core network vs. width w.

Figure 4: **Influence of MixMo on network utilization**. (a) The encoders have separate channels: the two subsequent classifiers can differentiate the two inputs. (b) Less filters are strongly active ($\|f_i\|_1 \geq 0.4\times \max_{f\in\text{layer}}\|f\|_1$) in wider networks: Cut-MixMo reduces this negative point.

is the same as manifold mixing [17, 46, 76]: $M = 2$ inputs are encoded into a latent space to be mixed before being fed to the rest of the network. Yet, while they mix at varying depths, we only mix in the shallowest space. Specifically, we only mix in features - and not in pixels - to allow separate encodings of the inputs: they need to remain distinct in the mixed representation for the subsequent classifiers.

Hence our two key differences: *first*, MixMo uses two separated encoders (one for each input), and *second*, it outputs two predictions instead of a single one. Indeed, MSDAs use a single classifier that targets a unique soft label reflecting the different classes via linear interpolation. MixMo instead chooses to fully leverage the composite nature of mixed samples and trains **separated dense layers**, d_0 and d_1, ensembled "for free" at test time.

Section 4.3.5 demonstrates that MixMo works because it also uses two **different encoders** c_0 and c_1. While training two classifiers may seem straightforward in MSDA, it actually raises a troubling question: which input should each classifier predicts ? Having two encoders provides a simple solution: the network is divided in two subnetworks, one for each input. Their separability is easily observed: Fig. 4a shows the l_1-norm of the 16 filters for the two encoders (WRN-28-10 on CIFAR-100). Each filter norm is far from zero in only one of the two encoders: $c_0(x_0)$ and $c_1(x_1)$ separate the inputs in different dimensions which allows subsequent layers to treat them differently.

This leads MixMo to use most available filters. Following the structured pruning literature [47], we consider in Fig. 4b that a filter (in a layer of the core network) is active if its l_1-norm is at least 40% of the l_1-norm from its layer's most active filter (see Appendix 6.2). This illustrates the known increase in sparsity in wider networks. Conversely, having 2 subnetworks in MixMo enables the weights ignored by one subnetwork to be leveraged by the other.

3.5. Generalization to $M \geq 2$ subnetworks

Most of the framework is easily extended by optimizing $\mathcal{L}_{\text{MixMo}} = \sum_{0\leq i<M} M \frac{\kappa_i^{1/r}}{\sum_j \kappa_j^{1/r}} \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i)$ with $\{\kappa_i\} \sim \text{Dir}(\alpha)$ from a Dirichlet distribution (see Appendix 6.3). The key change is that $\mathcal{M}$ now needs to handle more than 2 inputs: $\{c_i(x_i)\}_{0\leq i<M}$. While linear interpolation is easily generalized, Cut-MixMo has several possible extensions: in our experiments, we first linearly interpolate between $M-1$ inputs and then patch in a region from the M-th.

4. Experiments

We evaluate MixMo efficiency on standard image classification datasets: CIFAR-{10,100} [42] and Tiny ImageNet [10]. We equally track accuracies (Top{1,5}, $\uparrow$) and the *calibrated Negative Log-Likelihood* (NLL$_c$, $\downarrow$). Indeed, [2] shows that we should compare in-domain uncertainty estimations after temperature scaling (TS) [25]: we thus split the test set in two and calibrate (after averaging in ensembles) with the temperature optimized on the other half, as in [50, 64]. We nonetheless report NLL (without TS) along with the Expected Calibration Error [54] in Appendix 6.5.

4.1. Implementation details

We mostly study the Linear-MixMo and Cut-MixMo variants with M=2. We set **hyper-parameter** r=3 (see Section 4.3.3). α=2 performs better than 1 (see Appendix 6.8). In contrast, MIMO [30] refers to linear summing, like Linear-MixMo, but with κ=0.5 instead of $\kappa \sim \text{Beta}(\alpha, \alpha)$.

Different mixing methods create a strong **train-test distribution gap** [5, 51]. Thus, in Cut-MixMo we actually substitute $\mathcal{M}_{\text{Cut-MixMo}}$ for $\mathcal{M}_{\text{Linear-MixMo}}$ with probability $1-p$ to accommodate for the summing in $\mathcal{M}$ at inference. We set the probability of patch mixing during training to p=0.5, with linear descent to 0 over the last twelfth of training epochs (see pseudocode 1 in Appendix).

When MixMo is combined with CutMix, the pixels inputs are: $(m_x(x_i, x_k, \lambda), m_x(x_j, x_{k'}, \lambda'))$ with interpolated targets $(\lambda y_i + (1-\lambda)y_k, \lambda' y_j + (1-\lambda')y_{k'}))$, where k, k' are randomly sampled and $\lambda, \lambda' \sim \text{Beta}(1, 1)$.

MIMO duplicates samples b times via **batch repetition**: x_i will be associated with $x_{\pi(i)}$ and $x_{\pi'(i)}$ in the same batch if b=2. As the batch size remains fixed, the count of unique samples per batch and the learning rate is divided by b. Conversely, the number of steps is multiplied by b. Overall, this stabilizes training but multiplies its cost by b. We thus indicate an estimated (training/inference) overhead (wrt. vanilla training) in the *time* column of our tables. Note that some concurrent approaches also lengthen training: *e.g.* GradAug [82] via multiple subnetworks predictions ($\approx \times 3$).

We provide more details in Appendix 6.4 and will open source our PyTorch [59] implementation.

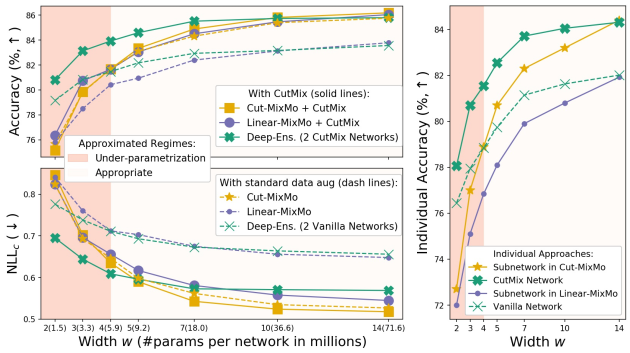
Table 1: **Main results**: WRN-28-10 on CIFAR. **Bold** highlights best scores, † marks approaches not re-implemented.

Dataset		CIFAR-100			CIFAR-10	
Approach	Time Tr./Inf.	Top1 %, ↑	Top5 %, ↑	NLL_c 10^{-2}, ↓	Top1 %, ↑	NLL_c 10^{-2}, ↓
Vanilla	1/1	81.63	95.49	73.9	96.34	12.6
Mixup		83.44	95.92	65.7	97.07	11.2
Manifold Mixup†		81.96	95.51	73.4	97.45	12.2
CutMix		84.05	96.09	64.8	97.23	9.9
ResizeMix†		84.31	-	-	97.60	-
Puzzle-Mix†	2/1	84.31	96.46	66.8	-	-
GradAug†	3/1	84.14	96.43	-	-	-
+ CutMix†		85.51	96.86	-	-	-
Mixup BA†	7/1	84.30	-	-	**97.80**	-
DE (2 Nets)	2/2	83.17	96.37	66.4	96.67	11.1
+ CutMix		85.74	96.82	57.1	97.52	8.6
MIMO	2/1	82.40	95.78	68.8	96.38	12.1
Linear-MixMo		82.54	95.99	67.6	96.56	11.4
+ CutMix		84.69	97.12	57.2	97.32	9.4
Cut-MixMo		84.38	96.94	56.3	97.31	8.9
+ CutMix		85.18	97.20	54.5	97.45	8.4
MIMO	4/1	83.06	96.23	66.1	96.74	11.4
Linear-MixMo		83.08	96.26	65.6	96.91	10.8
+ CutMix		85.47	97.04	55.8	97.68	8.7
Cut-MixMo		85.40	97.22	53.5	97.51	8.1
+ CutMix		**85.77**	**97.42**	**52.4**	97.73	**7.9**

4.2. Main results on CIFAR-100 and CIFAR-10

Tab. 1 reports averaged scores over 3 runs for our main experiment on CIFAR with WRN-28-10 [84]. We re-use the hyper-parameters given in MIMO [30]. Cut-MixMo reaches (85.40% Top1, 0.535 NLL_c) on CIFAR-100 with b=4: it surpasses our Linear-MixMo (83.08%, 0.656) and MIMO (83.06%, 0.661). Cut-MixMo sets a new state of the art when combined with CutMix (85.77%, 0.524). Results remain strong when b=2: Cut-MixMo (84.38%, 0.563) proves better on its own than traditional DE [43], and MSDAs like MixUps [86, 76] or the stronger CutMix variant [83]. On CIFAR-10, we see similar trends: Cut-MixMo reaches 0.081 in NLL_c, 0.079 with CutMix. Yet, the costlier batch augmented Mixup BA [36] edges it out in Top1.

Fig. 5 shows how MixMo grows stronger than DE (green curves) as width w in WRN-28-w increases. The parameterization becomes appropriate at w=4: Cut-MixMo (yellow curves) then matches DE - with half the parameters - in Fig. 5a and its subnetworks match a vanilla network in Fig. 5b. Beyond, MixMo better uses over-parameterization: Cut-MixMo+CutMix surpasses DE+CutMix in NLL_c for $w \geq 5$, and this is true in Top1 for $w \geq 10$. Compared to our strong Linear-MixMo+CutMix (purple curves), Cut-MixMo performs similarly in Top1, and better with CutMix for $w \geq 4$. While Linear-MixMo and DE learn from occlusion, Cut-MixMo also benefits from CutMix, notably from the induced label smoothing. Overall, Cut-MixMo, even without CutMix, significantly better estimates uncertainty.



(a) Ensemble Top1 and NLL_c. (b) Individual Top1.

Figure 5: **Parameters efficiency** (metrics/#params). CIFAR-100 with WRN-28-w, b=4. Comparisons between (a) ensemble and some of their (b) individual counterparts.

4.3. MixMo analysis on CIFAR-100 w/ WRN-28-10

4.3.1 Training time

We have just seen that CutMix improves Linear-MixMo at varying widths w, but not enough to match Cut-MixMo in NLL_c: CutMix can not fully compensate for the advantages from patch mixing over linear interpolation. We recover this finding in Fig. 6, this time at varying batch repetition $b \in \{1, 2, 4\}$ when w=10. Moreover, Cut-MixMo outperforms DE for the **same training time**. Indeed, MixMo variants trained with a given b matches the training time of DE with N=b networks. In the rest of this section, we set b=2.

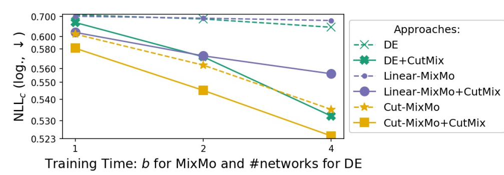


Figure 6: **NLL_c(↓) improves with longer training**, via batch repetitions (MixMo) or additional networks (DE).

4.3.2 The mixing block $\mathcal{M}$

Tab. 2 compares performance for several mixing blocks [17, 29, 68, 83]. No matter the **shape** (illustrated in Appendix 6.7), binary masks perform better than linear mixing: the cow-spotted mask (84.17%, 0.561) [20, 21] notably performs well. The basic CutMix patching (84.38%, 0.563) is nevertheless more accurate and was our main focus.

Table 2: $\mathcal{M}$ inspired by various **MSDA approaches**.

$\mathcal{M}$ approach	Mixup [86]	Horiz. Concat.	Vertical Concat.	PatchUp 2D [17]	FMix [29]	CowMask [20, 21]	CutMix [83]
Top1 ↑	82.5	82.78	84.00	84.16	83.76	84.17	**84.38**
NLL_c ↓	0.676	0.627	0.573	0.581	0.602	**0.561**	0.563

We further study the impact of patch mixing through the lens of the *ensemble diversity/individual accuracy trade off*. As in [64], we measure diversity via the pairwise ratio-error [1] (d_{re}, $\uparrow$), defined as the ratio between the number of different errors and simultaneous errors for two predictors. In Fig. 7 and 8, we average metrics over the last 10 epochs.

As argued in Section 3.2, patch mixing increases diversity compared to linear mixing in Fig. 7. As the probability p of patch mixing grows, so does diversity: from $d_{re}(p{=}0.0){\approx}0.78$ (Linear-MixMo) to $d_{re}(p{=}0.5){\approx}0.85$ (Cut-MixMo). We provide associated training dynamics in Appendix 6.6. In contrast, DE has $d_{re}{\approx}0.76$ while MIMO has $d_{re}{\approx}0.77$ on the same setup. Increasing p past 0.6 boosts diversity even more at the cost of subnetworks' accuracies: this is due to underfitting and an increased test-train distribution gap. $p \in [0.5, 0.6]$ is thus the best trade off.

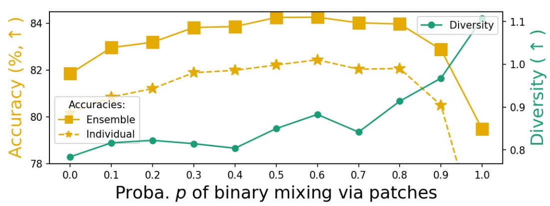


Figure 7: **Diversity/accuracy** as function of p with $r{=}3$.

4.3.3 Weighting function w_r

We analyze the impact of the parameter r in the reweighting function w_r. Higher values tend to remove reweighting, as shown in Appendix 6.1: they strongly decrease diversity in Fig. 8. The opposite extreme with $r{=}1$ increases diversity via lopsided gradient updates but it degrades accuracy. We speculate it under-emphasizes hard samples. The range $r \in [3, 6]$ strikes a good balance: results remain high and stable.

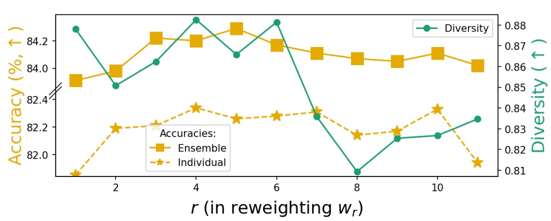


Figure 8: **Diversity/accuracy** as function of r with $p{=}0.5$.

4.3.4 Generalization to $M \geq 2$ subnetworks

We try to generalize MixMo to more than $M = 2$ subnetworks in Fig. 9. Cut-MixMo's subnetworks perform at 82.3% when $M{=}2$ vs. 79.5% when $M{=}3$. In MIMO, it's 79.8% vs. 77.7%. Because subnetworks do not share features, higher M degrades their results: only two can fit seamlessly. Ensemble Top1 overall decreases in spite of the additional predictions, as already noticed in MIMO [30].

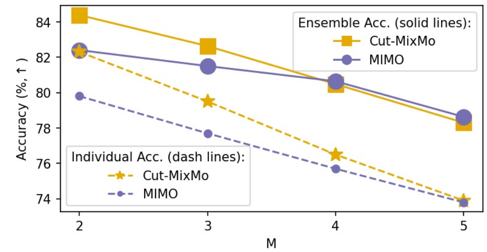


Figure 9: **Ensemble/individual** accuracies for $M \geq 2$.

This reflects MixMo's strength in over-parametrized regimes, but also its limitations with fewer parameters when subnetworks underfit (recall previous Fig. 5). Facing similar findings, MIMO [30] introduced input repetition so that subnetworks share their features, at the cost of drastically reducing diversity. Our generalization may be extended by future approaches whose mixing blocks (perhaps not inspired by MSDA) would tackle these issues.

4.3.5 Multiple encoders and classifiers

In Section 3.4, we compared MixMo and MSDA. Tab. 3 confirms the need for **2 encoders and 2 classifiers**. With 1 classifier and linearly interpolated labels (in the same spirit as [7]), the 2 encoders perform worse than 1 encoder. With 1 shared encoder and 2 classifiers, it is not clear which input each classifier should target. In the first naive $\ominus$, we randomly associate the 2 classifiers and the 2 inputs (encoded with the same encoder). This $\ominus$ variant yields poor results. In $\otimes$, the first classifier tries to predict the label from the predominant input, the second targets the other input: $\otimes$ reaches 0.598 vs. 0.563 for Cut-MixMo.

Table 3: Number of encoders/classifiers.

# Enc.	# Clas.	NLL$_c$ $\downarrow$
1	1	0.604
2	1	0.666
1	$2^{\ominus}$	0.687
1	$2^{\otimes}$	0.598
2	2	**0.563**

4.4. Robustness to image corruptions

Deep networks' results decrease when facing unfamiliar samples. To measure robustness to train-test distribution gaps, [34] corrupted CIFAR-100 test images into CIFAR-100-c (more details in Appendix 6.4). As in Puzzle-Mix [41], we report WRN-28-10 results with and without AugMix [35], a pixels data augmentation technique specifically introduced for this task. Tab. 4 shows that Cut-MixMo ($b{=}4$) best complements AugMix and reaches 71.1% Top1.

Table 4: **Robustness comparison on CIFAR-100-c**.

Approach	1 Net.		CutMix	Puzzle-Mix†		DE (2 Nets)		MIMO	Linear-MixMo		Cut-MixMo	
AugMix	-	✓	-	-	✓	-	✓	-	-	✓	-	✓
Top1 $\uparrow$	52.2	67.8	51.93	58.09	70.46	53.8	69.9	53.6	55.6	70.4	57.0	**71.1**
Top5 $\uparrow$	73.7	87.5	72.03	77.3	87.7	74.9	88.9	74.9	76.1	89.4	77.4	**89.5**
NLL $\downarrow$	2.50	1.38	2.13	1.96	1.34	2.27	1.24	2.66	2.33	1.22	2.04	**1.16**

4.5. Pushing MixMo further: Tiny ImageNet

At a larger scale and with more varied 64×64 images, Cut-MixMo reaches a new state of the art of 70.24% on Tiny ImageNet [10] in Tab. 5. We re-use the hyper-parameters given in previous state of the art Puzzle-Mix [41]. With w=1, PreActResNet-18 [32] is not sufficiently parametrized for MixMo's advantages to express themselves on this challenging dataset. MixMo's full potential shines with wider networks: with w=2 and 44.9M parameters, Cut-MixMo reaches (69.13%, 1.28) vs. (67.76%, 1.33) for CutMix. Compared to DE with 3 networks, Cut-MixMo performs {worse, similarly, better} for width $w \in \{1, 2, 3\}$. At (almost) the same numbers of parameters, Cut-MixMo when w=2 performs better (69.13%, 1.28) than DE with 4 networks when w=1 (67.51%, 1.31).

Table 5: **Results**: PreActResNet-18-w on Tiny ImageNet.

Width w (# params)		$w=1$ (11.2M)		$w=2$ (44.9M)		$w=3$ (100.5M)	
Approach	Time Tr./Inf.	Top1 %,↑	NLL_c ↓	Top1 %,↑	NLL_c ↓	Top1 %,↑	NLL_c ↓
Vanilla	1/1	62.56	1.53	64.80	1.51	65.78	1.53
Mixup	1/1	63.74	1.62	66.62	1.50	67.27	1.51
Manifold Mixup†	1/1	58.70	1.92	-	-	-	-
Co-Mixup†	1/1	64.15	-	-	-	-	-
CutMix	1/1	65.09	1.58	67.76	1.33	68.95	1.29
Puzzle-Mix†	2/1	64.48	1.65	-	-	-	-
DE (2 Nets)	2/2	65.53	1.39	68.06	1.37	68.38	1.36
DE (3 Nets)	3/3	66.76	1.34	69.05	1.29	69.36	1.28
DE (4 Nets)	4/4	**67.51**	**1.31**	**69.94**	**1.24**	69.72	1.26
Linear-MixMo	2/1	61.58	1.61	66.62	1.41	68.18	1.36
Cut-MixMo	2/1	63.78	1.48	68.30	1.30	69.89	1.26
Linear-MixMo	4/1	62.91	1.51	67.03	1.41	68.38	1.38
Cut-MixMo	4/1	64.44	1.48	69.13	1.28	**70.24**	**1.19**

4.6. Ensemble of MixMo

Since MixMo adds very little parameters ($\approx +1\%$), we can combine independently trained MixMo like in DE. This ensembling of ensemble of subnetworks leads in practice to the averaging of $M \times N = 2 \times N$ predictions. Fig. 10 compares ensembling for vanilla networks and Cut-MixMo on CIFAR-100. We first recover the Memory Split Advantage [9, 50, 77, 88] (MSA): at similar parameter counts, N=5 vanilla WRN-28-3 do better than a single vanilla WRN-28-7 (+0.10 in NLL_c). **Cut-MixMo challenges this MSA**: we bridge the gap between using one network or several smaller networks (-0.04 on same setup). Visually, Cut-MixMo's curves remain closer to the lower envelope: performances are less dependent on how the memory budget is split. This is because Cut-MixMo is effective mainly for larger architectures by better leveraging their parameters.

We also recover that wide vanilla networks tend to be less diverse [55], and thus gain less from ensembling [50]: N=2 vanilla WRN-28-14 (83.47% Top1, 0.656 NLL_c) perform not much better than N=2 WRN-28-7 (82.94%, 0.673). Contrarily, **Cut-MixMo facilitates the ensembling of large networks** with (86.58%, 0.488) vs. (85.50%, 0.516) (more comparisons in Appendix 6.10).

When combined with CutMix [83], Cut-MixMo previously set a new state of the art of 85.77% with N=1 WRN-28-10. Final Tab. 6 shows it further reaches 86.63% with N=2 and even 86.81% with N=3.

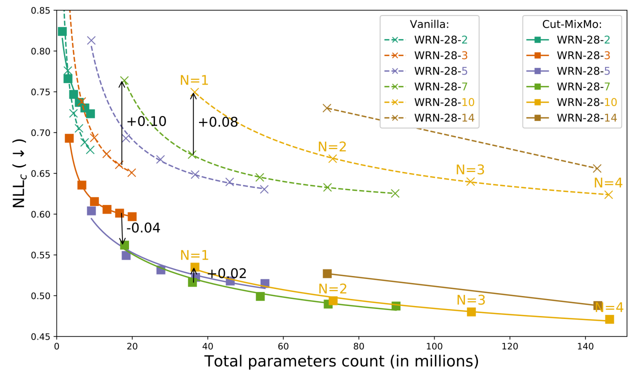


Figure 10: **Ensemble effectiveness** (NLL_c/#params), for different widths w in WRN-28-w and numbers of members N. Standard data augmentations on CIFAR-100 with b=4. Curves interpolated through power laws [50].

Table 6: **Best results for WRN-28-10 on CIFAR-100** via Cut-MixMo + CutMix [83] + N-ensembling and b=4. Recent Top1 SoTAs: 85.23 [63], 85.51 [82], 85.74 [88].

N	# params	Average Top1 ↑	Average Top5 ↑	Average NLL_c ↓	Best run Top1 ↑	Best run Top5 ↑	Best run NLL_c ↓
1	36.6M	85.77 $\pm$ 0.14	97.36 $\pm$ 0.02	0.524 $\pm$ 0.005	85.92	97.36	0.518
2	73.2M	86.63 $\pm$ 0.19	97.73 $\pm$ 0.05	0.479 $\pm$ 0.003	86.75	97.80	0.475
3	109.8M	86.81 $\pm$ 0.17	97.85 $\pm$ 0.04	0.464 $\pm$ 0.002	86.94	97.83	0.464

5. Conclusion

We introduce the MixMo framework that generalizes the multi-input multi-output ensembling strategy. MixMo can be analyzed as either an ensembling method or a mixed samples data augmentation, while remaining complementary to works from both lines of research. Finally, MixMo better exploits wide networks and improves the state of the art on CIFAR-100, CIFAR-100-c and Tiny ImageNet.

Acknowledgments

This work was performed using HPC resources from GENCI-IDRIS (Grant 2021-AD011012262), with financial supports from the ANR agency in the chair VISA-DEEP (ANR-20-CHIA-0022-01), and from Rémy's CIFRE grant between Thales Land and Air Systems and Sorbonne University. We thank Andrei Bursuc for his detailed feedbacks.

References

[1] Matti Aksela. Comparison of classifier selection methods for improving committee performance. In *MCS*, 2003.

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

[3] Kyungjune Baek, Duhyeon Bang, and Hyunjung Shim. Gridmix: Strong regularization through local context mapping. *Pattern Recognition*, 2021.

[4] Leo Breiman. Bagging predictors. *Machine learning*, 1996.

[5] Luigi Carratino, Moustapha Cissé, Rodolphe Jenatton, and Jean-Philippe Vert. On mixup regularization. *ArXiv preprint*, 2020.

[6] Rich Caruana. Multitask learning. *Machine learning*, 1997.

[7] John Chen, Samarth Sinha, and Anastasios Kyrillidis. Imclr: Implicit contrastive learning for image classification. *ArXiv preprint*, 2020.

[8] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, 2018.

[9] Nadezhda Chirkova, Ekaterina Lobacheva, and Dmitry P. Vetrov. Deep ensembles on a fixed memory budget: One wide network or several thinner ones? *ArXiv preprint*, 2020.

[10] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *ArXiv preprint*, 2017.

[11] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In *NeurIPS*, 2020.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[13] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv preprint*, 2017.

[14] Thomas G Dietterich. Ensemble methods in machine learning. In *MCS*, 2000.

[15] Nikita Durasov, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Masksembles for uncertainty estimation. *ArXiv preprint*, 2020.

[16] 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.

[17] Mojtaba Faramarzi, Mohammad Amini, Akilesh Badrinaaraayanan, Vikas Verma, and Sarath Chandar. Patchup: A regularization technique for convolutional neural networks. *ArXiv preprint*, 2020.

[18] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *ArXiv preprint*, 2019.

[19] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[20] Geoff French, Timo Aila, Samuli Laine, Michal Mackiewicz, and Graham Finlayson. Semi-supervised semantic segmentation needs strong, high-dimensional perturbations. *ArXiv preprint*, 2019.

[21] Geoff French, Avital Oliver, and Tim Salimans. Milking cowmask for semi-supervised image classification. *ArXiv preprint*, 2020.

[22] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[23] Yuan Gao, Zixiang Cai, and Lei Yu. Intra-ensemble in neural networks. *ArXiv preprint*, 2019.

[24] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *NeurIPS*, 2018.

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

[26] Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. In *AAAI*, 2019.

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

[28] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 1990.

[29] Ethan Harris, Antonia Marcu, Matthew Painter, Mahesan Niranjan, Adam Prügel-Bennett, and Jonathon Hare. Fmix: Enhancing mixed sample data augmentation. *ArXiv preprint*, 2020.

[30] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Liu, Jasper Roland Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *ICLR*, 2021.

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

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[33] Zhuoxun He, Lingxi Xie, Xin Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Data augmentation revisited: Rethinking the distribution gap between clean and augmented data. *ArXiv preprint*, 2019.

[34] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[35] Dan Hendrycks, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *ICLR*, 2019.

[36] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *CVPR*, 2020.

[37] 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.

[38] Hiroshi Inoue. Data augmentation by pairing samples for images classification. *ArXiv preprint*, 2018.

[39] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *UAI*, 2018.

[40] JangHyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. Co-mixup: Saliency guided joint mixup with supermodular diversity. In *ICLR*, 2021.

[41] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *ICML*, 2020.

[42] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[43] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, 2017.

[44] Yann Lecun, J. S. Denker, Sara A. Solla, R. E. Howard, and L.D. Jackel. Optimal brain damage. In *NeurIPS*, 1990.

[45] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *ArXiv preprint*, 2015.

[46] Boyi Li, Felix Wu, Ser-Nam Lim, Serge Belongie, and Kilian Q. Weinberger. On feature normalization and data augmentation. *ArXiv preprint*, 2020.

[47] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[48] Daojun Liang, Feng Yang, Tian Zhang, and Peter Yang. Understanding mixup training methods. *IEEE Access*, 2018.

[49] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

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

[51] Raphael Gontijo Lopes, Sylvia J. Smullin, Ekin D. Cubuk, and Ethan Dyer. Affinity and diversity: Quantifying mechanisms of data augmentation. In *CoRR*, 2020.

[52] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. In *ICLR*, 2017.

[53] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? In *NeurIPS*, 2019.

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

[55] 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*, 2018.

[56] Jeremy Nixon, Michael W Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran. Measuring calibration in deep learning. In *CVPR workshop*, 2019.

[57] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 1999.

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

[59] 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.

[60] Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subsetsum: Logarithmic over-parameterization is sufficient. In *NeurIPS*, 2020.

[61] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR workshop*, 2017.

[62] Michael Perrone and Leon Cooper. When networks disagree: Ensemble methods for hybrid neural networks. *Neural networks for speech and image processing*, 1993.

[63] Jie Qin, Jiemin Fang, Qian Zhang, Wenyu Liu, Xingang Wang, and Xinggang Wang. Resizemix: Mixing data with preserved object information and true labels. *ArXiv preprint*, 2020.

[64] Alexandre Rame and Matthieu Cord. Dice: Diversity in deep ensembles via conditional redundancy adversarial estimation. In *ICLR*, 2021.

[65] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

[66] Masoumeh Soflaei, Hongyu Guo, Ali Al-Bashabsheh, Yongyi Mao, and Richong Zhang. Aggregated learning: A vector-quantization approach to learning neural network classifiers. In *AAAI*, 2020.

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[68] Cecilia Summers and Michael J Dinneen. Improved mixed-example data augmentation. In *WACV*, 2019.

[69] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[70] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020.

[71] Naftali Tishby. The information bottleneck method. In *Allerton Conference on Communication, Control and Computation*, 2001.

[72] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *CVPR*, 2018.

[73] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Learning from between-class examples for deep sound recognition. In *ICLR*, 2018.

[74] A. F. M. Shahab Uddin, Mst. Sirazam Monira, Wheemyung Shin, TaeChoong Chung, and Sung-Ho Bae. Saliencymix: A saliency guided data augmentation strategy for better regularization. In *ICLR*, 2021.

[75] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

[76] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, 2019.

[77] Xiaofang Wang, Dan Kondratyuk, Kris M. Kitani, Yair Movshovitz-Attias, and Elad Eban. Multiple networks are more efficient than one: Fast and accurate models via ensembles and cascades. *ArXiv preprint*, 2020.

[78] Abdul Wasay, Brian Hentschel, Yuze Liao, Sanyuan Chen, and Stratos Idreos. Mothernets: Rapid deep ensemble learning. In *MLSys*, 2020.

[79] Yeming Wen, Ghassen Jerfel, Rafael Muller, Michael W Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran. Combining ensembles and data augmentation can harm your calibration. In *ICLR*, 2021.

[80] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *NeurIPS*, 2020.

[81] Yoichi Yaguchi, Fumiyuki Shiratani, and Hidekazu Iwaki. Mixfeat: Mix feature in latent space learns discriminative space. *Openreview preprint*, 2019.

[82] Taojiannan Yang, Sijie Zhu, and Chen Chen. Gradaug: A new regularization method for deep neural networks. *NeurIPS*, 2020.

[83] 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 *ICCV*, 2019.

[84] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[85] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017.

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

[87] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? *ICLR*, 2021.

[88] Shuai Zhao, Liguang Zhou, Wenxiao Wang, Deng Cai, Tin Lun Lam, and Yangsheng Xu. Splitnet: Divide and co-training. *ArXiv preprint*, 2020.