Embedding Space Interpolation Beyond Mini-Batch, Beyond Pairs and Beyond Examples

Shashanka Venkataramanan, Ewa Kijak, Laurent Amsaleg, Yannis Avrithis

Embedding Space Interpolation Beyond Mini-Batch, Beyond Pairs and Beyond Examples

Shashanka Venkataramanan[1] Ewa Kijak[1] Laurent Amsaleg[1] Yannis Avrithis[2]

[1]Inria, Univ Rennes, CNRS, IRISA [2]IARAI

Abstract

Mixup refers to interpolation-based data augmentation, originally motivated as a way to go beyond *empirical risk minimization* (ERM). Its extensions mostly focus on the definition of interpolation and the space (input or embedding) where it takes place, while the augmentation process itself is less studied. In most methods, the number of generated examples is limited to the mini-batch size and the number of examples being interpolated is limited to two (pairs), in the input space.

We make progress in this direction by introducing *MultiMix*, which generates an arbitrarily large number of interpolated examples beyond the mini-batch size, and interpolates the entire mini-batch in the embedding space. Effectively, we sample on the entire *convex hull* of the mini-batch rather than along linear segments between pairs of examples.

On sequence data we further extend to *Dense MultiMix*. We densely interpolate features and target labels at each spatial location and also apply the loss densely. To mitigate the lack of dense labels, we inherit labels from examples and weight interpolation factors by attention as a measure of confidence.

Overall, we increase the number of loss terms per mini-batch by orders of magnitude at little additional cost. This is only possible because of interpolating in the *embedding space*. We empirically show that our solutions yield significant improvement over state-of-the-art mixup methods on four different benchmarks, despite interpolation being only linear. By analyzing the embedding space, we show that the classes are more tightly clustered and uniformly spread over the embedding space, thereby explaining the improved behavior.

1 Introduction

Mixup [47] is a data augmentation method that interpolates between pairs of training examples, thus regularizing a neural network to favor linear behavior in-between examples. Besides improving generalization, it has important properties such as reducing overconfident predictions and increasing the robustness to adversarial examples. Several follow-up works have studied interpolation in the *latent* or *embedding* space, which is equivalent to interpolating along a manifold in the input space [36], and a number of nonlinear and attention-based interpolation mechanisms [45, 16, 15, 33, 3]. However, little progress has been made in the augmentation process itself, *i.e.*, the number n of generated examples and the number m of examples being interpolated.

Mixup was originally motivated as a way to go beyond *empirical risk minimization* (ERM) [34] through a vicinal distribution expressed as an expectation over an interpolation factor λ, which is equivalent to the set of linear segments between all pairs of training inputs and targets. In practice however, in every training iteration, a single scalar λ is drawn and the number of interpolated pairs is limited to the size b of the mini-batch ($n = b$), as illustrated in Figure 1(a). This is because if

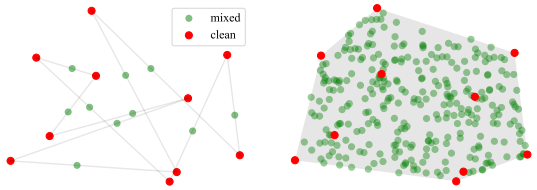
37th Conference on Neural Information Processing Systems (NeurIPS 2023).

METHOD	SPACE	TERMS	MIXED	FACT	DISTR
Mixup [47]	input	b	2	1	Beta
Manifold mixup [36]	embedding	b	2	1	Beta
ζ-Mixup [1]	input	b	25	1	RandPerm
SuperMix [6]	input	b	3	1	Dirichlet
MultiMix (ours)	embedding	n	b	n	Dirichlet
Dense MultiMix (ours)	embedding	nr	b	nr	Dirichlet

Table 1: *Interpolation method properties*. SPACE: Space where interpolation takes place; TERMS: number of loss terms per mini-batch; MIXED: maximum number m of examples being interpolated; FACT: number of interpolation factors λ per mini-batch; DISTR: distribution used to sample interpolation factors; RandPerm: random permutations of a fixed discrete probability distribution. b: mini-batch size; n: number of generated examples per mini-batch; r: spatial resolution.

interpolation takes place in the input space, it would be expensive to increase the number of pairs per iteration. To our knowledge, these limitations exist in all mixup methods.

In this work, we argue that a data augmentation process should increase the data seen by the model, or at least by its last few layers, as much as possible. In this sense, we follow *manifold mixup* [36] and generalize it in a number of ways to introduce *MultiMix*.



(a) mixup (b) MultiMix (ours)

Figure 1: Data augmentation in a mini-batch B of $b = 10$ points in two dimensions. (a) *mixup*: sampling $n = b$ points on linear segments between b pairs of points using the same interpolation factor λ. (b) *MultiMix*: sampling $n = 300$ points in the convex hull of B.

First, we increase the number n of generated examples beyond the mini-batch size b. In practice, n is orders of magnitude greater ($n \gg b$). This is possible by interpolating at the deepest layer, *i.e.*, just before the classifier, which happens to be the most effective choice. To our knowledge, we are the first to investigate $n > b$.

Second, we increase the number m of examples being interpolated from $m = 2$ (pairs) to $m = b$ (a single *tuple* containing the entire mini-batch). Effectively, instead of linear segments between pairs of examples in the mini-batch, we sample on their entire *convex hull* as illustrated in Figure 1(b). This idea has been investigated in the input space: the original mixup method [47] found it non-effective, while [6] found it effective only up to $m = 3$ examples and [1] went up to $m = 25$ but with very sparse interpolation factors. To our knowledge, we are the first to investigate $m > 2$ in the embedding space and to show that it is effective up to $m = b$.

Third, instead of using a single scalar value of λ per mini-batch, we draw a different vector $\lambda \in \mathbb{R}^m$ for each interpolated example. A single λ works for standard mixup because the main source of randomness is the choice of pairs (or small tuples) out of b examples. In our case, because we use a single tuple of size $m = b$, the only source of randomness being λ.

We also argue that, what matters more than the number of (interpolated) examples is the total number of *loss terms* per mini-batch. A common way to increase the number of loss terms per example is by *dense* operations when working on sequence data, *e.g.* patches in images or voxels in video. This is common in dense tasks like segmentation [29] and less common in classification [17]. We are the first to investigate this idea in mixup, introducing *Dense MultiMix*.

In particular, this is an extension of MultiMix where we work with feature tensors of spatial resolution r and densely interpolate features and targets at each spatial location, generating r interpolated features per example and $nr > n$ per mini-batch. We also apply the loss densely. This increases the number of loss terms further by a factor r, typically one or two orders of magnitude, compared with MultiMix. Of course, for this to work, we also need a target label per feature, which we inherit from the corresponding example. This is a *weak* form of supervision [52]. To carefully select the most representative features per object, we use an *attention map* representing our confidence in the target label per spatial location. The interpolation vectors λ are then weighted by attention.

Table 1 summarizes the properties of our solutions against existing interpolation methods. Overall, we make the following contributions:

1. We generate an *arbitrary large number of interpolated examples* beyond the mini-batch size, each by interpolating the entire mini-batch in the embedding space, with one interpolation vector per example. (subsection 3.2).
2. We extend to attention-weighted *dense* interpolation in the embedding space, further increasing the number of loss terms per example (subsection 3.3).
3. We improve over state-of-the-art (SoTA) mixup methods on *image classification*, *robustness to adversarial attacks*, *object detection* and *out-of-distribution detection*. Our solutions have little or no additional cost while interpolation is only linear (section 4).
4. Analysis of the embedding space shows that our solutions yield classes that are *tightly clustered* and *uniformly spread* over the embedding space (section 4).

2 Related Work

Mixup: interpolation methods In general, mixup interpolates between pairs of input examples [47] or embeddings [36] and their corresponding target labels. Several follow-up methods mix input images according to spatial position, either at random rectangles [45] or based on attention [33, 16, 15], in an attempt to focus on a different object in each image. Other follow-up method [24] randomly cuts image at patch level and obtains its corresponding mixed label using content-based attention. We also use attention in our dense MultiMix variant, but in the embedding space. Other definitions of interpolation include the combination of content and style from two images [14], generating out-of-manifold samples using adaptive masks [26], converts images to their spectrum domain to generate binary masks using a threshold [11] and spatial alignment of dense features [35]. Our dense MultiMix also uses dense features but without aligning them and can interpolate a large number of interpolated samples. Our work is orthogonal to these methods as we focus on the sampling process of augmentation rather than on the definition of interpolation. We refer the reader to [19] for a comprehensive study of mixup methods.

Mixup: sampling To the best of our knowledge, the only methods that interpolate more than two examples for image classification are OptTransMix [53], SuperMix [6] and ζ-Mixup [1]. All three methods operate in the *input space* and limit the number of generated examples to the mini-batch size; whereas our MultiMix generates an arbitrary number of interpolated examples (more than 1000 in practice) in the *embedding space*. To determine the interpolation weights, OptTransMix uses a complex optimization process and only applies to images with clean background; ζ-Mixup uses random permutations of a fixed vector; and SuperMix uses a Dirichlet distribution over *not more than* 3 examples in practice. We also use a Dirichlet distribution but over as many examples as the mini-batch size. Beyond classification, m-Mix [48] uses graph neural networks in a self-supervised setting with pair-based loss functions. The interpolation weights are deterministic and based on pairwise similarities. This operation resembles a layer of a graph neural network.

Dense loss functions Although standard in dense tasks like semantic segmentation [29, 12], where dense targets commonly exist, dense loss functions are less common otherwise. Few examples are in *weakly-supervised segmentation* [52, 2], *few-shot learning* [22, 20], where data augmentation is of utter importance, *attribution methods* [17] and *unsupervised representation learning*, *e.g.* dense contrastive learning [30, 38], learning from spatial correspondences [42, 40] and masked language or image modeling [7, 41, 21, 51]. To our knowledge, we are the first to use dense interpolation and a dense loss function for mixup.

3 Method

3.1 Preliminaries and background

Problem formulation Let $x \in \mathcal{X}$ be an input example and $y \in \mathcal{Y}$ its one-hot encoded target, where $\mathcal{X} = \mathbb{R}^D$ is the input space, $\mathcal{Y} = \{0, 1\}^c$ and c is the total number of classes. Let $f_\theta : \mathcal{X} \to \mathbb{R}^d$ be an encoder that maps the input x to an embedding $z = f_\theta(x)$, where d is the dimension of the embedding. A classifier $g_W : \mathbb{R}^d \to \Delta^{c-1}$ maps z to a vector $p = g_W(z)$ of predicted probabilities

over classes, where $\Delta^n \subset \mathbb{R}^{n+1}$ is the unit n-simplex, *i.e.*, $p \geq 0$ and $\mathbf{1}_c^\top p = 1$, and $\mathbf{1}_c \in \mathbb{R}^c$ is an all-ones vector. The overall network mapping is $f := g_W \circ f_\theta$. Parameters (θ, W) are learned by optimizing over mini-batches.

Given a mini-batch of b examples, let $X = (x_1, \ldots, x_b) \in \mathbb{R}^{D \times b}$ be the inputs, $Y = (y_1, \ldots, y_b) \in \mathbb{R}^{c \times b}$ the targets and $P = (p_1, \ldots, p_b) \in \mathbb{R}^{c \times b}$ the predicted probabilities of the mini-batch, where $P = f(X) := (f(x_1), \ldots, f(x_b))$. The objective is to minimize the cross-entropy

$$H(Y, P) := -\mathbf{1}_c^\top (Y \odot \log(P)) \mathbf{1}_b / b \tag{1}$$

of predicted probabilities P relative to targets Y averaged over the mini-batch, where $\odot$ is the Hadamard (element-wise) product. In summary, the mini-batch loss is

$$L(X, Y; \theta, W) := H(Y, g_W(f_\theta(X))). \tag{2}$$

The total number of loss terms per mini-batch is b.

Mixup Mixup methods commonly interpolate pairs of inputs or embeddings and the corresponding targets at the mini-batch level while training. Given a mini-batch of b examples with inputs X and targets Y, let $Z = (z_1, \ldots, z_b) \in \mathbb{R}^{d \times b}$ be the embeddings of the mini-batch, where $Z = f_\theta(X)$. *Manifold mixup* [36] interpolates the embeddings and targets by forming a convex combination of the pairs with interpolation factor $\lambda \in [0, 1]$:

$$\widetilde{Z} = Z(\lambda I + (1 - \lambda)\Pi) \tag{3}$$

$$\widetilde{Y} = Y(\lambda I + (1 - \lambda)\Pi), \tag{4}$$

where $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$, I is the identity matrix and $\Pi \in \mathbb{R}^{b \times b}$ is a permutation matrix. *Input mixup* [47] interpolates inputs rather than embeddings:

$$\widetilde{X} = X(\lambda I + (1 - \lambda)\Pi). \tag{5}$$

Whatever the interpolation method and the space where it is performed, the interpolated data, *e.g.* $\widetilde{X}$ [47] or $\widetilde{Z}$ [36], replaces the original mini-batch data and gives rise to predicted probabilities $\widetilde{P} = (p_1, \ldots, p_b) \in \mathbb{R}^{c \times b}$ over classes, *e.g.* $\widetilde{P} = f(\widetilde{X})$ [47] or $\widetilde{P} = g_W(\widetilde{Z})$ [36]. Then, the average cross-entropy $H(\widetilde{Y}, \widetilde{P})$ (1) between the predicted probabilities $\widetilde{P}$ and interpolated targets $\widetilde{Y}$ is minimized. The number of generated examples per mini-batch is $n = b$, same as the original mini-batch size, and each is obtained by interpolating $m = 2$ examples. The total number of loss terms per mini-batch is again b.

3.2 MultiMix

Interpolation The number of generated examples per mini-batch is now $n \gg b$ and the number of examples being interpolated is $m = b$. Given a mini-batch of b examples with embeddings Z and targets Y, we draw interpolation vectors $\lambda_k \sim \mathrm{Dir}(\alpha)$ for $k = 1, \ldots, n$, where $\mathrm{Dir}(\alpha)$ is the symmetric Dirichlet distribution and $\lambda_k \in \Delta^{m-1}$, that is, $\lambda_k \geq 0$ and $\mathbf{1}_m^\top \lambda_k = 1$. We then interpolate embeddings and targets by taking n convex combinations over all m examples:

$$\widetilde{Z} = Z\Lambda \tag{6}$$

$$\widetilde{Y} = Y\Lambda, \tag{7}$$

where $\Lambda = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{b \times n}$. We thus generalize manifold mixup [36]:

1. from b to an arbitrary number $n \gg b$ of generated examples: interpolated embeddings $\widetilde{Z} \in \mathbb{R}^{d \times n}$ (6) *vs.* $\mathbb{R}^{d \times b}$ in (3), targets $\widetilde{Y} \in \mathbb{R}^{c \times n}$ (7) *vs.* $\mathbb{R}^{c \times b}$ in (4);
2. from pairs ($m = 2$) to a tuple of length $m = b$, containing the entire mini-batch: m-term convex combination (6),(7) *vs.* 2-term in (3),(4), Dirichlet *vs.* Beta distribution;
3. from fixed λ across the mini-batch to a different λ_k for each generated example.

Loss Again, we replace the original mini-batch embeddings Z by the interpolated embeddings $\widetilde{Z}$ and minimize the average cross-entropy $H(\widetilde{Y}, \widetilde{P})$ (1) between the predicted probabilities $\widetilde{P} = g_W(\widetilde{Z})$ and the interpolated targets $\widetilde{Y}$ (7). Compared with (2), the mini-batch loss becomes

$$L_M(X, Y; \theta, W) := H(Y\Lambda, g_W(f_\theta(X)\Lambda)). \tag{8}$$

The total number of loss terms per mini-batch is now $n \gg b$.

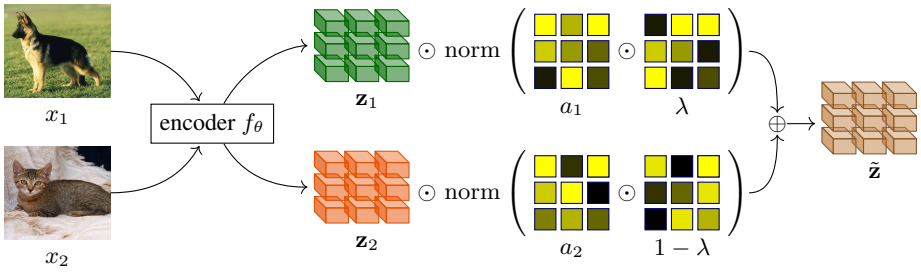


Figure 2: *Dense MultiMix* (subsection 3.3) for the special case $m = 2$ (two examples), $n = 1$ (one interpolated embedding), $r = 9$ (spatial resolution 3×3). The embeddings $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^{d \times 9}$ of input images x_1, x_2 are extracted by encoder f_θ. Attention maps $a_1, a_2 \in \mathbb{R}^9$ are extracted (9), multiplied element-wise with interpolation vectors $\lambda, (1-\lambda) \in \mathbb{R}^9$ (10) and ℓ_1-normalized per spatial position (11). The resulting weights are used to form the interpolated embedding $\tilde{\mathbf{z}} \in \mathbb{R}^{d \times 9}$ as a convex combination of $\mathbf{z}_1, \mathbf{z}_2$ per spatial position (12). Targets are interpolated similarly (13).

3.3 Dense MultiMix

We now extend to the case where the embeddings are structured, *e.g.* in tensors. This happens *e.g.* with token *vs.* sentence embeddings in NLP and patch *vs.* image embeddings in vision. It works by removing spatial pooling and applying the loss function densely over all tokens/patches. The idea is illustrated in Figure 2. For the sake of exposition, our formulation uses sets of matrices grouped either by example or by spatial position. In practice, all operations are on tensors.

Preliminaries The encoder is now $f_\theta : \mathcal{X} \to \mathbb{R}^{d \times r}$, mapping the input x to an embedding $\mathbf{z} = f_\theta(x) \in \mathbb{R}^{d \times r}$, where d is the number of channels and r is its spatial resolution—if there are more than one spatial dimensions, these are flattened.

Given a mini-batch of b examples, we have again inputs $X = (x_1, \dots, x_b) \in \mathbb{R}^{D \times b}$ and targets $Y = (y_1, \dots, y_b) \in \mathbb{R}^{c \times b}$. Each embedding $\mathbf{z}_i = f_\theta(x_i) = (z_i^1, \dots, z_i^r) \in \mathbb{R}^{d \times r}$ for $i = 1, \dots, b$ consists of features $z_i^j \in \mathbb{R}^d$ for spatial position $j = 1, \dots, r$. We group features by position in matrices $Z^1, \dots, Z^r$, where $Z^j = (z_1^j, \dots, z_b^j) \in \mathbb{R}^{d \times b}$ for $j = 1, \dots, r$.

Attention In the absence of dense targets, each spatial location inherits the target of the corresponding input example. This is weak supervision, because the target object is not visible everywhere. To select the most reliable locations, we define a level of confidence according to an attention map. Given an embedding $\mathbf{z} \in \mathbb{R}^{d \times r}$ with target $y \in \mathcal{Y}$ and a vector $u \in \mathbb{R}^d$, the *attention map*

$$a = h(\mathbf{z}^\top u) \in \mathbb{R}^r \tag{9}$$

measures the similarity of features of $\mathbf{z}$ to u, where h is a non-linearity, *e.g.* softmax or ReLU followed by ℓ_1 normalization. There are different ways to define vector u. For example, $u = \mathbf{z}\mathbf{1}_r/r$ by global average pooling (GAP) of $\mathbf{z}$, or $u = Wy$ assuming a linear classifier with $W \in \mathbb{R}^{d \times c}$, similar to class activation mapping (CAM) [50]. In case of no attention, $a = \mathbf{1}_r/r$ is uniform.

Given a mini-batch, let $a_i = (a_i^1, \dots, a_i^r) \in \mathbb{R}^r$ be the attention map of embedding $\mathbf{z}_i$ (9) for $i = 1, \dots, b$. We group attention by position in vectors $a^1, \dots, a^r$, where $a^j = (a_1^j, \dots, a_b^j) \in \mathbb{R}^b$ for $j = 1, \dots, r$. Figure 6 in the supplementary shows the attention obtained by (9). We observe high confidence on the entire or part of the object. Where confidence is low, we assume the object is not visible and thus the corresponding interpolation factor should be low.

Interpolation There are again $n \gg b$ generated examples per mini-batch, with $m = b$ examples being densely interpolated. For each spatial position $j = 1, \dots, r$, we draw interpolation vectors $\lambda_k^j \sim \mathrm{Dir}(\alpha)$ for $k = 1, \dots, n$ and define $\Lambda^j = (\lambda_1^j, \dots, \lambda_n^j) \in \mathbb{R}^{m \times n}$. Since input examples are assumed to contribute according to the attention vector $a^j \in \mathbb{R}^m$, we scale the rows of Λ^j accordingly and normalize its columns back to Δ^{m-1} to define convex combinations:

$$M^j = \mathrm{diag}(a^j)\Lambda^j \tag{10}$$

$$\hat{M}^j = M^j \,\mathrm{diag}(\mathbf{1}_m^\top M^j)^{-1} \tag{11}$$

We then interpolate embeddings and targets by taking n convex combinations over m examples:

$$\widetilde{Z}^j = Z^j \hat{M}^j \tag{12}$$

$$\widetilde{Y}^j = Y \hat{M}^j. \tag{13}$$

This is similar to (6),(7), but there is a different interpolated embedding matrix $\widetilde{Z}^j \in \mathbb{R}^{d\times n}$ as well as target matrix $\widetilde{Y}^j \in \mathbb{R}^{c\times n}$ per position, even though the original target matrix Y is one. The total number of interpolated features and targets per mini-batch is now nr.

Classifier The classifier is now $g_W : \mathbb{R}^{d\times r} \to \mathbb{R}^{c\times r}$, maintaining the same spatial resolution as the embedding and generating one vector of predicted probabilities per spatial position. This is done by removing average pooling or any down-sampling operation. The interpolated embeddings $\widetilde{Z}^1, \ldots, \widetilde{Z}^r$ (12) are grouped by example into $\widetilde{\mathbf{z}}_1, \ldots, \widetilde{\mathbf{z}}_n \in \mathbb{R}^{d\times r}$, mapped by g_W to predicted probabilities $\widetilde{\mathbf{p}}_1, \ldots, \widetilde{\mathbf{p}}_n \in \mathbb{R}^{c\times r}$ and grouped again by position into $\widetilde{P}^1, \ldots, \widetilde{P}^r \in \mathbb{R}^{c\times n}$.

In the simple case where the original classifier is linear, *i.e.* $W \in \mathbb{R}^{d\times c}$, it is seen as 1×1 convolution and applied densely to each column (feature) of $\widetilde{Z}^j$ for $j = 1, \ldots, r$.

Loss Finally, we learn parameters θ, W by minimizing the *weighted cross-entropy* $H(\widetilde{Y}^j, \widetilde{P}^j; s)$ of $\widetilde{P}^j$ relative to the interpolated targets $\widetilde{Y}^j$ again densely at each position j, where

$$H(Y, P; s) := -\mathbf{1}_c^\top (Y \odot \log(P)) s / (\mathbf{1}_n^\top s) \tag{14}$$

generalizes (1) and the weight vector is defined as $s = \mathbf{1}_m^\top M^j \in \mathbb{R}^n$. This is exactly the vector used to normalize the columns of M^j in (11). The motivation is that the columns of M^j are the original interpolation vectors weighted by attention: A small ℓ_1 norm indicates that for the given position j, we are sampling from examples of low attention, hence the loss is to be discounted. The total number of loss terms per mini-batch is now nr.

4 Experiments

4.1 Setup

We use a mini-batch of size $b = 128$ examples in all experiments. Following manifold mixup [36], for every mini-batch, we apply MultiMix with probability 0.5 or input mixup otherwise. For MultiMix, the default settings are given in subsection 4.5. We use PreActResnet-18 (R-18) [13] and WRN16-8 [46] as encoder on CIFAR-10 and CIFAR-100 datasets [18]; R-18 on TinyImagenet [43] (TI); and Resnet-50 (R-50) and ViT-S/16 [8] on ImageNet [32]. To better understand the effect of mixup in ViT, we evaluate MultiMix and Dense MultiMix on ImageNet without using strong augmentations like Auto-Augment [4], Rand-Augment [5], random erasing [49] and CutMix [45]. We reproduce TransMix [3] and TokenMix [24] using these settings.

We report the mean and standard deviation of the top-1 accuracy (%) for five runs on image classification. On robustness to adversarial attacks (subsection 4.2), we report the top-1 error. We also experiment on object detection (subsection 4.3) and out-of-distribution detection (subsection A.3).

4.2 Results: Image classification and robustness

Image classification In Table 2 we observe that MultiMix and Dense MultiMix already outperform SoTA on all datasets except CIFAR-10 with R-18, where they are on par with Co-Mixup. Dense MultiMix improves over vanilla MultiMix and its effect is complementary on all datasets. On TI for example, Dense MultiMix improves over MultiMix by 1.33% and SoTA by 1.59%. We provide additional analysis of the embedding space on 10 classes of CIFAR-100 in subsection 4.4.

In Table 3 we observe that on ImageNet with R-50, vanilla MultiMix outperforms all methods except AlignMixup. Dense MultiMix outperforms all SoTA with both R-50 and ViT-S/16, bringing an overall gain of 3% over the baseline with R-50 and 2.2% with ViT-S/16. The gain over AlignMixup with R-50 is small, but it is impressive that it comes with only linear interpolation. To better isolate the effect of each method, we reproduce TransMix [3] and TokenMix [24] with ViT-S/16 using their official code with our settings, *i.e.*, without strong regularizers like CutMix, Auto-Augment, Random-Augment *etc*. MultiMix is on par, while Dense MultiMix outperforms them by 1%.

Dataset	CIFAR-10		CIFAR-100		TI
Network	R-18	W16-8	R-18	W16-8	R-18
Baseline†	95.41±0.02	94.93±0.06	76.69±0.26	78.80±0.55	56.49±0.21
Manifold mixup [36]†	97.00±0.05	96.44±0.02	80.00±0.34	80.77±0.26	59.31±0.49
PuzzleMix [16]†	97.04±0.04	97.00±0.03	79.98±0.05	80.78±0.23	63.52±0.42
Co-Mixup [15]†	**97.10±0.03**	96.44±0.08	80.28±0.13	80.39±0.34	64.12±0.43
AlignMixup [35]†	97.06±0.04	96.91±0.01	81.71±0.07	81.24±0.02	66.85±0.07
ζ-Mixup [1]⋆	96.26±0.04	96.35±0.04	80.46±0.26	79.73±0.15	63.18±0.14
MultiMix (ours)	97.07±0.03	97.06±0.02	81.82±0.04	81.44±0.03	67.11±0.04
Dense MultiMix (ours)	97.09±0.02	**97.09±0.02**	**81.93±0.04**	**81.77±0.03**	**68.44±0.05**
Gain	**-0.01**	**+0.09**	**+0.22**	**+0.53**	**+1.59**

Table 2: *Image classification* on CIFAR-10/100 and TI (TinyImagenet). Mean and standard deviation of Top-1 accuracy (%) for 5 runs. R: PreActResnet, W: WRN. ⋆: reproduced, †: reported by AlignMixup, **Bold black**: best; Blue: second best; underline: best baseline. Gain: improvement over best baseline. Additional comparisons are in subsection A.2.

Attack	FGSM					PGD			
Dataset	CIFAR-10		CIFAR-100		TI	CIFAR-10		CIFAR-100	
Network	R-18	W16-8	R-18	W16-8	R-18	R-18	W16-8	R-18	W16-8
Baseline†	88.8±0.11	88.3±0.33	87.2±0.10	72.6±0.22	91.9±0.06	99.9±0.0	99.9±0.01	99.9±0.01	99.9±0.01
Manifold mixup [36]†	76.9±0.14	76.0±0.04	80.2±0.06	56.3±0.10	89.3±0.06	97.2±0.01	98.4±0.03	99.6±0.01	98.4±0.03
PuzzleMix [16]†	57.4±0.22	60.7±0.02	78.8±0.09	57.8±0.03	83.8±0.05	97.7±0.01	97.0±0.01	96.4±0.02	95.2±0.03
Co-Mixup [15]†	60.1±0.05	58.8±0.10	77.5±0.02	56.5±0.04	–	97.5±0.02	96.1±0.03	95.3±0.03	94.2±0.01
AlignMixup [35]†	54.8±0.03	56.0±0.05	74.1±0.04	55.0±0.03	78.8±0.03	95.3±0.04	96.7±0.03	90.4±0.01	92.1±0.03
ζ-Mixup [1]⋆	72.8±0.23	67.3±0.24	75.3±0.21	68.0±0.21	84.7±0.18	98.0±0.06	98.6±0.03	97.4±0.10	96.1±0.10
MultiMix (ours)	**54.1±0.09**	55.3±0.04	73.8±0.04	54.5±0.01	77.5±0.01	94.2±0.04	94.8±0.01	90.0±0.01	91.6±0.01
Dense MultiMix (ours)	**54.1±0.01**	**53.3±0.03**	**73.5±0.03**	**52.9±0.04**	**75.5±0.04**	**92.9±0.04**	**92.6±0.01**	**88.6±0.03**	**90.8±0.01**
Gain	**+0.7**	**+2.7**	**+0.6**	**+2.1**	**+3.3**	**+2.4**	**+3.5**	**+1.4**	**+1.3**

Table 4: *Robustness to FGSM & PGD attacks*. Mean and standard deviation of Top-1 error (%) for 5 runs: lower is better. ⋆: reproduced, †: reported by AlignMixup. **Bold black**: best; Blue: second best; underline: best baseline. Gain: reduction of error over best baseline. TI: TinyImagenet. R: PreActResnet, W: WRN. Comparison with additional baselines is given in subsection A.2.

Training speed Table 3 also shows the training speed as measured on NVIDIA V-100 GPU including forward and backward pass. The vanilla MultiMix has nearly the same speed with the baseline, bringing an accuracy gain of 2.49% with R-50. Dense MultiMix is slightly slower, increasing the gain to 3.10%. The inference speed is the same for all methods.

Robustness to adversarial attacks We follow the experimental settings of AlignMixup [35] and use $8/255$ l_∞ ϵ-ball for FGSM [10] and $4/255$ l_∞ ϵ-ball with step size 2/255 for PGD [27] attack. In Table 4 we observe that MultiMix is already more robust than SoTA on all datasets and settings. Dense MultiMix also increases the robustness and is complementary.

The overall gain is more impressive than in classification according to Table 2. For example, against the strong PGD attack on CIFAR-10 with W16-8, the SoTA Co-Mixup improves the baseline by 3.8% while Dense MultiMix improves it by 7.3%, which is double. MultiMix and Dense MultiMix outperform Co-Mixup and PuzzleMix by 3-6% in robustness on CIFAR-10, even though they are on-par on classification. There is also a significant gain over SoTA AlignMixup by 1-3% in robustness to FGSM on TinyImageNet and to the stronger PGD.

Network	ResNet-50		ViT-S/16	
Method	Speed	Acc	Speed	Acc
Baseline†	1.17	76.32	1.01	73.9
Manifold mixup [36]†	1.15	77.50	0.97	74.2
PuzzleMix [16]†	0.84	78.76	0.73	74.7
Co-Mixup [15]†	0.62	–	0.57	74.9
TransMix [3]⋆	–	–	1.01	75.1
TokenMix [24]⋆	–	–	0.87	75.3
AlignMixup [35]†	1.03	79.32	–	–
MultiMix (ours)	1.16	78.81	0.98	75.2
Dense MultiMix (ours)	0.95	**79.42**	0.88	**76.1**
Gain		**+0.1**		**+1.2**

Table 3: *Image classification and training speed* on ImageNet. Top-1 accuracy (%): higher is better. Speed: images/sec ($\times 10^3$): higher is better. †: reported by AlignMixup; ⋆: reproduced. **Bold black**: best; Blue: second best; underline: best baseline. Gain: improvement over best baseline. Comparison with additional baselines is given in subsection A.2.

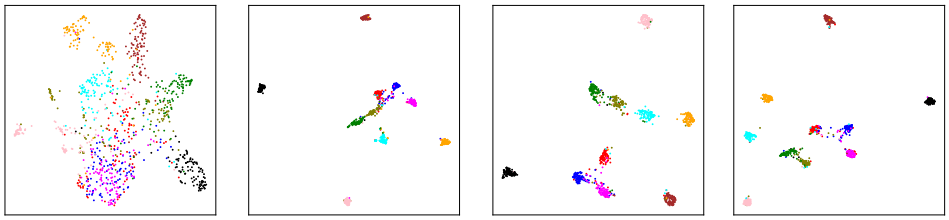


Figure 3: *Embedding space visualization* for 100 test examples per class of 10 randomly chosen classes of CIFAR-100 with PreActResnet-18, using UMAP [28].

4.3 Results: Transfer learning to object detection

We evaluate the effect of mixup on the generalization ability of a pre-trained network to object detection as a downstream task. Following the settings of CutMix [45], we pre-train R-50 on ImageNet with mixup methods and use it as the backbone for SSD [25] with fine-tuning on Pascal VOC07+12 [9] and Faster-RCNN [31] with fine-tuning on MS-COCO [23].

In Table 5, we observe that, while MultiMix is slightly worse than AlignMixup on Pascal VOC07+12, Dense MultiMix brings improvements over the SoTA on both datasets and is still complementary. This is consistent with classification results. Compared with the baseline, our best setting brings a gain of 0.8 mAP on Pascal VOC07+12 and 0.35 mAP on MS-COCO.

Dataset	VOC07+12		MS-COCO	
Detector	SSD	Speed	FR-CNN	Speed
Baseline†	76.7	9.7	33.27	23.6
Input mixup†	76.6	9.5	34.18	22.9
CutMix†	77.6	9.4	35.16	23.2
AlignMixup†	78.4	8.9	35.84	20.4
MultiMix (ours)	77.9	9.6	35.93	23.2
Dense MultiMix (ours)	**79.2**	8.8	**36.19**	19.8
Gain	**+0.8**		**+0.35**	

Table 5: *Transfer learning* to object detection. Mean average precision (mAP, %): higher is better. †: reported by AlignMixup. **Bold black**: best; Blue: second best; underline: best baseline. Gain: increase in mAP. Speed: images/sec: higher is better.

4.4 Analysis of the embedding space

Qualitative analysis We qualitatively analyze the embedding space on 10 CIFAR-100 classes in Figure 3. We observe that the quality of embeddings of the baseline is extremely poor with severely overlapping classes, which explains its poor performance on image classification. All mixup methods result in clearly better clustered and more uniformly spread classes. AlignMixup [35] yields five somewhat clustered classes and five moderately overlapping ones. Our best setting, *i.e.*, Dense MultiMix, results in five tightly clustered classes and another five somewhat overlapping but less than all competitors.

Quantitative analysis We also quantitatively assess the embedding space on the CIFAR-100 test set using alignment and uniformity [37]. *Alignment* measures the expected pairwise distance of examples in the same class. Lower alignment indicates that the classes are more tightly clustered. *Uniformity* measures the (log of the) expected pairwise similarity of all examples using a Gaussian kernel as a similarity function. Lower uniformity indicates that classes are more uniformly spread in the embedding space. On CIFAR-100, we obtain alignment 3.02 for baseline, 2.04 for AlignMixup, 1.27 for MultiMix and 0.92 for Dense MultiMix. We also obtain uniformity -1.94 for the baseline, -2.38 for AlignMixup [35], -4.77 for MultiMix and -5.68 for Dense MultiMix. These results validate the qualitative analysis of Figure 3.

4.5 Ablations

All ablations are performed using R-18 on CIFAR-100. We study the effect of the layer where we interpolate, the number n of generated examples per mini-batch, the number m of examples being interpolated and the Dirichlet parameter α. More ablations are given in the supplementary.

Interpolation layer For MultiMix, we use the entire network as the encoder f_θ by default, except for the last fully-connected layer, which we use as classifier g_W. Thus, we *interpolate embeddings* in the deepest layer by default. Here, we study the effect of different decompositions of the network

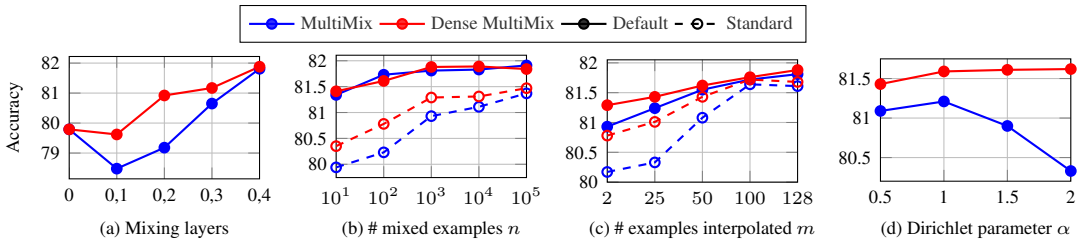


Figure 4: *Ablation study* on CIFAR-100 using R-18. (a) Interpolation layers (R-18 block; 0: input mixup). (b) Number n of interpolated examples per mini-batch with $m = b$ (Default) and $m = 2$ (Standard). (c) Number m of examples being interpolated, with $n = 1000$ (Default) and $n = 100$ (Standard). (d) Fixed value of Dirichlet parameter α.

$f = g_W \circ f_\theta$, such that interpolation takes place at a different layer. In Figure 4(a), we observe that mixing at the deeper layers of the network significantly improves performance. The same behavior is observed with Dense MultiMix, which validates our default choice.

It is interesting that the authors of input mixup [47] found that convex combinations of three or more examples in the input space with weights from the Dirichlet distribution do not bring further gain. This agrees with the finding of SuperMix [6] for four or more examples. Figure 4(a) suggests that further gain emerges when mixing in deeper layers.

Number n of generated examples per mini-batch This is important since our aim is to increase the amount of data seen by the model, or at least part of the model. We observe from Figure 4(b) that accuracy increases overall with n and saturates for $n \geq 1000$ for both variants of MultiMix. The improvement is more pronnounced when $m = 2$, which is standard for most mixup methods. Our best solution, Dense MultiMix, works best at $n = 1000$ and $n = 10,000$. We choose $n = 1000$ as default, given also that the training cost increases with n. The training speed as a function of n is given in the supplementary and is nearly constant for $n \leq 1000$.

Number m of examples being interpolated We vary m between 2 (pairs) and $b = 128$ (entire mini-batch) by using $\Lambda' \in \mathbb{R}^{m \times n}$ drawn from Dirichlet along with combinations (subsets) over the mini-batch to obtain $\Lambda \in \mathbb{R}^{b \times n}$ with m nonzero elements per column in (6),(7). We observe in Figure 4(c) that for both MultiMix and Dense MultiMix the performance increases with m. The improvement is more pronnounced when $n = 100$, which is similar to the standard setting ($n = b = 128$) of most mixup methods. Our choice of $m = b = 128$ brings an improvement of 1-1.8% over $m = 2$. We use this as our default setting.

Dirichlet parameter α Our default setting is to draw α uniformly at random from $[0.5, 2]$ for every interpolation vector (column of Λ). Here we study the effect of a fixed value of α. In Figure 4(d), we observe that the best accuracy comes with $\alpha = 1$ for most MultiMix variants, corresponding to the uniform distribution over the convex hull of the mini-batch embeddings. However, all measurements are lower than the default $\alpha \sim U[0.5, 2]$. For example, from Table 2(a) (CIFAR-100, R-18), Dense MultiMix has accuracy 81.93, compared with 81.59 in Figure 4(d) for $\alpha = 1$.

5 Conclusion

In terms of input interpolation, the take-home message of this work is that, instead of devising smarter and more complex interpolation functions in the input space or the first layers of the representation, it is more beneficial to just perform linear interpolation in the very last layer where the cost is minimal, and then increase as much as possible the number of interpolated embeddings beyond the mini-batch size as well as the number of examples being interpolated up to the mini-batch size. This is more in line with the original motivation of mixup as a way to go beyond ERM. It is also beneficial to increase the number of loss terms per generated example by dense interpolating elements (patches, tokens) when working with sequence data.

A natural extension of this work is to settings other than supervised classification. A limitation is that it is not straightforward to combine the sampling scheme of MultiMix with complex interpolation methods, unless they are fast to compute in the embedding space.

References

[1] Kumar Abhishek, Colin J Brown, and Ghassan Hamarneh. Multi-sample ζ-mixup: Richer, more realistic synthetic samples from a p-series interpolant. *arXiv preprint arXiv:2204.03323*, 2022. 2, 3, 7, 1

[2] Jiwoon Ahn, Sunghyun Cho, and Suha Kwak. Weakly supervised learning of instance segmentation with inter-pixel relations. In *CVPR*, 2019. 3

[3] Jie-Neng Chen, Shuyang Sun, Ju He, Philip HS Torr, Alan Yuille, and Song Bai. Transmix: Attend to mix for vision transformers. In *CVPR*, 2022. 1, 6, 7, 2

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 6

[5] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, 2020. 6

[6] Ali Dabouei, Sobhan Soleymani, Fariborz Taherkhani, and Nasser M. Nasrabadi. Supermix: Supervising the mixing data augmentation. In *CVPR*, 2021. 2, 3, 9, 1, 4

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 3

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 6

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010. 8

[10] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015. 7

[11] Ethan Harris, Antonia Marcu, Matthew Painter, Mahesan Niranjan, Adam Prügel-Bennett, and Jonathon Hare. Fmix: Enhancing mixed sample data augmentation. *arXiv preprint arXiv:2002.12047*, 2020. 3

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017. 3

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

[14] Minui Hong, Jinwoo Choi, and Gunhee Kim. Stylemix: Separating content and style for enhanced data augmentation. In *CVPR*, 2021. 3, 1, 2, 4

[15] Jang-Hyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. Co-mixup: Saliency guided joint mixup with supermodular diversity. In *ICLR*, 2021. 1, 3, 7, 2, 4

[16] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *ICML*, 2020. 1, 3, 7, 2, 4

[17] Jae Myung Kim, Junsuk Choe, Zeynep Akata, and Seong Joon Oh. Keep calm and improve visual feature attribution. In *ICCV*, 2021. 2, 3

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 6

[19] Siyuan Li, Zedong Wang, Zicheng Liu, Di Wu, and Stan Z. Li. Openmixup: Open mixup toolbox and benchmark for visual representation learning. *arXiv preprint arXiv:2209.04851*, 2022. 3

[20] Wenbin Li, Lei Wang, Jinglin Xu, Jing Huo, Yang Gao, and Jiebo Luo. Revisiting local descriptor based image-to-class measure for few-shot learning. In *CVPR*, 2019. 3

[21] Zhaowen Li, Zhiyang Chen, Fan Yang, Wei Li, Yousong Zhu, Chaoyang Zhao, Rui Deng, Liwei Wu, Rui Zhao, Ming Tang, et al. MST: Masked self-supervised transformer for visual representation. In *NeurIPS*, 2021. 3

[22] Yann Lifchitz, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. Dense classification and implanting for few-shot learning. In *CVPR*, 2019. 3

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 8

[24] Jihao Liu, Boxiao Liu, Hang Zhou, Hongsheng Li, and Yu Liu. Tokenmix: Rethinking image mixing for data augmentation in vision transformers. In *ECCV*, 2022. 3, 6, 7, 2

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 8

[26] Zicheng Liu, Siyuan Li, Di Wu, Zihan Liu, Zhiyuan Chen, Lirong Wu, and Stan Z Li. Automix: Unveiling the power of mixup for stronger classifiers. In *ECCV*, 2022. 3

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018. 7

[28] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 2018. 8

[29] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 2, 3

[30] Pedro O Pinheiro, Amjad Almahairi, Ryan Benmalek, Florian Golemo, and Aaron Courville. Unsupervised learning of dense visual representations. In *NeurIPS*, 2020. 3

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 8

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 6

[33] A F M Uddin, Mst. Monira, Wheemyung Shin, TaeChoong Chung, and Sung-Ho Bae. SaliencyMix: A saliency guided data augmentation strategy for better regularization. In *ICML*, 2021. 1, 3, 2, 4

[34] VN Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 1999. 1

[35] Shashanka Venkataramanan, Ewa Kijak, Laurent Amsaleg, and Yannis Avrithis. Alignmixup: Improving representation by interpolating aligned features. In *CVPR*, 2022. 3, 7, 8, 1, 2, 4

[36] 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. 1, 2, 3, 4, 6, 7

[37] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*, 2020. 8

[38] Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. Dense contrastive learning for self-supervised visual pre-training. In *CVPR*, 2021. 3

[39] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 4

[40] Zhenda Xie, Yutong Lin, Zheng Zhang, Yue Cao, Stephen Lin, and Han Hu. Propagate yourself: Exploring pixel-level consistency for unsupervised visual representation learning. In *CVPR*, 2021. 3

[41] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. *arXiv preprint arXiv:2111.09886*, 2021. 3

[42] Yuwen Xiong, Mengye Ren, Wenyuan Zeng, and Raquel Urtasun. Self-supervised representation learning from flow equivariance. In *ICCV*, 2021. 3

[43] Leon Yao and John Miller. Tiny imagenet classification with convolutional neural networks. Technical report, Standford University, 2015. 6

[44] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 4

[45] 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. 1, 3, 6, 8, 2, 4

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

[47] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 1, 2, 3, 4, 9

[48] Shaofeng Zhang, Meng Liu, Junchi Yan, Hengrui Zhang, Lingxiao Huang, Xiaokang Yang, and Pinyan Lu. M-mix: Generating hard negatives via multi-sample mixing for contrastive learning. In *SIGKDD*, 2022. 3

[49] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020. 6

[50] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016. 5

[51] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. iBOT: Image bert pre-training with online tokenizer. In *ICLR*, 2022. 3

[52] Yanzhao Zhou, Yi Zhu, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Weakly supervised instance segmentation using class peak response. In *CVPR*, 2018. 2, 3

[53] Jianchao Zhu, Liangliang Shi, Junchi Yan, and Hongyuan Zha. Automix: Mixup networks for sample interpolation via cooperative barycenter learning. In *ECCV*, 2020. 3

A More experiments

A.1 More on setup

Settings and hyperparameters We train MultiMix and Dense MultiMix with mixed examples only. We use a mini-batch of size $b = 128$ examples in all experiments. Following Manifold Mixup [36], for every mini-batch, we apply MultiMix with probability 0.5 or input mixup otherwise. For input mixup, we interpolate the standard $m = b$ pairs (5). For MultiMix, we use the entire network as the encoder f_θ by default, except for the last fully-connected layer, which we use as classifier g_W. We use $n = 1000$ tuples and draw a different $\alpha \sim U[0.5, 2.0]$ for each example from the Dirichlet distribution by default. For multi-GPU experiments, all training hyperparameters including m and n are per GPU.

For Dense MultiMix, the spatial resolution is $r = 4 \times 4 = 16$ on CIFAR-10/100 and $r = 7 \times 7 = 49$ on Imagenet by default. We obtain the attention map by (9) using GAP for vector u and ReLU followed by ℓ_1 normalization as non-linearity h by default. To predict class probabilities and compute the loss densely, we use the classifier g_W as 1×1 convolution by default; when interpolating at earlier layers, we follow the process described in subsection 3.3.

CIFAR-10/100 training Following the experimental settings of AlignMixup [35], we train MultiMix and its variants using SGD for 2000 epochs using the same random seed as AlignMixup. We set the initial learning rate to 0.1 and decay it by a factor of 0.1 every 500 epochs. The momentum is set to 0.9 and the weight decay to 0.0001. We use a batch size $b = 128$ and train on a single NVIDIA RTX 2080 TI GPU for 10 hours.

TinyImageNet training Following the experimental settings of PuzzleMix [16], we train MultiMix and its variants using SGD for 1200 epochs, using the same random seed as AlignMixup. We set the initial learning rate to 0.1 and decay it by a factor of 0.1 after 600 and 900 epochs. The momentum is set to 0.9 and the weight decay to 0.0001. We train on two NVIDIA RTX 2080 TI GPUs for 18 hours.

ImageNet training Following the experimental settings of PuzzleMix [16], we train MultiMix and its variants using the same random seed as AlignMixup. We train R-50 using SGD with momentum 0.9 and weight decay 0.0001 and ViT-S/16 using AdamW with default parameters. The initial learning rate is set to 0.1 and 0.01, respectively. We decay the learning rate by 0.1 at 100 and 200 epochs. We train on 32 NVIDIA V100 GPUs for 20 hours.

Tasks and metrics We use top-1 accuracy (%, higher is better) and top-1 error (%, lower is better) as evaluation metrics on *image classification* and *robustness to adversarial attacks* (subsection 4.2 and subsection A.2). Additional datasets and metrics are reported separately for *transfer learning to object detection* (subsection 4.3) and *out-of-distribution detection* (subsection A.3).

A.2 More results: Classification and robustness

Using the experimental settings of subsection A.1, we extend Table 2, Table 3 and Table 4 of subsection 4.2 in Table 6, Table 7 and Table 8 respectively by comparing MultiMix and its variants with additional mixup methods. The additional methods are Input mixup [47], Cutmix [45], SaliencyMix [33], StyleMix [14], StyleCutMix [14], SuperMix [6] and ζ-Mixup [1]. We reproduce ζ-Mixup and SuperMix using the same settings. For SuperMix, we use the official code[1], which first trains the teacher network using clean examples and then the student using mixed. For fair comparison, we use the same network as the teacher and student models.

In Table 6, Table 7 and Table 8, we observe that MultiMix and its variants outperform all the additional mixup methods on image classification. Furthermore, they are more robust to FGSM and PGD attacks as compared to these additional methods. The remaining observations in subsection 4.2 are still valid.

Dataset	CIFAR-10		CIFAR-100		TI
Network	R-18	W16-8	R-18	W16-8	R-18
Baseline†	95.41±0.02	94.93±0.06	76.69±0.26	78.80±0.55	56.49±0.21
Input mixup [47]†	95.98±0.10	96.18±0.06	79.39±0.40	80.16±0.1	56.60±0.16
CutMix [45]†	96.79±0.04	96.48±0.04	80.56±0.09	80.25±0.41	56.87±0.39
Manifold mixup [36]†	97.00±0.05	96.44±0.02	80.00±0.34	80.77±0.26	59.31±0.49
PuzzleMix [16]†	97.04±0.04	97.00±0.03	79.98±0.05	80.78±0.23	63.52±0.42
Co-Mixup [15]†	**97.10±0.03**	96.44±0.08	80.28±0.13	80.39±0.34	64.12±0.43
SaliencyMix [33]†	96.94±0.05	96.27±0.05	80.36±0.56	80.29±0.05	66.14±0.51
StyleMix [14]†	96.25±0.04	96.27±0.04	80.01±0.79	79.77±0.17	63.88±0.27
StyleCutMix [14]†	96.94±0.05	96.95±0.04	80.67±0.07	80.79±0.04	66.55±0.13
SuperMix [6]‡	96.03±0.05	96.13±0.05	79.07±0.26	79.42±0.05	64.43±0.39
AlignMixup [35]†	97.06±0.04	96.91±0.01	81.71±0.07	81.24±0.02	66.85±0.07
ζ-Mixup [1]⋆	96.26±0.04	96.35±0.04	80.46±0.26	79.73±0.15	63.18±0.14
MultiMix (ours)	97.07±0.03	97.06±0.02	81.82±0.04	81.44±0.03	67.11±0.04
Dense MultiMix (ours)	97.09±0.02	**97.09±0.02**	**81.93±0.04**	**81.77±0.03**	**68.44±0.05**
Gain	**-0.01**	**+0.09**	**+0.22**	**+0.53**	**+1.59**

Table 6: *Image classification* on CIFAR-10/100 and TI (TinyImagenet). Top-1 accuracy (%): higher is better. R: PreActResnet, W: WRN. ⋆: reproduced, †: reported by AlignMixup, ‡: reproduced with same teacher and student model. **Bold black**: best; Blue: second best; underline: best baseline. Gain: improvement over best baseline.

Network	ResNet-50		ViT-S/16	
Method	Speed	Acc	Speed	Acc
Baseline†	1.17	76.32	1.01	73.9
Input mixup [47]†	1.14	77.42	0.99	74.1
CutMix [45]†	1.16	78.60	0.99	74.2
Manifold mixup [36]†	1.15	77.50	0.97	74.2
PuzzleMix [16]†	0.84	78.76	0.73	74.7
Co-Mixup [15]†	0.62	–	0.57	74.9
SaliencyMix [33]†	1.14	78.74	0.96	74.8
StyleMix [14]†	0.99	75.94	0.85	74.8
StyleCutMix [14]†	0.76	77.29	0.71	74.9
SuperMix [6]‡	0.92	77.60	–	–
TransMix [3]⋆	–	–	1.01	75.1
TokenMix [24]⋆	–	–	0.87	75.3
AlignMixup [35]†	1.03	79.32	–	–
MultiMix (ours)	1.16	78.81	0.98	75.2
Dense MultiMix (ours)	0.95	**79.42**	0.88	**76.1**
Gain		**+0.1**		**+1.2**

Table 7: *Image classification and training speed* on ImageNet. Top-1 accuracy (%): higher is better. Speed: images/sec ($\times 10^3$): higher is better. †: reported by AlignMixup; ⋆: reproduced; ‡: reproduced with same teacher and student model. **Bold black**: best; Blue: second best; underline: best baseline. Gain: improvement over best baseline.

Attack	FGSM					PGD			
Dataset	CIFAR-10		CIFAR-100		TI	CIFAR-10		CIFAR-100	
Network	R-18	W16-8	R-18	W16-8	R-18	R-18	W16-8	R-18	W16-8
Baseline†	88.8±0.11	88.3±0.33	87.2±0.10	72.6±0.22	91.9±0.06	99.9±0.0	99.9±0.01	99.9±0.01	99.9±0.01
Input mixup [47]†	79.1±0.07	79.1±0.12	81.4±0.23	67.3±0.06	88.7±0.08	99.7±0.02	99.4±0.01	99.9±0.01	99.3±0.02
CutMix [45]†	77.3±0.06	78.3±0.05	86.9±0.06	60.2±0.04	88.6±0.03	99.8±0.03	98.1±0.02	98.6±0.01	97.9±0.01
Manifold mixup [36]†	76.9±0.14	76.0±0.04	80.2±0.06	56.3±0.10	89.3±0.06	97.2±0.01	98.4±0.03	99.6±0.01	98.4±0.03
PuzzleMix [16]†	57.4±0.22	60.7±0.02	78.8±0.09	57.8±0.03	83.8±0.05	97.7±0.01	97.0±0.01	96.4±0.02	95.2±0.03
Co-Mixup [15]†	60.1±0.05	58.8±0.10	77.5±0.02	56.5±0.04	–	97.5±0.02	96.1±0.03	95.3±0.03	94.2±0.01
SaliencyMix [33]†	57.4±0.08	68.0±0.05	77.8±0.10	58.1±0.06	81.1±0.06	97.4±0.03	97.0±0.04	95.6±0.03	93.7±0.05
StyleMix [14]†	80.0±0.23	71.2±0.21	80.6±0.15	68.2±0.17	85.1±0.16	98.1±0.09	97.5±0.07	98.3±0.09	98.3±0.09
StyleCutMix [14]†	57.7±0.04	56.0±0.07	77.4±0.05	56.8±0.03	80.5±0.04	97.8±0.04	96.7±0.02	91.8±0.01	93.7±0.01
SuperMix [6]‡	60.0±0.11	58.2±0.12	78.8±0.13	58.3±0.19	81.1±0.12	97.6±0.02	97.2±0.09	91.4±0.03	92.7±0.01
AlignMixup [35]†	54.8±0.03	56.0±0.05	74.1±0.04	55.0±0.03	78.8±0.03	95.3±0.04	96.7±0.03	90.4±0.01	92.1±0.03
ζ-Mixup [1]⋆	72.8±0.23	67.3±0.24	75.3±0.21	68.0±0.21	84.7±0.18	98.0±0.06	98.6±0.03	97.4±0.10	96.1±0.10
MultiMix (ours)	**54.1**±0.09	55.3±0.04	73.8±0.04	54.5±0.01	77.5±0.01	94.2±0.04	94.8±0.01	90.0±0.01	91.6±0.01
Dense MultiMix (ours)	**54.1**±0.01	**53.3**±0.03	**73.5**±0.03	**52.9**±0.04	**75.5**±0.04	**92.9**±0.04	**92.6**±0.01	**88.6**±0.03	**90.8**±0.01
Gain	**+0.7**	**+2.7**	**+0.6**	**+2.1**	**+3.3**	**+2.4**	**+3.5**	**+1.4**	**+1.3**

Table 8: *Robustness to FGSM & PGD attacks*. Top-1 error (%): lower is better. ⋆: reproduced, †: reported by AlignMixup. ‡: reproduced, same teacher and student model. **Bold black**: best; Blue: second best; underline: best baseline. Gain: reduction of error over best baseline. TI: TinyImagenet. R: PreActResnet, W: WRN.

Task	Out-Of-Distribution Detection											
Dataset	LSUN (crop)				iSUN				TI (crop)			
Metric	Det Acc	AuROC	AuPR (ID)	AuPR (OOD)	Det Acc	AuROC	AuPR (ID)	AuPR (OOD)	Det Acc	AuROC	AuPR (ID)	AuPR (OOD)
Baseline†	54.0	47.1	54.5	45.6	66.5	72.3	74.5	69.2	61.2	64.8	67.8	60.6
Input mixup [47]†	57.5	59.3	61.4	55.2	59.6	63.0	60.2	63.4	58.7	62.8	63.0	62.1
Cutmix [45]†	63.8	63.1	61.9	63.4	67.0	76.3	81.0	77.7	70.4	84.3	87.1	80.6
Manifold mixup [36]†	58.9	60.3	57.8	59.5	64.7	73.1	80.7	76.0	67.4	69.9	69.3	70.5
PuzzleMix [16]†	64.3	69.1	80.6	73.7	73.9	77.2	79.3	71.1	71.8	76.2	78.2	81.9
Co-Mixup [15]†	70.4	75.6	82.3	70.3	68.6	80.1	82.5	75.4	71.5	84.8	86.1	80.5
SaliencyMix [33]†	68.5	79.7	82.2	64.4	65.6	76.9	78.3	79.8	73.3	83.7	87.0	82.0
StyleMix [14]†	62.3	64.2	70.9	63.9	61.6	68.4	67.6	60.3	67.8	73.9	71.5	78.4
StyleCutMix [14]†	70.8	78.6	83.7	74.9	70.6	82.4	83.7	76.5	75.3	82.6	82.9	78.4
SuperMix [6]‡	70.9	77.4	80.1	72.3	71.0	76.8	79.6	76.7	75.1	82.8	82.5	78.6
AlignMixup [35]†	74.2	79.9	84.1	75.1	72.8	83.2	84.1	80.3	77.2	85.0	87.8	85.0
ζ-Mixup [1]⋆	68.1	73.2	80.8	73.1	72.2	82.3	82.2	79.4	74.4	84.3	82.2	77.2
MultiMix (ours)	79.2	82.6	85.2	77.6	75.6	85.1	87.8	83.1	78.3	86.6	89.0	**88.2**
Dense MultiMix (ours)	**80.8**	**84.3**	**85.9**	**78.0**	**76.8**	**85.4**	**88.0**	**84.6**	**81.4**	89.0	**90.8**	88.0
Gain	**+6.6**	**+4.4**	**+1.8**	**+2.9**	**+2.9**	**+2.2**	**+3.9**	**+4.3**	**+4.2**	**+4.0**	**+3.0**	**+3.2**

Table 9: *Out-of-distribution detection* using R-18. Det Acc (detection accuracy), AuROC, AuPR (ID) and AuPR (OOD): higher is better. ⋆: reproduced, †: reported by AlignMixup. ‡: reproduced, same teacher and student model. **Bold black**: best; Blue: second best; underline: best baseline. Gain: increase in performance. TI: TinyImagenet.

METHOD	VANILLA	DENSE
Baseline	76.76	78.16
Input mixup [47]	79.79	80.21
CutMix [45]	80.63	81.40
Manifold mixup [36]	80.20	80.87
PuzzleMix [16]	79.99	80.62
Co-Mixup [15]	80.19	80.84
SaliencyMix [33]	80.31	81.21
StyleMix [14]	79.96	80.76
StyleCutMix [14]	80.66	81.41
SuperMix [6]‡	79.01	80.12
AlignMixup [35]	81.71	81.36
MultiMix (ours)*	**81.81**	81.84
MultiMix (ours)	**81.81**	**81.88**

Table 10: *Image classification* on CIFAR-100 using R-18: The effect of Dense loss on SoTA mixup methods. Top-1 accuracy (%): higher is better. ‡: reproduced with same teacher and student model. *: Instead of Dense MultiMix, we only apply the loss densely.

A.3 More results: Out of distribution detection

This is a standard benchmark for evaluating over-confidence. Here, *in-distribution* (ID) are examples on which the network has been trained, and *out-of-distribution* (OOD) are examples drawn from any other distribution. Given a mixture of ID and OOD examples, the network should predict an ID example with high confidence and an OOD example with low confidence, *i.e.*, the confidence of the predicted class should be below a certain threshold.

Following AlignMixup [35], we compare MultiMix and its variants with SoTA methods trained using R-18 on CIFAR-100 as ID examples, while using LSUN [44], iSUN [39] and TI to draw OOD examples. We use detection accuracy, Area under ROC curve (AuROC) and Area under precision-recall curve (AuPR) as evaluation metrics. In Table 9, we observe that MultiMix and Dense MultiMix outperform SoTA on all datasets and metrics by a large margin. Although the gain of MultiMix and Dense MultiMix over SoTA mixup methods is small on image classification, they significantly reduce over-confident incorrect predictions and achieve superior performance on out-of-distribution detection.

A.4 More ablations

As in subsection 4.5, all ablations here are performed using R-18 on CIFAR-100.

Mixup methods with dense loss In Table 6 we observe that dense interpolation and dense loss improve MultiMix. Here, we study the effect of the dense loss only when applied to SoTA mixup methods; dense interpolation is not straightforward or not applicable in general with other methods.

Given a mini-batch of b examples, we follow the mixup strategy of the SoTA mixup methods to obtain the mixed embedding $\widetilde{Z}^j \in \mathbb{R}^{d\times b}$ for each spatial position $j = 1, \ldots, r$. Then, as discussed in subsection 3.3, we obtain the predicted class probabilities $\widetilde{P}^j \in \mathbb{R}^{c\times b}$ again for each $j = 1, \ldots, r$. Finally, we compute the cross-entropy loss $H(\widetilde{Y}, \widetilde{P}^j)$ (1) densely at each spatial position j, where the interpolated target label $\widetilde{Y} \in \mathbb{R}^{c\times b}$ is given by (4).

In Table 10, we observe that using a dense loss improves the performance of all SoTA mixup methods. The baseline improves by 1.4% accuracy (76.76 $\rightarrow$ 78.16) and manifold mixup by 0.67% (80.20 $\rightarrow$ 80.87). On average, we observe a gain of 0.7% brought by the dense loss. An exception is AlignMixup [35], which drops by 0.35% (81.71 $\rightarrow$ 81.36). This may be due to the alignment process, whereby the interpolated dense embeddings are not very far from the original. MultiMix and Dense MultiMix still improve the state of the art under this setting.

Training speed In Figure 5, we analyze the training speed of MultiMix and Dense MultiMix as a function of number n of interpolated examples. In terms of speed, MultiMix is on par with the baseline up to $n = 1000$, while bringing an accuracy gain of 5%. The best performing method—

[1] https://github.com/alldbi/SuperMix

METHOD	u	h	ACC
Uniform	–	–	81.33
Attention (9)	CAM	softmax	81.21
	CAM	$\ell_1 \circ \mathrm{relu}$	81.63
	GAP	softmax	81.78
	GAP	$\ell_1 \circ \mathrm{relu}$	**81.88**

Table 11: *Variants of spatial attention* in Dense MultiMix, on CIFAR-100 using R-18. Top-1 accuracy (%): higher is better. GAP: Global Average Pooling; CAM: Class Activation Maps [50]; $\ell_1 \circ \mathrm{relu}$: ReLU followed by ℓ_1 normalization.

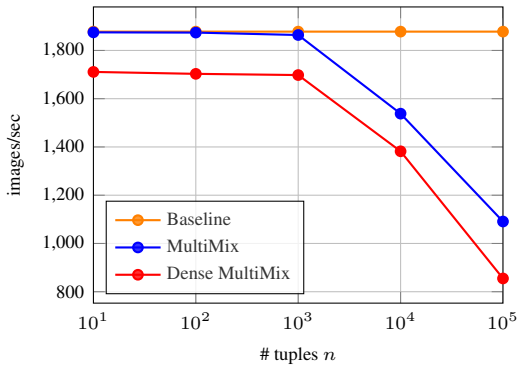


Figure 5: *Training speed* (images/sec) of MultiMix and its variants *vs*. number of tuples n on CIFAR-100 using R-18. Measured on NVIDIA RTX 2080 TI GPU, including forward and backward pass.

Dense MultiMix—is only slower by 10.6% at $n = 1000$ as compared to the baseline, which is arguably worth given the impressive 5.12% accuracy gain. Further increasing beyond $n > 1000$ brings a drop in training speed, due to computing Λ and then using it to interpolate (6),(7). Because $n > 1000$ also brings little performance benefit according to Figure 4(b), we set $n = 1000$ as default for all MultiMix variants.

Dense MultiMix: Spatial attention In subsection 3.3, we discuss different options for attention in dense MultiMix. In particular, no attention amounts to defining a uniform $a = \mathbf{1}_r/r$. Otherwise, a is defined by (9). The vector u can be defined as $u = \mathbf{z}\mathbf{1}_r/r$ by global average pooling (GAP) of $\mathbf{z}$, which is the default, or $u = Wy$ assuming a linear classifier with $W \in \mathbb{R}^{d \times c}$. The latter is similar to class activation mapping (CAM) [50], but here the current value of W is used online while training. The non-linearity h can be softmax or ReLU followed by ℓ_1 normalization ($\ell_1 \circ \mathrm{relu}$), which is the default. Here, we study the affect of these options on the performance of dense Multimix.

In Table 11, we observe that using GAP for u and $\ell_1 \circ \mathrm{relu}$ as h yields the best performance overall. Changing GAP to CAM or $\ell_1 \circ \mathrm{relu}$ to softmax is inferior. The combination of CAM with softmax is the weakest, even weaker than uniform attention. CAM may fail because of using the non-optimal value of W while training; softmax may fail because of being too selective. Compared to our best setting, uniform attention is clearly inferior, by nearly 0.6%. This validates that the use of spatial attention in dense MultiMix is clearly beneficial. Our intuition is that in the absence of dense targets, assuming the same target of the entire example at every spatial position naively implies that the object of interest is present everywhere, whereas spatial attention provides a better hint as to where the object may really be. We validate this hypothesis in Figure 6, where we visualize the attention maps obtained using our best setting with u as GAP and h as $\ell_1 \circ \mathrm{relu}$. This shows that the attention map enables dense targets to focus on the object regions, which explains its superior performance.

Dense MultiMix: Spatial resolution We study the effect of spatial resolution on dense MultiMix. By default, we use a resolution of 4×4 at the last residual block of R-18 on CIFAR-100. Here, we additionally investigate 1×1 (downsampling by average pooling with kernel size 4, same as GAP), 2×2 (downsampling by average pooling with kernel size 2) and 8×8 (upsampling by using stride 1 in the last residual block). We measure accuracy 81.07% for spatial resolution 1×1, 81.43% for for 2×2, 81.88% for 4×4 and 80.83% for 8×8. We thus observe that performance improves with

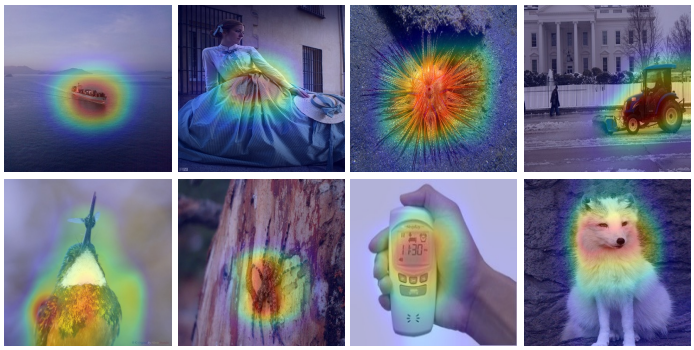


Figure 6: *Attention visualization.* Attention maps obtained by (9) with u as GAP and h as $\ell_1 \circ \mathrm{relu}$ using Resnet-50 on the validation set of ImageNet. The attention localizes the complete or part of the object with high confidence.

spatial resolution up to 4×4, which is the optimal, and then drops at 8×8. This drop may be due to assuming the same target at each spatial position. The resolution 8×8 is also more expensive computationally.