AlignMixup: Improving Representations By Interpolating Aligned Features

Shashanka Venkataramanan\textsuperscript{1} Ewa Kijak\textsuperscript{1} Laurent Amsaleg\textsuperscript{1} Yannis Avrithis\textsuperscript{2}
\textsuperscript{1}Inria, Univ Rennes, CNRS, IRISA \textsuperscript{2}IARAI, Athena RC

Abstract

Mixup is a powerful data augmentation method that interpolates between two or more examples in the input or feature space and between the corresponding target labels. However, how to best interpolate images is not well defined. Recent mixup methods overlay or cut-and-paste two or more objects into one image, which needs care in selecting regions. Mixup has also been connected to autoencoders, because often autoencoders generate an image that continuously deforms into another. However, such images are typically of low quality.

In this work, we revisit mixup from the deformation perspective and introduce AlignMixup, where we geometrically align two images in the feature space. The correspondences allow us to interpolate between two sets of features, while keeping the locations of one set. Interestingly, this retains mostly the geometry or pose of one image and the appearance or texture of the other. We also show that an autoencoder can still improve representation learning under mixup, without the classifier ever seeing decoded images. AlignMixup outperforms state-of-the-art mixup methods on five different benchmarks. Code available at https://github.com/shashankvkt/AlignMixup_CVPR22.git

1 Introduction

Data augmentation \cite{27, 33, 7} is a powerful regularization method that increases the amount and diversity of data, be it labeled or unlabeled \cite{12}. It improves the generalization performance and helps learning invariance \cite{38} at almost no cost, because the same example can be transformed in different ways over epochs. However, by operating on one image at a time and limiting to label-preserving transformations, it has limited chances of exploring beyond the image manifold.

Mixup operates on two or more examples at a time, interpolating between them in the input space \cite{51} or feature space \cite{45}, while also interpolating between target labels for image classification. This flattens class representations \cite{45}, reduces overly confident incorrect predictions, and smoothens decision boundaries far away from training data. However, input mixup images are overlays and tend to be unnatural \cite{49}. Interestingly, recent mixup methods focus of combining two \cite{49, 25} or more \cite{24} objects from different images into one in the input space, making efficient use of training pixels. However, randomness in the patch selection and thereby label mixing may mislead the classifier to learn uninformative features \cite{44}, which raises the question: what is a good interpolation of images?

Bengio et al. \cite{3} show that traversing along the manifold of representations obtained from deeper layers of the network more likely results in finding realistic examples. This is because the interpolated points smoothly traverse the underlying manifold of the data, capturing salient characteristics of the two images. Furthermore, \cite{4} show the ability of autoencoders to capture semantic correspondences obtained by decoding mixed latent codes. This is because the autoencoder may disentangle
Alignment refers to finding a geometric correspondence between image elements before interpolation. The feature tensor is ideal for this purpose, because its spatial resolution is low,
Our feature tensor alignment is based on optimal transport theory [46] and Sinkhorn distance (SD) [8] in particular. Let \( \mathbf{A} := \mathcal{F}(x), \mathbf{A}' := \mathcal{F}(x') \) be the \( c \times w \times h \) feature tensors of images \( x, x' \in \mathcal{X} \). We reshape them to \( c \times r \) matrices \( \mathbf{A}, \mathbf{A}' \) by flattening the spatial dimensions, where \( r := hw \). Then, every column \( a_j, a'_j \in \mathbb{R}^c \) of \( \mathbf{A}, \mathbf{A}' \) for \( j = 1, \ldots, r \) is a feature vector representing a spatial position in the original image \( x, x' \). Let \( M \) be the \( r \times r \) cost matrix with its elements being the pairwise distances of these vectors:

\[
m_{ij} := \|a_i - a'_j\|^2
\]

for \( i, j \in \{1, \ldots, r\} \). We are looking for a transport plan, that is, a \( r \times r \) matrix \( P \in U_r \), where

\[
U_r := \{ P \in \mathbb{R}^{r \times r} : P1 = P^T1 = 1/r \}
\]

and \( 1 \) is an all-ones vector in \( \mathbb{R}^r \). That is, \( P \) is non-negative with row-wise and column-wise sum \( 1/r \), representing a joint probability over spatial positions of \( \mathbf{A}, \mathbf{A}' \) with uniform marginals. It is chosen to minimize the expected pairwise distance of their features, as expressed by the linear cost function \( \langle P, M \rangle \), under an entropic regularizer:

\[
P^* = \arg \min_{P \in U_r} \langle P, M \rangle - \epsilon H(P),
\]

where \( H(P) := -\sum_{ij} p_{ij} \log p_{ij} \) is the entropy of \( P \), \( \langle \cdot, \cdot \rangle \) is Frobenius inner product and \( \epsilon \) is a regularization coefficient. The optimal solution \( P^* \) is unique and can be found by forming the \( r \times r \) similarity matrix \( e^{-M/\epsilon} \) and then applying the Sinkhorn-Knopp algorithm [26], i.e., iteratively normalizing rows and columns. A small \( \epsilon \) leads to sparser \( P \), which improves one-to-one matching but makes the optimization harder [1], while a large \( \epsilon \) leads to denser \( P \), causing more correspondences and poor matching.

**Interpolation** The assignment matrix \( R := rP^* \) is a doubly stochastic \( r \times r \) matrix whose element \( r_{ij} \) expresses the probability that column \( a_i \) of \( \mathbf{A} \) corresponds to column \( a'_j \) of \( \mathbf{A}' \). Thus, we align \( \mathbf{A} \) and \( \mathbf{A}' \) as follows:

\[
\tilde{\mathbf{A}} := \mathbf{A}' R^T
\]

\[
\tilde{\mathbf{A}}' := \mathbf{A} R.
\]

Here, column \( \tilde{a}_j \) of \( c \times r \) matrix \( \tilde{\mathbf{A}} \) is a convex combination of columns of \( \mathbf{A}' \) that corresponds to the same column \( a_i \) of \( \mathbf{A} \). We reshape \( \tilde{\mathbf{A}} \) back to \( c \times w \times h \) tensor \( \tilde{\mathbf{A}} \) by expanding spatial dimensions and we say that \( \tilde{\mathbf{A}} \) represents \( \mathbf{A} \) aligned to \( \mathbf{A}' \). We then interpolate between \( \tilde{\mathbf{A}} \) and the original feature tensor \( \mathbf{A} \):

\[
\text{mix}_\lambda (\mathbf{A}, \tilde{\mathbf{A}}).
\]
We use a decoder to study images generated with or without feature alignment. Let \( f : \mathbb{R}^{c \times w \times h} \rightarrow \mathbb{R}^d \) be a FC layer mapping tensor \( A \) to embedding \( e = f(A) \). We use \( f \circ D \) as an encoder and a decoder \( D : \mathbb{R}^d \rightarrow \mathcal{X} \) mapping \( e \) back to the image space, reconstructing image \( \hat{x} = D(e) \). The autoencoder is trained using only clean images (without mixup) using reconstruction loss \( L_r \) between \( x \) and \( \hat{x} \), where \( L_r(x, x') := \|x - x'\|^2 \) is the squared Euclidean distance. We use generated images only for visualization purposes below, but we also use the decoder optionally during AlignMixup training in section 3.

**2.3 Visualization and discussion**

**Decoder** We use a decoder to study images generated with or without feature alignment. Let \( f : \mathbb{R}^{c \times w \times h} \rightarrow \mathbb{R}^d \) be a FC layer mapping tensor \( A \) to embedding \( e = f(A) \). We use \( f \circ D \) as an encoder and a decoder \( D : \mathbb{R}^d \rightarrow \mathcal{X} \) mapping \( e \) back to the image space, reconstructing image \( \hat{x} = D(e) \). The autoencoder is trained using only clean images (without mixup) using reconstruction loss \( L_r \) between \( x \) and \( \hat{x} \), where \( L_r(x, x') := \|x - x'\|^2 \) is the squared Euclidean distance. We use generated images only for visualization purposes below, but we also use the decoder optionally during AlignMixup training in section 3.

**Discussion** For different \( \lambda \in [0, 1] \), we interpolate the feature tensors \( A, A' \) of \( x, x' \) without or with alignment, using (11) or (12), and we generate a new image by decoding the resulting embedding through the decoder \( D \).

In Figure 2, we visualize such generated images. Interestingly, by aligning \( A \) to \( A' \) and mixing using (11) with \( \lambda = 0 \), the generated image retains the pose of \( x \) and the texture of \( x' \). In Figure 2(a) in particular, when \( x \) is ‘penguin’ and \( x' \) is ‘dog’, the generated image retains the pose of the penguin, while the texture of the dog aligns to the body of the penguin. Similarly, in Figure 2(c), the texture from the goldfish is aligned to that of the stork, while the pose of the stork is retained. Vice versa, as shown in Figure 2(b,d), by aligning \( A' \) to \( A \) and mixing using (12) with \( \lambda = 0 \), the generated image retains the pose of \( x' \) and the texture of \( x \). By contrast, the image generated from unaligned features appears to be an overlay.

Randomly sampling several values of \( \lambda \in [0, 1] \) during training generates an abundance of samples, capturing texture from one image and the pose from another. This allows the model to explore beyond the image manifold, thereby improving its generalization and enhancing its performance across multiple benchmarks, as discussed in section 3.
As shown in Table 1(a), AlignMixup is on par or outperforms the SOTA methods by achieving the lowest top-1 error, especially on large datasets. On CIFAR-10, AlignMixup is on par with Co-Mixup and PuzzleMix with R-18 and WRN16-8. On CIFAR-100, AlignMixup outperforms Manifold mixup by 1.51% and 0.46% with R-18 and WRN16-8, respectively. On TI, AlignMixup outperforms Co-Mixup by 2.72% using R-18. From Table 1(b), AlignMixup outperforms PuzzleMix by 0.56% on ImageNet.

Robustness to FGSM and PGD attacks Following the evaluation protocol of [25] to evaluate AlignMixup robustness to FGSM and PGD attacks As shown in Table 2, AlignMixup is more robust comparing to SOTA methods. While AlignMixup is on par with PuzzleMix and Co-Mixup on CIFAR-10 image classification, it outperforms Co-Mixup and PuzzleMix by 5.36% and 2.28% in terms of robustness to FGSM attacks. There is also significant gain of robustness to FGSM on Tiny-ImageNet and to the stronger PGD on CIFAR-100.

4 Conclusion

We have shown that mixup of a combination of input and latent representations is a simple and very effective pairwise data augmentation method. The gain is most prominent on large datasets and in combating overconfidence in predictions, as indicated by out-of-distribution detection. Interpolation of feature tensors boosts performance significantly, but only if they are aligned. Our work is a compromise between a “good” hand-crafted interpolation in the image space and a fully learned one in the latent space. A challenge is to make progress in the latter direction without compromising speed and simplicity, which would affect wide applicability.
References


[37] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover’s distance as a metric for image retrieval. *IJCV*, 2000. 8


[39] Oriane Siméoni, Yannis Avrithis, and Ondrej Chum. Local features and visual words emerge in activations. In *CVPR*, 2019. 8

[40] Cecilia Summers and Michael J Dinneen. Improved mixed-example data augmentation. In *WACV*, 2019. 8


[49] 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, 5, 8, 9, 10, 11, 12


[51] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 1, 2, 5, 8, 9, 10, 11, 12


A Related Work

Mixup [51], concurrently with similar methods [23, 43], introduce mixup, augmenting data by linear interpolation between two examples. While [51] apply mixup on intermediate representations, it is [45] who make this work, introducing manifold mixup. Without alignment, the result is an overlay of either images [51] or features [45]. [17] eliminate “manifold intrusion”—mixed data conflicting with true data. Unlike manifold mixup, AlignMixup interpolates feature tensors from deeper layers after aligning them.

Nonlinear mixing over random image regions is an alternative, e.g. from masking square regions [10] to cutting a rectangular region from one image and pasting it onto another [49], as well as several variants using arbitrary regions [41, 40, 19]. Instead of choosing regions at random, saliency can be used to locate objects from different images and fit them in one [44, 34, 25, 24]. Exploiting the knowledge of a teacher network to mix images based on saliency has been proposed in [9]. Instead of combining more than one objects in an image, AlignMixup attempts to deform one object into another.

Another alternative is Automix [53], which employs a U-Net rather than an autoencoder, mixing at several layers. It is limited to small datasets and provides little improvement over manifold mixup [45]. StyleMix and StyleCutMix [21] interpolate content and style between two images, using AdaIN [22], a style transfer autoencoder network. By contrast, AlignMixup aligns feature tensors and interpolates matching features directly, without using any additional network.

Alignment Local correspondences from intra-class alignment of feature tensors have been used in image registration [6, 31], optical flow [47], semantic alignment [36, 18] and image retrieval [39]. Here, we mostly use inter-class alignment. In few-shot learning, local correspondences between query and support images are important in finding attention maps, used e.g. by CrossTransformers [11] and DeepEMD [50]. The earth mover’s distance (EMD) [37], or Wasserstein metric, is an instance of optimal transport [46], addressed by linear programming. To accelerate, [8] computes optimal matching by Sinkhorn distance with entropic regularization. This distance is widely applied between distributions in generative models [14, 32].

EMD has been used for mixup in the input space, for instance point mixup for 3D point clouds [5] and OptTransMix for images [53], which is the closest to our work. However, aligning coordinates only applies to images with clean background. We rather align tensors in the feature space, which is generic. We do so using the Sinkhorn distance, which is orders of magnitude faster than EMD [8].

B Algorithm

AlignMixup and AlignMixup/AE are summarized in algorithm 1. By default (AlignMixup), for each mini-batch, we uniformly draw at random one among three choices (line 2) over mixup on input (x) or feature tensors (A, using either (11) or (12) for mixing). For AlignMixup/AE, there is a fourth choice where we only use reconstruction loss on clean examples (line 7).

For mixup, we use only classification loss (5) (line 24). Following [45], we form, for each example (x, y) in the mini-batch, a paired example (x’, y’) from the same mini-batch regardless of class labels, by randomly permuting the indices (lines 1,10). Inputs x, x’ are mixed by (2),(3) (line 12). Feature tensors A and A’ are first aligned and then mixed by (2),(11) (A aligns to A’) or (2),(12) (A’ aligns to A) (lines 14,23).

In computing loss derivatives, we backpropagate through feature tensors A, A’ but not through the transport plan P∗ (line 20). Hence, although the Sinkhorn-Knopp algorithm [26] is differentiable, its iterations take place only in the forward pass. Importantly, AlignMixup is easy to implement and does not require sophisticated optimization like [25, 24].

C Hyperparameter settings

CIFAR-10/CIFAR-100 We train AlignMixup using SGD for 2000 epochs with an initial learning rate of 0.1, decayed by a factor 0.1 every 500 epochs. We set the momentum as 0.9 with a weight decay of 0.0001 and use a batch size of 128. The interpolation factor is drawn from Beta(α, α)
Algorithm 1: AlignMixup/AE (parts involved in the AE variant indicated in blue)

```
Input: encoders $F$, embedding $e$, decoder $D$; classifier $g$
Input: mini-batch $B := \{(x_i, y_i)\}_{i=1}^b$
Output: loss values $L := \{\ell_i\}_{i=1}^b$

1. $\pi \sim \text{unif}(S_b)$  \hspace{1cm} \triangleright \text{random permutation of } \{1, \ldots, b\}
2. $\text{mode} \sim \text{unif}\{\text{clean}, \text{input}, \text{feat}, \text{feat}'\}$ \hspace{1cm} \triangleright \text{mixup?}
3. for $i \in \{1, \ldots, b\}$ do
4. \hspace{1cm} $(x, y) \leftarrow (x_i, y_i)$  \hspace{1cm} \triangleright \text{current example}
5. \hspace{1cm} if $\text{mode} = \text{clean}$ then
6. \hspace{2cm} $\hat{x} \leftarrow D(e(F(x)))$ \hspace{1cm} \triangleright \text{encode/decode}
7. \hspace{2cm} $\ell_i \leftarrow L_r(x, \hat{x})$ \hspace{1cm} \triangleright \text{reconstruction loss}
8. \hspace{1cm} else
9. \hspace{2cm} $\lambda \sim \text{Beta}(\alpha, \alpha)$ \hspace{1cm} \triangleright \text{interpolation factor}
10. \hspace{2cm} $(x', y') \leftarrow (x_{\pi(i)}, y_{\pi(i)})$ \hspace{1cm} \triangleright \text{paired example}
11. \hspace{2cm} if $\text{mode} = \text{input}$ then
12. \hspace{3cm} out $\leftarrow F(\text{mix}_a(x, x'))$ \hspace{1cm} \triangleright (2),(3)
13. \hspace{3cm} else \hspace{1cm} \triangleright \text{choose (12) over (11)}
14. \hspace{3cm} if $\text{mode} = \text{feat}$ then
15. \hspace{4cm} $\text{SWAP}(x, x'), \text{SWAP}(y, y')$ \hspace{1cm} \triangleright \text{as in [51]}
16. \hspace{4cm} $A \leftarrow F(x), A' \leftarrow F(x')$ \hspace{1cm} \triangleright \text{feature tensors}
17. \hspace{4cm} $A \leftarrow \text{RESHAPE}_{e \times r}(A)$ \hspace{1cm} \triangleright \text{to matrix}
18. \hspace{4cm} $A' \leftarrow \text{RESHAPE}_{e \times r}(A')$ \hspace{1cm} \triangleright \text{pairwise distances (6)}
19. \hspace{4cm} $P' \leftarrow \text{SINKHORN}(\exp(-M/\epsilon))$ \hspace{1cm} \triangleright \text{tran. plan (8)}
20. \hspace{4cm} $R \leftarrow \text{DETACH}(P'')$ \hspace{1cm} \triangleright \text{assignments}
21. \hspace{4cm} $\tilde{A} \leftarrow A'R''$ \hspace{1cm} \triangleright \text{alignment (9)}
22. \hspace{4cm} $\tilde{A} \leftarrow \text{RESHAPE}_{e \times w \times h}(\tilde{A})$ \hspace{1cm} \triangleright \text{to tensor}
23. \hspace{4cm} out $\leftarrow f(\text{mix}_a(A, \tilde{A}))$ \hspace{1cm} \triangleright (2),(11)
24. \hspace{2cm} $\ell_i \leftarrow L_c(g(out), \text{mix}_a(y, y'))$ \hspace{1cm} \triangleright \text{classification loss (5)}
```

where $\alpha = 2.0$. Using these settings, we reproduce the results of SOTA mixup methods for image classification, robustness to FGSM and PGD attacks, calibration and out-of-distribution detection. For alignment, we apply the Sinkhorn-Knopp algorithm [26] for 100 iterations with entropic regularization coefficient $\epsilon = 0.1$.

**TinyImageNet** We follow the training protocol of Kim et al. [25], training R-18 as stage-1 encoder $F$ using SGD for 1200 epochs. We set the initial learning rate to 0.1 and decay it by 0.1 at 600 and 900 epochs. We set the momentum as 0.9 with a weight decay of 0.0001 and use a batch size of 128 on 2 GPUs. The interpolation factor is drawn from $\text{Beta}(\alpha, \alpha)$ where $\alpha = 2.0$. For alignment, we apply the Sinkhorn-Knopp algorithm [26] for 100 iterations with entropic regularization coefficient $\epsilon = 0.1$.

**ImageNet** We follow the training protocol of Kim et al. [25], where training R-50 as $F$ using SGD for 300 epochs. The initial learning rate of the classifier and the remaining layers is set to 0.1 and 0.01, respectively. We decay the learning rate by 0.1 at 100 and 200 epochs. We set the momentum as 0.9 with a weight decay of 0.0001 and use a batch size of 100 on 4 GPUs. The interpolation factor is drawn from $\text{Beta}(\alpha, \alpha)$ where $\alpha = 2.0$. For alignment, we apply the Sinkhorn-Knopp algorithm [26] for 100 iterations with entropic regularization coefficient $\epsilon = 0.1$.

We also train R-50 on ImageNet for 100 epochs, following the training protocol described in Kim et al. [24].

**CUB200-2011** For weakly-supervised object localization (WSOL), we use VGG-GAP and R-50 pretrained on ImageNet as $F$. The training strategy for WSOL is the same as image classification and the network is trained without bounding box information. In R-50, following [49], we modify the last residual block (layer 4) to have stride 2 instead of 1, resulting in a feature map of spatial
Network RESNET-50
Baseline 24.03
Input [51] 22.97
Manifold [45] 23.30
CutMix [29] 22.92
PuzzleMix [25] 22.49
StyleMix [21] 24.06
StyleCutMix [21] 22.71
AlignMixup (ours) 22.0

Gain +0.39


<table>
<thead>
<tr>
<th>TASK</th>
<th>OUT-OF-DISTRIBUTION DETECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATASET</td>
<td>LSUN (CROP)</td>
</tr>
<tr>
<td>METRIC</td>
<td>DET Acc</td>
</tr>
<tr>
<td>Baseline</td>
<td>54.0</td>
</tr>
<tr>
<td>Cutmix [49]</td>
<td>63.8</td>
</tr>
<tr>
<td>Manifold [45]</td>
<td>58.9</td>
</tr>
<tr>
<td>PuzzleMix [25]</td>
<td>64.3</td>
</tr>
<tr>
<td>Co-Mixup [24]</td>
<td>70.4</td>
</tr>
<tr>
<td>SaliencyMix [24]</td>
<td>68.5</td>
</tr>
<tr>
<td>StyleMix [21]</td>
<td>62.3</td>
</tr>
<tr>
<td>StyleCutMix [21]</td>
<td>70.8</td>
</tr>
</tbody>
</table>

Gain +6.1 +3.8 +3.0 +4.5 +1.7 +1.7 +2.2 +1.9 +1.7 +3.2 +2.6 +3.8

Table 4: Out-of-distribution detection using PreActResnet18. Det Acc (detection accuracy), AuROC, AuPR (ID) and AuPR (OOD): higher is better; Blue: second best. Gain: increase in performance. TI: TinyImagenet. Additional results are in the supplementary material.

Resolution 14 × 14. The modified architecture of VGG-GAP is the same as described in [52]. The classifier is modified to have 200 classes instead of 1000.

For fair comparisons with [49], during training, we resize the input image to 256 × 256 and randomly crop the resized image to 224 × 224. During testing, we directly resize to 224 × 224. We train the network for 600 epochs using SGD. For R-50, the initial learning rate of the classifier and the remaining layers is set to 0.01 and 0.001, respectively. For VGG, the initial learning rate of the classifier and the remaining layers is set to 0.001 and 0.0001, respectively. We decay the learning rate by 0.1 every 150 epochs. The momentum is set to 0.9 with weight decay of 0.0001 and batch size of 16.

![Calibration plots](image)

Figure 3: Calibration plots on CIFAR-100 using PreActResnet18: near diagonal is better. We plot accuracy vs. confidence, that is, probability for the predicted class.

D Additional experiments

**ImageNet classification** Following the training protocol of [24], Table 3 reports classification performance when training for 100 epochs on ImageNet. Using the top-1 error (%) reported for
<table>
<thead>
<tr>
<th>Metric</th>
<th>ECE</th>
<th>OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>10.25</td>
<td>1.11</td>
</tr>
<tr>
<td>Input [51]</td>
<td>18.50</td>
<td>1.42</td>
</tr>
<tr>
<td>CutMix [49]</td>
<td>7.60</td>
<td>1.05</td>
</tr>
<tr>
<td>Manifold [45]</td>
<td>18.41</td>
<td>0.79</td>
</tr>
<tr>
<td>PuzzleMix [25]</td>
<td>8.22</td>
<td>0.61</td>
</tr>
<tr>
<td>Co-Mixup [24]</td>
<td>5.83</td>
<td>0.55</td>
</tr>
<tr>
<td>SaliencyMix [44]</td>
<td>5.89</td>
<td>0.59</td>
</tr>
<tr>
<td>StyleMix [21]</td>
<td>11.43</td>
<td>1.31</td>
</tr>
<tr>
<td>StyleCutMix [21]</td>
<td>9.30</td>
<td>0.87</td>
</tr>
<tr>
<td>AlignMixup (ours)</td>
<td>5.78</td>
<td>0.41</td>
</tr>
<tr>
<td>AlignMixup/AE (ours)</td>
<td>5.06</td>
<td>0.48</td>
</tr>
</tbody>
</table>


competitors by [24], AlignMixup outperforms all methods, including Co-Mixup [24]. Importantly, while the overall improvement by SOTA methods over Baseline is around 1.64%, AlignMixup improves SOTA by another 0.4%.

**Experiments using transformers** We apply mixup to LeViT-128S [15] on ImageNet for 100 epochs. For AlignMixup, we align the feature tensors in the last layer of the convolution stem. The top-1 accuracy is: baseline 67.4%, input mixup 68.3%, manifold mixup 67.8%, CutMix 68.7%, AlignMixup 69.9%. Thus, we outperform input mixup and CutMix by 1.6% and 1.2% respectively, which in turn outperform the baseline by 0.9% and 1.3% respectively. This means that the improvement brought by mixing is roughly doubled.

**Out-of-distribution detection** We compare AlignMixup with SOTA methods, training R-18 on CIFAR-100. At inference, ID examples are test images from CIFAR-100, while OOD examples are test images from LSUN [48] and Tiny-ImageNet, resizing OOD examples to 32 × 32 to match the resolution of ID images [49]. We also use test images from CIFAR-100 with Uniform and Gaussian noise as OOD samples. Uniform is drawn from $U(0, 1)$ and Gaussian from $N(\mu, \sigma)$ with $\mu = \sigma = 0.5$. All SOTA mixup methods are reproduced using the same experimental settings. Following [20], we measure detection accuracy (Det Acc) using a threshold of 0.5, area under ROC curve (AuROC) and area under precision-recall curve (AuPR).

As shown in Table 4, AlignMixup outperforms SOTA methods under all metrics by a large margin, indicating that it is better in reducing over-confident predictions.

**Calibration** We compare AlignMixup with SOTA methods, training R-18 on CIFAR-100. All SOTA mixup methods are reproduced using the same experimental settings. We compare qualitatively by plotting accuracy vs. confidence. As shown in Figure 3, while Baseline is clearly over-confident and Input and Manifold mixup are clearly under-confident, AlignMixup results in the best calibration among all competitors. We also compare quantitatively, measuring the expected calibration error (ECE) [16] and overconfidence error (OE) [42]. As shown in Table 5, AlignMixup outperforms SOTA methods by achieving lower ECE and OE, indicating that it is better calibrated.

**Qualitative results of WSOL** Qualitative localization results shown in Figure 4 indicate that AlignMixup encodes semantically discriminative representations, resulting in better localization performance.

**Object detection** Following the settings of CutMix [49], we use Resnet-50 pretrained on ImageNet using AlignMixup as the backbone of SSD [29] and Faster R-CNN [35] detectors and fine-tune it on Pascal VOC07 [13] and MS-COCO [28] respectively. AlignMixup outperforms CutMix mAP by 0.8% (77.6 → 78.4) on Pascal VOC07 and 0.7% (35.16 → 35.84) on MS-COCO.
Figure 4: *Localization examples* using ResNet-50 on CUB200-2011. Red boxes: predicted; green: ground truth.

### E Additional ablations

<table>
<thead>
<tr>
<th>Iterations (i)</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignMixup</td>
<td>80.98</td>
<td>80.96</td>
<td>81.31</td>
<td>81.42</td>
<td>81.71</td>
<td>81.50</td>
<td>81.34</td>
<td>81.28</td>
</tr>
</tbody>
</table>

Table 6: *Ablation* of the number of iterations in Sinkhorn-Knopp algorithm using R-18 on CIFAR-100. Top-1 classification accuracy(%): higher is better.

**Iterations in Sinkhorn-Knopp** The default number of iterations for the Sinkhorn-Knopp algorithm in solving (8) is $i = 100$. Here, we investigate more choices, as shown in Table 6. The case of $i = 0$ is similar to cross-attention. In this case, we only normalize either the rows or columns in (7) once, such that $P \mathbf{1} = 1/r$ (when $A$ aligned to $A'$) or $P^\top \mathbf{1} = 1/r$ (when $A'$ aligned to $A$). We observe that while AlignMixup outperforms the best baseline—StyleCutMix (80.66)—in all cases, it performs best for $i = 100$ iterations.