# SUPPLEMENTARY MATERIAL FOR CO-MIXUP: SALIENCY GUIDED JOINT MIXUP WITH SUPERMODULAR DIVERSITY

**Anonymous authors**
Paper under double-blind review

## 1 SUPPLEMENTARY NOTES FOR OBJECTIVE

### 1.1 NOTATIONS

In Table 1, we provide a summary of notations in the main text.

| Notation | Meaning |
|---|---|
| $m$, $m'$, $n$ | # inputs, # outputs, dimension of data |
| $c_k \in \mathbb{R}^m$ $(1 \le k \le n)$ | labeling cost for $m$ input sources at the $k^{th}$ location |
| $z_{j,k} \in \mathcal{L}^m$ $(1 \le j \le m',\ 1 \le k \le n)$ | input source ratio at the $k^{th}$ location of the $j^{th}$ output |
| $z_j \in \mathcal{L}^{m \times n}$ | labeling of the $j^{th}$ output |
| $o_j \in \mathbb{R}^m$ | aggregation of the labeling of the $j^{th}$ output |
| $A \in \mathbb{R}^{m \times m}$ | compatibility between inputs |

Table 1: A summary of notations.

### 1.2 INTERPRETATION OF COMPATIBILITY

In our main objective Equation (1), we introduce a compatibility matrix $A = (1-\omega)I + \omega A_c$ between inputs. By minimizing $\langle o_j, o_{j'} \rangle_A$ for $j \neq j'$, we encourage each individual mixup examples to have high compatibility within. Figure 1 explains how the compatibility term works by comparing simple cases. Note that our framework can reflect any compatibility measures for the optimal mixup.

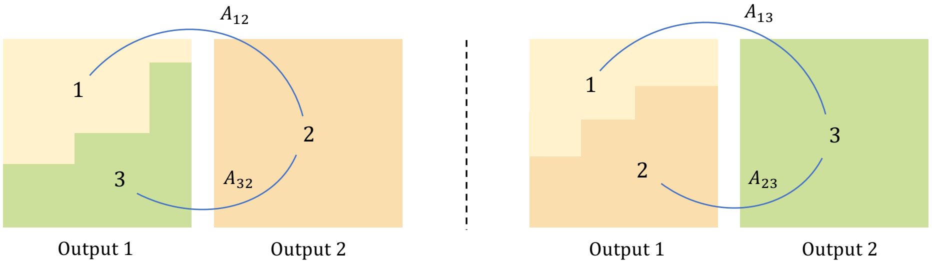

Figure 1: Let us consider Co-Mixup with three inputs and two outputs. The figure represents two Co-Mixup results. Each input is denoted as a number and color-coded. Let us assume that input 1 and input 2 are more compatible, *i.e.*, $A_{12} \gg A_{23}$ and $A_{12} \gg A_{13}$. Then, the left Co-Mixup result has a larger inner-product value $\langle o_1, o_2 \rangle_A$ than the right. Thus the mixup result on the right has higher compatibility than the result on the left within each output example.

# 2 PROOFS

## 2.1 PROOF OF PROPOSITION 1

**Lemma 1.** *For a positive semi-definite matrix* $A \in \mathbb{R}_+^{m\times m}$ *and* $x, x' \in \mathbb{R}^m$, $s(x, x') = x^\intercal A x'$ *is pairwise supermodular.*

*Proof.* $s(x, x) + s(x', x') - s(x, x') - s(x', x) = x^\intercal A x + x^\intercal A x - 2x^\intercal A x' = (x - x')^\intercal A(x - x')$, and because $A$ is positive semi-definite, $(x - x')^\intercal A(x - x') \geq 0$. ☐

**Proposition 1.** *The compatibility term* $f_c$ *in Equation (1) is pairwise supermodular for every pair of* $(z_{j_1,k}, z_{j_2,k})$ *if* $A$ *is positive semi-definite.*

*Proof.* For $j_1$ and $j_2$, *s.t.*, $j_1 \neq j_2$, $\max\left\{\tau, \sum_{j=1}^{m'} \sum_{j'=1,j'\neq j}^{m'} (\sum_{k=1}^{n} z_{j,k})^\intercal A(\sum_{k=1}^{n} z_{j',k})\right\} = \max\{\tau, c + 2z_{j_1,k}^\intercal A z_{j_2,k}\} = -\min\{-\tau, -c - 2z_{j_1,k}^\intercal A z_{j_2,k}\}$, for $\exists c \in \mathbb{R}$. By Lemma 1, $-z_{j_1,k}^\intercal A z_{j_2,k}$ is pairwise submodular, and because a budget additive function preserves submodularity (Horel and Singer, 2016), $\min\{-\tau, -c - 2z_{j_1,k}^\intercal A z_{j_2,k}\}$ is pairwise submodular with respect to $(z_{j_1,k}, z_{j_2,k})$. ☐

## 2.2 PROOF OF PROPOSITION 3

**Proposition 3.** *The modularization given by Equation (2) satisfies the criteria.*

*Proof.* Note, by the definition in Equation (1), the compatibility between the $j^{th}$ and $j'^{th}$ outputs is $o_{j'}^\intercal A o_j$, and thus, $v_{\text{-}j}^\intercal o_j$ represents the compatibility between the $j^{th}$ output and the others. In addition, $\|o_j\|_1 = \|\sum_{k=1}^{n} z_{j,k}\|_1 = \sum_{k=1}^{n} \|z_{j,k}\|_1 = n$. In a local view, for the given $o_j$, let us define a vector $o'_j$ as $o'_j[i_1] = o_j[i_1] + \alpha$ and $o'_j[i_2] = o_j[i_2] - \alpha$ for $\alpha > 0$. Without loss of generality, let us assume $v_{\text{-}j}$ is sorted in ascending order. Then, $v_{\text{-}j}^\intercal o_j \leq v_{\text{-}j}^\intercal o'_j$ implies $i_1 > i_2$, and because the max function preserves the ordering, $\max\{\tau', v_{\text{-}j}\}^\intercal o_j \leq \max\{\tau', v_{\text{-}j}\}^\intercal o'_j$. Thus, the criterion 1 is locally satisfied. Next, for $\tau' > 0$, $\|\max\{\tau', v_{\text{-}j}\}^\intercal o_j\|_1 \geq \tau'\|o_j\|_1 = \tau' n$. Let $\exists i_0$ s.t. for $i < i_0, v_{\text{-}j}[i] < \tau'$, and for $i \geq i_0, v_{\text{-}j}[i] \geq \tau'$. Then, for $o_j$ containing positive elements only in indices smaller than $i_0$, $\max\{\tau', v_{\text{-}j}\}^\intercal o_j = \tau' n$ which means there is no extra penalty from the compatibility. In this respect, the proposed modularization satisfies the criterion 2 as well. ☐

# 3 IMPLEMENTATION DETAILS

We perform the optimization after down-sampling the given inputs and saliency maps to the specified size ($4 \times 4$). After the optimization, we up-sample the optimal labeling to match the size of the inputs and then mix inputs according to the up-sampled labeling. For the saliency measure, we calculate the gradient values of training loss with respect to the input data and measure $\ell_2$ norm of the gradient values across input channels (Simonyan et al., 2013). In classification experiments, we retain the gradient information of network weights obtained from the saliency calculation for regularization. For the distance in the compatibility term, we measure $\ell_1$-distance between the most salient regions.

For the initialization in Algorithm 1, we use *i.i.d.* samples from a categorical distribution with equal probabilities. We use *alpha-beta* swap algorithm from pyGCO[1] to solve the minimization step in Algorithm 1, which can find local-minima of a multi-label submodular function. However, the worst-case complexity of *alpha-beta* swap algorithm with $|\mathcal{L}| = 2$ is $O(m^2 n)$, and in the case of mini-batch with 100 examples, iteratively applying the algorithm can become a bottleneck during the network training. To mitigate the computational overhead, we partition the mini-batch (each of size 20) and then apply Algorithm 1 independently per each partition.

---

[1] https://github.com/Borda/pyGCO

The worst-case complexity theoretic of the naive implementation of Algorithm 1 increases exponentially as $|\mathcal{L}|$ increases. Specifically, the worst-case theoretic complexity of the *alpha-beta* swap algorithm is proportional to the square of the number of possible states of $z_{j,k}$, which is proportional to $m^{|\mathcal{L}|-1}$. To reduce the number of possible states in a multi-label case, we solve the problem for binary labels ($|\mathcal{L}| = 2$) at the first inner-cycle and then extend to multi labels ($|\mathcal{L}| = 3$) only for the currently assigned indices of each output in the subsequent cycles. This reduces the number of possible states to $O(m + \bar{m}^{|\mathcal{L}|-1})$ where $\bar{m} \ll m$. Here, $\bar{m}$ means the number of currently assigned indices for each output.

Based on the above implementation, we train models with Co-Mixup in a feasible time. For example, in the case of ImageNet training with 16 Intel I9-9980XE CPU cores and 4 NVIDIA RTX 2080Ti GPUs, Co-Mixup training requires 0.964s per batch, whereas the vanilla training without mixup requires 0.374s per batch. Note that Co-Mixup requires saliency computation, and when we compare the algorithm with Puzzle Mix, which performs the same saliency computation, Co-Mixup is only slower about 1.04 times. Besides, as we down-sample the data to the fixed size regardless of the data dimension, the additional computation cost of Co-Mixup relatively decreases as the data dimension increases. Finally, we present the empirical time complexity of Algorithm 1 in Figure 2. As shown in the figure, Algorithm 1 has linear time complexity over $|\mathcal{L}|$ empirically. Note that we use $|\mathcal{L}| = 3$ in all of our main experiments, including a classification task.

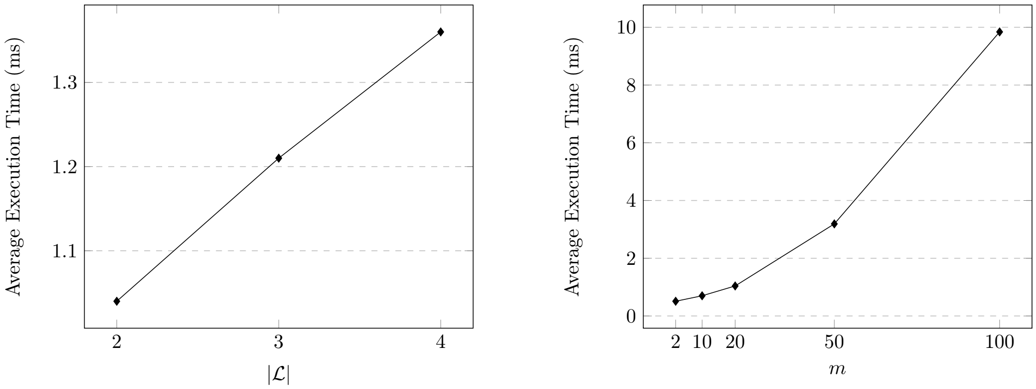

Figure 2: Mean execution time (ms) of Algorithm 1 per each batch of data over 100 trials. The left figure shows the time complexity of the algorithm over $|\mathcal{L}|$ and the right figure shows the time complexity over the number of inputs $m$. Note that the other parameters are fixed equal to the classification experiments setting, $m = m' = 20$, $n = 16$, and $|\mathcal{L}| = 3$.

# 4 ALGORITHM ANALYSIS

In this section, we perform comparison experiments to analyze the proposed Algorithm 1. First, we compare our algorithm with the exact brute force search algorithm to inspect the optimality of the algorithm. Next, we compare our algorithm with the BP algorithm proposed by Narasimhan and Bilmes (2005).

## 4.1 COMPARISON WITH BRUTE FORCE

To inspect the optimality of the proposed algorithm, we compare the function values of the solutions of Algorithm 1, brute force search algorithm, and random guess. Due to the exponential time complexity of the brute force search, we compare the algorithms on small scale experiment settings. Specifically, we test algorithms on settings of ($m = m' = 2,\ n = 4$), ($m = m' = 2,\ n = 9$), and ($m = m' = 3,\ n = 4$) varying the number of inputs and outputs ($m,\ m'$) and the dimension of data $n$. We generate unary cost matrix in the objective $f$ by sampling data from uniform distribution.

We perform experiments with 100 different random seeds and summarize the results on Table 2. From the table, we find that the proposed algorithm achieves near optimal solutions over various settings. We also measure relative errors between ours and random guess, $(f(z_{\text{ours}}) - f(z_{\text{brute}}))/(f(z_{\text{random}}) - f(z_{\text{brute}}))$. As a result, our algorithm achieves relative error less than 0.01.

| Configuration | Ours | Brute force (optimal) | Random guess | Rel. error |
|---|---|---|---|---|
| $(m = m' = 2,\ n = 4)$ | 1.91 | 1.90 | 3.54 | 0.004 |
| $(m = m' = 2,\ n = 9)$ | 1.93 | 1.91 | 3.66 | 0.01 |
| $(m = m' = 3,\ n = 4)$ | 2.89 | 2.85 | 22.02 | 0.002 |

Table 2: Mean function values of the solutions over 100 different random seeds. Rel. error means the relative error between ours and random guess.

### 4.2 Comparison with another BP algorithm

We compare the proposed algorithm and the BP algorithm proposed by Narasimhan and Bilmes (2005). We evaluate function values of solutions by each method using a random unary cost matrix from a uniform distribution. We compare methods over various scales by controlling the number of mixing inputs $m$.

Table 3 shows the averaged function values with standard deviations in the parenthesis. As we can see from the table, the proposed algorithm achieves much lower function values and deviations than the method by Narasimhan and Bilmes (2005) over various settings. Note that the method by Narasimhan and Bilmes (2005) has high variance due to randomization in the algorithm. We further compare the algorithm convergence time in Table 4. The experiments verify that the proposed algorithm is much faster and effective than the method by Narasimhan and Bilmes (2005).

| Algorithm | $m = 5$ | $m = 10$ | $m = 20$ | $m = 50$ | $m = 100$ |
|---|---|---|---|---|---|
| Ours | 3.1 (1.7) | 15 (6.6) | 54 (15) | 205 (26) | 469 (31) |
| Narasimhan | 269 (58) | 1071 (174) | 4344 (701) | 24955 (4439) | 85782 (14337) |
| Random | 809 (22) | 7269 (92) | 60964 (413) | 980973 (2462) | 7925650 (10381) |

Table 3: Mean function values of the solutions over 100 different random seeds. We report the standard deviations in the parenthesis. Random represents the random guess algorithm.

| Algorithm | $m = 5$ | $m = 10$ | $m = 20$ | $m = 50$ | $m = 100$ |
|---|---|---|---|---|---|
| Ours | 0.02 | 0.04 | 0.11 | 0.54 | 2.71 |
| Narasimhan | 0.06 | 0.09 | 0.27 | 1.27 | 7.08 |

Table 4: Convergence time (s) of the algorithms.

## 5 Hyperparameter settings

We perform Co-Mixup after down-sampling the given inputs and saliency maps to the pre-defined resolutions regardless of the size of the input data. In addition, we normalize the saliency of each input to sum up to 1 and define unary cost using the normalized saliency. As a result, we use an identical hyperparameter setting for various datasets; CIFAR-100, Tiny-ImageNet, and ImageNet. In details, we use $(\beta, \gamma, \eta, \tau) = (0.32, 1.0, 0.05, 0.83)$ for all of experiments. Note that $\tau$ is normalized according to the size of inputs $(n)$ and the ratio of the number of inputs and outputs $(m/m')$, and we use an isotropic Dirichlet distribution with $\alpha = 2$ for prior $p$. For a compatibility matrix, we use $\omega = 0.001$.

For baselines, we tune the mixing ratio hyperparameter, *i.e.,* the beta distribution parameter (Zhang et al., 2018), among $\{0.2, 1.0, 2.0\}$ for all of the experiments if the specific setting is not provided in the original papers.

## 6 ADDITIONAL EXPERIMENTAL RESULTS

### 6.1 ANOTHER DOMAIN: SPEECH

In addition to the image domain, we conduct experiments on the speech domain, verifying Co-Mixup works on various domains. Following (Zhang et al., 2018), we train LeNet (LeCun et al., 1998) and VGG-11 (Simonyan and Zisserman, 2014) on the Google commands dataset (Warden, 2017). The dataset consists of 65,000 utterances, and each utterance is about one-second-long belonging to one out of 30 classes. We train each classifier for 30 epochs with the same training setting and data pre-processing of Zhang et al. (2018). In more detail, we use $160 \times 100$ normalized spectrograms of utterances for training. As shown in Table 5, we verify that Co-Mixup is still effective in the speech domain.

| Model | Vanilla | Input | Manifold | CutMix | Puzzle Mix | Co-Mixup |
|---|---|---|---|---|---|---|
| LeNet | 11.24 | 10.83 | 12.33 | 12.80 | 10.89 | **10.67** |
| VGG-11 | 4.84 | 3.91 | 3.67 | 3.76 | 3.70 | **3.57** |

Table 5: Top-1 classification test error on the Google commands dataset. We stop training if validation accuracy does not increase for 5 consecutive epochs.

### 6.2 CALIBRATION

In this section, we summarize the expected calibration error (ECE) (Guo et al., 2017) of classifiers trained with various mixup methods. For evaluation, we use the official code provided by the TensorFlow-Probability library[2] and set the number of bins as 10. As shown in Table 6, Co-Mixup classifiers have the lowest calibration error on CIFAR-100 and Tiny-ImageNet. As pointed by Guo et al. (2017), the Vanilla networks have over-confident predictions, but however, we find that mixup classifiers tend to have under-confident predictions (Figure 3; Figure 4). As shown in the figures, Co-Mixup successfully alleviates the over-confidence issue and does not suffer from under-confidence predictions.

| Dataset | Vanilla | Input | Manifold | CutMix | Puzzle Mix | Co-Mixup |
|---|---|---|---|---|---|---|
| CIFAR-100 | 3.9 | 17.7 | 13.1 | 5.6 | 7.5 | **1.9** |
| Tiny-ImageNet | 4.5 | 6.2 | 6.8 | 12.0 | 5.6 | **2.5** |
| ImageNet | 5.9 | **1.2** | 1.7 | 4.3 | 2.1 | 2.1 |

Table 6: Expected calibration error (%) of classifiers trained with various mixup methods on CIFAR-100, Tiny-ImageNet and ImageNet. Note that, at all of three datasets, Co-Mixup outperforms all of the baselines in Top-1 accuracy.

### 6.3 SENSITIVITY ANALYSIS

We measure the Top-1 error rate of the model by sweeping the hyperparameter to show the sensitivity using PreActResNet18 on CIFAR-100 dataset. We sweep the label smoothness coefficient $\beta \in \{0, 0.16, 0.32, 0.48, 0.64\}$, compatibility coefficient $\gamma \in \{0.6, 0.8, 1.0, 1.2, 1.4\}$, clipping level $\tau \in \{0.79, 0.81, 0.83, 0.85, 0.87\}$, compatibility matrix parameter $\omega \in \{0, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}\}$, and the size of partition $m \in \{2, 4, 10, 20, 50\}$. Table 7 shows that Co-Mixup outperforms the best baseline (PuzzleMix, 20.62%) with a large pool of hyperparameters. We also find that Top-1 error rate increases as the partition batch size $m$ increases until $m = 20$.

---

[2] https://www.tensorflow.org/probability/api_docs/python/tfp/stats/expected_calibration_error

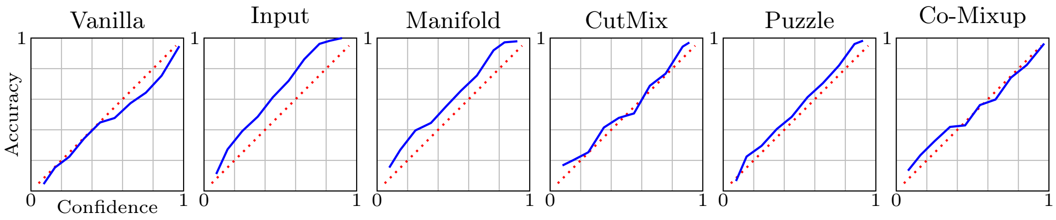

Figure 3: Confidence-Accuracy plots for classifiers on CIFAR-100. Note, ECE is calculated by the mean absolute difference between the two values.

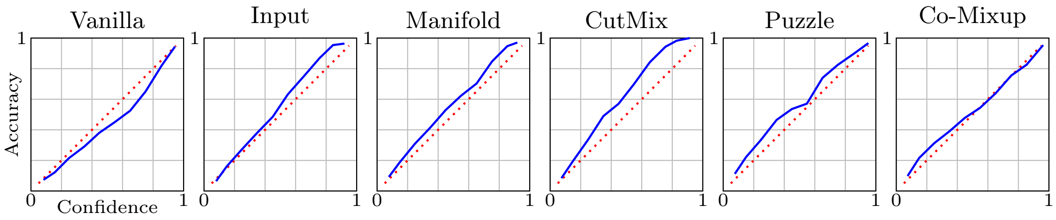

Figure 4: Confidence-Accuracy plots for classifiers on Tiny-ImageNet.

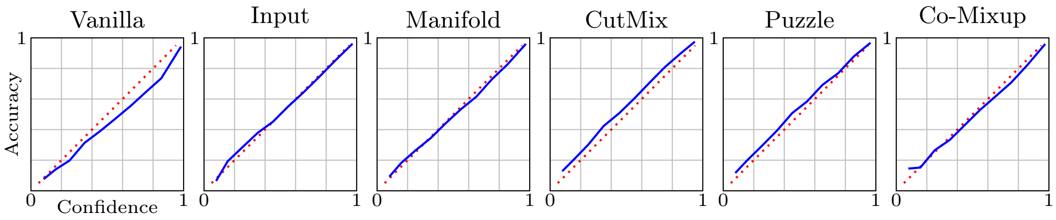

Figure 5: Confidence-Accuracy plots for classifiers on ImageNet.

| | | | | | |
|---|---|---|---|---|---|
| Smoothness coefficient, $\beta$ | $\beta = 0$<br>20.29 | $\beta = 0.16$<br>20.18 | $\beta = 0.32$<br>**19.87** | $\beta = 0.48$<br>20.35 | $\beta = 0.64$<br>21.24 |
| Compatibility coefficient, $\gamma$ | $\gamma = 0.6$<br>20.3 | $\gamma = 0.8$<br>19.99 | $\gamma = 1.0$<br>**19.87** | $\gamma = 1.2$<br>20.09 | $\gamma = 1.4$<br>20.13 |
| Clipping parameter, $\tau$ | $\tau = 0.79$<br>20.45 | $\tau = 0.81$<br>20.14 | $\tau = 0.83$<br>**19.87** | $\tau = 0.85$<br>20.15 | $\tau = 0.87$<br>20.23 |
| Compatibility matrix parameter, $\omega$ | $\omega = 0$<br>20.51 | $\omega = 5 \cdot 10^{-4}$<br>20.42 | $\omega = 10^{-3}$<br>**19.87** | $\omega = 5 \cdot 10^{-3}$<br>20.18 | $\omega = 10^{-2}$<br>20.14 |
| Partition size, $m$ | $m = 2$<br>20.3 | $m = 4$<br>20.22 | $m = 10$<br>20.15 | $m = 20$<br>**19.87** | $m = 50$<br>19.96 |

Table 7: Hyperparameter sensitivity results (Top-1 error rates) on CIFAR-100 with PreActResNet18. We report the mean values of three different random seeds.

### 6.4 COMPARISON WITH NON-MIXUP BASELINES

We compare the generalization performance of Co-Mixup with non-mixup baselines, verifying the proposed method achieves the state of the art generalization performance not only for the mixup-based methods but for other general regularization based methods. One of the regularization methods called VAT (Miyato et al., 2018) uses virtual adversarial loss, which is defined as the KL-divergence of predictions between input data against local perturbation. We perform the experiment with VAT regularization on CIFAR-100 with PreActResNet18 for 300 epochs in the supervised setting. We tune $\alpha$ (coefficient of VAT regularization term) in {0.001, 0.01, 0.1}, $\epsilon$ (radius of $\ell$-inf ball) in {1, 2}, and the number of noise update steps in

{0, 1}. Table 8 shows that Co-Mixup, which achieves Top-1 error rate of 19.87%, outperforms the VAT regularization method.

| VAT loss coefficient | # update=0 $\epsilon = 1$ | # update=0 $\epsilon = 2$ | # update=1 $\epsilon = 1$ | # update=1 $\epsilon = 2$ |
|---|---|---|---|---|
| $\alpha = 0.001$ | 23.38 | 23.62 | 24.76 | 26.22 |
| $\alpha = 0.01$ | 23.14 | 23.67 | 28.33 | 31.95 |
| $\alpha = 0.1$ | 23.65 | 23.88 | 34.75 | 39.82 |

Table 8: Top-1 error rates on CIFAR-100 dataset with PreActResNet18. We report the mean values of three different random seeds.

### 6.5 SIGNIFICANCE TEST

We run experiments with ten different random seeds to compare the Top-1 error rates between Puzzle Mix and Co-Mixup (CIFAR-100, PreActResNet18, 300 epochs). Based on the results, we perform a paired t-test, and obtain T-statistics: 2.28 and P-value: 0.02. This result demonstrates that Co-Mixup outperforms Puzzle Mix with statistical significance.

## 7 DETAILED DESCRIPTION FOR BACKGROUND CORRUPTION

We build the background corrupted test datasets based on ImageNet validation dataset to compare the robustness of the pre-trained classifiers against the background corruption. ImageNet consists of images $\{x_1, ..., x_M\}$, labels $\{y_1, ..., y_M\}$, and the corresponding ground-truth bounding boxes $\{b_1, ..., b_M\}$. We use the ground-truth bounding boxes to separate the foreground from the background. Let $z_j$ be a binary mask of image $x_j$, which has value 1 inside of the ground-truth bounding box $b_j$. Then, we generate two types of background corrupted sample $\tilde{x}_j$ by considering the following operations:

1. Replacement with another image as $\tilde{x}_j = x_j \odot z_j + x_{i(j)} \odot (1 - z_j)$ for a random permutation $\{i(1), ..., i(M)\}$.
2. Adding Gaussian noise as $\tilde{x}_j = x_j \odot z_j + \epsilon \odot (1 - z_j)$, where $\epsilon \sim N(0, 0.1^2)$. We clip pixel values of $\tilde{x}_j$ to [0, 1].

Figure 6 visualizes subsets of the background corruption test datasets. We will release the datasets upon acceptance.

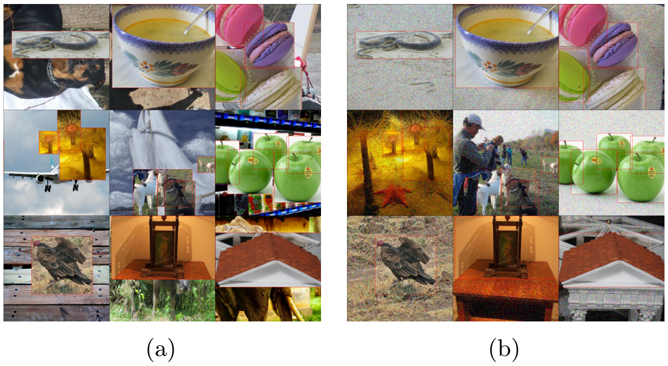
(a) (b)

Figure 6: Each subfigure shows background corrupted samples used in the robustness experiment. We consider three types of corruption: (a) Replacement with another image in ImageNet. (b) Adding Gaussian noise. The red boxes on the images represent ground-truth bounding boxes.

## 8 Co-Mixup generated samples

In Figure 8, we present Co-Mixup generated image samples by using images from ImageNet. We use an input batch consisting of 24 images, which is visualized in Figure 7. As can be seen from Figure 8, Co-Mixup efficiently mix-matches salient regions of the given inputs maximizing saliency and creates diverse outputs. In Figure 8, inputs with the target objects on the left side are mixed with the objects on the right side, and objects on the top side are mixed with the objects on the bottom side. In Figure 9, we present Co-Mixup generated image samples with larger $\tau$ using the same input batch. By increasing $\tau$, we can encourage Co-Mixup to use more inputs to mix per each output.

## References

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. *In ICML*, 2017.

T. Horel and Y. Singer. Maximization of approximately submodular functions. *In NeurIPS*, 2016.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

M. Narasimhan and J. A. Bilmes. A submodular-supermodular procedure with applications to discriminative structure learning. *UAI*, 2005.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

P. Warden. *URL https://research.googleblog.com/2017/08/launching-speech-commands-dataset.html.*, 2017.

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *In ICLR*, 2018.

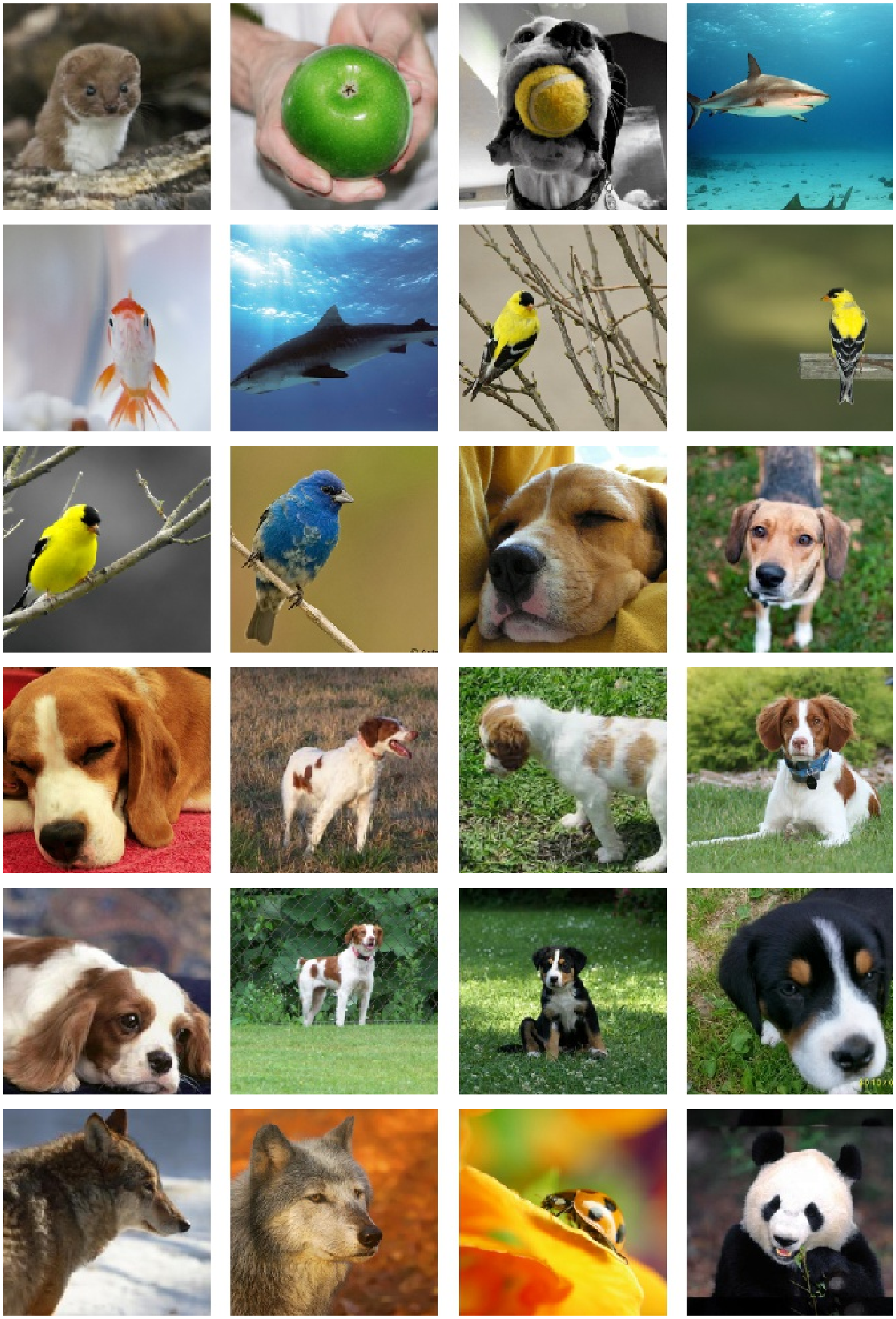

Figure 7: Input batch.

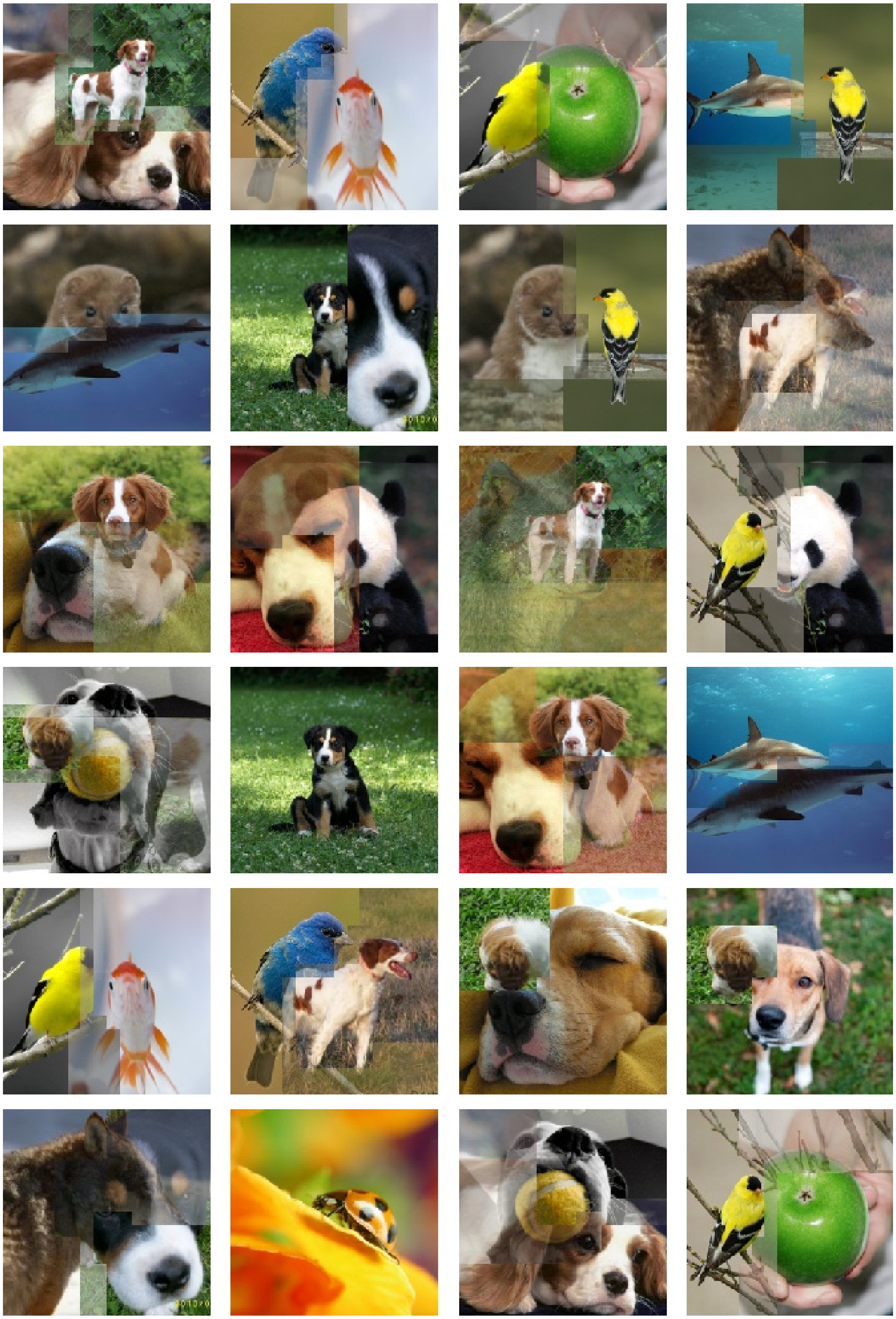

Figure 8: Mixed output batch.

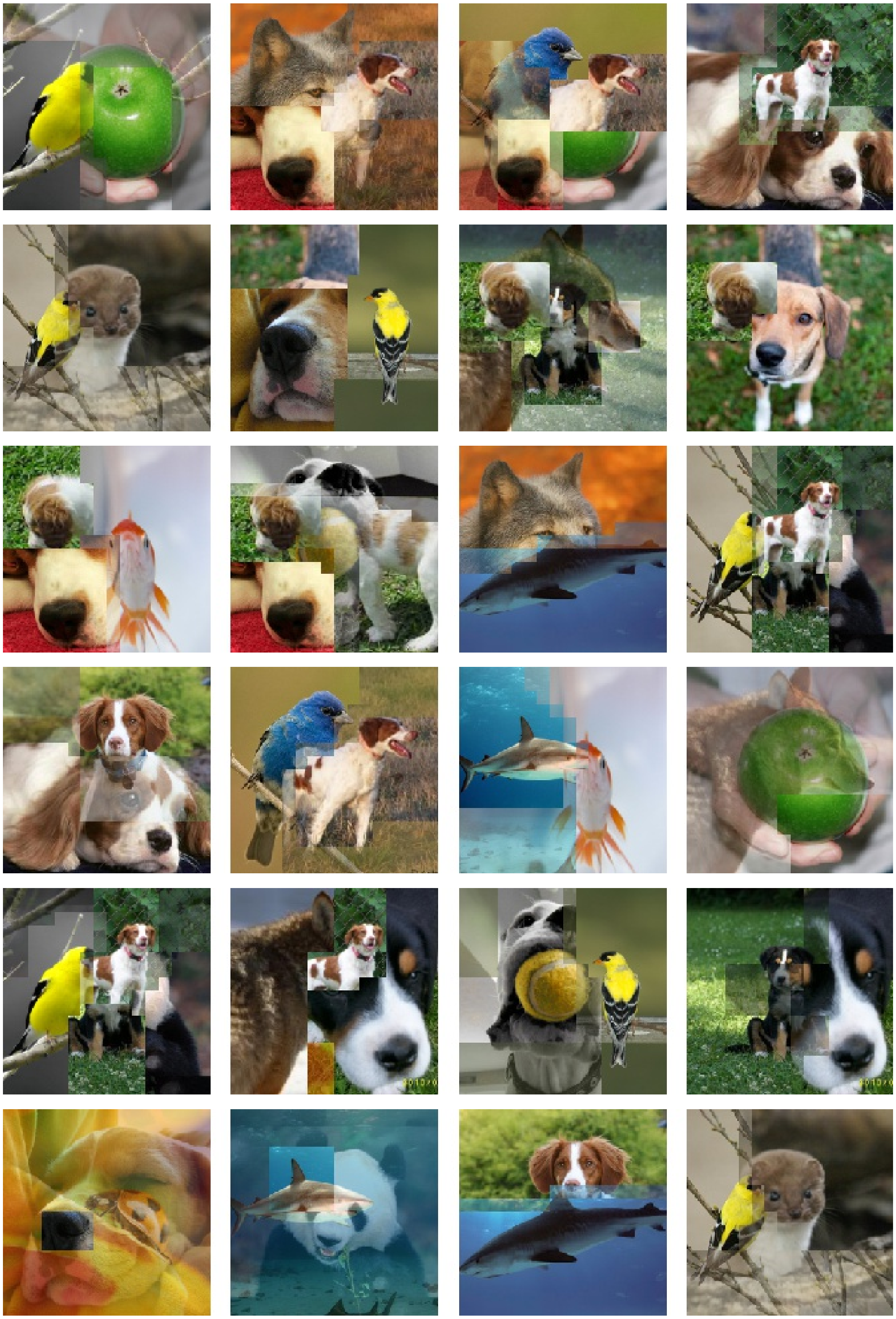

Figure 9: Another mixed output batch with larger $\tau$.