Puzzle Mix: Exploiting Saliency and Local Statistics for Optimal Mixup

Jang-Hyun Kim 1 2  Wonho Choo 1 2  Hyun Oh Song 1 2

Abstract
While deep neural networks achieve great performance on fitting the training distribution, the learned networks are prone to overfitting and are susceptible to adversarial attacks. In this regard, a number of mixup based augmentation methods have been recently proposed. However, these approaches mainly focus on creating previously unseen virtual examples and can sometimes provide misleading supervisory signal to the network. To this end, we propose Puzzle Mix, a mixup method for explicitly utilizing the saliency information and the underlying statistics of the natural examples. This leads to an interesting optimization problem alternating between the multi-label objective for optimal mixing mask and saliency discounted optimal transport objective. Our experiments show Puzzle Mix achieves the state of the art generalization and the adversarial robustness results compared to other mixup methods on CIFAR-100, Tiny-ImageNet, and ImageNet datasets, and the source code is available at https://github.com/snu-milab/PuzzleMix.

1. Introduction
Deep neural network models are the bedrock of modern AI tasks such as object recognition, speech, natural language processing, and reinforcement learning. However, these models are known to memorize the training data and make overconfident predictions often resulting in degraded generalization performance on test examples (Srivastava et al., 2014; Zhang et al., 2016). Furthermore, the problem is exacerbated when the models are evaluated on examples under slight distribution shift (Ben-David et al., 2010).

To this end, data augmentation approaches aim to alleviate some of these issues by improving the model generalization performance (He et al., 2015; DeVries & Taylor, 2017). Recently, a line of research called mixup has been proposed. These methods mainly focus on creating previously unseen virtual mixup examples via convex combination or local replacement of data for training (Zhang et al., 2018; Verma et al., 2019; Yun et al., 2019; Guo et al., 2019).

However, the underlying data domains contain rich regional saliency information (i.e., foreground objects in vision, prominent syllables in speech, informative textual units in language) (Simonyan et al., 2013; Kalinli & Narayanan, 2007; Erkan & Radev, 2004) and exhibit local regularity structure far from random matrices of numbers (Huang & Mumford, 1999; Zhang et al., 2017; Smith, 2003). Thus, completely disregarding these aspects of data could lead to creating mixup examples which could misguide the training model and undermine the generalization performance.

Motivated by this intuition, we propose Puzzle Mix, a mixup method for explicitly leveraging the saliency information and the underlying local statistics of natural examples. Our proposed method jointly seek to find (1) the optimal mask for deciding how much of the two inputs to reveal versus conceal in the given region and for (2) the transport for finding the optimal moves in order to maximize the exposed saliency under the mask. The optimization process is remi-
Puzzle Mix: Exploiting Saliency and Local Statistics for Optimal Mixup

We introduce Puzzle Mix, a novel data augmentation method that leverages saliency and local statistics to improve the generalization performance and robustness of neural networks. Unlike traditional mixup methods that simply linearly interpolate two input data, Puzzle Mix alternates between finding the optimal mask and optimizing the transport plans, which enables a more precise saliency resolution. Our method is complementary to these techniques and could be used in conjunction in order to further increase the generalization and robustness performance.

Saliency  
Simonyan et al. (2013) detects object saliency by computing gradients of a pre-trained deep neural network. Subsequently, other methods were introduced to obtain more precise saliency (Zhao et al., 2015; Wang et al., 2015). However, these methods require modifying the pre-trained network or training new models to compute the saliency. Zhou et al. (2016) and Selvaraju et al. (2017) proposed methods with the reduced computation cost but at the cost of saliency resolution. We follow the method from Simonyan et al. (2013), which does not require any modification to the model, to compute the saliency map.

Optimal transport  
A transport plan is a distribution over two measures, with the goal of minimizing the cost of moving one measure to another. In mixup, the cost function is the Euclidean distance. Several optimal transport algorithms have been developed, including entropic regularization (CutMix) and training with a Wasserstein distance (MixMatch).

Optimal transport  
A transport plan that moves a given distribution to another at the minimal cost is called the optimal transport (Villani, 2008). The optimal transport problem is widely applied in various application areas such as color transfer (Rabin et al., 2014) or domain adaptation (Courty et al., 2016). We formulate a binary transport problem for the optimal move, which maximizes the exposed saliency under the mask.

### 2. Related Works

**Data augmentation**  
Methods that implement data augmentation aim to regularize the models from overfitting to the training distribution and improve the generalization performance by generating virtual training examples in the vicinity of the given training dataset (Bishop, 2006). Some of the most commonly used data augmentation techniques are random cropping, flipping, and adding random noise (Bishop, 1995). Recently, a data augmentation method called AugMix is proposed to improve both the generalization performance and the corruption robustness (Hendrycks et al., 2020). Our method is complementary to these techniques and could be used in conjunction in order to further increase the generalization and robustness performance.

**Mixup**  
Input mixup creates virtual training examples by linearly interpolating two input data and corresponding one-hot labels (Zhang et al., 2018). The method induces models to have smoother decision boundaries and reduces overfitting to the training data. Manifold mixup extends this concept from input space to feature space (Verma et al., 2019). Also, Guo et al. (2019) proposed an adaptive mixup method, which improves Input mixup by preventing the generation of improper mixup data. Yun et al. (2019) proposed CutMix which implants a random rectangular region of the input into another. However, these methods can generate improper examples by randomly removing important regions of the data, which may mislead the neural network (see Figure 1). Our mixup method aims to prevent these issues by utilizing the saliency signal while preserving the local properties of the input data.
Manifold input mixup

Figure 2. (a) Mixed saliency \( ||h(s(x_0), s(x_1))||_1 \). Note the saliency map of each input \( s(x_k) \) is normalized to sum up to 1. (b) Total variation of mixed data. (c) Cross entropy loss of mixup data and the corresponding soft-label evaluated by the vanilla classifier (ResNet18). (d) Top-1 prediction accuracy of mixed data. Prediction is counted as correct if the Top-1 prediction belongs to the correct class. (e) Top-2 prediction accuracy of mixed data. Prediction is counted as correct if the Top-2 predictions are equal to the correct class. Manifold mixup is omitted in (a) and (b) as manifold mixup generates mixup examples in the hidden space not in the input space.

<table>
<thead>
<tr>
<th>Method</th>
<th>Mixup function ( h(x_0, x_1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input mixup</td>
<td>((1 - \lambda)x_0 + \lambda x_1)</td>
</tr>
<tr>
<td>Manifold mixup</td>
<td>((1 - \lambda)f(x_0) + \lambda f(x_1))</td>
</tr>
<tr>
<td>CutMix</td>
<td>((1 - \mathcal{I}_B) \circ x_0 + \mathcal{I}_B \odot x_1)</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>((1 - z) \odot \Pi_0^n s(x_0) + z \odot \Pi_1^n s(x_1))</td>
</tr>
</tbody>
</table>

Table 1. Summary of various mixup functions.

4. Methods

Our goal is to maximally utilize the saliency information of each input while respecting the underlying local statistics of the data. First, in order to maximally utilize the saliency information, we seek to find the optimal mixing mask \( z \) and the optimal transport plans \( \Pi \) under the following criteria.

- Given a pair of transported data and a specific region, the mask \( z \) should optimally reveal more salient data of the two while masking the less salient one in the given region.
- Given a data \( x \) and the mask \( z \), the transport \( \Pi \) should find the optimal moves that would maximize the saliency of the revealed portion of the data.

The criteria above motivates us to maximize for \( (1 - z) \odot \Pi_0^n s(x_0) + z \odot \Pi_1^n s(x_1) \). Note, we denote the saliency of the input \( x \) as \( s(x) \) which is computed by taking \( \ell_2 \) norm of the gradient values across input channels. Figure 2 (a) shows the proposed mixup function well preserves the saliency information after mixup. Second, in order to respect the underlying local statistics of the data (Huang & Mumford, 1999; Zhang et al., 2017; Smith, 2003), we consider the following criteria.

- The saliency information can be noisy, which could lead to a suboptimal solution. Therefore, we add spatial regularization terms \( \psi \) and \( \phi_{i,j} \) to control the smoothness of the mask and regional smoothness of the resulting mixed example. Figure 2 (b) compares the local smoothness measured in total variation.
- We ensure the structural integrity within each data is generally preserved by considering the transport cost \( C_{ij} \) defined as the distance between the locations \( i \) and \( j \). Also, to further ensure the local salient structure of the data is preserved without being dispersed across after the transport, we optimize for the binary transport plans as opposed to continuous plans.

Evaluation results on the pretrained vanilla classifier in Figure 2 (c), (d), (e) show our mixup examples have the smallest loss and the highest accuracy compared to other methods, verifying our intuitions above. Moreover, we optimize the main objective after down-sampling the saliency information \( s(x) \) with average pooling to support multi-scale transport and masking. From now on, we denote \( n \) as the down-sampled dimension. In practice, we select the down-sampling resolution randomly per each mini-batch.

To optimize the mask \( z \), we first discretize the range of the mask value. Let \( \mathcal{L} \) denote the discretized range \( \{ \frac{1}{m} \mid t = 0, 1, \ldots, m \} \). In addition, to control the mixing ratio of given inputs, we add a prior term \( p(z_i) \), which follows a binomial distribution. We now formalize the complete objective in Equation (3).

\[
\begin{align*}
\min_{z \in \mathcal{L}^n} & - ||(1 - z) \odot \Pi_0^n s(x_0)||_1 \\
& - ||z \odot \Pi_1^n s(x_1)||_1 \\
& + \beta \sum_{(i,j) \in \mathcal{N}} \psi(z_i, z_j) + \gamma \sum_{(i,j) \in \mathcal{N}} \phi_{i,j}(z_i, z_j) \\
& - \eta \sum_{z_i} \log p(z_i) + \xi \sum_{k=0,1} \langle \Pi_k, C \rangle \\
\text{subject to } & \Pi_k 1_n = 1_n, \Pi_k^2 1_n = 1_n \text{ for } k = 0, 1.
\end{align*}
\]
After solving the optimization problem in Equation (3), we obtain the mixed example \( h(x_0, x_1) = (1 - z^*) \odot \Pi_0^T x_0 + z^* \odot \Pi_1^T x_1 \) which is then used for the model training as in Equation (1). Figure 3 illustrates the mask \( z \) and the transport plan \( \Pi \) optimized with Equation (3).

We solve this optimization problem via alternating minimization through iterating first over \( z \) and then simultaneously over \( \Pi_0 \) and \( \Pi_1 \). In mixup augmentation, however, one needs to be able to efficiently generate the mixed examples as the generation process takes place per each mini-batch. Therefore, we optimize for one complete cycle of the alternating minimization, as repeated cycles require additional network evaluations, for efficiency. As for the initialization, we optimize the mask \( z \) with \( \Pi_k \) initialized as identity transport, and then optimize each \( \Pi_k \) with the previously solved \( z \). We now formally discuss individual optimization problems in Section 4.1 and Section 4.2.

4.1. Optimizing Mask

Given \( \Pi_0 \) and \( \Pi_1 \), we seek to solve the following discrete optimization problem over \( z \) in Equation (4). The objective is to decide how to best mix the two transported inputs jointly based on the region saliency measure (unary), the label and data local smoothness (pairwise), and the mixing weight log prior (mix prior) criteria.

\[
\text{minimize}_{z \in \mathcal{L}^n} \sum_{i=1}^{N} u_i(z_i) + \beta \sum_{(i,j) \in \mathcal{N}} \psi(z_i, z_j) + \gamma \sum_{(i,j) \in \mathcal{N}} \phi_{i,j}(z_i, z_j) - \eta \sum_{i} \log p(z_i),
\]

where the unary term \( u_i(z_i) \) is defined as \( z_i(\Pi_0^T s(x_0))_i + (1 - z_i)(\Pi_1^T s(x_1))_i \). We define the neighborhood \( \mathcal{N} \) as a set of adjacent regions, and use the following pairwise terms and the prior term. Figure 4 visualizes different components in Equation (4).

Definition 1. (Label smoothness) \( \psi(z_i, z_j) := (z_i - z_j)^2 \).

For data local smoothness, we measure the distance between input regions. Let \( d_p \) denote the distance function. First, we define pairwise terms under the binary case, \( \mathcal{L} = \{0, 1\} \), and then extend them to the multi-label case.

Definition 2. (Data local smoothness for binary labels)
Let \( x_{k,i} \) represent the \( i^{th} \) region of data \( x_k \). Then, \( \phi_{i,j}^k(z_i, z_j) := d_p(x_{z_i,i}, x_{z_j,j}) \).

The discrete optimization problem in Equation (4) is a type of multi-label energy minimization problem and can be efficiently solved via \( \alpha-\beta \) swap algorithm (Boykov et al., 2001), which is based on the graph-cuts. In the binary label case, finding the minimum s-t cut in the graph returns an equivalent optimal solution if the pairwise term satisfies the submodularity condition (Kolmogorov & Zabih, 2004).

In our problem, the pairwise term is \( e_{i,j}(z_i, z_j) = \beta \psi(z_i, z_j) + \gamma \phi_{i,j}(z_i, z_j) \). We now assume that the function values of \( d_p \) are bounded in \([0, 1]\), which is generally satisfied when data values are bounded in \([0, 1]\).

Proposition 1. Suppose \( d_p \) function is bounded in \([0, 1]\) and \( \phi = \phi^h \). If \( \gamma \leq \beta \), then \( e_{i,j}(z_i, z_j) \) satisfies submodularity for \( z_i, z_j \in \{0, 1\} \).

\[
\text{Proof.} \quad e(0, 0) + e(1, 1) = \gamma \phi_{i,j}(0, 0) + \gamma \phi_{i,j}(1, 1) \leq 2\gamma \leq 2\beta = \beta \psi(0, 1) + \beta \psi(1, 0) \leq e(0, 1) + e(1, 0). \]

For multi-label case, the \( \alpha-\beta \) swap algorithm iteratively applies graph-cut as a sub-routine and converges to a...
local-minimum if the pairwise term satisfies pairwise-submodularity (Schmidt & Alahari, 2011). We can guarantee pairwise-submodularity by slightly modifying \( \phi_{i,j} \) as

\[
\begin{align*}
\phi_{i,j}'(0, 0) &= \phi_{i,j}(0, 0) + (\phi_{i,j}'(0, 1) + \phi_{i,j}(1, 0))/2 \\
\phi_{i,j}'(0, 1) &= \phi_{i,j}(0, 1) + (\phi_{i,j}'(0, 0) + \phi_{i,j}(1, 1))/2 \\
\phi_{i,j}'(1, 0) &= \phi_{i,j}(1, 0) + (\phi_{i,j}'(0, 0) + \phi_{i,j}(1, 1))/2 \\
\phi_{i,j}'(1, 1) &= \phi_{i,j}(1, 1) + (\phi_{i,j}'(0, 1) + \phi_{i,j}(1, 0))/2.
\end{align*}
\]

It is important to note that, \( \phi_{i,j}'(1, 0) + \phi_{i,j}'(0, 1) - \phi_{i,j}'(0, 0) - \phi_{i,j}'(1, 1) = 0 \).

**Definition 3.** (Data local smoothness for the multi labels)

\[
\phi_{i,j}(z_i, z_j) = \sum_{k=1}^n \phi_{i,j}'(z_i') + \phi_{i,j}'(z_j') + (1 - z_i)z_j \phi_{i,j}'(1, 1) + (1 - z_j)z_i \phi_{i,j}'(0, 0), \quad \forall z_i, z_j \in \mathcal{L}.
\]

**Proposition 2.** With \( \phi_{i,j} \) defined as Definition 3, \( e_{i,j} \) satisfies pairwise submodularity.

**Proof.** We can represent \( \phi_{i,j} \) as follows:

\[
\begin{align*}
\phi_{i,j}(z_i, z_j) &= f(z_i, z_j) \phi_{i,j}'(1, 0) + \phi_{i,j}'(0, 1) - \phi_{i,j}'(0, 0) - \phi_{i,j}'(1, 1) \\
&= z_i \phi_{i,j}'(1, 0) + \phi_{i,j}'(1, 1) - \phi_{i,j}'(0, 0) - \phi_{i,j}'(1, 1) \\
&= z_j \phi_{i,j}'(0, 1) + \phi_{i,j}'(1, 1) - \phi_{i,j}'(0, 0) - \phi_{i,j}'(1, 1) \\
&= \phi_{i,j}(0, 0),
\end{align*}
\]

where \( f(z_i, z_j) = (1 - z_i)z_j + z_i(1 - z_j) \). By definition \( \phi_{i,j}'(0, 0) + \phi_{i,j}'(0, 1) - \phi_{i,j}'(0, 0) - \phi_{i,j}'(1, 1) = 0 \), and thus, \( \phi_{i,j}(z_i, z_j) \) can be represented as the form of \( z_i \phi_{i,j}' + z_j \phi_{i,j}' + c \). Thus, \( \phi_{i,j}' = x \phi_{i,j}' + y \phi_{i,j}' + c \).

By definition \( \psi \) satisfies pairwise submodularity, and by Lemma 1, \( e_{i,j} \) satisfies pairwise submodularity.

**Lemma 1.** If \( \psi, \phi \) satisfies pairwise submodularity and \( \gamma, \beta \in \mathbb{R}_+ \), then \( \beta \psi + \gamma \phi \) satisfies pairwise submodularity.

**Proof.** See Supplementary A.1.

Finally, we use the prior term to control the ratio of inputs in the mixed output. For the given mixing weight \( \lambda \), which represents the ratio of \( x_1 \) with respect to \( x_0 \), we define the prior term \( p \) to satisfy \( \mathbb{E}_{z_i \sim p}[z_i] = \lambda, \forall i \). Specifically, for the label space \( \mathcal{L} = \{ \frac{t}{m} | t = 0, \ldots, m \} \), we define the prior term as \( p(z_i = \frac{t}{m}) = \binom{m}{t} \lambda^t (1 - \lambda)^{m-t} \) for \( t = 0, 1, \ldots, m \). In other words, \( z_i \sim \frac{1}{m} B(m, \lambda) \).

In Figure 5, we provide the resulted mixup images using the optimal mask from Equation (4). Specifically, we visualize how the Puzzle Mix images change by increasing the mixing weight \( \lambda \) and the coefficients of the smoothness terms, \( \beta \) and \( \gamma \).

**4.2. Optimizing Transport**

After optimizing the mask \( z \), we optimize the transportation plans for the input data under the optimal mask \( z^* \). Our objective with respect to transportation plans is the following.

\[
\begin{align*}
\min_{\Pi_0,\Pi_1 \in \{0,1\}^n \times n} & \quad - \| (1 - z^*) \odot \Pi_0 s(x_0) \|_1 \\
& - \| z^* \odot \Pi_1 s(x_1) \|_1 \\
& + \xi \sum_{k=0}^n (\Pi_k, C) \\
\text{subject to} \quad & \Pi_0 1_n = 1_n, \quad \Pi_k 1_n = 1_n \quad \text{for } k = 0, 1.
\end{align*}
\]

Note the problem is completely separable as two independent optimization problems of each \( \Pi_k \). Let \( s(x_1)_i \) be the probability of assigning the label \( i \) to the mixed output image. The objective function can be expressed as:

\[
\begin{align*}
\min_{\Pi_0,\Pi_1 \in \{0,1\}^n \times n} & \quad - \sum_{i=1}^n \sum_{j=1}^n \sum_{k=0}^n \Pi_k i \Pi_j j \Pi_0 i \Pi_1 j s(x_1)_i s(x_0)_j \\
& - \sum_{i=1}^n \sum_{k=0}^n \Pi_k i \Pi_0 i s(x_0)_i \\
& + \xi \sum_{k=0}^n (\Pi_k, C)
\end{align*}
\]

The problem is solved using standard methods for solving linear programs.
Algorithm 1 Masked Transport

\textbf{Input:} mask \( z^* \), cost \( C' \), large value \( v \)

Initialize \( C^{(0)} = C' \), \( t = 0 \)

\textbf{repeat}

\begin{itemize}
  \item target = \text{argmin}(C^{(t)}, \text{dim} = 1)
  \item \( \Pi = 0_{n \times n} \)
  \item for \( i = 0 \) to \( n - 1 \) do
      \begin{itemize}
        \item \( \Pi[i, \text{target}[i]] = 1 \)
      \end{itemize}
\end{itemize}

\textbf{end for}

\( C_{\text{conflict}} = C^{(t)} \odot \Pi + v(1 - \Pi) \)

\( \text{source} = \text{argmin}(C_{\text{conflict}}, \text{dim} = 0) \)

\( \Pi_{\text{win}} = 0_{n \times n} \)

\textbf{for} \( j = 0 \) to \( n - 1 \) do

\begin{itemize}
  \item \( \Pi_{\text{win}}[\text{source}[j], j] = 1 \)
\end{itemize}

\textbf{end for}

\( \Pi_{\text{win}} = \Pi_{\text{win}} \odot \Pi \)

\( C^{(t+1)} = C^{(t)} + v\Pi_{\text{lose}} \)

\( t = t + 1 \)

\textbf{until convergence}

\textbf{Return:} \( \Pi_{\text{win}} \)

\[ \text{denote the } i^{th} \text{ entry of the } n \times n \text{ diagonal column vector } s(x_1). \text{ The term } \|z^* \odot \Pi \|_1 s(x_1)\|_1 \text{ can be represented as } \sum_{i,j} z^*_j s(x_1)_{i,j} = C_{ij}, j = 1, \ldots, n. \text{ Finally, the transport optimization problem of } \Pi \text{ becomes} \]

\[ \begin{aligned}
\min_{\Pi \in (0,1)^{n \times n}} & \quad \langle \Pi, C' \rangle \\
\text{subject to} & \quad \Pi_{11} = 1, \quad \Pi_{1n} = 1, \quad \Pi_{n1} = 1, \quad \Pi_{nn} = 1,
\end{aligned} \]

where \( C' = \Xi C - s(x_1)z^* \), \( C_{ij} \) is the cost of moving the \( i^{th} \) region to the \( j^{th} \) position, which consists of two components. The first component is the distance \( \Xi C_{ij}, \) which is defined as a distance from \( i \) to \( j \). The second component is the saliency term, which discounts the transport cost with the saliency value of the \( i^{th} \) region if the mask of \( j^{th} \) position is non-zero. Briefly speaking, the larger the saliency value, the more the discount in the transport cost.

The optimization problem in Equation (5) can be solved exactly by using the Hungarian algorithm and its variants with time complexity of \( O(n^3) \) (Munkres, 1957; Jonker & Volgenant, 1987). As we need to efficiently generate mixup examples per each mini-batch, this can be a computational bottleneck as \( n \) increases. Thus, we propose an approximate algorithm that can be parallelized on GPUs and efficiently computed in batches. The proposed algorithm can quickly decrease the objective \( \langle \Pi, C' \rangle \) and converges to a local-minimum within \( n(n - 1)/2 + 1 \) steps. Experimental results comparing the wall clock execution time and the relative error are in Supplementary B.

Algorithm 1 progressively alternates between row-wise and column-wise optimizations. The algorithm first minimizes \( \langle \Pi, C' \rangle \) only with the \( \Pi_{1n} = 1 \) constraint. However, since the optimization is done without the column constraint, there can be multiple 1 values in a column of \( \Pi \). In the following step, the column with multiple 1 values leaves only one 1 in the row with the smallest cost. We denote the result as \( \Pi_{\text{win}} \) in Algorithm 1. The corresponding cost entries for the rows that do not remain in \( \Pi_{\text{win}} \) are penalized with a large additive value, and the 1 values are moved to the other columns in the next iteration.

Our algorithm can also take advantage of intermediate \( \Pi_{\text{win}} \) as a solution, supported by the following two properties. We suppose that transport cost matrix \( C \) has zeros in diagonal entries and positive values in others. In addition, let \( \Pi^{(t)} \) and \( \Pi^{(t)}_{\text{win}} \) denote \( \Pi \) and \( \Pi_{\text{win}} \) at the end of \( t^{th} \) step in Algorithm 1.

\textbf{Proposition 3.} Suppose \( z^* \) has values in \( \{0, 1\} \). Then for \( j \) s.t. \( z^*_j = 1 \), \( j^{th} \) column of \( \Pi^{(t)}_{\text{win}} \) has exactly one 1.

\textbf{Proof.} By the definition of \( C^{(0)} = \Xi C - s(x_0)z^* \), for \( j \) s.t. \( z^*_j = 1 \), \( j^{th} \) row of \( C^{(0)} \) has a minimum at \( j^{th} \) entry. Thus, \( j^{th} \) column of \( \Pi^{(0)}_{\text{win}} \) has exactly one 1 and others are 0. Suppose that, the claim is satisfied for \( \Pi^{(t)}_{\text{win}} \) and \( \Pi^{(t)}_{\text{win}}[i(j), j] \) is 1 for \( j \) s.t. \( z^*_j = 1 \). Then, by the definition of \( \Pi^{(t)}_{\text{win}} \), \( \Pi^{(t)}_{\text{win}}[i(j), j] \) is the minimum of \( i(j)^{th} \) row of \( C^{(t)} \) and the row will not be updated in \( C^{(t+1)} \). Thus, \( i(j)^{th} \) row of \( C^{(t+1)} \) has a minimum at \( j^{th} \) entry and \( j^{th} \) column of \( \Pi^{(t+1)}_{\text{win}} \) has exactly one 1. By induction, the claim holds.

\textbf{Proposition 4.} Under the assumption of Proposition 3, the partial objective \( \langle \Pi^{(t)}_{\text{win}}, C'z^* \rangle \) decreases as \( t \) increases.

\textbf{Proof.} By Proposition 3, for \( j \) s.t. \( z^*_j = 1 \), \( j^{th} \) column of \( \Pi^{(t)}_{\text{win}} \) has exactly one 1. Let \( i(j; t) \) denote the corresponding row index with the entry 1. Then, it is enough to prove that \( C'[i(j; t + 1), j] \leq C'[i(j; t), j] \). However, in the last part of the proof of Proposition 3, we showed that \( i(j; t)^{th} \) row of \( C^{(t+1)} \) has a minimum at \( j^{th} \) entry, and thus \( \Pi^{(t+1)}[i(j; t), j] = 1 \). By Algorithm 1, index \( i(j; t + 1) \) satisfies \( C^{(t+1)}[i(j; t + 1), j] \leq C^{(t+1)}[i(j), j] \), \( \forall i \) s.t. \( \Pi^{(t+1)}[i, j] = 1 \). Thus, \( C^{(t+1)}[i(j; t + 1), j] \leq C^{(t+1)}[i(j; t), j] \). Finally, \( \Pi^{(t+1)}[i, j] = 1 \) means that cost from \( i \) to \( j \) is not updated, \( i.e., C^{(t+1)}[i, j] = C'[i, j] \).

Finally, we introduce the convergence property of Algorithm 1.

\textbf{Proposition 5.} Algorithm 1 converges to a local-minimum with respect to the update rate at most \( n(n - 1)/2 + 1 \) steps.

\textbf{Proof.} See Supplementary A.2.
As shown in Algorithm 2, we add the adversarial perturbation method of Wong et al. (2020), which adds size of the graph, \( L \), space of the neural network. In our experiments, we use label sampling. Note that, in our experiments, the computational quickly, since we restrict the size of the graph with down sampling. Although the minimization is performed \( \alpha \), example-wise in CPUs, the \( \beta \) python wrapper to the mask \( z \). First, to solve the discrete optimization problem with respect \( z \), an adversarial signal to the corresponding input and location to the proper location of the mixed output, \( \delta \). Since our mix-up strategy utilizes the gradients of the loss \( \frac{\partial \ell(x_i, y_i)}{\partial x_i} \) at each mini-batch.

All of the computations in our algorithm except \( \alpha \), \( \beta \) swap are done in mini-batch and can be performed in parallel in GPUs. Note that for-loops in Algorithm 1 can be done in parallel by using the scatter function of PyTorch (Paszke et al., 2017).

Since we calculate the saliency information by back-propagating the gradient of loss function through the model, we can utilize this gradient information without any computational overhead. We regularize the gradient of the model with mixup data as \( \nabla_{x_i} \ell(h(x_i, x_1), g(y_0, y_1); \theta) + \frac{1}{2} \lambda \nabla_{\theta} \ell(h(x_0, y_0; \theta) + \nabla_{\theta} \ell(x_1, y_1; \theta)) \). This additional regularization helps us to improve generalization performance on Tiny-ImageNet and ImageNet.

### 6. Experiments

We train and evaluate classifiers on CIFAR-100 (Krizhevsky & Geoffrey, 2009), Tiny-ImageNet (Chrabaszcz et al., 2017), and ImageNet (Deng et al., 2009) datasets. We first study the generalization performance and adversarial robustness of our method (Section 6.1). Next, we show that our method can be used in conjunction with the existing augmentation method (AugMix) to simultaneously improve the corruption robustness and generalization performance (Section 6.2). Finally, we perform ablation studies for our method (Section 6.3).

#### 6.1. Generalization Performance and Adversarial Robustness

##### 6.1.1. CIFAR-100

We train two residual neural networks (He et al., 2015): WRN28-10 (Zagoruyko & Komodakis, 2016) and PreActResNet18 (He et al., 2016). We follow the training protocol of Verma et al. (2019), which trains WRN28-10 for 400 epochs and PreActResNet18 for 1200 epochs. Hyperparameter settings are available in Supplementary C.1. We reproduce the mixup baselines (Zhang et al., 2018; Verma et al., 2019; Yun et al., 2019; Hendrycks et al., 2020) and compare the baselines with our method under the same experimental settings described above. We denote the experiments as Vanilla, Input, Manifold, CutMix, AugMix, Puzzle Mix in the experiment tables.

Note however, our mixup method requires an additional

---

**Algorithm 2 Stochastic Adversarial Puzzle Mix**

**Input:** data \( x_0, x_1 \), attack ball \( \epsilon \), step \( \tau \), probability \( p \)

\[ x_{i,\text{clean}} = x_i \] for \( i = 0, 1 \)

Sample \( \nu_i \sim B(1, p) \) for \( i = 0, 1 \)

for \( i = 1, 2 \) do

if \( \nu_i = 1 \) then

\[ \kappa_i \sim \text{Uniform}(-\epsilon, \epsilon) \]

\[ x_i \leftarrow x_i + \kappa_i \]

end if

end for

Calculate gradient \( \nabla_{x_i} l(x_i) \) for \( i = 0, 1 \)

Optimize \( z^* \) and \( \Pi^T \) in Equation (3)

Sample \( \delta \sim \text{Uniform}(0, 1) \)

for \( i = 0, 1 \) do

if \( \nu_i = 1 \) then

\[ \kappa_i \leftarrow \kappa_i + \tau \text{sign}(\nabla_{x} l(x_i)) \]

\[ \kappa_i \leftarrow \text{clip}(\kappa_i, -\epsilon, \epsilon) \]

\[ x_i \leftarrow x_{i,\text{clean}} + \delta \kappa_i \]

end if

end for

Return: \((1 - z^*) \odot \Pi^T_0 x_0 + z^* \odot \Pi^T_1 x_1\)

---

4.3. Adversarial Training

Since our mix-up strategy utilizes the gradients of the loss function with respect to the given inputs for saliency computation, we can incorporate adversarial training in our mix-up method without any additional computation cost.

For adversarial training on mixup data, we adapt the fast adversarial training method of Wong et al. (2020), which adds a uniform noise before creating an adversarial perturbation. As shown in Algorithm 2, we add the adversarial perturbation to the proper location of the mixed output, i.e., adding an adversarial signal to the corresponding input and location specified by \( z \). Note that the adversarial perturbation is added to each data probabilistically to prevent possible degradation in the generalization performance.

5. Implementation Details

First, to solve the discrete optimization problem with respect to the mask \( z \), we use \( \alpha \), \( \beta \) swap algorithm from the pyGCO python wrapper\(^1\). Although the minimization is performed example-wise in CPUs, the \( \alpha \), \( \beta \) swap algorithm converges quickly, since we restrict the size of the graph with down-sampling. Note that, in our experiments, the computational bottleneck of the method is in the forward-backward passes of the neural network. In our experiments, we use label space \( \mathcal{L} = \{0, \frac{1}{2}, 1\} \). In addition, we randomly sample the size of the graph, i.e., size of mask \( z \), from \( \{2 \times 2, 4 \times 4, 8 \times 8, 16 \times 16\} \), and down-sample the given mini-batch for all experiments.

We normalize the down-sampled saliency map, which is used as the unary term, to sum up to 1. This allows us to use consistent hyperparameters across all the models and datasets. To measure the distance between the two adjacent data regions, we compute the mean of the absolute values of differences on the boundaries. For the mixing ratio \( \lambda \), we randomly sample \( \lambda \) from Beta(\( \alpha \), \( \alpha \)) at each mini-batch. All of the computations in our algorithm except \( \alpha \), \( \beta \) swap are done in mini-batch and can be performed in parallel in GPUs. Note that for-loops in Algorithm 1 can be done in parallel by using the scatter function of PyTorch (Paszke et al., 2017).

---

\(^1\)https://github.com/Borda/pyGCO
forward-backward evaluation of the network per mini-batch to calculate the saliency signal. For some practitioners, a fairer comparison would be to compare the performances at a fixed number of network evaluations (i.e., for power conservation). In order to compare our method in this condition, we also test our method trained for half the epochs and with twice the initial learning rate. We denote this experiment as Puzzle Mix (half) in the experiment tables.

In addition, we report experiments with the adversarial training described in Algorithm 2 with \( p = 0.1 \). We denote this experiment as Puzzle Mix (adv) in the tables. We assess the adversarial robustness against FGSM attack of 8/255 \( \ell_{\infty} \) epsilon ball following the evaluation protocol of Zhang et al. (2018); Verma et al. (2019); Yun et al. (2019) for fair comparison. The results are summarized in Table 2 and Table 3, and adversarial robustness results against the PGD attack (Madry et al., 2017) are in Supplementary D.2.

We observe that Puzzle Mix outperforms other mixup baselines in generalization and adversarial robustness with WRN28-10 (Table 2) and PreActResNet18 (Table 3). With WRN28-10, Puzzle Mix improves Top-1 test error over the best performing baseline by 1.45%, and Puzzle Mix (half) outperforms by 1.17%. Puzzle Mix (adv) improves FGSM error rate over 8.41% than AugMix while achieving 1.39% lower Top-1 error rate than Manifold mixup, which had the best Top-1 score among baselines. We observe similar results with PreActResNet18. Puzzle Mix (adv) reduces the Top-1 error rate by 1.14% and the FGSM error rate by 12.98% over baselines.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Top-5 Error(%)</th>
<th>FGSM Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>21.14</td>
<td>6.33</td>
<td>63.92</td>
</tr>
<tr>
<td>Input</td>
<td>18.27</td>
<td>4.98</td>
<td>56.60</td>
</tr>
<tr>
<td>Manifold</td>
<td>17.40</td>
<td>4.37</td>
<td>60.70</td>
</tr>
<tr>
<td>Manifold†</td>
<td>18.04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CutMix</td>
<td>17.50</td>
<td>4.69</td>
<td>79.34</td>
</tr>
<tr>
<td>AugMix</td>
<td>20.44</td>
<td>5.74</td>
<td>55.59</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>15.95</td>
<td>3.92</td>
<td>63.71</td>
</tr>
<tr>
<td>Puzzle Mix (half)</td>
<td>16.23</td>
<td>3.90</td>
<td>66.74</td>
</tr>
<tr>
<td>Puzzle Mix (adv)</td>
<td>16.01</td>
<td>3.91</td>
<td>47.18</td>
</tr>
<tr>
<td>Puzzle Mix (half, adv)</td>
<td>16.39</td>
<td>3.94</td>
<td>46.95</td>
</tr>
</tbody>
</table>

Table 2. Top-1 / Top-5 / FGSM error rates on CIFAR-100 dataset of WRN28-10 trained with various mixup methods (400 epochs). † denotes the result reported in the original paper. Top-1 and Top-5 results are median test errors of models in the last 10 epochs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Top-5 Error(%)</th>
<th>FGSM Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>42.77</td>
<td>26.35</td>
<td>91.85</td>
</tr>
<tr>
<td>Input</td>
<td>43.41</td>
<td>26.98</td>
<td>88.68</td>
</tr>
<tr>
<td>Manifold</td>
<td>41.99</td>
<td>25.88</td>
<td>89.25</td>
</tr>
<tr>
<td>Manifold†</td>
<td>41.30</td>
<td>26.41</td>
<td>-</td>
</tr>
<tr>
<td>CutMix</td>
<td>43.33</td>
<td>24.48</td>
<td>87.19</td>
</tr>
<tr>
<td>AugMix</td>
<td>44.03</td>
<td>25.32</td>
<td>90.00</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>36.52</td>
<td>18.95</td>
<td>92.52</td>
</tr>
<tr>
<td>Puzzle Mix (half)</td>
<td>37.64</td>
<td>19.37</td>
<td>92.57</td>
</tr>
<tr>
<td>Puzzle Mix (adv)</td>
<td>38.55</td>
<td>20.48</td>
<td>82.07</td>
</tr>
<tr>
<td>Puzzle Mix (half, adv)</td>
<td>38.14</td>
<td>19.70</td>
<td>83.91</td>
</tr>
</tbody>
</table>

Table 3. Top-1 / Top-5 / FGSM error rates on CIFAR-100 dataset of PreActResNet18 trained with various mixup methods.

Supplementary C.2.

As in the CIFAR-100 experiment, Puzzle Mix shows significant performance gains both on the generalization performance and the adversarial robustness compared to other mixup baselines (Table 4).

Puzzle Mix trained with the same number of epochs achieves 36.52% in Top-1 test error, 5.47% lower than the strongest baseline, and the model trained with same network evaluations (half) outperforms the best baseline by 4.35%. Puzzle Mix trained with stochastic adversarial method (adv) achieves best Top-1 and FGSM error rate (\( \epsilon = 4/255 \)) compared to other mixup baselines providing 3.44% lower Top-1 error rate and 5.12% lower FGSM error rate.

6.1.3. IMAGE NET

In ImageNet experiment, we use ResNet-50 to compare the performance. In order to train the model on ImageNet more efficiently, we utilize the cyclic learning rate, and use pre-resized images following the training protocol in Wong et al. (2020). Hyperparameter settings are in Supplementary C.3. Consistent with the previous experiments on CIFAR-100 and Tiny-ImageNet, Puzzle Mix shows the best performance.
in both Top-1 / Top-5 error rate, achieving 0.43%, 0.24% improvement each, compared to the best baseline (Table 5).

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Error(%)</th>
<th>Top-5 Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>24.31</td>
<td>7.34</td>
</tr>
<tr>
<td>Input</td>
<td>22.99</td>
<td>6.48</td>
</tr>
<tr>
<td>Manifold</td>
<td>23.15</td>
<td>6.50</td>
</tr>
<tr>
<td>CutMix</td>
<td>22.92</td>
<td>6.55</td>
</tr>
<tr>
<td>AugMix</td>
<td>23.25</td>
<td>6.70</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>22.49</td>
<td>6.24</td>
</tr>
</tbody>
</table>

Table 5. Top-1 / Top-5 error rates on ImageNet on ResNet-50.

6.2. Robustness Against Corruption

Hendrycks et al. (2020) proposed AugMix which performs Input mixup between clean and augmented images to improve robustness against corrupted datasets as well as the generalization performance. AugMix uses Jensen-Shannon divergence (JSD) between network outputs of a clean image and two AugMix images as a consistency loss. However, computing the JSD term requires triple the network evaluations compared to other mixup methods to train the network.

We found that simply using our mixup algorithm between two AugMix images, improves both the generalization and corruption robustness over the training strategy with the JSD objective. Note that our method requires only one additional (versus two) network evaluation per each mini-batch. We denote this experiment setting as Puzzle Mix (aug).

We use CIFAR-100-C dataset (Hendrycks & Dietterich, 2019) to evaluate the corruption robustness. The dataset consists of 19 types of corruption, including noise, blur, weather, and digital corruption types. In Table 6, we report average test errors on CIFAR-100-C dataset as well as test errors on the clean CIFAR-100 test dataset. Table 6 demonstrates that our method using AugMix images improves both the generalization performance and the corruption accuracy by 3.95% and 2.31% each over AugMix baseline.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Corruption Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>21.14</td>
<td>49.08</td>
</tr>
<tr>
<td>AugMix</td>
<td>20.45</td>
<td>32.22</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>15.95</td>
<td>42.46</td>
</tr>
<tr>
<td>Puzzle Mix (aug)</td>
<td>16.50</td>
<td>29.91</td>
</tr>
</tbody>
</table>

Table 6. Top-1 / Corruption error rates on CIFAR-100 and CIFAR-100-C on WRN28-10.

6.3. Ablation Study

The generalization performance of Puzzle Mix stems from saliency-based multi-label masking and transport. We verified the effectiveness of these two factors in comparative experiments on CIFAR-100 with WRN28-10. Table 7 shows that Puzzle Mix with the binary label space (binary) has 1.44% higher Top-1 error rate than multi-label case, and Puzzle Mix without transport (mask only) has 0.43% higher Top-1 error rate than Puzzle Mix with transport.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Top-5 Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>21.14</td>
<td>6.32</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>15.95</td>
<td>3.92</td>
</tr>
<tr>
<td>Puzzle Mix (binary)</td>
<td>17.39</td>
<td>4.34</td>
</tr>
<tr>
<td>Puzzle Mix (mask only)</td>
<td>16.38</td>
<td>3.78</td>
</tr>
</tbody>
</table>

Table 7. Top-1 / Top-5 rates on CIFAR-100 dataset of WRN28-10 trained with our mixup methods.

We also verify the effects of different factors in stochastic adversarial training. In Algorithm 2, we add an adversarial perturbation to each data based on each Bernoulli sample \(\nu_i\) and apply linear decay with \(\delta\) sampled from the uniform distribution. From Table 8, we observe that using two independent random variables \(\nu_0\) and \(\nu_1\) (adv) has significant improvement in adversarial robustness over using one variable \(\nu_0 = \nu_1\). In the absence of linear decaying (fgsm), there is improvement in the FGSM error rate of 4.02%, but the Top-1 error increases by 0.41%. In all experiments, \(p\) is set to 0.1. We use FGSM attack of 8/255 \(\ell_\infty\) epsilon-ball and 7-step PGD attack with a 2/255 step size. Additional experiments regarding the effect of \(p\) value in adversarial training are available in Supplementary D.1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>FGSM Error(%)</th>
<th>PGD Error(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puzzle Mix (adv)</td>
<td>16.01</td>
<td>47.18</td>
<td>90.18</td>
</tr>
<tr>
<td>Puzzle Mix (fgsm)</td>
<td>16.42</td>
<td>43.16</td>
<td>91.19</td>
</tr>
<tr>
<td>Puzzle Mix (\nu_0 = \nu_1)</td>
<td>16.66</td>
<td>65.90</td>
<td>94.05</td>
</tr>
</tbody>
</table>

Table 8. Top-1 / FGSM / PGD error rates on CIFAR-100 dataset of WRN28-10 trained with our adversarial mixup methods.

7. Conclusion

We have presented Puzzle Mix, a mixup augmentation method for optimally leveraging the saliency information while respecting the underlying local statistics of the data. Puzzle Mix efficiently generates the mixup information in a mini-batch stochastic gradient descent setting and outperforms other mixup baseline methods both in the generalization performance and the robustness against adversarial perturbations and data corruption by a large margin on CIFAR-100, Tiny-ImageNet, and ImageNet datasets.
Acknowledgements

This research was supported in part by Samsung Electronics and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning). Hyun Oh Song is the corresponding author.

References


