Regularization Strategy for Point Cloud via Rigidly Mixed Sample Supplementary Material

Dogyoon Lee[1] Jaeha Lee[1] Junhyeop Lee[1] Hyeongmin Lee[1] Minhyeok Lee[1]
Sungmin Woo[1] Sangyoun Lee[1*]

[1]Yonsei University

{nemotio,jaeha0725,jun.lee,minimonia,hydragon516,smw3250,syleee}@yonsei.ac.kr

A. Implementation Details

We implemented RSMix with Nvidia RTX 2080Ti GPU using released codes from PointNet++ [6] and DGCNN [7] using TensorFlow [1] and PyTorch [4], respectively. We adopted original configurations of released codes except a training epoch for DGCNN [7]. Specifically, for PointNet [5] and PointNet++ [6], we adopted 1024 points without normals, batch size$=16$, and training epoch$=250$ using Adam optimizer [3] with initial learning rate$=0.001$, decay rate$=0.7$, and decay step$=200000$. For DGCNN [7], we also adopted 1024 points without normals, batch size$=32$, k$=20$, and training epoch$=500$ using SGD solver with momentum$=0.9$ and initial learning rate$=0.1$, which decays according to a cosine annealing strategy [2]. In addition, we adopted four conventional augmentations: jitter($\sigma^2 = 0.01$); scaling($0.8 \sim 1.25$); shifting(range$=0.1$); and y-axis rotation *i.e.* gravity, when training the networks as we mentioned in our paper. For DropPoint, we applied drop ratio($=0 \sim 0.875$) as same as PointNet++ [6]. Lastly, we applied RSMix with a probability of 0.5.

B. Visualized Samples

Figure 1 shows qualitative results of mixed samples with RSMix. Yellow and purple parts indicate Rigid Subsets (RSs) to be extracted from the each sample to generate mixed samples, which comprise of red and green parts of point clouds. Visualization demonstrates that RSMix successfully transfers semantic structural information of the each sample to the synthesized virtual sample by preserving shape of original data.

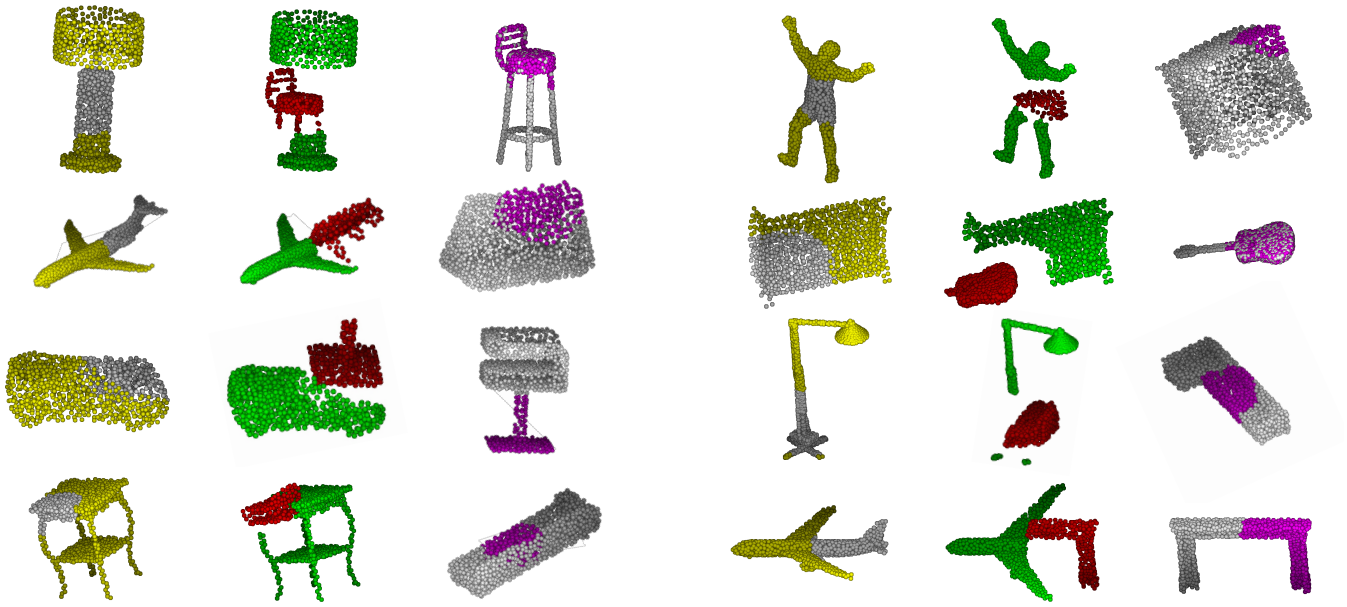

Figure 1: Qualitative results of mixed samples with RSMix. Yellow (left) and purple (right) colors indicate Rigid Subsets from each samples to generate virtual mixed samples (middle), which are comprised of red and green colors, respectively.

C. Rigid Subset Mix Algorithm

This section presents three simple code-level pseudo-codes to explain our algorithm in detail.

C.1. Overall Algorithm

Algorithm 1 describes the overall algorithm of RSMix. B, N, and C denote the size of minibatch data $\mathcal{P}^{\alpha}_{batch}$, the cardinality ($|\cdot|$) of each point sets, and information of each point, which includes three-dimensional coordinate and features.

First, we sample the r_{rigid} for RSMix from beta distribution Beta(θ, θ). Then we randomly shuffle the order of the input minibatch data $\mathcal{P}^{\alpha}_{batch}$ and label L^{α}_{batch} along the first axis of the tensors to generate paired minibatch data $\mathcal{P}^{\beta}_{batch}$ and L^{β}_{batch}. Second, we create empty minibatch data $\mathcal{P}^{\alpha}_{mix,batch}$ and mixture ratio λ to store outputs processed through RSMix algorithm. Third, we find coordinates of query points q^{α} and q^{β}, whose indices q^{α}_{idx} and q^{β}_{idx} are randomly sampled from $\mathcal{P}^{\alpha}_{batch}$ and $\mathcal{P}^{\beta}_{batch}$. We also explore indices of neighbored points, $\mathcal{S}^{\alpha}_{idx}$ and $\mathcal{S}^{\beta}_{idx}$, through Neighboring Rigid Subset Algorithm (Algorithm 2). Lastly, we extract Rigid Subsets (RSs) from given samples and mix them to generate virtual data $\mathcal{P}^{\alpha}_{mix,batch}$ and λ, mixed and stored informations with Extraction & Insertion algorithm (Algorithm 3).

Algorithm 1 Overall Rigid Subset Mix Algorithm

During Training,
Input: $\mathcal{P}^{\alpha}_{batch}$, L^{α}_{batch}, θ(usually 1.0) ▷ $\mathcal{P}^{\alpha}_{batch}$: input = (B×N×C), L^{α}_{batch} : Label = (B×1), where C is 3 or 6(w/ Normal).
Output: $\mathcal{P}^{\alpha}_{mix,batch}$, L^{α}_{batch},L^{β}_{batch}, λ ▷ $\mathcal{P}^{\alpha}_{mix,batch}$: Mixed data(B×N×C), L^{α}_{batch} : Original label(B×1), L^{β}_{batch} : Mixed label(B×1).
▷ λ:(B,).
if RSMix == True **then**
  r_{rigid} = Beta(θ, θ)
  $\mathcal{P}^{\beta}_{batch}$, L^{β}_{batch}= batch_wise_random_shuffle($\mathcal{P}^{\alpha}_{batch}$, L^{α}_{batch})
  $\mathcal{P}^{\alpha}_{mix,batch}$, λ Create ▷ $\mathcal{P}^{\alpha}_{mix,batch}$:(B×N×C), λ :(B,) are empty: to store mixed data and lambda.
  q^{α}_{idx}, q^{β}_{idx} = Randomly sampled indices from $\mathcal{P}^{\alpha}_{batch}$, $\mathcal{P}^{\beta}_{batch}$ ▷ Randomly choose query points and save indices. (B,).
  $\mathcal{S}^{\alpha}_{idx}$, q^{α} =**Neighboring Rigid Subset Algorithm**($\mathcal{P}^{\alpha}_{batch}$, q^{α}_{idx}, r_{rigid}, n^{max}) ▷ Note: $n^{max} == \frac{N}{2}$.
  $\mathcal{S}^{\beta}_{idx}$, q^{β} =**Neighboring Rigid Subset Algorithm**($\mathcal{P}^{\beta}_{batch}$, q^{β}_{idx}, r_{rigid}, n^{max}) ▷ $\mathcal{S}^{\alpha}_{idx}$, $\mathcal{S}^{\beta}_{idx}$: Masked indices. (B×1×n^{max}).
  ▷ q^{α}, q^{β} : query points. (B×1×C).
  $\mathcal{P}^{\alpha}_{mix,batch}, \lambda$ =**Extraction & Insertion Algorithm**($\mathcal{S}^{\alpha}_{idx}, \mathcal{S}^{\beta}_{idx}, \mathcal{P}^{\alpha}_{batch}, \mathcal{P}^{\beta}_{batch}, q^{\alpha}, q^{\beta}$, r_{rigid}, $n^{max}, \mathcal{P}^{\alpha}_{mix,batch}, \lambda$)
  return $\mathcal{P}^{\alpha}_{mix,batch}, L^{\alpha}_{batch}, L^{\beta}_{batch}, \lambda$
end if

C.2. Neighboring Rigid Subset

We implemented a neighboring algorithm using loop structure and batch-wise computation.

First, we specify exact coordinates of query points q from q_{idx} and $\mathcal{P}_{batch}$ using for-loop. Second, we extract indices of neighbored points through neighboring functions $\mathcal{A}_{ball}$ or $\mathcal{A}_{knn}$. Each function computes adjacencies of points from query points q using function Query_ball with r_{rigid}, which is implemented on PointNet++ [6] and K-Nearest Neighbor(KNN) with k_{sample}, respectively. Specifically, Query_ball function extracts indices of the points within the distance r_{rigid} from a query point $q[i][0]$. Neighboring algorithm is implemented with batch-wise computation for effectiveness of RSMix.

Algorithm 2 Neighboring Rigid Subset Algorithm

Input: $\mathcal{P}_{batch}$, q_{idx}, r_{rigid}, n^{max} ▷ $\mathcal{P}_{batch}$: input = (B×N×C), q_{idx} : (B,),r_{rigid}, n^{max} : scalar.
Output: $\mathcal{S}_{idx}$, q ▷ $\mathcal{S}_{idx}$:Masked indices (B×1×n^{max}), q :query point (B×1×C).

q = (B×1×C) empty data create
for i =1, 2, . . . , B **do**
  $q[i][0]$=$\mathcal{P}_{batch}[i][q_{idx}[i][0]]$ ▷ Initalize query point value.
end for
if $\mathcal{A} == \mathcal{A}_{ball}$ **then** ▷ Query_ball function of PointNet++ [6].
  $\mathcal{S}_{idx}$ = Query_ball($\mathcal{P}_{batch}$, q_{idx}, r_{rigid}, n^{max})
else if $\mathcal{A} == \mathcal{A}_{knn}$ **then** ▷ KNN function manually revised from query_ball function of PointNet++ [6].
  $k_{sample} = n^{max} \times r_{rigid}$ ▷ k_{sample}: the cardinality of neighboring point set.
  $\mathcal{S}_{idx}$ = KNN($\mathcal{P}_{batch}$, q_{idx}, k_{sample}, n^{max})
end if
return $\mathcal{S}_{idx}$, q

C.3. Extraction & Insertion

Algorithm 3 describes the Extraction & Insertion process in detail. As we defined a label mixture ratio λ into three cases in our paper, Extraction & Insertion algorithm is implemented with the three branches to deal with the cases. Notations in below algorithm 3 are identical to those of above algorithm 1 except new ones in algorithm 3.

First of all, we pre-compute q_{dist}, which are translation vectors for each batch data to insert a RS $\mathcal{S}^\beta$ to a RS $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$, with only coordinate information. Due to different cardinalities of each neighbored point set in batch, which are extracted through above algorithm 2, we process the Extraction & Insertion using for-loop, since it is hard to apply the batch-wise computation. In addition, the algorithm is divided into three branches as we mentioned above: $\mathcal{S}^\alpha_{idx}[i][0][0]$ ==N; $\mathcal{P}^\alpha = \mathcal{S}^\alpha$N; and otherwise, which are same as three cases in our paper: $\mathcal{P}^\alpha = \mathcal{S}^\alpha$; $\mathcal{S}^\beta = \emptyset$; and otherwise, respectively.

For first case, we use an original data $\mathcal{P}^\alpha_{batch}[i]$ as processed data $\mathcal{P}_{tmp}$ intactly and define a corresponding mixture ratio $\lambda_{tmp} = 0$. For second case, we use an extracted subset $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$ as processed data $\mathcal{P}_{tmp}$ randomly duplicating points in $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$ to maintain a cardinality of the point set as N, since there is no point in RS $\mathcal{S}^\beta$ and we also set $\lambda_{tmp} = 0$. This strategy also make RSMix can cover an additional case of partially-removed samples. For last case, before insertion, we have to control the $|\mathcal{S}^\beta|$ as much as difference between $|\mathcal{S}^\beta|$ and $|\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha|$, since they are usually different. We introduce random sampling or duplicating points in the $\mathcal{S}^\beta$ to preserve overall shapes of extracted samples. Finally, we generate a mixed sample $\mathcal{P}_{tmp}$ by inserting $\mathcal{S}^\beta$ to $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$ using pre-computed vector q_{dist} with ith index. For label mixture ratio λ_{tmp} in this case, we set it as the ratio of $|\mathcal{S}^\beta|$ w.r.t. $|\mathcal{P}_{tmp}| (= |\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha|)$. This temporal mixed data $\mathcal{P}_{tmp}$ and mixture ratio λ are successively saved to already created container $\mathcal{P}^\alpha_{mix,batch}$ and λ. Therefore, we effectively generate virtual mixed data $\mathcal{P}^\alpha_{mix,batch}$ and corresponding mixture ratio λ.

Algorithm 3 Extraction & Insertion Algorithm

Input: $\mathcal{S}^\alpha_{idx}, \mathcal{S}^\beta_{idx}, \mathcal{P}^\alpha_{batch}, \mathcal{P}^\beta_{batch}, q^\alpha, q^\beta, r_{rigid}, n^{max}, \mathcal{P}^\alpha_{mix,batch}, \lambda$ ▷ Same notations as defined in above algorithms.
Output: $\mathcal{P}^\alpha_{mix,batch}, \lambda$

$q_{dist} = q^\alpha[:, :, :3] - q^\beta[:, :, :3]$ ▷ To translate the extracted subset to another extracted subset
for i =1, 2, . . . , B **do**
  if $\mathcal{S}^\alpha_{idx}[i][0][0]$==N **then** ▷ Because mask values for indices are N, which is the original cardinality of point set in each batch.
    $\mathcal{P}_{tmp} = \mathcal{P}^\alpha_{batch}[i]$ ▷ if $\mathcal{S}^\alpha_{idx}[i][0][j]$ ==N, that means that a point $\mathcal{P}^\alpha_{batch}[i][j]$ is out of neighboring function.
    λ_{tmp} =0
  else if $\mathcal{S}^\beta_{idx}[i][0][0]$==N **then** ▷ $|\cdot|$: Cardinality of point cloud.
    $\mathcal{S}^\alpha_{idx} = \mathcal{S}^\alpha_{idx}$[i] with removing invalid indices ▷ Filter the indices to extract Rigid Subset $\mathcal{S}^\alpha$ from $\mathcal{P}^\alpha_{batch}[i]$.
    $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$ =Remove elements whose indices are in $\mathcal{S}^\alpha_{idx}$ from $\mathcal{P}^\alpha_{batch}[i]$ ▷ Extract $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$:($|\mathcal{P}^\alpha_{batch} - \mathcal{S}^\alpha| \times$C).
    $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$ = Randomly duplicate the points in $\mathcal{P}^\alpha_{batch}$[i]$- \mathcal{S}^\alpha$ as much as length of $\mathcal{S}^\alpha_{idx}$ ▷ To maintain cardinality of point set as N.
    $\mathcal{P}_{tmp} = \mathcal{P}^\alpha_{batch}$[i]$- \mathcal{S}^\alpha$ ▷ $\mathcal{P}_{tmp}$: (B×N×C).
    λ_{tmp} =0
  else
    $\mathcal{S}^\alpha_{idx} = \mathcal{S}^\alpha_{idx}$[i] with removing invalid indices
    $\mathcal{S}^\beta_{idx} = \mathcal{S}^\beta_{idx}$[i] with removing invalid indices ▷ Filter the indices to extract Rigid Subset $\mathcal{S}^\beta$ from $\mathcal{P}^\beta_{batch}[i]$.
    if $|\mathcal{S}^\alpha_{idx}| == |\mathcal{S}^\beta_{idx}|$ **then**
      $\mathcal{S}^\beta_{idx,ctrl} = \mathcal{S}^\beta_{idx}$
    else if $|\mathcal{S}^\alpha_{idx}| > |\mathcal{S}^\beta_{idx}|$ **then**
      $\mathcal{S}^\beta_{idx,ctrl}$ =Randomly duplicate the indices in $\mathcal{S}^\beta_{idx}$ as much as $(|\mathcal{S}^\alpha_{idx}| - |\mathcal{S}^\beta_{idx}|)$
    else
      $\mathcal{S}^\beta_{idx,ctrl}$ =Randomly sample the indices in $\mathcal{S}^\beta_{idx}$ as much as $(|\mathcal{S}^\alpha_{idx}|)$
    end if
    $\mathcal{P}^\alpha_{batch}$[i]$- \mathcal{S}^\alpha$ =Remove elements whose indices are in $\mathcal{S}^\alpha_{idx}$ from $\mathcal{P}^\alpha_{batch}$[i] ▷ $\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha$:($|\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha| \times$C)
    $\mathcal{S}^\beta$ =Take elements whose indices are in $\mathcal{S}^\beta_{idx,ctrl}$ from $\mathcal{P}^\beta_{batch}[i]$ ▷ $\mathcal{S}^\beta$:($|\mathcal{S}^\beta| \times$C).
    $\mathcal{S}^\beta[:, :3] = q_{dist}[i] + \mathcal{S}^\beta[:, :3]$ ▷ Translated Rigid Subset.
    $\mathcal{P}_{tmp}$ =Concatenate($\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha, \mathcal{S}^\beta$) ▷ $\mathcal{P}_{tmp}$: (B×N×C)
    $\lambda_{tmp} = \frac{|\mathcal{S}^\beta|}{|\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha| + |\mathcal{S}^\beta|}$ ▷ $|\mathcal{P}^\alpha_{batch}[i] - \mathcal{S}^\alpha| + |\mathcal{S}^\beta|$ =N in this case.
  end if
  $\mathcal{P}^\alpha_{mix,batch}[i][:, :] = \mathcal{P}_{tmp}[:, :]$
  $\lambda[i] = \lambda_{tmp}$
end for
return $\mathcal{P}^\alpha_{mix,batch}, \lambda$

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[6] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[7] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.