Appendix of *Divide and Conquer: Compositional Experts for Generalized Novel Class Discovery*

This is Appendix of the CVPR 2022 paper titled *Divide and Conquer: Compositional Experts for Generalized Novel Class Discovery*. Clicking on each entry in Contents below will easily navigate you to its corresponding section.

Contents

A. Motivations

In GNCD, base and novel sets contain different training targets — for base set we have accurate and consistent ground-truth labels, while for novel set the training targets (generated pseudo labels) are less accurate and may change during training. We aim to build a model able to handle both base and novel classes, which is challenging due to (1) the ambiguity between base and novel classes, and (2) the lack of supervision on novel data. To address (1), we propose to model base and novel data using two groups of compositional experts with complementary specialties from both batch-wise and class-wise perspectives. For (2), we propose a Local Aggregation strategy to refine the pseudo labels by encouraging local consistency among novel data.

Our idea of the compositional structure comes from the observations towards GNCD output space, which can be clearly characterized with a batch-class view (*cf.* Fig. 1b in our paper). From this view the GNCD output space exhibits a compositional nature in both batch-wise and class-wise perspectives. We propose to model this compositional nature using a group of batch-wise experts (Fig. 1c) and another group of class-wise experts (Fig. 1d). Each group of experts is capable of characterizing the whole dataset, yet presents different specialties — with batch-wise experts capturing separability between base and novel sets, and class-wise experts modeling discriminability within each set of classes. Considering the challenges of GNCD, we keep both groups of experts to make the most of their learning abilities by allowing complementary collaborations.

We further propose to address the aforementioned inconsistent training targets using local aggregation for novel samples, which complement the global-to-local pseudo-labeling with local-to-local regularization. Extensive experiments have validated the effectiveness of the above ideas, which compose our proposed ComEx and contribute to its superior performance in GNCD.

B. Implementation Details

B.1. Over-Clustering and Multi-Head Clustering

We follow [2, 6, 10] to use over-clustering and multi-head clustering strategies for better clustering performance. Over-clustering [10] is to output more fine-grained partitions of the novel set, which can enhance the feature representations. Concretely, an over-clustering head serves as another novel-batch/class expert with $C^b + m \times C^n$ or $m \times C^n$ output neurons, where m is the over-clustering factor.

Multi-head clustering [2, 10] smooths down possible clustering degeneration with multiple clustering heads. Following [6], we apply this strategy to both clustering and over-clustering heads in novel-batch/class expert, and iterate over these heads in training. We refer the reader to [6] for more details. In experiments we use the same head number and over-clustering factor as in [6] for fair comparisons.

B.2. Dataset Details

Three datasets are used for evaluations, namely CIFAR10/100 [11] and ImageNet [5]. The CIFAR10/100[1]

[1] https://www.cs.toronto.edu/~kriz/cifar.html

dataset is released under the MIT license, and the ImageNet[2] dataset's terms of access allow non-commercial use for education and research purposes. In detail, CIFAR10/100 contains 50,000 images from 10/100 classes, each of which is in size of 32×32, while ImageNet is a large-scale dataset containing 1.28 million images from 1000 classes with an average image resolution of 469×387. For CIFAR10/100, we use 5/5, 80/20, 50/50, 30/70 (introduced in Sec. C.2) as base/novel class splits. For ImageNet, following [8, 9], we divide the whole 1000 classes into 882 and 118 classes, in which the 882 classes are used as the base classes. We further sample three subsets from the 118 classes, each of which contains 30 classes, and we use one of them at a time as the novel set. For evaluations on ImageNet, the results are averaged over the three subsets. The split information can also be found in Tab. 2 in our paper.

B.3. Training Details

We follow [6] to pre-train our model for 200 epochs on the base set to learn basic semantic discriminability, and then fine-tune it for another 200 epochs on both base and novel sets to learn a unified model. In practice, we directly fine-tune our model based on the pre-trained model officially provided in [6] for fair comparisons. For model optimization, we use SGD with momentum 0.9 as the optimizer, linear warmup (10 epochs) + cosine annealing (0.2 base, 0.001 min) as the learning rate scheduler, with the weight decay rate set to 1.5×10^{-4} and a 256 batch size. The temperature parameter τ in Eq. (1) and (5) is set to 0.1. We use a size of 500 for the queue $\mathcal{Q}$, and the hyperparameter α in Eq. (6) is set to 0.5. For pseudo-labeling, we inherit the implementation in [3] and use $\epsilon = 0.05$ and 3 iterations for the Sinkhorn-Knopp algorithm [4]. We provide in Algorithm A the pseudocode of our proposed ComEx in a PyTorch-like fashion, where we leave out ℓ_2-normalizations, queuing function, over-clustering, multi-head clustering, and SGD update for simplicity. In Lines 46 and 80–84 we use a distribution sharpening strategy, inspired by [1], to reduce entropy of the pseudo labels, which helps to generate better training targets for novel samples.

B.4. Evaluation Details

As mentioned in our paper, we involve two types of evaluations, namely ***(a)*** *task-aware* and ***(b)*** *task-agnostic* evaluations, which can be further divided into four specific protocols based on which set an evaluation takes place: ***(a1)*** *training split of the novel set*, ***(a2)*** *testing split of the novel set*, ***(a3)*** *testing split of the base set*, and ***(b1)*** *both testing splits of the base and novel sets*.

For evaluations on *(a1)* and *(a2)*, we only use the predictions from novel-batch and novel-class experts; for *(a3)* we only use base-batch and base-class experts. For evaluations on *(b1)*, we use the combination of both batch- and class-wise experts. Note that in our paper we only report comparisons with state-of-the-art methods under *(a1)* and *(b1)*, which is due to 1) no existing results under the other protocols can be found, and 2) these two protocols should be enough for comprehensive evaluations. We did not involve the ImageNet dataset for task-agnostic evaluations for two reasons: 1) the training is considerably time-consuming (one typical run takes over 3 days on an NVIDIA RTX 2080 Ti GPU for 60 epochs with 256 batch size); 2) novel classes (30) are too few compared to base classes (882), making it inconclusive for task-agnostic evaluations. As a result, to our best knowledge, task-agnostic evaluations on ImageNet are currently not seen in any existing works. Hopefully a smaller base set will enable further evaluations on ImageNet, which we leave as our future work.

[2] https://image-net.org/

Protocol →	Task-agnostic			-aware
Method ↓	Base	Novel	All	Novel
UNO (Orig.)	93.5	93.3	93.4	96.1±0.5
UNO (Reprod.)	93.6	89.9	91.8	92.6±0.5
ComEx (Ours)	**95.0**	**92.6**	**93.8**	**93.6±0.3**

Table A. Comparison with UNO [6] on *CIFAR10*. Results are reported in classification/clustering accuracy (%) using task-agnostic ***(b1)*** and task-aware ***(a1)*** evaluation protocols.

Protocol →	Task-agnostic			-aware
Method ↓	Base	Novel	All	Novel
UNO (Orig.)	73.2	73.1	73.2	85.0±0.6
UNO (Reprod.)	74.4	68.0	73.1	81.3±0.6
ComEx (Ours)	**75.2**	**77.3**	**75.6**	**85.7±0.7**

Table B. Comparison with UNO [6] on *CIFAR100-20*. Results are reported in classification/clustering accuracy (%) using task-agnostic ***(b1)*** and task-aware ***(a1)*** evaluation protocols.

Protocol →	Task-agnostic			-aware
Method ↓	Base	Novel	All	Novel
UNO (Orig.)	71.5	50.7	61.1	52.9±1.4
UNO (Reprod.)	72.3	47.0	59.7	49.3±1.3
ComEx (Ours)	**75.3**	**53.5**	**64.4**	**53.4±1.3**

Table C. Comparison with UNO [6] on *CIFAR100-50*. Results are reported in classification/clustering accuracy (%) using task-agnostic ***(b1)*** and task-aware ***(a1)*** evaluation protocols.

C. Additional Results

C.1. Reproduced Results of UNO

The official code of the current state-of-the-art method, UNO [6], had a peculiar bug that results in uncommon

high accuracy on all datasets, especially CIFAR10. To be specific, the softmax normalization was mistakenly applied to the batch dimension (`dim=1`) instead of the logits dimension (`dim=-1`) when calculating the prediction loss (*cf.* Lines 49 and 51 in Algorithm A). The authors have acknowledged and removed this bug, and further developed UNO v2 with 1) stronger image augmentations, and 2) more training epochs (500, originally 200), which indeed outperforms original UNO on CIFAR100-20 and CIFAR100-50 by a large margin. We refer the reader to this GitHub issue page[3] for more details.

Since our proposed ComEx was built upon original UNO, while the time did not allow us to migrate our model to computationally expensive UNO v2, we provide here the reproduced results of original UNO with bug removed. In particular, we found that its default hyperparameters are less optimal after bug removed, and thus we rerun the experiments with the same hyperparameters used in our ComEx, yielding better results. We report here the reproduced results (marked with "Reprod.") as well as the ones copied from their paper ("Orig.") on CIFAR10, CIFAR100-20 and CIFAR100-50 in Tabs. A to C, from which we can observe the authentic improvement of our proposed ComEx against the current state-of-the-art UNO under both task-aware and task-agnostic evaluations.

Concerning the high functionality of the bug, a rational explanation could be that normalizing at a wrong dimension actually increases the difficulty of learning both base and novel classes, which 1) encourages the model to focus more on the novel classes by 2) alleviating the over-fitting issue towards the base classes that are much easier to learn due to accurate and consistent training targets. This over-fitting issue is more obvious on CIFAR100-20 in Tab. B, where more base classes are involved. In contrast, ComEx is able to balance between base and novel classes thanks to 1) the collaboration of experts and 2) the strong learning signal for novel classes from local aggregation. This phenomenon, again, suggests that we should not blindly focus on the novel set performance, but should take into consideration both base and novel classes, which is the essence to let GNCD stand out from unsupervised clustering.

C.2. Results on CIFAR100-70

We further evaluate our proposed ComEx on an imbalanced data split setting that involves more novel classes, namely CIFAR100-70, containing 30 base classes and 70 novel classes. The evaluation results are shown in Tab. D. From Tabs. A to D we can observe that our proposed ComEx is able to produce robust GNCD results, consistently outperforming the current state-of-the-art UNO.

[3] https://github.com/DonkeyShot21/UNO/issues/4

Protocol →	Task-agnostic			-aware
Method ↓	Base	Novel	All	Novel
UNO [6]	70.0	38.1	47.7	38.5±1.4
ComEx (Ours)	**73.5**	**41.5**	**51.1**	**41.8±0.7**

Table D. Comparison with UNO [6] on *CIFAR100-70*. Results are reported in classification/clustering accuracy (%) using task-agnostic ***(b1)*** and task-aware ***(a1)*** evaluation protocols.

D. Parameter Analysis

We study the sensitivity of queue size N^q and α in Eq. (6), as shown in Fig. A. The experiments are conducted on CIFAR100-50 with varying N^q or α by setting all the other hyperparameters to their default.

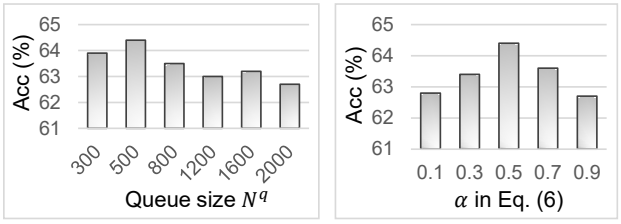

Figure A. Parameter sensitivity analysis of queue size N^q and α in Eq. (6). We use the task-agnostic ***(b1)*** evaluation protocol, and report here the accuracy of "All".

From Fig. A we can observe that a large queue size does not always bring about better performance. The reason can be that a moderate size should provide enough diversity since all N^q elements are used in a soft way, different from selecting k-NNs [12] that benefits from a larger queue. However, carefully tuning the hyperparameters (such as temperature τ in Eq. (5)) for a larger queue may arguably further increase the performance, but we intended to use a small queue size for computational efficiency. As to α, the performance reaches optimal when $\alpha = 0.5$, suggesting that equally treating global-to-local and local-to-local regularization should maximize the learning efficiency.

E. More Discussions

E.1. Limitations

Our proposed ComEx assumes a known number of novel classes. This assumption does not always hold in real-world scenarios where novel classes are continuously showing up. Although we can resort to the off-the-shelf tool [7] to estimate the number of novel classes beforehand, it remains an open problem to integrate class number estimation into an end-to-end GNCD training. Another limitation is the increased computational requirement. Although this should be unnoticeable in most cases since experts are only shallow-layer MLPs, it can still make a difference on computation-limited equipment like smartphones and other

portable devices. We would like to see these limitations as the future direction, aiming for a flexible and light-weighted GNCD solution.

E.2. Negative Societal Impact

Our proposed approach is designed to recognize both known and unknown classes, which benefits the deployment on systems dealing with possible new classes. As developed based on the curated data, our proposed method may not work as expected on real-world data that poses more noise and diversity. Putting too much faith in its predictions can result in grave consequence under life-critical scenarios, such as medical diagnosis and autonomous driving. In view of this, a stand-alone validation before any real-world deployment should be necessary.

References

[1] Mahmoud Assran, Mathilde Caron, Ishan Misra, Piotr Bojanowski, Armand Joulin, Nicolas Ballas, and Michael Rabbat. Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples. In *Proc. ICCV*, 2021. 2

[2] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proc. ECCV*, 2018. 1

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Proc. NeurIPS*, 2020. 2

[4] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Proc. NeurIPS*, 2013. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009. 1

[6] Enrico Fini, Enver Sangineto, Stéphane Lathuilière, Zhun Zhong, Moin Nabi, and Elisa Ricci. A unified objective for novel class discovery. In *Proc. ICCV*, 2021. 1, 2, 3

[7] Kai Han, Andrea Vedaldi, and Andrew Zisserman. Learning to discover novel visual categories via deep transfer clustering. In *Proc. ICCV*, 2019. 3

[8] Yen-Chang Hsu, Zhaoyang Lv, and Zsolt Kira. Learning to cluster in order to transfer across domains and tasks. In *Proc. ICLR*, 2018. 2

[9] Yen-Chang Hsu, Zhaoyang Lv, Joel Schlosser, Phillip Odom, and Zsolt Kira. Multi-class classification without multi-class labels. In *Proc. ICLR*, 2019. 2

[10] Xu Ji, Joao F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proc. ICCV*, 2019. 1

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 1

[12] Zhun Zhong, Enrico Fini, Subhankar Roy, Zhiming Luo, Elisa Ricci, and Nicu Sebe. Neighborhood contrastive learning for novel class discovery. In *Proc. CVPR*, 2021. 3

Algorithm A Core implementation of ComEx, PyTorch-like.

```
# load two batches from two sets (N_b, N_n samples)
for x_b, x_n in zip(base_loader, novel_loader):
  # get two views of each using two augmentations
  x_b1, x_b2 = t1(x_b), t2(x_b)
  x_n1, x_n2 = t1(x_n), t2(x_n)

  # extract visual features of two views
  # z_b: [2, N_b, d_], z_n: [2, N_n, d_], d_: feat dim
  z_b = image_encoder(cat([x_b1.unsqueeze(0),
                           x_b2.unsqueeze(0)], dim=0))
  z_n = image_encoder(cat([x_n1.unsqueeze(0),
                           x_n2.unsqueeze(0)], dim=0))
  z = cat([z_b, z_n], dim=1)  # [2, N_b+N_n, d_]

  # get expert outputs and low-dim feats; C_b/n: cls num
  batch_y_b = batch_expert_b(z_b)     # [2, N_b, C_b+C_n]
  batch_z_n = batch_expert_n_(z_n)    # [2, N_n, d]
  batch_y_n = batch_expert_n(batch_z_n)#[2,N_n, C_b+C_n]

  class_y_b = class_expert_b(z)       # [2, N_b+N_n, C_b]
  class_z_n = class_expert_n_(z)      # [2, N_b+N_n, d]
  class_y_n = class_expert_n(class_z_n)#[2,N_b+N_n, C_n]

  mean_z_n = (batch_z_n + class_z_n[:, N_b:, :]) / 2

  # create targets for x_b and x_n
  target_b = one_hot(ground_truth_b) # [N_b, C_b]
  target = zeros(2, N_b+N_n, C_b+C_n)
  for i in range(2):
    target[i, :N_b, :C_b]= target_b
    target[i, N_b:, C_b:]= (sk(batch_y_n[i, :, C_b:])+
                            sk(class_y_n[i, N_b:, :]))/2

  # stack into queue as queue_z_n, queue_tar_n; size N_q
  # queue_z_n: [2, N_q, d], queue_tar_n: [2, N_q, C_n]
  queuing(mean_z_n, target[:, N_b:, C_b:])

  # calculate sim weighted neighbor target [2, N_n, C_n]
  sim = einsum("vnd,vqd->vnq", mean_z_n, queue_z_n)
  sim = softmax(sim / temperature, dim=-1)
  neighbor_tar = einsum("vnq,vqc->vnc",sim, queue_tar_n)

  # final training target for x_n
  target[:, N_b:, C_b:] = alpha * target[:, N_b:, C_b:]+
                          (1-alpha) * neighbor_tar
  target[:, N_b:, C_b:] = sharpen(target[:, N_b:, C_b:])

  # gather predictions from four experts
  batch_pred = cat([batch_y_b, batch_y_n], dim=1)
  batch_pred = softmax(batch_pred / temperature, dim=-1)
  class_pred = cat([class_y_b, class_y_n], dim=-1)
  class_pred = softmax(class_pred / temperature, dim=-1)

  # calculate swap prediction losses
  batch_loss = -mean(target[0] * log(batch_pred[1])+
                     target[1] * log(batch_pred[0])) / 2
  class_loss = -mean(target[0] * log(class_pred[1])+
                     target[1] * log(class_pred[0])) / 2

  # calculate regularization loss on batch experts
  reg_loss = (norm(batch_y_b[:, :, C_b:])+
              norm(batch_y_n[:, :, :C_b])) / 2

  # total loss
  loss = batch_loss + class_loss + reg_loss

# Sinkhorn-Knopp algorithm
@ torch.no_grad()
def sk(logits, eps=0.05, iters=3):
  Y = exp(logits / eps).T
  Y /= sum(Y)
  K, B = Y.shape
  u, r, c = zeros(K), ones(K)/K, ones(B)/B
  for _ in range(iters):
    u = sum(Y, dim=1)
    Y *= (r / u).unsqueeze(1)
    Y *= (c / sum(Y, dim=0)).unsqueeze(0)
  return (Y / sum(Y, dim=0, keepdim=True)).T

# reduce entropy of probability distribution
def sharpen(prob, p=0.5):
  sharpened = prob ** (1. / p)
  sharpened /= sum(sharpened, dim=-1, keepdim=True)
  return sharpened
```