We thank the reviewers for their valuable comments and recognition of the novelty and results of our method, *e.g.*, this work is well structured and motivated [R4] / explained in sufficient detail [R5]; the results show improvements [R2 R5] / effectiveness [R4]; experiments and ablation studies are sound [R2] / comprehensive [R4] / reasonable [R5]; "I'm positive to see such a novel idea" [R4]; "the proposed method features some technical novelty" [R5]; "I think this is a good submission" [R5]. We respond to the major comments below but will address all feedback in our revised version.

[R2] **Roles of $\mathbf{S}^{\text{pos}}$, graphs and proxies.** $\mathbf{S}^{\text{pos}}$, together with proxies, graphs, and all other modules, are designed to learn a better embedding network (in our case, the Inception backbone), and will be totally removed during testing, like ProxyNCA [12]. During training, the global graph $\mathbf{S}$ is used to construct k-NN sub-graphs $\mathbf{W}$, and immediately $\mathbf{W}$ serves for the proposed *reverse label propagation* in Eq. (6), during which the label message of proxy nodes will be passed to the sample nodes with a certain probability score according to $\mathbf{W}$. Thus, graphs are indispensable in our method, and constructing k-NN sub-graphs is conducive to learning discriminative proxies by focusing more on the local manifold of each sample. Proxies are globally learnable "cluster centers" while Clustering [13] directly regards input samples as cluster centers, and we will further clarify the differences between our method and other approaches.

[R2 R5] **Constraints among proxies.** There are actually two types of constraints among proxies in our method, *i.e.*, a "soft" constraint, by encouraging proxies to be close to their anchor samples (**L135–143**), and a "hard" one, by forcing similar proxies to be close to each other while dissimilar ones far away from each other (**L192–195**). The soft constraint allows capturing diverse intra-class differences while the hard one concentrates on the discrimination of the embedding space. In practice, similar proxies tend to be sufficiently close to each other in the later training stage. We argue that, during training, our model dynamically trades off between diversity and discrimination to achieve better performance.

[R4] **Selecting N and r.** Empirically, a large N is suitable for datasets with large intra-class variance while a small sub-graph (with $r = 0.05$) is optimal in most cases. Accordingly, the effects of N and r follow similar patterns on CUB as on Cars196 (both with $N = 12$); for SOP dataset the optimal case is $N = 1$ due to its low intra-class variance.

[R4 R5] **Performance on SOP.** Though our method does not consistently outperform the most competitive baselines on SOP (11318 classes) under all metrics, it achieves comparable results with a much less computational cost. Specifically, MS [24] adopts a very large batch size 1000 (see its appendix) to achieve its best performance, which is difficult for us to reproduce even using four GPUs, each with 12 GB memory. Proxy-Anchor (CVPR 2020) also requires a large batch size 180 and calculates each sample with all 11318 proxies; SoftTriple [15] employs two parallel FC layers to classify 11318 classes. In contrast, our method only needs to calculate and update the gradients of $\lceil 0.05 \times 11318 \times 1 \rceil$ (see Eq. (5)) proxies for each sample during back-propagation, and we use a small batch size 32 for *inheriting the advantage* of original ProxyNCA. As future work, we will focus more on addressing such datasets with huge inter-class variance.

[R5] **Backbone and parameters.** We use the same backbone as in MS [24] and SoftTriple [15] — Inception pretrained on ImageNet; actually we strictly follow the setting of SoftTriple, *i.e.*, data pre-processing, backbone (followed by an average pooling layer), optimizer, *etc*. We will follow the suggestion to indicate the backbones used by other methods. Regarding the concern about whether the performance improvement is brought by additional parameters in the graphs, actually only proxies are trainable parameters — we do not involve any extra conv, FC layers or other forms of parameters. In fact, our method contains a similar number of parameters to SoftTriple while performing better in most cases; besides, only $\frac{1}{20}$ of these parameters are adaptively calculated and updated during each back-propagation. Therefore, we confirm that our performance improvement is not caused by switch in backbone or increased parameters.

[R5] **More comparisons.** Following the suggestion, we compare our ProxyGML with the mentioned Proxy-Anchor (CVPR 2020) using its official code, and the Recall@1 results are shown in the table. In particular, we have found that Proxy-Anchor relies on a large batch size, and is implemented with three additional engineering skills, *i.e.*, 1) a combination of an average- and a max- pooling layer following the Inception backbone, 2) a warm-up strategy for stabilizing proxy learning, and 3) an AdamW optimizer instead of the original Adam. For fair comparison, we evaluate Proxy-Anchor under our setting — with batch size 32 and the three engineering skills removed; it is also evaluated with the three skills enabled (indicated by "*"), and with its optimal batch size 180. Since the time does not allow any further tuning for Proxy-Anchor, we report here the result with batch size 30 (also the skills are used) provided in its paper for reference. We note that this is only a preliminary experiment, and in the revised version we will further involve more experiments of Proxy-Anchor with careful tuning, and ProxyGML with large batch size and the three skills added. Still, we can infer from the table the advantage of our ProxyGML against Proxy-Anchor. Considering that the mentioned EDMS (CVPR 2019) is practically an extension of N-pair loss with heavy backbone ensembles and has no code released, we do not involve it for comparison (same as Proxy-Anchor), but will add it for discussion. Also, we will tone down the claims of being the first to use graph classification for DML and clarify the difference between the mentioned SCDMLGE (WWW 2020) and our ProxyGML.

Method	CUB	Cars196	SOP
ProxyGML$_{32}$	**66.6**	**85.5**	**78.0**
Proxy-Anchor$_{32}$	35.8	20.3	41.4
Proxy-Anchor$^{*}_{32}$	65.4	83.1	75.7
Proxy-Anchor$_{180}$	66.1	84.2	54.5
Proxy-Anchor$^{*}_{30}$	65.9	84.6	76.0