LPOSS: Label Propagation Over Patches and Pixels for Open-vocabulary Semantic Segmentation

Supplementary Material

6. Analysis of performance

6.1. Quantitative analysis per image

In Figure 5a and Figure 5b, we present the comparison of mIoU and Boundary IoU performance of LPOSS and LPOSS+. We observe that LPOSS+ consistently improves the performance of LPOSS with respect to both metrics.

Figure 5c presents the comparison of mIoU and Boundary IoU performance of LPOSS+. We see that although the two metrics are correlated to some extent, they are still complementary to each other. Boundary IoU performance is usually lower than mIoU performance, which is consistent with the definition that Boundary IoU measures the fine-grained performance at segment borders, which consist of the hardest pixels to classify.

We present the comparison of the mIoU performance of the oracle experiment and LPOSS in Figure 5d. We observe that oracle performance varies a lot across images, which we attribute to the differences in object size and shape. Additionally, the oracle performance acts as an upper bound in the majority of cases. Exceptions are justified by the effect of bilinear interpolation and combining predictions across many windows.

6.2. Per image/class comparison with MaskCLIP

LPOSS refines MaskCLIP predictions, so we look at how successful LPOSS is in improving these predictions. Figure 6 shows the comparison of the mIoU for LPOSS and MaskCLIP on the image and class level. We observe that LPOSS successfully improves MaskCLIP predictions in the vast majority of cases. Rarely, an image, with already very low performance, is further harmed if the MaskCLIP output is spatially very noisy, as shown in Figure 7.

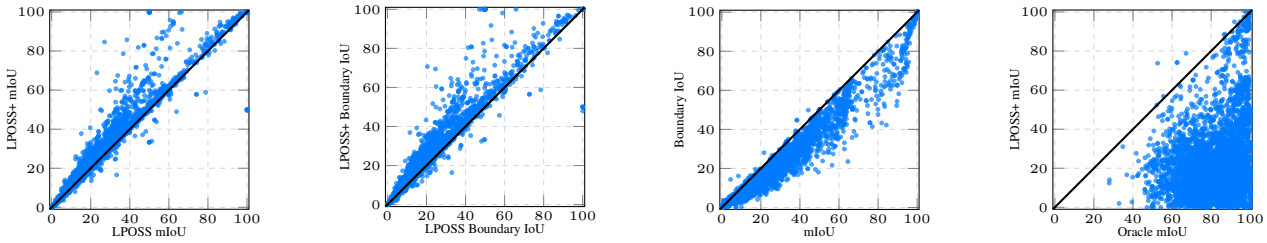
6.3. The impact of window size

Furthermore, we look into the impact of window size on CLIP-DINOiser and LPOSS. In Figure 8, we visualize the results of both methods as well as MaskCLIP using two different window sizes. We observe that there are some cases when MaskCLIP has better output for the large windows, used by CLIP-DINOiser, vs the small ones, used by LPOSS. In these cases, CLIP-DINOiser outperforms LPOSS. We also observe that there are cases where LPOSS performs well for both large and small window sizes, while CLIP-DINOiser performs better for the larger window size. Based on this, we propose to run our methods using an ensemble of large and small window sizes, and observe that this further improves the performance of LPOSS and LPOSS+, as presented in Section 4.3.

7. Ablation study

Design of S_a and S_p. We construct S_a as a k-nearest neighbor graph with edge weights in the form s^γ, as it is a common choice for the adjacency matrix in the label propagation literature [21, 38]. Another choice could be $\exp\left(-\frac{1-s}{\sigma}\right)$, which we found to perform slightly worse for LPOSS (-0.1%). For S_p, we chose the RBF kernel as in CRFs and bilateral filters. We further tested a linear kernel, which performs a bit worse (-0.2%).

LPOSS and LPOSS+ construct adjacency matrices S_{P} and $S_{\tilde{\text{P}}}$ in a different fashion. The primary difference comes from the way they control the sparsity of the graph. LPOSS uses k-nearest neighbors on top of the appearance-based adjacency S_a. On the other hand, LPOSS+ controls the sparsity using a binary spatial affinity $S_{\tilde{p}}$ that has non-zero elements only within the $r \times r$ neighborhood of the pixel. This difference is motivated by two reasons. First,



(a) mIoU comparison of LPOSS and LPOSS+ (b) Boundary IoU comparison of LPOSS and LPOSS+ (c) Comparison between mIoU and Boundary IoU for LPOSS+ (d) mIoU comparison of Oracle (patch-level res.) and LPOSS

Figure 5. **Analysis of LPOSS and LPOSS+ performance** per image. The plot shows 5000 randomly selected test images from all eight datasets used in the paper.

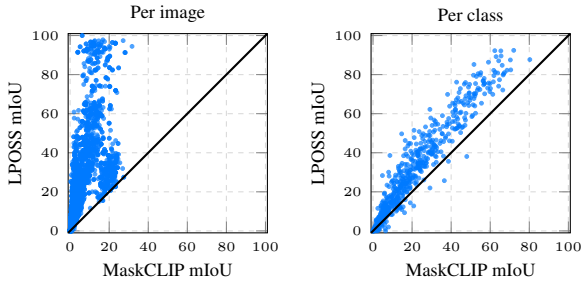


Figure 6. **Comparison of MaskCLIP and LPOSS performance** per image and per class. The plot shows 5000 randomly selected test images and all classes from all eight datasets in our experiments.

with LPOSS+, we aim to refine the predictions around the boundaries, which can be accomplished by looking at the neighborhood of each pixel. Second, the color-based features used in LPOSS+ can be very noisy, which can create issues for k-nearest neighbor search. To validate this, we try implementing LPOSS+ using the functions of LPOSS. However, we have found that this makes LPOSS+ ineffective and actually produces performance results worse than LPOSS. Additionally, we also experiment with using an RBF kernel in $S_{\tilde{p}}$ and find that it performs a bit worse than

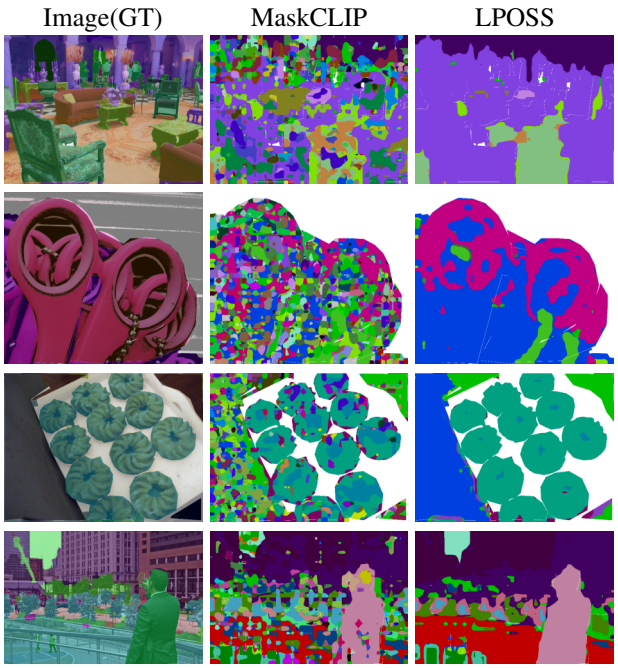


Figure 7. **Examples of the LPOSS failure cases that are due to the MaskCLIP noisy predictions.** MaskCLIP produces very spatially noisy predictions in some cases, which then translates to the bad performance of LPOSS. Pixels shown in white are pixels that do not have a class in the ground-truth.

Method	S_a	S_p	Coupled feat. extraction and prediction	Pix. feature	Avg
LPOSS	DINO	✓	✗	-	**41.3**
LPOSS	CLIP	✓	✗	-	38.3
LPOSS	DINO	✗	✗	-	40.6
LPOSS	DINO	✓	✓	-	39.2
LPOSS+	DINO	✓	✗	Lab	42.1
LPOSS+	CLIP	✓	✗	Lab	39.0
LPOSS+	DINO	✗	✗	Lab	41.4
LPOSS+	DINO	✓	✓	Lab	39.8
LPOSS+	DINO	✓	✗	depth	**42.2**
LPOSS+	DINO	✓	✗	Lab+depth	**42.2**

Table 4. **Ablations for LPOSS and LPOSS+.** We report mIoU averaged across 8 datasets. Default setups used in the paper are marked with .

the proposed method (achieving the same performance as the proposed method when the RBF kernel converges to the proposed binary values).

Features used in adjacency S_a. We replace DINO features for the construction of S_a with CLIP features and observe a significant drop in performance for both proposed methods, as shown in Table 4.

Impact of spatial adjacency S_p. We remove the use of S_p in the construction of S_{P} , *i.e.* set it to a matrix full of ones, and observe, as shown in Table 4, that using S_p improves the results both for LPOSS and LPOSS+. Additionally, in Figure 9, we visualize the impact of S_p on the predictions. We observe that the use of S_p cleans the predictions.

Impact of window-based predictions. We perform an experiment where we do feature extraction per window, as always, but also prediction per sliding window, as other methods do too [23, 26, 46]. We observe, as shown in Table 4 that our choice of performing prediction jointly across all windows is indeed beneficial to the performance of LPOSS and LPOSS+.

Pixel features. We switch pixel features from the default choice of Lab color space to depth predictions from DepthAnythingV2 [49] and their combination. We observe, as shown in Table 4, that using predicted depth marginally improves the results, but we opt not to use it in our default setup as it introduces another model during inference. We conclude that the use of different pixel-level features is worth future exploration.

Impact of hyper-parameters k, σ, and r. In Figure 10, we show the impact of hyper-parameters k, σ, and r on the performance. Good performance is achieved over a wide range of values. For LPOSS+, r controls the performance/speed trade-off (via sparsity), and we found $r = 13$ to be a good compromise.

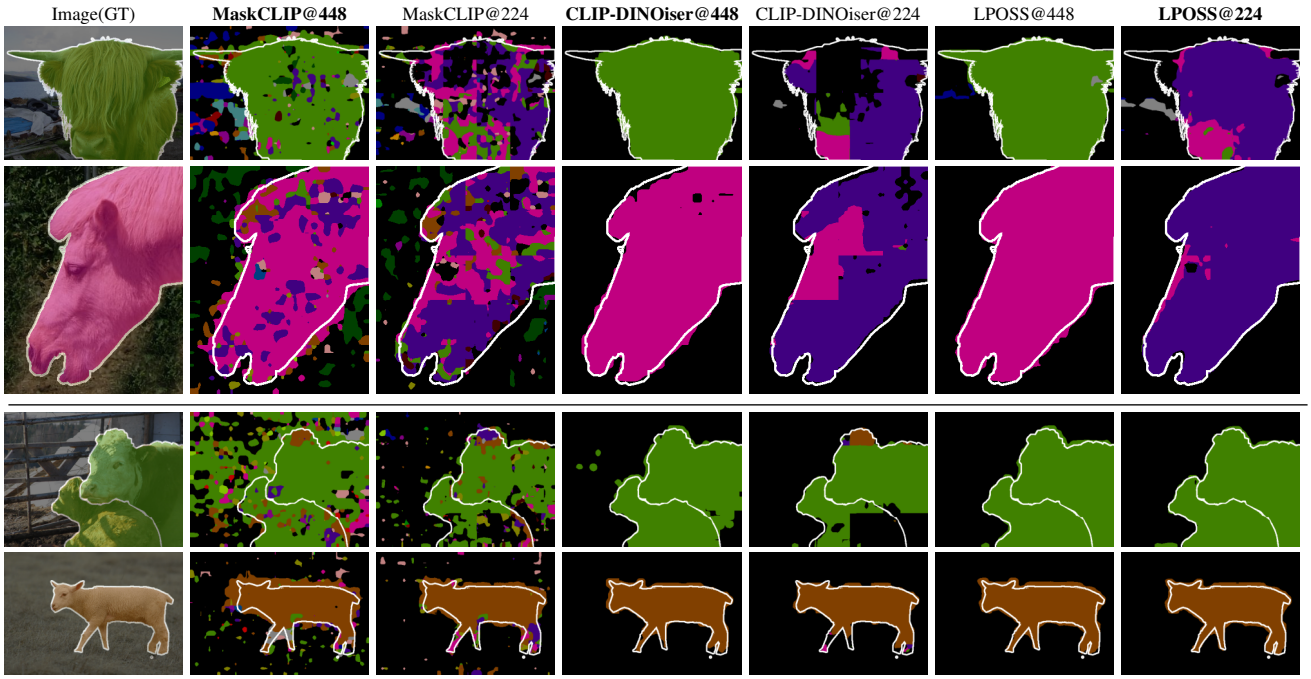


Figure 8. **Impact of the window size on different methods.** The top rows show examples where both CLIP-DINOiser and LPOSS benefit from the large window size. The bottom rows show examples where LPOSS works well for both window sizes while CLIP-DINOiser performs better for the larger window size. Pixels shown in white are pixels that do not have a class in the ground-truth. Default setup of each method is shown in **bold**.

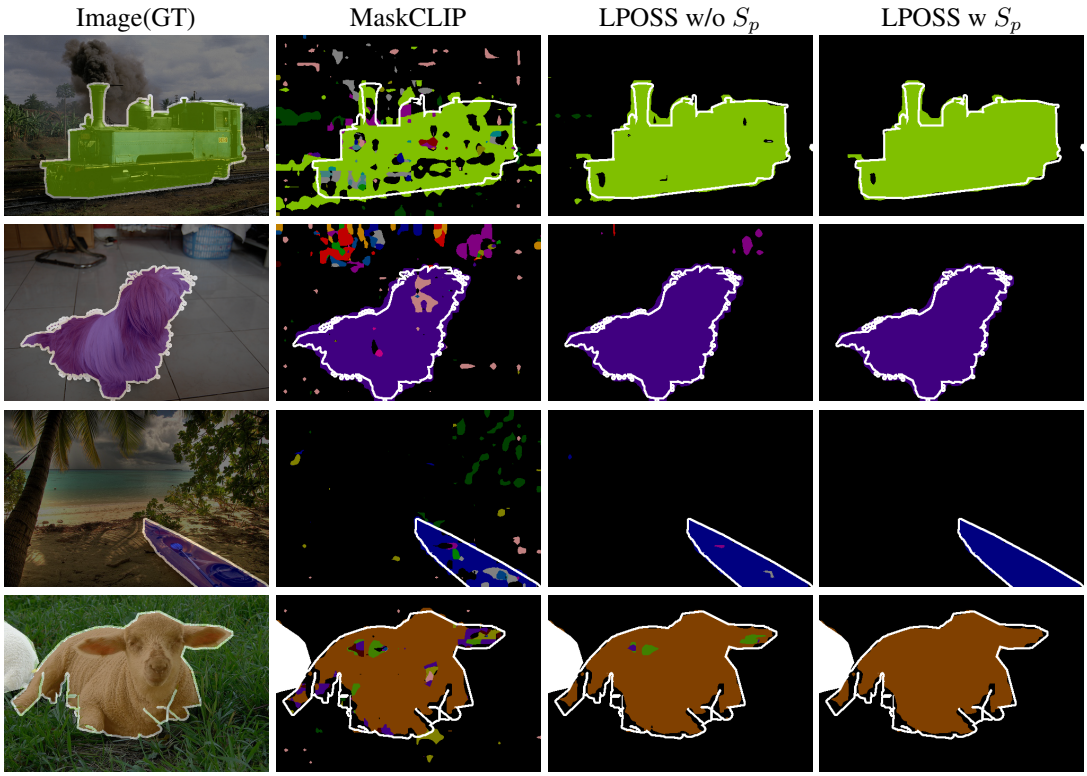


Figure 9. **Impact of the spatial adjacency S_p.** A comparison of segmentation maps of LPOSS when applied without or with S_p. Pixels shown in white are pixels that do not have a class in the ground-truth.

8. Additional results

8.1. Method comparison using a different VM.

Throughout the paper, we use DINO [5] with ViT-B/16 backbone as a VM. However, here we show that LPOSS and LPOSS+ can be applied with other VM backbones as well. Concretely, we use DINO [5] with ViT-B/8 backbone and DINOv2 [5] with ViT-B/14 backbone. Considering that the VM now uses a patch size different from that of a VLM, the feature vectors coming from the VM and VLM are of a different size. To apply LPOSS in such a situation, we upsample the VLM features to match the size of VM features.

We present the results of this experiment in Table 5 and compare the results of LPOSS and LPOSS+ with ProxyCLIP [26] and LaVG [23], which also report the performance of such experiments.

DINO with ViT-B/8. For DINO with ViT-B/8, we observe that both LPOSS and LPOSS+ outperform LaVG, while ProxyCLIP outperforms LPOSS and performs on par with LPOSS+. However, ProxyCLIP again significantly outperforms LPOSS and LPOSS+ only on Object and VOC20 datasets, for which we provide an explanation in Section 4.3. We also observe that going to pixel-level predictions in LPOSS+ improves the results even when the patch size is as small as 8.

Additionally, we observe that after using ViT-B/8 backbone for the VM, the graph defined by $S^{(\mathcal{W})}$ in (8) has four times as many nodes compared to the default setup of using ViT-B/16 backbone, with the increase coming from the fact that there are four times more feature vectors for each sliding-window. So we propose to just increase the value of the hyper-parameter k from 400 to 800, while keeping all other hyper-parameters fixed, to allow better connectivity between nodes coming from different sliding windows. With this change, we observe that LPOSS performs on par with ProxyCLIP, while LPOSS+ outperforms it, as shown in Table 5.

DINOv2 with ViT-B/14. For DINOv2, we observe that LPOSS and LPOSS+ perform slightly worse than for the case of using DINO with ViT-B/16. However, LPOSS+ still outperforms ProxyCLIP, while LPOSS performs on par with it. We note that due to the very different distribution of similarities coming from DINOv2, compared with DINO, we use a value of $\gamma = 7.0$ for LPOSS and LPOSS+.

8.2. Complementarity with other methods.

We build LPOSS and LPOSS+ by applying them on top of initial predictions coming from the VLM; in particular as MaskCLIP computes them. However, these initial predictions can come from any model, so here we show that we can apply them on top of CLIP-DINOiser [46], by simply using Y_{DINOiser} instead of Y_{vlm}. Compared to the main paper experiments, we also use the sliding window setup used in CLIP-DINOiser, as it significantly improves CLIP-DINOiser performance as shown in Table 3.

We present the results of these experiments in Table 6. We observe that LPOSS and LPOSS+ are complementary to CLIP-DINOiser and that they can further improve its performance. Additionally, we show that using a better initialization of CLIP-DINOiser compared to MaskCLIP improves LPOSS and LPOSS+ results as well.

8.3. LPOSS+ *vs*. other post-processing methods.

We compare LPOSS+ with two other post-processing pixel refinement methods, PAMR [1] and DenseCRF [24] by applying them on top of LPOSS predictions. PAMR, with 5 iterations and dilations 8 and 16, achieves 41.8 mIoU (*vs*. 42.1 of LPOSS+) while DenseCRF was unable to improve LPOSS at all.

9. Computation requirements

We measure the average time necessary to perform the inference per image on the VOC20 dataset using an NVIDIA A100 GPU. LPOSS processes each image in under 0.1s, which is comparable to all methods except LAVG (6.5s). LPOSS+ for pixel-level post-processing takes 0.5s/image, comparable to or faster than pixel-level post-processing methods PAMR [1] (0.5s) or DenseCRF [24] (1.6sec). Note that LPOSS+ speed can be controlled via hyper-parameter r (see Figure 10).

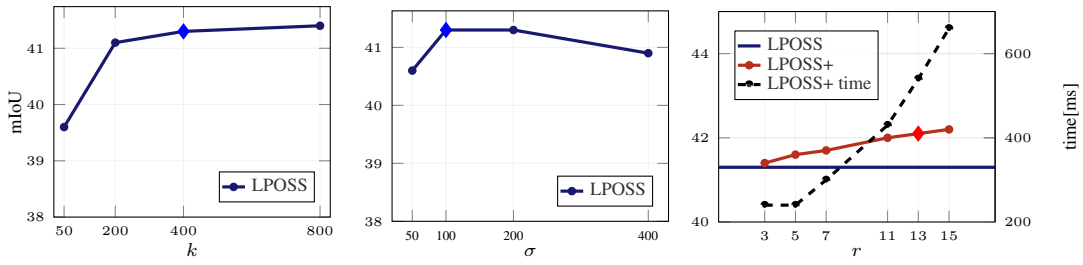


Figure 10. **Impact of hyper-parameters k, σ, and r.** We report mIoU averaged across 8 datasets. Default setups in the paper are marked with ♦.

Method	VM	VOC	Object	Context	C59	Stuff	VOC20	ADE20k	City	Avg
ProxyCLIP* [26]	DINO(ViT-B/16)	59.3	36.3	34.4	38.0	25.7	79.7	19.4	36.0	41.1
LaVG* [23]	DINO(ViT-B/16)	61.8	33.3	31.5	34.6	22.8	81.9	14.8	25.0	38.2
LPOSS	DINO(ViT-B/16)	61.1	33.4	34.6	37.8	25.9	78.8	21.8	37.3	41.3
LPOSS+	DINO(ViT-B/16)	62.4	34.3	35.4	38.6	26.5	79.3	22.3	37.9	42.1
ProxyCLIP [26]	DINO(ViT-B/8)	61.3	**37.5**	35.3	**39.1**	26.5	80.3	20.2	38.1	42.3
LaVG [23]	DINO(ViT-B/8)	62.1	34.2	31.6	34.7	23.2	**82.5**	15.8	26.2	38.8
LPOSS	DINO(ViT-B/8)	61.4	33.5	34.9	38.2	26.3	77.6	22.6	**40.2**	41.8
LPOSS ($k = 800$)	DINO(ViT-B/8)	62.2	34.1	35.2	38.5	26.4	78.7	22.5	39.4	42.1
LPOSS+	DINO(ViT-B/8)	62.2	34.2	35.5	38.9	**26.8**	78.0	**23.0**	**40.2**	42.3
LPOSS+ ($k = 800$)	DINO(ViT-B/8)	**63.0**	34.8	**35.8**	**39.1**	**26.8**	79.0	22.8	39.3	**42.6**
ProxyCLIP [26]	DINOv2(ViT-B/14)	58.6	**37.4**	33.8	37.2	25.4	**83.0**	19.7	33.9	41.1
LPOSS ($\gamma = 7.0$)	DINOv2(ViT-B/14)	59.7	33.3	34.3	37.5	25.6	80.0	21.9	36.0	41.0
LPOSS+ ($\gamma = 7.0$)	DINOv2(ViT-B/14)	**60.8**	34.3	**35.1**	**38.3**	**26.2**	80.4	**22.4**	**36.7**	**41.8**

Table 5. **Performance comparison in terms of mIoU on 8 datasets** using ViT-B/16 backbone for VLM and DINO ViT-B/8, DINO ViT-B/16, or DINOv2 ViT-B/14 backbone for VM. Default setups of hyper-parameters used in the paper are marked with . * denotes the methods for which we reproduce the performance.

Method	VOC	Object	Context	C59	Stuff	VOC20	ADE20k	City	Avg
MaskCLIP* [53]	32.9	16.3	22.9	25.5	17.5	61.8	14.2	25.0	27.0
LPOSS (MaskCLIP)	61.1	32.5	32.9	36.3	25.2	82.8	20.4	31.7	40.4
LPOSS+ (MaskCLIP)	61.8	33.2	**33.4**	36.9	**25.6**	83.1	**20.7**	**31.9**	40.8
CLIP-DINOiser* [46]	62.2	34.7	32.5	36.0	24.6	80.8	20.1	31.1	40.2
LPOSS (CLIP-DINOiser)	65.0	36.3	32.9	36.6	25.1	84.0	19.7	29.3	41.1
LPOSS+ (CLIP-DINOiser)	**66.1**	**36.8**	**33.4**	**37.2**	25.4	**84.2**	19.9	29.5	**41.6**

Table 6. **Performance comparison in terms of mIoU on 8 datasets** applying LPOSS and LPOSS+ on top of MaskCLIP (default choice in the main paper) or CLIP-DINOiser [46]. ViT-B/16 backbone is used both for VLM and VM. Sliding-window size 448×448 and stride 224×224 as used in CLIP-DINOiser. * denotes the methods for which we reproduce the performance.