Learning Propagation for Arbitrarily-structured Data
Supplementary Material

Sifei Liu[1], Xueting Li[1,2], Varun Jampani[1], Shalini De Mello[1], Jan Kautz[1]
[1]NVIDIA, [2]University of California, Merced

In this supplementary material, we provide additional details that we could not include in the paper due to a limitation on the number of pages. These include the a mathematical proof of SGPN being equivalent to linear diffusion on graphs (Section 1), the computational and experimental settings, and run-times (Section 2), the method for re-organization of vertices (Section 2.2), processing of point clouds (Section 2.3), and additional experimental results (Section 3).

1. Diffusion on Graphs: Detailed Proof

As described in the paper, with a parallel formulation, we denote the corresponding "unary" and "propagated" features for the vertices in the p^{th} group, before and after propagation as u_p and h_p, respectively. We perform propagation for each group as a linear combination of all its previous groups:

$$h_p = (I - d_p)u_p + \sum_{q=1}^{p-1} w_{pq} h_q. \tag{1}$$

Also from the paper $H = [h_1, ..., h_T] \in \mathbb{R}^{N \times c}, U = [u_1, ..., u_T] \in \mathbb{R}^{N \times c}$ are the features of all groups concatenated together. By expanding the second term of Eq. (1), the propagation can be re-written as refining the feature U through a global linear transformation $H - U = -LU$. To meet the requirement of a linear diffusion process, L should be a Laplacian matrix with the summation of the entries of each row equal to zero. Suppose we reduce Eq. (1) to one-step (denoted as $t = 1$) propagation where h_p only links to h_{p-1}. Then the propagation is similar to the form in [4]:

$$H = (I - D_t + A_t)U, \quad t \in \{1\}, \tag{2}$$

where D is a degree matrix for the entire feature set, which places all $\{d\}$ along its diagonal and A is the affinity matrix which expands the $\{w\}$ and describes the affinities among all pairs of vertices. We use t to denote time steps.

By rewriting Eq. (3) in the paper (see the definition of m_q in the paper), the degree matrix between two adjacent groups can be regarded as the summation of the individual degree matrix for different time steps (we use $x(:)$ to denote all the entries in x):

$$d_p = \sum_{q=1}^{p-1} \sum_{j=1}^{m_q} w_{pq}(:, j) = \sum_{q=1}^{p-1} d_{pq}. \tag{3}$$

Correspondingly, by replacing q with t to represent the global affinity matrix of different steps, the matrix formulation is:

$$D = \textstyle\sum_t D_t, \quad t \in \{1, 2, 3, ...\}, \quad where \quad D_t E = A_t E. \tag{4}$$

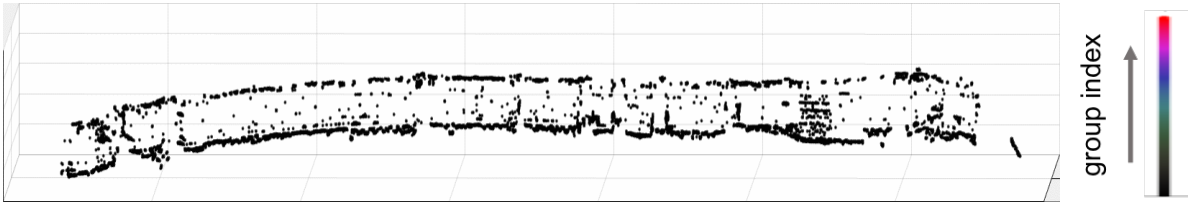


Figure 1. Click to play the animation: the dynamics of the propagation process (from the 1^{st} to the n^{th} group) on a bottom-to-top DAG on the point cloud for buildings, in the RueMonge [7] dataset. The points are ordered into 1136 groups in this case, on a global scale.

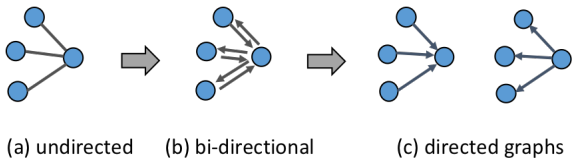


Figure 2. We construct the DAGs by: (a) establishing an undirected graph; (b) taking (a) as bi-directional graph, and (c) decomposing it into several directed graphs.

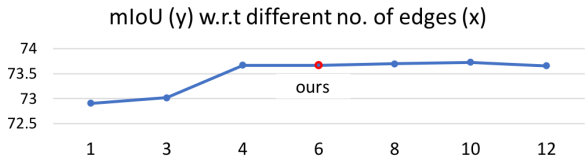


Figure 3. Perturbing the number of edges (axis-x) for each vertex in the constructed DAGs. Performance is shown using the *+PF+PG/TG* model (see Tabel.2 in the main paper).

The term $A_t E$ is the expansion of $\sum_{j=1}^{m_q} w_{pq}(:, j)$. Given Eq. (4), we can re-formulate Eq. (1) as the summation of multiple time steps:

$$H = (I - \sum_t D_t + \sum_t A_t)U, \quad t \in \{1, 2, 3, ...\}, \tag{5}$$

where $L = \sum_t (D_t - A_t) = \sum_t L_t$. Consequently for each t, we have $L_t E = (D_t - A_t)E$, such that L_t has each row sum to zero. Therefore, we can draw the conclusion that L is a Laplacian matrix and that propagation via our proposed SGPN module represents a standard diffusion process on graphs.

2. More Implementation Details

We provide more details about our implementation of the experiments presented in the paper, including the computational settings and run-times (Section 2.1), re-organization of arbitrarily-structured data (Section 2.2), the detailed training and testing procedures for point clouds (Section 2.3), as well as the network architectures (Section 2.4).

2.1. Computational Settings and Run-times

Computation. We implement all our experiments on servers equipped with four 16GB Titan V100 GPUs. All scripts are implemented with the PyTorch framework[1] and will be made available to the public upon publication.

Training. Specifically, we utilize the data parallel module of PyTorch to accelerate training and testing, but we **do not** use any multi-GPU batch normalization techniques even for the Deeplab-related models, in order to maintain a basic setting so that we can focus on the capability of, and the improvement that can be brought by SGPN. We train all hybrid models, *i.e.*, CNN block+SGPN jointly with weights initialized for the CNN from ImageNet. The only exception to this is the experiment with NLNN [8], where better performance is achieved when the CNN is initialized with weights from the baseline backbone segmentation network (see Table 1 in the paper). We use most of the hyper-parameter values that were originally proposed to train the baseline networks (*e.g.*, learning rate, weight decay, etc.). In addition, we use the SGD and Adam solvers for the DRN and Deeplab-related networks, respectively.

Run-times. The inference time of one image (1024×2048) from the Cityscapes dataset is 0.2 and 0.5 second for DRN-based SGPN on pixels (Section 5.3) and superpixels (Section 5.4), respectively (on a V100 Nvidia GPU). It takes approximately two minutes to carry out inference with our hybrid networks (*e.g.*, the CNN block and the SGPN on the entire point cloud in Fig. 4), for all the images and points in the validation set.

2.2. Discussion about DAGs

DAG Construction. For constructing DAGs, we first search for the K-nearest neighbors of each point (please refer the next paragraph for more details), then re-organize them into different groups for parallel computation (see Section 3.1 in the paper). Re-organization of vertices in a DAG can be easily carried out via a modified topological sorting algorithm 1. We use p to denote the index of groups, as described in Section 3.1 in the paper and in Eq. (1). Specifically, finding the neighbors of superpixels can be conducted by searching for the adjacent superpixels that have shared boundaries. Note that since the superpixels are irregular, it can sometimes happen that two superpixel's center overlap. This will cause there to be directed cycles in the directed graphs. For such situations, we assign random small offsets to such overlapping centers to avoid this degnerate case. For a point cloud, we fix the number of nearest neighbor to $K = 6$ for all experiments presented in the paper. After constructing the DAGs, the linear propagation in Eq. (1) is conducted along a constructed DAG, *e.g.*, from the first group to the last one in the bottom-to-top DAG, as shown in Fig. 1.

[1] https://pytorch.org/

Structures of the DAGs: a symmetric affinity matrix. A undirected graph structure is preferable to eliminate any structural biases. Although we propose the directed graphs, we do ensure that the linear propagation on DAGs equal to a symmetric affinity. Ideally, an undirected graph should be established and a single weight is attached to each edge. We ensure the symmetry in two aspects: (a). *The graph is bi-directional*: we first construct an undirected graph by finding the nearest neighbors for each element while maintaining an symmetric adjacency matrix (Fig. 9 (a)). Then, we take it as a bi-directional graph and decompose it into several directed graphs w.r.t different directions (Fig. 9 (b)(c)). (b). *The weight is symmetric*: Each pair of bi-directional links share a single weight since the kernels are symmetric, i.e., $\kappa(x_i, x_j) = \kappa(x_j, x_i)$. See Section 3.2 in the paper. In addition, we consider all possible directions (i.e., 4 for 2D and 6 for 3D domain) to ensure the bi-directional graph is fully utilized by the propagation module.

Algorithm 1. Re-ordering of vertices in a DAG

G: the DAG to be re-ordered.
L: an empty list that will contain the groups for all the "time-steps".
S: the parent nodes without any incoming edges.
$p = 1$ denotes the index of the group, starting from 1.
top:
if L collected all the vertices **then return**
loop:
remove S from G
$L(p) \leftarrow S$ (Note that $L(p)$ is a sub-list of L.)
for each node m with edges e coming from S **do**
    remove e from G
update S based on G, $p = p + 1$
goto *top*.

Different choices of established DAGs. We conduct ablation studies for different choices of DAGs in two aspects: (a) changing the directions on the construction of a DAG, and (b) perturbing the number of connections for each nodes. We note that since we proposed a factorized edge representation, our network is quite flexible to be directly tested on the above settings, without re-training. We show that the performance is not sensitive to the variants of DAGs, which also validates that it is similar to an undirected graph.

(a). *Changing the directions.* On the RueMonge2014 dataset, we rotate the point cloud globally in the XY plane with an angle of 20 degrees, and re-establish the DAGs. The neighbors remain the same for each point, but the directions for propagation are different (i.e., rotating the axis-X for propagation with 20 degrees). With the *+PF+PG/TG* model, we obtain similar mIoU (73.32%) as the original one (73.66%).

(b). *Perturbing the number of edges.* This study is introduced in l-(201-203) in the supplementary material. In detail, we perturb the number of neighbors K for each vertex in a DAG from 1 to 12. We obtain similar performance especially when $K > 3$. See Fig. 3.

2.3. Processing of point clouds

The training and testing procedures for point clouds in this work are slightly different from that of superpixels. We alluded to them at a high-level in the paper, but here, we specify them in more detail and provide more intuition.

How to augment for propagation on graphs? Augmenting 3D points is usually conducted by extending data augmentation techniques used to augment image patches to 3D space [3, 5, 6]. In our work, however, this is made even simpler by the fact that all the augmentations can be conducted simply on 2D space – the cropped image patches, while training a CNN with images. Since we maintain an index map of the same size as an input image, all augmentations conducted on images can be simultaneously applied to their index maps. The augmented points are associated with these augmented index maps.

Propagation on a global scope. During training, we utilize multiple GPUs where each one takes a mini-batch of image patches, aggregates image features according to the indices of the subset of *local* points corresponding to those patches, and conducts propagation and MLP before computing the loss w.r.t the ground-truth labels. However, the testing procedure for a

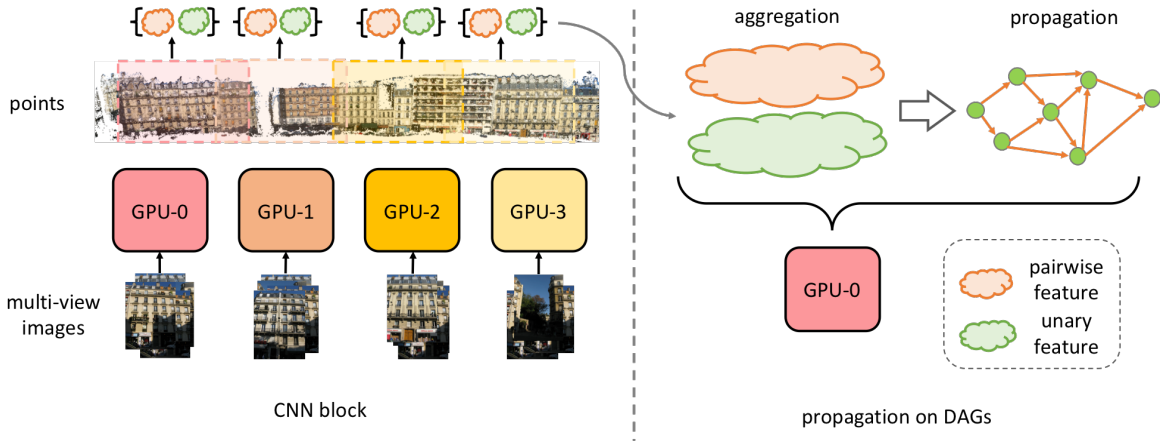


Figure 4. Inference of multi-view images + an entire point cloud (we use the cloud-shaped blobs to present the the aggregated features for the 3D points). CNN-block: it infers all the multi-view images in parallel, and aggregates the unary and pairwise features on each GPU independently. Note that points between images and across GPUs are highly overlapped. Aggregation (across GPUs): the unary and pairwise features from each GPUs are re-aggregated across GPUs according to their indices, in order to avoid duplication. Propagation on DAGs: The SPN module is run on a single GPU.

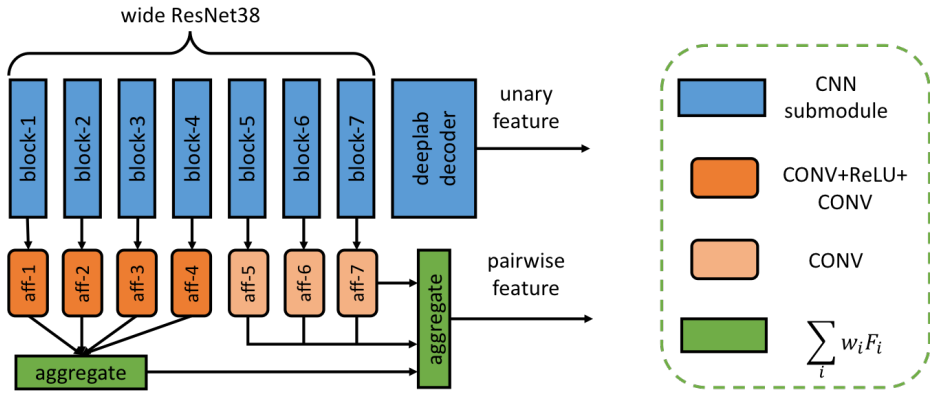


Figure 5. CNN block for the Deeplab network-based [1] SGPN.

point cloud is different from its training procedure, in the sense that while inference of individual images is distributed across different GPUs, the propagation is conducted *globally* on the entire point cloud, by collecting information across the GPUs. In contrast, if we were to test/process each image along with its corresponding set of points individually, propagation would be limited to only a local set of points. To verify this, we test each image individually, and for every 3D point aggregate DRN results of all pixels across all images that map to it, as opposed to processing the images collectively and propagating information globally in the point cloud (*image to points* in Table 2 in the paper), and the mean IoU drops from 69.16 to 65.50. Clearly, aggregating and propagating on a global level is of great importance for labeling points. Fortunately, the flexibility of our algorithm brought about by the factorization of the edge representation (Section 3.3), enables the algorithm to be implemented in a highly flexible, yet efficient manner as shown in Fig. 4.

In detail, we divide the inference phase into the stages of (a) inference of all the testing images, which can be done concurrently for each image on multiple GPUs (the CNN block in Fig. 4), and (b) global aggregation of all the unary and pairwise features from all images and mapping them to the entire point cloud, which can be done on a single GPU (the propagation on DAGs in Fig. 4). We show our implementation in Fig. 4, where the CNN block is configured to perform inference on the images in parallel (*e.g.*, on 4 GPUs), and the propagation (right) is done globally, after collecting the feature of all the images from multiple GPUs on to a single GPU, and aggregating them for the entire point cloud.

2.4. Network Architecture of the CNN Block

Here we show the architecture of a Deeplab-based CNN block, as discussed in Section 5.3 in the paper, in Fig. 5. This is also a detailed version of the first module in Figure 4 of the paper. Specifically, the CNN sub-module contains a series of residual blocks [9]. The aggregation of different feature levels of the CNN is conducted with $\sum_i w_i F_i$, where w_i is a learnable, scalar parameter for level i initialized to 1, and F_i denotes the feature map from level i.

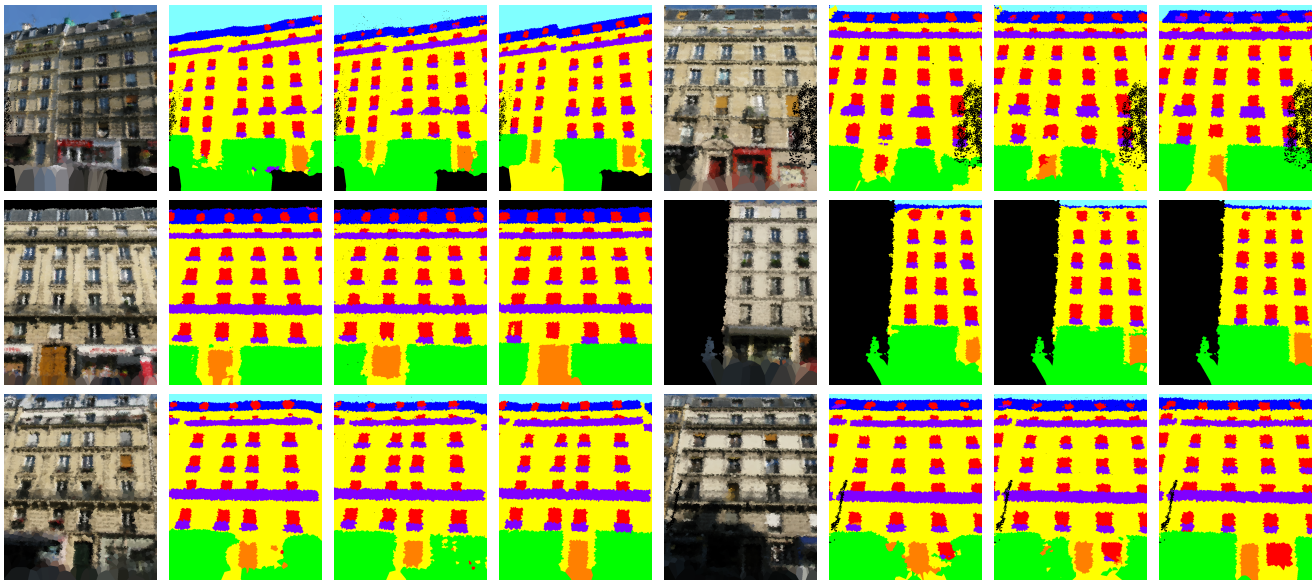
3. More Experimental Results

3.1. Semantic Segmentation

We present more quantitative results of semantic segmentation for the Cityscapes dataset in Table 1. In addition, we show more qualitative comparisons for the RueMonge [7] facade segmentation in Fig. 6 and 8. We also present the comparison, using the DRN-based network, between the SGPN on superpixels and pixels in Fig. 7 (c) and (d), respectively. Note that superpixel-based segmentation shows slightly different appearance due to the local pixel grouping operation.

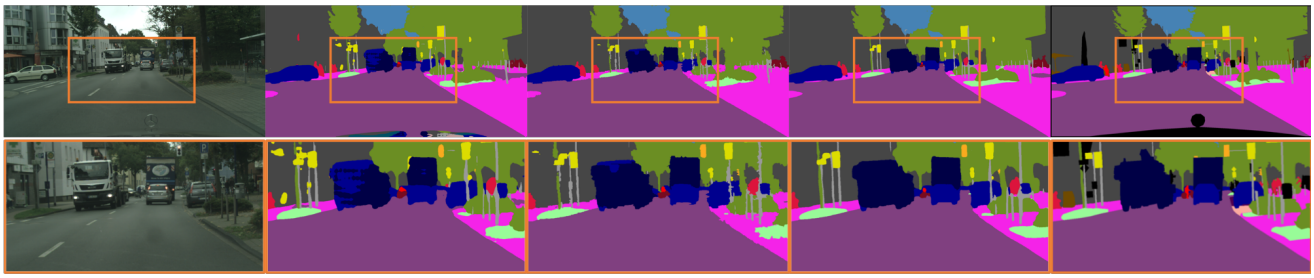
categories	road	sidewalk	building	wall	fence	pole	trafficlight	trafficsign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorcycle	bicycle	mIoU
Baseline DRN [10]	97.4	80.9	91.1	32.9	54.9	60.6	65.6	75.9	92.1	59.1	93.4	79.2	57.8	92.0	42.9	65.2	55.2	49.4	75.2	69.5
+SPN (single) [4]	97.7	82.7	91.3	34.4	54.6	61.8	65.9	76.4	92.2	62.2	94.4	78.5	56.8	92.7	47.8	70.2	63.6	52.0	75.3	71.1
+NLNN [8]	97.4	81.1	91.2	43.0	52.9	60.3	66.1	74.9	91.7	60.6	93.4	79.2	57.7	93.3	54.4	73.5	54.7	54.2	74.4	71.3
+SGPN-embed	98.0	83.8	92.2	48.5	**59.7**	64.1	70.1	79.4	92.6	**63.8**	**94.7**	82.0	60.7	94.9	62.5	77.7	51.1	**62.8**	**77.6**	74.5
+SGPN-prod	**98.1**	**84.4**	**92.2**	51.8	56.5	**65.8**	**71.2**	**79.4**	**92.7**	63.2	94.3	**82.7**	**65.1**	**94.9**	**73.8**	78.0	43.2	59.7	77.4	**75.0**
+SGPN-superpixels	97.6	82.4	91.0	**52.7**	52.9	58.4	66.1	75.9	91.8	62.2	93.6	79.4	58.2	93.3	62.4	**79.7**	57.1	60.2	75.1	73.2
Baseline Deeplab [1]	98.0	85.0	93.0	52.9	63.6	67.0	72.6	80.0	92.4	56.8	95.0	84.1	67.3	95.4	84.0	88.5	78.5	67.8	78.5	79.0
+SGPN-embed	**98.3**	**85.4**	93.1	**54.6**	62.0	69.3	74.0	**82.6**	**92.7**	63.4	95.0	85.0	69.2	95.9	**85.5**	89.3	85.3	69.5	78.0	80.4
+SGPN-prod	98.1	85.2	**93.3**	50.6	**65.8**	**70.7**	**74.9**	81.5	92.6	63.4	**95.2**	**85.1**	**69.3**	**96.0**	85.0	**91.8**	**86.7**	**71.8**	**80.1**	**80.9**

Table 1. The full results for **DRN**- and **Deeplab**-based networks on the Cityscapes validation set. We show multi-scale testing results except for the +SPN, which shows better performance in single scale setting. *embed* and *prod* denote the embedded Gaussian and inner product kernels. For superpixels, we use the embedded Gaussian kernel since it achieves significantly higher mIoU.



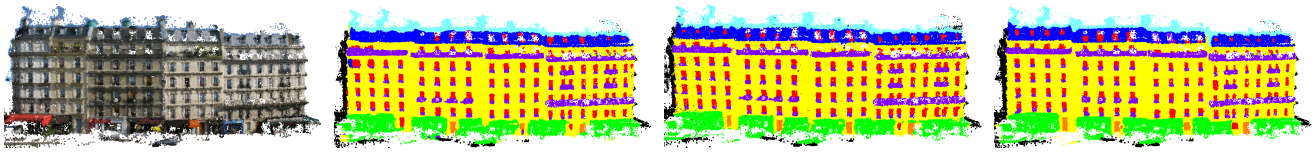
(a) input (b) DRN (c) +SGPN (d) GT (a) input (b) DRN (c) +SGPN (d) GT

Figure 6. A qualitative comparison of different methods for facade segmentation. The results are visualized by mapping the (a) diffuse RGB, (b) predicted and (c) gt label of points to pixels. DRN is the result of directly mapping from the DRN image's segmentation to a point cloud, and SGPN denotes our method (see Section 5.5 in the paper).



(a) input (b) DRN (c) +SGPN-superpixel (d) +SGPN-prod (e) GT

Figure 7. Qualitative comparisons of the Cityscapes semantic segmentation results. (a) DRN-D-22 (baseline); (b) DRN with SGPN on superpixels (see Section 5.4 in the paper); (c) DRN with SGPN on pixels with the inner product kernel (see Section 5.3 in the paper).



(a) image (b) DRN (c) +SGPN (d) GT

Figure 8. Visualized labeling results for (b) DRN as the baseline network and (c) our method (see Section 5.5 in the paper).

(a) 1% (b) 5% (c) 10% (d) 20% (e) ground truth

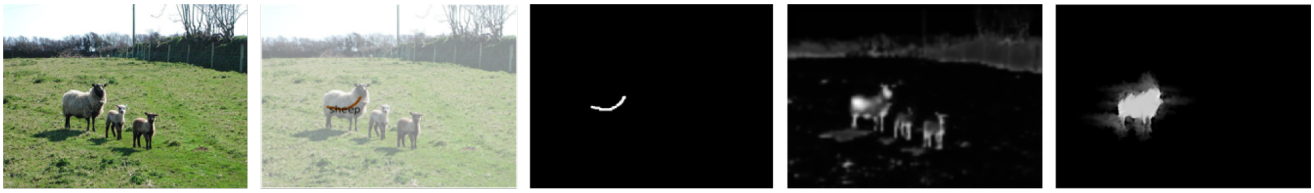
Figure 9. Color restorations with different ratios (a-d) of colored pixels retained from the original image (e).

3.2. Color Restoration and Scribble Propagation

Color restoration. We show that the SGPN itself is general and can be used for diffusing different type of properties, *e.g.* color, HDR and masks. Here we conduct *sparse color restoration* using SGPN: In the Lab color space, we randomly mask out pixels in the *ab* channels. The SGPN is used to propagate the color information from the sparse colored pixels to the non-colored regions purely based on the affinities learned from the L channel. We use a light-weight CNN and the dot-product kernel to compute strengths of the edges of our graph (the orange links in Fig. 4 in paper), while the unary feature (the green nodes in Fig. 4 in paper) is replaced by the sparse color pixels. We train a single network where we retain only $1\% \sim 3\%$ of randomly selected colored pixels during training. During testing, we show that the model can generalize well to different proportions of randomly selected colored pixels, e.g., $[1\%, 5\%, 10\%, 20\%]$, as shown in Fig. 9. The results reveal that the SGPN: (a) successfully learns general affinity and does not overfit to a certain ratio of the available colored pixels, and (b) the propagation is global so that very sparse colored pixels (e.g., 1% in Fig. 9 (a)) can "travel long distances" to fill-in the whole image with color.

Scribble propagation. In addition, we show that the learned affinity from the task of color restoration, can be directly used for propagation of other properties as well, such as scribbles. We show an example of this in Fig. 10, where a single scribble is drawn on one of the sheep in the image. With the same model trained that we trained for color restoration, but with a scribble mask as the "unary" feature to propagate (see Fig. 10 (c)) instead of the color channels (we duplicate the mask to fit to the input dimension of the network), our SGPN can faithfully propagate to spatial regions ranging in shape from a thin line, to the whole body of the sheep, while maintaining the fine structures and details, such as the legs of the sheep (see Fig. 10 (e)).

Additionally, we also train a color restoration network with NLNN [8], with a similar strategy: we append a non-local module to a CNN, which takes gray-scale images as input to learn the non-local affinity for color propagation. Different from the proposed spatial propagation module, in NLNN, the restored color map is obtained by directly multiplying the affinity



(a) image (b) draw a scribble (c) scribble mask (d) NLNN propagation (e) SGPN propagation

Figure 10. Scribble propagation via affinities learned for the task of color restoration. Different propagation modules are utilized, including (d) NLNN [8] and (e) SGPN as proposed in the paper.

matrix with the sparse color maps. We adopt the first 4 layers of ResNet18 [2] with pretrianed weights from ImageNet as the CNN (the light-weight CNN trained from scratch for our SGPN does not converge for NLNN) before the non-local module. Similar to SGPN, we directly apply the learned affinity to propagate the same scribble mask (see Fig. 10 (c)). Due to the non-local nature of the algorithm, the scribble is propagated globally to all the other regions with similar colors as the annotated sheep, which results in a much more noisy mask (see Fig. 10 (d)), compared with SGPN. This is also a real-world example revealing the importance of path-aware propagation, as we explained in Fig. 1 in the paper.

References

[1] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 4, 5

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 7

[3] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn: Convolution on x-transformed points. In *NIPS*, 2018. 3

[4] S. Liu, S. D. Mello, J. Gu, G. Zhong, M.-H. Yang, and J. Kautz. Learning affinity via spatial propagation networks. In *NIPS*, 2017. 1, 5

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 3

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 3

[7] H. Riemenschneider, A. Bodis-Szomoru, J. Weissenberg, and L. V. Gool. Learning where to classify in multi-view semantic segmentation. In *ECCV*, 2014. 1, 5

[8] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *CVPR*, 2018. 2, 5, 6, 7

[9] Z. Wu, C. Shen, and A. van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. arXiv:1611.10080, 2016. 5

[10] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *CVPR*, 2017. 5