RangeNet++
Fast and Accurate LiDAR Semantic Segmentation

IROS 2019

SqueezeSeg V1, V2

(i) The projection needs to be extended to include the full LiDAR scan, since the SqueezeSeg framework only uses the frontal 90 degrees of the scan, where the objects of the original KITTI dataset labels are annotated by bounding boxes.

(ii) The SqueezeNet backbone is not descriptive enough to infer all the 19 semantic classes provided by our dataset (SemanticKITTI).

(ii) We replace the CRF, which operates in the image domain by an efficient, GPU-based nearest neighbor search acting directly on the full, un-ordered point cloud.

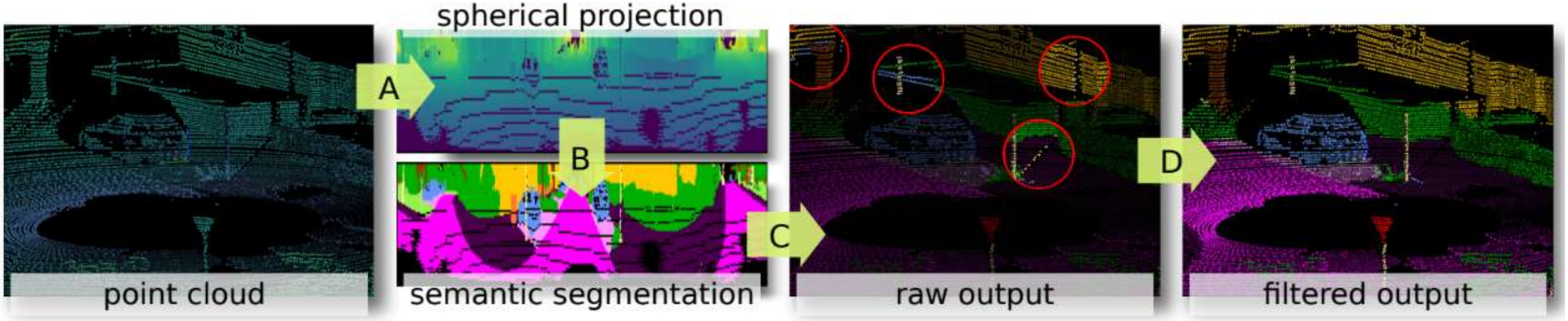


Fig. 2: Block diagram of the approach. Each of the arrows corresponds to one of our modules.

A. Range Image Point Cloud Proxy Representation

B. Fully Convolutional Semantic Segmentation

C. Point Cloud Reconstruction from Range Image

D. Efficient Point Cloud Post-processing

A. Range Image Point Cloud Proxy Representation

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left[1 - \arctan(y,x)\,\pi^{-1}\right] & w \\ \left[1 - \left(\arcsin(z\,r^{-1}) + \mathrm{f}_{\mathrm{up}}\right)\mathrm{f}^{-1}\right] & h \end{pmatrix}$$

where (u, v) are said image coordinates, (h, w) are the height and width of the desired range image representation, $\mathrm{f} = \mathrm{f}_{\mathrm{up}} + \mathrm{f}_{\mathrm{down}}$ is the vertical field-of-view of the sensor, and $r = ||\mathbf{p}_i||_2$ is the range of each point.

for each $\mathbf{p}_i$, its range *r*, its *x*, *y*, and *z* coordinates, and its *remission*, are stored in the image, creating a $[5 \times h \times w]$ tensor.

$w = 2048, 1024, 512 \quad h = 64$

B. Fully Convolutional Semantic Segmentation

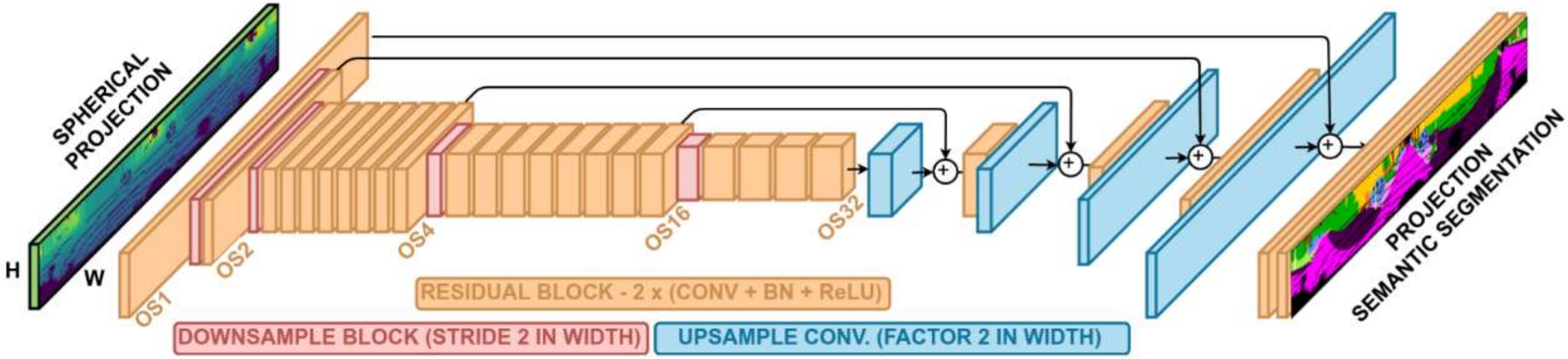
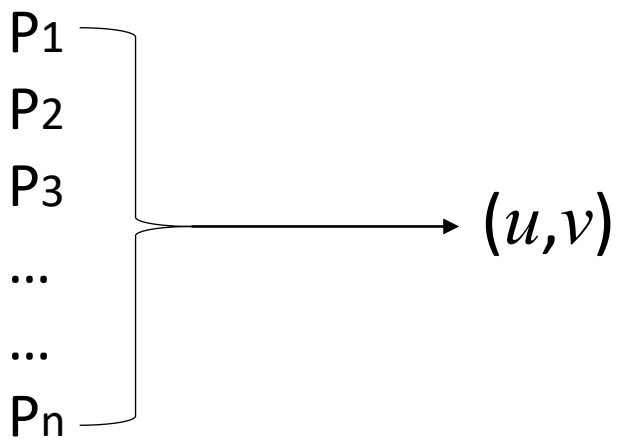


Fig. 3: Our fully convolutional semantic segmentation architecture. RangeNet53 is inspired in a Darknet53 Backbone [16].

$$\hat{y}_c = \frac{e^{\text{logit}_c}}{\sum_c e^{\text{logit}_c}} \qquad \mathcal{L} = -\sum_{c=1}^{C} w_c \, y_c \, \log\left(\hat{y}_c\right), \text{ where } w_c = \frac{1}{\log\left(f_c + \epsilon\right)}$$

C. Point Cloud Reconstruction from Range Image



130000 points per scan → [64 × 2048] 32768 points

D. Efficient Point Cloud Post-processing

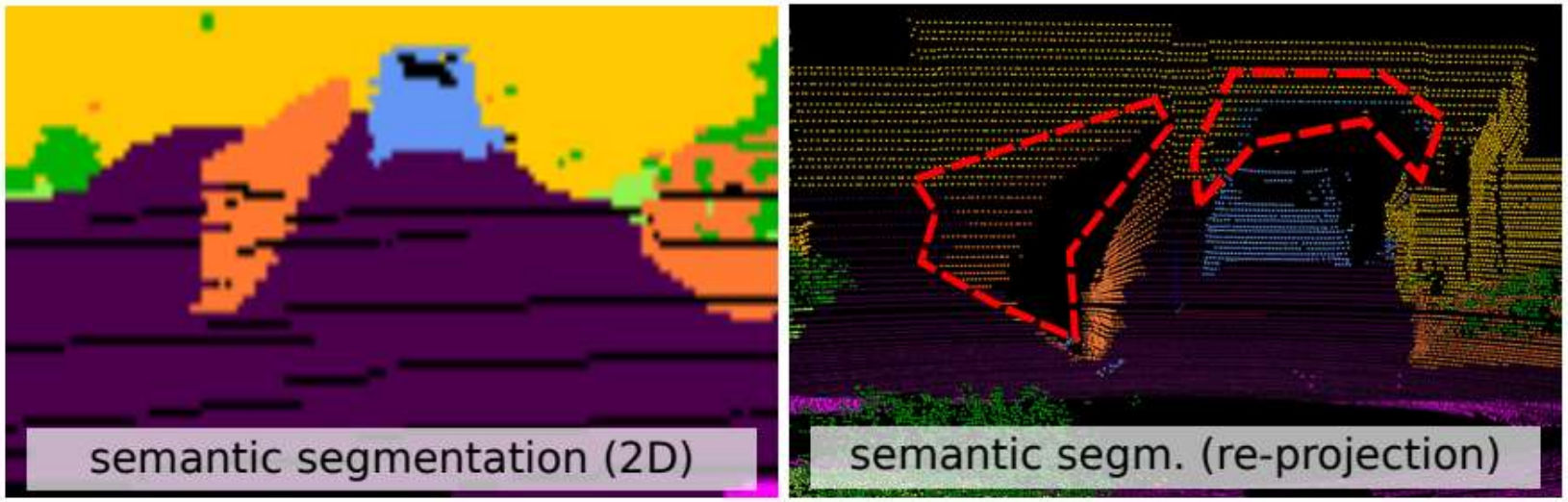


Fig. 4: Illustration of the label re-projection problem. Both the fence and the car in the range image (left) were given the proper semantic label, but during the process of sending the semantics back to the original points (right), the labels were also projected as “shadows”.

D. Efficient Point Cloud Post-processing

This algorithm requires setting four different hyperparameters: (i) S which the size of the search window, (ii) k which is number of nearest neighbors, (iii) *cut-off* which is the maximum allowed range difference for the k, and (iv) σ for the inverse gaussian.

Algorithm 1: Efficient Projective Nearest Neighbor Search for Point Labels

Data: Range Image $\mathbf{I}_{\text{range}}$ of size $W \times H$,
Label Image $\mathbf{I}_{\text{label}}$ of predictions of size $W \times H$,
Ranges $\mathcal{R}$ for each point $\mathbf{p} \in \mathcal{P}$ of size N,
Image coordinates (u, v) of each point in R.
Result: Labels $\mathbf{L}_{\text{consensus}}$ for each point of size N.

Let be $[l{:}u] = \{i \mid l \leq i \leq u\}$ the range from l to u.
/* Get S^2 neighbors $\mathbf{N}'$ for each pixel */
foreach $(u, v) \in [1{:}W] \times [1{:}H]$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{N}'[v \cdot W + u, j \cdot S + i] = \mathbf{I}_{\text{range}}[u + i, v + j]$
/* Get neighbors $\mathbf{N}$ for each point */
foreach $(u, v) \in \mathcal{C}$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{N}[v \cdot W + u, i \cdot S + j] = \mathbf{N}'[v \cdot W + u, i \cdot S + j]$
/* Fill in real point ranges */
foreach $i \in [1{:}N]$ **do**
  $\mathbf{N}[i, \lfloor (S \cdot S - 1)/2 \rfloor] = \mathcal{R}(i)$
/* Label neighbors $\mathbf{L}'$ for each pixel */
foreach $u \in [1{:}W]$, $v \in [1{:}H]$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{L}'[v \cdot W + u, i \cdot S + j] = \mathbf{I}_{\text{label}}[u + i, v + j]$
/* Get label neighbors $\mathbf{L}$ for each point */
foreach $(u, v) \in \mathcal{C}$ **do**
  foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
    $\mathbf{L}[v \cdot W + u, i \cdot S + j] = \mathbf{L}'[v \cdot W + u, i \cdot S + j]$
/* Distances to neighbors $\mathbf{D}$ for each point */
foreach $i \in n\,[1{:}N]$ **do**
  foreach $j \in n\,[1{:}S \cdot S]$ **do**
    $\mathbf{D}[i, j] = |\mathbf{N}[i, j] - \mathcal{R}(i)|$
/* Compute inverse Gaussian Kernal */
Let $\mathcal{N}(u \mid \mu, \sigma)$ be a Gaussian with mean μ and std. deviation σ.
foreach $(i, j) \in [1{:}S] \times [1{:}S]$ **do**
  $\mathbf{G}'[j \cdot S + i] = \mathcal{N}(i \mid 0, \sigma) \cdot \mathcal{N}(j \mid 0, \sigma)$
Let be $G_{\max} = \max\{\mathbf{G}'[i] \mid i \in [1{:}S \cdot S]\}$ the maximum of $\mathbf{G}'$
foreach $i \in [1{:}S \cdot S]$ **do**
  $\mathbf{G}[i] = 1 - G_{\max} \cdot \mathbf{G}'[i]$
/* Weight neighbors with inverse Gaussian kernel */
foreach $i \in [1{:}N]$ **do**
  foreach $j \in [1{:}S \cdot S]$ **do**
    $\mathbf{D}[i, j] = \mathbf{D}[i, j] \cdot \mathbf{G}[j]$
/* Find k-nearest neighbors $\mathcal{S}$ for each point */
foreach $i \in [1{:}N]$ **do**
  $\mathcal{S}[i] = \{j \mid |\{n \in [1{:}S \cdot S] \mid \mathbf{D}[i, n] < \mathbf{D}[i, j]\}| \leq k\}$
/* Gather votes. */
foreach $i \in [1{:}N]$ **do**
  $n = 1$
  foreach $j \in \mathcal{S}[i]$ **do**
    if $D[i, j] > \delta_{\text{cutoff}}$ **then**
      $\mathbf{L}_{\text{knn}}[i, n] = C + 1$
    else
      $\mathbf{L}_{\text{knn}}[i, n] = \mathbf{L}[i, j]$
    $n = n + 1$
/* Accumulate votes. */
foreach $i \in [1{:}N]$ **do**
  foreach $j \in [1{:}k]$ **do**
    if $\mathbf{L}_{\text{knn}}[i, j] \leq C$ **then**
      $\mathbf{V}[i, \mathbf{L}_{\text{knn}}[i, j]] = \mathbf{V}[i, \mathbf{L}_{\text{knn}}[i, j]] + 1$
/* Find maximum consensus. */
foreach $i \in [1{:}N]$ **do**
  $\mathbf{L}_{\text{consensus}}[i] = \arg\max_c \mathbf{L}_{\text{knn}}[i, c]$

A. Performance of RangeNet++ w.r.t. State-of-the-art

TABLE I: IoU [%] on test set (sequences 11 to 21). RangeNet21 and RangeNet53 represent the new baselines with augmented Darknet backbones (21 and 53 respectively), and the versions with (++) are treated with our fast point cloud post-processing based on range.

Approach	Size	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU	Scans/sec
Pointnet [14]	50000pts	46.3	1.3	0.3	0.1	0.8	0.2	0.2	0.0	61.6	15.8	35.7	1.4	41.4	12.9	31.0	4.6	17.6	2.4	3.7	14.6	2
Pointnet++ [15]		53.7	1.9	0.2	0.9	0.2	0.9	1.0	0.0	72.0	18.7	41.8	5.6	62.3	16.9	46.5	13.8	30.0	6.0	8.9	20.1	0.1
SPGraph [10]		68.3	0.9	4.5	0.9	0.8	1.0	6.0	0.0	49.5	1.7	24.2	0.3	68.2	22.5	59.2	27.2	17.0	18.3	10.5	20.0	0.2
SPLATNet [19]		66.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	70.4	0.8	41.5	0.0	68.7	27.8	72.3	35.9	35.8	13.8	0.0	22.8	1
TangentConv [20]		86.8	1.3	12.7	11.6	10.2	17.1	20.2	0.5	82.9	15.2	61.7	9.0	82.8	44.2	75.5	42.5	55.5	30.2	22.2	35.9	0.3
SqueezeSeg [21]	64 × 2048 px	68.8	16.0	4.1	3.3	3.6	12.9	13.1	0.9	85.4	26.9	54.3	4.5	57.4	29.0	60.0	24.3	53.7	17.5	24.5	29.5	**66**
SqueezeSeg-CRF [21]		68.3	18.1	5.1	4.1	4.8	16.5	17.3	1.2	84.9	28.4	54.7	04.6	61.5	29.2	59.6	25.5	54.7	11.2	36.3	30.8	55
SqueezeSegV2 [22]		81.8	18.5	17.9	13.4	14.0	20.1	25.1	3.9	88.6	45.8	67.6	17.7	73.7	41.1	71.8	35.8	60.2	20.2	36.3	39.7	50
SqueezeSegV2-CRF [22]		82.7	21.0	22.6	14.5	15.9	20.2	24.3	2.9	88.5	42.4	65.5	18.7	73.8	41.0	68.5	36.9	58.9	12.9	41.0	39.6	40
RangeNet21 [Ours]		85.4	26.2	26.5	18.6	15.6	31.8	33.6	4.0	91.4	57.0	74.0	26.4	81.9	52.3	77.6	48.4	63.6	36.0	50.0	47.4	20
RangeNet53 [Ours]	64 × 2048 px	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	**91.8**	64.8	74.6	**27.9**	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9	13
	64 × 1024 px	84.6	20.0	25.3	24.8	17.3	27.5	27.7	7.1	90.4	51.8	72.1	22.8	80.4	50.0	75.1	46.0	62.7	33.4	43.4	45.4	25
	64 × 512 px	81.0	9.9	11.7	19.3	7.9	16.8	25.8	2.5	90.1	49.9	69.4	2.0	76.0	45.5	74.2	38.8	62.7	25.5	38.1	39.3	52
RangeNet53++ [Ours+kNN]	64 × 2048 px	**91.4**	**25.7**	**34.4**	**25.7**	**23.0**	**38.3**	**38.8**	**4.8**	**91.8**	**65.0**	**75.2**	27.8	**87.4**	**58.6**	**80.5**	**55.1**	**64.6**	**47.9**	**55.9**	**52.2**	12
	64 × 1024 px	90.3	20.6	27.1	25.2	17.6	29.6	34.2	7.1	90.4	52.3	72.7	22.8	83.9	53.3	77.7	52.5	63.7	43.8	47.2	48.0	21
	64 × 512 px	87.4	9.9	12.4	19.6	7.9	18.1	29.5	2.5	90.0	50.7	70.0	2.0	80.2	48.9	77.1	45.7	64.1	37.1	42.0	41.9	38

B. Ablation Studies

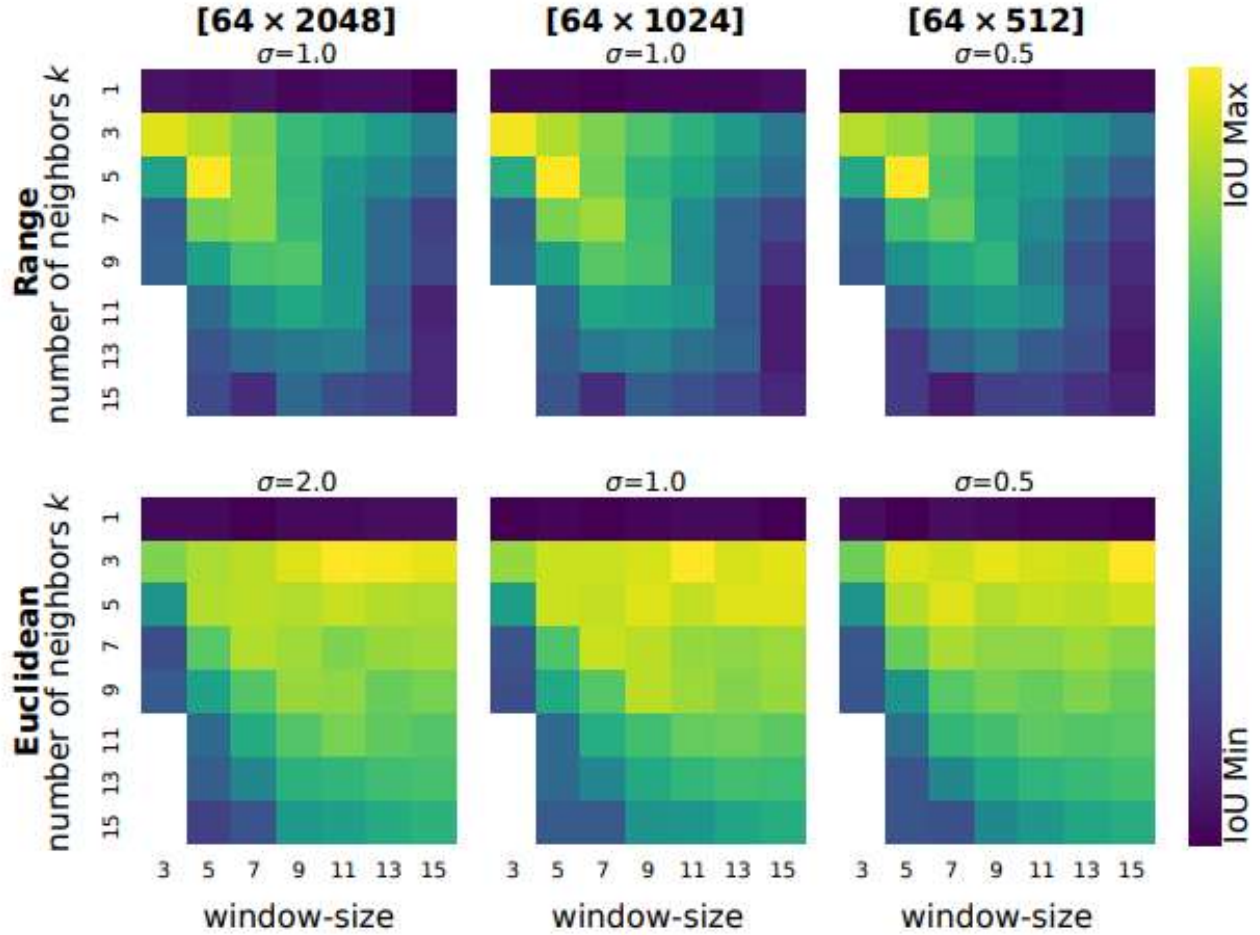


Fig. 5: Hyperparameter search post-processing for both Range (top row) and Euclidean distance (bottom row) using RangeNet53++, and different input resolutions. All experiments used *cutoff* = 1.0 m.

C. Post-Processing Influence

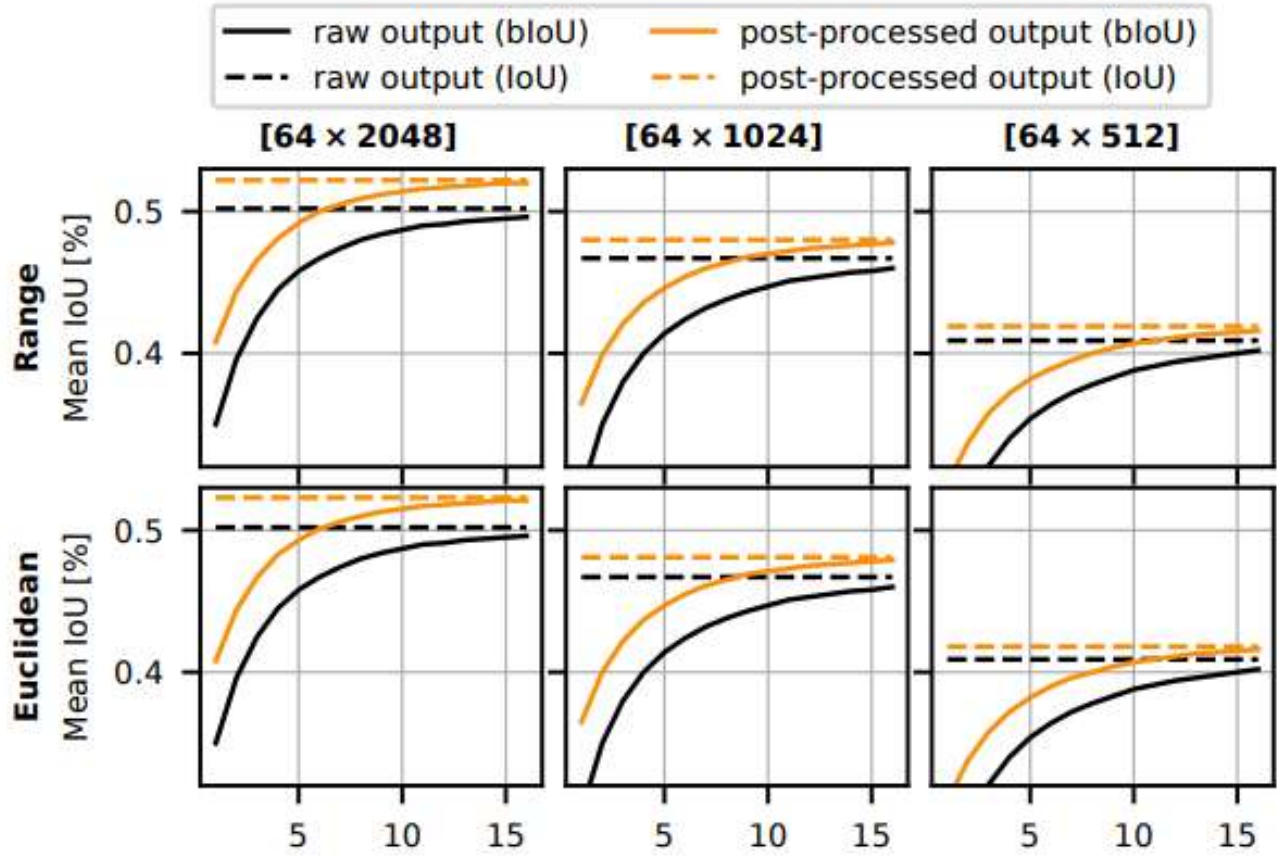


Fig. 6: Border IoU (bIoU) and IoU as a function of the distance to label change. This plot shows that our post-processing improves the IoU, and significantly improves the borderIoU, which means that it recovers blurry mask and discretization errors better.

D. Runtime

TABLE II: Runtime of RangeNet53++.

Hardware	Resolution (px)	Processing time (ms) CNN	Range	Euclid	Total	FPS
Quadro P6000	64×512	19	7	11	26	38
	64×1024	40			47	21
	64×2048	75			82	12
Jetson AGX	64×512	45	35	52	80	13
	64×1024	87			122	8
	64×2048	153			188	5

SqueezeSeg V3 ECCV 2020 Spatially-Adaptive Convolution (SAC)

Method	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	Motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU	Scans/sec
PNet [35]	46.3	1.3	0.3	0.1	0.8	0.2	0.2	0.0	61.6	15.8	35.7	1.4	41.4	12.9	31.0	4.6	17.6	2.4	3.7	14.6	2
PNet++ [36]	53.7	1.9	0.2	0.9	0.2	0.9	1.0	0.0	72.0	18.7	41.8	5.6	62.3	16.9	46.5	13.8	30.0	6.0	8.9	20.1	0.1
SPGraph [22]	68.3	0.9	4.5	0.9	0.8	1.0	6.0	0.0	49.5	1.7	24.2	0.3	68.2	22.5	59.2	27.2	17.0	18.3	10.5	20.0	0.2
SPLAT [43]	66.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	70.4	0.8	41.5	0.0	68.7	27.8	72.3	35.9	35.8	13.8	0.0	22.8	1
TgConv [46]	86.8	1.3	12.7	11.6	10.2	17.1	20.2	0.5	82.9	15.2	61.7	9.0	82.8	44.2	75.5	42.5	55.5	30.2	22.2	35.9	0.3
RLNet [15]	**94.0**	19.8	21.4	**42.7**	**38.7**	**47.5**	**48.8**	4.6	90.4	56.9	67.9	15.5	81.1	49.7	78.3	**60.3**	59.0	44.2	38.1	50.3	22
SSG [56]	68.8	16.0	4.1	3.3	3.6	12.9	13.1	0.9	85.4	26.9	54.3	4.5	57.4	29.0	60.0	24.3	53.7	17.5	24.5	29.5	**65**
SSG‡ [56]	68.3	18.1	5.1	4.1	4.8	16.5	17.3	1.2	84.9	28.4	54.7	4.6	61.5	29.2	59.6	25.5	54.7	11.2	36.3	30.8	53
SSGV2 [58]	81.8	18.5	17.9	13.4	14.0	20.1	25.1	3.9	88.6	45.8	67.6	17.7	73.7	41.1	71.8	35.8	60.2	20.2	36.3	39.7	50
SSGV2‡ [58]	82.7	21.0	22.6	14.5	15.9	20.2	24.3	2.9	88.5	42.4	65.5	18.7	73.8	41.0	68.5	36.9	58.9	12.9	41.0	39.6	39
RGN21 [30]	85.4	26.2	26.5	18.6	15.6	31.8	33.6	4.0	91.4	57.0	74.0	26.4	81.9	52.3	77.6	48.4	63.6	36.0	50.0	47.4	20
RGN53 [30]	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	**91.8**	64.8	74.6	**27.9**	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9	12
RGN53* [30]	91.4	25.7	34.4	25.7	23.0	38.3	38.8	4.8	**91.8**	**65.0**	**75.2**	27.8	87.4	58.6	80.5	55.1	64.6	47.9	55.9	52.2	11
SSGV3-21	84.6	31.5	32.4	11.3	20.9	39.4	36.1	**21.3**	90.8	54.1	72.9	23.9	81.1	50.3	77.6	47.7	63.9	36.1	51.7	48.8	16
SSGV3-53	87.4	35.2	33.7	29.0	31.9	41.8	39.1	20.1	**91.8**	63.5	74.4	27.2	85.3	55.8	79.4	52.1	64.7	38.6	53.4	52.9	7
SSGV3-21*	89.4	33.7	34.9	11.3	21.5	42.6	44.9	21.2	90.8	54.1	73.3	23.2	84.8	53.6	80.2	53.3	64.5	46.4	57.6	51.6	15
SSGV3-53*	92.5	**38.7**	**36.5**	29.6	33.0	45.6	46.2	20.1	91.7	63.4	74.8	26.4	**89.0**	**59.4**	**82.0**	58.7	**65.4**	**49.6**	**58.9**	**55.9**	6