This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Using a Waffle Iron for Automotive Point Cloud Semantic Segmentation

Gilles Puy[1] Alexandre Boulch[1] Renaud Marlet[1,2]

[1]valeo.ai, Paris, France [2]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France

Abstract

Semantic segmentation of point clouds in autonomous driving datasets requires techniques that can process large numbers of points efficiently. Sparse 3D convolutions have become the de-facto tools to construct deep neural networks for this task: they exploit point cloud sparsity to reduce the memory and computational loads and are at the core of today's best methods. In this paper, we propose an alternative method that reaches the level of state-of-the-art methods without requiring sparse convolutions. We actually show that such level of performance is achievable by relying on tools a priori unfit for large scale and high-performing 3D perception. In particular, we propose a novel 3D backbone, WaffleIron, made almost exclusively of MLPs and dense 2D convolutions and present how to train it to reach high performance on SemanticKITTI and nuScenes. We believe that WaffleIron is a compelling alternative to backbones using sparse 3D convolutions, especially in frameworks and on hardware where those convolutions are not readily available. The code is available at `https://github.com/valeoai/WaffleIron`.

1. Introduction

Lidar sensors deliver rich information about the 3D environment surrounding autonomous vehicles. Semantic segmentation of point clouds delivered by these lidars permits to autonomous vehicles to make sense of this 3D information in order to take proper and safe decisions. When studying the leaderboard of SemanticKITTI [2], we rapidly notice that all the top methods leverage sparse 3D convolutions. For example, the recent work 2DPASS [43] relies on an adapted version of SPVCNN [34] which, once trained with the help of images of the scene captured synchronously with the lidar, is currently the state-of-the-art method. As another example, Cylinder3D [51], later improved in [12], use sparse 3D convolutions on cylindrical voxels (particularly adapted to rotating lidars) with asymmetrical kernels suited to capture the geometry of the main objects in driving scenes.

Despite the undeniable success and efficiency of sparse convolutions, we seek here for 3D backbones which are free of them. Indeed, sparse convolutions remain available in a limited number of deep learning frameworks and hardware (essentially PyTorch and NVIDIA GPUs). One reason might be because they are challenging to implement efficiently [33]. Another reason may be because they are not as widely used as, e.g., dense 2D convolutions, and are thus not the first to be implemented in a new framework. Therefore, we would like to construct a 3D backbone (i) built with tools more broadly available than sparse convolutions, but which (ii) can reach the level of performance of the top methods on automotive datasets, while (iii) remaining easy to implement and to use. This would offer a compelling alternative to sparse 3D backbone, especially when sparse convolutions are not available.

We actually construct a novel 3D backbone built almost exclusively with standard MLPs and dense 2D convolutions, both readily available in all deep learning frameworks thanks to their wide use in the whole field of computer vision. Our backbone architecture, WaffleIron, is illustrated in Fig. 1, and is inspired by the recent MLP-Mixer [36]. It takes as input a point cloud with a token associated to each point. All these point tokens are then updated by a sequence of layers, each containing a token-mixing step (made of dense 2D convolutions) and a channel-mixing step (made of a MLP shared across points).

In addition, we explain how to train WaffleIron to make it reach the performance of the current best methods on automotive semantic segmentation benchmarks. The performance we obtain shows that standard MLPs and dense 2D convolutions, despite being a priori unfit for 3D segmentation, are sufficient to construct a 3D backbone reaching the state of the art.

Finally, WaffleIron is at least as easy to implement and to tune as any other backbone. The implementation consists in repeated applications of basic layers directly on the point tokens (an example of complete implementation is available in the supplementary material). The performance increases with the network width and depth, until an eventual saturation. The main hyperparameter to tune is the resolution of the 2D grid used for discretization before 2D convolution, but for which we observe stable results over a wide range

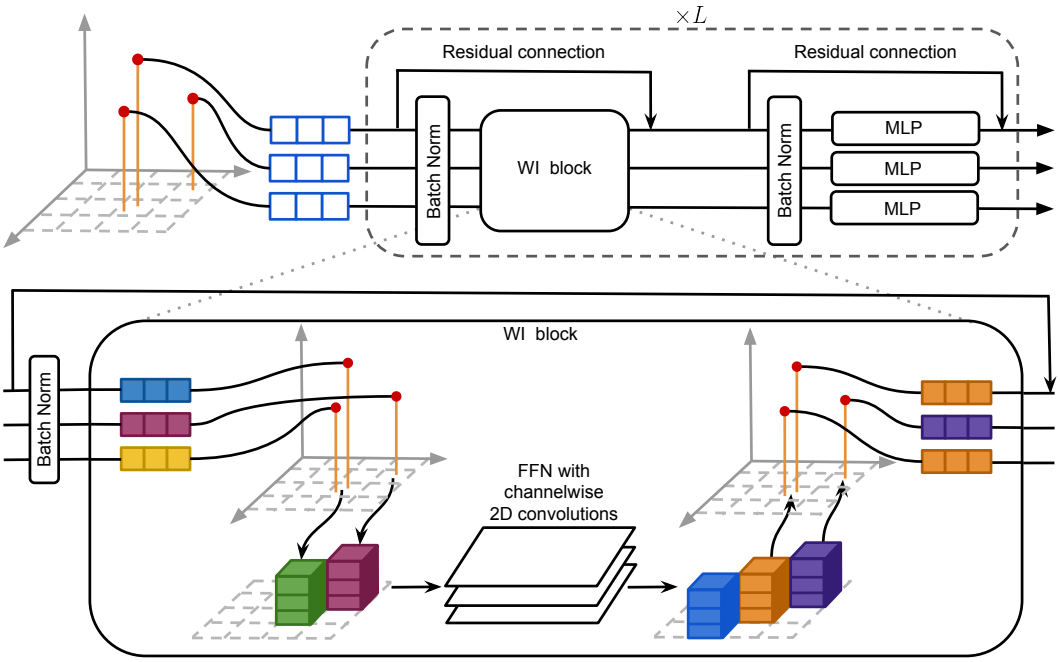

Figure 1. **WaffleIron backbone**. This 3D backbone takes as input point tokens, provided by an embedding layer (not represented), and updates these point representations L times via a point token-mixing layer (containing the WI block) followed by a channel-mixing layer. The WI block consists of a 2D projection along one of the main axes, a feed-forward network (FFN) with two dense channel-wise 2D convolutions with a ReLU activation in the hidden layer, and a simple copy of the 2D features to the 3D points. The channel-mixing layer contains a batch-norm, a MLP shared accross each point, and a residual connection. The WaffleIron backbone is free of any point downsampling or upsampling layer, farthest point sampling, nearest neighbor search, or sparse convolution.

of values (facilitating its tuning). The two most technical components to implement are reduced to: (i) the embedding layer used before WaffleIron and providing the point tokens, and (ii) the 2D projections followed by feature discretizations (applied before dense 2D convolutions).

In summary, our contributions are the following.

- We propose a novel and easy-to-implement 3D backbone for automotive point cloud semantic segmentation, which is essentially made of standard MLPs and dense 2D convolutions.
- We show that the hyperparameters of WaffleIron are easy to tune: the performance increases with the width and depth, until an eventual saturation; the performance is stable over a large range of 2D grid resolutions.
- We present how to train WaffleIron to reach the performance of top-entries on two autonomous driving benchmarks: SemanticKITTI [2] and nuScenes [5]. This shows that standard MLPs and dense 2D convolutions are actually sufficient to compete with the state of the art.

2. Related Work

We divide the related works into four categories: *point-based methods*, that work directly on points and update point representations throughout the network; *projection-based methods*, that project the points on a 2D grid at the input of the network, extract pixel-wise representations with a 2D network, and finally back-project the features in 3D for segmentation at the output of the network; *sparse convolution-based methods*, which voxelize the point clouds and uses sparse convolutions; *fusion-based methods*, which leverage different point cloud representations in parallel and fuse the corresponding features.

Point-based methods. PointNet [25] is the first method that appeared in this category, quickly followed by its improved version, PointNet++ [26]. Several methods then followed to improve the definition of point convolution, e.g., [35, 38, 4], to scale to large point clouds by exploiting point clustering, e.g., [18, 6], to optimize point sampling, e.g., [44, 13], or make point convolution faster to compute, e.g., [32]. Following the trend in image understanding, we also witness a growing amount of works, e.g., [48, 23, 17], exploiting transformer architectures, which are particularly suited to handle unordered set of points. Recently, Point-

Next [27] revisited and optimized PointNet++ with more modern tools and showed that it is still highly competitive in several benchmarks. In general, point-based methods are particularly effective to process dense point clouds such as those obtained with depth cameras in indoor scenes. These methods, unless combined with other point cloud representations, are seldomly used to process sparse outdoor lidar point clouds.

Among these point-based methods, let us discuss in more details two works which share some similarities with ours. The first work is PointMixer [8] which takes inspiration from the MLP-Mixer [36]. Despite the same source of inspiration, we remark several fundamental differences with our work. (i) The architecture differs significantly from WaffleIron: PointMixer is a U-Net architecture with downsampling/upsampling layers, while we keep the resolution of point cloud fixed and do not use any skip connection between the early and last layers. (ii) The spatial-mixing step is also fundamentally different as it is constructed using several sets of nearest neighbors points, while we use dense 2D convolutions. (iii) The method is used on dense point clouds captured in indoor scenes. The second work is PointMLP [21] which proposes a simple point-based network made only of MLPs. The PointMLP architecture is also very different from ours, starting with the spatial-mixing strategy which is done by aggregating information over sets of k-nearest neighbors. In addition, the application of PointMLP is limited to small scale point clouds for shape classification and part segmentation.

Projection-based methods. Projection-based methods are more used to process point clouds acquired with rotating lidars than point-based approaches. By working almost entirely on 2D feature maps, they usually benefit from very fast computations. Yet, their performance remains below methods leveraging sparse convolutions. Among these methods, we find some using the spherical (range) projection [22] or the bird's eye view projection [47]. Recent improvements have been achieved by making the convolution kernels better suited to the type of "images" produced by projection of the point clouds [40], by using techniques that reduce the loss information in the 2D encoder-decoder architectures [10], by solving an auxiliary tasks such as surface reconstruction [30], adding a learned post-processing step in 3D [15], or exploiting vision transformers pretained on image datasets [1].

Sparse convolution-based methods. These type of methods leverage point cloud sparsity to reduce the computational and memory load. In particular, they compute the result of the convolution only on occupied voxels [9]. These methods become particularly efficient on autonomous driving scenes, e.g., when, adapting the shape of the voxels to the point sampling structure [51]. Recently, some improvements have been obtained on these architectures by leveraging knowledge distillation techniques [12, 19]. Finally, some attention mechanisms are also now exploited on top of sparse convolution-based architectures to, e.g., adapt the classification layer to the input point cloud [50] or to improve feature quality [7, 45, 49, 16].

Fusion-based methods. These methods try to combine the advantage of different point representations to improve semantic segmentation. They rely on, e.g., bird's eye view and range representations used in a sequence [11], or used in parallel for fusing deep features [20, 28]. Another strategy is to combine fine-grained features provided by point representation with high-level voxel representations [34, 46, 24]. RPVNet [41] fuses features extracted at multiple layers of three different networks, each dealing with range, point or voxel representations.

3. Our Method

3.1. WaffleIron Backbone

High-level description. WaffleIron is illustrated in Fig. 1. It takes as input a point cloud with a F-dimensional token associated to each point. These point tokens, obtained by an embedding layer described in Sec. 3.2, are updated L times thanks to token-mixing layers and channel-mixing layers. The core component of the token-mixing layer is our novel WI block. It is made of a 2D projection along one of the main axes, a discretization of the features on a 2D grid, and a feed-forward network (FFN) with dense 2D convolutions. The channel mixing layer is essentially made of an MLP shared across each point.

Formal definition. WaffleIron takes as input a point cloud with N points whose Cartesian xyz-coordinates are denoted by $\boldsymbol{p}_i \in \mathbb{R}^3$, $i = 1, \ldots, N$. Each point is associated with a point token $\boldsymbol{f}_i^{(0)} \in \mathbb{R}^F$ provided by a embedding layer (see Sec. 3.2). To simplify the following equations, we group all the point tokens in a large matrix $\mathsf{F}^{(0)}$ of size $F \times N$. These tokens are then transformed by a series of L layers, each satisfying

$$\mathsf{G}^{(\ell)} = \mathsf{F}^{(\ell)} + \mathrm{WI}\left(\mathrm{BN}\left(\mathsf{F}^{(\ell)}\right)\right), \quad (1)$$

$$\mathsf{F}^{(\ell+1)} = \mathsf{G}^{(\ell)} + \mathrm{MLP}\left(\mathrm{BN}\left(\mathsf{G}^{(\ell)}\right)\right), \quad (2)$$

to obtain the deep point features $\mathsf{F}^{(L)} \in \mathbb{R}^{F \times N}$, then used to classify each point thanks to a single linear layer.[1] Eq. (1) and Eq. (2) corresponds to the token-mixing step and channel-mixing step, respectively. BN denotes batch normalization. The MLP is applied point-wise and contains two layers with a ReLU activation after the first layer.

[1] In our implementation, we also used two layerscale layers [37]: one after the WI block and one after the MLP.

The WI block mixes the features spatially as illustrated in the lower part of Fig. 1. It processes input 3D features $\mathsf{F} \in \mathbb{R}^{F \times N}$ in three steps to obtain the residual which satisfies $\mathrm{WI}(\mathsf{F}) = \mathrm{Inflat} \circ \mathrm{Conv} \circ \mathrm{Flat}(\mathsf{F})$. These three steps are described below.

1. $\mathrm{Flat}(\cdot)$: Project ("flatten") the points on one of the planes (x, y), (x, z) or (y, z). Discretize the chosen plane into M cells of size $\rho \times \rho$. Within each 2D cell, average the 3D features of all points falling in this cell. We thus obtain the 2D feature map $\mathrm{Flat}(\mathsf{F}) \in \mathbb{R}^{F \times M}$.
2. $\mathrm{Conv}(\cdot)$: Process the 2D feature map $\mathrm{Flat}(\mathsf{F})$ with a feed-forward network (FFN) consisting of two layers of channel-wise 2D convolutions and a ReLU activation in the hidden layer. We obtain the 2D feature map $\mathrm{Conv}(\mathrm{Flat}(\mathsf{F}))$.
3. $\mathrm{Inflat}(\cdot)$: For each 3D point, find the 2D cell into which this point falls into, and copy ("inflate") the corresponding feature from $\mathrm{Conv}(\mathrm{Flat}(\mathsf{F}))$. This yield the residual $\mathrm{WI}(\mathsf{F}) \in \mathbb{R}^{F \times N}$.

The name of our method, WaffleIron, is inspired by the effect of the first step on the point cloud: it is flattened and imprinted with a regular 2D grid, as if it was compressed between the plates of a waffle iron.

$\mathrm{Flat}(\cdot)$ and $\mathrm{Inflat}(\cdot)$ implementations. The computations in $\mathrm{Flat}(\cdot)$ and $\mathrm{Inflat}(\cdot)$ are cheap. Both steps can be implemented using a sparse-dense matrix multiplication. It is sufficient to store a sparse matrix $\mathsf{S} \in \mathbb{R}^{N \times M}$ with N non-zero entries and structured as follows. For each 3D point $\boldsymbol{p}_i$: *(a)* compute the index $j \in \{1, \dots, M\}$ of the 2D cell into which this point falls into (by quantizing $\boldsymbol{p}_i$); *(b)* set the entry in the ith row and the jth column of S to 1. Then, the 2D feature map in the $\mathrm{Flat}(\cdot)$ step satisfies $\mathrm{Flat}(\mathsf{F}) = \mathsf{F}\,\mathsf{S} \oslash \mathsf{N}\,\mathsf{S}$, where $\mathsf{N} \in \mathbb{R}^{F \times N}$ is a matrix where all entries are set to 1 and $\oslash$ is the element-wise division. Note that $\mathsf{N}\,\mathsf{S}$ indicates the number of 3D points falling in each 2D cell, ensuring a proper average of 3D features falling in the same cells. Finally, the 3D residual $\mathrm{WI}(\mathsf{F})$ obtained in the $\mathrm{Inflat}(\cdot)$ step satisfies $\mathrm{WI}(\mathsf{F}) = \mathrm{Conv}(\mathrm{Flat}(\mathsf{F}))\,\mathsf{S}^\intercal$.

3.2. Practical Considerations

Choice of the projection plane. In our proposed architecture, we repeatedly project along each main axis. Concretely, we sequentially project on planes (x, y), (x, z) and (y, z) at layer $\ell = 1$, $\ell = 2$, and $\ell = 3$, respectively, and repeat this sequence until layer $\ell = L$. In our experiments, we thus choose L as a multiple of 3. We nevertheless study the impact of different projection strategies in Sec. 4.7.

Resolution of the 2D grids. For simplicity, we choose a single resolution $\rho \times \rho$ for all 2D grids used in the network.

2D convolutions. We use basic 2D kernels of size 3×3 for all layers throughout the network.

Embedding layer. Let $\boldsymbol{h}_i$ denote the low-level features readily available at point $\boldsymbol{p}_i$, e.g., the height, range and lidar intensity of the point. Inspired by DGCNN [38], the embedding layer extracting the initial tokens $\boldsymbol{f}_i^{(0)}$ merges global and local information around each point:

$$\boldsymbol{f}_i^{(0)} = \mathrm{LN}\big(\,[\mathrm{LN}(\boldsymbol{h}_i),\, \max_{j \in \mathcal{N}_i} \mathrm{MLP}(\boldsymbol{h}_j - \boldsymbol{h}_i)]\,\big) \qquad (3)$$

where LN denotes linear layers and $\mathcal{N}_i$ the set of k nearest points to $\boldsymbol{p}_i$. The features $\boldsymbol{h}_i$ are pre-normalized by a batch normalization layer before applying (3).

3.3. Discussion

Ease of implementation. A PyTorch implementation of WaffleIron is available in the supplementary material: it consists of repeated applications of basic layers directly on the point tokens, highlighting the implementation simplicity. We have successfully tested this implementation on NVIDIA GPUs but also, up to minor adaptations, on AMD GPUs, on which, as far as we know, no efficient implementation of sparse convolutions are readily available. This illustrates that WaffleIron is easily usable on different hardwares.

We chose to keep the resolution of the point cloud constant all the way through the backbone. This avoids the implementation of point downsampling and upsampling layers, the tuning of the associated point sampling technique, and the multiple nearest neighbors searches that are usually involved. Despite the absence of such layers, WaffleIron requires reasonable computing capacity: the model used to obtain our final result on SemanticKITTI can be trained on a single NVIDIA Tesla V100 GPU with 32 GB of memory. Nevertheless, improvements of WaffleIron could include downsampling layers to optimize the computation and memory loads.

Besides the embedding layer, the other most technical step to implement is the projection on 2D planes followed by feature discretization on a 2D grid. We greatly simplified this step: we project only along one of the main axes (so the projected coordinates are available without extra-computation); we use a single 2D grid resolution; feature discretization can be done by multiplication with a fixed (non-learnable) sparse matrix constructed thanks to a simple quantization of the point coordinates.

Ease of hyperparameter tuning. We show in Sec. 4.4 that the performance improves on all datasets when increasing the width F and depth L of WaffleIron until a potential saturation. The final choice for these values could, for example, be guided essentially by the desired or available

Method	SpConv free	**mIoU%**	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign
RandLA-Net [13]	✓	53.9	94.2	26.0	25.8	40.1	38.9	49.2	48.2	7.2	90.7	60.3	73.7	20.4	86.9	56.3	81.4	61.3	66.8	49.2	47.7
KPConv [35]	✓	58.8	96.0	30.2	42.5	33.4	44.3	61.5	61.6	11.8	88.8	61.3	72.7	31.6	90.5	64.2	84.8	69.2	69.1	56.4	47.4
SalsaNext [10]	✓	59.5	91.9	48.3	38.6	38.9	31.9	60.2	59.0	19.4	91.7	63.7	75.8	29.1	90.2	64.2	81.8	63.6	66.5	54.3	62.1
NAPL [50]		61.6	96.6	32.3	43.6	47.3	47.5	51.1	53.9	36.5	89.6	67.1	73.7	31.2	91.9	67.4	84.8	69.8	68.8	59.1	59.2
PCSCNet [24]		62.7	95.7	48.8	46.2	36.4	40.6	55.5	68.4	55.9	89.1	60.2	72.4	23.7	89.3	64.3	84.2	68.2	68.1	60.5	63.9
KPRNet [15]	✓	63.1	95.5	54.1	47.9	23.6	42.6	65.9	65.0	16.5	93.2	**73.9**	80.6	30.2	91.7	68.4	85.7	69.8	71.2	58.7	64.1
Lite-HDSeg [29]	✓	63.8	92.3	40.0	54.1	37.7	39.6	59.2	71.6	54.1	93.0	68.2	78.3	29.3	91.5	65.0	78.2	65.8	65.1	59.5	67.7
SVASeg [49]		65.2	96.7	56.4	57.0	49.1	56.3	70.6	67.0	15.4	92.3	65.9	76.5	23.6	91.4	66.1	85.2	72.9	67.8	63.9	65.2
AMVNet [20]	✓	65.3	96.2	59.9	54.2	48.8	45.7	71.0	65.7	11.0	90.1	71.0	75.8	32.4	92.4	69.1	85.6	71.7	69.6	62.7	67.2
GFNet [28]	✓	65.4	96.0	53.2	48.3	31.7	47.3	62.8	57.3	44.7	**93.6**	72.5	**80.8**	31.2	**94.0**	**73.9**	85.2	71.1	69.3	61.8	68.0
JS3C-Net [42]		66.0	95.8	59.3	52.9	54.3	46.0	69.5	65.4	39.9	88.8	61.9	72.1	31.9	92.5	70.8	84.5	69.8	68.0	60.7	68.7
SPVNAS [34]		66.4	97.3	51.5	50.8	**59.8**	58.8	65.7	65.2	43.7	90.2	67.6	75.2	16.9	91.3	65.9	86.1	73.4	71.0	64.2	66.9
2DPASS* [43]		67.4	96.3	51.1	55.8	54.9	51.6	76.8	79.8	30.3	89.8	62.1	73.8	33.5	91.9	68.7	86.5	72.3	71.3	63.7	70.2
Cylinder3D [51]		67.8	97.1	67.6	64.0	50.8	58.6	73.9	67.9	36.0	91.4	65.1	75.5	32.3	91.0	66.5	85.4	71.8	68.5	62.6	65.6
$(AF)^2$-S3Net [7]		69.7	94.5	65.4	**86.8**	39.2	41.1	**80.7**	80.4	**74.3**	91.3	68.8	72.5	**53.5**	87.9	63.2	70.2	68.5	53.7	61.5	71.0
RPVNet [41]		70.3	**97.6**	68.4	68.7	44.2	61.1	75.9	74.4	43.4	93.4	70.3	80.7	33.3	93.5	72.1	86.5	75.1	71.7	64.8	61.4
SDSeg3D [19]		70.4	97.4	58.7	54.2	54.9	**65.2**	70.2	74.4	52.2	90.9	69.4	76.7	41.9	93.2	71.1	86.1	74.3	71.1	65.4	70.6
GASN [45]		70.7	96.9	65.8	58.0	59.3	61.0	80.4	**82.7**	46.3	89.8	66.2	74.6	30.1	92.3	69.6	**87.3**	73.0	**72.5**	66.1	71.6
WaffleIron	✓	70.8	97.2	**70.0**	69.8	40.4	59.6	77.1	75.5	41.5	90.6	70.4	76.4	38.9	93.5	72.3	86.7	**75.7**	71.7	**66.2**	**71.9**
PVKD [12]		**71.2**	97.0	67.9	69.3	53.5	60.2	75.1	73.5	50.5	91.8	70.9	77.5	41.0	92.4	69.4	86.5	73.8	71.9	64.9	65.8

Table 1. Semantic segmentation performance on SemanticKITTI test set. The second column indicates if the method is free of sparse convolutions (SpConv). The best and second-best IoUs are bold and underlined, respectively. The scores are obtained from the official leaderboard of SemanticKITTI when available, otherwise from the respective paper. Regarding 2DPASS*, we report the results of the baseline of [43] trained with lidar data but *no* images, i.e., in the same setting as the other methods in this table. *The table contains the score of methods published before the date of submission.*

computation resources. The sole remaining parameter to tune in WaffleIron is the resolution $\rho \times \rho$ of the 2D grid in the $\mathrm{Flat}(\cdot)$ step. The optimal value of this parameter is dataset-dependent but we noticed that results remain stable for a wide range of values, which makes intensive fine-tuning unnecessary. In particular, a resolution of $50\ \mathrm{cm}$ is nearly optimal for both SemanticKITTI and nuScenes.

4. Experiments

4.1. Datasets

We conduct experiments on two large-scale autonomous driving datasets: SemanticKITTI [2] and nuScenes [5].

SemanticKITTI. This dataset contains 22 sequences where each point cloud is segmented into 19 semantic classes. We use the usual split where the first 11 sequences constitute the training set, except the 8th sequence used for validation, and the last 11 sequences constitute the test set.

nuScenes. Each point in this dataset [5] is annotated with one of the 16 considered semantic classes. The dataset contains 1000 scenes acquired in Boston and Singapore. We use the official split with 700 scenes for training, 150 scenes for validation and 150 scenes for test.

4.2. Implementation Details

During training and test, the point clouds are slightly downsampled by keeping only one point per voxel of size 10 cm. We use mixed precision for computations. At test time, the predicted labels are propagated to all points of the original point cloud by nearest neighbor interpolation.

Training. To control the memory usage and facilitate batch processing, we pre-process the point cloud as follows. We keep the size M of the 2D grids used in the WI blocks fixed. This is achieved by cropping the input point cloud to a fixed range. On SemanticKITTI, we use a range of $(-50\,\mathrm{m},\ 50\,\mathrm{m})$ along the x, y axes and $(-3\,\mathrm{m},\ 2\,\mathrm{m})$ along the z-axis, as in [51]. On nuScenes, we use the same range of along the x, y axes and $(-5\,\mathrm{m},\ 5\,\mathrm{m})$ along the z-axis.

Method	SpConv free	**mIoU%**	barrier	bicycle	bus	car	const. veh.	motorcycle	pedestrian	traffic cone	trailer	truck	driv. surf.	other flat	sidewalk	terrain	manmade	vegetation
$(AF)^2$-S3Net [7]		62.2	60.3	12.6	82.3	80.0	20.1	62.0	59.0	49.0	42.2	67.4	94.2	68.0	64.1	68.6	82.9	82.4
RangeNet++ [22]	✓	65.5	66.0	21.3	77.2	80.9	30.2	66.8	69.6	52.1	54.2	72.3	94.1	66.6	63.5	70.1	83.1	79.8
PolarNet [47]	✓	71.0	74.7	28.2	85.3	90.9	35.1	77.5	71.3	58.8	57.4	76.1	96.5	71.1	74.7	74.0	87.3	85.7
SalsaNext [10]	✓	72.2	74.8	34.1	85.9	88.4	42.2	72.4	72.2	63.1	61.3	76.5	96.0	70.8	71.2	71.5	86.7	84.4
SVASeg [49]		74.7	73.1	44.5	88.4	86.6	48.2	80.5	77.7	65.6	57.5	82.1	96.5	70.5	74.7	74.6	87.3	86.9
AMVNet [20]	✓	76.1	79.8	32.4	82.2	86.4	**62.5**	81.9	75.3	**72.3**	**83.5**	65.1	97.4	67.0	78.8	74.6	90.8	87.9
GFNet [28]	✓	76.1	**81.1**	31.6	76.0	90.5	60.2	80.7	75.3	71.8	82.5	65.1	**97.8**	67.0	**80.4**	**76.2**	**91.8**	**88.9**
Cylinder3D [51]		76.1	76.4	40.3	91.2	**93.8**	51.3	78.0	78.9	64.9	62.1	84.4	96.8	71.6	76.4	75.4	90.5	87.4
2DPASS*† [43]		76.2	75.3	43.5	**95.3**	91.2	54.5	78.9	72.8	62.1	70.0	83.2	96.3	73.2	74.2	74.9	88.1	85.9
RPVNet [41]		77.6	78.2	43.4	92.7	93.2	49.0	85.7	80.5	66.0	66.9	84.0	96.9	73.5	75.9	76.0	90.6	**88.9**
WaffleIron (ours)	✓	77.6	78.7	51.3	93.6	88.2	47.2	86.5	81.7	68.9	69.3	83.1	96.9	74.3	75.6	74.2	87.2	85.2
SDSeg3D [19]		77.7	77.5	49.4	93.9	92.5	54.9	86.7	80.1	67.8	65.7	86.0	96.4	74.0	74.9	74.5	86.0	82.8
SDSeg3D† [19]		78.7	78.2	52.8	94.5	93.1	54.5	88.1	82.2	69.4	67.3	**86.6**	96.4	74.5	75.2	75.3	87.1	84.1
WaffleIron† (ours)	✓	**79.1**	79.8	**53.8**	94.3	87.6	49.6	**89.1**	**83.8**	70.6	72.7	84.9	97.1	**75.8**	76.5	75.9	87.8	86.3

Table 2. Semantic segmentation performance on nuScenes validation set. The second column indicates if the method is free of sparse convolutions (SpConv). Best and second-best scores are bold and underlined. The scores of each method are obtained from their respective paper, except for RangeNet++, PolarNet, SalsaNext for which they were obtained from [51], and for AMVNet obtained from [41]. Regarding 2DPASS*, we report the scores obtained for the network trained using lidar data and *no images*, i.e., in the same setting as the other methods in this table. Test time augmentations (TTA), indicated by †, are used in some methods; we thus report the score of WaffleIron with and without TTA. *The table contains the score of methods published before the date of submission.*

We also keep the number of points $N = 20\,000$ fixed. If the input point cloud has a size larger than N, then we pick a point at random and keep its closest $N-1$ points, otherwise the point cloud is zero padded.

All models are trained using AdamW for 45 epochs, with a weight decay of 0.003, a batch size of 4, and a learning rate scheduler with a linear warmup phase from 0 to 0.001 during the first 4 epochs followed by a cosine annealing phase that decreases the learning rate to 10^{-5} at the end of the last epoch. The loss is the sum of the cross-entropy and the Lovász loss [3]. The point tokens are computed with 16 nearest neighbors in the embedding layer (3). We apply classical point cloud augmentations on nuScenes and SemanticKITTI: random rotation around the z-axis, random flip of the direction of the x and y-axis, and random scaling. Unless mentioned otherwise, we also use stochastic depth [14] with a layer drop probability of 0.2.

Test and validation. Because the range along each axis considered at train time is sufficiently large to contain nearly the whole point clouds, we continue cropping the points clouds on the same range during validation and test. The labels of the points outside the range are obtained by nearest neighbors interpolation. We use all the input points after voxel downsampling (hence do not constraint N) during test and validation. Some methods leverage test time augmentations, e.g., [51, 12, 43]; when applied, we average the softmax pointwise probabilities obtained with 10 different augmentations (random rotation, flip and stochastic depth activated). We do *not* use model ensemble to boost the test or validation performance.

Input features. Unless mentioned otherwise, the input feature $\boldsymbol{h}_i$ to the embedding layer is a 5-dimensional vector which contains the lidar intensity, the Cartesian coordinates xyz and range of the corresponding point $\boldsymbol{p}_i$.

4.3. Performance on Autonomous Driving Datasets

On both datasets, we train a WaffleIron backbone with $L = 48$ layers. We use $F = 256$-dimensional point tokens and a grid resolution ρ of 40 cm on SemanticKITTI . We use $F = 384$ and $\rho = 60$ cm on nuScenes. These choices of hyperparameters are justified in the next sections.

SemanticKITTI. We evaluate our method on the test split. We adopt the training and inference practices used by the best performing techniques, e.g., [41, 51, 12, 43]. In particular, the model is trained using both the training and validation splits, and test time augmentations are used at inference. In addition to the classical rotation, flip, scaling augmentations during training, [41, 43] use instance cutmix augmentations. Taking also inspiration from the sugges-

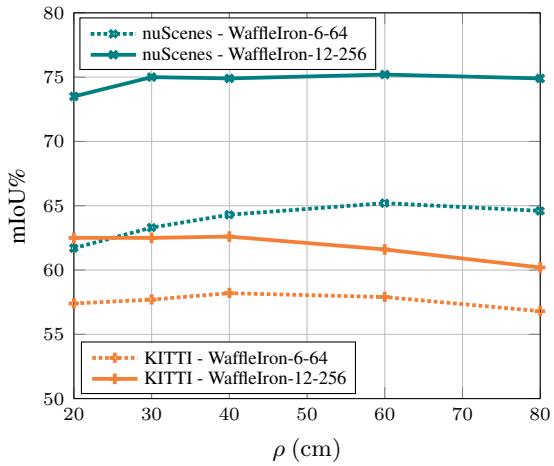

Figure 2. Influence of the grid resolution ρ on the performance of WaffleIron. We train each backbone on the training set of nuScenes or SemanticKITTI and compute the mIoU on the corresponding validation set. We report the average mIoU% obtained at the last training epoch of two independent runs.

	nuScenes ($\rho = 60$cm)			
	$L = 6$	$L = 12$	$L = 24$	$L = 48$
$F = 64$	65.2	-	-	-
$F = 128$	70.8	-	-	-
$F = 256$	73.2	75.2	75.4	**76.1**
	KITTI ($\rho = 40$cm)			
	$L = 6$	$L = 12$	$L = 24$	$L = 48$
$F = 64$	58.2	-	-	-
$F = 128$	61.4	-	-	-
$F = 256$	61.8	**62.6**	-	62.5

Table 3. Influence of the width F and depth L on the performance of WaffleIron. We train each backbone on the training set of nuScenes or SemanticKITTI and compute the mIoU on the corresponding validation set. We report the average mIoU% obtained at the last training epoch of two independent runs.

tions made in the official code repository of [12], we combine instance cutmix with polarmix [39]. We provide further details about instance cutmix and polarmix in Sec. 4.5.

We present the results obtained on the test set in Tab. 1. WaffleIron is ranked second in term of global mIoU, just 0.4 point away from PVKD. We surpasses the mIoU obtained with popular methods such as Cylinder3D and SPVNAS. It is interesting to notice that WaffleIron is among the best methods in the segmentation of small and rare objects such as bicycles, motorcycles, poles and traffic-signs. The take-home message is that WaffleIron is among the top performing methods on SemanticKITTI, making it a compelling alternative if, e.g., one is constrained to using regular deep network layers.

nuScenes. The model is trained on the official training split. We present in Tab. 2 the scores obtained by WaffleIron and other methods on the validation set. Once again the results show that WaffleIron can reach the current best mIoUs. As before, it is interesting to notice that WaffleIron performs well on rare and small objects such as bicyles, motorcycles and pedestrians. It confirms it is possible to reach the top of the leaderboard on nuScenes with WaffleIron.

4.4. Sensitivity to Hyperparameters

We denote by WaffleIron-L-F a backbone with L layers and F-dimensional point tokens. We only use here 3-dimensional vectors $\boldsymbol{h}_i$ (lidar intensity, height and range of $\boldsymbol{p}_i$) and do not use stochastic depth for training. We justify the use of 5-dimensional vectors $\boldsymbol{h}_i$ and stochastic depth in the next section.

2D grid resolution. We study the impact of ρ on each dataset for two versions of our network: WaffleIron-6-64 and WaffleIron-12-256. We notice in Fig. 2 that the performance are stable for a large range of grid resolutions. On nuScenes, the mIoU% varies by at most one point for ρ between 40 cm and 80 cm with a maximum reached at 60 cm. Similarly, on SemanticKITTI, the mIoU% varies by at most one point for ρ between 20 cm and 60 cm, with a maximum reached at 40 cm. In summary, WaffleIron is only mildly sensitive to the grid resolution, and, therefore, can accommodate a coarse tuning of this parameter. In particular, $\rho = 50$ cm could be a good default value to accommodate nearly optimally both datasets.

Choice of F and L. We study in Tab. 3 the impact of increasing L and F in WaffleIron. We notice the same behavior on both datasets with an increase of performance as both L and F increases, with the start of a saturation on SemanticKITTI. On SemanticKITTI, we did not notice any improvement or degradation for $F \geqslant 256$ at $L = 48$. We chose WaffleIron-48-256 to obtain our result on the test set. On nuScenes, we were able to improve the results when using $F = 384$ (see supp. mat.), hence our choice in Sec. 4.3.

4.5. Regularizations and input features

In this section, we show the benefit of using more regularizations via data augmentations with instance cutmix and polarmix (only on SemanticKITTI), and via the use of stochastic depth during training. We also justify the use of 5-dimensional input vectors $\boldsymbol{h}_i$, as opposed to the 3-dimensional $\boldsymbol{h}_i$ used in Sec. 4.4. We present here the results on SemanticKITTI. A similar study is available in the supp. mat. for nuScenes. "Baseline" refers to a WaffleIron-48-256 backbone trained with 3-dimensional input vectors

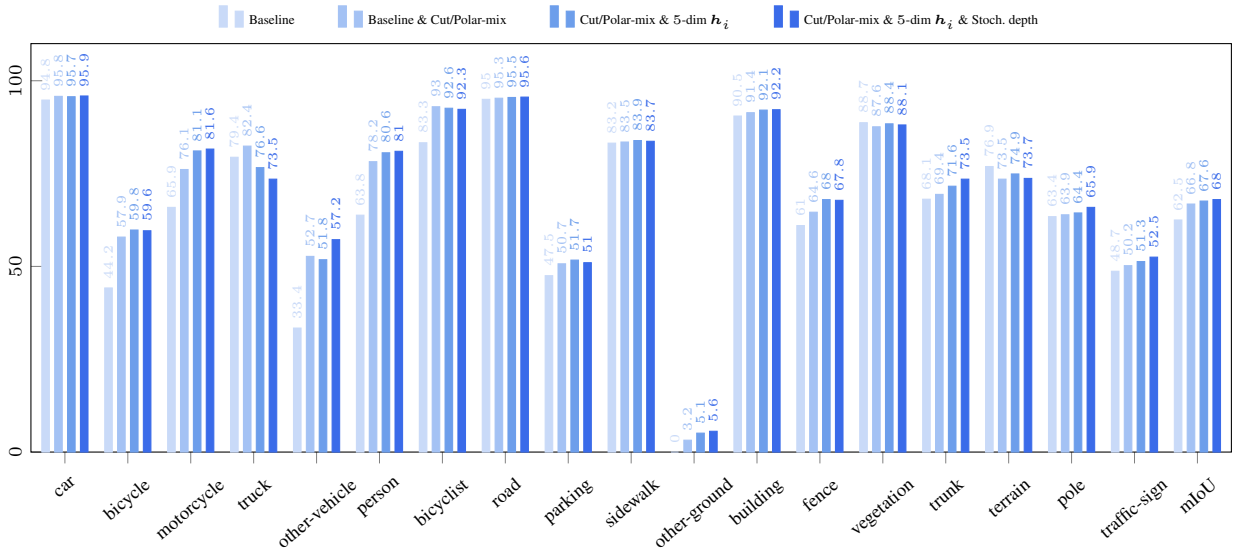

Figure 3. Influence of polarmix, instance cutmix, input vectors $\boldsymbol{h}_i$, and stochastic depth on the performance of WaffleIron on SemanticKITTI. We train and evaluate WaffleIron-48-256 backbones on the official train and validation set, respectively. We report the average mIoU% obtained at the last training epoch of two independent runs. To improve readability, we omitted the IoU% on motorcyclist, which varies between 0.0 and 1.3.

(lidar intensity, height and range), no stochastic depth, no instance cutmix or polarmix.

Instance cutmix & polarmix on SemanticKITTI. Following [41, 43], we use instance cutmix on rare-class objects to improve the segmentation performance on SemanticKITTI. In our implementation, we extract all instances of the following classes: bicycle, motorcycle, person, bicyclist, other vehicles. During training, we randomly select at most 40 instances of each class; we apply a random rotation around the z-axis, a random flip along the direction of the x or y-axes, and a random rescaling on each instance; we place each instance at a random location on a road, parking or sidewalk. We did not apply instance cutmix on motorcyclists. Indeed, our method (like many others) reaches very low score on this class on the validation set. Tuning instance cutmix on motorcyclists is thus impossible as we cannot measure its beneficial or adverse effect. We therefore make the choice to not apply instance cutmix on motorcyclists. In addition, we also use polarmix [39] on the same classes as instance cutmix.

The impact of these augmentations is presented in Fig. 3. The mIoU% improves from 62.5 to 66.8, with, as expected, most of the improvement due to a large boost of performance in the classes used for these augmentations.

Input features & stochastic depth. We start by showing the interest of using 5-dimensional input vectors $\boldsymbol{h}_i$ (intensity, x, y, z, and range of $\boldsymbol{p}_i$) instead of 3-dimensional input vectors (intensity, height=z, range). The impact of this change of input vector is presented in Fig. 3 on top instance cutmix and polarmix: the mIoU% increases from 66.8 to 67.6 with an improvement on most classes. Finally, using stochastic depth on top of all presented recipes permits us to achieve our best mIoU% of 68.0 on the validation set.

Time (ms)	Mink34	Mink18	SPVCNN (orig.)	SPVCNN† (2DPASS)	Ours
nuScenes	94	66	74	94	92
SemKITTI	114	91	104	80	193

Table 4. Inference time of several backbones and WaffleIron-48-256 (ours: embedding + backbone + classification) estimated on the validation sets of nuScenes and semanticKITTI, using a batch size of 1 and a NVIDIA GeForce RTX 2080 Ti.

4.6. Inference time

We report the *inference* time (embedding + WaffleIron + classification) of WaffleIron-48-256 on nuScenes and SemanticKITTI in Tab. 4. Note that, here, we used the function `torch.gather` instead of a matrix-vector multiplication with sparse matrices to implement $\mathrm{Inflat}(\cdot)$ and the batch normalization in Eq. (2) were merged with the first linear layer of the following MLP.

The inference time of WaffleIron-48-256 is comparable to other sparse convolution-based methods on nuScenes and a bit slower ($\times$ 1.7) than the well-known MinkUNet34 on SemanticKITTI. Note that the modified SPVCNN in 2DPASS (SPVCNN†) is wider and deeper on nuScenes than on SemanticKITTI, hence the faster running time on the latter.

Projection	nuScenes ($\rho = 60$cm)			
	Baseline	Reverse	Parallel	BEV
WaffleIron-12-256	75.2	75.0	73.4	74.8
WaffleIron-48-256	76.1	-	-	76.0
Projection	**KITTI ($\rho = 40$cm)**			
	Baseline	Reverse	Parallel	BEV
WaffleIron-12-256	62.6	60.9	61.2	63.3
WaffleIron-48-256	62.5	-	-	63.7
WaffleIron-48-256†	66.8	-	-	66.8

Table 5. Influence of the projection strategy on the performance of WaffleIron. We train each backbone on the training set of nuScenes or SemanticKITTI and compute the mIoU on the corresponding validation set. We report the average mIoU% obtained at the last training epoch of two independent runs. † indicates that the backbone was trained with instance cutmix and polarmix augmentations.

4.7. Other choices of projection strategy

We present in Tab. 5 the effect of using different projection strategies in our WI block. These strategies are the following. *Baseline* corresponds to the sequence of projections described in Sec. 3.2, i.e., used to produce all our results so far. *Reverse* consists in reversing the order of the projections used in *Baseline*. *Parallel* consists in performing three projections on (x, y), (x, z) and (y, z) in parallel at each layer. The projected feature maps are processed by different 2D FFNs. The resulting feature maps are then inflated, added together, and used as residual in (1). We choose to compare this projection strategy to the others while keeping the number of 2D convolutions fixed. The actual depth of the network with this strategy is thus divided by three. *BEV* consists in projecting on the (x, y) plane at all layers. All experiments are conducted in the same setting as in Sec. 4.4.

First, reversing the sequence of projections has almost no effect on nuScenes where the mIoU% decrease by 0.2 point. We notice however a decrease in mIoU on SemanticKITTI. We will see below that, on this dataset, projecting only on (x, y) permits to improve the performance in absence of strong augmentations. We suppose that starting by projecting on (x, y) has a positive effect thanks to, maybe, a better start in identifying the main structures.

Second, computing multiple projections in parallel is less optimal than computing them in a series: we loose -1.4 point and -1.8 point in mIou% with respect to the baseline on SemanticKITTI and nuScenes, respectively.

Finally, projecting only in BEV has a negligible impact on the average mIoU on nuScenes: we loose at most -0.4 point in mIou% with respect to the baseline sequence of projections. We explain this result because most structures and objects remain well identifiable in the bird's eye view in autonomous driving datasets. On SemanticKITTI, the baseline sequence of projections yields the same performance than the BEV projections only if strong data augmentations (instance cutmix and polarmix) are used during training. In absence of these augmentations, projecting only in bird's eye view might have played the role of a regularization which helped the generalization to unseen data.

5. Conclusion

We proposed WaffleIron, a novel and easy-to-implement 3D backbone for automotive point cloud semantic segmentation, which is essentially made of standard MLPs and dense 2D convolutions. We showed that its hyperparameters are easy to tune and that it can reach the mIoU of top entries on two autonomous driving benchmarks.

Thanks to the use of dense 2D convolutions, we foresee other potential applications where WaffleIron could be useful. In particular, the tasks semantic completion and or occupancy completion, see, e.g., [30, 31], where the WI layer could be used to densify the input point cloud.

Acknowledgments. We thank the Astra-vision team at Inria Paris for helpful discussions and insightful comments. We also acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-21-CE23-0032 (project MultiTrans). This work was granted access to the HPC resources of CINES under the allocation GDA2213 for the Grand Challenges AdAstra GPU made by GENCI.

References

[1] Angelika Ando, Spyros Gidaris, Andrei Bursuc, Gilles Puy, Alexandre Boulch, and Renaud Marlet. RangeViT: Towards Vision Transformers for 3D Semantic Segmentation in Autonomous Driving. In *CVPR*, 2023. 3

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *ICCV*, 2019. 1, 2, 5

[3] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *CVPR*, 2018. 6

[4] Alexandre Boulch, Gilles Puy, and Renaud Marlet. Fkaconv: Feature-kernel alignment for point cloud convolution. In *ACCV*, November 2020. 2

[5] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 2, 5

[6] Mingmei Cheng, Le Hui, Jin Xie, Jian Yang, and Hui Kong. Cascaded Non-local Neural Network for Point Cloud Semantic Segmentation. In *IROS*, 2020. 2

[7] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. (AF)2-S3Net: Attentive Feature Fusion With Adaptive Feature Selection for Sparse Semantic Segmentation Network. In *CVPR*, 2021. 3, 5, 6

[8] Jaesung Choe, Chunghyun Park, Francois Rameau, Jaesik Park, and In So Kweon. PointMixer: MLP-Mixer for Point Cloud Understanding. In *ECCV*, 2022. 3

[9] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *CVPR*, June 2019. 3

[10] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. SalsaNext: Fast, Uncertainty-Aware Semantic Segmentation of LiDAR Point Clouds. In *Advances in Visual Computing*, 2020. 3, 5, 6

[11] Martin Gerdzhev, Ryan Razani, Ehsan Taghavi, and Liu Bingbing. TORNADO-Net: mulTiview tOtal vaRiatioN semAntic segmentation with Diamond inceptiOn module. In *ICRA*, 2021. 3

[12] Yuenan Hou, Xinge Zhu, Yuexin Ma, Chen Change Loy, and Yikang Li. Point-to-Voxel Knowledge Distillation for LiDAR Semantic Segmentation. In *CVPR*, 2022. 1, 3, 5, 6, 7

[13] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. In *CVPR*, 2020. 2, 5

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016. 6

[15] Deyvid Kochanov, Fatemeh Karimi Nejadasl, and Olaf Booij. KPRNet: Improving projection-based LiDAR semantic segmentation. *arXiv:2007.12668*, 2020. 3, 5

[16] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical Transformer for LiDAR-Based 3D Recognition. In *CVPR*, 2023. 3

[17] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified Transformer for 3D Point Cloud Segmentation. In *CVPR*, 2022. 2

[18] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018. 2

[19] Jiale Li, Hang Dai, and Yong Ding. Self-distillation for robust lidar semantic segmentation in autonomous driving. In *ECCV*, 2022. 3, 5, 6

[20] Venice Erin Liong, Thi Ngoc Tho Nguyen, Sergi Widjaja, Dhananjai Sharma, and Zhuang Jie Chong. AMVNet: Assertion-based Multi-View Fusion Network for LiDAR Semantic Segmentation. *arXiv:2012.04934*, 2020. 3, 5, 6

[21] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking Network Design and Local Geometry in Point Cloud: A Simple Residual MLP Framework. In *ICLR*, 2022. 3

[22] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. In *IROS*, 2019. 3, 6

[23] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast Point Transformer. In *CVPR*, 2022. 2

[24] Jaehyun Park, Chansoo Kim, and Kichun Jo Soyeong Kim and. PCSCNet: Fast 3D semantic segmentation of LiDAR point cloud for autonomous car using point convolution and sparse convolution network. *Expert Systems with Applications*, 2023. 3, 5

[25] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*, 2017. 2

[26] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NeurIPS*, 2017. 2

[27] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. In *NeurIPS*, 2022. 3

[28] Haibo Qiu, Baosheng Yu, and Dacheng Tao. GFNet: Geometric Flow Network for 3D Point Cloud Semantic Segmentation. *Transactions on Machine Learning Research*, 2022. 3, 5, 6

[29] Ryan Razani, Ran Cheng, Ehsan Taghavi, and Liu Bingbing. Lite-HDSeg: LiDAR Semantic Segmentation Using Lite Harmonic Dense Convolutions. In *ICRA*, 2021. 5

[30] Christoph B. Rist, David Schmidt, Markus Enzweiler, and Dariu M. Gavrila. SCSSnet: Learning Spatially-Conditioned Scene Segmentation on LiDAR Point Clouds. In *IEEE Intelligent Vehicles Symposium*, 2020. 3, 9

[31] Luis Roldão, Raoul de Charette, and Anne Verroust-Blondet. LMSCNet: Lightweight Multiscale 3D Semantic Completion. In *3DV*, 2020. 9

[32] Radu Alexandru Rosu, Peer Schütt, Jan Quenzel, and Sven Behnke. LatticeNet: Fast point cloud segmentation using permutohedral lattices. In *Robotics: Science and Systems*, 2020. 2

[33] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. TorchSparse: Efficient Point Cloud Inference Engine. In *MLSys*, 2022. 1

[34] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV*, 2020. 1, 3, 5

[35] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *ICCV*, October 2019. 2, 5

[36] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP Architecture for Vision. In *NeurIPS*, 2021. 1, 3

[37] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going Deeper With Image Transformers. In *ICCV*, 2021. 3

[38] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions On Graphics*, 2019. 2, 4

[39] Aoran Xiao, Jiaxing Huang, Dayan Guan, Kaiwen Cui, Shijian Lu, and Ling Shao. PolarMix: A General Data Augmentation Technique for LiDAR Point Clouds. In *NeurIPS*, 2022. 7, 8

[40] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV*, 2020. 3

[41] Jianyun Xu, Ruixiang Zhang, Jian Dou, Yushi Zhu, Jie Sun, and Shiliang Pu. RPVNet: A Deep and Efficient Range-Point-Voxel Fusion Network for LiDAR Point Cloud Segmentation. In *ICCV*, 2021. 3, 5, 6, 8

[42] Xu Yan, Jiantao Gao, Jie Li, Ruimao Zhang, Zhen Li, Rui Huang, and Shuguang Cui. Sparse single sweep lidar point cloud segmentation via learning contextual shape priors from scene completion. In *AAAI*, 2021. 5

[43] Xu Yan, Jiantao Gao, Chaoda Zheng, Chao Zheng, Ruimao Zhang, Shuguang Cui, and Zhen Li. 2DPASS: 2D Priors Assisted Semantic Segmentation on LiDAR Point Clouds. In *ECCV*, 2022. 1, 5, 6, 8

[44] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. PointASNL: Robust Point Clouds Processing Using Nonlocal Neural Networks With Adaptive Sampling. In *CVPR*, 2020. 2

[45] Maosheng Ye, Rui Wan, Tongyi Cao Shuangjie Xu, and Qifeng Chen. Efficient Point Cloud Segmentation with Geometry-Aware Sparse Networks. In *ECCV*, 2022. 3, 5

[46] Feihu Zhang, Jin Fang, Benjamin Wah, and Philip Torr. Deep FusionNet for Point Cloud Semantic Segmentation. In *ECCV*, 2020. 3

[47] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. PolarNet: An Improved Grid Representation for Online LiDAR Point Clouds Semantic Segmentation. In *CVPR*, 2020. 3, 6

[48] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H.S. Torr, and Vladlen Koltun. Point Transformer. In *ICCV*, 2021. 2

[49] Lin Zhao, Siyuan Xu, Liman Liu, Delie Ming, and Wenbing Tao. SVASeg: Sparse Voxel-Based Attention for 3D LiDAR Point Cloud Semantic Segmentation. *Remote Sens.*, 2022. 3, 5, 6

[50] Yangheng Zhao, Jun Wang, Xiaolong Li, Yue Hu, Ce Zhang, Yanfeng Wang, and Siheng Chen. Number-Adaptive Prototype Learning for 3D Point Cloud Semantic Segmentation. In *ECCVW*, 2022. 3, 5

[51] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation. In *CVPR*, 2021. 1, 3, 5, 6