IEEE International Conference on Robotics and Automation (ICRA), Philadelphia (PA), USA, May 2022.

Abstract Flow for Temporal Semantic Segmentation on the Permutohedral Lattice

Peer Schütt, Radu Alexandru Rosu and Sven Behnke

***Abstract*— Semantic segmentation is a core ability required by autonomous agents, as being able to distinguish which parts of the scene belong to which object class is crucial for navigation and interaction with the environment. Approaches which use only one time-step of data cannot distinguish between moving objects nor can they benefit from temporal integration. In this work, we extend a backbone LatticeNet to process temporal point cloud data. Additionally, we take inspiration from optical flow methods and propose a new module called Abstract Flow which allows the network to match parts of the scene with similar abstract features and gather the information temporally. We obtain state-of-the-art results on the SemanticKITTI dataset that contains LiDAR scans from real urban environments. We share the PyTorch implementation of TemporalLatticeNet at** https://github.com/AIS-Bonn/temporal_latticenet.

I. Introduction

Semantic segmentation of 3D point clouds is the process of predicting a class for every point in the cloud. This is especially challenging for 3D point clouds, due to under-sampling of the scene and a lack of explicit structure in the cloud.

Current approaches rely on projecting the 3D point cloud to 2D images [1], [2] or embed it into a dense volumetric grid [3], [4]. These approaches are, however, suboptimal since 2D images are not a natural representation for 3D point clouds while volumetric grids can be slow and memory intensive. We propose to use the permutohedral lattice as an alternative representation to apply convolutions on a data structure that more closely resembles the input and can process full scans at interactive speeds.

In this work, we consider the point clouds as being recorded continuously by a sensor like a laser scanner or a depth camera. Processing individual point clouds ignores the temporal information, rendering the agent incapable of distinguishing between moving and stationary objects or integrating evidence over time [1], [2], [5].

LatticeNet [6] is a efficient network based on the permutohedral lattice that has shown state-of-the-art results for semantic segmentation of point clouds. It has recently been extended to process temporal information [5]. However, the temporal fusion was performed only with simple recurrence using fully connected layers.

We present TemporalLatticeNet, an extension to LatticeNet that utilizes recurrent processing by evaluating Long Short-Term Memory (LSTM) [7] and Gated Recurrent Units (GRU) [8] as modules for temporal fusion. Additionally, we propose a novel module called Abstract Flow (AFlow) which can gather temporal information by matching abstract features in lattice space.

All authors are with Autonomous Intelligent Systems Group, Computer Science VI, University of Bonn, Germany, {schuett, rosu, behnke}@ais.uni-bonn.de.

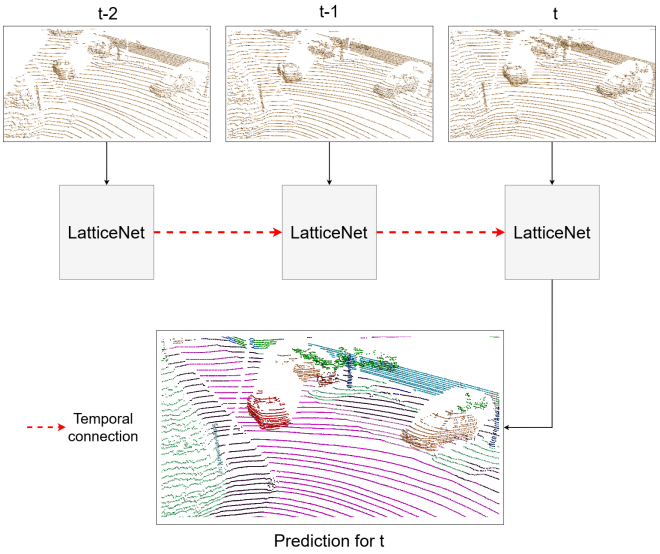


Fig. 1: We use multiple consecutive point clouds as input to our backbone network. The feature matrices of different timesteps are fused together to allow information propagation through time. The per-point semantic classes of the last point cloud in the sequence are predicted.

We show competitive results on the temporal SemanticKITTI dataset [9], but with a faster processing speed than other methods.

Our contributions presented in this work include:

- Extensions to the LatticeNet architecture in order to improve recurrent processing,
- Novel module inspired by optical flow methods adapted to the permutohedral lattice that improves the aggregation of temporal information, and
- Competitive results on the SemanticKITTI dataset but with faster processing speed than other methods.

II. Related Work

In this section, temporal semantic segmentation will be discussed. For an overview on general semantic segmentation of point clouds without temporal information we refer to Rosu et al. [5]

Temporal semantic segmentation approaches can be divided in two types, depending on their input: i) methods which process sequences of clouds in a recurrent manner

to predict the class labels and ii) methods that accumulate multiple clouds into one single cloud to solve the task as a single-frame segmentation.

The advantage of the first type of methods is that at each time-step only the current point cloud needs to be processed as the past information has already been summarized and stored in memory. However, they require more complex architectures than the methods that naïvely segment a large aggregated point cloud.

Shi et al. [1] present SpSequenceNet, a U-Net based architecture for temporal semantic segmentation. Two consecutive point clouds are voxelized and given as input to a shared encoder network. Vertical connection are added between the two timesteps in order to gather temporal information before decoding the representation into class probabilities of the last point cloud. They designed two modules to combine the information from the two consecutive point clouds: the Cross-frame Global Attention (CGA) and the Cross-frame Local Interpolation (CLI) module. CGA is inspired by self-attention and allows the network to gather and fuse global information from the previous cloud. CLI allows to attend to local information by fusing the features from nearby points with a dynamic per-point weighting. In contrast to our approach, Shi et al. [1] can only process sequences of length two and they voxelize the point cloud, which leads to a loss of information and discretization artifacts. Additionally, CLI per-point weighting is calculated using distances to k-nearest neighbors in 3D space, while our approach calculates the distance in the abstract feature space using the 1-hop neighborhood of a permutohedral lattice.

Duerr et al. [2] present their recurrent architecture TemporalLidarSeg that uses temporal memory alignment to predict the semantic labels of sequences of point clouds. Their sequences have the potential of unlimited length. They project the 3D point clouds onto the 2D plane and use a U-net backbone network to output per-frame feature matrices. These feature matrices are then combined with the feature matrices of the hidden state using the temporal memory unit, which uses the real-world poses of each point cloud to compute the transformation from the coordinate system of the hidden to the current state. The 2D semantic segmentation is then projected back into the 3D representation. Similar to our approach, they require the poses of the point clouds, but they additionally need the mapping from 3D to 2D and therefore don't work directly on the point cloud. Additionally, our approach uses multiple fusion points in contrast to one in their approach.

Kernel Point Convolution (KPConv) [10] operates directly on the point clouds by assigning convolution weights to a set of kernel points located in Euclidean space. Points in the vicinity of these kernel points are weighted and summed together to feature vectors. The kernel function is defined as the correlation between the location of the kernel point and the distance to the points in the radius neighborhood. To be robust to varying densities, the input clouds are subsampled at every layer of the network using a grid subsampling and the radius neighborhood of the convolution is adapted. The kernel points are usually static, but can also be learned by the network itself to adapt to more challenging tasks. Due to memory limitations, their approach cannot process one full point cloud for outdoor scenes. Therefore, Thomas et al. fit multiple overlapping spheres into the point cloud and evaluate them individually. The final results are generated by a voting scheme. In contrast to our method, KPConv [10] performs temporal semantic segmentation by accumulating all clouds of the sequence into one large point cloud and uses no recurrent architecture.

DarkNet53Seg [9] and TangentConv [11] were used as the two baseline networks for the segmentation of 4D point clouds in the SemanticKITTI [9] dataset. The input for these were accumulated clouds of the sequences. DarkNet53Seg [9] is an extension of SqueezeSeg [12] — a U-Net architecture with skip connections that uses the spherical projection of LiDAR point clouds to predict a point-wise label map that is refined by a conditional random field and subsequent clustering. TangentConv [11] is based on the notion of tangent convolution — a different approach to construct convolutional networks on surfaces that assumes that the data is sampled from locally Euclidean surfaces. The input points are projected onto local tangent planes which are used as 2D grids for convolutions. Based on this input, Tatarchenko et al. [11] design a U-type network with skip connections. In contrast to our approach, both DarkNet53Seg and TangentConv were designed to output dense per-point predictions for single point clouds and contain no recurrent connections.

III. Fundamentals

We use bold upper-case characters to denote matrices and bold lower-case characters to denote vectors. We denote with point p a single element from the point cloud and with vertex v an element from the permutohedral lattice.

The points of a cloud are defined as a tuple $p = (\mathbf{g}_p, \mathbf{f}_p)$, with $\mathbf{g}_p \in \mathbb{R}^d$ denoting the coordinates of the point and $\mathbf{f}_p \in \mathbb{R}^{f_d}$ representing the features stored at point p (normals, reflectance, etc.). The full point cloud containing m points is denoted by $P = (\mathbf{G}, \mathbf{F})$ with $\mathbf{G} \in \mathbb{R}^{m \times d}$ denoting the positions matrix and $\mathbf{F} \in \mathbb{R}^{m \times f_d}$ the feature matrix.

The vertices of the d-dimensional permutohedral lattice [6], [13] are defined as a tuple $v = (\mathbf{c}_v, \mathbf{x}_v)$, with $\mathbf{c}_v \in \mathbb{Z}^{(d+1)}$ denoting the coordinates of the vertex and $\mathbf{x}_v \in \mathbb{R}^{v_d}$ representing the values stored at vertex v. A full lattice contains k vertices and is denoted with $V = (\mathbf{C}, \mathbf{X})$, with $\mathbf{C} \in \mathbb{Z}^{k \times (d+1)}$ representing the coordinate matrix and $\mathbf{X} \in \mathbb{R}^{k \times v_d}$ the value matrix. For $d = 3$ the input space is tessellated into uniform tetrahedra. We denote the set of neighbors of vertex v with $N(v)$. The vertices of the permutohedral lattice are stored in a sparse manner using a hash map. Hence, we only allocate the simplices that contain the 3D surface of interest.

Tetrahedras scale linearly in the number of vertices and not quadratically like cubical voxels. This allows for fast interpolation of data from the point cloud to the lattice and backwards. For further details we refer to Rosu et al. [6].

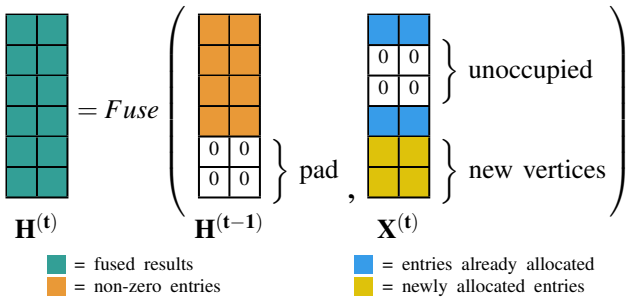


Fig. 2: Temporal fusion: The feature matrix from the previous time-step $\mathbf{H}^{(t-1)}$ is zero-padded in order to account for the new vertices that were allocated at the current time-step $\mathbf{X}^{(t)}$. The feature matrices are afterwards fused by the chosen recurrent layer.

IV. Architecture

Our recurrent neural network (RNN) TemporalLatticeNet is an extension of LatticeNet [6]. LatticeNet is extended with recurrent connections at multiple resolutions on which temporal information is allowed to flow (Fig. 3).

A. Method

Input to our network is a sequence of clouds $P = (P_0, P_1, ..., P_{n-1})$, where $P_i = (\mathbf{G}, \mathbf{F})$ with $n \in \mathbb{N}^+$ and $0 \leq i < n$. We refer to n as the sequence length. The network outputs the likelihood for each possible class for every point $p \in P_n$ (Fig. 1). We assume that the points have been transformed in a common reference frame. The positions $\mathbf{G}$ are scaled by a factor $\boldsymbol{\sigma} \in \mathbb{R}^d$ as $\mathbf{G}_s = \mathbf{G}/\boldsymbol{\sigma}$ which controls the influence area of the permutohedral lattice. If not otherwise stated, we refer to $\mathbf{G}_s$ as $\mathbf{G}$. The matrix $\mathbf{F}$ denoting the per-point features contains the reflectance value from the LiDAR scanner or is set to zeros in the case of a sensor which doesn't output reflectance.

We insert recurrent connections at various points of the LatticeNet architecture (Fig. 3) where the states of two lattices $V^{(t-1)}$ and $V^{(t)}$ have to be fused. We refer to the feature matrix of each lattice as $\mathbf{X}^{(t-1)}$ and $\mathbf{X}^{(t)}$, respectively, as state of the network. To compute the hidden state $\mathbf{H}^{(t)}$, we fuse the previous hidden state $\mathbf{H}^{(t-1)}$ together with the current state of the network $\mathbf{X}^{(t)}$. For this, a correspondence between the coordinate matrices $\mathbf{C}$ of both lattices has to be known, because they define the order in which the feature vectors $\mathbf{x}_v \in \mathbf{X}$ are saved. This correspondence is achieved by transforming the point clouds into a common frame and using its hash map. By keeping the hash map the same for the whole sequence, the Distribute operation of LatticeNet maps 3D points to the same coordinate vectors $\mathbf{c}$ across timesteps and allows comparing the feature vectors $\mathbf{x}$ with each other. Vertices corresponding to previously unknown areas in the input are inserted at the end of the matrices $\mathbf{C}$ and $\mathbf{X}$ (Fig. 2).

B. Position of the Recurrence

Our RNN is a many-to-one deep RNN whose recurrent layers are positioned at different layers of the network.

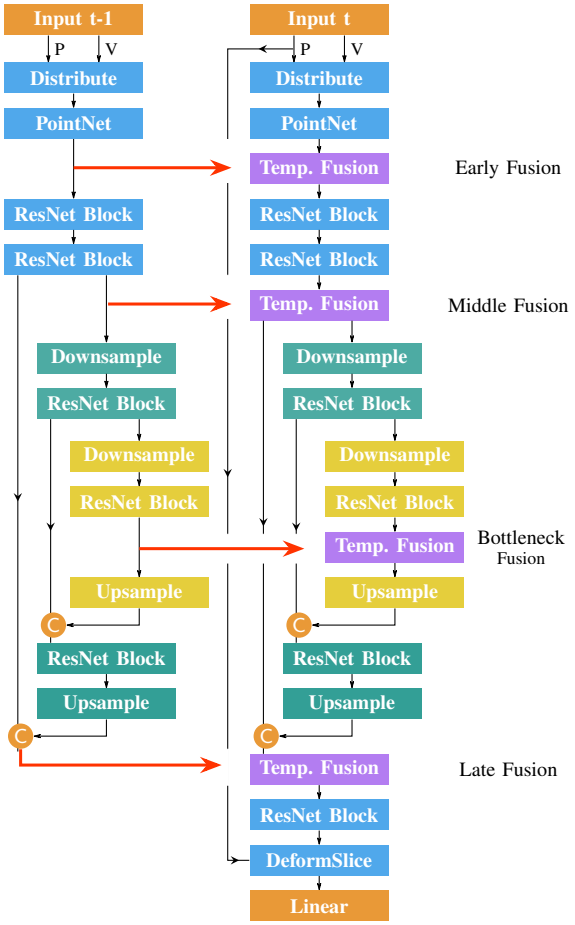


Fig. 3: Recurrent architecture: The features from previous time-steps are fused in the current time-step at multiple levels of the network. This allows the network to distinguish dynamic and static objects. Our addition to the LatticeNet architecture are the temporal connections (→) and the temporal fusion blocks (■).

Four promising positions for the recurrent connections were chosen: Early Fusion, Middle Fusion, Bottleneck Fusion, and Late Fusion as shown in Fig. 3.

C. Recurrent Layers

In this section, multiple different recurrent layers are presented and discussed that can be used to fuse $\mathbf{H}^{(t-1)}$ and $\mathbf{X}^{(t)}$. For the last point cloud P_t of the sequence, the previous hidden state $\mathbf{H}^{(n-1)}$ is used to generate the network's prediction. For $t = 0$, no computation is performed with $\mathbf{H}^{(0)} = \mathbf{X}^{(0)}$. In order to fuse $\mathbf{H}^{(t-1)}$ and $\mathbf{X}^{(t)}$, they need the same shape and therefore $\mathbf{H}^{(t-1)}$ is padded with zeros (Fig. 2).

Long short-term memory (LSTM): LSTMs [7] are often used to counteract the problem of vanishing or exploding gradients in sequence learning and show good results for these problems [14]. Therefore we chose them as one recurrent module for our network.

Gated recurrent units (GRU): As an extension to LSTMs, GRUs were introduced by Cho et al. [15]. They

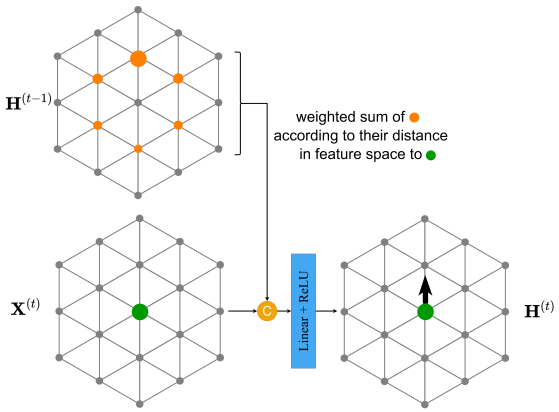


Fig. 4: Abstract Flow module: Features from the one-hop neighborhood of the previous timestep $\mathbf{H}^{(t-1)}$ are compared with the center feature $\mathbf{x}_v$ at the current time. A weighted sum is computed based on the distance. The result is concatenated with $\mathbf{x}_v$ and fused using an MLP. A direction can also be established in lattice space between the center feature and the most similar feature from the previous time-step which gives a coarse notion of the movement in the 3D scene as indicated by the arrow. Please note that this figure depicts a 2D example.

only use an input and a hidden gate and no cell state like LSTMs and are therefore possibly better suited for our task. The performance of GRUs is comparable to LSTMs, while having a lower memory consumption [14].

Abstract Flow (AFlow): This module is inspired by the CLI module presented by Shi et al. [1] that aims at combining local information and capturing temporal information between two point clouds. Our AFlow module can be seen as a convolution with an adaptive convolution kernel. This resembles the ideas presented in Pixel-Adaptive Convolution [16]. Therefore, AFlow is designed to extract partial differences between $\mathbf{X}^{(t)}$ and $\mathbf{H}^{(t-1)}$. First, the nearest neighbors $N(v)$ of each lattice vertex v with $v \in V^{(t)}$ in the lattice from the previous timestep $V^{(t-1)}$ are found (Fig. 4). They are used to generate a new local feature vector to fuse temporal information and at the same time summarize the surrounding area from the previous timestep. The neighbors $N(v)$ for each lattice vertex are given by the one-hop neighborhood. The neighboring feature vectors from $V^{(t-1)}$ are denoted with $N_{\mathbf{H}}(v)$. For $d = 3$, the number of neighbors is given by $|N_{\mathbf{H}}(v)| = 8$. The feature vectors of the vertices in $N_{\mathbf{H}}(v)$ are weighted according to their distance to the feature vector $\mathbf{x}_v$ in $\mathbf{X}^{(t)}$. The weight is calculated as

$$\forall i \in N_{\mathbf{H}}(v) : w_i = (\alpha - min(dist(\mathbf{x}_v, \mathbf{h}_i), \alpha)) \cdot \beta, \quad (1)$$

where α and β are learnable parameters that are initialized with $\alpha, \beta = 0.1$. Parameter α impacts the maximum distance a neighbor can have from the feature vector we are evaluating at the moment, while β controls the maximum value of the resulting feature. We denote with *dist* the Euclidean distance between the feature vectors $\mathbf{x}_v$ and $\mathbf{h}_i$. The AFlow feature matrix $\mathbf{L}$ has the same dimensionality as $\mathbf{X}^{(t)}$. Its feature vectors $\mathbf{l}_v$ are calculated as

$$\mathbf{l}_v = \sum_{i=1}^{8} \mathbf{h}_i \cdot w_i. \quad (2)$$

$\mathbf{L}$ is then concatenated with $\mathbf{X}^{(t)}$ along the channels dimension and passed through a linear layer followed by a non-linearity to obtain the new feature matrix $\mathbf{H}^{(t)}$.

The weights calculated in AFlow measure the similarity between features at different timesteps. Similar features that move through time correspond to moving objects in 3D space. We visualize in Fig. 4 the design of the module and the direction between the center feature and the most similar feature at the previous timestep. Further experiments with the directionality are discussed in Sec. V-E.

D. Network Architectures

The recurrent layers and the recurrence positions can be combined arbitrarily. To distinguish the different architectures, the following notation is used: The four recurrence layers are separated by a hyphen, e.g. GRU-GRU-AFlow-GRU refers to a network that has a GRU for the early, middle and late fusion and an AFlow module for the bottleneck fusion.

V. Experiments

We evaluate our network architectures by calculating the mean Intersection-over-Union (mIoU) on the SemanticKITTI [9] dataset. It provides 3D LiDAR-scans from real urban environments and semantic per-point annotations for moving and non-moving classes. We additionally evaluate the impact of the per-point feature matrix $\mathbf{F}$ onto the network: The two possibilities are an empty feature matrix, which forces the network to predict based only on the point positions, and an $\mathbf{F}$ filled with the points' reflectance values.

A. Implementation

All lattice operators with forward and backward passes are implemented on the GPU [6] and exposed to PyTorch [17]. All convolutions are pre-activated using a ReLU unit [18], [19]. For the lattice scale $\boldsymbol{\sigma} = 0.6$ was used and the batch size was chosen as 1.

The models were trained using the Adam optimizer [20], [21] with a learning rate of 0.001 and a weight decay of 10^{-4}. The learning rate was reduced by a cosine annealing scheduler [22]. The number of epochs between two restarts was chosen as three since less frequent restarts tended to cause over-fitting.

B. SemanticKITTI Dataset

The SemanticKITTI [9] dataset contains semantically annotated LiDAR scans from the KITTI dataset [23]. The annotations are done for a total of 19 different classes in the single scan task and 25 different classes for the multiple scans task. The scans vary in size from 82K to 129K points with a total of 4,549 million points annotated. In addition to the x, y, and z coordinates, the reflectance value for each point is given.

TABLE I: Results on SemanticKITTI for different versions of our own architecture. The network with the best result is used in Tab. II for comparisons.

Approach	mIoU	with reflectance
LSTM-LSTM-AFlow-LSTM	46.7	✓
GRU-GRU-AFlow-AFlow	46.9	✓
GRU-GRU-AFlow-GRU	**47.1**	✓
GRU-GRU-/-GRU	44.1	✓
GRU-GRU-AFlow-GRU	42.8	x
LatticeNet-MLP [5]	45.2	x

We process each scan entirely without any cropping. The training data is augmented with random rotations around the height-axis, mirroring, translation around the other two axes, and random per-point noise.

C. Generating Predictions

The hyperparameters sequence length n and cloud scope s, with $s \in \mathbb{N}$, have to be chosen for the dataset. The sequence length n defines the cardinality of the input for the network and was chosen as $3 \leq n \leq 5$. We observed that $n < 3$ doesn't allow the network to aggregate enough information, while $n > 5$ leads to memory and time constraints. A sequence length of $n = 4$ worked best for our models. For SemanticKITTI $s = 3$ was chosen, which means that between clouds in the sequence P two clouds in the dataset are skipped. We've found that using directly consecutive clouds ($s = 1$) can negatively impact the segmentation results since the input clouds are too similar and therefore choose for all our experiments a cloud scope $s = 3$.

It is to be noted that during training we need to keep all input clouds in memory in order to perform back-propagation through time, while during inference we evaluate only the cloud at the current time-step since the features from previous time-step are already stored in memory. This recursive formulation yields inference speed that is similar to the original LatticeNet architecture which operated on a single-scan.

D. Quantitative Results

We evaluated five different variants of our architectures on the test set of SemanticKITTI and report their resulting mIoU in Tab. I. We observe that the AFlow module resulted in an improved mIoU of 3.0 points, compared to the base-network that only utilizes GRUs. Adding another AFlow layer at the late fusion instead of an GRU resulted in slightly worse results, which can be explained by the lower feature dimension and therefore a lack of comparability in feature space.

Without using the reflectance as input, the result deteriorated significantly. The reason for this might be that the reflectance is a very useful feature for distinguishing similar feature vectors. This applies to the other recurrent blocks as well, albeit not as severely. Additionally, we evaluate the performance of the original LatticeNet [6] on temporal data. For this experiment, we accumulated all clouds of a sequence and used this cloud as input. The results were worse than all comparable temporal networks.

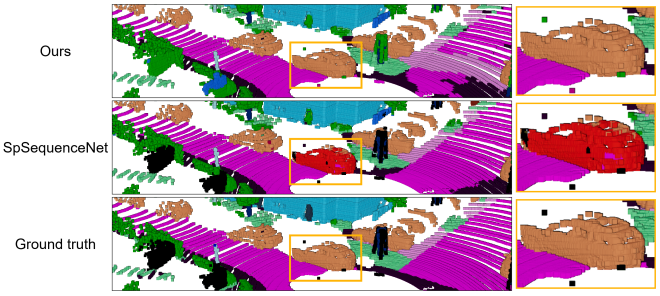


Fig. 5: In comparison to SpSequenceNet, we are able to better segment stationary (■) and moving cars (■) in small streets with a high number of cars in the vicinity. SpSequenceNet [1] was the best network that provided pretrained models and was therefore chosen as the comparison. In this example, it is able to better distinguish the road (■) from a sidewalk (■). Unlabeled (■) points are ignored during training.

To compare our network's results to the state-of-the-art on SemanticKITTI, we chose the best performing network GRU-GRU-AFlow-GRU. The IoU for all 25 classes are presented in Tab. II. We improved performance in relation to our previously published architecture LatticeNet-MLP [6].

Our best network has outstanding performance in the segmentation of the classes 'vegetation', 'terrain', 'pole', and 'traffic sign'. An explanation for the results on 'traffic sign' and 'pole' is that they are only represented by a small number of points and having multiple clouds can help with a better segmentation quality.

Our network's performance is comparable to Temporal-LidarSeg [2], but is outperformed by KPConv [10] with a mIoU that is smaller by 4.1 points. It is to be noted that KPConv [10] cannot process the whole cloud due to memory constraints, but has to rely on fitting multiple spheres into the cloud to ensure that each point is tested multiple times. The final result is then determined by a voting scheme, in contrast to our approach that processes the whole cloud at once with a single prediction. TemporalLidarSeg [2], on the other hand, relies on the spherical projection of the 3D cloud to perform 2D operations, while our approach is able to utilize the 3D cloud without any projection.

We present the performance results in Tab. III. The measurements were taken on a NVIDIA GeForce RTX 3090 and the inference time was measured on the validation set. Each AFlow module increases the inference time and memory consumption, caused by the high number of weights in the AFlow module and the distance calculation per vertex. We are able to segment the cloud faster than KPConv [10], because we are able to reuse feature matrices from previous segmentations due to our recurrent architecture.

TABLE II: State-of-the-art results on SemanticKITTI in comparison to our best performing network.[1]

Approach	mIoU	car	truck	other-vehicle	person	bicyclist	motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic sign	moving-car	moving-bicyclist	moving-person	moving-motorcyclist	moving-other-vehicle	moving-truck
TangentConv [11]	34.1	84.9	2.0	18.2	21.1	18.5	1.6	83.9	38.3	64.0	15.3	85.8	49.1	79.5	43.2	56.7	36.4	31.2	40.3	1.1	6.4	1.9	**30.1**	**42.2**
DarkNet53Seg [9]	41.6	84.1	30.4	32.9	20.2	20.7	7.5	91.6	**64.9**	75.3	**27.5**	85.2	56.5	78.4	50.7	64.8	38.1	53.3	61.5	14.1	15.2	0.2	28.9	37.8
SpSequenceNet [1]	43.1	88.5	24.0	26.2	29.2	22.7	6.3	90.1	57.6	73.9	27.1	91.2	**66.8**	84.0	66.0	65.7	50.8	48.7	53.2	41.2	26.2	36.2	2.3	0.1
KPConv [10]	**51.2**	**93.7**	44.9	**47.2**	**42.5**	**38.6**	**21.6**	86.5	58.4	70.5	26.7	90.8	64.5	**84.6**	**70.3**	66.0	57.0	53.9	**69.4**	**67.4**	**67.5**	47.2	4.7	5.8
TemporalLidarSeg [2]	47.0	92.1	**47.7**	40.9	39.2	35.0	14.4	**91.8**	59.6	**75.8**	23.2	89.8	63.8	82.3	62.5	64.7	52.6	**60.4**	68.2	42.8	40.4	12.9	12.4	2.1
LatticeNet-MLP [5]	45.2	91.1	16.8	25.0	29.7	23.1	6.8	89.7	60.5	72.5	26.9	**91.9**	64.7	82.9	65.0	63.7	54.7	47.1	54.8	44.6	49.9	**64.3**	0.6	3.5
Ours	47.1	91.6	35.4	36.1	26.9	23.0	9.4	91.5	59.3	75.3	**27.5**	89.6	65.3	**84.6**	66.7	**70.4**	**57.2**	**60.4**	59.7	41.7	9.4	48.8	5.9	0.0

TABLE III: Average time used by the forward pass and the maximum memory used during training.

	SemanticKITTI [ms]	[GB]
LSTM-LSTM-AFlow-LSTM	151	20
GRU-GRU-AFlow-GRU	154	20
GRU-GRU-AFlow-AFlow	159	22
GRU-GRU-/-GRU	140	18
KPConv [10]	225	15
SpSequenceNet [1]	477	3

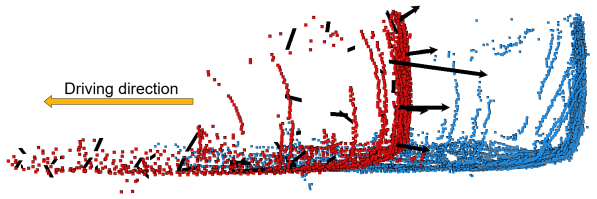
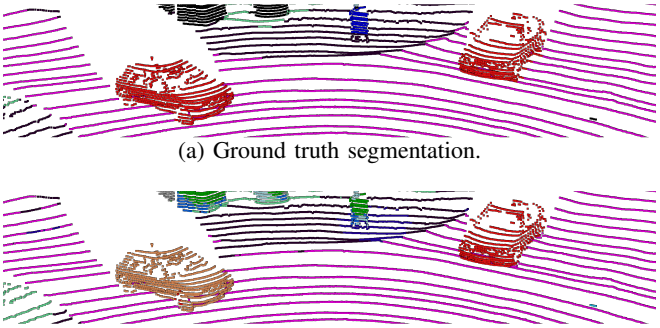


Fig. 6: Visualization of the AFlow module on the segmentation: Birds-eye view of a car at two different time-steps. The correspondence between the car in the previous timestep ( ) and the current timestep ( ) is made by the module and therefore the car is correctly segmented as moving-car.



(a) Ground truth segmentation.

(b) Predicted segmentation.

Fig. 7: Failure case: The prediction fails for the car on the left side, because it is predicted as car ( ) instead of moving-car ( ). The reason for this is that the car is waiting at the crossroads for many timesteps.

E. Qualitative Results

We present a visual comparison of the segmentation quality of our method with SpSequenceNet in Fig. 5.

In order to analyze the effects of the AFlow model, we mapped the previously mentioned directionality to 3D space in order to show a coarse direction of movement of the 3D objects. Lattice vertices are approximated in 3D by the average of the 3D points that contribute to them. In Fig. 6, we show one car at two different timesteps. For each lattice vertex in 3D, we draw an arrow showing the direction of the most similar feature from the current time towards the previous timestep. We see that for a car driving towards the left, the directionality from AFlow corresponds to the inverse of the driving direction.

A failure case of our architecture is moving objects that are waiting/standing still for a duration that exceeds our temporal scope. An example of a car that is waiting at a crossroad is presented in Fig. 7b. This shouldn't result in problems for an autonomous agent that takes actions based on this segmentation, because the object actually is standing still and is correctly classified as moving once it starts driving again.

VI. Conclusion

We presented an extension to the original LatticeNet [6] for temporal semantic segmentation. We evaluate different recurrence modules and propose a novel Abstract Flow module that better integrates temporal information. On the SemanticKITTI dataset we achieve comparative results with other baselines while running faster and being able to process the full point cloud at once.

Acknowledgment

This work has been funded by the German Federal Ministry of Education and Research (BMBF) in the project "Kompetenzzentrum: Aufbau des Deutschen Rettungsrobotik-Zentrums" (A-DRZ).

[1]We only compare to already published approaches.

REFERENCES

[1] H. Shi, G. Lin, H. Wang, T.-Y. Hung, and Z. Wang, "SpSequenceNet: Semantic Segmentation Network on 4D Point Clouds," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] F. Duerr, M. Pfaller, H. Weigel, and J. Beyerer, "LiDAR-based Recurrent 3D Semantic Segmentation with Temporal Memory Alignment," in *Intl. Conf. on 3D Vision (3DV)*. IEEE, 2020, pp. 781–790.

[3] L. Tchapmi, C. Choy, I. Armeni, J. Gwak, and S. Savarese, "SEGCloud: Semantic segmentation of 3D point clouds," in *Intl. Conf. on 3D Vision (3DV)*. IEEE, 2017, pp. 537–547.

[4] D. Rethage, J. Wald, J. Sturm, N. Navab, and F. Tombari, "Fully-convolutional point networks for large-scale point clouds," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 596–611.

[5] R. A. Rosu, P. Schütt, J. Quenzel, and S. Behnke, "LatticeNet: Fast Spatio-Temporal Point Cloud Segmentation Using Permutohedral Lattices," *Autonomous Robots (AURO)*, vol. 46(I), pp. 45–60, 2022.

[6] ——, "LatticeNet: Fast Point Cloud Segmentation Using Permutohedral Lattices," *Robotics: Science and Systems (RSS)*, 2020.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014, pp. 103–111.

[9] 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 *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019.

[10] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "KPConv: Flexible and Deformable Convolution for Point Clouds," *IEEE Int. Conf. on Computer Vision (ICCV)*, 2019.

[11] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent convolutions for dense prediction in 3D," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 3887–3896.

[12] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSeg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3D lidar point cloud," in *IEEE Int. Conference on Robotics and Automation (ICRA)*, 2018, pp. 1887–1893.

[13] A. Adams, J. Baek, and M. A. Davis, "Fast high-dimensional filtering using the permutohedral lattice," in *Computer Graphics Forum*, vol. 29, no. 2, 2010, pp. 753–762.

[14] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv:1412.3555*, 2014.

[15] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[16] H. Su, V. Jampani, D. Sun, O. Gallo, E. Learned-Miller, and J. Kautz, "Pixel-adaptive convolutional neural networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 166–11 175.

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 630–645.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[21] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv:1711.05101*, 2017.

[22] Loshchilov, Ilya and Hutter, Frank, "SGDR: Stochastic gradient descent with warm restarts," in *International Conference on Learning Representations, (ICLR)*, 2017.

[23] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.