3D Perception & Learning-Based Data Fusion

3D Lidar Segmentation on SemanticKITTI

Trimigno Giuseppe
giuseppe.trimigno@studenti.unipr.it

October 2023

1 Introduction

1.1 LiDAR

In this project, we focus on *LiDAR* data. *LiDARs* (*LIght Detection And Ranging*), or *Laser Range Sensors*, are exteroceptive (i.e. they measure something from the environment) and active (i.e. they emit their energy and measure the reaction from the environment) sensors, which are widely used for range distance measurement in autonomous driving, despite their high complexity and cost, because they provide high resolution data.
Basically, they are composed of a *transmitter*, which illuminates a target with a laser beam, and a *receiver*, which detects the time needed for a round trip, in order to estimate the range.
Furthermore, *LiDARs* are essentially based two operating principles: (1) Pulsed laser (the standard today), which directly measure the elapsed time with a single transmission; (2) Phase shift measurement on multiple transmissions to produce a range estimation (easier but not as good as the previous one).
In this sensor, the uncertainty (i.e. the error) on the estimated range is inversely proportional to the square of the received signal amplitude.

1.2 3D Semantic Segmentation

The task chosen for this project is *3D Semantic Segmentation*, in which we want to infer the label of each three-dimensional point in the point clouds.
This task is quite hard, especially because of the density and the intensity of objects, which can vary a lot, e.g. closer objects will have a dense resolution (more points), while objects further away are really sparse (less points), or challenges related to changes in sensing conditions etc.
These drawback are mainly related to intrinsic characteristics of *LiDAR* sensors, e.g. they have a shorter range, and are less robust against adverse weather conditions (fog, snow, wet ground) or external disturbances (motion blur, beam missing), compared to other sensors, like *Radars*.
We can split *3D Semantic Segmentation* into two subsets: (1) *Single scan evaluation*, in which we don't distinguish between moving and non-moving objects, i.e. they are all mapped to a single class, as in our case; (2) *Multiple scan evaluation*, where we distinguish between moving and non-moving objects, so it is harder than the other, since the model has also to decide if something is or not dynamic.
As evaluation metric, we chosen the most common one for this task, i.e. *mean Jaccard* (or *mean IoU - Intersection-over-Unit*), mathematically defined as $\frac{1}{C}\sum_{k=1}^{C}\frac{TP_k}{TP_k+FP_k+FN_k}$.

2 Dataset

2.1 Classical data

Based on previous considerations, we chosen *SemanticKITTI* as dataset. It is a large scale dataset, directly obtained from KITTI benchmark suite, by manually infer annotations for each point of LiDAR scans in all 22 scenes.
Specifically, the dataset is split in accordance with the following procedure:

- *Training set*: contains all sequences from 00 to 10, except for sequence 08 (see below), for a total of $\approx$ 20.000 samples, each one composed of fully labeled points;
- *Validation set*: contains the sequence 08, composed of 4.071 samples, each one with fully labeled points;
- *Test set*: contains all sequences from 11 to 21, but labels are not publicly available.

The dataset is composed of 28 different classes, but we used 20 for our training and testing (as suggested in the official SemanticKITTI competition). Specifically, the following mapping procedure has been adopted:

(i) *unlabeled* label was introduced;

(ii) all *moving-** labels were mapped to *** labels (e.g. *moving-car* was mapped to *car*);

(iii) *lane-marking* was mapped to *road*;

(iv) *other-structure* and *outlier* were mapped to *unlabeled*;

(v) *on-rails* and *bus* were mapped to *other-vehicle*;

2.2 Corrupted data

As presented in chapter 1, *LiDAR*s typically struggle a lot versus adverse weather conditions or external disturbances.
For this reason, in order to test the robustness of presented models, which are trained on classical SemanticKITTI (i.e. not with really challenging data), we decided to generate augmented data, directly starting from classical data. This data will be used for testing how models react to a - quite interesting - change in sensing conditions, both with and without a further fine-tuning on this augmented data.
Specifically, three *under out-of-distribution* (*OoD*) scenarios (first two related to adverse weather conditions, the third one related to external disturbances) have been considered: (1) Fog; (2) Wet ground; (3) Motion blur.
In particular, we generated this augmented data, for each under *OoD* scenario, from sequences 03, 04 and 06 for fine-tuning, and from sequence 08 for test (in this way, we can make a quite fair comparison of model behavior in each case).

3 Models

3.1 SqueezeSegV3

Before the introduction of *SqueezeSegV3*, the most common method for large-scale point cloud segmentation was to project a 3D point cloud to get a 2D LiDAR image and use convolutions to

process it.
The main problem is that the feature distribution of LiDAR images changes drastically at different image locations, so using standard convolutions to process LiDAR images is problematic, because convolution filters pick up local features that are only active in specific regions in the image.
As a result, the capacity of the network is under-utilized and the segmentation performance decreases.
To solve this problem, *SAC* (*Spatially-Adaptive Convolution*) has been introduced. The basic idea is to adopt different filters for different locations of the input image, and this is perfect for LiDAR images, where the feature distribution across the image are no longer identical.
Moreover, *SAC* can be computed efficiently since it can be implemented as a series of element-wise multiplications and standard convolution.
The general architecture of *SqueezeSegV3* with *SAC* is shown in figure 1.

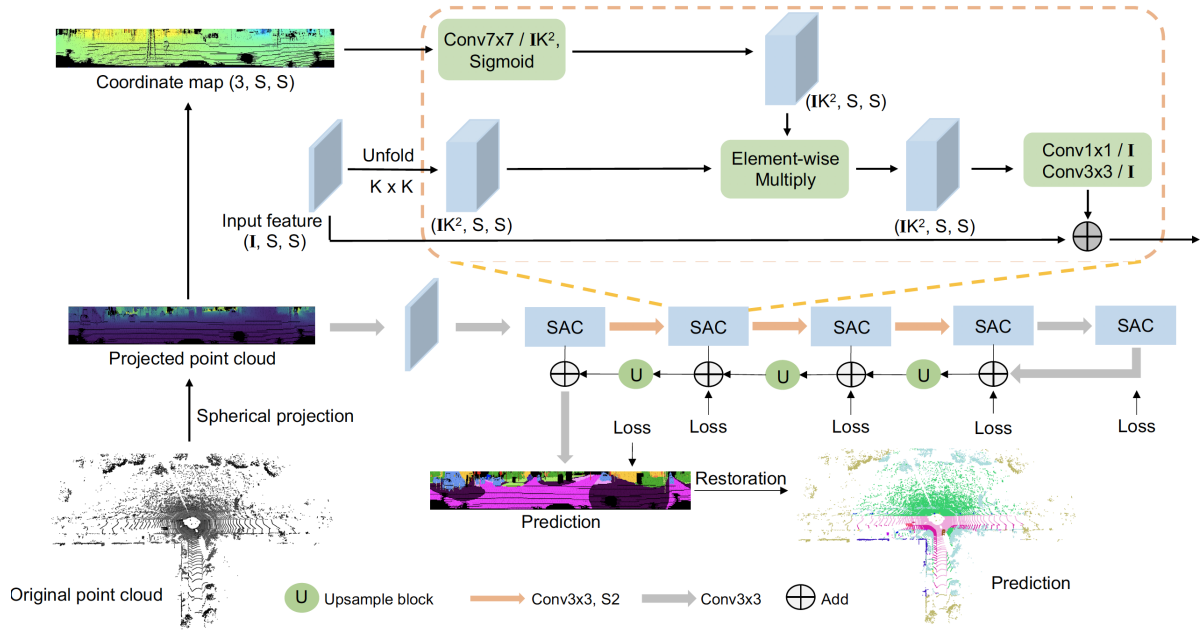

Figure 1: General architecture of SqueezeSegV3

As for hyperparameters, loss function is the classical *Cross Entropy* loss. Learning rate has been set to 0.005 for training and 0.0003 for fine tuning. The chosen optimizer is *SGD*, with a 0.9 momentum and a 3.0^{-4} weight decay.

3.2 SPVCNN

Given the limited hardware resources, most 3D perception models are not able to recognize small instances (e.g., pedestrians, cyclists) very well due to the low-resolution voxelization and aggressive downsampling.
For example, *PVCNN*s (build upon *Point Voxel Convolution*) is composed by a *point-based branch*, which transforms each point individually, and a *voxel-based branch*, base on large voxel grids, which convolves over the voxelized input from the point-based branch, in order to improve the locality. The problem is that, given a large outdoor scene, each voxel grid will correspond to a fairly large area. In this case, the small instances (e.g., pedestrians) will only occupy a few voxel grids (i.e.

few points), so the network can hardly learn any useful information from the voxel-based branch, leading to a relatively low performance.
Another example is volumetric *Sparse Convolution*, which basically skips the non-activated regions to significantly reduce the memory consumption. In this case, the problem is that the network cannot be very deep due to the limited computation resource. As a result, the network has to downsample very aggressively in order to achieve a sufficiently large receptive field, which is very lossy.
The introduction of *SPVC* (*Sparse Point-Voxel Convolution*) aims to overcome these limitations. As shown in figure 2, the idea is to have two branches:

- *Point-Based Branch*, which always keeps the high-resolution representation, capturing fine details in large scenes;
- *Sparse Voxel-Based Branch*, which applies *Sparse Convolution* to model across different receptive field size.

allowing them to communicate without high cost through *sparse voxelization* and *sparse devoxelization* (i.e. *point_to_voxel* and *voxel_to_point* functions).

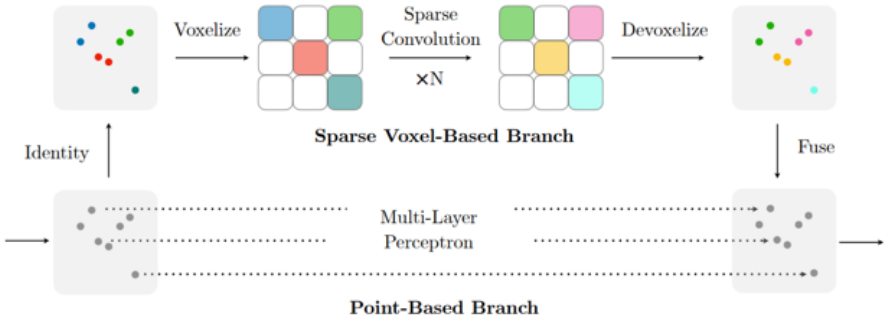

Figure 2: Sparse Point-Voxel Convolution

Specifically, it is possible to see the architecture of chosen *SPVCNN* in figure 3.
The choice was to train SPVCNN from scratch, for a total of 25 epochs (it turned out that epoch 22 is the best one), with first epoch as warm-up using sequences 00-10 (except 08) of SemanticKITTI, in order to have a fair comparison point of *SPVCNN* against *SqueezeSegV3*.
As for hyperparameters, all hidden sizes are shown in figure 3. Loss function is a combination of *Cross Entropy* and *Lovasz* losses. Learning rate has been set to 0.024 for training and 0.007 for fine tuning, using *Cosine Annealing Warm Restarts* as learning rate scheduler. The chosen optimizer is *SGD*, with a 0.9 momentum and a 3.0^{-6} weight decay.

4 Results

All models were trained in part on a Nvidia A100 80gb, and in the other part on a Nvidia P100 12gb, for a total of several training/tuning/testing days.
In table 1 below, we can see testing results (*mean accuracy* and *mean IoU*) for both *SqueezeSegV3* and *SPVCNN* models (fine-tuned and not), evaluated over *classical* and *corrupted* (fog, wet ground ans motion blur) data, while in table 2, same results are available but for each single classes.

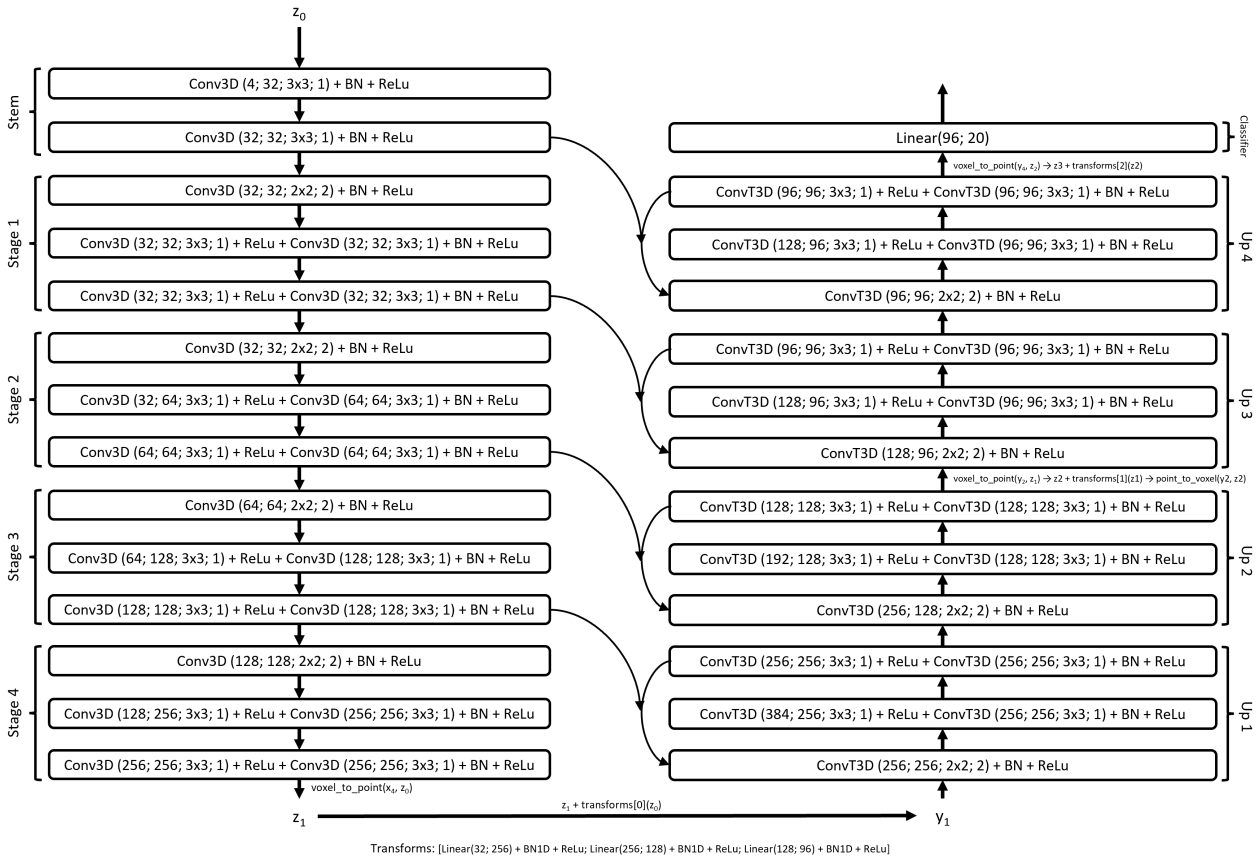

Figure 3: Chosen SPVCNN's architecture

Model	**Data**	**mIoU**	**Accuracy**
SqueezeSegV3	Classical	0.527	**0.891**
SqueezeSegV3	Fog	0.444	0.825
SqueezeSegV3	Motion Blur	0.370	0.789
SqueezeSegV3	Wet Ground	0.496	0.868
SqueezeSegV3 Fine-tuned	Fog	**0.486**	**0.877**
SqueezeSegV3 Fine-tuned	Motion Blur	0.468	0.867
SqueezeSegV3 Fine-tuned	Wet Ground	0.502	**0.87**
SPVCNN	Classical	**0.604**	0.875
SPVCNN	Fog	-	-
SPVCNN	Motion Blur	0.552	0.836
SPVCNN	Wet Ground	0.575	0.865
SPVCNN Fine-tuned	Fog	-	-
SPVCNN Fine-tuned	Motion Blur	**0.630**	**0.875**
SPVCNN Fine-tuned	Wet Ground	**0.627**	0.863

Table 1: Testing results (mean accuracy and mean IoU) for both SqueezeSegV3 and SPVCNN models (fine-tuned and not), evaluated over classical and corrupted (fog, wet ground ans motion blur) data.

Model	Data	Car	Bicycle	Motorcycle	Truck	Other-veichle	Person	Bicyclist	Motorcyclist	Road	Parking	Sidewalk	Other-ground	Building	Fence	Vegetation	Trunk	Terrain	Pole	Traffic-sign
SqueezeSegV3	Classical	0.861	0.309	0.478	0.507	0.424	0.520	0.521	0.000	0.945	0.473	0.816	0.003	0.802	0.472	0.825	0.524	0.720	0.423	0.381
SqueezeSegV3	Fog	0.853	0.245	0.417	0.489	0.325	0.421	0.428	0.000	0.815	0.231	0.662	0.005	0.754	0.279	0.760	0.440	0.609	0.384	0.319
SqueezeSegV3	Motion Blur	0.797	0.217	0.244	0.120	0.201	0.249	0.318	0.000	0.756	0.205	0.544	0.002	0.709	0.308	0.747	0.407	0.545	0.332	0.337
SqueezeSegV3	Wet Ground	0.870	0.296	0.424	0.476	0.389	0.484	0.500	0.000	0.843	0.328	0.679	0.005	0.798	0.455	0.827	0.523	0.718	0.425	0.388
SqueezeSegV3 Fine-tuned	Fog	0.851	0.241	0.453	0.395	0.324	0.471	0.505	0.000	0.935	0.423	0.797	0.005	0.788	0.442	0.806	0.509	0.692	0.327	0.270
SqueezeSegV3 Fine-tuned	Motion Blur	0.838	0.237	0.422	0.402	0.223	0.414	0.478	0.000	0.920	0.391	0.782	0.003	0.786	0.424	0.793	0.472	0.695	0.321	0.28
SqueezeSegV3 Fine-tuned	Wet Ground	0.865	0.283	0.478	0.389	0.330	0.491	0.563	0.000	0.903	0.436	0.738	0.003	0.794	0.460	0.819	0.520	0.724	0.390	0.355
SPVCNN	Classical	0.921	0.462	0.655	0.704	0.481	0.614	0.758	0.003	0.849	0.295	0.743	0.001	0.907	0.649	0.890	0.678	0.777	0.578	0.506
SPVCNN	Fog	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SPVCNN	Motion Blur	0.898	0.461	0.645	0.477	0.405	0.509	0.765	0.010	0.770	0.204	0.606	0.000	0.854	0.603	0.868	0.662	0.683	0.560	0.497
SPVCNN	Wet Ground	0.928	0.499	0.657	0.458	0.476	0.594	0.757	0.001	0.701	0.222	0.620	0.001	0.904	0.654	0.892	0.675	0.792	0.580	0.506
SPVCNN Fine-tuned	Fog	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SPVCNN Fine-tuned	Motion Blur	0.949	0.477	0.673	0.849	0.481	0.714	0.878	0.000	0.909	0.366	0.765	0.010	0.897	0.636	0.864	0.697	0.693	0.611	0.504
SPVCNN Fine-tuned	Wet Ground	0.957	0.508	0.671	0.896	0.439	0.739	0.881	0.000	0.857	0.322	0.681	0.034	0.899	0.614	0.864	0.699	0.714	0.616	0.520

Table 2: Single-class testing mean IoU for both SqueezeSegV3 and SPVCNN models (fine-tuned and not), evaluated over classical and corrupted (fog, wet ground ans motion blur) data

Clearly, as expected, *SPVCNN* outperforms *SqueezeSegV3*, on both classical and corrupted data, in both fine-tuned and not versions.
As we can see, there is a performance drop when testing the not-fine-tuned model with corrupted data, but it is quite reasonable, since we do inference with a type of data quite different from training one. As expected, performance increase by fine-tuning the model, especially for *SPVCNN*, where the fine-tuned model outperform the baseline one, also considering corrupted vs classical data.