Point Cloud Object Detection

(Project Report)

Foundations of Machine Learning (CS 725)
Computer Science and Engineering

By

Santhosh	Shivansh	Helen	Sreekant	Siva	Nadesh
23d0369	22m2120	22m2115	23M0794	23M0747	21q050003

Guided By
Prof. Sunita Sarawagi

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
October 31, 2023

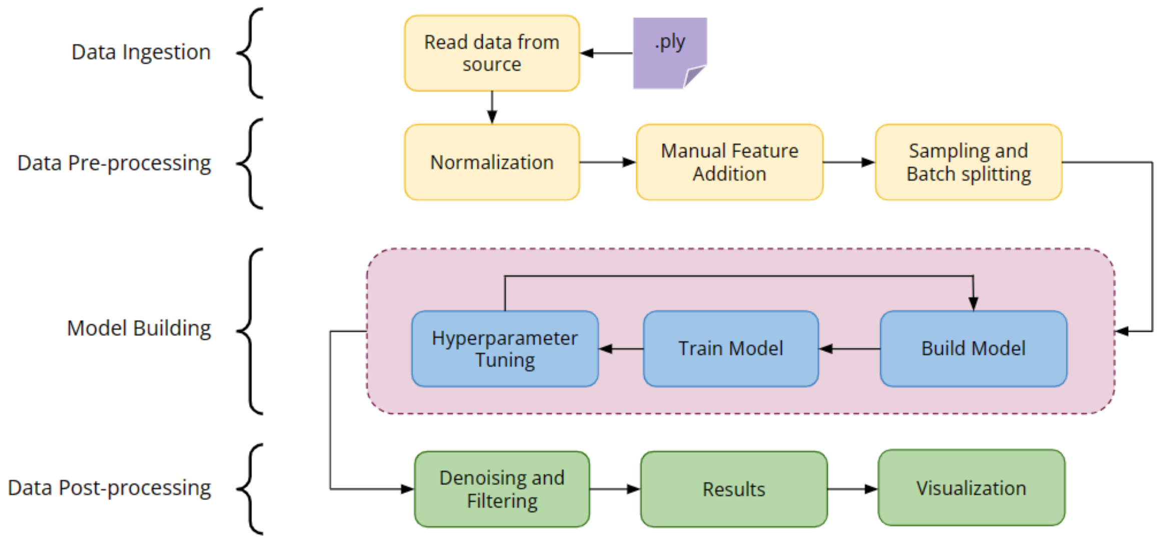
Table of Contents

1. Introduction

World Health Organization has estimated an average of 1.3 million road deaths every year. Road marking extraction is emerging as an important remote sensing application to meet United Nations goal to reduce road injuries by half by 2030. One of the ways to reduce Highway injuries is to ensure Road marking, Lighting Poles and GuardRails are in place and in proper condition. The process of analysis of these highway objects is a manual, tedious and time-consuming process. With advancements in data capture technologies, cost of 3D point cloud data capture have drastically reduced over the last decade. Today point cloud and photogrammetry data capture methods are widely used and processed to identify road assets and its conditions to monitor and maintain road safety.

Machine learning algorithms for point cloud have evolved since 2017. PointNet was the first algorithm to use 3D based convolution in 2019. Since then, different methods of 3D based neural network algorithms have evolved. Different methods of convolution for 3D point cloud include voxelization, graph network methods and point cloud transformers each with its own advantages and disadvantages.

In this project, we would look at exploring three different neural network models (Convolution based, Graph Based and Transformer based) for classification of Fence, Road Marking and Poles that are described in Section 2. We would compare the performance of this model using Triplet Loss and see if the model performance is better (given limited labeled datasets). Section 3 has a brief description of the dataset we are using and in Section 4, we describe the proposed solution. Section 5 has the list of activities done till now and roadmap for the remainder of the semester. Finally, Section 6 has preliminary results.



2. Related Work

In this section we are presenting a brief overview of the three deep learning neural network models PointNet, DGCNN and SPT we are using in our project.

2.1. PointNet

The PointNet architecture is designed to directly consume point cloud data, which are sets of points in 3D space. It can perform various 3D recognition tasks, such as object classification, part segmentation, and scene semantic parsing. The architecture has the following properties:

- It is simple and efficient, using only fully connected layers and max pooling to process point sets.
- It is invariant to the order of input points, using a symmetric function to aggregate information from all points.
- It is robust to input perturbation and corruption, using spatial and feature transformations to align the data.
- It can learn both global and local features of point sets, using a combination of global shape signature and per-point features.

The PointNet architecture consists of two networks: a classification network and a segmentation network. The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification score for m classes. The segmentation network is an extension to the classification net. Each of the n inputs needs to assign one of the m segmentation classes. Because segmentation relies on local and global features, the points in the 64-dimensional space are concatenated with the global feature space, resulting in possible feature space of $n * \mathbb{R}^{108}$. The PointNet architecture has these key modules: the max-pooling layer, a local and global combination structure, and two joint alignment networks that align both local and global networks

2.2. Dynamic Graph CNN (DGCNN)

Dynamic Graph Convolutional Neural Network, a deep learning architecture that reads graphs directly and learns a classification function. The main building block of DGCNN, which operates on graphs dynamically computed in each layer of the network1. It generates edge features that describe the relationships between a point and its neighbors, and aggregates them using a symmetric function2. EdgeConv has several appealing properties, such as permutation invariance, partial translation invariance, and non-local diffusion of information throughout the point cloud3. DGCNN is used for point-cloud-related high-level tasks including category classification, semantic segmentation, and part segmentation. It achieves state-of-the-art performance on several benchmark datasets.

2.3. SuperPoint Transformer (SPT)

The superpoint transformer architecture used in this paper is a novel method for semantic segmentation of large point clouds1. It has the following main components:

- Hierarchical superpoint partition: It segments the input point cloud into a multi-scale structure of geometrically homogeneous superpoints, which reduces the input size and adapts to the local complexity of the data2.
- Superpoint transformer: It uses a U-Net-like network that operates on the superpoints at different scales, using self-attention to capture the relationships between superpoints within and across levels3. It also uses handcrafted features and adjacency encoding to characterize the superpoints and their interactions.
- Hierarchical supervision and augmentation: It leverages the hierarchical partition structure to supervise the model with both label frequency and distribution, and to augment the data by randomly dropping superpoints at different levels.

2.4. Triplet Loss

Triplet loss is a way to teach machine-learning algorithms to recognize not only the similarity, but also differences between items. The objective of triplet loss is to reduce the loss between similar items and increase the gap between different items. The mathematical depiction is shown below:

$$\sum_{i}^{N}\left[\left|\left|f(x_i^a) - f(x_i^p)\right|\right|_2^2 - \left|\left|f(x_i^a) - f(x_i^n)\right|\right|_2^2 + \propto\right]$$

Where

- f(x) accepts an input of x and generates a 128-dimensional vector w
- I represents the i'th input
- The subscript a denotes an anchor image, p is a positive image, and n is a negative image refers to the bias

The goal is to minimize the above equation by minimizing the first term and maximizing the second term, and bias acts as a threshold.

3. Datasets

A Point in space is generally characterized by three parameters, the X, Y, and Z coordinates. A large collection of these points is termed Point Cloud. These are particularly useful for object classification and detection in 3D, which is then utilized for autonomous driving systems. This data is usually captured using 3D scanners or LiDAR technology (Light Detection And Ranging). In practice, each data point consists of various parameters as well apart from coordinates.

- XYZ Coordinates
- RGB color values
- Intensity
- GPS time
- Scan angle details

There are multiple widely-used point cloud datasets available. Some are them are:

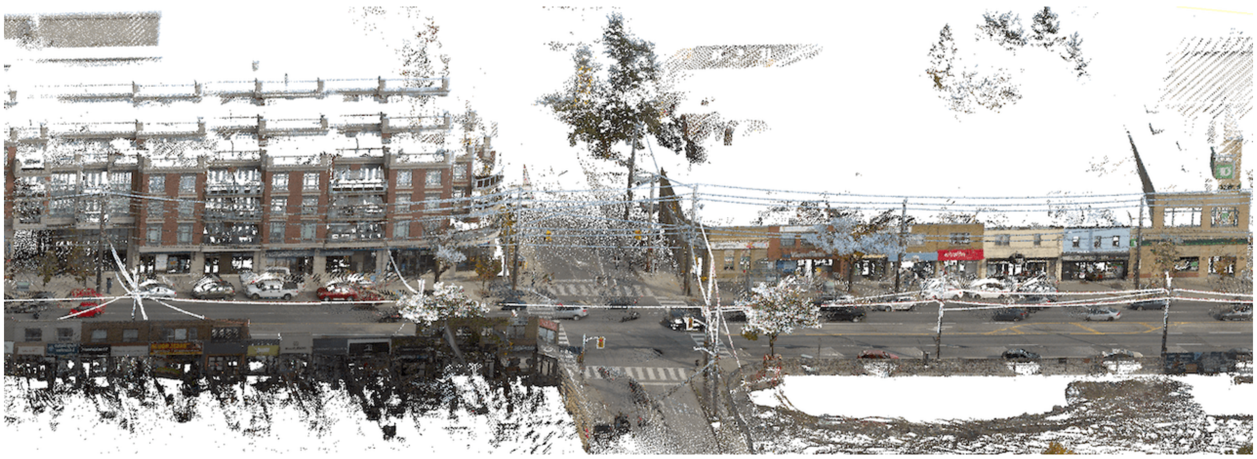
- Semantic KITTI - http://www.semantic-kitti.org/dataset.html
- Paris-Lille-3D - https://npm3d.fr/paris-lille-3d
- Toronto-3D - https://github.com/WeikaiTan/Toronto-3D

Here is a rough number of classes in each dataset related to each broad category.

	Semantic KITTI	**Paris-Lille-3D**	**Toronto-3D**
Road/Ground	4	3	1
Buildings/Structures	2	2	1
Vehicles	5	15	1
Vegetation	2	3	1
People/Bicyclists	3	1	0
Road/Traffic Elements	5	9	3
Furniture/Objects	1	15	1

SemanticKITTI extends KITTI Vision by adding semantic annotations to existing dataset sequences, providing numerous scans for a full 360-degree view with LiDAR. Labeled at 10 Hz, it uses temporal data for semantic scene understanding, featuring classes for moving and stationary traffic participants (e.g., cars, trucks, pedestrians).

Paris-Lille-3D is generated using a Mobile Laser System (MLS) in Paris and Lille, France. It's hand-labeled with 50 diverse classes, used for developing automated point cloud segmentation and classification algorithms.



4 Classes:

1. Fence
2. Pole
3. Road Marking
4. Unknown

L001
L002
L003
L004

Overall : **78.3 million points**
Train dataset : Test dataset = 4:1
9 attributes:

1. XYZ (Pointnet SPT, and DGCNN)
2. RGB (SPT)
3. Intensity (SPT)
4. GPS time
5. Scan angle rank

Toronto-3D is a large-scale urban outdoor point cloud dataset acquired in Toronto,Canada. It covers approximately 1km of road and consists of about 78.3 million points. This dataset includes 9 attributes in which we are mainly focusing on the Fence, Road marking, Pole attributes. In this project, we will be focusing on this dataset.

4. Solution Implementation

Given the limited availability of labeled data for 3D point cloud datasets, we used encoder based transformers for point cloud segmentation. In order to improve the accuracy for small objects like Fences, Electric Poles, we implemented Siamese Triplet Loss to increase the accuracy of the same.

The detailed implementation changes can be found at
https://github.com/shivansh1010/FML-project/tree/main

Below section details each of the changes done to existing algorithms.

4.1 Dataset Processing

Given the large volume of point cloud, we had to undertake preprocessing algorithms to improve the training and inference time. We performed the following activities on the Toronto 3D dataset.

1. X, Y, Z offset normalization to avoid jumps and also to prevent location specific learning of data positions.
2. Normalized RGB and intensity in the range for 0-1 instead of 0-255 to enable real number computation of weights.
3. Pre computed KNN based clustering to identify voxel and reduce the number of points. This is very similar to Embeddings where we take a one-hot encoder and compress the information into smaller embedding dimensions.
4. Mini batch getItem implementation to randomize the point selections for each run..

4.2 PointNet and DGCNN

Existing PointNet and DGCNN were predominantly implemented for object identification (not point classification). As our focus was segmentation of individual points in 3D for the entire scene, we have to modify both PointNet and DGCNN implementation to support segmentation. To achieve scene level classification, we had to increase the number of layers using Covn2D. As the number of parameters increased, we introduced a dropout layer just before the feed forward layer.
Due to focus on object identification, Toronto 3D dataset was not integrated with existing implementations of DGCNN and PointNet. We used pytorch DataModule to enable Toronto 3D.

The existing Pointnet model is trained for object classification on ModelNet40 dataset. The test accuracy and train accuracy of the training test is 0.831 and 0.842. The test accuracy and train accuracy of the validation test is 0.008 and 0.025. We implemented a pointnet model on the Toronto 3D dataset. Toronto 3D dataset can’t be used directly for training the Pointnet model. So, we integrated Toronto 3D into the model by changing the format. In the Toronto 3D dataset, each point needs to be classified. So, we reimplemented the pointnet model for semantic segmentation. Semantic segmentation recognizes a collection of pixels into distinct categories such as traffic signs, poles, pavements etc.. Series of MLP are done on input features and output scores of each pixel is obtained.

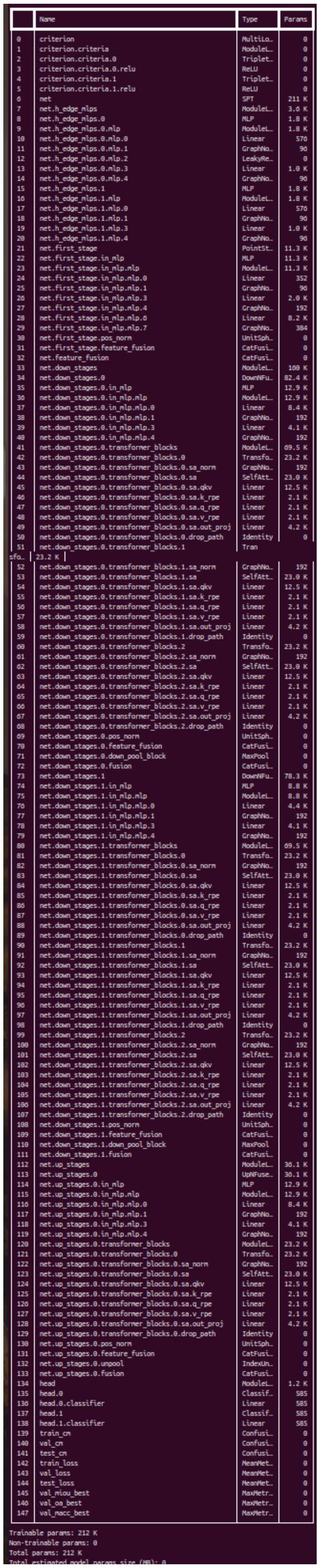
The existing DGCNN model is trained for object classification on ModelNet40 dataset. The test accuracy and train accuracy of the training test is 0.998 and 0.888. The test accuracy and train accuracy of the validation test is 0.008 and 0.025. We implemented the DGCNN model on the Toronto 3D dataset. The Toronto 3D dataset can't be used directly for training the DGCNN model. So, we integrated Toronto 3D into the model by changing the format. In the Toronto 3D dataset, each point needs to be classified. So, we reimplemented the DGCNN model for semantic segmentation.

Semantic segmentation recognizes a collection of pixels into distinct categories such as traffic signs, poles, pavements etc.. Series of MLP are done on input features and output scores of each pixel is obtained.

4.3 Superpoint Transformers

Due to the large size of point cloud datasets, various algorithms have been proposed for data reduction. Voxel based data reduction is one of the popular techniques used in Point Cloud. Super Point enhances Voxel implementation without reducing the information. Also Super Point approach supports manual feature additions on top of deep learning.

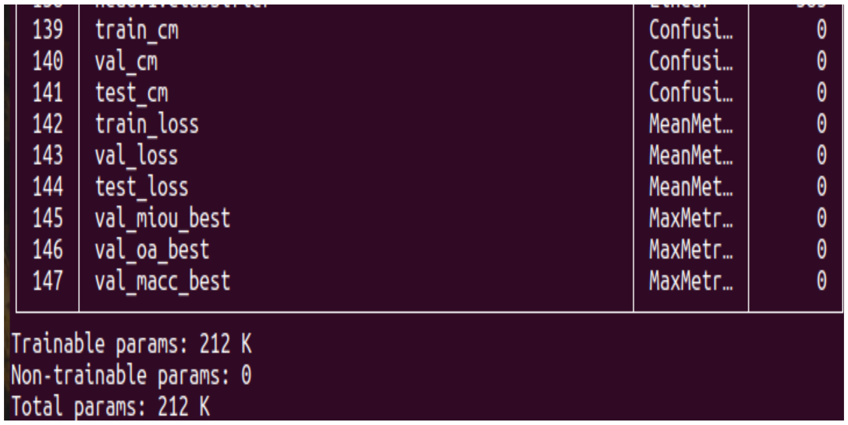
With increased performance using Transformer, we looked at existing Transformer implementations for Point Cloud. Super Point Transformer is an open source implementation that uses Encoder Transformer for point classification and uses Cross Entropy loss. Given our focus on improving classification accuracy for small objects, we implemented Triplet Losses. The existing SuperPoint Transformer is using a cross entropy loss function. The test accuracy and train accuracy for the training test is 87.778% and 76.976% . The validation test accuracy is 77.136%. We used Siamese Networks to get better classification. We use two identical networks with the same parameters and weights to compute the similarity and dissimilarity between inputs. Since the Siamese a neural network uses pairwise learning, we cannot use cross entropy loss. SNNs involve pairwise learning, we cannot use cross entropy loss, hence have used triplet loss function. Included intensity of data feature for SPT



```
     Name                                                    Type        Params

0    criterion                                               MultiLo…    0
1    criterion.criteria                                      ModuleL…    0
2    criterion.criteria.0                                    Triplet…    0
3    criterion.criteria.0.relu                               ReLU        0
4    criterion.criteria.1                                    Triplet…    0
5    criterion.criteria.1.relu                               ReLU        0
6    net                                                     SPT         211 K
7    net.h_edge_mlps                                         ModuleL…    3.6 K
8    net.h_edge_mlps.0                                       MLP         1.8 K
9    net.h_edge_mlps.0.mlp                                   ModuleL…    1.8 K
10   net.h_edge_mlps.0.mlp.0                                 Linear      576
11   net.h_edge_mlps.0.mlp.1                                 GraphNo…    96
12   net.h_edge_mlps.0.mlp.2                                 LeakyRe…    0
13   net.h_edge_mlps.0.mlp.3                                 Linear      1.0 K
14   net.h_edge_mlps.0.mlp.4                                 GraphNo…    96
15   net.h_edge_mlps.1                                       MLP         1.8 K
16   net.h_edge_mlps.1.mlp                                   ModuleL…    1.8 K
17   net.h_edge_mlps.1.mlp.0                                 Linear      576
18   net.h_edge_mlps.1.mlp.1                                 GraphNo…    96
19   net.h_edge_mlps.1.mlp.3                                 Linear      1.0 K
20   net.h_edge_mlps.1.mlp.4                                 GraphNo…    96
21   net.first_stage                                         PointSt…    11.3 K
22   net.first_stage.in_mlp                                  MLP         11.3 K
23   net.first_stage.in_mlp.mlp                              ModuleL…    11.3 K
24   net.first_stage.in_mlp.mlp.0                            Linear      352
25   net.first_stage.in_mlp.mlp.1                            GraphNo…    96
26   net.first_stage.in_mlp.mlp.3                            Linear      2.0 K
27   net.first_stage.in_mlp.mlp.4                            GraphNo…    192
28   net.first_stage.in_mlp.mlp.6                            Linear      8.2 K
29   net.first_stage.in_mlp.mlp.7                            GraphNo…    384
30   net.first_stage.pos_norm                                UnitSph…    0
31   net.first_stage.feature_fusion                          CatFusi…    0
32   net.feature_fusion                                      CatFusi…    0
33   net.down_stages                                         ModuleL…    160 K
34   net.down_stages.0                                       DownNFu…    82.4 K
35   net.down_stages.0.in_mlp                                MLP         12.9 K
36   net.down_stages.0.in_mlp.mlp                            ModuleL…    12.9 K
37   net.down_stages.0.in_mlp.mlp.0                          Linear      8.4 K
38   net.down_stages.0.in_mlp.mlp.1                          GraphNo…    192
39   net.down_stages.0.in_mlp.mlp.3                          Linear      4.1 K
40   net.down_stages.0.in_mlp.mlp.4                          GraphNo…    192
41   net.down_stages.0.transformer_blocks                    ModuleL…    69.5 K
42   net.down_stages.0.transformer_blocks.0                  Transfo…    23.2 K
43   net.down_stages.0.transformer_blocks.0.sa_norm          GraphNo…    192
44   net.down_stages.0.transformer_blocks.0.sa               SelfAtt…    23.0 K
45   net.down_stages.0.transformer_blocks.0.sa.qkv           Linear      12.5 K
46   net.down_stages.0.transformer_blocks.0.sa.k_rpe         Linear      2.1 K
47   net.down_stages.0.transformer_blocks.0.sa.q_rpe         Linear      2.1 K
48   net.down_stages.0.transformer_blocks.0.sa.v_rpe         Linear      2.1 K
49   net.down_stages.0.transformer_blocks.0.sa.out_proj      Linear      4.2 K
50   net.down_stages.0.transformer_blocks.0.drop_path        Identity    0
51   net.down_stages.0.transformer_blocks.1                  Tran
sfo… | 23.2 K |
52   net.down_stages.0.transformer_blocks.1.sa_norm          GraphNo…    192
53   net.down_stages.0.transformer_blocks.1.sa               SelfAtt…    23.0 K
54   net.down_stages.0.transformer_blocks.1.sa.qkv           Linear      12.5 K
55   net.down_stages.0.transformer_blocks.1.sa.k_rpe         Linear      2.1 K
56   net.down_stages.0.transformer_blocks.1.sa.q_rpe         Linear      2.1 K
57   net.down_stages.0.transformer_blocks.1.sa.v_rpe         Linear      2.1 K
58   net.down_stages.0.transformer_blocks.1.sa.out_proj      Linear      4.2 K
59   net.down_stages.0.transformer_blocks.1.drop_path        Identity    0
60   net.down_stages.0.transformer_blocks.2                  Transfo…    23.2 K
61   net.down_stages.0.transformer_blocks.2.sa_norm          GraphNo…    192
62   net.down_stages.0.transformer_blocks.2.sa               SelfAtt…    23.0 K
63   net.down_stages.0.transformer_blocks.2.sa.qkv           Linear      12.5 K
64   net.down_stages.0.transformer_blocks.2.sa.k_rpe         Linear      2.1 K
65   net.down_stages.0.transformer_blocks.2.sa.q_rpe         Linear      2.1 K
66   net.down_stages.0.transformer_blocks.2.sa.v_rpe         Linear      2.1 K
67   net.down_stages.0.transformer_blocks.2.sa.out_proj      Linear      4.2 K
68   net.down_stages.0.transformer_blocks.2.drop_path        Identity    0
69   net.down_stages.0.pos_norm                              UnitSph…    0
70   net.down_stages.0.feature_fusion                        CatFusi…    0
71   net.down_stages.0.down_pool_block                       MaxPool     0
72   net.down_stages.0.fusion                                CatFusi…    0
73   net.down_stages.1                                       DownNFu…    78.3 K
74   net.down_stages.1.in_mlp                                MLP         8.8 K
75   net.down_stages.1.in_mlp.mlp                            ModuleL…    8.8 K
76   net.down_stages.1.in_mlp.mlp.0                          Linear      4.4 K
77   net.down_stages.1.in_mlp.mlp.1                          GraphNo…    192
78   net.down_stages.1.in_mlp.mlp.3                          Linear      4.1 K
79   net.down_stages.1.in_mlp.mlp.4                          GraphNo…    192
80   net.down_stages.1.transformer_blocks                    ModuleL…    69.5 K
81   net.down_stages.1.transformer_blocks.0                  Transfo…    23.2 K
82   net.down_stages.1.transformer_blocks.0.sa_norm          GraphNo…    192
83   net.down_stages.1.transformer_blocks.0.sa               SelfAtt…    23.0 K
84   net.down_stages.1.transformer_blocks.0.sa.qkv           Linear      12.5 K
85   net.down_stages.1.transformer_blocks.0.sa.k_rpe         Linear      2.1 K
86   net.down_stages.1.transformer_blocks.0.sa.q_rpe         Linear      2.1 K
87   net.down_stages.1.transformer_blocks.0.sa.v_rpe         Linear      2.1 K
88   net.down_stages.1.transformer_blocks.0.sa.out_proj      Linear      4.2 K
89   net.down_stages.1.transformer_blocks.0.drop_path        Identity    0
90   net.down_stages.1.transformer_blocks.1                  Transfo…    23.2 K
91   net.down_stages.1.transformer_blocks.1.sa_norm          GraphNo…    192
92   net.down_stages.1.transformer_blocks.1.sa               SelfAtt…    23.0 K
93   net.down_stages.1.transformer_blocks.1.sa.qkv           Linear      12.5 K
94   net.down_stages.1.transformer_blocks.1.sa.k_rpe         Linear      2.1 K
95   net.down_stages.1.transformer_blocks.1.sa.q_rpe         Linear      2.1 K
96   net.down_stages.1.transformer_blocks.1.sa.v_rpe         Linear      2.1 K
97   net.down_stages.1.transformer_blocks.1.sa.out_proj      Linear      4.2 K
98   net.down_stages.1.transformer_blocks.1.drop_path        Identity    0
99   net.down_stages.1.transformer_blocks.2                  Transfo…    23.2 K
100  net.down_stages.1.transformer_blocks.2.sa_norm          GraphNo…    192
101  net.down_stages.1.transformer_blocks.2.sa               SelfAtt…    23.0 K
102  net.down_stages.1.transformer_blocks.2.sa.qkv           Linear      12.5 K
103  net.down_stages.1.transformer_blocks.2.sa.k_rpe         Linear      2.1 K
104  net.down_stages.1.transformer_blocks.2.sa.q_rpe         Linear      2.1 K
105  net.down_stages.1.transformer_blocks.2.sa.v_rpe         Linear      2.1 K
106  net.down_stages.1.transformer_blocks.2.sa.out_proj      Linear      4.2 K
107  net.down_stages.1.transformer_blocks.2.drop_path        Identity    0
108  net.down_stages.1.pos_norm                              UnitSph…    0
109  net.down_stages.1.feature_fusion                        CatFusi…    0
110  net.down_stages.1.down_pool_block                       MaxPool     0
111  net.down_stages.1.fusion                                CatFusi…    0
112  net.up_stages                                           ModuleL…    36.1 K
113  net.up_stages.0                                         UpNFuse…    36.1 K
114  net.up_stages.0.in_mlp                                  MLP         12.9 K
115  net.up_stages.0.in_mlp.mlp                              ModuleL…    12.9 K
116  net.up_stages.0.in_mlp.mlp.0                            Linear      8.4 K
117  net.up_stages.0.in_mlp.mlp.1                            GraphNo…    192
118  net.up_stages.0.in_mlp.mlp.3                            Linear      4.1 K
119  net.up_stages.0.in_mlp.mlp.4                            GraphNo…    192
120  net.up_stages.0.transformer_blocks                      ModuleL…    23.2 K
121  net.up_stages.0.transformer_blocks.0                    Transfo…    23.2 K
122  net.up_stages.0.transformer_blocks.0.sa_norm            GraphNo…    192
123  net.up_stages.0.transformer_blocks.0.sa                 SelfAtt…    23.0 K
124  net.up_stages.0.transformer_blocks.0.sa.qkv             Linear      12.5 K
125  net.up_stages.0.transformer_blocks.0.sa.k_rpe           Linear      2.1 K
126  net.up_stages.0.transformer_blocks.0.sa.q_rpe           Linear      2.1 K
127  net.up_stages.0.transformer_blocks.0.sa.v_rpe           Linear      2.1 K
128  net.up_stages.0.transformer_blocks.0.sa.out_proj        Linear      4.2 K
129  net.up_stages.0.transformer_blocks.0.drop_path          Identity    0
130  net.up_stages.0.pos_norm                                UnitSph…    0
131  net.up_stages.0.feature_fusion                          CatFusi…    0
132  net.up_stages.0.unpool                                  IndexUn…    0
133  net.up_stages.0.fusion                                  CatFusi…    0
134  head                                                    ModuleL…    1.2 K
135  head.0                                                  Classif…    585
136  head.0.classifier                                       Linear      585
137  head.1                                                  Classif…    585
138  head.1.classifier                                       Linear      585
139  train_cm                                                Confusi…    0
140  val_cm                                                  Confusi…    0
141  test_cm                                                 Confusi…    0
142  train_loss                                              MeanMet…    0
143  val_loss                                                MeanMet…    0
144  test_loss                                               MeanMet…    0
145  val_miou_best                                           MaxMetr…    0
146  val_oa_best                                             MaxMetr…    0
147  val_macc_best                                           MaxMetr…    0

Trainable params: 212 K
Non-trainable params: 0
Total params: 212 K
Total estimated model params size (MB): 0
```



```
139  train_cm         Confusi…    0
140  val_cm           Confusi…    0
141  test_cm          Confusi…    0
142  train_loss       MeanMet…    0
143  val_loss         MeanMet…    0
144  test_loss        MeanMet…    0
145  val_miou_best    MaxMetr…    0
146  val_oa_best      MaxMetr…    0
147  val_macc_best    MaxMetr…    0

Trainable params: 212 K
Non-trainable params: 0
Total params: 212 K
```

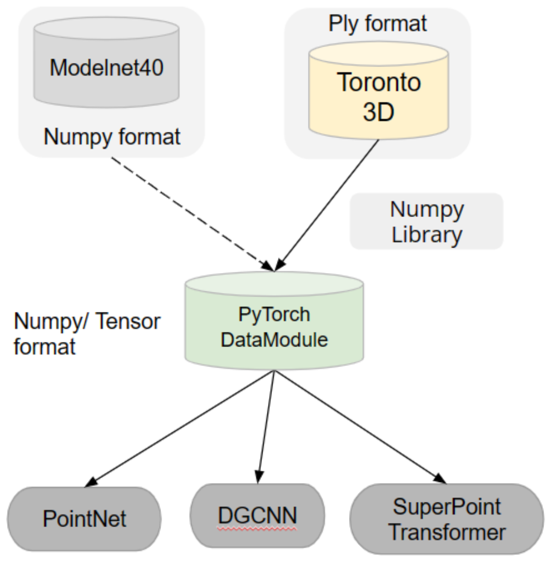
4.4 Triplet Loss Implementation

There were two types of Triple Losses supported in PyTorch. We used the online (soft) triplet loss as this did not require any manual specifications for hard negatives.

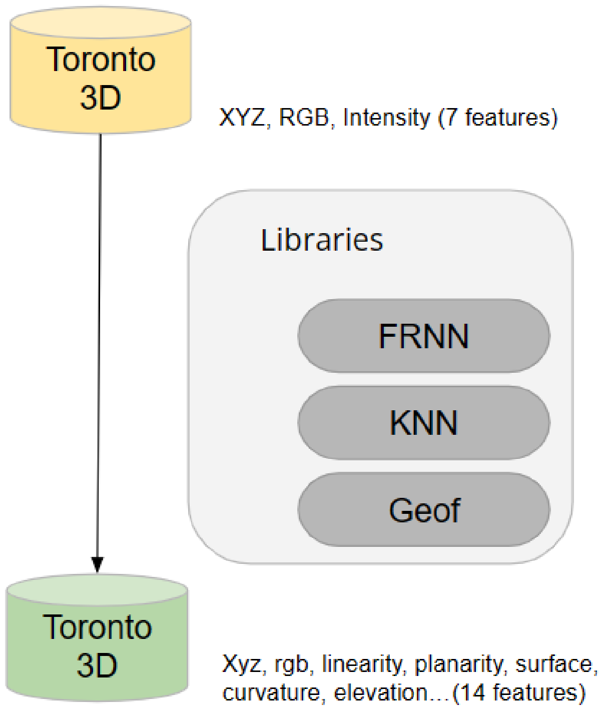
```
import torch.nn.functional as F
def batch all triplet loss(anchor, positive, negative, margin=0.2):
”””
Compute triplet loss using the batch all strategy.
”””
distance matrix = compute distance matrix(anchor, positive, negative)
loss = torch.max(torch.tensor(0.0), distance matrix[:, 0] - distance matrix[:, 1] +
margin)
loss += torch.max(torch.tensor(0.0), distance matrix[:, 0] - distance matrix[:, 2]
+ margin)
return torch.mean(loss)
```

The batch all triplet loss function computes the triplet loss using the batch-all strategy for a given set of anchor, positive, and negative samples and calculates the triplet loss based on the computed distance matrix, enforcing a margin of 0.2 between the positive and negative distances. The function returns the mean of the computed loss across all samples in the batch.

4.5 Dataset Ingestion



4.6 Dataset Preprocessing (manual features)


Toronto 3D
XYZ, RGB, Intensity (7 features)
Libraries
FRNN
KNN
Geof
Toronto 3D
Xyz, rgb, linearity, planarity, surface, curvature, elevation…(14 features)

4.7 Challenges Faced

1. Three of the four files in Toronto-3D files are greater than 1GB. Hence gives CUDA out-of-memory error (on a 16GB GPU machine). We have disabled use of GPU and running on CPU mode.
2. SuperPoint Transformers uses FRNN module which uses a very old version of CUDA library. Downgrading the CUDA library is a challenge as the server is a shared server. We are looking at CPU only FRNN library
3. Faced a lot of issues in early days related to library version mismatch. We explored various versions and now have finalized on a “conda” environment that is working.

5. Preliminary results

5.1 DGCN Results:

5.1.1 Training Results:

```
Train 249, loss: 1.286651, train acc: 0.998779, train avg acc: 0.997489
Test 249, loss: 1.442699, test acc: 0.918963, test avg acc: 0.888419
```

5.1.2 Validation Results:

```
Namespace(batch_size=32, dataset='modelnet40', dropout=0.3, emb_dims=1024, epochs=250, eval=True, exp_name='dgcnn_1024_eval', k=18, lr=0.01, model='dgcnn', model_path='checkpoints/dgcnn_1024/models/model.t7', momentum=0.1, no_cuda=False, num_points=1024, seed=1, test_batch_size=16, use_sgd=True)
Using GPU : 0 from 1 devices
```

Test :: test acc: 0.008104, test avg acc: 0.025000

5.2 PointNet results

5.2.1 Training Results

```
Train 249, loss: 1.582438, train acc: 0.890472, train avg acc: 0.831194
Test 249, loss: 1.527525, test acc: 0.894652, test avg acc: 0.842297
```

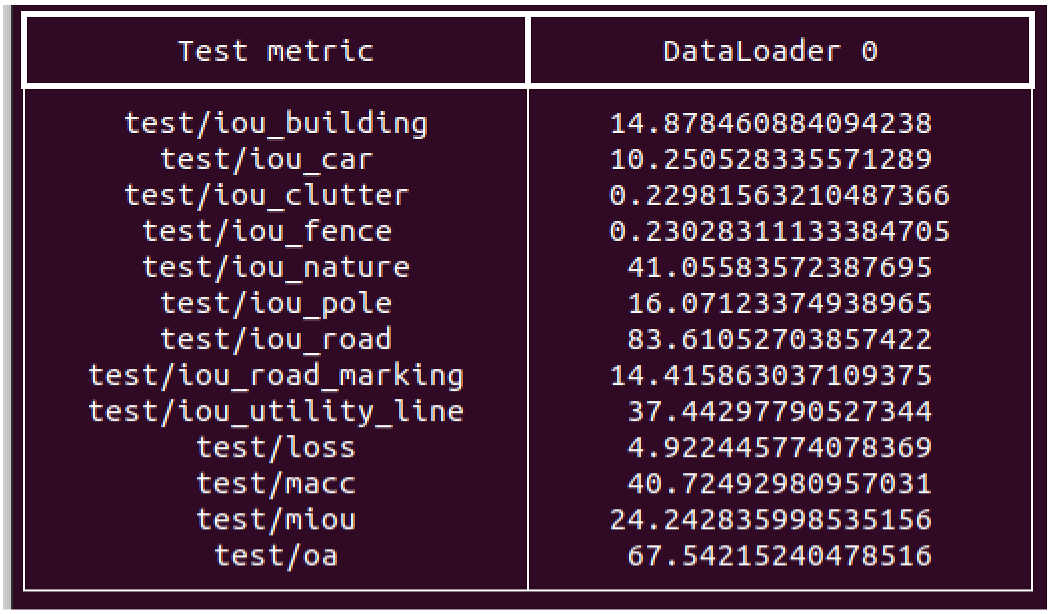
5.2.2 Validation Results

```
Namespace(batch_size=32, dataset='modelnet40', dropout=0.5, emb_dims=1024, epochs=250, eval=True, exp_name='pointnet_1024_eval', k=18, lr=0.001, model='pointnet', model_path='checkpoints/pointnet_1024/models/model.t7', momentum=0.9, no_cuda=False, num_points=1024, seed=1, test_batch_size=16, use_sgd=True)
Using GPU : 0 from 1 devices
```

Test :: test acc: 0.008104, test avg acc: 0.025000

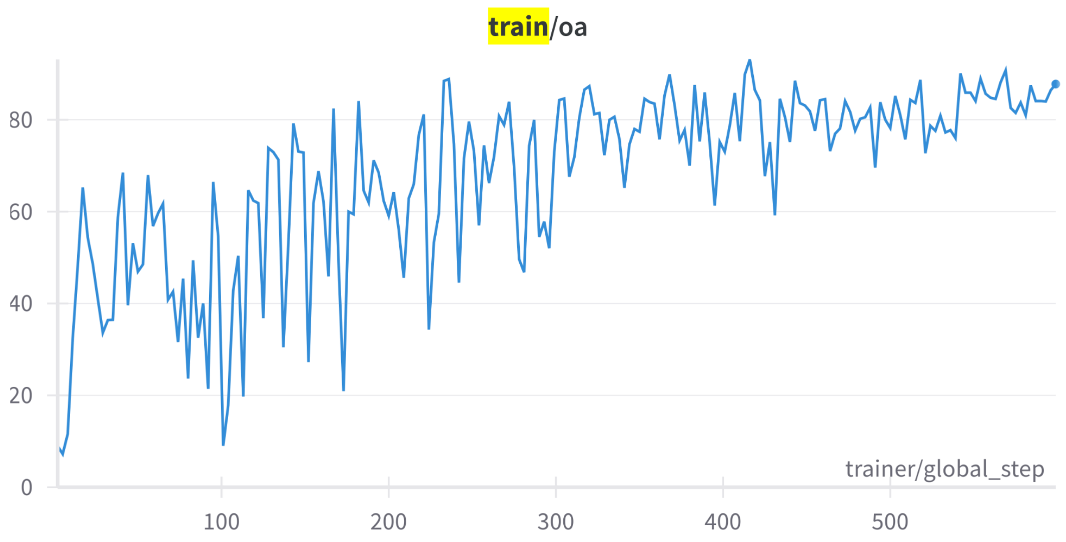
5.3 Super Point Transformer results

5.3.1 Training Results

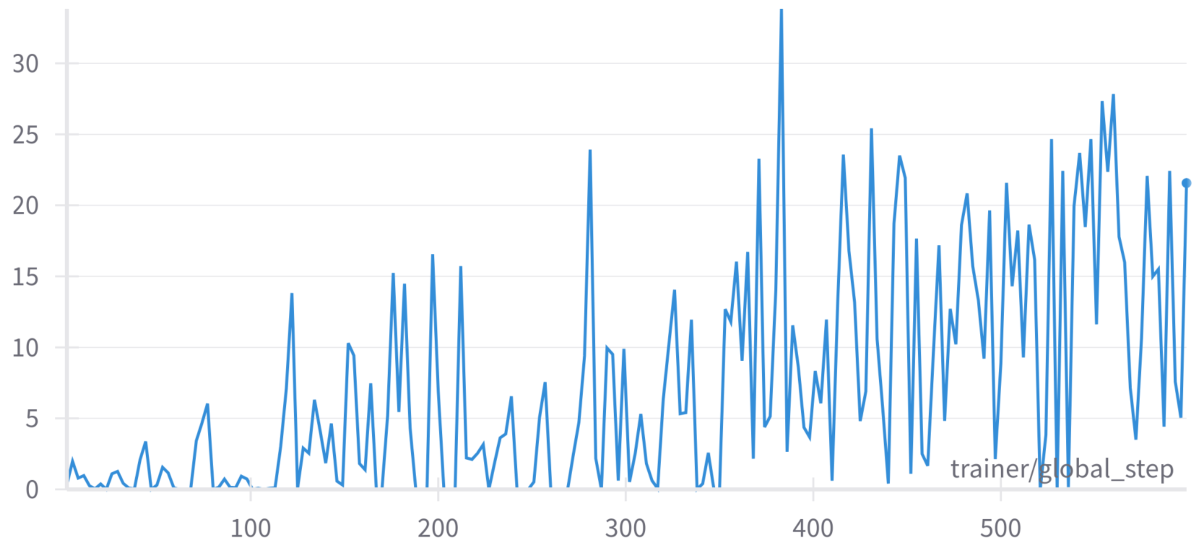


```
┃        Test metric        ┃       DataLoader 0        ┃
│     test/iou_building     │    14.878460884094238     │
│       test/iou_car        │    10.250528335571289     │
│     test/iou_clutter      │    0.22981563210487366    │
│      test/iou_fence       │    0.23028311133384705    │
│      test/iou_nature      │     41.05583572387695     │
│       test/iou_pole       │     16.07123374938965     │
│       test/iou_road       │     83.61052703857422     │
│   test/iou_road_marking   │    14.415863037109375     │
│   test/iou_utility_line   │     37.44297790527344     │
│         test/loss         │     4.922445774078369     │
│         test/macc         │     40.72492980957031     │
│         test/miou         │    24.242835998535156     │
│          test/oa          │     67.54215240478516     │
```

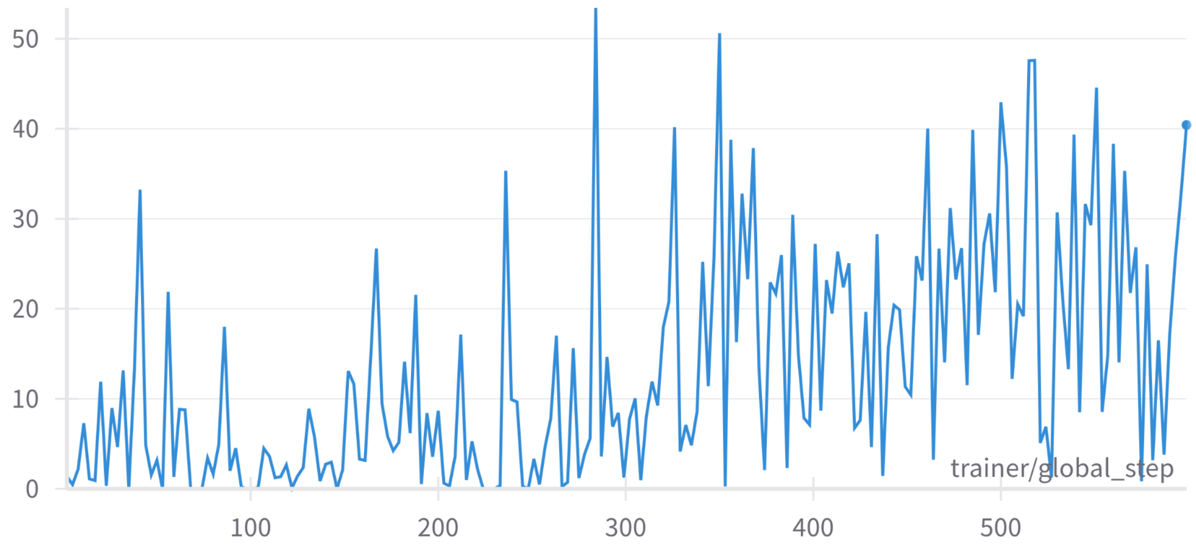
5.3.2 Validation Results

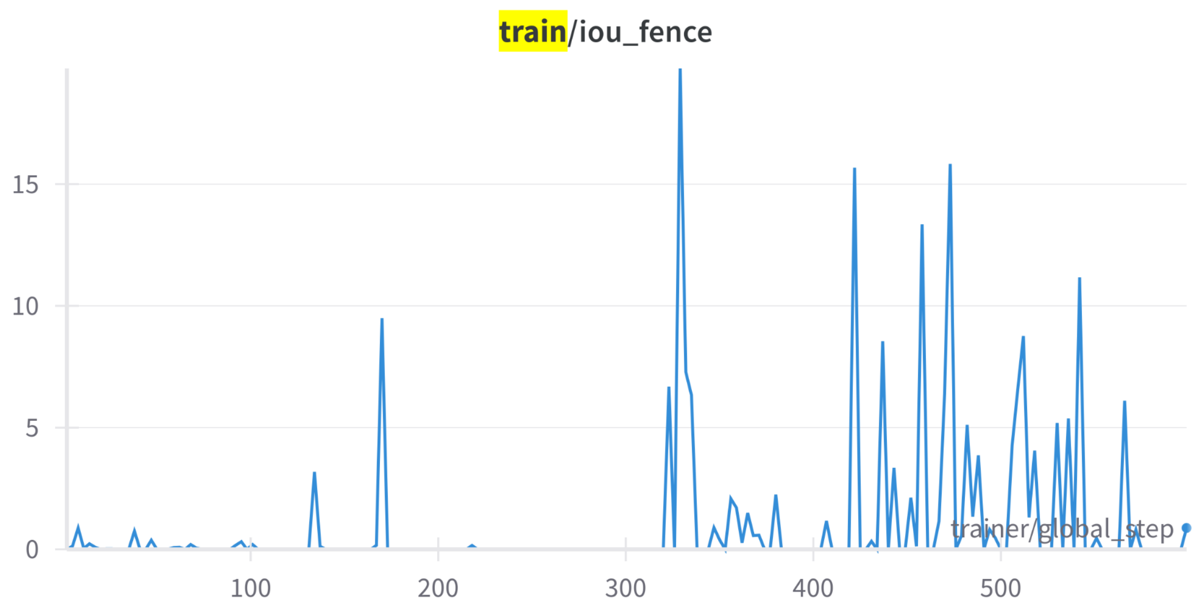


train/iou_road_marking



train/iou_pole





Model Name	Train Accuracy	Test Accuracy	Validation Accuracy	Baseline Accuracy
SPT	49.4%	59.77%	59.475%	31.96% (small objects)
Pointnet	50%	0	0	98.7% (large objects)
DGCNN	50%	0	0	99.8% (large objects)

5.4 Future work and roadmap

We currently have only implemented Online Triplet Loss implementation. In order to increase the accuracy for small object classification, we could leverage PyTorch hard Negative identification using similarity distance to improve accuracy.

Current Transformer is an encoder only network. We could enhance the network to include decoder based transformers to increase self-attention for smaller objects

6. Conclusion

PointNet and DGCNN basic algorithm gives very low training accuracy and was not able to handle the point cloud data point during inference. We could look at dimension reductions before use of PointNet and DGCNN algorithms.
Super Point Transformer with Triplet loss (online version) should provide better performance for small objects. This could be further improved using hard negatives. Overall accuracy of Triplet loss is less than Cross Entropy losses due to poor performance for fence object identification. We could enhance the transformer models to include decoder blocks to increase the attention for sparsely spread data points.

7. References

1. Robert, Damien, Hugo Raguet, and Loic Landrieu. "Efficient 3D Semantic Segmentation with Superpoint Transformer." arXiv preprint arXiv:2306.08045 (2023).
2. Nguyen, Anh, and Bac Le. "3D point cloud segmentation: A survey." In *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*, pp. 225-230. IEEE, 2013.
3. Wang, Yue. "DGCNN: learning point cloud representations by dynamic graph CNN." PhD diss., Massachusetts Institute of Technology, 2020.
4. Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
5. Chefer, Hila, Shir Gur, and Lior Wolf. "Generic attention-model explainability for interpreting bi-modal and encoder-decoder transformers." In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 397-406. 2021.
6. Ge, Weifeng. "Deep metric learning with hierarchical triplet loss." In *Proceedings of the European conference on computer vision (ECCV)*, pp. 269-285. 2018.
7. Qian, Qi, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, and Rong Jin. "Softtriple loss: Deep metric learning without triplet sampling." In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6450-6458. 2019.
8. Implementation Repository Link - https://github.com/shivansh1010/FML-project
9. Pointnet Slides - http://stanford.edu/~rqi/pointnet/
10. Pointnet Paper - https://arxiv.org/pdf/1612.00593.pdf
11. Pointnet Github Code - https://github.com/charlesq34/pointnet
12. DGCNN Paper - https://arxiv.org/pdf/1801.07829.pdf
13. DGCNN Github Code - https://github.com/WangYueFt/dgcnn
14. Superpoint Transformer Paper - https://arxiv.org/abs/2306.08045
15. Superpoint Transformer Github Code - https://github.com/drprojects/superpoint_transformer/tree/master
16. Datasets-
    a. Semantic KITTI - http://www.semantic-kitti.org/dataset.html
    b. Paris-Lille-3D - https://npm3d.fr/paris-lille-3d
    c. Toronto-3D - https://github.com/WeikaiTan/Toronto-3D

Appendix:

1. Feedback Comments

Your report must include details on the following:

1. An enhanced and well-defined problem statement, taking into account the received feedback. - Section 1, 5
2. Description of the proposed solution approach - Section 4
3. A code survey where you include links to the relevant codebases that you refer to while implementing the solution
    a. Section 2 and links are provided in the reference section
4. Datasets - Section 3
5. Implementation details - Section 5
6. Preliminary results - Section 6
7. A roadmap for the remainder of the semester - Section 5.2

2. Abstract Feedback - Response

The proposed task looks good. feasible to finish within time. There are several things you need to take care of:

1. The performance of the proposed models is already reported (as per the dataset details). The project should not just be reproducing those results. You need to introduce originality and should try new approaches to improve upon the mentioned models. Please be more precise about the method (how exactly are you going to tackle the mentioned task).

- The project is not proposing reproducing the results. We have identified these classes Fence, Road Marking and Poles that have low accuracy and propose to apply the concepts learned from class to improve these accuracies.

2. It can't be straightaway judged but please be mindful of the compute power required.

- Yes, we have access to server with Nvidia RTX A4000 GPU with 16GB memory and Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz with 60GB RAM

3. The project should incorporate topics from class also.

- Topics from the class are Kernel Methods, Siamese networks, Triplet Loss, CNN Model and Transformers