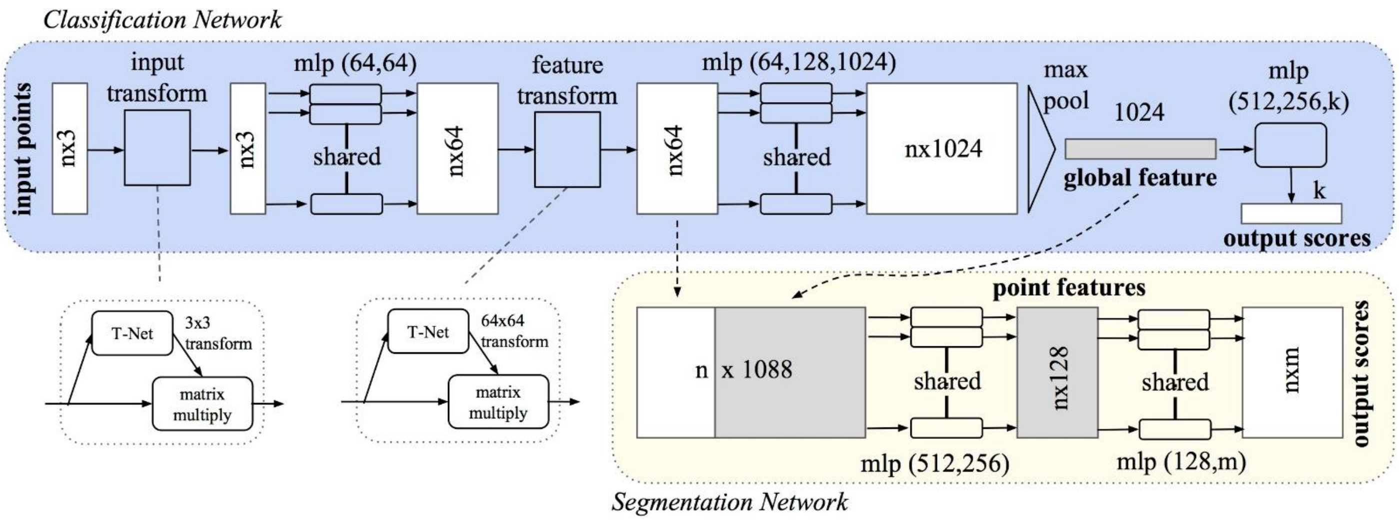
REPORT LAB 4:
Point Cloud Segmentation

Boscolo Simone simone.boscolo.2@studenti.unipd.it N.Matricola: 2026826

1. INTRODUCTION

Point Cloud Segmentation aims to create partitions of a point cloud. In this assignment it is use PointNet, a deep neural network, to achieve this goal. In particular, the network is it trained and tested on the Semantic-Kitti dataset, famous dataset for testing autonomous driving. The dataset contains 30 segmentation labels, that for our lab are reduced to 3: traversable, non-traversable and unknown (outliers).
The assignment is developed with a python notebook, by using Google Colab.



2. IMPLEMENTATION

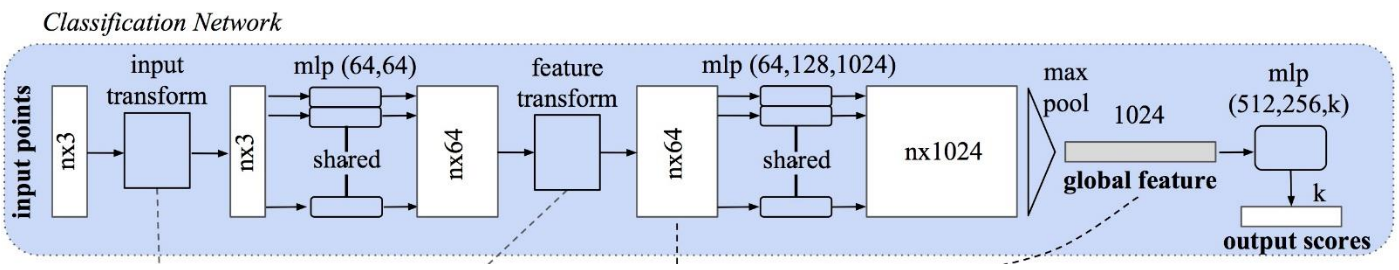
In the provided notebook is left to complete the two main network modules for the segmentation, which are:

A. PointNet (classification network)
B. PointNetSeg (segmentation network)

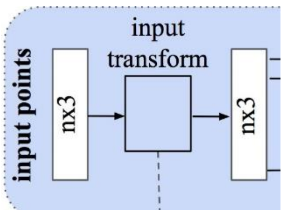
These modules are implemented by two different classes and it is requested to develop the `_init_` and `forward` methods of each class.

In particular: in `_init_` methods are defined the inner module structures, instead in the `forward` methods are defined the feed-forward part of a neural network.

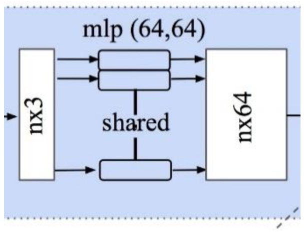
A. PointNet Module



This module is the basement of the PointNet network and it is usually used for classification. In our case the network stops at the `global feature` vector, because we are interested to segment the point cloud.



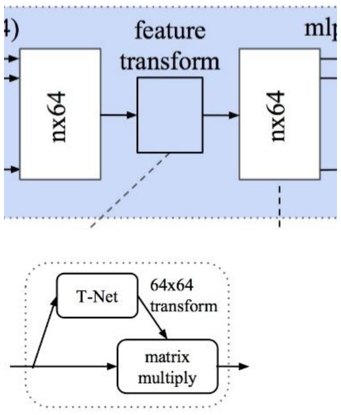
Also, the input transformation is already given with the provided notebook (that is the first part of the previous image)



The transformed input is fed to a MLP network with shared weights and it is implemented by defining 2 MLP objects, respectively:

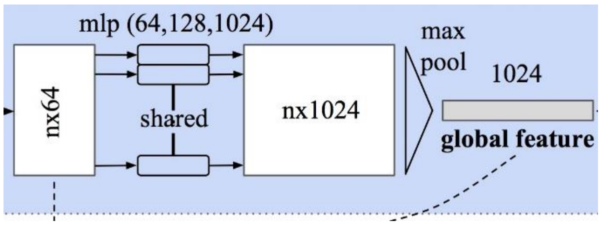
```
1.  mlp1_1 = MLP(3,64)
2.  mlp1_2 = MLP(64,64)
```

where MLP is the Multi-Layer Perceptron class already provided with the notebook.



After that, the feature transformation is implemented with a `TNet` object (that is also already provided as MLP) with `k = 64`, that means it produces a square matrix of size equals to 64, that is the <u>feature transformation matrix</u>. This matrix is computed by feeding the TNet with the output of the previous MLP network.

Then, to complete the feature transformation, the <u>feature transformation matrix</u> is multiplied with the input of the TNet (that is the output of the previous MLP network) by using `torch.bmm` method (that is the batch matrix multiplication provided by the Torch library).



The output of the feature transformation is fed to another MLP network with shared weights and it is defined with 3 MLP objects, as follow:

1. `mlp2_1 = MLP(64,64)`
2. `mlp2_2 = MLP(64,128)`
3. `mlp2_3 = MLP(128,1024)`

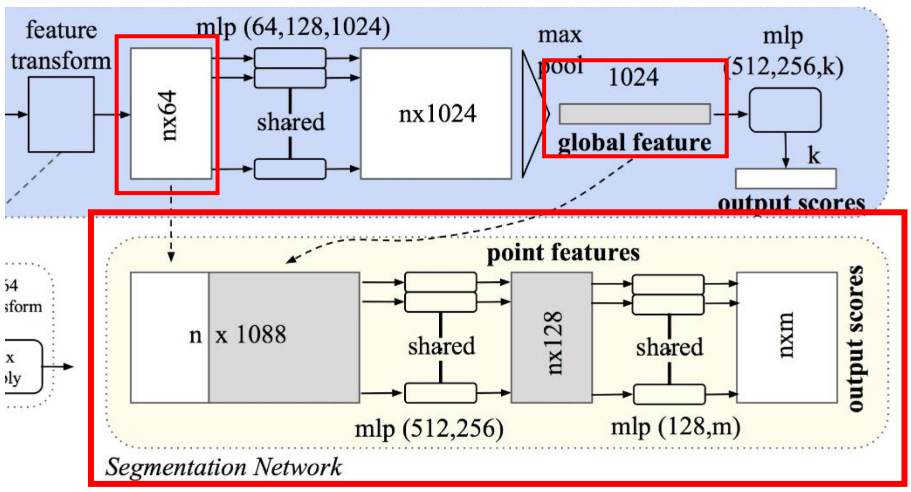
Then the output is passed to a max pooling layer and then to a flatten layer to compute the `global feature` vector, which are respectively implemented by using:

1. torch.nn.MaxPool1d
2. torch.nn.Flatten

At the end, the forward method returns:

- feature transformation output
- global feature matrix of size n x 1024, that is the `global feature` vector repeated n times, where n is the size of the input
- the input transformation matrix
- the feature transformation matrix

B. PointNetSeg Module



In this module the concatenation among the feature transform output and the global feature and the cost function, that is the log of the softmax function, are already given with the provided notebook.

Here the 2 MLP networks are implemented with 4 MLP objects:

```
1. mlp1_1 = MLP (1088, 512)
2. mlp1_2 = MLP (512, 128)
3. mlp2_1 = MLP (128,128)
4. mlp2_2 = MLP (128, m)
```

where m = 3 in this experiment and the output of `mlp1_2` is the point features matrix.

The forward method returns the result of the cost function that has as input the vector that contains the probabilities of a point to belong to each class.

3. EXPERIMENTS

It has been also added to the end of the notebook three personal experiments where:

1. I have **deleted** the `mlp2_1` layer in the **PointNetSeg** module, with respect to the original implementation
2. In **PointNet** module I have **added** module, between `mlp2_2` and `mlp2_3`, two more MLP layers which are: MLP(128, 256) and MLP(256,512) (at the end, this modification results with a MLP(64,128,256,512,1024)), always with respect to the original implementation
3. The third experiment is the sum up of the previous two experiments.

4. RESULTS AND COMMENTS

	TEST ACCURACY	TOTAL TIME	AVG TIME
ORIGINAL IMPLEMENTATION	81.52766666666666 %	23.85502028465271 s	0.795167342821757 s
EXPERIMENT #1	89.396 %	12.973624229431152 s	0.4324541409810384 s
EXPERIMENT #2	88.97766666666666 %	13.170599937438965 s	0.43901999791463214 s
EXPERIMENT #3	92.9485 %	13.365534782409668 s	0.44551782608032225 s

It's interesting to notice that the 3 personal experiments provide much more better results in terms of accuracy and time, with the respect of the original implementation, in this dataset.

It would be interesting to test the same experiments with different datasets and by enlarging the number of classes that it needs to segment, in order to see if it can be obtained the same results.

ALSO, I WANT TO HIGHLIGHT *that by running multiple times the notebook, the results are different but the 3 experiments are still the best ones.*