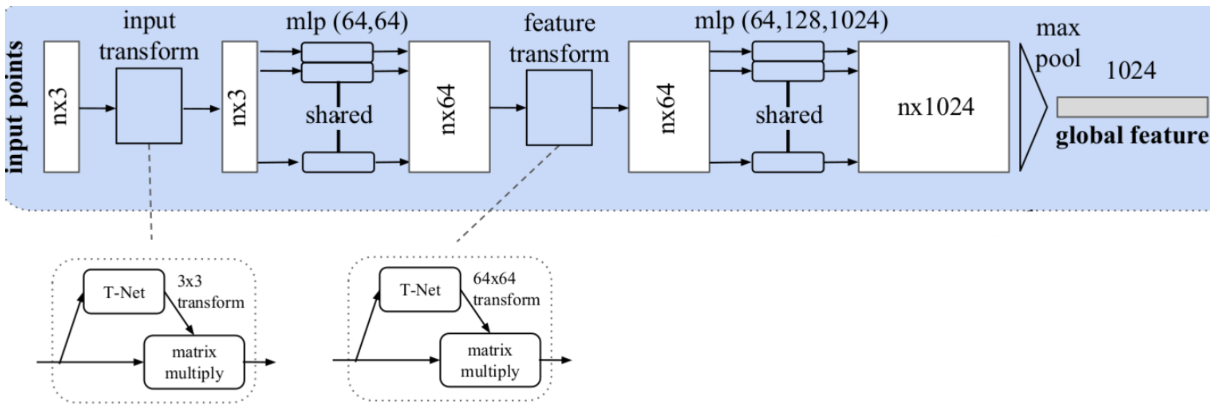
*3D DATA PROCESSING - LAB 4 (**Individual assignment**)*

Topic: Deep 3D descriptors
Goal: Design a modified PointNet architecture that is able to extract 3D descriptors to be used for matching.

Introduction

A 3D local descriptor (or 3D local feature) is a compact representation (i.e., an n-dimensional vector) of the geometric properties of a point *p* in its neighborhood. The neighborhood can be defined as the set of points falling within a spherical region (or support window) of radius *r* around point *p*.

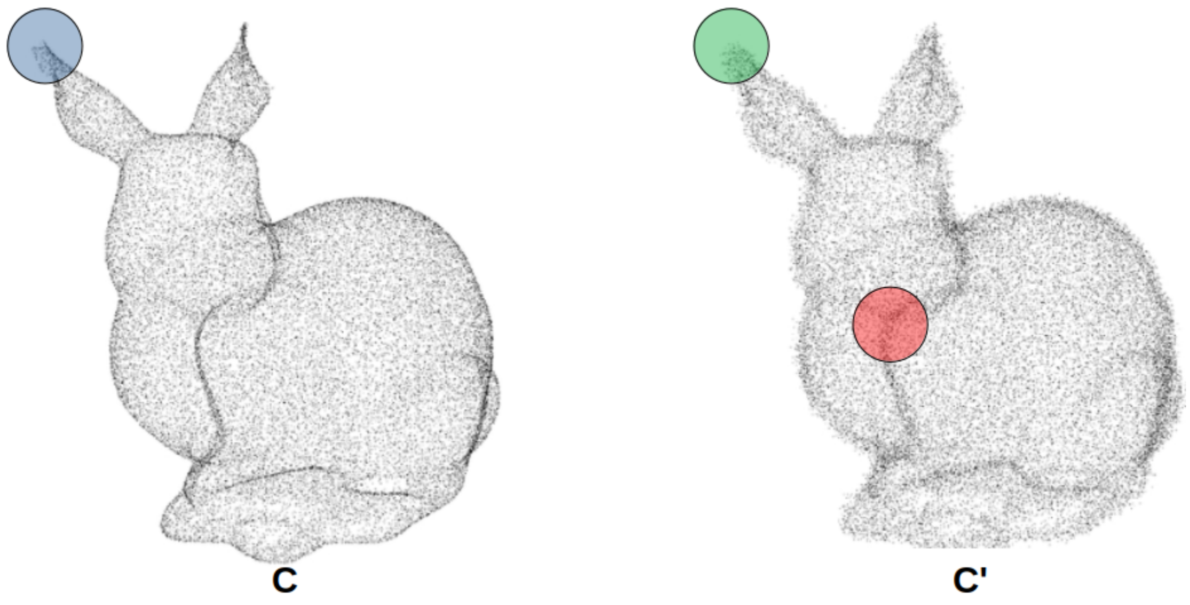
The goal of this assignment is to design a reduced version of the PointNet architecture we call here "**TinyPointNet**" that learns a 3D local feature descriptor from training data. The idea is to use the n-dimensional global feature learned by TinyPointNet directly as a descriptor of a locality of points (neighborhood of a point *p*), removing from PointNet the last MLP (multi-layer perceptron) used for classification tasks.



Unlike the original PointNet where the global feature has dimensions 1024, TinyPointNet could provide a lower dimensional (e.g., 256) global feature. The inner layers of its MLPs could be reduced as well. TinyPointNet should be trained by using a Triplet loss *L* as a loss function:

$$\mathcal{L}(A, P, N) = \max(\| \mathrm{f}(A) - \mathrm{f}(P)\|_2 - \| \mathrm{f}(A) - \mathrm{f}(N)\|_2 + \alpha, 0)$$

where f(X) is the output feature from TinyPointNet for the set of points X (i.e., the neighborhood of some point), A is a reference neighborhood of points (called **anchor**), P is a neighborhood that represents a correct match with respect to A, while N is a neighborhood that represents a wrong match with respect to A. The L2 norm is used to obtain distances from descriptors, α is a margin between positive (A,P) and negative (A,N) pairs.



A dataset composed of 6 point clouds is provided, divided into training and test sets. For each point cloud **C**, a noised version **C’** is created (see picture above). Each descriptor f(X) computed in a neighborhood of a point p on **C** (e.g., the blue spherical support window in the figure) should be correctly matched with a descriptor f(X’) computed in a neighborhood of the same 3D point p (or close to p) on **C’** (e.g., the green spherical support window in the figure), i.e. f(X) should be very close to f(X’). Conversely, if the descriptors are computed at distant points (e.g., the blue and red spherical support windows in the figure), they can be very different. Note that the radius r of the spherical support window is a critical parameter to define the descriptor's locality.

The main step of the assignment are:

- Sample generation.
  - Generates a set of positive and negative pairs of point sets. Each point set is composed of all the points included in a spherical support region with radius r (see picture above) around some point. The support region radius is a parameter that needs to be tuned.
- Design the TinyPointNet architecture
- Train TinyPointNet with the samples previously generated by using the triplet loss function
- Test the TinyPointNet descriptors with a new dataset (i.e., the test set). Note here to match descriptors you should extract the nearest neighbors of descriptors in terms of L2 function.

To perform the assignment complete the missing parts on the provided **Colab notebook**.

Instructions

Complete `__getitem__(self, idx)` of the class `PointCloudData(Dataset)` in order to generate an anchor and a positive and a negative sample. Extract a random point (anchor) and its neighborhood from the starting point cloud `self.pcd1`. The closest point of the noisy point cloud `self.pcd2` will be selected as a positive sample, along with its neighborhood, while the negative sample will be extracted between points far from the anchor. To normalize, subtract the coordinates of the three selected points (anchor, positive and negative sample) from their corresponding sets.

In `TinyPointNet(nn.Module)` complete the `__init__(self)` and `forward(self, input)` following the same structure of TinyPointNet (see the figure above). Set the dimension of the output feature to 256 instead of the original value 1024.

On `train(...)` define `tinypointnetloss` used to train the network.

Note: Inside the notebook you can find comments and step-by-step instructions: refer to them for more details.

What you need to deliver

- Your Colab notebook with the source code you produced
- A short written report with:
  - A brief description of the work done, and the problems encountered, if any, clearly specifying who did what;
  - Some quantitative results in terms of matching accuracy in the test set

Disclaimer

To speed up the training process, and therefore the execution of the assignment, we provided a reduced-size train dataset. On the other hand, a small dataset does not allow to obtain high-accuracy results. To this end, if you can get a final accuracy above 50% that's perfectly fine.