Simone Mosco 2055298

Lab 4: Deep 3D Descriptors

Task 1: Anchors Generation

In the first task it is required to generate an anchor and a positive and a negative sample. To do this, a random point is extracted from the starting point cloud, together with its neighborhood using the Open3d `search_radius_vector_3d()` function, specifying the *radius*. This operation is performed also for the positive and negative sample, using the noisy point cloud. With the `search_knn_vector_3d()` function, the closest point to the previous one is extracted from the noisy point cloud, with its neighborhood. Regarding the negative sample, a random point from the noisy point cloud is selected, making sure it is at a certain distance from the anchor. This check is performed in a simple while loop. Finally the samples are normalized with respect to the center of the sphere. The figure below shows an example of the 3 samples.

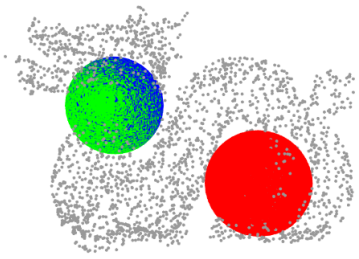

Figure 1: Anchor, positive and negative samples

Task 2: TinyPointNet Structure

The structure of Tiny PointNet is represented in Figure 2. In order to construct the network, the already defined MLP and TNet classes are used in the `__init__()` method, along with the `forward()` method in the `TinyPointNet` class. The first layer employed is a TNet with input $k = 3$, followed by two shared MLPs that map input size 3 to 64 and then 64 to 64. Then anothe TNet is used, this time with $k = 64$ and again some MLPs that maps input size 64 to 256 with an intermediate dimension of 128. At the end a max pooling operation is performed to generate the final global features, a vector of dimension 256.

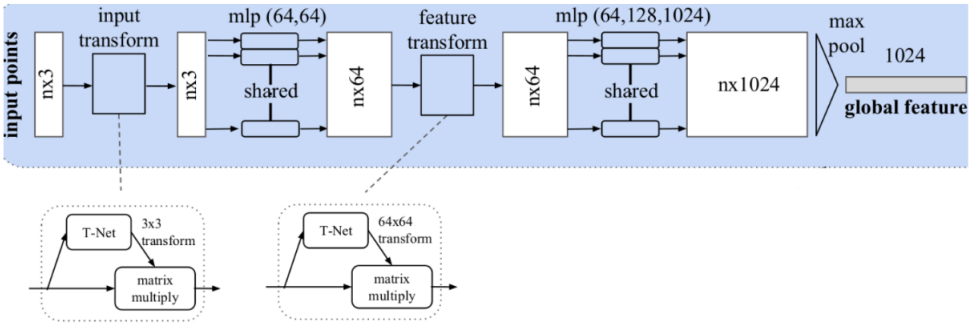

Figure 2: TinyPointNet structure

Task 3: Loss Function

The training operation is performed with a Triplet Loss Function with default parameters as margin equals to 1.0 (α in the equation) and p, the norm degree for pairwise distance as 2. In Pytorch, the loss is defined as `nn.TripletMarginLoss()`.

$$L(A, P, N) = max(\|f(A) - f(P)\| - \|f(A) - f(N)\| + \alpha, 0) \tag{1}$$

Results

Many test have been carried out, trying different values of learning rate and radius, in order to obtain good results and prevent over-fitting. The best values seems to be learning rate equals to 0.001 and radius $3 \cdot 10^{-3}$. This two hyper-parameters allowed the network to reach a lower loss value, both on train and validation sets, as shown in Figure 3.

Regarding the test part, the network is able to achieve an accuracy value of 88.235% on the test set. However, it is important to highlight that different runs produced different accuracy values, most of them in the range $80 - 90\%$, and only once it performed a lower value of 50%.

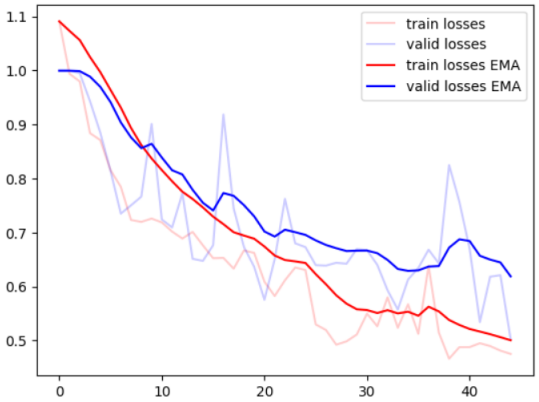

Figure 3: Train and validation losses