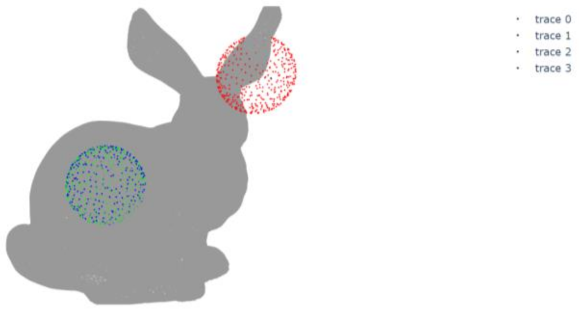
Report Laboratory 4:

Deep 3D Descriptors

Christian Francesco Russo 2087922

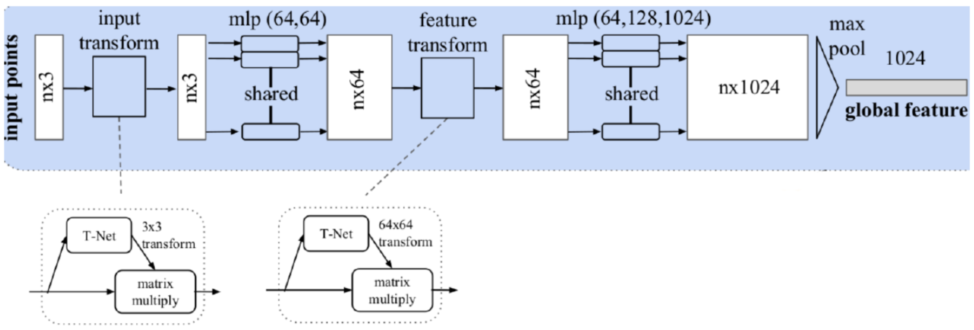
Task 1: Sample generation

To generate an anchor sample we extracted a random point from the point set `pcd_points`, along with its neighboring points using the function `search_radius_vector_3d()` from the Open3D library.
To generate a positive sample we extracted from the noisy point cloud the closest point to the anchor point found before, along with its neighboring points using the function `search_radius_vector_3d()`.
To generate a negative sample we extract a random point from the noisy point cloud, ensuring that this point was farther than a certain threshold from the anchor point found before (by generating random points in a while loop until a sufficiently distant point was found), and with it we extracted also its neighboring points using the function `search_radius_vector_3d()`.



Task 2: TinyPointNet architecture

To implement the TinyPointNet network we followed along the lines the structure of the original PointNet architecture:



with the following differences:

- The first T-Net layer was replaced by the computation of a canonical rotation matrix, which has the same size 3×3, as done in SHOT (Signatures of Histograms of OrienTations).

- The last MLP layers used for classification in the original architecture were removed, in order to get the n-dimensional global feature (in output from max-pooling) learned by the network, and to use it as a 3D local feature descriptor.
- The 1024-dimensional global feature was resized to a 256-dimensional global feature.

The inner layers retained the same size as in the original architecture:

- MLP from input size 3 to output size 64
- MLP from input size 64 to output size 64
- T-Net layer of size 64×64.
- MLP from input size 64 to output size 64
- MLP from input size 64 to output size 128
- MLP from input size 128 to output size 256

Finally, we have a max-pooling layer which produces in output a global feature vector of size 256.
Note: these layers were defined in the function `__init__()` and were coupled in the function `forward()` for building the network.

Task 3: Local reference frame rotation matrix (as for SHOT decriptors)

To implement the Local Reference Frame rotation matrix, we computed a weighted covariance matrix by scaling the centered features by their respective weights and then performing a matrix multiplication with the transpose of the centered features using the PyTorch function `torch.matmul()`.
Then, we computed the eigenvalues and eigenvectors of the weighted covariance matrix using the symmetric/Hermitian eigen-decomposition using the PyTorch function `torch.linalg.eigh()`, which guarantees real eigenvalues and eigenvectors.
These eigenvalues and their corresponding eigenvectors were sorted in decreasing order.

Task 4: Triplet loss function

The loss function was defined as a simple triplet loss function:

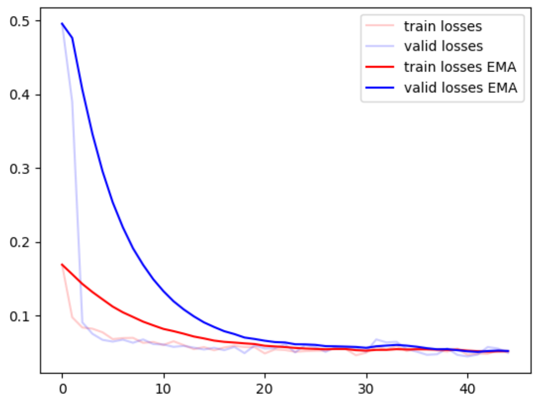
$$\mathcal{L}(A, P, N) = \max(\|f(A) - f(P)\|_2 - \|f(A) - f(N)\|_2 + \alpha, 0)$$

using the PyTorch function `TripletMarginLoss()`, where different tests were performed by changing its margin α.

Results

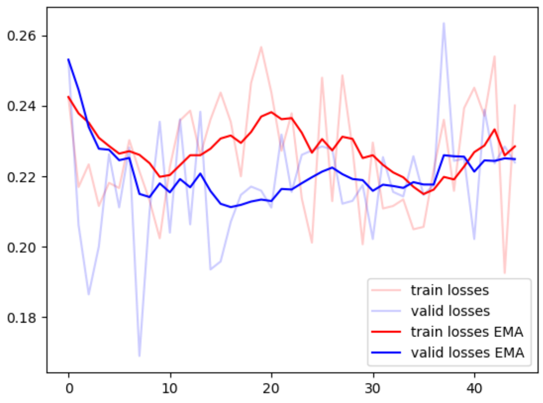
Several tests were conducted to find the optimal combination of hyperparameters:

- **Test 0:** as required by the assignment, the first test was performed by using as first layer the first T-Net (as in the original PointNet architecture), in order to compare the performance with the next tests where the SHOT canonical rotation matrix was instead used:
    - **Training and validation losses:**



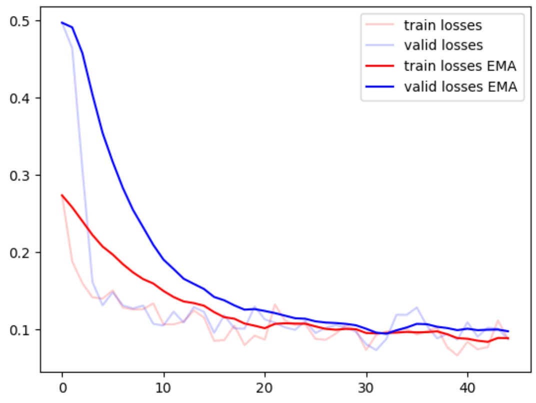
- **Accuracy:** 89.850%.

- **Test 1:** Default parameters (Radius = 0.02, Triplet loss margin = 1.0, lr = 0.0005):
  - **Training and validation losses:**



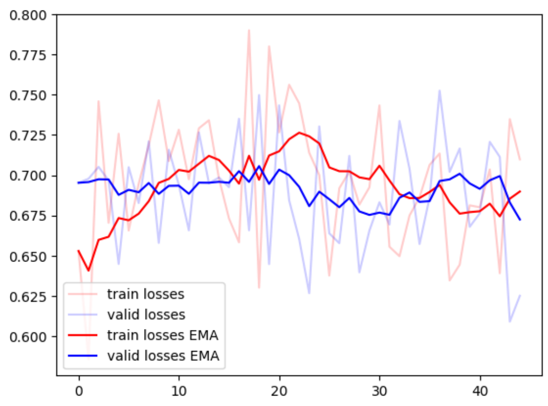
  - **Accuracy:** different runs: 49.150%, 33.100%, 51.850%.

- **Test 2:** Radius = 0.02, Triplet loss margin = 0.5, lr = 0.0005:
  - **Training and validation losses:**



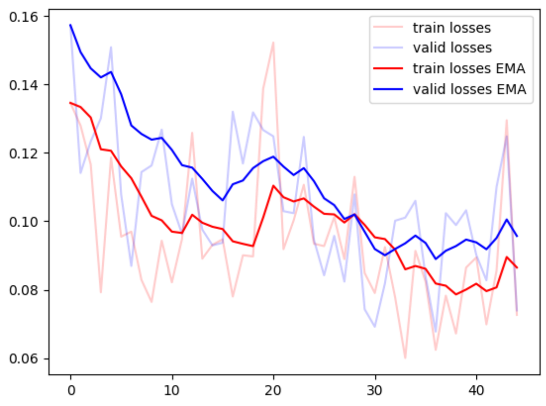
  - **Accuracy:** different runs: 49.900%, 29.500%, 52.200%.

- **Test 3:** Radius = 0.02, Triplet loss margin = 2.0, lr = 0.0005:
  - **Training and validation losses:**



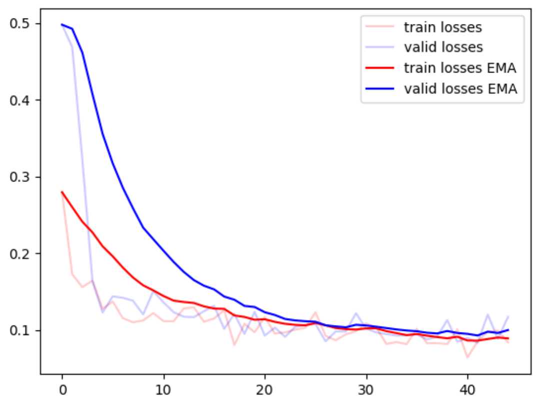
- **Accuracy:** different runs: 30.500%, 23.100%, 51.100%.

- **Test 4:** Radius $= 0.02$, Triplet loss margin $= 0.5$, lr $= 0.001$:
  - **Training and validation losses:**



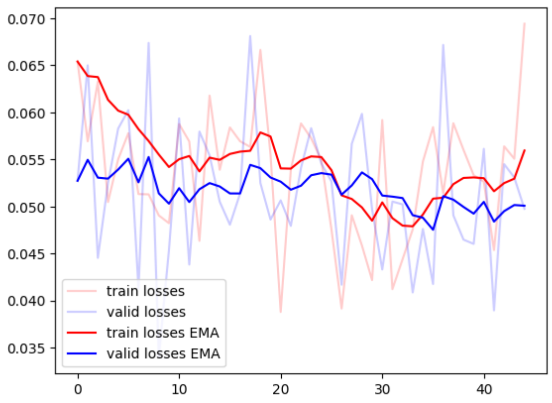
  - **Accuracy:** different runs: 48.480%, 32.400, 44.140%.

- **Test 5:** Radius $= 0.02$, Triplet loss margin $= 0.5$, lr $= 0.0001$:
  - **Training and validation losses:**



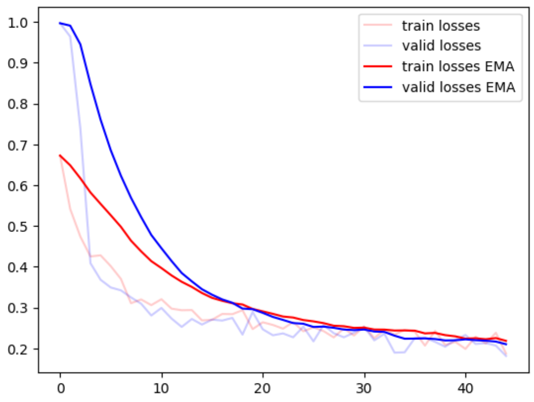
  - **Accuracy:** different runs: 40.300%, 50.450%, 54.300%.

- **Test 6:** Radius $= 0.02$, Triplet loss margin $= 0.5$, lr $= 0.00005$:
  - **Training and validation losses:**



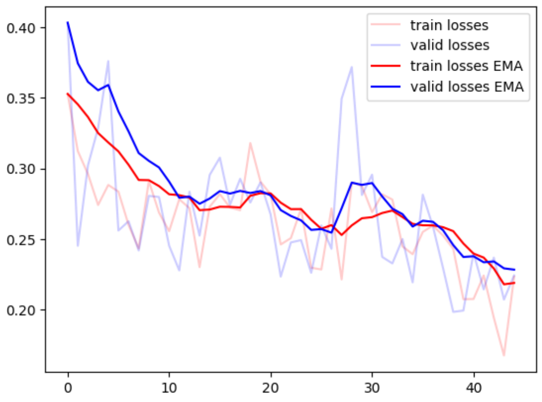
- **Accuracy:** different runs: 31.750%, 40.500%, 41.120%.

- **Test 7:** Radius = 0.02, Triplet loss margin = 1, lr = 0.0001:
  - **Training and validation losses:**



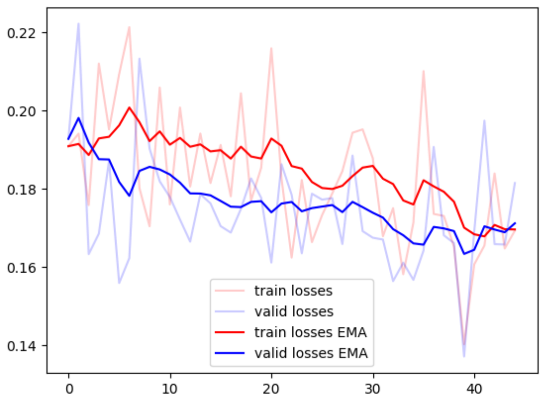
  - **Accuracy:** different runs: 60.200%, 52.470%, 61.320%.

- **Test 8:** Radius = 0.02, Triplet loss margin = 1, lr = 0.001:
  - **Training and validation losses:**

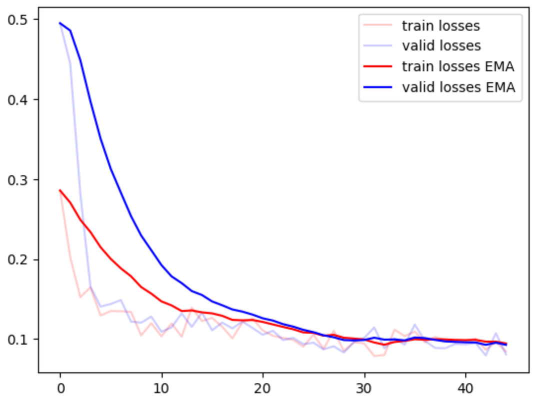


  - **Accuracy:** 31.950%.

- **Test 9:** Radius = 0.02, Triplet loss margin = 1, lr = 0.00005:
  - **Training and validation losses:**



    - **Accuracy:** 30.100%.
- **Test 10:** Radius = 0.2, Triplet loss margin = 1, lr = 0.00005:
    - **Training and validation losses:**



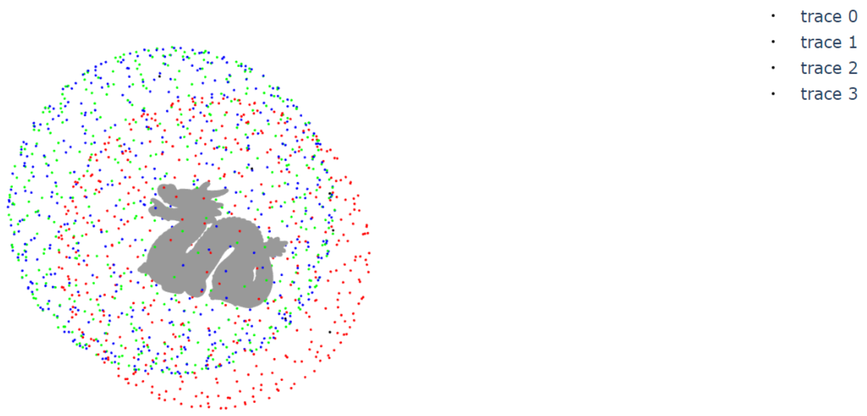
    - **Accuracy: 99.950%.**

The results shows that using as first layer T-Net, as in the original PointNet architecture, the accuracy, with default parameters, is 89.850% on the test set, which is pretty good.

Instead, substituting this first layer with the SHOT canonical rotation matrix with default parameters we have really bad performance.

In order to achieve good performance, we had to perform several tests. Finally, we found that with the combination of parameters Radius = 0.2, Triplet loss margin = 1, lr = 0.00005, we were able to reach 99.950% of accuracy on the test set, which is almost perfect!

Although, we have to mention that this setting is not ideal since, as shown in the picture below, the radius for the samples contains the whole point cloud.



Additionally, it is worth to report that with default, or “almost default” parameters, the accuracy may change a lot over the different runs, i.e., it has a really high variance: generally $\pm 15\%$, which is not good.

Finally, we mention that the best accuracy of 61.320%, achieved without touching the radius, was obtained with the following parameters $\text{Triplet loss margin} = 1$, $\text{lr} = 0.0001$.