3D DATA PROCESSING

LAB 4

Giuseppe Labate mat.2095665

Topic: Deep 3D descriptors
Goal: Design a modified PointNet architecture that can extract 3D descriptors for matching. Replace the initial 3x3 T-Net with a rotation matrix extracted as in the Local Reference Frame of SHOT descriptors.

1) Work description

Task 1 – `PointCloudData`

For this task, the objective was to find anchor, positive and negative points along with their respective neighborhood.

To sample the anchor point, a random point belonging to pcd_points is selected using:
`random_idx = np.random.randint(0, len(pcd_points))` (to select a random index)

`anchor_pt = pcd_points[random_idx]`

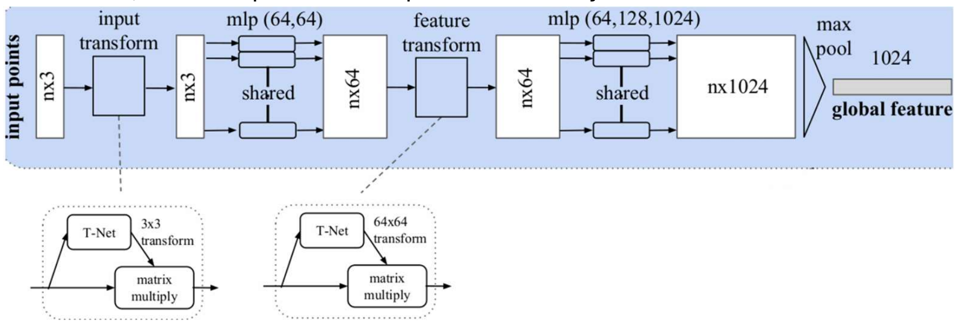
To sample the positive point, the anchor_pt nearest neighbor index is found in the noise point cloud by leveraging a KD-Tree.
This index is then used to select the positive point in the noise point cloud.

To sample the negative point, a random point is selected from the noise point cloud.
It's then checked if its distance from the anchor_pt is greater than the min_dist.
If so, the negative point is found.
Otherwise, another candidate point will be selected.

For each point, their neighborhood is determined by selecting all the point cloud points within a radius of `self.radius`.

Task 2 – `TinyPointNet`

In this task, it was requested to implement the TinyPointNet architecture:



To do this, the pipeline was followed, placing a first MLP (64, 64) followed by a feature transform and finally an MLP (64,128,256) followed by a maxpool.

- `mlp1=MLP(3,64)`
- `mlp2=MLP(64,64)`
- `mlp3=MLP(64,64)`
- `mlp4=MLP(64,128)`
- `mlp5=MLP(128,256)`

The first two lines were used to compute MLP(64,64) and the last three for MLP(64,128,256).

Instead of using a 3x3 TNet at the beginning of the network, the SHOT canonical rotation is utilized.

Here, a weighted covariance matrix is calculated and then eigenvalues and eigenvectors are extracted.
To compute the weighted covariance the j-th vector is multiplied with itself to get the outer product, then multiplied by its corresponding weight.
Next, all the outer products are summed to produce the weighted covariance matrix.

Finally, to evaluate the eigenvalues and eigenvectors, it is used the following line:
`torch.linalg.eig()`

Note that this method may give in output not sorted and complex eigenvectors and eigenvalues. To solve this problem, these lines are exploited.

```
eigvals = eigvals.real

eigvecs = eigvecs.real

indices = torch.sort(eigvals, descending=True)[1]

V = eigvecs [:, indices]
```

This is done to obtain the rotation matrix to be applied to the point cloud.

Task 3 – Loss Function

In task 3 the objective was to use the correct loss function.
For this project, a triplet loss was utilized as loss function:
$$\mathcal{L}\left(A,P,N\right) = \max(\|\,\mathrm{f}(A) - \mathrm{f}(P)\|_2 - \|\,\mathrm{f}(A) - \mathrm{f}(N)\|_2 + \alpha, 0)$$
Where A, P and N stand for anchor, positive and negative, respectively.

The loss is computed as the maximum between 0 and the difference of the L2 distances between anchor and positive descriptors and the anchor and negative descriptors.

To implement this loss, the following code was used:
```
tinypointnetloss = nn.TripletMarginLoss(margin = 1, p = 2).
```
Here margin = 1 is the α, margin between positive (A,P) and negative (A, N) pairs and p = 2 means that we're using L2 distance.

2) Encountered problems

The biggest problems I've encountered during this project are:

- The need to constantly monitor the execution on Colab every 5 minutes because, during training, it frequently disconnected from the runtime or asked me if I was a robot.
  These disconnections forced me to rerun the code multiple times, resulting in significant time wastage, because of the long training computation.
  Additionally, I had limited number of attempts with Colab GPU and often I had to use the much slower CPU.
- The accuracy wasn't great enough (less than 25%) unless I increased the radius.
  While this adjustment led to better results, it also resulted in much larger spheres, some as big as half of the point cloud, which isn't ideal.

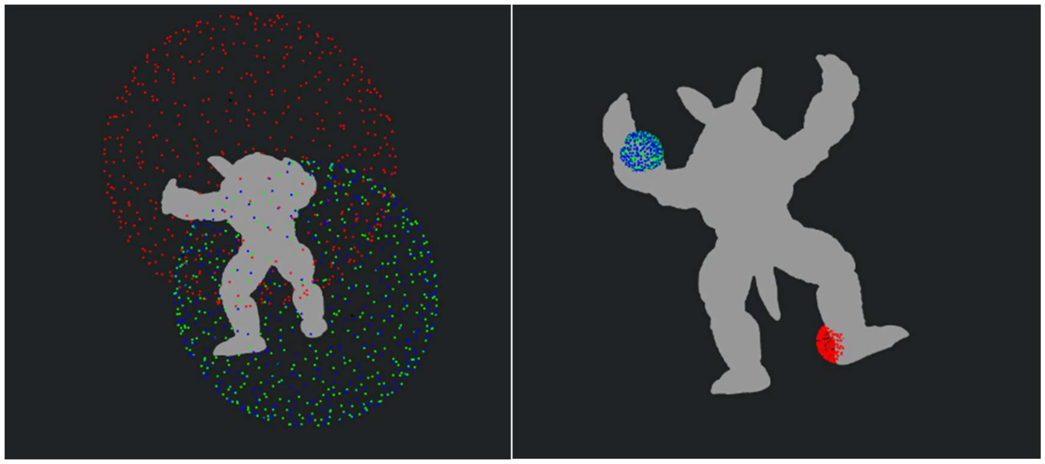


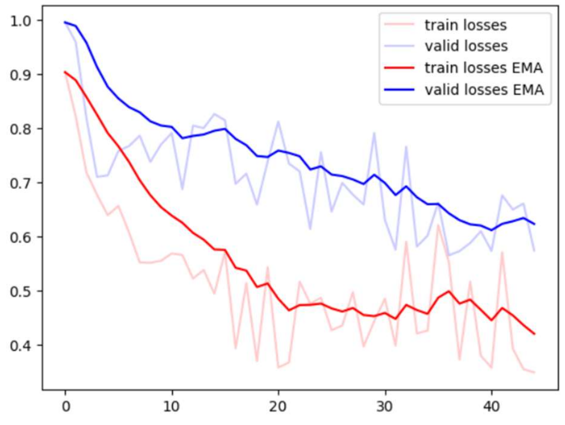
Figure 1: difference in sphere size between radius = 0.2(left) and radius = 0.02(right)

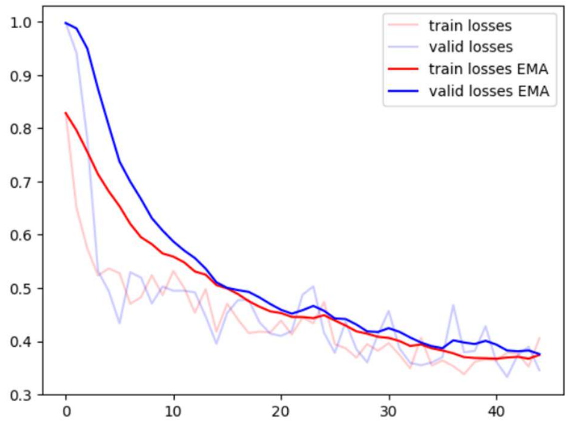
3) Results

After running multiple times, I've achieved a max of 23.65% of accuracy with the given parameters.
By incrementing the radius the results are:

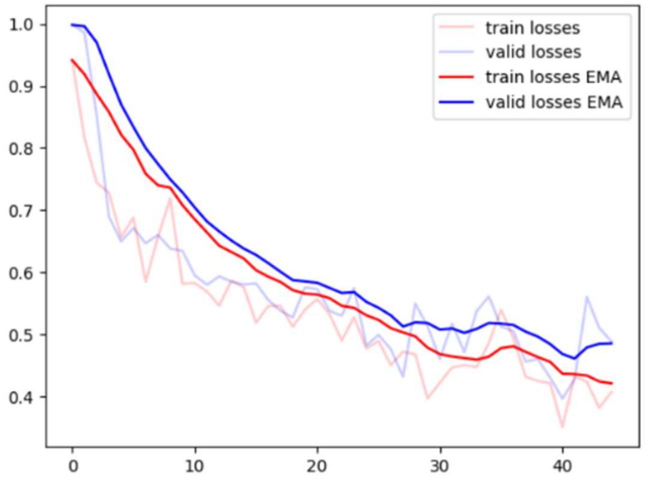
RADIUS	ACCURACY(%)
0.02	23.65%
0.035	26.8%
0.07	57.40%
0.0726	63.70%
0.1	74.10%
0.2	99.95%

Radius = 0.02 (left 18%, right 22%)

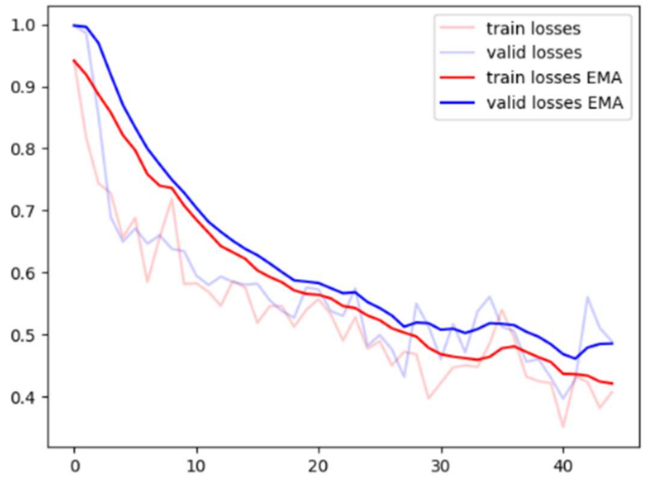




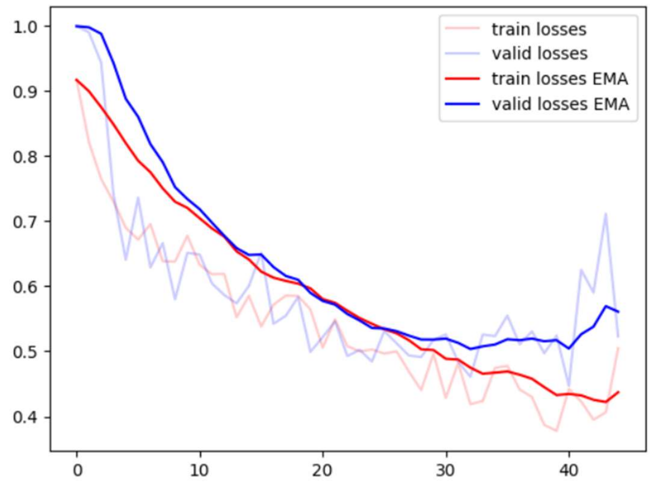
Radius = 0.03



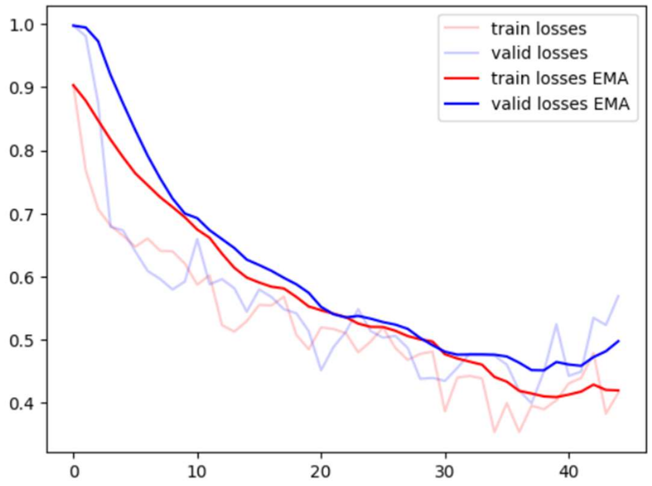
Radius = 0.035



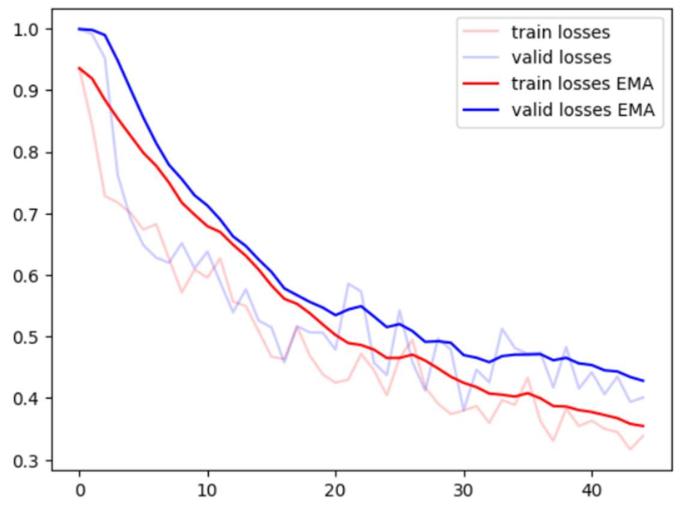
Radius = 0.07



Radius = 0.0726



Radius = 0.2


train losses
valid losses
train losses EMA
valid losses EMA
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40