Lab 4: Deep 3D descriptors

Mohammadhossein Akbari Moafi

Task 1: sample_anchor_point

The sample_anchor_point function performs the following steps:

1. **Random Selection**: Initially, a random index within the point cloud is chosen to select an anchor point.
2. **Neighborhood Extraction**: The neighborhood of the selected anchor point is obtained using a KDTree structure, considering points within a defined radius.

Task 2: sample_positive_point

The positive sample is selected from the noisy version of the point cloud, aiming to find a corresponding point to the anchor point. This selection process is vital for generating informative triplets. The sample_positive_point function conducts the following steps:

1. **Nearest Neighbor Search**: Utilizing the KDTree structure of the noisy point cloud, the point closest to the anchor point is identified as the positive sample.
2. **Neighborhood Extraction**: Similar to the anchor point selection, the neighborhood of the positive sample is obtained for further context.

Task 3: sample_negative_point

The negative sample is chosen to be sufficiently distant from the anchor point, ensuring that it provides valuable contrast during model training. The sample_negative_point function executes the following steps:

1. **Random Point Selection**: Initially, a random point from the noisy point cloud is chosen as a candidate negative sample.
2. **Distance Verification**: The distance between the candidate point and the anchor point is computed. If the distance exceeds a predefined minimum threshold, the candidate point is accepted as the negative sample; otherwise, the selection process continues until a suitable candidate is found.
3. **Neighborhood Extraction**: Similar to the other samples, the neighborhood of the negative sample is extracted for context.

Task4: Model Completion: TinyPointNet

MLP Layers(__init__):
Multiple MLP layers were defined to capture intricate patterns and relationships within the input data. This includes two MLP layers (mlp_11 and mlp_12) with dimensions (3, 64) and (64, 64) respectively, as well as three additional MLP layers (mlp_21, mlp_22, and mlp_23) with dimensions (64, 64), (64, 128), and (128, 256) respectively.

MLP Layers(forward):

The transformed input was sequentially passed through multiple MLP layers (mlp_11 and mlp_12), enabling the model to capture complex features and patterns within the data.

Following the initial feature extraction, a feature transformation step was applied using the feature_transform module, refining the learned features to enhance discriminability and robustness.

The transformed features were further processed through three additional MLP layers (mlp_21, mlp_22, and mlp_33), with dimensions (64, 64), (64, 128), and (128, 256) respectively, enabling the model to capture hierarchical abstractions and representations.

Finally, global max pooling was applied to aggregate the extracted features across all points, producing a compact global feature representation.

Task 5: shot_canonical_rotation

Weighted Covariance Matrix: The weighted covariance matrix is computed using the centered features and their corresponding weights. This matrix captures the statistical relationships and variability within the input data, providing essential information for subsequent computations.

Eigenvector Calculation: Once the weighted covariance matrix is computed, the next step is to determine the eigenvectors of this matrix. These eigenvectors represent the principal axes of variation within the data.

Task 6: Loss Function

The model was trained using a custom triplet loss function with a margin of 0.2. This loss function ensures that the anchor point is closer to the positive point than to the negative point by at least the margin. The Triplet Loss function calculates the Euclidean distances between the anchor-positive and anchor-negative pairs. The loss is then computed by subtracting the anchor-negative distance from the anchor-positive distance and adding a margin (alpha). The result is clamped to ensure non-negative values, and the mean loss over the batch is taken.

During experimentation, I tested margins of 1.0 and 0.2 for both the custom triplet loss function and PyTorch's TripletMarginLoss. The margin of 0.2 for the custom loss function resulted in better accuracy,

Quantitative Results

Each train epoch and validation epoch took 8 seconds to complete. The model was evaluated on a test set to determine its accuracy in matching descriptors. The **Custom Triplet Loss(Margin 0.2)** with **72.450%** accuracy has the best performance. By uncommenting the bonus study part with default values with **Custom Triplet Loss(Margin 0.2)** the accuracy reached **94.118%** .

	Best Loss	**Accuracy(Default)**	**Accuracy(Bonus)**
Custom Triplet Loss(Margin 1.0)	0.04218264184892177	69.650%	88.235%
Custom Triplet Loss(Margin 0.2)	**0.010953671601600944**	**72.450%**	**94.118%**
PyTorch TripletMarginLoss (Margin 1.0)	0.18183238431811333	41.650%	83.333%
PyTorch TripletMarginLoss (Margin 0.2)	0.0237608103081584	60.050%	90.444%

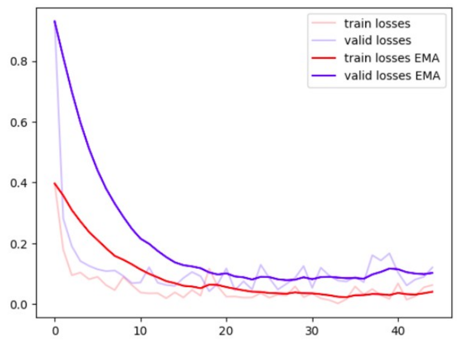

Figure 1: Custom Loss Function(Margin:1.0)

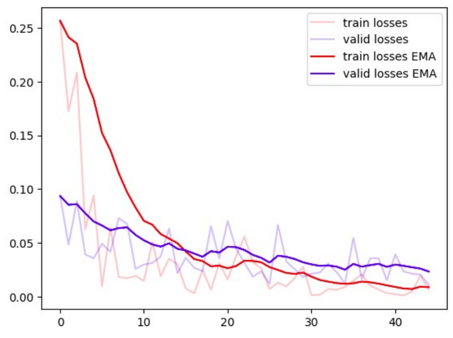

Figure 2: Custom Loss Function(Margin:0.2)

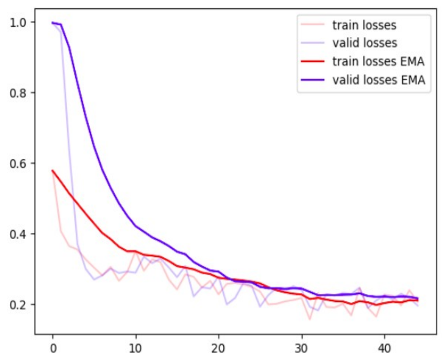

Figure 3: PyTorch Loss Function(Margin:1.0)

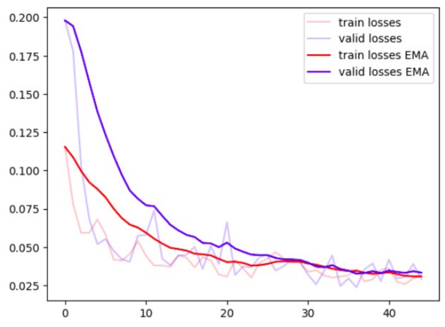

Figure 4: PyTorch Loss Function(Margin:0.2)