

Intelligent Visual Computing [Spring 2023]

[Moodle home](#) / [My courses](#) / [COMPSCI574 COMPSCI674 147255 SP23](#) / [Week 7: Point-based Nets](#)

/ [Assignment 3: Point descriptors and alignment](#)

Assignment 3: Point descriptors and alignment

Overview

In this assignment you will learn to implement a neural network architecture for processing point clouds and mesh vertices with PyTorch. Given an input polygon mesh of a 3D character and a point cloud representing a similar human (see Figure 1 below), your goal is to compute:

- (a) per-vertex descriptors for the mesh and per-point descriptors for the point cloud,
- (b) use these descriptors to compute correspondences between mesh vertices and points in the point cloud,
- (c) compute a correspondence mask, which is 1 for a mesh vertex that has a corresponding point, and 0 for a mesh vertex with no correspondence; the reason for this mask is that the point cloud is partial, capturing only the front of the human. Thus, the correspondences must be partial: only a subset of mesh vertices have correspondences.
- (d) use these correspondences to compute a rotation matrix that best aligns the point cloud with the mesh

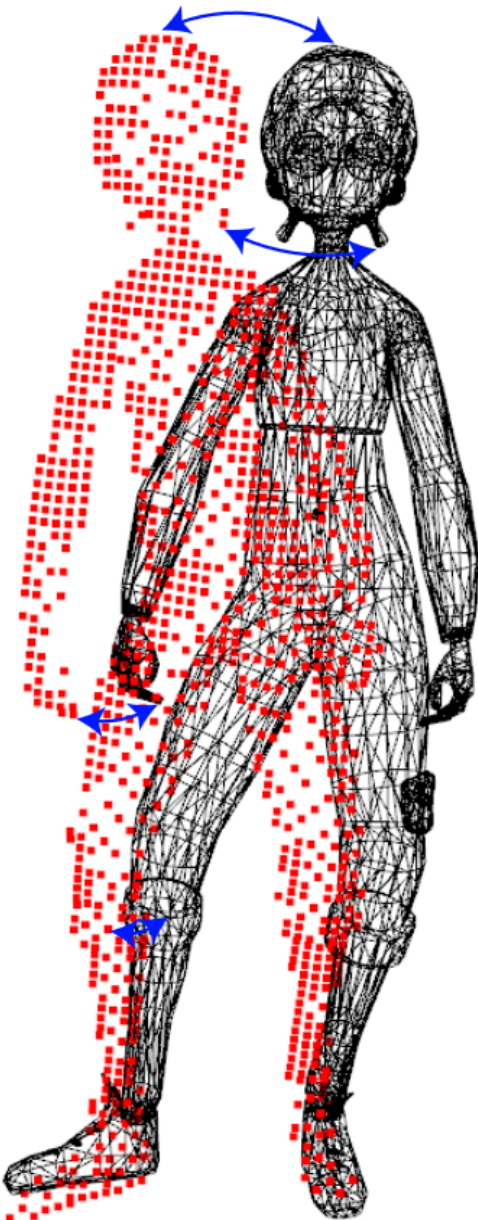


Figure 1: A polygon mesh of a human figure, shown as black wireframe, and a partial point cloud shown as red. A few correspondences are highlighted with blue arrows.

To perform the above tasks, you will use a neural network architecture based on PointNet. The assignment provides you with a (toy) training dataset consisting of 60 pairs of meshes and point clouds (human at different poses). The dataset also contains a hold-out validation dataset with 20 pairs, and test dataset of 20 pairs where you will evaluate your network on each pair.

This assignment counts for **20 points** towards your final grade if you are taking the 574 section. **If you are taking the 674 section, multiply your total points by 0.75** (i.e., the assignment counts for 15 points).

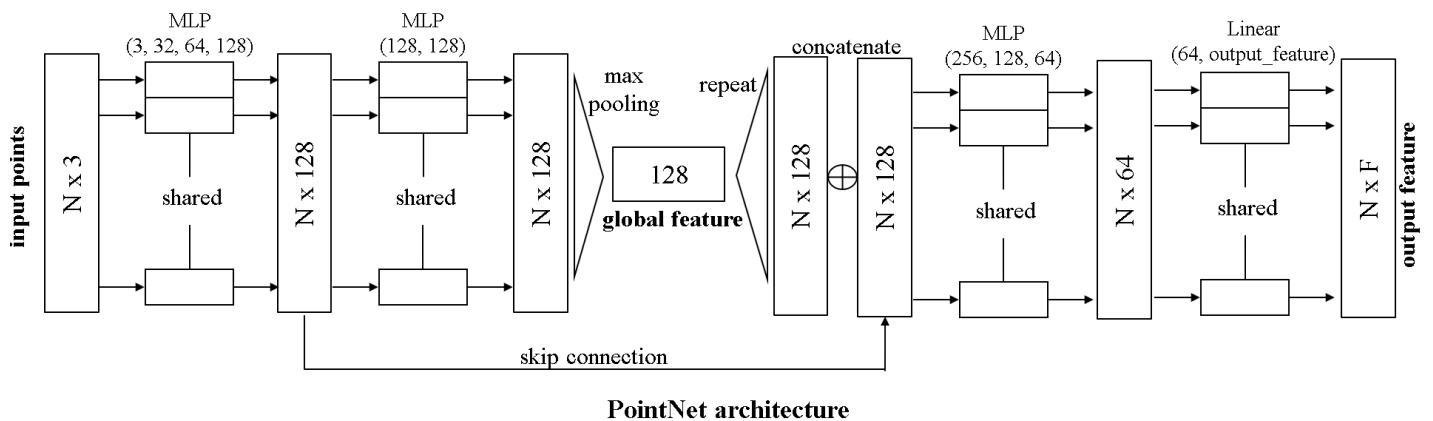
Instructions

You will need to install the [pytorch](#) library to run the code successfully. Download also the starter code (starter.zip) and the dataset (dataset.zip) below. Unzip the dataset inside the starter code folder (i.e., the 'train' folder should be *starter/dataset/train*). When you are done, you can start working on the "model.py" and "train.py" scripts, as required by this assignment. I would recommend you to start by reading the comments and understanding the code in "train.py", then "model.py".

Note: the network you will implement is small, the dataset is small, and can be trained on the CPU (i.e., it takes ~5 min in my laptop). If you want to use a GPU (this is not required by this assignment), first you need to have access to a graphics card with at least 2GB memory, and also enable CUDA in your pytorch implementation. Also note that this task has been extremely simplified so that you train/test the network on the CPU. In general, the correspondence problem is much harder, and you would need much more powerful networks trained on much more complex datasets containing many more different shapes and poses.

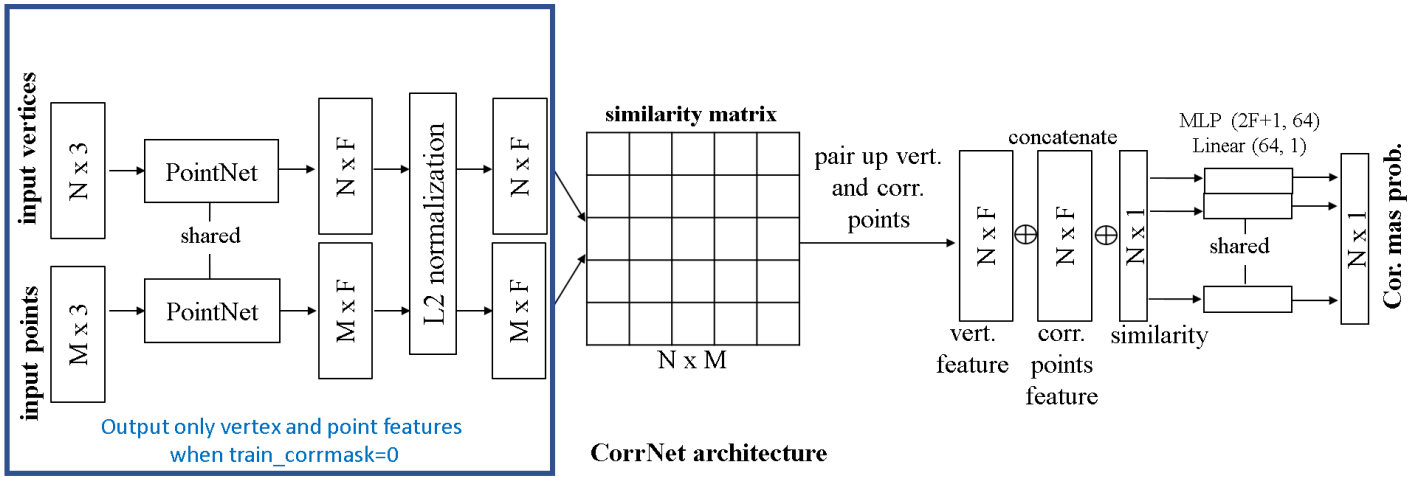
What You Need To Do

Task A [25%]: Change the starter code in "model.py" (class *PointNet*) so that you define the following PointNet:



The module takes as input a set of points ($N \times 3$) represented by their raw 3D positions, which can be either the mesh vertices or the points from the point cloud. The output is a descriptor, or in other words, a feature representation for each input point (or vertex). You will use the same PointNet to process both. Each point i is first transformed to a 128-dimensional representation \mathbf{f}_i through a MLP with 3 hidden layers consisting of 32, 64, and 128 nodes. Each hidden layer uses a leaky ReLU activation and group normalization (to this end, use the MLP block provided in the starter code). Another MLP with a single hidden layer of 128 nodes processes each point representation into the 128-dimensional representation \mathbf{h}_i . A max pooling layer collapses the point representations into a global shape 128-dimensional representation \mathbf{g} . Then the representations \mathbf{f}_i are concatenated with \mathbf{g} , resulting in a new per-point 256-dimensional representation $[\mathbf{f}_i, \mathbf{g}]$. This is transformed into the final output, per-point descriptor \mathbf{y}_i through a MLP (see figure above) and a linear transformation layer. The dimensionality of the output descriptor is 32-dimensional (output_feature=32 in the above figure), as set in the starter code.

Task B [25%]: Change the starter code in "model.py" (class *CorrNet*) so that you define the following network for computing descriptors for both the point cloud and mesh, then use those to compute the correspondence mask:



The module CorrNet takes as input a flag, called *train_corrmask*. When it is 0, the module outputs a $N \times F$ matrix \mathbf{Y} storing the per-vertex mesh descriptors (N is the number of vertices, $F=32$ the descriptor dimensionality), and the $M \times F$ matrix \mathbf{Y}' storing the per-point point descriptors from the point cloud (M is the number of points, $F=32$ the descriptor dimensionality). Both descriptors should be extracted by the same PointNet module. The per-point and per-vertex descriptors should be normalized to have unit length.

When *train_corrmask*=1, the module also outputs the correspondence mask. Specifically, you use the cosine similarity to create a similarity matrix $N \times M$ where each entry (i,j) stores the similarity of mesh vertex i with point j . Then for each mesh vertex i , you will find the most similar point in the point cloud and store it in an array i.e., $n[i]$ returns the most similar point to vertex i according to the above similarity matrix. Then concatenate the following matrices (a) the $N \times F$ matrix \mathbf{Y} storing the per-vertex mesh descriptors, (b) the $N \times F$ matrix \mathbf{X} storing for each row the descriptor of the point $n[i]$ of the point cloud, (c) the $N \times 1$ vector S storing the similarity of $(i, n[i])$ from the above similarity matrix. The result is a new matrix with dimensionality $N \times (2F+1)$. Transform this matrix into a $N \times 1$ correspondence mask through a MLP with one hidden layer of 64 nodes applies to each row, followed by a linear transformation layer that outputs a per-vertex correspondence confidence.

Task C [15%]: Change the code in "*train.py*" (function *NTcrossentropy*) to implement a loss for training the CorrNet and PointNet such that they output descriptors that are as-similar-as possible for ground-truth corresponding points (i,j) between the mesh and the point cloud, and different for non-corresponding pairs. The function takes as input the $N \times F$ matrix \mathbf{Y} storing the per-vertex mesh descriptors (*vtx_feature*), and the $M \times F$ matrix \mathbf{Y}' (*pts_feature*) storing the per-point point descriptors from the point cloud. It also takes as input a $K \times 2$ matrix (*corr*) whose each entry stores the ground-truth correspondence pairs (i,j) . Note that their number of pairs can be different for each training pair of mesh and point cloud. The loss for a ground-truth corresponding pair (i,j) ("positive pair") should be the following one, a form of "Normalized Temperature-scaled Cross Entropy Loss" [https://paperswithcode.com/method/nt-xent]:

$$L_{i,j} = -\log \frac{\exp(\mathbf{y}_i \bullet \mathbf{y}_j' / \tau)}{\sum_{k=1}^M \exp(\mathbf{y}_i \bullet \mathbf{y}_k' / \tau)}$$

where (i,j) is the ground-truth corresponding pair, \mathbf{y}_i is the descriptor of vertex i , \mathbf{y}_j' is the descriptor of point j corresponding to vertex i , \mathbf{y}_k' is the descriptor of (any) point k in the point cloud. The temperature τ is used to scale the cosine similarities (set to 0.07 in the starter code). The loss should be summed over all corresponding pairs.

Task D [10%]: At each training epoch, the starter code evaluates the accuracy of correspondences, and the accuracy of the correspondence mask (see *calc_matching_accuracy*, *calc_mask_accuracy*). Train the architecture with *train_corrmask*=0, and report the correspondence (matching) accuracy at distance thresholds **0.01**, **0.02**, and **0.04** (see *distance_threshold* input parameter) at the best epoch (epoch with highest hold-out validation accuracy). Then train the architecture with *train_corrmask*=1 and report the correspondence mask accuracy (one value) at the best epoch. Include all the above values in a README.TXT file in your ZIP submission.


Task E [25%]: At each training epoch, the starter code calls the function *fit_rotation*. The function is supposed to compute the optimal rotation matrix aligning each point cloud to the mesh. Use the material for the alignment lecture (i.e., computation of optimal rotation through SVD) to compute the optimal rotation aligning each point cloud to the mesh based on the predicted correspondences and mask. You should use "reliable" correspondences i.e., correspondences where the confidence mask is high (above 0.5). The starter code already computes the Euler rotation angles based on the rotation matrix you provide. The angles are averaged over all pairs of test meshes and point clouds (all point clouds are rotated wrt to the mesh using the same amount of rotation). Report the 3D rotation angles for the test set at the best epoch in your README.TXT.

Submission:

Please follow the **Submission** instructions in the course policy to upload your zip file to Gradescope. The zip file should contain all the code and a short TXT for your report. **Please do not include the pytorch libraries, trained model, checkpoints, or the dataset in your submission!**

-  [dataset.zip](#)  March 28 2023, 5:12 PM
-  [starter.zip](#)  April 7 2023, 1:20 PM

Submission status

Submission status	This assignment does not require you to submit anything online
Grading status	Not graded
Due date	Thursday, April 13, 2023, 5:00 PM
Time remaining	5 days 18 hours
Last modified	-
Submission comments	 Comments (0)

◀ Thu Class Meeting

Jump to...

Lecture Notes ▶