THIS FILE WAS ABOUT DEEP LEARNING SO I'LL TRY NOT TO WRITE SO MANY THINGS ABOUT OTHER APPROACHES

TABLE OF CONTENTS

I. INTRODUCTION

For more information about the task definition (point set/cloud registration, non-rigid or rigid transformation, approaches classification,...) please read !B.0.0 Task definition

As Zhang2020 he confines his discussion to point-to-point matching unless otherwise stated. For completeness, however, he briefly mentioned recent studies related to the six aspects below

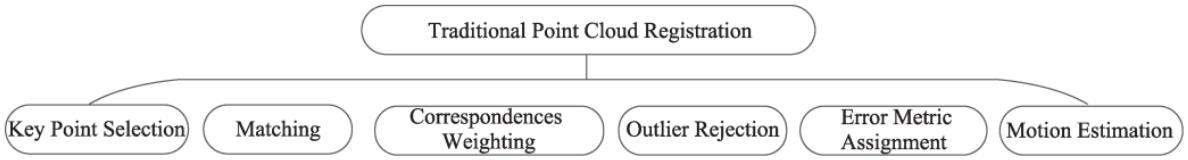
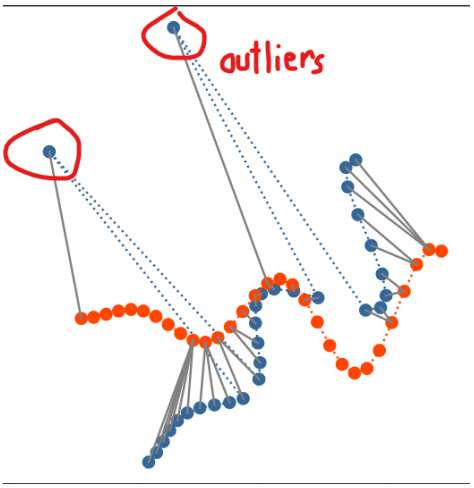


Figure 3 Six phases used in traditional point cloud registration.

For example in the ICP case (ICP implementations), six phases to perform registration

- Selecting Key Points and extracting features of **Key Points**: deciding which spots to use as control points. The more unique the point is, such as corners, the faster and more accurate your transformation matrix will be
- Matching: Compare the similarity of the features of Key Points in the first point set against the Key Points of the second point setto find the right correspondences
- Weighting: Some Key Points are more important and special than others
- Outlier Rejection: remove noise, which can affect on finding reliable correspondence process



- Error Metric Assignment: minimizing the error and finding optimal solutions.

Non-linear Least-squares based ICP

We can alternatively treat every iteration of ICP as a least squares minimization problem. The function we want to minimize is the squared sum of distances between the points of the scans:

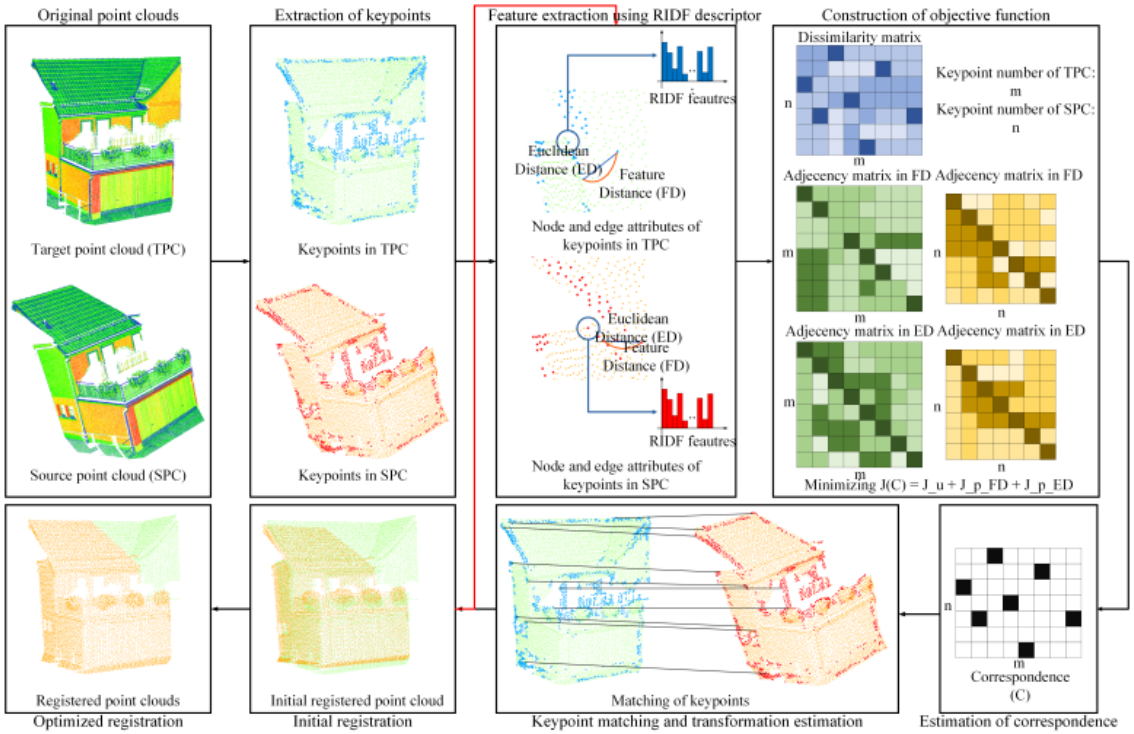
$$E = \sum_i [\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i]^2 \rightarrow \min \tag{8}$$

- Motion Estimation: Compute Rotation matrix, Translation vector (in rigid case) and other parameter/matrix (in non-rigid case) from reliable correspondences

"**Learning-based methods** improve traditional frameworks, which consist of four parts: a feature extractor, matching, outlier rejection, and motion estimation" Zhang2020

That's where we can apply deep learning, I will add some low-level examples that may represent my understanding (I don't know if it's right or wrong/ I will update them later)

- Feature extractor: CNN
- Matching: using KNN to see if the two descriptors are the same
- Outlier rejection: classify whether the point is outliers or inliers with SVM/ ANN
- Motion estimation: reinforcement learning



Example for correspondence-based method (Huang2021)

II. PRELIMINARY KNOWLEDGE

1. Feature extraction

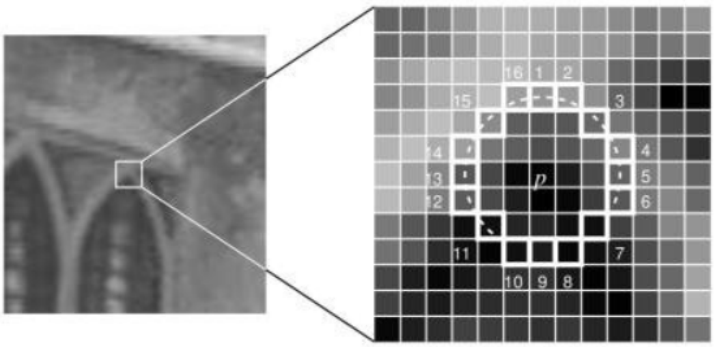
I think feature extraction includes two main parts: keypoint detection and descriptors. Therefore I'll bring out some traditional methods to help you understand what they are, and then talk about how recent works applied Deep-Learning to tackle this task.

Please note that some of the techniques include/implement both keypoint detection and making a descriptor at the same time.

1.1. Keypoint Detection

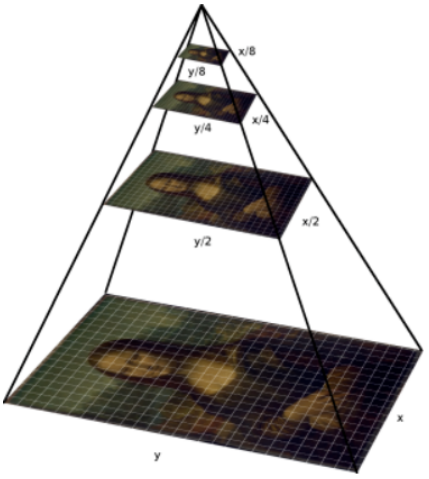
1.1.1. Non-DL methods

FAST (Features from Accelerated and Segments Test) is one of the fastest ways to detect corners of an image by calculating the number of surrounding pixels that lighter or darker than a threshold



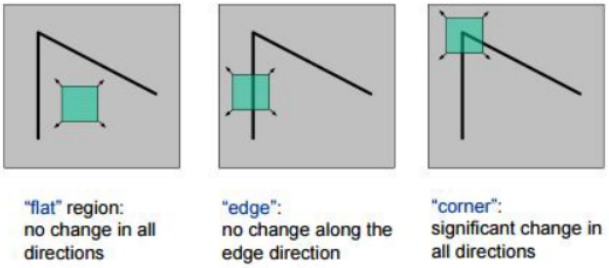
FAST

However FAST is not scale-invariant (scale and rotation-invariance are the most important factors of a detector). BRISK (Binary Robust Invariant Scalable Keypoint) is an upgrade of it. BRISK just simply do FAST on a multiscale image pyramid. FAST, BRISK, ORB



BRISK

Harris Corner Detector "The idea is to consider a small window around each pixel p in an image. We want to identify all such pixel windows that are unique. Uniqueness can be measured by shifting each window by a small amount in a given direction and measuring the amount of change that occurs in the pixel values"



Harris Corner Detector

1.1.2. DL methods

Those are some traditional methods to detect corners - the special points as I previously mentioned. Of course, you can detect lines, curves or so if you think it is suitable. Obviously, with Deep Learning, we can easily detect the corners. You just need to use the object detection models like CNN, YOLO, RCNN... in association with labeled data. As this picture below is from (Extract information from Vietnamese identification cards)

Step 1: Detect 4 corners

The figure below shows how PointNet can learn the key points to successfully solve the registration task. We can see that with PointNet, not just corners are assessed as key points, this means more information is received than from non-DL methods. Simultaneously, it can reduce the points needed to be considered to faster the process

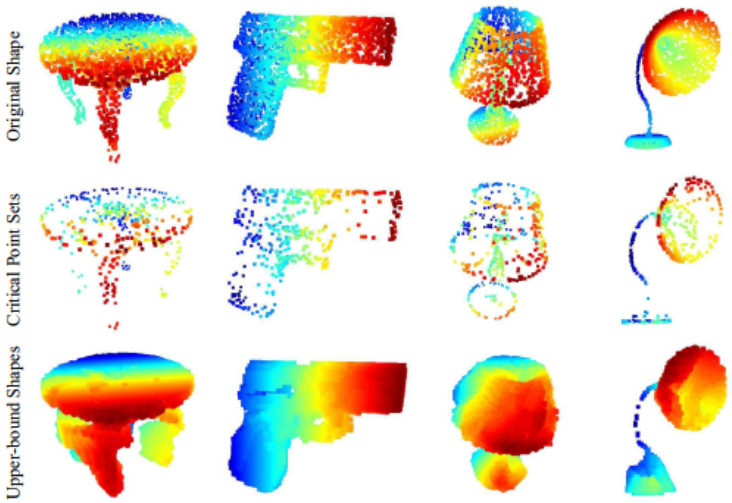


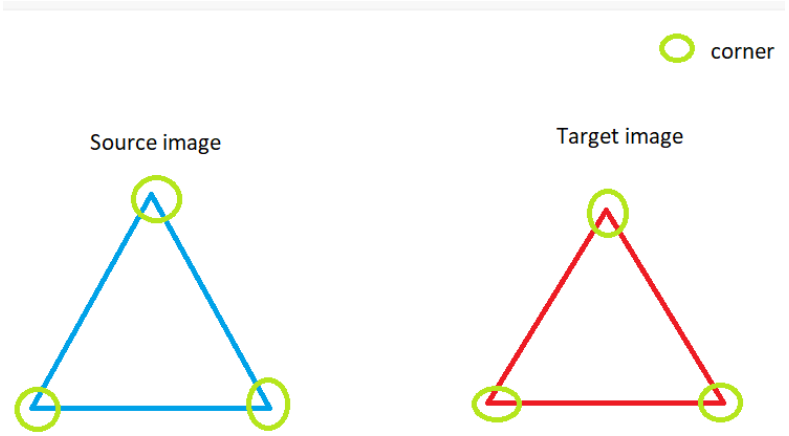
Figure 7. **Critical points and upper bound shape.** While critical points jointly determine the global shape feature for a given shape, any point cloud that falls between the critical points set and the upper bound shape gives exactly the same feature. We color-code all figures to show the depth information.

1.2. Descriptors

"**Descriptiveness** and **rotation-invariance** are important criteria for a competent **feature descriptor**. The descriptiveness highly depends on the descriptor's performance and the

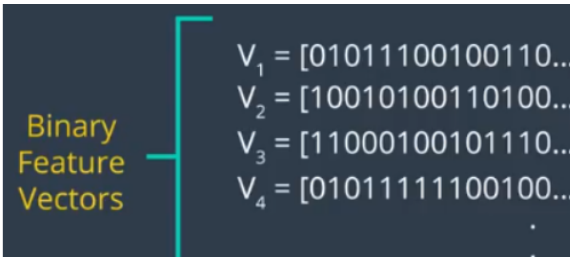
geometry of the described area, which changes from methods to methods. However, the rotation-invariance is a must for all the descriptors" Huang2021

After having a lot of corners/key points in both source and target picture. How do you know which ones are the same? You need a **descriptor** (which describes the corners like angle, position,... - **correspondence**) before comparing them all to know the answer (matching). The more information the descriptor contains, the better that is called **Descriptiveness.** Some papers shows that the descriptors which associates both local and global features achieve higher result



1.2.1. Non-DL methods

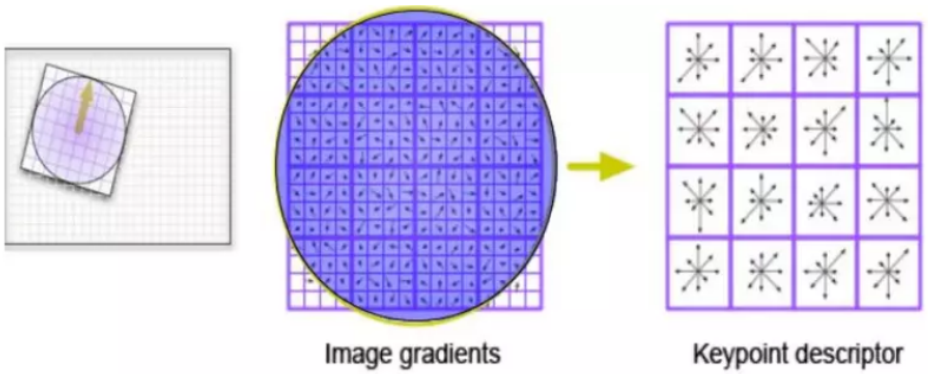
BRIEF (Binary robust independent elementary feature): It just simply represents the surrounding environment (the value of surrounding pixels) of a point with a binary vector. For example, you can describe me as a "tall, white man, living in HCM city" and then binarize the whole text.



BRIEF

SIFT (Scale-invariant feature transform) & HOG (Histogram of Oriented Gradients): they're separate methods but I think they have a lot of things in common. Using the pixel value (0-256) to describe a feature is not a good way (it see each point as an independent point, they're just nonsense numbers), SIFT and HOG calculate the gradients which describe

better the connection/the change between points/ areas. After that, they convert the gradient matrices into vector



SIFT

“In contrast to **SIFT** descriptor, which is a **local image descriptor**, the resulting histograms of oriented gradients (**HOG**) descriptor is a **regional image descriptor**.”

1.2.2. DL methods

In Deep-Learning, we can use a model to **learn the descriptors.** CNN is the first thing that came to my mind, we can describe a point (making vector) directly from an image. It’s like an image-captioning algorithm.

Looking at the image above. Should we describe him with his characteristics as “a man with a violet shirt, black glasses, and a beard” Or more of his personality “he is an angry man”. Or just associate them all? The same with “Using pixel value or gradients to describe a key point?” The model will learn which way is the best to describe them based on what we need.

Some researchers also use Attention Mechanism (like DCP) to increase the performance of a descriptor (which part of an image is more important than others?) or just simply use a ANN with input is the descriptor vector above

When working with point cloud data, there are several design constraints to resolve:

- **Permutation invariant:** The order of points may vary but does not influence the category of the point cloud. The model output should not be influenced by order of input points. PointNet solves this problem with an MLP layer and symmetric functions (SUM or MAX). Graph-based methods automatically ignore the input order of nodes
- **Transformation invariant:** The relative location and direction between the sensor and the objects may change in real-world applications, resulting in point cloud translation and rotation. PointNet and DGCNN both start with a spatial transform layer with the main part being a matrix multiplication. FCGF uses sparse CNN which is so powerful with transformation invariant itself. Furthermore, PointNet++ and LDGCNN are trained with augmented data (rotate, translate, scale data randomly, and add random noise) to achieve this characteristic
- **Extracting local and global features**: Local features are the relationships between a point and its surrounding, whereas global feature is information aggregated from all points. PointNet forgot to use local features which was solved in PointNet++ by a hierarchical network. DGCNN extracts local features after layers and layers of EdgeConv and directly concatenates them to form global features. FCGF extracts local features with kernels at early stages and forms global features in later stages.

2. Matching

2.1. Non-DL methods

We can use some simple techniques to measure the distance/ similarity between two vectors/ matrices like Euclidean distance, Hamming distance, L2 Distance…
Or a dot product/inner product: [vectors - how does the dot product determine similarity?]

$$\cos(\theta) = \frac{u \cdot v}{||u||\,||v||}$$

In some papers we can see that they add a softmax function to get a probability distribution vector with a scalar type [0,1] like in [DCP]

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \operatorname{softmax}(\Phi_{\mathcal{Y}} \Phi_{\boldsymbol{x}_i}^{\top}).$$

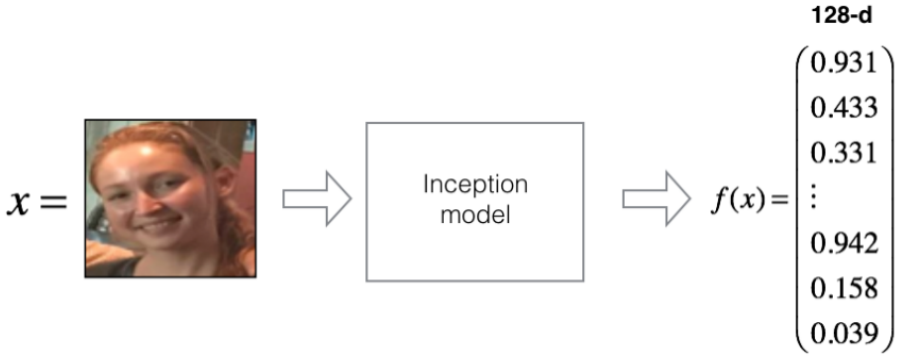
2.2. DL methods

Traditionally, we would choose a standard distance metric (Euclidean, City-Block, Cosine ...) using a priori knowledge of the domain. However, it is often difficult to design metrics that are well-suited to the particular data and task of interest.

More specifically, How can we set up a threshold T to decide whether they are similar or not? Furthermore, if we just use the techniques to just determine either similarity or dissimilarity. The model would overfit to the training data and have a tendency to find everything to be similar. Therefore, they have come up with new ways to automatically learn the metric (Distance metric learning/ Metric learning) based on the association between similarity and dissimilarity

“Moreover, point matching involves computing a similarity score, which is usually computed using the **inner product or L2 distance** between feature vectors. This simple matching method does not take into consideration the **interaction of features of different point pairs**” Li2020 This means these traditional techniques have a problem that a point in source may have multiple possible corresponding points so we need a bigger view to understand the interaction between a point in source to every point in target. The authors first connect the feature extracted from source, target, and distance. After that, they apply a convolution layer to learn the similarity matrix

“The closest work to our proposal is [46]. Instead of constructing an End-to-End registration framework, it focuses on joint learning of key points and descriptors that can **MAXIMIZE local distinctiveness** and **similarity between point-cloud pairs**.” Lu2019
This reminds me of an idea
I read about facial recognition algorithms. Similarly, the task requires us to extract the feature of the face and convert them to a 128d vector. But the point is that we need a model that turns a face image into a vector that faces of the same person will have similar vectors (really similar). Otherwise, faces of different people will have different vectors (really different). And then I came up with **Contrastive, Triplet and Quadruplet Loss** (Updated version)



128d feature vector extracted from a face image

"Triplet loss is a loss function for machine learning algorithms where a reference input (called the anchor) is compared to a matching input (called positive) and a non-matching input (called negative). The distance from the anchor to the positive is minimized, and the distance from the anchor to the negative input is maximized" Wikipedia

Triplet loss (link post in the picture)

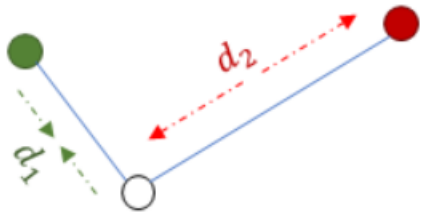
So we can use triplet loss in matching descriptors

Contrastive Loss: **2 inputs** at each time step. Then we compare. These 2 inputs could be similar or dissimilar

Contrastive Loss

Triplet Loss:

- 3 inputs at the same time
- Have Anchor (2 positives and 1 negative)



Triplet Loss

Quadruplet Loss:

- 4 inputs at each time step
- Have an anchor to compare with 2 other objects (1 pos and 1 neg)
- Have another negative object which is dissimilar to every other of the three objects

Reference: [How to choose your loss when designing a Siamese Neural Network ? Contrastive, Triplet or Quadruplet ?]

In [Choy2019], they created new losses (Hardest-contrastive and Hardest Triplet) which can be understood with the picture below. I'll write about it later in other part

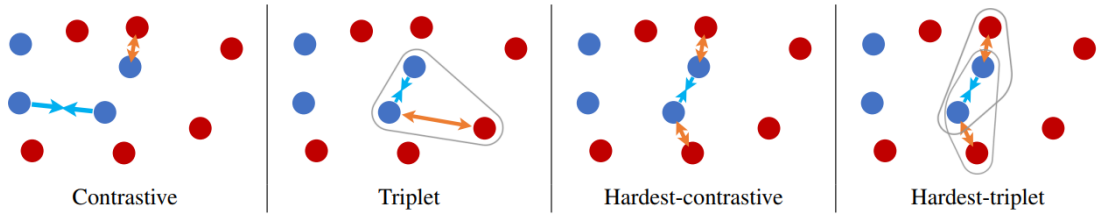


Figure 3: Sampling and negative-mining strategy for each method. Traditional contrastive and triplet losses use random sampling. Our hardest-contrastive and hardest-triplet losses use the hardest negatives.

3. Outlier Rejection

Outliers is usually noise, for point set registration, the outlier can have bad effects on the correspondence finding. Therefore outlier rejection is an important part. As with some papers, this part is quite similar to Robust fitting. Robust fitting is a special model fitting technique designed to handle the case where input data is contaminated by outliers.

For example, the Least Square (Non-robust) calculate the regression tasks by calculate the square distance of all the data points to the prediction line.

$$S = \sum_{i=1}^{n} r_i^2.$$

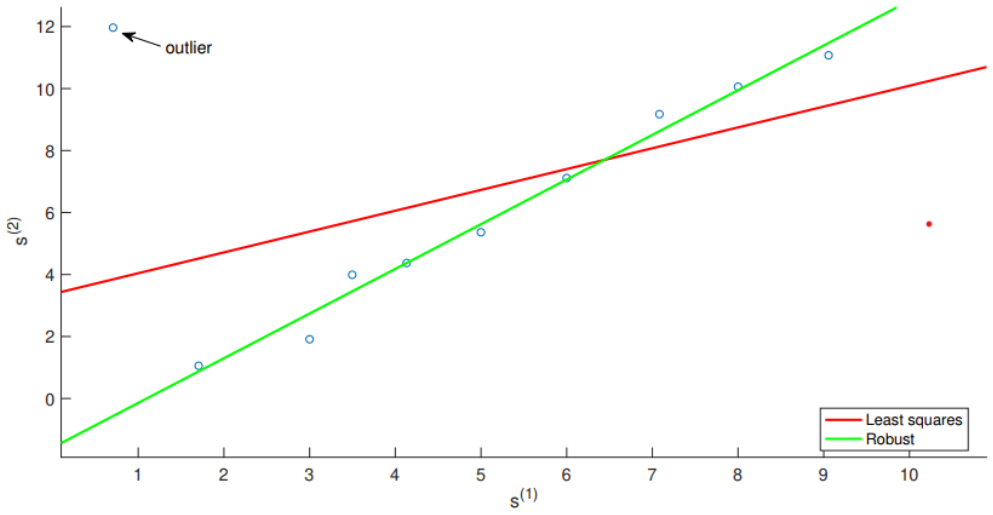


FIGURE 1.1: Line fitting example. The outlier (the upper left point) does not follow the linear distribution of other points. The Least Squares solution is far away from the desired location due to the existence of outliers.

As can be seen from the figure above, the correct line must be the green line. However, the outlier in the upper left of the image makes the final prediction (which is calculated by Least Square) is the red line. We can say that the model based on Least Square is Non-robust because it is highly sensitive to outliers.

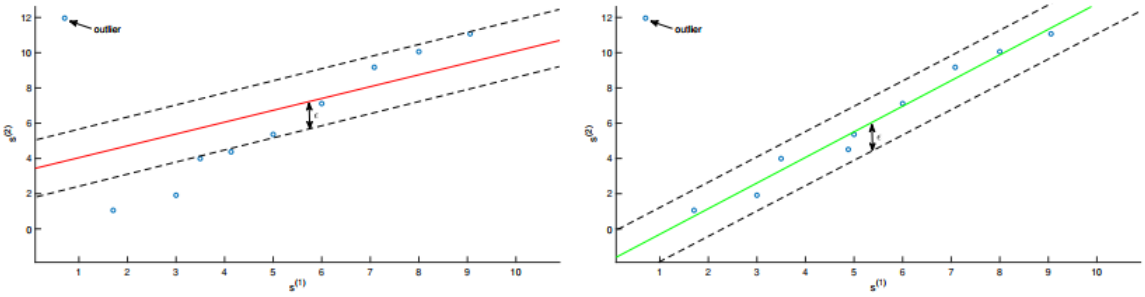
However, there are situations when an outlier cannot be considered noise (which contains little/no information, its drawbacks are more than the benefits), but rather a "black swan event". This concept could provide us with some future puzzle pieces to solve.

It has a lot of things to discuss about Robust fitting like the Robust Criteria (Consensus Maximization, M-estimator, Least Median Square) or how to optimize algorithms for robust fitting (Random sampling based, Gradient based deterministic, Globally Optimal). However, I won't focus on them all, unless there is something related in the future. You can read them in Consensus Maximization Theoretical and analysis.pdf

3.1. Non-DL methods

To solve the problem as in the figure above. One of the most powerful Random sampling based algorithms is RANSAC (Random Sample Consensus). This technique repeatedly draws a line from two random points from the dataset. It uses a threshold to distinguish Inliers and outliers. Then it counts the numbers of inliers and the final result is the line that has the most inliers (Consensus Maximization). Although simple and efficient, the result of

this method depends on the initial pair of points. Many RANSAC variants were proposed later on to further improve the performance



(a) Least Squares solution (Consensus = 6). (b) Desired solution (Consensus = 9).

Or we can preprocess the data by using threshold or some data characteristics (Point locations) to efficiently remove the outliers. However, this techniques depends on the task requirements and suitable for small data set (less than a thousand points, I guess)

Or with the Least Square, we can weight the data based on the distance, far point has lower weight, this can reduce the influence of outliers

3.2. DL methods

Based on what I've read till now, I realized that there are very few methods that have a separate outlier rejection part. The feature extraction/key points extraction is usually responsible for this part. For example, In IDAM, they used a two-point elimination stage to remove unreliable points/correspondences. DCP or some paper used an attention layer/weighting layer to focus more on key points extraction. In my perspective, these methods significantly reduce the influence of outliers at the same time.

I think the reason why not so many people try to add a separate part for outlier rejection is that the task is simply a classification algorithm to classify if a point is an inlier or outlier. Therefore, we can easily use a neural network to solve it, and this NN can be combined with the key point extraction part. In general, this task is quite the same with segmentation. We classify a point into 2 classes (Outlier vs Inlier) instead of classify them into a lot more classes

I'll update this part after reading Supervised and unsupervised methods as Zhang2020. I don't know if the problem is easy as I wrote above

III. RELATED WORK

This part is for related techniques/ terms…

1. Terms

1.1. Invariant and Equivariant

Equivariant : means that a translation of input features results in an equivalent translation of outputs. So if your pattern 0,3,2,0,0 on the input results in 0,1,0,0 in the output, then the pattern 0,0,3,2,0 might lead to 0,0,1,0

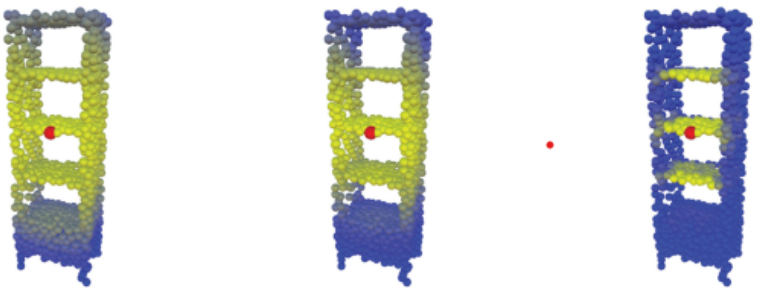
Invariant: means that a translation of input features doesn't change the outputs at all. So if your pattern 0,3,2,0,0 on the input results in 0,1,0 in the output, then the pattern 0,0,3,2,0 would also lead to 0,1,0

“For feature maps in convolutional networks to be useful, they typically need both properties in some balance. The equivariance allows the network to generalise edge, texture, shape detection in different locations. The invariance allows precise location of the detected features to matter less. “What is the difference between "equivariant to translation" and "invariant to translation"

2. Techniques

2.1. PCA and T-SNE

DCP and T-SNE are vital techniques in deep learning. The main idea of these techniques is to reduce the dimension of data for us to represent the relationship between data classes (in which I’ll focus on in this part)

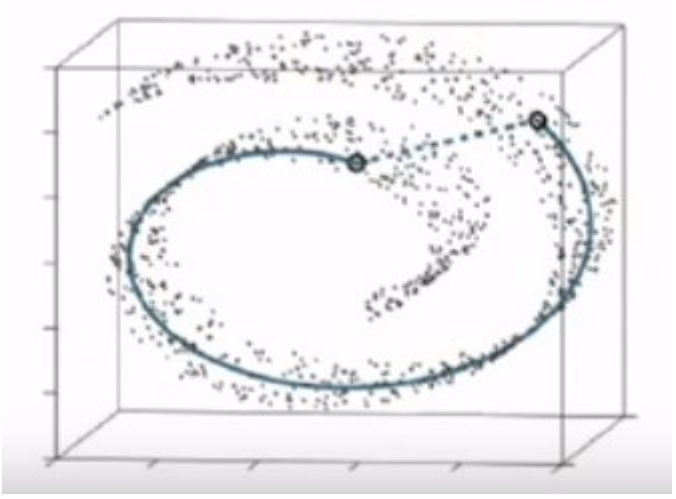


The relationship of points in DGCNN

For example, each point in point cloud above has a different feature with a 32-dim vector. But we cannot represent the similar points in such a high dimensional space like that. This is where PCA and T-NSE come in. They assist us to represent the similar points in 2D space like the image above.

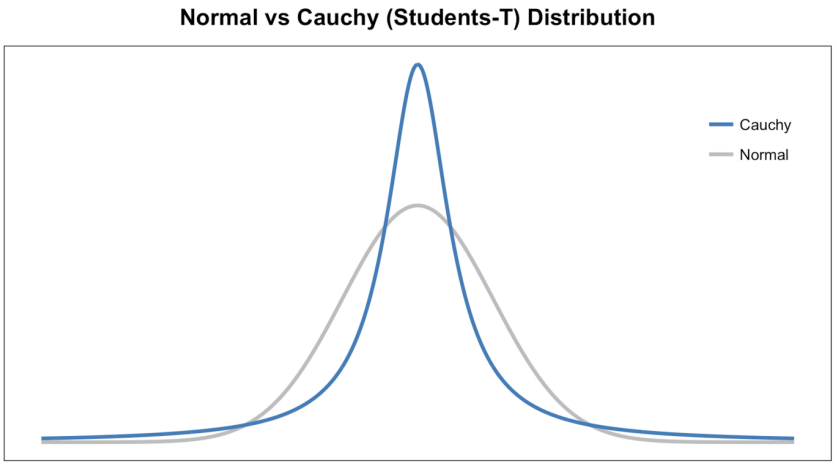
The main idea of PCA is: to reduce from D dimension to K < D is to keep K - most important features. In other words, K features bring the most information and have the greatest effect on the data and the output. Obviously, This is one of the simple and effective methods so that It is used in a bunch of today's applications even though it was first developed in 1933

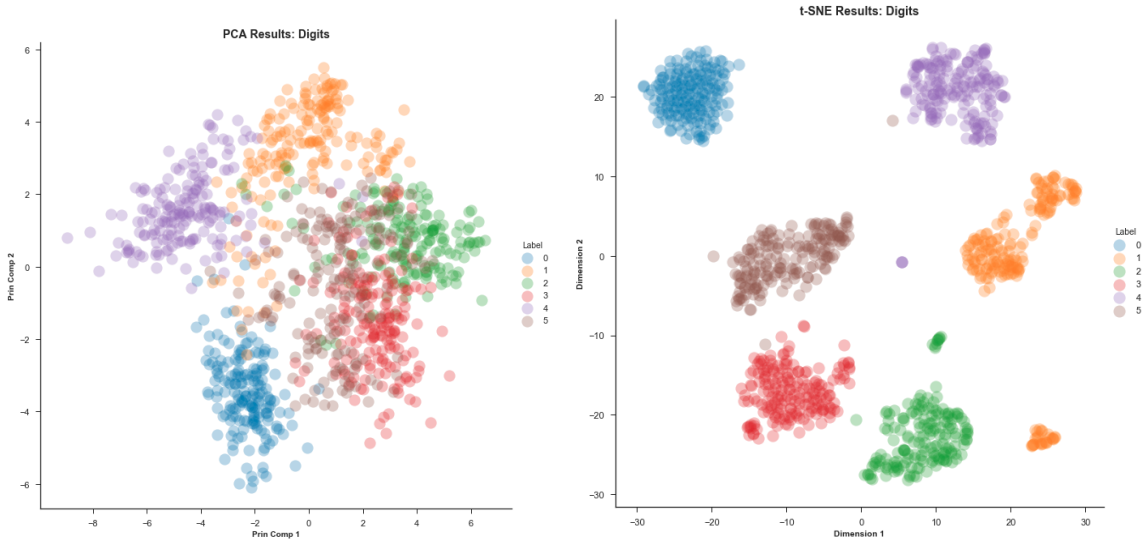
However, PCA is a linear dimension reduction technique that seeks to **maximise variance** and **preserves large pairwise distances** which leads to a poor performance with manifold structure like the image below. (A manifold is a mathematical object that can be curved but looks flat locally)



Look at the two circled points. The dotted line between the two circled spots is what PCA is interested in, whereas the solid line represents the structure that we could be interested in. The dotted-line path is significantly shorter than the journey along the solid-line, which appears to be the correct data structure here.
PCA will fail to find this non-linear (solid-line) path , but it seems we would find the path illustrated by the solid line if we just focus on going along only **the closest points** (preserving only **small pairwise distances** or **local similarities**).

So how does T-SNE work? It's more of a Probability-based method. In simple words. Let's say In high dimension, point x has a P probability to be similar to point x' so in lower dimension, it must have the Q probability similar to point x' too. (Q is equivalent to P). P is calculated by Gaussian distribution, Q is calculated by Cauchy (Student-T) distribution





Reference:

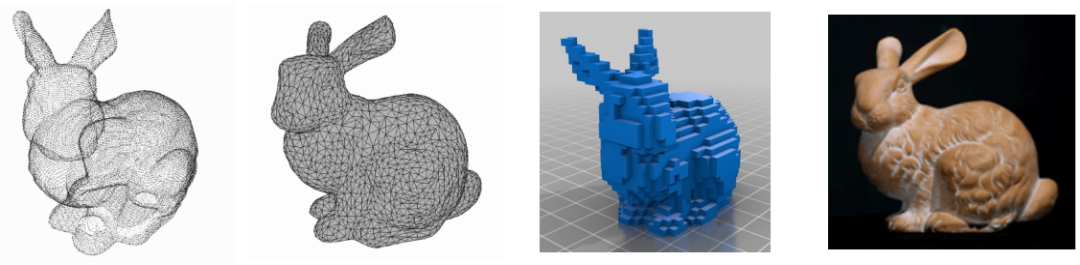
- An Introduction to t-SNE with Python Example | by Andre Violante | Towards Data Science
- (5) What is the difference between PCA and T-SNE? - Quora
- (20) Visualizing Data Using t-SNE - YouTube

IV. DEEP-LEARNING BASED POINT CLOUD REGISTRATION

1. PointNet, PointNet++, DeepVCP

1.1. PointNet

PointNet is used for extracting the features of the points which helps solve some problems like classification, segmentation, registration… directly from **points**, without any further transforming to other representations like mesh, volumetric, RGB(D)



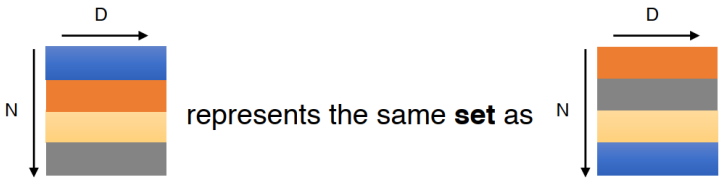
3D representations

The three special features of PointNet are:

- Unordered pointset as input: The input order of points/components in a point cloud/function won't affect the result. That's why they use Max or Sum function (symmetric function)
- Interaction among points (Descriptiveness): capture both local and global features. The features of each point also come from their neighbor
- Invariant under transformation like rotation, scale

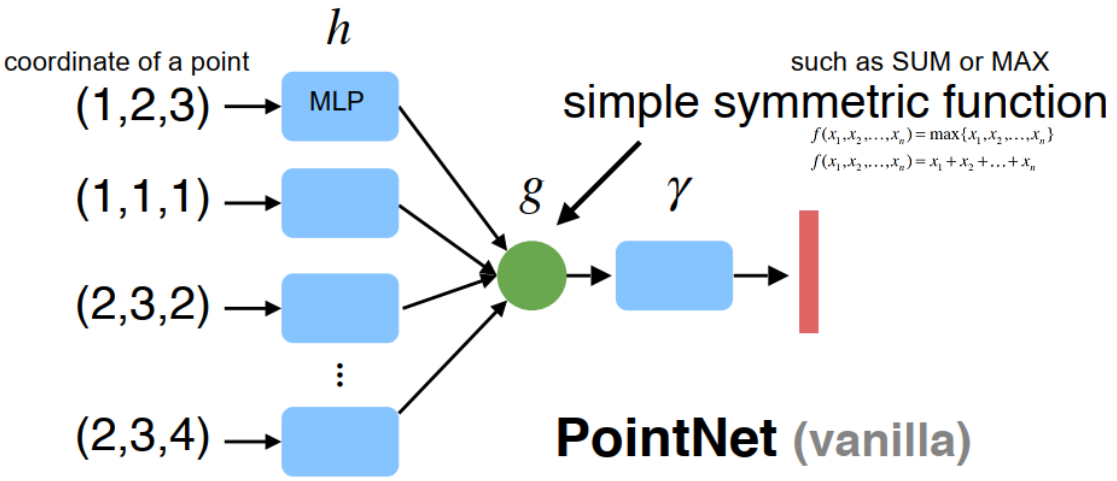
1.1.1. Unordered pointset as input

Point cloud: N **orderless** points, each represented by a D dim vector

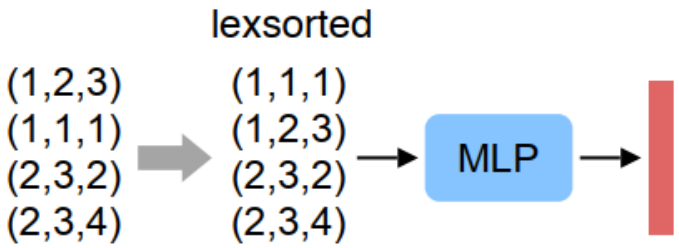


N: the number of points

D: Dim vector. Usually D = 3 = (x,y,z) in Euclidian space. DeepVCP uses the point intensity in addition so D = 4. Similarly, D = 2 in 2D space



How about sorting points before feeding them into the model. The question is should we sort them with x, y, z coordinates first or with intensity? “Unfortunately, there is no canonical order in high dim space”

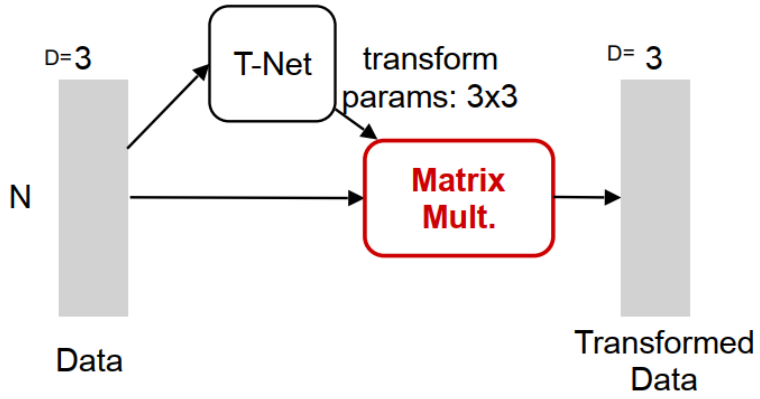


An example of sorting

How about using RNNs? The authors tried to train RNN with permutation augmentation. But as previously mentioned. There is no canonical order in high dim space and not to mention a vast number of points (millions of them) can create a …. number of augmentations that makes it impossible to implement.

After all, human beings don’t classify a 3D object by seeing and processing the object by scanning it from left to right/ up to down or with any specific order

1.1.2. Invariant under transformation



Input Alignment by Transformer Network

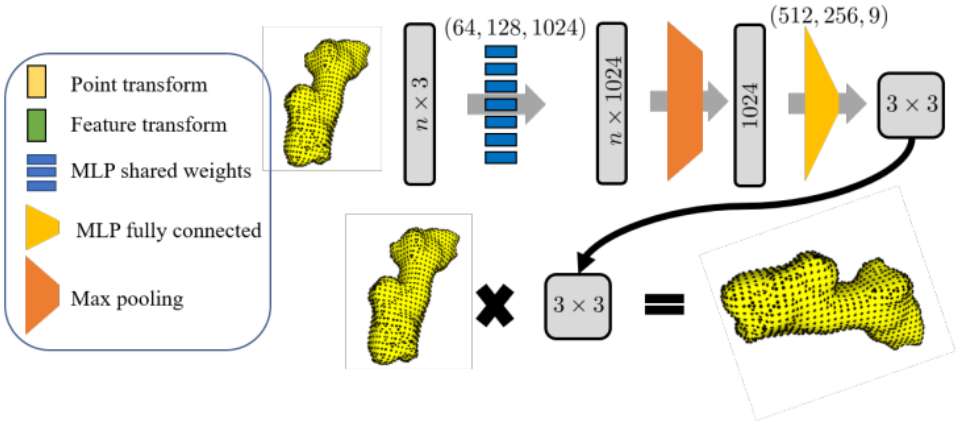
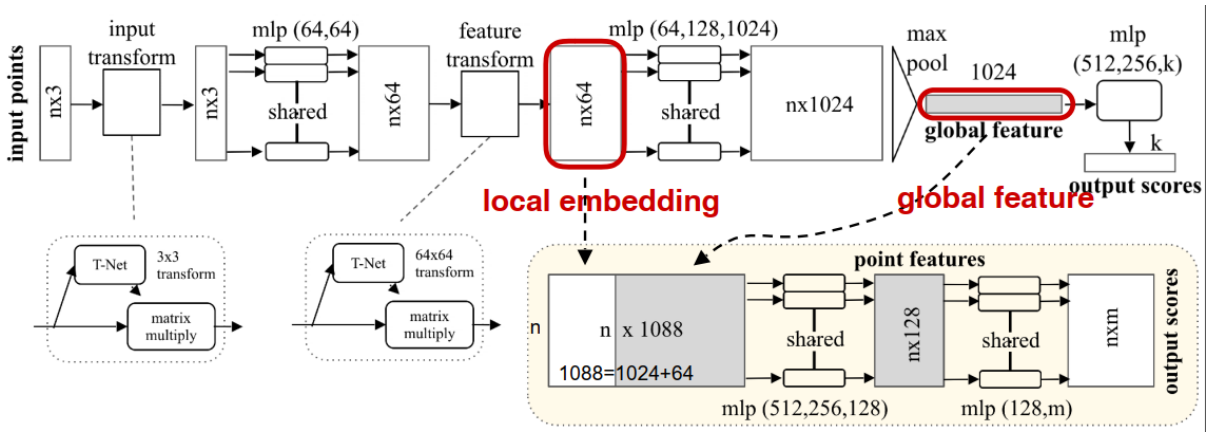


Fig. 2. Transformation network (T-Net) for predicting a transformation matrix to map a point cloud to canonical space before processing. A similar network is used to transform the features; the only difference is that the output corresponds to a 64 × 64 matrix.

PointNet Architecture



The network has three key modules: Max pooling as a symmetric function to aggregate information from all the points, a local and global combination for segmentation problems, two joint alignment networks that align both input points and point features

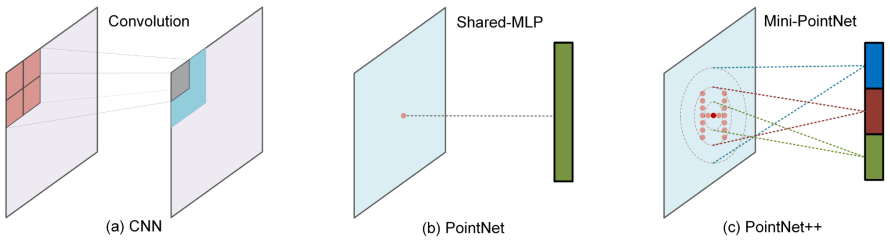
The output global feature vector can be fed into a classifier (ANN = MLP, SVM, decision tree, KNN…) to address the classification issue, or it can be utilised as a descriptor vector for registration tasks. We can combine the local embedding of each point with the global feature to have a 1088-dim vector, which means **each point now have both local and global feature** compared to just a 3-dim vector (coordinate) at the beginning **(increasing descriptiveness**)

In fact, segmentation is quite the same as classification. Instead of classifying the whole object, now we just classify each point. In the case of segmentation, the author used the combined vector as the input and receive the n x m matrix (n: the number of points, m is the number of classes)

1.2. PointNet++

PointNet has some disadvantages:

- Point sets are usually sampled with **varying densities**. PointNet just captures the coordinates of points but not the distance between them
- "A CNN takes data defined on regular grids as the input and is able to progressively capture features at increasingly larger scales along a multi-resolution hierarchy. At lower levels neurons have smaller receptive fields whereas at higher levels they have larger receptive fields. The ability to abstract local patterns along the **hierarchy** allows better generalizability to unseen cases" Qi2017.
- "PointNet consumes the point cloud directly without quantization and aggregates the information at the last stage of the network, so the accurate data locations are intact but the computation cost grows linearly with the number of points. " Xu202. This is not solved in PointNet++



CNN, PointNet, PointNet++

"We call our **hierarchical network** with **density adaptive PointNet** layers as PointNet++." That's why they have **two +**. PointNet++ is an updated version to solve those problems by:

1. Partitioning the set of points into overlapping local regions by the distance metric
2. Similar to CNNs, they extract local features capturing fine geometric structures from small neighbourhoods
3. Local features are further grouped into larger units and processed to produce higher-level features.
4. Repeating the process until they obtain the features of the whole point set

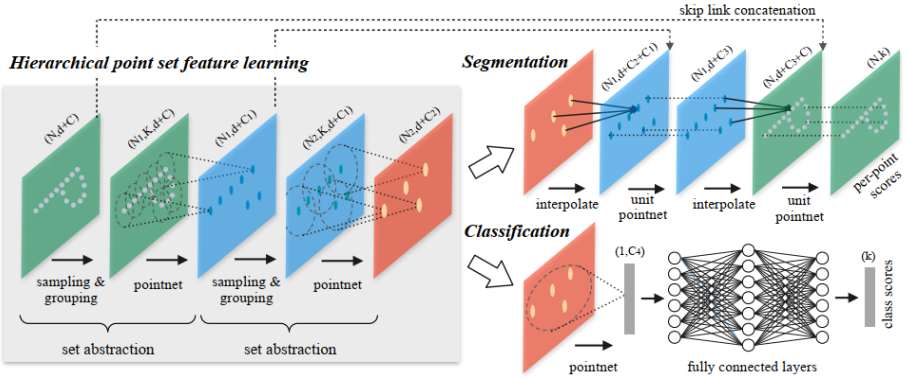
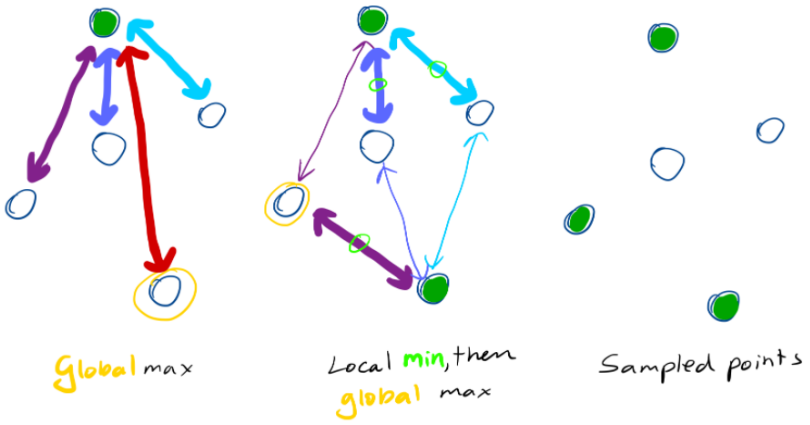


Figure 2: Illustration of our hierarchical feature learning architecture and its application for set segmentation and classification using points in 2D Euclidean space as an example. Single scale point grouping is visualized here. For details on density adaptive grouping, see Fig. 3

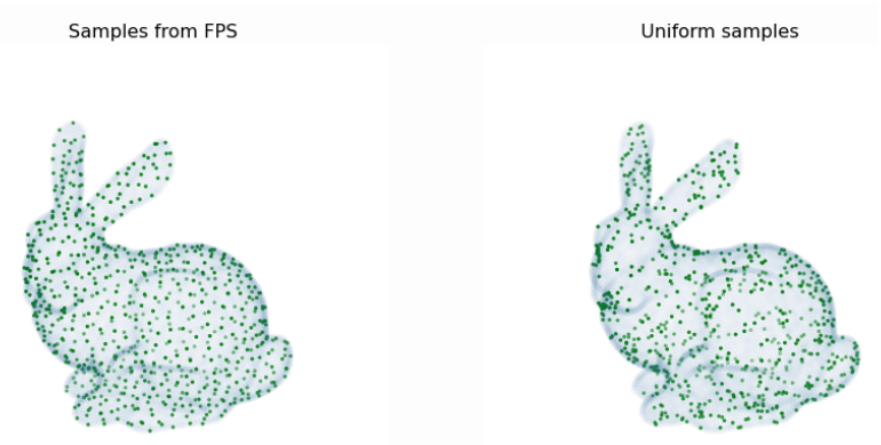
1.2.1. Hierarchical Network

The hierarchical structure is composed of several set abstraction levels. “A set abstraction level takes an N × (d + C) matrix as input that is from N points with d-dim coordinates and C-dim point feature. It outputs an N’ × (d + C’) matrix of N’ subsampled points with d-dim coordinates and new C’-dim feature vectors summarizing local context” The set abstraction layer includes three main layers: Sampling, Grouping, and PointNet

Sampling layers: select a set of points from input points to define as the centroids of local regions. They use iterative farthest point sampling (FPS)



Farthest point sampling



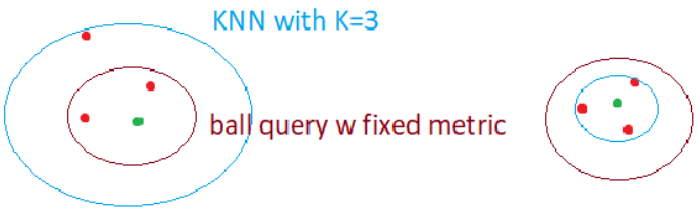
Comparison between FSP and Uniform samples

We can see that furthest point sampling provides much better coverage of the point cloud compared to uniform sampling and the point density doesn't vary much

Grouping layers: finding "neighboring" points around the centroids. The number and locations of neighbors can create different features. "The input to this layer is a point set of size N × (d + C) and the coordinates of a set of centroids of size N' × d. The output is groups of point sets of size N' × K × (d + C), where each group corresponds to a local region and K is the number of points in the neighborhood of centroid points."

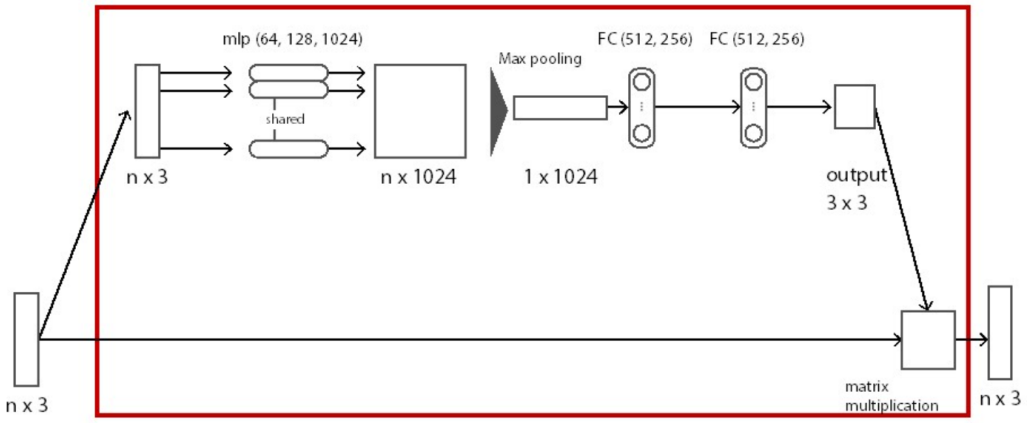
There are two methods to find the neighboring points:

1. Ball query: finds all the points within a radius to the query point, but an upper limit of points (K) is set.
2. K nearest neighbor (KNN): finds a fixed number of neighboring points with respect to the distance metrics.



Comparison between KNN and Ball Query

PointNet layers: They use a mini-pointnet to encode the local pattern into a feature vector. In this layer, the input are N' local regions of points with data size N'×K×(d+C). Output data size is N' × (d + C'). C' is a more informative feature than C with the same dimension.



Input transform.

Mini-pointnet: "Φ(X) = **MLP**$\left(\mathbf{Max}\left(L(X)\right)\right)$ **and** Φ(Y)**"** Zhang2020

1.2.2. Robust Feature Learning under Non-Uniform Sampling Density (density adaptive PointNet)

As discussed earlier, the model trained with a low-density point set can't properly work with a high-density set and vice versa. Furthermore, each abstraction level contains grouping and feature extraction of just a single scale.

The author proposes **density adaptive PointNet layers** that learn to combine features from regions of different scales when the input sampling density changes by extracting multiple scales of local patterns and combining them intelligently according to local point densities

There are two types of density adaptive layers: Multi-scale grouping (MSG) and Multi-resolution grouping (MRG)

- Multi-scale grouping (MSG): It's like BRISK using FAST on a multi-scale pyramid to be scale-invariant. This method is simple, effective but computationally expensive
- Multi-resolution grouping (MRG): "One vector (left in figure) is obtained by summarizing the features at each subregion from the lower level Li-1 using the set abstraction level. The other vector (right) is the feature that is obtained by directly processing all raw points in the local region using a single PointNet.". This method is much faster because it just conducts PointNet for once rather than doing it several times on several scales and "avoids the feature extraction in large scale neighborhoods at lowest levels"

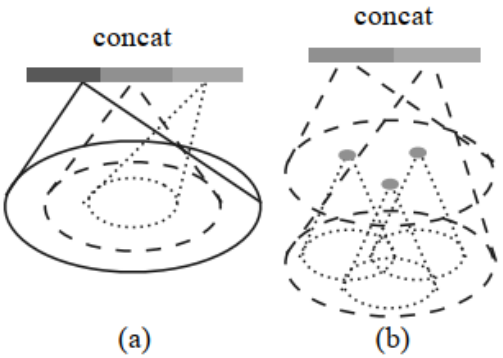


Figure 3: (a) Multi-scale grouping (MSG); (b) Multi-resolution grouping (MRG).

1.2.3. Disadvantages

- The data structuring cost (FPS, K-NN) up to 88% which makes it slow and not computational efficient
- PointNet++ is made up of fading local features that are gradually reduced from layer to layer by grouping and sampling.
- Grouping points in Euclidean space, DGCNN groups points in both Euclidean and feature space and achieves better results. Distant points can have similar features in DGCNN

1.3. DeepVCP

DeepVCP (Deep Virtual Corresponding Points) is an end-to-end learning-based method to accurately align two different point sets with 4 main steps:

1. Extract the semantic features of each point from 2 point clouds with PointNet++. Make use of the weight pointing layer to further define key points
2. Create a **virtual corresponding point** based on the mini-PointNet descriptor
3. Calculate Loss between the source keypoint and the generated corresponding point
4. Using SVD to find Rotation and Translation matrices with reliable correspondences

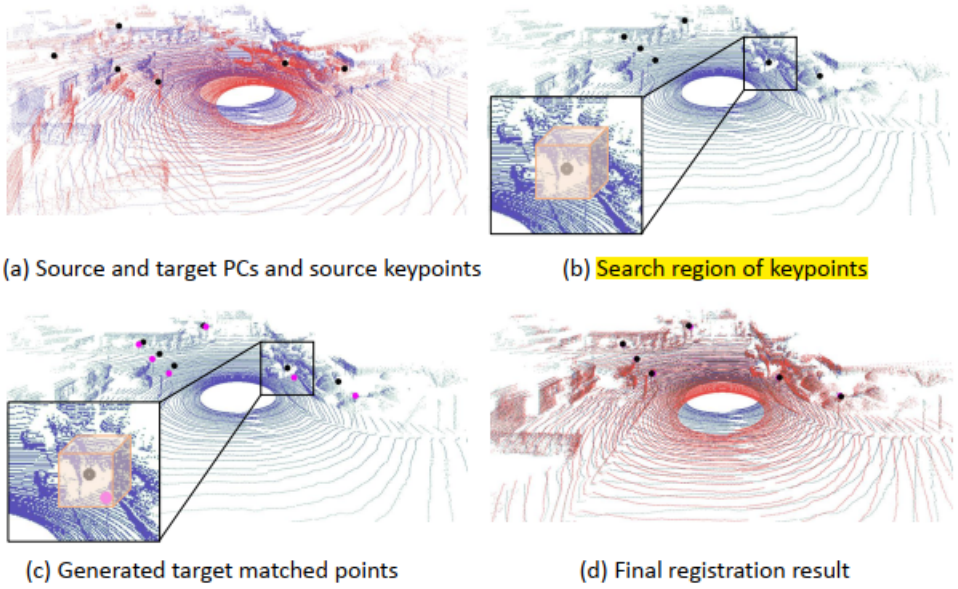


Figure 1. The illustration of the major steps of our proposed end-to-end point cloud registration method: (a) The source (red) and target (blue) point clouds and the keypoints (black) detected by the point weighting layer. (b) A search region is generated for each keypoint and represented by grid voxels. (c) The matched points (magenta) generated by the corresponding point generation layer. (d) The final registration result computed by performing SVD given the matched keypoint pairs.

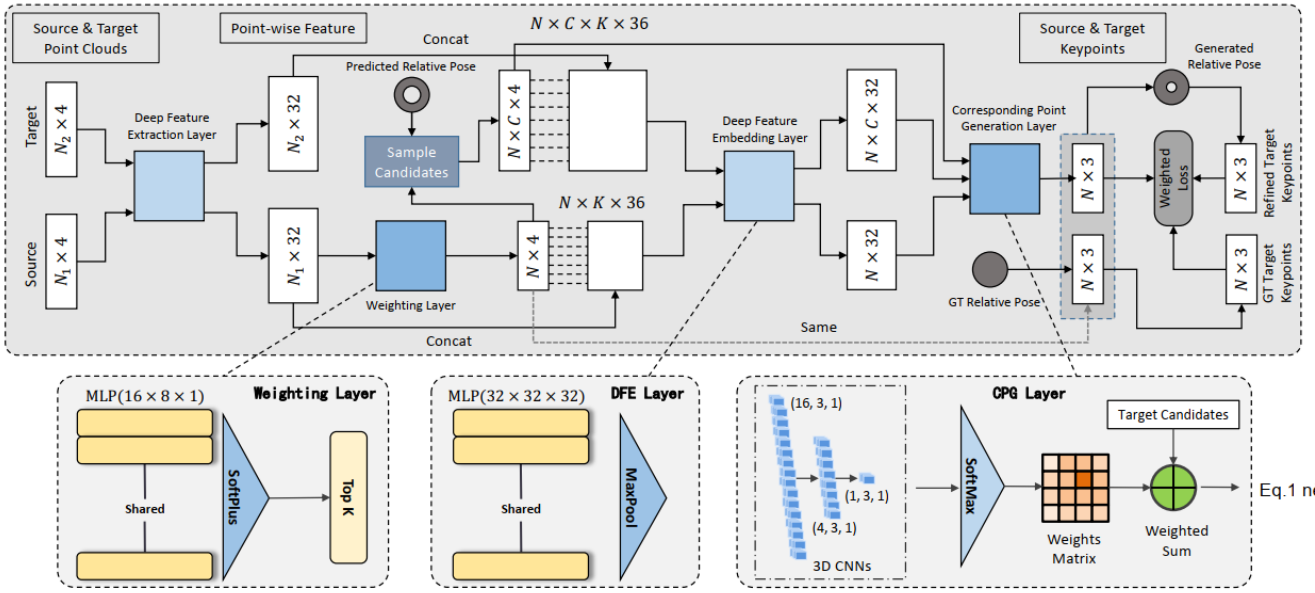
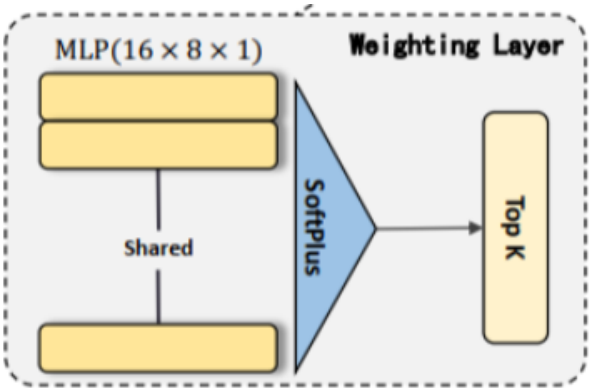


Figure 2. The architecture of the proposed end-to-end learning network for 3D point cloud registration, DeepVCP. The source and target point clouds are fed into the deep feature extraction layer, then N keypoints are extracted from the source point cloud by the weighting layer. $N \times C$ candidate corresponding points are selected from the target point cloud, followed by a deep feature embedding operation. The corresponding keypoints in the target point cloud are generated by the corresponding points generation layer. Finally, we propose to use the combination of two losses those encode both the global geometric constraints and local similarities.

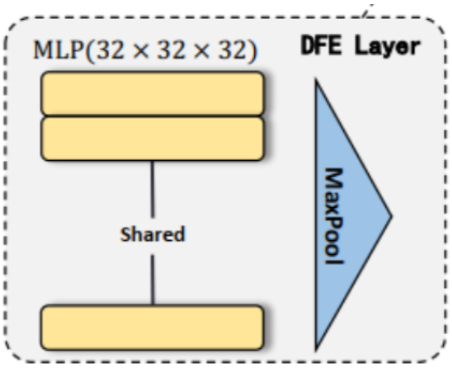
1.3.1. Feature extraction

Deep feature extraction: Apply PointNet++ for both source and target point sets. Input is Nx4 tensor (4 is x,y,z coordinate, and one more parameter but the author doesn't write anything about that but I guess it is point intensity). The Output is Nx32

Point Weighting: "points with invariant and distinct features on static objects should be assigned higher weights" remember that source and target point clouds are much different because of dynamic objects. Imagine LiDAR receiving 2 point sets at 2 distinct moments t1 and t2, so we just need to align the common parts.
Nx32 local features are fed into an MLP layer to a softplus activation to a top k operators



Deep Feature Embedding: Associate the properties of K neighboring points to generate a more informative descriptor for the key points. The authors use mini-PointNet. The input NxKx36 (N: number of key points, K: number of neighbors, 36: coordinates, point intensity, and 32-dim feature vector). The output is Nx32



To sum up, the feature extraction part of DeepVCP includes 3 main parts:

- Create preliminary information for every point with PointNet++
- Filter out the key points with simple MLP layers

- Combining the neighbouring points to create a more informative descriptor for key points

1.3.2. Matching and Motion estimation

Corresponding point generation: As in common perspective, we will create descriptors for key points of set 1, repeat that with set 2, and then compare each point in set 1 with every point in set 2 to create a map. This will cost a lot of time and computational expense and not to mention “there are actually no exact corresponding points in the target point cloud to the source due to its sparsity nature”. That’s why they propose a virtual point generation from the extracted features and the similarity represented by them

I think Zhang2020 writes this part in a simpler way than I do, I will have my comment in red to further explain what it means

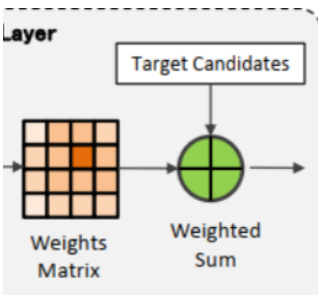
“DeepVCP focuses on a subset constituted by key points denoted as S ∈ X, NS ≪ NX, where NS and NX are the point set sizes. The process is given briefly as follows. First, S is transformed using the input initial parameters R0 and t0 (The author doesn’t say anything about how to have Ro, To. I guess they are from the center of mass like ICP), and generates the corresponding point x'i for xi. The neighbouring space of x'i is divided into(2r/s + 1, 2r/s + 1, 2r/s + 1) 3D grid voxels, where r is the searching radius and s is the voxel size, both of which are predefined.

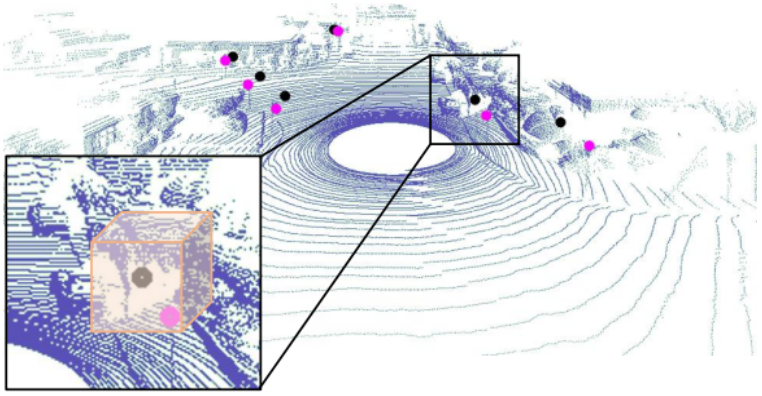
Denote the centers of the 3D voxels as y'j, j = 1,...,C, which are considered candidate corresponding points (x’i is the center of big voxels which is divided into C smaller voxels with y’j as the center). Next, all candidates are fed into the feature extractor. A three-layer 3D CNN is applied to learn the similarity (Matching) between the features of the source point and the candidate points. (As I mentioned before, instead of searching for all of the point cloud space, DeepVCP limits its searching area). More importantly, it can smooth (regularize) the matching volume and suppress the matching noise. The softmax operation is applied to convert the matching costs into the probabilities.

Finally, the target corresponding point yi is calculated through a weighted-sum operation as follows:

$$y_i = \frac{1}{\sum_{j=1}^{C} w_j} \sum_{j=1}^{C} w_j \cdot y'_j,$$

----->





(c) Generated target matched points

Black point is x'i in the target cloud, pink dots are predicted yi

This method uses two kinds of Losses: They use yi for local Loss, and R, T for global Loss. the model will learn to update itself from these Losses to better "Guess" the transformation

local

$$Loss_1 = \frac{1}{N}\sum_{i=1}^{N}|\bar{y}_i - y_i|.$$

global cũng là yi

$$Loss_2 = \frac{1}{N}\sum_{i=1}^{N}|\bar{y}_i - (Rx_i + T)|.$$

$$Loss = \alpha Loss_1 + (1-\alpha)Loss_2,$$

Since we already have the reliable correspondences between black and pink dots, we can easily calculate the transformation R, T by SVD (The same as ICP)

2. FCGF, Multi-view 3D point cloud reg

2.1. FCGF (Fully Convolutional Geometric Features)

As previously mentioned in feature extraction. CNN can be used as a descriptor by using convolution kernels to extract features. To have a good descriptor with CNN, we usually need to be concerned about 2 things: the **model structure** to extract features and **Loss function** to update the parameters (since we don't know the output vector is good enough,

please review Triplet loss in Matching) for the descriptor. FCGF is like a updated version with 2 prominent features:

- Sparse tensor for sparse data (faster the process and efficiently using the computer resources)
- New Losses (Hardest - contrastive and Hardest triplet): greater performance in generating the reliable correspondences

In this part, I just want to discuss the new things of FCGF, not the model structure.

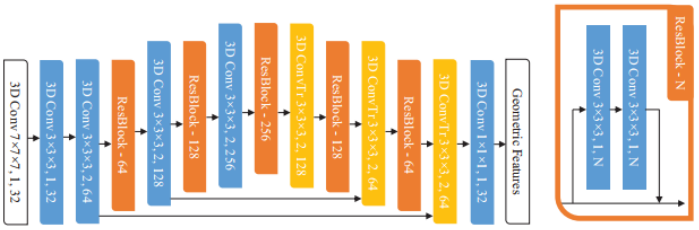


Figure 2: We use a ResUNet architecture. The white blocks indicate input and output layers. Each block is characterized by three parameters: kernel size, stride, and channel dimensionality. All convolutions except the last layer are accompanied by batch normalization followed by a nonlinearity (ReLU).

2.1.1. Sparse convolution

Disadvantages of CNN (Dense convolution):

- Submanifold Dilation

Figure 1: Example of "submanifold" dilation. **Left:** Original curve. **Middle:** Result of applying a regular 3×3 convolution with weights $1/9$. **Right:** Result of applying the same convolution again. The example shows that regular convolutions substantially reduce the sparsity of the feature maps.

- Not efficiently using the computer resources. The parameters we need to calculate squarely growth in 2D, and Triple in 3D space, while most of the point in our task - point set reg is zero. (17) Intro to Sparse Tensors and Spatially Sparse Neural

Networks - YouTube The author said that we are wasting about 78% - 98% by using **dense convolution** as the image resolution gets higher

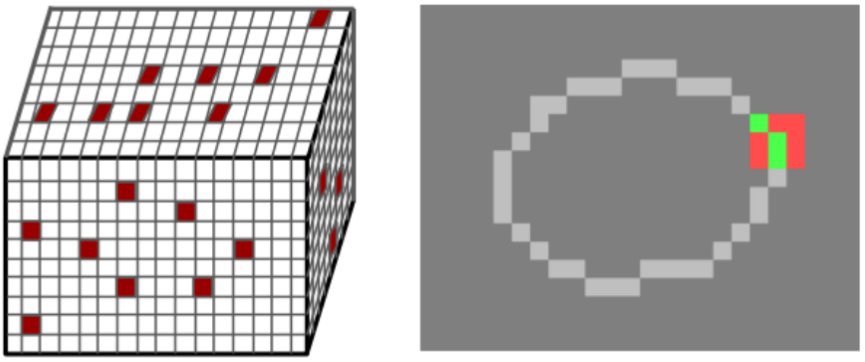
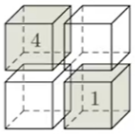


Fig.1 **(Left)**: a 3D grid with sparse voxels by Xuesong Li [3]. **(Right)**: an image with sparse pixels, where dark gray pixels are all zeros, and light gray pixels stand for the non-zero data points. The red square stands for the convolution kernel by Graham, Benjamin [1].

Reference: Sparse Sabmanifold Convolutions. The convolution algorithm for real-time… | by Vipin Sharma | Geek Culture | Medium (You should leave the Harsh table and Rulebook cause it's hard to understand :)))

So how can we **solve those problems**? First, we store the data in a different way. Instead of storing the matrix with zero values, we can just store the coordinates of non-zero value and its value

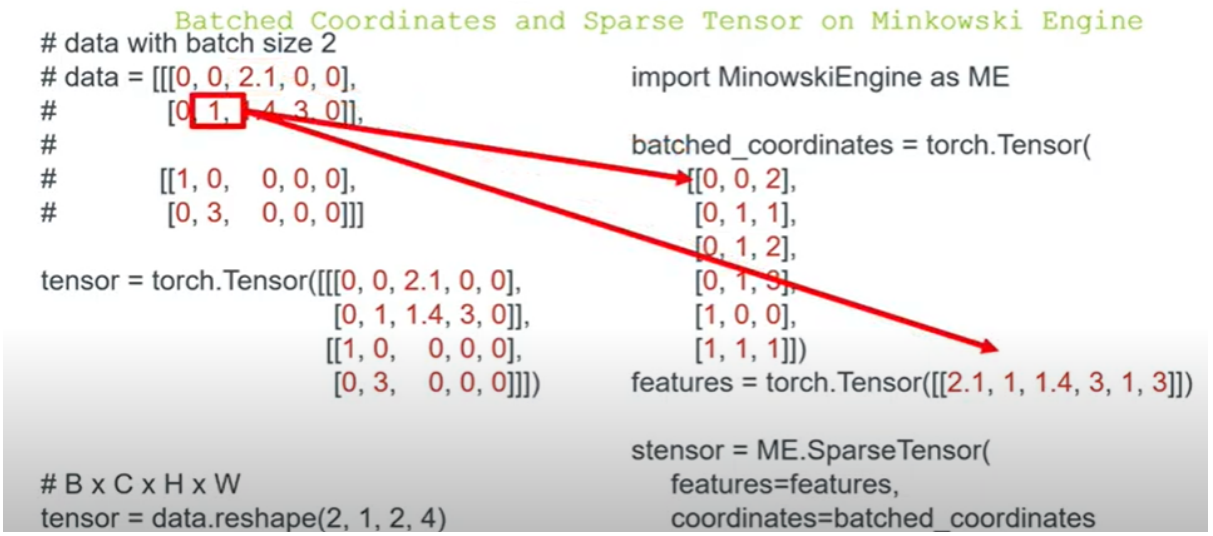


$$\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

{Coordinate: Value}

(0, 0) : 4

(1, 1) : 1

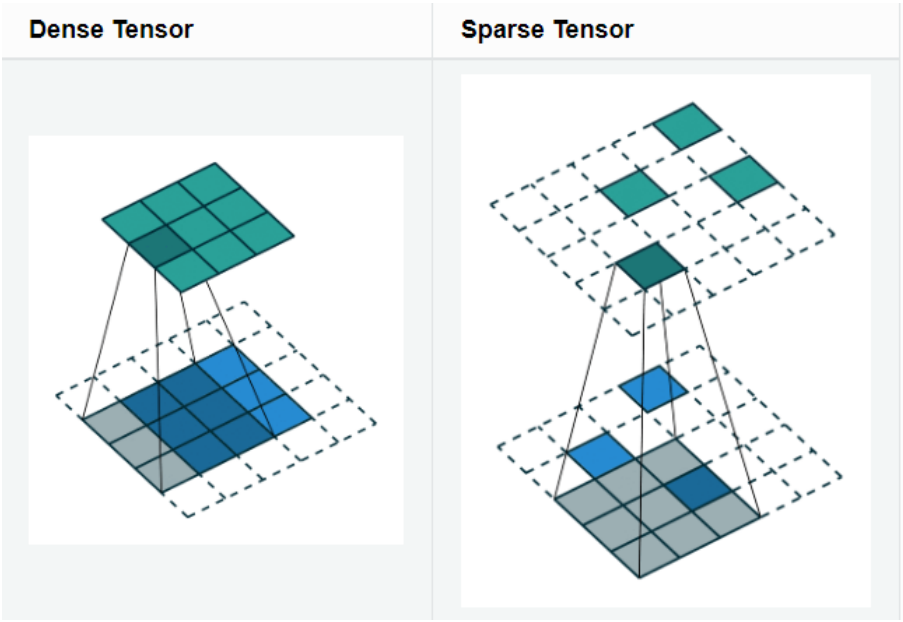


This is the maths in Choy2019 - FCGF.pdf. He stores 2 matrices. C is the matrix of coordinates (x,y,z) and index of that point (b). F is the feature matrix (the feature of that point)

$$C = \begin{bmatrix} x_1 & y_1 & z_1 & b_1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & b_N \end{bmatrix}, \; F = \begin{bmatrix} \mathbf{f}_1^T \\ \vdots \\ \mathbf{f}_N^T \end{bmatrix}$$

Different from traditional convolution which the kernel will slide all over the space and calculate all the parameters. We have two kinds of sparse convolution

- Sparse convolution: This one is similar to the regular convolution. If the convolution kernel covers a region on the input which is not having any active site, the output is not computed for that region.
- Submanifold convolution: Just calculate if and only if the kernel centre is in the exact location with the point



Why do we need Sparse convolution when it's mostly the same with Regular convolution. The answer is the Submanifold convolution may lose a lot of the information

2.1.2. Loss Function

I wrote about some losses for similarity/dissimilarity in Matching. They are Contrastive, Triplet, and Quadruplet Loss. The goal is to shorten the distance between similar points and to push dissimilar points far away from each other.

"For both contrastive and triplet losses, the sampling strategy affects the performance greatly as the decision boundary is defined by very few **hardest negatives**." Choy2019

"Traditional metric learning assumes that the features are independent and identically distributed (i.i.d.) (**two neighbouring connected components are treated completely independently**). However, in fully-convolutional feature extraction, **adjacent features are locally correlated**. Thus, hard-negative mining could find features adjacent to anchors, and they are false negatives. Thus, filtering out the false negatives is crucial" Choy2019

"FCGF generates ~40k features for a pair of scans FCGF generates ~40k features for a pair of scans. Thus, it is not feasible to use all pairwise distances within a batch as in standard metric learning."

Let me illustrate an example for you to understand three passages above in Choy2019. Because it was hard for me to understand them at first. Let's say I have an image of a race. I'll have 2 classes (runner/positive and non-runner/negative) by applying bounding boxes

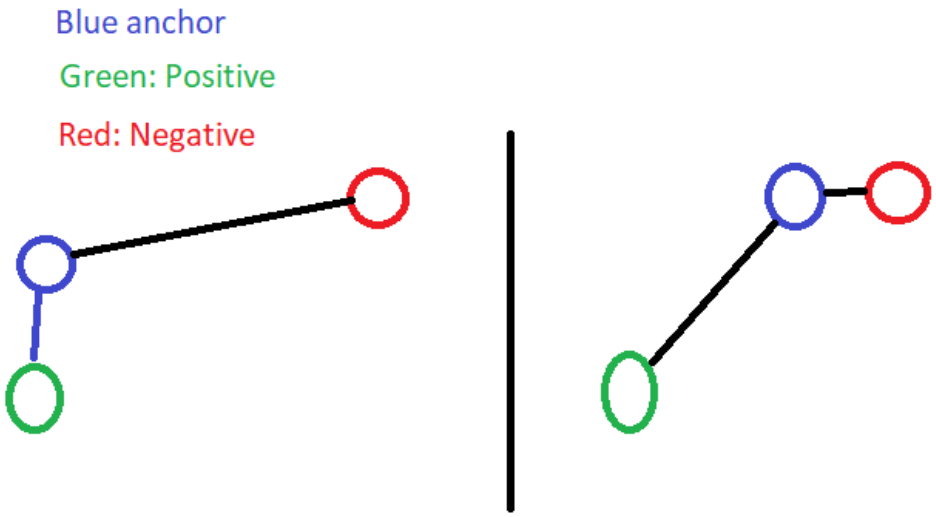
around people to have a positive and randomly cut an area which doesn't include a person to label it as a negative.

Ok, so you have positives and negatives, so you train a classifier, and to test it out, you run it on your training images again with a sliding window. But it turns out that your classifier isn't very good, because it throws a bunch of false positives (people detected where there aren't actually people).

A hard negative is when you take that falsely detected patch, and explicitly create a negative example out of that patch, and add that negative to your training set. When you retrain your classifier, it should perform better with this extra knowledge, and not make as many false positives.

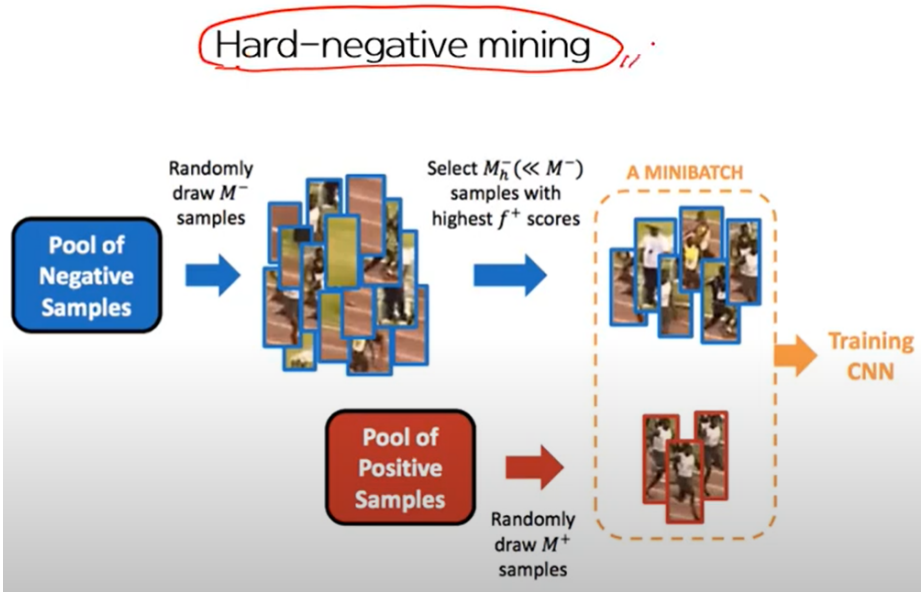
"In whatever you choose to do, do it because it is hard, not because it is easy" Neil DeGrasse Tyson

From the example above we know that the left case doesn't provide more information than the right case. Because the positive is already closer to the anchor and the negative is already far away from the anchor. There is nothing left to learn. Now let's think about 3 passages in the beginning. I think you may understand them now.



There are two terms that you need to understand

- Hard triplets: Triplets that are not in the correct configuration, where the anchor-positive similarity is less than the anchor-negative similarity
- Hard negative mining : A triplet selection strategy that seeks hard triplets, by selecting for an anchor, the most similar negative example



Reference: [Hard negative examples are hard, but useful]

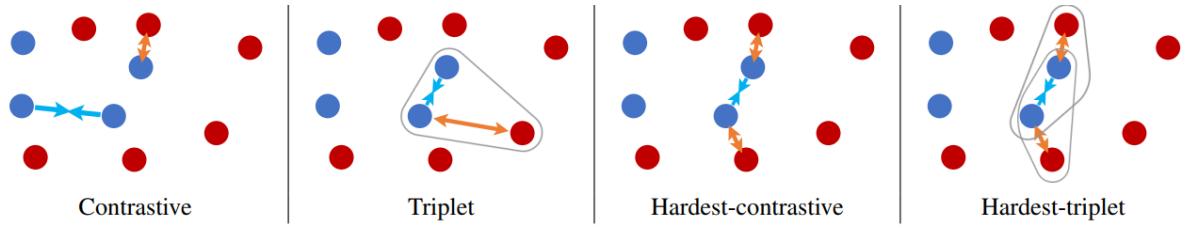


Figure 3: Sampling and negative-mining strategy for each method. Traditional contrastive and triplet losses use random sampling. Our hardest-contrastive and hardest-triplet losses use the hardest negatives.

In sum, the special thing about Hardest-contrastive or Hardest-triplet is it has a different sampling technique that focuses on finding the hardest negative false.

“We find the hardest negatives for all positive pairs and filter out the hardest negatives that fall within the vicinity of positive pairs by comparing the hash keys. Filtering out hash keys can be implemented efficiently using two sets of sorted lists.” Choy2019

2.2. Multi-view 3D point cloud reg

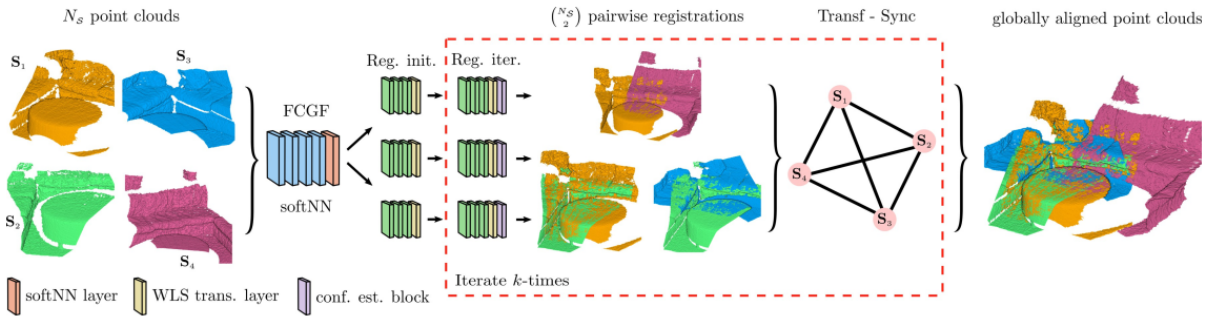


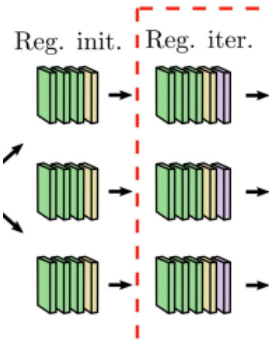
Figure 2. Proposed pipeline for end-to-end multiview 3D point cloud registration. For each of the input point clouds $\mathbf{S}_i$ we extract FCGF [16] features that are fed to the softNN layer to compute the stochastic correspondences for $\binom{N_S}{2}$ pairs. These correspondences are used as input to the initial registration block (i.e. *Reg. init.*) that outputs the per-correspondence weights, initial transformation parameters, and per-point residuals. Along with the correspondences, the initial weights and residuals are then input to the registration refinement block (i.e. *Reg. iter.*), whose outputs are used to build the graph. After each iteration of the *Transf-Sync* layer the estimated transformation parameters are used to pre-align the correspondences that are concatenated with the weights from the previous iteration and the residuals and feed anew to *Reg. iter.* block. We iterate over the *Reg. iter.* and *Transf-Sync* layer for four times.

In this part, I just write in detail about the FCGF and how the author increases the reliability of correspondences. I will not discuss the synchronisation among multiview points cause I don't understand them (I'll try to update this part in the future).

2.2.1. Deep pairwise registration

Despite the good performance of FCGF. There remains some false correspondence. Furthermore, we can't see inliers and outliers as noise (cause they occur regularly and they have a much different meaning than noise). So in this case, the author uses a Deep Neural Network as a Weighting layer (to determine inliers and outliers). And then using a 3D outliers filtering to remove outliers. After having reliable correspondences, they calculate R and t.

In addition, the paper repeats the process with R,t and w to increase the performance. "w(k) are then, again fed together with the initial correspondences to obtain the refined pairwise transformation parameters. " Gojcic2020



Beside that, to ensure the permutation-invariant, a PointNet-like structure are used in both registration blocks

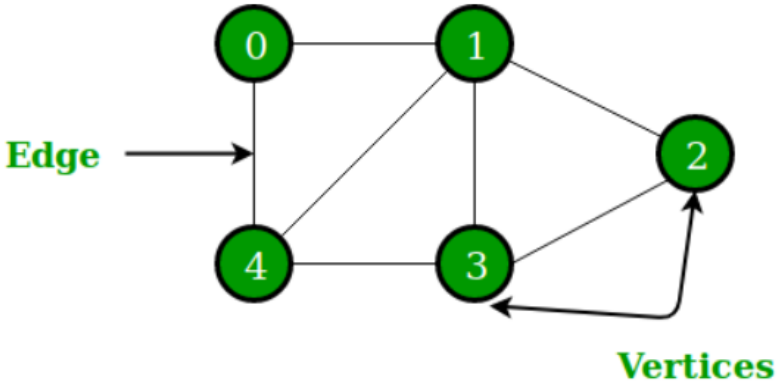
2.2.2. Transformation Synchronisation: Update later

3. Dynamic Graph CNN (DGCNN), Linked Dynamic Graph CNN (LDGCNN), Deep Closest Point (DCP)

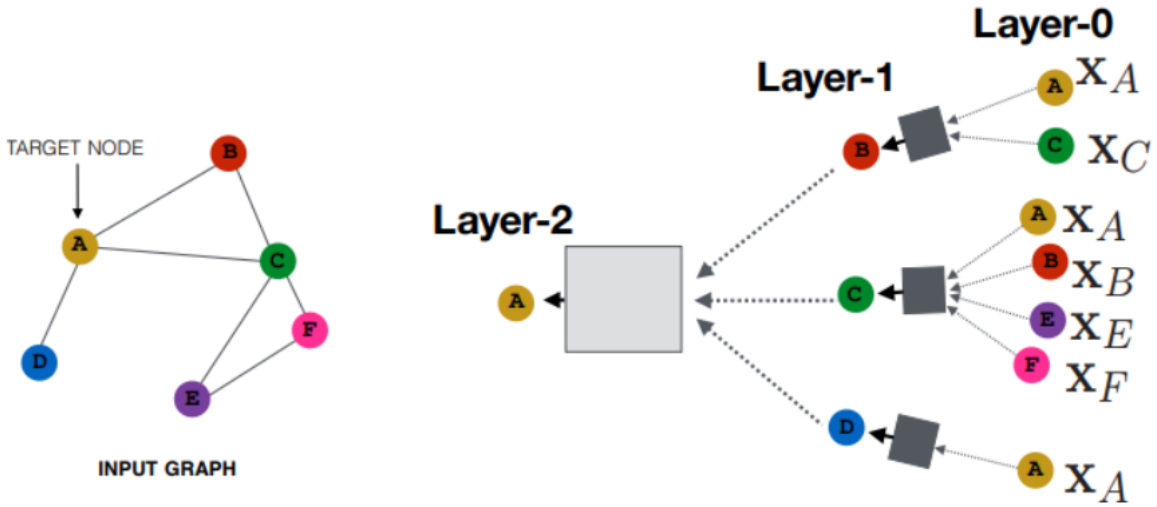
3.1. DGCNN (Dynamic Graph CNN)

3.1.1. Graph CNN

First, the graph is a combination of vertices/nodes and edges which can be described as a connection between nodes. For example, each person is a node and has node features (the weight, height, gender,...), while the edge feature is the relationship between 2 people (are they married, friend,...)



Basically we can use neighbouring node and edge features to classify each node/ edge. But usually in machine learning, we just use node features or just in a simple way. Like the graph above, If we want to label the node 4, we need information of node 0,1, and 3. But we forget to take node 2 into account (cause it has influence on node 1, 3. Herein, It definitely affects node 4 too)



Classify node A. X is the node features

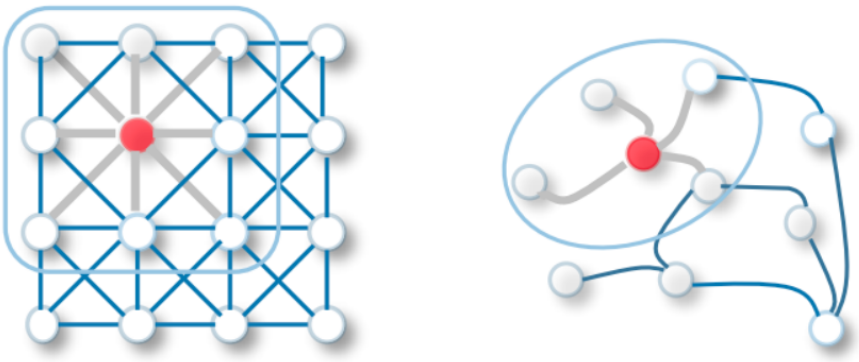
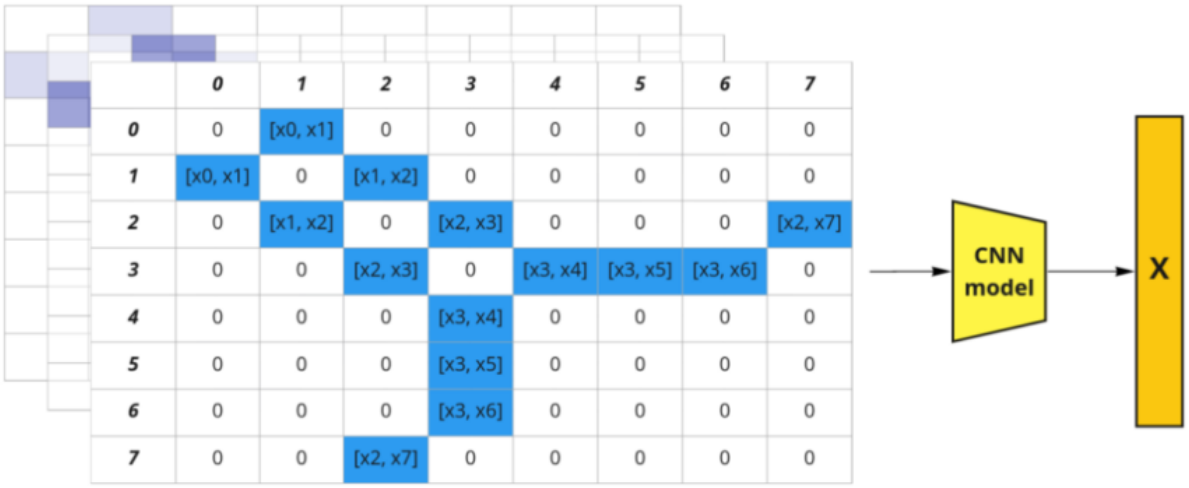


Illustration of 2D Convolutional Neural Networks (left) and Graph Convolutional Networks (right), via source

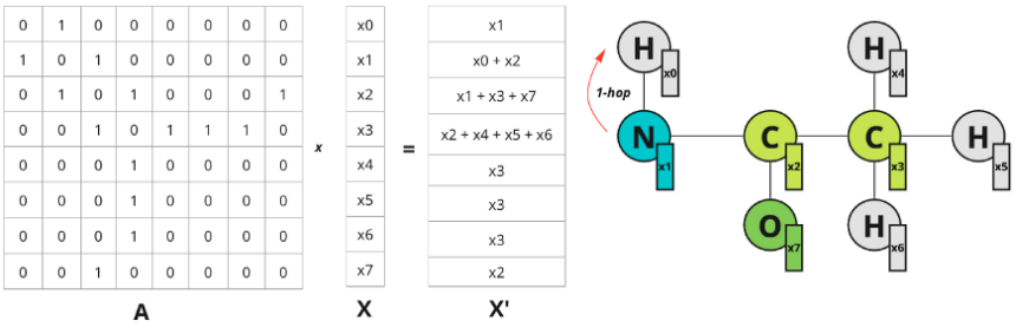
In general, we can describe a graph as a matrix (like image) with the number of row/column is a number of nodes and a value as a link/features between nodes, and then slide a kernel on that matrix like CNN. We call that matrix A (Adjacency matrix)



	0	1	2	3	4	5	6	7
0	0	[x0, x1]	0	0	0	0	0	0
1	[x0, x1]	0	[x1, x2]	0	0	0	0	0
2	0	[x1, x2]	0	[x2, x3]	0	0	0	[x2, x7]
3	0	0	[x2, x3]	0	[x3, x4]	[x3, x5]	[x3, x6]	0
4	0	0	0	[x3, x4]	0	0	0	0
5	0	0	0	[x3, x5]	0	0	0	0
6	0	0	0	[x3, x6]	0	0	0	0
7	0	0	[x2, x7]	0	0	0	0	0

Graph structure can be represented as an adjacency matrix. Node features can be represented as channels in images (brackets stand for concatenation) [Image by author].

But as the Adjacency matrix above, if we change the order of column or row. The result would be different. That's when we come up with Symmetric functions (SUM or MAX) to get permutation-invariant (Remember those terms in PointNet?)



An example for scalar value node features. 1-hop distance is illustrated only for node 0, but the same holds for all other nodes [Image by author].

After all, we have X'=A.X. Of course things are much more complicated in real life. But this is enough for you to understand DGCNN

Reference:

- The Intuition Behind Graph Convolutions and Message Passing | by Gleb Kumichev | Towards Data Science
- Understanding Graph Convolutional Networks for Node Classification | by Inneke Mayachita | Towards Data Science
- Graph Convolutional Networks (GCNs) made simple - YouTube

3.1.2. DGCNN

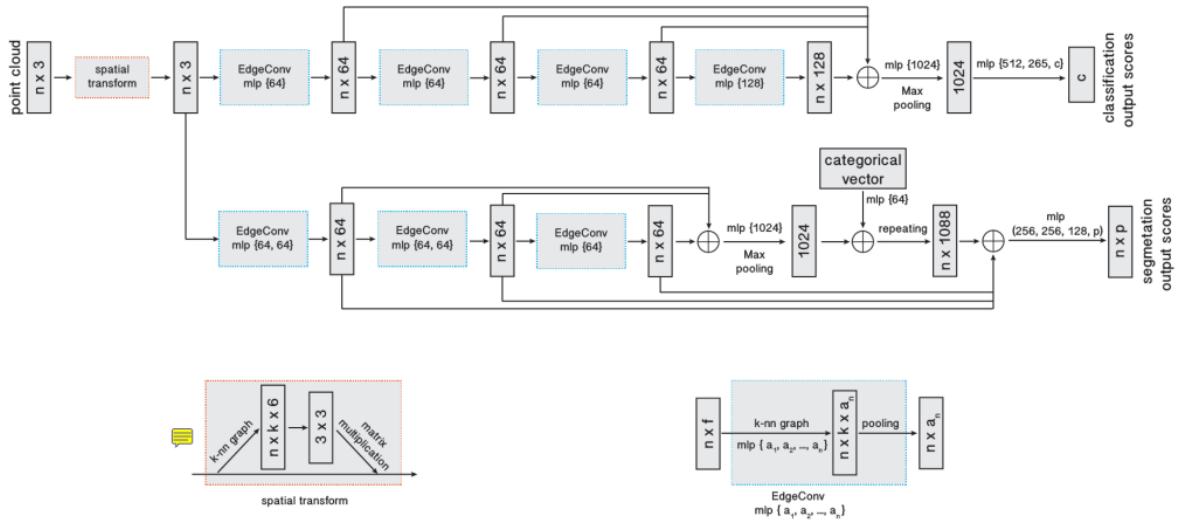


Fig. 3. Model architectures: The model architectures used for classification (top branch) and segmentation (bottom branch). The classification model takes as input n points, calculates an edge feature set of size k for each point at an EdgeConv layer, and aggregates features within each set to compute EdgeConv responses for corresponding points. The output features of the last EdgeConv layer are aggregated globally to form an 1D global descriptor, which is used to generate classification scores for c classes. The segmentation model extends the classification model by concatenating the 1D global descriptor and all the EdgeConv outputs (serving as local descriptors) for each point. It outputs per-point classification scores for p semantic labels. ⊕: concatenation. Point cloud transform block: The point cloud transform block is designed to align an input point set to a canonical space by applying an estimated 3×3 matrix. To estimate the 3×3 matrix, a tensor concatenating the coordinates of each point and the coordinate differences between its k neighboring points is used. EdgeConv block: The EdgeConv block takes as input a tensor of shape $n \times f$, computes edge features for each point by applying a multi-layer perceptron (mlp) with the number of layer neurons defined as $\{a_1, a_2, \ldots, a_n\}$, and generates a tensor of shape $n \times a_n$ after pooling among neighboring edge features.

In this part, I will compare this technique with PointNet++ for us to understand more about not just DGCNN but also PointNet/PointNet++ under the graph-based perspective.

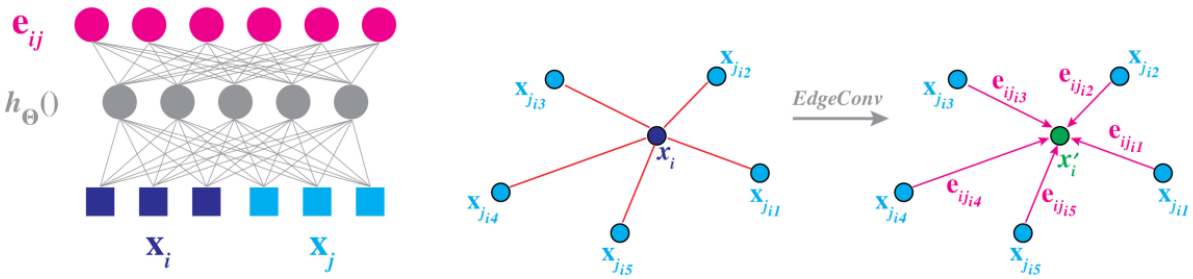


Fig. 2. Left: Computing an edge feature, $\mathbf{e}_{ij}$ (top), from a point pair, $\mathbf{x}_i$ and $\mathbf{x}_j$ (bottom). In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. Right: The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

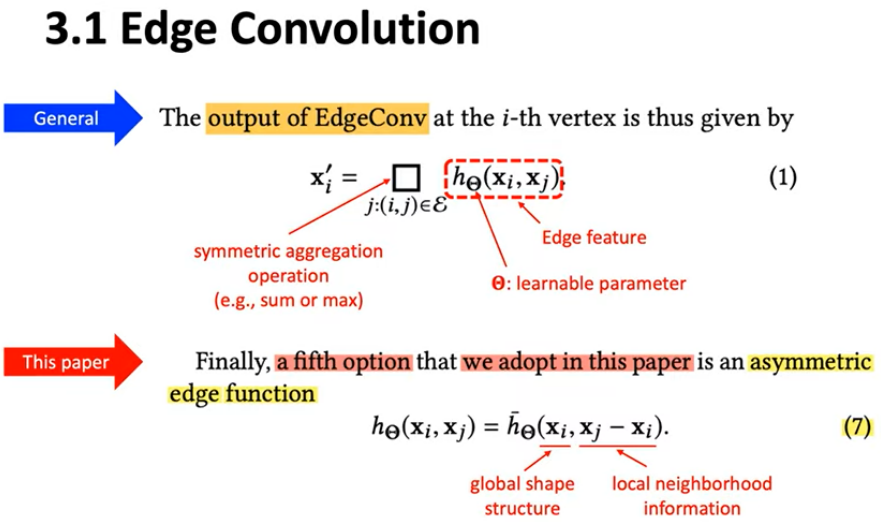
DGCNN is a descriptor which is translation-invariant and rotation-invariant. It's quite hard to understand the difference between DGCNN and Graph CNN

“Our experiments suggest that it is beneficial to **recompute the graph** using **nearest neighbors** in the **feature space** produced by each layer. This is a crucial distinction of our method from graph CNNs working on a **fixed input graph**”

At first, I don’t really understand the term “fixed” and “dynamic”. Which part is fixed and which is dynamic?? The node, edge, or the feature?? Let rewind the Graph Convolutional Networks (GCNs) made simple. As you can see, Graph CNN maintains the same graph form while updating the features. Because they employ the K-NN for a given feature (i.e. point coordinates) across all layers, the shape stays the same. However, different features are used in each layer of DGCCN to determine the similarity between points, which means the graph has to change. Higher layers of DGCNN bring semantic points closer together in feature space: Dynamic Graph CNN for Learning on Point Cloud.

For example, graph CNN organised a book shelves by the author’s name for 2 layers. DGCNN organised a book shelves by the author’s name in the first layer, and by color in the second layer.

For each point in a point set. We take k-nn points around xi (in feature space) to update the feature of that node xi’ and the edge features eij after layers and layers (That’s why they’re **Dynamic**). Finally we concatenate those graphs to have the final result



3.1 Edge Convolution

General

The output of EdgeConv at the i-th vertex is thus given by

$$\mathbf{x}'_i = \square_{j:(i,j)\in\mathcal{E}} h_\Theta(\mathbf{x}_i, \mathbf{x}_j), \quad (1)$$

symmetric aggregation operation (e.g., sum or max)

Edge feature

Θ: learnable parameter

This paper

Finally, a fifth option that we adopt in this paper is an asymmetric edge function

$$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = \bar{h}_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i). \quad (7)$$

global shape structure

local neighborhood information

If you remember the architecture of PointNet++, it used grouping and sampling in Euclidean space (In some way, we can say that PointNet++ is a kind of Graph CNN in Euclidean space, and the graph is fixed too), while EdgeConv grouping points in feature space and Euclidean space which is better local descriptiveness in some way. Furthermore, in contrast to DGCNN, PointNet++ intuitively indicates that distant points (in Euclidean space) frequently have different properties (distant points still have the same features).

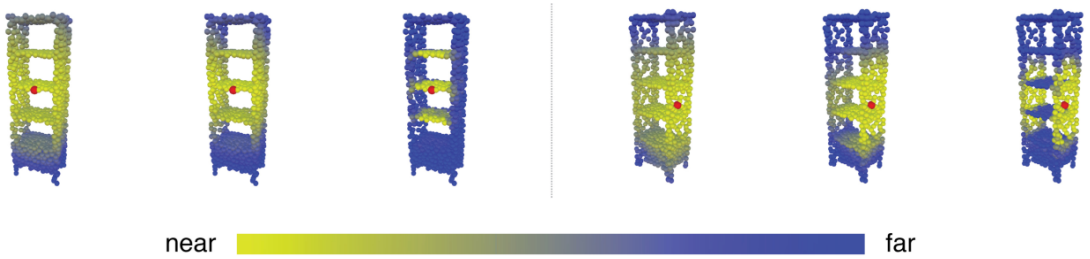


Fig. 4. Structure of the feature spaces produced at different stages of our shape classification neural network architecture, visualized as the distance between the red point to the rest of the points. For each set, Left: Euclidean distance in the input $\mathbb{R}^3$ space; Middle: Distance after the point cloud transform stage, amounting to a global transformation of the shape; Right: Distance in the feature space of the last layer. Observe how in the feature space of deeper layers semantically similar structures such as shelves of a bookshelf or legs of a table are brought close together, although they are distant in the original space.

The global feature of PointNet++ is made up of fading local features that are gradually reduced from layer to layer by grouping and sampling. Those techniques just retain some keypoints and features of their neighbours. DGCNN, on the other hand, directly concatenates local features from all layers into a single global feature.

These are the reasons why I feel DGCNN outperforms PointNet++ in terms of local and global features. In fact, DGCNN is 7 times faster than PointNet++ and has a higher accuracy (1 to 2%).

In contrast, in some YouTube videos, DGCNN's segmentation task is good but not state-of-the-art. The question of focusing on edegs but not relative location of points, and are high-level spatial linkages robust still remain. Dynamic Graph CNN for Learning on Point Cloud

3.2. LDGCNN (Linked Dynamic Graph CNN)

LDGCNN is an updated version of DGCNN which helps it achieve state-of-the-art accuracy in classification while reduce the processing time, and model size compared to PointNet, PointNet++, DGCNN.
All the source code is here: GitHub - KuangenZhang/ldgcnn: Linked Dynamic Graph CNN: Learning through Point Cloud by Linking Hierarchical Features

Method	Input	MA (%)	OA (%)
PointNet [18]	1024 points	86.0	89.2
PointNet++ [20]	5000 points+normal	-	90.7
KD-Net [28]	1024 points	-	91.8
DGCNN [21]	1024 points	90.2	92.2
SpiderCNN [22]	1024 points+normal	-	92.4
PointCNN [23]	1024 points	88.1	92.2
Ours	1024 points	**90.3**	**92.9**

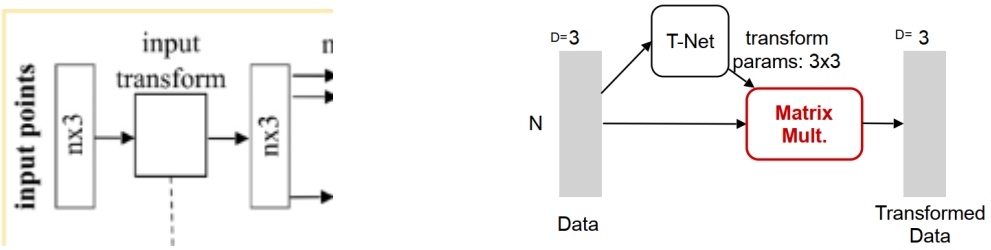
Overall accuracy (OA) of LDGCNN compared to other methods

The 3 key contributions of LDGCNN include:

- Remove the transformation network from the DGCNN and demonstrate that MLP can be used to extract transformation invariant features
- They link hierarchical features from different dynamic graphs to calculate informative edge vectors and avoid vanishing gradient problem
- Increase the performance by freezing the feature extractor and retraining the classifier.

3.2.1. Remove the transformation network

PointNet or DGCNN use a transformation network to estimate the Rotation and translation (R,t) to offset the point cloud (It's like the Center of Mass in ICP). Nevertheless, this layer will double the size of the whole model



Transformation net usually include a matrix multiply

$$\boldsymbol{P}_{\text{offset}} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

The math of Transformation net (x,y,z: coordinates; rij: rotation value)

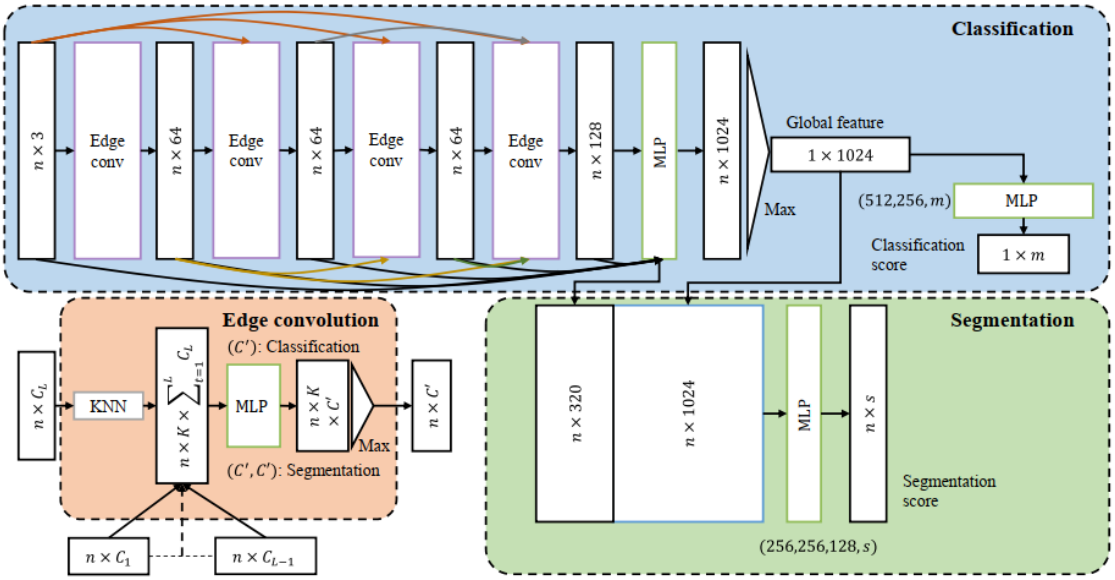
However, the authors came up with a clever idea by rewriting the math above into MLP with Weight and Bias. We can see that the two functions are similar. “The difference is that the transformation network can estimate a specific matrix for each point cloud, whereas the MLP is static for all point cloud” Zhang2019

$$\boldsymbol{h}(\boldsymbol{P}) = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1C'} \\ w_{21} & w_{22} & \cdots & w_{2C'} \\ w_{31} & w_{32} & \cdots & w_{3C'} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \cdots & b_{C'} \end{bmatrix}$$

The math of MLP

Furthermore, they augment data like in PointNet++ (rotate, translate, scale and random noise) to achieve invariant characteristics as well.

3.2.2. Link hierarchical features



Apart from black connections as we saw in DGCNN, we can see LDGCNN also has red, orange, grey connections from early EdgeConv to later EdgeConv stages. The primary reasons for this are:

- To avoid vanishing gradient problem for deep neural networks
- The neighbours of current features are similar features that may cause the edge vetor approach to zero. As a consequence, the author hopes to get a better outcome by combining prior and present characteristics.

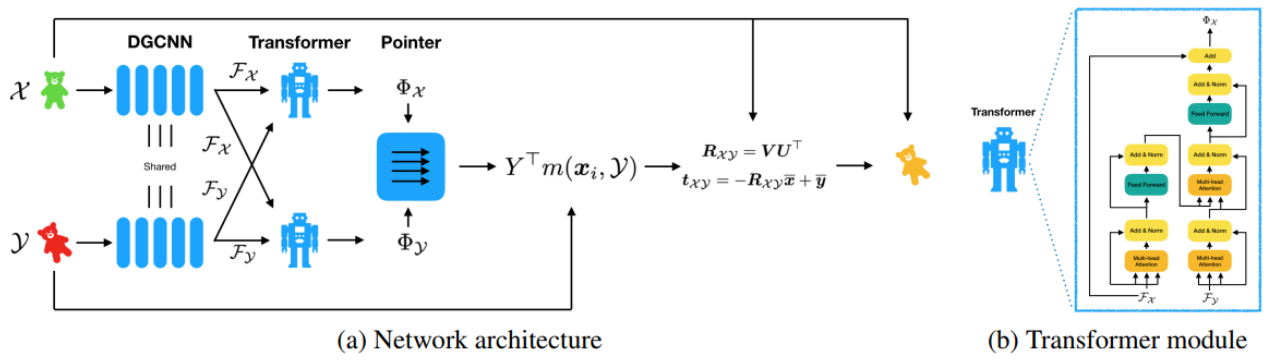
3.2.3. Freezing feature extractor and retraining the classifier

Theoretically, training the whole model can reach a global minimum. However, there are some reasons for the opposite:

- There are too many parameters to train which is impossible to find a global minimum (we are talking about the non-transformation-layer model LDGCNN). The computational cost increase as the number of points increase
- In contrast, due to the enormous number of parameters, the network may fall into a local minimum.

This help LDGCNN achieve better performance

3.3. DCP (Deep Closest Point)



(a) Network architecture (b) Transformer module

Figure 2. Network architecture for DCP, including the Transformer module for DCP-v2.

The DCP is a point set registration technique including 3 main parts

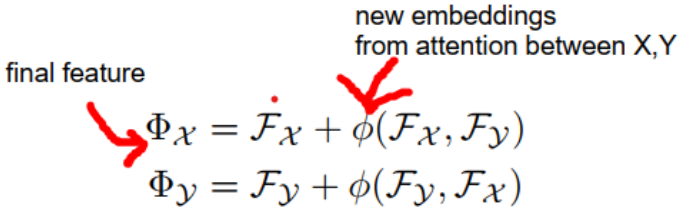
- Feature extraction with **DGCNN**
- Using **Attention** module combining **pointer network** to predict a soft matching between the point clouds
- Using **SVD** to transform reliable correspondences into R and t: SVD stimulation in ICP

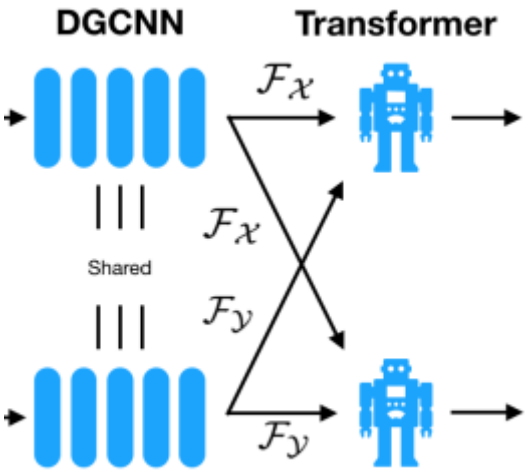
The authors demonstrate that using ICP as a refinement can improve outcomes.

Because there is a lot of discussion on SVD on the internet, and I have a stimulation link for that approach above, I'll simply focus on the Attention layer and pointer network in this section.

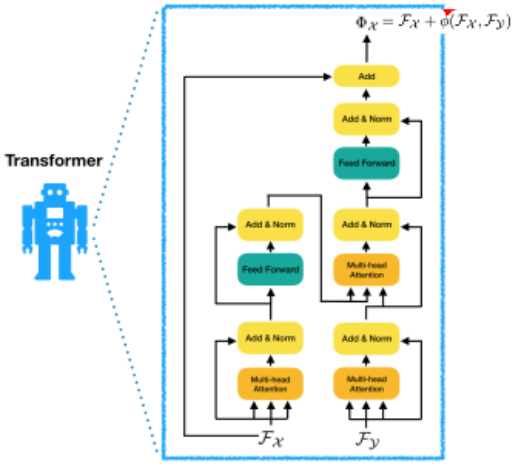
3.3.1. Attention

As a result of the experiences gained on both PointNet and DGCNN, the optimal descriptor is a combination of local and global information. As a consequence of this concept, the author wishes to associate both X and Y point cloud features, which differs from previous approaches in which the features of X and Y are extracted separately and then compared.
They design a module to learn co-context information by capturing self attention and conditional attention
FX and FY to be the embeddings generated by DGCNN.





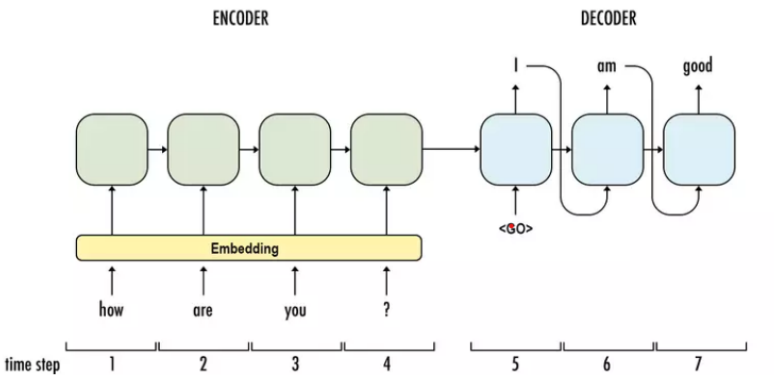
They choose Φ as an **asymmetric function** given by a **transformer**, because they find that the matching in rigid alignment is analogous to the seq-to-seq of NLP problem (using their positional embeddings to describe where words are in a sentence)



(b) Transformer module

So what is a Transformer? Let's say RNN in NLP have a bunch of problems like:

- vanishing/exploding gradient which makes it hard to remember the information from far away in the sentence
- The learning mechanism from left to right of RNN (inductive bias) leads to the long training time as well as it can not learn the structure features. For example, in the sentence "She is eating a green apple." Obviously, "apple" has more to do with "eating" than any other word.

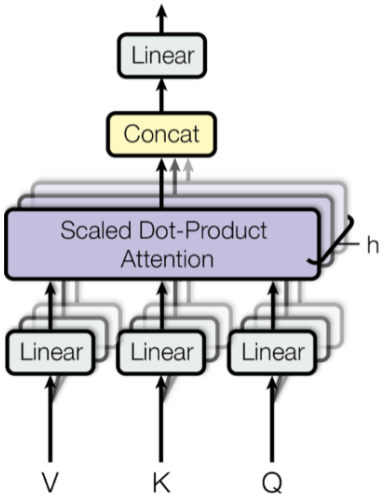


RNN structure

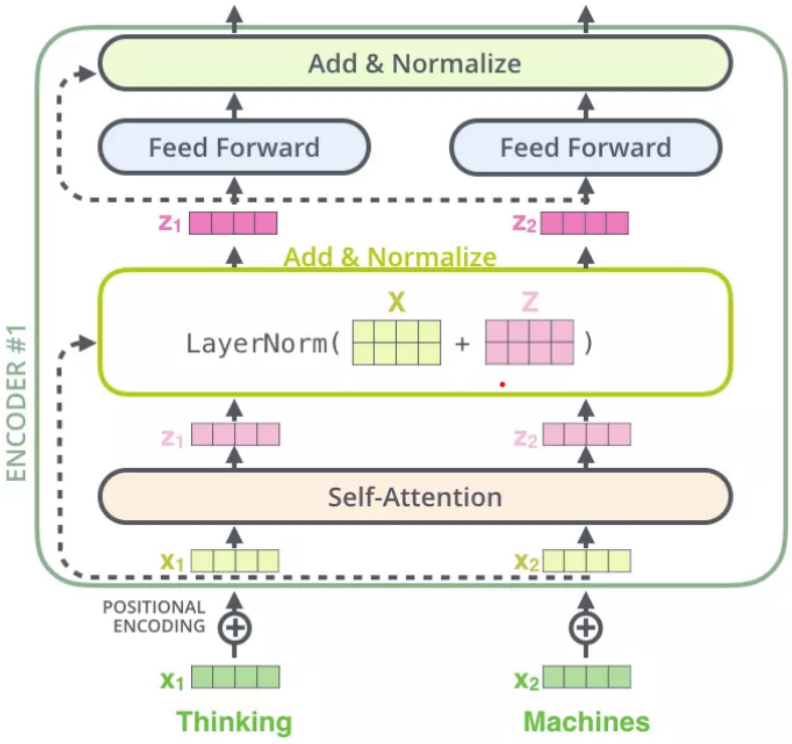
Transformer helps solve those problems:

- Multi-head attention: to learn with parallel inputs which means we can train the model faster with GPU. There is no term “time step” in transformer
- Self attention: to extract the relationships among words, and it helps focus on which word important
- Positional encoding: The position and order of words in a sentence is essential to all linguistic models

“**Multi-head Attention** is a module for attention mechanisms which runs through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension. Intuitively, multiple attention heads allows for attending to parts of the sequence differently“



Multi-Head Attention



Encoder of Transformer architecture

3.3.2. Pointer generation

In fact, it's a matching process. They will calculate the similarity of xi feature in point cloud X to all point features in point cloud Y and convert them into a probability vector by a softmax function. The higher the probability, the higher the confidence for that matching pair.

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \mathrm{softmax}(\Phi_{\mathcal{Y}} \Phi_{\boldsymbol{x}_i}^{\top}).$$

"ΦY 2 $R^{N\times P}$ denotes the embedding of Y generated by the attention module, and Φxi denotes the i-th row of the matrix ΦX from the attention module. We can think of m(xi; Y) as a soft pointer from each xi into the elements of Y" Wang2019-DCP

3.3.3. Disadvantage

"DCP [35] assumes that all the points in the source point cloud have correspondences in the target point cloud." Li2020

This assumption is quite wrong because of the sparsity and partiality, two point clouds do not always have point-to-point correspondences

4. PointCNN - Convolution on X-transformed points

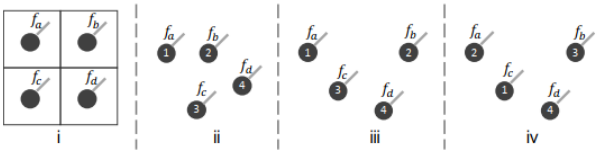


Figure 1: Convolution input from regular grids (i) and point clouds (ii-iv). In (i), each grid cell is associated with a feature. In (ii-iv), the points are sampled from local neighborhoods, in analogy to local patches in (i), and each point is associated with a feature, an order index, and coordinates.

$$
\begin{aligned}
f_{ii} &= Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T), \\
f_{iii} &= Conv(\mathbf{K}, [f_a, f_b, f_c, f_d]^T), \\
f_{iv} &= Conv(\mathbf{K}, [f_c, f_a, f_b, f_d]^T).
\end{aligned} \tag{1a}
$$

$$
\begin{aligned}
f_{ii} &= Conv(\mathbf{K}, \mathcal{X}_{ii} \times [f_a, f_b, f_c, f_d]^T), \\
f_{iii} &= Conv(\mathbf{K}, \mathcal{X}_{iii} \times [f_a, f_b, f_c, f_d]^T), \\
f_{iv} &= Conv(\mathbf{K}, \mathcal{X}_{iv} \times [f_c, f_a, f_b, f_d]^T).
\end{aligned} \tag{1b}
$$

The figure above illustrates the challenges of data represented in point clouds which are irregular and unordered. Eq 1a) shows that a direct convolution applied on leads to a deserting shape information (i.e., fii ≡ fiii),while retaining variance to the ordering (i.e., fiii ≠ fiv). The author came up with an idea that we have to learn **X = MLP (p1, p2...)** to **weigh and permute** input points at the same time before applying convolution on **representative points.**

It's quite familiar to see MLP in achieving permutation-invariance. The difference here is the input of MLP. PointNet uses coordinates of points as input of MLP (which make it so bad with local features), whereas PointCNN, LDGCNN… use the relative locations of that point to its neighbours as MLP's input. In addition, the MLP of PointNet is followed by a symmetric function. But PointCNN uses asymmetric function

4.1. Hierarchical Convolution

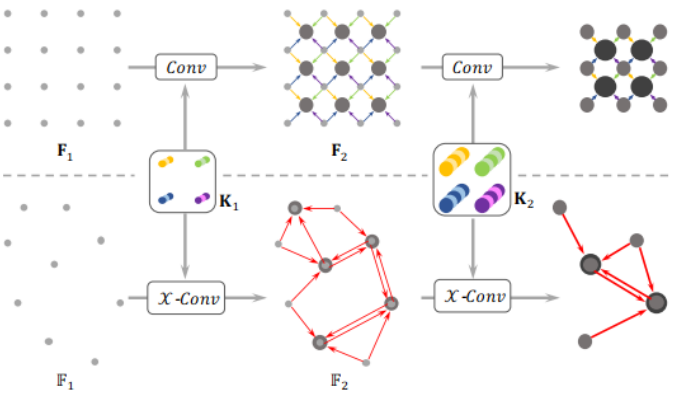


Figure 2: Hierarchical convolution on regular grids (upper) and point clouds (lower). In regular grids, convolutions are recursively applied on local grid patches, which often reduces the grid resolution ($4 \times 4 \rightarrow 3 \times 3 \rightarrow 2 \times 2$), while increasing the channel number (visualized by dot thickness). Similarly, in point clouds, $\mathcal{X}$-Conv is recursively applied to "project", or "aggregate", information from neighborhoods into fewer representative points ($9 \rightarrow 5 \rightarrow 2$), but each with richer information.

Honestly, I see PointCNN is quite like PointNet++. PointCNN uses FPS (farthest point sampling) to choose representative points and then use X=MLP (PointNet++ use PointNet which is quite like a MLP structure) to aggregate their higher features from their neighbour's

information. Furthermore, the two models both remove the neighbouring points after that layer which reduce the resolution of the point cloud. The problem of this is the information loss as some papers have stated.

In sum, I guess the primary differences between PointCNN and PoinNet++ are the X-transformer/PointNet to learn higher level representation of the key points, and the convolution at the end of PointCNN. Not to mention the PointCNN structure is quite simple compared to the result it bring to us

4.2. X-Conv Operator

ALGORITHM 1: $\mathcal{X}$-Conv Operator

Input : $\mathbf{K}, p, \mathbf{P}, \mathbf{F}$
Output : $\mathbf{F}_p$ ▷ Features "projected", or "aggregated", into representative point p

1: $\mathbf{P}' \leftarrow \mathbf{P} - p$ ▷ Move $\mathbf{P}$ to local coordinate system of p
2: $\mathbf{F}_\delta \leftarrow MLP_\delta(\mathbf{P}')$ ▷ **Individually** lift each point into C_δ dimensional space
3: $\mathbf{F}_* \leftarrow [\mathbf{F}_\delta, \mathbf{F}]$ ▷ Concatenate $\mathbf{F}_\delta$ and $\mathbf{F}$, $\mathbf{F}_*$ is a $K \times (C_\delta + C_1)$ matrix
4: $\mathcal{X} \leftarrow MLP(\mathbf{P}')$ ▷ Learn the $K \times K$ $\mathcal{X}$-transformation matrix
5: $\mathbf{F}_\mathcal{X} \leftarrow \mathcal{X} \times \mathbf{F}_*$ ▷ Weight and permute $\mathbf{F}_*$ with the learnt $\mathcal{X}$
6: $\mathbf{F}_p \leftarrow \text{Conv}(\mathbf{K}, \mathbf{F}_\mathcal{X})$ ▷ Finally, typical convolution between $\mathbf{K}$ and $\mathbf{F}_\mathcal{X}$

Line 1: The X-Conv operates on local features (capture the relationship between neighbouring points), so that it uses P-p as input instead of p coordinates like in PointNet. (P: neighbouring points features; p: chosen representative points features after FPS layer)

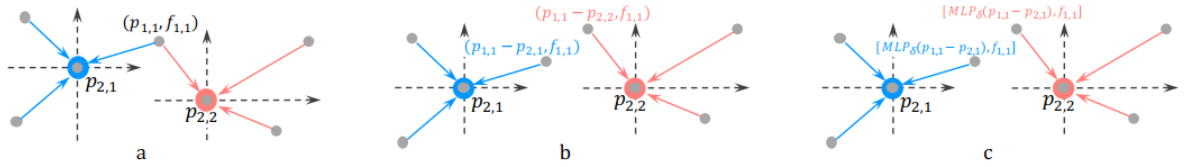


Figure 3: The process for converting point coordinates to features. Neighboring points are transformed to the local coordinate systems of the representative points (a and b). The local coordinates of each point are then individually lifted and combined with the associated features (c).

Line 2: MLPδ(·) is a multilayer perceptron applied individually on each point, as in PointNet. "However, the lifted features are not processed by a symmetric function." Li2018 .We need to lift the dimension of P-p to combine it with F in line 3 because the local coordinates are of a different dimensionality and representation than the associated features.

Line 3: Concatenate feature of each point in current layer F and feature of P-p which is Fδ to form associated feature F*. This F* is still permutation-variant

Line 4: Learning X-Conv matrix by an MLP layer which is quite similar to Remove the transformation network. It weigh and permute input points to turn F* into Fx which is permutation-invariant. This is much different from PointNet, they use a symmetric function which totally ignores the order of input points. On the other hand, PointCNN is capable of realising equivariance from the input features, and therefore has to be aware of the specific input features.

For me, I believe that PointNet uses a much simpler way to learn, but it reduces a lot of information. In PointCNN case, it retains as much information as possible but it performs quite bad with unseen data as well as it needs a lot of data to achieve this permutation-equivariant.

To further understand the difference between them, you have to understand the two terms "**Invariance** vs. **Equivariance**. A line of pioneering work aiming at achieving equivariance has been proposed to address the information loss problem of pooling in achieving invariance" Li2018

4.3. PointCNN architectures

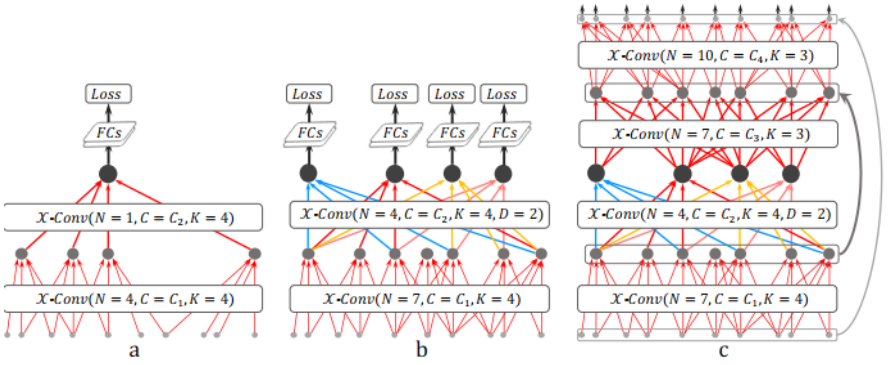


Figure 4: PointCNN architecture for classification (a and b) and segmentation (c), where N and C denote the output representative point number and feature dimensionality, K is the neighboring point number for each representative point, and D is the $\mathcal{X}$-Conv dilation rate.

Figure 4a shows a structure with 2 X-Conv layers, it reduces the resolution of the point cloud but each representative point has richer information. However, "Note that the number of training samples for the top X-Conv layers drops rapidly (Figure 4a), making it inefficient to train them thoroughly." Li2018. To resolve this problem, the authors want to make a structure with denser connections and retain more representative points as figure 4b. They maintained the depth of the model, while increasing the receptive field rate from K/N to (K+D)/N which means we have more key points at output, each having a bigger view of the point cloud shape. Which means they're better at global features, I guess.

To reduce overfitting, the authors used Dropout before the last FC layer. Theoretically, the receptive field rate = 1 is best because the representative points can see the whole picture of the point cloud. However, the rate needed to be set at less than 1 to push the model to learn harder from the partial information which would perform better at test time and with unseen data. They also employ data augmentation, in which they suggest randomly sampling and shuffling the input points owing to the knowledge about specific input order

5. Grid-GCN for fast and scalable Point Cloud Learning

This paper helps me summarize the knowledge that I've gained from others like PointNet, FCGF, DGCNN… So this section will be somewhat lengthy since I'll be concentrating on the process of each method as well as comparing it to others that we've learnt.

This is a state-of-the-art technique which was released in 2021. This method can process faster for more point set data while achieving the same or better result compared to other powerful methods like PointNet++, DGCNN,...

Input (xyz as default)		OA	latency (ms)
	OA < 84.0		
PointNet[32]	8×4096	73.9	20.3
OctNet[34]	volume	76.6	-
PointNet++[33]	8×4096	83.7	72.3
Grid-GCN$_{(0.5\times K)}$	4×8192	**83.9**	**16.6**
	OA $\geqslant 84.0$		
SpecGCN[40]	-	84.8	-
PointCNN[26]	12×2048	85.1	250.0
Shellnet[53]	-	85.2	-
Grid-GCN$_{(1\times K)}$	4×8192	**85.4**	**20.8**
A-CNN[19]	1×8192	**85.4**	92.0
Grid-GCN$_{(1\times K)}$	1×8192	**85.4**	**7.48**

Table 4: Results on ScanNet[9]. Grid-GCN achieves $10\times$ speed-up on average over other models. Under batch size of 4 and 1, we test our model with $1\times K$ neighbor nodes. A compact model with $0.5 \times K$ is also reported.

The reason for this outstanding outcome is because it solves two problems of the old methods: volumetric and point-based methods.

- Data structuring: Point-based techniques such as PointNet++ use FPS (finding key points) and KNN/Ball Query (seeking neighbors) throughout the whole point cloud, making this the method's bottleneck (Imagine the time to process hundreds of thousands points). In the case of Volumetric methods, they group points into a grid. This technique lowers the resolution of the point set that helps process much faster.

- Computational cost: Point-based computational cost increases linearly with the number of input points. Volumetric methods' computation and memory usage increase cubically as the locations of points expand in three dimensions. Not to mention most of the computation is wasted by processing no information (the sparse of the point cloud)

To solve these problems, the authors propose Grid-GCN a lot of GridConv layers, each one includes two main stages which make it fast and scalable:

- Coverage-aware grid query (**CAGQ**) module to achieve efficient data structuring through voxelization that accelerates **centers sampling** and **neighbors query.** Simultaneously provide more complete **space coverage**.
- Grid context aggregation (**GCA**): to exploit the point cloud by building local graphs. The authors propose **Grid context pooling** and **coverage weight** to outperform which benefits the edge relation without any cost extra

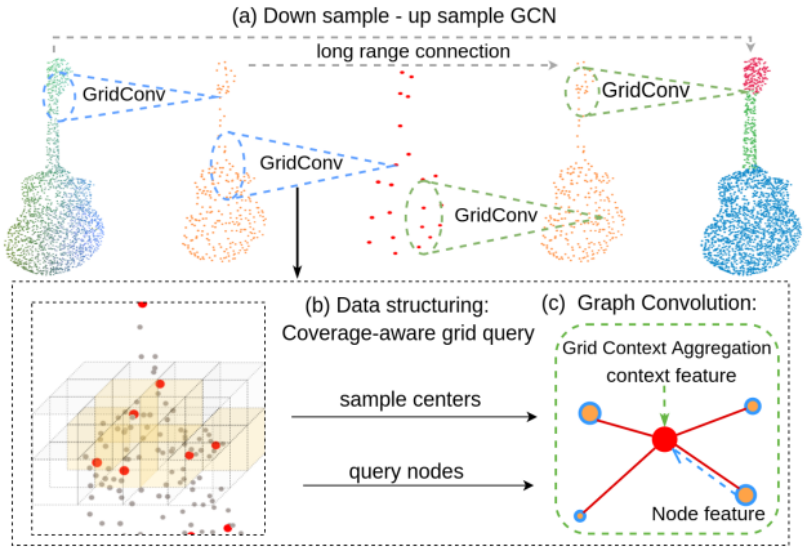


Figure 1: Overview of the Grid-GCN model. (a) Illustration of the network architecture for point cloud segmentation. Our model consists of several GridConv layers, and each can be used in either a downsampling or an upsampling process. A GridConv layer includes two stages: (b) For the data structuring stage, a Coverage-Aware Grid Query (CAGQ) module achieves efficient data structuring and provides point groups for efficient computation. (c) For the convolution stage, a Grid Context Aggregation (GCA) module conducts graph convolution on the point groups by aggregating local context.

5.1. Coverage Aware Grid Query (CAGQ)

The data structuring method usually has two parts: center sampling (keypoint detection) and neighbor query (for information aggregation). In center sampling we have learned 3 techniques

- RPS (Random point sampling): each sample has an equal probability of being chosen. The higher density point area has more probability to be chosen, that's why it can cover the sparse area.
- FPS (Farthest point sampling): more in FPS. This method choosing points that far from each other which is usually the edges or corners of the objects, which causes the gap between centers
- Learning with PointNet: In DeepVCP, to discover critical points, they employ a deep learning model that includes PointNet++ and a point weighting layer. This is usually correct but computationally expensive.

For neighbor query we've learned two techniques KNN and Ball Query in Hierarchical Network. Because we don't divide the point set into grid so we need to search for the whole point cloud to find neighbors which cost a lot of time

Now let's move on to the process of CAGQ to understand why it is so fast

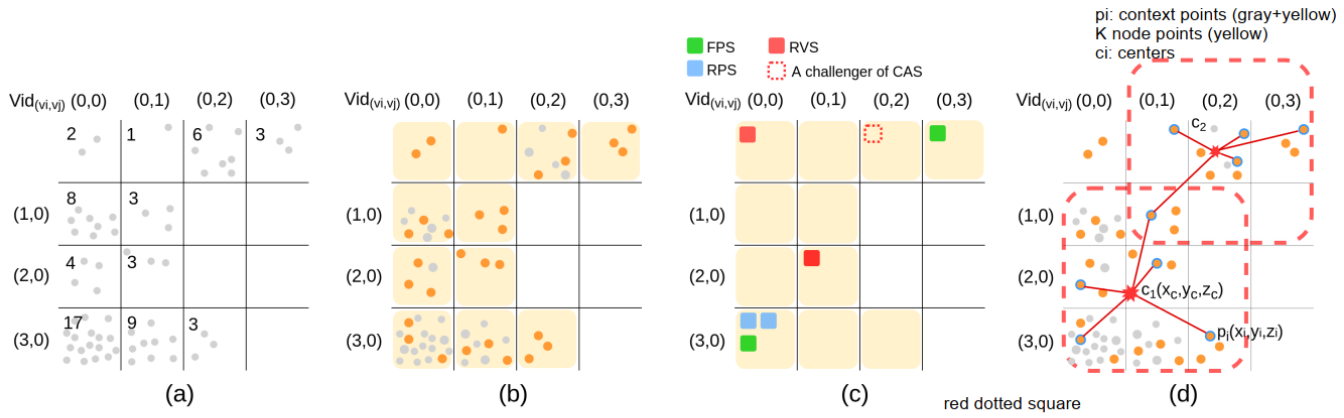


Figure 2: Illustration of Coverage-Aware Grid Query (CAGQ). Assume we want to sample $M = 2$ point groups and query $K = 5$ node points for each group. **(a)** The input is N points (grey). The voxel id and number of points is listed for each occupied voxel. **(b)** We build voxel-point index and store up to $n_v = 3$ points (yellow) in each voxel. **(c)** Comparison of different sampling methods: FPS and RPS prefer the two centers inside the marked voxels. Our RVS could randomly pick any two occupied voxels (e.g. (2,0) and (0,0)) as center voxels. If our CAS is used, voxel (0,2) will replace (0,0). **(d)** Context points of center voxel (2,1) are the yellow points in its neighborhood (we use 3×3 as an example). CAGQ queries 5 points (yellow points with blue ring) from these context points, then calculate the locations of the group centers.

1. Divide the space into grids, then classify empty and non-empty squares/voxels. We just focus on non-empty/occupied voxels
2. Sample M center voxels, from there we list there neighbor voxels π(vi), all points in π(vi) are called context points
3. Pick K node points form context points of each group

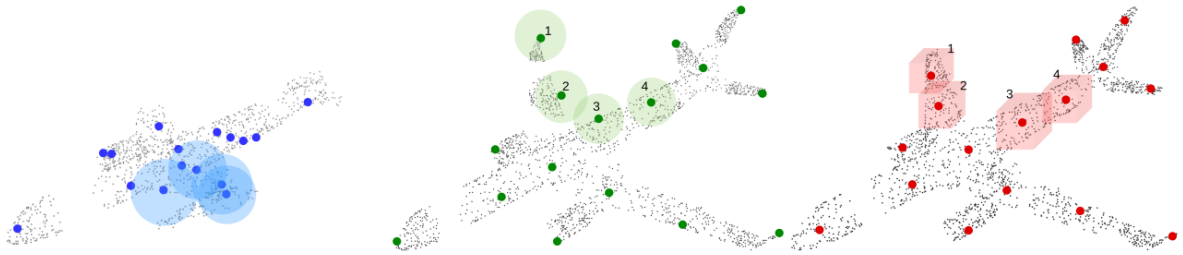
As previously mentioned, even voxelizing the input space and concentrating exclusively on occupied voxels, as described in step 1, speeds up the process. However, the authors suggest two more powerful strategies for selecting **M center voxels** and **K neighbors (node points)** in steps 2 and 3.

There are 2 methods for **Sampling Center Voxels**

- RVS (Random Voxel Sampling): It's like RPS, but they randomly choose voxels instead of points. "The group centers calculated inside these center voxels are **more evenly distributed**" Xu2021
- CAS (Coverage-Aware Sampling): This technique focuses on maximizing the space covered by using **incumbents and challengers.** They first randomly pick M center voxels as incumbents. Then pick another voxel as a challenger from another voxel. If overall coverage increases, the challengers will be maintained as incumbents, and vice versa.

The authors provide 2 ways of **node points querying**

- Cube Query: randomly select K node points from all context points. It can cover more space with imbalances density compared to Ball Query used in PointNet++
- KNN: In contrast to the traditional KNN in PointNet++. KNN in CAGQ just need to search among context points, making the query faster



(a) Random Point Sampling (b) Farthest Point Sampling (c) Coverage-Aware Sampling

Figure 5: The visualization of the sampled group center and the queried node points by RPS, FPS, and CAS. The blue and green balls indicate Ball Query. The red squares indicate Cube Query. The ball and cube have the same volume. (a) RPS covers 45.6% of the occupied space, while FPS covers 65% and CAS covers 75.2%.

As we can see the complexity of the algorithms in the table below, the methods in CAGQ are less complex while cover more space, more resilient with imbalance density and faster

Sample centers	RPS $O(N)$	FPS[11] $O(NlogN)$	RVS* $O(N)$	CAS* $O(N)$
Query nodes	Ball Query $O(MN)$	Cube Query* $O(MK)$	k-NN[8] $O(MN)$	CAGQ k-NN* $O(Mn_v)$

Table 1: Time complexity: We sample M centers from N points and query K neighbors per center. We limit the maximum number of points in each voxel to n_v. In practice, $K < N$, and n_v is usually of the same magnitude to K. Approximate FPS algorithm can be $O(NlogN)$[10]. * indicates our methods. See the supplementary for deduction details.

5.2. Grid Context Aggregation (GCA)

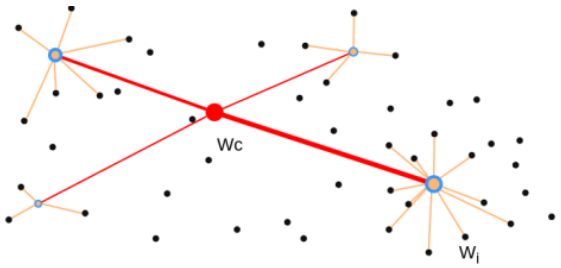


Figure 3: The red point is the group center. Yellow points are its node points. Black points are node points of the yellow points in the previous layer. The coverage weight is an important feature as it encodes the number of black points that have been aggregated to each yellow point.

After the CAGQ section, they build a local graph to aggregate information. There are 2 factors that make GCA a faster and better aggregation method: **Coverage Weight** and **Grid context pooling.** The GCA module can be described as equations 4 and 5

$$\widetilde{f}_{c,i} = e(\chi_i, f_i) * \mathcal{M}(f_i) \quad (4)$$

$$\widetilde{f}_c = \mathcal{A}(\{\widetilde{f}_{c,i}\}, i \in 1, ..., K) \quad (5)$$

“fc,i is the contribution from a node, and Xi is the xyz location of the node. M is a multi-layer perceptron (MLP), e is the edge attention function, and A is the aggregation function” Xu2021

Coverage weight: This technique based on a simple idea is that the node point that connects more neighbors should be assigned more weight. In addition, the weighting function is simply the total number of points that have been aggregated to a node in previous layers.

Grid context pooling: Remember that the semantic relation is important. In DGCNN, the authors improve that by associating the features in both Euclidean and feature spaces can improve the outcome. However, in GridGCN a group center is calculated as the barycenter of the node points (which is not always a physical point and don’t have features). GCA extracts context features fcxt by **pooling from all context points** for that virtual point. This technique have some benefits:

- Assign features to a virtual group point to calculate the semantic relation between center and its node points
- Even if the group center is picked on a physical point, that point still has more information

- Pooling is a light-weighted operation and need no learnable weights which requires little computational

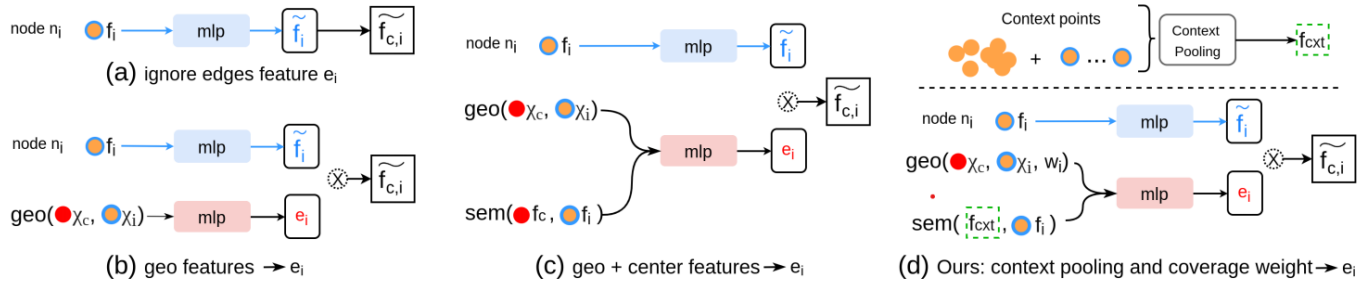


Figure 4: Different strategies to compute the contribution $\widetilde{f_{c,i}}$ from a node n_i to its center c. f_i, χ_i are the feature maps and the location of n_i. e_i is the edge feature between n_i and c calculated from the edge attention function. **(a)** Pointnet++ [33] ignores e_i. **(b)** computes e_i based on low dimensional geometric relation between n_i and c. **(c)** also consider semantic relation between the center and the node point, but c has to be sampled on one of the points from the previous layer. **(d)**. Grid-GCN's geo-relation also includes the coverage weight. It pools a context feature f_{cxt} from all stored neighbors to provide a semantic reference in e_i computing.

As can be seen from the figure above, the example for a) is pointNet, b) is PointNet++, c) is DGCNN

6. IDAM/IMP Iterative distance-aware similarity matrix convolution with Mutual-supervised Point elimination for efficient point cloud registration

I like the name of this paper. It include all of the special things of this methods:

- **Distance-aware** features in association with other feature gained form FPHF or GNN
- **Similarity matrix convolution** for matching
- Different parts of this methods **mutually supervised** each other
- **Two-stage point elimination** as an attention layer or outlier rejection

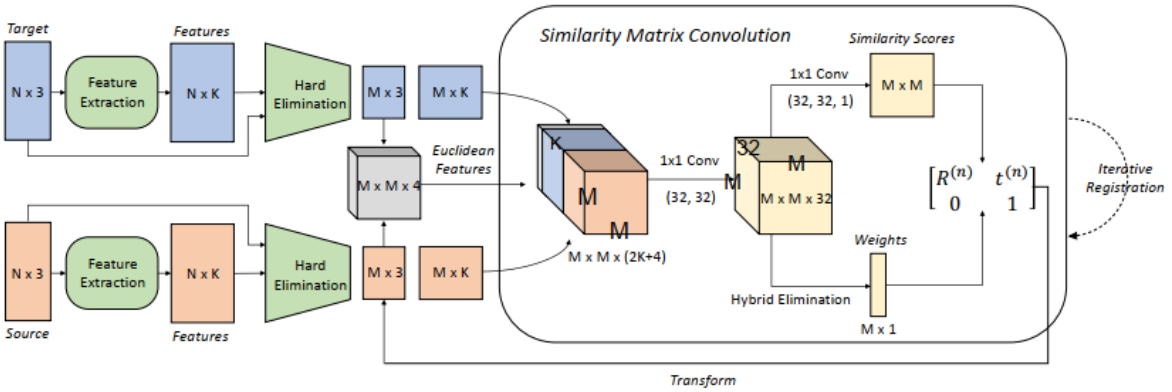


Fig. 1. The overall architecture of the IDAM registration pipeline. Details of hard point

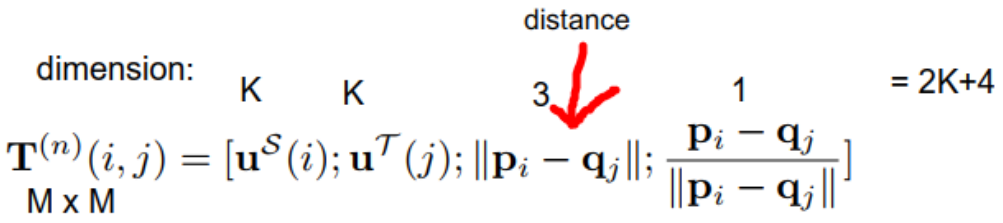
This method, for me, has a lot in common as well as the differences with the DCP method. I'll discuss about it later

IMP uses two ways to extract features at the beginning stage: Hand crafting features from FPFH or learning features from a simple GNN model. So I won't focus on this part

6.1. Similarity matrix convolution

As I previously mentioned in DL methods Matching, calculating the similarity by inner product of L2-distance has some problems. That's why they propose **distance-aware similarity matrix convolution.** Honestly, I misunderstood the word distance at first, because I thought it was the relative location of key points compared to their neighbors (which is quite similar in DGCNN, Grid-GCN…). However, the "distance" here is the distance of one source point to every point in the target point cloud. The idea here is quite the same as ICP, the correspondence is the distance to the closest point

First, they connect the feature for source, target and the distance to form (2K+4)-dimension distance-augmented feature tensor



Second they apply 1x1 convolution two times on T to extract similarity scores for each point pair (32 or 1 is the number of kernels), a softmax function is applied to turn the result into probabilities. The convolution learns their weight through **point matching loss.**

"IMP is presented to select the most similar point as a corresponding point. With IMP, a similarity matrix is formed using a dot product operation" Zhang2020. This is quite wrong because as we know, this is a learning process, not simply a dot product

6.2. Two-stage point elimination

There are two stage elimination:

- Hard elimination to remove points that are nor necessary
- Hybrid elimination to remove point pairs/correspondences that are not reliable

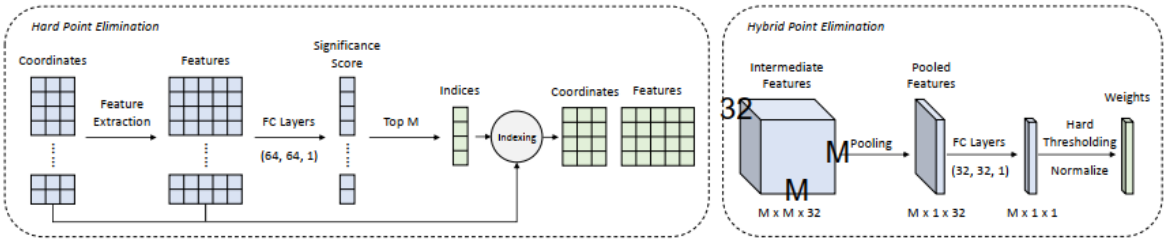


Fig. 2. Comparison of hard point elimination and hybrid point elimination. Hard point elimination filters points based on the features extracted independently for each point, while hybrid point elimination utilizes the joint information of the point pairs to compute weights for the orthogonal Procrustes algorithm.

Hard point elimination: The technique is to simply use an MLP layer to learn and assign a significant score for every point and then filter out M key points that have highest scores. M = N/6 according to the paper. This is to focus on important points like corners to speed the process and reduce the computational cost. However, this technique is quite simple, and we can intuitively realize that this method is not complex and fast enough like GridGCN

Hybrid point elimination: This technique is the mixture of hard elimination and gift elimination to remove point pairs that are reliable. At some point set registration models, the higher the similarity, the more reliable the correspondence is. On the other hand, for IMP, the model is set up to not 100% believe in the similarity matrix, they have to learn what correspondence is good even if the similarity value is high.
The weight for the point pair is defined as:

$$w_i = \frac{v(i) \cdot \mathbb{1}[\![v(i) \geq \text{median}_k(v(k))]\!]}{\sum_i v(i) \cdot \mathbb{1}[\![v(i) \geq \text{median}_k(v(k))]\!]}$$

Which means it gives w = 0 for the lowest validity score (hard elimination), and weighs the rest proportionally to the validity scores (soft elimination).
“a point in the source point cloud may be mistakenly eliminated in hard elimination” Li2020

6.3. Mutual-supervision loss

It’s quite common to calculate one loss at the final stage with y-hat and y. Here in IMP we have 3 different losses. Because the authors don’t have the labels of each point in the point cloud. So they suggest using the result of one part as a supervised signal for another part.

Point Matching Loss: It is a standard cross entropy loss used to supervise the similarity matrix convolution If the distance of points between source and target is larger than r, they can not be seen as correspondences, and no supervision signal is applied on them

Negative Entropy Loss: It uses the result of point matching loss to supervise hard point elimination. This mutual-supervision loss based on an idea that "if a point pi is a prominent point (high significance score), the probability distribution defined by the ith row of similarity matrix should have **low entropy** (which means the value doesn't change much, that's why they use negative entropy to calculate the degree of order) because it is confident in matching. On the other hand, the supervision on the similarity matrices has no direct relationship to hard point elimination (This is why I told you the model don't believe in the similarity matrix)" Li2020

Hybrid Elimination Loss: They use a probability that there exists a point in Target which is the correspondence of point pi in source as the supervision signal for (validity score)

6.4. DCP and IMP comparison

Similarity:

- They are correspondence-based methods with key point extraction, attention, matching, weighting layer, motion estimate with SVD
- Using deep-learning for point set registration
- Simple and easy to implement

Difference:

	DCP	**IDAM/IMP**
Feature extraction	DGCNN (complicated but efficient)	Using simple FPHF (hand-craft feature) or GNN (learning feature). However the performance of these techniques are different depended on the task
Attention	Using Transformer with information from both	Using two-stage point elimination (remove

	source and target point set (It keeps all the point, just focus on important points)	unnecessary point to reduce the computational cost)
Matching	Quite simple by using dot product to calculate similarity	Using Convolution and a learning-based process to continuingly update the similarity matrix
Loss	Simple loss with R,t and the ground truth	Mutual-supervision loss which is much complicated
Refinement	ICP (simple and quite efficient with easy task)	Start all over again the whole process which is better for unseen/noise tasks
Speed	Faster with small number of points, slower as that number increasing	Faster with large number of points
Correspondence idea	Assume that all the points in the source point cloud have correspondences in the target point cloud	Not all the points in source have correspondence point in target (that's why they have hybrid point elimination to weight correspondences)

7. 3DRegNet

Given a set of correspondences, 3DRegNet aims to use Deep neural network to solve these 2 challenges:

- Classify a point pair to decide whether it's a inlier/outlier
- Turn a registration task (transformation estimation in particular) into a Regression problem.

With regression, the authors proposed 2 ways to solve it: Deep neural network (DNN) or Procrustes (ICP and SVD)

Problem statement: The input are N correspondences. The output consists of NxMx3 variables. N: The inlier/outlier result for N pairs. M: rotation parameter. T=(t1,t2,t3)=(tx,ty,tz): translation vector.

I first thought we always have M=3 (3 rotation params) but it turns out different as the figure below

Representation	Rotation [deg] Mean	Rotation [deg] Median	Translation [m] Mean	Translation [m] Median	Time [s]	Classification Accuracy
m=3 Lie Algebra	**1.37**	**0.90**	**0.054**	**0.042**	0.0281	**0.96**
4 Quaternions	1.55	1.11	0.067	0.054	0.0284	0.95
9 Linear	5.78	4.78	0.059	0.042	0.0275	0.95
Procrustes	1.65	1.52	0.235	0.233	0.0243	0.52

Table 2: *Evaluation of different representations for the rotations.*

Beside the classification and regression blocks, the author proposed some more techniques to improve the outcome like refinement with the second 3DRegNet, 3DRegNet + ICP, and 3DRegNet + Umeyama refinement techniques, Data Augmentation

Refinement	Rotation [deg] Mean	Rotation [deg] Median	Translation [m] Mean	Translation [m] Median	Time [s]	Classification Accuracy
without	1.37	0.90	0.054	0.042	0.0281	0.96
with	1.19	0.89	0.053	0.044	0.0327	0.94

Table 4: *Evaluation of the use of 3DRegNet refinement.*

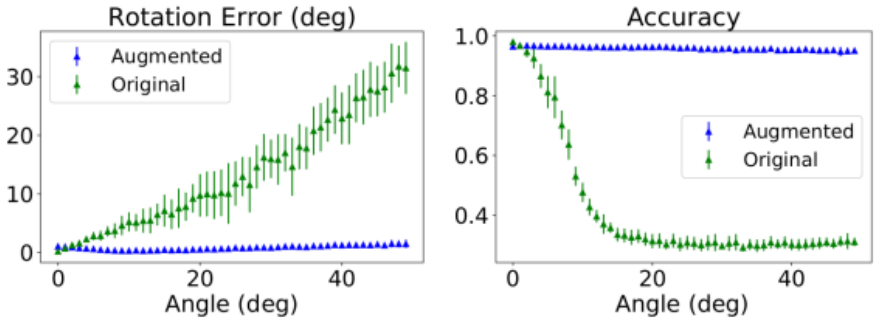
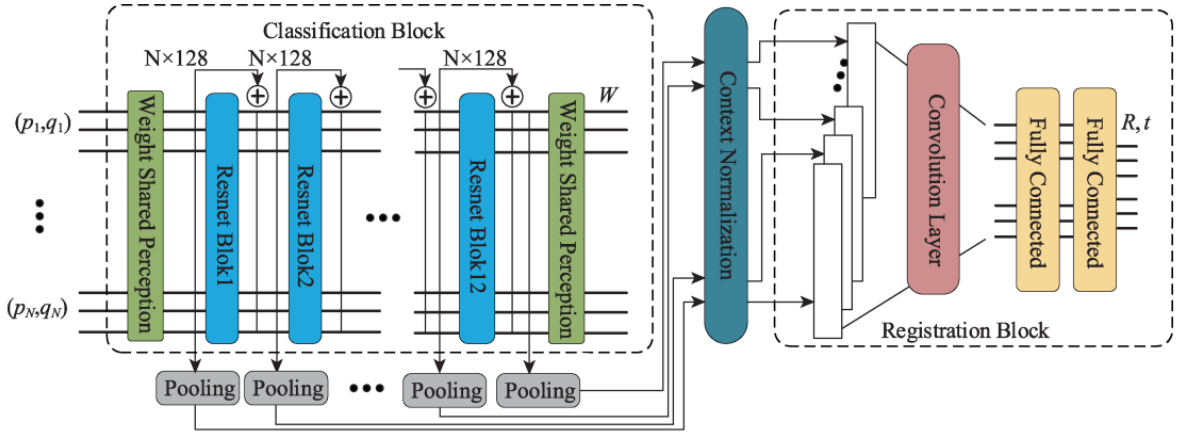


Figure 4: *Training with and without data augmentation. It is observed an improvement on the test results when perturbances are applied. The data augmentation regularizes the network for other rotations that were not included in the original dataset.*

7.1. Classification



Classification block and DNN regression block

The input is a 6-tuples set of 3D point correspondences. Each 3D point correspondence is processed by a fully connected layer with 128 ReLU activation functions. There is a weight sharing for each of the individual N point correspondences, and the output is of dimension N × 128, where they generate 128 dimensional features from every point correspondence.

The N × 128 output is then passed through C deep ResNets, with weight-shared fully connected layers instead of convolutional layers. At the end, they use another fully connected layer with ReLU followed by tanh units to produce the weights w=[0;1)

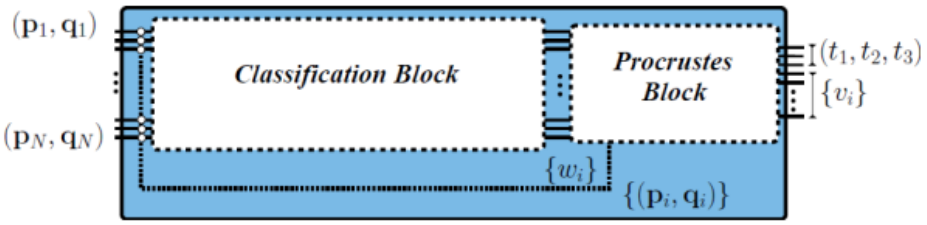
They compare w with a threshold T to decide inlier/outlier. T depends on how many point pairs they want to keep). The number C of deep ResNets depends on the complexity of the transformation (They used the second 3DRegNet for refinement “C = 8 for the first 3DRegNet and C = 4 for the refinement 3DRegNet”)

7.2. Regression with DNN

As the figure above, after each ResNet they process Nx128-dimension features into a Max Pooling Layer. The total C+1 feature vectors are then applied into a context normalization to have features with size of (C+1)x128 (this feature is independent of N). Finally, this features are then passed onto a convolution layer and fully connected layer to output m rotation params and translation vector

7.3. Regression with Procrustes

This is the alternation for the technique in Regression with DNN. First, we filter out the outliers and compute the centroid of the inliers, using this as the origin. Now, we just need to calculate the rotation from SVD. This is quite like the ICP+SVD that we knew ICP based on SVD. The result is worse than DNN. The runtime, transformation estimation, and even classification accuracy as in table 2



(b) *Representation of the 3DRegNet with Procrustes.*

8. Deep-3DAligner: Unsupervised learning-based

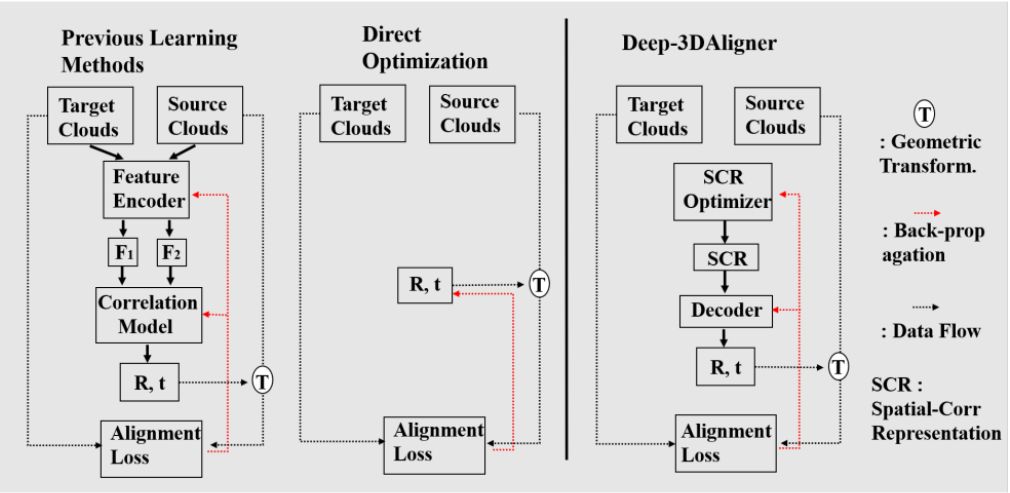


Fig. 1. Comparison of the pipeline between previous learning methods, direct optimization methods and our Deep-3DAligner for point set registration. Our method starts with optimizing a randomly initialized latent spatial correlation representation (SCR) feature, which is then decoded to the desired geometric transformation to align source and target point clouds, avoiding the explicit design of feature encoder and correlation module which is often challenging for point clouds input and increasing the model complexity by leveraging the structure of deep neural networks in comparison to direct optimization methods.

Previous learning based methods like DCP or IDAM usually focus on feature extractor and a correlation model (matching) to improve the point set registration task. However, these methods have some disadvantages:

- Large training dataset
- Not properly working with unseen shapes/categories, noise (which requires more data)

- It is hard to choose the right feature extractors (there are several methods like DGCNN with graphs, FCGF with convolution, PointNet…) as well as the correlation module (dot product or convolution in IDAM).
- “the main challenge concerns how to effectively model the “geometric relationship” between source and target objects” Wang2020

Deep-3DAligner is a novel **unsupervised learning-based** model (we can classify it as correspondence-free as Zhang2020 or an end-to-end learning based as Hoang2021) which contains three main parts:

- **SCR optimizer**: where Deep SCR features are **randomly initialized** and continuingly updated through Back propagation. We don’t need to worry about the feature extractor
- **Transformation decoder**: Decode the SCR features input into transformation matrix R,t. This transformation means the model turns the point set registration task into a **regression task**
- **Alignment Loss**: calculate the difference between source and target set with Chamfer distance loss. The model can use this loss as an supervised signal to update the SCR features

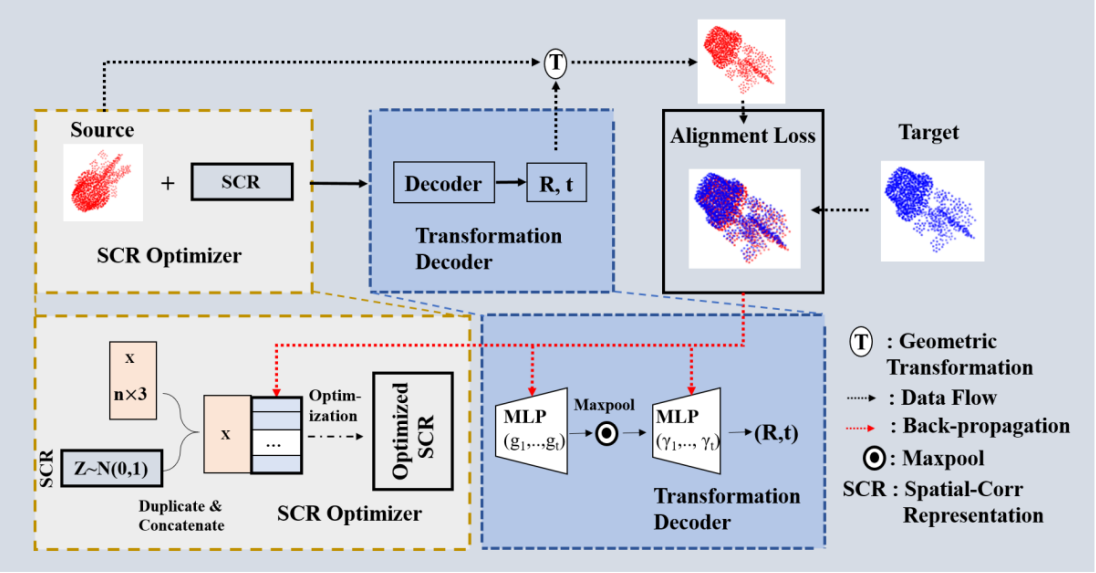


Fig. 2. Our pipeline. For a pair of input source and target point sets, our method starts with the SCR optimization process to generate a spatial-correlation representation feature, and a transformation regression process further decodes the SCR feature to the desired geometric transformation. The alignment loss is back-propagated to update the weight of the transformation decoder and the SCR feature during the training process. For testing, the weights of the transformation decoder remain constantly without updating.

Owing to the unsupervised learning based, Deep-3DAligner can achieve comparative performance compared to other learning methods like DCP. Deep-3DAligner resists well with noise, outlier, and unseen cases compared to DCP or Direct optimization models like ICP and its variants

Model	MSE(R)	MAE(R)	MSE(t)	MAE(t)
Ours	2.9240	0.8541	0.0002	0.012
DCP	12.040397	2.190995	0.000642	0.015552

TABLE 3
Quantitative result for 3D point set registration in presence of P.D. noise.

Model	MSE(R)	MAE(R)	MSE(t)	MAE(t)
Ours	7.3354	1.4702	0.0008	0.0222
DCP	34.624447	3.720148	0.002301	0.032245

TABLE 4
Quantitative result for 3D point set registration in presence of D.I. noise.

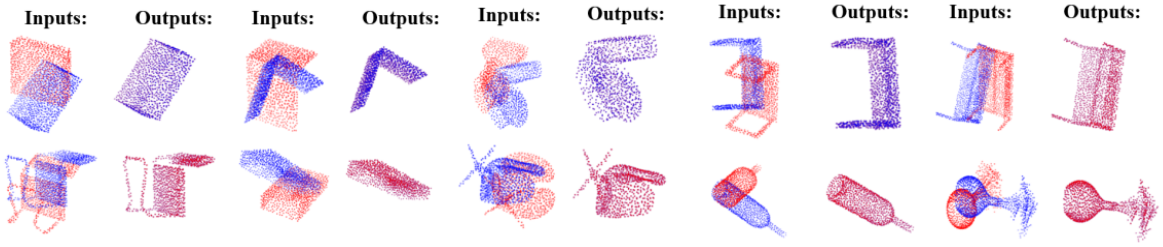
Model	MSE(R)	MAE(R)	MSE(t)	MAE(t)
Ours	16.5751	1.3631	0.0060	0.0255
DCP	46.992622	4.586546	0.002941	0.037136

TABLE 5
Quantitative result for 3D point set registration in presence of D.O. noise.

PD: Point Drift: randomly translate points

DI: Data incompleteness: randomly remove points

DO: Data Outlier: randomly add Outliers



Qualitative results

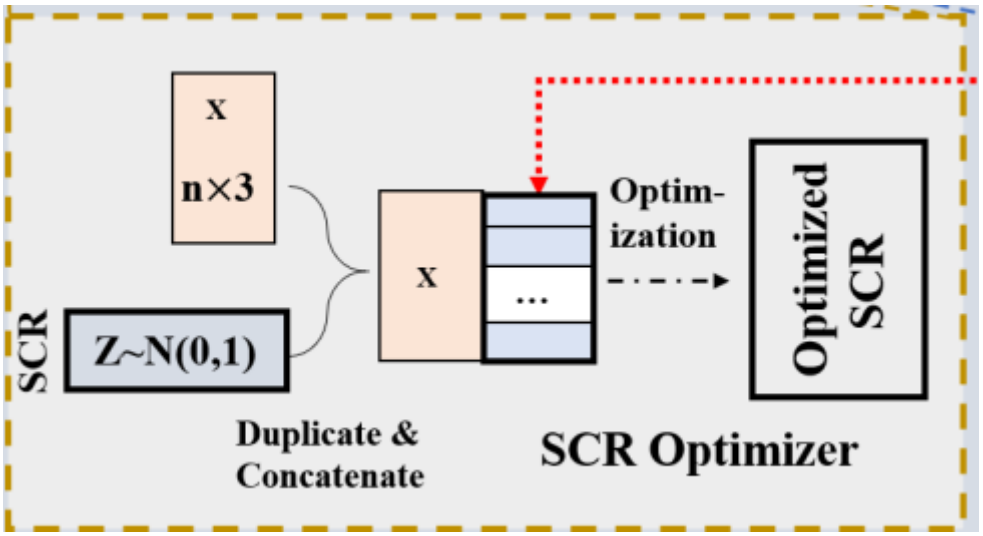
8.1. Spatial correlation representation (SCR)

In this paper, the authors define SCR as the **latent feature** that characterizes the essence of **spatial correlation** from **a given pair** of source and target point sets

As can be seen from the figure below, SCR features z that is first randomly initialized by a Gaussian distribution N(0,1) with a dimension of 2048 for each pair of source and target point. It is then stacked with the coordinates of each point x in source. This SCR features would be continually updated throughout the alignment loss back-propagation process

This reminds me of GAN (generative adversarial network) which includes 2 models: the generator model that we train to generate new examples, and the discriminator model that

tries to classify examples as either real (from the domain) or fake (generated). The common thing is the generator and SCR both start by producing a matrix or vector with a Gaussian distribution and then keep learning the pattern of the task over time in order to produce the real matrix or vector. I'm thinking of developing this model by using GAN instead of SCR

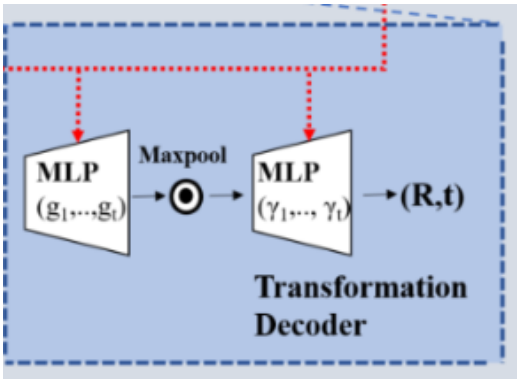


The implicit design of SCR allows Deep-3DAligner more flexible in the spatial correlation learning that is more adaptive for alignment of unseen cases

8.2. Transformation decoder

In this part, the input is the concatenation of point location x and SCR vector z to form the [x,z] vector. The input is passed onto the first MLP layers. Then, they use a max pool layer to exact the global feature L. They further decode the global feature L to the transformation parameters by a second network, which includes t successive MLP layers with a ReLU activation function

Something confused here is that the author states in the 3.3 (Transformation decoder) in the paper that in two MLP networks. The number of output channels is more than that for the inputs. Obviously we can see it in the figure below, they deliberately draw the MLP to be bigger towards the outputs. However, when I read the 4.2 (experiment settings) in the paper. They say that "For the decoding network, the first part includes 2 MLP layers with dimensions (256,128) and a max pool layer. Then, we use 3 additional MLPs with dimensions of (128, 64, 3) for decoding the rotation matrix and with dimensions of (128, 64, 3) for decoding the translation matrix." means that the output dimension must be smaller compared to the input, at least for the second MLP because we truly need 6-dimension vector to output R,t.



After having R and t, we can transform the source point set from S to S'. Then using Alignment Loss to update the weights for the whole process

8.3. Loss Function and Optimization Strategy

Since this method has no ground truth transformation as well as assume no direct correspondences between the 2 point sets. The most suitable and efficient loss they can think of is Chamfer Distance loss which finds the nearest point in the other point set for each point in each cloud, and sums the square of distance up. You can read more about Chamfer Distance here

$$L_{\text{Chamfer}}(T_\phi(\mathbf{S}), \mathbf{G}) = \sum_{x \in T_\phi(\mathbf{S})} \min_{y \in \mathbf{G}} ||x - y||_2^2 + \sum_{y \in \mathbf{G}} \min_{x \in T_\phi(\mathbf{S})} ||x - y||_2^2$$

However, the authors doubt that because Chamfer distance-based loss is less sensitive to the translation, it possibly contributes to the deficit of translation prediction performance.

Model	MSE(R)	RMSE(R)	MAE(R)	MSE(t)	RMSE(t)	MAE(t)
ICP [14]	892.601135	29.876431	23.626110	0.086005	0.293266	0.251916
Go-ICP [21]	192.258636	13.865736	2.914169	0.000491	0.022154	0.006219
FGR [22]	97.002747	9.848997	1.445460	0.000182	0.013503	0.002231
PointNetLK [38]	306.323975	17.502113	5.280545	0.000784	0.028007	0.007203
DCPv1+SVD (Supervised) [15]	19.201385	4.381938	2.680408	0.000025	0.004950	0.003597
DCPv2+SVD (Supervised) [15]	9.923701	3.150191	2.007210	**0.000025**	**0.005039**	**0.003703**
Deep-3DAligner (MLP-based, Unsupervised)	**3.715267**	**1.485832**	**1.040233**	0.000822	0.026767	0.022763

TABLE 2
ModelNet40: Test on unseen categories. Our model is trained in an unsupervised manner without ground-truth labels. Our model does not require SVD-based fine-tuning processes.

The optimization is splitted into 2 phases:

For training dataset D: these latent vectors z are optimized along with the weights of network decoder using a stochastic gradient descent-based algorithm

$$\underset{\text{weights}}{\theta^{\mathbf{optimal}}}, \underset{\text{SCR}}{\mathbf{z}^{\mathbf{optimal}}} = \underset{\theta,\mathbf{z}}{\operatorname{argmin}}[\mathbb{E}_{(\mathbf{S_i},\mathbf{G_j})\sim\mathbf{D}}[\mathcal{L}(\mathbf{S_i},\mathbf{G_j},g_\theta(\mathbf{S_i},\mathbf{z}))]]. \quad (6)$$

For the test dataset W: They keep the weight and only optimize the SCR feature

$$\mathbf{z}^{\mathbf{optimal}} = \underset{\mathbf{z}}{\operatorname{argmin}}[\mathbb{E}_{(\mathbf{S_i},\mathbf{G_j})\sim\mathbf{W}}[\mathcal{L}(\mathbf{S_i},\mathbf{G_j},g_{\tilde{\theta}}(\mathbf{S_i},\mathbf{z}))]]. \quad (7)$$

From this point on, I consider the issue of this method's speed. In this article or online, I can't find any speed tests. I think that processing the z optimization in testing will take a long time.